2016

An Efficient FPGA Implementation of Optical Character Recognition System for License Plate Recognition

Yuan Jing
University of Windsor

An Efficient FPGA Implementation of Optical Character Recognition System for License Plate Recognition

by

Yuan Jing

A Thesis
Submitted to the Faculty of Graduate Studies through the
Department of Electrical and Computer Engineering in Partial Fulfillment
of the Requirements for the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

An Efficient FPGA Implementation of Optical Character Recognition System for License Plate Recognition

by

Yuan Jing

APPROVED BY:

Dr. Rupp Carriveau
Civil and Environmental Engineering

Dr. Roberto Muscedere
Electrical and Computer Engineering

Dr. Mitra Mirhassani, Advisor
Electrical and Computer Engineering

August 18^{th}, 2016

Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyones copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Optical Character Recognition system (OCR) has been found very useful in the field of intelligent transportation. In this work, a FPGA-based OCR system aimed at image-based License Plate Recognition (LPR) has been designed and tested. A feed-forward neural networks has been chosen as the core of the proposed OCR system. The neural network parameters are acquired beforehand and will not change during its operation time. A set of Matlab programs have been made in the network's design process. The verification process includes Matlab simulation where programs using binary numbers which has the same representation format as the system to compute the results, and Modelsim simulation where data is computed and transferred between modules under clock signals' control. The synthesis process is done in the Altera's FPGA design software - Quartus II. The result shows that calculation speed of the system implemented in hardware is much faster than software running on a PC while it maintains a high recognition accuracy. The proposed image recognition system is used with a set of images that are generally difficult for such networks to handle. Most images include shadows and other imperfections in the image. The proposed network was able to achieve 95% accuracy in recognizing the correct character despite the image imperfections. Moreover, it takes advantage of very compact and efficient

non-liner sigmoid function.

Dedication

Dedicated to my beloved parents.

Acknowledgments

While I was working on this thesis I received numerous help from many people and I would like to take this chance to express my thankfulness. I received valuable advices and help from my fellow lab members Babak, Iman, Bahar and Rose and they are very good persons to work with. I also want to thank members of my committee Dr. Rupp Carriveau and Dr. Roberto Muscedere for their valuable time and feedbacks. Last but not least, I want to extend my sincere appreciation to my supervisor Dr. Mitra Mirhassani who guided me and helped me all this time. Thank you.

Contents

List of Figures

List of Tables

List of Symbols

net_i^j : The summation of the multiplications of inputs and corresponding weights of ith neuron in jth layer.

act_i^j : The output of ith neuron in jth layer.

w_{ki}^j : the weight between ith neuron to the kth output in the previous layer.

b_i^j : The bias of ith neurons in jth layer.

$numE$: The number of exponential bits.

$numM$: The number of mantissa bits.

$tanh$: Hyperbolic tangent function.

e_a : Approximation error.

e_q : Quantization error.

N_{one} : The place of first non-zero bits in a binary number(except the sign bit).

x_{paq} : The boundary between pass region and processing region.

x_{sq} : The boundary between processing region and saturation region.

N_i : The number of integer bits.

N_f : The number of fractional bits.

N_{sub} : The number of smallest intervals in a subregion.

Glossary

OCR : Optical Character Recognition

FPGA : Field Programmable Gate Array

LPR : License Plate Recognition

LPD : License Plate Detection

CS : Character Segmentation

CCA : Connected Component Analysis

SVM : Support Vector Machine

HMM : Hidden Markov Model

ANN : Artificial Neural Network

CPU : Central Processing Unit

FXP : Fixed-Point

FLP : Floating-Point

LUT : Look-Up Table

PWL : Piece-Wise Linear

MSB : Most Significant Bit

LSB : Least Significant Bit

SOP : Sum Of Product

MAC : Multiply-ACcumulate

LE : Logic Element

PLL : Phase-Locked Loop

RAM : Random Access Memory

MUX : Multiplexer

RTL : Register Transfer Level

Chapter 1

Introduction

Automatic License Plate Recognition has been applied in applications such as automatic toll collection, traffic control and monitoring, as well as parking lot control and access [6, 26, 16]. These systems are becoming more important as the automotive industry moves toward autonomous driving and smart roads. In these applications, the system should be able to find and recognize the characters on the plate and be operational in a variable environment. The road and weather condition, as well as lighting, creates numerous challenges for these systems. Although the task is very easy for humans, a fully automated system with full recognition, in the presence of non-ideal environmental conditions, is not yet realized.

1.1 Problem Statement

1.1.1 Challenges

The input characters that are fed to an LPR are not as complicated as those in other applications such as handwriting or facial recognition application. However, the input image still creates challenges for the recognition processing [31].

Due to its application, the location of a plate is not at the same place between different images. The cars' relative location to the camera can differ. Therefore the camera has to look for a plate in all areas of a picture. Moreover, an image might contain one or multiple plates. Each of these plates has to be processed and recognized by the system.

Other issues include changes in font, color and background images on a plate. For example, plates can be different in color and font, and may contain a logo, symbols and other features that generally are not present in regular plates. Even standard plates differ between various provinces as well as countries.

In addition to these, dirt or obstructive material on the plate can cause the system to fail to detect and recognize the characters. Environmental issues such as image background prevent most systems to be adopted for in an ad hoc manner. Each system has to be installed, trained, and tested which in turn increases the total cost and complexity of the scheme.

Another important variation that can cause problems is the quality of the license plates. License plates in poor conditions can make the input characters very noisy. Therefore, the capability of recognizing noisy or deformed input can be another element in causing the system to fail or make mistakes.

Most systems cannot resolve all of the features with cognitive computing resources. Moreover, most of the works reported in the literature are based lab test, and image databases are pre-processed and uniform. The contribution of this research work is

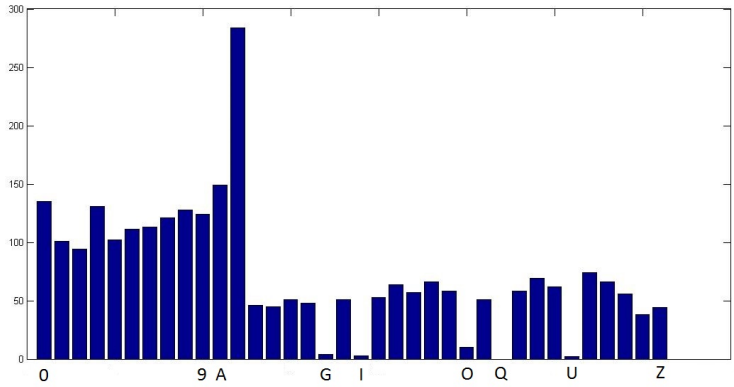

Figure 1.1: Frequency of characters in the dataset

that the system is not limited and focuses on using images that have various font colors, and illumination and lighting are non-uniform. In the next section, general LPR system and its sub-blocks are introduced.

1.1.2 Dataset

The images of license plates used in this work are taken in Ontario, Canada. The latest version of Ontario's license plates uses the font of 'Driver Gothic'. The dataset contains many images of segmented characters. These images are acquired from 386 images taken at parking lots at different locations. The images of characters are segmented from the images containing whole license plates. Each license plates contains six to eight characters. The letter 'I' and the number '1' will be considered as the same character. Moreover, the letter 'O' and the number '0' are the same. The frequency of appearance of alphabetical characters in the dataset is shown in Fig. 1.1. It is a common fact that not all the characters are used in LPR systems [33]. In general, most license plates avoid using both of these characters on the same plate. The system in this work will keep all the four outputs, as these characters can look different and can frequently be used in license plates from other regions. When the

recognition accuracy is calculated, it will be considered as a correct recognition for both '0' and 'O'.

1.1.3 Objectives

The system designed in this thesis is the decision-making system of an Optical Character Recognition system for License Plate Recognition applications. The inputs to the system are images of characters resized to the same size uniformly. The system is supposed to be able to determine the input character.

The aim of this work is to implement an FPGA-based feedforward neural network which is suitable for a particular classification problem. The FPGA gives a popular platform for this implementation task. The flexibility of designs in FPGA makes the network capable of dealing with similar problems with minor modification of conception. More importantly, the hardware-based neural network tends to have a faster processing speed with similar or fewer hardware resources compared with PC-based neural networks implementations. In software-based ANNs like PC-based neural networks, neurons are modeled by programs running on general-purpose CPUs. However, the PC-based model is running on the processors that executing instructions sequentially, which essentially in contradiction with the parallelism of neural networks. FPGA is chosen as the platform for the implementation of ANN for this work. The basic processing units in FPGAs give them the parallel data processing ability.

1.2 General System Configuration

Figure 1.2 is a general block diagram of a typical License Plate Recognition (LPR) system.

The images can be taken by a color, black, and white, or an infrared camera. Most parts of the picture information will be discarded after the plate detection. Therefore

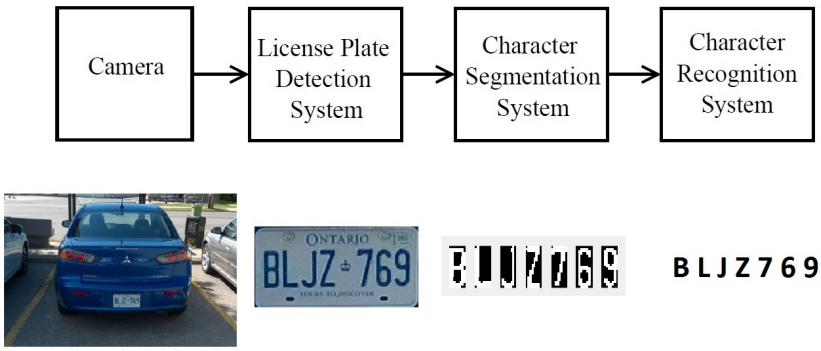

Figure 1.2: General block diagram of a License Plate Recognition system

using a very high-resolution camera is discouraged in most applications.

The task of the License Plate Detection (LPD) system is to determine if the input image contains the image of a license plate and to find its location on that picture. At this stage, the unnecessary parts of the image are discarded, and the system focuses on the essential component of the image that contains the plate(s). An optimal system should be able to detect the presence of a license plate in an image from all the background information, as well as overcome the challenges in the image.

Next gate is the Character Segmentation (CS) system, which takes the image of a plate and separates the characters from each other. At this stage, elements such as dirt, shadow, angled image, and vanity plate backgrounds can cause issues for the system. The outcome of the CS system is only to separate the characters from each other. Recognizing them is the job of the next block which is the focus of this research.

The Optical Character Recognition (OCR) is an integral part of LPR system. Efficient hardware implementation of the OCR is the center of this study, and will be explained in detail in later chapters. In the rest of this section, a brief overview of the LPD and the segmentation systems is provided. It should be noted that these

units were not implemented in this research, and results from simulations were used in order to feed the hardware implementation of the OCR system.

1.3 Overview of License Plate Detection System

There are a number of works available in the literature that deal with various aspects of a License Plate Detection. In order to implement an LPD, different types of features are used to detect the license plate area from the background. Some of the most common features used in these works include color, shape and texture [27, 4, 31].

In [27] color is used as the feature to detect license plates, while [4] used the geometric character of the license plates to detect them. However, These methods are not very reliable since the desired background color of a license plate or the rectangular shape of the license plate can be absent because of the complex conditions in the real environment.

In comparison, methods that rely on features such as the texture of the license plates are more reliable [31]. Vertical gradient has been found to be a very efficient method to select license plate areas. The license plate area when compared with the background, is a vertical edge intensive area due to the present of characters. Methods using vertical edges [31] usually convert the images into a gray-scale image, then the vertical gradient is computed for the whole picture. A filter is then applied to the result to select the area with the highest vertical edges. The threshold for selecting the highest range can be adaptive, and the result may contain several candidates. By setting a threshold or aspect ratio of the size of license plates, the non-license plate areas will be discarded, and the license plate area will be acquired.

1.4 Overview of Character Segmentation System

When the presence of a license plate is detected in an image, the characters on the plate has to be separated from each. The Character Segmentation System has to acquire the full image of a license plate and to deliver a set of separated characters to the next block. Different methods are proposed in literature for implementation of the CS system [25, 7, 13].

Vertical and horizontal projection [25, 7] are common methods in Character Segmentation. Because of the gap between each character on the plate, the vertical projection of the license plate image will observe a low-value region and characters can be separated. However, this method can not process images with tilted license plates. Since the relative position between a license plate and the camera can vary, all characters in a picture will not always appear on a horizontal line. Therefore the gap between the adjacent characters is no longer vertical.

This problem can be solved by a rotation correction done by the Hough transform [13]. However, this method is computationally intensive.

The Connected component analysis(CCA) is a useful method for segmentation method. The CCA analyses the image, and all adjacent pixels holding certain value or above will be considered as one component. By applying the CCA to the image of the license plate, each character will be seen as a part. All characters will be acquired by filtering all of the elements through a threshold value such as the size of a component.

1.5 Overview of Optical Character Recognition System

Both the detection and segmentation systems act before the OCR are the necessary pre-processing steps. This thesis focuses on the hardware design and implementation of the OCR portion of the scheme.

Many different classification methods have been proposed for the OCR applications [11, 5, 19]. Classification is a process to that separates the observations into different, predefined categories based on the features of the observations [11]. This is a restricted definition of general classification but is similar to the supervised learning in machine learning field.

The classification rules in statistics are called statistical models. From the concept that whether or not there are a clearly defined statistical models, classifiers can be categorized into statistical classification methods and machine learning classification methods [5].

Statistical classification methods tend to look at the classification problem as a statistic process. The other methods tend to emphasize on the performance of the OCR system other than understanding the underlying mathematical model. Common statistical classification methods used in OCR include support vector machines (SVM)[19], hidden Markov models (HMM), and template matching. On the other hand, artificial neural networks(ANN) can be seen as machine learning classifiers.

Table 1.1 provides a high-level comparison between several OCR systems. There is no significant difference in the recognition success rate between the various methods. However, due to the different conditions that testing data are acquired and different size of the testing dataset, a simple comparison between table values is not sufficient. More analysis and tests are required to verify the success rate of a certain method when used for any specific application.

References	Character Recognition Method	Recognition Accuracy	Processing Time
[29]	HMM	95.2%	0.1s
[10]	HMM	95.7%	Not reported
[28]	HMM	97.5%	0.1s
[14]	Template Matching	95.7%	Not reported
[17]	Template Matching	97.3% *	0.9s
[30]	SVM	97.2%	less than 1s
[8]	ANN	97.7%	not reported

Table 1.1: Comparison of different classification methods in OCR

* The OCR recognition accuracy is not provided by the author. The value used here is estimated based on the overall success rate and the success rate in character segmentation step.

Another issue that needs to be considered is the complexity and computation requirements of different methods. It is critical to have an algorithm with less computational complexity for hardware implementations of the OCR system. The algorithms used for the OCR usually contain two parts; feature extraction and classification [31, 5, 35]. The extra step for performing the feature extraction requires more arithmetic operations which mean more adders and registers in hardware. The success of the SVM methods and HMM methods is heavily based on a good selection of extracted features. On the other hand, ANN and template matching methods can process the raw data from the character segmentation steps.

Template matching methods calculate the Euclidian distance or Hamming distance between pixels in testing input and templates. The ANN method can use all of the pixel values from the input images directly as its inputs. Apart from complicated arithmetic operations that template matching method requires, it also needs a significant number of temples in different conditions in order to be able to recognize characters in all different conditions correctly. Taking all these factors into consideration, ANN was chosen as the classification method, and an FPGA implementation

of it was developed.

1.6 Thesis Organization

The thesis is organized as follows. Chapter two discussed the feedforward neural networks - the OCR method in this work. An introduction of feedforward neural network as well as the procedure of determining the structure and size of the feedforward neural networks in this work are given. In chapter three, issues in implementing the network in hardware are discussed. This includes the arithmetic format used in this system, implementation of the activation function and the bit width of the numbers in this system. Chapter four implements the system on a FPGA. The parallel scheme as well as the design of data path and control path of the system are given in this chapter. The system is a synchronized system. Using different clock signals in different blocks is a critical design process for realizing parallelism and pipelining in the system. RTL simulations are given in chapter five. The performance of this system is compared with other hardware and software OCR systems. In chapter six, the conclusion is made. The contribution of this work and future works are discussed.

Chapter 2

Feedforward Neural Network

Feedforward neural network belongs to a class of artificial neural networks. In order to implement the OCR section a feedforward Neural Network is used. This type of network is inspired by the biologic neural network even though they are very different in terms of their essential characteristics.

The basic unit in an ANN is a neuron, whose name came from the cells in a nerve system. An artificial model of a neuron in an ANN is more computationally complex than its biological counterpart. In this chapter, the theory of the feedforward neural networks and characteristics of artificial neurons are reviewed. Design criteria, limitations, and used process for the proposed network are provided. These are used to set the network size and the required number of neurons in each layer. Simulation results of the network during the training and recognition stages using Matlab are presented in this chapter.

2.1 Introduction of Artificial Neural Networks

Like machine learning, ANNs' ultimate goal is to bring intelligence to machines [11]. That is to let machines have consciousness instead of merely giving a response to the stimuli. Artificial Intelligence has many achievements and is applied in many applications in different areas such as image recognition and speech recognition[21]. However, there is still a long way to go before machines can act, make decisions, or think like a human.

Despite the astonishing computing power and the high computation accuracy, computers lack some of the critical abilities such as adaptability due to experience. The ANN is provided by this feature and has been used in applications that require interaction with humans such as advertising. The main difference between the ANNs and a conventional computer is that an ANN is composed of massively parallel yet simple computing units [23].

2.1.1 History of Artificial Neural Network

In 1949 Donald Hebb proposed a theory in his book [15] of how the neuron works based on his observation. The idea is that the more stimulation, the stronger connection between these two neurons. Based on this he proposed the Hebbian learning rule.

The first practical implementation of ANN is the perceptron which was invented in late 1950's by Frank Rosenblatt. The perceptron networks are good at classification problems. However, the classes must be linearly separable. This problem is well recognized after Marvin Minsky, and Seymour Papert pointed it out in their book [22] published in 1969.

After that, the research of ANN started to cool down. For a long time, many researchers tried to find a way to overcome this challenge. In the late 1960's when the backpropagation algorithm was invented by Arthur E. Bryson and Yu-Chi Ho,

implementation of multilayer feedforward neural networks became possible[3]. This method started to draw attentions by its reintroduction by Paul J. Werbos, Geoffrey E. Hinton, and Ronald J. Williams in 1980's [24].

The back-propagation learning algorithm enables the network architectures to have multi-layered structures. Hence these are far stronger than the perceptron networks. The multi-layered networks can theoretically model any non-linear functions in any degree of complexity. The back-propagation training algorithm is a significant step forward for the research of ANN and made the implementation of network structures possible. Nowadays, the deep learning techniques make feedforward neural networks more powerful in solving many problems. The training process will be done in anther platform, which makes it an off-line version of ANN. The reason for this is that for the same application, there is no need to train the network once the network is properly trained. Also, the training phase compared with the recall phase is much more computationally intensive, requires a lot of memory. FPGA may not be capable of doing the training process since the gradient calculations involved in the training process. Some researchers have proposed new learning algorithms which are different from the gradient calculation based method, and suitable for the FPGA platform. However, finding a suitable ANN solution for the real life complex problem sometimes need the trial and error technique, and many different architectures are required to be tested in the companion of the corresponding training procedures. Feedforward neural networks are data-driven applications of supervised training. Training data is crucial to the training phase, and how good the training phase is done determines how well the network will perform. In this chapter, the theory of a feedforward neural network, including the structure of its core units - neurons and the training process is discussed. The dataset which consists of training data and testing data are described here. Finally, the process that how the network size is chosen is given.

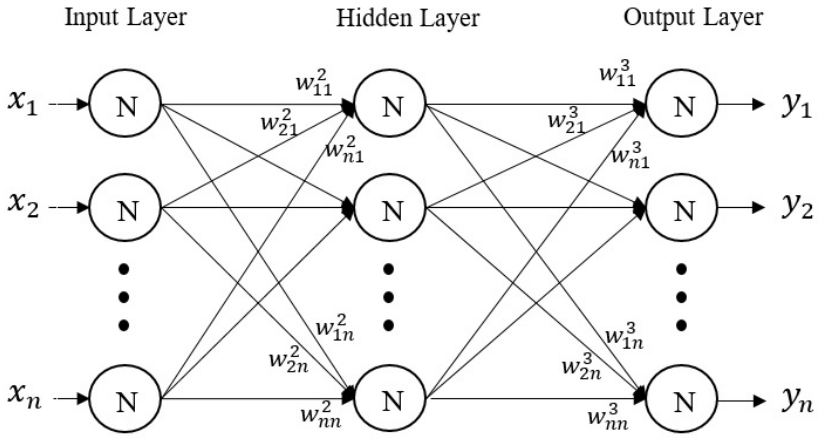

Figure 2.1: Generic structure of a 3-layer feedforward network

2.2 General Theory of Feedforward Neural Network

The general structure of a feedforward neural network is showed in Figure 2.1. Each circle in the figure represents a neuron. A feedforward neural network consists of many neurons; these neurons form the layers in neuron network. There is no data transfer between neurons in the same layer; neurons only send data to neurons in the next layer. Feedforward neural network is a widely used type of neural network structure because of its relatively simple structure and effectiveness in classification problems.

2.2.1 Forward Propagation Phase

The outputs of the feedforward neural network can be acquired by calculating the outputs of all neurons layer by layer. This process of calculating the outputs of a neural network is the forward propagation phase. Since the neuron is the basic structure of the network, it is worth to take a closer look at the structure of a single

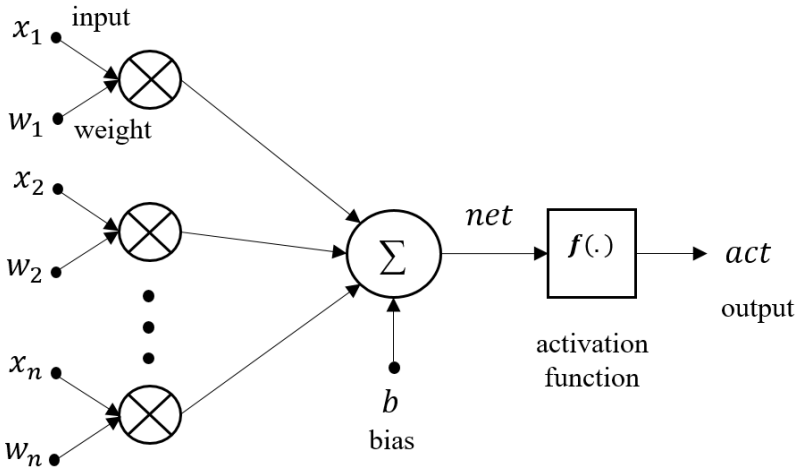

Figure 2.2: Computation model of an neuron in an artificial neural network

neuron.

Figure 2.2 shows the structure of a neuron. Each neuron in a multilayer feedforward neural network computes a weighted sum of its inputs. These inputs are outputs of all the neurons from the previous layer. The neuron's output is the weighted sum regulated by an activation function. The computation of each neuron's output can be expressed as

$$act_i^j = f(net_i^j) \tag{2.1}$$

where f is the activation function, and act_i^j is the output of the activation function of the ith neuron in the jth layer, and net_i^j is the summation of the multiplications of inputs and corresponding weights of ith neuron in jth layer.

The value of the net_i^j can be obtained as follows:

$$net_i^j = \sum_{k=1}^{m} act_k^{j-1} w_{ki}^j + b_i^j \tag{2.2}$$

where m is the number of neurons in the $(j-1)^{th}$ layer, b_i^j is the bias of i_{th} neurons in j^{th} layer, and w_{ki}^j is the weight between this neuron to the k^{th} output in the previous

layer.

2.2.2 Training Phase

Most neural networks and in particular feedforward networks can not be used before the networks have been properly trained. Feedforward neural networks are trained by the supervised learning algorithms, where input-target pairs will be utilized during the training phase. A cost function is used to indicate how well the training is done. The cost function used in training phase is a mean square error function:

$$E = \frac{1}{2}\sum_{i=1}^{n}(act_i - t_i)^2 \tag{2.3}$$

where n is the number of neurons in the output layer, act_i and t_i are the actual output and ideal output of the network respectively.

This cost function represents the difference between the ideal and actual outputs. The training of the neural network is a process of adjusting neural network parameters in such a way that the real output is closer to ideal output. The study shows that the feedforward neural networks will get the best performance by minimizing this cost function.

$$E(w_{new}) = E(w + \Delta w) \tag{2.4}$$

Using the Taylor series, an estimation of this equation can be obtained:

$$E(w + \Delta w) = E(w) + \frac{\partial E}{\partial w}\Delta w \tag{2.5}$$

To minimize E we need to minimize $\frac{\partial E}{\partial w}\Delta w$. $\frac{\partial E}{\partial w}$ is the gradient which is an n-dimensional vector pointing to the directions which make the function E increases the most. Due to the fact that the dot product of two vectors will result in a smaller value if both vectors pointed to the opposite directions while they have the same length, the above equation can be modified as follows:

$$\Delta w = -\alpha\frac{\partial E}{\partial w} \tag{2.6}$$

where α is a coefficient that controls the steps of learning.

The training technique for feedforward neural networks is called backpropagation, which calculates the partial derivatives using the chain rule.

2.3 System Description

The training phase will be done in the design process, and the system in hardware will only perform the forward propagation phase. The value of network parameters will be calculated in software and sent to the memories in the hardware system.

In designing the feedforward neural network, Matlab has been used as the design software. Its powerful computation ability and versatile design approaches make it very suitable for the task. Also, the neural network toolbox integrated into Matlab makes the training of the networks much faster and the design time will be greatly shortened. The network design process in Matlab will be two parts, first is to use Matlab design a feedforward neural network, during this process all the network parameters will be determined, this includes the size of the network and the size and format of the input data. The second step is to verify the simulation results using Matlab. However, the codes are specially designed so that all the operations are done in binary format to simulate the computations in FPGA.

2.4 Determining the Size of the Neural Network

The number of nodes in the second layer is used to increase the network accuracy. However, a larger number of nodes in the network requires more data for it's supervised learning phase. No equation can provide the correct size of a neural network. Therefore in this section, simulations are used to create an early estimate of the network size, using Matlab simulations. Using the images in the dataset a Matlab cost function for a three-layer feedforward network was formed to determine the most

efficient size for the network.

2.4.1 Three Layers Versus Four Layers

It has been claimed in [18] that a three-layer feedforward neuron network with at least one hidden layer is able to model any systems with any degree of complexity. However, feedforward neural networks with more than 3 layers can also be used in classification applications. In this section, structures of feedforward neural networks with 3 layers and 4 layers will be compared in terms of performance and computation intensity.

The optimized structure of a 3-layer feedforward neural network for this application whose determining process is described in this chapter has a structure of 189-160-36. These numbers are the numbers of nodes in the input layer, 2nd layer and output layer, respectively. A feedforward neural network with 4 layers is used to compare with. The two networks will have the same number of hidden neurons and to make the design simple, the two hidden layers of the 4-layer feedforward neural network will have equal number of neurons. Therefore, the structure of the 4-layer feedforward neural network is 189-80-80-36.

Because of that the inputs are 1-bit binary numbers, the real multiplications between input signals and their corresponding weights of the neurons in a layer happens in the third layer and the fourth layer. The number of multiplications needs to be done in a complete recognition process for one character is 160×36 for the 3-layer network and $80/times80 + 80/times36$. That is a 61.11% more multiplications need to be performed for the 4-layer network.

On the other hand, the performance of the network is another concern. Table 2.1 listed the recognition accuracies of feedforward neural networks with different structures. The first three rows are the networks with 4 layers and the last row is the network with 3 layers. As the number of neurons in the hidden layer increases, the

Network structure	Recognition accuracy
189-60-60-36	95.16%
189-70-70-36	96.15%
189-80-80-36	96.34%
189-160-36	96.81%

Table 2.1: Recognition accuracies of 3-layer and 4-layer feedforward neural networks

performance get better. However, even though the network with the same number of hidden neurons has a performance which is less than its 3-layered counterpart. Considering the reasons discussed above, the proposed system in this thesis will have a three-layered structure.

2.4.2 Number of Inputs to the First Layer

Trained by a supervised learning algorithm, the feedforward neural network is a data-driven classifier. Its performance is highly related to the format of input data. Moreover, the features used for this application are all the pixels from the re-sized, and segmented images. Therefore the size of the network, or more specifically, the size of the input layer of the network depends on the size of input vectors if every image is treated as a vector.

The used camera in this work was set around 2 to 3 meters away from the license plate when the images were taken. The input of the neural network depends on the resolution of character images for a fixed size network. Therefore, all of the character images were resized to a uniform value. The expected size of a character is 56×23, with the aspect ratio(height divided by width) equal to 2.368. These values were derived from the average resolution images. The sample size is 10% of the dataset. However, the number of the nodes in the first layer will be equal to the resolution of the input images if each input is a pixel from the character images. Therefore,

Reference	[6]	[9]	[33]
Input Size	12×9	16×16	34×22

Table 2.2: Input size of several feedforward neural networks used for LPR application

this resolution is quite large, and it will inevitably lead to a larger size for the neural network. The size of the network is a critical aspect which needs to be considered especially for hardware implementation.

Table 2.2 listed the sizes of input layer of feedforward network in some LPR systems. Clearly 56×23 is a resolution larger than any of these. Such big resolution will lead to a big network whose memory and multiplier requirements can be hardly supported by a single FPGA chip with high parallelism in its computation process. Therefore, a resize process for the raw character images was necessary and a smaller resolution for the input characters needs to be determined.

A much smaller resolution can still be able to represent all the English letters and Arabic numbers. However, smaller resolutions require higher image quality in order to be recognized correctly. In other words, smaller resolutions are more sensitive to noises and deformed images. Images with relatively more complex strokes such as 'H', 'M' and 'W' are easily become impossible to be recognized in small resolutions with some noises. So the desired size of input image should be less than 56×23 at the same time not too small to recognize character images with noise. So the rules to choose a smaller size of input images are first, keep the same aspect ratio to keep the characters the same as the original images as possible. The aspect ratio found in 10% of the samples of the data set is 2.34. Among the resolutions fall in the range of resolutions similar to those listed in Table 2.2, 14×6, 21×9, 28×12 and 35×15 have the closest aspect ratios to 2.34. The last resolution will lead to an extensive network which is hard to be implemented in a single FPGA. Therefore, only the first three resolutions will be considered to be as the possible resolutions. The second

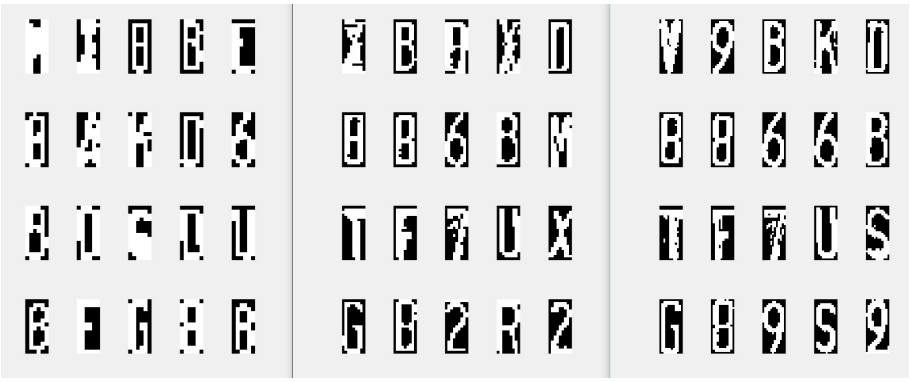

Figure 2.3: Characters in different resolutions

rule is to leave enough space between vertical strokes. Figure 2.3 list characters in these resolutions. Characters on the left and middle and right have the resolutions of 14×6, 21×9 and 28×12, respectively. It can be seen that some characters in the resolution of 14×6 lost the details that are crucial to the correct recognition while the other characters generally keep the details.

Therefore, the second resolution which is 21×9 is chosen, and all input characters will be resized to this resolution, and input layer has a size of 189.

2.4.3 Number of Nodes in the Second Layer

An experiment was done to test the performance of the networks with different sizes in order to determine the network's optimal size. In this experiment, the character images in 96 out of 386 license plate images were used as the testing input. The remaining 290 license plate images were used as the training data.

In this setup, the number of nodes in the second layer was set the variable component. The experiment tested the performance of networks with a various umber of nodes in the second layer. The range of the number of nodes for the second layer was limited from 100 to 200 with a step of 20. This range is broad enough to cover the optimal number.

For each condition, the network was trained, and its performance was calculated 10 times. The average value was used for comparison purposes. Figure 2.4 shows the recognition accuracy of different network settings.

In Figure 2.4 the horizontal axis is the number of nodes in the second layer. It can be seen that the recognition accuracy increases with the number of neurons in the second layer, and it reaches its highest point where the number of neurons is 160. From this result 160 is the most suitable number of nodes for the network's second layer. There must be some errors as only 10 samples are used to estimate the expectation of the population. However, the trend will be the same.

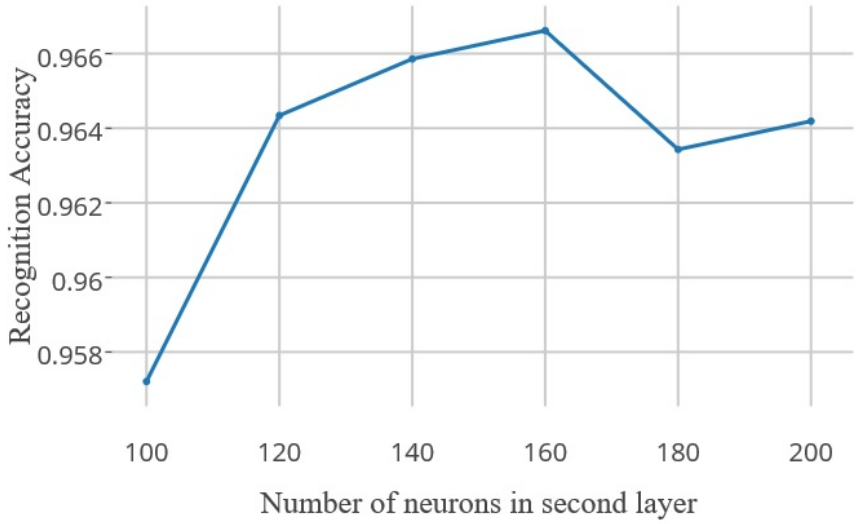

Figure 2.4: Recognition accuracies of networks with different sizes

2.5 Network's Training and Testing

In this section, the network's performance will be evaluated after the network is trained. The date used for training is necessary for the network's performance. Some artificial training data is crafted from pictures of characters in the same font.

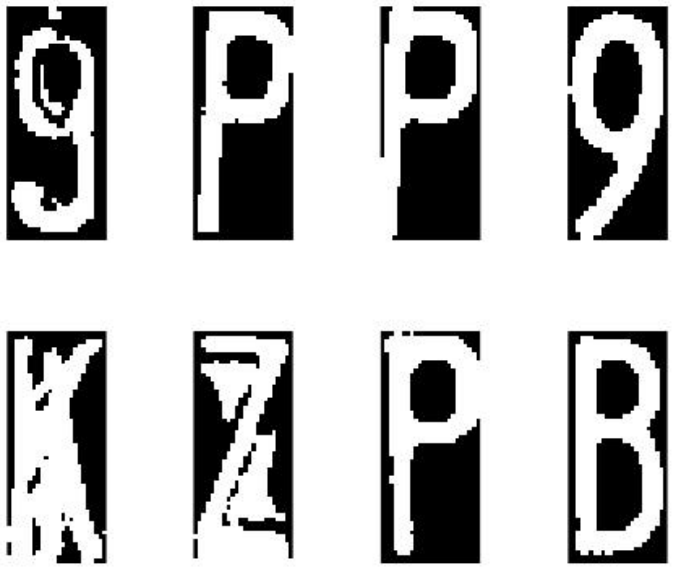

Figure 2.5: Samples of characters in the dataset

The difficulty in recognition task comes from two elements; one is that some characters have very similar shape. The second element is the distortion. These two problems can be alleviated to some degree by training the network with more data. Thus, a training set with enough training data is essential for a successful training.

For an ANN, the number of required training data is related to the number of parameters it has. A significant input layer has more parameters to train, and that requires a larger dataset of training data. Moreover, as it can be seen from Figure 1.1, the frequencies of letters appeared in all images are far less than those of numbers. Although some letters are not prohibited in license plates, but they only appear in some special cases. For example, 'G' and 'U' only appeared less than 5 times in all samples. Therefore, there is no chance for the feedforward neural network to be able to recognize these characters successfully with such a small size of the training dataset.

2.5.1 Recognition Accuracy of the Network

When the optimum size for the feedforward network was determined, the network was trained and tested with the dataset. This feedforward neural network is a three-layered network with 189 inputs, 36 outputs and 160 neurons in a hidden layer.

The first layer or input layer's neurons only send input data to the hidden layer. The neurons in the second layer or the hidden layer compute the functional value of the weighted sum of input data. The neurons in the third layer or output layer compute the final output of the network.

Usually, the activation functions for a neural network are the same in neurons in the same layer. Functions that are typically used as activation functions include logistic function, linear function, and hyperbolic function. Experiments show that for this classification problem, hyperbolic functions are the best choice for both neurons in the hidden layer and output layer. In this thesis neurons in hidden layer and output layers use the hyperbolic tangent function as the activation function.

The dataset including images created from ideal character images was divided into training data and testing data. The test data was only used to verify network's performance after the training is finished. In other words, the network has not seen the testing data during the training phase.

To choose the images for the training data, 10% of images were randomly selected and marked as the validation images. During each iteration, validation images were processed by the network. However, these results were not used to guide the network and were only used to compare network's performance through iterations.

If the error kept growing for a certain number of iterations, then the training was stopped to prevent the problem of over-fitting. This technique is particularly useful when the size of training data is small compared with the number of parameters that the network has. In here the maximum value 6 was used.

Because of the limited size of the dataset, all the data used in network design

process were reused in network verification process. In order to keep the verification result as convincing as possible, a n-fold cross-validation was used for testing network's performance.

Considering the size of the available dataset, a $n = 4$ is used here. A total of 386 images was divided into four groups, each containing 96, 96, 96, and 98 images respectively. Each group was used as testing data and the network's performance over all groups was averaged. During each cluster as testing data, the rest data in the dataset was used as training data. In this way, the network can be tested using all samples in the dataset.

The learning algorithm used in this case is scaled conjugate gradient, which is a technique to solve optimization problems. This algorithm is suitable for optimization problems with a large amount of parameters such as the feedforward neural network in this thesis.

Every time network's parameters are initialized with different values, and the network's recognition accuracy varies from time to time. As the starting point changes every time, it will settle down at the various minimum point at the surface. The highest recognition accuracy for different initializations can not represent this network's recognition accuracy because the network can't see the testing data during the training phase. Once the training phase is done, the network's parameter will not change while it is processing the data.

Data created from the method described in the previous section will be used as a supplement to the testing data from the dataset. In order to show the effectiveness of the supplementary training data, a comparison of recognition accuracy of a network which is trained by different training datasets, will be presented.

Table 2.3 shows the results of the recognition accuracy of a network trained with various training data. The numbers in the second row are the accuracy rate of testing the network with training data. The values shown in the last row are the recognition

Testing Data	Group1	Group2	Group3	Group4	Average
recall accuracy	0.9927	0.9960	0.9942	0.9938	0.9942
recognition accuracy	0.9833	0.9681	0.9653	0.9856	0.9756

Table 2.3: Recognition accuracies from different groups of training data

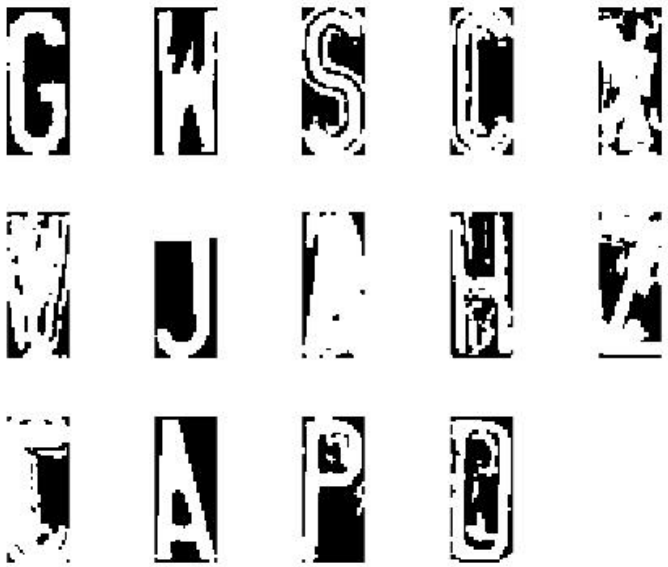

Figure 2.6: Examples of misclassified characters

accuracy of the network for different testing data. It can be seen that this value gets better for the group 1 and group 4 more than the others. This is because that most of the failed recognitions are caused by distorted character images, and the images in adverse conditions are not uniformly distributed across the whole dataset.

Figure 2.6 shows the misclassified characters from a testing of data in group 3.

A value of 97.56% is the expected recognition accuracy achieved by the network from the Matlab simulations.

2.6 Summary

In this chapter an introduction of feedforward neural network is given. A simulation-based design methodology is provided for designing a feedforward neural network for this application. The simulation results shows the potential performance of the hardware version of this network design. The parameters of the network acquired from simulations using data from group 1 as testing data will be used in the designing of the network in hardware. In the rest of this thesis, the requirement of recognition accuracy for the designed network is to reach the same level as the results of Matlab simulations.

Chapter 3

Network Implementation

As the structure and parameters of the network has been decided, the next task is to implement the network on an FPGA platform. For a hardware implementation of the network, more effort is required to maintain the same level of accuracy in comparison with the simulations due to the limitations of resources in the hardware devices. One major issue related to the accuracy is the limited number of bits that can be used to represent the Sigmoid activation function. Rough estimates of the activation function will prevent the network from getting trained properly.

Therefore, the goal in this hardware implementation was to control and keep the error of the activation function within a reasonable range. The process for this constraint is discussed in this section.

3.1 Arithmetic Format

From a computational perspective, a neuron is a nonlinear function of its inputs. Multiplication, addition, and a non-linear activation function are the essential computational operations of a neuron.

The format of the number representation has a significant influence on the computational precision and the final area that is required for implementing the network.

This is a common trade-off between the cost and the performance. Therefore, an appropriate numerical precision should be chosen that can satisfy the error requirements while it does not use too much hardware resources.

