ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

An Implementation of MJPEG Video Codec Using STM32 and FPGA

SENIOR DESIGN PROJECT

Ahmet Bedri YORULMAZ
Burak ŞİMŞEK
Mehmet Hakan Yüksekkaya

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

AUGUST , 2021

ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

An Implementation of MJPEG Video Codec Using STM32 and FPGA

SENIOR DESIGN PROJECT

Ahmet Bedri YORULMAZ
(040160050)

Burak ŞİMŞEK
(040160206)

Mehmet Hakan YÜKSEKKAYA
(040160099)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

Project Advisor: Dr. Öğr. Üyesi Tankut AKGÜL

AUGUST , 2021

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

An Implementation of MJPEG Video Codec Using STM32 and FPGA

LİSANS BİTİRME TASARIM PROJESİ

Ahmet Bedri YORULMAZ
(040160050)

Burak ŞİMŞEK
(040160206)

Mehmet Hakan YÜKSEKKAYA
(040160099)

Proje Danışmanı: Dr. Öğr. Üyesi Tankut AKGÜL

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

AĞUSTOS, 2021

We are submitting the Senior Design Project Report entitled as "An Implementation of MJPEG Video Codec Using STM32 and FPGA". The Senior Design Project Report has been prepared as to fulfill the relevant regulations of the Electronics and Communication Engineering Department of Istanbul Technical University. We hereby confirm that we have realized all stages of the Senior Design Project work by ourselves and we have abided by the ethical rules with respect to academic and professional integrity

Ahmet Bedri YORUlMAZ .....
(040160050)

Burak ŞİMŞEK .....
(040160206)

Mehmet Hakan YÜKSEKKAYA .....
(040160099)

FOREWORD

We would like to thank our esteemed teacher Dr. Öğr Tankut AKGÜL, who supported us throughout our graduation project, whose knowledge and experience we benefited from, who never lost their enthusiasm, desire and determination to produce new things, and who we always took as an example in this regard.
In addition, we would like to thank our family for their support in all matters, our department friends who did not spare their knowledge and help, and with whom we shared very good news for 4 years.

Aug 2021

Ahmet Bedri YORULMAZ
Burak ŞİMŞEK
Mehmet Hakan YÜKSEKKAYA

TABLE OF CONTENTS

Page

ABBREVIATIONS

JPEG : Joint Photographic Experts Group
DCT : Discrete Cosine Transform
IDCT : Inverse Discrete Cosine Transform
I^2C : Inter-Integrated Circuit
SPI : Serial Peripheral Interface
RGB : Red Green Blue
SCCB : Serial Camera Control Bus
MISO : Master In Slave Out
MOSI : Master Out Slave In
CS : Chip Select
SCK : Serial Clock
FIFO : First In First Out
FSM : Finite State Machine
ISO : International Organization for Standardization
IEEE : Institute of Electrical and Electronics Engineer
LUT : Look up table
FPS : Frame per second
HSync : Horizontal Sync
VSync : Vertical Sync

LIST OF TABLES

Page

LIST OF FIGURES

Page

PROJECT TITLE IN ENGLISH HERE

SUMMARY

In this project, it is aimed to compress the image taken from the camera and print it on the screen by using FPGA and STM32 cards. Compression was performed using the MJPEG format. Codes were written in C programming language using STM32 board in STM32CubeIDE environment and HAL library, and FPGA in Xilinx environment using Verilog language. The project basically consists of three parts. In the first part, the YUV image data taken from the camera configured with the FPGA will be encoded and compressed in jpeg format by coming to the FPGA. While encoding the image from the camera, Discrete Cosine Transform (DCT), Quantization and Zig-Zag methods will be used. Then it will be transmitted to STM32 via SPI communication protocol and the image will be decoded there. Decode stages will consist of Dequantization, Zig-Zag and Inverse Discrete Cosine Transform (IDCT). Finally, after the decoding process is finished, the image will be drawn on the LCD screen configured by STM32 by converting from YUV image format to RGB format. This cycle will be performed for each frame and the video will be created.In the second part of the project, only FPGA will be used. The YUV image data taken from the camera configured with the FPGA will have created the compressed image data after the encoding and decoding processes in the FPGA. All of the DCT, Zig-Zag, Quantization , Inverse Zig-Zag, Dequantization and IDCT stages in encode and decode will be implemented on the FPGA. The YUV image format obtained after these stages will be converted to RGB format, and it will be converted from RGB to DVI format thanks to the hardware designed on the FPGA. As a result, the data obtained will be projected to the screen as a video with the HDMI protocol.In the third part, the effect of the hardware on the compression process will be measured by comparing the Frame Per Second (FPS) and resolution obtained from the decoding process using both FPGA and STM32 and the encoding process using only FPGA.

TÜRKÇE PROJE BAŞLIĞI BURAYA YAZILIR

ÖZET

Bu projede FPGA ve STM32 kartları kullanılarak kameradan alınan görüntünün sıkıştırılması ve ekrana yazdırılması hedeflenmiştir. Sıkıştırma işlemi MJPEG formatı kullanılarak gerçekleştirilmiştir. STM32 kartı STM32CubeIDE ortamında ve HAL kütüphanesi kullanılarak C programlama dilinde , FPGA ise Xilinx ortamında Verilog dili kullanılarak kodlar yazılmıştır.

Proje temelde üç bölümden oluşmaktadır. İlk bölümde FPGA ile konfigüre edilmiş olan kameradan alınan YUV görüntü verisi FPGA' ye gelerek encoding edilecek ve jpeg formatında sıkıştırılacaktır. Kameradan gelen görüntü encode edilirken, DCT, Quantization ve Zig-Zag metodları kullanılacaktır. Ardından SPI haberleşme protokolü ile STM32'ye ulaştırılacak ve burada görüntü decode edilecektir. Decode aşamaları ise Dequantization, Zig-Zag ve IDCT'den oluşacaktır. En son, decode işlemi bittikten sonra, YUV görüntü formatından RGB formatına çevirilerek, STM32 tarafından konfigüre edilen LCD ekrana görüntü çizdirilecektir. Bu döngü her frame için gerçekleştirilecek olup video oluşturulucaktır.

Projenin ikinci kısmında sadece FPGA kullanılacaktır. FPGA ile konfigüre edilmiş kameradan alınan YUV görüntü verileri FPGA'de yapılacak encoding ve decoding işlemlerinden sonra sıkıştırılmış olan görüntü verilerini oluşturmuş olacaktır. Encode ve decode'da yapılan DCT, Zig-Zag , Quantization ,Ters Zig-Zag , Dequantization ve IDCT aşamalarının tümü FPGA üzerinde gerçeklenmiş olacaktır. Bu aşamalardan sonra elde edilen YUV görüntü formatı RGB formatına dönüştürülerek, yine FPGA üzerinde tasarlanmış olan donanım sayesinde RGB'den DVI formatına dönüştürülecektir. Sonuç olarak elde edilen veri HDMI protokolü ile ekrana video olarak yansıtılacaktır.

Üçüncü bölümde ise hem FPGA hem de STM32 kullanılarak yapılan codec işlemi ile sadece FPGA kullanılarak yapılan codec işleminden alınan FPS ve çözünürlük karşılaştırılarak donanımın kod çözme işlemine olan etkisi ölçülmüş olacaktır.

1. INTRODUCTION

1.1 Purpose of project

In this project, the encoding process is implemented only in FPGA. It is aimed to compare the two hardware by performing the decode process in both FPGA and STM32.

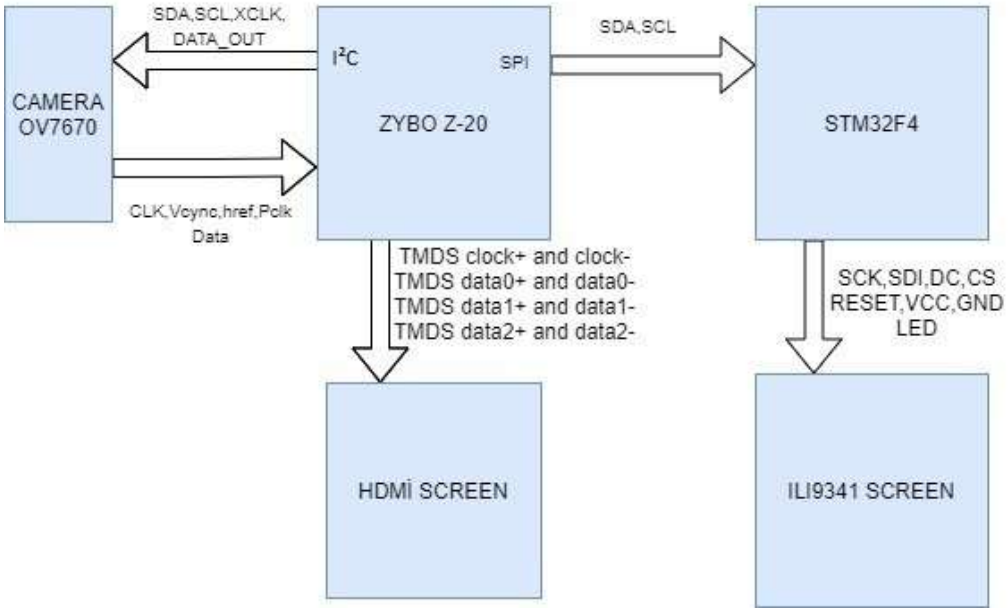


Fig 1.1 : Diagram of the project

1.2 Suggested work plan and possible changes in the project proposal

At the beginning of the project, it was planned to be designed on FPGA only to speed up Discrete Cosine Transform (DCT) and Inverse Discrete Cosine Transform (IDCT) operations. However, since there will be a loss of time while sending the data over FPGA, the encoding part was carried out on the FPGA by connecting the camera to the FPGA.

It was planned to use 320x240 resolution, but 160x120 resolution was used because operations on STM32 are slow.

1.3 Technologies planned to be used

Zybo Z7-20 Board , STM32F407vg Discovery Kit, OV7670 camera module, HDMI monitor and LCD TFT screen with ILI9341 driver are used in the system.

1.3.1 Zybo Z7-20 Board

It has a 667 MHz, dual core and powerful processor like Cortex-A9. Along with having 630 KB block RAM, it has 8 communication interfaces as 2xSPI, 2x I^2C , 2xCAN, 2x UART . With its extremely powerful hardware, it easily performed DCT/IDCT processes that would take a long time in video compression.

.

Figure 1.2 : Zybo Z7-20 Board

1.3.1.1 I^2C Communication interface

Inter Integrated Circuit (I^2C) works with low bandwidth and this communication protocol has two communication channels, Serial Clock (SCL) and Serial Data (SDA). Data is transmitted over SDA, data synchronization is provided between the receiver and the transmitter with SCL. SDA and SCL transmission lines can have multiple devices. The party that initiates the communication is called the master and the other party is called the slave. The master determines when the data will be sent, read and terminated. Communication between master and slave is bidirectional.

1.3.2 STM32F407vg Discovery Kit

This development board was chosen due to its powerful hardware with 32-bit FPU core and ARM Cortex® -M4 processor. In addition, having a large library provided great comfort while writing code. While it has an external crystal oscillator that can give 168 MHz; It has 15 communication interfaces (I^2C 3 pcs, USART/UART 4 pcs, SPIR 3 pcs, CAN 2 pcs). And it has 140 I/O ports with interrupt capability in total.

