ARTIFICIAL NEURAL NETWORK (ANN)
ON
FIELD PROGRAMMABLE GATE ARRAY(FPGA)

JORGE L. ALBERTO

Overview

What is an FPGA?

What is an ANN?

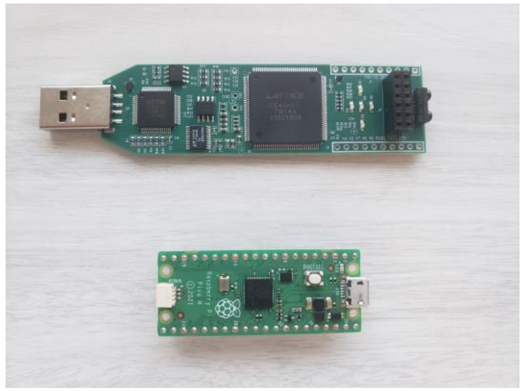
Why implement ANNs on FPGAs?

ANN on an FPGA

What is an FPGA?

General, Reconfigurable, & Fast Integrated Circuits

Field
Programmable
Gate
Array



	Microcontroller	ASIC	FPGA
General	✓		✓
Reconfigurable			✓
Speed		✓+	✓
Power Consumption	✓	✓	
Cost (Low Volumes)	✓+		✓
Cost (High Volumes)	✓	✓	

Reconfigurable – circuit components may change

What is an FPGA?

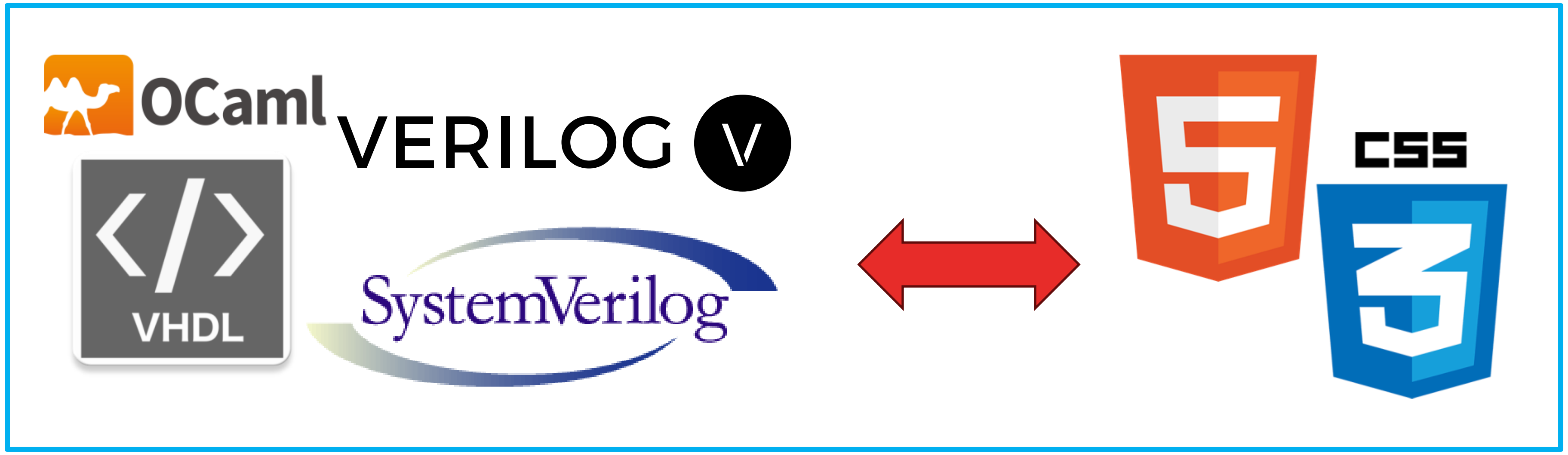
Describe Functionality of Circuit

Field

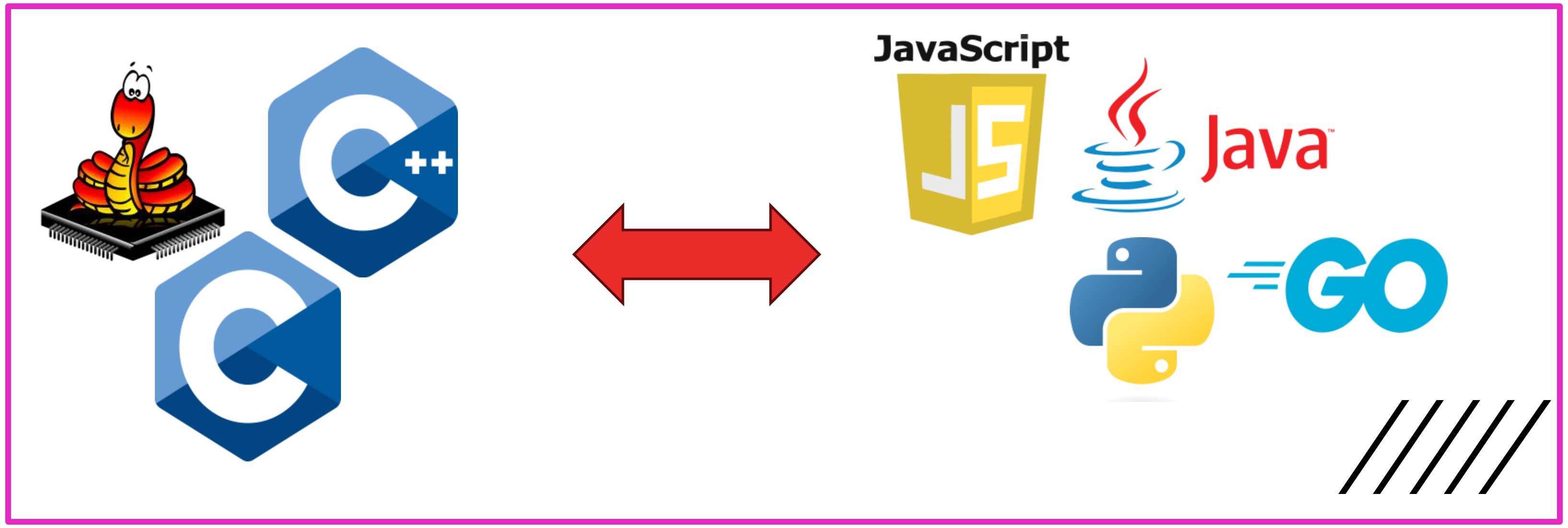
Programmable

Gate

Array



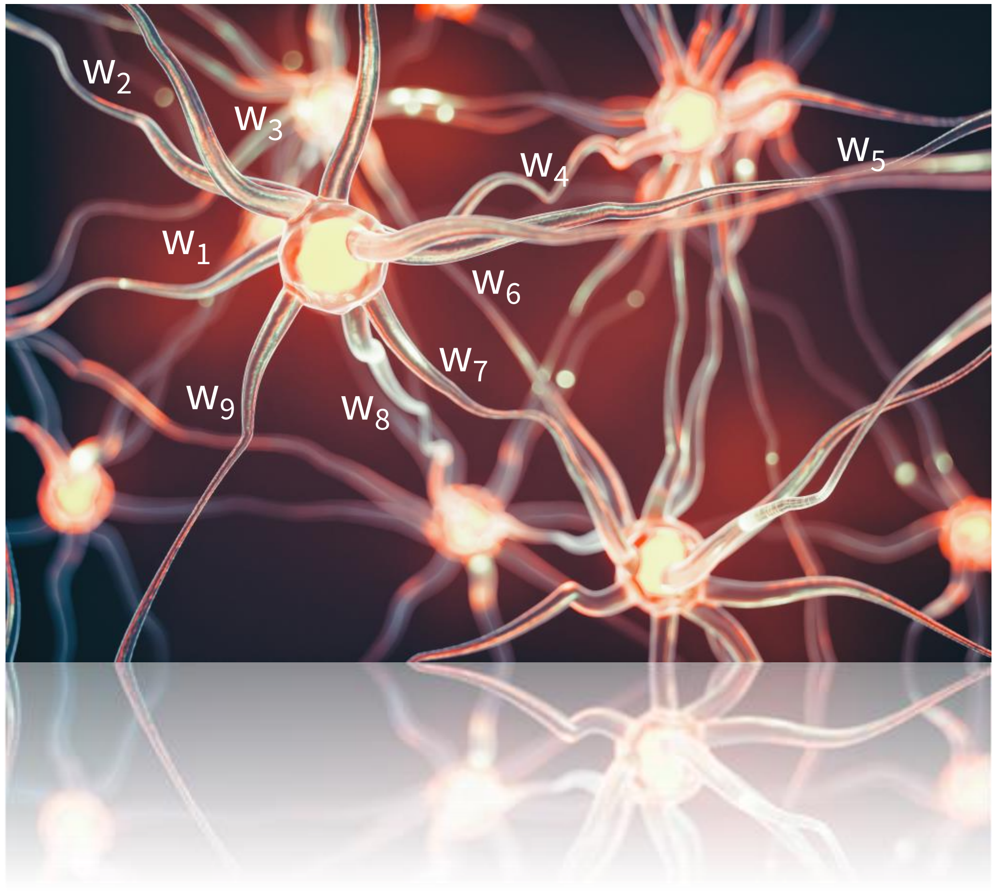
- Describe Circuit/Website Layout
- Implement functionality based off circuit/website layout



What is an ANN?

