Final Project Report
ECE-528

Jamil Hasibul
DWAG Tag: 855574994
Date: December 4, 2020

1 Hardware Implementation of a Neural Network node

1.1 Milestone 1: Multiplier and Accumulator (MAC) Module Design

A MAC module has been designed in this step that computes following function : $y = \sum_{i=1}^{16} w_i \cdot x_i$

The schematic of this module is shown in figure 1:

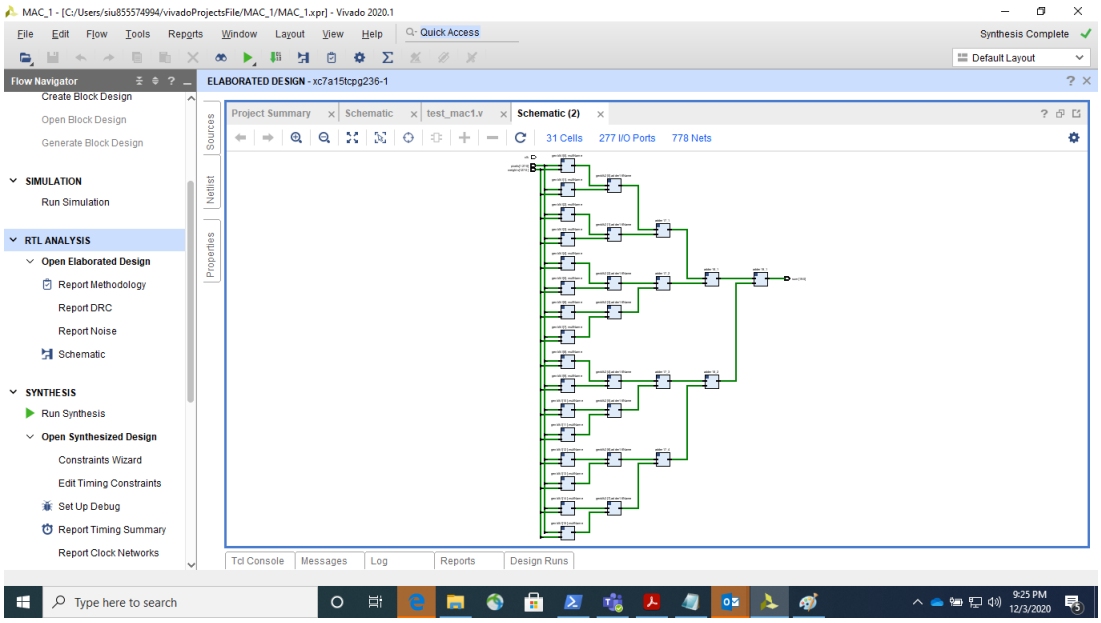

Figure 1: The schematic for MAC module

This MAC consist of 16 multiplier 8 bit multiplier which outputs is 16 bit. The module also consist of 8 16 bit adder, 4 17 bit adder, 2 18 bit adder and 1 19 bit adder. The Module's output is 20 bit and it takes 16*8= 128 bit weight and 128 bit pixel value. The 128 bit inputs are distributed into 16, 8 bit chunks and multiplied with 16 multiplier.

The behavioral and after synthesize, obtained waveform are given in figure 2 and 3.

There are some timing glitch observed in the output when I did post synthesis timing simulation. This is due to datapath variation for different elements of the module.

One important observation I found is if I implement pipeline (include pipeline registers in the data-path to make circuit faster in terms of supported clock frequency), the output of the module is wrong for the first instance during the synthesis phase of the module. This is likely related to initialization timing used in vivado environment. If I get rid of the pipeline registers, the output is as expected. I was implementing 3 stage of pipelines.