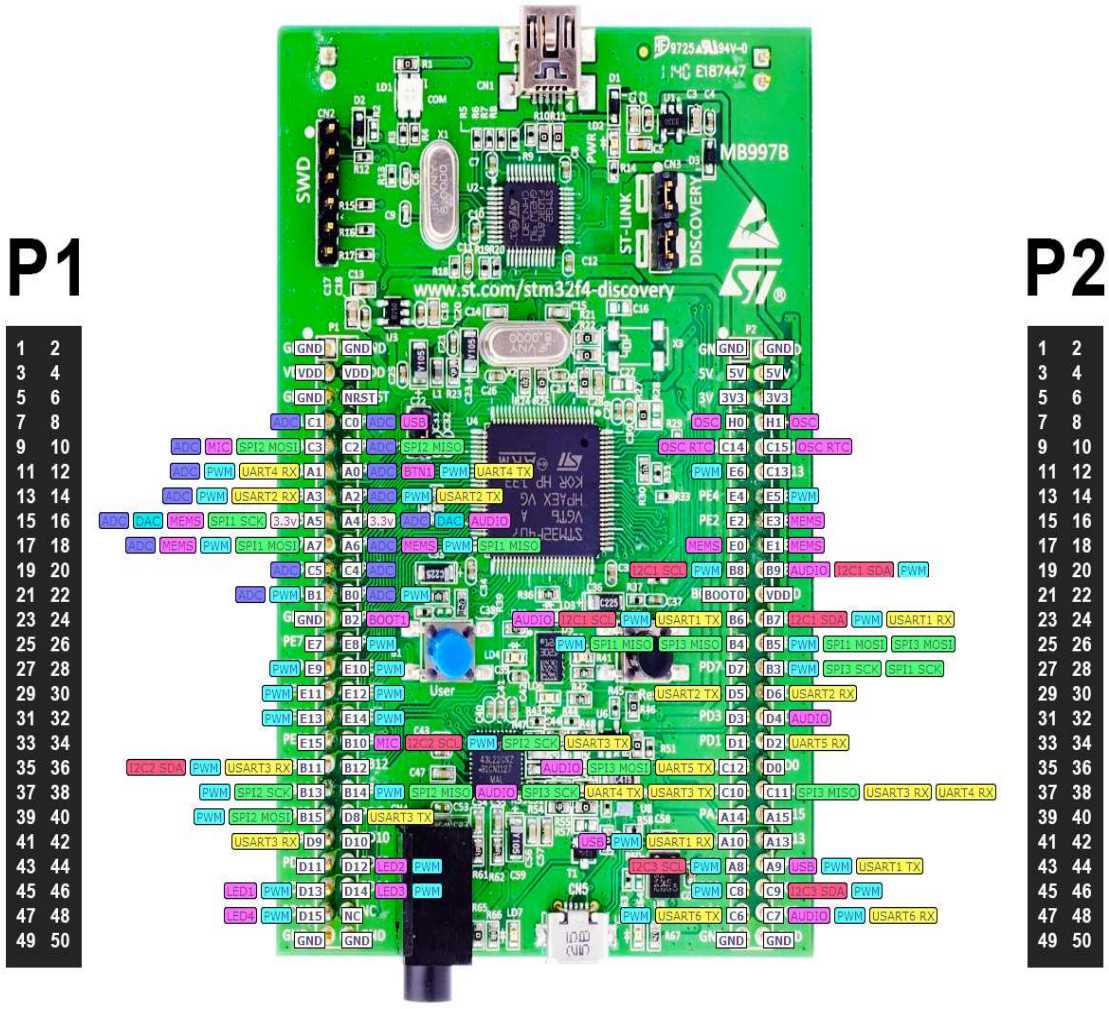


Figure 1.3 : STM32F407vg Discovery

1.3.2.1 SPI

Similar to I^2C , Serial Peripheral Interface (SPI) also has a master-slave relationship. It has 4 lines: Chip Select (CS), SCK, Master Out Slave In (MOSI) and Master In Slave Out (MISO). With the CS (SS) pin, the device to which data transfer will be made is determined. SCK provides synchronization with the clock signals produced by the master. With MOSI, the data from the master to the slave is transferred, also with MISO, the data from the slave is sent to the master. Unlike I^2C , the data flow is unidirectional and there is faster data flow in SPI. The SPI protocol is used to communicate with the FPGA and STM32 board, as well as to communicate with the LCD screen.

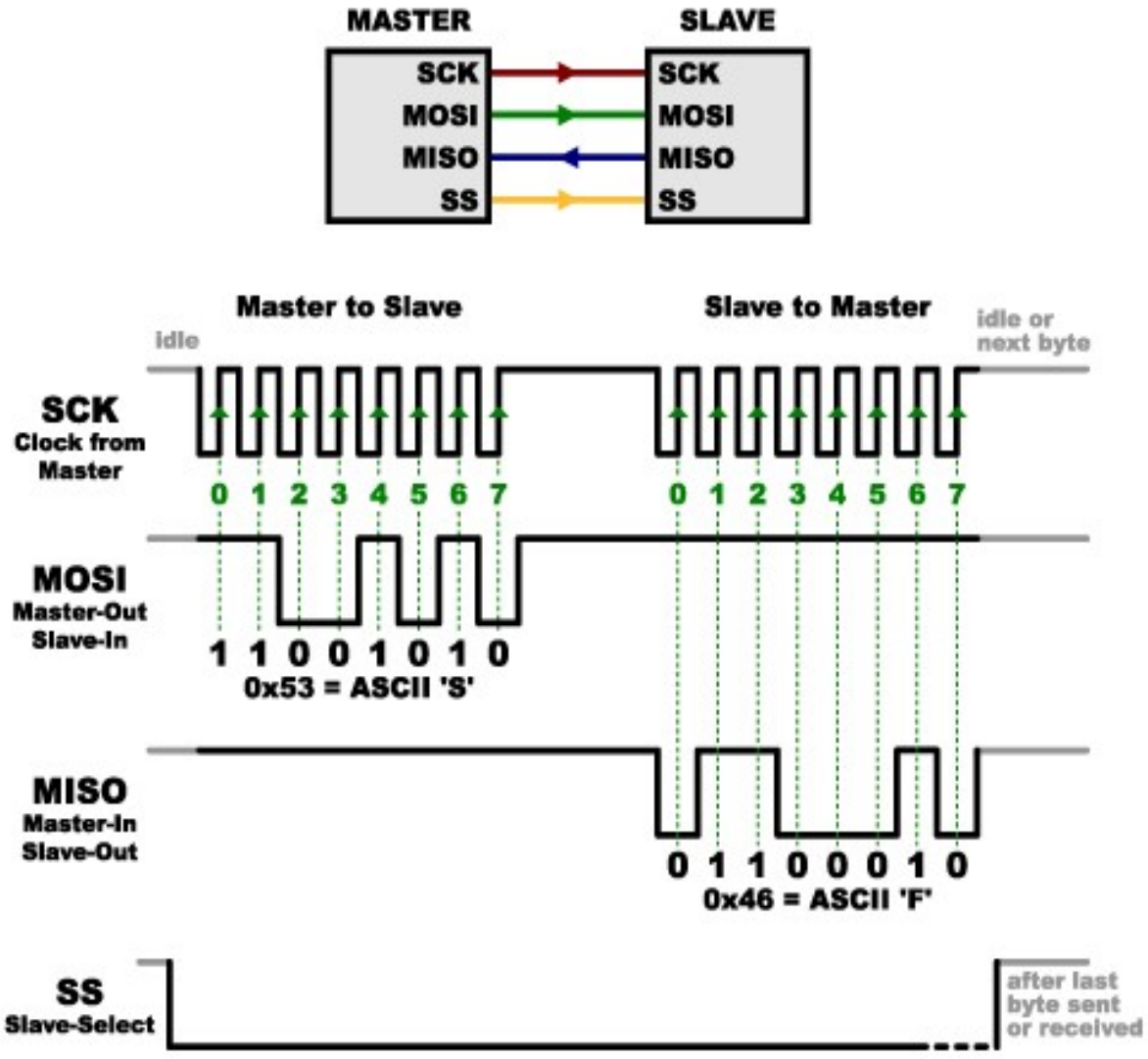


Figure 1.4 : SPI protocol

1.3.3 OV7670 camera module

The FIFO version of the OV7670 camera module was used in the project. This module, which offers 640x480 resolution with its 0.3 megapixel camera, gives an video quality of 30 FPS. The camera is used by connecting it to FPGA card.

Figure 1.5 : OV7670 Camera module

1.3.4 LCD TFT screen

The LCD TFT screen with ILI9341 driver chip has a resolution of 320x240, has a size of 2.4 and communicates with SPI. SD card slot is also possible on it, but not used in this project.

Figure 1.6: LCD TFT screen

1.4 Literature Review

In the literature search, the Joint Photographic Experts Group (JPEG) structure was examined.

1.4.1 JPEG standard

JPEG or JPG is a lossy compression method for digital images that is widely used. Although it is an old technology, it is used both in the internet environment and in cameras or phones. Although JPEG is lossy compression, it produces a significant reduction in file size. Therefore, compression technology must be used, especially in straming media technologies. It is tried to obtain the closest image to the original by trying to minimize the loss experienced with the compression process. JPEG uses the DCT algorithm when compressing. After the DCT process, Quantization and Zig-Zag scanning are also performed and the file is compressed. In order to decode the image, Inverse Zig-Zag, Dequantization and IDCT processes are performed and a lossy image is obtained. [1]

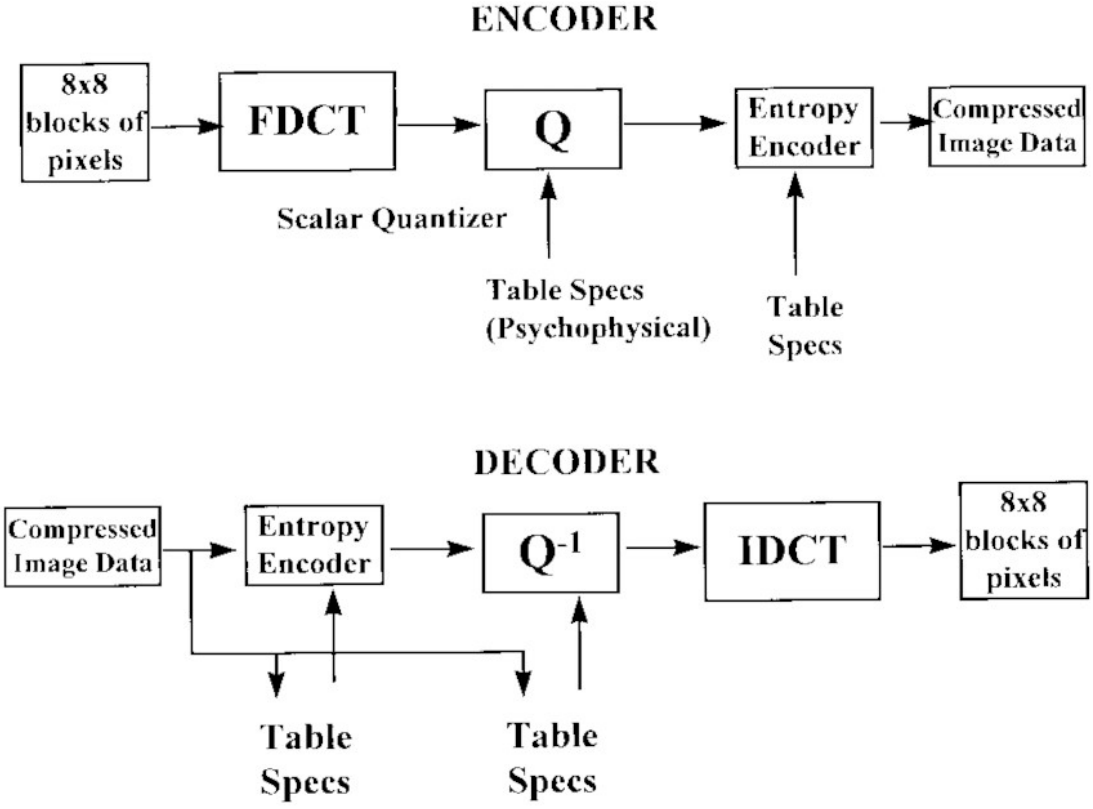


Figure 1.7: JPEG process

1.4.1.1 Color Transform

The real-time image is stored in the RGB color space, but a high bandwidth is required to transmit the image. Therefore, the image must first be converted from RGB color space to YCbCr color space.

RGB color space consists of 24 bits. 8 bits are used for each color palette. It is expressed as red (255,0,0), green (0,255,0), and blue (0,0,255).

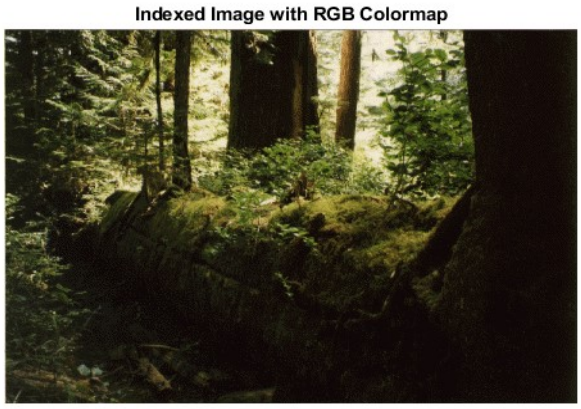


Figure 1.8: Image in RGB format

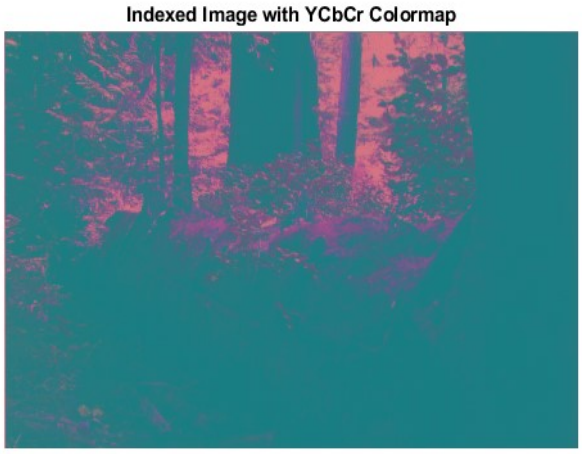


Figure 1.9: Image in YCbCr format

The YCbCr color space was introduced as a result of digitization. Y is the luma component, that is, it represents the brightness of colors. The most important ingredient is Luma, as the human eye is most sensitive to brightness. The other two elements are the chroma component. Cb represents blue and Cr represents red. Image can convert from RGB to YCbCr using the following equations. [2]

$$Y = 16 + \frac{65.738R}{256} + \frac{129.057G}{256} + \frac{25.064B}{256}$$

$$Cb = 128 - \frac{37.945R}{256} - \frac{74.494G}{256} + \frac{112.439B}{256}$$

$$Cr = 128 + \frac{112.439R}{256} - \frac{94.154G}{256} - \frac{18.285B}{256}$$

Figure 1.10: RGB to YCbCr convertion equation

1.4.1.2 Discrete Cosine Transform (DCT)

DCT was first designed by Nasir Ahmed in 1972. However, in 1974 they published DCT II, which they developed further. We use DCT II directly as the DCT used today. In the DCT process, the incoming image data is converted to the frequency region and these values carry the information of the image. Since the matrix that will come from the image is two-dimensional, the DCT block is also two-dimensional. Before starting the DCT process, the image must be split into 8x8 blocks. Each block is processed from left to right and top to bottom.[3]

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7} f(x,y)\cos[\frac{\Pi(2x+1)u}{16}]\cos[\frac{\Pi(2y+1)v}{16}]$$
$$u = 0...7, v = 0...7$$

Figure 1.11 : DCT formula

1-D DCT is used to accelerate DCT on hardware. Two 1-D DCTs are linked by a transpose buffer to create a 2-D DCT.

