Mini Project 2

EE 278 Digital Design for AI and DSP

Simple Fixed-Point DNN/MLP Inference Engine

Submitted By

Bhavin Patel

SJSU-ID 015954770

MSEE – Fall 2022

San Jose State University

Department of Electrical Engineering

1. Introduction

An artificial neural network algorithm primarily consists of MAC operations. So, implementing them directly in hardware can allow large gains when scaling the network sizes. In this work, a neural network is designed with the pipelined MAC units that were designed as a part of Mini-Project 1. With five 6-bit input patterns, 4 hidden nodes, and 3x 32-bit outputs, the implemented hardware neural network makes decisions on a set of input patterns in 58 clock cycles. The project helps understand the computational requirements of a neural network operation and also how parallel MACs can significantly reduce the overall latency of the network as compared to the CPU which does the operations sequentially. It also helps understand the process of implementing hardware for artificial neural networks.

2. Neural Network Description

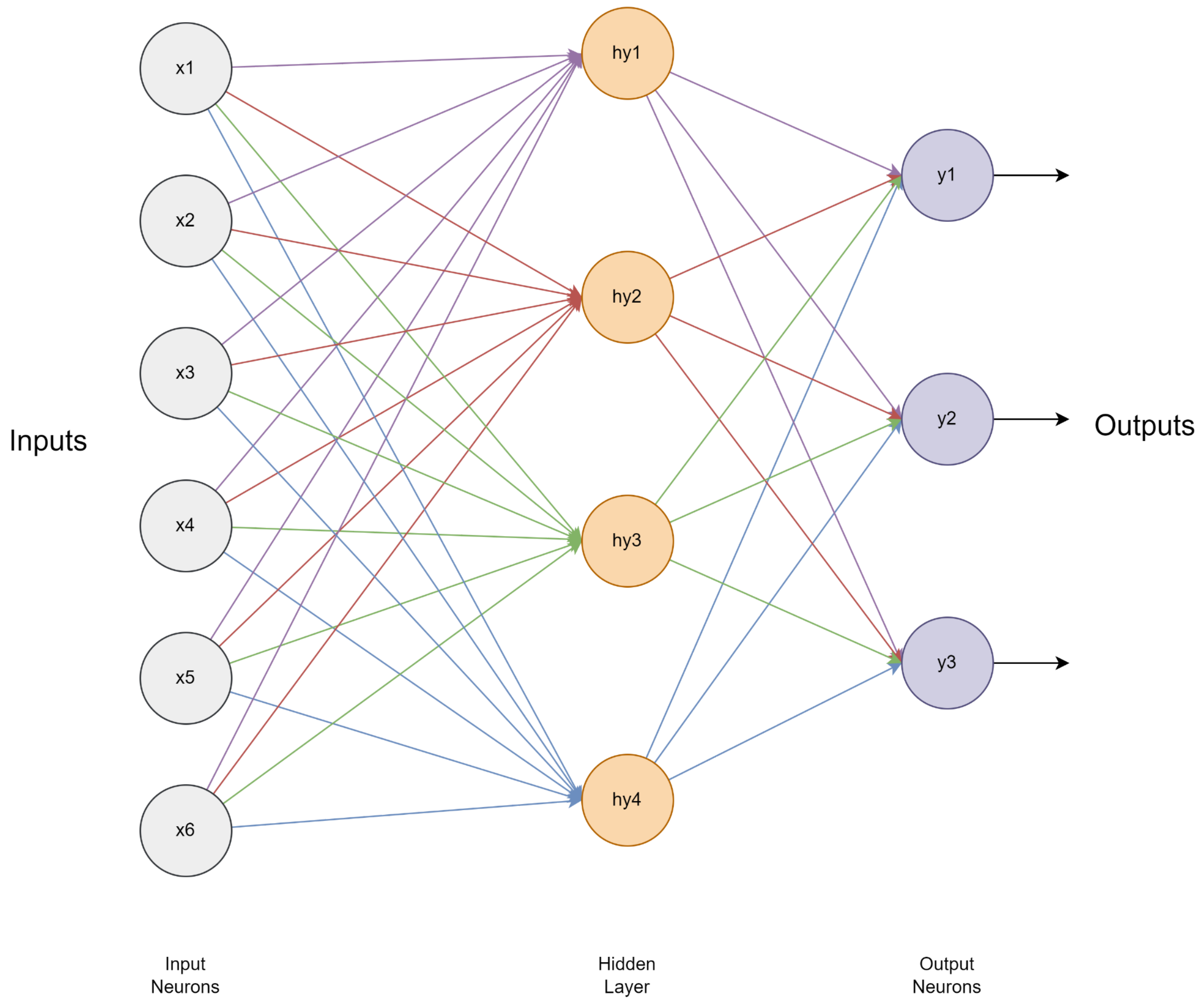

Fig.1 Neural Network Diagram

The given neural network has 6 inputs named x1-x6 in the design. The hidden layer has 4 neurons that receive x1-x6 and produces 4 outputs hy1-hy4. These outputs from the hidden layer are connected to the inputs of the output neurons. The output layer has 3 neurons named y1-y3 in the design.

$$z = \left(\sum_{i=1}^{n} x_i \times w_i \right)$$

Each neuron does the above operation, which is essentially the Multiply-and-Accumulate (MAC) function. Therefore the MAC unit designed from the Mini-Project 1 is reused in this project to implement the MAC operation. The pipelined MAC is implemented using an FP multiplier, FP adder, and an output buffer as below. All the input values, synapse weights, and neuron output are represented by IEEE754 single precision floating point format.

