



*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

---

<b>Chapter 1: Introduction.....</b>	<b>7</b>
1.1 Introduction.....	7
1.2 Background.....	7
1.3 Motivation.....	8
1.4 Thesis Description.....	9
1.5 Program of work.....	10
1.6 Deliverables.....	10
1.7 Thesis Foundation.....	11
1.7.1 Time/Schedule.....	11
1.7.2 Technical Limitations.....	11
1.7.3 Potential Hazards.....	12
1.8 Report Guideline.....	12
<b>Chapter 2: Literature Survey.....</b>	<b>13</b>
2.1 Introduction.....	13
2.2 General Discussion.....	13
2.3 Relevant Past Work.....	14
2.4 Summary.....	16

<b>Chapter 3: Theoretical Discussion.....</b>	<b>17</b>
3.1 Introduction.....	17
3.2 Hu’s Moment Invariant.....	17
3.3 Kohonen Artificial Neural Network.....	19
3.4 K-means Clustering.....	23
3.5 FPGA.....	24
3.6 IEEE 754 floating point format.....	26
3.7 Summary.....	28
<b>Chapter 4: Methodology and Algorithmic development.....</b>	<b>29</b>
4.1 Introduction.....	29
4.2 Moment invariant computation.....	29
4.3 Training of the system.....	32
4.4 Classification Mode.....	34
4.5 General Discussion on VHDL coding.....	35
4.6 Explanation of VHDL Code.....	36
4.7 Discussion on MATLAB Coding of Training Algorithm.....	40
4.8 Summary.....	41
<b>Chapter 5: Discussion.....</b>	<b>42</b>
5.1 Introduction.....	42
5.2 Result and Timing Analysis.....	42
5.2.1 Moment Calculation.....	42
5.2.2 Training of the System.....	43
5.2.3 Classification Mode.....	44
5.3 Analysis & Discussion for Real Time object classification .....	44
5.4 Summary .....	45

<b>Chapter 6: Conclusion and Future Work.....</b>	<b>46</b>
6.1 Discussion.....	46
6.2 Conclusion.....	46
6.2.1 Why FPGA.....	47
6.2.2 Implementation of ANN on FPGA.....	47
6.2.3 Potential Pros and Cons.....	47
6.3 Future Work.....	48
6.3.1 Online Training.....	48
6.3.2 System Integration with Camera Interface.....	49
6.4 Summary .....	49
<b>Reference &amp; Bibliography.....</b>	<b>50</b>

**Appendix A: VHDL Source Code**

**Appendix B: MATLAB Source Code**

**Appendix C: Memory Initialisation file**

**Publication**



# 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 glue logic 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, its 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	05 Jun '06							12 Jun '06						
					S	M	T	W	T	F	S	S	M	T	W	T	F	
1	Background Work	27 days	Mon 05/06/06	Tue 11/07/06	[Red bar]													
2	Background study	4 days	Mon 05/06/06	Thu 08/06/06	[Blue bar: Background study]													
3	Literature review	4 days	Tue 06/06/06	Fri 09/06/06	[Blue bar: Literature review]													
4	Understanding Hu's invariant	4 days	Mon 12/06/06	Thu 15/06/06	[Blue bar: Understanding Hu's invariant]													
5	Understanding ANN	11 days	Wed 14/06/06	Wed 28/06/06	[Blue bar: Understanding ANN]													
6	Understanding Stratix FPGA	7 days	Mon 19/06/06	Tue 27/06/06	[Blue bar: Understanding Stratix FPGA]													
7	Learning VHDL	9 days	Thu 29/06/06	Tue 11/07/06	[Blue bar: Learning VHDL]													
8																		
9	Implementation	24 days	Wed 12/07/06	Mon 14/08/06	[Blue bar: Implementation]													
10	Moment	5 days	Wed 12/07/06	Tue 18/07/06	[Blue bar: Moment]													
11	Hu's invariant	6 days	Wed 19/07/06	Wed 26/07/06	[Blue bar: Hu's invariant]													
12	Neural Network	13 days	Thu 27/07/06	Mon 14/08/06	[Blue bar: Neural Network]													
13																		
14	Testing	7 days	Tue 15/08/06	Wed 23/08/06	[Blue bar: Testing]													
15	Phase-I (Hu's Invariant)	4 days	Tue 15/08/06	Fri 18/08/06	[Blue bar: Phase-I (Hu's Invariant)]													
16	Phase-II (Training of FPGA, ANN)	4 days	Tue 22/08/06	Fri 25/08/06	[Blue bar: Phase-II (Training of FPGA, ANN)]													
17	Phase-III (Testing with unknown obj)	3 days	Mon 28/08/06	Wed 30/08/06	[Blue bar: Phase-III (Testing with unknown obj)]													
18																		
19	Report Writing	14 days	Tue 15/08/06	Fri 01/09/06	[Blue bar: Report Writing]													

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 connected 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:

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

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:

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

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

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

Normally individual moment values do not have the descriptive power to uniquely represent arbitrary shapes, nor do those possess 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.

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

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

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

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

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

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

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

The derivation of these seven invariants and the computation of  $\eta$  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 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.

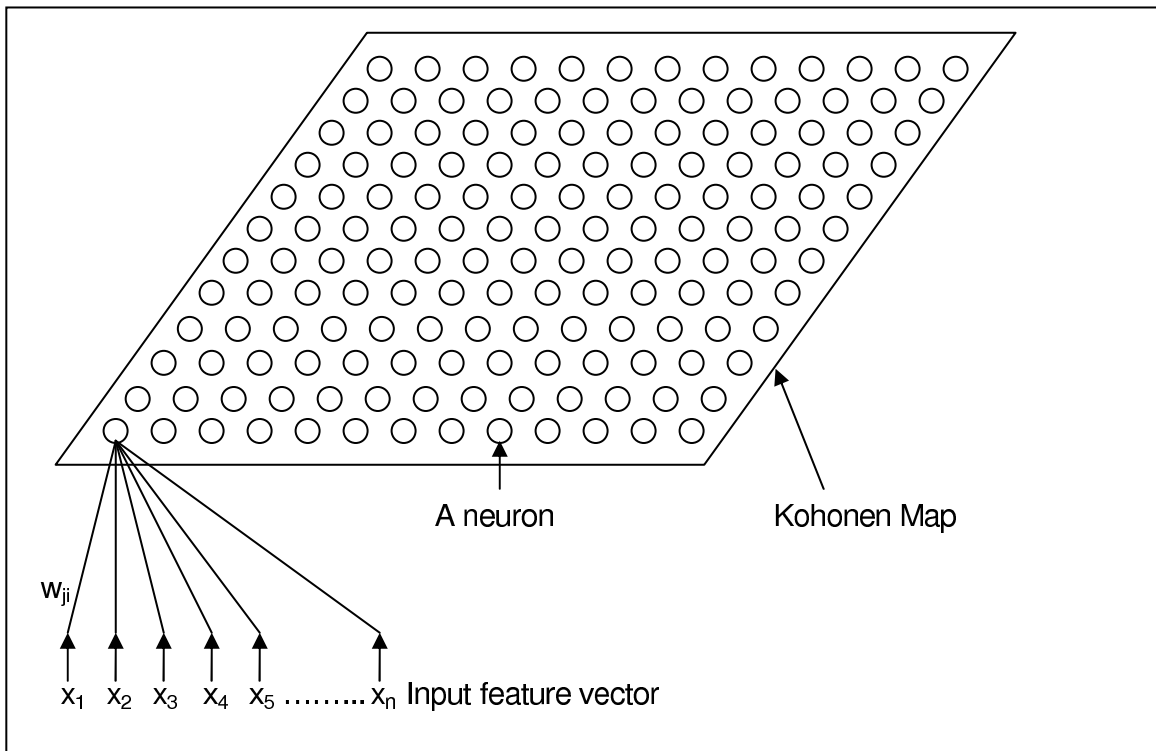


Figure 3.1: A Kohonen network

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.

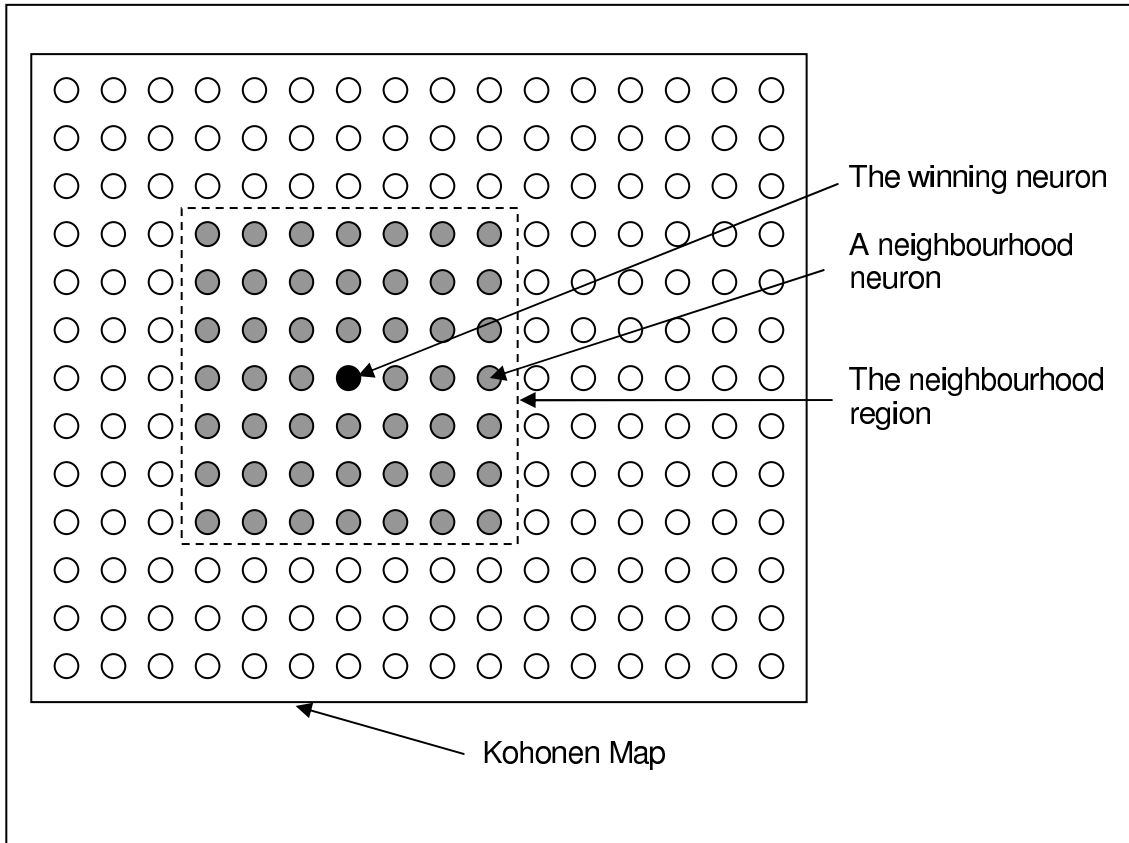


Figure 3.2: A Kohonen network with winning node and neighbourhood region

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  $j^{\text{th}}$  neuron from  $i^{\text{th}}$  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  $j^{\text{th}}$  neuron with each input pattern has  $N$  elements.

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

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.

$$\text{Manhattan distance ("L}_1 \text{ Norm")}: \sum_{i=1}^k |x_i - y_i| \quad (\text{eqn. 3.12})$$

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:

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



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

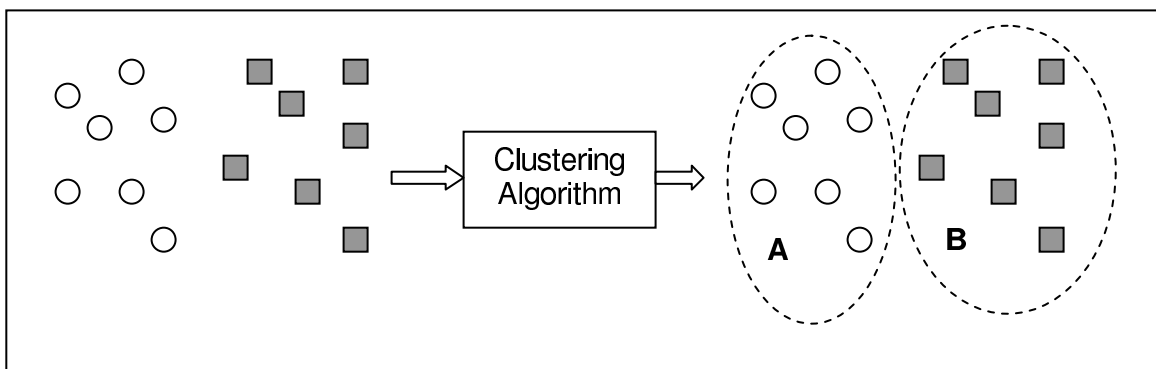


Figure 3.3: The role of clustering algorithm

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 assumed to be in a vector space. It tries to minimize total intra-cluster variance which can be represented by the equation 3.14

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

Where there are  $k$  clusters  $S_i$ ,  $i = 1, 2, \dots, k$  and  $\mu_i$  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.

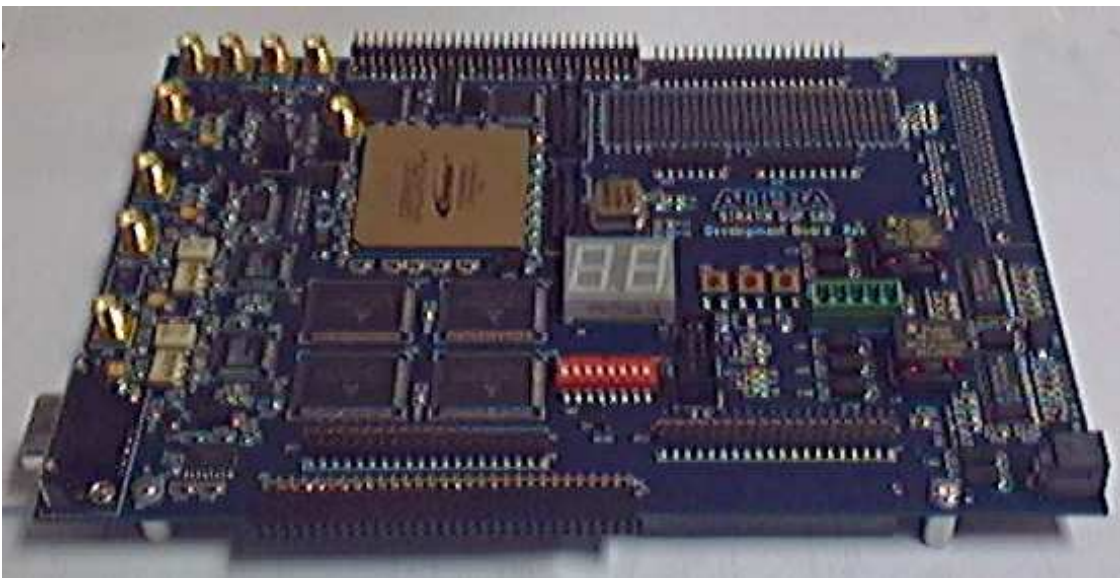


Figure 3.4: the Altera EP1S80 development board

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 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 ( $\geq 43$ -bit, not commonly used) and double-extended precision ( $\geq 79$ -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 field 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.

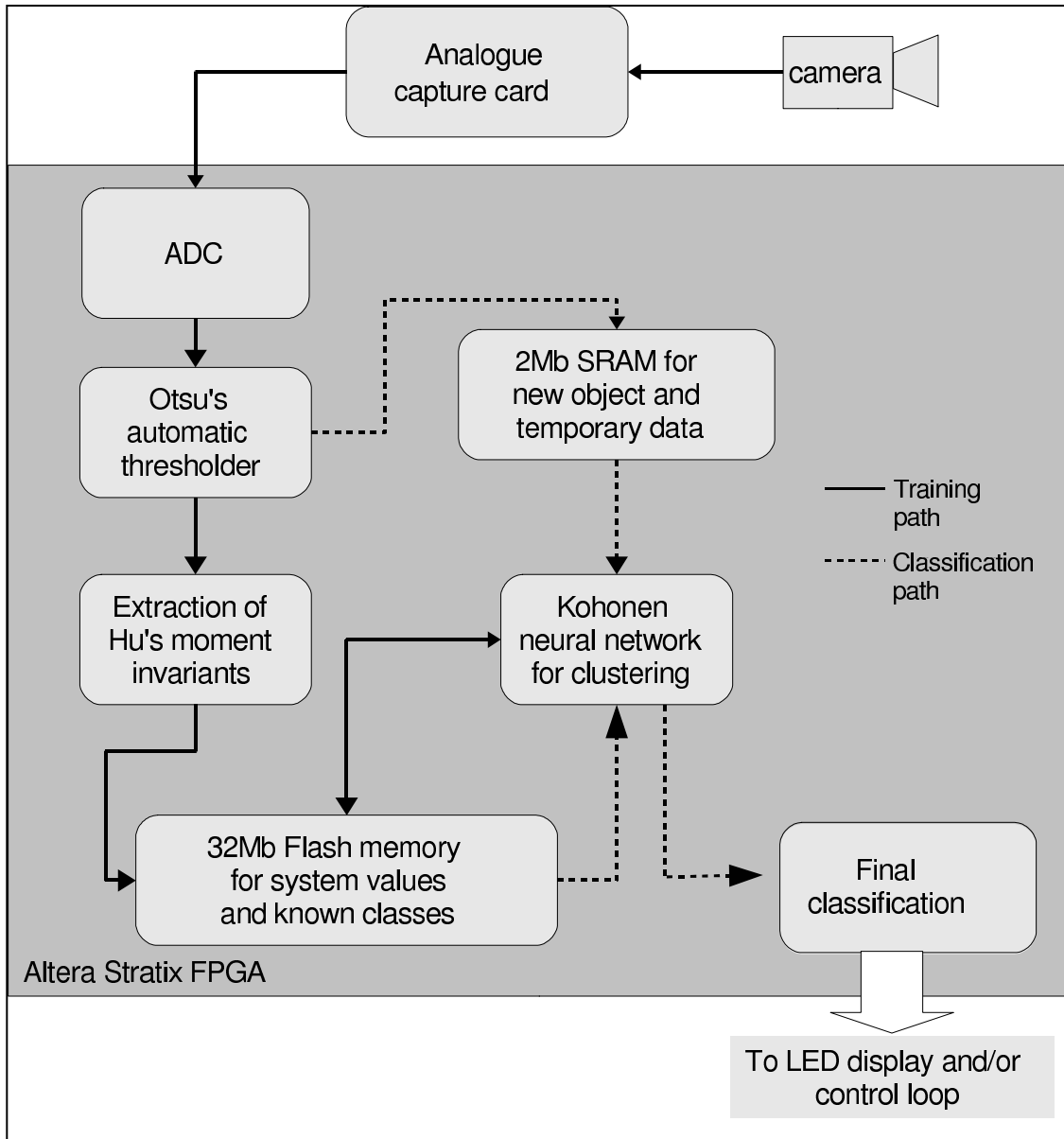


Figure 4.1: Training and classifying objects (the proposed system)

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.

