DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

FACE RECOGNITION USING NEURAL NETWORKS ON FIELD PROGRAMMABLE GATE ARRAY

by
Recep DOĞAN

March, 2011
İZMİR

FACE RECOGNITION USING NEURAL NETWORKS 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
Recep DOĞAN

March, 2011
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"FACE RECOGNITION USING NEURAL NETWORKS ON FIELD PROGRAMMABLE GATE ARRAY"** completed by **RECEP DOĞAN** 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 Erdas ÖZKURT

Supervisor

Yrd. Doç. Dr. Yavuz Şenol

(Jury Member)

Yrd. Doç. Dr. Taner Akkan

(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. This research is successfully completed thanks to her goodwill and selfless assistance.

I also would like to thank my family for endless support and their motivation during my reserch.

Recep DOĞAN

FACE RECOGNITION USING NEURAL NETWORKS ON FIELD PROGRAMMABLE GATE ARRAY

ABSTRACT

Biometric is a science of digital technology which is used to identify people based on unique physical or biological characteristics. There are several biometric technologies such as fingerprint, face, iris and speech recognition. The feature extraction techniques play important role for biometric recognition system design.

Recently, the Field Programmable Gate Arrays (FPGAs) have been commonly used in several applications such as digital signal processing, biometric recognition, medical imaging aerospace and defense systems, computer vision. Basically, FPGAs are the programmable logic devices. Each function of logic block can be organized by user. FPGAs are preffered in a variety of applications.

In this thesis, a face recognition system which is implemented on FPGA has been introduced. The principle component analysis (PCA) has been used for feature extraction and recognition has been accomplished by artificial neural network (ANN).

Since the training of the artificial neural network is a long process using only one processor on FPGA, a hierarchical classification with multiple processor approach has been followed. Thus, 47.2% system speedup has been obtained for a recognition rate of 93.9%.

Keywords : Face recognition, Neural network, Multiprocessor system, FPGA (Field Programmable Gate Arrays), PCA (Principle Component Analysis).

SAHADA PROGRAMLANABİLEN KAPI DİZİLERİNDE YAPAY SİNİR AĞLARI İLE YÜZ TANIMA

ÖZ

Biyometrik, kendine özgü fiziksel veya biyolojik niteliklerine dayalı olarak insanların kimliğini tespit etmek için kullanılan dijital teknolojiden faydalanma bilimidir. 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.

SPDK (Sahada Programlanabilir Kapı Dizileri) içeren uygulamalar, sayısal işaret işleme, biyometrik tanıma, medikal görüntü işleme, uzay ve savunma sistemleri, bilgisayar görüntüsü alanlarında kullanılmaktadır. SPKD programlanabilir mantık elemanlarıdır. Her bir mantık bloğunun işlevi kullanıcı tarafından düzenlenebilmektedir. SPDK çok sayıda uygulamada tercih edilmektedir.

Bu tezde SPDK üzerinde gerçekleştirilen yüz tanıma işlemi tanıtılmıştır. Öznitelik çıkarma işlemi için temel bileşen analizi (TBA) kullanılmıştır ve tanıma işlemi yapay sinir ağı (YSA) tarafından gerçekleştirilmiştir.

SPKD (Sahada Programlanabilir Kapı Dizileri) üzerinde bir işlemci kullanılarak, yapay sinir ağının eğitilmesi uzun süren bir işlemdir. Bu nedenle, hiyerarşik sınıflama yöntemi kullanılarak çok işlemcili sistem geliştirilmiştir. Böylelikle, %93.9 tanıma oranı için sistemin %47.2 daha hızlı çalışması sağlanmıştır.

Anahtar Sözcükler : Yüz tanıma, Yapay sinir ağı, Çok işlemcili sistem, Sahada Programlanabilir Kapı Dizileri (SPKD), Temel Bileşen Analizi (TBA).

CONTENTS

Page

CHAPTER ONE
INTRODUCTION

1.1 General Overview to Biometric Systems

Biometric is a science of digital technology which is used to identify people based on unique physical or biological characteristics. A number of biometric technologies have been developed such as fingerprint, face, iris and speech. Feature extraction techniques play important role for biometric recognition system design.

A biometric system is essentially a pattern recognition system that operates by acquiring biometric data from an individual, extracting a feature set from the acquired data, and comparing this feature set against the template set in the database (A. K. Jain, A. Ross, & S. Prabhakar, 2004). Depending on the application, a biometric system may be called either in verification system or identification system:

- In the verification mode, the system validates a person's identity by comparing the captured biometric data with her own biometric template(s) stored system database. In such a system, an individual who desires to be recognized claims an identity, usually via a PIN (Personal Identification Number), a user name, a smart card, etc., and the system conducts a one-to-one comparison to determine whether the claim is true or not (e.g., "Does this biometric data belong to Bob?"). Identity verification is typically used for positive recognition, where the aim is to prevent multiple people from using the same identity (J. L. Wayman, 2001).

- In the identification mode, the system recognizes an individual by searching the templates of all the users in the database for a match. Therefore, the system conducts a one-to-many comparison to establish an individual's identity (or fails if the subject is not enrolled in the system database) without the subject having to claim an identity (e.g., "Whose biometric data is this"). Identification is a critical component in negative recognition applications where the system establishes whether the person is

who she (implicitly or explicitly) denies to be. The purpose of negative recognition is to prevent a single person from using multiple identities (J. L. Wayman, 2001).

The block diagrams of a verification system and an identification system are shown in Figure 1.1.

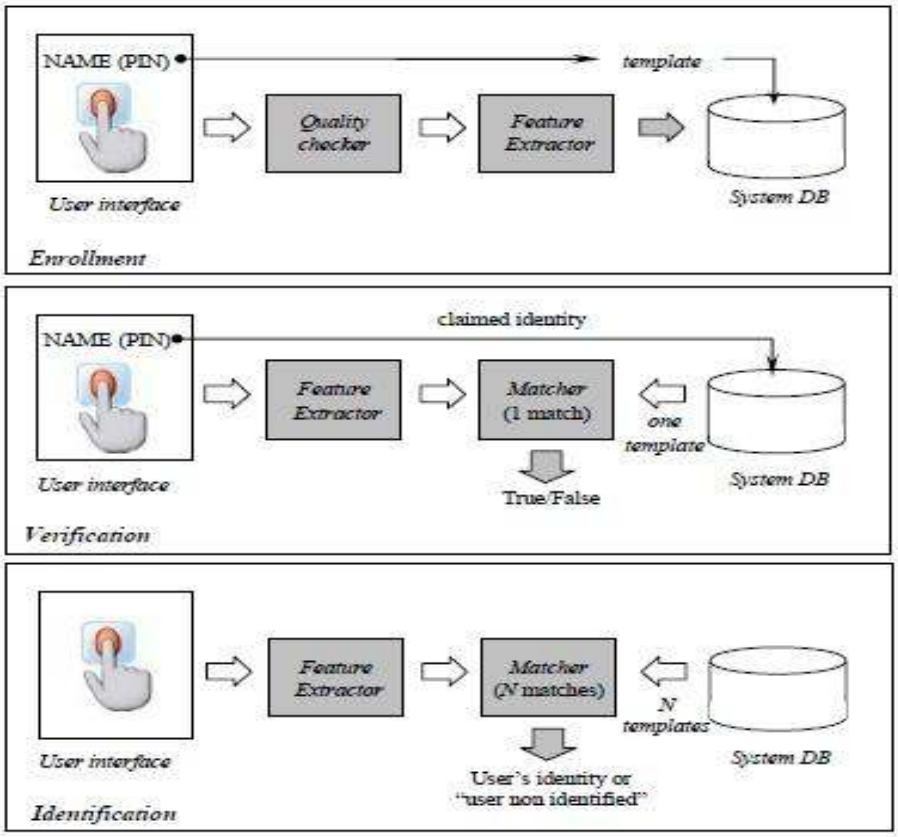

Figure 1.1 Block diagrams of enrollment, verification and identification tasks are shown using the four main modules of biometric system (A. K. Jain, A. Ross, & S. Prabhakar, 2004).

1.2 History of Face Recognition Systems

The intuitive way to do face recognition is to look at the major features of the face and compare these feature with the same features on the other faces. The first attempts to do this began in the 1960's with semi-automated system. During 1964 and 1965, Bledsoe, along with Helen Chan and Charles Bisson, worked on using the computer to recognize human faces (W. W. Bledsoe, 1966a, & 1966b; W. W.

Bledsoe, & H. Chan, 1965). Marks were made on photographs to locate the major features, it used features such as eyes, ears, noses, and mouths. Distances and ratios were computed from these marks to a common reference point and compared to reference 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. (A. J. Goldstein, L. D. Harmon, & B. Lesk, 1971). This proved even harder to automate due to the subjective nature of many of the measurements still made completely 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 (M. Escarra, M. Robinson, J. Krueger, & D. Kochelek, 2004) .

The first fully automated system to be developed utilized very general pattern recognition. It compared faces to a generic face model of expected features and created a series of patterns for an image relative to this model. This approach is mainly statistical and relies on histograms and the grayscale value. Kirby and Sirovich pioneered the eigenface approach in 1988 at Brown University (M. Escarra, M. Robinson, J. Krueger, & D. Kochelek, 2004) . 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 (L. Sirovich & M. 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. A discovery was enabled reliable real-time automated face recognition systems. Although the approach was somewhat constrained by environmental factors, it nonetheless created significant interest in furthering development of automated face recognition technologies (M. A. Turk & A. P. Pentland, 1991).

Since then, many different approaches have been developed for face recognition over the years such as Neural Network, Dynamic Link Architectures (DLA), Gabor Wavelet Transform, Elastic Bunch Graph, Hidden Markov Models. In 2010, M. Agarwal, N. Jain, H. Agrawal and M. Kumar worked on face recognition using principle component analysis (PCA), eigenface and neural network. This approach presents a methodology for face recognition based on information theory approach of coding and decoding the face image. Proposed methodology is connection of two stages: Feature extraction using principle component analysis and recognition using the feed forward back propagation Neural Network. The algorithm has been tested 400 images (40 classes). A recognition score for test lot is calculated by considering almost all the variants of feature extraction. The proposed methods were tested on Olivetti and Oracle Research Laboratory (ORL) face database. Test results gave a recognition rate of 97.018% (M. Agarwal, N. Jain, H. Agrawal, & M. Kumar, 2010).

Increasing of face recognition systems bring about hardware solutions such as application specific integrated circuit (ASIC) designs and field programmable gate arrays (FPGA). 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 1.2 shows this implementation (T. Nakano, T. Morie, & A. Iwata, 2003).

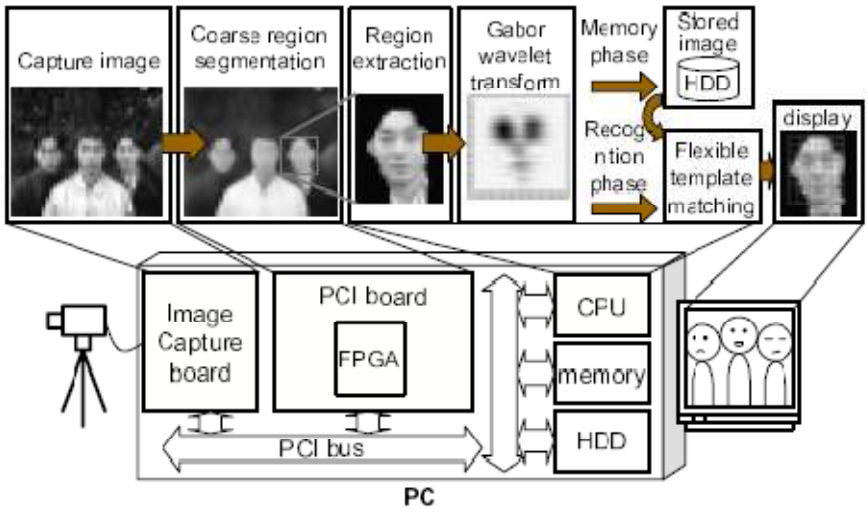

Figure 1.2 The face/object recognition system (T. Nakano, T. Morie, & A. Iwata, 2003).

1.3 General Overview to Multiprocessor and FPGA Systems

Advances in Field-Programmable Gate Array (FPGA) technologies have led to programmable devices with greater density, speed and functionality. It is possible to implement a highly complex System-on-Programmable-Chip (SoPC) using on-chip FPGA resources (e.g., DSP blocks, PLLs, RAM blocks, etc.) and vendor-provided intellectual property (IP) cores. Furthermore, it is possible to build Multiprocessor on a Programmable Chip (MPoPC) systems, where the number of softcore processors that can be used in a MPoPC system is only limited by device resources (A. Hung, W. Bishop, & A. Kennings, 2005).

There are several multiprocessor system designs which are implemented to increase performance of systems (C. Y. Tseng & Y.C. Chen, 2008). In study of C. Y. Tseng and Y.C. Chen, the performance of one, two, three, and four processors systems have been observed by running the benchmark program to measure the speedup. At the beginning, they run benchmark program to one processor system and test its execution time. Then, distribute their benchmark program for two, three, and four processors system independently. The speedup of two benchmark programs is

shown in Figure 1.3. The figure shows that the slope of these two lines becomes gradually small (C. Y. Tseng & Y.C. Chen, 2008).

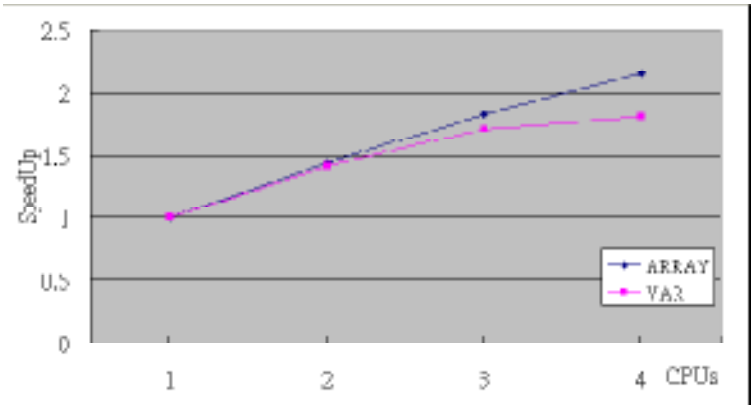

Figure 1.3 The speedup of two benchmarks (C. Y. Tseng & Y.C. Chen, 2008).

In VAR experiment, the one processor system as the standard system is made. Execution time of system is about 99.78 seconds. When VAR benchmark runs on two processors system, system spends 70.75 seconds. The speedup for two processors system is 1.41. When program runs in three and four processors system, system spends 58.26 and 54.95 seconds. Their speedups are 1.71 and 1.82 (C. Y. Tseng & Y.C. Chen, 2008).

In Array experiment result, the execution time for one processor system is 63.01 seconds, execution time for two processors is 43.6 seconds, execution time for three processors is 34.32 seconds and execution time for four processors is 29.27 seconds. The speedups are 1, 1.445, 1.836, and 2.152 (C. Y. Tseng & Y.C. Chen, 2008).

According to an another study, A. Tumeo, F. Regazzoni, G. Palermo, F. Ferrandi, and D. Sciuto presented the design of a reliable face recognition system implemented on Field Programmable Gate Array (FPGA). The proposed implementation uses the concept of multiprocessor architecture, paralel software and dynamic reconfiguration to satisfy the requirement of a reliable system. The target multiprocessor architecture is extended to support the dynamic reconfiguration of the processing unit to provide reliability to processors fault. The experimental results show that, due to the

multiprocessor architecture, the parallel face recognition algorithm can achieve a speed up of 63% with respect to the sequential version (A. Tumeo, F. Regazzoni, G. Palermo, F. Ferrandi, & D. Sciuto, 2010).

1.4 Aim of the Thesis

The aim of the thesis is to improve a previously implemented face recognition system running on Field Programmable Gate Array (FPGA). The proposed system rely on artificial neural networks for recognition while the previous system use Euclidian distance comparison. Furthermore, in order to have a faster training hierarchical classification with multiple processors approach has been followed.

The database of face images are stored in the host computer. Then, images are resized to increase calculation speed and combined in one database matrix and PCA features are extracted in MATLAB. This database matrix are sent to FPGA via serial port using RS-232 protocol. The neural network is trained with these features. Feed forward backpropagation algorithm is used as a neural network learning algorithm. Neural network system consist of 3 layers which are input layer, hidden layer and output layer. Input layer includes 10 neurons, hidden layer includes 5 neurons and output layer includes 1 neuron.

Since the training phase takes too long when only a single processor is used, a multiprocessor system with two processor is designed to reduce the training time. The speed of the multiprocessor system is approximately doubled.

Upon completion of training phase, the feature vector of test image is extracted by PCA and sent to the FPGA in order to find the owner of the image by neural network.

1.5 Outline of Thesis

This thesis is composed of seven chapters including the Introduction. Chapter 2 reviews face recognition processes, feature extraction methods and Principle

Component Analysis (PCA). Chapter 3 defines ANN; describe its properties and the algorithm which is used in this project. In Chapter 4, programmable logic device is introduced with the device that is used throughout project. In Chapter 5 design of multiprocessor system has been considered. Chapter 6 summarizes the face recognition system using field programmable gate array (FPGA) and explains the operation. The experiments and final results are also presented in this chapter. The last chapter of the thesis, Chapter 7, includes conclusions, advantages and disadvantages of the system, cost analysis, troubleshooting and 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 identfy or verify a person from images or videos. Face recognition systems can be operated in the following two modes:

- **Face Verification:**

A one to one comparison of a captured biometric with a stored template to verify that the individual is who he claims to be. It can be done conjuction with a smart card, username or ID number. The operation of verification system is shown in Figure 2.1.