Here, the two most commonly used binary number representation formats in digital circuits are considered; fixed-point, and the floating-point representations. These two representation formats will be examined in terms of precision for numbers with same or similar ranges as well as their areas requirements for implementation of adders and multipliers.

3.1.1 Binary Fixed-Point Format

A fixed-point binary number can be represented by

$$SI_mI_{m-1}...I_2I_1I_0.F_{-1}F_{-2}...F_{-n+1}F_{-n} \tag{3.1}$$

where S is the sign bit, I is the integer part, F is the fractional part, and $i \in [-n, m]$. Each S, I, and F, represent a single bit binary number. Together, they form a fixed-point value.

The radix point sits between the integer and fractional parts. The sign-magnitude value for this representation is calculated as follows

$$V = (-1)^S \times (\Sigma_{i=0}^{m} I_i \times 2^i + \Sigma_{i=-n}^{-1} F_i \times 2^i) \tag{3.2}$$

For the sign-magnitude, fixed-point binary representations, the range of values that can be represented is between $2^{-n} - 2^m$ to $2^m - 2^{-n}$.

The range of this representation is mainly controlled by the number of the integer bits. If the fractional bits' contribution to the representation range is ignored, then it will be doubled every time the number of integer bits increases by one.

The precision of this representation is controlled by the number of the fractional bits. All of the values that can be represented by this format are evenly distributed across the range at 2^{-n} intervals. By increasing the number of fractional bits, the precision increases.

3.1.2 Binary Floating-Point Format

Floating-point numbers consist of one sign bit, mantissa bits, and exponential bits.

$$SM_mM_{m-1}...M_2M_1M_0.E_{-1}E_{-2}...E_{-n+1}E_{-n} \tag{3.3}$$

The value of a binary floating point number is calculated as follows:

$$V = (-1)^S \times 1.M \times 2^{E-bias} \tag{3.4}$$

where

$$E = \Sigma_{i=1}^{n} E_{-i} \times 2^{n-i} \tag{3.5}$$

and

$$bias = 2^{numE-1} - 1 \tag{3.6}$$

Where $numE$ is the number of bits for the exponential parts, $numM$ is defined as the number of the mantissa bits.

The representable range of values for floating point numbers depends on the number of exponential bits. As $numE$ increases, the range increases exponentially. Therefore, one advantage of binary floating-point format over fixed-point format is the larger representation range.

Figure 3.1 is an example of the positive part of a 1-2-1 floating-point number. This representation has 1 sign bit, 2 mantissa bit and 1 exponential bit. The $bias$ can

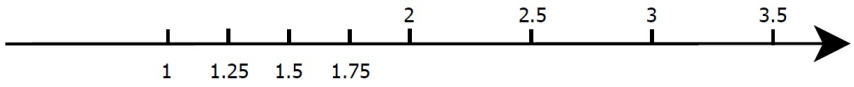

Figure 3.1: Positive range of an 1-2 floating-point number

E	M_1	M_0	value
0	0	0	1
0	0	1	1.25
0	1	0	1.5
0	1	1	1.75
1	0	0	2
1	0	1	2.5
1	1	0	3
1	1	1	3.5

Table 3.1: All positive values represented by 1-2-1 floating point numbers

be derived from the equation 3.6 that $2^{numE-1} - 1 = 0$ when $numE = 1$. In order to get a clearer view of this representation, all the possible positive values of it are listed in Table 3.1.

The precision of the representation depends on both the number of exponential bits and mantissa bits. To better illustrate the impact that $numE$ and $numM$ have on the precision, two terms are defined here. A *largeinterval* means a range of values that can be represented by the same exponential bits. A *basicinterval* means the interval between two adjacent values that can be represented by this number format.

The value of the $numE$ determines the number of large intervals (in this case the two large intervals are 1 to 1.75 and 2 to 3.5). Each large intervals is then divided

into basic intervals.

The value of the $numM$ determines the number of basic intervals that large interval includes and it is the same for all large intervals. However, the range of large intervals grows as number increases. This, in turn, causes the basic intervals to grows as well. Therefore the precision of binary floating-point representations is dynamic in its range. The closer to the origin the more precise the number will be.

3.1.3 Comparison Between the Two Representation Formats

In this section, the performance of the two representation formats is compared. This early stage comparison allows the designer to make a more informed decision.

The values of integer bits, fractional bits, mantissa bits and exponential bits are represented as $numI$,$numF$,$numM$ and $numE$ respectively. The range that a Fixed-Point (FXP) or an Floating-Floating (FLP) number can express is determined by $numI$ for the FXP and $numE$ for the FLP. Both $numF$ and $numM$ ascertain the accuracy of the FXP numbers and FLP numbers respectively.

The previous analysis showed that the advantages of floating-point representation over fixed-point are that it can cover a larger range if $numI$ and $numE$ are the same. However, this larger range is acquired at the cost of low accuracy for larger numbers.

This is pointed out by Figure 3.1, where the number of basic intervals is the same for all regions divided by the different value of E. As the range of region increases, the accuracy of this number representation drops.

Table 3.2 illustrates the number of bits to acquire certain range and number representation accuracy for both FXP numbers and FLP.

From the table, it can be seen that for the task of representing numbers in a certain range while keeping the same representation accuracy over the whole range, FLP requires more bits than the FXP.

It has been shown that for adders and multipliers with similar precision and range,

range	accuracy	FXP	FLP
$2^4 - 1$	2^{-4}	4-4	3-7
	2^{-5}	4-5	3-8
	2^{-6}	4-6	3-9
$2^5 - 1$	2^{-4}	5-4	4-8
	2^{-5}	5-5	4-9
	2^{-6}	5-6	4-10
$2^6 - 1$	2^{-4}	6-4	4-9
	2^{-5}	6-5	4-10
	2^{-6}	6-6	4-11

Table 3.2: Number of bits of FXP and FLP for different requirement

the implementation of the adder in floating-point representation requires as much as ten times of hardware as it in fixed-point representation implementation [32] shows. However, the multiplier implementation in floating-point requires about half of hardware resources of it in fixed-point implementation [32] shows. Considering the difference of hardware requirement for implementation as well as the number of adders will be much more than the number of multipliers, it is better to use fixed-point representation than using floating-point representation for this implementation of the feedforward neural network.

3.2 Activation Function

The fixed-point number format is chosen as the arithmetic format used in this work. Therefore, the next step is to implement the Sigmoid activation function in the fixed-point format.

The general definition of the Hyperbolic tangent function can be expressed as

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.7}$$

This function can not be directly implemented in hardware efficiently because of the division and exponential computations. Therefore, different approximation methods have been proposed for implementing this function in hardware.

3.2.1 Approximation Method

Look-up table method [12] samples hyperbolic function at constant intervals and stores the values into memories. The output of the look-up table is the stored function value. In order to keep the approximation error in a small range, a certain number of samples are needed. The more samples stored in the memories, the more precise the functions will be. Therefore, the size of the look-up table (LUT) function approximation method will be a concern.

Piece-wise linear (PWL) [1] approximation has a different strategy. The nonlinear function is replaced with many linear functions in various parts in its domain. The curve of the hyperbolic tangent function is divided into many areas. Straight lines are used to approximate the curve in each of these small regions. The approximated value of the input to this function is calculated based on the function of the corresponding line. Instead of storing all the values at different points, the PWL method [1] requires to implement several linear functions, and the approximation error is closely related to the number of piece-wised linear functions.

The procedure in [34] has been used in this work to implement the activation function. This method is similar to an LUT method. However, the memory requirement is small for same approximation criterion. This method implements the activation function using fewer circuits while the error is guaranteed to be less than a certain value ϵ. This improved LUT method is based on the observation of the sigmoid shape

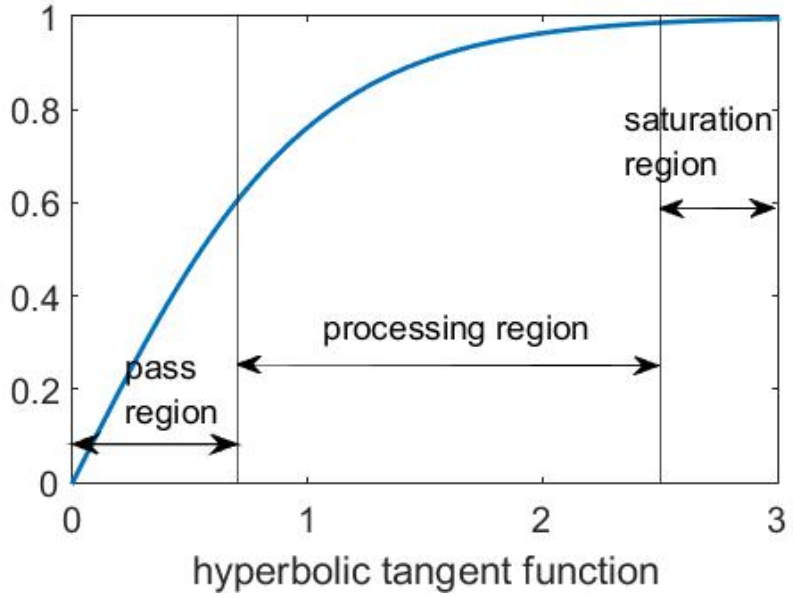

Figure 3.2: Different regions defined in hyperbolic tangent function

of the hyperbolic tangent function. The main idea is to divide the whole domain into three regions. Here the analysis will only consider the positive numbers since the hyperbolic tangent function is symmetric with respect to the origin.

Pass region covers the range that close to the origin. Since Equation 3.7 is differentiable in its domain and $f'(x) = 1 - f^2(x)$, the approximation value around $x = 0$ by using first order Taylor series can be expressed as

$$f(x) = f(0) + f'(0)(x - 0) \tag{3.8}$$

From Equation 3.7, $f(0) = 0$, and $f'(0) = 1$, the function can be approximated as

$$f(x) = x \tag{3.9}$$

when x is close to 0.

When the input exceeds a certain value, it enters saturation region. In this region, the sigmoid function has a value close to 1. Therefore, if the boundary of saturation

region is properly set, the output can be expressed as a constant

$$f(x) = C \tag{3.10}$$

while keeping the error within an accepted range.

The processing region is the region between the pass region and saturation regions. In this region, the derivative of the function with respect to the input decreases when the input increases. This means that the function tends to change at a slower speed with increasing input. Therefore, if a binary number is used to represent a range of the function, then the range is not the same over all of the processing region. This method is called range-addressable look-up table.

3.2.2 Errors of Approximation Method

The total error in this approximation consists of quantization error, e_q, and approximation error e_a. The quantization error is caused by the binary format of the input number, and e_a is caused by approximation method as well as the binary representation of output number. This can be seen clearly in Figure 3.3 : The curve stands for the target function which needs to be approximated, and the step function is an approximation of it. The three consecutive inputs are displayed as x_1, x_2 and x_3 respectively. When the value of an input such as x falls into the region of x_2, the output for this input will be $f_a(x_2)$. The quantization error e_q in this case is $|f(x_2) - f(x)|$, and the approximation error is $|f_a(x_2) - f(x_2)|$. Therefore the total error is as follows:

$$\begin{aligned} e &= e_q + e_a \\ &= |(f(x_2) - f(x)) + (f_a(x_2) - f(x_2))| \end{aligned} \tag{3.11}$$

where $f(x_2) - f(x)$ and $f_a(x_2) - f(x_2)$ may have different signs and the two errors sometimes may partly canceled by each other.

So this situation illustrated in Figure 3.3 is a worse case where they have same signs.

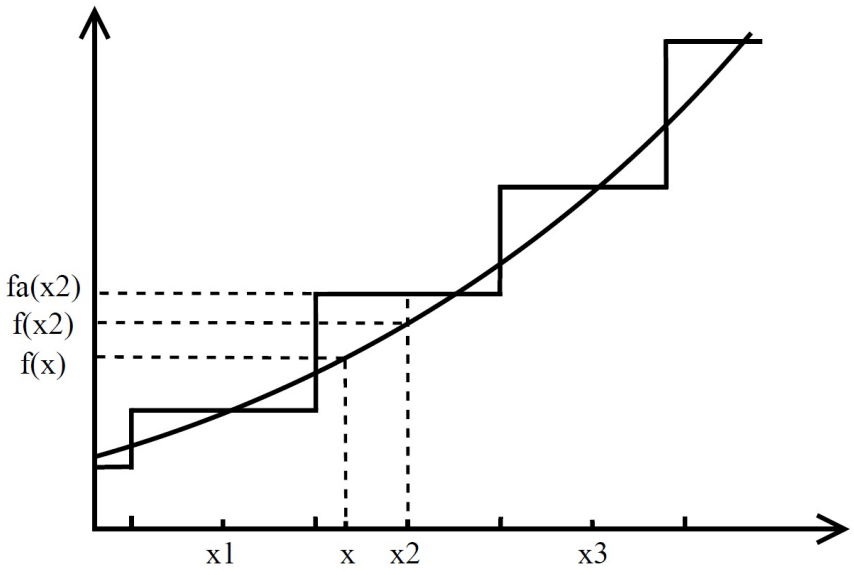

Figure 3.3: Quantization error and estimation error in this activation function approximation method

For inputs in pass region which are close to the origin, the difference between the hyperbolic function and $f(x) = x$ approximation can be omitted. This means that e_a is almost zero and error comes from e_q.

As the input moves towards positive infinity, e_a increases. In saturation region, since the output is a constant, there is no e_q and only e_a is left. In the processing region, both e_q and e_a need to be considered.

The boundary between the pass and processing regions and between the processing and saturation regions, as well as the number of fractional bits for both input and output, have to be decided carefully to make certain activation function's approximation error is less than a maximum error of ϵ.

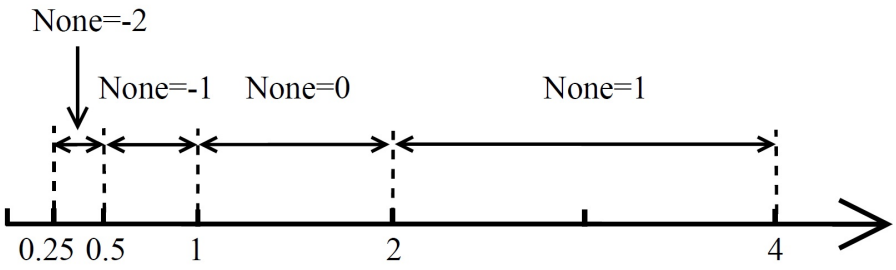

Figure 3.4: Regions determined by different N_{one}

3.2.3 Determining the Boundaries Between the Regions

In this section, the two boundaries, x_{paq} and x_{sq}, which divided the whole domain into three regions will be determined.

The input value determines which region and which subranges in the processing region the output is placed.

One way to make comparisons between two binary numbers is to compare them bit by bit from most significant bit(MSB). Therefore, a variable N_{one} is defined to indicates the place of the first non-zero bit (except the sign bit) in a binary number in the form of sign-magnitude. The N_{one} has the same value as the weight of this bit when the value of a binary number is calculated using equation 3.2.

For example, a binary number 0110.0111 has a $N_{one} = 2$ and 0000.1111 has a $N_{one} = -1$. The place of the first non-zero bit, or N_{one}, determines the possible value of this number. This can be seen more clearly in the Figure 3.4. Every time N_{one} increases by 1, the range of the number it can represent is doubled.

First, the boundary between the pass and processing regions x_{paq} is going to be determined.

Figure 3.5 shows the error of the activation function represented by $f(x) = x$ in the pass region. The top figure shows three functions. The curved line is the hyperbolic tangent function. The straight line is the function $f(x) = x$, and the step

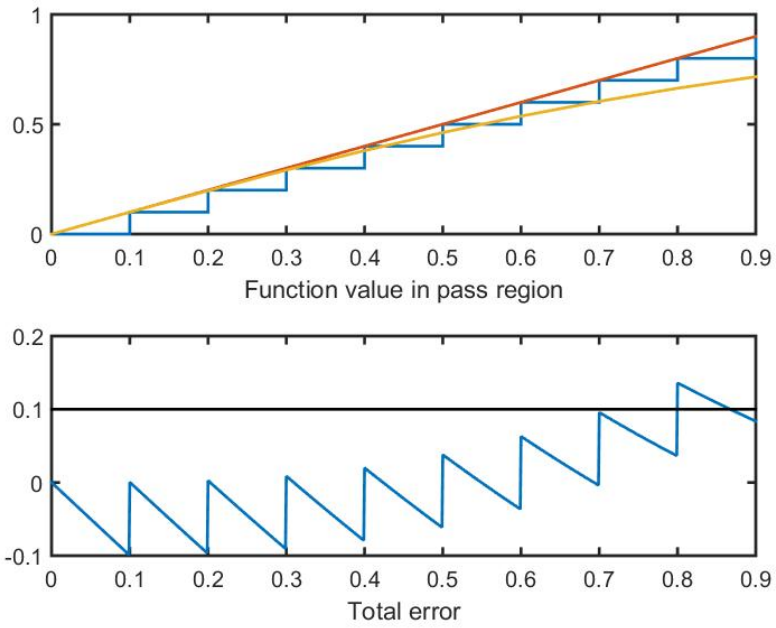

Figure 3.5: Error in pass region

function is $f(x) = x$ implemented in binary. This example uses a step of 0.1 for the reason of demonstration. In the implementation the step size is equal to 2^{-n} where n is the number of fractional bits.

The lower figure shows the difference between the step function and hyperbolic tangent function. When the input is close to the origin, the straight line and hyperbolic tangent function are almost the same. However, the error represented as the comb-like shape in the bottom figure is still unacceptable.

This is the quantization error which is caused by the binary representation and it has a maximum value of 2^{-N_f}. The value of the N_f has to be large enough to make sure $2^{-N_f} \leq \epsilon$.

As the input increases, the error has a trend of increasing. The quantization error is still the same. If the ϵ is set to a certain value such as 0.1 in this example (the horizontal line in Figure 3.5), then the boundary x_{paq} can be found.

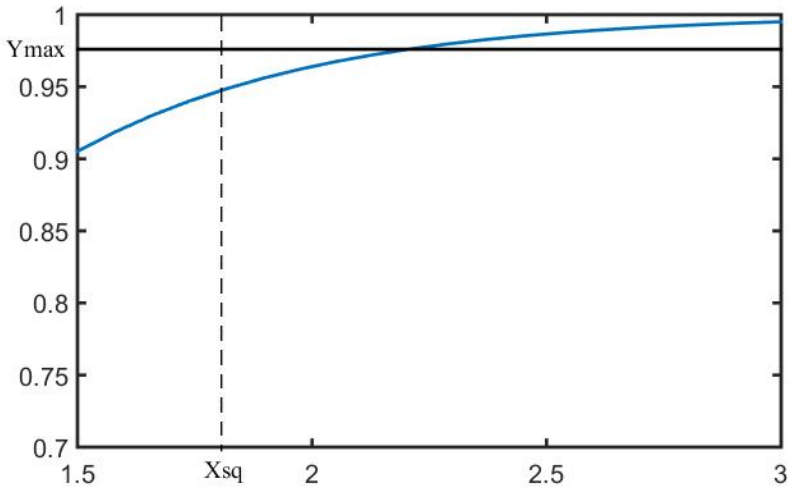

Figure 3.6: Activation function in saturation region

The first boundary x_{paq} is determined by $f_{step} - tanh(x) = \epsilon$, that is when the total error is greater than the maximum allowable error. However, when this function is computed, the step function, f_{step}, is replaced with $f(x) = x$. This is because if *epsilon* is set as 2^{-N_f}, then the error will not be greater than ϵ before the step function and hyperbolic tangent function separate.

After these two functions separate, the error only increases every time when $f(x) = x$ and the step function share the same value. Therefore, if ϵ is set to be equal to 0.04, by solving this function $x_{paq} = 0.5098$.

The next step is to calculate the boundary between the processing and saturation regions, x_{sq}. Figure 3.6 shows the hyperbolic function. The maximum value the function's binary representation can have is $Y_{max} = 1 - 2^{-N_f}$.

Moreover, as the input increases, the hyperbolic function approaches 1. Therefore, the value of the error in the saturation region is equal to $1-(1-2^{-N_f}) = 2^{-N_f}$. In order to keep the error less than ϵ, the requirement for N_f is the same as the requirement from pass region. The boundary x_{sq} can be determined from $1-2^{-N_f}-tanh(x_{sq}) = \epsilon$, again assume $\epsilon = 0.04$, by solving this function we can get $x_{sq} = 1.6492$.

3.2.4 Processing Region

So far, the two boundaries have been found by assuming $\epsilon = 0.04$. If x_{paq} is considered equal to 0.5098 and $x_{sq} = 1.6492$, it can be seen that the maximum value the function's binary representation can have falls within the range of $N_{one} = -1$ and $N_{one} = 0$. Therefore, inputs that are within these two ranges will be processed using processing region's method. All regions with different N_{one} inside processing region will be divided into N subregions evenly.

$$N = 2^{N_f + N_{one} - i} \tag{3.12}$$

This approximation method uses one value to represent all of the function values of the inputs that fall in one subregion. The number N defined by Equation 3.12 is different for different N_{one} to keep the sizes of all subregions in different regions the same. i is a non-negative integer, it is used to select the size of subregions. The region of N_{one} has a range of $2^{N_{one}}$, therefore, using Equation 3.12 the range of a subregion can be calculated by

$$\begin{aligned} \frac{2^{N_{one}}}{N} &= 2^{N_{one}} 2^{i - N_{one} - N_f} \\ &= 2^{i - N_f} \end{aligned} \tag{3.13}$$

Another variable $N_{sub} = 2^i$ is used to represent the number of smallest intervals 2^{-N_f} in a subregion.

$$\begin{aligned} N_{sub} &= \frac{2^{i - N_f}}{2^{-N_f}} \\ N_{sub} &= 2^i \end{aligned} \tag{3.14}$$

When i increases, the subregions become larger while the number of subregions, N, in one region becomes smaller. The smallest subregions are when $i = 0$. It should be noted that i can not be negative since the smallest subregions are the smallest intervals of input binary numbers can be represented.

The value of i should be as large as possible in order to decrease the complexity of the implementation of the hyperbolic function while keeping the error less than $\epsilon = 0.04$.

This approximation method assigns a number to each subregion. If the approximation error e_a in this subregion exceeds 0.04 due to the fact that the subregion is too large for using one value to approximate it, then a smaller subregion will be found by calculating the approximation error of a larger N, or a smaller N_{sub}. This process will be repeated until the approximation error meets the criteria. Because Equation 3.12 and Equation 3.14 state that N or N_{sub} depend on i, this process will start with $i = N_f + N_{one}$ and i will increase by one at a time if the approximation error still exceeds the limit. The values to represent each subregion is calculated using

$$\sum_{k=0}^{2^i} \frac{tanh(2^{N_{one}}(1 + \frac{j}{N}) + k \times 2^{-N_f})}{2^i + 1} \tag{3.15}$$

This process is done using Matlab simulations. After all of the approximated values for every subregion are found, they will be converted to binary numbers. Since only the estimate of the errors is considered during the calculation of values in all of the intervals, the output of the approximation method should be examined with both approximation errors and quantization errors.

For every possible binary input, the ideal hyperbolic function of this input and half step before and after this input are compared with the approximated value to see if their absolute difference exceeds ϵ. Simulations showed that this error is within ϵ for all inputs in the processing region.

To sum up, this function approximation method passes the input to the output without change, in the pass region. For every two consecutive input numbers in the first part of the processing region, one value is used for the output. The same value is used as the output for every four consecutive input numbers in the second part of the processing region. The output is a constant for inputs in the saturation region.

3.3 Determining the Size of the Fixed-Point Numbers

The fixed-point number representation format has been chosen as the number format that will be used in the FPGA hardware implementation. Therefore, a proper bit-length is required. This includes the number of integer bits N_i, and the number of fractional bits, N_f. If x is the value of a fixed-point binary number which has N_i integer bits and N_f fractional bits, then the range of x is

$$2^{-N_f} - 2^{N_i} \leq x \leq 2^{N_i} - 2^{-N_f} \tag{3.16}$$

Since $|2^{N_i}| \gg |2^{-N_f}|$, the range is mainly determined by the number of integer bits, N_i.

From the simulations $N_i = 6$ is large enough for the number of integer bits. The number of fractional bits, N_f, is 5 as this is the result derived from section 3.2. However, in some part of this system, a higher resolution of number representation method may be required in order to keep the error small. In order to find this requirement, computations of the system's recognition process in binary numbers are performed. The feedforward neural network can be expressed in a matrix form

$$output_i = f(W_2 \times f(W_1 \times I_i + b_1) + b_2) \tag{3.17}$$

where I_i is the input, W_1 is the weight matrix of the second layer's neurons, b_1 is the bias of the second layer's neurons, W_2 and b_2 are the parameters of the output layer's neurons , and f represents the hyperbolic tangent activation function.

The sizes and dimensions of these vectors are based on the set up of the binary number representation. The input I_i is an 189 by 1 vector, W_1 is a 160 by 189 matrix, b_1, is an 80 by 1 vector. The second layer values, W_2 and b_2, are 36 by 160 and 36 by 1 respectively. In this system, all of these numbers are in the binary format. A 2's complement binary representation is used for addition and subtraction operations.

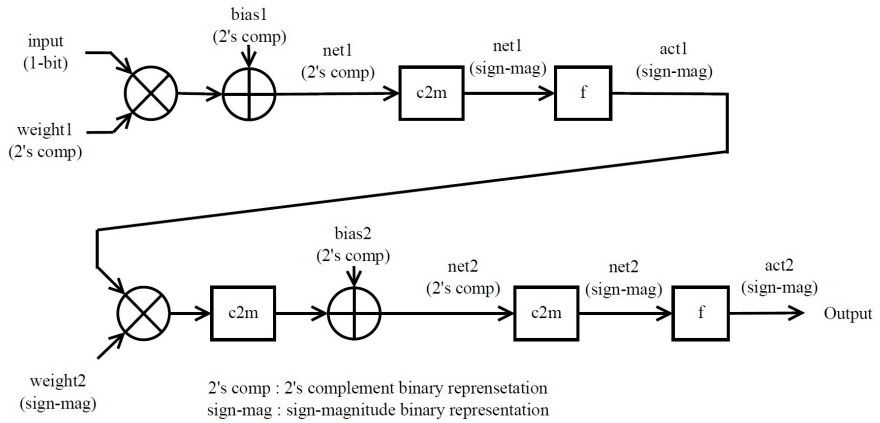

Figure 3.7: Computations in ANN in binary

The sign-magnitude binary representation is used for the multiplication operations and activation function. Figure 3.7 is the floating chart of the computational process of this system in binary form. The top half is the computations of the second layer, and the bottom half is the computations in the output layer. Some duplicate elements, for example, the other 188 input and weight multiplications are omitted for the sake of demonstration simplicity.

The $c2m$ is a 2's complement to sign-magnitude binary number conversion module, and f is the activation function. The first module is a multiplier. However, its one-bit operand simplifies this module to a multiplexer. Therefore, 2's complement numbers are used to represent $weights1$.

$$\begin{aligned} net_1 &= W_1 \times I_i + b_1 \\ act_1 &= f(net_1) \end{aligned} \tag{3.18}$$

$$\begin{aligned} net_2 &= W_2 \times act_1 + b_2 \\ act_2 &= f(net_2) \end{aligned} \tag{3.19}$$

Equation 3.18 and Equation 3.19 are the computational operations in the second layer, and output layer respectively. The term net is the weighted sum of all inputs

and bias, which is processed by the activation function. Output of the activation function is shown by *act*.

In order to execute these computational operations with binary numbers, and because there are no default programs that handles binary numbers which can be both positive and negative with fractional bits in Matlab, functions have been created to realize these operations. All these numbers are stored in Matlab as vectors with elements of 1 and 0, and each vector represents one binary number.

The first task is to convert all decimal numbers to binary numbers. Therefore, the first function is *decimal*2*binary*, which converts the decimal numbers to binary numbers in the form of 2's complement. The number of integer bits and fractional bits needs to be specified.

After that, some of the 2's complement binary numbers need to be converted to sign-magnitude numbers in order to perform multiplications. The function *com*2*mag* is used for this task. It is worth noticing that the binary numbers in the form of 2's complement and signed magnitude can be converted from one to the other by using the same method.

Once all of the numbers are ready, the next functions are the adder and the multiplier. The function *binaryadder* computes the result of the addition of two binary numbers in the form of 2's complement. The function *binarymultiplier* multiplies the two inputs in the form of sign-magnitude. The way the multiplication is conducted is to multiply one operand by every bit of the second operand. As the multiplication can be done by shifting, the result is a summation of shifted numbers, and the previous function *binaryadder* is used in this multiplication function.

Activation function is replaced with look-up tables acquired using the method in section 3.2. The last used operation is *binary*2*decimal* which converts sign-magnitude binary numbers to decimal numbers as it's much easier to inspect the results with a decimal number.

		Case 1	Case 2	Case 3	Case 4	Case 5
net1	Number of fractional bits	5	5	5	5	5
	Mean error	0.0779	0.0779	0.0779	0.0779	0.0779
act1	Number of fractional bits	5	5	5	5	5
	Mean error	0.0291	0.0291	0.0291	0.0291	0.0291
net2	Number of fractional bits	5	6	6	7	7
	Mean error	0.2246	0.1330	0.1330	0.0652	0.0652
act2	Number of fractional bits	5	5	6	6	5
	Mean error	0.2107	0.1195	0.1260	0.0590	0.0590

Table 3.3: Comparison of errors caused by fixed-point number representation

One program has been made using all these functions to realize the network that computes in binary numbers. One set of inputs is used to test the error caused by number representations. This experiment compares the results from this program computing in binary numbers and the results computed using high-precision floating-point numbers. Table 3.3 shows the results.

$net1$, $act1$, $net2$ and $act2$ are the variables in Equation 3.18 and Equation 3.19.

The mean error is the average difference between the results of Equation 3.18 and Equation 3.19 computed by fixed-point number system and floating point number system.

Case 1 shows that there is a huge increase in error when signals reach the output layer before they were passed through activation function. Therefore, in case 2 the number of fractional bits of the output layer's neurons' parameters is increased by 1. The result shows that this change decreases the errors caused by it dramatically.

In case 3 the number of fractional bits of the activation functions' outputs in the output layer is increased by one. However, it does not help with decreasing the error caused by the binary representation of the output layer's activation functions'

Network Settings		Test Results (Average value of 15 tests)	
Number of Fractional Bits in net1	5	Error of net1	0.07842
Number of Fractional Bits - act1	5	Error of act1	0.02901
Number of Fractional Bits in net2	7	Error of net2	0.06420
Number of Fractional Bits - act2	5	Error of act2	0.05263

Table 3.4: Errors caused by fixed-point format in hidden layer and output layer

outputs.

In case 4, the number of fractional bits of output layer's neurons parameters is further increased by 1. And the number of fractional bits in output layer's outputs is decreased by 1. In case 5, the result shows that the key is the number of fractional bits of the parameters in the output layer.

A reasonable conclusion from this test is that the error caused by the fixed-point representation can be kept less than 0.06 by letting the output layer's parameters have a number of fractional bits of 7, and other numbers have a fractional number of 5. The reason that output parameters need to be more precise than hidden layer's parameter is the multiplication operations in the output layer. In the second layer, the multiplications are actually done by multiplexers. Multiplication operations will cause more errors since the errors caused by fixed-point representation are also multiplied. Also, one of the two operands of multiplications in the second layer does not have an error since they are either 0 or 1, whereas both operands of multiplications in output layer have errors caused by fixed-point representation.

Table 3.4 is the network testing results with the same network setting. A fixed-

point binary number of 1-6-5 are used in the network except for the output layer's parameters which are 1-6-7, which is 2 bits longer in fractional parts than other numbers in the system.

The results shows that the error of this network when implemented in hardware using this method could be controlled to stay in an acceptable range.

For fixed-point numbers with 5 fractional bits, the intervals between every numbers it can represent is $2^{-N_f} = 2^{-5} = 0.03125$. If we consider the case that each binary number will cover the ranges of numbers both greater than its value and less than its value, then the maximum quantization error is $2^{-N_f-1} = 0.015625$. However, the errors showed in the table are not much greater than this considering each net is a summation of hundreds of numbers with quantization errors less than 0.015625. Another fact is that this errors caused by fixed-point representation tend to decrease after the data were passed through activation functions which means the hyperbolic tangent functions also squash the errors.

3.4 Summary

This chapter solves some theoretical problems of implementing the network in hardware. Number representation format and activation function approximation method are chosen carefully for this application. Desired bit-width for numbers at different computation steps are found from simulation results. The analysis of errors caused by activation functions and binary format are given. The data flow in binary format is formed to make the mapping from software programs to hardware more smooth. At last, a Matlab simulation in binary form was performed to verify the feasibility of this design.

Chapter 4

Network's Implementation in Hardware

In this chapter, the system designed in previous chapters will be implemented in hardware. The implemented system is supposed to have the same accuracy as the neural network simulated in Matlab. The network structure should be optimized to reach a right balance between the speed and the required area. Considering the network's scale, Altera's Cyclone IV serious FPGAs was used to meet the requirements. The implemented neural network, like the one simulated in Matlab, is initialized with parameters acquired from training phase. The network is designed using Verilog HDL codes, which are provided in Appendix A.

4.1 System Description

The Matlab simulations of this system in Chapter 3 verified the feasibility of this network to be implemented in hardware. However, the simulations performed all of the computational operations sequentially. From Figure 2.1 and Figure 2.2 it can be seen that the system has a layered structure. In each layer, the input data of this layer are available for all neurons in this layer at the same time in a parallel form.

For example, a neuron in the second layer is considered. Its output can be calculated using Equation 2.1 and Equation 2.2. If these multiplication operations can be executed simultaneously, the time to process one character can be greatly shortened. To exploit the parallel character of neural networks is the main reason to implement neural networks in custom or semi-custom hardware.

4.1.1 Parallelism

The faster processing speed of parallel computing requires extra hardware resources. Because in the parallel mode all of the computations in the same level are executed simultaneously, the related hardware components need to be duplicated multiple times.

For a feedforward neural network, the parallelism can be found at two levels. The first level is inside the neuron. For this type of parallelism, all multiplication operations in one neuron as described by Equation 2.2 will be executed simultaneously.

For the second level of parallelism, the computational operations of all the neurons in the same layer will be performed at the same time.

In terms of the computational processes of a single neuron, the first level of parallelism has the form of a sum of product (SOP) operations. The second level of parallelism has the form of multiply-accumulate (MAC) operations. These two types of parallelism are referred as SOP parallelism and MAC parallel. Both methods are going to be examined in terms of speed, area, and memory requirements.

For the SOP parallel network, the multiplication operations of one neuron are performed at the same time, which means the both operands of all multiplications in one neuron need to be ready at the same time. The number of required multipliers or multiplexers is equal to the number of units in the previous layer. The required memory for storing the intermediate results, which are ready for the summation computation, are equal to the number of multipliers or multiplexers.

For the MAC parallel network, all neurons in the same layer work at the same time have to compute the output. However, there is no requirement for SOP parallelism to happen under this circumstance. In other words, the multiplications of each neuron can be executed sequentially. Therefore, the number of required multipliers (or multiplexers for hidden layer) is equal to the number of units in this layer.

Because the operands become ready one by one, the summation result can be achieved by using accumulators. Each time an operand of the summation is ready, it will be added to the accumulator. When all the multiplications (or multiplexing) are done, the summation will be finished by accumulation. The number of accumulators is the same as the number of neurons in this layer.

In order to demonstrate the speed and area requirements of these two kinds of parallelisms, the number of multipliers and multiplexers, the number of adders, as well as processing time are listed in table 4.1.

For SOP parallelism, the number of multiplexers that are required for the hidden layer is 189 or one multiplexer for each input. The number of units in its previous layer, which is the input layer is 189, and the other operand of multiplexing come from the inputs of the system.

The processing time is counted as a number of clock cycles. It is the time for the summation and the time for all of the second layer's neurons' input multiplexing. However, the time for other operations such as loading the parameters from memories is not included.

	SOP Parallelism		MAC Parallelism	
	Hidden Layer	Output Layer	Hidden layer	Output layer
Number of MUXs or Multipliers	189	160	160	36
Number of Adders	189	160	161	36
Processing Time(Number of Clock Cycles)	167	44	190	161

Table 4.1: Comparison of SOP parallelism and MAC parallelism

For MAC parallelism, the number of required multiplexers in the second layer is 181 because only one multiplexer is implemented for each neuron in this layer. These are repeatedly used for all the inputs. The number of required multipliers in the output layer is 36. The number of clock cycles required by the second layer using inter-neuron parallel scheme is 190; this is the number of network's inputs plus 1 as the accumulation will start at the time when the second input are being processed. The summation is not counted in because the summation will be done as accumulations.

It is clear that the SOP parallelism needs more area and resources. However, it has a faster processing speed. In comparison, the MAC parallelism requires fewer resources, but it takes a longer time for processing each input pattern. As for the memory requirement,both these methods require the same amount of memory since the network in both configurations has the same number of parameters.

Another way to incorporate spatial parallelism is to use both SOP and MAC parallelism. A feedforward neural network can have both types of parallelism. However, this will require a significant amount of hardware resources, and therefore is not

applied here.

The multiplications in every neuron in the same layer are performed at the same time. Therefore the time necessary for processing one character will be reduced dramatically. However, the numbers of multipliers and multiplexers and the adders, as well as the memories required by this method, is too huge to implement a neural network in this size on a single chip. For example, the required number of multipliers, which are much larger compared with adders and multiplexers, will be $160 \times 36 = 5760$ if both types of parallelism are used, and this is only part of the hardware requirement in the output layer. Needless to say, it poses a huge challenge for the bus of the system to transfer such amount of data at the same time.

It should be pointed out that only one level of parallelism will make the network fast enough. The SOP parallelism's processing speed is higher for both layers while the hardware requirement is still affordable for a single FPGA chip. Therefore, it was applied in this network.

The analysis above omitted the biases in this network for the sake of simplicity as they only take a small part of hardware area compared with weights, and also they do not require multiplication operations.

4.1.2 FPGA

A Field Programmable Gate Array (FPGA) is a chip that contains a significant amount of simple units called Logic Elements (LE) or logic cells. The basic structure of an LE is a look-up table; carry logic and registers. The LUT is used for realizing logic or arithmetic functions, and the registers are used to hold the results. An LE can also be configured as memories. The data can be transferred between any of two LEs, but the transfer will take less time for two LEs in the same local area.

A large set of this basic structures can realize powerful functions, and it is this type of configuration gives FPGA the reconfigurability which is very welcome in de-

signing and fast prototyping. Apart from LEs, an FPGA usually has other functional blocks including a Phase-Locked Loop (PLL), multipliers, and memories. The PLLs are used to provide the system clock signals; embedded multipliers can save the resources for implementing other functions because the implementation of multipliers will consume a lot of LEs. A guideline to apply this network is to try to use the embedded multipliers and memories instead of using LEs to implement them.

After one clock cycle, the summation operation will start because all the operands it needs are ready. However, the summation will take several clock cycles; the time depends on the number of operands in this summation. For example, the summation in output layer where 160 numbers need to be added up is a 7-stage adder tree. The number of adders required for the adder tree is roughly equal to the number of units in the same layer. Because the inputs to this adder are calculated every clock cycle, the summation can be a pipelined operation. Extra registers are required for every stage in adder tree.

4.1.3 Network Parameters

The network parameters including weights and biases of neurons in the second layer and the third layer are acquired from network training in Chapter 3.

As stated in Chapter 3, parameters of neurons in the second layer are 12-bit binary numbers in the form of 2's complement, while parameters in the third layer are 14-bit binary numbers in the form of sign-magnitude.

Since the number of parameters that each neuron have is equal to the number of neurons in its previous layer plus 1, the number of parameters of this neural network are 190×160 12-bit numbers and 161×36 14-bit numbers.

For the implementation of the structure of one neuron, all parameters of one neuron have to be available at the same time during computation. Therefore, for the implementation of one neuron, the number of memories has to be the same as the

number of parameters. Moreover, since this single neuron is going to perform all of the computations in the same layer, the depth of each memory has to be equal to the number of neurons in this layer.

The embedded memory blocks in Cyclone IV serious chips are M9K, which can be configured as a Random Access Memory (RAM) of 1024×9 bits. Since the bit-length has to be 12 or 14, each M9K memory block can have a maximum address range of 512. For the memories that are used to store the parameters of the second layer neurons, the depth is 160, and the number is 36 for the output layer neurons.

4.2 Data Path

The network realized the functions stated in Equation 3.18 by using SOP parallelism. In Equation 3.18 $W_1 \times I_i$ and $W_2 \times act_1$ are multiplications between the matrices and the vectors, each multiplication of an vector in matrix W_1 or W_2 and I_i and act_1 is a SOP. The hardware realization of the network only needs to implement the structure of one neuron in each layer, and the summation function will be pipelined. In this section, all the modules will be discussed in terms of their functions.

4.2.1 First and Second Layer

The first layer is the input of this system; there are no computations involved in the first layer so it will be dealt within the designing of a test bench for this system.

The first task of the second layer of this neural network is to realize the function in Equation 3.18. The value of each neuron is loaded from memories. As it has been discussed in previous sections, the memories used to store parameters from neurons in the second layer are M9K blocks, which can be configured using the Altera Megafunction Wizard [2].

The block diagram of M9K memories can be seen in Figure 4.4. The bit length

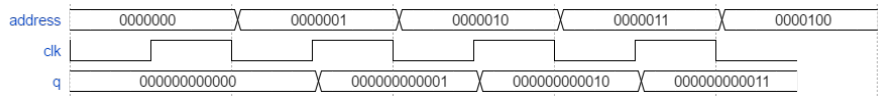

Figure 4.1: Waveform of inputs and output of a memory block

Figure 4.2: Waveform of inputs and output of a multiplexer

of these parameters is 12. Therefore these memory blocks are named as 'm9k12' to differ from the memory blocks in the third layer. The same structure is repeatedly used to realize the SOP operations for every neuron in the second layer. Therefore, the 190 'm9k12' blocks will have a depth of 160.

For one memory block, as the address signal increases, it will output the same parameter w_{ij} of the next neuron. The outputs of these memory blocks are controlled by rising edges of clock signals.

The files used for memory initialization are .mif files. They contain binary numbers of parameters converted from decimal numbers of network parameters acquired from simulations in Chapter 2. In order to test the memory block itself, a test bench file has been made. The address sent to the memory block will start from 0 and increase by 1 every clock cycle.

