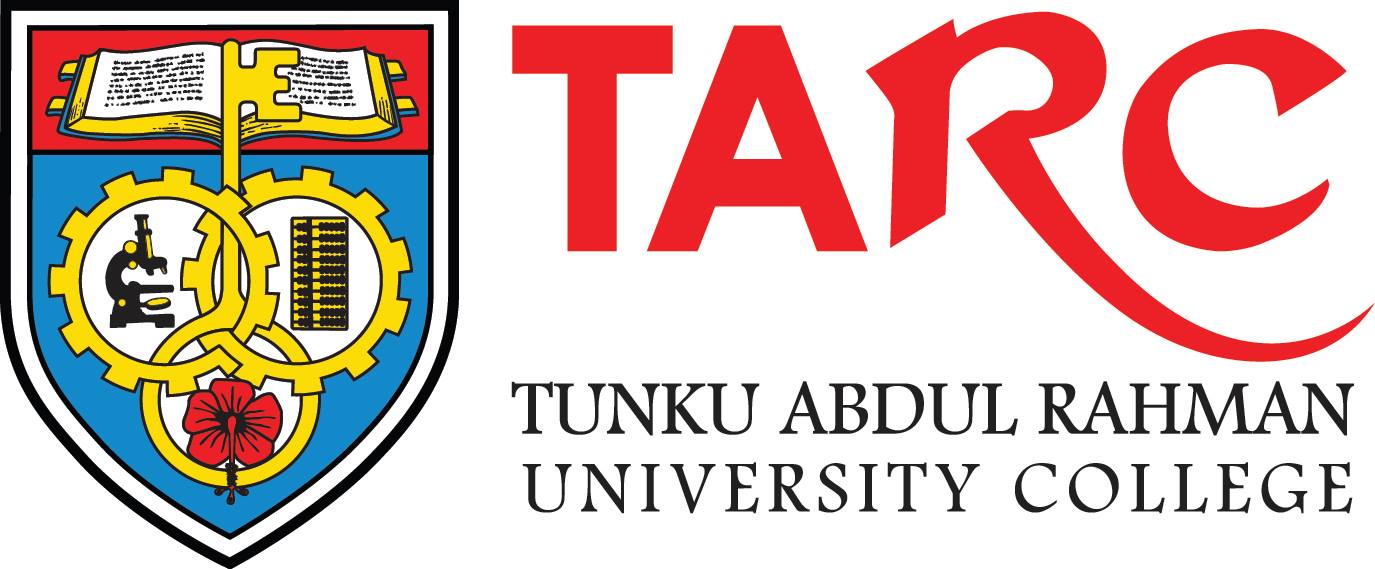
**Development of an FPGA-Based Iris Recognition System:**

**Iris Normalization and Compression**

**By**

**Jason Chuah Kwong Hooi**



**Faculty of Applied Sciences and Computing**

**Tunku Abdul Rahman University College**

**Kuala Lumpur**

**2015/2016**

Final Year Project

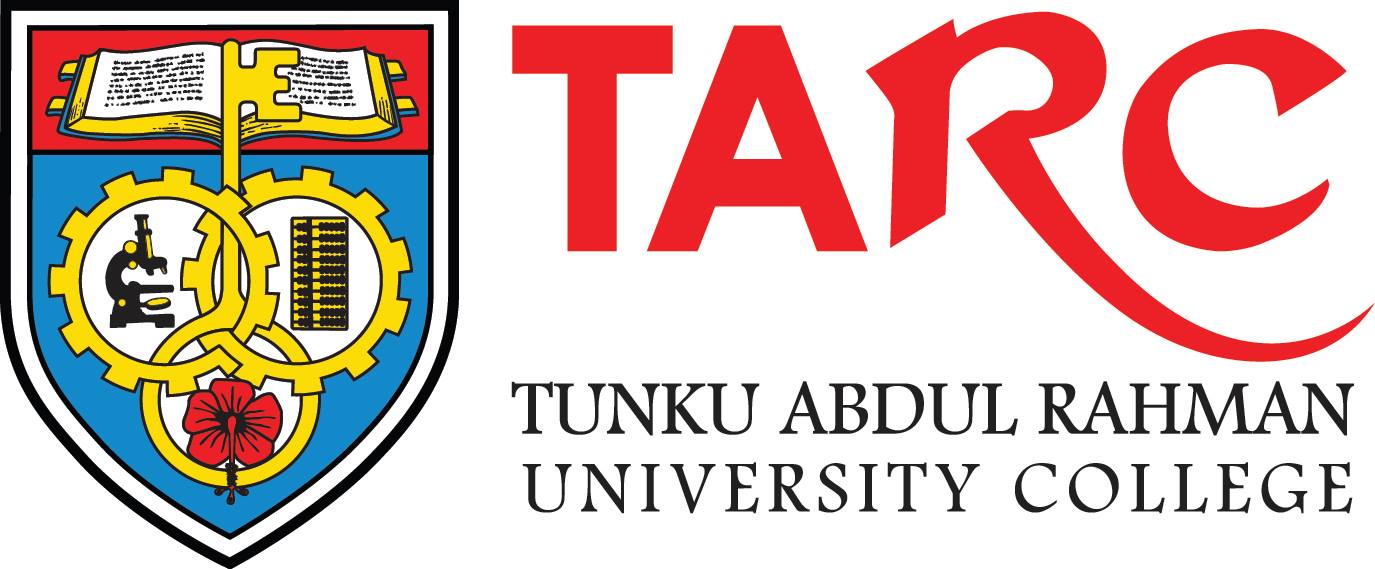
Development of an FPGA based Iris Recognition System:

Iris Normalization and Compression

By

Jason Chuah Kwong Hooi

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.

Department of Physical Science

Faculty of Applied Sciences and Computing

Tunku Abdul Rahman University College

Kuala Lumpur

Contents

[**AKNOWLEDGMENT** 1](#_Toc440362477)

[**ABSTRACT** 2](#_Toc440362478)

[**CHAPTER 1: INTRODUCTION** 3](#_Toc440362479)

[**1.1 Objectives** 3](#_Toc440362480)

[**1.2 Problem Statement** 4](#_Toc440362481)

[**1.3 Background** 6](#_Toc440362482)

[**CHAPTER 2: LITERATURE REVIEW** 10](#_Toc440362483)

[**2.1.2 Image Registration** 11](#_Toc440362484)

[**2.1.3 Virtual Circles** 12](#_Toc440362485)

[**2.2 Conventional Methods for Feature Extraction** 13](#_Toc440362486)

[**2.2.1 Gabor Filters** 13](#_Toc440362487)

[**2.2.2 Wavelet Transform** 16](#_Toc440362488)

[**2.2.3 Discrete Cosine Transform** 17](#_Toc440362489)

[**CHAPTER 3: METHODOLOGY** 18](#_Toc440362490)

[**3.1 Iris Normalization Block** 20](#_Toc440362491)

[**3.2 Compression Block** 26](#_Toc440362492)

[**CHAPTER 4: RESULTS & DISCUSSION** 31](#_Toc440362493)

[**4.2.1: Compression** 35](#_Toc440362494)

[**4.2: Analysis** 37](#_Toc440362495)

[**4.3: Integrated Result** 41](#_Toc440362496)

**[CHAPTER 5: CONCLUSION](#_Toc440362498)** [43](#_Toc440362498)

[**5.1: Conclusion** 43](#_Toc440362499)

[**5.1: Future Trends & Recommendation** 44](#_Toc440362500)

[**REFERENCE** 45](#_Toc440362501)

[**Appendix** 47](#_Toc440362502)

[**Integrated Code: Main module** 47](#_Toc440362503)

# **AKNOWLEDGMENT**

I would like to express my sincere gratitude to my final year project (fyp) supervisor, Miss Michelle Lim Sern Mi for her guidance, encouragement and support during my time in doing fyp. Thanks to her insights and suggestions on theory explanation and technical implementation issues, my fyp was able to be completed on time with a satisfactory outcome before the day of presentation. Next, I would like to thank her for her patience and constructive comments on the weekly meeting up until the final week of presentation. Also, I want to thank her for improving my group presentation skills before the actual presentation date.

Furthermore, I would like to thank my group mates, Yap Ken Mun and Lee Shyan Feng for helping me throughout the entire fyp preparation including their suggestions on improving my sub-system and also throughout the integration sessions.

# **ABSTRACT**

This research focuses on the image post-processing (Iris Normalization) and compression of an Iris Recognition System (IRS). This proposed architecture mainly consists of two parts i.e. iris normalization and compression. The initial step for normalization is to extract a specific region from the segmented iris image which has a resolution of 640x480, then the normalized iris image which is now 120x80 in resolution will be compressed into a binary template before being sent for matching. The proposed normalization extracts a smaller region from the segmented image using Scale Invariant Feature Transform (SIFT) technique. The extracted iris region will then be compressed into a 10x10 matrix which consists of only 1 bit in each element. The aim of this paper is to develop an IRS post processing and compression module using a different method that will resolve issues faced by conventional post-processing blocks particularly in terms of resource usage and speed. This proposed post-processing block introduces region extraction in the lower half of the iris image and Haar Wavelet Transform (HWT) to improve processing time and reduce the resources used in hardware. The proposed IRS will be modeled, designed and simulated in Verilog using Modelsim in Mentor Graphics environment. Lastly, the designed Verilog codes will be integrated and implemented onto the FPGA board for functional verification purposes. The expected output is a 10x10 with 1 bit in each element. The proposed design achieves a moderately high accuracy of 96.67% with a matching possibility of 66.67% if each user’s iris were trained in multiples of 3 as opposed to one trained sample per user.. This proposed research contributes to the security field and the IRS is most likely to be applied in Security Recognition Systems or for verification purposes in banking services.

# **CHAPTER 1: INTRODUCTION**

# **1.1 Objectives**

1. To investigate the architecture of iris processing systems and all sub-blocks related to post processing (iris normalization) and compression for the Iris Recognition System (IRS) from past literature.

2. To model, design and simulate the IRS which involves only the post-processing iris normalization and compression sections and all related sub-blocks using Verilog HDL in Modelsim simulator of Mentor Graphics environment.

3. To analyse, integrate and implement the designed Verilog Codes of the image post-processing (iris normalization) and compression modules into the complete IRS using FPGA for hardware verification purposes.

## **1.2 Problem Statement**

The aim of this research is to address the problems faced by researchers in IRS image post- processing and compression. There are two main processes in this image post processing and compression i.e. the iris normalization and compression. Here two conventional image post-processing and compression methods will be discussed. The first technique for image post-processing and compression is the Wildes’s Image Registration as shown in Figure 1.1. This technique has the advantage of its capability to compensate the unwanted factors such as variations in rotation and scaling (Wildes, 1996). However, this technique is time consuming in identification applications (Wildes, 1996).

The second conventional technique for image post processing and compression is shown in Figure 1.2 to compress the image right after image pre-processing. In this technique, the image post processing step will be skipped. This resulted in the reduction of processing time as well as the reduction in iris signature size (Birgale, 2010). However, without the image post-processing steps, the size of the variation caused by the captured distance to the eye will be huge. Moreover, the unwanted factors such as influence of rotation and translation of scale cannot be compensated (Yang, 2012).

Thus, after comparing the two conventional techniques for IRS image post-processing and compression, the proposed solution is to compensate the size of variation caused by the captured distance to the eye and also overcome the influence of rotation as well as reducing the processing time for image post processing and compression without getting rid of the iris normalization process.

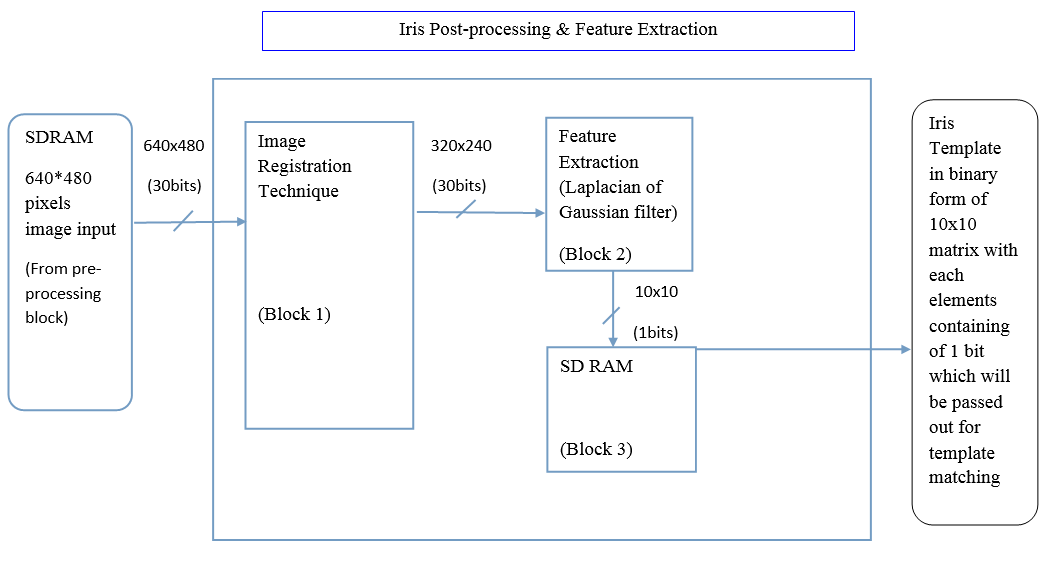


Figure 1.1: Conventional Block Diagram of Wildes’s Image Recognition System (Wildes, 1996)

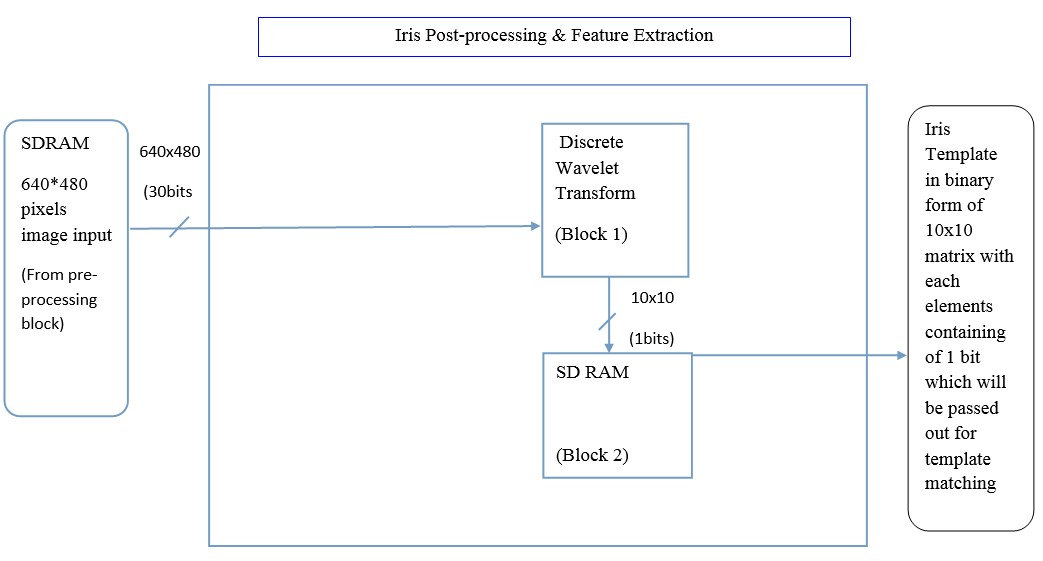


Figure 1.2: Conventional Block Diagram of Birgale’s Image Recognition System (Birgale, 2010)

## **1.3 Background**

Security system is the need of the day. Recognition speed is the current demand. Facial features, voice patterns, hand geometry, retinal patterns, voice recognition and fingerprints have all been explored as biometric identifiers with varying levels of success. However, iris being unique and stable for a life period is the most reliable biometric identifier (Birgale, 2010). Iris as biometric recognition for identification formed the active research area since the 90s (Birgale, 2010). The uniqueness of iris patterns was identified since then. The iris pattern does not change throughout the user’s whole life (Kokare, 2010). It is impossible to modify the iris surgically without any risk of vision damage (Kokare, 2010). In recent years, Iris Biometric Identification systems have found major applications around the globe, it is being used in offices as an entry logging system, in passport offices and at airports to associate the visa details of a person upon arrival, and even to enroll the entire population of a country’s legal residents and immigrants. The sophistication of the system, along with the very low false rejection rate, has made it reach the pinnacle of biometric security systems by being both reliable and secure (Sarin, 2014).

