CADD -- JOB ASSIGNMENT:-

**Problem Statement: 8-bit BCD Adder**

CODE

module BCD\_Adder (

input logic [7:0] A,B,

input Cin,

output [7:0]S,

output c

);

wire [3:0]sum0,sum1;

wire x0,x1;

wire [3:0]y,z;

wire cc0,cc1;

assign {x0,sum0}=A[3:0]+B[3:0]+Cin;

assign cc0=(sum0>4'd9)?1'b1:1'b0;

assign y=cc0?sum0+4'd6:sum0;

assign {x1,sum1}=A[7:4]+B[7:4]+cc0;

assign cc1=(sum1 > 4'd9)?1'b1:1'b0;

assign z=cc1?sum1+4'd6:sum1;

assign S={y,z};

assign c=x1|cc1;

endmodule

TESTBENCH

module BCD\_Adder\_tb;

reg [7:0] A, B;

reg Cin;

wire [7:0] S;

wire c;

BCD\_Adder uut (A,B,Cin,S,c);

initial

begin

A = 8'b00000010;B = 8'b00000011;Cin = 0;#10;

A = 8'b00001001;B = 8'b00000010;Cin = 0;#10;

A = 8'b01011000;B = 8'b01001001;Cin = 0;#10;

A = 8'b00000101;B = 8'b00000101;Cin = 1;#10;

A = 8'b10011001;B = 8'b10011001; Cin = 0;#10;

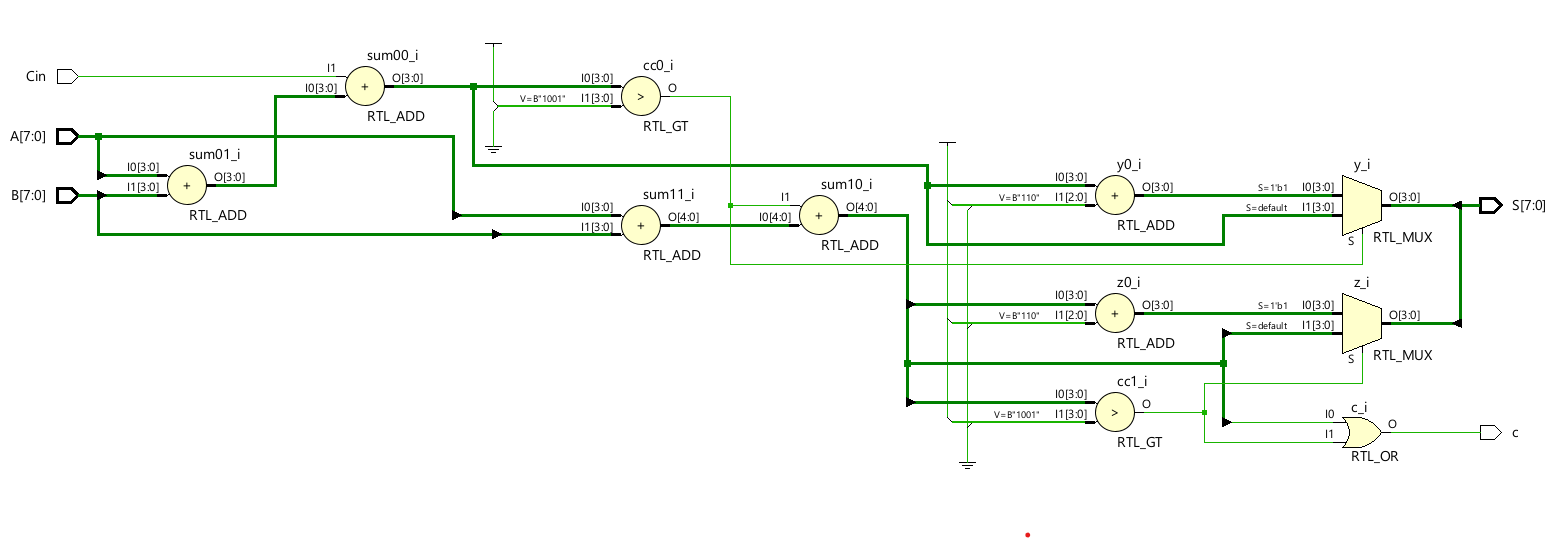
A = 8'b00000000;B = 8'b00000000;Cin = 0;#10;

A = 8'b00100110;B = 8'b00010101;Cin = 1;#10;

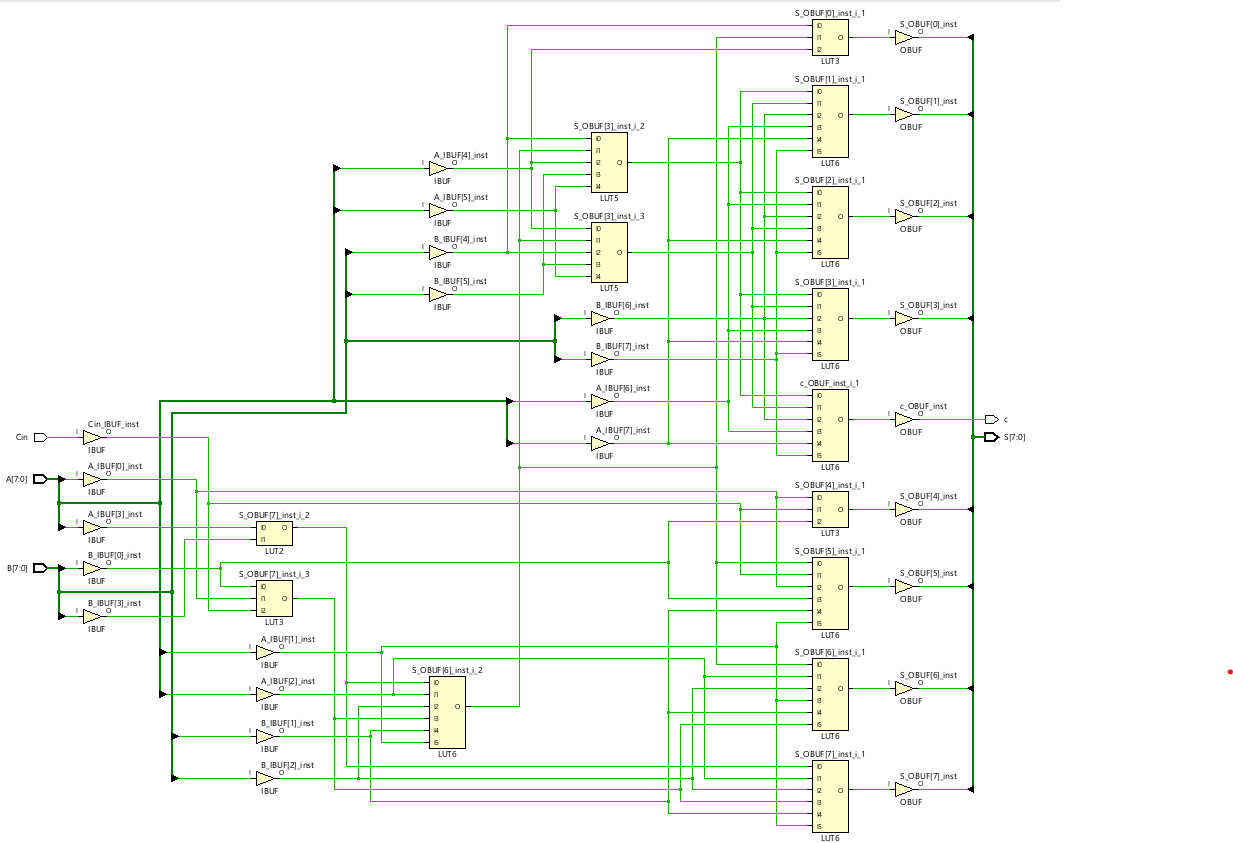
end

endmodule

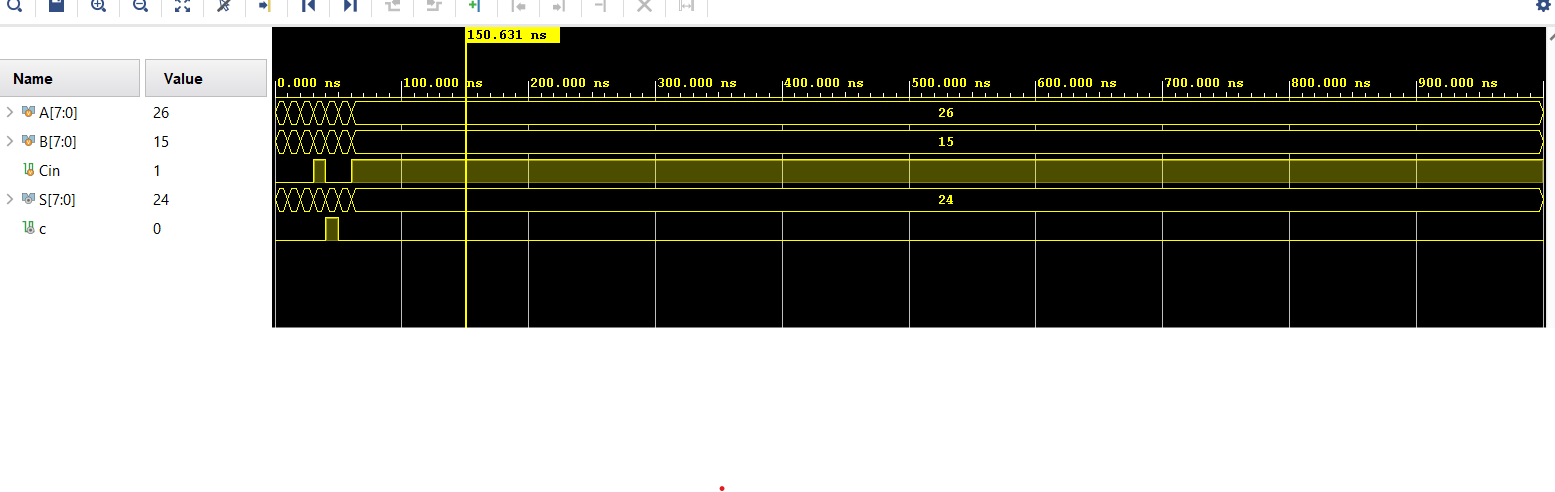
RTL



*Technology schematic*



WAVEFORM



**. Problem Statement: FPGA Flow – Combinational & Sequential circuits**

CODE

module logic1 (

input logic A, B, C, D,

output logic Y);

assign Y = ~A | (A&B&D) | (A&C&(~D)) | (A&(~B)&(~C));

endmodule

TESTBENCH

module logic1\_tb;

reg A,B,C,D;

wire Y;

logic1 dut(A,B,C,D,Y);

initial

begin

#20;A=0;B=0;C=0;D=0;

#10;A=0;B=0;C=0;D=1;

#10;A=0;B=0;C=1;D=0;

#10;A=0;B=0;C=1;D=1;

#10;A=0;B=1;C=0;D=0;

#10;A=0;B=1;C=0;D=1;

#10;A=0;B=1;C=1;D=0;

#10;A=0;B=1;C=1;D=1;

#10;A=1;B=0;C=0;D=0;

#10;A=1;B=0;C=0;D=1;

#10;A=1;B=0;C=1;D=0;

#10;A=1;B=0;C=1;D=1;

#10;A=1;B=1;C=0;D=0;

#10;A=1;B=1;C=0;D=1;

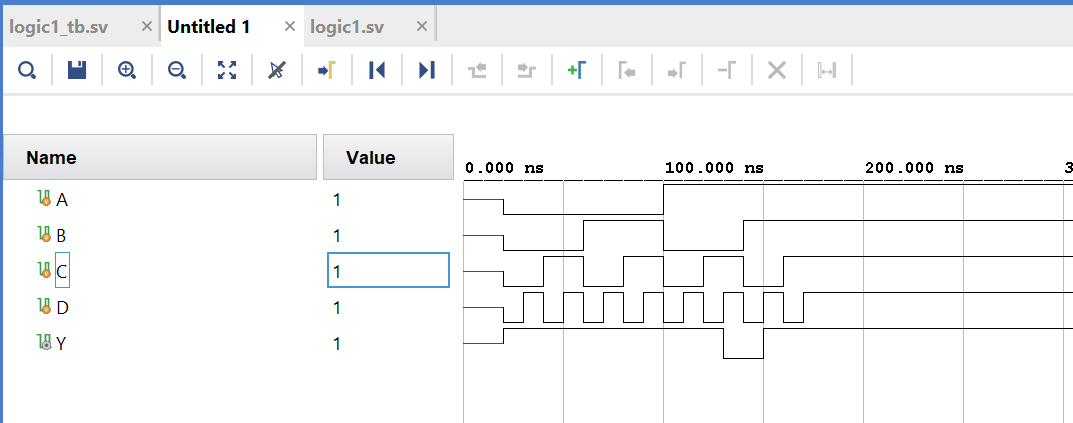
#10;A=1;B=1;C=1;D=0;

