DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

FACE AND FINGERPRINT RECOGNITION ON FIELD PROGRAMMABLE GATE ARRAY

by

Enes DİLCAN

October, 2010

İZMİR

FACE AND FINGERPRINT RECOGNITION ON FIELD PROGRAMMABLE GATE ARRAY

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical and Electronics Engineering

by
Enes DİLCAN

October, 2010
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"FACE AND FINGERPRINT RECOGNITION ON FIELD PROGRAMMABLE GATE ARRAY"** completed by **ENES DİLCAN** under supervision of **ASST. PROF. DR. NALAN ERDAŞ ÖZKURT** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Nalan Erdaş ÖZKURT

Supervisor

(Jury Member)	(Jury Member)

Prof.Dr. Mustafa SABUNCU
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to thank my advisor Asst. Prof. Dr. Nalan Erdaş ÖZKURT for her guidance and support in every stage of my research. The technique background and the research experience I have gained under her care will be valuable asset to me in the future.

I also would like to thank to my dear friend Gökhan ÇETİN for his absolute guidance, supports and patience during this research. His endless friendship and encouragement always motivated me.

Finally, I would like to thank Çiğdem COŞKUN and my family for their motivation and never ending support throughout my life.

Enes DİLCAN

FACE AND FINGERPRINT RECOGNITION ON FIELD PROGRAMMABLE GATE ARRAY

ABSTRACT

Biometric recognition refers to use of distinctive physiological and behavioral characteristics for automatically recognizing a person. A number of biometric technologies have been developed such as fingerprint, face, iris and speech are the ones that most commonly used. Feature extraction techniques play important role for biometric recognition system design.

Field Programmable Gate Arrays (FPGAs) are the programmable logic devices that can be configured by the customer after manufacturing. FPGAs are preffered in a variety of applications due to ease of programming with low cost. Applications of FPGAs include digital signal processing, biometric recognition, medical imaging aerospace and defense systems, computer vision and a growing range of other areas.

In this thesis, face and fingerprint recognition systems are implemented on FPGA. This study has two working phases. In the offline training phase, face and fingerprint images are collected by MATLAB. Then, this database is sent to FPGA to extract features. Principal Component Analysis (PCA) is the feature extraction algorithm that is used in this study. After all features of face and fingerprint images are extracted, the features are stored on the memory of FPGA. In the online test phase or recognition phase, the features of test images are extracted then these are compared to restored values of the database from the memory of FPGA. The result of comparison is then displayed. This thesis also proposes a multibiometric recognition system which is constituted from face and fingerprint recognition systems by using the fusion at the decision level.

Keywords : Face recognition, fingerprint recoginiton, multibiometric recognition, FPGA, PCA

SAHADA PROGRAMLANABİLİR KAPI DİZİLERİ ÜZERİNDE YÜZ VE PARMAK İZİ TANIMA

ÖZ

Biyometrik tanıma, otomatik olarak bir kişiyi ayırıcı fiziksel ve davranışsal niteliklerine göre tanımaya karşılık gelir. Çok sayıda biyometrik teknoloji geliştirilmiştir. Parmak izi, yüz, iris ve ses tanıma en yaygın kullanılan biyometrik teknolojilerdir. Özellik çıkarma metotları, biyometrik sistem tasarımında önemli bir rol oynamaktadır.

Sahada Programlanabilir Kapı Dizileri (SPKD), üretimden sonra müşteri tarafından yeniden yapılandırılabilen programlanabilir mantık elemanlarıdır. SPKD'ler düşük maliyetle ve programlanabilme kolaylığı ile çok sayıda uygulamada tercih edilmektedir. SPKD içeren uygulamalar sayısal işaret işleme, biyometrik tanıma, medikal görüntü işleme, uzay ve savunma sistemleri, bilgisayar görüntüsü alanlarında kullanmakta ve kullanım alanları giderek artmaktadır.

Bu tezde, SPKD üzerinde yüz ve parmak izi tanıma sistemi gerçeklenmiştir. Bu sistemin iki çalışma aşaması vardır. Çevrimdışı öğrenme aşamasında, yüz ve parmak izi resimleri MATLAB tarafından toplanır. Daha sonra oluşturulan bu veritabanı öznitelik çıkarma için SPKD'ye gönderilir. Bu çalışmada öznitelik çıkarmak için Temel Bileşen Analizi (TBA) algorithması kullanılmıştır. Yüz ve parmak izindeki tüm öznitelikler çıkarıldıktan sonra, bu öznitelikler SPKD'nin hafızasında saklanır. Çevrimiçi deneme ya da tanıma aşamasında, öncelikle yüz ve parmak izi resimlerinden öznitelikler çıkarılır, daha sonra bu öznitelikler SPKD'nin hafızasında saklanan veritabanı ile karşılaştırılır. Karşılaştırma sonucu, tanımlama sonucudur. Bu tezde ayrıca yüz ve parmak izi tanıma sistemlerininden oluşan, birleştirmenin karar verme seviyesinde yapıldığı çoklu biyometrik tanıma sistemi tasarlanmıştır.

Anahtar Sözcükler : Yüz tanıma, parmak izi tanıma, çoklu biyometrik tanıma, Sahada Programlanabilir Kapı Dizileri, Temel Bileşen Analizi

CONTENTS

Page

CHAPTER ONE
INTRODUCTION

1.1 Biometric Systems

Biometric recognition term refers to the use of distinctive physiological and behavioral characteristics that are called biometric identifiers for automatically recognizing individuals (Maltoni, Maio, Jain, & Prabhakar, 2003). A number of biometric technologies have been developed and several of them are being used in a variety of applications in differet areas. Among these, face, iris, fingerprints, speech and hand geometry are the ones that most commonly used in biometric systems. Each biometric has its strengths and weakness, so choice of a particular biometric typically depends on the requirements of an application.

A biometric system is a pattern recognition system that responsible for recognizing a person by determining the authenticity of a specific physiological and/or behavioral characteristic possessed by that person. The most important issue in designing a practical biometric system is to determine how this biometric individual is recognized. Depending on the application, a biometric system may be called a verification system or an identification system:

• a verification system authenticates a person’s identity by comparing the captured individual characteristic with his/her own biometric template(s) that is stored in the system. One-to-one comparison is done to determine whether the identity claimed by the individual is true. A verification system either rejects or accepts the user,

• an identification system recognizes an individual characteristic by searching the entire template database. One-to-many comparisons is done to establish the identity of the individual. The identification system establishes a subject’s identity without the subject having to claim an identity.

1.2 Multibiometric Systems

Most of the biometric systems deployed in real world are unimodal and they are relied on the evidence of a single source of information. Unimodal biometric systems have to contend with a variety of problems such as noisy data, intra-class variations, spoof attacks and unacceptable error rates (Ross, & Jain, 2004). Multimodal or multibiometric systems, which include multiple sources of information, are offered to overcome these limitations in unimodal biometric systems.

Multibiometric systems represent the fusion of two or more unimodal biometric systems. The fusion can occur at the data or feature level, match score level and decision level. Figure 1.1 shows the levels of fusion in a biomodal biometric system (Ross, & Jain, 2004). Note that; FU, MM and DM stand for fusion, matching and decision module respectively in Figure 1.1.

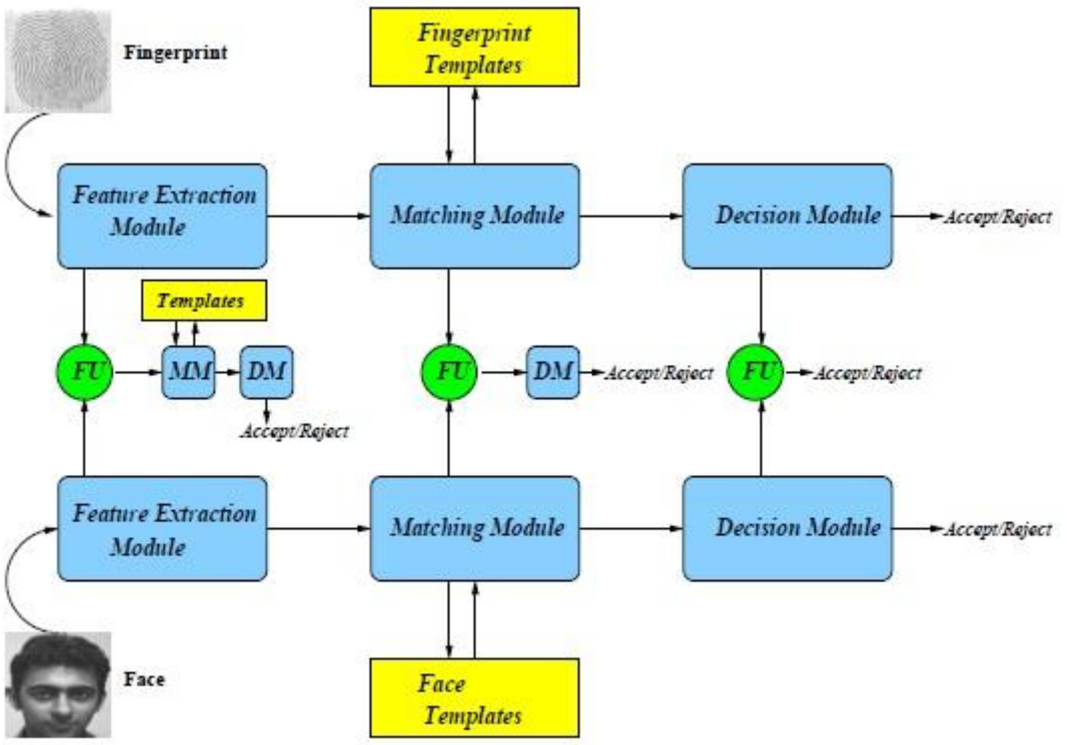


Figure 1.1 Levels of a fusion in a biomdoal biometric system (Ross, & Jain, 2004).

Depending on the number of traits, sensors, and feature sets used, a variety of scenarios are possible in a multimodal biometric system (Ross, & Jain, 2004). Figure 1.2 shows that these various scenarios (Prabhakar, & Jain, 2002);

1) multiple sensors for tracking the same biometric behavior such as using optical and capacitive sensors together;
2) multiple biometrics for the same person such as using face and fingerprint;
3) multiple units such as right index and middle fingers of a person;
4) multiple snapshots of the same biometric such as taking for two templates of the right finger of a person
5) multiple matchers such as using minutiae and non-minutiae based matchers are the examples of the applications of multibiometric systems.

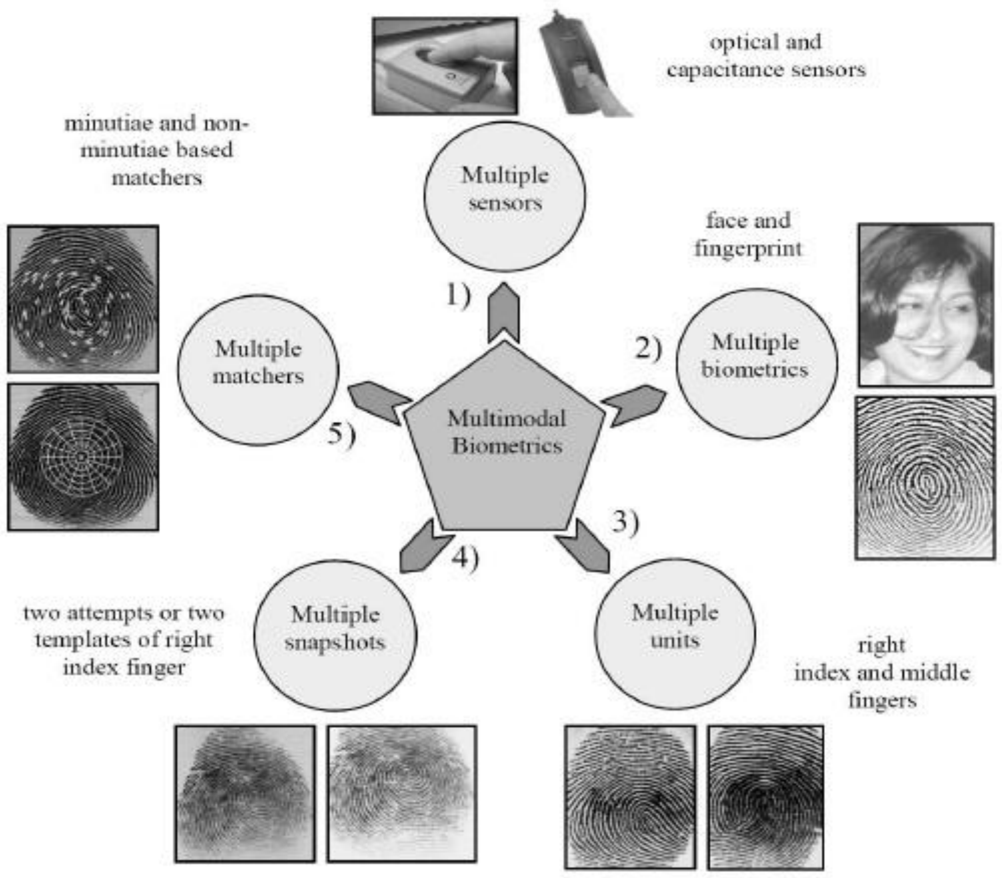


Figure 1.2 Scenarios in a multimodal biometric system (Prabhakar, & Jain, 2002).

Several factors should be considered when designing a multibiometric system. Some of these factors are;

- the choice and the number of biometric behaviors,
- the level in biometric system at which information provided by multiple types should be integrated,
- the methodology adopted to integrate the information,
- the cost versus matching performance trade-off,
- system is user friendly or not (Anwar, Rahman, & Azad, 2009).

By combining multiple sources of information, these systems improve matching performance, increase population coverage, deter spoofing, and facilitate indexing (Ross, & Jain, 2004). So, multibiometric systems are expected to be more reliable due to presence of multiple independent pieces of evidence.

1.3 History of Face Recognition Systems

The first way to do face recognition is to look at the major features of the face and compare these features with the same features on the other faces. During 1964 and 1965, Bledsoe, along with Helen Chan and Charles Bisson, worked on using the computer to recognize human faces (Bledsoe 1966a, & 1966b; Bledsoe, & Chan 1965). By using a semi-automated machine, later called man-machine, marks were made on photographs. These marks are used to locate major features of the faces such as mouths, noses, eyes and ears. The distances and ratios were computed by using these marks, then these are compared to reference enrollment data.

In the early 1970's Goldstein, Harmon and Lesk used 21 subjective markers such as hair color and lip thickness to create a face recognition system. (Goldstein, Harmon, & Lesk, 1971). Because of difficulties in order to automate due to subjective nature, many of the measurements were still made by hand.

A more automated approach to recognition began with Fisher and Elschlagerb just a few years after the Goldstein paper. This approach measured the features above using templates of features of different pieces of the face and them mapped them all onto a global template. After continued research it was found that these features do not contain enough unique data to represent an adult face. Another approach is the Connectionist approach, which seeks to classify the human face using a com-bination of both range of gestures and a set of identifying markers. This is usually implemented using 2-dimensional pattern recognition and neural net principles. Most of the time this approach requires a huge number of training faces to achieve decent accuracy; for that reason it has yet to be implemented on a large scale (Escarra, Robinson, Krueger, & Kochelek, 2004) .

The major problem of the early face recognition solutions is the most of feature measurements and face locations were computed manually. In 1980's, the first fully auotomated face recognition method is created depending on statistical approach. In 1988, Kirby and Sirovich applied principle component analysis at Brown University. This was considered a milestone in face recognition, because their approach is showed that less than one hundred values were required to accurately code a suitably aligned and normalized face image (Sirovich, & Kirby, 1987).

In 1991, Turk and Pentland discovered that the residual error coud be used to detect face in images while using the eigenfaces technique (Turk, & Pentland, 1991). This discovery was enabled to develop reliable real-time automated face recognition systems and increase significant interest on face recognition automation field. Since then, many different approaches have been published for face recognition over the years such as Neural Network, Fisher Linear Discriminant Model (FLD), Dynamic Link Architectures (DLA), Hidden Markov Models, Gabor Wavelet Transform, Elastic Bunch Graph. Some of these techniques were covered on Section 2.3.

The face recognition technology first captured the public's attention from the media reaction to a trial implementation at the January 2001 Super Bowl, which captured surveillance images and compared them to a database of digital mugshots.

This demonstration initiated much-needed analysison how to use the technology to support national needs while being considerate of public's social and privacy concerns. Today, face recognition technolgy is being used to combat passport fraud, support law enforcement, identify missing children, and minimize benefit/identify fraud (Smith, Ross, & Colbry, 2006).

Increase in the automation of face recognition provides hardware solutions such as application specific integrated circuit (ASIC) designs and field programmable gate arrays (FPGA). Using FPGA has many benefits over ASICs, because of low cost rapid prototyping and flexibility. One of the first publications implementing FPGA as a hardware is released by T. Nakano, T.Morie and A.Iwata in 2003. The face/object recognition system using coarse region segmentation and flexible template matching was presented and the resistive-fuse network circuit was implemented in an FPGA by a pixel serial approach, and coarse region segmentation of real images with 64×64 pixels at the video rate was achieved. The flexible template matching using dynamic-link architecture was performed in the PC system. Figure 1.3 shows this implementation (Nakano, Morie, & Iwata, 2003).

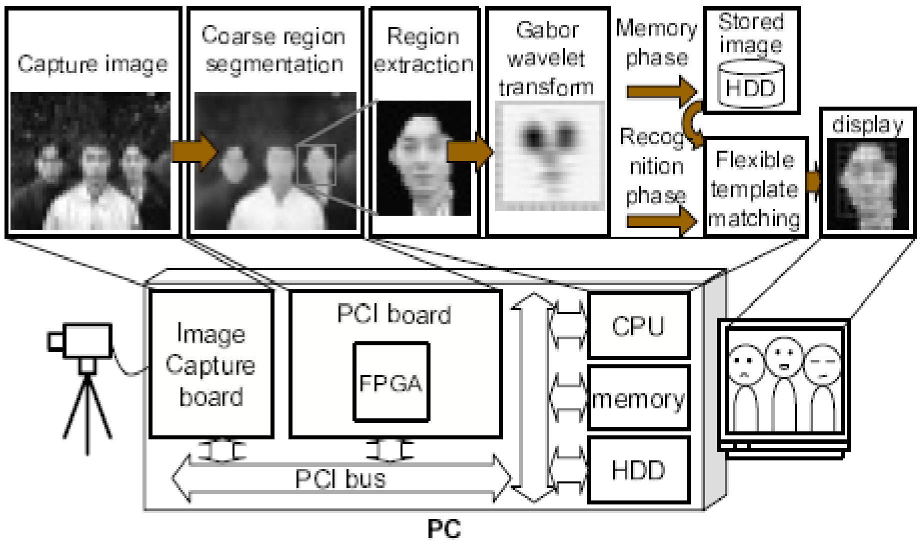


Figure 1.3 The face/object recognition system (Nakano, Morie, & Iwata, 2003).

One of the latest research by I. Sajid, M. M. Ahmed, I. Taj, M. Humayun, & F. Hameed in 2008, presents a fixed point tecnique with software hardware co-design (SHcoD) due to the floatingpoint operations based on eigenvalue algorithms are complex in terms of hardware.

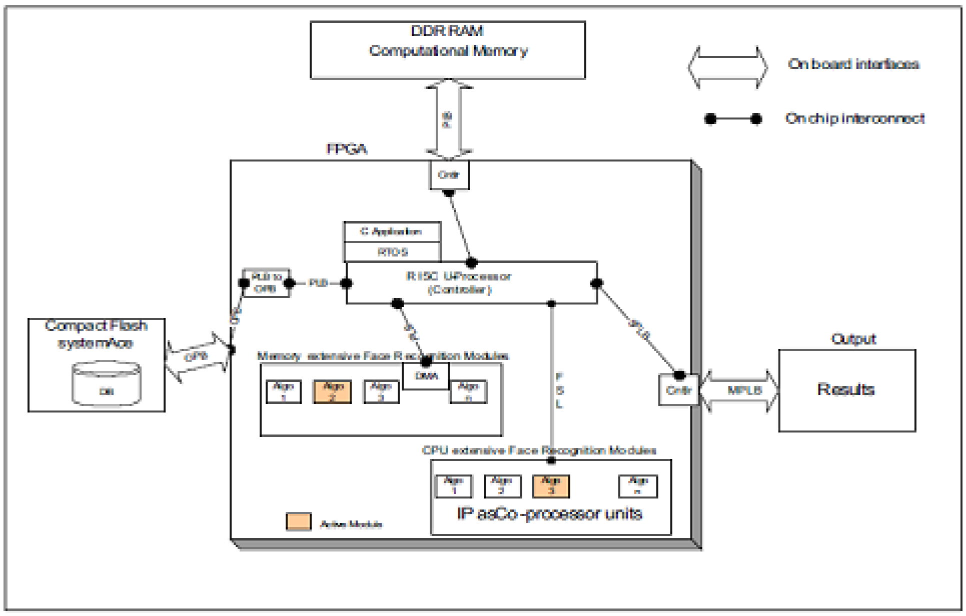


Figure 1.4 Fpga-based system architecture (Sajid, Ahmed, Taj, Humayun, & Hameed, 2008).

They have also stated that fixed point implementation of householder (HH) algorithm saves thousands of machine cycles in the cost of losing 0.008 percent weight in highest three Eigen value. The system architecture can be seen in Figure 1.4 (Sajid, Ahmed, Taj, Humayun, & Hameed, 2008).

1.4 History of Fingerprinting

Human fingerprints have been discovered on a large number of archaeological and historical items. These findings provide evidence to show that ancient people were aware of the individuality of fingerprints, such awareness does not appear to have any scientific basis (Lee, & Gaensslen, 2001). The modern scientific fingerprint technique was first initiated in the start of sixteenth century. In 1684, Nehemiah

Grew, published the first scientific paper reporting his systematic study on the ridge, furrow, and pore structure in fingerprints (Lee, & Gaensslen, 2001).

Since then, a large number of researchers interested in fingerprint studies. In 1788, a detailed description of the anatomical formations of fingerprints was made by Mayer (Moenssens, 1971). Thomas Bewick began to use his fingerprint as his trademark in 1809. This is believed to be one of the most important milestones in the scientific study of fingerprint recognition (Moenssens, 1971). In 1823, Purkinje proposed the first fingerprint classification scheme, that classified fingerprints into nine categories according to the ridge structures (Moenssens, 1971). In 1880, Henry Fauld, first scientifically suggested the individuality of fingerprints based on an empirical observation and Herschel asserted that he had practiced fingerprint recognition for about 20 years (Lee, & Gaensslen, 2001 and Moenssens, 1971). In the late nineteenth century, Sir Francis Galton conducted an extensive study on fingerprints (Galton, 1892). In 1888, Galton introduced the minutiae features for fingerprint matching. Important advance in fingerprint recognition was made in 1899 by Edward Henry. Henry established the well-known “Henry system” of fingerprint classification (Lee, & Gaensslen, 2001).

In the early twentieth century, fingerprint recognition was formally accepted as a valid personal identification method and became a standard routine in forensics (Lee, & Gaensslen, 2001). Fingerprint identification agencies were set up worldwide and criminal fingerprint databases were established (Lee, & Gaensslen, 2001). Various fingerprint recognition techniques such as fingerprint acquisition, fingerprint classification, and fingerprint matching were developed. For example, the FBI fingerprint identification division was set up in 1924 with a database of 810,000 fingerprints (Federal Bureau of Investigation, 1984).

Starting in the early 1960s, the FBI, Home Office in the UK, and Paris Police Department began to invest a large amount of effort in developing automatic fingerprint identification systems (Lee, & Gaensslen, 2001). Based on the observations of how human fingerprint experts perform fingerprint recognition, three major problems in designing automatic fingerprint identification systems (AFISs) were identified and investigated: digital fingerprint acquisition, local ridge characteristic extraction, and ridge characteristic pattern matching and their efforts were so successful that today almost every law enforcement agency worldwide uses an AFIS (Maltoni, Maio, Jain, & Prabhakar, 2003).

Automatic fingerprint recognition technology has now rapidly grown in civilian applications and fingerprint-based biometric systems are so popular for their recognition rate.

1.5 Aim of Thesis

The aim of the thesis is to create a fingerprint and face recognition system which is established on a Field Programmable Gate Array (FPGA). Principle Component Analyis (PCA), is used for extracting features. The fingerprint and face images are transformed into PCA basis subspace that is composed from eigenvalues and eigenvectors. System development in FPGA includes embedded microprocessor design, SDRAM implementation for memory needs, CFI Flash implementation for storing PCA results and communication interface for host computer. These parts of the design are discussed to develop usability and compability of the system. Comparision methods are used to identify the user in the most accurate way.

This thesis proposes a system to acquire a face or a fingerprint image of any user and process it to understand if he/she is one of people in the training database. This project are combined with two main parts. First and second part can be called as offline-training and online-test respectively. Figure 1.5 shows these two parts briefly for face recognition.

In the offline-training part for face recognition, face photos are taken from people and stored in the host computer. Then, images are resized to increase calculation speed and combined in one database matrix in MATLAB. This database matrix are sent to FPGA via serial port using RS-232 protocol. At the end of this transmission, PCA feature extraction methods are started in FPGA to create PCA basis and project database images to face subspace. At the end of offline-training part, PCA basis matrix and projected training matrix of database images are stored in CFI Flash memory. The offline-training part for fingerprint recognition is too similar to the face recognition and the only difference is the device that used for acquiring images. Face images are taken from web-camera and fingerprints are taken via fingerprint reader.

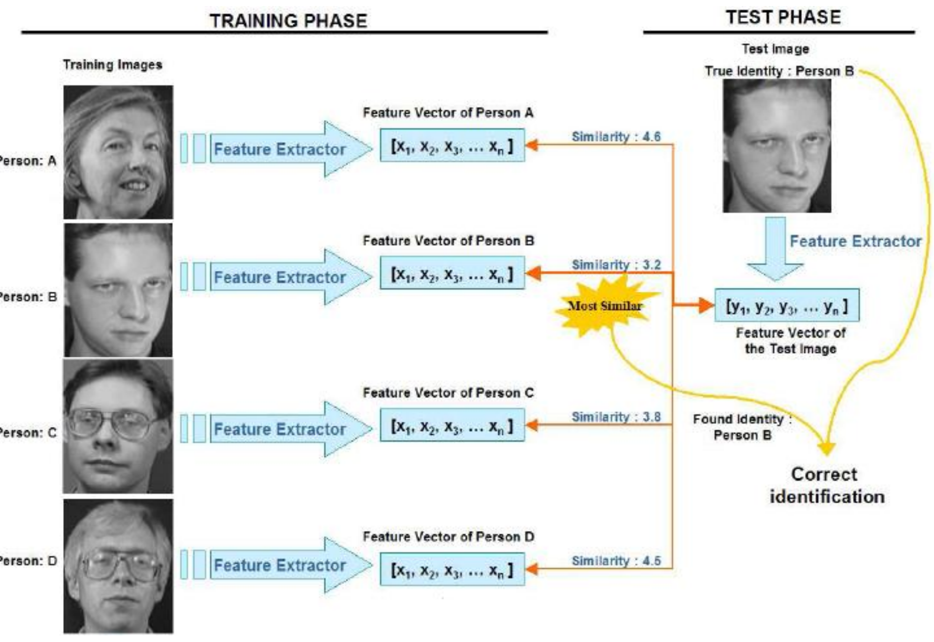


Figure 1.5 Offline-training and online test parts for face recognition.

Online-test part starts to procedure by taking a photo or a fingerprint of tester. This image is read, resized and sent to FPGA by MATLAB like in the offline-training method. After FPGA gets the test image, FPGA restores PCA basis matrix and projected training matrix of database images from CFI Flash. After projecting test image to face or fingerprint subspace by multiplying PCA basis matrix, it is compared with projected training matrix and returns result to the host computer via serial port.

After the implementation of the face and fingerprint recognition systems separately, a multibiometric recognition system, which offers more reliable recognition, is implemented by combining these two systems.

1.6 Outline of Thesis

This thesis composed of six chapters including the Introduction. Chapter 2 reviews face recognition processes and Chapter 3 summarizes fingerprint analysis and representation techniques. In Chapter 4, programmable logic devices are introduced with the devices that are used throughout project. Chapter 5 summarizes the system and explains the operation. The preliminary experiments and final results are also presented in this chapter. After completing the design of face and fingerprint recognition systems separately, they are combined together to construct a multibiometric recognition system. Chapter 6 describes this implementation. The last chapter of the thesis, Chapter 7, includes conclusions, advantages and disadvantages of the system, future works. The algorithm of whole system is in the Appendix part of the thesis.

CHAPTER TWO
FACE RECOGNITION

2.1 Face Recognition System

Face recognition systems automatically identify faces from images and videos. Two operation modes are defined for these systems: face verification and face identification, which are described briefly as follows:

a) Face Verification:

The verification task is responsible for verifying faces at the point of access. The operation of verification system is shown in Figure 2.1. The user enters his/her name or PIN (Personal Identification Number) through a keyboard or a keypad and the biometric reader the characteristic of the face to be recognized and converts it to a digital format. The digital formatted face data is processed by the feature extractor to produce a compact digital representation. The resulting representation is fed to the feature matcher to compare it against the template of a single user which is retrieved from the system database based on the user's PIN.

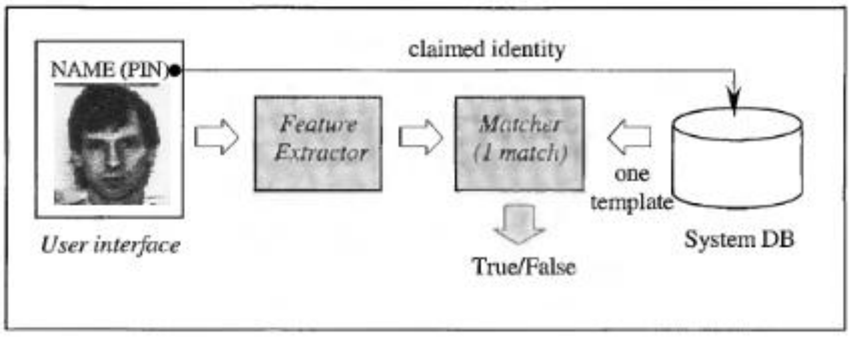


Figure 2.1 Face verification system.

b) Face Identification:

PIN isn’t provided by the user in the face identification. This task is to compare the representation of the input faces against the templates of all the users in the system database. This system identifies of an enrolled user or producing an alert message such as “user not identified”.

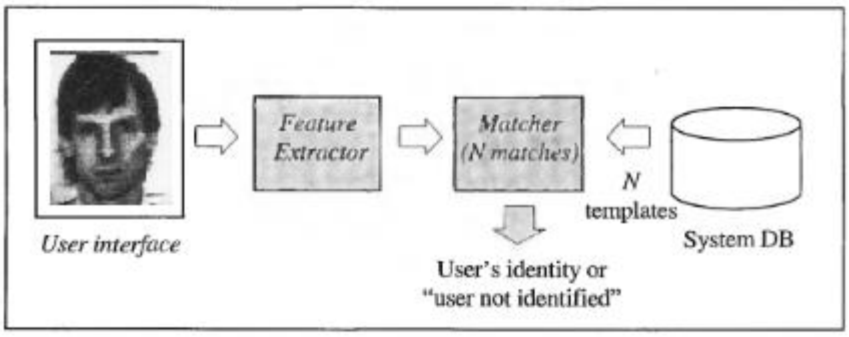


Figure 2.2 Face identification system.

2.2 Face Recognition Processing

Face recognition is a visual pattern recognition problem. A face is identified from two-dimensional images which are extracted from three-dimensional images. Since these real face images vary with pose, expression, illumination and so on, the problem is a challenging one. A face recognition process consists of four processes and these are shown in Figure 2.3.

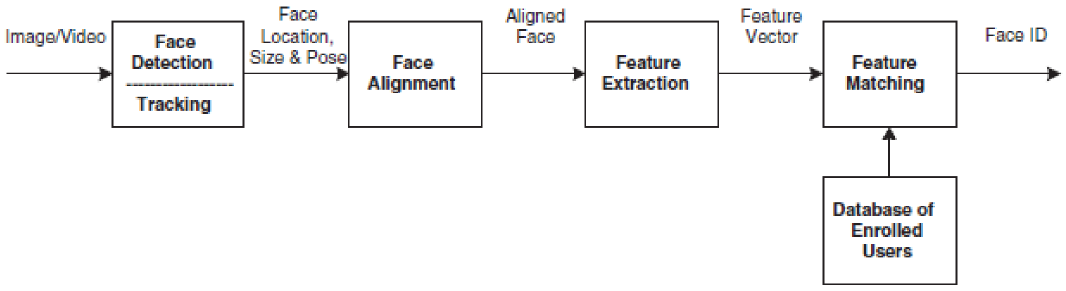


Figure 2.3 Face recognition processing flow scheme (Li, & Jain, 2004).

Face detection segments the face areas from the background. In the case of video, the detected faces may need to be tracked using a face tracking component. Face alignment is aimed at achieving more accurate localization and at normalizing faces thereby whereas face detection provides coarse estimates of the location and scale of each detected face. Facial components, such as eyes, nose, and mouth and facial outline, are located; based on the location points, the input face image is normalized with respect to geometrical properties, such as size and pose, using geometrical transforms or morphing. The face is usually further normalized with respect to photometrical properties such illumination and gray scale. After a face is normalized geometrically and photometrically, feature extraction is performed to provide effective information that is useful for distinguishing between faces of different persons and stable with respect to the geometrical and photometrical variations. For face matching, the extracted feature vector of the input face is matched against those of enrolled faces in the database; it outputs the identity of the face when a match is found with sufficient confidence or indicates an unknown face otherwise (Li, & Jain, 2004).

2.3 Face Recognition Techniques

This section try to describe the basic feature extraction and face recognition techniques such as principal component analysis (PCA), independent component analysis (ICA), linear discriminant analysis (LDA), Elastic Bunch Graph Matching (EBGM) and neural networks with mathematical theories.

2.3.1 Principal Component Analysis (PCA)

PCA algorithm is common feature extraction technique which is used for face recognition. PCA is also used in this thesis, thus this technique is described in detail. First section is an overview of PCA, second section shows the mathematical background and the last section describes the usage of PCA in face recognition field.

2.3.1.1 Overview of PCA

PCA is a standard linear algebra technique and pioneered by Kirby and Sirovich in 1988. This technique is commonly referred to as the use of eigenfaces in face recognition. To use this technique, database and test images must be at the same size and must first be normalized to line up the eyes and mouth of the subjects within the images. After normalization, PCA is used to reduce the dimension of the data by means of data compression basics. This operation reveals the most effective low dimensional structure of the facial patterns. The reduction in dimensions removes the unuseful information and decomposes the face into orthogonal (or uncorrelated) components, which are also known as eigenfaces.

Each face image may be represented as a weighted sum of the eigenfaces and these eigenfaces are stored in a 1D array. This 1D array also known as a feature vector in PCA literature. When test image is compared to database image, this feature vector is used to measure the distance. The PCA approach typically requires the full frontal face to be presented each time; otherwise the image results in poor performance (Bolme, Beveridge, Teixeira, & Draper, 2003). PCA technique can reduce the data needed to identify the individual to 1/1000th of the data presented.

Figure 2.4 shows an example of eigenfaces (MIT Media Laboratory, 2002). Feature vectors are derived using eigenfaces.

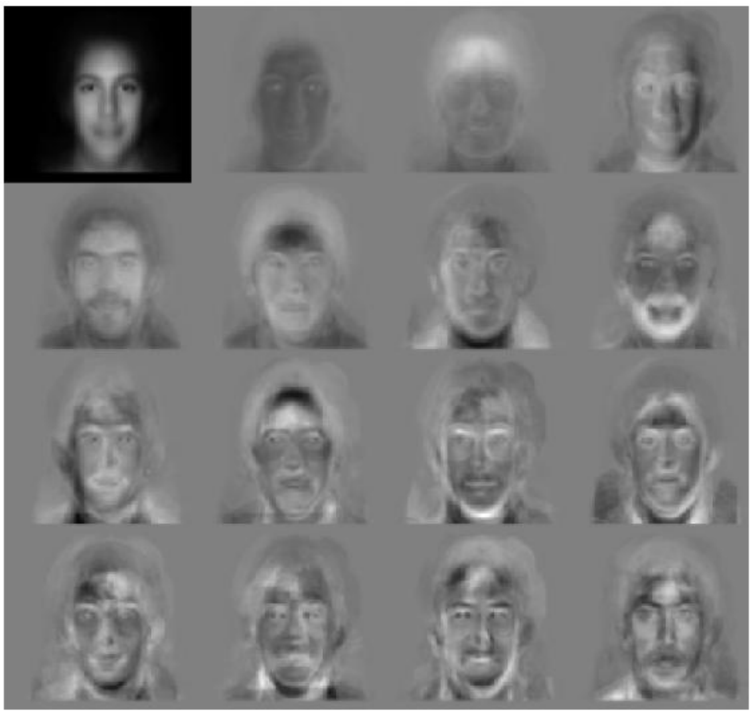


Figure 2.4 An example of eigenfaces.

2.3.1.2 Theory of PCA

Let the training set of M face images be I_1, I_2, I_3, ... , I_M. The average of the training set is, μ,

$$\mu = \frac{1}{M}\sum_{n=1}^{M} I_n \tag{2-1}$$

The difference of each image from the average is defined as;

$$\theta_i = I_i - \mu \tag{2-2}$$

This set of very large vectors is then subject to PCA, which seeks a set of M orthonormal vectors, u_n, which are describing the distribution of whole data. The kth vector of this vector,

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M} (u_k^T \theta_n)^2 \tag{2-3}$$

is a maximum subject to

$$u_l^T u_k = \zeta_{lk} = \begin{cases} 1, & \text{if } l = k \\ 0, & \text{otherwise} \end{cases} \tag{2-4}$$

The vectors u_k are eigenvectors and the scalars λ_k are eigenvalues of the covariance matrix which is shown in the following,

$$\begin{aligned} C &= \frac{1}{M}\sum_{n=1}^{M} \theta_n \theta_n^T \\ &= AA^T \end{aligned} \tag{2-5}$$

where C is the covariance matrix and $A = [\theta_1, \theta_2, \ldots, \theta_M]$.

