VLSI Design Lab -EE705

Video Capture followed by Compression on FPGA Final Report

Simranjeet Singh (183076005) Imtiyaz Ansari (174070026) Akhil Gakhar (173079027) Abhishek Kumar (173079006)

May 3, 2019

1 Introduction

Video compression play a major roll in the new era of internet and high speed transfer. Our project is to interface the camera on FPGA followed by Video compression for fast and accurate compression. Our ultimate goal of video compression is the bit reduction of video for storage and transmission. We have researched about the various compression algorithm used currently and decided to implement the MPEG-4 (Moving Picture Expert Group) Compression algorithm. We have tested the algorithm in python before implementing to FPGA(described in later section). In parallel we are interfacing the camera with FPGA.

2 Algorithm

Compression is the technique to reduce the number of bits used in frame. There are some information in the videos that we can not see so that we can throw such irrelevant data. We can compress a video in two ways:

- Lossy - A reasonable quality of video can reconstructed
- Lossless - Original video can be constructed

We decided to use the MPEG-4 compression algorithm based on the complexity and compression parameters. MPEG-4 is good for video streaming and television broadcasting and capable of 1/50 compression factor [5].

2.1 Color System

This section describe the color system in the each frame of the image. Each frame of the video consists of 640x480 pixels. each pixel of the image is combination of different color to produce the different color. There are different ways of producing these color combinations:

- RGB: RGB is mos popular because the human eyes are easy to build up on these colors. Read green and blue color are available on each pixel. Data format of these color also vary. Normally 8-bit values to store each color.
- HSV or HSL: This is the combination of color into hue, saturation and luminance. This is more natural way to describe the colors
- YC_rC_b: This color space is used in the image and video compression because its reflects the fact that eyes are less sensitive to fine color detail of the **brightness**. In this color space one luma(y) component and two chroma(C_b and C_r) components are used to represent the image.

- Gray scale: In this value of each pixel is represented in the single sample representing the amount of light. This is kind of Black and white or monochrome images.

In color Image or video compression YC_bC_r is used. We can convert the color from one color space to another using various formulas.

2.2 Block Diagram

This the simplest way of compress the Frame of video. We have tested our algorithm on gray scale image [1].

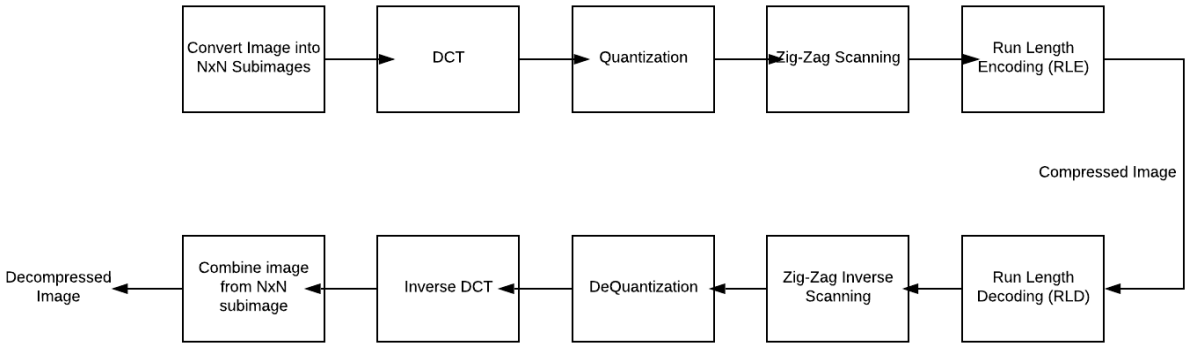

Figure 1: Block Diagram of frame compression

2.2.1 Covert Image into NxN sub image

We convert the image into sub image to make the process faster. We have divided the image into 8x8 subimage and the further operation is applied on the 8x8 sub image. As our image resolution is 640x480 = 307200 pixel and now in sub image we have 8x8 = 64 pixel. Total sub images we got is 307200/64 = 4800. The DCT (Discrete cosine transform) is performed on each subimage

2.2.2 DCT

DCT- Discrete Cosine transform is conversion of pixel values into spatial frequency transformation in vertical and horizontal orientation. This transformation is performed on 8x8 subimage means 64 pixels. It will return the 64 frequency coefficients which means no data loss after DCT.
DCT is better than Fourier transform because the transformation is not periodic and neighboring pixel vaules are two different while Fourier transform assume the periodicity at the end of each block.
We have used the Opencv Function to convert the sub image into DCT. After DCT conversion each pixel value we got into -128 to 127.
The following picture shows the transformation on DCT.

