Sheffield Hallam University

An FPGA based real-time image classification system

Deepayan Bhowmik

M.Sc. in Electronics and Information Technology
2005-06

First supervisor: Dr. Bala P. Amavasai
Second supervisor: Mr. Tim J. Mulroy

This thesis is submitted in partial fulfilment
of the requirements for the degree of
Masters of Science
in Electronics and Information Technology, School of Engineering,
Sheffield Hallam University

Acknowledgement

Writing an acknowledgement is probably the most difficult part of the thesis. I was thinking whether it is necessary to write a narrative acknowledgement with stylish English and properly written sentences. From an engineers point of view the best way to write something is by mentioning the facts point-wise. However finally I realised that this is the real opportunity to express my deep sense of gratitude to the people directly or indirectly involved with the project.

It is an honour to present this thesis as an MSc student in Electronics and Information technology, School of Engineering, Sheffield Hallam University, Sheffield, UK. I am thankful to the school to give me this opportunity. First of all most of my sincere thanks and regards should go to my supervisor Dr. Bala Amavasai who was a constant source of inspiration and director throughout this project even before starting it. He was the main source of motivation for this project. I am also grateful to my second supervisor Mr. Tim J. Mulroy. I am greatly indebted to my supervisors for their valuable guidance and painstaking attention for helping me in this endeavour.

This project was funded by EPSRC Nanorobotics project GR/S85696/01. I am thankful to EPSRC to allow me to carry out this project. I like to thank Dr. Saatchi Reza, Sheffield Hallam University for his valuable teaching and guidance in the field of Artificial Intelligence. Dr. John Row, Sheffield Hallam University has helped me to clear doubt about VHDL coding. My special thanks should go to Dr. Clay Gloster, RARE project, Howard University, Washington, USA to allow me to use his VHDL code for floating point division.

It is my mother and father who encouraged and motivated me for my education through out my life. For them only I have enrolled for a Master's degree. My lovely younger brother Neelanjan has made some good points (basically pinched me!) which drives me towards perfection.

Friends in Sheffield specially Subrato, Sanchita, Subroto, Ritu, Samujjwal, Sinjini, Sumon, Gautam, Rahul, Amit and Swaraj made my life miserable with continuous fun and enjoyment during my stay in Sheffield. Joking apart they are the persons without whom I may not be able to carry out and concentrate so much in this project. They are the person who gave every possible and impossible support during my bad times. Now I got this opportunity to express my gratitude and say thanks to them.

Finally I have to convey my thanks to my best pals Ashok, Nishanta, Suparna and Sanjay with whom I share each and every small thing. They gave every support to make my project as good as good as possible.

Last but not the least, I express my sincere gratitude to Machine Vision Lab researchers, lab technician Mr. Misko and all those who have directly or indirectly contributed to the completion of this project work.

Publication

[1] **Deepayan Bhowmik**, Bala P. Amavasai, T. J. Mulroy, "Real-time object classification on FPGA using moment invariants and Kohonen neural networks", Proceedings of the IEEE SMC UK-RI Chapter Conference 2006 on Advances in Cybernetic Systems, September 7-8, Sheffield, UK, ISSN 1744-9189

Abstract

Machine vision is an integral part of machine intelligence. The primary focus of any machine vision system is to recognise and classify objects in the surrounding area. The concept of Artificial Intelligence has been deployed in number of machine intelligence application keeping in mind the fact that a machine will one day be able to imitate a human being. It is also applicable for an object recognition task.

In this thesis we address the issue of object classification using a different approach. Our objective of object classification can be divided into two parts, namely image processing and recognition, or classification of processed image. The image processing stage is handled by a moment calculation of the captured images. An Artificial Neural Network is implemented in the recognition or object classification stage. The main constraint of this work is to reduce the response time and make the system as fast as possible in order to fulfill real-time objectives. The use of dedicated hardware is probably one of the possible solutions. Hence we make use of reconfigurable hardware like Field Programmable Gate Array (FPGA).

In this thesis, the use of moment invariants and Kohonen neural networks for real time object classification is addressed. The implementation of such a scheme using a reconfigurable hardware FPGA (Field Programmable Gate Array) device is described. In the image processing stage, the Hu's moment invariants algorithm has been implemented in hardware and the issues surrounding this implementation is discussed. Following the image processing stage, a neural network is employed for the classification stage. By using the Kohonen unsupervised neural network the system is essentially self-supervised and it is able to perform an all parallel neural computation for classification purposes. A discussion of the concept and real simulation results are provided.

Contents

Chapter 1: Introduction

1.1 Introduction

The requirement for the recognition and classification of objects in real-time is important for many real world tasks, especially in robotics and industrial-type applications. A dedicated hardware for this purpose is highly. The use of of artificial neural network for classification allows the system to adapt to the environment. The aim of the thesis is to explore the possibility of developing a dedicated hardware for image processing and by using artificial neural network to make it adapt to the environment. The following issues have been addressed during the course of this project:

- The use of moment invariants for image classification
- Hu's moment invariant
- The use of Artificial Neural Network (ANN) as efficient mode of image classification
- Kohonen Neural Network as self-adaptive learning system
- The use of Field Programmable Gate Array (FPGA) for designing hardware
- Timing analysis as a constraint of real-time application

1.2 Background

The greatest challenge in current times, of any algorithm and circuitry, is speed. The idea of dedicated hardware for a specific type of application is to handle complexity with better time response. One of the ways to implement and test a circuit is to map and synthesis the algorithm in FPGA (Field Programmable

Gate Array). The domain of application of FPGAs has traditionally been in digital logic and digital signal processing (DSP). DSP-type problems are easily mapped to FPGAs due to the fact that most DSP functions are made up of sum-of-product type operations that consist of simple logic. Since images are essentially 2-D signals, image processing algorithms have also been widely implemented on FPGA.

However the implementation of higher level machine vision algorithms (with ANN optimisation) that consist of a number of decision making stages is somewhat limited. This is largely due to the limited number of directly mapped arithmetic functions available and the complexity in designing algorithms that are able to adapt or optimise online, since FPGA designs are often static. The latter problem can be overcome by parameterising the algorithms and using external memory to store these parameters.

1.3 Motivation

Machine vision is one of the most interesting and current research areas where researchers are trying to develop intelligent machine. The primary focus of machine vision in industry is to classify objects as quick as possible. Real-time classification is important for many real world and industrial applications especially in robotics. A variety of blob and shape based algorithms exist, but many of these do not meet real-time constraints.

In recent times, with the advent of sophisticated software tools, the use of Field Programmable Gate Arrays (FPGAs) has changed from being simply a gluelogic type component to a vehicle for complete delivered solutions. FPGAs are made up of programmable logic components and programmable interconnects. Unlike other technologies, that force the programmer or designer to make critical decisions in the early part of development, FPGAs allow the development of the application to be adapted and improved over time.

Developing FPGA solutions is comparable to developing solutions in software (rather than hard-coding).

The use of ANN is quite interesting to improve the timing performance of the system. However the Van-Neuman computer architecture (the normal computer processor used now-a-days *e.g.* Pentium series from Intel) does not directly support the construction of parallel ANNs. As far as ANN is concerned, it performs activities are performed in parallel on each neuron and that is where a sequential computer lacks. A neuro-chip based processor might solve the problem. But for a specialized application like object classification, probably a dedicated hardware is more useful. And here comes the concept to use of FPGA in this purpose.

The above discussion is the main motivation of this project. The project tries to explore the opportunity how suitable a FPGA board is to perform an object classification with the application of ANN.

1.4 Thesis Description

This thesis, describes an attempt to implement an object classification paradigm that is able to classify objects in real-time using an FPGA solution. The process makes use of two well studied algorithms in the area of machine vision and neural networks.

Hu [1] moment invariants is one of the well proven algorithm for 2-D image processing using a non-orthogonal central moments. Seven Hu descriptors is unique given a specific pattern or shape.

Kohonen [2] unsupervised neural network is a popular self-organising unsupervised learning Neural Network. It is very useful for data classification with an all parallel computation. A K-means clustering algorithm can classify and give identity to a self organizing unsupervised system.

The computation of moment invariants has been implemented in hardware. However the Kohonen neural network algorithm is divided into two parts, namely training mode, and detection mode. Currently only training of the system is performed using MATLAB. The result of the training is used for object classification and this has been implemented (and simulated) on an FPGA device.

1.5 Programme of work

At the beginning of the project the work programme has been decided. The project follows the work program as a guide line. Following is the work program for the project.

a) Algorithm development for 2-D image recognition (Hu's moment invariant chosen here)
b) Algorithm optimisation using Artificial Neural Network (ANN)
c) Proposed ANN optimisation: Kohonen Neural Network
d) Implementation of Hu's moment invariants in FPGA
e) Hardware realization using VHDL coding
f) VHDL implementation of Kohonen ANN
g) Initial training of the system by MATLAB
h) Simulation and testing of VHDL code
i) Development and simulation the Stratix FPGA board in Altera Quartus software
j) Timing analysis for next pahse

1.6 Deliverables

The deliverables from this project are listed as follows

a) A real time 2-D object classification with very small response time
b) An implementation of Hu's moment invariant in FPGA board
c) An implementation of artificial neural network in FPGA board
d) An algorithm optimisation for image processing using ANN

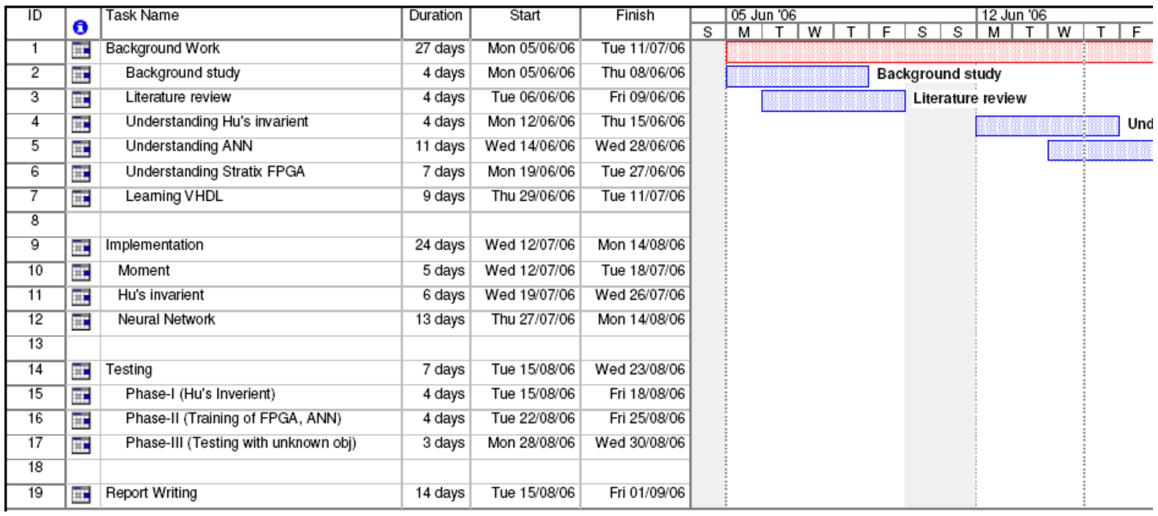
e) Dedicated hardware for real time image classification

1.7 Thesis Formulation

This section discusses constraints such as schedule, technical limitations, possible hazards etc. and develops a work plan for the entire thesis.

1.7.1 Time/Schedule

Table 1.1 shows the Gantt chart of the thesis time line.



ID		Task Name	Duration	Start	Finish
1		Background Work	27 days	Mon 05/06/06	Tue 11/07/06
2		Background study	4 days	Mon 05/06/06	Thu 08/06/06
3		Literature review	4 days	Tue 06/06/06	Fri 09/06/06
4		Understanding Hu's invarient	4 days	Mon 12/06/06	Thu 15/06/06
5		Understanding ANN	11 days	Wed 14/06/06	Wed 28/06/06
6		Understanding Stratix FPGA	7 days	Mon 19/06/06	Tue 27/06/06
7		Learning VHDL	9 days	Thu 29/06/06	Tue 11/07/06
8					
9		Implementation	24 days	Wed 12/07/06	Mon 14/08/06
10		Moment	5 days	Wed 12/07/06	Tue 18/07/06
11		Hu's invarient	6 days	Wed 19/07/06	Wed 26/07/06
12		Neural Network	13 days	Thu 27/07/06	Mon 14/08/06
13					
14		Testing	7 days	Tue 15/08/06	Wed 23/08/06
15		Phase-I (Hu's Inverient)	4 days	Tue 15/08/06	Fri 18/08/06
16		Phase-II (Training of FPGA, ANN)	4 days	Tue 22/08/06	Fri 25/08/06
17		Phase-III (Testing with unknown obj)	3 days	Mon 28/08/06	Wed 30/08/06
18					
19		Report Writing	14 days	Tue 15/08/06	Fri 01/09/06

Table 1.1: Gantt chart of the project time line

1.7.2 Technical Limitations

This section focuses on various limitations of the thesis as listed below

- Difficulty on hardware realization of sequential algorithm
- Handling floating point arithmetic in VHDL, specially multiplication and division operation
- Handling function and process statements in VHDL
- Large compilation and simulation time made the experiment time longer

1.7.3 Potential Hazards

The precautions that were strictly followed during the entire project period were

- An erect sitting posture was maintained while working on the computer
- Breaks were taken at regular intervals to avoid cramps and sprains that can be caused due to sitting in front of the computer for long hours
- Hazards related for handling electronic equipment like FPGA (i.e. electro static discharge, ground problem etc)
- Hazards related to main power supply

1.8 Thesis organisation

The thesis has been written in the same way the project has been carried out. Chapter 2 discusses the background of the work. Details of literature review and discussion of similar kind of work has been described here. Chapter 3 is basically the discussion of theories namely Hu's moment invariant, Kohonen Artificial Neural Network *etc*. Based on these well proven theories the next part of the project was carried out. Algorithmic development & implementation is probably the main feature of the project. All details about the methodology and Algorithmic development are written in chapter 4. So far the thesis frames the idea and suggested the implementation method. Chapter 5 shows the results after simulation and does the timing analysis of the same. Finally chapter 6 concludes the project and suggests some future work.

Chapter 2: Literature Survey

2.1 Introduction

The chapter focuses on the previous work by other researchers in the area of this thesis. The literature survey discusses the image classification method on FPGA board and related hardware using Artificial Neural Network concept. In an overall view, the main limitation faced by researcher in the field is the hardware capacity of the FPGA board. However number people have developed an integrated system combined with a computer.

2.2 General Discussion

The domain of application of Field Programmable Gate Arrays has traditionally been in digital logic and digital signal processing. Digital Signal Processing type problems are easily mapped to FPGAs due to the fact that most Digital Signal Processing functions are made up of sum-of-product type operations that consist of simple logic. Since images are essentially 2-D signals, image processing algorithms have also been widely implemented on FPGA.

The implementation of higher level machine vision algorithms that consist of a number of decision making stages is more limited. This is largely due to the limited number of directly mapped arithmetic functions available and the complexity in designing algorithms that are able to adapt or optimise online, since FPGA designs are often static. Parameterising the algorithms and using external memory to store these parameters can overcome this problem.

Another limitation with FPGA is to handle floating point arithmetic. The signal processing calculations are sometimes quite complicated and deal with lots of floating point number. The precision of floating point values is also important in this regards. The researchers in the field have always faced difficulty dealing with floating point arithmetic. In case of floating point calculation IEEE 754 format is followed and the arithmetic with IEEE 754 is different from a normal arithmetic calculation. This is probably the reason why hardware design engineers use to hate floating point numbers where as software engineers love to play with floating points.

Floating point can be described in IEEE 754 single precision (32 bit) or a double precision (64 bit) or even with an extended single precision (43 bit). As the floating points increase complexity of the design, often it is preferred to deal with single precision number. The next section discussed about different approach followed and explored by previous researchers in this area.

2.3 Relevant Past Work

In recent times Hirai *et. al.* [2] have designed complete single-task vision systems on FPGA, with the objective of detecting the position and orientation of distinctly visible planar objects. Their system consisted of three parts, namely the computation of image gravity centre, the detection of object orientation using radial projection and the computation of the Hough transform, albeit discrete. They reported being able to process images at a rate of around 200 fps, far more than the standard frame-rate of a PAL camera.

In a separate study, Arribas and Maciá [3] implemented the Santos-Victor paradigm to compute motion fields for use in robot guidance applications. The technique required the computation of optical flow fields in real-time. The flow fields were computed using both standard differential and correlation methods.

Recent approaches have been made towards the implementation of real-time object recognition using reconfigurable hardware. Most of these systems have been implemented using multiple FPGA boards or similar devices together with a sequential computer, due to the limited number of gates available on each device.

For instance, Neema *et. al.* [6] have used multiple Digital Signal Processing processors and a sequential computer for their Automated Image Recognition system. They have described the model-based development of a real-time, embedded Automated Image Recognition (AIR) system. The idea behind a distributed processing using multiple Digital Signal Processing board is to implement a system capable of processing high sampling rate and large input images. DSP processors are used here to handle the complexity of the applied Discrete-classifier correlation filter algorithm. However the full system integration is only possible with a sequential computer along with the DSP boards.

In another paper, Yasin *et. al.* [7] have demonstrated an effective FPGA prototype for iris recognition. However, the system itself is not completely hardware based, since images are preprocessed using MATLAB, and FPGAs are only used for the recognition part. In their design, the authors make use of Multi Layer Perceptrons (MLPs) for classification. The numbers of neurons are limited due to hardware constraints and hence the computation complexity had to be reduced for implementation on an FPGA board.

Jean *et. al.* [8] implemented a system to accelerate the recognition process in infra-red images using an FPGA device. However once again, the system is dependent on a separate computer and in order to reduce complexity, the mathematical calculations have been performed with full precision integer values instead of floating point operations.

Y. Y. Chung *et. al.* [10] describes and implemented a partially conected neural network by Giga-Ops Spectrum G800 FPGAs based custom computer for a high speed neural network based classifier which can be used real time application. Again the custom computer consists multiple (up to 32) Xilinx logic chips. They demanded that the system can provide a very high speed classifier for real time image recognition purpose.

Neural networks are an ideal example where the use of FPGAs can offer an advantage. This is due to the fact that the operations carried out are largely parallel. A number of efforts have been made for mapping neural networks onto hardware. For example Aibe *et. al.* [9] implemented a probabilistic neural network in FPGA hardware which exhibited a parallel architecture.

2.4 Summary

A brief literature review has been carried out in this chapter. The main focus is on use of FPGA for the implementation of image processing algorithm and Artificial Neural Network. Researcher has tried to implement a parallel and dedicated hardware in order to improve the speed over a sequential computer. But the main limitation faced by almost everybody is the number of gates available in FPGA. Number of times either people used multiple boards and a sequential computer to overcome this limitation. Interestingly everybody tries reconfigurable hardware namely Field Programmable Gate Arrays to design a dedicated hardware due the high cost of manufacturing a customised chip. The interests of researcher's show that the real-time image recognition is one of most lucrative and interesting field for FPGA application.

Chapter 3: Theoretical Discussion

3.1 Introduction

The aim of this chapter is to discuss the theory of the implemented concept and algorithms of the project. The project is clearly separated in two different parts namely moment calculation and object classification. The moment classification is based on Hu's Moment Invariants where as the object classification part has been dealt with Kohonen Artificial Neural Network. Both the algorithms are well proven and quite popular in its won field. The K-means clustering algorithm discussed here is useful for object classification in training mode. A brief discussion about Field Programmable Gate Array (FPGA) and Atera Stratix board is then carried out. One of the main issues faced during the VHDL coding was floating point arithmetic. A brief description on floating point arithmetic hardware, specially the details about IEEE 754 format might be useful at this stage.

3.2 Hu's Moment Invariant

In the field of image processing specially in machine vision and related fields, image moments are popular as it has some unique property or interpretation [16]. The moments are basically certain particular weighted averages (moments) of the image pixels' intensities or function of other moments. Moments can describe the object efficiently after segmentation.

One of the well studied methods of describing 2-D objects using moment invariants is Hu's moment invariant or descriptors [1]. The regular moment of a shape in an M by N binary image is defined in equation 3.1:

$$u_{pq} = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} i^p j^q f(i,j) \qquad \text{(eqn. 3.1)}$$

Where f(x; y) is the intensity of the pixel (either 1 or 0) at the coordinates (x; y) and p+q is said to be the order of the moment.

The calculation is a function of the distance between shape pixels. So the origin measurements are taken relative to the shapes centroid (x'; y') to remove translational variability. The coordinates of the centroid are determined using the equation above as described in equation 3.2:

$$i' = \frac{u_{10}}{u_{00}} \quad \text{and} \quad j' = \frac{u_{01}}{u_{00}} \qquad \text{(eqn. 3.2)}$$

Now the relative moments are calculated with respect to the central moments and the same is described in equation 3.3

$$u_{pq} = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} (i - i')^p (j - j')^q f(i,j) \qquad \text{(eqn. 3.3)}$$

Normally individual moment values do not have the descriptive power to uniquely represent arbitrary shapes, nor do those posses the required invariance characteristics, but, sets of functions based on these moments can be determined which do [17]. Hu derived a set of seven rotational invariant moment functions which form a suitable shape representation (or vector). Hu descriptors are based on non-orthogonalised central moments that are invariant towards rotation, translation and scale. Hu descriptors are thus computed from normalised centralised moments up to the third order, and consist of seven moments in total. The seven formulae given in equation set 3.4 to 3.10 are normally used for algorithmic implementation.

$I_1 = \eta_{20} + \eta_{02}$ (eqn. 3.4)

$I_2 = (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2$ (eqn. 3.5)

$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$ (eqn. 3.6)

$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$ (eqn. 3.7)

$I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$ (eqn. 3.8)

$I_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})]$ (eqn. 3.9)

$I_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$ (eqn. 3.10)

The derivation of these seven invariants and the computation of η is available in [1], and so will not be described here. The combination of these seven Hu descriptors is unique given a specific pattern or shape. Hence it can be used in conjunction with a matching scheme to uniquely identify the object being input to the system. Computing these descriptors is straightforward and fast, and hence it is widely used in the area of real-time vision in robotic applications [18].

3.3 Kohonen Artificial Neural Network

Kohonen Artificial Neural Network is a popular unsupervised learning neural network. The Kohonen ANN is essentially a self-organizing unsupervised mapping system that can map input vectors of arbitrary length onto a lower dimension map. It is frequently described as a sheet-like neural network array. Patterns that are close together in Euclidean space will remain close in the final map, and are topologically ordered. The learning process of Kohonen ANNs optimizes the mapping until the weight change becomes negligible [11].

The Kohonen Neural Network is consists of a Kohonen map which has a single layer of neurons. All input vectors are connected to each neuron in the map. Kohonen map can be arranged in different topologies namely rectangular and hexagonal. A neuron map of rectangular shape is quite popular as shown in figure 3.1. The weight vector connected to each neuron is represented by (w_{ji}). The learning algorithm is determined by the Euclidean distance between an input vector and the corresponding weights of each neuron. The weights associated with the neuron which provides the smallest Euclidean distance is considered as the winning node. Then the weights associated with the winning neurons are updated. This weight update method has done in such a way that the winning neuron becomes representative of a specific pattern or class.

