San Francisco Bay University  
EE461 Verilog-HDL  
 Homework #2 Date: 10/04/2024

S A SABBIRUL MOHOSIN NAIM (20176)

**1.**

// Code your design here

// Code your testbench here

// or browse Examples

module arithmetic\_operations();

reg signed [3:0] a, b;

reg signed [4:0] b5bit;

reg signed [3:0] result\_add, result\_sub, result\_mul, result\_mod;

initial begin

// Assign values

a = 4'b1111; // 'a' is 4'b1111, which is -1 in signed 4-bit

b = -5'b00010; // 'b' is -2 in 5-bit

b5bit = -5'b01xz; // 'b' contains unknown values

// Perform arithmetic operations with valid b = -2

result\_add = a + b[3:0]; // Using lower 4 bits of 'b'

result\_sub = a - b[3:0];

result\_mul = a \* b[3:0];

result\_mod = a % b[3:0];

// Display the results for b = -5'b00010

$display("When b = -5'b00010:");

$display("a + b = %d", result\_add);

$display("a - b = %d", result\_sub);

$display("a \* b = %d", result\_mul);

$display("a %% b = %d", result\_mod);

// Operations with b5bit = -5'b01xz, handle unknown 'x' or 'z' bits

result\_add = a + b5bit[3:0];

result\_sub = a - b5bit[3:0];

result\_mul = a \* b5bit[3:0];

result\_mod = a % b5bit[3:0];

// Display the results for b = -5'b01xz

$display("When b = -5'b01xz:");

$display("a + b = %d", result\_add);

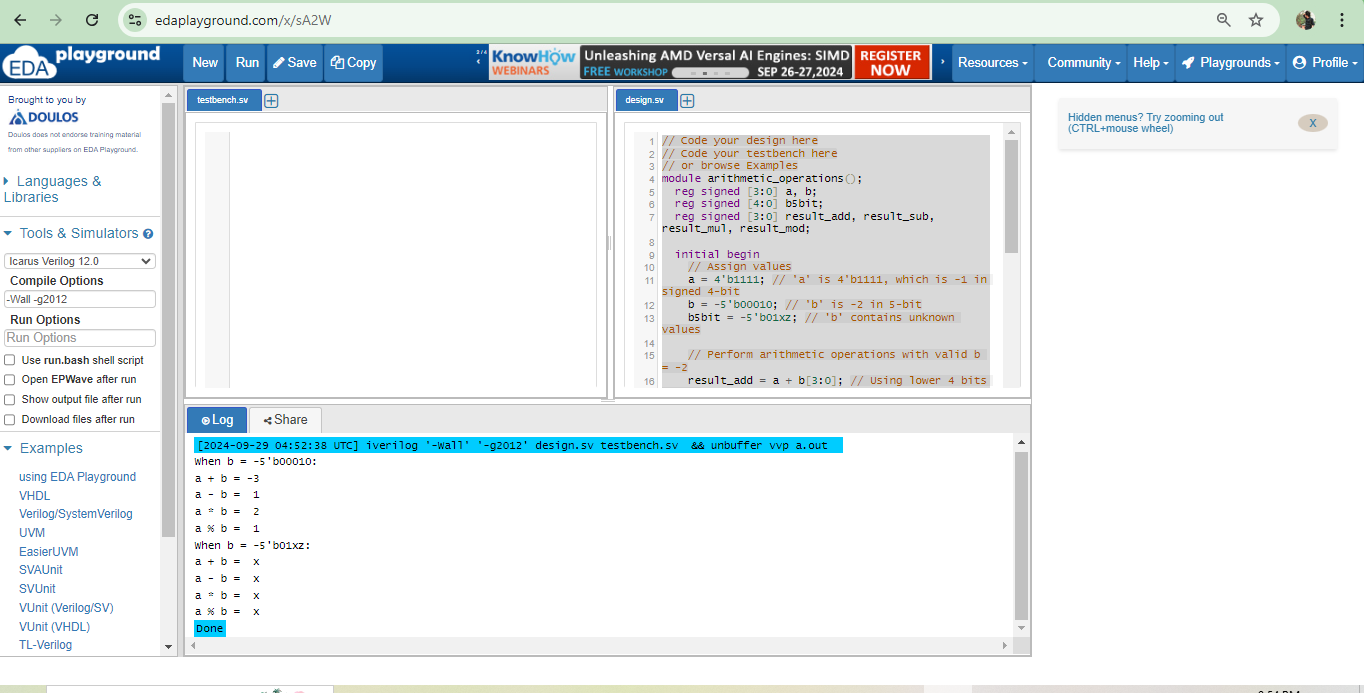
$display("a - b = %d", result\_sub);

$display("a \* b = %d", result\_mul);

$display("a %% b = %d", result\_mod);

end

endmodule



**2.**

// Code your design here

module compare\_values();

reg [1:0] a\_2bit; // 2-bit value of 'a'

reg [2:0] b\_3bit; // 3-bit value of 'b'

reg [3:0] a\_4bit; // 4-bit value of 'a' for the second test

initial begin

// Case 1: a = 2'b1z and b = 3'b11z

a\_2bit = 2'b1z; // 'a' = 1z (unknown bit in 2-bit)

b\_3bit = 3'b11z; // 'b' = 11z (unknown bit in 3-bit)

// Perform comparisons and display results for a = 2'b1z, b = 3'b11z

$display("Case 1: a = 2'b1z and b = 3'b11z:");

$display("a > b: %b", (a\_2bit > b\_3bit));

$display("a >= b: %b", (a\_2bit >= b\_3bit));

$display("a < b: %b", (a\_2bit < b\_3bit));

$display("a <= b: %b", (a\_2bit <= b\_3bit));

// Case 2: a = 4'b01xz

a\_4bit = 4'b01xz; // 'a' = 01xz (unknown bits in 4-bit)

// Perform comparisons with a 4-bit value of 'a'

$display("\nCase 2: a = 4'b01xz and b = 3'b11z:");

$display("a > b: %b", (a\_4bit > b\_3bit));

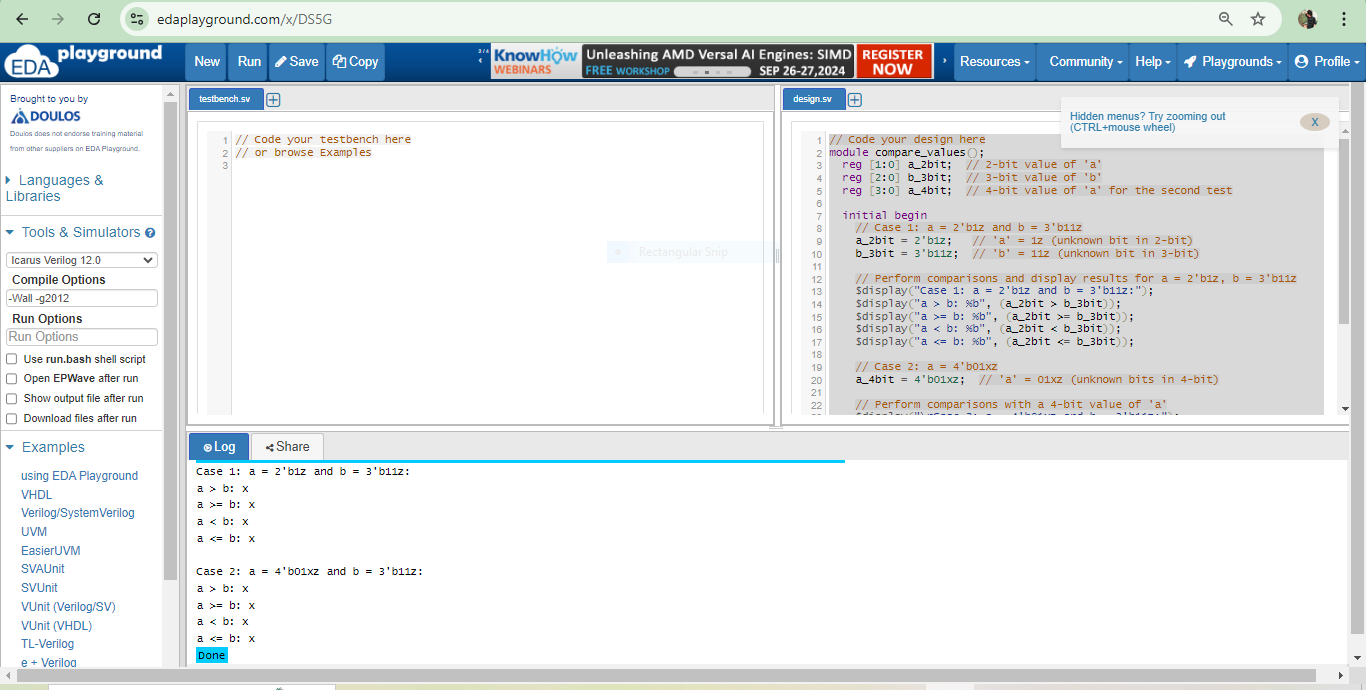
$display("a >= b: %b", (a\_4bit >= b\_3bit));

$display("a < b: %b", (a\_4bit < b\_3bit));

$display("a <= b: %b", (a\_4bit <= b\_3bit));

