Computer Assignment #3 – RTL Design with VHDL

Amirmahdi
Joudi,
810101325

Abstract— **In this assignment we designed an Iris MLP consists of two hidden layers and one output layer. The coefficients of the neural network are extracted. Neurons are designed and put together to make layers. Then layes make the network.**

Keywords— **MLP, neuron, later**

I. Introduction

Iris MLP is used for detection of Iris flower species: Setosa, Versicolor, and Virginica. In this assignment, a neural network from extracted coefficients is designed and data is applied to neural network. Then the output is analyzed.

II. Design

A. Neuron

The architecture of a neuron is showed in Figure 1. It consists of a multiplier which multiplies 8-bit data and weight and produces 16-bit output. The multiplier outputs are added together in a sequential form using a register. The final result is added with bias. Then this result is also registered for next steps.

B. Layer

Each layer is a set of neurons and ROMs. ROMs include wights and bias for each neuron. So each layer needs a ROM. The data for a neuron is prepared using ROMs and multiplexed input data. A counter is put for generating address. In each cycle, an address for all ROMs is generated to put proper weight or bias, in last phase, for neurons. This address also multiplexes the input data for other input of neurons. After all inputs are applied, the results go through an activation function, ReLU, and final results are put in output bus. The ReLU diagram is shown in Figure 2. Also controller and datapath of layers is shown in Figure 3 and Figure 4 respectively. After a positive edge in start input, the calculations are started. The registers and counter are initialized. Then depending on size of layers, adresses are generated and weight calculations are done. In next state, the bias addition is done. When it is done, a pulse is put in done signal.

C. MLP

This part includes different layers. The system works sequentially. The *start* signal of system is for first layer and each layer's *done* signal is the next layer's *start* signal. It means the output layer's *done* is the *done* for whole system. Also there is a bypass signal *sel* for each layer to not put output from ReLU's output. This option is good for output layer which does not need activation function. It is important to note that layers inputs are 8-bit inputs and outputs are 16-bit outputs. So the output signals have to cut down for next layer. After analyzing system, I decided to remove 2 most significant bits and keep the next 8-bits in every layer. Under this situation an accuracy of 92% is achieved. The view of this assignement is shown in Figure 5.

III. Test Bench

The coefficients for each neuron are stored in ROMs. The data for each ROM is written in a separate file. The name of each file is in this format: "layer_name & neuron_number". The layer_name is a parameter given to each layer and neuron_number is a the for loop iteration variable image. The whole systems including layer and MLP is written using GENERATE statements. For MLP, the last layer bypasses the ReLU. The code of this part is shown in Figure 6. Layers size have to put in an array of integers and pass to module as a parameter. The outputs cut down also is done using generate stattements. The inputs and also outputs are put together and made a large vector, one for input and one for output. It meand a layer with 8 neurons and 8-bit inputs, has one 64-bit inputs and one 128-bit output. The output is then cut down and becomes 64-bit for next layer.

For testing, all dataset of Iris, including 150 tests, is written in a file in proper format. Then it is read in testbench and applied to design. The output is actually three bytes, each for a flower. It means one for Setosa, one for Versicolor and one for Virginica. The most positive byte is considered as one and others as 0. The one shows the answer. The outputs are compared with actual outpus and the accuracy is calculated. The waveform of this is shown in Figure 7. There are 8 wrong answers and 4 no answers which were for equal bytes. The actual accuracy of that dataset is 93.33%. In this design with some approximation in coefficients and also data cut down between layers is as below:

$$accuracy = \frac{150 - 12}{150} * 100 = 92\ \%$$

This accuracy seems so proper for this design and goal.