The content in this memory block are numbers that increases by 1 start from 1. From a simulation result showed in Figure 4.1 it can be seen that the content takes one clock cycle to propagate to the output. As parameters are read from memory blocks, the next step is to compute the multiplications of these parameters and their corresponding one-bit inputs, which can be realized as multiplexers. The one-bit inputs will be used as the select signal, and a diagram of this block can be seen in Figure 4.4. These blocks are named as 'mux12' as the bit length of the parameters are

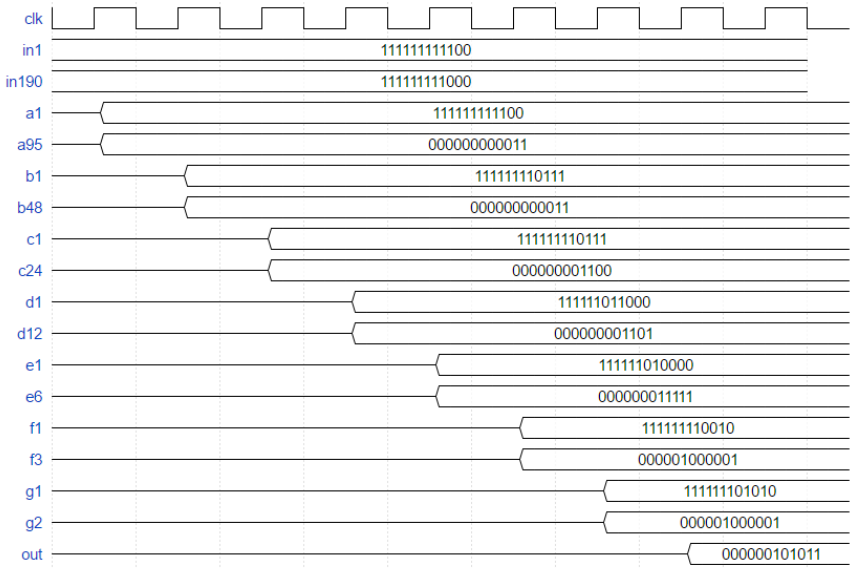

Figure 4.3: Waveform of inputs and output of an adder tree

12. It is also to differentiate these from the MUX which is an Altera mega function.

The input 'in' receives data from 'm9k12' memory blocks, and the output 'out' will have the same value as 'in' if the 'sel' receive a high value, and it will be 0 if the 'sel' is 0. And these multiplexing is controlled by the clock signal and read enable signal from input 'rd', the output will be updated only if there is a rising edge of the clock signal as well as the enable signal 'rd' is high. This module is tested and from the waveform in Figure 4.2 it can be seen that the output will be updated in the same clock cycle.

The next step is the summation of all the product calculated by 'mux12' modules. This module is named as 'adder190' because it's an adder tree to realize the summation of 190 numbers. A block diagram of this module is in Figure 4.4. It has 190 inputs named from 'in1' to 'in190', one clock input 'clk' and one output 'out'. From simulation result shown in Figure 4.3, it can be seen that it takes 7 clock cycles to finish the summation operation.

The intermediate results acquired after 'adder190' is a binary number and in the

form of 2's complement. Before it can be processed by activation function module, it needs to be converted into sign-magnitude numbers. This can be seen from Figure 3.7. Module 'com2mag165' is created for this number format conversion task. It differs from 'com2mag167' because it's input and output numbers are fixed-point binary numbers with 5 fractional bits. This module is implemented as pure combinational circuits based on the method that how 2's complement form is calculated.

From [20] we know that the conversion can be calculated as it's bitwise inverse plus 1 if it is a negative value. If it is positive, the result is the same as the input.

Therefore, the multiplication is implemented in the following way: The MSB will keep the same. If the input is a negative number, the LSB of the result will be the same of its corresponding input bit. For the rest bits, their results depend on the bits to their right. Only in the case that the bits on their right side are all 0, there is a carry when the bit-wise inverted input plus 1. The simulation of the binary format conversion module shows that the converting operation happens in the same clock cycle due to the combinational circuit implementation way.

The activation function is realized in the way of a look-up table. A block diagram of this module can be found in 4.4. Bit 10 to bit 6 are used to determine if the input is in the saturation region. Bit 5 and bit 4 are used to establish if it is in pass region.

After the data passed the activation module, the output of one neuron in the hidden layer is acquired. For one input pattern to this system, there are 160 intermediate results in the second layer. Because SOP parallelism requires the inputs to be available at the same time, the memory block 'ram1' is designed to hold the output of neurons in the second layer which will be used in computations in next layer. A block diagram of this module can be found in Figure 4.4.

Its input terminals include data, address, clock and write-enable signal. It also has 160 output terminals for the realization of spatial parallelism in the output layer. As it has been discussed before, the neurons in the same layers are processed sequentially.

The computations in the output layer will not start until all the neurons' outputs in the second layer are calculated.

4.2.2 Third Layer

The third layer's computations have a similar structure to those in the second layer. However, there are still several major differences other than the different size of the 3rd layer. These will be discussed in the following module descriptions for the third layer.

First, the multiplexers in the second layer are replaced with multipliers here in the third layer. Since the multiplications are the first computation operations in the third layer, it's operands needs to be in the format of sign-magnitude. The outputs from the second layer are in this format. Therefore there is no need for a conversion before the multiplication. The parameters of the third layer are also in the format of sign-magnitude and have a bit length of 1-6-7. They are stored in the memory blocks named 'm9k14'. They have the same structure except that the parameters stored in this module are 14-bit long.

The multipliers in this network which named as 'mult14' have two data inputs, one clock input, and one data output. The operations are triggered by the positive edge of the clock signal and Figure 4.4 has a block diagram of the multiplier. The output is truncated to the same length as inputs. There is a total of 160 multipliers to realize the parallelism in the third layer.

Before the summation can be performed, the output from multipliers needs to be converted to 2's complement numbers. And this is done by the modules of 'com2mag167'. They are similar to 'com2mag165' in the second layers. The only difference is that the input and output of a 'com2mag167' has a bit length of 1-6-7.

The 'adder161' will perform the summation in the same way as 'adder190' in the 2nd layer. Like 'adder190', the summation in 'adder161' also takes 7 clock cycles.

The result from 'adder161' will go through another 'com2mag167' block to convert to sign-magnitude format in order to be processed by activation function module.

Instead of storing all the outputs from third layer neurons as in the second layer, only the ordinal number of the largest output among all outputs of third layer neuron will be stored and output as the output of this system. In order to achieve this, the biggest number of all outputs needs to be found, and this is done by the module 'comparator'.

Figure 4.4 contains a block diagram of this module. It starts to work when the input 'wr _en' receives a high value. The input 'in' receives the results from activation function in the 3rd layer, and the input 'addr' is the ordinal number of each input. When the input 'counter' reaches a certain number, the ordinal number of the max output will be sent to the output.

The bias involved operations are implemented in different ways. The bias in the second layer are multiplexers with the select signal of the constant of 1. Therefore the output of these multiplexers will always output the biases value instead of 0. The biases in the third layer are implemented with buffers instead of multipliers. Therefore this system requires a total number of 160 14 by 14 multipliers.

4.3 Control Path

The system in this work is a synchronized system. Every blocks that contains sequential circuits are governed by clock signals. In this section, all the modules will be connected to form a system, and the timing of every block, as well as the pipelining of some of the blocks, will be considered by carefully assign clock signals and control signals to every module, also the modules that generate all clock signals and enable signals will be discussed, too.

4.3.1 Clock Signals

This system uses 3 clock signals - 'CLK', 'clk_inv' and 'clk_phs'. 'clk_inv' is the inverse of 'CLK' and they have a phase shift of 180 degrees. 'clk_phs' has a negative 90-degree phase shift with 'CLK' and a positive 90-degree phase shift with 'clk_inv'. They are generated by the module 'clk_gen'. The reason that more than one clock signals are used is to leave enough time for a signal to settle down before reading it. For the modules that use positive edge of clock signals as enable signal, its input data and clock signal can not change at the same time, the time period between the input data signal starts to change, and the rising edge of clock signal has to be big enough to let the changing of input data to settle down. It is implemented using a phase lock loop embedded in the FPGA. The output 'start' will become low once the clock signals are locked.

Figure 4.4 is a block diagram of the feedforward network. It has 3 inputs and 1 output. Input 'in' is a 189-bit binary number, each bit is connected to one 'mux12' module. Input 'clk_source' is the outside clock source to this network. Input 'rst' will reset the network when it is high, it is connected to the 'controller' module through an OR gate with the 'start' signal from 'clk_gen' module, means the system will not start to run if the 'rst' signal is high or the clock signals have not been locked by PLL. The 'out' is a 6-bit number that indicates the ordinal number of the max output among the outputs of neurons in the 3rd layer.

Whenever the 'rst' input signal to 'controller' module becomes low, the register inside the 'controller' module will start to count. It will start from 0 and increase by 1 every time there is a positive edge of 'CLK' signal. 'counter' is an important variable which is related to many enable signals and address signals. It will not be reset and keeps increasing by 1 every clock cycle during the whole process of processing one system input.

The memory blocks 'm9k12' which store network's first layer parameters will ac-

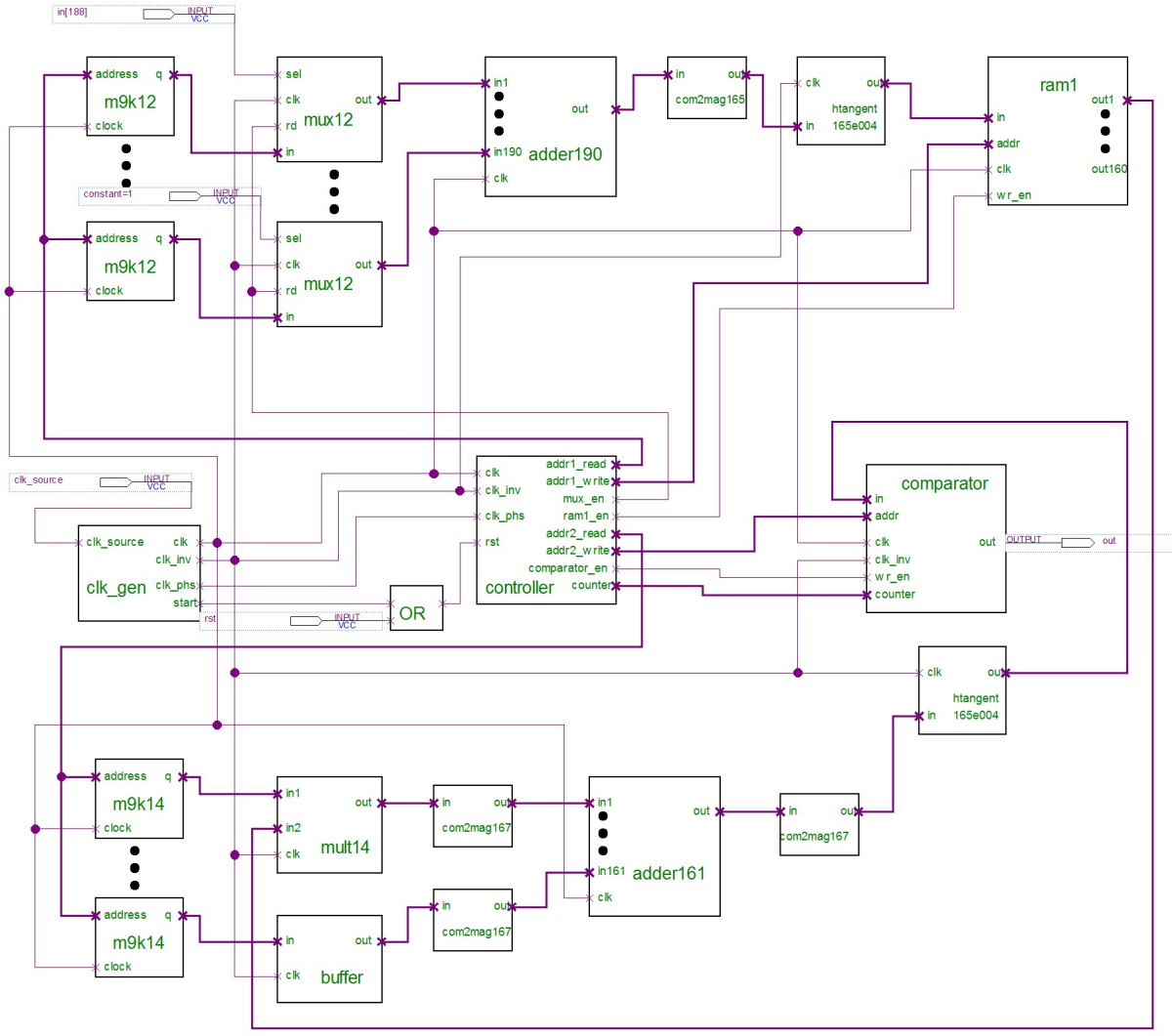

Figure 4.4: Block diagram of the feedforward network

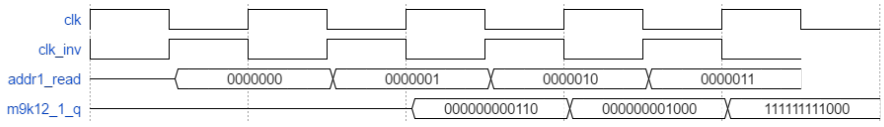

Figure 4.5: Loading second layer's parameters

cess the data at the input address when there is a positive edge of its clock signal. This clock signal used by 'm9k12' is 'CLK' from 'clock_gen' module. It is safe for the rising edge appears in the middle of every clock cycle of the address signal. This can be achieved by using 'addr1_read' ,which is generated by 'counter' and the 'clk_inv' signal, as the address signal for memory blocks in the 2nd layer. This is evident by the Figure 4.5. And as pointed out in the previous section, memory blocks take one clock cycle to output the signal, so the data of the first address in memory blocks will reach the multiplexers when the counter register is 2.

It is necessary for the data signal to settle down before there is a rising edge of the clock signal, so the clock signal sent to multiplexers will be 'clk_inv'. The first results from multiplexers will reach 'adder190' in the middle of the clock cycle when the counter is 2, so the CLK signal to synchronize 'adder190' is 'CLK'. The summation in the second layer will take 5 clock cycles, so the data will reach the activation function block when the counter is 8. Therefore, the 'ram1_en' that enables writing to 'ram1' will become high when the counter is greater than 7. And the address 'addr1_write' sent to 'ram1' will start to count when the counter is 8. When the counter reaches 88, the writing enable signal to 'ram1' will become low, and the output of second layers neurons are all stored in 'ram1'.

4.3.2 Other Control Signals

The controller will start to send address signal of numbers counting from 0 to memory blocks in the third layer when the counter variable reaches 89, that is the time when calculations in the second layer are finished. And like those in the second layer, address signals are generated from 'clk_inv' signal so that they will have half a clock cycle to settle down.

The data in the address will appear in the output of memory blocks after one clock cycle, and then will be processed by multipliers. Multipliers 'mult14' are synchronized

by 'clk_inv' clock signal. And the results of multiplications will be calculated in the same clock cycle of input signals. Then the results from multipliers will be converted to 2's complement numbers before the 'adder81'. The inputs of 'adder81' changes at every rising edge of 'clk_inv', therefore this block can be synchronized by 'CLK' signal. The summation of 81 numbers in the adder tree will take 6 clock cycles.

In the end, the summation results will be converted back to sign-magnitude numbers right before the activation function block. And the 'htangent165e004' is the same block as the activation function block in the second layer which is connected to the 'clk_inv' signal.

The data sent to 'in' port of 'comparator' block changes at rising edge of 'clk_inv' signal, so the 'comparator' block uses 'CLK' signal and a writing enable signal from 'controller' as the enable signals for writing input data to the registers inside comparator block. Because the summation takes 6 clock cycles, the first data reaches 'in' port of 'comparator' should be at the rising edge of 'clk_inv' inside the clock cycle when 'counter' reaches 98, and it changes at every rising edge of 'clk_inv' after that.

Two control signals, 'addr2_write' and 'comparator_en', are sent to 'comparator' from 'controller'. 'addr2_write' offers an ordinal number so the comparator can know that which number the new input is each time it compares it to the current largest number. 'comparator_en' is same as the clock signal that has a phase shift while 'comparator_and' is high. It can be seen from Figure 4.6 that after the first result from third layer neurons is ready, the comparator enable signal will be high at each rising edge of the clock signal and the 'addr2_write' will start to count from 0.

4.4 Summary

In this chapter, two types of parallelism in feedforward neural networks are discussed first. All blocks that are used to form the system including the memory blocks are presented. Data path is formed in the way discussed in the previous chapter, and

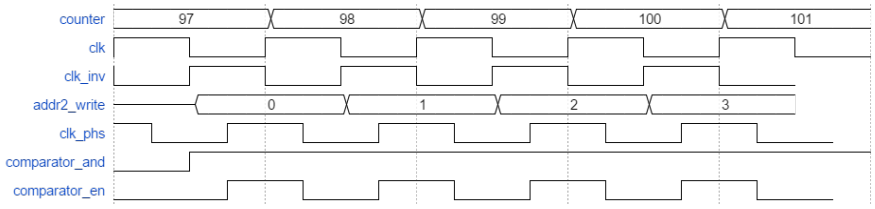

Figure 4.6: Clock and control signals of comparator

the difference between the second layer and third layer are given. The control path is formed in the way to make sure the data propagated in the feedforward neural network are consecutive. The clock signals and control signals used in this system are discussed. The RTL level simulations are provided in the next chapter.

Chapter 5

RTL Simulations

In this chapter, the feedforward network implemented on FPGA device with Verilog code will be tested with the dataset used in chapter 2. A comparison of the recognition accuracies of the designed network for hardware implementation and it's software version in chapter 2 will be made. The Verilog codes are compiled by Altera's Quartus II programmable logic device design software, and are provided in Appendix A. The synthesized network was tested with ModelSim simulation software.

5.1 Network Synthesis

The device used in the compilation of this system including synthesis, fitting, and timing analysis is EP4CE115F23C7. This device has a total number of $114,480$ logic units and has $1,161,216$ memory bits and 9 by 9 embedded multipliers of 232.

From Figure 5.1 it can be seen that this network only used $13,909$ logic units, which is 12.15% of its total logic units. It also uses 320 out of the 532 embedded

Flow Summary	
Flow Status	Successful - Tue Aug 02 23:53:22 2016
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	ann3
Top-level Entity Name	ann3
Family	Cyclone IV E
Device	EP4CE115F23C7
Timing Models	Final
Total logic elements	13,909
Total combinational functions	10,622
Dedicated logic registers	10,278
Total registers	10278
Total pins	197
Total virtual pins	0
Total memory bits	445,986
Embedded Multiplier 9-bit elements	320
Total PLLs	1

Figure 5.1: Synthesis results of feedforward network

multipliers to implement 160 14 by 14 multipliers. If the multipliers are implemented using LEs, it requires 42 LEs for implementation of one 9 by 9 multiplier. Therefore, even though no embedded multipliers are used in this design, it still only requires $2,7349$ LEs which is 23.89% of the total logic units in this device.

Figure 5.2 is a block diagram of this network. It uses 197 out of 281 user-defined I/Os in total. This includes 189-bit input pins, one clock source input, one reset input and 6-bit output pins.

5.2 RTL Simulations

After the network is synthesized successfully by using QuartusII software, an RTL simulation is done using ModelSim software. A test bench file is created in Verilog HDL. In this simulation, the network parameters from a successful Matlab simulation in Chapter 2 for the first group of testing data are used. A text file containing all of the inputs from 96 license plates is also included in the test bench file. The 96 license

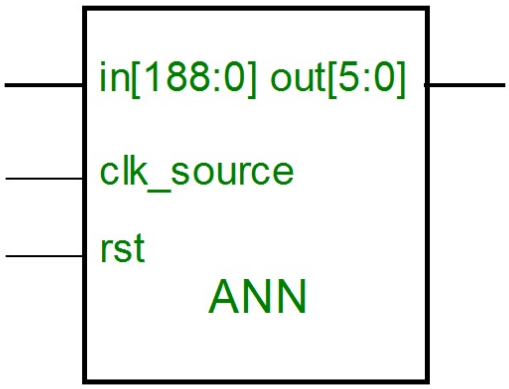

Figure 5.2: A block diagram of this feedforward network

plates contain 665 characters, and an 189 by 665 register matrix is used to store all the input data.

The time scale of this simulation is set to 1 ns, a clock signal which has a period of 20 ns and 50 % duty cycle is created as the source of the clock signal. Because the calculations for one character inputs of this network takes 218 clock cycles, the new input patterns arrive every 223 clock cycle to give the network enough time for processing the previous inputs and generating the results.

The results are 6-bit binary numbers, and the value of the binary numbers identifies the input character. The simulation results are recorded in the transcript with all the output values in it.

In order to get a better interpretation of these binary numbers and to compare the results with the ideal outputs, a transcript with all of the output values are processed by a Matlab program. It converts the outputs from the binary numbers to a string of characters. In the same manner, as the simulations in Chapter 2, characters 'I' and '1', 'O' and '0' are considered the same. In total 665 characters, 653 characters are recognized successfully. This is a recognition accuracy of 98.20 %. The recognition

	Proposed network in Matlab	Proposed network in Verilog
Recognition accuracy(calculated in number of characters)	98.35%	98.20%
Processing time (for one LP)	0.348s	30.52 μs

Table 5.1: Comparison of software simulation and hardware simulation

accuracy is 89.58% if it is calculated in number of license plates. A processing time of 4.36μs is acquired assume the system runs at a clock rate of 50MHz.

Table 5.1 compares the network implemented using Matlab code and Verilog code. The recognition accuracy of the network's Matlab version has a recognition accuracy of 98.35%. However, considering the different number representation formats, the 0.15% performance drop is acceptable, and this result marks the success of the hardware implementation.

5.3 Comparison with Other Works

In this section, the proposed system is compared with other OCR systems for LPR that are realized in hardware. The comparison is conducted in terms of performance, area, and speed.

Table 5.2 compares this network with another neural network-based OCR system which is also implemented on FPGA. The used dataset in this work contains many low-quality images, unlike other works.

In order to compare the area of these two FPGA-based OCR systems more explicitly, the required RAM and multipliers are all converted into number of LEs. As it has been mentioned in Section 5.1, one 9 by 9 multipliers can be implemented using

Reference		Proposed System	[33]
Recognition Accuracy		98.2%	97.3%
Area	LE	13909	6848
	RAM	351 9-Kbit RAM blocks	4 18-Kbit RAM blocks
	Multipliers	320 9×9 Multipliers	8 18×18 Multipliers
Speed	Processing Time	$4.36\mu s$	$671.93\mu s$
	Clock Cycles	218	44148

Table 5.2: Comparison of different hardware-implemented networks

42 LEs. Each LE contains one 4-input LUT, which can be used as a 16-bit memory. Therefore, the system requirement can be all converted into the number of required LEs. The proposed system requires 55,221 LEs. [33] only gives the number of memory blocks it required instead of number of bits. Assuming all the number of bits in these memory blocks will be used, the equivalent number of LEs required by the system in [33] is 12,340. Comparing the area and speed of these two systems, we can see that the proposed system's hardware resources requirement is roughly as 5 times as that of the system in [33], while the processing speed of the proposed system is much faster than that system. This is mainly due to the realization of parallelism which is an intrinsic character of feedforward neural network. To realize the parallelism is a main incentive to implement the feedforward neural network in hardware.

Table 5.3 is a comparison between the proposed OCR system with other software-based OCRs in LPR systems. This table shows that the processing time of the

Reference	Proposed system	[6]	[9]
OCR method	ANN	Probabilistic NN	Self-organizing character recognition
Recognition accuracy (calculated in number of LPs)	89.58%	89.1%	95.6%
Time to process one LP	$30.52\mu s$	128ms	2s

Table 5.3: Compare the proposed network with software-based OCRs

proposed system is much shorter than the time required by other OCR systems.

Figure 5.3 is some examples of successfully recognized characters by two different systems. The license plates on the left side are from the database used in this work, and those on the right side are from an image in [9]. It can been seen from this figure that the dataset used by this work contains some highly noisy characters as well as characters in different fonts. The quality of the license plates is a very important issue and have a big influence on the recognition accuracy rate.

The higher recognition accuracy is due to the proper setup and selection of network size. The parallel scheme incorporated in this network makes it much faster. The properly selected number format representation also reduced the size of the network. The recognition accuracy in both works is calculated in the number of characters.

5.4 Summary

In this chapter the network's synthesis results and RTL simulation results are provided. A comparison between this work's performance and other OCR systems im-

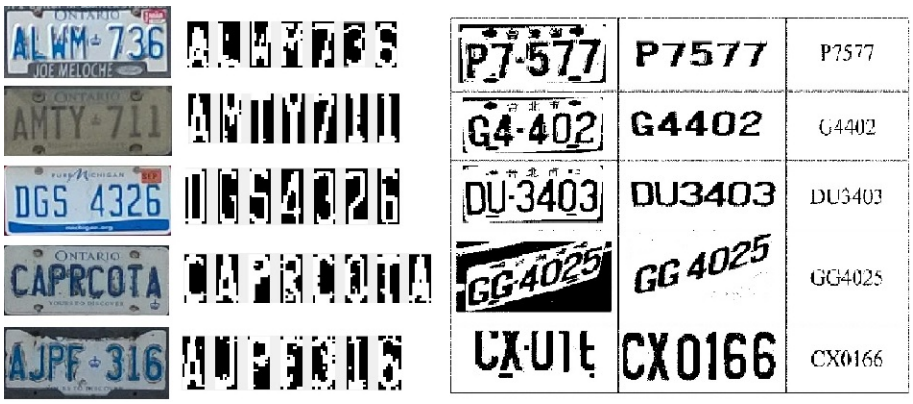

Figure 5.3: Comparison of examples of successfully recognized LPs

plemented in both hardware and software are made. In conclusion, the parallelism in this network makes its processing speed much faster than its software and hardware counterparts. The recognition accuracy achieved is competitive when compared with other methods considering the poor conditions of the images.

Chapter 6

Conclusion

6.1 Contributions

In this work, a feedfroward neural network-based OCR system has been designed. The massive parallel computations in the feedforward neural network and the possibility to use FXP number format makes FPGA the perfect platform for this task.

The main contributions of this work can be summarized as

- A standard procedure to determine the size of the network.

  This procedure includes extracting training data, testing data and target data from the row dataset and training and testing the network with different network settings to find the optimum size of the network.

- Matlab programs which can find the optimum sizes of FXP numbers in different places in the feedforward neural network.

  The network's computations are all executed in FXP numbers. However, the

length of the FXP numbers can be different for numbers in different stages of the data flow. These programs can perform computations of binary positive or negative numbers with fractional parts to simulate the computations in the feedforward neural network which uses FXP numbers. In this way the preferable accuracy of binary numbers in different size of the network can be determined. It can also verify the accuracy of the network computed in FXP numbers before the design process of the hardware version of this network is started to test the feasibility of the network design.

- A set of Verilog programs to map the software version of this network into hardware which targets at FPGA implementation.

  The computations of a feedforward network are realized in many modules and each module serves for a certain purpose. Therefore, the network can be easily reconfigured to a different size. The control circuits are carefully designed to make sure the data flows in the network consecutively. The parallel configuration of this network gives it a huge advantage in processing speed.

In conclusion, a feedforward neural network-based OCR system for a particular dataset of license plate characters is designed. In this design process, the size of the network as well as the number representation format are properly determined to reduce the size of the network and achieve a good performance at the same time. The application of FXP number representations only has a minor influence on the recognition accuracy whereas the size of the network is greatly reduced. The system is verified before using Matlab programs which runs the algorithm totally in FXP numbers. The designed system is a synchronized system where clock signals and enable signals controls the data flow inside the network. A hardware version of this network which is intended to be implemented in FPGA device is designed in verilog HDL. And the RTL simulation of the system shows the system meets the criteria.

6.2 Suggestions for Future Work

Feedforward neural networks can be found useful in many other situations. After minor modification, this work can be adjusted to deal with different datasets. For some more complicated applications, the network can be expended to a larger size. If the number of neurons are increased, the bandwidth will face a challenge if there is still parallelism in this network. If the number of layers are increased, different learning algorithm or different activation functions can be considered. The training of the network can also be realized on hardware to accelerate the training phase. For some applications where training happens more frequently, the time to train the network should be taking into consideration and the back-propagation calculations can be realized by the system, in that case the number representation should be carefully determined as errors will prevent the network from converging.

References

[1] E. P. Scilingo A. Armato, L. Fanucci and D. De Rossi. Low-error digital hardware implementation of artificial neuron activation functions and their derivative. *Microprocessors and Microsystems*, 35(6):557–567, 2011.

[2] Altera. Memory blocks in cyclone iv devices.

[3] Yu-Chi Ho Arthur Earl Bryson. Applied optimal control: optimization, estimation, and control. *Blaisdell Publishing Company*, page 481, 1969.

[4] C. Nelson Kennedy Babu and K. Nallaperumal. Robust license plate recognition based on dynamic projection warping. *International Journal on Imaging Science and Engineering*, 2(2):189–194, 2008.

[5] I. Psoroulas V. Loumos C. Anagnostopoulos, I. Anagnostopoulos and E. Kayafas. License plate recognition from still images and video sequences: A survey. *IEEE Transaction on Intelligent Transportation Systems*, 9(3):377 – 391, 2008.

[6] V. Loumos C. Anagnostopoulos, I. Anagnostopoulos and E. Kayafas. A license plate-recognition algorithm for intelligent transportation system applications. *IEEE Transaction on Intelligent Transportation Systems*, 7(3):377–392, 2006.

[7] Hakan Caner, H. Selcuk Gecim, and Ali Ziya Alkar. Efficient embedded neural-network-based license plate recognition system. *IEEE Transactions on Vehicular Technology*, 57(5):2675–2683, 2008.

[8] A. Capar and M. Gokmen. Concurrent segmentation and recognition with shape-driven fast marching methods. *Proceeding of International Conference on Pattern Recognition*, 1:155–158, 2006.

[9] Shyang-Lih Chang, Li-Shien Chen, Yun-Chung Chung, and Sei-Wan Chen. Automatic license plate recognition. *IEEE Transactions on Intelligent Transportation Systems*, 5(1):42–53, 2004.

[10] V. Palazon D. Llorens, A. Marzal and J. M. Vilar. Car license plates extraction and recognition based on connected components analysis and hmm decoding. *Chapter on Pattern Recognition and Image Analysis, Volume 3522 of the series Lecture Notes in Computer Science*, pages 571–578, 2005.

[11] C.C. Taylor D. Michie, D. J. Spiegelhalter. Machine learning, neural and statistical classification. *Overseas Press; 2009 edition (August 28, 2009)*, 2009.

[12] G. Juarez J. O. Perez F. Ortega-Zamorano, J. M. Jerez and L. Franco. High precision fpga implementation of neural network activation functions. *IEEE Transaction on VLSI Systems*, 331(1):55–60, 2014.

[13] Jing-Ming Guo. License plate localization and character segmentation with feedback self-learning and hybrid binarization techniques. *IEEE transactions on vehicular technology*, 57(3):1417 – 1424, 2008.

[14] S. Y. Chen H. J. Lee and S. Z. Wang. Extraction and recognition of license plates of motorcycles and vehicles on highways. *Proceeding of International conference on Pattern Recognition*, pages 356–359, 2004.

[15] Donald O. Hebb. The organization of behavior: A neuropsychological theory. *Psychology Press edition, ISBN 978-0805843002*, 1949.

[16] G. S. Hsu, J. C. Chen, and Y. Z. Chung. Application-oriented license plate recognition. *IEEE Transactions on Vehicular Technology*, 62(2):552–561, 2013.

[17] [Online] Available: https://www.research.ibm.com/haifa/research.shtml.

[18] M. Stinchcombe K. Hornik and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(1):359–366, 1989.

[19] S. H. Park K. I. Kim, K. Jung and H. J. Kim. Support vector machines for texture classification. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(11):1542–1550, 2002.

[20] Israel Koren. Computer arithmetic algorithms. *A. K. Peters, Natick, MA, 2002 (ISBN 9781568811604)*, 2002.

[21] M. R. G. Meireles, P. E. M. Almeida, and M. G. Simoes. A comprehensive review for industrial applicability of artificial neural networks. *IEEE Transactions on Industrial Electronics*, 50(3):585–601, 2003.

[22] Marvin Minsky and Seymour Papert. Perceptrons, an introduction to computational geometry. *The MIT Press; expanded edition edition (December 28, 1987)*, 1969.

[23] Raul Rojas. Neural networks: A systematic introduction. *Springer-Verlag, Berlin, New-York*, 1996.

[24] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, 1986.

[25] B. Shan. Vehicle license plate recognition based on text-line construction and multilevel rbf neural network. *Elsevier Journal of Computing*, 6(2):246–253, 2011.

[26] T. Sirithinaphong and K. Chamnongthai. The recognition of car license plate for automatic parking system. *Signal Processing and Its Applications, 1999. ISSPA '99. Proceedings of the Fifth International Symposium on*, 1:455–457, 1999.

[27] K. T. Li T. H. Wang, F. C. Ni and Y. P. Chen. Robust license plate recognition based on dynamic projection warping. *Proceeding of IEEE International Conference on Networks, Sensing, and Control*, pages 784–788, 2004.

[28] D. A. Duc T.D. Duan and T.L.H. Du. Combining hough transform and contour algorithm for detecting vehicles' license-plates. *Proceeding of International Sympsium on Intelligent Multimedia Video Speech Process*, pages 747–750, 2004.

[29] T. V. Phuoc T.D. Duan, T. L. H. Du and N. V. Hoang. Building an automatic vehicle license plate recognition system. *Proceeding of International Conference on Computing Science*, 23(2):59–63, 2005.

[30] H. Zhang W. Jia and X. He. Region-based license plate detection. *Elsevier Journal on Network Computing Application*, 30(4):1324–1333, 2007.

[31] Shen-Zheng Wang and Hsi-Jian Lee. A cascade framework for a real-time statistical plate recognition system. *IEEE Transactions on Information Forensics and Security*, 2(2):267 – 282, 2007.

[32] Medhat Moussa Xiaoguang Li and Shawki Areibi. Arithmetic formats for implementing aritificial neural networks on fpgas. *Canadian Journal of Electrial and Computer Engineering*, 331(1):31–40, 2006.

[33] Faycal Bensaali Xiaojun Zhai and Reza Sotudeh. Real-time optical character recognition on field programmable gate array for automatic number plate recognition system. *IET Circuits, Devices and Systems*, 7(6):337–344, 2013.

[34] Babak Zamanlooy and Mitra Mirhassani. Efficient vlsi implementation of neural networks with hyperbolic tangent activation function. *IEEE Transaction on VLSI Systems*, 33(2):39–4, 2014.

[35] Fa-Liang Chang Zhen-Xue Chen, Cheng-Yun Liu and Guo-You Wang. Automatic license-plate location and recognition based on feature salience. *IEEE Transations on Vehicular Technology*, 58(7):3781 – 3785, 2009.

Appendix A

Source Codes

Index

----------Matlab code----------

1. networkdesign_t1.m

```
% -------------------------networkdesign_t1-----------------------------------------
% This program test the network with the first 96 license plates. The
% network is first trained by using the data from the rest 290 license
% paltes.
% follow the 2 steps then run the program
%--------------------------------------------------------------------------------------
% 1. change the content in addpath('') to the directory where 'cropped
% images' is stored at. eg. addpath('D:\cropped images')
% 2. change the parameters (optional)
%--------------------------------------------------------------------------------------
addpath('C: \cropped images')
close all
clearvars -except k
load allCharacters
nTrainingData = 96;     %number of images are used for testing;
imageHeight = 21;       % size of input images;
imageWidth = 9;         % size of input images;
nNodes = 160;           % number of nodes in the second layer;
nIterations = 1000;   % number of training iterations;
margin = 0;             % a certain value that the correct output(largest output) needs to be larger than the
2nd largest output.
%% step 1 - accquire allInput
```

```
clearvars -except trainingInput trainingTarget nTrainingData ...
   modelProportion imageHeight imageWidth grayLevel allCharacters...
   nNodes nIterations k margin
allInput = getinput(imageHeight,imageWidth);   % allInput containing input patterns of all the charcters
(21*9)*8*386
%% step 2 - accqurire trainingdata ( need to use allInput so put it after getinput)
clearvars -except trainingInput trainingTarget nTrainingData ...
   modelProportion imageHeight imageWidth grayLevel allCharacters...
   nNodes nIterations k margin allInput
trainingInput = zeros(imageHeight*imageWidth,1);
trainingTarget = zeros(36,1)-1;
for m = nTrainingData+1:386  %LPs used as trainingdata
      for n = 1:8
         if (ischar(allCharacters{m*2,n}))
            trainingInput = [trainingInput allInput(:,n,m)];
            addedTarget = zeros(36,1);
            ascii = double(allCharacters{m*2,n});
            if ascii == 48                % '0'
               addedTarget(10) = 1;
            elseif ascii <58              %'1' - '9'
               addedTarget(ascii-48) = 1;
            else                          %'A' - 'Z'
               addedTarget(ascii - 54) = 1;
            end
            trainingTarget = [trainingTarget addedTarget];
         end
      end
end
trainingInput = trainingInput(:,2:end);
trainingTarget = trainingTarget(:,2:end);
%% step 3 - prepare testing input
testingInput1 = allInput(:,:,1:nTrainingData);     %testingCharacters1 is part of the allCharacters excluded
those used as training data
testingCharacters1 = allCharacters(1:nTrainingData*2,:);
testingInput2 = allInput(:,:,nTrainingData+1:end);  %testingInput2 and testingCharacters2 are used for
test the network with trainingdata
testingCharacters2 = allCharacters(nTrainingData*2+1:end,:);
%% step 4 - network training
for k = 1:1 % loop k controls the number of training and testing times
[neuralNetwork,trainingPerf] = networktraining(trainingInput,trainingTarget,nNodes,nIterations);
%% step 5 - network testing
clearvars -except trainingInput trainingTarget nTrainingData ...
   modelProportion imageHeight imageWidth grayLevel allCharacters testingInput2 testingCharacters2...
   nNodes nIterations margin allInput neuralNetwork testingInput1 testingCharacters1 k trainingInput
trainingTarget
for p = 1:1 % loop p controls the number of testing times
   [recognitionAccuracy,testResult] =
networktesting(neuralNetwork,testingCharacters2,testingInput2,imageHeight,imageWidth,margin);
   fprintf('k = %d,p = %d\n',k,p)
   fprintf('recallAccuracy= %f\n',recognitionAccuracy)
   clearvars recognitionAccuracy testResult
```

```
   [recognitionAccuracy,testResult] =
networktesting(neuralNetwork,testingCharacters1,testingInput1,imageHeight,imageWidth,margin);
   fprintf('recognitionAccuracy = %f\n',recognitionAccuracy)
end
end
```

2. getinput.m

```
% Fx() = getinput()
% PARAMETERS :   iamgeHeight imageWidth
function testingInput = getinput(imageHeight,imageWidth)
load allCharacters
testingInput = zeros(imageHeight*imageWidth,8,size(allCharacters,1)/2);
for m = 1:size(allCharacters,1)/2
   if (ischar(allCharacters{m*2,1})
   filename = sprintf('%d.jpg',m);
   tempLicense = imread(filename);
   for n = 1:8
      if (ischar(allCharacters{m*2,n}))
         tempImage = imcrop(tempLicense,allCharacters{m*2-1,n}); % crop character image
         tempImage = rgb2gray(tempImage);                        % gray
         tempImage = imcomplement(tempImage);       % invert black and white (make the background 0)
         tempImage = imresize(tempImage,[imageHeight,imageWidth]);       % resize
         grayLevel = graythresh(tempImage);                       % threshold
         tempImage = im2bw(tempImage,grayLevel);                  % binary
         testingInput(:,n,m) = reshape(tempImage,[imageHeight*imageWidth,1]); %make the binary
image a vector
      else
         break
      end
   end
   else
      ;
   end
end
```

3 . networktraining.m

```
% PARAMETERS : 1 nNodes
%              2 nIterations
function [neuralNetwork,trainingPerf] = networktraining(trainingInput,trainingTarget,nNodes,nIterations)
x = double(trainingInput);
t = trainingTarget;
c_network = network;
%-----------------Network dimensions-------------
c_network.numInputs = 1; % This number is the number of input sources,
% different from c_network.inputs{1}.size
c_network.numLayers = 2;
%-----------------Network connections------------
% in network connections configuration, elements are either 1 or 0.
c_network.biasConnect = [1;1];
```

```
c_network.inputConnect = [1;0];
c_network.layerConnect = [0 0;1 0];
c_network.outputConnect = [0 1];
%-----------------|Subobjects|---------------------
c_network.inputs{1}.size =size(trainingInput,1);
c_network.layers{1}.size = nNodes;
c_network.layers{1}.transferFcn = 'tansig';
c_network.layers{1}.initFcn = 'initnw';
% if initFcn is assigned to each layer, then in the functions: just leave
% initFcn as 'initlay'.
c_network.layers{2}.size = 36;
c_network.layers{2}.transferFcn = 'tansig';
c_network.layers{2}.initFcn = 'initnw';
%-----------------Functions-------------------------
c_network.trainFcn = 'trainscg';
c_network.divideFcn = 'dividerand';
c_network.divideParam.trainRatio = 0.9;
c_network.divideParam.valRatio = 0.1;
c_network.trainParam.goal = 10e-5;
c_network.trainParam.epochs = nIterations;
c_network.trainParam.max_fail = 6;
c_network.inputs{1}.processFcns={};
c_network.outputs{2}.processFcns={};

net = c_network;
[neuralNetwork,tr]= train(net,x,t);
trainingPerf =tr.best_perf;
fprintf('networkPerf=%f\t',trainingPerf)
```

