**Q1.**

module designModule (

input wire a\_i, // input a

input wire b\_i, // input b

input wire c\_i, // input c

output reg f\_o // output f, now using reg because we will use if-else conditions

);

always @(\*) begin

// 'x' or 'z' cases explicitly for inputs a, b, or c

if (a\_i === 1'bz || b\_i === 1'bz || c\_i === 1'bz) begin

f\_o = 1'bx; // Set output to 'x' if any input is 'z'

end else begin

// Proceed with normal logic for the given function f = ~( (a + b) & c )

f\_o = ~((a\_i | b\_i) & c\_i);

end

end

endmodule

module testbench;

reg a\_r;

reg b\_r;

reg c\_r;

wire f\_w;

// Instantiate the design module

designModule u (

.a\_i (a\_r),

.b\_i (b\_r),

.c\_i (c\_r),

.f\_o (f\_w)

);

initial begin

// Display header for clarity in simulation output

$display("a\_r b\_r c\_r | f\_w");

$monitor("%b %b %b | %b", a\_r, b\_r, c\_r, f\_w);

// Test all input combinations

a\_r = 1'b0; b\_r = 1'b0; c\_r = 1'b0;

#5 a\_r = 1'b0; b\_r = 1'b0; c\_r = 1'b1;

#5 a\_r = 1'b0; b\_r = 1'b1; c\_r = 1'b0;

#5 a\_r = 1'b0; b\_r = 1'b1; c\_r = 1'b1;

#5 a\_r = 1'b1; b\_r = 1'b0; c\_r = 1'b0;

#5 a\_r = 1'b1; b\_r = 1'b0; c\_r = 1'b1;

#5 a\_r = 1'b1; b\_r = 1'b1; c\_r = 1'b0;

#5 a\_r = 1'b1; b\_r = 1'b1; c\_r = 1'b1;

// Additional test for unknown values (x and z)

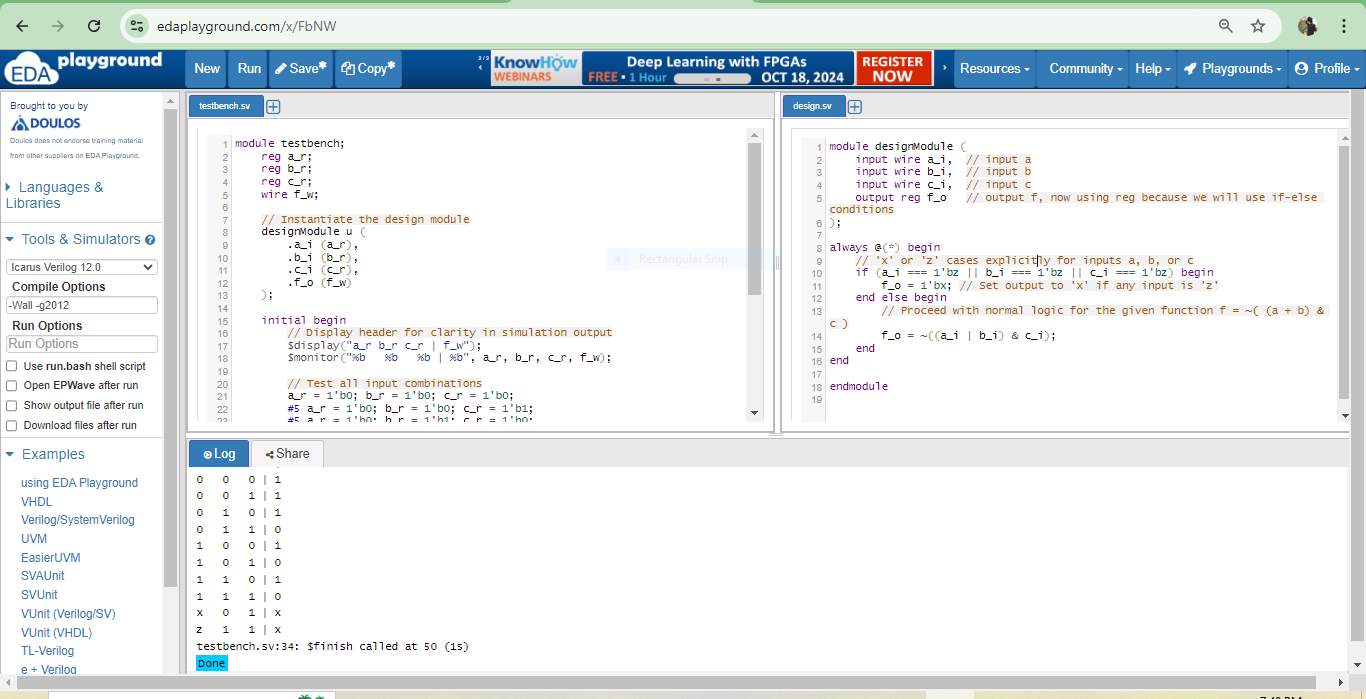
#5 a\_r = 1'bx; b\_r = 1'b0; c\_r = 1'b1;

