**III Exercises**

1)

// or browse Examples

module ReductionOperators();

initial begin

// Bit Wise AND reduction

$display (" & 4'b1001 = %b", (& 4'b1001));

$display (" & 4'bx111 = %b", (& 4'bx111));

$display (" & 4'bz111 = %b", (& 4'bz111));

// Bit Wise NAND reduction

$display (" ~& 4'b1001 = %b", (~& 4'b1001));

$display (" ~& 4'bx001 = %b", (~& 4'bx001));

$display (" ~& 4'bz001 = %b", (~& 4'bz001));

// Bit Wise OR reduction

$display (" | 4'b1001 = %b", (| 4'b1001));

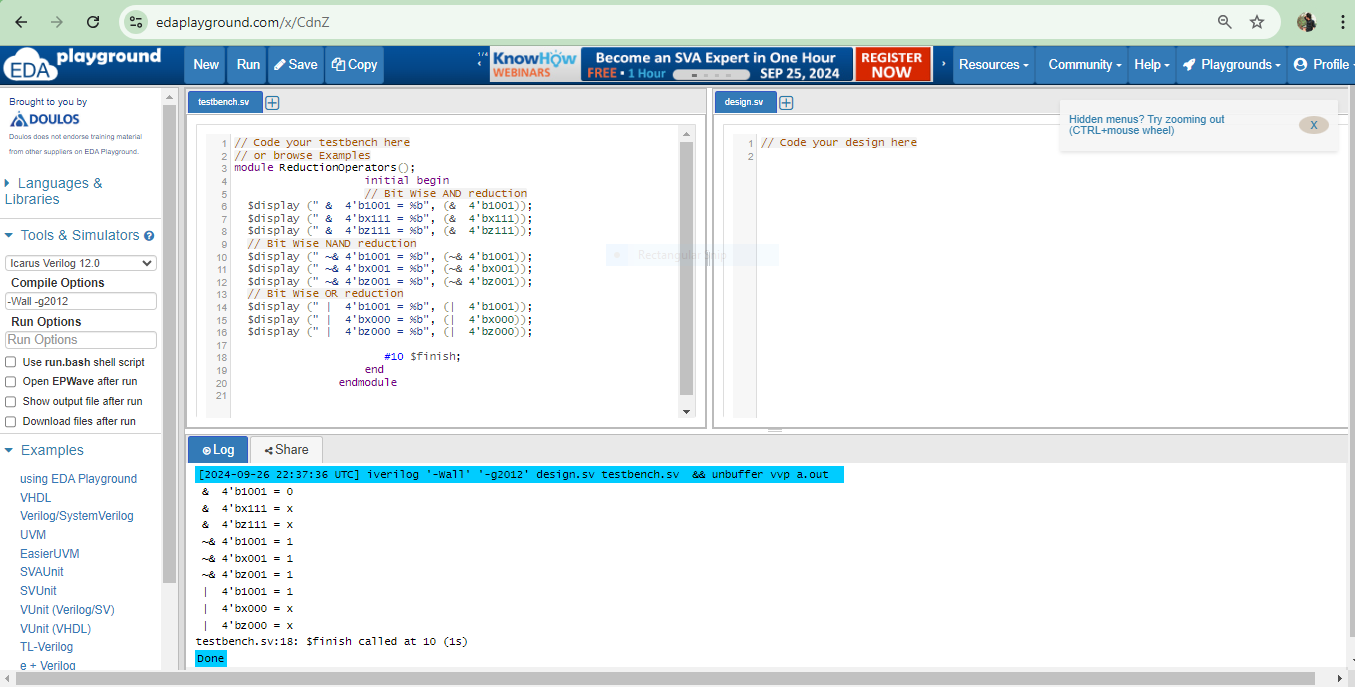
$display (" | 4'bx000 = %b", (| 4'bx000));

$display (" | 4'bz000 = %b", (| 4'bz000));

#10 $finish;

end

endmodule



module ShiftOperators();

initial begin

// Left Shift

$display (" 4'b1001 << 1 = %b", (4'b1001 << 1));