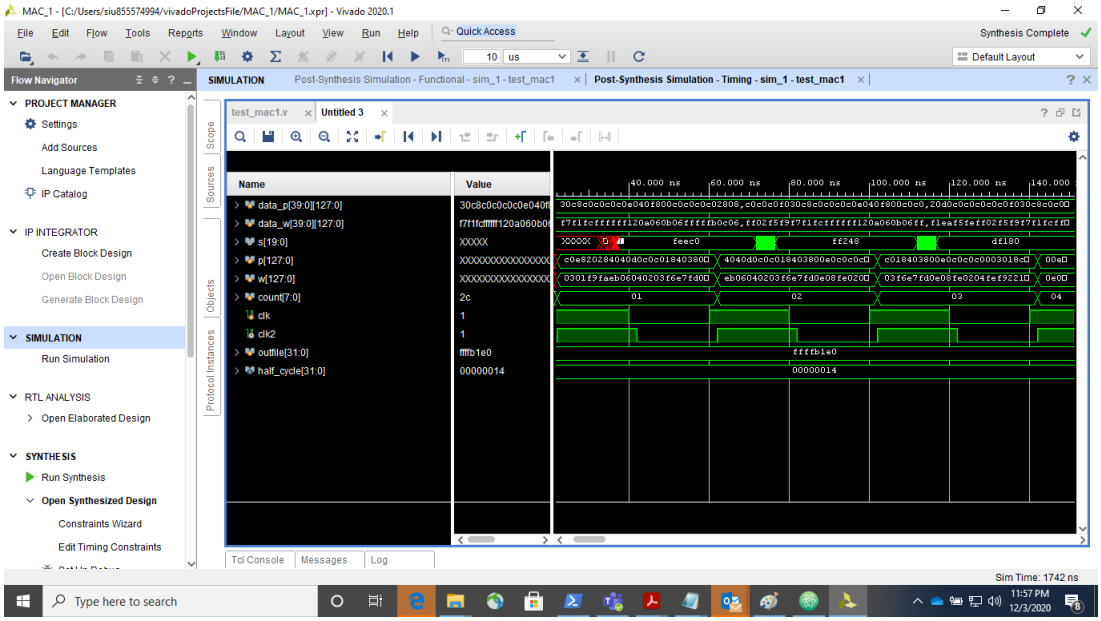

Figure 2: Behavioral waveform obtained in MAC module

Top module for this part in mac1.v and testbench is mac2_tb.v

```
1
2  module multiplier(
3      input signed [7:0] a,
4      input signed [7:0] b,
5      output signed [15:0] p
6  );
7      assign p=a*b;
8  endmodule
```

```
1
2  module adder #(
3      parameter WIDTH=16,
4      parameter A_WIDTH=WIDTH,
5      parameter B_WIDTH=WIDTH)
6      (
7      input signed[A_WIDTH-1:0] a,
8      input signed[B_WIDTH-1:0] b,
9      output signed[WIDTH:0] s
10     );
11
12     assign s=a+b;
13
14 endmodule
```

```
1
2  module mac1(
3      input [127:0] pixels,
4      input [127:0] weights,
5      output [19:0] sum);
6
7      // reg [127:0] pixels;
```

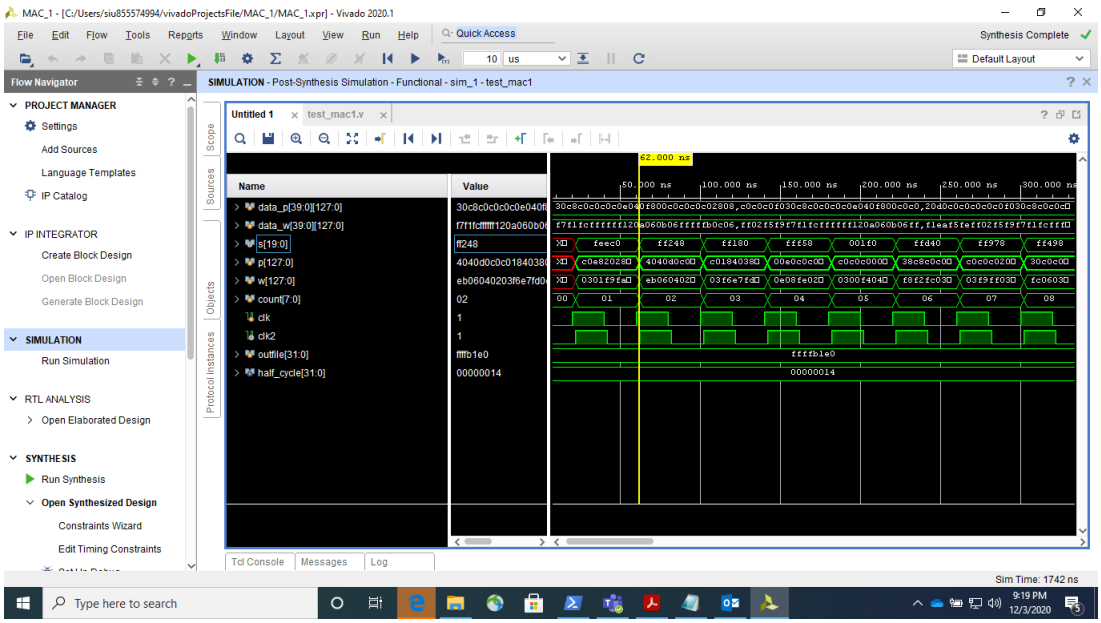

Figure 3: waveform obtained after synthesis for MAC module

```
    // reg [127:0] weights;
    // reg [19:0] sumOUT;
    // wire [19:0] sum;

    wire[255:0] p;
    wire[135:0] s1;
    wire[71:0] s2;
    wire[37:0] s3;

    //Generate the multipliers
    genvar i;
    generate
        for(i=0;i<=15;i=i+1)
            multiplier multName(pixels[(127-8*i):(127-8*i-7)],
                                weights[(127-8*i):(127-8*i-7)],
                                p[(255-16*i):(255-16*i-15)]);
    endgenerate

    //Generate adders
    genvar k;
    generate
    for(k=0;k<=7;k=k+1)
        adder #(16) adderName(p[(255-16*2*k):(255-16*2*k-15)],
                              p[(255-16*(2*k+1)): (255-16*(2*k+1)-15)],
                              s1[(135-17*k):(135-17*k-16)]);
    endgenerate

    adder #(17) adder_17_1(s1[135:119], s1[118:102], s2[71:54]);
    adder #(17) adder_17_2(s1[101:85],s1[84:68],s2[53:36]);
    adder #(17) adder_17_3(s1[67:51],s1[50:34],s2[35:18]);
    adder #(17) adder_17_4(s1[33:17],s1[16:0],s2[17:0]);