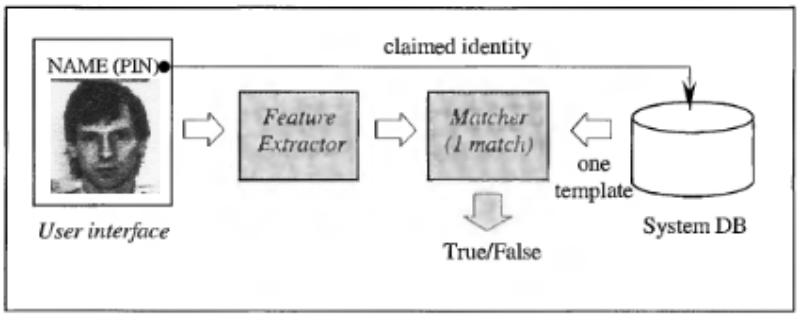

Figure 2.1 Face verification system (E. Dilcan, 2010).

- **Face Identification:**

A one to many comparison of the captured biometric against a biometric database in attempt to identify an unknown individual. The identification only succeeds in identifying the individual if the comparison of the biometric sample to a template in the database falls within a previously set threshold. The operation of identification system is shown in Figure 2.2.

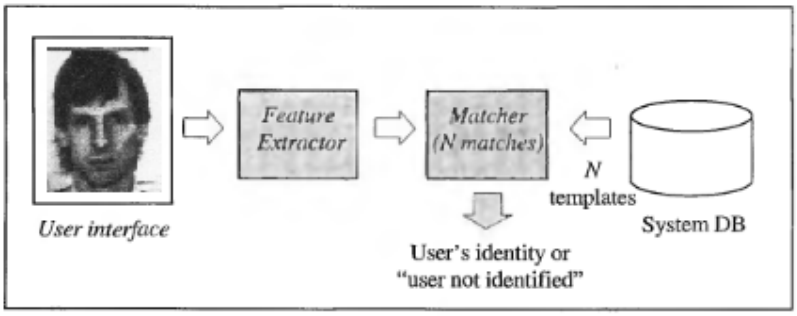

Figure 2.2 Face identification system (E. Dilcan, 2010).

2.2 Face Recognition Processing

Face recognition is a visual pattern recognition problem. A face recognition system generally consist of four main parts as shown in Figure 2.3: detection, alignment, feature extraction and matching.

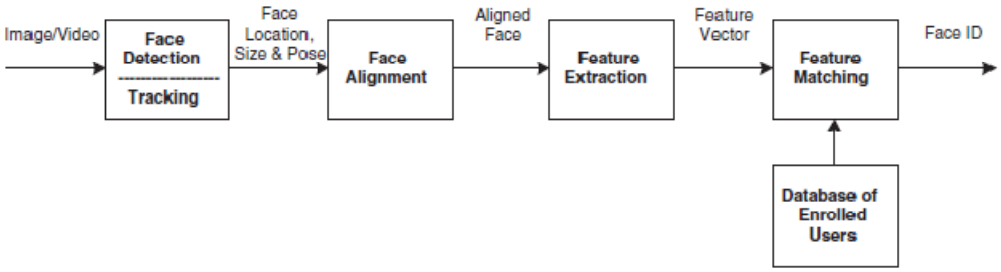

Figure 2.3 Face recognition processing flow scheme (S. Z. Li & A. K. Jain, 2004).

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 (S. Z. Li & A. K. Jain, 2004).

2.3 Face Recognition Techniques

Face recognition is a very active research area specialising on how to recognize faces within images or videos. There are many algorithms to perform face recognition including: 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 commonly used feature extraction technique for face recognition. Principle Component Analysis (PCA) is mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has as high a variance as possible (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it be orthogonal to (uncorrelated with) the preceding components. Principal components are guaranteed to be independent only if the data set is jointly normally distributed. PCA is sensitive to the relative scaling of the original variables.

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. PCA is used to reduce the dimension of the data by means of data compression basics. The reduction in dimensions removes the unuseful information and decomposes the face into orthogonal (or uncorrelated) components, which are also known as eigenfaces.

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

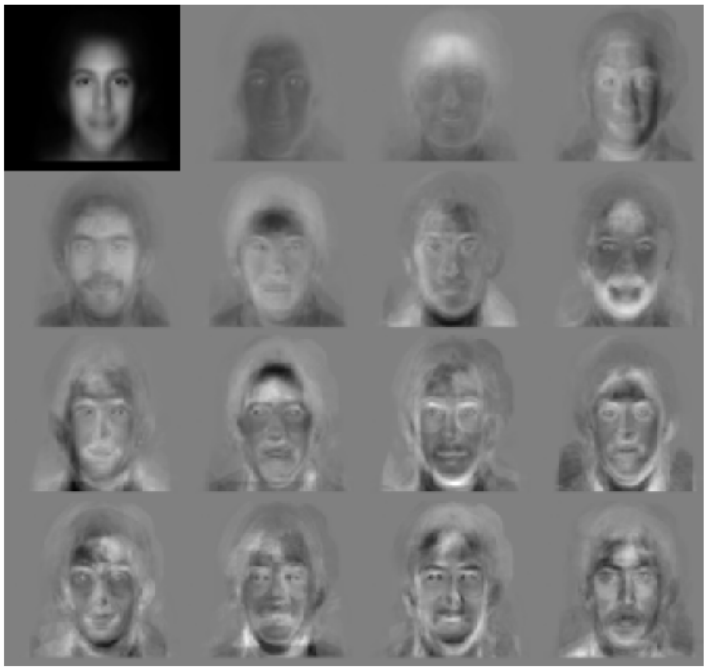

Figure 2.4 An example of eigenfaces (MIT Media Laboratory, 2002).

Theory of PCA is described below:

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 & 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,

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

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

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

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 (M. Turk & A. Pentland, 1991).

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 (D. Bolme, R. Beveridge, M. Teixeira, & B. 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 (J. Lu, K. N. Plataniotis, & A. N. 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 (J. Lu, K. N. Plataniotis, & A. N. Venetsanopoulos, 2003).

Figure 2.5 An example of six classes using LDA (J. Lu, K. N. Plataniotis, & A. N. Venetsanopoulos, 2003).

Theory of LDA is described below:

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 $\{x_1, x_2, ..., x_n\}$ where each image belongs to one of c classses $\{X_1, X_2, ..., X_c\}$. 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;

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T \qquad (2\text{-}10)$$

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:

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T \tag{2-11}$$

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;

$$W_{opt} = \arg\max \frac{\left| W^T S_B W \right|}{\left| W^T S_W W \right|} = \left[w_1 \; w_2 \; \ldots \; w_m \right] \tag{2-12}$$

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 $\{\lambda_i \mid i = 1, 2, \ldots, m\}$, which can be shown that as in following;

$$S_B w_i = \lambda_i S_W w_i \quad \text{where } i = 1, 2, \ldots, m \tag{2-13}$$

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;

$$\begin{aligned} y &= \theta^T x \quad \text{(only PCA)} \\ z &= W_x^T x \quad \text{(only LDA)} \\ z &= W_y^T y \quad \text{(PCA + LDA)} \end{aligned} \tag{2-14}$$

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 (M. S. Bartlett, J. R. Movellan, & T. J. Sejnowski, 2002).

ICA algorithm relies on the infomax algorithm. It receives an n-dimensional random vector as input. PCA algorithm is used to reduce the size of random vector. The higher order relationships aren't affected from dimensional reduction. Then, ICA algorithm finds the covariance matrix of the result and its factorized form is obtained. Then, some defined methods are performed to obtain the independent components that each face images in face space includes. These methods are whitening, rotation and normalization (Hyvarinen, 1999).

Theory of ICA is described below:

ICA of a random vector searches for a linear transformation which minimizes the statistical dependence between its components (P. Comon, 1994). Let, the image is represented by a random vector, $X \in R^N$, 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 (C. Liu & H. Wechsler, 1999). The covariance matrix of X can be expressed by using expectation operator, $E(.)$, as in the following;

$$C_X = E\{[X - E(X)][X - E(X)]^T\} \tag{2-15}$$

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

$$C_X = F \Delta F^T \tag{2-16}$$

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;

$$X = FZ \tag{2-17}$$

To find the transformation F, Comon developed an algorithm that consists of three operations: whitening, rotation and normalization (P. 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;

$$X = \varphi A^{1/2} U \tag{2-18}$$

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

$$C_X = \varphi A \varphi^T \tag{2-19}$$

where $\varphi = [\varphi_1, \varphi_2, ..., \varphi_N]$ 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 (P. Comon, 1994).

2.3.4 Bayesian Face Recognition Method

Bayesian Method propose a new technique for direct visual matching of images for the purposes of face recognition and image retrieval, using a probabilistic measure of similarity, based primarily on a Bayesian (MAP) analysis of image differences. The performance advantage of this probabilistic matching technique over Standard Euclidean nearest-neighbor eigenface matching was demonstrated using results from DARPA's 1996 FERET face recognition competition, in which this Bayesian matching alogrithm was found to be the top performer (B. Moghaddam, T. Jebara, & A. Pentland, 2000).

A Bayesian approach presents a probabilistic similarity measure based on the Bayesian belief that the image intensity differences, denoted by $\Delta = I_1 - I_2$, are characteristic of typical variations in appearance of an individual. In particular, we define two classes of facial image variations: intrapersonal variations Ω_I (corresponding, for example, to di!erent facial expressions of the same individual) and extrapersonal variations Ω_E (corresponding to variations between diwerent individuals). Our similarity measure is then expressed in terms of the probability.

$$S(I_1, I_2) = P(\Delta \epsilon \Omega_I) = P(\Omega_I|\Delta) \tag{2-20}$$

where $P(\Omega_I|\Delta)$ is the a posteriori probability given by Bayes rule, using estimates of the likelihoods $P(\Delta \epsilon \Omega_I)$ and $P(\Delta \epsilon \Omega_E)$. These likelihoods are derived from training data using an efficient subspace method for density estimation of high-dimensional data.

Given these likelihoods we can evaluate a similarity score $S(I_1, I_2)$ between a pair of images directly in terms of the intrapersonal a posteriori probability as given by Bayes rule:

$$S(I_1, I_2) = \frac{P(\Delta|\Omega_I)P(\Omega_I)}{P(\Delta|\Omega_I)P(\Omega_I)+P(\Delta|\Omega_E)P(\Omega_E)} \tag{2-21}$$

where the priors $P(\Omega)$ can be set to reflect specific operating conditions (e.g., number of test images vs. the size of the database) or other sources of a priori knowledge regarding the two images being matched. Note that this particular Bayesian formulation casts the standard face recognition task (essentially an M-ary classification problem for M individuals) into a binary pattern classification problem with Ω_I and Ω_E. This simpler problem is then solved using the maximum a posteriori (MAP) rule -- i.e, two images are determined to belong to the same individual if $P(\Omega_I|\Delta) > P(\Omega_E|\Delta)$, or equivalently, if $S(I_1, I_2) > \frac{1}{2}$ (B. Moghaddam, T. Jebara, & A. Pentland, 2000).

Figure 2.6 shows an orthogonal decomposition of the vector space $\Re^N$ into two mutually exclusive subspaces: the principal subspace *F* containing the first *M* principal components and its orthogonal complement *F*, which contains the residual of the expansion.

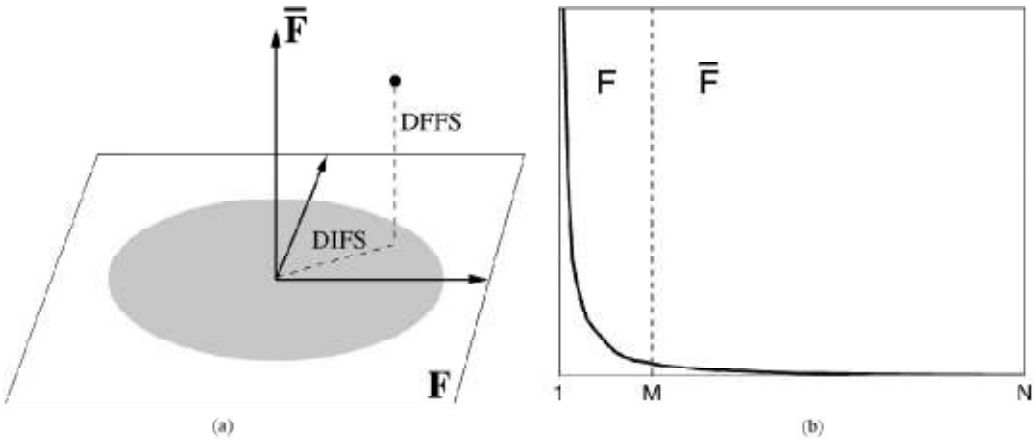

Figure 2.6 (a) Decomposition of $\Re^N$ into the principal subspace F and its orthogonal complement $\bar{F}$ for a Gaussian density, (b) a typical eigenvalue spectrum and its division into the two orthogonal subspaces (B. Moghaddam, T. Jebara, & A. Pentland, 2000).

CHAPTER THREE
ARTIFICIAL NEURAL NETWORK

3.1 Introduction

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones.

3.2 Biological Neuron

In the human brain, a typical neuron collects signals from others through a host of fine structures called dendrites. The neuron sends out spikes of electrical activity through a long, thin stand known as an axon, which splits into thousands of branches. At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity from the axon into electrical effects that inhibit or excite activity in the connected neurones. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes. Biological neuron components, nucleus, cell body, dendrites, axon, synapse are shown in Figure 3.1.

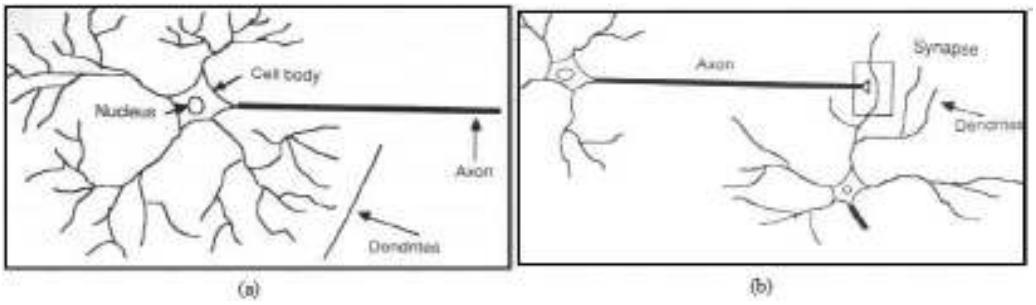

Figure 3.1 (a) Components of a neuron (b) The synapse

3.3 Neural Network Model

3.3.1 Simple Single Unit Network

A simple artificial neural networks consists of five sections: inputs, weights, summation function, activation function and outputs as diagram is shown in Figure 3.2.

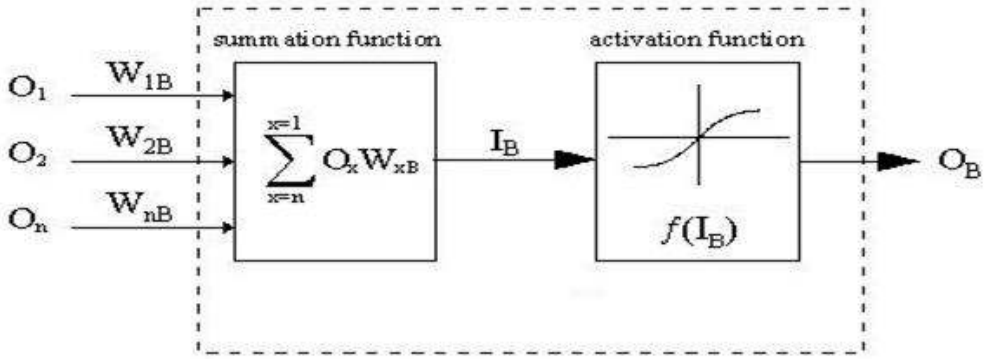

Figure 3.2 Simple artificial neural network

Neural networks are models of biological neural structures. Neuron in Figure 3.2 consists of multiple inputs and a single output. Each input is modified by a weight, which multiplies with the input value. The neuron will combine these weighted inputs and, with reference to a threshold value and activation function, use these to determine its output.

3.3.2 Multilayer Perceptron

3.3.2.1 Introduction to Multilayer Perceptron

The network consisting of a set of sensory units (neurons) that constitute the input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. These neural networks are commonly referred to as multilayer perceptrons (MLPs) (S. Haykin, 2001). A multilayer perceptron consists of minimum three sections: input layer, hidden layer and output layer as shown in Figure 3.3.