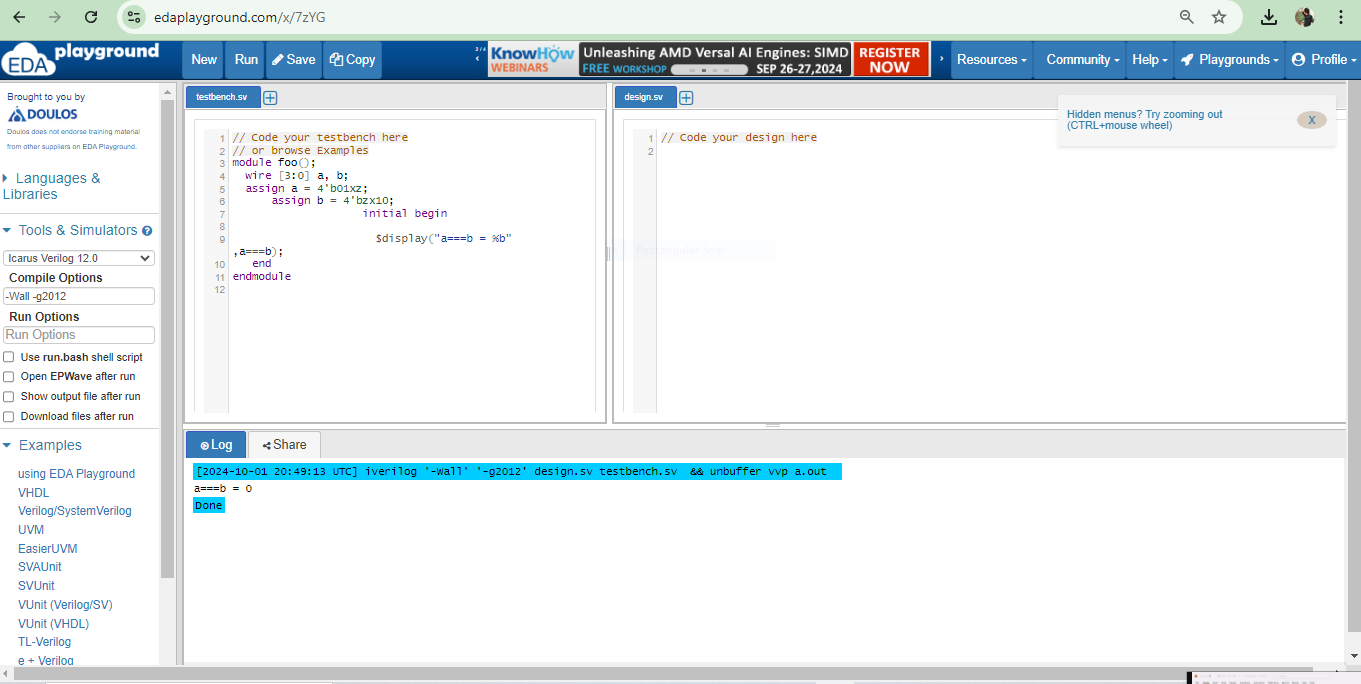
end

endmodule



**3.**

if a = 4’b01xz and b = 4’bzx10 for (a===b)



**//For (a !== b )**

module foo();

wire [3:0] a, b;

assign a = 4'b01xz;

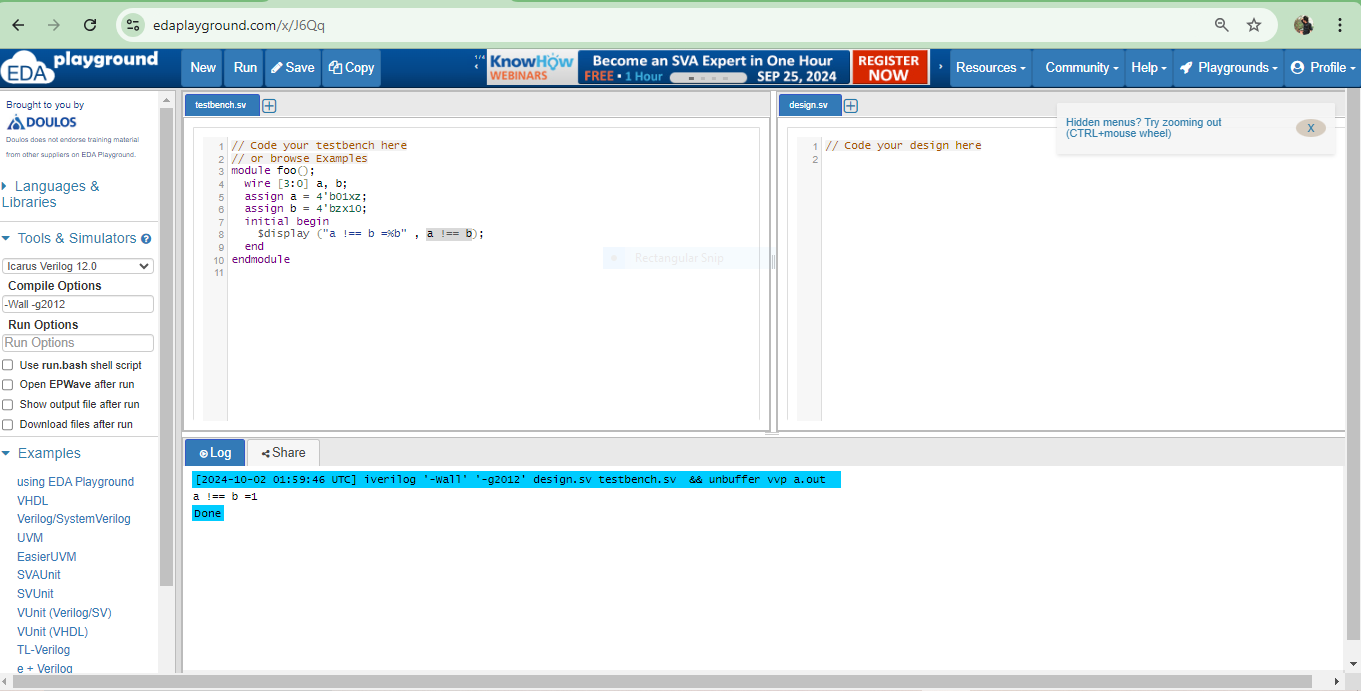
assign b = 4'bzx10;

initial begin

$display ("a !== b =%b" , a !== b);

end

endmodule



**//For (a == b)**

// Code your testbench here

// or browse Examples

module foo();

wire [3:0] a, b;

assign a = 4'b01xz;

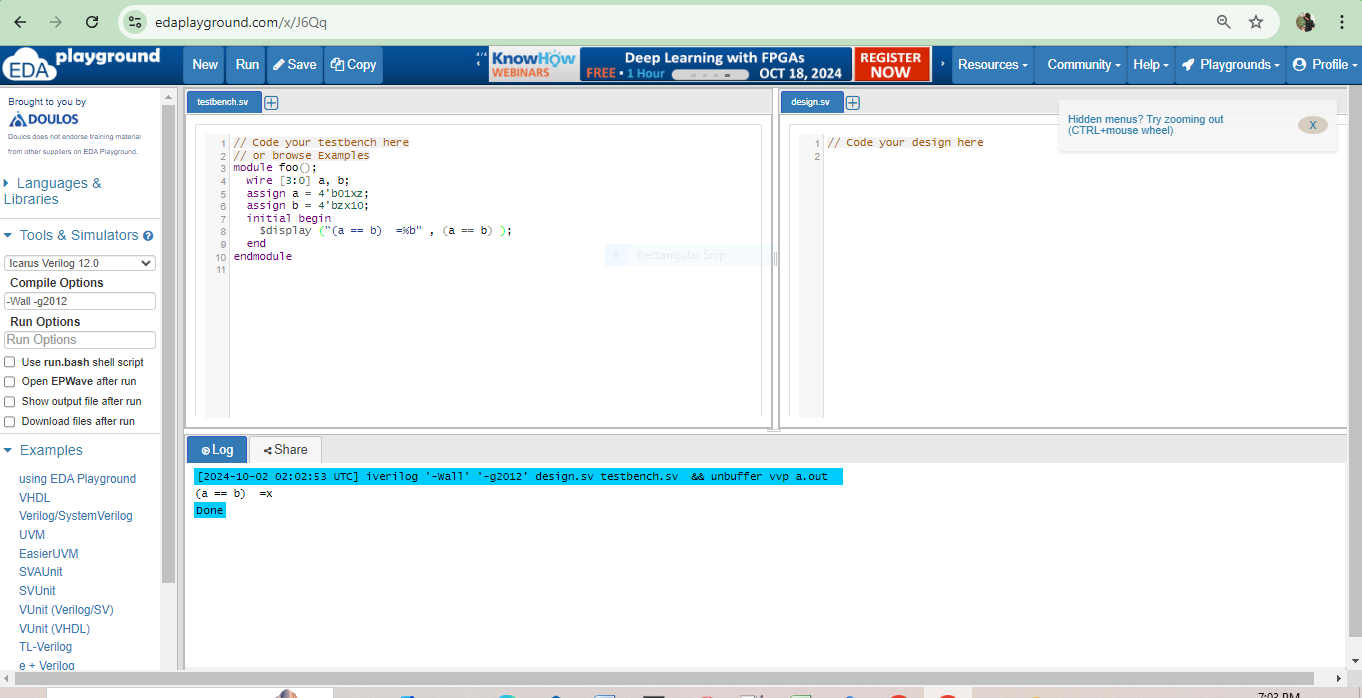
assign b = 4'bzx10;

initial begin

$display ("(a == b) =%b" , (a == b) );

end

endmodule



**//for (a != b)**

// Code your testbench here

// or browse Examples

module foo();

wire [3:0] a, b;

assign a = 4'b01xz;

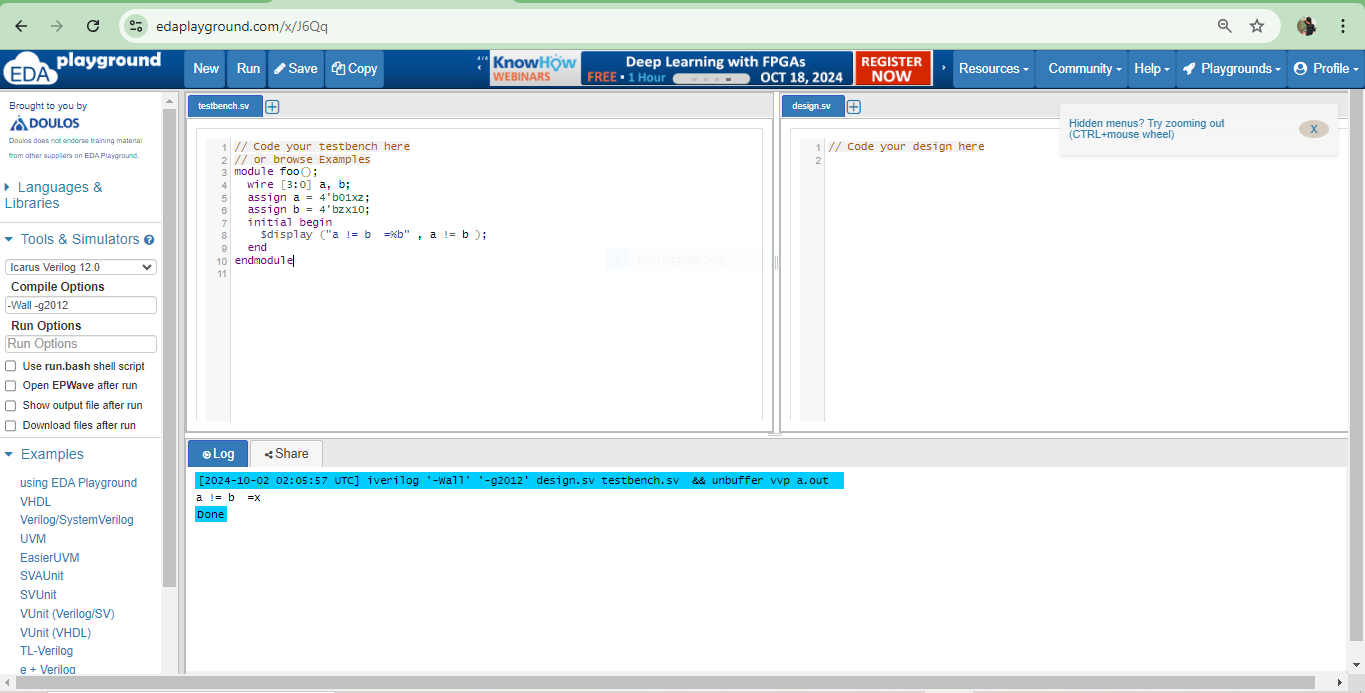
assign b = 4'bzx10;

initial begin

$display ("a != b =%b" , a != b );

end

endmodule



//If a = 4'b01zz and b=4'b0100;

