1. Problem Statement: 8-bit BCD Adder

Design code

module bcd\_adder(

input [7:0] A,

input [7:0] B,

input Cin,

output [7:0] S,

output Cout

);

logic [3:0] lsum, hsum;

logic lcarry, hcarry;

assign {lcarry, lsum} = A[3:0] + B[3:0] + Cin;

assign {hcarry, hsum} = A[7:4] + B[7:4] + (lsum > 4'd9);

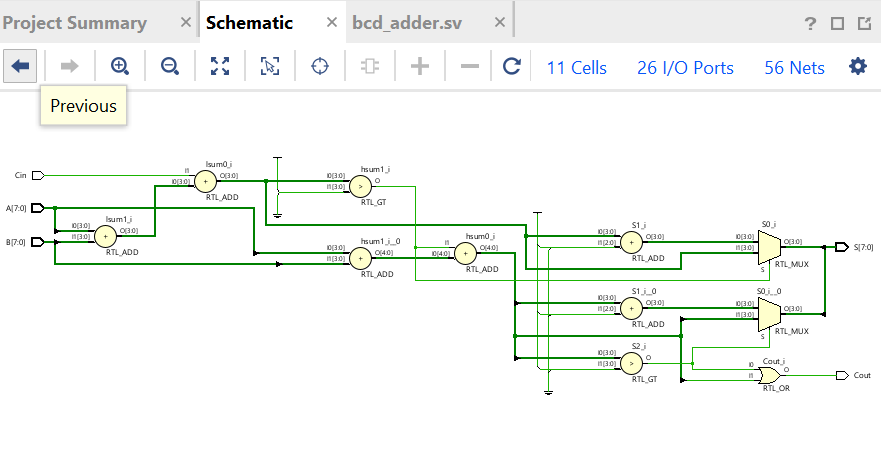
assign S = { (hsum > 4'd9) ? (hsum + 4'd6) : hsum,

(lsum > 4'd9) ? (lsum + 4'd6) : lsum };

assign Cout = (hsum > 4'd9) || hcarry;

endmodule

RTL



Testbenh:

module testbench;

reg [7:0] A;

reg [7:0] B;

reg Cin;

wire [7:0] S;

wire Cout;

bcd\_adder uut (

.A(A),

.B(B),

.Cin(Cin),

.S(S),

.Cout(Cout)

);

initial begin

$monitor("Time: %0d | A=%d B=%d Cin=%b | S=%d Cout=%b", $time, A, B, Cin, S, Cout);

A = 8'd45; B = 8'd36; Cin = 0; #10;

A = 8'd99; B = 8'd99; Cin = 0; #10;

A = 8'd12; B = 8'd89; Cin = 1; #10;

A = 8'd50; B = 8'd50; Cin = 0; #10;

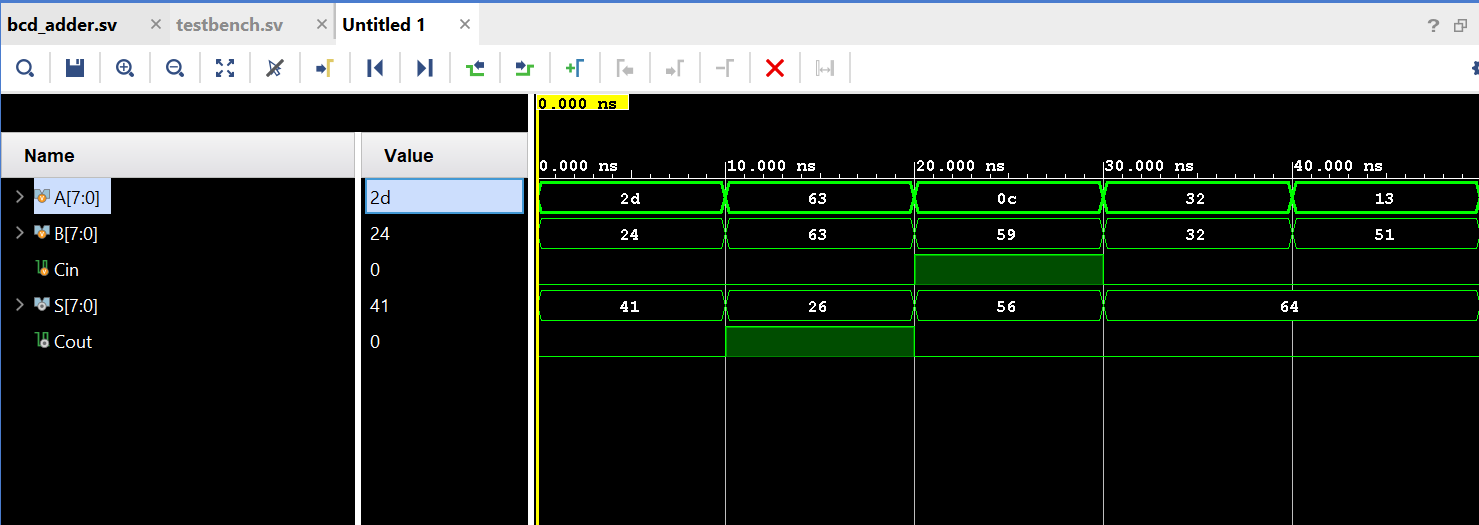
A = 8'd19; B = 8'd81; Cin = 0; #10;

$finish;

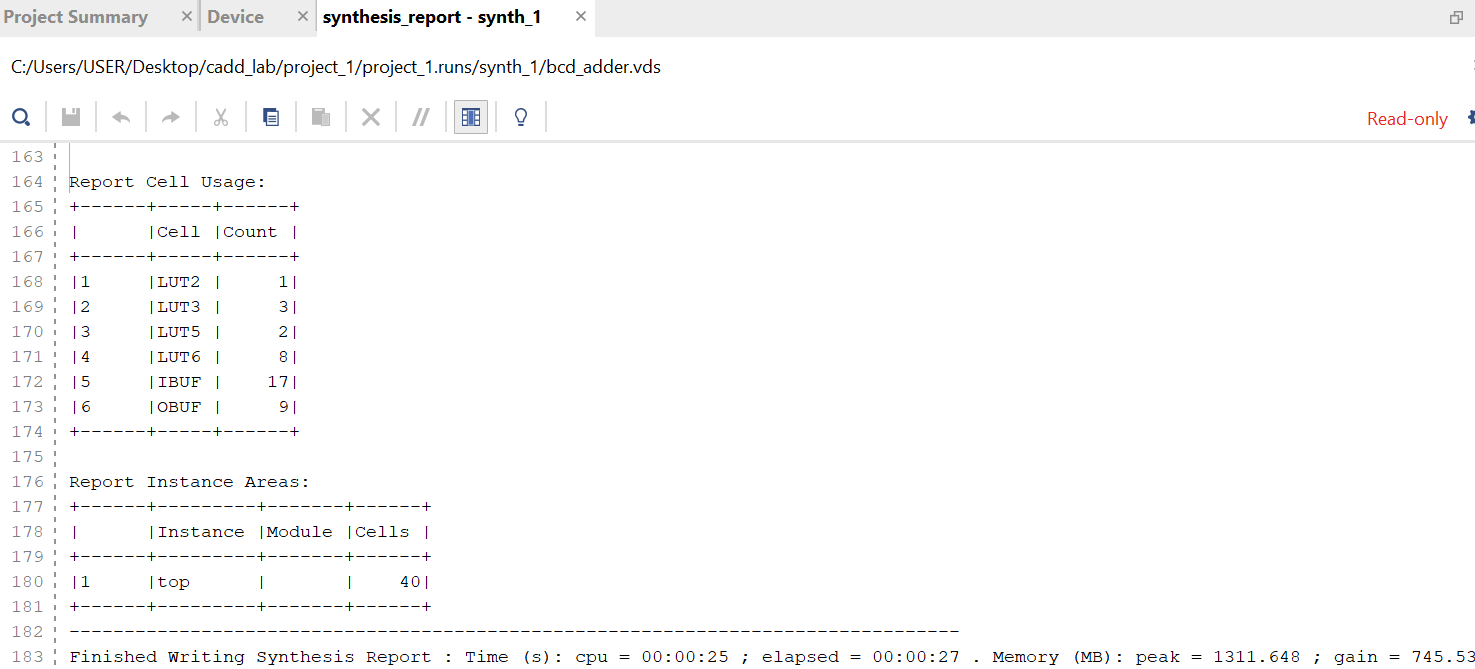
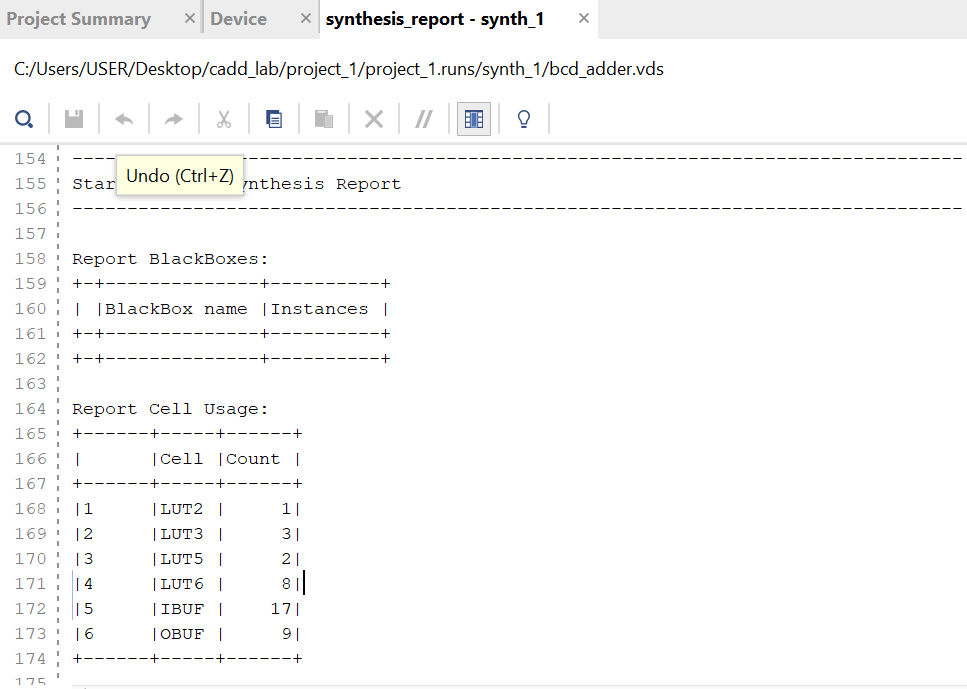
end

endmodule

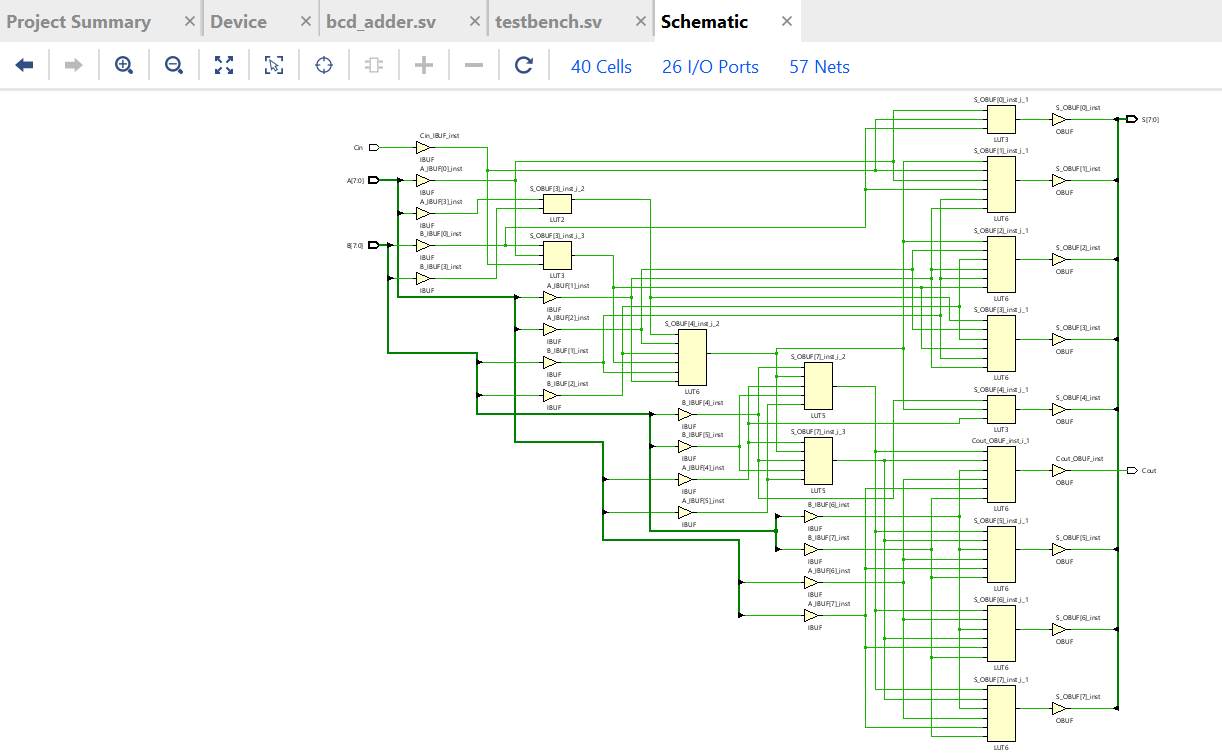
wave form:



Synthesis reort:



TECHNOLOGY DIAGRAM



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

**(a) Boolean Simplification**

Design code:

module combinational\_a (

input A, B, C, D,

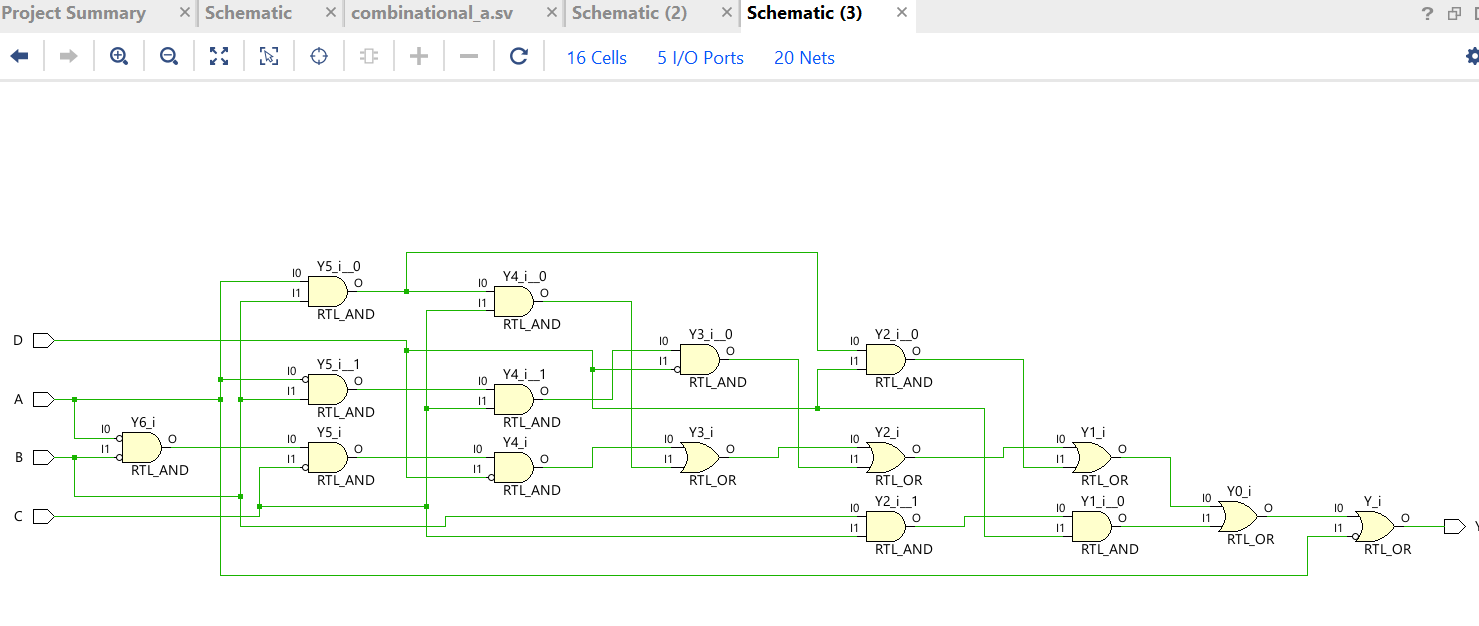
output Y

);

