Convolutional neural network

FINAL_PROJECT

Praveen khemalapure

011432876

Design Architecture

In this CNN network we have two stage networks. In the input layer we have the raw input data which in this project we have taken it to be 20*20 matrix. Next the trained weights for the first layer are around 400*25 because we have 25 nodes in the hidden layer. In the final layer we have 10nodes at the output. So, the second set of trained weights are 25*10. The block diagram of the CNN looks as shown below.

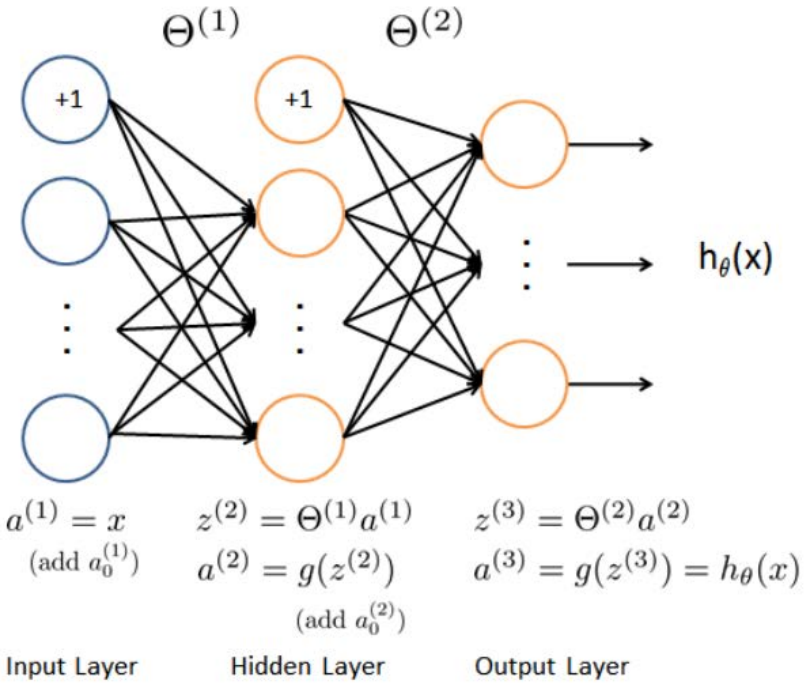
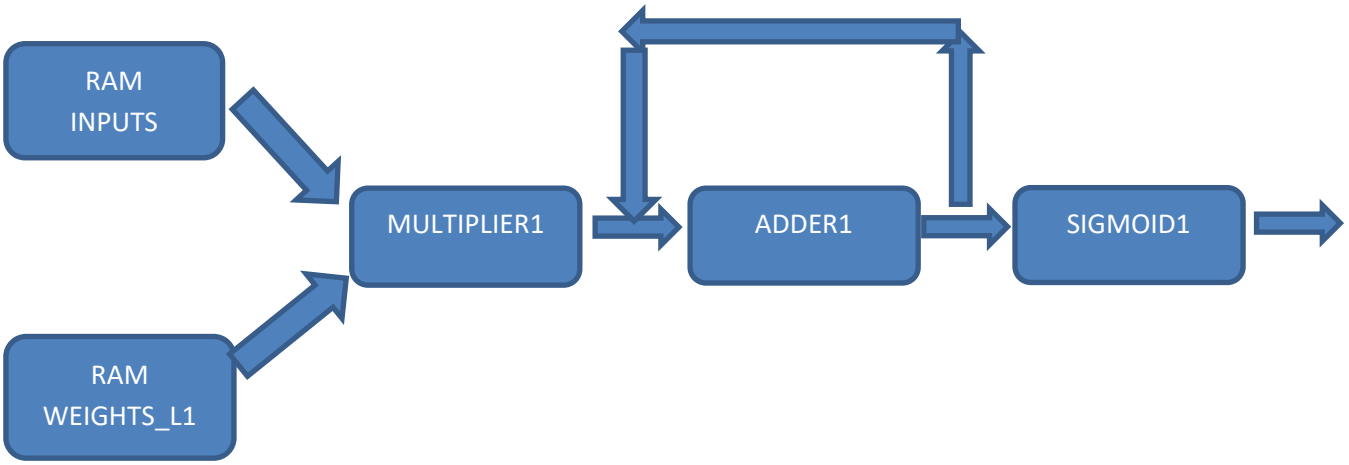
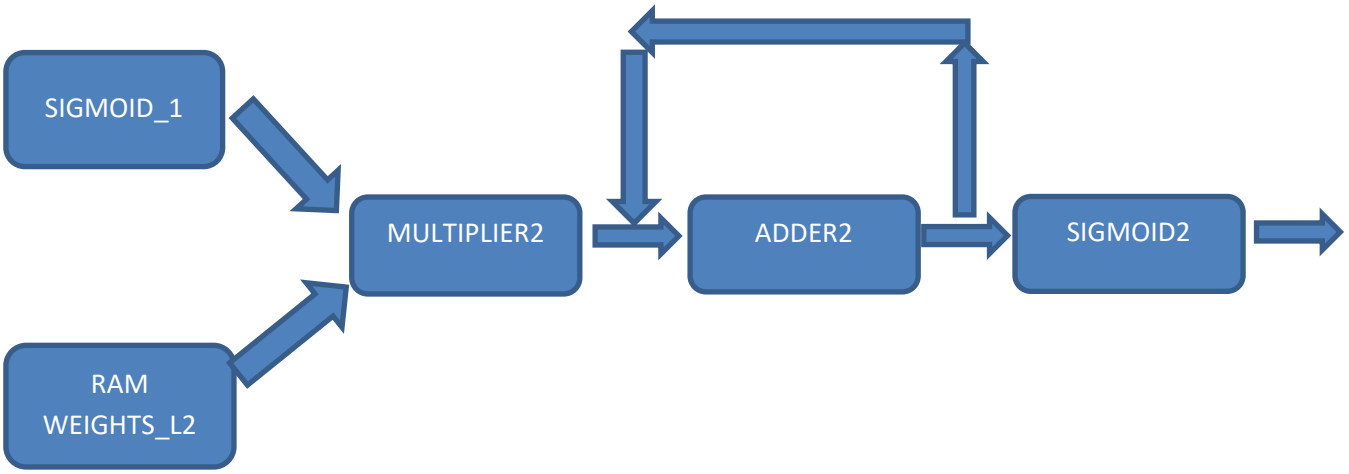


