ASSIGNMENT 1 REPORT

NAME: SAGAR LADLA

BITSID: 2024HT01123

SUB: HARDWARE SOFTWARE CO-DESIGN

For the simple feed forward neural network with specifications:

i) Input layer: 4 neurons each 8-bit wide
ii) Hidden layer: 3 neurons with ReLU function
iii) Output layer: 2 neurons each 8-bit wide

I. INPUT LAYER

I have taken input layers as:

i) input [7:0] i0,
ii) input [7:0] i1,
iii) input [7:0] i2,
iv) input [7:0] i3,

And weights of input layers as:

For i0 3 weights as:

i) input [7:0] i0w0,
ii) input [7:0] i0w1,
iii) input [7:0] i0w2,

For i1 3 weights as:

i) input [7:0] i1w0,
ii) input [7:0] i1w1,
iii) input [7:0] i1w2,

For i2 3 weights as:

i) input [7:0] i2w0,
ii) input [7:0] i2w1,
iii) input [7:0] i2w2,

For i3 3 weights as:

i) input [7:0] i3w0,
ii) input [7:0] i3w1,
iii) 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:

i) wire [15:0] h00;
ii) wire [15:0] h01;
iii) wire [15:0] h02;
iv) wire [15:0] h03;

They are computed as:

i) assign h00 = i0 * i0w0;
ii) assign h01 = i1 * i1w0;
iii) assign h02 = i2 * i2w0;
iv) assign h03 = i3 * i3w0;