#5 a\_r = 1'bz; b\_r = 1'b1; c\_r = 1'b1;

#5 $finish;

end

endmodule



**Q2.**

**//2 to 1 mux udp code**

primitive mux2to1\_udp (out, sel, in0, in1);

output out;

input sel, in0, in1;

// Truth table for 2-to-1 multiplexer

table

// sel in0 in1 : out

0 0 ? : 0; // Select in0

0 1 ? : 1; // Select in0

1 ? 0 : 0; // Select in1

1 ? 1 : 1; // Select in1

? x ? : x; // If in0 is unknown, output unknown

? ? x : x; // If in1 is unknown, output unknown

endtable

endprimitive

**//top module**

module mux2to1(

input wire sel,

input wire in0,

input wire in1,

output wire out

);

assign out = (sel == 0) ? in0 : in1;

endmodule

//tb

module testbench;

reg sel, in0, in1;

wire out;

mux2to1 dut(

.sel(sel),

.in0(in0),

.in1(in1),

.out(out)

);

initial begin

$monitor("Time=%0t sel=%b in0=%b in1=%b out=%b", $time, sel, in0, in1, out);

// Test case 1: sel = 0, should select in0

sel = 0; in0 = 0; in1 = 1; #10;

sel = 0; in0 = 1; in1 = 0; #10;

// Test case 2: sel = 1, should select in1

sel = 1; in0 = 0; in1 = 1; #10;

sel = 1; in0 = 1; in1 = 0; #10;

// Test case 3: X and Z values

sel = 0; in0 = 1'bx; in1 = 1'bz; #10;

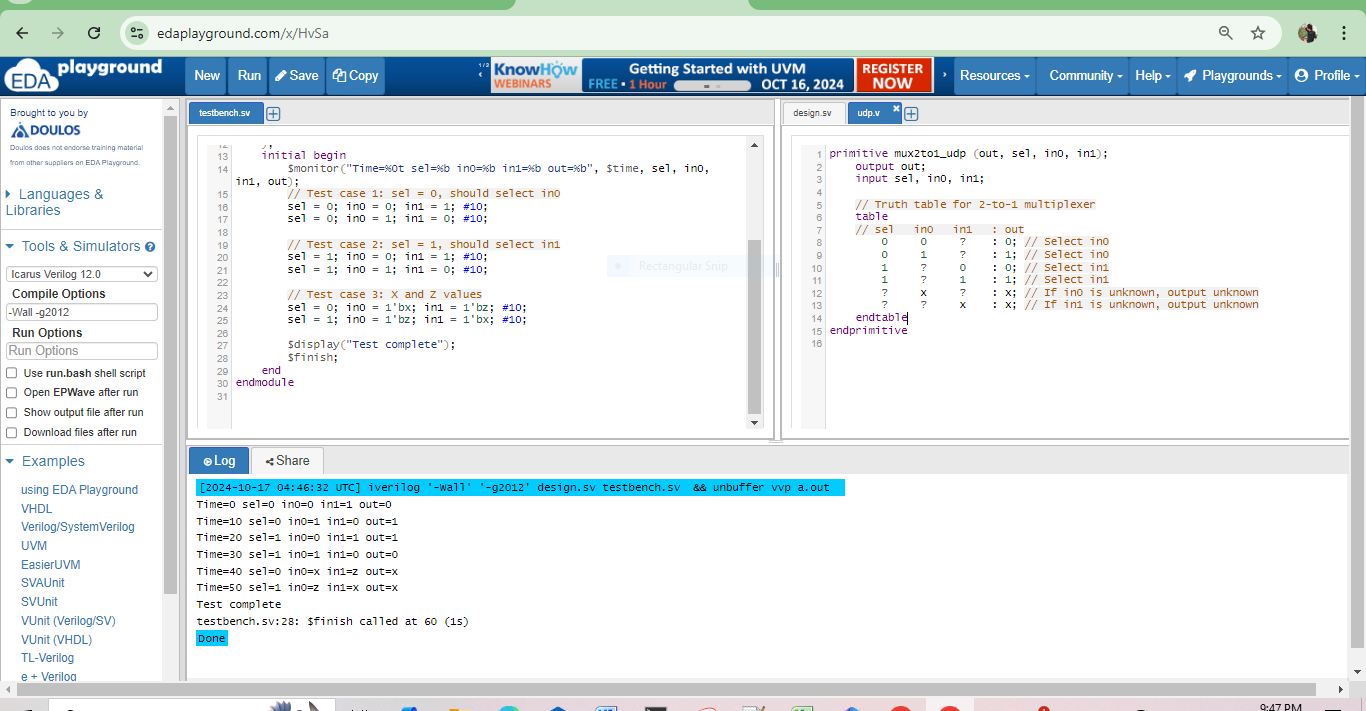
sel = 1; in0 = 1'bz; in1 = 1'bx; #10;

