FPGA IMPLEMENTATION OF PROBABILISTIC NEURAL NETWORK

Main Project Report

Submitted in partial fulfillment of the requirements

for the award of the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

of

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

By

Jithin K S
Sujith K S
Sunil V
Vijay Viswanath S

Department of Electronics and Communication Engineering

Government Model Engineering College

Thrikkakara

Cochin – 21

2007

GOVERNMENT MODEL ENGINEERING COLLEGE

THRIKKAKARA, COCHIN –21

C E R T I F I C A T E

This is to certify that the project report entitled

. . . .

Submitted by

. . . .

. . ..

is a bona£ide account of the work done by him under our supervision

Guide(s) Co-ordinator Head of the Department

ABSTRACT

The project is intended to implement a neural network classifier called Probabilistic Neural Networks (PNN) in FPGA (Field Programmable Gate Array).Most of the neural networks are implemented in software, not fully utilizing the parallel architecture of neural networks. The hardware implementation accounts for faster highly parallel neural systems.PNN is one of the best pattern classifiers which is easier and faster to train compared to other neural networks. This neural network can be applied to various areas like alphanumeric character recognition, object identification tasks in robots etc. Here intention is to test the PNN with character (alphabets) recognition task.

ACKNOWLEDGEMENT

First of all, we would like to thank God Almighty for showering his blessings, without which the project could not have been successfully completed.

We would like to express our gratitude to our Principal, Prof. Dr. Suresh Kumar P for allowing us to operate on the project.

We would also like to thank our Head of the Department, Prof. T.K.Mani for his valuable help that lead to the timely completion of the project

We would also like to thank our project coordinator, Mrs. Sumitha Mathew., Lecturer, Electronics Department for her valuable help.

We express our heartfelt gratitude to our project guide, Mr. Vinu Thomas, Lecturer, Department of Electronics, for channeling us in the right path at all times.

We also extend our gratitude to all our teachers, lab technicians and friends who have always been very helpful and co-operative.

TABLE OF CONTENTS

ABSTRACT i
ACKNOWLEDGEMENT ii
1 INTRODUCTION 1
1.1 ARTIFICIAL NEURAL NETWORKS 1
1.2 WHY ANN IN HARDWARE? 1
2 PROBABILISTIC NEURAL NETWORKS 3
2.1 WHY PNN? 3
2.2 PNN ARCHITECTURE 5
2.3 PNN CLASSIFIER 6
2.4 PNN TRAINING 9
2.5 PNN ALGORITHM 9
3 APPLICATION AREA 11
3.1 CLASSIFICATION 11
3.2 CHARACTER RECOGNITION 11
3.3 IMAGE DIGITIZATION 12
3.4 LEARNING MECHANISM 13
3.5 PERFORMANCE ISSUES 14
4 FIELD PROGRAMMABLE GATE ARRAYS 16
4.1 GENERAL DESCRIPTION 16
4.2 FPGA ARCHITECTURE 16
4.2.1 CYCLONE II DEVICE ARCHITECTURE 17
4.3 FPGA DEVELOPMENT BOARD 19
4.4 FPGA DESIGN FLOW 20
4.4.1 DESIGN ENTRY 20
4.4.2 SYNTHESIS 20
4.4.3 SIMULATION 20
4.4.4 IMPLEMENTATION 22
4.4.5 PROGRAMMING 22

5 SOFTWARE TOOLS......23

5.1 QUARTUS II 6.0 SOFTWARE......23

6 VERILOG......25

6.1 BACKGROUND......25

6.2 IMPORTANCE OF VERILOG HDL......25

6.3 VERILOG LANGUAGE......26

7 PROJECT DESCRIPTION......28

8 SYSTEM BLOCK DIAGRAM......29

9 HARDWARE MODELLING......30

9.1 PROPOSED DESIGN......30

9.2 INPUT LAYER......30

9.3 RADIAL BASIS LAYER......31

9.4 SUMMATION LAYER......32

9.5 COMPETITIVE LAYER......32

9.6 DYNAMICALLY ADDDING NEW TRAINING VECTORS......33

9.7 ROADBLOCKS......33

9.8 HARDWARE DESCRIPTION BLOCK DIAGRAM......35

10 FPGA COMPUTATIONAL FLOW......37

11 SOFTWARE IMPLEMENTATION......38

11.1 MATLAB 7.1......38

11.2 GUI INTERFACE......38

11.3 PARALLEL PORT INTERFACE......40

11.4 EXPONENTIAL LUT TRANSMISSION......41

11.5 LOADING TRAINING VECTORS......42

11.6 LOADING TEST INPUT......43

11.7 DYNAMICALLY ADDING NEW TRAINING VECTORS......43

11.8 ROADBLOCKS......43

12 CONCLUSION......44

APPENDIX 1: CYCLONE II FPGA..........45

1.1 LOGIC ELEMENTS..........45

1.2 LOGIC ARRAY BLOCKS..........46

1.3 I/O STRUCTURE AND FEATURES..........47

APPENDIX 2: FPGA DEVELOPMENT BOARD..........49

REFERENCES..........52

LIST OF FIGURES AND TABLES

Figure 2.1 PNN ARCHITECTURE..5
Figure 2.2 ACTIVATION VS DISTANCE...7
Figure 2.3 RADIAL BASIS TRANSFER FUNCTION...8
Figure 2.4 WEIGHTED SUM OF RADIAL BASIS TRANSFER FUNCTIONS.................8
Figure 2.5 EFFECT OF SIGMA ON CLASSIFICATION 9
Figure 3.1 IMAGE DIGITIZATION..12
Figure 3.2 CHARACTER VARIANTS OF THE SAME CLASS 14
Figure 4.1 FPGA ARCHITECTURE 17
Figure 4.2 CYCLONE II EP2C20 DEVICE BLOCK DIAGRAM...............................18
Figure 4.3 DE1 BOARD...19
Figure 4.4 FPGA DESIGN FLOW...21
Figure 5.1 DESIGN FLOW 23
Figure 8.1 SYSTEM BLOCK DIAGRAM……………………………………………...29
Figure 9.1 HARDWARE DESCRIPTION BLOCK DIAGRAM………………………….36
Figure 10.1 FPGA COMPUTATIONAL FLOW…………………………………………....37
Figure 11.1 SCREENSHOT OF THE MATLAB GUI INTERFACE……………………….39
Figure A1.1 A CYCLONE II LE……………………………………………………………..45
Figure A1.2 CYCLONE II LAB Structure…………………………………………………...47
Figure A1.3 CYCLONE II IOE STRUCTURE……………………………………………….48
Figure A2.1 DEI BLOCK DIAGRAM…………………………………………………………….49

TABLE 11.1 PIN ASSIGNMENTS OF THE D-TYPE 25 PIN PARALLEL PORT CONNECTOR……………………………………………………………………41

1 INTRODUCTION

Man's quest for automation has been never-ending. In this journey, man found one new processor that can abide all existing processing algorithms, for certain sophisticated applications. That was his brain itself. Then he began thinking of implementing his brain explicitly to suit his application needs. That was the beginning of the ANN or Artificial Neural Networks.

1.1 ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANN) are biologically inspired; that is, they are composed of elements that perform in a manner that is analogous to the most elementary functions of the biological neuron. Neural Network is the term used to describe the group of neuron in the mammalian brain. A neuron is a unit of the brain processing block, thousands of these is interconnected in the human brain and functions simultaneously. ANN is essentially the way to simulate this biological neural network artificially.

One of the main differences, between human brain operation and a conventional computer system, is that the human brain can learn and process new information. A conventional computer system would depend on the programmer to tell it what to do and what decisions to make for a predefined situation possibility. Therefore learning by example is an important feature in an ANN. Once trained, an ANN should be able to make decisions and generate output based on its training 'experience'. Common application of ANN includes image recognition, data validation, and forecasting.

1.2 WHY ANN IN HARDWARE?

Artificial Neural Networks have a highly parallel architecture .In this, input patterns are fed in parallel and the whole computations are also done in a parallel fashion for each vector. But nowadays most of the ANN is implemented in software, which is run from a machine which does everything sequentially. The main advantage that a suitable neural network hardware implementation can offer is the ability to directly implement the parallelism and concurrency of the neural network algorithm.

The hardware implementation thus will be many times faster than its software counterpart. This provides us with quicker output especially in computationally expensive algorithms. Here we propose the hardware implementation of the Probabilistic Neural Network (PNN) Paradigm.

2 PROBABILISTIC NEURAL NETWORKS (PNN)

The probabilistic neural network was proposed by **Donald Specht [1]**. His network architecture was first presented in two papers, Probabilistic Neural Networks for Classification, Mapping or Associative Memory and Probabilistic Neural Networks, released in 1988 and 1990, respectively. This network provides a general solution to pattern classification problems by following an approach developed in statistics, called **Bayesian classifiers**. Bayes theory, developed in the 1950's, takes into account the relative likelihood of events and uses a priori information to improve prediction. The network paradigm also uses **Parzen Estimators** which were developed to construct the probability density functions required by Bayes theory.

The PNN uses a **supervised training** set to develop distribution functions within a pattern layer. These functions, in the recall mode, are used to estimate the likelihood of an input feature vector being part of a learned category, or class. The learned patterns can also be combined, or weighted, with the a priori probability, also called the relative frequency, of each category to determine the most likely class for a given input vector. If the relative frequency of the categories is unknown, then all categories can be assumed to be equally likely and the determination of category is solely based on the closeness of the input feature vector to the distribution function of a class.

2.1 WHY PNN?

The Bayesian classifier and its outgrowth, the probabilistic neural network (PNN) have been used successfully to solve a diverse group of classification problems. These include vector-cardiogram interpretation (spect 1967),radar/target identification (Huynen, Bjorn, and spect 1967), and, more recently, hull-to-emitter correlations on radar hits (Maloney and spect 1989) Although the PNN has not yet been fully evaluated relative to other techniques, if it lives up to its promise it may emerge as the method of choice for many difficult classification problems.

Compared with backpropagation[6], the PNN offers the following advantages:

1. Rapid training: The PNN process is as much as five orders of magnitude faster than backpropagation. The days or weeks of iterative training of a backpropagation network are replaced by little more than reading in the training set.
2. With enough training data a PNN is guaranteed to converge to a Bayesian classifier (the usual definition of optimality), despite an arbitrarily complex relationship between the training vectors and the classification. There is no such guarantee with backpropagation; long training periods can terminate in a local optimum that may be an unsatisfactory solution.
3. The PNN algorithm allows data to be added or deleted from the training set without lengthy retraining; where as, any modification to backpropagation training set will generally require a repetition of the entire training process. This characteristic of the PNN makes it more compatible with many real world problems. As with human experience, network learning is often a continuous process. Additional input data collected during operation can improve the performance of the classifier – if it can be incorporated without major difficulties.
4. PNN provides an out put including the amount of evidence upon which it is basing its decision. A backpropagation system provides no such confidence indication. Given an input that is quite unlike it has been trained on, it may provide a totally erroneous answer.

The PNN overcomes many of the objections to backpropagation, yet retains its most important characteristics, including the following:

1. **Learning** The network is capable of learning the most complicated relationships between the training vectors and their correct classifications. Arbitrary nonlinear decision boundaries in a space of any dimensionality can be extracted solely from the training set without human intervention.
2. **Generalization** inputs that are similar, but identical to those in the training set will, within limits, be correctly classified. Also erroneous, noisy, or incomplete

training or data inputs do not have a disproportionate effect on classification accuracy.

3. **Concurrency** The inherently parallel nature of algorithms allows efficient use of multiprocessor systems. Since the classification time is almost exactly inversely proportional to the number of processors, arbitrary speed improvements can be achieved by adding the hardware. With modular VLSI implementation, systems providing high performance/ price ratios are feasible.

2.2 PNN ARCHITECTURE

PNN consists of feed-forward four layers, that is, an input layer, a kernel layer, a summation layer, and a decision layer.

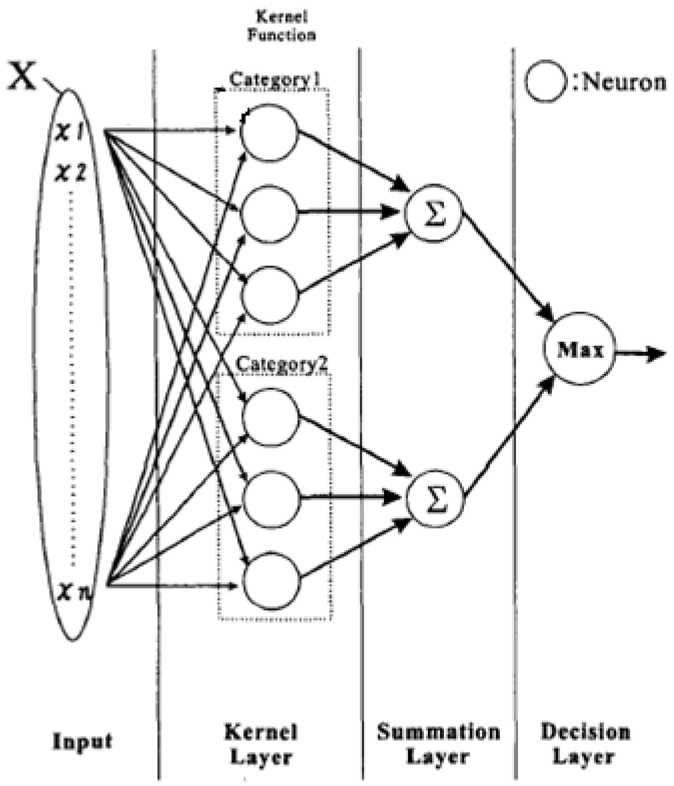


Figure 2.1 PNN ARCHITECTURE

All PNN networks have four layers:

1. **Input layer** — There is one neuron in the input layer for each predictor variable. The input neurons (or processing before the input layer) standardizes the range of

the values by subtracting the median and dividing by the interquartile range. The input neurons then feed the values to each of the neurons in the hidden layer.