There are various methods for calculating 1-D DCT. The first method adopts the work of Agostini that implemented Arai scaled 1-D DCT algorithm [4]. In equtation (1.1) ,Variable x is 8 point vector. C is Arai's DCT matrix and y' is vector of scaled DCT coefficients. To get the real DCT coefficients, y' must be element by element multiplied with post-scaling factor.In equation (1.2), constant s is vector of post-scaling factor.

$$\mathbf{y' = C\,x} \tag{1.1}$$

$$\mathbf{y = s\,.*\,y'} \tag{1.2}$$

Step 1:	$a_0 = x_0 + x_7$ $a_1 = x_1 + x_6$ $a_2 = x_3 - x_4$ $a_3 = x_1 - x_6$ $a_4 = x_2 + x_5$ $a_5 = x_3 + x_4$ $a_6 = x_2 - x_5$ $a_7 = x_0 - x_7$
Step 2:	$b_0 = a_0 + a_5$ $b_1 = a_1 - a_4$ $b_2 = a_2 + a_6$ $b_3 = a_1 + a_4$ $b_4 = a_0 - a_5$ $b_5 = a_3 + a_7$ $b_6 = a_3 + a_6$ $b_7 = a_7$
Step 3:	$d_0 = b_0 + b_3;$ $d_1 = b_0 - b_3;$ $d_2 = b_2;$ $d_3 = b_1 + b_4;$ $d_4 = b_2 - b_5;$ $d_5 = b_4;$ $d_6 = b_5;$ $d_7 = b_6;$ $d_8 = b_7;$
Step 4:	$e_0 = d_0;$ $e_1 = d_1;$ $e_2 = m_3 * d_2;$ $e_3 = m_1 * d_7;$ $e_4 = m_4 * d_6;$ $e_5 = d_5;$ $e_6 = m_1 * d_3;$ $e_7 = m_2 * d_4;$ $e_8 = d_8;$

Step 5:	f0 = e0;
	f1 = e1;
	f2 = e5 + e6;
	f3 = e5 - e6;
	f4 = e3 + e8;
	f5 = e8 - e3;
	f6 = e2 + e7;
	f7 = e4 + e7;

Step 6:	$y'_0 = f_0;$ $y'_1 = f_4 + f_7;$ $y'_2 = f_2;$ $y'_3 = f_5 - f_6;$ $y'_4 = f_1;$ $y'_5 = f_5 + f_6;$ $y'_6 = f_3;$ $y'_7 = f_4 - f_7;$

Figure 1.12: Agostini Algorithm

The second algorithm used in this project is obtained by multiplying the DCT coefficients using adder and subtractor. In 1-D DCT operation, the DCT coefficient matrix in the figure is used. [5]

$$T=\frac{1}{2}\begin{pmatrix} A & A & A & A & A & A & A & A \\ D & E & F & G & -G & -F & -E & -D \\ B & C & -C & -B & -B & -C & C & B \\ E & -G & -D & -F & F & D & G & -E \\ A & -A & -A & A & A & -A & -A & A \\ F & -D & G & E & -E & -G & D & -F \\ C & -B & B & -C & -C & B & -B & C \\ G & -F & E & -D & D & -E & F & -G \end{pmatrix}$$

Where $A=\cos(\frac{\pi}{4})$, $B=\cos(\frac{\pi}{8})$, $C=\sin(\frac{\pi}{8})$, $D=\cos(\frac{\pi}{16})$, $E=\cos(\frac{3\pi}{16})$, $F=\sin(\frac{3\pi}{16})$, $G=\sin(\frac{\pi}{16})$.

Figure 1.13: DCT coefficients

The input and output matrix of the 1-D DCT process is shown in the figure. This algorithm was used in the 1-D DCT process used in this project.

$$\begin{pmatrix} Y(0) \\ Y(2) \\ Y(4) \\ Y(6) \\ Y(1) \\ Y(3) \\ Y(5) \\ Y(7) \end{pmatrix} = \frac{1}{2}\begin{pmatrix} A & A & A & A & 0 & 0 & 0 & 0 \\ B & C & -C & -B & 0 & 0 & 0 & 0 \\ A & -A & -A & A & 0 & 0 & 0 & 0 \\ C & -B & B & -C & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D & E & F & G \\ 0 & 0 & 0 & 0 & E & -G & -D & -F \\ 0 & 0 & 0 & 0 & F & -D & G & E \\ 0 & 0 & 0 & 0 & G & -F & E & -D \end{pmatrix}\begin{pmatrix} X(0)+X(7) \\ X(1)+X(6) \\ X(2)+X(5) \\ X(3)+X(4) \\ X(0)-X(7) \\ X(1)-X(6) \\ X(2)-X(5) \\ X(3)-X(4) \end{pmatrix}$$

Figure 1.14: DCT matrix

1.4.1.3 Quantization

Quantization is performed after DCT. Here, the high-frequency elements in the image matrix in the frequency domain are set to zero. Because the human eye is more sensitive to low frequencies. Therefore, when the high frequencies are eliminated, the image is not distorted much and the image is compressed. As a result, the number of elements to be processed in the next block will be reduced. So here is the most important part of the compression process. All 64-element matrix obtained after DCT processing needs to be quantized. Although the Quantization matrix is standardized, the elements in the Quantization matrix can be changed between 1 and 255. As seen in Figure 1.15, Quantization matrix values corresponding to higher frequency elements are bigger. As a result of the operation shown below, the high-frequency elements are reset [6].

$$F^{Q}(u,v) = Integer\ Round\left(\frac{F(u,v)}{Q(u,v)}\right)$$

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 1.15: Quantization formula and Quantization matrix [6]

1.4.1.4 Zig-Zag scanning

In the next step, Zig-Zag scanning is performed. With this block, the 8x8 matrix is converted into an array. The matrix elements are ordered from low frequency to high frequency. In order to achieve this, Zig-Zag scanning is performed as in the Figure 1.16 [7].

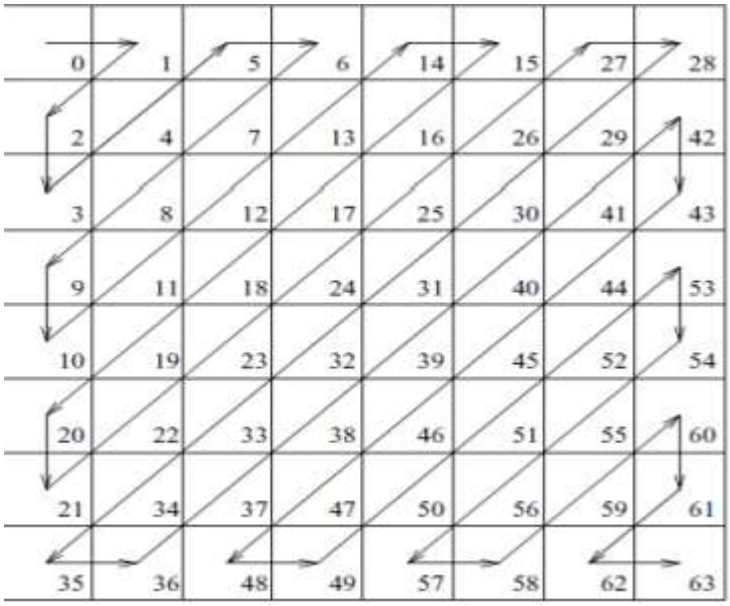


Figure 1.16: Zig-Zag scanning [7]

1.4.1.5 Dequantization

It is the first operation done when decoding the compressed image. Here, exactly the reverse of the Quantization process is done.

$$F^{Q'}(u,v) = F^{Q}(u,v) * Q(u,v)$$

Figure 1.17: Dequantization formula [8]

1.4.1.6 Inverse Discrete Cosinus Transform (IDCT)

IDCT is referred to as type III DCT and is the last step before the image is decoded. Although it is another time-consuming process with DCT, here the image is tried to be reconstructed to be closest to the original.

The IDCT process of the DCT algorithm used in the project was performed by the matrix multiplication of the coefficients in Figure 1.18 [3].

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} a & c & a & f \\ a & f & -a & -c \\ a & -f & -a & c \\ a & -c & a & -f \end{bmatrix} \begin{bmatrix} X(0) \\ X(2) \\ X(4) \\ X(6) \end{bmatrix} + \begin{bmatrix} b & d & e & g \\ d & -g & -b & -e \\ e & -b & g & d \\ g & -e & d & -b \end{bmatrix} \begin{bmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{bmatrix}$$

$$\begin{bmatrix} Y(7) \\ Y(6) \\ Y(5) \\ Y(4) \end{bmatrix} = \begin{bmatrix} a & c & a & f \\ a & f & -a & -c \\ a & -f & -a & c \\ a & -c & a & -f \end{bmatrix} \begin{bmatrix} X(0) \\ X(2) \\ X(4) \\ X(6) \end{bmatrix} - \begin{bmatrix} b & d & e & g \\ d & -g & -b & -e \\ e & -b & g & d \\ g & -e & d & -b \end{bmatrix} \begin{bmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{bmatrix}.$$

Figure 1.18: IDCT matrix [3]

2. Realization level of the stagese mentioned in the project proposal

2.1 FPGA

2.1.1 FPGA camera configuration

The OV7670 camera was used in this project. The datasheet was used to configure the OV7670 camera and for data capture .

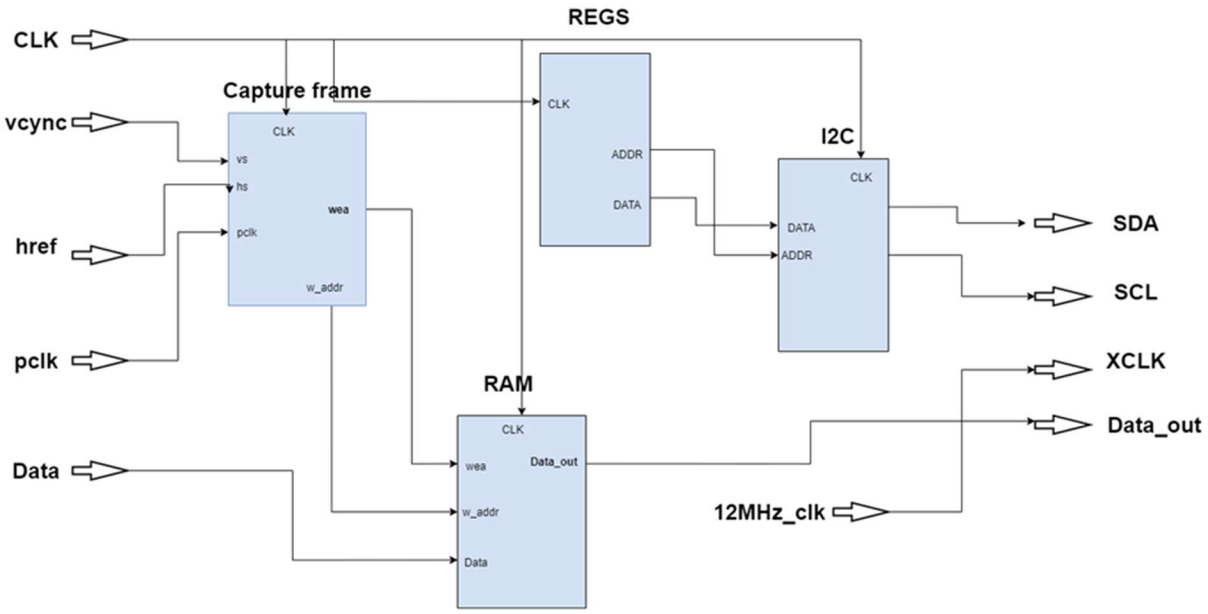


Figure 2.1: Camera configuration

The modules used for camera configuration and frame capture in this project are shown in Figure 2.1. These modules are written in Verilog. The Vertical Sync (VSync), href and pclk pins of the camera are connected to the frame capture module as inputs. In this module, 120x160 frames are captured and the address value in the RAM is recorded pixel by pixel. The regs module is used for configuring the register values of the OV7670 camera. The register values in this module are sent to the camera using the I^2C protocol. The SDA and SCL pins of the camera are connected to the I^2C module. The XCLK pin of the camera is connected to the 12MHz clock.

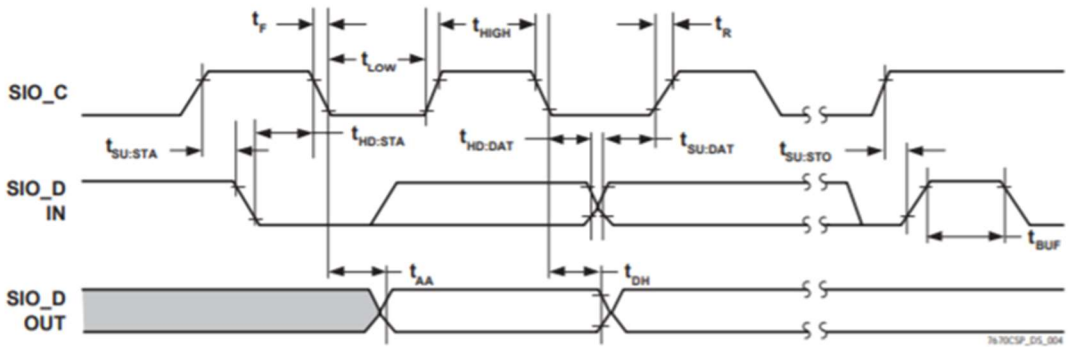


Figure 2.2 : SPI timing chart of the camera

The I^2C protocol is used to communicate with the camera. The I^2C protocol was written in Verilog using the Finite State Machine (FSM) structure. As seen in the timing chart above, the clock and data pins and the registers of the camera were accessed and their values were configured.

