FPGA Implementation of Neural Network

Cuong T. Pham, Thu H. Nguyen, Huong L. D. Hoang

Dept. School of Electronics and Telecommunications

Hanoi University of Science and Technology, 1 Dai Co Viet Road, Ha Noi, Viet Nam

[phamcuong21478@gmail.com](mailto:phamcuong21478@gmail.com), [hathunguyen967@gmail.com](mailto:hathunguyen967@gmail.com), [hoangledieuhuong156@gmail.com](mailto:hoangledieuhuong156@gmail.com)

*Abstract*—Backpropagation  is a method used in [Artificial Neural Networks](https://en.wikipedia.org/wiki/Artificial_neural_network) (ANN), belongs to the group of supervised learning algorithms. In this paper, we propose an Artificial Neural Network Perceptron Model and training backpropagation algorithm for this network on FPGAs. We have implemented the ANN in many different sizes that solved lots of different problems and evaluated the system’s performance when the circuit size or the problem changes. The ALTERA Quartus II software is used for circuit synthesis, which indicates that our system can run at frequencies up to 72Mhz, the training speed approximates the speed of reading/writing data from memory. The performance of the instance with 60 input neurons, 60 hidden neurons and 3 output neurons is up to 97.07%.

Keywords—FPGA, Backpropagation, ANN

# Introduction

According to many previous studies of perceptron networks have shown, it is obvious that the more neurons the network has, the more weight and bias are there, therefore, training time will be longer. Implementing this perceptron network on LSI technology could accelerate parallel computing, which could dramatically reduce training time.

In this paper, we propose a way to implement a perceptron network with a backpropagation algorithm on it. The entire design was written in Verilog, using the ModelSim-Altera software to simulate, using the Quatus II software to compile and load to the DE2 kit for verification.

Our implementation is based on two techniques that are used to design the processor: **multi cycle** and **pipeline**. The **multi cycle** allows our design to make better use of resources, but takes more time to consume. The combination of **pipeline** and **multi cycle** allows both resource utilization and reduced the average necessary time. It is suitable for problems having lots of mathematical works.

And to evaluate our design, we ran simulations and tests on the system to solve some classification problems from simple to complex. First, we solve a simple classification problem using a *2: 3: 2* network architecture, then the problem of classifying vehicles (bicycles, motorcycles or cars) using *15: 15: 3* architecture network. Through these problems, our design has been proven and tested for computational efficiency.

# Backpropagation Algorithm Overview

## Multi-Layer Perceptron

Perceptron is an ANN consisting of an input layer, one or more hidden layers, and an output layer. The neurons in one layer will be fully connected to the neurons in the next layer, with no loop connections. Furthermore, there can be also a bias neuron in each input and hidden layers with the output value of these neuron always equal 1. The following figure shows a 2: 3: 2 perceptron network (2 input, 3 hidden, 2 output).

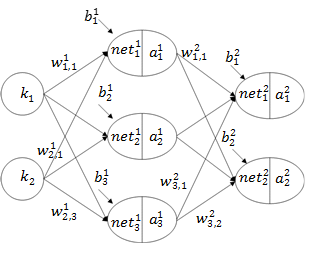


Fig. 1 an instance of 3-layer perceptron.

In the above figure, is the input vector of the network; is the weight of the connection to *pth* layer, between the two neurons: *pth* neuron in the previous layer and *jth* one in the *ith* layer; is the bias value of the *pth* neuron in *ith* layer (as known as *bias*); is the network input value of the *jth* neuron in *ith* layer; is the activation value of the *pth* neuron in *ith* layer, also known as the output of a neuron. For the input layer, output of a input neuron is the value of the network’s input vector. For the output neurons, its activation value is used to calculate the whole system’s output value. Normally, the output *y* is the activation of the neuron output.

(1)



In our design, we use sigmoid function as activation function.

(3)

## Backpropagation

To start the backpropagation algorithm, firstly the perceptron network must be initialized. Weight and bias values will be assigned to random values other than 0.