2. **Kernel layer / Hidden layer** — This layer has one neuron for each case in the training data set. The neuron stores the values of the predictor variables for the case along with the target value. When presented with the *x* vector of input values from the input layer, a hidden neuron computes the Euclidean distance of the test case from the neuron's center point and then applies the RBF kernel function using the sigma value(s). The resulting value is passed to the neurons in the pattern layer.
3. **Pattern layer / Summation layer** —In this layer, there is one pattern neuron for each category of the target variable. The actual target category of each training case is stored with each hidden neuron; the weighted value coming out of a hidden neuron is fed only to the pattern neuron that corresponds to the hidden neuron's category. The pattern neurons add the values for the class they represent
4. **Decision layer** —This layer compares the weighted votes for each target category accumulated in the pattern layer and uses the largest vote to predict the target category.

2.3 PNN CLASSIFIER

The general classification problem is to determine the class membership of a multivariate random vector ***X*** into one of *N* possible groups *N=(n1,n2...nm)*.If we know the *Probability Density Functions(PDF*), $h_n(X)$, for all groups, then according to the Bayes optimal decision rule we classify ***X*** into population i if the following inequality holds ,

$$p_i \, c_i \, h_i(X) \ > \ p_j \, c_j \, h_j(X)$$

where,

p = the prior probability of membership in a group *n*, and

c = the cost of misclassification into group *n*.

Although the implementation is very different, probabilistic neural networks are conceptually similar to *K-Nearest Neighbor* (k-NN) models. The basic idea is that a predicted target value of an item is likely to be about the same as other items that have close values of the predictor variables.

The distance is computed from the point being evaluated to each of the other points, and a *radial basis function* (RBF) (also called a *kernel function*) is applied to the distance to compute the weight (influence) for each point. The radial basis function is so named because the radius distance is the argument to the function.

***Weight* = RBF(*distance*)**

The farther some other point is from the new point, the less influence it has.

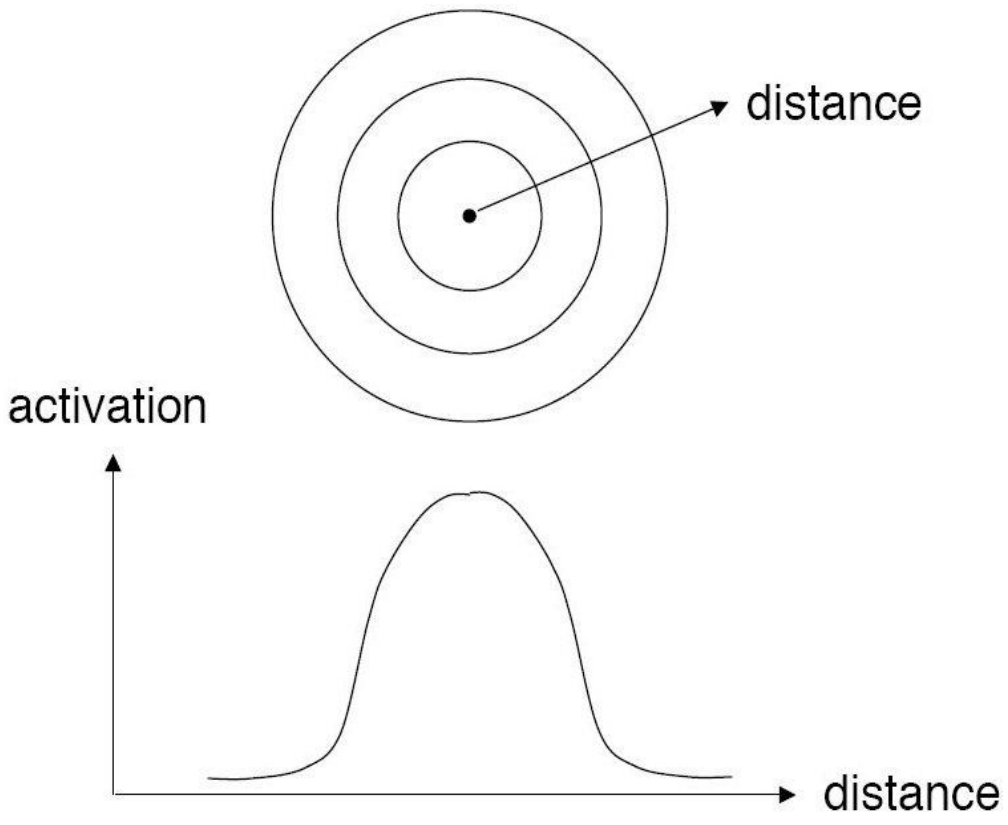


Figure 2.2 Activation Vs Distance

Different types of radial basis functions could be used, but the most common is the Gaussian function:

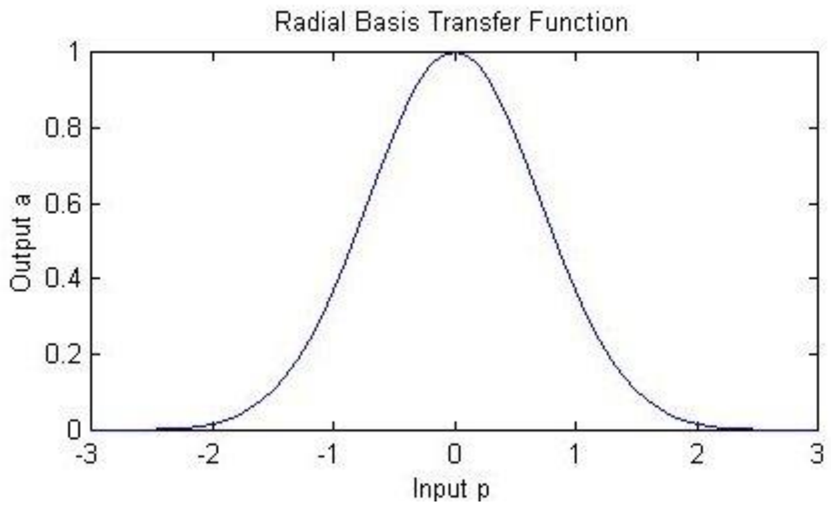


Figure 2.3 Radial basis transfer function

If there is more than one predictor variable, then the RBF function has as many dimensions as there are variables.

The best predicted value for the new point is found by summing the values of the other points weighted by the RBF function.

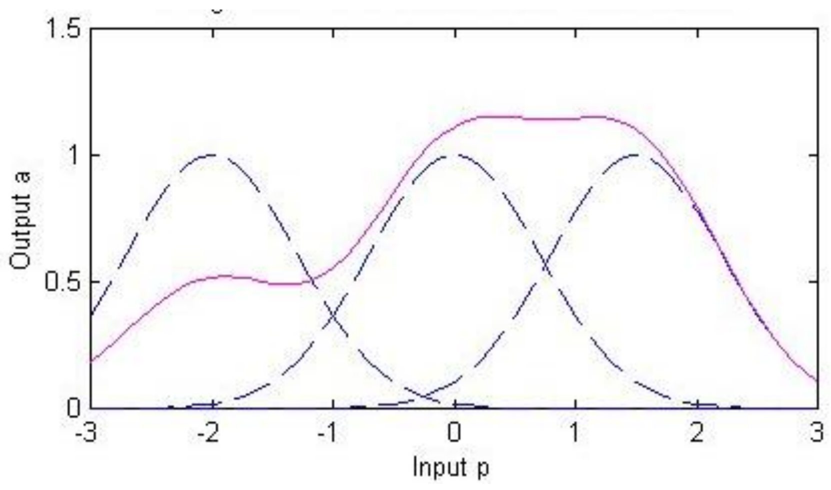


Figure 2.3 Weighted sum of Radial basis transfer functions

The peak of the radial basis function is always centered on the point it is weighting. The sigma value (σ) of the function determines the spread of the RBF function; that is, how quickly the function declines as the distance increased from the point.

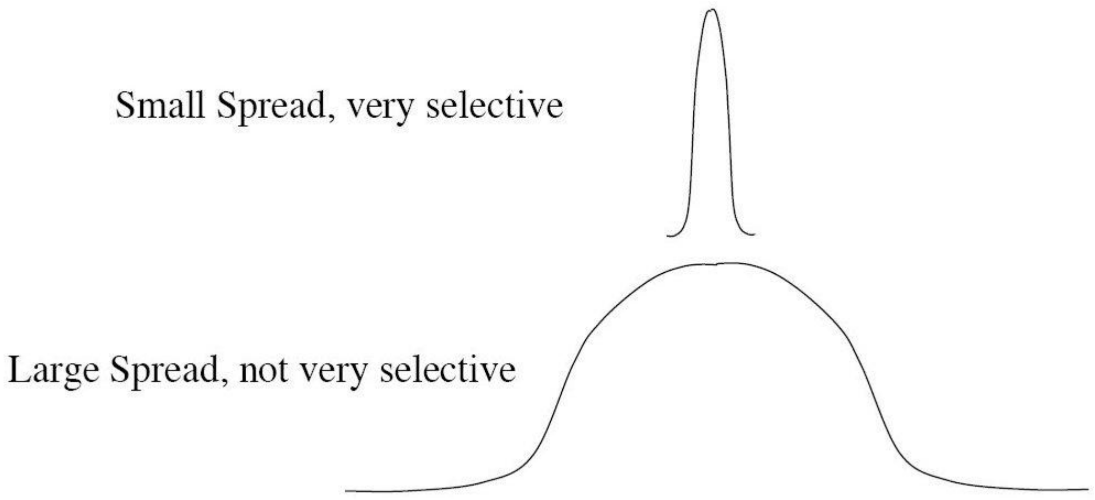

Small Spread, very selective
Large Spread, not very selective

Figure 2.4 Effect of sigma on classification

With larger sigma values and more spread, distant points have a greater influence.

2.4 PNN TRAINING

Training process is very fast in PNN.It just involves loading the training vectors corresponding to each class as the input weights of the kernel layer. Also the vectors corresponding to a single class have to be mapped together.

2.5 PNN ALGORITHM

The PNN uses Parzen probability density function (pdf) estimators which asymptotically approach the underlying parent density provided that it is smooth and continuous. One way the PNN does this, is by using sums of spherical Gaussian functions centered at each training vector to estimate the class pdfs. Consequently, the PNN is able to make a classification decision in accordance with the Bayes' strategy for decision rules and provide probability and reliability measures for each classification. Training involves choosing a single suitable smoothing factor which is the common bandwidth far all the Gaussians. The network is tolerant of erroneous training vectors, and sparse data samples can be adequate for optimal performance. It is both easy to use and fast to train for moderately sized data bases,

A sum of spherical Gaussian basis functions, as defined equation (I), can be used to implement the PNN.

$$f_i(x) = \sum_{j=1}^{M_i} \exp\left[\frac{-(x-x_{ij})^T(x-x_{ij})}{2\sigma^2}\right] \text{......................(I)}$$

where :

T indicates the transpose.

i indicates the class number.

j indicates the pattern number.

$\boldsymbol{x_{ij}}$ the jth training vector from class i.

x the test vector.

$\boldsymbol{\sigma}$ the common basis function bandwidth.

$\boldsymbol{M_j}$ the number of training vectors in **class *i*.**

P the input vector dimensions.

N the number of classes.

If the losses associated with making an incorrect decision and the a priori probability of occurrences of each class are consider to be equal then test vector x is said to belong to class *i* if , $\mathbf{f_i(x)} > \mathbf{f_k(x)}$ for $k \neq i$.

The PNN network is simply a parallel 3-layer feed-forward architecture which implements Bayes' decision function estimators for each class from its representative samples. The input units are the distribution points for thc vector elements and they supply the same values to each of the pattern units for each class. Each pattern unit represents a training vector x_{ij} and performs the operation defined by equation (I). These operations are summed to produce $f_i\ (x)$ for each class *i*. The highest $f_i(x)$ value determines the class decision for the unknown vector x.

3 APPLICATION AREA

3.1 CLASSIFICATION

Classification has become one of the prime fields of research because of its wide scope of application. The act of pattern classification is more vivid and is used in real-time image and signal processing. Whether it be oceanographic research or the medical field, an efficient pattern classifier has immense demand. Here as a starting stage we test the neural network, with a task of classification which is recognition of alphabetic characters from A to Z.

3.2 CHARACTER RECOGNITION

The recognition of optical characters is known to be one of the earliest applications of Artificial Neural Networks, which partially emulate human thinking in the domain of artificial intelligence.