The matrix C, is N^2 by N^2, and determining the N^2 eigenvectors and eigenvalues is an intractable task for typical image sizes, so a computationally feasible method to find these eigenvectors must be implemented. If the number of data points in the image space is less than the dimension of the space ($M < N^2$), there is only $M - 1$, rather than N^2 meaningful eigenvectors (Turk and Pentland, 1991). By using this approach the eigenvectors v_i of A^TA is,

$$A^T A v_i = \beta_i v_i \tag{2-6}$$

multipliying both sides by A,

$$AA^T A v_i = \beta_i A v_i \tag{2-7}$$

Eq. (2-7) shows that Av_i are the eigenvectors of $C = AA^T$. By using this analysis, M x M matrix, $L = A^TA$ is constructed. The L is,

$$L_{mn} = \theta_m^T \theta_n \tag{2-8}$$

and shows the M eigenvectors, v_l, of L.These vectors are used to determine the linear combinations of the M training set face images to form the eigenfaces u_l.

$$u_l = \sum_{k=1}^{M} v_{lk}\theta_k, \qquad l = 1, 2, ..., M \tag{2-9}$$

With this analysis the calculations are greatly reduced, from the order of the number of pixels in the images (N^2) to order of the number of images in the training set (M) and in practice, the training set of face images will be relatively small and the calculations become quite managable (Turk and Pentland, 1991).

2.3.1.3 How to use PCA in Face Recognition

To create a face space from *M* number of the face images, first *L* matrix (*M x M*) must be calculated. This *L* matrix has *M* eigenvectors. M_1 significant eigenvectors are chosen from this *L* matrix which are containing the highest associated eigenvalues. Then, by combining the normalized training images according to Eq. (2-9) to produce the eigenfaces u_k.

For the test step, first the new face image (I_T) is projected into facespace by a simple operation,

$$\omega_k = u_k(I_T - \mu) \qquad \text{for } k = 1, 2, ..., M_1 \tag{2-10}$$

and ω is the weights and these weights form the pattern vector, φ^T,

$$\varphi^T = [\omega_1, \omega_2, ..., \omega_{M_1}] \tag{2-11}$$

The pattern vector describes the contribution of each eigenface in representing the input face image. After generating pattern vector, the simplest method for determining which face class provides the best description of an input face image is to find the face class k which minimizes the Euclidean distance, ε,

$$\varepsilon_k = \left\| (\varphi - \varphi_k \right\|^2 \tag{2-12}$$

where φ_k is a vector describing the *k*th face class and the face classes φ_i are calculated by averaging the results of the eigenfaces over a small number of face images of each individual. The minimum ε_k, if provides the recognition condition under a pre-determined threshold value, *k*th person is determined the output of recognition system.

2.3.2 Linear Discriminant Analysis (LDA)

LDA is a statistical approach for classifying samples of unknown classes based on the training samples with known classes (Bolme, Beveridge, Teixeira, & Draper, 2003). LDA is the technique which aims to maximize variance across the users or formerly named between-classes, and minimize variance within the users which is also expressed within-class formerly.

In the Figure 2.5, an example of six classes using LDA is shown (Lu, Plataniotis, & Venetsanopoulos, 2003). In this figure, each block represents a class. There are large variances between-classes, but the variance within-classes is very little. When dealing with high dimensinal face data, this technique faces the sample size problem that arises where there are a small number of avaliable training samples compared to the dimensionality of the sample space (Lu, Plataniotis, & Venetsanopoulos, 2003).

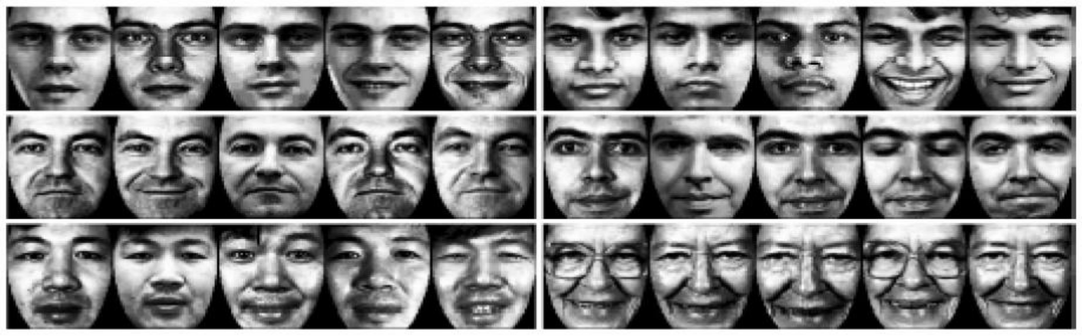


Figure 2.5 An example of six classes using LDA.

2.3.2.1 Theory of LDA

As mentioned above, all instances of the same person’s face as being in one class and the faces of different subjects as being in different class for all subjects in the training must be defined before computing LDA. LDA is a class specific method that represents data set make it useful for classification. Given a set of *N* imgaes $\{x_1, x_2, \ldots, x_n\}$ where each image belongs to one of *c* classses $\{X_1, X_2, \ldots, X_c\}$. LDA selects a linear tranformation matrix *W* that is the ratio of the between-class scatter and the with-in class scatter is maximized.

S_B is the between-class scatter matrix which represents the scatter of the conditional mean vectors, μ_i's; around the overall mean vector, μ. S_B can be expressed by the following formula;

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T \tag{2-13}$$

where μ_i denotes the mean of image class X_i, μ denotes the mean of entire data set, N_i denotes the number of images in class X_i.

S_W is the within-class scatter matrix which represents the average scatter of the sample vectors x of different class C_i around their respective mean μ_i;

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T \tag{2-14}$$

If the within-class scatter matrix S_W is not singular, LDA finds an orthonormal matrix W_{opt} which maximizes the ratio of the determinant of the between-class scatter matrix to the determinant of the within-class scatter matrix. This matrix can be expressed by the following formula;

$$W_{opt} = \arg\max \frac{\left|W^T S_B W\right|}{\left|W^T S_W W\right|} = \left[w_1 \; w_2 \; \ldots \; w_m\right] \tag{2-15}$$

The set of solution $\{w_i \mid i = 1, 2, \ldots, m\}$ is that of generalized eigenvectors of S_B and S_W corresponding to the m largest eigenvalues $\{\lambda_i \mid \mathrm{i} = 1, 2, \ldots, m\}$, which can be shown that as in following;

$$S_B w_i = \lambda_i S_W w_i \quad \text{where } i = 1, 2, \ldots, m \tag{2-16}$$

In face recognition applications, generally S_W is singular, so to overcome this singularity, PCA algorithm is first used to reduce the vector dimensions. Combining PCA and LDA, first input image *x* projected into face space *y*, then projected into classification space *z*;

$$\begin{aligned} y &= \theta^T x \quad \text{(only PCA)} \\ z &= W_x^T x \quad \text{(only LDA)} \\ z &= W_y^T y \quad \text{(PCA + LDA)} \end{aligned} \tag{2-17}$$

2.3.3 Independent Component Analysis (ICA)

ICA is another algorithm for face recognition. To better understand the concept, it is useful to compare ICA with PCA. PCA depends on the pairwise relationships between pixels, but ICA depends on the higher order relationships among pixels in the image database. So that, PCA can only represent second order interpixel relationships, or relationships that capture the amplitude spectrum of an imgage but not its phase spectrum. On the other hand, ICA use high order relationships between the pixels and ICA algorithms are capable of capturing the phase spectrum (Bartlett, Movellan, & Sejnowski, 2002).

In the ICA implementation of face recognition, input face image represented as an n-dimensional random vector. Then, PCA is used to reduce this random vector, without losing the higher order statistical components. After that, covariance matrix of the result is calculated and factorized form of covariance matrix is obtained. Whitening, rotation and normalization are performed respectively to obtain the face space of the individuals. Because of using high order relationships between pixels, ICA is robust in the presence of noise.

2.3.3.1 Theory of ICA

ICA of a random vector searches for a linear transformation which minimizes the statistical dependence between its components (Comon, 1994). Let, the image is represented by a random vector, $X \in \mathrm{R}^{N}$, where N is the dimensionality of the image space. The vector is formed by concatenating the rows or the coloumns of the image which may be normalized to have a unit norm and/or an equalized histogram (Liu, & Wechsler, 1999). The covariance matrix of X can be expressed by using expectation operator, *E(.)*, as in the following;

$$C_X = E\{[X - E(X)][X - E(X)]^T\} \qquad (2\text{-}18)$$

where $C_X \in \mathrm{R}^{NxN}$. The ICA of X factorizes the covariance matrix into the following expression;

$$C_X = F \Delta F^T \qquad (2\text{-}19)$$

where Δ is diagonal real positive and F transforms the original data set X to new data set Z which are independent or the most independent possible data set. Z can be expressed as;

$$X = FZ \qquad (2\text{-}20)$$

To find the transformation F, Comon developed an algorithm that consists of three operations: whitening, rotation and normalization (Comon, 1994). The whitening operation transforms a random vector X to U which has a unit covariance matrix and U can be expressed by the following formula;

$$X = \varphi A^{1/2} U \qquad (2\text{-}21)$$

where φ and A are derived by solving the following eigenvalue operation;

$$C_X = \varphi A \varphi^T \tag{2-22}$$

where $\varphi = [\varphi_1, \varphi_2, \ldots, \varphi_N]$ is an orthonormal eigenvector matix and $A = diag\ \{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ is a diagonal eigenvalue matrix of C_X. After whitening operation, rotation operations performs source separation by minimizing the mutual information approximated using high order cumulants to derive independent components. Finally, the normalization operation derives unique independent components in terms of orientation, unit norm, and order of projections (Comon, 1994).

2.3.4 Elastic Bunch Graph Matching (EBGM)

This algorithm relies on the concept of the non-linear characteristics of the real face images, such as pose, expression and variations in illumination. Because, these non-linear characteristics are not addressed by the linear analysis methods, such as PCA and LDA. An example of elastic bunch graph matching is shown in Figure 2.6 (Wiskott, 1996).

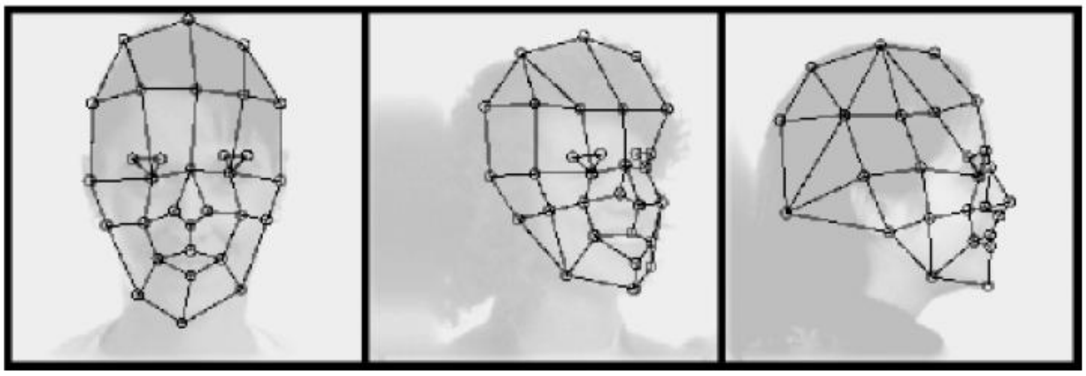


Figure 2.6 Elastic bunch graph matching (EBGM).

Gabor Wavelet Transform is used to create a dynamic link architecture that projects the face image onto an elastic grid. The nodes on the elastic grid that are notated by the circles in the previous figure, are formerly called as gabor jets. Gabor jets describe the image behaviour around a given pixel. This is the result of a convolution of the image with Gabor filter. Gabor filter is used to extract features and detect shapes. Recognition is based on comparing Gabor filter response on each Gabor node.

The difficulty with his method is the requirement of accurate landmark localization, which can sometimes be achieved by combining PCA and LDA methods (Bolme, Beveridge, Teixeira, & Draper, 2003). As mentioned above, EBGM based on Gabor Wavelet Transform (GWT), so in the next section the thery of GWT is described.

2.3.4.1 Theory of GWT

Dennis Gabor proposed Gabor functions as a tools for signal detection under noise effect. Gabor showed that the conjonit time-frequency domain for 1D signals must be quantized so that no signal or filter can occupy less than certain minimal area in it (D. Gabor, 1946). Gabor also discovered that Gaussian modulated complex exponentials provide the best trade off between frequency and time resolution. Gabor functions are generalized and reorganized to 2D by Daugman, to use in computer vision applications which is expressed below (Daugman, 1980);

$$G_i(\vec{x}) = \frac{\left\|\vec{k}_i\right\|^2}{\sigma^2} e^{-\frac{\left\|\vec{k}_i\right\|^2 \left\|\vec{x}\right\|^2}{2\sigma^2}} \left[e^{j\vec{k}_i\vec{x}} - e^{-\frac{\sigma^2}{2}} \right] \tag{2-23}$$

where *Gi* is a plane wave characterized by the vector *ki* enveloped by a Gaussian function and σ is the standard deviation of this Gaussian envelope. The center frequency of the *i*th filter is given by the characteristic wave vector which have a scale k_v and orientation θ_μ,

$$k_i = \begin{pmatrix} k_{ix} \\ k_{iy} \end{pmatrix} = \begin{pmatrix} k_v \cos\theta_\mu \\ k_v \sin\theta_\mu \end{pmatrix} \tag{2-24}$$

Daugman proposed that an ensemble of simple cells is best modeled as a family of 2D Gabor wavelets sampling the frequency domain in a log-polar manner (Daugman, 1980). This is equivalent to coherent states generated by rotation and dilation. The decomposition of an input image *I* into these states is called the wavelet transform and expressed as;

$$R_i(\vec{x}) = \int I(\vec{x}')G_i(\vec{x}-\vec{x}')d\vec{x}' \qquad (2\text{-}25)$$

Combining Eq. (2-24) and Eq. (2-25), Gabor wavelets are used first by determining wave vector scale k_v and orientation θ_μ. Kepenekci show that Gabor filters with spatial frequency (v = 0, ..., 4) and 8 orientation (μ = 0, ..., 8) in Figure 2.7 and convolving the input image (Figure 2.8a) with Gabor filters (Figure 2.8b) captures the whole frequency spectrum (Kepenekci, 2001).

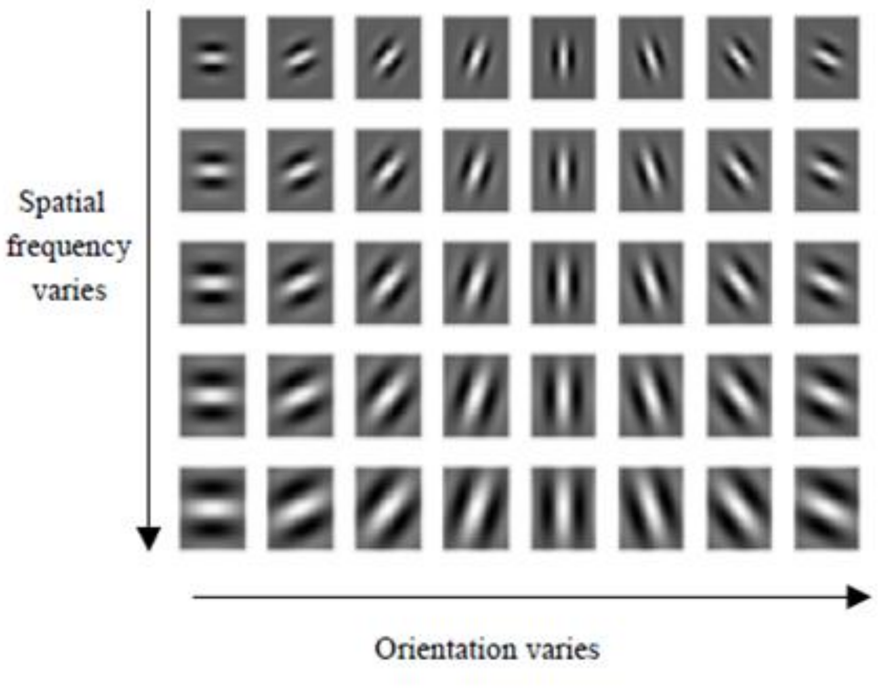


Figure 2.7 Gabor filters correspond to 5 spatial frequency and 8 orientation.

From the responses of the face image to Gabor filters, peaks are found by searching the locations (Figure 2.8c) by using windowing methods to find eyes, nose and mouth in the face (Kepenekci, 2001).

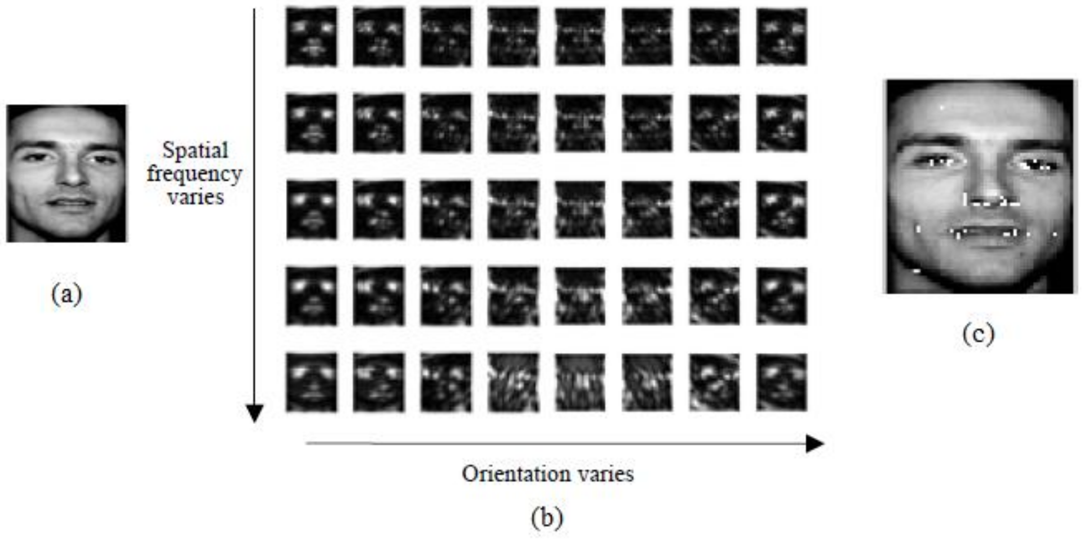


Figure 2. 8 (a) An example face image from Stirling database (b) Filter responses (c) High energized points of Gabor wavelet responses

2.3.5 Neural Networks

Most of the face recognition systems use smart algorithms to recognize the faces from the extracted features such as eigenfaces. One of the common technique is the artificial neural networks. This algorithm is biogogically inspired and based on the functionality of neurons. The equivalent of neurons in neural network are perceptrons. Neurons sum the strengths of all electric inputs. Similarly, perceptrons generates a weighted sum on their numerical inputs. A neural network is formed for each person in the face database by using these perceptrons.

The neural networks usually consist of three or more layers (Li, & Areibi, 2004). First, database images are dimensionally reduced by using PCA. The input layer of neural network takes these reduced images. The output layer of a neural network produces a numerical value between -1 and 1. In between of these two layers, there exist one or more hidden layers which are depend on the application. When using neural network for face recognition, using one hidden layer provides a good balance between accuracy and complexitiy. Increasing the number of hidden layer, training time of the system exponentially increases.

When the neural network is formed for each person, first it must be trained to recognize that person. The most common training method is the back propagation algorithm (Li, & Areibi, 2004). By using this algorithm, the weights of the connections between neurons are set. The result of these connections are high output value (near to 1) belong to the person it represents and low output value (near to -1) for other people. In the recognition face, neural network system returns the highest numerical output for this person.

The biggest problem of neural networks is that, there is no clear method to find the initial network topologies. Since training takes a long time, experimenting with such topologies becomes a difficult task (Li, & Areibi, 2004). Another main issue occurs when neural networks are tried to use online training, time consuming task and the difficulty of adding a new person to database is not well suited for real-time applications.

CHAPTER THREE
FINGERPRINT ANALYSIS AND REPRESENTATION

3.1 Introduction

A fingerprint is the reproduction of a fingertip epidermis and is produced when a finger is pressed against a smooth surface. The most evident structural characteristic of a fingerprint is a pattern of interleaved ridges and valleys; in a fingerprint image (Figure 3.1), ridges (also called ridge lines) are dark whereas valleys are bright (Maltoni, Maio, Jain, & Prabhakar, 2003). The size of the ridges vary in width from 100 µm, for very thin ridges, to 300 µm for thick ridges. Generally, the period of a ridge/valley cycle is about 500 µm (Stosz, & Alyea, 1994). Injuries such as burns or cuts do not affect the underlying ridge structure, and the original pattern is duplicated when the new skin grows.

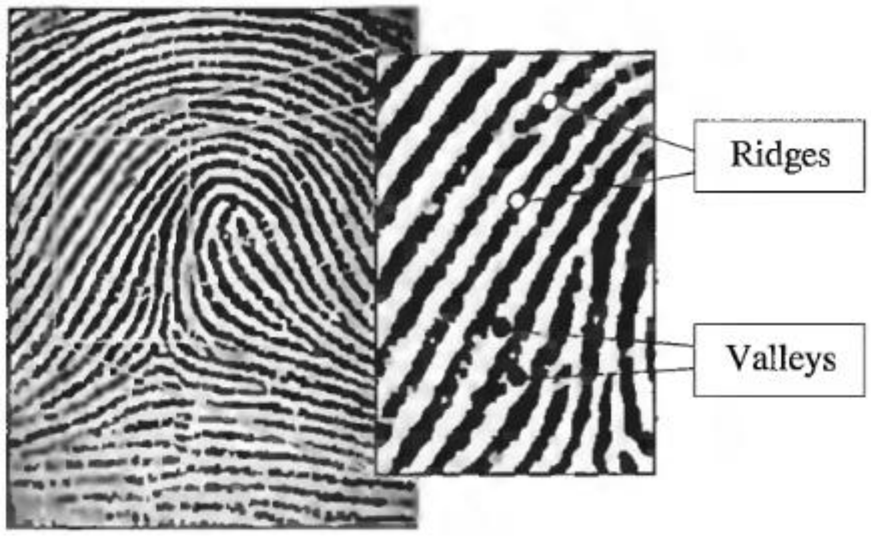


Figure 3.1 Ridges and valleys on a fingerprint image.

In fingerprint, ridges and valleys often run in parallel. Sometimes, ridges and valleys bifurcate or terminate. If fingerprint is analyzed at the global level, the fingerprint pattern exhibits one or more regions where the ridge lines assume distinctive shapes. These regions are called singularities or singular regions and they can be classified at major and local levels. When major level discussed, it can be seen that, singular regions may be classified into three typologies: loop, delta, and whorl. Singular regions of a fingerprint belonging to loop, delta, and whorl types are characterized by ∩, Δ, and O shapes, respectively. Figure 3.2 (Maltoni, Maio, Jain, & Prabhakar, 2003) shows that major singular regions. This figure also shows that the center point of the fingerprint or formerly called core.

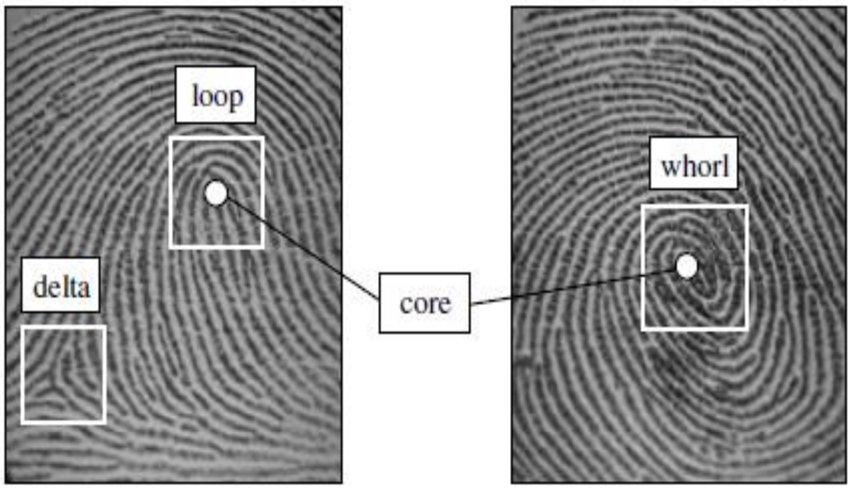


Figure 3.2 Singular regions and core points in fingerprint images.

When fingerprint patterns discussed in local level, other important features, called minutiae can be found. Minutia means small detail; in the context of fingerprints, it refers to various ways that the ridges can be discontinuous and for example, a ridge can suddenly come to an end (termination), or can divide into two ridges (bifurcation) (Maltoni, Maio, Jain, & Prabhakar, 2003). Figure 3.3.(a) shows that the most common minutia types such as termination, bifurcation, lake or crossover.

These minutiae types are commonly used for fingerprint recognition. The American National Standards Institute (ANSI) proposed a minutiae taxonomy method based on four classes. These classes are terminations, bifurcations, trifurcations (or crossovers), and undetermined. But, the FBI minutiae-coordinate model considers only terminations and bifurcations: each minutia is denoted by its class, the x- and y-coordinates and the angle between the tangent to the ridge line at the minutia position and the horizontal axis (Figure. 3.3.(b) and 3.3.(c)) (Wegstein, 1982).

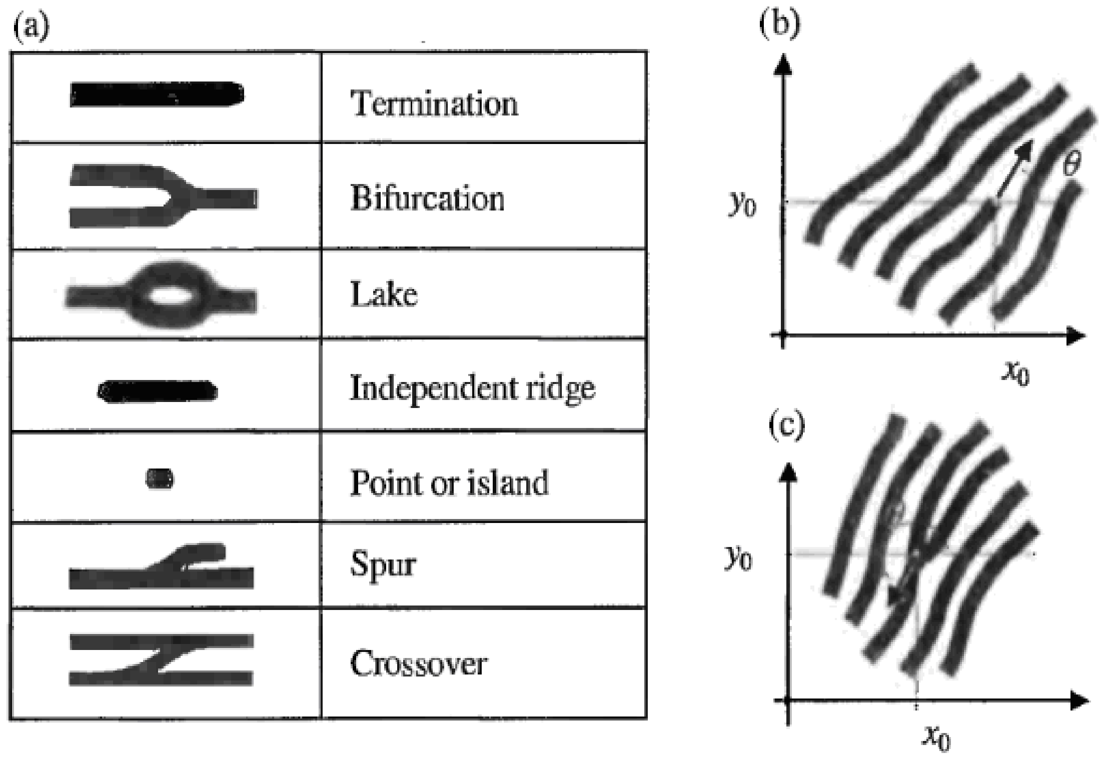


Figure 3.3 a) The most common minutiae types; b) Termination minutia : [x_0 ,y_0] are the minutia coordinates; θ is the angle that the minutia tangent forms with the horizontal axis; c) A bifurcation minutia; θ is now defined by means of the termination minutia corresponding to the original bifurcation that exists in the negative image (Maltoni, Maio, Jain, & Prabhakar, 2003).

3.2 Fingerprint Image Processing and Feature Extraction

Most of the fingerprint recognition and classification algorithms require a feature extraction stage for identifying remarkable features. The features extracted from fingerprint images often have a direct physical counterpart such as singularities or minutiae, but sometimes they are not directly related to any physical traits such as local orientation image or filter responses. These features may be used directly for matching or an intermediate step for the derivation of other features. For example, some preprocessing and enhancement steps are often performed to simplify the task of minutiae extraction (Maltoni, Maio, Jain, & Prabhakar, 2003).

A fingerprint image, *I*, is often represented as a two-dimensional surface. When *I* be a gray-scale fingerprint image with *g* gray-levels, bright pixels associated with with gray-levels close to g-1 and dark pixels associated with gray-levels close to 0.

3.3 Estimation of Local Ridge Orientation

Let [*x*, *y*] be a generic pixel in a fingerprint image. The local ridge orientation at [*x*, *y*] is the angle θ_{xy} that the fingerprint ridges, crossing through an arbitrary small neighborhood centered at [*x*, *y*], form with the horizontal axis (Maltoni, Maio, Jain, & Prabhakar, 2003).

Instead of computing local ridge orientation at each pixel, most of the fingerprint processing and feature extraction methods estimate the local ridge orientation at discrete positions such as local windows. The size of the local windows can be varied depending on the application. The fingerprint orientation image is a matrix *D* whose elements encode the local orientation of the fingerprint ridges. Figure 3.4 shows the orientation of a fingerprint image (Maltoni, Maio, Jain, & Prabhakar, 2003). Note that each element θ_{ij} shows that the orientation of each window. An additional value r_{ij} is often associated with each element θ_{ij} to denote the reliability of the orientation. The simplest and most natural approach for extracting local ridge orientation on a fingerprint image is based on computation of gradients.

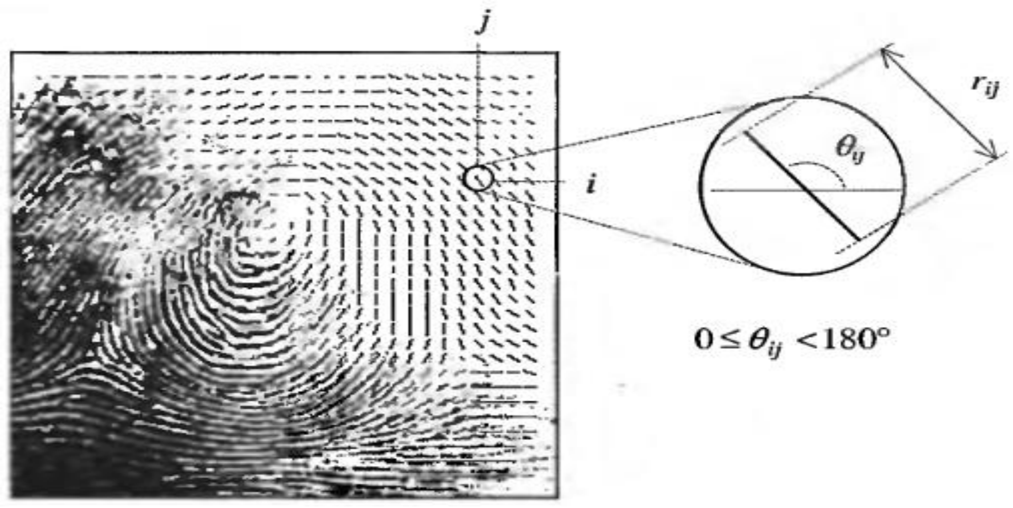


Figure 3.4 A fingerprint image faded into the corresponding orientation image computed over a 16 x 16 local windows. Each element denotes the local orientation of the fingerprint ridges, θ_{ij}; the element length is proportional to its reliability, r_{ij}.

3.4 Estimation of Local Ridge Frequency

The local ridge frequency (or density) f_{xy} at point [*x*, *y*] is the inverse of the number of ridges per unit length along a hypothetical segment centered at [*x*, *y*] and orthogonal to the local ridge orientation θ_{xy} (Maltoni, Maio, Jain, & Prabhakar, 2003). A frequency image *F* can be defined if the frequency is estimated at discrete positions and arranged into a matrix. First, 32 x 16 oriented window centered at [x_i , y_j] is defined. Then the x-signature of the gray-levels is obtained by accumulating, for each column *x*, the gray-levels of the corresponding pixels in the oriented window. This kind of averaging makes the gray-level profile more smoother and prevents ridge peaks. f_{ij} is determined as the inverse of the average distance between two consecutive peaks of the x-signature. Figure 3.5 shows the estimation of local ridge frequency.

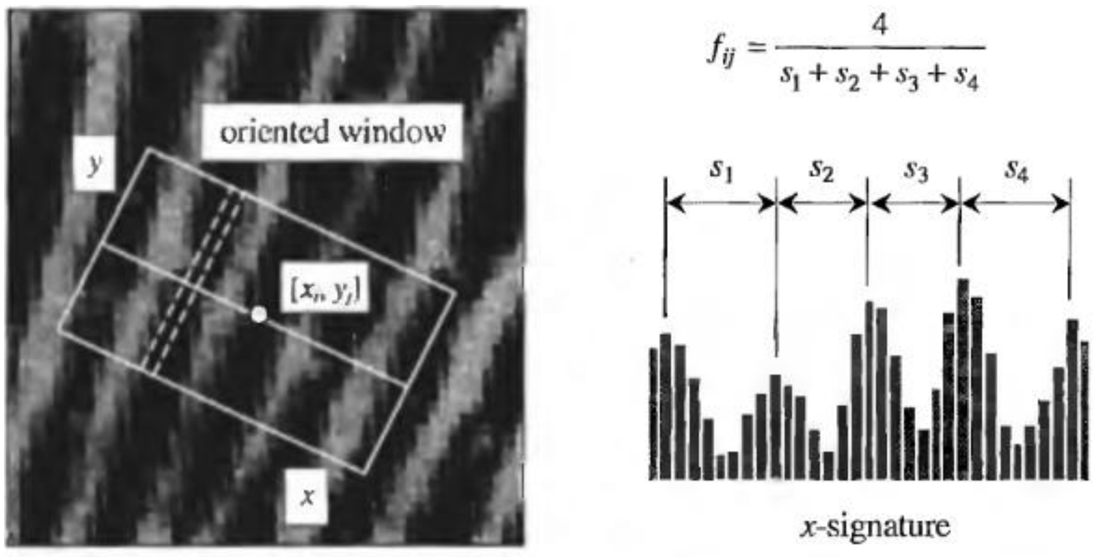


Figure 3.5 Estimation of local ridge frequency. An oriented window centered at [x_i , y_j]. The dashed lines show the pixels whose gray-levels are accumulated for a given column of the x-signature. The x-signature on the right clearly exhibits five peaks; the four distances between consecutive peaks are averaged to determine the local ridge frequency (Maltoni, Maio, Jain, & Prabhakar, 2003).

3.5 Singularity and Core Detection

Most of the approaches proposed in the literature for singularity detection operate on the orientation of the fingerprint image. Poincarè index method is the most common method used for detecting singularities and core on a fingerprint pattern. This method is summarized in the following section.

3.5.1 Poincarè Index Method

Define *G* is a vector field and *C* be a curve in *G*; then the Poincarè index $P_{G,C}$ is defined as the total rotation of the vectors of *G* along curve *C* (Figure 3.6).

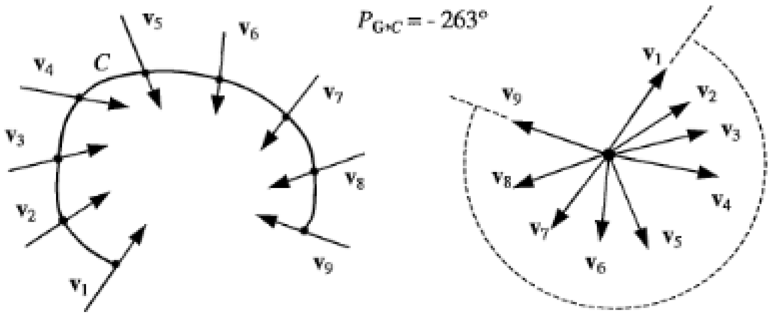


Figure 3.6 The Poincarè index computed over a curve *C* immersed in vector field *G* (Maltoni, Maio, Jain, & Prabhakar, 2003).

Let *G* be the field associated with a fingerprint orientation image *D* and let $[i, j]$ be the position of the element θ_{ij} in the orientation image; then the Poincarè index $P_{G,C}(i, j)$ at $[i, j]$ is computed by first taking the curve *C* is a closed path defined as an ordered sequence of some elements of *D*. Usually the element $[i, j]$ of D is internal point. $P_{G,C}(i, j)$ is computed by algebraically summing the orientation differences between adjacent elements of curve *C*. Summing orientation differences require a direction to be associated at each orientation. For solving this problem, the direction of the first element is randomly selected and the direction of the other elements is found by assigning the closest direction to that of the previous element to each successive element. On closed curves, the Poincarè index assumes only one of the discrete values: 0°, ±180°, and ±360°. Singularities on a fingerprint image are defined in Eq. (2-1).

$$P_{G,C} = \begin{cases} 0^\circ \text{ , if } [i, j] \text{ does not belong to any singular region} \\ 360^\circ \text{ , if } [i, j] \text{ belongs to a whorl type singular region} \\ 180^\circ \text{ , if } [i, j] \text{ belongs to a loop type singular region} \\ -180^\circ \text{ , if } [i, j] \text{ belongs to a delta type singular region} \end{cases}. \qquad (3\text{-}1)$$

In 3 x 3 windowing, the path defining curve *C* is the ordered sequence of the eight elements d_k (*k* = 0, ..., 7) surrounding the internal point [*i*, *j*]. The direction of the elements d_k is chosen as follows: d_0 is directed upward; d_k (*k* = 0, ..., 7) is directed so that the absolute value of the angle between d_k and d_{k-1} is less than or equal to 90° (Maltoni, Maio, Jain, & Prabhakar, 2003). The computation of Poincarè index method is in Eq. (2-2) and an example of this method is shown in Figure 3.7 (Maltoni, Maio, Jain, & Prabhakar, 2003).

$$P_{G,C}(i, j) = \sum_{k=0,\ldots,7} angle\left(d_k, d_{(k+1)\bmod 8}\right). \qquad (3\text{-}2)$$

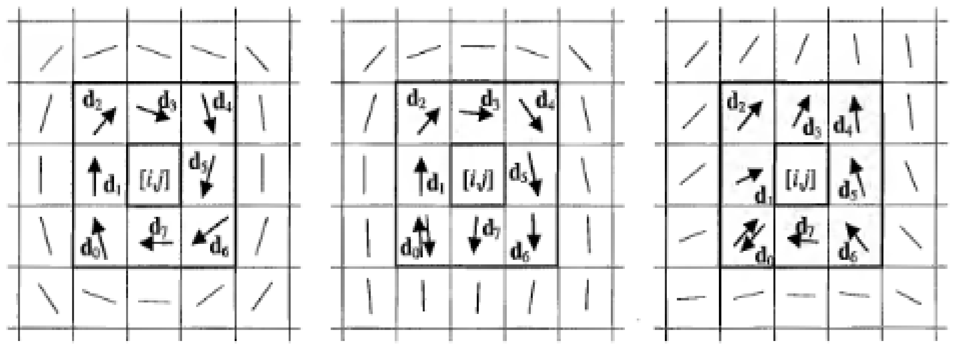


Figure 3.7 Example of computation of the Poincare index in the 8-neighborhood of points belonging (from the left to the right) to a whorl, loop, and delta singularity, respectively.

