VHDL Computer Assignment #3

Negin Safari
810101339

***Abstract*—This is the report of VHDL course computer assignment number three. In this computer assignment a kind of neural network named MLP is implemented using VHDL which is a hardware description language. This Hardware is trained to guess the type of iris flower by knowing four features of it.**

***Keywords*—Iris, MLP, neural network, VHDL.**

I. Introduction

A multi-layer perceptron (MLP) is a class of feedforward artificial neural networks (ANN). In a feedforward network, the neurons in each layer feed their output forward to the next layer until we get the final output from the neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and, an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. In recent developments of deep learning the Rectifier Linear Unit (ReLU) is more frequently used as an activation function. Since MLPs are fully connected, each node in one layer connects with a certain weight to every node in the following layer. Each neuron in the hidden layer often has its own bias constant. This bias matrix is added to the weighted input matrix before the hidden layer applies ReLU. In this computer assignment MLP is used for detection of the type of iris flower. The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician, eugenicist, and biologist Ronald Fisher. The data set consists of 50 samples from each of three species of Iris (Iris Setosa, Iris Virginica, and Iris Versicolor). Figure 4 shows these species. Four features, i.e., the length and width of the sepals and those of the petals, were measured in centimeters from each sample. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other. The perceptron is a simple algorithm intended to perform classification, i.e., it predicts whether the input belongs to a certain category of interest. In this assignment, you should design an MLP to classify the Iris flower species. It receives the features of a flower as its four inputs and recognizes the type of flower as output.

II. Design Phase

In this part the proposed design will be discussed. For desining this MLP, a neuron is designed which is used for making layers. The designs are all generic and a by multiple instantiation of layes and by assigning correct values to the parameters, MLP is implemented. A layer is made of a datapath and a controller. The datapath of a layer is made of some neurons, ROMs, a counter, ReLU, and a multiplexer. Each neuron has one combinational multiplier and an adder right after that and a register after that. In this way we can multiply each input by its corresponding weight and accumulate the results. After that by getting the bias value, it will be added to the accumulated value and it will be saved into another register. 8 bits of this data will be passed to the output as the final result of the neuron. In Figure 1 the design of one neuron is provided. As you can see the selected bits are 13 to 6 which leads to a good accuracy of computation. The second register is for saving the result in a place to have the ability to pipeline the design.

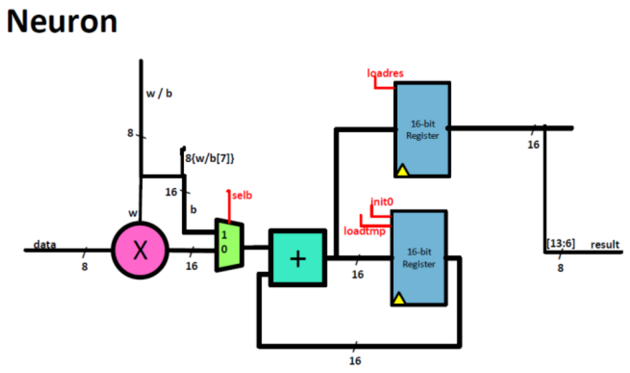


Figure 1. Neuron schematic

As it was described before, by multiple instantiation of this neuron and some other components such as counter and multiplexer. A ROM is provided to save weight and bias values in it. We have one ROM for each neuron. This ROM is initialized by text files containing corresponding values in hexadecimal. In Figure 3 one schematic is provided which is for the second hidden layer of this project. It has got ten input data, 8 neurons, and obviously 8 ROMs having 10 weights and one bias value in each of them. There is also ReLU function available on the output of the hidden layers. ReLU has a super simple structure. It has a comparator and a multiplexer and it passes the positive values as they are but for negative value it will put zero on output.