The recognition of characters from scanned images of documents has been a problem that has received much attention in the fields of image processing, pattern recognition and artificial intelligence. Classical methods in pattern recognition do not as such suffice for the recognition of visual characters due to the following reasons:

- ✓ The 'same' characters differ in sizes, shapes and styles from person to person and even from time to time with the same person.
- ✓ Like any image, visual characters are subject to spoilage due to noise.
- ✓ There are no hard-and-fast rules that define the appearance of a visual character. Hence rules need to be heuristically deduced from samples.

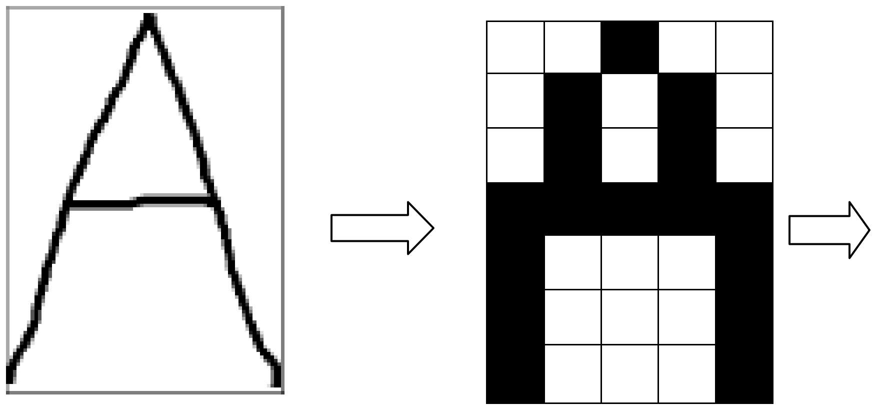
As such, the human system of vision is excellent in the sense of the following qualities:

- ✓ The human brain is adaptive to minor changes and errors in visual patterns. Thus we are able to read the handwritings of many people despite different styles of writing.
- ✓ The human vision system learns from experience: Hence we are able to grasp newer styles and scripts with amazingly high speed.

- ✓ The human vision system is immune to most variations of size, aspect ratio, color, location and orientation of visual characters.

3.3 IMAGE DIGITIZATION

When a document is put to visual recognition, it is expected to be consisting of printed (or handwritten) characters pertaining to one or more *scripts* or *fonts*. This document however, may contain information besides optical characters alone. For example, it may contain pictures and colors that do not provide any useful information in the instant sense of character recognition. In addition, characters which need to be *singly* analyzed may exist as *word clusters* or may be located at various points in the document. Such an image is usually processed for noise-reduction and separation of individual characters from the document. It is convenient for comprehension to assume that the submitted image is freed from noise and that individual characters have already been located (using for example, a suitable clustering algorithm). This situation is synonymous to the one in which a single noise-free character has been submitted to the system for recognition.



$$\begin{array}{ccccc} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{array}$$

(a) **(b)** **(c)**

Figure 3.1 Image digitization

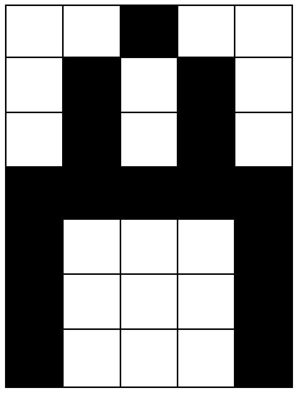
The process of digitization is important for the neural network used in the system. In this process, the input image is sampled into a binary window which forms the input to the recognition system. In the above figure, the alphabet *A* has been digitized into 5*X7=35* digital cells, each having a single color, either black or white. It becomes important for us to encode this information in a form meaningful to a computer. For this, we assign a value *+1* to each black pixel and *0* to each white pixel and create the binary image matrix *I* which is shown in the Fig. (6). So much of conversion is enough for neural networking which is described next. Digitization of an image into a binary matrix of specified dimensions makes the input image invariant of its actual dimensions. Hence an image of whatever size gets transformed into a binary matrix of fixed pre-determined dimensions. This establishes uniformity in the dimensions of the input and stored patterns as they move through the recognition system.

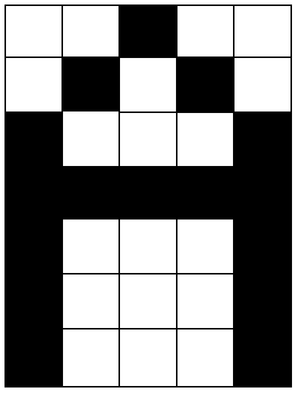
3.4 LEARNING MECHANISM

In the PNN method of classification, various characters are *taught* to the network in a supervised manner. A character is presented to the system and is assigned a particular *label*. Several variant patterns of the same character are taught to the network under the same label. Hence the network learns various possible variations of a single pattern and becomes adaptive in nature. During the *training process*, the input to the neural network is the input matrix as shown in fig 6 (c).

Whenever a character is to be taught to the network, an input pattern representing that character is submitted to the network. The network is then *instructed* to identify this pattern as, say, the k^{th} character in a *knowledge base* of characters. That means that the pattern is assigned a label *k*.

The following figure shows the digitization of three input patterns representing ***A*** that are presented to the system for it to learn.


(a)


(b)

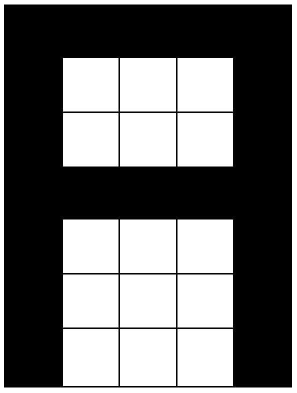

(c)

Figure 3.2 Character variants of the same class

Note that the patterns slightly differ from each other, just as handwriting differs from person to person (or time to time) and like printed characters differ from machine to machine.

3.5 PERFORMANCE ISSUES

The neural system has some direct advantages that become apparent at this stage:

1. The recognition is tolerant to minor errors and changes in patterns.
2. The knowledge base of the system can be modified by teaching it newer characters or teaching different variants of earlier characters.
3. The system is highly general and is invariant to size and aspect ratio.
4. The system can be made user specific: User-profiles of characters can be maintained, and the system can be made to recognize them as per the orientation of the user.

The dimensions of the input matrix need to be adjusted for performance. Greater the dimensions, higher the resolution and better the recognition. This however increases the time-complexity of the system which can be a sensitive issue with slower computers, but in dedicated PNN hardware, that’s not much of a concern. But memory is a real constraint.Typically, *32X32* matrices have been empirically found sufficient for the recognition of English handwritten characters. For intricate scripts, greater resolution of the matrices is required. As already illustrated, efficient supervised teaching is essential for the proper performance. Neural expert systems are therefore typically used where human-centered training is preferred against rigid and inflexible system-rules.

4 FIELD PROGRAMMABLE GATE ARRAYS

4.1 GENERAL DESCRIPTION

Field Programmable Gate Array or FPGA is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates (such as AND, OR, XOR, NOT) or more complex combinatorial functions such as decoders or simple math functions.

A hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer (hence the term "field-programmable") so that the FPGA can perform whatever logical function is needed.

4.2 FPGA ARCHITECTURE

Each FPGA vendor has its own FPGA architecture, but in general terms they are all a variation of that shown in Figure 8. The architecture consists of configurable logic blocks, configurable I/O blocks, and programmable interconnect. Also, there will be clock circuitry for driving the clock signals to each logic block, and additional logic resources such as ALUs, memory, and decoders may be available. The two basic types of programmable elements for an FPGA are Static RAM and anti-fuses.

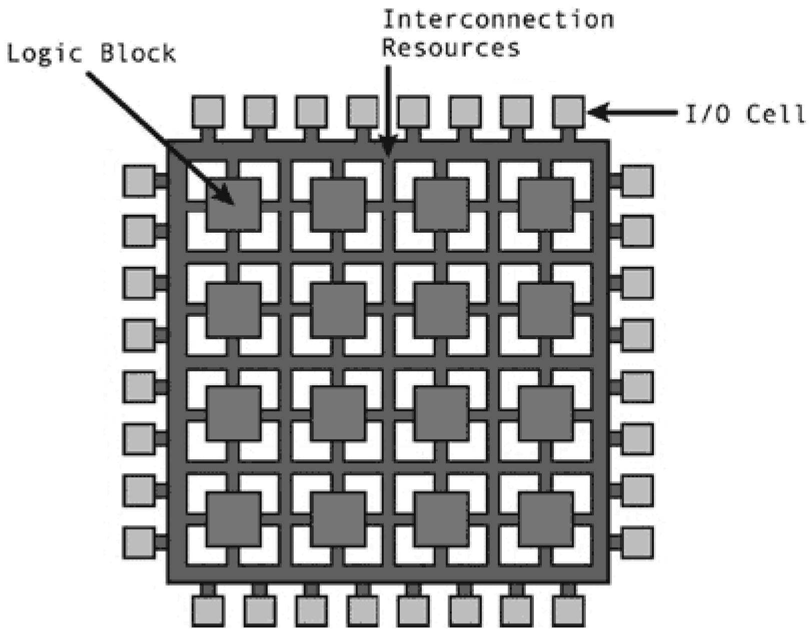


Figure 4.1 FPGA Architecture

4.2.1 CYCLONE II DEVICE ARCHITECTURE

Cyclone® II devices by Altera contain a two-dimensional row- and column-based architecture to implement custom logic. Column and row interconnects of varying speeds provide signal interconnects between logic array blocks (LABs), embedded memory blocks, and embedded multipliers.

The logic array consists of LABs, with 16 logic elements (LEs) in each LAB. An LE is a small unit of logic providing efficient implementation of user logic functions. LABs are grouped into rows and columns across the device. Cyclone II devices range in density from 4,608 to 68,416 LEs.

Cyclone II devices provide a global clock network and up to four phase-locked loops (PLLs). The global clock network consists of up to 16 global clock lines that drive throughout the entire device. The global clock network can provide clocks for all resources within the device, such as input/output elements (IOEs), LEs, embedded multipliers, and embedded memory blocks. The global clock lines can also be used for other high fan-out signals. Cyclone II PLLs provide general-purpose clocking with clock synthesis and phase shifting as well as external outputs for high-speed differential I/O support.

M4K memory blocks are true dual-port memory blocks with 4K bits of memory plus parity (4,608 bits). These blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 36-bits wide at up to 260 MHz. These blocks are arranged in columns across the device in between certain LABs. Cyclone II devices offer between 119 to 1,152 Kbits of embedded memory.

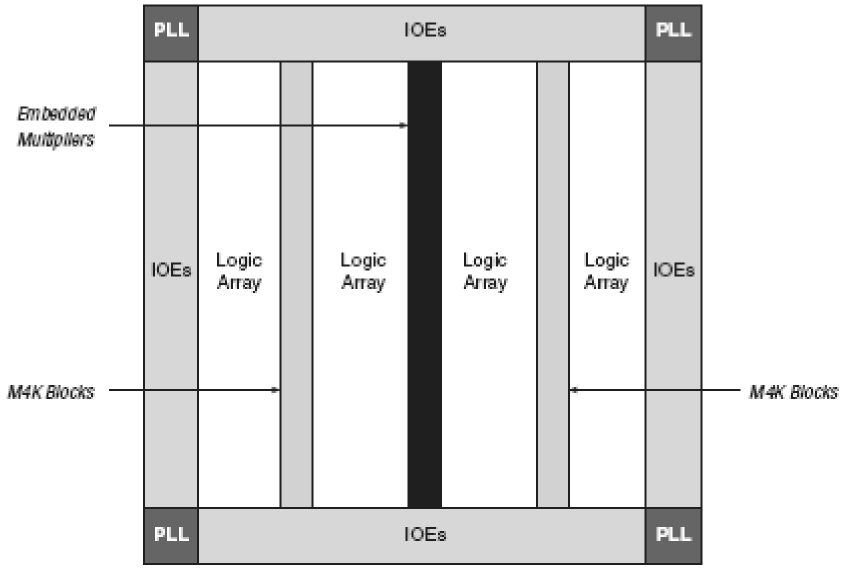


Figure 4.2 CYCLONE II EP2C20 Device Block Diagram

Each embedded multiplier block can implement up to either two 9 × 9-bit multipliers, or one 18 × 18-bit multiplier with up to 250-MHz performance. Embedded multipliers are arranged in columns across the device. Each Cyclone II device I/O pin is fed by an IOE located at the ends of LAB rows and columns around the periphery of the device. I/O pins support various single-ended and differential I/O standards, such as the 66- and 33-MHz, 64- and 32-bit PCI standard, PCI-X, and the LVDS I/O standard at a maximum data rate of 805 megabits per second (Mbps) for inputs and 640 Mbps for outputs.

Each IOE contains a bidirectional I/O buffer and three registers for registering input, output, and output-enable signals. Dual-purpose DQS, DQ, and DM pins along with delay chains (used to phase-align double data rate (DDR) signals) provide interface support for external memory devices such as DDR, DDR2, and single data rate (SDR) SDRAM, and QDRII SRAM devices at up to 167 MHz.

