Design and VLSI Implementation of High Performance Face Recognition System

A report submitted to department of Electrical & Electronic Engineering, BRAC University in partial fulfillment of the requirements for thesis work.

Priyanka Das Dewan - 10221078

Tasnim Harun Shamma – 09221032

Afifa Abbas - 10221073

Raktim Kumar Mondol - 09221232

April 2013

Declaration

We do hereby declare that the thesis titled "Design and VLSI Implementation of High Performance Face Recognition System" is submitted to the Department of Electrical and Electronics Engineering of BRAC University in partial fulfillment of the Bachelor of Science in Electronics and Electrical Engineering. This is our original work and was not submitted elsewhere for the award of any other degree or any other publication.

Date:

Supervisor

_____

Professor Dr. A. B. M Harun- Ur- Rashid

_____

(Priyanka Das Dewan)
Student's ID: 10221078
priyanka.bracu@gmail.com

_____

(Tasnim Harun Shamma)
Student ID: 09221032
tasnim.h.shamma@gmail.com

_____

(Afifa Abbas)
Student ID: 10221073
afifa.abbas118@gmail.com

_____

(Raktim Kumar Mondol)
Student ID: 09221232
raktim.live@gmail.com

Abstract:

In this paper, we have proposed a novel hardware architecture for face-recognition system. In order to make the system cost effective we have used a simple yet efficient algorithm of face-recognition system. We have designed, implemented and verified the algorithm in a cyclone III Field Programmable Gate Array (FPGA) chip. Altera DE0 development board which contains a cyclone III chip on it have been used for debugging purpose. We have also ensured for low power consumption such that the chip could be used universally in a wide range of security systems.

To develop a simple yet efficient face recognition algorithm (such as PCA, FFT etc.) on digital hardware, we have researched on various face recognition algorithms using Matlab codes and studied their detection efficiency under various posture and background and also the complexity of the algorithm. To save hardware resource and at the same time to obtain an acceptable level of recognition we have chosen to use Fast Fourier Transform.

The search database is developed by taking pictures of BRAC University students in various background and postures and used them to evaluate the developed face recognition system. Images were captured using TRDB_D5M camera module and digital data from the camera was transferred to the SDRAM of the DE0 board using GPIO interface. A NIOS2 microprocessor was synthesized in the cyclone III chip which controlled the total recognition system and the communication between the FFT core, SDRAM and On-chip memory. The performance of the hardware is now under evaluation.

Keywords:

FFT, FPGA, Face Recognition, Nios2, TRDB_D5M.

Preface:

One of the most important reasons for choosing this task as our undergraduate thesis is that it gave us the possibility to use the theory and knowledge that we have gained over the years to make something useful and practical. We also believed that the design task would be good preparation for the future challenges. We have always been fascinated by electronics and the wide area of application this technology presents. Since our interest include both VLSI and working with FPGA, this project became a great opportunity to combine our interest and education.

The reason behind choosing FPGA is that in our country very few people worked with this board and we took it as a challenge. This challenge was the most effective way to learn new things. We have learned a lot about image processing, DE0 board, TRDB_D5M camera, Quartus12.0 and NIOSII. The Altera DE0 platform was a very good platform to work with. Many projects can be implemented using this board.

Acknowledgements:

This thesis is submitted to BRAC University in partial fulfillment of the requirements for the degree of bachelor in Electrical and Electronic Engineering.

We are thankful to our Almighty Allah for his blessings upon us and bestowing us courage to go with such task, and also our parents for their support, love and patience.

This project would not have been possible without all the help and the support we have received. We would like to thank our supervisor, Professor Dr. A. B. M Harun- Ur- Rashid (Bangladesh University of Engineering and Technology) for all the help, support, fresh perspectives we have got. We are also thankful to Sir Jahangir Alam , Lecturer of BRAC University for his constant guidance, when we were stuck. We highly appreciate the assistance and guidance of Shafiur Rahman, student of BUET and Ahsan Ashfaq, an Alumni of Halmstad University, Sweden, throughout the process.

April 13, 2013.

Table of Contents

Table List

Figure List

Chapter 1

Introduction:
1.1 Background

A facial recognition system is a computer application for automatically identifying or verifying a person from digital image or a video frame from a video source. Therefore there are two types of approaches for face recognition. One is image based and another one is video- based. There are more classifications to it now. One is partially automated systems and the other is fully automates systems.

Face Recognition has become a well-liked and popular area of research in image analysis, understanding and in computer vision as well. This topic has raised curiosity among computer science researchers, neurologists and psychologists.

Basically, face recognition in our case is given still images of a person; it can verify and identify one or more persons using a stored database of faces. For the research help purpose already many databases have been created. As for example AT&T face database, Yale face databases etcetera comprising of different poses and illumination conditions. Many universities and institutions have shown interest in this image processing and recognition systems from very early time and still aspire to excel in this field.

Recognition algorithms is divided into two main categories or approached in two different ways. One is geometric another one is photometric. Geometric approach focuses on distinguished features of a face and photometric approaches statistically, that distils an image into some values and these values are compared with some templates so that variances are eliminated.

The researches' directions include recognition from outdoor images, non-frontal facial images, increased understanding of the effects of demographic factors on the performance, develop improved models for predicting identification performance on very large galleries and many more [1].

1.2 Motivation and objectives:

After extensive research in the field of face recognition [35] [36], we discovered that none of the projects included FFT as a face recognition algorithm. We have designed the hardware architecture from a new perspective. Most of the available algorithms are implemented in software .As a result, the recognition speed is not as expected. On the other hand, hardware implementation has many promises. Therefore, we emphasized on hardware implementation. The improvement includes robustness of the speed and accuracy of the system. An FPGA can provide us necessary resources to achieve such improvements in face recognition. The resources includes built in blocks, various communication interfaces, millions of logic gates, scopes to run C codes into the digital hardware circuitry, high level design tools, performance, long term maintenance, reliability etcetera.

The objective of our project is to work with still image based algorithm and to implement it on a cyclone III FPGA chip from Altera Inc. The cyclone chip is relatively cheaper and includes ROM. DE0 board has been chosen as a tool for debugging process. We have emphasized on using FFT. Since FPGA implementation itself is a huge challenge, we will start with a simpler function that is FFT (Fast Fourier Transform). We have used the Cooley-Tukey algorithm for FFT. In addition, we have gone through for a hardware/software co-design approach. Our aim was to do the whole recognition in Matlab using FFT and PCA, verified that if they had worked properly, then we compared the algorithms. After that we focused on implementing FFT first on the board using Nios2 processors, SDRAM, on chip memory, DMA blocks etcetera.

1.3 Research Goal:

Our research goal is to get acquainted with FPGA board, to learn how to use it. On the other hand, our goal was to enter into the huge area of image processing. Combining these two fields together can definitely broaden our knowledge. One of the prime concerns of our research is to start with the simpler algorithm, to confirm that it is possible to implement any other algorithm using FPGA, so that we can work on it in future. FPGA itself is complex device. Therefore we couldn't take our goal to the benchmark. Hopefully we will learn from mistakes and can go for further algorithms.

1.4 Problem Formulation:

The first approach we had was to create our own small database with the images of our university students comprising different facial expressions. Then we attempted to apply some of the recognized algorithms on our database using MATLAB. Then we used our proposed algorithm using Fast Fourier Transform to assess the feasibility of using FFT as a face recognition algorithm. We afterwards moved towards hardware part which was our main interest. We used the FPGA board with the digital camera that is compatible with the board. We took images and stored them in the board's SDRAM. We then took these images from the memory and applied FFT on them and kept the transformed images to the SDRAM again. The next step was to compare between the values of the transformed images to verify if our algorithm was working.

Chapter 2

2.1 Algorithms for face recognition

2.1.1 Principle component Analysis:

Principle component Analysis (PCA) was invented in 1901 by Karl Pearson. This algorithm consists extracting relevant information in a face image which is called the principle component and encode that information in a suitable data structure. For recognition it takes the sample image and encodes it in the same way and compares it with the set of encoded images. In mathematical terms we want to find Eigen vectors and Eigen values of a covariance matrix of images, where one image is just a single point in high dimensional space [n *n], where n*n are the dimensions of an image. There can be many Eigen vectors for a covariance matrix but very few of them are principle one's. Each Eigen vector can be used for finding different amount of variations among the face image. However we are emphasizing only in principle Eigen vectors because these can show account for substantial variations among a bunch of images. They can show the most significant relationship between the data dimensions. Eigenvectors with highest Eigen values are the principle component of the image set. We may lose some information if we ignore components of lesser significance. But if the Eigen values are small then we won't lose much. Using those set of Eigen vectors we can construct Eigen faces.

The goal of PCA is to reduce the dimensionality of the data while retaining as much as possible of the variation present in the original dataset. PCA allows us to compute a linear transformation that maps data from high dimensional space to low dimensional sub-space. [1] [2].

$$
\begin{aligned}
b_1 &= t_{11}a_1 + t_{12}a_2 + \ldots + t_{1n}a_N \\
b_2 &= t_{21}a_1 + t_{22}a_2 + \ldots + t_{2n}a_N \\
&\cdots \\
b_K &= t_{K1}a_1 + t_{K2}a_2 + \ldots + t_{KN}a_N
\end{aligned}
$$

$$
\text{or } y = Tx \text{ where } T = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1N} \\ t_{21} & t_{22} & \cdots & t_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ t_{K1} & t_{K2} & \cdots & t_{KN} \end{bmatrix}
$$

- Lower dimensionality basis

- Approaximate vectors by finding a basis in an approapriate lower dimensional space.

(1) Higher dimensional space representation:

$$x = a_1 v_1 + a_2 v_2 + \cdots + a_N v_N$$

$v_1, v_2, ..., v_N$ is a basis of the N-dimensional space

(2)Lower dimensional space representation:

$$\hat{x} = b_1 u_1 + b_2 u_2 + \cdots + b_K u_K$$

$u_1, u_2, ..., u_K$ is a basis of the K-dimensional space

- *Note:* if both bases have the same size ($N = K$), then $x = \hat{x}$)

Example:

$$v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{(standard basis)}$$

$$x_v = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} = 3v_1 + 3v_2 + 3v_3$$

$$u_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, u_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, u_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{(some other basis)}$$

$$x_u = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} = 0u_1 + 0u_2 + 3u_3$$

$$\text{thus, } x_v = x_u$$

- Methodology

- Suppose $x_1, x_2, ..., x_M$ are N x 1 vectors

Step 1: $\bar{x} = \frac{1}{M} \sum_{i=1}^{M} x_i$

Step 2: subtract the mean: $\Phi_i = x_i - \bar{x}$

Step 3: form the matrix $A = [\Phi_1 \; \Phi_2 \cdots \Phi_M]$ (NxM matrix), then compute:

$$C = \frac{1}{M} \sum_{n=1}^{M} \Phi_n \Phi_n^T = AA^T$$

(sample **covariance** matrix, NxN, characterizes the *scatter* of the data)

Step 4: compute the eigenvalues of C: $\boldsymbol{\lambda}_1 > \boldsymbol{\lambda}_2 > \cdots > \boldsymbol{\lambda}_N$

Step 5: compute the eigenvectors of C: $u_1, u_2, \ldots, u_N$

- Since C is symmetric, $u_1, u_2, \ldots, u_N$ form a basis, (i.e., any vector x or actually $(x - \bar{x})$, can be written as a linear combination of the eigenvectors):

$$x - \bar{x} = b_1 u_1 + b_2 u_2 + \cdots + b_N u_N = \sum_{i=1}^{N} b_i u_i$$

Step 6: (**dimensionality reduction step**) keep only the terms corresponding to the K largest eigenvalues:

$$\hat{x} - \bar{x} = \sum_{i=1}^{K} b_i u_i \text{ where } K << N$$

- The representation of $\hat{x} - \bar{x}$ into the basis $u_1, u_2, ..., u_K$ is thus

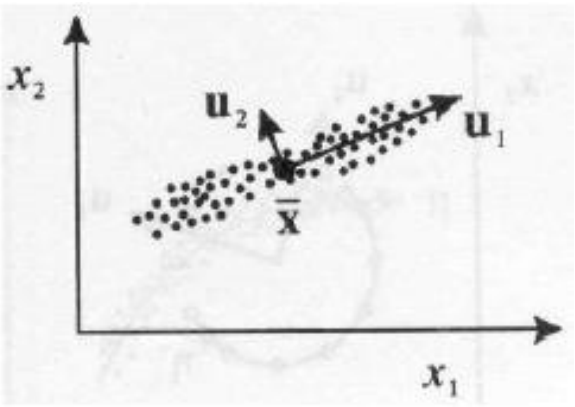
$$\begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_K \end{bmatrix}$$

- Linear transformation implied by PCA

- The linear transformation $R^N \rightarrow R^K$ that performs the dimensionality reduction is:

$$\begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_K \end{bmatrix} = \begin{bmatrix} u_1^T \\ u_2^T \\ ... \\ u_K^T \end{bmatrix} (x - \bar{x}) = U^T (x - \bar{x})$$

- Geometric Interpretation:
  - PCA projects the data along the directions where the data varies the most.
  - These directions are determined by the eigenvectors of the covariance matrix corresponding to the largest Eigenvalues.
  - The magnitude of the Eigenvalues corresponds to the variance of the data along the eigenvector directions.



- Main Idea behind Eigenfaces:

- Suppose Γ is an N^2x1 vector, corresponding to an NxN face image I.

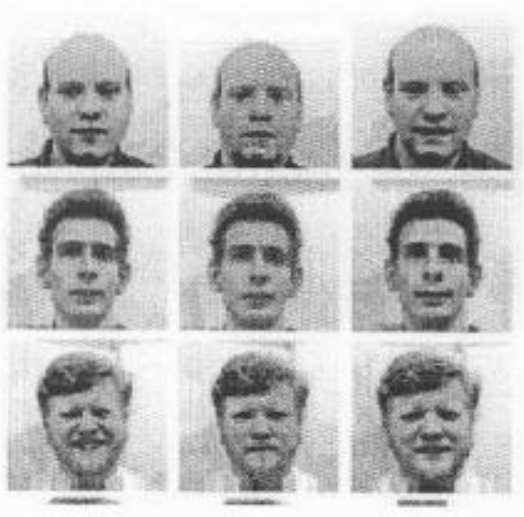
- The idea is to represent Γ (Φ=Γ - mean face) into a low-dimensional space:

$$\hat{\Phi} - mean = w_1 u_1 + w_2 u_2 + \cdots w_K u_K \ (K \ll N^2)$$

- Computaion of the eigenfaces:

Step 1: obtain face images $I_1, I_2, ..., I_M$ (training faces)

(**very important:** the face images must be *centered* and of the same *size*)



Step 2: represent every image I_i as a vector Γ_i

Step 3: compute the average face vector Ψ:

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$$

Step 4: subtract the mean face:

$$\Phi_i = \Gamma_i - \Psi$$

Step 5: compute the covariance matrix C:

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T \quad (N^2 \text{x} N^2 \text{ matrix})$$

$$\text{where } A = [\Phi_1\ \Phi_2 \cdots \Phi_M] \quad (N^2 \text{x} M \text{ matrix})$$

Step 6: compute the eigenvectors u_i of AA^T

The matrix AA^T is very large --> not practical !!

Step 6.1: consider the matrix $A^T A$ (M x M matrix)

Step 6.2: compute the eigenvectors v_i of $A^T A$

$$A^T A v_i = \mu_i v_i$$

What is the relationship between us_i and v_i?

$A^T A v_i = \mu_i v_i$ => $AA^T A v_i = \mu_i A v_i$ =>

$CAv_i = \mu_i Av_i$ or $Cu_i = \mu_i u_i$ where $u_i = Av_i$

Thus, AA^T and $A^T A$ have the same eigenvalues and their eigenvectors are related as follows: $u_i = Av_i$!!

Note 1: AA^T can have up to N^2 eigenvalues and eigenvectors.

Note 2: $A^T A$ can have up to M eigenvalues and eigenvectors.

Note 3: The M eigenvalues of $A^T A$ (along with their corresponding eigenvectors) correspond to the M *largest* eigenvalues of AA^T (along with their corresponding eigenvectors).

Step 6.3: compute the M best eigenvectors of AA^T: $u_i = Av_i$

(**important:** normalize u_i such that $\|u_i\| = 1$)

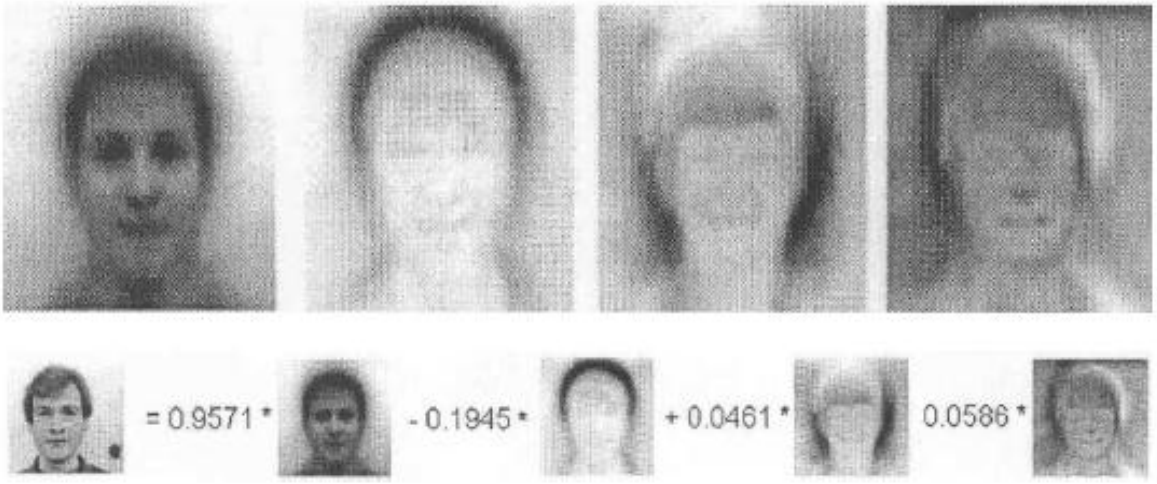
Step 7: keep only K eigenvectors (corresponding to the K largest eigenvalues)

- Representing faces on to this basis

- Each face (minus the mean) Φ_i in the training set can be represented as a linear combination of the best K eigenvectors:

$$\hat{\Phi}_i - mean = \sum_{j=1}^{K} w_j u_j, \quad (w_j = u_j^T \Phi_i)$$

(we call the u_j's *eigenfaces*)



- Each normalized training face Φ_i is represented in this basis by a vector:

$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ \dots \\ w_K^i \end{bmatrix}, \quad i = 1, 2, \dots, M$$

- Face Recognition Using Eigenfaces

- Given an unknown face image Γ (centered and of the same size like the training faces) follow these steps:

Step 1: normalize Γ: $\Phi = \Gamma - \Psi$

Step 2: project on the eigenspace

$$\hat{\Phi} = \sum_{i=1}^{K} w_i u_i \quad (w_i = u_i^T \Phi)$$

Step 3: represent Φ as: $\Omega = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_K \end{bmatrix}$

Step 4: find $e_r = \min_l \|\Omega - \Omega^l\|$

Step 5: if $e_r < T_r$, then Γ is recognized as face l from the training set.

- The distance e_r is called *distance within the face space* (*difs*)
- Comment: we can use the common Euclidean distance to compute e_r, however, it has been reported that the *Mahalanobis distance* performs better:

$$\|\Omega - \Omega^k\| = \sum_{i=1}^{K} \frac{1}{\lambda_i} (w_i - w_i^k)^2$$

(variations along all axes are treated as equally significant)

In the recognition phase, a subject face is normalized with respect to the average face and then projected onto face space using the eigenvector matrix. Next, the Euclidean distance is computed between this projection and all known projections. The minimum value of these comparisons is selected ans compared with the threshold calculated during the training phase. Based on this, if the value is greater than the threshold, the face is new. Otherwise, it is a known face.

2.1.2 LDA :

Linear discriminant analysis (LDA) is another effective algorithm for face recognition. It is closely related to PCA and factor analysis in that they both look for linear combination of varaibles whice best explain the data. LDA explicitly attempts to model the difference between the classes of data. PCA on the other hand doesnot take into account any difference in class, and factor analysis builds the feature combinations based on differences rather than similarities. The face space created in LDA gives higher weight to the variations between individuals than those of the same individual. LDA is less sensitive than the phase spectrum. Indeed, it is the phase spectrum that contains information which humans use to identify faces. [4].