The Iris Recognition System (IRS) in this project is implemented by means of Artificial Neural Network (ANN) for security augmented system, The IRS is separated into three main blocks: The Image Acquisition and Iris Segmentation block, Image Post-Processing and Compression block and the last one is the ANN block for matching purposes. The Image Post-Processing and Compression block will be focused and discussed here. The two main tasks involved are the iris unwrapping and feature extraction. The following paragraphs will explain both tasks in detail.

Firstly, iris unwrapping will begin when the iris region is segmented. The segmented iris image will be normalized to enable the generation of the iris code and their comparisons (Nabti, 2008).The normalization process involve transforming the iris region from Cartesian coordinates to polar coordinates (Daugman 2013). The proposed method is capable of compensating the unwanted variations due to the distance of the eye from the camera and its position with respect to the camera.

Secondly, after the normalized image is obtained, the next step is to encode the image for template matching. This can be done by extracting the important and unique features of the normalized images in feature extraction. This section will filter out the redundant information in the normalized iris and then compress (TSAI, 2007). A 1-D log Gabor Filter will be used to perform these tasks.

Looking at the work of other researchers, different methods were applied and different results were presented. Designed system on CYCLONE II DE2 Board is used by (Hentati, 2010) to complete her research. It resulted in the Hough transform and Hamming distance being implemented in FPGA which showed high performance, reliability and speed. Another research is done by (Jimenez, 2005) using the full hardware solution for processing the iris biometric. His method had introduced the pipeline structure for hardware acceleration and faster processing time. However, his method had a flaw in that the feature extraction performance is worse than software. Besides that, (Grabowski, 2006) used the optimized Iris Recognition Algorithm to perform hardware implementation. His research proposed that iris identification based on 2D discrete wavelet transform can be optimized for embedded system. His research showed that the image acquired in the database does not have any eyelids or eyelashes. This implies that the segmented iris area would not have any redundancy. Moreover, (Buethna, 2012) also used hardware implementation for iris matching.

This method achieved significant reduction in execution time as compared with conventional software based applications. Lastly, the method proposed by Patil is the iris feature extraction and classification using FPGA. The researcher implemented SVD algorithm by hardware architecture to extract the iris feature. This result for SVD value in MATLAB and FPGA is quite close (Patil, 2012).

Among all the literatures listed in Table 1, the method used by Patil (2012) is similar to the proposed IRS. The image processing method is the same in terms of the hardware implementation (Patil, 2012). The application of IRS is the banking system that will be implemented in the future (Chabrow, 2014). In the future, IRS will become dominant in the security system of banking services. This is an important aspect in raising the security level.

Table 1: Past literature on Iris Recognition System

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Researcher  (Year) | Title | Method applied | Results |
| 1 | Hentati (2010) | An Embedded  System for iris  Recognition | Implemented the  designed system on  CYCLONE II DE2  Board using the  NIOSII  Processor(HW/SW  implementation) | Hough transform and Hamming distance are implemented in FPGA which presents high performance, reliability and speed. |
| 2 | Jimenez (2005) | Full Hardware solution for processing iris biometric | Gabor filter is used for  feature extraction &  used pipelined  architectures for  hardware system | This paper introduces pipeline structure for hardware acceleration and reduces processing time by 80% but within the feature extraction block performance is worse than the Software versions.. |
| 3 | Grabowski (2006) | Iris Recognition  Algorithm  Optimized  for Hardware  Implementation | This paper proposed  iris identification  based on 2 D Discrete  Wavelet Transform  which can be  optimized for  embedded systems | The image in this database does not have any eyelids or eyelashes inside the iris area & does not have any disturbances in the iris pattern. |
| 4 | Buethna (2012) | Hardware  Implementation of  Iris Matching | Implemented  matching part of iris  recognition algorithm  on Spartan 3 FPGA | This method achieves significant reduction in execution time as compared to conventional software based applications. |
| 6 | Patil (2012) | Iris Feature  Extraction and  Classification using  FPGA | Implemented SVD algorithm by using FPGA to extract the feature of the iris image. | The result for SVD value in FPGA is very close. The iris is identified in less than 250µs. |
| 7 | This  work (2015) | Iris Normalization and Iris Feature  Extraction  implemented using FPGA | Implemented  SIFT extraction on iris image and HWT to compress the iris image into binary image | The result for threshold value in FPGA is very close. The iris is identified in less than 250µs and the matching rate is expected to reach 60%. |

In a nutshell, this report is a design and development of iris post processing and compression implemented on FPGA. It begins with an introduction to the background, past to current research on the topologies and strategies of conventional design in Chapter 1, Later, Chapter 2 reviews past literature on post processing and compression techniques while the design methodology is proposed in Chapter 3. Chapter 4 is the results and discussion and finally Chapter 5 concludes this work.

# **CHAPTER 2: LITERATURE REVIEW**

Iris recognition has become a popular research topic in recent years. Many approaches have been proposed for iris recognition. There are three main stages in an iris recognition system: iris acquisition and iris segmentation, image post processing and template matching. The following subsections will review several conventional techniques used particularly for Iris Normalization and Compressions.

**2.1 Conventional Methods for Iris Post Processing: Iris Normalization**

**2.1.1 Daugman’s Rubber Sheet Model**

Iris may be captured in different sizes with varying imaging distance. Due to illumination variations, the radial size of the pupil may also be different. In some cases, the pupil and iris may be non-concentric. Therefore, the iris region needs to be normalized to compensate for these variations. The homogeneous rubber sheet model algorithm remaps each pixel in the localized iris region from the Cartesian coordinates (*x,y*) to rectangular polar coordinates *(r,θ)* as shown in Figure 2.1 where *r* is on the interval [0,1] and *θ* is angle [0,2π] (Daugman, 2013).

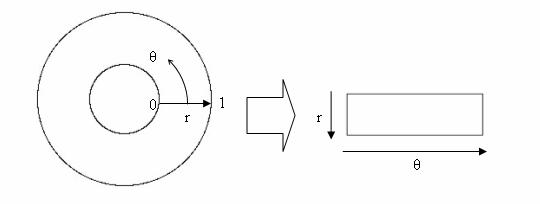


Figure 2.1 Illustration of Cartesian to Rectangular Coordinate Conversion of an Iris (Daughman, 2013)

The remapping of the iris region from *(x,y)* Cartesian coordinates to the normalized non-concentric polar representation is modelled as the expressions shown in (2.1), (2.2)and (2.3):-

*I(x(r,θ),y(r,θ))I(r,θ)* (2.1)

with

*x(r,θ)=(1-r)(θ)+r(θ)*  (2.2)

*y(r,θ)=(1-r)(θ)+r(θ)* (2.3)

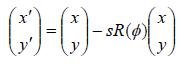
where *I(x,y)* is the iris region image, *(x,y)* are the original Cartesian coordinates, (r,θ) are the corresponding normalized polar coordinates, and*,*  and *, ,*  are the coordinates of the pupil and iris boundaries along the *θ* direction (Daugman, 2013). The rubber sheet model takes into account pupil dilation and size inconsistencies in order to produce a normalized representation with constant dimensions. In this way the iris region is modelled as a flexible rubber sheet anchored at the iris boundary with the pupil centre as the reference point.

Even though the homogeneous rubber sheet model accounts for pupil dilation, imaging distance and non-concentric pupil displacement, it does not compensate for rotational inconsistencies (Daugman, 2013). In the Daugman system, rotation is accounted for during matching by shifting the iris templates in the *θ* direction until two iris templates are aligned (Daugman, 2013).

### **2.1.2 Image Registration**

The Wildes *et al*. employs an image registration technique, which geometrically warps a newly acquired image,  *(x,y)* into alignment with a selected database image  *(x,y)* as shown in expression 2.4 (R.Wildes, 1996). When choosing a mapping function *(u(x,y),v(x,y))* to transform the original coordinates, the image intensity values of the new image are made to be close to those of corresponding points in the reference image. The mapping function as shown in (2.4) must be chosen so as to minimize while being constrained to capture a similarity transformation of image coordinates *(x,y)* to *(x’,y’)*, that is given by the expression in (2.5).

(2.4)

 (2.5)

with *s* as a scaling factor and *R(φ)*as a matrix representing rotation by *φ.* In implementation, given a pair of iris images and, the warping parameters *s* and *φ* are recovered via an iterative minimization procedure (R.Wildes, 1996).

### **2.1.3 Virtual Circles**

In the Boles system, iris images are first scaled to have constant diameters so that when comparing two images, one is considered as the reference image. This works differently from the other techniques, since normalization is not performed until an attempt is made to match two iris regions. Once the two irises have the same dimensions, features are extracted from the iris region by storing the intensity values along virtual concentric circles, with origin at the center of the pupil (W. Boles, 1998).

A normalization resolution is selected, so that the number of data points extracted from each iris is the same. This is essentially the same as Daugman’s rubber sheet model. However scaling is done during matching, and is relative to the iris comparison region, rather than scaling to some constant dimensions. Also, it is not mentioned by Boles, how rotational invariance is obtained (W. Boles, 1998).

## **2.2 Conventional Methods for Feature Extraction**

Iris has an abundance of unique texture features, especially at the inner section of the iris. Feature extraction transforms the enhanced iris image into a set of discriminating signatures. The significant features are encoded into templates for identification.

### **2.2.1 Gabor Filters**

Daugman proposed 2D Gabor filters to extract iris feature (Daugman, 2013). Gabor filter's impulse response is defined as a harmonic function multiplied by a Gaussian function. It provides optimum localization in both spatial and frequency domain (Daugman, 2013).

Each isolated iris pattern is demodulated to extract its phase information using quadrature 2D Gabor wavelets. The phase information is extracted for recognizing irises because it is discriminating. It does not depend on extraneous factors, such as imaging contrast, illumination and camera gain (Daugman, 2013).

1D Log Gabor filter is also used to extract frequency information which represents the iris texture. A Log Gabor filter is a Gaussian transfer function on a logarithmic scale (Field, 1987). It is a band pass filter that removes the DC components caused by background brightness. The 1D Log Gabor filter on the linear frequency scale has a transfer function as shown below in the expression given by (2.6).

G(w)=exp((-log(w/ (2.6)

where *ω0* denotes the filter’s center frequency and *k* denotes the bandwidth of the filter. The plot of 1D Log Gabor filter in frequency domain is shown in Figure 2.2, while Figure 2.3 and Figure 2.4 show the real and imaginary part of the 1D Gabor Filter respectively.



Figure 2.2: 1D Log Gabor filter in the Frequency Domain [Ref…,Year]

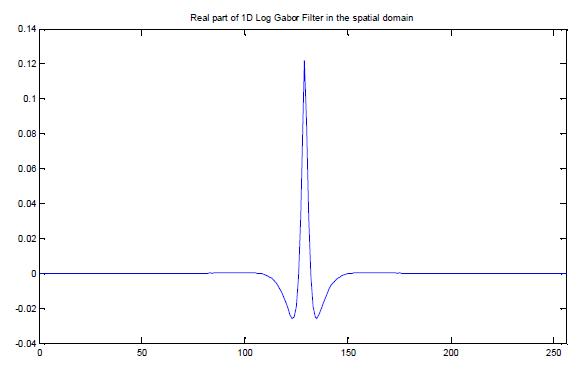


Figure 2.3: Real part of 1D Log Gabor Filter in the Spatial Domain [Ref…,Year]

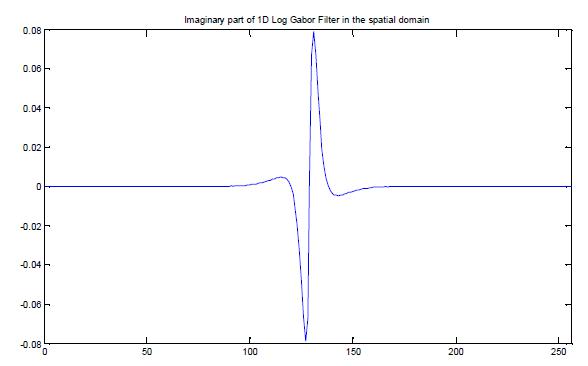


Figure 2.4: Imaginary part of 1D Log Gabor Filter in the Spatial Domain [Ref…,Year]

After applying 1D Log Gabor filter on each row of the enhanced iris image, a series of real and imaginary numbers are generated. The phase information is quantized into four quadrants in the complex plane as shown in Figure 2.5. The equations of the phase demodulation process are defined by the equations shown below from (2.7) to (2.10). The template consist of binary digits only (Daugman, 2013) where *f(x,y)* denotes the filtered image after performing inverse Fast Fourier Transform (FFT).

(2.7)

(2.8)

(2.9)

(2.10)

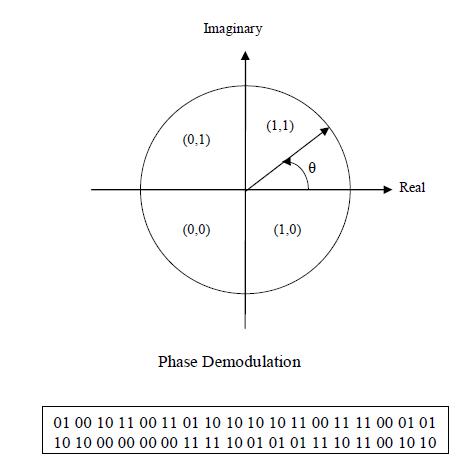


Figure 2.5: Phase Demodulation Process

### **2.2.2 Wavelet Transform**

Wavelet transform decomposes the iris region into components with different resolutions. The commonly used wavelets are Daubechies, Biorthogonal, Haar and Mexican Hat wavelet (Poursaberi and Araabi, 2005; Chen *et al*., 2006; Rydgren*et al*., 2004; Boles and Boashash, 1998; Sanchez-Avila *et al*., 2002).

A band of wavelet filters is applied to the normalized iris region. Each filter is tuned for each resolution with each wavelet defined by a scaling function. The output of the filters is encoded to generate a compact biometric template.

The advantage of wavelet transform over Fourier transform is that is has both space and frequency resolution. The features are localized in both space and frequency domains with varying window sizes.

### **2.2.3 Discrete Cosine Transform**

This iris coding method is based on differences of Discrete Cosine Transform (DCT) coefficients of angular patches from normalized iris image (Monro *et al*., 2007). The normalized iris image is divided into diagonal 8x12 sub-blocks as shown in Figure 2.6. The average width is windowed using a Hanning window to reduce the effects of noise. A similar Hanning window and DCT is applied to the patch along its length.

The differences between the DCT coefficients of adjacent sub-blocks are obtained. A binary template is generated from the zero crossings of the differences between the DCT coefficients. This iris coding method has low complexity and good interclass separation. It is superior to other approaches in terms of both speed and accuracy. The general equation for 1D DCT is defined by the equation expressed in (2.11).

*F(u) = (* (2.11)

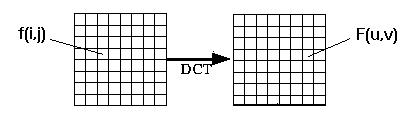


Figure 2.6: Images Separated into Sub-blocks

# **CHAPTER 3: METHODOLOGY**

The proposed IRS block is as shown in Figure 3.1 and it consists of three sub-systems. The first sub-system is the Image Acquisition & Image Segmentation block. This block acquires the iris image to be segmented, whereby the iris is separated from the pupil on top of performing noise reduction. The second sub-system is the Image Post Processing (Iris Normalization) and Compression module which is detailed in Figure 3.2. This sub-system is used to extract a region in the iris image and to compress this extracted region into a binary image. The final sub-system is the Self Organizing Map (SOM) block which is used to compare the iris template captured from the camera to the iris template from the database that has been previously trained.

This research focuses on the Image Post Processing (Iris Normalization) & Compression subsystem which consist of two main blocks as shown in Figure 3.2. In this sub-system, the iris extraction block (BLOCK 1) is used to extract a region in the iris image. Then the next block which is the compression block (BLOCK 2) will compress the extracted iris region into a binary image of smaller size and output this data for matching purposes.