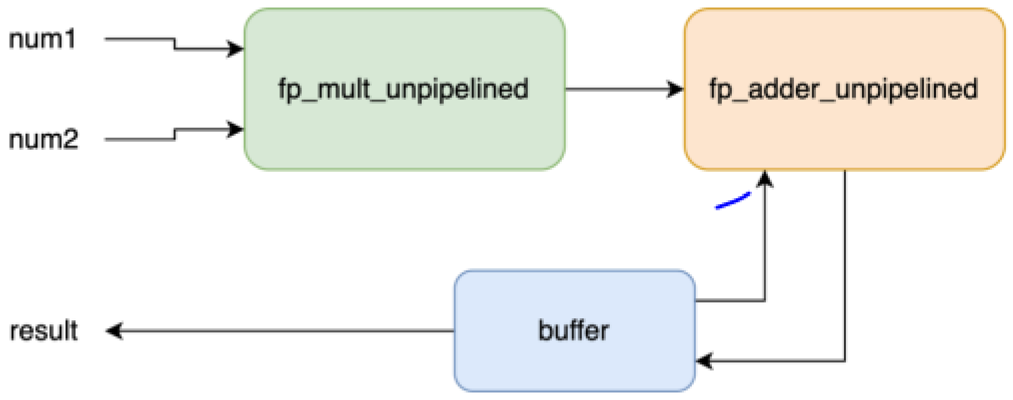

Fig.2 Block diagram of MAC Unit Designed in MiniPr Implements a Neuron

3. Block Diagram of the Implementation and Resource Requirements

3.1 The Hidden Layer Verilog Design

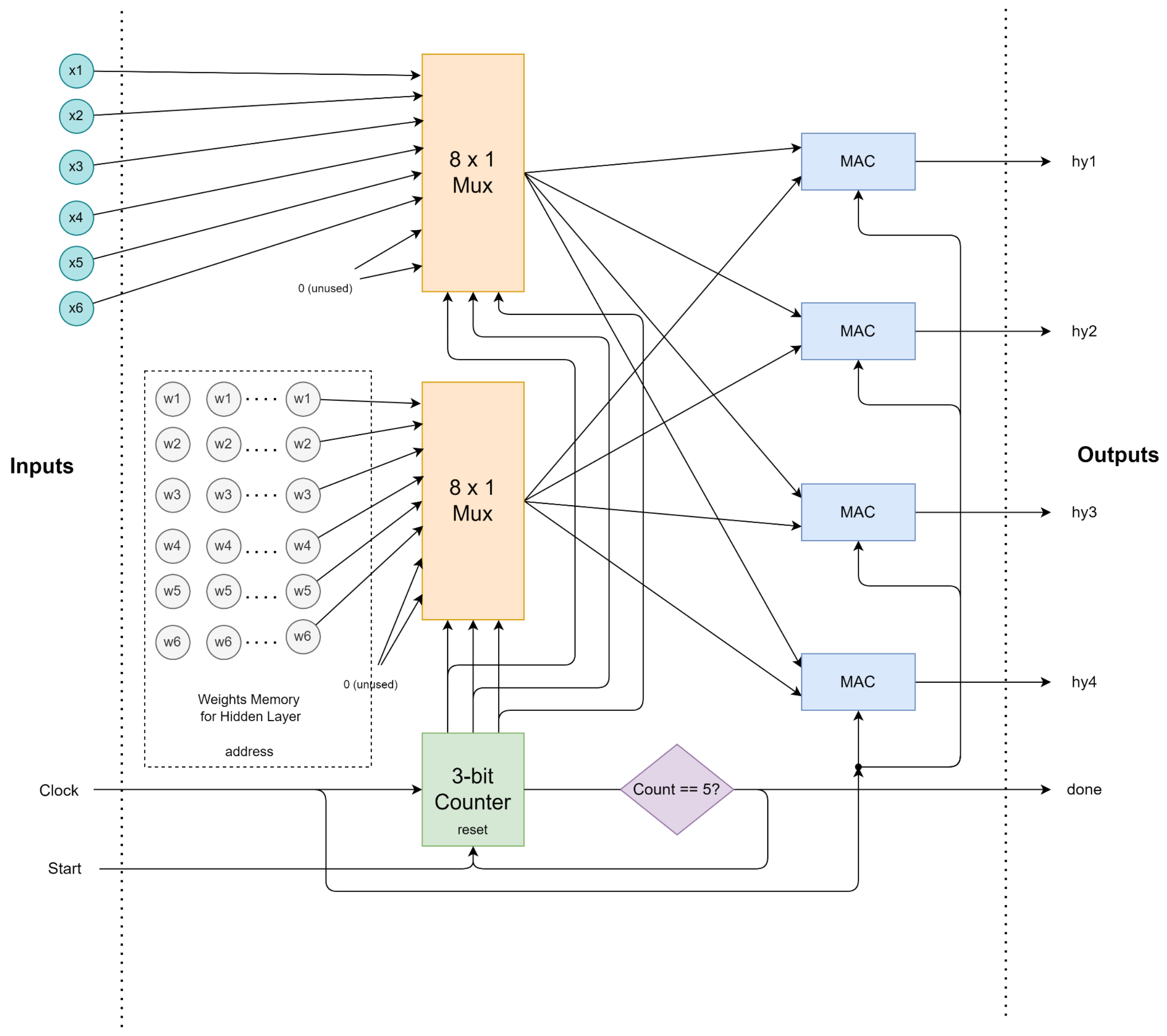

Fig.3 Implementation of Hidden Layer

3.2 Resource Requirements for Hidden Layer

The Hidden Layer has the following hardware resource requirements

1. 32-bit MAC (4 units)
2. 8*1 Mux (2 units)
3. 96 Byte memory to store 24 different 32-bit floating point weights
4. 3-bit counter

The 4 MAC units compute the output of each output neuron in the hidden layer. It receives two inputs on every clock, one is the input neuron value X and the other is the W weight. On every clock, the 3-bit counter is updated to pick each weight. When all the weights are multiplied and accumulated with all the inputs, the *done* signal is used as an indication of that.

In summary, it will do the below operations for the 1st MAC unit on each clock number.

```
1st => X1 * W11

2nd => X2 * W21 + X1 * W11

3rd => X3 * W31 + X2 * W21 + X1 * W11

4th => X4 * W41 + X3 * W31 + X2 * W21 + X11 * W11

5th => X5 * W51 + X4 * W41 + X3 * W31 + X2 * W21 + X11 * W11

6th => X6 * W61 + X5 * W51 + X4 * W41 + X3 * W31 + X2 * W21 +
X11 * W11
```

Similarly,

2nd MAC will do the same operations with Wi2 (where i=1 to 6)
2nd MAC will do the same operations with Wi3 (where i=1 to 6)
2nd MAC will do the same operations with Wi4 (where i=1 to 6)

Combined, Each MAC will perform the below operation in 6 clock cycles.

$$z = \left(\sum_{i=1}^{n} x_i \times w_i \right)$$

3.3 The Output Layer Verilog Design

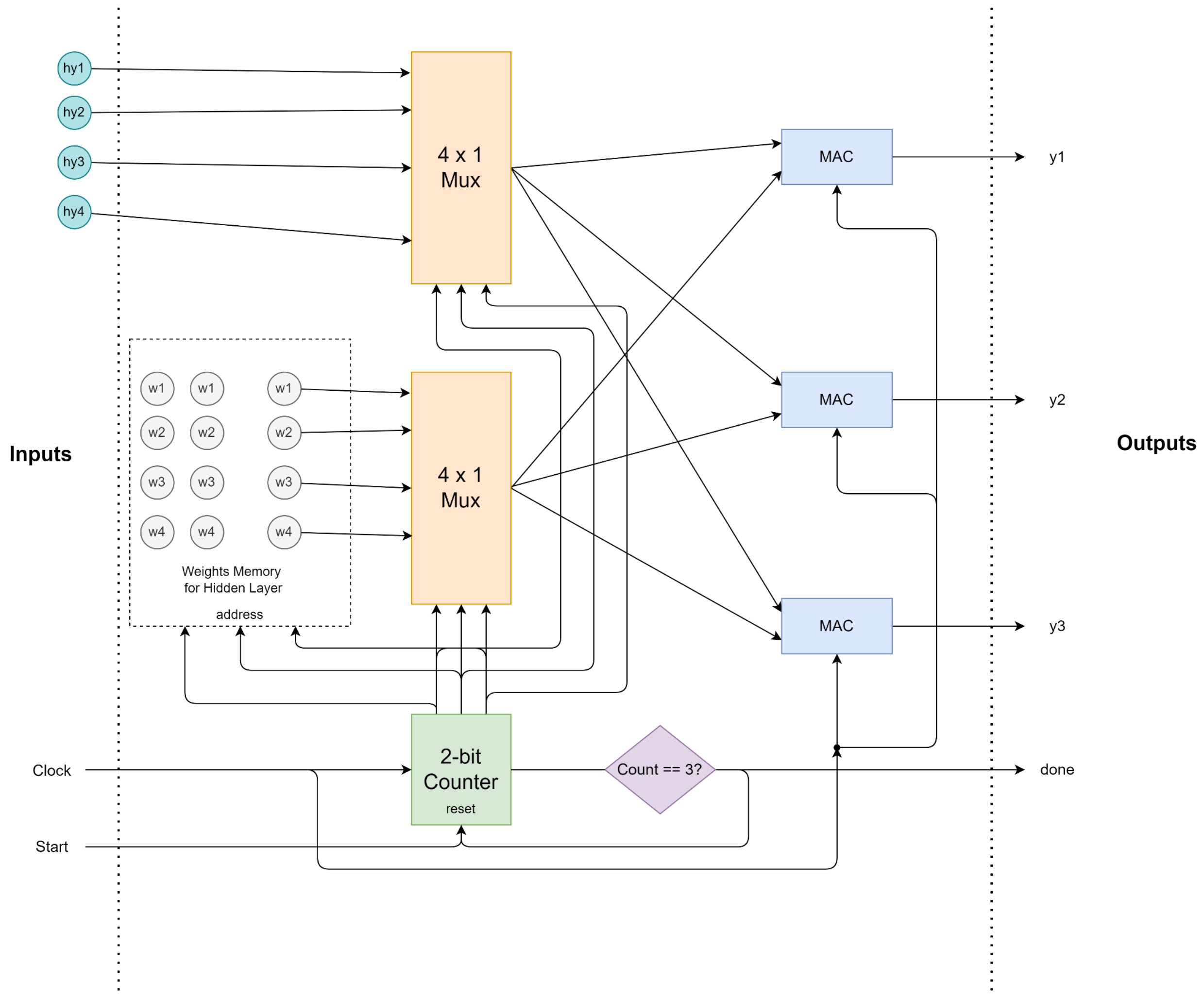

Fig.4 Implementation of Output Layer

3.4 Resource Requirements for Output Layer

The Output Layer has the following hardware resource requirements

1. 32-bit MAC (3 units)
2. 4*1 Mux (2 units)
3. 48 Byte memory to store 12 different 32-bit floating point weights
4. 2-bit counter

The 3 MAC units compute the output of each output neuron in the output layer. It receives two inputs on every clock, one is the output of hidden layer neuron value Hy and the other is the W weight. On every clock, the 2-bit counter is updated to pick each weight. When all the weights are multiplied and accumulated with all the inputs, the *done* signal is used as an indication of that.

In summary, it will do the below operations for the 1st MAC unit on each clock number.

```
1st => Hy1 * W11

2nd => Hy2 * W21 + Hy1 * W11

3rd => Hy3 * W31 + Hy2 * W21 + Hy1 * W11

4th => Hy4 * W41 + Hy3 * W31 + Hy2 * W21 + Hy11 * W11
```

Similarly,

2nd MAC will do the same operations with Wi2 (where i=1 to 6)
2nd MAC will do the same operations with Wi3 (where i=1 to 6)
2nd MAC will do the same operations with Wi4 (where i=1 to 6)

Combined, Each MAC will perform the below operation in 4 clock cycles.

$$z = \left(\sum_{i=1}^{n} x_i \times w_i \right)$$

3.5 Activation Function

Each neuron uses the ReLU(Rectified Linear Unit) activation function. The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. It can be implemented easily just by doing the logical AND of the individual output bits with its sign bit.