Figure 2: Neural network model.

DESIGN BLOCK DIAGRAM:



LAYER 1



LAYER 2

DESIGN PROCEDURE

1. All the raw inputs (X) are stored in RAM INPUTS. And all the L1 weights are stored in RAM WEIGHTS_L1.
2. For the first convolution layer, all the inputs and weights are multiplied and added.
3. When all the 25node outputs are available, they are passed through the sigmoid function which is loaded in SIGMOID1 RAM.
4. The output of sigmoid1 becomes input to the second convolution layer. Weights for L2 are stored in RAM WEIGHTS_L2.
5. Again, all the inputs and weights are multiplied and added.
6. When all the 10 output node results are available, they are passed through the sigmoid2 function which is stored in SIGMOID2.
7. Finally, the ten outputs are checked for the maximum weight and that will give us the character recognition result.

HARDWARE SPECIFICATION

We shall discuss few hardware specifications of the cyclone2 board, which are very important while designing the convolutional neural network.

Table 1–1. Cyclone II FPGA Family Features (Part 1 of 2)

Feature	EP2C5 (2)	EP2C8 (2)	EP2C15 (1)	EP2C20 (2)	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000
Embedded multipliers (3)	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4

1. **RAM MEMORY.**

   In cyclone2 board we have total of 239616bits of RAM memory available. Since we had to Store all the raw inputs(X), weights in the RAM memory. We had carefully select the Precision of both weights(W) as well as inputs(X).

WHY 7bit precision was choose for this project?

Initially we had chosen 8 bits as precision for both inputs(X) and weights(W). But when we calculate the total amount of RAM bits using 8bit precision the total bits exceeded the total available memory units. Because of this reason we fixed on 7bit precision.

The comparison data for 7 and 8 bits is as shown below in the table. Since it was crossing more that 180Kb we decided to go with 7bit precision.

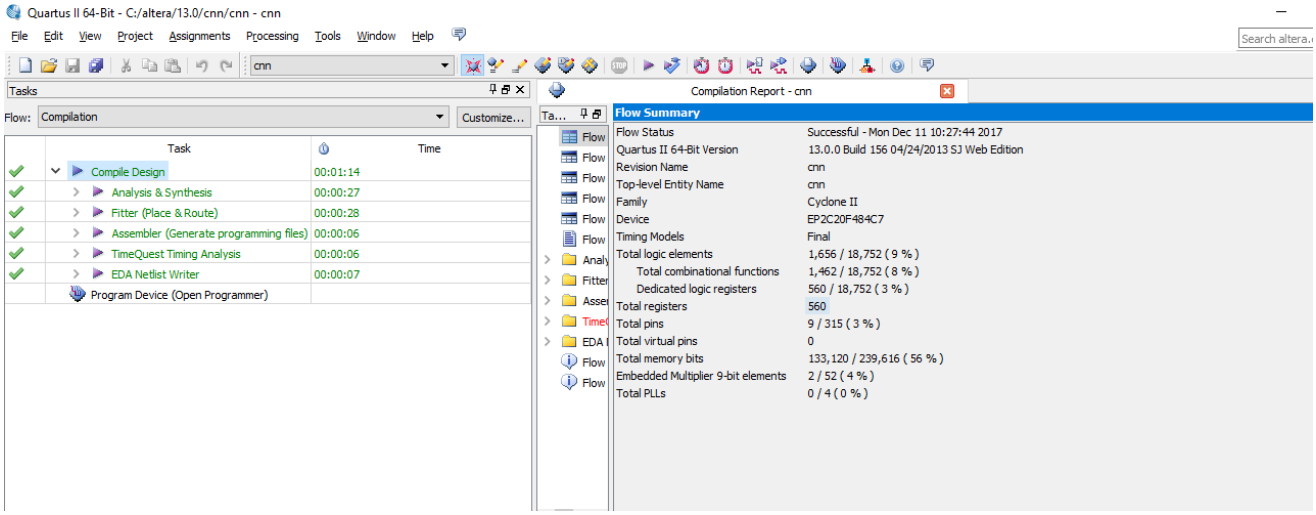
	7bit precision	8bit precision
INPUTS	15000bits	30000bits
WEIGHTS L1	55000	80000
WEIGHTS L2	55000	80000
APPROX TOTAL BITS	~133184	~180000