```

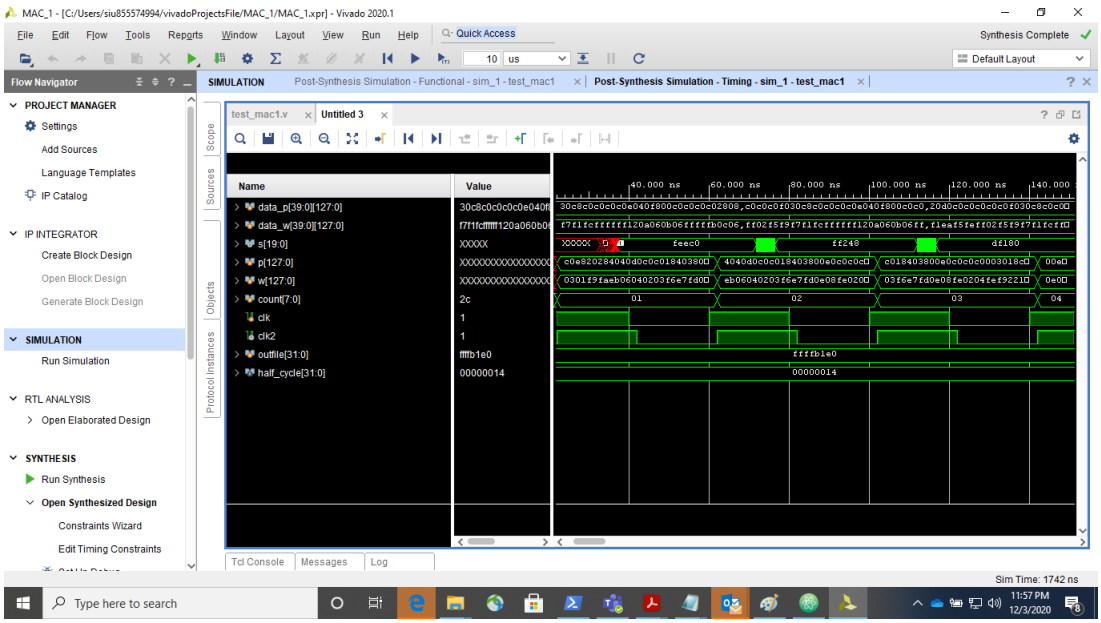

Figure 4: Timing waveform obtained after synthesis for MAC module

```
    adder #(18) adder_18_1(s2[71:54],s2[53:36],s3[37:19]);
    adder #(18) adder_18_2(s2[35:18], s2[17:0], s3[18:0]);

    adder #(19) adder_19_1(s3[37:19],s3[18:0],sum);


    // //Pipeline
    // always @(posedge clk) begin
    //     pixels <= pixelsIN;
    //     weights <= weightsIN;
    //     sumOut <= sum;
    // end
endmodule
```

```

// `timescale 1ns/1ps

module mac2_tb();

    parameter half_cycle = 20;
    reg[127:0] data_p[39:0];
    reg[127:0] data_w[39:0];
    wire [19:0] s;

    reg[127:0] p,w;
    reg[7:0] count;
    reg clk;
    wire clk2;

    integer outfile;

    assign #2 clk2=clk;

```

```
    mac2 uut(clk,p,w,s);

    initial begin
        $readmemh("digits_hex.txt", data_p);
        $readmemh("weights_hex.txt", data_w);
        outfile=$fopen("simout.txt","w");
        clk=0;
        count=0;
    end

    always #half_cycle clk=!clk;

    //Wriete to a file
    always @(posedge clk)
        if(count>0)
            $fdisplay(outfile,"%h",s);

    always @(posedge clk2)
    begin
        p=data_p[count];
        w=data_w[count];
        count=count+1;
        if(count==41) begin
            $fclose(outfile);
            $finish;
        end
    end

endmodule
```

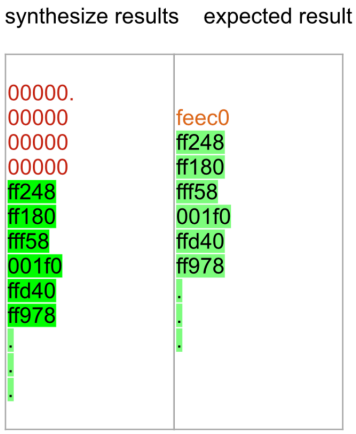

Figure 5: With pipeline registers, the MAC output glitch

As shown above, I am missing feec0 in synthesized result instead I am getting a 00000 on 4th clock cycle . The next results are consistent and correct. This is not happening to my behavioral simulation though. This is avoided by using no pipeline registers.

1.2 Milestone 2: Accumulator (ACC) Module Design

Computing the output of a hidden node in the neural network requires adding 64 multiplication results. However, the MAC module I have designed earlier only compute 16 multiplication and add them together. Thus, I plan to use the same MAC module four times and accumulate the results as shown in Figure 6. Since MAC output is 20 bits and four MAC results are to be accumulated, the ACC will take four clock cycles to produce a result and the result has 22 bits.

The reason for accumulation is to decrease hardware cost by re-utilizing same module in expense of a slower circuit.