$display (" 4'b10x1 << 1 = %b", (4'b10x1 << 1));

$display (" 4'b10z1 << 1 = %b", (4'b10z1 << 1));

// Right Shift

$display (" 4'b1001 >> 1 = %b", (4'b1001 >> 1));

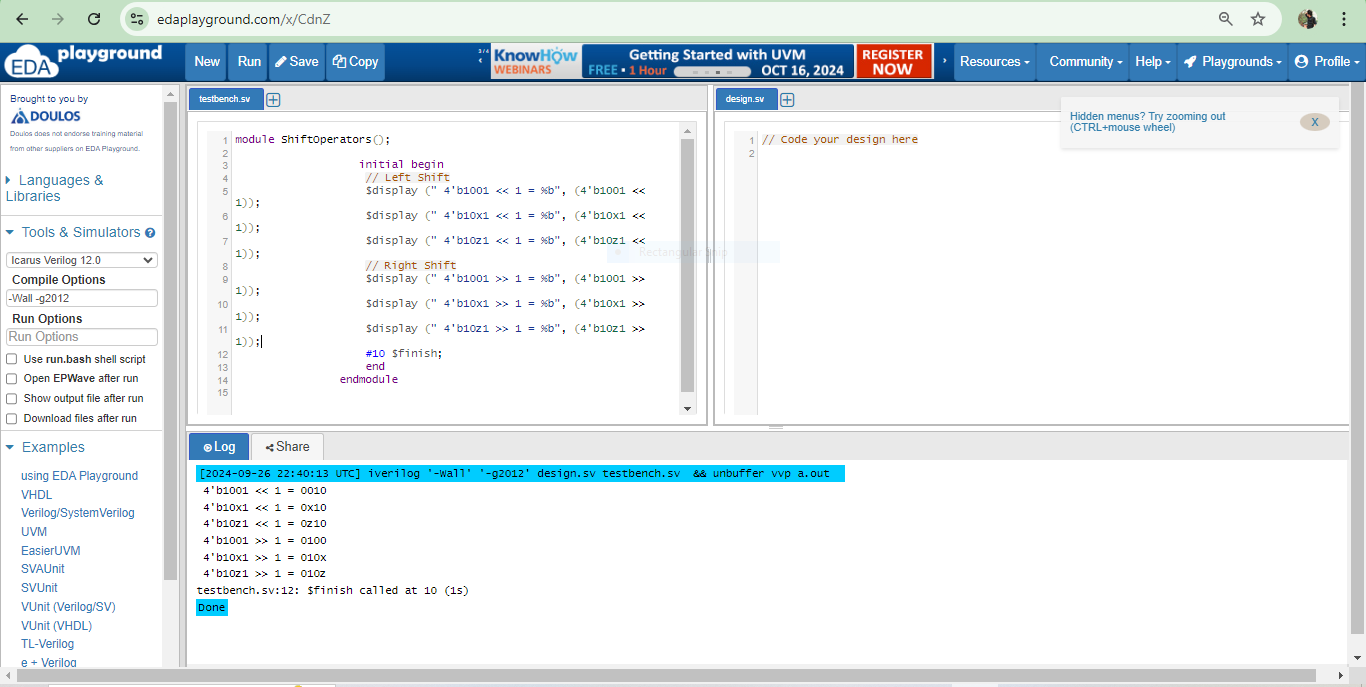
$display (" 4'b10x1 >> 1 = %b", (4'b10x1 >> 1));

$display (" 4'b10z1 >> 1 = %b", (4'b10z1 >> 1));