NOTE: The input order is like below:

Petal.Width, Petal.Length, Sepal.Width, Sepal.Length

And the output order is like below:

Virginica, Versicolor, Setosa

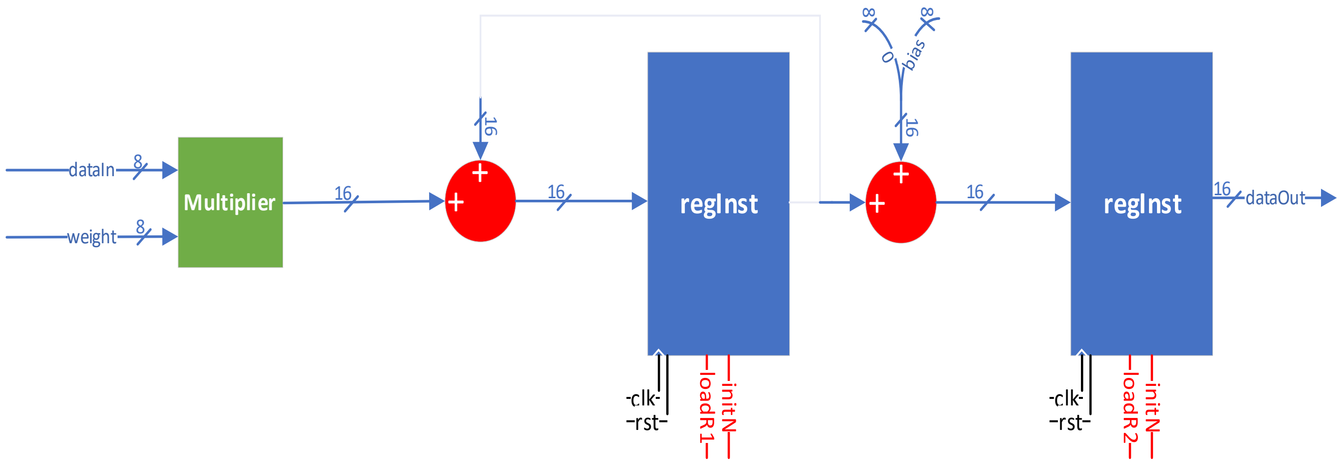


Figure 1 - neuron design

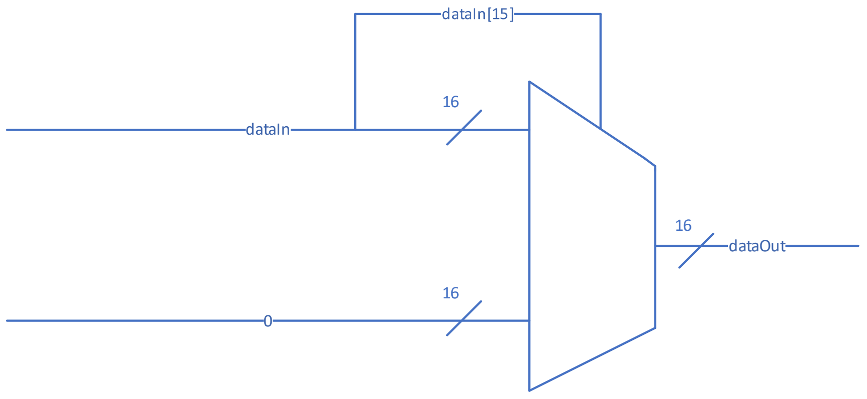


Figure 2 - ReLU function. Passes positive values and zeros non-positive values.

