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

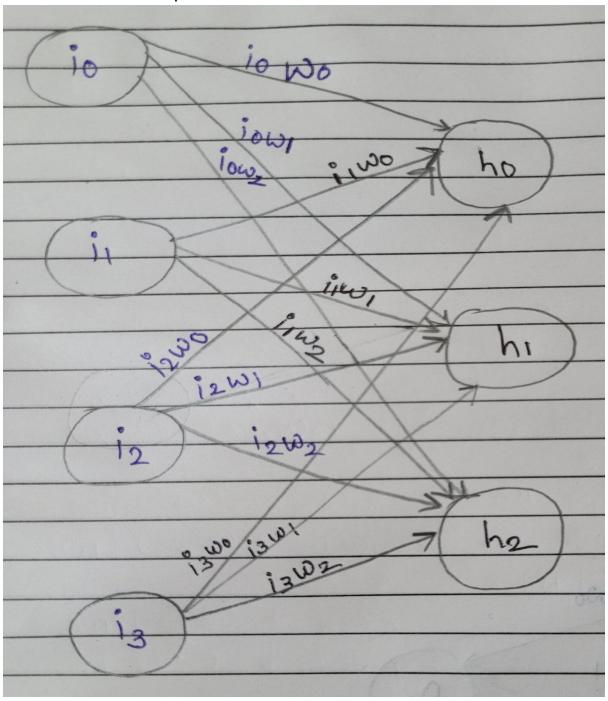
Then we will pass the result of SOP to ReLU function of hidden neuron.

• 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:



+ h0b (h2 bias)

+ h1b (h2 bias)

$$(i_0 * i_0 \omega_2)$$
 $\{ \ge \rightarrow h_2$
 $(i_1 * i_1 \omega_2)$
 $(i_2 * i_2 \omega_2)$
 $(i_3 * i_3 \omega_2)$

+ 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:

≨ 1 +	Msgs							
±	2	2						
⊕ ∳ /ffnn_tb/UUT/i1	3	3						
→ /ffnn_tb/UUT/i2	4	4						
⊕ ♣ /ffnn_tb/UUT/i3	5	5						
#-4 /ffnn_tb/UUT/i0w0	1	1						
#-4 /ffnn_tb/UUT/i1w0	1	1						
#-4 /ffnn_tb/UUT/i2w0	1	1						
+-4 /ffnn_tb/UUT/i3w0	1	1						
#-4 /ffnn_tb/UUT/i0w1	2	2						
+ // /ffnn_tb/UUT/i1w1	2	2						
+ /ffnn_tb/UUT/i2w1	2	2						
+-4 /ffnn_tb/UUT/i3w1	2	2						
#-4 /ffnn_tb/UUT/i0w2	3	3						
+ // /ffnn_tb/UUT/i1w2	3	3						
+ // /ffnn_tb/UUT/i2w2	3	3						
+ // /ffnn_tb/UUT/i3w2	3	3						
#-4 /ffnn_tb/UUT/h0b	1							
#_4 /ffnn_tb/UUT/h1b	1							
#_4 /ffnn_tb/UUT/h2b	1	1						
+ /ffnn_tb/UUT/h0	15	15						
± ♦ /ffnn_tb/UUT/h1	29	29						
→ /ffnn_tb/UUT/h2	43	43						
→ /ffnn_tb/UUT/h00	2	2						
- → /ffnn_tb/UUT/h01	3	3						
- → /ffnn_tb/UUT/h02	4	4						
+- /ffnn_tb/UUT/h03	5	5						
+- /ffnn_tb/UUT/h10	4	4						
+- /ffnn_tb/UUT/h11	6	6						
+- /ffnn_tb/UUT/h12	8	8						
+- /ffnn_tb/UUT/h13	10	10						
+- /ffnn_tb/UUT/h20	6	6						
+- /ffnn_tb/UUT/h21	9	9						
+- /ffnn_tb/UUT/h22	12	12						
+- /ffnn_tb/UUT/h23	15	15						
+-4 /ffnn_tb/UUT/h_0		15						
- → /ffnn_tb/UUT/h_1		29						
- → /ffnn_tb/UUT/h_2		43						
△ ■ ONOW	1000000000 ps	1 1 0000			1 1	00500 00	 	1 1
6 ∕ 9 Cursor 1	0 ps	33333	99200 ps		999999	9600 ps		1000000
COLOUR S	o po	A.						