After that, all the vectors in training set will sequentially be taken to the network to calculate the output and from that calculate the cost function by this formula:

Where

With *i* is the corresponding *ith* training sample.

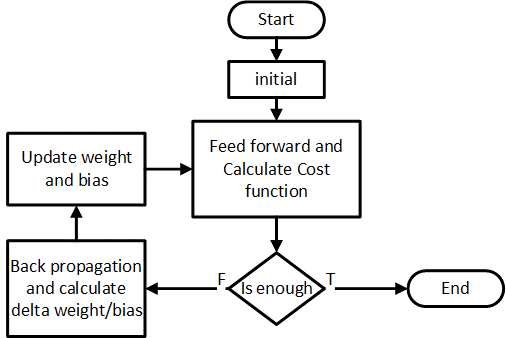


Fig. 2 Backpropagation algorithm.

After calculating the cost function, we will have to check the algorithm stop criteria. When this criteria is met, the algorithm will stop, otherwise the algorithm will continue. The stop criteria we use is **the limit of the number of interactions**. This algorithm will only run the specified number of interactions.

Thereafter finished calculating the value of the cost function, there will be the back propagation. At this stage, the partial derivatives of the Cost function depending on *w* and *b* are computed.

In the output layer, unit error value is calculated by this formula:

(4)

In which *p* is the order of output neurons, *y* is the activation value of the output neuron considered. We can calculate:

(5)



For hidden layers:

(7)

Through it, we also can calculate:

(8)



The partial derivative value is calculated by the formula:

(10)



Finally, the values of *w* and *b* will be updated according to the formula:

(12)







Value of in (12) and (13) equations are learning rate. It heavily affects the learning process of the algorithm. It usually approx.

# Proposed Design Architecture

## Original Idea

As we can see from formula (1), we needs many multiplications to calculate a net value. It is expensive to develop a module with many multipliers. Therefore, we have difficulties in designing neural networks which have a number of neurons. Our solution is that calculating net value in many clock cycles (‘multi cycles’ technique). Using this technique, calculating time is increase but the circuit’s size will be decrease significantly. Moreover, we have the same difficulty in calculating other given expressions in the algorithm. So multi cycle technique become more efficient in perceptron network implementation.

In addition, data in perceptron network are transferred from input layer to hidden layers, then come to output layer. Calculations in each layer must be completed before the calculations in next layer are carried out. Same thing happens in back propagation stage. Unit errors’ value in a layer must be calculated completely before calculating unit error values in previous layer. This gave us the ideal of separation calculating process into many phases.

When dividing calculating time into many phases, we can carry out many calculation phase by using pipeline technique. This makes average data processing speed improve significantly. So, this technique help us overcome calculating time increment when applying multi cycle technique.

Associating multi cycle technique and pipeline technique is our original idea in perceptron implementation. Below is our design, which is applied these techniques.

## Data path

As shown in the figure below, our design sequentially takes the elements of the input vector on a single channel (input channel). In the hidden layers, all of the neurons (each with a separate module) are connected to the input channel. Calculating activation value requires both receiving the input data, and receiving weight and bias data.