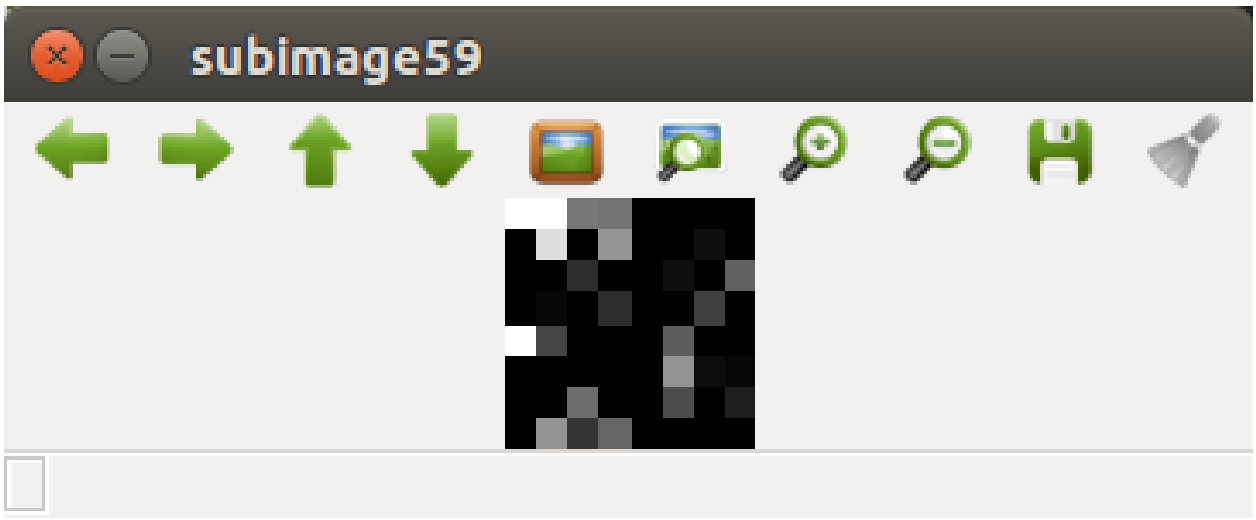

Figure 2: DCT of Subimage

Figure 3: DCT of Subimages

2.2.3 Quantization

This is the process to reduce the number of bits by divided the particular number. Suppose our values are varying between 0-255 means we have to use 8 bits to represents the range of value. if we divide the complete set of values by a factor(20) the range will be changed to 0 to 13 and we can represent this value into 4 bit.

In our case we have used the 8x8 matrix to divide the each pixel of the subimage by a factor. Each pixel of image subimage[0][0] to subimage[7][7] is divided by different set of value.
The same value should be used during the dequatization to retrieve the original value

2.2.4 Zig-Zag scanning

Zig-Zag scanning is used to collect the number of values in the same order. Actually its a conversion of $2-D$ array of image into $1-D$ array so that Run length encoding can be perform. an example of this scanning is shown below:

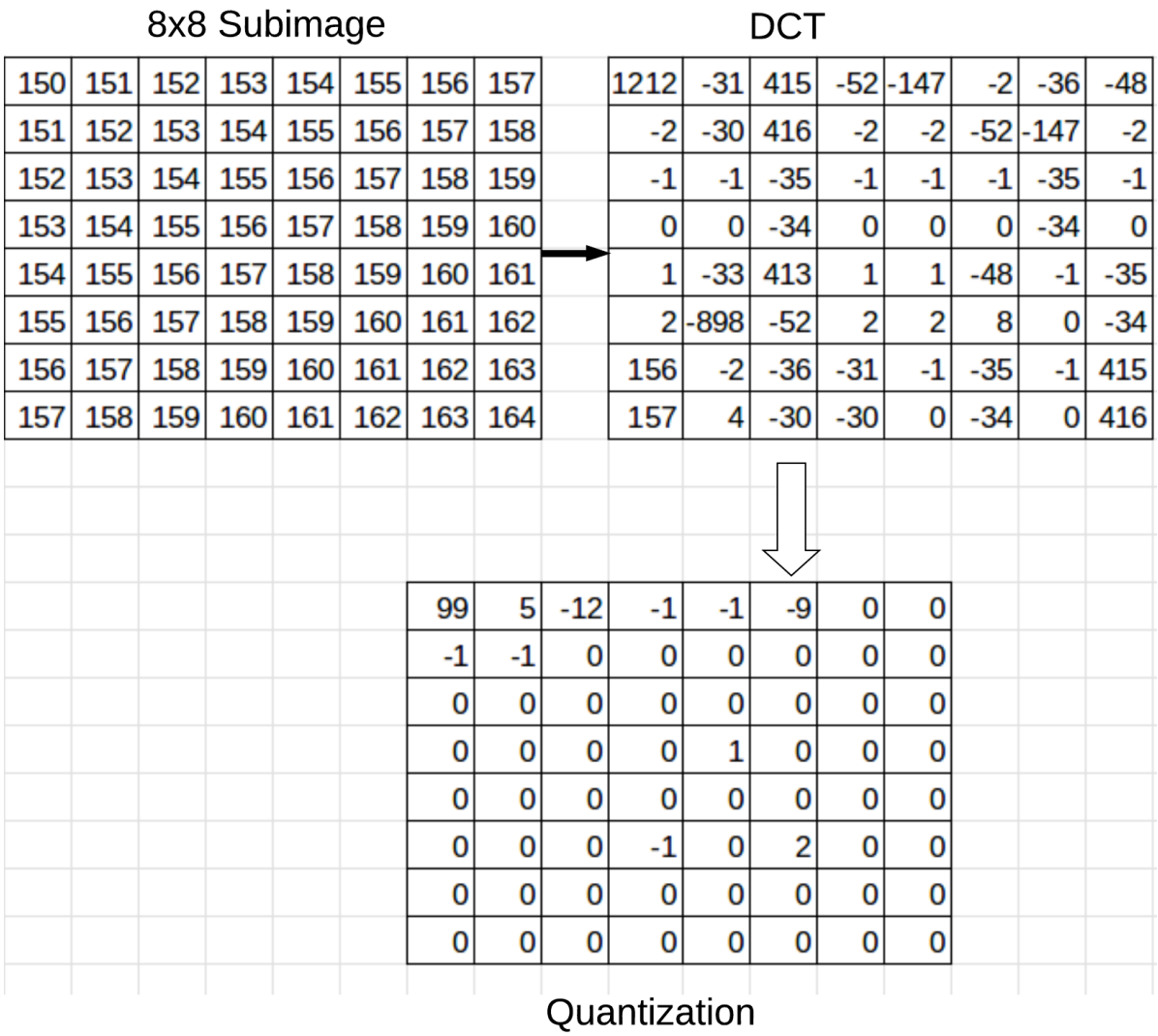

Figure 4: Conversion

Note: The above image data is dummy data for example.
As you can see the data after quantization in the table. There are number of zeros in the image matrix. we will scan this image in zig-zag manner so that number on zeros come together. this will help us to perform the run length encoding.