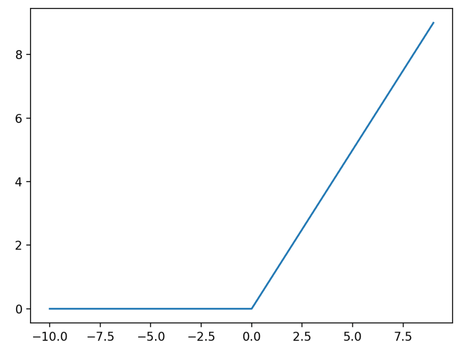

Fig.5 ReLU Activation Function

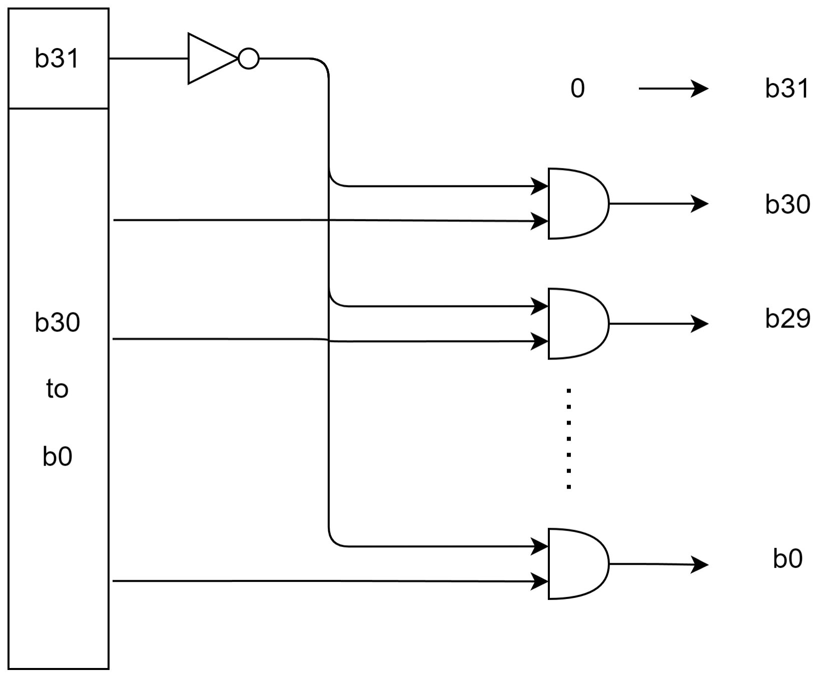

Fig.6 Implementation of ReLU Activation Function

3.6 Cascading Hidden Layer and Output Layer

To form the final neural network as shown in Fig. 1, The hidden layer and output layers are executed sequentially to get the final output of the given network as shown below.

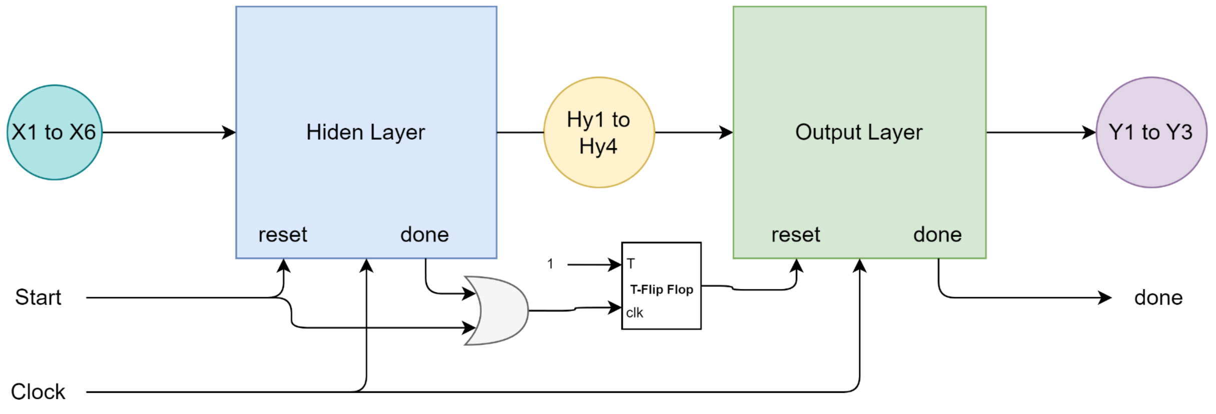

Fig.7 Implementation Block Diagram of the Neural Network

Here, the hidden layer will first calculate the outputs Hy1 to Hy4 when the start signal is applied. When the hidden layer is done calculating all the outputs, it will send an indication through the *done* signal. Until then, the output layer will stay idle. Once the hidden layer has indicated *done*, the output layer will be activated and then it will calculate the outputs Y1 to Y3.

Hardware Requirements for Cascading are low.

1. T-Flip Flop
2. OR gate

3.7 Overall Resource Requirement for the given Neural Network

Resource	Units
32-bit MAC	7
4 to 1 Mux	2
8 to 1 Mux	3
144-Byte memory to store all the weights	1
2-bit counter	1
3-bit counter	1
T-Flip Flop	1
AND gates	30
OR gate	1

4. Simulation Results

4.1 Simulation Waveforms

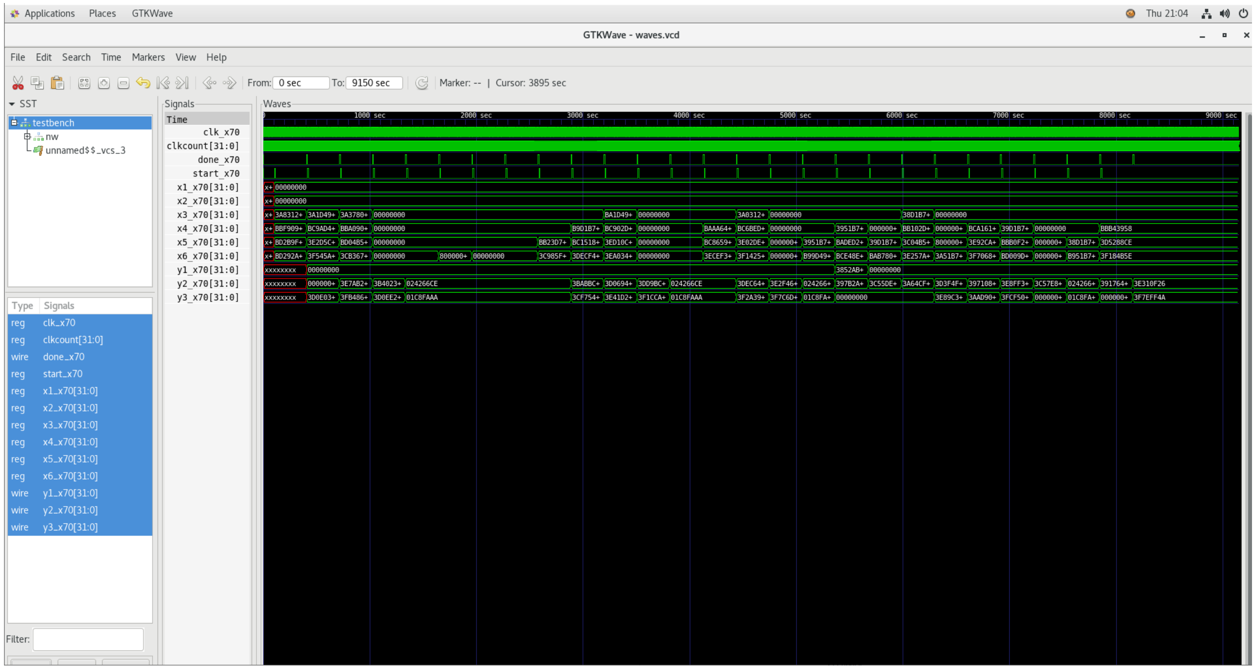

Fig.7 Top-Level Simulation Waveforms

In this simulation log, a new input is given on every pulse on the *start* signal. This start signal acts as a reset for the input layer. When the Neural Network has performed all the MAC operations and has calculated the final outputs, it sends an indication on the *done* signal that is coming from the output neural network. After this *done* indication, the next set of inputs is given and the cycle repeats for all 26 inputs. Here each value of X and Y is represented in single-precision floating point format. This neural network takes 58 clocks to calculate the output for each input combination. The actual values of the input and outputs will be more clear from the simulation log in the next chapter.

4.2 Simulation Log

The simulation prints all the outputs for each neuron to clearly show what is happening on each clock. First, when the inputs are given, the hidden layer displays its operation on the *start* indication. It will multiply and accumulate the input (X1 to X6) with each weight (Wij) one by one to produce the final output of the hidden layer(Hy1 to Hy6). In the end, the ReLU function is applied to limit the negative values to zero. After that, the output layer displays its operation on the *done* indication coming from the hidden layer. Once it is finished, it will apply ReLU and send the final outputs (Y1 to Y3) along with its done indication. The final outputs are displayed at the end of the simulation for every 26 inputs.

```
                         Chronologic VCS (TM)
               Version R-2020.12 -- Thu Oct 20 20:50:41 2022
                  Copyright (c) 1991-2020 by Synopsys Inc.
                            ALL RIGHTS RESERVED

This program is proprietary and confidential information of Synopsys Inc.
and may be used and disclosed only as authorized in a license agreement
controlling such use and disclosure.

The design hasn't changed and need not be recompiled.
If you really want to, delete file simv.daidir/.vcs.timestamp and
run VCS again.

Chronologic VCS simulator copyright 1991-2020
Contains Synopsys proprietary information.
Compiler version R-2020.12; Runtime version R-2020.12;  Oct 20 20:50 2022
NOTE: automatic random seed used: 984974416


###########################

 ##### started hidden layer @t=105

      x1=  0.000000 * w11=  0.289000 w12= -0.142700 w13=  0.135700 w14=
-0.040200
                      hy1=  0.000000 hy2=  0.000000 hy3=  0.000000 hy4=
0.000000
      x2=  0.000000 * w21=  0.007700 w22= -0.001600 w23=  0.024700 w24=
0.012700
                      hy1=  0.000000 hy2=  0.000000 hy3=  0.000000 hy4=
0.000000
      x3=  0.001000 * w31= -0.000300 w32= -0.005500 w33=  0.000900 w34=
-0.000900
                      hy1= -0.000000 hy2= -0.000006 hy3=  0.000001 hy4=
-0.000001
      x4= -0.007600 * w41= -0.843000 w42=  0.045500 w43=  0.048700 w44=
0.100000
                      hy1=  0.006407 hy2= -0.000351 hy3= -0.000369 hy4=
-0.000761
```

```
      x5= -0.041900 * w51= -0.305200 w52=  0.068000 w53= -0.047200 w54=
0.398300
                      hy1=  0.019194 hy2= -0.003200 hy3=  0.001608 hy4=
-0.017450
      x6= -0.041300 * w61= -0.397700 w62=  0.472200 w63= -0.312500 w64=
-0.018400
                      hy1=  0.035619 hy2= -0.022702 hy3=  0.014515 hy4=
-0.016690

      After ReLU      hy1=  0.035619 hy2=  0.000000 hy3=  0.014515 hy4=
-0.016690

 ##### hidden layer done_x70, started output layer @t=285

      x1=  0.035619 * w11= -0.886100 w12= -0.354500 w13=  1.347100
                      y1_x70 = -0.031562  y2_x70= -0.012627  y3_x70=  0.047983
      x2=  0.000000 * w21= -1.811800 w22=  0.608500 w23=  3.505400
                      y1_x70 = -0.031562  y2_x70= -0.012627  y3_x70=  0.047983
      x3=  0.014515 * w31= -1.520500 w32=  1.960700 w33= -2.138600
                      y1_x70 = -0.053632  y2_x70=  0.015832  y3_x70=  0.016942
      x4= -0.016690 * w41=  2.700800 w42=  1.113900 w43= -1.062300
                      y1_x70 = -0.098708  y2_x70= -0.002759  y3_x70=  0.034671

      After ReLU      y1_x70=  0.000000 y2_x70=  0.000000 y3_x70=  0.034671

 ##### Final NN output => y1_x70:0.000000, y2_x70:0.000000, y3_x70:0.034671
(took 58 clocks to execute)

 ##### output layer done_x70 @t=405

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

[Omitted ~ 1000 more similar lines for each input]