4.3 FPGA DEVELOPMENT BOARD

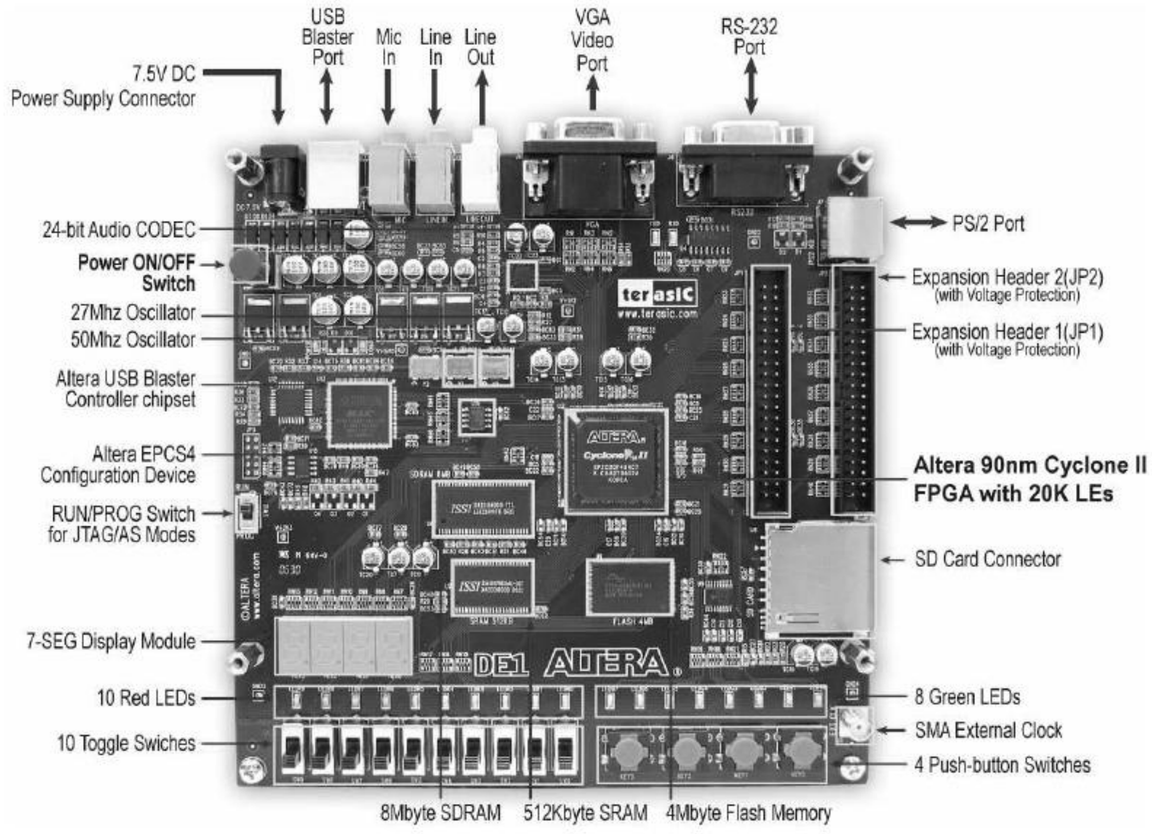


Figure 4.3 DE1 BOARD

The heart of the development board is the ALTERA Cyclone II FPGA.It contains various interfacing circuitry for programming the FPGA ,as well as for applying inputs and testing the output.

The following hardware is provided on the DE1 board. It is powered by either a 7.5V DC adapter or a USB cable

- **Cyclone II 2C20 FPGA**
- **Serial Configuration device and USB Blaster circuit**

- **SRAM**
- **SDRAM**
- **Flash memory**
- **SD card socket**
- **Pushbutton switches**
- **Toggle switches**
- **Clock inputs**
- **Audio CODEC**
- **VGA output**
- **Serial ports**
- **Two 40-pin expansion headers**

4.4 FPGA DESIGN FLOW

4.4.1 DESIGN ENTRY

User can enter the design by writing the descriptions of any digital system using HDL (Hardware Description Language), which can be either VHDL or Verilog HDL.

4.4.2 SYNTHESIS

Synthesis is the process, which converts HDL CODE into Gate level circuit in the form of NETLIST. This process is Target Technology dependent hence user must select proper Device, Family, Part Number and Speed Grade. Also user may select Synthesis Settings like Clock Frequency, Optimization for Speed or Area.

4.4.3 SIMULATION

Simulation, user can verify the functionality of his design by applying various input signal combination and observing the output results. The simulation is performed on gate level Netlist.

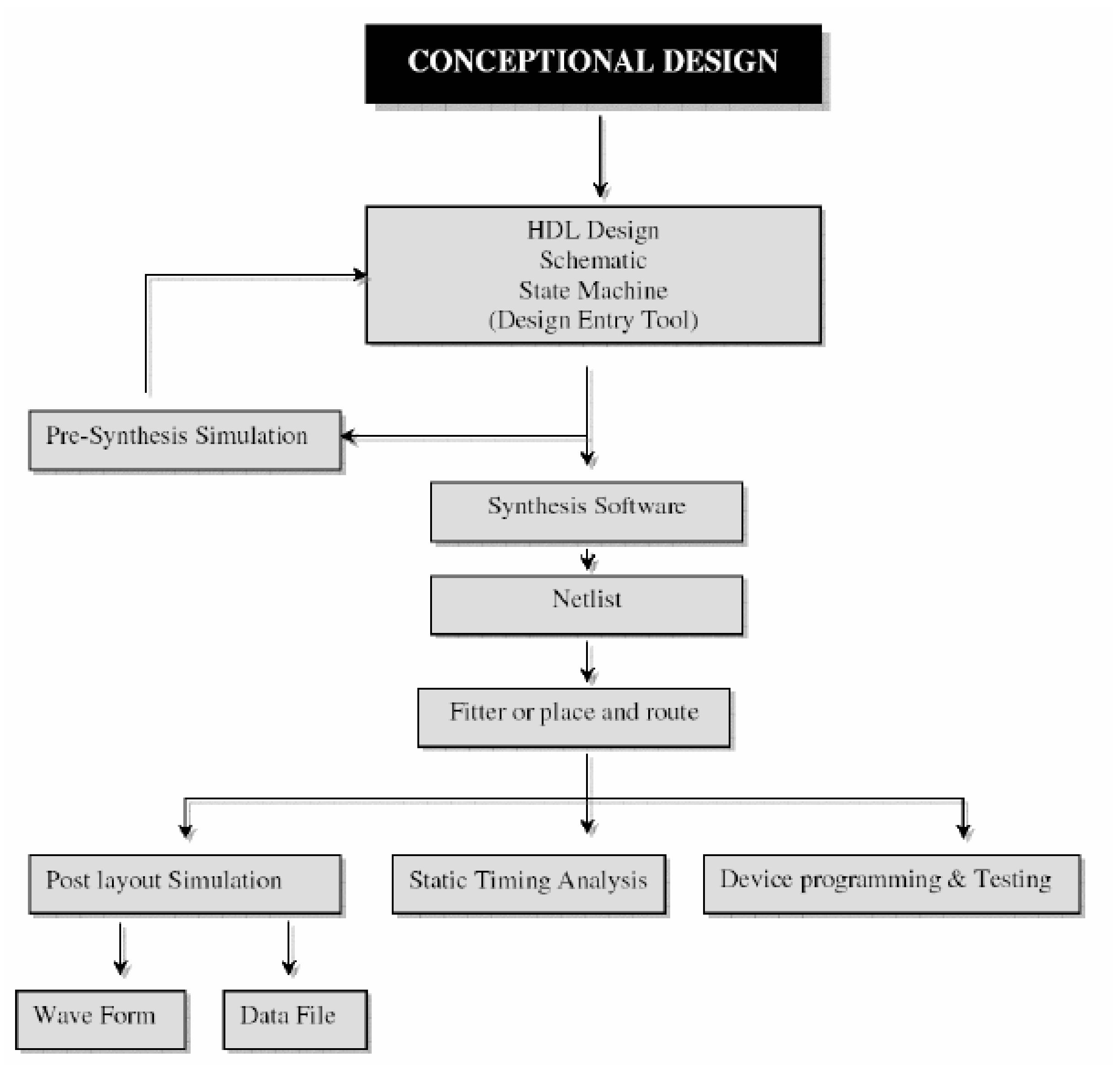


Fig 4.4 : FPGA Design Flow

4.4.4 IMPLEMENTATION

Implementation is the process in which the design is passed through various stages by TRANSLATE, MAPPING, TIME ANALYSIS and BIT STREAM. For locking input and output signal to particular pins of the device, user must use assignment editor before implementation and guide the same file to implementation tool through the option set control files. Output of implementation is .bit file which has to be converted into PROM file i.e .POF file.

4.4.5 PROGRAMMING

This is the process by which user can physically download the architecture from computer to target device using programming cable. Use an FPGA Development Platform to download the Data to the FPGA development board.

5 SOFTWARE TOOLS

EDA tools are an integral part of FPGA design process. These tools include different steps in FPGA design. Most of them incorporate design entry, fitting or placing and routing, assembling, functional and timing simulation, chip editor, programming tools etc. The software used in this project is Altera Quartus II 6.0 .

5.1 QUARTUS II 6.0 SOFTWARE

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. In addition, the Quartus II software allows you to use the Quartus II graphical user interface and command-line interface for each phase of the design flow The Quartus II software includes solutions for all phases of FPGA and CPLD design. FPGA Design Flow in Quartus is as follows,

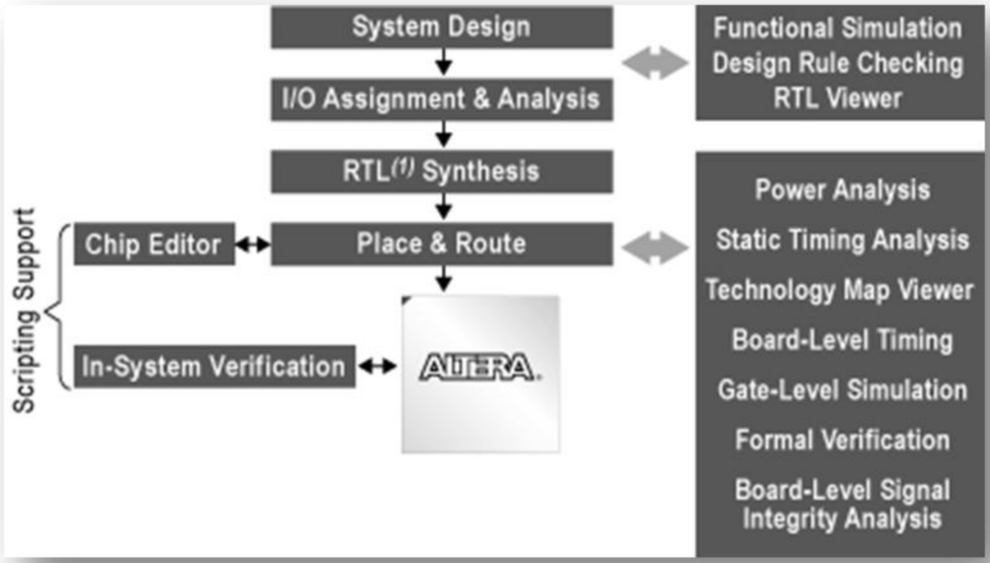


Figure 5.1 Design Flow

Quartus II software enables the highest levels of productivity and the fastest path to design completion for both high-density and low-cost FPGA design.

The Latest features of Quartus are as follows,

- **Different Design entry schemes** : Verilog HDL,VHDL,Block diagram Design etc
- **Full Compilation at one click**
- **Vector Waveform Simulation**
- **Chip Editor** : View of System level implementation of the design
- **Netlist Viewers** :RTL viewer, Technology Map viewer, State Machine viewer
- **Mega Wizard Plug-in Manager** : LPM generation (Library Of Parameterized Modules)
- **Support for a Wide range of devices**
- **TimeQuest timing analyzer** is a new, next-generation ASIC-strength timing analyzer supporting the industry-standard Synopsys Design Constraints (SDC)-based timing analysis methodology
- **PowerPlay power** analysis and optimization technology provides automated power optimization capabilities and helps you effectively manage power from design concept through implementation.
- **Incremental compilation** supports the bottom-up design flow that allows design blocks to be created and optimized independently. System architects can incrementally integrate optimized design blocks while the performance of the design blocks is preserved throughout the integration process.
- **SOPC Builder** eliminates mundane and error-prone system integration tasks and allows you to build systems in minutes.
- **Push-button physical synthesis technology and the automated Design Space Explorer** simplify design optimization.
- **Extensive cross-probing** support between tools helps identify and correct design issues.
- **The pin planner feature (PDF)** enables easy I/O pin assignment planning, assignment, and validation.
- **Complete command-line and tool command language (Tcl)** scripting interfaces give you advanced scripting capabilities.

6 VERILOG

6.1 BACKGROUND

The Verilog Hardware Description Language (HDL) is a language for describing the behaviour and structure of electronic circuits, and is an IEEE standard (IEEE Std. 1364-2001).