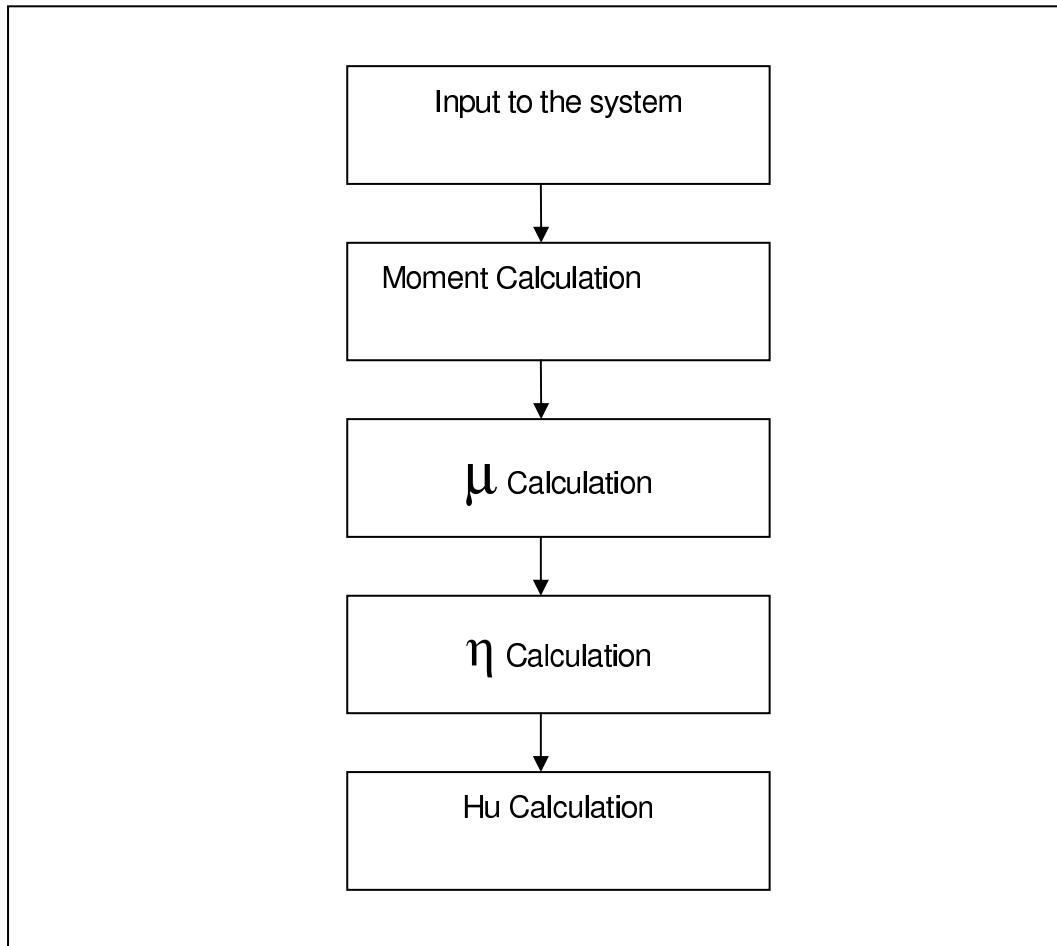


Figure 4.2: Algorithm for moment calculation.

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  $\mu$  parameters (essentially high order moments) are computed in parallel mode. The  $\mu$  values are required to compute  $\eta$  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

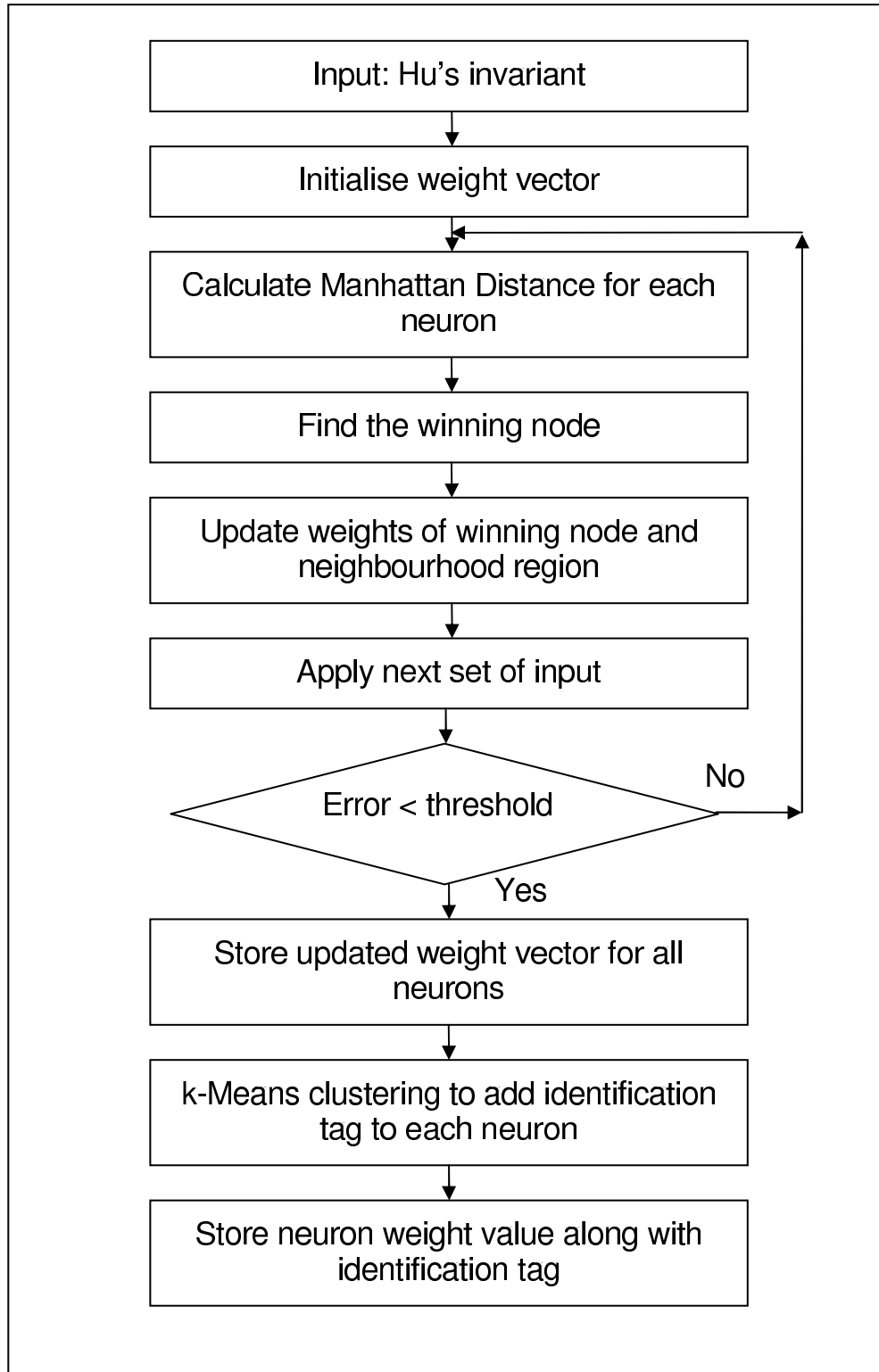


Figure 4.3: Algorithm for Kohonen training and clustering

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:

$$\text{Manhattan distance ("L}_1\text{ Norm")}: \sum_{i=1}^k |x_i - y_i| \quad (\text{eqn. 4.1})$$

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:

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

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.

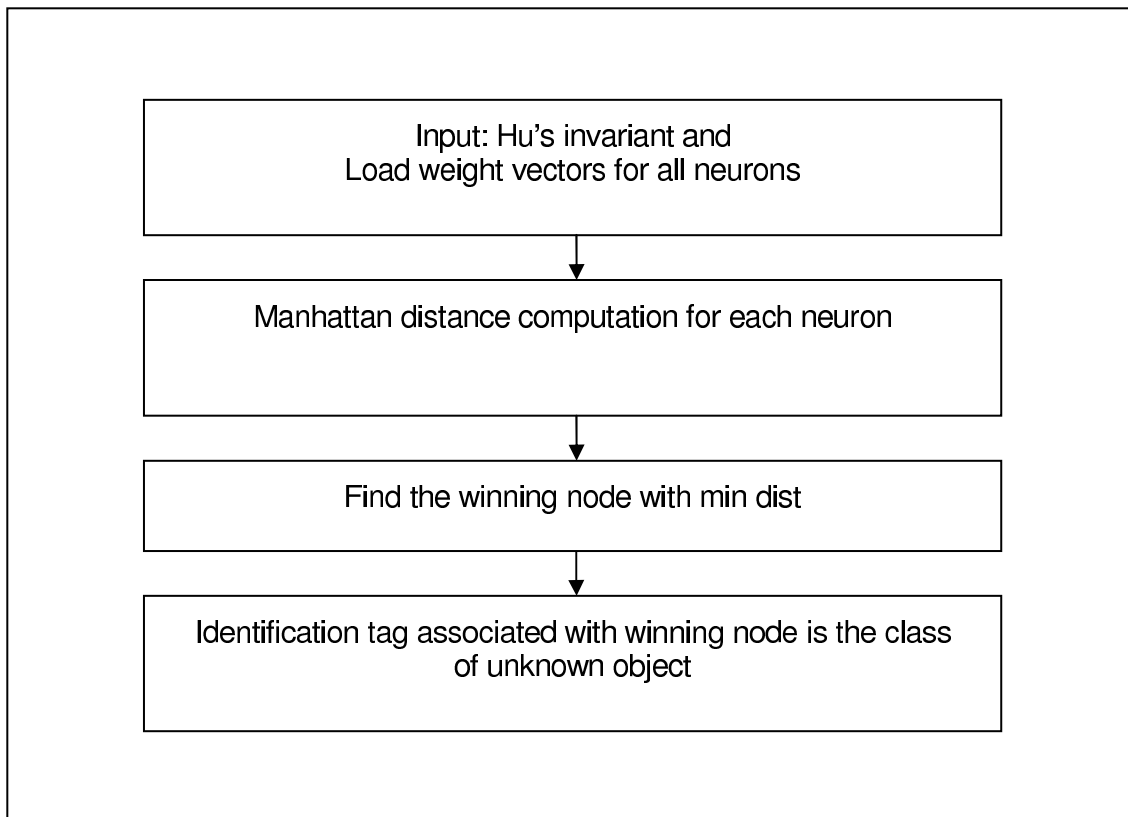


Figure 4.4: Algorithm for classification mode

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;
    end if;
end process;

```

```

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 mantissa : 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";
        mantissa<=int_ip(30 downto 8);

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

    -----
    -----

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

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

```
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

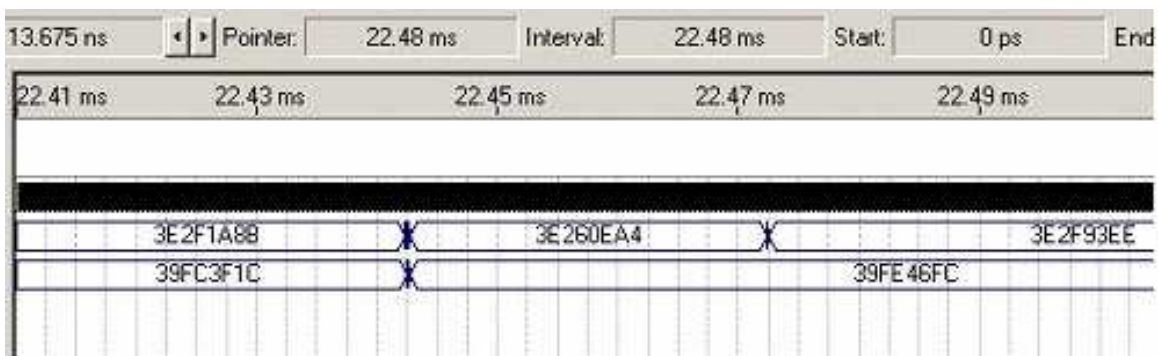


Figure 5.1: Timing analysis of Moment Calculation

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

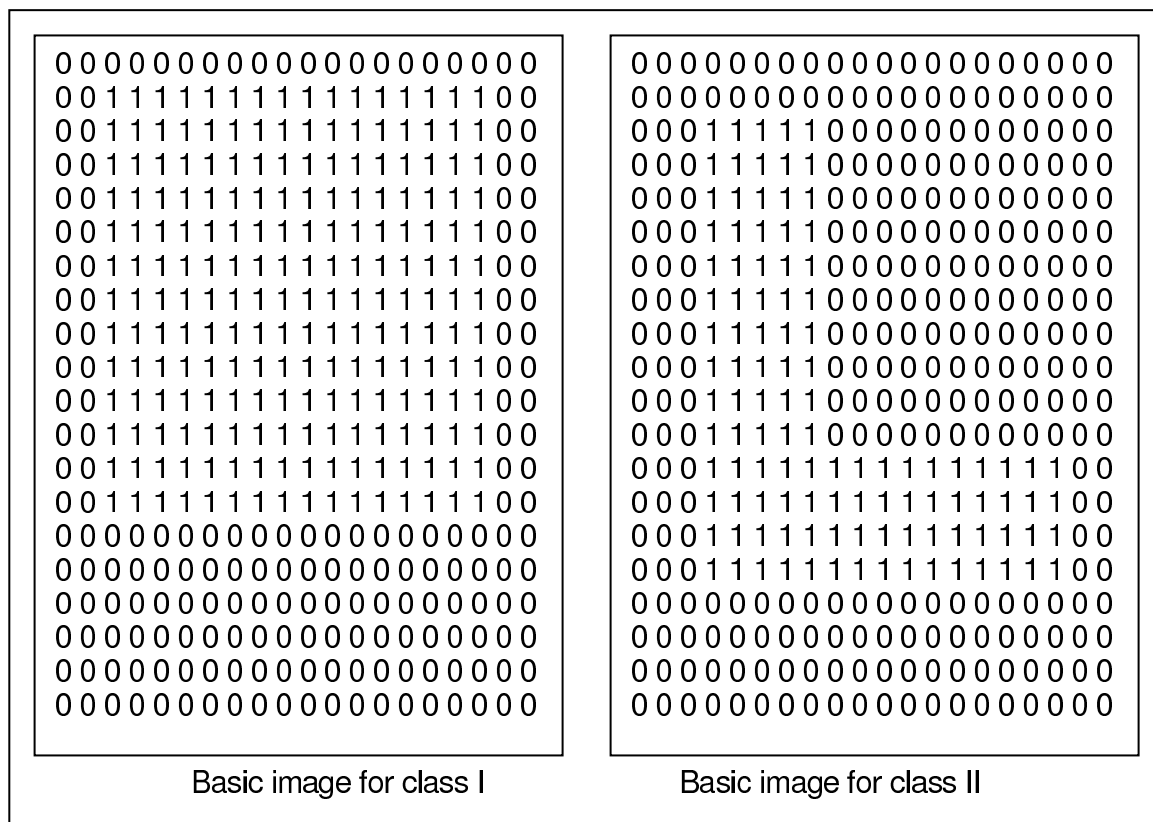


Figure 5.2: Predefined test images for two different classes

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

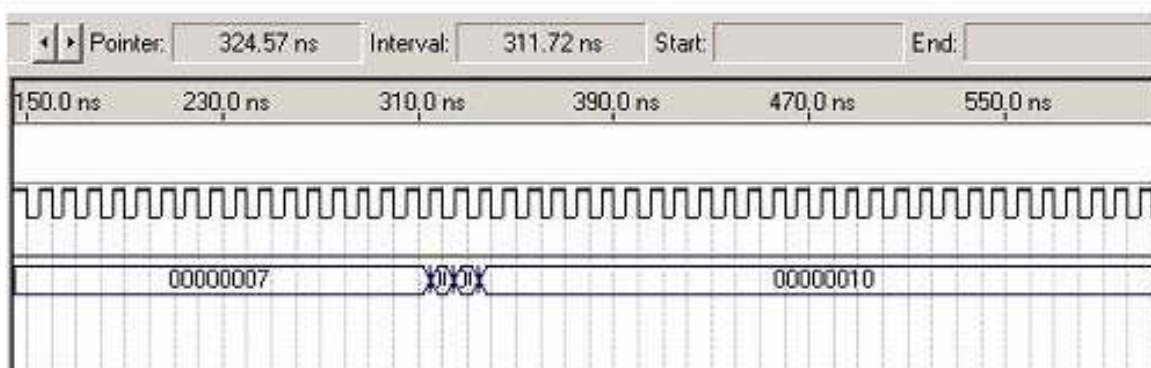


Figure 5.3: Timing analysis of Object Classification mode

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 VHDL, 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](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](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](http://en.wikipedia.org/wiki/K-means_clustering_algorithm) last visited on 14.09.2006
- [16] [http://en.wikipedia.org/wiki/Image\\_moments](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**

```
1 -----
2 --Appendix
3
4 --Appendix A: VHDL Code
5 --           Author: Deepayan Bhowmik
6 --           deepayan.bhowmik@yahoo.com
7 --           MSc. Electronics & IT 2005-06
8 --           Sheffield Hallam University
9 --           Copyrighted
10 -----
11 LIBRARY ieee;
12 USE ieee.std_logic_1164.ALL;
13 USE ieee.std_logic_arith.ALL;
14
15
16 entity user is
17
18     port(   clk :in std_logic;
19           out_test: out STD_LOGIC;
20           test1 : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)
21
22
23         );
24
25
26 end user;
27
28 architecture behaviour of user is
29
30 --type moment_type is array (1 to 25) of bit_vector(15 downto 0);
31
32 type pic_type is array (0 to 15) of std_logic_vector(0 to 15);
33
34 signal count, en_count : integer range 0 to 1000;
35 signal en_signal : std_logic;
36 signal x1,x2,x3 : integer range 0 to 100000;
37 signal y1,y2,y3 : integer range 0 to 100000;
38 signal m11,m12,m13,m14 : integer range 0 to 100000;
39 signal m21,m22,m23 : integer range 0 to 100000;
40 signal m31,m32 : integer range 0 to 100000;
41 signal m41 : integer range 0 to 100000;
42 signal x,y : integer range 0 to 100;
43 signal pic1: pic_type;
44 signal count2,count3 : natural range 0 to 1000;
45
46
47 signal moment11,moment12,moment13,moment14 : STD_LOGIC_VECTOR (31 DOWNTO 0);
48 signal moment21,moment22,moment23 : STD_LOGIC_VECTOR (31 DOWNT0 0);
49 signal moment31,moment32 : STD_LOGIC_VECTOR (31 DOWNT0 0);
50 signal moment41 : STD_LOGIC_VECTOR (31 DOWNT0 0);
51
52 signal xmean,ymean : STD_LOGIC_VECTOR (31 DOWNT0 0);
53 signal mu11,mu31,mu13,mu22,mu41,mu23,mu32,mu14 : STD_LOGIC_VECTOR (31 DOWNT0 0);
54 signal eta31,eta13,eta22,eta41,eta23,eta32,eta14 : STD_LOGIC_VECTOR (31 DOWNT0 0);
55 signal hu1,hu2,hu3,hu4,hu5,hu6,hu7 : STD_LOGIC_VECTOR (31 DOWNT0 0);
56
57 signal mu_mult31,mu_mult13,mu_mult22,xmean_sq,ymean_sq : STD_LOGIC_VECTOR (31 DOWNT0 0);
58 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);
59 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);
60 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);
61 signal mu_mult14_11,mu_mult14_12,mu_add14,mu_mult14_21,mu_mult14_22,
    mu_sub14 : STD_LOGIC_VECTOR (31 DOWNTO 0);
62
63 signal eta_sq : STD_LOGIC_VECTOR (31 DOWNTO 0);
64
65 signal hu_sub2,hu_mult2_1,hu_mult2_2,hu_mult2_3,hu_add2 : STD_LOGIC_
    VECTOR (31 DOWNTO 0);
66 signal hu_add4_1,hu_add4_2,hu_mult4_1,hu_mult4_2 : STD_LOGIC_VECTOR
    (31 DOWNTO 0);
67 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);
68 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);
69 signal hu_sub6_1,hu_mult6_1,hu_mult6_2,hu_mult6_3,hu_mult6_4: STD_LO
    GIC_VECTOR (31 DOWNTO 0);
70 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);
71
72
73 signal dummy_add_nan,dummy_add_overflow,dummy_add_underflow : STD_LO
    GIC ;
74 signal dummy_sub_nan,dummy_sub_overflow,dummy_sub_underflow : STD_LO
    GIC ;
75 signal dummy_mult_nan,dummy_mult_overflow,dummy_mult_underflow : STD
    _LOGIC ;
76
77 signal EnR,EnL,reset,EnN: STD_LOGIC;
78
79 -----
80 -- Signals for user detection mode
81 -----
82
83 type manhatan_distance is array (0 to 7) of std_logic_vector(0 to 31
    );
84 type image_type is array (0 to 7) of std_logic_vector(0 to 11);
85
86 signal distance: manhatan_distance;
87 signal im_type: image_type;
88
89 signal add1,add2,add3,add4,add5,add6 : STD_LOGIC_VECTOR(11 downto 0)
    ;
90 signal input1,input2,input3,input4,input5,input6,input7 : STD_LOGIC_
    VECTOR(31 DOWNTO 0);
91 signal mask,temp : STD_LOGIC_VECTOR(31 DOWNTO 0);
92 signal index,check: integer range 0 to 7;
93
94
95
96 --Signal for neuron 1
97 signal addr1_1,addr1_2,addr1_3,addr1_4,addr1_5,addr1_6,addr1_7 : STD
    _LOGIC_VECTOR(11 DOWNTO 0);
98 signal mem1_1,mem1_2,mem1_3,mem1_4,mem1_5,mem1_6,mem1_7,mem1_8 : STD
    _LOGIC_VECTOR(31 DOWNTO 0);
99 signal sub1_1,sub1_2,sub1_3,sub1_4,sub1_5,sub1_6,sub1_7 : STD_LOGIC_
    VECTOR(31 DOWNTO 0);
100 signal abs1_1,abs1_2,abs1_3,abs1_4,abs1_5,abs1_6,abs1_7 : STD_LOGIC_
    VECTOR(31 DOWNTO 0);

```