3.6 Normalization

In an ideal fingerprint image, ridges and valleys alternate and flow in a locally constant direction but in practice the input images must be enhanced before minutiae extraction to increase the performance of fingerprint recognition techniques. Normalization is one of the most commonly used enhancement method for determining the new intensity value of each pixel in an fingerprint image as;

$$I'[x,y] = \left\{ \begin{array}{ll} m_0 + \sqrt{(I[x,y]-m)^2 \cdot v_0 / v}\,, & \text{if } I[x,y] > m \\ m_0 - \sqrt{(I[x,y]-m)^2 \cdot v_0 / v}\,, & \text{otherwise} \end{array} \right\}. \tag{3-3}$$

where *m* and *v* are the image mean and variance and m_0 and v_0 are the desired mean and variance after the normalization process.

Normalization technique is a pixel-wise operation and does not change the ridge and valley structures. Figure 3.8 shows an example of normalization process (Maltoni, Maio, Jain, & Prabhakar, 2003). Input image is normalized with desired mean and variance values.

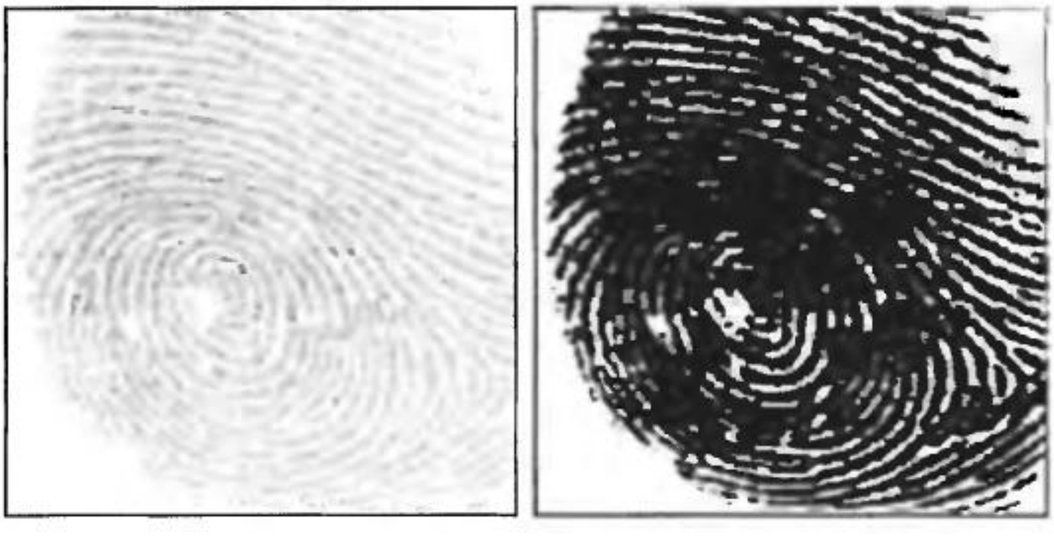


Figure 3.8 An example of normalization with values of m_0 =100 and v_0 =100.

3.7 Minutiae Detection

Most of the automatic fingerprint identification systems used minutiae matching for fingerprint comparison so, reliable minutiae extraction is an extremely important task and a lot of research has been devoted on this topic. Most of the proposed methods for minutiae detection require the fingerprint gray-scale image to be converted into a binary image. Some of the binarization processes are dilation, erosion, opening, closing, thinning and thicking. These processes are greatly benefit from an a priori enhancement. The binary images obtained by the binarization process are usually submitted to a thinning stage which allows for the ridge line thickness to be reduced to one pixel and finally, a simple image scan allows the detection of pixels that correspond to minutiae (Figure 3.9) (Maltoni, Maio, Jain, & Prabhakar, 2003).

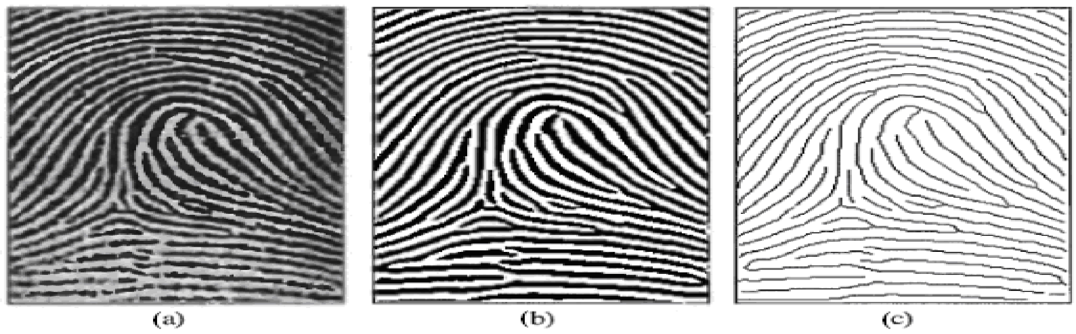


Figure 3.9 a) A fingerprint gray-scale image; b) The image obtained after a binarization of the image in (a); c) The image obtained after a thinning of the image in (b).

Once a binary skeleton of a fingerprint image has been obtained, a simple image scan allows the pixel corresponding to minutiae to be detected. In fact the pixels corresponding to minutiae are characterized by a crossing number and the crossing number *cn*(*p*) of a pixel *p* in a binary image is defined as half the sum of the differences between pairs of adjacent pixels in the 8-neighborhood of *p*;

$$cn(p) = \frac{1}{2} \sum_{i=1,..8} | val(p_{i,\mathrm{mod}8}) - val(p_{i-1}) |, \qquad (3\text{-}4)$$

where p_0, p_1,..., p_7 are the pixels belonging to an ordered sequence of pixels defining the 8- neighborhood of *p* and *val*(*p*) $\in \{0,1\}$ is the pixel value (Maltoni, Maio, Jain, & Prabhakar, 2003). Figure 3.10 shows and defines a pixel *p* with *val*(*p*) = 1 according to crossing number for 3x3 window;

• is an intermediate ridge point if *cn*(*p*) = 2;

• corresponds to a termination minutia if *cn*(*p*) = 1;

• defines a more complex minutia (bifurcation, crossover, etc.) if *cn*(*p*) $\geq$ 3 (Maltoni, Maio, Jain, & Prabhakar, 2003).

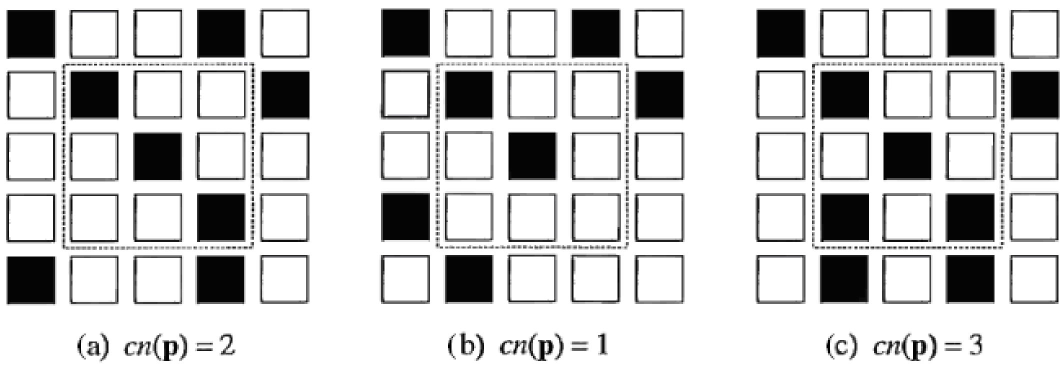


Figure 3.10 a) Intra-ridge pixel, b) Termination minutia, c) Bifurcation minutia.

Some authors have proposed that minutiae extraction approaches that work directly on the gray-scale images without binarization and thinning because a significant amount of information may be lost these processes and these processes are time consuming rather than using gray-scale image. Image quality also affects the performance of binarization processes.

3.8 Estimation of Ridge Count

Orientation, frequency, absolute position, and type of minutiae are not the only features that can be used for fingerprint recognition. The latest studies show that using ridge count is increasing the reliability of analysis.

Ridge count is a measurement of the distances between any two points in the fingerprint image. Let *a* and *b* be two points in a fingerprint; then the ridge count between point *a* and point *b* is the number of ridges intersected by segment *ab* (Figure 3.11) (Maltoni, Maio, Jain, & Prabhakar, 2003).

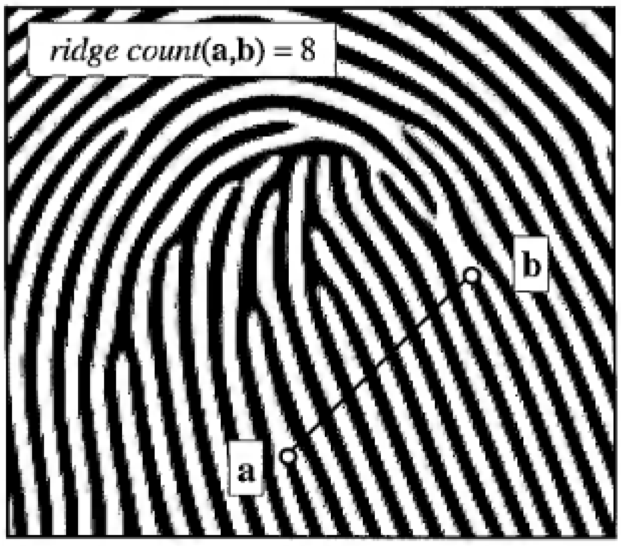


Figure 3.11 In this example the number of ridges intersected by segment *ab* (ridge count between *a* and *b*) is 8.

3.9 Fingerprint Matching

A fingerprint matching algorithm compares two given fingerprints and returns either a degree of similarity or a binary decision such as mated or non-mated like in the recognition system. The large number of approaches to fingerprint matching can be classified into three families:

1) Correlation-based matching: Two fingerprint images are correlated and the correlation between corresponding pixels is computed for different alignments. In this thesis one of the most popular correlation-based matching, principal component analysis (PCA), is used.

2) Minutiae-based matching: This is the most popular and widely used technique for fingerprint comparison. This technique is also being the basis of the fingerprint analysis. First, minutiae are extracted from the two fingerprints and stored as sets of points. Then, matching algorithm is used for finding the alignment between the template and the input minutiae sets. Final result is estimated in the maximum number of minutiae pairings.

3) Ridge feature-based matching: Minutiae-based extraction is difficult in very low-quality fingerprint images. However, whereas other features of the fingerprint ridge pattern (e.g., local orientation and frequency, ridge shape, texture information) may be extracted more reliably than minutiae, their distinctiveness is generally lower (Maltoni, Maio, Jain, & Prabhakar, 2003). The approaches belonging to ridge feature-based matching compare fingerprints in term of features extracted from the ridge pattern.

CHAPTER FOUR
PROGRAMMABLE LOGIC DEVICES

This chapter details the brief history of programmable logic devices from simple architectures to modern complex architectures. Interconnect types of programmable logic devices and configuration techniques are also discussed. Also two Altera FPGA based development kits: UP3 Education Kit and DE2-70 Development Kit used for this project has been introduced.

4.1 History of Programmable Logic

By the late of 1970s, printed circuit boards are loaded with standard logic devices. Then to offer the ultimate in design flexibility, Ron Cline from Signetics came up with the idea of two programmable planes. These planes are provided any combination of “AND” and “OR” gates, as well as sharing of AND terms across multiple ORs (Xilinx, 2006).

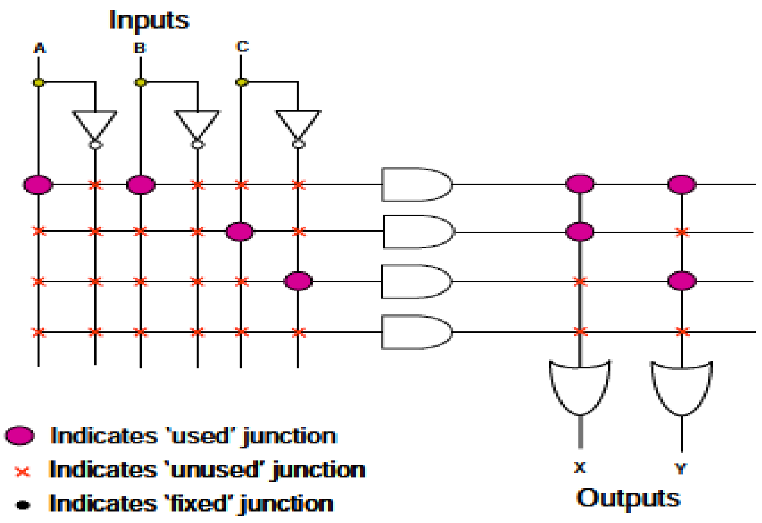

Inputs
A
B
C
Indicates 'used' junction
Indicates 'unused' junction
Indicates 'fixed' junction
X
Y
Outputs

Figure 4.1 Simple PLA architecture.

MMI brings a new technology by fixing one of the programmable planes to decrease the propagation delay time through the device and complexity of the software. One programmable AND and Fixed OR arrays are called as Programmable

Array Logic (PAL) or Simple Programmable Logic Device (Simple PLD). Figure 4.2 shows the architecture of PAL.

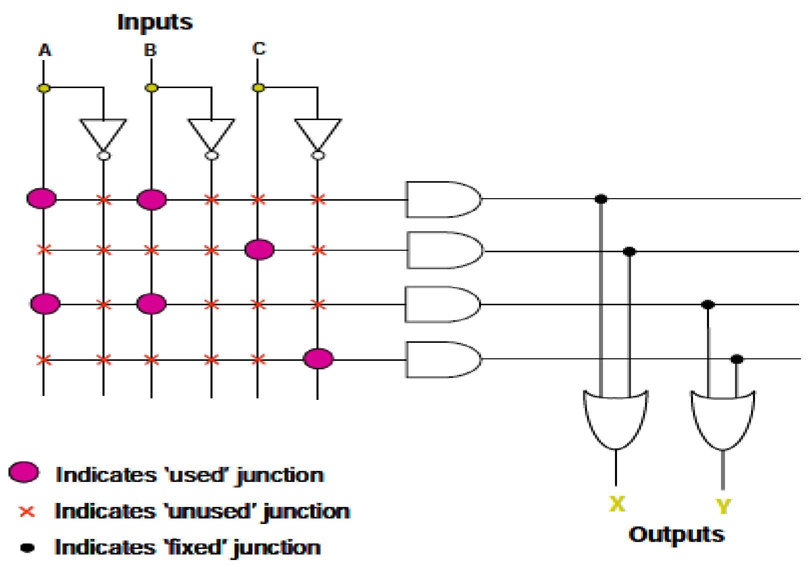

Inputs
A
B
C
Indicates 'used' junction
Indicates 'unused' junction
Indicates 'fixed' junction
X
Y
Outputs

Figure 4.2 PAL or simple PLD.

With the improvement in PLA and PAL devices, new type of PLD devices are introduced which extend the density of SPLDs. These devices are called as Complex Programmable Logic Device (CPLD). CPLD's brings major advantages to industry such as;

- Ease of design,
- Lower Development Costs,
- Reduced board area,
- Higher speeds,
- On-system programming.

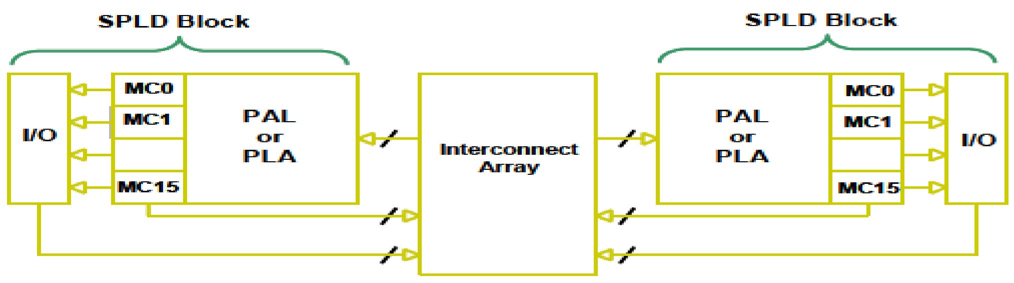


Figure 4.3 CPLD architecture.

In 1985, Xilinx introduced Field Programmable Gate Array (FPGA) which is an alternative type of PLD. FPGA architecture and device configuration methods are discussed in the following section.

4.2 FPGA Architecture

There are two types of FPGAs: SRAM-based programmable FPGA and One time programmable FPGA. The most commonly used design is SRAM-based design. The advantage of this design is reprogramming ability. But, SRAM-based FPGA needs reprogramming everytime when it’s powered up. So, most of the designs use a serial PROM for storing programming data.

FPGA architecture consists of an array of logic blocks, routing channels and I/O communication interconnects. Depending on the vendor these logic blocks called as Configure Logic Block (CLB) for Xilinx and Logic Array Block (LAB) for Altera. In our thesis, we used Altera based development kits that use Cyclone FPGAs, so Altera LAB architecture is described in the next section.

4.2.1 Logic Element (LE)

The smallest unit of logic in the Cyclone II architecture is the Logic Element (LE). LE provides advanced features with efficient logic utilization.
Each LE features:

- A four-input look-up table (LUT), which is a function generator that
can implement any function of four variables,
- A programmable register
- A register chain and a carry chain connection
- The ability to drive all types of interconnects: local, row, column,
register chain, and direct link interconnects
- Support for register feedback
- Support for register packing (Cyclone II Handbook, Altera Corp., 2007).

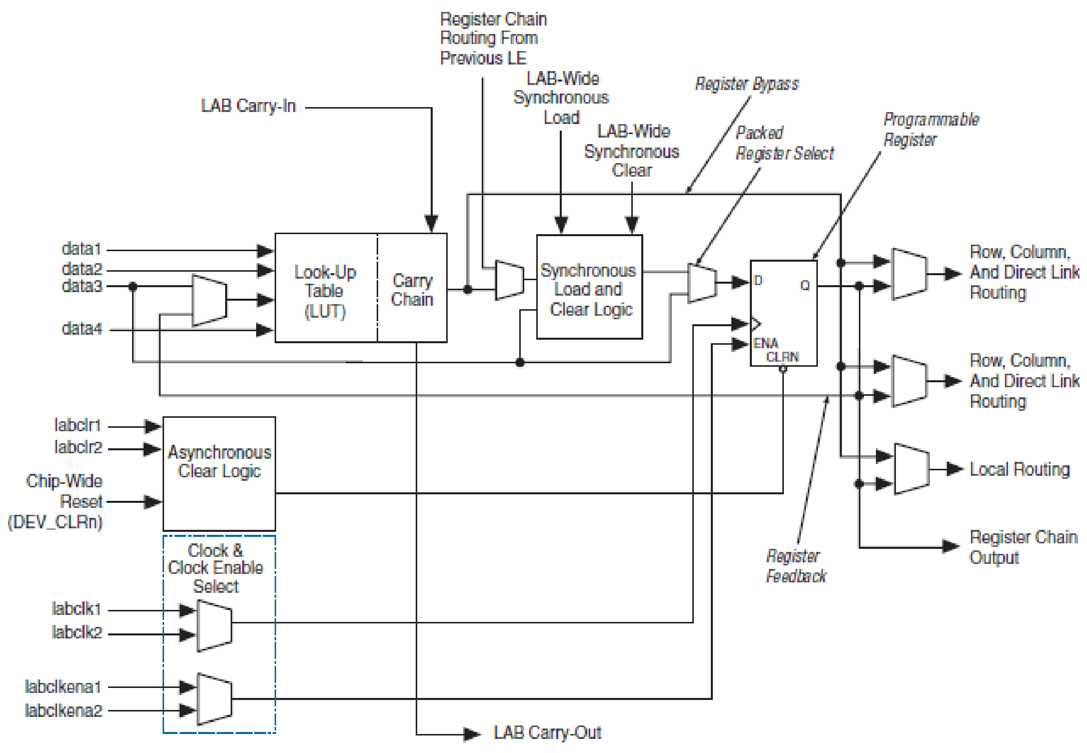


Figure 4.4 Cyclone II logic element.

LE operates in two modes by using different resources depending on the application. First mode is normal mode and this mode is suitable for combinational functions and common logic applications. The second mode is arithmetic mode and this mode provides adders, accumulators, counters and comparators.

4.2.2 Logic Array Block (LAB)

Each LAB consists of the following:

- 16 LEs
- LAB control signals
- LE carry chains
- Register chains
- Local interconnect

The local interconnect transfers signals between LEs in the same LAB. Register chain connections transfer the output of one LE's register to the adjacent LE's register within an LAB (Cyclone II Handbook, Altera Corp., 2007). Figure 4.5 shows Cyclone II LAB architecture.

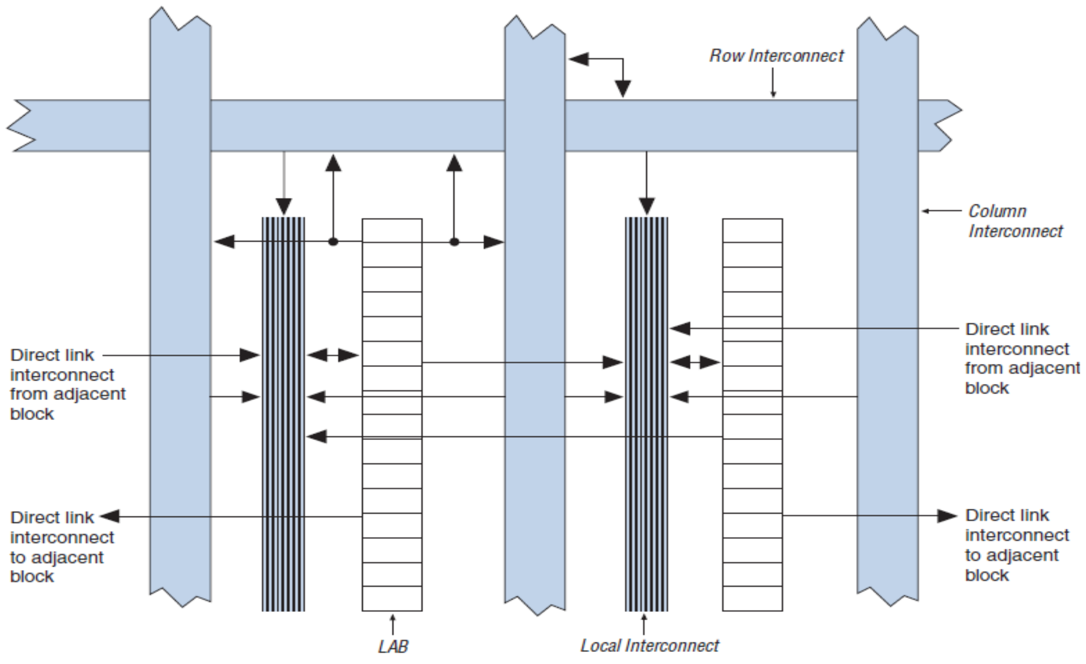

Row Interconnect
Column Interconnect
Direct link interconnect from adjacent block
Direct link interconnect from adjacent block
Direct link interconnect to adjacent block
Direct link interconnect to adjacent block
LAB
Local Interconnect

Figure 4.5 Cyclone II LAB architecture.

Wide variety of connections can be made by investigating Figure 4.5. However, this flexible routing increase the logic complexity.

4.3 FPGA Configuration

FPGAs can be configured in several ways such as schematic design entry, using hardware description languages (HDLs) and using high-level languages. These methods are described in the following sections according to their abstraction level from lowest level schematic design entry to highest level high-level language compilers.

4.3.1 Schematic Design Entry

Schematic design entry is the lowest level of FPGA configuration. Schematic design includes standard logic gates, multiplexers, I/O buffers, storage elements and macros for device specific functions such as adders or plls. The macros can be constructed from primitive logic elements to further use in large circuit designs.

Schematic design entry is the least popular method of describing hardware, because when the complexity of the circuit increases, it is difficult to follow connection nodes in the schematic.

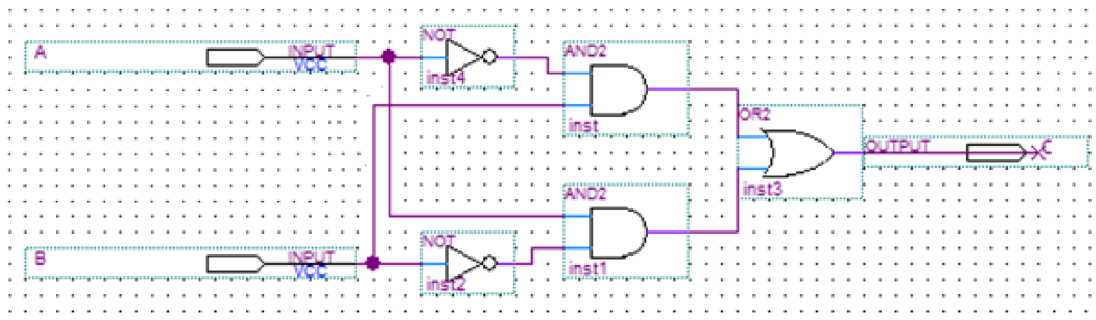


Figure 4.6 An example by using schematic design entry.

4.3.2 Hardware Description Languages

Hardware Description Languages (HDLs) are text-based depictions of the behaviour of the digital circuit. The differences of HDLs from software programming languages are the ways of describing the propagation of time and signal dependencies. The most popular HDLs are VHDL and Verilog HDL. These languages are similar but use different notations.

VHDL stands for VHSIC Hardware description language where VHSIC stands for very high speed integrated circuit. VHDL was originally develop by the US Department of Defense and released in 1985.

Verilog HDL development started in Gateway Design Automation Inc. in 1985. Cadence Design Systems purchase Gateway Design Automation in 1990. With this purchase, Verilog is started to use in public and very popular in industry from this date.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.ALL;

entity halfadder is
   port (in_A           : in  std_logic;
         in_B           : in  std_logic;
         sum            : out std_logic; -- sum out from A+B
         carry          : out std_logic  -- carry out from A+B
        );
end halfadder;

architecture rtl of halfadder is

begin

   sum <= (in_A XOR in_B);
   carry <=  in_A AND in_B;

end rtl;
```

Figure 4.7 Half adder implementation by using VHDL.

```
module halfadder(in_A,in_B,sum,carry);

input in_A;
input in_B;
output sum;
output carry;

   assign sum = in_A ^ in_B;
   assign carry = in_A & in_B;

endmodule
```

Figure 4.8 Half adder implementation by using Verilog HDL.

4.3.3 High-Level Languages

Using high-level programming languages for FPGA design is the increasing interest in the industry. The custom language such as C or phyton is compiled to generate a Verilog HDL or VHDL circuit description. SystemC, Celoxia's DK Design suite and MyHDL are an example of high-level languages.

Half adder implementation by using VHDL, Verilog HDL and SystemC is shown in Figure 4.7, Figure 4.8 and Figure 4.9 respectively.

```
#include "systemc.h"
SC_MODULE(half_adder) {
  sc_in<bool>a, b;
  sc_out<bool>sum, carry;
  void proc_half_adder();
  SC_CTOR(half_adder) {
    SC_METHOD (proc_half_adder);
    sensitive << a << b;
  }
};