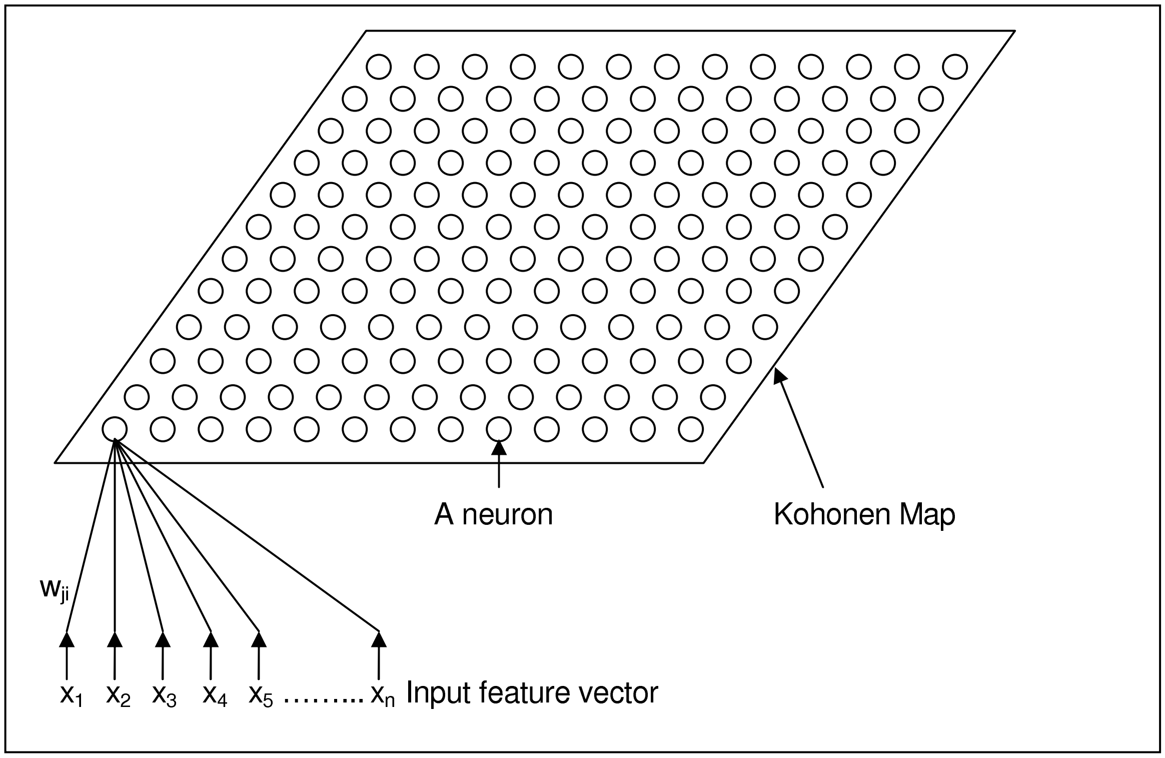

A neuron
Kohonen Map
w_{ji}
x_1 x_2 x_3 x_4 x_5 ……... x_n Input feature vector

Figure 3.1: A Kohonen network

Though the winning node is main point of interest, in practice the weights associated with the neighborhood region of the winning node has also updated in a similar fashion. This method is quiet useful to specify a region of the

Kohonen map to be associated with different pattern or class. Figure 3.2 has shown a complete idea about the winning node and its neighborhood region.

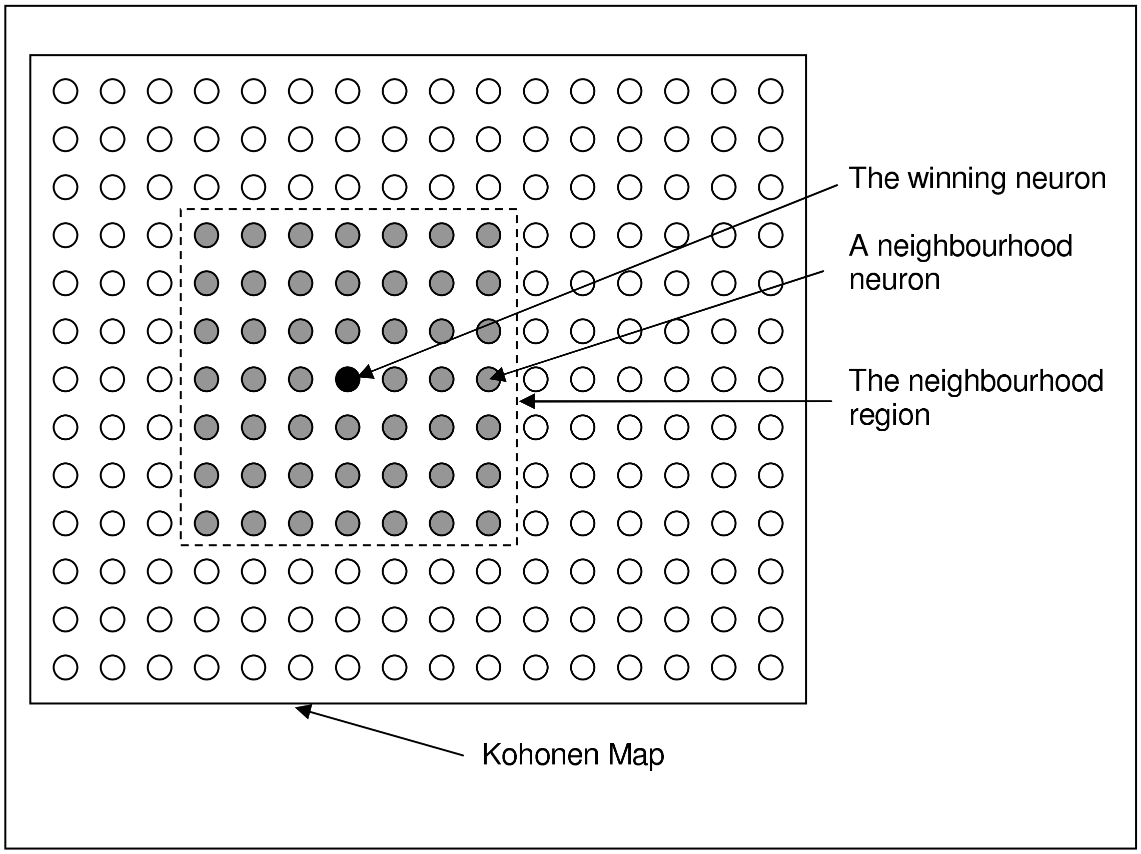


Figure 3.2: A Kohonen network with winning node and neighbourhood region

The normal method to deal with Kohonen map is to initialize weights of the neurons and standard neighborhood region. The initial weight values are chosen randomly with a value between 0.45 and 0.55. A neighborhood size of 5 is fair enough for a Kohonen map of 10 x 10 neurons.

The inputs data is mostly normalize for the application. Normally this normalization method varies in different application. All input data are given to the Kohonen map to all neurons.

The next step is to determine the Euclidean distance between the weight vector and the input vector of each neuron. Equation 3.11 represents the Euclidean distance measurement equation. Basically this computes the Euclidean distance (d_j) between the input pattern (x) and the network connection weights w_{ij} of each neuron. The connection to *j*th neuron from *i*th input feature of the pattern is represented by the suffix *ij.* As described earlier the winning node or winning neuron will be the neuron associated with the smallest Euclidean distance. Equation 3.11 computes the value of (d_j) for *j*th neuron with each input pattern has N elements.

Euclidean distance: $$d_j = \sqrt{\sum_{i=0}^{N-1}(x_i - w_{ij})^2} \quad \text{(eqn. 3.11)}$$

Where x_i is the input to *i* and w_{ij} is the weight from input node *i* to output node *j.*

For simplicity of the calculation sometimes Euclidean distance calculation can be replaced by Manhattan distance. This is basically the summation of the absolute distances (d_j) measured between the input vector x_i and the weight vector w_{ij}. The Equation 3.12 represents the mathematical form of Manhattan distance calculation.

Manhattan distance (“L1 Norm”): $$\sum_{i=1}^{k}| x_i - y_i | \quad \text{(eqn. 3.12)}$$

Now the weights of the neurons in the Kohonen map needs to be updated. The neuron corresponding to the minimum distance is the winning node. The weights of the winning node and the neighbourhood region around the winning node are updated with the following update rule referred in Equation 3.13:

$$w_{ij}(n+1) = w_{ij}(n) + \lambda(n)(x_i(n) - w_{ij}(n)) \quad for 0 \le i \le N-1 \quad \text{(eqn. 3.13)}$$

Where $w_{ij}(n+1)$ is the updated weight, $\lambda(n)$ is the learning rate and the term $(x_i - w_{ij})$ represents the error. The value of the learning rate normally lies between 0 and 1 and it controls convergence speed and stability. The value of the learning rate is normally determined experimentally. The learning process is iterative and continues until the network has converged satisfactorily. At the same time at predefined intervals the learning rate and the neighborhood size are gradually reduced. Normally a monitoring process has employed at regular intervals for the performance of the network. A network is said to be converged when the patterns of the same category active neurons in the same regions of the Kohonen map [11].

3.4 K-Means Clustering

Clustering is an unsupervised learning technique which allows a set of recorder data to be partitioned into two or more group [11]. The idea has been illustrated in figure 3.3

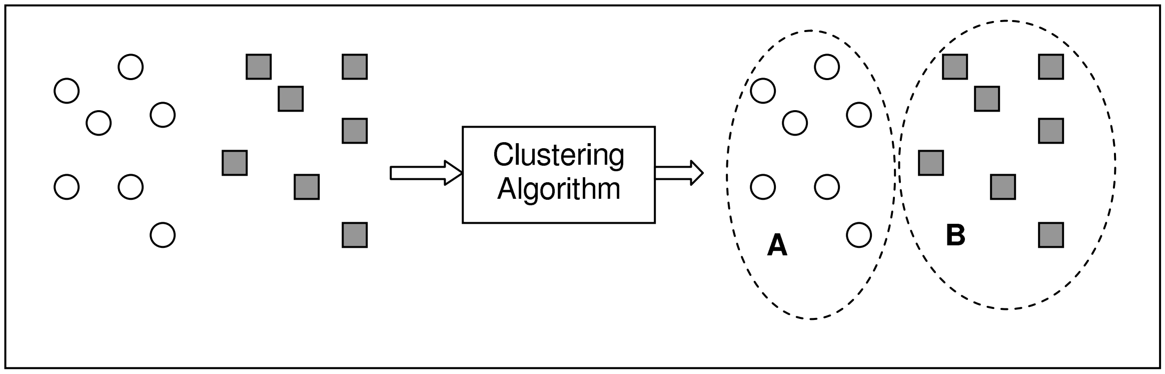


Figure 3.3: The role of clustering algorithm

The K-means algorithm clusters the objects based on attributes into k partitions [15]. The goal of this algorithm is to determine the k means of data generated from Gaussian distributions. The object attributes are asumed to be in a vector space. It tries to minimize total intra-cluster variance which can be represented by the equation 3.14

$$V = \sum_{i=1}^{k} \sum_{j \in S_i} | x_j - \mu_i |^2 \qquad \text{(eqn. 3.14)}$$

Where there are k clusters S_i, i = 1, 2... k and μ_i is the centroid or mean point of all the points $x_j \in S_i$.

The k-Means algorithm starts by partitioning the input points into k initial sets, either at random or using some heuristic data and calculates the mean point or centroid [15]. A new partition is then formed by associating each point with the closet mean. The algorithm is then repeated until convergence which is obtained when the points now longer switch between clusters.

3.5 Field Programmable Gate Array (FPGA)

A Field Programmable Gate Array (FPGA) is a reconfigurable hardware which containing programmable logic and programmable interconnects [14]. A combinational or sequential logic can be programmed with the help of basic gate functions. The FPGAs are capable of handling a hardware implementation of simple or complex mathematical function and formulae. In most of the FPGAs the programmable logic components (commonly called logic elements [LE]) consists of basic gates like AND, OR, XOR etc. and also include memory element like a simple flip-flop or more complex block of memories.

The programmable interconnects with proper hierarchy allows logic elements (LE) of the FPGA to be interconnected as required by the hardware designer. The logic elements and the interconnections among them can be programmed after the manufacturing process by the designer. A custom made design is possible depending on the application requirement. And hence it is called field programmable and so the FPGA can perform whatever logical function is needed. Though FPGAs are slower than a application specific integrated circuit (ASIC) and some more pros, designer prefer to use an FPGA because of their reconfiguration facility (probably easy to fix a bug) and lower non-recurring cost.

Now-a-days FPGAs have a very wide range of application field including DSP, software defined radio, aerospace, defence systems, ASIC prototyping, medical imaging, computer vision, bio-informatics etc. [14]. FPGAs especially find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture.

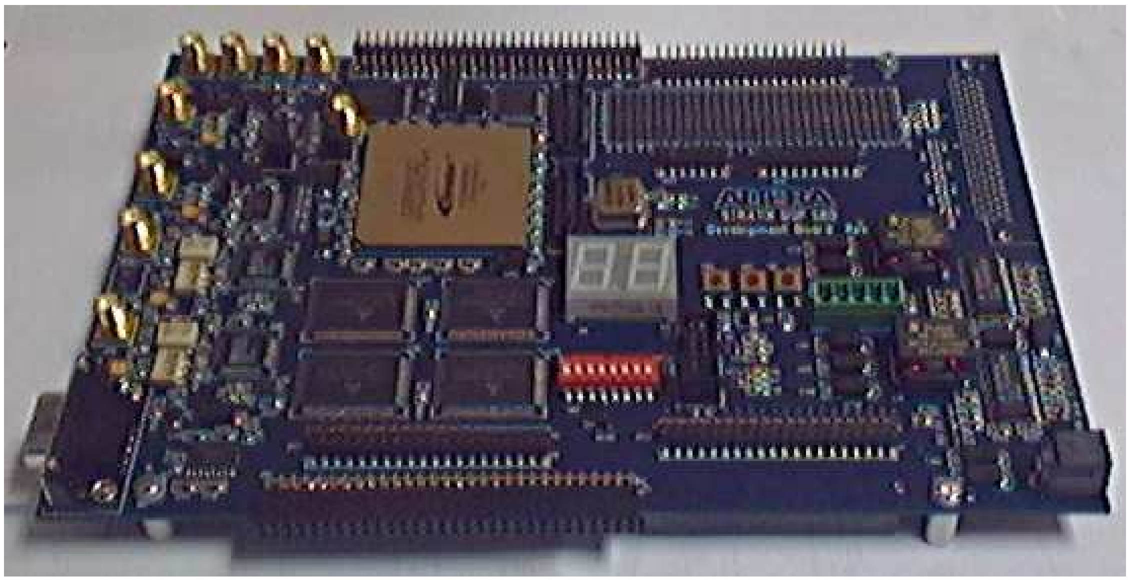


Figure 3.4: the Altera EP1S80 development board

The FPGA hardware used in this thesis is the Altera Stratix EP1S80 (by EPSRC Nanorobotics project GR/S85696/01), shown in Figure 3.4. The EP1S80 device from Altera Corp. is one of the largest 0.13-micron FPGA devices available. This board consists of 79,040 logic elements (LEs), 7.2 Mbits of embedded RAM, and 1,238 user I/Os. This board is a powerful development platform for digital signal Funded processing (DSP) designs, and features the Stratix EP1S80 device in the fastest speed grade (-6) 956-pin package. It consists of two 12-bit 125-MHz A/D converters, two 14-bit 165-MHz D/A converters, single ended or differential inputs and single-ended outputs, 2 MBytes of 7.5-ns synchronous SRAM configured as two independent 36-bit buses, 64 Mbits of flash memory, dual seven-segment display, one 8-pin dipswitch, three user-definable pushbutton switches, one 9-pin RS-232 connector, two user-definable LEDs, on-board 80-MHz oscillator and a single 5-V DC power supply. For debugging interfaces, two Mictor-type connectors for Hewlett Packard (HP) logic analyzers and several 0.1- inch headers are available.

3.6 IEEE 754 Floating Point Format

IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most common representation for real numbers in computers. IEEE 754 specifies four formats for representing floating-point values: single-precision (32-bit), double-precision (64-bit), single-extended precision (≥ 43-bit, not commonly used) and double-extended precision (≥ 79-bit, usually implemented with 80 bits) [12]. The main IEEE standard is described under the title of IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985), and it is also known as IEC 60559:1989, Binary floating-point arithmetic for microprocessor systems (originally the reference number was IEC 559:1989) [12].

This part of the thesis will describe the normal formats used in 32 bit single precision floating point number. Normally IEEE 754 numbers have three basic components:

- Sign
- Exponent
- Mantissa

Usually the Mantissa is composed of the fraction and an implicit leading digit [13].

There are two basic number formats in IEEE 754: single precision and double precision. Table 3.1 shows the formats for single and double precision floating point numbers. The number of bits for each filed are shown below where as bit ranges are in square brackets.

	Sign	Exponent	Fraction
Single Precision	1[31]	8[30-23]	23[22-0]
Double Precision	1[63]	11[62-52]	52[51-0]

Table 3.1: IEEE 754 floating point number format

The sign bit represents the sign of the number. 0 denotes the positive number, where as a 1 denote a negative number.

The next field is the exponent field which needs to be represented both positive and negative exponents. For this purpose a bias is added to the actual exponent to get the stored exponent. A bias value of 127 is added for a single precision number (8 bit) and a double precision number (11 bit exponent) has a bias of 1023. For example a stored value of 167 indicates an exponent of (167 - 127) or 40 in single precision value. On the other hand an exponent of 5 means that (127 + 5) or 132 is stored in the exponent field in a single precision value.

The Mantissa represents the precision bits of the floating points. The Mantissa normally has an implicit bit and 23 fraction bits for a single precision number or 52 fraction bits for a double precision number. The floating point numbers are typically stored in normalized form to put the radix after the first non-zero digit. In a base 2 binary number has only possible non-zero number '1' and thus the IEEE 754 format does the optimization by assuming a leading digit of 1 where it is not represented explicitly. So the Mantissa has effectively 24 bit resolution by way of 23 fraction bits.

There are some special cases available in IEEE 754 format and those are listed below for reference only:

- Zero
- NaN (Not a Number)
- Demoralized number

3.7 Summary

This chapter here has discussed about the related theory of the implemented algorithm. The algorithm is mainly based on two popular and well studied theories namely Hu's moment invariant and Kohonen Neural network. The theories discussed here tried to give a general overview rather than a detail derivation or analysis.

Chapter 4: Methodology and Algorithmic Development

4.1 Introduction

Methodology and Algorithmic Development have been discussed and described in this chapter. The basic system diagram of the complete architecture of the proposed vision system is shown in Figure 4.1. As mentioned previously, classifying of objects consists of two modes, the training mode and the classification mode. The Otsu thresholder is commonly used to automatically binarise images by optimising a threshold level.

The whole implementation procedure has been divided in 3 parts:

i) Moment invariant computation,

ii) Training of the system

iii) Classification mode.

4.2 Moment Invariant Computation

In this mode, sample 2-D objects (patterns) are captured via a composite camera. The gray-valued image is then thresholded and the binarised values are presented to the system. The Hu descriptors that represent the pattern in the image are then extracted.

Floating point calculations have been achieved by using the Altera Megacore library or by using the author's own floating-point implementation. Implementation in hardware has been achieved using VHDL code with Altera Quartus II software.

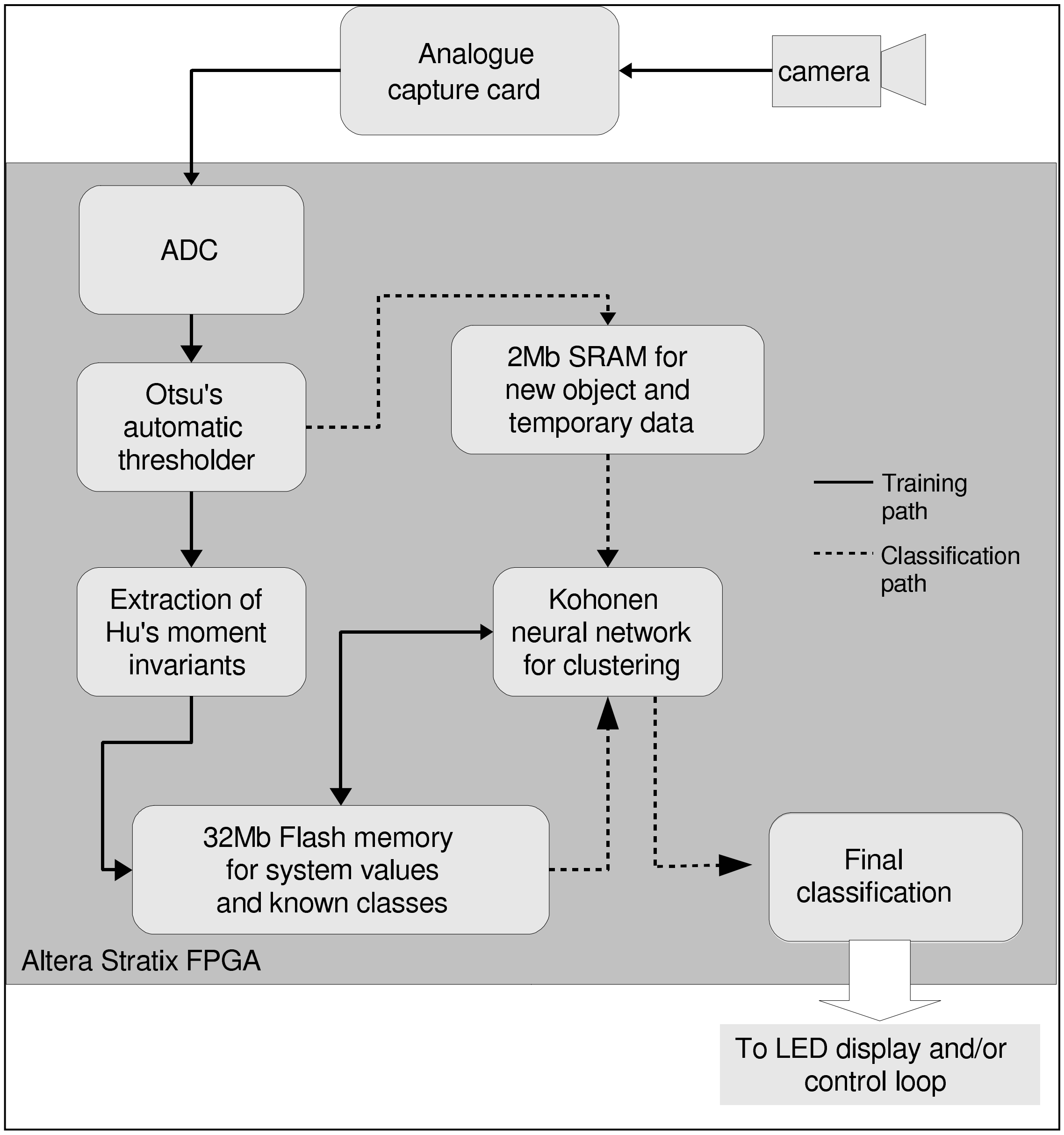

Analogue capture card
camera
ADC
Otsu's automatic thresholder
2Mb SRAM for new object and temporary data
Training path
Classification path
Extraction of Hu's moment invariants
Kohonen neural network for clustering
32Mb Flash memory for system values and known classes
Final classification
Altera Stratix FPGA
To LED display and/or control loop

Figure 4.1: Training and classifying objects (the proposed system)

To ease and reduce computation, the moment computation is simplified by replacing all floating point powers with its nearest integer values. This has been experimentally observed to produce good results.