Verilog is used to simulate the functionality of digital electronic circuits at levels of abstraction ranging from stochastic and pure behaviour down to gate and switch level, and is also used to synthesize (i.e. automatically generate) gate level descriptions from more abstract (Register Transfer Level) descriptions. Verilog is commonly used to support the high level design (or language based design) process, in which an electronic design is verified by means of thorough simulation at a high level of abstraction before proceeding to detailed design using automatic synthesis tools. Verilog is also widely used for gate level verification of ICs, including simulation, fault simulation and timing verification.

The Verilog HDL was originally developed together with the Verilog-XL simulator by Gateway Design Automation, and introduced in 1984. In 1989 Cadence Design Systems acquired Gateway, and with it the rights to the Verilog language and the Verilog-XL simulator. In 1990 Cadence placed the Verilog language (but not Verilog-XL) into the public domain. A non profit making organization, Open Verilog International (OVI) was formed with the task of taking the language through the IEEE standardization procedure, and Verilog became an IEEE standard in 1995.

Verilog is very C-like and liked by electrical and computer engineers as most learn the C language in college.

6.2 IMPORTANCE OF VERILOG HDL

The Verilog language provides the digital designer with a means of describing a digital system at a wide range of levels of abstraction, and, at the same time, provides access to computer-aided design tools to aid in the design process at these levels.

Verilog allows hardware designers to express their design with behavioral constructs, deterring the details of implementation to a later stage of design in the design. An abstract representation helps the designer explore architectural alternatives through simulations and to detect design bottlenecks before detailed design begins.

Our goal in the course is not to create VLSI chips but to use Verilog to precisely describe the *functionality* of *any* digital system, for example, a computer. However, a VLSI chip designed by way of Verilog’s behavioral constructs will be rather slow and be wasteful of chip area. The lower levels in Verilog allow engineers to optimize the logical circuits and VLSI layouts to maximize speed and minimize area of the VLSI chip.

6.3 VERILOG LANGUAGE

A hierarchical portion of a hardware design is described in Verilog by a *Module*. The *Module* defines both the interface to the block of hardware (i.e. the inputs and outputs) and its internal structure or behaviour.

A number of primitives, or *Gates*, are built into the Verilog language. They represent basic logic gates (e.g. and, or). In addition *User Defined Primitives* (UDPs) may be defined.

The structure of an electronic circuit is described by making *Instances* of *Modules* and Primitives (*UDPs* and *Gates*) within a higher level *Module*, and connecting the *Instances* together using *Nets*. A *Net* represents an electrical connection, a wire or a bus. A list of *Port* connections is used to connect *Nets* to the *Ports* of a *Module* or Primitive *Instance*, where a *Port* represents a pin. *Registers* may also be connected to the input *Ports* (only) of an *Instance.*

Nets (and *Registers*) have values formed from the logic values 0, 1, X (unknown or uninitialized) and Z (high impedance or floating). In addition to logic values, *Nets* also have a *Strength* value. *Strengths* are used extensively in switch level models, and to resolve situations where a net has more than one driver.

The behaviour of an electronic circuit is described using *Initial* and *Always* constructs and *Continuous Assignments.* Along with *UDP*s and *Gates* these represent the leaves in the hierarchy tree of the design. Each *Initial, Always, Continuous Assignment,* *UDP* and *Gate Instance* executes concurrently with respect to all others, but the *Statements* inside an *Initial or Always* are in many ways similar to the statements in a software programming language. They are executed at times dictated by *Timing Controls*, such as delays, and (simulation) event controls. *Statements* execute in sequence in a *Begin-End* block, or in parallel in a *Fork-Join* block. A *Continuous Assignment* modifies the values of *Nets*. An *Initial* or *Always* modifies the values of *Registers* An *Initial* or *Always* can be decomposed into named *Tasks* and *Functions*, which can be given arguments. There are also a number of built in *System Tasks and Functions*. The *Programming Language Interface* (*PLI*) is an integral part of the Verilog language, and provides a means of calling functions written in C in the same way as *System Tasks and Functions*.

7 PROJECT DESCRIPTION

The objective of the project is to develop a hardware system which is capable of recognizing all characters from A to Z. The inputs to the system are fed from MATLAB software in PC through the parallel port interface.

The system can be divided into two parts--

- the software part in PC
- the hardware part in FPGA.

The software part is written in MATLAB 7.1.It has a GUI interface to feed the input vectors which are characters in 5x7 matrix format as explained before in character recognition description, to the neural processor to train the PNN.There is provision for transmitting the exponential Look Up Table(LUT).Initially the exponential LUT and the training vectors are loaded into the FPGA via parallel port of PC through the Expansion header I/O in the DE1 board. Then test inputs are applied to the PNN in the same way.

The hardware part consists of the DE1 board, which has the FPGA,Cyclone II 2C20 from Altera. The development board has all the interfacing circuits, like the Expansion header I/O,LEDs,7-segment displays etc. which are connected to the FPGA.Verilog HDL code for PNN written in Quartus II 6.0 software is compiled and programmed into the FPGA through USB-Blaster interface. The classified output will be displayed in the 7 segment displays in the DE1 board.

8 SYSTEM BLOCK DIAGRAM

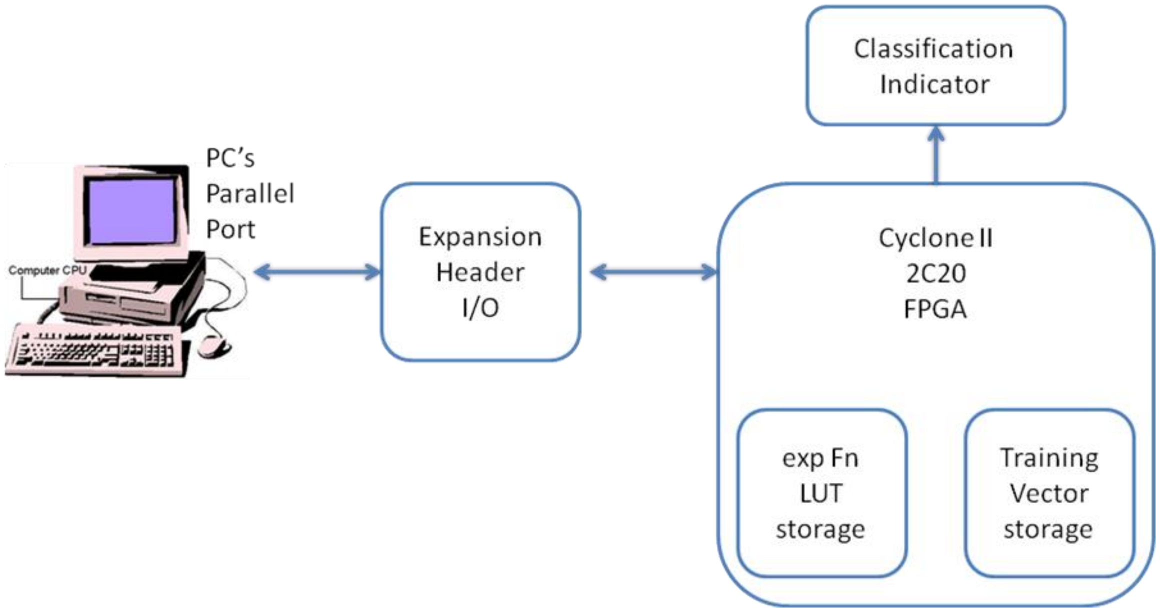


Figure 8.1 System block diagram

A block diagram of the overall system is shown above. The MATLAB program communicates with the FPGA through PC's parallel port interface. The data pins are directly connected to the Expansion header I/O in the DE1 board. The pins of this expansion header are assigned to FPGA's I/O cells. The Cyclone II 2C20 FPGA is the brain of the whole system which forms the neural processor. There is a provision for storing the training vectors as well as the exponential LUT in embedded memory blocks in the FPGA. These values are fed and retrieved whenever needed. The PNN in FPGA, after processing produces the classified result, which is displayed on the 7-segment displays on the DE1 board.

9 HARDWARE MODELLING

FPGA logic is a class of digital integrated circuitry whose internal logic structure can be configured in parallel computing units and be repeatedly altered or programmed after manufacture. The property of hardware reprogrammability is very attractive for ANN implementation since it allows for maximum network flexibility. Also, the ability to program parallel computing units makes FPGA logic very suitable for implementing the parallel structure of the PNN.

9.1 PROPOSED DESIGN

A design concept is introduced for the implementation of the Probabilistic Neural Network classifier using standard binary Field Programmable Gate Array logic. The hardware implementation of the PNN classifier accounts for making each layer of PNN Architecture in silicon. This neural network when embedded in hardware may be called a Neural Processor. This System On a Chip (SoC) design constitutes the following layers.

9.2 INPUT LAYER

The input vectors are fed to the network through this layer. Each character is represented by a 35-bit vector (5x7), which is fed to the FPGA I/O cells from the Expansion header I/O in DE1 board. This expansion header is directly connected to the parallel port of the PC.These vectors are stored in the registers in FPGA for further processing.

The input layer to radial basis layer weights are formed by the training vectors itself. Here in our application of character recognition, these vectors are all binary. A distance measuring method is required to compute the deviation of the input vector from the trained vector. The Euclidean distance, defined by equation **(1),** is the most commonly used distance measure.

$$d = \sqrt{(x - x_{ij})^T (x - x_{ij})} \quad \text{.........................} \ (1)$$

To simplify the required computations an approximation to the Euclidean distance can be used. Some suitable ones include the square distance, the hamming and city block measures. The square distance is defined by equation (2).

$$d = \text{maximum}\left|x - x_{ij}\right| \text{.............................. (2)}$$

The general hamming distance between vectors x and x_{ij} of dimension p is defined by equation (3), the sum of the absolute differences of the vector elements x and x_{ij}.

$$d = \sum_{k=1}^{p} \left|x_k - x_{ijk}\right| \text{ .... (3)}$$

For real valued vectors the hamming distance is equivalent to the city block distance. The hamming distance because of its simplicity is commonly used to compare binary vectors because it can be computed using an exclusive OR function as defined by equation,

$$d = \sum_{k=1}^{p} x_k \text{ XOR } x_{ijk} \text{.. (4)}$$

So the city block distance can be easily performed in hardware by XORing the corresponding bits and applying summation to all the 35 dimensions. Now the summed output for each of the training patterns is performed by a single bit 35-input adder. A lower value indicates closer match and higher value indicates bad match.

9.3 RADIAL BASIS LAYER

This layer performs the transfer function on the distance measure for each hidden node to form a vector whose elements indicate how close the input is to a training input. This radial basis function (RBF) produces response in such a way that an input vector close to a training vector will be represented by a number close to one in the output vector. The spherical Gaussian radial basis function is a Parzen pdf estimator which is given by the equation as explained on the PNN classifier section.

Due to difficulties in implementing exponential function in FPGA, we went for a LUT (Look Up Table) implementation for this. This LUT formation is described in the software implementation section. The 36 word 14-bit length LUT is stored in the RAM embedded in FPGA which is fed from the MATLAB program in PC.

The input layer’s output produces a value of the range 0 to 35 according to the input vector match with the trained vector. Now the output of the input layer acts as the address for the LUT. The value corresponding to this address is then retrieved from the RAM, which will be a 14-bit value. Thus we have a 14-bit value corresponding to each hidden layer class vectors in the network. Suppose if there are three training vectors for each class or alphabet, then the output of the kernel layer will have 3x26=78 outputs. This completes the radial basis transfer function.

9.4 SUMMATION LAYER

The output of the kernel layer is fed to the summation layer which performs the addition of the Gaussian functions in a particular class. Here the 14-bit outputs corresponding to each hidden neuron is summed up in such a way that, they belong to the same class.

For example, each variant of the character ‘A’ trained initially will have a 14-bit value output after passing it through radial basis transfer function. Let there be three variants for character ‘A’. Now what the summation layer does is that it performs addition of these three 14-bit vectors, thus forming a 16-bit sum. This is implemented using a 14-bit three input adder.

Thus there is a pattern neuron for each category of the target variable. In the application of recognizing 26 characters or alphabets, we have a total of 26 outputs of 16-bits each from the summation layer pointing to each class.

9.5 COMPETITIVE LAYER

This is the final layer of the PNN which takes the decision for the classification. Here in accordance with the Bayes’ decision rule, the outputs of each class are compared. It performs the magnitude comparator operation, by way of which the class corresponding to the maximum valued output is declared the winner.

For this we have a 16-bit 26 input comparator, which performs magnitude comparison over the 26 classes. Whichever class inputs the maximum value, will be the classified output. Thus the fed input vector is classified into a particular class in accordance with the pre-loaded training vectors for each class. The label corresponding to each class, which in this case are alphabets from 'A' to 'Z', are displayed on the 7-segment displays.

9.6 DYNAMICALLY ADDING NEW TRAINING VECTORS

The system also provides the feature of adding new training vectors real-time. This is accomplished by allocating a memory bank corresponding to each character's training class. The maximum number of vectors that can represent a class has been fixed to be 3, due to memory limitation. This can be very easily extended to include as many training vectors as required. So there are a total of 26 memory banks with each bank having a maximum of 3 training vectors.