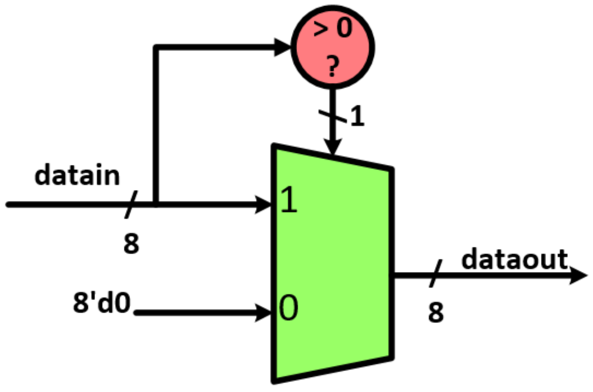


Figure 2: Relu schematic

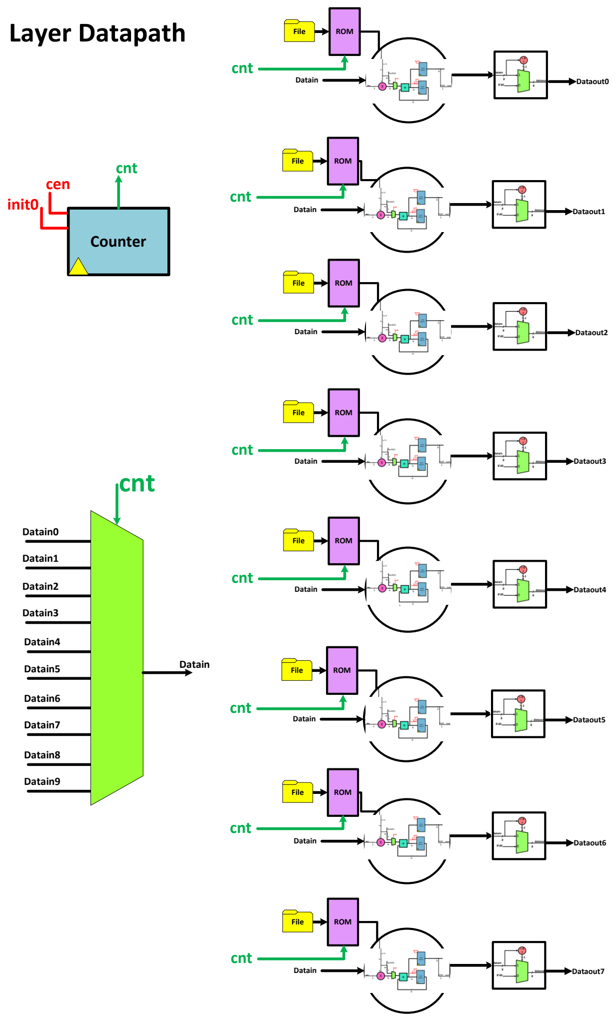


Figure 3: Layer Datapath (for second hidden

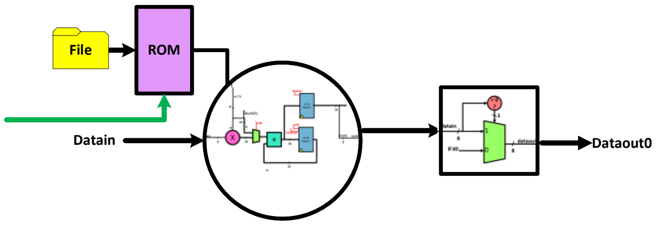


Figure 4: Combination of neuron and Relu and ROM

In Figure 4, a close up picture is provided to indicate how a neuron is actually connected to a ReLU function and also we wanted to show the connection of ROM and a neuron and how file and io functions help to initialize a memory element like ROM. It is important to add that reading from this ROM is within a clock and there is no need to wait for any cycles to get the data.

In Figure 5, the diagram of controller is illustrated. In this design, after having a positive value on start signal, the state changes from IDLE to INIT state. In INIT state counters and registers will be initialized to zero. The important point is that the result register doesn't have any initi0 port so it won't be initialized. After that in the next cycle the present state will be CAL1. In this state the multiplication of weight and data will happen and the results will be accumulated in temp register. In this state cen is set and the counter will count up. The counted value of counter is the address of our ROM and the select of the multiplexer that selects between input data. When the count value equals the number of inputs, the state changes into CAL2. In this state due to the value of the counter, the value of bias will be provided and the addition of accumulated data and sign extended bias will be executed and the result will be saved in result register. After one clock cycle the state changes into ENDING state. In this state we initialize the temp register and issue the done signal. At this point by acknowledging the done signal we can say that the result is ready. In Figure 4 the schematic of this finite state machine is provided. The VHDL code of the controller and the datapath is written according to these schematics.