2.1.3 ICA:

As PCA considers the 2^{nd} order moments only it lacks information on higher order statistics. The Independent Component Analysis (ICA) accounts for higher order statistics and it identifies the independent source components from their linear mixtures. ICA thus provides a more powerful data representation than PCA [5] as its goal is that of providing an independent image rather than uncorrelated image decomposition and representation. ICA of a random vector searches for a linear transformation which minimizes the statistical dependence between its components [6]. ICA represents the input as an n-dimensional random vector. This random vector is then reduced using PCA, without losing the higher order statistics. Then, the ICA algorithm finds the covariance matrix of the result and obtain its factorized form. Finally, whitening , rotation and normalization are performed to obtain the Independent components that constitute the fce space of the individuals. Since the higher order relationships between pixels are used, ICA is robust in the presence of noise. Thus, recognition is less sensitive to “lighting conditions, changes in hair, make-up and facial ecxpressions” [7].

2.1.4 Trace Transform:

The Trace transform [8], a generalizarion of the Radon transform, is a new tool for image processing which can be used for recognition objects under transformations, rotations, translation and scaling. To produce the Trace Transform one computes a functional along tracing lines of an image. Each line is characterized by two parameters, namely its distance

from the centre of the axes and the orientation.The trace transform is a global transform, applicable to full images. If we are going to use it to recognize faces, we must consider the local version of it. One of the key properties of the Trace transform is that it can be used to construct features invariant to rotation, translation and scaling. We should point out that invariance to rotation and scaling is harder to achieve than invariance to translation. It is assumed that an object is subjected to linear distortions like rotations,translations, and scaling. It is equivalent to saying that the image remains the same but viewed from the linearly distorted coordinate system .

2.1.5 Neural Network:

A Neural Network is a system of programs and data structures that approximates the operation of the human brain. A neural network usually involves a large number of processors operating in parallel, each with its small sphere of knowledge and access to data in its local memory. Typically, a neural network is initially trained or fed large amounts of data and rules according to the data. A program can tell the network how to behave in response to an external stimulus or can initiate activity on its own. The main disadvantage of neural networks is that there is no clear method to find the initial topologies. The training takes long time. For face recognition, a neural network must be trained to recognize an individual. That is time consuming and not well suited for real-time applications [9].

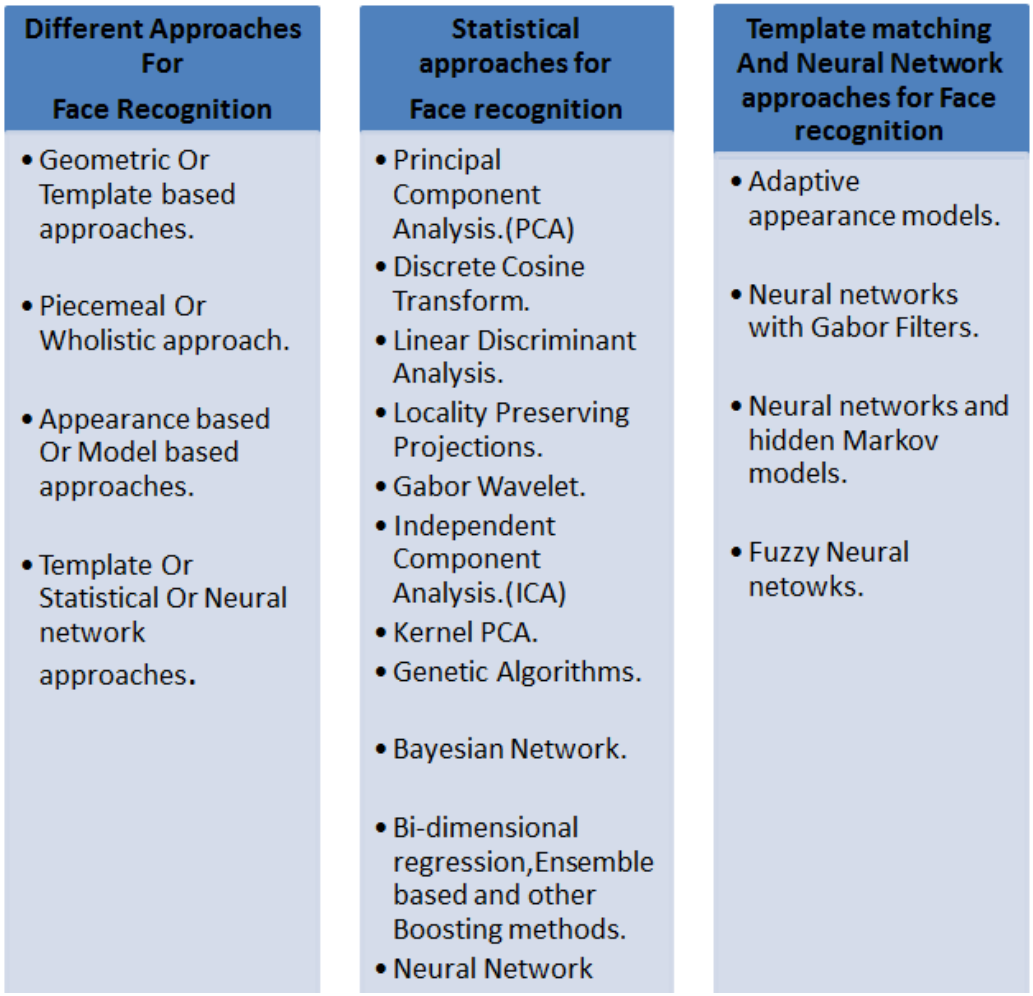


FIGURE 2.1: Face Recognition approaches

2.2 FFT:

The fasr fourier transform (FFT) is simply a fast (computationally efficient) way to calculate the Discrete Fouries Transform.FFT algortithm was first published by Cooley and Tukey in 1965. This is a clever algorithm which can be used to transform a signal from time domain to fequency domain.The FFT greatly reduces the amount of calculation. It also reduces the noise of a signal that are present in time domain.

Functionally, the FFT decomposes the set of data to be transformed into a series of smaller data sets to be transformed. Then, it decomposes those smaller sets into even smaller sets. At each stage of preocessing, the results of previous stages are combined in special way. Finally, it calculates the DFT of each small data set. For example, an FFT of size 32 is broken into 2

FFTs of size16, which are broken into broken 4 FFTs of size 8,which are broken into 8 FFTs of size 4, which are broken into 16 FFTs of size 2. [10]

The number of complex multiplication and addition operations required by the simple forms both the Discrete Fourier Transform (DFT) and Inverse Fourier Transform(IDFT) is of order N^2 as there are N data points to calculate, each of which requires N complex arithmatic operations.

For length n input vector x, the DFT is a length n vector X, with n elements:

$$f_j = \sum_{k=0}^{n-1} x_k e^{-(2\pi i/n)jk} \qquad j = 0, \ldots, n-1.$$

On the other hand, DFT has algorithm complexity and hence is not a very efficient method. It will not be very useful for the majority of practical DSP applications. However, there are number of different Fast Fourier Transform (FFT) algorithms that enable the calculation of a signal much faster than DFT**.**

2.3 How does FFT work:

As discussed earlier, the FFT operated by decomposing an N point time domain signal into each composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum [11].

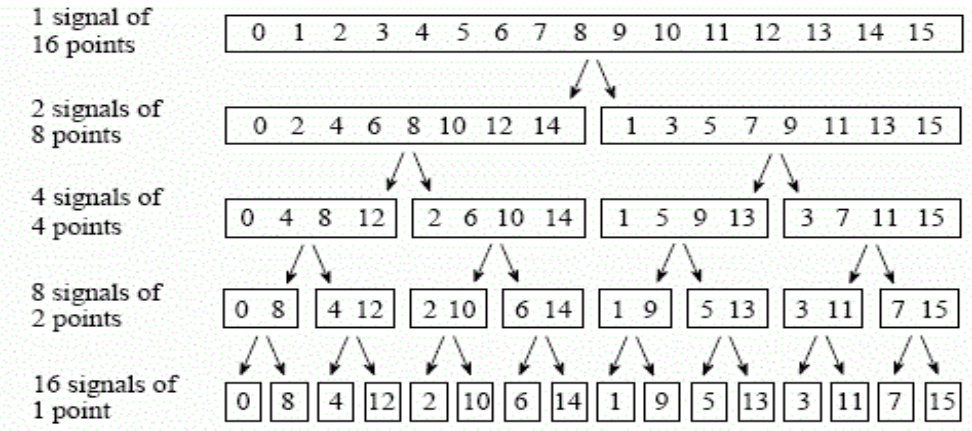


Figure 2.2: Time Domain Decomposition.

The above figure shows an example of the time domain decomposition used in the FFT. In this example, a 16 point signal is decomposed through four separate stages. The first stage breaks the 16 point signal into two signals each consisting of 8 points. The second stage decomposes the data into four signals of 4 points. This pattern continues until there are N signals composed of a single point. An interlaced decomposition is used each time a signal is broken in two, that is, the signal is separated into its even and odd numbered samples. After understanding the structure of decomposition we can say that using it, any N point signal can be easily simplified. It is nothing more than a reordering of the samples in the signal.

Sample numbers in normal order			Sample numbers after bit reversal	
Decimal	*Binary*		*Decimal*	*Binary*
0	0000		0	0000
1	0001		8	1000
2	0010		4	0100
3	0011		12	1100
4	0100		2	0010
5	0101		10	1010
6	0110	⇨	6	0100
7	0111		14	1110
8	1000		1	0001
9	1001		9	1001
10	1010		5	0101
11	1011		13	1101
12	1100		3	0011
13	1101		11	1011
14	1110		7	0111
15	1111		15	1111

Figure 2.3: Rearrangement pattern required.

The given figure shows the rearrangement pattern required. On the left, the sample numbers of the original signal are listed along with their binary equivalents. On the right, the rearranged sample numbers are listed, also along with their binary equivalents. The important part is that the binary numbers are the reversals of each other. For example, sample 3 (0011) is exchanged with sample number 12 (1100). Likewise, sample number 14 (1110) is swapped with sample number 7 (0111), and so forth. The FFT time domain decomposition is usually carried out by a bit reversal sorting algorithm. This involves rearranging the order of the N time domain samples by counting in binary with the bits flipped left-for-right.

The next step in the FFT algorithm is to find the frequency spectra of the 1 point time domain signals. The frequency spectra of the 1 point signal is equal to itself, that means nothing is

required to do this step. Now each of the 1 point signals is a frequency spectrum, not a time domain signal.

The last step in the FFT is to combine the N frequency spectra in the exact reverse order that the time domain decomposition took place. The algorithm gets messy here. There is no shortcut for bit reversal. It is must to go back one stage at a time. In the first stage, 16 frequency spectra (1 point each) are synthesized into 8 frequency spectra (2 point each). In the second stage, the 8 frequency spectra (2 point each) are synthesized into 4 frequency spectra (4 point each) and so on. The last stage results the output of the FFT, a 16 point frequency spectrum.

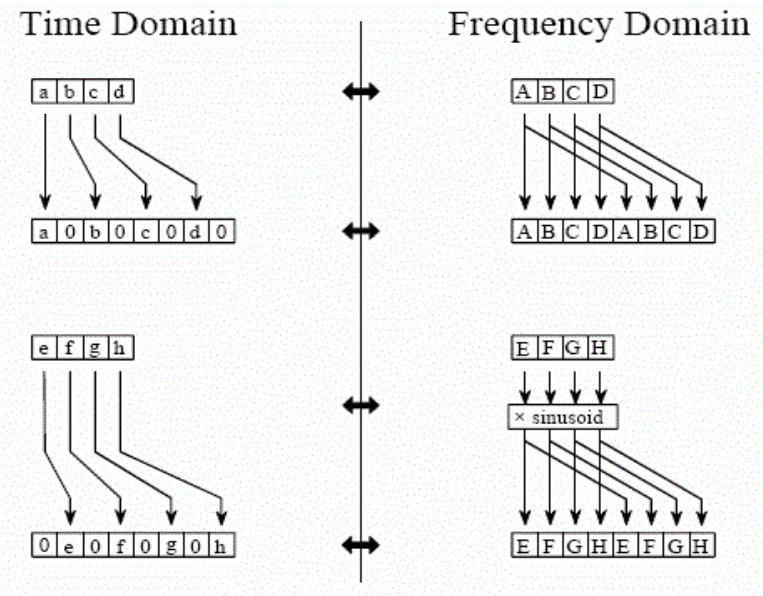


Figure 2.4: Time domain to Frequency domain.

The figure shows how two frequency spectra, each composed of 4 points, are combined into a single frequency spectrum of 8 points. This synthesis must undo the interlaced decomposition done in the time domain. In other work=ds, the frequency domain operation must correspond to the time domain procedure of combining two 4 point signals by interlacing. Considering two time domain signals, abcd and efgh. An 8 point time domain signal can be formed by two steps: dilute each 4 point signal with zeroes to make it an 8 point signal and then add the signals together. That is abcd becomes a0b0c0d0, and efgh becomes e0f0g0h0. Adding these two 8 point signal produces aebfcgdh. Diluting the time domain with zeroes corresponds to the duplication of the frequency spectrum. Therefore, the frequency spectra are combined in the FFT by duplicating them and then, adding the duplicated spectra together.

FFT synthesis flow diagram. This shows the method of combining two 4 point frequency spectra into a single 8 point frequency spectrum. The ×S operation means that the signal is multiplied by a sinusoid with an appropriately selected frequency.

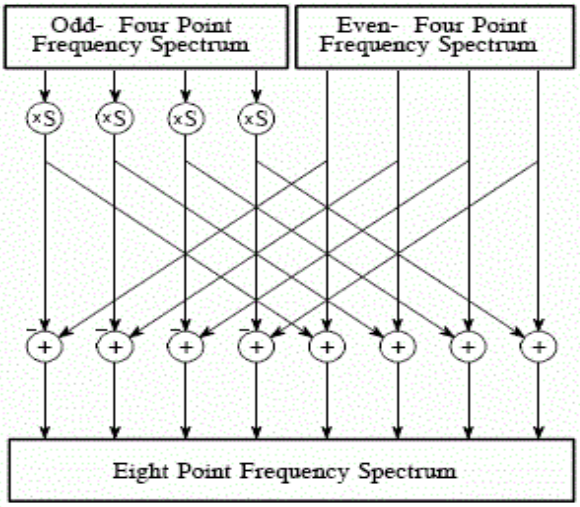


Figure 2.5: FFT Synthesis flow diagram.

In order to match up added, the two time domain signals diluted with zeroes in a slightly different way. In one signal, the odd points are zero while in the other signal, the even points are zero. In other words, one of the time domain signals (0e0f0g0h) is shifted to the right by one sample. This time domain shift corresponds to multiplying the spectrum by a sinusoid. A shift in the time domain is equivalent to convolving the signal with a shifted delta function. This multiplies the signal's spectrum with the spectrum of the shifted delta function. The spectrum of a shifted delta function is a sinusoid. This was the basic of FFT. In case of image it may work differently, which has discussed later [11].

2.4 FFT Algorithms:

As it has been discussed earlier DFT is a complex algorithm and not that efficient. Due to slow processing, it is not applicable in real world problems. To make DFT calculation faster and efficient there are number of FFT algorithms. Such as Radix-2, Butterfly, Cooley-tukey, Prime-factor FFT algorithm, Bruun's FFT algorithm, Radar's FFT algorithm, Bluestein's FFT algorithm etc. In our project we have used Cooley-tukey algorithm of FFT for recognition.

Cooley-tukey algorithm is the most common FFT algorithm. It is named after J.W Cooley and John Tukey. It re-expresses the Discrete Fourier Transform (DFT) of an arbitrary composite size N = N_1N_2 in terms of smaller DFTs of sizes N_1 and N_2, recursively, in order to reduce the computation time to O(NlogN) for highly-composite N.

The Cooley-Tukey algorithm can be combined arbitrarily with any other algorithm, as it breaks the DFT into smaller DFTs [12].

2.4.1 FFT implementation in NIOS 2 using Cooley-tukey Algorithom:

We have implemented FFT in NIOS2 using Cooley Tukey Algorithm. To achieve this we first created a processor using Qsys. We added various components such as CPU, SDRAM, PLL, Tri state bridge, Onchip memory etc. We made connection by connecting master to slave, source to sink, assigned base address and connected clock through PLL. After adding all the components, it automatically generates a blank code which we will use in our Verilog project.

After that we have written our Verilog code to interface in our FPGA through pin assignment. Then we included our SOPC code in Verilog code and interface with our board's pin which generates the .SOF file. Finally we have completed our hardware configuration.

Next, we have written our C code for FFT in Eclipse.

Finally, we wrote code for Cooley–Tukey Algorithm in C and implemented on NIOS 2 processor. We compiled the code and saw the result in the console pane.

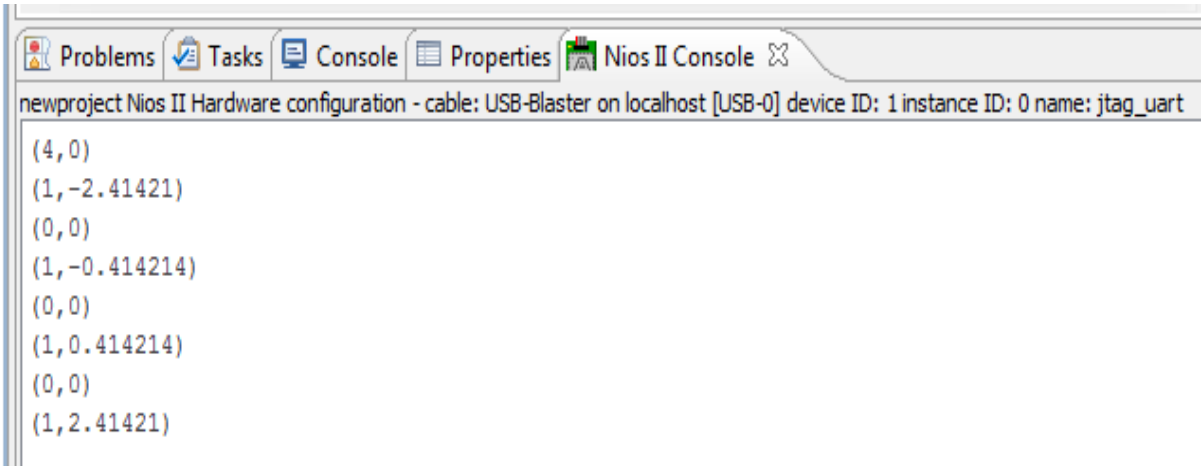


Figure 2.6: Result of Cooley Tukey.

If we compare the result with the Matlab's result, it matches. Therefore, it can be said that our approach is correct.

2.5 Applying FFT on an image:

Fast Fourier Transform on image is a representation of the image in frequency domain. Its function on image is to decompose it into its real and imaginary components. If we take an image as an input then the number of frequencies in the frequency domain is equal to the number of pixels in the original image [13].

The inverse FFT re-transforms the image from frequency domain to spatial domain or time domain. The FFT and its inverse of a 2D image are given by the following equations:

$$F(x) = \sum_{n=0}^{N-1} f(n) e^{-j2\pi(x\frac{n}{N})}$$

$$f(n) = \frac{1}{N} \sum_{n=0}^{N-1} F(x) e^{j2\pi(x\frac{n}{N})}$$

Here f (m, n) is the pixel at f (m, n) coordinates, F(x,y) is the value of the image in the frequency domain at (x,y) coordinates. M and N are the dimensions of the image. Since image is two dimensional, we applied 2D FFT on it. The 2D transform can be done as two 1D transforms as shown below (shown only the horizontal direction) –one in the horizontal direction followed by the other in the vertical direction on the result of the horizontal transform. The end result is equivalent to perform the 2D transform in the frequency space.

$$F(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) e^{-j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

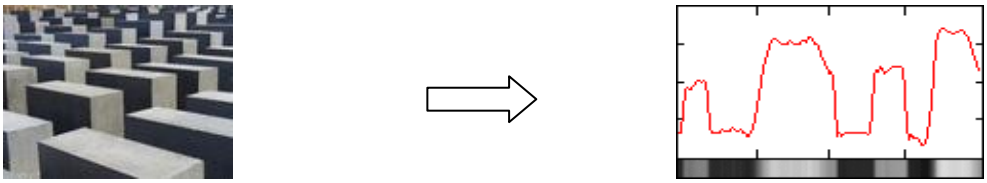
$$f(m,n) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F(x,y) e^{j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

The FFT that's implemented in the application here requires that the dimensions of the image are power of two. An interesting property of FFT is that the transform of N points can be written as the sum of two N/2 transforms. This is important because some of the computations can be reused thus eliminating expensive operations [13].