```
101 signal add1_12,add1_34,add1_56,add1_27,add1_46 : STD_LOGIC_VECTOR(31
    DOWNT0 0);
102
103 --Signal for neuron 2
104 signal addr2_1,addr2_2,addr2_3,addr2_4,addr2_5,addr2_6,addr2_7 : STD
    _LOGIC_VECTOR(11 DOWNT0 0);
105 signal mem2_1,mem2_2,mem2_3,mem2_4,mem2_5,mem2_6,mem2_7,mem2_8 : STD
    _LOGIC_VECTOR(31 DOWNT0 0);
106 signal sub2_1,sub2_2,sub2_3,sub2_4,sub2_5,sub2_6,sub2_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
107 signal abs2_1,abs2_2,abs2_3,abs2_4,abs2_5,abs2_6,abs2_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
108 signal add2_12,add2_34,add2_56,add2_27,add2_46 : STD_LOGIC_VECTOR(31
    DOWNT0 0);
109
110 --Signal for neuron 3
111 signal addr3_1,addr3_2,addr3_3,addr3_4,addr3_5,addr3_6,addr3_7 : STD
    _LOGIC_VECTOR(11 DOWNT0 0);
112 signal mem3_1,mem3_2,mem3_3,mem3_4,mem3_5,mem3_6,mem3_7,mem3_8 : STD
    _LOGIC_VECTOR(31 DOWNT0 0);
113 signal sub3_1,sub3_2,sub3_3,sub3_4,sub3_5,sub3_6,sub3_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
114 signal abs3_1,abs3_2,abs3_3,abs3_4,abs3_5,abs3_6,abs3_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
115 signal add3_12,add3_34,add3_56,add3_27,add3_46 : STD_LOGIC_VECTOR(31
    DOWNT0 0);
116
117 --Signal for neuron 4
118 signal addr4_1,addr4_2,addr4_3,addr4_4,addr4_5,addr4_6,addr4_7 : STD
    _LOGIC_VECTOR(11 DOWNT0 0);
119 signal mem4_1,mem4_2,mem4_3,mem4_4,mem4_5,mem4_6,mem4_7,mem4_8 : STD
    _LOGIC_VECTOR(31 DOWNT0 0);
120 signal sub4_1,sub4_2,sub4_3,sub4_4,sub4_5,sub4_6,sub4_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
121 signal abs4_1,abs4_2,abs4_3,abs4_4,abs4_5,abs4_6,abs4_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
122 signal add4_12,add4_34,add4_56,add4_27,add4_46 : STD_LOGIC_VECTOR(31
    DOWNT0 0);
123
124 --Signal for neuron 5
125 signal addr5_1,addr5_2,addr5_3,addr5_4,addr5_5,addr5_6,addr5_7 : STD
    _LOGIC_VECTOR(11 DOWNT0 0);
126 signal mem5_1,mem5_2,mem5_3,mem5_4,mem5_5,mem5_6,mem5_7,mem5_8 : STD
    _LOGIC_VECTOR(31 DOWNT0 0);
127 signal sub5_1,sub5_2,sub5_3,sub5_4,sub5_5,sub5_6,sub5_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
128 signal abs5_1,abs5_2,abs5_3,abs5_4,abs5_5,abs5_6,abs5_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
129 signal add5_12,add5_34,add5_56,add5_27,add5_46 : STD_LOGIC_VECTOR(31
    DOWNT0 0);
130
131 --Signal for neuron 6
132 signal addr6_1,addr6_2,addr6_3,addr6_4,addr6_5,addr6_6,addr6_7 : STD
    _LOGIC_VECTOR(11 DOWNT0 0);
133 signal mem6_1,mem6_2,mem6_3,mem6_4,mem6_5,mem6_6,mem6_7,mem6_8 : STD
    _LOGIC_VECTOR(31 DOWNT0 0);
134 signal sub6_1,sub6_2,sub6_3,sub6_4,sub6_5,sub6_6,sub6_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
135 signal abs6_1,abs6_2,abs6_3,abs6_4,abs6_5,abs6_6,abs6_7 : STD_LOGIC_
    VECTOR(31 DOWNT0 0);
136 signal add6_12,add6_34,add6_56,add6_27,add6_46 : STD_LOGIC_VECTOR(31
    DOWNT0 0);
```

```

137
138 --Signal for neuron 7
139 signal addr7_1,addr7_2,addr7_3,addr7_4,addr7_5,addr7_6,addr7_7 : STD
   _LOGIC_VECTOR(11 DOWNTO 0);
140 signal mem7_1,mem7_2,mem7_3,mem7_4,mem7_5,mem7_6,mem7_7,mem7_8 : STD
   _LOGIC_VECTOR(31 DOWNTO 0);
141 signal sub7_1,sub7_2,sub7_3,sub7_4,sub7_5,sub7_6,sub7_7 : STD_LOGIC_
   VECTOR(31 DOWNTO 0);
142 signal abs7_1,abs7_2,abs7_3,abs7_4,abs7_5,abs7_6,abs7_7 : STD_LOGIC_
   VECTOR(31 DOWNTO 0);
143 signal add7_12,add7_34,add7_56,add7_27,add7_46 : STD_LOGIC_VECTOR(31
   DOWNTO 0);
144
145 --Signal for neuron 8
146 signal addr8_1,addr8_2,addr8_3,addr8_4,addr8_5,addr8_6,addr8_7 : STD
   _LOGIC_VECTOR(11 DOWNTO 0);
147 signal mem8_1,mem8_2,mem8_3,mem8_4,mem8_5,mem8_6,mem8_7,mem8_8 : STD
   _LOGIC_VECTOR(31 DOWNTO 0);
148 signal sub8_1,sub8_2,sub8_3,sub8_4,sub8_5,sub8_6,sub8_7 : STD_LOGIC_
   VECTOR(31 DOWNTO 0);
149 signal abs8_1,abs8_2,abs8_3,abs8_4,abs8_5,abs8_6,abs8_7 : STD_LOGIC_
   VECTOR(31 DOWNTO 0);
150 signal add8_12,add8_34,add8_56,add8_27,add8_46 : STD_LOGIC_VECTOR(31
   DOWNTO 0);
151
152
153
154
155 -----
156 -- END ---          Signals for user detection mode
157 -----
158
159
160
161
162
163
164
165 component rom
166     PORT
167     (
168         address      : IN STD_LOGIC_VECTOR (11 DOWNTO 0);
169         clock         : IN STD_LOGIC ;
170         q              : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
171     );
172 end component;
173
174
175
176 component FpDivCore_16
177     PORT(clk          : IN std_logic;
178         reset        : IN std_logic;
179         EnR          : IN std_logic;
180         EnL          : IN std_logic;
181         Op_A         : IN std_logic_vector(31 DOWNTO 0);
182         Op_B         : IN std_logic_vector(31 DOWNTO 0);
183         Op_Q         : OUT std_logic_vector(31 DOWNTO 0);
184         EnN          : OUT std_logic);
185 end component;
186
187
188 COMPONENT mega_add

```

```

189     PORT
190     (
191         clock      : IN STD_LOGIC ;
192         dataa      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
193         datab     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
194         nan       : OUT STD_LOGIC ;
195         overflow   : OUT STD_LOGIC ;
196         result    : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
197         underflow  : OUT STD_LOGIC
198     );
199
200 END COMPONENT;
201
202
203
204 COMPONENT mega_sub
205     PORT
206     (
207         clock      : IN STD_LOGIC ;
208         dataa      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
209         datab     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
210         nan       : OUT STD_LOGIC ;
211         overflow   : OUT STD_LOGIC ;
212         result    : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
213         underflow  : OUT STD_LOGIC
214     );
215
216 END COMPONENT;
217
218 COMPONENT mega_mult
219     PORT
220     (
221         clock      : IN STD_LOGIC ;
222         dataa      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
223         datab     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
224         nan       : OUT STD_LOGIC ;
225         overflow   : OUT STD_LOGIC ;
226         result    : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
227         underflow  : OUT STD_LOGIC
228     );
229
230 END COMPONENT;
231
232
233 COMPONENT int2fp
234     PORT (input : integer range 0 to 100000;
235
236         fp_op : OUT STD_LOGIC_VECTOR(31 downto 0));
237 END COMPONENT;
238
239
240 --signal moment_parameter: moment_type;
241
242 begin
243
244     pic1(0) <= "0001111111111111";
245     pic1(1) <= "0011111111111111";
246     pic1(2) <= "0010111111111111";
247     pic1(3) <= "0011111111111111";
248     pic1(4) <= "0011111111111111";
249     pic1(5) <= "0011111111111111";
250     pic1(6) <= "0011011111111111";

```

```

251     pic1(7) <=      "0011111111111111";
252     pic1(8) <=      "0111111111111111";
253     pic1(9) <=      "0010111111111111";
254     pic1(10) <=     "0011111111111111";
255     pic1(11) <=     "0011101111111111";
256     pic1(12) <=     "0011111111111111";
257     pic1(13) <=     "0011111101111111";
258     pic1(14) <=     "0011111111111111";
259     pic1(15) <=     "0111111111111111";
260
261     const_2<=X"40000000";
262     const_3<=X"40400000";
263     const_4<=X"40800000";
264
265     --x, y value starts from 0,0
266     -- x<=1;
267     -- y<=1;
268
269     process(clk,en_signal,count,en_count,
270            x1,x2,x3,y1,y2,y3,
271            m11,m12,m13,m14,
272            m21,m22,m23,
273            m31,m32,
274            m41)
275     begin
276         if(clk'EVENT AND clk='1') then --the calculations are based
on clock cycle
277             en_count<=en_count+1;
278
279             if(y<16)then
280                 if(x<16)then
281
282                     if(pic1(y) (x)='1' AND en_count=1)then
283                         en_signal<='1';
284                         count<=count+1;           --1 clock cycle is simulat
ed in 10ns
285
286         --Calculations start here || Pipeline architecture used here
287
288                 y1<=y;
289                 x1<=x;
290
291
292         --First phase data update, assumed clock cycle 4
293
294                 if(count=1)then
295                     y2<=y1*y1;
296                     x2<=x1*x1;
297
298                     m11<=m11+1;
299                     m12<=m12+y1;
300                     m21<=m21+x1;
301                     m22<=m22+x1*y1;
302
303                 end if;
304
305         --2nd phase data update, assumed clock cycle 8
306
307                 if(count=4)then
308                     y3<=y2*y1;
309                     x3<=x2*x1;
310

```

```
311         m13<=m13+y2;
312         m23<=m23+x1*y2;
313         m31<=m31+x2;
314         m32<=m32+x2*y1;
315
316         end if;
317
318 --3rd phase data update, assumed clock cycle 8
319
320         if(count=7)then
321             m14<=m14+y3;
322             m41<=m41+x3;
323
324         end if;
325
326         if(count=10 AND y<16)then
327             count<=0;
328
329             en_signal<='0';
330             en_count<=0;
331
332             x<=x+1;
333
334             if(x>15)then
335                 x<=0;
336                 y<=y+1;
337             end if;
338
339         end if;
340
341     end if;
342 end if;
343 end if;
344
345     if(en_count=2 AND en_signal='0' AND y<16) then
346         x<=x+1;
347         en_count<=0;
348
349         if(x>15)then
350             x<=0;
351             y<=y+1;
352         end if;
353
354     end if;
355
356 end if;
357
358 end process;
359
360 -----
361 -- Integer to Floating point Conversion
362 -----
363
364 conv_fp_1: int2fp
365 port map (m11,moment11);
366
367 conv_fp_2: int2fp
368 port map (m12,moment12);
369
370 conv_fp_3: int2fp
```

```

371     port map (m13,moment13);
372
373     conv_fp_4: int2fp
374     port map (m14,moment14);
375
376     conv_fp_5: int2fp
377     port map (m21,moment21);
378
379     conv_fp_6: int2fp
380     port map (m22,moment22);
381
382     conv_fp_7: int2fp
383     port map (m23,moment23);
384
385     conv_fp_8: int2fp
386     port map (m31,moment31);
387
388     conv_fp_9: int2fp
389     port map (m32,moment32);
390
391     conv_fp_10: int2fp
392     port map (m41,moment41);
393
394
395
396 -----
397 --           Calculation of Mu
398 -----
399
400 -- Calculation of xmean & ymean
401     EnR<='1';
402     EnL<='1';
403     reset<='0';
404
405     fp_div_xmean: FpDivCore_16
406     port map (clk, reset, EnR, EnL, moment21, moment11, xmean, EnN);
407
408     fp_div_ymean: FpDivCore_16
409     port map (clk, reset, EnR, EnL, moment12, moment11, ymean, EnN);
410
411
412 --Calculation of mul1
413     mul1<=moment11;
414
415 --Calculation of mu31
416     mult_mu31: mega_mult
417     port map (clk, xmean, moment21, dummy_mult_nan, dummy_mult_overflow,
mu_mult31, dummy_mult_underflow);
418     sub_mu31: mega_sub
419     port map (clk, moment31, mu_mult31, dummy_sub_nan, dummy_sub_overflow,
mu31, dummy_sub_underflow);
420
421 --Calculation of mul3
422     mult_mu13: mega_mult
423     port map (clk, ymean, moment12, dummy_mult_nan, dummy_mult_overflow,
mu_mult13, dummy_mult_underflow);
424     sub_mu13: mega_sub
425     port map (clk, moment13, mu_mult13, dummy_sub_nan, dummy_sub_overflow,
w, mu13, dummy_sub_underflow);
426
427 --Calculation of mu22
428     mult_mu22: mega_mult

```

```
429     port map (clk, ymean, moment21, dummy_mult_nan, dummy_mult_overflow,
430              mu_mult22, dummy_mult_underflow);
431     sub_mu22: mega_sub
432     port map (clk, moment22, mu_mult22, dummy_sub_nan, dummy_sub_overflow,
433              w, mu22, dummy_sub_underflow);
434
435 --Calculation of mu41
436     sq_xmean: mega_mult
437     port map (clk, xmean, xmean, dummy_mult_nan, dummy_mult_overflow, xmean_sq,
438              dummy_mult_underflow);
439     mult_mu41_21: mega_mult
440     port map (clk, const_2, moment21, dummy_mult_nan, dummy_mult_overflow,
441              w, mu_mult41_21, dummy_mult_underflow);
442     mult_mu41_22: mega_mult
443     port map (clk, xmean_sq, mu_mult41_21, dummy_mult_nan, dummy_mult_overflow,
444              mu_mult41_22, dummy_mult_underflow);
445     mult_mu41_11: mega_mult
446     port map (clk, const_3, moment31, dummy_mult_nan, dummy_mult_overflow,
447              w, mu_mult41_11, dummy_mult_underflow);
448     mult_mu41_12: mega_mult
449     port map (clk, mu_mult41_11, xmean, dummy_mult_nan, dummy_mult_overflow,
450              mu_mult41_12, dummy_mult_underflow);
451     sub_mu41: mega_sub
452     port map (clk, moment41, mu_mult41_12, dummy_sub_nan, dummy_sub_overflow,
453              mu_sub41, dummy_sub_underflow);
454     add_mu41: mega_add
455     port map (clk, mu_sub41, mu_mult41_22, dummy_add_nan, dummy_add_overflow,
456              mu41, dummy_add_underflow);
457
458 --Calculation of mu23
459     sq_ymean: mega_mult
460     port map (clk, ymean, ymean, dummy_mult_nan, dummy_mult_overflow, ymean_sq,
461              dummy_mult_underflow);
462 --mu_mult41_21 came from previous calculation
463     mult_mu23_32: mega_mult
464     port map (clk, ymean_sq, mu_mult41_21, dummy_mult_nan, dummy_mult_overflow,
465              mu_mult23_32, dummy_mult_underflow);
466     mult_mu23_21: mega_mult
467     port map (clk, xmean, moment13, dummy_mult_nan, dummy_mult_overflow,
468              mu_mult23_21, dummy_mult_underflow);
469     mult_mu23_11: mega_mult
470     port map (clk, const_2, moment22, dummy_mult_nan, dummy_mult_overflow,
471              w, mu_mult23_11, dummy_mult_underflow);
472     mult_mu23_12: mega_mult
473     port map (clk, mu_mult23_11, ymean, dummy_mult_nan, dummy_mult_overflow,
474              mu_mult23_12, dummy_mult_underflow);
475     sub_mu23_1: mega_sub
476     port map (clk, moment23, mu_mult23_12, dummy_sub_nan, dummy_sub_overflow,
477              mu_sub23_1, dummy_sub_underflow);
478     sub_mu23_2: mega_sub
479     port map (clk, mu_sub23_1, mu_mult23_21, dummy_sub_nan, dummy_sub_overflow,
480              mu_sub23_2, dummy_sub_underflow);
481     add_mu23: mega_add
482     port map (clk, mu_sub23_2, mu_mult23_32, dummy_add_nan, dummy_add_overflow,
483              mu23, dummy_add_underflow);
484
485 --Calculation of mu32
486     mult_mu32_31: mega_mult
487     port map (clk, const_2, moment12, dummy_mult_nan, dummy_mult_overflow,
488              w, mu_mult32_31, dummy_mult_underflow);
```