4. networktesting.m

```
% networktesting
% this function compares the largest output and the second largest output
% a margin is set to ensure the correct output.
function [recognitionAccuracy,testResult] =
networktesting(neuralNetwork,testingCharacters,testingInput,imageHeight,imageWidth,margin)
networkOutput = NaN(size(testingCharacters,1)/2,8);
testResult = cell(size(testingCharacters,1)/2,8);
for m = 1:size(testingCharacters,1)/2
   for n = 1:8
      if (ischar(testingCharacters{m*2,n}))
         tempSequence = neuralNetwork(testingInput(:,n,m));
         largestOutput= find(tempSequence == max(tempSequence));
         tempSequence2 = tempSequence;     %tempSequence2 is a copy of the tempSequence
         tempSequence2(largestOutput) = 0; %in order to keep the original data after largestoutput being
set zero
         secondLargestOutput = find(tempSequence2 == max(tempSequence2));
        if tempSequence(largestOutput)-tempSequence(secondLargestOutput) > margin
         networkOutput(m,n) = largestOutput;
         switch networkOutput(m,n)
            case 1
```

```
    testResult{m,n} = '1';
case 2
    testResult{m,n} = '2';
case 3
    testResult{m,n} = '3';
case 4
    testResult{m,n} = '4';
case 5
    testResult{m,n} = '5';
case 6
    testResult{m,n} = '6';
case 7
    testResult{m,n} = '7';
case 8
    testResult{m,n} = '8';
case 9
    testResult{m,n} = '9';
case 10
    testResult{m,n} = '0';
case 11
    testResult{m,n} = 'A';
case 12
    testResult{m,n} = 'B';
case 13
    testResult{m,n} = 'C';
case 14
    testResult{m,n} = 'D';
case 15
    testResult{m,n} = 'E';
case 16
    testResult{m,n} = 'F';
case 17
    testResult{m,n} = 'G';
case 18
    testResult{m,n} = 'H';
case 19
    testResult{m,n} = 'I';
case 20
    testResult{m,n} = 'J';
case 21
    testResult{m,n} = 'K';
case 22
    testResult{m,n} = 'L';
case 23
    testResult{m,n} = 'M';
case 24
    testResult{m,n} = 'N';
case 25
    testResult{m,n} = 'O';
case 26
    testResult{m,n} = 'P';
```

```
            case 27
               testResult{m,n} = 'Q';
            case 28
               testResult{m,n} = 'R';
            case 29
               testResult{m,n} = 'S';
            case 30
               testResult{m,n} = 'T';
            case 31
               testResult{m,n} = 'U';
            case 32
               testResult{m,n} = 'V';
            case 33
               testResult{m,n} = 'W';
            case 34
               testResult{m,n} = 'X';
            case 35
               testResult{m,n} = 'Y';
            case 36
               testResult{m,n} = 'Z';
          end
        else
           ;
        end
      else
         ;
      end
   end
end
nCharacters = 0;
nCorrectCharacters = 0;
misChar = zeros(size(testingInput,1),1);
for m = 1:(size(testingCharacters,1)/2)
     for n = 1:8
       if (ischar(testingCharacters{m*2,n}))
            nCharacters = nCharacters + 1;
            if testingCharacters{m*2,n} == testResult{m,n}
               nCorrectCharacters = nCorrectCharacters + 1;
            elseif testingCharacters{m*2,n} == 'I'
               if testResult{m,n} == '1'
                  nCorrectCharacters = nCorrectCharacters + 1;
               else
               fprintf('target %c -> %c\n',testingCharacters{m*2,n},testResult{m,n})
               misChar = [misChar,testingInput(:,n,m)];
               end
            elseif testingCharacters{m*2,n} == '1'
               if testResult{m,n} == 'I'
                  nCorrectCharacters = nCorrectCharacters + 1;
               else
               fprintf('target %c -> %c\n',testingCharacters{m*2,n},testResult{m,n})
               misChar = [misChar,testingInput(:,n,m)];
```

```
            end
        elseif testingCharacters{m*2,n} == 'O'
            if testResult{m,n} == '0'
                nCorrectCharacters = nCorrectCharacters + 1;
            else
            fprintf('target %c -> %c\n',testingCharacters{m*2,n},testResult{m,n})
            misChar = [misChar,testingInput(:,n,m)];
            end
        elseif testingCharacters{m*2,n} == '0'
            if testResult{m,n} == 'O'
                nCorrectCharacters = nCorrectCharacters + 1;
            else
            fprintf('target %c -> %c\n',testingCharacters{m*2,n},testResult{m,n})
            misChar = [misChar,testingInput(:,n,m)];
            end
        else
            fprintf('target %c -> %c\n',testingCharacters{m*2,n},testResult{m,n})
            misChar = [misChar,testingInput(:,n,m)];
        end
    end
  end
end
misChar = misChar(:,2:end);
s = size(misChar,2);
r = ceil(s/5);
figure
for m = 1:s
  subplot(r,5,m)
  imshow(reshape(misChar(:,m),[imageHeight,imageWidth]));
end
recognitionAccuracy = nCorrectCharacters/nCharacters;
```

5. fixedpointnetwork.m

```
% This program is to test the network running in binary numbers
% input         w1         w2
% [1*189]  [189*160   [160*36]
% parameters : bitWidth1 - number of fractional bits
%                 resolution1
%                 bitWidth2
%                 resolution2
% evaluations : act1ae  (average error of act1)
%                 net1ae
%                 act2ae
%                 net2ae
% change the content in addpath('') to the directory where 'Binary
% Operation' is stored at. eg. addpath('D:\Binary Operation')
clear
addpath('D:\m_project\Matlab code backup\fixed point network\Binary Operation')
for i = 73:75% select tested license plates
  for j = 1:7% select characters (1-7: all characters in one license plate)
```

```
fprintf('i = %d; j = %d;\n',i,j)
bitWidth1 = 5;
resolution1 = 5;
bitWidth2 = 5;
resolution2 = 5;
%% load input, w1,b1,w2,b2
load('network160_t1_0983459.mat')
load 'htangent1.mat'
load 'htangent2.mat'
w1 = neuralNetwork.IW{1};
w1 = w1.';
w2 = neuralNetwork.LW{2};
w2 = w2.';
b1 = neuralNetwork.b{1};
b2 = neuralNetwork.b{2};
%w1 = [w1;b1'];
%w2 = [w2;b2'];

load('testingInput1_160.mat')
testingInput1 = testingInput1(:,:,i); %testingInput1 size = [189*8*96]
testingInput1 = testingInput1';
testingInput1 = testingInput1(j,:);
iAct1 = testingInput1*w1+b1';
iNet1 = tanh(iAct1);
iAct2 = iNet1*w2+b2';
iNet2 = tanh(iAct2);
tic
%% act1 = input*w1 + b1  'act' is activation
act1 = zeros(160,1+6+bitWidth1);
for m = 1:160
   for n = 1:size(testingInput1,2)
      if testingInput1(n) == 1
        act1(m,:) = binaryadder(act1(m,:),decimal2binary(w1(n,m),6,bitWidth1));
      end
      a1 = 1; % let program stops here. doing nothing.
   end
   act1(m,:) = binaryadder(act1(m,:),decimal2binary(b1(m),6,bitWidth1));   %act1 2's complement
end
%% net1 = f(act1)        f() is tansig
net1 = zeros(160,1+resolution1);
for m = 1:160
   temp = com2mag(act1(m,:));
   seq = binary2decimal([0 temp(2:6+resolution1+1)],0);
   if seq == 0
      net1(m,:) = zeros(1,1+resolution1);
   else
      if resolution1 == 5
         net1(m,:)=htangent1(seq,:);
      elseif resolution1 == 6
         net1(m,:)=htangent2(seq,:);
      end
```

```
   end
   if act1(m,1) == 0
      ;
   else
      net1(m,1) = 1;                % net1 sign magnitude
   end
end
%% act2 = net1*w2 + b2
%multiplication: 1-6-7 * 1-6-7 = 1-6-7
act2 = zeros(36,1+6+bitWidth2);
for m = 1:36
      xyz = 1; % <====== debug1 doing nothing, just let program stops here
   debug1 = zeros(160,14); % <=======debug1
   for n = 1:160
      if net1(n,1) == 0
         temp = signedmultiplier([0 0 0 0 0 0 0 net1(n,2:end) zeros(1,bitWidth2-
5)],com2mag(decimal2binary(w2(n,m),6,bitWidth2)));

         % signedmultiplier(sign magnitude)
         act2(m,:) = binaryadder(act2(m,:),com2mag([temp(1),temp(8:13),temp(14:13+bitWidth2)]));
         % binaryadder(2's complement)
      else
         temp = signedmultiplier([1 0 0 0 0 0 0 net1(n,2:end) zeros(1,bitWidth2-
5)],com2mag(decimal2binary(w2(n,m),6,bitWidth2)));
         act2(m,:) = binaryadder(act2(m,:),com2mag([temp(1),temp(8:13),temp(14:13+bitWidth2)]));
      end
  %    debug1(n,:) =com2mag( [temp(1),temp(8:13),temp(14:13+bitWidth2)]); %
<=============debug1
 %      xyz = 1; % <====== debug1 doing nothing, just let program stops here
   end
   act2(m,:) = binaryadder(act2(m,:),decimal2binary(b2(m),6,bitWidth2));          %act2 2's complement
end
%% net2 = f(act2)
net2 = zeros(36,1+resolution2);
for m = 1:36
   temp = com2mag(act2(m,:));
   seq = binary2decimal([0 temp(2:6+resolution2+1)],0);
   if seq == 0
      net2(m,:) = zeros(1,resolution2+1);
   else
      if resolution2 == 5
        net2(m,:) = htangent1(seq,:);
      elseif resolution2 == 6
        net2(m,:) = htangent2(seq,:);
      end
   end
   if act2(m,1) == 0
      ;
   else
      net2(m,1) = 1;             % net2 sign magnitude
   end
```

```
end
%% evaluation
act1e = zeros(1,160);
for m = 1:160
   act1e(m) = binary2decimal(com2mag(act1(m,:)),(size(act1,2)-1-6));
end
net1e = zeros(1,160);
for m = 1:160
   net1e(m) = binary2decimal(net1(m,:),resolution1);
end
act2e = zeros(1,36);
for m = 1:36
   act2e(m) = binary2decimal(com2mag(act2(m,:)),(size(act2,2)-1-6));
end
net2e = zeros(1,36);
for m = 1:36
   net2e(m) = binary2decimal(net2(m,:),resolution2);
end
toc
mean(abs(act1e-iAct1))
mean(abs(net1e-iNet1))
mean(abs(act2e-iAct2))
mean(abs(net2e-iNet2))
   end
end
```

6. binary2decimal.m

```
% This function converts signed binary numbers to decimal numbers.
function result = binary2decimal(input,fraction)
sign = input(1);
s = size(input);
s = s(2);
a = input(2:s);
a = fliplr(a);
result = bi2de(a)*power(2,-fraction);
if sign == 1
   result = -result;
else
   ;
end
```

7. binaryadder.m

```
% This function add two numbers in the form of 2's complement
function answer = binaryadder(input1,input2)
s = size(input1);
s = s(2);
c = 0;
answer = zeros(1,s);
for m = 1 : s
```

```
   a = input1(s + 1 - m);
   b = input2(s + 1 - m);
   answer(s+1-m) =  xor(xor(a,b),c);
      if and(and((a == 0) , (b == 0)), (c == 0))
         c = 0;
      elseif and(and((a == 0) , (b == 0)), (c == 1))
         c = 0;
      elseif and(and((a == 0) , (b == 1)), (c == 0))
         c = 0;
      elseif and(and((a == 1) , (b == 0)), (c == 0))
         c = 0;
      else
         c = 1;
      end
end
```

8 . com2mag.m

```
% This function converts binary numbers in the form of
% 2's complement or signed magnitude from one to another.
function answer = com2mag(input)
s = size(input,2);
c = 0;
answer = zeros(1,s);
if input(1) == 0
   answer = input;
elseif sum(input,2) == 1
   answer = zeros(1,s);
else
   answer(1)=1;
   for m = 2:s
      if input(m) == 0
         answer(m) = 1;
      else
         answer(m) = 0;
      end
   end
   adder = zeros(1,s);
   adder(s) = 1;
   for n = 1:s-1
      a = answer(s+1-n);
      b = adder(s+1-n);
      answer(s+1-n) =  xor(xor(a,b),c);
      x = 10;
      if sum([a b c])<2
         c = 0;
      else
         c = 1;
      end

   end
```

```
end
```

9. decimal2binary.m

```
% This function convert decimal numbers to
% binary numbers in the form of 2's complimentary.
% generic parameters:
%                          integer : number of integer bits
%                          fraction : number of fractional bits
% variables :
%                 input,
%                   a,
%                   alast,
%                   s,
%                   val,
%                   vlast,
%                   error,
%                   answer
% intermediate variables:
%                          m, n
function [answer, val] = decimal2binary(input,integer,fraction)

val = 0;
a = 0;
error = power(2,integer+1);
if ( input >= power(2,integer) )or (input <= -power(2,integer))
   answer = 'out of range';
else
   for m = 1:power(2,integer+fraction)
      alast = a; % alast is a variable to store the value of 'a' in the last cycle
      vlast = val;
      a = de2bi(m-1);
      a = fliplr(a);
      s = size(a);
      s = s(2);
      while s < (integer + fraction)
         a = [0 a];
         s = s + 1;
      end
      val = 0;
      for n = 1:(integer+fraction)
         val = val + power(2,integer - n).*a(n);
      end
         if input >= 0
            ;
         else
            val = val - power(2,integer);
         end
        if m == power(2,integer+fraction);  % if the loop reaches the last cycle but
            answer = a;                     % the error is still decreasing, then the
         else                                 %  current a is the best result.
```

```
            ;
        end
    if abs(val-input) < error
        error = abs(val-input);
    else
        answer = alast;
        val = vlast;
        break
    end
  end
end
if input >= 0
  answer = [0 answer];
else
  answer = [1 answer];
end
```

10. signedmultiplier.m

```
% This function multiply two binary numbers in the form of signed magitude
function result = signedmultiplier(input1,input2)
s = size(input1,2);
accu = 0;
in1 = input1(2:s);
in2 = input2(2:s);
in2 = fliplr(in2);
for m = 1:s-1
  if m == 1
    if in2(m) == 1
      acc = in1;
    else
      acc = zeros(1,s-1);
    end
  elseif m == 2
        acc = [0 0 acc];
        in1 = [0 in1 0];
      if in2(m) == 1
    acc = binaryadder(in1,acc);
      else
        ;
      end
  else
        acc = [0 acc];
        in1 = [in1 0];
      if in2(m) == 1
    acc = binaryadder(in1,acc);
      else
        ;
      end
  end
end
```

```
 s1 = input1(1);
 s2 = input2(1);
 if xor(s1,s2)
    result = [1 acc];
 else
    result = [0 acc];
 end
```

11. tanhfunction.m

```
% this function is to automatically build a hyperbolic tangent function
% based on the method in the paper of EFFICIENT VLSI IMPLEMENTATION OF ...

% input   :   e - Maximum allowable error
%             ri - Input range

% output  :  Ni   - Number of interger bits
%            Nf   - Number of fractional bits
%            Np   - The place of the first non-zero bit of the input which
%                   makes it still in the passing region.
%            Ns   - The place of the first non-zero bit of the input which
%                   makes it still in the saturation region.
%            Nprocessing - each element in this vector represents the
%                          number of sub-subranges in that subrange.
%            Papprox - a cell array containing the approximate values for
%                              each subrange.
%% calculate Ni and Nf (integer, franctional)
function [Ni Nf Np Ns Nprocessing Papprox] = tanhfunction(e,ri)

Ni = 0;
while (power(2,Ni)<ri)
   Ni = Ni + 1;
end
Nf = 0;
while (power(2,-Nf)>e)
   Nf = Nf + 1;
end
%% decide xpaq and xsq (boundries between 3 regions)
syms x
equation = 1 - power(2,-Nf)-tanh(x) == e;
xsq = solve(equation,x);
xsq = double(xsq);
equation = x - tanh(x) == e;
xpaq = solve(equation,x);
xpaq = double(xpaq);
%% calculate Ns
Ns = 0;              % None = Ns when it enters saturation region
while (power(2,Ns) < xsq)
   Ns = Ns + 1;
end
```

```
Np = 0;         % None = Np when it enters passing region
%% calculate Np
while (power(2,Np) >xpaq)
   Np = Np - 1;
end
Np = Np - 1;
%% calculate Nprocessing
ea = 0.04; % ea = e - power(2,-Nf-1);   % approximation error (eq.23)
Nprocessing = zeros(1,Ns-Np-1);
for q = 1:Ns-Np-1  %========= for None = -1:0
   None = Np + q;
   flag = 1;
      for m = 1:Nf+None+1    %========= for i = Nf+None : 0     when i = 0, N get max value
            if flag == 1
               i = Nf+None+1-m;   % when i = Nf+None, N = 1
               N = power(2,None+Nf-i);
                     for n = 1:N        % ======== for each subrange, ...
                        j = n - 1;     % j is the ordinal number of subranges
                        Nsubsubrange = power(2,i); % Nsubsubrange is the number of 2^(-Nf) within one
subrange
                        approx = 0;
                              for p = 1: Nsubsubrange+1  % ======== for each basic intervals in a subrange
                                 k = p - 1;
                                 approx = approx + tanh(power(2,None)*(1+j/N)+k*power(2,-Nf));
%(eq.12)
                              end
                        approx = approx/(Nsubsubrange+1);
                              if abs(tanh(power(2,None)*(1+j/N))-approx)>ea ||
abs(tanh(power(2,None)*(1+(j+1)/N))-approx)>ea
                                 break % if error exceeds the limit, go back to the m loop, N = N*2
                              elseif n == N
                                Nprocessing(q) = N;
                                flag = 0;
                                break
                              end
                     end
            else
               break
            end
      end
end
%% calculate Papprox  - approximate values for subranges
Papprox = cell(1,size(Nprocessing,2));
for m = 1: size(Nprocessing,2)
   Papprox{m} = zeros(Nprocessing(m),1);
end
for p = 1:size(Nprocessing,2)        %----ranges

   for m = 1:Nprocessing(p)     %----subranges
      j = m - 1;                % j ranges from 0 to N-1 (N is the number of subranges)
      approx = 0;
```

```
      for n = 1:(power(2,Np+p+Nf)/Nprocessing(p))+1 %----sub-subranges
          k = n - 1;              % k ranges from 0 to 2^i (2^i is the number of sub-subranges)
          approx = approx + tanh(power(2,Np+p)*(1+j/Nprocessing(p))+k*power(2,-Nf));
      end
      Papprox{p} (m)= approx/(power(2,Np+p+Nf)/Nprocessing(p)+1);
   end
end
```

----------Verilog code----------

1. adder161.v

```
module adder161(in1,in2,in3,in4,in5,in6,in7,in8,in9,in10,
                in11,in12,in13,in14,in15,in16,in17,in18,in19,in20,
                in21,in22,in23,in24,in25,in26,in27,in28,in29,in30,
                in31,in32,in33,in34,in35,in36,in37,in38,in39,in40,
                in41,in42,in43,in44,in45,in46,in47,in48,in49,in50,
                in51,in52,in53,in54,in55,in56,in57,in58,in59,in60,
                in61,in62,in63,in64,in65,in66,in67,in68,in69,in70,
                in71,in72,in73,in74,in75,in76,in77,in78,in79,in80,
                in81,in82,in83,in84,in85,in86,in87,in88,in89,in90,
                in91,in92,in93,in94,in95,in96,in97,in98,in99,in100,
                in101,in102,in103,in104,in105,in106,in107,in108,in109,in110,
                in111,in112,in113,in114,in115,in116,in117,in118,in119,in120,
                in121,in122,in123,in124,in125,in126,in127,in128,in129,in130,
                in131,in132,in133,in134,in135,in136,in137,in138,in139,in140,
                in141,in142,in143,in144,in145,in146,in147,in148,in149,in150,
                in151,in152,in153,in154,in155,in156,in157,in158,in159,in160,
                in161,
                clk,out);

input [13:0]   in1,in2,in3,in4,in5,in6,in7,in8,in9,in10,
                in11,in12,in13,in14,in15,in16,in17,in18,in19,in20,
                in21,in22,in23,in24,in25,in26,in27,in28,in29,in30,
                in31,in32,in33,in34,in35,in36,in37,in38,in39,in40,
                in41,in42,in43,in44,in45,in46,in47,in48,in49,in50,
                in51,in52,in53,in54,in55,in56,in57,in58,in59,in60,
                in61,in62,in63,in64,in65,in66,in67,in68,in69,in70,
                in71,in72,in73,in74,in75,in76,in77,in78,in79,in80,
                in81,in82,in83,in84,in85,in86,in87,in88,in89,in90,
                in91,in92,in93,in94,in95,in96,in97,in98,in99,in100,
                in101,in102,in103,in104,in105,in106,in107,in108,in109,in110,
                in111,in112,in113,in114,in115,in116,in117,in118,in119,in120,
                in121,in122,in123,in124,in125,in126,in127,in128,in129,in130,
                in131,in132,in133,in134,in135,in136,in137,in138,in139,in140,
                in141,in142,in143,in144,in145,in146,in147,in148,in149,in150,
                in151,in152,in153,in154,in155,in156,in157,in158,in159,in160,
                in161;
input clk;
output [13:0] out;
reg [13:0] out;
```

```
reg [13:0] a1,a2,a3,a4,a5,a6,a7,a8,a9,a10;
reg [13:0] a11,a12,a13,a14,a15,a16,a17,a18,a19,a20;
reg [13:0] a21,a22,a23,a24,a25,a26,a27,a28,a29,a30;
reg [13:0] a31,a32,a33,a34,a35,a36,a37,a38,a39,a40;
reg [13:0] a41,a42,a43,a44,a45,a46,a47,a48,a49,a50;
reg [13:0] a51,a52,a53,a54,a55,a56,a57,a58,a59,a60;
reg [13:0] a61,a62,a63,a64,a65,a66,a67,a68,a69,a70;
reg [13:0] a71,a72,a73,a74,a75,a76,a77,a78,a79,a80;
reg [13:0] a81;

reg [13:0] b1,b2,b3,b4,b5,b6,b7,b8,b9,b10;
reg [13:0] b11,b12,b13,b14,b15,b16,b17,b18,b19,b20;
reg [13:0] b21,b22,b23,b24,b25,b26,b27,b28,b29,b30;
reg [13:0] b31,b32,b33,b34,b35,b36,b37,b38,b39,b40;
reg [13:0] b41;

reg [13:0] c1,c2,c3,c4,c5,c6,c7,c8,c9,c10;
reg [13:0] c11,c12,c13,c14,c15,c16,c17,c18,c19,c20;
reg [13:0] c21;

reg [13:0] d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11;

reg [13:0] e1,e2,e3,e4,e5,e6;

reg [13:0] f1,f2,f3;

reg [13:0] g1,g2;

always@(posedge clk)
begin

a1 <= in1+in2;
a2 <= in3+in4;
a3 <= in5+in6;
a4 <= in7+in8;
a5 <= in9+in10;
a6 <= in11+in12;
a7 <= in13+in14;
a8 <= in15+in16;
a9 <= in17+in18;
a10 <= in19+in20;
a11 <= in21+in22;
a12 <= in23+in24;
a13 <= in25+in26;
a14 <= in27+in28;
a15 <= in29+in30;
a16 <= in31+in32;
a17 <= in33+in34;
a18 <= in35+in36;
a19 <= in37+in38;
a20 <= in39+in40;
```

```
a21 <= in41+in42;
a22 <= in43+in44;
a23 <= in45+in46;
a24 <= in47+in48;
a25 <= in49+in50;
a26 <= in51+in52;
a27 <= in53+in54;
a28 <= in55+in56;
a29 <= in57+in58;
a30 <= in59+in60;

a31 <= in61+in62;
a32 <= in63+in64;
a33 <= in65+in66;
a34 <= in67+in68;
a35 <= in69+in70;
a36 <= in71+in72;
a37 <= in73+in74;
a38 <= in75+in76;
a39 <= in77+in78;
a40 <= in79+in80;
a41 <= in81+in82;
a42 <= in83+in84;
a43 <= in85+in86;
a44 <= in87+in88;
a45 <= in89+in90;
a46 <= in91+in92;
a47 <= in93+in94;
a48 <= in95+in96;
a49 <= in97+in98;
a50 <= in99+in100;
a51 <= in101+in102;
a52 <= in103+in104;
a53 <= in105+in106;
a54 <= in107+in108;
a55 <= in109+in110;
a56 <= in111+in112;
a57 <= in113+in114;
a58 <= in115+in116;
a59 <= in117+in118;
a60 <= in119+in120;

a61 <= in121+in122;
a62 <= in123+in124;
a63 <= in125+in126;
a64 <= in127+in128;
a65 <= in129+in130;
a66 <= in131+in132;
a67 <= in133+in134;
a68 <= in135+in136;
a69 <= in137+in138;
```

```
a70 <= in139+in140;
a71 <= in141+in142;
a72 <= in143+in144;
a73 <= in145+in146;
a74 <= in147+in148;
a75 <= in149+in150;
a76 <= in151+in152;
a77 <= in153+in154;
a78 <= in155+in156;
a79 <= in157+in158;
a80 <= in159+in160;
a81 <= in161;

b1 <= a1+a2;
b2 <= a3+a4;
b3 <= a5+a6;
b4 <= a7+a8;
b5 <= a9+a10;
b6 <= a11+a12;
b7 <= a13+a14;
b8 <= a15+a16;
b9 <= a17+a18;
b10 <= a19+a20;
b11 <= a21+a22;
b12 <= a23+a24;
b13 <= a25+a26;
b14 <= a27+a28;
b15 <= a29+a30;
b16 <= a31+a32;
b17 <= a33+a34;
b18 <= a35+a36;
b19 <= a37+a38;
b20 <= a39+a40;
b21 <= a41+a42;
b22 <= a43+a44;
b23 <= a45+a46;
b24 <= a47+a48;
b25 <= a49+a50;
b26 <= a51+a52;
b27 <= a53+a54;
b28 <= a55+a56;
b29 <= a57+a58;
b30 <= a59+a60;
b31 <= a61+a62;
b32 <= a63+a64;
b33 <= a65+a66;
b34 <= a67+a68;
b35 <= a69+a70;
b36 <= a71+a72;
b37 <= a73+a74;
b38 <= a75+a76;
```

```
b39 <= a77+a78;
b40 <= a79+a80;
b41 <= a81;

c1 <= b1+b2;
c2 <= b3+b4;
c3 <= b5+b6;
c4 <= b7+b8;
c5 <= b9+b10;
c6 <= b11+b12;
c7 <= b13+b14;
c8 <= b15+b16;
c9 <= b17+b18;
c10 <= b19+b20;
c11 <= b21+b22;
c12 <= b23+b24;
c13 <= b25+b26;
c14 <= b27+b28;
c15 <= b29+b30;
c16 <= b31+b32;
c17 <= b33+b34;
c18 <= b35+b36;
c19 <= b37+b38;
c20 <= b39+b40;
c21 <= b41;

d1 <= c1+c2;
d2 <= c3+c4;
d3 <= c5+c6;
d4 <= c7+c8;
d5 <= c9+c10;
d6 <= c11+c12;
d7 <= c13+c14;
d8 <= c15+c16;
d9 <= c17+c18;
d10 <= c19+c20;
d11 <= c21;

e1 <= d1+d2;
e2 <= d3+d4;
e3 <= d5+d6;
e4 <= d7+d8;
e5 <= d9+d10;
e6 <= d11;

f1 <= e1+e2;
f2 <= e3+e4;
f3 <= e5+e6;

g1 <= f1+f2;
g2 <= f3;
```

```
out <= g1+g2;

end
endmodule
```

2. adder190.v

```
module adder190(in1,in2,in3,in4,in5,in6,in7,in8,in9,in10,
                in11,in12,in13,in14,in15,in16,in17,in18,in19,in20,
                in21,in22,in23,in24,in25,in26,in27,in28,in29,in30,
                in31,in32,in33,in34,in35,in36,in37,in38,in39,in40,
                in41,in42,in43,in44,in45,in46,in47,in48,in49,in50,
                in51,in52,in53,in54,in55,in56,in57,in58,in59,in60,
                in61,in62,in63,in64,in65,in66,in67,in68,in69,in70,
                in71,in72,in73,in74,in75,in76,in77,in78,in79,in80,
                in81,in82,in83,in84,in85,in86,in87,in88,in89,in90,
                in91,in92,in93,in94,in95,in96,in97,in98,in99,in100,
                in101,in102,in103,in104,in105,in106,in107,in108,in109,in110,
                in111,in112,in113,in114,in115,in116,in117,in118,in119,in120,
                in121,in122,in123,in124,in125,in126,in127,in128,in129,in130,
                in131,in132,in133,in134,in135,in136,in137,in138,in139,in140,
                in141,in142,in143,in144,in145,in146,in147,in148,in149,in150,
                in151,in152,in153,in154,in155,in156,in157,in158,in159,in160,
                in161,in162,in163,in164,in165,in166,in167,in168,in169,in170,
                in171,in172,in173,in174,in175,in176,in177,in178,in179,in180,
                in181,in182,in183,in184,in185,in186,in187,in188,in189,in190,
                clk,out);

input [11:0]   in1,in2,in3,in4,in5,in6,in7,in8,in9,in10,
                in11,in12,in13,in14,in15,in16,in17,in18,in19,in20,
                in21,in22,in23,in24,in25,in26,in27,in28,in29,in30,
                in31,in32,in33,in34,in35,in36,in37,in38,in39,in40,
                in41,in42,in43,in44,in45,in46,in47,in48,in49,in50,
                in51,in52,in53,in54,in55,in56,in57,in58,in59,in60,
                in61,in62,in63,in64,in65,in66,in67,in68,in69,in70,
                in71,in72,in73,in74,in75,in76,in77,in78,in79,in80,
                in81,in82,in83,in84,in85,in86,in87,in88,in89,in90,
                in91,in92,in93,in94,in95,in96,in97,in98,in99,in100,
                in101,in102,in103,in104,in105,in106,in107,in108,in109,in110,
                in111,in112,in113,in114,in115,in116,in117,in118,in119,in120,
                in121,in122,in123,in124,in125,in126,in127,in128,in129,in130,
                in131,in132,in133,in134,in135,in136,in137,in138,in139,in140,
                in141,in142,in143,in144,in145,in146,in147,in148,in149,in150,
                in151,in152,in153,in154,in155,in156,in157,in158,in159,in160,
                in161,in162,in163,in164,in165,in166,in167,in168,in169,in170,
                in171,in172,in173,in174,in175,in176,in177,in178,in179,in180,
                in181,in182,in183,in184,in185,in186,in187,in188,in189,in190;
input clk;
output [11:0] out;
reg [11:0] out;
```

```
reg [11:0] a1,a2,a3,a4,a5,a6,a7,a8,a9,a10;
reg [11:0] a11,a12,a13,a14,a15,a16,a17,a18,a19,a20;
reg [11:0] a21,a22,a23,a24,a25,a26,a27,a28,a29,a30;
reg [11:0] a31,a32,a33,a34,a35,a36,a37,a38,a39,a40;
reg [11:0] a41,a42,a43,a44,a45,a46,a47,a48,a49,a50;
reg [11:0] a51,a52,a53,a54,a55,a56,a57,a58,a59,a60;
reg [11:0] a61,a62,a63,a64,a65,a66,a67,a68,a69,a70;
reg [11:0] a71,a72,a73,a74,a75,a76,a77,a78,a79,a80;
reg [11:0] a81,a82,a83,a84,a85,a86,a87,a88,a89,a90;
reg [11:0] a91,a92,a93,a94,a95;

reg [11:0] b1,b2,b3,b4,b5,b6,b7,b8,b9,b10;
reg [11:0] b11,b12,b13,b14,b15,b16,b17,b18,b19,b20;
reg [11:0] b21,b22,b23,b24,b25,b26,b27,b28,b29,b30;
reg [11:0] b31,b32,b33,b34,b35,b36,b37,b38,b39,b40;
reg [11:0] b41,b42,b43,b44,b45,b46,b47,b48;

reg [11:0] c1,c2,c3,c4,c5,c6,c7,c8,c9,c10;
reg [11:0] c11,c12,c13,c14,c15,c16,c17,c18,c19,c20;
reg [11:0] c21,c22,c23,c24;

reg [11:0] d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12;

reg [11:0] e1,e2,e3,e4,e5,e6;

reg [11:0] f1,f2,f3;

reg [11:0] g1,g2;

always@(posedge clk)
begin

a1 <= in1+in2;
a2 <= in3+in4;
a3 <= in5+in6;
a4 <= in7+in8;
a5 <= in9+in10;
a6 <= in11+in12;
a7 <= in13+in14;
a8 <= in15+in16;
a9 <= in17+in18;
a10 <= in19+in20;
a11 <= in21+in22;
a12 <= in23+in24;
a13 <= in25+in26;
a14 <= in27+in28;
a15 <= in29+in30;
a16 <= in31+in32;
a17 <= in33+in34;
a18 <= in35+in36;
```

```
a19 <= in37+in38;
a20 <= in39+in40;
a21 <= in41+in42;
a22 <= in43+in44;
a23 <= in45+in46;
a24 <= in47+in48;
a25 <= in49+in50;
a26 <= in51+in52;
a27 <= in53+in54;
a28 <= in55+in56;
a29 <= in57+in58;
a30 <= in59+in60;

a31 <= in61+in62;
a32 <= in63+in64;
a33 <= in65+in66;
a34 <= in67+in68;
a35 <= in69+in70;
a36 <= in71+in72;
a37 <= in73+in74;
a38 <= in75+in76;
a39 <= in77+in78;
a40 <= in79+in80;
a41 <= in81+in82;
a42 <= in83+in84;
a43 <= in85+in86;
a44 <= in87+in88;
a45 <= in89+in90;
a46 <= in91+in92;
a47 <= in93+in94;
a48 <= in95+in96;
a49 <= in97+in98;
a50 <= in99+in100;
a51 <= in101+in102;
a52 <= in103+in104;
a53 <= in105+in106;
a54 <= in107+in108;
a55 <= in109+in110;
a56 <= in111+in112;
a57 <= in113+in114;
a58 <= in115+in116;
a59 <= in117+in118;
a60 <= in119+in120;

a61 <= in121+in122;
a62 <= in123+in124;
a63 <= in125+in126;
a64 <= in127+in128;
a65 <= in129+in130;
a66 <= in131+in132;
a67 <= in133+in134;
```

```
a68 <= in135+in136;
a69 <= in137+in138;
a70 <= in139+in140;
a71 <= in141+in142;
a72 <= in143+in144;
a73 <= in145+in146;
a74 <= in147+in148;
a75 <= in149+in150;
a76 <= in151+in152;
a77 <= in153+in154;
a78 <= in155+in156;
a79 <= in157+in158;
a80 <= in159+in160;
a81 <= in161+in162;
a82 <= in163+in164;
a83 <= in165+in166;
a84 <= in167+in168;
a85 <= in169+in170;
a86 <= in171+in172;
a87 <= in173+in174;
a88 <= in175+in176;
a89 <= in177+in178;
a90 <= in179+in180;
a91 <= in181+in182;
a92 <= in183+in184;
a93 <= in185+in186;
a94 <= in187+in188;
a95 <= in189+in190;

b1 <= a1+a2;
b2 <= a3+a4;
b3 <= a5+a6;
b4 <= a7+a8;
b5 <= a9+a10;
b6 <= a11+a12;
b7 <= a13+a14;
b8 <= a15+a16;
b9 <= a17+a18;
b10 <= a19+a20;
b11 <= a21+a22;
b12 <= a23+a24;
b13 <= a25+a26;
b14 <= a27+a28;
b15 <= a29+a30;
b16 <= a31+a32;
b17 <= a33+a34;
b18 <= a35+a36;
b19 <= a37+a38;
b20 <= a39+a40;
b21 <= a41+a42;
b22 <= a43+a44;
```

```
b23 <= a45+a46;
b24 <= a47+a48;
b25 <= a49+a50;
b26 <= a51+a52;
b27 <= a53+a54;
b28 <= a55+a56;
b29 <= a57+a58;
b30 <= a59+a60;
b31 <= a61+a62;
b32 <= a63+a64;
b33 <= a65+a66;
b34 <= a67+a68;
b35 <= a69+a70;
b36 <= a71+a72;
b37 <= a73+a74;
b38 <= a75+a76;
b39 <= a77+a78;
b40 <= a79+a80;
b41 <= a81+a82;
b42 <= a83+a84;
b43 <= a85+a86;
b44 <= a87+a88;
b45 <= a89+a90;
b46 <= a91+a92;
b47 <= a93+a94;
b48 <= a95;


c1 <= b1+b2;
c2 <= b3+b4;
c3 <= b5+b6;
c4 <= b7+b8;
c5 <= b9+b10;
c6 <= b11+b12;
c7 <= b13+b14;
c8 <= b15+b16;
c9 <= b17+b18;
c10 <= b19+b20;
c11 <= b21+b22;
c12 <= b23+b24;
c13 <= b25+b26;
c14 <= b27+b28;
c15 <= b29+b30;
c16 <= b31+b32;
c17 <= b33+b34;
c18 <= b35+b36;
c19 <= b37+b38;
c20 <= b39+b40;
c21 <= b41+b42;
c22 <= b43+b44;
c23 <= b45+b46;
```

```
c24 <= b47+b48;

d1 <= c1+c2;
d2 <= c3+c4;
d3 <= c5+c6;
d4 <= c7+c8;
d5 <= c9+c10;
d6 <= c11+c12;
d7 <= c13+c14;
d8 <= c15+c16;
d9 <= c17+c18;
d10 <= c19+c20;
d11 <= c21+c22;
d12 <= c23+c24;

e1 <= d1+d2;
e2 <= d3+d4;
e3 <= d5+d6;
e4 <= d7+d8;
e5 <= d9+d10;
e6 <= d11+d12;

f1 <= e1+e2;
f2 <= e3+e4;
f3 <= e5+e6;

g1 <= f1+f2;
g2 <= f3;

out <= g1+g2;

end
endmodule
```

3. clk_gen.v

```
/* This module is the clock generator module
it uses a megawizard generated pll file - lpm_pll as core module.*/
module clk_gen(clk_source,clk,clk_inv,clk_phs,start);
input clk_source;
output clk,clk_inv,clk_phs,start;
reg a,b;
wire c;
lpm_pll pll_1(clk_source,clk,clk_inv,clk_phs,c);

always @(posedge clk_source)
begin
a <= c;
b <= a;
end
assign start = ~b; // the output port 'start' from 'locked' is
```

```
endmodule          // going through two FF and an Inverter
```

4. com2mag165.v

```
module com2mag165(in,out);

input [11:0] in;
output [11:0] out;

wire [11:0] in,out;

assign out[0]=in[0];
assign out[1]=in[1]^(in[11]&in[0]);
assign out[2]=in[2]^(in[11]&(in[0]|in[1]));
assign out[3]=in[3]^(in[11]&(in[0]|in[1]|in[2]));
assign out[4]=in[4]^(in[11]&(in[0]|in[1]|in[2]|in[3]));
assign out[5]=in[5]^(in[11]&(in[0]|in[1]|in[2]|in[3]|in[4]));
assign out[6]=in[6]^(in[11]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]));
assign out[7]=in[7]^(in[11]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]));
assign out[8]=in[8]^(in[11]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7]));
assign out[9]=in[9]^(in[11]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7]|in[8]));
assign out[10]=in[10]^(in[11]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7]|in[8]|in[9]));
assign out[11]=in[11];

endmodule
```

5. com2mag167.v

```
module com2mag167(in,out);

input [13:0] in;
output [13:0] out;

wire [13:0] in,out;

assign out[0]=in[0];
assign out[1]=in[1]^(in[13]&in[0]);
assign out[2]=in[2]^(in[13]&(in[0]|in[1]));
assign out[3]=in[3]^(in[13]&(in[0]|in[1]|in[2]));
assign out[4]=in[4]^(in[13]&(in[0]|in[1]|in[2]|in[3]));
assign out[5]=in[5]^(in[13]&(in[0]|in[1]|in[2]|in[3]|in[4]));
assign out[6]=in[6]^(in[13]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]));
assign out[7]=in[7]^(in[13]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]));
assign out[8]=in[8]^(in[13]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7]));
assign out[9]=in[9]^(in[13]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7]|in[8]));
assign out[10]=in[10]^(in[13]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7]|in[8]|in[9]));
assign out[11]=in[11]^(in[13]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7]|in[8]|in[9]|in[10]));
assign out[12]=in[12]^(in[13]&(in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7]|in[8]|in[9]|in[10]|in[11]));
assign out[13]=in[13]&&((((in[1]||in[2])||(in[3]||in[4]))||((in[5]||in[6])||(in[7]||in[8]))) ||
(((in[9]||in[10])||(in[11]||in[12]))||in[0]) );
```

endmodule

6. comparator.v

```
module comparator(in,addr,clk,clk_inv,wr_en,counter,out);
input [5:0] in;
input [5:0] addr;
input clk,clk_inv,wr_en;
input [7:0] counter;
output [5:0] out;

reg [5:0] result,out;

reg [5:0] a,b,subtract1,subtract2,subtract3,max;
always @ (posedge clk)
begin
	if (wr_en == 1)
	begin
		if (addr == 0)
		begin
			a = in;		// a is the current largest number
			b = 6'b111_111;  // b is the one to compare with
			max = addr;
		end
		else
		begin
			b = in;
			subtract1 = {1'b0,a[4:0]};
			subtract2 = {1'b0,b[4:0]};
			subtract3 = subtract1 - subtract2;
			case ({a[5],b[5]})
			2'b10:
				begin
					a = b;
					max = addr;
				end
			2'b00:
				begin
					if (subtract3[5] == 1)
					begin
						a = b;
						max = addr;
					end
				end
			2'b11:
				begin
					if (subtract3[5] == 0)
					begin
						a = b;
						max = addr;
					end
```

```
                                end
                        endcase
                end
                result = max;
        end
        else    result = 6'b111_111;
end
always @ (posedge clk_inv)
begin
if (counter == 8'b1101_1010)     //88-123 98-133 (1000_0110 = 134) 218 = 11011010
out = result;
end
endmodule
```