#10;A=1;B=1;C=1;D=1;

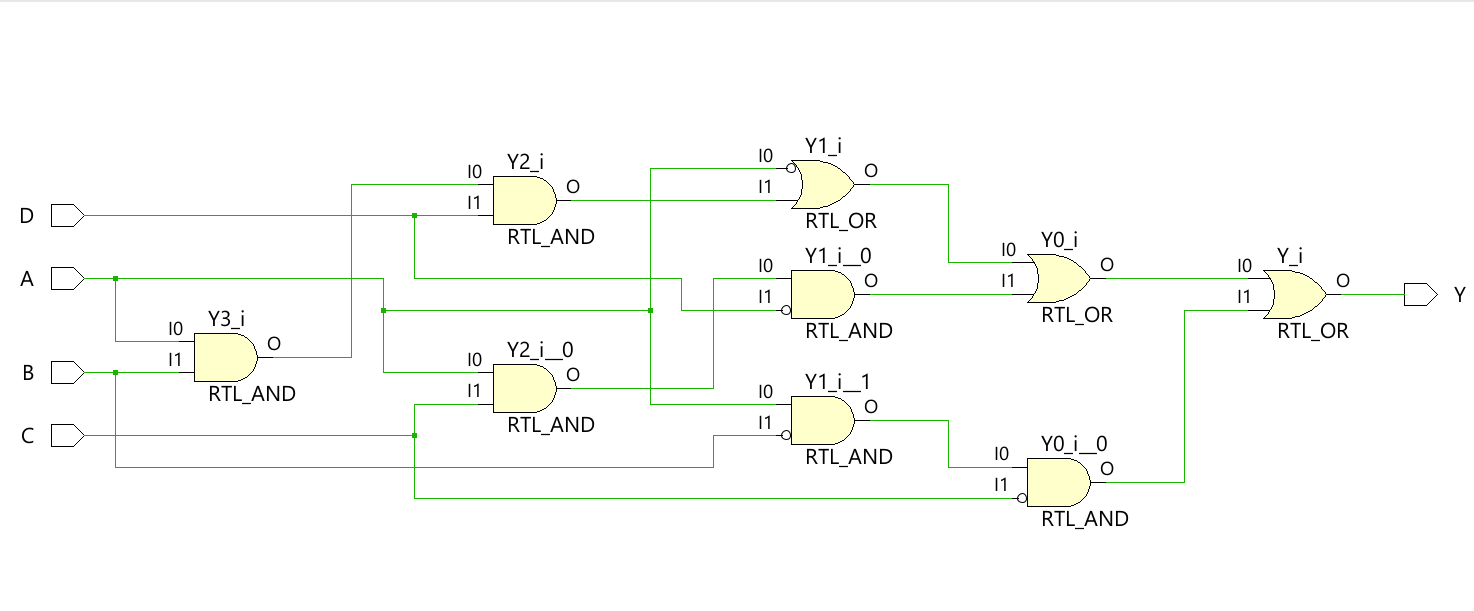
end

endmodule

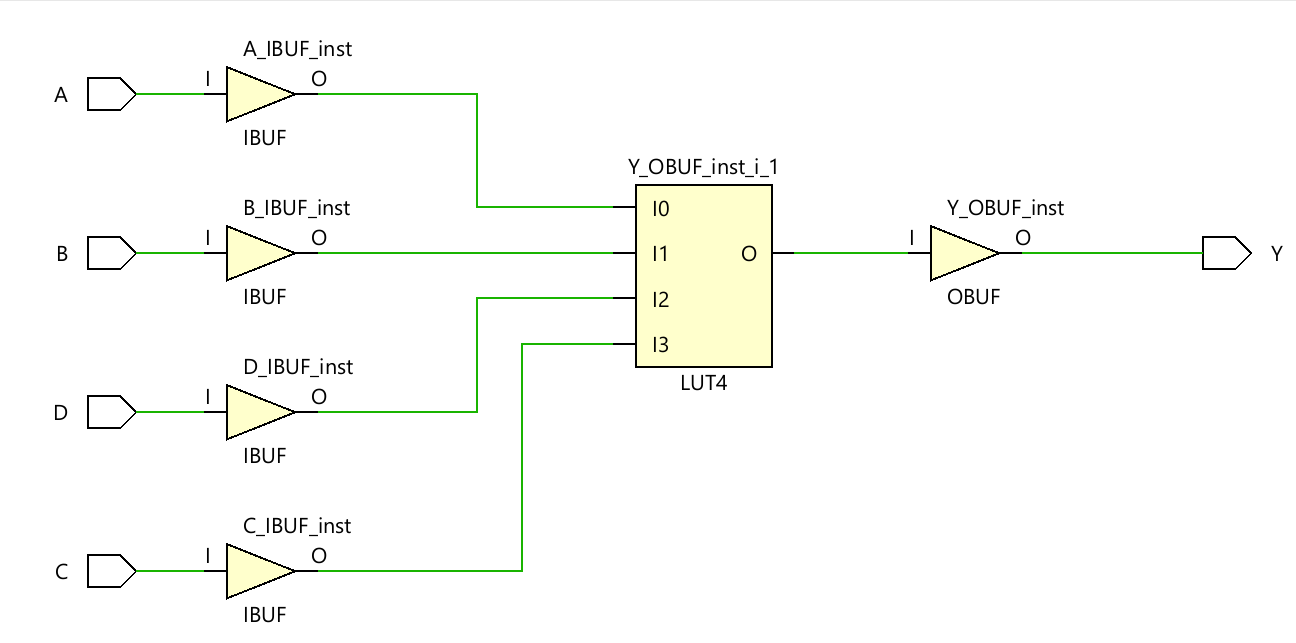
WAVEFORM



RTL



*Technology schematic*



(b) The combinational function

CODE

module LOGIC2(A,B,C,D,E,Y);

input logic A,B,C,D,E;

output logic Y;

assign Y=( A&B&C | A&B&D | A&B&E | A&C&D | A&C&E | ~(A|B|E) | ~B&~C&D | ~B&~C&E | ~B&~D&~E | ~C&~D&~E );

endmodule

TESTBENCH

module LOGIC2\_tb;

reg A,B,C,D,E;

wire Y;

LOGIC2 dut(A,B,C,D,E,Y);

initial

begin

#10;A=0;B=0;C=0;D=0;E=0;

#10;A=0;B=0;C=0;D=0;E=1;

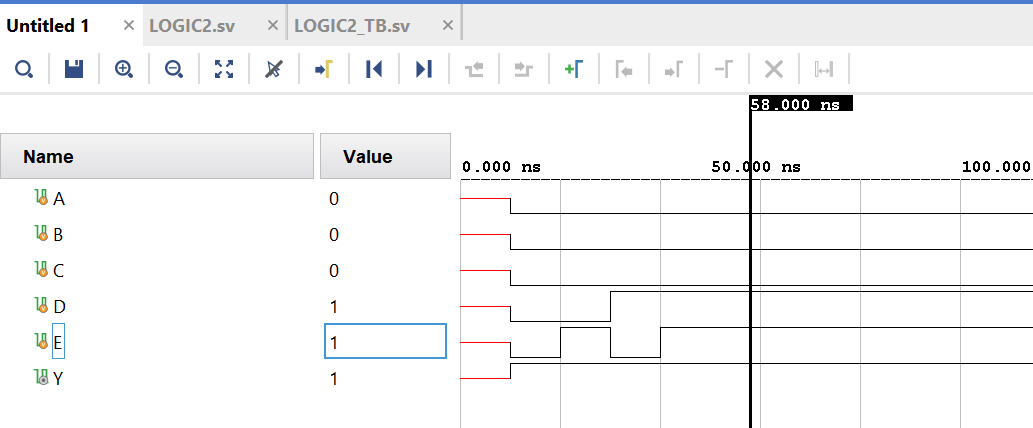
#10;A=0;B=0;C=0;D=1;E=0;

#10;A=0;B=0;C=0;D=1;E=1;

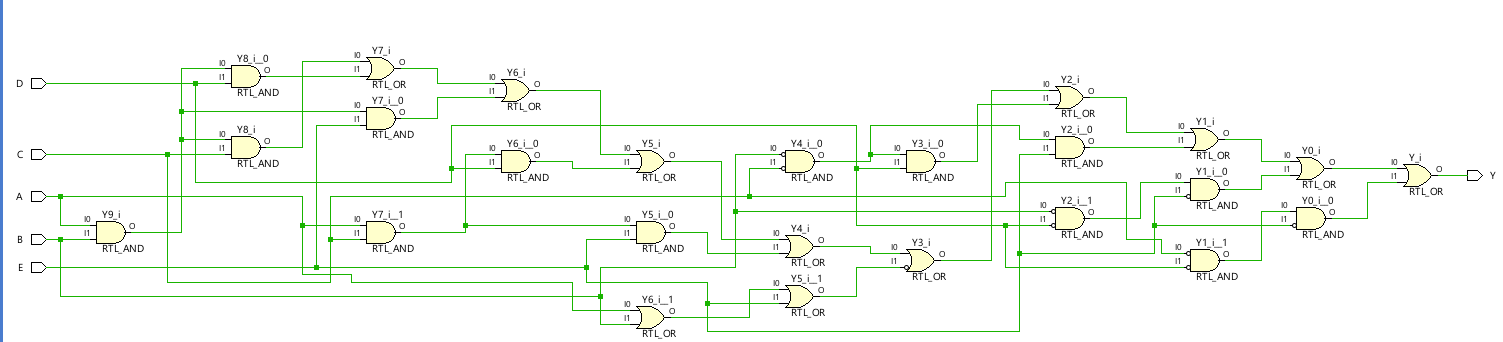
end

endmodule

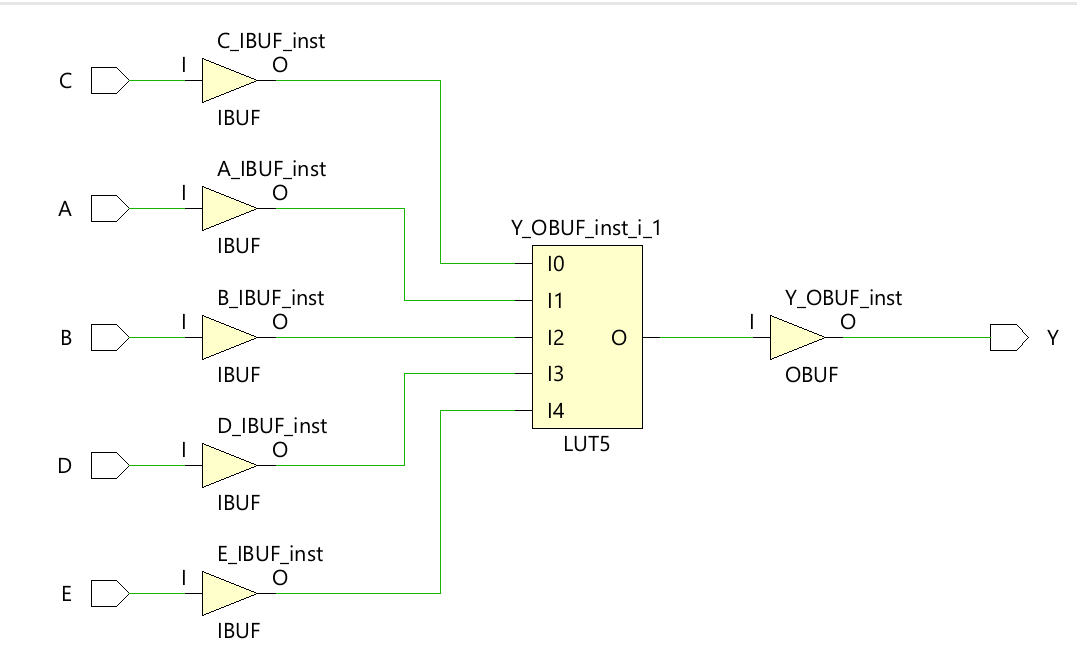
WAVEFORM



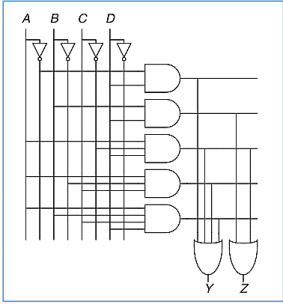
RTL



*Technology schematic*



The two-output function



CODE

module logic3(A,B,C,D,Y,Z);

input logic A,B,C,D;

output logic Y,Z;

assign Y= ( ~A&D | A&~C&D | A&~B&C | A&B&C&D );

assign Z= ( B&D | A&~C&D );

endmodule

TESTBENCH

module logic3\_tb;

reg A,B,C,D;

wire Y,Z;

logic3 dut(A,B,C,D,Y,Z);

initial

begin

#20;A=0;B=0;C=0;D=0;