9.7 ROADBLOCKS

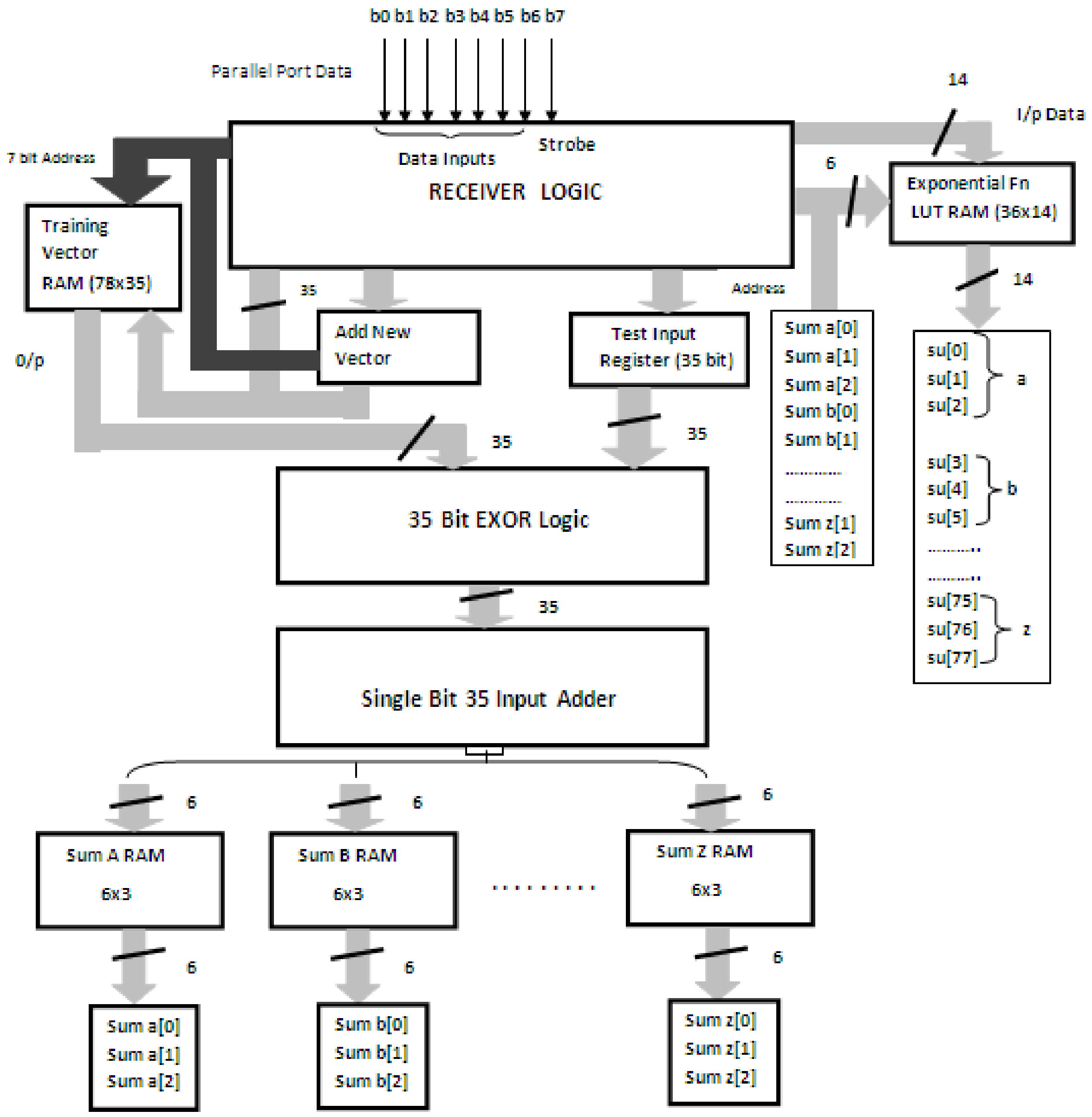
Throughout the hardware development of the project, there were many problems at several occasions. However, with much study on the subject, they were solved.

- As some of the HDL codes run in parallel, logic had to be designed so, which is totally a new experience for persons who have background knowledge in software and microcontroller coding.

- Code optimization is another problem faced. After completing the logic and primary HDL entry, the code had to be revised in order to optimize resources available in Cyclone II FPGA. All vectors, training vectors and exponential LUTs were stored in simple registers which will be synthesized in the LEs itself. Thus we were out of LEs very fast .About 28,000 LEs were required for this, but our FPGA device supported only 18.752 LEs .Storing all the memory elements in the embedded M4K memory blocks in FPGA solved this. Then it was found that the used LE count had come down to about 16,000.

- Verilog codes are basically two types, codes that are synthesizable and codes that are used for simulation and non-synthesizable. Behavioral statement ‘initial’ had been used in the code for initializing some variables. Later it has found that ‘initial’ is not synthesizable.

9.8 HARDWARE DESCRIPTION BLOCK DIAGRAM



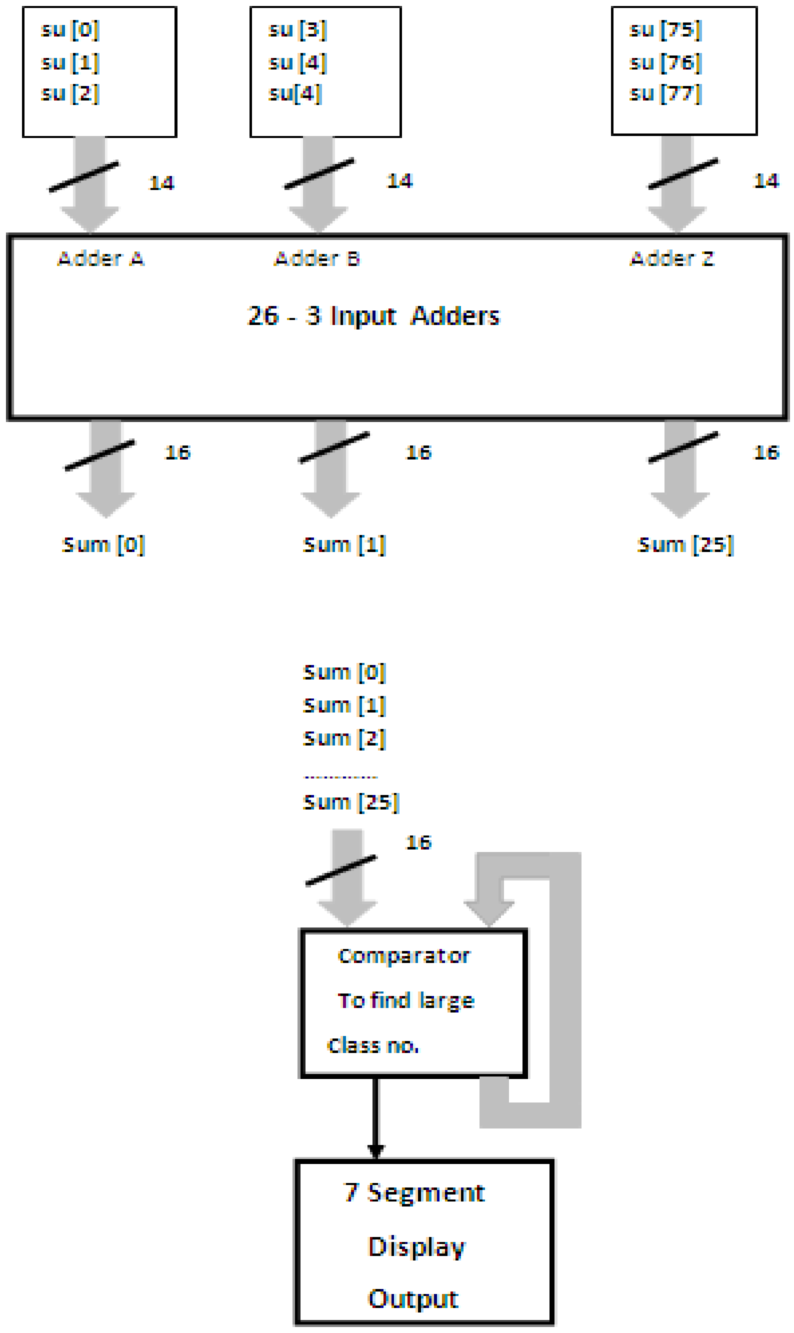


Figure 9.1 Hardware Description Block Diagram

10 FPGA COMPUTATIONAL FLOW

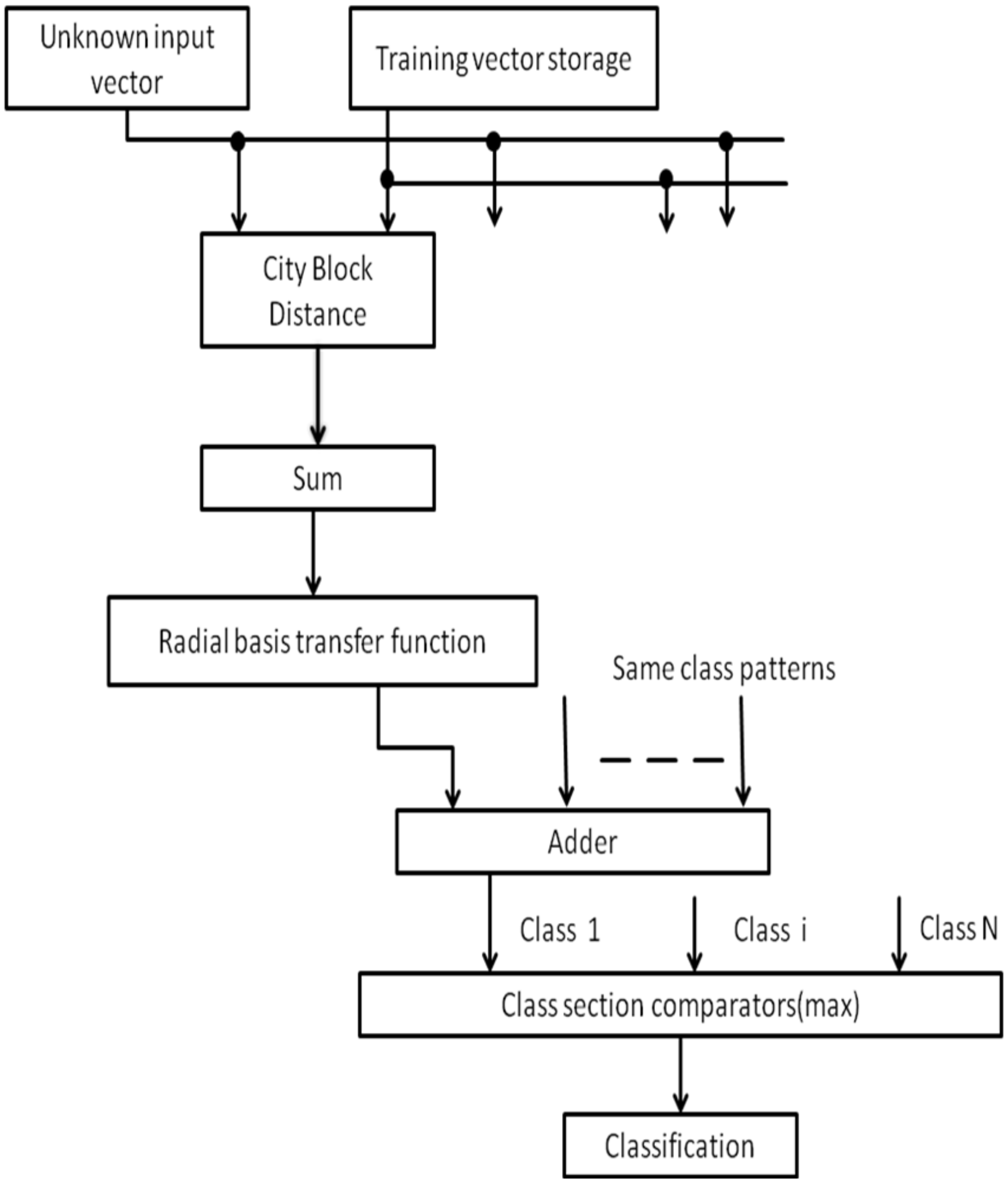


Figure 10.1 FPGA Computational Flow

11 SOFTWARE IMPLEMENTATION

11.1 MATLAB 7.1

This language of technical computing provides with a lot of inbuilt modules for easy simulation and testing.

The Probabilistic Neural Network was simulated in MATLAB 7.1.The Neural Network Toolbox was utilized for creating a Probabilistic neural network and then simulating it with test inputs.

After implementing PNN in FPGA, input vectors are fed to the hardware system through the PC interface, because it provides more flexibility. The input vectors to the FPGA are sent from the MATLAB program into the FPGA through parallel port interface of the PC.

11.2 GUI INTERFACE

The intuitive GUI interface provides the user with the ultimate user friendliness. The ‘GUIDE’ tool in MATLAB can be used to initialize this. Shown below is a screenshot of our MATLAB GUI interface.

Onto to the upper side of the left panel of the GUI, we have the character entry through a toggle button array interface, which represents the 5x7 vector corresponding to each character. In the figure 21, character ‘S’ is entered. This entry provides the user with a very simple way to enter the characters. The state of the toggle buttons can be retrieved real-time for conversion into the input vector format.