#10 $finish;

end

endmodule



module ConcatenationOperator();

initial begin

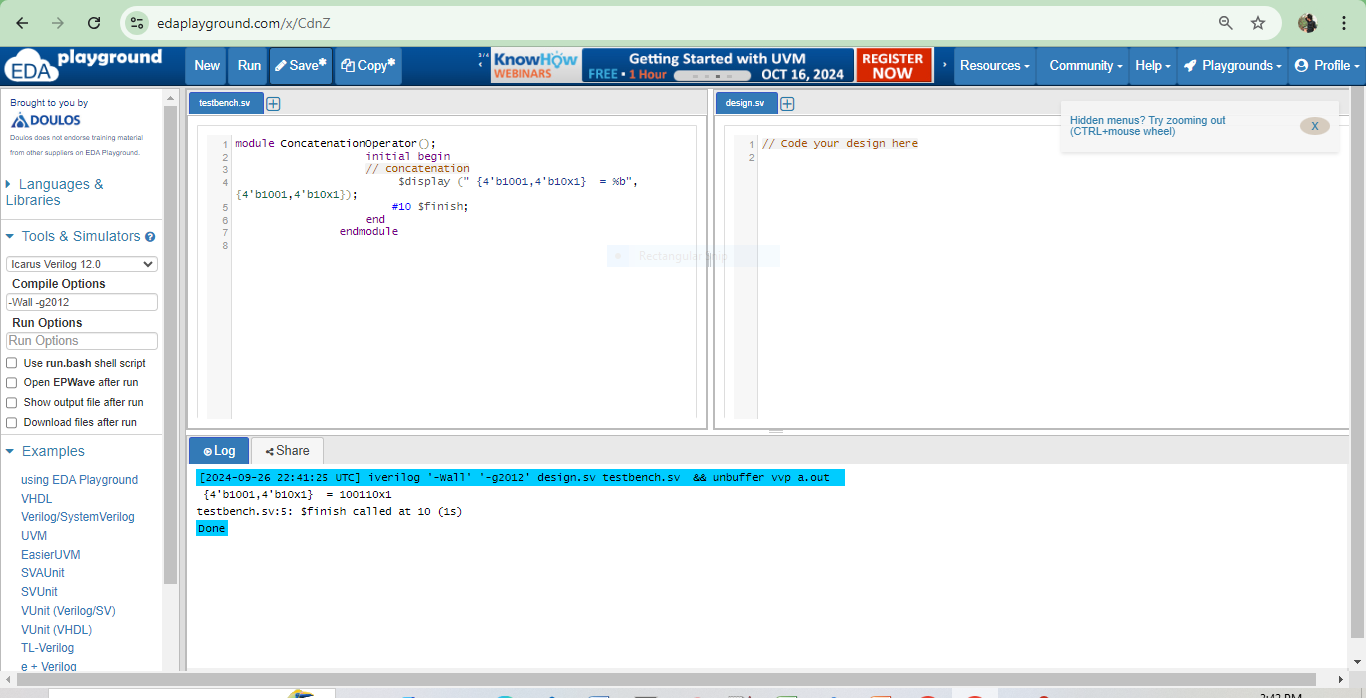
// concatenation

$display (" {4'b1001,4'b10x1} = %b",{4'b1001,4'b10x1});

#10 $finish;

end

endmodule



module ReplicationOperator();

initial begin

// replication

$display (" {4{4'b1001}} = %b", {4{4'b1001}});