#10;A=0;B=0;C=0;D=1;

#10;A=0;B=0;C=1;D=0;

#10;A=0;B=0;C=1;D=1;

#10;A=0;B=1;C=0;D=0;

#10;A=0;B=1;C=0;D=1;

#10;A=0;B=1;C=1;D=0;

#10;A=0;B=1;C=1;D=1;

#10;A=1;B=0;C=0;D=0;

#10;A=1;B=0;C=0;D=1;

#10;A=1;B=0;C=1;D=0;

#10;A=1;B=0;C=1;D=1;

#10;A=1;B=1;C=0;D=0;

#10;A=1;B=1;C=0;D=1;

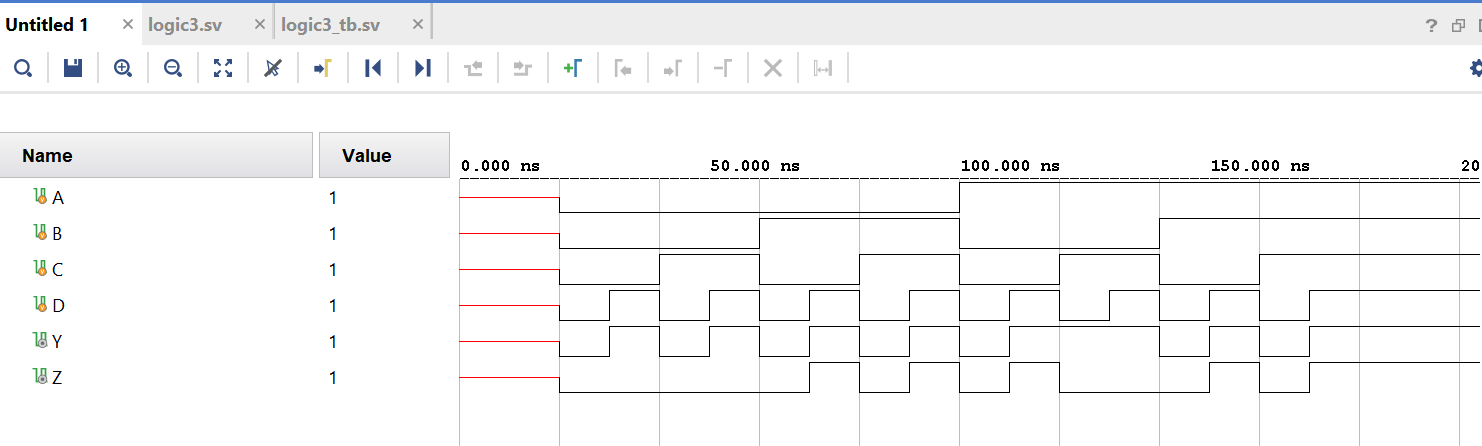
#10;A=1;B=1;C=1;D=0;

#10;A=1;B=1;C=1;D=1;

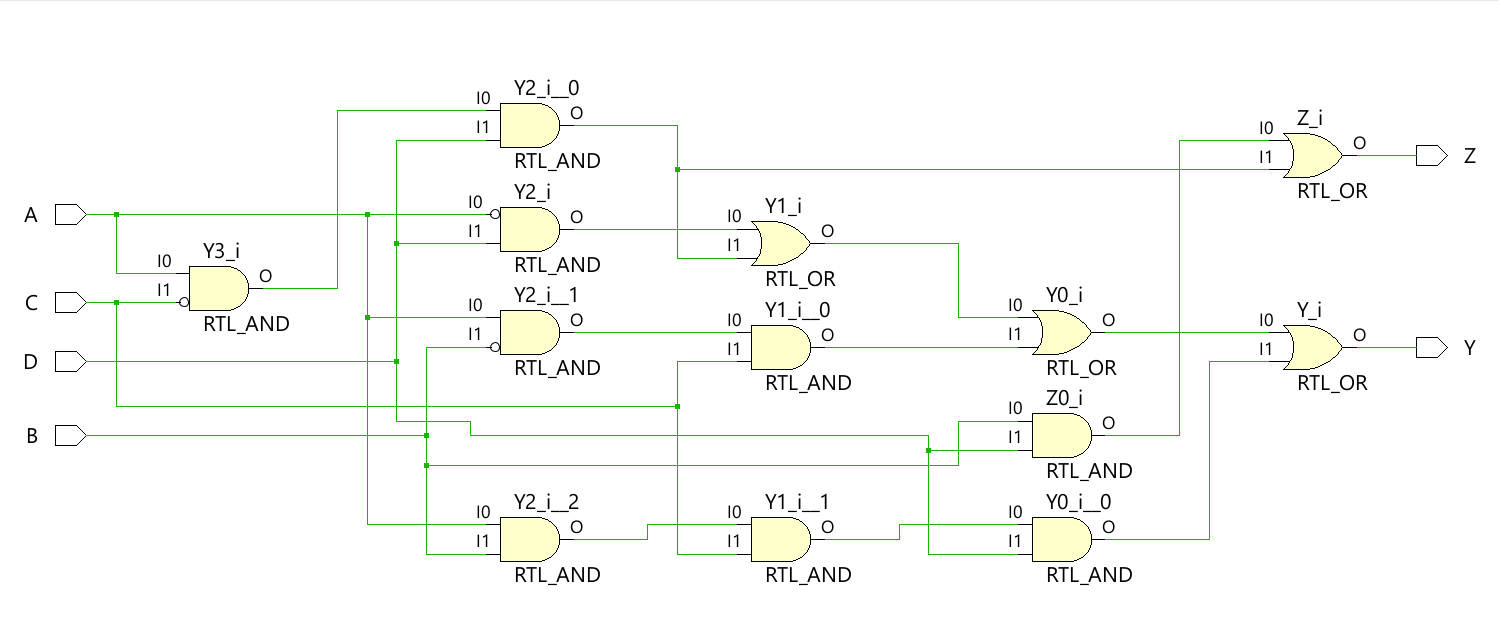
end

endmodule

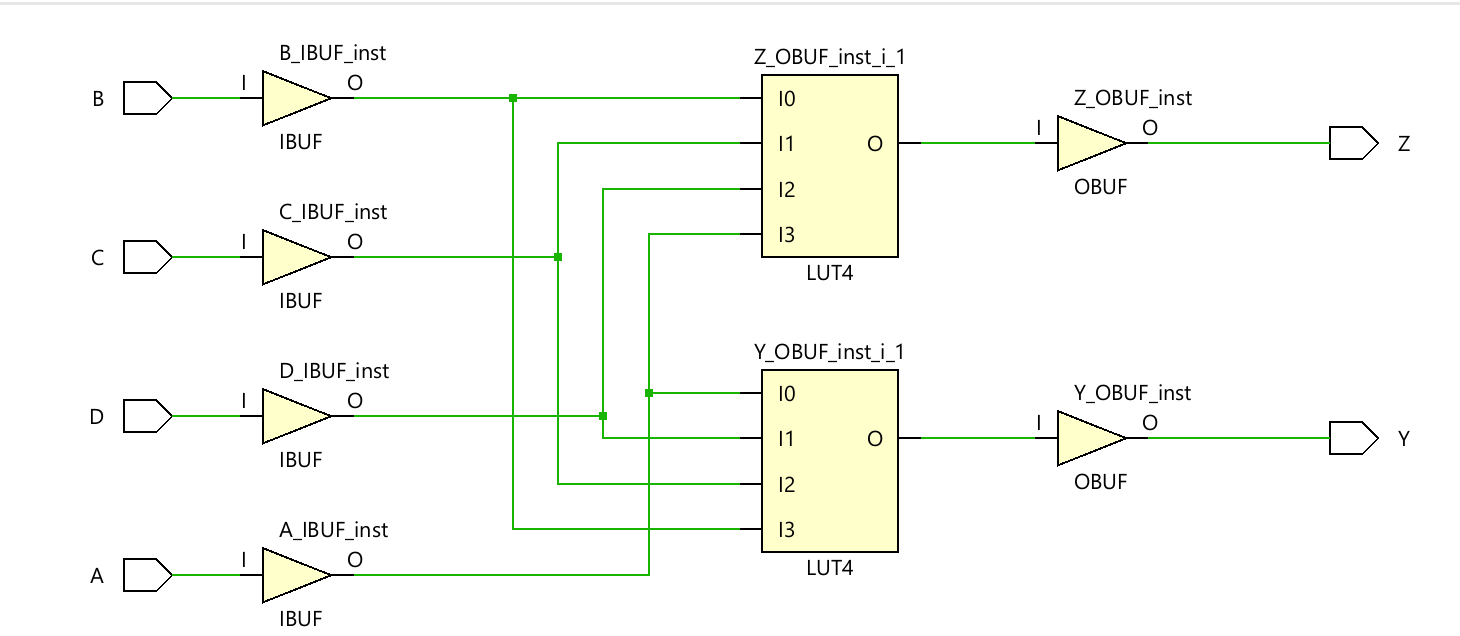
WAVEFORM



RTL



Technology schematic



.

CODE

module PIME(

input logic [3:0] A,

output logic P,D

);

assign P = ((!A[3])&(!A[2])&A[1]&(!A[0]))|((!A[3])&(!A[2])&A[1]&A[0])|((!A[3])&A[2]&(!A[1])&A[0])|((!A[3])&A[2]&A[1]&A[0])|(A[3]&(!A[2])&A[1]&A[0])|(A[3]&A[2]&(!A[1])&A[0]);

assign D=((!A[3])&(!A[2])&(!A[1])&(!A[0]))|((!A[3])&(!A[2])&A[1]&A[0])|((!A[3])&A[2]&A[1]&(!A[0]))|(A[3]&(!A[2])&(!A[1])&A[0])|(A[3]&A[2]&(!A[1])&(!A[0]))|(A[3]&A[2]&A[1]&A[0]);