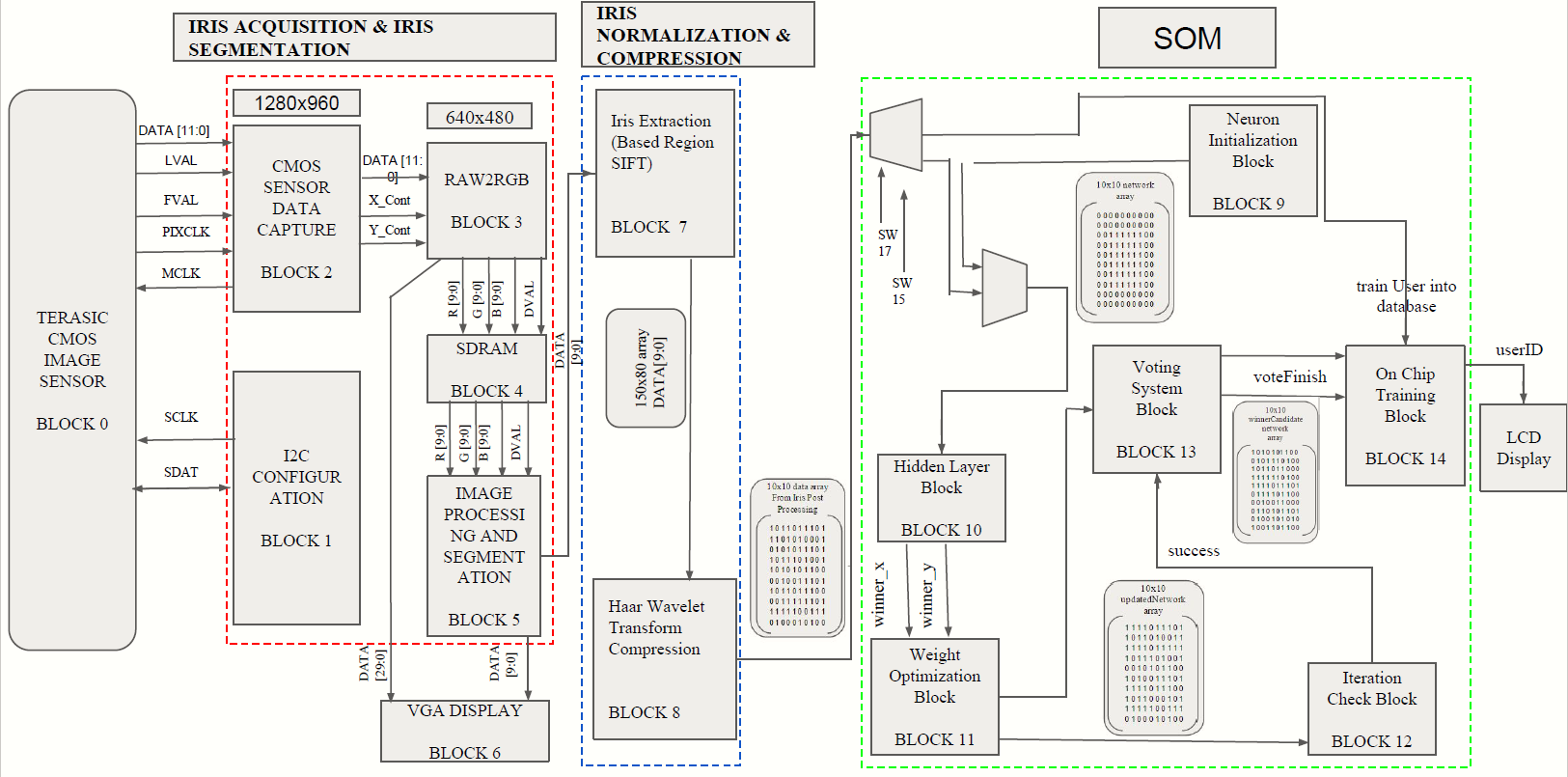


Figure 3.1: Iris Recognition System Block Diagram

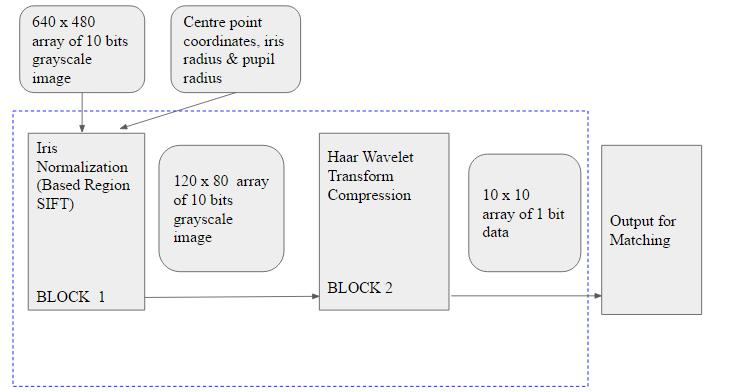


Figure 3.2: Iris Normalization and Compression Block Diagram

## **3.1 Iris Normalization Block**

In the iris normalization block, the iris image is represented in a 640x480 array. Each array element represents a 10-bit pixel that will be transformed into a fixed size dimension by extracting a specific iris region from the iris image. The size of this extracted iris region is a 120x80 array of 10-bits pixel element. This iris region will be sent to the next block which is the compression block. A fixed iris region is important for iris matching as it will compensate the noise such as eyelid and eyelashes.

This block uses the method known as region-based Scale Invariance Feature Transform (SIFT). The iris image shown in Figure 3.3 is separated into four regions: upper region, lower region, left region and right region (Belcher, 2009). Features are only matched in the same region, which can eliminate mismatching. The process of iris normalization begins from the lower region of the iris. This is due to the upper region contains noise such as eyelid and eyelashes. The left and right region also not considered for extraction because both the regions exceed the iris boundaries.

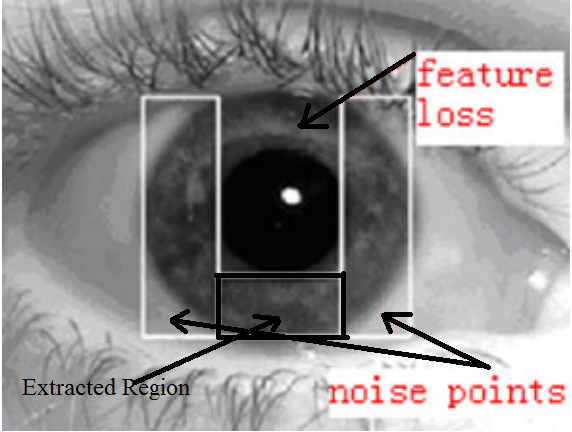


Figure 3.3: Iris Extraction Region [Ref… Year]

In order to extract the iris region, the coordinates of the pupil and iris needs to be calculated by using two inputs from the Image Acquisition and Image Segmentation sub-systems. These inputs are centre points of the grayscale image which is represented by *(x,y)* and also the radius of pupil and iris. The Hough Transform algorithm is used to calculate the pupil’s left right and bottom coordinates as well as the iris bottom coordinates. The general equation of the Hough Transform algorithm to calculate *x* and *y* coordinates are given by the expression in (3.1) and (3.2) respectively.

*X = + \* COS(θ)*  (3.1)

*Y = + \* SIN(θ)* (3.2)

where is the *x* coordinates of pupil center point, is the *y* coordinates of pupil’s center points, is the *x* coordinate of pupil’s or iris’s radius and is the *y* coordinates of pupil’s or iris’s radius. Figure 3.4 shows the visualized process to calculate the coordinates of pupil and iris while the Verilog HDL code for this algorithm is illustrated in Figure 3.5. In the partial code, o*utputCos* and *outputSin* are the representation of the cosine and sine function. Since the function of cosine and sine needs to be explicitly coded in a HDL, both functions were computed using the Verilog HDL based algorithm for *outputCos* and *outputSin* as shown in Figure 3.6.

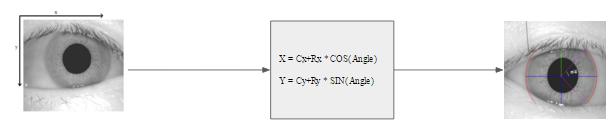


Figure 3.4: Visualization of Hough Transform Algorithm

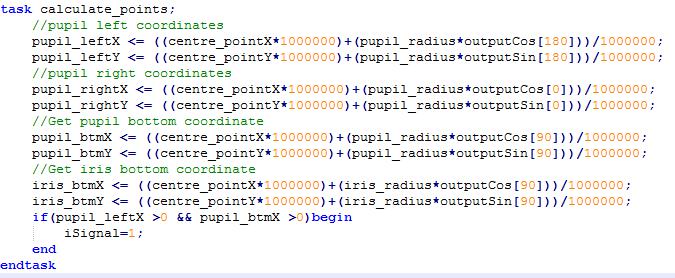


Figure 3.5: Hough Transform Algorithm Code in Verilog HDL

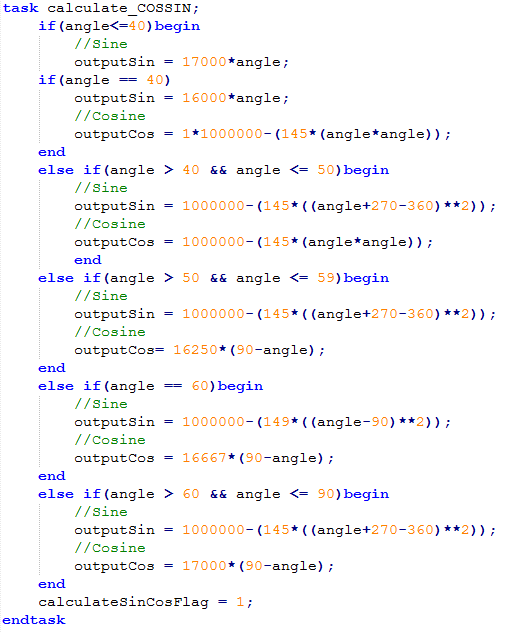


Figure 3.6: Cosine and Sine Function Code

Next, the flowchart for the iris normalization block is as shown in Figure 3.7. The process of the flow begins when the inputs are received from the previous sub-system (segmentation). Once the inputs are received, a flag will indicate 1 before performing pupil and iris coordinates calculation. After the coordinates of the pupil and iris is calculated, the range of rows and columns needed for iris extraction is calculated based on the coordinates of the pupil and iris.

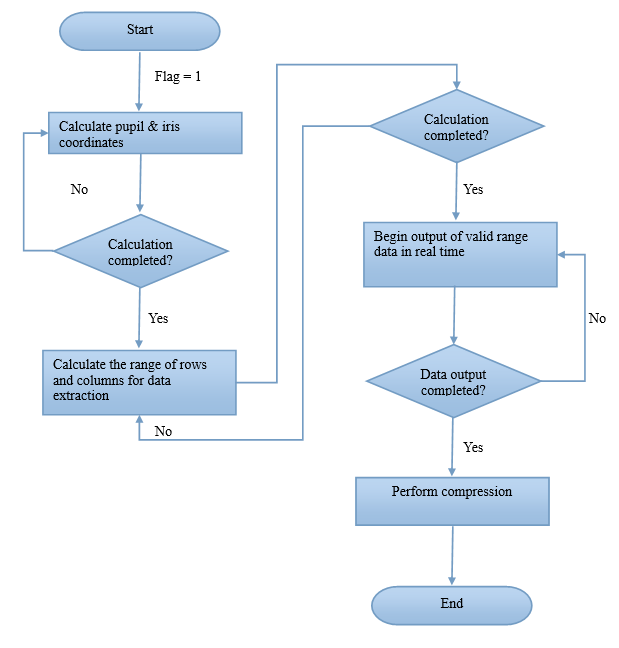


Figure 3.7: Iris Normalization Flowchart

The starting point of the extraction region for the row is the pupil’s left-most x-coordinates while the ending point is the pupil’s right-most x-coordinates. For the column, the starting point of the extraction region is the pupil’s bottom-most y-coordinates and the ending point is the End Point where it is expressed by (3.3), where the iris bottom-most coordinates is used to ensure that the End Point will not exceed the iris boundaries.

*End Point* = *pupil bottom y-coordinate* + 80 (3.3)

In the extraction process, the input grayscale values will be handled in real time. When the first point of the extraction region is matched, the grayscale values within the extract region begin to output one frame per second in real time to the next block. By using real time to transfer data, the data does not require overwhelming memory storage. This can save the immerse resources on the DE2 board and increase the processing speed. Figure 3.8 is an example of how the grayscale values are being extracted. The example used is a 10x8 array to perform extraction. Value 31 is the first point of the extraction region and value 37 is the last point of the extraction region. Grayscale values which are outside the extraction region will not be output to the next block.

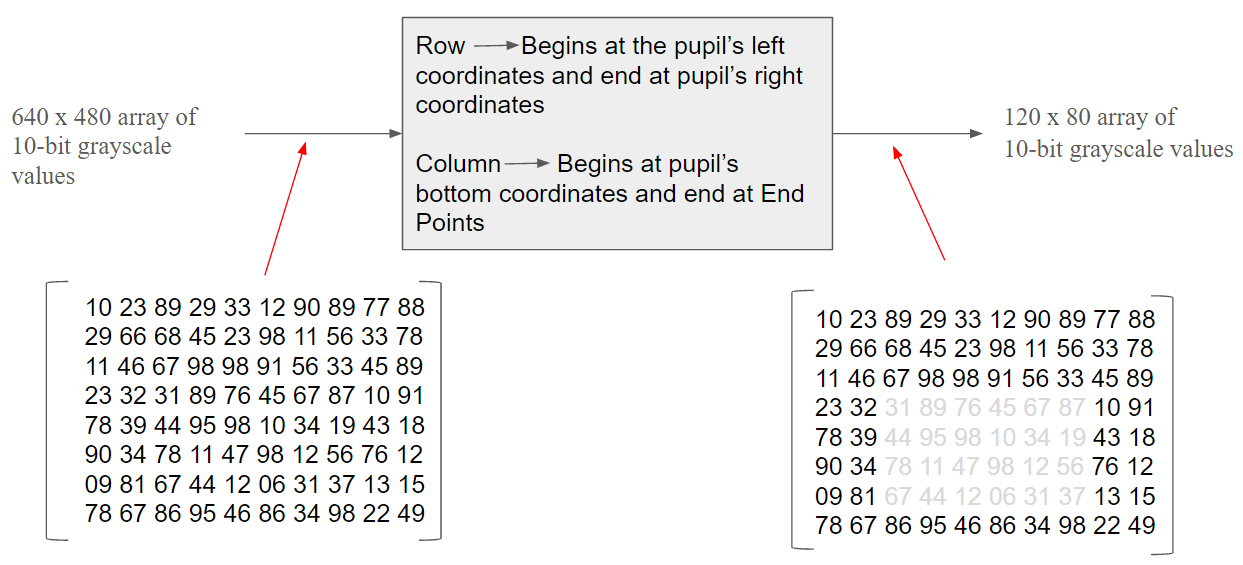


Figure 3.8: Sample Input and Output Matrixes for Iris Normalization

## **3.2 Compression Block**

In the compression block, the extracted iris region which is a 120x80 array is compressed into a 10x10 array. The bit size still remained as 10 bits. Next, the 10 bits of each array element is compressed into 1 bit via thresholding. As the result, grayscale image will be represented by a binary template. After compression is done, the binary template is being sent to the last sub-system for training and matching purposes.

Compression of the iris region into a 10x10 array is based on Discrete Haar Wavelet Transformation (HWT). The basic idea of HWT is to treat the digital iris image as an array of grayscale numbers. For instance, a grayscale image of resolution 256x256 is represented by a 256x256 matrix. Next, the HWT algorithm divides this matrix into 8x8 blocks and assigns matrix for each block. Figure 3.9 shows the image which is comprised of pixels represented by grayscale values.

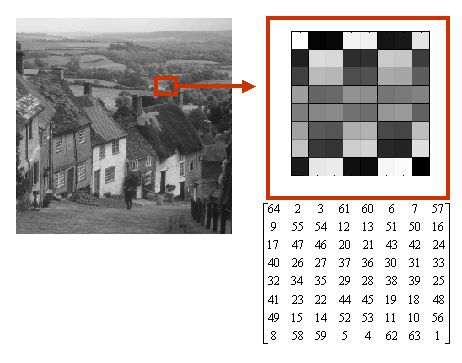


Figure 3.9: Images Represented in Grayscale Values

The extracted iris region is a 120x80 matrix which will require a 2D HWT for compress. The simple approach of 2D HWT implementation is to perform a 1D HWT row-wise to produce the intermediate results and then perform the same 1D HWT column-wise on the intermediate results to get the final results. Thus, for each row of 120 grayscale values, the grayscale values is divided into 10 elements by taking 12 values at one time. This step is required to obtain the intermediate average value for each part. Once 10 elements with average values are obtained for the first row, the same steps are applied for the following rows until the final row is completely computed. The intermediate matrix will be a 10x80 matrix. Column-wise, there are 80 values in each column, every 8 values are taken at one time, and the average values are obtained. This step is iterated for the next columns until the last column is completed. The final matrix will be a compressed into a 10x10 matrix. Figure 3.10 is an example of a 2D HWT in row-wise and then column-wise while Figure 3.11 and Figure 3.12 shows the Verilog HDL code for 2D HWT compression for rows and columns respectively.

