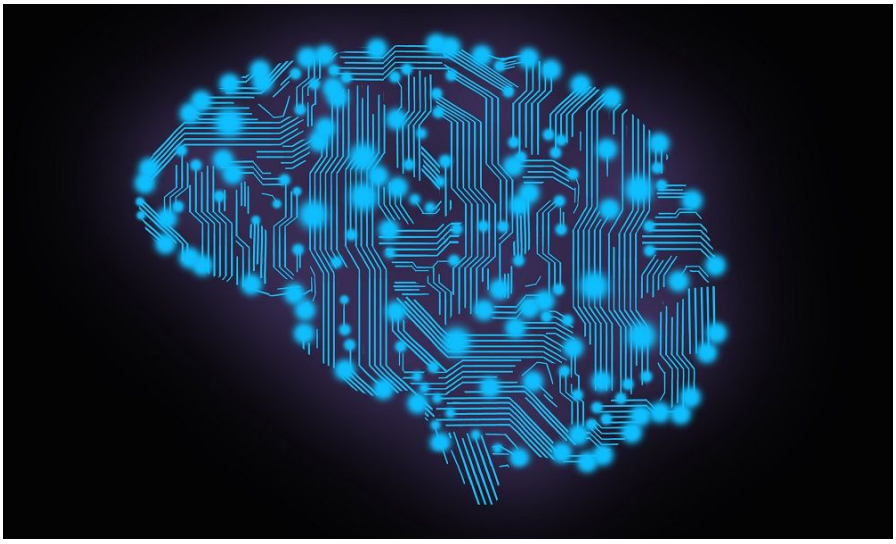
Neuron Model - an FPGA approach

EE2702 - Digital Circuits Lab Project Report



Arko Sharma

15.11.2017

INTRODUCTION

In this project , the design of a single simplified neuron which contains a unit activation function was suggested and implemented using the Zybo board.

The endeavour of this neuron design was to explore the challenges that arise in developing a brain computer interface at a simplistic level and utilise the parallel processing prowess of the FPGA in order to implement a design that was best suited towards it. Furthermore, its goal is design a neural network with any number of hidden layers with a different number of this neuron and any number of output neurons, in easy manner using FPGA.

The core of this project deals with a single neuron model that should be able to learn one of the six basic combinational logic functions (AND ,OR ,NOT,XOR,XNOR,NAND,NOR) . The neuron model developed was , in reality a digital circuit which was pre-trained with a few logic functions and upon being fed with input vectors and labels , could classify them as to belonging with the correct set of logic function. The circuit would function as per this logic and would learn to deactivate the logic functions which were found to be false for a particular set of input vectors.

THEORY

Neural networks have been used broadly in many fields, either for software or for research. They are extensively used to solve a great variety of problems that are difficult to be resolved by other methods. ANN has been used in diverse applications in science and engineering. Although, neural networks have been implemented mostly in software, hardware versions are gaining relevance ; the launch of the recent iphoneX , which deploys state of the art *face recognition* techniques in real time being an appropriate exemplar of the fact. Software versions dominate in terms of being easy to implement, but with poor performance. Hardware versions are generally more difficult and time-ineffiecient to implement, but with better performance than software. As we know with rapid increase in demand of digital circuit in industry , there is aneed to launch the products quickly without sacrificing integrated circuit (IC) quality, so testing of digital circuit in Very Large Scale Integration (VLSI) has become crucial . To deal with these and many such related Engineering tasks, acceleration of the conventionally used machine learning and Artificial Intelligence algorithms known to us are gaining importance. It has already been forecasted that the next species to rule over the earth would be the one with the maximum amount of intelligence - be it natural or artificial and therefore the task of accelerating brain computer interface may have a crucial say in evolution of mankind and uses of FPGAs to accelerate these endeavours are gaining importance.