7. controller.v

```
module
controller(clk,clk_inv,clk_phs,rst,addr1_read,addr1_write,mux_en,ram1_en,addr2_read,addr2_write,com
parator_en,counter);
input clk,clk_inv,clk_phs,rst; // clk_ps: phase shifted clk
output [7:0] addr1_read,addr1_write;
output mux_en,ram1_en,comparator_en;
output [5:0] addr2_read,addr2_write;
output [7:0] counter;
reg [7:0] counter;

reg [7:0] addr1_read,addr1_write;
reg [5:0] addr2_read,addr2_write;
reg mux_en,ram1_en,comparator_en,ram1_and,comparator_and;
//========= counter ==========
always @ (posedge clk)
begin
if (rst == 1)
counter <= 8'b1111_1111;
else
        begin
        if (counter < 220)   //result appears at 218
        counter <= counter + 8'b0000_0001;
        else if (counter == 8'b1111_1111)
        counter <= counter + 8'b0000_0001;
        end
end
//======== addr1_read & addr1_write ==========
always @(posedge clk_inv)
begin
if (rst == 0)
begin
if (counter <= 159)
addr1_read = counter;
end
end
```

```
//-------------------------
always @ (posedge clk_inv)
begin
if (rst == 0)
begin
if ( counter >=8'b0000_1010 && counter < 8'b1010_1010)
addr1_write = counter - 8'b0000_1010;
end
end
// ========== mux_en========
always @ (counter)
begin
if (counter == 2)
mux_en = 1;
else if (counter == 162)
mux_en = 0;
end
// ========= ram1_en ========
always @ (posedge clk_inv)
begin
if (counter >= 10 && counter <= 169)
ram1_and = 1;
else
ram1_and = 0;
end
always @ (ram1_and or clk_phs)
begin
ram1_en = ram1_and && clk_phs;
end
//========== addr2_read & addr2_write ==========
reg [7:0] addr2_temp1;
always @ (posedge clk_inv)
begin
if (counter >= 171 && counter <= 207)
addr2_temp1 = counter - 8'b1010_1011; // 171
addr2_read = addr2_temp1[5:0];
end
//---------------------------------
reg [7:0] addr2_temp2;
always @ ( posedge clk_inv)
begin
if( counter >= 8'b1011_0101 && counter <= 8'b1101_1001)
addr2_temp2 = counter - 8'b1011_0101;
addr2_write = addr2_temp2[5:0];

end
//========== comparator_en ============
always @ (posedge clk_inv)
begin
if (counter >= 181 && counter <= 217)
comparator_and = 1;
```

```
else
comparator_and = 0;
end
always @ (comparator_and or clk_phs)
begin
comparator_en = comparator_and && clk_phs;
end
endmodule
```

8. htangent165e004.v

```
module htangent165e004(clk,in,out);

input clk;
input [11:0] in;
output [5:0] out;
reg [5:0] out;

wire r1,r2,r3;

assign r1=in[10]|in[9]|in[8]|in[7]|in[6];
assign r2=!(in[10]|in[9]|in[8]|in[7]|in[6])&in[5];
assign r3=!(in[10]|in[9]|in[8]|in[7]|in[6])&!in[5]&in[4];

always @( posedge clk )

begin
if(r1==1)   out=5'b11111;//64-255
else if(r2==1)
begin
case(in[4:2])
0: out=5'b11001;//32-35
1: out=5'b11010;//36-39
2: out=5'b11100;//40-43
3: out=5'b11100;//44-47
4: out=5'b11101;//48-51
5: out=5'b11110;//52-55
6: out=5'b11110;//56-59
7: out=5'b11111;//60-63
endcase
end
else if(r3==1)
begin
case(in[3:1])
0: out=5'b01111;
1: out=5'b10001;
2: out=5'b10010;
3: out=5'b10011;
4: out=5'b10101;
5: out=5'b10110;
6: out=5'b10111;
```

```
7: out=5'b11000;
endcase
end
else out[4:0]=in[4:0]; //r3 == 1
out[5] = in[11];
end
endmodule
```

9. mult14.v

```
module mult14(in1,in2,out,clk);
  input clk;
  input [13:0] in1,in2;
  reg [12:0] mult1,mult2;
  reg s1,s2;
  reg [25:0] result;
  output [13:0] out;
  wire [13:0] out;

  always @(posedge clk)
  begin
    s1 = in1[13];
    s2 = in2[13];
    mult1 = in1[12:0];
    mult2 = in2[12:0];
    result = mult1*mult2;
  end
  assign out = {{s1^s2},result[19:7]};
endmodule
```

10. mux12.v

```
module mux12(sel,clk,rd,in,out);
input sel,clk,rd;
input [11:0] in;
output [11:0] out;
reg [11:0] out;
always@(posedge clk)
begin
if (rd == 1)
            begin
            if (sel == 1)
            out <= in;
            else
            out <= 12'b000000000000;
            end
end
endmodule
```

11. ram1.v

```
module ram1(in,addr,clk,wr_en,
        out1,out2,out3,out4,out5,out6,out7,out8,out9,out10,
        out11,out12,out13,out14,out15,out16,out17,out18,out19,out20,
        out21,out22,out23,out24,out25,out26,out27,out28,out29,out30,
        out31,out32,out33,out34,out35,out36,out37,out38,out39,out40,
        out41,out42,out43,out44,out45,out46,out47,out48,out49,out50,
        out51,out52,out53,out54,out55,out56,out57,out58,out59,out60,
        out61,out62,out63,out64,out65,out66,out67,out68,out69,out70,
        out71,out72,out73,out74,out75,out76,out77,out78,out79,out80,
        out81,out82,out83,out84,out85,out86,out87,out88,out89,out90,
        out91,out92,out93,out94,out95,out96,out97,out98,out99,out100,
        out101,out102,out103,out104,out105,out106,out107,out108,out109,out110,
        out111,out112,out113,out114,out115,out116,out117,out118,out119,out120,
        out121,out122,out123,out124,out125,out126,out127,out128,out129,out130,
        out131,out132,out133,out134,out135,out136,out137,out138,out139,out140,
        out141,out142,out143,out144,out145,out146,out147,out148,out149,out150,
        out151,out152,out153,out154,out155,out156,out157,out158,out159,out160
);
input [5:0] in;
input [7:0] addr;
input clk,wr_en;
output [13:0]  out1,out2,out3,out4,out5,out6,out7,out8,out9,out10,
        out11,out12,out13,out14,out15,out16,out17,out18,out19,out20,
        out21,out22,out23,out24,out25,out26,out27,out28,out29,out30,
        out31,out32,out33,out34,out35,out36,out37,out38,out39,out40,
        out41,out42,out43,out44,out45,out46,out47,out48,out49,out50,
        out51,out52,out53,out54,out55,out56,out57,out58,out59,out60,
        out61,out62,out63,out64,out65,out66,out67,out68,out69,out70,
        out71,out72,out73,out74,out75,out76,out77,out78,out79,out80,
        out81,out82,out83,out84,out85,out86,out87,out88,out89,out90,
        out91,out92,out93,out94,out95,out96,out97,out98,out99,out100,
        out101,out102,out103,out104,out105,out106,out107,out108,out109,out110,
        out111,out112,out113,out114,out115,out116,out117,out118,out119,out120,
        out121,out122,out123,out124,out125,out126,out127,out128,out129,out130,
        out131,out132,out133,out134,out135,out136,out137,out138,out139,out140,
        out141,out142,out143,out144,out145,out146,out147,out148,out149,out150,
        out151,out152,out153,out154,out155,out156,out157,out158,out159,out160;

reg [13:0] out [0:159];

always @ (posedge clk)
begin
if (wr_en == 1)
begin
if (in[5]==0)
out[addr] = {{7'b0},{in[4:0]},{2'b0}};
else
out[addr] = {{7'b1000000},{in[4:0]},{2'b0}};
end
end
assign out1 = out[0];
```

```
assign out2 = out[1];
assign out3 = out[2];
assign out4 = out[3];
assign out5 = out[4];
assign out6 = out[5];
assign out7 = out[6];
assign out8 = out[7];
assign out9 = out[8];
assign out10 = out[9];
assign out11 = out[10];
assign out12 = out[11];
assign out13 = out[12];
assign out14 = out[13];
assign out15 = out[14];
assign out16 = out[15];
assign out17 = out[16];
assign out18 = out[17];
assign out19 = out[18];
assign out20 = out[19];
assign out21 = out[20];
assign out22 = out[21];
assign out23 = out[22];
assign out24 = out[23];
assign out25 = out[24];
assign out26 = out[25];
assign out27 = out[26];
assign out28 = out[27];
assign out29 = out[28];
assign out30 = out[29];
assign out31 = out[30];
assign out32 = out[31];
assign out33 = out[32];
assign out34 = out[33];
assign out35 = out[34];
assign out36 = out[35];
assign out37 = out[36];
assign out38 = out[37];
assign out39 = out[38];
assign out40 = out[39];
assign out41 = out[40];
assign out42 = out[41];
assign out43 = out[42];
assign out44 = out[43];
assign out45 = out[44];
assign out46 = out[45];
assign out47 = out[46];
assign out48 = out[47];
assign out49 = out[48];
assign out50 = out[49];
assign out51 = out[50];
assign out52 = out[51];
```

```
assign out53 = out[52];
assign out54 = out[53];
assign out55 = out[54];
assign out56 = out[55];
assign out57 = out[56];
assign out58 = out[57];
assign out59 = out[58];
assign out60 = out[59];
assign out61 = out[60];
assign out62 = out[61];
assign out63 = out[62];
assign out64 = out[63];
assign out65 = out[64];
assign out66 = out[65];
assign out67 = out[66];
assign out68 = out[67];
assign out69 = out[68];
assign out70 = out[69];
assign out71 = out[70];
assign out72 = out[71];
assign out73 = out[72];
assign out74 = out[73];
assign out75 = out[74];
assign out76 = out[75];
assign out77 = out[76];
assign out78 = out[77];
assign out79 = out[78];
assign out80 = out[79];
assign out81 = out[80];
assign out82 = out[81];
assign out83 = out[82];
assign out84 = out[83];
assign out85 = out[84];
assign out86 = out[85];
assign out87 = out[86];
assign out88 = out[87];
assign out89 = out[88];
assign out90 = out[89];
assign out91 = out[90];
assign out92 = out[91];
assign out93 = out[92];
assign out94 = out[93];
assign out95 = out[94];
assign out96 = out[95];
assign out97 = out[96];
assign out98 = out[97];
assign out99 = out[98];
assign out100 = out[99];
assign out101 = out[100];
assign out102 = out[101];
assign out103 = out[102];
```

```
assign out104 = out[103];
assign out105 = out[104];
assign out106 = out[105];
assign out107 = out[106];
assign out108 = out[107];
assign out109 = out[108];
assign out110 = out[109];
assign out111 = out[110];
assign out112 = out[111];
assign out113 = out[112];
assign out114 = out[113];
assign out115 = out[114];
assign out116 = out[115];
assign out117 = out[116];
assign out118 = out[117];
assign out119 = out[118];
assign out120 = out[119];
assign out121 = out[120];
assign out122 = out[121];
assign out123 = out[122];
assign out124 = out[123];
assign out125 = out[124];
assign out126 = out[125];
assign out127 = out[126];
assign out128 = out[127];
assign out129 = out[128];
assign out130 = out[129];
assign out131 = out[130];
assign out132 = out[131];
assign out133 = out[132];
assign out134 = out[133];
assign out135 = out[134];
assign out136 = out[135];
assign out137 = out[136];
assign out138 = out[137];
assign out139 = out[138];
assign out140 = out[139];
assign out141 = out[140];
assign out142 = out[141];
assign out143 = out[142];
assign out144 = out[143];
assign out145 = out[144];
assign out146 = out[145];
assign out147 = out[146];
assign out148 = out[147];
assign out149 = out[148];
assign out150 = out[149];
assign out151 = out[150];
assign out152 = out[151];
assign out153 = out[152];
assign out154 = out[153];
```

```
assign out155 = out[154];
assign out156 = out[155];
assign out157 = out[156];
assign out158 = out[157];
assign out159 = out[158];
assign out160 = out[159];
endmodule
```

12. ann.v

```
module ann(clk_source,rst,in,out);
input clk_source,rst;
input [188:0] in;
output [5:0] out;
// --------wire - clk_gen-----------
wire clk_gen_clk,clk_gen_clk_inv,clk_gen_clk_phs,clk_gen_start;
wire or_rst_start;
// module clk_gen
clk_gen
clk_gen1(.clk_source(clk_source),.clk(clk_gen_clk),.clk_inv(clk_gen_clk_inv),.clk_phs(clk_gen_clk_phs
),.start(clk_gen_start));
// --------wire - controller--------
wire [7:0] controller_addr1_read,controller_addr1_write;
wire controller_mux_en,controller_ram1_en,controller_comparator_en;
wire [5:0] controller_addr2_read,controller_addr2_write;
wire [7:0] controller_counter;
assign or_rst_start = clk_gen_start || rst;
// --------module - controller---------
controller
controller1(.clk(clk_gen_clk),.clk_inv(clk_gen_clk_inv),.clk_phs(clk_gen_clk_phs),.rst(or_rst_start),.add
r1_read(controller_addr1_read),.addr1_write(controller_addr1_write),.mux_en(controller_mux_en),.ram1
_en(controller_ram1_en),.addr2_read(controller_addr2_read),.addr2_write(controller_addr2_write),.comp
arator_en(controller_comparator_en),.counter(controller_counter));
// ----------wire - m9k12------------
wire [11:0] m9k12_1_q;
wire [11:0] m9k12_2_q;
wire [11:0] m9k12_3_q;
wire [11:0] m9k12_4_q;
wire [11:0] m9k12_5_q;
wire [11:0] m9k12_6_q;
wire [11:0] m9k12_7_q;
wire [11:0] m9k12_8_q;
wire [11:0] m9k12_9_q;
wire [11:0] m9k12_10_q;
wire [11:0] m9k12_11_q;
wire [11:0] m9k12_12_q;
wire [11:0] m9k12_13_q;
wire [11:0] m9k12_14_q;
wire [11:0] m9k12_15_q;
wire [11:0] m9k12_16_q;
wire [11:0] m9k12_17_q;
```

```
wire [11:0] m9k12_18_q;
wire [11:0] m9k12_19_q;
wire [11:0] m9k12_20_q;
wire [11:0] m9k12_21_q;
wire [11:0] m9k12_22_q;
wire [11:0] m9k12_23_q;
wire [11:0] m9k12_24_q;
wire [11:0] m9k12_25_q;
wire [11:0] m9k12_26_q;
wire [11:0] m9k12_27_q;
wire [11:0] m9k12_28_q;
wire [11:0] m9k12_29_q;
wire [11:0] m9k12_30_q;
wire [11:0] m9k12_31_q;
wire [11:0] m9k12_32_q;
wire [11:0] m9k12_33_q;
wire [11:0] m9k12_34_q;
wire [11:0] m9k12_35_q;
wire [11:0] m9k12_36_q;
wire [11:0] m9k12_37_q;
wire [11:0] m9k12_38_q;
wire [11:0] m9k12_39_q;
wire [11:0] m9k12_40_q;
wire [11:0] m9k12_41_q;
wire [11:0] m9k12_42_q;
wire [11:0] m9k12_43_q;
wire [11:0] m9k12_44_q;
wire [11:0] m9k12_45_q;
wire [11:0] m9k12_46_q;
wire [11:0] m9k12_47_q;
wire [11:0] m9k12_48_q;
wire [11:0] m9k12_49_q;
wire [11:0] m9k12_50_q;
wire [11:0] m9k12_51_q;
wire [11:0] m9k12_52_q;
wire [11:0] m9k12_53_q;
wire [11:0] m9k12_54_q;
wire [11:0] m9k12_55_q;
wire [11:0] m9k12_56_q;
wire [11:0] m9k12_57_q;
wire [11:0] m9k12_58_q;
wire [11:0] m9k12_59_q;
wire [11:0] m9k12_60_q;
wire [11:0] m9k12_61_q;
wire [11:0] m9k12_62_q;
wire [11:0] m9k12_63_q;
wire [11:0] m9k12_64_q;
wire [11:0] m9k12_65_q;
wire [11:0] m9k12_66_q;
wire [11:0] m9k12_67_q;
wire [11:0] m9k12_68_q;
```

```
wire [11:0] m9k12_69_q;
wire [11:0] m9k12_70_q;
wire [11:0] m9k12_71_q;
wire [11:0] m9k12_72_q;
wire [11:0] m9k12_73_q;
wire [11:0] m9k12_74_q;
wire [11:0] m9k12_75_q;
wire [11:0] m9k12_76_q;
wire [11:0] m9k12_77_q;
wire [11:0] m9k12_78_q;
wire [11:0] m9k12_79_q;
wire [11:0] m9k12_80_q;
wire [11:0] m9k12_81_q;
wire [11:0] m9k12_82_q;
wire [11:0] m9k12_83_q;
wire [11:0] m9k12_84_q;
wire [11:0] m9k12_85_q;
wire [11:0] m9k12_86_q;
wire [11:0] m9k12_87_q;
wire [11:0] m9k12_88_q;
wire [11:0] m9k12_89_q;
wire [11:0] m9k12_90_q;
wire [11:0] m9k12_91_q;
wire [11:0] m9k12_92_q;
wire [11:0] m9k12_93_q;
wire [11:0] m9k12_94_q;
wire [11:0] m9k12_95_q;
wire [11:0] m9k12_96_q;
wire [11:0] m9k12_97_q;
wire [11:0] m9k12_98_q;
wire [11:0] m9k12_99_q;
wire [11:0] m9k12_100_q;
wire [11:0] m9k12_101_q;
wire [11:0] m9k12_102_q;
wire [11:0] m9k12_103_q;
wire [11:0] m9k12_104_q;
wire [11:0] m9k12_105_q;
wire [11:0] m9k12_106_q;
wire [11:0] m9k12_107_q;
wire [11:0] m9k12_108_q;
wire [11:0] m9k12_109_q;
wire [11:0] m9k12_110_q;
wire [11:0] m9k12_111_q;
wire [11:0] m9k12_112_q;
wire [11:0] m9k12_113_q;
wire [11:0] m9k12_114_q;
wire [11:0] m9k12_115_q;
wire [11:0] m9k12_116_q;
wire [11:0] m9k12_117_q;
wire [11:0] m9k12_118_q;
wire [11:0] m9k12_119_q;
```

```
wire [11:0] m9k12_120_q;
wire [11:0] m9k12_121_q;
wire [11:0] m9k12_122_q;
wire [11:0] m9k12_123_q;
wire [11:0] m9k12_124_q;
wire [11:0] m9k12_125_q;
wire [11:0] m9k12_126_q;
wire [11:0] m9k12_127_q;
wire [11:0] m9k12_128_q;
wire [11:0] m9k12_129_q;
wire [11:0] m9k12_130_q;
wire [11:0] m9k12_131_q;
wire [11:0] m9k12_132_q;
wire [11:0] m9k12_133_q;
wire [11:0] m9k12_134_q;
wire [11:0] m9k12_135_q;
wire [11:0] m9k12_136_q;
wire [11:0] m9k12_137_q;
wire [11:0] m9k12_138_q;
wire [11:0] m9k12_139_q;
wire [11:0] m9k12_140_q;
wire [11:0] m9k12_141_q;
wire [11:0] m9k12_142_q;
wire [11:0] m9k12_143_q;
wire [11:0] m9k12_144_q;
wire [11:0] m9k12_145_q;
wire [11:0] m9k12_146_q;
wire [11:0] m9k12_147_q;
wire [11:0] m9k12_148_q;
wire [11:0] m9k12_149_q;
wire [11:0] m9k12_150_q;
wire [11:0] m9k12_151_q;
wire [11:0] m9k12_152_q;
wire [11:0] m9k12_153_q;
wire [11:0] m9k12_154_q;
wire [11:0] m9k12_155_q;
wire [11:0] m9k12_156_q;
wire [11:0] m9k12_157_q;
wire [11:0] m9k12_158_q;
wire [11:0] m9k12_159_q;
wire [11:0] m9k12_160_q;
wire [11:0] m9k12_161_q;
wire [11:0] m9k12_162_q;
wire [11:0] m9k12_163_q;
wire [11:0] m9k12_164_q;
wire [11:0] m9k12_165_q;
wire [11:0] m9k12_166_q;
wire [11:0] m9k12_167_q;
wire [11:0] m9k12_168_q;
wire [11:0] m9k12_169_q;
wire [11:0] m9k12_170_q;
```

```
wire [11:0] m9k12_171_q;
wire [11:0] m9k12_172_q;
wire [11:0] m9k12_173_q;
wire [11:0] m9k12_174_q;
wire [11:0] m9k12_175_q;
wire [11:0] m9k12_176_q;
wire [11:0] m9k12_177_q;
wire [11:0] m9k12_178_q;
wire [11:0] m9k12_179_q;
wire [11:0] m9k12_180_q;
wire [11:0] m9k12_181_q;
wire [11:0] m9k12_182_q;
wire [11:0] m9k12_183_q;
wire [11:0] m9k12_184_q;
wire [11:0] m9k12_185_q;
wire [11:0] m9k12_186_q;
wire [11:0] m9k12_187_q;
wire [11:0] m9k12_188_q;
wire [11:0] m9k12_189_q;
wire [11:0] m9k12_190_q;
// --------module m9k12--------
m9k12 m9k12_1(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_1_q));
defparam m9k12_1.init_file = "m9k12_std_1.mif";
m9k12 m9k12_2(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_2_q));
defparam m9k12_2.init_file = "m9k12_std_2.mif";
m9k12 m9k12_3(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_3_q));
defparam m9k12_3.init_file = "m9k12_std_3.mif";
m9k12 m9k12_4(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_4_q));
defparam m9k12_4.init_file = "m9k12_std_4.mif";
m9k12 m9k12_5(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_5_q));
defparam m9k12_5.init_file = "m9k12_std_5.mif";
m9k12 m9k12_6(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_6_q));
defparam m9k12_6.init_file = "m9k12_std_6.mif";
m9k12 m9k12_7(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_7_q));
defparam m9k12_7.init_file = "m9k12_std_7.mif";
m9k12 m9k12_8(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_8_q));
defparam m9k12_8.init_file = "m9k12_std_8.mif";
m9k12 m9k12_9(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_9_q));
defparam m9k12_9.init_file = "m9k12_std_9.mif";
m9k12 m9k12_10(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_10_q));
defparam m9k12_10.init_file = "m9k12_std_10.mif";
m9k12 m9k12_11(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_11_q));
defparam m9k12_11.init_file = "m9k12_std_11.mif";
m9k12 m9k12_12(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_12_q));
defparam m9k12_12.init_file = "m9k12_std_12.mif";
m9k12 m9k12_13(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_13_q));
defparam m9k12_13.init_file = "m9k12_std_13.mif";
m9k12 m9k12_14(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_14_q));
defparam m9k12_14.init_file = "m9k12_std_14.mif";
m9k12 m9k12_15(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_15_q));
defparam m9k12_15.init_file = "m9k12_std_15.mif";
```

```
m9k12 m9k12_16(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_16_q));
defparam m9k12_16.init_file = "m9k12_std_16.mif";
m9k12 m9k12_17(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_17_q));
defparam m9k12_17.init_file = "m9k12_std_17.mif";
m9k12 m9k12_18(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_18_q));
defparam m9k12_18.init_file = "m9k12_std_18.mif";
m9k12 m9k12_19(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_19_q));
defparam m9k12_19.init_file = "m9k12_std_19.mif";
m9k12 m9k12_20(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_20_q));
defparam m9k12_20.init_file = "m9k12_std_20.mif";
m9k12 m9k12_21(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_21_q));
defparam m9k12_21.init_file = "m9k12_std_21.mif";
m9k12 m9k12_22(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_22_q));
defparam m9k12_22.init_file = "m9k12_std_22.mif";
m9k12 m9k12_23(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_23_q));
defparam m9k12_23.init_file = "m9k12_std_23.mif";
m9k12 m9k12_24(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_24_q));
defparam m9k12_24.init_file = "m9k12_std_24.mif";
m9k12 m9k12_25(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_25_q));
defparam m9k12_25.init_file = "m9k12_std_25.mif";
m9k12 m9k12_26(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_26_q));
defparam m9k12_26.init_file = "m9k12_std_26.mif";
m9k12 m9k12_27(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_27_q));
defparam m9k12_27.init_file = "m9k12_std_27.mif";
m9k12 m9k12_28(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_28_q));
defparam m9k12_28.init_file = "m9k12_std_28.mif";
m9k12 m9k12_29(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_29_q));
defparam m9k12_29.init_file = "m9k12_std_29.mif";
m9k12 m9k12_30(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_30_q));
defparam m9k12_30.init_file = "m9k12_std_30.mif";
m9k12 m9k12_31(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_31_q));
defparam m9k12_31.init_file = "m9k12_std_31.mif";
m9k12 m9k12_32(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_32_q));
defparam m9k12_32.init_file = "m9k12_std_32.mif";
m9k12 m9k12_33(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_33_q));
defparam m9k12_33.init_file = "m9k12_std_33.mif";
m9k12 m9k12_34(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_34_q));
defparam m9k12_34.init_file = "m9k12_std_34.mif";
m9k12 m9k12_35(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_35_q));
defparam m9k12_35.init_file = "m9k12_std_35.mif";
m9k12 m9k12_36(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_36_q));
defparam m9k12_36.init_file = "m9k12_std_36.mif";
m9k12 m9k12_37(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_37_q));
defparam m9k12_37.init_file = "m9k12_std_37.mif";
m9k12 m9k12_38(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_38_q));
defparam m9k12_38.init_file = "m9k12_std_38.mif";
m9k12 m9k12_39(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_39_q));
defparam m9k12_39.init_file = "m9k12_std_39.mif";
m9k12 m9k12_40(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_40_q));
defparam m9k12_40.init_file = "m9k12_std_40.mif";
m9k12 m9k12_41(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_41_q));
```

```
defparam m9k12_41.init_file = "m9k12_std_41.mif";
m9k12 m9k12_42(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_42_q));
defparam m9k12_42.init_file = "m9k12_std_42.mif";
m9k12 m9k12_43(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_43_q));
defparam m9k12_43.init_file = "m9k12_std_43.mif";
m9k12 m9k12_44(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_44_q));
defparam m9k12_44.init_file = "m9k12_std_44.mif";
m9k12 m9k12_45(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_45_q));
defparam m9k12_45.init_file = "m9k12_std_45.mif";
m9k12 m9k12_46(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_46_q));
defparam m9k12_46.init_file = "m9k12_std_46.mif";
m9k12 m9k12_47(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_47_q));
defparam m9k12_47.init_file = "m9k12_std_47.mif";
m9k12 m9k12_48(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_48_q));
defparam m9k12_48.init_file = "m9k12_std_48.mif";
m9k12 m9k12_49(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_49_q));
defparam m9k12_49.init_file = "m9k12_std_49.mif";
m9k12 m9k12_50(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_50_q));
defparam m9k12_50.init_file = "m9k12_std_50.mif";
m9k12 m9k12_51(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_51_q));
defparam m9k12_51.init_file = "m9k12_std_51.mif";
m9k12 m9k12_52(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_52_q));
defparam m9k12_52.init_file = "m9k12_std_52.mif";
m9k12 m9k12_53(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_53_q));
defparam m9k12_53.init_file = "m9k12_std_53.mif";
m9k12 m9k12_54(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_54_q));
defparam m9k12_54.init_file = "m9k12_std_54.mif";
m9k12 m9k12_55(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_55_q));
defparam m9k12_55.init_file = "m9k12_std_55.mif";
m9k12 m9k12_56(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_56_q));
defparam m9k12_56.init_file = "m9k12_std_56.mif";

m9k12 m9k12_57(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_57_q));
defparam m9k12_57.init_file = "m9k12_std_57.mif";
m9k12 m9k12_58(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_58_q));
defparam m9k12_58.init_file = "m9k12_std_58.mif";
m9k12 m9k12_59(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_59_q));
defparam m9k12_59.init_file = "m9k12_std_59.mif";

m9k12 m9k12_60(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_60_q));
defparam m9k12_60.init_file = "m9k12_std_60.mif";
m9k12 m9k12_61(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_61_q));
defparam m9k12_61.init_file = "m9k12_std_61.mif";
m9k12 m9k12_62(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_62_q));
defparam m9k12_62.init_file = "m9k12_std_62.mif";
m9k12 m9k12_63(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_63_q));
defparam m9k12_63.init_file = "m9k12_std_63.mif";
m9k12 m9k12_64(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_64_q));
defparam m9k12_64.init_file = "m9k12_std_64.mif";
m9k12 m9k12_65(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_65_q));
defparam m9k12_65.init_file = "m9k12_std_65.mif";
```

```
m9k12 m9k12_66(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_66_q));
defparam m9k12_66.init_file = "m9k12_std_66.mif";
m9k12 m9k12_67(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_67_q));
defparam m9k12_67.init_file = "m9k12_std_67.mif";
m9k12 m9k12_68(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_68_q));
defparam m9k12_68.init_file = "m9k12_std_68.mif";
m9k12 m9k12_69(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_69_q));
defparam m9k12_69.init_file = "m9k12_std_69.mif";

m9k12 m9k12_70(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_70_q));
defparam m9k12_70.init_file = "m9k12_std_70.mif";
m9k12 m9k12_71(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_71_q));
defparam m9k12_71.init_file = "m9k12_std_71.mif";
m9k12 m9k12_72(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_72_q));
defparam m9k12_72.init_file = "m9k12_std_72.mif";
m9k12 m9k12_73(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_73_q));
defparam m9k12_73.init_file = "m9k12_std_73.mif";
m9k12 m9k12_74(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_74_q));
defparam m9k12_74.init_file = "m9k12_std_74.mif";
m9k12 m9k12_75(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_75_q));
defparam m9k12_75.init_file = "m9k12_std_75.mif";
m9k12 m9k12_76(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_76_q));
defparam m9k12_76.init_file = "m9k12_std_76.mif";
m9k12 m9k12_77(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_77_q));
defparam m9k12_77.init_file = "m9k12_std_77.mif";
m9k12 m9k12_78(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_78_q));
defparam m9k12_78.init_file = "m9k12_std_78.mif";
m9k12 m9k12_79(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_79_q));
defparam m9k12_79.init_file = "m9k12_std_79.mif";

m9k12 m9k12_80(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_80_q));
defparam m9k12_80.init_file = "m9k12_std_80.mif";
m9k12 m9k12_81(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_81_q));
defparam m9k12_81.init_file = "m9k12_std_81.mif";
m9k12 m9k12_82(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_82_q));
defparam m9k12_82.init_file = "m9k12_std_82.mif";
m9k12 m9k12_83(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_83_q));
defparam m9k12_83.init_file = "m9k12_std_83.mif";
m9k12 m9k12_84(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_84_q));
defparam m9k12_84.init_file = "m9k12_std_84.mif";
m9k12 m9k12_85(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_85_q));
defparam m9k12_85.init_file = "m9k12_std_85.mif";
m9k12 m9k12_86(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_86_q));
defparam m9k12_86.init_file = "m9k12_std_86.mif";
m9k12 m9k12_87(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_87_q));
defparam m9k12_87.init_file = "m9k12_std_87.mif";
m9k12 m9k12_88(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_88_q));
defparam m9k12_88.init_file = "m9k12_std_88.mif";
m9k12 m9k12_89(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_89_q));
defparam m9k12_89.init_file = "m9k12_std_89.mif";
```

```
m9k12 m9k12_90(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_90_q));
defparam m9k12_90.init_file = "m9k12_std_90.mif";
m9k12 m9k12_91(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_91_q));
defparam m9k12_91.init_file = "m9k12_std_91.mif";
m9k12 m9k12_92(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_92_q));
defparam m9k12_92.init_file = "m9k12_std_92.mif";
m9k12 m9k12_93(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_93_q));
defparam m9k12_93.init_file = "m9k12_std_93.mif";
m9k12 m9k12_94(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_94_q));
defparam m9k12_94.init_file = "m9k12_std_94.mif";
m9k12 m9k12_95(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_95_q));
defparam m9k12_95.init_file = "m9k12_std_95.mif";
m9k12 m9k12_96(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_96_q));
defparam m9k12_96.init_file = "m9k12_std_96.mif";
m9k12 m9k12_97(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_97_q));
defparam m9k12_97.init_file = "m9k12_std_97.mif";
m9k12 m9k12_98(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_98_q));
defparam m9k12_98.init_file = "m9k12_std_98.mif";
m9k12 m9k12_99(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_99_q));
defparam m9k12_99.init_file = "m9k12_std_99.mif";

m9k12 m9k12_100(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_100_q));
defparam m9k12_100.init_file = "m9k12_std_100.mif";
m9k12 m9k12_101(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_101_q));
defparam m9k12_101.init_file = "m9k12_std_101.mif";
m9k12 m9k12_102(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_102_q));
defparam m9k12_102.init_file = "m9k12_std_102.mif";
m9k12 m9k12_103(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_103_q));
defparam m9k12_103.init_file = "m9k12_std_103.mif";
m9k12 m9k12_104(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_104_q));
defparam m9k12_104.init_file = "m9k12_std_104.mif";
m9k12 m9k12_105(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_105_q));
defparam m9k12_105.init_file = "m9k12_std_105.mif";
m9k12 m9k12_106(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_106_q));
defparam m9k12_106.init_file = "m9k12_std_106.mif";
m9k12 m9k12_107(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_107_q));
defparam m9k12_107.init_file = "m9k12_std_107.mif";
m9k12 m9k12_108(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_108_q));
defparam m9k12_108.init_file = "m9k12_std_108.mif";
m9k12 m9k12_109(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_109_q));
defparam m9k12_109.init_file = "m9k12_std_109.mif";

m9k12 m9k12_110(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_110_q));
defparam m9k12_110.init_file = "m9k12_std_110.mif";
m9k12 m9k12_111(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_111_q));
defparam m9k12_111.init_file = "m9k12_std_111.mif";
m9k12 m9k12_112(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_112_q));
defparam m9k12_112.init_file = "m9k12_std_112.mif";
m9k12 m9k12_113(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_113_q));
defparam m9k12_113.init_file = "m9k12_std_113.mif";
m9k12 m9k12_114(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_114_q));
```

```
defparam m9k12_114.init_file = "m9k12_std_114.mif";
m9k12 m9k12_115(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_115_q));
defparam m9k12_115.init_file = "m9k12_std_115.mif";
m9k12 m9k12_116(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_116_q));
defparam m9k12_116.init_file = "m9k12_std_116.mif";
m9k12 m9k12_117(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_117_q));
defparam m9k12_117.init_file = "m9k12_std_117.mif";
m9k12 m9k12_118(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_118_q));
defparam m9k12_118.init_file = "m9k12_std_118.mif";
m9k12 m9k12_119(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_119_q));
defparam m9k12_119.init_file = "m9k12_std_119.mif";

m9k12 m9k12_120(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_120_q));
defparam m9k12_120.init_file = "m9k12_std_120.mif";
m9k12 m9k12_121(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_121_q));
defparam m9k12_121.init_file = "m9k12_std_121.mif";
m9k12 m9k12_122(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_122_q));
defparam m9k12_122.init_file = "m9k12_std_122.mif";
m9k12 m9k12_123(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_123_q));
defparam m9k12_123.init_file = "m9k12_std_123.mif";
m9k12 m9k12_124(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_124_q));
defparam m9k12_124.init_file = "m9k12_std_124.mif";
m9k12 m9k12_125(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_125_q));
defparam m9k12_125.init_file = "m9k12_std_125.mif";
m9k12 m9k12_126(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_126_q));
defparam m9k12_126.init_file = "m9k12_std_126.mif";
m9k12 m9k12_127(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_127_q));
defparam m9k12_127.init_file = "m9k12_std_127.mif";
m9k12 m9k12_128(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_128_q));
defparam m9k12_128.init_file = "m9k12_std_128.mif";
m9k12 m9k12_129(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_129_q));
defparam m9k12_129.init_file = "m9k12_std_129.mif";

m9k12 m9k12_130(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_130_q));
defparam m9k12_130.init_file = "m9k12_std_130.mif";
m9k12 m9k12_131(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_131_q));
defparam m9k12_131.init_file = "m9k12_std_131.mif";
m9k12 m9k12_132(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_132_q));
defparam m9k12_132.init_file = "m9k12_std_132.mif";
m9k12 m9k12_133(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_133_q));
defparam m9k12_133.init_file = "m9k12_std_133.mif";
m9k12 m9k12_134(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_134_q));
defparam m9k12_134.init_file = "m9k12_std_134.mif";
m9k12 m9k12_135(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_135_q));
defparam m9k12_135.init_file = "m9k12_std_135.mif";