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

endmodule

RTL:



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

TESTBENCH:

module testbench;

logic A, B, C, D;

logic Y;

combinational\_a uut (

.A(A), .B(B), .C(C), .D(D), .Y(Y) );

initial begin

$display("Testbench for comb\_function1");

$display("A B C D | Y");

A = 0; B = 0; C = 0; D = 0; #5;

$display("%b %b %b %b | %b", A, B, C, D, Y);

A = 1; B = 0; C = 1; D = 1; #5;

$display("%b %b %b %b | %b", A, B, C, D, Y);

A = 1; B = 1; C = 0; D = 1; #5;

$display("%b %b %b %b | %b", A, B, C, D, Y);

A = 0; B = 1; C = 1; D = 0; #5;

$display("%b %b %b %b | %b", A, B, C, D, Y);

A = 1; B = 1; C = 1; D = 1; #5;

$display("%b %b %b %b | %b", A, B, C, D, Y);

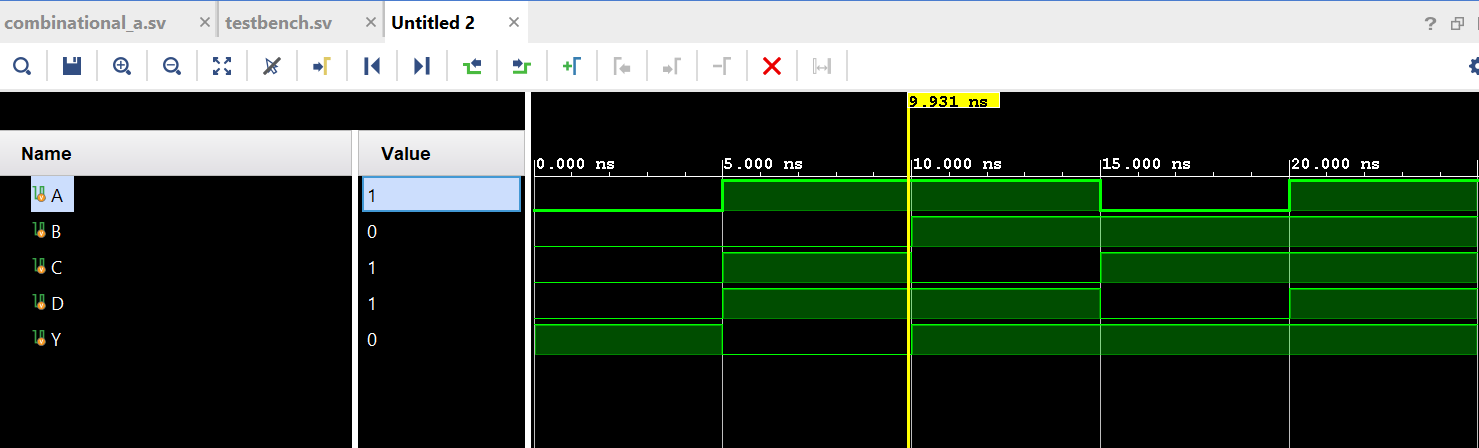
$display("Test complete.");

$finish;

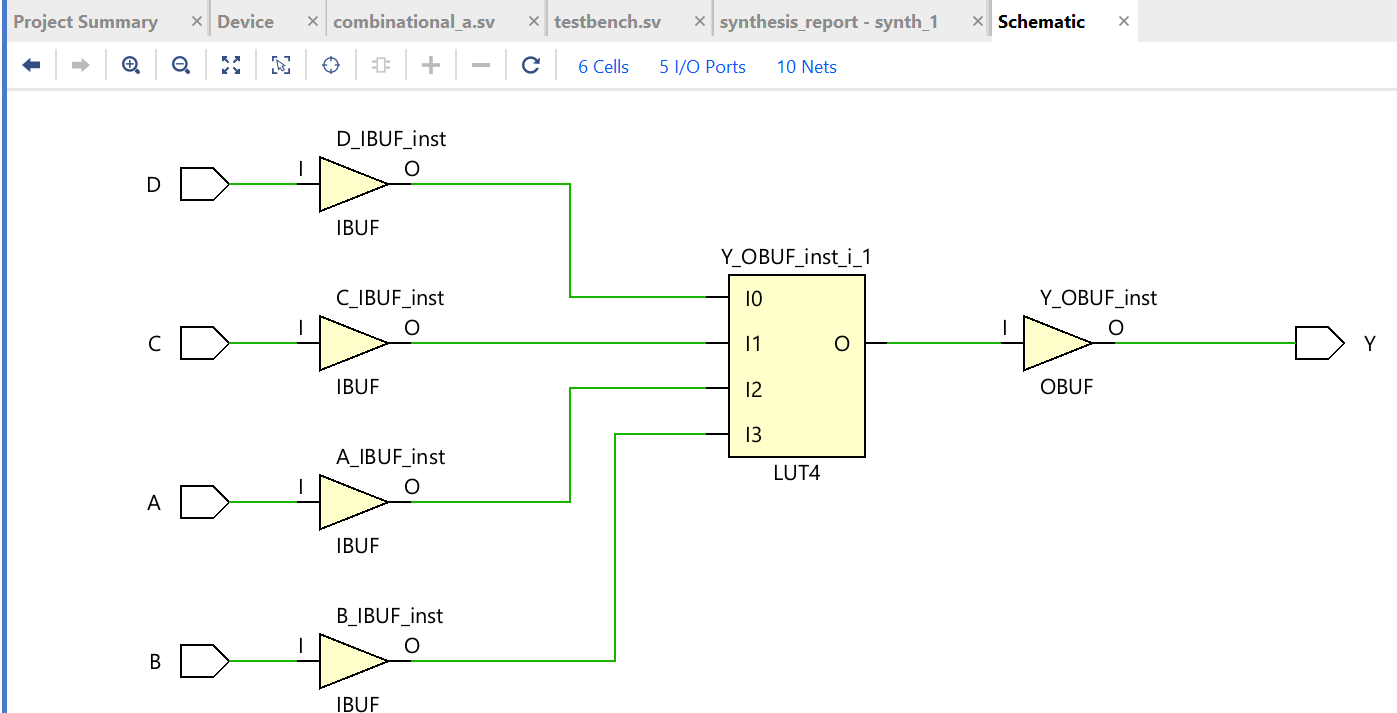
end

endmodule

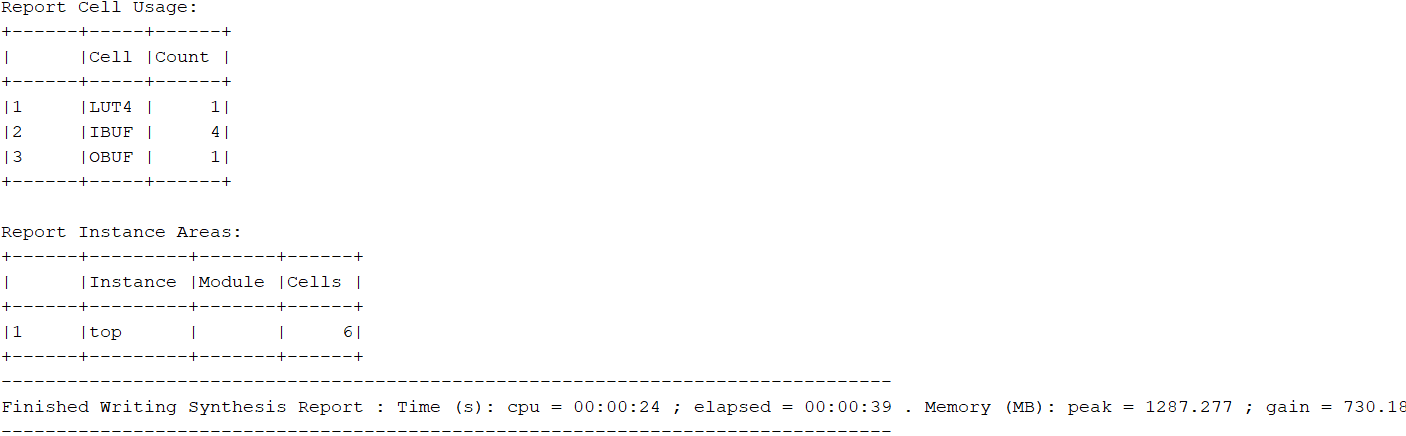
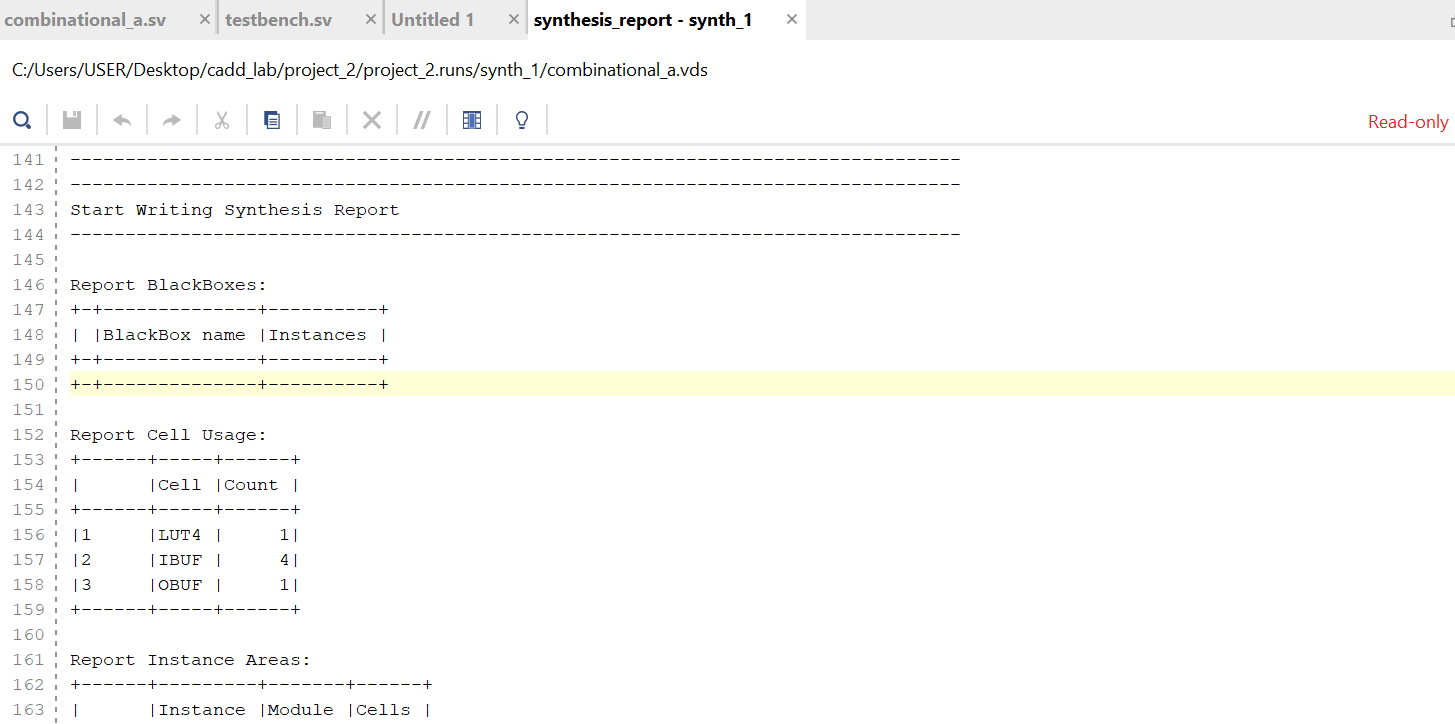
WAVEFORM:



TECHNOLOGY SCREENSHOT:



Synthesis REPORT:



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**(b)COMBINATIONAL\_B**

**DESIGN CODE:**

**module combinational\_b (**

**input A, B, C, D, E,**

**output Y**

**);**

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

**(A | D | E) | (B & C & D) | (B & C & E) | (B & D & E) | (C & D & E);**

**Endmodule**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Testbench**

**module testbench;**

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

**logic Y;**

**combinational\_b uut (**

**.A(A), .B(B), .C(C), .D(D), .E(E), .Y(Y) );**

**initial begin**

**$display("Testbench for comb\_function2");**

**$display("A B C D E | Y");**

**// Test cases**

**A = 0; B = 0; C = 0; D = 0; E = 0; #5; $display("%b %b %b %b %b | %b", A, B, C, D, E, Y);**

**A = 1; B = 0; C = 1; D = 1; E = 0; #5; $display("%b %b %b %b %b | %b", A, B, C, D, E, Y);**

**A = 0; B = 1; C = 0; D = 1; E = 1; #5; $display("%b %b %b %b %b | %b", A, B, C, D, E, Y);**

**A = 1; B = 1; C = 1; D = 0; E = 1; #5; $display("%b %b %b %b %b | %b", A, B, C, D, E, Y);**

**A = 1; B = 1; C = 1; D = 1; E = 1; #5; $display("%b %b %b %b %b | %b", A, B, C, D, E, Y);**

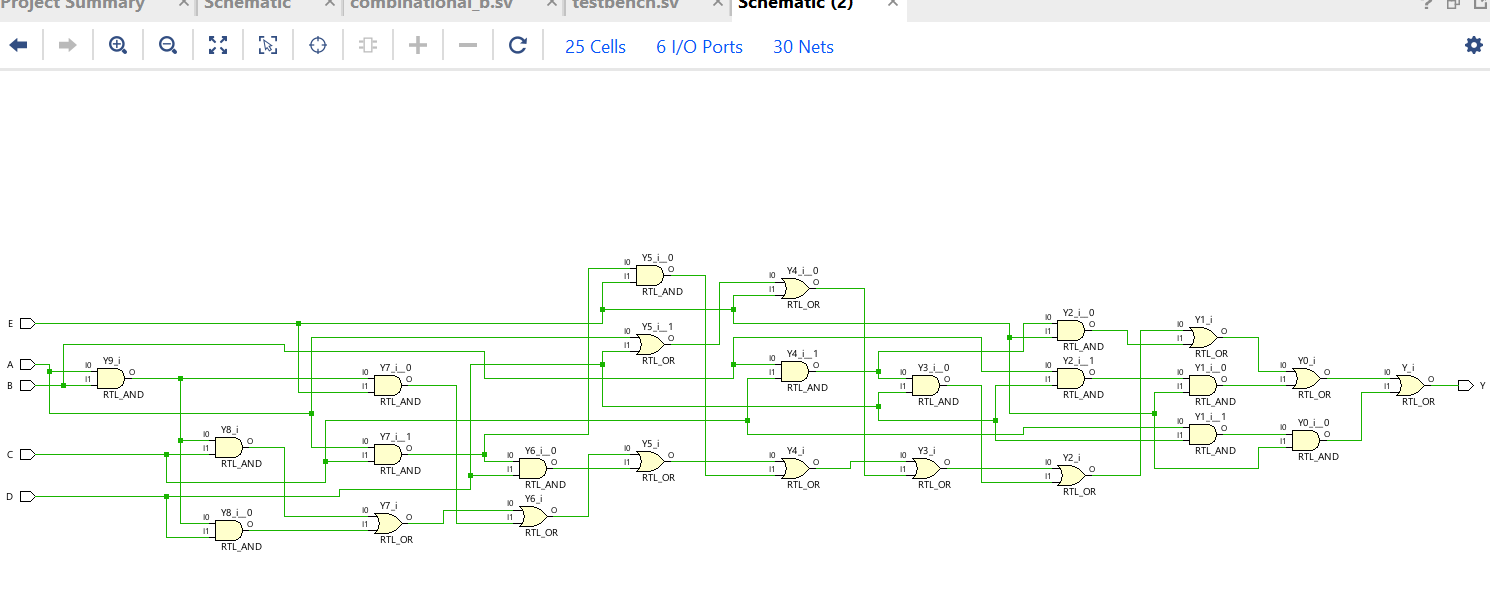
**$display("Test complete.");**

**$finish;**

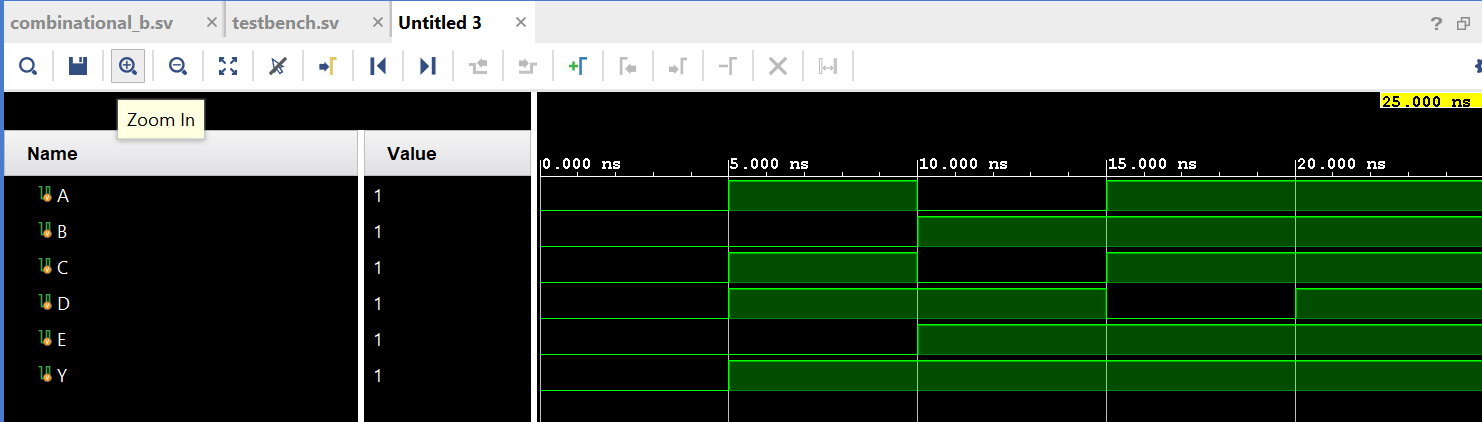
**end**

**endmodule**

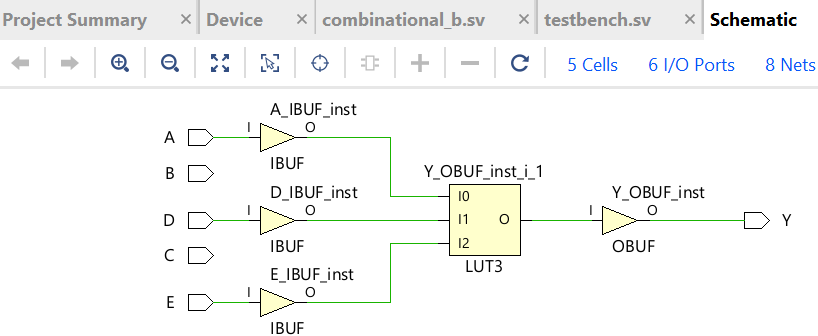
**RTL:**

****

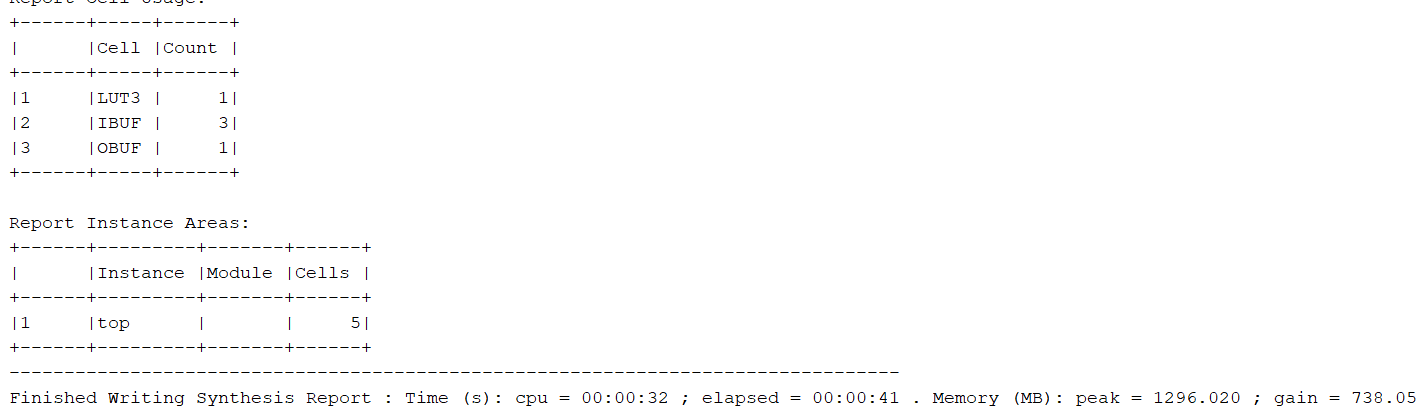
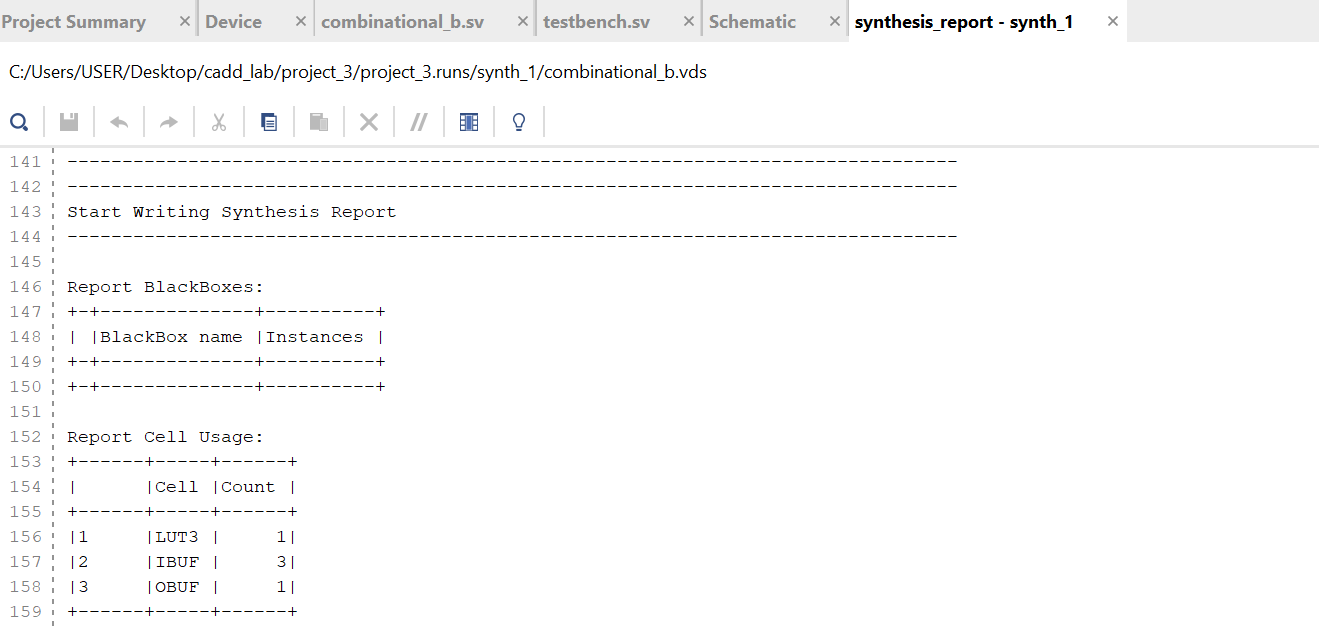
**WAVEFORM:**

****

TECNOLOGY SCREENSHOT:



SYNTHESIS REPORT:



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(c): **Two-Output Function**

**DESIGN CODE:**

module two\_output\_function(input A, B, C, D, output Y1, Y2);

assign Y1 = A | (B & C) | ~D;

assign Y2 = A & B & ~C & D;

endmodule

TESTBENCH:

module testbench;

reg A, B, C, D;

wire Y1, Y2;

two\_output\_function uut (.A(A), .B(B), .C(C), .D(D), .Y1(Y1), .Y2(Y2));

initial begin

$monitor("A=%b, B=%b, C=%b, D=%b -> Y1=%b, Y2=%b", A, B, C, D, Y1, Y2);

A = 0; B = 0; C = 0; D = 0;

#10;

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

#10;

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

#10;

A = 1; B = 1; C = 1; D = 1;

#10;

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

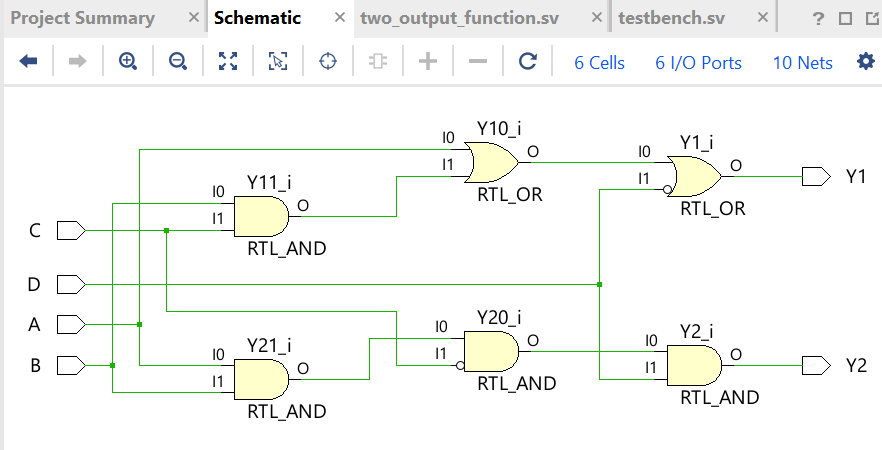
#10;

$finish;

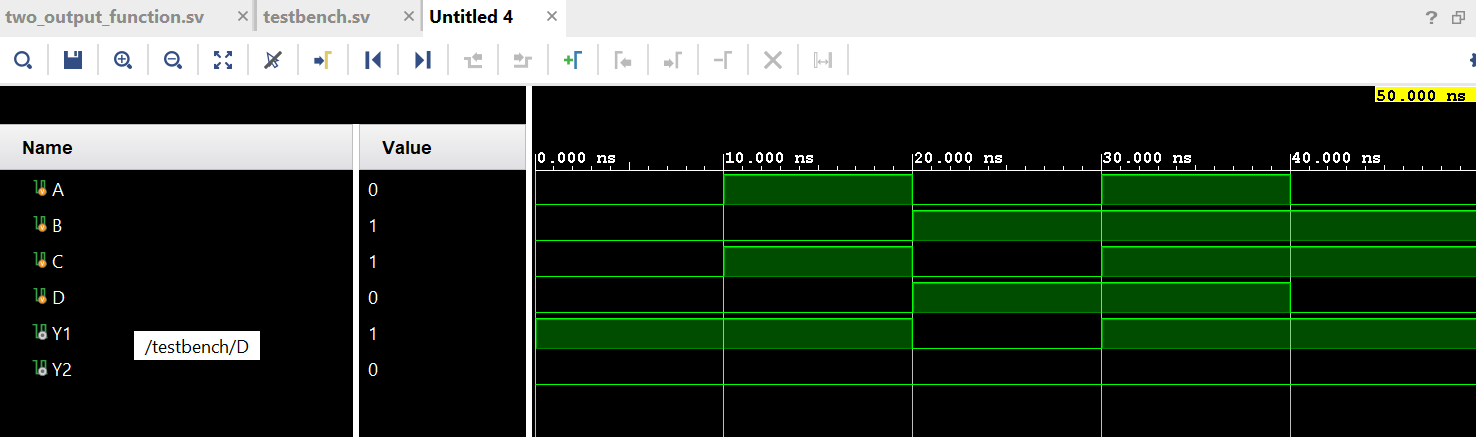
end

endmodule

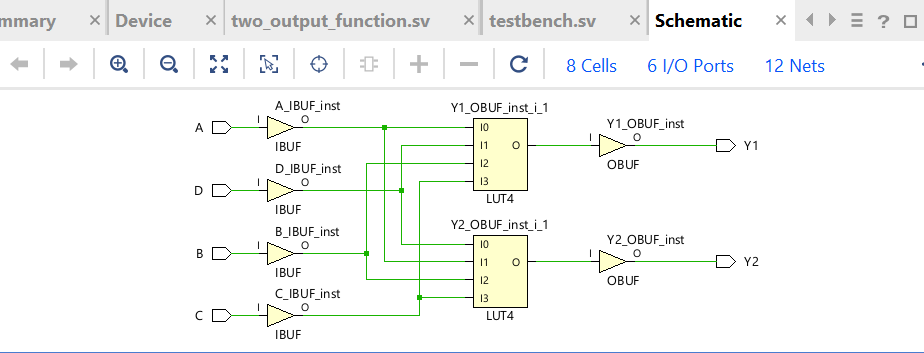
RTL:



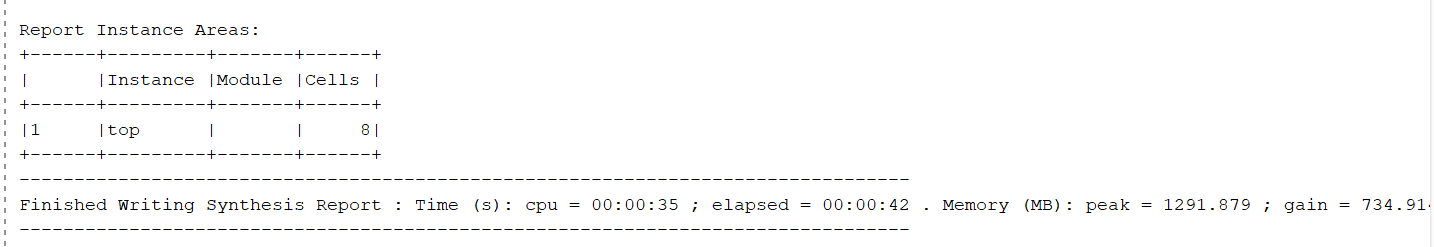
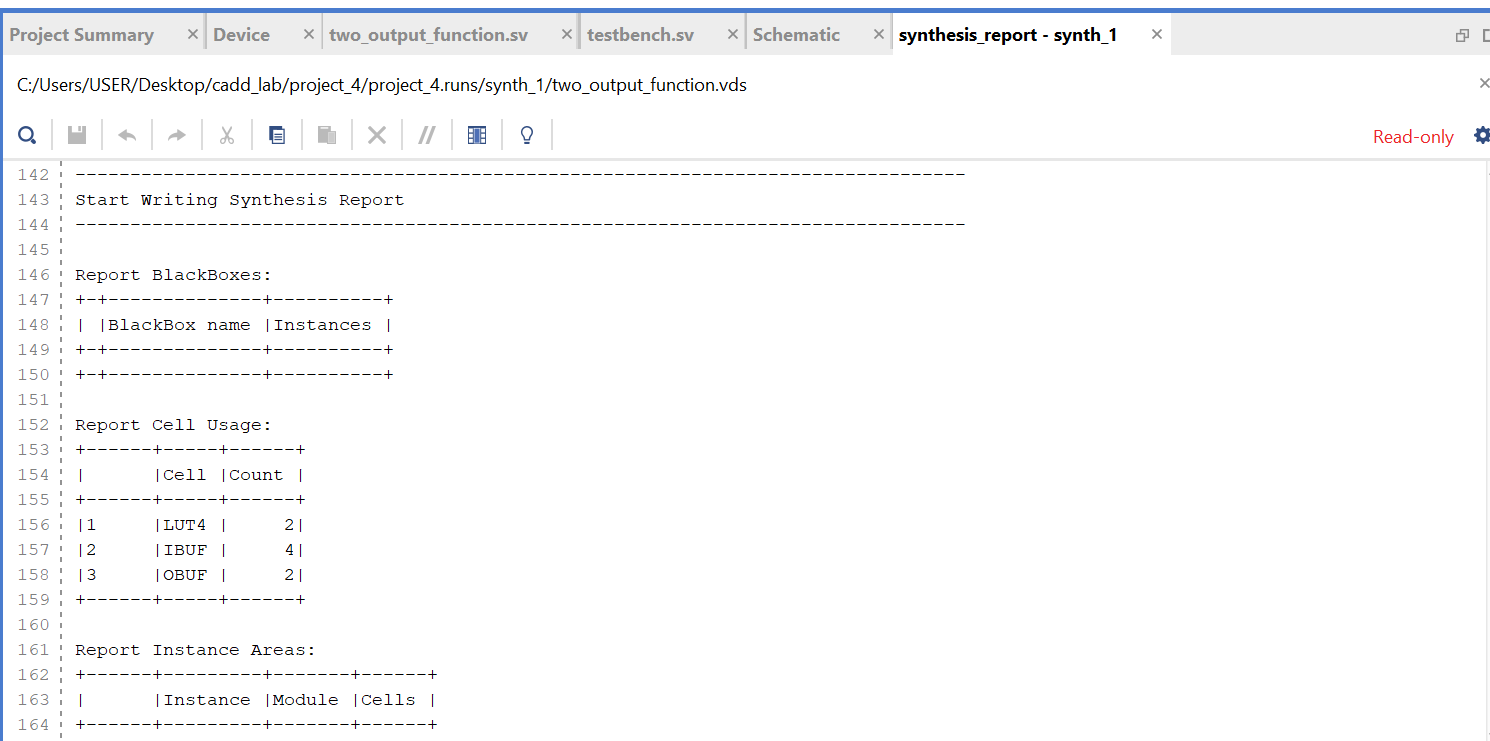
WAVEFORM;



TECHNOLOGY SCREENSHOT:



SYMTHESIS REPORT:



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**(d) Prime Number and Divisibility Logic**

**Design Code :**

**module divisibility(input [3:0] A, output P, D);**

**assign P = (A == 4'd2) | (A == 4'd3) | (A == 4'd5) |**

**(A == 4'd7) | (A == 4'd11) | (A == 4'd13);**

**assign D = (A % 3 == 0);**

**endmodule**

**TESTBENCH:**

**module testbench;**

**reg [3:0] A;**

**wire P, D;**

**divisibility uut (.A(A), .P(P), .D(D));**

**initial begin**

**$monitor("A=%b (%d) -> P=%b (Prime), D=%b (Divisible by 3)", A, A, P, D);**

**A = 4'd0; #10;**

**A = 4'd3; #10;**

**A = 4'd5; #10;**

**A = 4'd9; #10;**

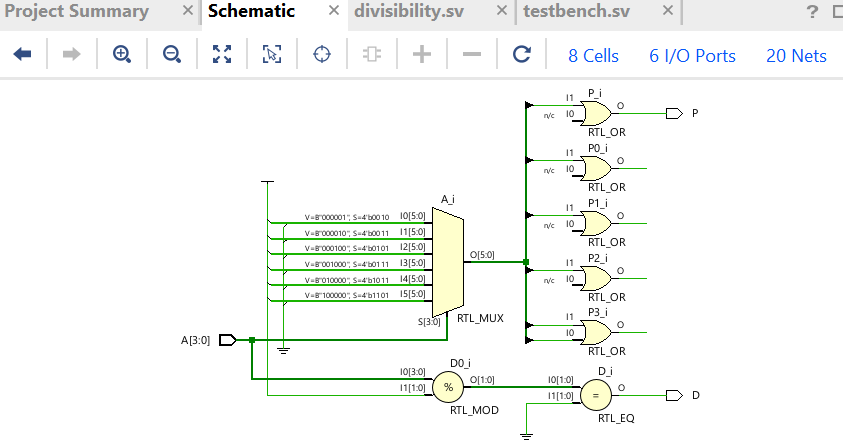
**A = 4'd14; #10;**

**$finish;**

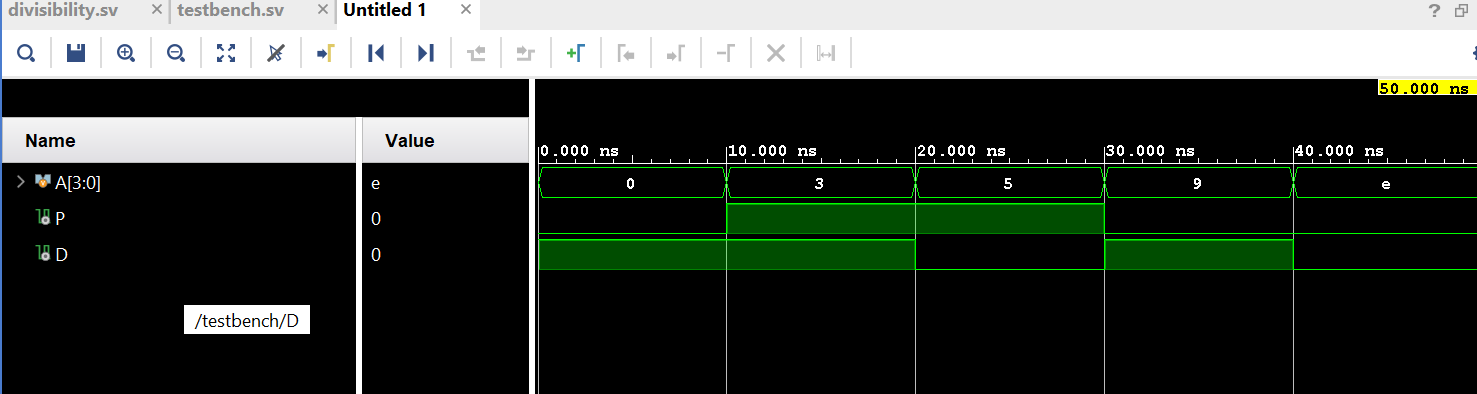
**end**

**endmodule**

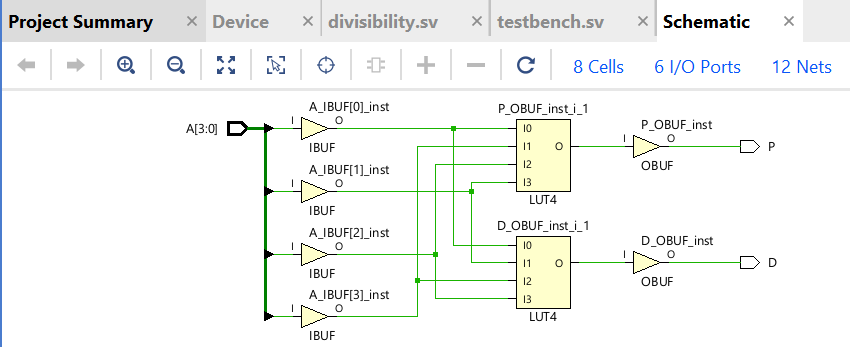
RTL:

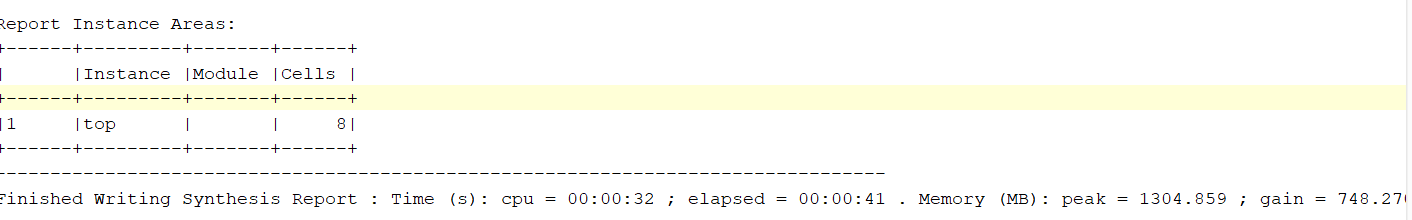
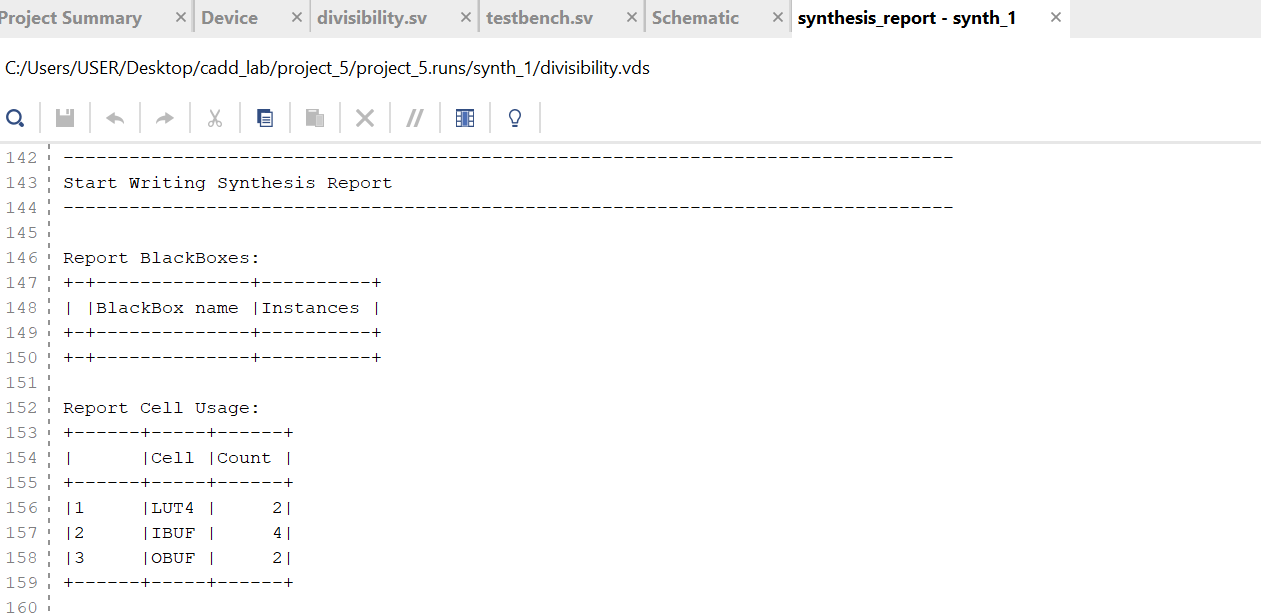


WAVEFORM:



TECNOLOGY SCREENSHOT:

SYNTHESIS REPORT:



(e): 4-Bit Priority Encoder

DESIGN CODE:

module priority\_encoder\_4(input [3:0] A, output logic [1:0] Y, output reg valid);

always @(\*) begin

valid = (A != 0);

casez (A)

4'b1???: Y = 2'b11;

4'b01??: Y = 2'b10;

4'b001?: Y = 2'b01;

4'b0001: Y = 2'b00;

default: Y = 2'b00;

endcase

end

endmodule

TESTBENCH

module testbench;

reg [3:0] A;

wire [1:0] Y;

wire valid;

priority\_encoder\_4 uut (.A(A), .Y(Y), .valid(valid));

initial begin

$monitor("A=%b -> Y=%b, Valid=%b", A, Y, valid);

A = 4'b0000; #10;

A = 4'b0001; #10;

A = 4'b1000; #10;