The Model

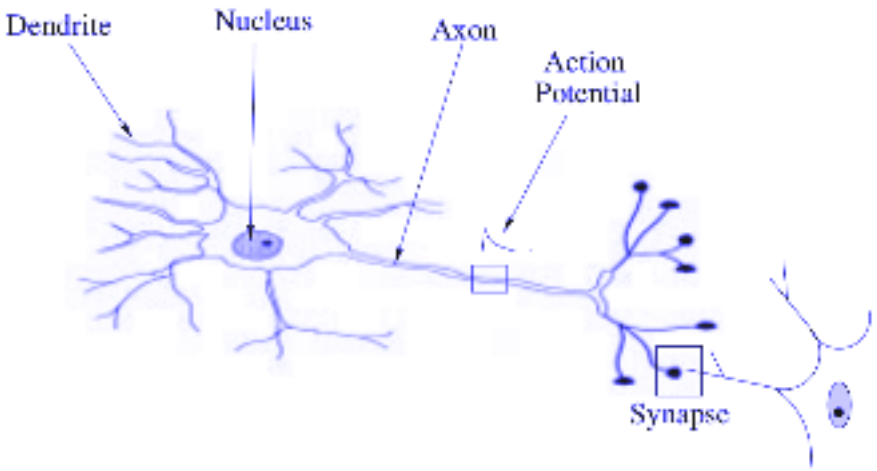


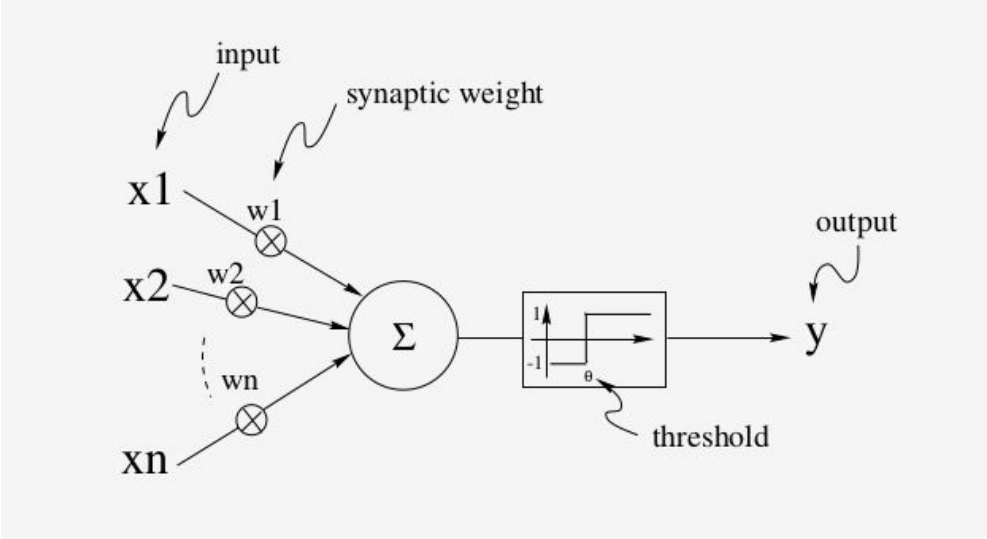
Figure showing the biological configuration of a neuron.

Although the brain exhibits a great diversity of neuron shapes, dendritic trees, axon lengths, etc., all neurons seem to process information in much the same way. Information is transmitted in the form of electrical impulses called action potentials via the axons from other neuron cells. Such action potentials have an amplitude of about 100 millivolts, and a frequency of approximately 1 KHz. When the action potential arrives at the axon terminal, the neuron releases chemical neurotransmitters which mediate the interneuron communication at specialized

connections called synapses.

The McCulloch-Pitts model

The computational model of the neuron presented by McCulloch and Pitts is binary and operates in discrete time. The output of a neuron is y (t) = 1 when an action potential is generated, and y (t) = 0 otherwise. A weight value w(i) is associated to each ith connection to the neuron. Such weights characterize the synapses as excitatory if w (i) > 0, and inhibitory if w(i) < 0. A neuron fires when the effect of inhibitions and excitations is larger than a certain threshold (see gure below).



