**ASSIGNMENT 1 REPORT**

**NAME: SAGAR LADLA**

**BITSID: 2024HT01123**

**SUB: HARDWARE SOFTWARE CO-DESIGN**

For the simple feed forward neural network with specifications:

1. Input layer: 4 neurons each 8-bit wide
2. Hidden layer: 3 neurons with ReLU function
3. Output layer: 2 neurons each 8-bit wide
4. INPUT LAYER

I have taken input layers as:

1. input [7:0] i0,
2. input [7:0] i1,
3. input [7:0] i2,
4. input [7:0] i3,

And weights of input layers as:

For i0 3 weights as:

1. input [7:0] i0w0,
2. input [7:0] i0w1,
3. input [7:0] i0w2,

For i1 3 weights as:

1. input [7:0] i1w0,
2. input [7:0] i1w1,
3. input [7:0] i1w2,

For i2 3 weights as:

1. input [7:0] i2w0,
2. input [7:0] i2w1,
3. input [7:0] i2w2,

For i3 3 weights as:

1. input [7:0] i3w0,
2. input [7:0] i3w1,
3. input [7:0] i3w2,

To hold the multiplication part of each hidden neuron I have used:

* Since two 8-bit integers (input neuron and weight) will produce 16-bit result, so I have taken wire variable as 16-bit

For h0 hidden neuron:

1. wire [15:0] h00;
2. wire [15:0] h01;
3. wire [15:0] h02;
4. wire [15:0] h03;

They are computed as:

1. assign h00 = i0 \* i0w0;
2. assign h01 = i1 \* i1w0;
3. assign h02 = i2 \* i2w0;
4. assign h03 = i3 \* i3w0;

To hold the result of SOP of each hidden neuron along with bias, we have used wire variables:

1. wire [11:0] h\_0\_sum;
2. wire [11:0] h\_1\_sum;
3. wire [11:0] h\_2\_sum;

Since addition of 4 variables is going to add more bits, that’s why we have used the wire variables with extra 4 bits

Their result is going to be like:

* We are discarding the lower 8-bits so as to give the precision of 8-bits (which is the size of hidden neuron as well)

assign h\_0\_sum = h00[15:8] + h01[15:8] + h02[15:8] + h03[15:8] + h0b;

* Here `h0b` is the bias of h0 hidden neuron

**ReLU activation function is designed in Verilog as:**

function [7:0] relu(input [7:0] sigma);

begin

relu = (sigma > 0) ? sigma : 0;

end

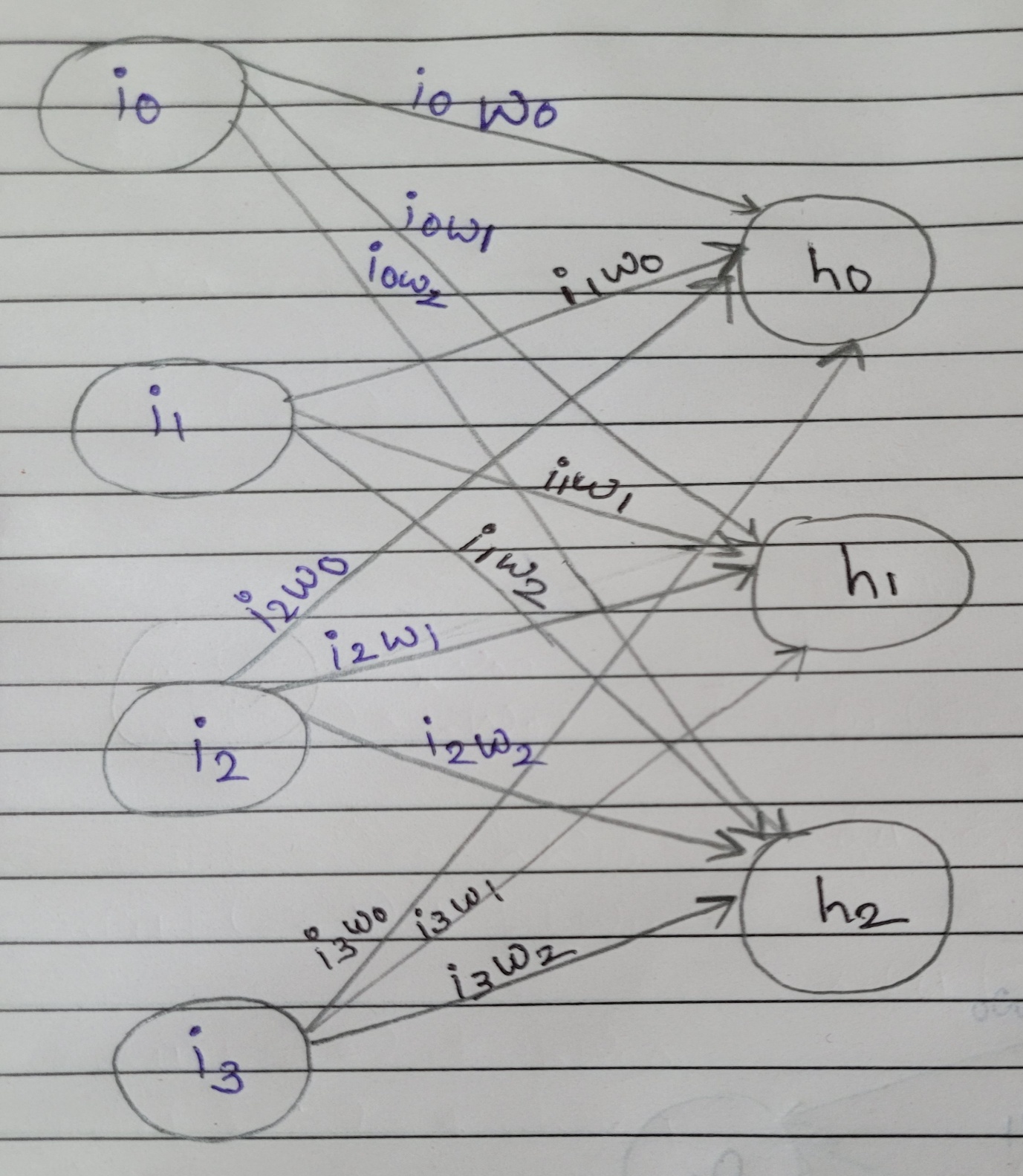
endfunction

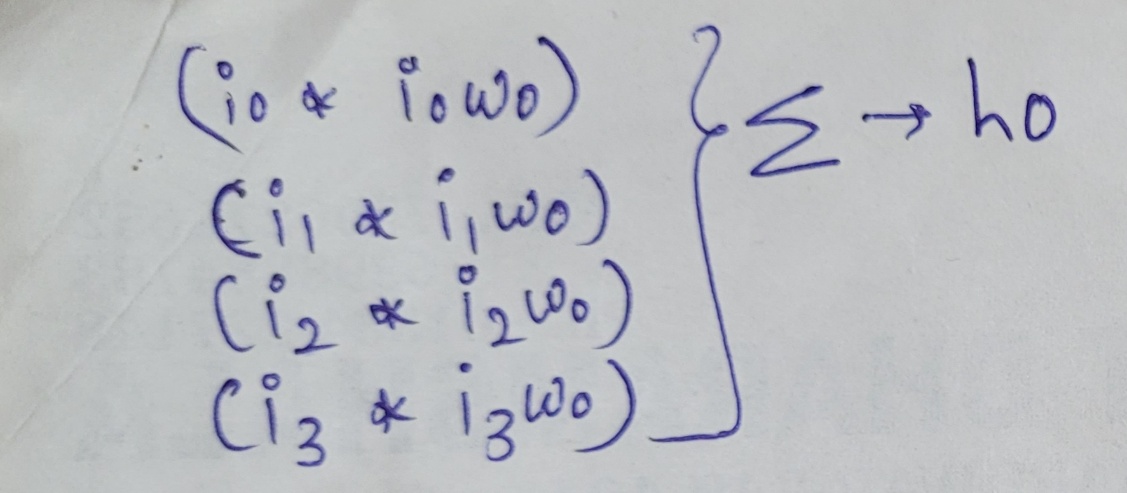
Then we will pass the result of SOP to ReLU function of hidden neuron.

* We are trimming the lower 4-bits to make result accommodate into 8-bit h0 hidden neuron.