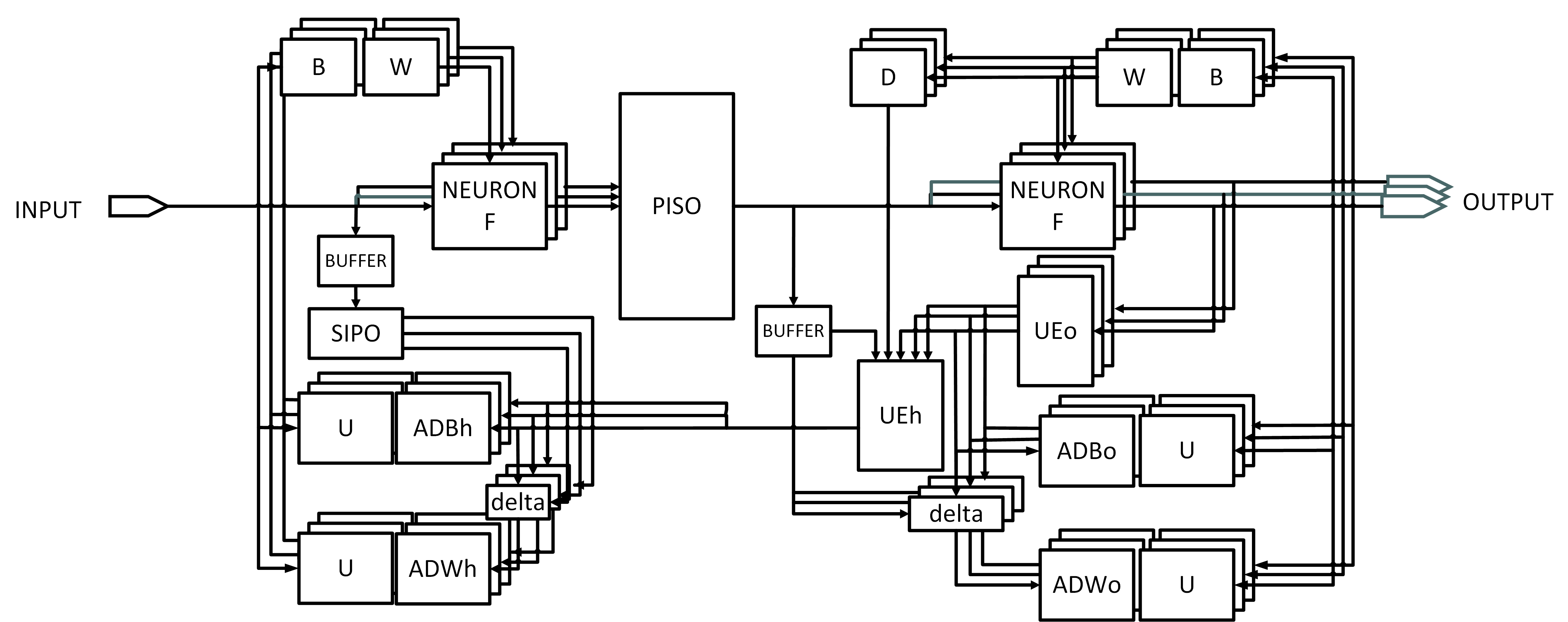


Fig. 3 Basic module diagrams and direction of data movement.

The separation between the hidden layers and the output layer is a PISO register. This register makes the calculation of the hidden layers repeatable for the output layer. And the architecture of the output layer is exactly the same as the one in the hidden layers.

The value of the output layer, besides being connected to the output of the whole system, is also connected to the unit error calculations for the output layer (UEo) performing the calculation in expression (4). Next, UEh will perform calculations for expression (7). Because the output layer is fixed to 3 neurons, we have designed this block to compute a unit error value for a hidden neuron in a clock cycle, and the whole system only needs this block.

Delta blocks is used for calculating (5) and (8) equations.

ADBo, ADWo, ADBh, ADWh blocks calculate variables in equations (10) and (11) for hidden and output layer.

There are two buffer registers are added to forward data between different phases. They are added when we apply the pipeline technique to the system.

The SIPO register is also added to the design during the optimization of the ADBh and ADWh blocks.

To avoid the confusing structure description, we did not show some other details of the system in the above figure. One of them is the module pool, which bridges the weight and bias values out.

## Control path

Our design have four main function, including: initial (*init*), training (*training*), run forward (*run*) and give out weight values and bias values of the neurons (*getW*). Two function *init* and *getW* can be implemented easily, *run* is actually a part of *training*. And the most complicated function is *training.*