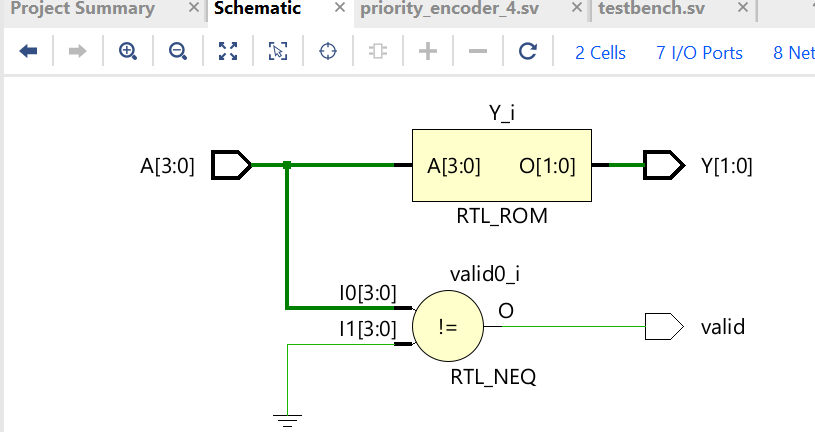
A = 4'b1010; #10;

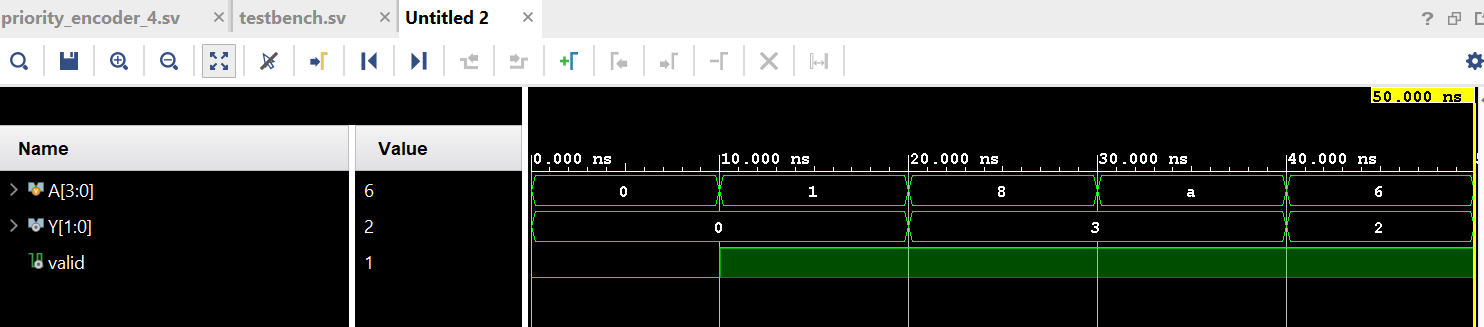
A = 4'b0110; #10;

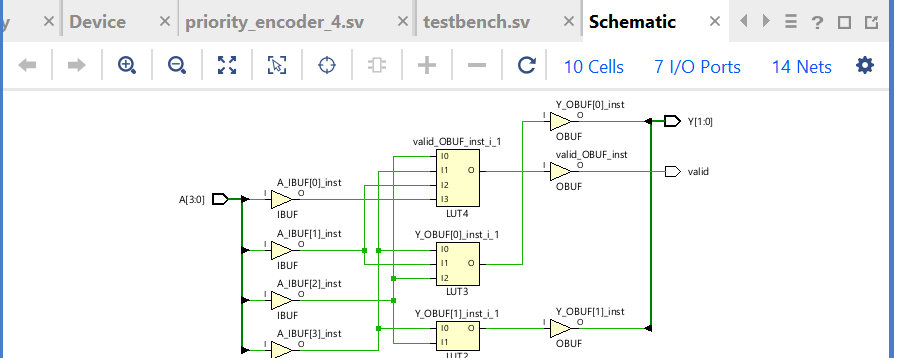
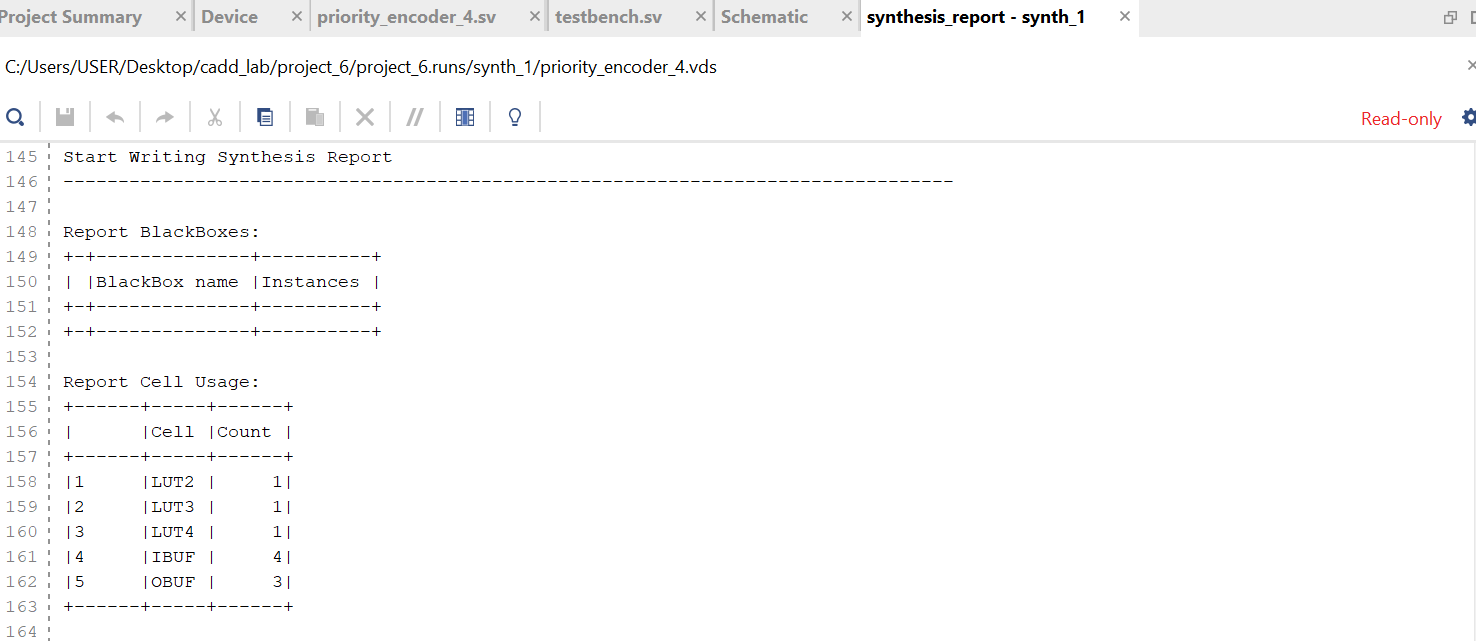
$finish;

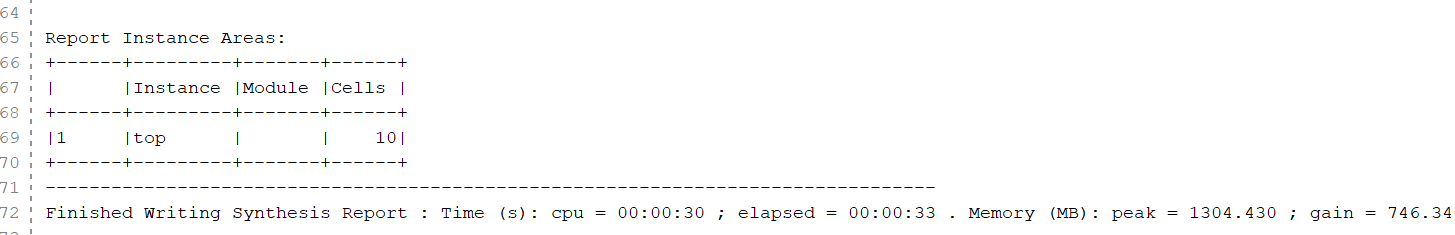
end

endmodule

RTL: WAVEFORM:

TECHNOLOGY SCREENSHOT:

SYNTHESIS REPORT: 



(f) 8-Bit Priority Encode

module priority\_encoder\_8(input [7:0] A, output logic [2:0] Y, output reg valid);

always @(\*) begin

valid = (A != 0);

casez (A)

8'b1???????: Y = 3'b111;

8'b01??????: Y = 3'b110;

8'b001?????: Y = 3'b101;

8'b0001????: Y = 3'b100;

8'b00001???: Y = 3'b011;

8'b000001??: Y = 3'b010;

8'b0000001?: Y = 3'b001;

8'b00000001: Y = 3'b000;

default: Y = 3'b000;

endcase

end

endmodule

TESTBENCH:

module testbench;

reg [7:0] A;

wire [2:0] Y;

wire valid;

priority\_encoder\_8 uut (.A(A), .Y(Y), .valid(valid));

initial begin

$monitor("A=%b -> Y=%b, Valid=%b", A, Y, valid);

A = 8'b00000000; #10;

A = 8'b00000001; #10;

A = 8'b10000000; #10;

A = 8'b01001000; #10;

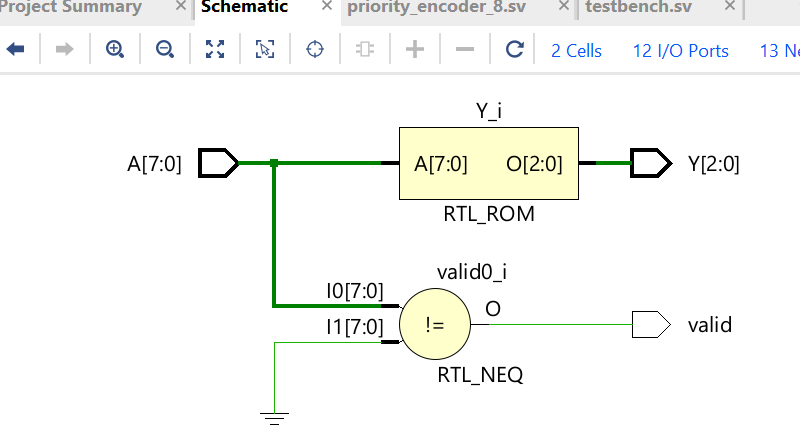
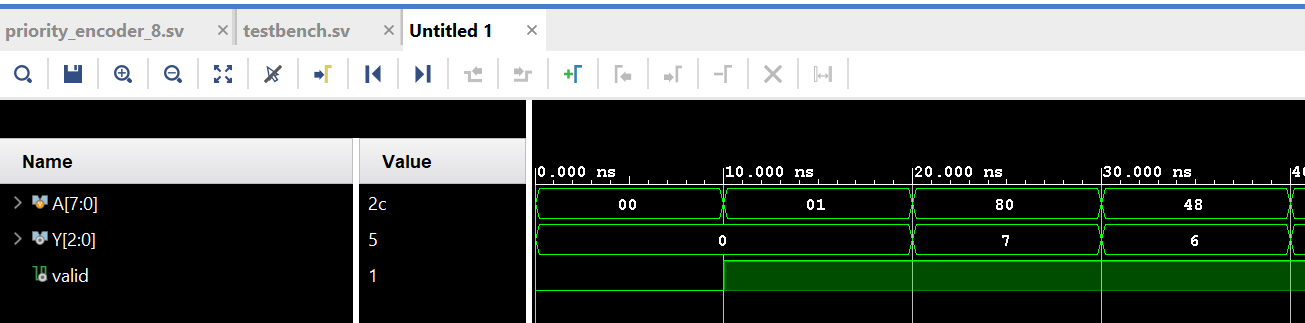
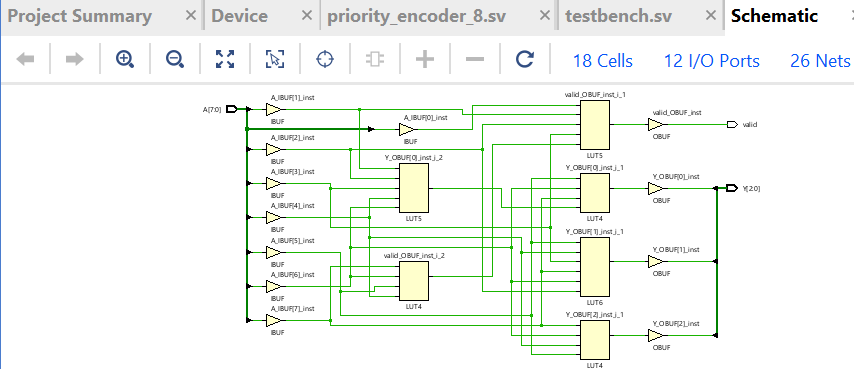
A = 8'b00101100; #10;

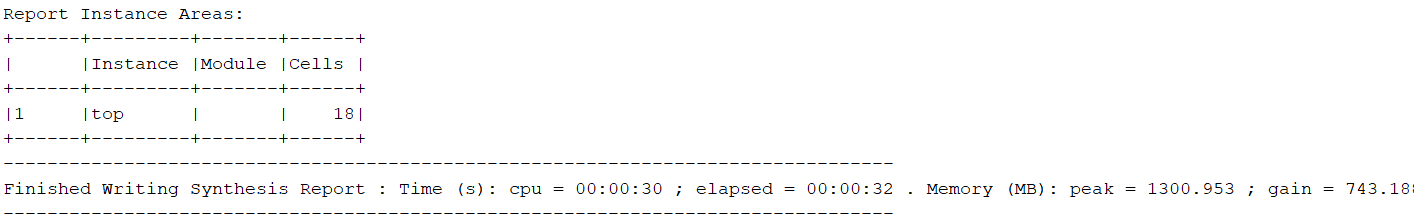
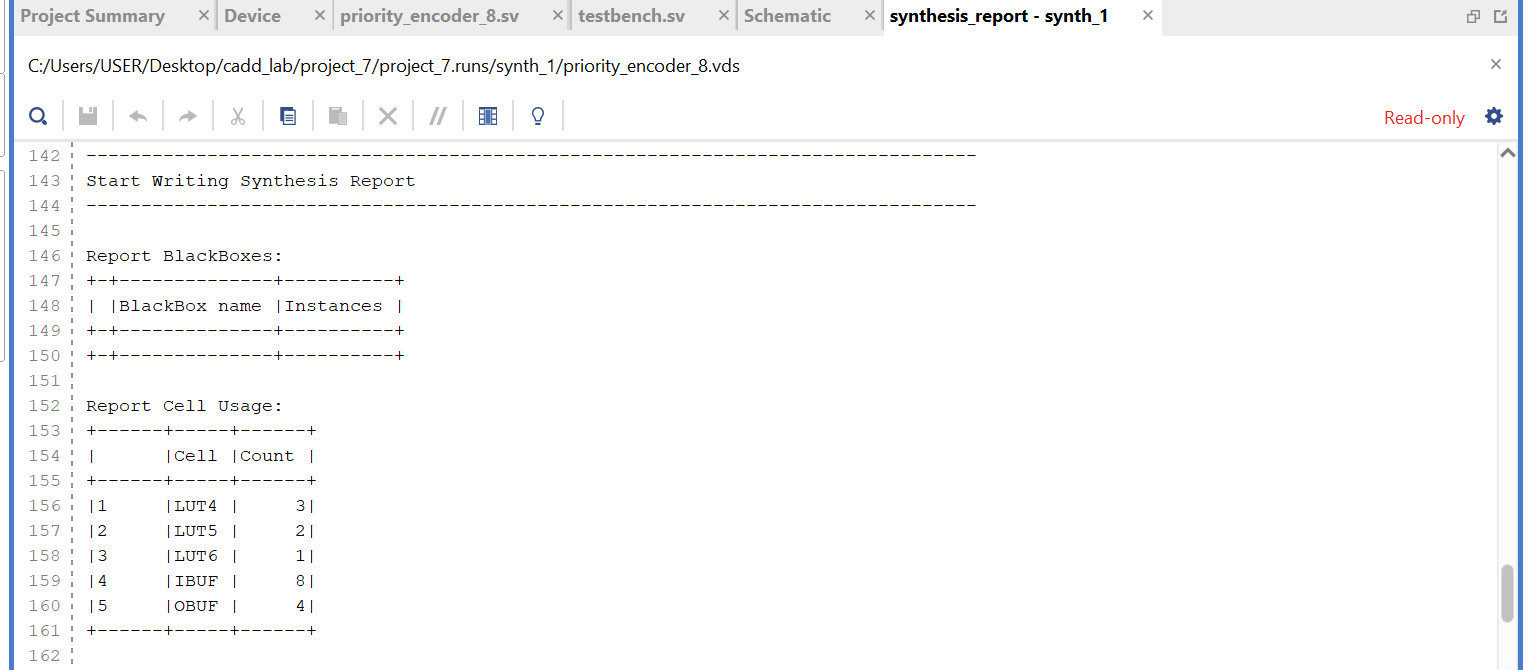
$finish;

end

endmodule

RTL:

WAVEFORM: TECHNOLOGY SCREESHOT

SYNTHESIS REPORT:

(i) FSM

Design code :

module fsm(input clk, reset, X, output logic [1:0] state);

parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;

always @(posedge clk or posedge reset) begin

if (reset)

state <= S0;

end

else begin

case (state)

S0: state <= X ? S1 : S0;

S1: state <= X ? S2 : S0;

S2: state <= X ? S2 : S1;

endcase

end

endmodule

TESTBENCH:

module testbench;

reg clk, reset, X;

wire [1:0] state;

fsm uut (.clk(clk), .reset(reset), .X(X), .state(state));

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

$monitor("Time=%0t | Reset=%b, X=%b -> State=%b", $time, reset, X, state);

reset = 1; X = 0; #10;

reset = 0; X = 1; #10;

X = 1; #10;

X = 0; #10;

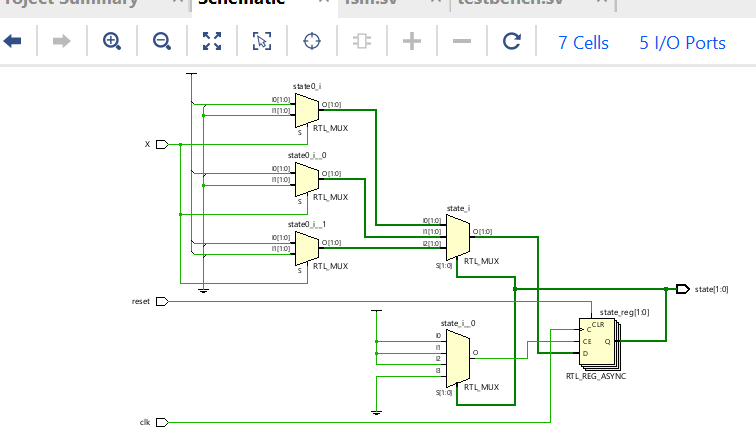
X = 0; #10;

$finish;

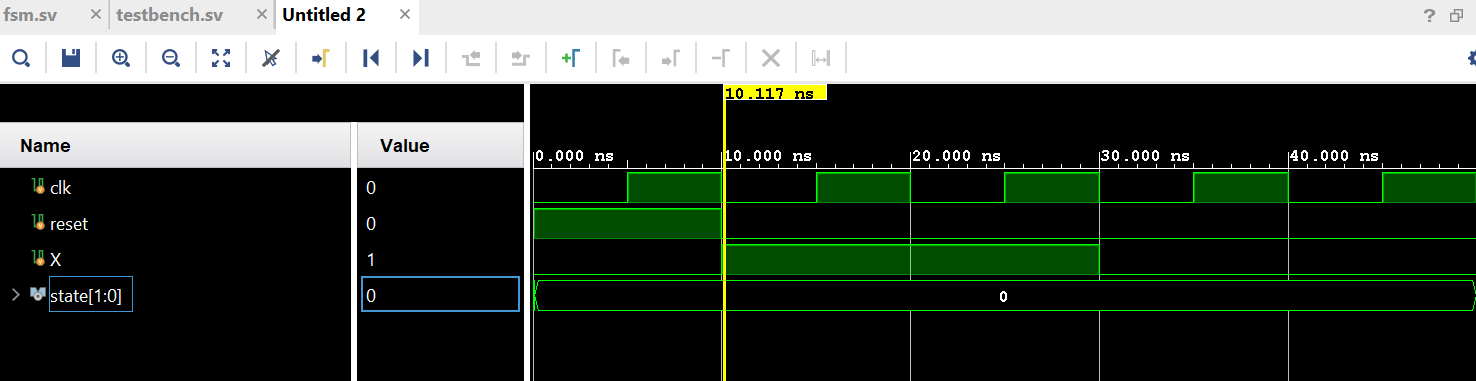
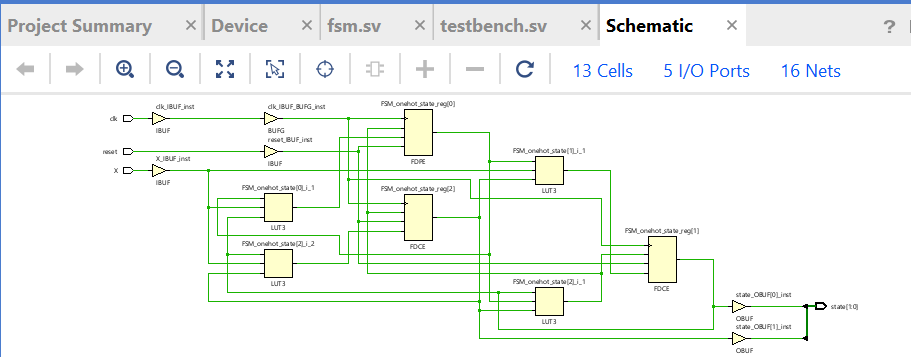
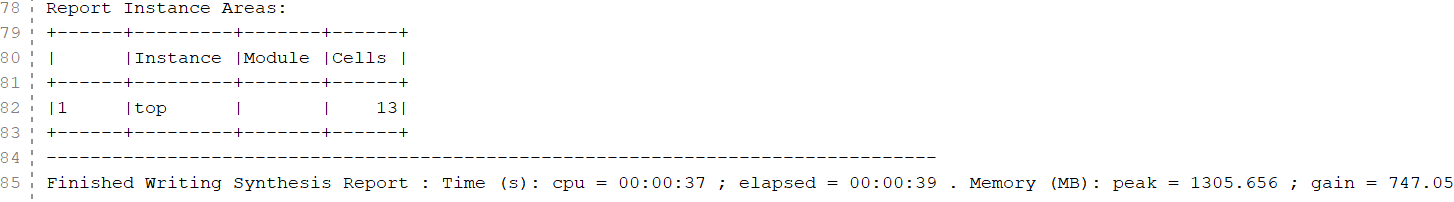
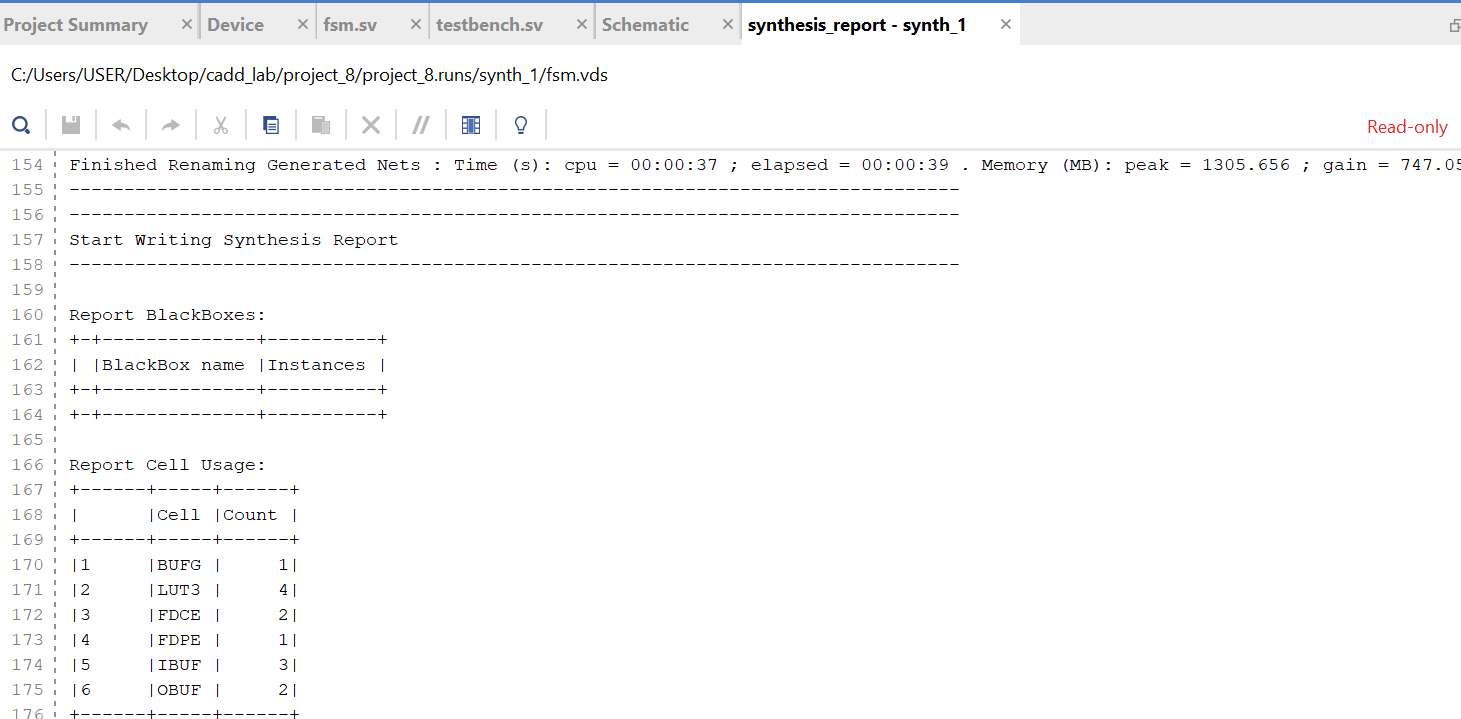
end

endmodule

RTL:



WAVEFORM:

TECHNOLOGY SCREENSHOT: SYNTHESIS REPORT:

(g) 3-to-8 Decoder

Design code;

module decoder\_3to8(input [2:0] A, output logic [7:0] Y);

always @(\*) begin

Y = 8'b00000000;

case (A)

3'b000: Y[0] = 1;

3'b001: Y[1] = 1;

3'b010: Y[2] = 1;

3'b011: Y[3] = 1;

3'b100: Y[4] = 1;

3'b101: Y[5] = 1;

3'b110: Y[6] = 1;

3'b111: Y[7] = 1;

endcase

end

endmodule

TESTBENCH:

module testbench;

reg [2:0] A;

wire [7:0] Y;

decoder\_3to8 uut (.A(A), .Y(Y));

initial begin

$monitor("A=%b -> Y=%b", A, Y);

A = 3'b000; #10;

A = 3'b001; #10;

A = 3'b010; #10;

A = 3'b100; #10;

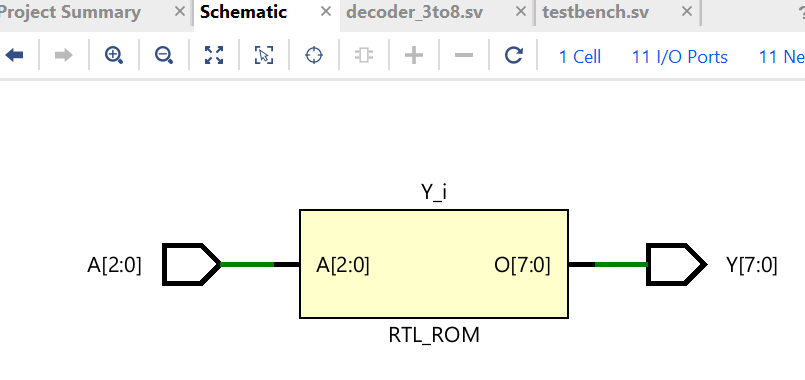
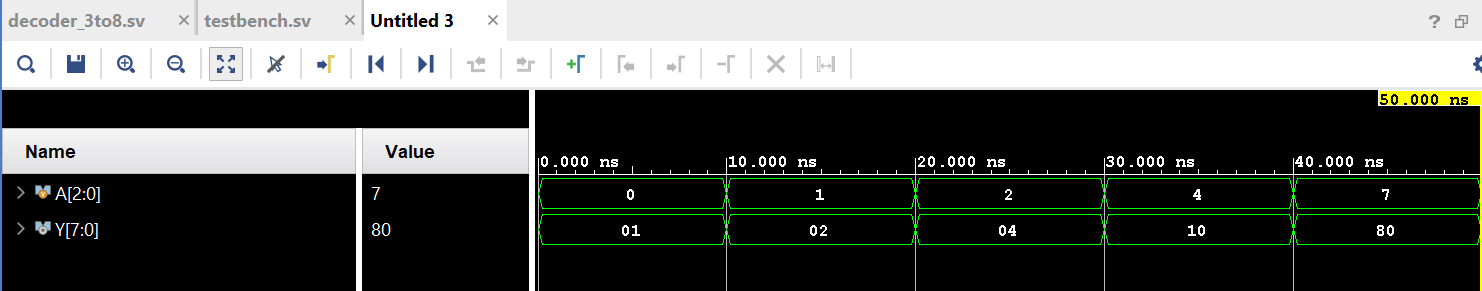
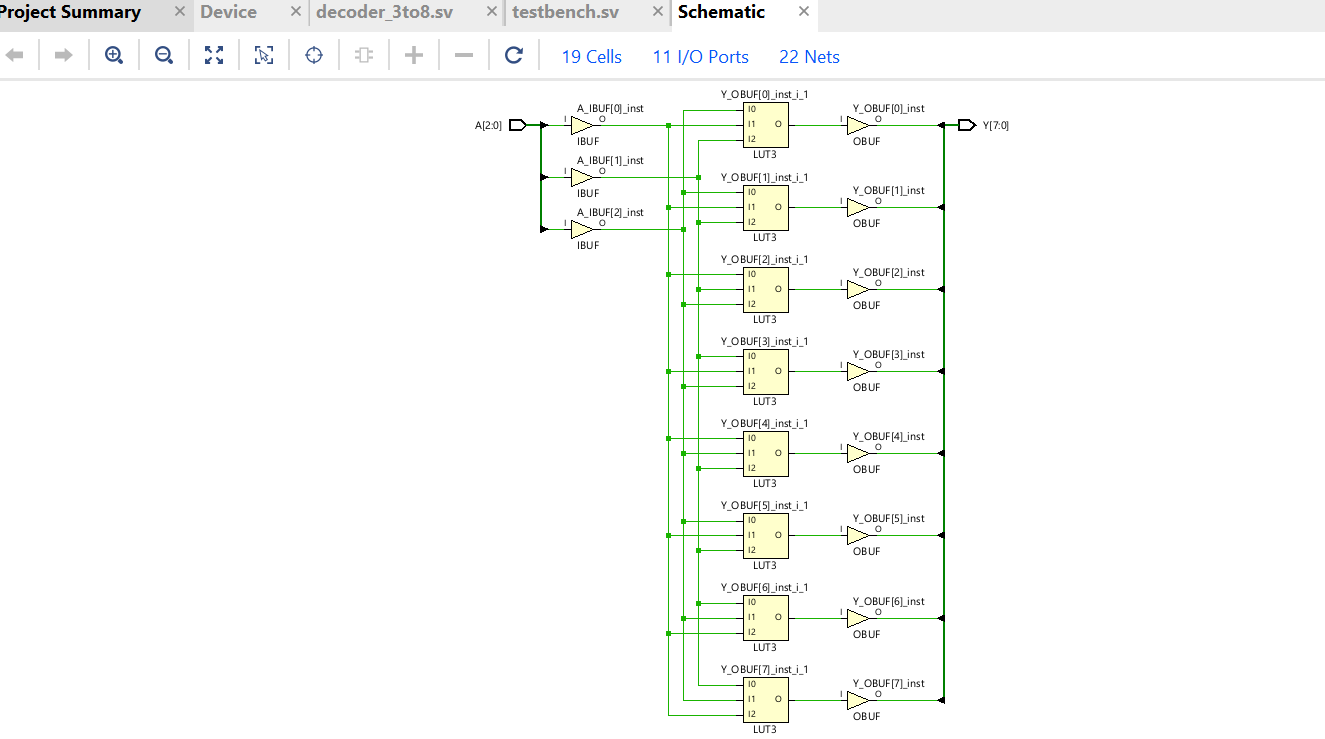
A = 3'b111; #10;