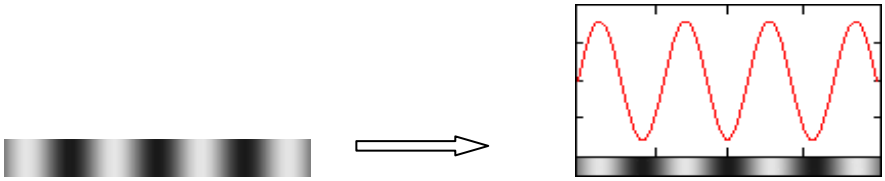
The output of the Fourier Transform is a complex number and has a much greater range than the image in the spatial domain. Therefore, to accurately store these values, they are stored as floats. Furthermore, the dynamic range of the Fourier coefficient is too large to be displayed on the screen and these values are jscaled to bring them within the range of values that can be displayed [13].

A modern interpretation of FFT states that, “any well-behaved function can be represented by a superposition (combination or sum) of sinusoidal waves. It can be said that, the frequency domain representation is just another way to store and reproduce the spatial domain image [14].

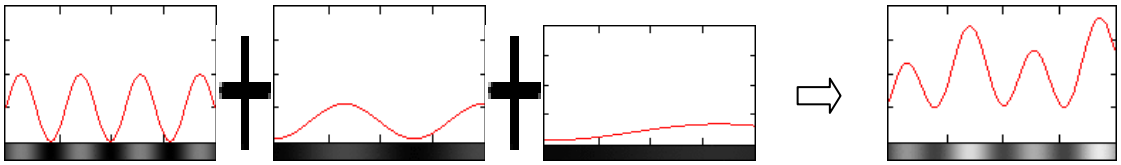
If we take a single row or column of pixel from any image and graph it, we will find that it looks more like a wave.



If the fluctuations are more regular in spacing and amplitude, we would get something more like a wave pattern. Such as,



If we were to add more waves together, we might get a pattern that is closer to the original image.



The superposition of waves or addition ojf waves in much closer, but still does not match the image pattern. However we can continue in this manner, adding more waves and adjusting them until the resulting composite wave gets closer and closer to the actual profile of the original image. Eventually by adding enough waves we can exactly reproduce the original image. Therefore, it can be said that images are nothing but the summation of sine and cosine waves.

In other words, by adding together a sufficient number of sine waves of the right frequency and amplitude, any fluctuating pattern can be reproduced. Fourier Transform generally works out to find out the waves that comprise an image [14].

The Fast Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components or waves. Undoubtedly, the output of FFT represents the image in the frequency domain, while the input image is the spatial domain or time domain equivalent. In the Fourier domain image, each point represents a particular frequency contained the spatial domain image.

If we want to access the geometric characteristic of a spatial domain image, then FFT can be used. Because the image in the Fourier domain is decomposed into its sinusoidal components, which is the easy way to examine or process certain frequencies of the image, that influences the geometric structure in the spatial domain [14].

In most implementations the Fourier image is shifted in such a way that the DC-value or the image mean is displayed in the center of the image. The further away from center of an image point is, the higher is its corresponding frequency [15] [16].

In general if we apply FFT on an image, we get the complex result. The magnitude calculated from the complex result is shown in [15] [16].

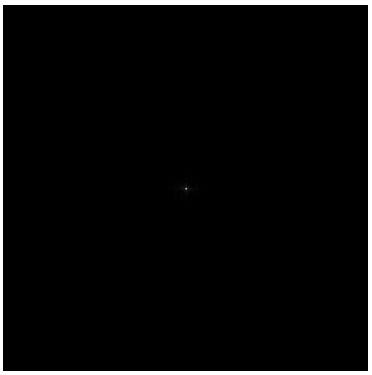


Figure 2.7: The magnitude calculated from the complex result

It is seen that the DC value is by far the largest component of the image. However, the intensity values in the Fourier image or the dynamic ranges of the Fourier coefficients is too large to be displayed on the screen, therefore all other values appear as black. If we apply logarithmic transformation to the image we obtain

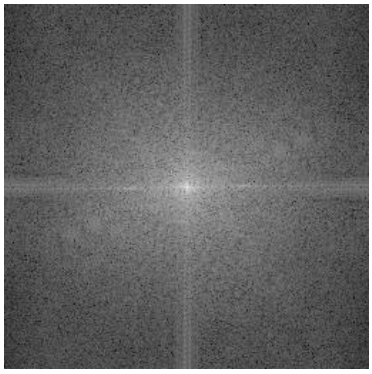


Figure 2.8: Magnitude after logarithmic transform.

We can see that the image contains component of all frequencies, but their magnitude gets smaller for higher frequencies. Hence, low frequencies contain more image information than the higher ones. The transformed image tells us that there are two dominating directions in the Fourier image, one passing vertically and one horizontally through the center. These originate from the regular patterns in the background of the original image.

The phase of the FFT of the same image can be shown as

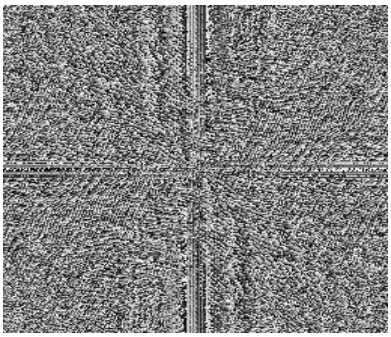


Figure 2.9: The phase of FFT

The value of each point determines the phase of the corresponding frequency. As in the magnitude image, we can identify the vertical and horizontal lines corresponding to patterns in the original image. The phase image does not contain much new information about the structures of the spatial domain image. Therefore, we will confine ourselves to displaying only the magnitude of the Fourier Transform unless our interest does not belong to reconstruct the image [15] [16].

On the other hand, if we do not separate the magnitude and phase part of an image, after applying FFT on that image we will obtain

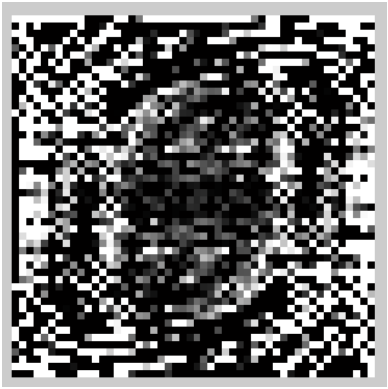


Figure 2.10: Magnitude and Phase of a Fourier image.

The above diagram contains both the magnitude and phase value of a Fourier image. In our project we have considered the both parts.

2.6 Applications:

Face recognition systems have achieved a huge popularity due to wide range of applications. It has been an area of research from very beginning. Applications exist in two main categories: practical application and research application.

From practical standpoint face recognition is extensively used in security systems. The FBI is already using it to identify suspects who are caught on surveillance cameras. The places like airports, International borders the need is raising for a face recognition system that identifies individuals. Face recognition systems can be used in entertainment purpose like video games.

In research applications, face recognition has paved the way for research in areas like image and video processing. Due to the increasing demand of this system into many sectors, researchers are working on developing many algorithms of face recognition. Principle Component Analysis (PCA and KPCA), Linear Discriminant Analysis (LDA), Independent component analysis (ICA), genetic algorithms, neural networks, FFT these are the algorithms established so far by the researchers. In our project we will be focusing on implementing FFT and PCA on the FPGA board. Then we will compare the results at the end. The FFT is used in a wide range of applications, such as image analysis, image filtering, image reconstruction, image compression and we used it for image recognition as well.

2.7 Benefits of using FPGA-

As it has mentioned earlier that one of the important objectives of our project is to get acquainted with FPGA board, since it is a complete new area for us. While researching we have known very interesting things about FPGA and had decided to choose our project based on this board.

An FPGA is exactly what the name suggests: a Field Programmable Gate Array. We program it as a piece of hardware. The FPGA basically implements look up tables. It is good at doing complex logic very fast. Using hardware programming languages such as VHDL and Verilog someone can create complex logic structures. Speed is the biggest advantage of FPGA. It is reprogrammable. More than one project can be implemented using same FPGA board.

FPGAs exceed the computing power of digital signal processors by taking the advantage of hardware parallelism. It accomplishes more per clock cycle. It has specialized functionality to closely match application requirements. It supports long term maintenance. As a product, functional enhancement can be made, without spending time on redesigning hardware or modifying board layout. However, FPGAs are much expensive than microcontrollers. If our design needs greater integration density then FPGAs are appropriate. For smaller projects we go for microcontrollers [17].

Chapter 3:

MATLAB Implementation

3.1 Basic Approach

Before implementing the process in hardware, we verified our project in Matlab first. In our project first we have used FFT as a basic algorithm.

- We have made our own database consisting of Brac University students and used them to develop our recognition system. The database that contains the images of different expressions of the students is named "Train Database". It has total 20 images. Here we have considered two different expression of an image.
- We have used another database which is named the "Test Database". The database that contains the image that will be compared with the train database's image is named the "Test Database". It may contain image inside or outside image of Train Database.
- Firstly we placed the Train Database containing 20 images in a directory using Matlab.
- Then we resized the image into 50:50 to ensure same dimension for every image.
- For the ease of further processing we converted the RGB data into Gray scale, which reduces the matrix dimension.
- After that we have applied FFT on the entire database using the Matlab function FFT2, as images are two dimensional.
- Next we have computed the mean value of the FFT images using mean2 function. We repeated the above steps on test database also.
- Then we made an array containing the differences of means between test and train databases. This is to mention that Test database can contain a single image.
- We have set a threshold by trial and error method which is 21.
- If the difference of mean is less than 21, it has been declared that the image is matched. For the values that are more than 21, we considered the images are not matched. Images that are not included in the Train Database will not match in the end.

The whole procedure can be shown in a following flow chart.

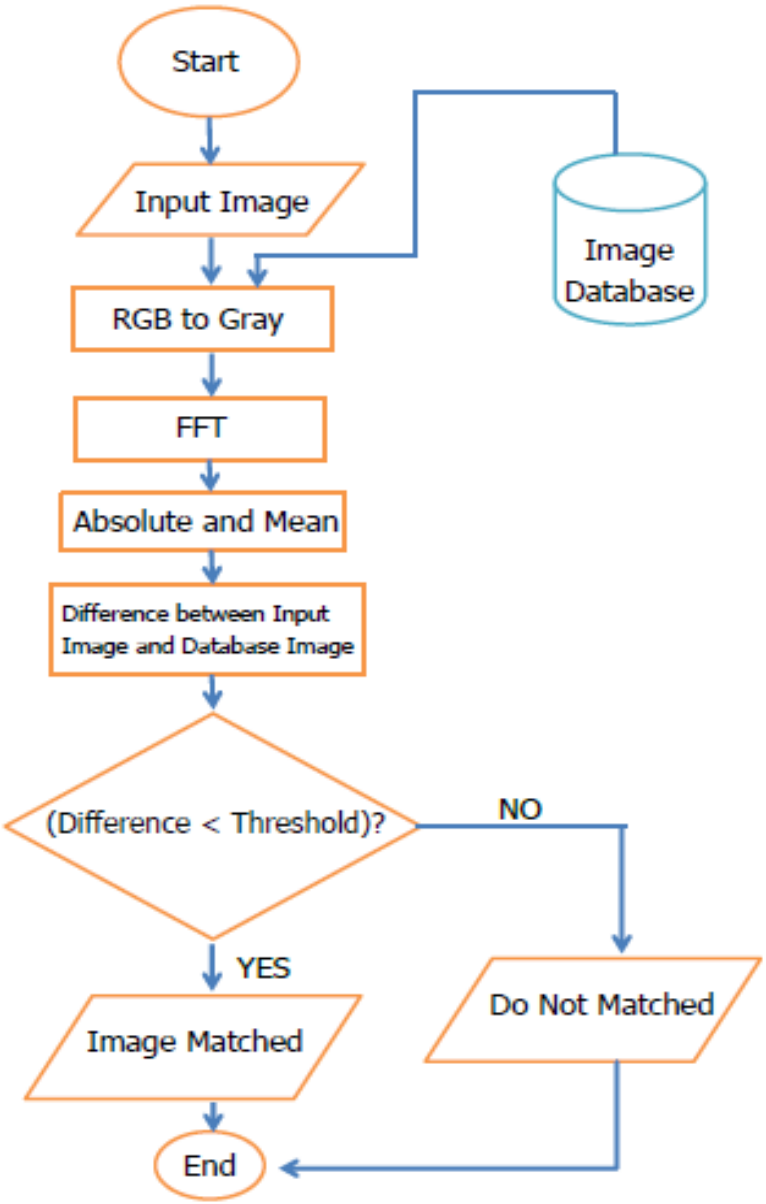


Figure 3.1: Flowchart of FFT based face recognition.

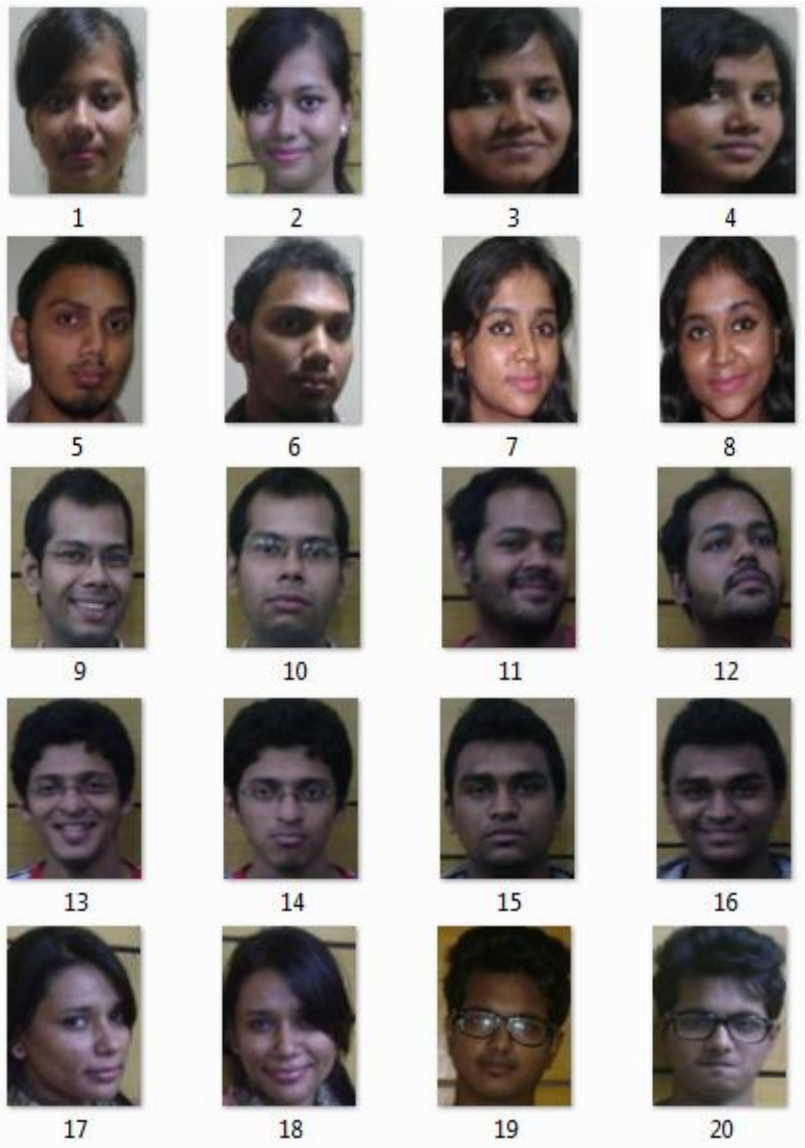


Figure 3.2: Train Database

In Figure3.2, our Train Database is shown. It is noticeable in the database, that the images we have taken are not uniform. One image has the light effect, another may not. One is smiling, another is not. It has been done intentionally, so that we can identify the limitations of FFT spontaneously. To check, FFT's performance whether it can match considering the light effect and different expressions.

Another Database we have the Test Database it may or may not be from the Train Database. It can include only one image. For a while let's consider the Test Database has the following image.

Figure 3.3: Test Database Image

Since another two expressions of the above image is present in the Train Database, if we run the system the answer will be "Matched".

3.2 Two dimensional FFT on an image:

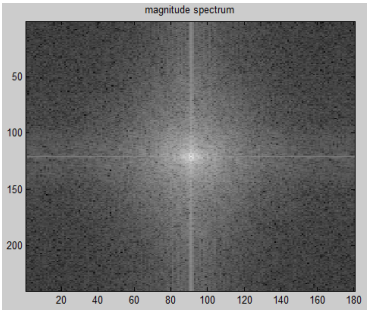
In image processing, the 2D FFT allows one to see the frequency spectrum of the data in both dimensions and lets one visualize filtering operations more easily. The 2D FFT is simply a Fourier transform of one dimension of the data, followed by a Fourier transform over the second dimension of the data. In the following example we have performed a 2D FFT on an image, switched the magnitude and phase content. Now we would get to see what actually happened in Matlab when we applied 2D FFT of an image from our own database.
Considering the code written below [19].

```
close all;

clear all;

img   = imread('Farhan.jpg','jpg');
imagesc(img)
img   = fftshift(img(:,:,2));
F     = fft2(img);

figure;
```

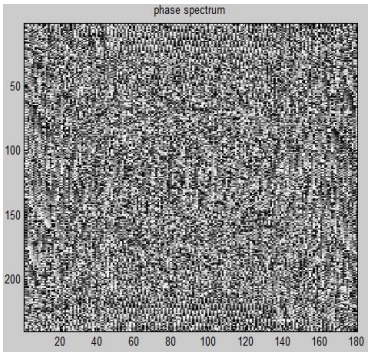
```
imagesc(100*log(1+abs(fftshift(F)))); colormap(gray);
title('magnitude spectrum');

figure;
imagesc(angle(F));  colormap(gray);
title('phase spectrum');
```

Before entering into our main recognition code, using the above code we applied FFT2 on an image to observe the output and it results

The Original Image



Magnitude Spectrum



Phase Spectrum

The 2D FFTs are accomplished using fft2. The image files are imported as unit8, so they should be converted to double arrays before doing the FFTs. The FFT of real, non-even data is complex, so the magnitude and phase of the 2D FFTs should be displayed. The function fftshift is used to shift the quadrant of the FFT around to see the lowest frequencies in the center of the plot [18].

If we look at the FFT of above image, it can be seen that most of the energy in the Fourier domain is present in the center on the image, which corresponds to low frequency data in the image domain. This corresponds to many gradual changes in the image. The phase of the FFT is hard to interpret and generally looks like noise. However, the phase holds a great deal of the information needed to reconstruct the image. To demonstrate the role of the phase of the FFT, we switched the magnitude and phase of the image. If we want to reconstruct the image it is necessary to show the magnitude and phase part separately. However, our project is not concerned with reconstructing the image using inverse 2D FFT; therefore we have considered the magnitude and phase part together in a single frame [18].

3.3 Functions used in Matlab:

In this section we will discuss the functions that have been used in Matlab for recognition and the results.

```
%Import images
sdirectory = 'Train Database';
tifffiles = dir([sdirectory '/*.jpg']);
I = cell(1,numel( tifffiles));
for k = 1:length(tifffiles)
filename = [sdirectory '/' tifffiles(k).name];
I{k} = imread(filename);

%Resize images
Rb=imresize(I{k},[50 50]);
%RGB to Gray images
 J=rgb2gray(Rb);
%2D FFT
fftb=fft2(J);
%Display images
figure,imshow(I{k})
imshow(Rb)
```

```
figure, imshow(J)

figure, imshow(uint8(fftb))
```

"Imread" is used to import the images into Matlab. This function can handle most of the standara image file formats, such as bmp, jpg, tiff and png [18]. In our code "Imshow" is usd to display the images. "Imshow" is one of several functions that plot images, but this function automatically eliminates the axes, displaying image nicely.

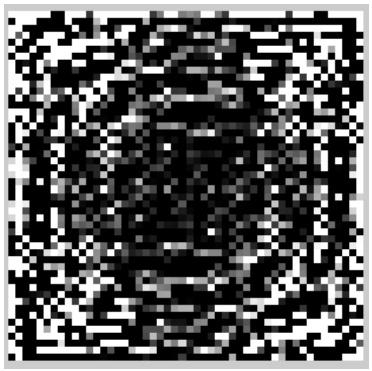
This function works well for original images. When we applied "Imshow" in our original image it shows

Original Image

After applying RGBtoGray function in the original image we obtained

Gray scale Image

After turning the original image into gray scale, we performed 2D FFT on the image considering both magnitude and the phase, it results



FFT Image

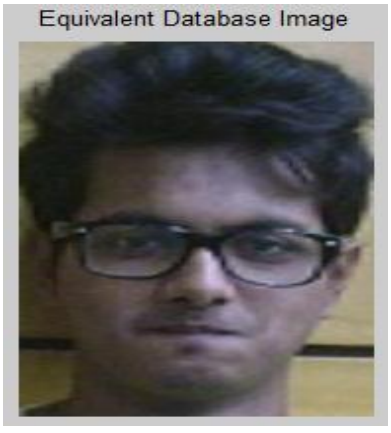
FFT based face recognition is able to recognize faces with slight change in expression. In Test Database we put an image of different expression of one the images of Train Database. After simulation they matched. Though it is not effective as PCA algorithm, yet to some extend it works perfectly and we get

85.096% Matched

Equivalent Database Image

We experimented taking another image of different expression along with glasses to verify whether FFT can recognize it and it showed

78.6761% Matched



Equivalent Database Image

Therefore, it can be said undoubtedly that FFT can recognize faces of different expressions successfully.