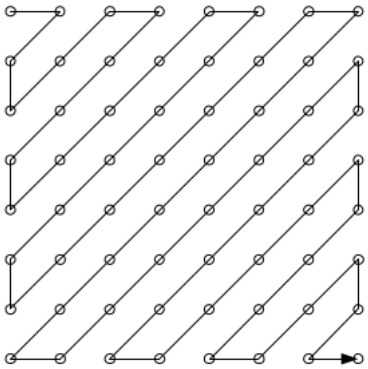

Figure 5: Zig-Zag scanning of Image [4]

Zig-Zag scanning return the $1-D$ array. [[data]] to [data]. Using this array i can perform the run length encoding. This scanning is perform on 8x8 subimage. Combined image after the zigzag scanned is shown below:

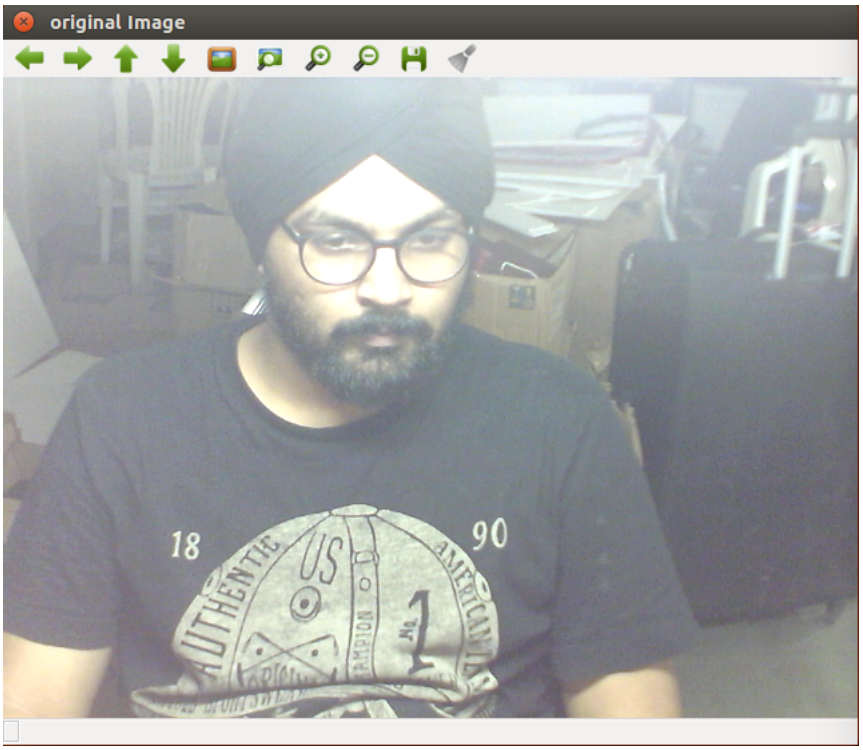

Figure 6: Original Image

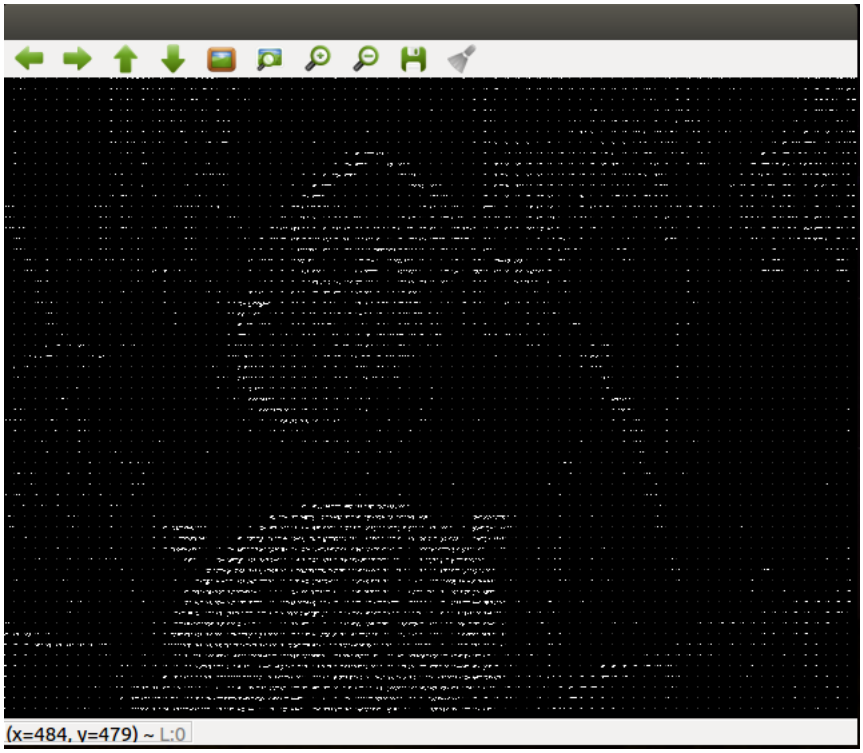

Figure 7: Scanned image

You can see the original image is in the format of RGB. This image shown is without run length encoding and padded on the black image. You can observe the shape of body in the pixel.