m9k12 m9k12_136(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_136_q));
defparam m9k12_136.init_file = "m9k12_std_136.mif";
m9k12 m9k12_137(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_137_q));
defparam m9k12_137.init_file = "m9k12_std_137.mif";
m9k12 m9k12_138(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_138_q));
defparam m9k12_138.init_file = "m9k12_std_138.mif";
```

```
m9k12 m9k12_139(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_139_q));
defparam m9k12_139.init_file = "m9k12_std_139.mif";

m9k12 m9k12_140(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_140_q));
defparam m9k12_140.init_file = "m9k12_std_140.mif";
m9k12 m9k12_141(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_141_q));
defparam m9k12_141.init_file = "m9k12_std_141.mif";
m9k12 m9k12_142(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_142_q));
defparam m9k12_142.init_file = "m9k12_std_142.mif";
m9k12 m9k12_143(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_143_q));
defparam m9k12_143.init_file = "m9k12_std_143.mif";
m9k12 m9k12_144(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_144_q));
defparam m9k12_144.init_file = "m9k12_std_144.mif";
m9k12 m9k12_145(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_145_q));
defparam m9k12_145.init_file = "m9k12_std_145.mif";
m9k12 m9k12_146(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_146_q));
defparam m9k12_146.init_file = "m9k12_std_146.mif";
m9k12 m9k12_147(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_147_q));
defparam m9k12_147.init_file = "m9k12_std_147.mif";
m9k12 m9k12_148(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_148_q));
defparam m9k12_148.init_file = "m9k12_std_148.mif";
m9k12 m9k12_149(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_149_q));
defparam m9k12_149.init_file = "m9k12_std_149.mif";

m9k12 m9k12_150(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_150_q));
defparam m9k12_150.init_file = "m9k12_std_150.mif";
m9k12 m9k12_151(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_151_q));
defparam m9k12_151.init_file = "m9k12_std_151.mif";
m9k12 m9k12_152(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_152_q));
defparam m9k12_152.init_file = "m9k12_std_152.mif";
m9k12 m9k12_153(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_153_q));
defparam m9k12_153.init_file = "m9k12_std_153.mif";
m9k12 m9k12_154(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_154_q));
defparam m9k12_154.init_file = "m9k12_std_154.mif";
m9k12 m9k12_155(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_155_q));
defparam m9k12_155.init_file = "m9k12_std_155.mif";
m9k12 m9k12_156(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_156_q));
defparam m9k12_156.init_file = "m9k12_std_156.mif";
m9k12 m9k12_157(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_157_q));
defparam m9k12_157.init_file = "m9k12_std_157.mif";
m9k12 m9k12_158(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_158_q));
defparam m9k12_158.init_file = "m9k12_std_158.mif";
m9k12 m9k12_159(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_159_q));
defparam m9k12_159.init_file = "m9k12_std_159.mif";

m9k12 m9k12_160(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_160_q));
defparam m9k12_160.init_file = "m9k12_std_160.mif";
m9k12 m9k12_161(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_161_q));
defparam m9k12_161.init_file = "m9k12_std_161.mif";
m9k12 m9k12_162(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_162_q));
defparam m9k12_162.init_file = "m9k12_std_162.mif";
```

```
m9k12 m9k12_163(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_163_q));
defparam m9k12_163.init_file = "m9k12_std_163.mif";
m9k12 m9k12_164(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_164_q));
defparam m9k12_164.init_file = "m9k12_std_164.mif";
m9k12 m9k12_165(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_165_q));
defparam m9k12_165.init_file = "m9k12_std_165.mif";
m9k12 m9k12_166(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_166_q));
defparam m9k12_166.init_file = "m9k12_std_166.mif";
m9k12 m9k12_167(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_167_q));
defparam m9k12_167.init_file = "m9k12_std_167.mif";
m9k12 m9k12_168(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_168_q));
defparam m9k12_168.init_file = "m9k12_std_168.mif";
m9k12 m9k12_169(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_169_q));
defparam m9k12_169.init_file = "m9k12_std_169.mif";

m9k12 m9k12_170(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_170_q));
defparam m9k12_170.init_file = "m9k12_std_170.mif";
m9k12 m9k12_171(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_171_q));
defparam m9k12_171.init_file = "m9k12_std_171.mif";
m9k12 m9k12_172(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_172_q));
defparam m9k12_172.init_file = "m9k12_std_172.mif";
m9k12 m9k12_173(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_173_q));
defparam m9k12_173.init_file = "m9k12_std_173.mif";
m9k12 m9k12_174(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_174_q));
defparam m9k12_174.init_file = "m9k12_std_174.mif";
m9k12 m9k12_175(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_175_q));
defparam m9k12_175.init_file = "m9k12_std_175.mif";
m9k12 m9k12_176(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_176_q));
defparam m9k12_176.init_file = "m9k12_std_176.mif";
m9k12 m9k12_177(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_177_q));
defparam m9k12_177.init_file = "m9k12_std_177.mif";
m9k12 m9k12_178(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_178_q));
defparam m9k12_178.init_file = "m9k12_std_178.mif";
m9k12 m9k12_179(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_179_q));
defparam m9k12_179.init_file = "m9k12_std_179.mif";

m9k12 m9k12_180(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_180_q));
defparam m9k12_180.init_file = "m9k12_std_180.mif";
m9k12 m9k12_181(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_181_q));
defparam m9k12_181.init_file = "m9k12_std_181.mif";
m9k12 m9k12_182(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_182_q));
defparam m9k12_182.init_file = "m9k12_std_182.mif";
m9k12 m9k12_183(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_183_q));
defparam m9k12_183.init_file = "m9k12_std_183.mif";
m9k12 m9k12_184(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_184_q));
defparam m9k12_184.init_file = "m9k12_std_184.mif";
m9k12 m9k12_185(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_185_q));
defparam m9k12_185.init_file = "m9k12_std_185.mif";
m9k12 m9k12_186(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_186_q));
defparam m9k12_186.init_file = "m9k12_std_186.mif";
m9k12 m9k12_187(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_187_q));
```

```
defparam m9k12_187.init_file = "m9k12_std_187.mif";
m9k12 m9k12_188(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_188_q));
defparam m9k12_188.init_file = "m9k12_std_188.mif";
m9k12 m9k12_189(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_189_q));
defparam m9k12_189.init_file = "m9k12_std_189.mif";
//--------bias-----------------
m9k12 m9k12_190(.address(controller_addr1_read),.clock(clk_gen_clk),.q(m9k12_190_q));
defparam m9k12_190.init_file = "m9k12_std_190.mif";

// ---------wire mux12-----------
wire [11:0] mux12_1_out;
wire [11:0] mux12_2_out;
wire [11:0] mux12_3_out;
wire [11:0] mux12_4_out;
wire [11:0] mux12_5_out;
wire [11:0] mux12_6_out;
wire [11:0] mux12_7_out;
wire [11:0] mux12_8_out;
wire [11:0] mux12_9_out;
wire [11:0] mux12_10_out;
wire [11:0] mux12_11_out;
wire [11:0] mux12_12_out;
wire [11:0] mux12_13_out;
wire [11:0] mux12_14_out;
wire [11:0] mux12_15_out;
wire [11:0] mux12_16_out;
wire [11:0] mux12_17_out;
wire [11:0] mux12_18_out;
wire [11:0] mux12_19_out;
wire [11:0] mux12_20_out;
wire [11:0] mux12_21_out;
wire [11:0] mux12_22_out;
wire [11:0] mux12_23_out;
wire [11:0] mux12_24_out;
wire [11:0] mux12_25_out;
wire [11:0] mux12_26_out;
wire [11:0] mux12_27_out;
wire [11:0] mux12_28_out;
wire [11:0] mux12_29_out;
wire [11:0] mux12_30_out;
wire [11:0] mux12_31_out;
wire [11:0] mux12_32_out;
wire [11:0] mux12_33_out;
wire [11:0] mux12_34_out;
wire [11:0] mux12_35_out;
wire [11:0] mux12_36_out;
wire [11:0] mux12_37_out;
wire [11:0] mux12_38_out;
wire [11:0] mux12_39_out;
wire [11:0] mux12_40_out;
wire [11:0] mux12_41_out;
```

```
wire [11:0] mux12_42_out;
wire [11:0] mux12_43_out;
wire [11:0] mux12_44_out;
wire [11:0] mux12_45_out;
wire [11:0] mux12_46_out;
wire [11:0] mux12_47_out;
wire [11:0] mux12_48_out;
wire [11:0] mux12_49_out;
wire [11:0] mux12_50_out;
wire [11:0] mux12_51_out;
wire [11:0] mux12_52_out;
wire [11:0] mux12_53_out;
wire [11:0] mux12_54_out;
wire [11:0] mux12_55_out;
wire [11:0] mux12_56_out;

wire [11:0] mux12_57_out;
wire [11:0] mux12_58_out;
wire [11:0] mux12_59_out;
wire [11:0] mux12_60_out;

wire [11:0] mux12_61_out;
wire [11:0] mux12_62_out;
wire [11:0] mux12_63_out;
wire [11:0] mux12_64_out;
wire [11:0] mux12_65_out;
wire [11:0] mux12_66_out;
wire [11:0] mux12_67_out;
wire [11:0] mux12_68_out;
wire [11:0] mux12_69_out;
wire [11:0] mux12_70_out;

wire [11:0] mux12_71_out;
wire [11:0] mux12_72_out;
wire [11:0] mux12_73_out;
wire [11:0] mux12_74_out;
wire [11:0] mux12_75_out;
wire [11:0] mux12_76_out;
wire [11:0] mux12_77_out;
wire [11:0] mux12_78_out;
wire [11:0] mux12_79_out;
wire [11:0] mux12_80_out;

wire [11:0] mux12_81_out;
wire [11:0] mux12_82_out;
wire [11:0] mux12_83_out;
wire [11:0] mux12_84_out;
wire [11:0] mux12_85_out;
wire [11:0] mux12_86_out;
wire [11:0] mux12_87_out;
wire [11:0] mux12_88_out;
```

```
wire [11:0] mux12_89_out;
wire [11:0] mux12_90_out;

wire [11:0] mux12_91_out;
wire [11:0] mux12_92_out;
wire [11:0] mux12_93_out;
wire [11:0] mux12_94_out;
wire [11:0] mux12_95_out;
wire [11:0] mux12_96_out;
wire [11:0] mux12_97_out;
wire [11:0] mux12_98_out;
wire [11:0] mux12_99_out;
wire [11:0] mux12_100_out;

wire [11:0] mux12_101_out;
wire [11:0] mux12_102_out;
wire [11:0] mux12_103_out;
wire [11:0] mux12_104_out;
wire [11:0] mux12_105_out;
wire [11:0] mux12_106_out;
wire [11:0] mux12_107_out;
wire [11:0] mux12_108_out;
wire [11:0] mux12_109_out;
wire [11:0] mux12_110_out;

wire [11:0] mux12_111_out;
wire [11:0] mux12_112_out;
wire [11:0] mux12_113_out;
wire [11:0] mux12_114_out;
wire [11:0] mux12_115_out;
wire [11:0] mux12_116_out;
wire [11:0] mux12_117_out;
wire [11:0] mux12_118_out;
wire [11:0] mux12_119_out;
wire [11:0] mux12_120_out;

wire [11:0] mux12_121_out;
wire [11:0] mux12_122_out;
wire [11:0] mux12_123_out;
wire [11:0] mux12_124_out;
wire [11:0] mux12_125_out;
wire [11:0] mux12_126_out;
wire [11:0] mux12_127_out;
wire [11:0] mux12_128_out;
wire [11:0] mux12_129_out;
wire [11:0] mux12_130_out;

wire [11:0] mux12_131_out;
wire [11:0] mux12_132_out;
wire [11:0] mux12_133_out;
wire [11:0] mux12_134_out;
```

```
wire [11:0] mux12_135_out;
wire [11:0] mux12_136_out;
wire [11:0] mux12_137_out;
wire [11:0] mux12_138_out;
wire [11:0] mux12_139_out;
wire [11:0] mux12_140_out;

wire [11:0] mux12_141_out;
wire [11:0] mux12_142_out;
wire [11:0] mux12_143_out;
wire [11:0] mux12_144_out;
wire [11:0] mux12_145_out;
wire [11:0] mux12_146_out;
wire [11:0] mux12_147_out;
wire [11:0] mux12_148_out;
wire [11:0] mux12_149_out;
wire [11:0] mux12_150_out;

wire [11:0] mux12_151_out;
wire [11:0] mux12_152_out;
wire [11:0] mux12_153_out;
wire [11:0] mux12_154_out;
wire [11:0] mux12_155_out;
wire [11:0] mux12_156_out;
wire [11:0] mux12_157_out;
wire [11:0] mux12_158_out;
wire [11:0] mux12_159_out;
wire [11:0] mux12_160_out;

wire [11:0] mux12_161_out;
wire [11:0] mux12_162_out;
wire [11:0] mux12_163_out;
wire [11:0] mux12_164_out;
wire [11:0] mux12_165_out;
wire [11:0] mux12_166_out;
wire [11:0] mux12_167_out;
wire [11:0] mux12_168_out;
wire [11:0] mux12_169_out;
wire [11:0] mux12_170_out;

wire [11:0] mux12_171_out;
wire [11:0] mux12_172_out;
wire [11:0] mux12_173_out;
wire [11:0] mux12_174_out;
wire [11:0] mux12_175_out;
wire [11:0] mux12_176_out;
wire [11:0] mux12_177_out;
wire [11:0] mux12_178_out;
wire [11:0] mux12_179_out;
wire [11:0] mux12_180_out;
```

```
wire [11:0] mux12_181_out;
wire [11:0] mux12_182_out;
wire [11:0] mux12_183_out;
wire [11:0] mux12_184_out;
wire [11:0] mux12_185_out;
wire [11:0] mux12_186_out;
wire [11:0] mux12_187_out;
wire [11:0] mux12_188_out;
wire [11:0] mux12_189_out;
wire [11:0] mux12_190_out;

// ---------module mux12------------
mux12
mux12_1(.sel(in[188]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_1_q),.out(mux12_1_out)
);
mux12
mux12_2(.sel(in[187]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_2_q),.out(mux12_2_out)
);
mux12
mux12_3(.sel(in[186]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_3_q),.out(mux12_3_out)
);
mux12
mux12_4(.sel(in[185]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_4_q),.out(mux12_4_out)
);
mux12
mux12_5(.sel(in[184]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_5_q),.out(mux12_5_out)
);
mux12
mux12_6(.sel(in[183]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_6_q),.out(mux12_6_out)
);
mux12
mux12_7(.sel(in[182]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_7_q),.out(mux12_7_out)
);
mux12
mux12_8(.sel(in[181]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_8_q),.out(mux12_8_out)
);
mux12
mux12_9(.sel(in[180]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_9_q),.out(mux12_9_out)
);
mux12
mux12_10(.sel(in[179]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_10_q),.out(mux12_10_
out));
mux12
mux12_11(.sel(in[178]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_11_q),.out(mux12_11_
out));
mux12
mux12_12(.sel(in[177]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_12_q),.out(mux12_12_
out));
mux12
mux12_13(.sel(in[176]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_13_q),.out(mux12_13_
out));
```

```
mux12
mux12_14(.sel(in[175]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_14_q),.out(mux12_14_
out));
mux12
mux12_15(.sel(in[174]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_15_q),.out(mux12_15_
out));
mux12
mux12_16(.sel(in[173]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_16_q),.out(mux12_16_
out));
mux12
mux12_17(.sel(in[172]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_17_q),.out(mux12_17_
out));
mux12
mux12_18(.sel(in[171]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_18_q),.out(mux12_18_
out));
mux12
mux12_19(.sel(in[170]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_19_q),.out(mux12_19_
out));
mux12
mux12_20(.sel(in[169]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_20_q),.out(mux12_20_
out));
mux12
mux12_21(.sel(in[168]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_21_q),.out(mux12_21_
out));
mux12
mux12_22(.sel(in[167]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_22_q),.out(mux12_22_
out));
mux12
mux12_23(.sel(in[166]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_23_q),.out(mux12_23_
out));
mux12
mux12_24(.sel(in[165]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_24_q),.out(mux12_24_
out));
mux12
mux12_25(.sel(in[164]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_25_q),.out(mux12_25_
out));
mux12
mux12_26(.sel(in[163]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_26_q),.out(mux12_26_
out));
mux12
mux12_27(.sel(in[162]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_27_q),.out(mux12_27_
out));
mux12
mux12_28(.sel(in[161]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_28_q),.out(mux12_28_
out));
mux12
mux12_29(.sel(in[160]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_29_q),.out(mux12_29_
out));
mux12
mux12_30(.sel(in[159]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_30_q),.out(mux12_30_
out));
```

```
mux12
mux12_31(.sel(in[158]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_31_q),.out(mux12_31_
out));
mux12
mux12_32(.sel(in[157]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_32_q),.out(mux12_32_
out));
mux12
mux12_33(.sel(in[156]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_33_q),.out(mux12_33_
out));
mux12
mux12_34(.sel(in[155]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_34_q),.out(mux12_34_
out));
mux12
mux12_35(.sel(in[154]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_35_q),.out(mux12_35_
out));
mux12
mux12_36(.sel(in[153]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_36_q),.out(mux12_36_
out));
mux12
mux12_37(.sel(in[152]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_37_q),.out(mux12_37_
out));
mux12
mux12_38(.sel(in[151]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_38_q),.out(mux12_38_
out));
mux12
mux12_39(.sel(in[150]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_39_q),.out(mux12_39_
out));
mux12
mux12_40(.sel(in[149]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_40_q),.out(mux12_40_
out));
mux12
mux12_41(.sel(in[148]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_41_q),.out(mux12_41_
out));
mux12
mux12_42(.sel(in[147]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_42_q),.out(mux12_42_
out));
mux12
mux12_43(.sel(in[146]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_43_q),.out(mux12_43_
out));
mux12
mux12_44(.sel(in[145]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_44_q),.out(mux12_44_
out));
mux12
mux12_45(.sel(in[144]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_45_q),.out(mux12_45_
out));
mux12
mux12_46(.sel(in[143]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_46_q),.out(mux12_46_
out));
mux12
mux12_47(.sel(in[142]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_47_q),.out(mux12_47_
out));
```

```
mux12
mux12_48(.sel(in[141]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_48_q),.out(mux12_48_
out));
mux12
mux12_49(.sel(in[140]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_49_q),.out(mux12_49_
out));
mux12
mux12_50(.sel(in[139]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_50_q),.out(mux12_50_
out));
mux12
mux12_51(.sel(in[138]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_51_q),.out(mux12_51_
out));
mux12
mux12_52(.sel(in[137]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_52_q),.out(mux12_52_
out));
mux12
mux12_53(.sel(in[136]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_53_q),.out(mux12_53_
out));
mux12
mux12_54(.sel(in[135]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_54_q),.out(mux12_54_
out));
mux12
mux12_55(.sel(in[134]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_55_q),.out(mux12_55_
out));

mux12
mux12_56(.sel(in[133]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_56_q),.out(mux12_56_
out));
mux12
mux12_57(.sel(in[132]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_57_q),.out(mux12_57_
out));
mux12
mux12_58(.sel(in[131]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_58_q),.out(mux12_58_
out));
mux12
mux12_59(.sel(in[130]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_59_q),.out(mux12_59_
out));
mux12
mux12_60(.sel(in[129]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_60_q),.out(mux12_60_
out));

mux12
mux12_61(.sel(in[128]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_61_q),.out(mux12_61_
out));
mux12
mux12_62(.sel(in[127]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_62_q),.out(mux12_62_
out));
mux12
mux12_63(.sel(in[126]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_63_q),.out(mux12_63_
out));
```

```
mux12
mux12_64(.sel(in[125]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_64_q),.out(mux12_64_
out));
mux12
mux12_65(.sel(in[124]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_65_q),.out(mux12_65_
out));
mux12
mux12_66(.sel(in[123]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_66_q),.out(mux12_66_
out));
mux12
mux12_67(.sel(in[122]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_67_q),.out(mux12_67_
out));
mux12
mux12_68(.sel(in[121]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_68_q),.out(mux12_68_
out));
mux12
mux12_69(.sel(in[120]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_69_q),.out(mux12_69_
out));
mux12
mux12_70(.sel(in[119]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_70_q),.out(mux12_70_
out));

mux12
mux12_71(.sel(in[118]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_71_q),.out(mux12_71_
out));
mux12
mux12_72(.sel(in[117]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_72_q),.out(mux12_72_
out));
mux12
mux12_73(.sel(in[116]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_73_q),.out(mux12_73_
out));
mux12
mux12_74(.sel(in[115]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_74_q),.out(mux12_74_
out));
mux12
mux12_75(.sel(in[114]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_75_q),.out(mux12_75_
out));
mux12
mux12_76(.sel(in[113]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_76_q),.out(mux12_76_
out));
mux12
mux12_77(.sel(in[112]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_77_q),.out(mux12_77_
out));
mux12
mux12_78(.sel(in[111]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_78_q),.out(mux12_78_
out));
mux12
mux12_79(.sel(in[110]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_79_q),.out(mux12_79_
out));
```

```
mux12
mux12_80(.sel(in[109]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_80_q),.out(mux12_80_
out));

mux12
mux12_81(.sel(in[108]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_81_q),.out(mux12_81_
out));
mux12
mux12_82(.sel(in[107]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_82_q),.out(mux12_82_
out));
mux12
mux12_83(.sel(in[106]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_83_q),.out(mux12_83_
out));
mux12
mux12_84(.sel(in[105]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_84_q),.out(mux12_84_
out));
mux12
mux12_85(.sel(in[104]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_85_q),.out(mux12_85_
out));
mux12
mux12_86(.sel(in[103]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_86_q),.out(mux12_86_
out));
mux12
mux12_87(.sel(in[102]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_87_q),.out(mux12_87_
out));
mux12
mux12_88(.sel(in[101]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_88_q),.out(mux12_88_
out));
mux12
mux12_89(.sel(in[100]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_89_q),.out(mux12_89_
out));
mux12
mux12_90(.sel(in[99]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_90_q),.out(mux12_90_o
ut));

mux12
mux12_91(.sel(in[98]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_91_q),.out(mux12_91_o
ut));
mux12
mux12_92(.sel(in[97]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_92_q),.out(mux12_92_o
ut));
mux12
mux12_93(.sel(in[96]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_93_q),.out(mux12_93_o
ut));
mux12
mux12_94(.sel(in[95]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_94_q),.out(mux12_94_o
ut));
mux12
mux12_95(.sel(in[94]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_95_q),.out(mux12_95_o
ut));
```

```
mux12
mux12_96(.sel(in[93]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_96_q),.out(mux12_96_o
ut));
mux12
mux12_97(.sel(in[92]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_97_q),.out(mux12_97_o
ut));
mux12
mux12_98(.sel(in[91]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_98_q),.out(mux12_98_o
ut));
mux12
mux12_99(.sel(in[90]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_99_q),.out(mux12_99_o
ut));
mux12
mux12_100(.sel(in[89]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_100_q),.out(mux12_10
0_out));

mux12
mux12_101(.sel(in[88]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_101_q),.out(mux12_10
1_out));
mux12
mux12_102(.sel(in[87]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_102_q),.out(mux12_10
2_out));
mux12
mux12_103(.sel(in[86]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_103_q),.out(mux12_10
3_out));
mux12
mux12_104(.sel(in[85]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_104_q),.out(mux12_10
4_out));
mux12
mux12_105(.sel(in[84]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_105_q),.out(mux12_10
5_out));
mux12
mux12_106(.sel(in[83]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_106_q),.out(mux12_10
6_out));
mux12
mux12_107(.sel(in[82]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_107_q),.out(mux12_10
7_out));
mux12
mux12_108(.sel(in[81]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_108_q),.out(mux12_10
8_out));
mux12
mux12_109(.sel(in[80]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_109_q),.out(mux12_10
9_out));
mux12
mux12_110(.sel(in[79]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_110_q),.out(mux12_11
0_out));

mux12
mux12_111(.sel(in[78]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_111_q),.out(mux12_11
1_out));
```

```
mux12
mux12_112(.sel(in[77]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_112_q),.out(mux12_11
2_out));
mux12
mux12_113(.sel(in[76]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_113_q),.out(mux12_11
3_out));
mux12
mux12_114(.sel(in[75]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_114_q),.out(mux12_11
4_out));
mux12
mux12_115(.sel(in[74]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_115_q),.out(mux12_11
5_out));
mux12
mux12_116(.sel(in[73]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_116_q),.out(mux12_11
6_out));
mux12
mux12_117(.sel(in[72]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_117_q),.out(mux12_11
7_out));
mux12
mux12_118(.sel(in[71]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_118_q),.out(mux12_11
8_out));
mux12
mux12_119(.sel(in[70]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_119_q),.out(mux12_11
9_out));
mux12
mux12_120(.sel(in[69]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_120_q),.out(mux12_12
0_out));

mux12
mux12_121(.sel(in[68]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_121_q),.out(mux12_12
1_out));
mux12
mux12_122(.sel(in[67]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_122_q),.out(mux12_12
2_out));
mux12
mux12_123(.sel(in[66]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_123_q),.out(mux12_12
3_out));
mux12
mux12_124(.sel(in[65]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_124_q),.out(mux12_12
4_out));
mux12
mux12_125(.sel(in[64]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_125_q),.out(mux12_12
5_out));
mux12
mux12_126(.sel(in[63]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_126_q),.out(mux12_12
6_out));
mux12
mux12_127(.sel(in[62]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_127_q),.out(mux12_12
7_out));
```

```
mux12
mux12_128(.sel(in[61]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_128_q),.out(mux12_12
8_out));
mux12
mux12_129(.sel(in[60]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_129_q),.out(mux12_12
9_out));
mux12
mux12_130(.sel(in[59]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_130_q),.out(mux12_13
0_out));

mux12
mux12_131(.sel(in[58]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_131_q),.out(mux12_13
1_out));
mux12
mux12_132(.sel(in[57]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_132_q),.out(mux12_13
2_out));
mux12
mux12_133(.sel(in[56]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_133_q),.out(mux12_13
3_out));
mux12
mux12_134(.sel(in[55]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_134_q),.out(mux12_13
4_out));
mux12
mux12_135(.sel(in[54]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_135_q),.out(mux12_13
5_out));
mux12
mux12_136(.sel(in[53]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_136_q),.out(mux12_13
6_out));
mux12
mux12_137(.sel(in[52]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_137_q),.out(mux12_13
7_out));
mux12
mux12_138(.sel(in[51]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_138_q),.out(mux12_13
8_out));
mux12
mux12_139(.sel(in[50]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_139_q),.out(mux12_13
9_out));
mux12
mux12_140(.sel(in[49]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_140_q),.out(mux12_14
0_out));

mux12
mux12_141(.sel(in[48]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_141_q),.out(mux12_14
1_out));
mux12
mux12_142(.sel(in[47]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_142_q),.out(mux12_14
2_out));
mux12
mux12_143(.sel(in[46]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_143_q),.out(mux12_14
3_out));
```

```
mux12
mux12_144(.sel(in[45]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_144_q),.out(mux12_14
4_out));
mux12
mux12_145(.sel(in[44]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_145_q),.out(mux12_14
5_out));
mux12
mux12_146(.sel(in[43]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_146_q),.out(mux12_14
6_out));
mux12
mux12_147(.sel(in[42]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_147_q),.out(mux12_14
7_out));
mux12
mux12_148(.sel(in[41]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_148_q),.out(mux12_14
8_out));
mux12
mux12_149(.sel(in[40]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_149_q),.out(mux12_14
9_out));
mux12
mux12_150(.sel(in[39]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_150_q),.out(mux12_15
0_out));

mux12
mux12_151(.sel(in[38]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_151_q),.out(mux12_15
1_out));
mux12
mux12_152(.sel(in[37]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_152_q),.out(mux12_15
2_out));
mux12
mux12_153(.sel(in[36]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_153_q),.out(mux12_15
3_out));
mux12
mux12_154(.sel(in[35]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_154_q),.out(mux12_15
4_out));
mux12
mux12_155(.sel(in[34]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_155_q),.out(mux12_15
5_out));
mux12
mux12_156(.sel(in[33]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_156_q),.out(mux12_15
6_out));
mux12
mux12_157(.sel(in[32]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_157_q),.out(mux12_15
7_out));
mux12
mux12_158(.sel(in[31]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_158_q),.out(mux12_15
8_out));
mux12
mux12_159(.sel(in[30]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_159_q),.out(mux12_15
9_out));
```

```
mux12
mux12_160(.sel(in[29]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_160_q),.out(mux12_16
0_out));

mux12
mux12_161(.sel(in[28]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_161_q),.out(mux12_16
1_out));
mux12
mux12_162(.sel(in[27]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_162_q),.out(mux12_16
2_out));
mux12
mux12_163(.sel(in[26]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_163_q),.out(mux12_16
3_out));
mux12
mux12_164(.sel(in[25]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_164_q),.out(mux12_16
4_out));
mux12
mux12_165(.sel(in[24]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_165_q),.out(mux12_16
5_out));
mux12
mux12_166(.sel(in[23]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_166_q),.out(mux12_16
6_out));
mux12
mux12_167(.sel(in[22]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_167_q),.out(mux12_16
7_out));
mux12
mux12_168(.sel(in[21]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_168_q),.out(mux12_16
8_out));
mux12
mux12_169(.sel(in[20]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_169_q),.out(mux12_16
9_out));
mux12
mux12_170(.sel(in[19]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_170_q),.out(mux12_17
0_out));

mux12
mux12_171(.sel(in[18]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_171_q),.out(mux12_17
1_out));
mux12
mux12_172(.sel(in[17]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_172_q),.out(mux12_17
2_out));
mux12
mux12_173(.sel(in[16]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_173_q),.out(mux12_17
3_out));
mux12
mux12_174(.sel(in[15]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_174_q),.out(mux12_17
4_out));
mux12
mux12_175(.sel(in[14]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_175_q),.out(mux12_17
5_out));
```

```
mux12
mux12_176(.sel(in[13]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_176_q),.out(mux12_17
6_out));
mux12
mux12_177(.sel(in[12]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_177_q),.out(mux12_17
7_out));
mux12
mux12_178(.sel(in[11]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_178_q),.out(mux12_17
8_out));
mux12
mux12_179(.sel(in[10]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_179_q),.out(mux12_17
9_out));
mux12
mux12_180(.sel(in[9]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_180_q),.out(mux12_180
_out));

mux12
mux12_181(.sel(in[8]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_181_q),.out(mux12_181
_out));
mux12
mux12_182(.sel(in[7]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_182_q),.out(mux12_182
_out));
mux12
mux12_183(.sel(in[6]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_183_q),.out(mux12_183
_out));
mux12
mux12_184(.sel(in[5]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_184_q),.out(mux12_184
_out));
mux12
mux12_185(.sel(in[4]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_185_q),.out(mux12_185
_out));
mux12
mux12_186(.sel(in[3]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_186_q),.out(mux12_186
_out));
mux12
mux12_187(.sel(in[2]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_187_q),.out(mux12_187
_out));
mux12
mux12_188(.sel(in[1]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_188_q),.out(mux12_188
_out));
mux12
mux12_189(.sel(in[0]),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_189_q),.out(mux12_189
_out));

mux12
mux12_190(.sel(1'b1),.clk(clk_gen_clk_inv),.rd(controller_mux_en),.in(m9k12_190_q),.out(mux12_190
_out));
// ---------wire - adder190--------
wire [11:0] adder190_out;
// --------module - adder190-------
```