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

i) wire [11:0] h_0_sum;
ii) wire [11:0] h_1_sum;
iii) 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
```

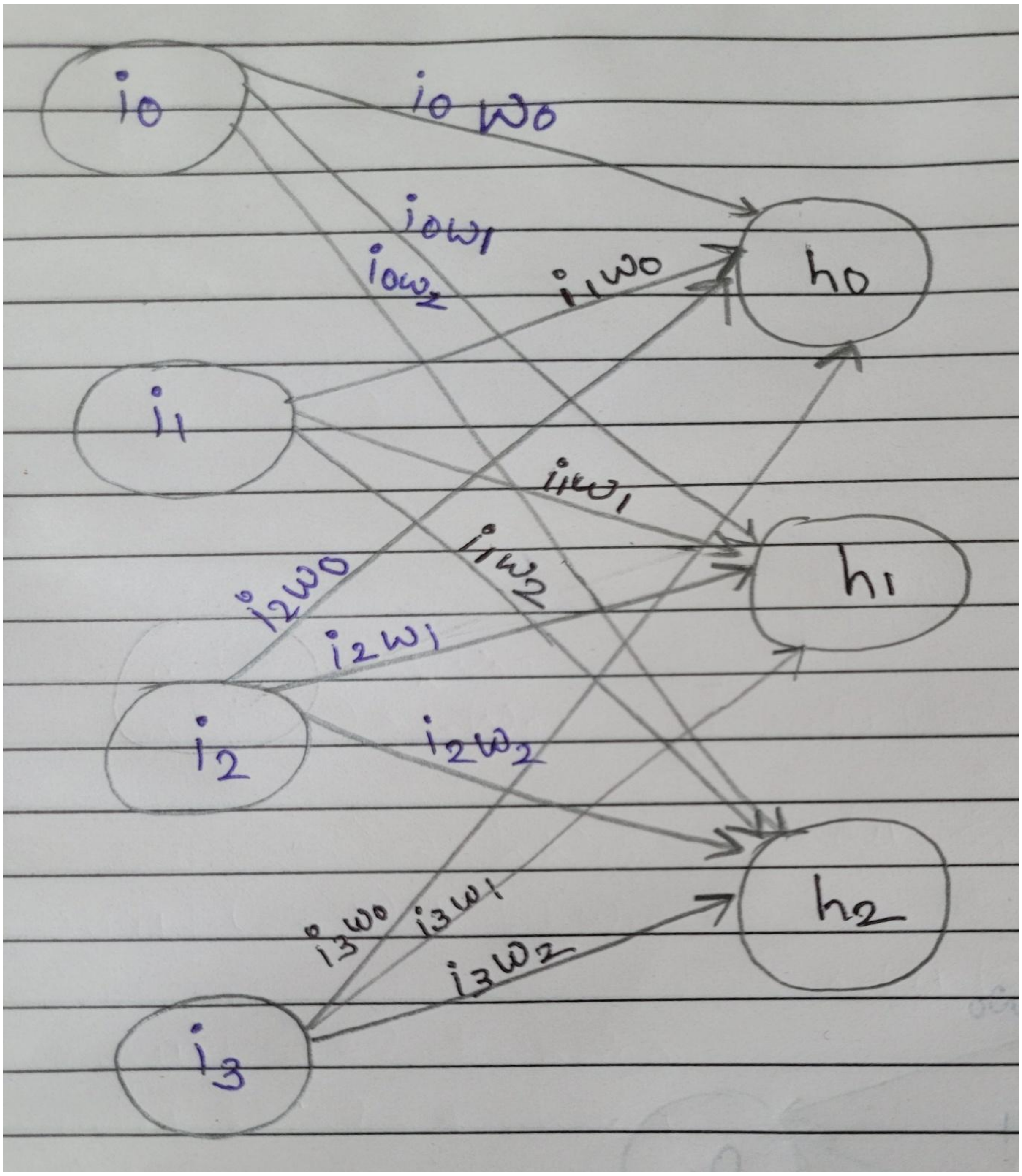
Then we will pass the result of SOP to ReLU function of hidden neuron.

- 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:



$$\left.\begin{array}{l}(i_0 * i_0w_0)\\(i_1 * i_1w_0)\\(i_2 * i_2w_0)\\(i_3 * i_3w_0)\end{array}\right\}\Sigma \rightarrow h0$$