Neuron

Artificial
Neural
Network



$$y = f(x)$$

$$y = w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

What is an ANN?

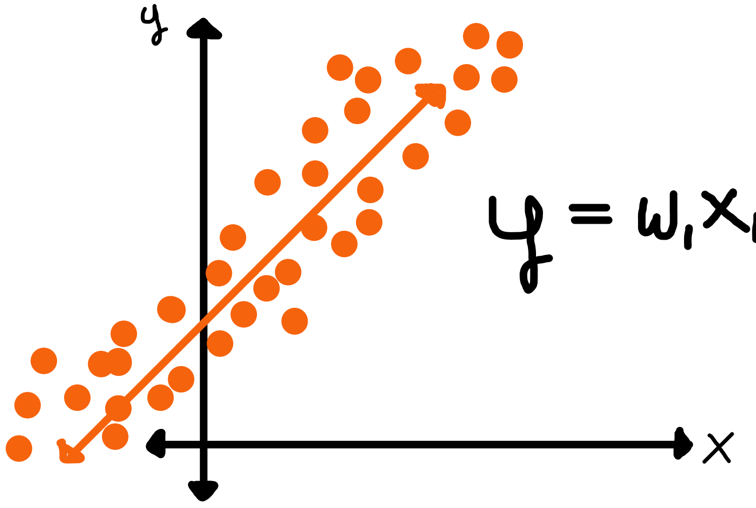
Neuron

Artificial

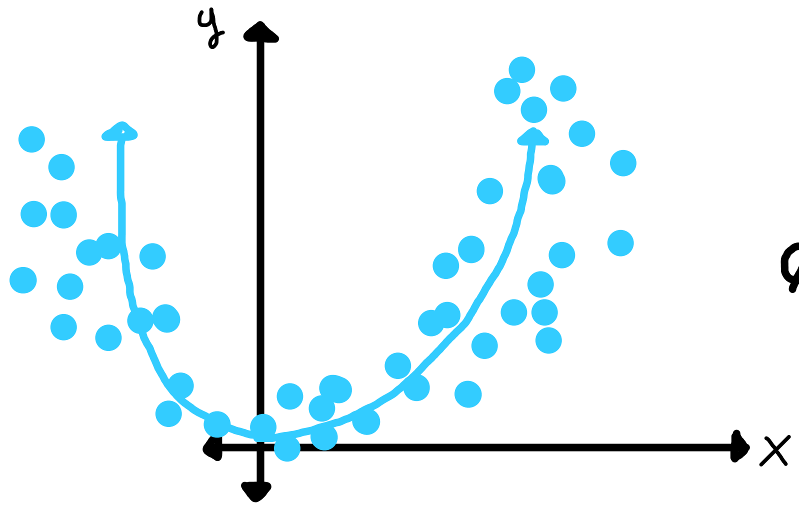
Neural

Network

"Every computable multivariable function can be implemented by a three-layer neural network with computable activation functions and weights." [1]



$$y = w_1x_1 + b$$



$$y = f(x)$$

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

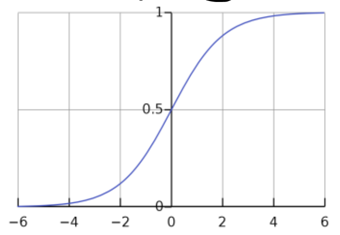
$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

$$v = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

$$y = \phi(v)$$

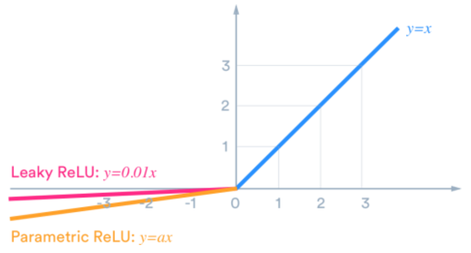
Sigmoid

$$\phi(v) = \frac{1}{1+e^{-v}}$$



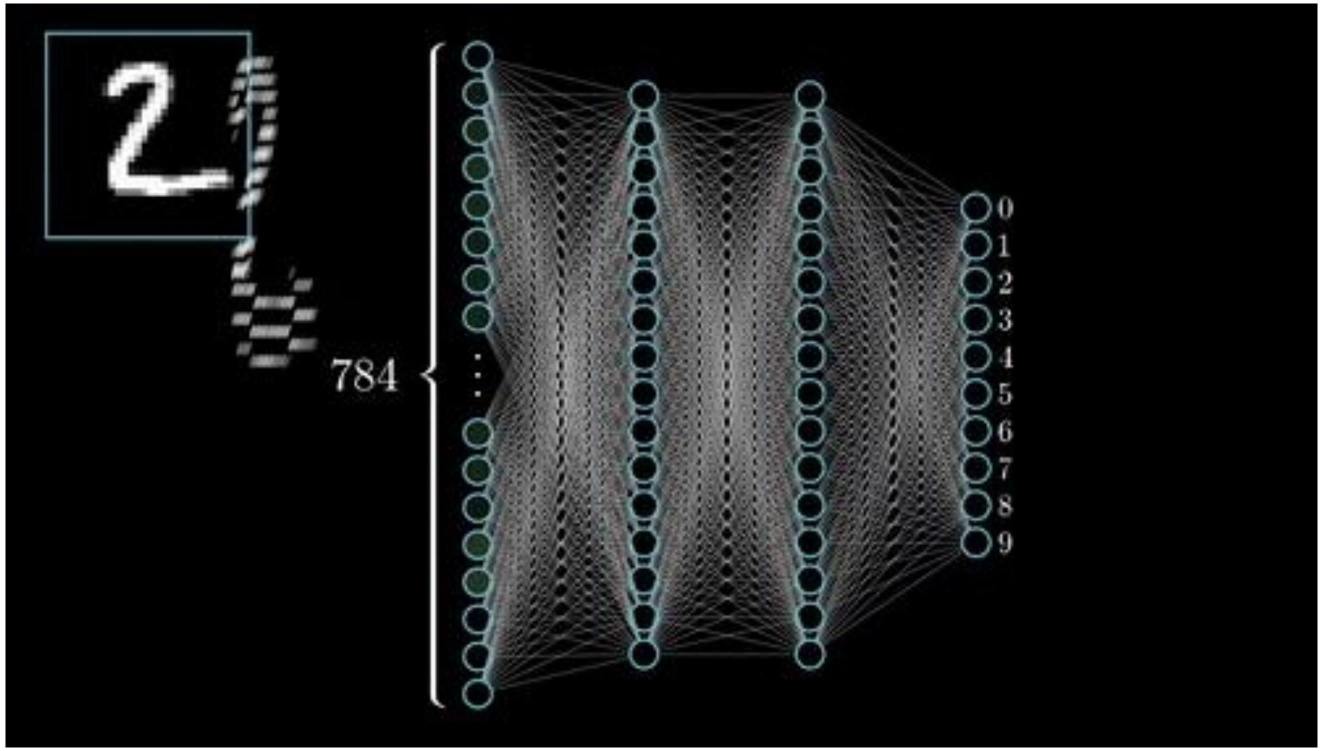
ReLU

$$\phi(v) = \begin{cases} 0 & ; v<0 \\ v & ; v\geq 0 \end{cases}$$



What is an ANN?