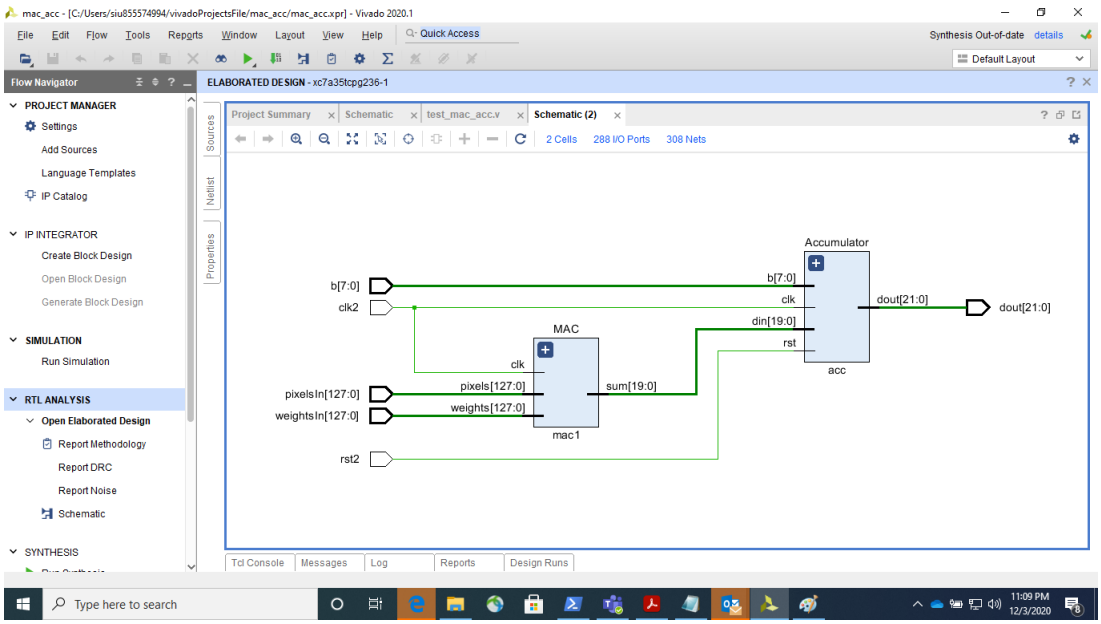

Figure 6: MAC and ACC module together

the bias was set to 11 and the given weight and pixel values are read through the testbench file. A accumulator control cirsuit was written to implement a state machine. The state machine controls the accumulation and cycles around a 4 clock cycle period. As the accumulator takes 4 MAC instance and accumulate them, for 1st clock cycle the ACC will add MAC output and bias b. For the remaining three clock cycle the adder accumulates the 3 instance of MAC module output and latch the results into the output register.

The schematic of the accumulator is shown in figure 7.

Obtained behavioral and post synthesis simulation results are shown in below two figures :

Some interesting finding was the vivado synthesize requires some initialization time so the reset signal should be close to 120ns.

the top module and components code and testbench is given below:

```
1
2 `timescale 1ns / 1ps
3 module mac_acc(
4   input clk2,
```