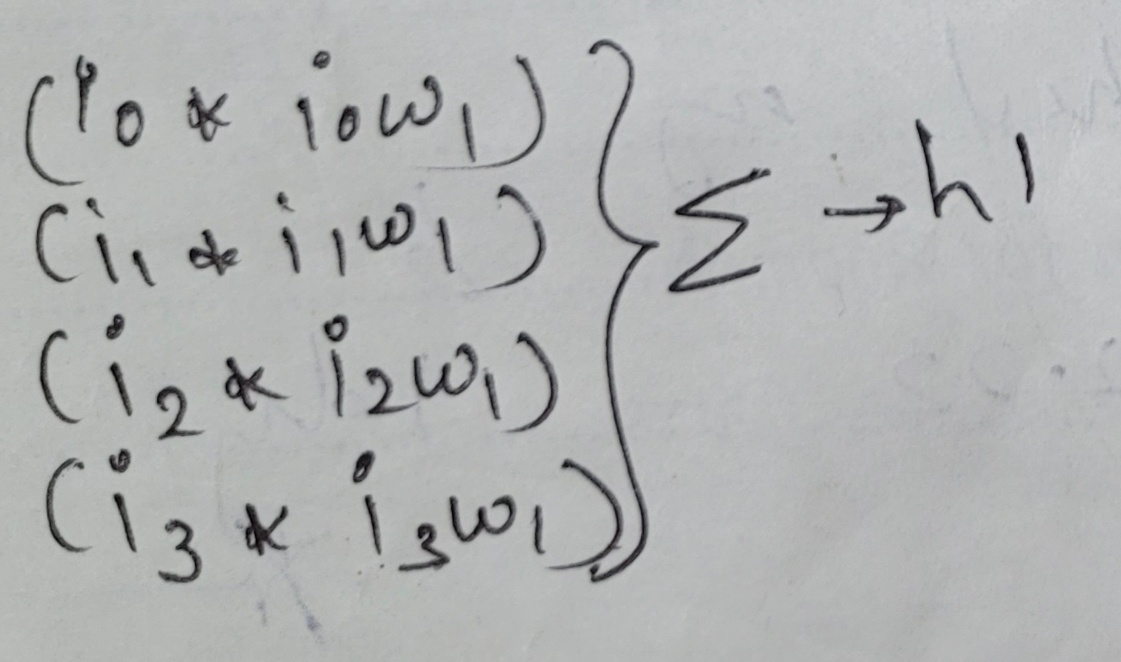
assign h0 = relu(h\_0\_sum[11:4]);

Same process we have also followed for the rest of hidden neurons – h1 and h2.

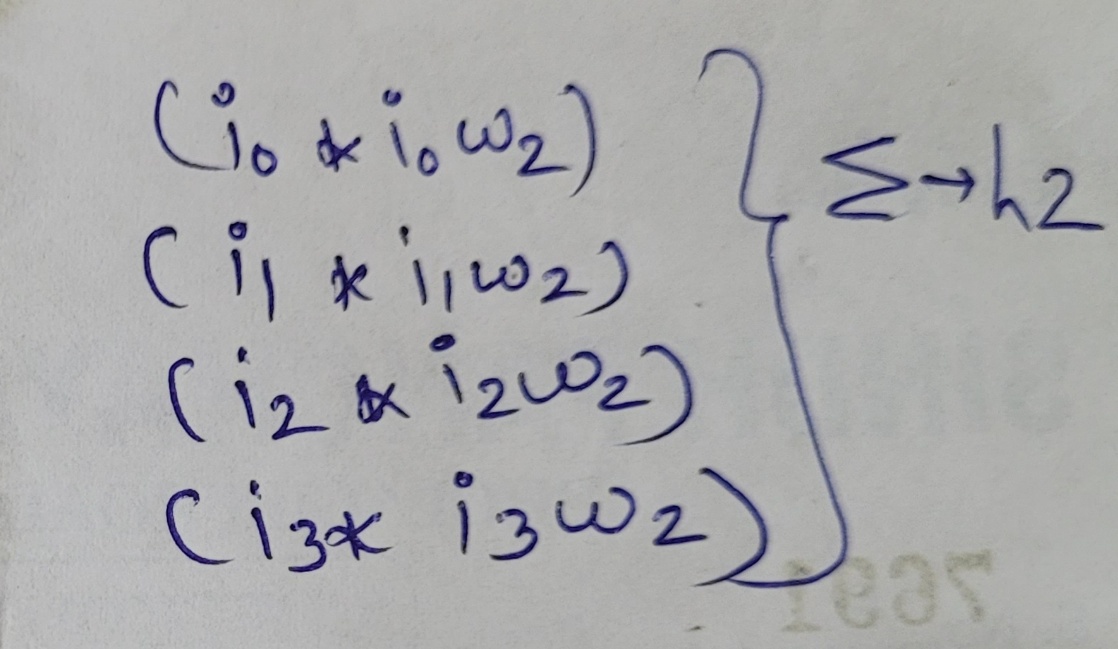
It can be summed up as:



+ h0b (h2 bias)



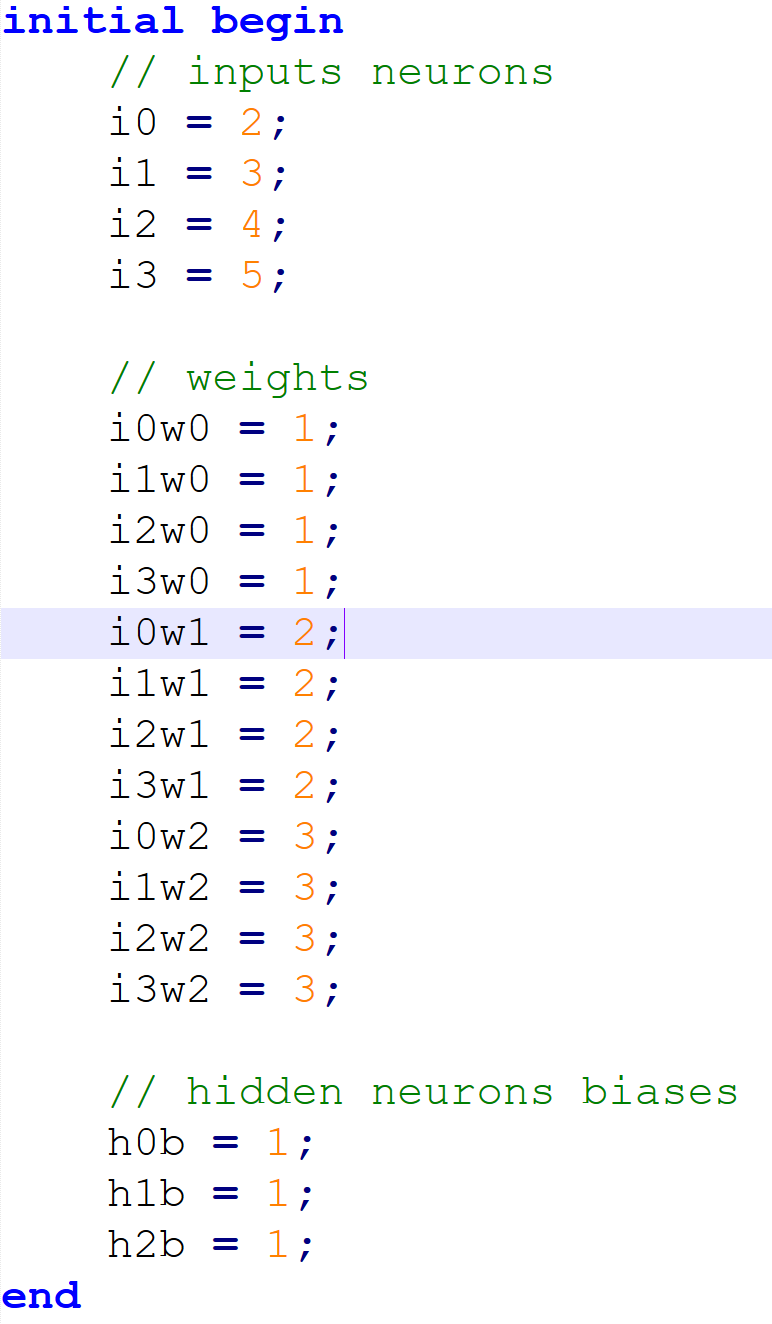
+ h1b (h2 bias)