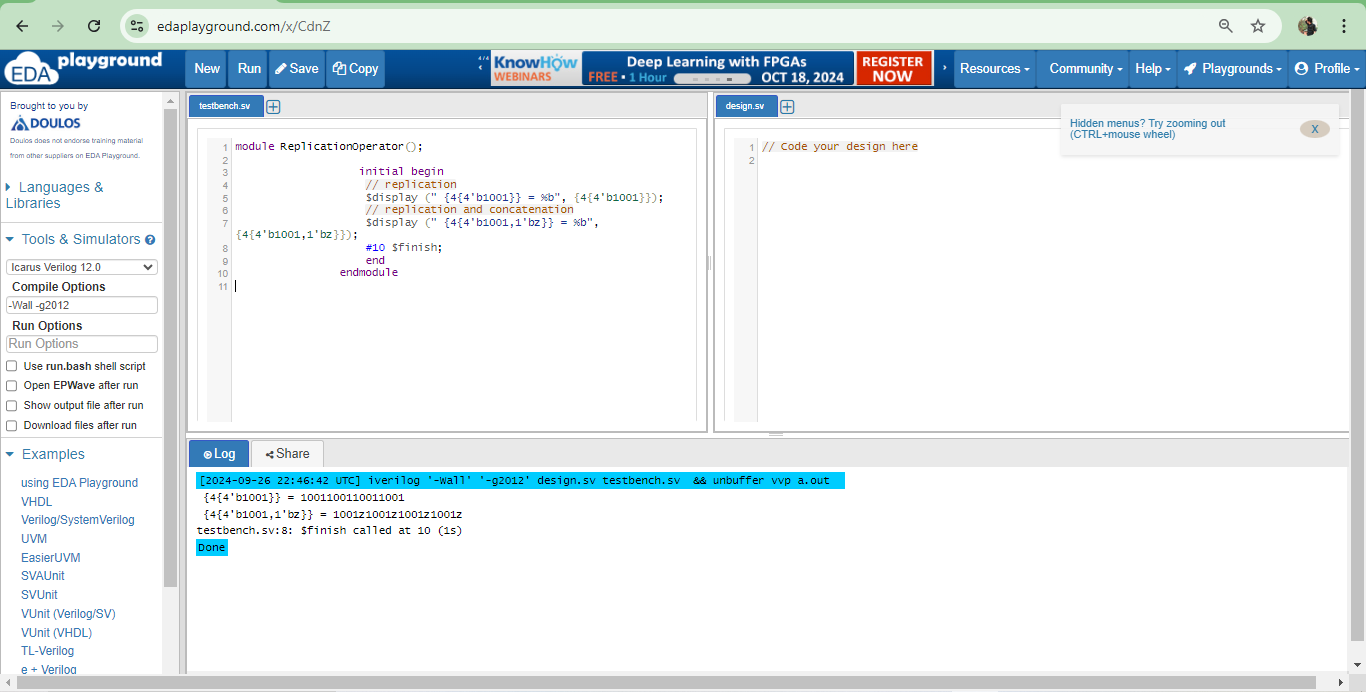
// replication and concatenation

$display (" {4{4'b1001,1'bz}} = %b", {4{4'b1001,1'bz}});

#10 $finish;

end

endmodule



**2)**

//-----------------------------------------------------

`include "addbit.v"

module adder\_hier (

result\_o, // Output of the // adder

carry\_o, // Carry output of // adder

r1\_i, // First input

r2\_i, // Second input

ci\_i // Carry input

);

// Input Port Declarations

input [3:0] r1\_i ;

input [3:0] r2\_i ;

input ci\_i ;

// Output Port Declarations

output [3:0] result\_o ;

output carry\_o ;

// Port Wires

wire [3:0] r1\_i ;

wire [3:0] r2\_i ;

wire ci\_i ;

wire [3:0] result\_o ;

wire carry\_o ;

// Internal variables

wire c1\_w ;

wire c2\_w ;

wire c3\_w ;

// Code Starts Here

addbit u0 (r1\_i[0],r2\_i[0],ci\_i,result\_o[0],c1\_w);

addbit u1 (r1\_i[1],r2\_i[1],c1\_w,result\_o[1],c2\_w);

addbit u2 (r1\_i[2],r2\_i[2],c2\_w,result\_o[2],c3\_w);

addbit u3 (r1\_i[3],r2\_i[3],c3\_w,result\_o[3],carry\_o);

endmodule // End Of Module adder

**module tb();**

reg [3:0] r1\_r, r2\_r;

reg ci\_r;

wire [3:0] result\_w;

wire carry\_w;

// Drive the inputs

initial begin

r1\_r = 0;

r2\_r = 0;

ci\_r = 0;

#10 r1\_r = 10;

#10 r2\_r = 2;

#10 ci\_r = 1;

#10 $display("+----------------------------+");

$finish;

end

// Connect the lower module

adder\_hier U (result\_w, carry\_w, r1\_r, r2\_r, ci\_r);

// Monitor the results of each adder bit

initial begin

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

$display("|r1|r2|ci|u0.sum|u1.sum|u2.sum|u3.sum|");