//For a == b

// Code your testbench here

// or browse Examples

module foo();

wire [3:0] a, b;

assign a = 4'b01zz;

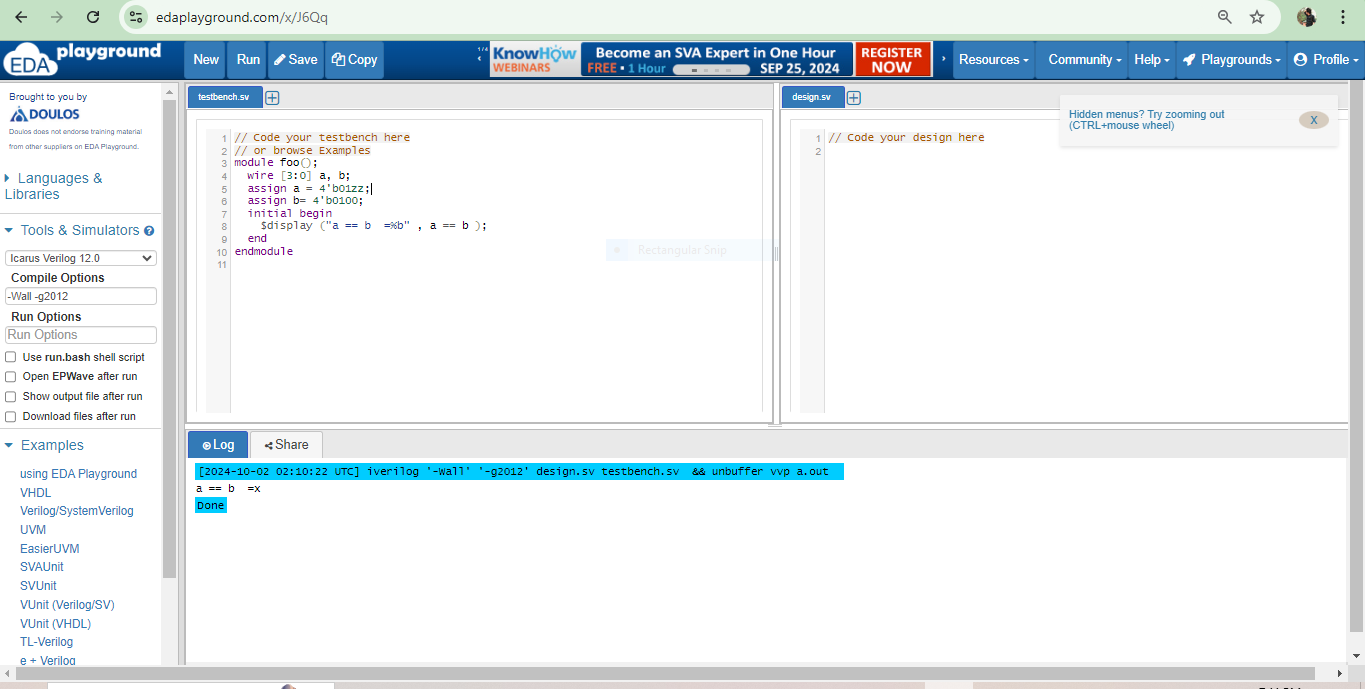
assign b= 4'b0100;

initial begin

$display ("a == b =%b" , a == b );

end

endmodule



**// for a!=b**

// or browse Examples

module foo();

wire [3:0] a, b;

assign a= 4'b01zz;

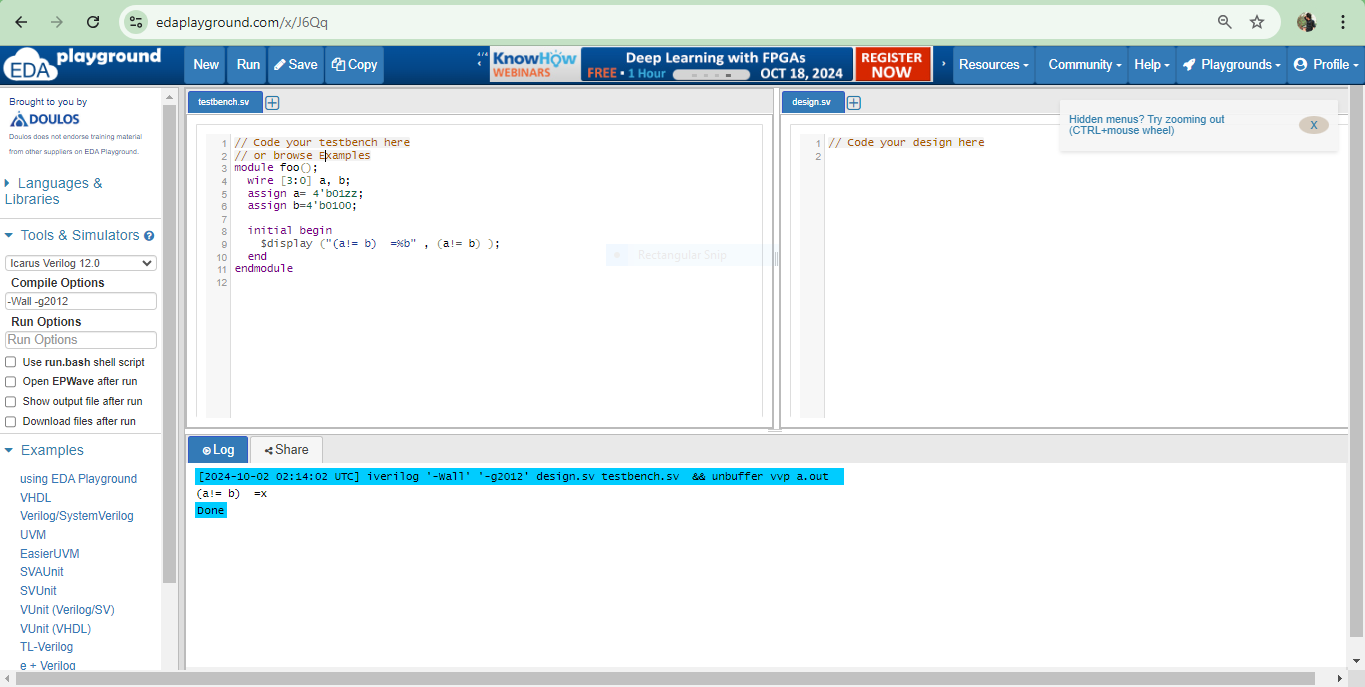
assign b=4'b0100;

initial begin

$display ("(a!= b) =%b" , (a!= b) );

end

endmodule



**If a = 4’b01xz and b = 4’b01xz;**

**//for** (a === b)

// Code your testbench here

// or browse Examples

module foo();

wire [3:0] a, b;

assign a=4'b01xz;

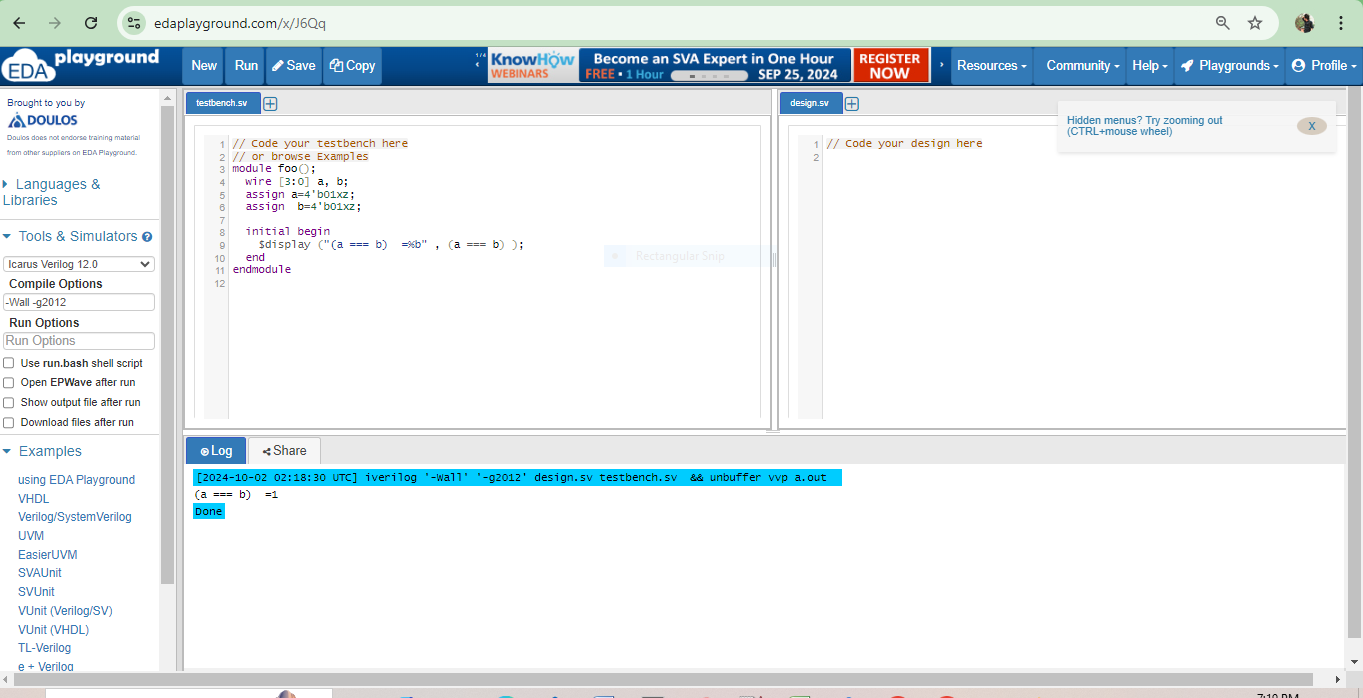
assign b=4'b01xz;

initial begin

$display ("(a === b) =%b" , (a === b) );