The Perceptron

In 1958, the American psychologist Frank Rosenblatt proposed a computational

model of neurons he called the perceptron . The essential innovation was the introduction of numerical interconnection weights instead of simple inhibitory/excitatory connections as in the McCulloch-Pitts model . The weights are typically real values. If the presence of a value x(i) in a given input pattern tends to fire the perceptron, the corresponding weight w(i) will be positive. If the value x i inhibits the perceptron, the weight w i will be negative. Such weights model synaptic Efficacity.

Threshold logic

The weights and threshold of a McCulloch Pitts neuron can be easily set to realize the logic functions. In this project , the 6 basic logic functions as described above are trained upon the perceptron . While the McCulloch-Pitts model no longer plays an important role in computational neuroscience , it is still widely used in neural computation (i.e., the technology based on networks of neuron-like" units), especially when it is generalized to continuous inputs and outputs . The computational capabilities described so far lead us to conclude that threshold logic units and perceptrons are computational units very well suited for **classification** tasks. This Classification is achieved by realizing decision boundaries, which in turn are weight dependent.

In order to determine such weights in order to separate properly two or more classes in the space of inputs , we follow the algorithm of supervised learning model.

Supervised learning

In supervised learning or learning with a “teacher", the training inputs are provided with the desired outputs. A basic principle of this kind of learning algorithms is error correction , that is, an error value is generated from the actual response of the network and the desired response, then, the weights are modified such that the error is gradually reduced.

IMPLEMENTATION

Completion of training process takes place when the model retains the functionality only of the learnt logic and eliminates all other functions.
To simplify the model an advanced layer of logic functions is provided to begin with, instead of the model manually learning the logic from the core . This idea can be easily be extended to any number of complicated functions with any number of input variables, given the appropriate amount of computational requirements in terms of hardware and memory.

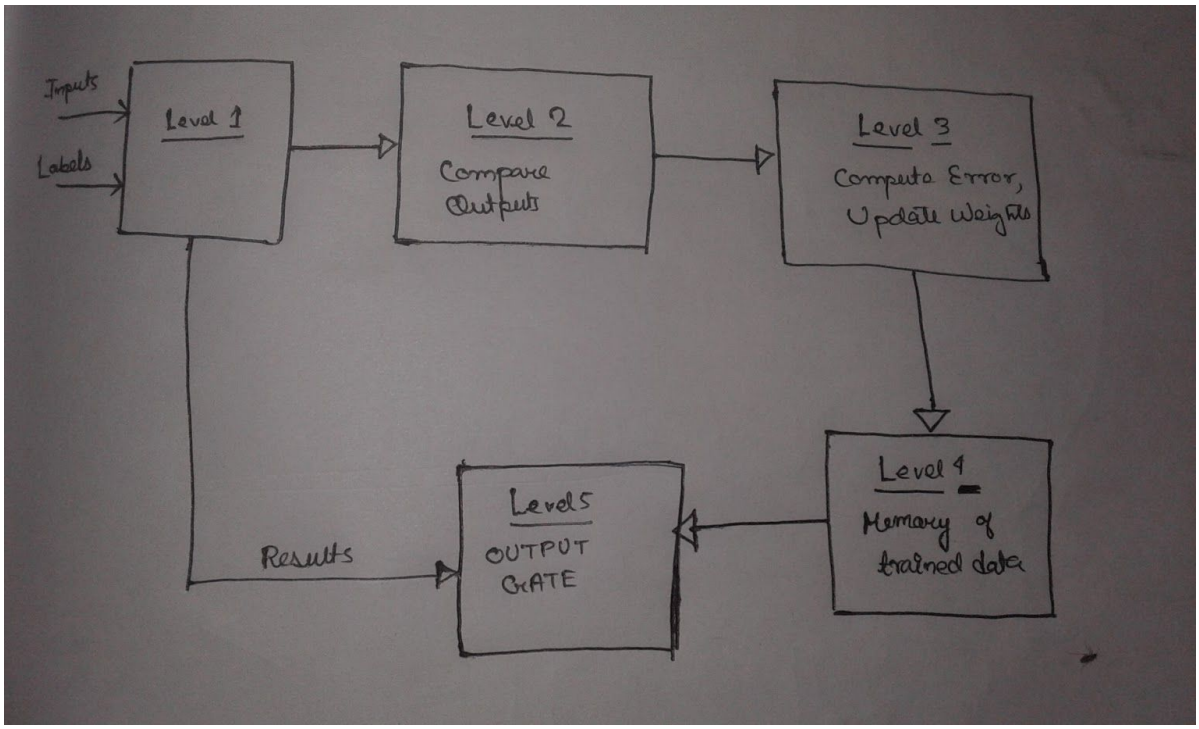
In the circuit, LEDs are used to represent the states of various logic gates. Upon giving a particular input set, the gates may be on or off depending upon the weights for that gate.