```
###########################

 ##### started hidden layer @t=7855

      x1=  0.000000 * w11=  0.289000 w12= -0.142700 w13=  0.135700 w14=
-0.040200
                      hy1=  0.000000 hy2=  0.000000 hy3=  0.000000 hy4=
0.000000
      x2=  0.000000 * w21=  0.007700 w22= -0.001600 w23=  0.024700 w24=
0.012700
                      hy1=  0.000000 hy2=  0.000000 hy3=  0.000000 hy4=
0.000000
      x3=  0.000000 * w31= -0.000300 w32= -0.005500 w33=  0.000900 w34=
-0.000900
                      hy1=  0.000000 hy2=  0.000000 hy3=  0.000000 hy4=
0.000000
      x4= -0.005500 * w41= -0.843000 w42=  0.045500 w43=  0.048700 w44=
0.100000
                      hy1=  0.004636 hy2= -0.000250 hy3= -0.000268 hy4=
-0.000550
```

```
      x5=  0.051400 * w51= -0.305200 w52=  0.068000 w53= -0.047200 w54=
0.398300
                      hy1= -0.011051 hy2=  0.003245 hy3= -0.002694 hy4=
0.019923
      x6=  0.594900 * w61= -0.397700 w62=  0.472200 w63= -0.312500 w64=
-0.018400
                      hy1= -0.247643 hy2=  0.284157 hy3= -0.188600 hy4=
0.008976

      After ReLU      hy1=  0.000000 hy2=  0.284157 hy3=  0.000000 hy4=
0.000000

 ##### hidden layer done_x70, started output layer @t=8035

      x1=  0.000000 * w11= -0.886100 w12= -0.354500 w13=  1.347100
                      y1_x70 =  0.000000  y2_x70=  0.000000  y3_x70=  0.000000
      x2=  0.284157 * w21= -1.811800 w22=  0.608500 w23=  3.505400
                      y1_x70 = -0.514835  y2_x70=  0.172909  y3_x70=  0.996083
      x3=  0.000000 * w31= -1.520500 w32=  1.960700 w33= -2.138600
                      y1_x70 = -0.514835  y2_x70=  0.172909  y3_x70=  0.996083
      x4=  0.000000 * w41=  2.700800 w42=  1.113900 w43= -1.062300
                      y1_x70 = -0.514835  y2_x70=  0.172909  y3_x70=  0.996083

      After ReLU      y1_x70=  0.000000 y2_x70=  0.172909 y3_x70=  0.996083

 ##### Final NN output => y1_x70:0.000000, y2_x70:0.172909, y3_x70:0.996083
(took 58 clocks to execute)

 ##### output layer done_x70 @t=8155

---------------------------



-----------------------
Final Outputs
-----------------------



  0.000000,   0.000000,   0.034671
  0.000000,   0.244823,   1.410359
  0.000000,   0.002932,   0.034884
  0.000000,   0.000000,   0.000000
  0.000000,   0.000000,   0.000000
  0.000000,   0.000000,   0.000000
  0.000000,   0.000000,   0.000000
  0.000000,   0.000000,   0.000000
  0.000000,   0.005241,   0.030192
  0.000000,   0.032857,   0.189279
  0.000000,   0.106316,   0.612457
  0.000000,   0.000000,   0.000000
  0.000000,   0.000000,   0.000000
  0.000000,   0.115426,   0.664935
  0.000000,   0.171168,   0.986050
  0.000000,   0.000000,   0.000000
  0.000050,   0.000240,   0.000000
  0.000000,   0.013054,   0.000000
  0.000000,   0.000873,   0.000000
  0.000000,   0.046707,   0.269066
  0.000000,   0.000230,   0.001324
  0.000000,   0.281153,   1.619643
  0.000000,   0.013178,   0.000000
  0.000000,   0.000000,   0.000000
  0.000000,   0.000144,   0.000000
  0.000000,   0.172909,   0.996083


$finish called from file "../tb.sv", line 114.
$finish at simulation time                 9155
           V C S   S i m u l a t i o n   R e p o r t
Time: 9155
CPU Time:      0.580 seconds;       Data structure size:   0.6Mb
Thu Oct 20 20:50:42 2022
```

5. Conclusion

The given neural network with 6 input neurons, 4 neurons in the hidden layer, and 3 output neurons with all the given synapses weight was successfully implemented in synthesizable HDL using the MAC unit designed as a part of the Mini Project 1. Also, the implementation can be made generic with very few changes to accommodate a neural network with any number of neurons, synapses, and multiple layers. The project helped understand how neural networks can be implemented in the hardware and what advantages it provides over the CPU/GPU implementation. The unit implemented in this project takes the advantage of parallel processing and performs 9 parallel MAC operations, therefore, it can outperform a single-core CPU which can only do one MAC at a time. On the other hand, the resource requirement was very low, and therefore it significantly reduces the power consumption compared to the GPUs. This is very important for implementing Edge Devices running neural networks as they need to do real-time computations with limited power availability.

APPENDIX

Verilog Code of the Design and Testbench

```verilog
////////////////////////////////////////////////////////////////////////////////////////////////
// FILE: top.v
// Description: Top level unit of Neural Network designed as a part of Mini-Project 2 of EE278(fall'22)
// Developed By: Bhavin patel (SJSU-ID: 01954770)
////////////////////////////////////////////////////////////////////////////////////////////////


`include "mac.v"
`include "hidden_layer.v"
`include "output_layer.v"


// 6 input, 4 neurons, 4 output
module Neural_network(
 input wire clk_x70,
 input wire start_x70,
 output reg done_x70,
 input [31:0] x1_x70,
 input [31:0] x2_x70,
 input [31:0] x3_x70,
 input [31:0] x4_x70,
 input [31:0] x5_x70,
 input [31:0] x6_x70,
 output [31:0] y1_x70,
 output [31:0] y2_x70,
 output [31:0] y3_x70
);

wire [31:0] hy1;
wire [31:0] hy2;
wire [31:0] hy3;
wire [31:0] hy4;

wire inp_layer_done;
reg out_layer_reset;
reg in_layer_reset;

initial begin
 out_layer_reset = 1;
 in_layer_reset = 1;
end

// hidden layer
Layer6to4 inp_layer(
 .clk_x70(clk_x70),
 .reset_x70(in_layer_reset),
 .x1_x70(x1_x70),
 .x2_x70(x2_x70),
 .x3_x70(x3_x70),
 .x4_x70(x4_x70),
 .x5_x70(x5_x70),
 .x6_x70(x6_x70),
 .y1_x70(hy1),
 .y2_x70(hy2),
 .y3_x70(hy3),
 .y4_x70(hy4),
 .done_x70(inp_layer_done)
);

// output layer
Layer4to3 out_layer(
 .clk_x70(clk_x70),
 .reset_x70(out_layer_reset),
 .x1_x70(hy1),
 .x2_x70(hy2),
 .x3_x70(hy3),
 .x4_x70(hy4),
 .y1_x70(y1_x70),
 .y2_x70(y2_x70),
 .y3_x70(y3_x70),
 .done_x70(done_x70)
);

always@(posedge start_x70) begin

 $display("\n\n################################\n");

 // step 1: restart input layer, and put output layer on hold
 $display(" ###### started hidden layer @t=%0t\n", $time);
 out_layer_reset = 1;
 in_layer_reset = 1;
 @(posedge clk_x70);
 in_layer_reset = 0;

 // step 2: let input layer finish
 @(posedge inp_layer_done);
 $display("\n ###### hidden layer done_x70, started output layer @t=%0t\n", $time);