void half_adder::proc_half_adder() {
  sum = a ^ b;
  carry = a & b;
}
```

Figure 4.9 Half adder implementation by using SystemC.

4.4 FPGA Development Kits

There are several development boards to explore the capabilities of FPGAs and to develop prototypes. In this section, two FPGA development kits used in this project have been demonstrated.

4.4.1 UP3 Education Kit

UP3 Education Kit is produced by System Level Solutions (SLS). The following informations include general features, UP3 board top view and UP3 development kit photo is taken from UP3 Education Reference Manual (Version 0.1, SLS 2004).

The general features of UP3 Education Kit are listed below;

- Features an Altera EP1C6Q240 Device and EPCS1 configuration devices
- Supports intellectual property based (IP-Based) design both with and without a microprocessor.
- USB 1.1 (Full speed & Low speed)
- RS 232 Port (Full Modem) and Parallel Port (IEEE1284)

- PS/2 Port
- VGA Port
- IDE (Integrated Drive Electronics)
- 128 KBytes of SRAM (64K x 16)
- 2 MBytes of FLASH (1M x 16)
- Supports multiple clocks like PCI clock, USB clock, IOAPIC clock and CPU clock
- JTAG and Active Serial download capability
- 5V Santa Cruz Long Expansion Card Header provides 72 I/O for the development of additional board providing various functionalities
- One user definable 4-bit switch block
- Four user definable push button switches, and one global reset switch
- Four user definable LEDs
- One 16 x 2 character display LCD Module
- I2C Real Time Clock.

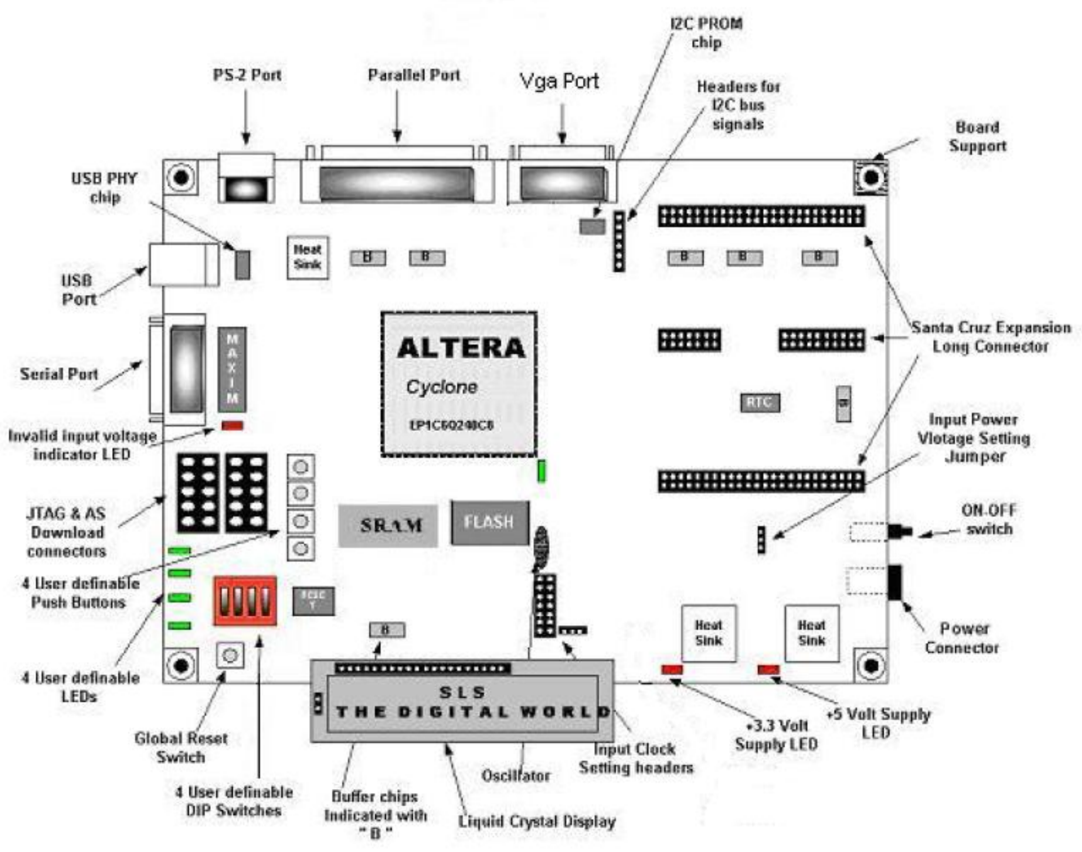


Figure 4.10 UP3 board top view.

The Device Features of Cyclone EP1C6Q240 FPGA:

- 5980 Logic Elements (LEs)
- 20 RAM Blocks
- 92160 Total RAM Bits
- 2 PLLs
- 185 Maximum number of I/Os

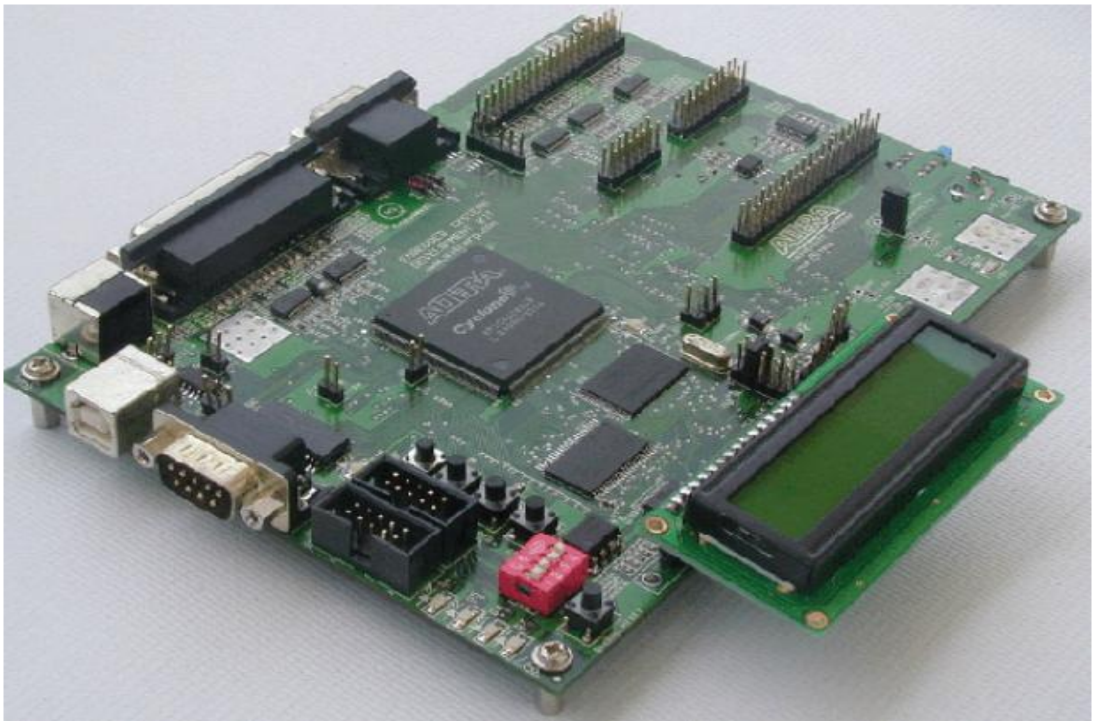


Figure 4.11 Photo of UP3 board.

4.4.2 DE2-70 Development Kit

The DE2-70 board is produced by Terasic Technologies. The general features of this device and a board photo is taken from Altera DE2-70 Development and Education Board User Manual (Version 1.08, Terasic Technologies 2009).

The following hardware is provided on the DE2-70 board:

- Altera Cyclone® II 2C70 FPGA device
- Altera Serial Configuration device - EPCS16

- USB Blaster (on board) for programming and user API control; both JTAG and Active Serial (AS) programming modes are supported
- 2-Mbyte SSRAM
- Two 32-Mbyte SDRAM
- 8-Mbyte Flash memory
- SD Card socket
- 4 pushbutton switches
- 18 toggle switches
- 18 red user LEDs
- 9 green user LEDs
- 50-MHz oscillator and 28.63-MHz oscillator for clock sources
- 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks
- VGA DAC (10-bit high-speed triple DACs) with VGA-out connector
- 2 TV Decoder (NTSC/PAL/SECAM) and TV-in connector
- 10/100 Ethernet Controller with a connector
- USB Host/Slave Controller with USB type A and type B connectors
- RS-232 transceiver and 9-pin connector
- PS/2 mouse/keyboard connector
- IrDA transceiver
- 1 SMA connector
- Two 40-pin Expansion Headers with diode protection

The Device Features of Cyclone II 2C70 FPGA:

- 68,416 Logic Elements
- 250 M4K RAM Block
- 1,152,000 total RAM bits
- 150 embedded multipliers
- 4 PLLs
- 622 user I/O pins
- FineLine BGA 896-pin package

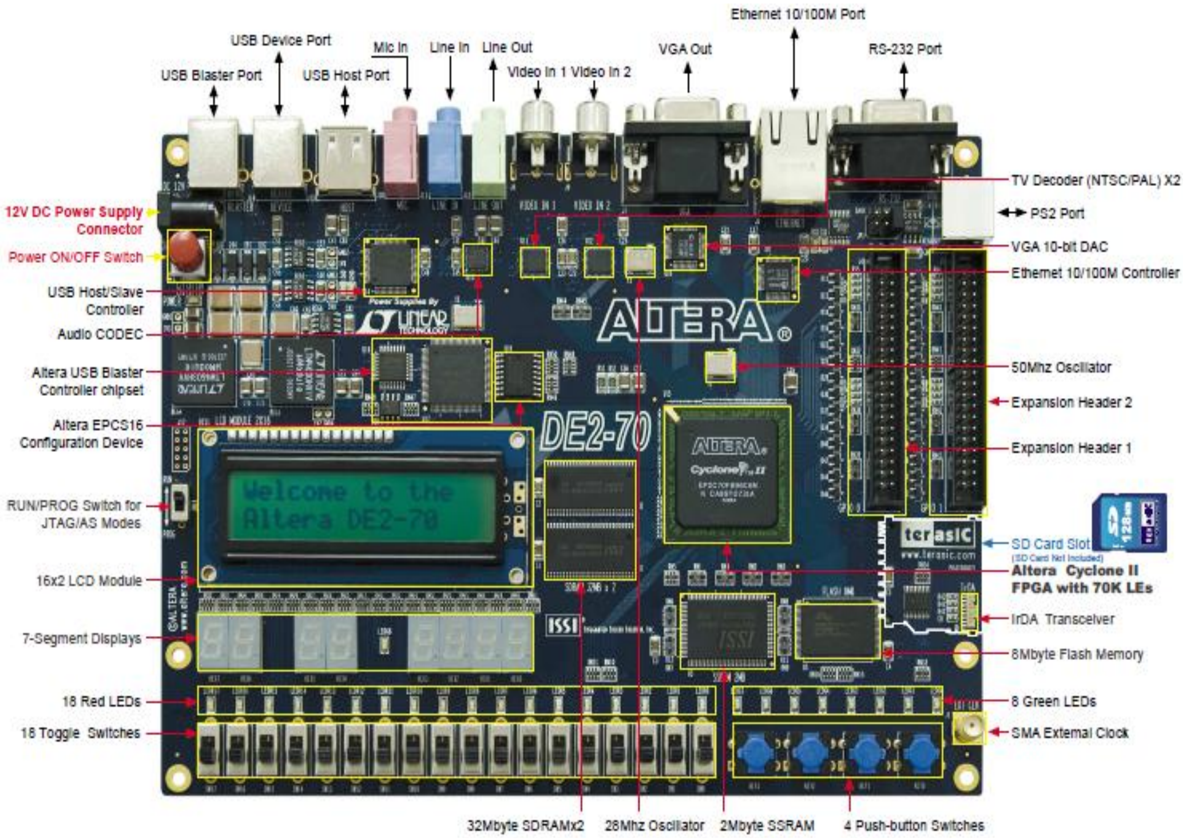


Figure 4.12 DE2-70 board top view.

CHAPTER FIVE
FPGA-BASED FACE AND FINGERPRINT RECOGNITION SYSTEM DESIGN

This chapter describes the implementation of FPGA-based face and fingerprint recognition system of this thesis. As mentioned in previous chapters two FPGA development kits are used to realize this recognition system. The purpose of using two development kits and implementation on these boards are also covered in this chapter. In the following sections of this chapter, the first project which are implemented to familiarize with FPGAs are also included to the help the FPGA beginners.

5.1 Face and Fingerprint Recognition System Design on UP3 Development Kit

5.1.1 Quartus II Software

The first study on UP3 development kit is to learn the usage of design tools. Analysis and synthesis tool, Quartus II software, is a powerful programmable logic design software that is released from Altera. The most important Quartus II features are;

- Implementation of VHDL and Verilog for hardware description
- Visual edition of logic circuits using schematic entity design
- Vector waveform simulation.

Quartus II software is a comprehensive environment for system-on-a-programmable-chip (SOPC) design and includes solutions for all phases of FPGA and CPLD design. Quartus II design flow is illustrated in Figure 5.1 which is provided by Altera. Altera’s Quartus II software allows us to use Quartus II graphical user interface, EDA tool interface or command-line interface for each phase of the design flow. Quartus II handbook describes the all features of Quartus II graphical user interface for each stage of the design flow. This is shown in Figure 5.2.

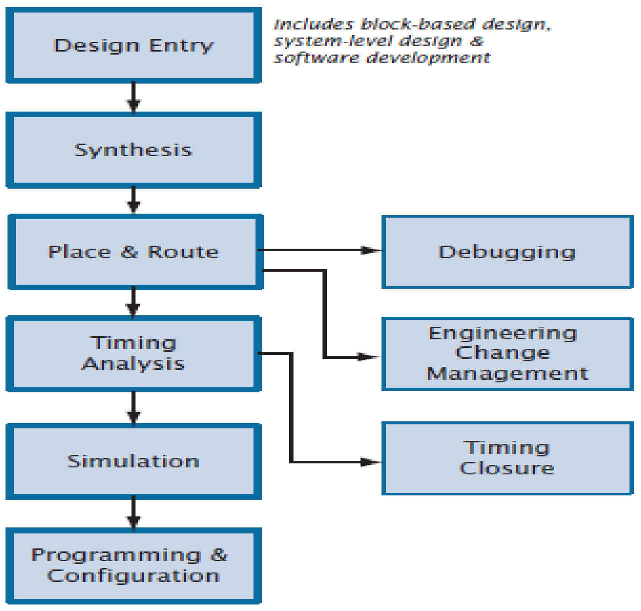

Design Entry
Includes block-based design, system-level design & software development
Synthesis
Place & Route
Debugging
Timing Analysis
Engineering Change Management
Simulation
Timing Closure
Programming & Configuration

Figure 5.1 Quartus II design flow.

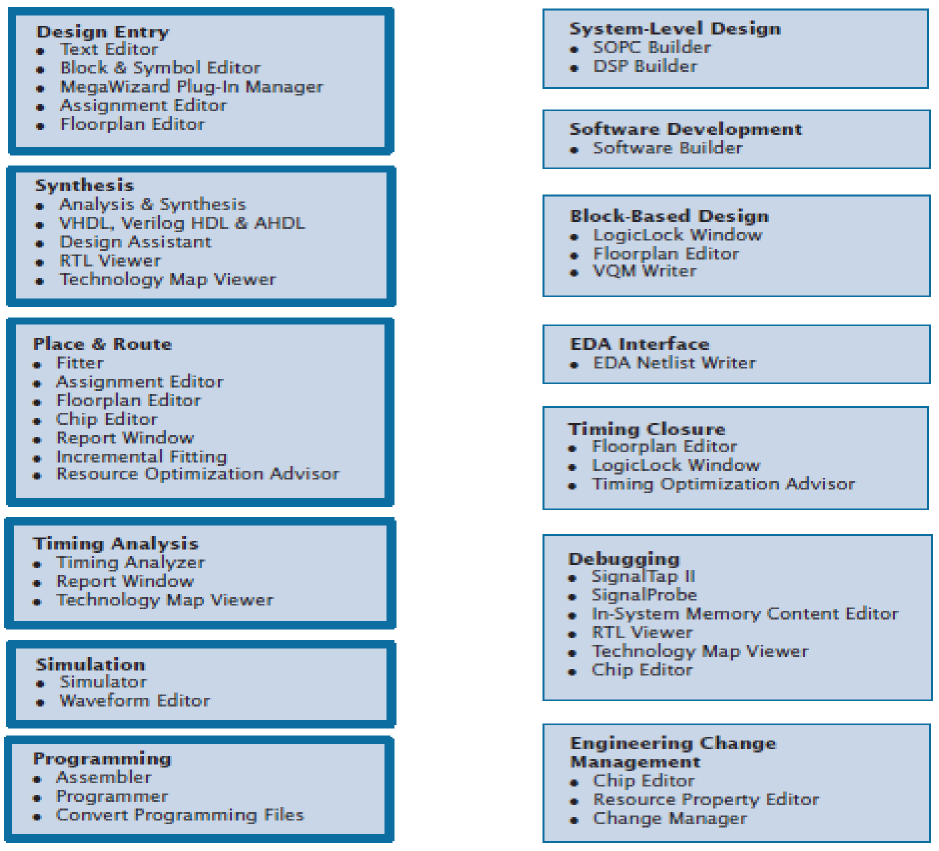


Figure 5.2 Quartus II graphical user interface features.

5.1.2 Preliminary Study on UP3 Development Kit

After learning the basic features of Quartus II such as how to compile a project and how to program the board from Quartus II handbook and some examples, second step is to design combinational circuits for getting used to the development board.

First idea is to configure FPGA with different types of combinational circuits, which have one or more inputs and similarly one or more outputs. Figure 5.3 shows and summarizes first study on UP3 development kit. In these examples DIP switches are used as inputs and LEDs are used for outputs.

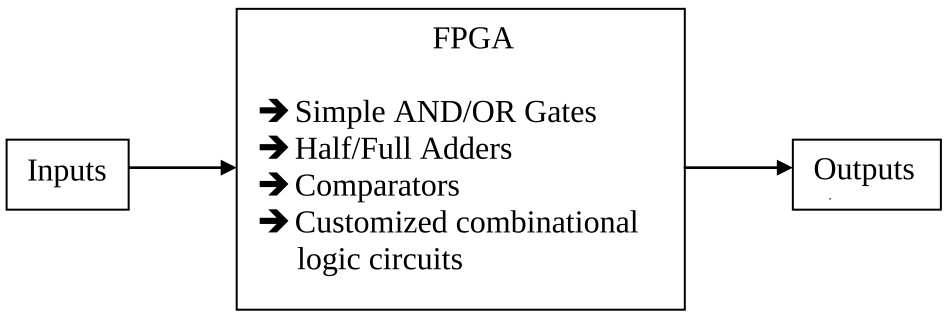


Figure 5.3 Overview of FPGA configuration examples using schematic design.

With studying combinational circuits by using schematic entity design, the basic concepts of circuit design are learnt, such as connecting components, pin assignments, adjusting Quartus II settings, compiling and testing the projects.

After implementing combinational circuits, next step is to design sequential circuits. Sequential logic is a type of logic circuit whose output depends not only on the present value input but also depends on the previous value of the input. To provide present and previous values together, memory or storage elements are used in sequential logic. Using memory or storage elements are the major difference between sequential logic and combinational logic circuits. Counter is a sequential logic device which stores and sometimes displays the number of times a particular event or process has occured. For counter design, a memory buffer with clock and counter pins are needed.

At design step, clock signal is connected to global clock signal of UP3, and reset signal is connected to global reset pushbutton. After testing these connections, clock signal is same value with global clock signal but, reset signal isn't same value with the of global reset pushbutton. To overcome this issue, reset signal is connected to one of DIP switches and tested it with new connection. Using a switch for reset signal is solved this problem so this approach has used for all sequential projects during the study. When solving reset signal issue, we have tested counter implementation by watching LEDs on UP3 development kit. Sequentially lighted and unlighted LEDs showed us the correctness of this implementation.

Previous examples with combinational and sequential logic circuits were implemented by using scheamatic entity design. After these examples next step is to start learning and making some examples with one of hardware description languages. VHDL and Verilog HDL are compared in different aspects such as, ease of use, ease of understand, ease of learn, etc... and VHDL is selected as a hardware description language. Previous examples are coded and tested in VHDL. Learning VHDL and understanding the parallelism concept are taken a very long time. Because if the complexity of the implementation increases, following and understanding the circuit behaviour becomes more difficult. On the otherhand, controlling hardware within a pre-determined rising edge or falling edge of clock signal is the biggest advantage of using VHDL.

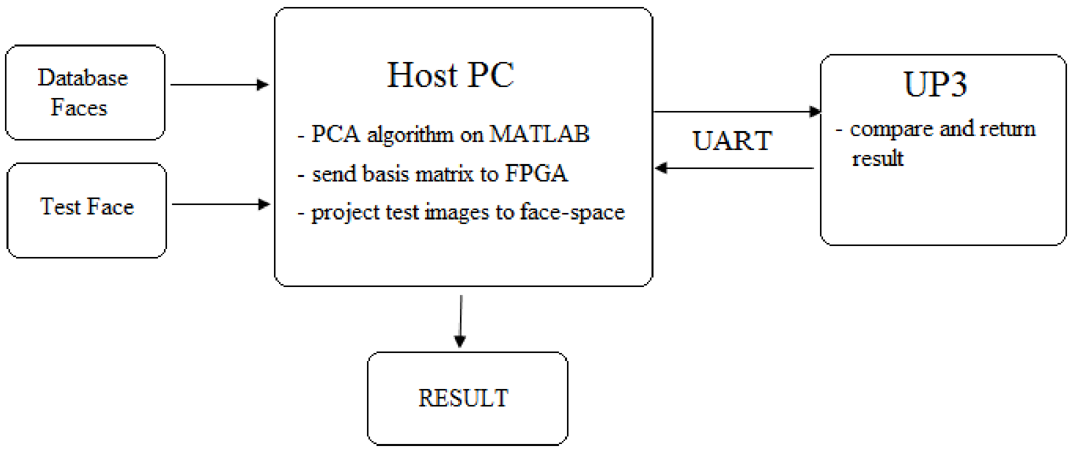

Database Faces
Test Face
Host PC
- PCA algorithm on MATLAB
- send basis matrix to FPGA
- project test images to face-space
UART
UP3
- compare and return result
RESULT

Figure 5.4 First approach of our face recognition system design.

For the implementation of face recognition system, a system that includes a PC and a UP3 development kit is preferred. Figure 5.4 summarizes this face recognition system. In this system, major operations such as capturing and processing image, computation of principal component analysis (PCA) is implemented on PC.

As described in Figure 5.4, PCA algorithm is run on MATLAB for creating features for database images and these features are sent to FPGA via serial port and will be stored in FPGA's SRAM Cells. When a face image is tested, this must be projected to the face space by using PCA basis matrix on host PC and then, projected test image will be sent to FPGA to compare and determine the result. At last, the result will be sent back to host PC to show which user is recognized. This system and design concepts are discussed in the following sections. In Section 5.1.3 PCA algorithm implementation in MATLAB is described. Section 5.1.4 gives the basic concepts of Universal Asynchronous Receiver/Transmitter (UART) structure and Section 5.1.5 shows UART implementation in VHDL. The results and findings are mentioned in Section 5.1.6.

5.1.3 PCA implementation on MATLAB

Before implementing PCA on UP3 development kit, first PCA algorithm is implemented in MATLAB. ORL Database is used for this implementation. There are 20 people and 10 images for each person in this database. The size of the images in this database are 112x92. Figure 5.5 shows some example images from this database.

As a start 8 images of 6 people are selected for creating the database and the rest 2 images are used for tests. The source code of this system is in the Appendix with the folder name of "5_1_3_PCA_MATLAB".

FaceRec.m is the main MATLAB function of this implementation. In this implementation, first the images that are used for database are resized from 112x92 to 48x48 and stored on a different folder. Each image on the new folder are transformed into vectors and placed one column of new population matrix. For example, first image in the folder is converted to a vector of 2304x1, then this vector is put to first column of population matrix. This process is continued for all images. By processing for 48 images (6 people and 8 images for each people), population matrix is created with size of 2304x48. *Loadpop.m* is responsible for this duty. This population matrix, *X*, is used for creating the basis function. *Makebasis.m* shows the PCA algorithm implementation. First, covariance of *X* is computed then eigenvalues and eigenfaces are found. By sorting eigenvalues in descending order, eigenface vector or face space vector, *A*, is computed. The size of *A* is 2304x2304. Next step is to project population matrix, *X*, to face space by multiplying the face space vector, *A*. The result of this multiplication is stated as *Ytrain* in the source code. To reduce the computation time, only the first a few eigenvectors are used. User enters the number of eigenvectors that will be used on creating *Ytrain* matrix in the start of the programme.

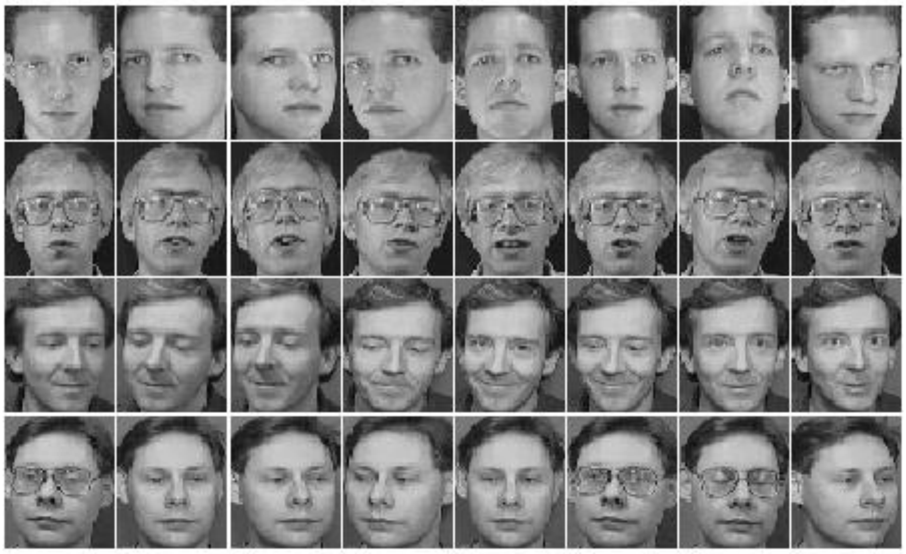
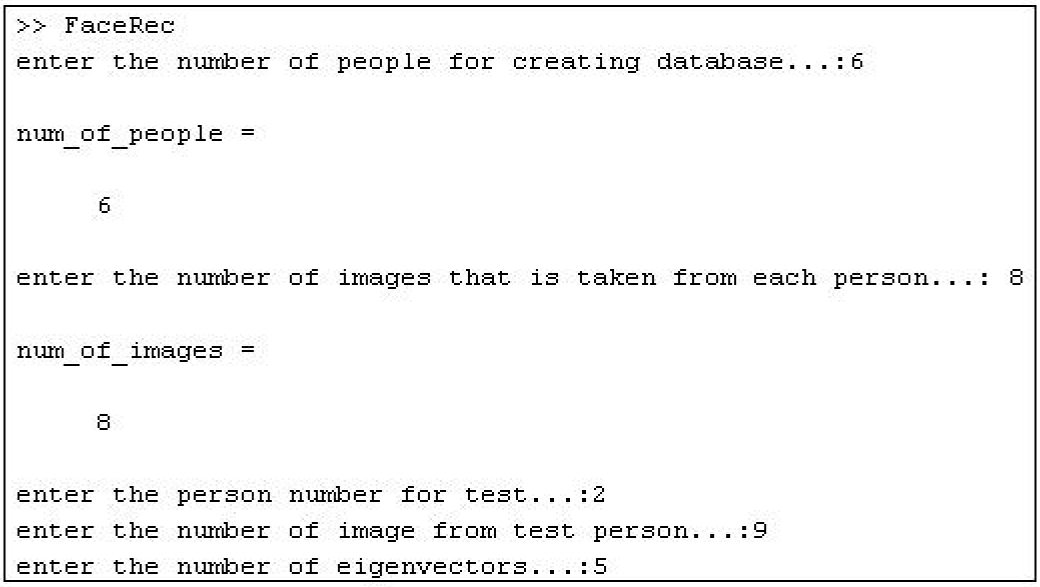


Figure 5.5 Some examples from ORL Database.

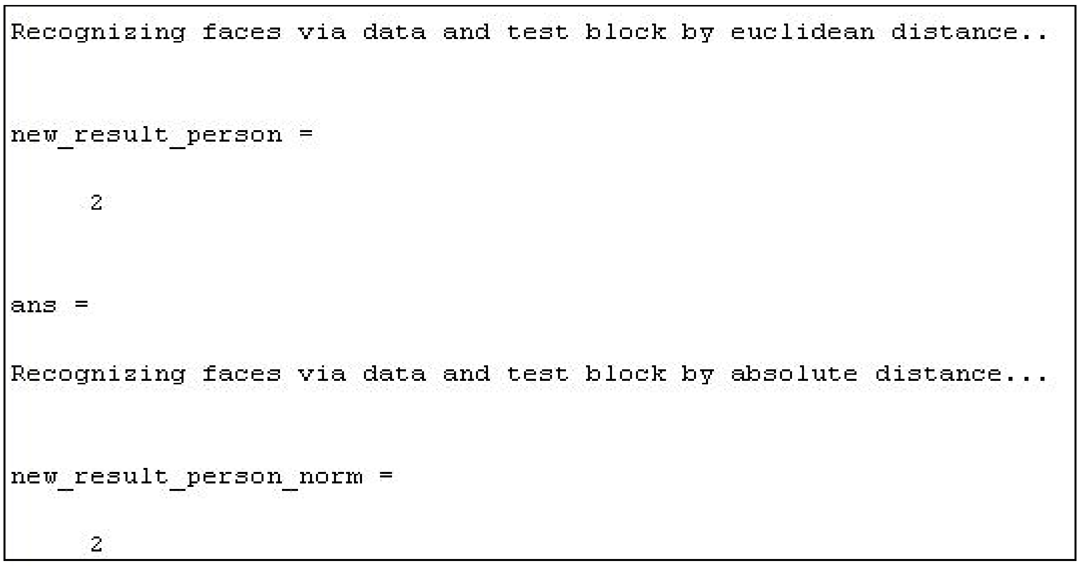
Test step is similar with database creation step. User also enters the person and image number that will be tested. First, image is resized then projected face space by multiplying by *A*. The result of this multiplication is *Xtest*. After computing *Xtest*, the last thing is to compare *Ytrain* and *Xtest* by using Euclidean distance and absolute distance, then to return result to the user.

Figure 5.6 shows the user’s inputs. As seen from this figure the number of people and images from each person for database creation is asked from the user. The image from will be tested is also asked with the number of eigenface that will be used.



```
>> FaceRec
enter the number of people for creating database...:6

num_of_people =

     6

enter the number of images that is taken from each person...: 8

num_of_images =

     8

enter the person number for test...:2
enter the number of image from test person...:9
enter the number of eigenvectors...:5
```

Figure 5.6 User’s inputs face recognition programme on MATLAB.



```
Recognizing faces via data and test block by euclidean distance..


new_result_person =

     2


ans =

Recognizing faces via data and test block by absolute distance...


new_result_person_norm =

     2
```

Figure 5.7 Output of the programme on MATLAB.

Figure 5.7 shows output of the programme. *new_result_person* and *new_result_person_norm* holds the result of comparision between *Ytrain* and *Xtest* by using Euclidean distance and absolute distance respectively.

Figure 5.8.a shows the tested image on this example and Figure 5.8.b shows the face images of test person in the database. Note that, the poses of the 2nd person images in the database are different, but PCA algorithm has successfuly identified the person. Figure 5.9 shows another test on the system.

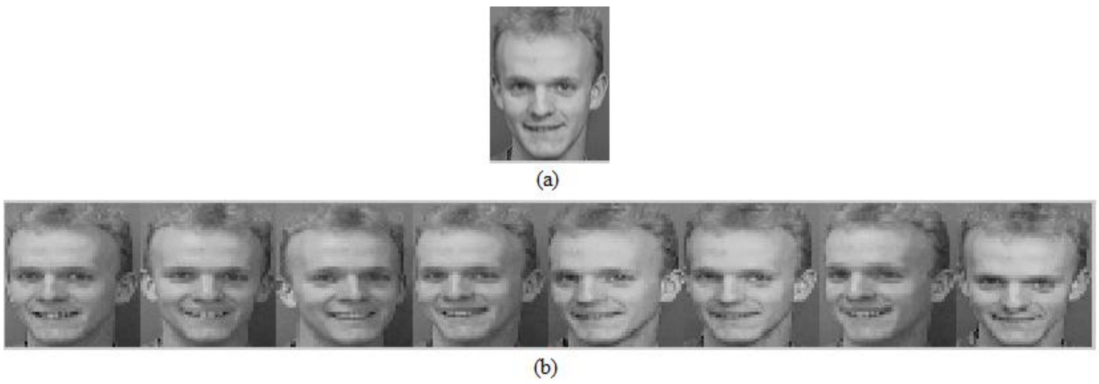


Figure 5.8 a) 9th image of 2nd person that used for test. b) First 6 image of 2nd person. Note that these images are included in database.

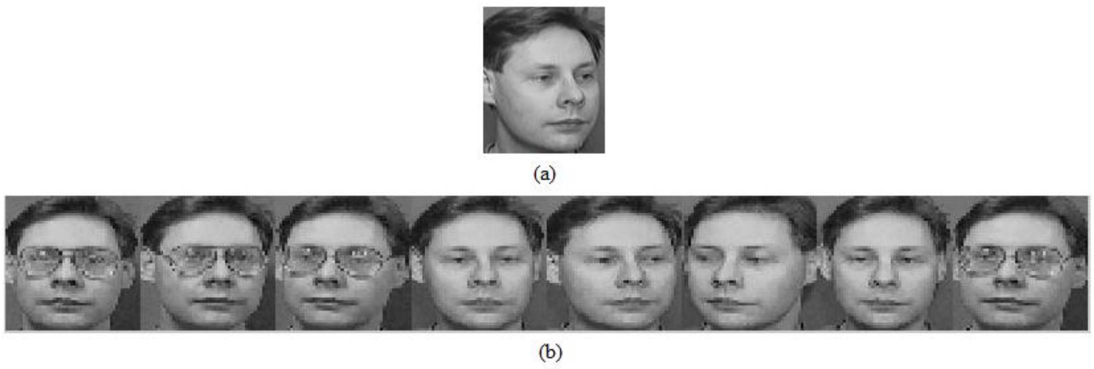


Figure 5.9 Another test result. a) 10th image of 1st person that used for test. b) First 6 image of 1st person. Note that these images are included in database.

Table 5.1 Recognition table for this system

Recognition Table		test image	
		9th	10th
person number	1st	+	+
	2nd	+	+
	3rd	+	+
	4th	+	+
	5th	+	+
	6th	+	+

To understand the performance of this system recognition rate must be computed with testing all images that are not used for database creation. Table 5.1 shows the test results for each images.

Note that “+” signs in Table 5.1 shows that recognition for this image is correct. The all results are correct so the recognition rate for this system is %100. Before implementing this system on FPGA, changing some parameters allows to understand the variying on recognition rate. By using the same PCA algorithm new systems are created by changing image numbers for database and changing the number of eigenvectors. “System-II”, “System-III” and “System-IV” includes 6 images from 6 people with the number of eigenvectors that used for creating face spaces are 5, 3 and 1 respectively. Recognition tables for these systems are given below. Note that, 7th, 8th, 9th and 10th images of each person are tested.

Table 5.2 Recognition table for System-II with 5 eigenvectors.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	+
	2nd	+	+	+	+
	3rd	+	+	+	+
	4th	+	+	+	+
	5th	+	+	+	+
	6th	+	+	+	+

Table 5.2 shows that the recognition rate of System-II is %100. When comparing Table 5.1 and 5.2, it can be seen that image numbers that are used for creating database can be decreased. It is too important because, the aim of the project is to implement this algorithm on FPGA. In embedded system designs, using less resources decrease the implementation cost and computation time.

Table 5.3 Recognition table for System-III with 3 eigenvectors.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	-
	2nd	+	+	+	+
	3rd	+	+	+	+
	4th	+	+	+	+
	5th	+	+	+	+
	6th	+	+	+	+

Table 5.4 Recognition table for System-IV with 1 eigenvector.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	-
	2nd	+	-	+	+
	3rd	+	+	+	+
	4th	+	-	-	-
	5th	-	+	-	-
	6th	-	-	+	-

Table 5.2, Table 5.3 and Table 5.4 shows the effect of the number of eigenvectors that are used for creating the system. Note that “-” marks in Table 5.3 and Table 5.4 show that the recognition is fail for that images. Table 5.3 shows that the recognition rate of System-III is approximately %96. By comparing Table 5.2 and Table 5.3, decreasing the number eigenvectors shows an acceptable recognition rate. Decreasing the number of eigenvectors is also important to decrease computation time. Table 5.4 shows that the recognition rate is nearly %54 in System-IV. This recognition rate is unacceptable for system design.

From the comparisons of Table 5.1 with Table 5.2 and Table 5.2 with Table 5.3, using 6 images for creating database and selecting 3 eigenvectors are enough for PCA algorithm implementation on FPGA. This know-how is used for further system implementations on FPGA.

After completing the PCA implementation on MATLAB, according to targeted implementation in Figure 5.4, serial communication with host PC and FPGA is studied. Following section briefly describes the basic concepts of UART.

5.1.4 UART

This section briefly describes a universal asynchronous receiver/transmitter (UART). UART is the part of computer hardware that translates data between parallel and serial forms. Today, UARTs are commonly included in microcontrollers and they are commonly used in conjunction with other communication standards such as EIA RS-232.

UART controller is the key component of the serial communications subsystem of computer hardware. The UART takes bytes of data and transmits the individual bits in a sequential bit-stream. At the destination, a second UART takes this bit-stream and re-assembles these bits into complete bytes. Serial transmission of digital information is much more cost effective than parallel transmission through multiple wires.

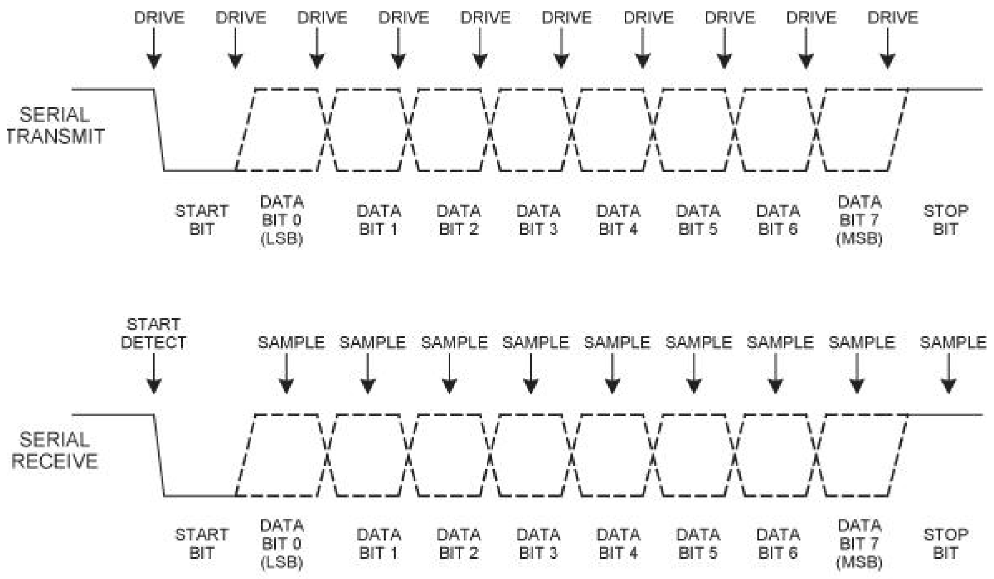


Figure 5.10 Transmit and receive waveforms for 10-bit asynchronous serial protocol (Maxim, 2007).

In asynchronous transmitting, UARTs send a "start" bit, five to eight data bits, from least significant bit (LSB) to most significant bit (MSB), an optional "parity" bit, and then one, one and a half, or two "stop" bits. The state of start bit is the opposite polarity of the data-line's idle state and the state of stop bit is the data-line's idle state. The stop bit provides a delay before the next character can start.

Asynchronous transmission allows data to be transmitted without sending the clock signal the receiver. So, the sender and receiver must agree on timing parameters. Framing error and parity check are the most common error detection methods that are used in UARTs.

Figure 5.11 shows the internal structure of the UART. UART has 3 main components: Baud Rate Generator, Transmitter and Receiver.

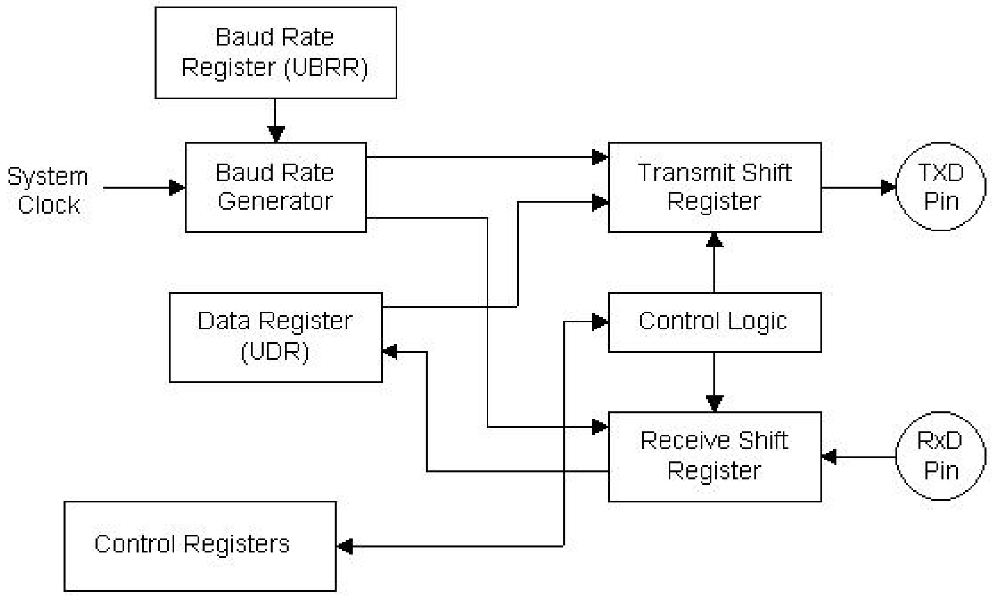


Figure 5.11 Internal structure of the UART (Weinstein, Volz, & Redecker, 2004).

5.1.4.1 UART Baud Rate Generator

The UART Baud Rate Generator defines and generates the clock used for transmitting and receiving data via the UART. UART clock can be divided very precisely to acquire an error-free bit transportation. Figure 5.12 shows the baud rate generator.

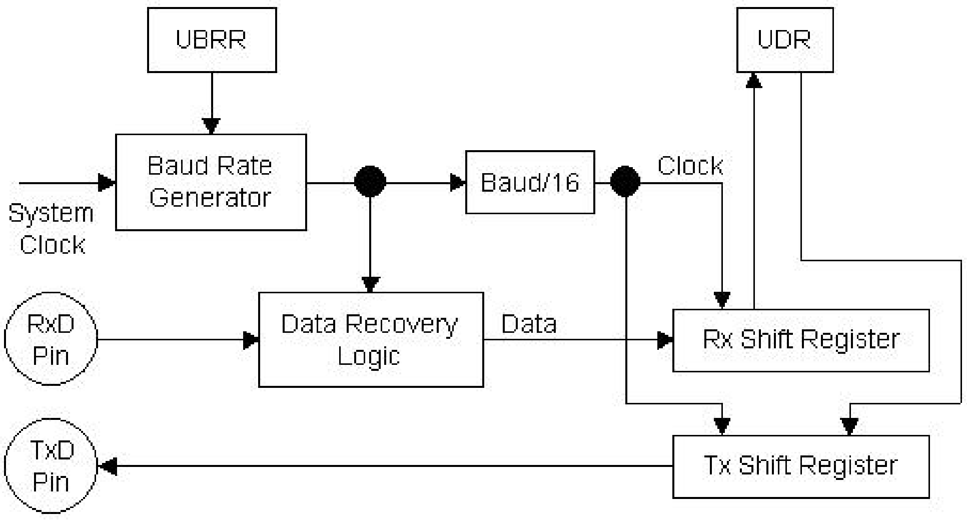


Figure 5.12 Baud rate generator (Weinstein, Volz, & Redecker, 2004).

In Figure 5.12, the clock generated by the UART baud rate generator is 16 times higher than the baud rate that needed for transferring data. This clock is used by the Data Recovery Logic.This module samples the data and filters it a bit, so that less errors occur in data receiving.

The clock used for shifting in the data is divided by 16 and therefore corresponds to the baud rate.The formula for calculating the Baud rate generated from a UART Baud Rate Register (UBRR) is;

$$BAUD = \frac{f_{CLK}}{16(UBRR+1)} \qquad (5\text{-}1)$$

But, in practice system clock and baud rate are constant and known values. So modifying Eq. (5-1) yield that Eq. (5-2). UBRR register is programmed hardware or software to satisfy this clock and baud rate.

$$UBRR = \frac{f_{CLK}}{(16xBAUD)} - 1 \qquad (5\text{-}2)$$

If system clock is 50 MHz and 115200 Baud is needed, the UBRR value 26.126736. So, it’s 26. The error due to baud rate is the actual baud rate divided by the desired baud rate. The actual baud rate can be computed from Eq. (5-1), 115740 baud. 115740/115200, is 1.005 and therefore error is %0.005.

5.1.4.2 UART Transmitter

The UART transmitter sends data from the UART core to some other device such as data logger, PC, another UART, etc...) at the specified Baud Rate. Figure 5.13 shows the internal structure of UART transmitter.

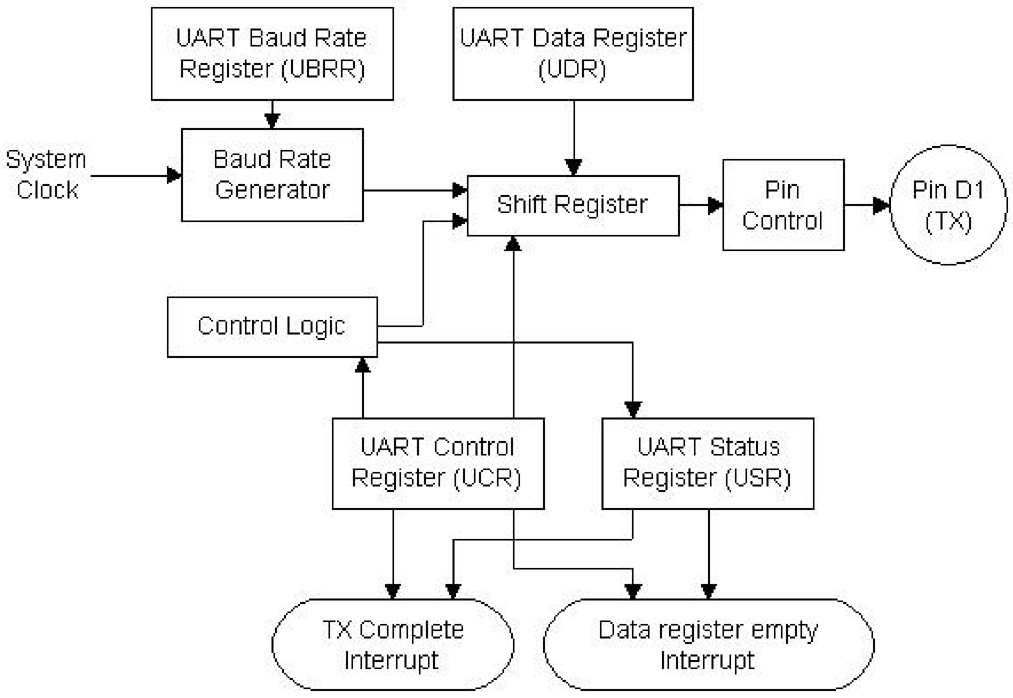


Figure 5.13 Internal structure of UART transmitter (Weinstein, Volz, & Redecker, 2004).

The transmission process is initiated by writing data to Uart Data Register (UDR). This data is then transferred to the TX shift register when the previously written byte has been shifted out completely. When a byte is transferred to the TX shift register, the Uart Data Register Empty (UDRE) flag is set. The interrupt service routine of UDRE can now write the next byte to UDR without corrupting the transmission in progress. When a byte is completely shifted out and no new data has been written to UDR, shows the transmission is over, transmit complete (TXC) or end of transmisson (EOT) flag is set.

5.1.4.3 UART Receiver

The UART receiver is basically built up like the transmitter. UART receiver uses data recovery logic for sampling the data and set an interrupt for the completion of data reception. The data is sampled in the middle of the bit to be received because the baud rate of the data recovery logic is 16 times of the actual baud rate. The baud rate value for shifting bits is same with the transmitter. When the reception is over, receiver complete (RXC) or end of reception(EOR) flag is set. Figure 5.14 shows the internal structure of UART receiver.

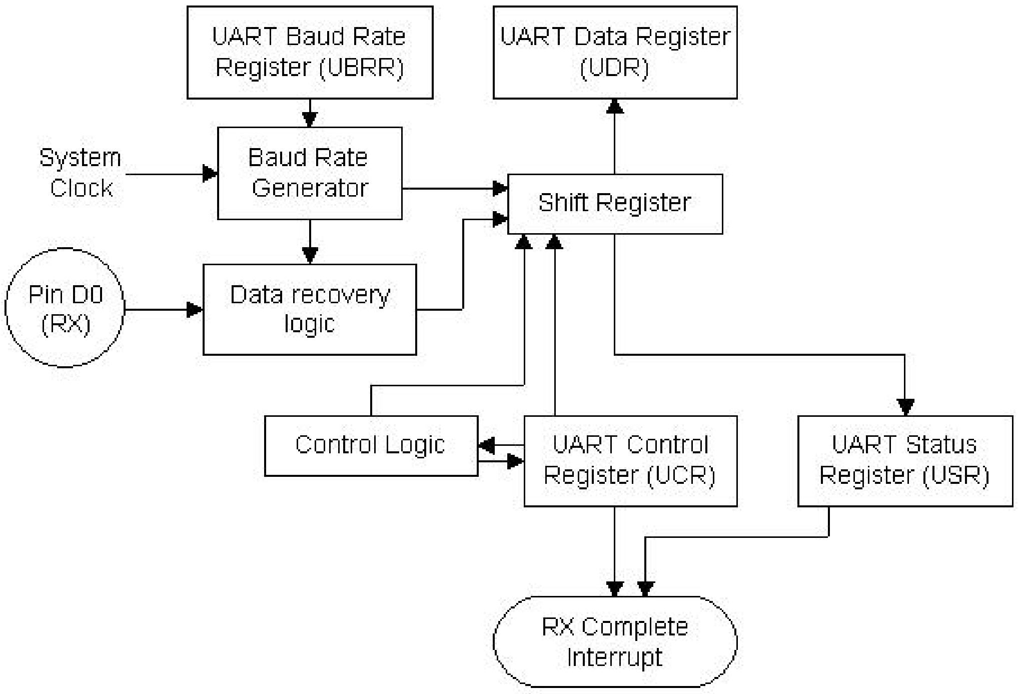


Figure 5.14 Internal structure of UART receiver (Weinstein, Volz, & Redecker, 2004).

5.1.5 UART Implementation in VHDL

In this thesis UART implementation in VHDL is started with baud generator design. Then transmitter and receiver parts are implemented respectively. In the test step, hyperterminal and MATLAB serial communication toolbox are used. Note that baud generator design is tested when transmitter design is finished. The implementation results are described in the Section 5.1.6.

5.1.5.1 Baud Generator Design in VHDL

Baud generator is a component that is used for both transmitter and receiver designs. Actually this module is simply a clock counter. For providing the transmission baud rate, baud generator counts clock ticks and create a new clock period. The entity of baud generator that used for transmitter is shown in Figure 5.15. This module takes a global clock input with transmitter's baud enable to start the transmission, and produces baud rate as an output. Figure 5.16 shows the code segment that counts clock ticks and used for generating 115200 baud rate as an output where *dividenum* is "01A0h".