2.2.5 Run Length encoding(RLE)

RLD is Lossless encoding used in the compression. As previously discussed we have changed the image into array. This image is stored in zig-zag manner. We can perform the encoding and reduce the number of bits.

- Example: consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. A hypothetical scan line, with B representing a black pixel and W representing white, might read as follows:

  WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWW
  WWWWWWWWWWWWWWWBWWWWWWWWWWWWWW

  With a run-length encoding (RLE) data compression algorithm applied to the above hypothetical scan line, it can be rendered as follows:

  12W1B12W3B24W1B14W This would be interpreted as a run of twelve Ws, a B, a run of twelve Ws, a run of three Bs, etc. In data where runs are less frequent, this can significantly improve the compression rate [Wikipedia of run length encoding]

  We have implemented this run length encoding on the 640x480 image and converted it in small size (Exact compression size is to be measured).

2.2.6 Decompression

Decompression is done by reversing the each step mentioned in the block diagram and able to extract the original image. As we have done the compression n gray scale the final image will be in black and white format. Check the original image constructed from the final compressed video [6].

Figure 8: Decompression

3 Hardware

We list and discuss here the main hardware components that we use in this project.

3.1 OV7670 CMOS Camera Module

The OV7670 is a CMOS image sensor + DSP that can operate at a maximum of 30 fps and 640 x 480 ("VGA") resolutions, which is the equivalent of 0.3 Mega Pixels. The captured image can be pre-processed by the DSP before sending it out. This preprocessing can be configured via the Serial Camera Control Bus (SCCB) or through I2C interface. Note that there are actually two versions of this camera module. The simpler and cheaper one is shown in Fig.2.1 - and it is the one we use in this project.

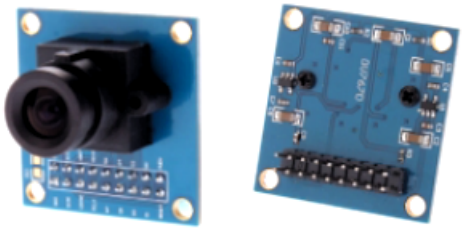

Figure 9: OV7070 camera module.

Signal	Usage	Active
3V3	3.3 power	
GND	Ground	
SIOC	Serial command bus clock (upto 400Khz)	Active high, configurable
SIOD	Serial command bus Data	Active high, configurable
VSYNC	Vertical Sync	
HREF	CE output for pixel sampling	
PCLK	Pixel Clock	
XCLK	System clock (10-48 Mhz, Typ. 24 Mhz)	
D0-D7	Pixel Data	
RESET	Device Reset	Active Low
PWDN	Device Power Down	Active High

Table 1: I/O signals of the camera module

A video is a succession of frames. A frame of the video is a still image taken at an instant of time. A frame is compromised of rows (or lines), and a row is compromised of a number of pixels - whose number on the row equals the number of columns. A pixel is the smallest part of a digital image, and it looks like a colored dot. For example, a simple 4x3 image is shown below.

	C0	C1	C2	C3
R0	0,0	0,1	0,2	0,3
R1	1,0	1,1	1,2	1,3
R2	2,0	2,1	2,2	2,3

Figure 10: Image with 4x3 pixels

There are many pixel formats. Some of the simplest pixel formats include monochrome and RGB. In monochromes images, each pixel is stored as 8 bits, representing gray scale levels from 0 to 255, where 0 is black, 255 is white and the intermediate values are shades of gray. In the RGB color model, any color can be decomposed in Red, Green and Blue light at different intensities. Because a color can be made by mixing Red, Green and Blue, it is called the RGB color system or model. It is also called an "Additive" color system, because it starts at black, and then color is added. Using this model, each pixel must be stored as three intensities of these red, green and blue lights. The most common format is RGB888. In this format each pixel is stored using 24 bits - the red, green and blue channels are stored in 8 bits each:

RRRRRRRRGGGGGGGGBBBBBBBB

For instance, the color red would be stored, in binary, as 24 bits as:

111111110000000000000000

or, as commonly shown in hexadecimal, as: FF0000
However, the RGB formats used by the OV7670 are: RGB565, RGB555 and RGB444. The difference, compared to RGB888 format, is the number of bits assigned to each channel. For example, in the RGB565 format, the red channel is stored as 5 bits, the green channel as 6 bits and the blue channel as 5 bits. These formats take less memory when stored but in exchange sacrifice the number of colors available.

Retrieving data from the camera module is relatively simple. As can be seen in the datasheet, and shown in Fig.2.3 and 2.4, signals VSYNC and HSYNC give us the "points of reference" we need to know when pixel data of a frame are being transmitted from the camera module.