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

$monitor("|%h |%h |%h |%h |%h |%h |%h |", r1\_r, r2\_r, ci\_r, tb.U.u0.sum\_o, tb.U.u1.sum\_o, tb.U.u2.sum\_o, tb.U.u3.sum\_o);

end

endmodule

module addbit (

input a\_i, // First input bit

input b\_i, // Second input bit

input ci\_i, // Carry input

output sum\_o, // Sum output

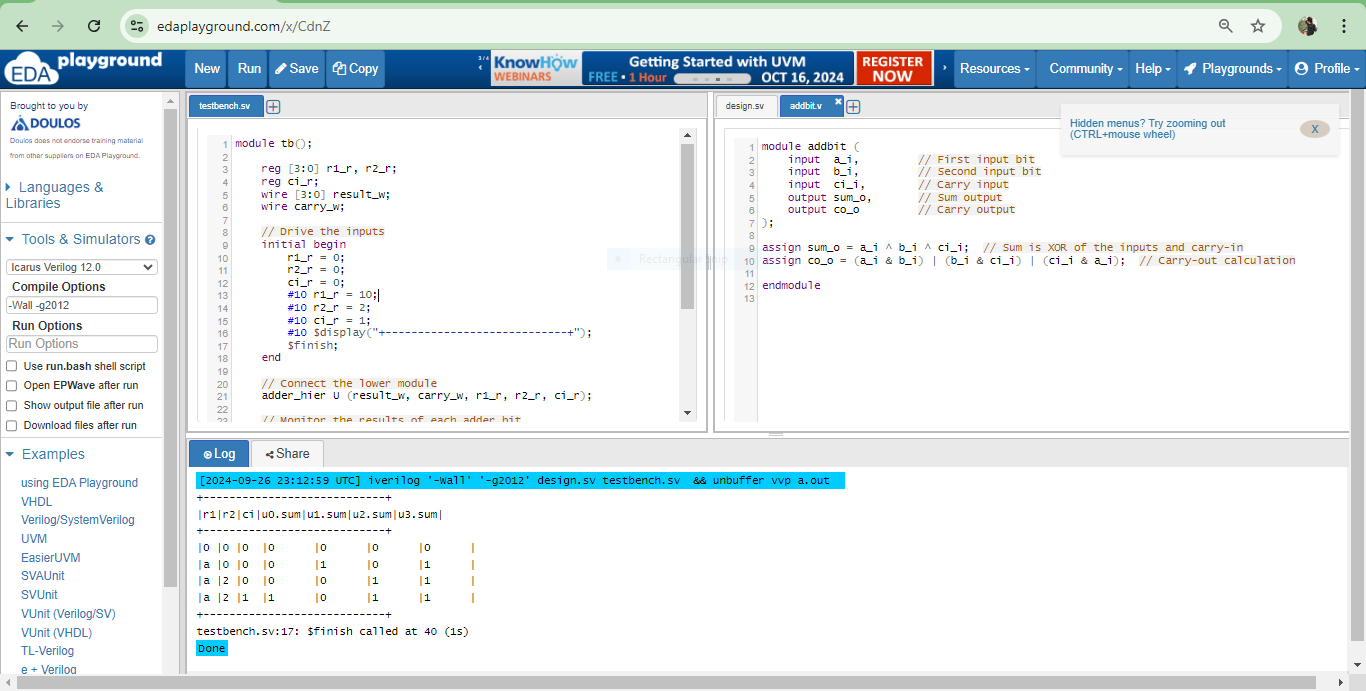
output co\_o // Carry output

);