```
473     mult_mu32_32: mega_mult
474     port map (clk, xmean_sq, mu_mult32_31, dummy_mult_nan, dummy_mult_ov
erflow, mu_mult32_32, dummy_mult_underflow);
475     mult_mu32_21: mega_mult
476     port map (clk, ymean, moment31, dummy_mult_nan, dummy_mult_overflow,
mu_mult32_21, dummy_mult_underflow);
477 --Calculation from last parameters
478     mult_mu32_12: mega_mult
479     port map (clk, mu_mult23_11, xmean, dummy_mult_nan, dummy_mult_overf
low, mu_mult32_12, dummy_mult_underflow);
480     sub_mu32_1: mega_sub
481     port map (clk, moment32, mu_mult32_12, dummy_sub_nan, dummy_sub_ov
erflow, mu_sub32_1, dummy_sub_underflow);
482     sub_mu32_2: mega_sub
483     port map (clk, mu_sub32_1, mu_mult32_21, dummy_sub_nan, dummy_sub_ov
erflow, mu_sub32_2, dummy_sub_underflow);
484     add_mu32: mega_add
485     port map (clk, mu_sub32_2, mu_mult32_32, dummy_add_nan, dummy_add_ov
erflow, mu32, dummy_add_underflow);
486
487
488 --Calculation of mu14
489     mult_mu14_21: mega_mult
490     port map (clk, const_2, moment12, dummy_mult_nan, dummy_mult_overflo
w, mu_mult14_21, dummy_mult_underflow);
491     mult_mu14_22: mega_mult
492     port map (clk, ymean_sq, mu_mult14_21, dummy_mult_nan, dummy_mult_ov
erflow, mu_mult14_22, dummy_mult_underflow);
493     mult_mu14_11: mega_mult
494     port map (clk, const_3, moment13, dummy_mult_nan, dummy_mult_overflo
w, mu_mult14_11, dummy_mult_underflow);
495     mult_mu14_12: mega_mult
496     port map (clk, mu_mult14_11, ymean, dummy_mult_nan, dummy_mult_overf
low, mu_mult14_12, dummy_mult_underflow);
497     sub_mu14: mega_sub
498     port map (clk, moment14, mu_mult14_12, dummy_sub_nan, dummy_sub_ov
erflow, mu_sub14, dummy_sub_underflow);
499     add_mu14: mega_add
500     port map (clk, mu_sub14, mu_mult14_22, dummy_add_nan, dummy_add_ov
erflow, mu14, dummy_add_underflow);
501
502
503
504 -----
505 --                               Calculation of eta
506 -----
507
508     eta_mult_1: mega_mult
509     port map (clk, mu11, mu11, dummy_mult_nan, dummy_mult_overflow, eta_s
q, dummy_mult_underflow);
510
511 --Calculation of eta31
512     fp_div_eta31: FpDivCore_16
513     port map (clk, reset, EnR, EnL, mu31, eta_sq, eta31, EnN);
514
515 --Calculation of eta13
516     fp_div_eta13: FpDivCore_16
517     port map (clk, reset, EnR, EnL, mu13, eta_sq, eta13, EnN);
518
519 --Calculation of eta22
```

```

520     fp_div_eta22: FpDivCore_16
521     port map (clk, reset, EnR, EnL, mu22, eta_sq, eta22, EnN);
522
523 --Calculation of eta41
524     fp_div_eta41: FpDivCore_16
525     port map (clk, reset, EnR, EnL, mu41, eta_sq, eta41, EnN);
526
527 --Calculation of eta23
528     fp_div_eta23: FpDivCore_16
529     port map (clk, reset, EnR, EnL, mu23, eta_sq, eta23, EnN);
530
531 --Calculation of eta32
532     fp_div_eta32: FpDivCore_16
533     port map (clk, reset, EnR, EnL, mu32, eta_sq, eta32, EnN);
534
535 --Calculation of eta14
536     fp_div_eta14: FpDivCore_16
537     port map (clk, reset, EnR, EnL, mu14, eta_sq, eta14, EnN);
538
539
540
541 -----
542 --                               Calculation of Hu's invariant
543 -----
544
545 --Calculation of hu1
546     add_hu1: mega_add
547     port map (clk, eta31, eta13, dummy_add_nan, dummy_add_overflow, hu1, d
ummy_add_underflow);
548
549 --Calculation of hu2
550     sub_hu2: mega_sub
551     port map (clk, eta31, eta13, dummy_sub_nan, dummy_sub_overflow, hu_su
b2, dummy_sub_underflow);
552     mult_hu2_1: mega_mult
553     port map (clk, hu_sub2, hu_sub2, dummy_mult_nan, dummy_mult_overflow
, hu_mult2_1, dummy_mult_underflow);
554     mult_hu2_2: mega_mult
555     port map (clk, eta22, eta22, dummy_mult_nan, dummy_mult_overflow, hu_
mult2_2, dummy_mult_underflow);
556     mult_hu2_3: mega_mult
557     port map (clk, const_4, hu_mult2_2, dummy_mult_nan, dummy_mult_overf
low, hu_mult2_3, dummy_mult_underflow);
558     add_hu2: mega_add
559     port map (clk, hu_mult2_1, hu_mult2_3, dummy_add_nan, dummy_add_ove
rflow, hu2, dummy_add_underflow);
560
561 --Calculation of hu3
562     mult_hu3_1: mega_mult
563     port map (clk, const_3, eta23, dummy_mult_nan, dummy_mult_overflow, h
u_mult3_1, dummy_mult_underflow);
564     sub_hu3_1: mega_sub
565     port map (clk, eta41, hu_mult3_1, dummy_sub_nan, dummy_sub_overflow,
hu_sub3_1, dummy_sub_underflow);
566     mult_hu3_2: mega_mult
567     port map (clk, hu_sub3_1, hu_sub3_1, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult3_2, dummy_mult_underflow);
568
569     mult_hu3_3: mega_mult
570     port map (clk, const_3, eta32, dummy_mult_nan, dummy_mult_overflow, h

```

```
u_mult3_3, dummy_mult_underflow);
571     sub_hu3_2: mega_sub
572     port map (clk, hu_mult3_3, eta14, dummy_sub_nan, dummy_sub_overflow,
hu_sub3_2, dummy_sub_underflow);
573     mult_hu3_4: mega_mult
574     port map (clk, hu_sub3_2, hu_sub3_2, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult3_4, dummy_mult_underflow);
575
576     add_hu3: mega_add
577     port map (clk, hu_mult3_2, hu_mult3_4, dummy_add_nan, dummy_add_ove
rflow, hu3, dummy_add_underflow);
578
579
580 --Calculation of hu4
581     add_hu4_1: mega_add
582     port map (clk, eta41, eta23, dummy_add_nan, dummy_add_overflow, hu_ad
d4_1, dummy_add_underflow);
583     mult_hu4_1: mega_mult
584     port map (clk, hu_add4_1, hu_add4_1, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult4_1, dummy_mult_underflow);
585     add_hu4_2: mega_add
586     port map (clk, eta32, eta14, dummy_add_nan, dummy_add_overflow, hu_ad
d4_2, dummy_add_underflow);
587     mult_hu4_2: mega_mult
588     port map (clk, hu_add4_2, hu_add4_2, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult4_2, dummy_mult_underflow);
589     add_hu4_3: mega_add
590     port map (clk, hu_mult4_1, hu_mult4_2, dummy_add_nan, dummy_add_ove
rflow, hu4, dummy_add_underflow);
591
592
593 --Calculation of hu5
594     mult_hu5_1: mega_mult
595     port map (clk, hu_sub3_1, hu_add4_1, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult5_1, dummy_mult_underflow);
596
597     mult_hu5_3: mega_mult
598     port map (clk, const_3, hu_mult4_2, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult5_3, dummy_mult_underflow);
599     sub_hu5_1: mega_sub
600     port map (clk, hu_mult4_1, hu_mult5_3, dummy_sub_nan, dummy_sub_ove
rflow, hu_mult5_2, dummy_sub_underflow);
601
602     mult_hu5_2: mega_mult
603     port map (clk, hu_mult5_1, hu_mult5_2, dummy_mult_nan, dummy_mult_ov
erflow, hu_sub5_1, dummy_mult_underflow);
604
605     mult_hu5_4: mega_mult
606     port map (clk, hu_sub3_2, hu_add4_2, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult5_4, dummy_mult_underflow);
607     mult_hu5_5: mega_mult
608     port map (clk, const_3, hu_mult4_1, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult5_5, dummy_mult_underflow);
609     sub_hu5_2: mega_sub
610     port map (clk, hu_mult5_5, hu_mult4_2, dummy_sub_nan, dummy_sub_ove
rflow, hu_sub5_2, dummy_sub_underflow);
611     mult_hu5_6: mega_mult
612     port map (clk, hu_mult5_4, hu_sub5_2, dummy_mult_nan, dummy_mult_ove
rflow, hu_mult5_6, dummy_mult_underflow);
613
614     add_hu5: mega_add
615     port map (clk, hu_sub5_1, hu_mult5_6, dummy_add_nan, dummy_add_ove
```

```

low,hu5,dummy_add_underflow);
616
617
618
619 --Calculation of hu6
620     sub_hu6_1: mega_sub
621     port map (clk,hu_mult4_1,hu_mult4_2,dummy_sub_nan,dummy_sub_ove
flow,hu_sub6_1,dummy_sub_underflow);
622     mult_hu6_1: mega_mult
623     port map (clk,hu_sub2,hu_sub6_1,dummy_mult_nan,dummy_mult_ove
flow,hu_mult6_1,dummy_mult_underflow);
624
625     mult_hu6_2: mega_mult
626     port map (clk,hu_add4_1,hu_add4_2,dummy_mult_nan,dummy_mult_ove
flow,hu_mult6_2,dummy_mult_underflow);
627     mult_hu6_3: mega_mult
628     port map (clk,const_4,eta22,dummy_mult_nan,dummy_mult_overflow,h
u_mult6_3,dummy_mult_underflow);
629     mult_hu6_4: mega_mult
630     port map (clk,hu_mult6_2,hu_mult6_3,dummy_mult_nan,dummy_mult_ove
rflow,hu_mult6_4,dummy_mult_underflow);
631
632     add_hu6: mega_add
633     port map (clk,hu_mult6_1,hu_mult6_4,dummy_add_nan,dummy_add_ove
rflow,hu6,dummy_add_underflow);
634
635
636 --Calculation of hu7
637     mult_hu7_1: mega_mult
638     port map (clk,hu_sub3_2,hu_add4_1,dummy_mult_nan,dummy_mult_ove
rflow,hu_mult7_1,dummy_mult_underflow);
639     sub_hu7_1: mega_sub
640     port map (clk,hu_mult4_1,hu_mult5_3,dummy_sub_nan,dummy_sub_ove
rflow,hu_sub7_1,dummy_sub_underflow);
641     mult_hu7_2: mega_mult
642     port map (clk,hu_mult7_1,hu_sub7_1,dummy_mult_nan,dummy_mult_ove
rflow,hu_mult7_2,dummy_mult_underflow);
643
644
645     mult_hu7_3: mega_mult
646     port map (clk,const_3,hu_sub5_2,dummy_mult_nan,dummy_mult_ove
rflow,hu_mult7_3,dummy_mult_underflow);
647     sub_hu7_2: mega_sub
648     port map (clk,eta23,eta41,dummy_sub_nan,dummy_sub_overflow,hu_su
b7_2,dummy_sub_underflow);
649     mult_hu7_4: mega_mult
650     port map (clk,hu_sub7_2,hu_add4_2,dummy_mult_nan,dummy_mult_ove
rflow,hu_mult7_4,dummy_mult_underflow);
651     mult_hu7_5: mega_mult
652     port map (clk,hu_mult7_3,hu_mult7_4,dummy_mult_nan,dummy_mult_ove
rflow,hu_mult7_5,dummy_mult_underflow);
653
654     add_hu7: mega_add
655     port map (clk,hu_mult7_2,hu_mult7_5,dummy_add_nan,dummy_add_ove
rflow,hu7,dummy_add_underflow);
656
657
658
659 -----
-----
660 --      Final Output
661 -----

```

```

-----
662
663
664
665
666 -----
-----
667 --                USER DETECTION ALGORITHM
668 -----
-----
669
670 mask<=X"7FFFFFFF";
671
672     input1<=hu1;
673     input2<=hu2;
674     input3<=hu3;
675     input4<=hu4;
676     input5<=hu5;
677     input6<=hu6;
678     input7<=hu7;
679
680
681
682 -----
683 --                COMPUTATION FOR NEURON 1                --
684 -----
685     addr1_1<=X"000";
686     addr1_2<=X"001";
687     addr1_3<=X"002";
688     addr1_4<=X"003";
689     addr1_5<=X"004";
690     addr1_6<=X"005";
691     addr1_7<=X"006";
692     im_type(0)<=X"007";
693
694
695     mem_read1_1: rom
696     port map (addr1_1,clk,mem1_1);
697     mem_read1_2: rom
698     port map (addr1_2,clk,mem1_2);
699     mem_read1_3: rom
700     port map (addr1_3,clk,mem1_3);
701     mem_read1_4: rom
702     port map (addr1_4,clk,mem1_4);
703     mem_read1_5: rom
704     port map (addr1_5,clk,mem1_5);
705     mem_read1_6: rom
706     port map (addr1_6,clk,mem1_6);
707     mem_read1_7: rom
708     port map (addr1_7,clk,mem1_7);
709
710     user_sub1_1: mega_sub
711     port map (clk,input1,mem1_1,dummy_sub_nan,dummy_sub_overflow,sub
1_1,dummy_sub_underflow);
712     user_sub1_2: mega_sub
713     port map (clk,input2,mem1_2,dummy_sub_nan,dummy_sub_overflow,sub
1_2,dummy_sub_underflow);
714     user_sub1_3: mega_sub
715     port map (clk,input3,mem1_3,dummy_sub_nan,dummy_sub_overflow,sub
1_3,dummy_sub_underflow);
716     user_sub1_4: mega_sub
717     port map (clk,input4,mem1_4,dummy_sub_nan,dummy_sub_overflow,sub

```

```

1_4, dummy_sub_underflow);
718   user_sub1_5: mega_sub
719   port map (clk, input5, mem1_5, dummy_sub_nan, dummy_sub_overflow, sub
1_5, dummy_sub_underflow);
720   user_sub1_6: mega_sub
721   port map (clk, input6, mem1_6, dummy_sub_nan, dummy_sub_overflow, sub
1_6, dummy_sub_underflow);
722   user_sub1_7: mega_sub
723   port map (clk, input7, mem1_7, dummy_sub_nan, dummy_sub_overflow, sub
1_7, dummy_sub_underflow);
724
725 --Calculating absolute value
726   abs1_1<=sub1_1 AND mask;
727   abs1_2<=sub1_2 AND mask;
728   abs1_3<=sub1_3 AND mask;
729   abs1_4<=sub1_4 AND mask;
730   abs1_5<=sub1_5 AND mask;
731   abs1_6<=sub1_6 AND mask;
732   abs1_7<=sub1_7 AND mask;
733
734 --Addition operation (Calculating Manhattan Distance)
735   user_add1_1: mega_add
736   port map (clk, abs1_1, abs1_2, dummy_sub_nan, dummy_sub_overflow, add
1_12, dummy_sub_underflow);
737   user_add1_2: mega_add
738   port map (clk, abs1_3, abs1_4, dummy_sub_nan, dummy_sub_overflow, add
1_34, dummy_sub_underflow);
739   user_add1_3: mega_add
740   port map (clk, abs1_5, abs1_6, dummy_sub_nan, dummy_sub_overflow, add
1_56, dummy_sub_underflow);
741   user_add1_4: mega_add
742   port map (clk, add1_12, abs1_7, dummy_sub_nan, dummy_sub_overflow, ad
d1_27, dummy_sub_underflow);
743   user_add1_5: mega_add
744   port map (clk, add1_34, add1_56, dummy_sub_nan, dummy_sub_overflow, a
dd1_46, dummy_sub_underflow);
745   user_add1_6: mega_add
746   port map (clk, add1_27, add1_46, dummy_sub_nan, dummy_sub_overflow, d
istance(0), dummy_sub_underflow);
747
748 -----
749 --           CALCULATION END FOR NEURON 1
750 -----
751
752
753 -----
754 --           COMPUTATION FOR NEURON 2           --
755 -----
756   addr2_1<=X"008";
757   addr2_2<=X"009";
758   addr2_3<=X"00A";
759   addr2_4<=X"00B";
760   addr2_5<=X"00C";
761   addr2_6<=X"00D";
762   addr2_7<=X"00E";
763   im_type(1)<=X"00F";
764
765
766   mem_read2_1: rom
767   port map (addr2_1, clk, mem2_1);
768   mem_read2_2: rom
769   port map (addr2_2, clk, mem2_2);

```

```

770     mem_read2_3: rom
771     port map (addr2_3, clk, mem2_3);
772     mem_read2_4: rom
773     port map (addr2_4, clk, mem2_4);
774     mem_read2_5: rom
775     port map (addr2_5, clk, mem2_5);
776     mem_read2_6: rom
777     port map (addr2_6, clk, mem2_6);
778     mem_read2_7: rom
779     port map (addr2_7, clk, mem2_7);
780
781     user_sub2_1: mega_sub
782     port map (clk, input1, mem2_1, dummy_sub_nan, dummy_sub_overflow, sub
2_1, dummy_sub_underflow);
783     user_sub2_2: mega_sub
784     port map (clk, input2, mem2_2, dummy_sub_nan, dummy_sub_overflow, sub
2_2, dummy_sub_underflow);
785     user_sub2_3: mega_sub
786     port map (clk, input3, mem2_3, dummy_sub_nan, dummy_sub_overflow, sub
2_3, dummy_sub_underflow);
787     user_sub2_4: mega_sub
788     port map (clk, input4, mem2_4, dummy_sub_nan, dummy_sub_overflow, sub
2_4, dummy_sub_underflow);
789     user_sub2_5: mega_sub
790     port map (clk, input5, mem2_5, dummy_sub_nan, dummy_sub_overflow, sub
2_5, dummy_sub_underflow);
791     user_sub2_6: mega_sub
792     port map (clk, input6, mem2_6, dummy_sub_nan, dummy_sub_overflow, sub
2_6, dummy_sub_underflow);
793     user_sub2_7: mega_sub
794     port map (clk, input7, mem2_7, dummy_sub_nan, dummy_sub_overflow, sub
2_7, dummy_sub_underflow);
795
796     --Calculating absolute value
797     abs2_1<=sub2_1 AND mask;
798     abs2_2<=sub2_2 AND mask;
799     abs2_3<=sub2_3 AND mask;
800     abs2_4<=sub2_4 AND mask;
801     abs2_5<=sub2_5 AND mask;
802     abs2_6<=sub2_6 AND mask;
803     abs2_7<=sub2_7 AND mask;
804
805     --Addition operation (Calculating Manhattan Distance)
806     user_add2_1: mega_add
807     port map (clk, abs2_1, abs2_2, dummy_sub_nan, dummy_sub_overflow, add
2_12, dummy_sub_underflow);
808     user_add2_2: mega_add
809     port map (clk, abs2_3, abs2_4, dummy_sub_nan, dummy_sub_overflow, add
2_34, dummy_sub_underflow);
810     user_add2_3: mega_add
811     port map (clk, abs2_5, abs2_6, dummy_sub_nan, dummy_sub_overflow, add
2_56, dummy_sub_underflow);
812     user_add2_4: mega_add
813     port map (clk, add2_12, abs2_7, dummy_sub_nan, dummy_sub_overflow, ad
d2_27, dummy_sub_underflow);
814     user_add2_5: mega_add
815     port map (clk, add2_34, add2_56, dummy_sub_nan, dummy_sub_overflow, a
dd2_46, dummy_sub_underflow);
816     user_add2_6: mega_add
817     port map (clk, add2_27, add2_46, dummy_sub_nan, dummy_sub_overflow, d
istance(1), dummy_sub_underflow);
818

```

