Input Layer:

a) Input Layer Source Code:

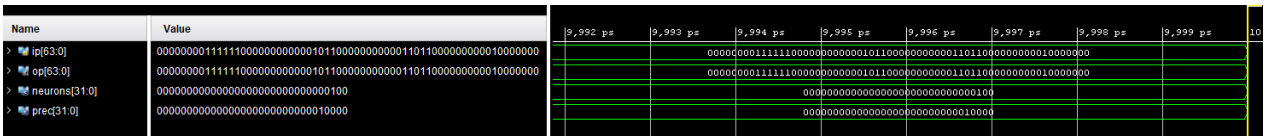
```
module input_layer#(
        neurons = 784,                          //Size of the flattenned input image
        prec = 16                               //Precision of the binary numbers. Set to 16 (Do not Change)
    )(
    input wire[(prec*neurons)-1:0] ip,          // input to the Neural Network
    output wire [(prec*neurons)-1:0] op         // output of the input layer
    );

//////////////Write your code here/////////////////
assign op = ip;
//////////////////Code Ends/////////////////////

endmodule
```

b) Input Layer Results (from TCL Console):

```
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window.
#   }
# }
# run 1000ns
The module input_layer.v works as intended
$finish called at time : 10 ns : File "D:/Documents/EEL4783_VHDL/project/testbench/input_layer_tb.v" Line 35
INFO: [USF-XSim-96] XSim completed. Design snapshot 'input_layer_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

- This shows that the input layer module is working as intended.

c) Input Layer Waveform:



- You can see that the input is the same as the output, which is what is expected.

Neuron:

a) Neuron Source Code:

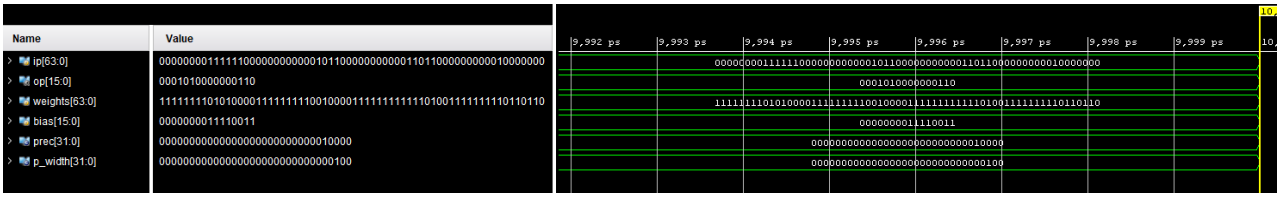
```
module neuron #(
    parameter p_width = 10,                             // Number of neurons of the previous layer
    parameter prec = 16                                 // Precision of the binary numbers. Set to 16 (Do not Change)
    )(
    input   wire signed [(prec*p_width)-1:0] ip,        // flattenned input to the neuron. The neurons recives the output of ev
    output  reg signed [prec-1:0] op,                   // 16 bit output of the neuron
    input   wire signed [(prec*p_width)-1:0] weights,   // flattenned weights. Width = neurons of previous layer * precision
    input   wire signed [prec-1:0] bias                 // 16 bit bias
    );

    wire signed [(2*prec*p_width)-1:0] mult;        //Each 16 bit multiplication will result in a 32 bit number that is trimmed
                                                    //Read the resources linked in section 3.2 of the project instructions for
//////////////////Write your code here//////////////////////////////////////

// Use a generate statement and a for loop to create p_width number of statements
// that each performs 16 bit multiplications using each input and weight

    genvar i;

    generate
        for(i = 0; i < p_width; i = i + 1) begin
            assign mult[(2*(prec*(i + 1))) - 1:2*prec*i] = ip[(prec*(i + 1)) - 1:prec*i] * weights[(prec*(i + 1)) - 1:prec*i];
        end
    endgenerate


// Use an always block and a for loop to perform the addition of the result of the
// multiplication and the bias.
// Hint: use blocking statements (=) so that each statement is executed one after another
// Hint: Vivado automatically unrolls the for loop and duplicates each statement. Use this
// to avoid typing all the statements manually

    integer j;

    always@(ip, weights, bias, mult) begin
        op = 0;
        for(j = 0; j < p_width; j = j + 1) begin
            op = op + mult[((2*prec*j)+(prec/2))+:prec];
        end
        op = op + bias;
    end
//////////////////////// Code ends//////////////////////////////////////
endmodule
```

b) Neuron Results (from TCL Console):