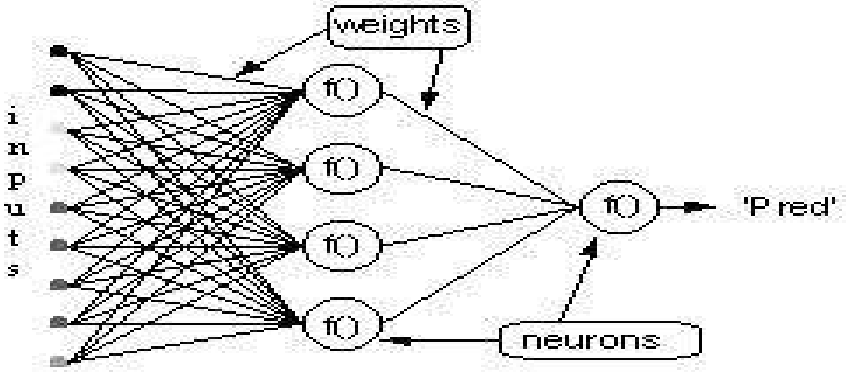

Figure 3.3 Multilayer perceptron

3.3.2.2 Backpropagation Algorithm

Error back-propagation learning consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network, and its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the networks are all fixed. During the backward pass, on the other hand, the synaptic weights are all adjusted in accordance with an error correction rule.

Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal. This error signal is then propagated backward through the network, against the direction of synaptic connections-hence the name "error back-propagation". The synaptic weights are adjusted to make actual response of the network move closer to the desired response in a statistical sense. The error back propagation algorithm is also referred to in the literature as the back-propagation algorithm (S. Haykin, 2001).

The backpropagation learning algorithm can be divided into two phases: propagation and weight update.

• Propagation:

Each propagation involves the following steps:

- Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
- Back propagation of the propagation's output activations through the neural network using the training pattern's target in order to generate the deltas of all output and hidden neurons.

• Weight update

For each weight-synapse:

- Multiply its output delta and input activation to get the gradient of the weight.
- Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight

This ratio influences the speed and quality of learning; it is called the learning rate. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

3.3.2.3 Theory of Backpropagation Algorithm

The error signal at the output of neuron j at time step n (n_{th} training example) is defined by;

$$e_j(n) = d_j(n) - y_j(n) \tag{3-1}$$

$d_j(n)$ is desired response for neuron j. Since all neurons have instantaneous error energy, the instantaneous total energy is computed as summing all the neurons in the output layer;

$$\xi(\mathrm{n}) = \frac{1}{2}\sum_{j=C} e_j^2\ (n) \tag{3-2}$$

where set C includes all neurons in the output layer. When there are N total number of examples the average squared error energy will be;

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(\mathrm{n}) \tag{3-3}$$

For a given training set ξ_{av} represents the cost function, as a measure of learning performance. The objective of learning is to minimize the cost function, when the cost function approaches zero, the network will be able to detect and classify all the inputs similar to the training set correctly.

The backpropagation algorithm applies a correction of $\Delta w_{ji}(\mathrm{n})$, which is proportional to the partial derivative of error to the synaptic weight and can be written according to the chain rule;

$$\frac{\partial \xi(\mathrm{n})}{\partial w_{ji}(\mathrm{n})} = \frac{\partial \xi(\mathrm{n})}{\partial e_j(\mathrm{n})}\frac{\partial e_j(\mathrm{n})}{\partial y_j(\mathrm{n})}\frac{\partial y_j(\mathrm{n})}{\partial v_j(\mathrm{n})}\frac{\partial v_j(\mathrm{n})}{\partial w_j(\mathrm{n})} \tag{3-4}$$

And with some differentiation with respect to error, output and the sum function, this equation is minimized to;

$$\frac{\partial \xi(\mathrm{n})}{\partial w_{ji}(\mathrm{n})} = -e_j(n)\varphi_j'(v_j(n))\, y_j(\mathrm{n}) \tag{3-5}$$

The correction applied to the synaptic weight is defined by the modified delta rule as;

$$\Delta w_{ji}(\mathrm{n}) = -\eta \frac{\partial \xi(\mathrm{n})}{\partial w_{ji}(\mathrm{n})} \tag{3-6}$$

Where η is the learning rate parameter and minus sign accounts for gradient descent (a direction) in weight space. Finally the correction is;

$$\Delta w_{ji}(\mathrm{n}) = \eta \delta_j(n)\, y_j(\mathrm{n}) \tag{3-7}$$

Where the local gradient $\delta_j(n)$ is defined as;

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \tag{3-8}$$

In the application of backpropagation algorithm, two passes are processed.

In the forward pass the weights remain unchanged through the network and output is calculated by neuron by neuron basis.

$$y_j(n) = \varphi_j(v_j(n)) \tag{3-9}$$

Where $v_j(n)$ is the induced local field of neuron and computed as;

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(\mathrm{n}) y_j(\mathrm{n}) \tag{3-10}$$

Where $w_{ji}(\mathrm{n})$ is the synaptic weight connecting neuron *i* to neuron *j* at time *n* and $y_j(\mathrm{n})$ is the input to the neuron *j*.If neuron *j* is the first layer then the input is the general input of the network, if it is in a hidden layer then its input is the output of

the previous layer and calculations are done in a standart way. But what the j^{th} neuron is in the output layer it is compared to the desired response then the backward pass accurs.

The backward pass starts output layer by passing the error signals leftward through the network, layer by layer and recursively computing local gradient for each neuron. For a neuron in the output layer, the local gradient δ is simply the error signal of that neuron multiplied by the first derivative of its nonlinearity.

Hyperbolic tangent function is commonly used form of sigmoidal non-linearity is the hyperbolic tangent function, which in its most general form is defined by;

$$\varphi_j(v_j(n)) = atanh(bv_j(n)), \qquad (a,b) > 0 \tag{3-11}$$

where *a* and *b* are constants. Its derivative with respect to $v_j(n)$ is given by;

$$\varphi_j'(v_j(n)) = absech^2(bv_j(n)) \tag{3-12}$$

For a neuron *j* located in the output layer, the local gradient is;

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \tag{3-13}$$

For a neuron j in a hidden layer, we have;

$$\delta_j(n) = \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(\mathrm{n}) \tag{3-14}$$

We may calculate the local gradient δ_j without requiring explicit knowledge of the activation function.

3.3.3 Learning in Neural Network

In neural network processing, several effecting factors is considered for a greater performance and a good generalization over the data. These factors are;

- Input Data Selection
- Preprocessing
- Cross-Validation
- Number of Hidden Neurons
- Initializing Weights
- Type of Activation Function

3.3.3.1 Input Data Selection

The performance of a neural network is dependent on the quality and relevance of its data. It is very important to choose appropriate input data for create a successful neural network system. In this thesis, Principle Component Analysis (PCA) results are used as input data of neural network. Principle Component Analysis (PCA) is applied to the face database to obtain input data. The dimension of the input vector is 10. These features are principal components of data which have the largest variance.

3.3.3.2 Preprocessing - Postprocessing

Neural network training can be made more efficient if you perform certain preprocessing steps on the network inputs and targets. The normalization step is applied to both the input vectors and the target vectors in the data set and all data scaled in a range from -1 to 1. In this way, the network output always falls into a normalized range . The network output can then be reverse transformed back into the units of the original target data when the network is put to use in the field.

3.3.3.3 Cross-validation

The data are split into two parts. The models are trained in the training data set and they are tested in the production (test) data set. By applying cross-validation, the over fitting problem is avoided and a good generalization is achieved.

In this thesis, Cross-validation is applied to generalize the results. %70 of data randomly selected from a database is used for training and %30 of data randomly selected from a database is used for testing. This process is repeated 20 times for the generalization of the results. The neural network results are evaluated by calculating the mean of these results.

3.3.3.4 Number of Hidden Neurons

The number of hidden units governs the expressive power and complexity of the network. Increasing the number of hidden units does not mean better performance when considering neural learning. Finding the appropriate number of hidden units is an ad-hoc process that is not exactly solved in neural processing. In this study, 5 hidden neurons are used in the hidden layer.

3.3.3.5 Initializing Weights

The starting point of the network is also one of the most important pre-conditions in the learning process of the neural network that is not also yet solved. In the error surface, you have to start in a point that will take you to the global minima.

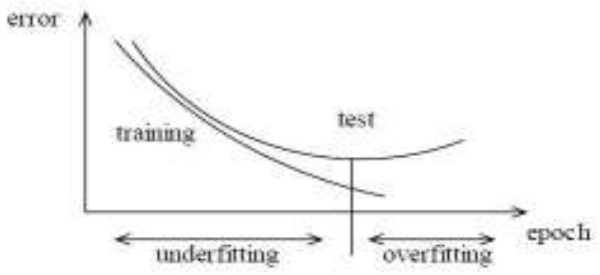

Figure 3.4 Error surface

There may be several local minima that will make the network not be able to generalize well or as desired. For this purpose, the initialization of the weights plays a crucial role in the network performance. You cannot initialize the weights to 0 otherwise learning cannot take place. In setting the weights, we choose weights randomly from a single distribution to help ensure uniform learning and have the network to generalize well.

3.3.3.6 Activation Functions

- Sigmoid Function:

This is the most widely used activation function in NN. Sigmoid function gives continues results to the inputs. Results are not discreet. This function is suitable for the problems which sensitive evaluation should be applied. Result of the sigmoid function is between 0 and 1.

Sigmoid function is;

$$f(x) = \frac{1}{1+e^{-\beta(x+b)}} \tag{3-15}$$

where β is gradient, x is input and b is the bias.

- Gaussian Function:

Gaussian function provides easier to prediction of the behaviour of the net when the input patterns differ strongly from all teaching patterns.
Gaussian function as activation function is;

$$f(x) = \frac{1}{\sqrt{2\pi\beta}} e^{\frac{-(x-\mu)^2}{2\beta^2}} \tag{3-16}$$

where β is gradient, x is input and μ is the learning rate.

- Unit Step Function*:*

If input is greater than 0, output is 1, otherwise output is 0. This function can be used for simple problems. This function is not useful for complex problems.

Unit step function is;

$$f(x) = 0 \text{ if } x<0 \quad (3\text{-}17)$$

$$f(x) = 1 \text{ if } x\geq 0 \quad (3\text{-}18)$$

- Hyperbolic Tangent:

Difference of that function from others, that function returns results between -1 and 1. Hyperbolic tangent activation function is used in this study.

Hyperbolic Tangent function is;

$$f(x) = (1\text{-}e^{-2x})/(1+e^{-2x}) \quad (3\text{-}19)$$

CHAPTER FOUR
FIELD PROGRAMMABLE GATE ARRAYS

4.1 Introduction to Field Programmable Gate Arrays (FPGA)

A Field-programmable Gate Array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing, hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

The FPGA industry sprouted from programmable read-only memory (PROM) and programmable logic devices (PLDs). PROMs and PLDs both had the option of being programmed in batches in a factory or in the field (field programmable), however programmable logic was hard-wired between logic gates.

In the late 1980s the Naval Surface Warfare Department funded an experiment proposed by Steve Casselman to develop a computer that would implement 600,000 reprogrammable gates. Casselman was successful and a patent related to the system was issued in 1992.

Some of the industry's foundational concepts and technologies for programmable logic arrays, gates, and logic blocks are founded in patents awarded to David W. Page and LuVerne R. Peterson in 1985.

Xilinx Co-Founders, Ross Freeman and Bernard Vonderschmitt, invented the first commercially viable field programmable gate array in 1985 – the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 boasted a mere 64 configurable logic blocks (CLBs), with two 3-input lookup tables (LUTs).

The 1990s were an explosive period of time for FPGAs, both in sophistication and the volume of production. In the early 1990s, FPGAs were primarily used in telecommunications and networking. By the end of the decade, FPGAs found their way into consumer, automotive, and industrial applications.

4.2 FPGA Architecture

FPGAs consist of an array of programmable logic blocks of potentially different types, including general logic, memory and multiplier blocks, surrounded by a programmable routing fabric that allows blocks to be programmably interconnected. The array is surrounded by programmable input/output block, labeled I/O in the Figure 4.1, that connect the chip to the outside world (I. Kuon, R. Tessier, & J. Rose, 2007).

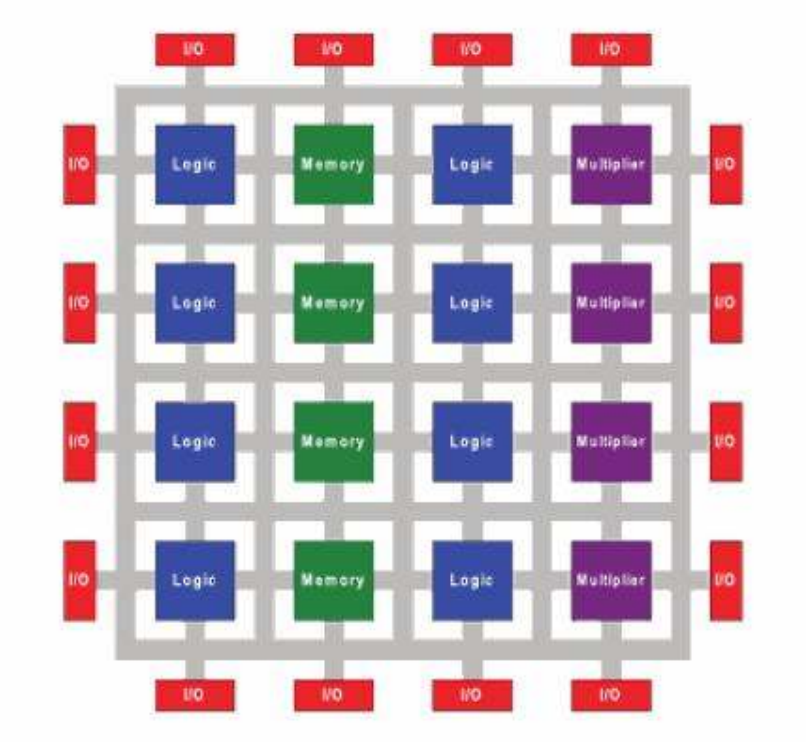

Figure 4.1 Basic FPGA Structure (I. Kuon, R. Tessier, & J. Rose, 2007).

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.

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).

The Cyclone II LE operates in one of the following modes:

- Normal mode
- Arithmetic mode

The normal mode is suitable for general logic applications and combinational functions. In normal mode, four data inputs from the LAB local interconnect are inputs to a four-input LUT. The arithmetic mode is ideal for implementing adders, counters, accumulators, and comparators. An LE in arithmetic mode implements a 2-bit full adder and basic carry chain.

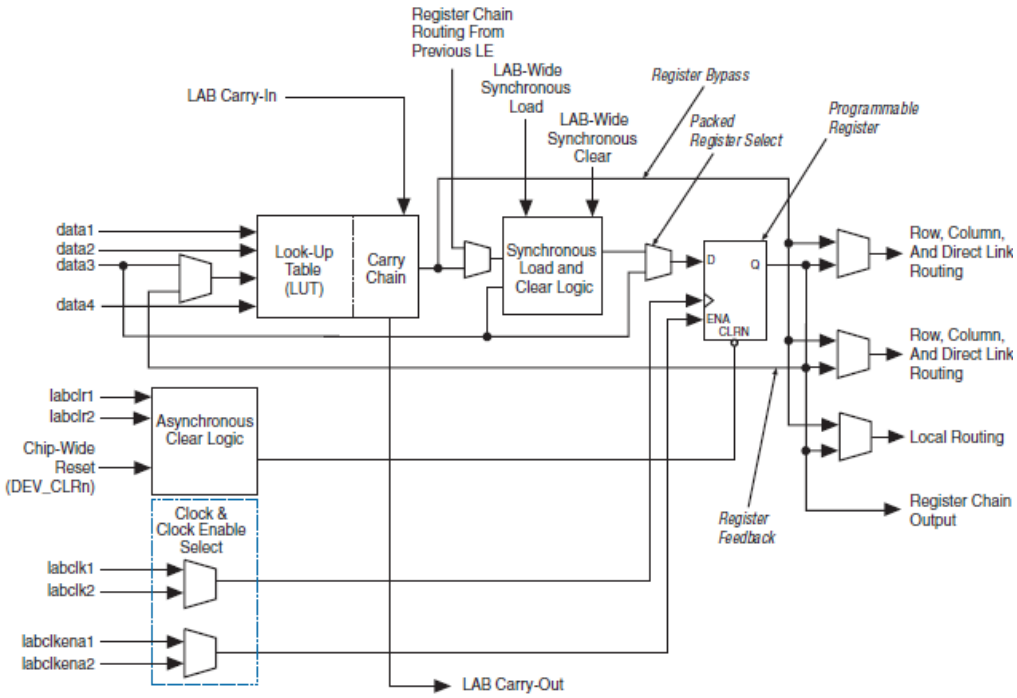

Figure 4.2 Cyclone II logic element (Cyclone II Handbook, Altera Corp., 2007).

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 4.3 shows Cyclone II LAB architecture.