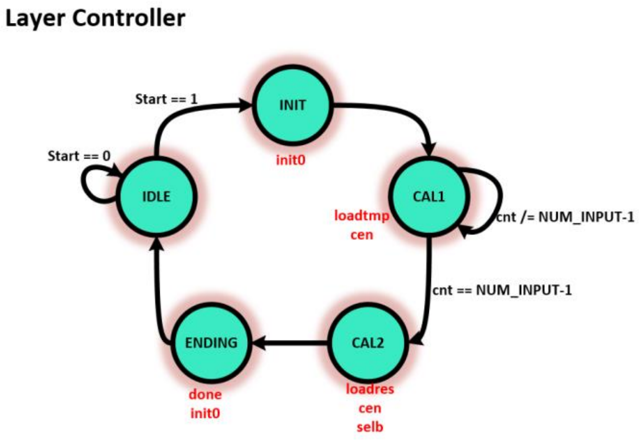


Figure 5: Controller of Layer

The files used for initializing ROMs are filled in a specific manner. First of all in each line the weights of that neuron are placed in order and in the end the bias value is placed. In Figure 6 a simple schematic is placed for the format of these files.

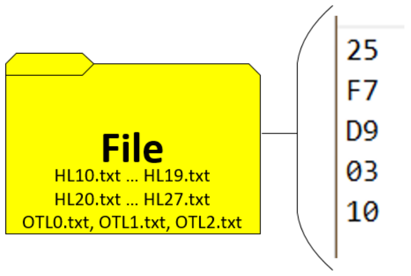


Figure 6: Initialization files

In Figure 7 the handshaking and connection of layer datapath and its controller is provided.