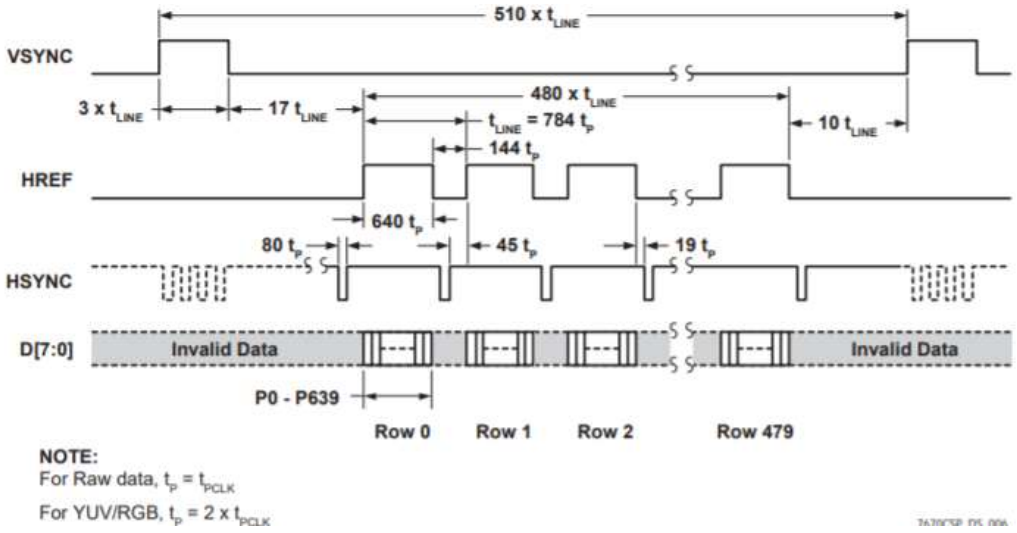


Figure 2.3: Frame capture video signals

During each frame capture process, when the VSync pin of the camera is low, the camera sends a frame. When the VSync is at the falling edge, the camera starts sending a new frame. When the VSync is low, when the href is high, the camera sends row

data. When the href is high, a row of data is sent. Since a row consists of 160 pixels in this project, every time the href is high, 160 pixel data is transmitted. This configuration was written in the project at the frame capture stage using Verilog.

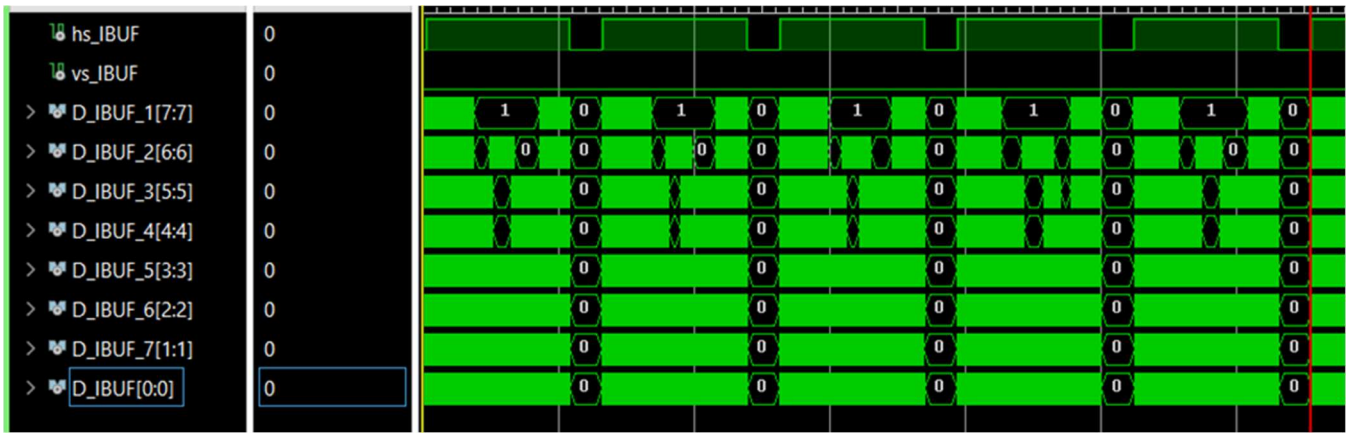


Figure 2.4: href , VSync and data outputs using ILA (Integrated Logic Analyzer) IP

```
assign frame_captured = (addr == (TOTAL_COL*TOTAL_ROW))? 1'b1 : 1'b0;

parameter idle = 1'b0,
          capture_data = 1'b1;

reg cap_state = idle;

always @(posedge pclk)
begin
if(start_capture)
begin
   case(cap_state)
       idle:   begin
                   if(VS)
                       cap_state <= capture_data;
               end

       capture_data:   begin
                           if(!frame_captured)
                           begin
                               write_to_buff <= 1'b0;
                               if(HS)
                                   begin
                                       write_to_buff <= 1'b1;
                                       addr <= addr + 1'b1;
                                   end
                           end
                           else
                               cap_state <= idle;
                       end
   endcase
end
else
   addr <= {(BUFF_BITS){1'b0}};
end
```

Figure 2.5 : Frame capture code snippet

```
always @(posedge clk)
begin
    case(addr)

        0:  b <= 16'h8012;
        1:  b <= 16'h1a3e;
        2:  b <= 16'h2272;
        3:  b <= 16'hF273;
        4:  b <= 16'h1617;
        5:  b <= 16'h0418;
        6:  b <= 16'ha432;
        7:  b <= 16'h0219;
        8:  b <= 16'h7a1a;
        9:  b <= 16'h0a03;
        10: b <= 16'h040C;
        11: b <= 16'h0012;
        12: b <= 16'h008C;
        13: b <= 16'h0004;
        14: b <= 16'hC040;
        15: b <= 16'h6A14;
        16: b <= 16'h804F;
        17: b <= 16'h8050;
        18: b <= 16'h0051;
        19: b <= 16'h2252;
        20: b <= 16'h5E53;
        21: b <= 16'h8054;
        22: b <= 16'h403D;
    endcase
end

endmodule
```

Figure 2.6 : Register configuration of the camera

2.1.2 JPEG encoding

The pixel data in YUV format coming from the camera are saved in the RAM. In the first step of the JPEG encoding part, the pixel values in the RAM are read by dividing them into 8x8 blocks. Afterwards, the 8x8 blocks DCT process is applied. After this stage, Zig-Zag and Quantization processes are performed. All stages are controlled with controller. These stages are shown in Figure 2.7.

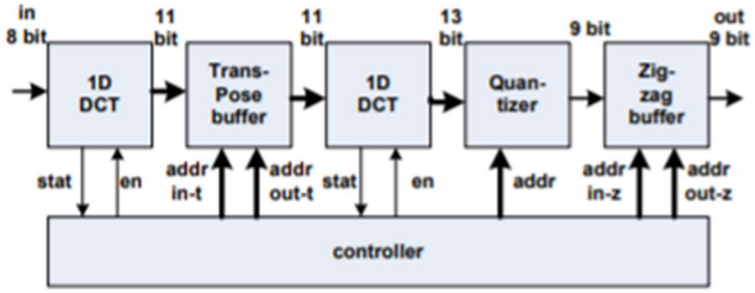


Figure 2.7 : JPEG encoding steps

2.1.2.1 DCT

In this project, the 2D-DCT process is implemented using 1D-DCT as seen in Figure 2.8. First of all, the first row of the 8x8 matrix is sent to the DCT module. After addition and subtraction, the coefficients are multiplied and sent to the transpose buffer. When the transpose at 64 pixels is saved in the buffer, the transpose is received and sent to the next DCT module and the same process is applied. In this way, the DCT process is accelerated. It is implemented on the hardware as seen in Figure 2.9.

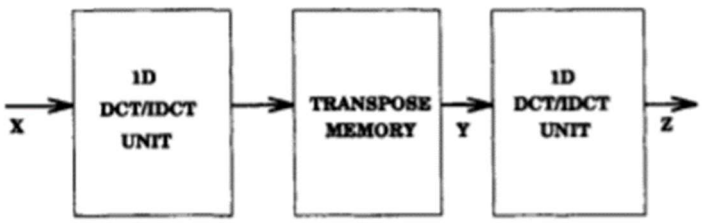


Figure 2.8 : 2-D DCT unit

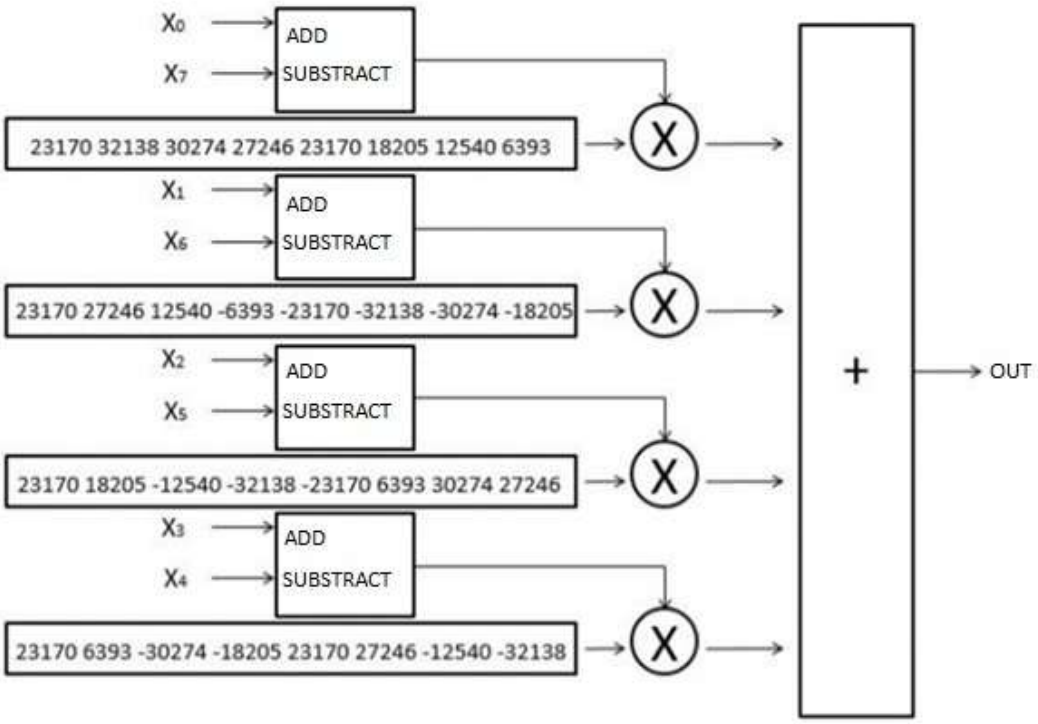


Figure 2.9: 1-D DCT hardware implementation

The coefficients used in the 1D-DCT module are shifted 16 bits to the left to be implemented in hardware, and then shifted to the right by 16 bits at the output of the DCT process.

```
always @( posedge clk)
begin
if(reset)
x[add]=0;
else if(wr)
x[add]= data_in;
end

sum_diff b1(t0_7,c0_7,x[0],x[7],1);
sum_diff b2(t1_6,c1_6,x[1],x[6],1);
sum_diff b3(t2_5,c2_5,x[2],x[5],1);
sum_diff b4(t3_4,c3_4,x[3],x[4],1);


assign y_0=a*(t0_7+t1_6+t2_5+t3_4);

assign y_1=b*c0_7+d*c1_6+e*c2_5+g*c3_4;

assign y_2=c*(t0_7-t3_4) + f*(t1_6-t2_5);

assign y_3=d*c0_7 - g*c1_6 - b*c2_5 - e*c3_4;

assign y_4=a*(t0_7 + t3_4 - t1_6 - t2_5);

assign y_5= e*c0_7 - b*c1_6 + g*c2_5 + d*c3_4;

assign y_6 = f*(t0_7 - t3_4) + c*(t2_5 - t1_6) ;

assign y_7 = g*c0_7 - e*c1_6 + d*c2_5 - b*c3_4;

assign y0=y_0>>16;

assign y1=y_1>>16;

assign y2=y_2>>16;

assign y3=y_3>>16;

assign y4=y_4>>16;

assign y5=y_5>>16;

assign y6=y_6>>16;

assign y7=y_7>>16;
```

Figure 2.10: 1-D DCT code snippet

While writing the DCT module in Verilog language, the data read from the RAM in the always block triggered by the clock was recorded in the registers. Then the recorded values were added and subtracted. It was then multiplied by the coefficients and output sequentially using the case structure.

The data from 1D-DCT is saved in the transpose buffer and transposed and sent to the second 1D-DCT. The second 1D-DCT output is saved to RAM for Zig-Zag and Quantization operations.