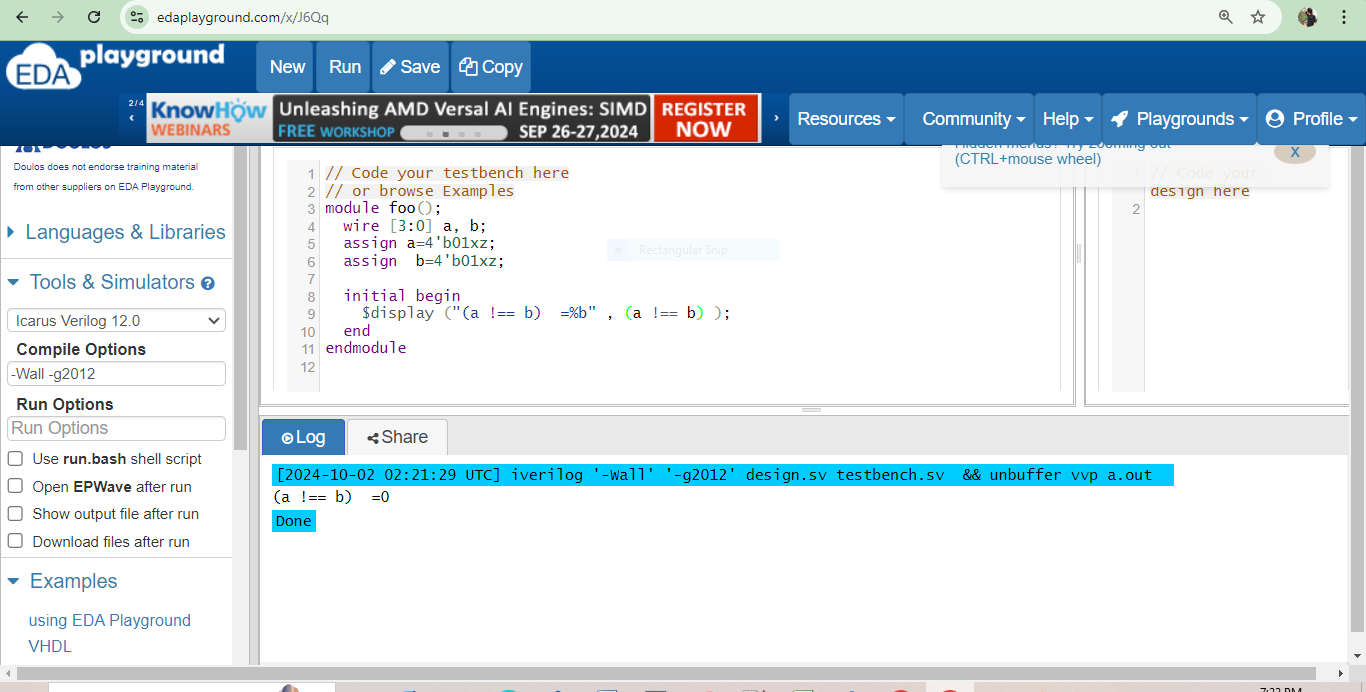
end

endmodule

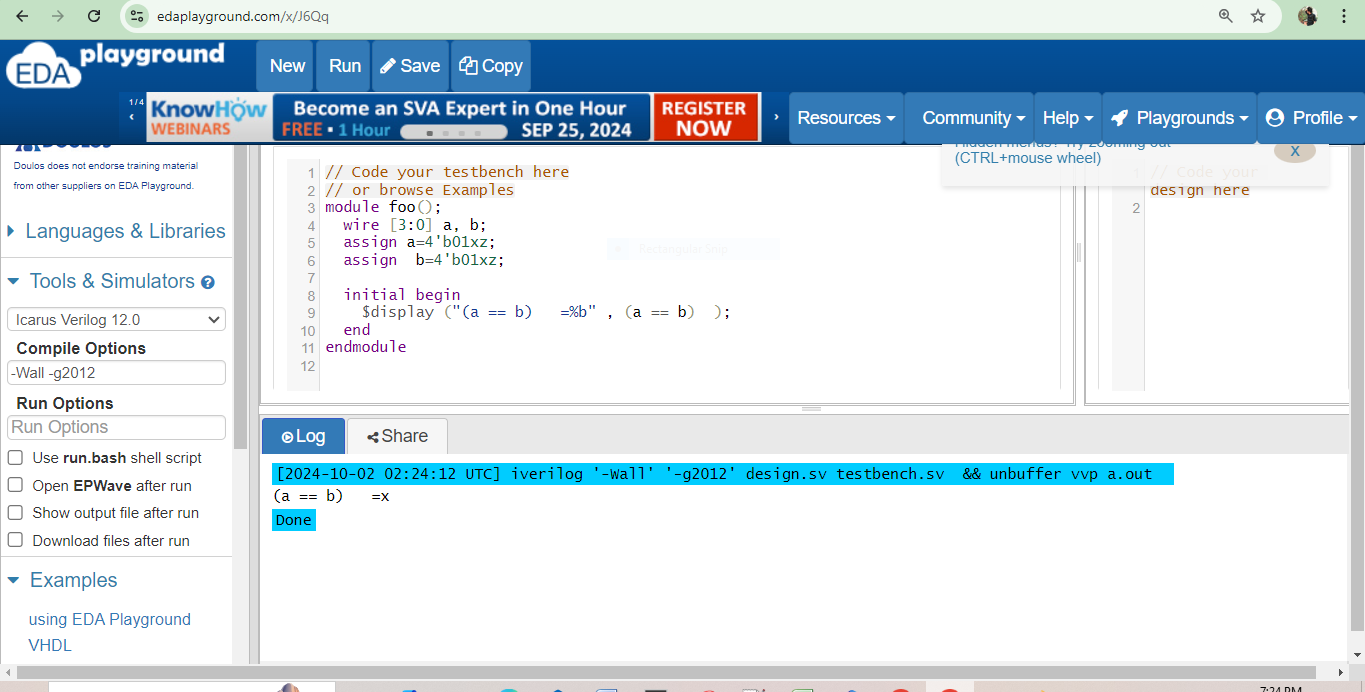


Now I just give the code and output screenshots together ( no code separately).