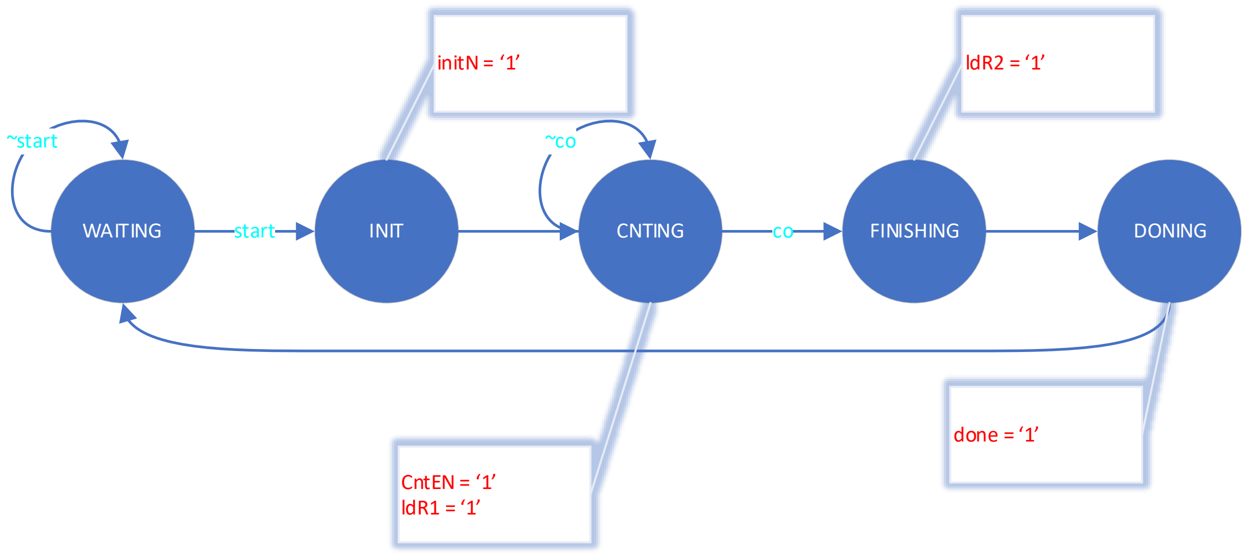


Figure 3 - layer controller