endmodule

*TESTBENCH*

*module PIME\_tb;*

*reg [3:0] A;*

*wire P, D;*

*PIME dut (A,P,D);*

*initial*

*begin*

*A = 4'd0; #10;*

*A = 4'd1; #10;*

*A = 4'd2; #10;*

*A = 4'd3; #10;*

*A = 4'd4; #10;*

*A = 4'd5; #10;*

*A = 4'd6; #10;*

*A = 4'd7; #10;*

*A = 4'd8; #10;*

*A = 4'd9; #10;*

*A = 4'd10; #10;*

*A = 4'd11; #10;*

*A = 4'd12; #10;*

*A = 4'd13; #10;*

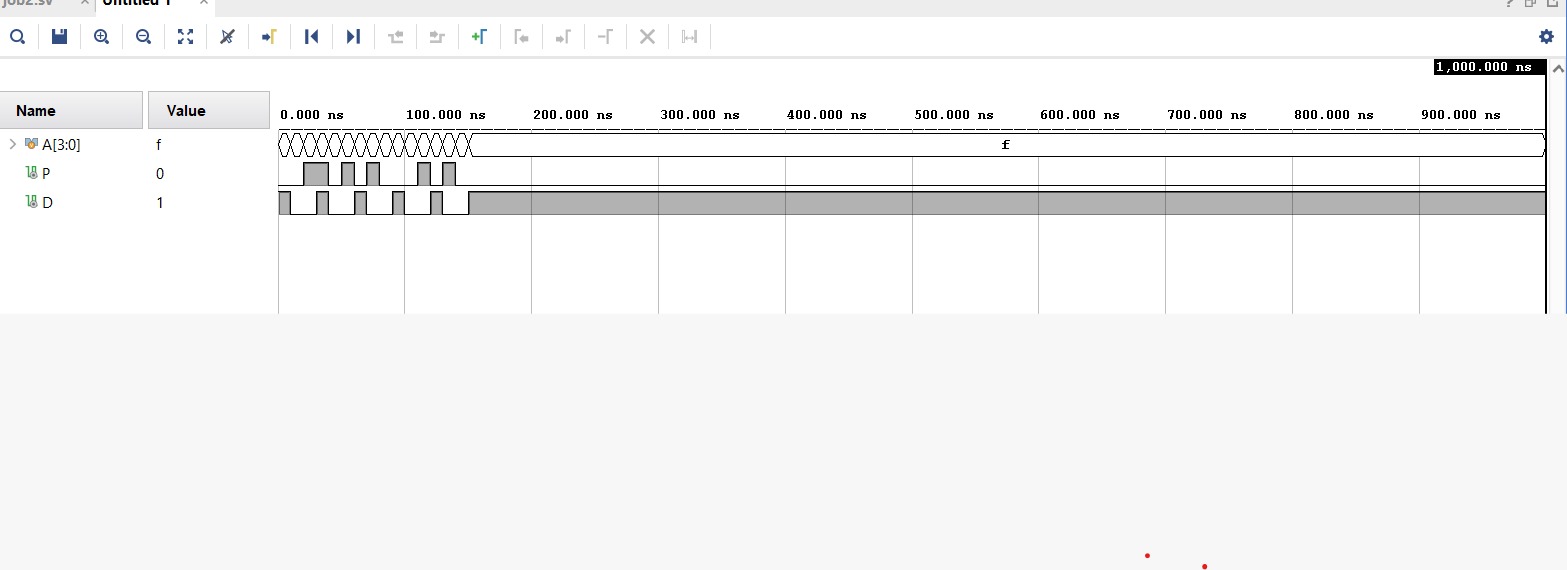
*A = 4'd14; #10;*

*A = 4'd15; #10;*

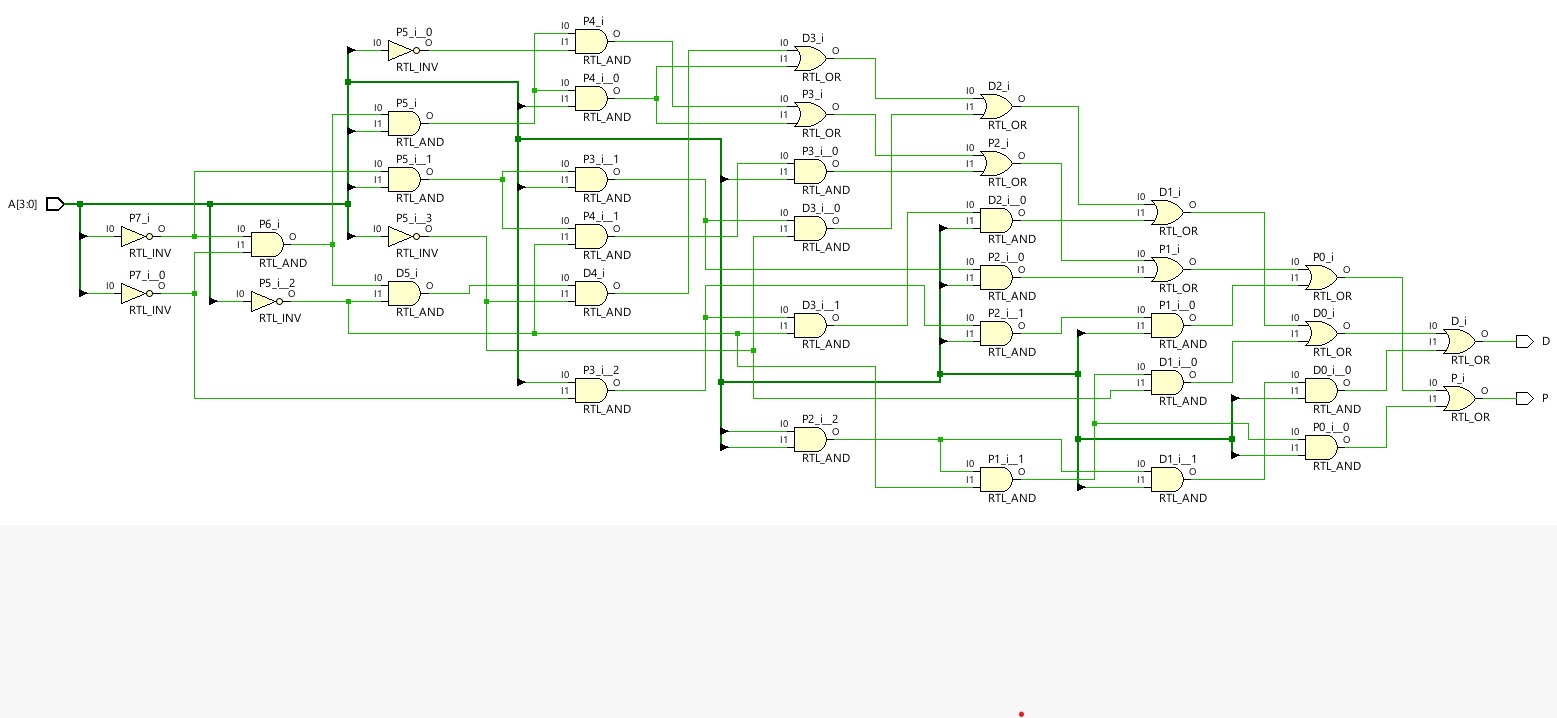
*end*

*endmodule*

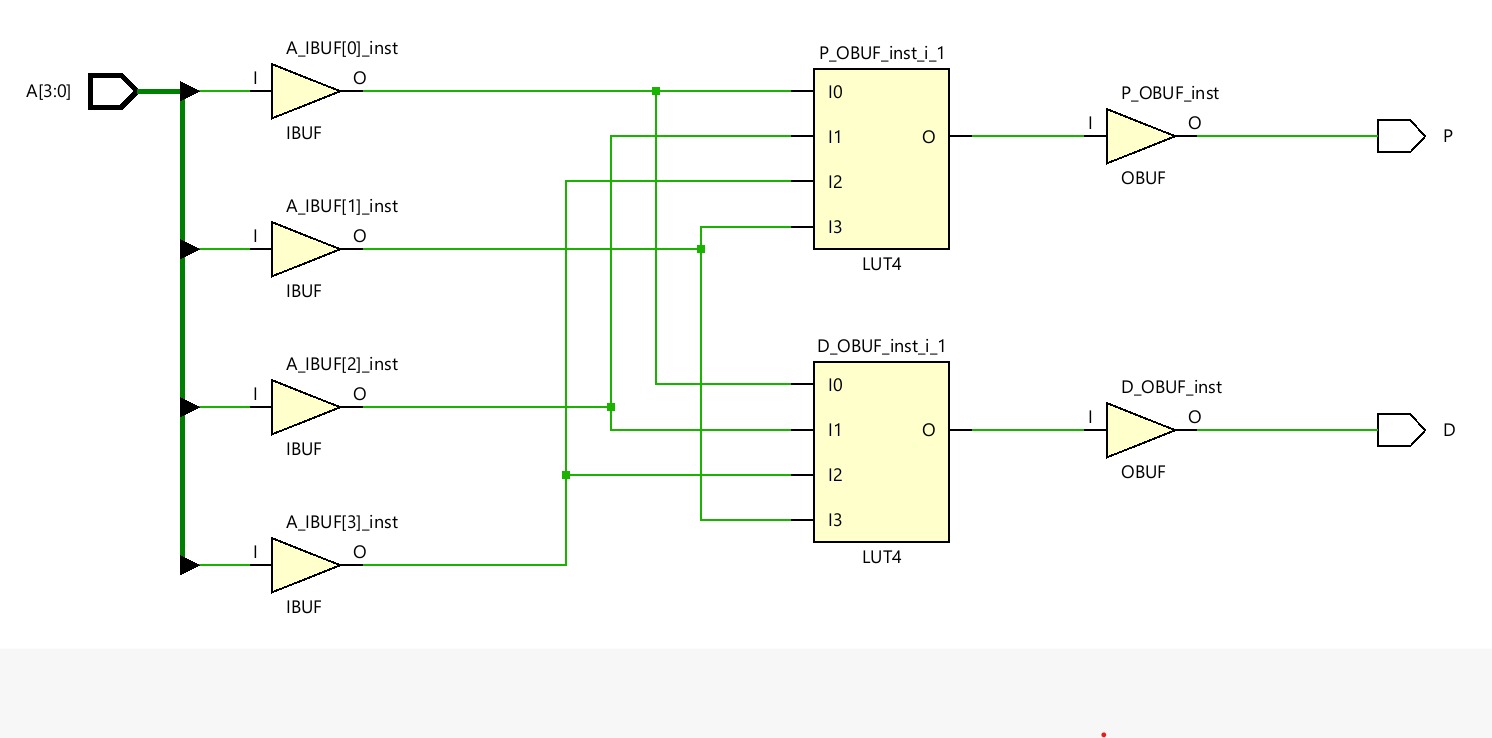
*WAVEFORM*

**

*RTL*

**

Technology schematic

**

A four-input priority encoder

CODE

*module prio\_enco(input logic [3:0] A, output logic [1:0] Y, output logic valid);*