Virtually all the operations of the algorithm have been mapped in parallel. This parallel operation significantly increases the speed of the system. Single precision floating point values have been used for all computation and conform to the IEEE 754 standard.

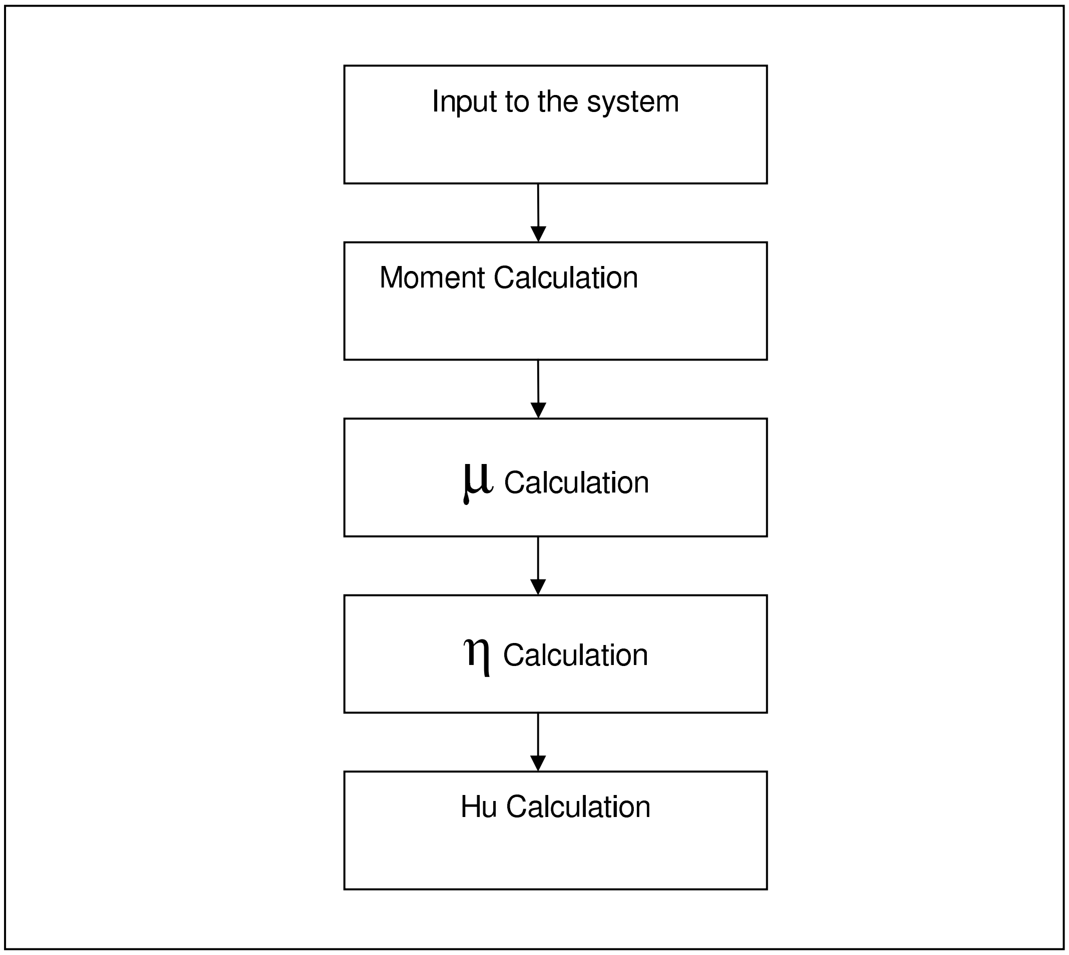


Figure 4.2: Algorithm for moment calculation.

As this work is a proof of a concept implementation, the input image matrix is hardcoded in VHDL. The computed seven moments have been used in the next stage of the system either for training (in the training mode) or classification (detection mode). The algorithm for the moment computation is shown in figure 4.2. The input data is iterated through in a sequential manner and then the different moment values have been calculated. After obtaining all moments the computation of the μ parameters (essentially high order moments) are computed in parallel mode. The μ values are required to compute η as shown in equation [3.4 to 3.10]. The final seven Hu moment invariants are calculated and passed on to the next phase.

4.3 Training of the System

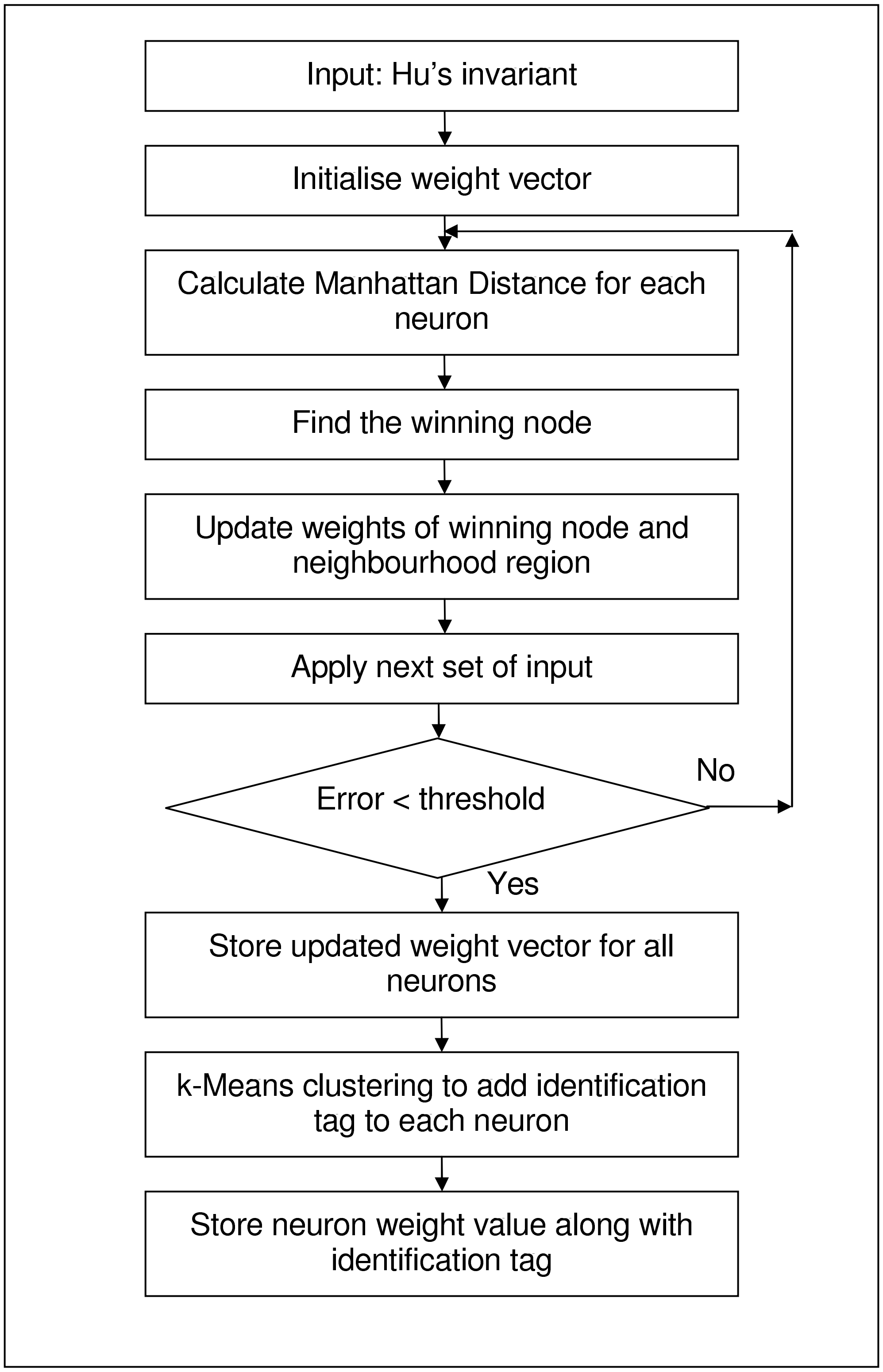

Input: Hu's invariant
Initialise weight vector
Calculate Manhattan Distance for each neuron
Find the winning node
Update weights of winning node and neighbourhood region
Apply next set of input
Error < threshold
No
Yes
Store updated weight vector for all neurons
k-Means clustering to add identification tag to each neuron
Store neuron weight value along with identification tag

Figure 4.3: Algorithm for Kohonen training and clustering

The training of the Kohonen ANN is currently implemented in MATLAB but the generated weight vectors are used in hardware. The proposed system is organised as follows. The Kohonen neural network will self-organise around the descriptors. The map that is produced is then stored onto non-volatile flash memory, so that it may be used later during the classification. The algorithm for the training part is shown in figure 4.2. A 2-dimensional neuron map is mapped with a random weight initialisation (The random weights are between 0.45 and 0.55). Seven weights are associated with each neuron. The input parameters are then fed to the ANN to find the Manhattan distance (Equation 4.1) for each neuron. The Manhattan distance is given as:

Manhattan distance ("L1 Norm"): $\sum_{i=1}^{k} | x_i - y_i |$ (eqn. 4.1)

The neuron corresponding to the minimum distance is the winning node. The weights of the winning node and the neighbourhood region around the winning node are updated with the following update rule:

$$w_{ij}(n+1) = w_{ij}(n) + \lambda(n)(x_i(n) - w_{ij}(n)) \qquad \text{for } 0 \le i \le N-1 \qquad \text{(eqn. 4.2)}$$

Where $w_{ij}(n+1)$ is the updated weight, $\lambda(n)$ is the learning rate and the term $(x_i - w_{ij})$ represents the error. The value of the learning rate normally lies between 0 and 1 and it controls convergence speed and stability. The learning process is iterative and continues until the network has converged satisfactorily.

To identify the different sets of neurons, a k-Means clustering algorithm is applied and one identification tag is attached to each neuron. The Manhattan distance measurement is applied again for clustering purpose. The k-Means clustering algorithm has been discussed in chapter 3.

4.4 Classification Mode

In classification mode, the weight vector map is first recalled from flash memory. Incoming patterns are stored onto the high speed SRAM in order to increase performance. The new pattern is then matched against the patterns stored on the Kohonen map and a final classification is produced. As described earlier a set of eight parallel neurons have been used for the detection of an unknown object. After obtaining the moment invariants of the unknown images it is passed to the inputs of the neurons. The Manhattan distance is then measured for each neuron. The minimum distance is now detected and the associated neuron with the minimum distance is declared as the winning node. The identification tag is then simulated and corresponding outputs are activated. Figure 4.4 represents the algorithm for classification mode.

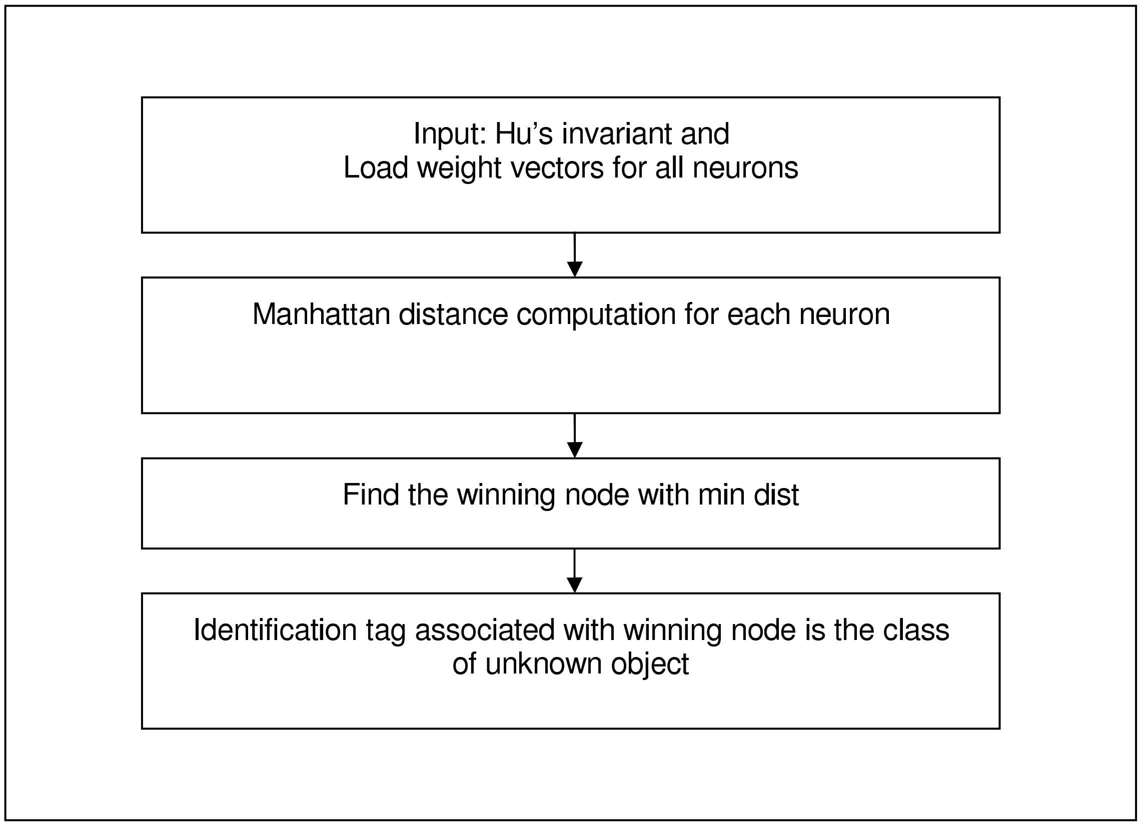

Input: Hu's invariant and
Load weight vectors for all neurons
Manhattan distance computation for each neuron
Find the winning node with min dist
Identification tag associated with winning node is the class of unknown object

Figure 4.4: Algorithm for classification mode

The complete system response and timing analysis is discussed in the results and discussion section. It has been observed that the response time for the system yields a good result due to the parallel design.

4.5 General Discussion on VHDL Implementation of Algorithm

The project has been coded in VHDL. VHDL is a very powerful high level hardware description language. Some problems have been addressed during the code implementation. These are described below for future reference.

- VHDL code normally generates all parallel hardware unlike a sequential C / C++ compiler. For implementation of a sequential algorithm, it is necessary to handle with the system clock information. A clock transition can control the sequential operation.
- Software engineers probably love to play with floating point numbers. On the contrary a hardware engineer does not like to handle floating points. IEEE 754 numbering formats are followed to implement a floating point number. As off now a floating point calculation is not included in VHDL IEEE library. As a result arithmetic operations need to be designed separately.

  In this project, maximum calculations are based on floating point values. Altera Megacore library provides some of the floating point arithmetic like addition, subtraction, multiplication etc. The project used a floating point division code from RARE project [19]. Other floating point operations like integer to floating conversion etc. are implemented by the author.
- Use of floating point library from Altera (specially Multiplication DSP block) demands more logic elements (LE). Currently the code can not be compiled fully in a Stratix board. In stead we used Stratix II environment. An optimisation of the code and separate floating point multiplication

implementation will be useful to accommodate and compile the full code in a Stratix board only.

- In some cases the same hardware block can be used instead of generating the hardware every time. This is mainly possible where a sequential logic works.

4.6 Explanation of VHDL Code

In this part of the thesis the implemented VHDL code has been explained briefly. For the testing purpose, images have been predefined in the code. The following part of the code has defined a 20x20 matrix of image.

```
----
            pic1(0) <=      "0001111111111111";
            pic1(1) <=      "0011111111111111";
            pic1(2) <=      "0010111111111111";
            pic1(3) <=      "0011111111111111";
            pic1(4) <=      "0011111111111111";
            pic1(5) <=      "0011111111111111";
            pic1(6) <=      "0011011111111111";
            pic1(7) <=      "0011111111111111";
            pic1(8) <=      "0111111111111111";
            pic1(9) <=      "0010111111111111";
            pic1(10) <=     "0011111111111111";
            pic1(11) <=     "0011101111111111";
            pic1(12) <=     "0011111111111111";
            pic1(13) <=     "0011111101111111";
            pic1(14) <=     "0011111111111111";
            pic1(15) <=     "0111111111111111";
----
```

The initial calculation is based on the black pixels available on the white background. Though normally white pixels are represented by '1' and black by '0' , here black is represented by '1' and white is '0' for testing purpose. The following part of the code has shown the sequential operation of the initial calculation and accumulation of moment values. The sequential operation is handled by the system clock:

```
process(clk,en_signal,count,en_count,x1,x2,x3,y1,y2,y3,m11,m12,m13……)
begin
        if(clk'EVENT AND clk='1') then  --the calculations are based on clock cycle
                en_count<=en_count+1;
```

```
                    if(y<16)then
                    if(x<16)then

                    if(pic1(y)(x)='1' AND en_count=1)then
                        en_signal<='1';
                        count<=count+1;        --1 clock cycle is simulated in 10ns

--Calculations start here || Pipeline architecture used here

                        y1<=y;
                        x1<=x;


--First phase data update, assumed clock cycle 4

                        if(count=1)then
                            y2<=y1*y1;
                            x2<=x1*x1;

                            m11<=m11+1;
                            m12<=m12+y1;
                            m21<=m21+x1;
                            m22<=m22+x1*y1;

                        end if;

--2nd phase data update, assumed clock cycle 8

                        --------

--3rd phase data update, assumed clock cycle 8

                    -----------------

                end if;

            end process;
```

The calculation so far has been done with integer number. To calculate eta and Hu's invariant, we need to have floating point number. A function called *int2fp*.vhd converts integer to 32 bit single precision floating point.

```
--------------------------------------------------------------------------------
-       Integer to IEEE 754 Floating Point format conversion
--------------------------------------------------------------------------------
ENTITY int2fp IS

PORT (input : integer range 0 to 100000;

         fp_op : OUT STD_LOGIC_VECTOR(31 downto 0));
```

```
END int2fp ;


ARCHITECTURE Behavior OF int2fp IS

signal int_ip : STD_LOGIC_VECTOR(31 downto 0);
signal sign_bit : STD_LOGIC ;
signal exponent : STD_LOGIC_VECTOR(7 downto 0);
signal mentissa : STD_LOGIC_VECTOR(22 downto 0);

BEGIN

        int_ip<=(conv_std_logic_vector(input,32));

        sign_bit<='0';

        process(int_ip)
        begin
                if(int_ip(31)='1')then
                        exponent<=X"9E";
                        mentissa<=int_ip(30 downto 8);

                elsif(int_ip(30)='1')then
                        exponent<=X"9D";
                        mentissa<=int_ip(29 downto 7);

                --------------
                ---------------------

                elsif(int_ip(1)='1')then
                        exponent<=X"80";
                        mentissa(22)<=int_ip(0);
                        mentissa(21 downto 0)<=(OTHERS=>'0');

                elsif(int_ip(0)='1')then
                        exponent<=X"7F";
                        mentissa(22 downto 0)<=(OTHERS=>'0');

                end if;

        end process;

fp_op(31)<=sign_bit;
fp_op(30 downto 23)<=exponent;
fp_op(22 downto 0)<=mentissa;


END Behavior;
```

The next part of the code implements the eta calculation and Hu's moment invariants. The code can be found in the appendix.

The next part of the code is for classification mode. All parallel neurons are implemented in VHDL:

```
---------------------------------------------------------------
--                          COMPUTATION FOR NEURON 1                    --
---------------------------------------------------------------
          addr1_1<=X"000";
          addr1_2<=X"001";
          addr1_3<=X"002";
          addr1_4<=X"003";
          addr1_5<=X"004";
          addr1_6<=X"005";
          addr1_7<=X"006";
          im_type(0)<=X"007";


          mem_read1_1: rom
          port map (addr1_1,clk,mem1_1);
          mem_read1_2: rom
          port map (addr1_2,clk,mem1_2);
          mem_read1_3: rom
          port map (addr1_3,clk,mem1_3);
-------
-----
user_add8_6: mega_add
          port map
(clk,add8_27,add8_46,dummy_sub_nan,dummy_sub_overflow,distance(7),dummy_sub_underflow);

------------------------------------------------------------------
--                          CALCULATION END FOR NEURON 8
------------------------------------------------------------------
```

Finally the shortest distance has been found out in order to find the winning node and the corresponding class. The following part of the VHDL code performs that operation.

```
          process(clock,distance(0),distance(1),distance(2),distance(3),distance(4),distance(5),distance(6),dis
tance(7))
                    begin

                    if(clock'EVENT AND clock='1') then          --the calculations are based on clock cycle
                                        count2<=count2+1;
          -------
          --------

                                                  temp<=distance(0);
                                                  index <= 0;
                    end if;

                    if (count2 = 21) then
                              if (temp > distance(1))then
                                        temp<=distance(1);
                                        index <= 1;
                              end if;
                    end if;
----------------------
----------------------
                    if (count2 = 26) then
                              if (temp > distance(6))then
                                        temp<=distance(6);
                                        index <= 6;
                              end if;
                    end if;

                    if (count2 = 27) then
                              if (temp > distance(7))then
                                        temp<=distance(7);
                                        index <= 7;
                              end if;
                    end if;
                    end if;

          end process;

          mem_read_final: rom
          port map (im_type(index),clock,test1);
```

4.7 Discussion on MATLAB Coding of Training Algorithm

The training of the system has been implemented with MATLAB coding. A database of seven Hu's moments is created using different predefined images. More the 75 images have been used for training purpose. This data base has been called in the MATLAB program and the training procedure has performed. Finally the updated weight values are stored in a separate database. The updated weight values of the trained neurons have been used for the classification mode which has been implemented in VHDL. The Kohonen Neural network updates the weight for neuron and clustering algorithm gives identification of different classes.

4.8 Summary

This chapter mainly discussed about the algorithmic implementation of the project. As a proof of concept predefined images has been used in place of a real image. Probably the next phase of the project will overcome this and deal with real images. Few problems have been addressed during the VHDL implementation. The success of the concept has been proved in the result and discussion chapter.

Chapter 5: Discussion

5.1 Introduction

So far the thesis described different aspect of the project namely literature survey, theory, implementation etc. This chapter shows the result of the system. It also analysed whether the concept is pragmatic. Different pros and cons of the exploration and implementation are also reflected here.

5.2 Result & Timing Analysis

The algorithm has been implemented in synthesizable VHDL code and a timing analysis has been performed. The simulation results exhibit a very good system timing performance, and the results indicate that the concept design fulfils the requirement for real time object recognition. As the algorithm is divided in three parts, the thesis has also analysed the result separately.