Network

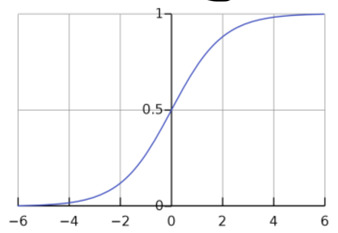


$$v = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

$$y = \phi(v)$$

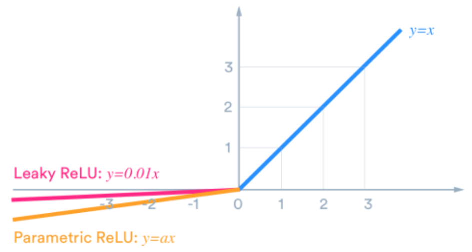
Sigmoid

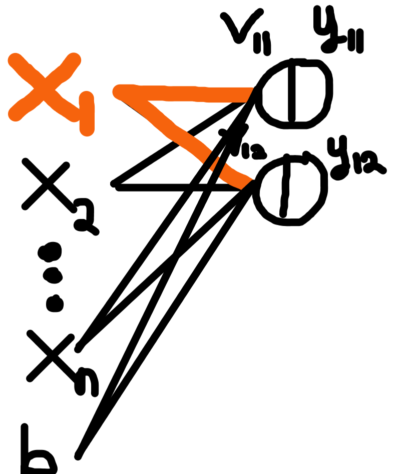
$$\phi(v) = \frac{1}{1+e^{-v}}$$



ReLU

$$\phi(v) = \begin{cases} 0 & ; v<0 \\ v & ; v\geq 0 \end{cases}$$





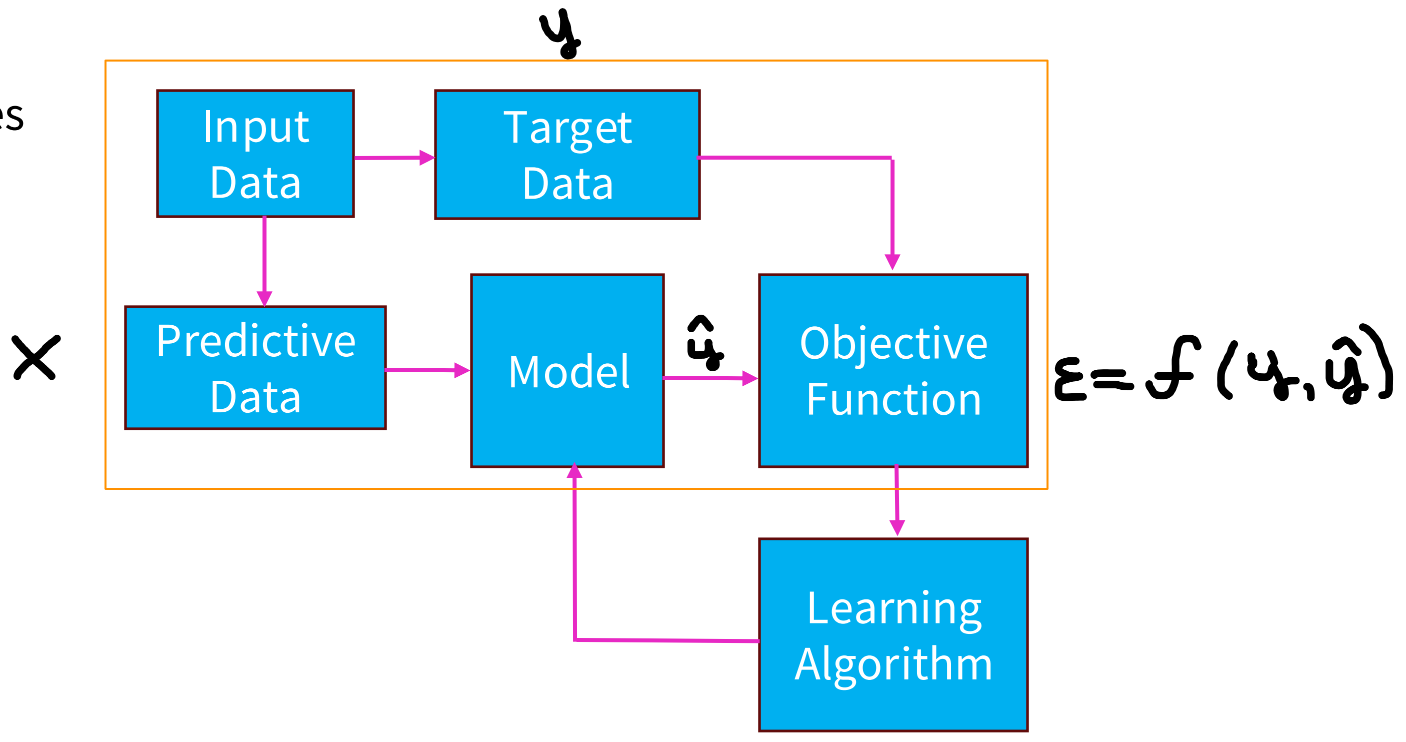
What is an ANN? **Model**

$$\hat{y} = f(\sigma(v), w, b)$$

$$v = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

Model Training

1. **Random Initialization** of all weights & biases
2. **Forward Pass**
   - Predicts output at every unit
   - Holds weights & biases fixed
3. **Backward Pass**
   - Optimizes weights & biases
   - Holding output at each unit fixed



Model Testing

1. **Forward Pass**
   - Predicts output at every unit
   - Holds weights & biases fixed

Why Implement ANNs on FPGA?

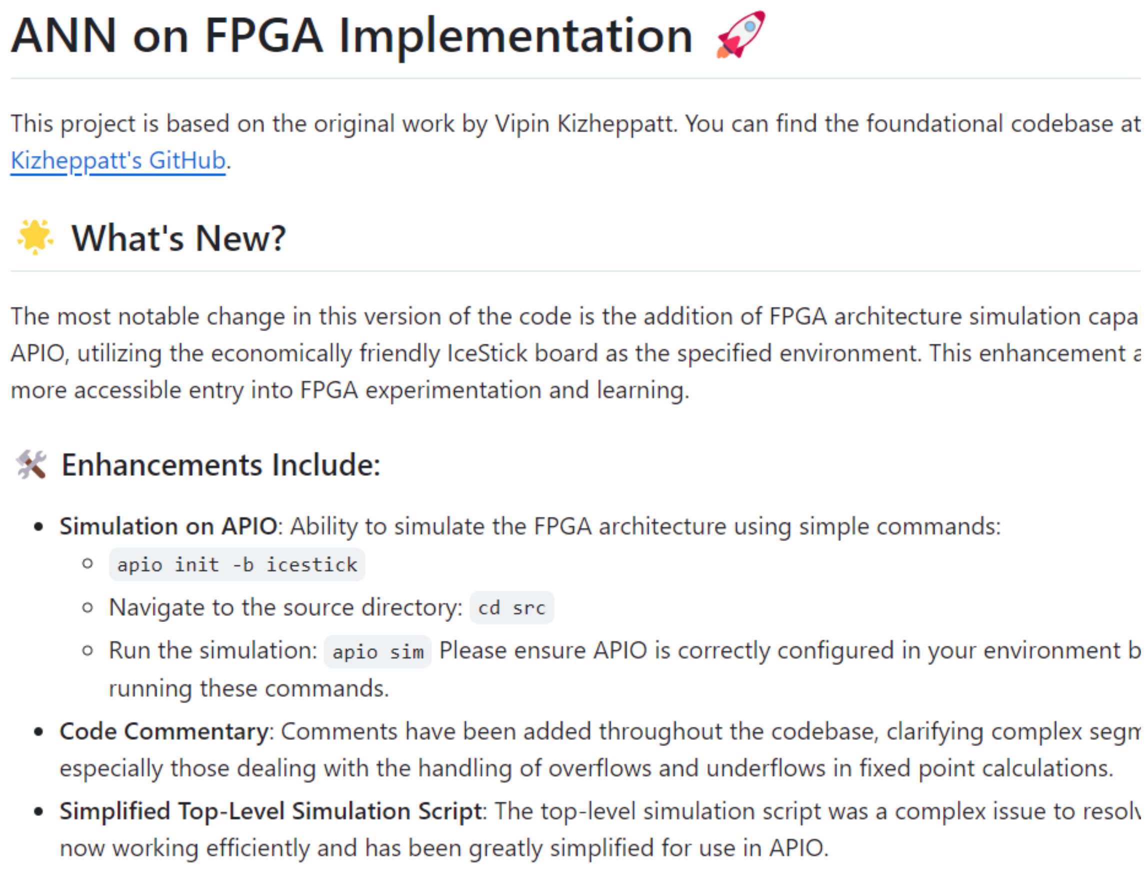
Edge Computing

- Process information closer to device to:
  - Reduce network BW requirements
  - Decrease latency
  - Improve data security

Edge Computing Industries

- Robotics
- Virtual Reality
- Closed Loop Biomedical Interfaces
- High Frequency Trading

ANN ON AN FPGA



ANN on FPGA Implementation 🚀

This project is based on the original work by Vipin Kizheppatt. You can find the foundational codebase at Kizheppatt's GitHub.