 // step 3: restart output layer, and put input layer on hold
 in_layer_reset = 1;
 out_layer_reset = 1;
 @(posedge clk_x70);
 out_layer_reset = 0;

 // step 4: let output layer finish, and put both layer on hold
 @(posedge done_x70);
 in_layer_reset = 1;
 out_layer_reset = 1;
 $display("\n ###### output layer done_x70 @t=%0t", $time);
 $display("\n--------------------------\n\n");

end

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////////////
// FILE: hidden_layer.v
// Description: Output Layer of Neural Network designed as a part of Mini-Project 2 of EE278(fall'22)
// Developed By: Bhavin patel (SJSU-ID: 01954770)
//////////////////////////////////////////////////////////////////////////////////////////

// 6 input, 4 neurons, 4 output
module Layer6to4(
 input wire clk_x70,
 input wire reset_x70,
 output reg done_x70,
 input [31:0] x1_x70,
 input [31:0] x2_x70,
 input [31:0] x3_x70,
 input [31:0] x4_x70,
 input [31:0] x5_x70,
 input [31:0] x6_x70,
 output reg [31:0] y1_x70,
 output reg [31:0] y2_x70,
 output reg [31:0] y3_x70,
 output reg [31:0] y4_x70
);

 wire [31:0] y1_out, y2_out, y3_out, y4_out;
 reg [31:0] mux_x_out, mux_w1_out, mux_w2_out, mux_w3_out, mux_w4_out;
 reg [3:0] inp_neuron_sel;
 reg [1:0] mac_stage;
 wire clkout;


 // Instances of MAC units
 Mac m1(clkout, reset_x70, mux_x_out, mux_w1_out, y1_out);
 Mac m2(clkout, reset_x70, mux_x_out, mux_w2_out, y2_out);
 Mac m3(clkout, reset_x70, mux_x_out, mux_w3_out, y3_out);
 Mac m4(clkout, reset_x70, mux_x_out, mux_w4_out, y4_out);

 // Weights from Mini-Project 2 document
 //  0.2890   -0.1427    0.1375   -0.0402
 //  0.0077   -0.0016    0.0247    0.0127
 // -0.0003   -0.0055    0.0009   -0.0009
 // -0.0843    0.0455    0.0487    0.1000
 // -0.3052   -0.0680   -0.0472    0.3983
 // -0.3977   -0.4722   -0.3125   -0.0184
 // -0.2209   -0.2556   -0.1011   -0.2383

 // axon weights[i][j] i=input side neuron j=output side neuron
 reg [31:0] weights[4][6] = '{
   '{$shortrealtobits(0.2890), $shortrealtobits(0.0077), $shortrealtobits(-0.0003), $shortrealtobits(-0.843), $shortrealtobits(-0.3052), $shortrealtobits(-0.3977)}, // weights to j1 from i[1-6]
   '{$shortrealtobits(-0.1427), $shortrealtobits(-0.0016), $shortrealtobits(-0.0055), $shortrealtobits(0.0455), $shortrealtobits(0.0680), $shortrealtobits(0.4722)}, // weights to j2 from i[1-6]
   '{$shortrealtobits(0.1357), $shortrealtobits(0.0247), $shortrealtobits(0.0009), $shortrealtobits(0.0487), $shortrealtobits(-0.0472), $shortrealtobits(-0.3125)}, // weights to j3 from i[1-6]
```

```verilog
  '{$shortrealtobits(-0.0402), $shortrealtobits(0.0127), $shortrealtobits(-0.0009), $shortrealtobits(0.1000), $shortrealtobits(0.3983), $shortrealtobits(-0.0184)}  // weights to j4 from i[1-6]
};


// clock for MACs
assign clkout = clk_x70;


// initialize
initial begin
  inp_neuron_sel = 0;
  mac_stage = 0;
  done_x70 = 0;
end


always @(posedge clk_x70) begin

  if(reset_x70) begin
    inp_neuron_sel = 0;
    mac_stage = 0;
    done_x70 = 0;
  end

  else begin

    // give X1 to X6 one-by-one
    if(mac_stage == 0) begin
      case(inp_neuron_sel)
      0: begin mux_x_out = x1_x70; mux_w1_out = weights[0][0]; mux_w2_out = weights[1][0]; mux_w3_out = weights[2][0]; mux_w4_out = weights[3][0]; end
      1: begin mux_x_out = x2_x70; mux_w1_out = weights[0][1]; mux_w2_out = weights[1][1]; mux_w3_out = weights[2][1]; mux_w4_out = weights[3][1]; end
      2: begin mux_x_out = x3_x70; mux_w1_out = weights[0][2]; mux_w2_out = weights[1][2]; mux_w3_out = weights[2][2]; mux_w4_out = weights[3][2]; end
      3: begin mux_x_out = x4_x70; mux_w1_out = weights[0][3]; mux_w2_out = weights[1][3]; mux_w3_out = weights[2][3]; mux_w4_out = weights[3][3]; end
      4: begin mux_x_out = x5_x70; mux_w1_out = weights[0][4]; mux_w2_out = weights[1][4]; mux_w3_out = weights[2][4]; mux_w4_out = weights[3][4]; end
      5: begin mux_x_out = x6_x70; mux_w1_out = weights[0][5]; mux_w2_out = weights[1][5]; mux_w3_out = weights[2][5]; mux_w4_out = weights[3][5]; end
      endcase
      $display("\tx%0d=%10f * w%0d1=%10f w%0d2=%10f w%0d3=%10f w%0d4=%10f",
      (inp_neuron_sel + 1), $bitstoshortreal(mux_x_out),
      (inp_neuron_sel + 1), $bitstoshortreal(mux_w1_out),
      (inp_neuron_sel + 1), $bitstoshortreal(mux_w2_out),
      (inp_neuron_sel + 1), $bitstoshortreal(mux_w3_out),
      (inp_neuron_sel + 1), $bitstoshortreal(mux_w4_out)
      );

    end
    mac_stage += 1;

    // 1 MAC operation takes 3 clocks, set the output when all X1 to X6 are MAC'ed
    if(mac_stage == 3) begin
```

```
        inp_neuron_sel += 1;
        mac_stage = 0;
        $display("\t                hy1=%10f hy2=%10f hy3=%10f hy4=%10f", $bitstoshortreal(y1_out),
$bitstoshortreal(y2_out), $bitstoshortreal(y3_out), $bitstoshortreal(y4_out));
        if(inp_neuron_sel == 6) begin

        // ReLU activation function
        y1_x70 = (y1_out[31] == 1) ? 'b0 : y1_out;
        y2_x70 = (y2_out[31] == 1) ? 'b0 : y2_out;
        y3_x70 = (y3_out[31] == 1) ? 'b0 : y3_out;
        y4_x70 = (y3_out[31] == 1) ? 'b0 : y4_out;

        inp_neuron_sel = 0;
        done_x70 = 1;
        $display("\n\tAfter ReLU      hy1=%10f hy2=%10f hy3=%10f hy4=%10f",
$bitstoshortreal(y1_x70), $bitstoshortreal(y2_x70), $bitstoshortreal(y3_x70), $bitstoshortreal(y4_x70));
        end
      end
    end

  end

endmodule
```

```
/////////////////////////////////////////////////////////////////////////////////////////
// FILE: output_layer.v
// Description: Hidden Layer of Neural Network designed as a part of Mini-Project 2 of EE278(fall'22)
// Developed By: Bhavin patel (SJSU-ID: 01954770)
/////////////////////////////////////////////////////////////////////////////////////////


// 4 input, 3 neurons, 3 output
module Layer4to3(
 input wire clk_x70,
 input wire reset_x70,
 output reg done_x70,
 input [31:0] x1_x70,
 input [31:0] x2_x70,
 input [31:0] x3_x70,
 input [31:0] x4_x70,
 output reg [31:0] y1_x70,
 output reg [31:0] y2_x70,
 output reg [31:0] y3_x70
);

 wire [31:0] y1_out, y2_out, y3_out;
 reg [31:0] mux_x_out, mux_w1_out, mux_w2_out, mux_w3_out;
 reg [3:0] inp_neuron_sel;
 reg [1:0] mac_stage;
 wire clkout;


 // Instances of MAC units
 Mac m1(clkout, reset_x70, mux_x_out, mux_w1_out, y1_out);
 Mac m2(clkout, reset_x70, mux_x_out, mux_w2_out, y2_out);
 Mac m3(clkout, reset_x70, mux_x_out, mux_w3_out, y3_out);


 // Weights from Mini-Project 2 document
 // -0.8861   -0.3545    1.3471
 // -1.8118    0.6085    3.5054
 // -1.5205    1.9607   -2.1386
 //  2.7008    1.1139   -1.0623
 // -1.0033   -0.8877   -0.8463

 // axon weights[i][j] i=input side neuron j=output side neuron
 reg [31:0] weights[3][4] = '{
   '{$shortrealtobits(-0.8861), $shortrealtobits(-1.8118), $shortrealtobits(-1.5205),
$shortrealtobits(2.7008)}, // weights to j1 from i[1-4]
   '{$shortrealtobits(-0.3545), $shortrealtobits(0.6085), $shortrealtobits(1.9607),
$shortrealtobits(1.1139)}, // weights to j2 from i[1-4]
   '{$shortrealtobits(1.3471), $shortrealtobits(3.5054), $shortrealtobits(-2.1386),
$shortrealtobits(-1.0623)}  // weights to j3 from i[1-4]
 };