5.2.1 Moment Calculation

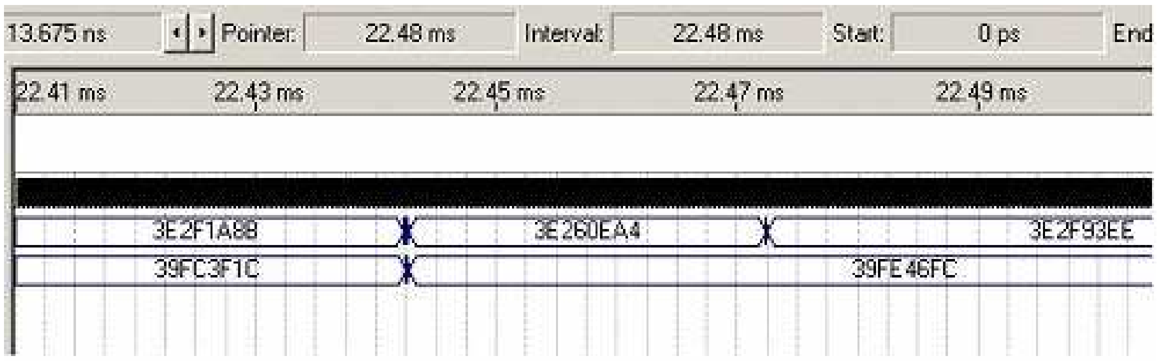


Figure 5.1: Timing analysis of Moment Calculation

The timing analysis of Hu's moment invariant calculation is shown in figure 5.1. The processing of input data consumes the most time in this part. For a 20x20 pixel input image it took an average of 22.47 milliseconds whereas the rest of the computation of Hu's invariant required approximately 1.5 milliseconds to complete (refer to table 5.1).

This is due to the number of sequential operations performed. If the input variables can be processed in parallel, the time response could be significantly improved. The floating point calculations, however, have been implemented efficiently as can be seen from the results.

5.2.2 Training of the System

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Basic image for class I

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Basic image for class II

Figure 5.2: Predefined test images for two different classes

As said earlier, the training of the system has been performed through a MATLAB programme. During the training phase, Hu's seven moment descriptors have been taken as the input vectors and calculated accordingly. In this project two types of pre-defined images (shown in figure 5.2) have been taken for the training purpose. Around 75 different images of these two classes have been prepared for the training purpose. It has been observed that the classification stage has an accuracy of 95% (with a classification of 20 unknown images of similar kind).

5.2.3 Classification Mode

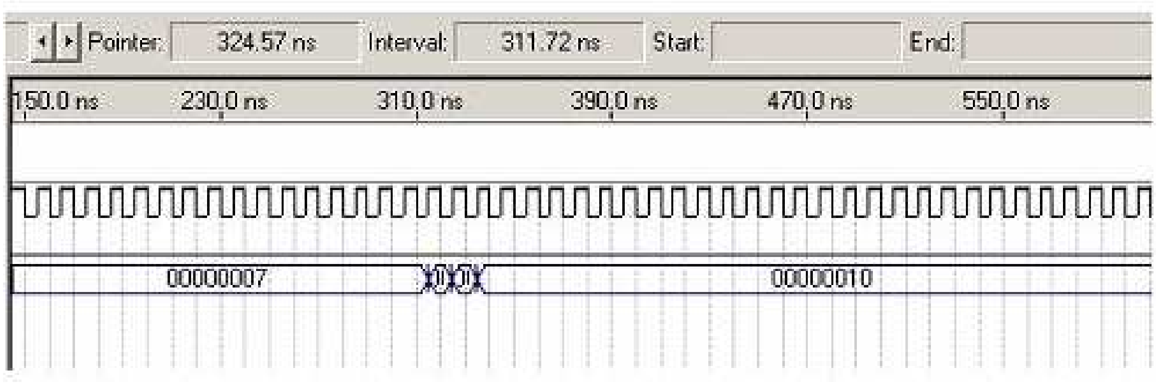


Figure 5.3: Timing analysis of Object Classification mode

The object classification mode indicates a good response time with parallel operation of trained neurons. For this application, eight neurons are designed to run in parallel, providing a response time of 310-360 nanoseconds, as shown in figure 5.3.

5.3 Analysis & Discussion for Real Time Object Classification

The timing analysis shows that the system requires an average of 24 milliseconds (average) for the detection of an unknown object. This indicates that even with the sequential computations in parts of the design, a recognition rate of 40 fps (frames per second) is achievable.

	Hu's Moment			Classification Mode
	i/p cal	other calculation	total	
Time	23 ms	1.5 ms	24.5 ms	310-360 ns

Table 5.1: Average time for different calculations

5.4 Summary

This chapter discussed about the result obtained during the experiment. It also discussed the exploration of the new concept. As a proof concept this project has shown a commendable result with a good future prospect with the given approach. A continuation of the similar work will be able to make a single board FPGA system for a real time image recognition classification with a very good time response.

Chapter 6: Conclusion and Future Work

6.1 Introduction

This chapter concludes the thesis and discusses about the possible future work that may be carried out. This thesis has addressed various aspects of the project. The introduction chapter provided an overview of the work and the motivation of the project. The following sections describe the theoretical aspects and the algorithm development. Finally the result and discussion completed the work.

6.2 Conclusion

In this thesis, we have demonstrated a possible solution of classifying objects in real-time using an FPGA solution. Object features are represented using Hu's moment invariant method, and an onboard Kohonen artificial neural network is used for initial clustering. This implementation uses the Altera Stratix FPGA device, which has the capability of performing complex mathematical operations. The estimated time to recognise an unknown object (20x20 pixels) is 24 milliseconds. We have shown that the classification of objects can be achieved using a single Altera Stratix board. The use of multiple FPGA boards working in parallel, however, can enhance system performance. At present, the input pixels to the system and the initial moment calculations are performed sequentially. As this is the most time consuming aspect of the process, overall system performance can be further improved by using a parallel implementation.

6.2.1 Why FPGA

The motivation of the project was to build an image recognition and classification system with a very good response time. A dedicated and customised hardware is always preferred in place of a software running in generalize computer. Obviously a stand alone and separate equipment for this purpose makes the whole system more robust and reliable, hence the idea of using reconfigurable hardware. The FPGAs offers a solution to the need for reconfigurable hardware. Normally FPGAs are more cost effective than designing a custom made ASIC. Also FPGAs can be reprogrammed to fix a bug or upgrade a system. Hence FPGAs are a natural selection of hardware designer.

6.2.2 Implementation of ANN on FPGA

Artificial Neural Networks are a powerful tool for machine intelligence. The ANN is based on the principle of human neuronal system. The main feature addressed by ANNs is the ability to operate in parallel. This feature of the ANN speeds up the system as compared to a traditional sequential computer. A true parallel system can never be realised in a general purpose sequential computer. The only solution is to use a neural-processor or design a dedicated parallel hardware. Hence the idea of use of a FPGA comes again. Hardware designer can easily accommodate a truly parallel hardware inside a FPGA.

6.2.3 Potential Pros & Cons

The project described has tried to explore the opportunities of image recognition system in FPGA board. During the experiment a few pros and cons have been found out.

Probably a FPGA solution is one of the best possible solutions to address the problem. FPGA can be programmed easily with high level hardware description language like VHLD, Verilog, System C *etc*. A block diagram design can also be useful. The FPGA operation can be simulated after the compilation of the program. It is easy to debug and test the system with simulation. A complete timing diagram gives the opportunity for analysis of the system.

However some problems exist. Apparently simple arithmetic operations might be complicated when implanted on FPGA. As an example, the use of floating point operation will make the FPGA system complex. In comparison with ASICs, FPGAs are generally slower. An interconnect delay sometimes plays a significant delay in overall system performance. So far the best available FPGAs have their system clocks rated in MHz where as a simple computer has system clocks running in the GHz range.

Now-a-days very advanced FPGAs are available in the market. But still numbers of logic elements are not sufficient enough to address certain problems. For example we have finally simulated the project in the Stratix II environment instead of available Stratix board. By optimising further, we may be able to accommodate the system on a single Stratix board. In the case of onboard online training which is proposed as future work, this limitation may play a vital role and good optimisation is required in order address this factor.

6.3 Future Work

The main aim of the project was to build a stand alone system for image recognition using a single FPGA board. But due time constraints some of the targetted features could not be implemented or explored during this project. Hence some future work has been proposed for the next stage of advancement.

6.3.1 Online Training

So far the *training* of the system has been performed using a MATLAB program. In future work, the onboard and online training of the Kohonen ANN will be addressed. The updated weight vectors of the neurons will be stored in the memory of the FPGA board, and will be made accessible during the classification mode.

6.3.2 System Integration with Camera Interface

The camera interface is currently simulated by uploading image data directly to the FPGA board. This has been successful as a proof of concept. But in order to deploy the system in a real working environment a camera interface is required in order to enable it to grab and classify images in real time. The Stratix FPGA development board has a number of interface options through high speed A/D converter or a RS 232 serial interface.

6.4 Summary

The current constraint of the system is the capacity of the FPGA board. By using a larger FPGA, the number of neurons can be increased and hence more parallel operations can be performed. Further code optimisation is also possible and some of the implemented sequential operations can be replaced by parallel hardware design. The use of multiple boards for a single system will also enhance the system performance. We have demonstrated that it is possible to build a full imaging system on a single FPGA board. As a proof of concept, the VHDL simulation has been conducted using the Altera Quartus II software environment. Analysis from timing diagrams has shown promising results..

Reference and Bibliography

[1] M-K. Hu, “Visual pattern recognition by moment invariants”, IRE Trans. on Information Theory, vol 8, pp. 179-187, 1962.

[2] S. Hirai, M. Zakouji and T.Tsuboi, “Implementing Image Processing Algorithms on FPGA-based Realtime Vision System”, Proc. 11th Synthesis and System Integration of Mixed Information Technologies (SASIMI 2003), pp. 378-385, Hiroshima, April, 2003.

[3] P.C. Arribas and F.M-H. Maciá, “FPGA implementation of Santos-Victor optical flow algorithm for real time image processing: an useful attempt”, Proceedings of SPIE's International Symposium on Microtechnologies for the New Millennium 2003 - Conference on VLSI Circuits and Systems, pp. 23-32, Canary Islands, Spain, May 19-21, 2003.

[4] P.J. Sanz, R. Marin and J.S. Sanchez, "Including efficient object recognition capabilities in online robots: from a statistical to a Neural-network classifier," IEEE Transactions on Systems, Man and Cybernetics, Part C, vol.35, no.1, pp. 87- 96, February, 2005.

[5] T. Kohonen, “Automatic formation of topological maps of patterns in a self-organizing system”, Proceedings of 2nd Scandinavian Conference on Image Analysis, Espoo, Finland, pp. 214--220, 1981.

[6] S. Neema, J. Scott, T. Bapty, "Real time reconfigurable image recognition system", Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference, 2001. Volume 1, 21-23 May 2001, pp. 350 - 355 vol.1

[7] F. Mohd-Yasin, A.L. Tan, M.I. Reaz, "The FPGA prototyping of iris recognition for biometric identification employing neural network", Proceedings of The 16th International Conference on Microelectronics, ICM 2004. 6-8 Dec. 2004, pp. 458 – 461

[8] J. Jean, Liang Xiejun, B. Drozd, K. Tomko, "Accelerating an IR automatic target recognition application with FPGAs", Proceedings of Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '99. 21-23 April 1999, pp. 290 – 291

[9] N. Aibe, M. Yasunaga, I. Yoshihara, J.H. Kim, "A probabilistic neural network hardware system using a learning-parameter parallel architecture", Proceedings of the 2002 International Joint Conference on Neural Networks, IJCNN '02., Volume 3, 12-17 May 2002, pp. 2270 – 2275

[10] Yuk Ying Chung, Man To Wong, Neil W. Bergmann, "High speed neural network based classifier for Real-time application", Proceedings of ICSP '98, pp. 506 – 509

[11] Dr. R. Saatchi, Handout given in classroom & laboratory session (16-7206-00S Artificial Intelligence), February - April 2006, Sheffield Hallam University

[12] http://en.wikipedia.org/wiki/IEEE_754 last visited on 14.09.2006

[13] Hsiao-Fen Fu, IEEE 754 Floating Point, CSCI 313 Tutorial, http://ftp.csci.csusb.edu/schubert/tutorials/csci313/w04/HsiaoFenFu_Tutorial_IEEE%20754%20Floating%20Point.pdf last visited on 14.09.2006

[14] http://en.wikipedia.org/wiki/FPGA last visited on 14.09.2006

[15] http://en.wikipedia.org/wiki/K-means_clustering_algorithm last visited on 14.09.2006

[16] http://en.wikipedia.org/wiki/Image_moments last visited on 14.09.2006

[17] A. Ashbrook and N. A. Thacker, Tutorial: Algorithms for 2-Dimensional Object Recognition, Tina Memo No. 1996-003, Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester

[18] P.J. Sanz, R. Marin and J.S. Sanchez, "Including efficient object recognition capabilities in online robots: from a statistical to a Neural-network classifier," IEEE Transactions on Systems, Man and Cybernetics, Part C, vol.35, no.1, pp. 87- 96, February, 2005.

[19] http://www.imappl.org/~cgloster/rare/vhdl/ last visited on 14.09.2006

[20] Timothy Masters, "Signal and Image Processing with Neural Networks, A C++ Sourcebook", John Wiley & Sons, Inc., New York, 1994

[21] William Kleitz, Digital Electronics with VHDL quartus II Version, Prentice Hall, 2006, New Jersey