*always\_comb*

*casez(A)*

*4'b0001, 4'b001x, 4'b01xx, 4'b1xxx : Y = 2'b00;*

*4'b0010, 4'b0011, 4'b01xx, 4'b1xxx : Y = 2'b01;*

*4'b0100, 4'b0101, 4'b011x, 4'b1xxx : Y = 2'b10;*

*4'b1000, 4'b1001, 4'b101x, 4'b11xx : Y = 2'b11;*

*default : Y = 2'b00;*

*endcase*

*assign valid = (A != 4'b0000);*

*Endmodule*

*TESTBENCH*

*module testbench;*

*reg [3:0] A;*

*wire [1:0] Y;*

*wire valid;*

*prio\_enco DUT (A, Y, valid);*

*initial begin*

*A = 4'b0000; #10;*

*A = 4'b0001; #10;*

*A = 4'b0010; #10;*

*A = 4'b0100; #10;*

*A = 4'b1000; #10;*

*A = 4'b1100; #10;*

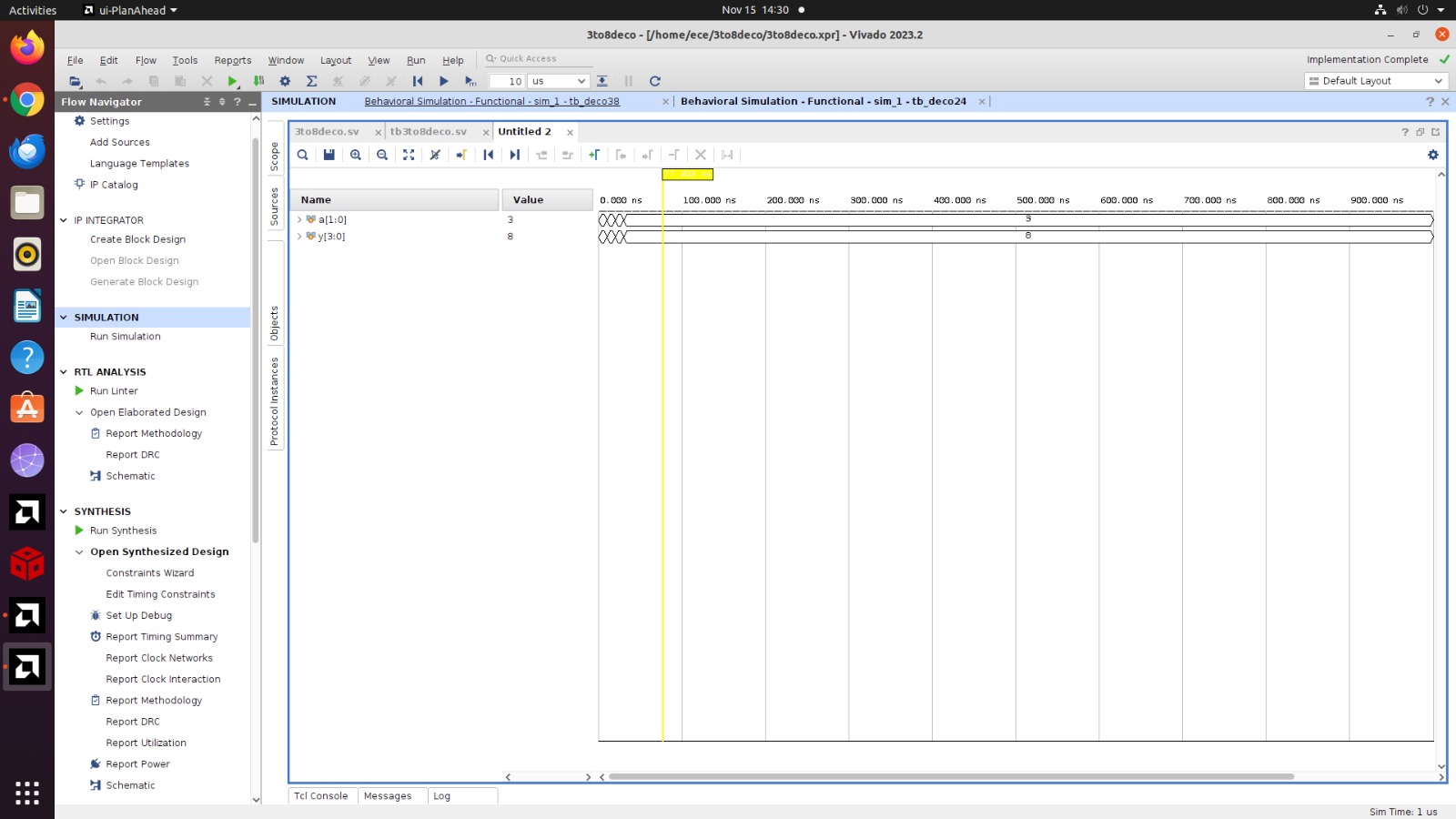
*A = 4'b1110; #10;*

*A = 4'b1111; #10;*

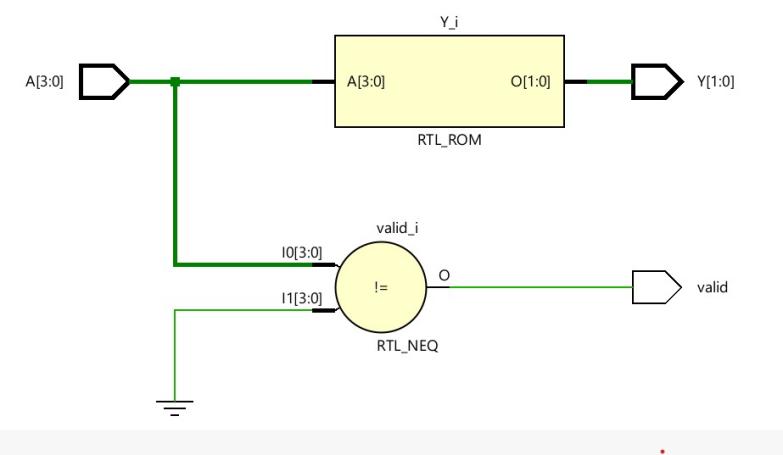
*end*

*endmodule*

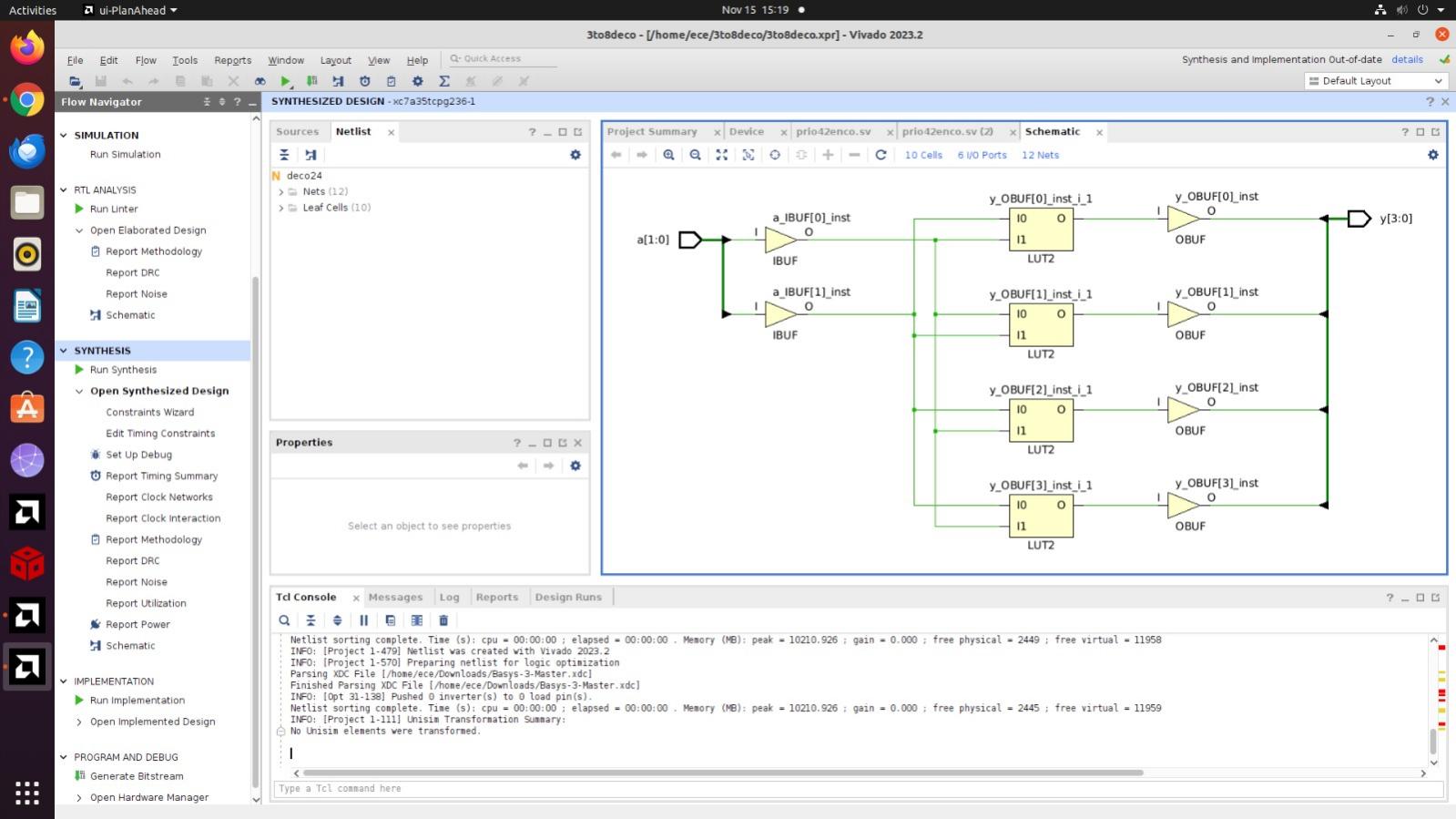
*WAVEFORM*

**

*RTL*

**

Technology schematic

**

An eight-input priority encoder

CODE

module prio\_enco(

input logic [7:0] A,

output logic [2:0] Y

);

always\_comb begin

casez (A)

8'b00000001, 8'b0000001x, 8'b000001xx, 8'b00001xxx, 8'b0001xxxx, 8'b001xxxxx, 8'b01xxxxxx, 8'b1xxxxxxx: Y = 3'b000;

8'b00000010, 8'b0000011x, 8'b00001xxx, 8'b0001xxxx, 8'b001xxxxx, 8'b01xxxxxx, 8'b1xxxxxxx: Y = 3'b001;

8'b00000100, 8'b000011xx, 8'b0001xxxx, 8'b001xxxxx, 8'b01xxxxxx, 8'b1xxxxxxx: Y = 3'b010;

8'b00001000, 8'b00011xxx, 8'b001xxxxx, 8'b01xxxxxx, 8'b1xxxxxxx: Y = 3'b011;

8'b00010000, 8'b001xxxxx, 8'b01xxxxxx, 8'b1xxxxxxx: Y = 3'b100;

8'b00100000, 8'b01xxxxxx, 8'b1xxxxxxx: Y = 3'b101;

8'b01000000, 8'b1xxxxxxx: Y = 3'b110;

8'b10000000: Y = 3'b111;

default: Y = 3'b000; // Default case for no valid input

endcase

end

endmodule

TESTBENCH

*module testbench;*

*reg [7:0] A;*

*wire [2:0] Y;*

*wire valid;*

*prio\_enco DUT (A, Y, valid);*

*initial begin*

*// Apply test cases*

*A = 8'b00000000; #10;*

*A = 8'b00000001; #10;*

*A = 8'b00000010; #10;*

*A = 8'b00000100; #10;*

*A = 8'b00001000; #10;*

*A = 8'b00010000; #10;*

*A = 8'b00100000; #10;*

*A = 8'b01000000; #10;*

*A = 8'b10000000; #10;*

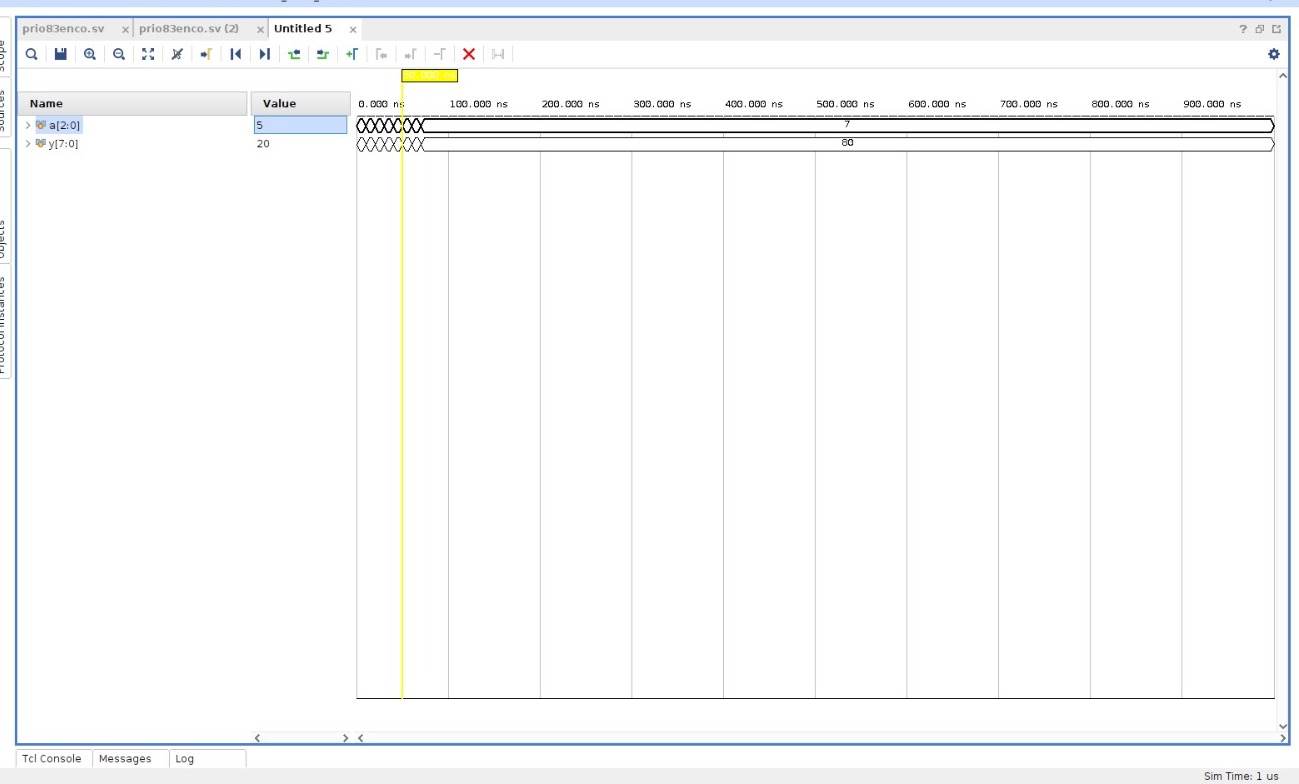
*A = 8'b11111111; #10;*

*$stop;*

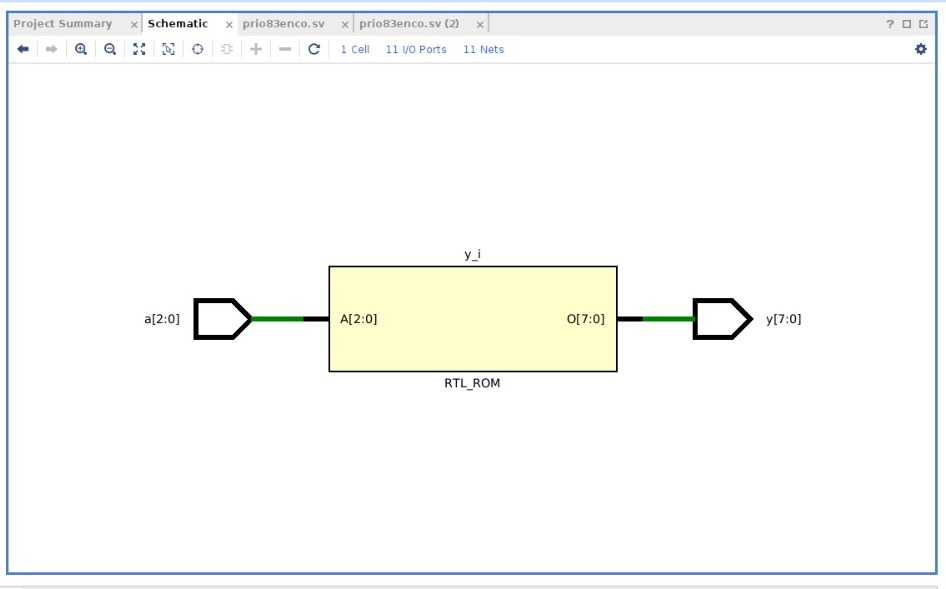
*end*

*Endmodule*

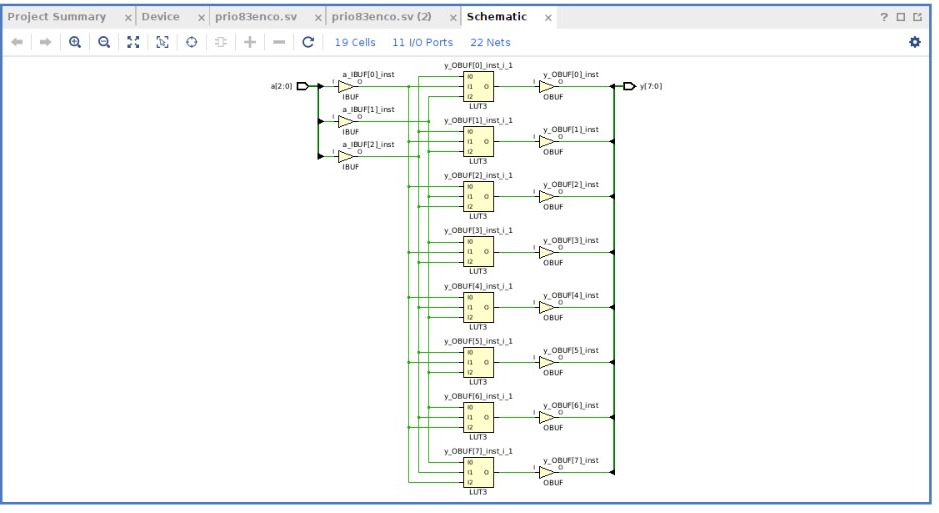
*WAVEFORM*

**

*RTL*

**

*Technology schematic*

**

(g) A 3:8 decoder.