 // clock for MACs
 assign clkout = clk_x70;


```

```verilog
// initialize
initial begin
  inp_neuron_sel = 0;
  mac_stage = 0;
  done_x70 = 0;
end

// main
always @(posedge clk_x70) begin

  if(reset_x70) begin
    inp_neuron_sel = 0;
    mac_stage = 0;
    done_x70 = 0;
  end

  else begin

    // give X1 to X4 one-by-one
    if(mac_stage == 0) begin
      case(inp_neuron_sel)
      0: begin mux_x_out = x1_x70; mux_w1_out = weights[0][0]; mux_w2_out = weights[1][0]; mux_w3_out = weights[2][0]; end
      1: begin mux_x_out = x2_x70; mux_w1_out = weights[0][1]; mux_w2_out = weights[1][1]; mux_w3_out = weights[2][1]; end
      2: begin mux_x_out = x3_x70; mux_w1_out = weights[0][2]; mux_w2_out = weights[1][2]; mux_w3_out = weights[2][2]; end
      3: begin mux_x_out = x4_x70; mux_w1_out = weights[0][3]; mux_w2_out = weights[1][3]; mux_w3_out = weights[2][3]; end
      endcase
      $display("\tx%0d=%10f * w%0d1=%10f w%0d2=%10f w%0d3=%10f",
      (inp_neuron_sel + 1), $bitstoshortreal(mux_x_out),
      (inp_neuron_sel + 1), $bitstoshortreal(mux_w1_out),
      (inp_neuron_sel + 1), $bitstoshortreal(mux_w2_out),
      (inp_neuron_sel + 1), $bitstoshortreal(mux_w3_out)
      );

    end
    mac_stage += 1;

    // 1 MAC operation takes 3 clocks, set the output when all X1 to X4 are MAC'ed
    if(mac_stage == 3) begin
      inp_neuron_sel += 1;
      mac_stage = 0;
      $display("\t                y1_x70 =%10f  y2_x70=%10f  y3_x70=%10f", $bitstoshortreal(y1_out), $bitstoshortreal(y2_out), $bitstoshortreal(y3_out));
      if(inp_neuron_sel == 4) begin

      // ReLU activation function
      y1_x70 = (y1_out[31] == 1) ? 'b0 : y1_out;
      y2_x70 = (y2_out[31] == 1) ? 'b0 : y2_out;
      y3_x70 = (y3_out[31] == 1) ? 'b0 : y3_out;

      inp_neuron_sel = 0;
      done_x70 = 1;
```

```
        $display("\n\tAfter ReLU      y1_x70=%10f y2_x70=%10f y3_x70=%10f",
$bitstoshortreal(y1_x70), $bitstoshortreal(y2_x70), $bitstoshortreal(y3_x70));
        end
      end
    end

  end

endmodule


```

```
/////////////////////////////////////////////////////////////////////////////////////////////
// FILE: testbench.v
// Description: Testbench of Neural Network designed as a part of Mini-Project 2 of EE278(fall'22)
// Developed By: Bhavin patel (SJSU-ID: 01954770)
/////////////////////////////////////////////////////////////////////////////////////////////

`include "top.v"

module testbench;

 reg clk_x70;
 reg start_x70;
 wire done_x70;
 reg [31:0] x1_x70;
 reg [31:0] x2_x70;
 reg [31:0] x3_x70;
 reg [31:0] x4_x70;
 reg [31:0] x5_x70;
 reg [31:0] x6_x70;
 wire [31:0] y1_x70;
 wire [31:0] y2_x70;
 wire [31:0] y3_x70;

 int clkcount;
 shortreal outputs [][];

 // Inputs from Mini-Project 2 document
 shortreal inputs [][] = '{
   '{0, 0,     0.0010,    -0.0076,    -0.0419,    -0.0413},
   '{0, 0,     0.0006,    -0.0189,     0.1693,     0.8295},
   '{0, 0,     0.0007,    -0.0049,    -0.0324,     0.0219},
   '{0, 0,          0,          0,          0,          0},
   '{0, 0,          0,          0,          0,          0},
   '{0, 0,          0,     0.0000,     0.0000,    -0.0000},
   '{0, 0,          0,          0,          0,          0},
   '{0, 0,          0,          0,          0,          0},
   '{0, 0,          0,     0.0000,    -0.0025,     0.0186},
   '{0, 0,          0,    -0.0004,    -0.0091,     0.1157},
   '{0, 0,    -0.0006,    -0.0176,     0.4083,     0.3129},
   '{0, 0,          0,          0,          0,          0},
   '{0, 0,          0,          0,          0,          0},
   '{0, 0,          0,    -0.0013,    -0.0164,     0.4042},
   '{0, 0,     0.0005,    -0.0144,     0.1278,     0.5787},
   '{0, 0,          0,          0,          0,          0},
   '{0, 0,          0,          0,     0.0002,    -0.0003},
   '{0, 0,          0,     0.0002,    -0.0017,    -0.0279},
   '{0, 0,          0,     0.0000,     0.0004,    -0.0014},
   '{0, 0,     0.0001,    -0.0022,     0.0081,     0.1616},
   '{0, 0,          0,     0.0000,    -0.0000,     0.0008},
   '{0, 0,     0.0000,    -0.0197,     0.2867,     0.9391},
   '{0, 0,     0.0000,     0.0004,    -0.0054,    -0.0314},
   '{0, 0,          0,          0,     0.0000,     0.0000},
   '{0, 0,          0,     0.0000,     0.0001,    -0.0002},
   '{0, 0,          0,    -0.0055,     0.0514,     0.5949}
 };

```

```systemverilog

// neural network design
Neural_network nw(.*);

// clock driver and counter
always begin
  clkcount += 1;
  #5 clk_x70 = ~clk_x70;
end
always @(negedge start_x70) clkcount = 0;

// main testbench code
initial begin
  $dumpfile("waves.vcd");
  $dumpvars;
  outputs = new[inputs.size()];

  clk_x70 = 0;
  start_x70 = 0;
  #100;

  for(int i = 0; i < inputs.size(); i++) begin

    // give inputs to the designed Neural Network
    x1_x70 = $shortrealtobits(inputs[i][0]);
    x2_x70 = $shortrealtobits(inputs[i][1]);
    x3_x70 = $shortrealtobits(inputs[i][2]);
    x4_x70 = $shortrealtobits(inputs[i][3]);
    x5_x70 = $shortrealtobits(inputs[i][4]);
    x6_x70 = $shortrealtobits(inputs[i][5]);

    // apply start
    @(posedge(clk_x70));
    start_x70 = 1;
    @(posedge(clk_x70));
    start_x70 = 0;

    // wait for done
    @(posedge(done_x70));

    // output displays
    $display("\n ##### Final NN output => y1_x70:%f, y2_x70:%f, y3_x70:%f (took %0d clocks to
execute)", $bitstoshortreal(y1_x70), $bitstoshortreal(y2_x70), $bitstoshortreal(y3_x70), clkcount);
    outputs[i] = '{$bitstoshortreal(y1_x70), $bitstoshortreal(y2_x70), $bitstoshortreal(y3_x70)};

   end

   // Show final outputs at the end
   #1000;
   $display("-----------------------");
   $display("Final Outputs");
   $display("-----------------------");
   $display("\n");
   foreach(outputs[i]) begin
     $display("%10f, %10f, %10f", outputs[i][0], outputs[i][1], outputs[i][2]);
   end
```

```

    $display("\n\n");
    $finish;
  end

endmodule



```

```verilog
/////////////////////////////////////////////////////////////////////////////////////////
// FILE: mac.v
// Description: MAC unit designed as a part of Mini-Project 1 of EE278(fall'22)
// Developed By: Bhavin patel (SJSU-ID: 01954770)
/////////////////////////////////////////////////////////////////////////////////////////

`include "fp_adder.v"
`include "fp_mult.v"

module Mac(
 input wire clk_x70,
 input wire reset_x70,
 input wire [31:0] inp1_x70,
 input wire [31:0] inp2_x70,
 output wire [31:0] out_x70
);

wire [31:0] mul_out_x70;
wire [31:0] sum_out_x70;
reg [31:0] sum_buff = 0;

assign out_x70 = sum_out_x70;
wire underflow_x70;
wire overflow_x70;

reg mul_clk_x70, add_clk_x70, buf_clk_x70;
integer clk_x70num = 0;

always @(posedge reset_x70) begin
 sum_buff = 0;
end

//always @(mul_out_x70)begin
//end

always@(posedge clk_x70) begin
 if(reset_x70) begin
   sum_buff = 0;
   clk_x70num = 0;
 end
 else begin
   case(clk_x70num)
     0: begin
       mul_clk_x70 = 1; add_clk_x70 = 0; buf_clk_x70 = 0;
     end
     1: begin
       //$display("mul_out: %f, inp1:%f, inp2:%f", $bitstoshortreal(mul_out_x70),
$bitstoshortreal(inp1_x70), $bitstoshortreal(inp2_x70));
       mul_clk_x70 = 0; add_clk_x70 = 1; buf_clk_x70 = 0;
     end
     2: begin
       mul_clk_x70 = 0; add_clk_x70 = 0; buf_clk_x70 = 1;
       sum_buff = sum_out_x70;
     end
   endcase

   clk_x70num++;
   if(clk_x70num > 2)
     clk_x70num = 0;
 end
end

fp_adder add1(.clk_x70(add_clk_x70), .inp1_x70(mul_out_x70), .inp2_x70(sum_buff),
.sum_x70(sum_out_x70));
fp_mul_unpipe mult( .inp1_x70(inp1_x70), .inp2_x70(inp2_x70), .out_x70(mul_out_x70),
.underflow_x70(underflow_x70),.overflow_x70(overflow_x70));
endmodule

```

