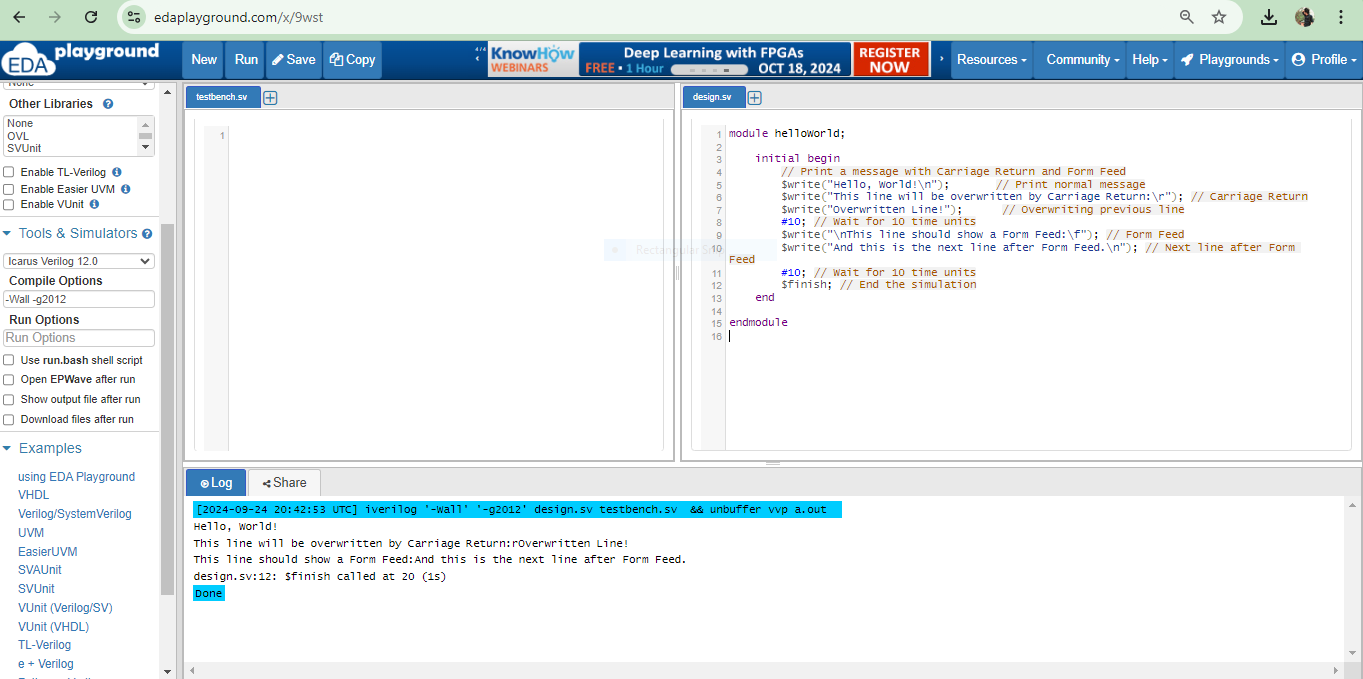
san Francisco Bay University DATE:9/27/2024 EE461 Verilog-HDL  
Homework #1