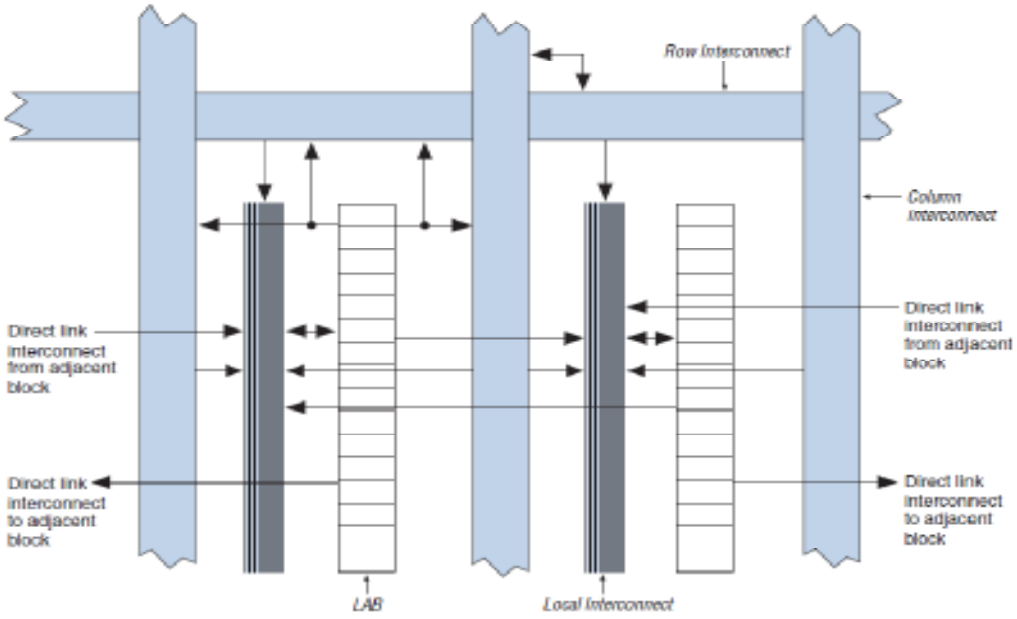

Figure 4.3 Cyclone II LAB architecture (Cyclone II Handbook, Altera Corp., 2007).

4.3 FPGA Configuration

FPGAs can be programed 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.

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 (E. Dilcan, 2010).

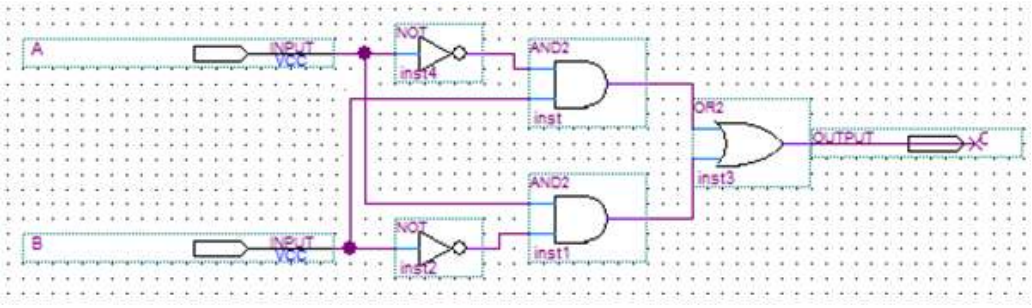

Figure 4.4 An example by using schematic design entry (E. Dilcan, 2010).

4.3.2 Hardware Description Languages

HDLs are standard text-based expressions of the spatial and temporal structure and behaviour of electronic systems. Like concurrent programming languages, HDL syntax and semantics includes explicit notations for expressing concurrency. However, in contrast to most software programming languages, HDLs also include an explicit notion of time, which is a primary attribute of

hardware. Languages whose only characteristic is to express circuit connectivity between a hierarchy of blocks are properly classified as netlist languages used on electric computer-aided design (CAD).

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 4.5 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 4.6 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 (E. Dilcan, 2010).

Half adder implementation by using VHDL, Verilog HDL and SystemC is shown in Figure 4.5, Figure 4.6 and Figure 4.7 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 4.7 Half adder implementation by using SystemC.

4.4 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

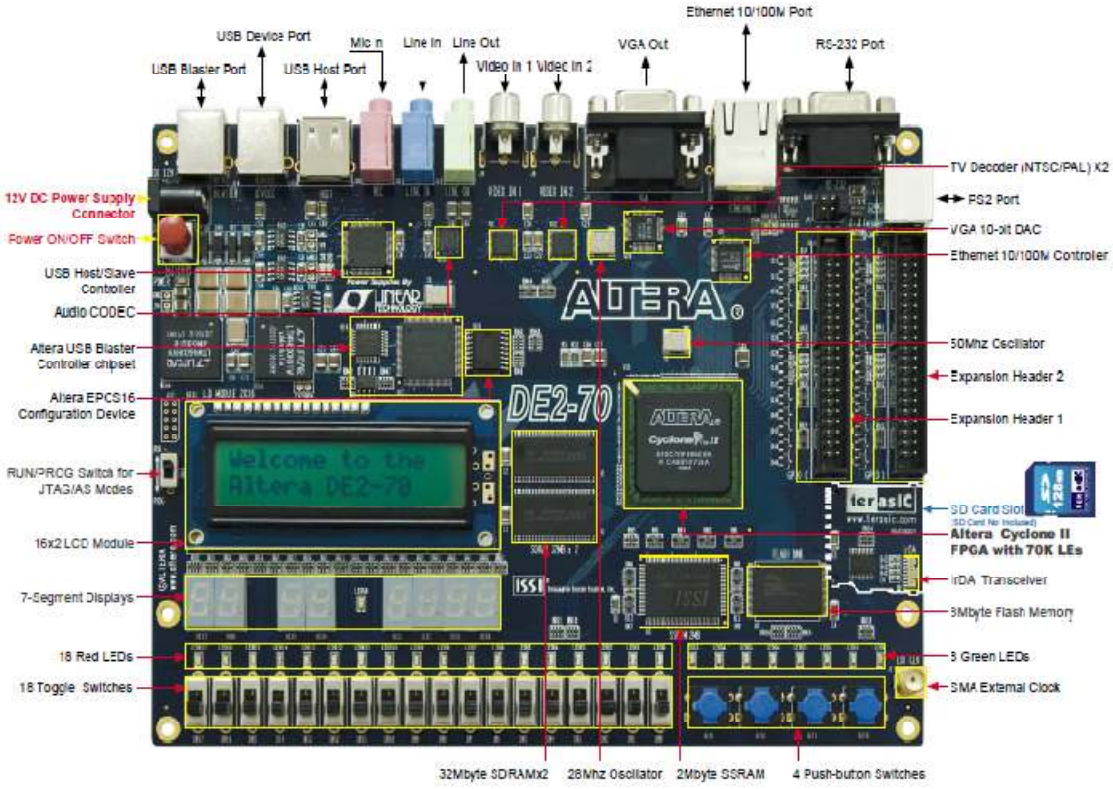

Figure 4.8 DE2-70 board top view.

CHAPTER FIVE
MULTIPROCESSOR SYSTEMS

5.1 Introduction to Multiprocessor Systems

Any system which incorporates two or more microprocessors working together to perform a task is commonly referred to as a multiprocessor system. Multiprocessor systems possess the benefit of increased performance, but nearly always at the price of significantly increased system complexity. For this reason, the use of multiprocessor systems has historically been limited to workstation and high-end PC computing using a complex method of load-sharing often referred to as symmetric multi processing (SMP). While the overhead of SMP is typically too high for most embedded systems, the idea of using multiple processors to perform different tasks and functions on different processors in embedded applications (asymmetrical) is showing increased interest. FPGAs provide an ideal platform for developing asymmetric embedded multiprocessor systems since the hardware can easily be modified (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

5.2 Hardware Design

In this section, the hardware design methodologies, autonomous and non-autonomous systems will be described. Hardware mutex core and shared system resources will be mentioned briefly.

5.2.1 Autonomous Multiprocessor

Autonomous multiprocessor systems contain multiple processors, these processors are completely autonomous and do not communicate with the others, much as if they were completely separate systems. Systems of this type are typically less complicated and pose fewer challenges, since the system's processors are incapable of interfering with each other's operation by design. Figure 5.1 shows a

block diagram of two autonomous processors in a multiprocessor system (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

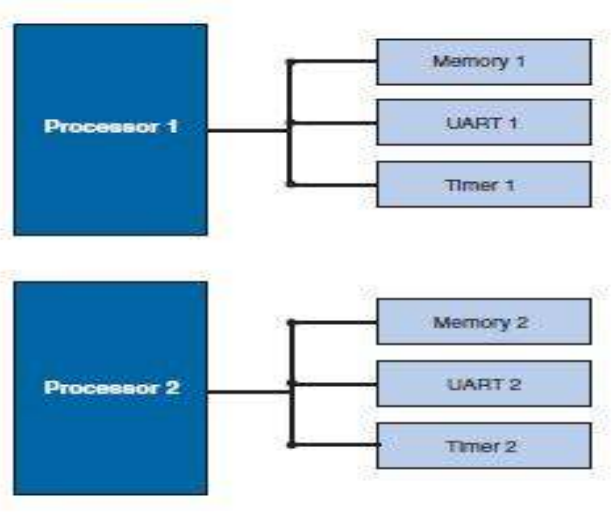

Figure 5.1 Autonomous Multiprocessor System (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

5.2.2 Non-Autonomous Multiprocessor

In this type of system, resources can be shared among processors. It is very useful to adopt resource-sharing mechanism in multiprocessor architectures, but it should be noticed the time which resource will be shared and how to cooperate each other among different processors while sharing the resources. Figure 5.2 shows a block diagram of a multiprocessor which includes two processors and the two processors share a single memory component (Y. C. Chen & C. Y. Tseng, 2008).

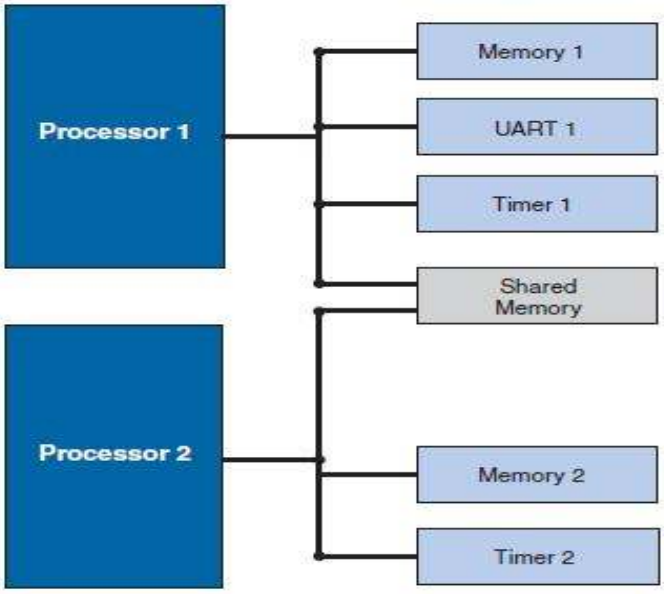

Figure 5.2 Multiprocessor System with Shared Resource (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

5.2.3 The Shared System Resources

5.2.3.1 Shared Memory

The most common type of shared resource in multiprocessor systems is memory. Shared memory can be used for anything from a simple flag whose purpose is to communicate status between processors, to complex data structures that are collectively computed by many processors simultaneously.

If a memory component is to contain the program memory for more than one processor, each processor sharing the memory is required to use a separate area for code execution. The processors cannot share the same area of memory for program space. Each processor must have its own unique .text, .rodata, .rwdata, heap, and stack sections.

If a memory component is to be shared for data purposes, its slave port needs to be connected to the data masters of the processors that are sharing the memory. Sharing data memory between multiple processors can be trickier than sharing instruction memory due to the fact that data memory can be written to as well as read. If one processor is writing to a particular area of shared data memory at the same time another processor is reading or writing to that area, data corruption will likely occur, causing application errors at the very least, and possibly a system crash. The processors sharing memory need a mechanism to inform one another when they are using a shared resource, so the other processors do not interfere (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

5.2.3.2 Shared Bus

The design of system bus plays a very important role for designing the architecture of System-on-a-Chip (SoC), especially while designing a multiprocessor system in a single chip. However, it will cause poor performance if we connect high-speed systems bus to low-speed peripherals during system designing period. To make data access more stable and faster, some processors will analyze the data you transfer and choose an appropriate system bus for data transmission (Y. C. Chen & C. Y. Tseng, 2008).

5.2.3.3 Shared Peripherals

The most important issue of shared peripherals in multiprocessor system is the management mechanism for interrupt handling. If a peripheral is allowed to interrupt all the processors that share it, there is no reliable way to make sure which processor will respond first and handle the interrupt. In this case, some multiprocessor systems share the same input peripheral have the problem of deciding which processor to process the data that the input peripheral send. It can be imaged that there is also a need to have a mechanism to protect the shared peripheral without causing errors. The tool chain Altera Corporation offers provides a Mutex Core which can be used

to protect the processors to mutually access the shared resources (Y. C. Chen & C. Y. Tseng, 2008).

5.2.4 Hardware Mutex Core

Multiprocessor environments can use the mutex core with Avalon interface to coordinate accesses to a shared resource. The mutex core provides a protocol to ensure mutually exclusive ownership of a shared resource. The mutex core provides a hardware-based atomic test-and-set operation, allowing software in a multiprocessor environment to determine which processor owns the mutex (Embedded Peripherals IP User Guide, Altera, 2010).

The mutex core acts as a shared resource, providing an atomic "test and set" operation in which a processor may test if the mutex is available and if so, acquire it in a single operation. When the processor is finished using the shared resource associated with the mutex, the processor releases the mutex. At this point, another processor may acquire the mutex and use the shared resource. Without the mutex, this kind of function would normally require two separate "test" and "set" instructions between which, another processor could also test for availability and succeed. This situation would leave two processors both thinking they successfully acquired mutually exclusive access to the shared resource when clearly they did not (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

5.3 Software Design

Running software on multiprocessor systems is much the same as for single-processor systems, but requires the consideration of a few additional aspects.

5.3.1 Program Memory

When creating multiprocessor systems, you may want to run the software for more than one processor out of the same physical memory device. Software for each

processor must be located in its own unique region of memory, but those regions are allowed to occupy the same physical memory device. For instance, imagine a two-processor system where both processors run out of SDRAM. The software for the first processor requires 128 Kbytes of program memory, and the software for the second processor requires 64Kbytes. The first processor could use the region between 0x0 and 0x1FFFF in SDRAM as its program space, and the second processor could use the region between 0x20000 and 0x2FFFF (Altera, Multiprocessor Tutorial).

Nios II and SOPC Builder provide a simple scheme of memory partitioning that allows multiple processors to run their software out of different regions of the same physical memory. The partitioning scheme uses the exception address for each processor, which is set in SOPC Builder, to determine the region of memory from which each processor will be allowed to run its software. Although the Nios II is ultimately responsible for the linking of the processors' software and determining where the software will reside in memory, Nios II looks at the exception addresses that were set for each processor in SOPC Builder to calculate where the different code sections will be linked. The Nios II provides each processor its own section within memory from which it can run its software. If the software for two different processors is linked to the same physical memory, then the exception address of each processor is used to determine the base address of the region which that processor's software can occupy. The end address of the region is determined by the next exception address found in that physical memory, or the end of that physical memory, whichever comes first (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

For any single or multiprocessor system, there are five primary code sections that need to be linked to fixed addresses in memory for each processor. These sections are:

- .text — the actual executable code
- .rodata — any constant data used in the execution of the code
- .rwdata — where read/write variables and pointers are stored

- heap — where dynamically allocated memory is located
- stack — where function call parameters and other temporary data is stored

These sections are shown in Figure 5.3 for a memory map of how these sections are typically linked in memory for a single processor Nios II system.