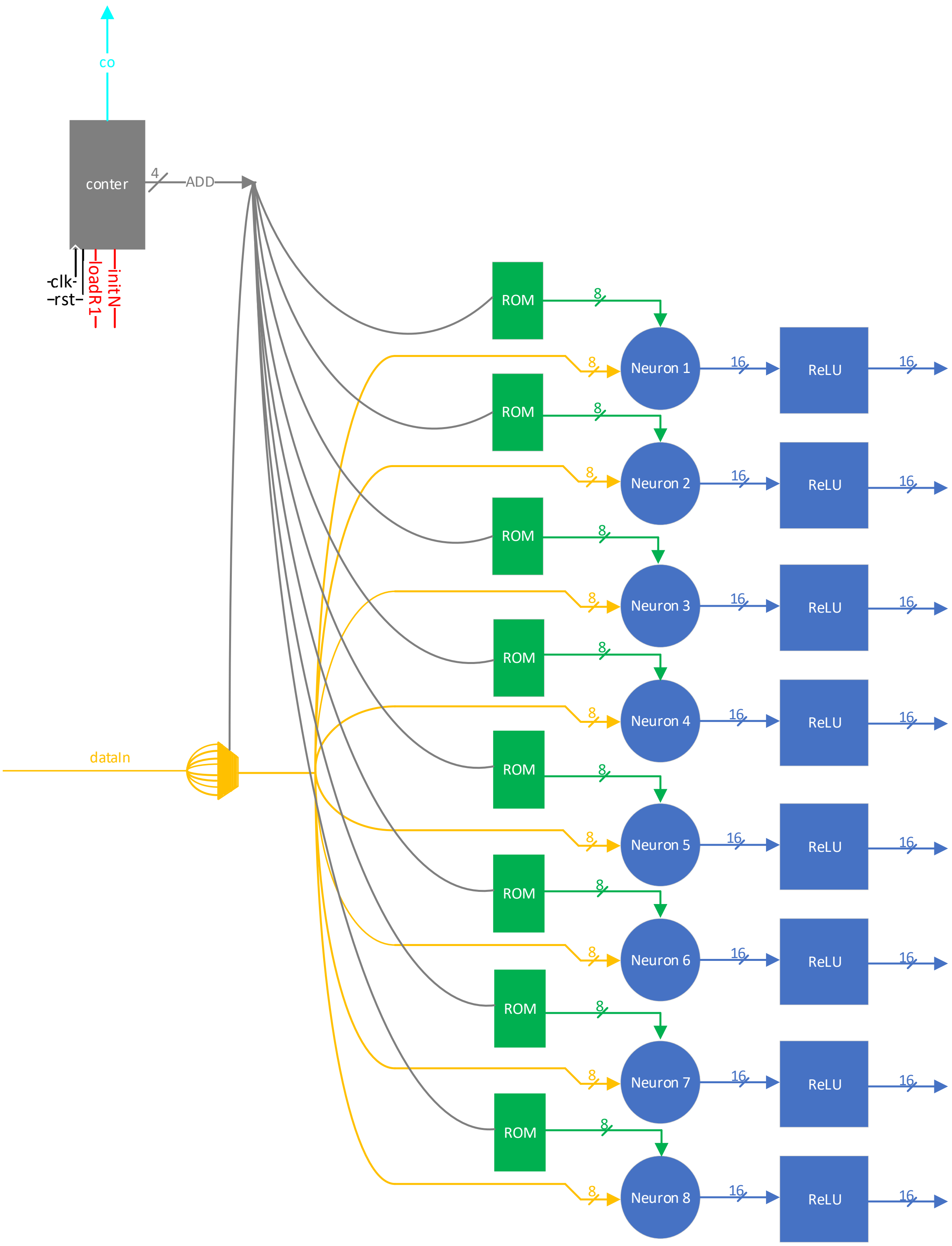


Figure 4 - an 8 neurons layer datapath with activation function