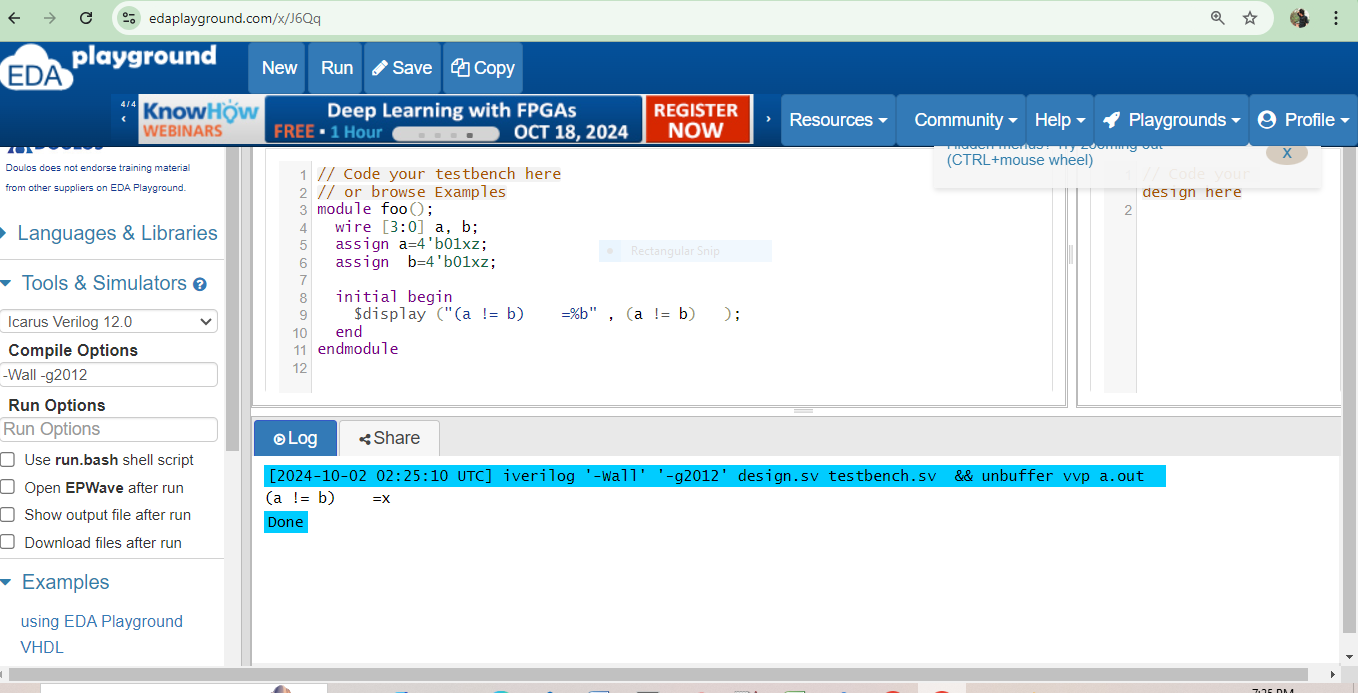
**//for a!==b**



**//for (a == b)**

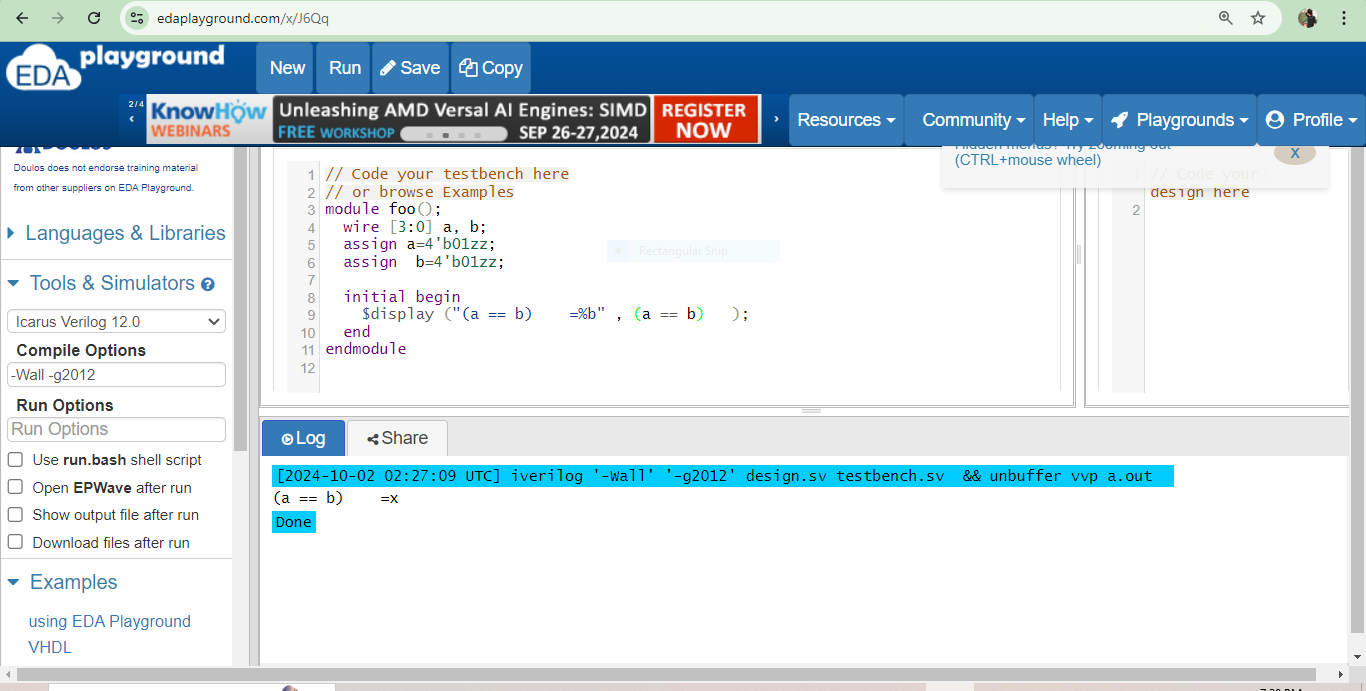


**//for** (a != b)

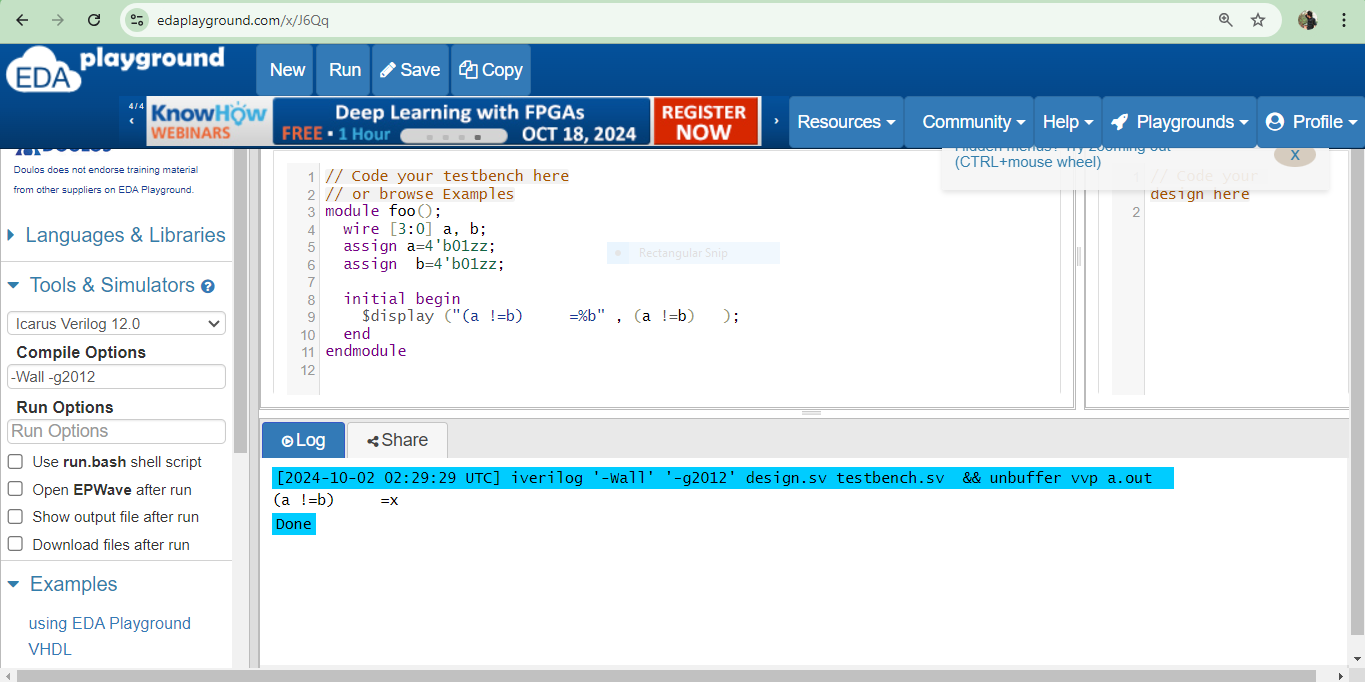


//If a = 4'01zz and b=4'b01zz

**//for (a == b)**



//for (a !=b)



**4.**

module test\_not\_operator();

reg [2:0] A; // Maximum 3-bit value for 'A', including cases with unknowns

initial begin

// Display header

$display("A | !A");

// Test cases for 'A' and the results of !A

// 1-bit cases

A = 1'bx;

$display("1'bx | %b", !A);

A = 1'bz;

$display("1'bz | %b", !A);

// 2-bit cases

A = 2'b1z;

$display("2'b1z | %b", !A);

A = 2'b0z;

$display("2'b0z | %b", !A);

A = 2'bxz;

$display("2'bxz | %b", !A);

// 3-bit cases

A = 3'bxxx;

$display("3'bxxx | %b", !A);

A = 3'b1xx;

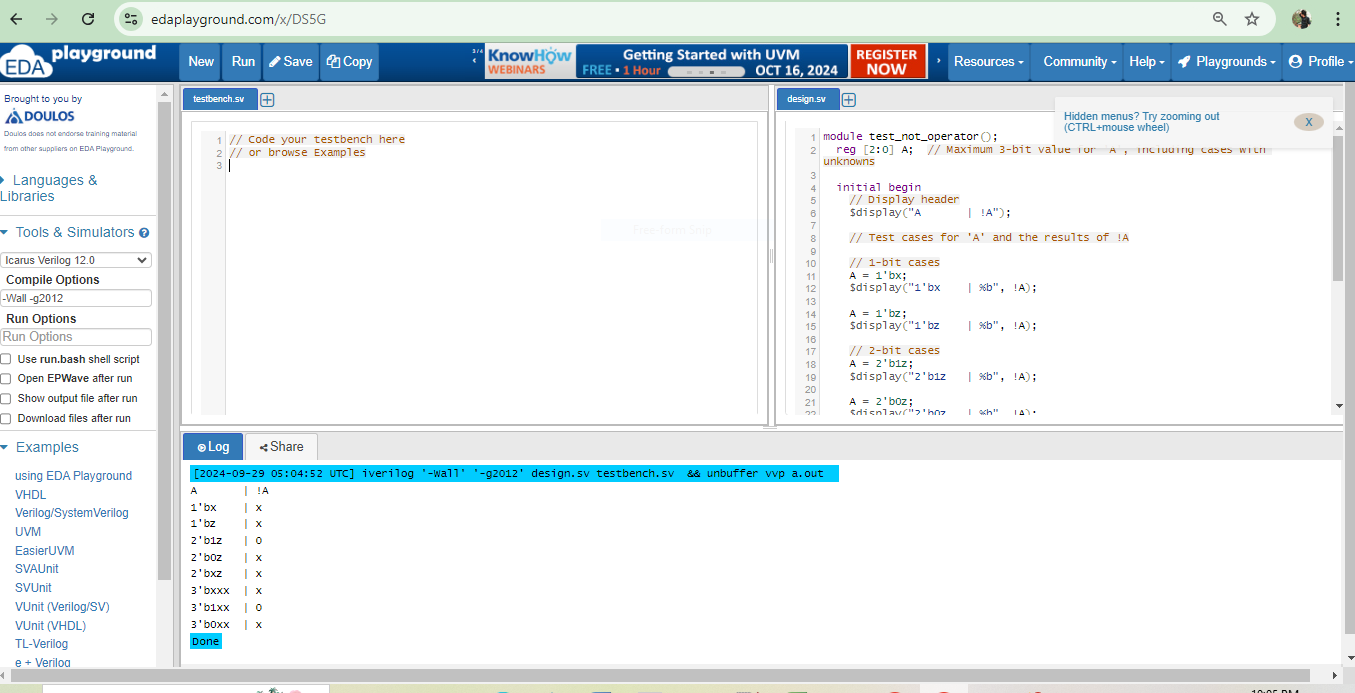
$display("3'b1xx | %b", !A);

A = 3'b0xx;

$display("3'b0xx | %b", !A);

end

endmodule



**5.**

module test\_logical\_operations();