Appendix

VHDL code

```vhdl
----------------------------------------------------------------------
--Appendix

--Appendix A: VHDL Code
--              Author: Deepayan Bhowmik
--                  deepayan.bhowmik@yahoo.com
--                  MSc. Electronics & IT 2005-06
--                  Sheffield Hallam University
--              Copyrighted
----------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;


entity user is

    port(   clk :in std_logic;
            out_test: out STD_LOGIC;
            test1 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)


        );


end user;

architecture behaviour of user is

--type moment_type is array (1 to 25) of bit_vector(15 downto 0);

type pic_type is array (0 to 15) of std_logic_vector(0 to 15);

signal count, en_count : integer range 0 to 1000;
signal en_signal : std_logic;
signal x1,x2,x3 : integer range 0 to 100000;
signal y1,y2,y3 : integer range 0 to 100000;
signal m11,m12,m13,m14 : integer range 0 to 100000;
signal m21,m22,m23 : integer range 0 to 100000;
signal m31,m32 : integer range 0 to 100000;
signal m41 : integer range 0 to 100000;
signal x,y : integer range 0 to 100;
signal pic1: pic_type;
signal count2,count3 : natural range 0 to 1000;


signal moment11,moment12,moment13,moment14 : STD_LOGIC_VECTOR (31 DO
WNTO 0);
signal moment21,moment22,moment23 : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal moment31,moment32 : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal moment41 : STD_LOGIC_VECTOR (31 DOWNTO 0);

signal xmean,ymean : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal mu11,mu31,mu13,mu22,mu41,mu23,mu32,mu14 : STD_LOGIC_VECTOR (3
1 DOWNTO 0);
signal eta31,eta13,eta22,eta41,eta23,eta32,eta14 : STD_LOGIC_VECTOR
(31 DOWNTO 0);
signal hu1,hu2,hu3,hu4,hu5,hu6,hu7 : STD_LOGIC_VECTOR (31 DOWNTO 0);

signal mu_mult31,mu_mult13,mu_mult22,xmean_sq,ymean_sq : STD_LOGIC_V
ECTOR (31 DOWNTO 0);
signal const_2,const_3,const_4,mu_mult41_11,mu_mult41_12,mu_add41,mu
```

```
_mult41_21,mu_mult41_22,mu_sub41 : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal mu_mult23_11,mu_mult23_12,mu_add23,mu_mult23_21,mu_mult23_32,
mu_sub23_1,mu_sub23_2 : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal mu_mult32_12,mu_add32,mu_mult32_21,mu_mult32_31,mu_mult32_32,
mu_sub32_1,mu_sub32_2 : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal mu_mult14_11,mu_mult14_12,mu_add14,mu_mult14_21,mu_mult14_22,
mu_sub14 : STD_LOGIC_VECTOR (31 DOWNTO 0);

signal eta_sq : STD_LOGIC_VECTOR (31 DOWNTO 0);

signal hu_sub2,hu_mult2_1,hu_mult2_2,hu_mult2_3,hu_add2 : STD_LOGIC_
VECTOR (31 DOWNTO 0);
signal hu_add4_1,hu_add4_2,hu_mult4_1,hu_mult4_2 : STD_LOGIC_VECTOR
(31 DOWNTO 0);
signal hu_sub3_1,hu_sub3_2,hu_mult3_1,hu_mult3_2,hu_mult3_3,hu_mult3
_4 : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal hu_mult5_1,hu_mult5_2,hu_mult5_3,hu_mult5_4,hu_mult5_5,hu_mul
t5_6,hu_sub5_1,hu_sub5_2 : STD_LOGIC_VECTOR (31 DOWNTO 0);
signal hu_sub6_1,hu_mult6_1,hu_mult6_2,hu_mult6_3,hu_mult6_4: STD_LO
GIC_VECTOR (31 DOWNTO 0);
signal hu_mult7_1,hu_mult7_2,hu_mult7_3,hu_mult7_4,hu_mult7_5,hu_sub
7_1,hu_sub7_2: STD_LOGIC_VECTOR (31 DOWNTO 0);


signal dummy_add_nan,dummy_add_overflow,dummy_add_underflow : STD_LO
GIC ;
signal dummy_sub_nan,dummy_sub_overflow,dummy_sub_underflow : STD_LO
GIC ;
signal dummy_mult_nan,dummy_mult_overflow,dummy_mult_underflow : STD
_LOGIC ;

signal EnR,EnL,reset,EnN: STD_LOGIC;

-----------------------------------------------------------------
-- Signals for user detection mode
-----------------------------------------------------------------

type manhatan_distance is array (0 to 7) of std_logic_vector(0 to 31
);
type image_type is array (0 to 7) of std_logic_vector(0 to 11);

signal distance: manhatan_distance;
signal im_type: image_type;

signal add1,add2,add3,add4,add5,add6 : STD_LOGIC_VECTOR(11 downto 0)
;
signal input1,input2,input3,input4,input5,input6,input7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal mask,temp : STD_LOGIC_VECTOR(31 DOWNTO 0);
signal index,check: integer range 0 to 7;



--Signal for neuron 1
signal addr1_1,addr1_2,addr1_3,addr1_4,addr1_5,addr1_6,addr1_7 : STD
_LOGIC_VECTOR(11 DOWNTO 0);
signal mem1_1,mem1_2,mem1_3,mem1_4,mem1_5,mem1_6,mem1_7,mem1_8 : STD
_LOGIC_VECTOR(31 DOWNTO 0);
signal sub1_1,sub1_2,sub1_3,sub1_4,sub1_5,sub1_6,sub1_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal abs1_1,abs1_2,abs1_3,abs1_4,abs1_5,abs1_6,abs1_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
```

```
signal add1_12,add1_34,add1_56,add1_27,add1_46 : STD_LOGIC_VECTOR(31
 DOWNTO 0);

--Signal for neuron 2
signal addr2_1,addr2_2,addr2_3,addr2_4,addr2_5,addr2_6,addr2_7 : STD
_LOGIC_VECTOR(11 DOWNTO 0);
signal mem2_1,mem2_2,mem2_3,mem2_4,mem2_5,mem2_6,mem2_7,mem2_8 : STD
_LOGIC_VECTOR(31 DOWNTO 0);
signal sub2_1,sub2_2,sub2_3,sub2_4,sub2_5,sub2_6,sub2_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal abs2_1,abs2_2,abs2_3,abs2_4,abs2_5,abs2_6,abs2_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal add2_12,add2_34,add2_56,add2_27,add2_46 : STD_LOGIC_VECTOR(31
 DOWNTO 0);

--Signal for neuron 3
signal addr3_1,addr3_2,addr3_3,addr3_4,addr3_5,addr3_6,addr3_7 : STD
_LOGIC_VECTOR(11 DOWNTO 0);
signal mem3_1,mem3_2,mem3_3,mem3_4,mem3_5,mem3_6,mem3_7,mem3_8 : STD
_LOGIC_VECTOR(31 DOWNTO 0);
signal sub3_1,sub3_2,sub3_3,sub3_4,sub3_5,sub3_6,sub3_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal abs3_1,abs3_2,abs3_3,abs3_4,abs3_5,abs3_6,abs3_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal add3_12,add3_34,add3_56,add3_27,add3_46 : STD_LOGIC_VECTOR(31
 DOWNTO 0);

--Signal for neuron 4
signal addr4_1,addr4_2,addr4_3,addr4_4,addr4_5,addr4_6,addr4_7 : STD
_LOGIC_VECTOR(11 DOWNTO 0);
signal mem4_1,mem4_2,mem4_3,mem4_4,mem4_5,mem4_6,mem4_7,mem4_8 : STD
_LOGIC_VECTOR(31 DOWNTO 0);
signal sub4_1,sub4_2,sub4_3,sub4_4,sub4_5,sub4_6,sub4_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal abs4_1,abs4_2,abs4_3,abs4_4,abs4_5,abs4_6,abs4_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal add4_12,add4_34,add4_56,add4_27,add4_46 : STD_LOGIC_VECTOR(31
 DOWNTO 0);

--Signal for neuron 5
signal addr5_1,addr5_2,addr5_3,addr5_4,addr5_5,addr5_6,addr5_7 : STD
_LOGIC_VECTOR(11 DOWNTO 0);
signal mem5_1,mem5_2,mem5_3,mem5_4,mem5_5,mem5_6,mem5_7,mem5_8 : STD
_LOGIC_VECTOR(31 DOWNTO 0);
signal sub5_1,sub5_2,sub5_3,sub5_4,sub5_5,sub5_6,sub5_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal abs5_1,abs5_2,abs5_3,abs5_4,abs5_5,abs5_6,abs5_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal add5_12,add5_34,add5_56,add5_27,add5_46 : STD_LOGIC_VECTOR(31
 DOWNTO 0);

--Signal for neuron 6
signal addr6_1,addr6_2,addr6_3,addr6_4,addr6_5,addr6_6,addr6_7 : STD
_LOGIC_VECTOR(11 DOWNTO 0);
signal mem6_1,mem6_2,mem6_3,mem6_4,mem6_5,mem6_6,mem6_7,mem6_8 : STD
_LOGIC_VECTOR(31 DOWNTO 0);
signal sub6_1,sub6_2,sub6_3,sub6_4,sub6_5,sub6_6,sub6_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal abs6_1,abs6_2,abs6_3,abs6_4,abs6_5,abs6_6,abs6_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal add6_12,add6_34,add6_56,add6_27,add6_46 : STD_LOGIC_VECTOR(31
 DOWNTO 0);
```

```

--Signal for neuron 7
signal addr7_1,addr7_2,addr7_3,addr7_4,addr7_5,addr7_6,addr7_7 : STD
_LOGIC_VECTOR(11 DOWNTO 0);
signal mem7_1,mem7_2,mem7_3,mem7_4,mem7_5,mem7_6,mem7_7,mem7_8 : STD
_LOGIC_VECTOR(31 DOWNTO 0);
signal sub7_1,sub7_2,sub7_3,sub7_4,sub7_5,sub7_6,sub7_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal abs7_1,abs7_2,abs7_3,abs7_4,abs7_5,abs7_6,abs7_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal add7_12,add7_34,add7_56,add7_27,add7_46 : STD_LOGIC_VECTOR(31
 DOWNTO 0);

--Signal for neuron 8
signal addr8_1,addr8_2,addr8_3,addr8_4,addr8_5,addr8_6,addr8_7 : STD
_LOGIC_VECTOR(11 DOWNTO 0);
signal mem8_1,mem8_2,mem8_3,mem8_4,mem8_5,mem8_6,mem8_7,mem8_8 : STD
_LOGIC_VECTOR(31 DOWNTO 0);
signal sub8_1,sub8_2,sub8_3,sub8_4,sub8_5,sub8_6,sub8_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal abs8_1,abs8_2,abs8_3,abs8_4,abs8_5,abs8_6,abs8_7 : STD_LOGIC_
VECTOR(31 DOWNTO 0);
signal add8_12,add8_34,add8_56,add8_27,add8_46 : STD_LOGIC_VECTOR(31
 DOWNTO 0);




----------------------------------------------------------------
-- END ---        Signals for user detection mode
----------------------------------------------------------------







component rom
        PORT
        (
            address     : IN STD_LOGIC_VECTOR (11 DOWNTO 0);
            clock       : IN STD_LOGIC ;
            q       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
end component;



component FpDivCore_16
    PORT(clk          : IN std_logic;
              reset    : IN std_logic;
              EnR              : IN std_logic;
              EnL              : IN std_logic;
              Op_A             : IN std_logic_vector(31 DOWNTO 0);
              Op_B             : IN std_logic_vector(31 DOWNTO 0);
              Op_Q             : OUT std_logic_vector(31 DOWNTO 0);
              EnN              : OUT std_logic);
end component;


COMPONENT mega_add
```

```
    PORT
        (
            clock        : IN STD_LOGIC ;
            dataa        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            datab        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            nan      : OUT STD_LOGIC ;
            overflow         : OUT STD_LOGIC ;
            result       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
            underflow        : OUT STD_LOGIC
        );

END COMPONENT;



COMPONENT mega_sub
    PORT
    (
        clock        : IN STD_LOGIC ;
        dataa        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        datab        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        nan      : OUT STD_LOGIC ;
        overflow         : OUT STD_LOGIC ;
        result       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        underflow        : OUT STD_LOGIC
    );

END COMPONENT;

COMPONENT mega_mult
    PORT
    (
        clock        : IN STD_LOGIC ;
        dataa        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        datab        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        nan      : OUT STD_LOGIC ;
        overflow         : OUT STD_LOGIC ;
        result       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        underflow        : OUT STD_LOGIC
    );

END COMPONENT;


COMPONENT int2fp
       PORT (input : integer range 0 to 100000;

             fp_op : OUT STD_LOGIC_VECTOR(31 downto 0));
END COMPONENT;


--signal moment_parameter: moment_type;

begin

        pic1(0) <=      "0001111111111111";
        pic1(1) <=      "0011111111111111";
        pic1(2) <=      "0010111111111111";
        pic1(3) <=      "0011111111111111";
        pic1(4) <=      "0011111111111111";
        pic1(5) <=      "0011111111111111";
        pic1(6) <=      "0011011111111111";
```

```vhdl
        pic1(7) <=      "0011111111111111";
        pic1(8) <=      "0111111111111111";
        pic1(9) <=      "0010111111111111";
        pic1(10) <=     "0011111111111111";
        pic1(11) <=     "0011101111111111";
        pic1(12) <=     "0011111111111111";
        pic1(13) <=     "0011111101111111";
        pic1(14) <=     "0011111111111111";
        pic1(15) <=     "0111111111111111";

        const_2<=X"40000000";
        const_3<=X"40400000";
        const_4<=X"40800000";

--x, y value starts from 0,0
--  x<=1;
--  y<=1;

    process(clk,en_signal,count,en_count,
            x1,x2,x3,y1,y2,y3,
            m11,m12,m13,m14,
            m21,m22,m23,
            m31,m32,
            m41)
    begin
        if(clk'EVENT AND clk='1') then  --the calculations are based on clock cycle
            en_count<=en_count+1;

            if(y<16)then
            if(x<16)then

            if(pic1(y)(x)='1' AND en_count=1)then
                en_signal<='1';
                count<=count+1;           --1 clock cycle is simulated in 10ns

--Calculations start here || Pipeline architecture used here

                y1<=y;
                x1<=x;


--First phase data update, assumed clock cycle 4

                if(count=1)then
                    y2<=y1*y1;
                    x2<=x1*x1;

                    m11<=m11+1;
                    m12<=m12+y1;
                    m21<=m21+x1;
                    m22<=m22+x1*y1;

                end if;

--2nd phase data update, assumed clock cycle 8

                if(count=4)then
                    y3<=y2*y1;
                    x3<=x2*x1;

                    m13<=m13+y2;
                    m23<=m23+x1*y2;
                    m31<=m31+x2;
                    m32<=m32+x2*y1;

                end if;

--3rd phase data update, assumed clock cycle 8

                if(count=7)then
                    m14<=m14+y3;
                    m41<=m41+x3;

                end if;

                if(count=10 AND y<16)then
                    count<=0;

                    en_signal<='0';
                    en_count<=0;

                    x<=x+1;

                    if(x>15)then
                        x<=0;
                        y<=y+1;
                    end if;

                end if;

            end if;
            end if;
            end if;

            if(en_count=2 AND en_signal='0' AND y<16) then
                x<=x+1;
                en_count<=0;

                if(x>15)then
                    x<=0;
                    y<=y+1;
                end if;

            end if;

        end if;

    end process;

--------------------------------------------------------------------
-------------
--              Integer to Floating point Conversion
--------------------------------------------------------------------
-------------

    conv_fp_1: int2fp
    port map (m11,moment11);

    conv_fp_2: int2fp
    port map (m12,moment12);

    conv_fp_3: int2fp
    port map (m13,moment13);

    conv_fp_4: int2fp
    port map (m14,moment14);

    conv_fp_5: int2fp
    port map (m21,moment21);

    conv_fp_6: int2fp
    port map (m22,moment22);

    conv_fp_7: int2fp
    port map (m23,moment23);

    conv_fp_8: int2fp
    port map (m31,moment31);

    conv_fp_9: int2fp
    port map (m32,moment32);

    conv_fp_10: int2fp
    port map (m41,moment41);



----------------------------------------------------
--          Calculation of Mu
----------------------------------------------------

-- Calculation of xmean & ymean
    EnR<='1';
    EnL<='1';
    reset<='0';

    fp_div_xmean: FpDivCore_16
    port map(clk,reset,EnR,EnL,moment21,moment11,xmean,EnN);

    fp_div_ymean: FpDivCore_16
    port map(clk,reset,EnR,EnL,moment12,moment11,ymean,EnN);


--Calculation of mu11
    mu11<=moment11;

--Calculation of mu31
    mult_mu31: mega_mult
    port map (clk,xmean,moment21,dummy_mult_nan,dummy_mult_overflow,
mu_mult31,dummy_mult_underflow);
    sub_mu31: mega_sub
    port map (clk,moment31,mu_mult31,dummy_sub_nan,dummy_sub_overflo
w,mu31,dummy_sub_underflow);

--Calculation of mu13
    mult_mu13: mega_mult
    port map (clk,ymean,moment12,dummy_mult_nan,dummy_mult_overflow,
mu_mult13,dummy_mult_underflow);
    sub_mu13: mega_sub
    port map (clk,moment13,mu_mult13,dummy_sub_nan,dummy_sub_overflo
w,mu13,dummy_sub_underflow);

--Calculation of mu22
    mult_mu22: mega_mult
```

```vhdl
    port map (clk,ymean,moment21,dummy_mult_nan,dummy_mult_overflow,mu_mult22,dummy_mult_underflow);
    sub_mu22: mega_sub
    port map (clk,moment22,mu_mult22,dummy_sub_nan,dummy_sub_overflow,mu22,dummy_sub_underflow);

--Calculation of mu41
    sq_xmean: mega_mult
    port map (clk,xmean,xmean,dummy_mult_nan,dummy_mult_overflow,xmean_sq,dummy_mult_underflow);
    mult_mu41_21: mega_mult
    port map (clk,const_2,moment21,dummy_mult_nan,dummy_mult_overflow,mu_mult41_21,dummy_mult_underflow);
    mult_mu41_22: mega_mult
    port map (clk,xmean_sq,mu_mult41_21,dummy_mult_nan,dummy_mult_overflow,mu_mult41_22,dummy_mult_underflow);
    mult_mu41_11: mega_mult
    port map (clk,const_3,moment31,dummy_mult_nan,dummy_mult_overflow,mu_mult41_11,dummy_mult_underflow);
    mult_mu41_12: mega_mult
    port map (clk,mu_mult41_11,xmean,dummy_mult_nan,dummy_mult_overflow,mu_mult41_12,dummy_mult_underflow);
    sub_mu41: mega_sub
    port map (clk,moment41,mu_mult41_12,dummy_sub_nan,dummy_sub_overflow,mu_sub41,dummy_sub_underflow);
    add_mu41: mega_add
    port map (clk,mu_sub41,mu_mult41_22,dummy_add_nan,dummy_add_overflow,mu41,dummy_add_underflow);


--Calculation of mu23
    sq_ymean: mega_mult
    port map (clk,ymean,ymean,dummy_mult_nan,dummy_mult_overflow,ymean_sq,dummy_mult_underflow);
--mu_mult41_21 came from previous calculation
    mult_mu23_32: mega_mult
    port map (clk,ymean_sq,mu_mult41_21,dummy_mult_nan,dummy_mult_overflow,mu_mult23_32,dummy_mult_underflow);
    mult_mu23_21: mega_mult
    port map (clk,xmean,moment13,dummy_mult_nan,dummy_mult_overflow,mu_mult23_21,dummy_mult_underflow);
    mult_mu23_11: mega_mult
    port map (clk,const_2,moment22,dummy_mult_nan,dummy_mult_overflow,mu_mult23_11,dummy_mult_underflow);
    mult_mu23_12: mega_mult
    port map (clk,mu_mult23_11,ymean,dummy_mult_nan,dummy_mult_overflow,mu_mult23_12,dummy_mult_underflow);
    sub_mu23_1: mega_sub
    port map (clk,moment23,mu_mult23_12,dummy_sub_nan,dummy_sub_overflow,mu_sub23_1,dummy_sub_underflow);
    sub_mu23_2: mega_sub
    port map (clk,mu_sub23_1,mu_mult23_21,dummy_sub_nan,dummy_sub_overflow,mu_sub23_2,dummy_sub_underflow);
    add_mu23: mega_add
    port map (clk,mu_sub23_2,mu_mult23_32,dummy_add_nan,dummy_add_overflow,mu23,dummy_add_underflow);


--Calculation of mu32
    mult_mu32_31: mega_mult
    port map (clk,const_2,moment12,dummy_mult_nan,dummy_mult_overflow,mu_mult32_31,dummy_mult_underflow);
    mult_mu32_32: mega_mult
    port map (clk,xmean_sq,mu_mult32_31,dummy_mult_nan,dummy_mult_overflow,mu_mult32_32,dummy_mult_underflow);
    mult_mu32_21: mega_mult
    port map (clk,ymean,moment31,dummy_mult_nan,dummy_mult_overflow,mu_mult32_21,dummy_mult_underflow);
--Calculation from last parameters
    mult_mu32_12: mega_mult
    port map (clk,mu_mult23_11,xmean,dummy_mult_nan,dummy_mult_overflow,mu_mult32_12,dummy_mult_underflow);
    sub_mu32_1: mega_sub
    port map (clk,moment32,mu_mult32_12,dummy_sub_nan,dummy_sub_overflow,mu_sub32_1,dummy_sub_underflow);
    sub_mu32_2: mega_sub
    port map (clk,mu_sub32_1,mu_mult32_21,dummy_sub_nan,dummy_sub_overflow,mu_sub32_2,dummy_sub_underflow);
    add_mu32: mega_add
    port map (clk,mu_sub32_2,mu_mult32_32,dummy_add_nan,dummy_add_overflow,mu32,dummy_add_underflow);


--Calculation of mu14
    mult_mu14_21: mega_mult
    port map (clk,const_2,moment12,dummy_mult_nan,dummy_mult_overflow,mu_mult14_21,dummy_mult_underflow);
    mult_mu14_22: mega_mult
    port map (clk,ymean_sq,mu_mult14_21,dummy_mult_nan,dummy_mult_overflow,mu_mult14_22,dummy_mult_underflow);
    mult_mu14_11: mega_mult
    port map (clk,const_3,moment13,dummy_mult_nan,dummy_mult_overflow,mu_mult14_11,dummy_mult_underflow);
    mult_mu14_12: mega_mult
    port map (clk,mu_mult14_11,ymean,dummy_mult_nan,dummy_mult_overflow,mu_mult14_12,dummy_mult_underflow);
    sub_mu14: mega_sub
    port map (clk,moment14,mu_mult14_12,dummy_sub_nan,dummy_sub_overflow,mu_sub14,dummy_sub_underflow);
    add_mu14: mega_add
    port map (clk,mu_sub14,mu_mult14_22,dummy_add_nan,dummy_add_overflow,mu14,dummy_add_underflow);



---------------------------------------------------------------------------------
--                   Calculation of eta
---------------------------------------------------------------------------------

    eta_mult_1: mega_mult
    port map (clk,mu11,mu11,dummy_mult_nan,dummy_mult_overflow,eta_sq,dummy_mult_underflow);

--Calculation of eta31
    fp_div_eta31: FpDivCore_16
    port map(clk,reset,EnR,EnL,mu31,eta_sq,eta31,EnN);

--Calculation of eta13
    fp_div_eta13: FpDivCore_16
    port map(clk,reset,EnR,EnL,mu13,eta_sq,eta13,EnN);

--Calculation of eta22
```

```
    fp_div_eta22: FpDivCore_16
    port map(clk,reset,EnR,EnL,mu22,eta_sq,eta22,EnN);

--Calculation of eta41
    fp_div_eta41: FpDivCore_16
    port map(clk,reset,EnR,EnL,mu41,eta_sq,eta41,EnN);

--Calculation of eta23
    fp_div_eta23: FpDivCore_16
    port map(clk,reset,EnR,EnL,mu23,eta_sq,eta23,EnN);

--Calculation of eta32
    fp_div_eta32: FpDivCore_16
    port map(clk,reset,EnR,EnL,mu32,eta_sq,eta32,EnN);

--Calculation of eta14
    fp_div_eta14: FpDivCore_16
    port map(clk,reset,EnR,EnL,mu14,eta_sq,eta14,EnN);



---------------------------------------------------------------------------------------
--                   Calculation of Hu's invarient
---------------------------------------------------------------------------------------

--Calculation of hu1
    add_hu1: mega_add
    port map (clk,eta31,eta13,dummy_add_nan,dummy_add_overflow,hu1,dummy_add_underflow);

--Calculation of hu2
    sub_hu2: mega_sub
    port map (clk,eta31,eta13,dummy_sub_nan,dummy_sub_overflow,hu_sub2,dummy_sub_underflow);
    mult_hu2_1: mega_mult
    port map (clk,hu_sub2,hu_sub2,dummy_mult_nan,dummy_mult_overflow,hu_mult2_1,dummy_mult_underflow);
    mult_hu2_2: mega_mult
    port map (clk,eta22,eta22,dummy_mult_nan,dummy_mult_overflow,hu_mult2_2,dummy_mult_underflow);
    mult_hu2_3: mega_mult
    port map (clk,const_4,hu_mult2_2,dummy_mult_nan,dummy_mult_overflow,hu_mult2_3,dummy_mult_underflow);
    add_hu2: mega_add
    port map (clk,hu_mult2_1,hu_mult2_3,dummy_add_nan,dummy_add_overflow,hu2,dummy_add_underflow);

--Calculation of hu3
    mult_hu3_1: mega_mult
    port map (clk,const_3,eta23,dummy_mult_nan,dummy_mult_overflow,hu_mult3_1,dummy_mult_underflow);
    sub_hu3_1: mega_sub
    port map (clk,eta41,hu_mult3_1,dummy_sub_nan,dummy_sub_overflow,hu_sub3_1,dummy_sub_underflow);
    mult_hu3_2: mega_mult
    port map (clk,hu_sub3_1,hu_sub3_1,dummy_mult_nan,dummy_mult_overflow,hu_mult3_2,dummy_mult_underflow);

    mult_hu3_3: mega_mult
    port map (clk,const_3,eta32,dummy_mult_nan,dummy_mult_overflow,h
u_mult3_3,dummy_mult_underflow);
    sub_hu3_2: mega_sub
    port map (clk,hu_mult3_3,eta14,dummy_sub_nan,dummy_sub_overflow,hu_sub3_2,dummy_sub_underflow);
    mult_hu3_4: mega_mult
    port map (clk,hu_sub3_2,hu_sub3_2,dummy_mult_nan,dummy_mult_overflow,hu_mult3_4,dummy_mult_underflow);

    add_hu3: mega_add
    port map (clk,hu_mult3_2,hu_mult3_4,dummy_add_nan,dummy_add_overflow,hu3,dummy_add_underflow);


--Calculation of hu4
    add_hu4_1: mega_add
    port map (clk,eta41,eta23,dummy_add_nan,dummy_add_overflow,hu_add4_1,dummy_add_underflow);
    mult_hu4_1: mega_mult
    port map (clk,hu_add4_1,hu_add4_1,dummy_mult_nan,dummy_mult_overflow,hu_mult4_1,dummy_mult_underflow);
    add_hu4_2: mega_add
    port map (clk,eta32,eta14,dummy_add_nan,dummy_add_overflow,hu_add4_2,dummy_add_underflow);
    mult_hu4_2: mega_mult
    port map (clk,hu_add4_2,hu_add4_2,dummy_mult_nan,dummy_mult_overflow,hu_mult4_2,dummy_mult_underflow);
    add_hu4_3: mega_add
    port map (clk,hu_mult4_1,hu_mult4_2,dummy_add_nan,dummy_add_overflow,hu4,dummy_add_underflow);


--Calculation of hu5
    mult_hu5_1: mega_mult
    port map (clk,hu_sub3_1,hu_add4_1,dummy_mult_nan,dummy_mult_overflow,hu_mult5_1,dummy_mult_underflow);

    mult_hu5_3: mega_mult
    port map (clk,const_3,hu_mult4_2,dummy_mult_nan,dummy_mult_overflow,hu_mult5_3,dummy_mult_underflow);
    sub_hu5_1: mega_sub
    port map (clk,hu_mult4_1,hu_mult5_3,dummy_sub_nan,dummy_sub_overflow,hu_mult5_2,dummy_sub_underflow);

    mult_hu5_2: mega_mult
    port map (clk,hu_mult5_1,hu_mult5_2,dummy_mult_nan,dummy_mult_overflow,hu_sub5_1,dummy_mult_underflow);

    mult_hu5_4: mega_mult
    port map (clk,hu_sub3_2,hu_add4_2,dummy_mult_nan,dummy_mult_overflow,hu_mult5_4,dummy_mult_underflow);
    mult_hu5_5: mega_mult
    port map (clk,const_3,hu_mult4_1,dummy_mult_nan,dummy_mult_overflow,hu_mult5_5,dummy_mult_underflow);
    sub_hu5_2: mega_sub
    port map (clk,hu_mult5_5,hu_mult4_2,dummy_sub_nan,dummy_sub_overflow,hu_sub5_2,dummy_sub_underflow);
    mult_hu5_6: mega_mult
    port map (clk,hu_mult5_4,hu_sub5_2,dummy_mult_nan,dummy_mult_overflow,hu_mult5_6,dummy_mult_underflow);

    add_hu5: mega_add
    port map (clk,hu_sub5_1,hu_mult5_6,dummy_add_nan,dummy_add_overf
low,hu5,dummy_add_underflow);



--Calculation of hu6
    sub_hu6_1: mega_sub
    port map (clk,hu_mult4_1,hu_mult4_2,dummy_sub_nan,dummy_sub_overflow,hu_sub6_1,dummy_sub_underflow);
    mult_hu6_1: mega_mult
    port map (clk,hu_sub2,hu_sub6_1,dummy_mult_nan,dummy_mult_overflow,hu_mult6_1,dummy_mult_underflow);

    mult_hu6_2: mega_mult
    port map (clk,hu_add4_1,hu_add4_2,dummy_mult_nan,dummy_mult_overflow,hu_mult6_2,dummy_mult_underflow);
    mult_hu6_3: mega_mult
    port map (clk,const_4,eta22,dummy_mult_nan,dummy_mult_overflow,hu_mult6_3,dummy_mult_underflow);
    mult_hu6_4: mega_mult
    port map (clk,hu_mult6_2,hu_mult6_3,dummy_mult_nan,dummy_mult_overflow,hu_mult6_4,dummy_mult_underflow);

    add_hu6: mega_add
    port map (clk,hu_mult6_1,hu_mult6_4,dummy_add_nan,dummy_add_overflow,hu6,dummy_add_underflow);


--Calculation of hu7
    mult_hu7_1: mega_mult
    port map (clk,hu_sub3_2,hu_add4_1,dummy_mult_nan,dummy_mult_overflow,hu_mult7_1,dummy_mult_underflow);
    sub_hu7_1: mega_sub
    port map (clk,hu_mult4_1,hu_mult5_3,dummy_sub_nan,dummy_sub_overflow,hu_sub7_1,dummy_sub_underflow);
    mult_hu7_2: mega_mult
    port map (clk,hu_mult7_1,hu_sub7_1,dummy_mult_nan,dummy_mult_overflow,hu_mult7_2,dummy_mult_underflow);


    mult_hu7_3: mega_mult
    port map (clk,const_3,hu_sub5_2,dummy_mult_nan,dummy_mult_overflow,hu_mult7_3,dummy_mult_underflow);
    sub_hu7_2: mega_sub
    port map (clk,eta23,eta41,dummy_sub_nan,dummy_sub_overflow,hu_sub7_2,dummy_sub_underflow);
    mult_hu7_4: mega_mult
    port map (clk,hu_sub7_2,hu_add4_2,dummy_mult_nan,dummy_mult_overflow,hu_mult7_4,dummy_mult_underflow);
    mult_hu7_5: mega_mult
    port map (clk,hu_mult7_3,hu_mult7_4,dummy_mult_nan,dummy_mult_overflow,hu_mult7_5,dummy_mult_underflow);

    add_hu7: mega_add
    port map (clk,hu_mult7_2,hu_mult7_5,dummy_add_nan,dummy_add_overflow,hu7,dummy_add_underflow);



-------------------------------------------------------------------------------------
--      Final Output
--------------------------------------------------------------------
```

```
--------------------




------------------------------------------------------------------------------------
--                  USER DETECTION ALGORITHM
------------------------------------------------------------------------------------

mask<=X"7FFFFFFF";

   input1<=hu1;
   input2<=hu2;
   input3<=hu3;
   input4<=hu4;
   input5<=hu5;
   input6<=hu6;
   input7<=hu7;



-----------------------------------------------------------------
--          COMPUTATION FOR NEURON 1                          --
-----------------------------------------------------------------
    addr1_1<=X"000";
    addr1_2<=X"001";
    addr1_3<=X"002";
    addr1_4<=X"003";
    addr1_5<=X"004";
    addr1_6<=X"005";
    addr1_7<=X"006";
    im_type(0)<=X"007";


    mem_read1_1: rom
    port map (addr1_1,clk,mem1_1);
    mem_read1_2: rom
    port map (addr1_2,clk,mem1_2);
    mem_read1_3: rom
    port map (addr1_3,clk,mem1_3);
    mem_read1_4: rom
    port map (addr1_4,clk,mem1_4);
    mem_read1_5: rom
    port map (addr1_5,clk,mem1_5);
    mem_read1_6: rom
    port map (addr1_6,clk,mem1_6);
    mem_read1_7: rom
    port map (addr1_7,clk,mem1_7);

    user_sub1_1: mega_sub
    port map (clk,input1,mem1_1,dummy_sub_nan,dummy_sub_overflow,sub1_1,dummy_sub_underflow);
    user_sub1_2: mega_sub
    port map (clk,input2,mem1_2,dummy_sub_nan,dummy_sub_overflow,sub1_2,dummy_sub_underflow);
    user_sub1_3: mega_sub
    port map (clk,input3,mem1_3,dummy_sub_nan,dummy_sub_overflow,sub1_3,dummy_sub_underflow);
    user_sub1_4: mega_sub
    port map (clk,input4,mem1_4,dummy_sub_nan,dummy_sub_overflow,sub
1_4,dummy_sub_underflow);
    user_sub1_5: mega_sub
    port map (clk,input5,mem1_5,dummy_sub_nan,dummy_sub_overflow,sub
1_5,dummy_sub_underflow);
    user_sub1_6: mega_sub
    port map (clk,input6,mem1_6,dummy_sub_nan,dummy_sub_overflow,sub
1_6,dummy_sub_underflow);
    user_sub1_7: mega_sub
    port map (clk,input7,mem1_7,dummy_sub_nan,dummy_sub_overflow,sub
1_7,dummy_sub_underflow);

--Calculating absolute value
    abs1_1<=sub1_1 AND mask;
    abs1_2<=sub1_2 AND mask;
    abs1_3<=sub1_3 AND mask;
    abs1_4<=sub1_4 AND mask;
    abs1_5<=sub1_5 AND mask;
    abs1_6<=sub1_6 AND mask;
    abs1_7<=sub1_7 AND mask;

--Addition operation (Calculating Manhatan Distance)
    user_add1_1: mega_add
    port map (clk,abs1_1,abs1_2,dummy_sub_nan,dummy_sub_overflow,add
1_12,dummy_sub_underflow);
    user_add1_2: mega_add
    port map (clk,abs1_3,abs1_4,dummy_sub_nan,dummy_sub_overflow,add
1_34,dummy_sub_underflow);
    user_add1_3: mega_add
    port map (clk,abs1_5,abs1_6,dummy_sub_nan,dummy_sub_overflow,add
1_56,dummy_sub_underflow);
    user_add1_4: mega_add
    port map (clk,add1_12,abs1_7,dummy_sub_nan,dummy_sub_overflow,ad
d1_27,dummy_sub_underflow);
    user_add1_5: mega_add
    port map (clk,add1_34,add1_56,dummy_sub_nan,dummy_sub_overflow,a
dd1_46,dummy_sub_underflow);
    user_add1_6: mega_add
    port map (clk,add1_27,add1_46,dummy_sub_nan,dummy_sub_overflow,d
istance(0),dummy_sub_underflow);

----------------------------------------------------------------------
--          CALCULATION END FOR NEURON 1
----------------------------------------------------------------------


------------------------------------------------------------------
--          COMPUTATION FOR NEURON 2                            --
------------------------------------------------------------------
    addr2_1<=X"008";
    addr2_2<=X"009";
    addr2_3<=X"00A";
    addr2_4<=X"00B";
    addr2_5<=X"00C";
    addr2_6<=X"00D";
    addr2_7<=X"00E";
    im_type(1)<=X"00F";


    mem_read2_1: rom
    port map (addr2_1,clk,mem2_1);
    mem_read2_2: rom
    port map (addr2_2,clk,mem2_2);
    mem_read2_3: rom
    port map (addr2_3,clk,mem2_3);
    mem_read2_4: rom
    port map (addr2_4,clk,mem2_4);
    mem_read2_5: rom
    port map (addr2_5,clk,mem2_5);
    mem_read2_6: rom
    port map (addr2_6,clk,mem2_6);
    mem_read2_7: rom
    port map (addr2_7,clk,mem2_7);

    user_sub2_1: mega_sub
    port map (clk,input1,mem2_1,dummy_sub_nan,dummy_sub_overflow,sub2_1,dummy_sub_underflow);
    user_sub2_2: mega_sub
    port map (clk,input2,mem2_2,dummy_sub_nan,dummy_sub_overflow,sub2_2,dummy_sub_underflow);
    user_sub2_3: mega_sub
    port map (clk,input3,mem2_3,dummy_sub_nan,dummy_sub_overflow,sub2_3,dummy_sub_underflow);
    user_sub2_4: mega_sub
    port map (clk,input4,mem2_4,dummy_sub_nan,dummy_sub_overflow,sub2_4,dummy_sub_underflow);
    user_sub2_5: mega_sub
    port map (clk,input5,mem2_5,dummy_sub_nan,dummy_sub_overflow,sub2_5,dummy_sub_underflow);
    user_sub2_6: mega_sub
    port map (clk,input6,mem2_6,dummy_sub_nan,dummy_sub_overflow,sub2_6,dummy_sub_underflow);
    user_sub2_7: mega_sub
    port map (clk,input7,mem2_7,dummy_sub_nan,dummy_sub_overflow,sub2_7,dummy_sub_underflow);

--Calculating absolute value
    abs2_1<=sub2_1 AND mask;
    abs2_2<=sub2_2 AND mask;
    abs2_3<=sub2_3 AND mask;
    abs2_4<=sub2_4 AND mask;
    abs2_5<=sub2_5 AND mask;
    abs2_6<=sub2_6 AND mask;
    abs2_7<=sub2_7 AND mask;

--Addition operation (Calculating Manhatan Distance)
    user_add2_1: mega_add
    port map (clk,abs2_1,abs2_2,dummy_sub_nan,dummy_sub_overflow,add2_12,dummy_sub_underflow);
    user_add2_2: mega_add
    port map (clk,abs2_3,abs2_4,dummy_sub_nan,dummy_sub_overflow,add2_34,dummy_sub_underflow);
    user_add2_3: mega_add
    port map (clk,abs2_5,abs2_6,dummy_sub_nan,dummy_sub_overflow,add2_56,dummy_sub_underflow);
    user_add2_4: mega_add
    port map (clk,add2_12,abs2_7,dummy_sub_nan,dummy_sub_overflow,add2_27,dummy_sub_underflow);
    user_add2_5: mega_add
    port map (clk,add2_34,add2_56,dummy_sub_nan,dummy_sub_overflow,add2_46,dummy_sub_underflow);
    user_add2_6: mega_add
    port map (clk,add2_27,add2_46,dummy_sub_nan,dummy_sub_overflow,distance(1),dummy_sub_underflow);