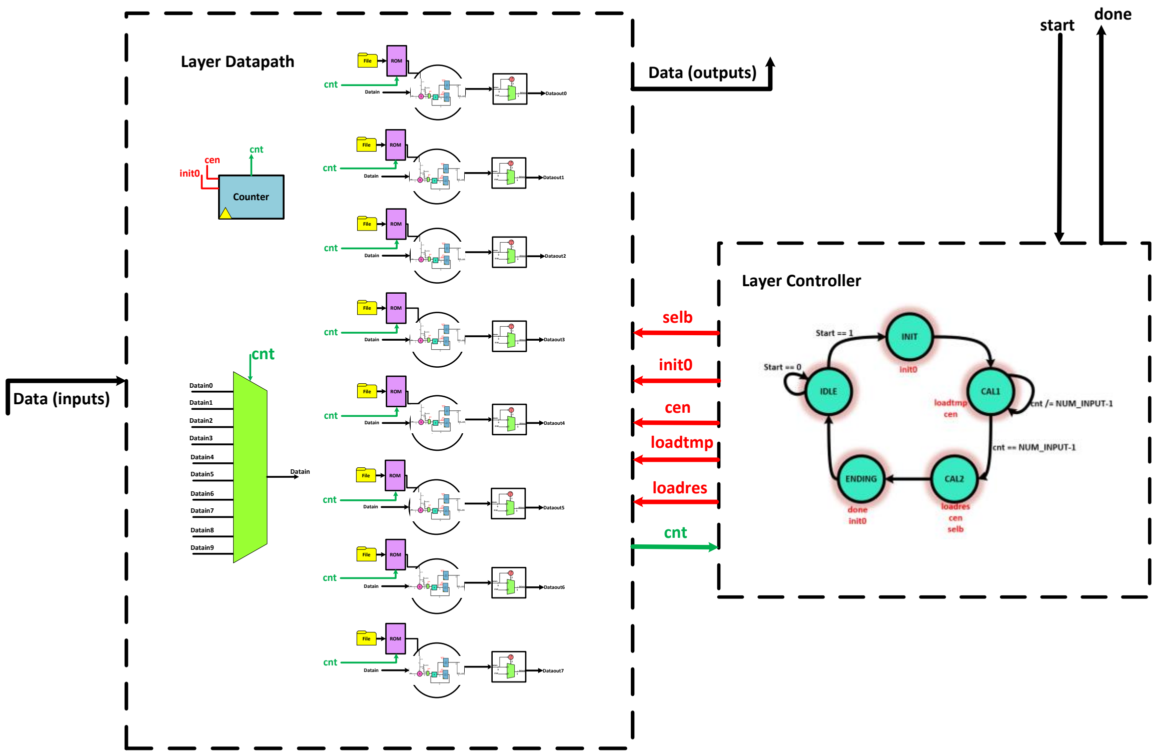


Figure 7: Handshaking within a layer

As you can see in Figure 7 the whole things related to a layer is provided and in previous paragraphs it all has been described. Now we are going to connect multiple layers and build a complete MLP. In this MLP we have two hidden layers and one output layer. The first hidden layer is consist of 10 neurons and 4 inputs. The second hidden layer is consist of 8 neurons and 10 inputs and the output layer is made of 3 neurons and it has 8 inputs. Hidden layers have ReLU but the output layer doesn't have it. In the end of calculations, we will have 3 output data coming from output layers, we should find the output which has the most value and the results are signed.So, the most positive output must be find and according to the neuron it is coming from, we can tell which type is it. We have three types. If that neuron was the first one, the type would be Setosa. If that neuron was the second one, the type would be Versicolor. If that neuron was the first one, the type would be Virginica. Overall, in Figure 8 there is a schematic of the whole MLP, here the pipeline is vivid. The done output of each stage is the start of the next one. Figure 8 draws the architecture of MLP.

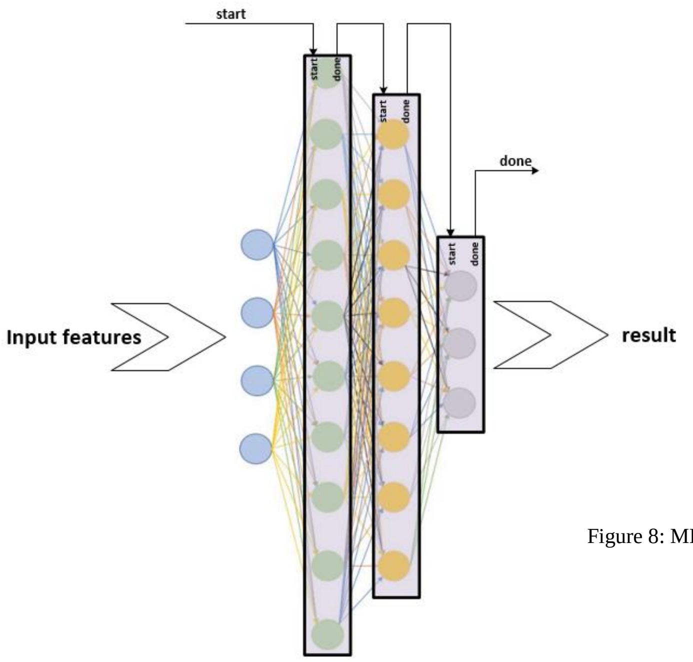


Figure 8: MLP architecture

III. Simulation Phase

In this part the design is going to be verified using a VHDL test bench. In this test bench 150 different input sets are applied to this design. In the following Figures, we can describe what is happening in this design.

First in Figure 9 you can see the computation of just one neuron. It is completely as discussed before and here we can see that the code is working.

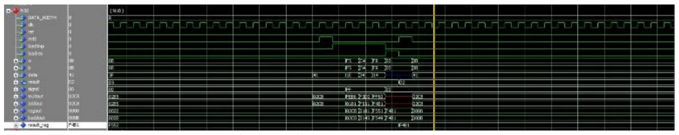


Figure 9: waveform of one neuron in first hidden layer

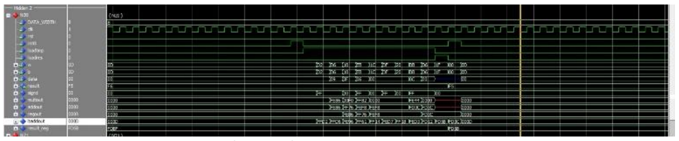


Figure 10: Waveform of one neuron in second hidden layer

In figure 11 we can see the computation and results for some input sets. As you can see according to the accuracy after training we can see that sometimes the type of the flower was not guessed correctly but some other times which are most of the times, the result is right.

