Challenge #8

1. Implement a multi-layer feed-forward perceptron network. The network should have two input neurons, two hidden neurons, and one output neuron.

// Top-level MLP Module with 2-input, 2-hidden, 1-output neuron

module perceptron\_net (

input clk,

input reset,

input signed [7:0] in1,

input signed [7:0] in2,

output reg signed [7:0] out

);

wire signed [7:0] input\_vec [1:0];

assign input\_vec[0] = in1;

assign input\_vec[1] = in2;

wire signed [7:0] hidden\_out [1:0];

wire signed [7:0] net\_out;

// Instantiate Hidden Layer Neurons

neuron #(.N(2)) hidden0 (

.clk(clk),

.reset(reset),

.inputs(input\_vec),

.weights('{8'sd1, 8'sd2}),

.bias(8'sd1),

.out(hidden\_out[0])

);

neuron #(.N(2)) hidden1 (

.clk(clk),

.reset(reset),

.inputs(input\_vec),

.weights('{8'sd3, 8'sd4}),

.bias(8'sd0),

.out(hidden\_out[1])

);

// Output Neuron

neuron #(.N(2)) output\_neuron (

.clk(clk),

.reset(reset),

.inputs(hidden\_out),

.weights('{8'sd2, 8'sd1}),

.bias(8'sd1),

.out(net\_out)

);

always @(posedge clk or posedge reset) begin

if (reset) begin

out <= 0;

end else begin

out <= net\_out;

end

end

endmodule

// Parameterized Neuron Module with Clocking

module neuron #(parameter N = 2)(

input clk,

input reset,

input signed [7:0] inputs [N-1:0],

input signed [7:0] weights [N-1:0],

input signed [7:0] bias,

output reg signed [7:0] out

);

integer i;

reg signed [15:0] acc;

always @(posedge clk or posedge reset) begin

if (reset) begin

acc <= 0;

out <= 0;

end else begin

acc = 0;

for (i = 0; i < N; i = i + 1) begin

acc = acc + inputs[i] \* weights[i];

end

acc = acc + bias;

// ReLU Activation

if (acc < 0)

out <= 0;

else

out <= acc[7:0];

end

end

endmodule

// Testbench

module tb\_perceptron\_net();

reg clk;

reg reset;

reg signed [7:0] in1, in2;

wire signed [7:0] out;

perceptron\_net uut (

.clk(clk),

.reset(reset),

.in1(in1),

.in2(in2),

.out(out)

);

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

reset = 1;

in1 = 0; in2 = 0;

#10 reset = 0;

in1 = 8'sd1; in2 = 8'sd2; #10;

in1 = 8'sd2; in2 = 8'sd3; #10;

in1 = -8'sd1; in2 = 8'sd1; #10;

in1 = 8'sd5; in2 = -8'sd2; #10;

$stop;

end

endmodule

The complete and synthesizable implementation of a multi-layer perceptron network in Verilog consists of:

* 2 input neurons
* 2 hidden neurons
* 1 output neuron
* ReLU activation
* Fully pipelined with clk and reset

1. Implement the backpropagation algorithm to train your network to solve the XOR logical function.

// Top-level MLP Module with Trainable Parameters for XOR

module perceptron\_net (

input clk,

input reset,

input train\_enable,

input signed [7:0] in1,

input signed [7:0] in2,

input signed [7:0] target,

output reg signed [7:0] out

);

wire signed [7:0] input\_vec [1:0];

assign input\_vec[0] = in1;

assign input\_vec[1] = in2;

wire signed [7:0] hidden\_out [1:0];

wire signed [7:0] net\_out;

wire signed [7:0] w\_hidden [1:0][1:0];

wire signed [7:0] b\_hidden [1:0];

wire signed [7:0] w\_output [1:0];

wire signed [7:0] b\_output;

wire update\_done;

training\_controller trainer (

.clk(clk),

.reset(reset),

.train\_enable(train\_enable),

.input\_vec(input\_vec),

.target(target),

.hidden\_out(hidden\_out),

.net\_out(net\_out),

.w\_hidden(w\_hidden),

.b\_hidden(b\_hidden),

.w\_output(w\_output),

.b\_output(b\_output),

.update\_done(update\_done)