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;
```

Hidden neurons h0, h1 and h2

	-1 3 4 -5 1	-1 3 4 -5								
/ffnn_b/UUT/i1 3 /ffnn_b/UUT/i2 4 /ffnn_b/UUT/i3 - /ffnn_b/UUT/i0w0 1 /ffnn_b/UUT/i1w0 1	1	3 4 -5								
 → /ffnn_tb/UUT/i2 → /ffnn_tb/UUT/i3 → - /ffnn_tb/UUT/i0w0 → - /ffnn_tb/UUT/i1w0 		4 -5								
	5 1 1	-5								
/ffnn_tb/UUT/i0w0 1 /ffnn_tb/UUT/i1w0 1	1 1									
#-4 /ffnn_tb/UUT/i1w0 1	1									
		1								
II		1								
/ffnn_tb/UUT/i3w0 -	1	-1								
/ffnn_tb/UUT/i0w1 2	2	2								
/ffnn_tb/UUT/i1w1 2	2	2								
/ffnn_tb/UUT/i2w1 2	2	2								
→ /ffnn_tb/UUT/i3w1 2	2	2								
	-3	-3								
/ffnn_tb/UUT/i1w2 3	3	3								
→ /ffnn_tb/UUT/i2w2 3	3	3								
→ /ffnn_tb/UUT/i3w2 3	3	3								
→ /ffnn_tb/UUT/h0b 1	1	1								
+ /ffnn_tb/UUT/h1b 1	1	1								
+ /ffnn_tb/UUT/h2b 1	1	1								
→ /ffnn_tb/UUT/h0 1	12	12								
→ /ffnn_tb/UUT/h1 3	3	3								
→ /ffnn_tb/UUT/h2 1	10	10								
+-/ /ffnn_tb/UUT/h00 -	1	-1								
 /ffnn_tb/UUT/h01 3	3	3								
 /ffnn_tb/UUT/h02 4	4	4								
- → /ffnn_tb/UUT/h03 5	5	5								
→ /ffnn_tb/UUT/h10 -:	-2	-2								
 → /ffnn_tb/UUT/h11 6	5	6								
#—→ /ffnn_tb/UUT/h12 8	3	8								
→ /ffnn_tb/UUT/h13 -	-10	-10								
# /ffnn_tb/UUT/h20 3	3	3								
 → /ffnn_tb/UUT/h21 9		9								
	12	12								
	-15	-15								
-	12	12								
- → /ffnn_tb/UUT/h_1 3	3	3								
II - /ffnn_tb/UUT/h_2 1	10	10								
Now	1000000000 ps		999999	1 1	1 1	999999	1 1	1 1	1000000	0000 p

Simulation 3

```
initial begin
    // input neurons
    i0 = 1;
    i1 = 3;
    i2 = -2;
    i3 = 3;
    // input weights
    iowo = 1;
    i1w0 = 1;
    i2w0 = 1;
    i3w0 = -1;
    iOw1 = 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