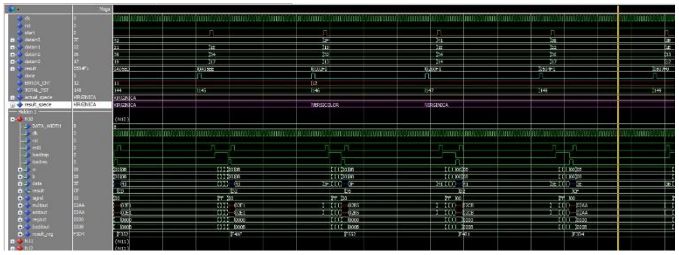


Figure 11: Waveform of multiple test sets

To check the accuracy of computation of the hardware, 150 tests are applied to this MLP and only 12 of them were guessed not right or there was not an answer. In figure 12 there is the waveform of the whole tests.

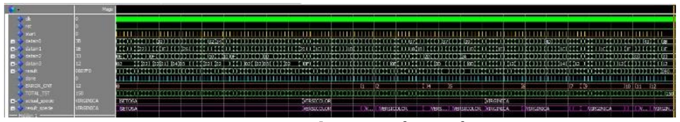
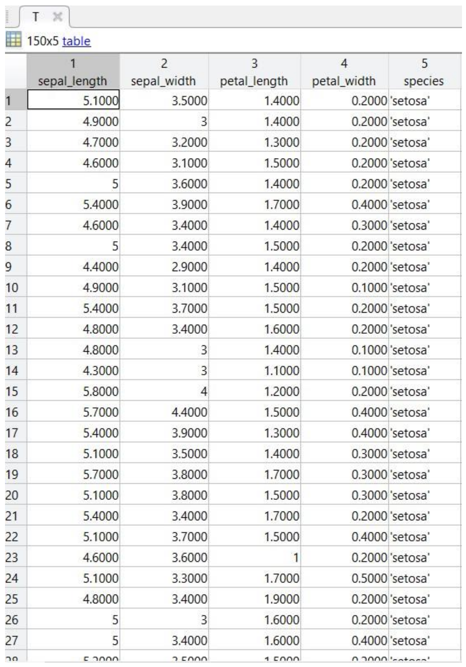


Figure 12: Complete waveform of tests

As you can see there was only 12 errors detected out of 150 test which means that the accuracy is 100 * (150-12)/150 = 92 %. So the accuracy is 92 percent and in the file containing w and b values, it is said that the accuracy is 93.3 percent and we can say that we have reached this value. This little difference is related to the approximations happened in selecting 8 bits out of the bits of each neuron.

For testing and having a good test bench some texts are provided that each have the input features and one of them have the actual results. This makes the comparison and checking much better. In figure 13 you can see a part of this test set.



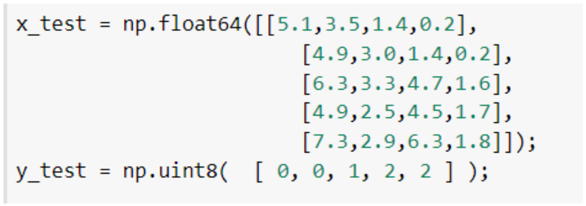
T

150x5 table

	1 sepal_length	2 sepal_width	3 petal_length	4 petal_width	5 species
1	5.1000	3.5000	1.4000	0.2000	'setosa'
2	4.9000	3	1.4000	0.2000	'setosa'
3	4.7000	3.2000	1.3000	0.2000	'setosa'
4	4.6000	3.1000	1.5000	0.2000	'setosa'
5	5	3.6000	1.4000	0.2000	'setosa'
6	5.4000	3.9000	1.7000	0.4000	'setosa'
7	4.6000	3.4000	1.4000	0.3000	'setosa'
8	5	3.4000	1.5000	0.2000	'setosa'
9	4.4000	2.9000	1.4000	0.2000	'setosa'
10	4.9000	3.1000	1.5000	0.1000	'setosa'
11	5.4000	3.7000	1.5000	0.2000	'setosa'
12	4.8000	3.4000	1.6000	0.2000	'setosa'
13	4.8000	3	1.4000	0.1000	'setosa'
14	4.3000	3	1.1000	0.1000	'setosa'
15	5.8000	4	1.2000	0.2000	'setosa'
16	5.7000	4.4000	1.5000	0.4000	'setosa'
17	5.4000	3.9000	1.3000	0.4000	'setosa'
18	5.1000	3.5000	1.4000	0.3000	'setosa'
19	5.7000	3.8000	1.7000	0.3000	'setosa'
20	5.1000	3.8000	1.5000	0.3000	'setosa'
21	5.4000	3.4000	1.7000	0.2000	'setosa'
22	5.1000	3.7000	1.5000	0.4000	'setosa'
23	4.6000	3.6000	1	0.2000	'setosa'
24	5.1000	3.3000	1.7000	0.5000	'setosa'
25	4.8000	3.4000	1.9000	0.2000	'setosa'
26	5	3	1.6000	0.2000	'setosa'
27	5	3.4000	1.6000	0.4000	'setosa'
28	5.2000	3.5000	1.5000	0.2000	'setosa'