```vhdl
entity baudgen is
   port (global_clock        : in  std_logic;
         tx_ck_enable_baud   : in  std_logic;
         baud_rate           : out std_logic
        );
end baudgen;
```

Figure 5.15 Baud generator entity that is designed for UART transmitter.

```vhdl
NEWBAUD : process (global_clock)

begin
if(global_clock = '1' and global_clock'event) then
        if (counter = dividenum) then
                        baud_rate <= '1';
                        counter <= (others => '0');
        elsif (tx_ck_enable_baud = '1') then
                        baud_rate <= '0';
                        counter <= counter + 1;
        else
                        baud_rate <= '0';
                        counter <= (others => '0');
        end if;
end if;
```

Figure 5.16 Code segment that is used for generating 115200 baud rate.

5.1.5.2 UART Transmitter Design in VHDL

Transmitter is dutied for transmitting bytes with a speed of baud rate. Figure 5.17 shows the entity of transmitter module. In this module, *g_clock* is the global clock signal that is routed to baud generator module, *tx_data* is the output bit of the state machine, *eoto* is the output that shows the end of transmission, *data* is the 8-bit input data that is sent via transmitter and *write_tr* is the transmitter enable input bit.

```
entity uart_transmitter is
   port (g_clock  : in  std_logic;
         tx_data  : out  std_logic;
         eoto     : out  std_logic;
         data     : in  std_logic_vector(7 downto 0);
         write_tr : in std_logic
        );
end uart_transmitter;
```

Figure 5.17 UART transmitter entity.

Figure 5.18 shows the transmitter state machine. State machine is a key component of transmitter component because in this component, one byte of 8-bit input data is decomposed as a bit in each state, then sent respect to baud rate. The working principle of the state machine is same with the serial transmit in Figure 5.10. The flow of these states are started with transmit enable (*tx_enable*). If transmitter does not send any data, it is always in S0 state. S0 state is the idle state of transmitter. When transmit is enabled, at S0 state, start bit is sent first. Then, each bit of the *data* is sent from LSB to MSB while states are processed from S1 to S8. After the transmission of each bit of *data* is completed, at S9 state, stop bit is sent and end of transmission flag is set. At S9 state, the next state is also set to S0 and if there exist another byte that waits for transmission, transmitter state machine is started again for the next byte, until the last byte is sent.

```
TX_STATE_MACHINE: process(current_state,tx_enable,datain,data)

        begin
                case current_state is

                        when S0 =>
                                datain <= data;
                                eot <= '0';
                                if (tx_enable = '1') then
                                        txd <= '0';
                                        next_state <= S1;
                                        TRANSMITTING<='1';
                                else
                                        txd <= '1';
                                        TRANSMITTING<='0';
                                        next_state <= S0;
                                end if;
                        when S1 =>
                                txd <= datain(0);
                                eot <= '0';
                                TRANSMITTING<='1';
                                next_state <= S2;
                        when S2 =>
                                txd <= datain(1);
                                eot <= '0';
                                TRANSMITTING<='1';
                                next_state <= S3;
                        when S3 =>
                                txd <= datain(2);
                                eot <= '0';
                                TRANSMITTING<='1';
                                next_state <= S4;
                        when S4 =>
                                txd <= datain(3);
                                eot <= '0';
                                TRANSMITTING<='1';
                                next_state <= S5;
                        when S5 =>
                                txd <= datain(4);
                                eot <= '0';
                                TRANSMITTING<='1';
                                next_state <= S6;
                        when S6 =>
                                txd <= datain(5);
                                eot <= '0';
                                TRANSMITTING<='1';
                                next_state <= S7;
                        when S7 =>
                                txd <= datain(6);
                                eot <= '0';
                                TRANSMITTING<='1';
                                next_state <= S8;
                        when S8 =>
                                txd <= datain(7);
                                eot <= '0';
                                TRANSMITTING<='1';
                                next_state <= S9;
                        when S9 =>
                                txd <= '1';
                                eot <= '1';
                                TRANSMITTING<='1';
                                next_state <= S0;
                        when others =>
                                null;

                end case;

        end process TX_STATE_MACHINE;
```

Figure 5.18 Transmitter state machine.

5.1.5.3 UART Receiver Design in VHDL

The receiver is responsible for capturing the input bit stream and composing these bits to a 8-bit data vector. Figure 5.19 shows the receiver entity. In this module, *gl_clock* is the global clock signal that is routed to baud generator module, *rx_data* is the input bit, *rx_enable* shows enable of receiver, *rx_ck_baud_enable* shows enable of receiver's baud generator, *data_out* is the 8-bit data vector output of receiver module, *data_rdy* is signaled when *data_out* is constructed and *eoro* is the end of receiver flag.

```
entity uart_receiver is

   port (gl_clock             : in  std_logic;
         rx_data              : in  std_logic;
         rx_enable            : in  std_logic;
         rx_ck_baud_enable    : in  std_logic;
         data_out             : out  std_logic_vector(7 downto 0);
         data_rdy             : out  std_logic;
         eoro                 : out std_logic
         );

end uart_receiver;
```

Figure 5.19 UART receiver entity.

Figure 5.19 shows the receiver state machine and receiver data shift mechanism. Note that, *rec_baud_clock* in data shifting process, is the output of receiver's baud generator. Receiver state machine is similar to the transmitter state machine. The working principle of the state machine is same with the serial receive in Figure 5.10. The flow of these states are started with receive enable (*start*). If receiver is on a wait condition, it is always in S0 state. S0 state is the idle state of receiver. When reception is started, at S0 state, start bit is received first. Then, each bit is shifted from LSB to MSB and while states are processed from S1 to S8. The shifted bits are stored in *data*. If stop bit is received at the end of the reception of each data bit, at S9 state, end of receiver flag is set. *data_rdy* flag is also set at receiver when reception is finished. At S9 state, the next state is also set to S0 and receiver is started to listen to the port.

```
RXD_STATE_MACHINE:process(start,current_state,rx_enable,receiving)
        begin
        if (rx_enable = '0') then
                if (start = '1' or receiving = '1') then
                        case current_state is
                                when S0 =>
                                        eor <='0';
                                        if (start = '1') then
                                                next_state <= S1;
                                                receiving <= '1';
                                        else
                                                next_state <= S0;
                                                receiving <= '0';
                                        end if;
                                when S1 =>
                                        receiving <= '1';
                                        eor <= '0';
                                        next_state <= S2;
                                when S2 =>
                                        receiving <= '1';
                                        eor <= '0';
                                        next_state <= S3;
                                when S3 =>
                                        receiving <= '1';
                                        eor <= '0';
                                        next_state <= S4;
                                when S4 =>
                                        receiving <= '1';
                                        eor <= '0';
                                        next_state <= S5;
                                when S5 =>
                                        receiving <= '1';
                                        eor <= '0';
                                        next_state <= S6;
                                when S6 =>
                                        receiving <= '1';
                                        eor <= '0';
                                        next_state <= S7;
                                when S7 =>
                                        receiving <= '1';
                                        eor <= '0';
                                        next_state <= S8;
                                when S8 =>
                                        receiving <= '1';
                                        eor <= '0';
                                        next_state <= S9;
                                when S9 =>
                                        receiving <= '1';
                                        eor <= '1';
                                        next_state <= S0;
                                when others =>
                                        null;
                        end case;
                end if;
        end if;
        end process RXD_STATE_MACHINE;

RXD_SHIFT : process (rec_baud_clock,rx_enable)
        begin
        if (rx_enable = '0') then
                if (rising_edge(rec_baud_clock)) then
                        if (eor = '0') then
                                data <= rx_data & data(7 downto 1);
                        end if;
                end if;
        end if;
        end process RXD_SHIFT;
```

Figure 5.19 Receiver state machine.

5.1.6 UART Implementation Results and Findings

To test the UART implementation; Quartus II simulator tool, hyperterminal and MATLAB serial communication toolbox are used. This section shows the results of some test subsystems such as, one byte transmitter design, one byte receiver design and array transmitter design. For face recognition test system, as illustrated in Figure 5.4, database and test features are extracted in MATLAB then these features are compared in UP3 development kit. First the comparision system is simulated then testing with MATLAB.

5.1.6.1 One Byte Transmitter

Quartus II Simulator Tool is used for simulating this system. Simulation is started after creating the functional netlist on the project. Transmitter is initiated by the code segment that shown in Figure 5.20. Note that the components of this module are mentioned in Section 5.1.5.2. *datainput* is the one byte data that will be transmitted. The source code of this system is in the Appendix with the folder name of "5_1_6_1_UART_Transmitter". Figure 5.21 and Figure 5.22 shows the simulation results when datainput is "55h" and "66h" respectively.

```
START_ONLY_TRANSMIT: uart_transmitter port map (CLK,TX_OUT,
EOT_LED,datainput,write_main);
```

Figure 5.20 Call of transmitter part from main function.

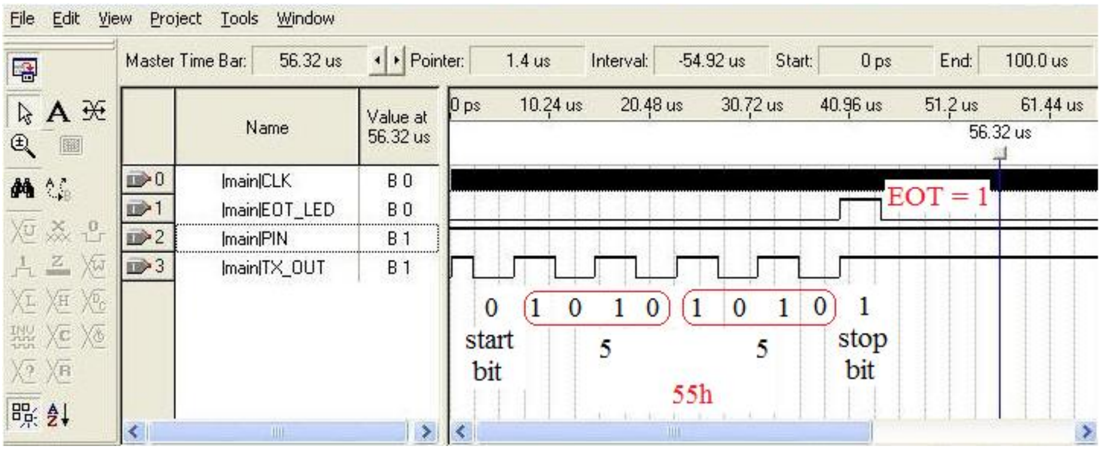


Figure 5.21 Transmitter simulation when datainput is 55h.

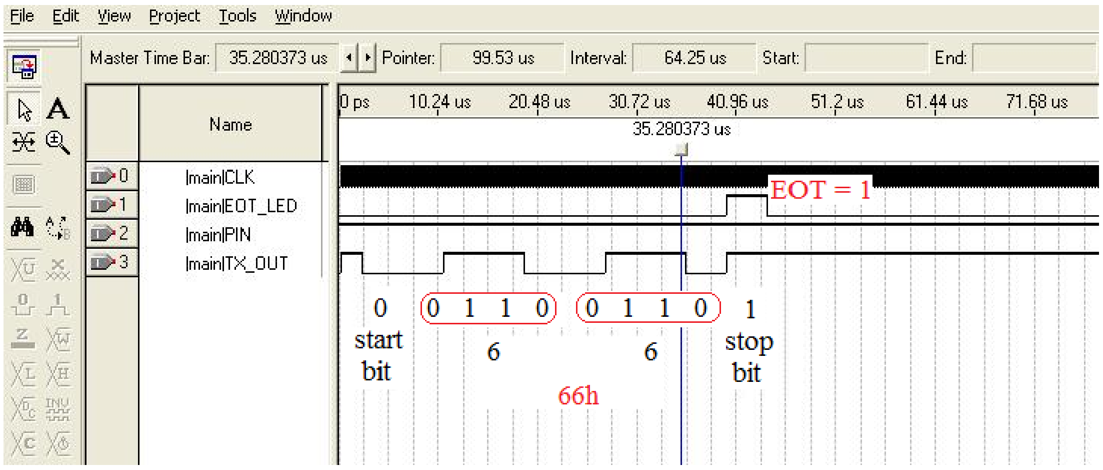


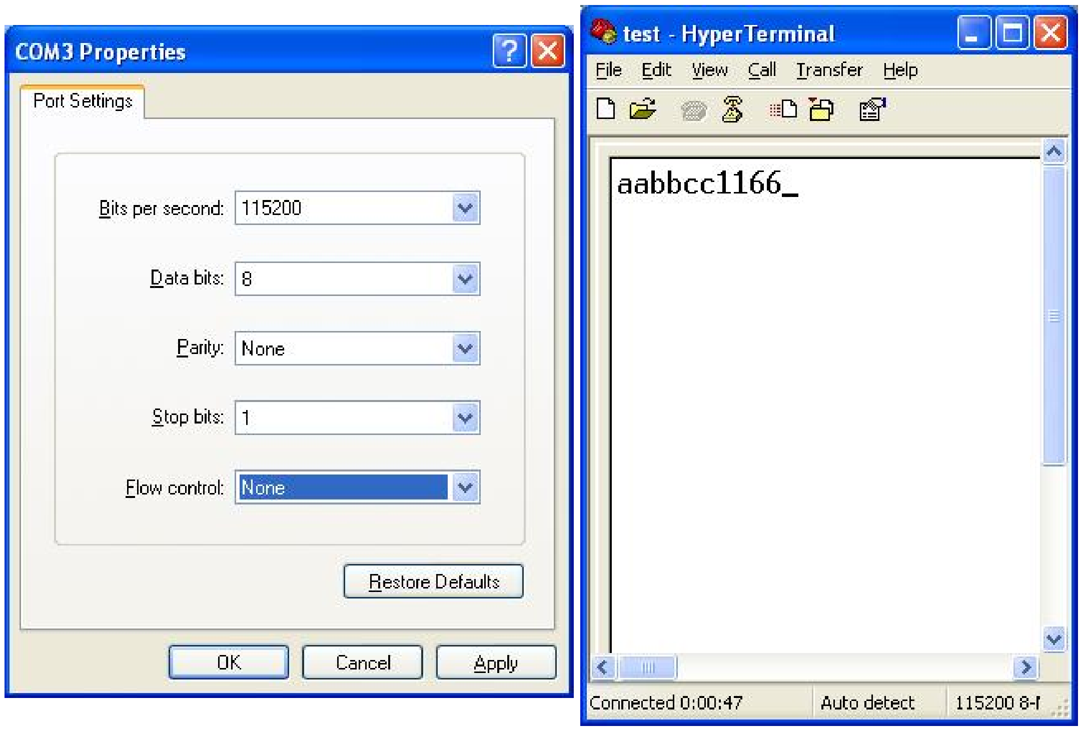
Figure 5.22 Transmitter simulation when datainput is 66h.

5.1.6.2 One Byte Receiver

After implementation and testing transmitter design, next step is to test receiver implementation. Figure 5.23 shows the code segment that used to call receiver. To observe if the receiver implementation works properly, the received data is sent back again via transmitter. The test of this subsytem is done by using hyperterminal. Figure 5.24 shows the test step with communication settings. When character “a” is sent from hyperterminal, the receiver module gets “a” and send back to hyperterminal via transmitter. “b,c,1,6” are also sent and received. The source code of this system is in the Appendix with the folder name of "5_1_6_2_UART_Receiver".

```
START_ONLY_RECEIVER: uart_receiver port map (CLK,RX_DATA,
receive_enable,reset_receiver,received_data,DATA_RDY,signal_EOR);

datainput <= received_data
EOR <= signal_EOR

START_ONLY_TRANSMIT: uart_transmitter port map (CLK,TX_OUT,
EOT_LED,datainput,write_main);
```

Figure 5.23 Call of receiver and transmitter part from main function.



Bits per second: 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

```
aabbcc1166_
```

Connected 0:00:47 Auto detect 115200 8-N

Figure 5.24 Communication port settings and receiver test.

5.1.6.3 Array Transmitter

Array transmitter is similar to the one byte transmitter. Figure 5.25 shows the definition of array. 10h, 20h, 30h and 40h are the elements of this array. One byte of this array is sent until the transmission of all bytes are completed. Figure 5.26 shows the simulation of array transmitter. The source code of this system is in the Appendix with the folder name of “5_1_6_3_UART_Array_Transmitter”.

```
type data_array is array (3 downto 0) of std_logic_vector (7
     downto 0);

signal data_block : data_array := ( 0 => x”10”,  1 => x”20”,
                                    2 => x”30”,  3 => x”40” );
```

Figure 5.25 Definition of array.

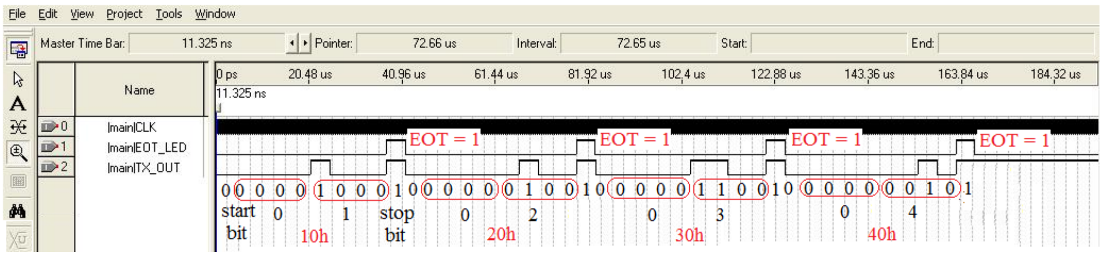


Figure 5.26 Simulation of array transmitter.

5.1.6.4 Simulation of Internal Database and Test Comparison

This study is the first step of comparison. Internal database and test block are created for testing this operation. These blocks are shown in Figure 5.27. *RAM* symbolizes the database which has 4 different images with 4 bytes of each element. *RAMtest* symbolizes the test image with 4 bytes.

```
type RAM is array (2 ** ADDRESS_WIDTH - 1 DOWNTO 0) ) of
std_logic_vector (DATA_WIDTH - 1 DOWNTO 0);
signal data_block : RAM := (0      => x"04",1      => x"04",
                            2      => x"04",3      => x"04",
                            4      => x"02",5      => x"02",
                            6      => x"02",7      => x"02",
                            8      => x"03",9      => x"03",
                           10      => x"03",11      => x"03",
                           12      => x"05",13      => x"05",
                           14      => x"05",15      => x"05");
type RAMtest is array (ADDRESS_WIDTH - 1 DOWNTO 0) OF of
std_logic_vector (DATA_WIDTH - 1 DOWNTO 0);
signal new_block : RAMtest := (0      => x"01",1      => x"01",
                               2      => x"01",3      => x"01");
```

Figure 5.27 Definition of internal database and test.

From Figure 5.27, it can be easily seen that the *RAMtest* is closer to *RAM*'s second element than the others. The result of this comparison is sent via transmitter, so it can be simulated like in the Figure 5.28. The source code of this system is in the Appendix with the folder name of "5_1_6_4_Internal_Database_And_Test".

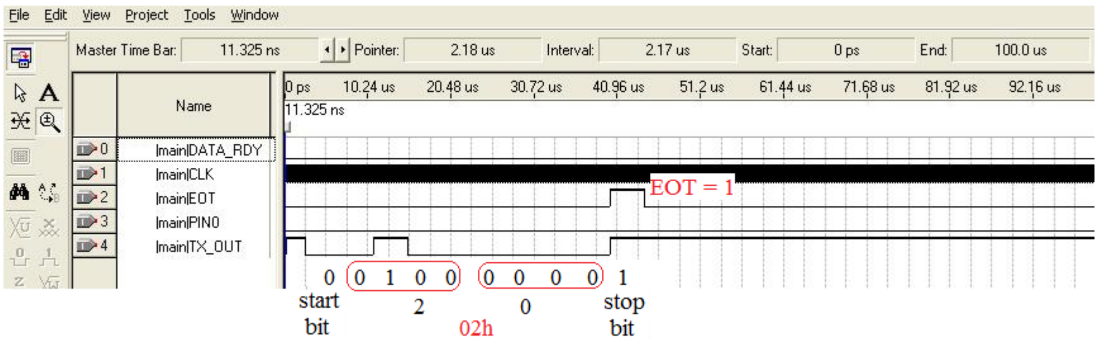

Master Time Bar: 11.325 ns Pointer: 2.18 us Interval: 2.17 us Start: 0 ps End: 100.0 us

Figure 5.28 Simulation of comparison result, 2, is sent via transmitter.

5.1.6.5 Comparing Database and Test after Receiving from MATLAB

After testing the comparison system internally, by using the same approach in Section 5.1.6.4, 16 bytes of database and 4 bytes of test are compared. The difference of this study is using UART implementation instead of simulation. Database and test are sent from MATLAB then these are compared on UP3, at last the result of comparison is sent to MATLAB again, and MATLAB shows the comparison result.

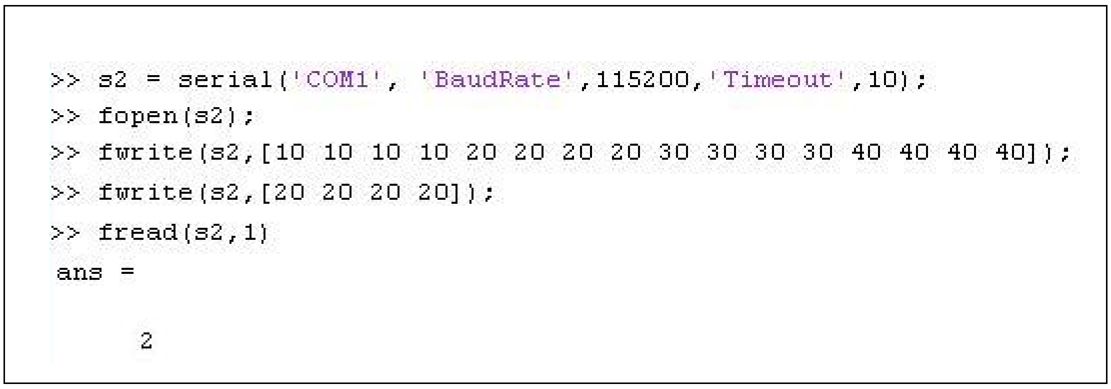

```
>> s2 = serial('COM1', 'BaudRate',115200,'Timeout',10);
>> fopen(s2);
>> fwrite(s2,[10 10 10 10 20 20 20 20 30 30 30 30 40 40 40 40]);
>> fwrite(s2,[20 20 20 20]);
>> fread(s2,1)
ans =

     2
```

Figure 5.29 Database and test are sent from MATLAB. Result is also read from MATLAB.

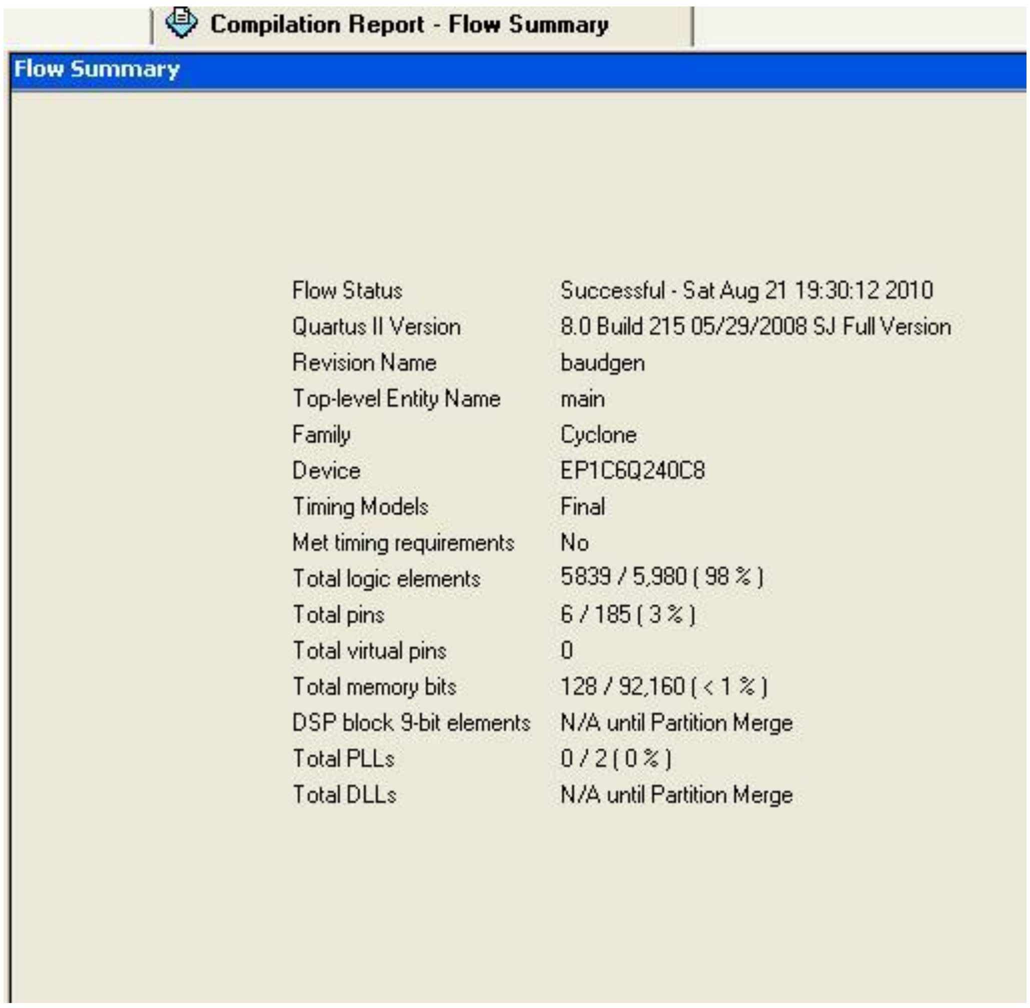
Figure 5.29 shows an example of this operation. At first a serial object, *s2*, is defined with baud rate 115200. Then serial port is opened to the communication by *fopen*. The database and test are sent to UP3 respectively by using *fwrite*. The result of the comparison is sent from UP3 and is readed by using *fread* in MATLAB. The source code of this system is in the Appendix with the folder name of "5_1_6_5_Database_and_Test_sent_from_MATLAB".

5.1.6.6 Face Recognition System on UP3 Development Kit

The last study on UP3 development kit is to test the system that shown in Figure 5.4. The aim is to realize the system that is given in Section 5.1.3. Face images from 6 people with 8 different poses images are used to create database. Then, this database must be stored on UP3 development kit. A database block is created with the size of 2304x48 and a test block is created with the size of 2304x1 like in the Figure 5.27. When, this system is tried to compile on Quartus II, compilation is ended with an error. This error states that this design cannot fit the device. The size of arrays are decreased to check whether the project is compile or not. Because the database block has more that 100000 elements. If the size of database block is decreased from 2304x48 to nearly 5000 elements, compilation is ended without an eror. But compilation report states that %98 percent logic elements of FPGA is used. This is shown in Figure 5.30.

As mentioned before, FPGA SRAM Cells are used as a memory on UP3. The error when the size of database block is 2304x48 shows that if the size of arrays are increased, UP3 is uncapable to store these arrays in internal SRAM Cells. After internal memory is not sufficient for this design, the 8 MB SDRAM on UP3 is planned to store these features. But to use 8 MB SDRAM on UP3 development kit, Nios II CPU, which is a soft-core processor, must be used to reach external memory resources. However, since the USB blaster cable to program Nios II CPU in UP3 kit is expensive, a new kit with higher memory resources has been purchased.

Fingerprint recognition implementation cannot be tested on UP3 development kit due to capability of this board The implementation of face and fingerprint recognition continoues with the new development kit. The implementation steps and the results are started to describe in the next section.