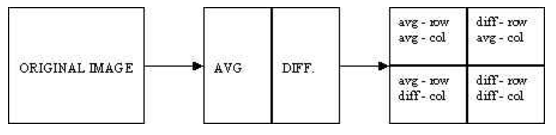


Figure 3.10: Row-Column Computation of 2D HWT

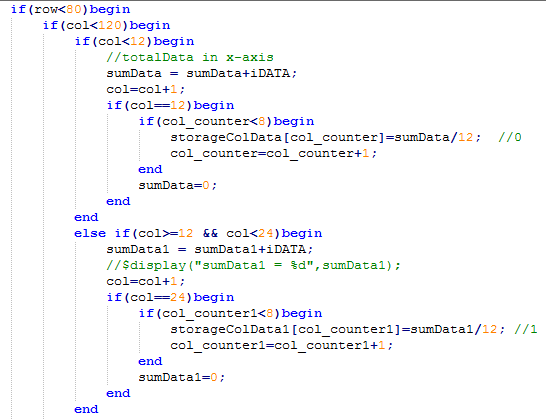


Figure 3.11: 2D HWT Row-wise Code



Figure 3.12: 2D HWT Column-wise Code

Once the iris region is successfully compressed into a 10x10 matrix, the next step is to compress the 10-bit grayscale value into a 1-bit binary value for each element. Firstly, the 10-bits grayscale value is being summed from the Least Significant Bit (LSB) to Most Significant Bit (MSB). If the total value of bit 0 to bit 9 is more than 5, then the output will be converted to a single bit of 1 else it will be 0. Figure 3.12 shows the simple block which describes the process of bits compression. The Verilog HDL code for compression is shown in Figure 3.13 and the flowchart of the compression block is as shown in Figure 3.14.

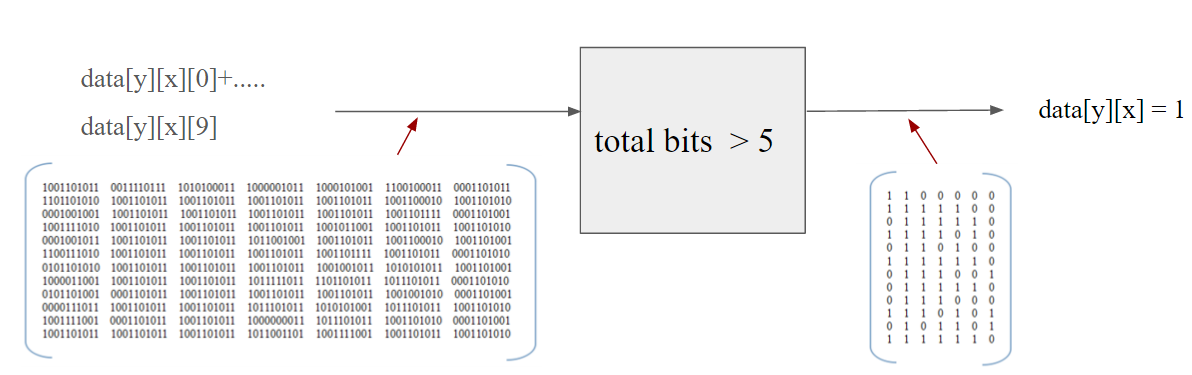


Figure 3.12: Sample Input and Output Matrices for Bit Compression

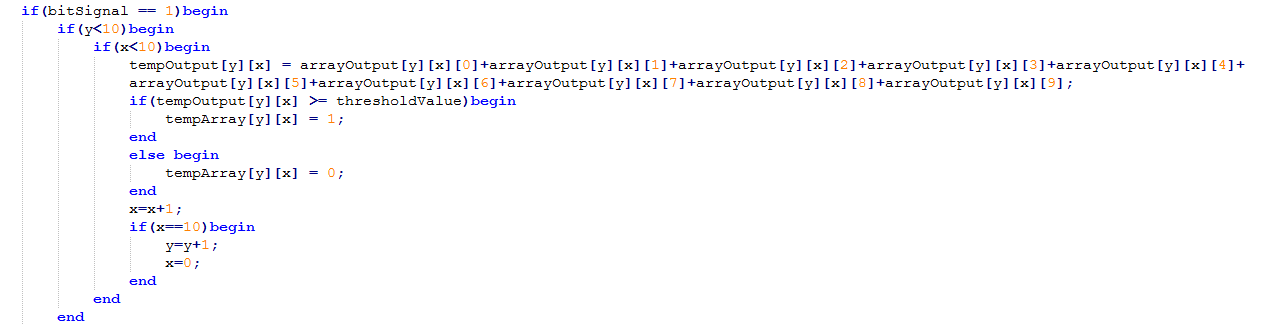


Figure 3.13: Compression Code

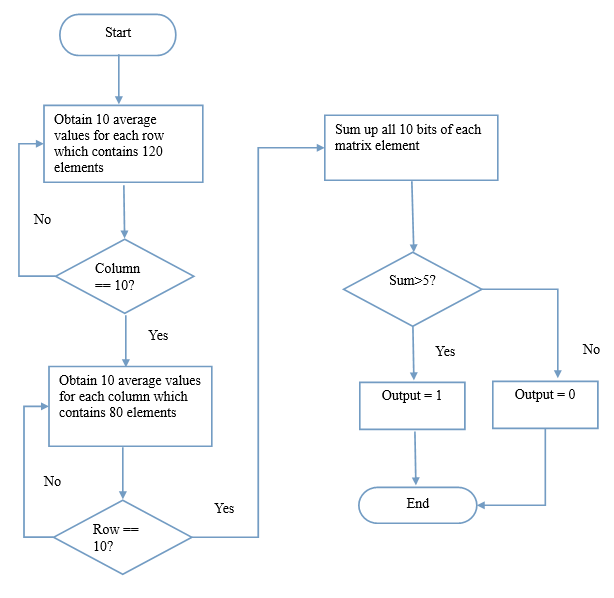


Figure 3.14: Iris Compression Flowchart

# **CHAPTER 4: RESULTS & DISCUSSION**

The results and discussions of the image post-processing (Iris Normalization) and compression blocks will be presented in three main sections. Firstly, in a block by block basis, secondly integration between both blocks and finally the outcome of fully integrated IRS system. The following subsections describe in detail the results and discussions of each block and as an integrated system. Finally, the total integrated result of all sub-system will be presented in section 4.3.

**4.1: Results & Discussions of the Sub-Blocks**

**4.1.1: Image Post Processing (Iris Normalization)**

Figure 4.1 to Figure 4.3 shows simulated results in ModelSim using ideal data. There are two sets of ideal data. The first set of the ideal data is the values for centre point of pupil and radius of the pupil and iris. The second set of ideal data is made up of random numbers ranging from 0 to 1023 which represents the 10-bit grayscale values. By using ideal data in ModelSim, the code can be tested and improved upon.

Figure 4.1 shows that the iris normalization block begin taking in parameters such as *iDATA* which consist of input data, *pupil\_radius*, *iris\_radius*, *centre\_pointX* and *centre\_pointY*. *oDATA* which is the output data still does not show any output at this point. After 850 ps, *oDATA* begins to output the *iDATA* in grayscale values. This means that region extraction begin at this point. There is a long delay after *oDATA* output value 260 as shown in Figure 4.2. This is because 260 is the last value to output in a specific row. The remaining values will not be output as there it exceeds the row-wise range. The *oDATA* will then continue to output the *iDATA* grayscale values when it iterates to the next row. Figure 4.3 shows that the value of 230 in *oDATA* is the last value to be outputted. The *iDATA* still continues to input the grayscale values but *oDATA* will no longer produce any output as the extraction of grayscale values for the specific region is completed.

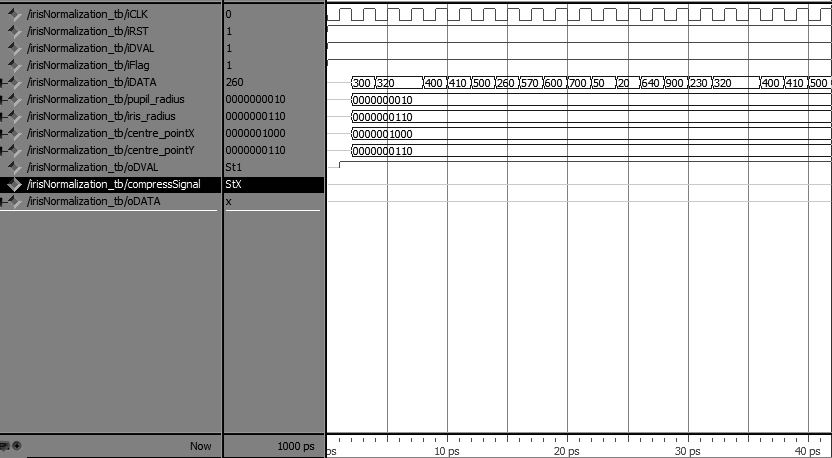


Figure 4.1: Iris Compression Waveform with Ideal Data from 0 to 40 *ps*

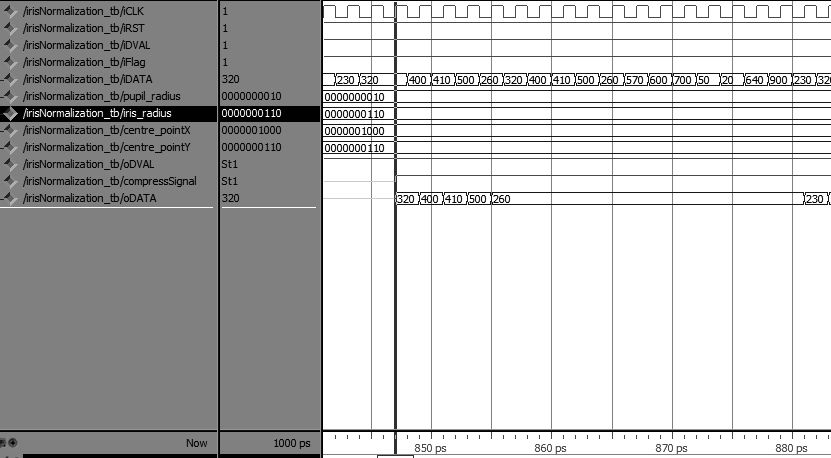


Figure 4.2: Iris Compression Waveform with Ideal Data from 840 to 880 *ps*

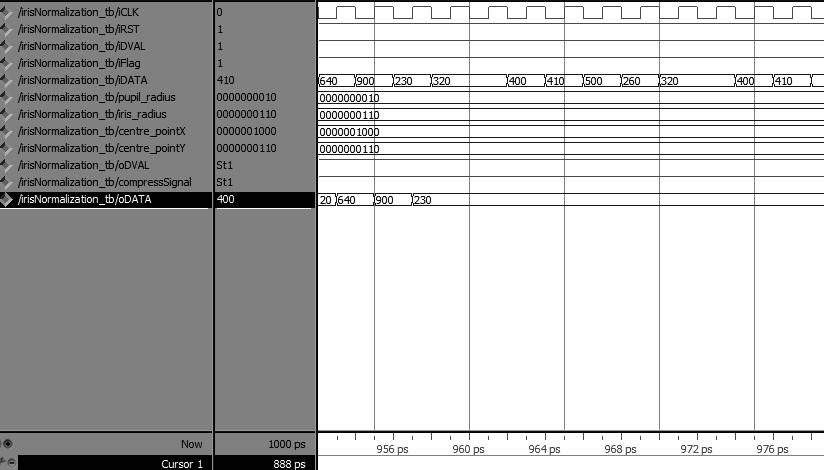


Figure 4.3: Iris Compression Waveform with Ideal Data from 950 to 976 *ps*

Figure 4.4 shows the result of iris normalization in a hardware test through the display monitor. The arrow indicates the center point coordinates. The starting point and ending point of the extraction region is also indicated in the diagram. The Square is the region in which the grayscale values will be extracted to the next block.



Figure 4.4: Result of Iris Normalization in Hardware

### **4.2.1: Compression**

Figure 4.5 to Figure 4.7 is the simulation result of compression performed in ModelSim of Mentor Graphics environment. In the Figure 4.5, the *oArray* has yet to output any value. This is because the iris region is being compressed from a 120x80 matrix to a 10x10 matrix at this point of time. After that, the 10x10 matrix which is still in 10-bit element will be compressed into a 1 bit binary format. Figure 4.6 shows that *oArray* begin to fetch in the binary values into the coordinates and Figure 4.7 shows that after the *oArray* has been completely filled up, as a 10x10 array of 1 bit elements, the *oArray* will trigger the *oMatchSignal* to indicate that the compression is complete and is ready to be passed onto the output for the next subsystem.