🌟 What's New?

The most notable change in this version of the code is the addition of FPGA architecture simulation capa APIO, utilizing the economically friendly IceStick board as the specified environment. This enhancement a more accessible entry into FPGA experimentation and learning.

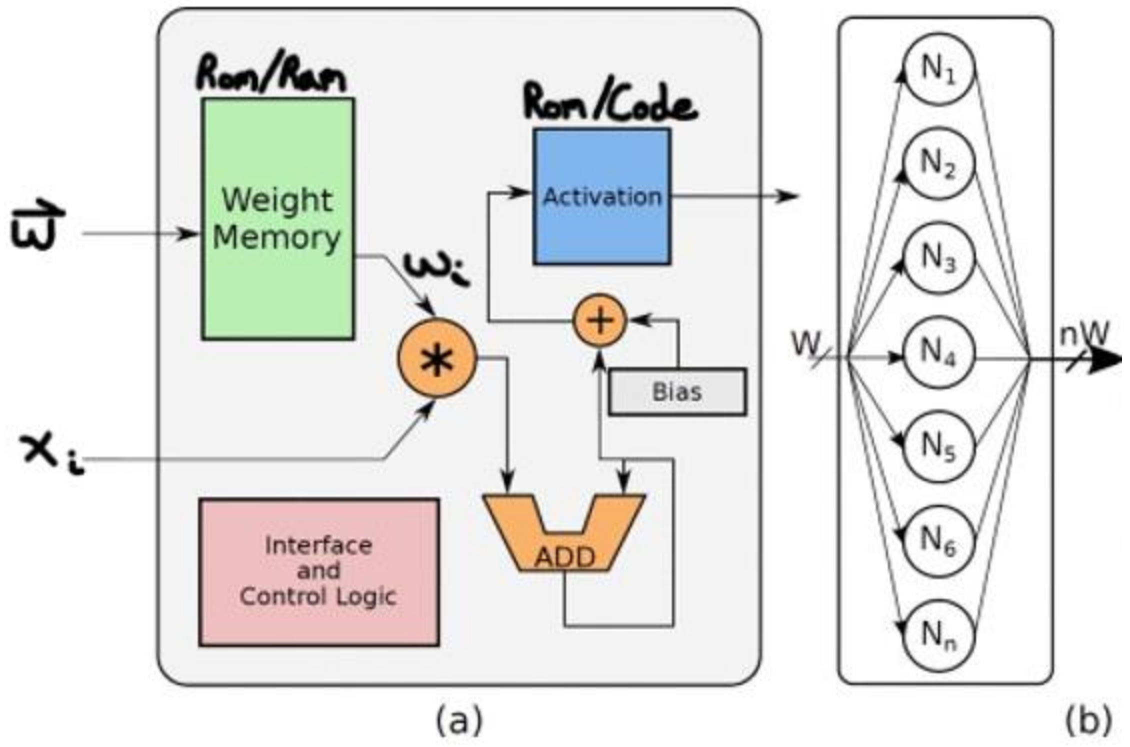
🛠️ Enhancements Include:

- **Simulation on APIO**: Ability to simulate the FPGA architecture using simple commands:
  - `apio init -b icestick`
  - Navigate to the source directory: `cd src`
  - Run the simulation: `apio sim` Please ensure APIO is correctly configured in your environment b running these commands.
- **Code Commentary**: Comments have been added throughout the codebase, clarifying complex segn especially those dealing with the handling of overflows and underflows in fixed point calculations.
- **Simplified Top-Level Simulation Script**: The top-level simulation script was a complex issue to resolv now working efficiently and has been greatly simplified for use in APIO.

ANN on an FPGA

Most Important Verilog Files

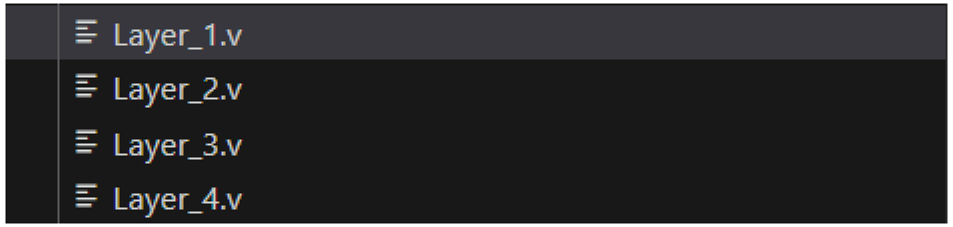
- Layer files
  - Neuron file
    - Activation function file
    - Weight Memory file



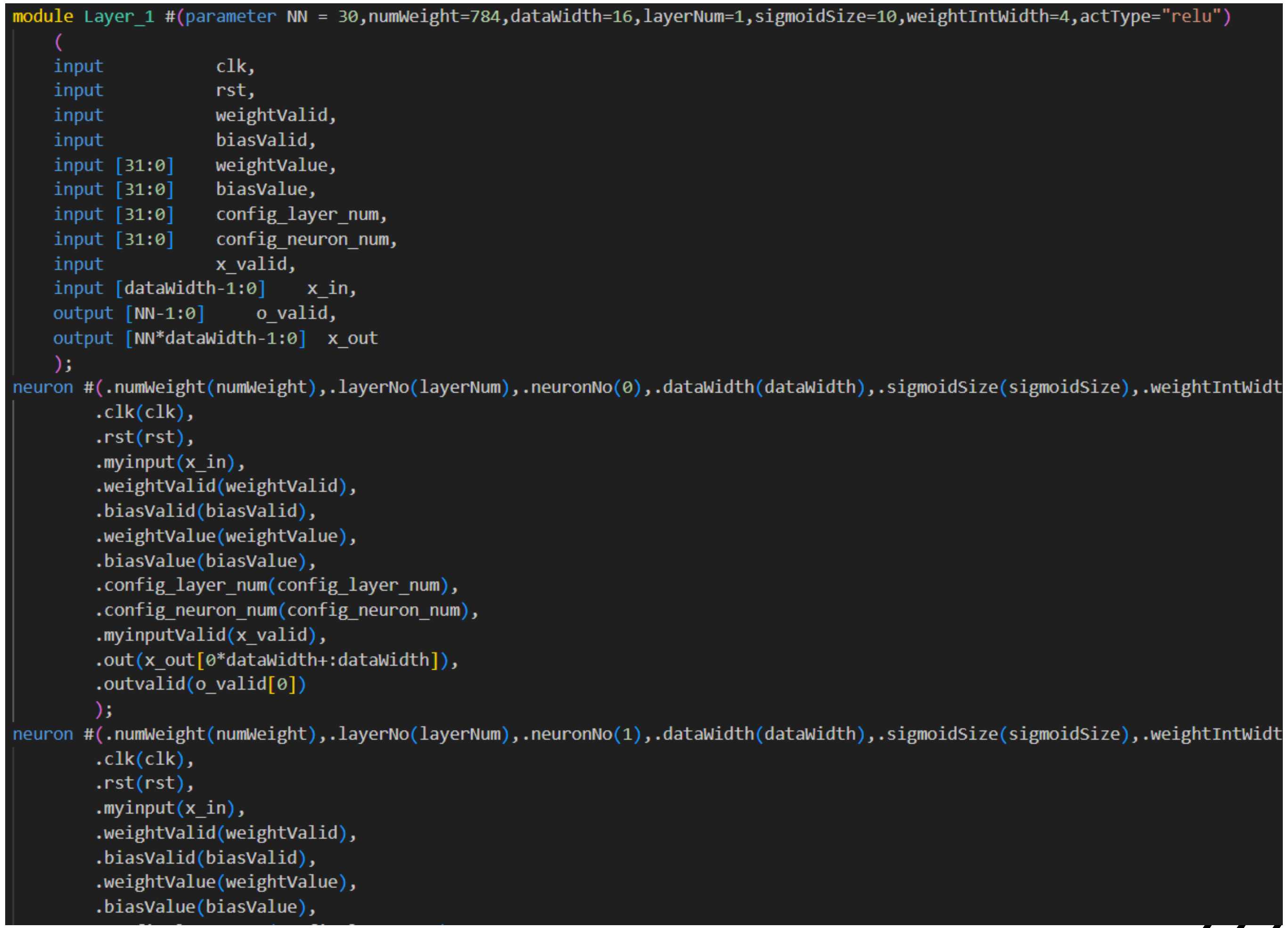
[2]