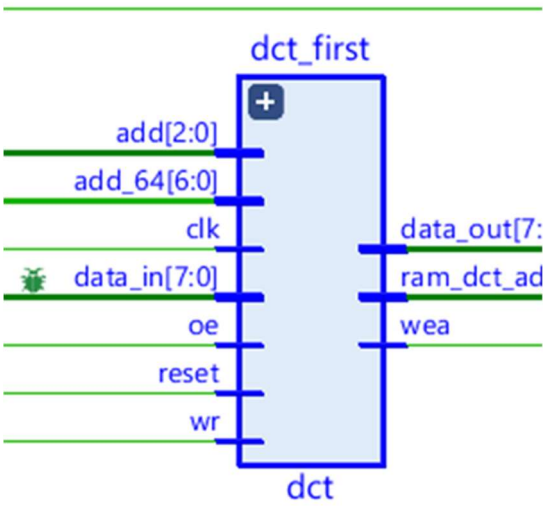


Figure 2.11: 1-D DCT block

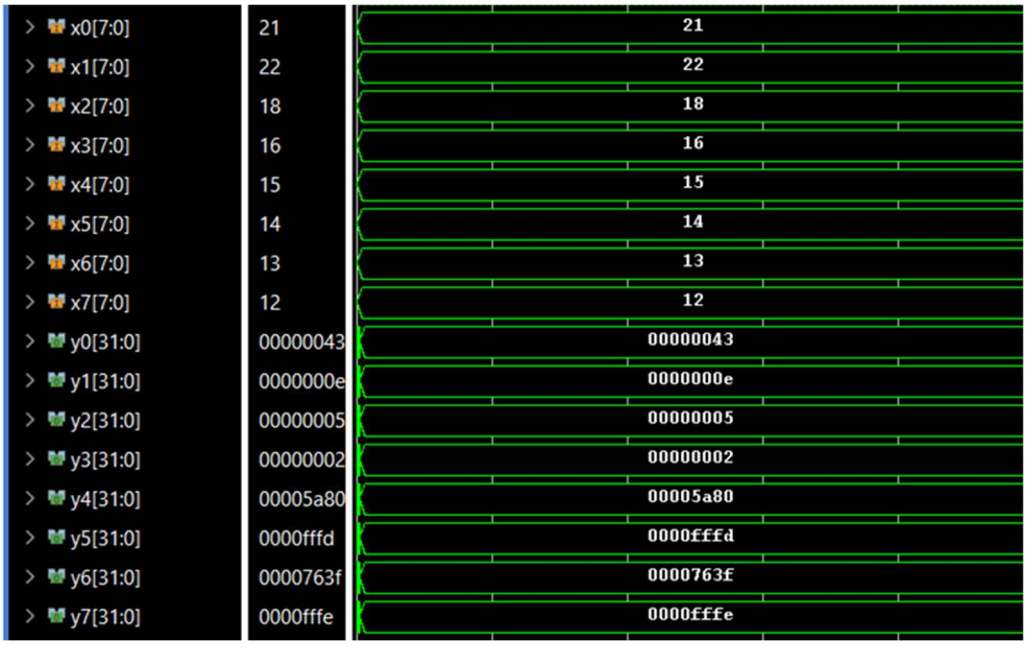


Figure 2.12: 1-D DCT simulation result

2.1.2.2 Quantization and Zig-Zag

The aim of Quantization is to suppress the highest frequency coeffiecients as these components contain less important DCT data. As we have more zeros, the image will be better compressed. The Quantization step removes a considerable amount of data and thus compresses the information significantly. This process uses the lower frequencies to reconstruct the image[9].

In this module, the Quantization matrix values in the text file are saved in ROM. Then the Quantization process is completed by dividing the 8x8 DCT values by the Quantization coefficients. The Quantization matrix we used in the project is shown in Figure 2.13.

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 2.13: Quantization matrix [9]

For the Zig-Zag operation, the text file containing the Zig-Zag values is saved to the ROM and the values are read from the RAM by taking the Zig-Zag by the control unit.

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

Figure 2.14: Zig-Zag table [10]

```
module quantizer(
    input clk,
    input [7:0] data,
    input [5:0] data_add,
    output   reg [7:0] data_out
    );

 reg[5:0]quantizer[0:63]     ; //quantizer
  initial $readmemh("quantizer.txt", quantizer);

  always@(data)
  begin
    data_out <= data / quantizer[data_add];
end
endmodule
```

Figure 2.15 : Quantizer code

2.1.2.3 Control Unit

The inputs, outputs of the modules, address controls of the RAM's and the working order of the modules are controlled by this module. A FSM is used in the internal structure of the control module. The states of FSM are setting the registers of the camera, capturing the frame from the camera, saving the captured frame to RAM, reading the data in 8x8 blocks from 160x120 RAM and transmitting it to 1-D DCT, transmitting the output values to the transpose buffer, transmitting the values from the transpose buffer to the second 1-D DCT, sending the values from the DCT to the Quantization module by Zig-Zag scanning. After the coding process, the data is transmitted with SPI.

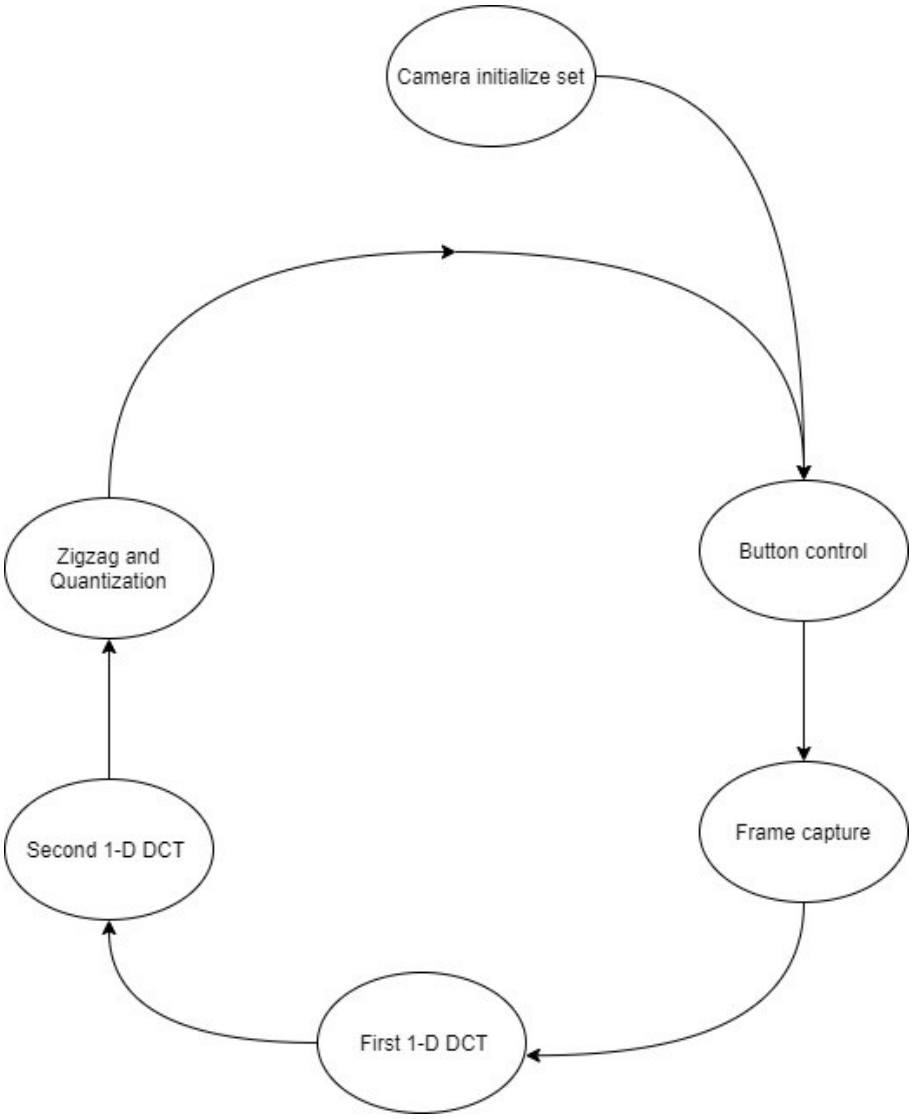


Figure 2.16 : FSM states

2.1.2.4 SPI

SPI communication protocol is used to send compressed data from FPGA to STM32. SPI communication protocol was designed with VHDL on FPGA. Compressed data is sent to STM32 as 8-bit data. In this project, since SPI communication is only from FPGA to STM32, data transfer is provided via MOSI (Master out slave in) pin. As seen in the simulation in the Figure 2.18 , the SPI module sends data at every clock cycle . As seen in Figure 2.17, the SPI protocol is designed with a case structure using FSM.

```
if rising_edge(clk) then  --write
    CASE state is

    WHEN "0000" =>
        if (enable = '1') then
            counter <= counter + 1;
            if (counter = 0) then
                data <= dataIn(7 - index);
            elsif (counter = 3) then
                sck <= '1';
            elsif (counter = 9) then
                sck <= '0';
            elsif (counter = 11) then
                data <= '0';
            elsif (counter = 12) then
                counter <= "0000";
                state <= "0001";
            end if;
        end if;
     WHEN "0001" =>
        if (index < 7) then
            index <= index + 1;
        elsif (index = 7) then
            state <= "0010";
            counter <= "0000";
        end if;
        if (index < 7) then
            state <= "0000";
        end if;

     WHEN "0010" =>
        index <= 0;
```

Figure 2.17: SPI code snippet

Figure 2.18 : SPI simulation

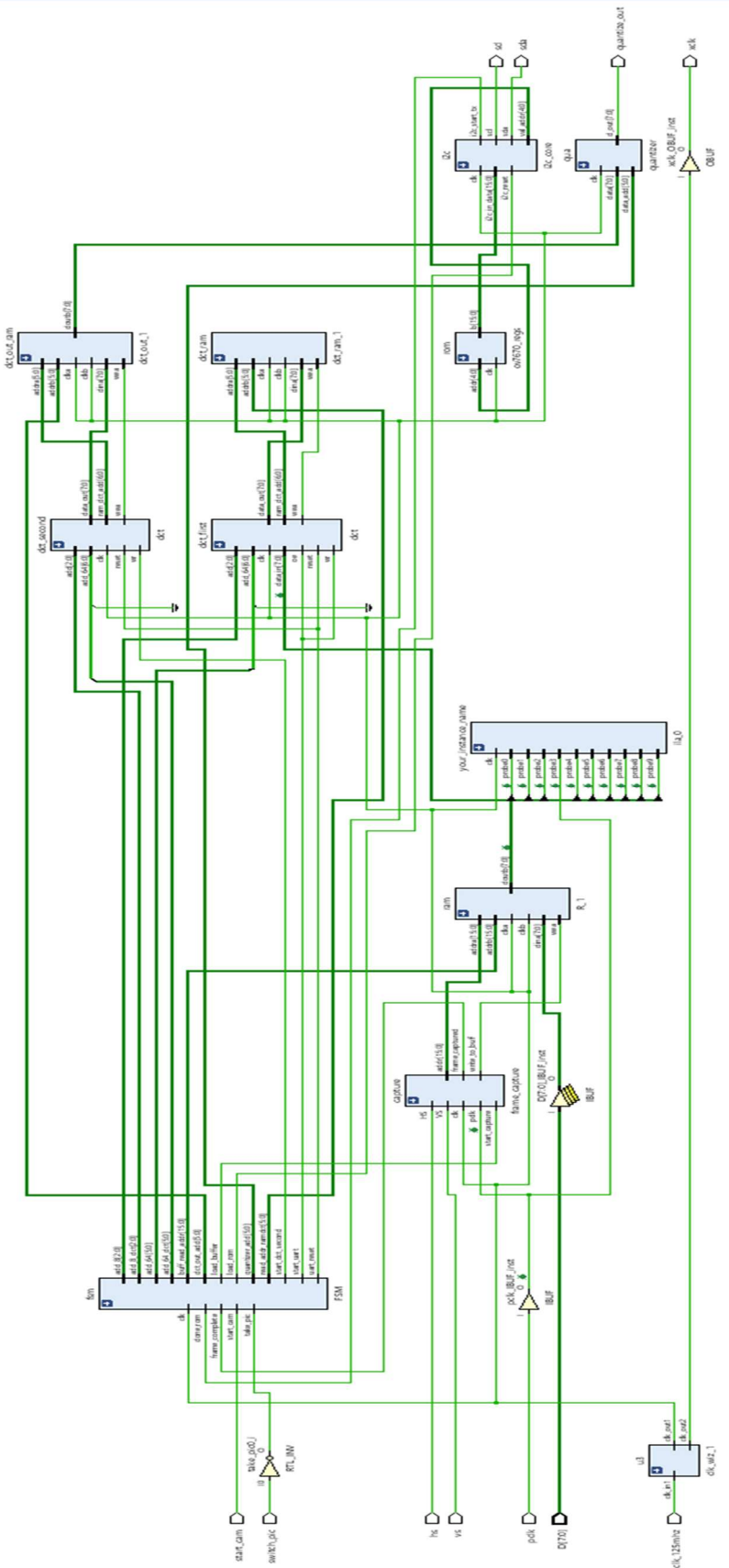


Fig 2.19: Encoding Schematic

2.1.3 JPEG Decoding

The encoded data is written into block RAM after Dequantization and inverse Zig-Zag operations. Then it is passed through 1-D IDCT and written to the transpose buffer. The transposed data is sent to the frame buffer after being re-processed with 1-D IDCT. After the frame buffer, YUV values are converted to RGB and displayed on the screen with HDMI.