+ h0b (h2 bias)

$$\left.\begin{array}{l}(i_0 * i_0w_1)\\(i_1 * i_1w_1)\\(i_2 * i_2w_1)\\(i_3 * i_3w_1)\end{array}\right\}\Sigma \rightarrow h1$$

+ h1b (h2 bias)

$$\left.\begin{array}{l}(i_0 * i_0w_2)\\(i_1 * i_1w_2)\\(i_2 * i_2w_2)\\(i_3 * i_3w_2)\end{array}\right\}\Sigma \rightarrow h2$$

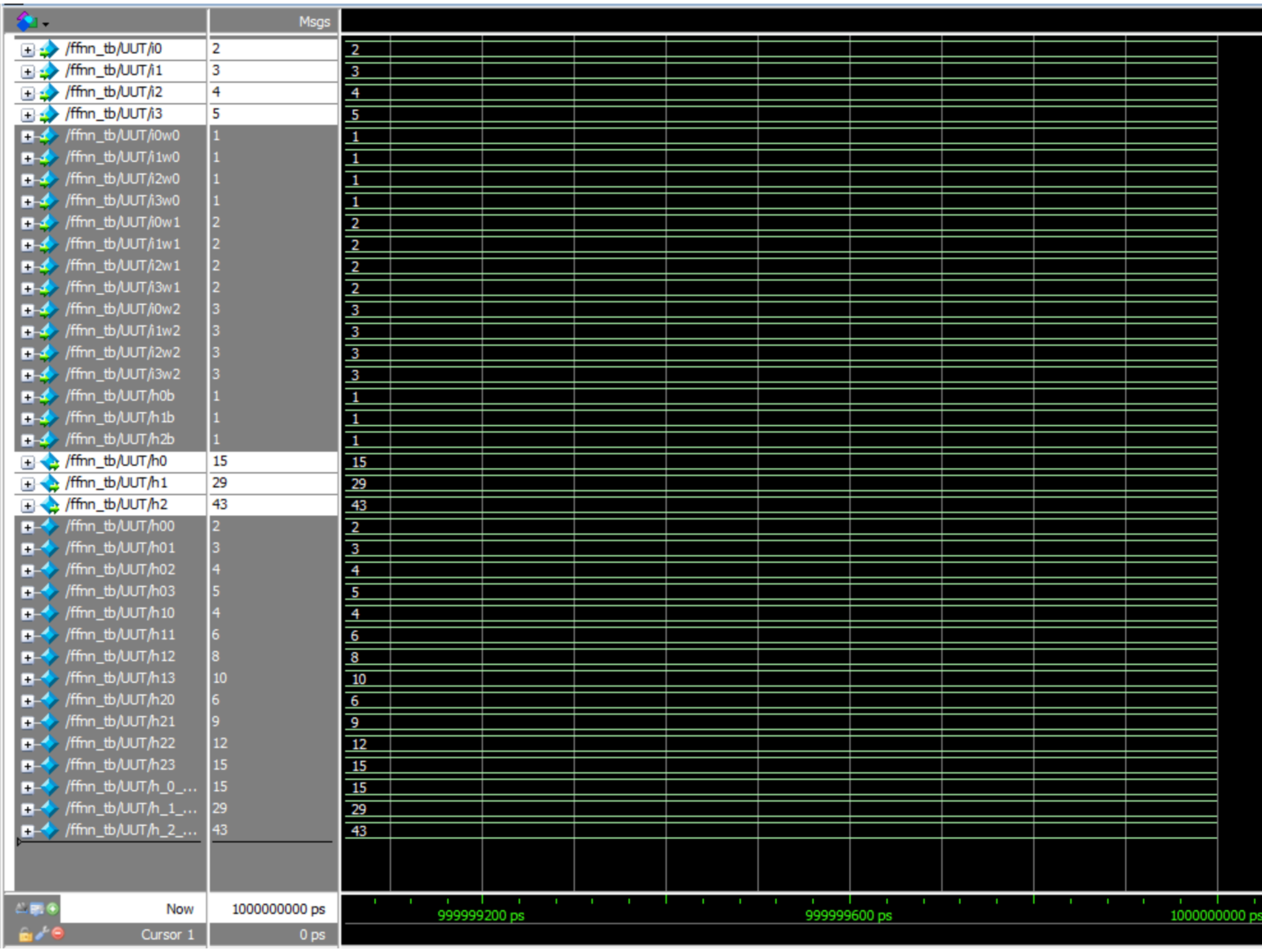
+ h2b (h2 bias)

Simulation 1

With input layers as