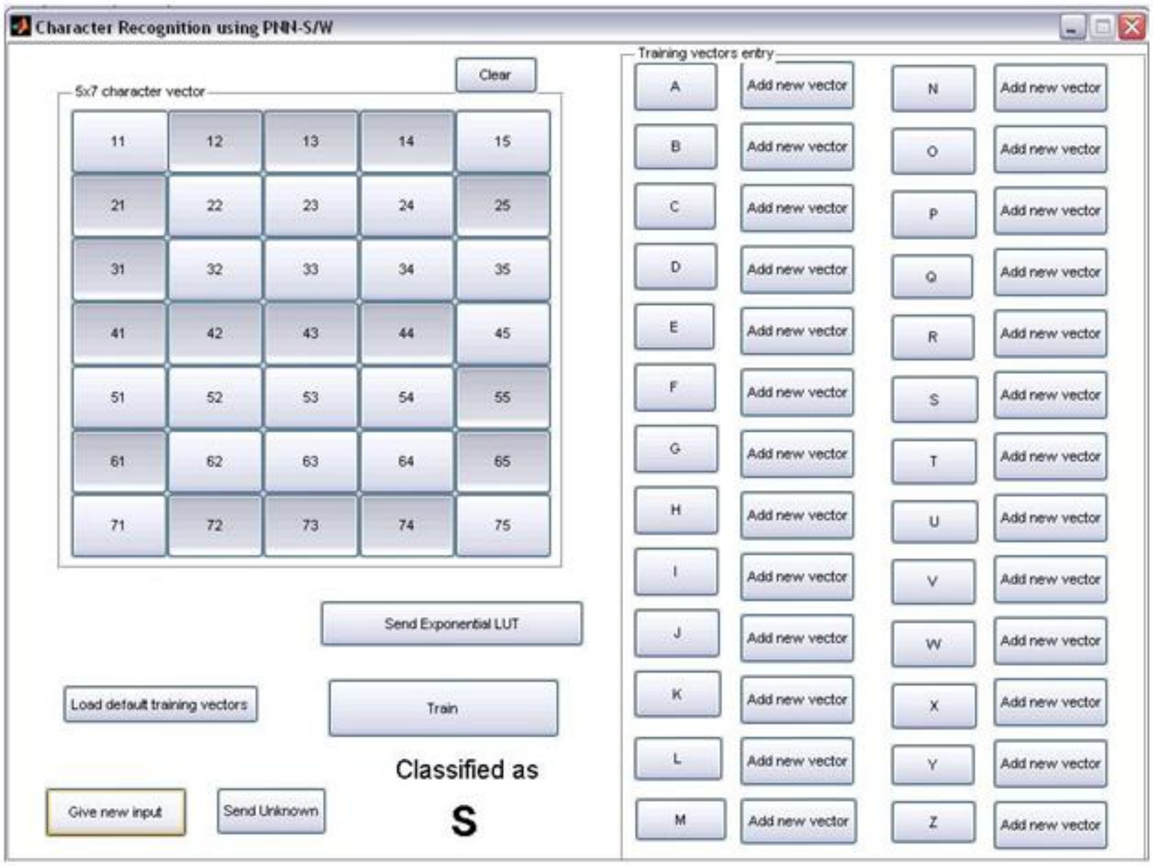


Figure 11.1 Screenshot of the MATLAB GUI Interface

On the lower side of the left panel and right panel, we have various controls whose functionality are explained below:

- ❖ .... **S**

  end Exponential LUT:

  This is the control which activates the function for generating the Look Up Table for the exponential Gaussian function and transmitting it to the FPGA.

- ❖ **Clear:**

  This is a very important control. The user has to clear the 5x7 matrix after each entry. Otherwise it may result in wrong input. This control clears all button data types.

- ❖ **Load default training vectors:**

  This is the simplified function where some default values are stored for each character. This is assigned as the training vectors from A to Z.Also the user can load his own training vectors by entering them from the right panel of the GUI, where there are buttons for each character.

- ❖ **Train:**

  This control performs the process of transferring the training vectors to the FPGA through the parallel port interface. There will be a vector corresponding to each class. So here we have a 35-bit vector for each character from A to Z.

- ❖ **Give new input:**

  On completion of training, we can apply test inputs to the PNN for classification. This button takes the input from the user from the 5x7 matrix interface.

- ❖ **Send Unknown:**

  The test input is fed to the FPGA through the parallel port interface. The neural processor performs computations on test input with training vectors. The output is displayed on the 7-segment displays on the DE1 board.

- ❖ **Add new vector:**

  This feature provides the facility to dynamically add training vectors to a particular class, if a faulty classification results.

11.3 PARALLEL PORT INTERFACE

The parallel port interface was selected to transfer data from MATLAB program in PC to the FPGA. The standard parallel port is capable of sending 50 to 100 kilobytes of data per second.

Only the 8 data pins of the parallel port were used to transfer data to the FPGA. Since training vectors as well as input vectors were 35-bit (5x7) sized, a single packet of data was fixed at 7-bits.The D0 to D6 pins were used for transmitting these packets. The D7 pin was used as a STROBE signal, to indicate the FPGA that the data is available at its ports for retrieval. In this way 5 packets are used to transmit the whole 35-bit vector.

The ‘DIGITALIO’ function was used to create an object of the parallel port. Later this object is referred to transmit data. Onto this object, we add digital lines by using the ‘ADDLINE’ function. So the 7 data lines are defined as data-out lines and the remaining one data line is defined as strobe line. After this, 7-bit data is appended to the output buffer by using the 'PUTVALUE' function. Now the strobe is made high and then low to indicate the FPGA that data is available at its ports. In this way, all packets are transmitted to the FPGA.

Pin No (D-Type 25)	**Pin No (Centronics)**	**SPP Signal**	**Direction In/out**	**Register**	**Hardware Inverted**
1	1	nStrobe	In/Out	Control	Yes
2	2	Data 0	Out	Data	
3	3	Data 1	Out	Data	
4	4	Data 2	Out	Data	
5	5	Data 3	Out	Data	
6	6	Data 4	Out	Data	
7	7	Data 5	Out	Data	
8	8	Data 6	Out	Data	
9	9	Data 7	Out	Data	
10	10	nAck	In	Status	
11	11	Busy	In	Status	Yes
12	12	Paper-Out PaperEnd	In	Status	
13	13	Select	In	Status	
14	14	nAuto-Linefeed	In/Out	Control	Yes
15	32	nError / nFault	In	Status	
16	31	nInitialize	In/Out	Control	
17	36	nSelect-Printer nSelect-In	In/Out	Control	Yes
18 - 25	19-30	Ground	Gnd		

Table 11.1 Pin Assignments of the D-Type 25 pin Parallel Port Connector.

The above table clearly depicts the parallel port pin configuration as well as their functions. In this pins 2 to 9 which act as Data 0 to Data 7 pins are used in this system to transmit data from the PC to the FPGA.

11.4 EXPONENTIAL LUT TRANSMISSION

The most demanding implementation issue is the efficient computation of the exponential function associated with the spherical Gaussian basis functions. Due to the difficulty in computing exponential function in hardware, we generate an LUT (Look Up Table) from MATLAB to feed into the FPGA.

First the exponential value is computed for the range of values encountered. The exponential LUT will be formulated in MATLAB software and then transferred to the FPGA registers. The neural FPGA processor will be now using this LUT to implement the transfer function.

The exponential function is given by,

$\exp(-n/\sigma^2)=0.\times\times\times\times$ to 1 (4-digit fractional precision)

This value is converted into binary by the “**quantizer**” and “**num2bin**” functions in MATLAB 7.1.This will be a 14-bit binary value. The ‘n’ value here can range from 0 to 35,since a ‘0’ represents perfect match and a ‘35’ represents worst match ,after the city block distance calculation. So we have an LUT with 36 rows, each row having 14-bits.The LUT value corresponding to n=0, which is the highest with all bits high, is fixed at the FPGA itself and is not transmitted from MATLAB. This LUT data is transmitted via parallel port to the FPGA before the training process itself.

11.5 LOADING TRAINING VECTORS

The training vectors can be entered by the user in two ways. One method is by clicking on “Load default training vectors”, which automatically loads the pre-defined vectors corresponding to each character for training the PNN.The second method is by entering the character one by one in the 5x7 matrix interface and clicking on corresponding character button.

The training vector consists of 35 bits for each character, thus constituting 26 such packets of data for representing all alphabets. After loading these vectors, they are transferred to the FPGA on hitting the “Train” button.

11.6 LOADING TEST INPUT

After training the PNN, with training vectors, it is time to test the neural network with new inputs which may be noisy. After passing through the PNN computational flow in FPGA, proper classification decisions are taken. The classified output is then displayed.

11.7 DYNAMICALLY ADDING NEW TRAINING VECTORS

After initial training and testing, we may sometimes encounter a case where the PNN performs misclassification. Now we can add this misclassified vector to our wanted class by clicking on the "Add new vector" corresponding to that character. This provides us with a very flexible classification system, which is efficient, at the same time, faster and easier to implement and train.

11.2 ROADBLOCKS

Throughout the MATLAB software development of the project, there were many problems at several occasions. However, with much study on the subject, they were solved.

- The states of the toggle buttons were not changing properly when characters are entered concurrently. It has been found that MATLAB GUI toggle buttons preserve their old values. So a 'CLEAR' button was included in the software, which has to be clicked before any character is entered. This solved the problem.

12 CONCLUSION

The implementation of neural networks in hardware FPGA is an innovative idea, apart from the existing implementations most of which have been software. We have been able to complete the hardware implementation of Probabilistic Neural Network with utmost satisfaction. The classification results have been good enough, providing with it, high speed results.

As a primary phase, we have gone for a simple application area of character recognition. But the scope of PNN in classification is immense. It has highly diverse applications, delivering its efficiency and accuracy to them.

As a future venture, the project can be extended for use in handwritten character recognition. Another application having high importance is the analysis of medical images. Let the Almighty give us the time and energy to achieve our dreams in this area.

APPENDIX 1

CYCLONE II FPGA ARCHITECTURE

1.1 LOGIC ELEMENTS

The smallest unit of logic in the Cyclone II architecture, the LE, is compact and 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 carry chain connection

■ A register chain connection

■ The ability to drive all types of interconnects: local, row, column, register chain, and direct link interconnects

■ Support for register packing

■ Support for register feedback

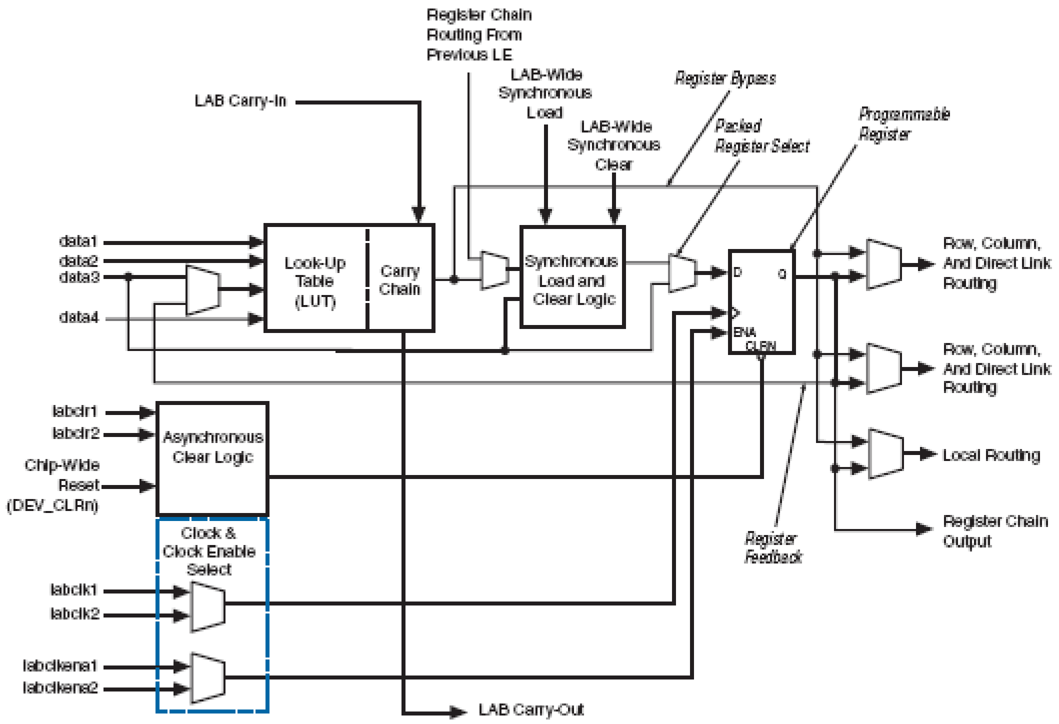


Figure A1.1 A CYCLONE II LE

Each LE’s programmable register can be configured for D, T, JK, or SR operation. Each register has data, clock, clock enable, and clear inputs. Signals that use the global clock network, general-purpose I/O pins, or any internal logic can drive the register’s clock and clear control signals. Either general-purpose I/O pins or internal logic can drive the clock enable. For combinational functions, the LUT output bypasses the register and drives directly to the LE outputs.