ANN on an FPGA

Layers



```
Layer_1.v
Layer_2.v
Layer_3.v
Layer_4.v
```



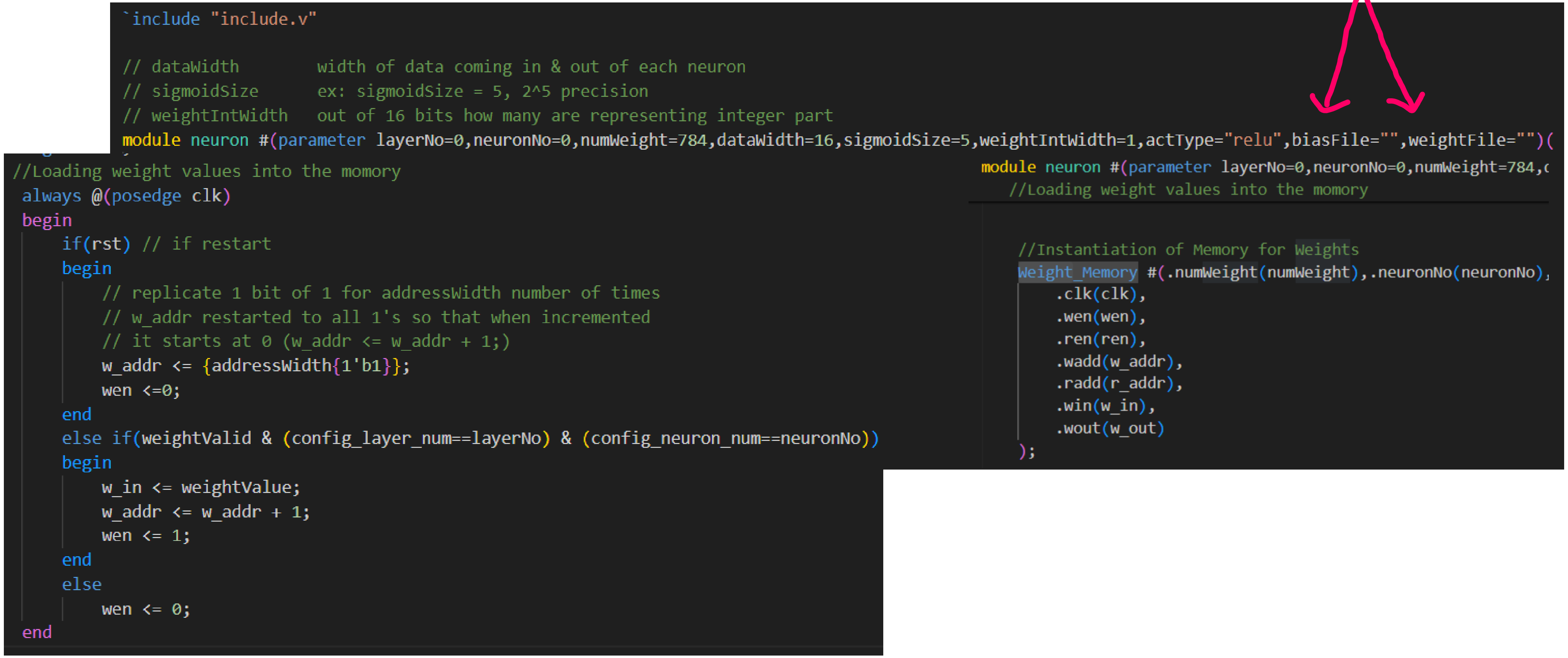
```
module Layer_1 #(parameter NN = 30,numWeight=784,dataWidth=16,layerNum=1,sigmoidSize=10,weightIntWidth=4,actType="relu")
    (
    input           clk,
    input           rst,
    input           weightValid,
    input           biasValid,
    input [31:0]    weightValue,
    input [31:0]    biasValue,
    input [31:0]    config_layer_num,
    input [31:0]    config_neuron_num,
    input           x_valid,
    input [dataWidth-1:0]    x_in,
    output [NN-1:0]     o_valid,
    output [NN*dataWidth-1:0]  x_out
    );
neuron #(.numWeight(numWeight),.layerNo(layerNum),.neuronNo(0),.dataWidth(dataWidth),.sigmoidSize(sigmoidSize),.weightIntWidt
        .clk(clk),
        .rst(rst),
        .myinput(x_in),
        .weightValid(weightValid),
        .biasValid(biasValid),
        .weightValue(weightValue),
        .biasValue(biasValue),
        .config_layer_num(config_layer_num),
        .config_neuron_num(config_neuron_num),
        .myinputValid(x_valid),
        .out(x_out[0*dataWidth+:dataWidth]),
        .outvalid(o_valid[0])
        );
neuron #(.numWeight(numWeight),.layerNo(layerNum),.neuronNo(1),.dataWidth(dataWidth),.sigmoidSize(sigmoidSize),.weightIntWidt
        .clk(clk),
        .rst(rst),
        .myinput(x_in),
        .weightValid(weightValid),
        .biasValid(biasValid),
        .weightValue(weightValue),
        .biasValue(biasValue),
```

ANN on an FPGA

Neuron

$$\hat{y} = f(x(v), w, b)$$

?



```
`include "include.v"

// dataWidth        width of data coming in & out of each neuron
// sigmoidSize      ex: sigmoidSize = 5, 2^5 precision
// weightIntWidth   out of 16 bits how many are representing integer part
module neuron #(parameter layerNo=0,neuronNo=0,numWeight=784,dataWidth=16,sigmoidSize=5,weightIntWidth=1,actType="relu",biasFile="",weightFile="")(
```

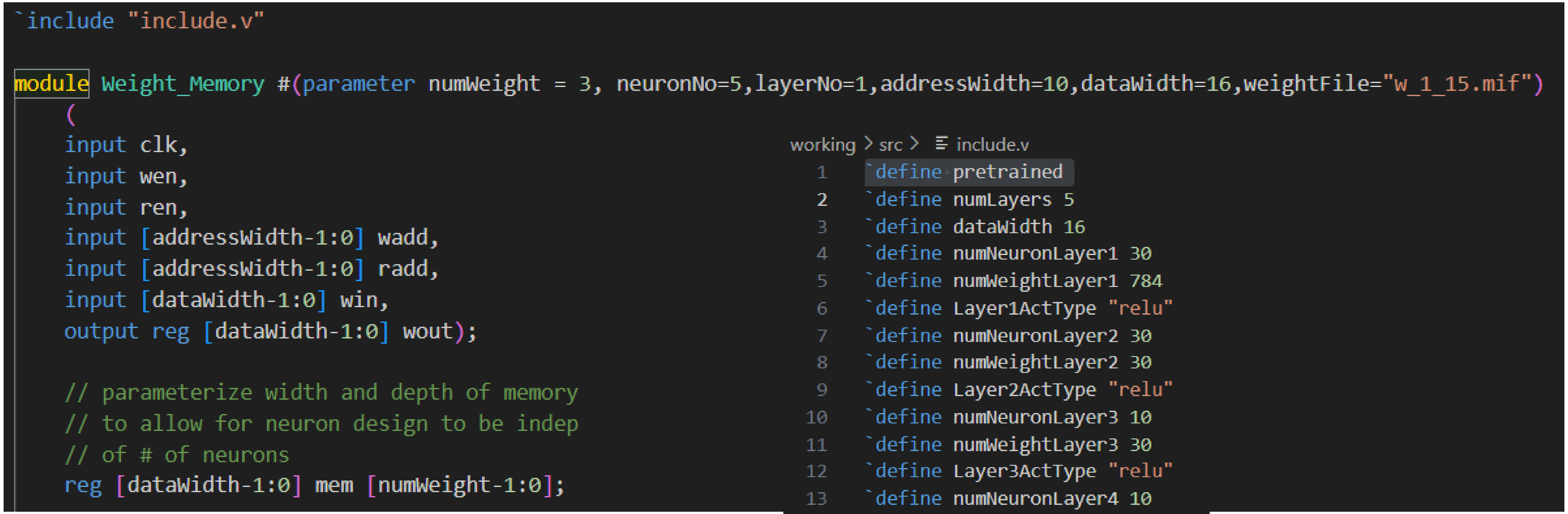
```
//Loading weight values into the momory
 always @(posedge clk)
 begin
     if(rst) // if restart
     begin
         // replicate 1 bit of 1 for addressWidth number of times
         // w_addr restarted to all 1's so that when incremented
         // it starts at 0 (w_addr <= w_addr + 1;)
         w_addr <= {addressWidth{1'b1}};
         wen <=0;
     end
     else if(weightValid & (config_layer_num==layerNo) & (config_neuron_num==neuronNo))
     begin
         w_in <= weightValue;
         w_addr <= w_addr + 1;
         wen <= 1;
     end
     else
         wen <= 0;
 end
```

```
module neuron #(parameter layerNo=0,neuronNo=0,numWeight=784,
   //Loading weight values into the momory


   //Instantiation of Memory for Weights
   Weight_Memory #(.numWeight(numWeight),.neuronNo(neuronNo),
       .clk(clk),
       .wen(wen),
       .ren(ren),
       .wadd(w_addr),
       .radd(r_addr),
       .win(w_in),
       .wout(w_out)
   );
```

