Development of a Drowsiness Detection System using FPGA based Image Capture and Processing

By

Wong Jang Sing

Faculty of Applied Sciences and Computing
Tunku Abdul Rahman University College
Kuala Lumpur

2016/2017

Final Year Project

Development of a Drowsiness Detection System using FPGA based Image Capture and Processing

By
Wong Jang Sing

Project supervisor
Miss Michelle Lim Sern Mi

This is a project dissertation submitted to the Faculty of Applied Sciences and Computing in partial fulfillment of the requirement for the award of Bachelor of Science Degree, Tunku Abdul Rahman University College and Campbell University, Buies Creek, U.S.A.

Department of Physical Science
Faculty of Applied Sciences and Computing
Tunku Abdul Rahman University College
Kuala Lumpur

Acknowledgement

I would like to express my gratitude towards my Final Year Project (FYP) supervisor, Miss Michelle Lim Sern Mi. She had given much guidance and ideas on problem solving. Also I am highly appreciative that she has encouraged me to not give up and continue on the project. Thanks to her patience on spending time and providing me with good guidelines and proper time managements leading me to the project completion. She has also guided me during pre-presentation and pre-demonstration which made my presentation pass smoothly.

Furthermore, I enjoy doing this project and learned a lot more about FPGA and its extra features. If given a second chance of doing another project, I would like to try out on more features inside the FPGA hardware as well as projects that interfaces with on board peripherals e.g. LCD, PS2, audio codecs etc. I think some of these features can be incorporated in mini projects for Microelectronics based subjects, so that juniors can have a try on the features and acknowledge that actually the FPGAs can perform this kind of tasks.

Abstract

This research focuses on the image capture and processing of the Driver Drowsiness Detection System. The image capturing process requires several processing steps. The proposed architecture uses a 5 Mega pixel Terasic CMOS camera to capture the RAW image from the driver’s face. The captured RAW image will be converted into a 36-bit RGB image format. The image processing is then separated into two sections which are the image segmentation and the image recognition. Before the image segmentation process, the image has been converted from RGB to grayscale using the averaging method while the grayscale to binary conversion uses the thresholding method. The image is localized and segmented out by tracking the edge of the binary image. The segmented image has also been recognized using the thresholding method. An HDL based test-bench is created for each sub-block for verification purposes in Modelsim of Mentor Graphic environment before implementation on the Cyclone IV FPGA can be performed. The Driver Drowsiness Detection System is designed to reduce the accident rate. Thus, the accuracy of the output is the demand of this system. The image taken in this research will be directly captured from the driver’s face but not from the CASIA database as was done previously. The accuracy of this design is at 96%. This current design methodology of the drowsiness detection system is less complex and it uses less area. This proposed research contributes to road safety, this system can work as a built-in system for each vehicle on the road. This system can also be applied to any other applications that requires the user to stay alert.

Contents

Chapter 1 Introduction

1.1 Objectives

1. To investigate the architecture of the image capture and image processing of the drowsiness detection system. The image processing includes image segmentation and image localization.

2. To model, design and simulate the drowsiness detection system using Verilog HDL in Modelsim environment.

3. To analyze, integrate and implement the designed Verilog Codes of the image capture and processing blocks of the drowsiness detection system using FPGA for functional verification purposes.

1.2 Problem Statement

The proposed work addresses the problems faced by the first conventional technique shown in Figure 1.1 by Yap (2015), Chuah (2015), and Lee (2015). In this research, the images will be captured directly from the human face but not from CASIA database. Also the design complexity of this research has been reduced. This research has a methodology that uses less logic element as compared to the conventional technique which uses 80% of the logic elements in the FPGA. The conventional block for the image capture and segmentation is as shown in Figure 1.1.

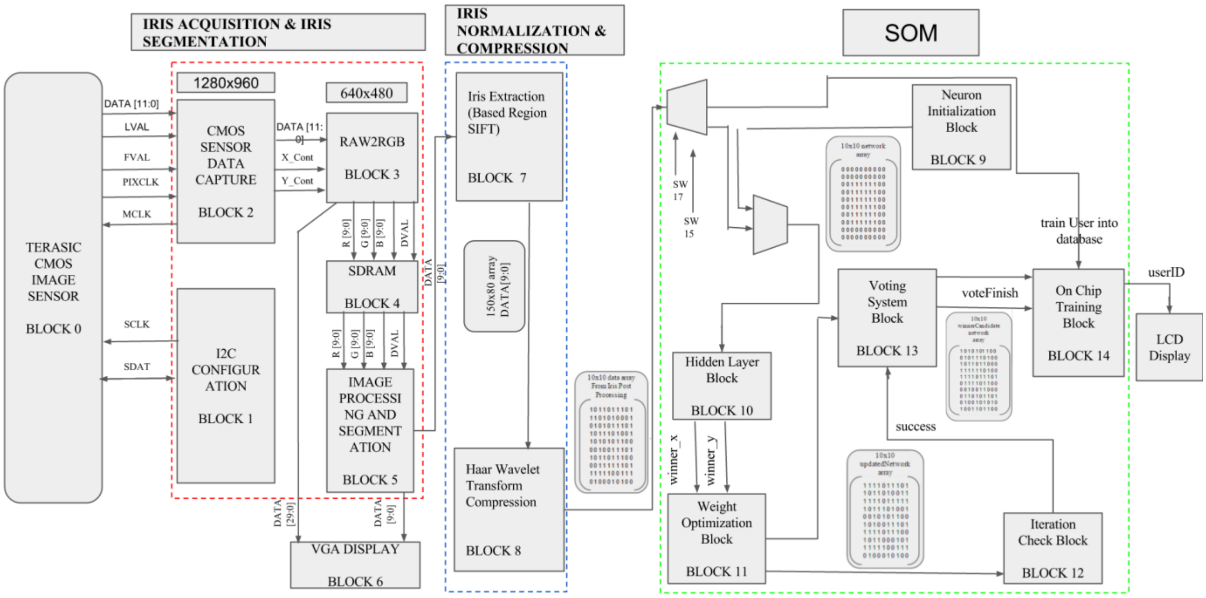


Figure 1.1 Conventional block for image capture and processing. (Yap, Chuah and Lee 2015)

From the previous research of Kuo and Hsu (2010) the drowsiness detection system detects both the mouth and the eyes. For the opening of the mouth, this might not be very convenient because the driver might be talking. Thus, in this currentwork, the detection portion has been narrowed down to the eyes only.

The other problem that has to be stated is from the past researcher Wang (2005) that implementing the Real-Time Driver Drowsiness tracking system. In this research, the system was built by using the Nios II soft-core embedded processor. Hence, the background operation of the hardware was not visible. Although this implementation has a lower complexity, the system has an overall correct judgment rate of only 62.5%.

1.3 Background

In today’s world, most of us are vehicle drivers. . For a driver that drives for hours, they will easily fall asleep or lose concentration while they are driving. This may cause serious accidents when the driver falls asleep. From the MIROS (Malaysia Institute of Road Safety Research) crash investigation and reconstruction annual statistical report from 2007 to 2010, the fatigue driving stands 9.9% which is the top 4 highest among all causes. Thus, the drowsiness detection system has been proposed to potentially avoid the stated problem.

From past research, the first fatigue detection system was developed by a car manufacturer named Volvo since 2007. But, in Malaysia, the drowsiness detection system in vehicles are still rarely found or implemented. The manufacturer in Malaysia such as Proton and Perodua, they both did not have a built in drowsiness detection system in today’s trend.

The capabilities of this proposed system is to detect whether the driver is sleepy or alert while driving. There are many ways to detect driver drowsiness, which is either to detect head motion, the eye closing intervals and the yawning action. Current research reduces the drowsiness detection complexity of the design by only detecting the eye closing intervals (Kuo and Hsu 2010). The reason not to detect the yawning action of the mouth is because the driver might be talking but not yawning when the mouth is opened. Hence, the results obtained will be inaccurate.

From almost all the past researches, there are only a few that are implementing the drowsiness detection system using hardware. They usually used software based methods to directly program the microprocessor which do not require comprehension of the low level operations. This work implements the drowsiness detection system using hardware based methods. The hardware used in this drowsiness detection system is FPGA, one of the main advantages of using FPGA is that the program will run concurrently rather than running sequentially.

The method of capturing the driver's face will be based on the design by Yap (2015). In which Image capture of the iris is performed while in this research the entire face is captured. There were some modifications made in order to achieve this objective. The method of localization in this design uses the thresholding method. Hence, this method can be applied to localize the eye of the driver. Furthermore, the thresholding method is also being used to recognize the driver drowsiness.

There are several techniques to perform image processing. Some of it has been listed down in Table 1 for the reference purposes. For image capture, in Yap, Chuah and Lee's (2015) work, the D5M CMOS camera has been used to capture the image and localize the image by using thresholding method to perform Iris Recognition System (IRS). This system also performs image normalization and compression. The idea from this research is to capture the image and perform iris recognition from the compressed image. The system achieves a recognition accuracy of 94% and a recognition speed of 1-2 seconds. There is also another method to perform a face detection system by Acasandrei (2005). This research uses the Viola Jones algorithm with integral images to perform face detection. The drowsiness detection system implemented by Flores (2008) uses visual information and Artificial intelligence to perform detection. This research has a minimum of 91.61% of correct detection rate. Alternatively, the Architecture of Wang (2005) uses the Nios II soft core embedded processor to perform Drowsiness Tracking.. This research has a correct judgment rate of 62.5% which is comparatively lower then this proposed architecture.

In summary, this work uses the D5M CMOS camera to capture the image of the driver. The face was localized using thresholding method. The captured image is then being processed and recognized using the thresholding method.

Table 1 Past researchers' on the related work of image processing and drowsiness detection system.

No.	Researcher (Year)	Architecture/System	Method applied	Results
1	Kassem *et al.* (2009)	Image Compression on FPGA using DCT	This paper presents a method to implement the DCT compression technique using Lee algorithm	The Maximum percentage error of 8% on the pixel range [0,256].
2	Wang *et al.* (2005)	Real-Time Driver Drowsiness Tracking System	This design adopts the Altera Nios II soft core embedded processor and combines image processing and mode identification functions	Reduced complexity using NIOS II with correct Judgment rate of 62.5% and 80ms response time
3	Acasandrei and Barriga (2012)	FPGA Implementation of an Embedded Face Detection System based on LEON3	Using Viola Jones algorithm with integral image to perform face detection.	The estimate static power consumption for the LEON3 core is 603mW
4	Mathur *et al.* (2012)	Image Compression Using Dft through Fast Fourier Transform Technique	Image compression method based on Fast Fourier Transformation.	The compression ratio is very good but the image quality degrades as the compression ratio is increased
5	Flores *et al.* (2008)	Real-Time Drowsiness Detection System for an Intelligent Vehicle	Advanced Driver Assistance System for automatic driver's drowsiness detection based on visual information and Artificial Intelligence.	It has a minimum of 91.61% of correct detection rate.
6	Yap and Chuah (2015)	Iris Normalization and Iris Feature Extraction implemented using FPGA	Compression of the iris region into a 10x10 array is based on Discrete Haar Wavelet Transformation (HWT).	achieves a recognition accuracy of 94% and a recognition speed of 1-2 seconds

Chapter 2 Literature Review

2.1 Conventional Technique for Drowsiness Detection System

In typical methodology of implementing the Drowsiness detection system, they consists of image capturing/acquisition, image processing, image segmentation and localization, image compression and image recognition.

2.1.1 Image Capturing/Acquisition

Image capturing is the main key to effect the output of result. An image sensor has to be used to capture the driver face. From the researcher Bharambe (2015), the image sensor used is simply a webcam. This research divides the video into frames and treats the frame as the image captured and fetch the frame whenever the face was detected. In research YC Kuo (2010) it also using the webcam to do the image acquisition.

2.1.2 Image Processing

The typical image processing topology used was Haar Wavelet and Viola Jones with the aids of OpenCV. The Viola Jones image processing method has been used byAcasandrei (2012) to do image processing. This method are able to perform face detection directly. While for the Haar Wavelet has been used by Chuah (2015) to do image processing together with the image compression.

Besides using the methodology above, the researcher YC Kuo (2010) perform image processing by converting the image to RGB, Grayscale and Binary to localize the face of the driver. The methodology used from Yap (2015) also using the same methodology which implement in the FPGA.

2.1.3 Image Recognition

The typical recognition system was using Artificial Intelligence such as Fuzzy Logic, ANN Neural Network, and etc. The researcher YC Kuo (2010) performed image recognition by using Fuzzy Logic. While the researcher Lee (2015) was using Self Organized Map (SOM) algorithm to perform image recognition.

Chapter 3 Methodology

Figure 1 shows the design flow chart of the image capture and compression of the Drowsiness Detection System. The description for each block will be discussed in this section.

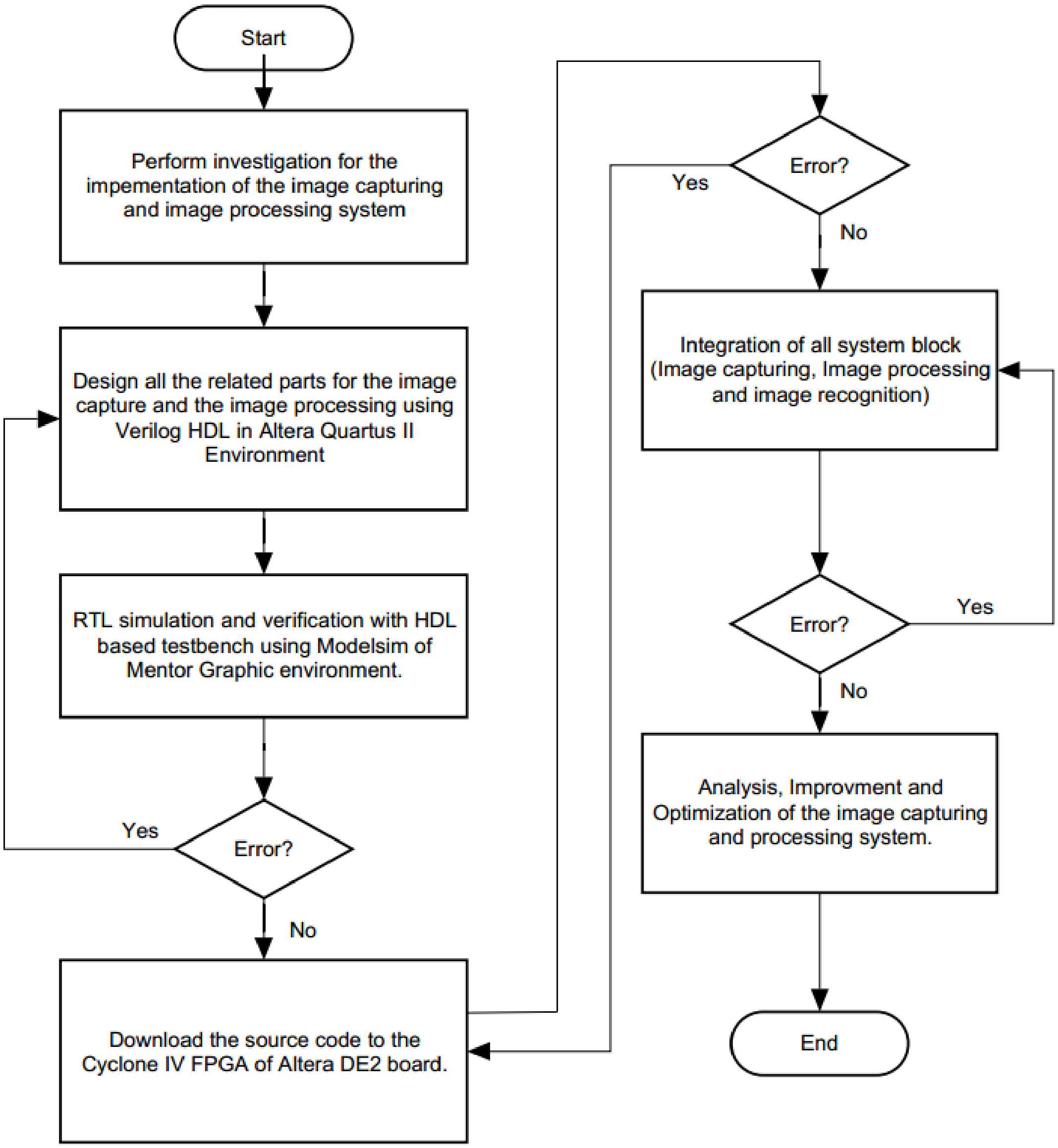

Start
Perform investigation for the impementation of the image capturing and image processing system
Design all the related parts for the image capture and the image processing using Verilog HDL in Altera Quartus II Environment
RTL simulation and verification with HDL based testbench using Modelsim of Mentor Graphic environment.
Yes
Error?
No
Download the source code to the Cyclone IV FPGA of Altera DE2 board.
Yes
Error?
No
Integration of all system block (Image capturing, Image processing and image recognition)
Error?
Yes
No
Analysis, Improvment and Optimization of the image capturing and processing system.
End

Figure 1 Design Flow Chart for the Drowsiness Detection System.

The first step of the design is to investigate through literature review regarding all sub-blocks related to the drowsiness detection system. After all sub-blocks have been investigated, these blocks will be modelled and designed by using Verilog HDL in Altera Quartus II. Each sub-block will be simulated by using Verilog HDL based testbench in Modelsim of Mentor Graphics environment. The result can be obtained from the waveform or console output produced by Modelsim. If there were any errors in the design, modification has to be made from the design source code. Eventually, the result will be verified with the hardware by downloading the bit stream on to the Cyclone IV FPGA of Altera's DE2 board.

If all the aforementioned processes doesn't incur any error, the hardware is now ready to be integrated with the recognition block. Error might occur during the integration process, so if there were any error during this stage, some modification has to be made on one of the erroneous designs.

The architecture of the Drowsiness Detection System is as shown in Figure 2. It contains five main sections which are the image capture, image processing, image segmentation and localization, image recognition and image display. The peripheral of the FPGA used in this system are the Terasic CMOS sensor, SDRAM, VGA and the audio codec.

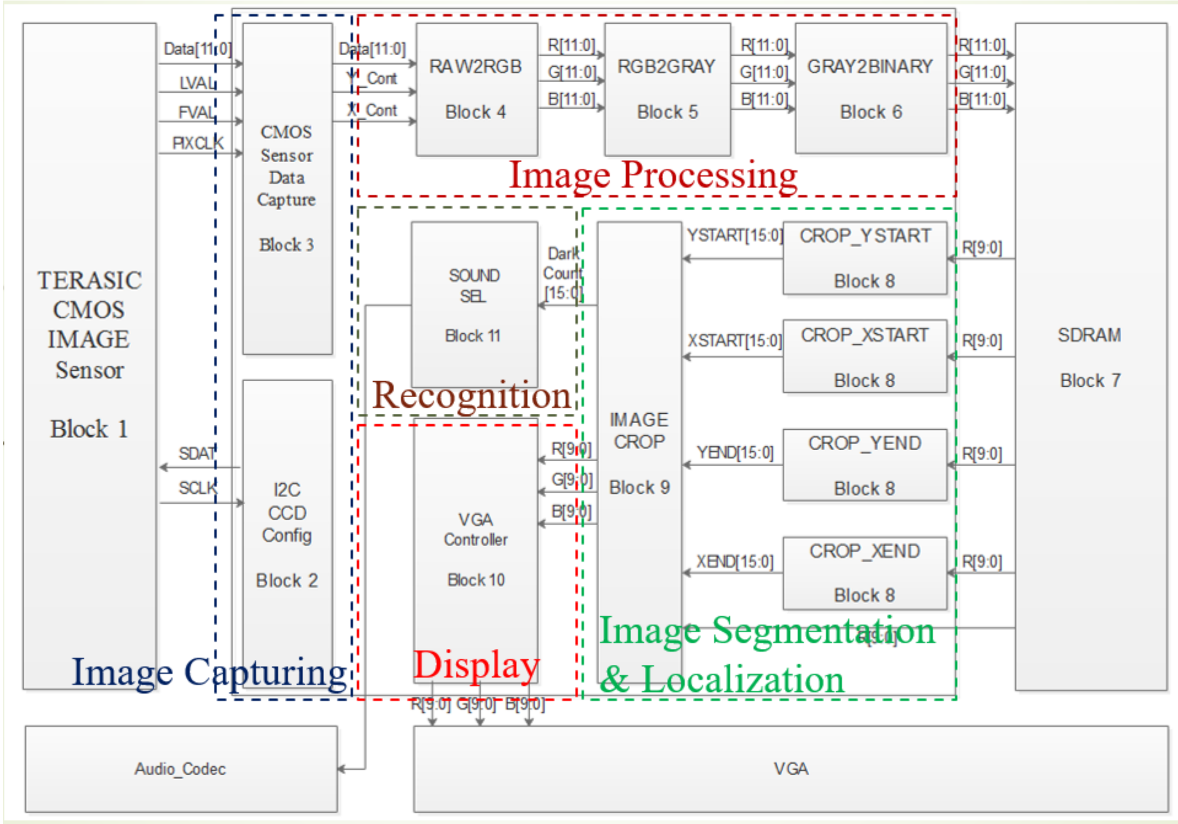


Figure 2 The Architecture for the Drowsiness Detection System.

The system starts with the Terasic CMOS Sensor (Block 1) will send appropriate signals to the second block when it receives a command by the I2C (Block 2). The CMOS Sensor Data Capture (Block 3) receives signal to begin image data capture.

In the image processing part, The captured 1280x960 12-bit RAW image data is then converted to 640x480 36-bit RGB format image data (12-bit for each R, G and B data) by the RAW2RGB (Block 4). Next, the 36-bit RGB image has been converted to 36-bit grayscale image by RGB2GRAY (Block 5). Lastly, the image processing part ends with the conversion of the 36-bit grayscale image to 36-bit binary image by the GRAY2BINARY block (Block 6). The 36-bit binary image has been truncated into a 30-bit binary image (10-bit for each R, G and B data) before being stored into the SDRAM.

During image segmentation and localization, the average of 30-bit binary image data will be read out from the SDRAM. Thus, the 10-bit image data will be passed on to the CROP_YSTART, CROP_XSTART, CROP_YEND and CROP_XEND blocks (Block 8). These blocks will give the coordinates of the image edges to IMAGE_CROP (Block 9) to segment out the image depending on the specified coordinates.

The segmented image will be displayed to the VGA display in which the parameter of the VGA display will be configured by the VGA controller (Block 10).During recognition, the black pixel count will be passed to the SOUND_SEL (Block 11) to determine whether or not to sound the audio codec.

The contribution of this work will be at the image segmentation, image localization as well as the recognition process.

3.1 CMOS Image Sensor

The camera TRDB-D5M as shown in Figure 3.1 is a 5 Megapixel camera with 2752-column and 2004-rows. The Camera is connected to the GPIO ports of the Altera DE2-115 FPGA Board. The frame rate of the D5M camera can go up to 150fps when being displayed in the VGA monitor. The connection of the camera is as shown in Figure 3.2.