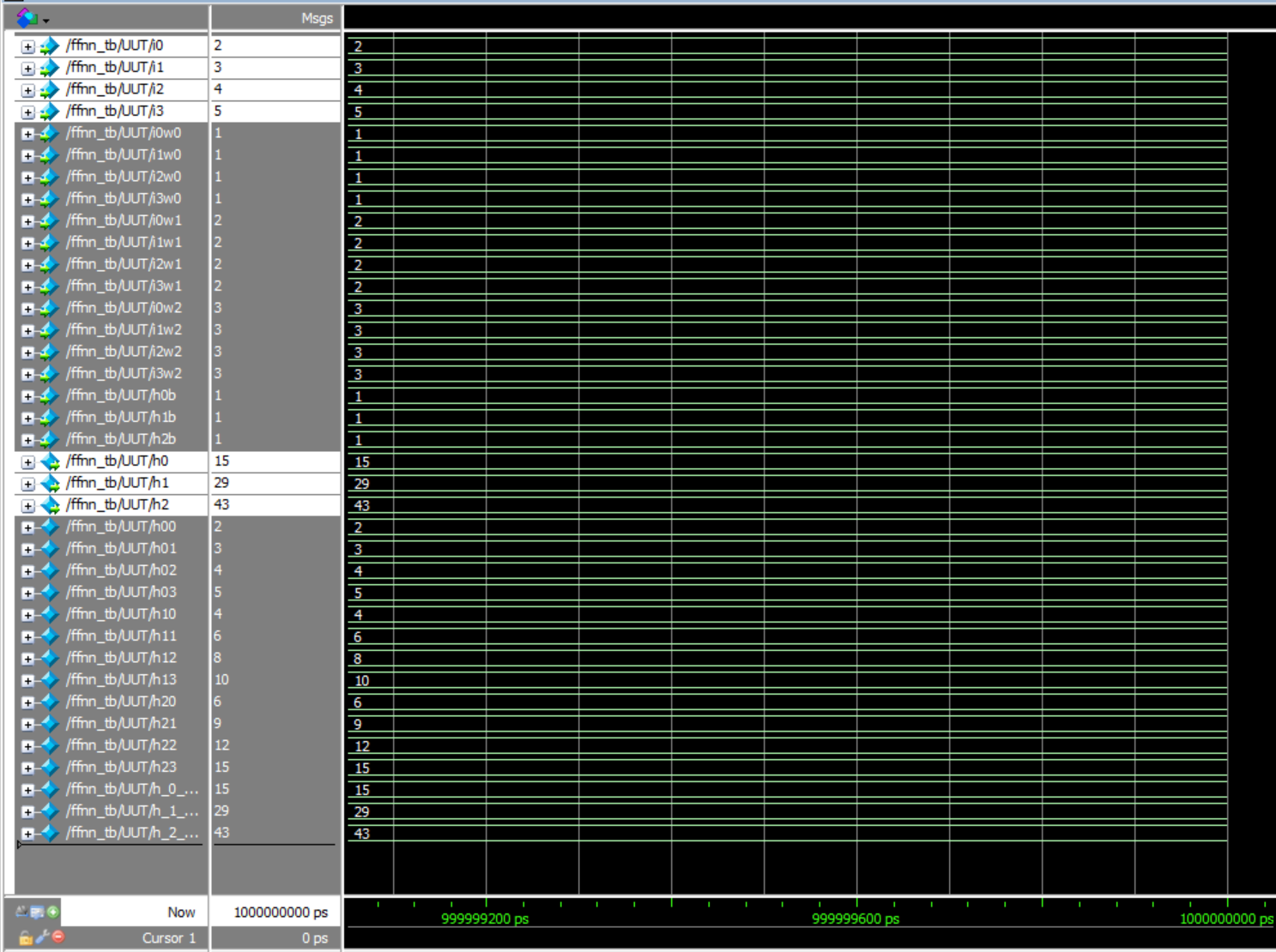


+ h2b (h2 bias)

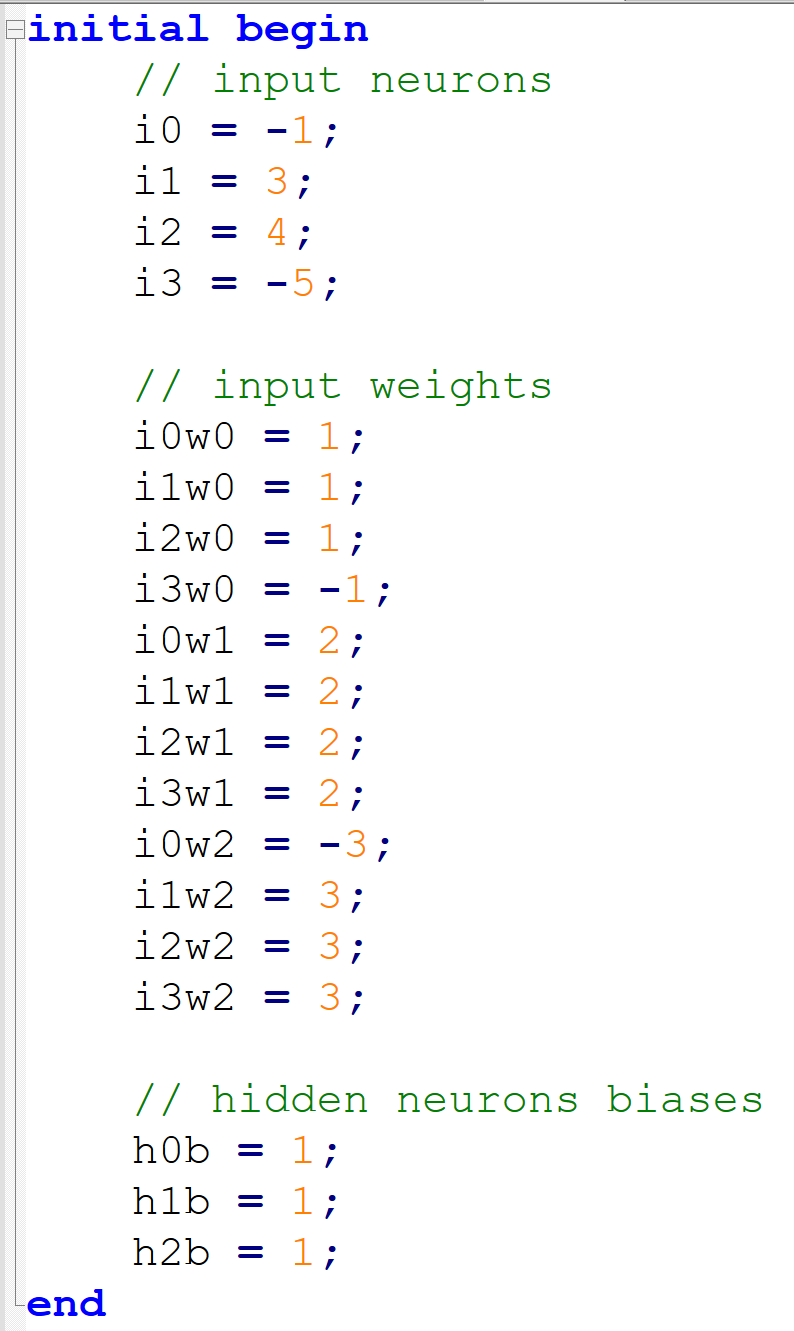
**Simulation 1**

With input layers as

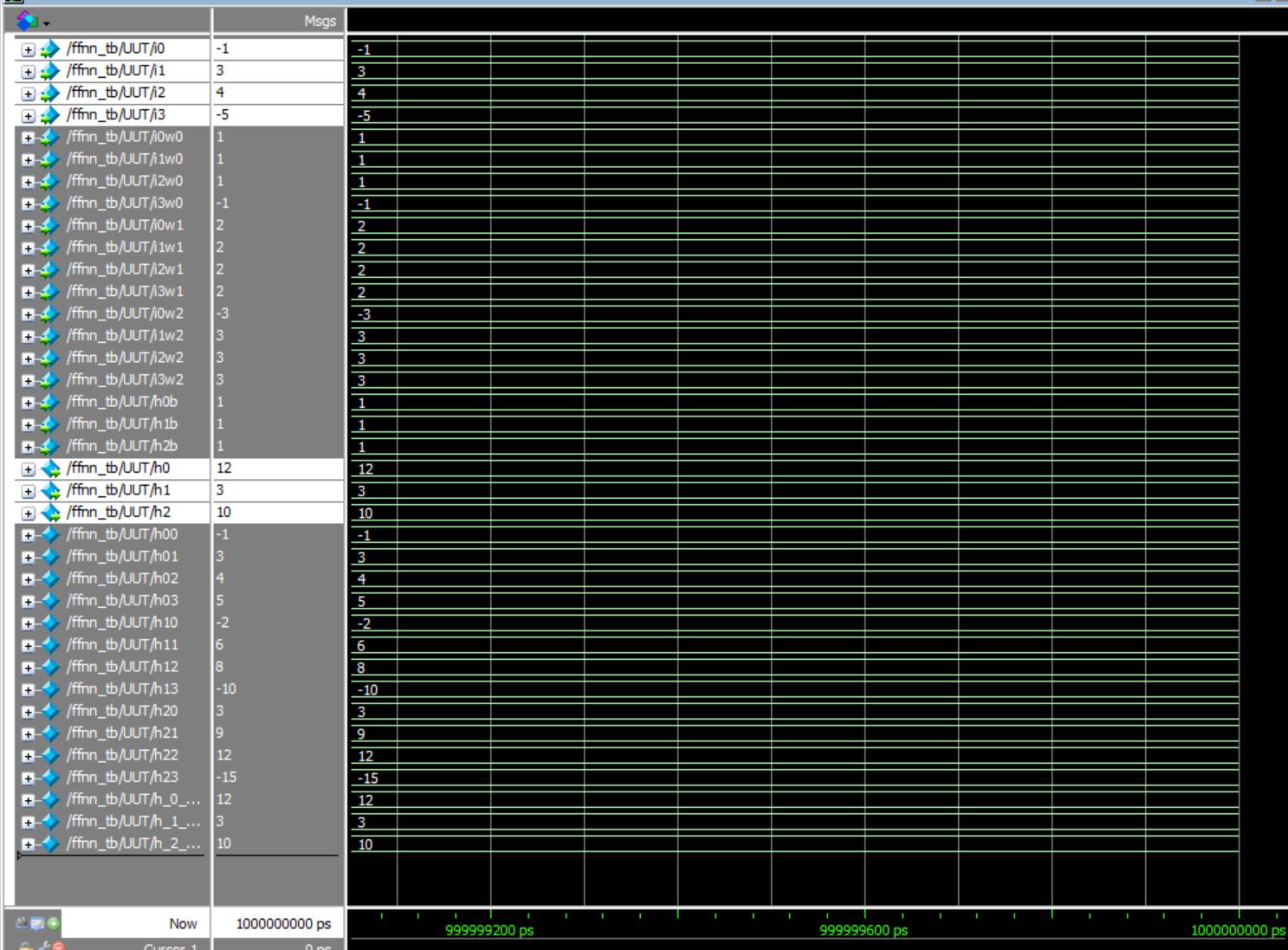


Hidden neurons h0, h1 and h2:

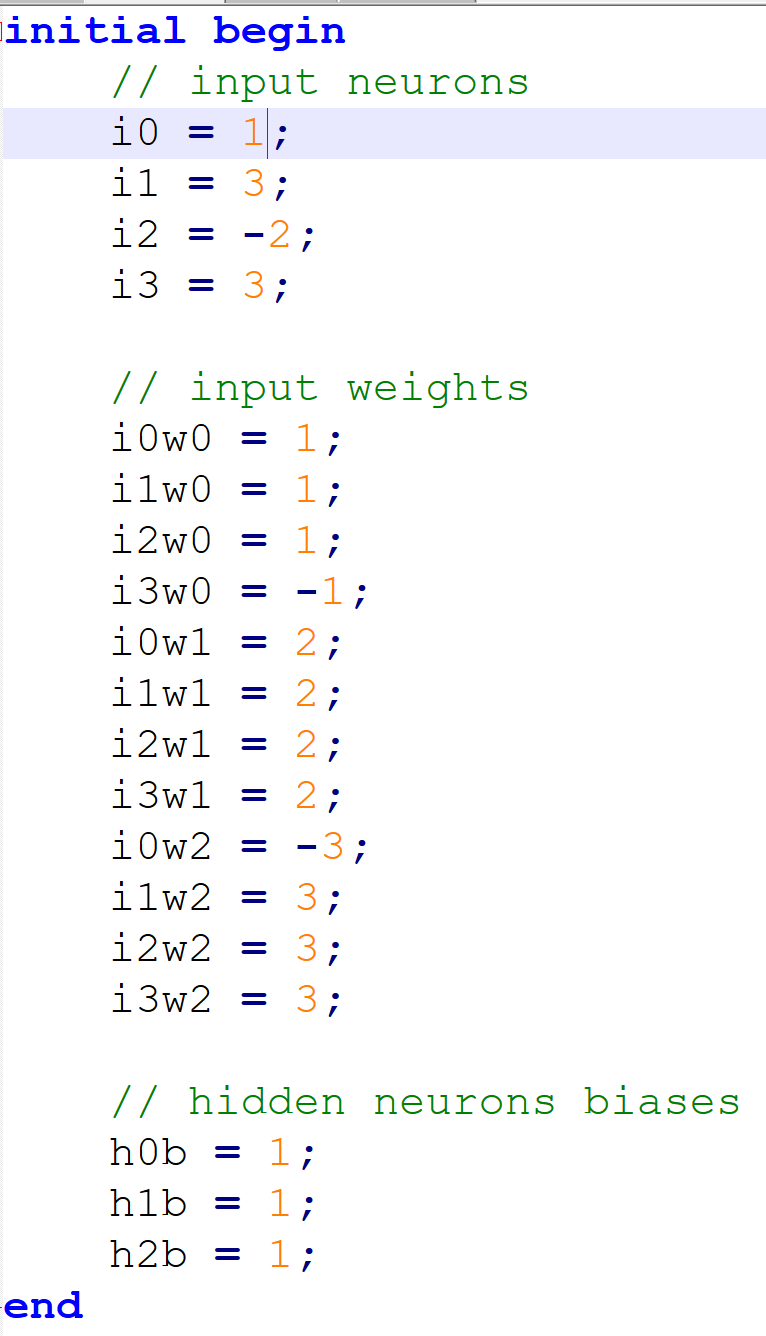
**Simulation 2**

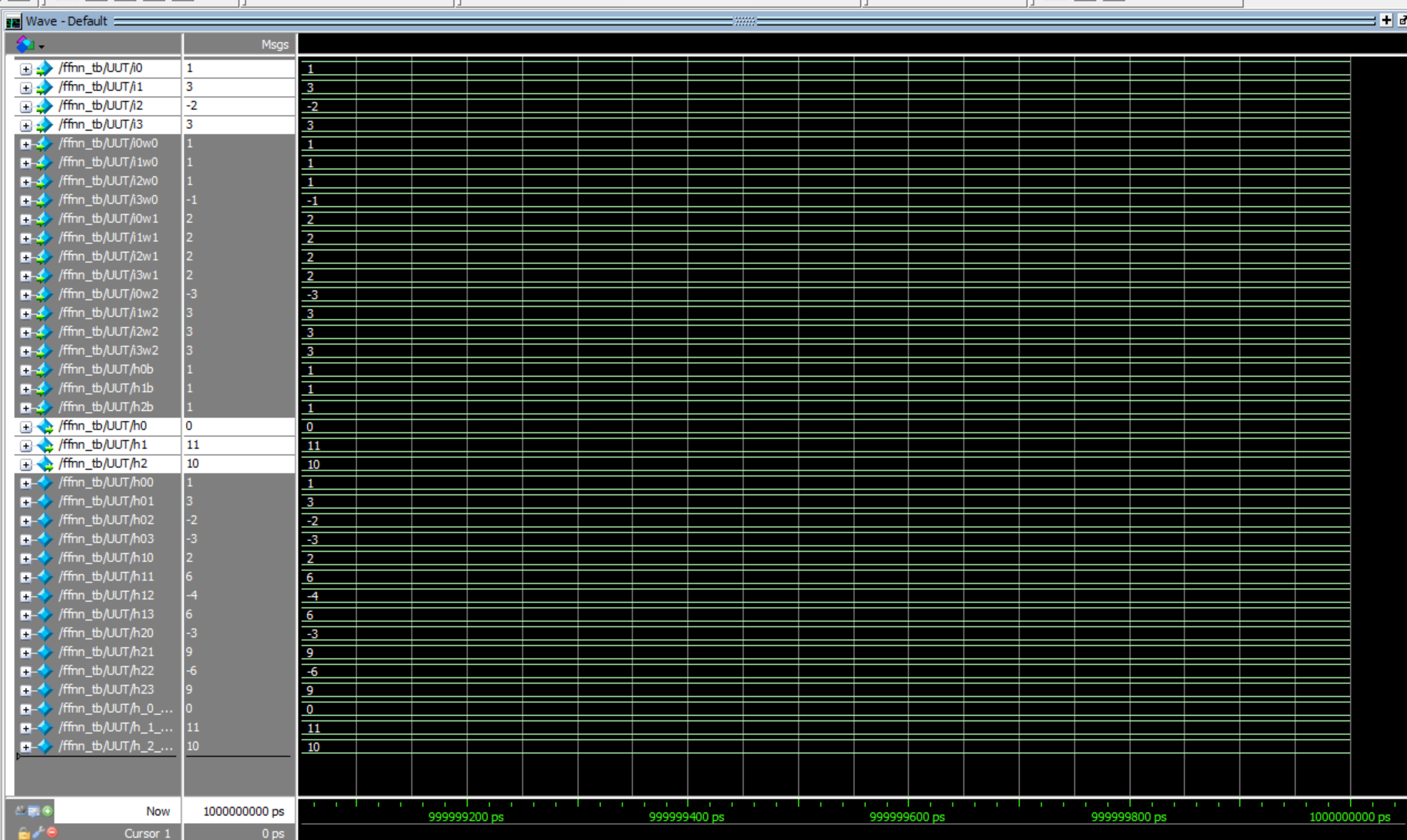


Hidden neurons h0, h1 and h2



**Simulation 3**

****

Hidden neurons h0, h1 and h2

II) HIDDEN INPUT LAYER

1. input [7:0] h0,
2. input [7:0] h1,
3. input [7:0] h2,

And weights of hidden input layers are:

For h0, 2 weights:

1. input [7:0] h0w0,
2. input [7:0] h0w1,

For h1, 2 weights:

1. input [7:0] h1w0,
2. input [7:0] h1w1,

For h2, 2 weights:

1. input [7:0] h2w0,
2. input [7:0] h2w1,

To hold the multiplication part of each output neuron I have used:

* Since two 8-bit integers (hidden input neuron and weight) will produce 16-bit result, so I have taken wire variable as 16-bit

For out\_0 output neuron:

1. wire [7:0] out\_00;
2. wire [7:0] out\_01;
3. wire [7:0] out\_02;

They are computed as,

1. assign out\_00 = h0 \* h0w0;
2. assign out\_01 = h1 \* h1w0;
3. assign out\_02 = h2 \* h2w0;

To hold the result of SOP of each output neuron along with bias, we have used wire variables:

1. wire [7:0] out\_0\_sum;
2. wire [7:0] out\_1\_sum;

Since addition of 4 variables is going to add more bits, that’s why we have used the wire variables with extra 4 bits

Their result is going to be like:

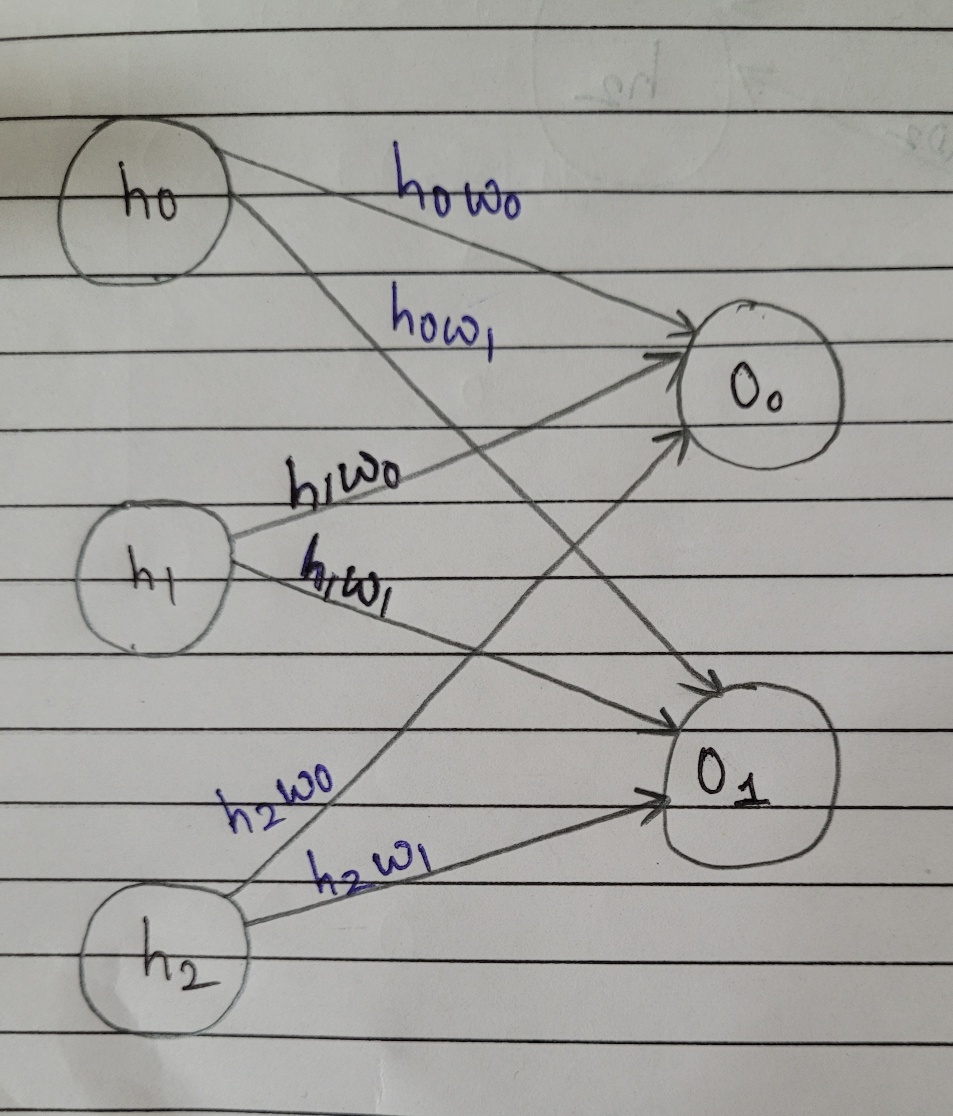
* We are discarding the lower 8-bits so as to give the precision of 8-bits (which is the size of hidden neuron as well)

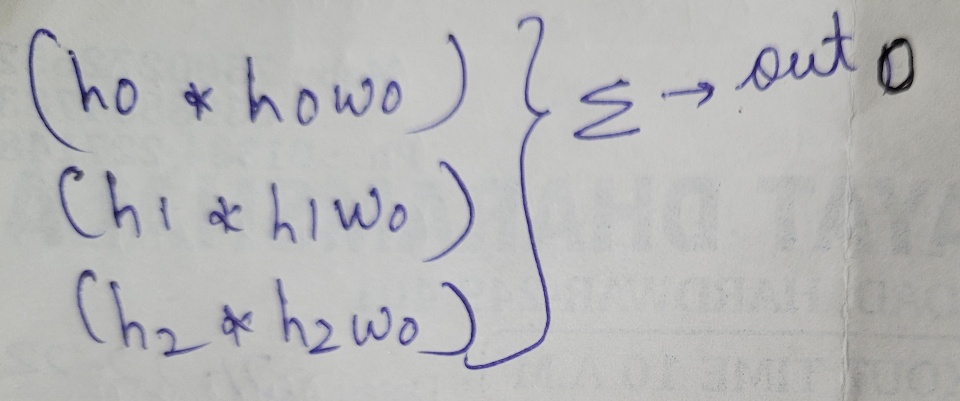
assign out\_0\_sum = out\_00 + out\_01 + out\_02 + out\_0b;

* Here `out\_0b` is the bias of out\_0 output neuron

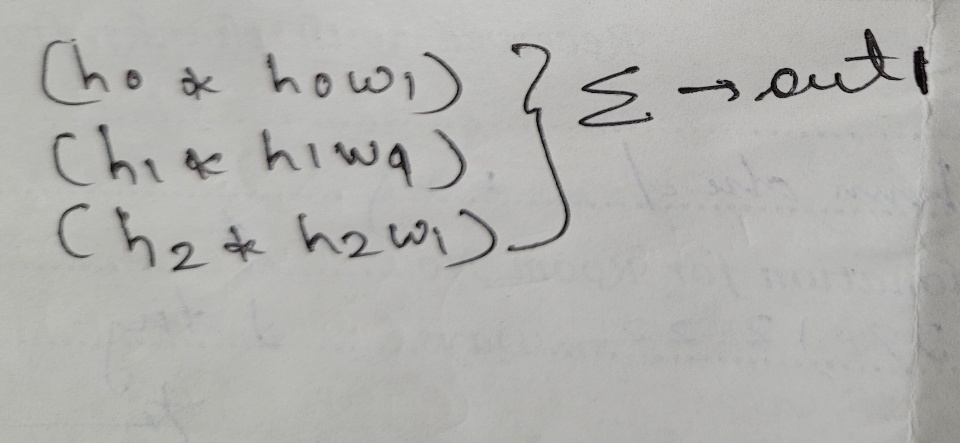
Same process we have also followed for out\_1 output neuron.

It can be summed up as:



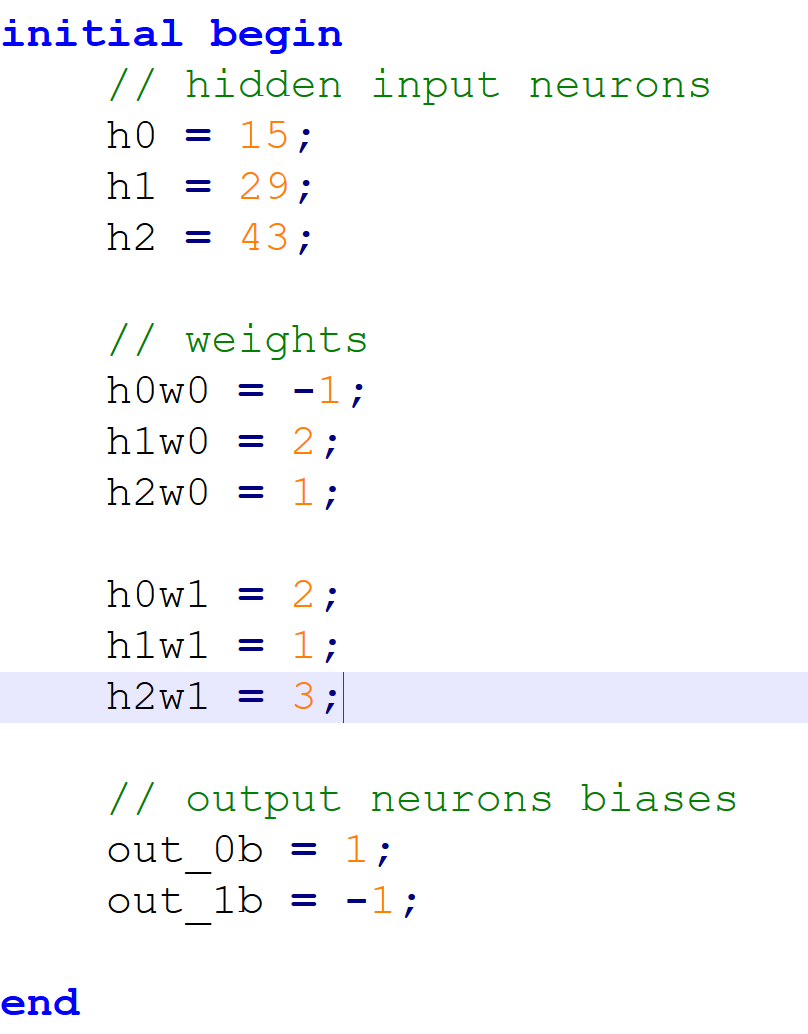


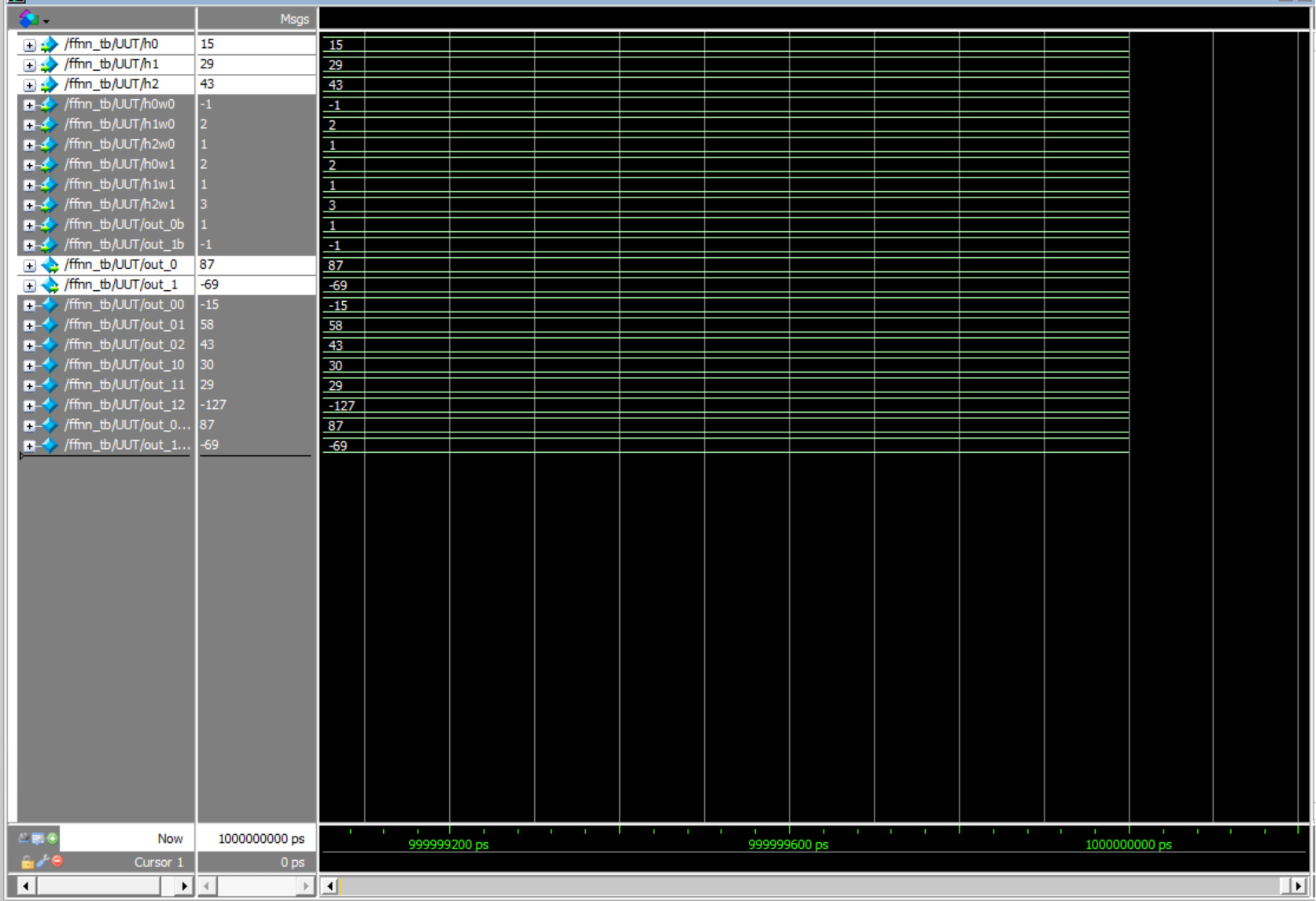
+ out\_0b



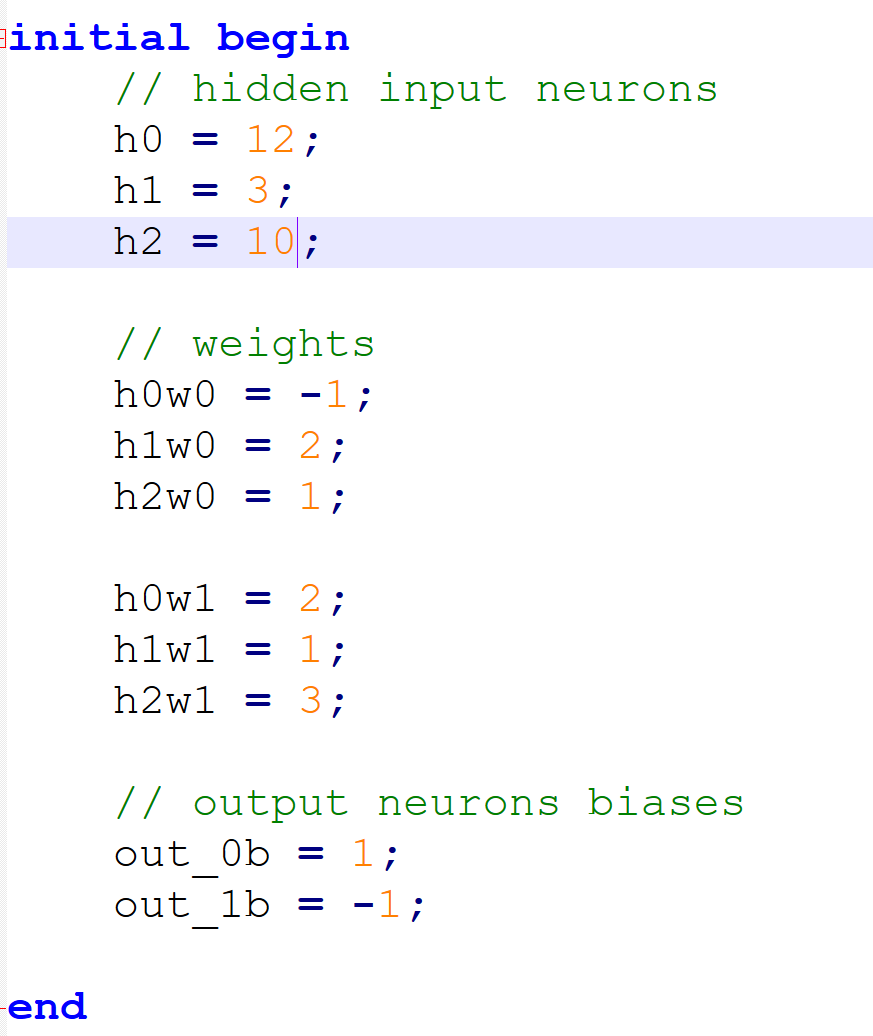
+ out\_1b

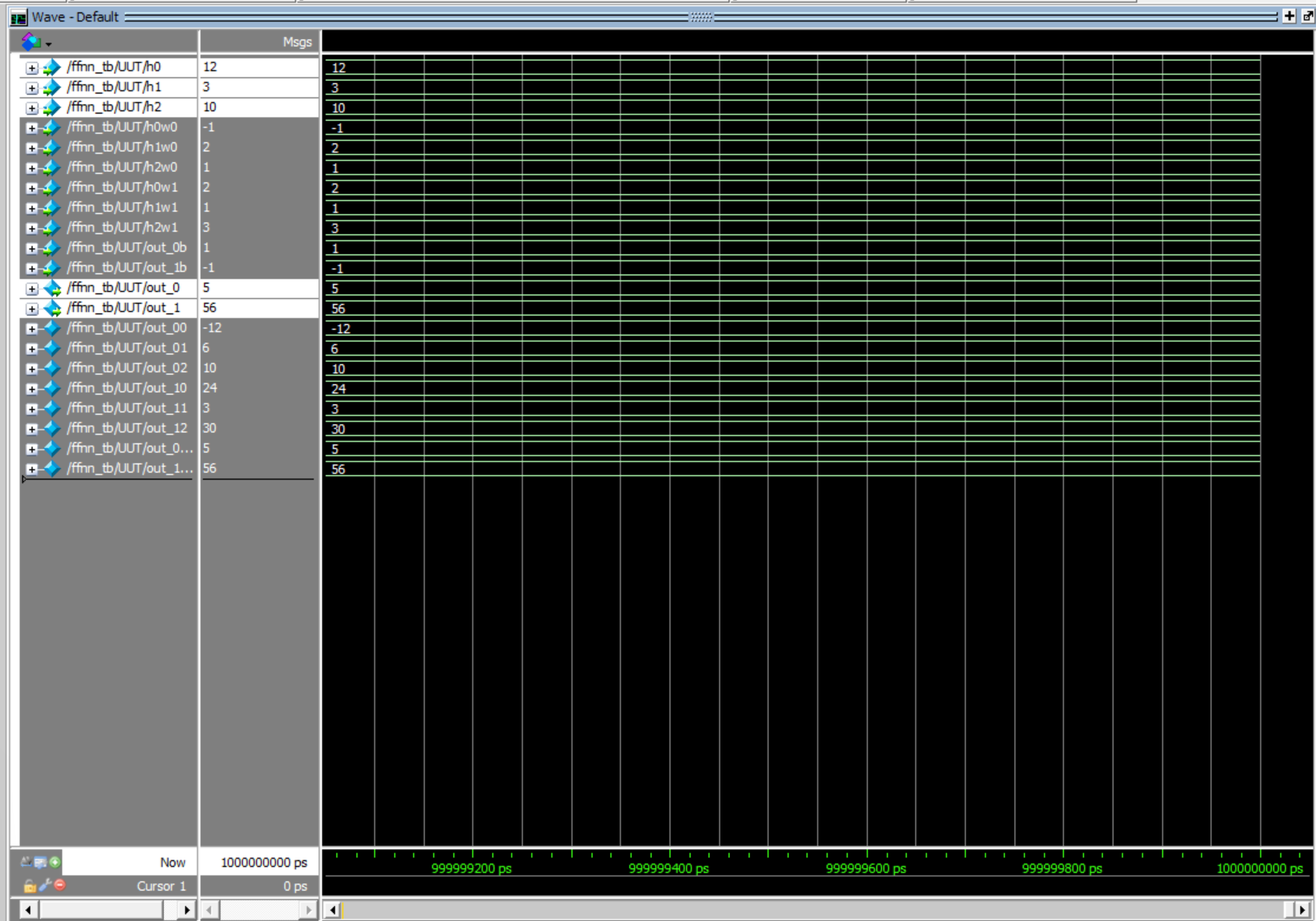
**Simulation 1:**

****

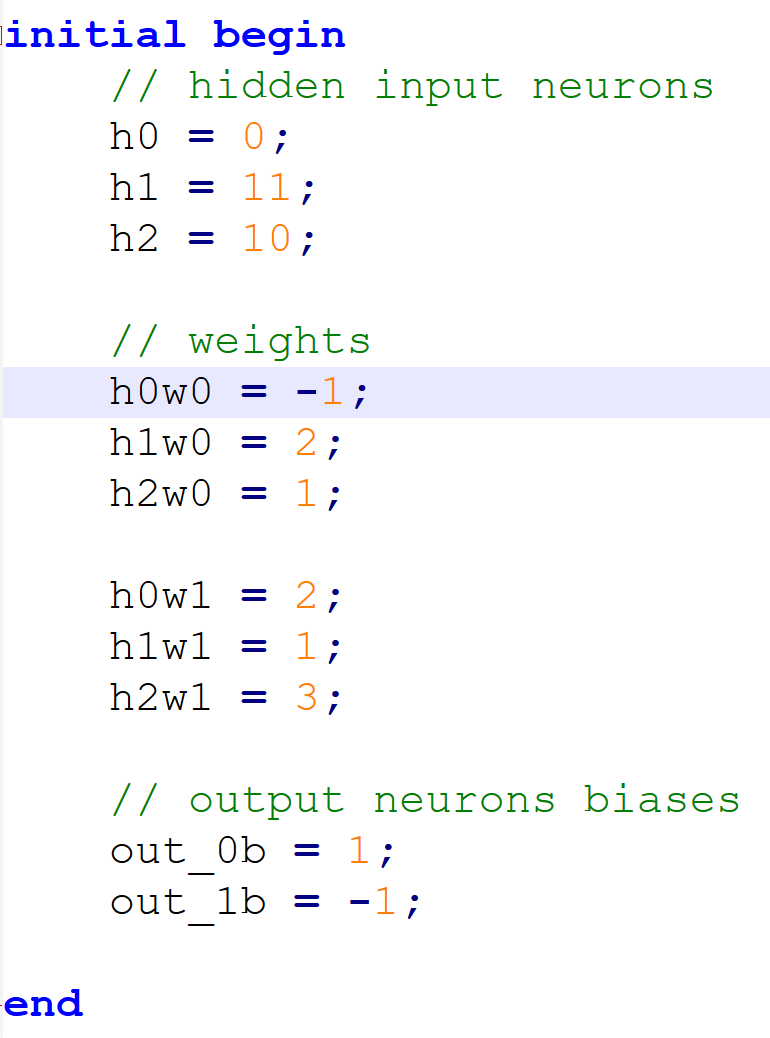
**Output neurons out\_0 and out\_1**

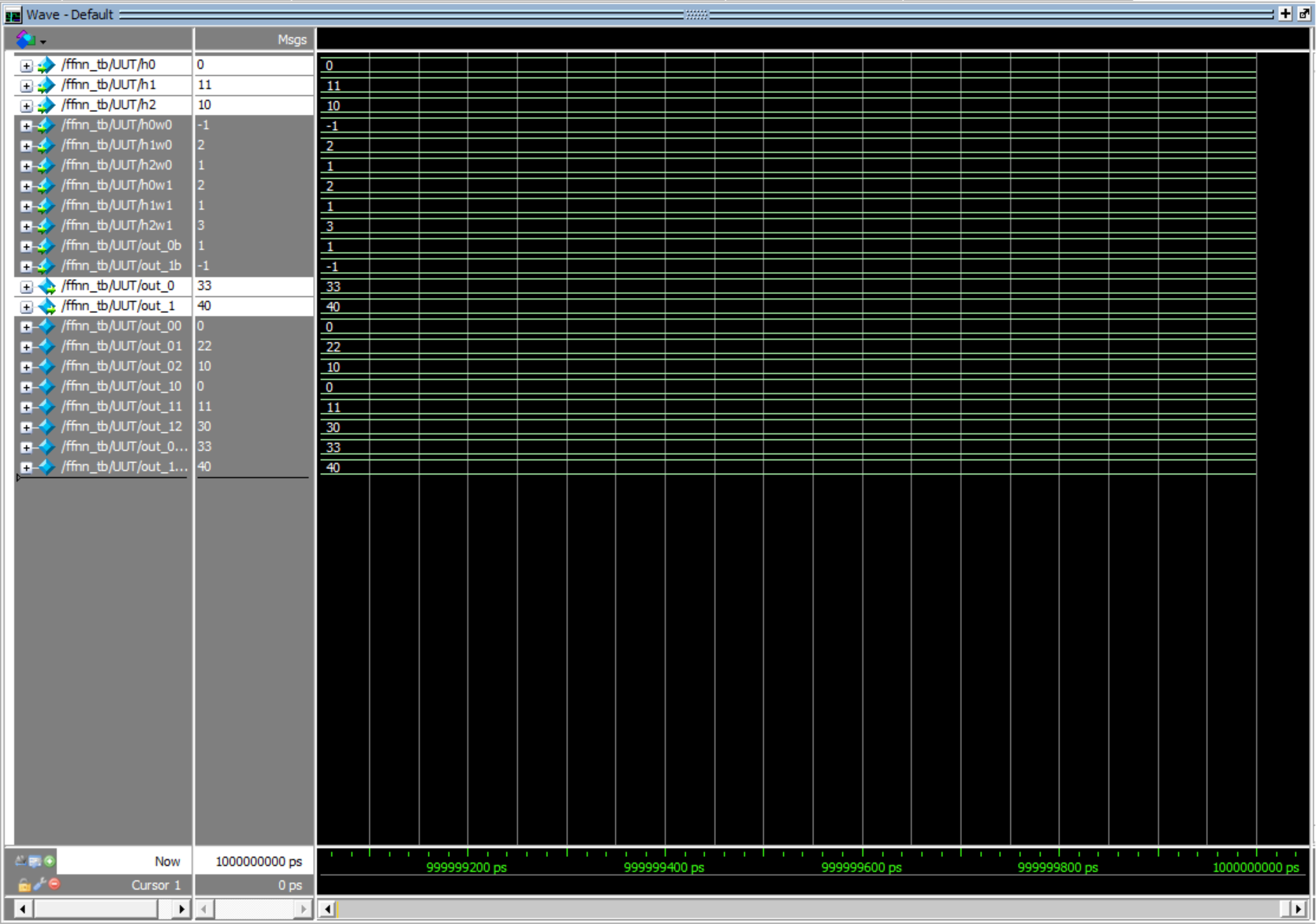
**Simulation 2:**

****

**Output neurons out\_0 and out\_1**

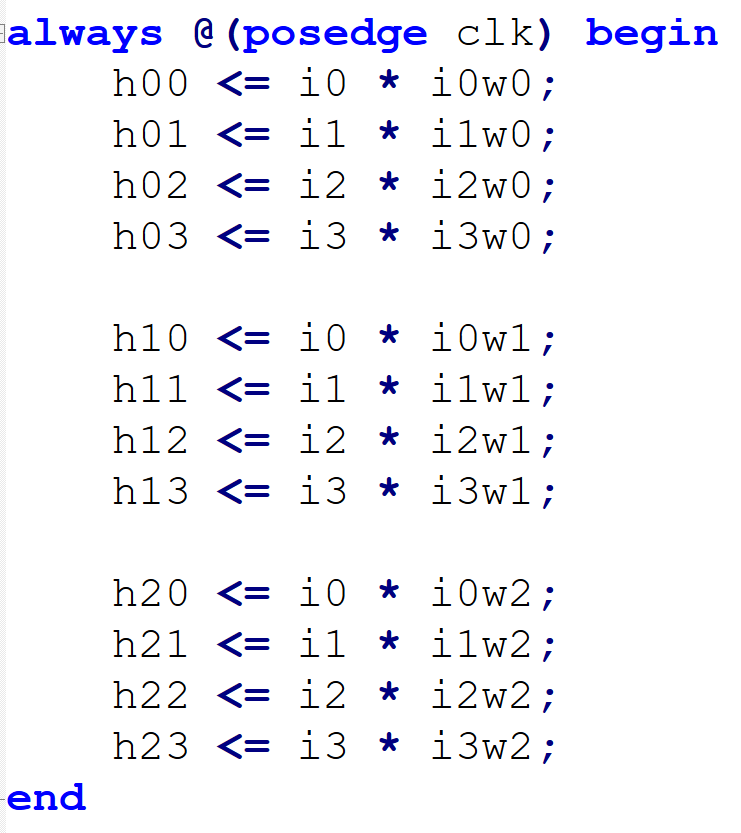
**Simulation 3:**

****

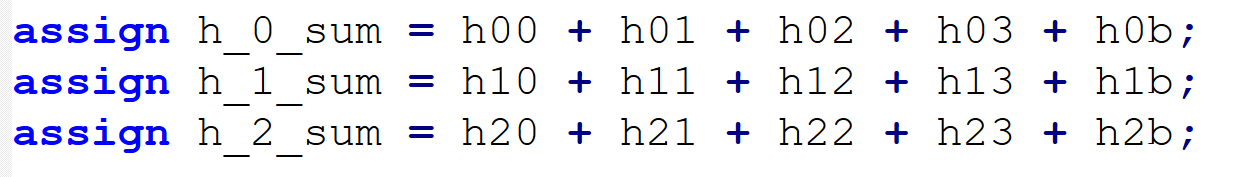