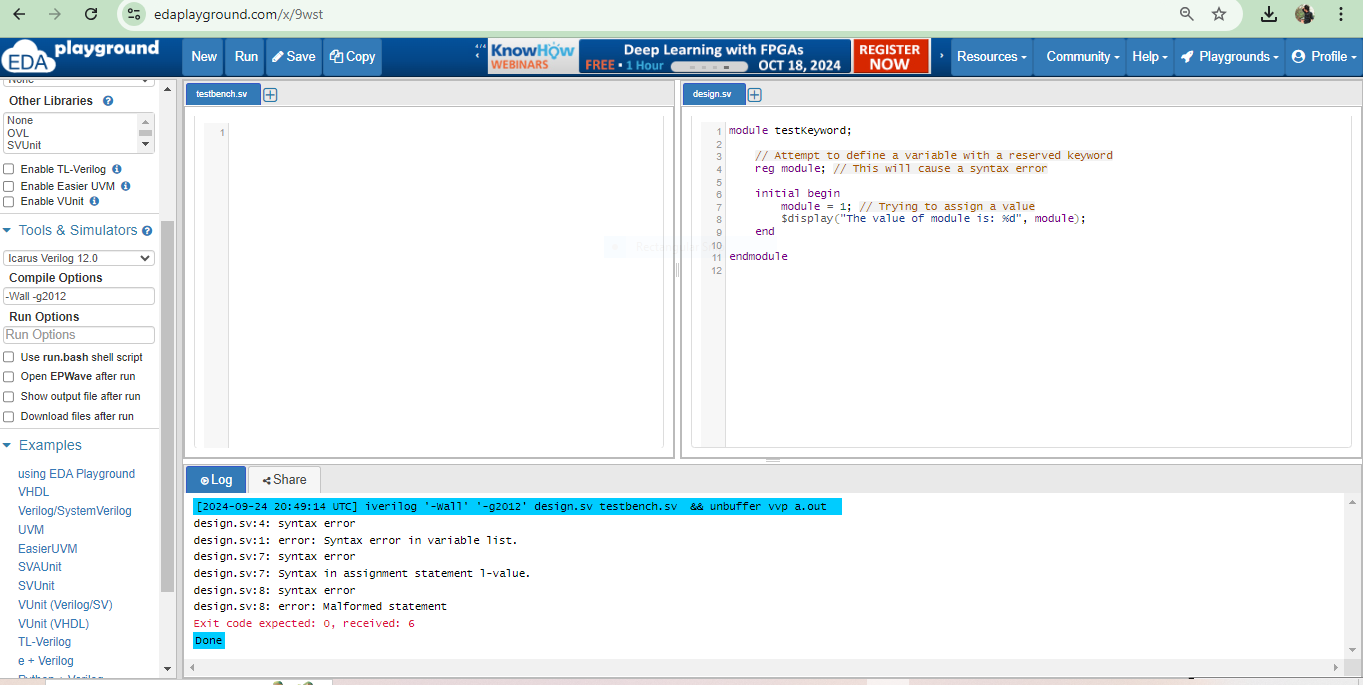
S A SABBIRUL MOHOSIN NAIM

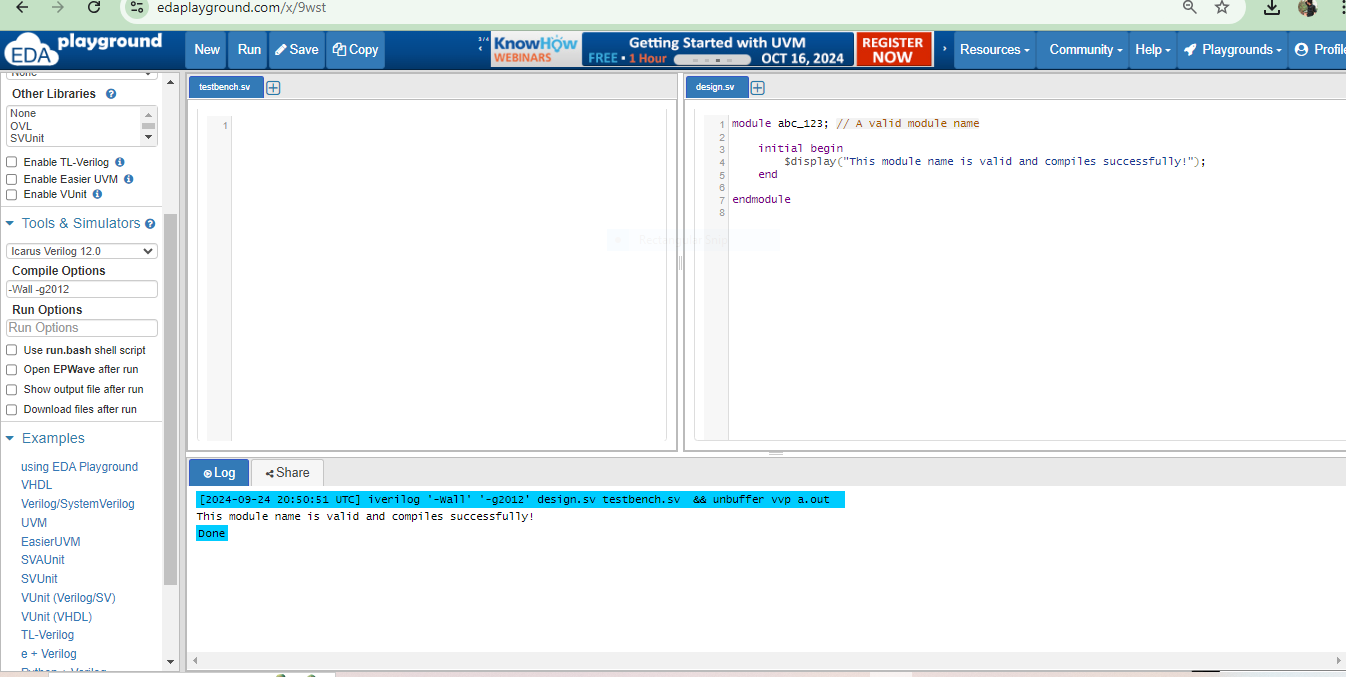
ID : 20176

**Q1.**



**Q2.**

 **Q3.**



**Q4.**

//design branch

module tri\_example;

// Define a tri-state net

tri net;

// Driver 1 driving the net with value '1'

assign net = 1'b1;

// Driver 2 driving the net with value '0'

// This will override the previous driver

assign net = 1'b0;

initial begin

// Display the value of the net

$monitor("Value of net: %b", net);

end

endmodule

**//testbranch**

module tri\_example\_z\_x;

// Define a tri-state net

tri net;

// Driver 1 driving the net with value 'z' (high impedance)

assign net = 1'bz;

// Driver 2 driving the net with value 'x' (unknown)

assign net = 1'bx;

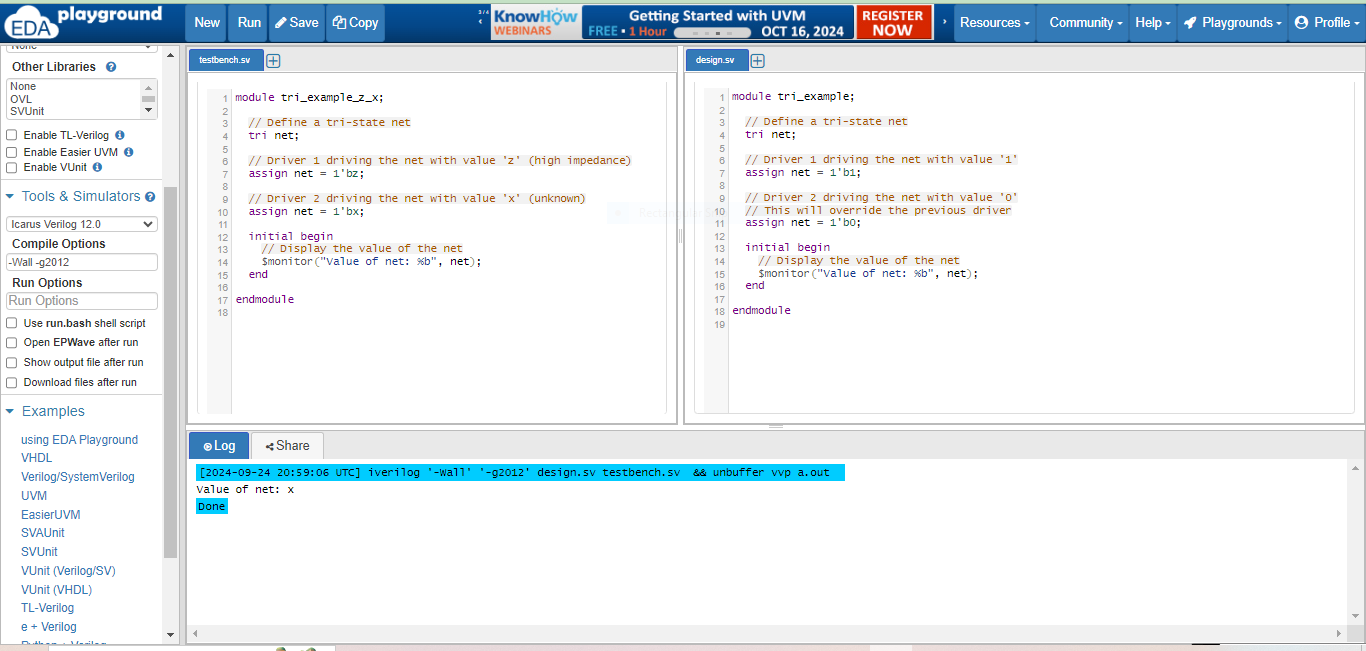
initial begin

// Display the value of the net

$monitor("Value of net: %b", net);

end

endmodule



**Q5.**

**//design**

// Design: wor\_wand\_design.sv

module wor\_wand\_design(

input logic [3:0] A, // 4-bit input

output wor output\_wor, // WOR output

output wand output\_wand // WAND output

);

// Compute the WOR and WAND outputs

assign output\_wor = A[0] | A[1] | A[2] | A[3]; // Wired OR

assign output\_wand = A[0] & A[1] & A[2] & A[3]; // Wired AND

endmodule

**// Testbench: tb\_wor\_wand.sv**

module tb\_wor\_wand;