```

```
--------------------------------------------------------------------
--          CALCULATION END FOR NEURON 2
--------------------------------------------------------------------

----------------------------------------------------------------
--          COMPUTATION FOR NEURON 3                          --
----------------------------------------------------------------
    addr3_1<=X"010";
    addr3_2<=X"011";
    addr3_3<=X"012";
    addr3_4<=X"013";
    addr3_5<=X"014";
    addr3_6<=X"015";
    addr3_7<=X"016";
    im_type(2)<=X"017";


    mem_read3_1: rom
    port map (addr3_1,clk,mem3_1);
    mem_read3_2: rom
    port map (addr3_2,clk,mem3_2);
    mem_read3_3: rom
    port map (addr3_3,clk,mem3_3);
    mem_read3_4: rom
    port map (addr3_4,clk,mem3_4);
    mem_read3_5: rom
    port map (addr3_5,clk,mem3_5);
    mem_read3_6: rom
    port map (addr3_6,clk,mem3_6);
    mem_read3_7: rom
    port map (addr3_7,clk,mem3_7);

    user_sub3_1: mega_sub
    port map (clk,input1,mem3_1,dummy_sub_nan,dummy_sub_overflow,sub
3_1,dummy_sub_underflow);
    user_sub3_2: mega_sub
    port map (clk,input2,mem3_2,dummy_sub_nan,dummy_sub_overflow,sub
3_2,dummy_sub_underflow);
    user_sub3_3: mega_sub
    port map (clk,input3,mem3_3,dummy_sub_nan,dummy_sub_overflow,sub
3_3,dummy_sub_underflow);
    user_sub3_4: mega_sub
    port map (clk,input4,mem3_4,dummy_sub_nan,dummy_sub_overflow,sub
3_4,dummy_sub_underflow);
    user_sub3_5: mega_sub
    port map (clk,input5,mem3_5,dummy_sub_nan,dummy_sub_overflow,sub
3_5,dummy_sub_underflow);
    user_sub3_6: mega_sub
    port map (clk,input6,mem3_6,dummy_sub_nan,dummy_sub_overflow,sub
3_6,dummy_sub_underflow);
    user_sub3_7: mega_sub
    port map (clk,input7,mem3_7,dummy_sub_nan,dummy_sub_overflow,sub
3_7,dummy_sub_underflow);

--Calculating absolute value
    abs3_1<=sub3_1 AND mask;
    abs3_2<=sub3_2 AND mask;
    abs3_3<=sub3_3 AND mask;
    abs3_4<=sub3_4 AND mask;
    abs3_5<=sub3_5 AND mask;
    abs3_6<=sub3_6 AND mask;
    abs3_7<=sub3_7 AND mask;
```

```vhdl

--Addition operation (Calculating Manhatan Distance)
    user_add3_1: mega_add
    port map (clk,abs3_1,abs3_2,dummy_sub_nan,dummy_sub_overflow,add
3_12,dummy_sub_underflow);
    user_add3_2: mega_add
    port map (clk,abs3_3,abs3_4,dummy_sub_nan,dummy_sub_overflow,add
3_34,dummy_sub_underflow);
    user_add3_3: mega_add
    port map (clk,abs3_5,abs3_6,dummy_sub_nan,dummy_sub_overflow,add
3_56,dummy_sub_underflow);
    user_add3_4: mega_add
    port map (clk,add3_12,abs3_7,dummy_sub_nan,dummy_sub_overflow,ad
d3_27,dummy_sub_underflow);
    user_add3_5: mega_add
    port map (clk,add3_34,add3_56,dummy_sub_nan,dummy_sub_overflow,a
dd3_46,dummy_sub_underflow);
    user_add3_6: mega_add
    port map (clk,add3_27,add3_46,dummy_sub_nan,dummy_sub_overflow,d
istance(2),dummy_sub_underflow);

---------------------------------------------------------------------
--          CALCULATION END FOR NEURON 3
---------------------------------------------------------------------


-----------------------------------------------------------------
--          COMPUTATION FOR NEURON 4                           --
-----------------------------------------------------------------
    addr4_1<=X"018";
    addr4_2<=X"019";
    addr4_3<=X"01A";
    addr4_4<=X"01B";
    addr4_5<=X"01C";
    addr4_6<=X"01D";
    addr4_7<=X"01E";
    im_type(3)<=X"01F";


    mem_read4_1: rom
    port map (addr4_1,clk,mem4_1);
    mem_read4_2: rom
    port map (addr4_2,clk,mem4_2);
    mem_read4_3: rom
    port map (addr4_3,clk,mem4_3);
    mem_read4_4: rom
    port map (addr4_4,clk,mem4_4);
    mem_read4_5: rom
    port map (addr4_5,clk,mem4_5);
    mem_read4_6: rom
    port map (addr4_6,clk,mem4_6);
    mem_read4_7: rom
    port map (addr4_7,clk,mem4_7);

    user_sub4_1: mega_sub
    port map (clk,input1,mem4_1,dummy_sub_nan,dummy_sub_overflow,sub
4_1,dummy_sub_underflow);
    user_sub4_2: mega_sub
    port map (clk,input2,mem4_2,dummy_sub_nan,dummy_sub_overflow,sub
4_2,dummy_sub_underflow);
    user_sub4_3: mega_sub
    port map (clk,input3,mem4_3,dummy_sub_nan,dummy_sub_overflow,sub
4_3,dummy_sub_underflow);
    user_sub4_4: mega_sub
    port map (clk,input4,mem4_4,dummy_sub_nan,dummy_sub_overflow,sub
4_4,dummy_sub_underflow);
    user_sub4_5: mega_sub
    port map (clk,input5,mem4_5,dummy_sub_nan,dummy_sub_overflow,sub
4_5,dummy_sub_underflow);
    user_sub4_6: mega_sub
    port map (clk,input6,mem4_6,dummy_sub_nan,dummy_sub_overflow,sub
4_6,dummy_sub_underflow);
    user_sub4_7: mega_sub
    port map (clk,input7,mem4_7,dummy_sub_nan,dummy_sub_overflow,sub
4_7,dummy_sub_underflow);

--Calculating absolute value
    abs4_1<=sub4_1 AND mask;
    abs4_2<=sub4_2 AND mask;
    abs4_3<=sub4_3 AND mask;
    abs4_4<=sub4_4 AND mask;
    abs4_5<=sub4_5 AND mask;
    abs4_6<=sub4_6 AND mask;
    abs4_7<=sub4_7 AND mask;

--Addition operation (Calculating Manhatan Distance)
    user_add4_1: mega_add
    port map (clk,abs4_1,abs4_2,dummy_sub_nan,dummy_sub_overflow,add
4_12,dummy_sub_underflow);
    user_add4_2: mega_add
    port map (clk,abs4_3,abs4_4,dummy_sub_nan,dummy_sub_overflow,add
4_34,dummy_sub_underflow);
    user_add4_3: mega_add
    port map (clk,abs4_5,abs4_6,dummy_sub_nan,dummy_sub_overflow,add
4_56,dummy_sub_underflow);
    user_add4_4: mega_add
    port map (clk,add4_12,abs4_7,dummy_sub_nan,dummy_sub_overflow,ad
d4_27,dummy_sub_underflow);
    user_add4_5: mega_add
    port map (clk,add4_34,add4_56,dummy_sub_nan,dummy_sub_overflow,a
dd4_46,dummy_sub_underflow);
    user_add4_6: mega_add
    port map (clk,add4_27,add4_46,dummy_sub_nan,dummy_sub_overflow,d
istance(3),dummy_sub_underflow);

--------------------------------------------------------------------
--          CALCULATION END FOR NEURON 4
--------------------------------------------------------------------

----------------------------------------------------------------
--          COMPUTATION FOR NEURON 5                          --
----------------------------------------------------------------
    addr5_1<=X"020";
    addr5_2<=X"021";
    addr5_3<=X"022";
    addr5_4<=X"023";
    addr5_5<=X"024";
    addr5_6<=X"025";
    addr5_7<=X"026";
    im_type(4)<=X"027";


    mem_read5_1: rom
    port map (addr5_1,clk,mem5_1);
```

```vhdl
    mem_read5_2: rom
    port map (addr5_2,clk,mem5_2);
    mem_read5_3: rom
    port map (addr5_3,clk,mem5_3);
    mem_read5_4: rom
    port map (addr5_4,clk,mem5_4);
    mem_read5_5: rom
    port map (addr5_5,clk,mem5_5);
    mem_read5_6: rom
    port map (addr5_6,clk,mem5_6);
    mem_read5_7: rom
    port map (addr5_7,clk,mem5_7);

    user_sub5_1: mega_sub
    port map (clk,input1,mem5_1,dummy_sub_nan,dummy_sub_overflow,sub5_1,dummy_sub_underflow);
    user_sub5_2: mega_sub
    port map (clk,input2,mem5_2,dummy_sub_nan,dummy_sub_overflow,sub5_2,dummy_sub_underflow);
    user_sub5_3: mega_sub
    port map (clk,input3,mem5_3,dummy_sub_nan,dummy_sub_overflow,sub5_3,dummy_sub_underflow);
    user_sub5_4: mega_sub
    port map (clk,input4,mem5_4,dummy_sub_nan,dummy_sub_overflow,sub5_4,dummy_sub_underflow);
    user_sub5_5: mega_sub
    port map (clk,input5,mem5_5,dummy_sub_nan,dummy_sub_overflow,sub5_5,dummy_sub_underflow);
    user_sub5_6: mega_sub
    port map (clk,input6,mem5_6,dummy_sub_nan,dummy_sub_overflow,sub5_6,dummy_sub_underflow);
    user_sub5_7: mega_sub
    port map (clk,input7,mem5_7,dummy_sub_nan,dummy_sub_overflow,sub5_7,dummy_sub_underflow);

--Calculating absolute value
    abs5_1<=sub5_1 AND mask;
    abs5_2<=sub5_2 AND mask;
    abs5_3<=sub5_3 AND mask;
    abs5_4<=sub5_4 AND mask;
    abs5_5<=sub5_5 AND mask;
    abs5_6<=sub5_6 AND mask;
    abs5_7<=sub5_7 AND mask;

--Addition operation (Calculating Manhatan Distance)
    user_add5_1: mega_add
    port map (clk,abs5_1,abs5_2,dummy_sub_nan,dummy_sub_overflow,add5_12,dummy_sub_underflow);
    user_add5_2: mega_add
    port map (clk,abs5_3,abs5_4,dummy_sub_nan,dummy_sub_overflow,add5_34,dummy_sub_underflow);
    user_add5_3: mega_add
    port map (clk,abs5_5,abs5_6,dummy_sub_nan,dummy_sub_overflow,add5_56,dummy_sub_underflow);
    user_add5_4: mega_add
    port map (clk,add5_12,abs5_7,dummy_sub_nan,dummy_sub_overflow,add5_27,dummy_sub_underflow);
    user_add5_5: mega_add
    port map (clk,add5_34,add5_56,dummy_sub_nan,dummy_sub_overflow,add5_46,dummy_sub_underflow);
    user_add5_6: mega_add
    port map (clk,add5_27,add5_46,dummy_sub_nan,dummy_sub_overflow,d
istance(4),dummy_sub_underflow);

---------------------------------------------------------------------
--           CALCULATION END FOR NEURON 5
---------------------------------------------------------------------


-----------------------------------------------------------------
--           COMPUTATION FOR NEURON 6                          --
-----------------------------------------------------------------
    addr6_1<=X"028";
    addr6_2<=X"029";
    addr6_3<=X"02A";
    addr6_4<=X"02B";
    addr6_5<=X"02C";
    addr6_6<=X"02D";
    addr6_7<=X"02E";
    im_type(5)<=X"02F";


    mem_read6_1: rom
    port map (addr6_1,clk,mem6_1);
    mem_read6_2: rom
    port map (addr6_2,clk,mem6_2);
    mem_read6_3: rom
    port map (addr6_3,clk,mem6_3);
    mem_read6_4: rom
    port map (addr6_4,clk,mem6_4);
    mem_read6_5: rom
    port map (addr6_5,clk,mem6_5);
    mem_read6_6: rom
    port map (addr6_6,clk,mem6_6);
    mem_read6_7: rom
    port map (addr6_7,clk,mem6_7);

    user_sub6_1: mega_sub
    port map (clk,input1,mem6_1,dummy_sub_nan,dummy_sub_overflow,sub
6_1,dummy_sub_underflow);
    user_sub6_2: mega_sub
    port map (clk,input2,mem6_2,dummy_sub_nan,dummy_sub_overflow,sub
6_2,dummy_sub_underflow);
    user_sub6_3: mega_sub
    port map (clk,input3,mem6_3,dummy_sub_nan,dummy_sub_overflow,sub
6_3,dummy_sub_underflow);
    user_sub6_4: mega_sub
    port map (clk,input4,mem6_4,dummy_sub_nan,dummy_sub_overflow,sub
6_4,dummy_sub_underflow);
    user_sub6_5: mega_sub
    port map (clk,input5,mem6_5,dummy_sub_nan,dummy_sub_overflow,sub
6_5,dummy_sub_underflow);
    user_sub6_6: mega_sub
    port map (clk,input6,mem6_6,dummy_sub_nan,dummy_sub_overflow,sub
6_6,dummy_sub_underflow);
    user_sub6_7: mega_sub
    port map (clk,input7,mem6_7,dummy_sub_nan,dummy_sub_overflow,sub
6_7,dummy_sub_underflow);

--Calculating absolute value
    abs6_1<=sub6_1 AND mask;
    abs6_2<=sub6_2 AND mask;
    abs6_3<=sub6_3 AND mask;
    abs6_4<=sub6_4 AND mask;
    abs6_5<=sub6_5 AND mask;
    abs6_6<=sub6_6 AND mask;
    abs6_7<=sub6_7 AND mask;

--Addition operation (Calculating Manhatan Distance)
    user_add6_1: mega_add
    port map (clk,abs6_1,abs6_2,dummy_sub_nan,dummy_sub_overflow,add
6_12,dummy_sub_underflow);
    user_add6_2: mega_add
    port map (clk,abs6_3,abs6_4,dummy_sub_nan,dummy_sub_overflow,add
6_34,dummy_sub_underflow);
    user_add6_3: mega_add
    port map (clk,abs6_5,abs6_6,dummy_sub_nan,dummy_sub_overflow,add
6_56,dummy_sub_underflow);
    user_add6_4: mega_add
    port map (clk,add6_12,abs6_7,dummy_sub_nan,dummy_sub_overflow,ad
d6_27,dummy_sub_underflow);
    user_add6_5: mega_add
    port map (clk,add6_34,add6_56,dummy_sub_nan,dummy_sub_overflow,a
dd6_46,dummy_sub_underflow);
    user_add6_6: mega_add
    port map (clk,add6_27,add6_46,dummy_sub_nan,dummy_sub_overflow,d
istance(5),dummy_sub_underflow);

---------------------------------------------------------------------
--          CALCULATION END FOR NEURON 6
---------------------------------------------------------------------

-----------------------------------------------------------------
--          COMPUTATION FOR NEURON 7                           --
-----------------------------------------------------------------
    addr7_1<=X"030";
    addr7_2<=X"031";
    addr7_3<=X"032";
    addr7_4<=X"033";
    addr7_5<=X"034";
    addr7_6<=X"035";
    addr7_7<=X"036";
    im_type(6)<=X"037";


    mem_read7_1: rom
    port map (addr7_1,clk,mem7_1);
    mem_read7_2: rom
    port map (addr7_2,clk,mem7_2);
    mem_read7_3: rom
    port map (addr7_3,clk,mem7_3);
    mem_read7_4: rom
    port map (addr7_4,clk,mem7_4);
    mem_read7_5: rom
    port map (addr7_5,clk,mem7_5);
    mem_read7_6: rom
    port map (addr7_6,clk,mem7_6);
    mem_read7_7: rom
    port map (addr7_7,clk,mem7_7);

    user_sub7_1: mega_sub
    port map (clk,input1,mem7_1,dummy_sub_nan,dummy_sub_overflow,sub
7_1,dummy_sub_underflow);
    user_sub7_2: mega_sub
    port map (clk,input2,mem7_2,dummy_sub_nan,dummy_sub_overflow,sub
7_2,dummy_sub_underflow);
```

```vhdl
    user_sub7_3: mega_sub
    port map (clk,input3,mem7_3,dummy_sub_nan,dummy_sub_overflow,sub7_3,dummy_sub_underflow);
    user_sub7_4: mega_sub
    port map (clk,input4,mem7_4,dummy_sub_nan,dummy_sub_overflow,sub7_4,dummy_sub_underflow);
    user_sub7_5: mega_sub
    port map (clk,input5,mem7_5,dummy_sub_nan,dummy_sub_overflow,sub7_5,dummy_sub_underflow);
    user_sub7_6: mega_sub
    port map (clk,input6,mem7_6,dummy_sub_nan,dummy_sub_overflow,sub7_6,dummy_sub_underflow);
    user_sub7_7: mega_sub
    port map (clk,input7,mem7_7,dummy_sub_nan,dummy_sub_overflow,sub7_7,dummy_sub_underflow);

--Calculating absolute value
    abs7_1<=sub7_1 AND mask;
    abs7_2<=sub7_2 AND mask;
    abs7_3<=sub7_3 AND mask;
    abs7_4<=sub7_4 AND mask;
    abs7_5<=sub7_5 AND mask;
    abs7_6<=sub7_6 AND mask;
    abs7_7<=sub7_7 AND mask;

--Addition operation (Calculating Manhatan Distance)
    user_add7_1: mega_add
    port map (clk,abs7_1,abs7_2,dummy_sub_nan,dummy_sub_overflow,add7_12,dummy_sub_underflow);
    user_add7_2: mega_add
    port map (clk,abs7_3,abs7_4,dummy_sub_nan,dummy_sub_overflow,add7_34,dummy_sub_underflow);
    user_add7_3: mega_add
    port map (clk,abs7_5,abs7_6,dummy_sub_nan,dummy_sub_overflow,add7_56,dummy_sub_underflow);
    user_add7_4: mega_add
    port map (clk,add7_12,abs7_7,dummy_sub_nan,dummy_sub_overflow,add7_27,dummy_sub_underflow);
    user_add7_5: mega_add
    port map (clk,add7_34,add7_56,dummy_sub_nan,dummy_sub_overflow,add7_46,dummy_sub_underflow);
    user_add7_6: mega_add
    port map (clk,add7_27,add7_46,dummy_sub_nan,dummy_sub_overflow,distance(6),dummy_sub_underflow);

--------------------------------------------------------------------
--           CALCULATION END FOR NEURON 7
--------------------------------------------------------------------


----------------------------------------------------------------
--           COMPUTATION FOR NEURON 8                         --
----------------------------------------------------------------
    addr8_1<=X"038";
    addr8_2<=X"039";
    addr8_3<=X"03A";
    addr8_4<=X"03B";
    addr8_5<=X"03C";
    addr8_6<=X"03D";
    addr8_7<=X"03E";
    im_type(7)<=X"03F";


    mem_read8_1: rom
    port map (addr8_1,clk,mem8_1);
    mem_read8_2: rom
    port map (addr8_2,clk,mem8_2);
    mem_read8_3: rom
    port map (addr8_3,clk,mem8_3);
    mem_read8_4: rom
    port map (addr8_4,clk,mem8_4);
    mem_read8_5: rom
    port map (addr8_5,clk,mem8_5);
    mem_read8_6: rom
    port map (addr8_6,clk,mem8_6);
    mem_read8_7: rom
    port map (addr8_7,clk,mem8_7);

    user_sub8_1: mega_sub
    port map (clk,input1,mem8_1,dummy_sub_nan,dummy_sub_overflow,sub8_1,dummy_sub_underflow);
    user_sub8_2: mega_sub
    port map (clk,input2,mem8_2,dummy_sub_nan,dummy_sub_overflow,sub8_2,dummy_sub_underflow);
    user_sub8_3: mega_sub
    port map (clk,input3,mem8_3,dummy_sub_nan,dummy_sub_overflow,sub8_3,dummy_sub_underflow);
    user_sub8_4: mega_sub
    port map (clk,input4,mem8_4,dummy_sub_nan,dummy_sub_overflow,sub8_4,dummy_sub_underflow);
    user_sub8_5: mega_sub
    port map (clk,input5,mem8_5,dummy_sub_nan,dummy_sub_overflow,sub8_5,dummy_sub_underflow);
    user_sub8_6: mega_sub
    port map (clk,input6,mem8_6,dummy_sub_nan,dummy_sub_overflow,sub8_6,dummy_sub_underflow);
    user_sub8_7: mega_sub
    port map (clk,input7,mem8_7,dummy_sub_nan,dummy_sub_overflow,sub8_7,dummy_sub_underflow);

--Calculating absolute value
    abs8_1<=sub8_1 AND mask;
    abs8_2<=sub8_2 AND mask;
    abs8_3<=sub8_3 AND mask;
    abs8_4<=sub8_4 AND mask;
    abs8_5<=sub8_5 AND mask;
    abs8_6<=sub8_6 AND mask;
    abs8_7<=sub8_7 AND mask;

--Addition operation (Calculating Manhatan Distance)
    user_add8_1: mega_add
    port map (clk,abs8_1,abs8_2,dummy_sub_nan,dummy_sub_overflow,add8_12,dummy_sub_underflow);
    user_add8_2: mega_add
    port map (clk,abs8_3,abs8_4,dummy_sub_nan,dummy_sub_overflow,add8_34,dummy_sub_underflow);
    user_add8_3: mega_add
    port map (clk,abs8_5,abs8_6,dummy_sub_nan,dummy_sub_overflow,add8_56,dummy_sub_underflow);
    user_add8_4: mega_add
    port map (clk,add8_12,abs8_7,dummy_sub_nan,dummy_sub_overflow,add8_27,dummy_sub_underflow);
    user_add8_5: mega_add
    port map (clk,add8_34,add8_56,dummy_sub_nan,dummy_sub_overflow,a
dd8_46,dummy_sub_underflow);
    user_add8_6: mega_add
    port map (clk,add8_27,add8_46,dummy_sub_nan,dummy_sub_overflow,d
istance(7),dummy_sub_underflow);

---------------------------------------------------------------------
--          CALCULATION END FOR NEURON 8
---------------------------------------------------------------------
    process(clock,distance(0),distance(1),distance(2),distance(3),di
stance(4),distance(5),distance(6),distance(7))
    begin

        if(clock'EVENT AND clock='1') then  --the calculations are b
ased on clock cycle
            count2<=count2+1;
            count3<=count3+1;


        if (count2 = 20) then

                temp<=distance(0);
                index <= 0;

        end if;

        if (count2 = 21) then
            if (temp > distance(1))then
                temp<=distance(1);
                index <= 1;
            end if;
        end if;

        if (count2 = 22) then
            if (temp > distance(2))then
                temp<=distance(2);
                index <= 2;
            end if;
        end if;

        if (count2 = 23) then
            if (temp > distance(3))then
                temp<=distance(3);
                index <= 3;
            end if;
        end if;

        if (count2 = 24) then
            if (temp > distance(4))then
                temp<=distance(4);
                index <= 4;
            end if;
        end if;

        if (count2 = 25) then
            if (temp > distance(5))then
                temp<=distance(5);
                index <= 5;
            end if;
        end if;

        if (count2 = 26) then
            if (temp > distance(6))then
```

```vhdl
                temp<=distance(6);
                index <= 6;
            end if;
        end if;

        if (count2 = 27) then
            if (temp > distance(7))then
                temp<=distance(7);
                index <= 7;
            end if;
        end if;
        end if;

    end process;

    mem_read_final: rom
    port map (im_type(index),clock,test1);





end behaviour;


---------------------------------------------------------------------------------------------
--             MEGA FUNCTION FLOATING POINT ADDITION
---------------------------------------------------------------------------------------------
-- megafunction wizard: %ALTFP_ADD_SUB%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altfp_add_sub

-- ============================================================
-- File Name: mega_add.vhd
-- Megafunction Name(s):
--          altfp_add_sub
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 5.1 Build 176 10/26/2005 SJ Full Version
-- ************************************************************


--=====================================================
--      SYSTEM GENERATED CODE. Not included in thesis
--=====================================================



---------------------------------------------------------------------------------------------
--             MEGA FUNCTION FLOATING POINT SUBTRACTION
---------------------------------------------------------------------------------------------


-- megafunction wizard: %ALTFP_ADD_SUB%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altfp_add_sub

-- ============================================================
-- File Name: mega_sub.vhd
-- Megafunction Name(s):
--          altfp_add_sub
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 5.1 Build 176 10/26/2005 SJ Full Version
-- ************************************************************

--========================================================
--      SYSTEM GENERATED CODE. Not included in thesis
--========================================================




---------------------------------------------------------------------
-------------------------
--             MEGA FUNCTION FLOATING POINT MULTIPLICATION
---------------------------------------------------------------------
-------------------------
-- megafunction wizard: %ALTFP_MULT%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altfp_mult

-- ============================================================
-- File Name: mega_mult.vhd
-- Megafunction Name(s):
--          altfp_mult
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 5.1 Build 176 10/26/2005 SJ Full Version
-- ************************************************************



LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY mega_mult IS
    PORT
    (
        clock       : IN STD_LOGIC ;
        dataa       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        datab       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        nan     : OUT STD_LOGIC ;
        overflow        : OUT STD_LOGIC ;
        result      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        underflow       : OUT STD_LOGIC
    );
```

```
END mega_mult;


ARCHITECTURE SYN OF mega_mult IS

    SIGNAL sub_wire0    : STD_LOGIC ;
    SIGNAL sub_wire1    : STD_LOGIC ;
    SIGNAL sub_wire2    : STD_LOGIC ;
    SIGNAL sub_wire3    : STD_LOGIC_VECTOR (31 DOWNTO 0);



    COMPONENT altfp_mult
    GENERIC (
        dedicated_multiplier_circuitry      : STRING;
        lpm_type        : STRING;
        pipeline        : NATURAL;
        reduced_functionality       : STRING;
        width_exp       : NATURAL;
        width_man       : NATURAL
    );
    PORT (
            dataa   : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            datab   : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            overflow    : OUT STD_LOGIC ;
            nan : OUT STD_LOGIC ;
            underflow   : OUT STD_LOGIC ;
            clock   : IN STD_LOGIC ;
            result  : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
    END COMPONENT;

BEGIN
    overflow    <= sub_wire0;
    nan    <= sub_wire1;
    underflow    <= sub_wire2;
    result    <= sub_wire3(31 DOWNTO 0);

    altfp_mult_component : altfp_mult
    GENERIC MAP (
        dedicated_multiplier_circuitry => "YES",
        lpm_type => "altfp_mult",
        pipeline => 5,
        reduced_functionality => "YES",
        width_exp => 8,
        width_man => 23
    )
    PORT MAP (
        dataa => dataa,
        datab => datab,
        clock => clock,
        overflow => sub_wire0,
        nan => sub_wire1,
        underflow => sub_wire2,
        result => sub_wire3
    );



END SYN;
```

```vhdl
-------------------------------------------------------------------
---------------
--              Integer to IEEE 754 Floating Point format conversion
-------------------------------------------------------------------
---------------

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;


-- Top-level entity
ENTITY int2fp IS

PORT (input : integer range 0 to 100000;

      fp_op : OUT STD_LOGIC_VECTOR(31 downto 0));


END int2fp ;


ARCHITECTURE Behavior OF int2fp IS

signal int_ip : STD_LOGIC_VECTOR(31 downto 0);
signal sign_bit : STD_LOGIC ;
signal exponent : STD_LOGIC_VECTOR(7 downto 0);
signal mentissa : STD_LOGIC_VECTOR(22 downto 0);

BEGIN

    int_ip<=(conv_std_logic_vector(input,32));

    sign_bit<='0';

    process(int_ip)
    begin
        if(int_ip(31)='1')then
            exponent<=X"9E";
            mentissa<=int_ip(30 downto 8);

        elsif(int_ip(30)='1')then
            exponent<=X"9D";
            mentissa<=int_ip(29 downto 7);

        elsif(int_ip(29)='1')then
            exponent<=X"9C";
            mentissa<=int_ip(28 downto 6);

        elsif(int_ip(28)='1')then
            exponent<=X"9B";
            mentissa<=int_ip(27 downto 5);

        elsif(int_ip(27)='1')then
            exponent<=X"9A";
            mentissa<=int_ip(26 downto 4);

        elsif(int_ip(26)='1')then
            exponent<=X"99";
            mentissa<=int_ip(25 downto 3);

        elsif(int_ip(25)='1')then
            exponent<=X"98";
            mentissa<=int_ip(24 downto 2);

        elsif(int_ip(24)='1')then
            exponent<=X"97";
            mentissa<=int_ip(23 downto 1);

        elsif(int_ip(23)='1')then
            exponent<=X"96";
            mentissa<=int_ip(22 downto 0);

        elsif(int_ip(22)='1')then
            exponent<=X"95";
            mentissa(22 downto 1)<=int_ip(21 downto 0);
            mentissa(0)<='0';

        elsif(int_ip(21)='1')then
            exponent<=X"94";
            mentissa(22 downto 2)<=int_ip(20 downto 0);
            mentissa(1 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(20)='1')then
            exponent<=X"93";
            mentissa(22 downto 3)<=int_ip(19 downto 0);
            mentissa(2 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(19)='1')then
            exponent<=X"92";
            mentissa(22 downto 4)<=int_ip(18 downto 0);
            mentissa(3 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(18)='1')then
            exponent<=X"91";
            mentissa(22 downto 5)<=int_ip(17 downto 0);
            mentissa(4 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(17)='1')then
            exponent<=X"90";
            mentissa(22 downto 6)<=int_ip(16 downto 0);
            mentissa(5 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(16)='1')then
            exponent<=X"8F";
            mentissa(22 downto 7)<=int_ip(15 downto 0);
            mentissa(6 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(15)='1')then
            exponent<=X"8E";
            mentissa(22 downto 8)<=int_ip(14 downto 0);
            mentissa(7 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(14)='1')then
            exponent<=X"8D";
            mentissa(22 downto 9)<=int_ip(13 downto 0);
            mentissa(8 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(13)='1')then
            exponent<=X"8C";
            mentissa(22 downto 10)<=int_ip(12 downto 0);
            mentissa(9 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(12)='1')then
            exponent<=X"8B";
            mentissa(22 downto 11)<=int_ip(11 downto 0);
            mentissa(10 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(11)='1')then
            exponent<=X"8A";
            mentissa(22 downto 12)<=int_ip(10 downto 0);
            mentissa(11 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(10)='1')then
            exponent<=X"89";
            mentissa(22 downto 13)<=int_ip(9 downto 0);
            mentissa(12 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(9)='1')then
            exponent<=X"88";
            mentissa(22 downto 14)<=int_ip(8 downto 0);
            mentissa(13 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(8)='1')then
            exponent<=X"87";
            mentissa(22 downto 15)<=int_ip(7 downto 0);
            mentissa(14 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(7)='1')then
            exponent<=X"86";
            mentissa(22 downto 16)<=int_ip(6 downto 0);
            mentissa(15 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(6)='1')then
            exponent<=X"85";
            mentissa(22 downto 17)<=int_ip(5 downto 0);
            mentissa(16 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(5)='1')then
            exponent<=X"84";
            mentissa(22 downto 18)<=int_ip(4 downto 0);
            mentissa(17 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(4)='1')then
            exponent<=X"83";
            mentissa(22 downto 19)<=int_ip(3 downto 0);
            mentissa(18 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(3)='1')then
            exponent<=X"82";
            mentissa(22 downto 20)<=int_ip(2 downto 0);
            mentissa(19 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(2)='1')then
            exponent<=X"81";
            mentissa(22 downto 21)<=int_ip(1 downto 0);
            mentissa(20 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(1)='1')then
            exponent<=X"80";
            mentissa(22)<=int_ip(0);
            mentissa(21 downto 0)<=(OTHERS=>'0');

        elsif(int_ip(0)='1')then
            exponent<=X"7F";
```

```
             mentissa(22 downto 0)<=(OTHERS=>'0');

         end if;

     end process;

fp_op(31)<=sign_bit;
fp_op(30 downto 23)<=exponent;
fp_op(22 downto 0)<=mentissa;


END Behavior;






---------------------------------------------------------------------
------------------
--                       CALCULATION OF FLOATING POINT DIVISION
---------------------------------------------------------------------
------------------

---------------------------------------------------------------------
---------------
--       Calculation of Floating point division
---------------------------------------------------------------------
---------------



--
---------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
--
-- Design: A Row of an Integer Divider
-- Name:    Tiffany Crawford and Clay Gloster
--
-- Date:    May 2001
-- Description: Generic Module
--  N is the width of the numerator and denominators
--  Produces one output bit Q
---------------------------------------------------------------
       library IEEE;
       use IEEE.std_logic_1164.all;
       use IEEE.std_logic_arith.all;
       use IEEE.std_logic_unsigned."-";
       use IEEE.std_logic_unsigned.">";
       use IEEE.std_logic_unsigned."=";



       entity row_generic is      --Declare a row with N cells
       generic  (N: integer);
         port( NUM: in std_logic_vector(N-1 downto 0);
               DEN : in std_logic_vector  (N-1 downto 0);
               DIFF : out std_logic_vector(N-1 downto 0);
               DIV_OUT: out std_logic_vector(N-1 downto 0);
               Q : out std_logic );
       end row_generic;