assign sum\_o = a\_i ^ b\_i ^ ci\_i; // Sum is XOR of the inputs and carry-in

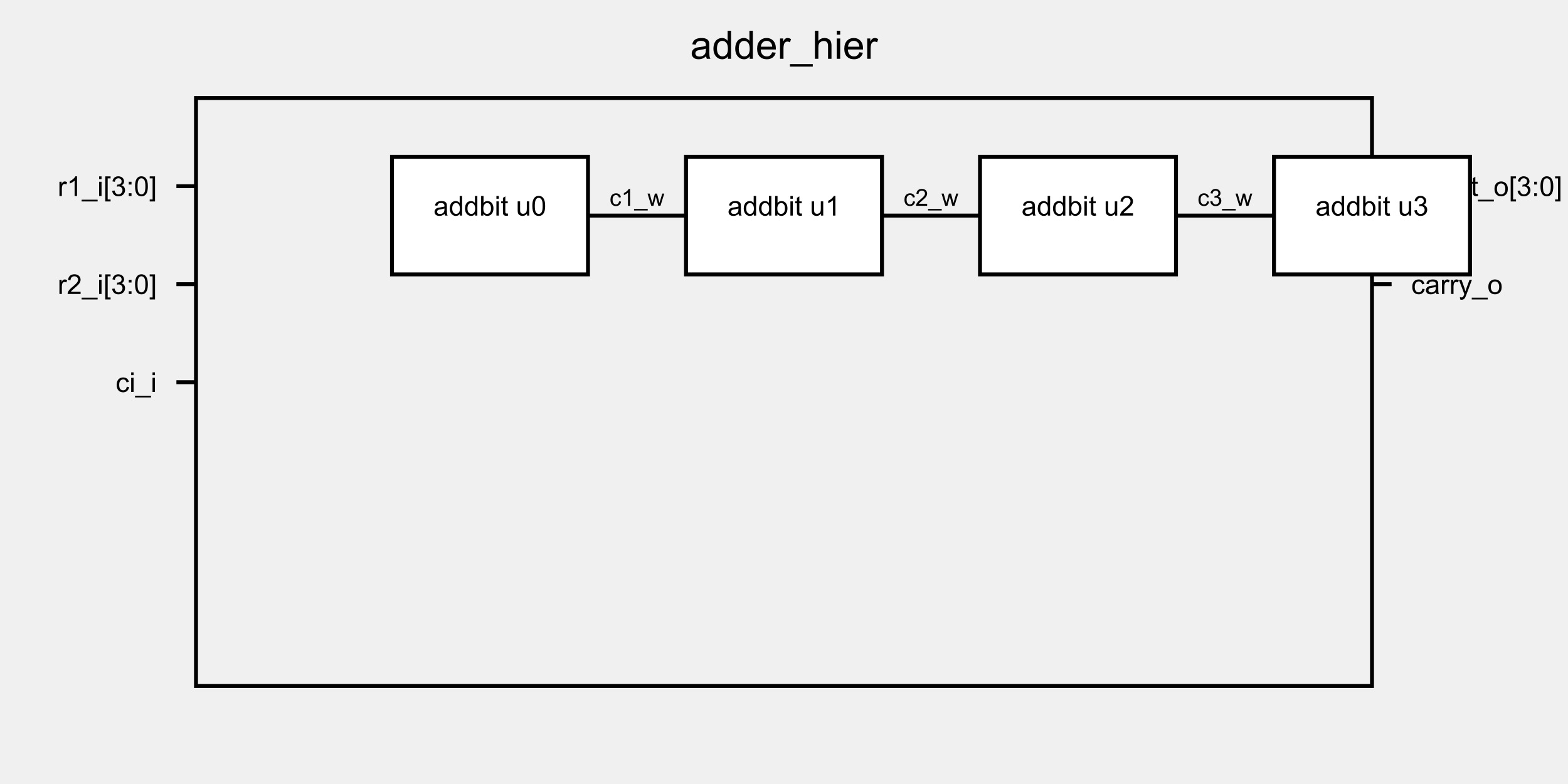
assign co\_o = (a\_i & b\_i) | (b\_i & ci\_i) | (ci\_i & a\_i); // Carry-out calculation

endmodule



**3)**

Based on the module *adder\_hier,* the circuit functional block diagram,



## 4)

## \* DFFSynch Module

module DFFSynch(

d\_i,

rst\_i,

clk\_i,

q\_o

);

input d\_i,rst\_i,clk\_i;

output q\_o;

reg q\_o;

always @(posedge clk\_i)begin

if(rst\_i) q\_o <= 0;

else q\_o <= d\_i;

end

endmodule

//testbr

module tb\_DFFSynch;