// Declare input and output signals

logic [3:0] A; // 4-bit input for testing

logic output\_wor; // WOR output from the design

logic output\_wand; // WAND output from the design

// Instantiate the design module

wor\_wand\_design uut (

.A(A),

.output\_wor(output\_wor),

.output\_wand(output\_wand)

);

// Initial block for testing

initial begin

// Display header for output

$display("A B C D | WOR | WAND");

$display("-----------------------");

// Iterate through all combinations of inputs (16 combinations)

for (int i = 0; i < 16; i++) begin

A = i[3:0]; // Assign binary values to A, B, C, D

#1; // Wait for a time unit to allow the outputs to settle

// Display the results

$display("%b %b %b %b | %b | %b", A[3], A[2], A[1], A[0], output\_wor, output\_wand);

end

$finish; // End the simulation

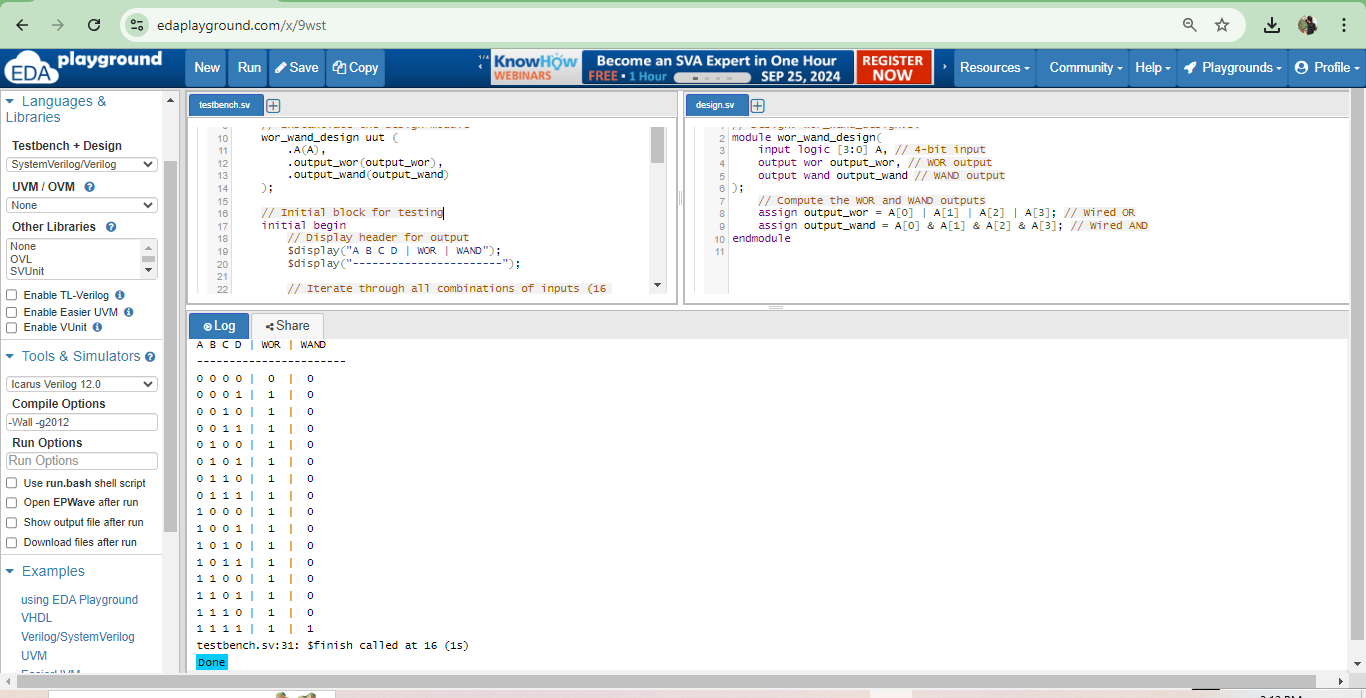
end

endmodule

**Truth table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | Wor Output | wand Output |
| 0 | **0** | **0** | **0** | **0** | **1** |
| 0 | **0** | **0** | **1** | **1** | **0** |
| 0 | **0** | **1** | **0** | **1** | **0** |
| 0 | **0** | **1** | **1** | **1** | **0** |
| 0 | **1** | **0** | **0** | **1** | **0** |
| 0 | **1** | **0** | **1** | **1** | **0** |
| 0 | **1** | **1** | **0** | **1** | **0** |
| 0 | **1** | **1** | **1** | **1** | **0** |
| 1 | **0** | **0** | **0** | **1** | **0** |
| 1 | **0** | **0** | **1** | **1** | **0** |
| 1 | **0** | **1** | **0** | **1** | **0** |
| 1 | **0** | **1** | **1** | **1** | **0** |
| 1 | **1** | **0** | **0** | **1** | **0** |
| 1 | **1** | **0** | **1** | **1** | **0** |
| 1 | **1** | **1** | **0** | **1** | **0** |
| 1 | **1** | **1** | **1** | **1** | **0** |

**OUTPUT:**



**OUTPUT SAME AS TRUTH TABLE**

**Q6.**

**//design**

`timescale 1ns/1ps

module tri\_demo;

// Declare tri-state variables

tri0 tri0\_var;

tri1 tri1\_var;

// Internal driver variables

reg drive\_tri0;

reg drive\_tri1;

// Continuous assignment to drive tri0\_var and tri1\_var

assign tri0\_var = drive\_tri0;

assign tri1\_var = drive\_tri1;

initial begin

// Drive different values to tri0\_var

drive\_tri0 = 0; // Assigning logic 0

#1 $display("tri0\_var = %b", tri0\_var); // Expected: 0

drive\_tri0 = 1; // Assigning logic 1

#1 $display("tri0\_var = %b", tri0\_var); // Expected: 1

drive\_tri0 = 1'bx; // Assigning unknown

#1 $display("tri0\_var = %b", tri0\_var); // Expected: x

drive\_tri0 = 1'bz; // Assigning high impedance

#1 $display("tri0\_var = %b", tri0\_var); // Expected: z

// Drive different values to tri1\_var

drive\_tri1 = 0; // Assigning logic 0

#1 $display("tri1\_var = %b", tri1\_var); // Expected: 0

drive\_tri1 = 1; // Assigning logic 1

#1 $display("tri1\_var = %b", tri1\_var); // Expected: 1

drive\_tri1 = 1'bx; // Assigning unknown

#1 $display("tri1\_var = %b", tri1\_var); // Expected: x

drive\_tri1 = 1'bz; // Assigning high impedance

#1 $display("tri1\_var = %b", tri1\_var); // Expected: z

// End simulation

$finish;

end

endmodule

//tb

`timescale 1ns/1ps

module tri\_demo\_tb;