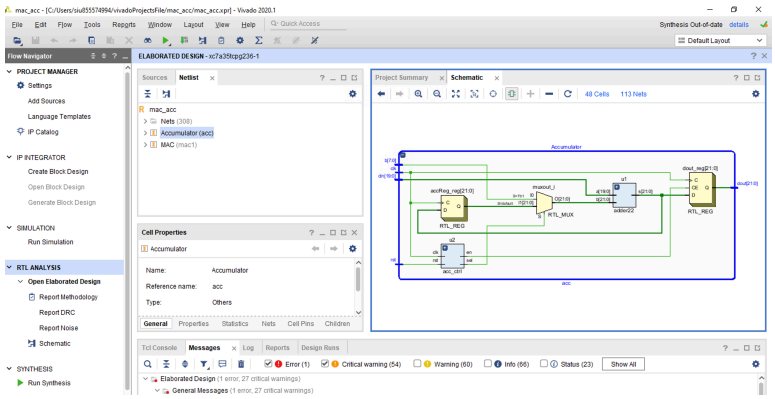

Figure 7: Accumulator Schematic

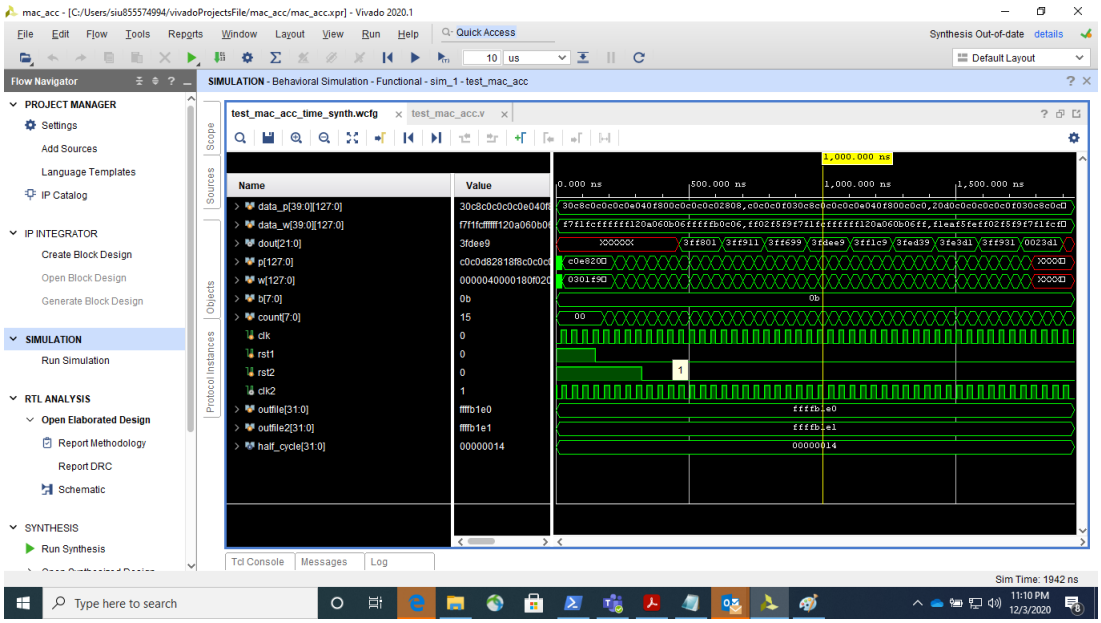

Figure 8: Behavioral wave-forms obtained from MAC+ACC module

```
  input rst2,
  input [127:0] pixelsIn,
  input [127:0] weightsIn,
  input [7:0] b,
  output [21:0] dout);

  wire [19:0] macOut;
  wire [21:0] dout;

  mac1 MAC(clk2,pixelsIn,weightsIn,macOut);
  acc Accumulator(macOut,b,clk2,rst2,dout);

  endmodule
```

```
`timescale 1ns / 1ps
module acc(
    input [19:0] din,
```

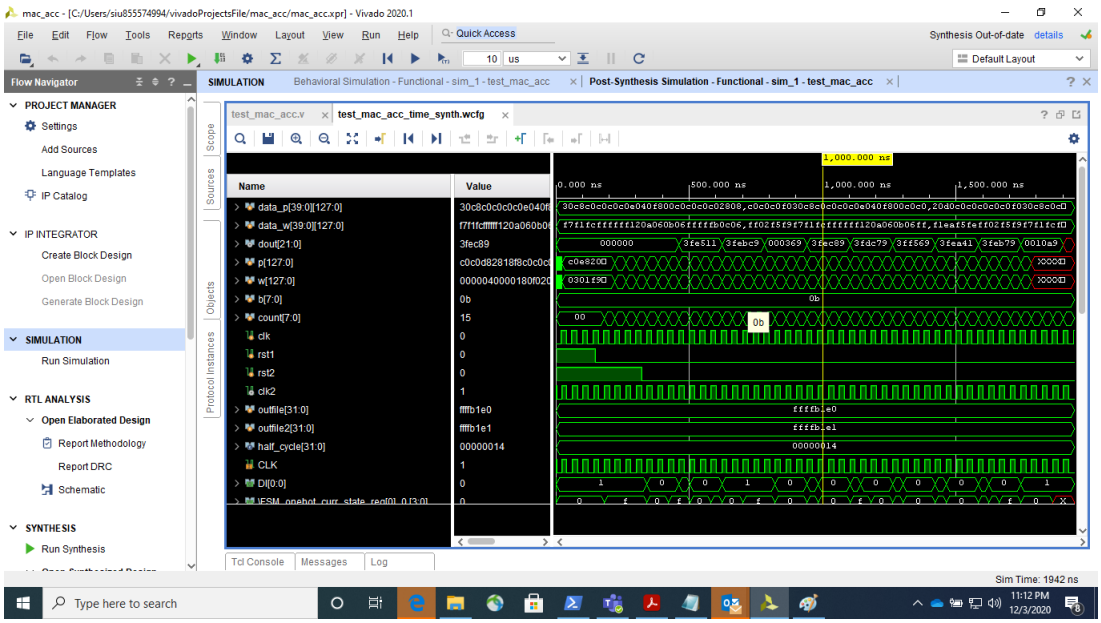

Figure 9: Post Synthesis wave-forms obtained from MAC+ACC module

```
    input [7:0] b,
    input clk,
    input rst,
    output [21:0] dout);

    reg [21:0] accReg, muxout;
    reg[21:0] dout;

    wire sel,en;
    wire [21:0] b_ext,sum;

    adder22 u1(din, muxout,sum);
    acc_ctrl u2(clk,rst,sel,en);

    assign b_est = {{14{b[7]}},b}; //This is sign extension

    always @(posedge clk) begin
        if(en) begin
            dout <=sum;
        end

    end

    always @(posedge clk)
        accReg <=sum;

    always @(*) begin
        if(sel)
            muxout = b_est;
        else
            muxout = accReg;