```
initial begin
      // inputs neurons
      i0 = 2;
      i1 = 3;
      i2 = 4;
      i3 = 5;

      // weights
      i0w0 = 1;
      i1w0 = 1;
      i2w0 = 1;
      i3w0 = 1;
      i0w1 = 2;
      i1w1 = 2;
      i2w1 = 2;
      i3w1 = 2;
      i0w2 = 3;
      i1w2 = 3;
      i2w2 = 3;
      i3w2 = 3;

      // hidden neurons biases
      h0b = 1;
      h1b = 1;
      h2b = 1;
end
```

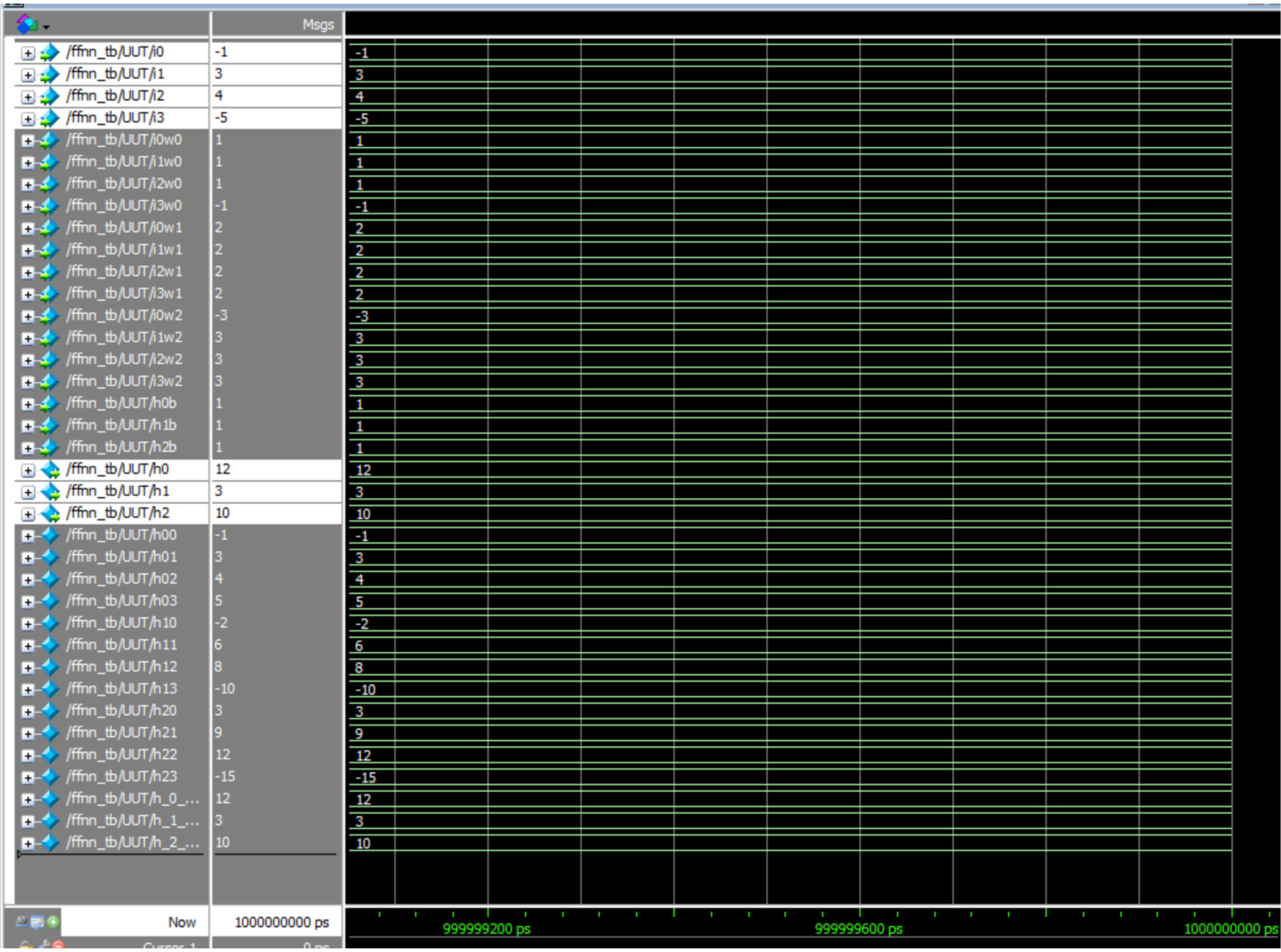
Hidden neurons h0, h1 and h2:



Simulation 2