ANN on an FPGA

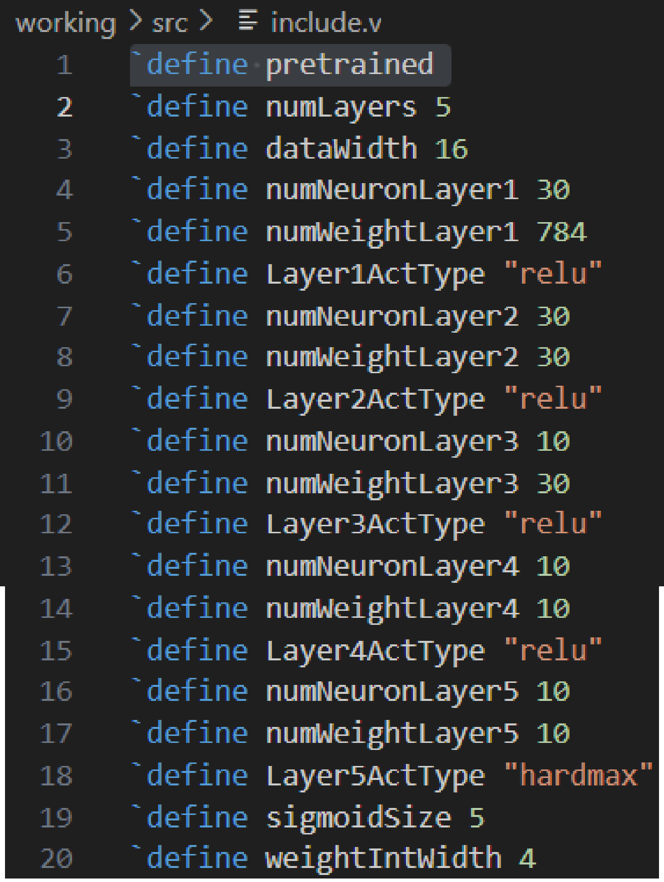
Weight Memory



```verilog
`include "include.v"

module Weight_Memory #(parameter numWeight = 3, neuronNo=5,layerNo=1,addressWidth=10,dataWidth=16,weightFile="w_1_15.mif")
    (
    input clk,
    input wen,
    input ren,
    input [addressWidth-1:0] wadd,
    input [addressWidth-1:0] radd,
    input [dataWidth-1:0] win,
    output reg [dataWidth-1:0] wout);

    // parameterize width and depth of memory
    // to allow for neuron design to be indep
    // of # of neurons
    reg [dataWidth-1:0] mem [numWeight-1:0];
```

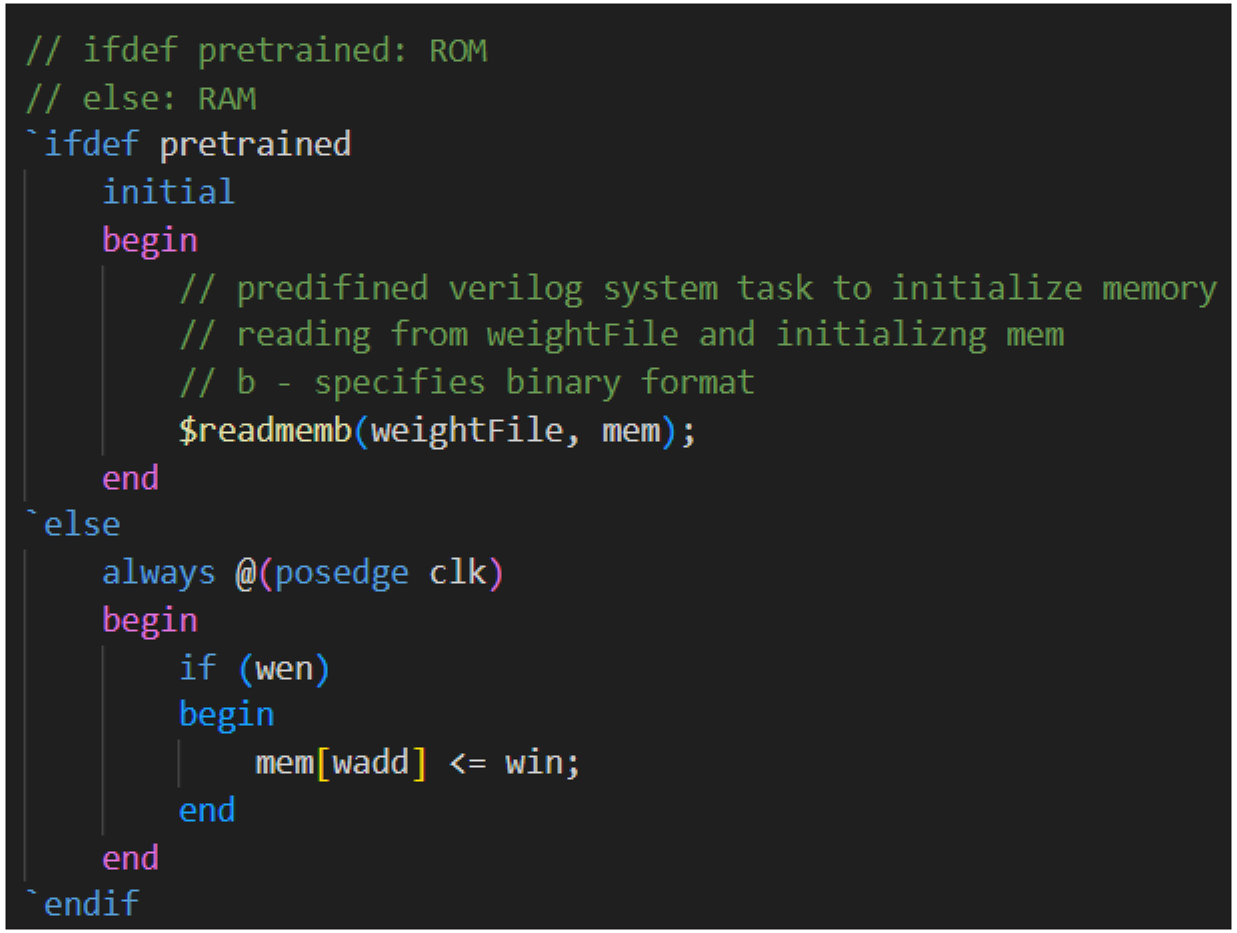


working > src > include.v

```verilog
`define pretrained
`define numLayers 5
`define dataWidth 16
`define numNeuronLayer1 30
`define numWeightLayer1 784
`define Layer1ActType "relu"
`define numNeuronLayer2 30
`define numWeightLayer2 30
`define Layer2ActType "relu"
`define numNeuronLayer3 10
`define numWeightLayer3 30
`define Layer3ActType "relu"
`define numNeuronLayer4 10
`define numWeightLayer4 10
`define Layer4ActType "relu"
`define numNeuronLayer5 10
`define numWeightLayer5 10
`define Layer5ActType "hardmax"
`define sigmoidSize 5
`define weightIntWidth 4
```