```

```


        architecture functional of row_generic is

         begin
         process(NUM,DEN)

         begin
         IF (NUM >= DEN) then DIFF <= NUM - DEN;
         Q <= '1';
         else
         DIFF <= NUM;
         Q <= '0';
         end if;
         DIV_OUT <= DEN;

         end process;
        end functional;
--------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
--
-- Design: A Connect Module that connects two rows of an integer div
ider
-- Name:    Tiffany Crawford and Clay Gloster
--
-- Date:    May 2001
-- Description: Generic Module
--  N is the width of the inputs and outputs
--  Passes div_out to den for the next row
--  Shifts diff for use in the next row numerator.
--------------------------------------------------------------

         library IEEE;
         use IEEE.std_logic_1164.all;


         entity row_connect is             --Connects the rows
       generic (N : integer);
         port (DIFF: in std_logic_vector(N-1 downto 0);
                 DIV_OUT: in std_logic_vector(N-1 downto 0);
             NUM: out std_logic_vector(N-1 downto 0);
                 DEN: out std_logic_vector(N-1 downto 0));
         end row_connect;


       architecture structural of row_connect is
         begin

      lpp1: for i in N-1 downto 1 generate  --Connects left side of t
he array
                 NUM(i) <= DIFF(i-1);
       end generate;
                 NUM(0) <= '0';
         lpp2: for i in N-1 downto 0 generate          --Connects rig
ht side of array
                 DEN (i) <= DIV_OUT(i);
         end generate;

         end structural;
--------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
--
```

```
-- Design: A Connect Module that connects the FIRST TWO rows of an integer divider
-- Name:    Tiffany Crawford and Clay Gloster
--
-- Date:    May 2001
-- Description: Generic Module
--  N is the width of the inputs and outputs
--  Passes div_out to den for the next row
--  Shifts diff for use in the next row numerator.
--------------------------------------------------------------


library IEEE;
        use IEEE.std_logic_1164.all;


        entity first_row_connect is              --Connects the rows
      generic (N : integer);
        port (DIFF: in std_logic_vector(N-1 downto 0);
                DIV_OUT: in std_logic_vector(N-1 downto 0);
            NUM: out std_logic_vector(N downto 0);
            DEN: out std_logic_vector(N downto 0) );
        end first_row_connect;


      architecture behavioral of first_row_connect is
        begin

      lpp1: for i in N downto 1 generate   --Connects left side of array
                NUM(i) <= DIFF(i-1);
      end generate;
                NUM(0) <= '0';


       DEN(N) <= '0';
        lpp2: for j in N-1 downto 0 generate   --  Connects right side of array
                DEN (j) <= DIV_OUT(j);

        end generate;

        end behavioral;


--------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
--
-- Design: The first row and interconnect of an integer divider
-- Name:    Tiffany Crawford and Clay Gloster
--
-- Date:    May 2001
-- Description: Generic Module
--  N is the width of the numerator and denominator
--  Outputs connecting to the second row are N+1 bits wide.
--  Produces one bit of the quotient.
--------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--Carries out first row of division
entity first_row_divider is
generic (N: integer);
```

```vhdl
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     NUM1: out std_logic_vector(N downto 0);
     DEN1: out std_logic_vector(N downto 0);
     Q: out std_logic );
end first_row_divider;


architecture structural of first_row_divider is


component row_generic
generic(N: INTEGER);
port( NUM : in std_logic_vector(N-1 downto 0);
            DEN : in std_logic_vector (N-1 downto 0);
            DIFF : out std_logic_vector(N-1 downto 0);
            DIV_OUT: out std_logic_vector(N-1 downto 0);
      Q : out std_logic );
end component;


component first_row_connect
      generic (N : integer);
        port (DIFF: in std_logic_vector(N-1 downto 0);
                DIV_OUT: in std_logic_vector(N-1 downto 0);
            NUM: out std_logic_vector(N downto 0);
            DEN: out std_logic_vector(N downto 0) );
end component;


signal SDIFF1,SDIV_OUT1:
             STD_LOGIC_VECTOR(N-1 DOWNTO 0);




        begin
         U0: row_generic generic map(N)
     port map(NUM,DEN, SDIFF1, SDIV_OUT1,Q);
       U1: first_row_connect generic map(N)
           port map(SDIFF1, SDIV_OUT1, NUM1, DEN1);


        end structural;


---------------------------------------------------------------------
------------


--------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
--
-- Design: A block of an integer divider (not including the first ro
w.)
-- Name:    Tiffany Crawford and Clay Gloster
--
-- Date:    May 2001
-- Description: Generic Module
--  N is the width of the numerator and denominator (combinational l
ogic)
--  Passes DEN to DIV_OUT for the next block of the divider
```

```vhdl
--------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.all;
entity block_generic is
generic (M, N: integer);
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     DIFF: out std_logic_vector(N-1 downto 0);
     DIV_OUT: out std_logic_vector(N-1 downto 0);
     Q: out std_logic_vector(M-1 downto 0));
end block_generic;


architecture structural of block_generic is


component row_connect
      generic (N : integer);
        port (DIFF: in std_logic_vector(N-1 downto 0);
            DIV_OUT: in std_logic_vector(N-1 downto 0);
            NUM: out std_logic_vector(N-1 downto 0);
            DEN: out std_logic_vector(N-1 downto 0));
end component;



component row_generic
generic(N: INTEGER);
port( NUM : in std_logic_vector(N-1 downto 0);
            DEN : in std_logic_vector (N-1 downto 0);
            DIFF : out std_logic_vector(N-1 downto 0);
            DIV_OUT: out std_logic_vector(N-1 downto 0);
      Q : out std_logic );
end component;


type sig2d is array (0 to M) of std_logic_vector (N-1 downto 0);
signal SNUM,SDEN : sig2d;
signal SDIFF, SDIV_OUT: sig2d;


        begin

        --Carries out remaining rows
      lpp1: for i in M downto 1 generate       --Produces an M bit Q
uotient
UNA: row_generic generic map (N)
     port map( SNUM(i),SDEN(i),SDIFF(i),SDIV_OUT(i),Q(i-1));
UNB: row_connect generic map(N)
     port map(SDIFF(i),SDIV_OUT(i),SNUM(i-1),SDEN(i-1) );
     end generate;
        SNUM(M)<= NUM;
      SDEN(M) <= DEN;


      DIFF<= SNUM(0);
      DIV_OUT <= SDEN(0);
end structural;
--------------------------------------------------------------------
------------
```

```vhdl
---------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
--
-- Design: A generic combinational integer divider
-- Name:    Tiffany Crawford and Clay Gloster
--
-- Date:    May 2001
-- Description: Generic Module
--  N is the width of the numerator and denominator
--  M is the width of the quotient
--  Passes dif and div_out to ouput for the next block
--  ---------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.all;
entity generic_divider is


generic (M, N: integer);
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     DIFF: out std_logic_vector(N downto 0);
     DIV_OUT: out std_logic_vector(N downto 0);
     Q: out std_logic_vector(M-1 downto  0));
end generic_divider;


architecture structural of generic_divider is


component first_row_divider
generic (N: integer);
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     NUM1: out std_logic_vector(N downto 0);
     DEN1: out std_logic_vector(N downto 0);
     Q: out std_logic );
end component;


component block_generic
generic (M, N: integer);
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     DIFF: out std_logic_vector(N-1 downto 0);
     DIV_OUT: out std_logic_vector(N-1 downto 0);
     Q: out std_logic_vector(M-1 downto 0));
end component;


signal SNUM1,SDEN1: std_logic_vector (N downto 0);
signal TQ: std_logic;
signal TTQ: std_logic_vector(M-2 downto 0);




begin


         --Carries out first row
```

```
U0: first_row_divider
     generic map(N)
     port map(NUM,DEN,SNUM1,SDEN1,TQ);
UNA:  block_generic
      generic map (M-1, N+1)
      port map (SNUM1, SDEN1, DIFF, DIV_OUT, TTQ);
        --Carries out remaining rows
      lpp1: for i in M-2 downto 0 generate       --Produces an M bit
 Quotient
       Q(i) <= TTQ(i);
        end generate;
        Q(M-1)<= TQ;

end structural;
----------------------------------------------------------------
--Copyright 2001 by Howard University All rights reserved.
--
--Design: Generic Register
--Name: Clay Gloster
--
--Date: May 2001
--Description:  Generic Register
--Width equal to one
--Does not have parallel
----------------------------------------------------------------
library IEEE;
use IEEE.STD_Logic_1164.all;
entity REG_GENERIC is
generic (Width : integer);
port (Datain : in std_logic_vector(Width-1 downto 0);
Clk,Reset : in std_logic;
Q: out std_logic_vector(Width-1 downto 0) );
end REG_GENERIC;
architecture RTL of REG_GENERIC is
begin
start: process(Clk)
variable i: integer;
begin
if (Clk'EVENT and Clk = '1') then
 if (Reset = '1') then  -- Changed Reset to Active High
  for i in 0 to Width-1 loop
   Q(i) <= '0';
  end loop;
 else
 Q <= Datain;
 end if;
end if;
end process;
end RTL;
----------------------------------------------------------------
--Copyright 2001 by Howard University All rights reserved.
--
--Design: 2 Dimensional Shift Register
--Name: Clay Gloster
--
--Date: May 2001
--Description:
--
--
----------------------------------------------------------------
library IEEE;
use IEEE.STD_Logic_1164.all;
```

```vhdl
entity register2d is
generic (Height, Width : integer);
port (Datain : in std_logic_vector(Width-1 downto 0);
Clk,Reset : in std_logic;
Q: out std_logic_vector(Width-1 downto 0) );
end register2d;
architecture structural of register2d is
component REG_GENERIC
  generic (Width : integer);
  port (Datain : in std_logic_vector(Width-1 downto 0);
        Clk,Reset : in std_logic;
        Q: out std_logic_vector(Width-1 downto 0) );
end component;


type anarray is array ( 0 to Height) of std_logic_vector(Width-1 downto 0);
signal REGIO : anarray;
begin
 lpp1: for i in  0 to Height-1 generate
 I0:   REG_GENERIC generic map (Width)
       port map ( REGIO(i), Clk, Reset, REGIO(i+1) );
        end generate;
 REGIO(0) <= Datain;
 Q <= REGIO(Height);

end structural;
---------------------------------------------------------------



library IEEE;
use IEEE.std_logic_1164.all;
--Pipelines the divider
entity pipelined_divider is
generic (M : integer := 30;
         N : integer := 24;
                 P: integer  := 15);
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     Reset,Clk : in std_logic;
     Q: out std_logic_vector(M-1 downto 0) );
end pipelined_divider;

architecture structural of pipelined_divider is
---  K = M/P                     (defines K)
component generic_divider
generic (M, N: integer);
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     DIFF: out std_logic_vector(N downto 0);
     DIV_OUT: out std_logic_vector(N downto 0);
     Q: out std_logic_vector(M-1 downto 0) );
end component;


component block_generic
generic (M, N: integer);
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     DIFF: out std_logic_vector(N-1 downto 0);
     DIV_OUT: out std_logic_vector(N-1 downto 0);
     Q: out std_logic_vector(M-1 downto 0));
end component;


component reg_generic
generic (Width : integer);
port (Datain : in std_logic_vector(Width-1 downto 0);
Clk, Reset : in std_logic;
Q: out std_logic_vector(Width-1 downto 0) );
end component;


component register2d
generic (height, width: integer);
port (Datain : in std_logic_vector(Width-1 downto 0);
      Clk, Reset : in std_logic;
      Q: out std_logic_vector(Width-1 downto 0) );
end component;


type anothersignal2d is array (P downto 1) of std_logic_vector((M/P)
 -1 downto 0);
        signal TQ,TTQ: anothersignal2d;
        signal SDIFF, SDIV_OUT: std_logic_vector  (N downto 0);
type signal2d is array (P-1 downto 0) of std_logic_vector (N downto
0);
signal SNUM,SDEN, SDIFF1, SDIV_OUT1: signal2d;



      begin
        UO: generic_divider
        generic map (M/P, N)
        port map (NUM, DEN, SDIFF, SDIV_OUT, TQ(P));


        U1: reg_generic
        generic map (N+1)
        port map (SDIFF, Clk, Reset, SNUM(P-1));


        U2: reg_generic
        generic map (N+1)
        port map (SDIV_OUT, Clk, Reset, SDEN(P-1));


        U3: register2d
        generic map (P,M/P)
        port map (TQ(P), Clk, Reset, TTQ(P));


        lpp1: for i in P-1 downto 1 generate


        U4: block_generic generic map(M/P, N+1)
        port map (SNUM(i), SDEN(i), SDIFF1(i), SDIV_OUT1(i), TQ(i));


        U5: reg_generic
        generic map (N+1)
        port map(SDIFF1(i), Clk, Reset, SNUM(i-1));


        U6: reg_generic
        generic map (N+1)
        port map(SDIV_OUT1(i), Clk, Reset, SDEN(i-1));



        U7: register2d
        generic map (i,M/P)
        port map(TQ(i), Clk, Reset, TTQ(i)); --Do the clock and rese
t



        end generate;



lpp3: for i in P-1 downto 0 generate
lpp2: for j in (M/P)-1 downto 0 generate
 Q( ((M/P)*i)+j )<= TTQ(i+1)(j);
end generate;
end generate;
-- Q(0) <= TTQ(1)(0);
end structural;






















-------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
-- Check for Divide by Zero Error Unit In the Floating Point Divider
-- Name:    Clay Gloster and Rosie Dickens
--
-- Date:    May 2001
-- Description:
--
-------------------------------------------------------------


library IEEE;
use IEEE.STD_Logic_1164.all;

entity StageDBZ is
```

```vhdl
port (Datain : in std_logic_vector(31 downto 0);
Clk,Reset : in std_logic;
Q: out std_logic);
end StageDBZ;

architecture RTL of StageDBZ is
begin
start: process(Clk)

begin
if (Clk'EVENT and Clk = '1') then
 if (Reset = '1') then  -- Changed Reset to Active High
    Q <= '0';
 else
    if ( Datain = "00000000000000000000000000000000") then
    Q <= '1';
    else
    Q <= '0';
    end if;
 end if;
end if;
end process;
end RTL;

library IEEE;
use IEEE.STD_Logic_1164.all;
entity dividebyzero is
port (Datain : in std_logic_vector(31 downto 0);
Clk,Reset : in std_logic;
Q: out std_logic );
end dividebyzero;
architecture structural of dividebyzero is
component register2d
generic (Height, Width : integer);
port (Datain : in std_logic_vector(Width-1 downto 0);
Clk,Reset : in std_logic;
Q: out std_logic_vector(Width-1 downto 0) );
end component;
component StageDBZ
port (Datain : in std_logic_vector(31 downto 0);
Clk,Reset : in std_logic;
Q: out std_logic);
end component;


signal Serror : std_logic;
signal VSerror,VSQ : std_logic_vector(0 downto 0);
begin

 I0:   StageDBZ  port map (Datain,Clk,Reset,Serror);
 I1:   register2d generic map(6,1)
       port map (VSerror,Clk,Reset,VSQ);
VSerror(0) <= Serror;
Q <= VSQ(0);

end structural;
```

```



-----------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
-- Subtract Exponents Floating Point Divider
-- Name:    Clay Gloster
--
-- Date:    May 2001
-- Description:
--
-----------------------------------------------------------------


Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned."+";
use IEEE.std_logic_unsigned."*";
use IEEE.std_logic_unsigned."-";
use IEEE.std_logic_unsigned.SHL;
use IEEE.std_logic_arith.CONV_STD_LOGIC_VECTOR;



ENTITY SubExp IS
        PORT(clk        : in std_logic;
                reset: in std_logic;
                EnR     : in std_logic;
                EnL     : in std_logic;
                Exp_A: in std_logic_vector(7 downto 0);
                Exp_B: in std_logic_vector(7 downto 0);
                SignA: in std_logic;
                SignB: in std_logic;
                Exp     : out std_logic_vector(7 downto 0);
                Sign : out std_logic;
                RunO    : out std_logic);
END SubExp;


ARCHITECTURE RTL OF SubExp IS
BEGIN
        PROCESS (CLK,reset)
                variable ExpAPrime : std_logic_vector(7 downto 0);
        BEGIN
                IF (clk = '1' and clk'event) THEN
                        IF (reset = '1') THEN
                                Exp <= (Others=> '0');
                                Sign <= '0';
                                RunO <= '0';
                        ELSE
                                RunO <= EnR and EnL;
                                IF (EnR = '1' and EnL = '1') THEN
                                        ExpAPrime := Exp_A - Exp_B;
                                        Exp <= "01111111" + ExpAPrim
e;-- Need to add 127
                                        Sign <= SignA XOR SignB;
                                END IF; -- END IF Run= '1'
                        END IF; -- END IF reset = 0
                END IF; -- END IF clk = '1' and clk'event
        END PROCESS;
END RTL;