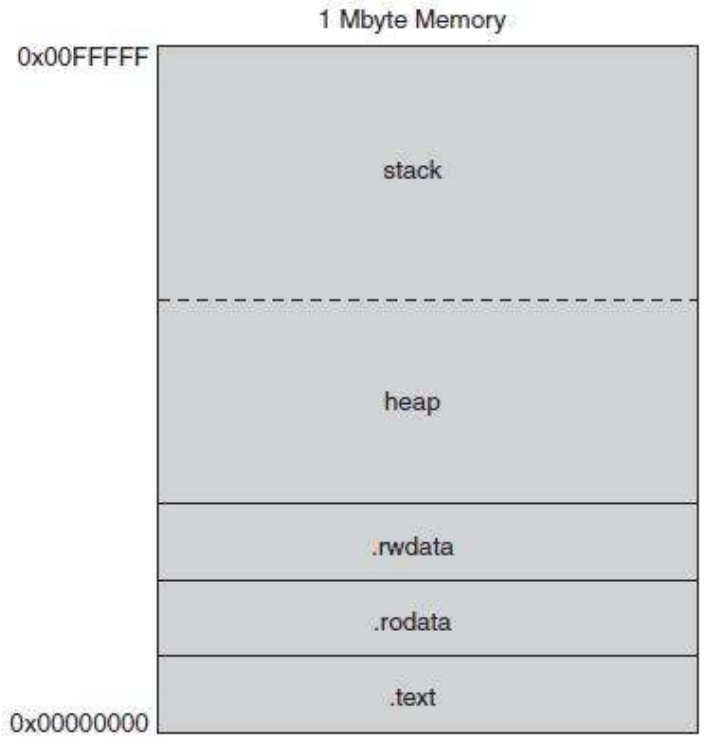

Figure 5.3 Single Processor Code Linked in Memory Map (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

In a multiprocessor system, it may be desirable to use a single memory to store all the code sections for each processor. In this case, the exception address set for each processor in SOPC Builder is used to define the boundaries between where one processor's code sections end and where the next processor's code sections begin. For instance, imagine a system where SDRAM occupies the address range 0x0 – 0xFFFFF and processors A, B and C each need 64 Kbytes of SDRAM to run their software. By using SOPC Builder to set their exception addresses 64 Kbytes apart in SDRAM. Partitioning of SDRAM is shown in Figure 5.4 for a memory map showing

how the SDRAM will be partitioned in this example system (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

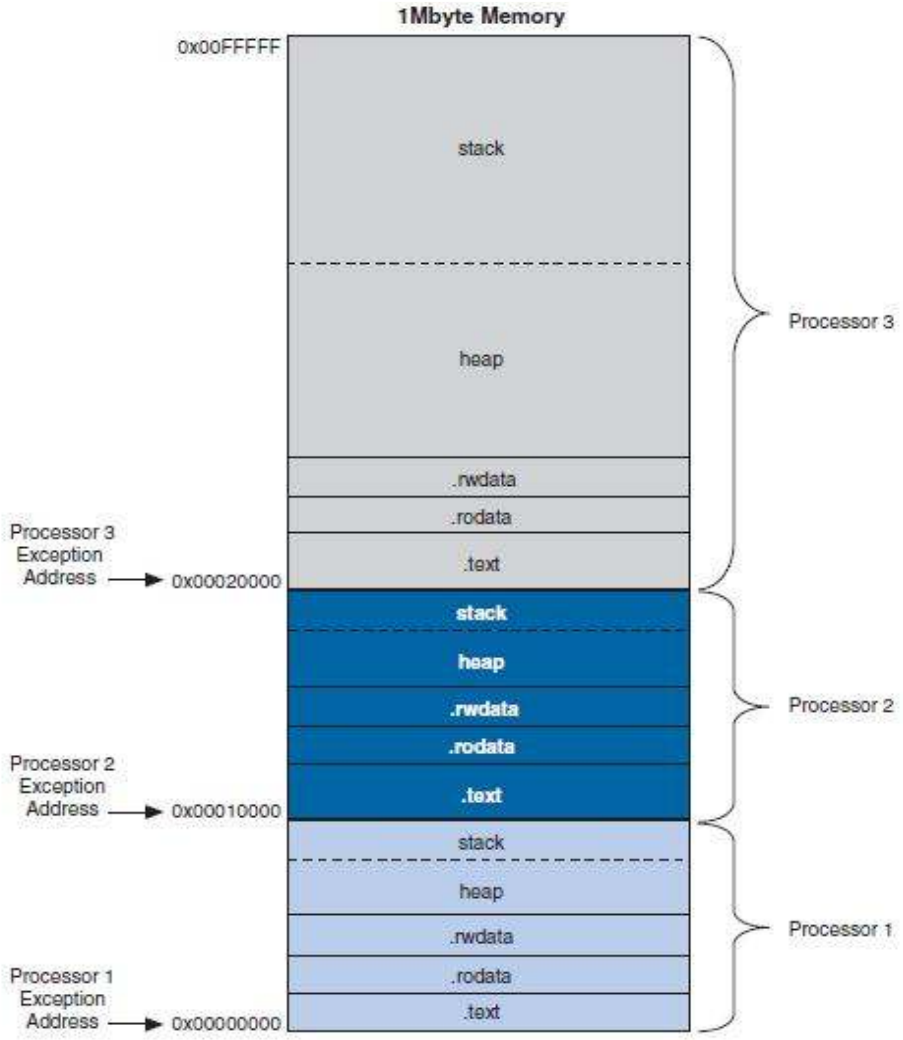

Figure 5.4 Partitioning of SDRAM Memory Map for Three Processors (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

Note that the lower six bits of the exception address are always set to 0x20. Offset 0x0 is where Nios II must run its reset code, so the exception address must be placed elsewhere. The offset of 0x20 is chosen because it corresponds to one instruction cache line. The 0x20 bytes of reset code initializes the instruction cache, and then branches around the exception section to the system startup code. Care must

be taken when partitioning a physical memory to contain the code sections of multiple processors. There are no safeguards in SOPC Builder or Nios II that guarantee you have provided enough code space for each processor's stack and heap in the partition. If inadequate code space is allotted in memory, the stack and heap may overflow and corrupt the processor's code execution (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

5.3.2 Boot Adresses

In multiprocessor systems, each processor must boot from its own piece of memory. More than one processor may not boot from the same bit of executable code at the same address in the same non-volatile memory. Boot memory can also be partitioned, much like program memory can, but the notion of sections and linking is not a concern as boot code typically just copies the real program code to where it has been linked in RAM, and then branches to the program code. To boot multiple processors out of separate regions with the same non-volatile memory device, simply set each processor's reset address to the location from where you wish to boot that processor. Be sure you leave enough space between boot addresses to hold the intended boot payload (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005). Flash device memory map is shown in Figure 5.5 for a memory map of one physical flash device from which three processors can boot.

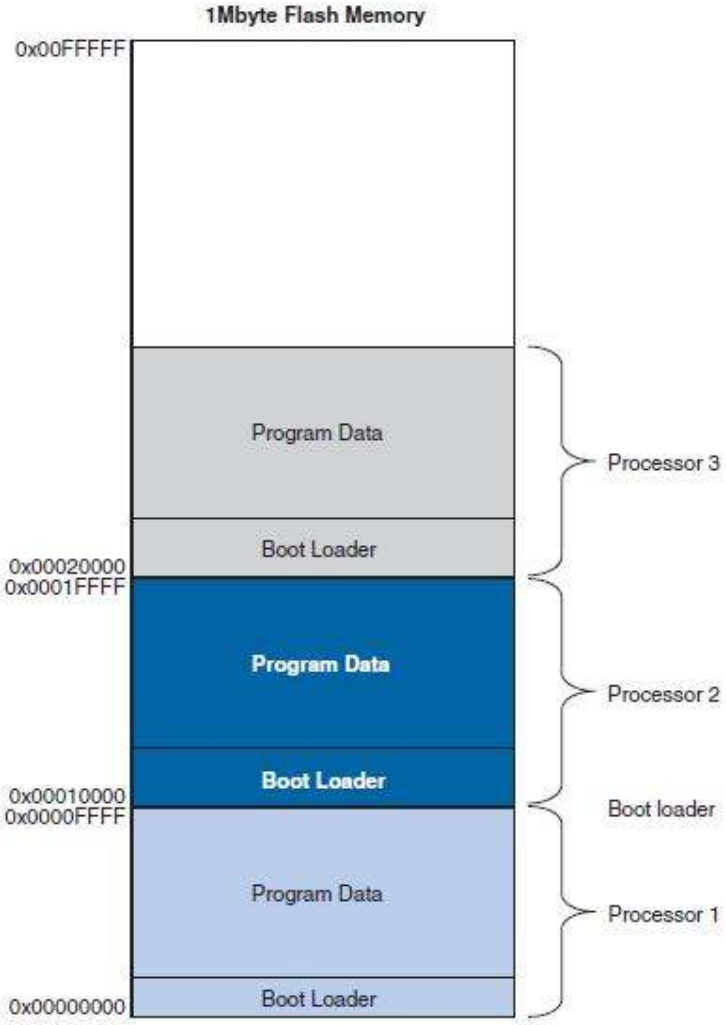

Figure 5.5 Flash Device Memory Map with Three Processors Booting (Creating Multiprocessor Nios II Systems Tutorial, Altera, 2005).

CHAPTER SIX
FPGA-BASED FACE RECOGNITION SYSTEM DESIGN

In this chapter, the implemented face recognition system will be described in detail. After, general overview of the face recognition system discussed in section 6.1, the results of a single-processor system will be assessed in section 6.2. The results of multi-processor system will be given in section 6.3. Finally, face recognition system performance will be evaluated in section 6.4.

6.1 Implementation of Face Recognition System Design on DE2-70

The face recognition system consists of several hardware and software parts. This section demonstrates the fundamental aspects of the system.

6.1.1 General Overview of Face Recognition System

In this study, the face recognition processes can be divided into two phases, processes implemented on Host PC and processes implemented on FPGA.

Processes implemented on Host PC are;

- Collecting database face images
- Image resizing
- Principle Component Analysis (PCA)
- Send database to FPGA

Block diagram of Host PC processes are shown in Figure 6.1.

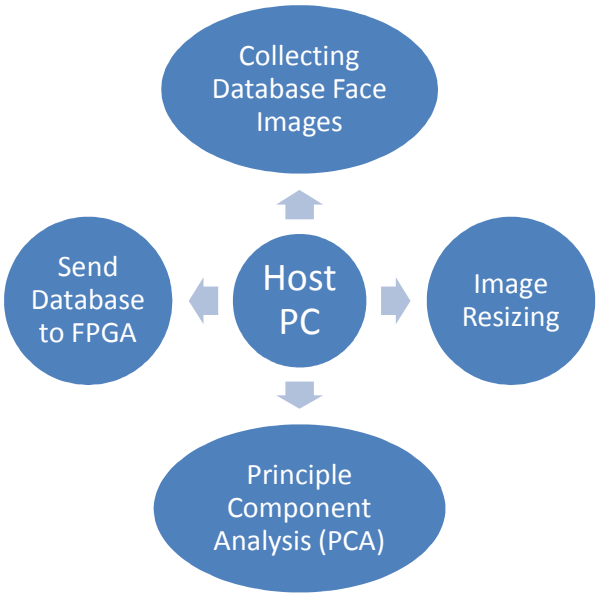

Figure 6.1 Block diagram of Host PC processes

Processes implemented on FPGA are;

- Recieve database from Host PC
- Normalization
- Neural network training
- Neural network testing

Block diagram of FPGA processes is shown figure 6.2.

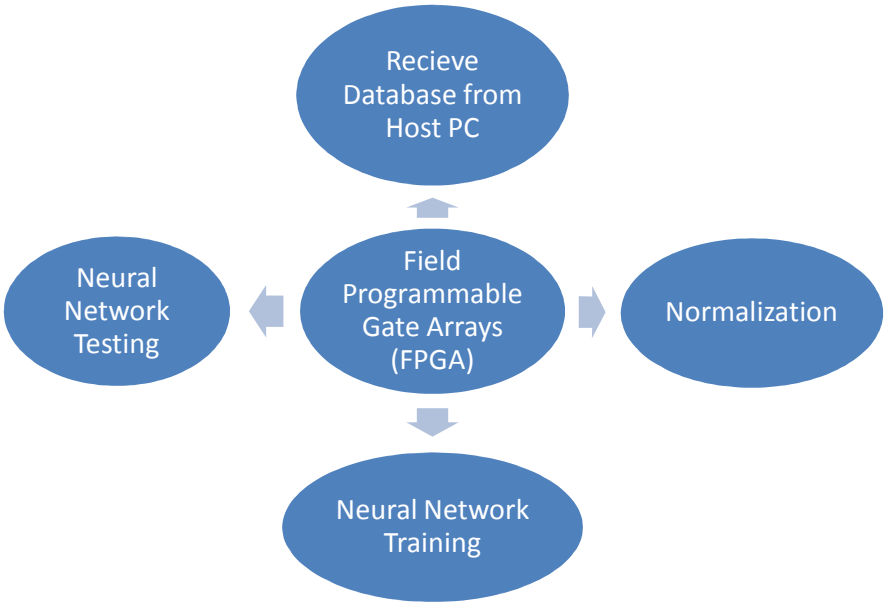

Figure 6.2 Block diagram of FPGA processes

ORL Database is used for this implementation of face recognition system. There are 40 people and 10 images for each person in this database. In this study, 60 images of 6 different person are used. The size of the images in this database is 112x92. Process on FPGA takes a long time because of large size of images. So there is a need to reduce the size of the images. In order to reduce the computational burden, the windowing method is applied. A window whose size is 3x3, is applied to the new images. Since the size of resized images is 40x40, 13x13 matrix is created for each image by windowing and mean processes. Windowing is applied to all images in the database. The row number of the feature matrix is M x N. Each image is represented by a 13x13 matrix so row number of the feature matrix is 169. Since 60 face images are used for database creation, the size of feature matrix is 169x60.

The most important step for neural network is feature extraction process. Principle Component Analaysis (PCA) algorithm is used for feature extraction process. Reducing size of data is provided by applying PCA to created feature matrix and independent variables are obtained. Eigenvectors are created by

applying PCA to feature matrix. Ten eigenvectors with largest variance are chosen to be given neural network for each image. It has been observed that, 10 eigenvalues are sufficient for proper training of neural network. Since image database has 60 images and 10 datas are obtained by PCA, finally 60x10 feature matrix is created. Randomly selected %70 of this feature matrix is used for training and %30 of this feature matrix is used for testing. This data is sent to FPGA by Universal Asynchronous Receiver/Transmitter (UART). The source code of HOST PC processes is in the Appendix with the folder name of "6_1_1_Database_PCA_Windowing_Matlab".

Code on FPGA consist of two part; training and testing. Training part begins to run after sending data are received by FPGA via UART. When data are sent from Host PC, code running on FPGA jumps to UART interrupt function and eigenvectors are saved in this UART interrupt function. Some preprocessing operations are applied to saved data. Normalization process is applied to this data to generate more correct results and to obtain code running faster. As a result of normalization process, elements of feature matrix are distributed between -1 and 1. Normalized data are given to neural network to be trained. Neural network algortihm is Feed Forward Backpropagation Algorithm. Neural network consists of 3 layers; input layer, hidden layer and output layer. Owing to the fact that 10 largest variance of data are chosen from PCA, input layer has 10 neurons. Hidden layer has 5 neurons and output layer has 1 neuron. The target vector is prepared to train the network to find out the person. Neural network training need to be repeated 2500 times to reach to desired learning rate. Neural network training is completed after 2500 epochs and neural network system is ready for testing.

Testing part starts to run after testing datas are sent from Host PC to FPGA via UART. Sent data are saved for testing in UART interrupt function. Normalization process is applied to testing data and data are scaled between -1 and 1. Owner of the images are identified by given normalized data to trained neural network.

6.1.2 Programs Used in the Project

In order to develop face recognition system the following programs have been used for preprocessing steps and FPGA programing:

- Matlab
- Quartus
- SOPC Builder
- Nios II

➢ Matlab:

Matlab (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, and Fortran.

➢ Quartus:

The Altera Quartus II design software provides a complete, multiplatform design environment that easily adapts to your specific design needs. It is a comprehensive environment for system-on-a-programmable-chip (SOPC) design. The Quartus II software includes solutions for all phases of FPGA design (Introduction to the Quartus II Software, Altera, 2010).

➢ SOPC Builder:

SOPC Builder is a powerful system development tool. SOPC Builder enables you to define and generate a complete system-on-a-programmable-chip (SOPC) in much less time than using traditional, manual integration methods. SOPC Builder is included as part of the Quartus II software (SOPC Builder User Guide, Altera, 2010).

- Nios II

Nios II is a 32-bit embedded-processor architecture designed specifically for the Altera family of FPGAs. Nios II incorporates many enhancements over the original Nios architecture, making it more suitable for a wider range of embedded computing applications, from DSP to system-control. The Nios II software development environment is called The Nios II integrated development environment (IDE). The Nios II IDE is based on the GNU C/C++ compiler and the Eclipse IDE, and provides a familiar and established environment for software development (Nios II Processor Reference Handbook, Altera, 2007).

6.1.3 Implementation Steps of Face Recognition System

6.1.3.1 Creating Database

ORL Database is used for this implementation of face recognition system. It contains a set of face images. There are 10 different images of each of 40 distinct subjects. For some subjects, the images are taken at different times, varying the lighting, facial details such as open/closed eyes, smiling/not smiling, glasses/no glasses etc. All the images have a dark homogeneous background with the subjects in an upright, frontal position. The size of each image is 92x112 pixels, with 256 grey levels per pixel. Figure 6.3 shows some example images from this database.

Figure 6.3 Some examples from ORL Database

In this study, 60 images of 6 different person are used and size of images is reduced.

6.1.3.2 Resizing Images

Size of the face image in ORL Database is 112x92. Firstly, the input image is taken from ORL image database. Then *imresize*, that is also one of the functions of MATLAB image processing toolbox, is used to resize the images. Figure 6.4.a shows the input image, Figure 6.4.b shows the resized image from 112x92 to 40x40. Note that, the process that is shown in Figure 6.4 is applied to all images sequentially on the database.