COMPARISON TABLE.

OTHER HARDWARE COMPONENTS USED ON CYCLORE2 BOARD.

COMPONENT TYPE	USED	AVAILABLE	PERCENTAGE USED
MULTIPLIER	2	52	4%
TOTAL COMBINATIONAL FUNCTIONS	1471	18752	8%
DEDICATED LOGIC REGISTERS	569	18752	3%
TOTAL PINS	10	315	3%
TOTAL REGISTERS	569		

Below figure will support the statistics stated above.



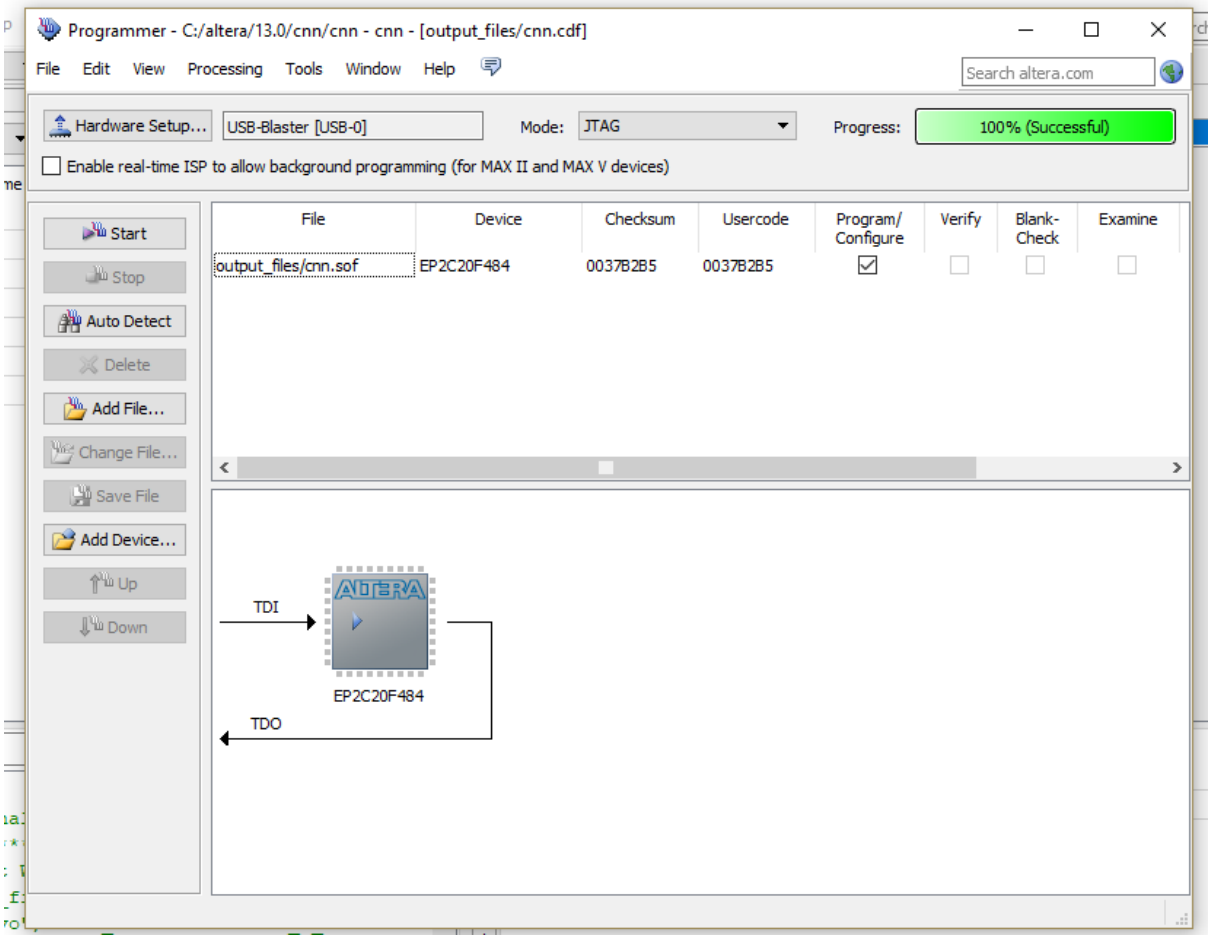
STEPS TO BE FOLLOED FOR IMPLEMENTATION ON CYCLONE II FPGA

STEPS FOLLOWED.