```verilog
/////////////////////////////////////////////////////////////////////////////////////////
// FILE: fp_mult.v
// Description: Floating Point Multiplier designed as a part of Mini-Project 1 of EE278(fall'22)
// Developed By: Bhavin patel (SJSU-ID: 01954770)
/////////////////////////////////////////////////////////////////////////////////////////

module sign_bit(
 output wire sign_x70,
 input wire [31:0] in1_x70,
 input wire [31:0] in2_x70
);
 xor (sign_x70, in1_x70 [31], in2_x70 [31]);
endmodule


// simply truncates multiplication of two mantissa into 240bits
module normalize(
 output wire[22:0] adj_mantissa_x70,
 output wire norm_flag_x70,
 input wire[47:0] prdt_x70
);

 and(norm_flag_x70, prdt_x70[47], 1'b1);
 // if leading 1 is at pos 47 then needs normalization, otherwise not

 wire [1:0][22:0] result;
 assign result[0] = prdt_x70[45:23];
 assign result[1] = prdt_x70[46:24];
 assign adj_mantissa_x70 = {result[norm_flag_x70 + 0]};
endmodule

//8 bit Ripple-carry adder
module ripple_8(
output wire[7:0] sum_x70,
output wire cout_x70,
input wire[7:0] in1_x70,
input wire[7:0] in2_x70,
input wire cin_x70
);
wire c1_x70,c2_x70,c3_x70,c4_x70,c5_x70,c6_x70,c7_x70;
full_adder FA1(sum_x70[0],c1_x70,in1_x70[0],in2_x70[0],cin_x70);
full_adder FA2(sum_x70[1],c2_x70,in1_x70[1],in2_x70[1],c1_x70);
full_adder FA3(sum_x70[2],c3_x70,in1_x70[2],in2_x70[2],c2_x70);
full_adder FA4(sum_x70[3],c4_x70,in1_x70[3],in2_x70[3],c3_x70);
full_adder FA5(sum_x70[4],c5_x70,in1_x70[4],in2_x70[4],c4_x70);
full_adder FA6(sum_x70[5],c6_x70,in1_x70[5],in2_x70[5],c5_x70);
full_adder FA7(sum_x70[6],c7_x70,in1_x70[6],in2_x70[6],c6_x70);
full_adder FA8(sum_x70[7],cout_x70,in1_x70[7],in2_x70[7],c7_x70);
endmodule

//1 bit Full Adder
module full_adder(
 output wire sum_x70,
 output wire cout_x70,
 input wire in1_x70,
 input wire in2_x70,
 input wire cin_x70
);
wire temp1_x70;
wire temp2_x70;
wire temp3_x70;
xor(sum_x70,in1_x70,in2_x70,cin_x70);
and(temp1_x70,in1_x70,in2_x70);
and(temp2_x70,in1_x70,cin_x70);
and(temp3_x70,in2_x70,cin_x70);
or(cout_x70,temp1_x70,temp2_x70,temp3_x70);
endmodule

//1 bit subtractor with subtrahend = 1
module full_subtractor_sub1(
 output wire diff_x70, //diff_x70erence
 output wire bout_x70,//borrow out
 input wire min_x70,//min_x70uend
 input wire bin_x70//borrow in
);

 //Here, the subtrahend is always 1. We can implement it
 xnor (diff_x70,min_x70, bin_x70);
 or (bout_x70, ~min_x70, bin_x70);
endmodule


//1 bit subtractor with subtrahend = 0
module full_subtractor_sub0(
 output wire diff_x70, //diff_x70erence
 output wire bout_x70, //borrow out
 input wire min_x70, //min_x70uend
 input wire bin_x70 //borrow in
);
 //Here, the subtrahend is always 0.We can implement it
 xor (diff_x70,min_x70, bin_x70);
 and (bout_x70, ~min_x70, bin_x70);
endmodule

module subtractor_9(
 output wire [8:0] diff_x70,
 output wire bout_x70,
 input wire [8:0] min_x70,
 input wire bin_x70
 );

  wire b1, b2,b3,b4, b5, b6,b7,b8;
  full_subtractor_sub1 sub1 (diff_x70 [0], b1,min_x70 [0], bin_x70);
  full_subtractor_sub1 sub2 (diff_x70 [1], b2,min_x70 [1],b1);
  full_subtractor_sub1 sub3 (diff_x70 [2], b3,min_x70 [2], b2);
  full_subtractor_sub1 sub4 (diff_x70 [3], b4,min_x70 [3],b3);
  full_subtractor_sub1 sub5 (diff_x70 [4], b5,min_x70 [4], b4);
  full_subtractor_sub1 sub6 (diff_x70 [5], b6,min_x70 [5], b5);
  full_subtractor_sub1 sub7 (diff_x70 [6], b7,min_x70 [6],b6);
 full_subtractor_sub0 sub8 (diff_x70 [7], b8,min_x70 [7],b7);
 full_subtractor_sub0 sub9 (diff_x70 [8], bout_x70,min_x70[8], b8);

```

```
endmodule

module block(
 output wire ppo_x70, //output partial product term
 output wire cout_x70, //output carry out
 output wire mout_x70, //output multiplicand term

 input wire min_x70, //input multiplicand term
 input wire ppi_x70, //input partial product term
 input wire q_x70, //input multiplier term
 input wire cin //input carry in
);
wire temp;
and (temp,min_x70,q_x70);
full_adder FA_x70(ppo_x70, cout_x70,ppi_x70, temp, cin); // connected to : a, b, cin, sum, cout_x70
or (mout_x70,min_x70, 1'b0);
endmodule


module row(
 output wire [23:0] ppo_x70,
 output wire [23:0] mout_x70,
 output wire sum,
 input wire [23:0] min_x70,
 input wire [23:0] ppi_x70,
 input wire q_x70
);

wire c1, c2, c3, c4, c5, c6, c7,c8, c9, c10;
wire c11, c12, c13, c14, c15, c16, c17, c18, c19, c20;
wire c21, c22, c23;

block b1 (sum,c1,mout_x70 [0],min_x70[0],ppi_x70[0],q_x70,1'b0);
block b2 (ppo_x70[0], c2, mout_x70 [1], min_x70 [1], ppi_x70[1], q_x70, c1);
block b3 (ppo_x70[1], c3, mout_x70 [2], min_x70 [2], ppi_x70 [2], q_x70, c2);
block b4 (ppo_x70[2], c4, mout_x70 [3], min_x70 [3], ppi_x70 [3], q_x70, c3);
block b5 (ppo_x70[3], c5, mout_x70 [4], min_x70 [4], ppi_x70 [4], q_x70, c4);
block b6 (ppo_x70[4], c6, mout_x70 [5], min_x70 [5], ppi_x70 [5], q_x70, c5);
block b7 (ppo_x70[5], c7, mout_x70 [6], min_x70 [6], ppi_x70 [6], q_x70, c6);
block b8 (ppo_x70[6], c8, mout_x70 [7], min_x70 [7], ppi_x70[7], q_x70, c7);
block b9 (ppo_x70[7], c9, mout_x70 [8], min_x70 [8], ppi_x70 [8], q_x70, c8);
block b10(ppo_x70[8], c10, mout_x70 [9], min_x70[9], ppi_x70 [9], q_x70, c9);
block b11(ppo_x70[9], c11, mout_x70 [10], min_x70 [10], ppi_x70 [10], q_x70, c10);
block b12(ppo_x70[10], c12, mout_x70 [11], min_x70 [11], ppi_x70 [11], q_x70, c11);
block b13(ppo_x70[11], c13, mout_x70 [12], min_x70 [12], ppi_x70 [12], q_x70, c12);
block b14(ppo_x70[12], c14, mout_x70 [13], min_x70 [13], ppi_x70 [13], q_x70, c13);
block b15(ppo_x70[13], c15, mout_x70 [14], min_x70 [14], ppi_x70 [14], q_x70, c14);
block b16(ppo_x70[14], c16, mout_x70 [15], min_x70 [15], ppi_x70 [15], q_x70, c15);
block b17(ppo_x70[15], c17, mout_x70 [16], min_x70 [16], ppi_x70 [16], q_x70, c16);
block b18(ppo_x70[16], c18, mout_x70 [17], min_x70 [17], ppi_x70 [17], q_x70, c17);
block b19(ppo_x70[17], c19, mout_x70 [18], min_x70 [18], ppi_x70 [18], q_x70, c18);
block b20(ppo_x70[18], c20, mout_x70 [19], min_x70 [19], ppi_x70 [19], q_x70, c19);
block b21(ppo_x70[19], c21, mout_x70 [20], min_x70 [20], ppi_x70 [20], q_x70, c20);
block b22(ppo_x70[20], c22, mout_x70 [21], min_x70 [21], ppi_x70 [21], q_x70, c21);
block b23(ppo_x70[21], c23, mout_x70 [22], min_x70 [22], ppi_x70 [22], q_x70, c22);
block b24(ppo_x70[22], ppo_x70 [23], mout_x70 [23], min_x70[23], ppi_x70 [23], q_x70, c23);
```