```
initial begin
      // input neurons
      i0 = -1;
      i1 = 3;
      i2 = 4;
      i3 = -5;

      // input weights
      i0w0 = 1;
      i1w0 = 1;
      i2w0 = 1;
      i3w0 = -1;
      i0w1 = 2;
      i1w1 = 2;
      i2w1 = 2;
      i3w1 = 2;
      i0w2 = -3;
      i1w2 = 3;
      i2w2 = 3;
      i3w2 = 3;

      // hidden neurons biases
      h0b = 1;
      h1b = 1;
      h2b = 1;
end
```

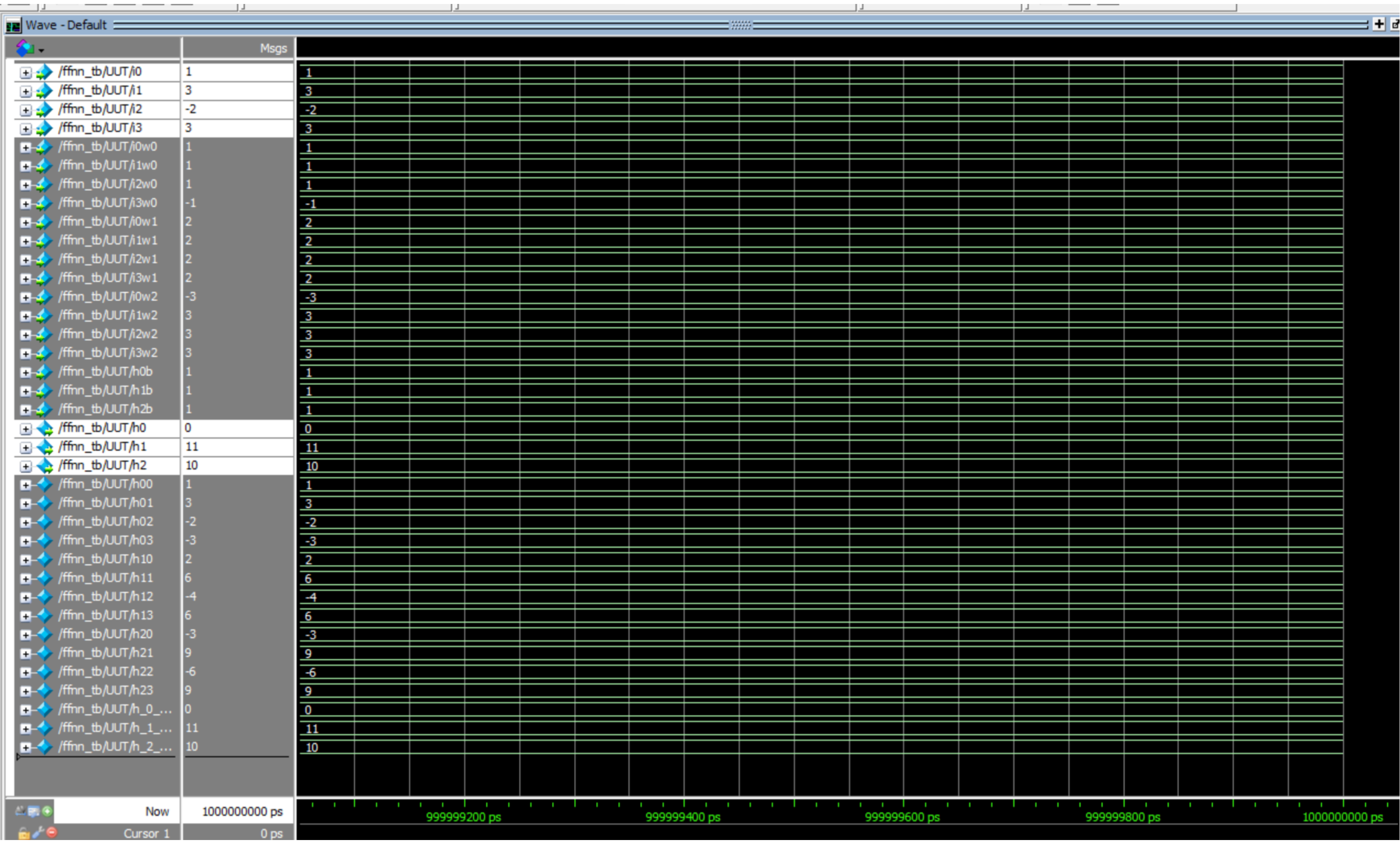
Hidden neurons h0, h1 and h2



Simulation 3

```
initial begin
	// input neurons
	i0 = 1;
	i1 = 3;
	i2 = -2;
	i3 = 3;

	// input weights
	i0w0 = 1;
	i1w0 = 1;
	i2w0 = 1;
	i3w0 = -1;
	i0w1 = 2;
	i1w1 = 2;
	i2w1 = 2;
	i3w1 = 2;
	i0w2 = -3;
	i1w2 = 3;
	i2w2 = 3;
	i3w2 = 3;