Figure 4.5: Compression Results 1

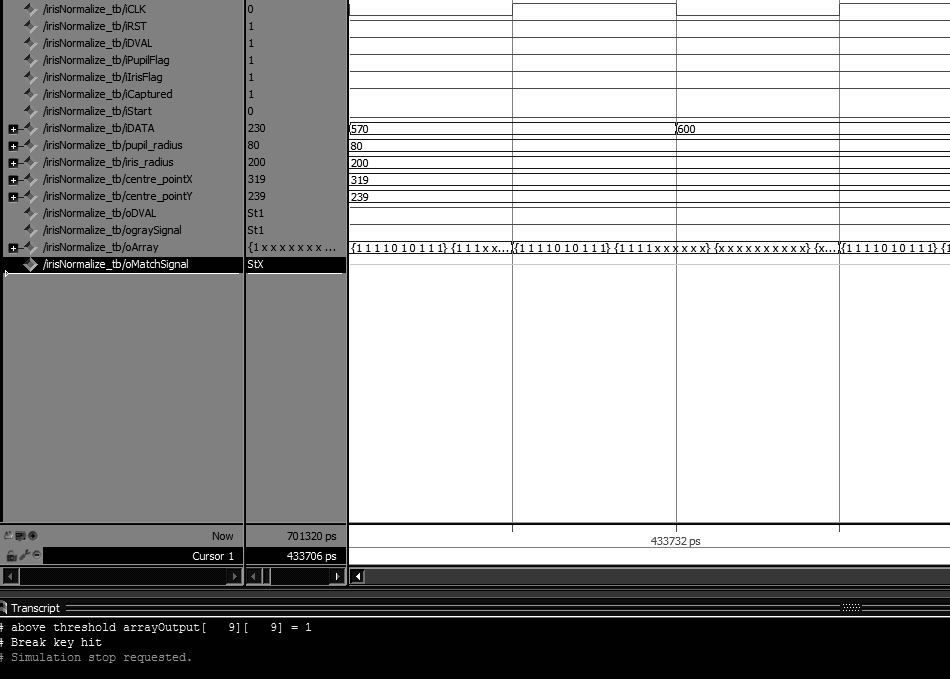


Figure 4.6: Compression Results 2

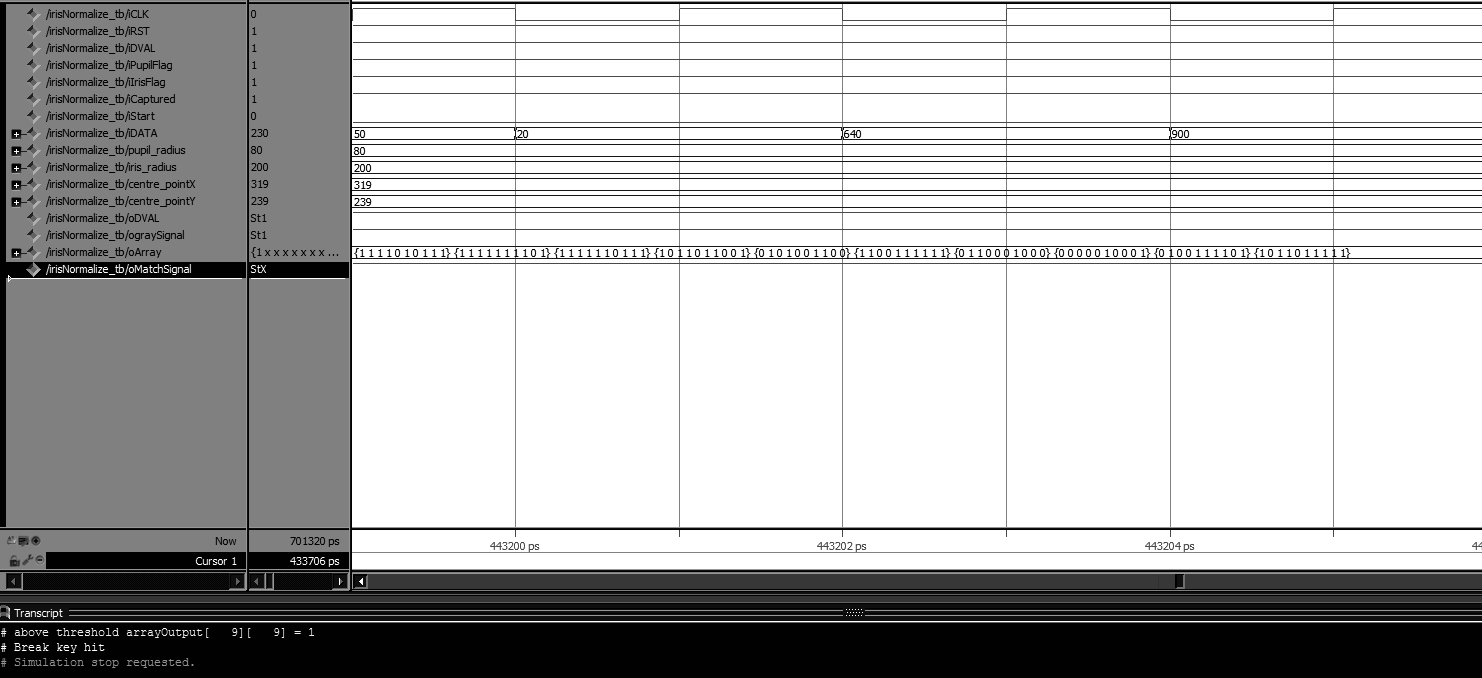


Figure 4.7: Compression Results 3

## **4.2: Analysis**

Figure 4.8 shows the total FPGA resources being utilized to implement iris normalization module. Figure 4.9 shows the compilation time for the iris normalization module. The compilation time for the fitter is the longest as iris extraction require more time to scan through the entire iris image starting from the top left-hand corner of the iris image. The power consumption for the iris normalization module is 117.76 mW which is shown in Figure 4.10.

Figure 4.11 shows the total FPGA resources being utilized to implement compression module. Figure 4.12 shows the compilation time to perform compression. The compilation time for analysis and synthesis is the longest as compression requires more time to calculate the average values for rows and columns. The power consumption for the iris normalization module is 123.36 mW which is shown in Figure 4.13.

Figure 4.14 shows the total FPGA resources being utilized to implement the fully integrated IRS. Figure 4.15 shows the compilation time of the integrated IRS. The compilation time for the fitter is the longest as the iris image is a 2D array which requires larger registers to store the grayscale values. The power consumption for the iris normalization module is 589.48 mW which is shown in Figure 4.16.

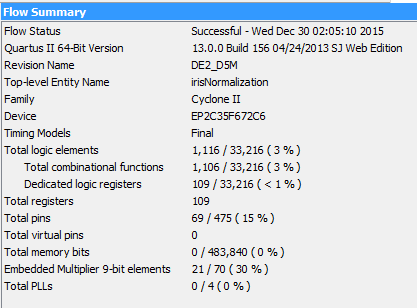


Figure 4.8: FPGA Resource Usage for the Iris Normalization Module

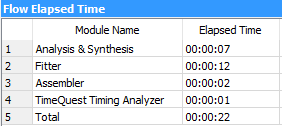


Figure 4.9: Iris Normalization Compilation Time

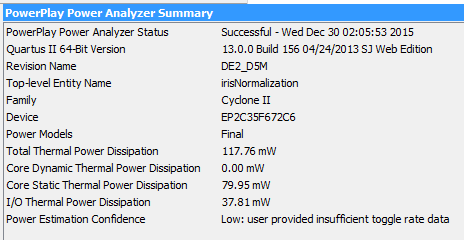


Figure 4.10: Power Consumption for the Iris Normalization Module

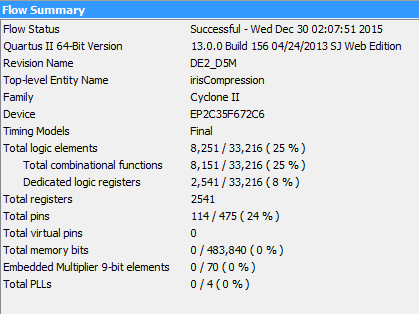


Figure 4.11: FPGA Resources usage for the Compression Module

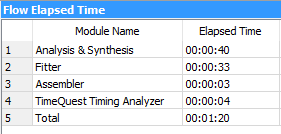


Figure 4.12: Compression Compilation Time

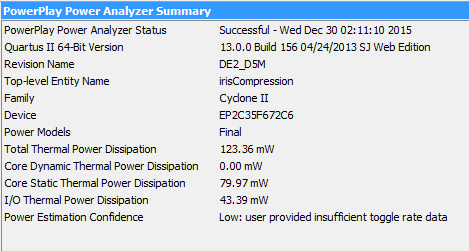


Figure 4.13: Power Consumption for the Compression Module

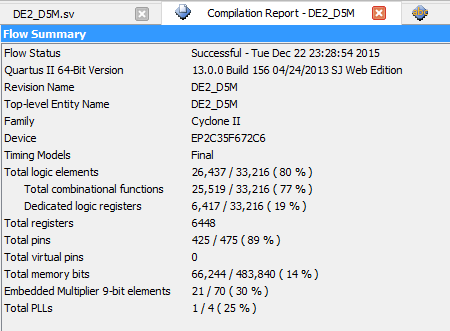


Figure 4.14: FPGA Resource Usage for the Fully Integrated IRS

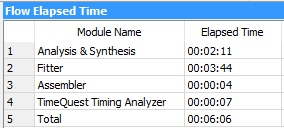


Figure 4.15: Fully Integrated IRS Compilation Time

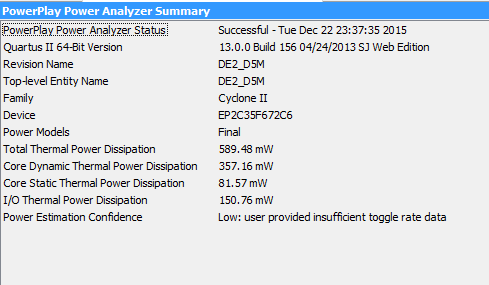


Figure 4.16: Power consumption for the Fully Integrated IRS

## **4.3: Integrated Result**

There are 3 Iris images from 3 different users being used for training and matching in this hardware integration test. Each user will be trained for 3 consecutive times which provides the database with a total of 9 images, three each from three different users.. The trained images are shown from Figure 4.18 to Figure 4.26.

Table 2 and Figure 4.17 show the integrated result of the Iris Recognition System. For each user, we will match for a total 6 times to calculate the accuracy. For User\_1, the input iris image can be matched correctly up to 4 times with the database we trained. User\_2 also successfully matched 4 times with the database. However, during the third matching test, it showed that the User\_2 has been wrongly matched to User\_3. This is due to the extracted region of both these 2 user’s iris image being quite similar after being compressed into a binary template. Next, for User\_3, the success rate of matching is only half which is the lowest among the 3 users.

The overall matching possibility is above 50%. For User\_1 and User\_2 the matching probability is 66.67% and 83.3% respectively while the matching possibility for User\_3 is exactly 50% for a total of 6 hardware trial tests for each user. The matching accuracy is 100% for both User\_1 and User\_3 while the matching accuracy for User\_2 is 90% for a total of 6 conducted hardware trial tests. The average accuracy is 96.67% while the average matching probability is 66.67% as summarized in Table 2 and illustrated in the bar chart of Figure 4.26.

# https://lh4.googleusercontent.com/jel8rlGvQyzx_fsxGNuqWIlDSk6Ot8FOyh-2l8uex6L4Oflq-78kloWvPkwbd1JJYMcw4iCtokdHdF9XalN2kCvgmH-KNoJifskG12lQjPOMrUMkAmQvZte5YDT3G8ZVaFRv_NxgBghttps://lh6.googleusercontent.com/hkiRhUeDkBh9Oy5pTwhU5avZ6eZR8SQ0rOr_hmurNtYROl5QS_X-PrsZ_WqHzmcGxtngqOVUdm2c4zTEuR32BLmlFmoPN4h0LsSpG3qNuDZdmZrDIDaZriaFa0XA59odO1yOxmW29Qhttps://lh3.googleusercontent.com/8qJ6PDrYcEDNHLDBb8_x-Q_0EBtHxt10f4k_e0_4kTLcN137766mmeEPF2mEbWHp9laCu2G2aQGmr4eRfW_L-kOqtMgotTdxJgxSfy_TTEunh1fa1Kci5MIpWKET-913Zql2Eq55yg

Figure 4.17: User\_1 (1st Sample) Figure 4.18: User\_1 (2nd Sample) Figure 4.19: User\_1 (3rd Sample)



Figure 4.20: User\_2 (1st Sample) Figure 4.21: User\_2 (2nd Sample) Figure 4.22: User\_2 (3rd Sample)

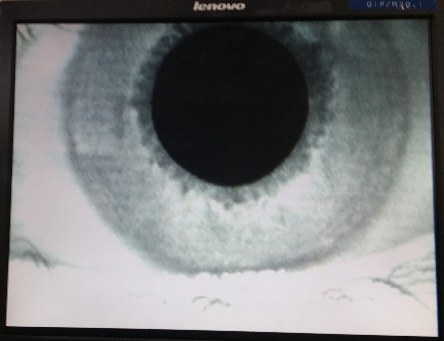
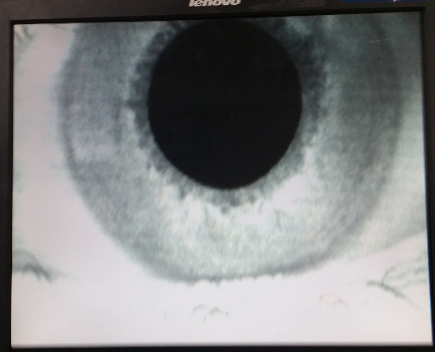
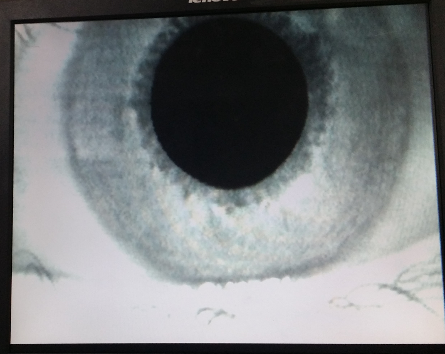


Figure 4.23: User\_3 (2nd Sample) Figure 4.24: User\_3 (2nd Sample) Figure 4.25: User\_3 (3rd Sample)

Table 2: Hardware Implementation Result

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Trained User | Matching Accuray | | | | | |
| User\_1 | 1 | 1 | 1 | 1 | X | X |
| User\_2 | 2 | 2 | 3 | 2 | 2 | X |
| User\_3 | 3 | 3 | 3 | X | X | X |

Figure 4.26: Bar Chart of Integrated Result

# 

# **CHAPTER 5: CONCLUSION**

# **5.1: Conclusion**

In this project, the iris recognition system is developed by using Verilog HDL based on FPGA and implemented on the DE2 board. Firstly, the iris image is captured by the CMOS camera. Then the iris image is used to detect the pupil and iris points for segmentation. The segmented iris image, which is a grayscale image will be normalized by extracting a specific region for comparison. This extracted region is compressed and a binary template is produced. Finally, this binary template is trained into the database in preparation for matching of the incoming user.

This work focuses on image post-processing (Iris Normalization) and compression which are the two main steps to extract and compress into a binary template. The main purpose of Iris Normalization is to extract a specific region from the iris image to make later comparison. It is important to normalize before comparison or matching as the iris is non-concentric. Furthermore, in order to use less resources and increase the processing time, the extraction region is compressed into a 10x10 matrix consisting of only 1 bit in each element. The proposed image post-processing and compression will be modeled, designed and simulated in Verilog HDL using ModelSim of Mentor Graphics environment and later integrated into Altera Cyclone II DE2 board for hardware verification purposes. This result is a faster processing time and lower resource usage of the DE2 board. However, the matching accuracy is relatively low if only one iris image is trained for each user for database matching purposes. With multiple (3) iris images trained for each user, matching accuracy reaches an average of 96.67% albeit with an average matching possibility of 66.67%. The lower accuracy when single iris image was used when performing hardware verification is due to compression from a large image to a smaller image as well as 10-bit being compressed into 1 bit. Finally, the proposed IRS system can be applied to security system which uses the IRS for faster access.

## **5.1: Future Trends & Recommendation**

There are several recommendations for future work. Firstly, the hardware setup can be improved by using a better camera with which autofocus function and also to compensate the lighting condition. Next, an image enhancement algorithm could be developed to enhance the extracted iris region before compression is done. The iris normalization algorithm could also be improved by using the John Daugman algorithm which normalizes the entire iris image to form an iris template for matching purposes. Furthermore, instead of compressing 10 bits to a 1 bit value, the 10 bits could be compressed into 2 bitsbefore matching is done.. By comparing 2 bits in each coordinates, the matching accuracy could be increased.

# **REFERENCE**

1. J. G. Daugman, “High confidence visual recognition of persons by a test of statistical independence,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, 2013, pp. 1148-1161.
2. J.G. Daugman, "The importance of being random: statistical principles of iris recognition", Pattern Recognition, Vol. 36, No. 2, pp. 279 – 291, February 2013.
3. R. Wildes, J. Asmuth, G. Green, S. Hsu, R. Kolczynski, J. Matey, S. McBride, A machine-vision system for iris recognition, Mach. Vision Appl. 9 (1996) 1–8.
4. 1. L. Birgale and M. Kokare, "Iris recognition without iris normalization," Journal of Computer Science, Vol. 6, No. 9, (2010), pp. 1042.
5. M. Nabti, L. Ghouti, and A. Bouridane, "An effective and fast iris recognition system based on a combined multiscale feature extraction technique," Elsevier, Pattern Recognition, Vol. 41, pp. 868–879, 2008.
6. RaidaHentati, MoncefBousselmi,MohamedAbid,“An Embedded System for iris Recognition, “International Conference on Design & Technology of Integrated Systems in Nanoscale Era. IEEE 2010.
7. Judith Liu-Jimenez, Raul Sanchez-Reillo, Carmen Sanchez-Avila, “Full Hardware solution for processing iris biometric,”.IEEE 2005
8. K. Grabowski, W. Sankowski, M. Napieralska, M. Zubert, A. Napieralski, “Iris Recognition Algorithm Optimized for Hardware Implementation,” IEEE 2006
9. Buethna, “Hardware Implementation of Iris Matching,”. International Journal of Scientific Research (IJSC) volume: 1/Issue: 6/ Nov 2012-ISSN No. 2277 -8179.
10. Babasaheb G. Patil, Nikhil Niwas Mane, ShailaSubbaraman, “Iris Feature Extraction and Classificatio-n using FPGA,” International Journal of Electrical and Computer Engineering, vol. 2, no. 2, April 2012.
11. A. Y. Yang, “Robust face recognition via sparse representation," IEEE PAMI, 2012.
12. Boles W. and Boashash B. (1998). A human identification technique using images of the iris and wavelet transform. IEEE Transactions on Signal Processing, vol. 46, no. 4, pp. 1185-1188.
13. Monro D. M., Rakshit S., and Zhang D. (2007). DCT-Based Iris Recognition. IEEE Transactions on PAMI, vol. 29, no. 4, pp. 586-595.
14. Poursaberi A. and Araabi B.N. (2005). A Novel Iris Recognition System Using Morphological Edge Detector and Wavelet Phase Features.GVIP (05), No. V6, pp. 9-15.
15. Belcher C, Du Y (2009) Region-based SIFT approach to iris recognition. Opt Lasers Eng 47(1):139–147
16. Why Do Math?. 2015. Why Do Math?. [ONLINE] Available at:<http://www.whydomath.org/node/wavlets/hwt.html>. [Accessed 29 December 2015].
17. Application to image compression. 2015. Application to image compression. [ONLINE] Available at:<http://aix1.uottawa.ca/~jkhoury/haar.htm>. [Accessed 29 December 2015].
18. M. Kokare, "Iris recognition without iris normalization," Journal of Computer Science, Vol. 6, No. 9, (2010), pp. 1042.
19. Sarin, “IRIS BIOMETRIC SYSTEM USING A HYBRID APPROACH”.IEEE 2014.
20. TSAI, J. S. TAUR AND C. W. TAO, “Iris Recognition Using Gabor Filters and the Fractal Dimension”. JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 25, 633-648 2007.
21. Chabrow,“Readying Iris Recognition for Prime Time”. http://www.bankinfosecurity.com/blogs/readying-iris-recognition-for-primetime-p-1545/op-1, 2014
22. Field D. (1987). Relations between the statistics of natural images and the response properties of cortical cells. Journal of the Optical Society of America, Vol. 4, Issue 12, pp. 2379-2394.
23. Chen Y., Dass S., and Jain A. (2006). Localized Iris Image Quality Using 2D Wavelets. Proceedings of International Conference on Biometrics, LNCS 3832, pp. 373-381.
24. Rydgren E., Ea T., Amiel F., Rossant F., and Amara A. (2004). Iris Feature Extraction Using Wavelet Packets. IEEE International Conference on Image Processing, vol. 2, pp. 861-864.
25. Sanchez-Avila C., Sanchez-Reillo R., and Martin-Roche D. De (2002). Iris-Based Biometric Recognition Using Dyadic Wavelet Transform. IEEE AESS System Magazines, vol. 17, no. 10, pp. 3-6.