On the other hand, if we place the same image to Test Database that is already stored to the Train Database, the accuracy is 100%. For example

100% Matched

Equivalent Database Image

This means if we test exactly the same image, this algorithm can identify the same image from the train database.

Therefore, we used the above steps to verify if our proposed algorithm is suitable, and also to what extent, as a face recognition algorithm.

Chapter 4

4.1 DE0 Board and TRDB_D5M Specifications:

The field-programmable gate array (FPGA) is a semiconductor device that can be programmed after manufacturing. We can use a FPGA to implement any logical function that an application-specific integrated circuit (ASIC) could perform. Unlike previous generation FPGAs using I/Os with programmable logic and interconnects, today's FPGAs consist of various mixes of configurable embedded SRAM, high-speed transceivers, high-speed I/Os, logic blocks, and routing. Most importantly, an FPGA contains programmable logic components called logic elements (LEs) and a hierarchy of reconfigurable interconnects that allow the LEs to be physically connected. We can configure LEs to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flipflops or more complete blocks of memory. In addition, newer FPGA families are being developed with hard embedded processors, transforming the devices into systems on a chip (SoC) [20].

Advantages of using FPGAs over ASICs and ASSPs are including:

- Rapid prototyping
- Shorter time to market
- The ability to re-program in the field for debugging
- Lower NRE costs
- Long product life cycle to mitigate obsolescence risk

4.2 Cyclone III FPGA: Architecture

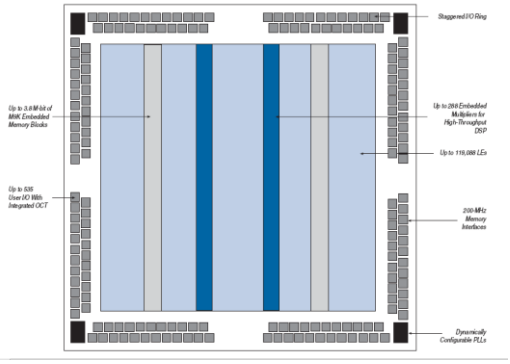


Figure 4.1 : Cyclone III Device Architecture Overview

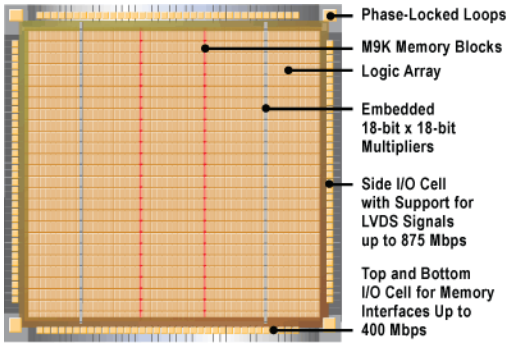


Figure 4.2 : Cyclone III FPGA Floorplan

Cyclone® III FPGAs has low power, high functionality, and low cost. The 65nm architecture consists of up to 120K vertically arranged logic elements (LEs), 4 Mbits of embedded memory arranged as 9-Kbit (M9K) blocks, and 200 18x18 embedded multipliers. Cyclone III LS FPGAs have a memory-rich and multiplier-rich floor plan consisting of up to 200K logic elements, 8.2 Mbits of embedded memory, and 396 embedded multipliers [20].

Both architectures include highly efficient interconnect and low-skew clock networks, providing connectivity between logic structures for clock and data signals. The logic and routing core fabric is surrounded by I/O elements (IOEs) and phase-locked loops (PLLs), as shown in Figure 4.2.

4.3 Logic Elements:

The logic array consists of LABs, with 16 LEs, LAB control signals, LE carry chains, Resister chains and local interconnect in each LAB. LABs are grouped into rows and columns across the device. Cyclone III devices range from 5,136 to 119,088 LEs. A LE, is compact and provides advanced features with efficient logic utilization. Each LE has four-input look-up table (LUT), a programmable register, a carry chain connection, a register chain connection and support for resister packing and resister feedback. Moreover, it has the ability to drive all types of interconnect: local, row, column, resister chain and direct link interconnect [20].

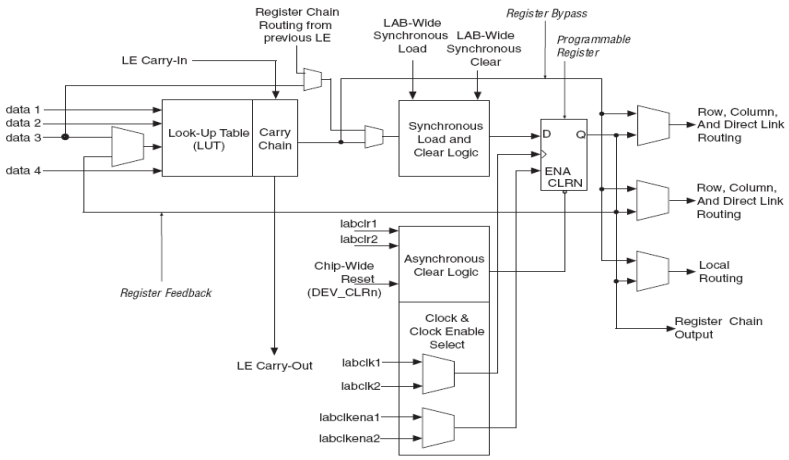


Figure 4.3: Cyclone III Logic Elements

4.4 Cyclone III FPGA: Applications

The Cyclone III FPGAs are the first to implement a complete suite of security features at the silicon, software, and IP level on a low-power, high-functionality FPGA platform [20].

Cyclone III FPGAs has the following application areas:

- Automotive
- Consumer
- Displays of all sizes
- Industrial
- Military
- Video and image processing
- Wireless communications

4.5 Altera Cyclone III 3C16 FPGA device:

The DE0 board has many features that allow the user to implement a wide range of designed circuits, from simple circuits to various multimedia projects. DE0 has Altera Cyclone® III 3C16 FPGA device , Altera Serial Configuration device – EPCS4, USB Blaster, 8-Mbyte SDRAM, 4-Mbyte Flash memory, SD Card socket , 3 pushbutton switches, 10 toggle switches, 10 green user LEDs,50-MHz oscillator for clock sources , VGA DAC with VGA-out connector, RS-232 transceiver, PS/2 mouse/keyboard connector, Two 40-pin Expansion Headers [20].

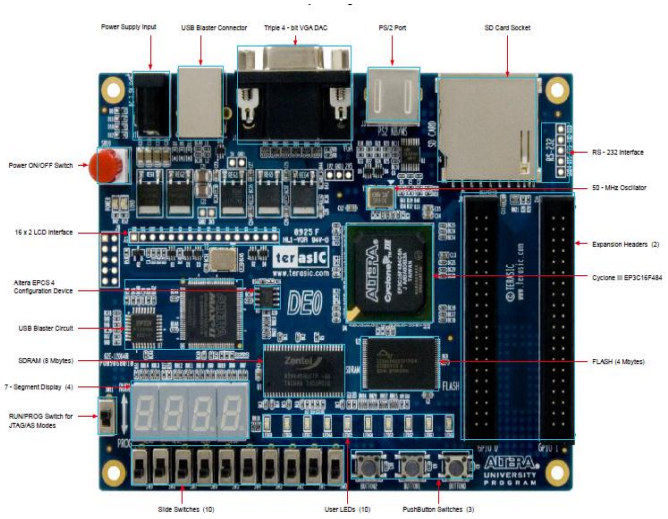


Figure 4.4: DE0 FPGA Specifications.

To provide maximum flexibility for the user, all connections are made through the Cyclone IIII FPGA device. Thus, the user can configure the FPGA to implement any system design.

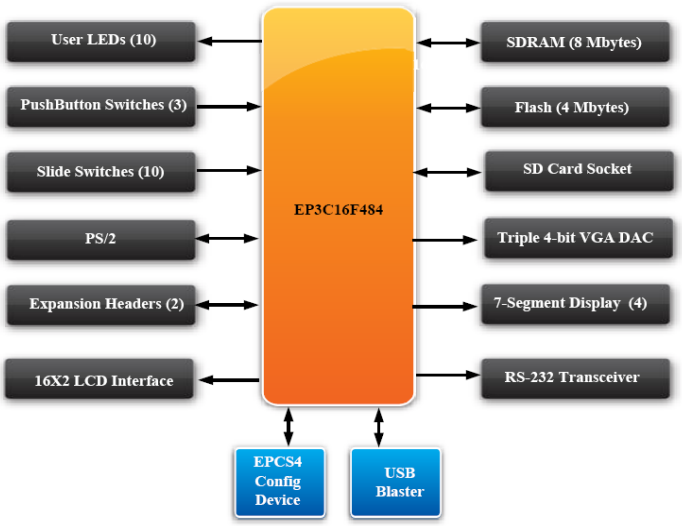


Figure 4.5: DE0 FPGA Components.

DE0 board has 50 MHz Clock input and Cyclone IIII 3C16 which has 15,408 LEs, 56 M9K Embedded Memory Blocks, 504K total RAM bits, 56 embedded multipliers, 4 PLLs, 346 user I/O pins and FineLine BGA 484-pin package. It has Built-in USB Blaster circuit, SDRAM which has one 8-Mbyte Single Data Rate Synchronous Dynamic memory chip and Supports 16-bits data bus. In addition it has 4-Mbyte NOR Flash memory which Support Byte (8-bits)/Word (16-bits) mode and General User Interfaces which includes 10 Green color LEDs (Active high), 4 seven-segment displays (Active low) and 16x2 LCD Interface (Not include LCD module). Moreover, it has SD card socket which Provides both SPI and SD 1-bit mod SD Card access. Furthermore, it has Pushbutton switches, Slide switches, VGA output, Serial ports and two 40-pin expansion headers [20].

4.6 Camera Module Pixel Array Structure:

TRDB-D5M Camera Module is used to capture the image of a person. The address start from (Column 0, Row 0) and it locates at the upper-right corner of the whole region.TRDB-D5M pixel array consists of 2,752 column by 2,004 row. However, whole region is not considered as an active region. Array consists of a 2,592 column by 1,944 row is considered as an active region including boundary region. In addition, boundary region is not used to show pictures

to avoid edge effects. Moreover, the black region which is surrounded by the boundary region is not used to display any pictures.

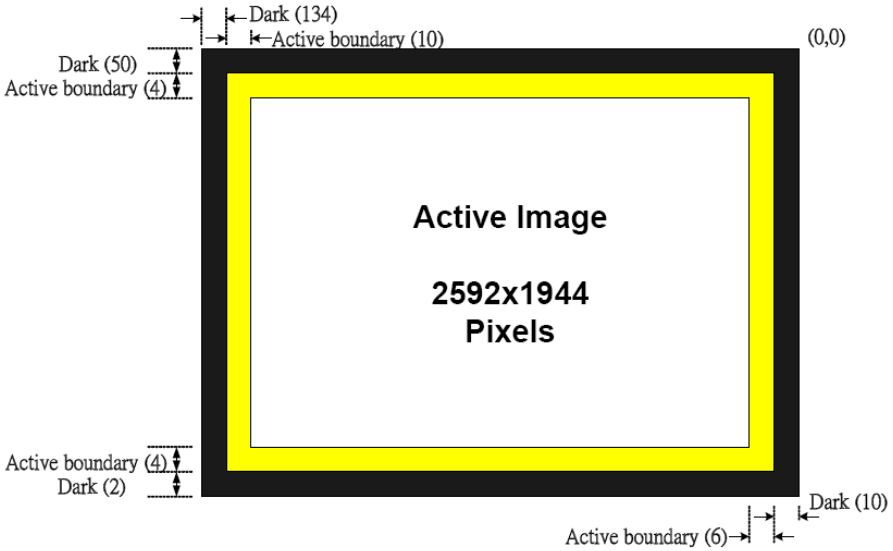


Figure 4.6: Pixel Array Description

Pixels are output in a Bayer pattern format consisting of four "colors"—Green1, Green2, Red, and Blue (G1,G2, R, B)—representing three filter colors. When no mirror modes are enabled, the first row output alternates between G1 and R pixels, and the second row output alternates between B and G2 pixels. The Green1 and Green2 pixels have the same color filter, but they are treated as separate colors by the data path and analog signal chain.

4.7 I2C Protocol:

In early 80's Philips designed I2C bus. This name is taken from Inter IC and mostly called as IIC or I2C [21]. It permits simple communication to achieve data communication between components that resides on same circuit board. It is not as famous as USB or Ethernet but much of electronic devices depend on I2C protocol. It is unique in the use of special combination of signal conditions and changes. It entails only 2 signals or bus lines for serial communications, one is clock and other is data, clock is recognized as SCL or SCK (for serial clock) and data is known as SDA. I2C protocol uses certain registers for common resolutions, their frame rates, LVAL, FVAL, exposure time, green gain, red gain and blue gain.

4.8 Camera Image Acquisition System:

When FPGA gets power to start, system initializes sensor chip and determines mode of operation and certain value of registers in image sensor controls corresponding parameters

[22]. From the following figure it can be seen that LVAL is vertical synchronization signal and FVAL is horizontal reference signal, PIXCLK represents pixel output synchronization signal. When FVAL signal goes high, the system sends out 1280 (number of columns) data at the same time, and the LVAL will appear 960 (number of rows) times high during the FVAL high. One frame image with resolution 1280*960 is collected completely when the next FVAL signal rising edge arrives.

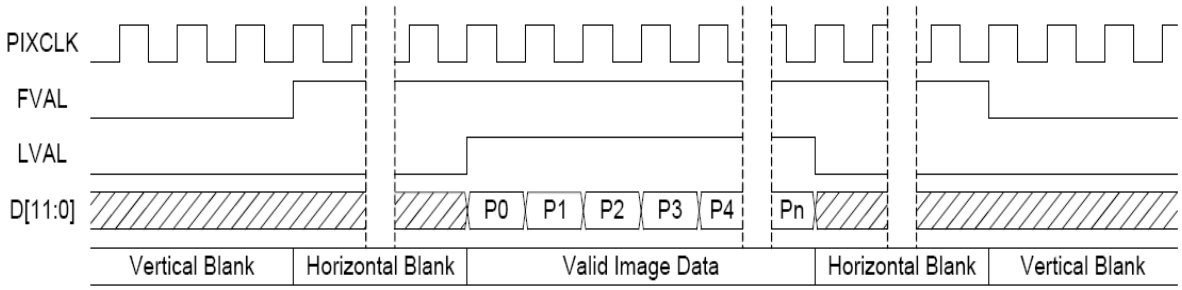


Figure 4.7: Default Pixel Output Timing

4.8.1 Frame Valid

This hardware pin is asserted during the total No. of active rows in the image. This pin is also responsible for the start and end of the pixel stream in the image. This pin goes high only once during each image provided by the camera. In above figure, FVAL goes high when camera provides image.

For a complete configuration, we also need to write the valid values for the various configuration registers in the camera. For example we configure the camera when to start row and columns and what should be the rate of images provided by the camera. Digital and analog gain for the three color components are adjusted to give best performance in specific environment.

4.8.2 Line Valid

This is the hardware pin on the camera which goes high during the valid pixels in a row of the image. This pin asserted number of row times in the image. For our configuration, this pin is asserted 960 times for one image. Each time “line valid” pin goes high, there are 1280 pixels transferred by the camera. Each pixel is transferred by triggering the “pixel clock” pin in the camera.

4.9 Bayer to RGB conversion in FPGA

Image sensor exports the image in Bayer format and in FPGA a Bayer color filter array converts Bayer pattern image into RGB. The pattern of this filter shows that half of its pixels are green while quarter of the total number is assigned for red and same for blue color. Odd pixel lines in the image sensor contain green and blue components, while the even lines contain red and green color components.

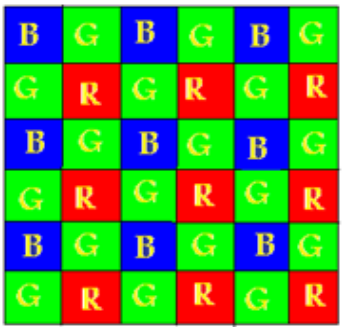


Figure 4.8: Bayer Pattern Filter

Above figure shows a bayer pattern filter and each pixel shows only one component of each primary color. To convert an image from Bayer format to RGB format, each pixel needs to have values of all three primary colors.

4.9.1 RGB conversion

Camera is configured in such a way that a Bayer image is getting 960 rows and 1280 columns with 5 frames per second. Camera outputs the data in Bayer pattern with 12 bit on parallel bus. In Bayer pattern format, each pixel contains one of three primary colors, which consists of four colors: green1, blue, red and green2. The layout is shown in following figure that means two of the remaining color components are missing in each pixel of Bayer pattern.

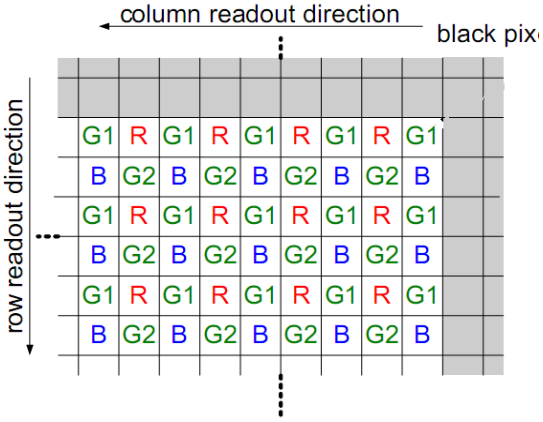


Figure 4.9: Bayer image Pixels

This bayer pattern data is then passed through a module which converts it into RGB values, and utilizes four pixels of Bayer pattern format to construct one pixel of RGB. After applying formula, other two component's value can be find out. Camera manages green pixels as two different colors depending on which line they are coming from. In Bayer format, when 1st complete row and only first 2 pixels of the second row complete scanning, then filter creates the 1st pixel of RGB.

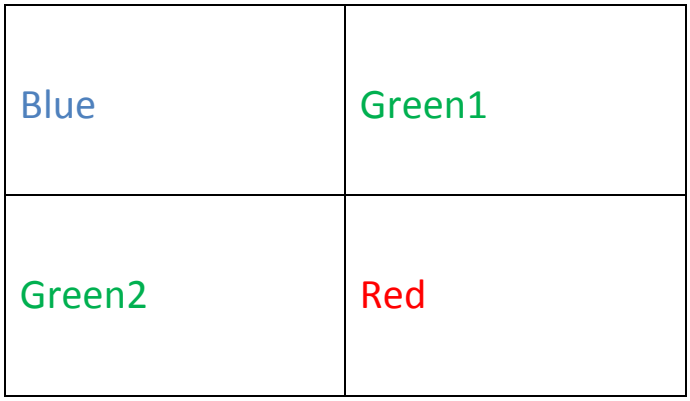


Figure 4.10: RGB pixel from Bayer format

Above figure shows a RGB pixel format. As the second row out of camera completes scanning, first complete row of RGB image is created. Similarly with the completion of 3rd and 4th row of Bayer pattern image a 2nd RGB pixel row completed. As the pixels are being received by the camera, they are simultaneously being transformed into RGB and simultaneously being sent to the memory module in the FPGA. After that we converted this RGB pixel into grayscale using the following formula

Grayscale = (Red+ Green+ Blue)/3

This conversion is used to reduce the matrix dimension. Next this memory module stores this pixel in the external SDRAM through external bus and so on.

Chapter 5
Hardware Implementation:

Altera Corporation is the pioneer of programmable logic solutions. And we have used Altera's FPGA board to use in our project. Our FPGA board is from the Cyclone III device family and its model number is DE0 [27].In our project we have used 'Qsys' extensively. Qsys is the Altera's system integrated tool.' Qsys' system integration tool saves significant amount of time and effort in the FPGA design process by automatically generating interconnect logic to connect intellectual property (IP) functions and subsystems. Qsys is the next-generation SOPC Builder tool that is powered by a new FPGA-optimized network-on-a-chip (NoC) technology delivering higher performance, enhanced design reuse, and faster verification compared to SOPC Builder [27].