1. First completely run the compilation and make sure that before proceeding the code is completely synthesized.
2. Once the code is completely synthesized, Go, to tools → click on programmer
    a. First make the hardware setup and select the USB-Blaster0 to burn the code onto the FPGA.
    b. Then import the design.sof file into the programmer.
    c. Then add the FPGA device which is present on the board. In this project the Device is EP2C20F484.
    d. Then click on START, progress bar will show the status of the run.

3. **Finally, w**hen the status is **100% successful.** Which means that code is burnt on FPGA.

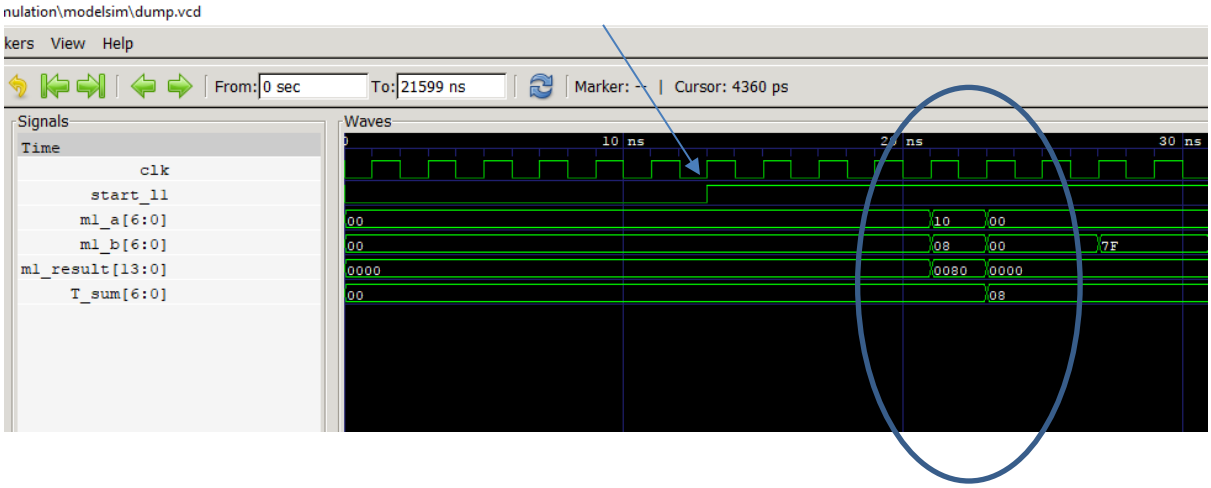
The below figure shows the programmer window, which shows that the code is successful dumped into the FPGA.



SIMULATION RESULTS:

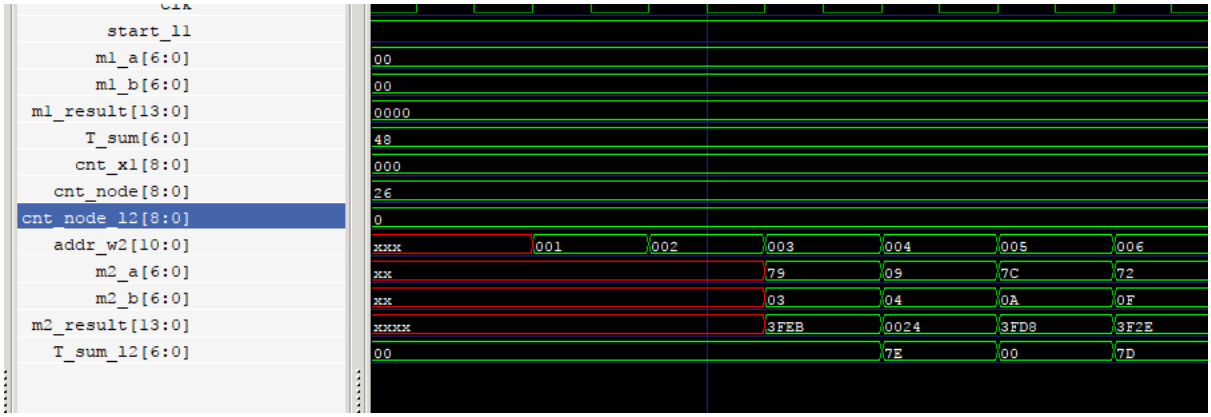
In this project we had taken one set of inputs of X(20*20).