$display("Test complete");

$finish;

end

endmodule



**Q3.**

**Identifying the Longest Path**

**The total delay for this path(A)** = 5 (AND) + 8 (first MUX) + 3 (NOT) + 8 (second MUX) = 24 units.

**The total delay for this path (B)** = 5 (OR) + 8 (first MUX) + 3 (NOT) + 8 (second MUX) = 24 units.

**The total delay for this path(C)** = 3+5+8=16 units

**D directly selects** the MUXes, and no delay is added.

**Longest Path**

**Path (A) and Path (B)** have the same delay time of 24 units, which is the longest path

// UDP for 2:1 MUX

primitive mux2to1 (out, sel, in0, in1);

output out;

input sel, in0, in1;

table

// sel in0 in1 : out

0 ? 1 : 1;

0 ? 0 : 0;

1 1 ? : 1;

1 0 ? : 0;

endtable

endprimitive

// Top Module to instantiate the MUXes

module top\_module(output out, input C, D);

wire w1, w2, w3, w4;

// First AND gate with delay Td=5

and #(5) g1(w1, C, D);

// First OR gate with delay Td=3

or #(3) g2(w2, w1, D);

// First 2:1 MUX with delay Td=8

mux2to1 #(8) mux1(w3, D, w2, C);

// Second AND gate with delay Td=5

and #(5) g3(w4, w3, C);

// Second 2:1 MUX with delay Td=8

mux2to1 #(8) mux2(out, D, w4, C);

endmodule

//tb

module tb\_top\_module;

reg C, D;

wire out;

// Instantiate the top module

top\_module UUT (out, C, D);

initial begin

// Initialize inputs to known values

C = 0;

D = 0;

// Wait for the circuit to stabilize before starting the monitoring

#8; // Wait until 8 ns, when the first output can be valid

// Display headers

$display("Time\tC\tD\tout");

$monitor("%g\t%b\t%b\t%b", $time, C, D, out);