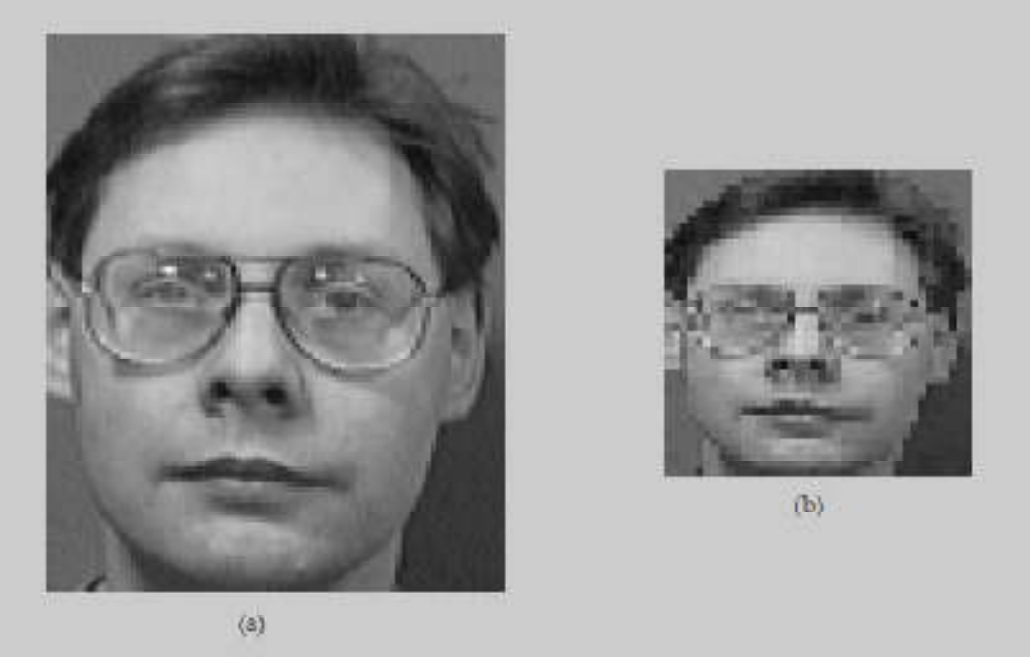

Figure 6.4(a) An original face image (b) the resized form of the face image

In order to reduce the computational burden, the windowing method is applied. After resizing the images, a window whose size is 3x3, is applied to the new images. The mean of each window become an element of the feature matrix. Since the size of resized images is 40x40, 13x13 matrix is created for each image by windowing and mean processes. Windowing is applied to all images in the database. The row number of the feature matrix is M x N. Each image is represented by a 13x13 matrix so row number of the feature matrix is 169. Figure 6.5 shows the new form of database images after windowing the whole image and taking the mean of the resultant windows.

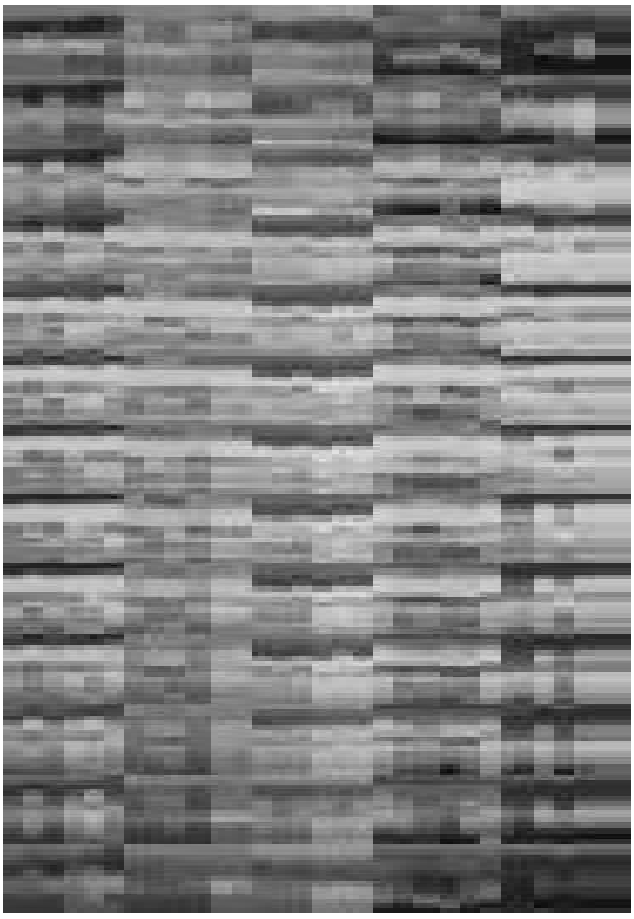

Figure 6.5 A face image after calculating mean of each window

6.1.3.3 Applying Principle Component Analysis (PCA)

The most important step for neural network is feature extraction process. Principle Component Analaysis (PCA) algorithm is used for feature extraction process. Images that are used for database are resized from 112x92 to 40x40. Since the size of resized images is 40x40, 13x13 matrix is created for each image by windowing and mean processes. Each image in the database are transformed into vectors and placed one column of new population matrix. For example, first image in the database is converted to a vector of 169x1, then this vector is put to first column of population matrix. This process is continued for all images. By processing for 60 images (6 people and 10 images for each people), population matrix is created with size of 169x60. This population matrix, *X*, is used for

creating the basis function. 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 169x169. Next step is to project population matrix, *X*, to face space by multiplying the face space vector, *A*. To reduce the computation time, only the first 10 eigenvalues are used. Selecting 10 datas are enough for neural network training. Since image database has 60 images and 10 datas are obtained by PCA, finally 60x10 feature matrix is created.

6.1.3.4 Sending Database to FPGA

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.

There are two main environment in the project, MATLAB and FPGA. UART provides the asynchronous communication between them. Communication in MATLAB is done by the help of MATLAB serial communication toolbox functions. Basically, four functions run sequentially in MATLAB. They are *serial* function to construct a serial port object associated with an existing port, *fopen* to connect the serial port object to FPGA, *fwrite* to write the feature matrix and other data to FPGA which is already connected to defined serial port object and *fclose* to disconnect the serial port object from FPGA. Baud rate is 9600 bits/second. A serial port object is defined in MATLAB to communicate with FPGA. Next, it is opened by *fopen.* If data transmission is required, *fwrite* is used. Also, if data reception is required, *fread* is used over the opened serial port object.

6.1.3.5 Receiving Database from Host PC

UART implementation in FPGA works to receive the data due to interrupts. When an interrupt is created to receive any data, interrupt service routine is called and data receive process starts. To use UART, 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. UART module is defined as a routine that serves when the serial port interrupt occurs. Note that, *UART_BASE* is the start address of UART. 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.

6.1.3.6 Normalization

Method of data normalization is a simple linear scaling of data. Data must be scaled into the range used by the input neurons in the neural network. This is typically the range of -1 to 1 or zero to 1. A linear scaling requires that the minimum and maximum values associated with the facts for a single data input be found. Let's call these values D_{min} and D_{max}, respectively. The input range required for the network must also be determined. Let's assume that the input range is from I_{min} to I_{max}. The formula for transforming each data value *D* to an input value *I* is:

$$I = I_{min} + (I_{max} - I_{min}) * (D - D_{min})/(D_{max} - D_{min}) \quad (6.1)$$

D_{min} and D_{max} must be computed on an input-by-input basis. This method of normalization will scale input data into the appropriate range but will not increase its uniformity.

6.1.3.7 Training Neural Network

Neural network algortihm is Feed Forward Backpropagation Algorithm. Neural network consists of 3 layers; input layer, hidden layer and output layer. Input layer has 10 neurons, hidden layer has 5 neurons and output layer has 1 neuron. Figure 6.6 shows block diagram of neural network structure.

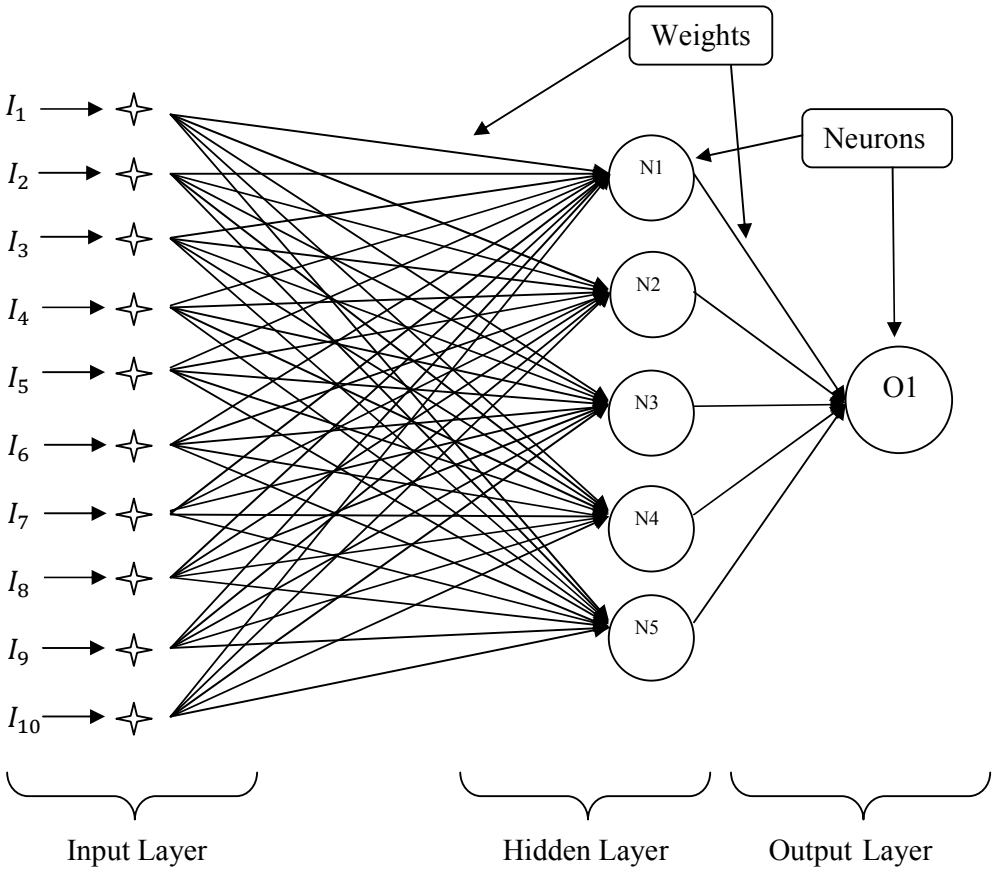

Figure 6.6 Block diagram of neural network structure.

Firstly, Inputs of neurons in hidden layer need to calculate to train neural network. Therefore, initial weights are assigned randomly. Inputs of neurons are;

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(\mathrm{n}) y_j(\mathrm{n}) \tag{6.2}$$

Where $v_j(n)$ is the induced local field of neuron and m is the total number of inputs applied to neuron j, $w_{ji}(\mathrm{n})$ is the synaptic weight connecting neuron *i* to neuron *j* at time *n* and $y_j(\mathrm{n})$ is the input to the neuron *j*. If neuron *j* is the first layer then the input is the general input of the network, if it is in a hidden layer then its input is the output of the previous layer and calculations are done in a standart way. But what the j^{th} neuron is in the output layer it is compared to the desired response then the backward pass accurs.

In the forward pass the weights remain unchanged through the network and output is calculated by neuron by neuron basis. Function signal $y_j(\mathrm{n})$ appering at the output of neuron j at iteration n is;

$$y_j(n) = \varphi_j(v_j(n)) \tag{6-3}$$

Activation function is used to calculate output of neuron. This activation function is hyperbolic tangent function. Hyperbolic tangent function is;

$$\varphi_j(v_j(n)) = atanh(bv_j(n)) , \qquad (a,b) > 0 \tag{6-4}$$

After output neuron is obtained, this output is compared with desired output value and system error is calculated. The error signal at the output is defined by;

$$e_j(n) = d_j(n) - y_j(n) \tag{6.5}$$

If number of output neuron has more than 1 neuron, error is;

$$\xi(n) = \frac{1}{2}\sum_{j=C} e_j^2\,(n) \tag{6-6}$$

All processes are repeated number of patterns times. Error is calculated for all patterns as follows;

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(n) \tag{6-7}$$

According to calculated error, system weights are reconfigured. All processes mentioned above are repeated until error reach to the desired value. After 2500 epochs, sistem reaches to desired value and the error is approximately 0,045.

6.1.3.8 Testing Neural Network

Testing images are taken from image database. Images that are used for testing are resized from 112x92 to 40x40. After by windowing and mean processes a 13x13 matrix is created for each image. Principle Component Analysis (PCA) is applied to testing data. 10 eigenvalues from PCA are used for testing. This data is sent to FPGA by UART.

Sent data are received by FPGA via UART. Then, testing phase begins. Sent data are saved for testing in UART interrupt function. Normalization process is applied to test data and data are scaled between -1 and 1. The owner of the images are identified by given 10 eigenvalues to trained neural network. The output of neural network system is calculated by processing feature vector with trained weights as described in section 6.1.3.7. In the training phase, indices are assigned as a target values as follows ;

- 1 to 10 images in the database: target value is 1
- 11 to 20 images in the database: target value is 2

- 21 to 30 images in the database: target value is 3
- 31 to 40 images in the database: target value is 4
- 41 to 50 images in the database: target value is 5
- 51 to 60 images in the database: target value is 6

In the testing phase, therefore, values need to be obtained vary between 1 and 6. If output value of network doesn't exceed the threshold value, owner of the image is identified. If output value of network exceeds the threshold value, owner of the image is not identified.

6.2 Single Processor Face Recognition System

6.2.1 Hardware Design

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 and the other system components are integrated to system design easily. 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
- 64 Mbyte SDRAM Controllers
- 4 Mbyte Flash Memory
- UART

There are 3 types of configurable Nios II CPU for Alera FPGAs. These are Nios II/f which is an optimized for the highest performance, Nios II/e which is an optimized for smallest size and Nios II/s which is balanced for performance and size. Nios II/f is selected for the system in this thesis.

External clock source that is provided by the crystal on the development kit is 50 MHz. The SDRAM on the DE2-70 operates at 85 MHz. In order to provide all components with same clock, three clocks are generated from the external 50 MHz clock by using Altera ALTPLL MegaWizard.

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.

SDRAM Controllers, *sdram_0* and *sdram_1*, are configured. 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.

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.

UART module allows communication between MATLAB and FPGA. 9600 baud rate is used for communication.

SOPC Builder screen at the end of configuration is shown in Figure 6.7.

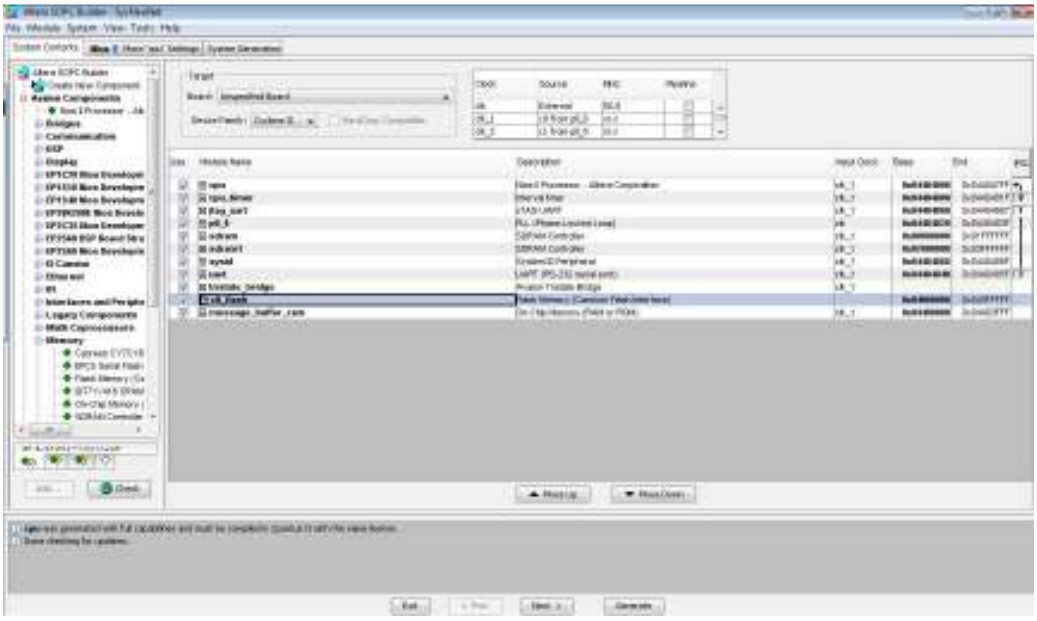

Figure 6.7 SOPC Builder screen at the end of configuration.

After configuring all components, reset addresses and exception addresses are arranged from SOPC Builder settings. Cpu settings are shown in Figure 6.8

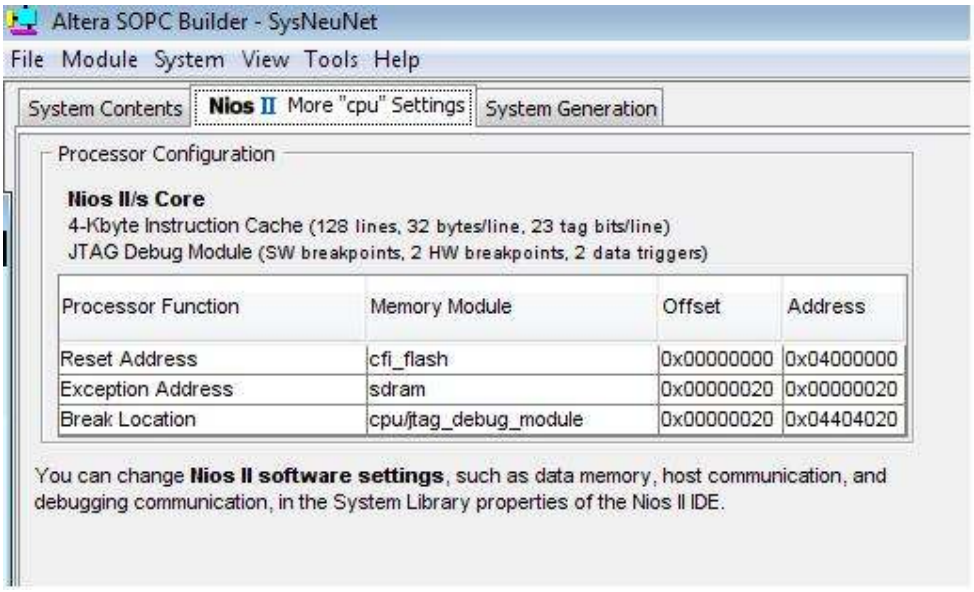

Figure 6.8 Cpu settings in processor configuration

The system is generated after configuring all components by using “Generate” button of SOPC Builder. After generating the system without any error, pins that are used on the board must be assigned. The hardware design of this system is completed with pin assignments. Figure 6.9 shows the pin assignments.