CODE

*module deco38(input logic [2:0] a, output logic [7:0] y);*

*always\_comb*

*case(a)*

*3'b000 : y = 8'b00000001;*

*3'b001 : y = 8'b00000010;*

*3'b010 : y = 8'b00000100;*

*3'b011 : y = 8'b00001000;*

*3'b100 : y = 8'b00010000;*

*3'b101 : y = 8'b00100000;*

*3'b110 : y = 8'b01000000;*

*3'b111 : y = 8'b10000000;*

*default: y = 8'bxxxxxxxx;*

*endcase*

*endmodule*

TESTBENCH

*module tb\_deco38;*

*logic [2:0] a;*

*logic [7:0] y;*

*deco38 dut (a, y);*

*initial begin*

*a = 3'b000; #10;*

*a = 3'b001; #10;*

*a = 3'b010; #10;*

*a = 3'b011; #10;*

*a = 3'b100; #10;*

*a = 3'b101; #10;*

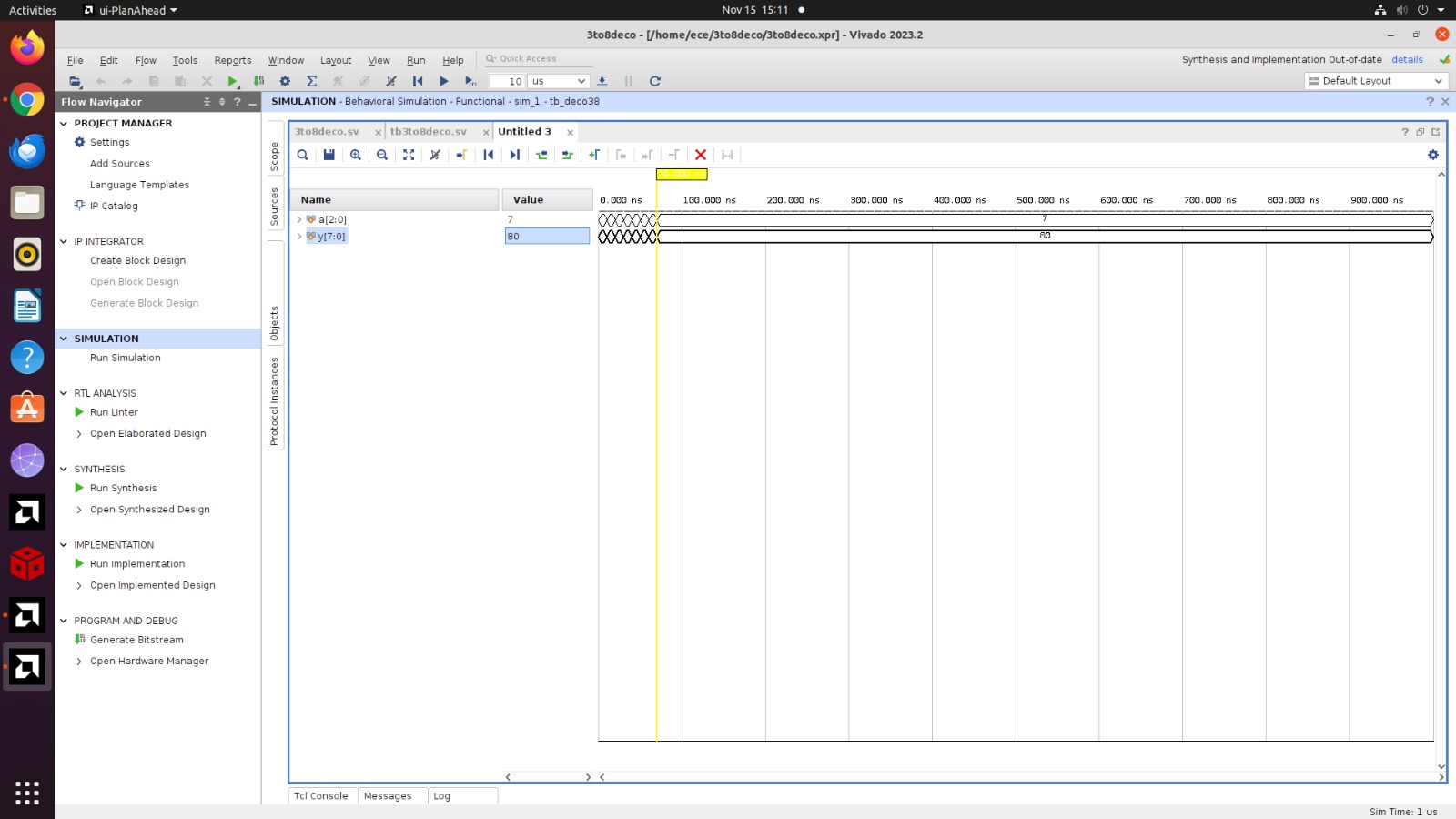
*a = 3'b110; #10;*

*a = 3'b111; #10;*

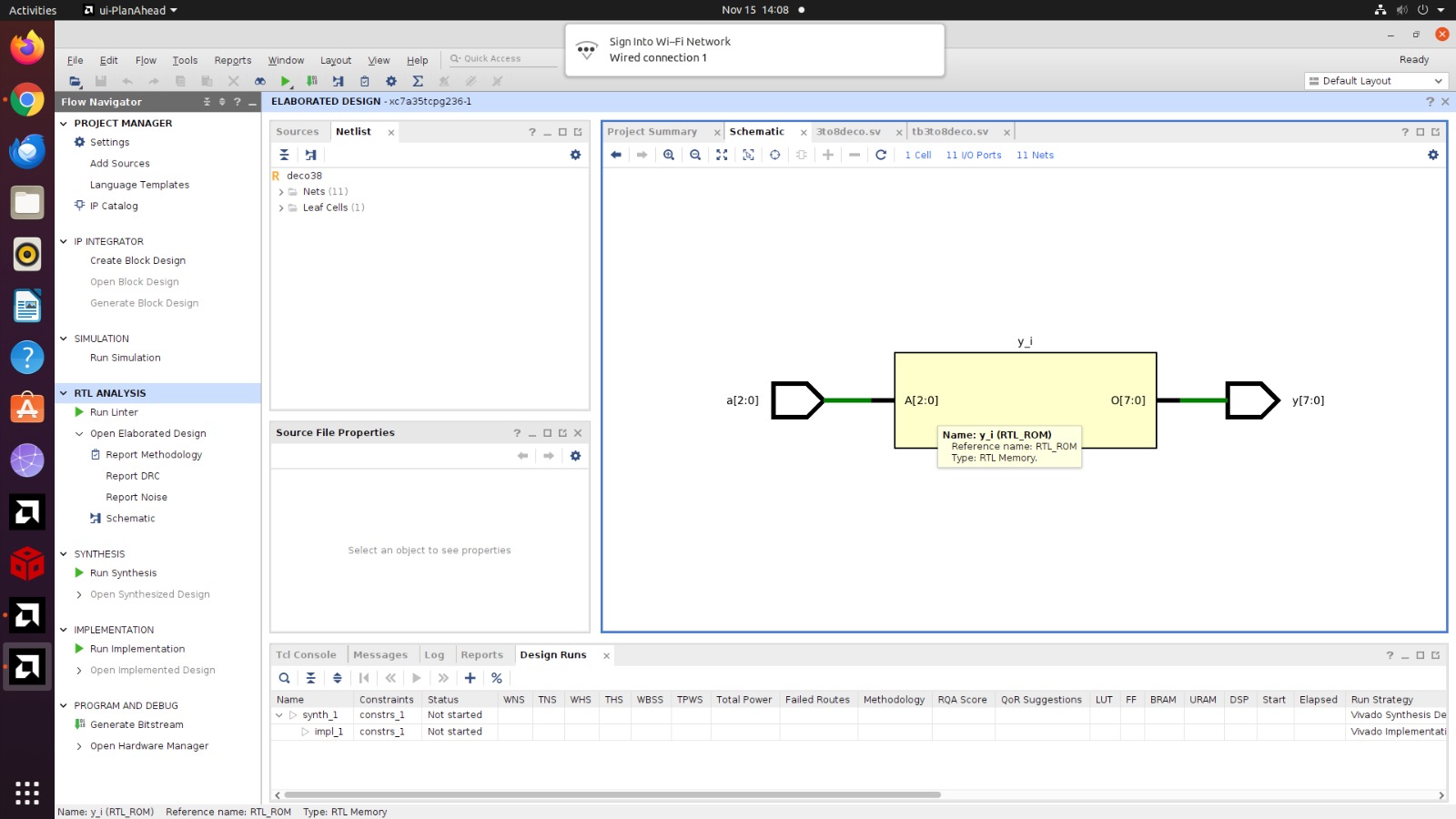
*end*

*endmodule*

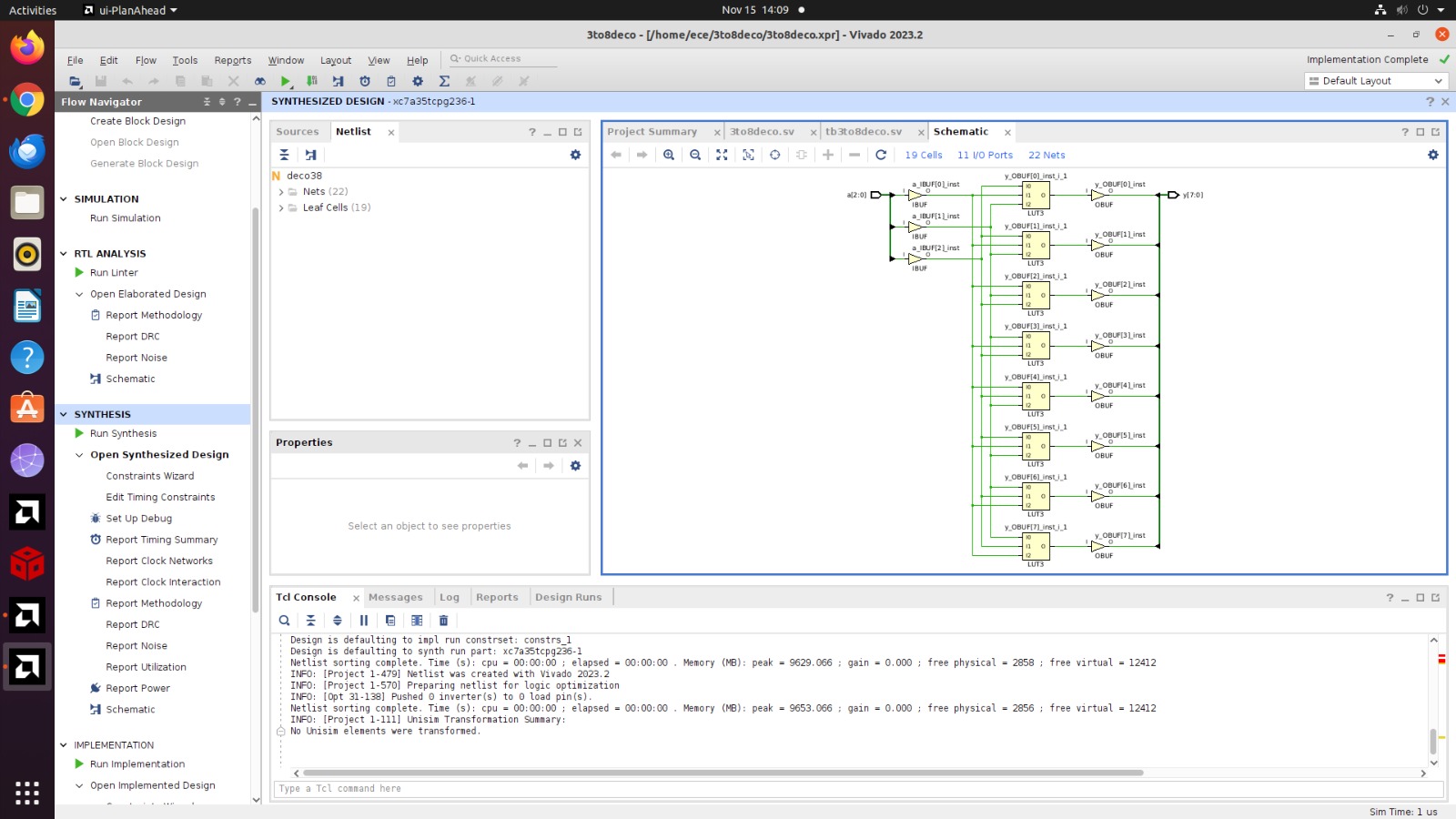
WAVEFORM

**

*RTL*

**

*Technology schematic*

**

(h) A 4-bit carry propagate adder (with no carry in or out).