**Output neurons out\_0 and out\_1**

**Report:**

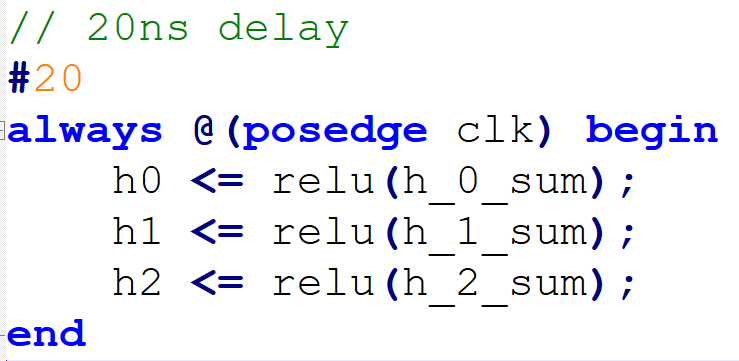
Initially I designed logic to implement the Arithmetic operations in a single clock cycle for faster operation (sequentially).



And then large additions combinatorically:



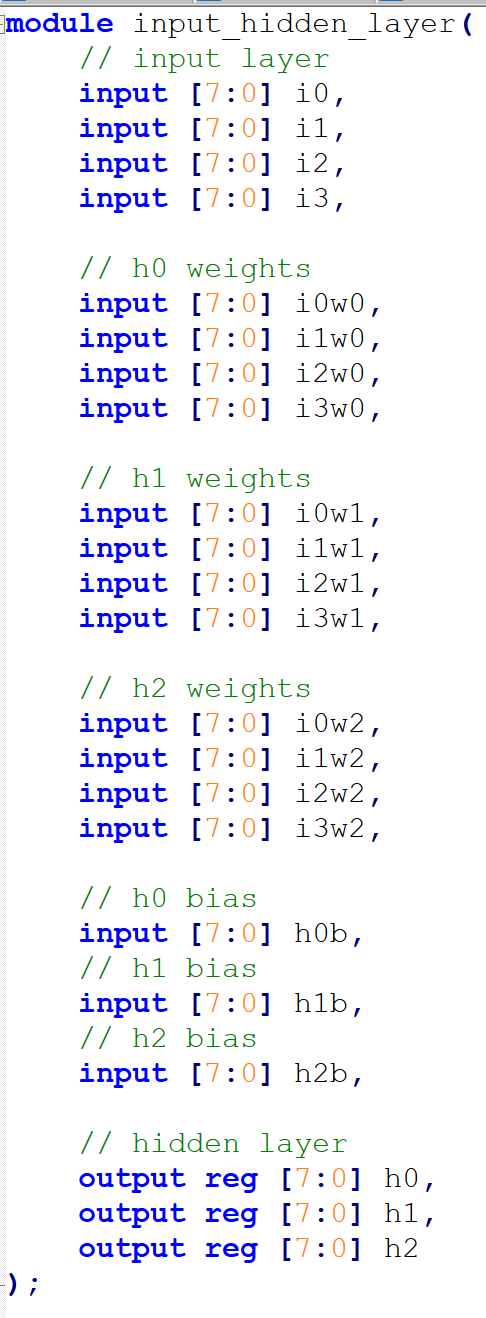
And delay for next sequential logic which required for combinatorial logic

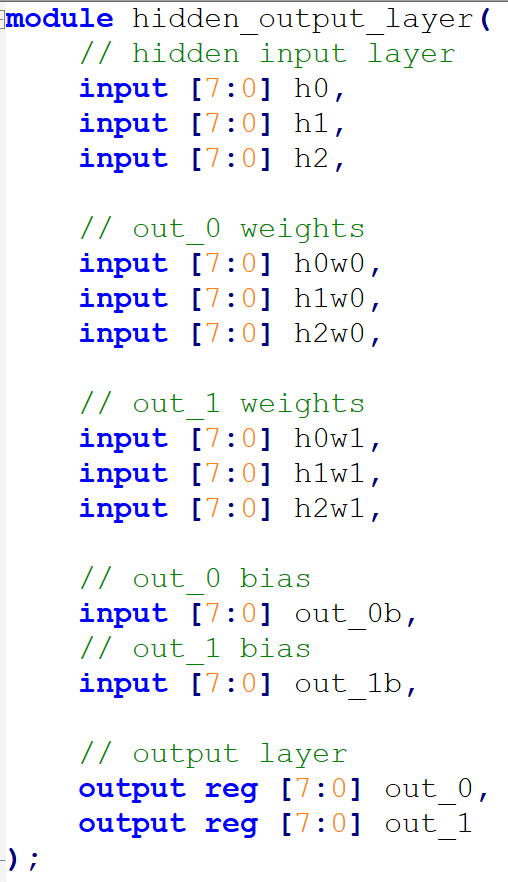


But faced few challenges as we cannot use delay in module logic itself.

So, I designed the logic to use combinatorial logic for all computations.

2. Second, I tried designing each layer as individual module and then combining them together into to parent ***ffnn*** module. But it was difficult debugging the design.





But was facing difficulties assembling both modules in top-level ***ffnn*** module.

2. Another challenge which I faced was handling hidden neuron layer, so I broke the design into

1. input to hidden neuron
2. hidden neuron to output

Then tested the both designs by wiring the output of (i) input to hidden neuron to (ii) hidden neuron to output, to get the Output result.