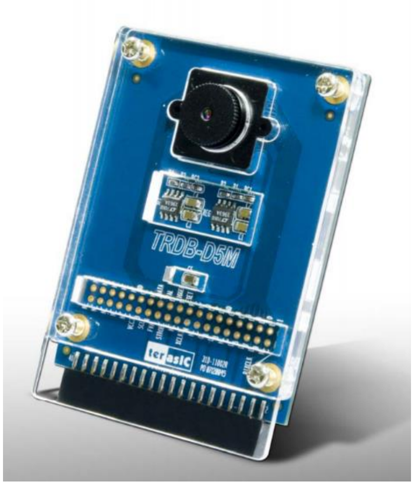


Figure 3.1 TRDB-D5M.

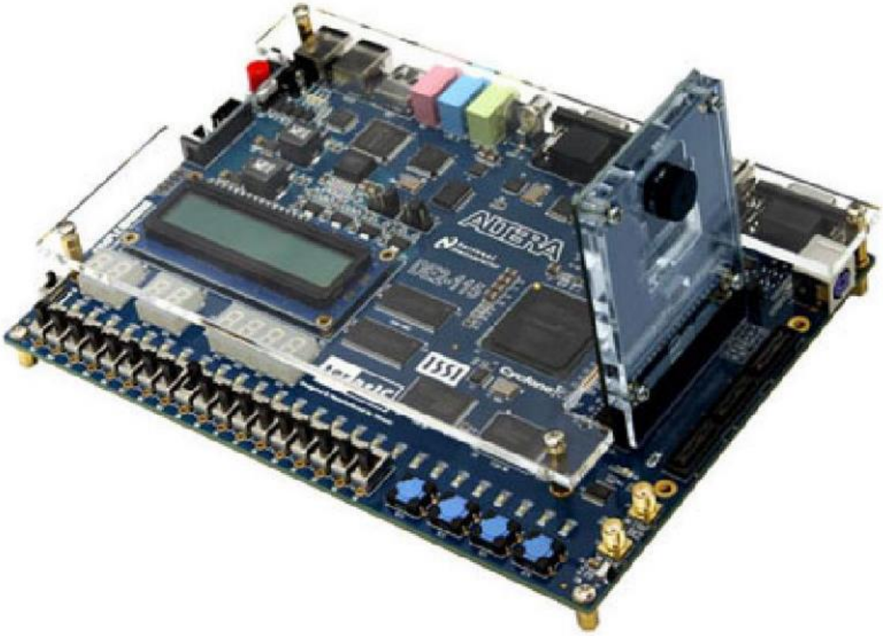
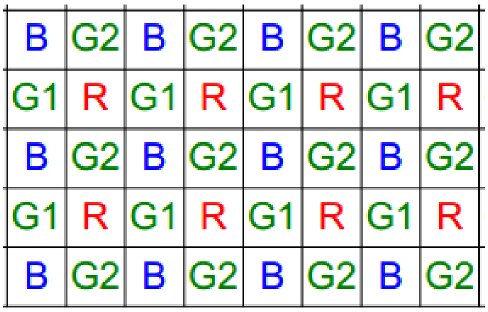


Figure 3.2 The Connection Setup for DE2-115.

The Pixels are outputted in a Bayer pattern format that consists of four colours which are the Green1, Green2, Red and Blue. The Bayer pattern is arranged as in Figure 3.3 and the boundaries for the pixel array is as shown in Figure 3.4.



B	G2	B	G2	B	G2	B	G2
G1	R	G1	R	G1	R	G1	R
B	G2	B	G2	B	G2	B	G2
G1	R	G1	R	G1	R	G1	R
B	G2	B	G2	B	G2	B	G2

Figure 3.3 Bayer arrangement of the Image sensor.

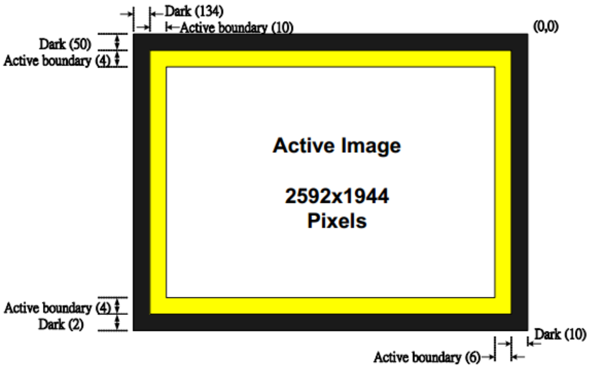


Figure 3.4 Pixel Array Description.

The configuration of the camera is set by the I2C Configuration Block (Block 2).The I2C will send instructions or commands to the camera through 32-bit SDATA. The image will form based on the configuration of the I2C. Each pixel of the Image is represented by a 12-bit data. The 12-bit data and Line_Valid (LVAL), Frame_Valid (FVAL) and PIXCLK signals will then output to the CMOS sensor data Capture (Block 3). The communication between the I2C and the camera is illustrated in the block diagram shown in Figure 3.5 below.

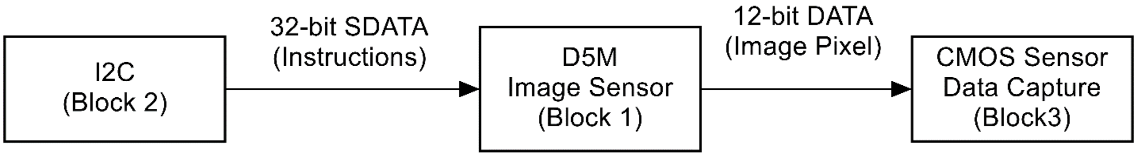


Figure 3.5 Communication between I2C and D5M camera.

3.2 I2C Controller and I2C Configuration

The I2C controller communicates with the D5M camera through the serial clock (SCLK) and the serial data (SDATA). Data is transferred in and out of the D5M through the SDATA line. In the process of transferring data though SDATA, it consists of five transmission codes. The transmission codes are the start bit, slave device 8-bit address, acknowledgement bit, 8-bit message and stop bit.

The sequence of the serial communication between the I2C and the D5M camera begins with the master sending a start bit. After that, the master will send the 8-bit address of the slave device. The last address bit represent whether to read or to write (‘1’ is to read and ‘0’ is to write). In this research, only writing is performed, so the last bit of the slave device address will always be ‘0’. The slave device will send an acknowledge bit to the master to inform the master that the transmission is successful.

For the writing sequence, the master will send an 8-bit register address of the selected slave device and wait for another acknowledgement bit. After selecting the slave device and its registers, the 16-bit data is ready to be sent. Note that the D5M can only transfer 8-bits of data at a time, thus two 8-bit transfers are needed to write to one register. The master stops writing by sending the stop bit. The flow of the serial communication is as shown in Figure 3.6. The HDL code representation for the serial communication is as shown in Figures 3.7 to 3.12.

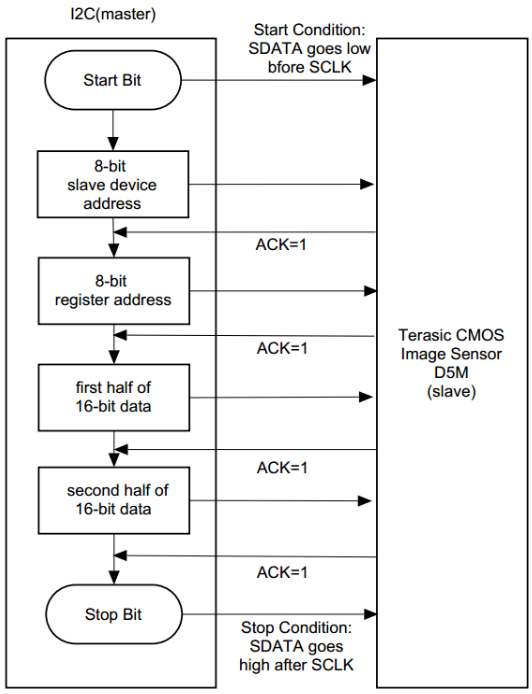
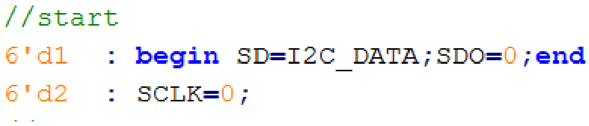
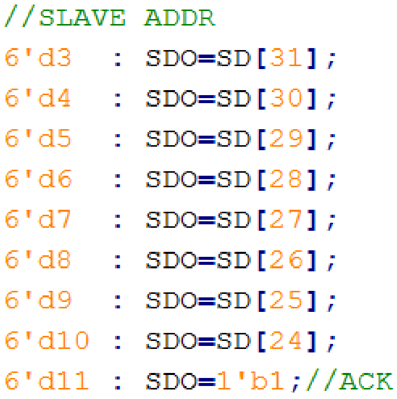

I2C(master)
Start Condition: SDATA goes low bfore SCLK
Start Bit
8-bit slave device address
ACK=1
8-bit register address
ACK=1
first half of 16-bit data
ACK=1
second half of 16-bit data
ACK=1
Stop Bit
Stop Condition: SDATA goes high after SCLK
Terasic CMOS Image Sensor D5M (slave)

Figure 3.6 Flow chart of serial communication.



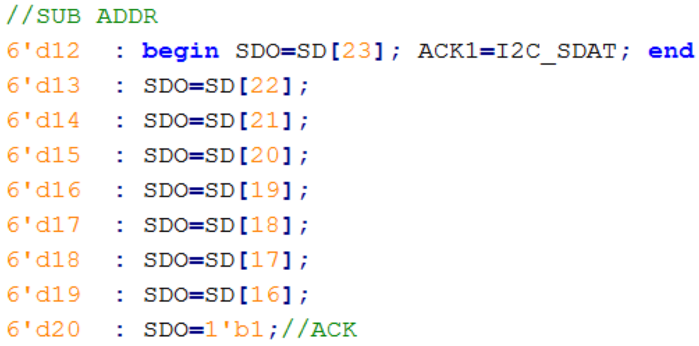
```
//start
6'd1   : begin SD=I2C_DATA;SDO=0;end
6'd2   : SCLK=0;
```

Figure 3.7 Start bit condition (SDATA goes low first then SCLK).



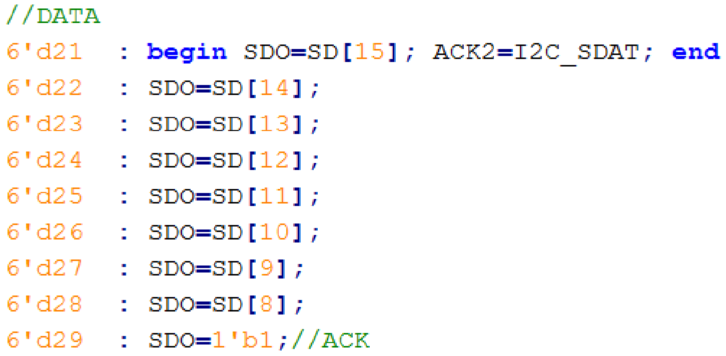
```
//SLAVE ADDR
6'd3   : SDO=SD[31];
6'd4   : SDO=SD[30];
6'd5   : SDO=SD[29];
6'd6   : SDO=SD[28];
6'd7   : SDO=SD[27];
6'd8   : SDO=SD[26];
6'd9   : SDO=SD[25];
6'd10  : SDO=SD[24];
6'd11  : SDO=1'b1;//ACK
```

Figure 3.8 Sending of the slave device address.



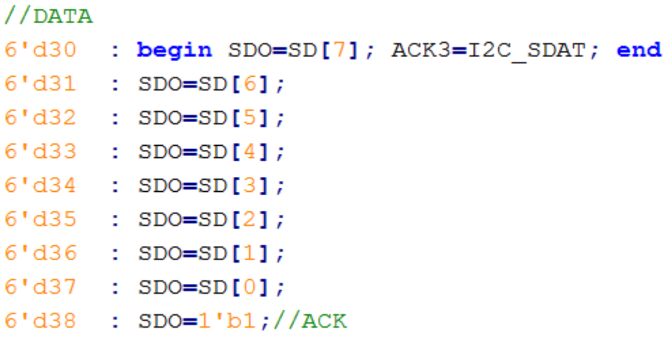
```
//SUB ADDR
6'd12   : begin SDO=SD[23]; ACK1=I2C_SDAT; end
6'd13   : SDO=SD[22];
6'd14   : SDO=SD[21];
6'd15   : SDO=SD[20];
6'd16   : SDO=SD[19];
6'd17   : SDO=SD[18];
6'd18   : SDO=SD[17];
6'd19   : SDO=SD[16];
6'd20   : SDO=1'b1;//ACK
```

Figure 3.9 Sending of register address.



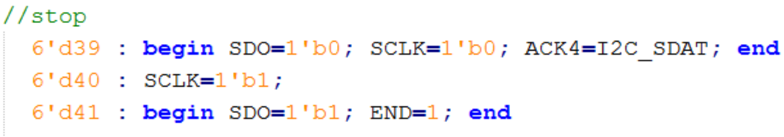
```
//DATA
6'd21   : begin SDO=SD[15]; ACK2=I2C_SDAT; end
6'd22   : SDO=SD[14];
6'd23   : SDO=SD[13];
6'd24   : SDO=SD[12];
6'd25   : SDO=SD[11];
6'd26   : SDO=SD[10];
6'd27   : SDO=SD[9];
6'd28   : SDO=SD[8];
6'd29   : SDO=1'b1;//ACK
```

Figure 3.10 Sending first half of the 16-bit data.



```
//DATA
6'd30   : begin SDO=SD[7]; ACK3=I2C_SDAT; end
6'd31   : SDO=SD[6];
6'd32   : SDO=SD[5];
6'd33   : SDO=SD[4];
6'd34   : SDO=SD[3];
6'd35   : SDO=SD[2];
6'd36   : SDO=SD[1];
6'd37   : SDO=SD[0];
6'd38   : SDO=1'b1;//ACK
```

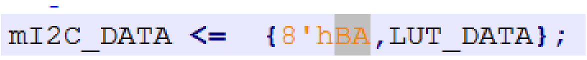
Figure 3.11 Sending second half of the 16-bit data.



```
//stop
  6'd39 : begin SDO=1'b0; SCLK=1'b0; ACK4=I2C_SDAT; end
  6'd40 : SCLK=1'b1;
  6'd41 : begin SDO=1'b1; END=1; end
```

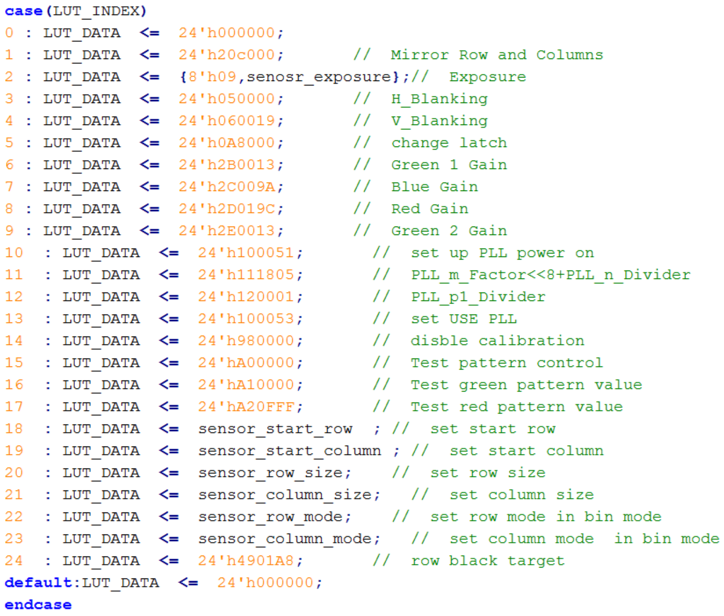
Figure 3.12 Stop bit condition (SCLK goes high first then SDATA).

Note that when sending the address and the data, the last bit is the acknowledgement bit which has been constantly set to '1'. The configuration of the I2C is as shown in Figures 3.13 to 3.14 shown below.



```
mI2C_DATA <=  {8'hBA,LUT_DATA};
```

Figure 3.13 Setting of I2C slave address.



```
case(LUT_INDEX)
0 : LUT_DATA  <=  24'h000000;
1 : LUT_DATA  <=  24'h20c000;       //  Mirror Row and Columns
2 : LUT_DATA  <=  {8'h09,senosr_exposure};//  Exposure
3 : LUT_DATA  <=  24'h050000;       //  H_Blanking
4 : LUT_DATA  <=  24'h060019;       //  V_Blanking
5 : LUT_DATA  <=  24'h0A8000;       //  change latch
6 : LUT_DATA  <=  24'h2B0013;       //  Green 1 Gain
7 : LUT_DATA  <=  24'h2C009A;       //  Blue Gain
8 : LUT_DATA  <=  24'h2D019C;       //  Red Gain
9 : LUT_DATA  <=  24'h2E0013;       //  Green 2 Gain
10  : LUT_DATA  <=  24'h100051;       //  set up PLL power on
11  : LUT_DATA  <=  24'h111805;       //  PLL_m_Factor<<8+PLL_n_Divider
12  : LUT_DATA  <=  24'h120001;       //  PLL_p1_Divider
13  : LUT_DATA  <=  24'h100053;       //  set USE PLL
14  : LUT_DATA  <=  24'h980000;       //  disble calibration
15  : LUT_DATA  <=  24'hA00000;       //  Test pattern control
16  : LUT_DATA  <=  24'hA10000;       //  Test green pattern value
17  : LUT_DATA  <=  24'hA20FFF;       //  Test red pattern value
18  : LUT_DATA  <=  sensor_start_row  ; //  set start row
19  : LUT_DATA  <=  sensor_start_column ; //  set start column
20  : LUT_DATA  <=  sensor_row_size;    //  set row size
21  : LUT_DATA  <=  sensor_column_size;   //  set column size
22  : LUT_DATA  <=  sensor_row_mode;    //  set row mode in bin mode
23  : LUT_DATA  <=  sensor_column_mode;   //  set column mode  in bin mode
24  : LUT_DATA  <=  24'h4901A8;       //  row black target
default:LUT_DATA  <=  24'h000000;
endcase
```

Figure 3.14 I2C configuration.

The value shown in Figure 3.14 consists of the slave register address and the data to be sent. While Figure 3.13 shows the slave device address that has been constantly set to 0xBA which is the address of the D5M. The exposure time, clock frequency, I2C frequency, and the LUT size value can be modified in the I2C_CCD_Config.v file. Also the value in Figure 3.14 can be modified to get a suitable image to perform image processing.

3.3 CMOS Sensor Data Capture

The CMOS Sensor Data Capture will receive signals from the CMOS camera. The signals include the 12-bit pixel data, Line_Valid (LVAL), Frame_Valid (FVAL) and PIXCLK. For the camera, rows are represented by **Frames** while columns are represented by **Lines**. The value of FVAL and LVAL will be set if the image at the specific columns and rows are respectively valid. The pixel output timing is as shown in Figure 3.15. The valid image to be captured is shown in Figure 3.16. However, the dark row and dark column have also to be taken into consideration. Thus, some of the spacing are needed for both dark row and dark column. The spacing are as shown in Figure 3.17.

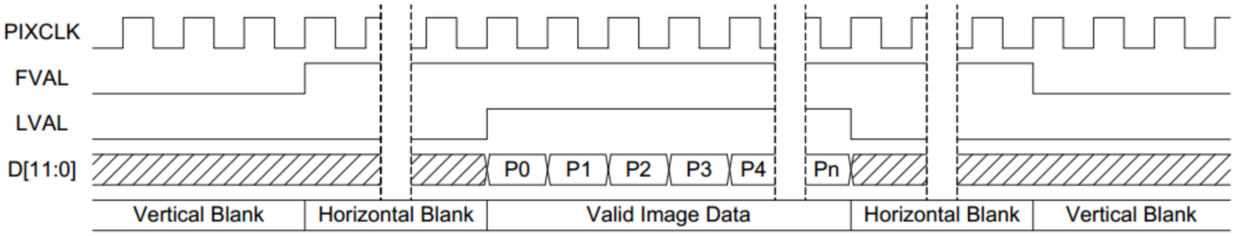


Figure 3.15 Pixel output timing.

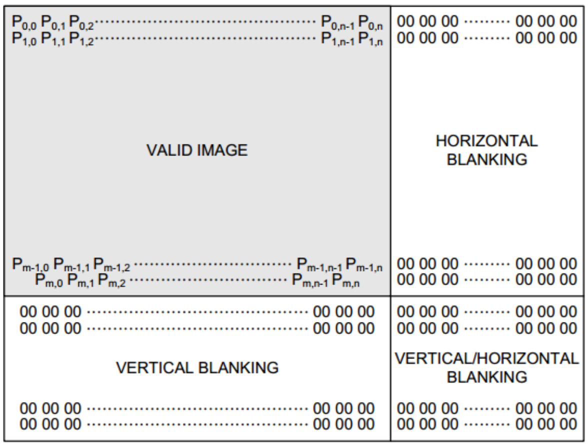

$P_{0,0}$ $P_{0,1}$ $P_{0,2}$ .... $P_{0,n-1}$ $P_{0,n}$
$P_{1,0}$ $P_{1,1}$ $P_{1,2}$ .... $P_{1,n-1}$ $P_{1,n}$
VALID IMAGE
$P_{m-1,0}$ $P_{m-1,1}$ $P_{m-1,2}$ .... $P_{m-1,n-1}$ $P_{m-1,n}$
$P_{m,0}$ $P_{m,1}$ $P_{m,2}$ .... $P_{m,n-1}$ $P_{m,n}$
00 00 00 .... 00 00 00
HORIZONTAL BLANKING
VERTICAL BLANKING
VERTICAL/HORIZONTAL BLANKING

Figure 3.16 The location of valid image captured by the camera.

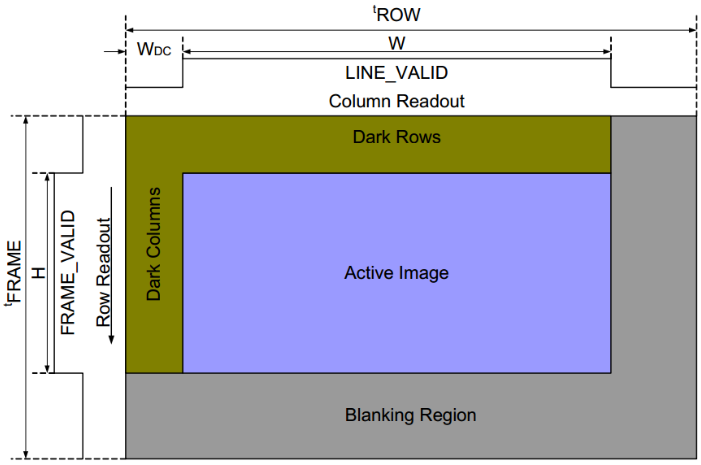
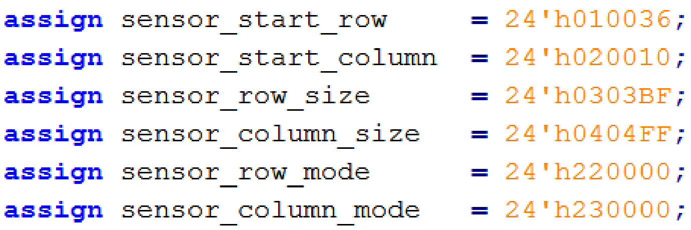


Figure 3.17 Location of Active Image captured by the camera.