reg A; // 1-bit value for 1'bx

reg [1:0] B; // 2-bit value for 2'bxz

reg [1:0] C; // 2-bit value for 2'b0x

reg [1:0] D; // 2-bit value for 2'b00

reg [1:0] E; // 2-bit value for 2'b1z

reg [3:0] F; // 4-bit value for 4'b01xz

initial begin

// Assign values

A = 1'bx; // 1'bx

B = 2'bxz; // 2'bxz

C = 2'b0x; // 2'b0x

D = 2'b00; // 2'b00

E = 2'b1z; // 2'b1z

F = 4'b01xz; // 4'b01xz

// Evaluate expressions and display results

$display("1'bx && 2'bxz | %b", A && B); // Logical AND

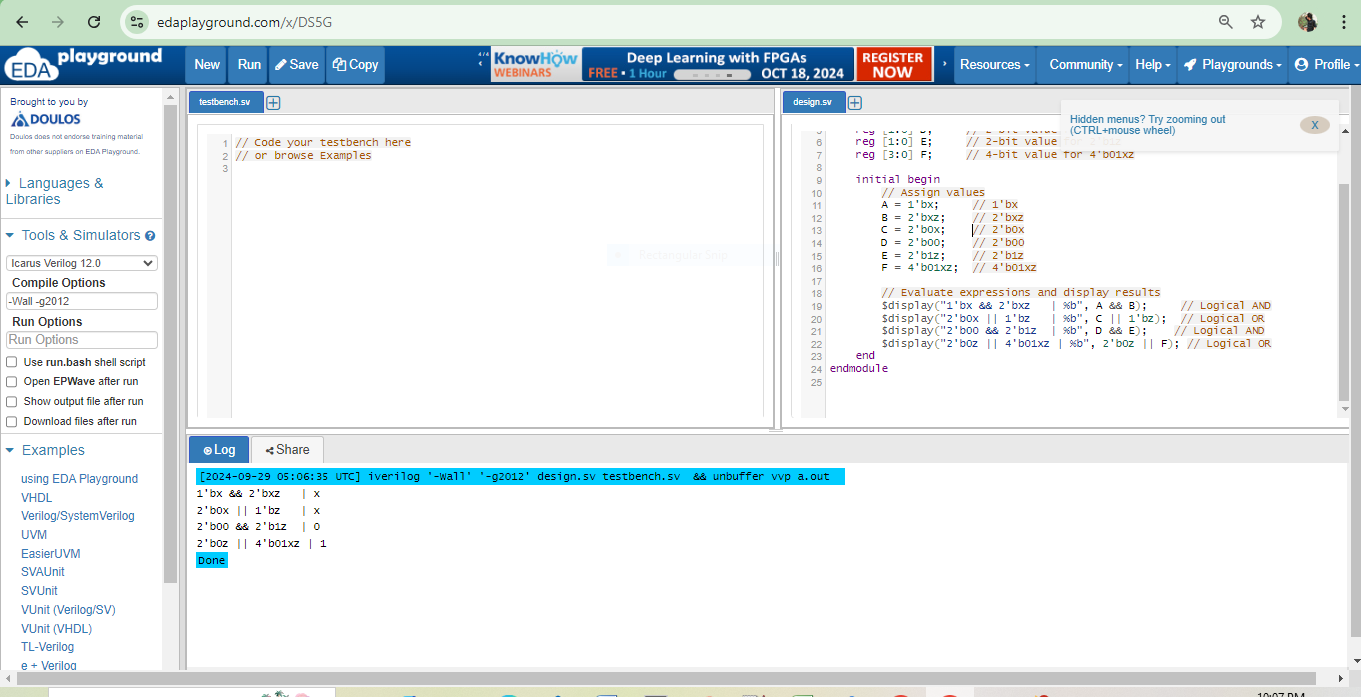
$display("2'b0x || 1'bz | %b", C || 1'bz); // Logical OR

$display("2'b00 && 2'b1z | %b", D && E); // Logical AND

$display("2'b0z || 4'b01xz | %b", 2'b0z || F); // Logical OR

end

endmodule



**6.**

module test\_operations();

reg [3:0] A; // 4-bit variable

reg [3:0] B; // 4-bit variable

reg [1:0] C; // 2-bit variable

initial begin

// Assign values

A = 4'b01xz; // Example value for A with x and z

B = 4'bzx01; // Example value for B with z

C = 2'bz1; // Example value for C with z

// Evaluate expressions and display results

$display("~4'b01xz = %b", ~A); // Bitwise NOT

$display("4'b01xz & 4'bzx01 = %b", A & B); // Bitwise AND

$display("4'b01xz | 4'bzx01 = %b", A | B); // Bitwise OR

$display("4'b01xz ^ 4'bzx01 = %b", A ^ B); // Bitwise XOR

// Note: '~~' is not a standard operation, consider just using '~' for NOT

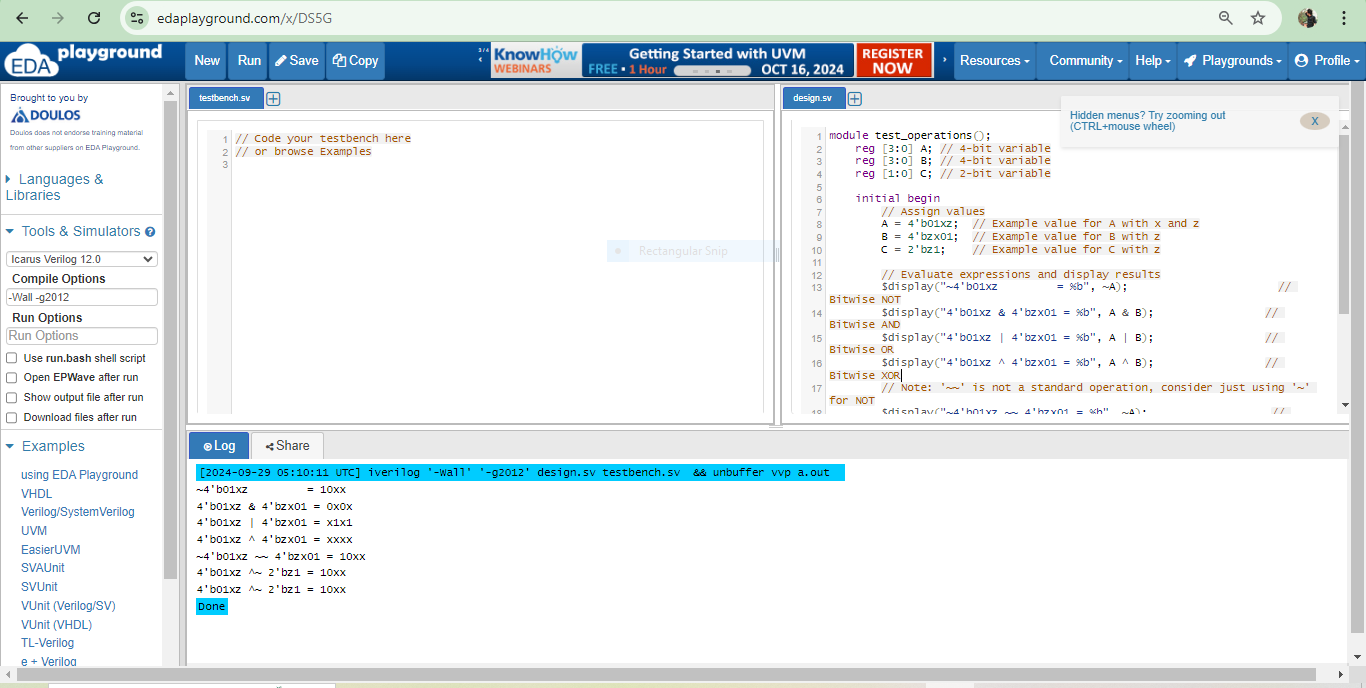
$display("~4'b01xz ~~ 4'bzx01 = %b", ~A); // Bitwise NOT (corrected)

$display("4'b01xz ^~ 2'bz1 = %b", A ^~ {2'b00, C}); // XNOR, padded to match bits

$display("4'b01xz ^~ 2'bz1 = %b", A ^~ {2'b00, C}); // For XNOR with 2'bz1 treated as 4'b?

end

endmodule



**7.**

module test\_operations();

reg [3:0] A; // 4-bit variable

initial begin

// Assign value

A = 4'b01xz; // Example value for A with x and z

// Evaluate expressions and display results

$display("& 4'b01xz = %b", &A); // Bitwise AND reduction

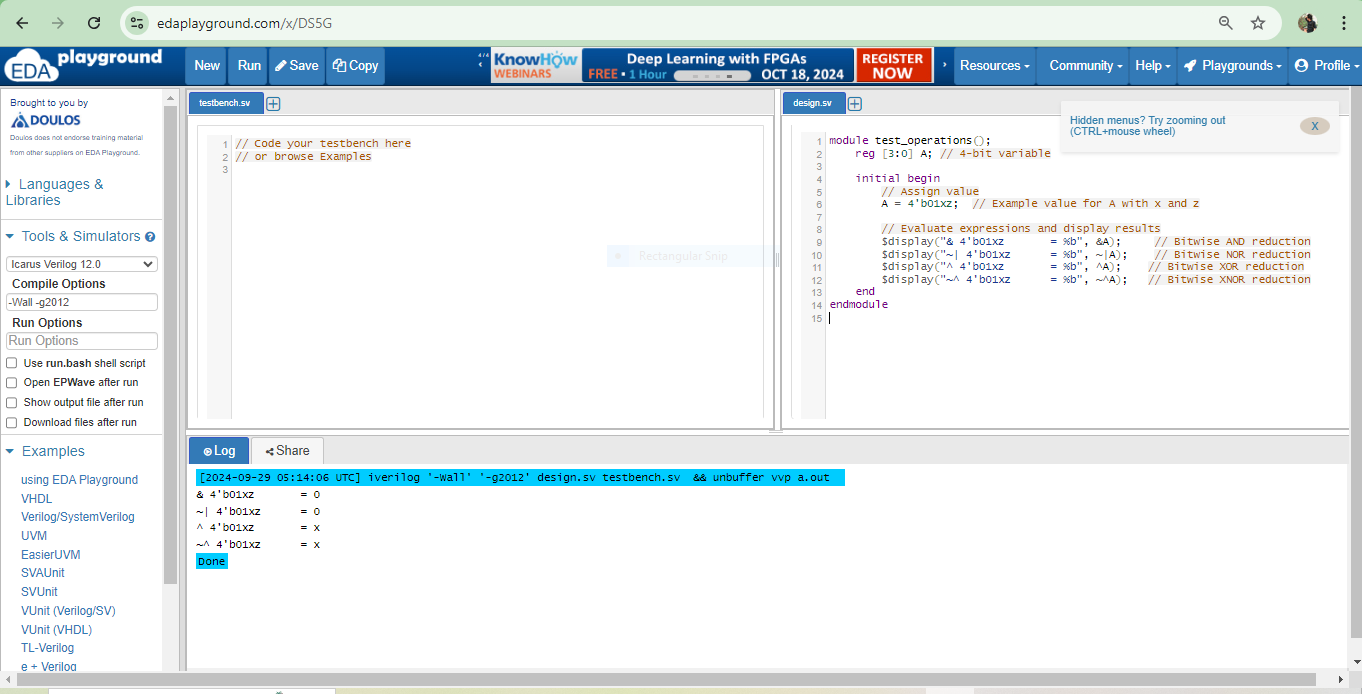
$display("~| 4'b01xz = %b", ~|A); // Bitwise NOR reduction

$display("^ 4'b01xz = %b", ^A); // Bitwise XOR reduction

$display("~^ 4'b01xz = %b", ~^A); // Bitwise XNOR reduction

end

endmodule



**8**.

module test\_bit\_shifting();

reg [3:0] A; // 4-bit variable

reg [1:0] shift\_amount\_right; // 2-bit shift amount for right shift

reg shift\_amount\_left; // 1-bit shift amount for left shift

reg [3:0] result\_left; // Result of left shift

reg [3:0] result\_right; // Result of right shift

initial begin

// Assign value

A = 4'b01xz; // Example value for A with x and z

shift\_amount\_left = 1'bz; // Indeterminate shift amount for left shift

shift\_amount\_right = 2'bxx; // Indeterminate shift amount for right shift

// Evaluate left shift and store result

result\_left = A << shift\_amount\_left;

// Evaluate right shift and store result

result\_right = A >> shift\_amount\_right;