	// hidden neurons biases
	h0b = 1;
	h1b = 1;
	h2b = 1;
end
```

Hidden neurons h0, h1 and h2



II) HIDDEN INPUT LAYER

i) input [7:0] h0,
ii) input [7:0] h1,
iii) input [7:0] h2,

And weights of hidden input layers are:

For h0, 2 weights:

i) input [7:0] h0w0,
ii) input [7:0] h0w1,

For h1, 2 weights:

i) input [7:0] h1w0,
ii) input [7:0] h1w1,

For h2, 2 weights:

i) input [7:0] h2w0,
ii) 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:

i) wire [7:0] out_00;
ii) wire [7:0] out_01;
iii) wire [7:0] out_02;

They are computed as,

i) assign out_00 = h0 * h0w0;
ii) assign out_01 = h1 * h1w0;
iii) assign out_02 = h2 * h2w0;

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

i) wire [7:0] out_0_sum;
ii) 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:

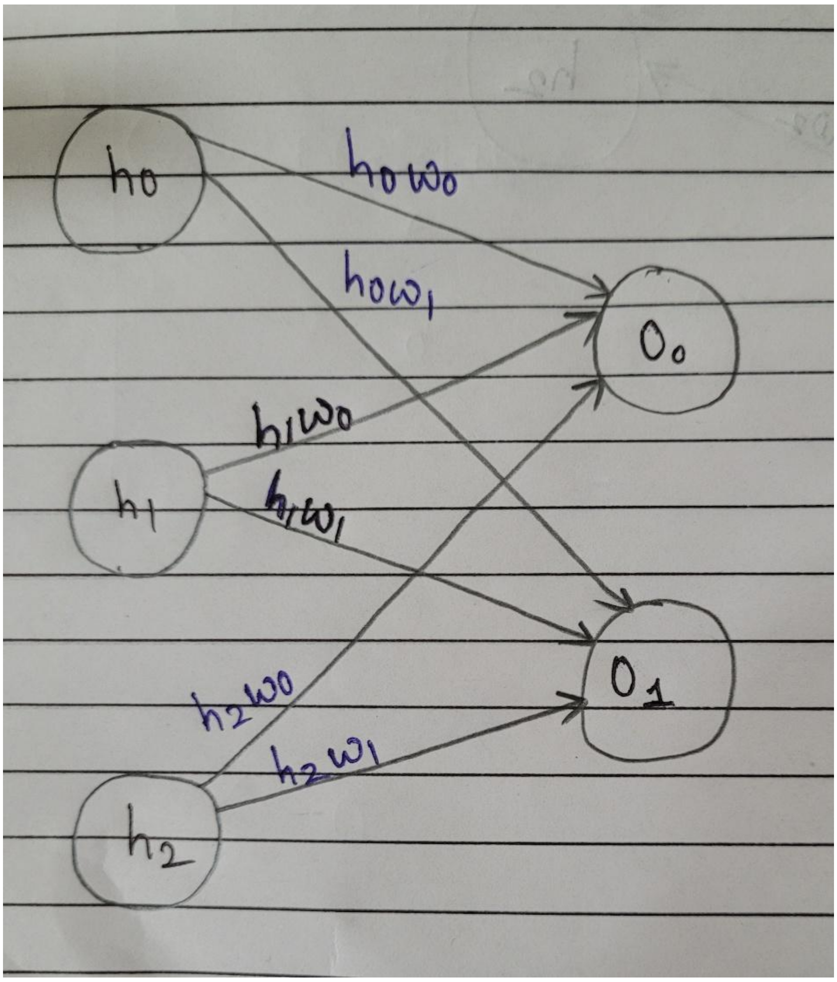
- We are discarding the lower 8-bits so as to give the precision of 8-bits (which is the size of hidden neuron as well)

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:



$$\left.\begin{array}{l}(h_0 * h_0w_0)\\(h_1 * h_1w_0)\\(h_2 * h_2w_0)\end{array}\right\}\Sigma \rightarrow out_0$$

+ out_0b

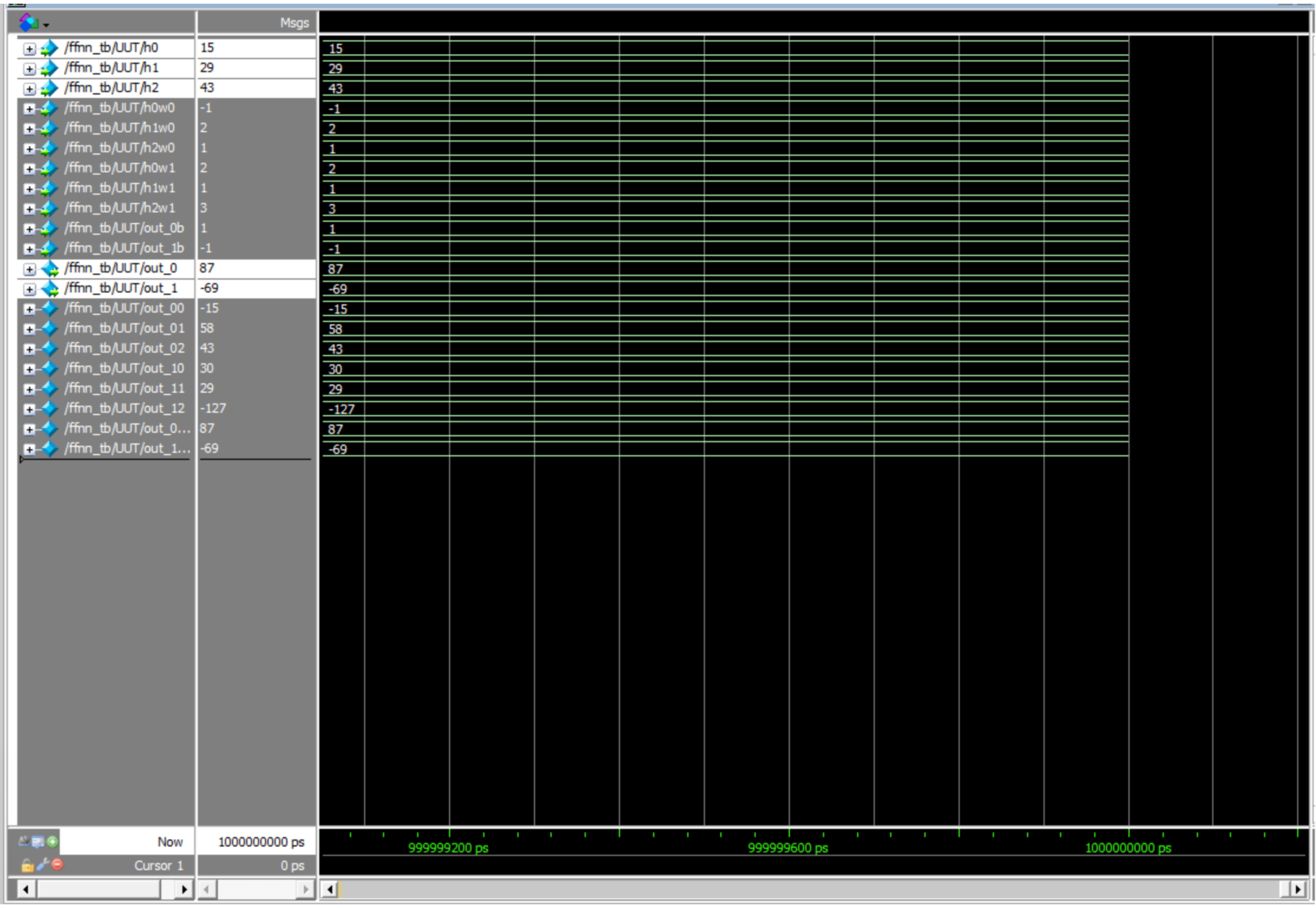
$$\left.\begin{array}{l}(h_0 * h_0w_1)\\(h_1 * h_1w_1)\\(h_2 * h_2w_1)\end{array}\right\}\Sigma \rightarrow out_1$$

+ out_1b

Simulation 1:

```
initial begin
      // hidden input neurons
      h0 = 15;
      h1 = 29;
      h2 = 43;

      // weights
      h0w0 = -1;
      h1w0 = 2;
      h2w0 = 1;

      h0w1 = 2;
      h1w1 = 1;
      h2w1 = 3;

      // output neurons biases
      out_0b = 1;
      out_1b = -1;

end
```

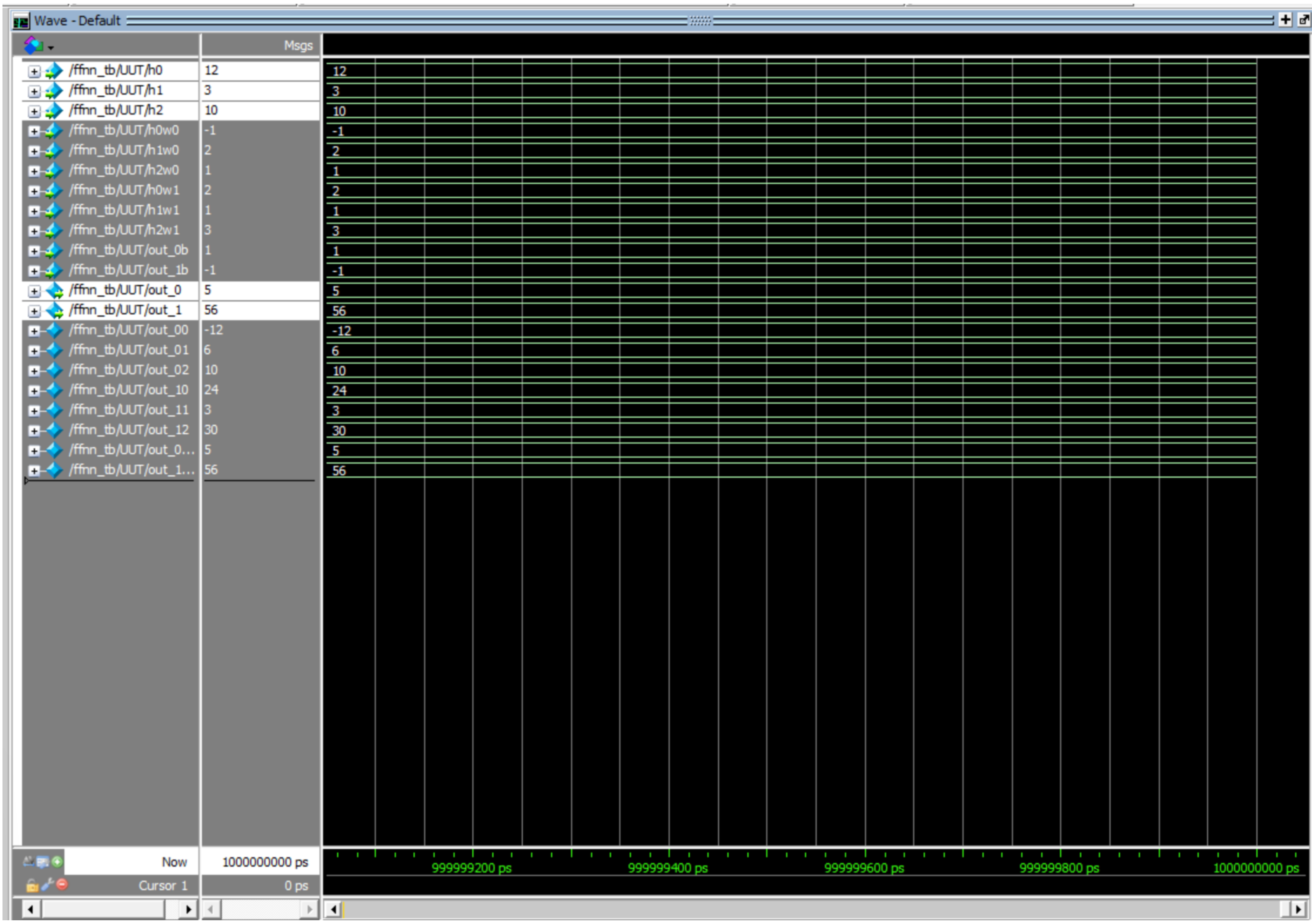
Output neurons out_0 and out_1



Simulation 2:

```
initial begin
      // hidden input neurons
      h0 = 12;
      h1 = 3;
      h2 = 10;

      // weights
      h0w0 = -1;
      h1w0 = 2;
      h2w0 = 1;

      h0w1 = 2;
      h1w1 = 1;
      h2w1 = 3;

      // output neurons biases
      out_0b = 1;
      out_1b = -1;