```
Flow Status                  Successful - Sat Aug 21 19:30:12 2010
Quartus II Version           8.0 Build 215 05/29/2008 SJ Full Version
Revision Name                baudgen
Top-level Entity Name        main
Family                       Cyclone
Device                       EP1C6Q240C8
Timing Models                Final
Met timing requirements      No
Total logic elements         5839 / 5,980 ( 98 % )
Total pins                   6 / 185 ( 3 % )
Total virtual pins           0
Total memory bits            128 / 92,160 ( < 1 % )
DSP block 9-bit elements     N/A until Partition Merge
Total PLLs                   0 / 2 ( 0 % )
Total DLLs                   N/A until Partition Merge
```

Figure 5.30 Compilation report with decreased size of database block.

5.2 Face and Fingerprint Recognition System Design on DE2-70

DE2-70 development kit has more powerful features than the UP3 development kit. These features are mentioned in Section 4.4.2. To use these features of new development kit, the implementation idea of the recognition system is changed. With the new kit, PCA algorithm is moved from MATLAB to FPGA. Both face and fingerprint recognition, only images are resized and sent from MATLAB, then the rest of all operations including PCA basis creation and comparison steps are implemented on DE2-70. Nios II CPU is designed with memory and I/O resources on Altera SOPC Builder and configured in Nios II IDE. Using Nios II IDE allows using C programming language instead of VHDL. So designing steps in C, is faster than VHDL.

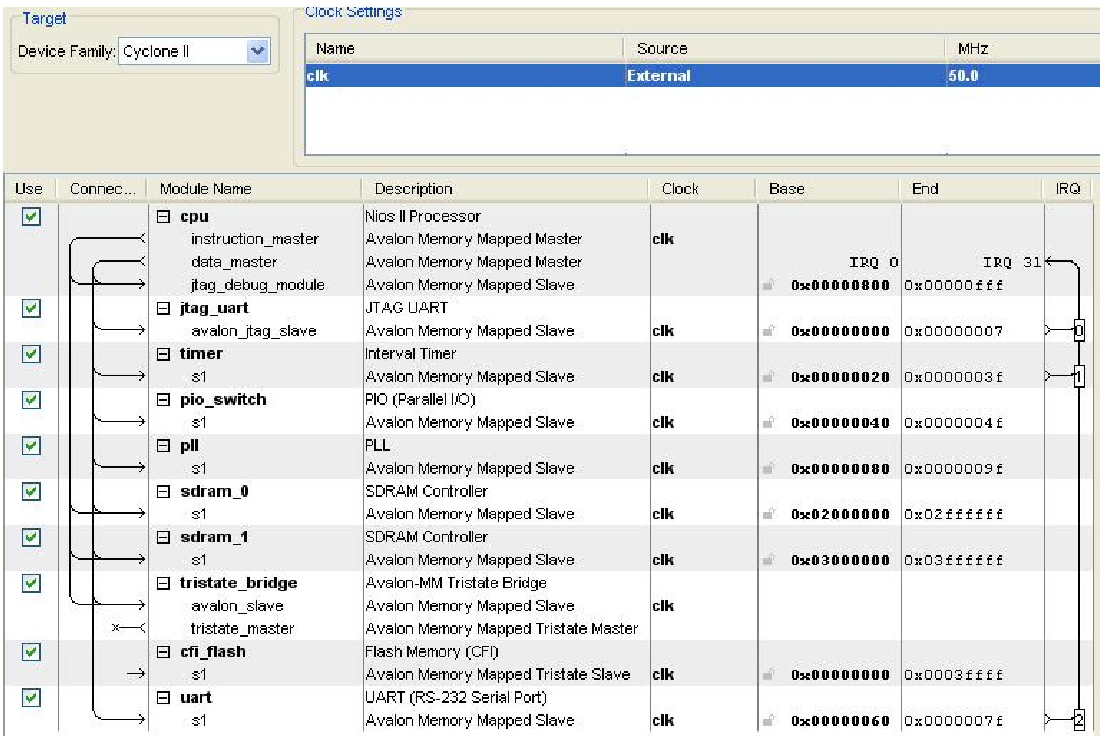
To use external memory resources on development kit, Nios II soft-core CPU must be used. Both UP3 and DE2-70 development kits have two programming modes: Parallel and Serial. In the parallel configuration mode, the EPCS programming flash is programmed and the configuration file isn't erased on power-off. Projects that are implemented on UP3 are configured in parallel programming mode. On DE2-70, serial programming mode is used, and system is programmed every power off via JTAG.

This section describes face and fingerprint recognition system by starting hardware design of the project. Section 5.2.1 shows and describes the hardware implementation. Section 5.2.2 describes the face recognition implementation on DE2-70 with results. In Section 5.2.3, the implementation of fingerprint recognition system is discussed.

5.2.1 Hardware Design of Face and Fingerprint Recognition System

Altera SOPC Builder is a tool of Quartus II software that is used for system on programmable chip (SOPC) designs. By using this tool, FPGA chip can be programmed as a CPU, Nios II CPU, and the other system components are integrated to system design easily. From Figure 5.30 to Figure 5.39, the integration of the components are shown. The design of the system starts with adding the design components. These components are Nios II CPU, phase locked loop (PLL), JTAG UART, interval timer, parallel input/output (PIO), SDRAM Controllers, Flash Memory and UART.

Figure 5.30 shows the SOPC Builder screen after all components added and the names of components are changed without any configuration. The next step is to configure these components.



Target

Device Family: Cyclone II

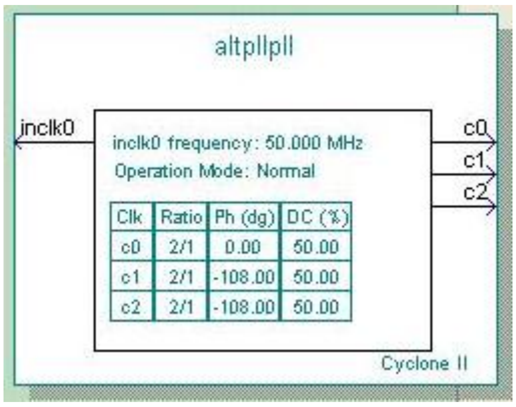
Clock Settings

Name	Source	MHz
clk	External	50.0

Use	Connec...	Module Name	Description	Clock	Base	End	IRQ
✓		cpu	Nios II Processor				
		instruction_master	Avalon Memory Mapped Master	clk			
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31	
		jtag_debug_module	Avalon Memory Mapped Slave		0x00000800	0x00000fff	
✓		jtag_uart	JTAG UART				
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk	0x00000000	0x00000007	0
✓		timer	Interval Timer				
		s1	Avalon Memory Mapped Slave	clk	0x00000020	0x0000003f	1
✓		pio_switch	PIO (Parallel I/O)				
		s1	Avalon Memory Mapped Slave	clk	0x00000040	0x0000004f	
✓		pll	PLL				
		s1	Avalon Memory Mapped Slave	clk	0x00000080	0x0000009f	
✓		sdram_0	SDRAM Controller				
		s1	Avalon Memory Mapped Slave	clk	0x02000000	0x02ffffff	
✓		sdram_1	SDRAM Controller				
		s1	Avalon Memory Mapped Slave	clk	0x03000000	0x03ffffff	
✓		tristate_bridge	Avalon-MM Tristate Bridge				
		avalon_slave	Avalon Memory Mapped Slave	clk			
		tristate_master	Avalon Memory Mapped Tristate Master				
✓		cfi_flash	Flash Memory (CFI)				
		s1	Avalon Memory Mapped Tristate Slave	clk	0x00000000	0x0003ffff	
✓		uart	UART (RS-232 Serial Port)				
		s1	Avalon Memory Mapped Slave	clk	0x00000060	0x0000007f	2

Figure 5.30 SOPC Builder screen after all components are added.

As shown in Figure 5.30, external clock source that is provided by the crystal on the development kit is 50 MHz. The SDRAM on the DE2-70 operates at 100 MHz, so to provide all components with same clock, PLL component is configured first. By using Altera ALTPLL MegaWizard, from the external 50 MHz clock, three clocks are generated. Figure 5.31 shows the generated clocks. In this figure, input clock, *inclk0*, is 50 MHz. *c0* is 100 MHz with the same phase of *inclk0*. *c1* and *c2* have a -108 degree phase difference even if the frequency is same with *c0*, 100 MHz. Clock phase shift setting is set to -3 ns from PLL wizard to provide this phase difference. Note that, SDRAM Controllers (*sdram_0* and *sdram_1*) that are illustrated in Figure 5.30, are gated same clock with CPU, but the memory components (real memory chips) on the development kit must be gated with the clocks (*c1* and *c2*) that have phase differences. -3 ns phase shift is found experimentally to equalize the CPU and SDRAM clocks.



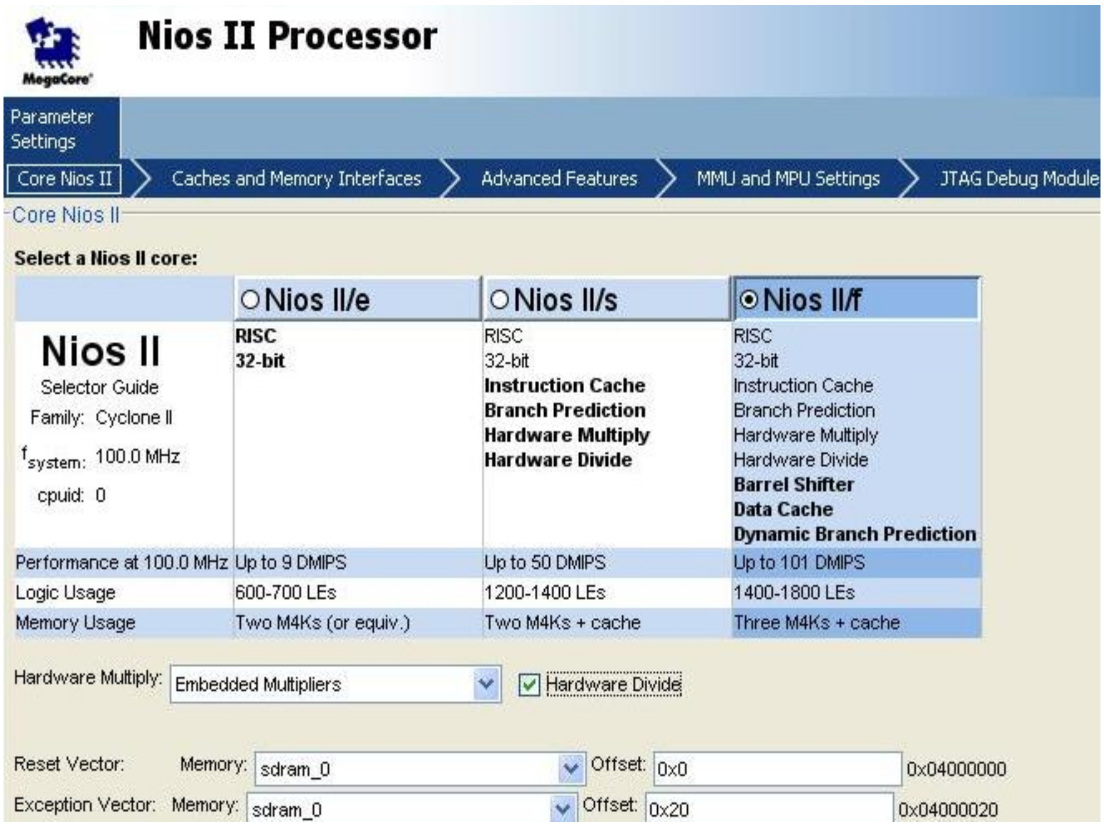
inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	2/1	0.00	50.00
c1	2/1	-108.00	50.00
c2	2/1	-108.00	50.00

Figure 5.31 Altera PLL output.

After adjusting clocks for the system, next step is to configure Nios II CPU. There are 3 types of configurable Nios II CPU for Alera FPGAs. These are Nios II/f (/f: fast) which is an optimized for the highest performance, Nios II/e (/e: economy) which is an optimized for smallest size and Nios II/s (/s: standard) which is balanced for performance and size. Nios II/f is selected for the system in this thesis. Embedded multipliers with hardware divide option is also selected. Note that, increasing the features of Nios II CPU, occupy more logic elements (LEs). For Nios II/f CPU 1400 - 1800 LEs are used. Reset vector and exception vector addresses must be determined in the design. In this system these vectors are relied on *sdram_0*. Figure 5.32 shows this configuration screen.

As mentioned in the introduction of Section 5.2, JTAG UART is used for serial configuration. JTAG UART core provides host access via JTAG pins on the FPGA. For time-based operations such as configuring watchdog timer or resetting the system in a pre-determined time are realized by interval timer block of SOPC Builder. JTAG UART and interval timer must be implemented on Nios II CPU designs. The settings of JTAG UART and interval timer is shown in Figure 5.33.



Nios II Processor

Parameter Settings

Core Nios II | Caches and Memory Interfaces | Advanced Features | MMU and MPU Settings | JTAG Debug Module

Core Nios II

Select a Nios II core:

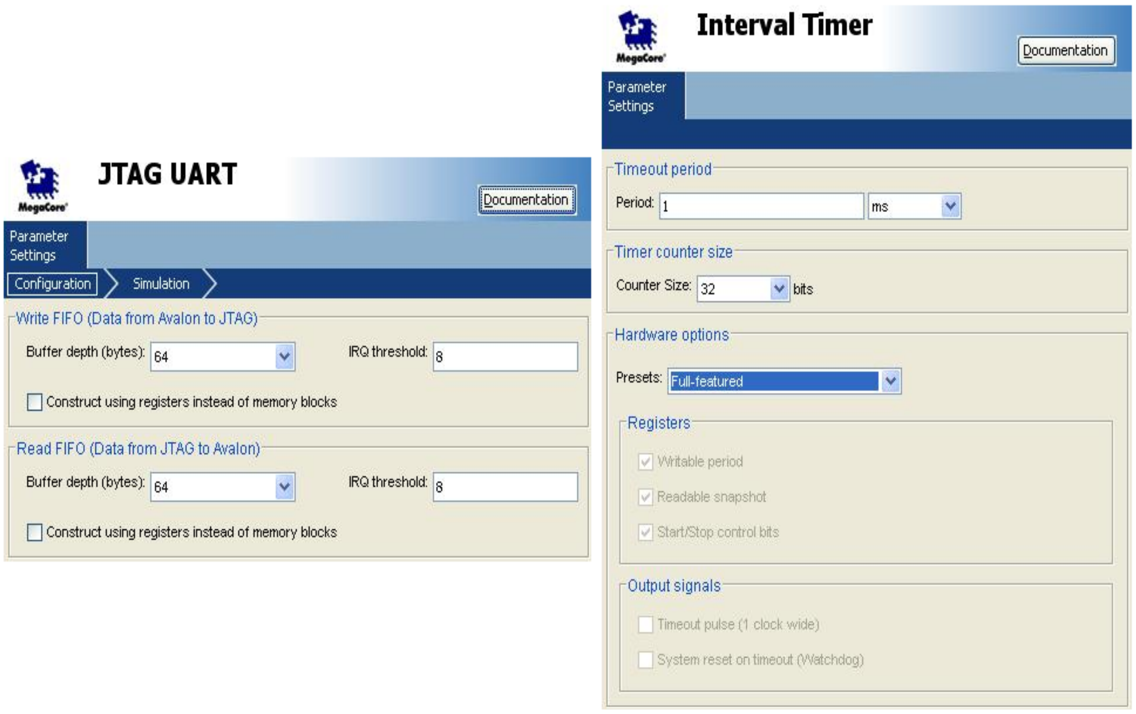
Nios II Selector Guide Family: Cyclone II f_{system}: 100.0 MHz cpuid: 0	○ Nios II/e	○ Nios II/s	● Nios II/f
	RISC **32-bit**	RISC 32-bit **Instruction Cache** **Branch Prediction** **Hardware Multiply** **Hardware Divide**	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide **Barrel Shifter** **Data Cache** **Dynamic Branch Prediction**
Performance at 100.0 MHz	Up to 9 DMIPS	Up to 50 DMIPS	Up to 101 DMIPS
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs
Memory Usage	Two M4Ks (or equiv.)	Two M4Ks + cache	Three M4Ks + cache

Hardware Multiply: Embedded Multipliers ☑ Hardware Divide

Reset Vector: Memory: sdram_0 Offset: 0x0 0x04000000

Exception Vector: Memory: sdram_0 Offset: 0x20 0x04000020

Figure 5.32 Nios II CPU configuration.



JTAG UART

Documentation

Parameter Settings

Configuration | Simulation

Write FIFO (Data from Avalon to JTAG)

Buffer depth (bytes): 64 IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Read FIFO (Data from JTAG to Avalon)

Buffer depth (bytes): 64 IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Interval Timer

Documentation

Parameter Settings

Timeout period

Period: 1 ms

Timer counter size

Counter Size: 32 bits

Hardware options

Presets: Full-featured

Registers

☑ Writable period

☑ Readable snapshot

☑ Start/Stop control bits

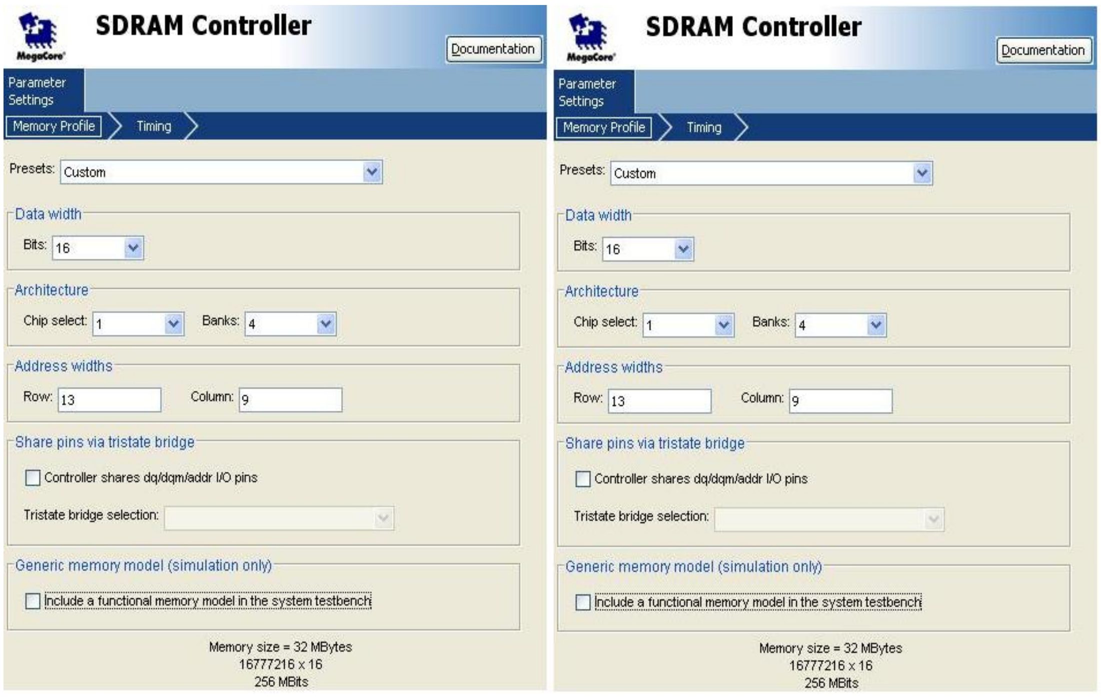
Output signals

☐ Timeout pulse (1 clock wide)

☐ System reset on timeout (Watchdog)

Figure 5.33 JTAG UART and interval timer configurations.

SDRAM Controllers, *sdram_0* and *sdram_1*, are configured as in the Figure 5.34. Data width is set to 16 bits and address widths are created by using 13 rows and 9 columns. The sizes of *sdram_0* and *sdram_1* are 32 MBytes (256 MBits) and totally 64 MBytes of SDRAM memory.



Presets: Custom

Data width
Bits: 16

Architecture
Chip select: 1 Banks: 4

Address widths
Row: 13 Column: 9

Share pins via tristate bridge
☐ Controller shares dq/dqm/addr I/O pins
Tristate bridge selection:

Generic memory model (simulation only)
☐ Include a functional memory model in the system testbench

Memory size = 32 MBytes
16777216 x 16
256 MBits

Presets: Custom

Data width
Bits: 16

Architecture
Chip select: 1 Banks: 4

Address widths
Row: 13 Column: 9

Share pins via tristate bridge
☐ Controller shares dq/dqm/addr I/O pins
Tristate bridge selection:

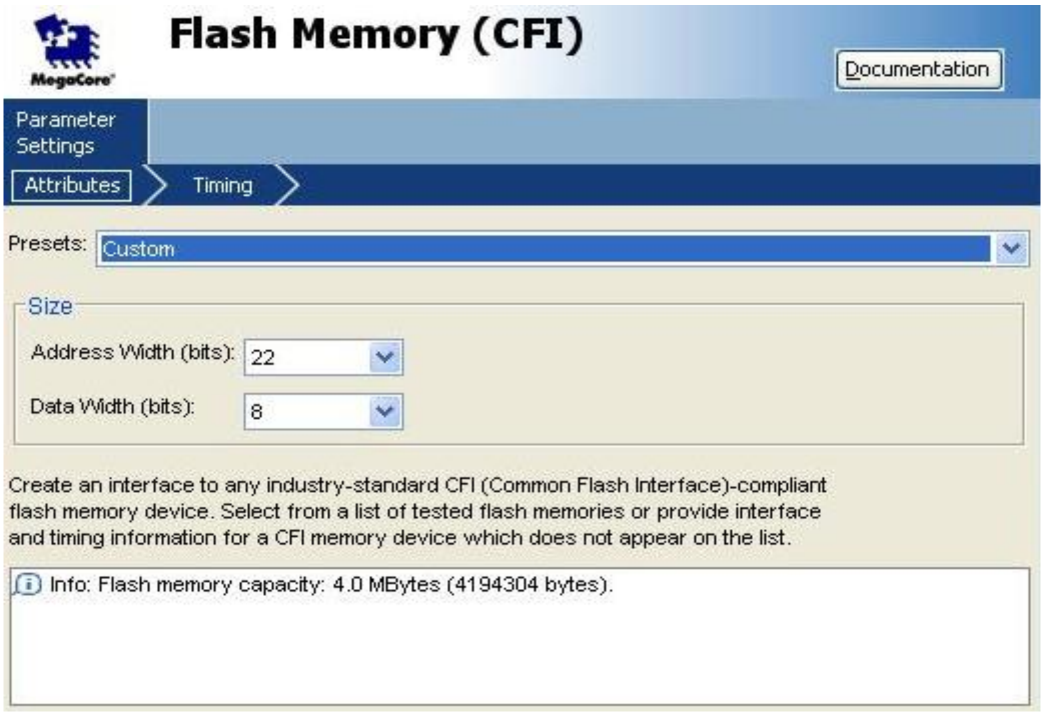
Generic memory model (simulation only)
☐ Include a functional memory model in the system testbench

Memory size = 32 MBytes
16777216 x 16
256 MBits

Figure 5.34 SDRAM controllers, sdram_0 and sdram_1, configurations.

The flash memory is used to store PCA basis matrix and projections of database to the PCA basis on this system. Flash memory is placed on behind of Avalon memory mapped tristate slave in SOPC Builder. 4 MBytes of flash memory by setting address width to 22 and data width to 8 is created in the system building environment that shown in Figure 5.35.

UART module allows communication with MATLAB like in the implementations on UP3 development kit. 115200 baud rate is used again. The other settings of UART is shown in Figure 5.36.


Flash Memory (CFI)

Documentation

Parameter Settings

Attributes | Timing

Presets: Custom

Size

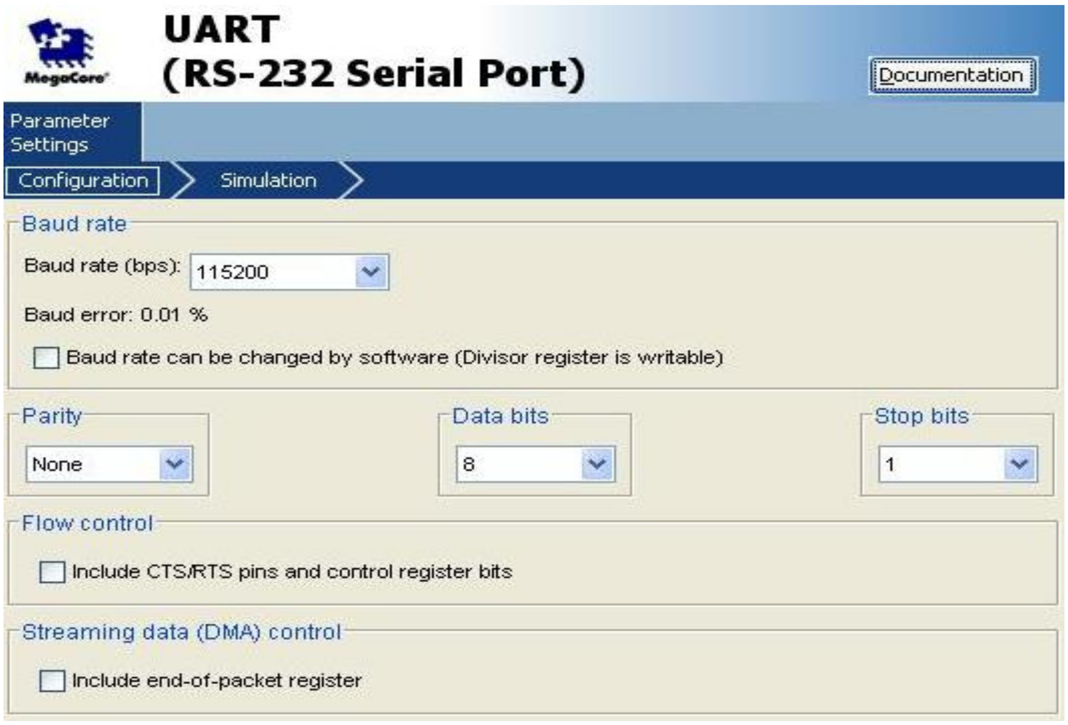
Address Width (bits): 22

Data Width (bits): 8

Create an interface to any industry-standard CFI (Common Flash Interface)-compliant flash memory device. Select from a list of tested flash memories or provide interface and timing information for a CFI memory device which does not appear on the list.

Info: Flash memory capacity: 4.0 MBytes (4194304 bytes).

Figure 5.35 Flash memory configuration.


UART (RS-232 Serial Port)

Documentation

Parameter Settings

Configuration | Simulation

Baud rate

Baud rate (bps): 115200

Baud error: 0.01 %

☐ Baud rate can be changed by software (Divisor register is writable)

Parity: None

Data bits: 8

Stop bits: 1

Flow control

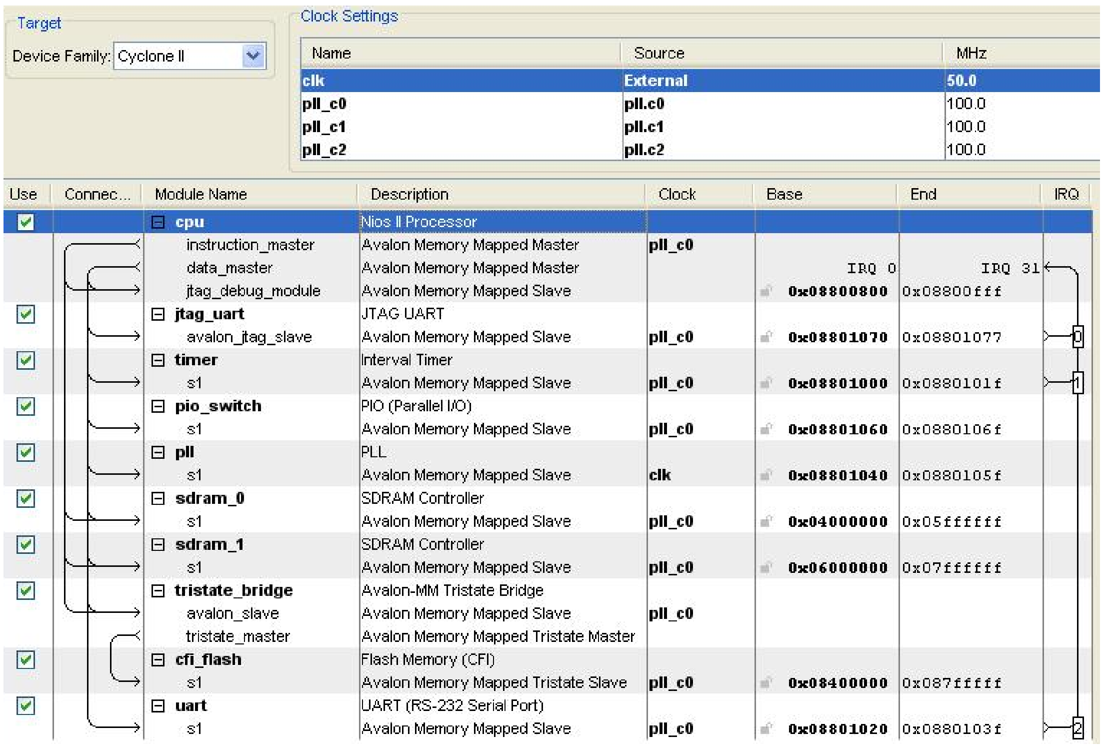
☐ Include CTS/RTS pins and control register bits

Streaming data (DMA) control

☐ Include end-of-packet register

Figure 5.36 UART configuration.

After configuring all components, base addresses and IRQ settings are arranged automatically from SOPC Builder settings. The last situation of SOPC Builder screen at the end of configuration and the address map is shown in Figure 5.37 and 5.38 respectively.



Target

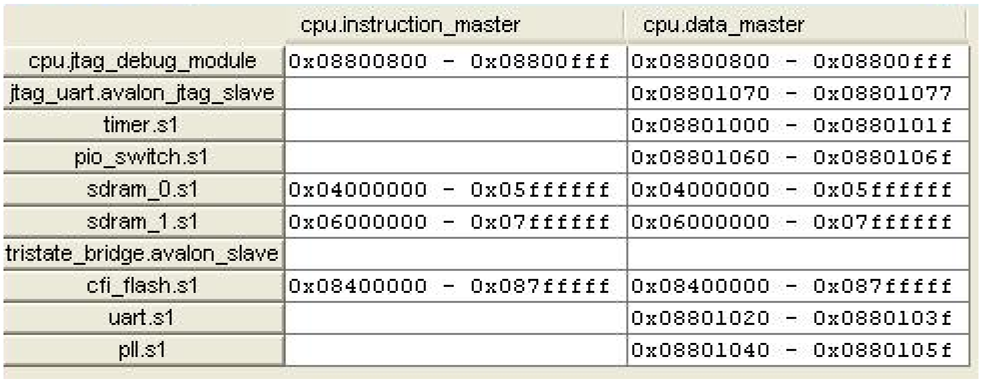
Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk	External	50.0
pll_c0	pll.c0	100.0
pll_c1	pll.c1	100.0
pll_c2	pll.c2	100.0

Use	Connec...	Module Name	Description	Clock	Base	End	IRQ
✓		cpu	Nios II Processor				
		instruction_master	Avalon Memory Mapped Master	pll_c0			
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31	
		jtag_debug_module	Avalon Memory Mapped Slave		0x08800800	0x08800fff	
✓		jtag_uart	JTAG UART				
		avalon_jtag_slave	Avalon Memory Mapped Slave	pll_c0	0x08801070	0x08801077	0
✓		timer	Interval Timer				
		s1	Avalon Memory Mapped Slave	pll_c0	0x08801000	0x0880101f	1
✓		pio_switch	PIO (Parallel I/O)				
		s1	Avalon Memory Mapped Slave	pll_c0	0x08801060	0x0880106f	
✓		pll	PLL				
		s1	Avalon Memory Mapped Slave	clk	0x08801040	0x0880105f	
✓		sdram_0	SDRAM Controller				
		s1	Avalon Memory Mapped Slave	pll_c0	0x04000000	0x05ffffff	
✓		sdram_1	SDRAM Controller				
		s1	Avalon Memory Mapped Slave	pll_c0	0x06000000	0x07ffffff	
✓		tristate_bridge	Avalon-MM Tristate Bridge				
		avalon_slave	Avalon Memory Mapped Slave	pll_c0			
		tristate_master	Avalon Memory Mapped Tristate Master				
✓		cfi_flash	Flash Memory (CFI)				
		s1	Avalon Memory Mapped Tristate Slave	pll_c0	0x08400000	0x087fffff	
✓		uart	UART (RS-232 Serial Port)				
		s1	Avalon Memory Mapped Slave	pll_c0	0x08801020	0x0880103f	2

Figure 5.37 SOPC Builder screen at the end of configuring components.



	cpu.instruction_master	cpu.data_master
cpu.jtag_debug_module	0x08800800 – 0x08800fff	0x08800800 – 0x08800fff
jtag_uart.avalon_jtag_slave		0x08801070 – 0x08801077
timer.s1		0x08801000 – 0x0880101f
pio_switch.s1		0x08801060 – 0x0880106f
sdram_0.s1	0x04000000 – 0x05ffffff	0x04000000 – 0x05ffffff
sdram_1.s1	0x06000000 – 0x07ffffff	0x06000000 – 0x07ffffff
tristate_bridge.avalon_slave		
cfi_flash.s1	0x08400000 – 0x087fffff	0x08400000 – 0x087fffff
uart.s1		0x08801020 – 0x0880103f
pll.s1		0x08801040 – 0x0880105f

Figure 5.38 Address map.

The system is generated after configuring all components by using “Generate” button of SOPC Builder. After generating the system without any error, pins that used on the board must be assigned. During system generation in SOPC Builder, system component of the design is also created with the name of system, *systop* in this case. The *systop* component are added to library of Quartus II schematic design like AND gate, or a multiplexer. This component is used to assign pins of the circuit. Figure 5.39 shows *systop* component.

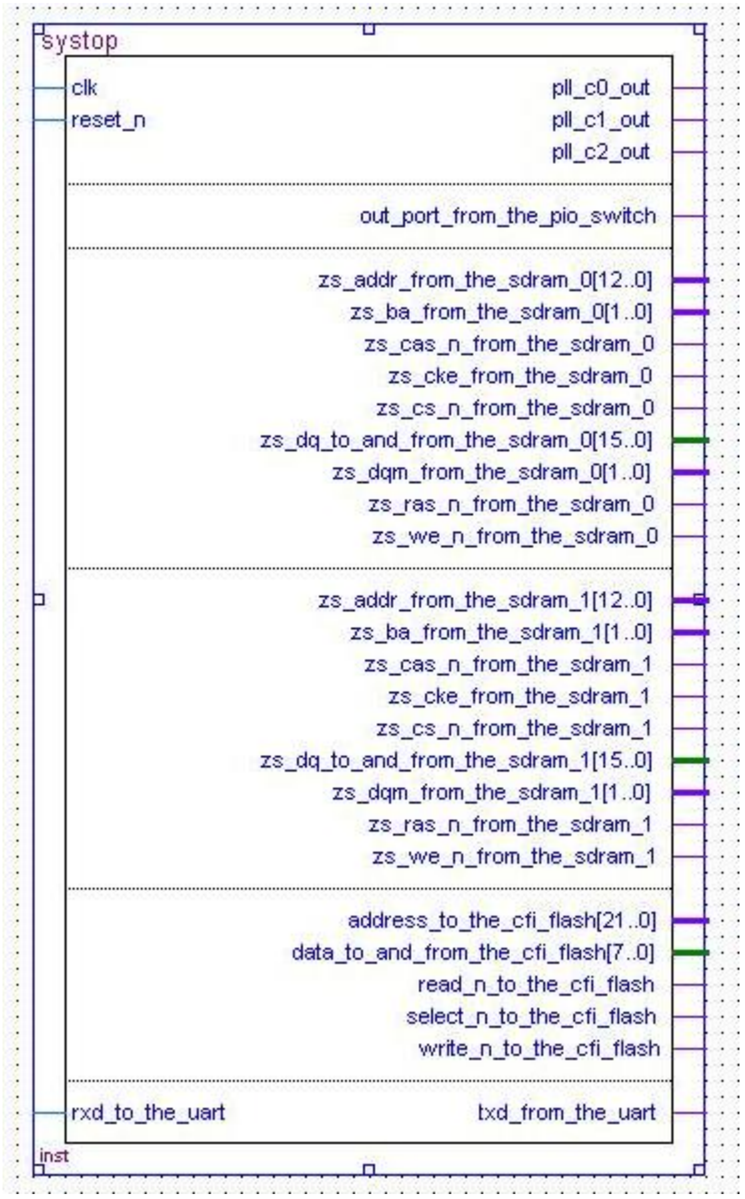

systop
clk
reset_n
pll_c0_out
pll_c1_out
pll_c2_out
out_port_from_the_pio_switch
zs_addr_from_the_sdram_0[12..0]
zs_ba_from_the_sdram_0[1..0]
zs_cas_n_from_the_sdram_0
zs_cke_from_the_sdram_0
zs_cs_n_from_the_sdram_0
zs_dq_to_and_from_the_sdram_0[15..0]
zs_dqm_from_the_sdram_0[1..0]
zs_ras_n_from_the_sdram_0
zs_we_n_from_the_sdram_0
zs_addr_from_the_sdram_1[12..0]
zs_ba_from_the_sdram_1[1..0]
zs_cas_n_from_the_sdram_1
zs_cke_from_the_sdram_1
zs_cs_n_from_the_sdram_1
zs_dq_to_and_from_the_sdram_1[15..0]
zs_dqm_from_the_sdram_1[1..0]
zs_ras_n_from_the_sdram_1
zs_we_n_from_the_sdram_1
address_to_the_cfi_flash[21..0]
data_to_and_from_the_cfi_flash[7..0]
read_n_to_the_cfi_flash
select_n_to_the_cfi_flash
write_n_to_the_cfi_flash
rxd_to_the_uart
txd_from_the_uart
inst

Figure 5.39 systop component.

The hardware design of this system is completed with pin assignments. Figure 5.40 shows the pin assignments of systop. Figure 5.41 and Figure 5.42 show the SDRAM and flash memory pin assignments respectively.

Note that, for global reset and flash memory reset signals, DIP switches are used. SW17 pin is assigned to global reset and SW1 pin is assigned to flash memory reset on DE2-70 development kit.

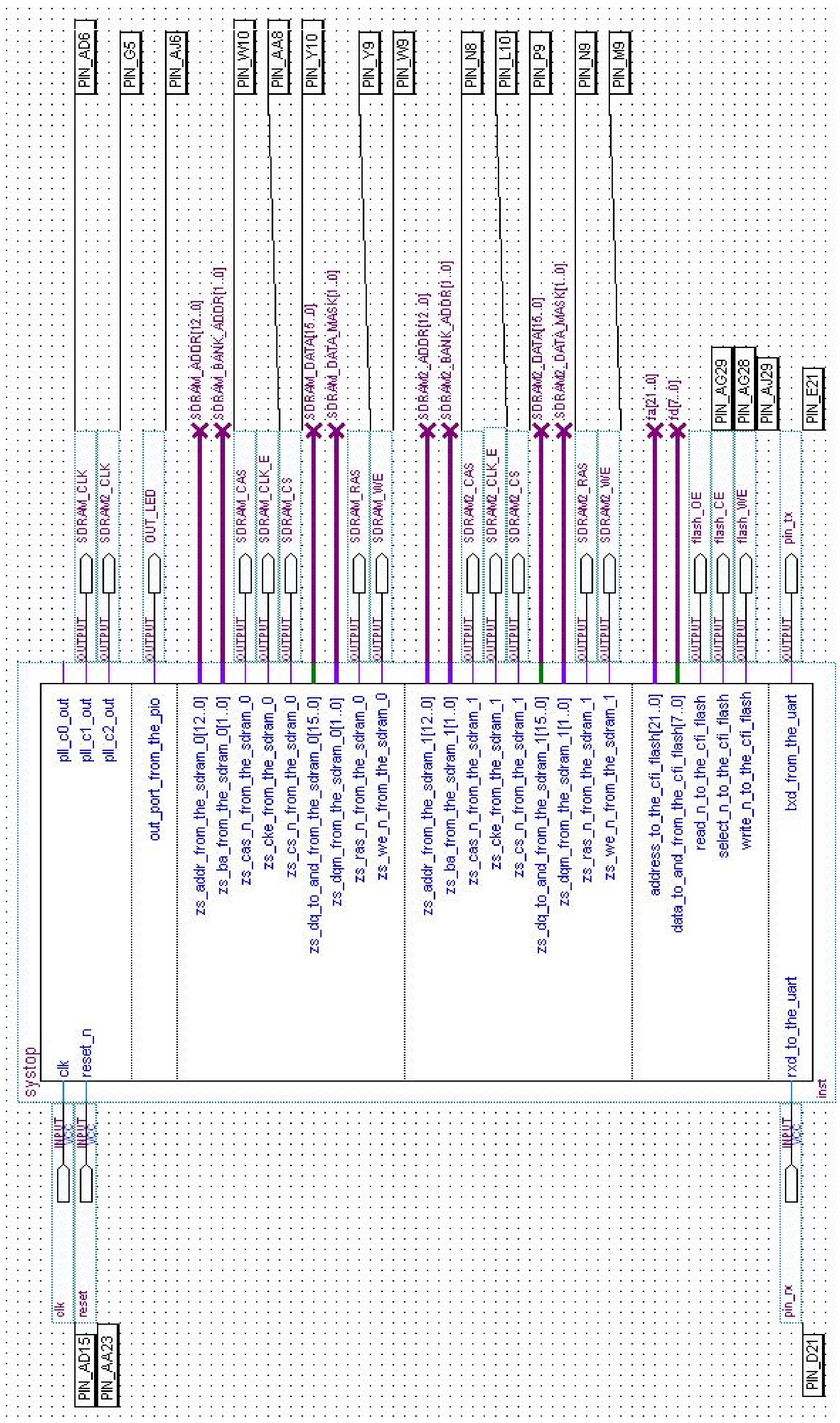


Figure 5.40 Pin assignments of systop component.

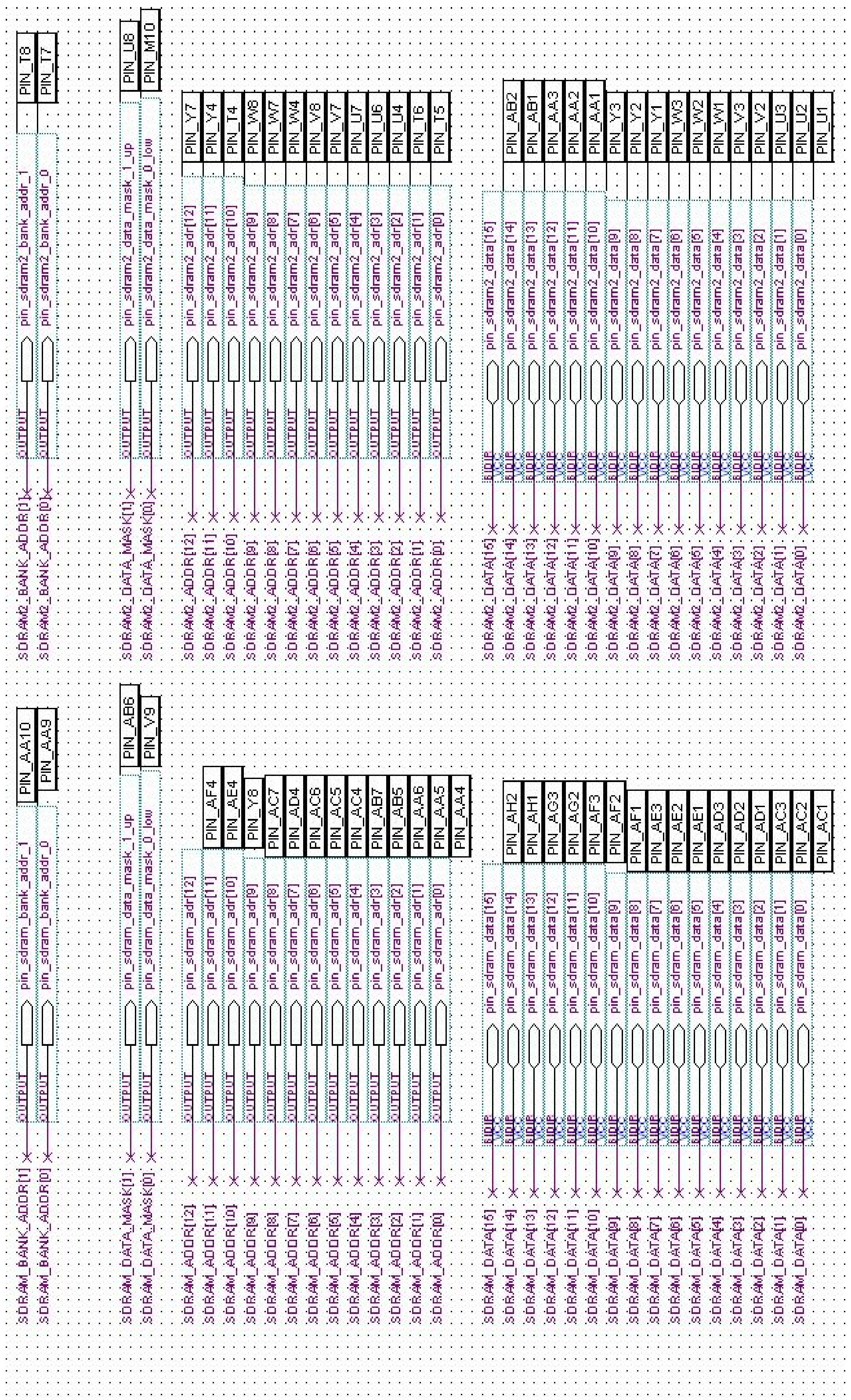

SDRAM2_BANK_ADDR[1] OUTPUT pin_sdram2_bank_addr_1 PIN_T8
SDRAM2_BANK_ADDR[0] OUTPUT pin_sdram2_bank_addr_0 PIN_T7
SDRAM2_DATA_MASK[1] OUTPUT pin_sdram2_data_mask_1_up PIN_U8
SDRAM2_DATA_MASK[0] OUTPUT pin_sdram2_data_mask_0_low PIN_M10
SDRAM2_ADDR[12] OUTPUT pin_sdram2_adr[12] PIN_Y7
SDRAM2_ADDR[11] OUTPUT pin_sdram2_adr[11] PIN_Y4
SDRAM2_ADDR[10] OUTPUT pin_sdram2_adr[10] PIN_T4
SDRAM2_ADDR[9] OUTPUT pin_sdram2_adr[9] PIN_W8
SDRAM2_ADDR[8] OUTPUT pin_sdram2_adr[8] PIN_W7
SDRAM2_ADDR[7] OUTPUT pin_sdram2_adr[7] PIN_W4
SDRAM2_ADDR[6] OUTPUT pin_sdram2_adr[6] PIN_V8
SDRAM2_ADDR[5] OUTPUT pin_sdram2_adr[5] PIN_V7
SDRAM2_ADDR[4] OUTPUT pin_sdram2_adr[4] PIN_U7
SDRAM2_ADDR[3] OUTPUT pin_sdram2_adr[3] PIN_U6
SDRAM2_ADDR[2] OUTPUT pin_sdram2_adr[2] PIN_U4
SDRAM2_ADDR[1] OUTPUT pin_sdram2_adr[1] PIN_T6
SDRAM2_ADDR[0] OUTPUT pin_sdram2_adr[0] PIN_T5
SDRAM2_DATA[15] BIDIR pin_sdram2_data[15] PIN_AB2
SDRAM2_DATA[14] BIDIR pin_sdram2_data[14] PIN_AB1
SDRAM2_DATA[13] BIDIR pin_sdram2_data[13] PIN_AA3
SDRAM2_DATA[12] BIDIR pin_sdram2_data[12] PIN_AA2
SDRAM2_DATA[11] BIDIR pin_sdram2_data[11] PIN_AA1
SDRAM2_DATA[10] BIDIR pin_sdram2_data[10] PIN_Y3
SDRAM2_DATA[9] BIDIR pin_sdram2_data[9] PIN_Y2
SDRAM2_DATA[8] BIDIR pin_sdram2_data[8] PIN_Y1
SDRAM2_DATA[7] BIDIR pin_sdram2_data[7] PIN_W3
SDRAM2_DATA[6] BIDIR pin_sdram2_data[6] PIN_W2
SDRAM2_DATA[5] BIDIR pin_sdram2_data[5] PIN_W1
SDRAM2_DATA[4] BIDIR pin_sdram2_data[4] PIN_V3
SDRAM2_DATA[3] BIDIR pin_sdram2_data[3] PIN_V2
SDRAM2_DATA[2] BIDIR pin_sdram2_data[2] PIN_U3
SDRAM2_DATA[1] BIDIR pin_sdram2_data[1] PIN_U2
SDRAM2_DATA[0] BIDIR pin_sdram2_data[0] PIN_U1
SDRAM_BANK_ADDR[1] OUTPUT pin_sdram_bank_addr_1 PIN_AA10
SDRAM_BANK_ADDR[0] OUTPUT pin_sdram_bank_addr_0 PIN_AA9
SDRAM_DATA_MASK[1] OUTPUT pin_sdram_data_mask_1_up PIN_AB6
SDRAM_DATA_MASK[0] OUTPUT pin_sdram_data_mask_0_low PIN_V9
SDRAM_ADDR[12] OUTPUT pin_sdram_adr[12] PIN_AF4
SDRAM_ADDR[11] OUTPUT pin_sdram_adr[11] PIN_AE4
SDRAM_ADDR[10] OUTPUT pin_sdram_adr[10] PIN_Y8
SDRAM_ADDR[9] OUTPUT pin_sdram_adr[9] PIN_AC7
SDRAM_ADDR[8] OUTPUT pin_sdram_adr[8] PIN_AD4
SDRAM_ADDR[7] OUTPUT pin_sdram_adr[7] PIN_AC6
SDRAM_ADDR[6] OUTPUT pin_sdram_adr[6] PIN_AC5
SDRAM_ADDR[5] OUTPUT pin_sdram_adr[5] PIN_AC4
SDRAM_ADDR[4] OUTPUT pin_sdram_adr[4] PIN_AB7
SDRAM_ADDR[3] OUTPUT pin_sdram_adr[3] PIN_AB5
SDRAM_ADDR[2] OUTPUT pin_sdram_adr[2] PIN_AA6
SDRAM_ADDR[1] OUTPUT pin_sdram_adr[1] PIN_AA5
SDRAM_ADDR[0] OUTPUT pin_sdram_adr[0] PIN_AA4
SDRAM_DATA[15] BIDIR pin_sdram_data[15] PIN_AH2
SDRAM_DATA[14] BIDIR pin_sdram_data[14] PIN_AH1
SDRAM_DATA[13] BIDIR pin_sdram_data[13] PIN_AG3
SDRAM_DATA[12] BIDIR pin_sdram_data[12] PIN_AG2
SDRAM_DATA[11] BIDIR pin_sdram_data[11] PIN_AF3
SDRAM_DATA[10] BIDIR pin_sdram_data[10] PIN_AF2
SDRAM_DATA[9] BIDIR pin_sdram_data[9] PIN_AF1
SDRAM_DATA[8] BIDIR pin_sdram_data[8] PIN_AE3
SDRAM_DATA[7] BIDIR pin_sdram_data[7] PIN_AE2
SDRAM_DATA[6] BIDIR pin_sdram_data[6] PIN_AE1
SDRAM_DATA[5] BIDIR pin_sdram_data[5] PIN_AD3
SDRAM_DATA[4] BIDIR pin_sdram_data[4] PIN_AD2
SDRAM_DATA[3] BIDIR pin_sdram_data[3] PIN_AD1
SDRAM_DATA[2] BIDIR pin_sdram_data[2] PIN_AC3
SDRAM_DATA[1] BIDIR pin_sdram_data[1] PIN_AC2
SDRAM_DATA[0] BIDIR pin_sdram_data[0] PIN_AC1

Figure 5.41 Pin assignments of SDRAM.

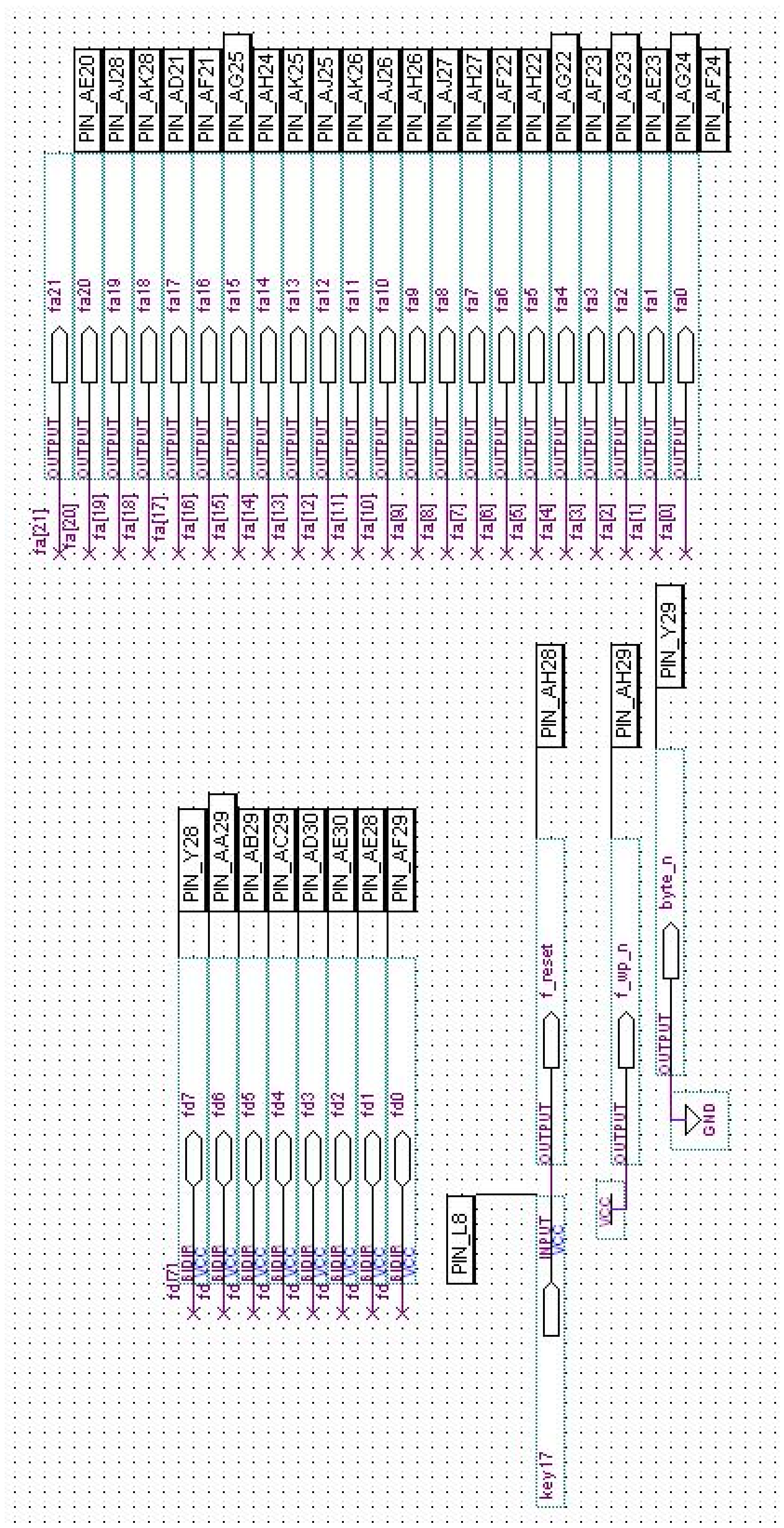


Figure 5.42 Pin assignments of flash memory.

5.2.2 Implementation of Face Recognition System on DE2-70

As mentioned in the introduction of Section 5.2, new implementation idea brings new design methods. According to this new idea, all operations excluding image taking and resizing, are implemented on the FPGA. Therefore, a new software mechanism is developed for the system that is designed in Section 5.2.1. The new software mechanism to configure FPGA is mentioned in Section 5.2.2.1 and the operations on MATLAB are described in Section 5.2.2.2. System designs with lower recognition rates that may be called as preliminary designs for face recognition, are introduced in Section 5.2.2.3. And the last section, Section 5.2.2.4, includes the final face recognition results that have the best recognition rate during this study.

5.2.2.1 Software Design on DE2-70

Nios II IDE is the environment of configuring FPGA by writing a high level language, C/C++. This tool has some useful features such as adding hardware and software breakpoints that are used for debugging the configuration software. This section introduces configuration software. The source code of this system is in the Appendix with the folder name of "5_2_2_1_Face_Recognition_Configuration_Sw".

The software for configuring the FPGA controls all of the parameters size by using two variables. Variable *KISI_SAY* states that the number of images that used for creating database and *FEATURE_SAY* states that the size of each image. For example *DATABASE_SIZE* equals to *KISI_SAY x FEATURE_SAY* and *TEST_SIZE* equals to *FEATURE_SAY*.

As already described in the previous sections that database and test images are received from MATLAB after resizing operation. UART Core that added to system during SOPC Builder design, is the module that allows to listen serial port and receive/transmit informations. To use UART core, two header files must be included: "altera_avalon_uart.h" that includes the UART device drivers and "altera_avalon_uart_regs.h" that includes the pre-defined status and control registers of UART. On the *main()* function, UART module is defined as a routine that serves when the serial port interrupt occurs. Figure 5.43 shows that the UART interrupt service routine that used in the system design. Note that, *UART_BASE* is the start address of UART that shown in Figure 5.38. When system is generated in SOPC Builder, this address is added to table in “system.h” file. In this service routine first the status register is controlled. If the receiver ready flag (*ALTERA_ AVALON_ UART_CONTROL_RRDY_MSK*) is set, UART is ready to receive data. *RxHeadData* shows the buffer assigned for the database. Receiving bytes and storing them to *RxHeadData* is continued until all of the database elements are sent. If the pointer of the buffer shows the exact number with database, the new received bytes are interpreted as the elements of test array and stored in *RxTest*. This approach is followed since the database and test features are sent respectively from MATLAB.