From the I2C Configuration, the starting row and the starting column has been set to 54 and 16 respectively. The starting row and column can be referred as the boundaries of the pixels array which was already discussed in the previous section. The starting row and the starting column configuration is as shown in Figure 3.18. With this configuration, the read out image excludes the dark rows and columns. Hence, the first FVAL and the first LVAL will be at 54 and 16 respectively.



```
assign sensor_start_row     = 24'h010036;
assign sensor_start_column  = 24'h020010;
assign sensor_row_size      = 24'h0303BF;
assign sensor_column_size   = 24'h0404FF;
assign sensor_row_mode      = 24'h220000;
assign sensor_column_mode   = 24'h230000;
```

Figure 3.18 Row and column of the I2C configuration.

Each of the 12-bit pixel data will have its own X position (X_Cont) and Y position (Y_Cont). These variables indicate the position of each pixel data. The origin is at the top right of the camera. However, the dark row and dark column has been excluded. Thus, when X_Cont = 0 and Y_Cont = 0, the pixel is located at pixel (16, 54). The flow chart of the CMOS Sensor Data Capture is as shown in Figure 3.19.

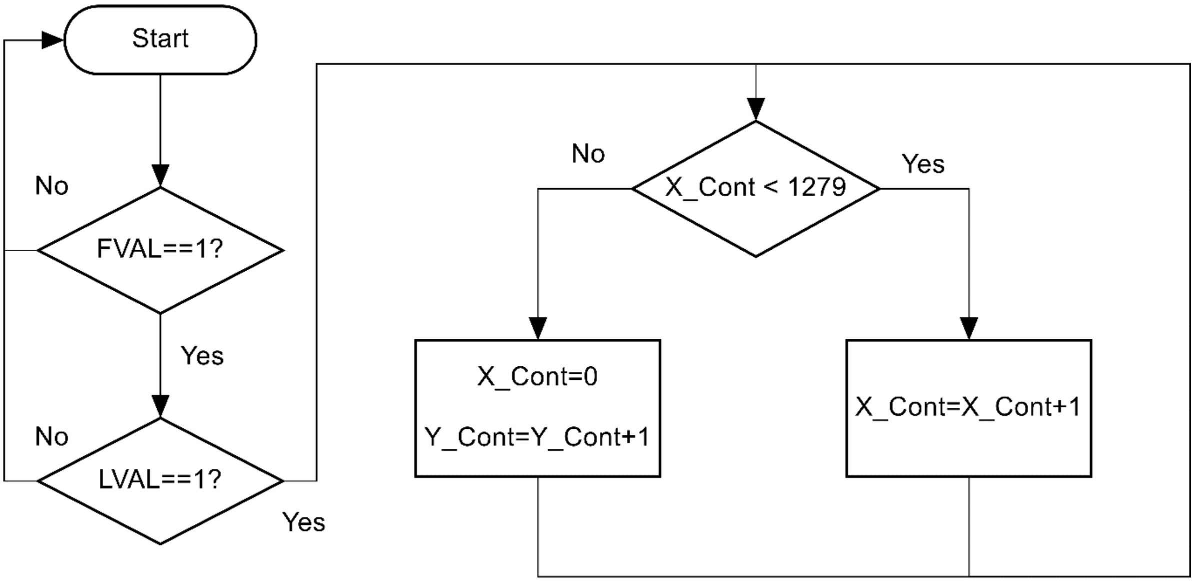
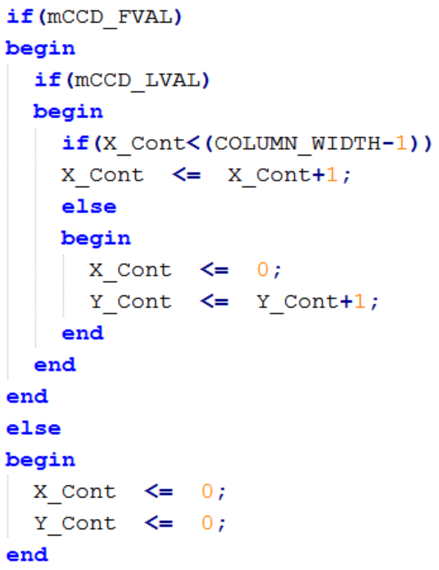


Figure 3.19 Flow Chart of the CMOS Sensor Data Capture.

Note that the data capturing flow shown in Figure 3.19 will run forever, the X_Cont will be reset but Y_Cont will increase until the maximum value of the register. The Verilog HDL code representation is as shown in Figure 3.20.



```
if(mCCD_FVAL)
begin
  if(mCCD_LVAL)
  begin
    if(X_Cont<(COLUMN_WIDTH-1))
    X_Cont  <=  X_Cont+1;
    else
    begin
      X_Cont  <=  0;
      Y_Cont  <=  Y_Cont+1;
    end
  end
end
else
begin
  X_Cont  <=  0;
  Y_Cont  <=  0;
end
```

Figure 3.20 Verilog HDL code representation of the CCD Capture.

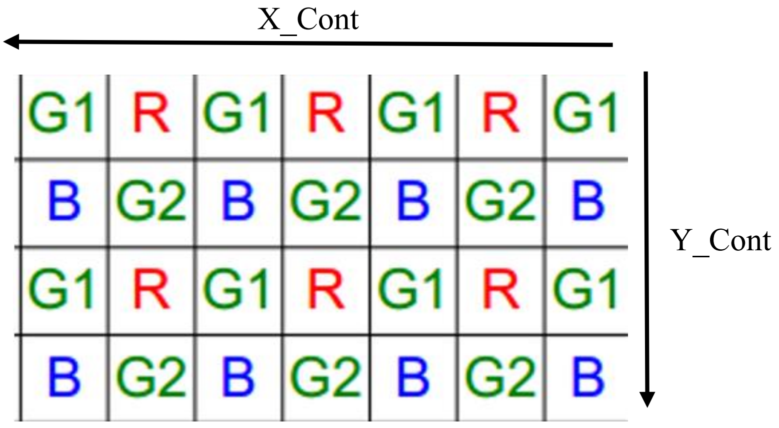
3.4 RAW2RGB

This module converts a 1280x960 12-bit RAW image data to a 640x480 36-bit RGB format image data (12-bit for each R, G and B data). This block is provided by Terasic. This block consists of a Mega function shift register that stores the image data. The data representation of the registers is as shown in Table 3.1. The R, G and B values are decided by the register value. The Bayer pattern captured by the previous block is as shown in Figure 3.21.

To produce the individual R, G and B data values from the RAW data, four scenarios have to be considered i.e. When both X_Cont and Y_Cont are even, X_Cont is odd and Y_Cont is even, X_Cont is even and Y_Cont is odd, both X_Cont and Y_Cont are odd. The four scenarios are as shown in Table 3.2 below.

Table 3.1 Shift register representation.

mDatad0	mData0
mDatad1	mData1



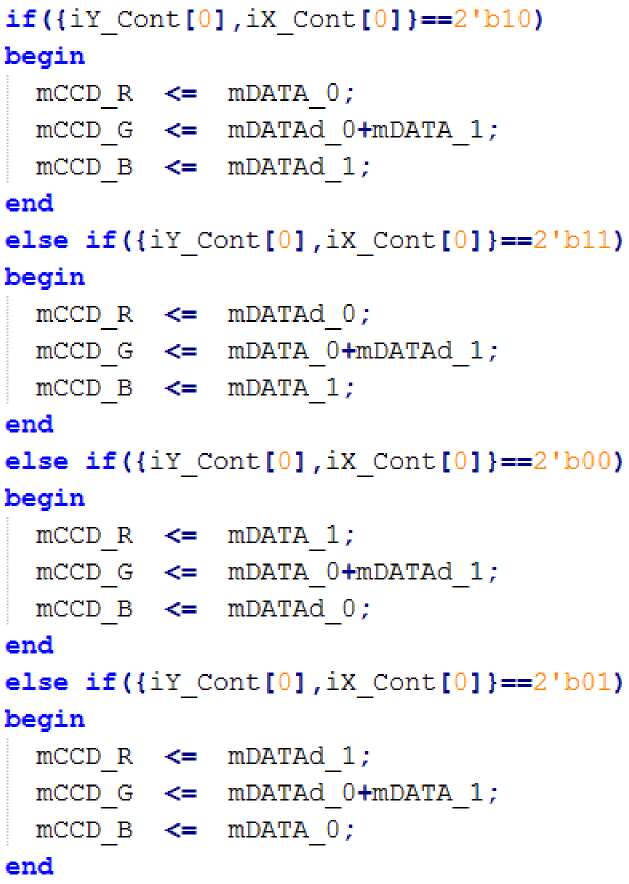
G1	R	G1	R	G1	R	G1
B	G2	B	G2	B	G2	B
G1	R	G1	R	G1	R	G1
B	G2	B	G2	B	G2	B

Figure 3.21 Bayer pattern captured by the CMOS Sensor Data Capture (Block 3).

Table 3.2 The four scenario to produce the R, G and B value.

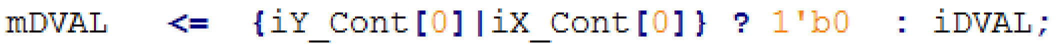
X_Cont[0]=0(even), Y_Cont[0]=0(even)		X_Cont[0]=1(odd), Y_Cont[0]=0(even)		X_Cont[0]=0(even), Y_Cont[0]=1(odd)		X_Cont[0]=1(odd), Y_Cont[0]=1(odd)	
R	G1	G1	R	G2	B	B	G2
G2	B	B	G2	R	G1	G1	R

The Verilog HDL code representation for the above scenarios is as shown in Figure 3.22 below. The R, G and B values has been stored. The RGB data will only been output to the following block only if both X_Cont and Y_Cont is even. Otherwise, the output is considered invalid for the following block. Hence, the size will be reduced from 1280x960 to 640x480. This HDL code representation of the operation is as shown in Figure 3.23.



```
if({iY_Cont[0],iX_Cont[0]}==2'b10)
begin
   mCCD_R  <=  mDATA_0;
   mCCD_G  <=  mDATAd_0+mDATA_1;
   mCCD_B  <=  mDATAd_1;
end
else if({iY_Cont[0],iX_Cont[0]}==2'b11)
begin
   mCCD_R  <=  mDATAd_0;
   mCCD_G  <=  mDATA_0+mDATAd_1;
   mCCD_B  <=  mDATA_1;
end
else if({iY_Cont[0],iX_Cont[0]}==2'b00)
begin
   mCCD_R  <=  mDATA_1;
   mCCD_G  <=  mDATA_0+mDATAd_1;
   mCCD_B  <=  mDATAd_0;
end
else if({iY_Cont[0],iX_Cont[0]}==2'b01)
begin
   mCCD_R  <=  mDATAd_1;
   mCCD_G  <=  mDATAd_0+mDATA_1;
   mCCD_B  <=  mDATA_0;
end
```

Figure 3.22 Verilog HDL code representation of the four scenarios.



```
mDVAL   <=  {iY_Cont[0]|iX_Cont[0]} ? 1'b0  : iDVAL;
```

Figure 3.23 Verilog HDL code representation of the output condition.

3.5 RGB2GRAY

This module converts a 640x480 36-bit RGB image data to a 640x480 12-bit grayscale image data. In the VGA display, the maximum value is read as white and the minimum value is read as black. The grayscale representation of an 8-bit data image is as shown in Figure 3.24 below.

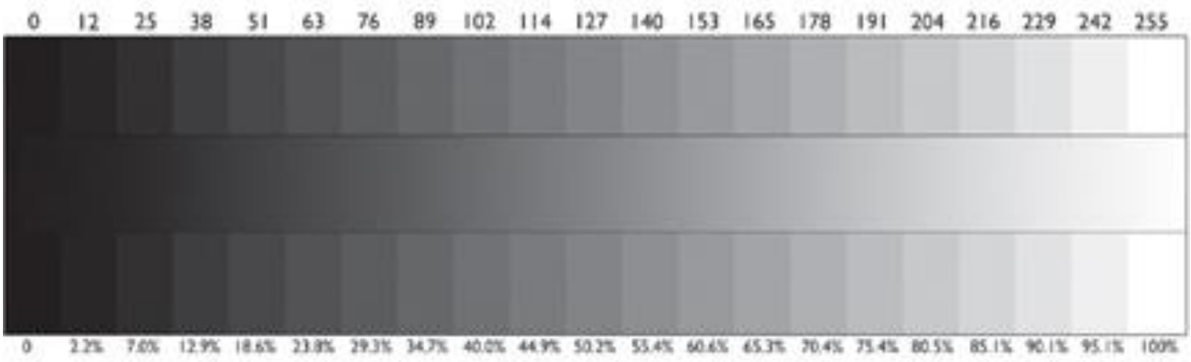
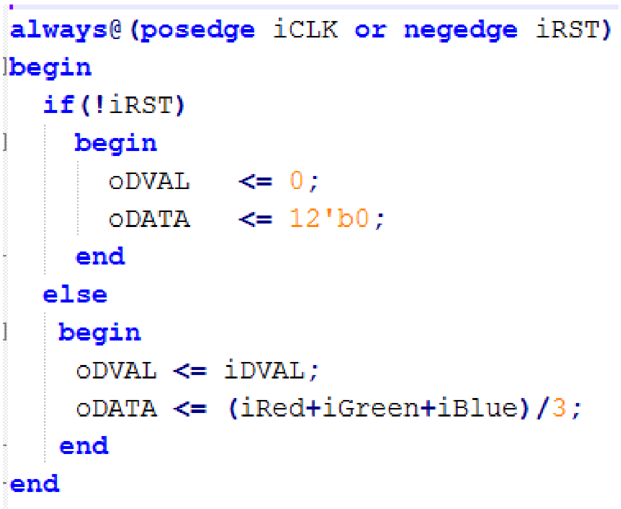


Figure 3.24 Grayscale representation of an 8-bit data image.

From the grayscale interval value shown in the above Figure 3.24, the interval of the grayscale depends on a percentage. Hence in this module, it is able to perform conversion although the data input is more than 8-bit. In current researches, the image data is in 12-bit which gives a maximum value of 4096. Hence, the value of 0 shows white while the value of 4096 shows black.

The averaging method (John, 2009) has been used to obtain the grayscale image. The equation of the averaging method is (R + G + B)/3. The averaging method was used because it is less complex. The grayscale image does not need high complexity to be implemented because the image will be converted to binary in the following step.

The Verilog HDL code representation for the conversion process is as shown in Figure 3.25 below.



```
always@(posedge iCLK or negedge iRST)
begin
  if(!iRST)
    begin
      oDVAL   <= 0;
      oDATA   <= 12'b0;
    end
  else
   begin
    oDVAL <= iDVAL;
    oDATA <= (iRed+iGreen+iBlue)/3;
   end
end
```

Figure 3.25 The Verilog HDL code representation of the grayscale conversion.

From the Figure 3.25 above, the output data has been obtained the grayscale equation. The grayscale data will be outputted every clock cycle. Which means the image will be formed pixel by pixel in one clock cycle. This process is not visible by human because the clock frequency is faster than the human eye capturing frequency.

3.6 GRAY2BINARY

This module converts a 640x480 12-bit grayscale image data to a 640x480 12-bit binary image data. From the previous module, the maximum value for a 12-bit (4096) is white while the minimum value for a 12-bit (0) is black. This module will convert the data to either black (0) or white (4096).

The thresholding method has been used to convert the grayscale image to a binary image in which if the input grayscale image data is higher than the threshold, the output data will be 4096 whereas if the input grayscale image data is lower than the threshold, the output data will be 0. The illustration of the binary output is as shown in Figure 3.26 below. From the Figure 3.26, if the threshold increases closer to the maximum value. There will be more grayscale input data that will be considered as a black output. The HDL code representation of the thresholding method is as shown in Figure 3.27. Note that the threshold is a hard-coded value to the hardware.

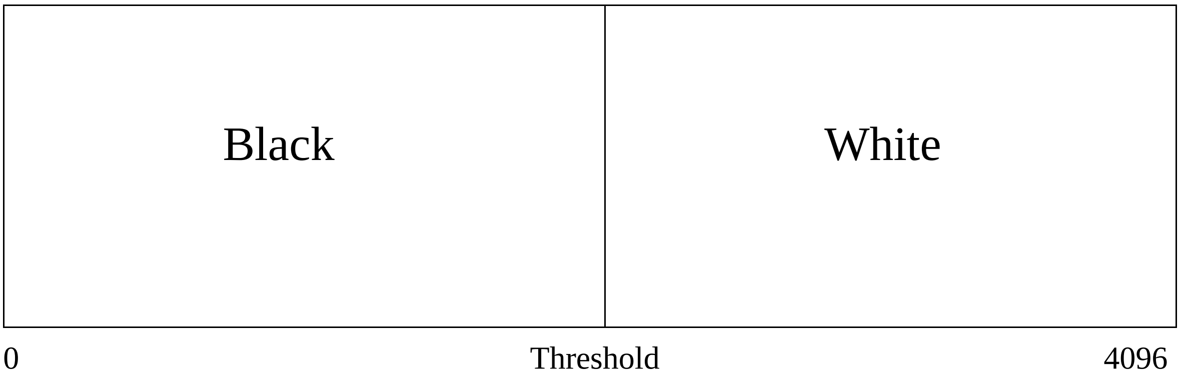
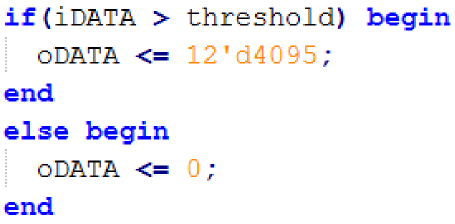


Figure 3.26 Illustration of the output of binary image.



```
if(iDATA > threshold) begin
  oDATA <= 12'd4095;
end
else begin
  oDATA <= 0;
end
```

Figure 3.27 Verilog HDL code representation of the thresholding method.

3.7 SDRAM

The SDRAM is a memory block storing the image that is captured by the CMOS Sensor Data Capture block (Block 3). The SDRAM is one of the on-board peripheral inside the DE2-115 FPGA. It has a maximum of 128MB of memory which was implemented by two 64MB of SDRAM. Each device consists of separate 16-bit data lines connected to the FPGA. This means that it has a maximum data size of 32-bit. The connection of the FPGA with the SDRAM is as shown in Figure 3.28 below. The Verilog HDL representation of the SDRAM configuration is as shown in Figure 3.29.

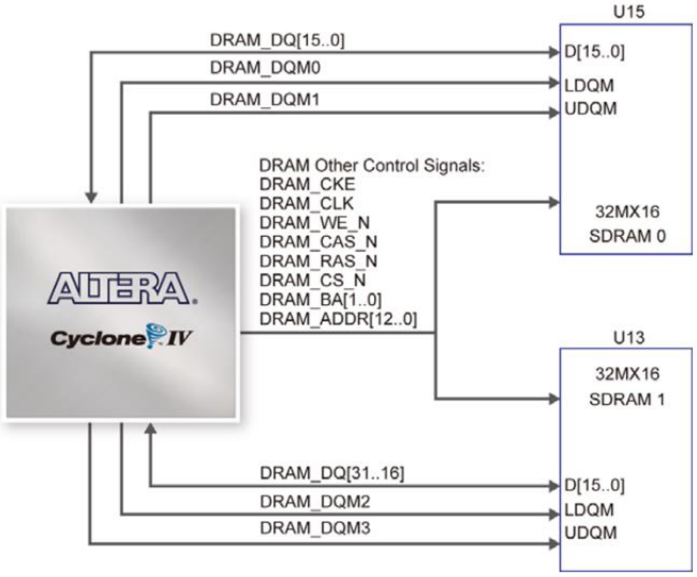
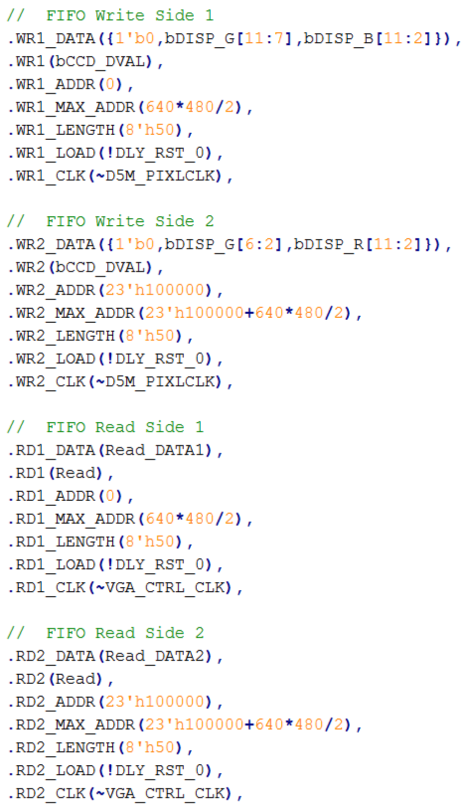


Figure 3.28 Connections between FPGA and the SDRAM.



```
//  FIFO Write Side 1
.WR1_DATA({1'b0,bDISP_G[11:7],bDISP_B[11:2]}),
.WR1(bCCD_DVAL),
.WR1_ADDR(0),
.WR1_MAX_ADDR(640*480/2),
.WR1_LENGTH(8'h50),
.WR1_LOAD(!DLY_RST_0),
.WR1_CLK(~D5M_PIXLCLK),

//  FIFO Write Side 2
.WR2_DATA({1'b0,bDISP_G[6:2],bDISP_R[11:2]}),
.WR2(bCCD_DVAL),
.WR2_ADDR(23'h100000),
.WR2_MAX_ADDR(23'h100000+640*480/2),
.WR2_LENGTH(8'h50),
.WR2_LOAD(!DLY_RST_0),
.WR2_CLK(~D5M_PIXLCLK),

//  FIFO Read Side 1
.RD1_DATA(Read_DATA1),
.RD1(Read),
.RD1_ADDR(0),
.RD1_MAX_ADDR(640*480/2),
.RD1_LENGTH(8'h50),
.RD1_LOAD(!DLY_RST_0),
.RD1_CLK(~VGA_CTRL_CLK),

//  FIFO Read Side 2
.RD2_DATA(Read_DATA2),
.RD2(Read),
.RD2_ADDR(23'h100000),
.RD2_MAX_ADDR(23'h100000+640*480/2),
.RD2_LENGTH(8'h50),
.RD2_LOAD(!DLY_RST_0),
.RD2_CLK(~VGA_CTRL_CLK),
```

Figure 3.29 Verilog HDL representation of the SDRAM configuration.

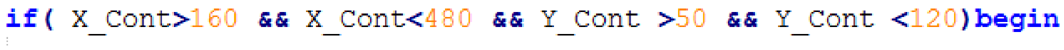
The 12-bit binary image data of each R, G, and B has been truncated and stored into the SDRAM. The B data and half of the G data have been stored into the first SDRAM. The starting address of the first SDRAM is at 0. The R data and the other half of the G data has been stored into the second SDRAM. The starting of the second SDRAM is at 23’h100000. The total number of bytes to be written for both SDRAM is defined to be 8’h50.

Note that the writing process uses the D5M pixel clock while the reading process uses the VGA control clock. The read and write process uses different clocks so that read and write will not occur at the same time.

3.8 CROP_YSTART, CROP_XSTART, CROP_YEND and CROP_XEND

These four modules will run concurrently. They are implemented to localize the eyes from a 10-bit binary image and it will output the starting and the ending point of the image to be segmented. The starting point was obtained by getting the position of the first existence of the black pixel while the ending point was obtained by getting the position of the last existence of the black pixel.