1. stage1 multiplier and adder output on start pulse. Since it is a pipelined design we get the output after 4 clock cycles.

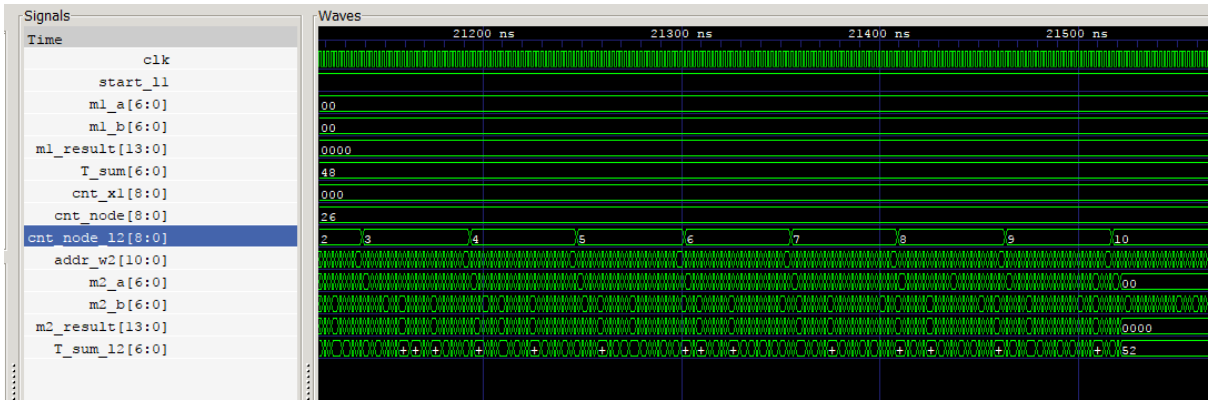


2. In the below figure we can see that we have reached the 26node. All mac operations are complete and ready to proceed to sigmoid function.

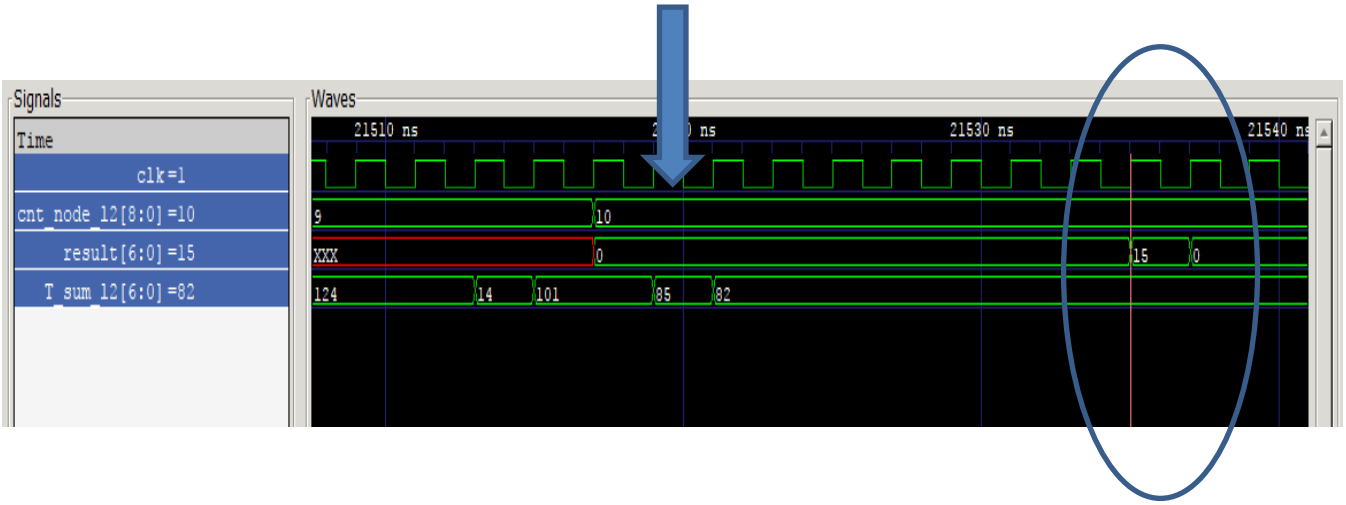
3. Stage2 multiplier and adder output. Since it is a pipelined design we get the output after 4 clock cycles.



4. In the below figure we can see that we have reached the 10node. All mac operations are complete and ready to proceed to sigmoid_l2 function.



5. In this figure we can see that character has been recognized based on the highest weight decoder. Result pin highlights the same. Arrow indicates that node 10 has been reached and result wire is updated after couple of cycles.