end
```

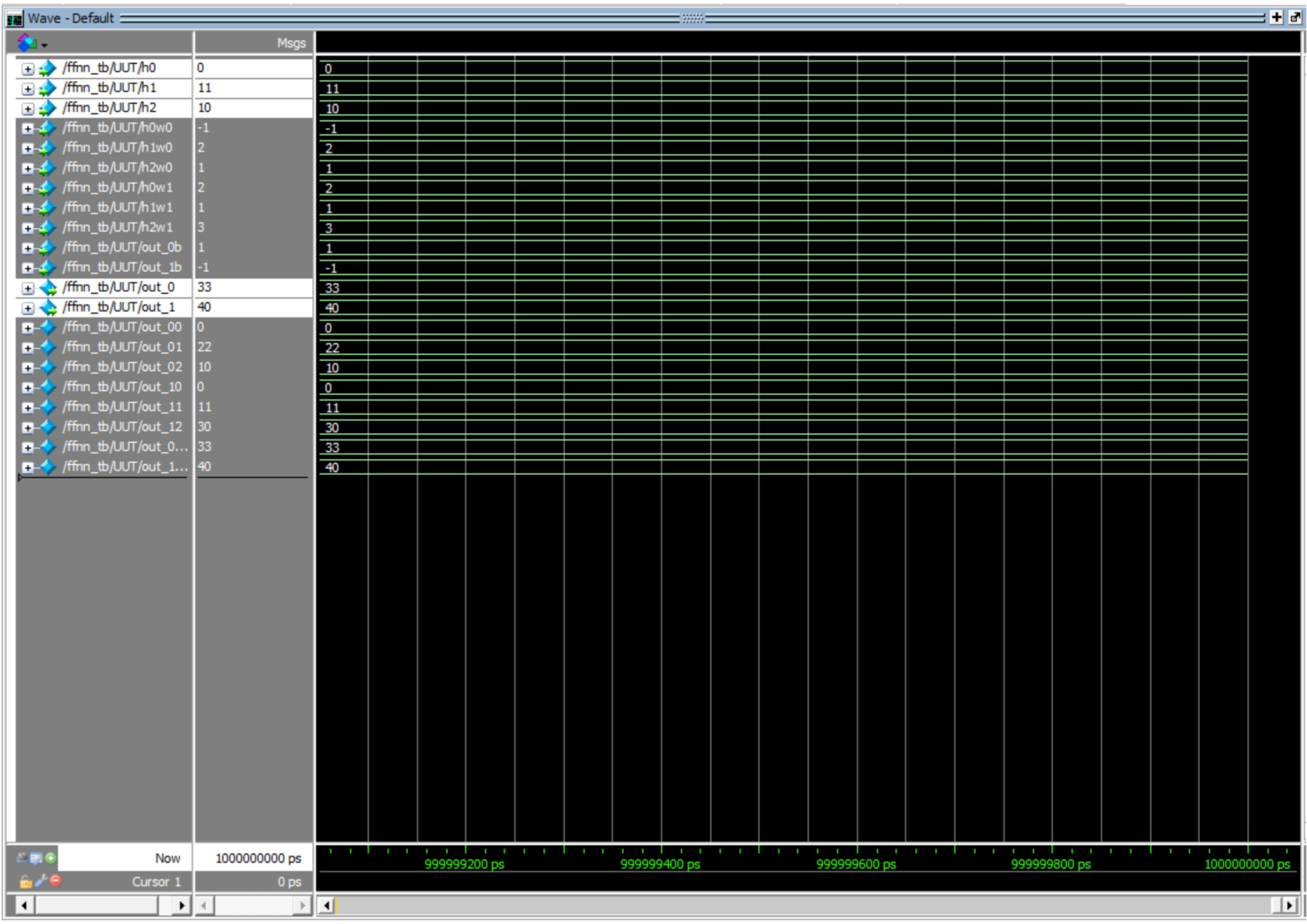
Output neurons out_0 and out_1



Simulation 3:

```
initial begin
    // hidden input neurons
    h0 = 0;
    h1 = 11;
    h2 = 10;

    // weights
    h0w0 = -1;
    h1w0 = 2;
    h2w0 = 1;

    h0w1 = 2;
    h1w1 = 1;
    h2w1 = 3;

    // output neurons biases
    out_0b = 1;
    out_1b = -1;

end
```

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).

```
always @(posedge clk) begin
      h00 <= i0 * i0w0;
      h01 <= i1 * i1w0;
      h02 <= i2 * i2w0;
      h03 <= i3 * i3w0;

      h10 <= i0 * i0w1;
      h11 <= i1 * i1w1;
      h12 <= i2 * i2w1;
      h13 <= i3 * i3w1;

      h20 <= i0 * i0w2;
      h21 <= i1 * i1w2;
      h22 <= i2 * i2w2;
      h23 <= i3 * i3w2;
end
```

And then large additions combinatorically:

```
assign h_0_sum = h00 + h01 + h02 + h03 + h0b;
assign h_1_sum = h10 + h11 + h12 + h13 + h1b;
assign h_2_sum = h20 + h21 + h22 + h23 + h2b;
```

And delay for next sequential logic which required for combinatorial logic

```
// 20ns delay
#20
always @(posedge clk) begin
     h0 <= relu(h_0_sum);
     h1 <= relu(h_1_sum);
     h2 <= relu(h_2_sum);
end
```

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.

```
module input_hidden_layer(
    // input layer
    input [7:0] i0,
    input [7:0] i1,
    input [7:0] i2,
    input [7:0] i3,

    // h0 weights
    input [7:0] i0w0,
    input [7:0] i1w0,
    input [7:0] i2w0,
    input [7:0] i3w0,

    // h1 weights
    input [7:0] i0w1,
    input [7:0] i1w1,
    input [7:0] i2w1,
    input [7:0] i3w1,

    // h2 weights
    input [7:0] i0w2,
    input [7:0] i1w2,
    input [7:0] i2w2,
    input [7:0] i3w2,

    // h0 bias
    input [7:0] h0b,
    // h1 bias
    input [7:0] h1b,
    // h2 bias
    input [7:0] h2b,

    // hidden layer
    output reg [7:0] h0,
    output reg [7:0] h1,
    output reg [7:0] h2
);
```

```
module hidden_output_layer(
    // hidden input layer
    input [7:0] h0,
    input [7:0] h1,
    input [7:0] h2,

    // out_0 weights
    input [7:0] h0w0,
    input [7:0] h1w0,
    input [7:0] h2w0,

    // out_1 weights
    input [7:0] h0w1,
    input [7:0] h1w1,
    input [7:0] h2w1,

    // out_0 bias
    input [7:0] out_0b,
    // out_1 bias
    input [7:0] out_1b,

    // output layer
    output reg [7:0] out_0,
    output reg [7:0] out_1
);
```

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

i) input to hidden neuron
ii) 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.