// Apply input combinations with some delays for better visibility

#10 C = 0; D = 1; // Change inputs at time 18ns

#10 C = 1; D = 0; // Change inputs at time 28ns

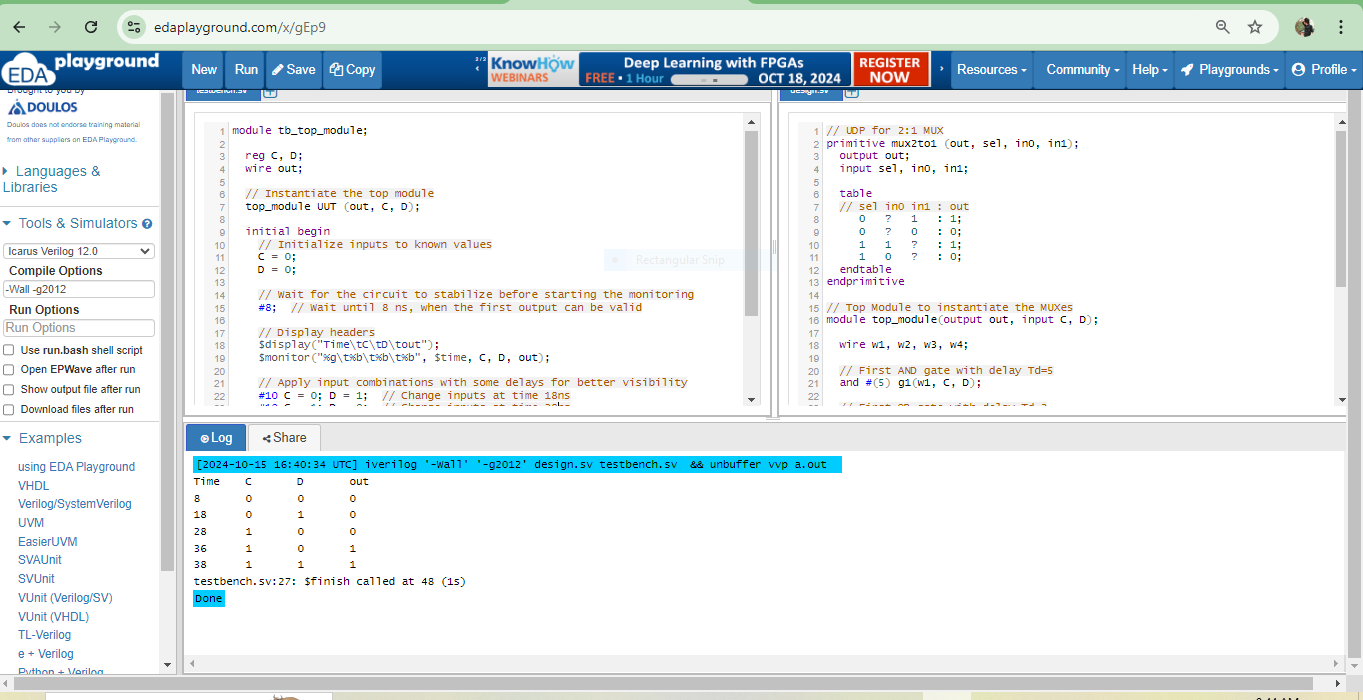
#10 C = 1; D = 1; // Change inputs at time 38ns

// Stop simulation

#10 $finish;

end

endmodule



**Q4.**

`include "basicPrimitive.v"

primitive example(a0, a1, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r0, r1);

output a0, a1, r0, r1;

input b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q;

reg a0, a1, r0, r1;

table

// a0 a1 : c r0 r1 : b d e f g h i j k l m n o p q

0 0 : 1 0 1 : 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0;

1 0 : 1 1 1 : 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ?;

0 0 : 0 0 1 : 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0;

// more entries as needed

endtable

endprimitive

**Key Fixes:**

* Used `include instead of #include.
* Declared c as an input (assuming its role as input).
* Split a and r into a0, a1, r0, and r1 (single-bit outputs).
* Corrected the table format by clearly matching input-output pairs.

**Q5.**

primitive udp\_dff (q, d, clk, rst);

output q;

input d, clk, rst;

reg q;

table

// clk d rst : q : q+ (next state)

? ? 1 : ? : 0; // Async reset

r 1 0 : ? : 1; // On rising clock, set to D

r 0 0 : ? : 0; // On rising clock, clear to D

f ? 0 : ? : -; // Falling edge, no change

? ? 0 : ? : -; // No clock, no change

endtable

endprimitive

`timescale 1ns/1ps

module counter\_3bit (

input clk,

input rst,

output reg [2:0] out

);

// Counter logic

always @(posedge clk or posedge rst) begin

if (rst) begin

out <= 3'b000; // Reset counter to 000

end else begin

out <= out + 1; // Increment the counter on clock's rising edge

end

end

endmodule

`timescale 1ns/1ps // Define the timescale for the testbench

module tb\_counter\_3bit;

reg clk;

reg rst;

wire [2:0] out;

// Instantiate the counter

counter\_3bit uut (

.clk(clk),

.rst(rst),

.out(out)

);

// Clock generation: toggle every 5 ns (10 ns period)

initial begin

clk = 0;

forever #5 clk = ~clk; // Clock toggles every 5 ns

end

// Test sequence

initial begin

rst = 1; // Assert reset

#10 rst = 0; // Deassert reset after 10 ns

// Monitor the output

$monitor("Time: %0t ns | rst: %b | out: %b", $time, rst, out);

// Run the simulation for enough time to see multiple counter outputs

#200 $finish; // End the simulation after 200 ns

end

endmodule

