Codefest Challenges -6

Challenges -18:

Implement a binary LIF neuron

Code:

```
module lif_neuron #(
  parameter WIDTH = 8,         // Bit width of the potential
  parameter LEAK = 8'd200,      // λ scaled: e.g., 0.8 * 256 = 204
  parameter SCALE = 8'd256,     // Fixed-point scale factor
  parameter THRESHOLD = 8'd128,  // Threshold to spike
  parameter RESET_VAL = 8'd0     // Reset value for potential
)(
  input logic clk,
  input logic rst,
  input logic I,              // Binary input
  output logic S             // Output spike
);

  logic [WIDTH-1:0] P;        // Membrane potential
  logic [WIDTH-1:0] P_next;

  always_ff @(posedge clk or posedge rst) begin
    if (rst) begin
      P <= RESET_VAL;
      S <= 0;
    end else begin
```

```
      if (S)
        P <= RESET_VAL;
      else
        P <= P_next;

      S <= (P_next >= THRESHOLD) ? 1 : 0;
    end
  end

  // Compute P_next = λ*P + I
  always_comb begin
    P_next = ((P * LEAK) >> 8) + I;
  end

endmodule
```

TestBench:

```
module tb_lif_neuron;

  logic clk, rst, I;
  logic S;

  // Instantiate neuron
  lif_neuron #(
    .WIDTH(8),
```

```
    .LEAK(8'd200),        // ~λ = 0.78
    .SCALE(8'd256),
    .THRESHOLD(8'd128),
    .RESET_VAL(8'd0)
) neuron (
    .clk(clk),
    .rst(rst),
    .I(I),
    .S(S)
);

// Clock generation
initial clk = 0;
always #5 clk = ~clk;

// Stimulus
initial begin
    $display("Time\tI\tS");
    $monitor("%4t\t%b\t%b", $time, I, S);

    // Reset
    rst = 1; I = 0; #10;
    rst = 0;

    // --- 1. Constant input below threshold ---
    $display("\n--- Constant input below threshold ---");
```

```
    repeat (10) begin I = 1; #10; end

    // --- 2. Input accumulates until reaching threshold ---
    $display("\n--- Accumulating input ---");
    rst = 1; #10; rst = 0;
    repeat (20) begin I = 1; #10; end

    // --- 3. Leakage with no input ---
    $display("\n--- Leakage with no input ---");
    rst = 1; #10; rst = 0;
    I = 1; #30; I = 0;
    repeat (10) #10;

    // --- 4. Strong input causing immediate spike ---
    $display("\n--- Strong input causing spike ---");
    rst = 1; #10; rst = 0;
    I = 1;
    force neuron.P = 8'd150;  // Manually set potential
    #10;
    release neuron.P;

    #50 $finish;
  end

endmodule
```

Challenge -19:

Introduction:

A resistive crossbar has:

- **4 word lines (WL)** = input voltages
- **4 bit lines (BL)** = output currents
- 16 programmable resistors (1 per cross-point)

Each output current Ij=∑iViRijI_j = \sum_i \frac{V_i}{R_{ij}}Ij=∑iRijVi

Matrix-vector multiplication I⃗=G·V⃗\vec{I} = G \cdot \vec{V}I=G·V, where G=1/RG = 1/RG=1/R

Code:

```
* 4x4 Resistive Crossbar

* Each R[i][j] connects WL[i] to BL[j]


* Input voltage sources

V1 WL1 0 DC 1.0

V2 WL2 0 DC 0.5

V3 WL3 0 DC 0.0

V4 WL4 0 DC 1.0


* Output bit lines connected to ground through ammeters

* We measure current through these resistors (acts as sinks)

RBL1 BL1 0 1MEG

RBL2 BL2 0 1MEG

RBL3 BL3 0 1MEG
```

RBL4 BL4 0 1MEG

* Crossbar resistive connections

* Format: R<WL><BL> <WL> <BL> <resistance>

R11 WL1 BL1 1k

R12 WL1 BL2 2k

R13 WL1 BL3 3k

R14 WL1 BL4 4k

R21 WL2 BL1 1k

R22 WL2 BL2 2k

R23 WL2 BL3 3k

R24 WL2 BL4 4k

R31 WL3 BL1 1k

R32 WL3 BL2 2k

R33 WL3 BL3 3k

R34 WL3 BL4 4k

R41 WL4 BL1 1k

R42 WL4 BL2 2k

R43 WL4 BL3 3k

R44 WL4 BL4 4k

* Analysis

.OP

.end