```verilog
endmodule


module product(
 output wire [47:0] sum,
 input wire [23:0] min_x70,
 input wire [23:0]q_x70
);
wire [23:0] temp1,temp2, temp3,temp4, temp5,temp6, temp7, temp8, temp9,temp10;
wire [23:0] temp11,temp12, temp13,temp14,temp15,temp16, temp17,temp18, temp19, temp20;
wire [23:0] temp21,temp22,temp23,temp24;

wire [23:0] ptemp1, ptemp2, ptemp3,ptemp4, ptemp5, ptemp6,ptemp7,ptemp8,ptemp9,ptemp10;
wire [23:0] ptemp11,ptemp12,ptemp13,ptemp14,ptemp15,ptemp16,ptemp17,ptemp18,ptemp19,ptemp20;
wire [23:0] ptemp21, ptemp22,ptemp23;

row r1 (ptemp1, temp1, sum[0], min_x70, 24'h000000, q_x70[0]);
row r2 (ptemp2, temp2, sum[1], temp1, ptemp1, q_x70[1]);
row r3 (ptemp3, temp3, sum[2], temp2, ptemp2, q_x70[2]);
row r4 (ptemp4, temp4, sum[3], temp3, ptemp3, q_x70[3]);
row r5 (ptemp5, temp5, sum[4], temp4, ptemp4, q_x70[4]);
row r6 (ptemp6, temp6, sum[5], temp5, ptemp5, q_x70[5]);
row r7 (ptemp7, temp7, sum[6], temp6, ptemp6, q_x70[6]);
row r8 (ptemp8, temp8, sum[7], temp7, ptemp7, q_x70[7]);
row r9 (ptemp9, temp9, sum[8], temp8, ptemp8, q_x70[8]);
row r10(ptemp10, temp10, sum [9], temp9, ptemp9, q_x70[9]);
row r11(ptemp11, temp11, sum [10], temp10, ptemp10, q_x70[10]);
row r12(ptemp12, temp12, sum[11], temp11, ptemp11, q_x70[11]);
row r13(ptemp13, temp13, sum [12], temp12, ptemp12, q_x70[12]);
row r14(ptemp14, temp14, sum [13], temp13, ptemp13, q_x70[13]);
row r15(ptemp15, temp15, sum [14], temp14, ptemp14, q_x70[14]);
row r16(ptemp16, temp16, sum[15], temp15, ptemp15, q_x70[15]);
row r17(ptemp17, temp17, sum[16], temp16, ptemp16, q_x70[16]);
row r18 (ptemp18, temp18, sum [17], temp17, ptemp17, q_x70[17]);
row r19 (ptemp19, temp19, sum [18], temp18, ptemp18, q_x70[18]);
row r20 (ptemp20, temp20, sum [19], temp19, ptemp19, q_x70[19]);
row r21 (ptemp21, temp21, sum [20], temp20, ptemp20, q_x70[20]);
row r22(ptemp22, temp22, sum [21], temp21, ptemp21, q_x70[21]);
row r23(ptemp23, temp23, sum [22], temp22, ptemp22, q_x70[22]);
row r24(sum[47:24], temp24, sum [23], temp23, ptemp23, q_x70[23]);
endmodule

//Control module to drive and regulate required modules in order
module fp_mul_unpipe(
 input wire [31:0] inp1_x70,
 input wire [31:0] inp2_x70,
 output wire [31:0] out_x70,
 output wire underflow_x70,
 output wire overflow_x70
);
 wire sign;
 wire [7:0] exp1;
 wire [7:0] exp2;
 wire [7:0] exp_out_x70;
 wire [7:0] test_exp;
 wire [22:0] manti;
 wire [22:0] mant2;
 wire [22:0] mant_out_x70;
 wire [31:0] tmp_out_x70;

 sign_bit sign_bit1_x70 (sign, inp1_x70, inp2_x70);

 wire [7:0] temp1;
 wire dummy; //to connect unused cout_x70 ports of adder
 wire carry;
 wire [8:0] add_out;
 wire [8:0] sub_temp;
 wire [7:0] sub_temp_low;
 wire [7:0] sub_temp_high;
 reg zero = 0;
 reg one = 1;


 assign exp1 = inp1_x70[30:23];
 assign exp2 = inp2_x70[30:23];
 assign sub_temp_low = sub_temp[7:0];
 assign sub_temp_high = sub_temp[8];
 assign add_out = {carry, temp1};

 ripple_8 rip1_x70 (temp1, carry, exp1, exp2, zero);
 subtractor_9 sub1_x70 (sub_temp, underflow_x70, add_out, zero);

 //if there is a carry out_x70 => underflow_x70
 and (overflow_x70, sub_temp_high, one); //if the exponent has more than 8

 //taking product of mantissa:
 wire [47:0] prdt;
 wire [23:0] p_in1;
 wire [23:0] p_in2;
 assign p_in1 = {1'b1, inp1_x70[22:0]};
 assign p_in2 = {1'b1, inp2_x70[22:0]};
 product pi_x70(prdt, p_in1, p_in2);

 // normalizing mantissa
 wire norm_flag;
 wire [22:0] adj_mantissa;
 normalize normi (adj_mantissa,norm_flag,prdt);
 wire [7:0] norm_in2;
 assign norm_in2 = {7'b0, norm_flag};
 ripple_8 ripple_norm_x70(test_exp, dummy, sub_temp_low, norm_in2, zero);

 assign tmp_out_x70 [31] = sign;
 assign tmp_out_x70 [30:23] = test_exp;
 assign tmp_out_x70 [22:0] = adj_mantissa;
 assign out_x70 = (inp1_x70[30:0] == 0 || inp2_x70[30:0] == 0) ? 'b0 : tmp_out_x70;

endmodule
```

```verilog
////////////////////////////////////////////////////////////////////////////////////////////
// FILE: fp_adder.v
// Description: Floating Point Adder designed as a part of Mini-Project 1 of EE278(fall'22)
// Developed By: Bhavin patel (SJSU-ID: 01954770)
////////////////////////////////////////////////////////////////////////////////////////////

module fp_adder(
 input wire [31:0] inp1_x70,
 input wire [31:0] inp2_x70,
  input wire [0:0] clk_x70,
  output reg [31:0] sum_x70
 );

 reg [7:0] exponent_1, exponent_2;
 reg [23:0] mantissa_1, mantissa_2;
 reg [7:0] new_exponent;
 reg [7:0] exponent_final;
 reg [22:0] mantissa_final;
 reg signed [23:0] mantissa_sum;
 reg [23:0] shifted_mantissa_1,shifted_mantissa_2;
 reg [24:0] add_mantissa_sum;
 reg sign_1, sign_2, res_sign;
 reg [7:0] diff;

 always @(posedge clk_x70)
 begin
  // get exponent and mantissa
  exponent_1=inp1_x70[30:23];
  exponent_2=inp2_x70[30:23];
  mantissa_1={1'b1,inp1_x70[22:0]};
  mantissa_2={1'b1,inp2_x70[22:0]};
  sign_1 = inp1_x70[31];
  sign_2 = inp2_x70[31];

  // adjust the mantissa and align the exponent
  if(exponent_1 == exponent_2)
  begin
    shifted_mantissa_1=mantissa_1;
    shifted_mantissa_2=mantissa_2;
    new_exponent=exponent_1;
  end
  else if(exponent_1>exponent_2)
  begin
    diff=exponent_1-exponent_2;
    shifted_mantissa_1=mantissa_1;
    shifted_mantissa_2=(mantissa_2>>>diff);
    new_exponent=exponent_1;
  end
  else if(exponent_2>exponent_1)
  begin
    diff=exponent_2-exponent_1;
    shifted_mantissa_2=mantissa_2;
    shifted_mantissa_1=(mantissa_1>>>diff);
    new_exponent=exponent_2;
  end

```

```verilog

//Compare the two aligned mantissas and determine which is the
//smaller of the two. Take 2’s complement of the smaller mantissa
//if the signs of the two numbers are different.
res_sign = sign_1;
if(sign_1 ^ sign_2) begin // different signs
  if(shifted_mantissa_1 < shifted_mantissa_2) begin
    res_sign = sign_2;
    shifted_mantissa_1 = (shifted_mantissa_1 ^ 24'hFF_FFFF) + 1;
  end
  else begin
    res_sign = sign_1;
    shifted_mantissa_2 = (shifted_mantissa_2 ^ 24'hFF_FFFF) + 1;
  end
end


//Add the two mantissa. Then, determine the amount of shifts
//required and the corresponding direction to normalize the result.
exponent_final=new_exponent;
add_mantissa_sum = shifted_mantissa_1 + shifted_mantissa_2;
//mantissa_final=add_mantissa_sum[22:0];
if(sign_1 ^ sign_2 == 0) begin
  if(add_mantissa_sum[24] == 1) begin //carry
    exponent_final += 1;
    add_mantissa_sum = add_mantissa_sum >> 1;
  end
end

//Stage 5: Shift the mantissa to the required direction by the required amount. Adjust the exponent
//accordingly and check for any exceptional condition (normalization-2).
repeat(23) begin
  if(add_mantissa_sum[23]==0)
  begin
    add_mantissa_sum =(add_mantissa_sum <<1'b1);
    exponent_final=exponent_final-1'b1;
  end
  else begin
    break;
  end
end
mantissa_final = add_mantissa_sum[22:0];

end

assign sum_x70 = {res_sign, exponent_final, mantissa_final};

endmodule

```