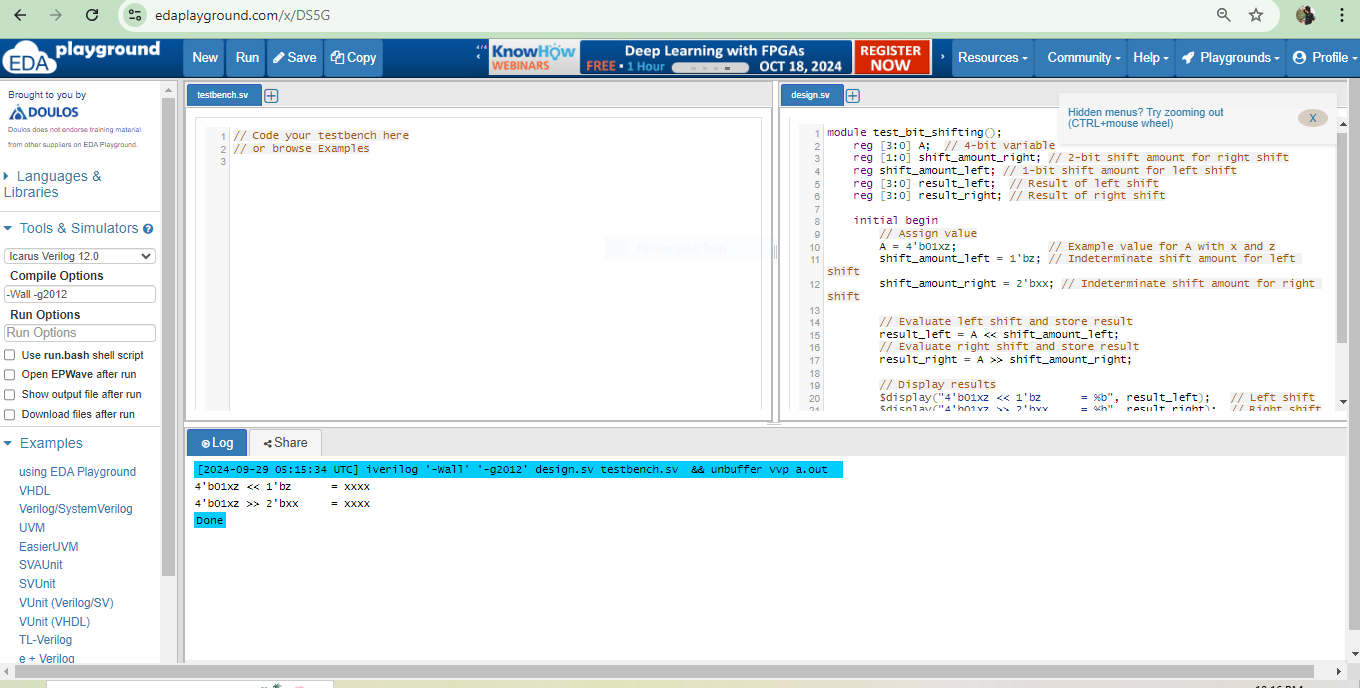
// Display results

$display("4'b01xz << 1'bz = %b", result\_left); // Left shift

$display("4'b01xz >> 2'bxx = %b", result\_right); // Right shift

end

endmodule



**9.**

module test\_conditional();

reg [2:0] B; // 3-bit variable for the second case

reg [1:0] B\_short; // 2-bit variable for the first case

reg [3:0] A; // 4-bit output

initial begin

// Case 1: B = 2'b1x

B\_short = 2'b1x; // Assign value

A = B\_short ? 4'b1100 : 5'b11ZX0; // Conditional assignment

$display("When B = 2'b1x, A = %b", A); // Display result

// Case 2: B = 3'b1xz

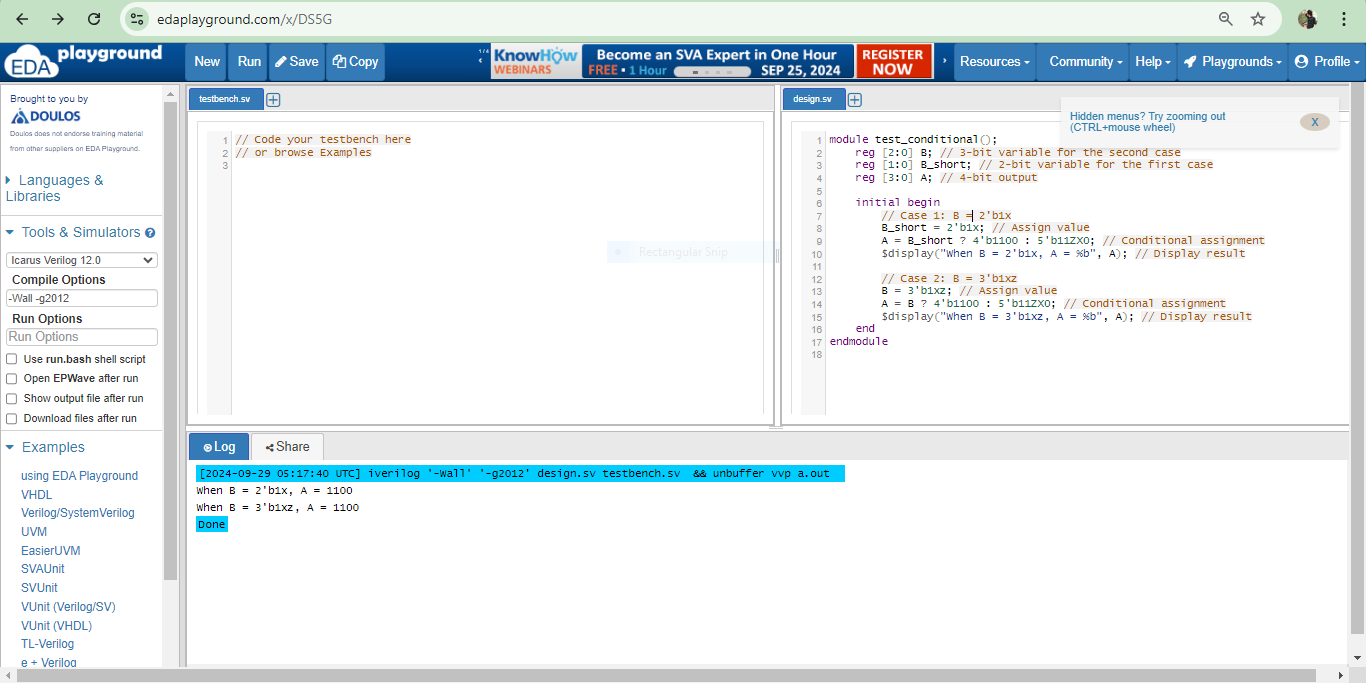
B = 3'b1xz; // Assign value

A = B ? 4'b1100 : 5'b11ZX0; // Conditional assignment

$display("When B = 3'b1xz, A = %b", A); // Display result

end

endmodule



**10.**

module testStrength1();

reg a, b; // Data type declaration for a and b

wire y; // Data type declaration for y

// Define buffer gates with different strengths

bufif1 g1 (y, a, 1'b1); // A strong buffer from a, control is always 1 (enabled)

bufif0 g2 (y, b, 1'b0); // A weak buffer from b, control is always 0 (disabled)

initial begin

a = 1; // Set a to strong 1

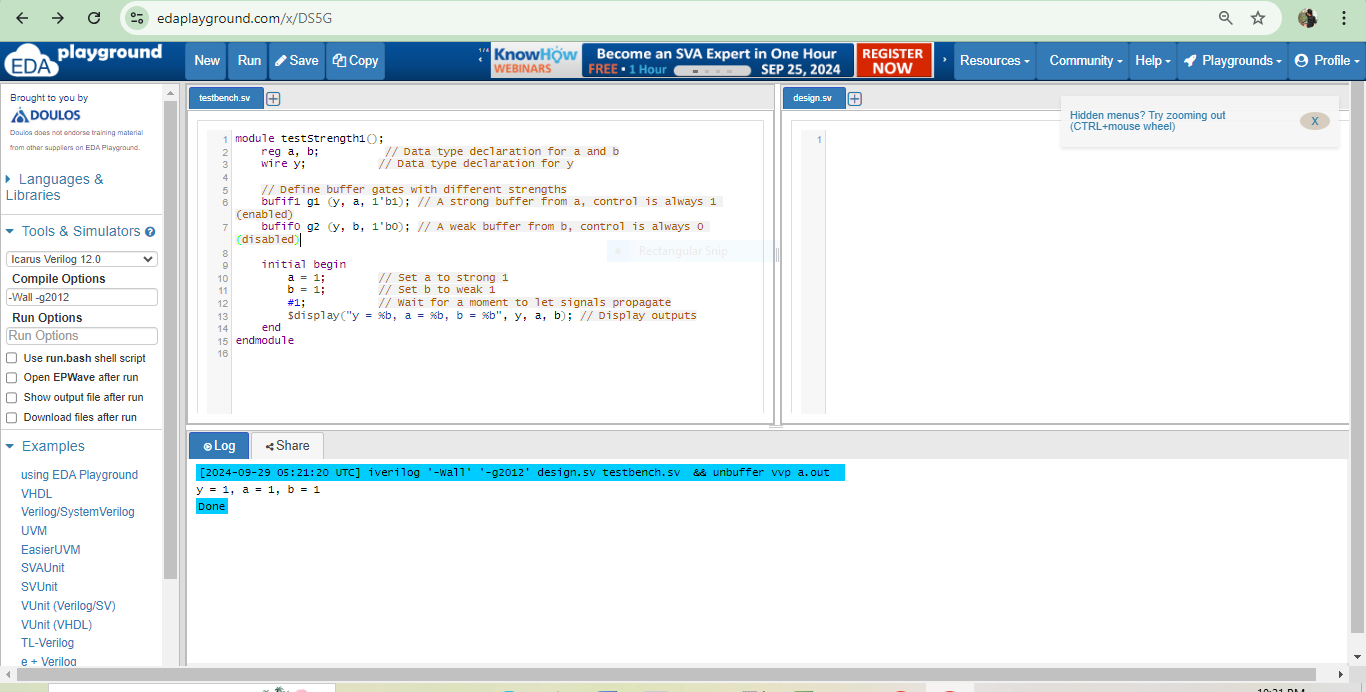
b = 1; // Set b to weak 1

#1; // Wait for a moment to let signals propagate

$display("y = %b, a = %b, b = %b", y, a, b); // Display outputs

end

endmodule



**Explanation:**

Since a is strong and b is weak, y will take the value of a. In this case, the expected output will be y = 1 because the strong driver (from a) will overpower the weak driver (from b).