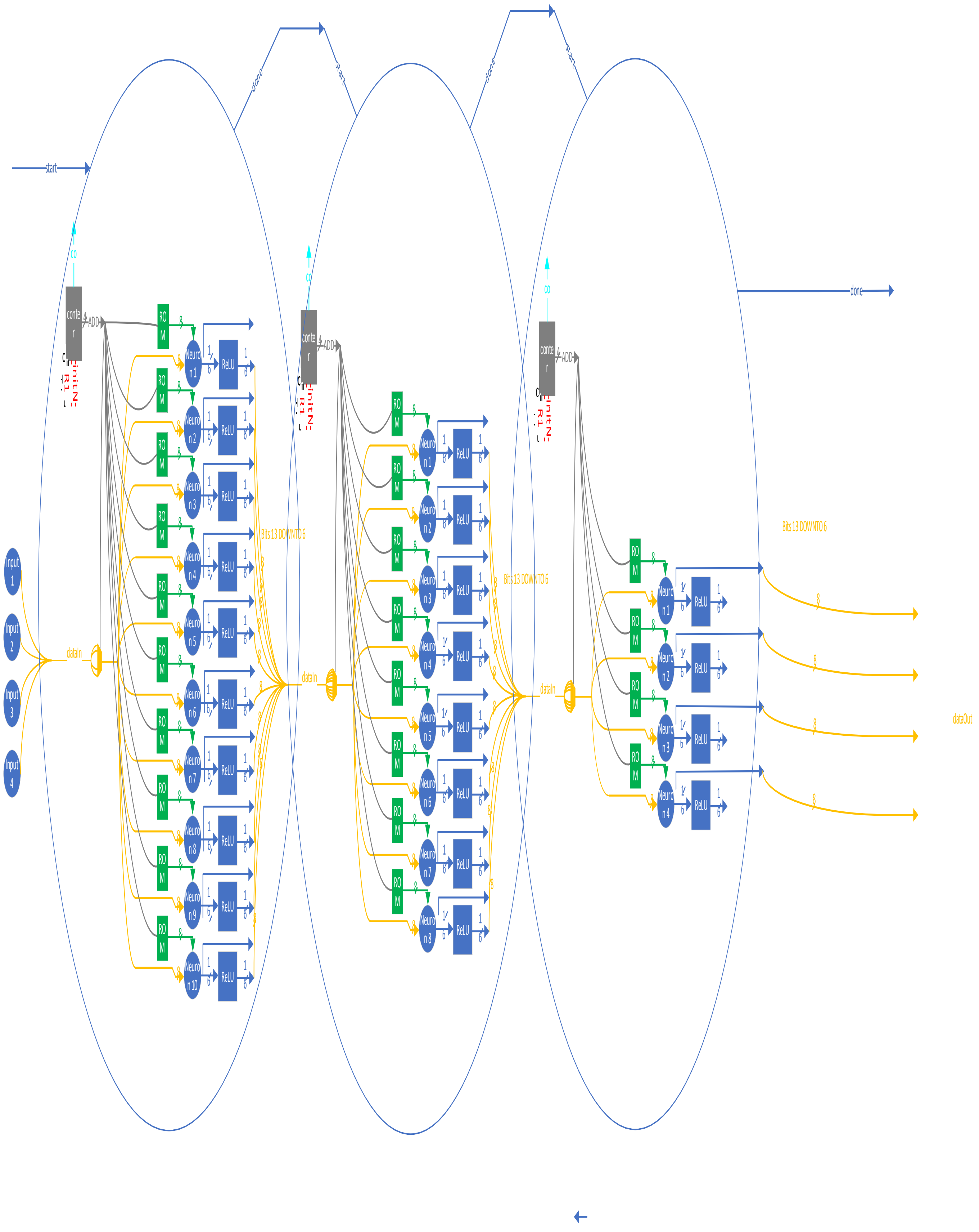
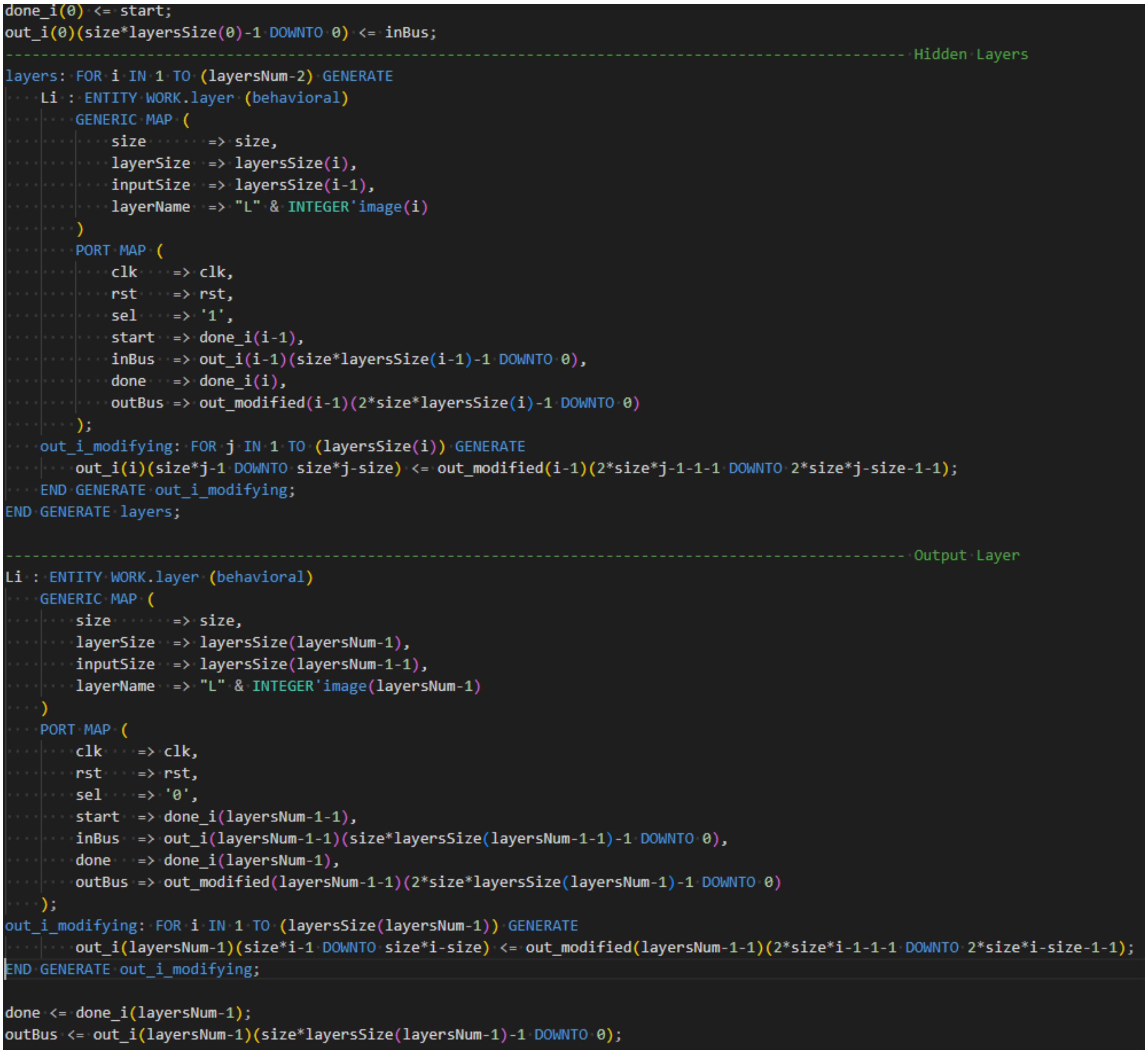