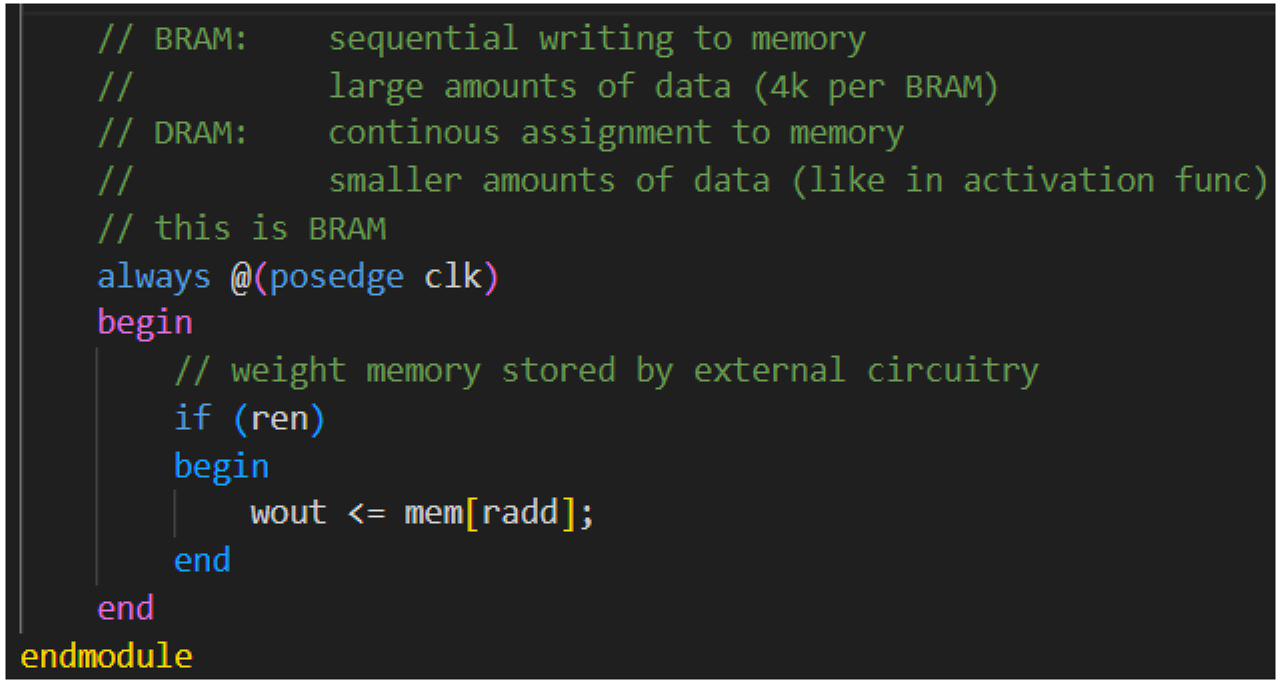
ANN on an FPGA

Weight Memory



```
// ifdef pretrained: ROM
// else: RAM
`ifdef pretrained
    initial
    begin
        // predifined verilog system task to initialize memory
        // reading from weightFile and initializng mem
        // b - specifies binary format
        $readmemb(weightFile, mem);
    end
`else
    always @(posedge clk)
    begin
        if (wen)
        begin
            mem[wadd] <= win;
        end
    end
`endif
```

ANN on an FPGA **Weight Memory**

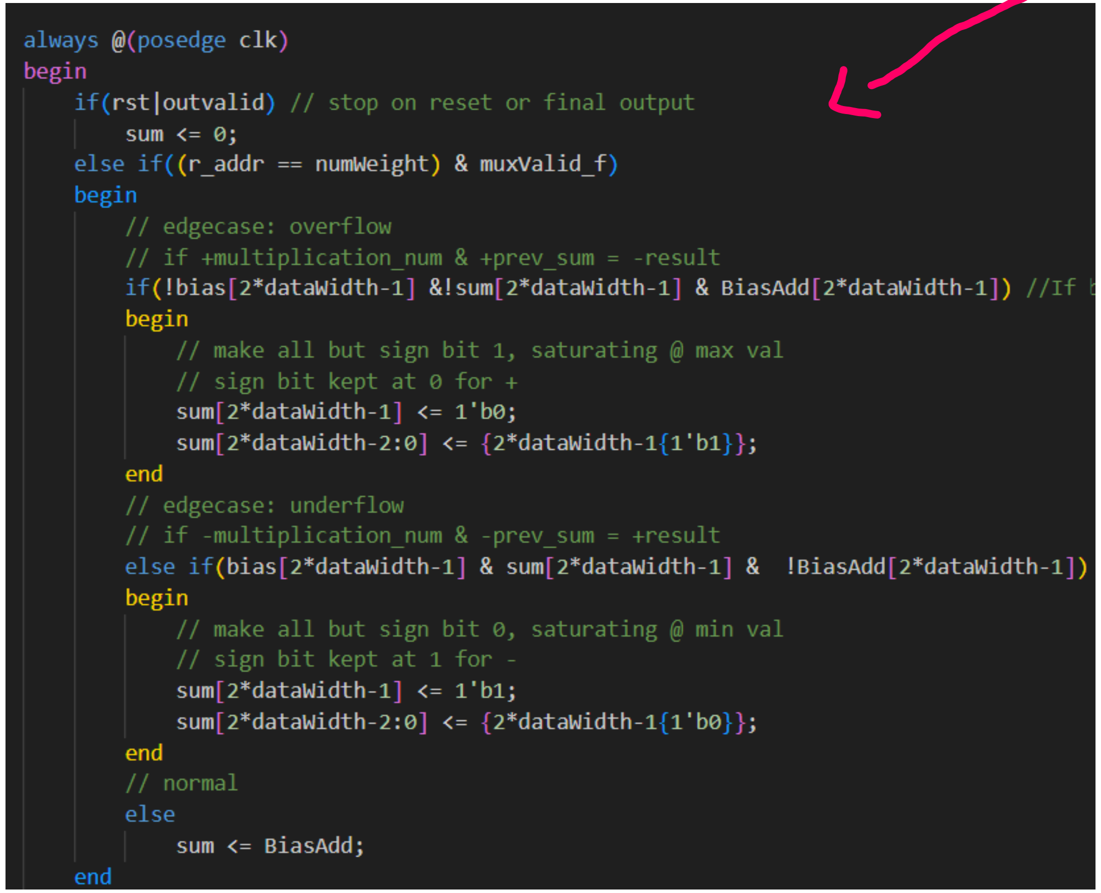


```
    // BRAM:    sequential writing to memory
    //          large amounts of data (4k per BRAM)
    // DRAM:    continous assignment to memory
    //          smaller amounts of data (like in activation func)
    // this is BRAM
    always @(posedge clk)
    begin
        // weight memory stored by external circuitry
        if (ren)
        begin
            wout <= mem[radd];
        end
    end
endmodule
```

ANN on an FPGA

Neuron

$$\hat{y} = f(\sigma(v), \underset{\checkmark}{w}, \underset{\checkmark}{b})$$

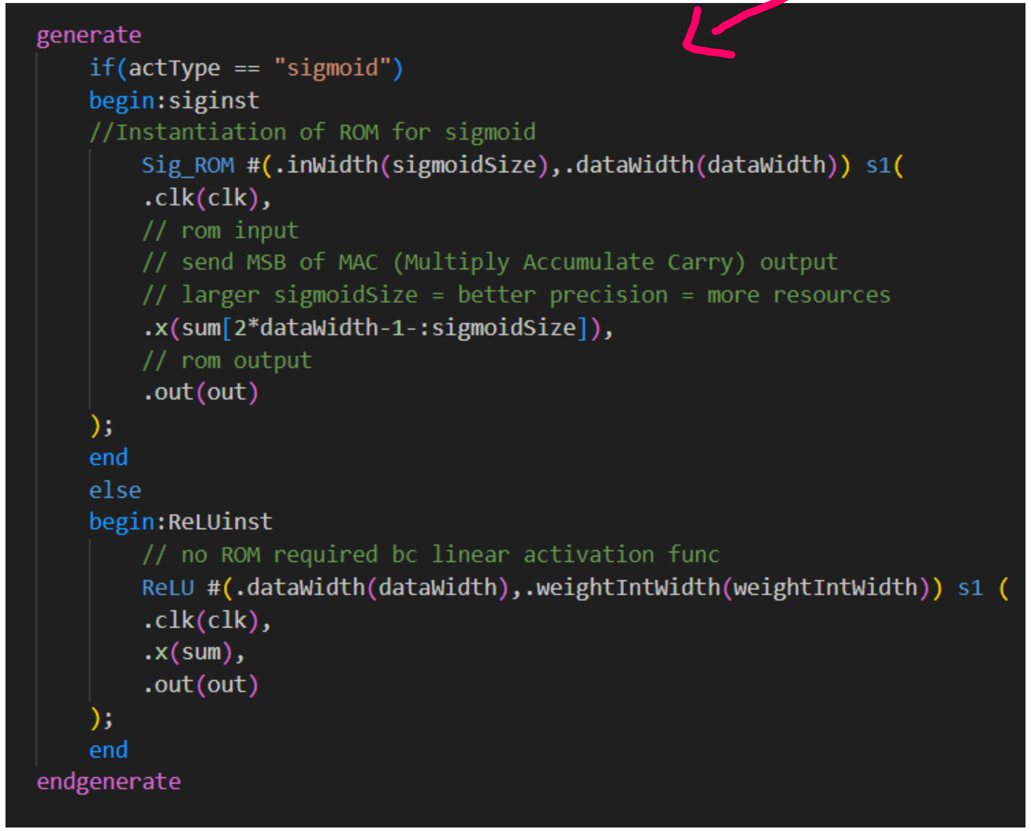


```
always @(posedge clk)
begin
    if(rst|outvalid) // stop on reset or final output
        sum <= 0;
    else if((r_addr == numWeight) & muxValid_f)
    begin
        // edgecase: overflow
        // if +multiplication_num & +prev_sum = -result
        if(!bias[2*dataWidth-1] &!sum[2*dataWidth-1] & BiasAdd[2*dataWidth-1]) //If
        begin
            // make all but sign bit 1, saturating @ max val
            // sign bit kept at 0 for +
            sum[2*dataWidth-1] <= 1'b0;
            sum[2*dataWidth-2:0] <= {2*dataWidth-1{1'b1}};
        end
        // edgecase: underflow
        // if -multiplication_num & -prev_sum = +result
        else if(bias[2*dataWidth-1] & sum[2*dataWidth-1] &  !BiasAdd[2*dataWidth-1])
        begin
            // make all but sign bit 0, saturating @ min val
            // sign bit kept at 1 for -
            sum[2*dataWidth-1] <= 1'b1;
            sum[2*dataWidth-2:0] <= {2*dataWidth-1{1'b0}};
        end
        // normal
        else
            sum <= BiasAdd;
    end
```