```
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave wi
#   }
# }
# run 1000ns
The module neuron.v works as intended
$finish called at time : 10 ns : File "D:/Documents/EEL4783_VHDL/project/testbench/neuron_tb.v" Line 41
INFO: [USF-XSim-96] XSim completed. Design snapshot 'neuron_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

- This shows that the Neuron module is working as intended.

c) Neuron Waveform:



d) Neuron Test Bench Condition:

```
if(op == 'b0001010000000110)
    $display("The module neuron.v works as intended");
else
    $display("The module neuron.v does not work as intended");
```

- Looking at the waveform, you can see that the output matches what is expected.

Hidden Layer:

a) Hidden Layer Source Code:

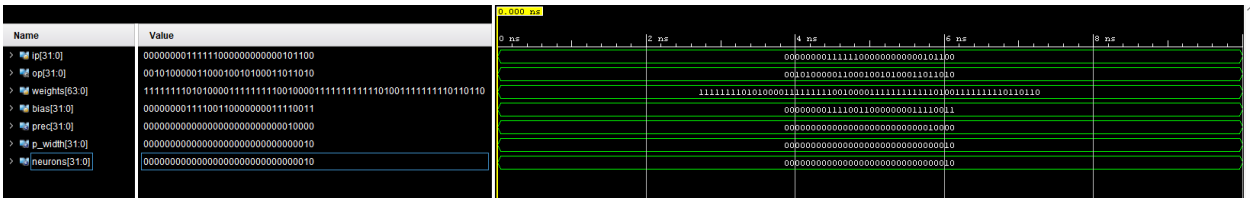
```
module hidden_layer#(
    neurons = 10,                                   // number of neurons in this layer
    p_width = 10,                                   // number of neurons in the previous layer
    prec = 16                                       // Precision of the binary numbers. Set to 16 (Do not Change)
    )(
    input wire signed [(prec*p_width)-1:0] ip,              // Flattenned input from the previous layer. Width = neurons of previous layer * precision
    output wire signed [(prec*neurons)-1:0] op,             // output of the current layer. Width = neurons of this layer * precision
    input wire signed [(prec*p_width*neurons)-1:0] weights,  // flattenned weights of the current layer. This holds all the weights of each neuron of the
    input wire signed [(prec*neurons)-1:0] bias              // flattenned bias of the neurons. Width = neurons of this layer * precision
    );

//////////////////Write your code here//////////////////////////////////////

// Use a generate statement similar to the one used in neuron.v and a for loop
// to create neurons number of instantiations and interface the appropriate weights
// and biases
    wire signed [(prec*neurons)-1:0] temp;
    reg signed [(prec*neurons)-1:0] internalReg;

    genvar i;

    generate
        for(i = 0; i < neurons; i = i + 1) begin : myNeurons
            neuron #(p_width, prec) n0(ip,temp[(prec*(i+1)-1):prec*i] , weights[(prec*p_width*(i+1)-1):prec*p_width*i], bias[(prec*(i+1)-1):prec*i]);
        end
    endgenerate
    //op[(prec*(i+1)-1):prec*i]

// Take the concatenated output of each neuron and pass them through the ReLU activation
// function. Refer to section 4.5 of the ProjectInstructions.pdf for the pseudo code

    integer j;
    always@(*) begin
        internalReg = temp;
        for(j = 0; j < neurons; j = j + 1) begin
                if(internalReg[(prec*j)+:prec] <= 0) begin
                    internalReg = 0;
                end
        end
    end

    assign op = internalReg;
//////////////////////// Code ends//////////////////////////////////////

endmodule
```

b) Hidden Layer Results (from TCL Console):

```
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window.
#   }
# }
# run 1000ns
The module hidden_layer.v works as intended
$finish called at time : 10 ns : File "D:/Documents/EEL4783_VHDL/project/testbench/hidden_layer_tb.v" Line 42
INFO: [USF-XSim-96] XSim completed. Design snapshot 'hidden_layer_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

- This shows that the Hidden Layer module is working as intended.

c) Hidden Layer Waveform:



d) Hidden Layer Test Bench Condition:

```
if(op == 'b00101000001100010010100011011010)
    $display("The module hidden_layer.v works as intended");
else
    $display("The module hidden_layer.v does not work as intended");
```

- Looking at the waveform, you can see that the output matches what is expected.