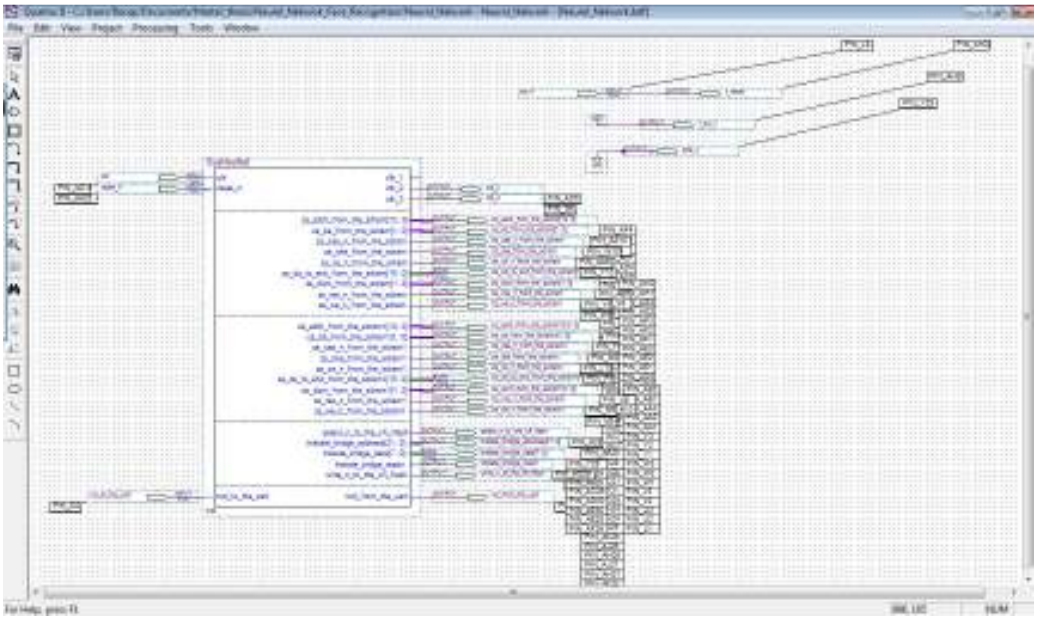

Figure 6.9 Pin assignments of system.

6.2.2 Software Design

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.

The images are received from MATLAB after resizing operation. UART core that is added to system during SOPC Builder design, listens to serial port and receive/transmit informations. Code on FPGA starts to run after sending data are recieved by FPGA via UART. When data are sent from Host PC, code running on FPGA branches to *void uart_isr(void* context,alt_u32 id)* interrupt function and eigenvectors are saved in this UART interrupt function. Some preprocessing

operations are applied to saved datas. Normalization process is applied to this datas in the *void Normalization(void)* function. Datas distributed between -1 and 1 are obtained as a result of this function. Normalized datas are given to neural network. Firstly, Initial weights must be assigned to start training. *void Initial_Weights(void)* function is used for this process. For training of neural network, there are 3 main functions. *void Calculate_Net(void)* calculates the current network output. According to the calculated error in the *Calculate_Net(),* weight outputs are reconfigured in the *void Weight_Changes_HO(void)* and weight inputs are reconfigured in the *void Weight_Changes_IH(void).* Testing part begins to run after testing datas are sent from Host PC to FPGA via UART. Sent datas are saved for testing in the *void Create_TestDatabase(void)* function. Normalization process is applied to testing datas and datas are scaled between -1 and 1 in the *void Normalization_Test(void)* function. Owner of the images are identified by given normalized datas to trained neural network. These results are displayed in the *void Display_TestResults(void).* Flow diagram of all this process is shown in Figure 6.10. The source code of single processor face receognition system is in the Appendix with the folder name of "6_2_2_Single_Processor_Face_Recognition_System_C_Code".

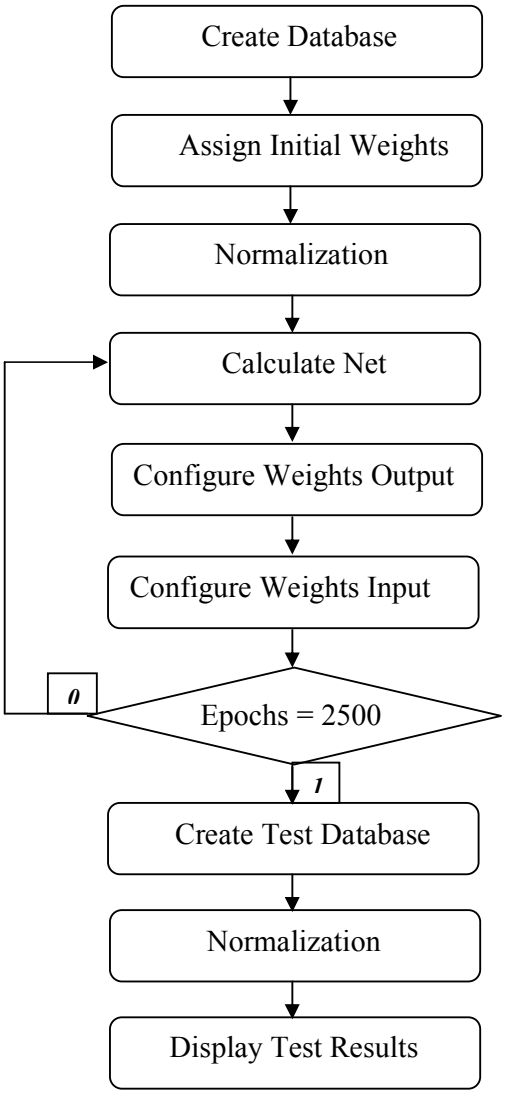

Figure 6.10 Flow diagram of single processor system

In order to select the train and the test samples, the cross-validation algorithm is implemented in MATLAB by using *crossvalind* function from Bioinformatics Toolbox of MATLAB. The output of the algorithm is a matrix and each row represents the images and each column represents the number of test subsystem. The elements of the cross-validation matrix are 0's and 1's. 1's show that the corresponding image must be selected for training and similarly 0's show that the corresponding image must be selected as test image. 20 test subsystems are created

by using cross-validation technique. The source code of cross-validation is in the Appendix with the folder name of "6_2_2_Cross_Validation_Matlab".

Using cross-validation matrix, all processes in flow diagram are repeated 20 times for the generalization of the results. The neural network recognition rate is evaluated by calculating the mean of these results.

Time of all this processes, number of processor, epoch number, time of system, system error and recognition rate are given in the Table 6.1.

Table 6.1 Results of single processor system

Number of Processor	Epoch Number	Time of System Training	System Error (RMS Error)	Recognition Rate
1 CPU	2500	49 min. 10 sec.	0.045	95.3%

The results indicate that although the recognition rate is acceptable, the process is slower. In the next section, the multiprocessor approach will be introduced to speed up the system.

6.3 Multiprocessor Face Recognition System

6.3.1 Hardware Design

Multi-processor system consists of the following elements;

- Nios II CPU1
- Nios II CPU2
- The Hardware Mutex Core
- On-Chip Memory
- Phase Locked Loop (PLL)
- JTAG UART
- Interval Timer

- 64 Mbyte SDRAM Controllers
- 4 Mbyte Flash Memory
- UART

All components except hardware mutex core are mentioned in section 6.2.1.We are building a multiprocessor system that shares a data memory between processors, so it is essential that a hardware mutex component be included to enable protection of that memory from data corruption. Multiprocessor environments can use the mutex core with Avalon interface to coordinate accesses to a shared resource. The mutex core provides a protocol to ensure mutually exclusive ownership of a shared resource. The mutex core provides a hardware-based atomic test and set operation, allowing software in a multiprocessor environment to determine which processor owns the mutex.

Most important step for multiprocessor system design is to connect instruction and data masters. All the resources that are shared between processors in the system need to be connected using SOPC Builder's connection matrix. Using the connection matrix, *sdram* and *sdram1* are connected to the instruction and data masters for each processor, allowing two processors to access *sdram* and *sdram1*. All the connection dots for the *sdram* and *sdram1* should be solid black.

ext_ram_bus is connected to the instruction and data masters for each processor, allowing two processors to access external RAM and flash memory. All the connection dots for *ext_ram_bus* should be solid black.

message_buffer_mutex is connected to the data masters for two processors and two instruction masters are disconnected, allowing two processors to access *message_buffer_mutex.*

message_buffer_ram is connected to the data masters for two processors and two instruction masters are disconnected, allowing all three processors to access that

memory only as data memory. SOPC Builder screen at the end of configuration is shown in Figure 6.11.

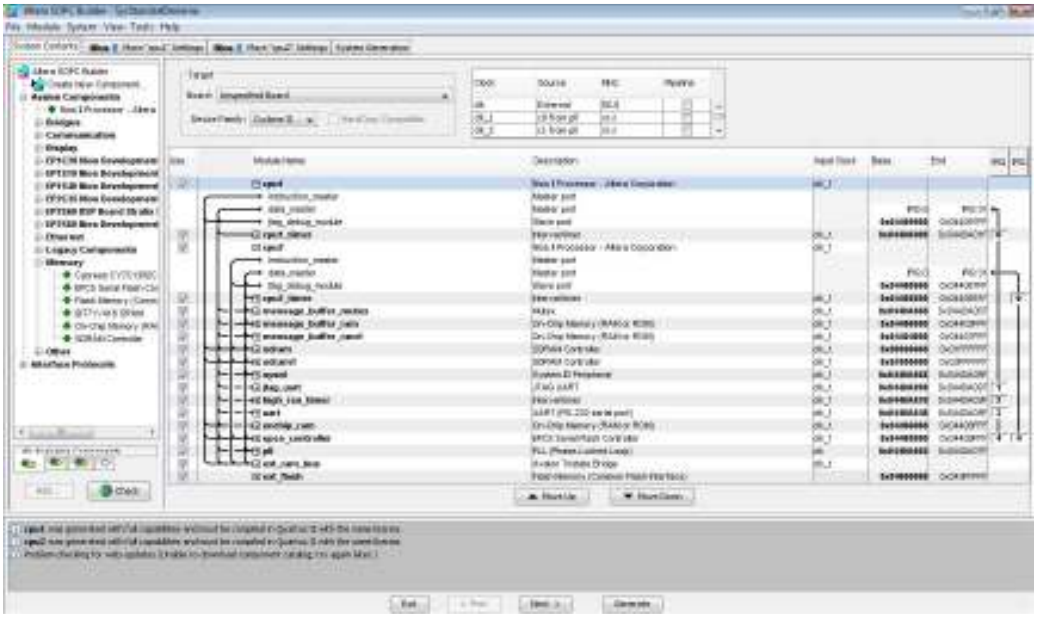

Figure 6.11 All components of multiprocessor system

After configuring all components, reset addresses and exception addresses are arranged from SOPC Builder settings. Cpu1 and Cpu2 settings are shown in Figure 6.12 and 6.13.

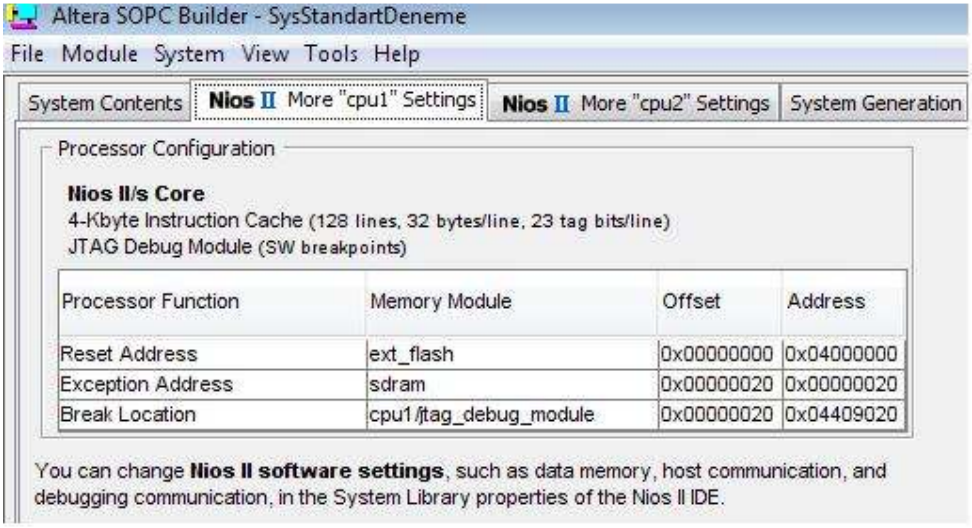

Figure 6.12 Cpu1 settings in processor configuration

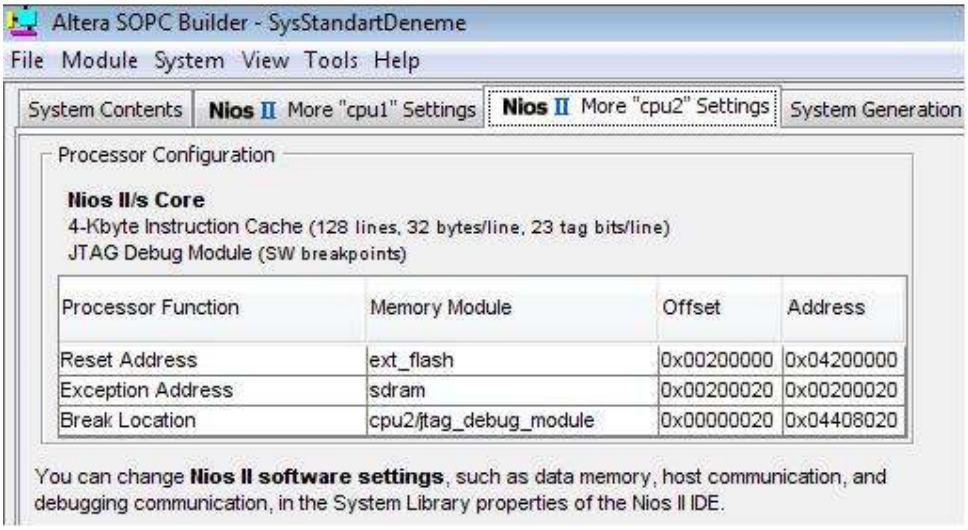

Figure 6.13 Cpu2 settings in processor configuration

6.3.2 Software Design

To obtain faster system, multiprocessor system is designed. Thus, training time is reduced. Code on FPGA described in the section 6.2.2 need to be designed in accordance with 2 processors system. Hence, an approach different from sequential system of classical processors should be developed and the system should be run in parallel.

In the single processor system, the processor recognizes the images of six different people means a classification problem with 6 classes. In the multiprocessor system, tasks should be assigned to each processor in the system and these tasks should reduce the burden of processing. For this purpose, instead of classifying 6 classes at once, each processor makes a sub-classification between the predefined classes. Burden of each processor of multiprocessor system is provided to be less than burden of processor of single processor system. Which classes will be assigned to processors is determined by using hierarchical classification/clustering approach. Its purpose is to train similar classes in the same processor.

6.3.2.1 Hierarchical Clustering

Hierarchical clustering groups data over a variety of scales by creating a cluster tree or dendrogram. The tree is not a single set of clusters, but rather a multilevel hierarchy, where clusters at one level are joined as clusters at the next level. This allows you to decide the level or scale of clustering that is most appropriate for your application. Results of hierarchical clustering are given Figure 6.14.