# **Appendix**

# **Integrated Code: Main module**

module DE2\_D5M

(

//////////////////// Clock Input ////////////////////

CLOCK\_27, // 27 MHz

CLOCK\_50, // 50 MHz

EXT\_CLOCK, // External Clock

//////////////////// Push Button ////////////////////

KEY, // Pushbutton[3:0]

//////////////////// DPDT Switch ////////////////////

SW, // Toggle Switch[17:0]

//////////////////// 7-SEG Dispaly ////////////////////

HEX0, // Seven Segment Digit 0

HEX1, // Seven Segment Digit 1

HEX2, // Seven Segment Digit 2

HEX3, // Seven Segment Digit 3

HEX4, // Seven Segment Digit 4

HEX5, // Seven Segment Digit 5

HEX6, // Seven Segment Digit 6

HEX7, // Seven Segment Digit 7

//////////////////////// LED ////////////////////////

LEDG, // LED Green[8:0]

LEDR, // LED Red[17:0]

//////////////////////// UART ////////////////////////

UART\_TXD, // UART Transmitter

UART\_RXD, // UART Receiver

//////////////////////// IRDA ////////////////////////

IRDA\_TXD, // IRDA Transmitter

IRDA\_RXD, // IRDA Receiver

///////////////////// SDRAM Interface ////////////////

DRAM\_DQ, // SDRAM Data bus 16 Bits

DRAM\_ADDR, // SDRAM Address bus 12 Bits

DRAM\_LDQM, // SDRAM Low-byte Data Mask

DRAM\_UDQM, // SDRAM High-byte Data Mask

DRAM\_WE\_N, // SDRAM Write Enable

DRAM\_CAS\_N, // SDRAM Column Address Strobe

DRAM\_RAS\_N, // SDRAM Row Address Strobe

DRAM\_CS\_N, // SDRAM Chip Select

DRAM\_BA\_0, // SDRAM Bank Address 0

DRAM\_BA\_1, // SDRAM Bank Address 0

DRAM\_CLK, // SDRAM Clock

DRAM\_CKE, // SDRAM Clock Enable

//////////////////// Flash Interface ////////////////

FL\_DQ, // FLASH Data bus 8 Bits

FL\_ADDR, // FLASH Address bus 22 Bits

FL\_WE\_N, // FLASH Write Enable

FL\_RST\_N, // FLASH Reset

FL\_OE\_N, // FLASH Output Enable

FL\_CE\_N, // FLASH Chip Enable

//////////////////// SRAM Interface ////////////////

SRAM\_DQ, // SRAM Data bus 16 Bits

SRAM\_ADDR, // SRAM Address bus 18 Bits

SRAM\_UB\_N, // SRAM High-byte Data Mask

SRAM\_LB\_N, // SRAM Low-byte Data Mask

SRAM\_WE\_N, // SRAM Write Enable

SRAM\_CE\_N, // SRAM Chip Enable

SRAM\_OE\_N, // SRAM Output Enable

//////////////////// ISP1362 Interface ////////////////

OTG\_DATA, // ISP1362 Data bus 16 Bits

OTG\_ADDR, // ISP1362 Address 2 Bits

OTG\_CS\_N, // ISP1362 Chip Select

OTG\_RD\_N, // ISP1362 Write

OTG\_WR\_N, // ISP1362 Read

OTG\_RST\_N, // ISP1362 Reset

OTG\_FSPEED, // USB Full Speed,0 = Enable, Z = Disable

OTG\_LSPEED, // USB Low Speed, 0 = Enable, Z = Disable

OTG\_INT0, // ISP1362 Interrupt 0

OTG\_INT1, // ISP1362 Interrupt 1

OTG\_DREQ0, // ISP1362 DMA Request 0

OTG\_DREQ1, // ISP1362 DMA Request 1

OTG\_DACK0\_N, // ISP1362 DMA Acknowledge 0

OTG\_DACK1\_N, // ISP1362 DMA Acknowledge 1

//////////////////// LCD Module 16X2 ////////////////

LCD\_ON, // LCD Power ON/OFF

LCD\_BLON, // LCD Back Light ON/OFF

LCD\_RW, // LCD Read/Write Select, 0 = Write, 1 = Read

LCD\_EN, // LCD Enable

LCD\_RS, // LCD Command/Data Select, 0 = Command, 1 = Data

LCD\_DATA, // LCD Data bus 8 bits

//////////////////// SD\_Card Interface ////////////////

SD\_DAT, // SD Card Data

SD\_DAT3, // SD Card Data 3

SD\_CMD, // SD Card Command Signal

SD\_CLK, // SD Card Clock

//////////////////// USB JTAG link ////////////////////

TDI, // CPLD -> FPGA (data in)

TCK, // CPLD -> FPGA (clk)

TCS, // CPLD -> FPGA (CS)

TDO, // FPGA -> CPLD (data out)

//////////////////// I2C ////////////////////////////

I2C\_SDAT, // I2C Data

I2C\_SCLK, // I2C Clock

//////////////////// PS2 ////////////////////////////

PS2\_DAT, // PS2 Data

PS2\_CLK, // PS2 Clock

//////////////////// VGA ////////////////////////////

VGA\_CLK, // VGA Clock

VGA\_HS, // VGA H\_SYNC

VGA\_VS, // VGA V\_SYNC

VGA\_BLANK, // VGA BLANK

VGA\_SYNC, // VGA SYNC

VGA\_R, // VGA Red[9:0]

VGA\_G, // VGA Green[9:0]

VGA\_B, // VGA Blue[9:0]

//////////// Ethernet Interface ////////////////////////

ENET\_DATA, // DM9000A DATA bus 16Bits

ENET\_CMD, // DM9000A Command/Data Select, 0 = Command, 1 = Data

ENET\_CS\_N, // DM9000A Chip Select

ENET\_WR\_N, // DM9000A Write

ENET\_RD\_N, // DM9000A Read

ENET\_RST\_N, // DM9000A Reset

ENET\_INT, // DM9000A Interrupt

ENET\_CLK, // DM9000A Clock 25 MHz

//////////////// Audio CODEC ////////////////////////

AUD\_ADCLRCK, // Audio CODEC ADC LR Clock

AUD\_ADCDAT, // Audio CODEC ADC Data

AUD\_DACLRCK, // Audio CODEC DAC LR Clock

AUD\_DACDAT, // Audio CODEC DAC Data

AUD\_BCLK, // Audio CODEC Bit-Stream Clock

AUD\_XCK, // Audio CODEC Chip Clock

//////////////// TV Decoder ////////////////////////

TD\_DATA, // TV Decoder Data bus 8 bits

TD\_HS, // TV Decoder H\_SYNC

TD\_VS, // TV Decoder V\_SYNC

TD\_RESET, // TV Decoder Reset

//////////////////// GPIO ////////////////////////////

GPIO\_0, // GPIO Connection 0

GPIO\_1 // GPIO Connection 1

);

//////////////////////// Clock Input ////////////////////////

input CLOCK\_27; // 27 MHz

input CLOCK\_50; // 50 MHz

input EXT\_CLOCK; // External Clock

//////////////////////// Push Button ////////////////////////

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

//////////////////////// DPDT Switch ////////////////////////

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

//////////////////////// 7-SEG Dispaly ////////////////////////

output [6:0] HEX0; // Seven Segment Digit 0

output [6:0] HEX1; // Seven Segment Digit 1

output [6:0] HEX2; // Seven Segment Digit 2

output [6:0] HEX3; // Seven Segment Digit 3

output [6:0] HEX4; // Seven Segment Digit 4

output [6:0] HEX5; // Seven Segment Digit 5

output [6:0] HEX6; // Seven Segment Digit 6

output [6:0] HEX7; // Seven Segment Digit 7

//////////////////////////// LED ////////////////////////////

output [8:0] LEDG; // LED Green[8:0]

output [17:0] LEDR; // LED Red[17:0]

//////////////////////////// UART ////////////////////////////

output UART\_TXD; // UART Transmitter

input UART\_RXD; // UART Receiver

//////////////////////////// IRDA ////////////////////////////

output IRDA\_TXD; // IRDA Transmitter

input IRDA\_RXD; // IRDA Receiver

/////////////////////// SDRAM Interface ////////////////////////

inout [15:0] DRAM\_DQ; // SDRAM Data bus 16 Bits

output [11:0] DRAM\_ADDR; // SDRAM Address bus 12 Bits

output DRAM\_LDQM; // SDRAM Low-byte Data Mask

output DRAM\_UDQM; // SDRAM High-byte Data Mask

output DRAM\_WE\_N; // SDRAM Write Enable

output DRAM\_CAS\_N; // SDRAM Column Address Strobe

output DRAM\_RAS\_N; // SDRAM Row Address Strobe

output DRAM\_CS\_N; // SDRAM Chip Select

output DRAM\_BA\_0; // SDRAM Bank Address 0

output DRAM\_BA\_1; // SDRAM Bank Address 0

output DRAM\_CLK; // SDRAM Clock

output DRAM\_CKE; // SDRAM Clock Enable

//////////////////////// Flash Interface ////////////////////////

inout [7:0] FL\_DQ; // FLASH Data bus 8 Bits

output [21:0] FL\_ADDR; // FLASH Address bus 22 Bits

output FL\_WE\_N; // FLASH Write Enable

output FL\_RST\_N; // FLASH Reset

output FL\_OE\_N; // FLASH Output Enable

output FL\_CE\_N; // FLASH Chip Enable

//////////////////////// SRAM Interface ////////////////////////

inout [15:0] SRAM\_DQ; // SRAM Data bus 16 Bits

output [17:0] SRAM\_ADDR; // SRAM Address bus 18 Bits

output SRAM\_UB\_N; // SRAM High-byte Data Mask

output SRAM\_LB\_N; // SRAM Low-byte Data Mask

output SRAM\_WE\_N; // SRAM Write Enable

output SRAM\_CE\_N; // SRAM Chip Enable

output SRAM\_OE\_N; // SRAM Output Enable

//////////////////// ISP1362 Interface ////////////////////////

inout [15:0] OTG\_DATA; // ISP1362 Data bus 16 Bits

output [1:0] OTG\_ADDR; // ISP1362 Address 2 Bits

output OTG\_CS\_N; // ISP1362 Chip Select

output OTG\_RD\_N; // ISP1362 Write

output OTG\_WR\_N; // ISP1362 Read

output OTG\_RST\_N; // ISP1362 Reset

output OTG\_FSPEED; // USB Full Speed, 0 = Enable, Z = Disable

output OTG\_LSPEED; // USB Low Speed, 0 = Enable, Z = Disable

input OTG\_INT0; // ISP1362 Interrupt 0

input OTG\_INT1; // ISP1362 Interrupt 1

input OTG\_DREQ0; // ISP1362 DMA Request 0

input OTG\_DREQ1; // ISP1362 DMA Request 1

output OTG\_DACK0\_N; // ISP1362 DMA Acknowledge 0

output OTG\_DACK1\_N; // ISP1362 DMA Acknowledge 1

//////////////////// LCD Module 16X2 ////////////////////////////

inout [7:0] LCD\_DATA; // LCD Data bus 8 bits

output LCD\_ON; // LCD Power ON/OFF

output LCD\_BLON; // LCD Back Light ON/OFF

output LCD\_RW; // LCD Read/Write Select, 0 = Write, 1 = Read

output LCD\_EN; // LCD Enable

output LCD\_RS; // LCD Command/Data Select, 0 = Command, 1 = Data

//////////////////// SD Card Interface ////////////////////////

inout SD\_DAT; // SD Card Data

inout SD\_DAT3; // SD Card Data 3

inout SD\_CMD; // SD Card Command Signal

output SD\_CLK; // SD Card Clock

//////////////////////// I2C ///////////////////////////////

inout I2C\_SDAT; // I2C Data

output I2C\_SCLK; // I2C Clock

//////////////////////// PS2 ////////////////////////////////

input PS2\_DAT; // PS2 Data

input PS2\_CLK; // PS2 Clock

//////////////////// USB JTAG link ////////////////////////////

input TDI; // CPLD -> FPGA (data in)

input TCK; // CPLD -> FPGA (clk)

input TCS; // CPLD -> FPGA (CS)

output TDO; // FPGA -> CPLD (data out)

//////////////////////// VGA ////////////////////////////

output VGA\_CLK; // VGA Clock

output VGA\_HS; // VGA H\_SYNC

output VGA\_VS; // VGA V\_SYNC

output VGA\_BLANK; // VGA BLANK

output VGA\_SYNC; // VGA SYNC

output [9:0] VGA\_R; // VGA Red[9:0]

output [9:0] VGA\_G; // VGA Green[9:0]

output [9:0] VGA\_B; // VGA Blue[9:0]

//////////////// Ethernet Interface ////////////////////////////

inout [15:0] ENET\_DATA; // DM9000A DATA bus 16Bits

output ENET\_CMD; // DM9000A Command/Data Select, 0 = Command, 1 = Data

output ENET\_CS\_N; // DM9000A Chip Select

output ENET\_WR\_N; // DM9000A Write

output ENET\_RD\_N; // DM9000A Read

output ENET\_RST\_N; // DM9000A Reset

input ENET\_INT; // DM9000A Interrupt

output ENET\_CLK; // DM9000A Clock 25 MHz

//////////////////// Audio CODEC ////////////////////////////

inout AUD\_ADCLRCK; // Audio CODEC ADC LR Clock

input AUD\_ADCDAT; // Audio CODEC ADC Data

inout AUD\_DACLRCK; // Audio CODEC DAC LR Clock

output AUD\_DACDAT; // Audio CODEC DAC Data

inout AUD\_BCLK; // Audio CODEC Bit-Stream Clock

output AUD\_XCK; // Audio CODEC Chip Clock

//////////////////// TV Devoder ////////////////////////////

input [7:0] TD\_DATA; // TV Decoder Data bus 8 bits

input TD\_HS; // TV Decoder H\_SYNC

input TD\_VS; // TV Decoder V\_SYNC

output TD\_RESET; // TV Decoder Reset

//////////////////////// GPIO ////////////////////////////////

inout [35:0] GPIO\_0; // GPIO Connection 0

inout [35:0] GPIO\_1; // GPIO Connection 1

///////////////////////////////////////////////////////////////////

//=============================================================================

// REG/WIRE declarations

//=============================================================================

// CCD

wire [11:0] CCD\_DATA;

wire CCD\_SDAT;

wire CCD\_SCLK;

wire CCD\_FLASH;

wire CCD\_FVAL;

wire CCD\_LVAL;

wire CCD\_PIXCLK;

wire CCD\_MCLK; // CCD Master Clock

wire [15:0] Read\_DATA1;

wire [15:0] Read\_DATA2;

wire VGA\_CTRL\_CLK;