CODE

module carry\_propagate\_adder(

input [3:0] A,

input [3:0] B,

output [3:0] Sum

);

assign Sum = A + B;

endmodule

TESTBENCH

module carry\_propagate\_adder\_tb;

reg [3:0] A, B;

wire [3:0] Sum;

carry\_propagate\_adder uut (

.A(A),

.B(B),

.Sum(Sum)

);

initial begin

A = 4'b0000; B = 4'b0000; #10;

A = 4'b0001; B = 4'b0001; #10;

A = 4'b0010; B = 4'b0010; #10;

A = 4'b0100; B = 4'b0100; #10;

A = 4'b1000; B = 4'b1000; #10;

A = 4'b1111; B = 4'b1111; #10;

A = 4'b1101; B = 4'b1011; #10;

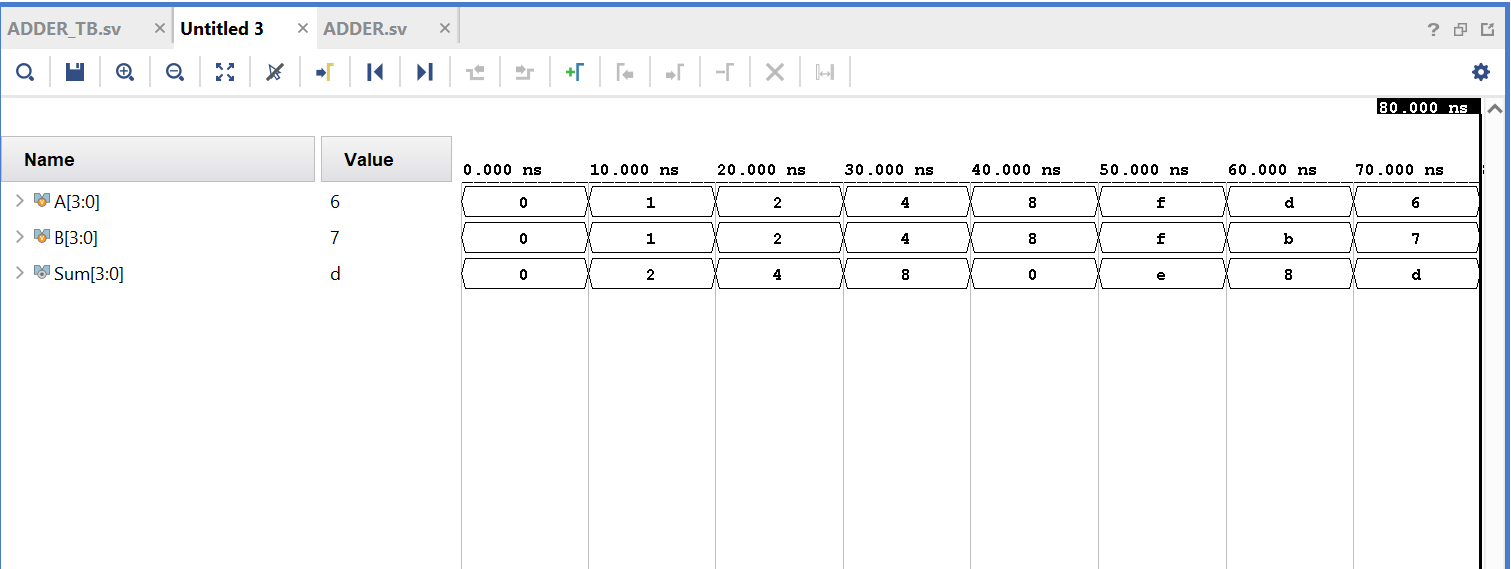
A = 4'b0110; B = 4'b0111; #10;

$finish;

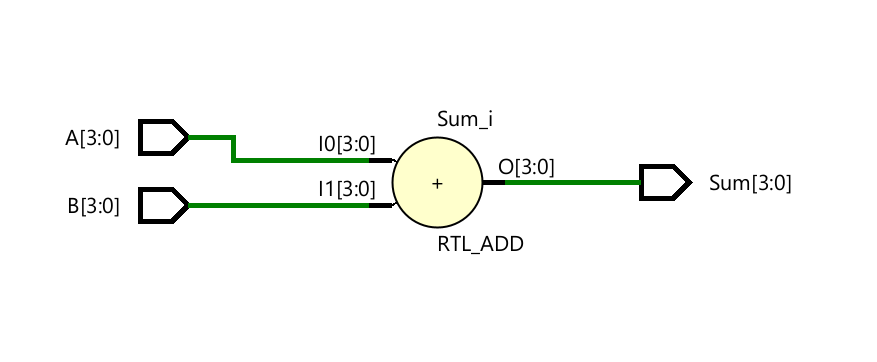
end

Endmodule

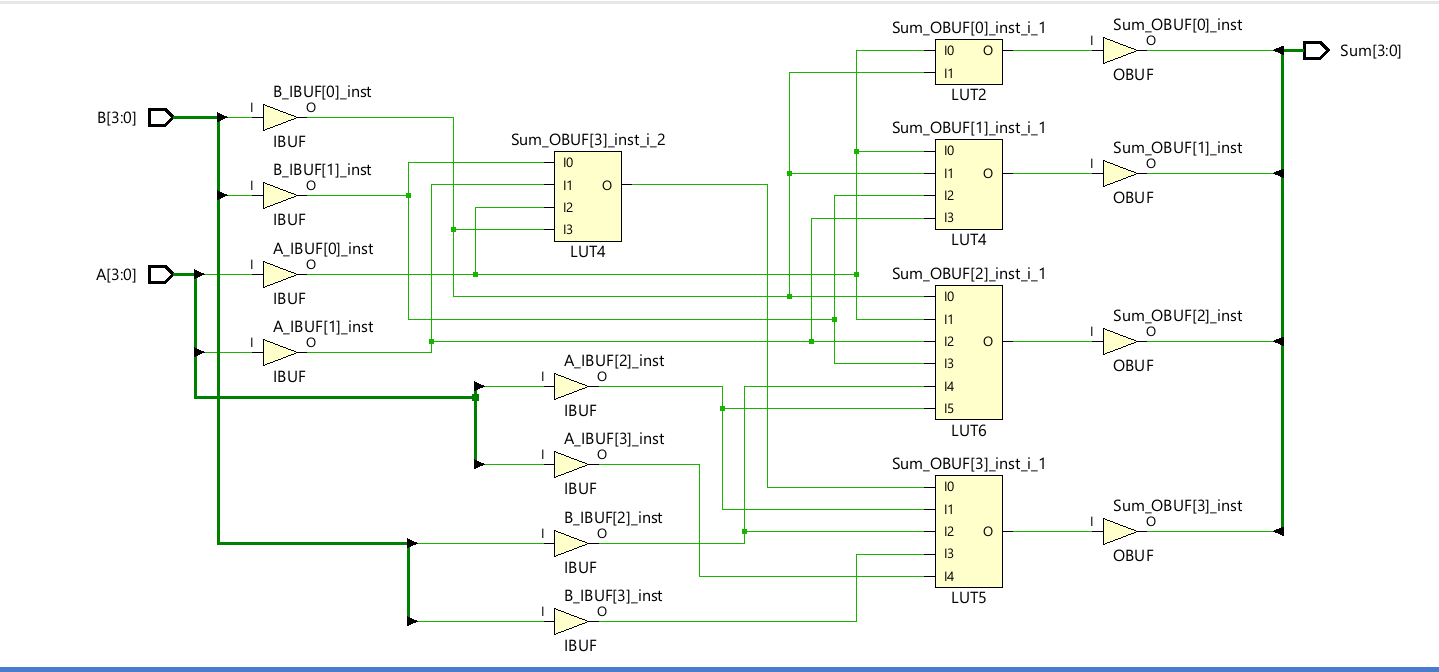
WAVEFORM



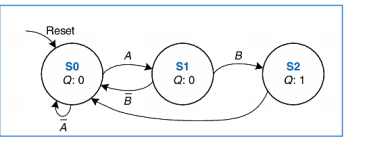
RTL



*Technology schematic*



1. The FSM



CODE

module fsm(

input logic clk,

input logic reset,

input logic A,

input logic B,

output logic Q

);

typedef enum logic [1:0] {S0 = 2'b00, S1 = 2'b01, S2 = 2'b10} state\_t;

state\_t state, next\_state;

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

if (reset)

state <= S0;

else

state <= next\_state;

end

always\_ff @(state or A or B) begin

case (state)

S0:

if (~A) next\_state = S0;

else next\_state = S1;

S1:

if (~B) next\_state = S1;

else next\_state = S2;

S2:

if (~A) next\_state = S2;

else next\_state = S0;

default: next\_state = S0;

endcase

end

assign Q = (state == S2);

endmodule

TESTBENCH

module fsm\_tb;

reg clk,reset,A,B;

wire Q;

fsm uut(.clk(clk),.reset(reset),.A(A),.B(B),.Q(Q));

always #5 clk=~clk;

initial begin

clk=0;reset=0;A=0;B=0;

#10 reset=1;#10 reset=0;

#10 A=1;B=0;#10 A=0;B=0;

#10 A=0;B=1;#10 A=0;B=0;

#10 A=1;B=0;#10 A=0;B=0;

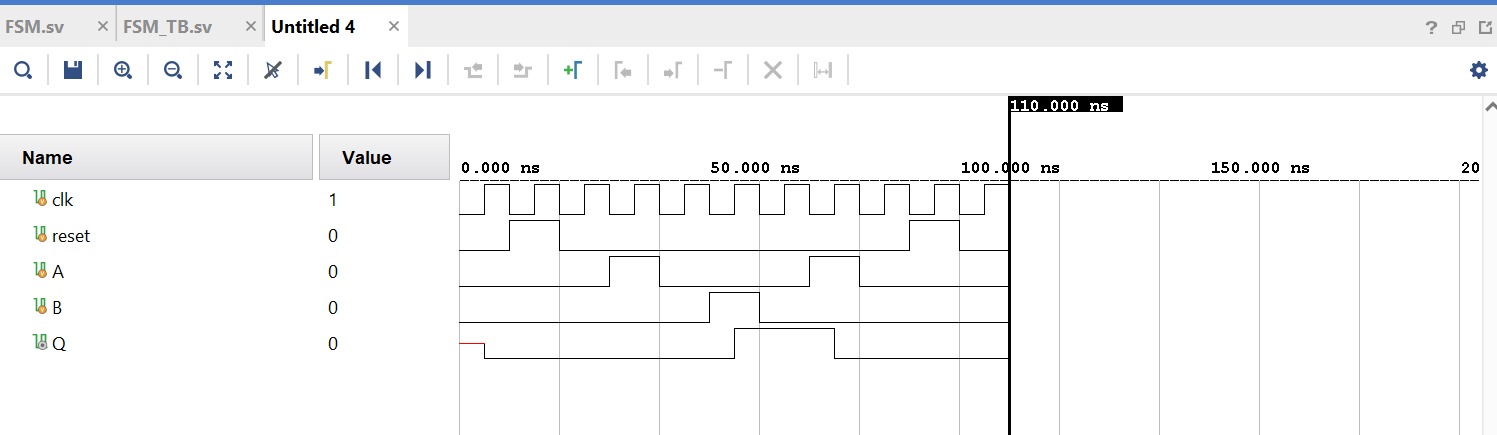
#10 reset=1;#10 reset=0;

#10 $finish;