The error is calculated in two steps- first by determining the true output of a gate for a given input and then comparing it to the “expected output ” (variable “y_obs” in code).

One layer is used by this model to compute the error after being given the inputs and the corresponding outputs, using this error through the unit activation functions to determine the weights of successive iterations of training procedure. The threshold value is simply binary 0 or 1 , meaning either the response corresponding to a particular gate can be either ON or OFF. Finally , the gate which remains ON determines the correct output and the weights of other gates are set to zero.
To achieve this , D flip flops are used as memory of the circuit, in order to preserve the zero weights of the eliminated gates. Now when further training samples are provided , the zero - weighted gates do not alter their states ie, their weights are permanently set to 0.
The activation function thus uses the binary threshold logic discussed above. In Mathematical terms, the binary threshold logic would divide the plane into decision surfaces of linear or nonlinear type, but since here a few pre computed gates are also present , therefore this does not pose a difficulty as the non linear XOR function is already computed and it only suffices to match the outputs to classify the input functions.
It is clear that the gate which survives in the end would be the correct gate in question and the circuit would be able to classify (rather declassify in this case) the incorrect gates in order to depict the correct functionality.

LOGIC

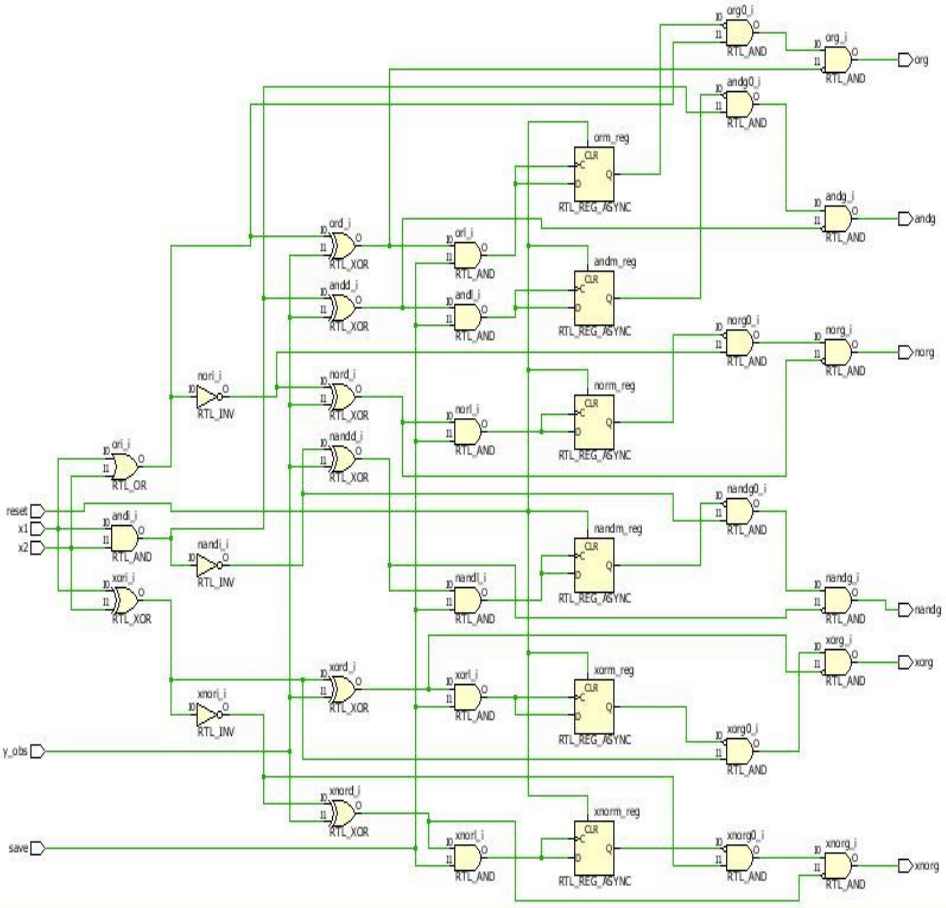
The circuit can be described using a block diagram shown below :