// Inputs

reg d\_i;

reg rst\_i;

reg clk\_i;

// Outputs

wire q\_o;

// Instantiate the design under test (DUT)

DFFSynch dut (

.d\_i(d\_i),

.rst\_i(rst\_i),

.clk\_i(clk\_i),

.q\_o(q\_o)

);

// Clock generation

initial begin

clk\_i = 0;

forever #5 clk\_i = ~clk\_i; // 10-unit time period clock

end

// Test procedure

initial begin

// Monitor the changes in inputs and outputs

$monitor("clk\_i=%b, rst\_i=%b, d\_i=%b -> q\_o=%b", clk\_i, rst\_i, d\_i, q\_o);

// Initialize inputs

rst\_i = 1; d\_i = 0; #10; // Test Case 1: Reset is active, output should be 0

rst\_i = 0; d\_i = 1; #10; // Test Case 2: Remove reset, d\_i=1, output should follow d\_i

d\_i = 0; #10; // Test Case 3: d\_i=0, output should follow d\_i

d\_i = 1; #10; // Test Case 4: d\_i=1, output should follow d\_i

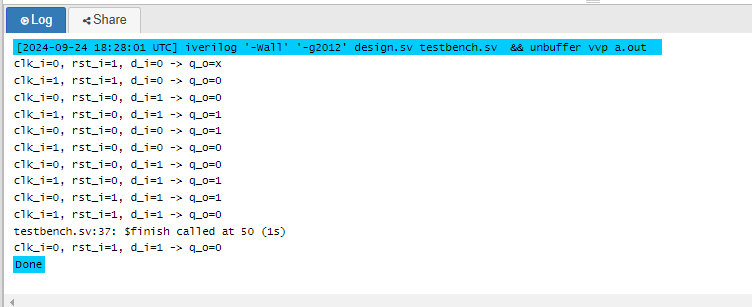
rst\_i = 1; #10; // Test Case 5: Activate reset, output should go to 0

// End the simulation

$finish;

end

endmodule



## \* DFFAsynch Module

module DFFAsynch(

d\_i,

rst\_i,

clk\_i,

q\_o

);

input d\_i,rst\_i,clk\_i;

output q\_o;

reg q\_o;

always @(posedge clk\_i or posedge rst\_i)begin

if(rst\_i) q\_o <= 0;

else q\_o <= d\_i;

end

endmodule

//tr

module tb\_DFFAsynch;

// Inputs

reg d\_i;

reg rst\_i;

reg clk\_i;

// Outputs

wire q\_o;

// Instantiate the design under test (DUT)

DFFAsynch dut (

.d\_i(d\_i),

.rst\_i(rst\_i),

.clk\_i(clk\_i),

.q\_o(q\_o)

);

// Clock generation

initial begin

clk\_i = 0;

forever #5 clk\_i = ~clk\_i; // 10-unit time period clock

end

// Test procedure

initial begin

// Monitor the changes in inputs and outputs

$monitor("clk\_i=%b, rst\_i=%b, d\_i=%b -> q\_o=%b", clk\_i, rst\_i, d\_i, q\_o);

// Initialize inputs

rst\_i = 1; d\_i = 0; #10; // Test Case 1: Reset is active, output should be 0

rst\_i = 0; d\_i = 1; #10; // Test Case 2: Remove reset, d\_i=1, output should follow d\_i

d\_i = 0; #10; // Test Case 3: d\_i=0, output should follow d\_i

d\_i = 1; #10; // Test Case 4: d\_i=1, output should follow d\_i

rst\_i = 1; #10; // Test Case 5: Activate reset, output should go to 0

rst\_i = 0; d\_i = 1; #10; // Test Case 6: After reset, d\_i=1, output should follow d\_i

// Add a delay to see the effect of the clock edge

d\_i = 0; #10; // Test Case 7: d\_i=0, output should follow d\_i on next clock edge

#10; // Wait for the clock edge

d\_i = 1; #10; // Test Case 8: d\_i=1, output should follow d\_i on next clock edge

// End the simulation

$finish;

end

endmodule