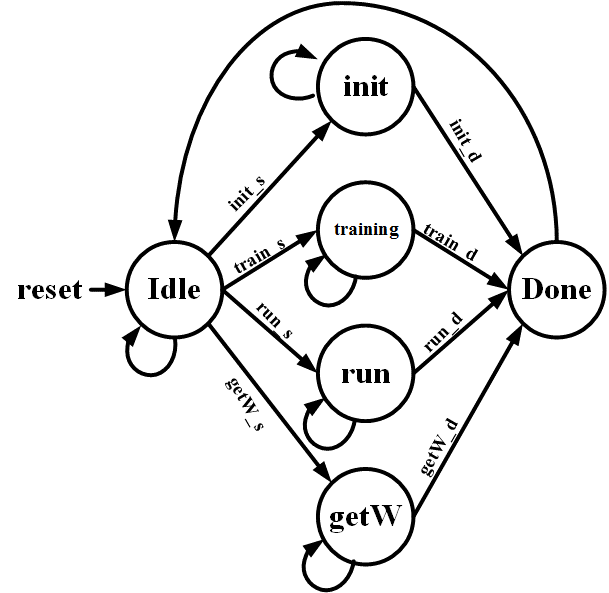


Fig. 4 FSM control system.

To implement training function, we divide calculating process into five phases. *Phase 1* performs computation at the hidden layer, *Phase 2* for the output layer, *Phase 3* and *Phase 4* perform calculations for back propagation. Finally, *Phase 5* performs the update of network weights.

Here is the timing diagram for phase of the 2: 3: 2 network.

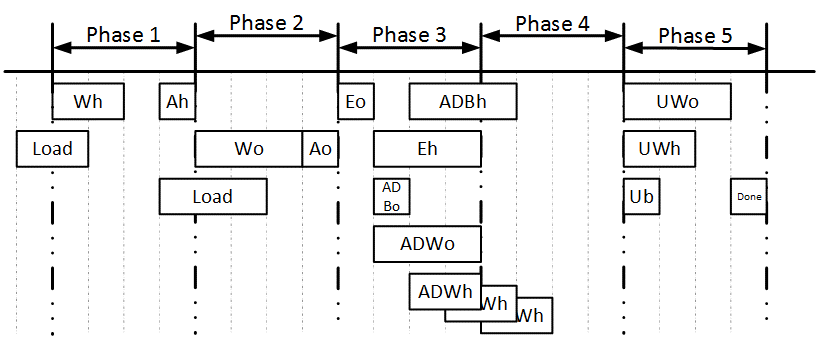


Fig. 5 Timing diagram for tasks in phases.

In the first two phases, *Wh* and *Wo* are respectively the net value computations for hidden and output layers. Before performing this calculation, the system need to read the relevant data from the memories. The process of reading this data (called *Load*) must be done a clock cycle before calculating net value. At the end of each phase, the design uses a clock cycle to calculate activation values based on net value (*Ah* and *Ao*).

In phase 3 and phase 4, the back propagation process will be performed. First, at the first clock cycle of phase 3, the unit error of the output layer is calculated (*Eo*). Soon after, in the next clock cycle, the accumulator of output layer will accumulates the delta bias (ADBo). At the same time, the delta weight of the output neurons is also started to calculate and accumulate (ADWo). Meanwhile, unit error of hidden layer is computed (Eh). As described in the Data path section, the block calculating unit error for the hidden layer will calculate for each neuron, therefore the Eh process will last three clock cycles (equal to the ADWo process). The implication is that the delta bias calculation (ADBh) process extends into three clock cycles for only one neuron. The delta weight (ADWh) calculation process of the hidden layer is stretched out as well. The ADWh calculation process for the last neuron in hidden layer will be completely calculated in phase 4.

In phase 5, updating weight value for the output neuron (UWo), neuron hidden (UWh) and bias of both layers (Ub) are performed as described in Fig. 5.

Each training sample has to go through the first four phases. After all the training samples have gone through the first four phases, the fifth phase will take place. In our design, the phases are independent, not conflicting, so they can work simultaneously. This allows our design to perform the pipeline technique described in Fig. 6.

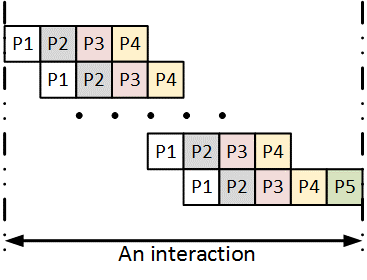


Fig. 6 The order of phases in an interaction using the pipeline technique.