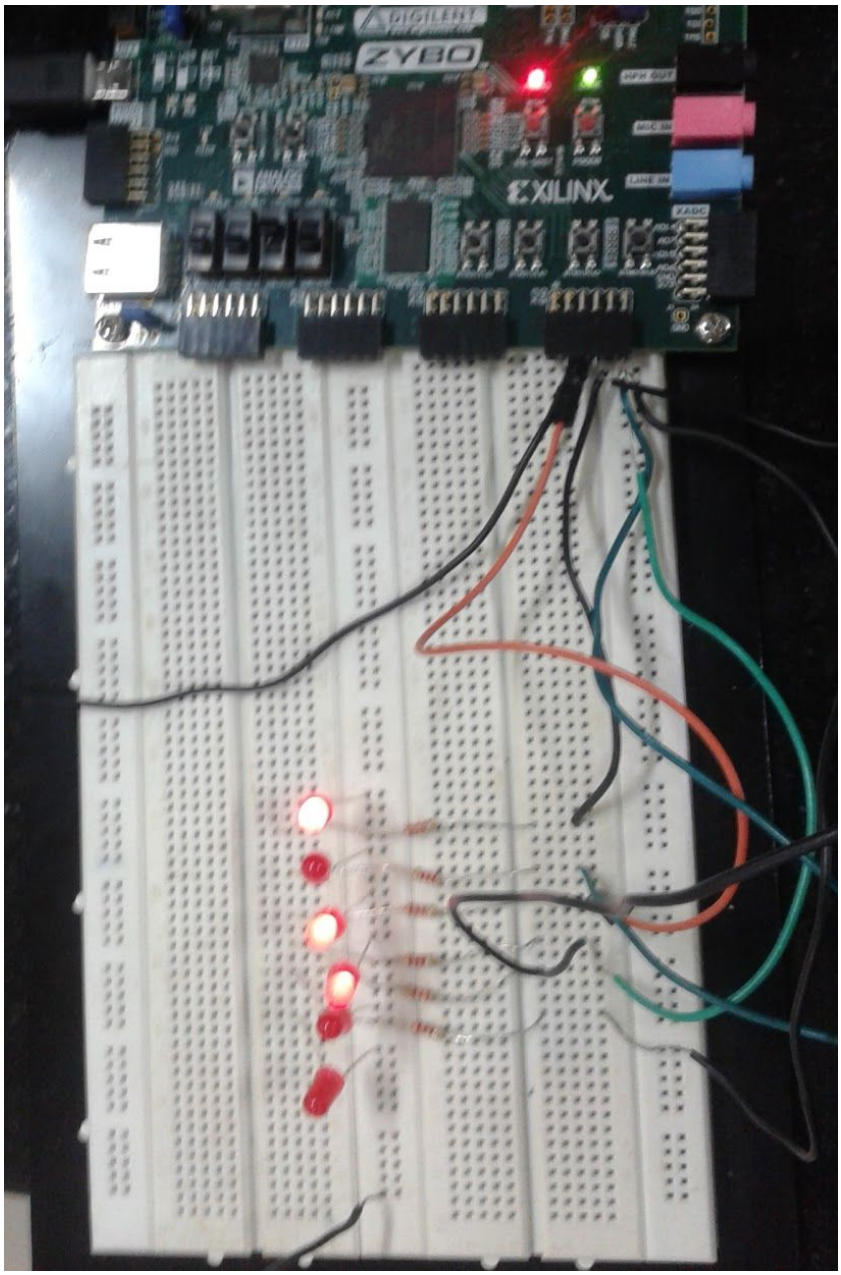


OBSERVATIONS AND RESULTS

4 combinations (00,01,10,11) of inputs along with their outputs as labels were provided to the model for every gate and the circuit classified these into the correct logic function, making of all other gates unresponsive. Pictures of the circuit trained with NAND logic are shown below.