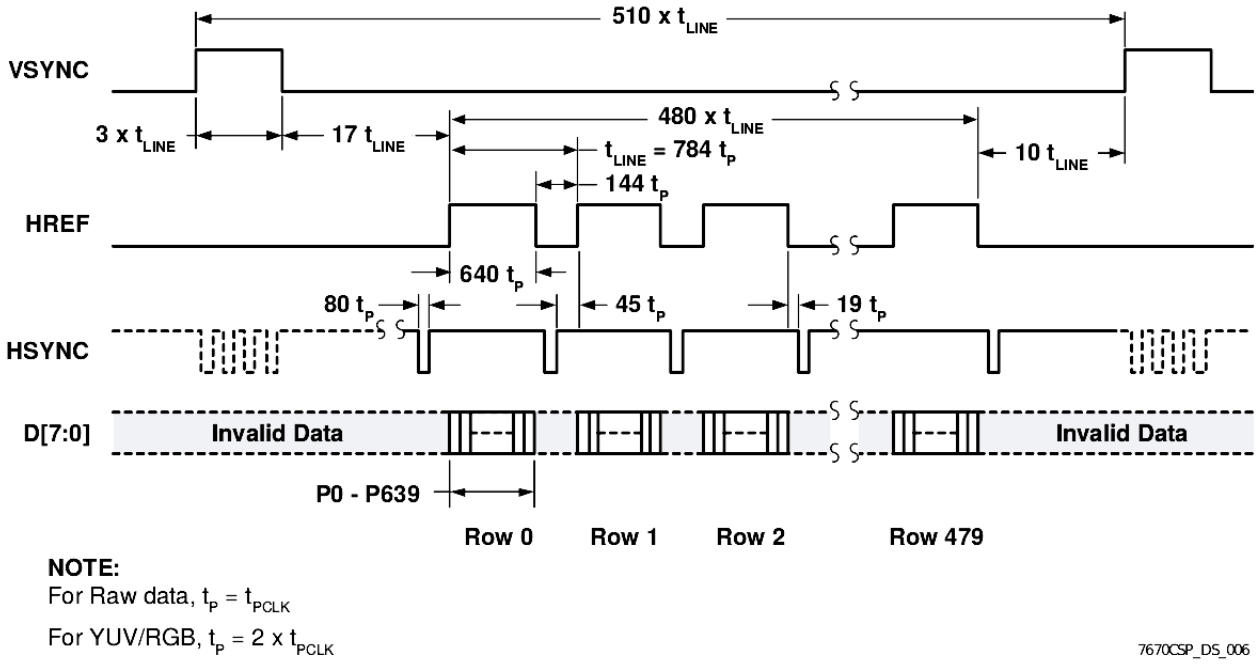

Figure 11: VGA frame timing

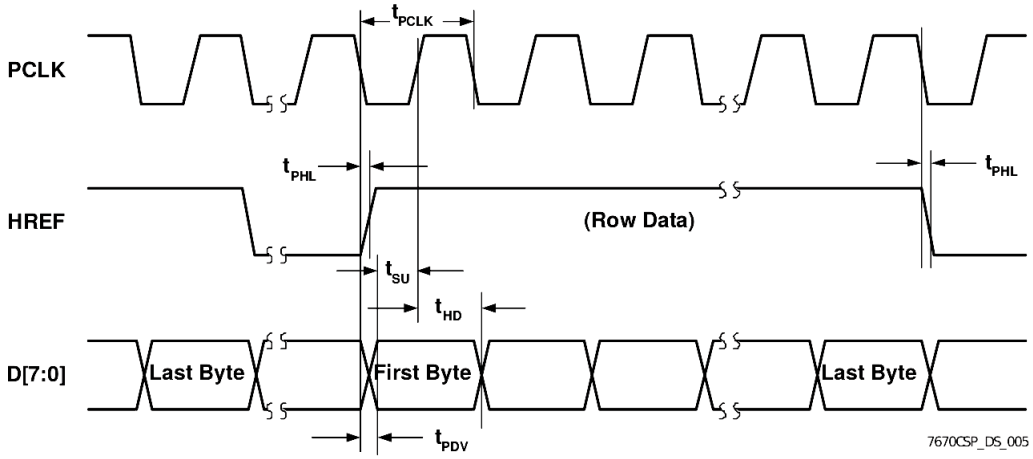

Figure 12: RGB 565 output timing diagram

3.2 Architecture

We simply build a preliminary design where we simply connect the OV7670 camera module to the FPGA, retrieve video frames from the camera module, store temporarily each image frame's data inside the FPGA, and use that data to drive a VGA monitor connected to the board. We'll complicate this design in follow-up implementations. Note that each implementation is a stand alone design by itself.