```

819 -----
820 --          CALCULATION END FOR NEURON 2
821 -----
822
823 -----
824 --          COMPUTATION FOR NEURON 3          --
825 -----
826     addr3_1<=X"010";
827     addr3_2<=X"011";
828     addr3_3<=X"012";
829     addr3_4<=X"013";
830     addr3_5<=X"014";
831     addr3_6<=X"015";
832     addr3_7<=X"016";
833     im_type(2)<=X"017";
834
835
836     mem_read3_1: rom
837     port map (addr3_1,clk,mem3_1);
838     mem_read3_2: rom
839     port map (addr3_2,clk,mem3_2);
840     mem_read3_3: rom
841     port map (addr3_3,clk,mem3_3);
842     mem_read3_4: rom
843     port map (addr3_4,clk,mem3_4);
844     mem_read3_5: rom
845     port map (addr3_5,clk,mem3_5);
846     mem_read3_6: rom
847     port map (addr3_6,clk,mem3_6);
848     mem_read3_7: rom
849     port map (addr3_7,clk,mem3_7);
850
851     user_sub3_1: mega_sub
852     port map (clk,input1,mem3_1,dummy_sub_nan,dummy_sub_overflow,sub
3_1,dummy_sub_underflow);
853     user_sub3_2: mega_sub
854     port map (clk,input2,mem3_2,dummy_sub_nan,dummy_sub_overflow,sub
3_2,dummy_sub_underflow);
855     user_sub3_3: mega_sub
856     port map (clk,input3,mem3_3,dummy_sub_nan,dummy_sub_overflow,sub
3_3,dummy_sub_underflow);
857     user_sub3_4: mega_sub
858     port map (clk,input4,mem3_4,dummy_sub_nan,dummy_sub_overflow,sub
3_4,dummy_sub_underflow);
859     user_sub3_5: mega_sub
860     port map (clk,input5,mem3_5,dummy_sub_nan,dummy_sub_overflow,sub
3_5,dummy_sub_underflow);
861     user_sub3_6: mega_sub
862     port map (clk,input6,mem3_6,dummy_sub_nan,dummy_sub_overflow,sub
3_6,dummy_sub_underflow);
863     user_sub3_7: mega_sub
864     port map (clk,input7,mem3_7,dummy_sub_nan,dummy_sub_overflow,sub
3_7,dummy_sub_underflow);
865
866 --Calculating absolute value
867     abs3_1<=sub3_1 AND mask;
868     abs3_2<=sub3_2 AND mask;
869     abs3_3<=sub3_3 AND mask;
870     abs3_4<=sub3_4 AND mask;
871     abs3_5<=sub3_5 AND mask;
872     abs3_6<=sub3_6 AND mask;
873     abs3_7<=sub3_7 AND mask;

```



```

874
875 --Addition operation (Calculating Manhattan Distance)
876     user_add3_1: mega_add
877     port map (clk, abs3_1, abs3_2, dummy_sub_nan, dummy_sub_overflow, add
3_12, dummy_sub_underflow);
878     user_add3_2: mega_add
879     port map (clk, abs3_3, abs3_4, dummy_sub_nan, dummy_sub_overflow, add
3_34, dummy_sub_underflow);
880     user_add3_3: mega_add
881     port map (clk, abs3_5, abs3_6, dummy_sub_nan, dummy_sub_overflow, add
3_56, dummy_sub_underflow);
882     user_add3_4: mega_add
883     port map (clk, add3_12, abs3_7, dummy_sub_nan, dummy_sub_overflow, ad
d3_27, dummy_sub_underflow);
884     user_add3_5: mega_add
885     port map (clk, add3_34, add3_56, dummy_sub_nan, dummy_sub_overflow, a
dd3_46, dummy_sub_underflow);
886     user_add3_6: mega_add
887     port map (clk, add3_27, add3_46, dummy_sub_nan, dummy_sub_overflow, d
istance(2), dummy_sub_underflow);
888
889 -----
890 --          CALCULATION END FOR NEURON 3
891 -----
892
893
894 -----
895 --          COMPUTATION FOR NEURON 4          --
896 -----
897     addr4_1<=X"018";
898     addr4_2<=X"019";
899     addr4_3<=X"01A";
900     addr4_4<=X"01B";
901     addr4_5<=X"01C";
902     addr4_6<=X"01D";
903     addr4_7<=X"01E";
904     im_type(3)<=X"01F";
905
906
907     mem_read4_1: rom
908     port map (addr4_1, clk, mem4_1);
909     mem_read4_2: rom
910     port map (addr4_2, clk, mem4_2);
911     mem_read4_3: rom
912     port map (addr4_3, clk, mem4_3);
913     mem_read4_4: rom
914     port map (addr4_4, clk, mem4_4);
915     mem_read4_5: rom
916     port map (addr4_5, clk, mem4_5);
917     mem_read4_6: rom
918     port map (addr4_6, clk, mem4_6);
919     mem_read4_7: rom
920     port map (addr4_7, clk, mem4_7);
921
922     user_sub4_1: mega_sub
923     port map (clk, input1, mem4_1, dummy_sub_nan, dummy_sub_overflow, sub
4_1, dummy_sub_underflow);
924     user_sub4_2: mega_sub
925     port map (clk, input2, mem4_2, dummy_sub_nan, dummy_sub_overflow, sub
4_2, dummy_sub_underflow);
926     user_sub4_3: mega_sub
927     port map (clk, input3, mem4_3, dummy_sub_nan, dummy_sub_overflow, sub

```

```

4_3, dummy_sub_underflow);
928 user_sub4_4: mega_sub
929 port map (clk, input4, mem4_4, dummy_sub_nan, dummy_sub_overflow, sub
4_4, dummy_sub_underflow);
930 user_sub4_5: mega_sub
931 port map (clk, input5, mem4_5, dummy_sub_nan, dummy_sub_overflow, sub
4_5, dummy_sub_underflow);
932 user_sub4_6: mega_sub
933 port map (clk, input6, mem4_6, dummy_sub_nan, dummy_sub_overflow, sub
4_6, dummy_sub_underflow);
934 user_sub4_7: mega_sub
935 port map (clk, input7, mem4_7, dummy_sub_nan, dummy_sub_overflow, sub
4_7, dummy_sub_underflow);
936
937 --Calculating absolute value
938 abs4_1<=sub4_1 AND mask;
939 abs4_2<=sub4_2 AND mask;
940 abs4_3<=sub4_3 AND mask;
941 abs4_4<=sub4_4 AND mask;
942 abs4_5<=sub4_5 AND mask;
943 abs4_6<=sub4_6 AND mask;
944 abs4_7<=sub4_7 AND mask;
945
946 --Addition operation (Calculating Manhattan Distance)
947 user_add4_1: mega_add
948 port map (clk, abs4_1, abs4_2, dummy_sub_nan, dummy_sub_overflow, add
4_12, dummy_sub_underflow);
949 user_add4_2: mega_add
950 port map (clk, abs4_3, abs4_4, dummy_sub_nan, dummy_sub_overflow, add
4_34, dummy_sub_underflow);
951 user_add4_3: mega_add
952 port map (clk, abs4_5, abs4_6, dummy_sub_nan, dummy_sub_overflow, add
4_56, dummy_sub_underflow);
953 user_add4_4: mega_add
954 port map (clk, add4_12, abs4_7, dummy_sub_nan, dummy_sub_overflow, ad
d4_27, dummy_sub_underflow);
955 user_add4_5: mega_add
956 port map (clk, add4_34, add4_56, dummy_sub_nan, dummy_sub_overflow, a
dd4_46, dummy_sub_underflow);
957 user_add4_6: mega_add
958 port map (clk, add4_27, add4_46, dummy_sub_nan, dummy_sub_overflow, d
istance(3), dummy_sub_underflow);
959
960 -----
961 --          CALCULATION END FOR NEURON 4
962 -----
963
964 -----
965 --          COMPUTATION FOR NEURON 5          --
966 -----
967 addr5_1<=X"020";
968 addr5_2<=X"021";
969 addr5_3<=X"022";
970 addr5_4<=X"023";
971 addr5_5<=X"024";
972 addr5_6<=X"025";
973 addr5_7<=X"026";
974 in_type(4)<=X"027";
975
976
977 mem_read5_1: rom
978 port map (addr5_1, clk, mem5_1);

```

```
979     mem_read5_2: rom
980     port map (addr5_2, clk, mem5_2);
981     mem_read5_3: rom
982     port map (addr5_3, clk, mem5_3);
983     mem_read5_4: rom
984     port map (addr5_4, clk, mem5_4);
985     mem_read5_5: rom
986     port map (addr5_5, clk, mem5_5);
987     mem_read5_6: rom
988     port map (addr5_6, clk, mem5_6);
989     mem_read5_7: rom
990     port map (addr5_7, clk, mem5_7);
991
992     user_sub5_1: mega_sub
993     port map (clk, input1, mem5_1, dummy_sub_nan, dummy_sub_overflow, sub
5_1, dummy_sub_underflow);
994     user_sub5_2: mega_sub
995     port map (clk, input2, mem5_2, dummy_sub_nan, dummy_sub_overflow, sub
5_2, dummy_sub_underflow);
996     user_sub5_3: mega_sub
997     port map (clk, input3, mem5_3, dummy_sub_nan, dummy_sub_overflow, sub
5_3, dummy_sub_underflow);
998     user_sub5_4: mega_sub
999     port map (clk, input4, mem5_4, dummy_sub_nan, dummy_sub_overflow, sub
5_4, dummy_sub_underflow);
1000     user_sub5_5: mega_sub
1001     port map (clk, input5, mem5_5, dummy_sub_nan, dummy_sub_overflow, sub
5_5, dummy_sub_underflow);
1002     user_sub5_6: mega_sub
1003     port map (clk, input6, mem5_6, dummy_sub_nan, dummy_sub_overflow, sub
5_6, dummy_sub_underflow);
1004     user_sub5_7: mega_sub
1005     port map (clk, input7, mem5_7, dummy_sub_nan, dummy_sub_overflow, sub
5_7, dummy_sub_underflow);
1006
1007 --Calculating absolute value
1008     abs5_1<=sub5_1 AND mask;
1009     abs5_2<=sub5_2 AND mask;
1010     abs5_3<=sub5_3 AND mask;
1011     abs5_4<=sub5_4 AND mask;
1012     abs5_5<=sub5_5 AND mask;
1013     abs5_6<=sub5_6 AND mask;
1014     abs5_7<=sub5_7 AND mask;
1015
1016 --Addition operation (Calculating Manhattan Distance)
1017     user_add5_1: mega_add
1018     port map (clk, abs5_1, abs5_2, dummy_sub_nan, dummy_sub_overflow, add
5_12, dummy_sub_underflow);
1019     user_add5_2: mega_add
1020     port map (clk, abs5_3, abs5_4, dummy_sub_nan, dummy_sub_overflow, add
5_34, dummy_sub_underflow);
1021     user_add5_3: mega_add
1022     port map (clk, abs5_5, abs5_6, dummy_sub_nan, dummy_sub_overflow, add
5_56, dummy_sub_underflow);
1023     user_add5_4: mega_add
1024     port map (clk, add5_12, abs5_7, dummy_sub_nan, dummy_sub_overflow, ad
d5_27, dummy_sub_underflow);
1025     user_add5_5: mega_add
1026     port map (clk, add5_34, add5_56, dummy_sub_nan, dummy_sub_overflow, a
dd5_46, dummy_sub_underflow);
1027     user_add5_6: mega_add
1028     port map (clk, add5_27, add5_46, dummy_sub_nan, dummy_sub_overflow, d
```

```

instance (4), dummy_sub_underflow);
1029
1030 -----
1031 --          CALCULATION END FOR NEURON 5
1032 -----
1033
1034
1035 -----
1036 --          COMPUTATION FOR NEURON 6          --
1037 -----
1038     addr6_1<=X"028";
1039     addr6_2<=X"029";
1040     addr6_3<=X"02A";
1041     addr6_4<=X"02B";
1042     addr6_5<=X"02C";
1043     addr6_6<=X"02D";
1044     addr6_7<=X"02E";
1045     im_type (5)<=X"02F";
1046
1047
1048     mem_read6_1: rom
1049     port map (addr6_1,clk,mem6_1);
1050     mem_read6_2: rom
1051     port map (addr6_2,clk,mem6_2);
1052     mem_read6_3: rom
1053     port map (addr6_3,clk,mem6_3);
1054     mem_read6_4: rom
1055     port map (addr6_4,clk,mem6_4);
1056     mem_read6_5: rom
1057     port map (addr6_5,clk,mem6_5);
1058     mem_read6_6: rom
1059     port map (addr6_6,clk,mem6_6);
1060     mem_read6_7: rom
1061     port map (addr6_7,clk,mem6_7);
1062
1063     user_sub6_1: mega_sub
1064     port map (clk,input1,mem6_1,dummy_sub_nan,dummy_sub_overflow,sub
6_1,dummy_sub_underflow);
1065     user_sub6_2: mega_sub
1066     port map (clk,input2,mem6_2,dummy_sub_nan,dummy_sub_overflow,sub
6_2,dummy_sub_underflow);
1067     user_sub6_3: mega_sub
1068     port map (clk,input3,mem6_3,dummy_sub_nan,dummy_sub_overflow,sub
6_3,dummy_sub_underflow);
1069     user_sub6_4: mega_sub
1070     port map (clk,input4,mem6_4,dummy_sub_nan,dummy_sub_overflow,sub
6_4,dummy_sub_underflow);
1071     user_sub6_5: mega_sub
1072     port map (clk,input5,mem6_5,dummy_sub_nan,dummy_sub_overflow,sub
6_5,dummy_sub_underflow);
1073     user_sub6_6: mega_sub
1074     port map (clk,input6,mem6_6,dummy_sub_nan,dummy_sub_overflow,sub
6_6,dummy_sub_underflow);
1075     user_sub6_7: mega_sub
1076     port map (clk,input7,mem6_7,dummy_sub_nan,dummy_sub_overflow,sub
6_7,dummy_sub_underflow);
1077
1078 --Calculating absolute value
1079     abs6_1<=sub6_1 AND mask;
1080     abs6_2<=sub6_2 AND mask;
1081     abs6_3<=sub6_3 AND mask;
1082     abs6_4<=sub6_4 AND mask;

```

```

1083     abs6_5<=sub6_5 AND mask;
1084     abs6_6<=sub6_6 AND mask;
1085     abs6_7<=sub6_7 AND mask;
1086
1087 --Addition operation (Calculating Manhattan Distance)
1088     user_add6_1: mega_add
1089     port map (clk, abs6_1, abs6_2, dummy_sub_nan, dummy_sub_overflow, add
1090 6_12, dummy_sub_underflow);
1091     user_add6_2: mega_add
1092     port map (clk, abs6_3, abs6_4, dummy_sub_nan, dummy_sub_overflow, add
1093 6_34, dummy_sub_underflow);
1094     user_add6_3: mega_add
1095     port map (clk, abs6_5, abs6_6, dummy_sub_nan, dummy_sub_overflow, add
1096 6_56, dummy_sub_underflow);
1097     user_add6_4: mega_add
1098     port map (clk, add6_12, abs6_7, dummy_sub_nan, dummy_sub_overflow, ad
1099 d6_27, dummy_sub_underflow);
1100     user_add6_5: mega_add
1101     port map (clk, add6_34, add6_56, dummy_sub_nan, dummy_sub_overflow, a
1102 dd6_46, dummy_sub_underflow);
1103     user_add6_6: mega_add
1104     port map (clk, add6_27, add6_46, dummy_sub_nan, dummy_sub_overflow, d
1105 istance(5), dummy_sub_underflow);
1106
1107 -----
1108 --          CALCULATION END FOR NEURON 6
1109 -----
1110
1111 -----
1112 --          COMPUTATION FOR NEURON 7          --
1113 -----
1114     addr7_1<=X"030";
1115     addr7_2<=X"031";
1116     addr7_3<=X"032";
1117     addr7_4<=X"033";
1118     addr7_5<=X"034";
1119     addr7_6<=X"035";
1120     addr7_7<=X"036";
1121     im_type(6)<=X"037";
1122
1123     mem_read7_1: rom
1124     port map (addr7_1, clk, mem7_1);
1125     mem_read7_2: rom
1126     port map (addr7_2, clk, mem7_2);
1127     mem_read7_3: rom
1128     port map (addr7_3, clk, mem7_3);
1129     mem_read7_4: rom
1130     port map (addr7_4, clk, mem7_4);
1131     mem_read7_5: rom
1132     port map (addr7_5, clk, mem7_5);
1133     mem_read7_6: rom
1134     port map (addr7_6, clk, mem7_6);
1135     mem_read7_7: rom
1136     port map (addr7_7, clk, mem7_7);
1137
1138     user_sub7_1: mega_sub
1139     port map (clk, input1, mem7_1, dummy_sub_nan, dummy_sub_overflow, sub
1140 7_1, dummy_sub_underflow);
1141     user_sub7_2: mega_sub
1142     port map (clk, input2, mem7_2, dummy_sub_nan, dummy_sub_overflow, sub
1143 7_2, dummy_sub_underflow);

```

```

1137     user_sub7_3: mega_sub
1138     port map (clk, input3, mem7_3, dummy_sub_nan, dummy_sub_overflow, sub
1139 7_3, dummy_sub_underflow);
1139     user_sub7_4: mega_sub
1140     port map (clk, input4, mem7_4, dummy_sub_nan, dummy_sub_overflow, sub
1141 7_4, dummy_sub_underflow);
1141     user_sub7_5: mega_sub
1142     port map (clk, input5, mem7_5, dummy_sub_nan, dummy_sub_overflow, sub
1143 7_5, dummy_sub_underflow);
1143     user_sub7_6: mega_sub
1144     port map (clk, input6, mem7_6, dummy_sub_nan, dummy_sub_overflow, sub
1145 7_6, dummy_sub_underflow);
1145     user_sub7_7: mega_sub
1146     port map (clk, input7, mem7_7, dummy_sub_nan, dummy_sub_overflow, sub
1147 7_7, dummy_sub_underflow);
1147
1148 --Calculating absolute value
1149     abs7_1<=sub7_1 AND mask;
1150     abs7_2<=sub7_2 AND mask;
1151     abs7_3<=sub7_3 AND mask;
1152     abs7_4<=sub7_4 AND mask;
1153     abs7_5<=sub7_5 AND mask;
1154     abs7_6<=sub7_6 AND mask;
1155     abs7_7<=sub7_7 AND mask;
1156
1157 --Addition operation (Calculating Manhattan Distance)
1158     user_add7_1: mega_add
1159     port map (clk, abs7_1, abs7_2, dummy_sub_nan, dummy_sub_overflow, add
1160 7_12, dummy_sub_underflow);
1160     user_add7_2: mega_add
1161     port map (clk, abs7_3, abs7_4, dummy_sub_nan, dummy_sub_overflow, add
1162 7_34, dummy_sub_underflow);
1162     user_add7_3: mega_add
1163     port map (clk, abs7_5, abs7_6, dummy_sub_nan, dummy_sub_overflow, add
1164 7_56, dummy_sub_underflow);
1164     user_add7_4: mega_add
1165     port map (clk, add7_12, abs7_7, dummy_sub_nan, dummy_sub_overflow, ad
1166 d7_27, dummy_sub_underflow);
1166     user_add7_5: mega_add
1167     port map (clk, add7_34, add7_56, dummy_sub_nan, dummy_sub_overflow, a
1168 dd7_46, dummy_sub_underflow);
1168     user_add7_6: mega_add
1169     port map (clk, add7_27, add7_46, dummy_sub_nan, dummy_sub_overflow, d
1170 istance(6), dummy_sub_underflow);
1170
1171 -----
1172 --          CALCULATION END FOR NEURON 7
1173 -----
1174
1175
1176 -----
1177 --          COMPUTATION FOR NEURON 8          --
1178 -----
1179     addr8_1<=X"038";
1180     addr8_2<=X"039";
1181     addr8_3<=X"03A";
1182     addr8_4<=X"03B";
1183     addr8_5<=X"03C";
1184     addr8_6<=X"03D";
1185     addr8_7<=X"03E";
1186     im_type(7)<=X"03F";
1187

```