```
void uart_isr(void* context,alt_u32 id)
{
alt_u32 status;

status = IORD_ALTERA_AVALON_UART_STATUS(UART_BASE);

if(status & ALTERA_AVALON_UART_CONTROL_RRDY_MSK)
{

 if(RxHeadData < DATABASE_SIZE)
 {
 RxDataBase[RxHeadData]=IORD_ALTERA_AVALON_UART_RXDATA(UART_BASE);
 IOWR_ALTERA_AVALON_UART_STATUS(UART_BASE,0);

  if((++RxHeadData) > (DATABASE_SIZE - 1))
  {
  //
  }
    }else{
   RxTest[RxHeadTest] = IORD_ALTERA_AVALON_UART_RXDATA(UART_BASE);
   IOWR_ALTERA_AVALON_UART_STATUS(UART_BASE,0);

        if((++RxHeadTest) > TEST_SIZE - 1)
        {
          //RxHeadTest = 0;
        }
    }
 }
}
```

Figure 5.43 UART interrupt service routine.

Offline training and online test steps are used along the implementation of this thesis. The implementation of the configuration file for FPGA, is also followed these two steps. Note that previously described UART implementation is an interrupt service routine, so it is called in both offline training and online test.

Offline training is started with programming FPGA. Serial configuration file that holds the hardware and software implementation of the project is programmed via JTAG. “data gelmiyor...” print message shows that database is not created on the system before. If this moment, the vectorized database is sent from MATLAB, FPGA receives this database and save this information on an unsigned char *TempMat* array. The size of this array is *KISI_SAY x FEATURE_SAY*. Next step is to arrange this array as a matrix with *KISI_SAY* coloumns and *FEATURE_SAY* rows.

data and *data2* matrices hold the resized images like in MATLAB. *data* is overwritten in PCA algorithm, so another matrix, *data2*, is also created to keep original input array. Note that, *vector()* and *matrix()* functions allocate a float vector and matrix respectively. Similarly, *free_vector()* and *free_matrix()* functions are used to deallocate the memory regions.

PCA algorithm is implemented with a similar way to Section 5.1.3. The algorithm start with taking covariance of database images. Covariance of database is calculated by using *covcol(). symmat* is returned from this function. Then *tred2()* function, which is called as householder function, started to produce real and symmetric tridiagonal matrix. This process is called as triangular decomposition. After triangular decomposition step, this symmetric tridiagonal matrix is reduced in *tqli()*. After *tqli(), evals* contains the eigenvalues and the coloumns of *symmat* contain the associated eigenvectors. Size of *symmat* is *FEATURE_SAY x FEATURE_SAY*, so the result of this multiplication is enormous when the features are from an image. As noted in Section 5.1.3 that after sorting the eigenvectors from higher to lower, selecting 3 eigenvectors are sufficient for successful recognition rates in MATLAB implementation.. As also noted that this approach is used on further system designs which are impelemented on FPGA. To select 3 eigenvectors, *symmat2* is generated from *symmat* matrix. This matrix, *symmat2*, can be called as a PCA basis matrix, because this matrix is used to project database and test features to the face space. *ytrain* is the projection of database features to the face space. The size of *ytrain* is *FEATURE_SAY x 3*..The offline training part ends with storing, the PCA basis matrix (*symmat2*) and projection of database to the face space (*ytrain*), to the flash memory. These matrices are formed as a vector and the all elements of the vector are converted from float to char before writing data to flash memory by using *ConvertAllAndWrite2FlashAsChar()*. This conversion is added to prevent errors during writing and reading from flash memory. *ytrain* is written to 0x10000 address, and *symmat2* is written to 0x60000 address. To make sure that the offline training is run once, control mechanism that is shown in the Figure 5.44 is added. 0xFF or 255 is the value of empty bytes and if one of these offsets have different values from 0xFF, this shows that the offline training has already finished, so the code is routed to *TEST_FONKSIYONU*. Note that *FlashTestYtrain* and *FlashTestPcaBasis* are 2 bytes of test data that readed from flash memory.

```
.
..
if ((FlashTestYtrain[0]!=255)||(FlashTestYtrain[1]!=255)
   ||(FlashTestPcaBasis[0]!=255)||(FlashTestPcaBasis[1]!=255))

      {
        printf ("flash does not empty \n");
        goto TEST_FONKSIYONU;

      }else
     {
        printf ("flash empty \n");
       while (1)
      {
        printf ("data gelmiyor...\n");
        ..
        .
       }
```

Figure 5.44 Offline training or online test are selected according to flash memory contents.

Online test step is started after the creation of database or it can be started after power-on if the flash memory is written before. “testi yolla...” print message shows that system is ready to get test data from MATLAB. After, test is sent from MATLAB, it is stored in unsigned char *TempMat2* array. The size of the array is

FEATURE_SAY. After the reception of test is completed, a new float *test* matrix is created. The reason of creating such a float type matrix because the numbers that are evaluated at offline training step are float type. Even if they are written to flash as char type, the actual value of them is identified with float type. So, communication and flash read/write functions use unsigned char type variables, but operations such as triangular decomposition or matrix multiplication need float type variables. As mentioned in Section 5.1.3, the projections of database and test to the face space must be compared to identify the correct people. From offline training part, projections of database features are stored in flash memory with PCA basis matrix. So, at this step, flash content must be restored. *RestoreAllFromFlash()* function restores the array, that is started the offset address that passed to this function, from flash memory. *FlashYtrain* and *FlashPcaBasis* are the restored, then the data types are changed from unsigned char to float. *NewYtrain* and *NewPcaBasis* are created after this data type change. Note that, *NewYtrain* and *NewPcaBasis* corresponds and holds same values with *ytrain* and *symmat2* respectively, when comparing with offline training phase. Online test phase ends with comparing the database and test projections. *toplam[]* holds the sum of absolute distances for each image. The minimum value of *toplam[]* shows that the nearest image in the database. The group number of this image is the recognition result of this system.

5.2.2.2 Implementation on MATLAB

After moving PCA algorithm to FPGA when getting the new FPGA development kit, DE2-70, MATLAB implementation is also changed. Now, MATLAB is only responsible for getting and resizing images. ORL Database is used again for verifying and testing the implementation.

In this section, an example of image preprocessing steps for one face image are shown. As mentioned in Section 5.1.3, the size of the face image in ORL Database is 112x92. First, the input image from ORL Database is taken from image database. Then *imresize*, that is also one of the funtions of MATLAB image processing toolbox, is used to resize the images. Figure 5.45.a shows the input image, Figure

5.45.b shows the resized image from 112x92 to 40x40. Note that, the process that is shown in Figure 5.45 is applied to all images sequentially on the database.

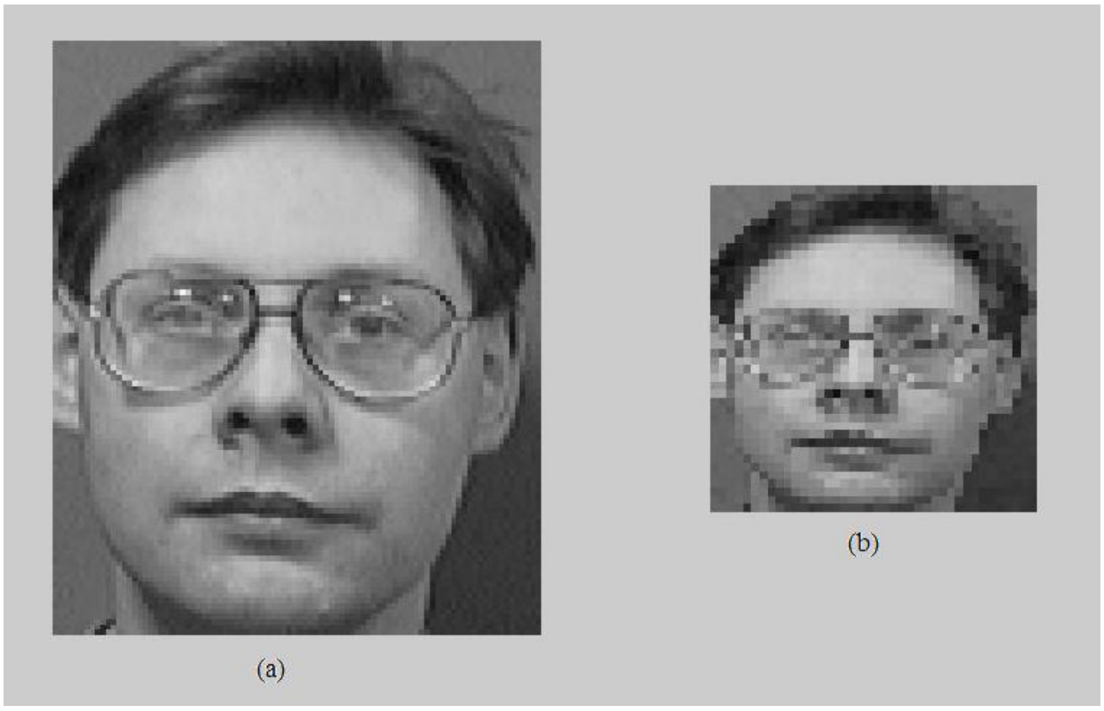


Figure 5.45 a) Input image b) Resized image.

The size of the image after resize operation, using *fdmex* algorithm and windowing the image are the experiments on the system for finding the best recognition rate. These are described with more detail in Section 5.2.2.3 and Section 5.2.2.4.

5.2.2.3 Preliminary Experiments

The first approach on the implementation is to use same approach that is described on Section 5.1.3. The working steps of the face recognition system is described for the first experiment, to better understand the process with showing the size of matrices. Other experiments are also followed the same instructions, but the size of matrices differ for each design. 48 face images of 6 people, with 8 face images with different poses belongs to same person , are used for creating database. The remaining 2 face images from each people are used for test phase. The size of database and test matrices are, after resizing the images from 112x92 to 48x48, 2304x48 and 2304x1 respectively. Note that, the rows represent pixels in the image and the columns represent the image number. To collect all data from UART module of the FPGA design, *KISI_SAY* is modified as 48 and *FEATURE_SAY* is modified as 2304. The database matrix that is created on FPGA, *data*, is the transpose of the database matrix that is created on MATLAB with size of 48x2304. PCA basis matrix, *symmat*, is created from data with the size of 2304x2304. After sorting eigenvalues from heighest to lowest, 3 of them is selected and stored in *symmat2* array. Face space projections, *ytrain*, can be calculated as by multiplying *data* and *symmat2*. Size of *ytrain* and *symmat2* are 48x3 and 2304x3 respectively and they are stored in flash memory to be used in online test phase. At the test phase, after restoring *ytrain* and *symmat2* from flash memory, the received test matrix, *test* with the size of 1x2304, is projected into to face space by multipliying *symmat2* then compared to *ytrain*. The comparison result gives the recognized person.

The performance of the system is determined by calculating the recognition rate. But, for the system in first experiment, the recognition rate have not been calculated. Because of the size of matrices are too big (for example: *symmat* size is 2304x2304), the computation and calculation with these float numbers take very long time. For this case, the generation of the offline training step takes nearly two and a half days. The unacceptability of this time, provide testing different system implementations instead of trying to calculate recognition rate.

First approach shows that, resizing image from 112x92 to 48x48 is not enough, because computation time is still too high. To reduce computation time, the first idea is to further decrease the image sizes. 15x15, 20x20, 30x30 and 40x40 sizes are tested. As expected from above, 15x15 image has 225 pixels so it is compiled quickly but recognition rate is nearly %40. During these tests, it has also been noticed that after the images largerthan 18x18 pixels, unexpected and unpredicted errors are seen on FPGA. So, for following tests, database size is created under 400 elements. With the new conditions, acquiring high recognition rate is more difficult. Two test systems are created to test the performance of the resizing.

In Table 5.5 and Table 5.6, recognition tables for two systems are shown. In these tables the rows represent the people for which the system is trained for. The columns show if the test image recognized for this recognition correctly or not. Table 5.5 shows "System-A" with face images of 5 people and 8 images from each person. In this table since that the first 8 images are used for training, the system is tested with 9th and 10th images, "+" and "-" signs show whether these images for these 5 people recognized or not, respectively. Table 5.6 shows "System-B" with with face images of 5 people and 6 images from each person. The recognition rates of System-A and System-B are %60 and %55 respectively.

Table 5.5 Recognition table for System-A.

Recognition Table		test image	
		9th	10th
person number	1st	+	+
	2nd	-	+
	3rd	+	-
	4th	+	-
	5th	-	+

Table 5.6 Recognition table for System-B.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	-	+	-	+
	2nd	+	-	+	+
	3rd	-	-	-	+
	4th	+	+	+	-
	5th	-	+	+	-

Second approach is based on *fdmex* function. After encountering problems on image resizing, some other algorithms are researched over the internet and one of them, *fdmex*, is selected. *fdmex* is a dynamic link library file that used in OpenCV. This function can be worked with C/C++ or MATLAB. The reason of using *fdmex* is decreasing the image size by cropping the faces of each images by calling a function only. Figure 5.46.a shows an input image, 5.46.b shows the cropped face image by *fdmex*. Then cropped image is resized to 15x15. Figure 5.46.c shows the resized cropped image.

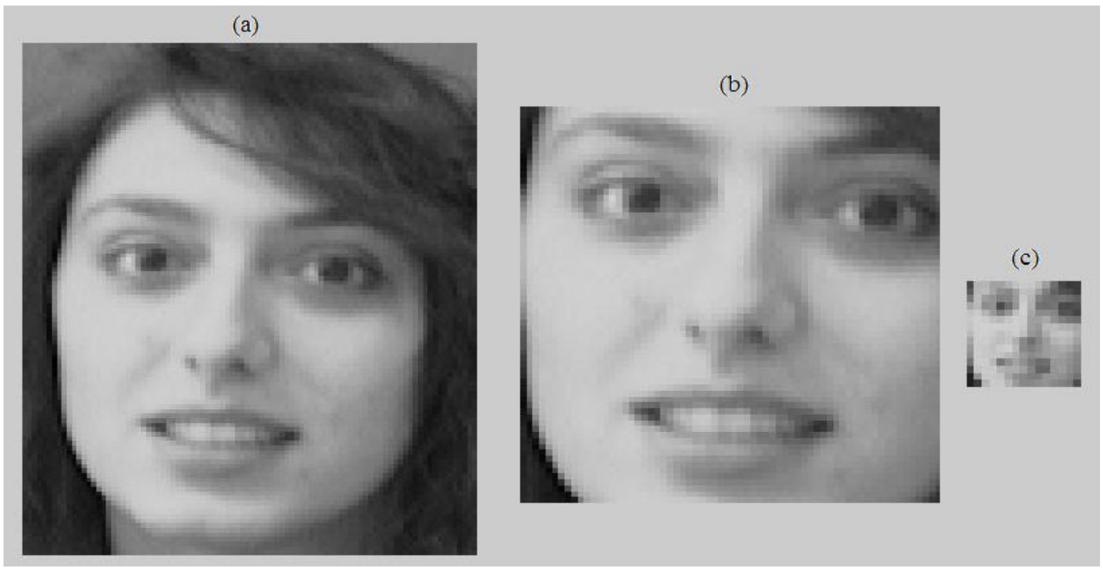


Figure 5.46 a) Input image b) Cropped image by fdmex. c) Cropped image is resized to 15x15.

In Table 5.7 and Table 5.8, recognition tables for two system that used fdmex are shown. Table 5.7 shows "System-C" with face images of 5 people and 7 images from each person, Table 5.8 shows "System-D" with face images of 5 people and 6 images from each person. The recognition rates of System-C and System-D are %50 and %47 respectively. However, it has been observed that *fdmex* algorithm does not crop faces successfully for each image which lowers the recognition rate. Therefore, *fdmex* has not been used in the later stages. The source code of this system is in the Appendix with the folder name of "5_2_2_3_fdmex_example".

Table 5.7 Recognition table for System-C.

Recognition Table		test image		
		8th	9th	10th
person number	1st	+	-	+
	2nd	-	+	+
	3rd	-	-	+
	4th	+	+	-
	5th	+	+	-

Table 5.8 Recognition table for System-D.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	-	-	-
	2nd	-	+	+	-
	3rd	-	+	-	-
	4th	+	+	-	-
	5th	-	+	-	+

Third approach is developed after failing with the previous two approaches. As mentioned before, a natural threshold for the image size according to test results is seen as 15x15. "If this size is used a region of an image instead of using for the total size of an image, maybe the recognition rate increases" is the start idea of creating a new test subsystem. It is thought that, image size after resize operation is 30x30, then image are divided into 4 regions. The size of each region are 15x15. Figure 5.47.a shows the input image, 5.47.b shows the resized image to 30x30 and 5.47.c shows the regions on the image.

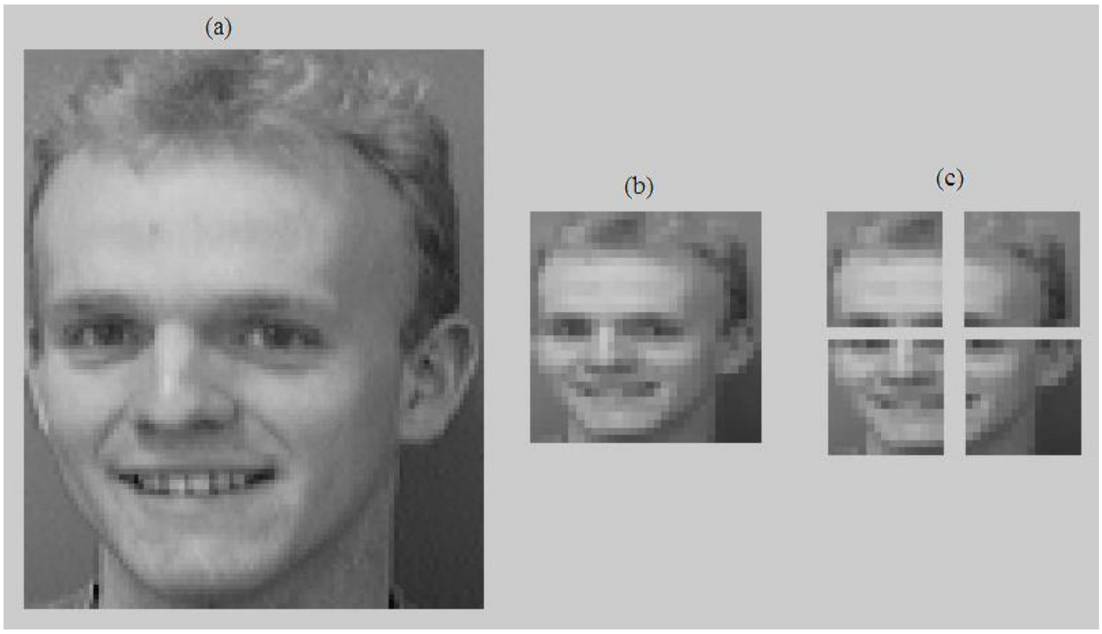


Figure 5.47 a) Input image b) Resized image (30x30) c) Four regions of 30x30 image.

Table 5.9, Table 5.10, Table 5.11 and Table 5.12 show the recognition results of 4 regions for "System-E". Region 1, Region 2, Region 3 and Region 4 has the recognition rate of %80, %55, %40 and %50 respectively.

Table 5.9 Recognition table for Region 1 of System-E.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	+
	2nd	+	+	+	+
	3rd	+	-	-	+
	4th	+	+	+	-
	5th	+	+	-	+

Table 5.10 Recognition table for Region 2 of System-E.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	-
	2nd	-	+	+	-
	3rd	-	+	+	+
	4th	+	-	+	-
	5th	-	-	-	+

Table 5.11 Recognition table for Region 3 of System-E.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	-	-	-	-
	2nd	-	-	+	+
	3rd	+	+	-	+
	4th	+	-	-	-
	5th	-	-	+	+

Even if the recognition rate is higher in the Region 1, the recognition rates of the other regions are unacceptable. Different 5 images are used for creating database, but it is seen that the results differ from this system and the recognition rate for this approach is dependent to the database images.

Table 5.12 Recognition table for Region 4 of System-E.

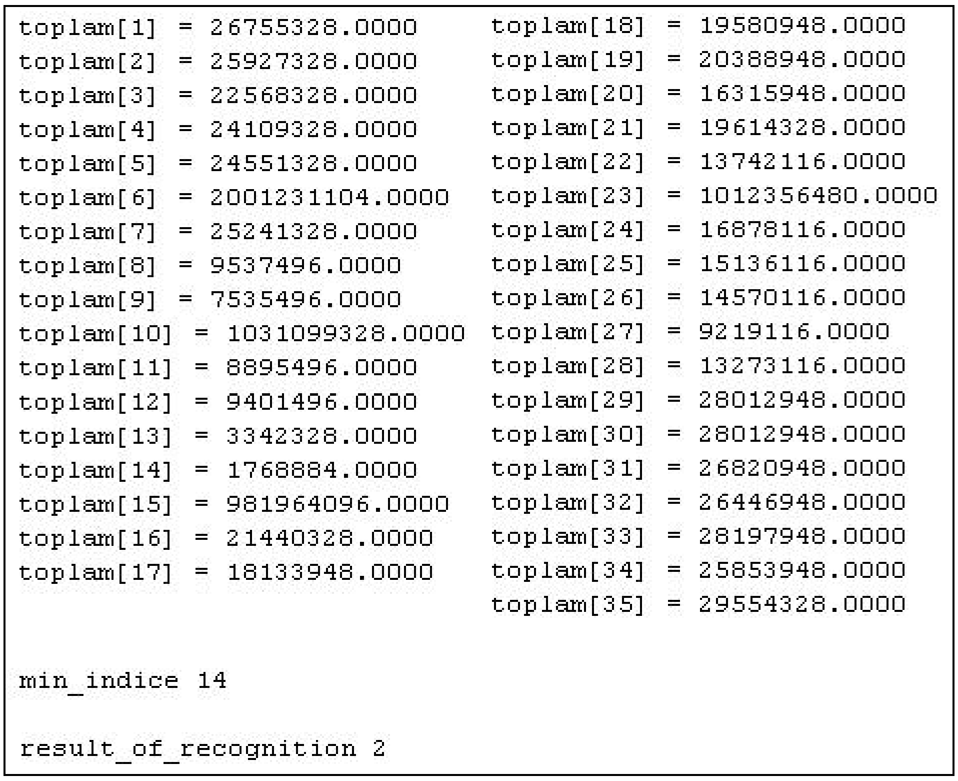
Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	+
	2nd	-	-	-	-
	3rd	+	+	-	+
	4th	+	+	-	-
	5th	-	+	-	-

5.2.2.4 Final Implementation

The difference of recognition rates between the regions that is described in the region based approach are led us to change implementation method again. And finally, the local windowing on the face image is tested. At this approach, first the input image is resized to 40x40. At previous implementations, if a size of 40x40 image is sent directly to FPGA, computation takes a very long time. After resizing the image to 40x40, 4x4 windows are created from this image. Instead of sending the pixel value directly to FPGA, the mean of these 4x4 windows are sent. So, for each image, local mean matrix with the size of 13x13 are extracted from resized image by using 4x4 local windowing approach. With this approach 169 (13x13) elements are sent for one image instead of sending 1600 (40x40) elements.

For creating the database, face images of 5 people and 7 images from each person, are taken to create a new test system, “System-F”. Since 35 face images are used for database creation, the size of feature matrix, which is calculating by taking mean of 4x4 local windows for each image, is 169x35. So, the variables that holds the image size, *KISI_SAY*, and feature size, *FEATURE_SAY* are set to 35 and 169 respectively. With these settings, FPGA implementation is compiled again by using the same instructions in Section 5.2.2.1. Note that, the maximum size is allocated for symmat matrix with 169x169 in this design. As comparing this size with the previous experiments, computation time of database is faster than before.

When database creation is completed after writing *ytrain* and *symmat2* to the flash memory, test image is asked from the system. When the size of 13x13 local mean matrix for test image is received from the system, it is projected to PCA basis space by multipliying *NewFlashPcaBasis*, then compared to *NewFlashYtrain*. *NewFlashPcaBasis* and *NewFlashYtrain* are the matrices that restored from flash and respects to *symmat2* and *ytrain* matrices on offline training respectively. The absolute distances from this comparison is stored on *toplam* vector. Note that, the size of *toplam* vector is same with *KISI_SAY*. After the minimum of *toplam* vector is computed, the group number, *result_of_recognition* that identifies the recognition result is assigned and printed to the console. The source code of this system is in the Appendix with the folder name of "5_2_2_4_Highest_Recognition_Rate_For_Face_ Recognition".