These modules consists of two counting registers which are X_Cont and Y_Cont. X_Cont and Y_Cont is used to represent that the position of the particular pixel is operating. The X_Cont and Y_Cont will count to the limit of 640 and 480 respectively and reset back to 0 if it reaches the maximum value. The localizing process has a fixed localizing range, the range of localizing is as shown in Figure 3.30 below.



```
if( X_Cont>160 && X_Cont<480 && Y_Cont >50 && Y_Cont <120)begin
```

Figure 3.30 Range of localization.

The flow chart for the CROP_YSTART, CROP_XSTART, CROP_YEND and CROP_XEND is as shown in Figure 3.31, Figure 3.32, Figure 3.33 and Figure 3.34 respectively.

The system will operate the process of the flow chart shown in Figures 3.31 to 3.34 at every positive edge of the clock cycle. These modules will pass the YSTART, XSTART, YEND and XEND value to the next module. Note that all the sub-modules labelled as Block 8 are actually running concurrently. Thus, the counting of the X_Cont and Y_Cont are synchronized.

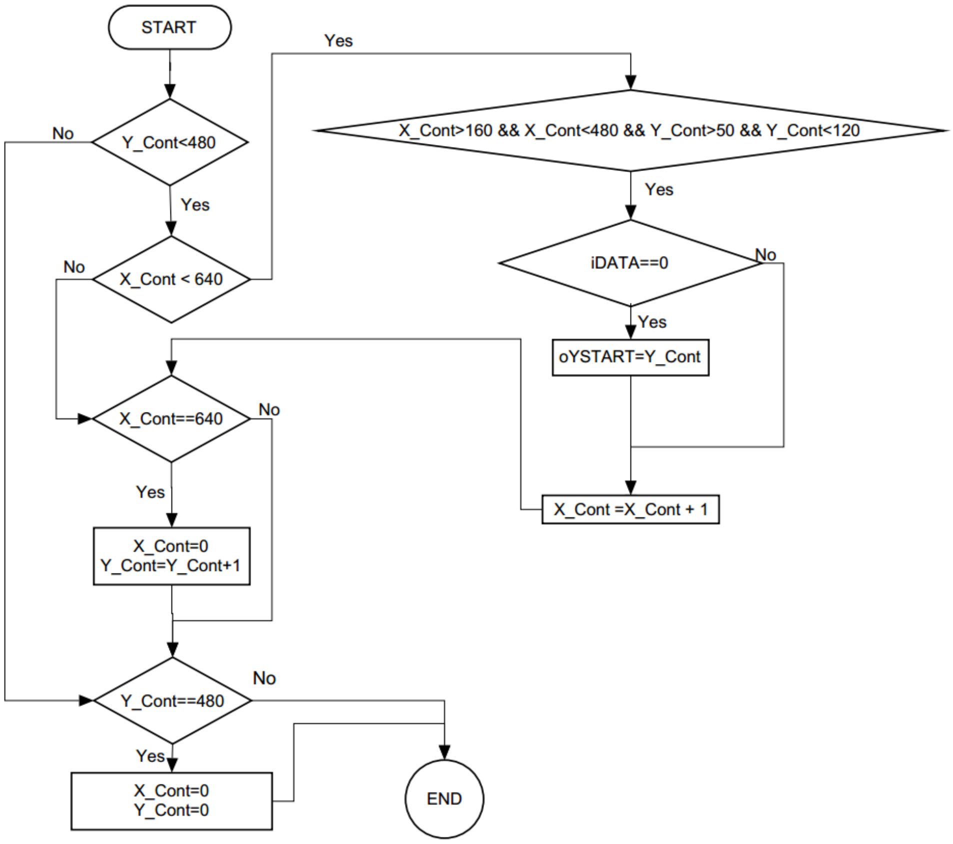


Figure 3.31 Flow chart to obtain CROP_YSTART point.

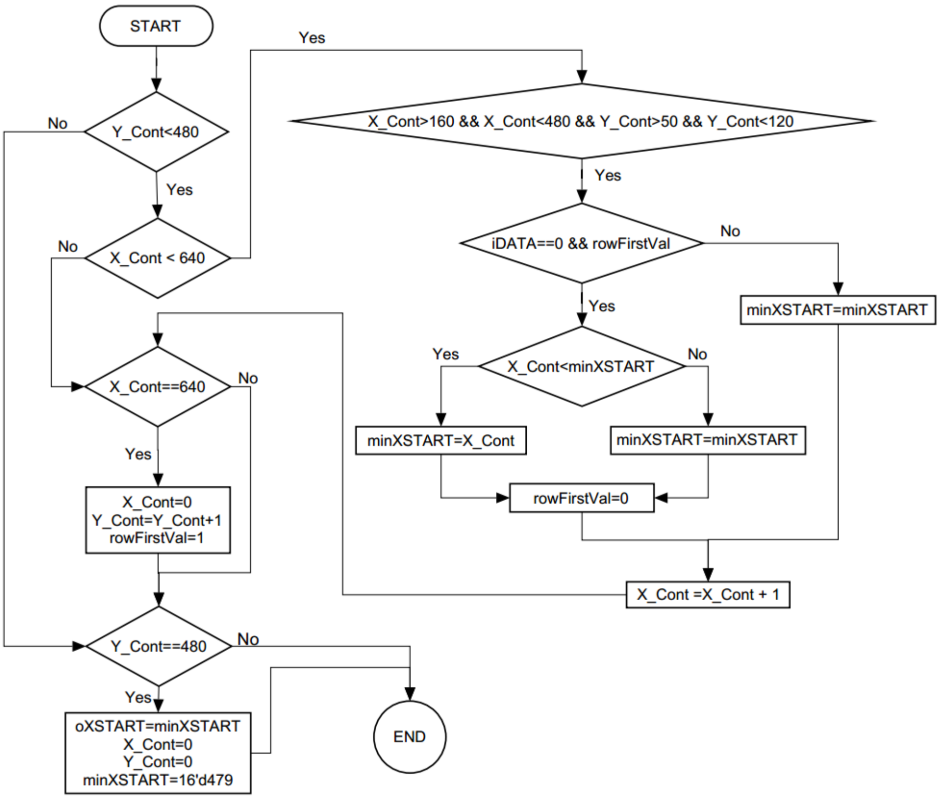


Figure 3.32 Flow chart to obtain CROP_XSTART point.

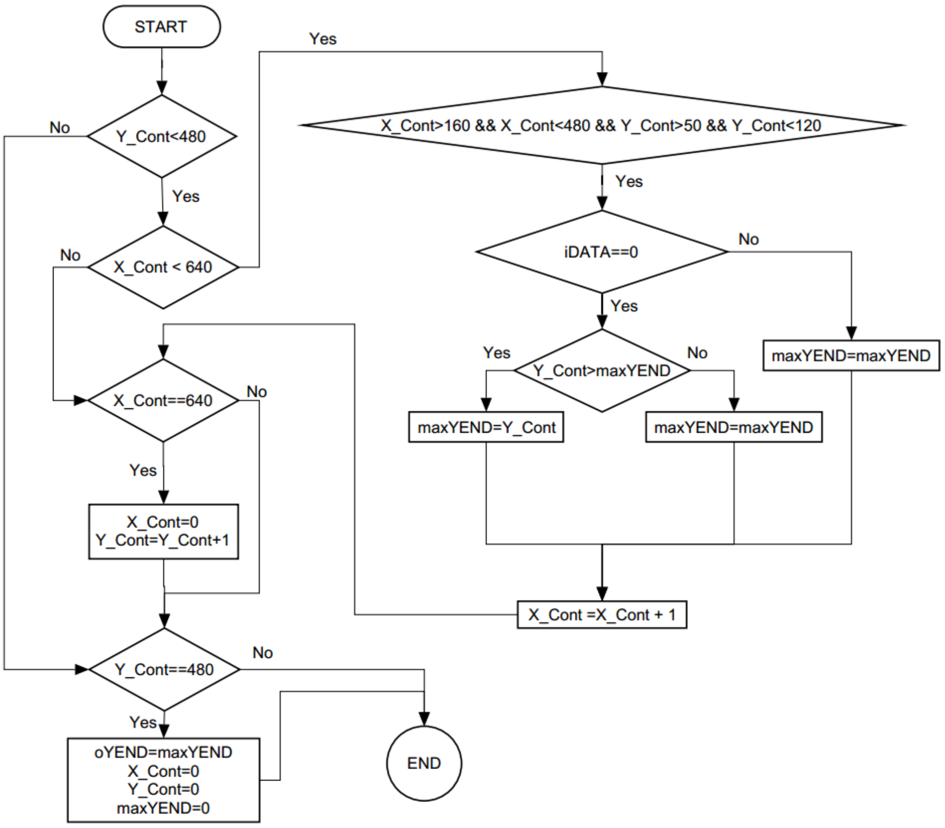


Figure 3.33Flow chart to obtain CROP_YEND point.

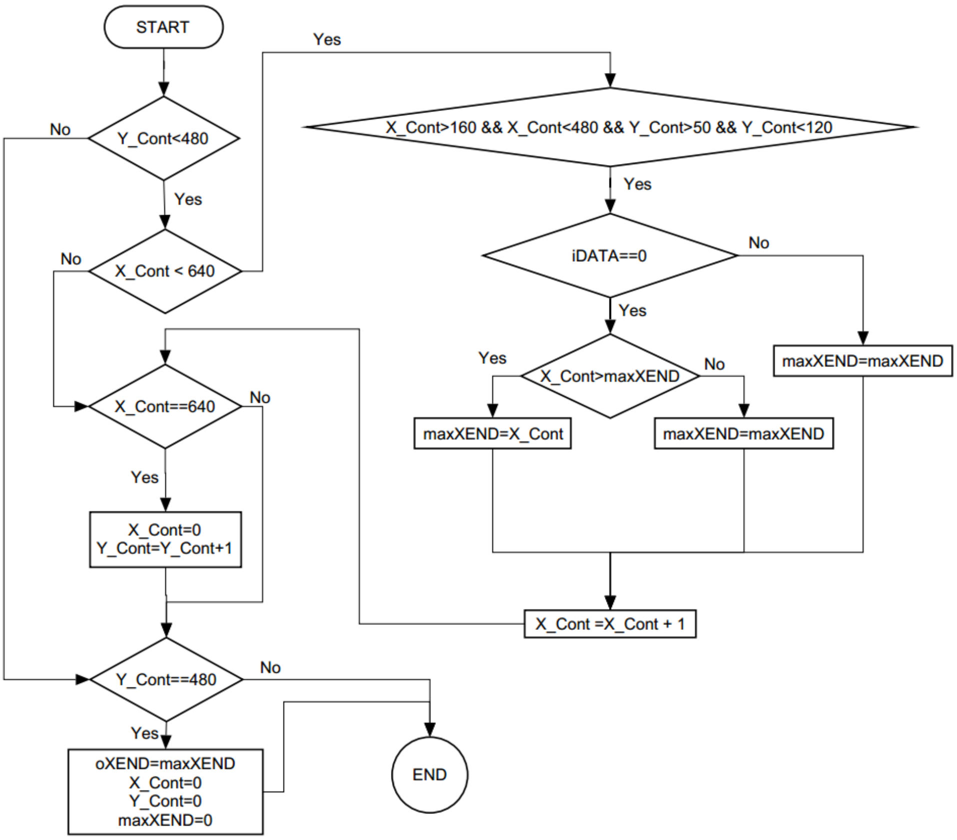
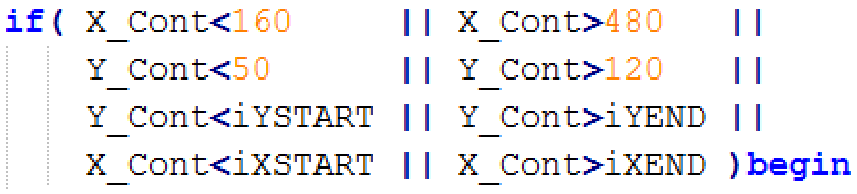


Figure 3.34Flow chart to obtain CROP_XEND point.

3.9 IMAGE_CROP

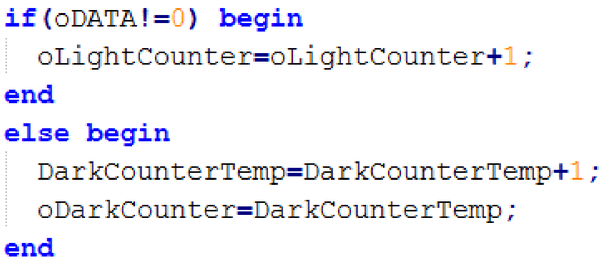
This module will receive the points from the previous block to segment the image by referring to these points. The image will segment from XSTART to XEND and from YSTART to YEND. The range condition of segmented image is as shown in the Figure 3.35. Note that when the X_Cont and Y_Cont matches the condition as specified below, the pixel data will be defined as black pixel data. Otherwise, the output data will follow the input data which is the segmented image.



```
if( X_Cont<160       || X_Cont>480    ||
    Y_Cont<50        || Y_Cont>120    ||
    Y_Cont<iYSTART || Y_Cont>iYEND ||
    X_Cont<iXSTART || X_Cont>iXEND )begin
```

Figure 3.35 Range condition for a segmented image.

This module will also count the amount of black pixel within the image segmented depending on the point specified. The Verilog HDL representation of the black pixel counting is as shown in Figure 3.36.

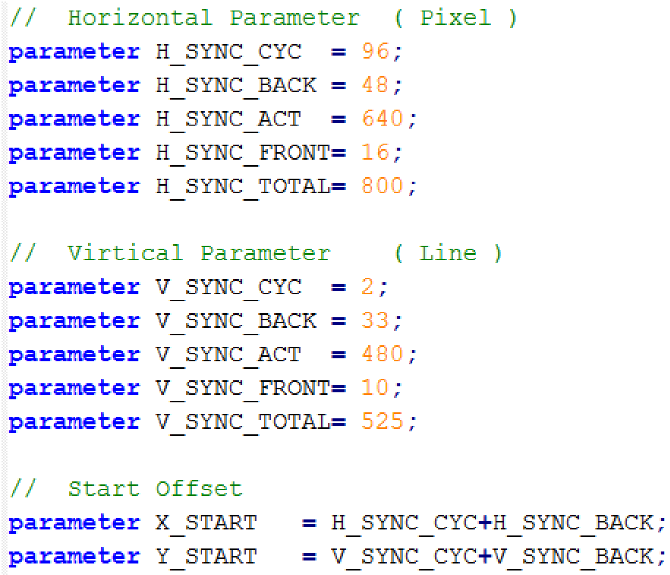


```
if(oDATA!=0) begin
  oLightCounter=oLightCounter+1;
end
else begin
  DarkCounterTemp=DarkCounterTemp+1;
  oDarkCounter=DarkCounterTemp;
end
```

Figure 3.36 Verilog HDL representation of Black pixel counting.

3.10 VGA Controller

The VGA Controller controls the VGA display peripheral. This module defines the parameters for the VGA display. The parameters defined are as shown in Figure 3.37. The parameters refer to the VGA's timing specification. The VGA horizontal timing specification and the VGA vertical timing specification is as shown in Table 3.38 and Table 3.39 respectively. The clock frequency used for the VGA controller is 25MHz. Note that the timing value shown in Table 3.2 has to be multiplied with the pixel clock in order to get the parameters defined as shown in Figure 3.37. The timing diagram of a VGA horizontal timing specification is as shown in Figure 3.38. The timing diagram for vertical timing specification can also be referred from Figure 3.38.



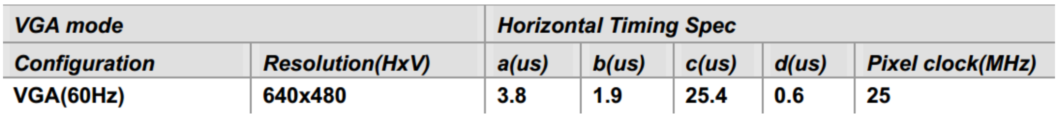
```
//  Horizontal Parameter  ( Pixel )
parameter H_SYNC_CYC  = 96;
parameter H_SYNC_BACK = 48;
parameter H_SYNC_ACT  = 640;
parameter H_SYNC_FRONT= 16;
parameter H_SYNC_TOTAL= 800;

//  Virtical Parameter    ( Line )
parameter V_SYNC_CYC  = 2;
parameter V_SYNC_BACK = 33;
parameter V_SYNC_ACT  = 480;
parameter V_SYNC_FRONT= 10;
parameter V_SYNC_TOTAL= 525;

//  Start Offset
parameter X_START    = H_SYNC_CYC+H_SYNC_BACK;
parameter Y_START    = V_SYNC_CYC+V_SYNC_BACK;
```

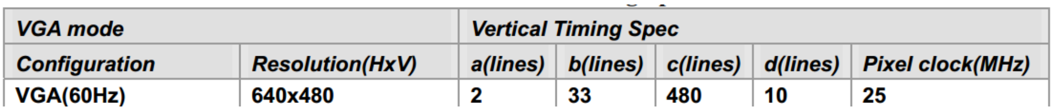
Figure 3.37 Parameters for VGA display.

Table 3.2 VGA's horizontal timing specification.



VGA mode		Horizontal Timing Spec				
Configuration	**Resolution(HxV)**	**a(us)**	**b(us)**	**c(us)**	**d(us)**	**Pixel clock(MHz)**
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25

Table 3.3 VGA's vertical timing specification.



VGA mode		Vertical Timing Spec				
Configuration	**Resolution(HxV)**	**a(lines)**	**b(lines)**	**c(lines)**	**d(lines)**	**Pixel clock(MHz)**
VGA(60Hz)	640x480	2	33	480	10	25

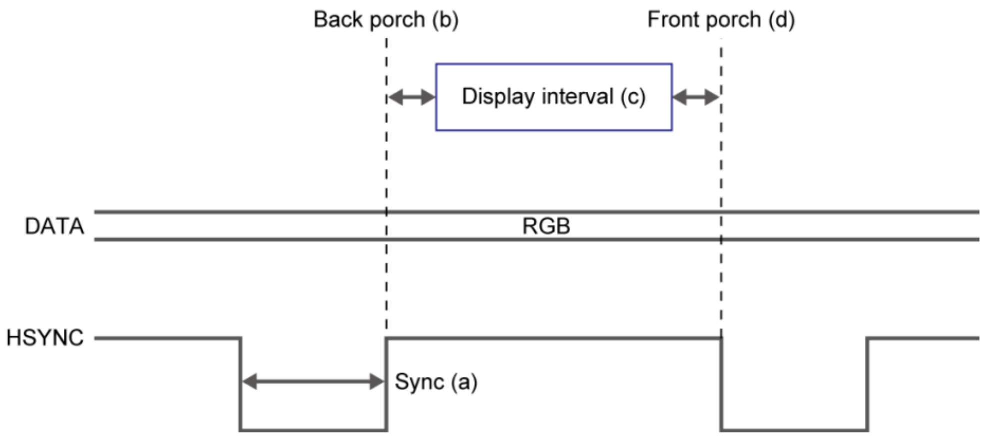
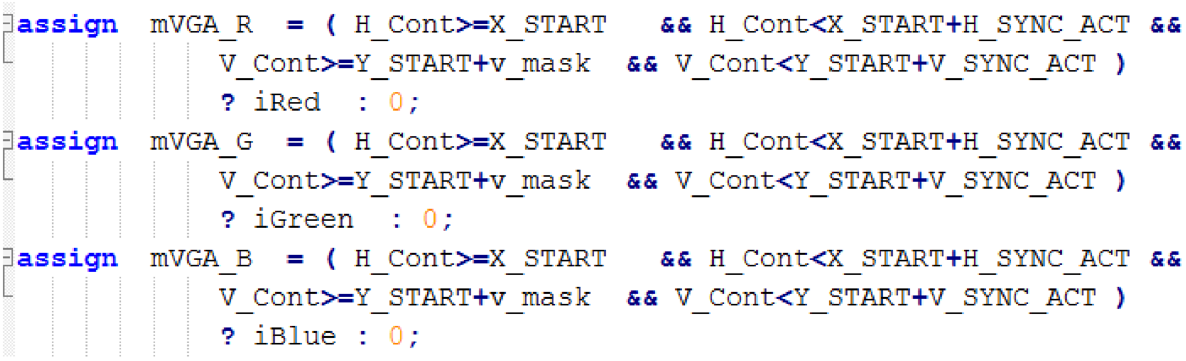


Figure 3.38 Timing diagram of VGA's horizontal timing specification.

The Verilog HDL representation of the outputting of the R, G and B value with the specification defined is as shown in Figure 3.39.

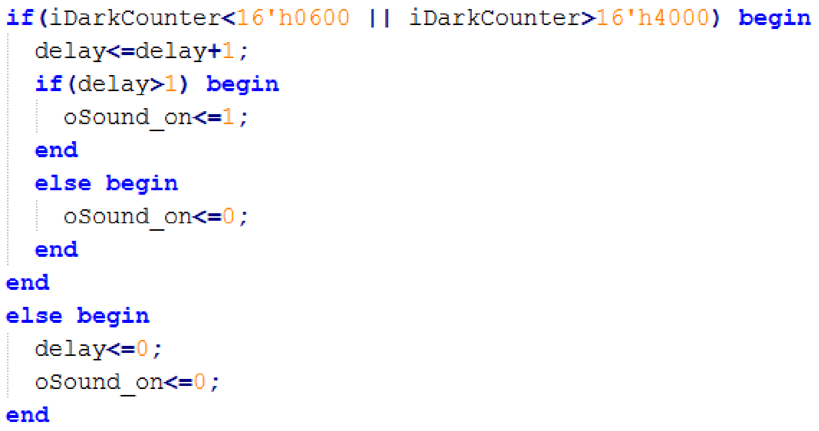


```
assign  mVGA_R  = ( H_Cont>=X_START   && H_Cont<X_START+H_SYNC_ACT &&
                    V_Cont>=Y_START+v_mask  && V_Cont<Y_START+V_SYNC_ACT )
                    ? iRed  : 0;
assign  mVGA_G  = ( H_Cont>=X_START   && H_Cont<X_START+H_SYNC_ACT &&
                    V_Cont>=Y_START+v_mask  && V_Cont<Y_START+V_SYNC_ACT )
                    ? iGreen  : 0;
assign  mVGA_B  = ( H_Cont>=X_START   && H_Cont<X_START+H_SYNC_ACT &&
                    V_Cont>=Y_START+v_mask  && V_Cont<Y_START+V_SYNC_ACT )
                    ? iBlue : 0;
```

Figure 3.39 Verilog HDL representation of the outputting of the R, G and B value with the specification defined.

3.11 SOUND SEL

This module is to recognize the image and decide whether or not the audio codec is to be activated. The recognition is determined by the Dark Counter. The Verilog HDL code representation of this module is as shown in Figure 3.40.



```
if(iDarkCounter<16'h0600 || iDarkCounter>16'h4000) begin
  delay<=delay+1;
  if(delay>1) begin
    oSound_on<=1;
  end
  else begin
    oSound_on<=0;
  end
end
else begin
  delay<=0;
  oSound_on<=0;
end
```

Figure 3.40 Verilog HDL code representation of the SOUND SEL.

From the Figure 3.40 above, if the number of black pixel count in Block 9 is matches the condition mentioned in Figure 3.40, the system will wait for 3 seconds before activating the audio codec. Otherwise, the audio codec will remain idle.