```

```vhdl

----------------------------------------------------------------
-- Copyright 2000 by Howard University All rights reserved.
--
-- Design: 9 bit register
-- Name:    Clay Gloster
--
-- Date:    April 2000
-- Description:
--  Pipelined.
----------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.STD_Logic_1164.all;


ENTITY Divreg10 IS
        PORT(Datain     : IN std_logic_vector(8 DOWNTO 0);
                Ck              : IN std_logic;
                Reset   : IN std_logic;
                Run             : IN std_logic;
                Q               : OUT std_logic_vector(8 DOWNTO 0);
                RunO            : OUT std_logic);
END Divreg10;


ARCHITECTURE RTL OF Divreg10 IS
        SIGNAL D : std_logic_vector(8 DOWNTO 0);
BEGIN
        PROCESS
        BEGIN
                WAIT UNTIL ((Ck'EVENT) AND (Ck= '1'));
                IF (Reset = '1') THEN           -- Changed Reset to Active High
                        D <= "000000000";
                        RunO <= '0';
                ELSE
                        RunO <= Run;
                        IF (Run = '1') THEN
                                D <= Datain;
                        ELSE
                                D <= D;
                        END IF;
                END IF ;
        END process;
        Q <= D;
END RTL;


----------------------------------------------------------------
-- Copyright 2001 by Howard University All rights reserved.
--
-- Design: Floating Point Adder
-- Name:    Ibrahim Sahin and Clay Gloster
--
-- Date:    June 2001
-- Description:
--  Pipelined.
----------------------------------------------------------------


```

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned."+";
use IEEE.std_logic_unsigned."-";
use IEEE.std_logic_unsigned.">";
use IEEE.std_logic_unsigned.SHL;
use IEEE.std_logic_arith.CONV_STD_LOGIC_VECTOR;



ENTITY DivNorm1 IS
         PORT(clk                : IN std_logic;
                reset   : IN std_logic;
                Run             : IN std_logic;
                UMan            : IN std_logic_vector (29 Downto 0)
;
                UExp            : IN std_logic_vector ( 8 DOWNTO 0)
;
                ZError      : IN std_logic;
                Result  : OUT std_logic_vector (31 DOWNTO 0);
                RunO            : OUT std_logic);
END DivNorm1;


ARCHITECTURE RTL OF DivNorm1 IS
BEGIN

        PROCESS (CLK,reset)
                variable ExpPrime : std_logic_vector(7 downto 0);
        variable TMan : std_logic_vector(22 downto 0);
        BEGIN
                IF (clk = '1' and clk'event) THEN
                        IF (reset = '1') THEN
                                Result <= (OTHERS  => '0');
                RunO <= '0';
                        ELSE

                        RunO <= Run;

                        IF ( (Run = '1') and (ZError = '0') and (UMa
n(29) = '1') ) THEN
                            Result(31) <= UExp(8);
                -- No need to adjust exponents
                            Result(30 DOWNTO 23) <= UExp(7 DOWNTO 0)
;
                            TMan := Uman(28 DOWNTO 6);
                IF ( (UMan(6) = '1') AND (UMan(5) = '1') ) THEN
                  TMan := Tman + 1;
                END IF;
            Result(22 DOWNTO 0) <= TMan;
            END IF;
            IF ( (Run = '1') and (ZError = '0') and (UMan(29) = '0')
 ) THEN
                            Result(31) <= UExp(8);
                -- We need to adjust exponents
                ExpPrime := UExp(7 DOWNTO 0);
                            Result(30 DOWNTO 23) <= ExpPrime - "0000
0001"; -- Subtract one
                TMan := Uman(27 DOWNTO 5);
                IF ( (UMan(5) = '1') AND (UMan(4) = '1') ) THEN
                  TMan := Tman + 1;
                END IF;
                Result(22 DOWNTO 0) <= TMan;
                END IF;

                IF ( (Run = '1') and (ZError = '1') ) THEN
                    Result <= "11111111111111111111111111111111"; -- Div
ide By Zero Error
                END IF;
            END IF; -- End if reset = 1

            END IF;

        END PROCESS;

END RTL;




-----------------------------------------------------------------
-- Copyright 2000 by Howard University All rights reserved.
--
-- Design:      Floating Point Divider Core
-- Name:                Clay Gloster
--
-- Date:    May 2001
-- Description:
-- Pipelined.
-----------------------------------------------------------------


Library IEEE;
use IEEE.std_logic_1164.all;


ENTITY FpDivCore_16 IS
        PORT(clk                : IN std_logic;
                reset   : IN std_logic;
                EnR             : IN std_logic;
                EnL             : IN std_logic;
                Op_A            : IN std_logic_vector(31 DOWNTO 0);
                Op_B            : IN std_logic_vector(31 DOWNTO 0);
                Op_Q            : OUT std_logic_vector(31 DOWNTO 0);
                EnN             : OUT std_logic);
END FpDivCore_16;


ARCHITECTURE STRUCTURAL OF FpDivCore_16 IS


COMPONENT SubExp
        PORT(clk        : in std_logic;
                reset: in std_logic;
                EnR     : in std_logic;
                EnL     : in std_logic;
                Exp_A: in std_logic_vector(7 downto 0);
                Exp_B: in std_logic_vector(7 downto 0);
                SignA: in std_logic;
                SignB: in std_logic;
                Exp     : out std_logic_vector(7 downto 0);
                Sign : out std_logic;
                RunO     : out std_logic);
END COMPONENT;


COMPONENT Divreg10
        PORT(Datain     : IN std_logic_vector(8 DOWNTO 0);
                Ck              : IN std_logic;
                Reset   : IN std_logic;
                Run             : IN std_logic;
                Q               : OUT std_logic_vector(8 DOWNTO 0);
                RunO            : OUT std_logic);
END COMPONENT;


COMPONENT pipelined_divider
        generic (M : integer := 28;
         N : integer := 24;
                 P: integer  := 7);
port(NUM: in std_logic_vector(N-1 downto 0);
     DEN: in std_logic_vector(N-1 downto 0);
     Reset,Clk : in std_logic;
     Q: out std_logic_vector(M-1 downto 0) );
END COMPONENT;



COMPONENT DivNorm1
        PORT(clk                :  IN std_logic;
                reset   :  IN std_logic;
                Run             :  IN std_logic;
                UMan            :  IN std_logic_vector (29 Downto 0)
;
                UExp            :  IN std_logic_vector ( 8 DOWNTO 0)
;
                ZError      :  IN std_logic;
                Result  : OUT std_logic_vector (31 DOWNTO 0);
                RunO            : OUT std_logic);
END COMPONENT;


COMPONENT RegStage8
        PORT(Clk         : IN std_logic;
                Reset   : IN std_logic;
                Run             : IN std_logic;
                Datain  : IN std_logic_vector(31 DOWNTO 0);
                Q               : OUT std_logic_vector(31 DOWNTO 0);
                RunO            : OUT std_logic);
END COMPONENT;


COMPONENT DivideByZero
        PORT (Datain : in std_logic_vector(31 downto 0);
            Clk,Reset : in std_logic;
                Q: out std_logic );
END COMPONENT;
```

```vhdl
SIGNAL Exp1_2    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp2_3    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp3_4    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp4_5    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp5_6    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp6_7    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp7_8    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp8_9    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp9_10    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp10_11    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp11_12    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp12_13    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp13_14    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp14_15    : std_logic_vector (8 DOWNTO 0);
SIGNAL Exp15_16    : std_logic_vector (8 DOWNTO 0);



SIGNAL Run01_2   : std_logic;
SIGNAL Run02_3   : std_logic;
SIGNAL Run03_4   : std_logic;
SIGNAL Run04_5   : std_logic;
SIGNAL Run05_6   : std_logic;
SIGNAL Run06_7   : std_logic;
SIGNAL Run07_8   : std_logic;
SIGNAL Run08_9   : std_logic;
SIGNAL Run09_10   : std_logic;
SIGNAL Run010_11   : std_logic;
SIGNAL Run011_12   : std_logic;
SIGNAL Run012_13   : std_logic;
SIGNAL Run013_14   : std_logic;
SIGNAL Run014_15   : std_logic;
SIGNAL Run015_16   : std_logic;


SIGNAL Result15_16: std_logic_vector (29 DOWNTO 0);



SIGNAL ZError   : std_logic;
SIGNAL Op_A01,Op_B01 :std_logic_vector(23 downto 0);







BEGIN


        Inst_Exp: SubExp PORT MAP(clk,reset,EnR,EnL,Op_A(30 DOWNTO 23),Op_B(30 DOWNTO 23),
        Op_A(31),Op_B(31),Exp1_2(7 DOWNTO 0),Exp1_2(8),Run01_2);
        Ints_R1 : Divreg10 PORT MAP (Exp1_2, clk, reset, Run01_2, Exp2_3, Run02_3);
        Ints_R2 : Divreg10 PORT MAP (Exp2_3, clk, reset, Run02_3, Exp3_4, Run03_4);
        Ints_R3 : Divreg10 PORT MAP (Exp3_4, clk, reset, Run03_4, Exp4_5, Run04_5);
        Ints_R4 : Divreg10 PORT MAP (Exp4_5, clk, reset, Run04_5, Exp5_6, Run05_6);
        Ints_R5 : Divreg10 PORT MAP (Exp5_6, clk, reset, Run05_6, Ex
p6_7, RunO6_7);
        Ints_R6 : Divreg10 PORT MAP (Exp6_7, clk, reset, RunO6_7, Exp7_8, RunO7_8);
    Ints_R7 : Divreg10 PORT MAP (Exp7_8, clk, reset, RunO7_8, Exp8_9, RunO8_9);

    Ints_R8 : Divreg10 PORT MAP (Exp8_9, clk, reset, RunO8_9, Exp9_10, RunO9_10);
        Ints_R9 : Divreg10 PORT MAP (Exp9_10, clk, reset, RunO9_10, Exp10_11, RunO10_11);
        Ints_R10 : Divreg10 PORT MAP (Exp10_11, clk, reset, RunO10_11, Exp11_12, RunO11_12);
        Ints_R11 : Divreg10 PORT MAP (Exp11_12, clk, reset, RunO11_12, Exp12_13, RunO12_13);
        Ints_R12 : Divreg10 PORT MAP (Exp12_13, clk, reset, RunO12_13, Exp13_14, RunO13_14);
        Ints_R13 : Divreg10 PORT MAP (Exp13_14, clk, reset, RunO13_14, Exp14_15, RunO14_15);
    Ints_R14 : Divreg10 PORT MAP (Exp14_15, clk, reset, RunO14_15, Exp15_16, RunO15_16);

        Op_A01(22 DOWNTO 0) <=Op_A(22 DOWNTO 0);
        Op_A01(23) <= '1';
        Op_B01(22 DOWNTO 0) <=Op_B(22 DOWNTO 0);
        Op_B01(23) <= '1';



        Inst_Div: pipelined_divider
        generic map (30,24,15)
        port map (Op_A01,Op_B01,reset,clk,Result15_16);



        Inst_N1 : DivNorm1 PORT MAP (clk, reset, RunO15_16, Result15_16, Exp15_16, ZError, Op_Q, EnN);
        Inst_DB : DivideByZero PORT MAP (Op_B,Clk,Reset,Zerror);





END STRUCTURAL;


-----------------------------------------------------------------------------------------------------
--                              Memory Read Operation with initial Value
-----------------------------------------------------------------------------------------------------

-- megafunction wizard: %LPM_ROM%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altsyncram

-- ============================================================
-- File Name: rom.vhd
-- Megafunction Name(s):
--          altsyncram
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 5.1 Build 176 10/26/2005 SJ Full Version
-- ************************************************************


--Copyright (C) 1991-2005 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors.  Please refer to the
--applicable agreement for further details.


LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY rom IS
    PORT
    (
        address     : IN STD_LOGIC_VECTOR (11 DOWNTO 0);
        clock       : IN STD_LOGIC ;
        q       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END rom;


ARCHITECTURE SYN OF rom IS

    SIGNAL sub_wire0    : STD_LOGIC_VECTOR (31 DOWNTO 0);



    COMPONENT altsyncram
    GENERIC (
        address_aclr_a      : STRING;
        init_file       : STRING;
        intended_device_family      : STRING;
        lpm_hint        : STRING;
        lpm_type        : STRING;
        numwords_a      : NATURAL;
        operation_mode      : STRING;
        outdata_aclr_a      : STRING;
        outdata_reg_a       : STRING;
        widthad_a       : NATURAL;
        width_a     : NATURAL;
        width_byteena_a     : NATURAL
    );
    PORT (
            clock0  : IN STD_LOGIC ;
            address_a   : IN STD_LOGIC_VECTOR (11 DOWNTO 0);
```

```
                q_a : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
    END COMPONENT;

BEGIN
    q    <= sub_wire0(31 DOWNTO 0);

    altsyncram_component : altsyncram
    GENERIC MAP (
        address_aclr_a => "NONE",
        init_file => "C:/Documents and Settings/Deepayan/Desktop/deep.mif",
        intended_device_family => "Stratix",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 4096,
        operation_mode => "ROM",
        outdata_aclr_a => "NONE",
        outdata_reg_a => "CLOCK0",
        widthad_a => 12,
        width_a => 32,
        width_byteena_a => 1
    )
    PORT MAP (
        clock0 => clock,
        address_a => address,
        q_a => sub_wire0
    );



END SYN;
```