]1						
1	Wave - Default :															<u></u> +
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3	≨ 1 +		Msgs													الريس)
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3		JUT/i0 1		1												
Description Prince Princ																
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3	± → /ffnn tb/U	JUT/i2 -														
Prince Martiface																کی ا
1			V	1												كروج
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1			4	1												
Description			4	1												
2			-1	-1				کری								
Thin_BhUT/Swi																
2			2	2			الكري	کریں:		<u> کر کی ب</u>		اکی ا				
See Minn BAUT Part See S			2	2						<u>کریں</u>						
Septembourness Sept			5	2			الكبيرية									كري
Septembourning Sept			-3	-3			اکس	کریں:		<u>کری</u>		اکی ا				كالراز
B.⇒./ fiffin_bbuT/hav2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3			8	3			الكسية					الكيارا				
3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1			8	3			الكسية			الرابع الأربية الأربية الأربية الأربية الأرب						
1	II - 4 /ffnn_tb/U	UUT/i3w2 3	3	3						4						
1			4	1						<u>کا کی ا</u>						
Time Dout			1	1			الكسية			الزيرية الأراكية		الكيارا				
(a) → /ffnn_b/JUT/h1 11 11 11 11 11 11 11 11 11 11 11 11 1	#-4 /ffnn_tb/U	UUT/h2b 1	1	1			الكسي	كري		خري		اکی ا				
(a)	±-< /ffnn_tb/U	UUT/h0 0				كالس						اکی ا				
(Fifth _b/UT/h0 1 1 1 1 1 1 1 1 1	± 🍁 /ffnn_tb/U	UUT/h1 1					الكليات			الرابع: الأربع:		الكيارا				
Thin Dhut This Thin Dhut This Thin Thin This Thin																
C → /ffin_b/UT/h01	II - /ffnn_tb/U	UUT/h00 1	1	1		کا کے	الكليلة					الكيار)				
C	#	UUT/h01 3	3	3				کالک		خنی		الكيل ا	كري	الكي		
C → /ffm_b/UUT/h10 C → /ffm_b/UUT/h12 C → /ffm_b/UUT/h20 C → /ffm_b/UUT/h21 C → /ffm_b/UUT/h22 C → /ffm_b/UUT/h23 C → /ffm_b/U	#	UUT/h02 -			کاری	<u>کاک</u>	الكوارات	كالك		خالی:		اکی ا		الكي		
C → /fffn_b\U\U\f\n11	# /ffnn_tb/U	UUT/h03 -			كسي	كالك		كالك		ختی		الكيل ا	کی	الكي		
C→ /ffin_b/UUT/h12 C→ /ffin_b/UUT/h22 C→ /ffin_b/UUT/h23 D 100000000 ps			2	2		کاک		کاک								
□ / /ffin_b/UUT/h12 □ / /ffin_b/UUT/h23 □ / /ffin_b/UUT/h22 □ / /ffin_b/UUT/h23 □ /	# /ffnn_tb/U	UUT/h11 6	6	6	کس	كالك	الكوارات	کاک	کی ا	ختی		اکی ا		الكي		
### ##################################	# /ffnn_tb/U	UUT/h12 -	4	-4	كسي	كالك	الكليلة	كالك	کی ا	ختی		اکی ا		الكي		
-3 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	# /ffnn_tb/U	UUT/h13 6			كسي	كالك		كالك		ختی		الكيار)	کی	الكي		
##	# /ffnn_tb/U	UUT/h20 -	-3	-3		کاکی"	الكوارا	كالك				اکی ا		الكي		
	# /ffnn_tb/U	UUT/h21 9	9	9		کاکی"	الكليلة	كالك		الرابع: الاربع:		اکی ا		الكي		
### ##################################	# /ffnn_tb/U	UUT/h22 -	-6	-6	کست	كالك		کاک		ختی		الكيار)	کاک	الكي		
### ##################################					کست	كالك		کالک		ختی		الكيل ا	کی	الكي		
### ##################################			5	0				کالک								
Now 100000000 ps 100000000 ps 100000000 ps 100000000 ps 100000000 ps 1000000000 ps 10000000000	I IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	/UUT/h_1 1	11	11		کاکی"	الكوارا	كالك		الرابع الأربي		اکی ا		الكي		
Now 100000000 ps 99999200 ps 99999400 ps 99999600 ps 99999800 ps 1000000000 ps							الكسي	کریں:		<u> کر کی ب</u>		اکی ا				
Now 1000000000 ps 99999200 ps 99999400 ps 99999600 ps 99999800 ps 1000000000 p							الكليلا		الكبيرية	كالبيها		الكليلا				
Now 1000000000 ps 99999200 ps 99999400 ps 99999600 ps 99999800 ps 1000000000 p																
999999200 ps 99999900 ps 100000000 ps	AL PROPERTY.	Now	1000000000						THE		THE RESERVE		Townson or the last			
Cursor 1 Ups	6.40				999999200 ps		999999	9400 ps		999999600	U ps		999999800 ps		1000	000000 ps
	<u>□</u>	Cursor 1	U ps													