ANN on an FPGA

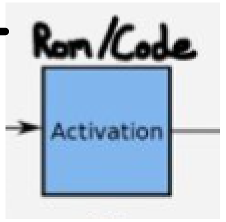
Neuron

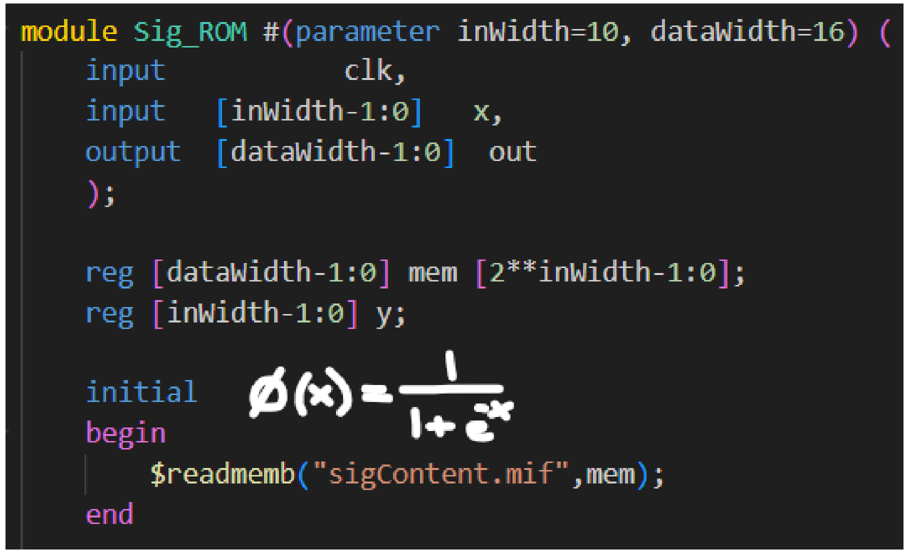
$$\hat{y} = f(\underbrace{\varphi(v)}, \underset{\checkmark}{w}, \underset{\checkmark}{b})$$



```
generate
    if(actType == "sigmoid")
    begin:siginst
    //Instantiation of ROM for sigmoid
        Sig_ROM #(.inWidth(sigmoidSize),.dataWidth(dataWidth)) s1(
        .clk(clk),
        // rom input
        // send MSB of MAC (Multiply Accumulate Carry) output
        // larger sigmoidSize = better precision = more resources
        .x(sum[2*dataWidth-1-:sigmoidSize]),
        // rom output
        .out(out)
    );
    end
    else
    begin:ReLUinst
        // no ROM required bc linear activation func
        ReLU #(.dataWidth(dataWidth),.weightIntWidth(weightIntWidth)) s1 (
        .clk(clk),
        .x(sum),
        .out(out)
    );
    end
endgenerate
```

ANN on an FPGA



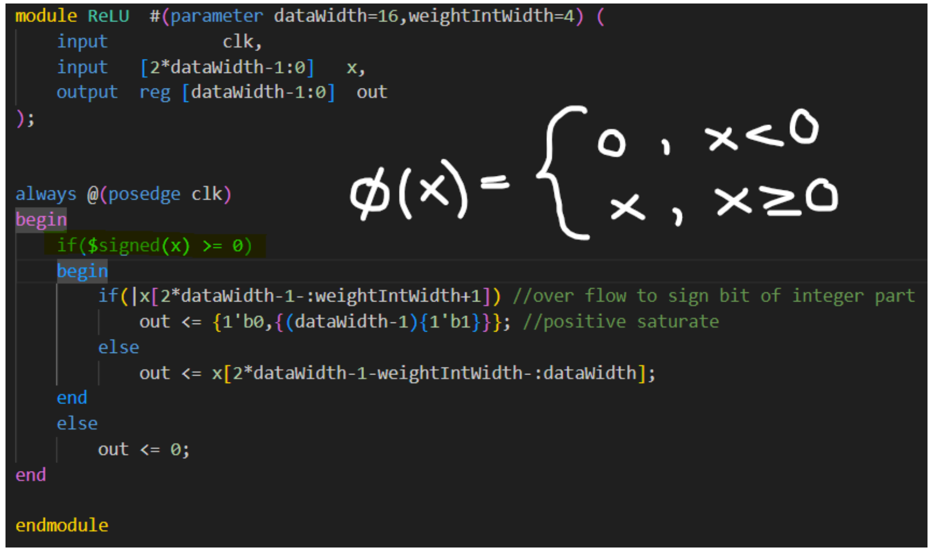


```verilog
module Sig_ROM #(parameter inWidth=10, dataWidth=16) (
    input           clk,
    input   [inWidth-1:0]   x,
    output  [dataWidth-1:0]  out
    );

    reg [dataWidth-1:0] mem [2**inWidth-1:0];
    reg [inWidth-1:0] y;

    initial
    begin
        $readmemb("sigContent.mif",mem);
    end
```

$$\phi(x) = \frac{1}{1+e^{-x}}$$



```verilog
module ReLU  #(parameter dataWidth=16,weightIntWidth=4) (
    input           clk,
    input   [2*dataWidth-1:0]   x,
    output  reg [dataWidth-1:0]  out
);


always @(posedge clk)
begin
    if($signed(x) >= 0)
    begin
        if(|x[2*dataWidth-1-:weightIntWidth+1]) //over flow to sign bit of integer part
            out <= {1'b0,{(dataWidth-1){1'b1}}}; //positive saturate
        else
            out <= x[2*dataWidth-1-weightIntWidth-:dataWidth];
    end
    else
        out <= 0;
end

endmodule
```

$$\phi(x) = \begin{cases} 0, & x<0 \\ x, & x\geq 0 \end{cases}$$

Future Goals

- Interface with camera to pass in real data
- Improve model speed?
- Enable weight upload through communication system
- Apply to realistic dataset with real application

Recommended Materials

- Introduction to FPGA YouTube Series by Digi-Key

https://www.youtube.com/watch?v=lLg1AgA2Xoo&list=PLEBQazB0HUyT1WmMONxRZn9NmQ_9CIKhb

- The Complete Mathematics of Neural Networks and Deep Learning

https://www.youtube.com/watch?v=Ixl3nykKG9M&t=2321s

- Neural Network implementation this Project is Based On
  - Paper: https://ieeexplore.ieee.org/abstract/document/8977883
  - YouTube Series: https://www.youtube.com/watch?v=rw_JITpbh3k&list=PLJePd8QU_LYKZwJnByZ8FHDg5l1rXtcIq&index=1
  - GitHub: https://github.com/vipinkmenon/neuralNetwork/tree/master
  - Updated GitHub: https://github.com/jorgelalberto/ANNonFPGA

References

[1] Ismailov VE. A three layer neural network can represent any discontinuous multivariate function. CoRR. 2020;abs/2012.03016. Available from: https://arxiv.org/abs/2012.03016

[2] Vipin K. ZyNet: Automating Deep Neural Network Implementation on Low-Cost Reconfigurable Edge Computing Platforms. 2019 International Conference on Field-Programmable Technology (ICFPT); 2019 Dec; Tianjin, China. IEEE; 2019. p. 323-326. doi: 10.1109/ICFPT47387.2019.00058