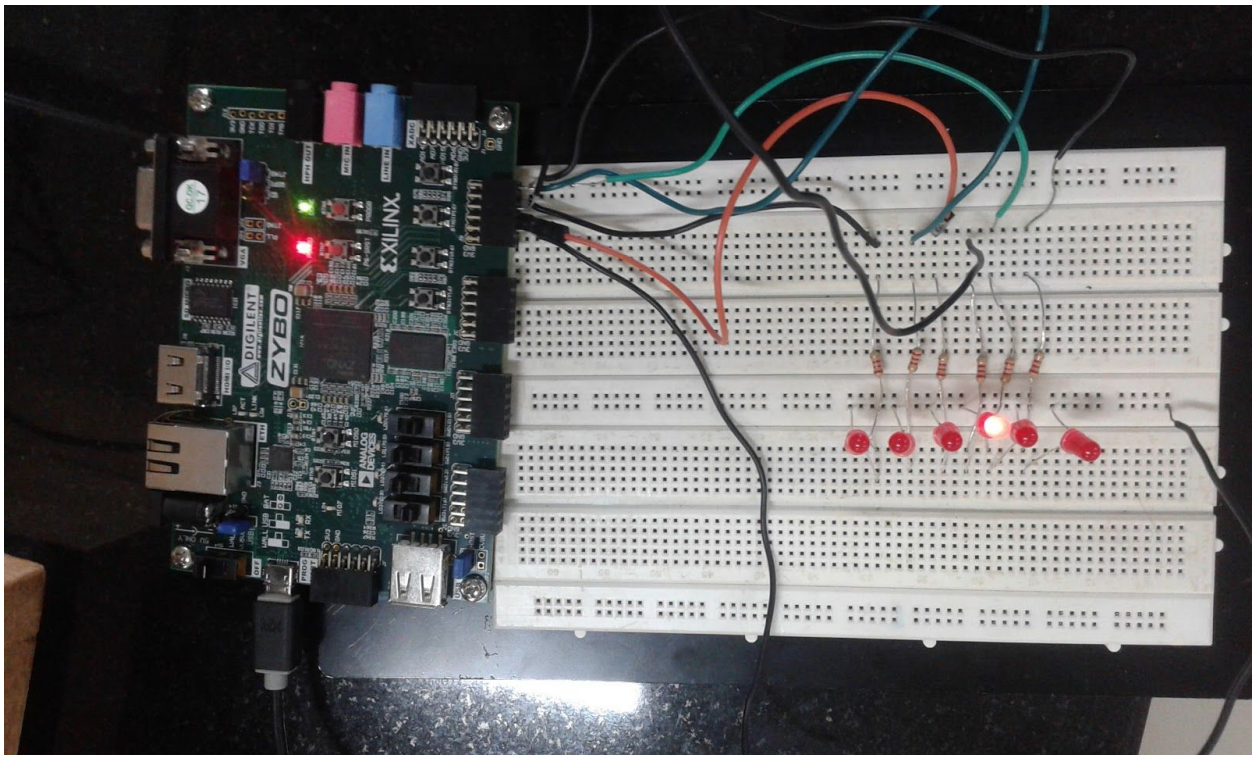
Following images show the RTL schematic and hardware realization on Zybo :