Chapter 4 Results and Discussions

The results and discussions of this work will be presented in this chapter. Results obtained from verification using HDL based test benches simulations in ModelSim as well as hardware analysis. The hardware used is the FPGA Cyclone IV and with the D5M CMOS camera.

4.1 CMOS Image Sensor

Before capturing the image, the D5M camera has to be connected to the GPIO of the FPGA and the CCD lens of the camera has to be located approximately 1.22m above the ground. The CCD camera lens has to be tuned to a suitable focal point to obtain a clearer image. The point captured image without a properly tuned focal point is as shown in Figure 4.1 below. While the captured image with a properly tuned focal point is as shown in Figure 4.2.

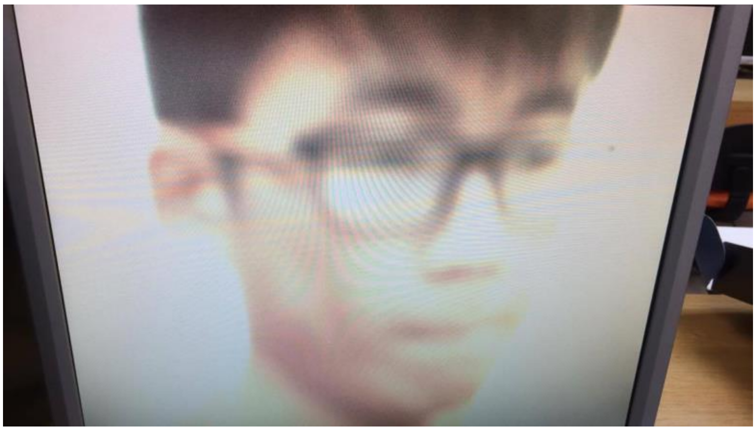


Figure 4.1 Captured image with the focal point not properly tuned.

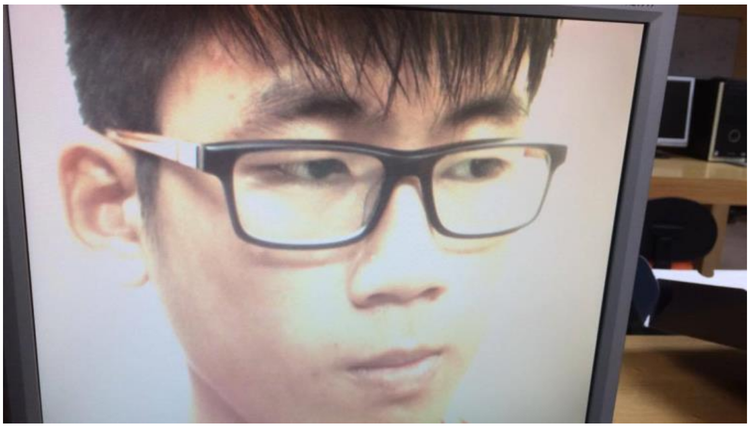


Figure 4.2 Captured image with the focal point properly tuned.

4.2 I2C Controller and I2C Configuration

Table 4.1 shows that the I2C configuration have been changed from the default value given by the example from the Terasic. The second column in Table 4.1 shows the default configuration given by Terasic while the third column in Table 4.1 is the proposed setting for the Drowsiness Detection System. The output VGA result obtained for the default setting and the proposed setting is as shown in Figure 4.3 and Figure 4.4 respectively.

The Column Mirror setting has been disabled in the proposed setting. So that the imaged observed is not reversed as shown in Figure 4.3. For the user who are looking into the camera, if the Column Mirror is disabled, the result observed will be like looking at the mirror. The exposure time has been reduced so that the video observed does not lag. However, decreasing the exposure time dims the image. Hence, the R, G and B gain has been increased to obtain a brighter image. The result can be observed by comparing Figures 4.3 and 4.4 respectively.

The Row size and column size has been set to 960 and 1280 respectively because the VGA standard size is 640x480. If the default value is used, dividing by 2 will not get the VGA to display a standard image size. The VGA is unable to display the entire image as can be observed in Figure 4.3. Observe the existence of the dark pixels at right edge and the bottom edge. The row and column mode represents the skipping mode and the binning mode which is not relevant to the current research. Thus, these modes have been disabled.

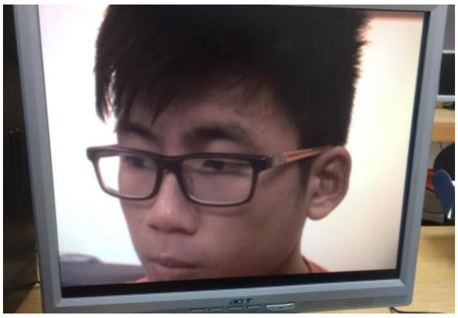


Figure 4.3 Image output for the default setting.

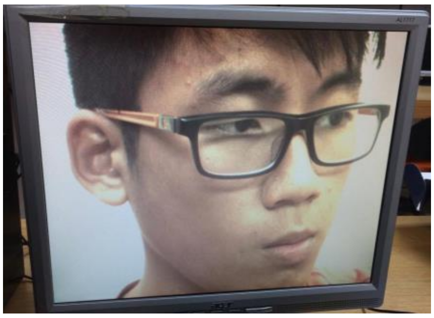


Figure 4.4 Image output for the proposed setting.

Table 4.1 I2C Configuration for the default setting and the proposed setting

Registers configuration	Figure 4.3 (Default Settings)	Figure 4.4 (Proposed Settings)
Mirror Row and Columns	49152	32768
Exposure Time	1984	1328
G1 Gain	19	4146
B Gain	154	4916
R Gain	412	4920
G2 Gain	19	4146
Row size	1200	960
Column size	1600	1280
Row Mode	17	0
Column Mode	17	0

4.3 CMOS Sensor Data Capture

The tested result for the CMOS Sensor Data Capture is as shown in Figure 4.5 and Figure 4.6. From the results shown in Figure 4.5, the X_Cont and Y_Cont will not start counting until if and only if the FVAL and LVAL have been set in the correct sequence. If they are set in the correct sequence, the iDATA will output to Block 4 with the values of X_Cont and Y_Cont. From the console shown in Figure 4.6, the X_Cont will reset back to zero when it reaches 1279 and the Y_Cont will run forever. Note that the iDATA is a random data that is set to test the program. The oFrame_Cont indicates the number of frame that has been passed.

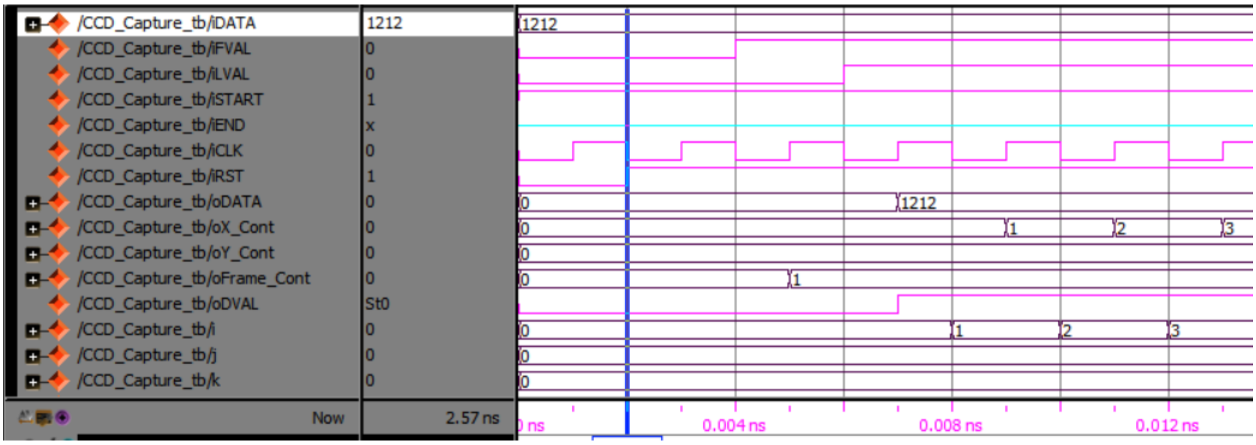
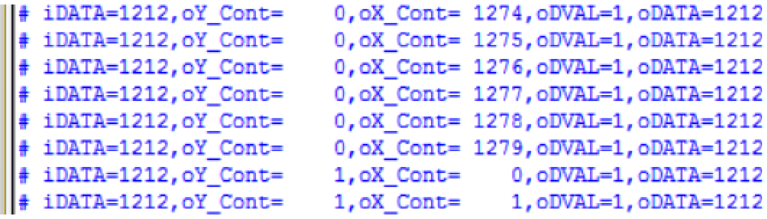


Figure 4.5 Waveform result of CCD Capture.



```
# iDATA=1212,oY_Cont=    0,oX_Cont= 1274,oDVAL=1,oDATA=1212
# iDATA=1212,oY_Cont=    0,oX_Cont= 1275,oDVAL=1,oDATA=1212
# iDATA=1212,oY_Cont=    0,oX_Cont= 1276,oDVAL=1,oDATA=1212
# iDATA=1212,oY_Cont=    0,oX_Cont= 1277,oDVAL=1,oDATA=1212
# iDATA=1212,oY_Cont=    0,oX_Cont= 1278,oDVAL=1,oDATA=1212
# iDATA=1212,oY_Cont=    0,oX_Cont= 1279,oDVAL=1,oDATA=1212
# iDATA=1212,oY_Cont=    1,oX_Cont=    0,oDVAL=1,oDATA=1212
# iDATA=1212,oY_Cont=    1,oX_Cont=    1,oDVAL=1,oDATA=1212
```

Figure 4.6 Console display of the CMOS data capture result

4.4 RAW2RGB

The RGB image is the original image that has been captured and displayed without any processing. These images are the images shown in Figure 4.2 and Figure 4.4. It can be noted that the topology used in Figure 3.22 was only applied on the column mirror enable together with the row mirror disable. If the mirror configuration was to be modified, the equation in Figure 3.22 has to be modified also.

4.5 RGB2GRAY

The RGB image has been converted to a gray scale image by using the averaging method. The images before and after the gray scale conversion is as shown in Figure 4.7 and Figure 4.8 respectively.

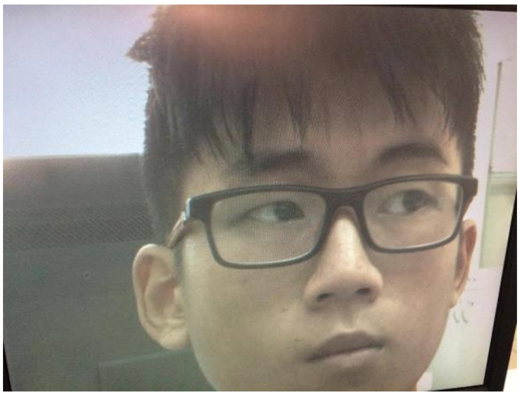


Figure 4.7 The VGA output image before conversion.

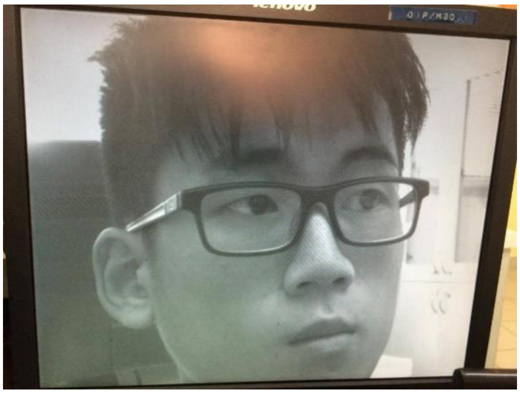


Figure 4.8 The VGA output image after conversion.

4.6 GRAY2BINARY

The Binary image is the key image to localize where the eyes of the driver is. The threshold has to be set to a value that the eyes are clearly visible without any noise. Usually the threshold is set to be lower in the darker room and set to be higher in a brighter room. But unfortunately, the eyes are no longer visible if the room brightness was too dark. The result of the image when the room is too dark is as shown in Figure 4.9. Thus, the environment has been constrained to be in a bright room. The system will begins to fail after 6.00pm when the sun begins to set.

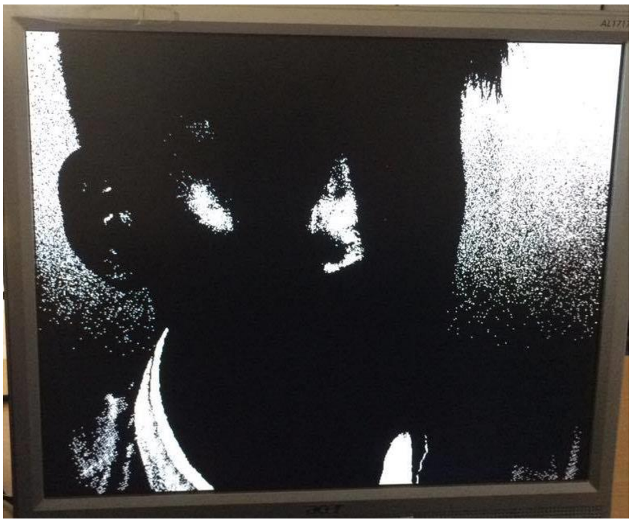


Figure 4.9 Image captured when the environment is too dark.

The threshold has been set to a value of 12’d3003 in a bright room. The output image obtained for the threshold is as shown in Figure 4.11 and the grayscale image before converting to binary image is as shown in Figure 4.10.

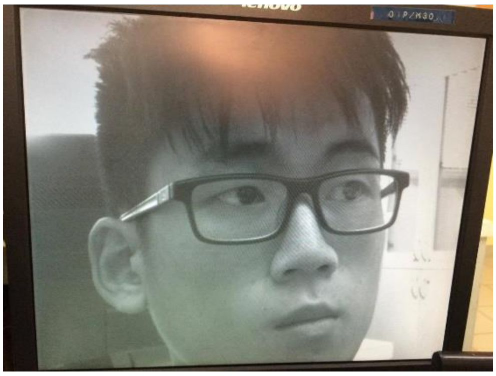


Figure 4.10 Gray scale image before conversion.

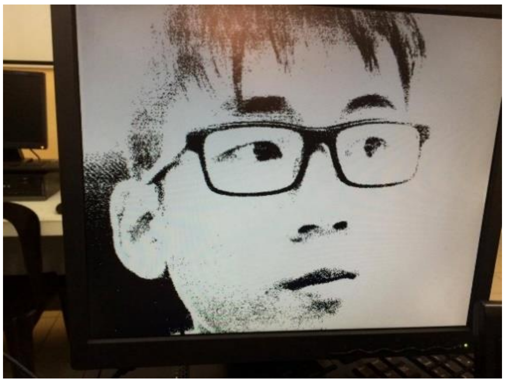


Figure 4.11 Binary image after conversion.

4.7 CROP_YSTART, CROP_XSTART, CROP_YEND and CROP_XEND

This drowsiness detection system is unable to recognize the user that is wearing spectacles because the spectacles ruins the localization process to track where the position of the eyes are really located. The binary image used to perform the localization is as shown in Figure 4.12.

The localization of the eyes from the binary image shown in Figure 4.12 with the condition mentioned in Figure 3.30 is as shown in Figure 4.13. The value of YSTART, XSTART, YEND and XEND will then be passed to the following module.

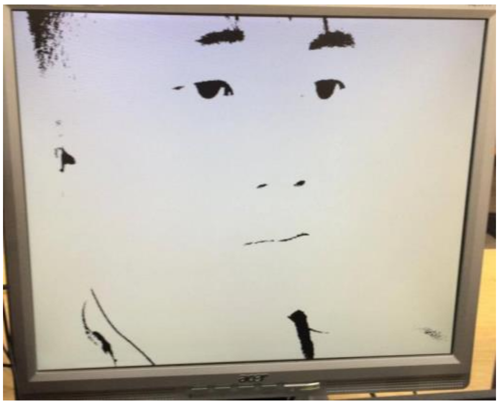


Figure 4.12 Binary image of the user without spectacles.

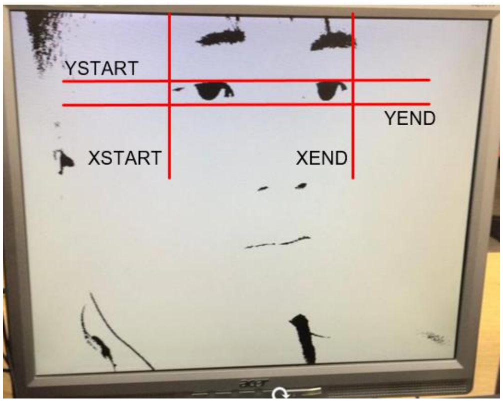


Figure 4.13 Localization of the eyes.

4.8 IMAGE_CROP

From Figure 4.13, this module will segment out the image and the image will then be left with only both the eyes. The result of image segmentation is as shown in Figure 4.14. The position of the eyes must be at around the position shown in 4.13. This process can be operated in real time which means the point given from the previous block will change immediately when the head of the user moved and it can be observed immediately at the VGA display.

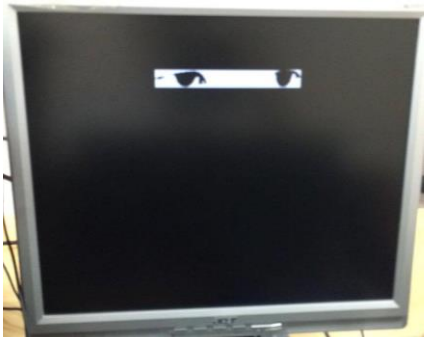
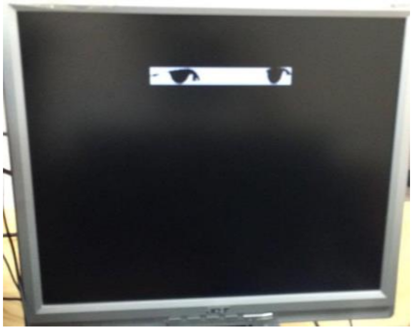


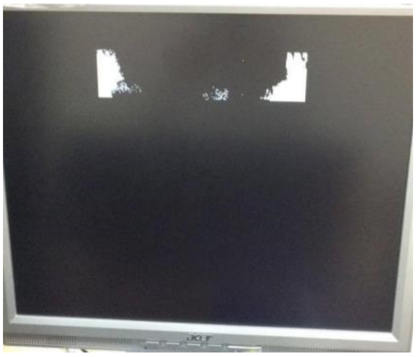
Figure 4.14 Result of segmentation.

4.9 SOUND SEL

The recognition module consists of certain condition to sound the audio codec. The conditions are when the user closes his/her eyes more than 3 seconds or the user is faced down. The possible scenarios for driver drowsiness detection (sounding of alarm) is as listedin Table 4.2 below.

Table 4.3 Different scenarios for the audio codec to response.

Scenarios	**Audio Codec**
Normal facing 	The audio codec does not enable whenever the image crop detected only the eyes.

Eyes closed < 3 seconds 	The audio codec will not enable also if the eyes are closed for less than 3 seconds. If the eyes reopens, the timer will reset.
Eyes closed ≥ 3seconds 	The audio codec will be enabled if the eyes are closed exceeding 3 seconds. The audio codec will be disabled immediately if the eyes reopen, the timer will then reset.
Head turns to the left or right 	The audio codec does not enable whenever the cropped image detects the side view of the user.
Head faced down 	The audio codec will be enabled if the user’s head faces down. The audio codec will also be disabled immediately if the user faces back up normally.

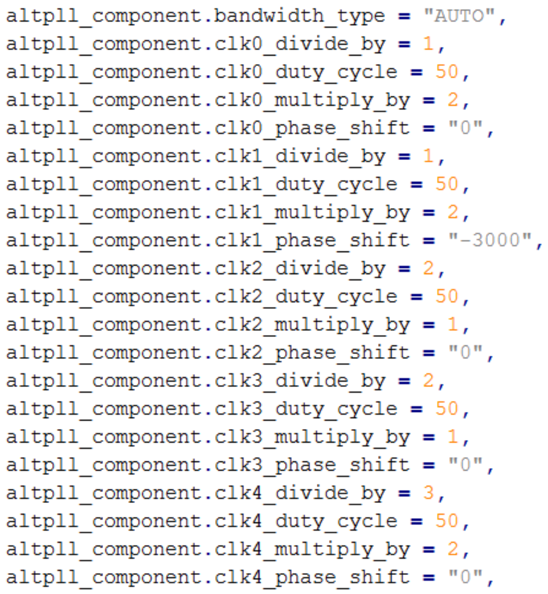
Head faced up 	The audio codec will also be enabled when the user faces back up. When the user faces up, the image detected will only be the nostrils.

4.10 Overall Results

Overall, this Drowsiness Detection System uses up 2493 (2%) of the logic elements available in the Cyclone IV FPGA and has a total thermal power dissipation of 311.61mW. This system has an accuracy of 96% correct judgement rate as long as the system was operated inside a bright room. The performance will be degraded if the room became darker.

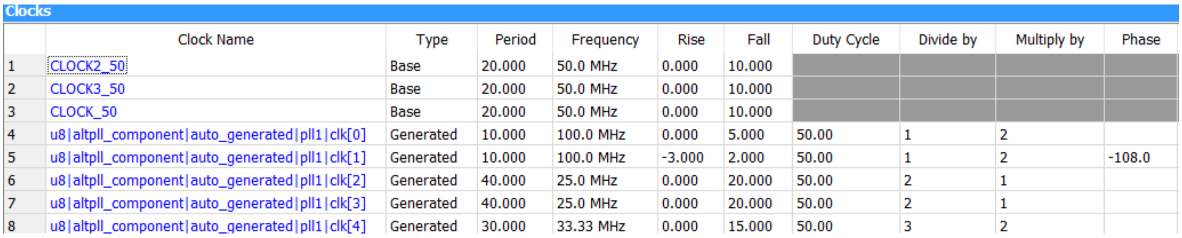
The main clock being used in this Drowsiness Detection system is the CLOCK2_50, which is a 50 MHz clock. With this 50 MHz clock, the Phase Lock Loop (PLL) has been used to divide the clock to be used in different peripheral such as D5M camera, SDRAM, VGA display and the audio codec. The clock used for each peripheral is as shown in Figure 4.15. From the Figure 4.15 below, the clock used for sdram_ctrl_clk, DRAM, D5M_PIXCLK, VGA_CLK and AUD_CTR_CLK are clk0, clk1, clk2, clk3 and clk4 respectively. Note that the sdram_ctrl_clk is the clock for the host side while the DRAM is the clock for the peripheral. The clock used for all the modules before SDRAM is D5M_PIXCLK and the clock used for all the modules after SDRAM is the VGA_CTRL_CLK. The VGA_CTRL_CLK is the inverse of the VGA_CLK in which VGA_CTRL_CLK is for the host and VGA_CLK is for peripheral.