```

```
    end
endmodule
```

```

`timescale 1ns / 1ps
module adder22(
    input signed[19:0] a,
    input signed[21:0] b,
    output signed[21:0] s
    );

    assign s=a+b;

endmodule
```

```

`timescale 1ns / 1ps
module acc_ctrl(
    input clk,
    input rst,
    output sel,
    output en
);

    reg[1:0] next_state, curr_state;
    parameter s1 = 2'b00;
    parameter s2 = 2'b01;
    parameter s3 = 2'b10;
    parameter s4 = 2'b11;

    always @(posedge clk) begin
        if(rst)
            curr_state = s1;
        else
            curr_state = next_state;
    end


    always @(curr_state) begin
        case(curr_state)
            s1: next_state = s2;
            s2: next_state = s3;
            s3: next_state = s4;
            s4: next_state = s1;
        endcase
    end

    assign sel = (curr_state ==s1) ? 1'b1 : 1'b0;
    assign en =  (curr_state ==s4) ? 1'b1 : 1'b0;

endmodule
```

```

`timescale 1ns / 1ps
module test_mac_acc();
        parameter half_cycle = 20;
```

```
5         reg [127:0] data_p[39:0];
6         reg [127:0] data_w[39:0];
7         wire [21:0] dout;
8         reg [127:0] p, w;
9         reg [7:0] b;
10        reg [7:0] count;
11        reg clk, rst1, rst2;
12        wire clk2;
13        integer outfile;
14        integer outfile2;
15
16        assign #2 clk2=clk; // delayed clk
17
18        mac_acc uut (clk2, rst2, p, w, b, dout);
19
20
21        initial begin
22            $readmemh("C:\\Users\\siu855574994\\OneDrive - Southern
    Illinois University\\Fall 2020\\ECE-528\\projects\\Codes\\digits_hex.txt
    ",data_p);
23            $readmemh("C:\\Users\\siu855574994\\OneDrive - Southern
    Illinois University\\Fall 2020\\ECE-528\\projects\\Codes\\weights_hex.
    txt",data_w);
24            outfile = $fopen("C:\\Users\\siu855574994\\OneDrive - Southern
    Illinois University\\Fall 2020\\ECE-528\\projects\\Codes\\simout_mac_acc
    .txt","w");
25            outfile2 = $fopen("C:\\Users\\siu855574994\\OneDrive - Southern
     Illinois University\\Fall 2020\\ECE-528\\projects\\Codes\\simout_mac.
    txt","w");
26            clk=0;
27            count=0;
28            rst1=1;
29            rst2=1;
30            b=11;
31            #150 rst1=0;
32        end
33
34        //Generate clock
35        always #half_cycle clk=!clk;
36        // write acc output to file
37            always @(posedge clk)
38                if ((count>7)&(count[1:0]==2'b00))
39                    $fdisplay(outfile, "%h", dout);
40
41            // write mac output to file
42
43         always @(posedge clk) begin
44            if(count>3)begin
45            $fdisplay(outfile2, "%h", uut.dout);
46            //rst2=1;
47            end
48            end
49         always @(posedge clk2) begin
50              p=data_p[count];
51              w=data_w[count];
```

```
52                if (!rst1) begin
53                  count= count+1;
54                  if (count==4)
55                     #half_cycle rst2=0;
56
57                  if (count==45) begin
58                    $fclose(outfile);
59                    $fclose(outfile2);
60                    $finish;
61                    end
62                  end
63                end
64          endmodule
```

1.3 Milestone 3: Integrating with sigmoid IP block

So far with MAC+ACC module I have computed following function.

$y = \sum_{i=1}^{N} w_i \cdot x_i + b_i$

But for a neural network node, a sigmoid non-linearity is needed to be incorporated to fullfill folowing expression.

$f(y) = \frac{1}{1+e^{-y}} - 0.5$

as the sigmoid module was given, I wrote a IP-wrapper module which takes the 22 bit output from the MAC+ACC module and pass it through the Sigmoid IP block to generate 8 bit final output.

The schematic for this module is shown in figure 10.