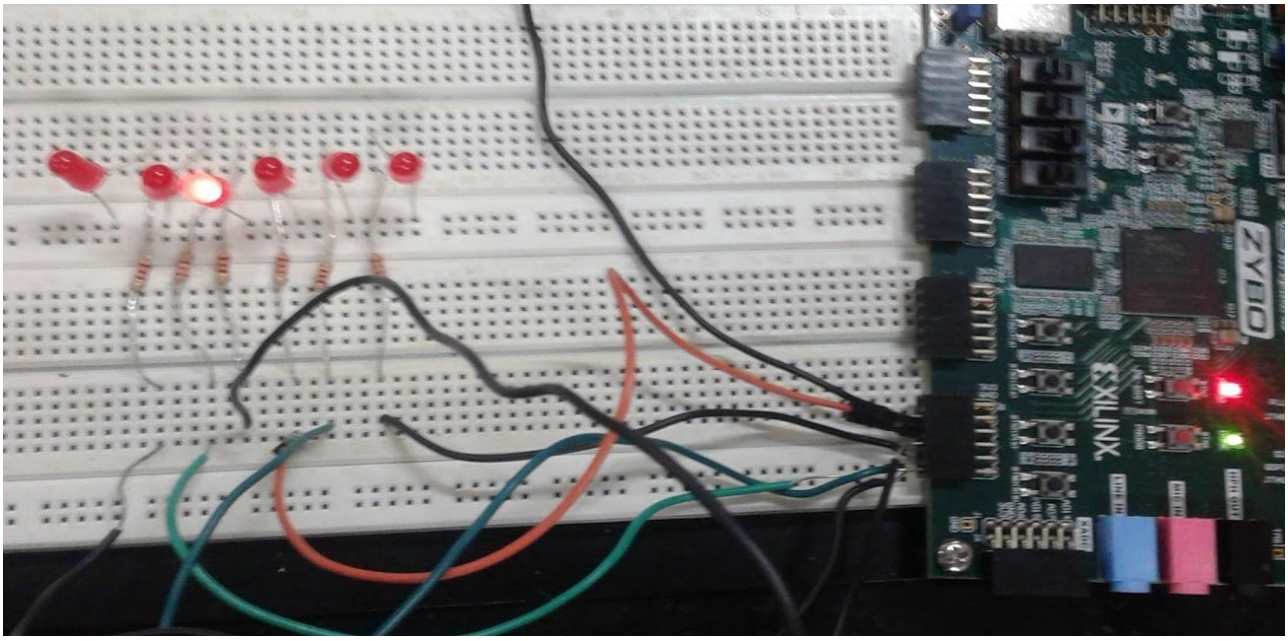
First step of training. As shown , the inputs are 0,0 and the output “observed” is given as 1. So initially 3 gates are possible - NOR, NAND or XNOR. The 3 lit LEDs signify these gates.

Rest of the 3 gates are eliminated here as their outputs are low they will not change as per the connections of the D Flip Flop as shown above.



In the second step an input of 0,0 and the corresponding “ y_obs “ of 1 is provided . Technically the training is complete as all the other gates will now be eliminated and only NAND gate would remain.

To test this , we re - enter the inputs of the first training set (0,0).



As shown above, only the NAND gate is still functional .