Note that in phase 3 and phase 4, when performing back-propagation computation, the system requires re-use of data from phase 1 and phase 2. To do this, two *buffers* has been added to the data path.

In practice, we have designed a lot of control signals for the modules, but all of them are aimed at achieving the timing scheme in Fig. 5.

## Training time

In this section, we will describe some of the design timing calculations when training. Firstly, assume that the design consists of the number of input neurons is , the number of hidden neurons is and . Fixed output neurons are 3 and do not affect training time. Assume the training set has training samples and the number of interactions is . According to the pipeline technique described in Fig. 6, the number of phases during training is:

(16)

Each phase requires N + 1 clock cycles for calculating, so the training time will be:

(17)

Assuming time to read the data as a standard. The time spent reading the data throughout the training process is

(18)

If considered the ratio between system’s data calculating speed and reading speed from memory to evaluate the system performance, the system performance will be calculated:

(19)

Looking at expression (19) we can see that: when the training set size and the number of input (hidden) neurons are higher, the higher the system performance. For the 2: 3: 2 neural network and the training set is only 4 elements, the performance will be .. For a 15: 15: 3 network, a 63-element training program will have efficiency. And if the neural network is 60: 60: 3, the 300-element training package will have a

# Implementation and Result

Our design *(NN\_core)* only train perceptron network and calculate network output. In order to perform the data processing functions, a dedicated module *(Input\_vector)* must be included in the training and input files. This module has an architecture depending on the problem that need to be solved.

In order to test multiple different problems for different perceptron network sizes, we designed module *Input\_vector* is a module containing only the input vectors and the corresponding supervised values. The entire input processing process to generate the input vector for the perceptron is done by MATLAB.

Our second solved problem is classification three kind of vehicles: bike, motorbike and car images. The input images is fixed in 640 x 480 pixels, it means each image’s data has 307200 dimensions. We resized each image to 160 x 120 pixel, then applied Principal Component Analysis (PCA) to reduction data’ size. Finally, processed input is 15 dimensions, which means it keep 90% original information. We used 63 samples in training dataset. Our third problem is classification three character images: A (or a), B (or b), C (or c). The input images size is 20 x 20 pixels. We applied the same mentioned pre - processing for each data. After that, we have 15 dimensions data, which means 82,5% original information. In this instance, we used 300 training image samples. This processing is done completely in MATLAB.



Fig. 7 Input images of second instance.

We use Altera Quartus II software to implement the design. We solved three problems, using three instances of different sizes of designs. The ModelSim-Altera software is used to simulate circuit activity with specific problems. The following table summarizes the characteristics of the three instances.

1. result of running classification problems

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Instance 1** | **Instance 2** | **Instance 3** |
| **Network size** | 2:3:2 | 15:15:3 | 60:60:3 |
| **Target chip** | EP2C35F672C6 | EP2C35F672C6 | EP3C120F780I7 |
| **Utilization** | 2 076 (LE)  36 (Multiplier)  20 736 (Mem) | 8 148 (LE)  70 (Multiplier)  84 480 (Mem) | 15 633 (LE)  274 (Multiplier)  457 728 (Mem) |
| **Fmax** | 72.94 MHz | 70.34 MHz | 62.69 MHz |
| **Solved Problem** | Classify 4 domains in the plane.  Training samples: 8 | Classify bike, motorbike and car images  Training samples: 63 | Classify A(a) , B(b), C(c) letters.  Training samples: 300 |
| **Training time (1000 interactions)** | 48 000 (cycle) | 1 072 000 (cycle) | 18 544 000 (cycle) |
| **Performance** | 50% | 88.15% | 97.07% |
| **Correct classification rate** | 100% | 70% | ?? |

To make sure that the circuit is working properly, we perform simulations to calculate the cost function of each interaction. Here is a graph of the cost function for the second instance. We can see that the cost function has started to converge. Due to the error of the calculation module and the neural network model is not really suitable to solve the problem therefore the correct rate is low.