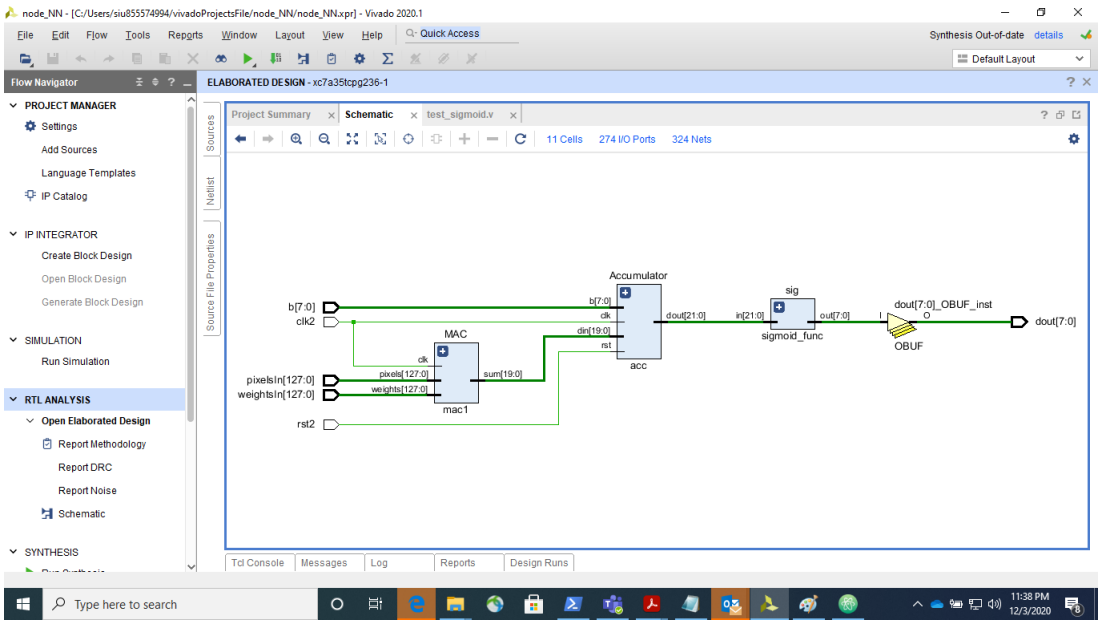

Figure 10: Schematic of MAC+ACC+Sigmoid block or a single neuron

The schematic of wrapper and sigmoid function is given in below figure :

The output obtained from this combination module : MAC+ACC+Sigmoid is shown in below:

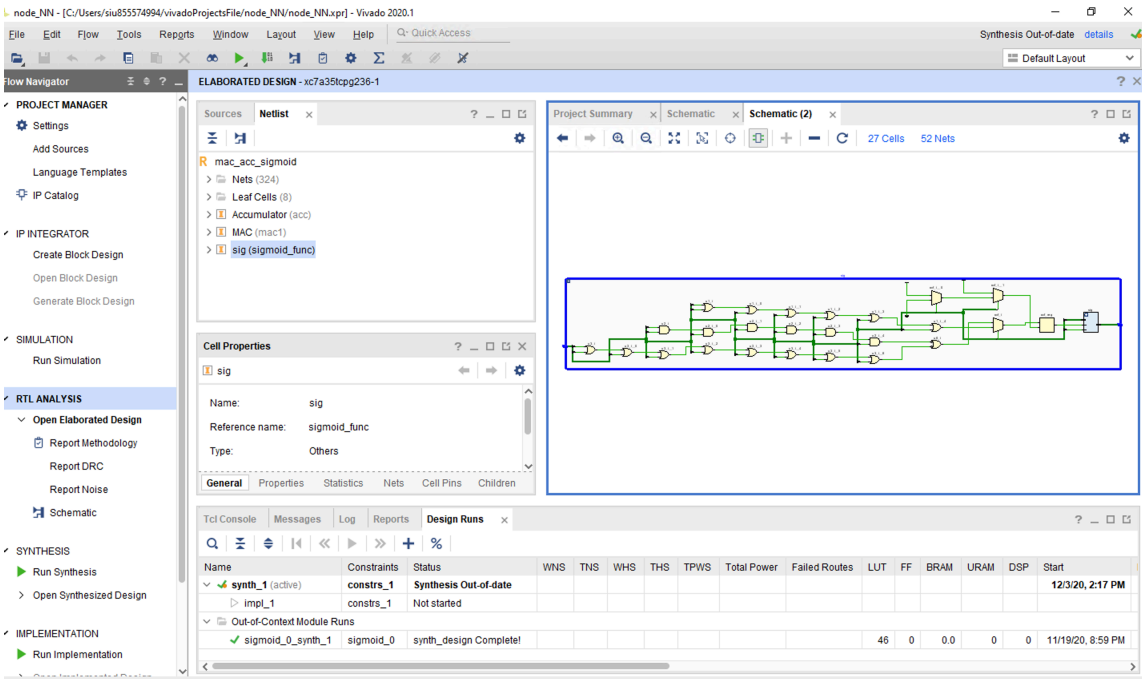

Figure 11: Schematic of IP wrapper and Sigmoid block

Top module, the IP-wrapper and sigmoid (sigmoid_func) and testbench codes are given below:

```
1
2  `timescale 1ns / 1ps
3  module mac_acc_sigmoid(
4    input clk2,
5    input rst2,
6    input [127:0] pixelsIn,
7    input [127:0] weightsIn,
8    input [7:0] b,
9    output [7:0] dout);
10
11   wire [19:0] macOut;
12   wire [21:0] dout_acc;
13   wire [7:0] dout;
14
15   mac1 MAC(clk2,pixelsIn,weightsIn,macOut);
16   acc Accumulator(macOut,b,clk2,rst2,dout_acc);
17   sigmoid_func sig(dout_acc,dout);
18
19   endmodule
```

```
1
2  module sigmoid_func(
3      input[21:0] in,
4      output[7:0] out
5  );
6
7  wire[16:0] y;
8  reg ovf;
```