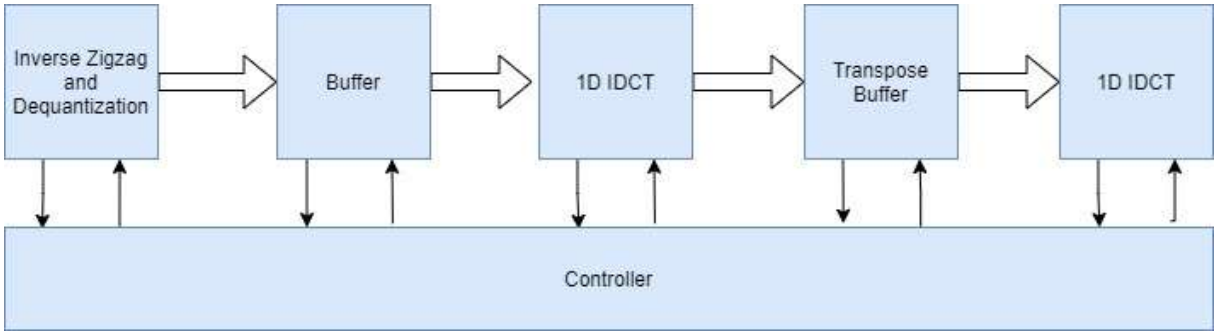


Figure 2.20 : JPEG decoding steps

2.1.3.1 Inverse Zig-Zag and Dequantization

Compressed data is Dequantized and written to RAM by Inverse Zig-Zag. While designing the Dequantization module, the Dequantization table read from the text file was used. Inverse Zig-Zag is applied by the control unit.

2.1.3.2 IDCT

2-D IDCT is designed as 1-D to speed up the process on hardware as well as encode. While the 2-D IDCT is being designed, the incoming data is passed through 1-D IDCT and then written to the transpose buffer. The transposed data is passed through 1-D IDCT again and the 2-D IDCT is completed.

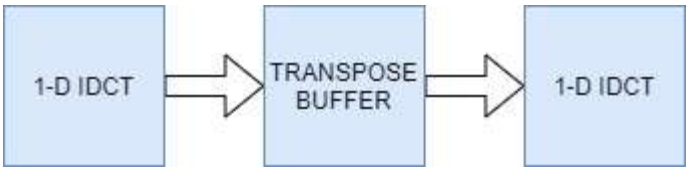


Figure 2.21 : 2-D IDCT unit

The IDCT process is performed in 2 stages to accelerate in hardware. In the first stage, 8 data is saved in RAM. It is then multiplied by the IDCT matrices and written to the transpose buffer sequentially. Then, 1-D IDCT is applied to the transposed 8x8 data again and the result is obtained.

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} a & c & a & f \\ a & f & -a & -c \\ a & -f & -a & c \\ a & -c & a & -f \end{bmatrix} \begin{bmatrix} X(0) \\ X(2) \\ X(4) \\ X(6) \end{bmatrix} + \begin{bmatrix} b & d & e & g \\ d & -g & -b & -e \\ e & -b & g & d \\ g & -e & d & -b \end{bmatrix} \begin{bmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{bmatrix}$$

$$\begin{bmatrix} Y(7) \\ Y(6) \\ Y(5) \\ Y(4) \end{bmatrix} = \begin{bmatrix} a & c & a & f \\ a & f & -a & -c \\ a & -f & -a & c \\ a & -c & a & -f \end{bmatrix} \begin{bmatrix} X(0) \\ X(2) \\ X(4) \\ X(6) \end{bmatrix} - \begin{bmatrix} b & d & e & g \\ d & -g & -b & -e \\ e & -b & g & d \\ g & -e & d & -b \end{bmatrix} \begin{bmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{bmatrix}.$$

Figure 2.22 : IDCT formula

```
assign y_0=a*x[0] + c*x[2] + a*x[4] + f*x[6] + b*x[1] + d*x[3] + e*x[5] + g*x[7] ;

assign y_1=a*x[0] + f*x[2] - a*x[4] - c*x[6] + d*x[1] - g*x[3] - b*x[5] - e*x[7] ;

assign y_2=a*x[0] - f*x[2] - a*x[4] + c*x[6] + e*x[1] - b*x[3] + g*x[5] + d*x[7] ;

assign y_3=a*x[0] - c*x[2] + a*x[4] - f*x[6] + g*x[1] - e*x[3] + d*x[5] - b*x[7] ;


assign y_4=a*x[0] + c*x[2] + a*x[4] + f*x[6] - b*x[1] + d*x[3] + e*x[5] + g*x[7] ;

assign y_5= a*x[0] + f*x[2] - a*x[4] - c*x[6] - d*x[1] - g*x[3] - b*x[5] - e*x[7] ;

assign y_6 =a*x[0] - f*x[2] - a*x[4] + c*x[6] - e*x[1] - b*x[3] + g*x[5] + d*x[7] ;

assign y_7 =a*x[0] - c*x[2] + a*x[4] - f*x[6] - g*x[1] - e*x[3] + d*x[5] - b*x[7] ;

assign y0=y_0>>16;

assign y1=y_1>>16;

assign y2=y_2>>16;

assign y3=y_3>>16;

assign y4=y_4>>16;

assign y5=y_5>>16;

assign y6=y_6>>16;
```

Figure 2.23 : IDCT code snippet

2.1.3.3 Control unit

Decode control unit provides control of modules. FSM structure was used while designing the control unit. States are determined and transitions between states are provided . The FSM structure is written in the Verilog language with the case structure as in the encode part. When the encoding process is completed for each 8x8 block, the enable pin becomes active and the decode starts. In the first case, Dequantization and Inverse Zig-Zag operations are performed. For Inverse Zig-Zag operation, the Zig-Zag table read from the text file is used. The data from the Dequantization process is written to the addresses in the RAM specified by the control unit. Then the IDCT process is completed in two stages. In the first stage, the values read line by line from the RAM are sent to 1-D IDCT. The branches coming out of 1-D IDCT are saved in the transpose buffer. In the second stage, the transposed 8x8 block is read again line by line and sent to the 1-D IDCT and the IDCT is completed. The decoded 8x8 block is written to RAM. When a frame is completed, YUV data is converted to RGB format and written to the screen using the HDMI module.

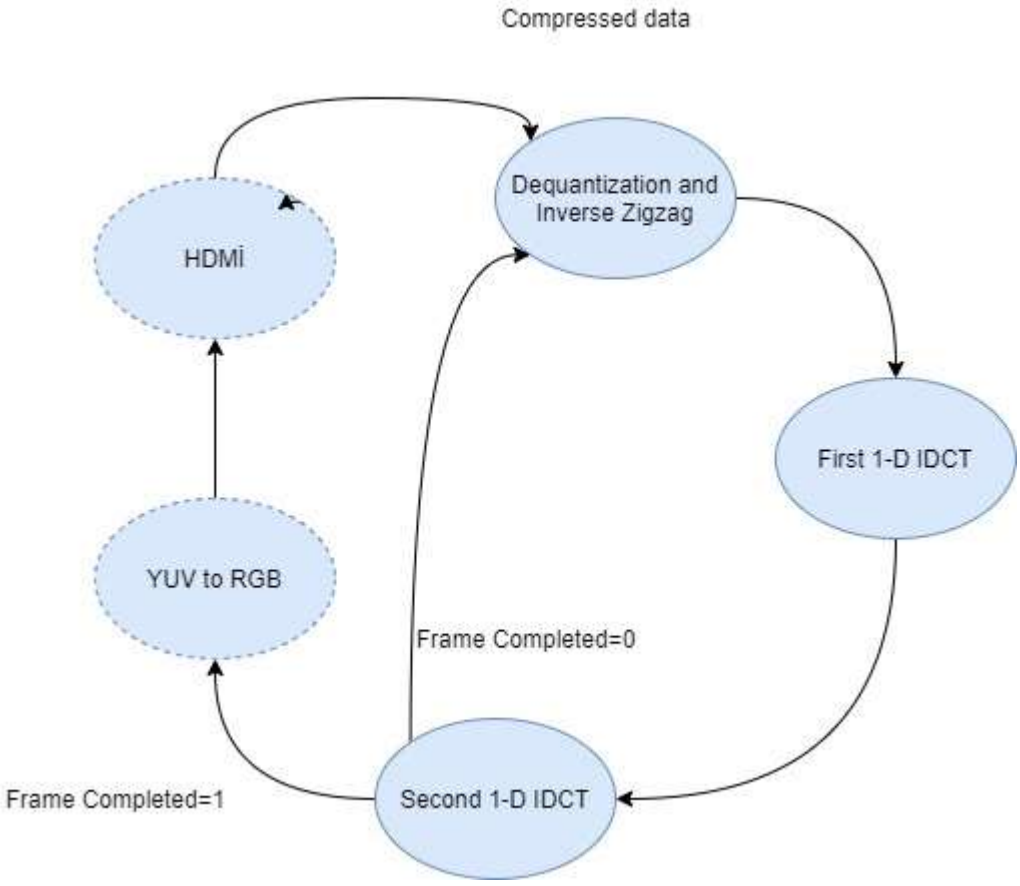


Figure 2.24 : Decoding FSM state

```
always @(posedge clk)
begin


    if(enable)
    begin
        state <= zigzag_quantization;

    end
    else
    begin

        case(state)

          zigzag_quantization: begin
                               if(count_zigzag_finish)
                                   begin
                                       count_zigzag<=6'd0;
                                       state <= idct_1;
                                    end
                               adress_buffer <=zigzag[count_zigzag];
                               count_zigzag <=count_zigzag+1;

                              end

        idct_1: begin
                               if(count_idct_finish)
                                   begin
                                       count_idct<=6'd0;
                                       state <= idct_2;
                                    end
```

Figure 2.25 : Decoding control unit code snippet

2.1.4 Display

2.1.4.1 YUV to RGB convertion

In order to display the data in the frame buffer on the screen with HDMI, YUV data must be converted to RGB format. For this reason, the YUV to RGB conversion is made by applying the matrix multiplication in the figure. While defining the coefficients, 16 bits are used to be implemented in hardware.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}.$$

Figure 2.26: RGB to YUV convertion formula

2.1.4.2 HDMI

It is necessary to produce VSync and Horizontal Sync (HSync) values for display on the screen by using the HDMI module of the RGB data obtained as a result of the decode. For this reason, VSync, HSync and data enable signals created in sync with the data were created with the signal adjust module. These signals and data were displayed on the screen with HDMI using the rgb2dvi module.

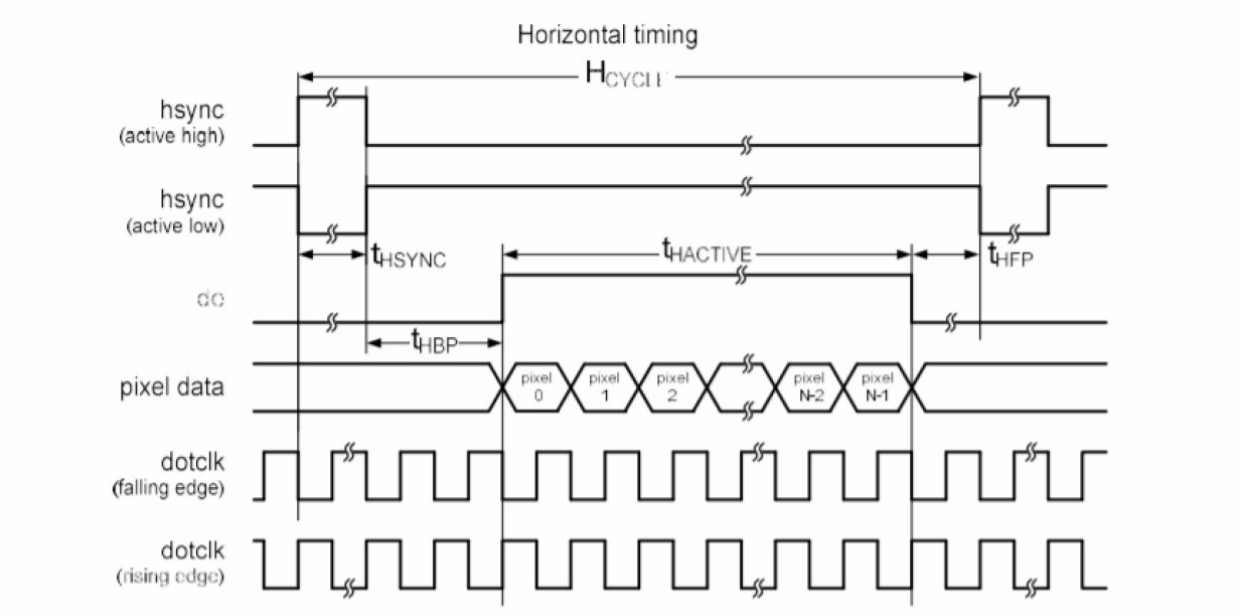


Figure 2.27 : HDMI timing diagram

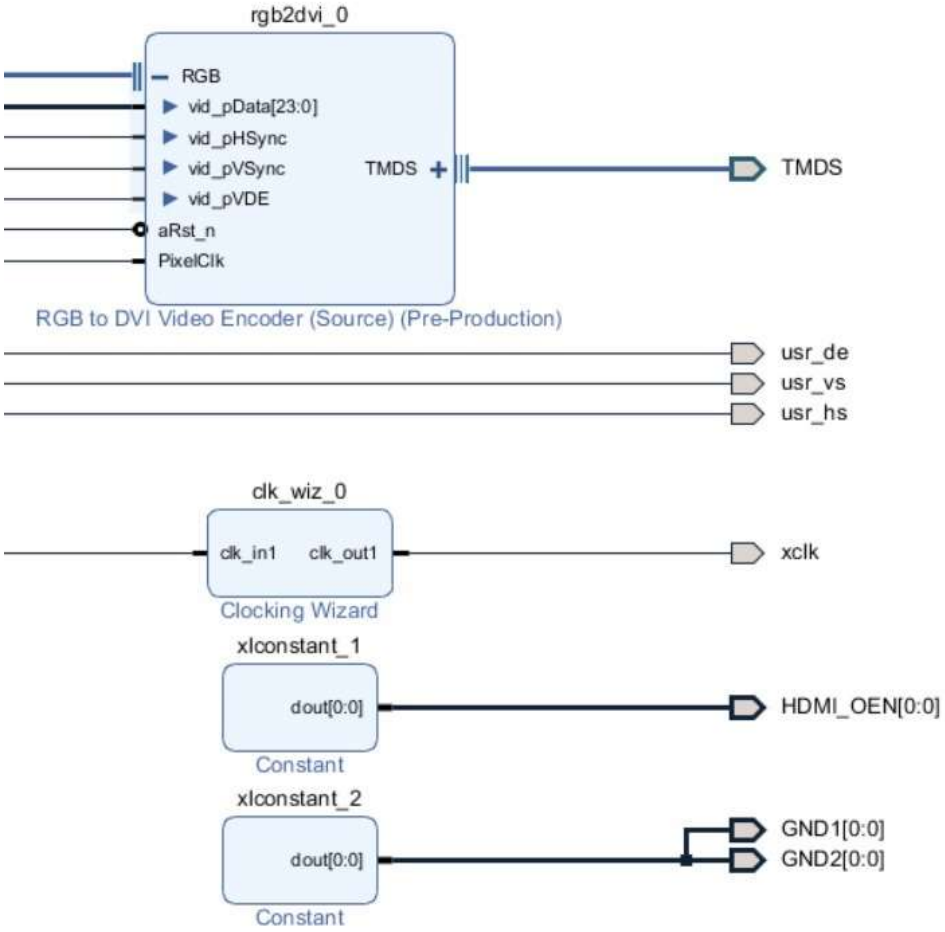


Figure 2.28 : HDMI module

2.2 STM32F4

2.2.1 STM32F4 configuration

Two SPI pins have been identified. One is set as a slave for the data to come from the FPGA, and the other is set as a master to draw the image on the screen.