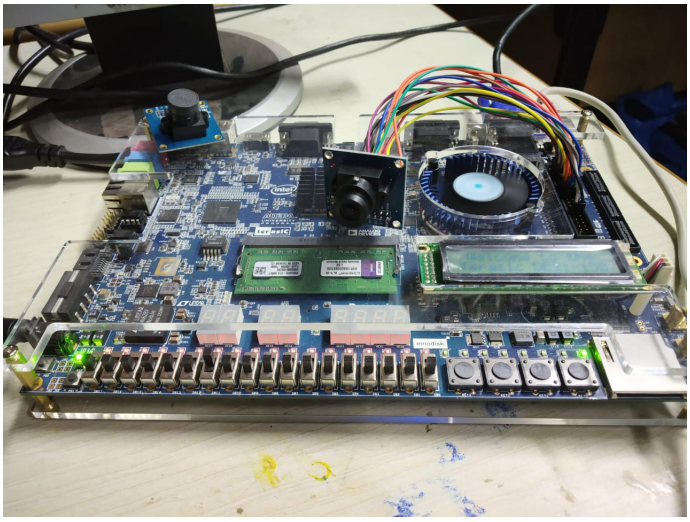

Figure 14: DE2i-150 Board

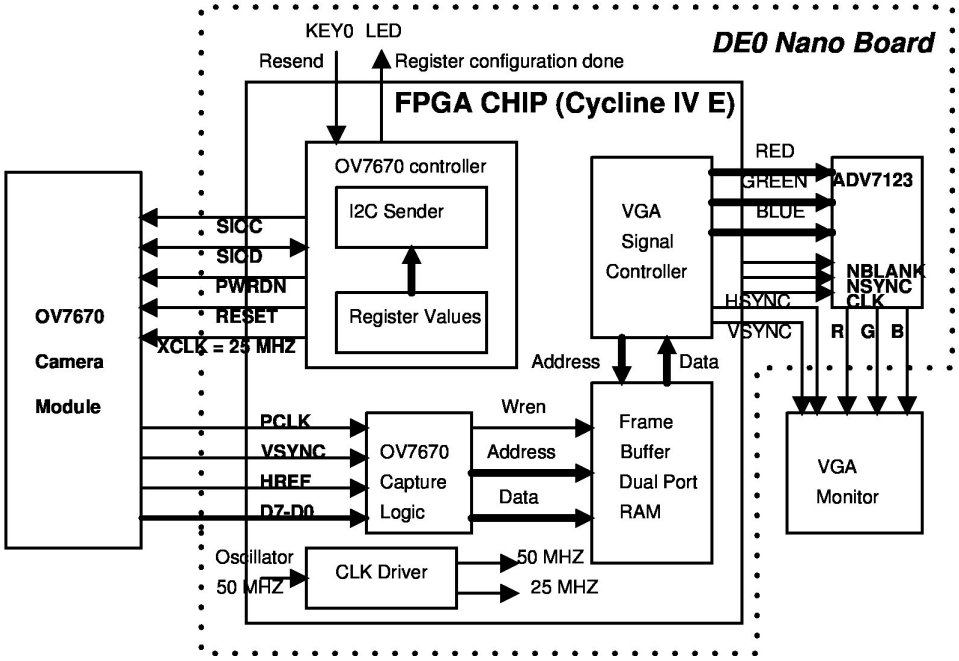

Figure 13: Block diagram of the implementation

4 Hardware Implementation

We used the DE2i-150 Board for the implementation because of provided SDRAM size and inbuild VGA interfacing.

4.1 Camera interfacing

Camera OV7670 interfaced with the FPGA and output of camera is shown on the monitor. Ouput of camera is shown in the following figure:

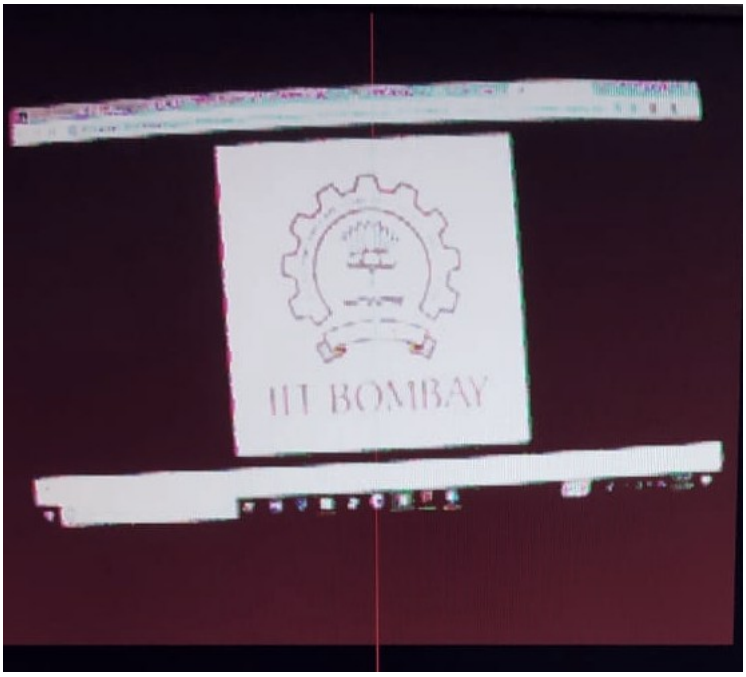

Figure 15: OV7670 Image on VGA monitor

4.2 VHDL

VHDL Implementation of above blocks was done as far as Video Compression is concerned, We tested individual blocks by their corresponding test-benches. The problems that we faced were non-availability of pads on the device as a result Gate-Level simulations were not performed. Improvising our code further, next challenging task was to Implement trigonometric functions, We tried different approaches including look-up table etc. We shifted over to NIOS Implementation owing to ease of integrating all modules and flexibility offered by it.

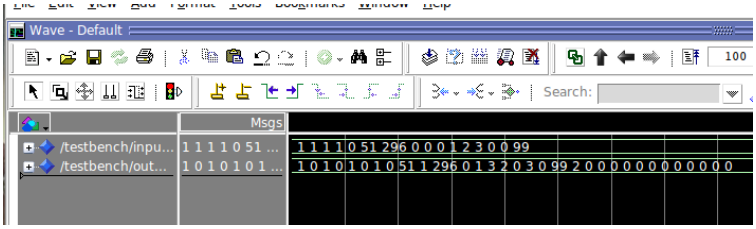

Figure 14: Run Length Encoding

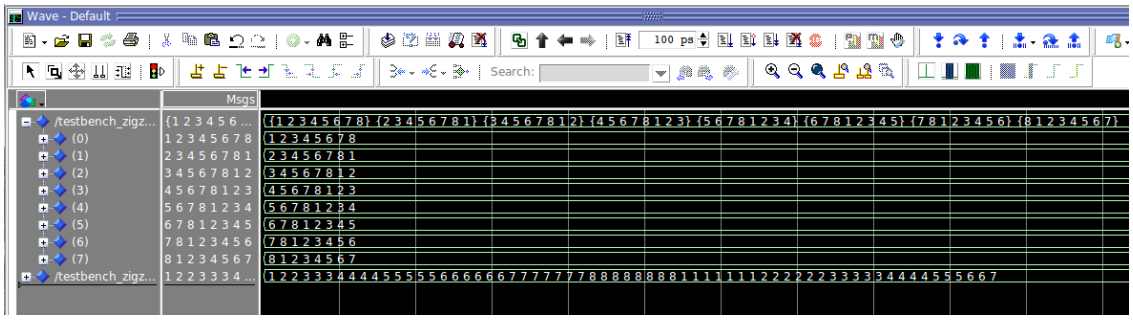

Figure 15: Zig_Zag Encoding

4.3 NIOS PROCESSOR

We implemented the compression algorithm on NIOS-II as the Mathematical function is easy to implement in the NIOS processor. We have decided to use the SDRAM from Platform designer along with CPU. DE2i-150 board is used for NIOS project as this board has 128MB SDRAM compare to DE0-NANO board which has 32MB SDRAM. DE2i-150 board provide 2 SDRAM each 64MB with 13 bit address and 16 bit different data bus.