```

1188
1189     mem_read8_1: rom
1190     port map (addr8_1, clk, mem8_1);
1191     mem_read8_2: rom
1192     port map (addr8_2, clk, mem8_2);
1193     mem_read8_3: rom
1194     port map (addr8_3, clk, mem8_3);
1195     mem_read8_4: rom
1196     port map (addr8_4, clk, mem8_4);
1197     mem_read8_5: rom
1198     port map (addr8_5, clk, mem8_5);
1199     mem_read8_6: rom
1200     port map (addr8_6, clk, mem8_6);
1201     mem_read8_7: rom
1202     port map (addr8_7, clk, mem8_7);
1203
1204     user_sub8_1: mega_sub
1205     port map (clk, input1, mem8_1, dummy_sub_nan, dummy_sub_overflow, sub
8_1, dummy_sub_underflow);
1206     user_sub8_2: mega_sub
1207     port map (clk, input2, mem8_2, dummy_sub_nan, dummy_sub_overflow, sub
8_2, dummy_sub_underflow);
1208     user_sub8_3: mega_sub
1209     port map (clk, input3, mem8_3, dummy_sub_nan, dummy_sub_overflow, sub
8_3, dummy_sub_underflow);
1210     user_sub8_4: mega_sub
1211     port map (clk, input4, mem8_4, dummy_sub_nan, dummy_sub_overflow, sub
8_4, dummy_sub_underflow);
1212     user_sub8_5: mega_sub
1213     port map (clk, input5, mem8_5, dummy_sub_nan, dummy_sub_overflow, sub
8_5, dummy_sub_underflow);
1214     user_sub8_6: mega_sub
1215     port map (clk, input6, mem8_6, dummy_sub_nan, dummy_sub_overflow, sub
8_6, dummy_sub_underflow);
1216     user_sub8_7: mega_sub
1217     port map (clk, input7, mem8_7, dummy_sub_nan, dummy_sub_overflow, sub
8_7, dummy_sub_underflow);
1218
1219     --Calculating absolute value
1220     abs8_1<=sub8_1 AND mask;
1221     abs8_2<=sub8_2 AND mask;
1222     abs8_3<=sub8_3 AND mask;
1223     abs8_4<=sub8_4 AND mask;
1224     abs8_5<=sub8_5 AND mask;
1225     abs8_6<=sub8_6 AND mask;
1226     abs8_7<=sub8_7 AND mask;
1227
1228     --Addition operation (Calculating Manhattan Distance)
1229     user_add8_1: mega_add
1230     port map (clk, abs8_1, abs8_2, dummy_sub_nan, dummy_sub_overflow, add
8_12, dummy_sub_underflow);
1231     user_add8_2: mega_add
1232     port map (clk, abs8_3, abs8_4, dummy_sub_nan, dummy_sub_overflow, add
8_34, dummy_sub_underflow);
1233     user_add8_3: mega_add
1234     port map (clk, abs8_5, abs8_6, dummy_sub_nan, dummy_sub_overflow, add
8_56, dummy_sub_underflow);
1235     user_add8_4: mega_add
1236     port map (clk, add8_12, abs8_7, dummy_sub_nan, dummy_sub_overflow, ad
d8_27, dummy_sub_underflow);
1237     user_add8_5: mega_add
1238     port map (clk, add8_34, add8_56, dummy_sub_nan, dummy_sub_overflow, a

```

```

dd8_46, dummy_sub_underflow);
1239   user_add8_6: mega_add
1240   port map (clk, add8_27, add8_46, dummy_sub_nan, dummy_sub_overflow, d
istance(7), dummy_sub_underflow);
1241
1242 -----
1243 --          CALCULATION END FOR NEURON 8
1244 -----
1245   process (clock, distance(0), distance(1), distance(2), distance(3), di
stance(4), distance(5), distance(6), distance(7))
1246   begin
1247
1248       if (clock'EVENT AND clock='1') then --the calculations are b
ased on clock cycle
1249           count2<=count2+1;
1250           count3<=count3+1;
1251
1252
1253           if (count2 = 20) then
1254
1255               temp<=distance(0);
1256               index <= 0;
1257
1258           end if;
1259
1260           if (count2 = 21) then
1261               if (temp > distance(1)) then
1262                   temp<=distance(1);
1263                   index <= 1;
1264               end if;
1265           end if;
1266
1267           if (count2 = 22) then
1268               if (temp > distance(2)) then
1269                   temp<=distance(2);
1270                   index <= 2;
1271               end if;
1272           end if;
1273
1274           if (count2 = 23) then
1275               if (temp > distance(3)) then
1276                   temp<=distance(3);
1277                   index <= 3;
1278               end if;
1279           end if;
1280
1281           if (count2 = 24) then
1282               if (temp > distance(4)) then
1283                   temp<=distance(4);
1284                   index <= 4;
1285               end if;
1286           end if;
1287
1288           if (count2 = 25) then
1289               if (temp > distance(5)) then
1290                   temp<=distance(5);
1291                   index <= 5;
1292               end if;
1293           end if;
1294
1295           if (count2 = 26) then
1296               if (temp > distance(6)) then

```



```

1297         temp<=distance(6);
1298         index <= 6;
1299     end if;
1300 end if;
1301
1302     if (count2 = 27) then
1303         if (temp > distance(7)) then
1304             temp<=distance(7);
1305             index <= 7;
1306         end if;
1307     end if;
1308 end if;
1309
1310 end process;
1311
1312 mem_read_final: rom
1313 port map (im_type(index), clock, test1);
1314
1315
1316
1317
1318
1319
1320 end behaviour;
1321
1322
1323 -----
1324 --                               MEGA FUNCTION FLOATING POINT ADDITION
1325 -----
1326 -- megafunction wizard: %ALTFP_ADD_SUB%
1327 -- GENERATION: STANDARD
1328 -- VERSION: WM1.0
1329 -- MODULE: altfp_add_sub
1330
1331 -- =====
1332 -- File Name: mega_add.vhd
1333 -- Megafunction Name(s):
1334 --         altfp_add_sub
1335 -- =====
1336 -- *****
1337 -- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
1338 --
1339 -- 5.1 Build 176 10/26/2005 SJ Full Version
1340 -- *****
1341
1342
1343 -----
1344 --         SYSTEM GENERATED CODE. Not included in thesis
1345 -----
1346
1347
1348
1349 -----
1350 --                               MEGA FUNCTION FLOATING POINT SUBTRACTION
1351 -----
1352
1353
1354 -- megafunction wizard: %ALTFP_ADD_SUB%

```

```

1355 -- GENERATION: STANDARD
1356 -- VERSION: WM1.0
1357 -- MODULE: altfp_add_sub
1358
1359 -- =====
1360 -- File Name: mega_sub.vhd
1361 -- Megafunction Name(s):
1362 --         altfp_add_sub
1363 -- =====
1364 -- *****
1365 -- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
1366 --
1367 -- 5.1 Build 176 10/26/2005 SJ Full Version
1368 -- *****
1369
1370 -----
1371 --         SYSTEM GENERATED CODE. Not included in thesis
1372 -----
1373
1374
1375
1376
1377 -----
1378 --         MEGA FUNCTION FLOATING POINT MULTIPLICATION
1379 -----
1380 -- megafunction wizard: %ALTFP_MULT%
1381 -- GENERATION: STANDARD
1382 -- VERSION: WM1.0
1383 -- MODULE: altfp_mult
1384
1385 -- =====
1386 -- File Name: mega_mult.vhd
1387 -- Megafunction Name(s):
1388 --         altfp_mult
1389 -- =====
1390 -- *****
1391 -- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
1392 --
1393 -- 5.1 Build 176 10/26/2005 SJ Full Version
1394 -- *****
1395
1396
1397
1398 LIBRARY ieee;
1399 USE ieee.std_logic_1164.all;
1400
1401 LIBRARY altera_mf;
1402 USE altera_mf.altera_mf_components.all;
1403
1404 ENTITY mega_mult IS
1405     PORT
1406     (
1407         clock      : IN STD_LOGIC ;
1408         dataa      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
1409         datab     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
1410         nan        : OUT STD_LOGIC ;
1411         overflow   : OUT STD_LOGIC ;
1412         result     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
1413         underflow  : OUT STD_LOGIC
1414     );

```

```
1415 END mega_mult;
1416
1417
1418 ARCHITECTURE SYN OF mega_mult IS
1419
1420     SIGNAL sub_wire0      : STD_LOGIC ;
1421     SIGNAL sub_wire1      : STD_LOGIC ;
1422     SIGNAL sub_wire2      : STD_LOGIC ;
1423     SIGNAL sub_wire3      : STD_LOGIC_VECTOR (31 DOWNT0 0);
1424
1425
1426
1427     COMPONENT altfp_mult
1428     GENERIC (
1429         dedicated_multiplier_circuitry      : STRING;
1430         lpm_type                             : STRING;
1431         pipeline                             : NATURAL;
1432         reduced_functionality                : STRING;
1433         width_exp                            : NATURAL;
1434         width_man                            : NATURAL
1435     );
1436     PORT (
1437         dataa      : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
1438         datab      : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
1439         overflow    : OUT STD_LOGIC ;
1440         nan         : OUT STD_LOGIC ;
1441         underflow   : OUT STD_LOGIC ;
1442         clock       : IN STD_LOGIC ;
1443         result      : OUT STD_LOGIC_VECTOR (31 DOWNT0 0)
1444     );
1445     END COMPONENT;
1446
1447 BEGIN
1448     overflow    <= sub_wire0;
1449     nan         <= sub_wire1;
1450     underflow   <= sub_wire2;
1451     result      <= sub_wire3(31 DOWNT0 0);
1452
1453     altfp_mult_component : altfp_mult
1454     GENERIC MAP (
1455         dedicated_multiplier_circuitry => "YES",
1456         lpm_type => "altfp_mult",
1457         pipeline => 5,
1458         reduced_functionality => "YES",
1459         width_exp => 8,
1460         width_man => 23
1461     )
1462     PORT MAP (
1463         dataa => dataa,
1464         datab => datab,
1465         clock => clock,
1466         overflow => sub_wire0,
1467         nan => sub_wire1,
1468         underflow => sub_wire2,
1469         result => sub_wire3
1470     );
1471
1472
1473
1474 END SYN;
1475
1476
```

```
1477 -----
1478 --          Integer to IEEE 754 Floating Point format conversion
1479 -----
1480
1481 LIBRARY ieee ;
1482 USE ieee.std_logic_1164.all ;
1483 USE ieee.std_logic_arith.ALL;
1484 USE ieee.std_logic_unsigned.ALL;
1485
1486
1487 -- Top-level entity
1488 ENTITY int2fp IS
1489
1490 PORT (input : integer range 0 to 100000;
1491       fp_op : OUT STD_LOGIC_VECTOR(31 downto 0));
1492
1493
1494
1495 END int2fp ;
1496
1497
1498 ARCHITECTURE Behavior OF int2fp IS
1499
1500 signal int_ip : STD_LOGIC_VECTOR(31 downto 0);
1501 signal sign_bit : STD_LOGIC ;
1502 signal exponent : STD_LOGIC_VECTOR(7 downto 0);
1503 signal mentissa : STD_LOGIC_VECTOR(22 downto 0);
1504
1505 BEGIN
1506
1507     int_ip<=(conv_std_logic_vector(input,32));
1508
1509     sign_bit<='0';
1510
1511     process(int_ip)
1512     begin
1513         if(int_ip(31)='1')then
1514             exponent<=X"9E";
1515             mentissa<=int_ip(30 downto 8);
1516
1517         elsif(int_ip(30)='1')then
1518             exponent<=X"9D";
1519             mentissa<=int_ip(29 downto 7);
1520
1521         elsif(int_ip(29)='1')then
1522             exponent<=X"9C";
1523             mentissa<=int_ip(28 downto 6);
1524
1525         elsif(int_ip(28)='1')then
1526             exponent<=X"9B";
1527             mentissa<=int_ip(27 downto 5);
1528
1529         elsif(int_ip(27)='1')then
1530             exponent<=X"9A";
1531             mentissa<=int_ip(26 downto 4);
1532
1533         elsif(int_ip(26)='1')then
1534             exponent<=X"99";
1535             mentissa<=int_ip(25 downto 3);
1536
```

```
1537     elsif(int_ip(25)='1') then
1538         exponent<=X"98";
1539         mentissa<=int_ip(24 downto 2);
1540
1541     elsif(int_ip(24)='1') then
1542         exponent<=X"97";
1543         mentissa<=int_ip(23 downto 1);
1544
1545     elsif(int_ip(23)='1') then
1546         exponent<=X"96";
1547         mentissa<=int_ip(22 downto 0);
1548
1549     elsif(int_ip(22)='1') then
1550         exponent<=X"95";
1551         mentissa(22 downto 1)<=int_ip(21 downto 0);
1552         mentissa(0)<='0';
1553
1554     elsif(int_ip(21)='1') then
1555         exponent<=X"94";
1556         mentissa(22 downto 2)<=int_ip(20 downto 0);
1557         mentissa(1 downto 0)<=(OTHERS=>'0');
1558
1559     elsif(int_ip(20)='1') then
1560         exponent<=X"93";
1561         mentissa(22 downto 3)<=int_ip(19 downto 0);
1562         mentissa(2 downto 0)<=(OTHERS=>'0');
1563
1564     elsif(int_ip(19)='1') then
1565         exponent<=X"92";
1566         mentissa(22 downto 4)<=int_ip(18 downto 0);
1567         mentissa(3 downto 0)<=(OTHERS=>'0');
1568
1569     elsif(int_ip(18)='1') then
1570         exponent<=X"91";
1571         mentissa(22 downto 5)<=int_ip(17 downto 0);
1572         mentissa(4 downto 0)<=(OTHERS=>'0');
1573
1574     elsif(int_ip(17)='1') then
1575         exponent<=X"90";
1576         mentissa(22 downto 6)<=int_ip(16 downto 0);
1577         mentissa(5 downto 0)<=(OTHERS=>'0');
1578
1579     elsif(int_ip(16)='1') then
1580         exponent<=X"8F";
1581         mentissa(22 downto 7)<=int_ip(15 downto 0);
1582         mentissa(6 downto 0)<=(OTHERS=>'0');
1583
1584     elsif(int_ip(15)='1') then
1585         exponent<=X"8E";
1586         mentissa(22 downto 8)<=int_ip(14 downto 0);
1587         mentissa(7 downto 0)<=(OTHERS=>'0');
1588
1589     elsif(int_ip(14)='1') then
1590         exponent<=X"8D";
1591         mentissa(22 downto 9)<=int_ip(13 downto 0);
1592         mentissa(8 downto 0)<=(OTHERS=>'0');
1593
1594     elsif(int_ip(13)='1') then
1595         exponent<=X"8C";
1596         mentissa(22 downto 10)<=int_ip(12 downto 0);
1597         mentissa(9 downto 0)<=(OTHERS=>'0');
1598
```

```
1599     elsif(int_ip(12)='1') then
1600         exponent<=X"8B";
1601         mentissa(22 downto 11)<=int_ip(11 downto 0);
1602         mentissa(10 downto 0)<=(OTHERS=>'0');
1603
1604     elsif(int_ip(11)='1') then
1605         exponent<=X"8A";
1606         mentissa(22 downto 12)<=int_ip(10 downto 0);
1607         mentissa(11 downto 0)<=(OTHERS=>'0');
1608
1609     elsif(int_ip(10)='1') then
1610         exponent<=X"89";
1611         mentissa(22 downto 13)<=int_ip(9 downto 0);
1612         mentissa(12 downto 0)<=(OTHERS=>'0');
1613
1614     elsif(int_ip(9)='1') then
1615         exponent<=X"88";
1616         mentissa(22 downto 14)<=int_ip(8 downto 0);
1617         mentissa(13 downto 0)<=(OTHERS=>'0');
1618
1619     elsif(int_ip(8)='1') then
1620         exponent<=X"87";
1621         mentissa(22 downto 15)<=int_ip(7 downto 0);
1622         mentissa(14 downto 0)<=(OTHERS=>'0');
1623
1624     elsif(int_ip(7)='1') then
1625         exponent<=X"86";
1626         mentissa(22 downto 16)<=int_ip(6 downto 0);
1627         mentissa(15 downto 0)<=(OTHERS=>'0');
1628
1629     elsif(int_ip(6)='1') then
1630         exponent<=X"85";
1631         mentissa(22 downto 17)<=int_ip(5 downto 0);
1632         mentissa(16 downto 0)<=(OTHERS=>'0');
1633
1634     elsif(int_ip(5)='1') then
1635         exponent<=X"84";
1636         mentissa(22 downto 18)<=int_ip(4 downto 0);
1637         mentissa(17 downto 0)<=(OTHERS=>'0');
1638
1639     elsif(int_ip(4)='1') then
1640         exponent<=X"83";
1641         mentissa(22 downto 19)<=int_ip(3 downto 0);
1642         mentissa(18 downto 0)<=(OTHERS=>'0');
1643
1644     elsif(int_ip(3)='1') then
1645         exponent<=X"82";
1646         mentissa(22 downto 20)<=int_ip(2 downto 0);
1647         mentissa(19 downto 0)<=(OTHERS=>'0');
1648
1649     elsif(int_ip(2)='1') then
1650         exponent<=X"81";
1651         mentissa(22 downto 21)<=int_ip(1 downto 0);
1652         mentissa(20 downto 0)<=(OTHERS=>'0');
1653
1654     elsif(int_ip(1)='1') then
1655         exponent<=X"80";
1656         mentissa(22)<=int_ip(0);
1657         mentissa(21 downto 0)<=(OTHERS=>'0');
1658
1659     elsif(int_ip(0)='1') then
1660         exponent<=X"7F";
```

```

1661         mentissa(22 downto 0) <= (OTHERS => '0');
1662     end if;
1663 end process;
1664
1665 end process;
1666
1667 fp_op(31) <= sign_bit;
1668 fp_op(30 downto 23) <= exponent;
1669 fp_op(22 downto 0) <= mentissa;
1670
1671
1672 END Behavior;
1673
1674
1675
1676
1677
1678
1679 -----
1680 --                                     CALCULATION OF FLOATING POINT DIVISION
1681 -----
1682
1683 -----
1684 --          Calculation of Floating point division
1685 -----
1686
1687
1688
1689 --
1690 -----
1691 -- Copyright 2001 by Howard University All rights reserved.
1692 --
1693 -- Design: A Row of an Integer Divider
1694 -- Name:    Tiffany Crawford and Clay Gloster
1695 --
1696 -- Date:    May 2001
1697 -- Description: Generic Module
1698 -- N is the width of the numerator and denominators
1699 -- Produces one output bit Q
1700 -----
1701     library IEEE;
1702     use IEEE.std_logic_1164.all;
1703     use IEEE.std_logic_arith.all;
1704     use IEEE.std_logic_unsigned."-";
1705     use IEEE.std_logic_unsigned.">";
1706     use IEEE.std_logic_unsigned."=";
1707
1708
1709
1710     entity row_generic is          --Declare a row with N cells
1711     generic (N: integer);
1712     port( NUM: in std_logic_vector(N-1 downto 0);
1713           DEN : in std_logic_vector (N-1 downto 0);
1714           DIFF : out std_logic_vector(N-1 downto 0);
1715           DIV_OUT: out std_logic_vector(N-1 downto 0);
1716           Q : out std_logic );
1717     end row_generic;
1718

```