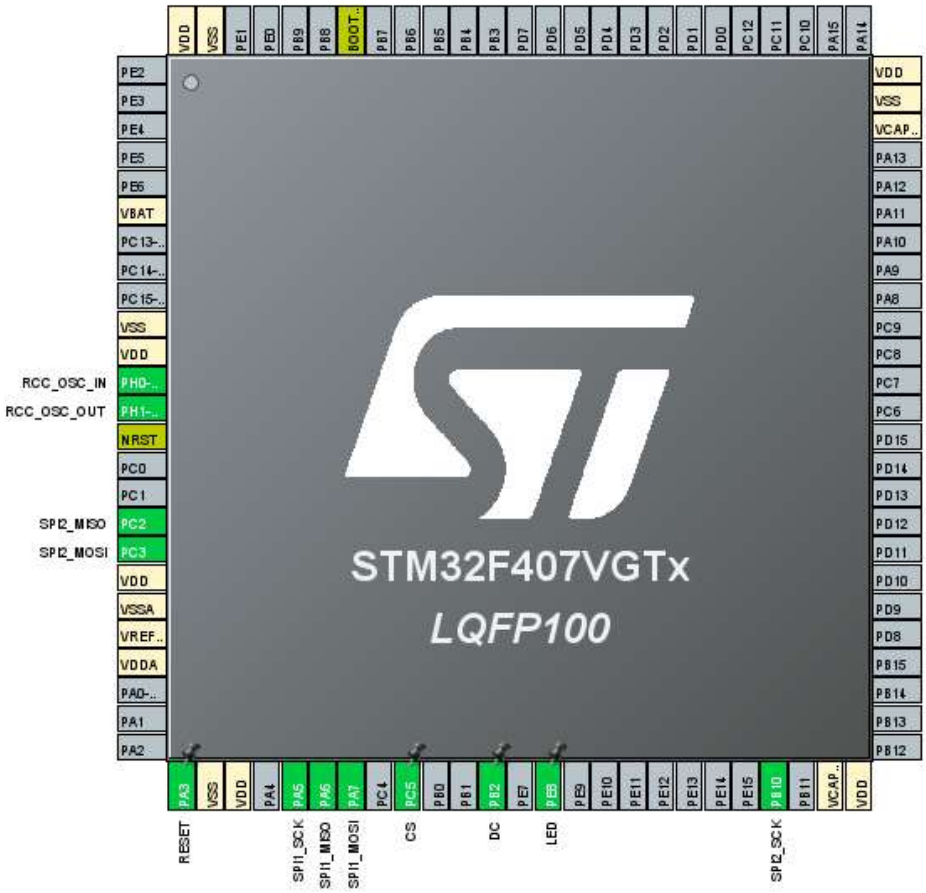


Figure 2.29: STM32CubeIDE configurations

SPI 1 pin is faster (42 Mbit/s) for data coming from FPGA, SPI 2 is (21 Mbit/s) is set to be used on the screen.

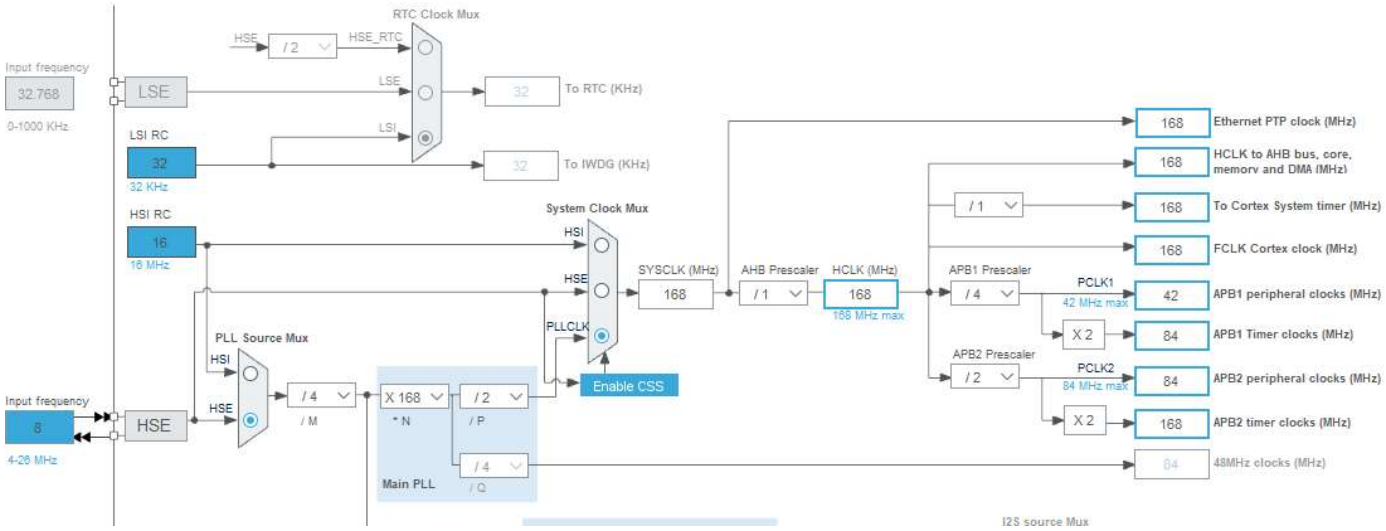


Figure 2.30: STM32 clock configuration

An external crystal clock is used when setting the clock. The maximum clock value of 168 MHz is used, the reason for the difference between SPI 1 and SPI 2 is that SPI1 uses the APB2 timer clock and SPI2 uses the APB1 timer clock.

2.2.2 JPEG decoding

Bit stream dequantization and Zig-Zag processes received from FPGA with SPI are kept in dq_buffer. After that, this buffer entering the IDCT process is kept in yuv_buffer to be converted from YUV to RGB. Then it is converted to RGB and the image is displayed on the screen.

```
while (1)
{
    //Receiving from fpga the bit stream of image data
  HAL_SPI_Receive(&hspi1, spi_buffer, 1, 1);

  //Dequantizing the received data and storing them
  for (int i=0;i<64;i++){
      dq_buffer[i] = dequantizer(spi_buffer,i);
      }

  //After the dequantization, idct function called and the frame data is constructed
  for (int j=0;j<8;j++){
          yuv_buffer[j]    = idct(dq_buffer[j],j);
          yuv_buffer[j+8]  = idct(dq_buffer[j+8],j);
          yuv_buffer[j+16] = idct(dq_buffer[j+16],j);
          yuv_buffer[j+24] = idct(dq_buffer[j+24],j);
          yuv_buffer[j+32] = idct(dq_buffer[j+32],j);
          yuv_buffer[j+40] = idct(dq_buffer[j+40],j);
          yuv_buffer[j+48] = idct(dq_buffer[j+48],j);
          yuv_buffer[j+56] = idct(dq_buffer[j+56],j);
          }


  //Converting YUV data to RGB565
  for(int k=0; k<64; k++)
  {
      frame_buffer[k] = yuv2rgb(yuv_buffer[k]);
  }

  //In order to write data to LCD, we send all the pixels repeatedly with SPI
  for(int l=0; l<19200; l++){
              *(volatile uint8_t *)&SPI1->DR = frame_buffer[l];
              while ((SPI1->SR & SPI_SR_TXE) == RESET);

      }
```

Figure 2.31: Main code snippet

2.2.2.1 Dequantization and Inverse Zig-Zag

The first step to decode the encoded data is the Dequantization process. In Figure 2.32, while y_encoded array is the data from FPGA, the dequantize array is the Dequantization table. y_quantized is the array where the data will be transferred after encoding.In the dequantizer function, the incoming data is multiplied by the table in Dequantization. As can be seen in the Figure 2.32, the elements that have changed their places after the Inverse Zig-Zag process have been brought into their normal order.Then, the data coming in the order in the Zig-Zag process is taken to the second for loop to restore it, and added to the frame8 array by editing the order in the Inverse Zig-Zag and is prepared for the IDCT process.

```
int y_quantized [64];
int dequantize [64]={16,11,10,16,24,40,51,61,12,12,14,
                     19,26,58,60,55,14,13,16,24,40,57,
                     69,56,14 ,17,22,29 ,51 ,87 ,80 ,62,
                     18 ,22 ,37 ,56 ,68 ,109 ,103 ,77,24
                     ,35 ,55 ,64 ,81 ,104 ,113 ,92,49 ,64
                     ,78 ,87 ,103 ,121 ,120 ,101,72 ,92 ,95
                      ,98 ,112 ,100 ,103 ,99};

int y_encoded [64];

int dequantizer ( uint8_t y_encoded[64], int k)
{
    int i;
    static int frame8[64];
    int zigzag[64] = {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};
    for (i=0;i<64;i++)
    {
        y_quantized[i]= dequantize[i] * y_encoded [i];
    }

    for (i=0;i<64;i++){
        frame8[i] = y_quantized[zigzag[i]];
    }

    return &frame8[k];
}
```

Figure 2.32: Dequantization and Inverse Zig-Zag code

2.2.2.2 IDCT

The bitstream encoded from the FPGA and received with the SPI communication protocol is processed in 8-bit rows with the IDCT matrix formula by using the IDCT coefficient values. The data coming according to the formula is multiplied by the IDCT coefficient values and transformed into the spatial domain.

```
float A[4][4] =      { { 0.7071,  0.38268,  0.7071,  0.55557 },
                       { 0.7071,  0.55557, -0.7071, -0.38268 },
                       { 0.7071, -0.55557, -0.7071,  0.38268 },
                       { 0.7071, -0.38268,  0.7071, -0.55557 } };

float B[4][4] =      { { 0.92388,  0.98079,  0.83147,  0.19509 },
                       { 0.98079, -0.19509, -0.92388, -0.83147 },
                       { 0.83147, -0.92388,  0.19509,  0.98079 },
                       { 0.19509, -0.83147,  0.98079, -0.92388 } };
static int y[8];

int idct (int x[8], int k)
{
        int i;
        int j;


        for(i=0; i<4; i++)
        {
            for(j=0; j<4; j++)
            {
                y[i] += ((A[i][j] * x[2*i]) + (B[i][j] * x[2*i + 1]));
            }
        }


        for(i=0; i<4; i++)
        {
            for(j=0; j<4; j++)
            {
                y[7-i] += ((A[i][j] * x[2*i]) - (B[i][j] * x[2*i + 1]));
            }
        }

        return &y[k];
}
```

Figure 2.33: IDCT code

2.2.3 YUV to RGB convertion

The image must be in RGB format to be able to print an image on the ILI9341 LCD screen. Therefore, after IDCT, the image is converted from YUV to RGB and prepared to be printed on the screen.

```
int W16;

int yuv2rgb (int frame)
{
    //YUV 422 to RGB 888
    int Y,U,V,R,G,B;

    Y = frame & 0xF0;  //bit masking with 1111 0000
    U = frame & 0x0C;  //bit masking with 0000 1100
    V = frame & 0x03;  //bit masking with 0000 0011

    Y -= 16;
    U -= 128;
    V -= 128;

    R = 1.164 * Y             + 1.596 * V;
    G = 1.164 * Y - 0.392 * U - 0.813 * V;
    B = 1.164 * Y + 2.017 * U;


    // Converting RGB 888 to RGB 565

    W16  = (R >> 3) << 11; // top 5 bits of R into top 5 bits of W16
    W16 |= (G >> 2) << 5 ; // top 6 bits of G into middle 6 bits of W16
    W16 |= (B >> 3) ;      // top 5 bits of B into lower 5 bits of W16

    return W16;
}
#endif /* SRC_YUV2RGB_H_ */
```

Figure 2.34: YUV to RGB code

2.2.4 ILI9341 display

The necessary data is sent to the registers on the LCD from STM32 via SPI. While doing this, the configurations were made using the following ILI9341 timing diagram.