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 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 [9:0] VGA\_R; // VGA Red[9:0]

wire [9:0] VGA\_G; // VGA Green[9:0]

wire [9:0] VGA\_B; // VGA Blue[9:0]

reg [1:0] rClk;

wire sdram\_ctrl\_clk;

//=============================================================================

// Structural coding

//=============================================================================

assign CCD\_DATA[0] = GPIO\_1[13];

assign CCD\_DATA[1] = GPIO\_1[12];

assign CCD\_DATA[2] = GPIO\_1[11];

assign CCD\_DATA[3] = GPIO\_1[10];

assign CCD\_DATA[4] = GPIO\_1[9];

assign CCD\_DATA[5] = GPIO\_1[8];

assign CCD\_DATA[6] = GPIO\_1[7];

assign CCD\_DATA[7] = GPIO\_1[6];

assign CCD\_DATA[8] = GPIO\_1[5];

assign CCD\_DATA[9] = GPIO\_1[4];

assign CCD\_DATA[10] = GPIO\_1[3];

assign CCD\_DATA[11] = GPIO\_1[1];

assign GPIO\_1[16] = CCD\_MCLK;

assign CCD\_FVAL = GPIO\_1[22];

assign CCD\_LVAL = GPIO\_1[21];

assign CCD\_PIXCLK = GPIO\_1[0];

assign GPIO\_1[19] = 1'b1; // tRIGGER

assign GPIO\_1[17] = DLY\_RST\_1;

/\*

assign LEDR = SW;

assign LEDG = Y\_Cont;

\*/

assign VGA\_CTRL\_CLK = rClk[0];

assign VGA\_CLK = ~rClk[0];

always@(posedge CLOCK\_50) rClk <= rClk+1;

always@(posedge CCD\_PIXCLK)

begin

rCCD\_DATA <= CCD\_DATA;

rCCD\_LVAL <= CCD\_LVAL;

rCCD\_FVAL <= CCD\_FVAL;

end

VGA\_Controller u1 ( // Host Side

.oRequest(Read),

.iRed(wDISP\_R),

.iGreen(wDISP\_G),

iBlue(wDISP\_B),

// VGA Side

.oVGA\_R(VGA\_R),

.oVGA\_G(VGA\_G),

.oVGA\_B(VGA\_B),

.oVGA\_H\_SYNC(VGA\_HS),

.oVGA\_V\_SYNC(VGA\_VS),

.oVGA\_SYNC(VGA\_SYNC),

.oVGA\_BLANK(VGA\_BLANK),

// Control Signal

.iCLK(VGA\_CTRL\_CLK),

.iRST\_N(DLY\_RST\_2)

);

Reset\_Delay u2 ( .iCLK(CLOCK\_50),

.iRST(KEY[0]),

.oRST\_0(DLY\_RST\_0),

.oRST\_1(DLY\_RST\_1),

.oRST\_2(DLY\_RST\_2)

);

CCD\_Capture u3 ( .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]),

.iEND(!KEY[2]),

.iCLK(CCD\_PIXCLK),

.iRST(DLY\_RST\_2)

);

RAW2RGB u4 ( .iCLK(CCD\_PIXCLK),

.iRST(DLY\_RST\_1),

.iDATA(mCCD\_DATA),

.iDVAL(mCCD\_DVAL),

.oRed(sCCD\_R),

.oGreen(sCCD\_G),

.oBlue(sCCD\_B),

.oDVAL(sCCD\_DVAL),

.iX\_Cont(X\_Cont),

.iY\_Cont(Y\_Cont)

);

SEG7\_LUT\_8 u5 ( .oSEG0(HEX0),.oSEG1(HEX1),

.oSEG2(HEX2),.oSEG3(HEX3),

.oSEG4(HEX4),.oSEG5(HEX5),

.oSEG6(HEX6),.oSEG7(HEX7),

.iDIG(Frame\_Cont[31:0])

);

sdram\_pll u6 (

.inclk0(CLOCK\_50),

.c0(sdram\_ctrl\_clk),

.c1(DRAM\_CLK)

);

assign CCD\_MCLK = rClk[0];

Sdram\_Control\_4Port u7 ( // HOST Side .REF\_CLK(CLOCK\_50), .RESET\_N(1'b1),

.CLK(sdram\_ctrl\_clk),

// FIFO Write Side 1

.WR1\_DATA({1'b0,sCCD\_G[11:7],sCCD\_B[11:2]}),

.WR1(sCCD\_DVAL),

.WR1\_ADDR(0),

.WR1\_MAX\_ADDR(640\*480),

.WR1\_LENGTH(9'h100),

.WR1\_LOAD(!DLY\_RST\_0),

.WR1\_CLK(~CCD\_PIXCLK),

// FIFO Write Side 2

.WR2\_DATA( {1'b0,sCCD\_G[6:2],sCCD\_R[11:2]}),

.WR2(sCCD\_DVAL),

.WR2\_ADDR(22'h100000),

.WR2\_MAX\_ADDR(22'h100000+640\*480),

.WR2\_LENGTH(9'h100),

.WR2\_LOAD(!DLY\_RST\_0),

.WR2\_CLK(~CCD\_PIXCLK),

// FIFO Read Side 1

.RD1\_DATA(Read\_DATA1),

.RD1(Read),

.RD1\_ADDR(0),

.RD1\_MAX\_ADDR(640\*480),

.RD1\_LENGTH(9'h100),

.RD1\_LOAD(!DLY\_RST\_0),

.RD1\_CLK(~VGA\_CTRL\_CLK),

// FIFO Read Side 2

.RD2\_DATA(Read\_DATA2),

.RD2(Read),

.RD2\_ADDR(22'h100000),

.RD2\_MAX\_ADDR(22'h100000+640\*480),

.RD2\_LENGTH(9'h100),

.RD2\_LOAD(!DLY\_RST\_0),

.RD2\_CLK(~VGA\_CTRL\_CLK),

// SDRAM Side

.SA(DRAM\_ADDR),

.BA({DRAM\_BA\_1,DRAM\_BA\_0}),

.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\_UDQM,DRAM\_LDQM})

);

assign UART\_TXD = UART\_RXD;

I2C\_CCD\_Config u8 ( // Host Side

.iCLK(CLOCK\_50),

.iRST\_N(DLY\_RST\_2),

.iZOOM\_MODE\_SW(SW[16]),

.iEXPOSURE\_ADJ(KEY[1]),

.iEXPOSURE\_DEC\_p(SW[0]),

// I2C Side

.I2C\_SCLK(GPIO\_1[24]),

.I2C\_SDAT(GPIO\_1[23])

);

wire [9:0] wVGA\_R = Read\_DATA2[9:0];

wire [9:0] wVGA\_G = {Read\_DATA1[14:10],Read\_DATA2[14:10]};

wire [9:0] wVGA\_B = Read\_DATA1[9:0];

//Coding

wire [9:0] gDATA;

wire gCCD\_DVAL;

wire wGFlag;

RGB2GRAY u9 (

.oDVAL(gCCD\_DVAL),

.oDATA(gDATA),

.oFlag(wGFlag),

.iRed(wVGA\_R),

.iGreen(wVGA\_G),

.iBlue(wVGA\_B),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(Read),

);

wire bDATA;

wire bCCD\_DVAL;

PupilBinaryImage u10 (

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(gCCD\_DVAL),

.iDATA(gDATA),

.oDATA(bDATA),

.oDVAL(bCCD\_DVAL),

);

wire oErosion;

wire eCCD\_DVAL;

Erosion u11 (

.iDATA(bDATA),

.iDVAL(bCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.oDATA(oErosion),

.oDVAL(eCCD\_DVAL),

);

wire oDilation;

wire dCCD\_DVAL;

Dilation u12 (

.iDATA(oErosion),

.iDVAL(eCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.oDATA(oDilation),

.oDVAL(dCCD\_DVAL),

);

wire [9:0] woC\_1;

wire [9:0] woR\_1;

wire [9:0] woC\_2;

wire [9:0] woR\_2;

wire [9:0] woC\_3;

wire [9:0] woR\_3;

wire [9:0] woC\_4;

wire [9:0] woR\_4;

wire [9:0] woC\_PC;

wire [9:0] woC\_PR;

wire [9:0] woPRadius;

wire wPFlagCoor;

wire oDetectPupil;

wire dpCCD\_DVAL;

Detect\_OuterPupil u13 (

.iEND(SW[17]),

.iDATA(oDilation),

.iDVAL(dCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.oC\_1(woC\_1), //top

.oR\_1(woR\_1),

.oC\_2(woC\_2),//left

.oR\_2(woR\_2),

.oC\_3(woC\_3),//right

.oR\_3(woR\_3),

.oC\_4(woC\_4),//btm

.oR\_4(woR\_4),

.oC\_PC(woC\_PC),

.oC\_PR(woC\_PR),

.oFlag(wPFlagCoor),

.oDATA(oDetectPupil),

.oDVAL(dpCCD\_DVAL),

.oPRadius(woPRadius),

);

wire [9:0]pPupilCoor;

wire ppCCD\_DVAL;

Output\_Point u14 (

.iDATA(ibDATA),

.iDVAL(ibCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iC\_1(woC\_1),

.iR\_1(woR\_1),

.iC\_2(woC\_2),

.iR\_2(woR\_2),

.iC\_3(woC\_3),

.iR\_3(woR\_3),

.iC\_4(woC\_4),

.iR\_4(woR\_4),

.iC\_PC(woC\_PC),

.iC\_PR(woC\_PR),

.iI\_R1(woI\_R1),

.iI\_C1(woI\_C1),

.iI\_R2(woI\_R2),

.iI\_C2(woI\_C2),

.oDVAL(ppCCD\_DVAL),

.oDATA(pPupilCoor),

);

wire ibData;

wire ibCCD\_DVAL;

IrisBinaryImage u15 (

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(gCCD\_DVAL),

.iDATA(gDATA),

.oDATA(ibDATA),

.oDVAL(ibCCD\_DVAL),

);

wire [9:0] woI\_R1;

wire [9:0] woI\_C1;

wire [9:0] woI\_R2;

wire [9:0] woI\_C2;

wire [9:0] woIRadius;

wire wIFlagCoor;

Detect\_InnerIris u16 (

.iDATA(ibDATA),

.iDVAL(ibCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iC\_PC(woC\_PC),

.iC\_PR(woC\_PR),

.oFlag(wIFlagCoor),

.oI\_R1(woI\_R1), //left

.oI\_C1(woI\_C1),

.oI\_R2(woI\_R2), //right

.oI\_C2(woI\_C2),

.oIRadius(woIRadius),

);

wire cgCCD\_DVAL;

wire [9:0] cgDATA;

GRAYCOPY u18 (

.oDVAL(cgCCD\_DVAL),

.oDATA(cgDATA),

.iRed(wVGA\_R),

.iGreen(wVGA\_G),

.iBlue(wVGA\_B),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(Read),

.iSignal(wSignal)

);

wire [9:0] wNDATA;

wire signalToCompress;

irisNormalization u17 (

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(cgCCD\_DVAL),

.iPupilFlag(wPFlagCoor),

.iIrisFlag(wIFlagCoor),

.iCaptured(!KEY[2]),

.iStart(!KEY[3]),

//.startGetData(wGFlag), //Receive signal from segmentation

.iDATA(cgDATA),

.pupil\_radius(woPRadius),

.iris\_radius(woIRadius),

.centre\_pointX(woC\_PC),

.centre\_pointY(woC\_PR),

.ograySignal(wSignal),

.oDATA(wNDATA),

.oSignal(signalToCompress),

);

assign LEDR[9] = signalToCompress;

assign LEDR[0] = wSignal;

wire arrayToSOM[0:9][0:9];

wire startSOM;

irisCompression compress (

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDATA(wNDATA),

.compressSignal(signalToCompress),

.oArray(arrayToSOM),

.oSignal(startSOM),

);

assign LEDR[10] = startSOM;

SOM som(

.CLOCK\_50(CLOCK\_50),

.iRST(DLY\_RST\_2),

.bRST(SW[7]),

.startRecognition(SW[17]),

.startTraining(SW[15]),

.trainUser(SW[14]),

.storeControl(SW[13]),

.storeUser(SW[12:9]),

.data(arrayToSOM),

.LED(LEDR[17:14]),

);

//output image

reg [9:0] pupilBinaryImage;

always@(posedge CLOCK\_50)begin

if(bDATA == 1)begin

pupilBinaryImage = 10'd1023;

end

else begin

pupilBinaryImage = 10'd0;

end

end

reg [9:0] erosionImage;

always@(posedge CLOCK\_50)begin

if(oErosion == 1)begin

erosionImage = 10'd1023;

end

else begin

erosionImage = 10'd0;

end

end

reg [9:0] dilationImage;

always@(posedge CLOCK\_50)begin

if(oDilation == 1)begin

dilationImage = 10'd1023;

end

else begin

dilationImage = 10'd0;

end

end

reg [9:0] irisBinaryImage;

always@(posedge CLOCK\_50)begin

if(ibDATA == 1)begin

irisBinaryImage = 10'd1023;

end

else begin

irisBinaryImage = 10'd0;

end

end

wire [9:0] wDISP\_R =

SW[6] ? pPupilCoor :

SW[5] ? irisBinaryImage:

SW[4] ? dilationImage :

SW[3] ? erosionImage :

SW[2] ? pupilBinaryImage:

SW[1] ? gDATA :

wVGA\_R;

wire [9:0] wDISP\_G =

SW[6] ? pPupilCoor :

SW[5] ? irisBinaryImage:

SW[4] ? dilationImage :

SW[3] ? erosionImage :

SW[2] ? pupilBinaryImage:

SW[1] ? gDATA :

wVGA\_G;

wire [9:0] wDISP\_B =

SW[6] ? pPupilCoor :

SW[5] ? irisBinaryImage:

SW[4] ? dilationImage :

SW[3] ? erosionImage :

SW[2] ? pupilBinaryImage:

SW[1] ? gDATA :

wVGA\_B;

endmodule

**Sub-system Code: Block 1**

module irisNormalization(iCLK,iRST,iDVAL,iPupilFlag,iIrisFlag,iCaptured,iStart,iDATA,

pupil\_radius,iris\_radius,

centre\_pointX,centre\_pointY,

oDVAL,oSignal,oDATA

);

//Input and Output

input iCLK;

input iRST;

input iDVAL;

input iPupilFlag;

input iIrisFlag;

input iCaptured;

input iStart;

//input startGetData; //Receive signal from segmentation

input [9:0] iDATA;

input [9:0] pupil\_radius;

input [9:0] iris\_radius;

input [9:0] centre\_pointX;

input [9:0] centre\_pointY;

output regoDVAL;

output regoSignal; // Give signal to matching

output reg [9:0] oDATA;

//Common Register

reg [9:0] row=0,col=0;

regiSignal=0,finalSignal=0;

reg unsigned [9:0] col\_range\_x=0 ,End\_Point=0,bottom\_pontY=0;

reg [9:0] tempData;

//Registers for sine and cosine

reg [6:0] counter1=1,counter2=1;

regfinishSinCos=0;

//Register for pupil

reg unsigned [9:0] pupil\_leftX,pupil\_leftY;

reg unsigned [9:0] pupil\_rightX=0,pupil\_rightY=0;

reg unsigned [9:0] pupil\_btmX=0,pupil\_btmY=0;

//Register for iris

reg unsigned [9:0] iris\_leftX=0,iris\_leftY=0;

regFlagStart = 0;

//Register for compression

//Register

reg signed [9:0] arrayOutput[0:9][0:9];

reg [30:0] sumData=0,sumData1=0,sumData2=0,sumData3=0,sumData4=0,sumData5=0,sumData6=0,sumData7=0,sumData8=0,sumData9=0;

reg [9:0] storageColData[0:79],storageColData1[0:79],storageColData2[0:79],storageColData3[0:79],storageColData4[0:79];

reg [9:0] storageColData5[0:79],storageColData6[0:79],storageColData7[0:79],storageColData8[0:79],storageColData9[0:79];

reg [9:0] col\_counter=0,col\_counter1=0,col\_counter2=0,col\_counter3=0,col\_counter4=0;

reg [9:0] col\_counter5=0,col\_counter6=0,col\_counter7=0,col\_counter8=0,col\_counter9=0;

//For case structure

reg [7:0] arrayAddr = 0;