A block diagram of our problem formulation for the 'Qsys' part is given below:

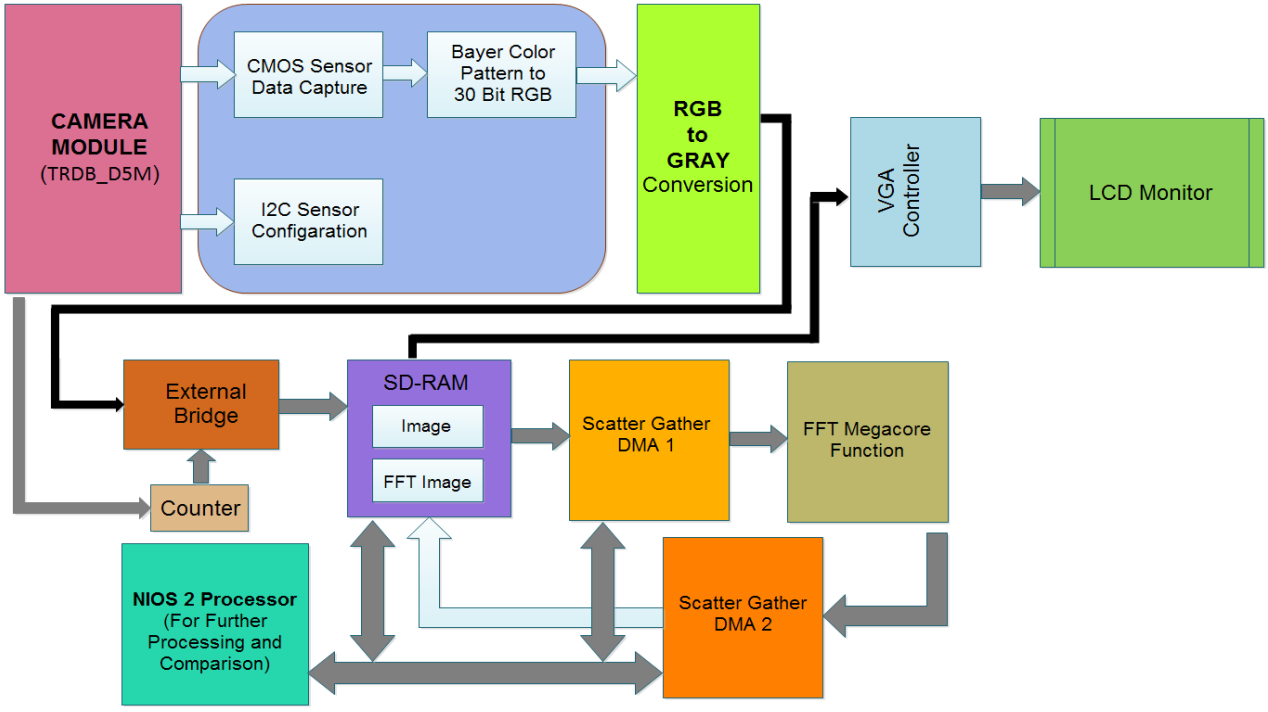


Figure 5.1: Block diagram of our proposed architecture.

The hardware architecture we proposed is as follows:

An image is captured from the FPGA board compatible camera module TRDB_D5M.

The output pixels or the raw data are in Bayer color Pattern.

Therefore the data is passed through Bayer color pattern to 30 bit RGB (RED, GREEN, BLUE) module.

Once the pixels are in RGB, they are then converted into Grayscale (to reduce down the number of planes, in this case from 3 planes to 1plane, to reduce the complexity of data manipulation).

This Grayscale data (or the captured image) is then stored in memory (SDRAM) through the assistance of the external bridge bus and SDRAM controller.

At this stage, to verify if the data is actually stored in the SDRAM we can include the VGA controller and the Video In decoder and display the data on a LCD monitor.

Once the data (or the image) is stored in the SDRAM, the data is accessed from the SDRAM through the Scatter Gather DMA (direct memory access) controller and is passed to the FFT block (Fast Fourier Transform block).

The output, that is, the data after FFT is again stored in SDRAM, this time at a different memory location (in order to keep both the stored data). At this stage another DMA controller is used to transfer these data and access the SDRAM.

Then both sets of data are now available in the SDRAM.

The above steps are followed again, to keep the information of another image (the concept of creating database).

Then Nios II carries out further processing of comparison and recognition for both the images (FFT values are compared).

The data route can be viewed from the RTL viewer to get an idea on the logic gate implementation for different blocks we have used.

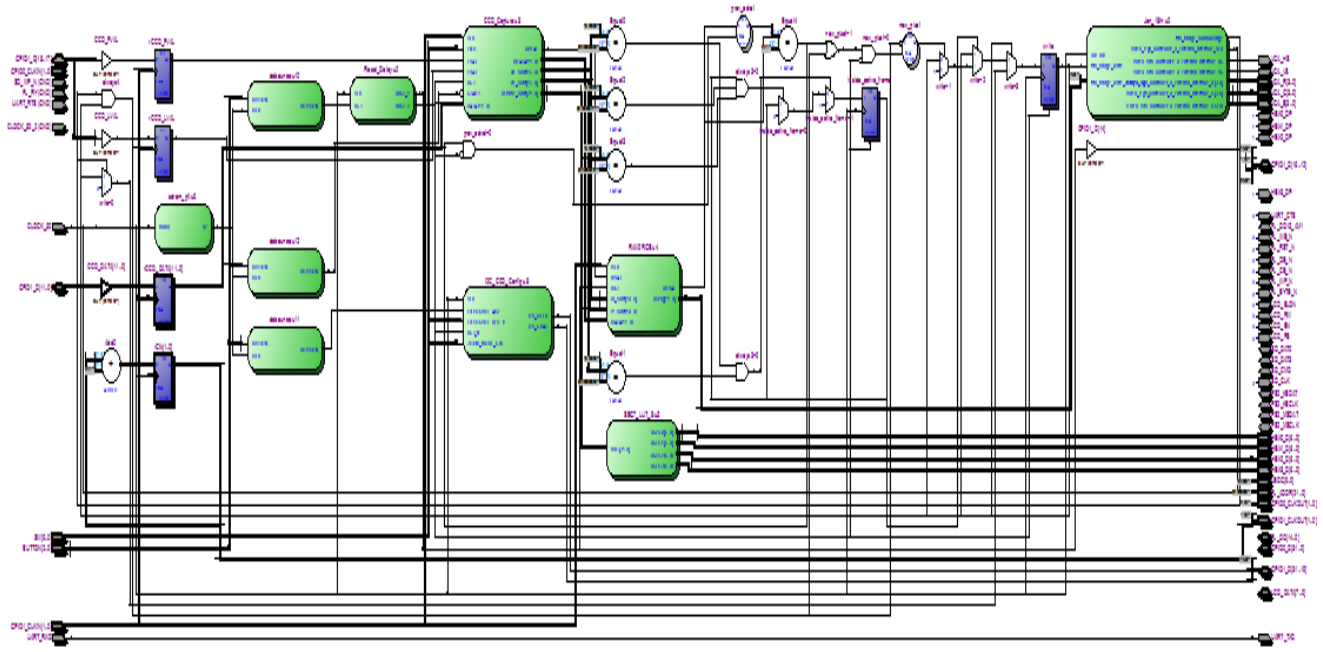


Figure 5.2: RTL Viewer

To accomplish the above processes we have used some 'Qsys' components.

The components that we have considered putting in our system are as follows--- (the order of the components might not be exactly as the following list)

1) External Bus to Avalon Bridge.
2) SDRAM controller.
3) Avalon ALTPLL.
4) SG-DMA (scatter-gather) Controllers.
5) VGA controller.
6) Video DMA Controller.
7) FFT block generated from Mega Wizard.
8) NIOSII Processor.
9) On-chip Memory (RAM).

5.1 External Bus to Avalon Bridge: We have used this IP core or component to make an interface with our external camera module to our system. This bridge provides a

simple interface for a peripheral device (in our case, the TRDB_D5M) to connect with the Avalon

Switch Fabric as a master device. The Bridge creates a bus-like interface to which one or more “master” peripherals can be connected.[34]

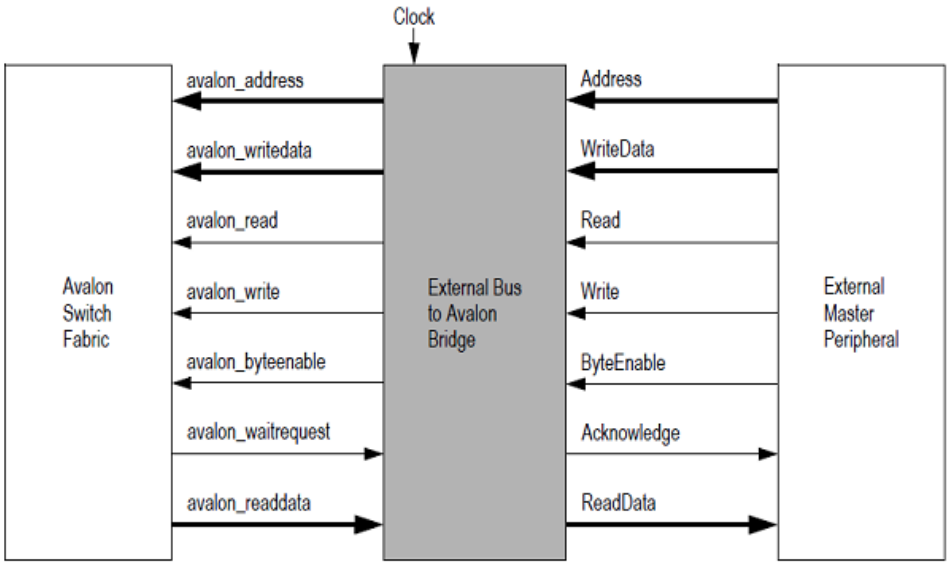


Figure 5.3 : External bus to avalon bridge.

The Bus signals provided are: 1) Address- k bits (up to 32).

2) Read-1 bit.

3) Write- 1 bit.

4) Byte Enable- 16,8,4,2 or 1 bit.

5) Write Data- 128,64,32,16 or 8 bits.

6) Read Data- 128,64,32,16 or 8 bits.

7) Acknowledge- 1 bit.

The bus is synchronous — all bus signals must be read by the master peripheral on the rising edge of the clock. A bus transfer happens when either Write or Read is high. For our project we coded in such a way that the bridge does the work of a write command as we want to write the data from the camera to the SDRAM.

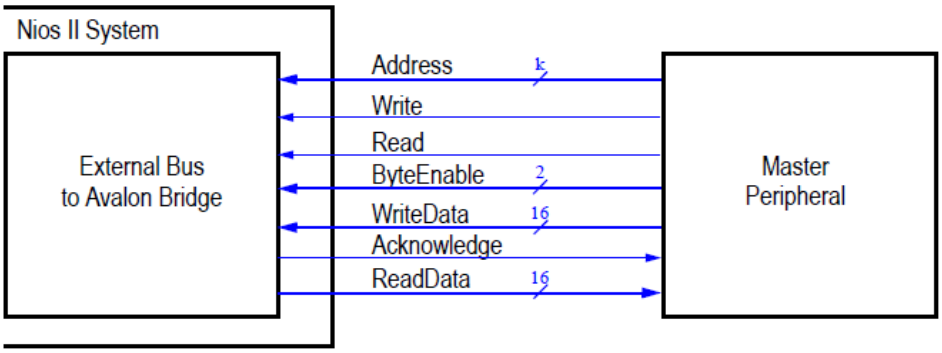


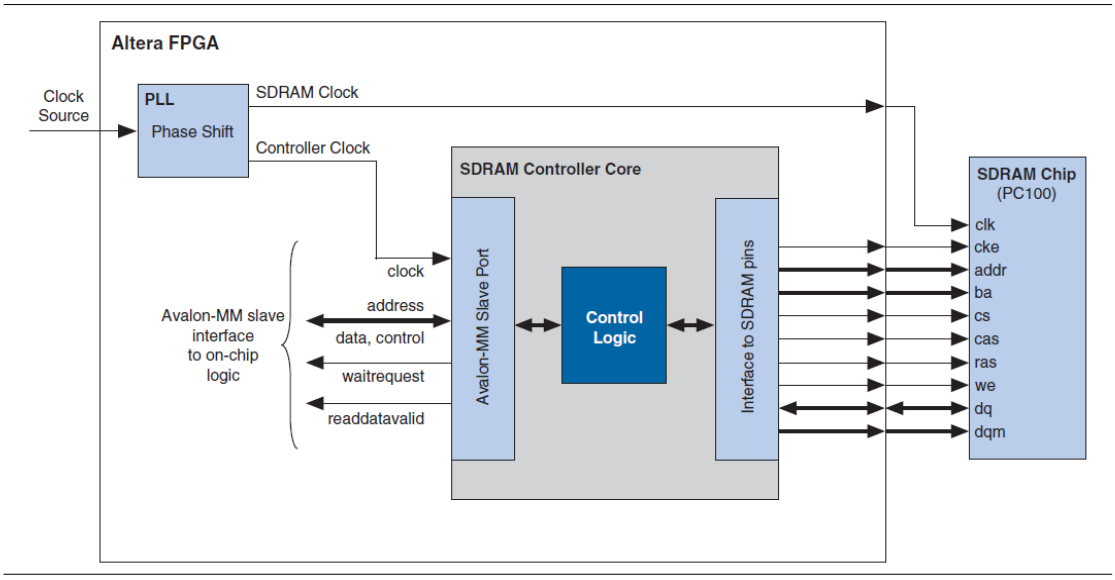
Figure : External Bus to Avalon Bridge with Nios II system

Two parameters are needed to specify in 'Qsys' External Bridge to Avalon core:

1) Data Width — the number of data bits involved in a transfer. The Bridge supports data widths of 8,16,32,64, and128 bits.
2) Address Range — the addressable space supported by the Bridge. It is possible to specify the address range of 1,

2,4,8,16,32,64,128,256,512,and 1024, in either bytes, kilobytes (kB) or megabytes (MB).

5.2 SDRAM controller: it allows designers to create custom systems in an Altera device that connect easily to SDRAM chips. This SDRAM controller connects to one or more SDRAM chips, and handles all SDRAM protocol requirements [23].



SDRAM controller with Avalon Interface block diagram.

Avalon MM interface

The Avalon-MM slave port is the user-visible part of the SDRAM controller core. The slave port presents a flat, contiguous memory space as large as the SDRAM chip(s). The Avalon-MM interface behaves as a simple memory interface. There are no memory-mapped configuration registers.

Signal Timing and Electrical Characteristics

The timing and sequencing of signals depends on the arrangement of the core. The hardware designer configures the core to match the SDRAM chip chosen for the system. The SDRAM controller Mega Wizard has two pages: **Memory Profile** and **Timing**. These can be configured by using the option 'Custom' or we could use any of the several predefined SDRAM configurations provided if the If the SDRAM subsystem on the target board (DE0 in our case) matches one of the preset configurations. Some of the preset configurations are for

- Micron MT8LSDT1664HG module
- Four SDR100 8 MByte × 16 chips
- Single Micron MT48LC2M32B2-7 chip
- Single Micron MT48LC4M32B2-7 chip
- Single NEC D4564163-A80 chip (64 MByte × 16)
- Single Alliance AS4LC1M16S1-10 chip
- Single Alliance AS4LC2M8S0-10 chip

But we have configured it for our convenience which was appropriate for our SDRAM subsystem.

The **Memory Profile** page allows one to indicate the structure of the SDRAM subsystem such as address and data bus widths, the number of chip select signals, and the number of banks.

Settings		Allowed Values	Default Values	Description
Data Width		8, 16, 32, 64	32	SDRAM data bus width. This value determines the width of the dq bus (data) and the dqm bus (byte-enable).
Architecture Settings	Chip Selects	1, 2, 4, 8	1	Number of independent chip selects in the SDRAM subsystem. By using multiple chip selects, the SDRAM controller can combine multiple SDRAM chips into one memory subsystem.
	Banks	2, 4	4	Number of SDRAM banks. This value determines the width of the ba bus (bank address) that connects to the SDRAM. The correct value is provided in the data sheet for the target SDRAM.
Address Width Settings	Row	11, 12, 13, 14	12	Number of row address bits. This value determines the width of the addr bus. The Row and Column values depend on the geometry of the chosen SDRAM. For example, an SDRAM organized as 4096 (2^{12}) rows by 512 columns has a Row value of 12.
	Column	>= 8, and less than Row value	8	Number of column address bits. For example, the SDRAM organized as 4096 rows by 512 (2^9) columns has a Column value of 9.
Share pins via tri-state bridge dq/dqm/addr I/O pins		On, Off	Off	When set to No, all pins are dedicated to the SDRAM chip. When set to Yes, the addr, dq, and dqm pins can be shared with a tristate bridge in the system. In this case, select the appropriate tristate bridge from the pull-down menu.
Include a functional memory model in the system testbench		On, Off	On	When on, SOPC Builder creates a functional simulation model for the SDRAM chip. This default memory model accelerates the process of creating and verifying systems that use the SDRAM controller. See "Hardware Simulation Considerations" on page 2–7.

Table 5.1: Descriptions of SDRAM parameters.

The **Timing** page allows designers to enter the timing specifications of the SDRAM chip(s) used. The correct values are available in the manufacturer's data sheet for the target SDRAM. For our case it is (IS42S16400).

Settings	Allowed Values	Default Value	Description
CAS latency	1, 2, 3	3	Latency (in clock cycles) from a read command to data out.
Initialization refresh cycles	1–8	2	This value specifies how many refresh cycles the SDRAM controller performs as part of the initialization sequence after reset.
Issue one refresh command every	—	15.625 µs	This value specifies how often the SDRAM controller refreshes the SDRAM. A typical SDRAM requires 4,096 refresh commands every 64 ms, which can be achieved by issuing one refresh command every 64 ms / 4,096 = 15.625 µs.
Delay after power up, before initialization	—	100 µs	The delay from stable clock and power to SDRAM initialization.
Duration of refresh command (t_rfc)	—	70 ns	Auto Refresh period.
Duration of precharge command (t_rp)	—	20 ns	Precharge command period.
ACTIVE to READ or WRITE delay (t_rcd)	—	20 ns	ACTIVE to READ or WRITE delay.
Access time (t_ac)	—	17 ns	Access time from clock edge. This value may depend on CAS latency.
Write recovery time (t_wr, No auto precharge)	—	14 ns	Write recovery if explicit precharge commands are issued. This SDRAM controller always issues explicit precharge commands.

Table 5.1: Descroptions of SDRAM parameters.

There are issues related to synchronizing signals from the SDRAM controller core with the clock that drives the SDRAM chip. During SDRAM transactions, the address, data, and control signals are valid at the SDRAM pins for a small window of time and during this time the SDRAM clock must toggle to capture the correct values. At slower clock frequencies, the clock naturally falls within the valid window but at higher frequencies the SDRAM clock must be compensated to align with the valid window. This is usually done by either calculating or analyzing the SDRAM pins with an oscilloscope.

5.3 PLL: A PLL (Phase Locked Loop) is used to adjust the phase of the SDRAM clock so that edges occur in the middle of the valid window. Tuning the PLL might require trial-and-error effort to align the phase shift to the properties of the target board. But usually Phase shift for 50MHz clock is -3ns and for 100 MHz is -1.5 ns[23][25]. The PLL that we select from 'Qsys' depends on the device family. Tor our three kinds are available. We have chosen ALT PLL for our Cyclone (III) family.

Example Calculation:

Parameter		Symbol	Value (ns) in -7 Speed Grade	
			Min.	Max.
Access time from CLK (pos. edge)	CL = 3	$t_{AC(3)}$	—	5.5
	CL = 2	$t_{AC(2)}$	—	8
	CL = 1	$t_{AC(1)}$	—	17
Address hold time		t_{AH}	1	—
Address setup time		t_{AS}	2	—
CLK high-level width		t_{CH}	2.75	—
CLK low-level width		t_{CL}	2.75	—

Timing Parameters for Micron MT48LC4M32B2 SDRAM Device

Table 5.2: PLL calculations.

Parameter		Symbol	Value (ns) in -7 Speed Grade	
			Min.	Max.
Clock cycle time	CL = 3	$t_{CK(3)}$	7	—
	CL = 2	$t_{CK(2)}$	10	—
	CL = 1	$t_{CK(1)}$	20	—
CKE hold time		t_{CKH}	1	—
CKE setup time		t_{CKS}	2	—
CS#, RAS#, CAS#, WE#, DQM hold time		t_{CMH}	1	—
CS#, RAS#, CAS#, WE#, DQM setup time		t_{CMS}	2	—
Data-in hold time		t_{DH}	1	
Data-in setup time		t_{DS}	2	
Data-out high-impedance time	CL = 3	$t_{HZ(3)}$		5.5
	CL = 2	$t_{HZ(2)}$	—	8
	CL = 1	$t_{HZ(1)}$	—	17
Data-out low-impedance time		t_{LZ}	1	—
Data-out hold time		t_{OH}	2.5	

Table 5.2: PLL calculations.