Figure 13: test sets

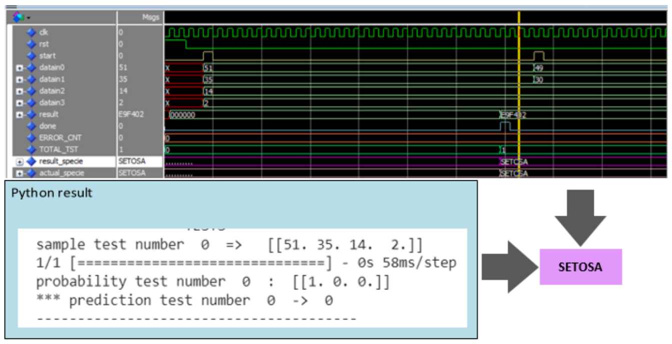
In order to test our design with python code, a set of tests are given to python code and our hardware. These tests are extracted from iris test data available on given links in reference area. In Figure 14 these tests are shown which are in python. There are 5 test sets.



```
x_test = np.float64([[5.1,3.5,1.4,0.2],
                     [4.9,3.0,1.4,0.2],
                     [6.3,3.3,4.7,1.6],
                     [4.9,2.5,4.5,1.7],
                     [7.3,2.9,6.3,1.8]]);
y_test = np.uint8(  [ 0, 0, 1, 2, 2 ] );
```

Figure 14: Tests sets in python

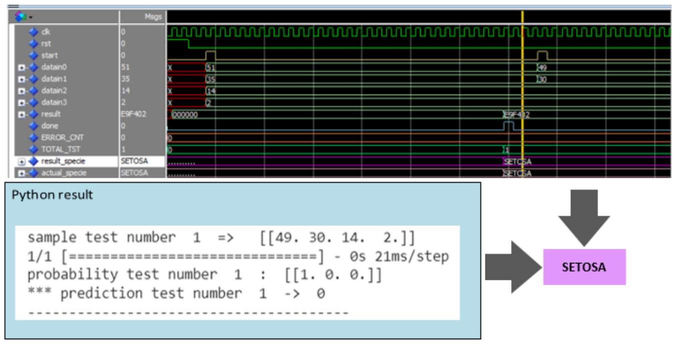
In Figure 15 you can see the first test and its result from hardware and python. The results are the same and it is Setosa.



```
sample test number  0  =>   [[51. 35. 14.  2.]]
1/1 [==============================] - 0s 58ms/step
probability test number  0  :  [[1. 0. 0.]]
*** prediction test number  0  ->  0
----------------------------------------
```

Figure 15: test number 0

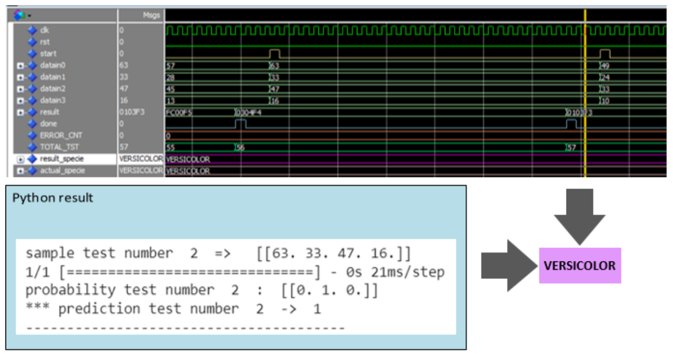
In Figure 16 you can see the next test and its result from hardware and python. The results are the same and it is Setosa.



```
sample test number  1  =>   [[49. 30. 14.  2.]]
1/1 [==============================] - 0s 21ms/step
probability test number  1  :  [[1. 0. 0.]]
*** prediction test number  1  ->  0
---------------------------------------
```