The generated clock for the above configuration is as shown in Figure 4.15. The speed of the system is as shown in Figure 4.17 with a positive slack of 9.306 ns using 50MHz clock.



```
altpll_component.bandwidth_type = "AUTO",
altpll_component.clk0_divide_by = 1,
altpll_component.clk0_duty_cycle = 50,
altpll_component.clk0_multiply_by = 2,
altpll_component.clk0_phase_shift = "0",
altpll_component.clk1_divide_by = 1,
altpll_component.clk1_duty_cycle = 50,
altpll_component.clk1_multiply_by = 2,
altpll_component.clk1_phase_shift = "-3000",
altpll_component.clk2_divide_by = 2,
altpll_component.clk2_duty_cycle = 50,
altpll_component.clk2_multiply_by = 1,
altpll_component.clk2_phase_shift = "0",
altpll_component.clk3_divide_by = 2,
altpll_component.clk3_duty_cycle = 50,
altpll_component.clk3_multiply_by = 1,
altpll_component.clk3_phase_shift = "0",
altpll_component.clk4_divide_by = 3,
altpll_component.clk4_duty_cycle = 50,
altpll_component.clk4_multiply_by = 2,
altpll_component.clk4_phase_shift = "0",
```

Figure 4.15 PLL component used to apply on the clock.



Clocks

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by	Phase
1	CLOCK2_50	Base	20.000	50.0 MHz	0.000	10.000				
2	CLOCK3_50	Base	20.000	50.0 MHz	0.000	10.000				
3	CLOCK_50	Base	20.000	50.0 MHz	0.000	10.000				
4	u8|altpll_component|auto_generated|pll1|clk[0]	Generated	10.000	100.0 MHz	0.000	5.000	50.00	1	2	
5	u8|altpll_component|auto_generated|pll1|clk[1]	Generated	10.000	100.0 MHz	-3.000	2.000	50.00	1	2	-108.0
6	u8|altpll_component|auto_generated|pll1|clk[2]	Generated	40.000	25.0 MHz	0.000	20.000	50.00	2	1	
7	u8|altpll_component|auto_generated|pll1|clk[3]	Generated	40.000	25.0 MHz	0.000	20.000	50.00	2	1	
8	u8|altpll_component|auto_generated|pll1|clk[4]	Generated	30.000	33.33 MHz	0.000	15.000	50.00	3	2	

Figure 4.16 Generated clock for the configuration made.

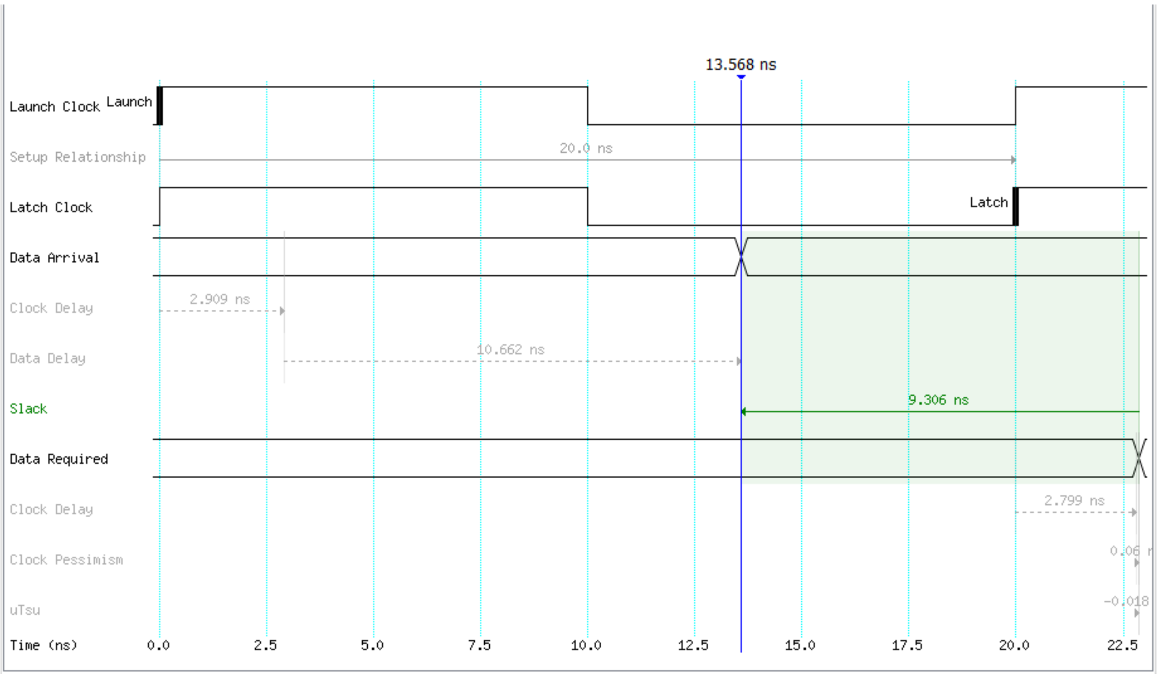


Figure 4.17 Speed of the system using 50MHz clock

Sometimes the obtained images will be as shown in Figure 4.18. There will be some unknown noise coming from nowhere. This issue can be solved by trial and error to modify the threshold in the GRAY2BINARY module to remove these noise. The reason behind threshold modifications to solve the problem is undetermined but this is how the problem was solved.

This solving method can be applied only when the VGA display shows some redundant lines and can also be applied when the audio codec is not functioning sometimes even when it is being enable or disabled.

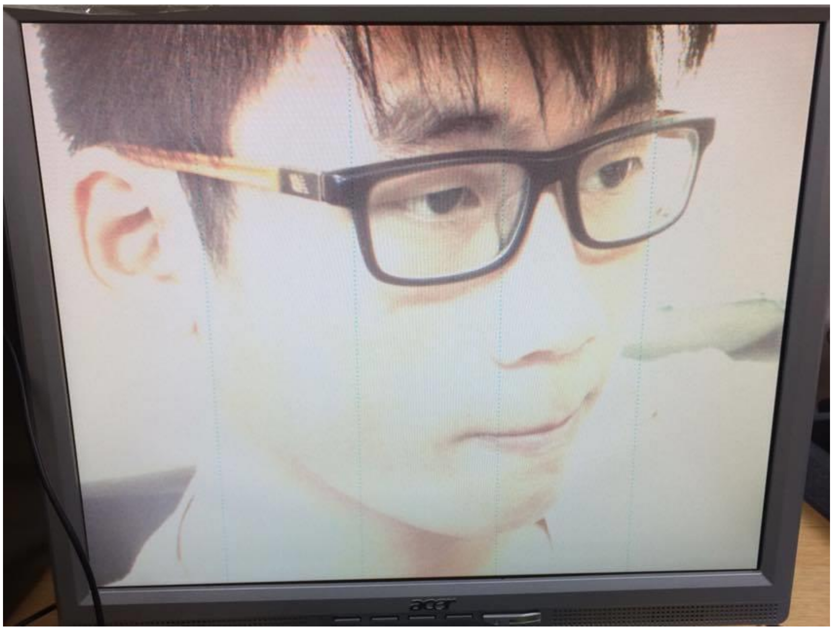


Figure 4.18 Noise existence in the VGA display.

4.11 Analysis

This system has been tested by 10 users. Each user has been taken for three attempts. Almost all result tested matches the expected response given. To be exact, there are only two attempts of failure response given by the system. Mostly the failure of response of the system is because of the distance between the face to the lens of the camera. That is because the distance are affecting the amount of black pixel to be detected. Therefore, the accuracy of the correct judgement rate is 96%.

This system has an accuracy of 96% and uses 2% of the logic elements in the FPGA. As compared to the previous research, the architecture of Yap (2015), Chuah (2015), and Lee (2015) has a recognition accuracy of 94% and used 80% of logic element in the FPGA in which this current work has a higher accuracy and less logic elements used. The accuracy of this current work is also higher than the architecture of Flores (2008).

The summarized Design specification of the Drowsiness detection system is as shown in Table 4.4 below.

Table 2 Design specification of the Drowsiness Detection System.

Design Specification	**Value**
Speed of the system	9.306 ns
Power Dissipation	311.61 mW
Total Logic Element used	2493
Accuracy of correct judgement rate	96%

Chapter 5 Conclusion

The Drowsiness Detection system includes capturing, processing and recognition and it has been implemented by using the FPGA Cyclone IV EP4CE115F29C7. This research focuses on the implementation of Drowsiness Detection System using a less complex methodology. This system also implements a simpler HDL based code as compared to the previous work accomplished by Yap (2015), Chuah (2015), and Lee (2015) in terms of the image segmentation. The image will be captured by using the Terasic CMOS camera and segmented using the thresholding method. This work will be designed by using Quatus II and verified by ModelSim using HDL based testbench. The result has also been verified using the VGA display in real time.

5.1: Future Improvements and Recommendations.

The recognition of the Drowsiness Detection System can be improved by adding an artificial intelligence system which is needed to train and differentiate both the "alert" and "drowsy" images into the system before any recognition can be performed. Examples of artificial intelligent systems are neural networks, fuzzy logic, genetic algorithm etc. A brightness tracking system can also be implemented to track the brightness of the surrounding and change the threshold depending on the brightness of the surrounding so that the system can be made more automated. Besides being used inside the car, this drowsiness detection system can also be used in the class room to avoid students from falling off to sleep or to alert the students that he is in a sleepy condition.

References

[1] Ying-CheKuo, & Wen-Ling Hsu, 2010 “Real-Time Drowsiness Detection System for Intelligent Vehicles”,Real -Time Drowsiness Detection System.

[2] Pavlidis, I., Morellas, V. and Papanikolopoulos, N., 2000. A vehicle occupant counting system based on near-infrared phenomenology and fuzzy neural classification. *IEEE Transactions on intelligent transportation systems*, *1*(2), pp.72-85.

[3] Kassem, A., Hamad, M. and Haidamous, E., 2009, July. Image compression on FPGA using DCT. In *Advances in Computational Tools for Engineering Applications, 2009. ACTEA'09. International Conference on* (pp. 320-323). IEEE.

[4] Acasandrei, L. and Barriga, A., 2012, January. FPGA implementation of an embedded face detection system based on LEON3. In *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)* (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[5] Mathur, M.K. and Mathur, G., 2012. Image Compression using DFT through Fast Fourier Transform Technique. *International Journal of Emerging Trends and Technology in Computer Science”, ISSN*, pp.2278-6856.

[6] Flores, M.J., Armingol, J.M. and de la Escalera, A., 2008, June. Real-time drowsiness detection system for an intelligent vehicle. In *Intelligent Vehicles Symposium, 2008 IEEE* (pp. 637-642). IEEE.

[7] Yap, Chuah and Lee, 2015, Development of an FPGA based Iris Recognition System.

[8] Bharambe, S.S. and Mahajan, P.M., 2015 Implementation of Real Time Driver Drowsiness Detection System.

[9] Terasic 2010. TRDB-D5m User Guide. 5 Mega Pixel Digital Camera Development.

[10] Terasic 2010. DE2_115_User_Manual.

[11] Terasic 2009. TRDB-D5M Hardware Specification.

Appendix:

Main Module (DE2_115_CAMERA)

```
module DE2_115_CAMERA(
  //////////// CLOCK //////////
  CLOCK_50,
  CLOCK2_50,
  CLOCK3_50,

  //////////// LED //////////
  LEDG,
  LEDR,

  //////////// KEY //////////
  KEY,

  //////////// SW //////////
  SW,

  //////////// SEG7 //////////
  HEX0,
  HEX1,
  HEX2,
  HEX3,
  HEX4,
  HEX5,
  HEX6,
  HEX7,

  //////////// VGA //////////
  VGA_B,
  VGA_BLANK_N,
  VGA_CLK,
  VGA_G,
  VGA_HS,
  VGA_R,
  VGA_SYNC_N,
  VGA_VS,

  //////////// Audio //////////
  AUD_ADCDAT,
  AUD_ADCLRCK,
  AUD_BCLK,
  AUD_DACDAT,
  AUD_DACLRCK,
  AUD_XCK,

  //////////// I2C for Audio Tv-Decoder  //////////
  I2C_SCLK,
  I2C_SDAT,

  //////////// TV Decoder //////////
  TD_CLK27,
  TD_DATA,
  TD_HS,
  TD_RESET_N,
  TD_VS,

  //////////// SDRAM //////////
  DRAM_ADDR,
  DRAM_BA,
  DRAM_CAS_N,
  DRAM_CKE,
  DRAM_CLK,
  DRAM_CS_N,
  DRAM_DQ,
  DRAM_DQM,
  DRAM_RAS_N,
  DRAM_WE_N,

  //////////// GPIO, GPIO connect to D5M - 5M Pixel Camera //////////
  D5M_D,
  D5M_FVAL,
  D5M_LVAL,
  D5M_PIXLCLK,
  D5M_RESET_N,
  D5M_SCLK,
  D5M_SDATA,
  D5M_STROBE,
  D5M_TRIGGER,
  D5M_XCLKIN
);

//========================================================
//  PORT declarations
//========================================================

/////////// CLOCK //////////
input               CLOCK_50;
input               CLOCK2_50;
input               CLOCK3_50;

/////////// LED //////////
output       [8:0]    LEDG;
output       [17:0]   LEDR;

/////////// KEY //////////
input        [3:0]    KEY;

/////////// SW //////////
input       [17:0]    SW;

/////////// SEG7 //////////
output                [6:0]  HEX0;
output                [6:0]  HEX1;
output                [6:0]  HEX2;
output                [6:0]  HEX3;
output                [6:0]  HEX4;
output                [6:0]  HEX5;
output                [6:0]  HEX6;
output                [6:0]  HEX7;

/////////// VGA //////////
output         [7:0]  VGA_B;
output                VGA_BLANK_N;
output                VGA_CLK;
output         [7:0]  VGA_G;
output                VGA_HS;
output         [7:0]  VGA_R;
output                VGA_SYNC_N;
output                VGA_VS;

/////////// Audio //////////
input            AUD_ADCDAT;
inout            AUD_ADCLRCK;
inout            AUD_BCLK;
output                   AUD_DACDAT;
inout            AUD_DACLRCK;
output                   AUD_XCK;

/////////// I2C for Audio Tv-Decoder  //////////
output                I2C_SCLK;
inout                 I2C_SDAT;

/////////// TV Decoder //////////
input                    TD_CLK27;
input                 [7:0]      TD_DATA;
input                    TD_HS;
output                   TD_RESET_N;
input                    TD_VS;

/////////// SDRAM //////////
output           [12:0]  DRAM_ADDR;
output           [1:0]   DRAM_BA;
output                           DRAM_CAS_N;
output                           DRAM_CKE;
output                   DRAM_CLK;
output                   DRAM_CS_N;
inout            [31:0]  DRAM_DQ;
output                 [3:0]     DRAM_DQM;
output                   DRAM_RAS_N;
output                   DRAM_WE_N;

//////////// GPIO, GPIO connect to D5M - 5M Pixel Camera //////////
input         [11:0]     D5M_D;
input                    D5M_FVAL;
input                    D5M_LVAL;
input                    D5M_PIXLCLK;
output                   D5M_RESET_N;
output                   D5M_SCLK;
inout                    D5M_SDATA;
input                    D5M_STROBE;
output                   D5M_TRIGGER;
output                   D5M_XCLKIN;


//========================================================
//  REG/WIRE declarations
//========================================================
wire  [15:0]  Read_DATA1;
wire  [15:0]  Read_DATA2;

wire  [11:0]  mCCD_DATA;
wire          mCCD_DVAL;
wire          mCCD_DVAL_d;
wire  [15:0]  X_Cont;
wire  [15:0]  Y_Cont;
wire  [9:0]   X_ADDR;
wire  [31:0]  Frame_Cont;
wire          DLY_RST_0;
wire          DLY_RST_1;
wire          DLY_RST_2;
wire          DLY_RST_3;
wire          DLY_RST_4;
wire          Read;
reg     [11:0]  rCCD_DATA;
reg               rCCD_LVAL;
reg               rCCD_FVAL;
wire  [11:0]  sCCD_R;
wire  [11:0]  sCCD_G;
wire  [11:0]  sCCD_B;
wire          sCCD_DVAL;

wire  sdram_ctrl_clk;
wire  [9:0]     oVGA_R;                                  //      VGA Red[9:0]
wire  [9:0]     oVGA_G;                                  //      VGA Green[9:0]
wire  [9:0]     oVGA_B;                                  //      VGA Blue[9:0]

//power on start
wire             auto_start;
//========================================================
//  Structural coding
//========================================================
// D5M
assign  D5M_TRIGGER      =       1'b1;  // tRIGGER
assign  D5M_RESET_N      =       DLY_RST_1;
assign  VGA_CTRL_CLK = ~VGA_CLK;

assign  LEDR             =       SW;
assign  LEDG             =       Y_Cont;

//fetch the high 8 bits
assign  VGA_R = oVGA_R[9:2];
assign  VGA_G = oVGA_G[9:2];
assign  VGA_B = oVGA_B[9:2];

//D5M read
always@(posedge D5M_PIXLCLK)
begin
```

```
        rCCD_DATA       <=      D5M_D;
        rCCD_LVAL       <=      D5M_LVAL;
        rCCD_FVAL       <=      D5M_FVAL;
end

//auto start when power on
assign auto_start = ((KEY[0])&&(DLY_RST_3)&&(!DLY_RST_4))? 1'b1:1'b0;
//Reset module
Reset_Delay                     u0      (
              .iCLK(CLOCK2_50),
              .iRST(KEY[0]),
              .oRST_0(DLY_RST_0),
              .oRST_1(DLY_RST_1),
              .oRST_2(DLY_RST_2),
              .oRST_3(DLY_RST_3),
              .oRST_4(DLY_RST_4)
                                                );
//D5M I2C control
I2C_CCD_Config          u1      (       //      Host Side
              .iCLK(CLOCK2_50),
              .iRST_N(DLY_RST_2),
              .iEXPOSURE_ADJ(KEY[1]),
              .iEXPOSURE_DEC_p(SW[0]),
              .iZOOM_MODE_SW(SW[16]),
              // I2C Side
              .I2C_SCLK(D5M_SCLK),
              .I2C_SDAT(D5M_SDATA)
                                );
//D5M image capture
CCD_Capture     u2      (
              .oDATA(mCCD_DATA),
              .oDVAL(mCCD_DVAL),
              .oX_Cont(X_Cont),
              .oY_Cont(Y_Cont),
              .oFrame_Cont(Frame_Cont),
              .iDATA(rCCD_DATA),
              .iFVAL(rCCD_FVAL),
              .iLVAL(rCCD_LVAL),
              .iSTART(!KEY[3]|auto_start),
              .iEND(!KEY[2]),
              .iCLK(D5M_PIXLCLK),
              .iRST(DLY_RST_2)
                        );
//D5M raw date convert to RGB data
RAW2RGB         u3      (
              .iCLK(D5M_PIXLCLK),
              .iRST(DLY_RST_1),
              .iDATA(mCCD_DATA),
              .iDVAL(mCCD_DVAL),
              .oRed(wVGA_R),
              .oGreen(wVGA_G),
              .oBlue(wVGA_B),
              .oDVAL(sCCD_DVAL),
              .iX_Cont(X_Cont),
              .iY_Cont(Y_Cont)
                        );

wire [11:0]wVGA_R ;
wire [11:0]wVGA_G ;
wire [11:0]wVGA_B ;

wire gCCD_DVAL;
wire [11:0] gDATA;

RGB2GRAY        u4(
        .oDVAL(gCCD_DVAL),
        .oDATA(gDATA),
        .iRed(wVGA_R),
        .iGreen(wVGA_G),
        .iBlue(wVGA_B),
        .iCLK(D5M_PIXLCLK),
        .iRST(DLY_RST_1),
        .iDVAL(sCCD_DVAL)
                );
```

```
wire [11:0]wDISP_R;
wire [11:0]wDISP_G;
wire [11:0]wDISP_B;

assign wDISP_R =SW[1] ? gDATA:wVGA_R;
assign wDISP_G =SW[1] ? gDATA:wVGA_G;
assign wDISP_B =SW[1] ? gDATA:wVGA_B;

//BLACK AND WHITE
wire bCCD_DVAL;
wire [11:0]bDATA;

GRAY2BINARY     u5(
        .oDVAL(bCCD_DVAL),
        .oDATA(bDATA),
        .iDATA(gDATA),
        .iCLK(D5M_PIXLCLK),
        .iRST(DLY_RST_2),
        .iDVAL(gCCD_DVAL)
                );

wire [11:0]bDISP_R;
wire [11:0]bDISP_G;
wire [11:0]bDISP_B;

assign bDISP_R =SW[2] ? bDATA:wDISP_R;
assign bDISP_G =SW[2] ? bDATA:wDISP_G;
assign bDISP_B =SW[2] ? bDATA:wDISP_B;

sdram_pll       u8      (
              .inclk0(CLOCK2_50),
              .c0(sdram_ctrl_clk),
              .c1(DRAM_CLK),
              .c2(D5M_XCLKIN), //25M
              .c3(VGA_CLK),     //25M
              .c4(AUD_CTRL_CLK)
                        );

//SDRam Read and Write as Frame Buffer
Sdram_Control   u9      (       //      HOST Side
              .RESET_N(KEY[0]),
              .CLK(sdram_ctrl_clk),

              // FIFO Write Side 1
              .WR1_DATA({1'b0,bDISP_G[11:7],bDISP_B[11:2]}),
              .WR1(bCCD_DVAL),
              .WR1_ADDR(0),
              .WR1_MAX_ADDR(640*480/2),
              .WR1_LENGTH(8'h50),
              .WR1_LOAD(!DLY_RST_0),
              .WR1_CLK(~D5M_PIXLCLK),

              // FIFO Write Side 2
              .WR2_DATA({1'b0,bDISP_G[6:2],bDISP_R[11:2]}),
              .WR2(bCCD_DVAL),
              .WR2_ADDR(23'h100000),
              .WR2_MAX_ADDR(23'h100000+640*480/2),
              .WR2_LENGTH(8'h50),
              .WR2_LOAD(!DLY_RST_0),
              .WR2_CLK(~D5M_PIXLCLK),

              // FIFO Read Side 1
              .RD1_DATA(Read_DATA1),
              .RD1(Read),
              .RD1_ADDR(0),
              .RD1_MAX_ADDR(640*480/2),
              .RD1_LENGTH(8'h50),
              .RD1_LOAD(!DLY_RST_0),
              .RD1_CLK(~VGA_CTRL_CLK),

              // FIFO Read Side 2
              .RD2_DATA(Read_DATA2),
              .RD2(Read),
              .RD2_ADDR(23'h100000),
              .RD2_MAX_ADDR(23'h100000+640*480/2),
              .RD2_LENGTH(8'h50),
              .RD2_LOAD(!DLY_RST_0),
              .RD2_CLK(~VGA_CTRL_CLK),

              // SDRAM Side
              .SA(DRAM_ADDR),
              .BA(DRAM_BA),
              .CS_N(DRAM_CS_N),
              .CKE(DRAM_CKE),
              .RAS_N(DRAM_RAS_N),
              .CAS_N(DRAM_CAS_N),
              .WE_N(DRAM_WE_N),
              .DQ(DRAM_DQ),
              .DQM(DRAM_DQM)
                         );

wire [9:0]SD_DATA_R ;
wire [9:0]SD_DATA_G ;
wire [9:0]SD_DATA_B ;

assign SD_DATA_R = Read_DATA2[9:0];
assign SD_DATA_G = {Read_DATA1[14:10],Read_DATA2[14:10]};
assign SD_DATA_B = Read_DATA1[9:0];

//Crop image possition
wire  [15:0]XSTART;
wire  [15:0]XEND;
wire  [15:0]YSTART;
wire  [15:0]YEND;

CROP_YSTART u10(
    .oYSTART(YSTART),
    .iDATA(SD_DATA_R),
    .iCLK(VGA_CTRL_CLK),
    .iRST(DLY_RST_2),
    .iDVAL(Read)
                );

CROP_XSTART u11(
        .oXSTART(XSTART),
        .iDATA(SD_DATA_R),
        .iCLK(VGA_CTRL_CLK),
        .iRST(DLY_RST_2),
        .iDVAL(Read)
                );

CROP_XEND u12(
        .oXEND(XEND),
        .iDATA(SD_DATA_R),
        .iCLK(VGA_CTRL_CLK),
        .iRST(DLY_RST_2),
        .iDVAL(Read)
                );

CROP_YEND u13(
        .oYEND(YEND),
        .iDATA(SD_DATA_R),
        .iCLK(VGA_CTRL_CLK),
        .iRST(DLY_RST_2),
        .iDVAL(Read)
                );


//Image Crop
wire cDVAL;
wire [9:0]cDATA;
wire [15:0]cXSTART;
wire [15:0]cXEND;
wire [15:0]cYSTART;
wire [15:0]cYEND;

assign cXSTART=SW[3]?XSTART:0;
assign cXEND  =SW[3]?XEND  :0;
assign cYSTART=SW[3]?YSTART:0;
assign cYEND  =SW[3]?YEND  :0;

IMAGE_CROP u14(
      .oDVAL(cDVAL),
      .oDATA(cDATA),
      .oDarkCounter(DarkCounter),
      .iXSTART(cXSTART),
      .iXEND(cXEND),
      .iYSTART(cYSTART),
      .iYEND(cYEND),
      .iDATA(SD_DATA_R),
      .iCLK(VGA_CTRL_CLK),
      .iRST(DLY_RST_2),
      .iDVAL(Read)
                );

wire [15:0]DarkCounter;
wire [15:0]LightCounter;

//VGA DISPLAY
wire [9:0]cVGA_R;
wire [9:0]cVGA_G;
wire [9:0]cVGA_B;

assign cVGA_R = SW[4] ? cDATA:SD_DATA_R;
assign cVGA_G = SW[4] ? cDATA:SD_DATA_G;
assign cVGA_B = SW[4] ? cDATA:SD_DATA_B;

VGA_Controller          u16     (       //      Host Side
                                                .oRequest(Read),
                                                .iRed  (cVGA_R),
                                                .iGreen(cVGA_G),
                                                .iBlue (cVGA_B),
                                                //      VGA Side
                                                .oVGA_R(oVGA_R),
                                                .oVGA_G(oVGA_G),
                                                .oVGA_B(oVGA_B),
                                                .oVGA_H_SYNC(VGA_HS),
                                                .oVGA_V_SYNC(VGA_VS),
                                                .oVGA_SYNC(VGA_SYNC_N),
                                                .oVGA_BLANK(VGA_BLANK_N),
                                                //      Control Signal
                                                .iCLK(VGA_CTRL_CLK),
                                                .iRST_N(DLY_RST_2),
                                        );

wire clock1sec;

Clock_Delay     u19     (
              .iCLK(CLOCK2_50),
              .iRST(KEY[0]),
              .oCLK_0(clock1sec),
                        );

wire sound_on;

SOUND_SEL  u20 (
              .oSound_on(sound_on),
              .oDisplayDIG(displayDIG),
              .iRST(KEY[0]),
              .iCLK(clock1sec),
              .iDarkCounter(DarkCounter)
              );

wire [31:0]displayDIG;

SEG7_LUT_8      u7    (
                        .oSEG0(HEX0),
        .oSEG1(HEX1),
        .oSEG2(HEX2),
        .oSEG3(HEX3),
        .iDIG(displayDIG)
```

```
                        );

wire                    I2C_END;
wire                    AUD_CTRL_CLK;


//  TV DECODER ENABLE
assign TD_RESET_N =1'b1;
//  I2C
I2C_AV_Config   u17(    //      Host Side
                .iCLK           (CLOCK_50),
                .iRST_N         ( KEY[0] ),
                .o_I2C_END      ( I2C_END ),
                           //   I2C Side
                .I2C_SCLK       ( I2C_SCLK ),
                .I2C_SDAT       ( I2C_SDAT )
                        );
//      AUDIO SOUND
        assign  AUD_ADCLRCK      =       AUD_DACLRCK;
        assign  AUD_XCK          =       AUD_CTRL_CLK;

// 2CH Audio Sound output -- Audio Generater //
wire AUD_KEY;
assign AUD_KEY=SW[5]?sound_on:0;

        adio_codec ad1  (
                // AUDIO CODEC //
                .oAUD_BCK       ( AUD_BCLK ),
                .oAUD_DATA      ( AUD_DACDAT ),
                .oAUD_LRCK      ( AUD_DACLRCK ),
                .iCLK_18_4      ( AUD_CTRL_CLK ),

                // KEY //
                .iRST_N         ( KEY[0] ),
                .iSrc_Select( 2'b00 ),

                // Sound Control //
                .key1_on        ( AUD_KEY ),//CH1 ON / OFF
                .sound1 ( 533 ),                                  // CH1 Freq
                .instru ( 0 )                                     // Instruction Select
                                        );

endmodule
```