$finish;

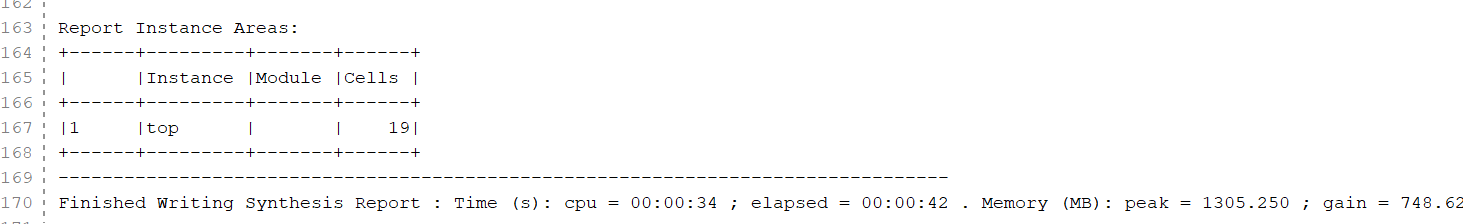
end

endmodule

RTL:

WAVEFORM:TECHNOLOGY SCREENSHOT:

SYNTHESIS REPORT:



(h) A 4-Bit carry propagate adder(with no carry in)

Design code

module carry\_propagate\_adder\_4bit\_no\_carry(input [3:0] A, B, output [3:0] Sum);

assign Sum = A + B;

endmodule

TESTBENCH:

module testbench;

reg [3:0] A, B;

wire [3:0] Sum;

carry\_propagate\_adder\_4bit\_no\_carry uut (.A(A), .B(B), .Sum(Sum));

initial begin

$monitor("A=%b (%d), B=%b (%d) -> Sum=%b (%d)", A, A, B, B, Sum, Sum);

A = 4'b0001; B = 4'b0010; #10;

A = 4'b0101; B = 4'b0011; #10;

A = 4'b1001; B = 4'b0110; #10;

A = 4'b1111; B = 4'b0001; #10;

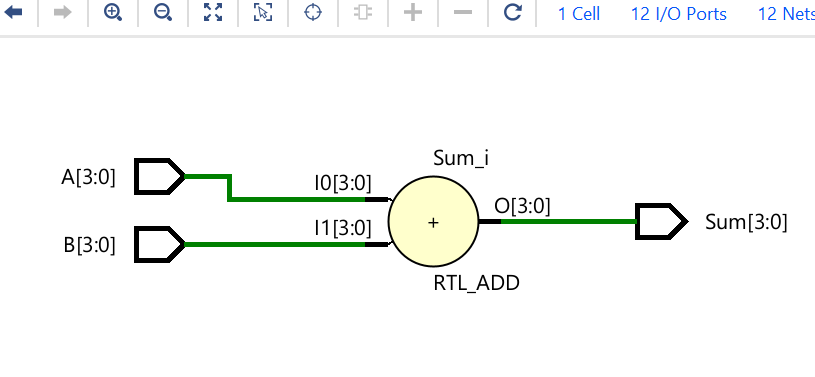
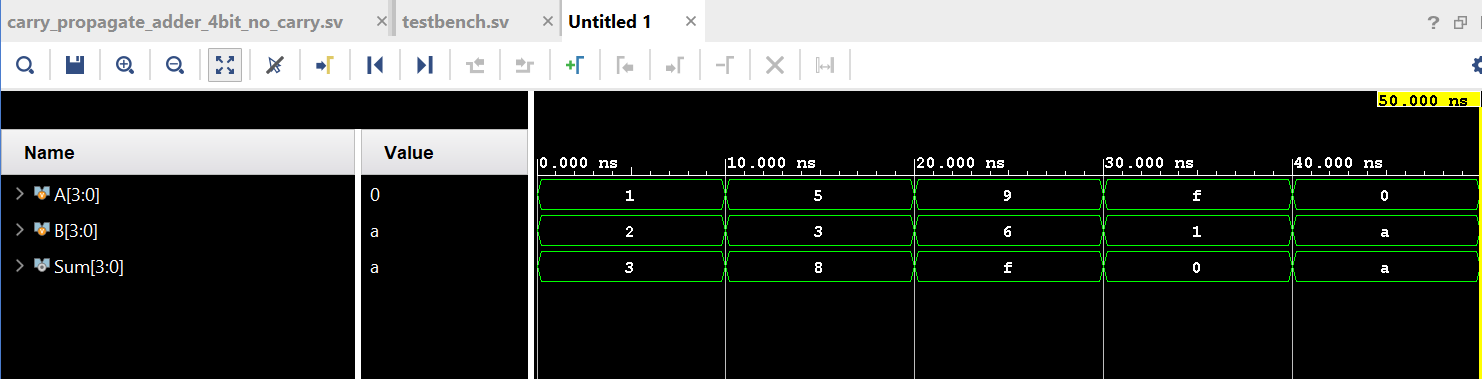
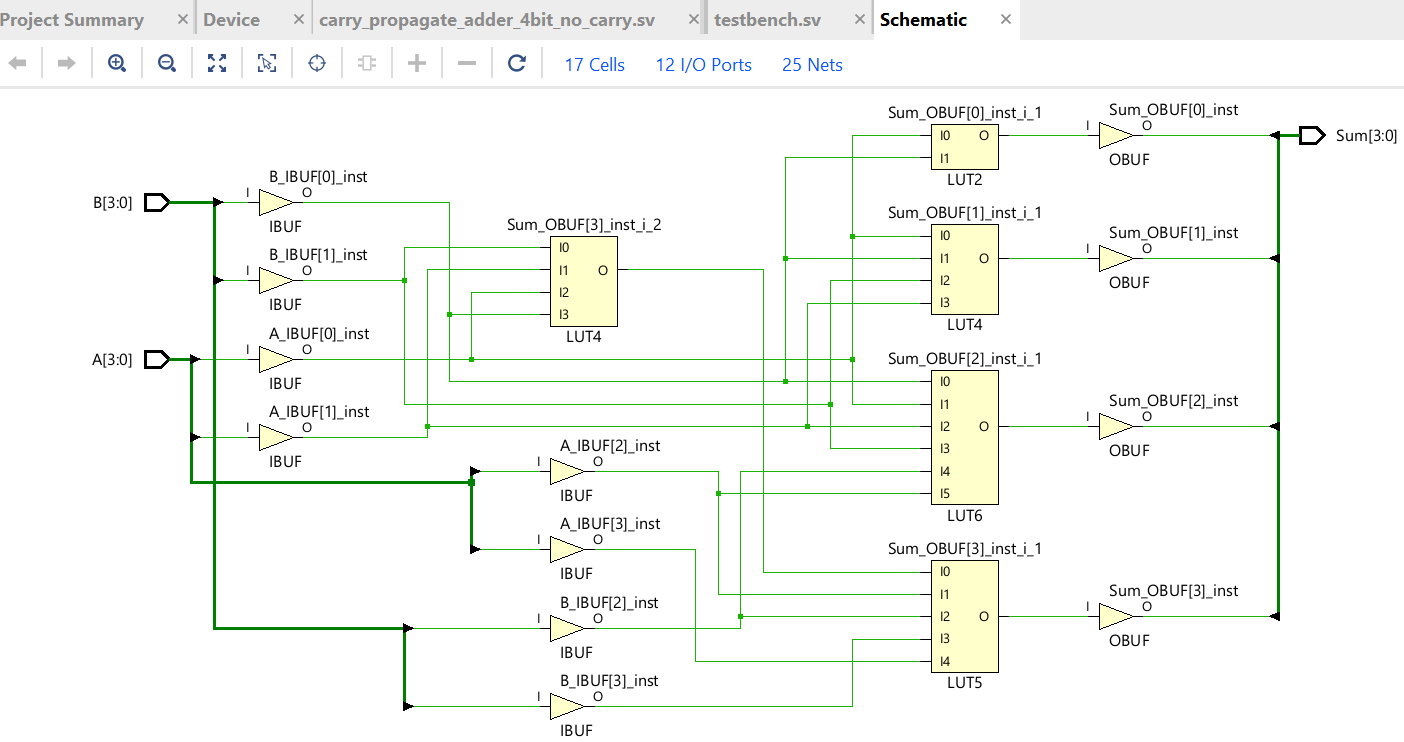
A = 4'b0000; B = 4'b1010; #10;

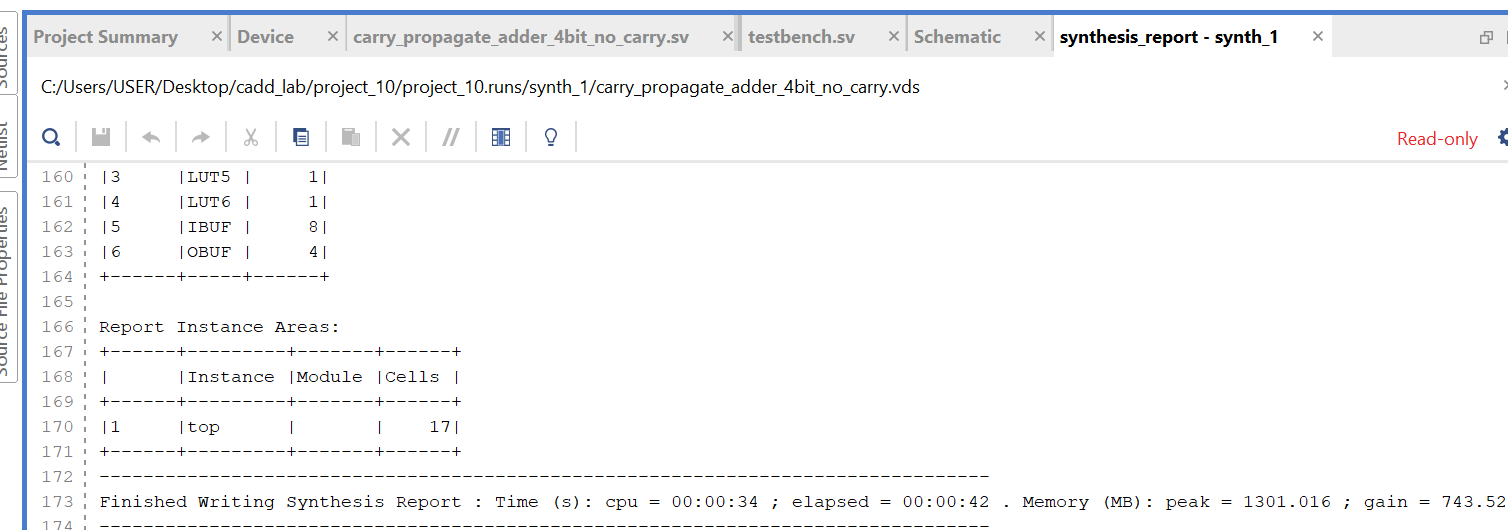
$finish;

end

endmodule

RTL:

WAVEFORM: TECHNOLOGY SCREENSHOT: 

SYNTHESIS REPORT: 

(j)The Gray code counter

DESIGN CODE:

module gray\_code\_counter (

input clk,

input reset,

output logic [2:0] gray\_code

);

logic [2:0] binary\_count;

always @(posedge clk or posedge reset) begin

if (reset) begin

binary\_count <= 3'b000;

end

else

begin

binary\_count <= binary\_count + 1;

end

end

always @(\*) begin

gray\_code = binary\_count ^ (binary\_count >> 1);

end

endmodule

TESTBENCH:

module testbench;

reg clk;

reg reset;

wire [2:0] gray\_code;

gray\_code\_counter uut (

.clk(clk),

.reset(reset),

.gray\_code(gray\_code)

);

always #5 clk = ~clk;

initial begin

clk = 0;

reset = 1;

#10;

reset = 0;

#100;

$stop;