Figure 16: test number 1

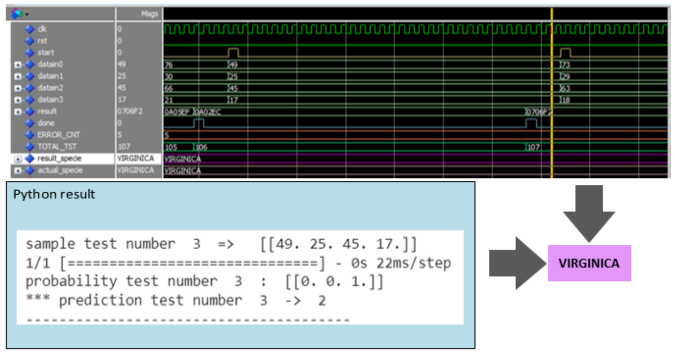
In Figure 17 you can see the next test and its result from hardware and python. The results are the same and it is Versicolor.



```
sample test number  2  =>   [[63. 33. 47. 16.]]
1/1 [==============================] - 0s 21ms/step
probability test number  2  :  [[0. 1. 0.]]
*** prediction test number  2  ->  1
---------------------------------------
```

Figure 17: test number 2

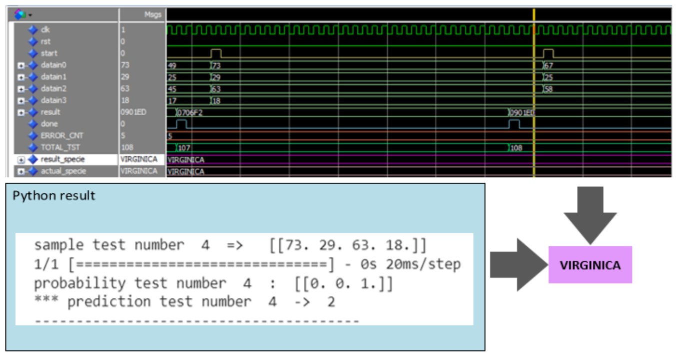
In Figure 18 you can see the next test and its result from hardware and python. The results are the same and it is Virginica.



```
sample test number  3  =>   [[49. 25. 45. 17.]]
1/1 [==============================] - 0s 22ms/step
probability test number  3  :  [[0. 0. 1.]]
*** prediction test number  3  ->  2
---------------------------------------
```

Figure 18: test number 3

In Figure 19 you can see the next test and its result from hardware and python. The results are the same and it is Virginica.

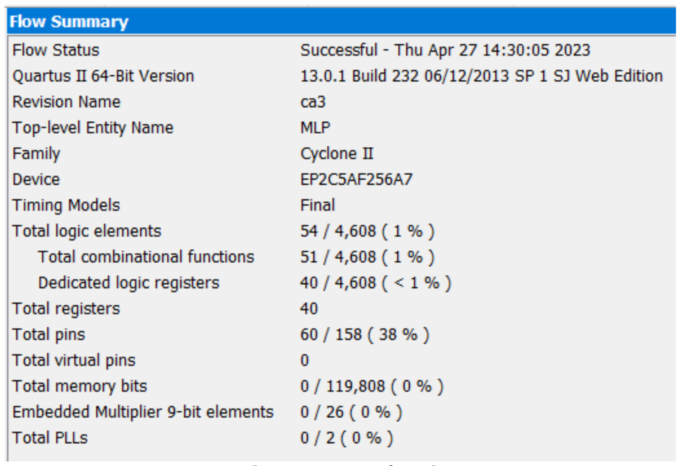


```
sample test number  4  =>   [[73. 29. 63. 18.]]
1/1 [==============================] - 0s 20ms/step
probability test number  4  :  [[0. 0. 1.]]
*** prediction test number  4  ->  2
---------------------------------------
```

Figure 19: test number 4

IV. Implementation Phase

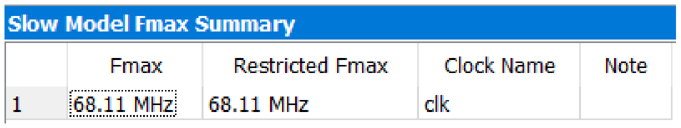
The main purpose of this part of the report is just to indicate that this VHDL code is synthesizable. Using Quartus II Synthesis tool the verified codes are synthesized nd the selected FPGA for this aim is Cyclone II EP2C5AF256A7. In Figure 20 the flow summery of this implementation is provided.