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:

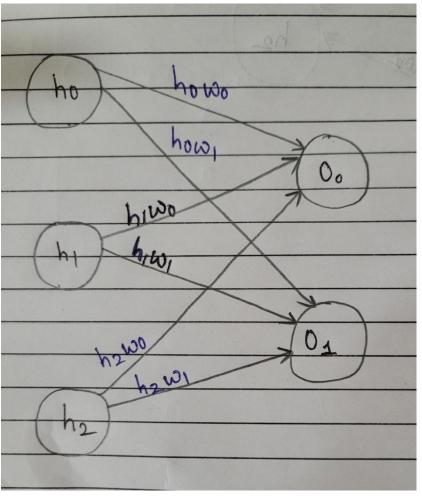
• We are discarding the lower 8-bits so as to give the precision of 8-bits (which is the size of hidden neuron as well)

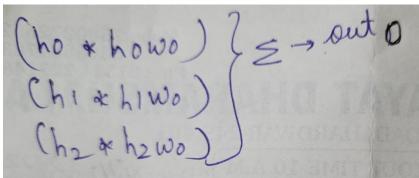
assign out_0_sum = out_00 + out_01 + out_02 + out_0b;

• Here 'out 0b' is the bias of out 0 output neuron

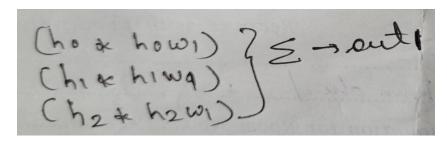
Same process we have also followed for out_1 output neuron.

It can be summed up as:





+ out_0b



+ out_1b

Simulation 1:

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

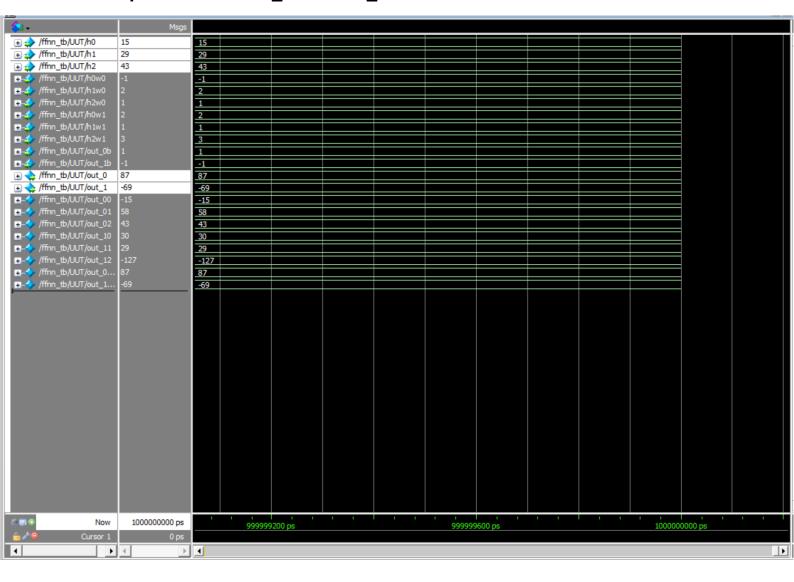
   // weights
   h0w0 = -1;
   h1w0 = 2;
   h2w0 = 1;