SOURCE CODE

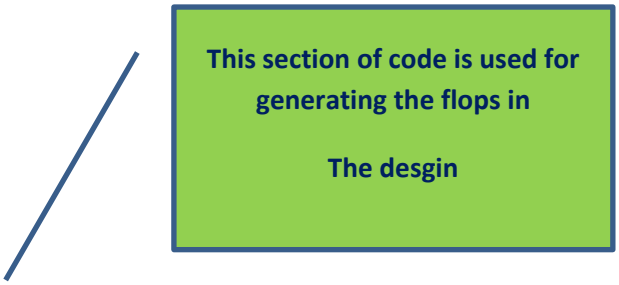
```
module cnn(rst, clk, result);

input wire rst,clk;
output wire [6:0]result;

reg [11:0]address_x1;
reg [11:0]address_xx1;
wire [6:0]x1_val,w1_val;
reg [6:0]T_summ,T_sum;
reg [8:0]cnt_x1,cnt_xx1;
reg [18:0]cnt_w1,cnt_ww1;
reg start_l1,output_ram_l1,output_ram_l11,output_ram_l111,output_ram_l1111;
reg [8:0]cnt_node,cnt_nodee;
reg [13:0]sum;
reg [13:0]addr_w1;
reg [13:0]addr_ww1;
reg [4:0]cnt_l1,cnt_l11;
reg [6:0]l1_results[30:0];
wire [6:0]m1_a,m1_b;
wire [13:0]m1_result;
reg
started_output,started_output1,started_output11,started_output111,started_output1111,started_
output11111;
```

```
reg started_output111111;

reg [6:0]cnt_sig_l1,cnt_sig_l11;

reg start_sigmoid;
reg [6:0]sigmoid_addr,sigmoid_addr_2;
wire[6:0]sigmoid_addr_in,sigmoid_addr_2_in;
reg [6:0]sigmoid_l1_results[26:0];
wire[6:0]sigmoid_data,sigmoid_data_2;

reg
start_l2,output_ram_l2,output_ram_l22,output_ram_l222,output_ram_l2222,output_ram_l22222;

reg [11:0]address_x1_l2;
reg [11:0]address_xx1_l2;
wire[6:0]w2_val;
reg [6:0]T_summ_l2,T_sum_l2;
reg [18:0]cnt_w1_l2,cnt_ww1_l2;
reg [8:0]cnt_node_l2,cnt_nodee_l2;
reg [13:0]sum_l2;
reg [13:0]addr_w;
reg [6:0]l2_results[14:0];
reg [10:0]addr_ww1_l2;
reg [50:0]cnt_x1_l2,cnt_xx1_l2;
reg [6:0]x2_val;
reg [6:0]cnt_l1_l2,cnt_l11_l2;
reg start_sigmoid_2;
wire [13:0]m2_result;
wire [10:0]addr_w2;
wire [6:0]m2_a,m2_b;
reg [10:0]cnt_xx2_l2,cnt_x2_l2;
reg start_collecting_l2,start_collecting_l22;
reg [6:0]result_out;
reg [6:0]final_data[9:0];
reg [4:0]cnt_final,cnt_finall;
```

// MODULE INISTANCIATION SECTION

```
ram_x1 x1(.address(address_x1),.clock(clk),.data(1'b0),.wren(1'b0),.q(x1_val));
w1 weight1(.address(addr_w1),.clock(clk),.data(1'b0),.wren(1'b0),.q(w1_val));
w2 weight2(.address(addr_w2),.clock(clk),.data(1'b0),.wren(1'b0),.q(w2_val));
multiplier1 m1(.dataa(m1_a),.datab(m1_b),.result(m1_result));
multiplier1 m2(.dataa(m2_a),.datab(m2_b),.result(m2_result));
sigmoid_value sig1(sigmoid_addr_in,sigmoid_data);
sigmoid_value2 sig2(sigmoid_addr_2_in,sigmoid_data_2);

assign m1_a      = x1_val;
assign m1_b      = w1_val;
assign result    = result_out;
```

```
assign sigmoid_addr_in = sigmoid_addr;
assign sigmoid_addr_2_in = sigmoid_addr_2;
assign addr_w2    = addr_ww1_l2;
assign m2_a       = w2_val;
assign m2_b       = x2_val;


always @(posedge clk or posedge rst)
if(rst)
begin
address_x1        <= 0;
start_l1          <= 0;
output_ram_l1  <= 0;
T_sum         <= 0;
cnt_l1        <= 0;
addr_w1       <= 0;
cnt_node      <= 0;
cnt_x1        <= 0;
cnt_w1        <= 0;
T_sum         <= 0;
sum           <= 0;
cnt_sig_l1    <= 0;
cnt_w1_l2     <= 0;
cnt_l1_l2     <= 0;
cnt_node_l2   <= 0;
cnt_x1_l2     <= 0;
T_sum_l2      <= 0;
cnt_x2_l2     <= 0;
cnt_final     <= 0;

end
else
begin
start_l1        <= 1;
output_ram_l1    <= start_l1;
output_ram_l11   <= output_ram_l1;
output_ram_l111  <= output_ram_l11;
output_ram_l1111 <= output_ram_l111;
started_output1   <= started_output;
started_output11   <= started_output1;
started_output111   <= started_output11;
started_output1111 <= started_output111;
started_output11111 <= started_output1111;
started_output111111 <= started_output11111;

output_ram_l2    <= start_l2;
output_ram_l22   <= output_ram_l2;
output_ram_l222  <= output_ram_l22;
output_ram_l2222 <= output_ram_l222;
```



```
T_sum          <= T_summ;
T_sum_l2       <= T_summ_l2;
cnt_l1                <= cnt_l11;
cnt_w1         <= cnt_ww1;
cnt_x1         <= cnt_xx1;
address_x1     <= address_xx1;
addr_w1        <= addr_ww1;
cnt_node       <= cnt_nodee;
cnt_sig_l1     <= cnt_sig_l11;
cnt_x1_l2      <= cnt_xx1_l2;
cnt_w1_l2      <= cnt_ww1_l2;
cnt_l1_l2      <= cnt_l11_l2;
cnt_node_l2    <= cnt_nodee_l2;
cnt_x2_l2      <= cnt_xx2_l2;
start_collecting_l2 <= start_collecting_l22;
cnt_final      <= cnt_finall;

//output_ram_l_l2 <= start_l2;
end
```