Flow Summary	
Flow Status	Successful - Thu Apr 27 14:30:05 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	ca3
Top-level Entity Name	MLP
Family	Cyclone II
Device	EP2C5AF256A7
Timing Models	Final
Total logic elements	54 / 4,608 (1 %)
Total combinational functions	51 / 4,608 (1 %)
Dedicated logic registers	40 / 4,608 (< 1 %)
Total registers	40
Total pins	60 / 158 (38 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)

Figure 20. Flow summery of implementation

The maximum frequency of this circuit is 68.11 MHz which is one of the results provided by synthesis tool and its timing analyser. In Figure 21 this result is shown.



Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	68.11 MHz	68.11 MHz	clk	

Figure 21. Maximum frequency of Circuit

By using RTL viewer we can get the schematic provided by synthesis tool after analysing our VHDL code. Figure 22 shows layers of MLP.

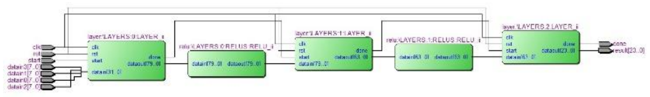


Figure 22. Connection schematic of Layers in Quartus

Figure 23 shows the schematic of datapath of a layer which is drawn by RTL viewer of synthesis tool after compiling and analysing our VHDL code.

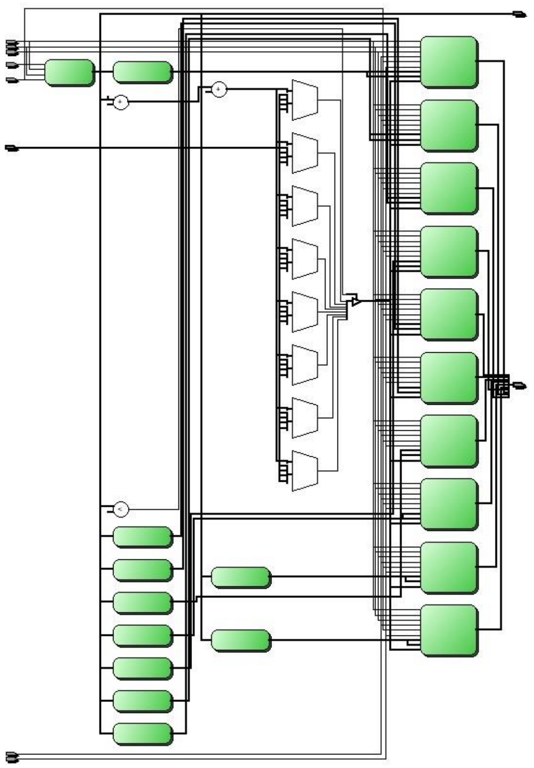


Figure 23. Datapath schematic generated in RTL viewer of Quartus

Figure 24 shows the schematic of control unit of a layer which is drawn by RTL viewer of synthesis tool after compiling and analysing our VHDL code. This schematic has also a simple state machine and it is exactly what it had to be.

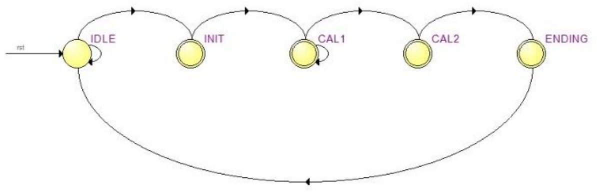


Figure 24. Controller schematic generated in RTL viewer of Quartus

V. Conclusion

In this computer assignment a circuit was designed to implement an MLP classifier. The dataset, iris.data file, contains a set of 150 records under four attributes - sepal length, sepal width, petal length, and petal width. The MLP had four inputs, two hidden layers, and three outputs. The weights based on data in the w-iris.txt file of the pre-trained network with 10 neurons for the first hidden layer and 8 neurons for the second one. The final accuracy gained in the hardware was 92 percent.

References

[1] Introduction au language VHDL (developpez.com)
[2] Data Types in VHDL (technobyte.org)
[3] The Iris Dataset (github.com)