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

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

end

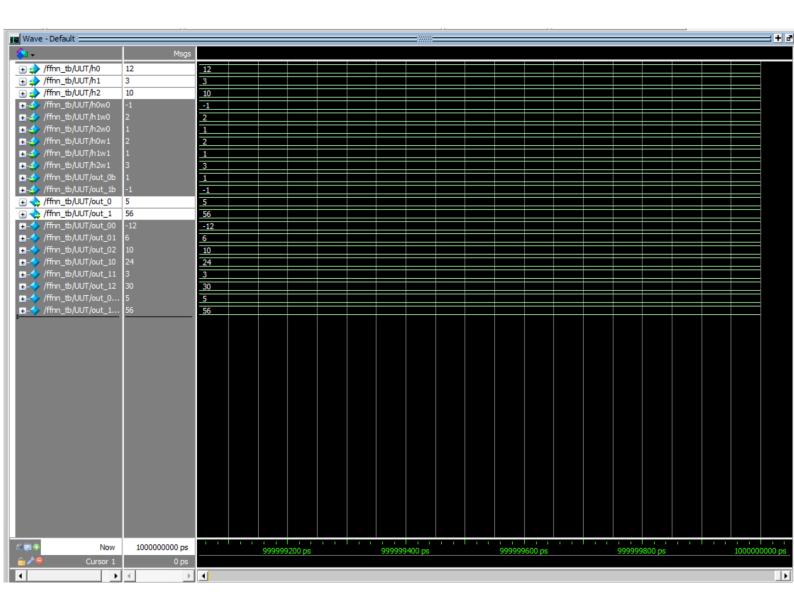
Output neurons out_0 and out_1



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
```

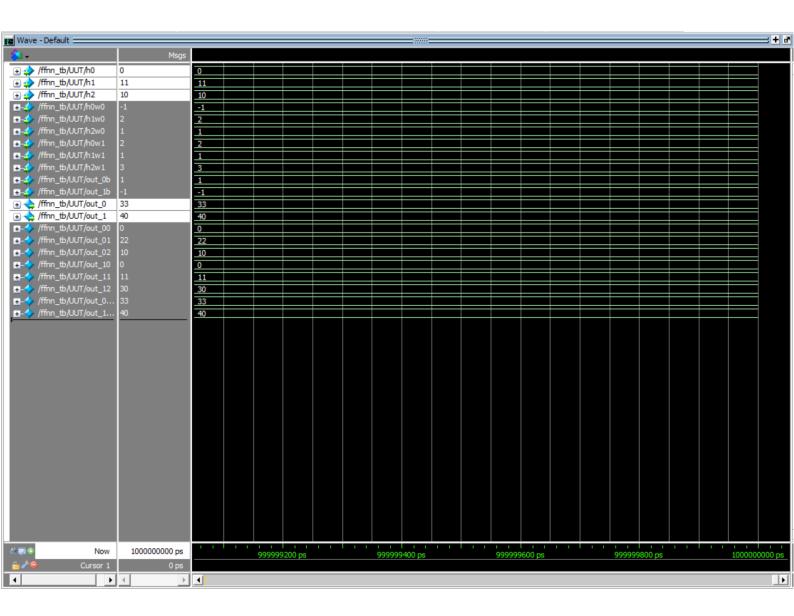
Output neurons out_0 and out_1



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).

```
lalways @ (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</pre>
```

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</pre>
```

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,
                         module hidden output layer (
                              // hidden input layer
   input [7:0] i2,
   input [7:0] i3,
                              input [7:0] h0,
                              input [7:0] h1,
   // h0 weights
                              input [7:0] h2,
   input [7:0] i0w0,
   input [7:0] i1w0,
   input [7:0] i2w0,
                              // out 0 weights
   input [7:0] i3w0,
                              input [7:0] h0w0,
                              input [7:0] h1w0,
   // h1 weights
   input [7:0] i0w1,
                              input [7:0] h2w0,
   input [7:0] i1w1,
   input [7:0] i2w1,
                              // out 1 weights
   input [7:0] i3w1,
                              input [7:0] h0w1,
   // h2 weights
                              input [7:0] h1w1,
   input [7:0] i0w2,
                              input [7:0] h2w1,
   input [7:0] i1w2,
   input [7:0] i2w2,
   input [7:0] i3w2,
                              // out 0 bias
                              input [7:0] out 0b,
   // h0 bias
                              // out 1 bias
   input [7:0] h0b,
   // h1 bias
                              input [7:0] out 1b,
   input [7:0] h1b,
   // h2 bias
                              // output layer
   input [7:0] h2b,
                              output reg [7:0] out 0,
   // hidden layer
                              output reg [7:0] out 1
   output reg [7:0] h0,
                         -);
   output reg [7:0] h1,
   output reg [7:0] h2
);
```

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.