// Instantiate the tri\_demo module

tri\_demo uut();

initial begin

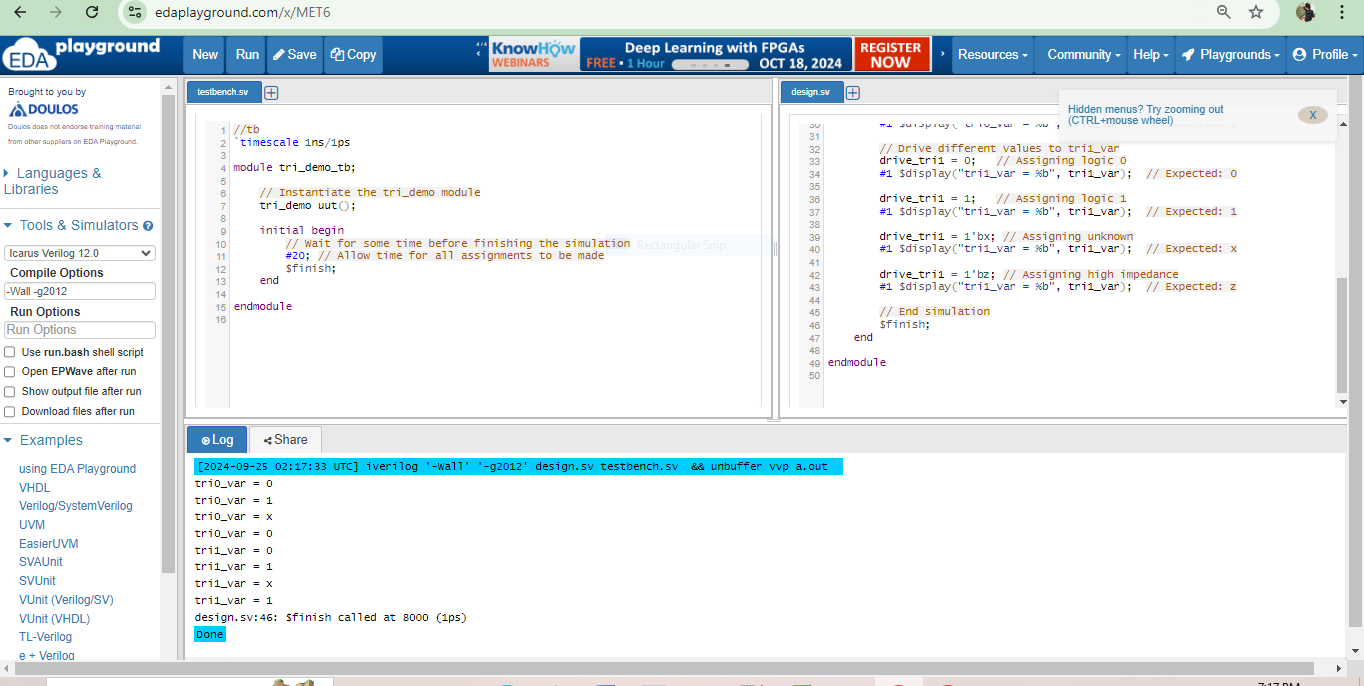
// Wait for some time before finishing the simulation

#20; // Allow time for all assignments to be made

$finish;

end

endmodule



**Q7.**

module testTrireg(

input logic A,

input logic B,

output logic tri\_signal // tri-state signal

);

// tri\_signal can be driven by A and B

assign tri\_signal = (A) ? 1'b1 : // If A is 1, tri\_signal is 1

(B) ? 1'b0 : // If B is 1, tri\_signal is 0

1'bz; // If both A and B are 0, tri\_signal is Z (high impedance)

Endmodule

module tb\_testTrireg();

reg A, B; // Test inputs

wire tri\_signal; // Tri-state output

// Instantiate the testTrireg module

testTrireg uut(

.A(A),

.B(B),

.tri\_signal(tri\_signal)

);

initial begin

// Monitor outputs

$monitor("%g: A = %b, B = %b, tri\_signal = %b", $time, A, B, tri\_signal);

// Test cases

// Initial case

A = 0; B = 0; #1; // Expect tri\_signal = Z

// Test case 1

A = 1; B = 0; #1; // Expect tri\_signal = 1

// Test case 2

A = 0; B = 1; #1; // Expect tri\_signal = 0

// Test case 3

A = 1; B = 1; #1; // Expect tri\_signal = 1 (priority to A)

// Test case 4

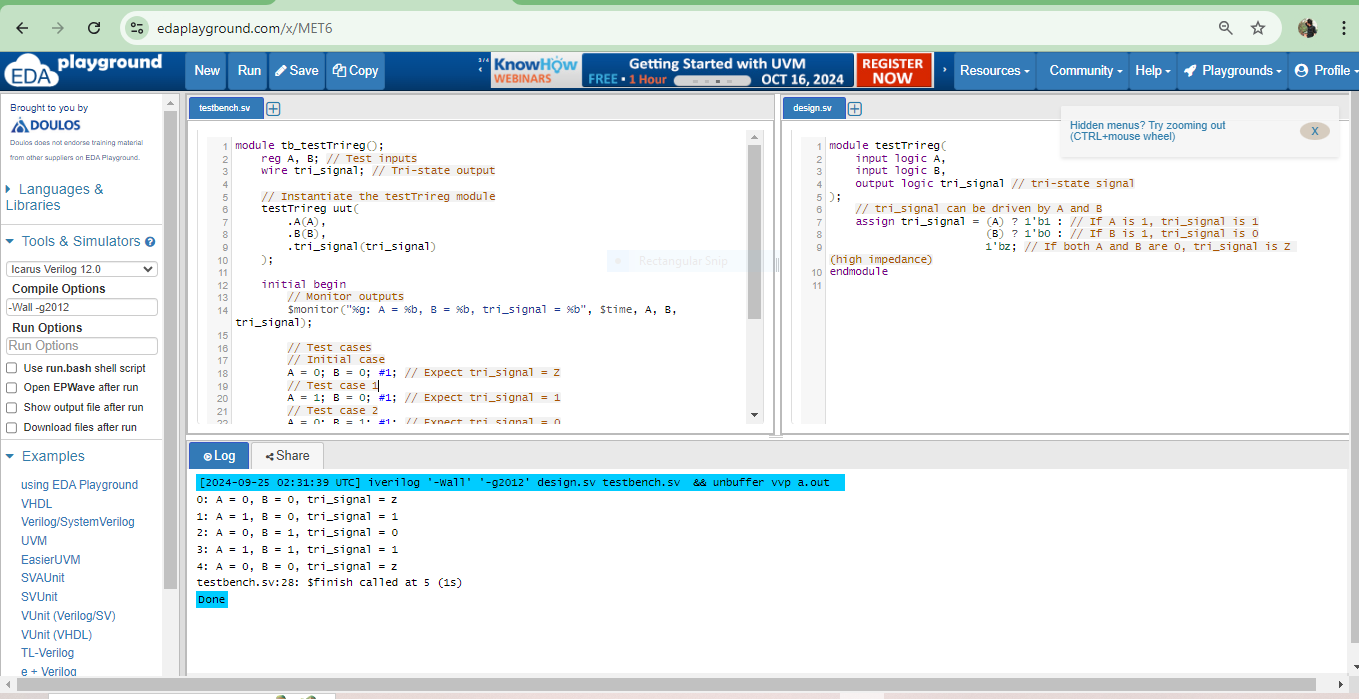
A = 0; B = 0; #1; // Expect tri\_signal = Z