THIS IS THE STARTING OF THE COMBINATIONAL BLOCK.

```
always @(*)
begin
T_summ     = T_sum;
T_summ_l2  = T_sum_l2;
cnt_ww1    = cnt_w1;
cnt_xx1    = cnt_x1;
cnt_l11    = cnt_l1;
address_xx1 = address_x1;
addr_ww1   = addr_w1;
cnt_nodee  = cnt_node;
sum        = sum;
cnt_sig_l11 = cnt_sig_l1;
cnt_xx1_l2  = cnt_x1_l2;
cnt_ww1_l2  = cnt_w1_l2;
cnt_l11_l2  = cnt_l1_l2;
cnt_nodee_l2 = cnt_node_l2;
cnt_xx2_l2  = cnt_x2_l2;
start_collecting_l22 = start_collecting_l2;
cnt_finall  = cnt_final;
```

START OF LAYER ONE MAC OPERATIONS

```
        if(start_l1 == 1)
                begin
                        if(cnt_node < 26)
                        begin
                                if(cnt_x1 < 401)
                                begin
                                cnt_xx1     = cnt_x1 + 1;
                                cnt_ww1     = cnt_w1 + 1;
                                address_xx1 = cnt_x1;
                                addr_ww1    = cnt_w1;
                                started_output     = 0;
                                end
                                else
                                begin
                                cnt_xx1          = 0;
//                              l1_results[cnt_l1]  = T_sum;
                                started_output     = 1;
//                              cnt_ll1          = cnt_l1 + 1;
                                cnt_nodee   = cnt_node + 1;
                                end
                        end
                        else
                        begin
                        cnt_ww1      = 0;
//                      cnt_l1       = 0;
                        end
                end


                if((started_output111111 == 1) && (cnt_l1 < 26))
                        begin
                                l1_results[cnt_l1]  = T_sum;
                                $display("ans nad index %d %d",cnt_l1,T_sum);
                                cnt_l11          = cnt_l1 + 1;
                                if(cnt_l1 == 25)
                                begin
                                start_sigmoid = 1;
                                end
                        end
                                        //start_sigmoid = 0;
                                        //cnt_11 = 0;



        if(output_ram_l1111 == 1)
                 begin
//                sum    = w1_val*x1_val;
                 sum    = m1_result;
                 T_summ = T_sum + sum[10:4];
```

```
                end

        if((start_sigmoid == 1) && (cnt_sig_l1 < 26))
                begin
                sigmoid_addr                                    =  l1_results[cnt_sig_l1];
                sigmoid_l1_results[cnt_sig_l1] =  sigmoid_data;
                $display("add data %b",sigmoid_addr,sigmoid_data);
                cnt_sig_l11                                                             =  cnt_sig_l1 + 1;
                if(cnt_sig_l1 == 25)
                        begin
                        start_l2 = 1;
                        end
                end
```