4.3.1 Implementation

We decided to send the 8x8 matrix to NIOS[2] for video compression. To generate the 8X8 subimage we have written a test bench in VHDL that read the each pixel value from the text file.This 8x8 subimage is passed to top level NIOS-Handler to transfer the data to NIOS-Processor.

We used 16 PIO(parallel IO) of 32-bit each to send the 8x8 matrix data. Each PIO can send 4 byte of data and total 64 byte data can be passed to NIOS through Avalon bus.

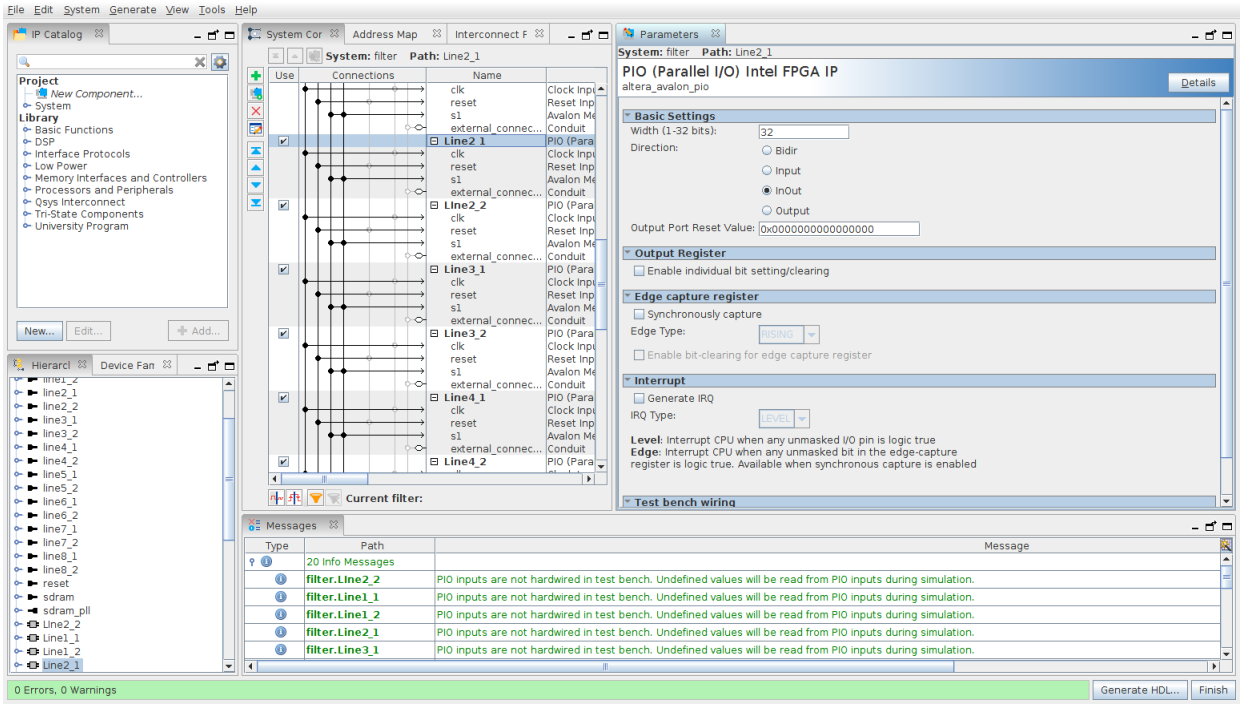

Figure 16: PIOs for 8x8 matrix

These PIOs are converted to register to avoid the PIN Assignments. 4 byte data can be transfer to each line and accessed by the NIOS for further computation. The following figure shows the 8x8 data transfered to NIOS avalon bus and out DCT[3] and quantization on matrix.

Output the simple 8x8 matrix is shown in the figure 15. The matrix is generated by reading the 32 bit line as:

NIOS C Code

```
    for(t = 0; t < 8; t++) //rows
    {
            for(r = 0; r < 8; r++) //column
            {
                    if(t == 0 && r < 4)  //read 4 byte at a time
                    {
                            matrix[t][r] =IORD_8DIRECT(LINE1_1_BASE, r);
                    }
                    if(t == 0 && r >= 4)
                    {
                            matrix[t][r] =IORD_8DIRECT(LINE1_2_BASE, r-4);
                    }
    }
}
```