// Finish simulation

$finish;

end

endmodule



**Q8.**

module testInteger(

input signed [31:0] a,

input signed [31:0] b,

output signed [31:0] sum

);

assign sum = a + b; // Perform addition

endmodule

module tb\_testInteger();

// Declare test inputs and output as signed integers

reg signed [31:0] a\_r, b\_r; // 32-bit signed registers for inputs

wire signed [31:0] sum\_w; // 32-bit signed wire for output

// Instantiate the testInteger module

testInteger uut(

.a(a\_r),

.b(b\_r),

.sum(sum\_w)

);

initial begin

// Monitor the outputs

$monitor("%g: a = %d, b = %d, sum = %d", $time, a\_r, b\_r, sum\_w);

// Test cases

a\_r = 10; b\_r = 20; #1; // Expect sum = 30

a\_r = -5; b\_r = 15; #1; // Expect sum = 10

a\_r = 100; b\_r = 50; #1; // Expect sum = 150

a\_r = 0; b\_r = 0; #1; // Expect sum = 0

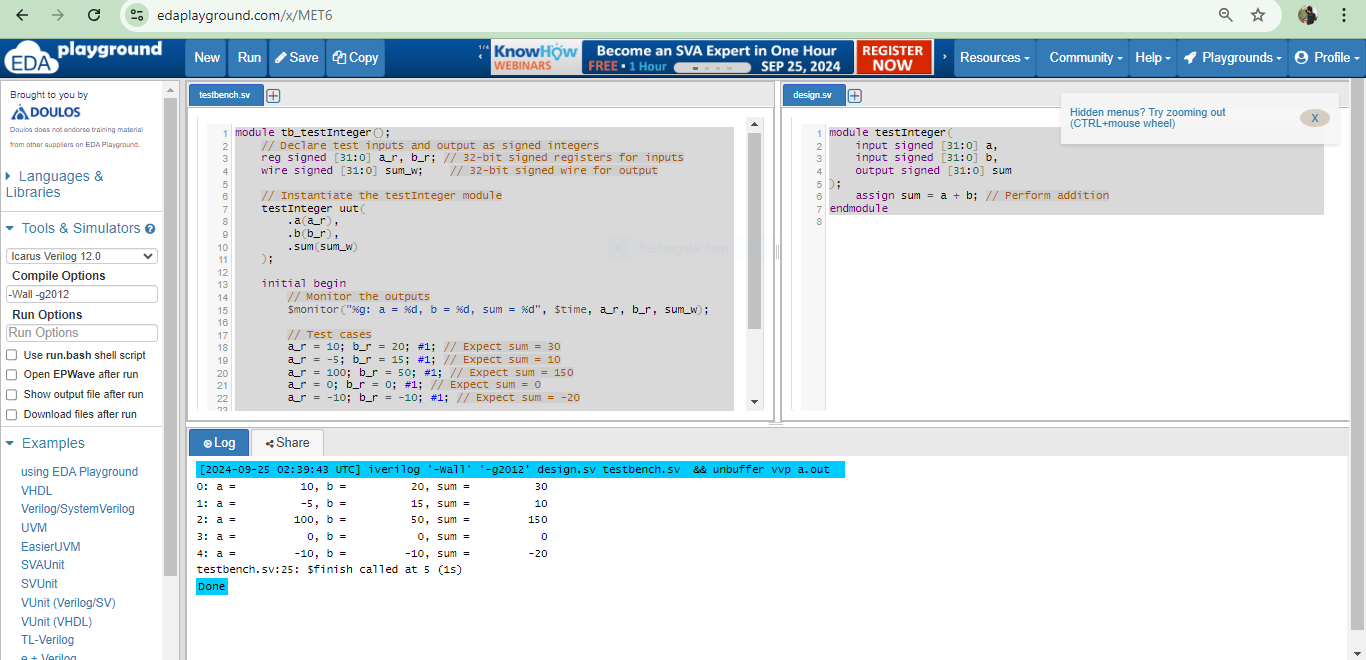
a\_r = -10; b\_r = -10; #1; // Expect sum = -20

// Finish simulation

$finish;

end

endmodule



**Q9.**

//design

module time\_example;

// Declare a time variable

time my\_time;

// Display simulation time and real time

initial begin

// Wait for some time to ensure the simulation has started

#10; // Wait 10 time units

// Assign values to my\_time and display them

my\_time = $stime;

$display("Simulation time (stime): %0t", my\_time);

my\_time = $realtime;

$display("Real time (realtime): %0t", my\_time);

end

endmodule

//tb

module tb\_time\_example;

// Instantiate the design module

time\_example uut();