HDL Code

```
// Create Date: 11/10/2017 02:07:47 PM


module neuron(
```

```
input x1,

input x2,

input y_obs,

input save,

input reset,

output andg,

output org,

output xorg,

output xnorg,

output norg,

output nandg




);


//level 1 -- storing the actual inputs

wire andi;

wire ori;

wire nandi;

wire nori;
```

```
wire xori;

wire xnori;


assign andi=x1 & x2;

assign ori=x1 | x2;

assign nandi=~(x1 & x2 );

assign nori=~(x1|x2);

assign xori = (x1 ^ x2);

assign xnori = ~(x1 ^ x2);


//level 2 -- check which outputs are correct

//          ie compare with desired output


wire andd;

wire ord;

wire nandd;

wire nord;

wire xord;

wire xnord;
```

```
assign andd=   andi^ y_obs;

assign ord=     ori^ y_obs;

assign nandd= nandi^ y_obs;

assign nord=   nori^ y_obs;

assign xord =  xori^ y_obs;

assign xnord =xnori^ y_obs;



// level 3- learning level

// here the error in the ouputs are computed

// in this model the weights are all 1 ( provided by "save" signal)

// if the output is wrong,then this error is used to "turn off " the particular gate

//          activation function is just the "and" operation with "save" signal

//  if the output is low after this then threshold is not obtained and the gate never works
again

//  so the "save" signal is used to  "remember" the wrong gates.



// l for learning



wire andl;

wire orl;

wire nandl;
```

```
wire norl;

wire xorl;

wire xnorl;



assign andl=   andd& save;

assign orl=      ord& save;

assign nandl= nandd& save;

assign norl=   nord& save;

assign xorl =  xord& save;

assign xnorl =xnord& save;




// level 4 - memory

// the outputs at various stages are stored and if they are 1 at the end then this is the required gate

// if the output in memory is low, then this gate was turned off as some input of it was wrong



// m for memory

reg andm;

reg orm;
```

```
reg nandm;

reg norm;

reg xorm;

reg xnorm;


always @ ( posedge andl or posedge reset)

if(reset)

begin

andm <= 1'b0;

end

else

begin

andm <= andl;

end


always @ ( posedge orl or negedge reset)

if(reset)

begin

orm <= 1'b0;

end
```

```
else

begin

orm <= orl;

end


always @ ( posedge nandl or posedge reset)

if(reset)

begin

nandm <= 1'b0;

end

else

begin

nandm <= nandl;

end


always @ ( posedge norl or posedge reset)

if(reset)

begin

norm <= 1'b0;

end
```

```
else
begin
norm <= norl;
end
always @ ( posedge xorl or negedge reset)
if(reset)
begin
xorm <= 1'b0;
end
else
begin
xorm <= xorl;
end
always @ ( posedge xnorl or negedge reset)
if(reset)
begin
xnorm <= 1'b0;
end
else
begin
```

```
        xnorm <= xnorl;
        end

        // level 5 - output stage

        // logic for output -- the gate should be on
        //              -- the actual output ( from level 1) is displayed
        //              -- the y_obs must be kept high (for completing circuit)

        assign andg =   ~(andm)  & andi & (~andd);
        assign org  =   ~(orm) & (ori) & (~ord);
        assign nandg=~(nandm)&(nandi)   &(~nord);
        assign norg =  ~(norm) &(nori) & (~nord);
        assign xorg =  ~(xorm)   & (xori) &(~xord);
        assign xnorg= ~(xnorm)   & (xnori) & (~xnord);
endmodule
```

Constraints

```
set_property -dict { PACKAGE_PIN T20 IOSTANDARD LVCMOS33 } [get_ports { andg }];
```

```
set_property -dict { PACKAGE_PIN U20 IOSTANDARD LVCMOS33 } [get_ports { org }];

set_property -dict { PACKAGE_PIN V20 IOSTANDARD LVCMOS33 } [get_ports { nandg }];

set_property -dict { PACKAGE_PIN W20 IOSTANDARD LVCMOS33 } [get_ports { norg }];

set_property -dict { PACKAGE_PIN Y18 IOSTANDARD LVCMOS33 } [get_ports { xorg }];

set_property -dict { PACKAGE_PIN Y19 IOSTANDARD LVCMOS33 } [get_ports { xnorg }];

set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { x1 }];

set_property -dict { PACKAGE_PIN W13 IOSTANDARD LVCMOS33 } [get_ports { x2 }];

set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { y_obs }];

set_property -dict { PACKAGE_PIN Y16 IOSTANDARD LVCMOS33 } [get_ports { save }];

set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { reset }];
```

CONCLUSION

The project presented an extremely simplified version of a neuron model but nevertheless provided an exciting opportunity to explore the frontiers of hardware developments in the field of Artificial Intelligence. The most difficult part of implementing an actual neural network in hardware is the realization of a reasonably accurate activation function. This project simply used a unit activation but the in the process of developing the project I came across numerous methods of approximating crucial functions such as the Sigmoid . The advancements in computing prowess in recent times presents the perfect incentive for following through with the ideas of this project, which form the roots of hardware based Machine Learning algorithms, especially as these are even being implemented extensively in the present day in both industry as well as household levels.

REFERENCES

1. *M.A. Arbib. Part I: Background. In Michael A. Arbib, editor, Handbook of Brain Theory and Neural Networks , page 11. MIT Press, 1995.*
2. *A.E. Alpaydin. Neural Models of Incremental Supervised and Unsupervised Learning . PhD thesis, Swiss Federal Institute of Technology, Lausanne, 1990 These 863.*