START OF LAYER 2 MAC OPERATIONS

```
        if(start_l2 == 1)
                begin
                        if(cnt_node_l2 < 11)
                        begin
                                if(cnt_x2_l2 < 26)
                                begin
                                cnt_ww1_l2     = cnt_w1_l2 + 1;
                                cnt_xx2_l2     = cnt_x2_l2 + 1;
                                addr_ww1_l2    = cnt_ww1_l2;
                                start_collecting_l22    = 0;
                                end
                                else
                                begin
                                cnt_nodee_l2                            = cnt_node_l2 + 1;
                                cnt_xx2_l2          = 0;
                                start_collecting_l22    = 1;
                                end
                        end
                        else
                        begin
                        cnt_ww1_l2      = 0;
                        start_sigmoid_2 = 1;
//                      cnt_l1_l2       = 0;
                        end
                end


        if(start_collecting_l2 == 1)
                begin
                                l2_results[cnt_l1_l2]  = T_sum_l2;
                                cnt_l11_l2          = cnt_l1_l2 + 1;
                                $display("Data & cnt %b, %d",l2_results[cnt_l1_l2],cnt_l1_l2);
```

```
            end

        if(cnt_node_l2 == 10)
            begin
            start_sigmoid_2 = 1;
            end

        if(start_sigmoid_2 == 1)
            begin
            sigmoid_addr_2          = l2_results[cnt_final];
            final_data[cnt_final]    = sigmoid_data_2;
            cnt_finall              = cnt_final + 1;
            if(cnt_final == 9)
                    begin
                    result_out = final_data[cnt_final];
                    end
            else
                    begin
                    result_out = 0;
                    end
            end

        if(output_ram_l22 == 1)
            begin
             //sum_l2      = w2_val*l1_results[cnt_x1_l2];
             x2_val      =  sigmoid_l1_results[cnt_x1_l2];

             T_summ_l2    = T_sum_l2 + m2_result[10:4];
             cnt_xx1_l2   = cnt_x1_l2 + 1;
                    if(cnt_xx1_l2 == 25)
                            begin
                            cnt_xx1_l2 = 0;
                            end

            end


end
endmodule
```

SIGMOID FOR L1

```
module sigmoid_value(in_sig,op_sig);
input  [6:0] in_sig;
output [6:0] op_sig;
reg    [6:0] sig_value;
assign op_sig = sig_value;
                    always @(*) begin
                      sig_value = 7'd0;
                            case(in_sig)
                                                7'b1101100: sig_value = 7'b0000011;
                                                7'b1110010: sig_value = 7'b0000100;
                                                7'b0001100: sig_value = 7'b0001010;
                                                7'b0111110: sig_value = 7'b0001111;
                                                7'b1010011: sig_value = 7'b0000000;
                                                7'b0100011: sig_value = 7'b0001110;
                                                7'b1000001: sig_value = 7'b0000000;
                                                7'b0101001: sig_value = 7'b0001110;
                                                7'b1011001: sig_value = 7'b0000001;
                                                7'b1010111: sig_value = 7'b0000001;
                                                7'b1101011: sig_value = 7'b0000011;
                                                7'b1101111: sig_value = 7'b0000100;
                                                7'b1000011: sig_value = 7'b0000000;
                                                7'b1001111: sig_value = 7'b0000000;
                                                7'b0001001: sig_value = 7'b0001010;
                                                7'b0110010: sig_value = 7'b0001111;
                                                7'b0101000: sig_value = 7'b0001110;
                                                7'b1100100: sig_value = 7'b0000010;
                                                7'b0101011: sig_value = 7'b0001110;
                                                7'b0000100: sig_value = 7'b0001000;
                                                7'b1001000: sig_value = 7'b0000000;
                                                7'b0110110: sig_value = 7'b0001111;
                                                7'b0011101: sig_value = 7'b0001101;
                                                7'b0110101: sig_value = 7'b0001111;
                                                7'b1000110: sig_value = 7'b0000000;
                                                7'b0110011: sig_value = 7'b0001111;
                                                7'b1011000: sig_value = 7'b0000001;
                                                7'b0110111: sig_value = 7'b0001111;
                                                7'b1110101: sig_value = 7'b0000101;
                            default: sig_value = 7'd0;
                            endcase
                    end

endmodule
```

SIGMOID FOR l2

```
module sigmoid_value2(in_sig,op_sig);

input  [6:0] in_sig;
output [6:0] op_sig;
reg    [6:0] sig_value;

assign op_sig = sig_value;

                    always @(*) begin
                      sig_value = 7'd0;
                            case(in_sig)
7'b0111110   :  sig_value = 7'b0000000;
7'b1110010   :  sig_value = 7'b0001011;
7'b1101110   :  sig_value = 7'b0001100;
7'b0100001   :  sig_value = 7'b0000001;
7'b1010101   :  sig_value = 7'b0001110;
7'b0101101   :  sig_value = 7'b0000000;
7'b0011010   :  sig_value = 7'b0000010;
7'b1011000   :  sig_value = 7'b0001110;
7'b0101110   :  sig_value = 7'b0000000;
7'b1100101   :  sig_value = 7'b0001111;
                            default: sig_value = 7'd0;
                            endcase
                    end

endmodule
```