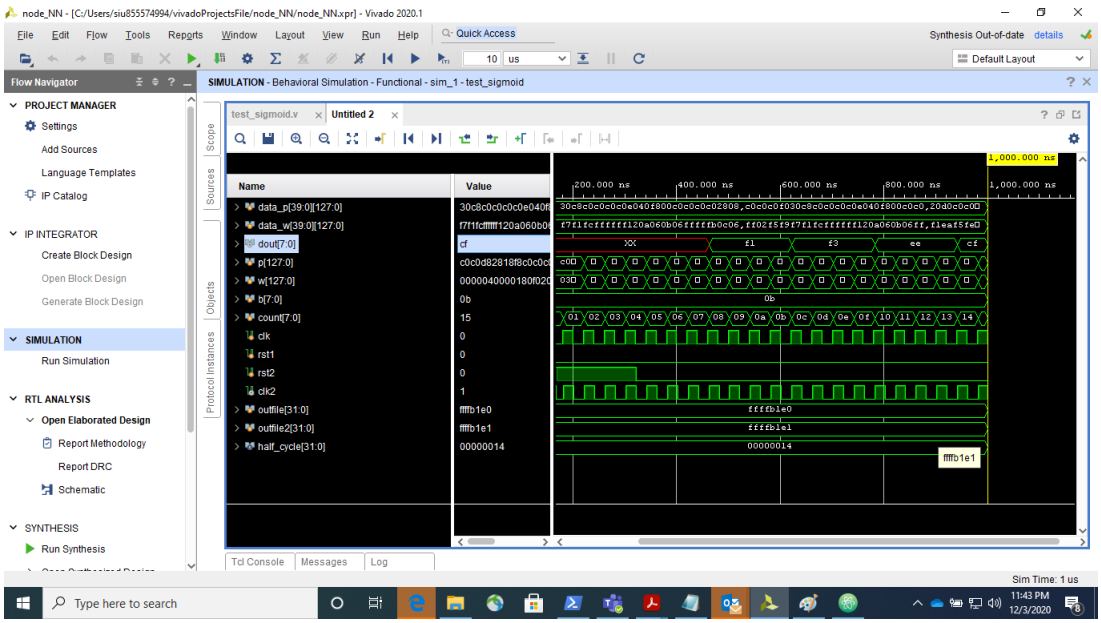

Figure 12: Final waveform obtained from a single node

```
wire sign;
wire out1,out2,out3,out4,out5,out6;
wire[8:0] data;

assign y=in[21:5];
assign sign=y[16];
assign data=y[8:0];
or u1(out1,y[15],y[14],y[13],y[12],y[11],y[10],y[9]);
and u2(out2,y[15],y[14],y[13],y[12],y[11],y[10],y[9]);
or u3(out3,y[8],y[7],y[6],y[5],y[4],y[3],y[2],y[1],y[0]);
not u4(out4,out2);
not u5(out5,out3);
or u6(out6,out4,out5);

always @(*) begin
  if (sign ==1'b0) begin
    ovf<=out1;
  end
  if (sign ==1'b1) begin
    ovf<=out6;
  end
end

sigmoid_0 sig(data,sign,ovf,out);

endmodule
```

```

`timescale 1ns / 1ps
module test_sigmoid( );

parameter half_cycle = 20;
reg [127:0] data_p[39:0];
```

```
reg [127:0] data_w[39:0];
wire [7:0] dout;
reg [127:0] p, w;
reg [7:0] b;
reg [7:0] count;
reg clk, rst1, rst2;
wire clk2;
integer outfile;
integer outfile2;

assign #2 clk2=clk; // delayed clk

mac_acc_sigmoid uut (clk2, rst2, p, w, b, dout);


  initial begin
  $readmemh("C:\\Users\\siu855574994\\OneDrive - Southern Illinois
    University\\Fall 2020\\ECE-528\\projects\\Codes\\digits_hex.txt",data_p)
    ;
  $readmemh("C:\\Users\\siu855574994\\OneDrive - Southern Illinois
    University\\Fall 2020\\ECE-528\\projects\\Codes\\weights_hex.txt",data_w
    );
  outfile = $fopen("C:\\Users\\siu855574994\\OneDrive - Southern Illinois
    University\\Fall 2020\\ECE-528\\projects\\Codes\\simout_sigmoid_node_syn
    .txt","w");
  outfile2 = $fopen("C:\\Users\\siu855574994\\OneDrive - Southern Illinois
    University\\Fall 2020\\ECE-528\\projects\\Codes\\simout_mac_sigmoid.txt
    ","w");
  clk=0;
  count=0;
  rst1=1;
  rst2=1;
  b=11;
  #150 rst1=0;
end

//Generate clock
always #half_cycle clk=!clk;
// write acc output to file
  always @(posedge clk)
      if ((count>7)&(count[1:0]==2'b00))
          $fdisplay(outfile, "%h", dout);

  // write mac output to file

always @(posedge clk) begin
  if(count>3)begin
  $fdisplay(outfile2, "%h", uut.dout);
  //rst2=1;
  end
  end
always @(posedge clk2) begin
    p=data_p[count];
    w=data_w[count];
    if (!rst1) begin
```

```
54        count= count+1;
55        if (count==4)
56           #half_cycle rst2=0;
57
58        if (count==45) begin
59          $fclose(outfile);
60          $fclose(outfile2);
61          $finish;
62          end
63        end
64      end
65 endmodule
```

1.4 Design with considerations

Beyond engineering one critical consideration was implemented in this design and that is economic consideration. In the accumulation unit, I utilize same one MAC module to do 4 different multiplication. The price paid was instead of multiplying each weight and pixel value pairs con-currently, 4 clock cycle is required to compute same computation sequentially. So the reason for accumulation unit is to decrease hardware cost by re-utilizing same module in expense of a slower circuit.