//row is y-axis, col is x-axis

reg [9:0]cRow=0,cCol=0,y=0,x=0;

regtempSignal=0,tempSignal2=0;

regcalculateSinCosFlag = 0;

regcheckCOL = 0;

reg [9:0] angle=0,tempSinAngle=1,tempCosAngle=1;

reg signed[20:0]outputCos=0,tempCos=0;

reg signed[20:0]outputSin=0,tempSin=0;

task calculate\_sin\_angle;

if(tempSinAngle<=90)begin

if(tempSinAngle<=40)begin

//Sine

tempSin = 17000\*tempSinAngle;

if(tempSinAngle == 40)

tempSin = 16000\*tempSinAngle;

end

else if(tempSinAngle> 40 &&tempSinAngle<= 50)begin

//Sine

tempSin = 1000000-(145\*((tempSinAngle+270-360)\*\*2));

end

else if(tempSinAngle> 50 &&tempSinAngle<= 59)begin

//Sine

tempSin = 1000000-(145\*((tempSinAngle+270-360)\*\*2));

end

else if(tempSinAngle == 60)begin

//Sine

tempSin = 1000000-(149\*((tempSinAngle-90)\*\*2));

end

else if(tempSinAngle> 60 &&tempSinAngle<= 90)begin

//Sine

tempSin = 1000000-(145\*((tempSinAngle+270-360)\*\*2));

end

tempSinAngle=tempSinAngle+1;

end

endtask

task calculate\_cos\_angle;

if(tempCosAngle<=90)begin

if(tempCosAngle<=40)begin

//Cosine

tempCos = 1\*1000000-(145\*(tempCosAngle\*tempCosAngle));

end

else if(tempCosAngle> 40 &&tempCosAngle<= 50)begin

//Cosine

tempCos = 1000000-(145\*(tempCosAngle\*tempCosAngle));

end

else if(tempCosAngle> 50 &&tempCosAngle<= 59)begin

//Cosine

tempCos= 16250\*(90-tempCosAngle);

end

else if(tempCosAngle == 60)begin

//Cosine

tempCos = 16667\*(90-tempCosAngle);

end

else if(tempCosAngle> 60 &&tempCosAngle<= 90)begin

//Cosine

tempCos = 17000\*(90-tempCosAngle);

end

tempCosAngle = tempCosAngle+1;

end

endtask

//This task is use to find value of sine and cosine for 0 to 360 degree

task calculate\_COSSIN;

if(angle<=40)begin

//Sine

outputSin = 17000\*angle;

if(angle == 40)

outputSin = 16000\*angle;

//Cosine

outputCos = 1\*1000000-(145\*(angle\*angle));

end

else if(angle > 40 && angle <= 50)begin

//Sine

outputSin = 1000000-(145\*((angle+270-360)\*\*2));

//Cosine

outputCos = 1000000-(145\*(angle\*angle));

end

else if(angle > 50 && angle <= 59)begin

//Sine

outputSin = 1000000-(145\*((angle+270-360)\*\*2));

//Cosine

outputCos= 16250\*(90-angle);

end

else if(angle == 60)begin

//Sine

outputSin = 1000000-(149\*((angle-90)\*\*2));

//Cosine

outputCos = 16667\*(90-angle);

end

else if(angle > 60 && angle <= 90)begin

//Sine

outputSin = 1000000-(145\*((angle+270-360)\*\*2));

//Cosine

outputCos = 17000\*(90-angle);

end

else if(angle > 90 && angle <=180)begin

//Sine

calculate\_cos\_angle;

outputSin = tempCos;

//Cosine

calculate\_sin\_angle;

outputCos = -tempSin;

end

calculateSinCosFlag = 1;

endtask

task calculate\_points;

//pupil left coordinates

pupil\_leftX<= centre\_pointX+(pupil\_radius\*-1);

pupil\_leftY<= centre\_pointY+(pupil\_radius\*0);

//pupil right coordinates

pupil\_rightX<= centre\_pointX+(pupil\_radius\*1);

pupil\_rightY<= centre\_pointY+(pupil\_radius\*0);

//Get pupil bottom coordinate

pupil\_btmX<= centre\_pointX+(pupil\_radius\*0);

pupil\_btmY<= centre\_pointY+(pupil\_radius\*1);

//Get iris right coordinate

iris\_leftX<= centre\_pointX+(iris\_radius\*-1);

iris\_leftY<= centre\_pointX+(iris\_radius\*0);

iSignal<= 1;

endtask

task calculate\_range;

if(calculateSinCosFlag == 0)begin

calculate\_COSSIN;

end

if(calculateSinCosFlag == 1)begin

if(checkCOL == 0)begin

col\_range\_x = ((centre\_pointX\*1000000)+(iris\_radius\*outputCos))/1000000;

checkCOL = 1;

end

else begin

if(col\_range\_x>= (pupil\_rightX-5) &&col\_range\_x<= (pupil\_rightX+5))begin

End\_Point = ((centre\_pointY\*1000000)+(iris\_radius\*outputSin))/1000000;

finalSignal = 1;

end

else begin

angle = angle+1;

checkCOL = 0;

calculateSinCosFlag = 0;

end

end

end

endtask

task get\_data\_in\_range;

if(row < 480 )begin

if(col < 640)begin

if(row >= pupil\_btmY&& row <= End\_Point)begin

if(col >= pupil\_leftX&& col <= pupil\_rightX)begin

oDATA<= iDATA;

oSignal<= 1;

end

else begin

oDATA<= 10'd0;

end

end

else begin

oDATA<= 10'd0;

end

col <= col+1;

end

if(col==640)begin

row <= row+1;

col <= 0;

end

end

if(row == 480)begin

row <= 0;

col <= 0;

end

endtask

always@(posedgeiCLK or negedgeiRST)begin

if(!iRST)begin

FlagStart<= 0;

end

else begin

if(iCaptured)

FlagStart<= 1;

if(iStart)

FlagStart<= 0;

end

end

//Main module

always@(posedgeiCLK or negedgeiRST)begin

if(!iRST)begin

oDATA<= 0;

oDVAL<=0;

calculateSinCosFlag<= 0;

checkCOL<= 0;

finalSignal<= 0;

row <= 0;

col <= 0;

oSignal<= 0;

end

else begin

//oDVAL<=iDVAL;

if(FlagStart == 1)begin

if(iPupilFlag == 1 &&iIrisFlag == 1)begin

calculate\_points;

if(iSignal==1)begin

calculate\_range;

if(finalSignal==1)begin

oDVAL<=iDVAL;

if(iDVAL==1)begin

get\_data\_in\_range;

end

end

end

end

end

end

end

endmodule

**Sub-system Code: Block 2**

module irisCompression(iCLK,iRST,iDATA,compressSignal,oArray,oSignal);

input iCLK;

input iRST;

input [9:0] iDATA;

input compressSignal;

output regoArray[0:9][0:9];

output regoSignal;

//Parameter

parameter thresholdValue = 10'd6;

//Register

reg [9:0] tempOutput[0:9][0:9];

regtempArray[0:9][0:9];

reg [9:0] arrayOutput[0:9][0:9];

reg [15:0] sumData=0,sumData1=0,sumData2=0,sumData3=0,sumData4=0,sumData5=0,sumData6=0,sumData7=0,sumData8=0,sumData9=0;

reg [9:0] storageColData[0:7],storageColData1[0:7],storageColData2[0:7],storageColData3[0:7],storageColData4[0:7];

reg [9:0] storageColData5[0:7],storageColData6[0:7],storageColData7[0:7],storageColData8[0:7],storageColData9[0:7];

reg [4:0] col\_counter=0,col\_counter1=0,col\_counter2=0,col\_counter3=0,col\_counter4=0;

reg [4:0] col\_counter5=0,col\_counter6=0,col\_counter7=0,col\_counter8=0,col\_counter9=0;

//row is y-axis, col is x-axis

reg [9:0]row=0,col=0;

reg [3:0]y=0,x=0;

regbitSignal;

regstopFlag = 1;

always@(posedgeiCLK or negedgeiRST)begin

if(!iRST)begin

oSignal<=0;

sumData<=0;

sumData1<=0;

sumData2<=0;

sumData3<=0;

sumData4<=0;

sumData5<=0;

sumData6<=0;

sumData7<=0;

sumData8<=0;

sumData9<=0;

col\_counter<=0;

col\_counter1<=0;

col\_counter2<=0;

col\_counter3<=0;

col\_counter4<=0;

col\_counter5<=0;

col\_counter6<=0;

col\_counter7<=0;

col\_counter8<=0;

col\_counter9<=0;

row<=0;

col<=0;

y<=0;

x<=0;

bitSignal<=0;

stopFlag = 1;

end

else begin

if(compressSignal==1)begin

if(stopFlag == 1)begin

if(row<80)begin

if(col<120)begin

if(col<12)begin

//totalData in x-axis

sumData = sumData+iDATA;

col=col+1;

if(col==12)begin

if(col\_counter<8)begin

storageColData[col\_counter]=sumData/12; //0

col\_counter=col\_counter+1;

end

sumData=0;

end

end

else if(col>=12 && col<24)begin

sumData1 = sumData1+iDATA;

//$display("sumData1 = %d",sumData1);

col=col+1;

if(col==24)begin

if(col\_counter1<8)begin

storageColData1[col\_counter1]=sumData1/12; //1

col\_counter1=col\_counter1+1;

end

sumData1=0;

end

end

else if(col>=24 && col<36)begin

sumData2 = sumData2+iDATA;

col=col+1;

if(col==36)begin

if(col\_counter2<8)begin

storageColData2[col\_counter2]=sumData2/12; //2

col\_counter2=col\_counter2+1;

end

sumData2=0;

end

end

else if(col>=36 && col<48)begin

sumData3 = sumData3+iDATA;

col=col+1;

if(col==48)begin

if(col\_counter3<8)begin

storageColData3[col\_counter3]=sumData3/12; //3

col\_counter3=col\_counter3+1;

end

sumData3=0;

end

end

else if(col>=48 && col<60)begin

sumData4 = sumData4+iDATA;

col=col+1;

if(col==60)begin

if(col\_counter4<8)begin

storageColData4[col\_counter4]=sumData4/12; //4

col\_counter4=col\_counter4+1;

end

sumData4=0;

end

end

else if(col>=60 && col<72)begin

sumData5 = sumData5+iDATA;

col=col+1;

if(col==72)begin

if(col\_counter5<8)begin

storageColData5[col\_counter5]=sumData5/12; //5

col\_counter5=col\_counter5+1;

end

sumData5=0;

end

end

else if(col>=72 && col<84)begin

sumData6 = sumData6+iDATA;

col=col+1;

if(col==84)begin

if(col\_counter6<8)begin

storageColData6[col\_counter6]=sumData6/12; //6

col\_counter6=col\_counter6+1;

end

sumData6=0;

end

end

else if(col>=84 && col<96)begin

sumData7 = sumData7+iDATA;

col=col+1;

if(col==96)begin

if(col\_counter7<8)begin

storageColData7[col\_counter7]=sumData7/12; //7

col\_counter7=col\_counter7+1;

end

sumData7=0;

end

end

else if(col>=96 && col<108)begin

sumData8 = sumData8+iDATA;

col=col+1;

if(col==108)begin

if(col\_counter8<8)begin

storageColData8[col\_counter8]=sumData8/12; //8

col\_counter8=col\_counter8+1;

end

sumData8=0; end

end

else if(col>=108 && col<120)begin

sumData9 = sumData9+iDATA;

col=col+1;

if(col==120)begin

if(col\_counter9<8)begin

storageColData9[col\_counter9]=sumData9/12; //9

col\_counter9=col\_counter9+1;

end

sumData9=0;

end

end

if(row == 7 && col == 120)begin

arrayOutput[0][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[0][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[0][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[0][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[0][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[0][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[0][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[0][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[0][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[0][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row == 15 && col == 120)begin

arrayOutput[1][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[1][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[1][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[1][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[1][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[1][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[1][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[1][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[1][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[1][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row == 23 && col ==120)begin

arrayOutput[2][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[2][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[2][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[2][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[2][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[2][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[2][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[2][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[2][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[2][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row ==31 && col ==120)begin

arrayOutput[3][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[3][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[3][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[3][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[3][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[3][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[3][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[3][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[3][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[3][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row ==39 && col ==120)begin

arrayOutput[4][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[4][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[4][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[4][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[4][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[4][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[4][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[4][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[4][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[4][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row ==47 && col ==120)begin

arrayOutput[5][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[5][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[5][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[5][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[5][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[5][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[5][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[5][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[5][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[5][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row == 55 && col ==120)begin

arrayOutput[6][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[6][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[6][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[6][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[6][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[6][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[6][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[6][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[6][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[6][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row ==63 && col ==120)begin

arrayOutput[7][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[7][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[7][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[7][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[7][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[7][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[7][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[7][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[7][8] = (storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+storageColData8[5]+

storageColData8[6]+storageColData8[7])/8;

arrayOutput[7][9] = (storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+storageColData9[5]+

storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row == 71 && col==120 )begin

arrayOutput[8][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[8][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[8][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[8][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[8][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[8][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[8][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[8][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[8][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[8][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

else if(row == 79 && col ==120)begin

arrayOutput[9][0]=(storageColData[0]+storageColData[1]+storageColData[2]+storageColData[3]+storageColData[4]+

storageColData[5]+storageColData[6]+storageColData[7])/8;

arrayOutput[9][1]=(storageColData1[0]+storageColData1[1]+storageColData1[2]+storageColData1[3]+storageColData1[4]+

storageColData1[5]+storageColData1[6]+storageColData1[7])/8;

arrayOutput[9][2]=(storageColData2[0]+storageColData2[1]+storageColData2[2]+storageColData2[3]+storageColData2[4]+

storageColData2[5]+storageColData2[6]+storageColData2[7])/8;

arrayOutput[9][3]=(storageColData3[0]+storageColData3[1]+storageColData3[2]+storageColData3[3]+storageColData3[4]+

storageColData3[5]+storageColData3[6]+storageColData3[7])/8;

arrayOutput[9][4]=(storageColData4[0]+storageColData4[1]+storageColData4[2]+storageColData4[3]+storageColData4[4]+

storageColData4[5]+storageColData4[6]+storageColData4[7])/8;

arrayOutput[9][5]=(storageColData5[0]+storageColData5[1]+storageColData5[2]+storageColData5[3]+storageColData5[4]+

storageColData5[5]+storageColData5[6]+storageColData5[7])/8;

arrayOutput[9][6]=(storageColData6[0]+storageColData6[1]+storageColData6[2]+storageColData6[3]+storageColData6[4]+

storageColData6[5]+storageColData6[6]+storageColData6[7])/8;

arrayOutput[9][7]=(storageColData7[0]+storageColData7[1]+storageColData7[2]+storageColData7[3]+storageColData7[4]+

storageColData7[5]+storageColData7[6]+storageColData7[7])/8;

arrayOutput[9][8]=(storageColData8[0]+storageColData8[1]+storageColData8[2]+storageColData8[3]+storageColData8[4]+

storageColData8[5]+storageColData8[6]+storageColData8[7])/8;

arrayOutput[9][9]=(storageColData9[0]+storageColData9[1]+storageColData9[2]+storageColData9[3]+storageColData9[4]+

storageColData9[5]+storageColData9[6]+storageColData9[7])/8;

col\_counter=0;col\_counter1=0;col\_counter2=0;col\_counter3=0;col\_counter4=0;col\_counter5=0;col\_counter6=0;

col\_counter7=0;col\_counter8=0;col\_counter9=0;

end

if(col==120)begin

col=0;

row=row+1;

end

end

end

else begin

bitSignal =1;

end

end

if(bitSignal == 1)begin

if(y<10)begin

if(x<10)begin

tempOutput[y][x]=arrayOutput[y][x][0]+arrayOutput[y][x][1]+arrayOutput[y][x][2]+

arrayOutput[y][x][3]+arrayOutput[y][x][4]+arrayOutput[y][x][5]+arrayOutput[y][x][6]+

arrayOutput[y][x][7]+arrayOutput[y][x][8]+arrayOutput[y][x][9];

if(tempOutput[y][x] >= thresholdValue)begin

tempArray[y][x] = 1;

end

else begin

tempArray[y][x] = 0;

end

x=x+1;

if(x==10)begin

y=y+1;

x=0;

end

end

end

else if(y>9)begin

oSignal = 1;

oArray = tempArray;

stopFlag = 0;

end

end

end

end

end

endmodule