// Run the simulation

initial begin

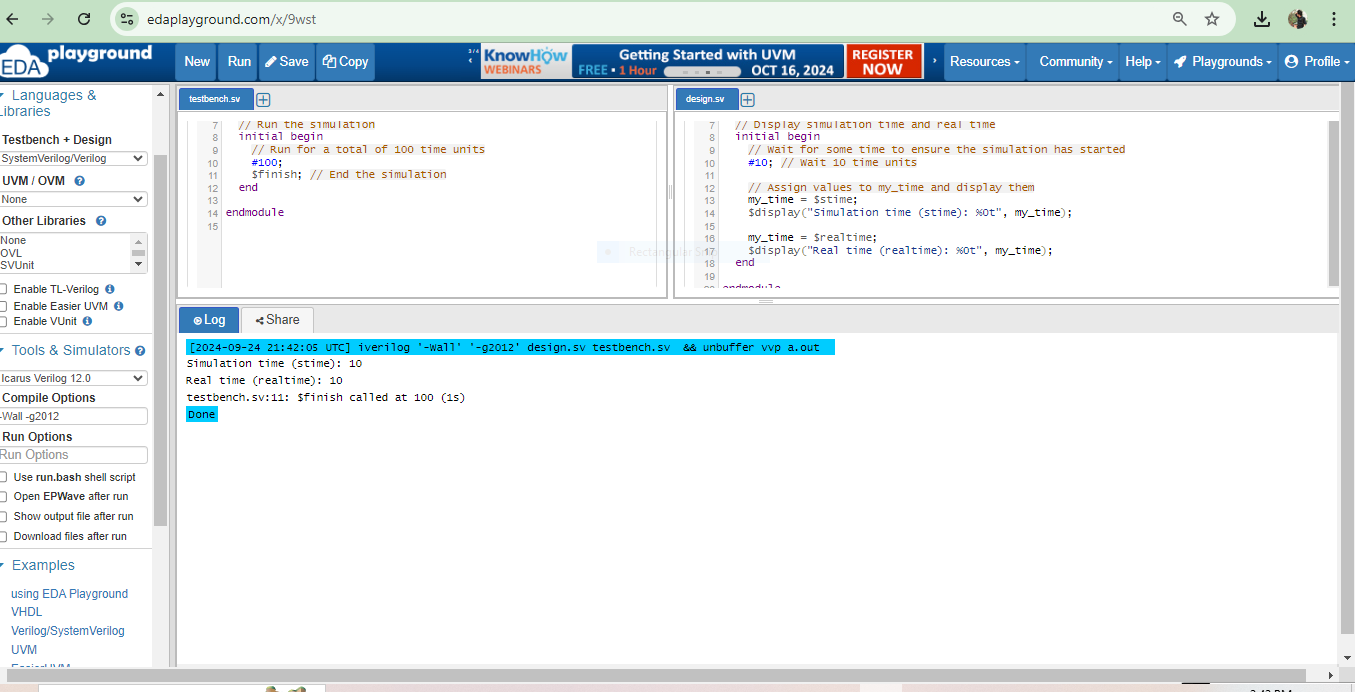
// Run for a total of 100 time units

#100;

$finish; // End the simulation

end

endmodule



**Q10.**

`timescale 10ns / 100ps

module DelayTest();

// Define a real variable for the delay time

real delay\_time = 2.71828; // Delay in time units

real real\_delay; // Variable to hold calculated real delay

initial begin

// Calculate the real delay in nanoseconds

real\_delay = delay\_time \* 10; // Convert time units to nanoseconds

// Introduce the delay

#delay\_time; // Apply delay

// Display the results

$display("Calculated Real Delay: %.6f ns", real\_delay);

$display("Current Simulation Time: %.6f ns", $time \* 10); // Convert simulation time to ns

$display("Difference: %.6f ns", (($time \* 10) - real\_delay));

end

endmodule

//tb

`timescale 10ns / 100ps

module tb();

// Instantiate the DelayTest module

DelayTest dt();

initial begin

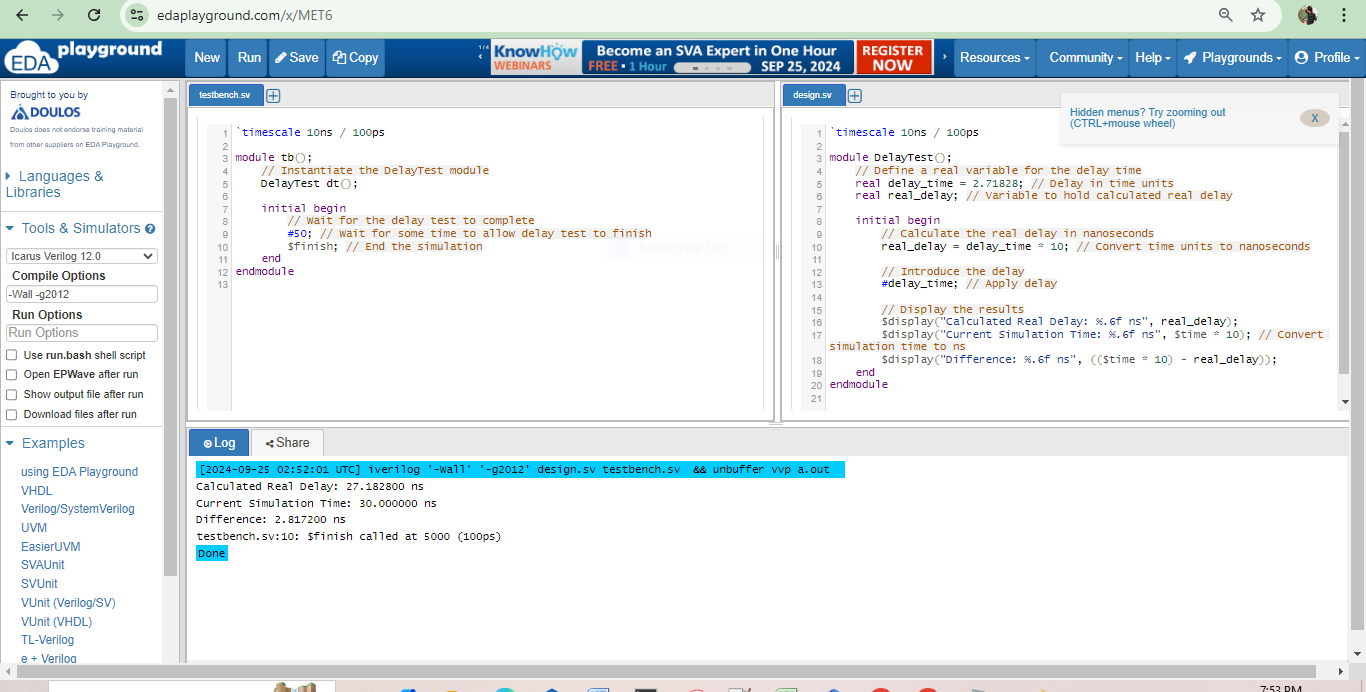
// Wait for the delay test to complete

#50; // Wait for some time to allow delay test to finish

$finish; // End the simulation

end

endmodule



**Q11.**

`timescale 1ns / 1ps // Define a timescale of 1 nanosecond and 1 picosecond

module signedNumber(

input signed [7:0] a, // 8-bit signed input

input signed [7:0] b, // 8-bit signed input

output signed [8:0] sum // 9-bit signed output to accommodate overflow

);

assign sum = a + b; // Perform addition of signed numbers

endmodule

//testbranch

`timescale 1ns / 1ps // Define a timescale of 1 nanosecond and 1 picosecond

module tb\_signedNumber();

reg signed [7:0] a\_r; // 8-bit signed register for input a

reg signed [7:0] b\_r; // 8-bit signed register for input b

wire signed [8:0] sum\_w; // 9-bit signed wire for output

// Instantiate the signedNumber module

signedNumber uut (

.a(a\_r),

.b(b\_r),

.sum(sum\_w)

);

initial begin

// Monitor the results

$monitor("Time: %0t | a: %0d | b: %0d | sum: %0d", $time, a\_r, b\_r, sum\_w);

// Test Cases

a\_r = 8'd10; b\_r = 8'd20; // 10 + 20 = 30

#10;

a\_r = 8'd50; b\_r = 8'd80; // 50 + 80 = 130

#10;

a\_r = -8'd30; b\_r = -8'd20; // -30 + -20 = -50

#10;

a\_r = 8'd100; b\_r = -8'd50; // 100 - 50 = 50

#10;

a\_r = -8'd128; b\_r = 8'd1; // -128 + 1 = -127

#10;

a\_r = 8'd127; b\_r = 8'd1; // 127 + 1 = 128 (overflow)