Fig.9, Fig.10 and Fig.10 illustrate simulation results of our design on ModelSim. As we can see from Fig.9, in first interaction of training phase, cost value accumulates from 0 to over 1,1 but in the next interactions, cost value decrease gradually as expected. In the final interaction of this phase, the bias values and weight values are exported. In conclusion, our design are simulated well on ModelSim as our original expectation.

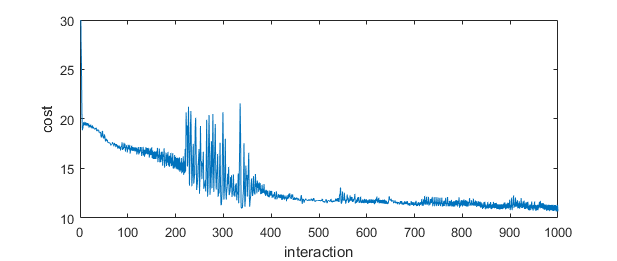


Fig. 8 Cost function for second instance.

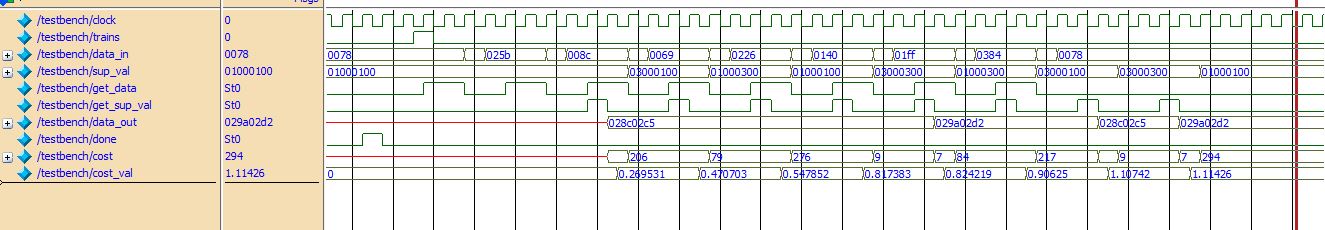


Fig. 9 Cost value accumulates in first interaction.

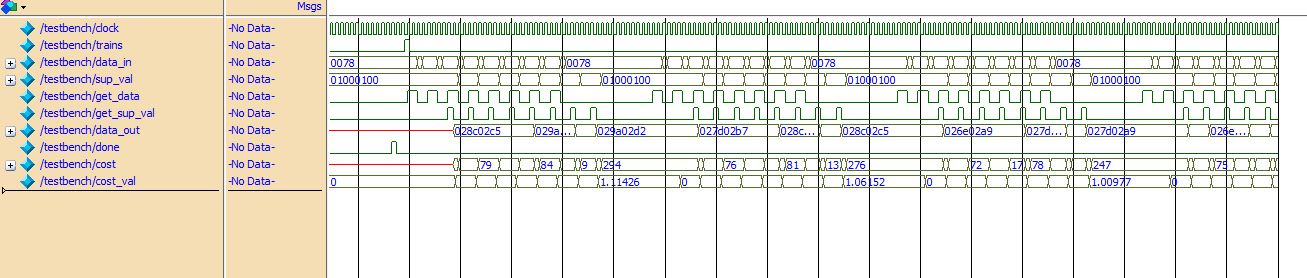


Fig. 10 Cost value for first instance decreases gradually through interactions in training phase.

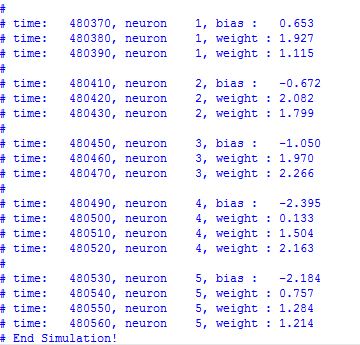


Fig. 11 Bias value and weight value are given out in final interaction of training phase for first instance.

# References

|  |  |
| --- | --- |
| [1] | D. Kriesel, A Brief Introduction to Neural Networks, 2007. |