Figure 5 - MLP view of assignement 3



```vhdl
done_i(0) <= start;
out_i(0)(size*layersSize(0)-1 DOWNTO 0) <= inBus;
---------------------------------------------------------------------------------------------------- Hidden Layers
layers: FOR i IN 1 TO (layersNum-2) GENERATE
    Li : ENTITY WORK.layer (behavioral)
        GENERIC MAP (
            size       => size,
            layerSize  => layersSize(i),
            inputSize  => layersSize(i-1),
            layerName  => "L" & INTEGER'image(i)
        )
        PORT MAP (
            clk    => clk,
            rst    => rst,
            sel    => '1',
            start  => done_i(i-1),
            inBus  => out_i(i-1)(size*layersSize(i-1)-1 DOWNTO 0),
            done   => done_i(i),
            outBus => out_modified(i-1)(2*size*layersSize(i)-1 DOWNTO 0)
        );
    out_i_modifying: FOR j IN 1 TO (layersSize(i)) GENERATE
        out_i(i)(size*j-1 DOWNTO size*j-size) <= out_modified(i-1)(2*size*j-1-1 DOWNTO 2*size*j-size-1-1);
    END GENERATE out_i_modifying;
END GENERATE layers;

---------------------------------------------------------------------------------------------------- Output Layer
Li : ENTITY WORK.layer (behavioral)
    GENERIC MAP (
        size       => size,
        layerSize  => layersSize(layersNum-1),
        inputSize  => layersSize(layersNum-1-1),
        layerName  => "L" & INTEGER'image(layersNum-1)
    )
    PORT MAP (
        clk    => clk,
        rst    => rst,
        sel    => '0',
        start  => done_i(layersNum-1-1),
        inBus  => out_i(layersNum-1-1)(size*layersSize(layersNum-1-1)-1 DOWNTO 0),
        done   => done_i(layersNum-1),
        outBus => out_modified(layersNum-1-1)(2*size*layersSize(layersNum-1)-1 DOWNTO 0)
    );
out_i_modifying: FOR i IN 1 TO (layersSize(layersNum-1)) GENERATE
        out_i(layersNum-1)(size*i-1 DOWNTO size*i-size) <= out_modified(layersNum-1-1)(2*size*i-1-1 DOWNTO 2*size*i-size-1-1);
END GENERATE out_i_modifying;

done <= done_i(layersNum-1);
outBus <= out_i(layersNum-1)(size*layersSize(layersNum-1)-1 DOWNTO 0);
```