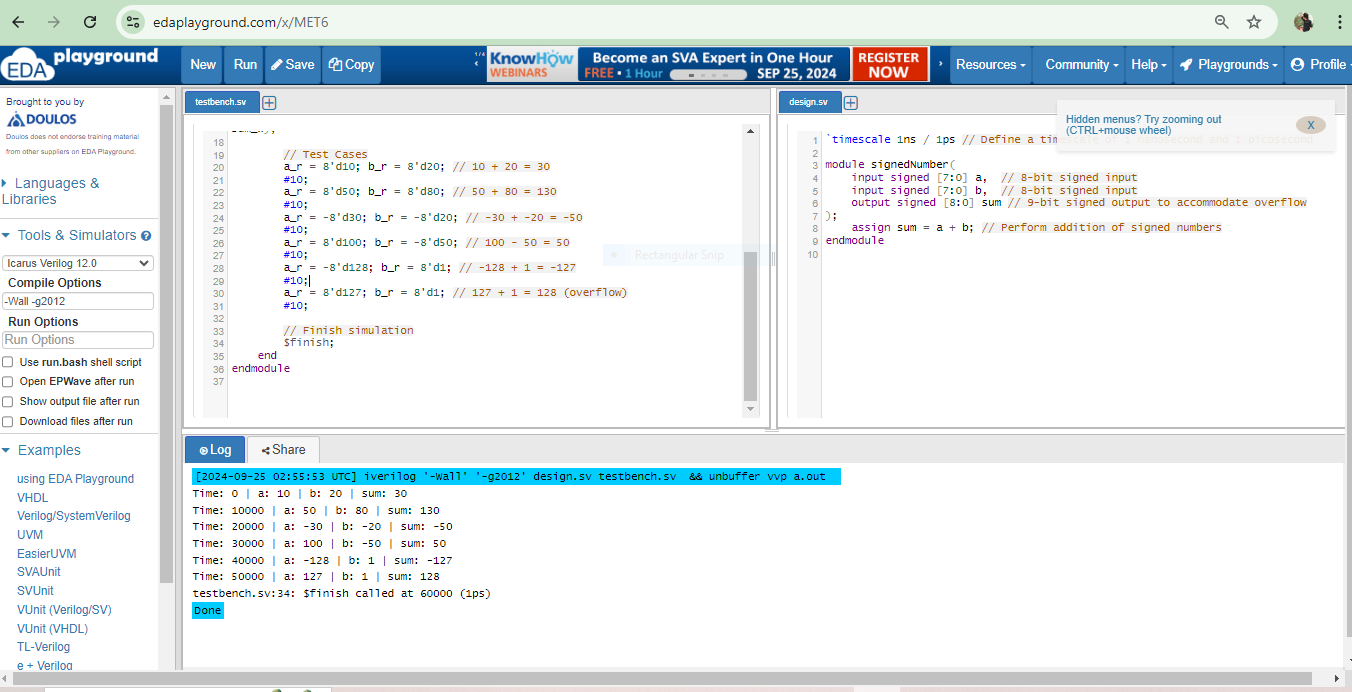
#10;

// Finish simulation

$finish;

end

endmodule



**Q12.**

`timescale 1ns/1ps

module helloWorld;

// Initial block to execute at the start of simulation

initial begin

// Print ASCII codes for the characters:

// Double quote (")

// Percent (%)

// At (@)

$display("ASCII code for double quote (\"): %d", 8'h22); // 0x22 is the hex code for "

$display("ASCII code for percent (%%): %d", 8'h25); // 0x25 is the hex code for %

$display("ASCII code for at (@): %d", 8'h40); // 0x40 is the hex code for @

// End the simulation

$finish;

end

endmodule

//test branch

`timescale 1ns/1ps

module helloWorld\_tb;

// Instantiate the helloWorld module

helloWorld hw();

// Initial block for simulation

initial begin

// Monitor simulation output

$monitor("Time: %0t", $time);

// Run the simulation for a short time to capture the output

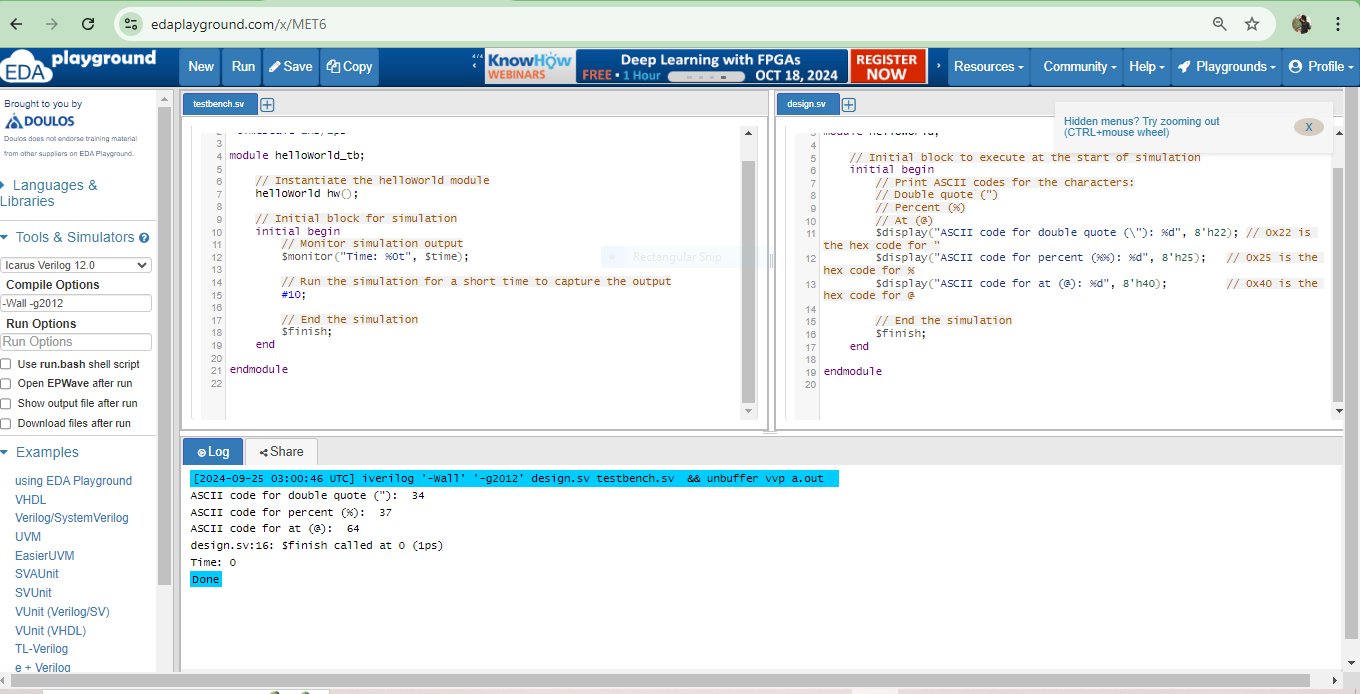
#10;