adder190
adder190_1(.in1(mux12_1_out),.in2(mux12_2_out),.in3(mux12_3_out),.in4(mux12_4_out),.in5(mux12_5_out),.in6(mux12_6_out),.in7(mux12_7_out),.in8(mux12_8_out),.in9(mux12_9_out),.in10(mux12_10_out),.in11(mux12_11_out),.in12(mux12_12_out),.in13(mux12_13_out),.in14(mux12_14_out),.in15(mux12_15_out),.in16(mux12_16_out),.in17(mux12_17_out),.in18(mux12_18_out),.in19(mux12_19_out),.in20(mux12_20_out),.in21(mux12_21_out),.in22(mux12_22_out),.in23(mux12_23_out),.in24(mux12_24_out),.in25(mux12_25_out),.in26(mux12_26_out),.in27(mux12_27_out),.in28(mux12_28_out),.in29(mux12_29_out),.in30(mux12_30_out),.in31(mux12_31_out),.in32(mux12_32_out),.in33(mux12_33_out),.in34(mux12_34_out),.in35(mux12_35_out),.in36(mux12_36_out),.in37(mux12_37_out),.in38(mux12_38_out),.in39(mux12_39_out),.in40(mux12_40_out),.in41(mux12_41_out),.in42(mux12_42_out),.in43(mux12_43_out),.in44(mux12_44_out),.in45(mux12_45_out),.in46(mux12_46_out),.in47(mux12_47_out),.in48(mux12_48_out),.in49(mux12_49_out),.in50(mux12_50_out),.in51(mux12_51_out),.in52(mux12_52_out),.in53(mux12_53_out),.in54(mux12_54_out),.in55(mux12_55_out),.in56(mux12_56_out),.in57(mux12_57_out),.in58(mux12_58_out),.in59(mux12_59_out),.in60(mux12_60_out),.in61(mux12_61_out),.in62(mux12_62_out),.in63(mux12_63_out),.in64(mux12_64_out),.in65(mux12_65_out),.in66(mux12_66_out),.in67(mux12_67_out),.in68(mux12_68_out),.in69(mux12_69_out),.in70(mux12_70_out),.in71(mux12_71_out),.in72(mux12_72_out),.in73(mux12_73_out),.in74(mux12_74_out),.in75(mux12_75_out),.in76(mux12_76_out),.in77(mux12_77_out),.in78(mux12_78_out),.in79(mux12_79_out),.in80(mux12_80_out),.in81(mux12_81_out),.in82(mux12_82_out),.in83(mux12_83_out),.in84(mux12_84_out),.in85(mux12_85_out),.in86(mux12_86_out),.in87(mux12_87_out),.in88(mux12_88_out),.in89(mux12_89_out),.in90(mux12_90_out),.in91(mux12_91_out),.in92(mux12_92_out),.in93(mux12_93_out),.in94(mux12_94_out),.in95(mux12_95_out),.in96(mux12_96_out),.in97(mux12_97_out),.in98(mux12_98_out),.in99(mux12_99_out),.in100(mux12_100_out),.in101(mux12_101_out),.in102(mux12_102_out),.in103(mux12_103_out),.in104(mux12_104_out),.in105(mux12_105_out),.in106(mux12_106_out),.in107(mux12_107_out),.in108(mux12_108_out),.in109(mux12_109_out),.in110(mux12_110_out),.in111(mux12_111_out),.in112(mux12_112_out),.in113(mux12_113_out),.in114(mux12_114_out),.in115(mux12_115_out),.in116(mux12_116_out),.in117(mux12_117_out),.in118(mux12_118_out),.in119(mux12_119_out),.in120(mux12_120_out),.in121(mux12_121_out),.in122(mux12_122_out),.in123(mux12_123_out),.in124(mux12_124_out),.in125(mux12_125_out),.in126(mux12_126_out),.in127(mux12_127_out),.in128(mux12_128_out),.in129(mux12_129_out),.in130(mux12_130_out),.in131(mux12_131_out),.in132(mux12_132_out),.in133(mux12_133_out),.in134(mux12_134_out),.in135(mux12_135_out),.in136(mux12_136_out),.in137(mux12_137_out),.in138(mux12_138_out),.in139(mux12_139_out),.in140(mux12_140_out),.in141(mux12_141_out),.in142(mux12_142_out),.in143(mux12_143_out),.in144(mux12_144_out),.in145(mux12_145_out),.in146(mux12_146_out),.in147(mux12_147_out),.in148(mux12_148_out),.in149(mux12_149_out),.in150(mux12_150_out),.in151(mux12_151_out),.in152(mux12_152_out),.in153(mux12_153_out),.in154(mux12_154_out),.in155(mux12_155_out),.in156(mux12_156_out),.in157(mux12_157_out),.in158(mux12_158_out),.in159(mux12_159_out),.in160(mux12_160_out),.in161(mux12_161_out),.in162(mux12_162_out),.in163(mux12_163_out),.in164(mux12_164_out),.in165(mux12_165_out),.in166(mux12_166_out),.in167(mux12_167_out),.in168(mux12_168_out),.in169(mux12_169_out),.in170(mux12_170_out),.in171(mux12_171_out),.in172(mux12_172_out),.in173(mux12_173_out),.in174(mux12_174_out),.in175(mux12_175_out),.in176(mux12_176_out),.in177(mux12_177_out),.in178(mux12_178_out),.in179(mux12_179_out),.in180(mux12_180_out),.in181(mux12_181_out),.in182(mux12_182_out),.in183(mux12_183_out),.in184(mux12_184_out),.in185(mux12_185_out),.in186(mux12_186_out),.in187(mux12_187_out),.in188(mux12_188_out),.in189(mux12_189_out),.in190(mux12_190_out),
.clk(clk_gen_clk),.out(adder190_out));
// -------wire - com2mag165---------
wire [11:0] com2mag165_out;
// ---------module - com2mag165--------
com2mag165 com2mag165_1(.in(adder190_out),.out(com2mag165_out));
// --------wire - htangent165e004-------
wire [5:0] htangent165e004_1_out;

```
// --------module - htangent165e004------
htangent165e004
htangent165e004_1(.clk(clk_gen_clk_inv),.in(com2mag165_out),.out(htangent165e004_1_out));
// ---------wire - ram1-----------
wire [13:0] ram1_out1;
wire [13:0] ram1_out2;
wire [13:0] ram1_out3;
wire [13:0] ram1_out4;
wire [13:0] ram1_out5;
wire [13:0] ram1_out6;
wire [13:0] ram1_out7;
wire [13:0] ram1_out8;
wire [13:0] ram1_out9;
wire [13:0] ram1_out10;
wire [13:0] ram1_out11;
wire [13:0] ram1_out12;
wire [13:0] ram1_out13;
wire [13:0] ram1_out14;
wire [13:0] ram1_out15;
wire [13:0] ram1_out16;
wire [13:0] ram1_out17;
wire [13:0] ram1_out18;
wire [13:0] ram1_out19;
wire [13:0] ram1_out20;
wire [13:0] ram1_out21;
wire [13:0] ram1_out22;
wire [13:0] ram1_out23;
wire [13:0] ram1_out24;
wire [13:0] ram1_out25;
wire [13:0] ram1_out26;
wire [13:0] ram1_out27;
wire [13:0] ram1_out28;
wire [13:0] ram1_out29;
wire [13:0] ram1_out30;
wire [13:0] ram1_out31;
wire [13:0] ram1_out32;
wire [13:0] ram1_out33;
wire [13:0] ram1_out34;
wire [13:0] ram1_out35;
wire [13:0] ram1_out36;
wire [13:0] ram1_out37;
wire [13:0] ram1_out38;
wire [13:0] ram1_out39;
wire [13:0] ram1_out40;
wire [13:0] ram1_out41;
wire [13:0] ram1_out42;
wire [13:0] ram1_out43;
wire [13:0] ram1_out44;
wire [13:0] ram1_out45;
wire [13:0] ram1_out46;
wire [13:0] ram1_out47;
```

```
wire [13:0] ram1_out48;
wire [13:0] ram1_out49;
wire [13:0] ram1_out50;
wire [13:0] ram1_out51;
wire [13:0] ram1_out52;
wire [13:0] ram1_out53;
wire [13:0] ram1_out54;
wire [13:0] ram1_out55;
wire [13:0] ram1_out56;
wire [13:0] ram1_out57;
wire [13:0] ram1_out58;
wire [13:0] ram1_out59;
wire [13:0] ram1_out60;
wire [13:0] ram1_out61;
wire [13:0] ram1_out62;
wire [13:0] ram1_out63;
wire [13:0] ram1_out64;
wire [13:0] ram1_out65;
wire [13:0] ram1_out66;
wire [13:0] ram1_out67;
wire [13:0] ram1_out68;
wire [13:0] ram1_out69;
wire [13:0] ram1_out70;
wire [13:0] ram1_out71;
wire [13:0] ram1_out72;
wire [13:0] ram1_out73;
wire [13:0] ram1_out74;
wire [13:0] ram1_out75;
wire [13:0] ram1_out76;
wire [13:0] ram1_out77;
wire [13:0] ram1_out78;
wire [13:0] ram1_out79;
wire [13:0] ram1_out80;

wire [13:0] ram1_out81;
wire [13:0] ram1_out82;
wire [13:0] ram1_out83;
wire [13:0] ram1_out84;
wire [13:0] ram1_out85;
wire [13:0] ram1_out86;
wire [13:0] ram1_out87;
wire [13:0] ram1_out88;
wire [13:0] ram1_out89;
wire [13:0] ram1_out90;

wire [13:0] ram1_out91;
wire [13:0] ram1_out92;
wire [13:0] ram1_out93;
wire [13:0] ram1_out94;
wire [13:0] ram1_out95;
wire [13:0] ram1_out96;
```

```
wire [13:0] ram1_out97;
wire [13:0] ram1_out98;
wire [13:0] ram1_out99;
wire [13:0] ram1_out100;

wire [13:0] ram1_out101;
wire [13:0] ram1_out102;
wire [13:0] ram1_out103;
wire [13:0] ram1_out104;
wire [13:0] ram1_out105;
wire [13:0] ram1_out106;
wire [13:0] ram1_out107;
wire [13:0] ram1_out108;
wire [13:0] ram1_out109;
wire [13:0] ram1_out110;
wire [13:0] ram1_out111;
wire [13:0] ram1_out112;
wire [13:0] ram1_out113;
wire [13:0] ram1_out114;
wire [13:0] ram1_out115;
wire [13:0] ram1_out116;
wire [13:0] ram1_out117;
wire [13:0] ram1_out118;
wire [13:0] ram1_out119;
wire [13:0] ram1_out120;

wire [13:0] ram1_out121;
wire [13:0] ram1_out122;
wire [13:0] ram1_out123;
wire [13:0] ram1_out124;
wire [13:0] ram1_out125;
wire [13:0] ram1_out126;
wire [13:0] ram1_out127;
wire [13:0] ram1_out128;
wire [13:0] ram1_out129;
wire [13:0] ram1_out130;

wire [13:0] ram1_out131;
wire [13:0] ram1_out132;
wire [13:0] ram1_out133;
wire [13:0] ram1_out134;
wire [13:0] ram1_out135;
wire [13:0] ram1_out136;
wire [13:0] ram1_out137;
wire [13:0] ram1_out138;
wire [13:0] ram1_out139;
wire [13:0] ram1_out140;

wire [13:0] ram1_out141;
wire [13:0] ram1_out142;
wire [13:0] ram1_out143;
```

```
wire [13:0] ram1_out144;
wire [13:0] ram1_out145;
wire [13:0] ram1_out146;
wire [13:0] ram1_out147;
wire [13:0] ram1_out148;
wire [13:0] ram1_out149;
wire [13:0] ram1_out150;

wire [13:0] ram1_out151;
wire [13:0] ram1_out152;
wire [13:0] ram1_out153;
wire [13:0] ram1_out154;
wire [13:0] ram1_out155;
wire [13:0] ram1_out156;
wire [13:0] ram1_out157;
wire [13:0] ram1_out158;
wire [13:0] ram1_out159;
wire [13:0] ram1_out160;
// --------module - ram1----------
ram1
ram1_1(.in(htangent165e004_1_out),.addr(controller_addr1_write),.clk(clk_gen_clk),.wr_en(controller_r
am1_en),.out1(ram1_out1),.out2(ram1_out2),.out3(ram1_out3),.out4(ram1_out4),.out5(ram1_out5),.out6(
ram1_out6),.out7(ram1_out7),.out8(ram1_out8),.out9(ram1_out9),.out10(ram1_out10),.out11(ram1_out1
1),.out12(ram1_out12),.out13(ram1_out13),.out14(ram1_out14),.out15(ram1_out15),.out16(ram1_out16),
.out17(ram1_out17),.out18(ram1_out18),.out19(ram1_out19),.out20(ram1_out20),.out21(ram1_out21),.o
ut22(ram1_out22),.out23(ram1_out23),.out24(ram1_out24),.out25(ram1_out25),.out26(ram1_out26),.out
27(ram1_out27),.out28(ram1_out28),.out29(ram1_out29),.out30(ram1_out30),.out31(ram1_out31),.out32
(ram1_out32),.out33(ram1_out33),.out34(ram1_out34),.out35(ram1_out35),.out36(ram1_out36),.out37(ra
m1_out37),.out38(ram1_out38),.out39(ram1_out39),.out40(ram1_out40),.out41(ram1_out41),.out42(ram
1_out42),.out43(ram1_out43),.out44(ram1_out44),.out45(ram1_out45),.out46(ram1_out46),.out47(ram1_
out47),.out48(ram1_out48),.out49(ram1_out49),.out50(ram1_out50),.out51(ram1_out51),.out52(ram1_ou
t52),.out53(ram1_out53),.out54(ram1_out54),.out55(ram1_out55),.out56(ram1_out56),.out57(ram1_out5
7),.out58(ram1_out58),.out59(ram1_out59),.out60(ram1_out60),.out61(ram1_out61),.out62(ram1_out62),
.out63(ram1_out63),.out64(ram1_out64),.out65(ram1_out65),.out66(ram1_out66),.out67(ram1_out67),.o
ut68(ram1_out68),.out69(ram1_out69),.out70(ram1_out70),.out71(ram1_out71),.out72(ram1_out72),.out
73(ram1_out73),.out74(ram1_out74),.out75(ram1_out75),.out76(ram1_out76),.out77(ram1_out77),.out78
(ram1_out78),.out79(ram1_out79),.out80(ram1_out80),.out81(ram1_out81),.out82(ram1_out82),.out83(ra
m1_out83),.out84(ram1_out84),.out85(ram1_out85),.out86(ram1_out86),.out87(ram1_out87),.out88(ram
1_out88),.out89(ram1_out89),.out90(ram1_out90),.out91(ram1_out91),.out92(ram1_out92),.out93(ram1_
out93),.out94(ram1_out94),.out95(ram1_out95),.out96(ram1_out96),.out97(ram1_out97),.out98(ram1_ou
t98),.out99(ram1_out99),.out100(ram1_out100),.out101(ram1_out101),.out102(ram1_out102),.out103(ra
m1_out103),.out104(ram1_out104),.out105(ram1_out105),.out106(ram1_out106),.out107(ram1_out107),.
out108(ram1_out108),.out109(ram1_out109),.out110(ram1_out110),.out111(ram1_out111),.out112(ram1
_out112),.out113(ram1_out113),.out114(ram1_out114),.out115(ram1_out115),.out116(ram1_out116),.out
117(ram1_out117),.out118(ram1_out118),.out119(ram1_out119),.out120(ram1_out120),.out121(ram1_ou
t121),.out122(ram1_out122),.out123(ram1_out123),.out124(ram1_out124),.out125(ram1_out125),.out126
(ram1_out126),.out127(ram1_out127),.out128(ram1_out128),.out129(ram1_out129),.out130(ram1_out13
0),.out131(ram1_out131),.out132(ram1_out132),.out133(ram1_out133),.out134(ram1_out134),.out135(ra
m1_out135),.out136(ram1_out136),.out137(ram1_out137),.out138(ram1_out138),.out139(ram1_out139),.
out140(ram1_out140),.out141(ram1_out141),.out142(ram1_out142),.out143(ram1_out143),.out144(ram1
_out144),.out145(ram1_out145),.out146(ram1_out146),.out147(ram1_out147),.out148(ram1_out148),.out
```

```
149(ram1_out149),.out150(ram1_out150),.out151(ram1_out151),.out152(ram1_out152),.out153(ram1_ou
t153),.out154(ram1_out154),.out155(ram1_out155),.out156(ram1_out156),.out157(ram1_out157),.out158
(ram1_out158),.out159(ram1_out159),.out160(ram1_out160));
// --------wire - m9k14--------
wire [13:0] m9k14_1_q;
wire [13:0] m9k14_2_q;
wire [13:0] m9k14_3_q;
wire [13:0] m9k14_4_q;
wire [13:0] m9k14_5_q;
wire [13:0] m9k14_6_q;
wire [13:0] m9k14_7_q;
wire [13:0] m9k14_8_q;
wire [13:0] m9k14_9_q;
wire [13:0] m9k14_10_q;
wire [13:0] m9k14_11_q;
wire [13:0] m9k14_12_q;
wire [13:0] m9k14_13_q;
wire [13:0] m9k14_14_q;
wire [13:0] m9k14_15_q;
wire [13:0] m9k14_16_q;
wire [13:0] m9k14_17_q;
wire [13:0] m9k14_18_q;
wire [13:0] m9k14_19_q;
wire [13:0] m9k14_20_q;
wire [13:0] m9k14_21_q;
wire [13:0] m9k14_22_q;
wire [13:0] m9k14_23_q;
wire [13:0] m9k14_24_q;
wire [13:0] m9k14_25_q;
wire [13:0] m9k14_26_q;
wire [13:0] m9k14_27_q;
wire [13:0] m9k14_28_q;
wire [13:0] m9k14_29_q;
wire [13:0] m9k14_30_q;
wire [13:0] m9k14_31_q;
wire [13:0] m9k14_32_q;
wire [13:0] m9k14_33_q;
wire [13:0] m9k14_34_q;
wire [13:0] m9k14_35_q;
wire [13:0] m9k14_36_q;
wire [13:0] m9k14_37_q;
wire [13:0] m9k14_38_q;
wire [13:0] m9k14_39_q;
wire [13:0] m9k14_40_q;
wire [13:0] m9k14_41_q;
wire [13:0] m9k14_42_q;
wire [13:0] m9k14_43_q;
wire [13:0] m9k14_44_q;
wire [13:0] m9k14_45_q;
wire [13:0] m9k14_46_q;
wire [13:0] m9k14_47_q;
```

```
wire [13:0] m9k14_48_q;
wire [13:0] m9k14_49_q;
wire [13:0] m9k14_50_q;
wire [13:0] m9k14_51_q;
wire [13:0] m9k14_52_q;
wire [13:0] m9k14_53_q;
wire [13:0] m9k14_54_q;
wire [13:0] m9k14_55_q;
wire [13:0] m9k14_56_q;
wire [13:0] m9k14_57_q;
wire [13:0] m9k14_58_q;
wire [13:0] m9k14_59_q;
wire [13:0] m9k14_60_q;
wire [13:0] m9k14_61_q;
wire [13:0] m9k14_62_q;
wire [13:0] m9k14_63_q;
wire [13:0] m9k14_64_q;
wire [13:0] m9k14_65_q;
wire [13:0] m9k14_66_q;
wire [13:0] m9k14_67_q;
wire [13:0] m9k14_68_q;
wire [13:0] m9k14_69_q;
wire [13:0] m9k14_70_q;
wire [13:0] m9k14_71_q;
wire [13:0] m9k14_72_q;
wire [13:0] m9k14_73_q;
wire [13:0] m9k14_74_q;
wire [13:0] m9k14_75_q;
wire [13:0] m9k14_76_q;
wire [13:0] m9k14_77_q;
wire [13:0] m9k14_78_q;
wire [13:0] m9k14_79_q;
wire [13:0] m9k14_80_q;
wire [13:0] m9k14_81_q;
wire [13:0] m9k14_82_q;
wire [13:0] m9k14_83_q;
wire [13:0] m9k14_84_q;
wire [13:0] m9k14_85_q;
wire [13:0] m9k14_86_q;
wire [13:0] m9k14_87_q;
wire [13:0] m9k14_88_q;
wire [13:0] m9k14_89_q;
wire [13:0] m9k14_90_q;
wire [13:0] m9k14_91_q;
wire [13:0] m9k14_92_q;
wire [13:0] m9k14_93_q;
wire [13:0] m9k14_94_q;
wire [13:0] m9k14_95_q;
wire [13:0] m9k14_96_q;
wire [13:0] m9k14_97_q;
wire [13:0] m9k14_98_q;
```

```
wire [13:0] m9k14_99_q;
wire [13:0] m9k14_100_q;
wire [13:0] m9k14_101_q;
wire [13:0] m9k14_102_q;
wire [13:0] m9k14_103_q;
wire [13:0] m9k14_104_q;
wire [13:0] m9k14_105_q;
wire [13:0] m9k14_106_q;
wire [13:0] m9k14_107_q;
wire [13:0] m9k14_108_q;
wire [13:0] m9k14_109_q;
wire [13:0] m9k14_110_q;
wire [13:0] m9k14_111_q;
wire [13:0] m9k14_112_q;
wire [13:0] m9k14_113_q;
wire [13:0] m9k14_114_q;
wire [13:0] m9k14_115_q;
wire [13:0] m9k14_116_q;
wire [13:0] m9k14_117_q;
wire [13:0] m9k14_118_q;
wire [13:0] m9k14_119_q;
wire [13:0] m9k14_120_q;
wire [13:0] m9k14_121_q;
wire [13:0] m9k14_122_q;
wire [13:0] m9k14_123_q;
wire [13:0] m9k14_124_q;
wire [13:0] m9k14_125_q;
wire [13:0] m9k14_126_q;
wire [13:0] m9k14_127_q;
wire [13:0] m9k14_128_q;
wire [13:0] m9k14_129_q;
wire [13:0] m9k14_130_q;
wire [13:0] m9k14_131_q;
wire [13:0] m9k14_132_q;
wire [13:0] m9k14_133_q;
wire [13:0] m9k14_134_q;
wire [13:0] m9k14_135_q;
wire [13:0] m9k14_136_q;
wire [13:0] m9k14_137_q;
wire [13:0] m9k14_138_q;
wire [13:0] m9k14_139_q;
wire [13:0] m9k14_140_q;
wire [13:0] m9k14_141_q;
wire [13:0] m9k14_142_q;
wire [13:0] m9k14_143_q;
wire [13:0] m9k14_144_q;
wire [13:0] m9k14_145_q;
wire [13:0] m9k14_146_q;
wire [13:0] m9k14_147_q;
wire [13:0] m9k14_148_q;
wire [13:0] m9k14_149_q;
```

```
wire [13:0] m9k14_150_q;
wire [13:0] m9k14_151_q;
wire [13:0] m9k14_152_q;
wire [13:0] m9k14_153_q;
wire [13:0] m9k14_154_q;
wire [13:0] m9k14_155_q;
wire [13:0] m9k14_156_q;
wire [13:0] m9k14_157_q;
wire [13:0] m9k14_158_q;
wire [13:0] m9k14_159_q;
wire [13:0] m9k14_160_q;

wire [13:0] m9k14_161_q;

// ---------module - m9k14----------
m9k14 m9k14_1(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_1_q));
defparam m9k14_1.init_file = "m9k14_std_1.mif";
m9k14 m9k14_2(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_2_q));
defparam m9k14_2.init_file = "m9k14_std_2.mif";
m9k14 m9k14_3(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_3_q));
defparam m9k14_3.init_file = "m9k14_std_3.mif";
m9k14 m9k14_4(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_4_q));
defparam m9k14_4.init_file = "m9k14_std_4.mif";
m9k14 m9k14_5(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_5_q));
defparam m9k14_5.init_file = "m9k14_std_5.mif";
m9k14 m9k14_6(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_6_q));
defparam m9k14_6.init_file = "m9k14_std_6.mif";
m9k14 m9k14_7(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_7_q));
defparam m9k14_7.init_file = "m9k14_std_7.mif";
m9k14 m9k14_8(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_8_q));
defparam m9k14_8.init_file = "m9k14_std_8.mif";
m9k14 m9k14_9(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_9_q));
defparam m9k14_9.init_file = "m9k14_std_9.mif";
m9k14 m9k14_10(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_10_q));
defparam m9k14_10.init_file = "m9k14_std_10.mif";
m9k14 m9k14_11(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_11_q));
defparam m9k14_11.init_file = "m9k14_std_11.mif";
m9k14 m9k14_12(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_12_q));
defparam m9k14_12.init_file = "m9k14_std_12.mif";
m9k14 m9k14_13(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_13_q));
defparam m9k14_13.init_file = "m9k14_std_13.mif";
m9k14 m9k14_14(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_14_q));
defparam m9k14_14.init_file = "m9k14_std_14.mif";
m9k14 m9k14_15(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_15_q));
defparam m9k14_15.init_file = "m9k14_std_15.mif";
m9k14 m9k14_16(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_16_q));
defparam m9k14_16.init_file = "m9k14_std_16.mif";
m9k14 m9k14_17(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_17_q));
defparam m9k14_17.init_file = "m9k14_std_17.mif";
m9k14 m9k14_18(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_18_q));
defparam m9k14_18.init_file = "m9k14_std_18.mif";
```

```
m9k14 m9k14_19(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_19_q));
defparam m9k14_19.init_file = "m9k14_std_19.mif";
m9k14 m9k14_20(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_20_q));
defparam m9k14_20.init_file = "m9k14_std_20.mif";
m9k14 m9k14_21(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_21_q));
defparam m9k14_21.init_file = "m9k14_std_21.mif";
m9k14 m9k14_22(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_22_q));
defparam m9k14_22.init_file = "m9k14_std_22.mif";
m9k14 m9k14_23(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_23_q));
defparam m9k14_23.init_file = "m9k14_std_23.mif";
m9k14 m9k14_24(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_24_q));
defparam m9k14_24.init_file = "m9k14_std_24.mif";
m9k14 m9k14_25(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_25_q));
defparam m9k14_25.init_file = "m9k14_std_25.mif";
m9k14 m9k14_26(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_26_q));
defparam m9k14_26.init_file = "m9k14_std_26.mif";
m9k14 m9k14_27(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_27_q));
defparam m9k14_27.init_file = "m9k14_std_27.mif";
m9k14 m9k14_28(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_28_q));
defparam m9k14_28.init_file = "m9k14_std_28.mif";
m9k14 m9k14_29(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_29_q));
defparam m9k14_29.init_file = "m9k14_std_29.mif";
m9k14 m9k14_30(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_30_q));
defparam m9k14_30.init_file = "m9k14_std_30.mif";
m9k14 m9k14_31(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_31_q));
defparam m9k14_31.init_file = "m9k14_std_31.mif";
m9k14 m9k14_32(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_32_q));
defparam m9k14_32.init_file = "m9k14_std_32.mif";
m9k14 m9k14_33(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_33_q));
defparam m9k14_33.init_file = "m9k14_std_33.mif";
m9k14 m9k14_34(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_34_q));
defparam m9k14_34.init_file = "m9k14_std_34.mif";
m9k14 m9k14_35(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_35_q));
defparam m9k14_35.init_file = "m9k14_std_35.mif";
m9k14 m9k14_36(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_36_q));
defparam m9k14_36.init_file = "m9k14_std_36.mif";
m9k14 m9k14_37(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_37_q));
defparam m9k14_37.init_file = "m9k14_std_37.mif";
m9k14 m9k14_38(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_38_q));
defparam m9k14_38.init_file = "m9k14_std_38.mif";
m9k14 m9k14_39(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_39_q));
defparam m9k14_39.init_file = "m9k14_std_39.mif";
m9k14 m9k14_40(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_40_q));
defparam m9k14_40.init_file = "m9k14_std_40.mif";
m9k14 m9k14_41(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_41_q));
defparam m9k14_41.init_file = "m9k14_std_41.mif";
m9k14 m9k14_42(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_42_q));
defparam m9k14_42.init_file = "m9k14_std_42.mif";
m9k14 m9k14_43(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_43_q));
defparam m9k14_43.init_file = "m9k14_std_43.mif";
m9k14 m9k14_44(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_44_q));
```

```
defparam m9k14_44.init_file = "m9k14_std_44.mif";
m9k14 m9k14_45(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_45_q));
defparam m9k14_45.init_file = "m9k14_std_45.mif";
m9k14 m9k14_46(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_46_q));
defparam m9k14_46.init_file = "m9k14_std_46.mif";
m9k14 m9k14_47(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_47_q));
defparam m9k14_47.init_file = "m9k14_std_47.mif";
m9k14 m9k14_48(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_48_q));
defparam m9k14_48.init_file = "m9k14_std_48.mif";
m9k14 m9k14_49(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_49_q));
defparam m9k14_49.init_file = "m9k14_std_49.mif";
m9k14 m9k14_50(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_50_q));
defparam m9k14_50.init_file = "m9k14_std_50.mif";
m9k14 m9k14_51(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_51_q));
defparam m9k14_51.init_file = "m9k14_std_51.mif";
m9k14 m9k14_52(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_52_q));
defparam m9k14_52.init_file = "m9k14_std_52.mif";
m9k14 m9k14_53(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_53_q));
defparam m9k14_53.init_file = "m9k14_std_53.mif";
m9k14 m9k14_54(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_54_q));
defparam m9k14_54.init_file = "m9k14_std_54.mif";
m9k14 m9k14_55(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_55_q));
defparam m9k14_55.init_file = "m9k14_std_55.mif";
m9k14 m9k14_56(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_56_q));
defparam m9k14_56.init_file = "m9k14_std_56.mif";
m9k14 m9k14_57(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_57_q));
defparam m9k14_57.init_file = "m9k14_std_57.mif";
m9k14 m9k14_58(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_58_q));
defparam m9k14_58.init_file = "m9k14_std_58.mif";
m9k14 m9k14_59(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_59_q));
defparam m9k14_59.init_file = "m9k14_std_59.mif";
m9k14 m9k14_60(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_60_q));
defparam m9k14_60.init_file = "m9k14_std_60.mif";
m9k14 m9k14_61(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_61_q));
defparam m9k14_61.init_file = "m9k14_std_61.mif";
m9k14 m9k14_62(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_62_q));
defparam m9k14_62.init_file = "m9k14_std_62.mif";
m9k14 m9k14_63(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_63_q));
defparam m9k14_63.init_file = "m9k14_std_63.mif";
m9k14 m9k14_64(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_64_q));
defparam m9k14_64.init_file = "m9k14_std_64.mif";
m9k14 m9k14_65(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_65_q));
defparam m9k14_65.init_file = "m9k14_std_65.mif";
m9k14 m9k14_66(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_66_q));
defparam m9k14_66.init_file = "m9k14_std_66.mif";
m9k14 m9k14_67(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_67_q));
defparam m9k14_67.init_file = "m9k14_std_67.mif";
m9k14 m9k14_68(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_68_q));
defparam m9k14_68.init_file = "m9k14_std_68.mif";
m9k14 m9k14_69(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_69_q));
defparam m9k14_69.init_file = "m9k14_std_69.mif";
```

```
m9k14 m9k14_70(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_70_q));
defparam m9k14_70.init_file = "m9k14_std_70.mif";
m9k14 m9k14_71(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_71_q));
defparam m9k14_71.init_file = "m9k14_std_71.mif";
m9k14 m9k14_72(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_72_q));
defparam m9k14_72.init_file = "m9k14_std_72.mif";
m9k14 m9k14_73(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_73_q));
defparam m9k14_73.init_file = "m9k14_std_73.mif";
m9k14 m9k14_74(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_74_q));
defparam m9k14_74.init_file = "m9k14_std_74.mif";
m9k14 m9k14_75(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_75_q));
defparam m9k14_75.init_file = "m9k14_std_75.mif";
m9k14 m9k14_76(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_76_q));
defparam m9k14_76.init_file = "m9k14_std_76.mif";
m9k14 m9k14_77(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_77_q));
defparam m9k14_77.init_file = "m9k14_std_77.mif";
m9k14 m9k14_78(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_78_q));
defparam m9k14_78.init_file = "m9k14_std_78.mif";
m9k14 m9k14_79(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_79_q));
defparam m9k14_79.init_file = "m9k14_std_79.mif";
m9k14 m9k14_80(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_80_q));
defparam m9k14_80.init_file = "m9k14_std_80.mif";
m9k14 m9k14_81(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_81_q));
defparam m9k14_81.init_file = "m9k14_std_81.mif";
m9k14 m9k14_82(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_82_q));
defparam m9k14_82.init_file = "m9k14_std_82.mif";
m9k14 m9k14_83(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_83_q));
defparam m9k14_83.init_file = "m9k14_std_83.mif";
m9k14 m9k14_84(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_84_q));
defparam m9k14_84.init_file = "m9k14_std_84.mif";
m9k14 m9k14_85(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_85_q));
defparam m9k14_85.init_file = "m9k14_std_85.mif";
m9k14 m9k14_86(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_86_q));
defparam m9k14_86.init_file = "m9k14_std_86.mif";
m9k14 m9k14_87(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_87_q));
defparam m9k14_87.init_file = "m9k14_std_87.mif";
m9k14 m9k14_88(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_88_q));
defparam m9k14_88.init_file = "m9k14_std_88.mif";
m9k14 m9k14_89(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_89_q));
defparam m9k14_89.init_file = "m9k14_std_89.mif";
m9k14 m9k14_90(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_90_q));
defparam m9k14_90.init_file = "m9k14_std_90.mif";
m9k14 m9k14_91(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_91_q));
defparam m9k14_91.init_file = "m9k14_std_91.mif";
m9k14 m9k14_92(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_92_q));
defparam m9k14_92.init_file = "m9k14_std_92.mif";
m9k14 m9k14_93(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_93_q));
defparam m9k14_93.init_file = "m9k14_std_93.mif";
m9k14 m9k14_94(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_94_q));
defparam m9k14_94.init_file = "m9k14_std_94.mif";
m9k14 m9k14_95(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_95_q));
```

```
defparam m9k14_95.init_file = "m9k14_std_95.mif";
m9k14 m9k14_96(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_96_q));
defparam m9k14_96.init_file = "m9k14_std_96.mif";
m9k14 m9k14_97(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_97_q));
defparam m9k14_97.init_file = "m9k14_std_97.mif";
m9k14 m9k14_98(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_98_q));
defparam m9k14_98.init_file = "m9k14_std_98.mif";
m9k14 m9k14_99(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_99_q));
defparam m9k14_99.init_file = "m9k14_std_99.mif";
m9k14 m9k14_100(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_100_q));
defparam m9k14_100.init_file = "m9k14_std_100.mif";

m9k14 m9k14_101(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_101_q));
defparam m9k14_101.init_file = "m9k14_std_101.mif";
m9k14 m9k14_102(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_102_q));
defparam m9k14_102.init_file = "m9k14_std_102.mif";
m9k14 m9k14_103(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_103_q));
defparam m9k14_103.init_file = "m9k14_std_103.mif";
m9k14 m9k14_104(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_104_q));
defparam m9k14_104.init_file = "m9k14_std_104.mif";
m9k14 m9k14_105(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_105_q));
defparam m9k14_105.init_file = "m9k14_std_105.mif";
m9k14 m9k14_106(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_106_q));
defparam m9k14_106.init_file = "m9k14_std_106.mif";
m9k14 m9k14_107(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_107_q));
defparam m9k14_107.init_file = "m9k14_std_107.mif";
m9k14 m9k14_108(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_108_q));
defparam m9k14_108.init_file = "m9k14_std_108.mif";
m9k14 m9k14_109(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_109_q));
defparam m9k14_109.init_file = "m9k14_std_109.mif";
m9k14 m9k14_110(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_110_q));
defparam m9k14_110.init_file = "m9k14_std_110.mif";
m9k14 m9k14_111(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_111_q));
defparam m9k14_111.init_file = "m9k14_std_111.mif";
m9k14 m9k14_112(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_112_q));
defparam m9k14_112.init_file = "m9k14_std_112.mif";
m9k14 m9k14_113(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_113_q));
defparam m9k14_113.init_file = "m9k14_std_113.mif";
m9k14 m9k14_114(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_114_q));
defparam m9k14_114.init_file = "m9k14_std_114.mif";
m9k14 m9k14_115(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_115_q));
defparam m9k14_115.init_file = "m9k14_std_115.mif";
m9k14 m9k14_116(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_116_q));
defparam m9k14_116.init_file = "m9k14_std_116.mif";
m9k14 m9k14_117(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_117_q));
defparam m9k14_117.init_file = "m9k14_std_117.mif";
m9k14 m9k14_118(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_118_q));
defparam m9k14_118.init_file = "m9k14_std_118.mif";
m9k14 m9k14_119(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_119_q));
defparam m9k14_119.init_file = "m9k14_std_119.mif";
m9k14 m9k14_120(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_120_q));
```

```
defparam m9k14_120.init_file = "m9k14_std_120.mif";
m9k14 m9k14_121(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_121_q));
defparam m9k14_121.init_file = "m9k14_std_121.mif";
m9k14 m9k14_122(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_122_q));
defparam m9k14_122.init_file = "m9k14_std_122.mif";
m9k14 m9k14_123(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_123_q));
defparam m9k14_123.init_file = "m9k14_std_123.mif";
m9k14 m9k14_124(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_124_q));
defparam m9k14_124.init_file = "m9k14_std_124.mif";
m9k14 m9k14_125(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_125_q));
defparam m9k14_125.init_file = "m9k14_std_125.mif";
m9k14 m9k14_126(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_126_q));
defparam m9k14_126.init_file = "m9k14_std_126.mif";
m9k14 m9k14_127(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_127_q));
defparam m9k14_127.init_file = "m9k14_std_127.mif";
m9k14 m9k14_128(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_128_q));
defparam m9k14_128.init_file = "m9k14_std_128.mif";
m9k14 m9k14_129(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_129_q));
defparam m9k14_129.init_file = "m9k14_std_129.mif";
m9k14 m9k14_130(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_130_q));
defparam m9k14_130.init_file = "m9k14_std_130.mif";
m9k14 m9k14_131(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_131_q));
defparam m9k14_131.init_file = "m9k14_std_131.mif";
m9k14 m9k14_132(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_132_q));
defparam m9k14_132.init_file = "m9k14_std_132.mif";
m9k14 m9k14_133(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_133_q));
defparam m9k14_133.init_file = "m9k14_std_133.mif";
m9k14 m9k14_134(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_134_q));
defparam m9k14_134.init_file = "m9k14_std_134.mif";
m9k14 m9k14_135(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_135_q));
defparam m9k14_135.init_file = "m9k14_std_135.mif";
m9k14 m9k14_136(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_136_q));
defparam m9k14_136.init_file = "m9k14_std_136.mif";
m9k14 m9k14_137(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_137_q));
defparam m9k14_137.init_file = "m9k14_std_137.mif";
m9k14 m9k14_138(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_138_q));
defparam m9k14_138.init_file = "m9k14_std_138.mif";
m9k14 m9k14_139(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_139_q));
defparam m9k14_139.init_file = "m9k14_std_139.mif";
m9k14 m9k14_140(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_140_q));
defparam m9k14_140.init_file = "m9k14_std_140.mif";
m9k14 m9k14_141(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_141_q));
defparam m9k14_141.init_file = "m9k14_std_141.mif";
m9k14 m9k14_142(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_142_q));
defparam m9k14_142.init_file = "m9k14_std_142.mif";
m9k14 m9k14_143(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_143_q));
defparam m9k14_143.init_file = "m9k14_std_143.mif";
m9k14 m9k14_144(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_144_q));
defparam m9k14_144.init_file = "m9k14_std_144.mif";
m9k14 m9k14_145(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_145_q));
defparam m9k14_145.init_file = "m9k14_std_145.mif";
```

```
m9k14 m9k14_146(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_146_q));
defparam m9k14_146.init_file = "m9k14_std_146.mif";
m9k14 m9k14_147(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_147_q));
defparam m9k14_147.init_file = "m9k14_std_147.mif";
m9k14 m9k14_148(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_148_q));
defparam m9k14_148.init_file = "m9k14_std_148.mif";
m9k14 m9k14_149(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_149_q));
defparam m9k14_149.init_file = "m9k14_std_149.mif";
m9k14 m9k14_150(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_150_q));
defparam m9k14_150.init_file = "m9k14_std_150.mif";
m9k14 m9k14_151(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_151_q));
defparam m9k14_151.init_file = "m9k14_std_151.mif";
m9k14 m9k14_152(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_152_q));
defparam m9k14_152.init_file = "m9k14_std_152.mif";
m9k14 m9k14_153(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_153_q));
defparam m9k14_153.init_file = "m9k14_std_153.mif";
m9k14 m9k14_154(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_154_q));
defparam m9k14_154.init_file = "m9k14_std_154.mif";
m9k14 m9k14_155(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_155_q));
defparam m9k14_155.init_file = "m9k14_std_155.mif";
m9k14 m9k14_156(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_156_q));
defparam m9k14_156.init_file = "m9k14_std_156.mif";
m9k14 m9k14_157(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_157_q));
defparam m9k14_157.init_file = "m9k14_std_157.mif";
m9k14 m9k14_158(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_158_q));
defparam m9k14_158.init_file = "m9k14_std_158.mif";
m9k14 m9k14_159(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_159_q));
defparam m9k14_159.init_file = "m9k14_std_159.mif";
m9k14 m9k14_160(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_160_q));
defparam m9k14_160.init_file = "m9k14_std_160.mif";

m9k14 m9k14_161(.address(controller_addr2_read),.clock(clk_gen_clk),.q(m9k14_161_q));
defparam m9k14_161.init_file = "m9k14_std_161.mif";
// -----------wire - mult14-----------
wire [13:0] mult14_1_out;
wire [13:0] mult14_2_out;
wire [13:0] mult14_3_out;
wire [13:0] mult14_4_out;
wire [13:0] mult14_5_out;
wire [13:0] mult14_6_out;
wire [13:0] mult14_7_out;
wire [13:0] mult14_8_out;
wire [13:0] mult14_9_out;
wire [13:0] mult14_10_out;
wire [13:0] mult14_11_out;
wire [13:0] mult14_12_out;
wire [13:0] mult14_13_out;
wire [13:0] mult14_14_out;
wire [13:0] mult14_15_out;
wire [13:0] mult14_16_out;
wire [13:0] mult14_17_out;
```

```
wire [13:0] mult14_18_out;
wire [13:0] mult14_19_out;
wire [13:0] mult14_20_out;
wire [13:0] mult14_21_out;
wire [13:0] mult14_22_out;
wire [13:0] mult14_23_out;
wire [13:0] mult14_24_out;
wire [13:0] mult14_25_out;
wire [13:0] mult14_26_out;
wire [13:0] mult14_27_out;
wire [13:0] mult14_28_out;
wire [13:0] mult14_29_out;
wire [13:0] mult14_30_out;
wire [13:0] mult14_31_out;
wire [13:0] mult14_32_out;
wire [13:0] mult14_33_out;
wire [13:0] mult14_34_out;
wire [13:0] mult14_35_out;
wire [13:0] mult14_36_out;
wire [13:0] mult14_37_out;
wire [13:0] mult14_38_out;
wire [13:0] mult14_39_out;
wire [13:0] mult14_40_out;
wire [13:0] mult14_41_out;
wire [13:0] mult14_42_out;
wire [13:0] mult14_43_out;
wire [13:0] mult14_44_out;
wire [13:0] mult14_45_out;
wire [13:0] mult14_46_out;
wire [13:0] mult14_47_out;
wire [13:0] mult14_48_out;
wire [13:0] mult14_49_out;
wire [13:0] mult14_50_out;
wire [13:0] mult14_51_out;
wire [13:0] mult14_52_out;
wire [13:0] mult14_53_out;
wire [13:0] mult14_54_out;
wire [13:0] mult14_55_out;
wire [13:0] mult14_56_out;
wire [13:0] mult14_57_out;
wire [13:0] mult14_58_out;
wire [13:0] mult14_59_out;
wire [13:0] mult14_60_out;
wire [13:0] mult14_61_out;
wire [13:0] mult14_62_out;
wire [13:0] mult14_63_out;
wire [13:0] mult14_64_out;
wire [13:0] mult14_65_out;
wire [13:0] mult14_66_out;
wire [13:0] mult14_67_out;
wire [13:0] mult14_68_out;
```

```
wire [13:0] mult14_69_out;
wire [13:0] mult14_70_out;
wire [13:0] mult14_71_out;
wire [13:0] mult14_72_out;
wire [13:0] mult14_73_out;
wire [13:0] mult14_74_out;
wire [13:0] mult14_75_out;
wire [13:0] mult14_76_out;
wire [13:0] mult14_77_out;
wire [13:0] mult14_78_out;
wire [13:0] mult14_79_out;
wire [13:0] mult14_80_out;
wire [13:0] mult14_81_out;
wire [13:0] mult14_82_out;
wire [13:0] mult14_83_out;
wire [13:0] mult14_84_out;
wire [13:0] mult14_85_out;
wire [13:0] mult14_86_out;
wire [13:0] mult14_87_out;
wire [13:0] mult14_88_out;
wire [13:0] mult14_89_out;
wire [13:0] mult14_90_out;
wire [13:0] mult14_91_out;
wire [13:0] mult14_92_out;
wire [13:0] mult14_93_out;
wire [13:0] mult14_94_out;
wire [13:0] mult14_95_out;
wire [13:0] mult14_96_out;
wire [13:0] mult14_97_out;
wire [13:0] mult14_98_out;
wire [13:0] mult14_99_out;
wire [13:0] mult14_100_out;

wire [13:0] mult14_101_out;
wire [13:0] mult14_102_out;
wire [13:0] mult14_103_out;
wire [13:0] mult14_104_out;
wire [13:0] mult14_105_out;
wire [13:0] mult14_106_out;
wire [13:0] mult14_107_out;
wire [13:0] mult14_108_out;
wire [13:0] mult14_109_out;
wire [13:0] mult14_110_out;
wire [13:0] mult14_111_out;
wire [13:0] mult14_112_out;
wire [13:0] mult14_113_out;
wire [13:0] mult14_114_out;
wire [13:0] mult14_115_out;
wire [13:0] mult14_116_out;
wire [13:0] mult14_117_out;
wire [13:0] mult14_118_out;
```

```
wire [13:0] mult14_119_out;
wire [13:0] mult14_120_out;
wire [13:0] mult14_121_out;
wire [13:0] mult14_122_out;
wire [13:0] mult14_123_out;
wire [13:0] mult14_124_out;