);

// Hidden Neurons

trainable\_neuron #(.N(2)) hidden0 (

.clk(clk), .reset(reset),

.inputs(input\_vec),

.weights(w\_hidden[0]), .bias(b\_hidden[0]),

.out(hidden\_out[0])

);

trainable\_neuron #(.N(2)) hidden1 (

.clk(clk), .reset(reset),

.inputs(input\_vec),

.weights(w\_hidden[1]), .bias(b\_hidden[1]),

.out(hidden\_out[1])

);

// Output Neuron

trainable\_neuron #(.N(2)) output\_neuron (

.clk(clk), .reset(reset),

.inputs(hidden\_out),

.weights(w\_output), .bias(b\_output),

.out(net\_out)

);

always @(posedge clk or posedge reset) begin

if (reset) out <= 0;

else out <= net\_out;

end

endmodule

// Trainable Neuron Module

module trainable\_neuron #(parameter N = 2)(

input clk,

input reset,

input signed [7:0] inputs [N-1:0],

input signed [7:0] weights [N-1:0],

input signed [7:0] bias,

output reg signed [7:0] out

);

integer i;

reg signed [15:0] acc;

always @(posedge clk or posedge reset) begin

if (reset) begin

acc <= 0;

out <= 0;

end else begin

acc = 0;

for (i = 0; i < N; i = i + 1) begin

acc = acc + inputs[i] \* weights[i];

end

acc = acc + bias;

out <= (acc < 0) ? 0 : acc[7:0];

end

end

endmodule

// Backpropagation-based Training Controller for XOR

module training\_controller (

input clk,

input reset,

input train\_enable,

input signed [7:0] input\_vec [1:0],

input signed [7:0] target,

input signed [7:0] hidden\_out [1:0],

input signed [7:0] net\_out,

output reg signed [7:0] w\_hidden [1:0][1:0],

output reg signed [7:0] b\_hidden [1:0],

output reg signed [7:0] w\_output [1:0],

output reg signed [7:0] b\_output,

output reg update\_done

);

parameter signed [7:0] LEARNING\_RATE = 8'sd1; // Basic learning rate

reg signed [7:0] error;

reg signed [7:0] d\_out;

reg signed [7:0] d\_hidden [1:0];

integer i, j;

always @(posedge clk or posedge reset) begin

if (reset) begin

w\_hidden[0][0] <= 8'sd1; w\_hidden[0][1] <= 8'sd1;

w\_hidden[1][0] <= -8'sd1; w\_hidden[1][1] <= -8'sd1;

b\_hidden[0] <= 8'sd0; b\_hidden[1] <= 8'sd0;

w\_output[0] <= 8'sd1; w\_output[1] <= 8'sd1;

b\_output <= 8'sd0;

update\_done <= 0;

end else if (train\_enable) begin

// Forward error

error = target - net\_out;

d\_out = (net\_out > 0) ? error : 8'sd0; // ReLU derivative

// Backprop into hidden layer

for (i = 0; i < 2; i = i + 1) begin

d\_hidden[i] = (hidden\_out[i] > 0) ? d\_out \* w\_output[i] : 8'sd0;

end

// Update output weights & bias

for (i = 0; i < 2; i = i + 1) begin

w\_output[i] <= w\_output[i] + ((LEARNING\_RATE \* d\_out \* hidden\_out[i]) >>> 4);

end

b\_output <= b\_output + ((LEARNING\_RATE \* d\_out) >>> 4);

// Update hidden weights & biases

for (i = 0; i < 2; i = i + 1) begin

for (j = 0; j < 2; j = j + 1) begin

w\_hidden[i][j] <= w\_hidden[i][j] + ((LEARNING\_RATE \* d\_hidden[i] \* input\_vec[j]) >>> 4);

end

b\_hidden[i] <= b\_hidden[i] + ((LEARNING\_RATE \* d\_hidden[i]) >>> 4);

end

update\_done <= 1;

end else begin

update\_done <= 0;

end

end

endmodule