```

1719
1720
1721     architecture functional of row_generic is
1722
1723         begin
1724             process (NUM, DEN)
1725
1726                 begin
1727                     IF (NUM >= DEN) then DIFF <= NUM - DEN;
1728                     Q <= '1';
1729                     else
1730                         DIFF <= NUM;
1731                         Q <= '0';
1732                     end if;
1733                     DIV_OUT <= DEN;
1734
1735                 end process;
1736             end functional;
1737 -----
1738 -- Copyright 2001 by Howard University All rights reserved.
1739 --
1740 -- Design: A Connect Module that connects two rows of an integer divider
1741 -- Name:      Tiffany Crawford and Clay Gloster
1742 --
1743 -- Date:      May 2001
1744 -- Description: Generic Module
1745 -- N is the width of the inputs and outputs
1746 -- Passes div_out to den for the next row
1747 -- Shifts diff for use in the next row numerator.
1748 -----
1749
1750     library IEEE;
1751     use IEEE.std_logic_1164.all;
1752
1753
1754     entity row_connect is                --Connects the rows
1755     generic (N : integer);
1756     port (DIFF: in std_logic_vector(N-1 downto 0);
1757           DIV_OUT: in std_logic_vector(N-1 downto 0);
1758           NUM: out std_logic_vector(N-1 downto 0);
1759           DEN: out std_logic_vector(N-1 downto 0));
1760     end row_connect;
1761
1762
1763     architecture structural of row_connect is
1764     begin
1765
1766         lpp1: for i in N-1 downto 1 generate --Connects left side of the array
1767             NUM(i) <= DIFF(i-1);
1768         end generate;
1769         NUM(0) <= '0';
1770         lpp2: for i in N-1 downto 0 generate --Connects right side of array
1771             DEN (i) <= DIV_OUT(i);
1772         end generate;
1773
1774     end structural;
1775 -----
1776 -- Copyright 2001 by Howard University All rights reserved.
1777 --

```



```

1778 -- Design: A Connect Module that connects the FIRST TWO rows of an i
1779 -- Name:      Tiffany Crawford and Clay Gloster
1780 --
1781 -- Date:      May 2001
1782 -- Description: Generic Module
1783 -- N is the width of the inputs and outputs
1784 -- Passes div_out to den for the next row
1785 -- Shifts diff for use in the next row numerator.
1786 -----
1787
1788
1789 library IEEE;
1790 use IEEE.std_logic_1164.all;
1791
1792
1793     entity first_row_connect is                --Connects the rows
1794     generic (N : integer);
1795     port (DIFF: in std_logic_vector(N-1 downto 0);
1796           DIV_OUT: in std_logic_vector(N-1 downto 0);
1797           NUM: out std_logic_vector(N downto 0);
1798           DEN: out std_logic_vector(N downto 0) );
1799     end first_row_connect;
1800
1801
1802     architecture behavioral of first_row_connect is
1803     begin
1804
1805     lpp1: for i in N downto 1 generate      --Connects left side of a
1806     rray
1807         NUM(i) <= DIFF(i-1);
1808     end generate;
1809         NUM(0) <= '0';
1810
1811         DEN(N) <= '0';
1812     lpp2: for j in N-1 downto 0 generate    -- Connects right si
1813     de of array
1814         DEN (j) <= DIV_OUT(j);
1815     end generate;
1816
1817     end behavioral;
1818
1819
1820 -----
1821 -- Copyright 2001 by Howard University All rights reserved.
1822 --
1823 -- Design: The first row and interconnect of an integer divider
1824 -- Name:      Tiffany Crawford and Clay Gloster
1825 --
1826 -- Date:      May 2001
1827 -- Description: Generic Module
1828 -- N is the width of the numerator and denominator
1829 -- Outputs connecting to the second row are N+1 bits wide.
1830 -- Produces one bit of the quotient.
1831 -----
1832 library IEEE;
1833 use IEEE.std_logic_1164.all;
1834 --Carries out first row of division
1835 entity first_row_divider is
1836 generic (N: integer);

```

```

1837 port (NUM: in std_logic_vector(N-1 downto 0);
1838         DEN: in std_logic_vector(N-1 downto 0);
1839         NUM1: out std_logic_vector(N downto 0);
1840         DEN1: out std_logic_vector(N downto 0);
1841         Q: out std_logic );
1842 end first_row_divider;
1843
1844
1845 architecture structural of first_row_divider is
1846
1847
1848 component row_generic
1849 generic(N: INTEGER);
1850 port( NUM : in std_logic_vector(N-1 downto 0);
1851       DEN : in std_logic_vector (N-1 downto 0);
1852       DIFF : out std_logic_vector(N-1 downto 0);
1853       DIV_OUT: out std_logic_vector(N-1 downto 0);
1854       Q : out std_logic );
1855 end component;
1856
1857
1858 component first_row_connect
1859 generic (N : integer);
1860 port (DIFF: in std_logic_vector(N-1 downto 0);
1861       DIV_OUT: in std_logic_vector(N-1 downto 0);
1862       NUM: out std_logic_vector(N downto 0);
1863       DEN: out std_logic_vector(N downto 0) );
1864 end component;
1865
1866
1867 signal SDIFF1,SDIV_OUT1:
1868         STD_LOGIC_VECTOR(N-1 DOWNT0 0);
1869
1870
1871
1872
1873     begin
1874         U0: row_generic generic map(N)
1875     port map(NUM,DEN, SDIFF1, SDIV_OUT1,Q);
1876         U1: first_row_connect generic map(N)
1877     port map(SDIFF1, SDIV_OUT1, NUM1, DEN1);
1878
1879
1880     end structural;
1881
1882
1883 -----
1884 -----
1885 -----
1886 -----
1887 -- Copyright 2001 by Howard University All rights reserved.
1888 --
1889 -- Design: A block of an integer divider (not including the first row.)
1890 -- Name: Tiffany Crawford and Clay Gloster
1891 --
1892 -- Date: May 2001
1893 -- Description: Generic Module
1894 -- N is the width of the numerator and denominator (combinational logic)
1895 -- Passes DEN to DIV_OUT for the next block of the divider

```

```

1896 -----
1897
1898
1899 library IEEE;
1900 use IEEE.std_logic_1164.all;
1901 entity block_generic is
1902 generic (M, N: integer);
1903 port (NUM: in std_logic_vector(N-1 downto 0);
1904       DEN: in std_logic_vector(N-1 downto 0);
1905       DIFF: out std_logic_vector(N-1 downto 0);
1906       DIV_OUT: out std_logic_vector(N-1 downto 0);
1907       Q: out std_logic_vector(M-1 downto 0));
1908 end block_generic;
1909
1910
1911 architecture structural of block_generic is
1912
1913
1914 component row_connect
1915     generic (N : integer);
1916     port (DIFF: in std_logic_vector(N-1 downto 0);
1917           DIV_OUT: in std_logic_vector(N-1 downto 0);
1918           NUM: out std_logic_vector(N-1 downto 0);
1919           DEN: out std_logic_vector(N-1 downto 0));
1920 end component;
1921
1922
1923
1924 component row_generic
1925 generic(N: INTEGER);
1926 port( NUM : in std_logic_vector(N-1 downto 0);
1927       DEN : in std_logic_vector (N-1 downto 0);
1928       DIFF : out std_logic_vector(N-1 downto 0);
1929       DIV_OUT: out std_logic_vector(N-1 downto 0);
1930       Q : out std_logic );
1931 end component;
1932
1933
1934 type sig2d is array (0 to M) of std_logic_vector (N-1 downto 0);
1935 signal SNUM,SDEN : sig2d;
1936 signal SDIFF, SDIV_OUT: sig2d;
1937
1938
1939     begin
1940
1941         --Carries out remaining rows
1942         lpp1: for i in M downto 1 generate           --Produces an M bit Q
1943             uotient
1944             UNA: row_generic generic map (N)
1945             port map( SNUM(i),SDEN(i),SDIFF(i),SDIV_OUT(i),Q(i-1));
1946             UNB: row_connect generic map(N)
1947             port map(SDIFF(i),SDIV_OUT(i),SNUM(i-1),SDEN(i-1) );
1948             end generate;
1949             SNUM(M) <= NUM;
1950             SDEN(M) <= DEN;
1951
1952             DIFF <= SNUM(0);
1953             DIV_OUT <= SDEN(0);
1954         end structural;
1955 -----

```

```

1956 -----
1957 -- Copyright 2001 by Howard University All rights reserved.
1958 --
1959 -- Design: A generic combinational integer divider
1960 -- Name:    Tiffany Crawford and Clay Gloster
1961 --
1962 -- Date:    May 2001
1963 -- Description: Generic Module
1964 -- N is the width of the numerator and denominator
1965 -- M is the width of the quotient
1966 -- Passes dif and div_out to ouput for the next block
1967 -- -----
1968
1969
1970 library IEEE;
1971 use IEEE.std_logic_1164.all;
1972 entity generic_divider is
1973
1974
1975 generic (M, N: integer);
1976 port (NUM: in std_logic_vector(N-1 downto 0);
1977       DEN: in std_logic_vector(N-1 downto 0);
1978       DIFF: out std_logic_vector(N downto 0);
1979       DIV_OUT: out std_logic_vector(N downto 0));
1980       Q: out std_logic_vector(M-1 downto 0));
1981 end generic_divider;
1982
1983
1984 architecture structural of generic_divider is
1985
1986
1987 component first_row_divider
1988 generic (N: integer);
1989 port (NUM: in std_logic_vector(N-1 downto 0);
1990       DEN: in std_logic_vector(N-1 downto 0);
1991       NUM1: out std_logic_vector(N downto 0);
1992       DEN1: out std_logic_vector(N downto 0);
1993       Q: out std_logic );
1994 end component;
1995
1996
1997 component block_generic
1998 generic (M, N: integer);
1999 port (NUM: in std_logic_vector(N-1 downto 0);
2000       DEN: in std_logic_vector(N-1 downto 0);
2001       DIFF: out std_logic_vector(N-1 downto 0);
2002       DIV_OUT: out std_logic_vector(N-1 downto 0);
2003       Q: out std_logic_vector(M-1 downto 0));
2004 end component;
2005
2006
2007 signal SNUM1,SDEN1: std_logic_vector (N downto 0);
2008 signal TQ: std_logic;
2009 signal TTQ: std_logic_vector(M-2 downto 0);
2010
2011
2012
2013
2014 begin
2015
2016
2017     --Carries out first row

```

```

2018 U0: first_row_divider
2019     generic map(N)
2020     port map(NUM,DEN,SNUM1,SDEN1,TQ);
2021 UNA: block_generic
2022     generic map (M-1, N+1)
2023     port map (SNUM1, SDEN1, DIFF, DIV_OUT, TTQ);
2024     --Carries out remaining rows
2025     lpp1: for i in M-2 downto 0 generate           --Produces an M bit
Quotient
2026         Q(i) <= TTQ(i);
2027     end generate;
2028     Q(M-1)<= TQ;
2029
2030 end structural;
2031 -----
2032 --Copyright 2001 by Howard University All rights reserved.
2033 --
2034 --Design: Generic Register
2035 --Name: Clay Gloster
2036 --
2037 --Date: May 2001
2038 --Description: Generic Register
2039 --Width equal to one
2040 --Does not have parallel
2041 -----
2042 library IEEE;
2043 use IEEE.STD_Logic_1164.all;
2044 entity REG_GENERIC is
2045     generic (Width : integer);
2046     port (Datain : in std_logic_vector(Width-1 downto 0);
2047           Clk,Reset : in std_logic;
2048           Q: out std_logic_vector(Width-1 downto 0) );
2049 end REG_GENERIC;
2050 architecture RTL of REG_GENERIC is
2051 begin
2052     start: process(Clk)
2053     variable i: integer;
2054     begin
2055         if (Clk'EVENT and Clk = '1') then
2056             if (Reset = '1') then -- Changed Reset to Active High
2057                 for i in 0 to Width-1 loop
2058                     Q(i) <= '0';
2059                 end loop;
2060             else
2061                 Q <= Datain;
2062             end if;
2063         end if;
2064     end process;
2065 end RTL;
2066 -----
2067 --Copyright 2001 by Howard University All rights reserved.
2068 --
2069 --Design: 2 Dimensional Shift Register
2070 --Name: Clay Gloster
2071 --
2072 --Date: May 2001
2073 --Description:
2074 --
2075 --
2076 -----
2077 library IEEE;
2078 use IEEE.STD_Logic_1164.all;

```

```

2079 entity register2d is
2080 generic (Height, Width : integer);
2081 port (Datain : in std_logic_vector(Width-1 downto 0);
2082 Clk,Reset : in std_logic;
2083 Q: out std_logic_vector(Width-1 downto 0) );
2084 end register2d;
2085 architecture structural of register2d is
2086 component REG_GENERIC
2087   generic (Width : integer);
2088   port (Datain : in std_logic_vector(Width-1 downto 0);
2089         Clk,Reset : in std_logic;
2090         Q: out std_logic_vector(Width-1 downto 0) );
2091 end component;
2092
2093
2094 type anarray is array ( 0 to Height) of std_logic_vector(Width-1 dow
nto 0);
2095 signal REGIO : anarray;
2096 begin
2097   lpp1: for i in 0 to Height-1 generate
2098     IO:  REG_GENERIC generic map (Width)
2099         port map ( REGIO(i), Clk, Reset, REGIO(i+1) );
2100   end generate;
2101   REGIO(0) <= Datain;
2102   Q <= REGIO(Height);
2103
2104 end structural;
2105 -----
2106
2107
2108
2109 library IEEE;
2110 use IEEE.std_logic_1164.all;
2111 --Pipelines the divider
2112 entity pipelined_divider is
2113 generic (M : integer := 30;
2114         N : integer := 24;
2115         P: integer := 15);
2116 port (NUM: in std_logic_vector(N-1 downto 0);
2117       DEN: in std_logic_vector(N-1 downto 0);
2118       Reset,Clk : in std_logic;
2119       Q: out std_logic_vector(M-1 downto 0) );
2120 end pipelined_divider;
2121
2122 architecture structural of pipelined_divider is
2123 --- K = M/P (defines K)
2124 component generic_divider
2125 generic (M, N: integer);
2126 port (NUM: in std_logic_vector(N-1 downto 0);
2127       DEN: in std_logic_vector(N-1 downto 0);
2128       DIFF: out std_logic_vector(N downto 0);
2129       DIV_OUT: out std_logic_vector(N downto 0);
2130       Q: out std_logic_vector(M-1 downto 0) );
2131 end component;
2132
2133
2134 component block_generic
2135 generic (M, N: integer);
2136 port (NUM: in std_logic_vector(N-1 downto 0);
2137       DEN: in std_logic_vector(N-1 downto 0);
2138       DIFF: out std_logic_vector(N-1 downto 0);
2139       DIV_OUT: out std_logic_vector(N-1 downto 0);

```

```
2140     Q: out std_logic_vector(M-1 downto 0));
2141 end component;
2142
2143
2144 component reg_generic
2145 generic (Width : integer);
2146 port (Datain : in std_logic_vector(Width-1 downto 0);
2147       Clk, Reset : in std_logic;
2148       Q: out std_logic_vector(Width-1 downto 0) );
2149 end component;
2150
2151
2152 component register2d
2153 generic (height, width: integer);
2154 port (Datain : in std_logic_vector(Width-1 downto 0);
2155       Clk, Reset : in std_logic;
2156       Q: out std_logic_vector(Width-1 downto 0) );
2157 end component;
2158
2159
2160 type anothersignal2d is array (P downto 1) of std_logic_vector((M/P
2161   -1 downto 0));
2162   signal TQ,TTQ: anothersignal2d;
2163   signal SDIFF, SDIV_OUT: std_logic_vector (N downto 0);
2164 type signal2d is array (P-1 downto 0) of std_logic_vector (N downto
2165   0);
2166   signal SNUM,SDEN, SDIFF1, SDIV_OUT1: signal2d;
2167
2168   begin
2169     U0: generic_divider
2170         generic map (M/P, N)
2171         port map (NUM, DEN, SDIFF, SDIV_OUT, TQ(P));
2172
2173
2174     U1: reg_generic
2175         generic map (N+1)
2176         port map (SDIFF, Clk, Reset, SNUM(P-1));
2177
2178
2179     U2: reg_generic
2180         generic map (N+1)
2181         port map (SDIV_OUT, Clk, Reset, SDEN(P-1));
2182
2183
2184     U3: register2d
2185         generic map (P,M/P)
2186         port map (TQ(P), Clk, Reset, TTQ(P));
2187
2188
2189     lpp1: for i in P-1 downto 1 generate
2190
2191
2192     U4: block_generic generic map(M/P, N+1)
2193         port map (SNUM(i), SDEN(i), SDIFF1(i), SDIV_OUT1(i), TQ(i));
2194
2195
2196     U5: reg_generic
2197         generic map (N+1)
2198         port map(SDIFF1(i), Clk, Reset, SNUM(i-1));
2199
```

```
2200
2201     U6: reg_generic
2202     generic map (N+1)
2203     port map(SDIV_OUT1(i), Clk, Reset, SDEN(i-1));
2204
2205
2206
2207     U7: register2d
2208     generic map (i,M/P)
2209     port map(TQ(i), Clk, Reset, TTQ(i)); --Do the clock and rese
t
2210
2211
2212
2213     end generate;
2214
2215
2216
2217 lpp3: for i in P-1 downto 0 generate
2218 lpp2: for j in (M/P)-1 downto 0 generate
2219     Q( (M/P)*i+j )<= TTQ(i+1)(j);
2220 end generate;
2221 end generate;
2222 -- Q(0) <= TTQ(1)(0);
2223 end structural;
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246 -----
2247 -- Copyright 2001 by Howard University All rights reserved.
2248 -- Check for Divide by Zero Error Unit In the Floating Point Divider
2249 -- Name:      Clay Gloster and Rosie Dickens
2250 --
2251 -- Date:      May 2001
2252 -- Description:
2253 --
2254 -----
2255
2256
2257 library IEEE;
2258 use IEEE.STD_Logic_1164.all;
2259
2260 entity StageDBZ is
```



```
2261 port (Datain : in std_logic_vector(31 downto 0);
2262 Clk,Reset : in std_logic;
2263 Q: out std_logic);
2264 end StageDBZ;
2265
2266 architecture RTL of StageDBZ is
2267 begin
2268 start: process(Clk)
2269 begin
2270 begin
2271 if (Clk'EVENT and Clk = '1') then
2272 if (Reset = '1') then -- Changed Reset to Active High
2273 Q <= '0';
2274 else
2275 if ( Datain = "00000000000000000000000000000000") then
2276 Q <= '1';
2277 else
2278 Q <= '0';
2279 end if;
2280 end if;
2281 end if;
2282 end process;
2283 end RTL;
2284
2285 library IEEE;
2286 use IEEE.STD_Logic_1164.all;
2287 entity dividebyzero is
2288 port (Datain : in std_logic_vector(31 downto 0);
2289 Clk,Reset : in std_logic;
2290 Q: out std_logic );
2291 end dividebyzero;
2292 architecture structural of dividebyzero is
2293 component register2d
2294 generic (Height, Width : integer);
2295 port (Datain : in std_logic_vector(Width-1 downto 0);
2296 Clk,Reset : in std_logic;
2297 Q: out std_logic_vector(Width-1 downto 0) );
2298 end component;
2299 component StageDBZ
2300 port (Datain : in std_logic_vector(31 downto 0);
2301 Clk,Reset : in std_logic;
2302 Q: out std_logic);
2303 end component;
2304
2305
2306 signal Serror : std_logic;
2307 signal VSerror,VSQ : std_logic_vector(0 downto 0);
2308 begin
2309
2310 I0: StageDBZ port map (Datain,Clk,Reset,Serror);
2311 I1: register2d generic map(6,1)
2312 port map (VSerror,Clk,Reset,VSQ);
2313 VSerror(0) <= Serror;
2314 Q <= VSQ(0);
2315
2316 end structural;
2317
2318
2319
2320
2321
2322
```