Each LE has three outputs that drive the local, row, and column routing resources. The LUT or register output can drive these three outputs independently. Two LE outputs drive column or row and direct link routing connections and one drives local interconnect resources, allowing the LUT to drive one output while the register drives another output. This feature, register packing, improves device utilization because the device can use the register and the LUT for unrelated functions. When using register packing, the LAB-wide synchronous load control signal is not available.

1.2 LOGIC ARRAY BLOCKS

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. The Quartus II Compiler places associated logic within an LAB or adjacent LABs, allowing the use of local, and register chain connections for performance and area efficiency.

Another special packing mode allows the register output to feed back into the LUT of the same LE so that the register is packed with its own fan-out LUT, providing another mechanism for improved fitting. The LE can also drive out registered and unregistered versions of the LUT output.

In addition to the three general routing outputs, the LEs within an LAB have register chain outputs. Register chain outputs allow registers within the same LAB to cascade together. The register chain output allows an LAB to use LUTs for a single combinational function and the registers to be used for an unrelated shift register implementation. These resources speed up connections between LABs while saving local interconnect resources.

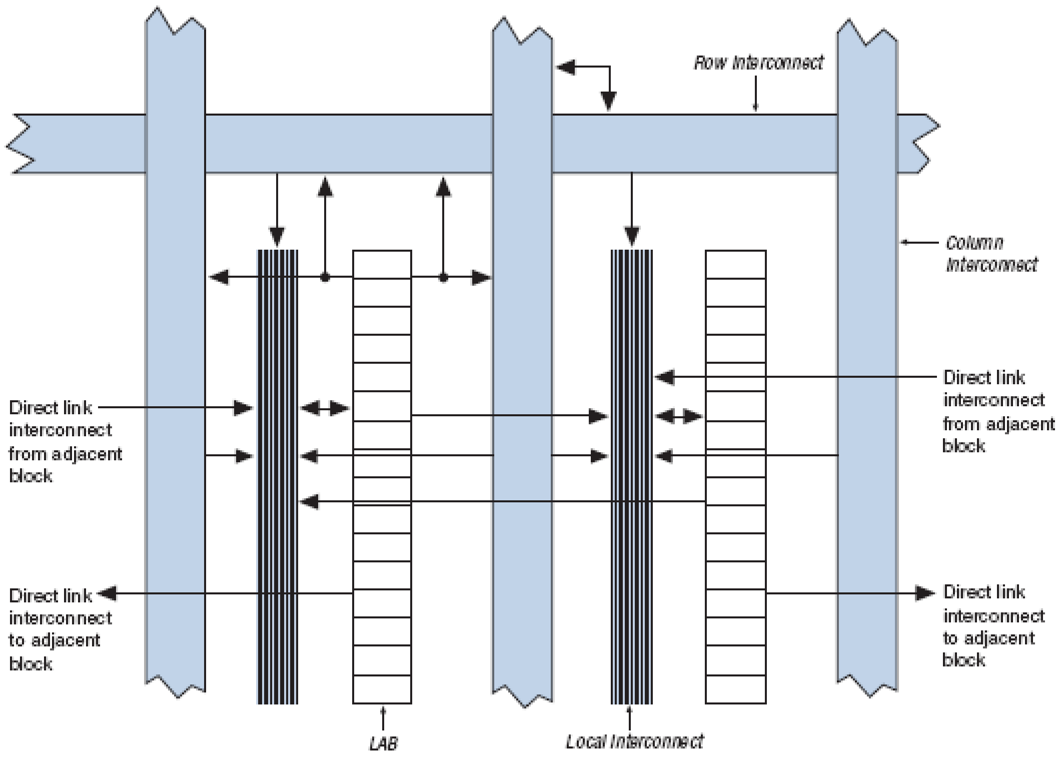

Row Interconnect
Column Interconnect
Direct link interconnect from adjacent block
Direct link interconnect from adjacent block
Direct link interconnect to adjacent block
Direct link interconnect to adjacent block
LAB
Local Interconnect

Figure A1.2 CYCLONE II LAB Structure

1.3 I/O STRUCTURE AND FEATURES

IOEs support many features, including:

- Differential and single-ended I/O standards
- 3.3-V, 64- and 32-bit, 66- and 33-MHz PCI compliance
- Joint Test Action Group (JTAG) boundary-scan test (BST) support
- Output drive strength control

- ■ Weak pull-up resistors during configuration
- ■ Tri-state buffers
- ■ Bus-hold circuitry
- ■ Programmable pull-up resistors in user mode
- ■ Programmable input and output delays
- ■ Open-drain outputs
- ■ DQ and DQS I/O pins
- ■ VREF pins

Cyclone II device IOEs contain a bidirectional I/O buffer and three registers for complete embedded bidirectional single data rate transfer. The IOE contains one input register, one output register, and one output enable register. You can use the input registers for fast setup times and output registers for fast clock-to-output times. Additionally, you can use the output enable (OE) register for fast clock-to-output enable timing. The Quartus II software automatically duplicates a single OE register that controls multiple output or bidirectional pins. You can use IOEs as input, output, or bidirectional pins.

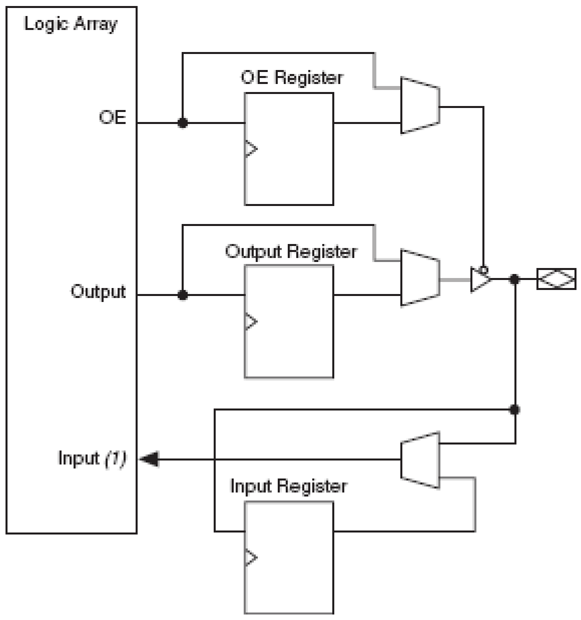


Figure A1.3 Cyclone II IOE Structure

APPENDIX 2

FPGA DEVELOPMENT BOARD

The following hardware is provided on the DE1 board. It is powered by either a 7.5V DC adapter or a USB cable

- **Cyclone II 2C20 FPGA**
  - ✓ 18,752 LEs
  - ✓ 52 M4K RAM blocks
  - ✓ 240K total RAM bits
  - ✓ 26 embedded multipliers
  - ✓ 4 PLLs
  - ✓ 315 user I/O pins
  - ✓ FineLine BGA 484-pin package
- **Serial Configuration device and USB Blaster circuit**
  - ✓ Altera’s EPCS4 Serial Configuration device
  - ✓ On-board USB Blaster for programming and user API control
  - ✓ JTAG and AS programming modes are

supported.

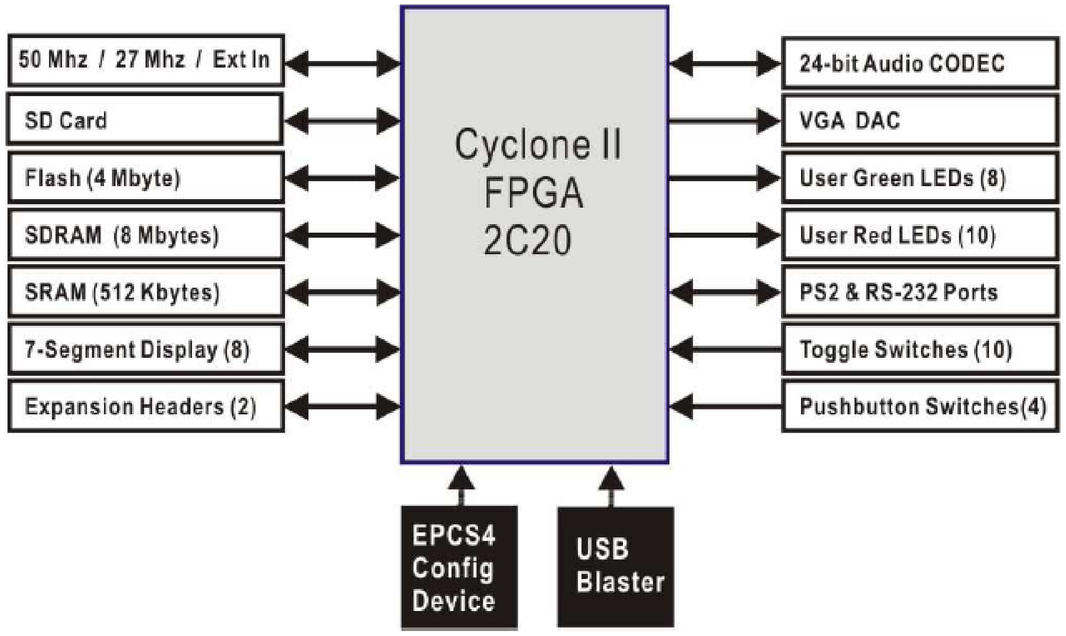


Figure A2.1 DEI BLOCK DIAGRAM

- **SRAM**
  - ✓ 512-Kbyte Static RAM memory chip
  - ✓ Organized as 256K x 16 bits
  - ✓ Accessible as memory for the Nios II processor and by the DE1 Control Panel
- **SDRAM**
  - ✓ 8-Mbyte Single Data Rate Synchronous Dynamic RAM memory chip
  - ✓ Organized as 1M x 16 bits x 4 banks
  - ✓ Accessible as memory for the Nios II processor and by the DE1 Control Panel
- **Flash memory**
  - ✓ 4-Mbyte NAND Flash memory (1 Mbyte on some boards)
  - ✓ 8-bit data bus
  - ✓ Accessible as memory for the Nios II processor and by the DE1 Control Panel
- **SD card socket**
  - ✓ Provides SPI mode for SD Card access
  - ✓ Accessible as memory for the Nios II processor with the DE1 SD Card Driver
- **Pushbutton switches**
  - ✓ Debounced by a Schmitt trigger circuit
  - ✓ 4 pushbutton switches
  - ✓ Normally high; generates one active-low pulse when the switch is pressed
- **Toggle switches**
  - ✓ 10 toggle switches for user inputs
  - ✓ A switch causes logic 0 when in the DOWN (closest to the edge of the DE1 board) position and logic 1 when in the UP position
- **Clock inputs**
  - ✓ 50-MHz oscillator
  - ✓ 27-MHz oscillator
  - ✓ SMA external clock input
- **Audio CODEC**
  - ✓ Wolfson WM8731 24-bit sigma-delta audio CODEC

- ✓ Line-level input, line-level output, and microphone input jacks
- ✓ Sampling frequency: 8 to 96 KHz
- ✓ Applications for MP3 players and recorders, PDAs, smart phones, voice recorders, etc.

▶ **VGA output**

- ✓ Uses a 4-bit resistor-network DAC
- ✓ With 15-pin high-density D-sub connector
- ✓ Supports up to 640x480 at 100-Hz refresh rate
- ✓ Can be used with the Cyclone II FPGA to implement a high-performance TV Encoder

▶ **Serial ports**

- ✓ One RS-232 port
- ✓ One PS/2 port
- ✓ DB-9 serial connector for the RS-232 port
- ✓ PS/2 connector for connecting a PS2 mouse or keyboard to the DE1 board

▶ **Two 40-pin expansion headers**

- ✓ 72 Cyclone II I/O pins, as well as 8 power and ground lines, are brought out to two 40-pin expansion connectors
- ✓ 40-pin header is designed to accept a standard 40-pin ribbon cable used for IDE hard drives
- ✓ Resistor protection is provided

REFERENCES

[1] Specht, D. F., “Probabilistic Neural Networks”, *Neural Networks,* 3(1): 109-118, 1990.

[2] G.Minchin, A.Zaknich. ”’A Design for FPGA Implementation of the Probabilistic Neural Network”, *6th International* Con*ference on Neural Information Processing* *p*ages *556 to 559*, 1999.

[3] P.D. Wasserman, Advanced Methods in Neural Computing, New York: Van Nostrand Reinhold, 1993 on pp. 155-61, and pp.35-55.

[4] Zaknich, A., "A vector quantization reduction method for the probabilistic neural network, IEEE Proceedings of the International Conference on Neural Networks (ICNN), Vol 11, Houston, Texas,USA, 9-12th June, 1997, pp. 11 17-1 120.

[5] Ryosuke Mizuno, Noriyuki Aibe, Moritoshi Yasunaga, and Ikuo Yoshihara, “Reconfigurable Architecture for Probabilistic Neural Network System”,pages 367 to 370,2003

[6] Probabilistic and General Regression Neural Networks http://www.dtreg.com/pnn.html

[7] Cyclone II Device Family Technical Information http://www.altera.com/support/devices/cyclone2/dev-cyclone2.html

[8] Quartus II Software Support http://www.altera.com/support/software/sof-quartus.html

[9] Samir Palnitkar, ‘Verilog HDL: A Guide to Digital Design and Synthesis’ Prentice Hall, New York, 1996.