Challenge #14

1. Below is the code of the Q-Updating unit and its testbench.

**Design code: -**

module QLearningFSM (

input logic clk,

input logic reset,

output logic [15:0] Q\_new

);

// Parameters

parameter BOARD\_SIZE = 5;

parameter ACTIONS = 4;

// FSM State Definition

typedef enum logic [1:0] {

INIT = 2'd0,

WAIT = 2'd1,

WRITE = 2'd2

} state\_t;

// Registers and signals

logic [4:0] s\_i, s\_j, s\_prime\_i, s\_prime\_j;

logic [1:0] a;

logic [15:0] q\_sa, q\_max\_sp, reward;

logic [15:0] q\_sa\_new;

logic [15:0] Q\_table [0:BOARD\_SIZE-1][0:BOARD\_SIZE-1][0:ACTIONS-1];

state\_t state, next\_state;

// Intermediate signals for QUpdateUnit logic

logic [31:0] discounted\_q;

logic [15:0] target, q1, q2;

// Initialization logic

always\_ff @(posedge clk or posedge reset) begin

if (reset) begin

state <= INIT;

// Initialize all Q-table entries to zero

for (int i = 0; i < BOARD\_SIZE; i++)

for (int j = 0; j < BOARD\_SIZE; j++)

for (int k = 0; k < ACTIONS; k++)

Q\_table[i][j][k] <= 16'd0;

// Optional: set specific entries explicitly

Q\_table[2][2][1] <= 16'd0;

Q\_table[3][2][0] <= 16'd0;

Q\_table[3][2][1] <= 16'd0;

Q\_table[3][2][2] <= 16'd0;

Q\_table[3][2][3] <= 16'd0;

end else begin

state <= next\_state;

if (state == WRITE)

Q\_table[s\_i][s\_j][a] <= q\_sa\_new;

end

end

// FSM state transitions

always\_comb begin

next\_state = state;

case (state)

INIT: next\_state = WAIT;

WAIT: next\_state = WRITE;

WRITE: next\_state = INIT; // Loop again for testing

endcase

end

// Q-update data path logic

always\_comb begin

// Define the state/action values explicitly

s\_i = 5'd2;

s\_j = 5'd2;

s\_prime\_i = 5'd3;

s\_prime\_j = 5'd2;

a = 2'd1;

// Get Q(s,a) and max Q(s',a)

q\_sa = Q\_table[s\_i][s\_j][a];

q\_max\_sp = Q\_table[s\_prime\_i][s\_prime\_j][0];

for (int k = 1; k < ACTIONS; k++) begin

if (Q\_table[s\_prime\_i][s\_prime\_j][k] > q\_max\_sp)

q\_max\_sp = Q\_table[s\_prime\_i][s\_prime\_j][k];

end

// Reward and Q-update calculation

reward = 16'hFFFF; // -1 in two's complement

discounted\_q = (16'd14 \* q\_max\_sp) >> 4; // gamma = 14

target = reward + discounted\_q[15:0];

q1 = ((16 - 16'd8) \* q\_sa) >> 4; // alpha = 8

q2 = (16'd8 \* target) >> 4;

q\_sa\_new = q1 + q2;

end

// Output

assign Q\_new = q\_sa\_new;

endmodule

/////////////////////////////////////////////////////////////////////////////////////////

**Testbench: -**

`timescale 1ns / 1ps

module QLearningFSM\_tb;

// Testbench signals

logic clk;

logic reset;

logic [15:0] Q\_new;

// Instantiate the DUT (Device Under Test)

QLearningFSM dut (

.clk(clk),

.reset(reset),

.Q\_new(Q\_new)

);

// Clock generation: 10ns period

always #5 clk = ~clk;

// Test procedure

initial begin

$display("\n--- Starting QLearningFSM Testbench ---");

$dumpfile("QLearningFSM.vcd");

$dumpvars(0, QLearningFSM\_tb);

// Initialize signals

clk = 0;

reset = 1;

// Apply reset

#10;

reset = 0;

// Wait for a few FSM cycles

repeat (10) begin

#10;

$display("Cycle %0t: Q\_new = %0d", $time, Q\_new);

end

$display("--- Testbench Completed ---\n");

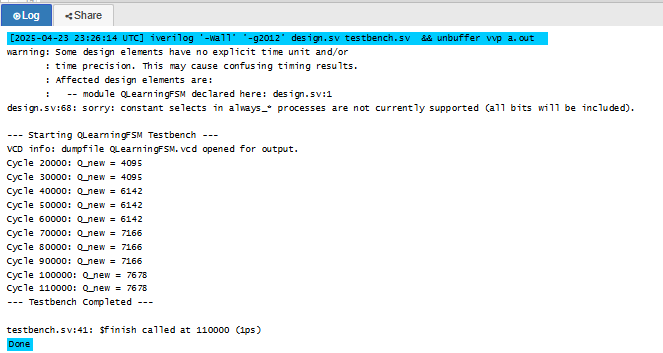
$finish;

end

endmodule

///////////////////////////////////////////////////////////////////////////////////////

**Output: -**

****

From this exercise, I have learned to convert a Python-based Q-learning agent into a synthesizable SystemVerilog module. I also explored FSM-based control logic for reinforcement learning, applied fixed-point arithmetic for Q-value updates, and resolved type mismatches between signed and unsigned logic—key skills for hardware-friendly machine learning implementation. I gained insight into how to debug and adapt HDL code by interpreting simulation errors, handling $display in non-synthesizable blocks, and managing signed arithmetic operations explicitly using $signed.

While working with ChatGPT to convert my Python-based Q-learning agent into a synthesizable SystemVerilog module, I found that it understood my prompts and goals well. I was trying to preserve the functionality of the original agent—like handling state transitions, Q-value updates, and reward logic and ChatGPT provided solid guidance on structuring the FSM, managing the Q-table, and simulating the episodes.

However, there were a couple of technical issues. Initially, it missed the need to explicitly cast signed and unsigned expressions in SystemVerilog, which led to a simulation error. It also placed $display inside an always\_ff block, which is not synthesizable. Overall, the collaboration was productive, and catching those issues helped reinforce my understanding of Verilog synthesis constraints.