wire [13:0] mult14_125_out;
wire [13:0] mult14_126_out;
wire [13:0] mult14_127_out;
wire [13:0] mult14_128_out;
wire [13:0] mult14_129_out;
wire [13:0] mult14_130_out;
wire [13:0] mult14_131_out;
wire [13:0] mult14_132_out;
wire [13:0] mult14_133_out;
wire [13:0] mult14_134_out;
wire [13:0] mult14_135_out;
wire [13:0] mult14_136_out;
wire [13:0] mult14_137_out;
wire [13:0] mult14_138_out;
wire [13:0] mult14_139_out;
wire [13:0] mult14_140_out;
wire [13:0] mult14_141_out;
wire [13:0] mult14_142_out;
wire [13:0] mult14_143_out;
wire [13:0] mult14_144_out;
wire [13:0] mult14_145_out;
wire [13:0] mult14_146_out;
wire [13:0] mult14_147_out;
wire [13:0] mult14_148_out;
wire [13:0] mult14_149_out;
wire [13:0] mult14_150_out;
wire [13:0] mult14_151_out;
wire [13:0] mult14_152_out;
wire [13:0] mult14_153_out;
wire [13:0] mult14_154_out;
wire [13:0] mult14_155_out;
wire [13:0] mult14_156_out;
wire [13:0] mult14_157_out;
wire [13:0] mult14_158_out;
wire [13:0] mult14_159_out;
wire [13:0] mult14_160_out;
//wire [13:0] mult14_161_out;

// ----------module - mult14----------
mult14 mult14_1(.in1(m9k14_1_q),.in2(ram1_out1),.clk(clk_gen_clk_inv),.out(mult14_1_out));
mult14 mult14_2(.in1(m9k14_2_q),.in2(ram1_out2),.clk(clk_gen_clk_inv),.out(mult14_2_out));
mult14 mult14_3(.in1(m9k14_3_q),.in2(ram1_out3),.clk(clk_gen_clk_inv),.out(mult14_3_out));
mult14 mult14_4(.in1(m9k14_4_q),.in2(ram1_out4),.clk(clk_gen_clk_inv),.out(mult14_4_out));
mult14 mult14_5(.in1(m9k14_5_q),.in2(ram1_out5),.clk(clk_gen_clk_inv),.out(mult14_5_out));
mult14 mult14_6(.in1(m9k14_6_q),.in2(ram1_out6),.clk(clk_gen_clk_inv),.out(mult14_6_out));
```

```
mult14 mult14_7(.in1(m9k14_7_q),.in2(ram1_out7),.clk(clk_gen_clk_inv),.out(mult14_7_out));
mult14 mult14_8(.in1(m9k14_8_q),.in2(ram1_out8),.clk(clk_gen_clk_inv),.out(mult14_8_out));
mult14 mult14_9(.in1(m9k14_9_q),.in2(ram1_out9),.clk(clk_gen_clk_inv),.out(mult14_9_out));
mult14 mult14_10(.in1(m9k14_10_q),.in2(ram1_out10),.clk(clk_gen_clk_inv),.out(mult14_10_out));
mult14 mult14_11(.in1(m9k14_11_q),.in2(ram1_out11),.clk(clk_gen_clk_inv),.out(mult14_11_out));
mult14 mult14_12(.in1(m9k14_12_q),.in2(ram1_out12),.clk(clk_gen_clk_inv),.out(mult14_12_out));
mult14 mult14_13(.in1(m9k14_13_q),.in2(ram1_out13),.clk(clk_gen_clk_inv),.out(mult14_13_out));
mult14 mult14_14(.in1(m9k14_14_q),.in2(ram1_out14),.clk(clk_gen_clk_inv),.out(mult14_14_out));
mult14 mult14_15(.in1(m9k14_15_q),.in2(ram1_out15),.clk(clk_gen_clk_inv),.out(mult14_15_out));
mult14 mult14_16(.in1(m9k14_16_q),.in2(ram1_out16),.clk(clk_gen_clk_inv),.out(mult14_16_out));
mult14 mult14_17(.in1(m9k14_17_q),.in2(ram1_out17),.clk(clk_gen_clk_inv),.out(mult14_17_out));
mult14 mult14_18(.in1(m9k14_18_q),.in2(ram1_out18),.clk(clk_gen_clk_inv),.out(mult14_18_out));
mult14 mult14_19(.in1(m9k14_19_q),.in2(ram1_out19),.clk(clk_gen_clk_inv),.out(mult14_19_out));
mult14 mult14_20(.in1(m9k14_20_q),.in2(ram1_out20),.clk(clk_gen_clk_inv),.out(mult14_20_out));
mult14 mult14_21(.in1(m9k14_21_q),.in2(ram1_out21),.clk(clk_gen_clk_inv),.out(mult14_21_out));
mult14 mult14_22(.in1(m9k14_22_q),.in2(ram1_out22),.clk(clk_gen_clk_inv),.out(mult14_22_out));
mult14 mult14_23(.in1(m9k14_23_q),.in2(ram1_out23),.clk(clk_gen_clk_inv),.out(mult14_23_out));
mult14 mult14_24(.in1(m9k14_24_q),.in2(ram1_out24),.clk(clk_gen_clk_inv),.out(mult14_24_out));
mult14 mult14_25(.in1(m9k14_25_q),.in2(ram1_out25),.clk(clk_gen_clk_inv),.out(mult14_25_out));
mult14 mult14_26(.in1(m9k14_26_q),.in2(ram1_out26),.clk(clk_gen_clk_inv),.out(mult14_26_out));
mult14 mult14_27(.in1(m9k14_27_q),.in2(ram1_out27),.clk(clk_gen_clk_inv),.out(mult14_27_out));
mult14 mult14_28(.in1(m9k14_28_q),.in2(ram1_out28),.clk(clk_gen_clk_inv),.out(mult14_28_out));
mult14 mult14_29(.in1(m9k14_29_q),.in2(ram1_out29),.clk(clk_gen_clk_inv),.out(mult14_29_out));
mult14 mult14_30(.in1(m9k14_30_q),.in2(ram1_out30),.clk(clk_gen_clk_inv),.out(mult14_30_out));
mult14 mult14_31(.in1(m9k14_31_q),.in2(ram1_out31),.clk(clk_gen_clk_inv),.out(mult14_31_out));
mult14 mult14_32(.in1(m9k14_32_q),.in2(ram1_out32),.clk(clk_gen_clk_inv),.out(mult14_32_out));
mult14 mult14_33(.in1(m9k14_33_q),.in2(ram1_out33),.clk(clk_gen_clk_inv),.out(mult14_33_out));
mult14 mult14_34(.in1(m9k14_34_q),.in2(ram1_out34),.clk(clk_gen_clk_inv),.out(mult14_34_out));
mult14 mult14_35(.in1(m9k14_35_q),.in2(ram1_out35),.clk(clk_gen_clk_inv),.out(mult14_35_out));
mult14 mult14_36(.in1(m9k14_36_q),.in2(ram1_out36),.clk(clk_gen_clk_inv),.out(mult14_36_out));
mult14 mult14_37(.in1(m9k14_37_q),.in2(ram1_out37),.clk(clk_gen_clk_inv),.out(mult14_37_out));
mult14 mult14_38(.in1(m9k14_38_q),.in2(ram1_out38),.clk(clk_gen_clk_inv),.out(mult14_38_out));
mult14 mult14_39(.in1(m9k14_39_q),.in2(ram1_out39),.clk(clk_gen_clk_inv),.out(mult14_39_out));
mult14 mult14_40(.in1(m9k14_40_q),.in2(ram1_out40),.clk(clk_gen_clk_inv),.out(mult14_40_out));
mult14 mult14_41(.in1(m9k14_41_q),.in2(ram1_out41),.clk(clk_gen_clk_inv),.out(mult14_41_out));
mult14 mult14_42(.in1(m9k14_42_q),.in2(ram1_out42),.clk(clk_gen_clk_inv),.out(mult14_42_out));
mult14 mult14_43(.in1(m9k14_43_q),.in2(ram1_out43),.clk(clk_gen_clk_inv),.out(mult14_43_out));
mult14 mult14_44(.in1(m9k14_44_q),.in2(ram1_out44),.clk(clk_gen_clk_inv),.out(mult14_44_out));
mult14 mult14_45(.in1(m9k14_45_q),.in2(ram1_out45),.clk(clk_gen_clk_inv),.out(mult14_45_out));
mult14 mult14_46(.in1(m9k14_46_q),.in2(ram1_out46),.clk(clk_gen_clk_inv),.out(mult14_46_out));
mult14 mult14_47(.in1(m9k14_47_q),.in2(ram1_out47),.clk(clk_gen_clk_inv),.out(mult14_47_out));
mult14 mult14_48(.in1(m9k14_48_q),.in2(ram1_out48),.clk(clk_gen_clk_inv),.out(mult14_48_out));
mult14 mult14_49(.in1(m9k14_49_q),.in2(ram1_out49),.clk(clk_gen_clk_inv),.out(mult14_49_out));
mult14 mult14_50(.in1(m9k14_50_q),.in2(ram1_out50),.clk(clk_gen_clk_inv),.out(mult14_50_out));
mult14 mult14_51(.in1(m9k14_51_q),.in2(ram1_out51),.clk(clk_gen_clk_inv),.out(mult14_51_out));
mult14 mult14_52(.in1(m9k14_52_q),.in2(ram1_out52),.clk(clk_gen_clk_inv),.out(mult14_52_out));
mult14 mult14_53(.in1(m9k14_53_q),.in2(ram1_out53),.clk(clk_gen_clk_inv),.out(mult14_53_out));
mult14 mult14_54(.in1(m9k14_54_q),.in2(ram1_out54),.clk(clk_gen_clk_inv),.out(mult14_54_out));
mult14 mult14_55(.in1(m9k14_55_q),.in2(ram1_out55),.clk(clk_gen_clk_inv),.out(mult14_55_out));
mult14 mult14_56(.in1(m9k14_56_q),.in2(ram1_out56),.clk(clk_gen_clk_inv),.out(mult14_56_out));
mult14 mult14_57(.in1(m9k14_57_q),.in2(ram1_out57),.clk(clk_gen_clk_inv),.out(mult14_57_out));
```

```
mult14 mult14_58(.in1(m9k14_58_q),.in2(ram1_out58),.clk(clk_gen_clk_inv),.out(mult14_58_out));
mult14 mult14_59(.in1(m9k14_59_q),.in2(ram1_out59),.clk(clk_gen_clk_inv),.out(mult14_59_out));
mult14 mult14_60(.in1(m9k14_60_q),.in2(ram1_out60),.clk(clk_gen_clk_inv),.out(mult14_60_out));
mult14 mult14_61(.in1(m9k14_61_q),.in2(ram1_out61),.clk(clk_gen_clk_inv),.out(mult14_61_out));
mult14 mult14_62(.in1(m9k14_62_q),.in2(ram1_out62),.clk(clk_gen_clk_inv),.out(mult14_62_out));
mult14 mult14_63(.in1(m9k14_63_q),.in2(ram1_out63),.clk(clk_gen_clk_inv),.out(mult14_63_out));
mult14 mult14_64(.in1(m9k14_64_q),.in2(ram1_out64),.clk(clk_gen_clk_inv),.out(mult14_64_out));
mult14 mult14_65(.in1(m9k14_65_q),.in2(ram1_out65),.clk(clk_gen_clk_inv),.out(mult14_65_out));
mult14 mult14_66(.in1(m9k14_66_q),.in2(ram1_out66),.clk(clk_gen_clk_inv),.out(mult14_66_out));
mult14 mult14_67(.in1(m9k14_67_q),.in2(ram1_out67),.clk(clk_gen_clk_inv),.out(mult14_67_out));
mult14 mult14_68(.in1(m9k14_68_q),.in2(ram1_out68),.clk(clk_gen_clk_inv),.out(mult14_68_out));
mult14 mult14_69(.in1(m9k14_69_q),.in2(ram1_out69),.clk(clk_gen_clk_inv),.out(mult14_69_out));
mult14 mult14_70(.in1(m9k14_70_q),.in2(ram1_out70),.clk(clk_gen_clk_inv),.out(mult14_70_out));
mult14 mult14_71(.in1(m9k14_71_q),.in2(ram1_out71),.clk(clk_gen_clk_inv),.out(mult14_71_out));
mult14 mult14_72(.in1(m9k14_72_q),.in2(ram1_out72),.clk(clk_gen_clk_inv),.out(mult14_72_out));
mult14 mult14_73(.in1(m9k14_73_q),.in2(ram1_out73),.clk(clk_gen_clk_inv),.out(mult14_73_out));
mult14 mult14_74(.in1(m9k14_74_q),.in2(ram1_out74),.clk(clk_gen_clk_inv),.out(mult14_74_out));
mult14 mult14_75(.in1(m9k14_75_q),.in2(ram1_out75),.clk(clk_gen_clk_inv),.out(mult14_75_out));
mult14 mult14_76(.in1(m9k14_76_q),.in2(ram1_out76),.clk(clk_gen_clk_inv),.out(mult14_76_out));
mult14 mult14_77(.in1(m9k14_77_q),.in2(ram1_out77),.clk(clk_gen_clk_inv),.out(mult14_77_out));
mult14 mult14_78(.in1(m9k14_78_q),.in2(ram1_out78),.clk(clk_gen_clk_inv),.out(mult14_78_out));
mult14 mult14_79(.in1(m9k14_79_q),.in2(ram1_out79),.clk(clk_gen_clk_inv),.out(mult14_79_out));
mult14 mult14_80(.in1(m9k14_80_q),.in2(ram1_out80),.clk(clk_gen_clk_inv),.out(mult14_80_out));

mult14 mult14_81(.in1(m9k14_81_q),.in2(ram1_out81),.clk(clk_gen_clk_inv),.out(mult14_81_out));
mult14 mult14_82(.in1(m9k14_82_q),.in2(ram1_out82),.clk(clk_gen_clk_inv),.out(mult14_82_out));
mult14 mult14_83(.in1(m9k14_83_q),.in2(ram1_out83),.clk(clk_gen_clk_inv),.out(mult14_83_out));
mult14 mult14_84(.in1(m9k14_84_q),.in2(ram1_out84),.clk(clk_gen_clk_inv),.out(mult14_84_out));
mult14 mult14_85(.in1(m9k14_85_q),.in2(ram1_out85),.clk(clk_gen_clk_inv),.out(mult14_85_out));
mult14 mult14_86(.in1(m9k14_86_q),.in2(ram1_out86),.clk(clk_gen_clk_inv),.out(mult14_86_out));
mult14 mult14_87(.in1(m9k14_87_q),.in2(ram1_out87),.clk(clk_gen_clk_inv),.out(mult14_87_out));
mult14 mult14_88(.in1(m9k14_88_q),.in2(ram1_out88),.clk(clk_gen_clk_inv),.out(mult14_88_out));
mult14 mult14_89(.in1(m9k14_89_q),.in2(ram1_out89),.clk(clk_gen_clk_inv),.out(mult14_89_out));
mult14 mult14_90(.in1(m9k14_90_q),.in2(ram1_out90),.clk(clk_gen_clk_inv),.out(mult14_90_out));
mult14 mult14_91(.in1(m9k14_91_q),.in2(ram1_out91),.clk(clk_gen_clk_inv),.out(mult14_91_out));
mult14 mult14_92(.in1(m9k14_92_q),.in2(ram1_out92),.clk(clk_gen_clk_inv),.out(mult14_92_out));
mult14 mult14_93(.in1(m9k14_93_q),.in2(ram1_out93),.clk(clk_gen_clk_inv),.out(mult14_93_out));
mult14 mult14_94(.in1(m9k14_94_q),.in2(ram1_out94),.clk(clk_gen_clk_inv),.out(mult14_94_out));
mult14 mult14_95(.in1(m9k14_95_q),.in2(ram1_out95),.clk(clk_gen_clk_inv),.out(mult14_95_out));
mult14 mult14_96(.in1(m9k14_96_q),.in2(ram1_out96),.clk(clk_gen_clk_inv),.out(mult14_96_out));
mult14 mult14_97(.in1(m9k14_97_q),.in2(ram1_out97),.clk(clk_gen_clk_inv),.out(mult14_97_out));
mult14 mult14_98(.in1(m9k14_98_q),.in2(ram1_out98),.clk(clk_gen_clk_inv),.out(mult14_98_out));
mult14 mult14_99(.in1(m9k14_99_q),.in2(ram1_out99),.clk(clk_gen_clk_inv),.out(mult14_99_out));
mult14
mult14_100(.in1(m9k14_100_q),.in2(ram1_out100),.clk(clk_gen_clk_inv),.out(mult14_100_out));

mult14
mult14_101(.in1(m9k14_101_q),.in2(ram1_out101),.clk(clk_gen_clk_inv),.out(mult14_101_out));
mult14
mult14_102(.in1(m9k14_102_q),.in2(ram1_out102),.clk(clk_gen_clk_inv),.out(mult14_102_out));
```

```
mult14
mult14_103(.in1(m9k14_103_q),.in2(ram1_out103),.clk(clk_gen_clk_inv),.out(mult14_103_out));
mult14
mult14_104(.in1(m9k14_104_q),.in2(ram1_out104),.clk(clk_gen_clk_inv),.out(mult14_104_out));
mult14
mult14_105(.in1(m9k14_105_q),.in2(ram1_out105),.clk(clk_gen_clk_inv),.out(mult14_105_out));
mult14
mult14_106(.in1(m9k14_106_q),.in2(ram1_out106),.clk(clk_gen_clk_inv),.out(mult14_106_out));
mult14
mult14_107(.in1(m9k14_107_q),.in2(ram1_out107),.clk(clk_gen_clk_inv),.out(mult14_107_out));
mult14
mult14_108(.in1(m9k14_108_q),.in2(ram1_out108),.clk(clk_gen_clk_inv),.out(mult14_108_out));
mult14
mult14_109(.in1(m9k14_109_q),.in2(ram1_out109),.clk(clk_gen_clk_inv),.out(mult14_109_out));
mult14
mult14_110(.in1(m9k14_110_q),.in2(ram1_out110),.clk(clk_gen_clk_inv),.out(mult14_110_out));
mult14
mult14_111(.in1(m9k14_111_q),.in2(ram1_out111),.clk(clk_gen_clk_inv),.out(mult14_111_out));
mult14
mult14_112(.in1(m9k14_112_q),.in2(ram1_out112),.clk(clk_gen_clk_inv),.out(mult14_112_out));
mult14
mult14_113(.in1(m9k14_113_q),.in2(ram1_out113),.clk(clk_gen_clk_inv),.out(mult14_113_out));
mult14
mult14_114(.in1(m9k14_114_q),.in2(ram1_out114),.clk(clk_gen_clk_inv),.out(mult14_114_out));
mult14
mult14_115(.in1(m9k14_115_q),.in2(ram1_out115),.clk(clk_gen_clk_inv),.out(mult14_115_out));
mult14
mult14_116(.in1(m9k14_116_q),.in2(ram1_out116),.clk(clk_gen_clk_inv),.out(mult14_116_out));
mult14
mult14_117(.in1(m9k14_117_q),.in2(ram1_out117),.clk(clk_gen_clk_inv),.out(mult14_117_out));
mult14
mult14_118(.in1(m9k14_118_q),.in2(ram1_out118),.clk(clk_gen_clk_inv),.out(mult14_118_out));
mult14
mult14_119(.in1(m9k14_119_q),.in2(ram1_out119),.clk(clk_gen_clk_inv),.out(mult14_119_out));
mult14
mult14_120(.in1(m9k14_120_q),.in2(ram1_out120),.clk(clk_gen_clk_inv),.out(mult14_120_out));
mult14
mult14_121(.in1(m9k14_121_q),.in2(ram1_out121),.clk(clk_gen_clk_inv),.out(mult14_121_out));
mult14
mult14_122(.in1(m9k14_122_q),.in2(ram1_out122),.clk(clk_gen_clk_inv),.out(mult14_122_out));
mult14
mult14_123(.in1(m9k14_123_q),.in2(ram1_out123),.clk(clk_gen_clk_inv),.out(mult14_123_out));
mult14
mult14_124(.in1(m9k14_124_q),.in2(ram1_out124),.clk(clk_gen_clk_inv),.out(mult14_124_out));
mult14
mult14_125(.in1(m9k14_125_q),.in2(ram1_out125),.clk(clk_gen_clk_inv),.out(mult14_125_out));
mult14
mult14_126(.in1(m9k14_126_q),.in2(ram1_out126),.clk(clk_gen_clk_inv),.out(mult14_126_out));
mult14
mult14_127(.in1(m9k14_127_q),.in2(ram1_out127),.clk(clk_gen_clk_inv),.out(mult14_127_out));
```

```
mult14
mult14_128(.in1(m9k14_128_q),.in2(ram1_out128),.clk(clk_gen_clk_inv),.out(mult14_128_out));
mult14
mult14_129(.in1(m9k14_129_q),.in2(ram1_out129),.clk(clk_gen_clk_inv),.out(mult14_129_out));
mult14
mult14_130(.in1(m9k14_130_q),.in2(ram1_out130),.clk(clk_gen_clk_inv),.out(mult14_130_out));
mult14
mult14_131(.in1(m9k14_131_q),.in2(ram1_out131),.clk(clk_gen_clk_inv),.out(mult14_131_out));
mult14
mult14_132(.in1(m9k14_132_q),.in2(ram1_out132),.clk(clk_gen_clk_inv),.out(mult14_132_out));
mult14
mult14_133(.in1(m9k14_133_q),.in2(ram1_out133),.clk(clk_gen_clk_inv),.out(mult14_133_out));
mult14
mult14_134(.in1(m9k14_134_q),.in2(ram1_out134),.clk(clk_gen_clk_inv),.out(mult14_134_out));
mult14
mult14_135(.in1(m9k14_135_q),.in2(ram1_out135),.clk(clk_gen_clk_inv),.out(mult14_135_out));
mult14
mult14_136(.in1(m9k14_136_q),.in2(ram1_out136),.clk(clk_gen_clk_inv),.out(mult14_136_out));
mult14
mult14_137(.in1(m9k14_137_q),.in2(ram1_out137),.clk(clk_gen_clk_inv),.out(mult14_137_out));
mult14
mult14_138(.in1(m9k14_138_q),.in2(ram1_out138),.clk(clk_gen_clk_inv),.out(mult14_138_out));
mult14
mult14_139(.in1(m9k14_139_q),.in2(ram1_out139),.clk(clk_gen_clk_inv),.out(mult14_139_out));
mult14
mult14_140(.in1(m9k14_140_q),.in2(ram1_out140),.clk(clk_gen_clk_inv),.out(mult14_140_out));
mult14
mult14_141(.in1(m9k14_141_q),.in2(ram1_out141),.clk(clk_gen_clk_inv),.out(mult14_141_out));
mult14
mult14_142(.in1(m9k14_142_q),.in2(ram1_out142),.clk(clk_gen_clk_inv),.out(mult14_142_out));
mult14
mult14_143(.in1(m9k14_143_q),.in2(ram1_out143),.clk(clk_gen_clk_inv),.out(mult14_143_out));
mult14
mult14_144(.in1(m9k14_144_q),.in2(ram1_out144),.clk(clk_gen_clk_inv),.out(mult14_144_out));
mult14
mult14_145(.in1(m9k14_145_q),.in2(ram1_out145),.clk(clk_gen_clk_inv),.out(mult14_145_out));
mult14
mult14_146(.in1(m9k14_146_q),.in2(ram1_out146),.clk(clk_gen_clk_inv),.out(mult14_146_out));
mult14
mult14_147(.in1(m9k14_147_q),.in2(ram1_out147),.clk(clk_gen_clk_inv),.out(mult14_147_out));
mult14
mult14_148(.in1(m9k14_148_q),.in2(ram1_out148),.clk(clk_gen_clk_inv),.out(mult14_148_out));
mult14
mult14_149(.in1(m9k14_149_q),.in2(ram1_out149),.clk(clk_gen_clk_inv),.out(mult14_149_out));
mult14
mult14_150(.in1(m9k14_150_q),.in2(ram1_out150),.clk(clk_gen_clk_inv),.out(mult14_150_out));
mult14
mult14_151(.in1(m9k14_151_q),.in2(ram1_out151),.clk(clk_gen_clk_inv),.out(mult14_151_out));
mult14
mult14_152(.in1(m9k14_152_q),.in2(ram1_out152),.clk(clk_gen_clk_inv),.out(mult14_152_out));
```

```
mult14
mult14_153(.in1(m9k14_153_q),.in2(ram1_out153),.clk(clk_gen_clk_inv),.out(mult14_153_out));
mult14
mult14_154(.in1(m9k14_154_q),.in2(ram1_out154),.clk(clk_gen_clk_inv),.out(mult14_154_out));
mult14
mult14_155(.in1(m9k14_155_q),.in2(ram1_out155),.clk(clk_gen_clk_inv),.out(mult14_155_out));
mult14
mult14_156(.in1(m9k14_156_q),.in2(ram1_out156),.clk(clk_gen_clk_inv),.out(mult14_156_out));
mult14
mult14_157(.in1(m9k14_157_q),.in2(ram1_out157),.clk(clk_gen_clk_inv),.out(mult14_157_out));
mult14
mult14_158(.in1(m9k14_158_q),.in2(ram1_out158),.clk(clk_gen_clk_inv),.out(mult14_158_out));
mult14
mult14_159(.in1(m9k14_159_q),.in2(ram1_out159),.clk(clk_gen_clk_inv),.out(mult14_159_out));
mult14
mult14_160(.in1(m9k14_160_q),.in2(ram1_out160),.clk(clk_gen_clk_inv),.out(mult14_160_out));
reg [13:0] mult14_161_out;
always @ (posedge clk_gen_clk_inv)
begin
mult14_161_out <= m9k14_161_q;
end
// ----------wire - com2mag167 ----------
wire [13:0] com2mag167_1_out;
wire [13:0] com2mag167_2_out;
wire [13:0] com2mag167_3_out;
wire [13:0] com2mag167_4_out;
wire [13:0] com2mag167_5_out;
wire [13:0] com2mag167_6_out;
wire [13:0] com2mag167_7_out;
wire [13:0] com2mag167_8_out;
wire [13:0] com2mag167_9_out;
wire [13:0] com2mag167_10_out;
wire [13:0] com2mag167_11_out;
wire [13:0] com2mag167_12_out;
wire [13:0] com2mag167_13_out;
wire [13:0] com2mag167_14_out;
wire [13:0] com2mag167_15_out;
wire [13:0] com2mag167_16_out;
wire [13:0] com2mag167_17_out;
wire [13:0] com2mag167_18_out;
wire [13:0] com2mag167_19_out;
wire [13:0] com2mag167_20_out;
wire [13:0] com2mag167_21_out;
wire [13:0] com2mag167_22_out;
wire [13:0] com2mag167_23_out;
wire [13:0] com2mag167_24_out;
wire [13:0] com2mag167_25_out;
wire [13:0] com2mag167_26_out;
wire [13:0] com2mag167_27_out;
wire [13:0] com2mag167_28_out;
wire [13:0] com2mag167_29_out;
```

```
wire [13:0] com2mag167_30_out;
wire [13:0] com2mag167_31_out;
wire [13:0] com2mag167_32_out;
wire [13:0] com2mag167_33_out;
wire [13:0] com2mag167_34_out;
wire [13:0] com2mag167_35_out;
wire [13:0] com2mag167_36_out;
wire [13:0] com2mag167_37_out;
wire [13:0] com2mag167_38_out;
wire [13:0] com2mag167_39_out;
wire [13:0] com2mag167_40_out;
wire [13:0] com2mag167_41_out;
wire [13:0] com2mag167_42_out;
wire [13:0] com2mag167_43_out;
wire [13:0] com2mag167_44_out;
wire [13:0] com2mag167_45_out;
wire [13:0] com2mag167_46_out;
wire [13:0] com2mag167_47_out;
wire [13:0] com2mag167_48_out;
wire [13:0] com2mag167_49_out;
wire [13:0] com2mag167_50_out;
wire [13:0] com2mag167_51_out;
wire [13:0] com2mag167_52_out;
wire [13:0] com2mag167_53_out;
wire [13:0] com2mag167_54_out;
wire [13:0] com2mag167_55_out;
wire [13:0] com2mag167_56_out;
wire [13:0] com2mag167_57_out;
wire [13:0] com2mag167_58_out;
wire [13:0] com2mag167_59_out;
wire [13:0] com2mag167_60_out;
wire [13:0] com2mag167_61_out;
wire [13:0] com2mag167_62_out;
wire [13:0] com2mag167_63_out;
wire [13:0] com2mag167_64_out;
wire [13:0] com2mag167_65_out;
wire [13:0] com2mag167_66_out;
wire [13:0] com2mag167_67_out;
wire [13:0] com2mag167_68_out;
wire [13:0] com2mag167_69_out;
wire [13:0] com2mag167_70_out;
wire [13:0] com2mag167_71_out;
wire [13:0] com2mag167_72_out;
wire [13:0] com2mag167_73_out;
wire [13:0] com2mag167_74_out;
wire [13:0] com2mag167_75_out;
wire [13:0] com2mag167_76_out;
wire [13:0] com2mag167_77_out;
wire [13:0] com2mag167_78_out;
wire [13:0] com2mag167_79_out;
wire [13:0] com2mag167_80_out;
```

```
wire [13:0] com2mag167_81_out;
wire [13:0] com2mag167_82_out;
wire [13:0] com2mag167_83_out;
wire [13:0] com2mag167_84_out;
wire [13:0] com2mag167_85_out;
wire [13:0] com2mag167_86_out;
wire [13:0] com2mag167_87_out;
wire [13:0] com2mag167_88_out;
wire [13:0] com2mag167_89_out;
wire [13:0] com2mag167_90_out;
wire [13:0] com2mag167_91_out;
wire [13:0] com2mag167_92_out;
wire [13:0] com2mag167_93_out;
wire [13:0] com2mag167_94_out;
wire [13:0] com2mag167_95_out;
wire [13:0] com2mag167_96_out;
wire [13:0] com2mag167_97_out;
wire [13:0] com2mag167_98_out;
wire [13:0] com2mag167_99_out;
wire [13:0] com2mag167_100_out;

wire [13:0] com2mag167_101_out;
wire [13:0] com2mag167_102_out;
wire [13:0] com2mag167_103_out;
wire [13:0] com2mag167_104_out;
wire [13:0] com2mag167_105_out;
wire [13:0] com2mag167_106_out;
wire [13:0] com2mag167_107_out;
wire [13:0] com2mag167_108_out;
wire [13:0] com2mag167_109_out;
wire [13:0] com2mag167_110_out;
wire [13:0] com2mag167_111_out;
wire [13:0] com2mag167_112_out;
wire [13:0] com2mag167_113_out;
wire [13:0] com2mag167_114_out;
wire [13:0] com2mag167_115_out;
wire [13:0] com2mag167_116_out;
wire [13:0] com2mag167_117_out;
wire [13:0] com2mag167_118_out;
wire [13:0] com2mag167_119_out;
wire [13:0] com2mag167_120_out;
wire [13:0] com2mag167_121_out;
wire [13:0] com2mag167_122_out;
wire [13:0] com2mag167_123_out;
wire [13:0] com2mag167_124_out;
wire [13:0] com2mag167_125_out;
wire [13:0] com2mag167_126_out;
wire [13:0] com2mag167_127_out;
wire [13:0] com2mag167_128_out;
wire [13:0] com2mag167_129_out;
wire [13:0] com2mag167_130_out;
```

```
wire [13:0] com2mag167_131_out;
wire [13:0] com2mag167_132_out;
wire [13:0] com2mag167_133_out;
wire [13:0] com2mag167_134_out;
wire [13:0] com2mag167_135_out;
wire [13:0] com2mag167_136_out;
wire [13:0] com2mag167_137_out;
wire [13:0] com2mag167_138_out;
wire [13:0] com2mag167_139_out;
wire [13:0] com2mag167_140_out;
wire [13:0] com2mag167_141_out;
wire [13:0] com2mag167_142_out;
wire [13:0] com2mag167_143_out;
wire [13:0] com2mag167_144_out;
wire [13:0] com2mag167_145_out;
wire [13:0] com2mag167_146_out;
wire [13:0] com2mag167_147_out;
wire [13:0] com2mag167_148_out;
wire [13:0] com2mag167_149_out;
wire [13:0] com2mag167_150_out;
wire [13:0] com2mag167_151_out;
wire [13:0] com2mag167_152_out;
wire [13:0] com2mag167_153_out;
wire [13:0] com2mag167_154_out;
wire [13:0] com2mag167_155_out;
wire [13:0] com2mag167_156_out;
wire [13:0] com2mag167_157_out;
wire [13:0] com2mag167_158_out;
wire [13:0] com2mag167_159_out;
wire [13:0] com2mag167_160_out;
wire [13:0] com2mag167_161_out;

// ----------module - com2mag167-----------
com2mag167 com2mag167_1(.in(mult14_1_out),.out(com2mag167_1_out));
com2mag167 com2mag167_2(.in(mult14_2_out),.out(com2mag167_2_out));
com2mag167 com2mag167_3(.in(mult14_3_out),.out(com2mag167_3_out));
com2mag167 com2mag167_4(.in(mult14_4_out),.out(com2mag167_4_out));
com2mag167 com2mag167_5(.in(mult14_5_out),.out(com2mag167_5_out));
com2mag167 com2mag167_6(.in(mult14_6_out),.out(com2mag167_6_out));
com2mag167 com2mag167_7(.in(mult14_7_out),.out(com2mag167_7_out));
com2mag167 com2mag167_8(.in(mult14_8_out),.out(com2mag167_8_out));
com2mag167 com2mag167_9(.in(mult14_9_out),.out(com2mag167_9_out));
com2mag167 com2mag167_10(.in(mult14_10_out),.out(com2mag167_10_out));
com2mag167 com2mag167_11(.in(mult14_11_out),.out(com2mag167_11_out));
com2mag167 com2mag167_12(.in(mult14_12_out),.out(com2mag167_12_out));
com2mag167 com2mag167_13(.in(mult14_13_out),.out(com2mag167_13_out));
com2mag167 com2mag167_14(.in(mult14_14_out),.out(com2mag167_14_out));
com2mag167 com2mag167_15(.in(mult14_15_out),.out(com2mag167_15_out));
com2mag167 com2mag167_16(.in(mult14_16_out),.out(com2mag167_16_out));
com2mag167 com2mag167_17(.in(mult14_17_out),.out(com2mag167_17_out));
com2mag167 com2mag167_18(.in(mult14_18_out),.out(com2mag167_18_out));
```

```
com2mag167 com2mag167_19(.in(mult14_19_out),.out(com2mag167_19_out));
com2mag167 com2mag167_20(.in(mult14_20_out),.out(com2mag167_20_out));
com2mag167 com2mag167_21(.in(mult14_21_out),.out(com2mag167_21_out));
com2mag167 com2mag167_22(.in(mult14_22_out),.out(com2mag167_22_out));
com2mag167 com2mag167_23(.in(mult14_23_out),.out(com2mag167_23_out));
com2mag167 com2mag167_24(.in(mult14_24_out),.out(com2mag167_24_out));
com2mag167 com2mag167_25(.in(mult14_25_out),.out(com2mag167_25_out));
com2mag167 com2mag167_26(.in(mult14_26_out),.out(com2mag167_26_out));
com2mag167 com2mag167_27(.in(mult14_27_out),.out(com2mag167_27_out));
com2mag167 com2mag167_28(.in(mult14_28_out),.out(com2mag167_28_out));
com2mag167 com2mag167_29(.in(mult14_29_out),.out(com2mag167_29_out));
com2mag167 com2mag167_30(.in(mult14_30_out),.out(com2mag167_30_out));
com2mag167 com2mag167_31(.in(mult14_31_out),.out(com2mag167_31_out));
com2mag167 com2mag167_32(.in(mult14_32_out),.out(com2mag167_32_out));
com2mag167 com2mag167_33(.in(mult14_33_out),.out(com2mag167_33_out));
com2mag167 com2mag167_34(.in(mult14_34_out),.out(com2mag167_34_out));
com2mag167 com2mag167_35(.in(mult14_35_out),.out(com2mag167_35_out));
com2mag167 com2mag167_36(.in(mult14_36_out),.out(com2mag167_36_out));
com2mag167 com2mag167_37(.in(mult14_37_out),.out(com2mag167_37_out));
com2mag167 com2mag167_38(.in(mult14_38_out),.out(com2mag167_38_out));
com2mag167 com2mag167_39(.in(mult14_39_out),.out(com2mag167_39_out));
com2mag167 com2mag167_40(.in(mult14_40_out),.out(com2mag167_40_out));
com2mag167 com2mag167_41(.in(mult14_41_out),.out(com2mag167_41_out));
com2mag167 com2mag167_42(.in(mult14_42_out),.out(com2mag167_42_out));
com2mag167 com2mag167_43(.in(mult14_43_out),.out(com2mag167_43_out));
com2mag167 com2mag167_44(.in(mult14_44_out),.out(com2mag167_44_out));
com2mag167 com2mag167_45(.in(mult14_45_out),.out(com2mag167_45_out));
com2mag167 com2mag167_46(.in(mult14_46_out),.out(com2mag167_46_out));
com2mag167 com2mag167_47(.in(mult14_47_out),.out(com2mag167_47_out));
com2mag167 com2mag167_48(.in(mult14_48_out),.out(com2mag167_48_out));
com2mag167 com2mag167_49(.in(mult14_49_out),.out(com2mag167_49_out));
com2mag167 com2mag167_50(.in(mult14_50_out),.out(com2mag167_50_out));
com2mag167 com2mag167_51(.in(mult14_51_out),.out(com2mag167_51_out));
com2mag167 com2mag167_52(.in(mult14_52_out),.out(com2mag167_52_out));
com2mag167 com2mag167_53(.in(mult14_53_out),.out(com2mag167_53_out));
com2mag167 com2mag167_54(.in(mult14_54_out),.out(com2mag167_54_out));
com2mag167 com2mag167_55(.in(mult14_55_out),.out(com2mag167_55_out));
com2mag167 com2mag167_56(.in(mult14_56_out),.out(com2mag167_56_out));
com2mag167 com2mag167_57(.in(mult14_57_out),.out(com2mag167_57_out));
com2mag167 com2mag167_58(.in(mult14_58_out),.out(com2mag167_58_out));
com2mag167 com2mag167_59(.in(mult14_59_out),.out(com2mag167_59_out));
com2mag167 com2mag167_60(.in(mult14_60_out),.out(com2mag167_60_out));
com2mag167 com2mag167_61(.in(mult14_61_out),.out(com2mag167_61_out));
com2mag167 com2mag167_62(.in(mult14_62_out),.out(com2mag167_62_out));
com2mag167 com2mag167_63(.in(mult14_63_out),.out(com2mag167_63_out));
com2mag167 com2mag167_64(.in(mult14_64_out),.out(com2mag167_64_out));
com2mag167 com2mag167_65(.in(mult14_65_out),.out(com2mag167_65_out));
com2mag167 com2mag167_66(.in(mult14_66_out),.out(com2mag167_66_out));
com2mag167 com2mag167_67(.in(mult14_67_out),.out(com2mag167_67_out));
com2mag167 com2mag167_68(.in(mult14_68_out),.out(com2mag167_68_out));
com2mag167 com2mag167_69(.in(mult14_69_out),.out(com2mag167_69_out));
```

```
com2mag167 com2mag167_70(.in(mult14_70_out),.out(com2mag167_70_out));
com2mag167 com2mag167_71(.in(mult14_71_out),.out(com2mag167_71_out));
com2mag167 com2mag167_72(.in(mult14_72_out),.out(com2mag167_72_out));
com2mag167 com2mag167_73(.in(mult14_73_out),.out(com2mag167_73_out));
com2mag167 com2mag167_74(.in(mult14_74_out),.out(com2mag167_74_out));
com2mag167 com2mag167_75(.in(mult14_75_out),.out(com2mag167_75_out));
com2mag167 com2mag167_76(.in(mult14_76_out),.out(com2mag167_76_out));
com2mag167 com2mag167_77(.in(mult14_77_out),.out(com2mag167_77_out));
com2mag167 com2mag167_78(.in(mult14_78_out),.out(com2mag167_78_out));
com2mag167 com2mag167_79(.in(mult14_79_out),.out(com2mag167_79_out));
com2mag167 com2mag167_80(.in(mult14_80_out),.out(com2mag167_80_out));
com2mag167 com2mag167_81(.in(mult14_81_out),.out(com2mag167_81_out));
com2mag167 com2mag167_82(.in(mult14_82_out),.out(com2mag167_82_out));
com2mag167 com2mag167_83(.in(mult14_83_out),.out(com2mag167_83_out));
com2mag167 com2mag167_84(.in(mult14_84_out),.out(com2mag167_84_out));
com2mag167 com2mag167_85(.in(mult14_85_out),.out(com2mag167_85_out));
com2mag167 com2mag167_86(.in(mult14_86_out),.out(com2mag167_86_out));
com2mag167 com2mag167_87(.in(mult14_87_out),.out(com2mag167_87_out));
com2mag167 com2mag167_88(.in(mult14_88_out),.out(com2mag167_88_out));
com2mag167 com2mag167_89(.in(mult14_89_out),.out(com2mag167_89_out));
com2mag167 com2mag167_90(.in(mult14_90_out),.out(com2mag167_90_out));
com2mag167 com2mag167_91(.in(mult14_91_out),.out(com2mag167_91_out));
com2mag167 com2mag167_92(.in(mult14_92_out),.out(com2mag167_92_out));
com2mag167 com2mag167_93(.in(mult14_93_out),.out(com2mag167_93_out));
com2mag167 com2mag167_94(.in(mult14_94_out),.out(com2mag167_94_out));
com2mag167 com2mag167_95(.in(mult14_95_out),.out(com2mag167_95_out));
com2mag167 com2mag167_96(.in(mult14_96_out),.out(com2mag167_96_out));
com2mag167 com2mag167_97(.in(mult14_97_out),.out(com2mag167_97_out));
com2mag167 com2mag167_98(.in(mult14_98_out),.out(com2mag167_98_out));
com2mag167 com2mag167_99(.in(mult14_99_out),.out(com2mag167_99_out));
com2mag167 com2mag167_100(.in(mult14_100_out),.out(com2mag167_100_out));

com2mag167 com2mag167_101(.in(mult14_101_out),.out(com2mag167_101_out));
com2mag167 com2mag167_102(.in(mult14_102_out),.out(com2mag167_102_out));
com2mag167 com2mag167_103(.in(mult14_103_out),.out(com2mag167_103_out));
com2mag167 com2mag167_104(.in(mult14_104_out),.out(com2mag167_104_out));
com2mag167 com2mag167_105(.in(mult14_105_out),.out(com2mag167_105_out));
com2mag167 com2mag167_106(.in(mult14_106_out),.out(com2mag167_106_out));
com2mag167 com2mag167_107(.in(mult14_107_out),.out(com2mag167_107_out));
com2mag167 com2mag167_108(.in(mult14_108_out),.out(com2mag167_108_out));
com2mag167 com2mag167_109(.in(mult14_109_out),.out(com2mag167_109_out));
com2mag167 com2mag167_110(.in(mult14_110_out),.out(com2mag167_110_out));
com2mag167 com2mag167_111(.in(mult14_111_out),.out(com2mag167_111_out));
com2mag167 com2mag167_112(.in(mult14_112_out),.out(com2mag167_112_out));
com2mag167 com2mag167_113(.in(mult14_113_out),.out(com2mag167_113_out));
com2mag167 com2mag167_114(.in(mult14_114_out),.out(com2mag167_114_out));
com2mag167 com2mag167_115(.in(mult14_115_out),.out(com2mag167_115_out));
com2mag167 com2mag167_116(.in(mult14_116_out),.out(com2mag167_116_out));
com2mag167 com2mag167_117(.in(mult14_117_out),.out(com2mag167_117_out));
com2mag167 com2mag167_118(.in(mult14_118_out),.out(com2mag167_118_out));
com2mag167 com2mag167_119(.in(mult14_119_out),.out(com2mag167_119_out));
```

```
com2mag167 com2mag167_120(.in(mult14_120_out),.out(com2mag167_120_out));
com2mag167 com2mag167_121(.in(mult14_121_out),.out(com2mag167_121_out));
com2mag167 com2mag167_122(.in(mult14_122_out),.out(com2mag167_122_out));
com2mag167 com2mag167_123(.in(mult14_123_out),.out(com2mag167_123_out));
com2mag167 com2mag167_124(.in(mult14_124_out),.out(com2mag167_124_out));
com2mag167 com2mag167_125(.in(mult14_125_out),.out(com2mag167_125_out));
com2mag167 com2mag167_126(.in(mult14_126_out),.out(com2mag167_126_out));
com2mag167 com2mag167_127(.in(mult14_127_out),.out(com2mag167_127_out));
com2mag167 com2mag167_128(.in(mult14_128_out),.out(com2mag167_128_out));
com2mag167 com2mag167_129(.in(mult14_129_out),.out(com2mag167_129_out));
com2mag167 com2mag167_130(.in(mult14_130_out),.out(com2mag167_130_out));
com2mag167 com2mag167_131(.in(mult14_131_out),.out(com2mag167_131_out));
com2mag167 com2mag167_132(.in(mult14_132_out),.out(com2mag167_132_out));
com2mag167 com2mag167_133(.in(mult14_133_out),.out(com2mag167_133_out));
com2mag167 com2mag167_134(.in(mult14_134_out),.out(com2mag167_134_out));
com2mag167 com2mag167_135(.in(mult14_135_out),.out(com2mag167_135_out));
com2mag167 com2mag167_136(.in(mult14_136_out),.out(com2mag167_136_out));
com2mag167 com2mag167_137(.in(mult14_137_out),.out(com2mag167_137_out));
com2mag167 com2mag167_138(.in(mult14_138_out),.out(com2mag167_138_out));
com2mag167 com2mag167_139(.in(mult14_139_out),.out(com2mag167_139_out));
com2mag167 com2mag167_140(.in(mult14_140_out),.out(com2mag167_140_out));
com2mag167 com2mag167_141(.in(mult14_141_out),.out(com2mag167_141_out));
com2mag167 com2mag167_142(.in(mult14_142_out),.out(com2mag167_142_out));
com2mag167 com2mag167_143(.in(mult14_143_out),.out(com2mag167_143_out));
com2mag167 com2mag167_144(.in(mult14_144_out),.out(com2mag167_144_out));
com2mag167 com2mag167_145(.in(mult14_145_out),.out(com2mag167_145_out));
com2mag167 com2mag167_146(.in(mult14_146_out),.out(com2mag167_146_out));
com2mag167 com2mag167_147(.in(mult14_147_out),.out(com2mag167_147_out));
com2mag167 com2mag167_148(.in(mult14_148_out),.out(com2mag167_148_out));
com2mag167 com2mag167_149(.in(mult14_149_out),.out(com2mag167_149_out));
com2mag167 com2mag167_150(.in(mult14_150_out),.out(com2mag167_150_out));
com2mag167 com2mag167_151(.in(mult14_151_out),.out(com2mag167_151_out));
com2mag167 com2mag167_152(.in(mult14_152_out),.out(com2mag167_152_out));
com2mag167 com2mag167_153(.in(mult14_153_out),.out(com2mag167_153_out));
com2mag167 com2mag167_154(.in(mult14_154_out),.out(com2mag167_154_out));
com2mag167 com2mag167_155(.in(mult14_155_out),.out(com2mag167_155_out));
com2mag167 com2mag167_156(.in(mult14_156_out),.out(com2mag167_156_out));
com2mag167 com2mag167_157(.in(mult14_157_out),.out(com2mag167_157_out));
com2mag167 com2mag167_158(.in(mult14_158_out),.out(com2mag167_158_out));
com2mag167 com2mag167_159(.in(mult14_159_out),.out(com2mag167_159_out));
com2mag167 com2mag167_160(.in(mult14_160_out),.out(com2mag167_160_out));
com2mag167 com2mag167_161(.in(mult14_161_out),.out(com2mag167_161_out));
// ----------wire - adder161----------
wire [13:0] adder161_1_out;
// ---------module - adder161----------
adder161
adder161_1(.in1(com2mag167_1_out),.in2(com2mag167_2_out),.in3(com2mag167_3_out),.in4(com2ma
g167_4_out),.in5(com2mag167_5_out),.in6(com2mag167_6_out),.in7(com2mag167_7_out),.in8(com2m
ag167_8_out),.in9(com2mag167_9_out),.in10(com2mag167_10_out),.in11(com2mag167_11_out),.in12(
com2mag167_12_out),.in13(com2mag167_13_out),.in14(com2mag167_14_out),.in15(com2mag167_15_
out),.in16(com2mag167_16_out),.in17(com2mag167_17_out),.in18(com2mag167_18_out),.in19(com2m
```

```
ag167_19_out),.in20(com2mag167_20_out),.in21(com2mag167_21_out),.in22(com2mag167_22_out),.in
23(com2mag167_23_out),.in24(com2mag167_24_out),.in25(com2mag167_25_out),.in26(com2mag167_
26_out),.in27(com2mag167_27_out),.in28(com2mag167_28_out),.in29(com2mag167_29_out),.in30(com
2mag167_30_out),.in31(com2mag167_31_out),.in32(com2mag167_32_out),.in33(com2mag167_33_out)
,.in34(com2mag167_34_out),.in35(com2mag167_35_out),.in36(com2mag167_36_out),.in37(com2mag1
67_37_out),.in38(com2mag167_38_out),.in39(com2mag167_39_out),.in40(com2mag167_40_out),.in41(
com2mag167_41_out),.in42(com2mag167_42_out),.in43(com2mag167_43_out),.in44(com2mag167_44_
out),.in45(com2mag167_45_out),.in46(com2mag167_46_out),.in47(com2mag167_47_out),.in48(com2m
ag167_48_out),.in49(com2mag167_49_out),.in50(com2mag167_50_out),.in51(com2mag167_51_out),.in
52(com2mag167_52_out),.in53(com2mag167_53_out),.in54(com2mag167_54_out),.in55(com2mag167_
55_out),.in56(com2mag167_56_out),.in57(com2mag167_57_out),.in58(com2mag167_58_out),.in59(com
2mag167_59_out),.in60(com2mag167_60_out),.in61(com2mag167_61_out),.in62(com2mag167_62_out)
,.in63(com2mag167_63_out),.in64(com2mag167_64_out),.in65(com2mag167_65_out),.in66(com2mag1
67_66_out),.in67(com2mag167_67_out),.in68(com2mag167_68_out),.in69(com2mag167_69_out),.in70(
com2mag167_70_out),.in71(com2mag167_71_out),.in72(com2mag167_72_out),.in73(com2mag167_73_
out),.in74(com2mag167_74_out),.in75(com2mag167_75_out),.in76(com2mag167_76_out),.in77(com2m
ag167_77_out),.in78(com2mag167_78_out),.in79(com2mag167_79_out),.in80(com2mag167_80_out),.in
81(com2mag167_81_out),.in82(com2mag167_82_out),.in83(com2mag167_83_out),.in84(com2mag167_
84_out),.in85(com2mag167_85_out),.in86(com2mag167_86_out),.in87(com2mag167_87_out),.in88(com
2mag167_88_out),.in89(com2mag167_89_out),.in90(com2mag167_90_out),.in91(com2mag167_91_out)
,.in92(com2mag167_92_out),.in93(com2mag167_93_out),.in94(com2mag167_94_out),.in95(com2mag1
67_95_out),.in96(com2mag167_96_out),.in97(com2mag167_97_out),.in98(com2mag167_98_out),.in99(
com2mag167_99_out),.in100(com2mag167_100_out),.in101(com2mag167_101_out),.in102(com2mag16
7_102_out),.in103(com2mag167_103_out),.in104(com2mag167_104_out),.in105(com2mag167_105_out
),.in106(com2mag167_106_out),.in107(com2mag167_107_out),.in108(com2mag167_108_out),.in109(co
m2mag167_109_out),.in110(com2mag167_110_out),.in111(com2mag167_111_out),.in112(com2mag167
_112_out),.in113(com2mag167_113_out),.in114(com2mag167_114_out),.in115(com2mag167_115_out),
.in116(com2mag167_116_out),.in117(com2mag167_117_out),.in118(com2mag167_118_out),.in119(co
m2mag167_119_out),.in120(com2mag167_120_out),.in121(com2mag167_121_out),.in122(com2mag167
_122_out),.in123(com2mag167_123_out),.in124(com2mag167_124_out),.in125(com2mag167_125_out),
.in126(com2mag167_126_out),.in127(com2mag167_127_out),.in128(com2mag167_128_out),.in129(co
m2mag167_129_out),.in130(com2mag167_130_out),.in131(com2mag167_131_out),.in132(com2mag167
_132_out),.in133(com2mag167_133_out),.in134(com2mag167_134_out),.in135(com2mag167_135_out),
.in136(com2mag167_136_out),.in137(com2mag167_137_out),.in138(com2mag167_138_out),.in139(co
m2mag167_139_out),.in140(com2mag167_140_out),.in141(com2mag167_141_out),.in142(com2mag167
_142_out),.in143(com2mag167_143_out),.in144(com2mag167_144_out),.in145(com2mag167_145_out),
.in146(com2mag167_146_out),.in147(com2mag167_147_out),.in148(com2mag167_148_out),.in149(co
m2mag167_149_out),.in150(com2mag167_150_out),.in151(com2mag167_151_out),.in152(com2mag167
_152_out),.in153(com2mag167_153_out),.in154(com2mag167_154_out),.in155(com2mag167_155_out),
.in156(com2mag167_156_out),.in157(com2mag167_157_out),.in158(com2mag167_158_out),.in159(co
m2mag167_159_out),.in160(com2mag167_160_out),.in161(com2mag167_161_out),
.clk(clk_gen_clk),.out(adder161_1_out));
// -------wire - com2mag167 (after adder)--------
wire [13:0] com2mag167_162_out;
// ---------module com2mag167 (after addfer)-----------
com2mag167 com2mag167_162(.in(adder161_1_out),.out(com2mag167_162_out));
// ----------wire - htangent165eo004---------
wire [5:0] htangent165e004_2_out;
// ---------module - htangent165e004---------
```

```
htangent165e004
htangent165e004_2(.clk(clk_gen_clk_inv),.in(com2mag167_162_out[13:2]),.out(htangent165e004_2_out
));
// module - ram2
//ram2
ram2_1(.in(htangent165e004_2_out),.addr(controller_addr2_write),.clk(clk_gen_clk),.wr_en(controller_r
am2_en),.out1(out1),.out2(out2),.out3(out3),.out4(out4),.out5(out5),.out6(out6),.out7(out7),.out8(out8),.o
ut9(out9),.out10(out10),.out11(out11),.out12(out12),.out13(out13),.out14(out14),.out15(out15),.out16(out
16),.out17(out17),.out18(out18),.out19(out19),.out20(out20),.out21(out21),.out22(out22),.out23(out23),.o
ut24(out24),.out25(out25),.out26(out26),.out27(out27),.out28(out28),.out29(out29),.out30(out30),.out31(
out31),.out32(out32),.out33(out33),.out34(out34),.out35(out35),.out36(out36));
// ----------module - comparator------------
comparator
comparator_1(.in(htangent165e004_2_out),.addr(controller_addr2_write),.clk(clk_gen_clk),.clk_inv(clk_
gen_clk_inv),.wr_en(controller_comparator_en),.counter(controller_counter),.out(out));
endmodule
```

16. ann_tb.v

```
`timescale  1ns / 100ps
module ann_tb;
reg clk_source;
reg rst;
reg [188:0] inputdata [0:665];
reg [188:0] in;
wire [5:0] out;
integer i,j;
integer c;
initial
begin
$readmemb("testingInput1_t1.txt",inputdata);
in = inputdata [0];
clk_source = 0;
rst = 0;
#2860
for (i = 1; i <= 665; i = i+1)
begin
rst = 1;
in = inputdata[i];
#40
rst = 0;
#4400
$display("%g The *******outputsignal******* is %d",$time,out+1);
#20
c =0;
end
#3000000 $finish; // every character takes #4460 to process. 4460*665 = 2,965,900;
end
always #10 clk_source = ~clk_source;
ann DUT(clk_source,rst,in,out);
endmodule
```

Vita Auctoris

Yuan Jing was born in 1988 in Tianjin, China. He received his bachelor degree in 2011 in Electrical Engineering from Changchun University of Science and Technology. He's research field is Artificial neural network. He has been working as a research assistant for Professor Mitra Mirhassani for the last three years.