Parameter	Symbol	Value (ns)
Clock period	t_{CLK}	20
Minimum clock-to-output time	t_{CO_MIN}	2.399
Maximum clock-to-output time	t_{CO_MAX}	2.477
Maximum hold time after clock	t_{H_MAX}	–5.607
Maximum setup time before clock	t_{SU_MAX}	5.936

FPGA I/O Timing Parameters

Table 5.2: PLL calculations.

The SDRAM clock can lag the controller clock by the lesser of *Read Lag* or *Write Lag*:

Read Lag = tOH(SDRAM) – tH_MAX(FPGA)

= 2.5 ns – (–5.607 ns) = 8.107 ns

or

Write Lag= tCLK – tCO_MAX(FPGA) – tDS(SDRAM)

= 20 ns – 2.477 ns – 2 ns = 15.523 ns

The SDRAM clock can lead the controller clock by the lesser of *Read Lead* or *Write Lead*:

Read Lead= tCO_MIN(FPGA) – tDH(SDRAM)

= 2.399 ns – 1.0 ns = 1.399 ns

or

Write Lead= tCLK – tHZ(3)(SDRAM) – tSU_MAX(FPGA)

= 20 ns – 5.5 ns – 5.936 ns = 8.564 ns

Therefore, for this example you can shift the phase of the SDRAM clock from –8.107 ns to 1.399 ns relative to the controller clock. Choosing a phase shift in the

Middle of this window results in the value (–8.107 + 1.399)/2 = –3.35 ns.

These values are collected from Datasheets of the corresponding devices.

To drive the SDRAM we required a PLL (Phase Locked Loop). Cyclone series supports only one type of PLL. A **phase-locked loop** (PLL) is a control system that generates an output signal whose phase will be related to the phase of an input "reference" signal. PLL circuitry is an electronic circuit consisting of a phase detector and a variable frequency oscillator. PLL measures up the phase of the input signal against the phase of the signal derived from its output oscillator and adjust the frequency of its oscillator to keep the phases matched.

The PLL can be used to generate stable frequencies, recover signals from a noisy communication channel, or distribute clock signals throughout the design.

Usually we chose -3ns for 50 MHz and -1.5ns for 100MHz.

5.4 Scatter Gather DMA (Direct Memory Access):

The Scatter-Gather Direct Memory Access (SG-DMA) controller core implements high-speed data transfer between two components [23].

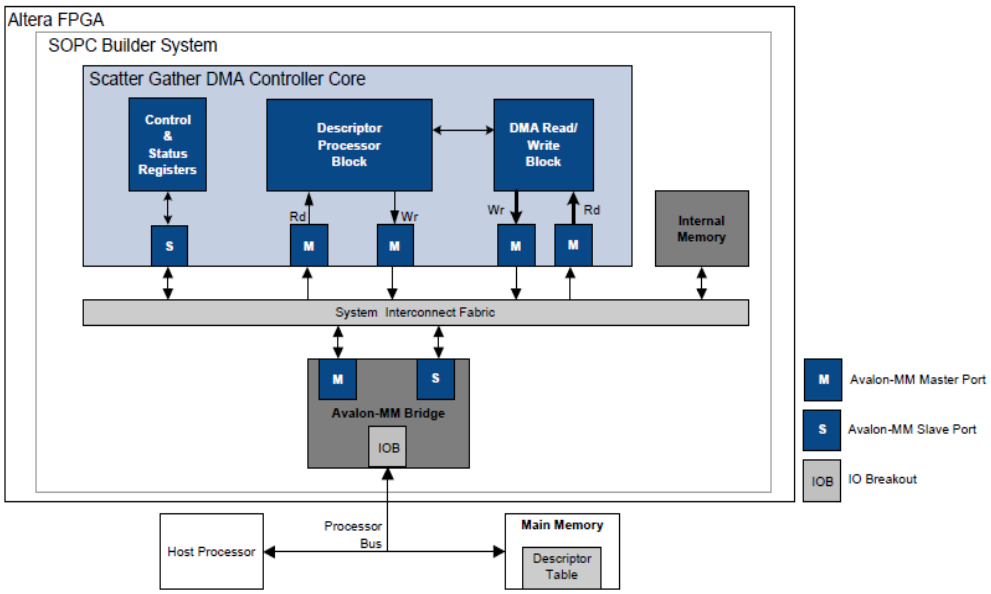
We can use the SG-DMA controller core to transfer data from:

- Data stream to memory.
- Memory to data stream.
- Memory to memory.

For our project to transfer data to the FFT block and to place the output data from the FFT block we have used the first two of the three processes. Firstly, we used memory to data stream, to access the data from SDRAM as the streaming input of the FFT block. Then we used data stream to memory to pass the output of the FFT block to the SDRAM again.

The SG-DMA controller core transfers and merges non-contiguous memory to a continuous address space, and vice versa. The core reads a series of descriptors that specify the data to be transferred. For applications requiring more than one DMA channel, (such as in our case) multiple instantiations of the core can provide the desired throughput. Each SG-DMA controller has its own series of descriptors those specify the data transfers. The SG-DMA controller core is 'Qsys' Builder-ready and integrates easily into any 'Qsys' Builder-generated system. The device drivers are provided in the Hardware Abstraction Layer (HAL) system library, if we want to use NIOS II processor, SG-DMA can be called from the library available in NIOS II.

Since we have used internal memory (SDRAM) here is an example of how SG-DMA controller core transfers data between an internal and external memory.



SG-DMA Controller Core with Internal and External Memory

Programming with SG-DMA Controller

The description of the device, descriptor data structures, and the application programming interface (API) for the SG-DMA controller core are given below.

```
typedef struct alt_sgdma_dev
{
  alt_llist                 llist;              // Device linked-list entry
  const char                *name;              // Name of SGDMA in SOPC System
  void                      *base;              // Base address of SGDMA
  alt_u32                   *descriptor_base;   // reserved
  alt_u32                   next_index;         // reserved
  alt_u32                   num_descriptors;    // reserved
  alt_sgdma_descriptor      *current_descriptor; // reserved
  alt_sgdma_descriptor      *next_descriptor;   // reserved
  alt_avalon_sgdma_callback callback;           // Callback routine pointer
  void                      *callback_context;  // Callback context pointer
  alt_u32                   chain_control;      // Value OR'd into control reg
} alt_sgdma_dev;
```

Fig: Device data structure

```
typedef struct {
    alt_u32   *read_addr;
    alt_u32   read_addr_pad;

    alt_u32   *write_addr;
    alt_u32   write_addr_pad;

    alt_u32   *next;
    alt_u32   next_pad;

    alt_u16   bytes_to_transfer;
    alt_u8    read_burst; /* Reserved field. Set to 0. */
    alt_u8    write_burst;/* Reserved field. Set to 0. */

    alt_u16   actual_bytes_transferred;
    alt_u8    status;
    alt_u8    control;

} alt_avalon_sgdma_packed alt_sgdma_descriptor;
```

Fig: Descriptor data structure

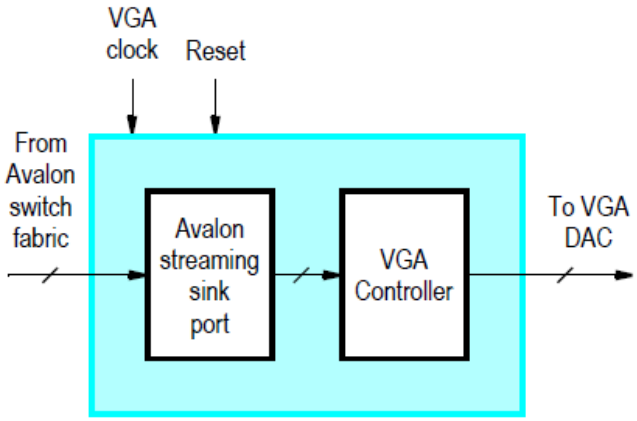
Name	Description
`alt_avalon_sgdma_do_async_transfer()`	Starts a non-blocking transfer of a descriptor chain.
`alt_avalon_sgdma_do_sync_transfer()`	Starts a blocking transfer of a descriptor chain. This function blocks both before transfer if the controller is busy and until the requested transfer has completed.
`alt_avalon_sgdma_construct_mem_to_mem_desc()`	Constructs a single SG-DMA descriptor in the specified memory for an Avalon-MM to Avalon-MM transfer.
`alt_avalon_sgdma_construct_stream_to_mem_desc()`	Constructs a single SG-DMA descriptor in the specified memory for an Avalon-ST to Avalon-MM transfer. The function automatically terminates the descriptor chain with a NULL descriptor.
`alt_avalon_sgdma_construct_mem_to_stream_desc()`	Constructs a single SG-DMA descriptor in the specified memory for an Avalon-MM to Avalon-ST transfer.
`alt_avalon_sgdma_enable_desc_poll()`	Enables descriptor polling mode. To use this feature, you need to make sure that the hardware supports polling.
`alt_avalon_sgdma_disable_desc_poll()`	Disables descriptor polling mode.
`alt_avalon_sgdma_check_descriptor_status()`	Reads the status of a given descriptor.
`alt_avalon_sgdma_register_callback()`	Associates a user-specific callback routine with the SG-DMA interrupt handler.
`alt_avalon_sgdma_start()`	Starts the DMA engine. This is not required when `alt_avalon_sgdma_do_async_transfer()` and `alt_avalon_sgdma_do_sync_transfer()` are used.
`alt_avalon_sgdma_stop()`	Stops the DMA engine. This is not required when `alt_avalon_sgdma_do_async_transfer()` and `alt_avalon_sgdma_do_sync_transfer()` are used.
`alt_avalon_sgdma_open()`	Returns a pointer to the SG-DMA controller with the given name.

Table 5.3: Function List

5.5 VGA Controller: The VGA controller IP core generates the timing signals required by the on-board VGA DAC on the DE-series

boards and Terasic's LCD with touchscreen daughtercards. In our project we use this controller IP core to view the data on the LCD monitor for the display and verification. Data is provided to the VGA Controller via its Avalon Streaming Interface. The controller takes the incoming data. Then it adds the suitable VGA timing signals and then sends that information to either the on-board VGA DAC(digital to analog) or the LCD with touchscreen daughtercard[23][28].

The VGA Controller core generates the timing signals as well as vertical and horizontal synchronization signals. The timing information generated by the VGA Controller core produces screen resolutions of 640 X 480, 800 X 480 and 800 X 600 pixels for the VGA DAC, the LCD with touchscreen (TRDB_LTM) and the 8 inch LCD on the tPad, respectively.



The parameters to be assigned for the Qsys configuration wizard are:

• DE-Series Board— Specifies the Altera DE-series board that the system is being designed for. For our project we have used DE0.

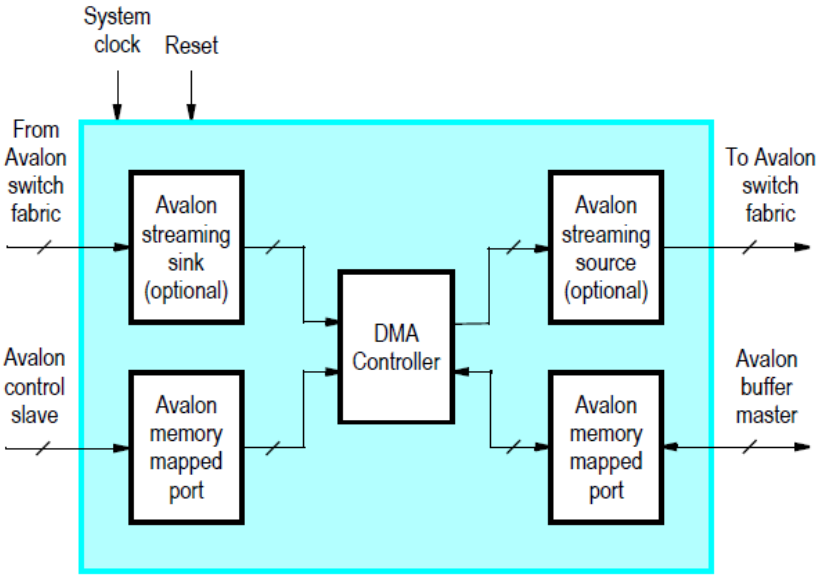
• Video Out Device—Specifies the VGA compatible device being used, and by extension the screen resolution. We have chosen VGA connector to suit our purpose.

5.6 Video DMA Controller: The DMA Controller IP core stores and retrieves video frames to and from memory. When in the "from stream to memory" mode, the core stores frames from an incoming stream to an external memory. The core uses its Avalon

Memory-mapped (MM) master interface to send the data to the memory. When in the "from memory to stream" mode, the

DMA controller uses its Avalon memory-mapped master interface to read video frames from an external memory.

Then, it sends those video frames out by means of its Avalon streaming interface [28].



Block Diagram for DMA controller core.

The DMA controller's configuration wizard is used to specify the desired characteristics. Such as-

- Mode
    - DMA Direction— specifies whether a video stream is to be stored to or retrieved from memory.
- Addressing Parameters
    - Addressing Mode— specifies the addressing mode.
    - Default Buffer Start Address— the start address of the buffer upon reset.
    - Default Back Buffer Start Address—the start address of the back buffer upon reset (can be equal to the Default Buffer Start Address, if no back buffer is desired).
    - Frame Resolution
        - Width (# of pixels) — specifies the incoming stream's width.
        - Height (# of lines) — specifies the incoming stream's height.

- Pixel Format
  - Color Bits— specifies he number of bits per color plane.
  - Color Planes— specifies the number of color planes

5.7 Fast Fourier transform (FFT) generated from megawizard:

The FFT MegaCore IP (which should be bought or can be used when I licensed version of Quartus is used) function is a high performance, highly-parameterizable FastFourier transform (FFT) processor. The FFT MegaCore function implements a complex FFT or inverse FFT (IFFT) for high-performance applications [29]. The FFT MegaCore function implements two architectures:

- Fixed transform size architecture.
- Variable streaming architecture.

To use this core installation and licensing procedures must be followed.

The FFT MegaCore function supports the following design flows:

- DSP Builder: Use this flow if you want to create a DSP Builder model that includes a FFT MegaCore function variation.
- MegaWizard Plug-In Manager: Use this flow if you would like to create a FFT MegaCore function variation that you can instantiate manually in your design.

In our project we have chosen the Mega Wizard Plug-In Manager.

The MegaWizard Plug-in Manager flow allows you to customize an FFT MegaCore function, and manually integrate the MegaCore function variation into a Quartus II design.

The steps are:

1. Create a new project using the **New Project Wizard** available from the File menu in the Quartus II software.

2. Launch **MegaWizard Plug-in Manager** from the Tools menu, and select the option to create a new custom megafunction variation.

Then we parameterized the core according to our purpose and suitability.

In our project we have chosen Input/output data flow as **'Streaming'** according to the suitability of the project.

To set up simulation, from the IP tool bench, the Step2: set up simulation and then Generate Simulation Model is turned on.

The Language we used was Verilog HDL. And then to generate the mega core, we have selected Generate from the IP tool bench.

A list of files will be generated.

After reviewing the generation report, we have to click YES on the Quartus II IP files prompt to add the **.qip** file to the current

Quartus II project.

5.8 Creating FFT block in Qsys:

In our project for easier manipulation and interconnectivity we transformed this mega core function into 'Qsys' IP following few steps. Mega core functions can be included as a new component in the 'Qsys' IP library. The Steps are given below.

- MegaCore Function FFT is not supported by Qsys, launched from the Tool menu of the Quartus II MegaWizard Plug-In Manager.
- From MegaWizard Plug-In Manager, a new custom megafunction variation is selected.
- Megafunction of FFT on the next page is created.
- In MegaCore Function, we clicked Parameterize.
- In Parameters tab, specified the size of the FFT and the Target Device Family.
- In Architecture tab, specified the I / O Data Flow. This time we have chosen Streaming.
- When finished, clicked the Generate screen FFT MegaCore Function.
- MegaCore Function is generated and added to the fft.qip Files of Quartus II.
- In order to capture the 'Qsys' the FFT MegaCore Function that is generated by MegaWizard, a wrapper module is created.
- To be added as a new component of the 'Qsys' FFT MegaCore Function, clicked New Component from the Component Library tab.
- In the HDL Files Tags Component Editor, created in as a Top Level Module, the wrapper.v is added.
- Signals in the next tab to set the Signal Type and Interface.
- Then opened the Interfaces tab. Error Master has no read or write interface that is eliminated when we clicked the Remove Interfaces with No Signals button.

- Error Interface must have an associated reset; the Associated Reset was resolved by choosing the appropriate reset signal from the pull-down menu.
- When a component is successfully generated, it is added as a new component to the Library. We added this FFT to the system.
- Then other connections are given according to our project's need [31].

5.9 On-chip Memory (RAM or ROM):

Altera FPGAs include on-chip memory blocks that can be used as RAM or ROM in 'Qsys' systems. On-chip memory has the following benefits for 'Qsys' systems:

- On-chip memory has fast access time, compared to off-chip memory.
- 'Qsys' Builder automatically instantiates on-chip memory inside the 'Qsys' system, so there is no fuss about making any manual connections.
- Certain memory blocks can have initialized contents when the FPGA powers up. This feature is useful, for example, for storing data constants or processor boot code.
- On-chip memories support dual port accesses, allowing two masters to access the same memory concurrently [23].

The configuration wizard for the On-chip Memory (RAM or ROM) component has the following options:

Memory type, Size, and Read latency.

Memory Type

The Memory type options define the structure of the on-chip memory:

- RAM (writable)—this setting creates a readable and writable memory.
- ROM (read only)—this setting creates a read-only memory.
- Dual-port access—this setting creates a memory component with two slaves, which allows two masters to access the memory simultaneously.
- Block type—this setting directs the Quartus II software to use a specific type of memory block when fitting the on-chip memory in the FPGA.

Because of the constraints on some memory types, it is frequently best to use the Auto setting. Auto allows the Quartus II software to choose a type and the other settings direct the Quartus II software to select a particular type.

Size

The Size options define the size and width of the memory.

- Data width—this setting determines the data width of the memory. The available choices are 8, 16, 32, 64, 128, 256, 512, or 1024 bits. Assign Data width to match the width of the master that accesses this memory the most frequently or has the most critical throughput requirements. Suppose if we connect the on-chip memory to the data master of a Nios II processor, we should set the data width of the on-chip memory to 32 bits, the same as the data-width of the Nios II data master. Otherwise, the access latency could be longer than one cycle because the Avalon interconnects fabric performs width translation.
- Total memory size—this setting determines the total size of the on-chip memory block. The total memory size must be less than the available memory in the target FPGA.
- Minimize memory block usage (may impact fmax)—Minimize memory block usage (may impact fmax)—this option is only available for devices that include M4K memory blocks. But we are **M9k** memory blocks in our system [24].

Read Latency

On-chip memory components use synchronous, pipelined Avalon-MM (Memory Mapped) slaves.

Non-Default Memory Initialization

For ROM memories, we can specify your own initialization file by selecting Enable non-default initialization file. This option allows the file you specify to be used to initialize the ROM in place of the default initialization file created by 'Qsys'.

Enable In-System Memory Content Editor Feature

Enables a JTAG interface used to read and write to the RAM while it is operating. We can use this interface to update or read the contents of the memory from your host PC. That is, on-chip memory contents van be viewed from this feature.

5.10 Nios II (processor): is one of the most resourceful and versatile embedded processors. Like any other processor, it interprets program instructions and processes data, makes the appropriate services available to other parts of the system, presents user interfaces and interprets the user input [26][27].

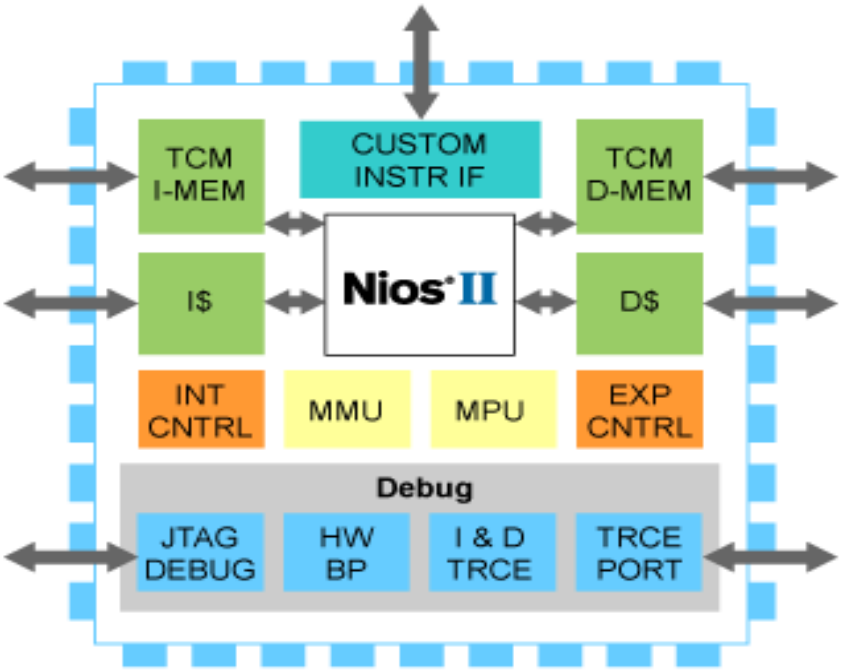


Figure 5.4 : Nios 2 processor

This processor is the most widely used soft processor in the FPGA industry. The Nios II processor delivers unparalleled flexibility and performance in cost-sensitive, real-time, ASIC-optimized, safety critical and applications processing needs. Nios II comprises three configurable cores which we have selected on the basis of individual's design needs.