I2C_CCD_Config

```
module I2C_CCD_Config (  //      Host Side
                                iCLK,
                                iRST_N,
                                iZOOM_MODE_SW,
                                iEXPOSURE_ADJ,
                                iEXPOSURE_DEC_p,
                                //      I2C Side
                                I2C_SCLK,
                                I2C_SDAT
                                );
//      Host Side
input           iCLK;
input           iRST_N;
input           iZOOM_MODE_SW;

//      I2C Side
output          I2C_SCLK;
inout           I2C_SDAT;
//      Internal Registers/Wires
reg     [15:0]  mI2C_CLK_DIV;
reg     [31:0]  mI2C_DATA;
reg             mI2C_CTRL_CLK;
reg             mI2C_GO;
```

```
wire            mI2C_END;
wire            mI2C_ACK;
reg     [23:0]  LUT_DATA;
reg     [5:0]   LUT_INDEX;
reg     [3:0]   mSetup_ST;
//////////////   CMOS sensor registers setting /////////////////////
input           iEXPOSURE_ADJ;
input           iEXPOSURE_DEC_p;
parameter       default_exposure        = 16'h0530;
parameter       exposure_change_value   = 16'd200;
reg     [24:0]  combo_cnt;
wire            combo_pulse;
reg     [1:0]   izoom_mode_sw_delay;
reg     [3:0]   iexposure_adj_delay;
wire            exposure_adj_set;
wire            exposure_adj_reset;
reg     [15:0]  senosr_exposure;
wire    [17:0]  senosr_exposure_temp;
wire [23:0] sensor_start_row;
wire [23:0] sensor_start_column;
wire [23:0] sensor_row_size;
wire [23:0] sensor_column_size;
wire [23:0] sensor_row_mode;
wire [23:0] sensor_column_mode;
assign sensor_start_row         = 24'h010036;
assign sensor_start_column      = 24'h020010;
assign sensor_row_size          = 24'h0303BF;
assign sensor_column_size       = 24'h0404FF;
assign sensor_row_mode          = 24'h220000;
assign sensor_column_mode       = 24'h230000;

always@(posedge iCLK or negedge iRST_N)
        begin
                if (!iRST_N)
                        begin
                                iexposure_adj_delay <= 0;
                        end
                else
                        begin
                                iexposure_adj_delay <= {iexposure_adj_delay[2:0],iEXPOSURE_ADJ};
                        end
        end

assign  exposure_adj_set = ({iexposure_adj_delay[0],iEXPOSURE_ADJ}==2'b10) ? 1 : 0 ;
assign  exposure_adj_reset = ({iexposure_adj_delay[3:2]}==2'b10) ? 1 : 0 ;
assign  senosr_exposure_temp  =  iEXPOSURE_DEC_p  ?(senosr_exposure  -  exposure_change_value)  :
(senosr_exposure + exposure_change_value);

always@(posedge iCLK or negedge iRST_N)
        begin
                if (!iRST_N)
                        senosr_exposure <= default_exposure;
                else if (exposure_adj_set|combo_pulse)
                        if (senosr_exposure_temp[17])
                                senosr_exposure <= 0;
                        else if (senosr_exposure_temp[16])
                                senosr_exposure <= 16'hffff;
                        else
                                senosr_exposure <= senosr_exposure_temp[15:0];
        end

always@(posedge iCLK or negedge iRST_N)
        begin
                if (!iRST_N)
```

```
                        combo_cnt <= 0;
                else if (!iexposure_adj_delay[3])
                        combo_cnt <= combo_cnt + 1;
                else
                        combo_cnt <= 0;
        end

assign combo_pulse = (combo_cnt == 25'h1fffff) ? 1 : 0;
wire    i2c_reset;
assign i2c_reset = iRST_N & ~exposure_adj_reset & ~combo_pulse ;
///////////////////////////////////////////////////////////////////
//      Clock Setting
parameter       CLK_Freq=       25000000;       //      25      MHz
parameter       I2C_Freq=       20000;          //      20      KHz
//      LUT Data Number
parameter       LUT_SIZE=       25;

////////////////////    I2C Control Clock       ///////////////////////
always@(posedge iCLK or negedge i2c_reset)
begin
        if(!i2c_reset)
        begin
                mI2C_CTRL_CLK   <=      0;
                mI2C_CLK_DIV    <=      0;
        end
        else
        begin
                if( mI2C_CLK_DIV< (CLK_Freq/I2C_Freq) )
                mI2C_CLK_DIV    <=      mI2C_CLK_DIV+1;
                else
                begin
                        mI2C_CLK_DIV    <=      0;
                        mI2C_CTRL_CLK   <=      ~mI2C_CTRL_CLK;
                end
        end
end
//////////////////////////////////////////////////////////////////
I2C_Controller  u0      (       .CLOCK(mI2C_CTRL_CLK),          //      Controller Work Clock
                                                .I2C_SCLK(I2C_SCLK),            //      I2C CLOCK
                                                .I2C_SDAT(I2C_SDAT),            //      I2C DATA
                                                .I2C_DATA(mI2C_DATA),           //
        DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
                                                .GO(mI2C_GO),                           //      GO
transfor
                                                .END(mI2C_END),                         //      END
transfor
                                                .ACK(mI2C_ACK),                         //      ACK
                                                .RESET(i2c_reset)
                                        );
//////////////////////////////////////////////////////////////////
/////////////////////   Config Control  ///////////////////////////
//always@(posedge mI2C_CTRL_CLK or negedge iRST_N)
always@(posedge mI2C_CTRL_CLK or negedge i2c_reset)
begin
        if(!i2c_reset)
        begin
                LUT_INDEX       <=      0;
                mSetup_ST       <=      0;
                mI2C_GO         <=      0;
        end
        else if(LUT_INDEX<LUT_SIZE)
                begin
                        case(mSetup_ST)
                        0:      begin
```

```
                                        mI2C_DATA       <=      {8'hBA,LUT_DATA};
                                        mI2C_GO         <=      1;
                                        mSetup_ST       <=      1;
                                end
                        1:      begin
                                        if(mI2C_END)
                                        begin
                                                if(!mI2C_ACK)
                                                mSetup_ST       <=      2;
                                                else
                                                mSetup_ST       <=      0;

                                                mI2C_GO         <=      0;
                                        end
                                end
                        2:      begin
                                        LUT_INDEX       <=      LUT_INDEX+1;
                                        mSetup_ST       <=      0;
                                end
                        endcase
                end
end
///////////////////////////////////////////////////////////////////
/////////////////////   Config Data LUT   /////////////////////////
always
begin
case(LUT_INDEX)
        0       :       LUT_DATA<=      24'h000000;
        1       :       LUT_DATA<=      24'h208000;             //      Mirror Row and Columns
        2       :       LUT_DATA<=      {8'h09,senosr_exposure}; //      Exposure
        3       :       LUT_DATA<=      24'h050000;             //      H_Blanking
        4       :       LUT_DATA<=      24'h060019;             //      V_Blanking
        5       :       LUT_DATA<=      24'h0A8000;             //      change latch
        6       :       LUT_DATA<=      24'h2B1032;             //      Green 1 Gain
        7       :       LUT_DATA<=      24'h2C1334;             //      Blue Gain
        8       :       LUT_DATA<=      24'h2D1338;             //      Red Gain
        9       :       LUT_DATA<=      24'h2E1032;     //      Green 2 Gain
        10      :       LUT_DATA<=      24'h100051;     //      set up PLL power on
        11      :       LUT_DATA<=      24'h111f04;     //      PLL_m_Factor<<8+PLL_n_Divider
        12      :       LUT_DATA<=      24'h120001;     //      PLL_p1_Divider
        13      :       LUT_DATA<=      24'h100053;     //      set USE PLL
        14      :       LUT_DATA<=      24'h980000;     //      disble calibration
        15      :       LUT_DATA<=      24'hA00000;     //      Test pattern control
        16      :       LUT_DATA<=      24'hA10000;     //      Test green pattern value
        17      :       LUT_DATA<=      24'hA20FFF;     //      Test red pattern value
        18      :       LUT_DATA<=      sensor_start_row ;      //      set start row
        19      :       LUT_DATA<=      sensor_start_column ;   //      set start column
        20      :       LUT_DATA<=      sensor_row_size;        //      set row size
        21      :       LUT_DATA<=      sensor_column_size;     //      set column size
        22      :       LUT_DATA<=      sensor_row_mode;        //      set row mode in bin mode
        23      :       LUT_DATA<=      sensor_column_mode;     //      set column mode  in bin mode
        24      :       LUT_DATA<=      24'h4901A8;             //      row black target
        default:LUT_DATA<=      24'h000000;
        endcase
end
endmodule
```

I2C_Controller

```
module I2C_Controller (
        CLOCK,
        I2C_SCLK,//I2C CLOCK
        I2C_SDAT,//I2C DATA
        I2C_DATA,//DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
        GO,      //GO transfor
        END,     //END transfor
        ACK,      //ACK
        RESET

);
        input  CLOCK;
        input  [31:0]I2C_DATA;
        input  GO;
        input  RESET;
        inout  I2C_SDAT;
        output I2C_SCLK;
        output END;
        output ACK;

reg SDO;
reg SCLK;
reg END;
reg [31:0]SD;
reg [6:0]SD_COUNTER;

wire I2C_SCLK=SCLK | ( ((SD_COUNTER >= 4) & (SD_COUNTER <=39))? ~CLOCK :0 );
wire I2C_SDAT=SDO?1'bz:0 ;

reg ACK1,ACK2,ACK3,ACK4;
wire ACK=ACK1 | ACK2 |ACK3 |ACK4;

//--I2C COUNTER
always @(negedge RESET or posedge CLOCK ) begin
if (!RESET) SD_COUNTER=6'b111111;
else begin
if (GO==0)
        SD_COUNTER=0;
        else
        if (SD_COUNTER < 41) SD_COUNTER=SD_COUNTER+1;
end
end
//----

always @(negedge RESET or  posedge CLOCK ) begin
if (!RESET) begin SCLK=1;SDO=1; ACK1=0;ACK2=0;ACK3=0;ACK4=0; END=1; end
else
case (SD_COUNTER)
        6'd0  : begin ACK1=0 ;ACK2=0 ;ACK3=0 ;ACK4=0 ; END=0; SDO=1; SCLK=1;end
        //start
        6'd1  : begin SD=I2C_DATA;SDO=0;end
        6'd2  : SCLK=0;
        //SLAVE ADDR
        6'd3  : SDO=SD[31];
        6'd4  : SDO=SD[30];
        6'd5  : SDO=SD[29];
        6'd6  : SDO=SD[28];
        6'd7  : SDO=SD[27];
        6'd8  : SDO=SD[26];
        6'd9  : SDO=SD[25];
        6'd10 : SDO=SD[24];
        6'd11 : SDO=1'b1;//ACK
        //SUB ADDR
        6'd12  : begin SDO=SD[23]; ACK1=I2C_SDAT; end
        6'd13  : SDO=SD[22];
        6'd14  : SDO=SD[21];
        6'd15  : SDO=SD[20];
        6'd16  : SDO=SD[19];
        6'd17  : SDO=SD[18];
        6'd18  : SDO=SD[17];
        6'd19  : SDO=SD[16];
        6'd20  : SDO=1'b1;//ACK
        //DATA
        6'd21  : begin SDO=SD[15]; ACK2=I2C_SDAT; end
        6'd22  : SDO=SD[14];
        6'd23  : SDO=SD[13];
        6'd24  : SDO=SD[12];
        6'd25  : SDO=SD[11];
        6'd26  : SDO=SD[10];
        6'd27  : SDO=SD[9];
        6'd28  : SDO=SD[8];
        6'd29  : SDO=1'b1;//ACK
        //DATA
        6'd30  : begin SDO=SD[7]; ACK3=I2C_SDAT; end
        6'd31  : SDO=SD[6];
        6'd32  : SDO=SD[5];
        6'd33  : SDO=SD[4];
        6'd34  : SDO=SD[3];
        6'd35  : SDO=SD[2];
        6'd36  : SDO=SD[1];
        6'd37  : SDO=SD[0];
        6'd38  : SDO=1'b1;//ACK
        //stop
    6'd39 : begin SDO=1'b0;      SCLK=1'b0; ACK4=I2C_SDAT; end
    6'd40 : SCLK=1'b1;
    6'd41 : begin SDO=1'b1; END=1; end

endcase
end

endmodule
```

CCD_Capture

```
module CCD_Capture(      oDATA,
                                        oDVAL,
                                        oX_Cont,
                                        oY_Cont,
                                        oFrame_Cont,
                                        iDATA,
                                        iFVAL,
                                        iLVAL,
                                        iSTART,
                                        iEND,
                                        iCLK,
                                        iRST
                                        );

input   [11:0]  iDATA;
input           iFVAL;
input           iLVAL;
input           iSTART;
input           iEND;
input           iCLK;
input           iRST;
output  [11:0]  oDATA;
output  [15:0]  oX_Cont;
output  [15:0]  oY_Cont;
output  [31:0]  oFrame_Cont;
output          oDVAL;
reg                     Pre_FVAL;
reg                     mCCD_FVAL;
reg                     mCCD_LVAL;
reg             [11:0]  mCCD_DATA;
reg             [15:0]  X_Cont;
reg             [15:0]  Y_Cont;
reg             [31:0]  Frame_Cont;
reg                             mSTART;

parameter COLUMN_WIDTH = 1280;


assign  oX_Cont         =       X_Cont;
assign  oY_Cont         =       Y_Cont;
assign  oFrame_Cont     =       Frame_Cont;
assign  oDATA           =       mCCD_DATA;
assign  oDVAL           =       mCCD_FVAL&mCCD_LVAL;

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        mSTART   <=     0;
        else
        begin
                if(iSTART)
                mSTART   <=     1;
                if(iEND)
                mSTART   <=     0;
        end
end

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        begin
                Pre_FVAL<=      0;
                mCCD_FVAL       <=      0;
                mCCD_LVAL       <=      0;
                X_Cont          <=      0;
                Y_Cont          <=      0;
        end
        else
        begin
                Pre_FVAL<=      iFVAL;
                if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
                mCCD_FVAL       <=      1;
                else if({Pre_FVAL,iFVAL}==2'b10)
                mCCD_FVAL       <=      0;
                mCCD_LVAL       <=      iLVAL;
                if(mCCD_FVAL)
                begin
                        if(mCCD_LVAL)
                        begin
                                if(X_Cont<(COLUMN_WIDTH-1))
                                X_Cont  <=      X_Cont+1;
                                else
                                begin
                                        X_Cont  <=      0;
                                        Y_Cont  <=      Y_Cont+1;
                                end
                        end
                end
                else
                begin
                        X_Cont  <=      0;
                        Y_Cont  <=      0;
                end
        end
end

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        Frame_Cont      <=      0;
        else
        begin
                if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
                Frame_Cont      <=      Frame_Cont+1;
        end
end

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
                mCCD_DATA       <=      0;
        else if (iLVAL)
                mCCD_DATA       <=      iDATA;
        else
                mCCD_DATA       <=      0;
end

reg     ifval_dealy;

wire ifval_fedge;
reg     [15:0]  y_cnt_d;


always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
                y_cnt_d <=      0;
        else
                y_cnt_d <=      Y_Cont;
end


always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
                ifval_dealy     <=      0;
        else
                ifval_dealy     <=      iFVAL;
end

assign ifval_fedge = ({ifval_dealy,iFVAL}==2'b10)?1:0;

endmodule
```

RAW2RGB

```
module RAW2RGB(
          oRed,
          oGreen,
          oBlue,
          oDVAL,
          iX_Cont,
          iY_Cont,
          iDATA,
          iDVAL,
          iCLK,
          iRST
          );

input   [10:0]  iX_Cont;
input   [10:0]  iY_Cont;
input   [11:0]  iDATA;
input           iDVAL;
input           iCLK;
input           iRST;
output  [11:0]  oRed;
output  [11:0]  oGreen;
output  [11:0]  oBlue;
output          oDVAL;
wire    [11:0]  mDATA_0;
wire    [11:0]  mDATA_1;
reg     [11:0]  mDATAd_0;
reg     [11:0]  mDATAd_1;
reg     [11:0]  mCCD_R;
reg     [12:0]  mCCD_G;
reg     [11:0]  mCCD_B;
reg             mDVAL;

assign  oRed    =       mCCD_R[11:0];
assign  oGreen  =       mCCD_G[12:1];
assign  oBlue   =       mCCD_B[11:0];
assign  oDVAL   =       mDVAL;

Line_Buffer1    u0      (       .clken(iDVAL),
                                .clock(iCLK),
                                .shiftin(iDATA),
                                .taps0x(mDATA_1),
                                .taps1x(mDATA_0) );

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        begin
                mCCD_R  <=      0;
                mCCD_G  <=      0;
                mCCD_B  <=      0;
                mDATAd_0<=      0;
                mDATAd_1<=      0;
                mDVAL   <=      0;
        end
        else
        begin
                mDATAd_0<=      mDATA_0;
                mDATAd_1<=      mDATA_1;
                mDVAL           <=      {iY_Cont[0]|iX_Cont[0]}  ?       1'b0    :       iDVAL;
                if({iY_Cont[0],iX_Cont[0]}==2'b10)
                begin
                        mCCD_R  <=      mDATA_0;
                        mCCD_G  <=      mDATAd_0+mDATA_1;
                        mCCD_B  <=      mDATAd_1;
                end
                else if({iY_Cont[0],iX_Cont[0]}==2'b11)
                begin
                        mCCD_R  <=      mDATAd_0;
                        mCCD_G  <=      mDATA_0+mDATAd_1;
                        mCCD_B  <=      mDATA_1;
                end
                else if({iY_Cont[0],iX_Cont[0]}==2'b00)
                begin
                        mCCD_R  <=      mDATA_1;
                        mCCD_G  <=      mDATA_0+mDATAd_1;
                        mCCD_B  <=      mDATAd_0;
                end
                else if({iY_Cont[0],iX_Cont[0]}==2'b01)
                begin
                        mCCD_R  <=      mDATAd_1;
                        mCCD_G  <=      mDATAd_0+mDATA_1;
                        mCCD_B  <=      mDATA_0;
                end
        end
end

endmodule
```