// End the simulation

$finish;

end

endmodule



**Q13.**

**//design**

`timescale 1ns/1ps

module format\_demo;

// Declare variables of different types

reg [7:0] binary\_val = 8'b10101010; // Binary value

reg [7:0] char\_val = 8'h41; // Character value (ASCII code for 'A')

reg [31:0] decimal\_val = 32'd12345; // Decimal value

reg [31:0] decimal\_val\_0 = 32'd12345; // Decimal value with leading zero

real float\_val = 123.456; // Floating-point value

real float\_val\_precise = 123.45678; // Floating-point value with precision

reg [7:0] hex\_val = 8'h2F; // Hexadecimal value (ASCII code for '/')

reg [7:0] octal\_val = 8'o57; // Octal value

reg [63:0] time\_val = 64'd1000000000; // Time value

// Initial block to execute at the start of simulation

initial begin

// Display binary value

$display("Binary value (%%b): %b", binary\_val);

// Display character value

$display("Character value (%%c): %c", char\_val);

// Display decimal value

$display("Decimal value (%%d): %d", decimal\_val);

// Display decimal value with leading zero

$display("Decimal value with leading zero (%%0d): %0d", decimal\_val\_0);

// Display scientific notation

$display("Scientific notation (%%e): %e", float\_val);

// Display floating-point value

$display("Floating-point value (%%f): %f", float\_val);

// Display floating-point value with field width and precision

$display("Floating-point value (%%6.2f): %6.2f", float\_val\_precise);

// Display floating-point value in the most compact form

$display("Most compact floating-point value (%%g): %g", float\_val\_precise);

// Display hexadecimal value

$display("Hexadecimal value (%%h): %h", hex\_val);

// Display octal value

$display("Octal value (%%o): %o", octal\_val);

// Display time value in simulation time

$display("Time value (%%t): %t", time\_val);

// Display time value with leading zeros

$display("Time value with leading zeros (%%00t): %00t", time\_val);

// End the simulation

$finish;

end

endmodule

**//testbranch**

`timescale 1ns/1ps

module format\_demo\_tb;

// Instantiate the format\_demo module

format\_demo fd();

// Initial block for simulation

initial begin

// Monitor simulation output

$monitor("Time: %0t", $time);

// Run the simulation for a short time to capture the output

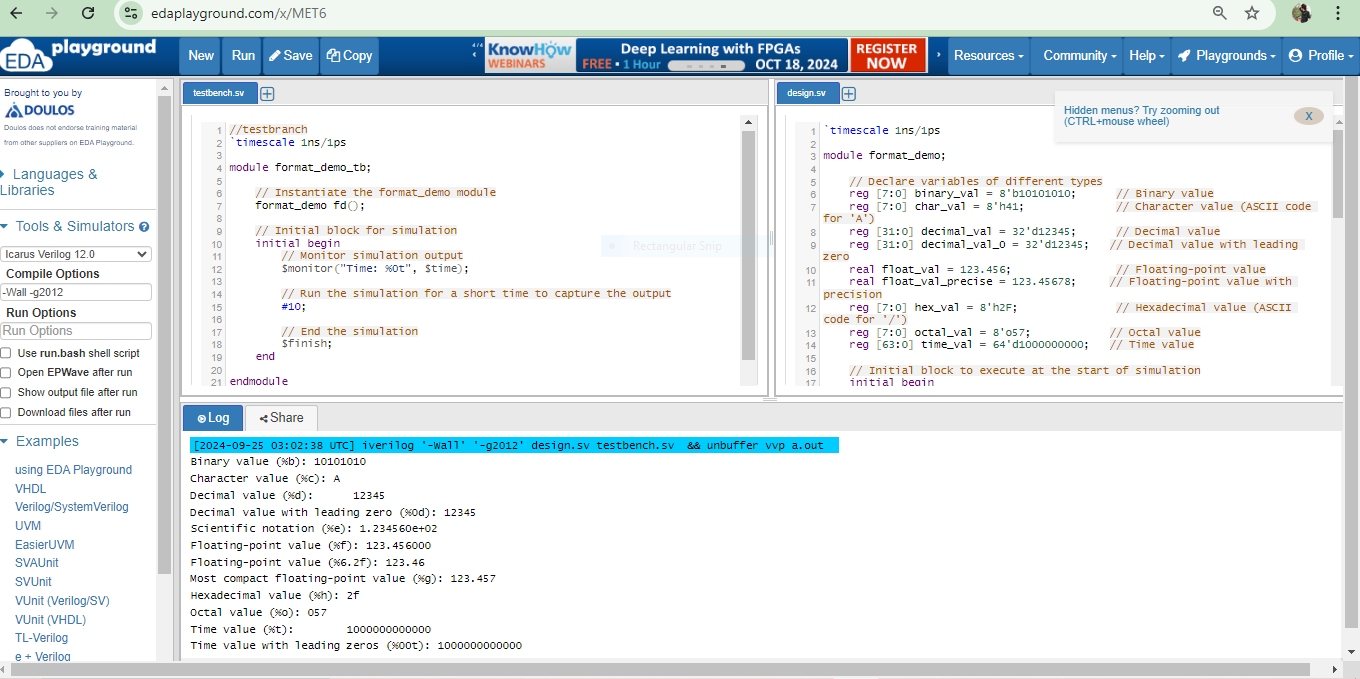
#10;

// End the simulation

$finish;

end

endmodule



**Q14.**

//design

`timescale 1ns / 1ps // Define timescale

module compareDisplayWriteMonitor;

reg [3:0] count; // A simple counter

initial begin

count = 0; // Initialize counter

// Using $monitor to observe changes in 'count'

$monitor("Monitor: Time = %0t, Count = %0d", $time, count);

// Using $write to print values without a newline

$write("Write: Initial Count = %0d ", count); // This won't add a newline

// Incrementing the counter in a loop

repeat (5) begin

#5 count = count + 1; // Delay for 5 time units

// Using $display to print values with a newline

$display("Display: Count = %0d", count);

end

// Final message

$display("Final Count = %0d", count);

$finish; // End simulation

end

endmodule

**//test branch**

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

module testbench;

// Instantiate the compareDisplayWriteMonitor module

compareDisplayWriteMonitor uut(); // uut: Unit Under Test

initial begin

// Simulation time

#40; // Run the simulation for 40 time units

$finish; // End simulation

end

endmodule