Nios II/f—The Nios II/f "fast" processor is designed for superior performance while presenting the majority configuration options which are unavailable in the other Nios II processors.

- Nios II/s—The Nios II/s "standard" processor is designed for small size while maintaining fair performance.
- Nios II/e—The Nios II/e "economy" processor is designed for the smallest possible processor size while providing sufficient performance.

A Summary of Features Supported by the Nios II processor is listed below.

- MMU (memory management unit).
- Memory protection unit (MPU).
- External Vector Interrupt Controller with up to 32 interrupts per controller.
- Advanced exception support.
- Separate instruction and data caches (configurable from 512 bytes to 64 KB).

- Access to up to 2 GB of external address space.
- Optional tightly-coupled memory for instructions and data.
- Up to six-stage pipeline to achieve maximum MIPS* (*Dhrystones 2.1 benchmark) per MHz.
- Single-cycle hardware multiplies and barrel shifter.
- Hardware divides option.
- Dynamic branch prediction.
- Up to 256 custom instructions and unlimited hardware accelerators.
- Configurable JTAG debug module.
- Optional JTAG debug module enhancements, including hardware breakpoints, data triggers, and real-time trace.

5.11 Hardware Abstraction Layer:

The HAL serves as a device driver package for Nios II processor systems. The HAL is a lightweight embedded runtime environment that provides a simple device driver interface for programs to connect to the underlying hardware. Moreover, HAL device driver abstraction provides a clear distinction between application and device driver software. The HAL application program interface (API) is integrated with the ANSI C standard library. The HALAPI allows us to access devices and files using familiar C library functions. The Nios II software development tools extract system information from our SOPC Information File (**.sopcinfo**). Most noteworthy thing is that we need not to write low-level routines to establish basic communication with the hardware. Therefore, Application programmers call the ANSI C or HAL API to access hardware, rather than calling your driver routines directly. HAL does not support MPU (Memory Protection Unit) and MMU (Memory Management Unit) hardware [30].

HAL Architecture:

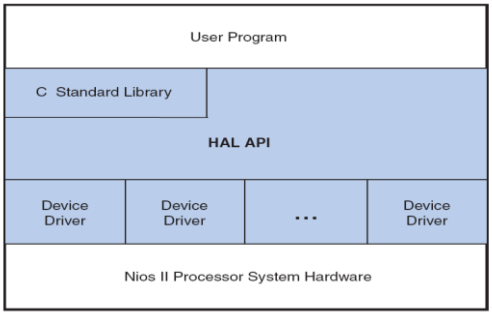


Figure 5.5: HAL (Hardware Abstraction Layer) Architecture

The HAL provides the following services:

- Integration with the newlib ANSI C standard library—provides the familiar C standard library functions.
- Device drivers—provide access to each device in the system.
- The HAL API—provides a consistent, standard interface to HAL services, such as device access, interrupt handling, and alarm facilities.
- System initialization—Performs initialization tasks for the processor and the runtime environment before main ().
- Device initialization—Instantiates and initializes each device in the system before main () runs.

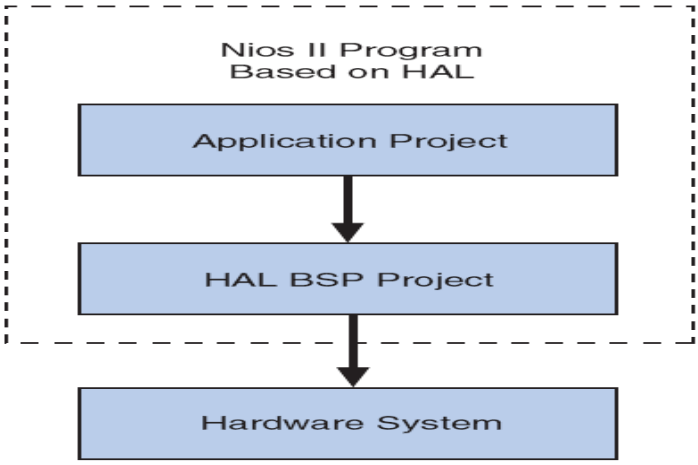


Figure 5.6: Nios II HAL Project Structure

Every HAL-based Nios II program consists of two Nios II projects. One is the user application project and another one is HAL BSP Project. The HAL drivers relevant to your hardware system are incorporated in the BSP project. The BSP project depends on the hardware system, defined by a SOPC Information File (**.sopcinfo**).

The procedure we have followed with the assistance of the HAL library is given as a flow chart below:

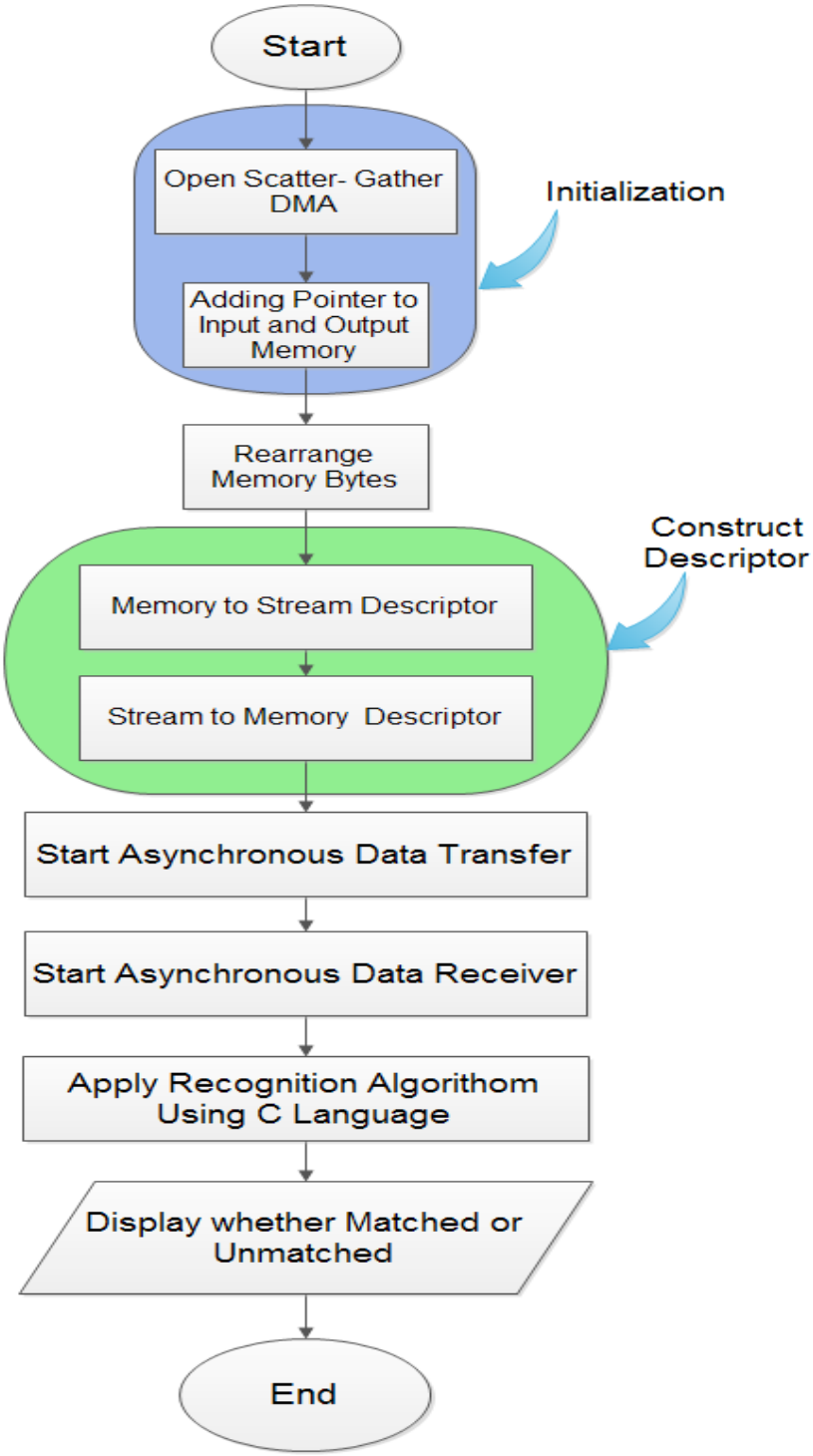


Figure 5.7: Flowchart for Nios II Instruction.

Our initial task was to read image from SD ram and after that to store the fast fourier transformed (FFT) image in another place of the SD ram so that it does not overlap. SD ram is connected to Scatter Gather DMA for asynchronous data transfer. Since SD ram has no software-configurable settings and no memory-mapped registers we have programmed SGDMA using it's built in library routine. For this, we have first opened SGDMA and added pointer to input output memory as part of the initialization. Before construct descriptor it was necessary to rearrange memory blocks. Then we called the two built in function and passed the required parameters to that function.

alt_avalon_sgdma_construct_mem_to_stream_desc()

alt_avalon_sgdma_construct_stream_to_mem_desc()

After that it was ready for asynchronous transfer and receiver. When transmission and receive was complete we used this value to compare it with existing database for recognition.

After configuring and adding all the components we finally get a 'Qsys' system content output. The connections to the components are given as per our project's requirements.

The image is given below:

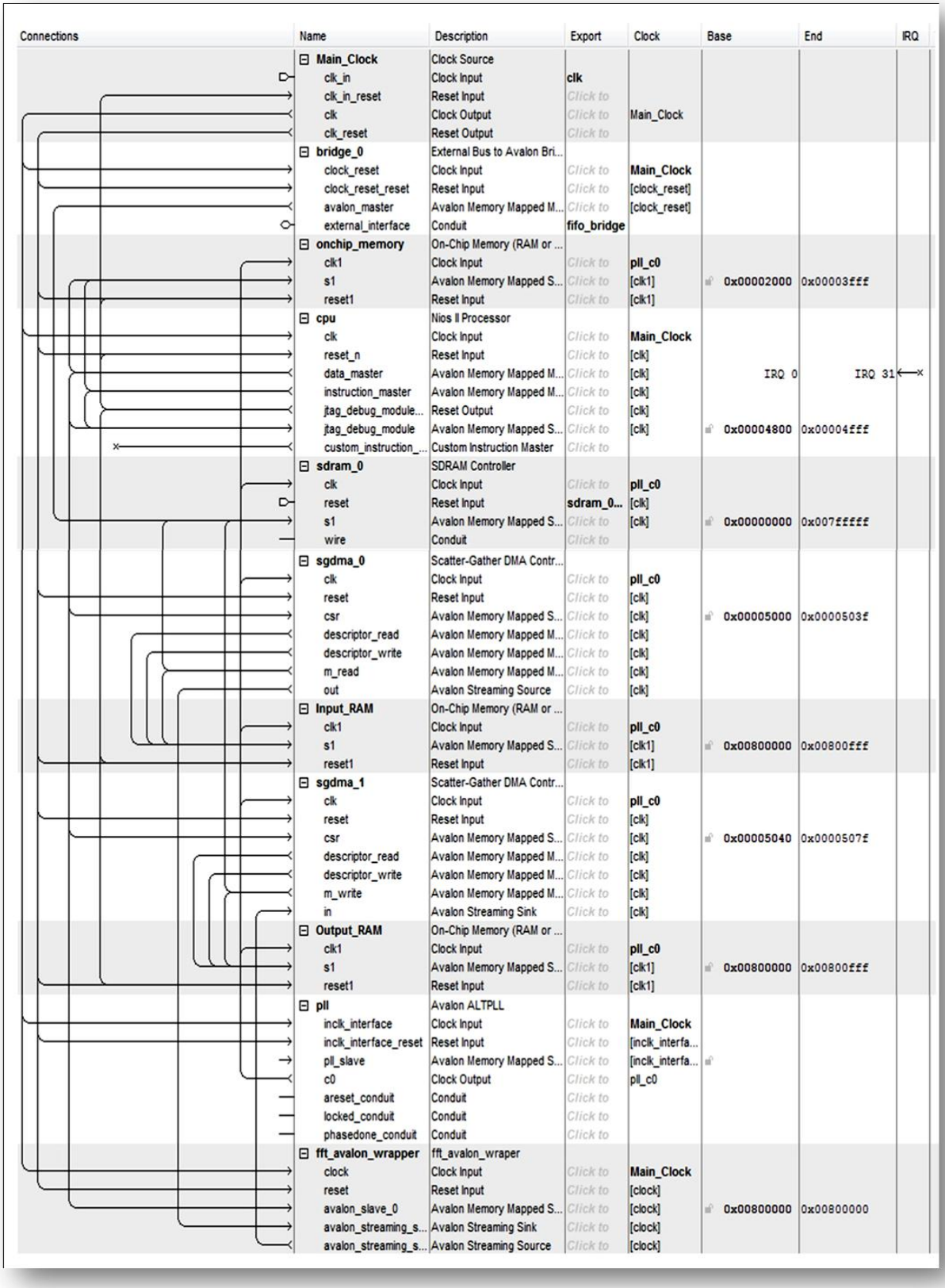


Figure 5.8: Qsys system content

After the connections were made, Verilog codes had to be written in order for the parameters of some of the components work.

That Is the HDL example generated from these 'Qsys' system output, these had to be assigned input and output signals.

As for example we had to write codes for the External bus to Avalon bridge work.

The logic we have created is shown by the help of a Flow chart below:

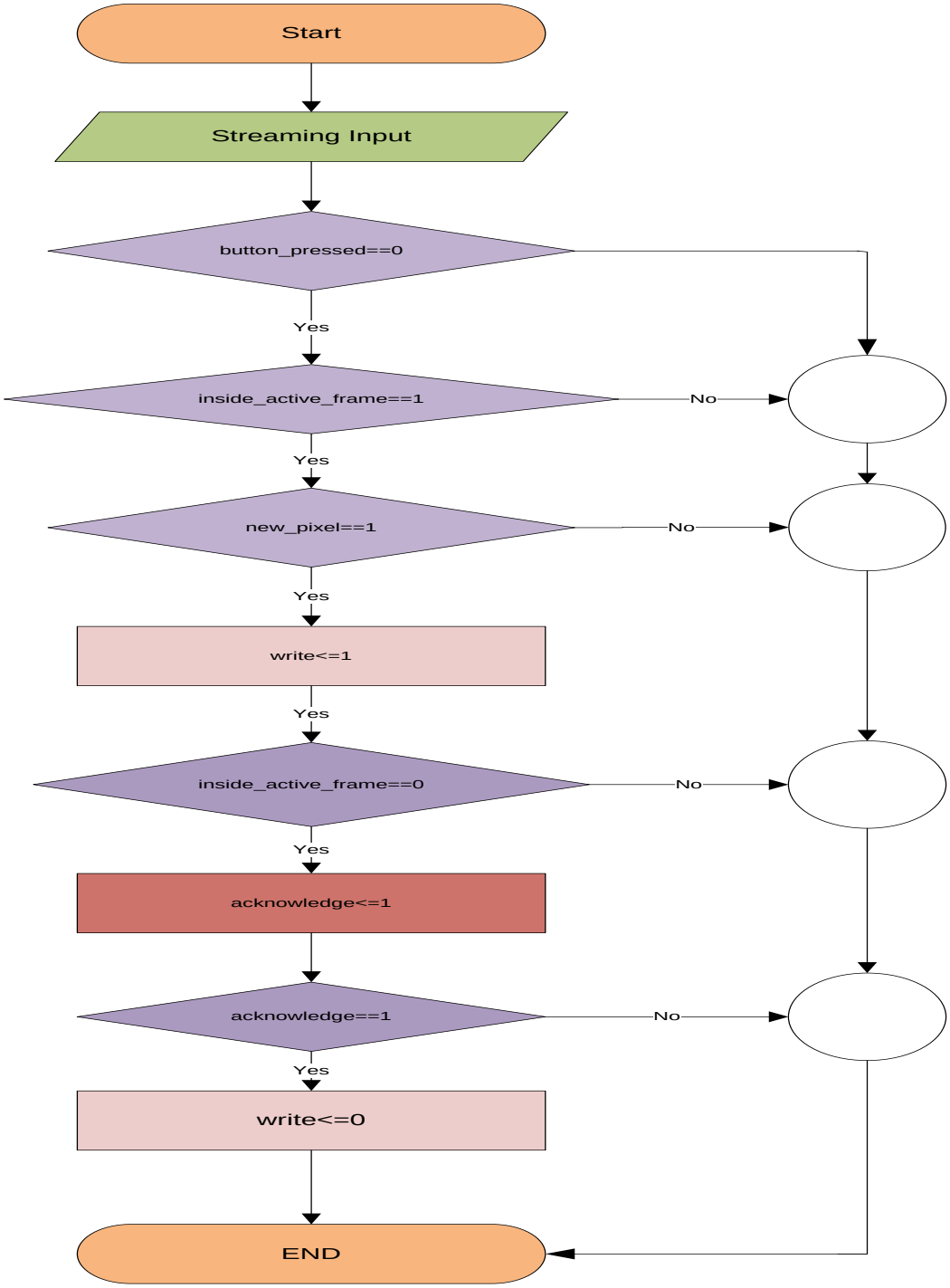


Figure 5.9: Flowchart for valid frame capture

The logic we proposed was when the button for capturing the image is pressed; a frame from the streaming data will be captured.

First we checked if the current frame, just when the button is pressed, meets the ‘inside active frame’ conditions. That is if the frame has a dimension of an active image (1944X 2592), it must be starting from x=0 and y=0, till the end of an active image frame. If this condition is satisfied, then this data will be transferred during the immediate next Odval (rising edge of the driving clock).This condition is defined by the ‘new pixel’ block. This will make the ‘write’ signal high. After writing the whole frame to SDRAM ‘inside active frame’ will be 0 and this will generate an ‘Acknowledge’ from the external bus. When Acknowledge is high (1), the write signal is made (0) to stop the writing process as we have successfully collected one frame, that is, the data of an image.

By following the above steps we tried to implement our proposed architecture in hardware.

Chapter 6

Results and Discussion:

6.1 Software:

Principal component analysis decomposes the covariance structure of the dependent variables into orthogonal components by calculating the eigenvalues and eigenvectors of the data covariance matrix. Eigenvalues assist in making decisions about the number of orthogonal components that will be used in further analysis, while eigenvectors assist in determining the relationship between the original variables and these new components. Eigenvalues and eigenvectors transform the original variable space into a 'new' set of variables, called principal components (PCs). [32] The First Fourier Transform (FFT) is the most common method of face recognition with respect to frequency spectrum. The FFT variables are ranked according to their variance, thereby reflecting a decreasing importance as to their ability to capture the whole information content of the original data-set for signal reconstruction purposes. By virtue of its ability to reduce the complexity of the resulting feature space, the PCA is widely used in a number of pattern recognition applications. [33]

Two face recognition strategies i.e. PCA (Principal Component Analysis) and FFT (Fast Fourier Transform) were implemented in our project. If we want to recognize the same image of a student with FFT algorithm the accuracy is 100% and if we take a slight changed expression of the same person the accuracy is 40%, where the PCA gives 70% accuracy in changed expressions.