```
toplam[1] = 26755328.0000      toplam[18] = 19580948.0000
toplam[2] = 25927328.0000      toplam[19] = 20388948.0000
toplam[3] = 22568328.0000      toplam[20] = 16315948.0000
toplam[4] = 24109328.0000      toplam[21] = 19614328.0000
toplam[5] = 24551328.0000      toplam[22] = 13742116.0000
toplam[6] = 2001231104.0000    toplam[23] = 1012356480.0000
toplam[7] = 25241328.0000      toplam[24] = 16878116.0000
toplam[8] = 9537496.0000       toplam[25] = 15136116.0000
toplam[9] = 7535496.0000       toplam[26] = 14570116.0000
toplam[10] = 1031099328.0000   toplam[27] = 9219116.0000
toplam[11] = 8895496.0000      toplam[28] = 13273116.0000
toplam[12] = 9401496.0000      toplam[29] = 28012948.0000
toplam[13] = 3342328.0000      toplam[30] = 28012948.0000
toplam[14] = 1768884.0000      toplam[31] = 26820948.0000
toplam[15] = 981964096.0000    toplam[32] = 26446948.0000
toplam[16] = 21440328.0000     toplam[33] = 28197948.0000
toplam[17] = 18133948.0000     toplam[34] = 25853948.0000
                               toplam[35] = 29554328.0000

min_indice 14

result_of_recognition 2
```

Figure 5.48 Output of System-F.

Figure 5.48 shows an example of the output of System-F. The 8th image of second person is tested. Minimum element of *toplam* is stated by *min_indice*, 14, shows that the closest image in the database. Result of recognition is shown as 2 which is a correct result.

Table 5.13 shows that the recognition table for System-F. Recognition fails for one image so the recognition rate for this system is %93.3.

Table 5.13 Recognition table for System-F.

Recognition Table		test image		
		8th	9th	10th
person number	1st	+	+	+
	2nd	+	+	+
	3rd	+	+	+
	4th	+	+	+
	5th	-	+	+

Another test subsystem is created by decreasing the image numbers that are taken from each person to 6. Database of “System-G” is created by taking face images of 5 people and 6 images from each person. Note that System-F and System-G are created with same people. Only the size of database is decreased. Table 5.14 shows that the recognition table for System-G. Recognition fails for three images, so the recognition rate for this system is %85.

Table 5.14 Recognition table for System-G.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	+
	2nd	+	+	+	+
	3rd	+	+	+	+
	4th	+	-	-	-
	5th	+	+	+	+

To make sure about the recognition rate, the images of different 5 people is used to create database. Like in the above test systems, first system is created by taking face images of 5 people and 7 images from each person, System-H; second system is created by taking face images of 5 people and 6 images from each person, System-J. The recognition tables of System-H and System-J is shown in Table 5.15 and Table 5.16 respectively.

Table 5.15 Recognition table for System-H.

Recognition Table		test image		
		8th	9th	10th
person number	1st	+	+	+
	2nd	+	+	+
	3rd	+	+	+
	4th	+	-	+
	5th	+	+	+

Table 5.16 Recognition table for System-J.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	+
	2nd	+	+	+	+
	3rd	+	+	+	+
	4th	+	+	-	+
	5th	+	-	+	+

From Table 5.15 and Table 5.16, recognition rates can be extracted. The recognition rate for System-H is %93.3 and the recognition rate for System-J is %90.

When comparing the last four tables; Table 5.13, Table 5.14, Table 5.15 and Table 5.16; the minimum recognition rate is %85 and the maximum recognition rate is %93.3. These results are the highest recognition rates when compared to previous implementations and recognition rates are acceptable for the implementation of a face recognition system.

5.2.2.5 General Performance of the Face Recognition System

The general performance and the total accuracy of the face recognition system are found by constructing a confusion matrix. Confusion matrix is typically called a matching matrix and shows the matching rate for each person. Each row of the confusion matrix represents the instances in a predicted class, while each column of the confusion matrix represents the instances in an actual class.

In the previous recognition tables, database images were selected from 5 people with the face images from 1st to 6th according to original ORL Database for each person. The rest of the face images were used as test images. But, to construct a confusion matrix, different face images must be selected for database and test without changing the percentage of database images (%60) and test images (%40) to total face images. Database and test images are selected by using cross-validation technique which is used for estimating the performance of a predictive model.

Cross-validation algorithm is implemented in MATLAB by using *crossvalind* function from Bioinformatics Toolbox of MATLAB. The source code of this system is in the Appendix with the folder name of "5_2_2_5_Cross_Validation". The output of the algorithm is a matrix and each row represents the images and each column represents the number of test subsytem. The elements of the cross-validation matrix are 0's and 1's. 1's show that the corresponding image must be selected for database an and similarly 0's show that the corresponding image must be selected as test image.

25 test subsystems are created by using cross-validation technique and recognition tables are found for each subsystems. Confusion matrices are constructed by combining the all results of these recognition tables. Table 5.17 shows the general performance of the face recognition system on the confusion matrix. Note that, the values on this table are shown with the percent (%).

Table 5.17 Results of the face recognition system on the confusion matrix.

Confusion Matrix	1st	2nd	3rd	4th	5th
1st	100	0	0	0	0
2nd	0	100	0	0	0
3rd	1	0	97	2	0
4th	0	0	12	88	0
5th	0	0	0	0	100

From Table 5.17, it can be seen that the images of 1st, 2nd and 5th person are recognized without any error. Even if the %1 and %2 images of the 3rd person are recognized as 1st and 4th person respectively, %97 images of the 3rd person are recognized successfully. The lowest performance of the face recognition system is seen on the 4th person with an error rate of %12. The reason of the lowest performance for this case is the pose variance in the images of the 4th person.

The total accuracy of the face recognition system can be found by dividing the number of true recognitions to the number of the all images. For this system, the total accuracy is %97. This total accuracy value is also allowed to implement the face recognition system as a part of multibiometric recognition system.

5.2.3 Implementation of Fingerprint Recognition System on DE2-70

In the under-graduate project in 2007, fingerprint recognition system for access control systems was designed by applying morphological image processing techniques (Dilcan, 2007). Local and global features were extracted from the input fingerprint image then these were used on comparison step. But in this study, the recognition rate of the fingerprint identification system is tried to find by using PCA algorithm. So, implementation of fingerprint recognition system is started after completing the face recognition system. Taking the mean of the windows as features approach, that mentioned in Section 5.2.2.4 which has a highest recognition rate, is also used directly on fingerprint recognition system. Since, approach is the same with face recognition, implementation takes less time than face recognition system.

Hardware design on the FPGA of the fingerprint recognition system, is also same with the face recognition system. The only difference is image acquistion. To take fingerprint images fingerprint scanner is used. Section 5.2.3.1 describes this scanner, in Section 5.2.3.2 preliminary experiments for the fingerprint recognition system is introduced and in Section 5.2.3.3 final impelementation is proposed with giving the recognition results.

5.2.3.1 Fingerprint Scanner

In this fingerprint recognition system, *U.are.U 4000B USB Fingerprint Reader* is employed to get fingerprint images. Figure 5.49 shows the images of this reader. When the user simply places his/her finger on the glowing scanner window, the reader quickly and automatically scans the fingerprint. On-board electronics of this scanner calibrate the reader and encrypt the scanned data before sending it over the USB interface. This encrypted data is stored with binary values in code memory of the device.

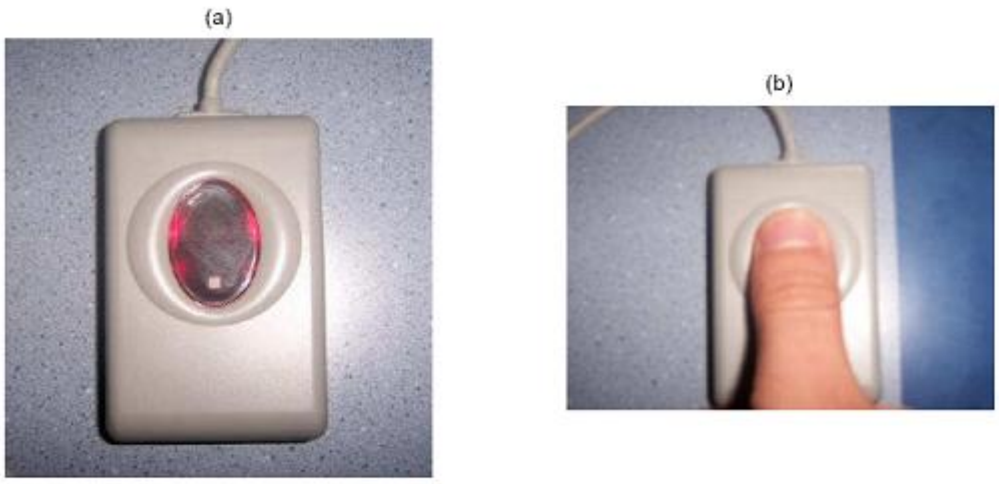
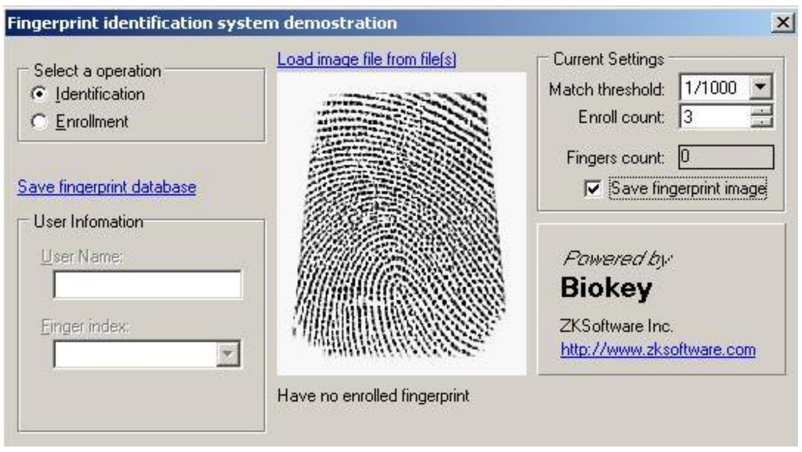


Figure 5.49 Images of U.are.U 4000B USB Fingerprint Reader (Dilcan, 2007).

In this system, the original fingerprint image is needed instead of encrypted binary data to use as an input to the PCA algorithm. For this reason, *biokeydemo*, that is released from ZK Software, is used for providing the fingerprint image in JPEG format. By using this software, enrollment and identification tasks of the biometric systems can be performed. The user interface of *biokeydemo* is shown in Figure 5.50. “Save fingerprint image” check box is used to store images that shown in *biokeydemo* user interface. Note that, most of the fingerprint scanners do not store the fingerprint as an image. Because, the fingerprint images of any person can be stored to custom database only for the test purposes.



Fingerprint identification system demostration

Select a operation
(● Identification) ○ Enrollment

Save fingerprint database

User Infomation
User Name:
Finger index:

Load image file from file(s)

Have no enrolled fingerprint

Current Settings
Match threshold: 1/1000
Enroll count: 3
Fingers count: 0
☑ Save fingerprint image

Powered by:
Biokey
ZKSoftware Inc.
http://www.zksoftware.com

Figure 5.50 biokeydemo user interface (Dilcan, 2007).

5.2.3.2 Preliminary Experiments

After a finger is pressed the fingerprint scan area, if “Save fingerprint image” check box of *biokeydemo* is enabled, it is stored as a JPEG image on the host computer. This fingerprint image is a colored image with the size of 500x550x3. To use this image in PCA algorithm, first it is converted into gray-scale then the most detailed part of the image is cropped to construct a new fingerprint image with the size of 400x400. Figure 5.51.a shows that the scanned image and the Figure 5.51.b shows the cropped image.

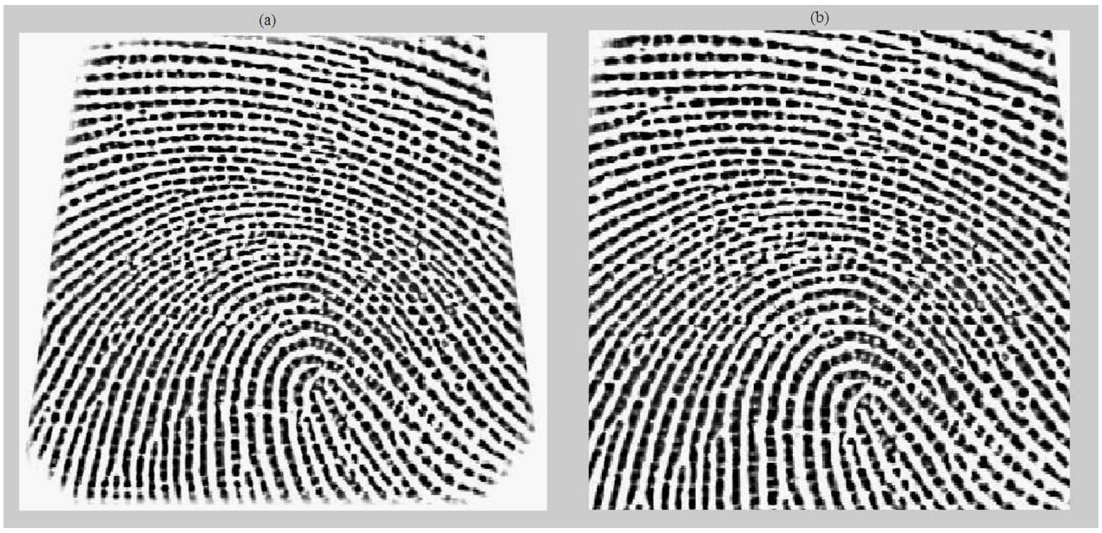


Figure 5.51 a) Scanned image (500x550x3) b) Cropped gray-scale image (400x400).

The cropped image is the input of offline training step just like in the face recognition system. Then, the gray-scale fingerprint image is resized by using *imresize* function. The new size of the fingerprint image is set to 100x100. Figure 5.52.a shows the gray-scale image with the size of 400x400, and Figure 5.52.b shows the new image that is resized to 100x100.

The approach in Section 5.2.2.4, that provides the highest recognition rate on face recognition system, is also used for fingerprint recognition system with a little difference. As mentioned in that section, input face image is resized to 40x40 then local mean matrix is extracted by using 4x4 local windows. But for fingerprint recognition, local mean matrix is extracted from 100x100 resized image by using 11x11 local windows. The reason for using with different size of local windows for the face and fingerprint recognition systems is the difference of the input images. The size of local windows for these systems are selected from the results of the extensive experiments.

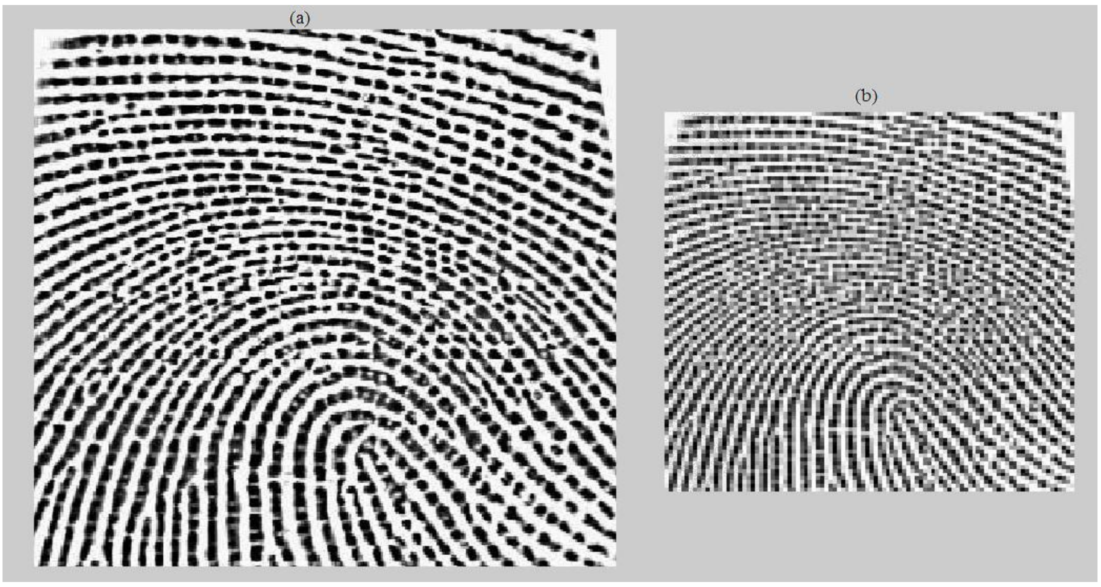


Figure 5.52 a) Cropped gray-scale image (400x400) b) Resized image (100x100).

The size of local mean matrix is 13x13, so the database size is 169x35 which is constructing by taking fingerprint images of 5 people and 7 images from each person. This system, System-K, is sent to FPGA via UART to start offline training phase. The variables *KISI_SAY* and *FEATURE_SAY* that used for configuring this system are 35 and 169 respectively. PCA algorithm is run after getting database features. PCA basis matrix, *symmat2*, and projections of fingerprint images to this basis, *ytrain*, are stored in the flash memory.

At online test phase, test image is sent to FPGA and projected to PCA basis space by multiplying *NewFlashPcaBasis* that is restored *symmat2* matrix from flash . This projection is compared to *NewFlashYtrain* that is also restored *ytrain* matrix. Absolute distances are computed and *toplam* vector is constituted. The minimum element of this *toplam* vector, *result_of_recognition*, holds the recognition result.

Table 5.18 shows the recognition table for System-K. The recognoition rate is very low by using the same approach in face recognition system implementation that mentioned in Section 5.2.2.4.

Table 5.18 Recognition table for System-K.

Recognition Table		test image		
		8th	9th	10th
person number	1st	+	-	-
	2nd	-	+	+
	3rd	-	-	-
	4th	+	-	+
	5th	-	+	-

Another system, System-L, is created to get better recognition rate. The big size of the input image, 400x400, is thought the reason of the low recognition rate in the System-K. To decrease the size of the input image local ridge orientation, which is mentioned in Section 3.3, is used to find the core region of a fingerprint. Note that, core region is the biggest orientation of the fingerprint image. By using the estimation of local ridge orientation approach, core region of the input image is found and cropped. The size of the core region is 180x260. Figure 5.53.a shows the gray-scale fingerprint image with the size of 400x400, and Figure 5.53.b shows the core region of the input image with the size of 180x260.

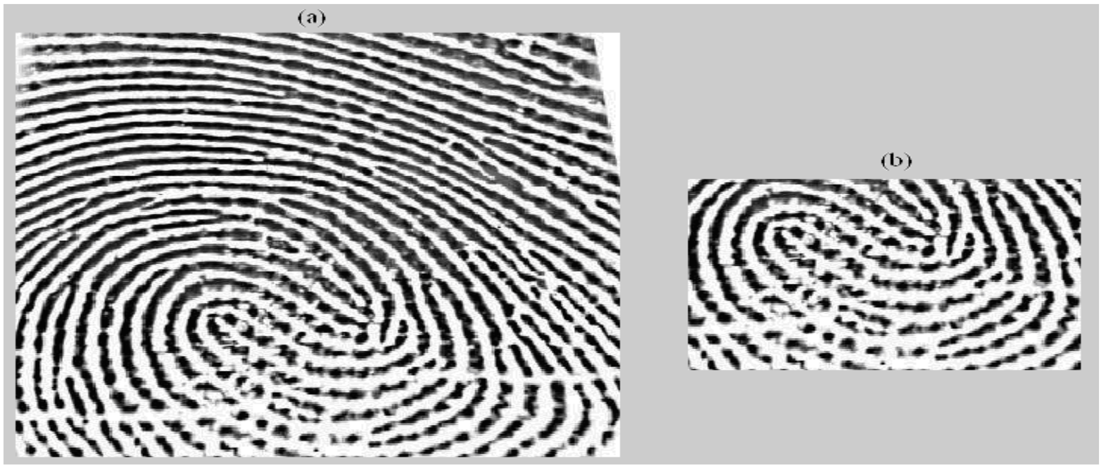


Figure 5.53 a) Input image (400x400) b) Core region of the input image (180x260).

Core region of the fingerprint image is then resized to 100x100. Figure 5.54 shows the core region of the fingerprint image after resize operation.

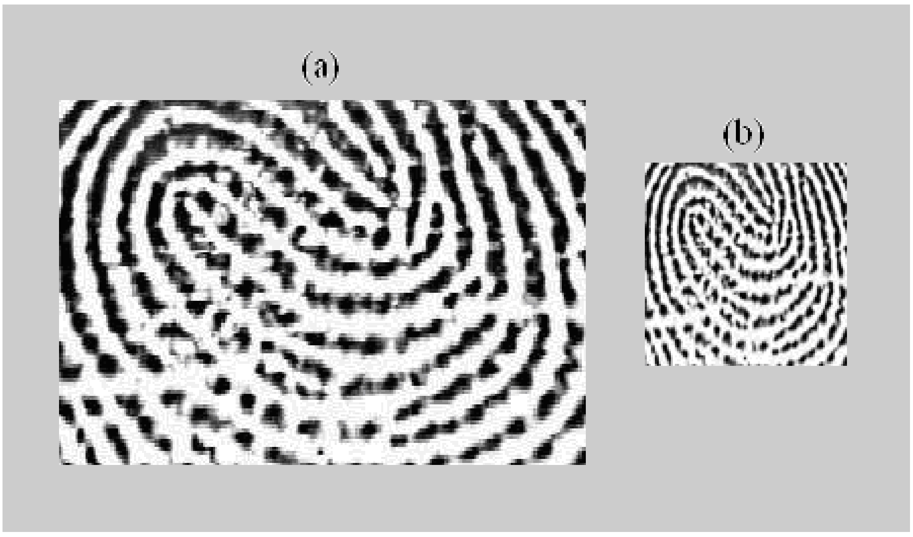


Figure 5.54 a) Core of the input image (180x260) b) Resized core region of the input image (100x100).

Local mean matrix with a size of 13x13, is extracted from resized core image. For offline training phase, the database is constructing by taking fingerprint images of 5 people and 7 images from each person. For System-L, offline training and online test phases are same with System-K and the recognition table is shown in Table 5.19.

Table 5.19 Recognition table for System-L.

Recognition Table		test image		
		8th	9th	10th
person number	1st	+	-	+
	2nd	-	+	+
	3rd	-	+	-
	4th	+	-	+
	5th	-	+	-

Table 5.19 shows that the recognition rate for System-L is %53.3.

5.2.3.3 Final Implementation

PCA algorithm in the previous trials are suspected from the low recognition rates. Therefore, the statistical variables such as mean and standart deviation of local windows are used for feature extraction with the estimation of local ridge orientation instead of PCA.

For offline training phase. first the input image with the size of 500x550x3 is converted to gray-scale and the most detail part of the image is cropped. The new size of the fingerprint image is 400x400. Local ridge orientations of the local windows, *theta*, with the size of 20x20, are calculated. After that, the fingerprint image with the size of 400x400, is resized to 100x100 by using *imresize* function of MATLAB. From this image means (*f_wmean*) and standard deviations (*f_wdev*) of 5x5 local windows are extracted. The extracted features; *theta*, *f_wmean* and *f_wdev* are converted to vectors then sent to FPGA. These features are stored on the flash memory on FPGA. The size of the features for one image is 1200x1. System-M is constructed with the fingerprint images of 5 people and 7 images from each person. The size of the database is 1200x35 for this system.

The features that are extracted at the offline training phase are also extracted for the fingerprint image that will be tested at the online test phase. Absolute distances are used for comparing. Recognition table for System-M is shown in Table 5.20.

Table 5.20 Recognition table for System-M.

Recognition Table		test image		
		8th	9th	10th
person number	1st	+	+	+
	2nd	+	+	+
	3rd	+	+	+
	4th	+	+	+
	5th	+	+	+

To make sure about the recognition rate, System-N is created by taking the fingerprint images of 5 people and 6 images from each person. Recognition table for System-N is shown in Table 5.21.

Table 5.21 Recognition table for System-N.

Recognition Table		test image			
		7th	8th	9th	10th
person number	1st	+	+	+	+
	2nd	+	+	+	+
	3rd	+	+	+	+
	4th	+	+	+	+
	5th	-	+	+	+

As seen from Table 5.20 and Table 5.21, after removing PCA algorithm the recognition rates are increased and higher from %90 percent. Even if the reason of this high recognition rate performance is dependent on the input image quality, it is totally acceptable for such a fingerprint recognition system. The source code of this system is in the Appendix with the folder name of "5_2_3_3_Highest_Recognition_ Rate_For_Fingerprint_Recognition".

5.2.3.4 General Performance of the Fingerprint Recognition System

For measuring the general performance of the fingerprint recognition system, same approach in Section 5.2.2.5 is used. The test subsystems are constructed by cross validation technique and the results are shown on the confusion matrix. Table 5.22 shows the general performance of the fingerprint recognition system on the confusion matrix. Note that, the values on this table are shown with the percent (%).

From Table 5.22, it can be seen that the images of 1st and 4th person are recognized without any error. The %1 images of the 2nd person, %4 images of the 1st person and %7 images of the 5th person are recognized as 3rd, 1st and 4th person respectively. The lowest performance of the fingerprint recognition system is seen on the 5th person with an error rate of %7.

Table 5.22 Results of the fingerprint recognition system on the confusion matrix.

Confusion Matrix	1st	2nd	3rd	4th	5th
1st	100	0	0	0	0
2nd	0	99	1	0	0
3rd	4	0	96	0	0
4th	0	0	0	100	0
5th	0	0	0	7	93

The total accuracy of the fingerprint recognition system is %97.6 This total accuracy value is also allowed to implement the fingerprint recognition system as a part of multibiometric recognition system.

CHAPTER SIX
FPGA-BASED MULTIBIOMETRIC RECOGNITION SYSTEM DESIGN

This chapter describes the implementation of FPGA-based multibiometric recognition system. After implementation of the face and fingerprint recognition systems successfully, the next step is to combine these two recognition systems in order to achieve a more reliable recognition system. This chapter summarizes the implementation steps of multibiometric recognition system and introduces the implementation results.

6.1 Implementation of Multibiometric Recognition System on DE2-70

Before introducing the multibiometric recognition system design, it is useful to describe that the hardware design is same with face and fingerprint recognition systems. Section 5.2.1 shows the basic system components and describes the hardware design procedure.

Software design of multibiometric recognition system is also similar with face and fingerprint recognition system with slight differences. As stated before, face images are resized and formed as the means of local windows then sent to FPGA as an input of the PCA algorithm in face recognition system. Unlike the face system, the statistical variables such as means and standard deviations with local ridge orientations are used as features instead of using PCA algorithm in fingerprint recognition system. Note that, MATLAB and FPGA source codes of this system are in the in the Appendix with the folder name of "6_1_ Implementation_of_ Multibiometric_Recognition_System".

In this multibiometric recognition system, each person is described by his/her face image and corresponding fingerprint image. For an example, if 4th face image of the 1st person is selected for creating the database, 4th fingerprint image of the 1st person is also selected. Offline training starts with the creation database block which is also similar to the previous implementations and *nsmultibiometric_train* function is used for this operation in MATLAB. In this function, first face images are resized. Then, the means of local windows are extracted from each resized face image. For 5 people with 6 face images from each person, total size of these features are 169x30 and they are stored in *Xbasis_face*. Instead of sending *Xbasis_face* directly to FPGA like in face recognition system, feature extraction step of fingerprint images is started in multibiometric recognition system. From the fingerprint images; means, standard deviations and ridge orientations of local windows are extracted. The total size of these features, for 5 people with 6 fingerprint images from each person, is 1200x30 and they are stored in *Xbasis_fing*. After completing the construction of each blocks separately, they are combined together in *Xbasis_tot*, then sent to FPGA. After FPGA receives this database, first the face and fingerprint blocks are separated. Then, the same offline training algorithm that is described in Section 5.2.2.4 is used for face portion and the same offline training algorithm that is described in Section 5.2.3.3 is used for fingerprint portion sequentially. Offline training ends by storing face and fingerprint features on the flash memory.

Online test phase is similar with previous face and fingerprint implementations. For MATLAB side, the difference is the combining test blocks together before sending to FPGA. The difference of the configuration software that runs on FPGA is the decision level of the algorithm. As mentioned in Section 1.2, in the multibiometric systems; the fusion can occur at the data or feature level, match score level and decision level. In this multibiometric recognition system implementation fusion occurs at the decision level. After *result_of_face_recognition* and *result_of_fing_recognition*, which are the recognition results of face and fingerprint recognition portions, are found sequentially the multibiometric recognition decision is calculated by ANDing these two results. As a summary, if same person is recognized in both of face and fingerprint portions of the system, the number of this person is printed to the console as a multibiometric recognition result. Otherwise, system gives an error message such as “person does not recognized”.

6.2 General Performance of the Multibiometric Recognition System

For measuring the general performance of the multibiometric recognition system, same approach in Section 5.2.2.5 and Section 5.2.3.4 is used. The test subsystems are constructed by cross validation technique and the results are shown on the confusion matrix. Table 6.1 shows the general performance of the multibiometric recognition system on the confusion matrix. Note that, the values on this table are shown with the percent (%).

Table 6.1 Results of the multibiometric recognition system on the confusion matrix

Confusion Matrix	1st	2nd	3rd	4th	5th
1st	100	0	0	0	0
2nd	0	99	1	0	0
3rd	4	0	94	2	0
4th	0	0	12	88	0
5th	0	0	0	7	93

From Table 6.1, it can be seen that the images of 1st person are recognized without any error. The %1 images of the 2nd person are recognized as 3rd person. %4 images of the 3rd person are recognized as 1st person, while %2 images are recognized as 4th person. %12 images of 4th person and %7 images of the 5th person are recognized as 3rd and 4th respectively. The lowest performance of the multibiometric recognition system is seen on the 4th person with an error rate of %12 which is the same error rate with face recognition system due to ANDing mechanism of the multibiometric recognition system.

The total accuracy of the multibiometric recognition system is %94.8. Using AND mechanism at the decision level reduces the recognition rate of the total system but offers more reliable biometric system implementation.

CHAPTER SEVEN
CONCLUSIONS

7.1 Summary of the Project

Biometric systems recognize a person automatically from his/her physiological and behavioral traits. Face and fingerprint recognition are the popular technologies of the biometric systems. In this thesis, real-time face and fingerprint recognition system is realized. This system has two working stages: offline training and online test.

To deploy a face and fingerprint recognition system, faces and fingerprints are processed. In offline training phase, the data is collected and feautures are extracted. In online testing phase, the test image is compared with the database. Data collection steps such as getting face images and scanning fingerprint images are implemented on host PC. Combination of these images and resizing images are provided by MATLAB on host PC. Data processing such as feature extraction, and comparison steps are implemented on FPGA. Principal Component Analysis (PCA) is used to extract features from face images and statistical variables with ridge orientations of local windows are used for fingerprint images.

The aim is to provide the highest recognition rate for each system. To reach this target, many different approaches and methods are tested such as changing the size of the image after resize operation, using face detection algorithm and windowing the image. The best accuracy has been obtained with PCA algorithm for face recognition and statistical variables with ridge orientations of local windows are used for fingerprint recognition. PCA algorithm on FPGA is fed by the output of these methods to reduce computation time at the feature extraction. The general performance of the system is calculated after generating 25 test subsystems by using cross-validation technique. Combining the recognition results of the test subsystems show that the total accuracy of the face and fingerprint recognition systems. For this implementation, the total accuracy of the face and fingerprint recognition system is

%97 and %97.6, respectively. After implementing face and fingerprint recognition systems successfully, these recognition systems are combined together to construct a multibiometric recognition system. In this multibiometric recognition system, fusion occurs at the decision level by using AND mechnasim. The general performance of the multibiometric recognition system is also calculated as results of the 25 test subsytems. The total accuracy of the multibiometric recognition system is %94.8. Since ANDing mechanism is used on multibiometric system, the recognition rate is reduced slighty when compared to face and fingerprint recognition systems.

For face recognition system, the computation time of the offline training phase on FPGA is nearly ten minutes after receiving database from MATLAB and online test phase just takes a few seconds to show the recognition result. For fingerprint recognition system, database is constructed under a minute and test phase just takes a few seconds. The computation time of the multibiometric system is nearly the summation of consumed time in face and fingerprint recognition systems.

7.2 Advantages – Disadvantages

The ability to update the functionality after shipping, partial re-configuration, various customization methods and the low non-recurring engineering costs are the most important features of the FPGAs. In this thesis, UP3 and DE2-70 development kits are used. VHDL is used for designing a UART module on UP3 and DE2-70 is used for implementing face, fingerprint and multibiometric recognition systems by using high level language at the configuration step. Various implementation methods on these two development kits can only be realized on FPGA with a short time period.

For implementing a real time recognition system, three important criteria must be considered. These are high recognition rate, short response time of the recognition system and low implementation cost. In this thesis, two of three features are successfully accomplished. This thesis offers three systems such as face, fingerprint and multibiometric recognition systems with high recognition rates and a person can be recognized in a few seconds. But, the integration of these systems to the real life is very difficult because the high cost of the project and using two separate environments such as MATLAB and FPGA.

For the personal aspect, using two FPGA development kits bring some advantages such as learning simulation and compilation of the projects on Quartus II, help for using hardware components such as memory or I/O elements directly from software, learning parallel configuration of a system by using VHDL and serial configuration of a system by C/C++ via Nios II IDE. The disadvantage is only the time that is spent on the first development kit, UP3. The reason of spending much time on UP3 are the difficulties of VHDL coding and understanding that the memory resources of UP3 cannot meet project specifications.

This study also helps for entering FPGA world by designing face, fingerprint and multibiometric recognition systems. As a hope that, interests of the people and applications of the FPGAs are increased to bring unique designs to FPGA world.

7.3 Troubleshooting

In the first development kit, UP3, for the communication between host PC and UP3, UART core is implemented in VHDL for using as a part of the recognition system. In this implementation, there are some clock-based issues such as missing a part of data during communication, but they are solved by shifting clock internally.

The most important problem faced with during this study on DE2-70 is the large dimension of the matrices in PCA algorithm. Because, if the input size of the PCA algorithm has bigger than 400 elements, unpredicted and unexpected errors are occured on the face recognition system. So, the input size of the PCA algorithm is adjusted and controlled by using local windowing to provide stability of the system.

For fingerprint recognition system, which is also implemented in DE2-70, statistical variables and ridge orientations of local windows are used to extract features from the input images after facing low recognition rates when PCA algorithm is used.

7.4 Cost Analysis

UP3 development kit is provided by the university, so this kit has not any effect on the total budget of the thesis. DE2–70 development kit has been purchased with a cost of $400, after facing memory issues on UP3. Fingerprint scanner, which is bought for the under-graduate project, is nearly $200. So, this scanner has not also any effect on the budget. Note that, the Quartus II software is provided with development kits.

7.5 Future Work

For a future study, all system components such as imaging and scanning devices that are connected to host PC can be moved and implemented on FPGA. The feature extraction method, PCA, can be changed by the other methods such as Independent Component Analysis (ICA) and Linear Discriminant Analysis (LDA). In this project, comparison method is the comparing absolute distances. In a future study, classification methods may be used.

REFERENCES

Anwar F., Rahman A., & Azad S. (2009). Multibiometric Systems Based Verification Technique. *European Journal of Scientific Research, ISSN 1450-216X Vol.34 No.2*, pp. 260-270.

Altera (2007). *Cyclone II Handbook*, Altera Corporations.

Bartlett M. S., Movellan J. R., & Sejnowski T. J. (2002). Face Recognition by Independent Component Analysis. *IEEE Trans. on Neural Networks, Vol. 13, No. 6, November 2002*, pp. 1450-1464.

Bledsoe, W. W., & Chan, H. (1965). A Man-Machine Facial Recognition System-Some Preliminary Results. *Technical Report PRI 19A*, Panoramic Research, Inc., Palo Alto, California.

Bledsoe, W. W. (1966a). Man-Machine Facial Recognition: Report on a Large-Scale Experiment. *Technical Report PRI 22*, Panoramic Research, Inc., Palo Alto, California.

Bledsoe, W. W. (1966b). Some Results on Multicategory Patten Recognition. *Journal of the Association for Computing Machinery 13* (2): 304-316.

Bolme D., Beveridge R., Teixeira M., & Draper B. (2003). The CSU Face Identification Evaluation System: Its Purpose, Features and Structure. *International Conference on Vision Systems, April 1-3*, Graz, Austria.

Comon P. (1994). Independent component analysis, a new concept? *Signal Processing*, 36: 287-314.

Daugman J. G. (1980). Two dimensional spectral analysis of cortial receptive field profile. *Vision Research, vol. 20.*, pp. 847-856.

Dilcan E. (2007). *Fingerprint Identification For Access Control Systems*, Dokuz Eylul University, Department of Electrical and Electronics Engineering, Undergraduate thesis, June 2007.

Escarra M., Robinson M., Krueger J., & Kochelek D. (2004). American Psychological Assocation Publication Manual: *Results of Eigenface Detection Tests*. Retrieved May, 2010 from http://cnx.org/content/m12536 /1.3/

Federal Bureau of Investigation. (1984). *The Science of Fingerprints*. Washington D.C.: U.S. Government Printing Office.

Gabor D. (1946). Theory of communication. *J. IEEE, vol. 93*, pp. 429-459.

Galton F. (1892). *Finger Prints*. London: Macmillan.

Goldstein A. J., Harmon L. D., & Lesk B. (1971). Identification of Human Faces. *Proc. IEEE, May 1971, Vol. 59, No. 5*, 748-760.

Kepenekci B. (2001). *Face Recognition Using Gabor Wavelet Transform*. Middle East Technical University, Department of Electrical and Electronics Engineering, PhD thesis.

Lee H. C., & Gaensslen R. E. (2001). *Advances in Fingerprint Technology, 2nd edition*, Elsevier, New York.

Li S. Z., & Jain A. K. (2004). *Handbook of Face Recognition*, Springer.

Li X., & Areibi S. (2004). A Hardware/Software Co-design Approach for Face Recognition. *Proc. 16th International Conference on Microelectronics*, Tunis, Tunisia, December 2004.

Liu C. & Wechsler H. (1999). Comparative Assesment of Independent Component Analysis (ICA) for Face Recognition. *Second International Conference on Audio- and*

Video- based Biometric Person Authentication, AVBPA'99, Washington D. C., USA, March 22-24.

Lu J., Plataniotis K. N., & Venetsanopoluos A. N. (2003). Boosting Linear Discriminant Analysis for Face Recognition. *Proc. IEEE, September 2003, Vol.1*, 657-660.

Lu J., Plataniotis K. N., & Venetsanopoluos A. N. (2003). Regularized Discriminant Analysis For the Small Sample Size Problem in Face Recognition. *Pattern Recognition Letters, December 2003, Vol. 24, Issue 16* : 3079-3087.

Maltoni D., Mario D., Jain A. K., & Prabhakar S. (2003), *Handbook of Fingerprint Recognition*, Springer.

MIT Media Laboratory Vision and Modeling Group (2002). *Photobook/Eigenfaces Demo*, Massachusetts Institute of Technology.

Moenssens A. (1971). *Fingerprint Techniques*, Chilton, London.

Nakano T., Morie T., & Iwata A. (2003). A Face/Object Recognition System Using FPGA Implementation of Coarse Region Segmentation, *SICE Annual Conference in Fukui, August 4-6*. Fukui University, Japan.

Prabhakar S., & Jain A. K. (2002). Decision-level fusion in fingerprint verification. *Pattern Recognition, vol. 35, no.4, pp. 861-874*, Springer.

Ross A., & Jain A. K. (2004). Multimodal Biometrics: An Overview. *Appeared in Proc. Of 12th European Signal Processing Conference (EUSIPCO), (Vienna, Austria)*, pp. 1221-1224.

Sajid I., Ahmed M. M., Taj I., Humayun M., & Hameed F. (2008). Design of High Performance FPGA Based Face Recognition System. *PIERS Proceedings, Cambridge,* USA, July 2.

Sirovich L., & Kirby M. (1987). A Low-Dimensional Procedure for the Characterization of Human Faces. *J. Optical Soc. Am. A, 1987, Vol. 4, No.3*, 519-524.

SLS (2004). *UP3 user manual Version 0.1,* SLS Corporations.

Smith K., Ross. A, & Colbry A. (2006). Face Recognition. *National Science and Technology Council (NSTC).*

Stosz J. D., & Alyea L.A. (1994). Automated System for Fingerprint Authentication Using Pores and Ridge Structure. *Proc. of SPIE (Automatic Systems for the Identification and Inspection of Humans), vol. 2277*, pp. 210–223, 1994

Terasic (2009). *DE2-70 user manual Version 1.08,* Terasic Technologies.

Turk. M. A., & Pentland A. P. (1991). Face Recognition Using Eigenfaces. *Proc. IEEE, 1991*, 586-591.

Wegstein J. H. S. (1992). An Automated Fingerprint Identification System. *T^NBS Special Publication 500-89,* February 1982.

Weinstein A., Volz J, & Redecer C. (2004). *Application note 4041: Implementing a Software UART on the MAXQ3210, May 03*, 2007.

Wiskott L. (1996). *Face Recognition by Elastic Bunch Graph Matching*. Retrieved June, 2010 from http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/ research/computerVision/graphMatching/identification/faceRecognition/contens .html.

Xilinx (2006). *Programmable Logic Handbook.* Xilinx Technologies.

APPENDIX

An *"Appendix CD"* is prepared which contains all MATLAB files, VHDL files and Nios II system designs that are used in this thesis. The folder names are dedicated to section numbers to reach source codes easily. Source code availability is mentioned in each section. As a remember, the content of "*Appendix CD*" is also given in the following with section name and corresponding folder name in the "*Appendix CD*";

- Section 5.1.3 PCA implementation on MATLAB - 5_1_3_PCA_MATLAB
- Section 5.1.6.1 One Byte Transmitter - 5_1_6_1_UART_Transmitter
- Section 5.1.6.2 One Byte Receiver - 5_1_6_2_UART_ Receiver
- Section 5.1.6.3 Array Transmitter - 5_1_6_3_UART_ Array_Transmitter
- Section 5.1.6.4 Simulation of Internal Database and Test Comparison - 5_1_6_4_Internal_Database_And_Test
- Section 5.1.6.5 Comparing Database and Test after Receiving from MATLAB - 5_1_6_5_Database_and_Test_sent_from_MATLAB
- Section 5.2.2.1 Software Design on DE2-70 - 5_2_2_1_Face_Recognition_Configuration_Sw
- Section 5.2.2.3 Preliminary Experiments - 5_2_2_3_fdmex_example
- Section 5.2.2.4 Final Implementation - 5_2_2_4_Highest_Recognition_Rate_For_Face_Recognition
- Section 5.2.2.5 General Performance of the Face Recognition System - 5_2_2_5_Cross_Validation
- Section 5.2.3.3 Final Implementation - 5_2_3_3_Highest_Recognition_ Rate_For_Fingerprint_Recognition
- Section 6.1 Implementation of Multibiometric Recognition System on DE2-70 - 6_1_ Implementation_of_ Multibiometric_Recognition_System