RGB2GRAY

```
module RGB2GRAY(
        oDVAL,
        oDATA,
        iRed,
        iGreen,
        iBlue,
        iCLK,
        iRST,
        iDVAL,
                );
input                   iDVAL;
input                   iCLK;
input                   iRST;
input           [11:0]  iRed;
input           [11:0]  iGreen;
input           [11:0]  iBlue;
output reg      [11:0]  oDATA;
output reg              oDVAL;

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
                begin
                        oDVAL   <= 0;
                        oDATA   <= 12'b0;
                end
        else
   begin
    oDVAL <= iDVAL;
    oDATA <= (iRed+iGreen+iBlue)/3;
   end
end
endmodule
```

GRAY2BINARY

```
module GRAY2BINARY(
        oDVAL,
        oDATA,
        iDATA,
        iCLK,
        iRST,
        iDVAL,
                );
input           iDVAL;
input           iCLK;
input           iRST;
input           [11:0] iDATA;
output reg      [11:0] oDATA;
output reg      oDVAL;
```

```
parameter  threshold  =  12'd3003;//increase  to  get  more  dark;  home  =>952  //D210  or  D203A=>2547
(night)//D203A =>

always@(posedge iCLK or negedge iRST)
begin
  if(!iRST)
    begin
      oDVAL   <= 0;
      oDATA   <= 0;
    end
  else
    begin
      oDVAL <= iDVAL;
      if(iDATA > threshold) begin
        oDATA <= 12'd4095;
      end
      else begin
        oDATA <= 0;
      end

    end
end
endmodule
```

CROP_YSTART

```
module CROP_YSTART(
        oYSTART,
        iDATA,
        iCLK,
        iRST,
        iDVAL
                );

input       iDVAL;
input       iCLK;
input       iRST;
input       [9:0] iDATA;
output reg [15:0]oYSTART;

reg             [15:0]  X_Cont;
reg             [15:0]  Y_Cont;

always@(posedge iCLK or negedge iRST)
begin
  if(!iRST)begin
    Y_Cont <= 0;
    X_Cont <= 0;
    oYSTART<= 0;
  end
  else begin
    if(iDVAL)begin
      if(Y_Cont<480)begin
        if(X_Cont<640)begin
          if( X_Cont>160 && X_Cont<480 && Y_Cont >50 && Y_Cont <120)begin
            if(iDATA==0) begin
              oYSTART = Y_Cont;
            end
          end
          X_Cont = X_Cont + 1;
        end
        if(X_Cont == 640) begin
          X_Cont = 0;
          Y_Cont = Y_Cont + 1;
        end
      end
      if(Y_Cont == 480) begin
        X_Cont = 0;
        Y_Cont = 0;
      end
    end
  end
end

endmodule
```

CROP_XSTART

```
module CROP_XSTART(
        oXSTART,
        iDATA,
        iCLK,
        iRST,
        iDVAL
                );

input       iDVAL;
input       iCLK;
input       iRST;
input       [9:0] iDATA;
output reg [15:0]oXSTART;

reg             [15:0]  X_Cont;
reg             [15:0]  Y_Cont;
reg   [15:0]  minXSTART;
reg   rowFirstVal;

always@(posedge iCLK or negedge iRST)
begin
  if(!iRST)begin
    Y_Cont      <= 0;
    X_Cont      <= 0;
    oXSTART     <= 0;
    minXSTART   <= 16'd479;
    rowFirstVal <= 1;
  end
  else begin
    if(iDVAL)begin
      if(Y_Cont<480)begin
        if(X_Cont<640)begin
          if( X_Cont>160 && X_Cont<480 && Y_Cont >50 && Y_Cont <120)begin
            if(iDATA==0 && rowFirstVal)begin
              if(X_Cont<minXSTART)begin
                minXSTART=X_Cont;
              end
              else begin
                minXSTART=minXSTART;
              end
              rowFirstVal=0;
            end
            else begin
              minXSTART=minXSTART;
            end
          end
          X_Cont = X_Cont + 1;
        end
        if(X_Cont == 640) begin
```

```
          X_Cont = 0;
          Y_Cont = Y_Cont + 1;
          rowFirstVal=1;
        end
      end
      if(Y_Cont == 480) begin
        oXSTART=minXSTART;
        X_Cont = 0;
        Y_Cont = 0;
        minXSTART=16'd479;
      end
    end
  end
end

endmodule
```

CROP_YEND

```
module CROP_YEND(
        oYEND,
        iDATA,
        iCLK,
        iRST,
        iDVAL
                );

input       iDVAL;
input       iCLK;
input       iRST;
input       [9:0] iDATA;
output reg [15:0]oYEND;

reg             [15:0]  X_Cont;
reg             [15:0]  Y_Cont;
reg   [15:0]  maxYEND;

always@(posedge iCLK or negedge iRST)
begin
  if(!iRST)begin
    Y_Cont      <= 0;
    X_Cont      <= 0;
    oYEND       <= 0;
    maxYEND     <= 0;
  end
  else begin
    if(iDVAL)begin
      if(Y_Cont<480)begin
        if(X_Cont<640)begin
          if( X_Cont>160 && X_Cont<480 && Y_Cont >50 && Y_Cont <120)begin
            if(iDATA==0)begin
              if(Y_Cont>maxYEND)begin
                maxYEND=Y_Cont;
              end
              else begin
                maxYEND=maxYEND;
              end
            end
            else begin
              maxYEND=maxYEND;
            end
          end
          X_Cont = X_Cont + 1;
        end
        if(X_Cont == 640) begin
          X_Cont = 0;
          Y_Cont = Y_Cont + 1;
        end
      end
      if(Y_Cont == 480) begin
        oYEND=maxYEND;
        X_Cont = 0;
        Y_Cont = 0;
        maxYEND= 0;
      end
    end
  end
end

endmodule
```

CROP_XEND

```
module CROP_XEND(
        oXEND,
        iDATA,
        iCLK,
        iRST,
        iDVAL
                );

input       iDVAL;
input       iCLK;
input       iRST;
input       [9:0] iDATA;
output reg [15:0]oXEND;

reg             [15:0]  X_Cont;
reg             [15:0]  Y_Cont;
reg   [15:0]  maxXEND;

always@(posedge iCLK or negedge iRST)
begin
  if(!iRST)begin
    Y_Cont      <= 0;
    X_Cont      <= 0;
    oXEND       <= 0;
    maxXEND     <= 0;
  end
  else begin
    if(iDVAL)begin
      if(Y_Cont<480)begin
        if(X_Cont<640)begin
          if( X_Cont>160 && X_Cont<480 && Y_Cont >50 && Y_Cont <120)begin
            if(iDATA==0)begin
              if(X_Cont>maxXEND)begin
                maxXEND=X_Cont;
              end
              else begin
                maxXEND=maxXEND;
              end
            end
            else begin
              maxXEND=maxXEND;
            end
          end
          X_Cont = X_Cont + 1;
        end
        if(X_Cont == 640) begin
          X_Cont = 0;
          Y_Cont = Y_Cont + 1;
        end
      end
      if(Y_Cont == 480) begin
        oXEND=maxXEND;
        X_Cont = 0;
        Y_Cont = 0;
        maxXEND= 0;
      end
    end
  end
end

endmodule
```

IMAGE_CROP

```
module IMAGE_CROP(
        oDVAL,
        oDATA,
        oDarkCounter,
        oLightCounter,
        iXSTART,
        iXEND,
        iYSTART,
        iYEND,
        iDATA,
        iCLK,
        iRST,
        iDVAL
                );

input    iDVAL;
input    iCLK;
input    iRST;
input    [9:0] iDATA;
input    [15:0]  iXSTART;
input    [15:0]  iXEND;
input    [15:0]  iYSTART;
input    [15:0]  iYEND;
output reg  [9:0] oDATA;
output reg      oDVAL;
output reg   [15:0] oDarkCounter;
output reg   [15:0] oLightCounter;


reg             [15:0]  X_Cont;
reg             [15:0]  Y_Cont;
reg             [15:0]  DarkCounterTemp;

always@(posedge iCLK or negedge iRST)
begin
  if(!iRST)begin
    Y_Cont <= 0;
    X_Cont <= 0;
    oDVAL  <= 0;
    oDATA  <= 0;
    oDarkCounter <=0;
    oLightCounter <=0;
    DarkCounterTemp <=0;
  end
  else begin
```

```
    oDVAL <= iDVAL;
    if(iDVAL)begin
      if(Y_Cont<480)begin
        if(X_Cont<640)begin
          if( X_Cont<160     || X_Cont>480   ||
              Y_Cont<50      || Y_Cont>120   ||
              Y_Cont<iYSTART || Y_Cont>iYEND ||
              X_Cont<iXSTART || X_Cont>iXEND )begin
            oDATA = 0;
          end
          else begin
            oDATA = iDATA;
            if(oDATA!=0) begin
              oLightCounter=oLightCounter+1;
            end
            else begin
              DarkCounterTemp=DarkCounterTemp+1;
              oDarkCounter=DarkCounterTemp;
            end
          end
          X_Cont = X_Cont + 1;
        end
        if(X_Cont == 640) begin
          X_Cont = 0;
          Y_Cont = Y_Cont + 1;
        end
      end
      if(Y_Cont == 480) begin
        DarkCounterTemp  =0;
        oLightCounter =0;
        X_Cont = 0;
        Y_Cont = 0;
      end
    end
  end
end
endmodule
```

VGA_Controller

```
module  VGA_Controller( //      Host Side
                        iRed,
                        iGreen,
                        iBlue,
                        oRequest,
                        //      VGA Side
                        oVGA_R,
                        oVGA_G,
                        oVGA_B,
                        oVGA_H_SYNC,
                        oVGA_V_SYNC,
                        oVGA_SYNC,
                        oVGA_BLANK,
                        //      Control Signal
                        iCLK,
                        iRST_N,
                        );

//      Horizontal Parameter    ( Pixel )
parameter       H_SYNC_CYC      =       96;
parameter       H_SYNC_BACK     =       48;
parameter       H_SYNC_ACT      =       640;
parameter       H_SYNC_FRONT=   16;
parameter       H_SYNC_TOTAL=   800;
```

```
//      Virtical Parameter              ( Line )
parameter       V_SYNC_CYC      =       2;
parameter       V_SYNC_BACK     =       33;
parameter       V_SYNC_ACT      =       480;
parameter       V_SYNC_FRONT=   10;
parameter       V_SYNC_TOTAL=   525;

//      Start Offset
parameter       X_START         =       H_SYNC_CYC+H_SYNC_BACK;
parameter       Y_START         =       V_SYNC_CYC+V_SYNC_BACK;
//      Host Side
input           [9:0]   iRed;
input           [9:0]   iGreen;
input           [9:0]   iBlue;
output  reg             oRequest;
//      VGA Side
output  reg     [9:0]   oVGA_R;
output  reg     [9:0]   oVGA_G;
output  reg     [9:0]   oVGA_B;
output  reg             oVGA_H_SYNC;
output  reg             oVGA_V_SYNC;
output  reg             oVGA_SYNC;
output  reg             oVGA_BLANK;

wire            [9:0]   mVGA_R;
wire            [9:0]   mVGA_G;
wire            [9:0]   mVGA_B;
reg                     mVGA_H_SYNC;
reg                     mVGA_V_SYNC;
wire                    mVGA_SYNC;
wire                    mVGA_BLANK;

//      Control Signal
input                   iCLK;
input                   iRST_N;

//      Internal Registers and Wires
reg             [12:0]  H_Cont;
reg             [12:0]  V_Cont;

wire    [12:0]          v_mask;
assign v_mask = 13'd0 ;

////////////////////////////////////////////////////////
assign  mVGA_BLANK      =       mVGA_H_SYNC & mVGA_V_SYNC;
assign  mVGA_SYNC       =       1'b0;

assign  mVGA_R  =       (       H_Cont>=X_START  && H_Cont<X_START+H_SYNC_ACT &&
                                                 V_Cont>=Y_START+v_mask  && V_Cont<Y_START+V_SYNC_ACT )
                                                 ?       iRed    :       0;
assign  mVGA_G  =       (       H_Cont>=X_START  && H_Cont<X_START+H_SYNC_ACT &&
                                                 V_Cont>=Y_START+v_mask  && V_Cont<Y_START+V_SYNC_ACT )
                                                 ?       iGreen  :       0;
assign  mVGA_B  =       (       H_Cont>=X_START  && H_Cont<X_START+H_SYNC_ACT &&
                                                 V_Cont>=Y_START+v_mask  && V_Cont<Y_START+V_SYNC_ACT )
                                                 ?       iBlue   :       0;

always@(posedge iCLK or negedge iRST_N)
        begin
                if (!iRST_N)
                        begin
                                oVGA_R <= 0;
                                oVGA_G <= 0;
        oVGA_B <= 0;
```

```
                                oVGA_BLANK <= 0;
                                oVGA_SYNC <= 0;
                                oVGA_H_SYNC <= 0;
                                oVGA_V_SYNC <= 0;
                        end
                else
                        begin
                                oVGA_R <= mVGA_R;
                                oVGA_G <= mVGA_G;
        oVGA_B <= mVGA_B;
                                oVGA_BLANK <= mVGA_BLANK;
                                oVGA_SYNC <= mVGA_SYNC;
                                oVGA_H_SYNC <= mVGA_H_SYNC;
                                oVGA_V_SYNC <= mVGA_V_SYNC;
                        end
        end

//      Pixel LUT Address Generator
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        oRequest<=      0;
        else
        begin
                if(     H_Cont>=X_START-2 && H_Cont<X_START+H_SYNC_ACT-2 &&
                        V_Cont>=Y_START && V_Cont<Y_START+V_SYNC_ACT )begin
      oRequest  <=      1;
    end
                else
                oRequest<=      0;
        end
end

//      H_Sync Generator, Ref. 40 MHz Clock
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                H_Cont          <=      0;
                mVGA_H_SYNC     <=      0;
        end
        else
        begin
                //      H_Sync Counter
                if( H_Cont < H_SYNC_TOTAL )begin
                H_Cont  <=      H_Cont+1;
                end
                else
                H_Cont  <=      0;
                //      H_Sync Generator
                if( H_Cont < H_SYNC_CYC )
                mVGA_H_SYNC     <=      0;
                else
                mVGA_H_SYNC     <=      1;
        end
end

//      V_Sync Generator, Ref. H_Sync
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                V_Cont          <=      0;
                mVGA_V_SYNC     <=      0;
        end
        else
        begin
                //      When H_Sync Re-start
                if(H_Cont==0)
                begin
                        //      V_Sync Counter
                        if( V_Cont < V_SYNC_TOTAL )begin
                        V_Cont  <=      V_Cont+1;
                        end
                        else
                        V_Cont  <=      0;
                        //      V_Sync Generator
                        if(     V_Cont < V_SYNC_CYC )
                        mVGA_V_SYNC     <=      0;
                        else
                        mVGA_V_SYNC     <=      1;
                end
        end
end
endmodule
```

SOUND_SEL

```
module SOUND_SEL(
  oSound_on,
  oDisplayDIG,
  iCLK,
  iRST,
  iDarkCounter
);

input iCLK;
input [15:0]iDarkCounter;
input iRST;
output reg oSound_on;
output reg [31:0]oDisplayDIG;

reg [2:0]delay;

always@(posedge iCLK or negedge iRST) begin
  if(!iRST)begin
    oSound_on<=0;
    oDisplayDIG<=0;
    delay<=0;
  end
  else begin
    oDisplayDIG<=iDarkCounter;
    if(iDarkCounter<16'h0600 || iDarkCounter>16'h4000) begin
      delay<=delay+1;
      if(delay>1) begin
        oSound_on<=1;
      end
      else begin
        oSound_on<=0;
      end
    end
    else begin
      delay<=0;
      oSound_on<=0;
    end
  end
end
endmodule
```

adio_codec:

```
module adio_codec (
output                  oAUD_DATA,
output                  oAUD_LRCK,
output  reg             oAUD_BCK,
input key1_on,
input key2_on,
input key3_on,
input key4_on,
input   [1:0]   iSrc_Select,
input                   iCLK_18_4,
input                   iRST_N,
input   [15:0]  sound1,
input   [15:0]  sound2,
input   [15:0]  sound3,
input   [15:0]  sound4,
input           instru
                                                );
parameter       REF_CLK                 =       18432000;       //      18.432  MHz
parameter       SAMPLE_RATE             =       48000;          //      48              KHz
parameter       DATA_WIDTH              =       16;                     //      16              Bits
parameter       CHANNEL_NUM             =       2;                      //      Dual Channel
parameter       SIN_SAMPLE_DATA =       48;
////////////    Input Source Number     //////////////
parameter       SIN_SANPLE              =       0;
//////////////////////////////////////////////////
//      Internal Registers and Wires
reg             [3:0]   BCK_DIV;
reg             [8:0]   LRCK_1X_DIV;
reg             [7:0]   LRCK_2X_DIV;
reg             [6:0]   LRCK_4X_DIV;
reg             [3:0]   SEL_Cont;
////////DATA Counter    ////////
reg             [5:0]   SIN_Cont;
////////////////////////////////////
reg                                                     LRCK_1X;
reg                                                     LRCK_2X;
reg                                                     LRCK_4X;
////////////    AUD_BCK Generator       //////////////
always@(posedge iCLK_18_4 or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                BCK_DIV         <=      0;
                oAUD_BCK<=      0;
        end
        else
        begin
                if(BCK_DIV >= REF_CLK/(SAMPLE_RATE*DATA_WIDTH*CHANNEL_NUM*2)-1 )
                begin
                        BCK_DIV         <=      0;
                        oAUD_BCK<=      ~oAUD_BCK;
                end
                else
                BCK_DIV         <=      BCK_DIV+1;
        end
end
//////////////////////////////////////////////////
////////////    AUD_LRCK Generator      //////////////
always@(posedge iCLK_18_4 or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                LRCK_1X_DIV     <=      0;
                LRCK_2X_DIV     <=      0;
                LRCK_4X_DIV     <=      0;
                LRCK_1X         <=      0;
                LRCK_2X         <=      0;
                LRCK_4X         <=      0;
        end
        else
        begin
                //      LRCK 1X
                if(LRCK_1X_DIV >= REF_CLK/(SAMPLE_RATE*2)-1 )
                begin
                        LRCK_1X_DIV     <=      0;
                        LRCK_1X <=      ~LRCK_1X;
                end
                else
                LRCK_1X_DIV             <=      LRCK_1X_DIV+1;
                //      LRCK 2X
                if(LRCK_2X_DIV >= REF_CLK/(SAMPLE_RATE*4)-1 )
                begin
                        LRCK_2X_DIV     <=      0;
                        LRCK_2X <=      ~LRCK_2X;
                end
                else
                LRCK_2X_DIV             <=      LRCK_2X_DIV+1;
                //      LRCK 4X
                if(LRCK_4X_DIV >= REF_CLK/(SAMPLE_RATE*8)-1 )
                begin
                        LRCK_4X_DIV     <=      0;
                        LRCK_4X <=      ~LRCK_4X;
                end
                else
                LRCK_4X_DIV             <=      LRCK_4X_DIV+1;
        end
end
assign  oAUD_LRCK       =       LRCK_1X;
/////////////////////////////////////////////////
//////////      Sin LUT ADDR Generator  //////////////
always@(negedge LRCK_1X or negedge iRST_N)
begin
        if(!iRST_N)
        SIN_Cont<=      0;
        else
        begin
                if(SIN_Cont < SIN_SAMPLE_DATA-1 )
                SIN_Cont<=      SIN_Cont+1;
                else
                SIN_Cont<=      0;
        end
end
//////////////////Wave-Source generate////////////////
////////////Timbre selection & SoundOut//////////////
        wire [15:0]music1_ramp;
        wire [15:0]music2_ramp;
        wire [15:0]music1_sin;
        wire [15:0]music2_sin;
        wire [15:0]music3_ramp;
        wire [15:0]music4_ramp;
        wire [15:0]music3_sin;
        wire [15:0]music4_sin;
        wire [15:0]music1=(instru)?music1_ramp:music1_sin;
        wire [15:0]music2=(instru)?music2_ramp:music2_sin;
        wire [15:0]music3=(instru)?music3_ramp:music3_sin;
        wire [15:0]music4=(instru)?music4_ramp:music4_sin;
        wire [15:0]sound_o;
        assign sound_o=music1+music2+music3+music4;
        always@(negedge oAUD_BCK or negedge iRST_N)begin
                if(!iRST_N)
                        SEL_Cont<=      0;
                else
                        SEL_Cont<=      SEL_Cont+1;
        end
        assign  oAUD_DATA       =       ((key4_on|key3_on|key2_on|key1_on)                          &&
(iSrc_Select==SIN_SANPLE))      ?       sound_o[~SEL_Cont]      :0;
//////////Ramp address generater/////////////
        reg  [15:0]ramp1;
        reg  [15:0]ramp2;
        reg  [15:0]ramp3;
        reg  [15:0]ramp4;
        wire [15:0]ramp_max=60000;
//////CH1 Ramp//////
        always@(negedge key1_on or negedge LRCK_1X)begin
        if (!key1_on)
                ramp1=0;
        else if (ramp1>ramp_max) ramp1=0;
        else ramp1=ramp1+sound1;
        end
//////CH2 Ramp//////
        always@(negedge key2_on or negedge LRCK_1X)begin
        if (!key2_on)
                ramp2=0;
        else if (ramp2>ramp_max) ramp2=0;
        else ramp2=ramp2+sound2;
        end
//////CH3 Ramp/////
        always@(negedge key3_on or negedge LRCK_1X)begin
        if (!key3_on)
                ramp3=0;
        else if (ramp3>ramp_max) ramp3=0;
        else ramp3=ramp3+sound3;
        end
//////CH3 Ramp/////
        always@(negedge key4_on or negedge LRCK_1X)begin
        if (!key4_on)
                ramp4=0;
        else if (ramp4>ramp_max) ramp4=0;
        else ramp4=ramp4+sound4;
        end
////////////Ramp address assign/////////////
        wire [5:0]ramp1_ramp=(instru)?ramp1[15:10]:0;
        wire [5:0]ramp2_ramp=(instru)?ramp2[15:10]:0;
        wire [5:0]ramp3_ramp=(instru)?ramp3[15:10]:0;
        wire [5:0]ramp4_ramp=(instru)?ramp4[15:10]:0;
        wire [5:0]ramp1_sin=(!instru)?ramp1[15:10]:0;
        wire [5:0]ramp2_sin=(!instru)?ramp2[15:10]:0;
        wire [5:0]ramp3_sin=(!instru)?ramp3[15:10]:0;
        wire [5:0]ramp4_sin=(!instru)?ramp4[15:10]:0;
////////String-wave Timbre///////
        wave_gen_string r1(
                .ramp(ramp1_ramp),
                .music_o(music1_ramp)
        );
        wave_gen_string r2(
                .ramp(ramp2_ramp),
                .music_o(music2_ramp)
        );
        wave_gen_string r3(
                .ramp(ramp3_ramp),
                .music_o(music3_ramp)
        );
        wave_gen_string r4(
                .ramp(ramp4_ramp),
                .music_o(music4_ramp)
        );

/////////Brass-wave Timbre////////
        wave_gen_brass s1(
                .ramp(ramp1_sin),
                .music_o(music1_sin)
        );
        wave_gen_brass s2(
                .ramp(ramp2_sin),
                .music_o(music2_sin)
        );
        wave_gen_brass s3(
                .ramp(ramp3_sin),
                .music_o(music3_sin)
        );
        wave_gen_brass s4(
                .ramp(ramp4_sin),
                .music_o(music4_sin)
        );

endmodule
```