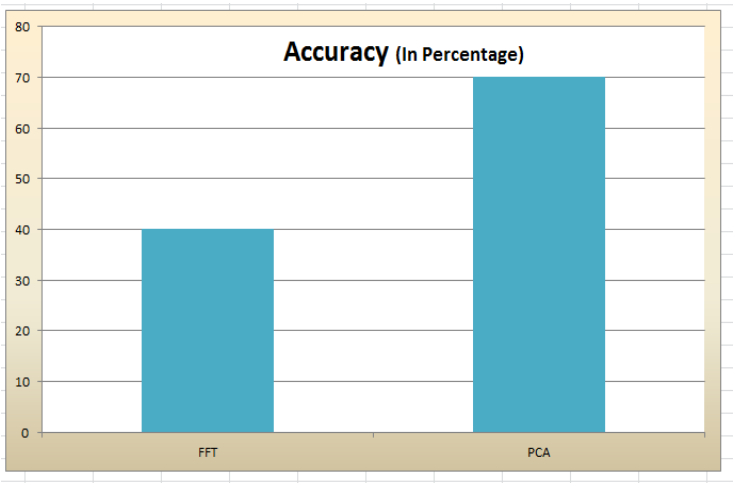


Figure 6.1: Accuracy rate of Face Recognition for PCA and FFT

Principal Component Analysis gave better results for varying poses. Fast Fourier Transform can recognize faces but if any person smiles and if that images contains too much light are taken into account. The results are pretty good for the test samples that we have considered.

Figure 6.2: Recognition result with FFT based algorithm

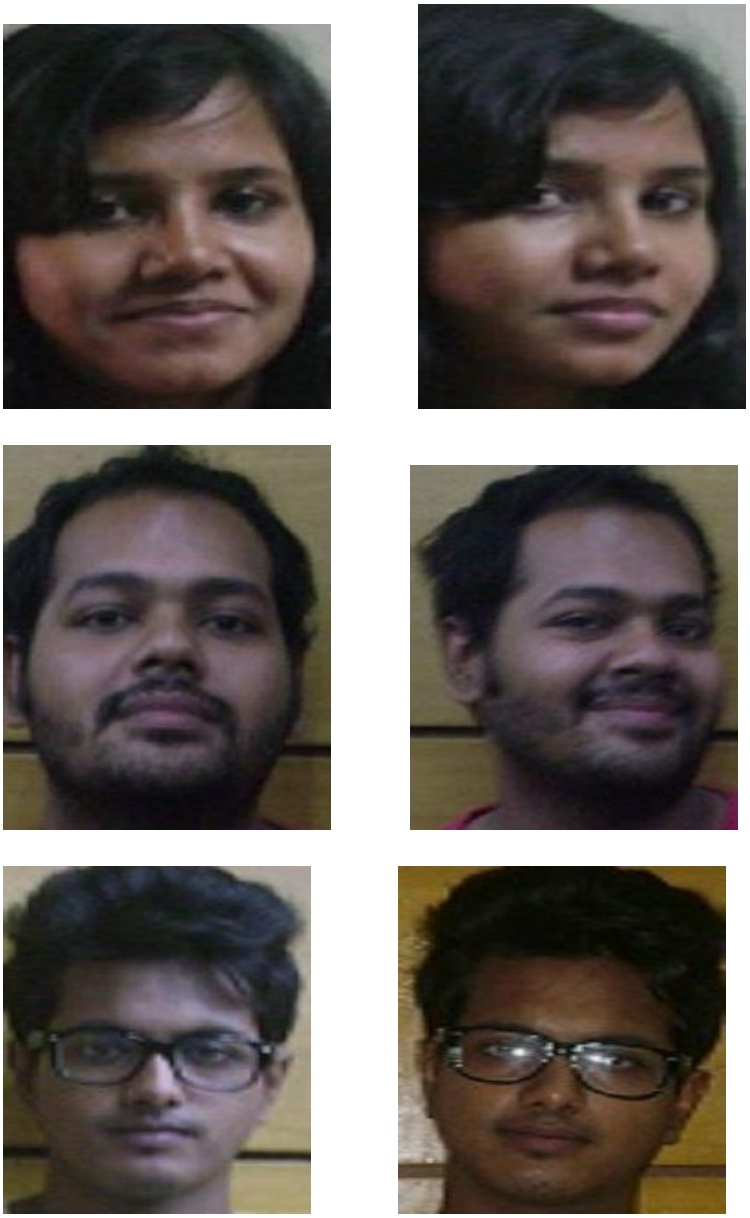


Figure 6.3: Recognition result with PCA based algorithm

6.2Hardware:

Individually we could verify many components of our proposed architecture like

- VGA display
- We have the grayscale data into the SDRAM.
- Using DMA1 we could call the stored image and put it in the FFT work by curetting a code in NIOS2
- Using DMA2 we stored the FFT image into the SDRAM again.
- We have prepared an approximate code for final recognition, in NIOS2.

6.3 Limitation:

6.3.1 Software:

The FFT is a complicated and non-effective algorithm still we tried to implement it to FPGA board as our topic contains two huge areas that is image processing and implanting it on the FPGA board. Since FPGA implementation is our priority, we started it with an easier algorithm using FFT. FFT may not be as perfect as other algorithms, however to reduce noise we are emphasizing on converting the image into frequency domain.

6.3.2 Hardware:

After we generate an IP for FFT from mega wizard .sof is not generated. We are assuming that this problem is due to the unavailability of the licensed version. Therefore we could not verify our proposed architecture. But individually we could verify many components of our proposed architecture as for example, if the data could be saved in the SD-RAM and FFT core is working.

6.4 Future Work:

As future improvements PCA algorithm could be implemented on FPGA. This algorithm must be implemented directly on the FPGA, to accelerate the encoding process and not overload the processor. This hardware implementation lead to a higher frame rate encoding, less data per frame, consequently less bandwidth used, and offer a wider range of choices for compression methods. It is also possible to implement several vision detection algorithms in software. It is possible to use already implemented algorithms, cross-compile them to this TERASIC (TRDB_D5M) camera. Other hardware or software applications can be implemented on this system, since the hardware present on the FPGA can be changed or increased with HDL modules, and the operating system allows easy software development (taking into account system constraints).

Another way in which the present implementation can be improved is by changing the input output process. The input output block remains idle when processing is going on. We cannot enter new sets of data as long as the entered set has been completely computed. The new proposed architectural modification takes care of the fact that when computation of one is going on, input and output blocks are not staying idle. This will lead to kind of pipelined input output architecture for the whole block.

6.5 Conclusion:

Face recognition is biometric identification by scanning a person's face and matching it against a library of known faces. The end result of this project focuses on developing a Face recognition system on FPGA. An advantage of developing this system on a FPGA was the ability to update the functionalities or correct any error by re-programming the FPGA with a system's new version. This system is targeted for access control, face databases, face identification, human computer interaction, law enforcement, smart cards, featuring important characteristics to achieve this goal.

Hardware development was done using an Altera DE0 development board with a Cyclone III FPGA, which was found to be appropriate for multimedia projects. The CMOS sensor from TERASIC (TRDB-D5M) with 5 megapixel resolution is from the same vendor and was specially made to use with DE0 board. This development kit includes Verilog HDL examples for the image acquisition, conversion and image storage. Some of them were used with a couple of changes to meet project's needs.

Using SOPC Builder or QSYS it was possible to implement a Nios II soft-core processor with all necessary options enabled. A Nios II system includes a processor core, an UART peripheral, an interval timer, input/output components, a SDRAM memory controller, a LCD module, an Ethernet interface, a SD/MMC card interface, and a CMOS slave controller. All of these modules use an Avalon Memory-Mapped interface and are connected using system interconnect fabric. The gate ware design was implemented with Verilog HDL using Quartus II 12.0 Web Edition.

This Face Recognition System can be used in any application purpose, but it is optimized for security issue. The fact that this system was developed in FPGA and with an open-source operating system, allows any developer to continue this project and implement more features.

Chapter 7

References:

1. Face Recognition , Editors: Kresimir Delac, Mislav Grgic and Marian Stewart Bartlett.
   IN- TECH, Vienna, Austria, 2008.
2. M. Turk, A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, 3(1), pp. 71-86, 1991.

3. D. Swets, J. Weng, "Using Discrminant Eigenfeatures for Image Retrieval", IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(8), pp. 831-836,1996.

4. P.N. Belhumeur, J.P. Hespanha, D.J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection." IEEE Trans. Pattern Anal. Machine Intell., vol. 19, pp. 711-720, May 1997.

5. J. Karhunen, E. Oja, L. Wang, R. Vigario, and J. Joutesensalo. A class of neural netowrks for independent component analysis. IEEE Trans. On Neural Networks, 8(3): 486-504, 1997.

6. P. Comon. Independent Component Analysis, a new concept, Signal Processing, 36:287-314,
   1994.

7. M.S. Bartlett, J.R. Movellan, T.J. Sejnowski, "Face Recognition by Independent Component Analysis", IEEE Trans. On Neural Networks, Vol. 13, No. 6, November 2002, pp. 1450-1464.

8. A. Kadyrov and M. Petrou, " The Trace Transform and Its Applications," IEEE Transactions on, Pattern Analysis and Machine Intelligence, Vol. 23, No. 8, pp. 811-828, August 2001.

9. http://www.authorstream.com/Presentation/gaurav22788-300314-face-recognition-using-neural-networks-final-present-2-science-technology-ppt-powerpoint /

10. dspGuru by Lowegian International , Digital Signal Processing Central, Fast Fourier Transform, (FFT) FAQ, http://www.dspguru.com/dsp/faqs/fft

11.The Scientist and Engineer's Guide to Digital Signal Processing by Steven W. Smith. Chap-12, Fast Fourier Transform.

12. James W. Cooley and John W. Tukey. " An algorithm for the machine calculation of complex Fourier series, "Math, Comput 19, 297-301 (1965).

13. Raghu Muthyalam, Implementation of Fast Fourier Transform for Image Processing in DirectX- 10.

14.ImageMagick v6 Examples- Fourier Transforms. http://www.imagemagick.org/Usage/fourier/

15. R. Gonzales, R. Woods Digital Image Processing, Addison- Wesley Publishing Company, 1992, pp 81-125.

16. A. Jain Fundamentals of Digital Image Processing, Prentice Hall 1989, pp 15-20.

17. http://answers.yahoo.com/question/index?qid=1005120802386

18. How to do a 2D Fourier Transform In matlab by Eric Verner, Matlab Geek, http://matlabgeeks.com/tips-tutorials/how-to-do-a-2-d-fourier- transform-in-matlab/

19. How to plot 2D FFT in Matlab , stackoverflow, http://stackoverflow.com/questions/13549186/how-to-plot-a-2d-fft-in-matlab

20. Terasic Technologies. (2011). DE0 User Manual .

21. http://www.i2c-bus.org/i2c-Interface
22. Terasic TRDB-D5M Hardware specification. 2010. www.terasic.com

23.http://www.altera.com/literature/ug/ug_embedded_ip.pdf

24.http://www.altera.com/literature/ug/ug_sopc_builder.pdf

25.http://www.altera.com/literature/ug/altera_pll.pdf

26.http://www.altera.com/literature/ds/ds_nios2_perf.pdf

27.www.altera.com

28.ftp://ftp.altera.com/up/pub/Altera_Material/9.1/University_Program_IP_Cores/Audio_Video/Video.pdf

29.http://www.altera.com/literature/ug/ug_fft.pdf

30.Altera Corporation. (2011, May). Nios II Software Developer's HandBook. U.S.A.

31.http://blog.goo.ne.jp.

32. Srinivasulu Asadi, Dr.Ch.D.V.Subba Rao and V. Saikrishna, November, 2010; A Comparative Study of Face Recognition with Principal Component Analysis and Cross-Correlation Technique. International Journal of Computer Applications, 10-8, 0975-8887.

33. Abhishek Kesh, Rachit Gupta, Siddharth S. Seth and T. Anish, August, 2004-05; Implement of Fast Fourier Transform (FFT) on FPGA using Verilog HDL. An Advanced VLSI-Design-Lab Term-Project, AVDL"76, Kharagpur India.
34. external_bus_to_avalon_bridge.pdf
35. Face Recognition On FPGA (Final year project).
36. Eigenfaces for recognition (MIT).

Chapter 8

Appendix

8.1 FFT Matlab code:

```
clc
clear all
sdirectory = 'Train Database';
tifffiles = dir([sdirectory '/*.jpg']);
 %length(tifffiles)
 I = cell(1,numel( tifffiles));
for k = 1:length(tifffiles)
filename = [sdirectory '/' tifffiles(k).name];
I{k} = imread(filename);
Rb=imresize(I{k},[50,50]);
  J=rgb2gray(Rb);
fftb=fft2(J);
figure,imshow(I{k})
imshow(Rb)
figure, imshow(J)
figure, imshow(uint8(fftb))
Absolutevalue_train{k}=abs(fftb);
mean_train{k}=mean2(Absolutevalue_train{k});
end
sdirectory1 = 'Test Database';
 tifffiles1= dir([sdirectory1 '/*.jpg']);
 %length(tifffiles1)
 Q = cell(1,numel( tifffiles1));
for R = 1:length(tifffiles1)
  filename1 = [sdirectory1 '/' tifffiles1(R).name];
  Q{R} = imread(filename1);
  Rb1=imresize(Q{R},[50,50]);
  J1=rgb2gray(Rb1);
  fftb1=fft2(J1);
figure,imshow(Q{R})
imshow(Rb1)
figure, imshow(J1);
figure, imshow(uint8(fftb1))
Absolutevalue_test{R}=abs(fftb1);
mean_test{R}=mean2(Absolutevalue_test{R});
end
for i= 1:1:length(tifffiles)
    j=1;
difference(i)= mean_train{i}- mean_test{j};
axxx(i)= abs(difference(i));
end
 t=min(axxx);
% to show the equvalent image
for us= 1:1:length(tifffiles)
```

```
post=min(axxx);
compare=axxx(us);
if post==compare
counter=us;
else
end
end

if  t<21     %threshold level
disp('Matched');
percentage=(((21-t)/21)*100);
figure,imshow(Q{R})
title([num2str(percentage),'%  Matched']);
figure,imshow(I{counter});
title('Equivalent Database Image');
else
disp('Not Matched')
percentage=(((t-21)/t)*100);
figure,imshow(Q{R})
title([num2str(percentage),'% Deviated   Not Matched']);
end
```

8.2 PCA based Face Recognition Matlab code: (Collected)

```
% A sample script, which shows the usage of functions, included in
% PCA-based face recognition system (Eigenface method)
%
% See also: CREATEDATABASE, EIGENFACECORE, RECOGNITION
clear all
clc
close all
% You can customize and fix initial directory paths
TrainDatabasePath = uigetdir('D:\Program Files\MATLAB\R2006a\work', 'Select training database path' );
TestDatabasePath = uigetdir('D:\Program Files\MATLAB\R2006a\work', 'Select test database path');
prompt = {'Enter test image name (a number between 1 to 10):'};
dlg_title = 'Input of PCA-Based Face Recognition System';
num_lines= 1;
def = {'1'};
TestImage  =inputdlg(prompt,dlg_title,num_lines,def);
TestImage = strcat(TestDatabasePath,'\',char(TestImage),'.jpg');
im = imread(TestImage);
%Time counting
start_time=cputime;
T = CreateDatabase(TrainDatabasePath);
[m, A, Eigenfaces] = EigenfaceCore(T);OutputName = Recognition(TestImage, m, A, Eigenfaces);
SelectedImage = strcat(TrainDatabasePath,'\',OutputName);
```

```
SelectedImage = imread(SelectedImage);

%Time resutl
Time=cputime-start_time;
imshow(im)
title('Test Image');
figure,imshow(SelectedImage);
title('Equivalent Image');
str = strcat('Matched image is :  ',OutputName);
disp(str)
```

8.3 FPGA Code:

8.3.1 Storing data from camera module to SDRAM

```
always @(posedge CLOCK_50 or negedge DLY_RST_2)
begin
if (~DLY_RST_2)
        begin
        write<= 0;
        end
else
begin
        if(write)
        begin
                if(acknowledge)
begin
                write<= 0;
end
        end
        else
        begin
        if(button_pressed==0) //used twice
        begin
                if(inside_active_frame)
                begin
                if(new_pixel)
begin
                        write<=1;
                end
end
end
end
end
end

always @( posedge CLOCK_50 or negedge DLY_RST_2)
begin
```

```
if (~DLY_RST_2)
begin
        inside_active_frame<= 0;
end
else if(X_Cont == 0 &&Y_Cont == 0)//when the inside a frame
begin
        inside_active_frame<= 1;
        end
        else if(X_Cont == 2047 &&Y_Cont == 1943 &&inside_active_frame)
        begin
                        inside_active_frame<=0;

        end
end

regprev_odval;
wirebusodval;
assignbusodval=sCCD_DVAL;
always @( posedge CLOCK_50 or negedge DLY_RST_2)
begin
if (~DLY_RST_2)
begin

        if({prev_odval,busodval} == 2'b01)
        begin
                new_pixel<=1;
        prev_odval<= busodval;
        end
        if(write && acknowledge)
        begin
                new_pixel<= 0;
                end
end
        end
```

8.3.2 Interfacing Qsys Component with FPGA board

```
Jan_18th u0 (
       .clk_clk(CLOCK_50), // clk.clk
      .fifo_bridge_write(write), // .write
      .fifo_bridge_write_data(grayscale),  //.write_data
   .fifo_bridge_acknowledge  (acknowledge),  //.acknowledge
      .sdram_clk_clk           (CLOCK_50_2),           //   sdram_clk.clk

.video_vga_controller_0_external_interface_CLK (CLOCK_50),              .
.video_vga_controller_0_external_interface.CLK
.video_vga_controller_0_external_interface_HS  (VGA_HS),                                .
.video_vga_controller_0_external_interface_VS  (VGA_VS),
.video_vga_controller_0_external_interface_R   (VGA_R),
```

```
      .video_vga_controller_0_external_interface_G   (VGA_G),
      .video_vga_controller_0_external_interface_B   (VGA_B)
   );
```

8.3.3 Creating SDRAM allocation

```
always@ (posedgesCCD_DVAL)
begin
if(address_count<5038848)
          begin
address_count<=address_count+1;
          end
else
begin
address_count<=0;
end
end
assignaddress_line=address_count;
```

8.4 Code for SGDMA:

```
/*
Code for SGDMA
 */

#include <stdio.h>
#include "altera_avalon_sgdma.h"
#include "altera_avalon_sgdma_descriptor.h"
#include "altera_avalon_sgdma_regs.h"

int main()
{
          //initialize scatter-gather dma

   //Memory To Stream
alt_sgdma_descriptor *desc;  //1
alt_sgdma_descriptor *next; //2
   alt_u32 *read_addr;//3
alt_u16 length; //leangth//4
intread_fixed;//5
intgenerate_sop;//6
intgenerate_eop;//7
   alt_u8 atlantic_channel;//8
voidalt_avalon_sgdma_construct_mem_to_stream_desc(,
      , , , , ,
      , );
```

```
alt_sgdma_dev *dev;
alt_avalon_sgdma_start(*dev);


 // Stream To Memory

alt_sgdma_descriptor *desc;//1
alt_sgdma_descriptor *next;//4 This does not need to be a complete or functional
   //descriptor, but must be properly allocated.
intwrite_fixed;   //3
   alt_u32 *write_addr; //
   alt_u16 length_or_eop;//5
voidalt_avalon_sgdma_construct_stream_to_mem_desc(,
   , , , );
   //stop scatter gather dma
alt_sgdma_dev *dev;
voidalt_avalon_sgdma_stop();

return 0;
}
```

8.5 Code for recognition in C (NIOS II)

```
#include<stdio.h>
#include<math.h>
#include<algorithm>

float mean(int m, int a[]) {   //mean fuction
int sum=0, i;
for(i=0; i<m; i++)
sum+=a[i];
return((float)sum/m);
}

void main(){
        intn,i,max,min;
        intabsvalue[4][8];
float mean(int,int[]);
        intmeancalc[4];
        int temp[8];
        int diff[4];
        int threshold =10;

        intfft_value[4][8]    =    {{90,42,21,5,-43,-9,-34,2},    {2,42,21,12,-43,-2,-34,2},    {-
2,4,32,100,-90,-2,-24,56}, {45,4,21,5,-43,45,-34,-23}};

for (int i=0;i<4;i++){
                for (int p=0;p<8;p++){
                absvalue[i][p] = abs(fft_value[i][p]);
        printf("abs_value=%d\n",absvalue[i][p]);
```

```
                }
        }

        for (int i=0;i<4;i++){
                for (int p=0;p<8;p++){
                        temp[p]=absvalue[i][p];
                //      printf("temp_value=%d\n",temp[p]);
                }
                meancalc[i] = mean(8,temp);
                printf("meancalc=%d\n",meancalc[i]);
        }

        for (int g=0; g<4; g++){
                diff [g]= meancalc[0] - meancalc[g];
        printf("diff=%d\n",diff[g]);
        }

max=abs(diff[1]);
min=abs(diff[1]);
for(int t=2;t<4;t++)
{
if(max<abs(diff[t]))
max=abs(diff[t]);
if(min>abs(diff[t]))
min=abs(diff[t]);
}
printf("min=%d\n",min);

if (min<threshold){

printf("Image is matched.");
}
else {
printf("Image DO NOT Matched");
}

}
```