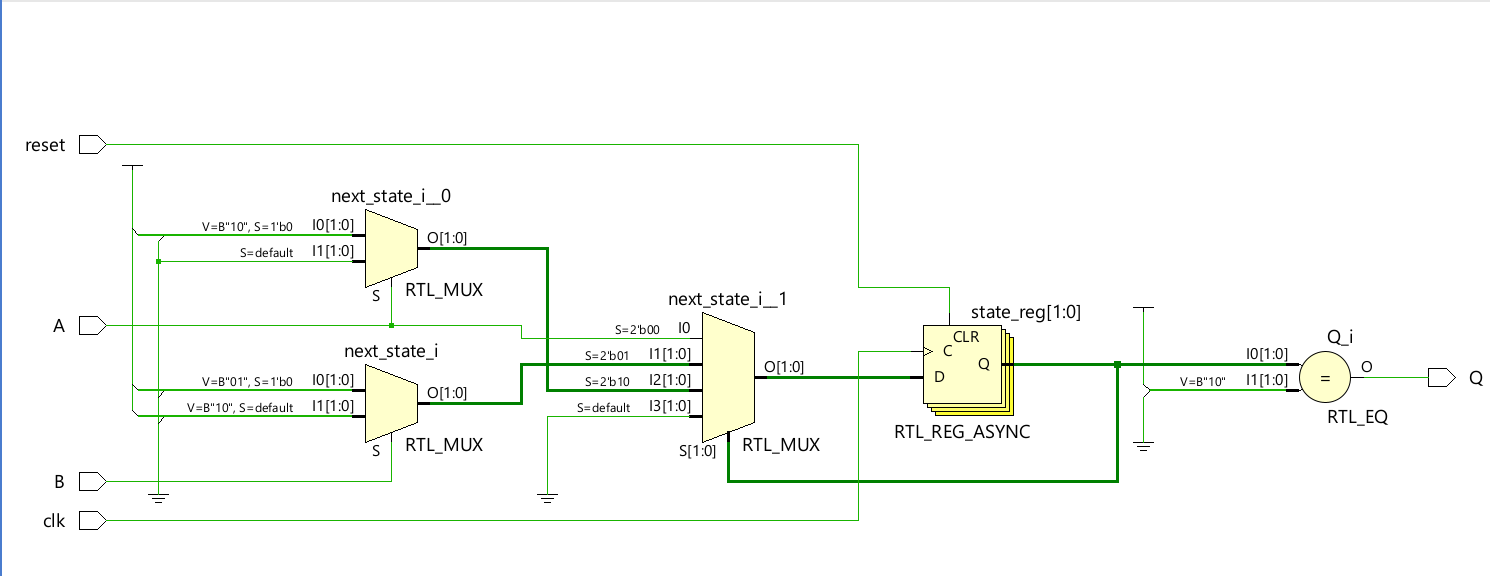
end

endmodule

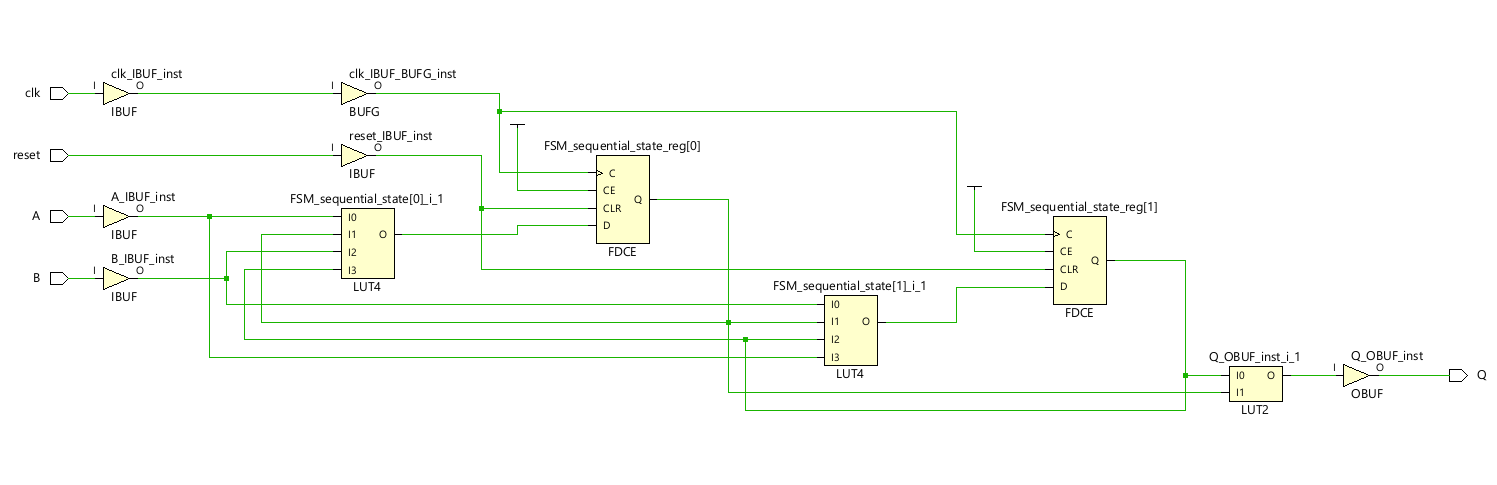
WAVEFORM



RTL



*Technology schematic*



The Gray code counter

CODE

module binary\_to\_gray (

input logic [3:0] B,

output logic [3:0] G

);

assign G[3] = B[3];

assign G[2] = B[3] ^ B[2];

assign G[1] = B[2] ^ B[1];

assign G[0] = B[1] ^ B[0];

endmodule

TESTBENCH

module tb\_binary\_to\_gray();

logic [3:0] B;

logic [3:0] G;

binary\_to\_gray uut (

.B(B),

.G(G)

);

initial begin

B = 4'b0000; #10;

B = 4'b0001; #10;

B = 4'b0010; #10;

B = 4'b0011; #10;

B = 4'b0100; #10;

B = 4'b0101; #10;

B = 4'b0110; #10;

B = 4'b0111; #10;

B = 4'b1000; #10;

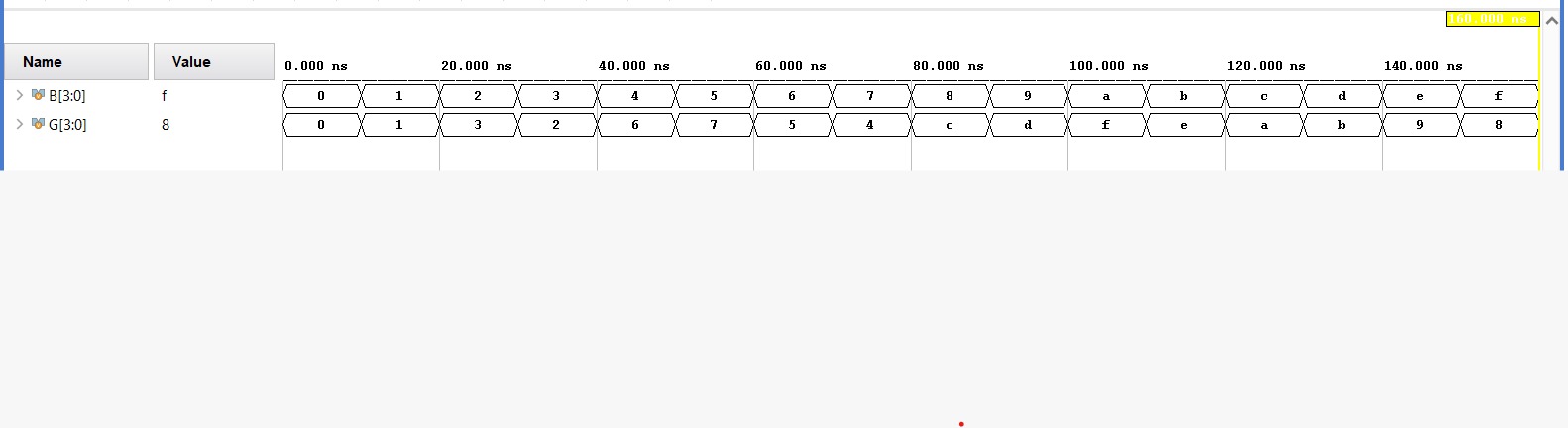
B = 4'b1001; #10;

$finish;

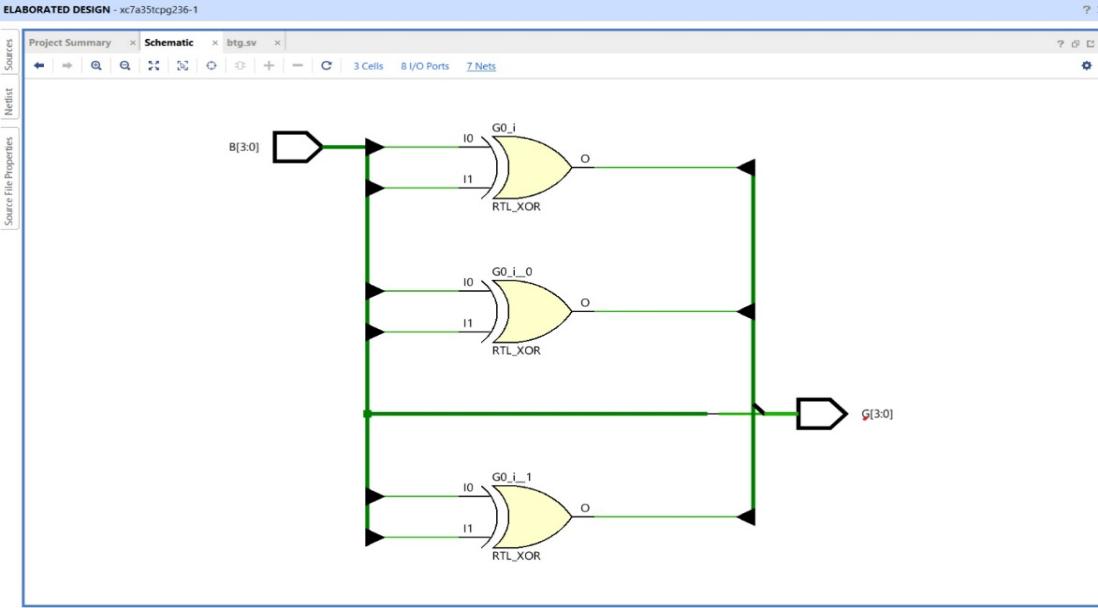
end

endmodule

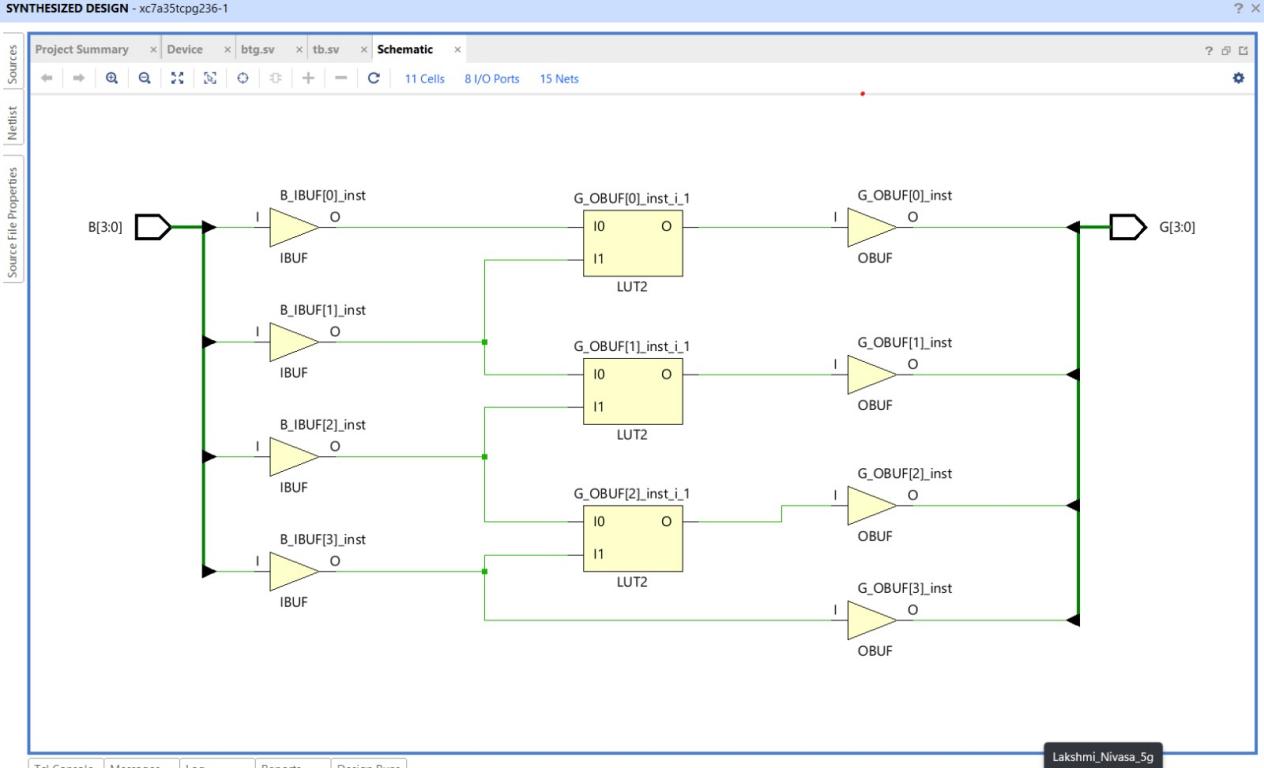
WAVEFORM

**

*RTL*

****

*Technology schematic*

****