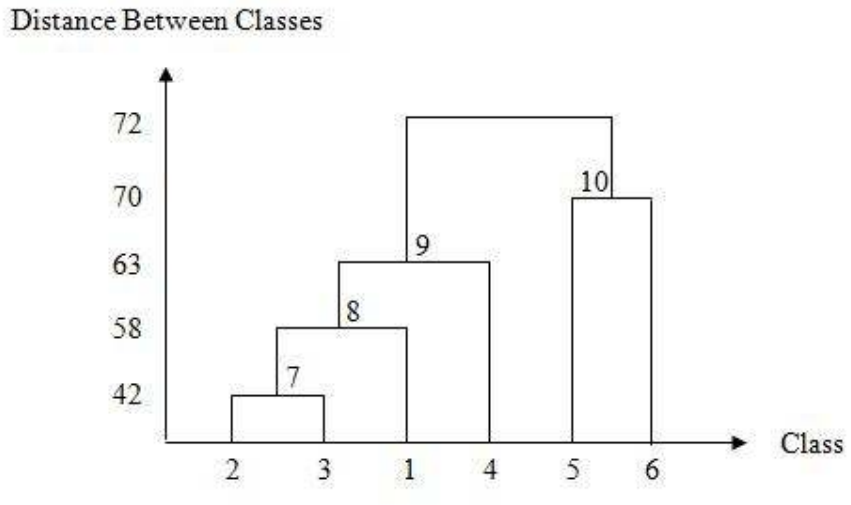

Figure 6.14 Results of hierarchical clustering

In the figure, the numbers along the horizontal axis represent the indices of the classes. Vertical axis indicates the distance between the classes. The link representing the cluster containing classes 2 and 3 has a height of 42. The link representing the cluster that groups class 1 together with classes 2 and 3 (which are already clustered as object 7) has a height of 58. Cluster that groups class 4 together with classes 2, 3 and 1 (which are already clustered as object 8) has a height of 63. Classes 5 and 6 (which are already clustered as object 10) has a height of 70. Cluster that groups class 5 and 6 (which are already clustered as object 10) together with classes 2, 3, 1 and 4 (which are already clustered as object 9) has a height of 72.

According to hierarchical classification results summarized in Figure 6.14, since 1st, 2nd and 3rd classes are closer to each other in the feature space, they are

given to Cpu1, and 4th, 5th and 6th classes are given to Cpu2. The source code of hierarchical classification is in the Appendix with the folder name of "6_3_2_1_Hierarchical_Classification_Matlab".

6.3.2.2 Multiprocessor System Software

Code on FPGA is described in the section 6.2.2. Multiprocessor system need to be designed in accordance with 2 processors system. Therefore, the system should be run in parallel. Flow diagram of multiprocessor system is shown in Figure 6.15.

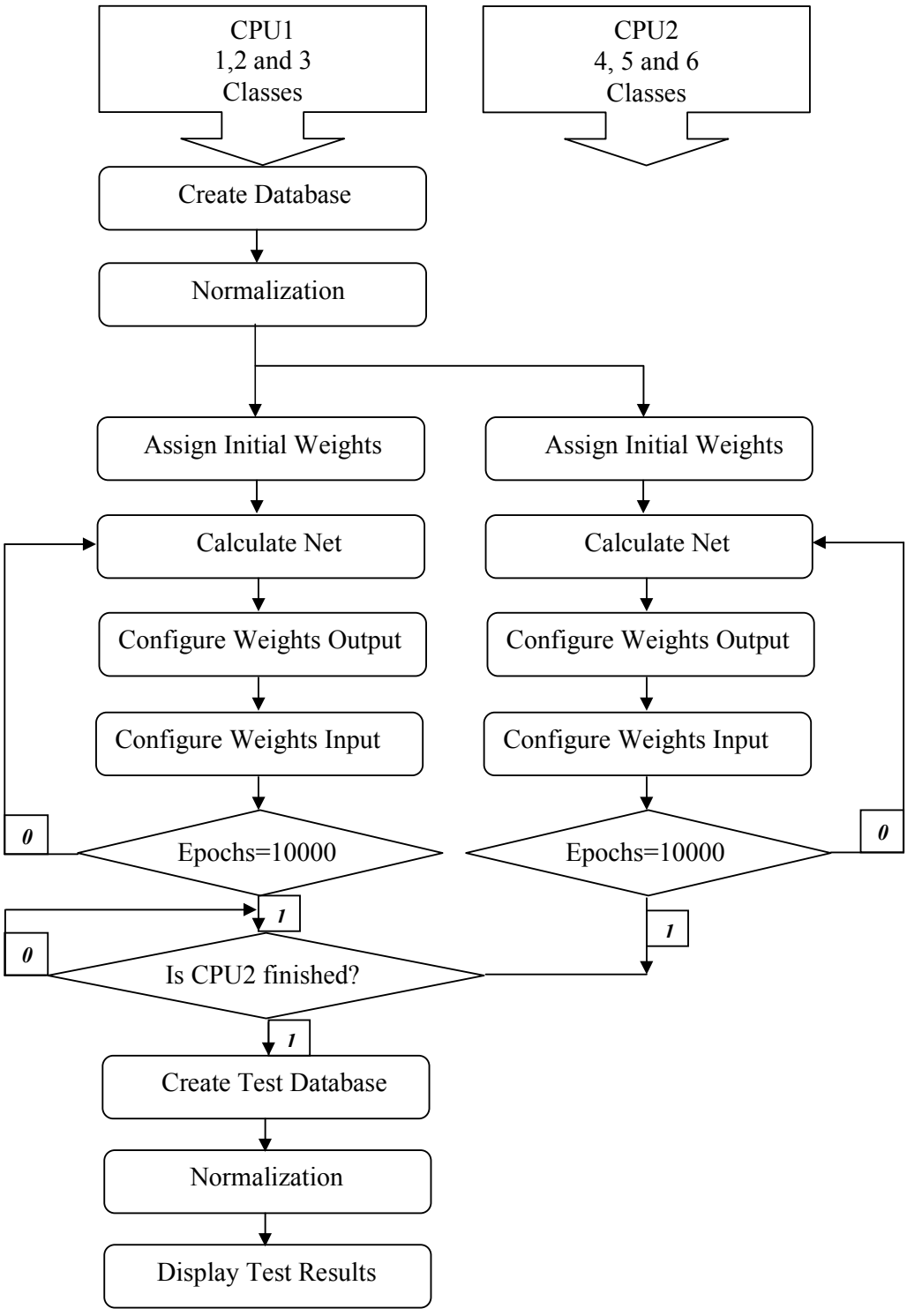

Figure 6.15 Flow diagram of multiprocessor system

Code on FPGA begins to run after data are recieved by FPGA via UART. When data are sent from Host PC, the code running on FPGA jumps to *void uart_isr(void* context,alt_u32 id)* interrupt function in the cpu1 and classes are saved in this UART interrupt function to the shared memory. Thus, 2 processors can access to saved data. Cpu1 can access to datas of 1, 2 and 3 classes and Cpu2 can access to data of 4, 5 and 6 classes. Normalization process is applied to all classes in the *void Normalization(void)* function in the cpu1. Initial weights must be assigned to start training for each processor. *void Initial_Weights(void)* function is used for this process. Initial weights for 1, 2 and 3 classes are generated in the cpu1 and initial weights for 4, 5 and 6 classes are generated in cpu2. For training of neural network, *void Calculate_Net(void), void Weight_Changes_HO(void)* and *void Weight_Changes_IH(void)* functions are run for each processor. These functions are repeated until error reach to the desired value. After training part is completed, testing part begins to run and testing data of all classes are sent from Host PC to FPGA via UART. Sent data are saved for testing in the *void Create_TestDatabase(void)* function in the cpu1. Normalization process is applied to test data of all classes in the *void Normalization_Test(void)* function. Owner of the images of all classes are identified by given test data of all classes to calculated weights in the processors. Test data are given to both of the networks trained in cpu1 and cpu2. Each cpu produces its output individually. The distance of each outcome to the target values are calculated. Two distances are compared and the result of processor which has smaller distance is the result of multiprocessor system. Due to reducing error, epoch number is increased to 10000. These results are displayed in the *void Display_TestResults(void).* The source code of multiprocessor face receognition system is in the Appendix with the folder name of "6_3_2_2_Multiprocessor_Face_Recognition_System_C_Code".

Using cross-validation, all processes in the flow diagram in the Figure 6.15 are repeated 20 times for the generalization of the results. The neural network recognition rate is evaluated by calculating the mean of these results.

Time of all this processes, number of processor, epoch number, time of system, system error and recognition rate are given in the Table 6.2.

Table 6.2 Results of multiprocessor system

Number of Processor	Epoch Number	Time of System Training	System Error (RMS Error)	Recognition Rate
2 CPU	10000	1hour 43 min. 45 sec.	CPU1 - 0.020 CPU2 - 0.021	93.9%

6.4 General Overview of Face Recognition System Performance

In a previous MSc study completed by E.Dilcan and Gökhan Çetin (E. Dilcan & G. Çetin, 2010), a face recognition system using principle component analysis (PCA) and Euclidean Distance has been implemented on FPGA. The 10 face images of 5 people have been taken for creating the database. %70 of database is used for training and %30 of database is used for testing. According to this study, recognition rate is 93.3%.

In this study, the previous study has been improved by implementing recognition task by artificial neural network. The obtained improved recognition rate is 95.3% using multilayer feed forward backpropagation network with one hidden layer of 5 neurons.

Moreover, a multiprocessor system is designed to speed up the system. Multiprocessor system is 47.2% faster than single processor system as multiprocessor system compares to the single processor system. The final multiprocessor recognition rate is 93.9%. The recognition results of different approaches are summarized in the Table 6.3.

Table 6.3 Results of different face recognition approaches

Face Recognition Approach	Recognition Rate
PCA + Euclidean Distance	93.3%
PCA + Neural Network + 1 CPU	95.3%
PCA + Neural Network + 2 CPU	93.9%

The time efficiency and recognition performances of single processor and multiprocessor systems are compared in the Table 6.4. In the multiprocessor system, epoch number is selected 10000, due to reducing error. The distance of each outcome to the target values are calculated for each processor. Two distance are compared and the result of processor which has smaller distance is the result of multiprocessor system. But rarely, wrong result is decided as a result of the comparison. Hence, recognition rate of multiprocessor system is less than recognition rate of single processor system, because of comparison error.

Table 6.4 Results of single processor system and multiprocessor system

Number of Processor	Epoch Number	Time of System Training	System Error (RMS Error)	Recognition Rate
1 CPU	10000	3 hour 16 min. 10 sec.	CPU1 - 0.020	95.3%
2 CPU	10000	1 hour 43 min. 45 sec.	CPU1 - 0.020 CPU2 - 0.021	93.9%

To complete face recognition processes in less time, a multiprocessor system with two processors is designed. It is observed that multiprocessor system is 47.2% faster than single processor system. Time of system training increases with increasing epoch number. But system error do not change significantly. According to epoch number, time of single processor and multiprocessor systems are compared in the Table 6.5.

Table 6.5 Time of single processor system and multiprocessor system

Epoch Number	Number of Processor	Time of System Training
2500	1 CPU	49 min. 10 sec.
	2 CPU	25 min. 50 sec.
5000	1 CPU	1 hour 36 min. 35 sec.
	2 CPU	50 min. 55 sec.
10000	1 CPU	3 hour 16 min. 10 sec.
	2 CPU	1 hour 43 min. 45 sec.

CHAPTER SEVEN
CONCLUSIONS

7.1 Summary of the Project

Biometrics is the method of recognizing a person based on a physiological or behavioral characteristics. Biometric technologies are becoming the foundation of an extensive array of highly secure identification and personal verification solutions. In this thesis, face recognition system was implemented.

The face recognition system acquires images from face database; the images were preprocessed to reduce size of images. Then PCA was applied as a feature extraction method and the neural network was trained with these features. In this study, multilayer perceptron network was used with one hidden network. The feed forward backpropagation algorithm was used to train neural network. In the recognition phase, the same preprocessing and feature extraction steps were repeated. Finally, the features were sent to the trained neural network to find the owner of the image.

In the first implementation, all face recognition processes were run on single processor system. The general performance of the system was calculated after generating 20 subsystems by using cross-validation technique. For this implementation, recognition rate of the face recognition system was 95.3%. To complete face recognition processes in less time, a multiprocessor system with two processors was designed. It was observed that multiprocessor system was 47.2% faster than single processor system. According to multiprocessor system, recognition rate of the face recognition system was 93.9%.

7.2 Advantages – Disadvantages

Field Programmable Gate Array (FPGA) is offering design flexibility and high performance system integration. FPGA is providing cost and power reductions, while increasing performance and functionality. DE2-70 development kit is used in this

thesis. DE2-70 is used for implementing face recognition system by using high level language. Face recognition program is written by C/C++ via Nios II IDE. Using FPGA development kit brings some advantages such as learning simulation and compilation of the projects on Quartus II, adding components to FPGA via SOPC Builder tool and learning parallel configuration of a system by designing multiprocessor system.

Since the neural network recognition is used in this project, the system is improved with the generalization property of the neural networks. Thus, the system has a better recognition performance. Also, because of the multiprocessor implementation, the system is trained faster.

Disadvantage of system hadware is the cost of the FPGA development board and the use of host computer. For the integration of the system into real life, the system should be fully implemented on FPGA chip.

7.3 Troubleshooting

Face recognition system is designed by using one processor and two processors. System with more than two processors cannot be design on this development board because of hardware limitations. For both *cpu1* and *cpu2*, if *sdram* is selected for *Program memory, Read-only data memory, Read/write data memory, Heap memory, and Stack memory* in the library property settings, the time consumption of the system cannot be reduce because of memory density. When *sdram* is selected for cpu1 and *sdram1* is selected for cpu2, reduction is obtained. Therefore, owing to presence of 2 sdram on FPGA, face recognition system which contains maximum two processors is experimented.

7.4 Cost Analysis

DE2–70 development kit is purchased with a cost of $400. The DE2–70 package includes DE2-70 board and other tools like USB cable for FPGA programming and

control, CD-ROMs containing Altera's Quartus® II Web Edition and the Nios® II Embedded Design Suit Evaluation Edition software.

7.5 Future Work

For future work, a portable face recognition system can be developed by adding camera for image capture. The preprocessing and feature extraction stages can be implemented on FPGA. The results of system can be displayed on screen via VGA out connector on FPGA. The number of processors for face recognition system can be increased and 4 or 8 processors can be experimented to increase speed of system.

REFERENCES

Agarwal M., Kumar M., Jain N., & Agrawal H. (2010). Face Recognition using Principle Component Analysis, Eigenface and Neural Network. *Signal Acquisition and Processing, 310-314.*

Altera (2005). *Creating Multiprocessor Nios II Systems Tutorial*, Altera Corporations.

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

Altera (2010). *Introduction to the Quartus II Software*, Altera Corporations.

Altera (2007). *Nios II Processor Reference Handbook*, Altera Corporations.

Altera (2010). *SOPC Builder User Guide*, Altera Corporations.

Barlett M. S., Movellan J.R., & Sejnowski T. J. (2002). Face Recognition by Independent Component Analysis, *IEEE Trans. on Neural Networks, 13, 6, November 2002, 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.

Dilcan E. (2010). *Face and Fingerprint Recognition on Field Programmable Gate Array*. Dokuz Eylul University, Graduate School of Natural and Applied Sciences, Master Thesis, November 2010.

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

Gokhan Ç. (2010). *Face and Speech Recognition on Field Programmable Gate Array*. Dokuz Eylul University, Graduate School of Natural and Applied Sciences, Master Thesis, November 2010.

Goldstein A. J., Harmon L. D., & Lesk B. (1971). *Identification of Human Faces*. Proc. IEEE, May 1971, 59, 5, 748-760.

Haykin S. (2001). *Neural Networks A Comprehensive Foundation, 2nd Editon,* Ontario, Canada.

Hung A., Bishop W., & Kennings A. (2005). *Symmetric Multiprocessing on Programmable Chips Made Easy*. Proceeding of the Design, Automation and Test in Europe Conference and Exhibition, 1530-1591/05.

Hyvarinen A. (1999). Survey on independent component analysis. *Neural Computing Surveys 2* : (94-128). Helsinki University of Technology, Finland.

Jain A. K., Ross A., & Prabhakar S. (2004). An Introduction to Biometric Recognition. *IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Image- and Video-Based Biometrics, 14, 1, January 2004.*

Kuon I., Tessier R., & Rose J. (2007). *FPGA Architecture: Survey and Challenges.* Foundations and Trends in Electronic Design Automation, 2, 2, 135-253, 2007.

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

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, 1, 657-660.

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

Moghaddam B., Jebara T., & Pentland A. (2000). Bayesian Face Recognition. *Pattern Recognition, 33, 11, 1771-1782, November, 2000.*

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.

Sirovich L., & Kirby M. (1987). A Low-Dimensional Procedure for the Characterization of Human Faces. *J. Optical Soc. Am. A, 1987, 4, 3*, 519-524.

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

Tseng C. Y., & Chen Y. C. (2008). *Design and Implementation of Multiprocessor System on a Chip (MPSoC) on FPGA*. Tatung University, Department of Computer Science and Engineering, Thesis for Master of Science, July 2008.

Tumeo A., Regazzoni F., Palermo G., Ferrandi F., & Sciuto D. (2010). *A Reconfigurable Multiprocessor Architecture for a Reliable Face Recognition Implementation*. Design, Automation & Test in Europe Conference & Exhibition.

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

Wayman J. L. (2001). Fundamentals of Biometric Authentication Technologies. *International Journal of Image and Graphics, 1, 1, 93-113.*

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 6.1.1 General Overview of Face Recognition System

6_1_1_Database_PCA_Windowing_Matlab

Section 6.2.2 Software Design

6_2_2_Single_Processor_Face_Recognition_System_C_Code

Section 6.2.2 Software Design

6_2_2_Cross_Validation_Matlab

Section 6.3.2.1 Hierarchical Clustering

6_3_2_1_Hierarchical_Classification_Matlab

Section 6.3.2.2 Multiprocessor System Software

6_3_2_2_Multiprocessor_Face_Recognition_System_C_Code