```

2323
2324
2325
2326 -----
2327 -- Copyright 2001 by Howard University All rights reserved.
2328 -- Subtract Exponents Floating Point Divider
2329 -- Name:      Clay Gloster
2330 --
2331 -- Date:      May 2001
2332 -- Description:
2333 --
2334 -----
2335
2336
2337 Library IEEE;
2338 use IEEE.std_logic_1164.all;
2339 use IEEE.std_logic_unsigned."+";
2340 use IEEE.std_logic_unsigned."*";
2341 use IEEE.std_logic_unsigned."-";
2342 use IEEE.std_logic_unsigned.SHL;
2343 use IEEE.std_logic_arith.CONV_STD_LOGIC_VECTOR;
2344
2345
2346
2347 ENTITY SubExp IS
2348     PORT(clk          : in std_logic;
2349           reset       : in std_logic;
2350           EnR         : in std_logic;
2351           EnL         : in std_logic;
2352           Exp_A       : in std_logic_vector(7 downto 0);
2353           Exp_B       : in std_logic_vector(7 downto 0);
2354           SignA       : in std_logic;
2355           SignB       : in std_logic;
2356           Exp         : out std_logic_vector(7 downto 0);
2357           Sign        : out std_logic;
2358           RunO        : out std_logic);
2359 END SubExp;
2360
2361
2362 ARCHITECTURE RTL OF SubExp IS
2363 BEGIN
2364     PROCESS (CLK,reset)
2365         variable ExpAPrime : std_logic_vector(7 downto 0);
2366     BEGIN
2367         IF (clk = '1' and clk'event) THEN
2368             IF (reset = '1') THEN
2369                 Exp <= (Others=> '0');
2370                 Sign <= '0';
2371                 RunO <= '0';
2372             ELSE
2373                 RunO <= EnR and EnL;
2374                 IF (EnR = '1' and EnL = '1') THEN
2375                     ExpAPrime := Exp_A - Exp_B;
2376                     Exp <= "01111111" + ExpAPrime;
2377                     Sign <= SignA XOR SignB;
2378                 END IF; -- END IF Run= '1'
2379             END IF; -- END IF reset = 0
2380         END IF; -- END IF clk = '1' and clk'event
2381     END PROCESS;
2382 END RTL;
2383

```

```

2384
2385 -----
2386 -- Copyright 2000 by Howard University All rights reserved.
2387 --
2388 -- Design: 9 bit register
2389 -- Name:    Clay Gloster
2390 --
2391 -- Date:    April 2000
2392 -- Description:
2393 -- Pipelined.
2394 -----
2395
2396
2397 LIBRARY IEEE;
2398 USE IEEE.STD_Logic_1164.all;
2399
2400
2401 ENTITY Divreg10 IS
2402     PORT(Datain      : IN std_logic_vector(8 DOWNTO 0);
2403           Ck         : IN std_logic;
2404           Reset      : IN std_logic;
2405           Run        : IN std_logic;
2406           Q          : OUT std_logic_vector(8 DOWNTO 0);
2407           RunO       : OUT std_logic);
2408 END Divreg10;
2409
2410
2411 ARCHITECTURE RTL OF Divreg10 IS
2412     SIGNAL D : std_logic_vector(8 DOWNTO 0);
2413 BEGIN
2414     PROCESS
2415     BEGIN
2416         WAIT UNTIL ((Ck'EVENT) AND (Ck= '1'));
2417         IF (Reset = '1') THEN -- Changed Reset to
Active High
2418             D <= "000000000";
2419             RunO <= '0';
2420         ELSE
2421             RunO <= Run;
2422             IF (Run = '1') THEN
2423                 D <= Datain;
2424             ELSE
2425                 D <= D;
2426             END IF;
2427         END IF ;
2428     END process;
2429     Q <= D;
2430 END RTL;
2431
2432
2433 -----
2434 -- Copyright 2001 by Howard University All rights reserved.
2435 --
2436 -- Design: Floating Point Adder
2437 -- Name:    Ibrahim Sahin and Clay Gloster
2438 --
2439 -- Date:    June 2001
2440 -- Description:
2441 -- Pipelined.
2442 -----
2443
2444

```

```

2445 Library IEEE;
2446 use IEEE.std_logic_1164.all;
2447 use IEEE.std_logic_unsigned."+";
2448 use IEEE.std_logic_unsigned."-";
2449 use IEEE.std_logic_unsigned.">";
2450 use IEEE.std_logic_unsigned.SHL;
2451 use IEEE.std_logic_arith.CONV_STD_LOGIC_VECTOR;
2452
2453
2454
2455 ENTITY DivNorm1 IS
2456     PORT(clk           : IN std_logic;
2457          reset        : IN std_logic;
2458          Run          : IN std_logic;
2459          UMan         : IN std_logic_vector (29 DOWNTO 0)
2460          ;
2461          UExp         : IN std_logic_vector ( 8 DOWNTO 0)
2462          ;
2463          ZError       : IN std_logic;
2464          Result       : OUT std_logic_vector (31 DOWNTO 0);
2465          RunO         : OUT std_logic);
2466 END DivNorm1;
2467
2468 ARCHITECTURE RTL OF DivNorm1 IS
2469 BEGIN
2470     PROCESS (CLK,reset)
2471         variable ExpPrime : std_logic_vector(7 downto 0);
2472         variable TMan : std_logic_vector(22 downto 0);
2473     BEGIN
2474         IF (clk = '1' and clk'event) THEN
2475             IF (reset = '1') THEN
2476                 Result <= (OTHERS => '0');
2477             RunO <= '0';
2478             ELSE
2479                 RunO <= Run;
2480             IF ( (Run = '1') and (ZError = '0') and (UMa
2481 n(29) = '1') ) THEN
2482                 Result(31) <= UExp(8);
2483                 -- No need to adjust exponents
2484                 Result(30 DOWNTO 23) <= UExp(7 DOWNTO 0)
2485             ;
2486                 TMan := Uman(28 DOWNTO 6);
2487                 IF ( (UMan(6) = '1') AND (UMan(5) = '1') ) THEN
2488                     TMan := Tman + 1;
2489                 END IF;
2490                 Result(22 DOWNTO 0) <= TMan;
2491                 END IF;
2492                 IF ( (Run = '1') and (ZError = '0') and (UMan(29) = '0')
2493 ) THEN
2494                     Result(31) <= UExp(8);
2495                     -- We need to adjust exponents
2496                     ExpPrime := UExp(7 DOWNTO 0);
2497                     Result(30 DOWNTO 23) <= ExpPrime - "0000
2498 0001"; -- Subtract one
2499                     TMan := Uman(27 DOWNTO 5);
2500                     IF ( (UMan(5) = '1') AND (UMan(4) = '1') ) THEN
2501                         TMan := Tman + 1;
2502                     END IF;

```

```

2501         Result(22 DOWNT0 0) <= TMan;
2502         END IF;

2503         IF ( (Run = '1') and (ZError = '1') ) THEN
2504             Result <= "111111111111111111111111111111111111"; -- Div
           ide By Zero Error
2505         END IF;
2506     END IF; -- End if reset = 1

2507
2508     END IF;
2509
2510 END PROCESS;
2511

2512 END RTL;
2513
2514
2515
2516
2517
2518 -----
2519 -- Copyright 2000 by Howard University All rights reserved.
2520 --
2521 -- Design:      Floating Point Divider Core
2522 -- Name:        Clay Gloster
2523 --
2524 -- Date:       May 2001
2525 -- Description:
2526 -- Pipelined.
2527 -----
2528
2529
2530 Library IEEE;
2531 use IEEE.std_logic_1164.all;
2532
2533
2534 ENTITY FpDivCore_16 IS
2535     PORT(clk                : IN std_logic;
2536          reset              : IN std_logic;
2537          EnR                : IN std_logic;
2538          EnL                : IN std_logic;
2539          Op_A               : IN std_logic_vector(31 DOWNT0 0);
2540          Op_B               : IN std_logic_vector(31 DOWNT0 0);
2541          Op_Q               : OUT std_logic_vector(31 DOWNT0 0);
2542          EnN                : OUT std_logic);
2543 END FpDivCore_16;
2544
2545
2546 ARCHITECTURE STRUCTURAL OF FpDivCore_16 IS
2547
2548
2549 COMPONENT SubExp
2550     PORT(clk                : in std_logic;
2551          reset: in std_logic;
2552          EnR                : in std_logic;
2553          EnL                : in std_logic;
2554          Exp_A: in std_logic_vector(7 downto 0);
2555          Exp_B: in std_logic_vector(7 downto 0);
2556          SignA: in std_logic;
2557          SignB: in std_logic;
2558          Exp                : out std_logic_vector(7 downto 0);

```

```

2559         Sign : out std_logic;
2560         RunO  : out std_logic);
2561 END COMPONENT;
2562
2563
2564 COMPONENT Divreg10
2565     PORT(Datain      : IN std_logic_vector(8 DOWNT0 0);
2566           Ck         : IN std_logic;
2567           Reset      : IN std_logic;
2568           Run        : IN std_logic;
2569           Q           : OUT std_logic_vector(8 DOWNT0 0);
2570           RunO       : OUT std_logic);
2571 END COMPONENT;
2572
2573
2574 COMPONENT pipelined_divider
2575     generic (M : integer := 28;
2576            N : integer := 24;
2577            P: integer := 7);
2578 port (NUM: in std_logic_vector(N-1 downto 0);
2579       DEN: in std_logic_vector(N-1 downto 0);
2580       Reset,Clk : in std_logic;
2581       Q: out std_logic_vector(M-1 downto 0) );
2582 END COMPONENT;
2583
2584
2585
2586 COMPONENT DivNorm1
2587     PORT(clk          : IN std_logic;
2588           reset       : IN std_logic;
2589           Run         : IN std_logic;
2590           UMan        : IN std_logic_vector (29 Downto 0)
2591           ;
2592           UExp        : IN std_logic_vector ( 8 DOWNT0 0)
2593           ;
2594           ZError      : IN std_logic;
2595           Result      : OUT std_logic_vector (31 DOWNT0 0);
2596           RunO        : OUT std_logic);
2597 END COMPONENT;
2598
2599 COMPONENT RegStage8
2600     PORT(Clk          : IN std_logic;
2601           Reset       : IN std_logic;
2602           Run         : IN std_logic;
2603           Datain      : IN std_logic_vector(31 DOWNT0 0);
2604           Q           : OUT std_logic_vector(31 DOWNT0 0);
2605           RunO        : OUT std_logic);
2606 END COMPONENT;
2607
2608 COMPONENT DivideByZero
2609     PORT (Datain : in std_logic_vector(31 downto 0);
2610           Clk,Reset : in std_logic;
2611           Q: out std_logic );
2612 END COMPONENT;
2613
2614
2615
2616
2617
2618

```

```
2619 SIGNAL Exp1_2      : std_logic_vector (8 DOWNT0 0);
2620 SIGNAL Exp2_3      : std_logic_vector (8 DOWNT0 0);
2621 SIGNAL Exp3_4      : std_logic_vector (8 DOWNT0 0);
2622 SIGNAL Exp4_5      : std_logic_vector (8 DOWNT0 0);
2623 SIGNAL Exp5_6      : std_logic_vector (8 DOWNT0 0);
2624 SIGNAL Exp6_7      : std_logic_vector (8 DOWNT0 0);
2625 SIGNAL Exp7_8      : std_logic_vector (8 DOWNT0 0);
2626 SIGNAL Exp8_9      : std_logic_vector (8 DOWNT0 0);
2627 SIGNAL Exp9_10     : std_logic_vector (8 DOWNT0 0);
2628 SIGNAL Exp10_11    : std_logic_vector (8 DOWNT0 0);
2629 SIGNAL Exp11_12    : std_logic_vector (8 DOWNT0 0);
2630 SIGNAL Exp12_13    : std_logic_vector (8 DOWNT0 0);
2631 SIGNAL Exp13_14    : std_logic_vector (8 DOWNT0 0);
2632 SIGNAL Exp14_15    : std_logic_vector (8 DOWNT0 0);
2633 SIGNAL Exp15_16    : std_logic_vector (8 DOWNT0 0);
2634
2635
2636
2637 SIGNAL Run01_2     : std_logic;
2638 SIGNAL Run02_3     : std_logic;
2639 SIGNAL Run03_4     : std_logic;
2640 SIGNAL Run04_5     : std_logic;
2641 SIGNAL Run05_6     : std_logic;
2642 SIGNAL Run06_7     : std_logic;
2643 SIGNAL Run07_8     : std_logic;
2644 SIGNAL Run08_9     : std_logic;
2645 SIGNAL Run09_10    : std_logic;
2646 SIGNAL Run010_11   : std_logic;
2647 SIGNAL Run011_12   : std_logic;
2648 SIGNAL Run012_13   : std_logic;
2649 SIGNAL Run013_14   : std_logic;
2650 SIGNAL Run014_15   : std_logic;
2651 SIGNAL Run015_16   : std_logic;
2652
2653
2654 SIGNAL Result15_16: std_logic_vector (29 DOWNT0 0);
2655
2656
2657
2658 SIGNAL ZError      : std_logic;
2659 SIGNAL Op_A01,Op_B01 :std_logic_vector(23 downto 0);
2660
2661
2662
2663
2664
2665
2666 BEGIN
2667
2668
2669     Inst_Exp: SubExp PORT MAP (clk, reset, EnR, EnL, Op_A(30 DOWNT0 2
31), Op_B(30 DOWNT0 23),
2670     Op_A(31), Op_B(31), Exp1_2(7 DOWNT0 0), Exp1_2(8), Run01_2);
2671     Ints_R1 : Divreg10 PORT MAP (Exp1_2, clk, reset, Run01_2, Ex
p2_3, Run02_3);
2672     Ints_R2 : Divreg10 PORT MAP (Exp2_3, clk, reset, Run02_3, Ex
p3_4, Run03_4);
2673     Ints_R3 : Divreg10 PORT MAP (Exp3_4, clk, reset, Run03_4, Ex
p4_5, Run04_5);
2674     Ints_R4 : Divreg10 PORT MAP (Exp4_5, clk, reset, Run04_5, Ex
p5_6, Run05_6);
2675     Ints_R5 : Divreg10 PORT MAP (Exp5_6, clk, reset, Run05_6, Ex
```

```

2676     p6_7, Run06_7);
2677     Ints_R6 : Divreg10 PORT MAP (Exp6_7, clk, reset, Run06_7, Ex
p7_8, Run07_8);
2678     Ints_R7 : Divreg10 PORT MAP (Exp7_8, clk, reset, Run07_8, Exp8_9
, Run08_9);
2679     Ints_R8 : Divreg10 PORT MAP (Exp8_9, clk, reset, Run08_9, Exp9_1
0, Run09_10);
2680     Ints_R9 : Divreg10 PORT MAP (Exp9_10, clk, reset, Run09_10,
Exp10_11, Run010_11);
2681     Ints_R10 : Divreg10 PORT MAP (Exp10_11, clk, reset, Run010_1
1, Exp11_12, Run011_12);
2682     Ints_R11 : Divreg10 PORT MAP (Exp11_12, clk, reset, Run011_1
2, Exp12_13, Run012_13);
2683     Ints_R12 : Divreg10 PORT MAP (Exp12_13, clk, reset, Run012_1
3, Exp13_14, Run013_14);
2684     Ints_R13 : Divreg10 PORT MAP (Exp13_14, clk, reset, Run013_1
4, Exp14_15, Run014_15);
2685     Ints_R14 : Divreg10 PORT MAP (Exp14_15, clk, reset, Run014_15, E
xp15_16, Run015_16);
2686
2687     Op_A01(22 DOWNTO 0) <=Op_A(22 DOWNTO 0);
2688     Op_A01(23) <= '1';
2689     Op_B01(22 DOWNTO 0) <=Op_B(22 DOWNTO 0);
2690     Op_B01(23) <= '1';
2691
2692
2693
2694     Inst_Div: pipelined_divider
2695     generic map (30,24,15)
2696     port map (Op_A01,Op_B01,reset,clk,Result15_16);
2697
2698
2699
2700     Inst_N1 : DivNorm1 PORT MAP (clk, reset, Run015_16, Result15
_16, Exp15_16, ZError, Op_Q, EnN);
2701     Inst_DB : DivideByZero PORT MAP (Op_B,Clk,Reset,Zerror);
2702
2703
2704
2705
2706
2707 END STRUCTURAL;
2708
2709
2710 -----
2711 --                                     Memory Read Operation with initial V
alue
2712 -----
2713
2714 -- megafunction wizard: %LPM_ROM%
2715 -- GENERATION: STANDARD
2716 -- VERSION: WM1.0
2717 -- MODULE: altsyncram
2718
2719 -- =====
2720 -- File Name: rom.vhd
2721 -- Megafunction Name(s):
2722 --         altsyncram
2723 -- =====

```



```

2724 -- *****
2725 -- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
2726 --
2727 -- 5.1 Build 176 10/26/2005 SJ Full Version
2728 -- *****
2729
2730
2731 --Copyright (C) 1991-2005 Altera Corporation
2732 --Your use of Altera Corporation's design tools, logic functions
2733 --and other software and tools, and its AMPP partner logic
2734 --functions, and any output files any of the foregoing
2735 --(including device programming or simulation files), and any
2736 --associated documentation or information are expressly subject
2737 --to the terms and conditions of the Altera Program License
2738 --Subscription Agreement, Altera MegaCore Function License
2739 --Agreement, or other applicable license agreement, including,
2740 --without limitation, that your use is for the sole purpose of
2741 --programming logic devices manufactured by Altera and sold by
2742 --Altera or its authorized distributors. Please refer to the
2743 --applicable agreement for further details.
2744
2745
2746 LIBRARY ieee;
2747 USE ieee.std_logic_1164.all;
2748
2749 LIBRARY altera_mf;
2750 USE altera_mf.altera_mf_components.all;
2751
2752 ENTITY rom IS
2753     PORT
2754     (
2755         address      : IN STD_LOGIC_VECTOR (11 DOWNT0 0);
2756         clock        : IN STD_LOGIC ;
2757         q            : OUT STD_LOGIC_VECTOR (31 DOWNT0 0)
2758     );
2759 END rom;
2760
2761
2762 ARCHITECTURE SYN OF rom IS
2763
2764     SIGNAL sub_wire0      : STD_LOGIC_VECTOR (31 DOWNT0 0);
2765
2766
2767
2768     COMPONENT altsyncram
2769     GENERIC (
2770         address_aclr_a      : STRING;
2771         init_file          : STRING;
2772         intended_device_family : STRING;
2773         lpm_hint            : STRING;
2774         lpm_type            : STRING;
2775         numwords_a         : NATURAL;
2776         operation_mode      : STRING;
2777         outdata_aclr_a      : STRING;
2778         outdata_reg_a       : STRING;
2779         widthad_a          : NATURAL;
2780         width_a            : NATURAL;
2781         width_byteena_a     : NATURAL
2782     );
2783     PORT (
2784         clock0 : IN STD_LOGIC ;
2785         address_a : IN STD_LOGIC_VECTOR (11 DOWNT0 0);

```

```
2786         q_a : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
2787     );
2788     END COMPONENT;
2789
2790 BEGIN
2791     q     <= sub_wire0(31 DOWNTO 0);
2792
2793     altsyncram_component : altsyncram
2794     GENERIC MAP (
2795         address_aclr_a => "NONE",
2796         init_file => "C:/Documents and Settings/Deepayan/Desktop/dee
2797 p.mif",
2798         intended_device_family => "Stratix",
2799         lpm_hint => "ENABLE_RUNTIME_MOD=NO",
2800         lpm_type => "altsyncram",
2801         numwords_a => 4096,
2802         operation_mode => "ROM",
2803         outdata_aclr_a => "NONE",
2804         outdata_reg_a => "CLOCK0",
2805         widthad_a => 12,
2806         width_a => 32,
2807         width_byteena_a => 1
2808     )
2809     PORT MAP (
2810         clock0 => clock,
2811         address_a => address,
2812         q_a => sub_wire0
2813     );
2814
2815
2816 END SYN;
```