Figure 6 - MLP generic parts

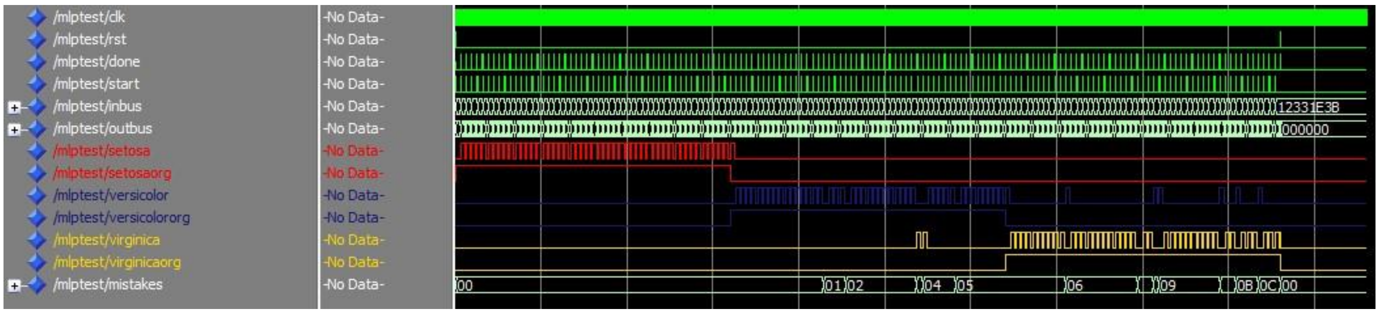
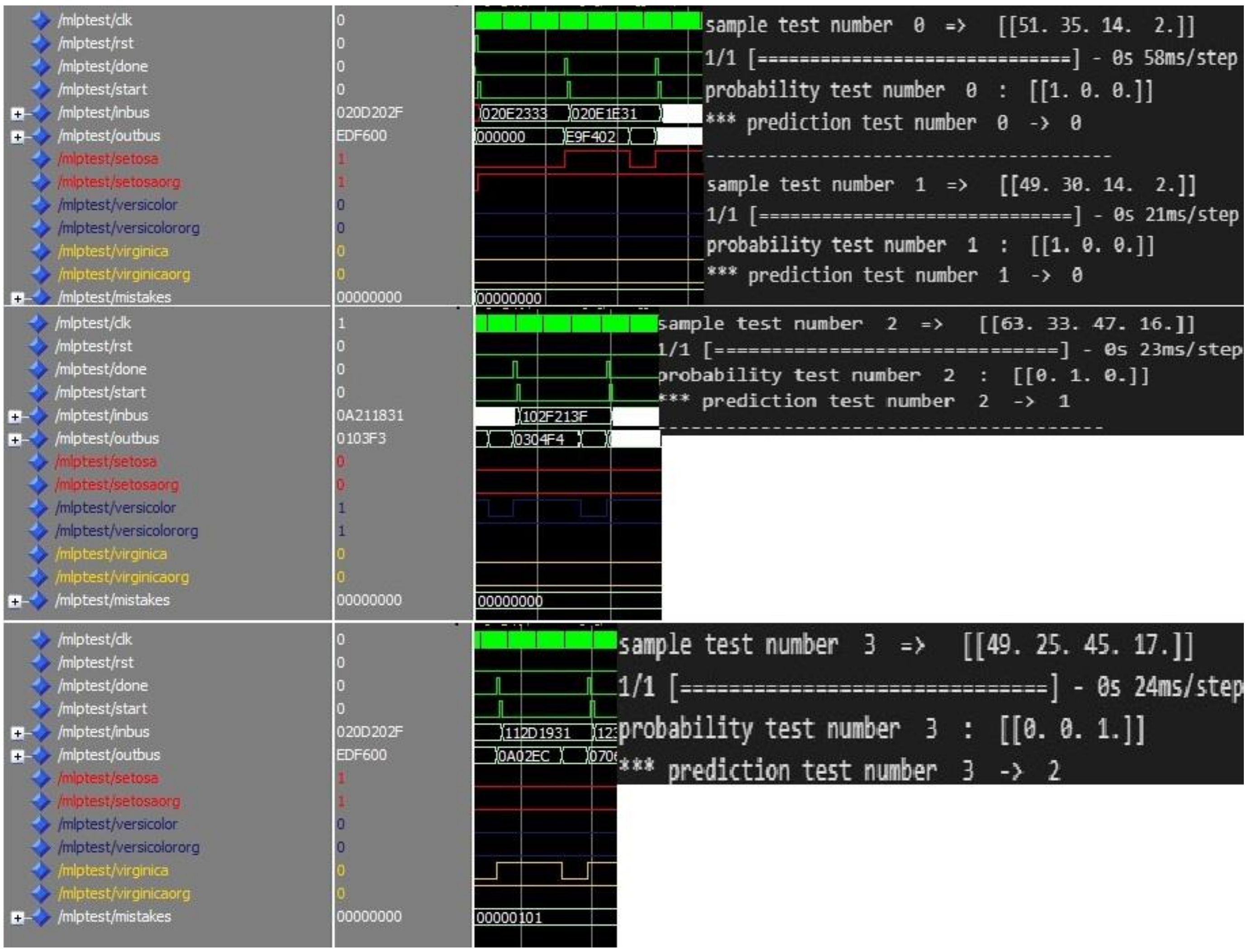