Verilog Code

```
Line1_1_IN <= {matrix_8[1], matrix_8[2], matrix_8[3], matrix_8[4]};
Line1_2_IN <= {matrix_8[5], matrix_8[6], matrix_8[7], matrix_8[8]};
Line2_1_IN <= {matrix_8[9], matrix_8[10], matrix_8[11], matrix_8[12]};
Line2_2_IN <= {matrix_8[13], matrix_8[14], matrix_8[15], matrix_8[16]};
Line3_1_IN <= {matrix_8[17], matrix_8[18], matrix_8[19], matrix_8[20]};
Line3_2_IN <= {matrix_8[21], matrix_8[22], matrix_8[23], matrix_8[24]};
Line4_1_IN <= {matrix_8[25], matrix_8[26], matrix_8[27], matrix_8[28]};
Line4_2_IN <= {matrix_8[29], matrix_8[30], matrix_8[31], matrix_8[32]};
Line5_1_IN <= {matrix_8[33], matrix_8[34], matrix_8[35], matrix_8[36]};
Line5_2_IN <= {matrix_8[37], matrix_8[38], matrix_8[39], matrix_8[40]};
Line6_1_IN <= {matrix_8[41], matrix_8[42], matrix_8[43], matrix_8[44]};
Line6_2_IN <= {matrix_8[45], matrix_8[46], matrix_8[47], matrix_8[48]};
Line7_1_IN <= {matrix_8[49], matrix_8[50], matrix_8[51], matrix_8[52]};
Line7_2_IN <= {matrix_8[53], matrix_8[54], matrix_8[55], matrix_8[56]};
Line8_1_IN <= {matrix_8[57], matrix_8[58], matrix_8[59], matrix_8[60]};
Line8_2_IN <= {matrix_8[61], matrix_8[62], matrix_8[63], matrix_8[64]};
```

We have successfully done the Compression on NIOS processor. But the main issue is the processing speed of NIOS. We have tried with NIOS-f.

5 Challenges

- Finding the best suited algorithm for FPGA implementation and should be doable in current course period. We have decided to implement the MPEG-4 algorithm. But due to lack of resources we implemented MJPEG algorithm.
- Implementation of 2-d DCT in VHDL is very challenging as there are no mathematical functions are available(sin and cos). We have to implement it using CORDIC transform.
- NIOS SDRAM interfacing was a challenge because some of the pin used by the altera nCEO. Pin assignments was giving error.

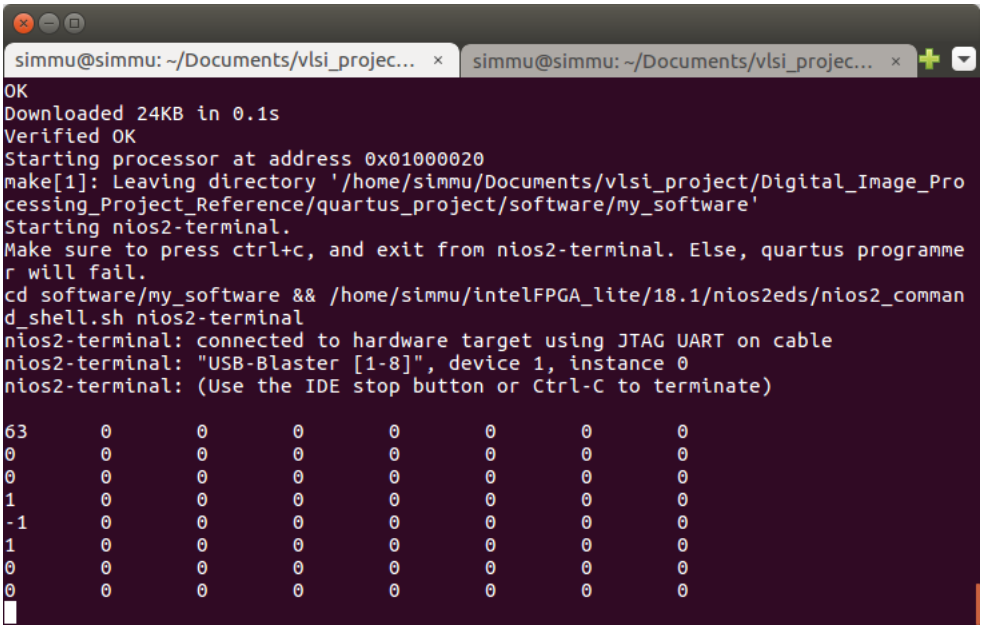

Figure 16: NIOS Output 8x8

6 Future Work

Implementation of DCT and other algorithm in NIOS is very slow. This complete code can be written in HDL from scratch and pipeline of the 8x8 matrix can be done to improve the speed of operation.

7 Resources Used in FPGA

- NIOS-II/E - DCT and quantization implementation.
- SDRAM: 64MB, 12-Bit address, 32-bit data line
- IP RAM: Dual Port RAM for storing Camera image.
- ALT PLL: Synchronization of SDRAM and VGA clocks.
- 16 PIOs: 32-bit Parallel input output port.

8 Individual Contribution to Project

We divided the project in two section as

- Compression Algorithm Development - Simranjeet Singh(183076005), Akhil Gakhar(173079027)
- Camera Interfacing- Imtiyaz Ansari(174070026), Abhishekh(173079006)

References

[1] Bogo to Bogo. *Digital Image compression*. URL: https://www.bogotobogo.com/OpenCV/DigitalImageProcessing_JPEG_Compression_Under_Sampling_DCT_Quantization_Entropy_Hoffman_Encoding.php. (accessed: 03.05.2019).

[2] Intel. *Nios II Core Implementation Details*. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu_nii51015.pdf. (accessed: 03.05.2019).

[3] J.SriKrishna. “DESIGN OF 2D DISCRETE COSINE TRANSFORM USING CORDIC ARCHITECTURES IN VHDL”. In: Mtech Thesisi, 2007. Chap. 5.

[4] Lain E.G Richardson. “H.264 and MPEG-4 Video compression”. In: Wiley, 1999. Chap. 5.

[5] S.Ponlathaand R.S.Sabeenian. “Comparison of Video Compression Standards”. In: International Journal of Computer and Electrical Engineering (2013).

[6] Linkopings University. *Course material*. URL: http://www.bk.isy.liu.se/courses/tsbk06/material/lect8-9b.pdf.