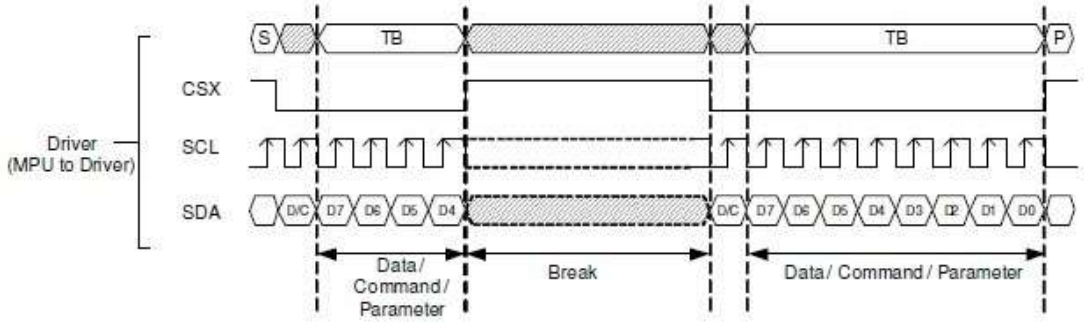


Figure 2.35 : ILI9341 configuration timing diagram

Some configuration values such as pixel resolution values, RGB data configuration and serial data stream configuration settings are transmitted to the registers specified in the datasheet, which are necessary for the screen to properly print the data coming from STM32 via SPI communication protocol to the screen.

2.3 Results

According to the synthesis result, Look up table (LUT) and Flop Lacth values are shown in Table 2.1.

Table 2.1 : Synthesis results

Module	LUT	Flop Latch
I^2C	116	77
Capture	8	18
1-D DCT	247	70
Quantization	29	
FSM	53	58

Table 2.2 : Module's Clock cycle and process time

Process	Clock cycle	Process time
1-D DCT	22	176 ns
Transpose buffer	65	520 ns
Quantization	64	512 ns
1-D IDCT	22	176 ns
Dequantization	64	512 ns

As can be seen in the Table 2.2 , an 8x8 block encoding process takes 481 clock cycles. A frame with a resolution of 320x240 has 1200 8x8 blocks. It takes a total of 577,200 clock cycles. Encoding a frame takes 4.6 ms. Since the encode process is the reverse of the decode process, 26.38 FPS is taken.

During the decode process, it takes 610 clock cycles in total, 65 clock cycles while saving the incoming data to RAM, 481 clock cycles during the decoding process of the 8x8 block, and 64

clock cycles from YUV to RGB. Since there are 1200 blocks of 8x8, it was completed in 732.000 clock cycles in total. Each frame takes 5.8 ms and a 25.57 FPS image is taken.
In STM32, 2 buffers are defined to hold frames. The frame to be decoded is transferred to the second buffer and the frame from the SPI in the first buffer is not transferred to the second buffer until the decode process is finished. For this reason, a new frame cannot be printed to the screen before the decode process is finished. In STM32, the decoding process took 153.9 ms and 5.34 FPS was obtained.

Table 2.3 : Comparison between FPGA and STM32 performance

	Resolution	FPS	Decode Time
FPGA	320x240	25,57	5,8 ms
STM32	160x120	5,34	153,9 ms

3. REALISTIC LIMITS CONCLUSIONS AND RECOMMENDATIONS

3.1 Realistic Constraints

With this project, it is aimed to implement the image compression technology, which is gaining more importance day by day, on a hardware level. Although JPEG is an old technology, it is still widely used on the Internet. This project is designed to understand image compression technology and to implement it in hardware, so it is not very suitable for use in real life. However, the image from the camera can be recorded on the SD card and the image can be drawn on the screen, and in this way a simple security camera system can be installed. Alternatively, the application such as a digital camera can also be made by using a higher quality camera and placing it in a suitable box.

3.2 Cost Analysis

The STM32F4 card costs 250 TL, the camera module costs 30 TL, and the LCD screen costs 150 TL. Although the FPGA is 2500 TL, no money has been spent because we have it. No money was paid for the software. The project was realized for exactly 430 TL.

3.3 Social, Environmental and Economic Impact

Thanks to the image compression process, large packaged images are reduced in size and can be brought into formats that can be easily accessed by everyone without the need for high system requirements and without the need to download large packages over the internet. In this way, even if you do not have a fast internet, you can easily access the images. In addition, much better decoded images are obtained with the optimization used in the codes, regardless

of any compression process. and with the ever-developing technologies, the videos and photos that people use every day are offered in a better quality way.

3.4 Standards

3.4.1 ISO/IEC 10918-1:1994

It was created by the Experts Group in 1986 to set the standard for sequential and incremental coding of grayscale and color images. The JPEG committee has established a compression standard to satisfy the needs of various applications, such as desktop publishing, visual arts, medical imaging, and scientific imaging, in addition to the applications covered by CCITT and ISO/IEC.

3.4.2 IEEE 1857.3-2013

The storage of video and audio defines both the existing features of the ISO base media file format and extensions to support certain features of the IEEE 1857 video codec and the IEEE 1857.2 audio codec. It includes, but is not limited to, application areas such as Internet media streaming, IPTV, video conferencing, etc.

3.4.3 IEEE 1364.1-2002 - IEEE Standard for Verilog Register Transfer Level Synthesis

This standard specifies the syntax and semantics for Verilog ® HDL-based RTL synthesis. By adhering to this created standard, users of synthesis tools will be able to construct well-defined designs with functional features that are independent of any particular synthesis implementation.

3.5 Diffuculties

The project faced many challenges. At first, the modules were interconnected without using a controller and were designed to be clocked. But when the control unit was not used, the frames overlapped. An FSM was used to prevent this situation. The states were determined and the transition between states was made. The disadvantage of using this structure was that one frame was not taken before the other was finished. This did not have a noticeable effect on the project, but with a proper clock setting, the project could have been done much faster.

Secondly, the project was originally written in VHDL language, but because of its complex structure and difficulties in syntax, Verilog language was preferred.

Thirdly, we were planning to decode the compressed data only on STM32, but when sending the data with SPI and after decoding, we displayed the image on the screen, but we could not get an efficient result, so we decoded the image on the FPGA and displayed the image on the screen with HDMI.

Fourthly, at the beginning of the project, the camera would be connected to the STM32 and the data would be sent to the FPGA for only DCT and IDCT operations, but since the process would be very slow, the camera was connected to the FPGA and the encoding process was carried out on the FPGA.

Finally, after the decoding of the YUV data, there were shifts in the low significant bits, which affects the color of the image, so monochrome image was used. Color components (U and V) in YUV have been omitted to make the image monochrome. Thus, the Luminance (Y) value is displayed on the screen.

3.6 Conclusion

In this project, it is aimed to accelerate the MJPEG algorithm by designing it on hardware. As a result, the 320x240 resolution image taken from the camera on the FPGA was encoded and decoded using the MJPEG algorithm and the image was displayed on the screen with hdmi. As a result of encode and decode operations with FPGA, an efficiency 25,57 FPS was obtained.

In the second part data encoded in FPGA was sent to STM32 and decoded, but high efficiency could not be obtained. When written to the screen from STM32 5,34 FPS was obtained. In STM32, FPS performance decreased by 79.2% due to the fact that parallel processing cannot be performed as in FPGA and matrix multiplication in IDCT takes longer in STM32, also due to lower clock speed.

Figure 3.1: FPGA codec HDMI result

Figure 3.2: STM32 codec ILI9341 result

3.7 Suggestions for the future

In this project, FSM is used for transitions between states. For this reason, while a block is being processed, the other block is waiting and this situation causes a waste of time. In future studies, faster processing can be done by configuring the clocks of the modules, thus achieving high FPS. The process can be accelerated even more by using components with more powerful hardware. In addition, faster processing can be achieved by using optimized algorithms other than the dct algorithms used in the project.

REFERENCES

[1] Haskell, B. G., Howard, P. G., LeCun, Y. A., Puri, A., Ostermann, J., Civanlar, M. R., ... & Haffner, P. (1998). Image and video coding emerging standards and beyond. *IEEE Transactions on Circuits and Systems for Video Technology*, *8*(7), 814-837.

[2] Poorna, M. J., Rao, M. G., Kumar, P. R., & Prasad, A. M. FPGA Implementation of Skin Tone Detection Accelerator for Face Detection. *Advance in Electronic and Electric Engineering. ISSN*, 2231-1297.

[3]Madisetti, A., & Willson, A. N. (1995). A 100 MHz 2-D 8/spl times/8 DCT/IDCT processor for HDTV applications. IEEE Transactions on circuits and systems for video technology, 5(2), 158-165.

[4] Kusuma, E. D., & Widodo, T. S. (2010, June). FPGA implementation of pipelined 2D-DCT and Dequantization architecture for JPEG image compression. In 2010 International symposium on information technology (Vol. 1, pp. 1-6). IEEE.

[5] El-Banna, H., El-Fattah, A. A., & Fakhr, W. (2003, December). An efficient implementation of the 1D DCT using FPGA technology. In Proceedings of the 12th IEEE International Conference on Fuzzy Systems (Cat. No. 03CH37442) (pp. 278-281). IEEE.

[6] Bağbaba, A. Ç. (2015). *Leon3 Mikroişlemcisi Tabanlı Sistem Gerçeklemesi* (Doctoral dissertation, Fen Bilimleri Enstitüsü).

[7] De Silva, A. M., Bailey, D. G., & Punchihewa, A. (2012, December). Exploring the Implementation of JPEG Compression on FPGA. In 2012 6th International Conference on Signal Processing and Communication Systems (pp. 1-9). IEEE.

[8] Saptariani, T., Madenda, S., & Ernastuti, W. S. (2018). Accelerating compression time of the standard JPEG by employing the quantized YCbCr color space algorithm. Int J Electr Comput Eng, 8(6), 4343-4351.

[9] Ayadi, W., Elhamzi, W., & Atri, M. (2016, November). A FPGA-based implementation of JPEG encoder. In *2016 International Image Processing, Applications and Systems (IPAS)* (pp. 1-4). IEEE.

[10] Sanjeevannanavar, S., & Nagamani, A. N. (2011, November). Efficient design and FPGA implementation of JPEG encoder using Verilog HDL. In International Conference on Nanoscience, Engineering and Technology (ICONSET 2011) (pp. 584-588). IEEE.

[11] Wallace, G. K. (1992). The JPEG still picture compression standard. IEEE transactions on consumer electronics, 38(1), xviii-xxxiv.

[12] Parate, P. P., & Mohota, N. A. FPGA Implementation of 2D-DCT for Image Compression. International Journal of Science and Research (IJSR).

[13] Deepthi, K., & Ramprakash, R. (2013). Design and Implementation of JPEG Image Compression and Decompression. International Journal of Innovations in Engineering and Technology (IJIET), 2(1), 90-98.

[14] Sun, S. H., & Lee, S. J. (2003). A JPEG chip for image compression and decompression. Journal of VLSI signal processing systems for signal, image and video technology, 35(1), 43-60.

[15]Url-1<*https://microcontrollerslab.com/stm32f4-discovery-board-pinout-features-examples/*>,accessed 10.08.2021

[16]Url-2<*https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual*>, accessed 10.08.2021

[17]cdn-shop.adafruit.com,2021[Online].Available:*https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf* [Accessed :10.08.2021]

[18]st.com,2021[Online].Available:https://www.st.com/resource/en/datasheet/dm00037051.pdf[Accessed :10.08.2021]

[19]voti.nl,2021[Online].Available:https://www.voti.nl/docs/OV7670.pdf.[Accessed: 10.08.2021]

[20]st.com,2021[Online].Available:https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf[Accessed :10.08.2021]

[21]Url-3<*https://www.iso.org/obp/ui/#iso:std:iso-iec:10918:-1:ed-1:v1:en*>,accessed 10.08.2021

[22]Url-4<*https://standards.ieee.org/standard/1857_3-2013.html*> ,accessed 10.08.2021

[23]Url-5<*https://standards.ieee.org/standard/1364_1-2002.html*>,accessed 10.08.2021

CURRICULUM VITAE

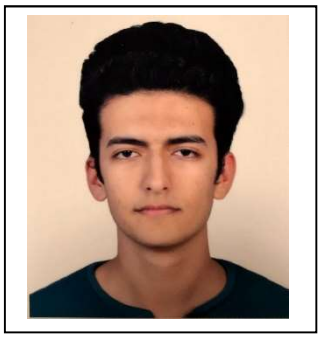
Name Surname : **Ahmet Bedri YORULMAZ**

Place and Date of Birth : **Essen-Germany/12.10.1998**

E-Mail : **yorulmaz16@itu.edu.tr**

He completed his high school education at Çamlıca Anadolu Erkek High School and started his undergraduate program in Electronics and Communication Engineering at Istanbul Technical University in September 2016. He still continues his education at Istanbul Technical University.

CURRICULUM VITAE



Name Surname **: Burak ŞİMŞEK**

Place and Date of Birth **: Mersin/ 28.07.1998**

E-Mail **: simsekbu16@itu.edu.tr**

He completed his high school education at Sesim Sarpkaya Science High School and started his undergraduate program in Electronics and Communication Engineering at Istanbul Technical University in September 2016. He still continues his education at Istanbul Technical University.

CURRICULUM VITAE

Name Surname **: Mehmet Hakan Yüksekkaya**

Place and Date of Birth **: Bursa 12.05.1998**

E-Mail **: yuksekkaya16@itu.edu.tr**

He completed his high school education at Bursa Anadolu Erkek High School and started his undergraduate program in Electronics and Communication Engineering at Istanbul Technical University in September 2016. He still continues his education at Istanbul Technical University.