/mlptest/clk -No Data-
/mlptest/rst -No Data-
/mlptest/done -No Data-
/mlptest/start -No Data-
/mlptest/inbus -No Data-
/mlptest/outbus -No Data-
/mlptest/setosa -No Data-
/mlptest/setosaorg -No Data-
/mlptest/versicolor -No Data-
/mlptest/versicolororg -No Data-
/mlptest/virginica -No Data-
/mlptest/virginicaorg -No Data-
/mlptest/mistakes -No Data-
12331E3B
000000
00
01
02
04
05
06
09
0B
0C
00

Figure 7 - Results waveforms



```
/mlptest/clk                0
/mlptest/rst                0
/mlptest/done               0
/mlptest/start              0
/mlptest/inbus              020D202F
/mlptest/outbus             EDF600
/mlptest/setosa             1
/mlptest/setosaorg          1
/mlptest/versicolor         0
/mlptest/versicolororg      0
/mlptest/virginica          0
/mlptest/virginicaorg       0
/mlptest/mistakes           00000000
```

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

```
/mlptest/clk                1
/mlptest/rst                0
/mlptest/done               0
/mlptest/start              0
/mlptest/inbus              0A211831
/mlptest/outbus             0103F3
/mlptest/setosa             0
/mlptest/setosaorg          0
/mlptest/versicolor         1
/mlptest/versicolororg      1
/mlptest/virginica          0
/mlptest/virginicaorg       0
/mlptest/mistakes           00000000
```

```
sample test number  2  =>   [[63. 33. 47. 16.]]
1/1 [==============================] - 0s 23ms/step
probability test number  2  :  [[0. 1. 0.]]
*** prediction test number  2  ->  1
---------------------------------------
```

```
/mlptest/clk                0
/mlptest/rst                0
/mlptest/done               0
/mlptest/start              0
/mlptest/inbus              020D202F
/mlptest/outbus             EDF600
/mlptest/setosa             1
/mlptest/setosaorg          1
/mlptest/versicolor         0
/mlptest/versicolororg      0
/mlptest/virginica          0
/mlptest/virginicaorg       0
/mlptest/mistakes           00000000
```

```
sample test number  3  =>   [[49. 25. 45. 17.]]
1/1 [==============================] - 0s 24ms/step
probability test number  3  :  [[0. 0. 1.]]
*** prediction test number  3  ->  2
```

Figure 8 - Testbench and Python comparison

IV. Conclusions

MLP implementation is very important. There are several approaches for this purpos or optimizing the network. This was a sequential approach done for MLP in this assignments.

References

rpubs.com

kaggle.com