module testStrength2();

reg i1, i2; // Data type declaration for i1 and i2

reg ctrl; // Control signal

wire y; // Data type declaration for y

// Define buffer gates with the same strength but different inputs

bufif0 g1 (y, i1, ctrl); // Buffer controlled by ctrl

bufif0 g2 (y, i2, ctrl); // Buffer controlled by ctrl

initial begin

ctrl = 1'bx; // Indeterminate state

i1 = 0; // Set i1 to 0

i2 = 1; // Set i2 to 1

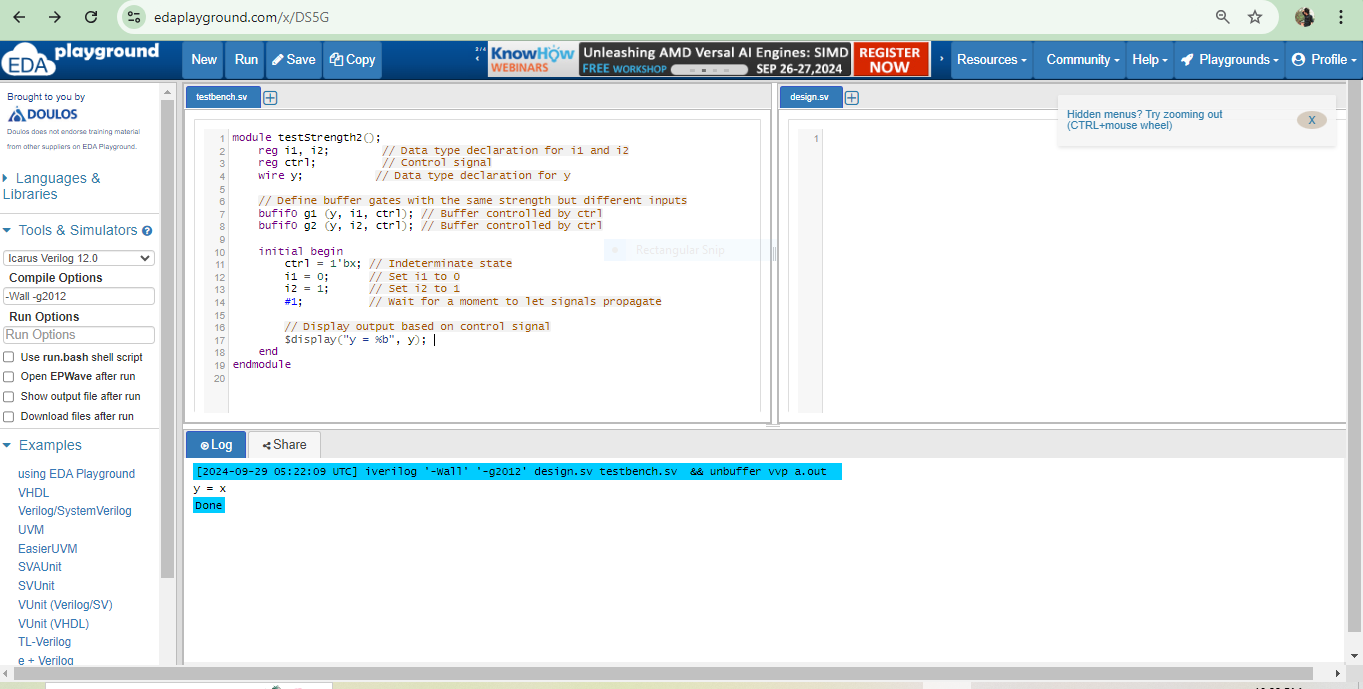
#1; // Wait for a moment to let signals propagate

// Display output based on control signal

$display("y = %b", y);

end

endmodule



**Explanation:**

Because ctrl is initially x, the output y will also be x until ctrl is assigned a definite value. Since both drivers (i1 and i2) are weak (driving when ctrl is 0), the output strength remains weak until ctrl is resolved to either 0 or 1.

**11.**

`timescale 1ns/1ps // Add this line

module fourOneMux(

input i0\_i, i1\_i, i2\_i, i3\_i, // Data inputs

input s0\_i, s1\_i, // Select inputs

output out\_o // Output

);

wire s0\_n, s1\_n; // Wires for NOT gates (inverted select lines)

wire y0, y1, y2, y3; // Wires for AND gate outputs

// Generate inverted select lines

not (s0\_n, s0\_i);

not (s1\_n, s1\_i);

// AND gates for each input and corresponding select lines

and (y0, i0\_i, s1\_n, s0\_n); // y0 = i0 & ~s1 & ~s0

and (y1, i1\_i, s1\_n, s0\_i); // y1 = i1 & ~s1 & s0

and (y2, i2\_i, s1\_i, s0\_n); // y2 = i2 & s1 & ~s0

and (y3, i3\_i, s1\_i, s0\_i); // y3 = i3 & s1 & s0

// OR gate to combine the AND gate outputs

or (out\_o, y0, y1, y2, y3); // out = y0 | y1 | y2 | y3

endmodule

//TB

`timescale 1ns/1ps // Add this line

module fourOneMuxTB;

// Inputs to the MUX

reg i0\_i, i1\_i, i2\_i, i3\_i;

reg s0\_i, s1\_i;

// Output from the MUX

wire out\_o;

// Instantiate the MUX

fourOneMux uut (

.i0\_i(i0\_i),

.i1\_i(i1\_i),

.i2\_i(i2\_i),

.i3\_i(i3\_i),

.s0\_i(s0\_i),

.s1\_i(s1\_i),

.out\_o(out\_o)

);

// Test process

initial begin

// Test Case 1: Select i0 (s1=0, s0=0)

i0\_i = 1; i1\_i = 0; i2\_i = 0; i3\_i = 0; s1\_i = 0; s0\_i = 0;

#10;

// Test Case 2: Select i1 (s1=0, s0=1)

i0\_i = 0; i1\_i = 1; i2\_i = 0; i3\_i = 0; s1\_i = 0; s0\_i = 1;

#10;

// Test Case 3: Select i2 (s1=1, s0=0)

i0\_i = 0; i1\_i = 0; i2\_i = 1; i3\_i = 0; s1\_i = 1; s0\_i = 0;

#10;

// Test Case 4: Select i3 (s1=1, s0=1)

i0\_i = 0; i1\_i = 0; i2\_i = 0; i3\_i = 1; s1\_i = 1; s0\_i = 1;

#10;

// Test Case 5: Random inputs and selections

i0\_i = 1; i1\_i = 0; i2\_i = 1; i3\_i = 1; s1\_i = 1; s0\_i = 0;

#10;

// Finish the simulation

$finish; // Use $finish to end the simulation cleanly

end

// Monitor the output during simulation

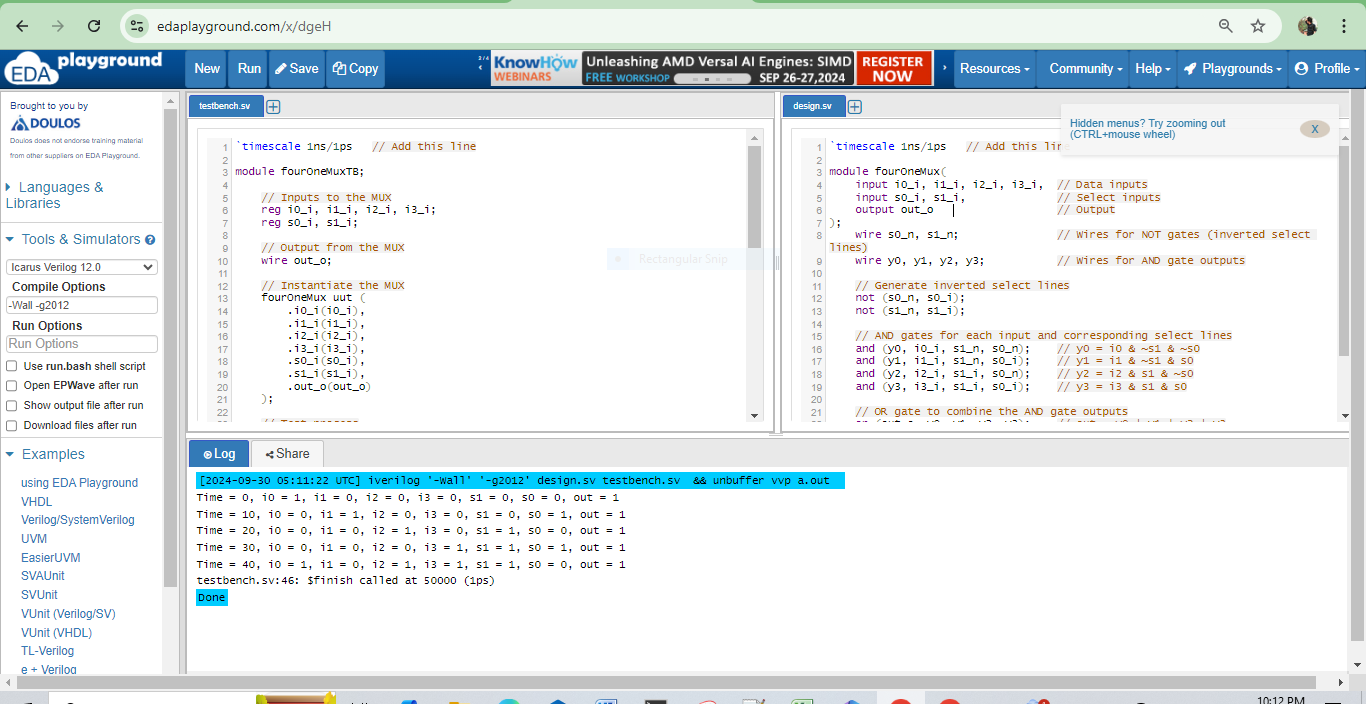
initial begin

$monitor("Time = %0d, i0 = %b, i1 = %b, i2 = %b, i3 = %b, s1 = %b, s0 = %b, out = %b",

$time, i0\_i, i1\_i, i2\_i, i3\_i, s1\_i, s0\_i, out\_o);

end

endmodule



**12.**

`timescale 1ns/1ps

module nor\_gate\_switch (

input A, B, // Inputs A and B

output reg Out // Output of the NOR gate

);

always @(\*) begin

Out = ~(A | B); // NOR operation

end

endmodule

//tb

`timescale 1ns/1ps

module nor\_gate\_tb;

// Inputs

reg A, B;

// Output

wire Out;

// Instantiate the NOR gate

nor\_gate\_switch uut (

.A(A),

.B(B),

.Out(Out)

);

// Test process

initial begin

// Initialize inputs

A = 0; B = 0; // Test case 1

#10;

A = 0; B = 1; // Test case 2

#10;

A = 1; B = 0; // Test case 3

#10;

A = 1; B = 1; // Test case 4

#10;

// Finish simulation

$finish;

end

// Monitor outputs

initial begin

$monitor("Time = %0d, A = %b, B = %b, Out = %b", $time, A, B, Out);

end

endmodule