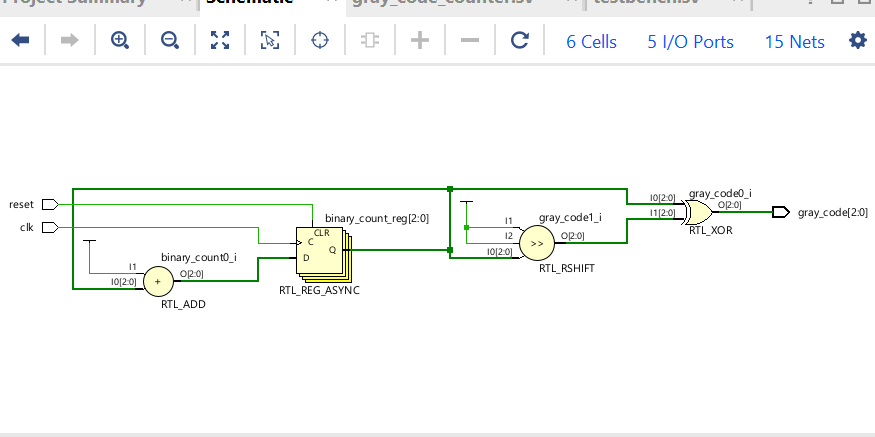
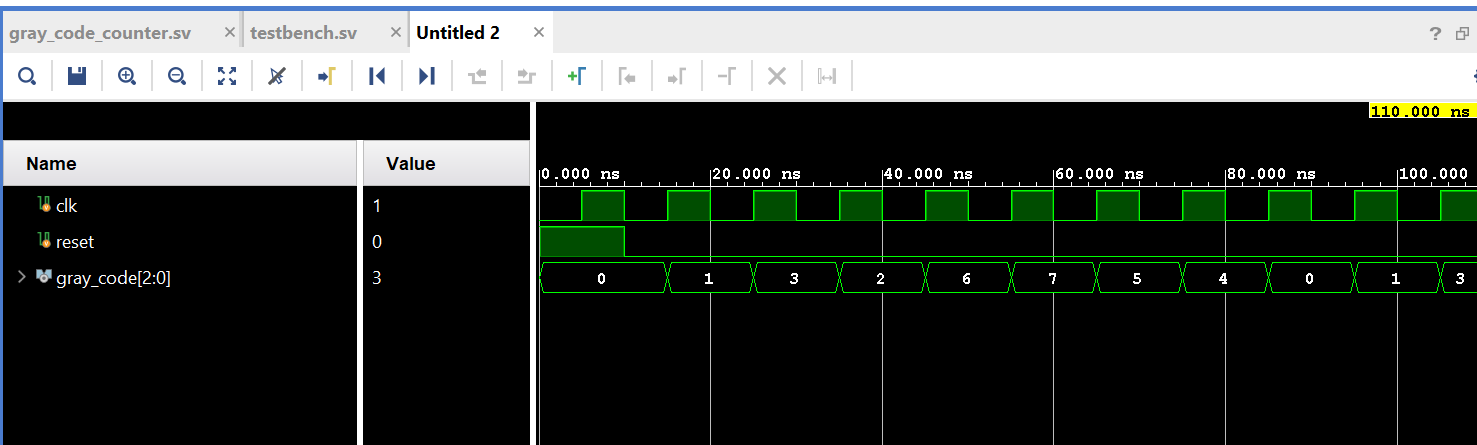
end

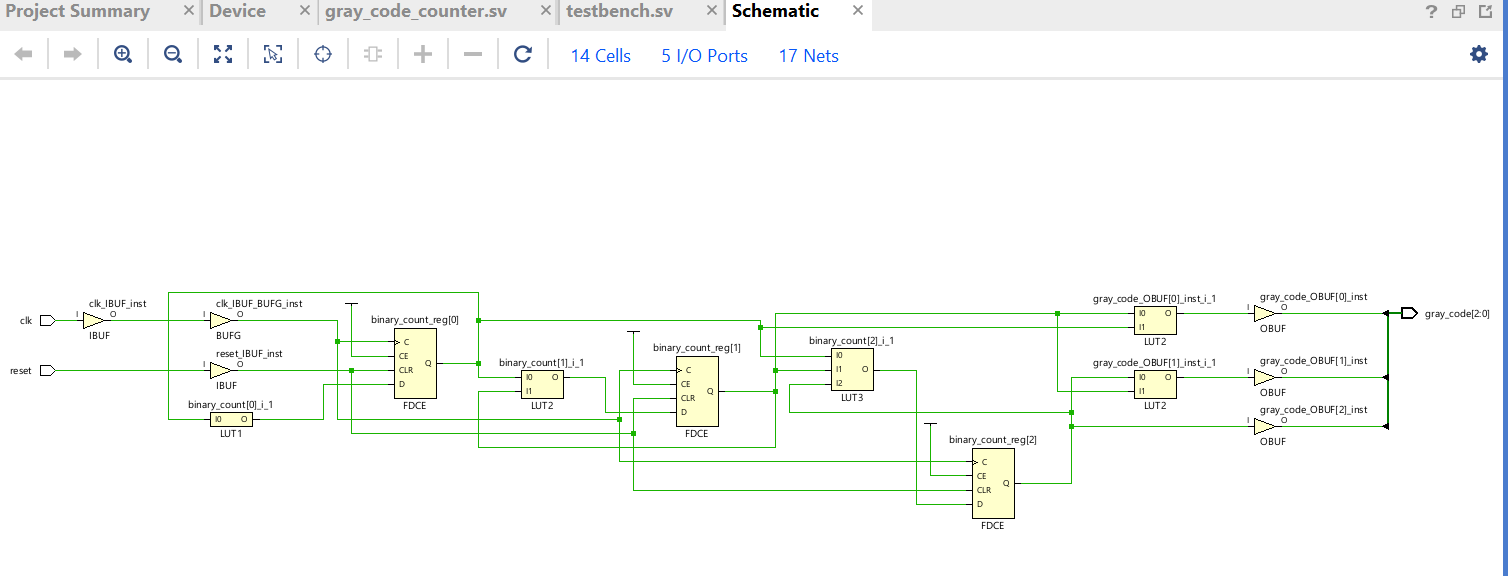
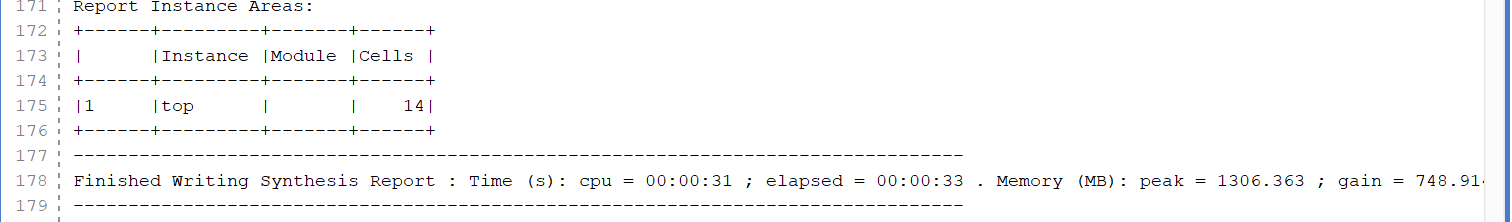
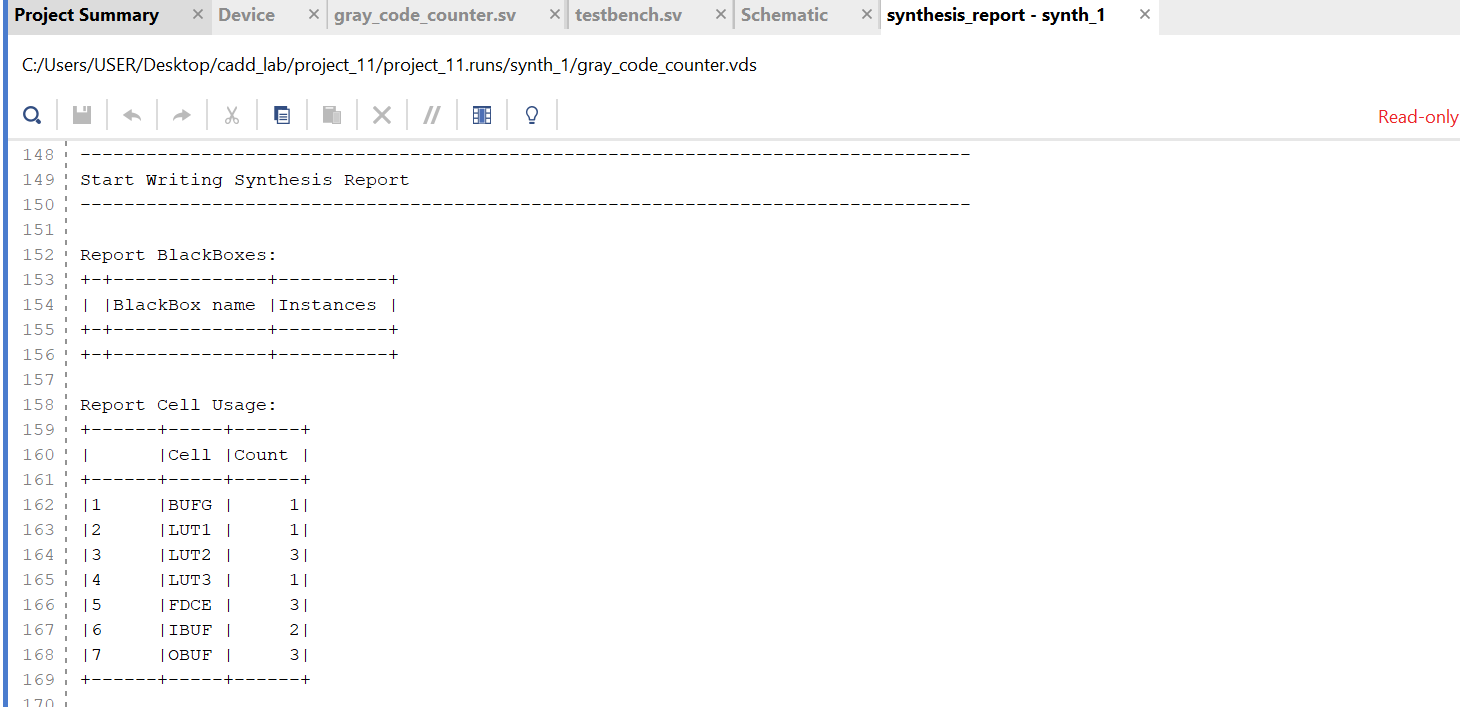
initial begin

$monitor("Time=%0t | Reset=%b | Gray Code=%b", $time, reset, gray\_code);

end

endmodule

RTL:  
WAVEFORM: TECHNOLOGY SCREENSHOT:

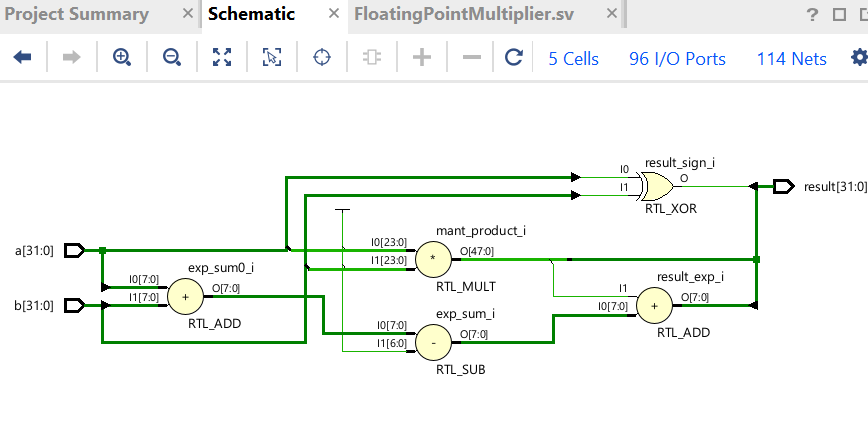
SYNTHESIS REPORT

**32-bit Floating Point Multiplier and Prefix Adder Design**

**32-bit Floating Point Multiplier**

**(a) Steps for 32-bit Floating Point Multiplication**

1. Input Extraction: Extract the sign, exponent, and mantissa bits from both inputs.  
2. Sign Calculation: Compute the result's sign using XOR of the input signs.  
3. Exponent Calculation: Add the input exponents and subtract the bias (127).  
4. Mantissa Multiplication: Multiply the mantissas and normalize the result.  
5. Rounding (Truncate): Discard extra bits beyond 23 bits in the mantissa.  
6. Handle Special Cases: Handle zeros, overflows, and underflows.  
7. Combine Results: Form the final 32-bit result by combining the sign, exponent, and mantissa.

**(b) Schematic Diagram**

**(c) Design code**

module FloatingPointMultiplier (  
 input [31:0] a, b,  
 output [31:0] result  
);  
 wire sign\_a = a[31];  
 wire sign\_b = b[31];  
 wire [7:0] exp\_a = a[30:23];  
 wire [7:0] exp\_b = b[30:23];  
 wire [23:0] mant\_a = {1'b1, a[22:0]};  
 wire [23:0] mant\_b = {1'b1, b[22:0]};  
  
 wire result\_sign = sign\_a ^ sign\_b;  
 wire [8:0] result\_exp = exp\_a + exp\_b - 8'd127;  
 wire [47:0] mant\_product = mant\_a \* mant\_b;  
  
 wire [7:0] final\_exp = result\_exp + mant\_product[47];  
 wire [22:0] final\_mant = mant\_product[46:24];  
  
 assign result = {result\_sign, final\_exp, final\_mant};  
endmodule

testbench:

module FloatingPointMultiplier\_tb;

reg [31:0] a, b;

wire [31:0] result;

FloatingPointMultiplier uut (

.a(a),

.b(b),

.result(result)

);

initial begin

$display("Starting Floating Point Multiplier Testbench");

a = 32'b01000000001000000000000000000000;

b = 32'b01000000010000000000000000000000;

#10;

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

a = 32'b00111111110000000000000000000000;

b = 32'b01000000001000000000000000000000;

#10;

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

a = 32'b00000000000000000000000000000000;

b = 32'b01000000100000000000000000000000;

#10;

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

a = 32'b01111111100000000000000000000000;

b = 32'b01000000000000000000000000000000;

#10;

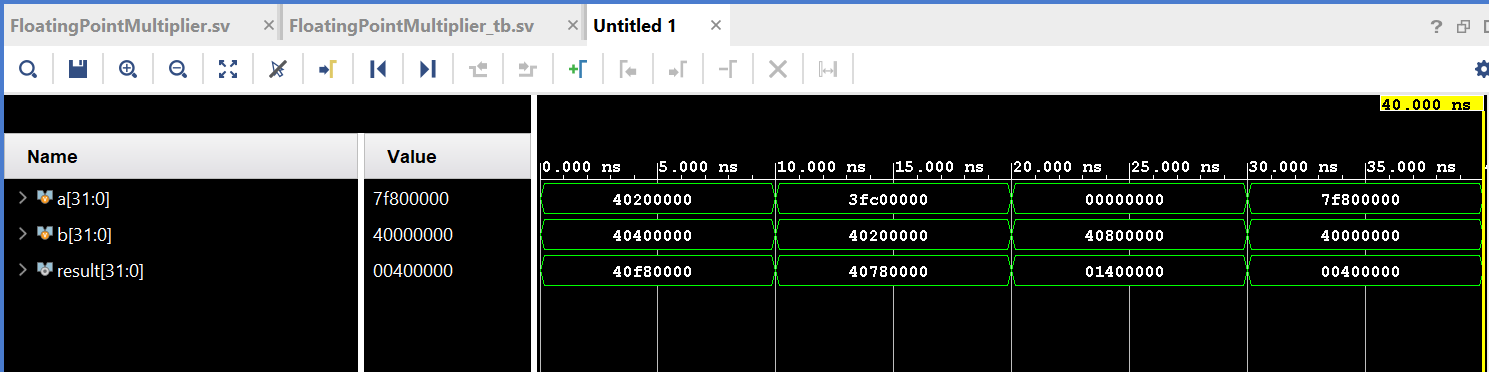
$display("a = %b, b = %b, result = %b", a, b, result);

$finish;

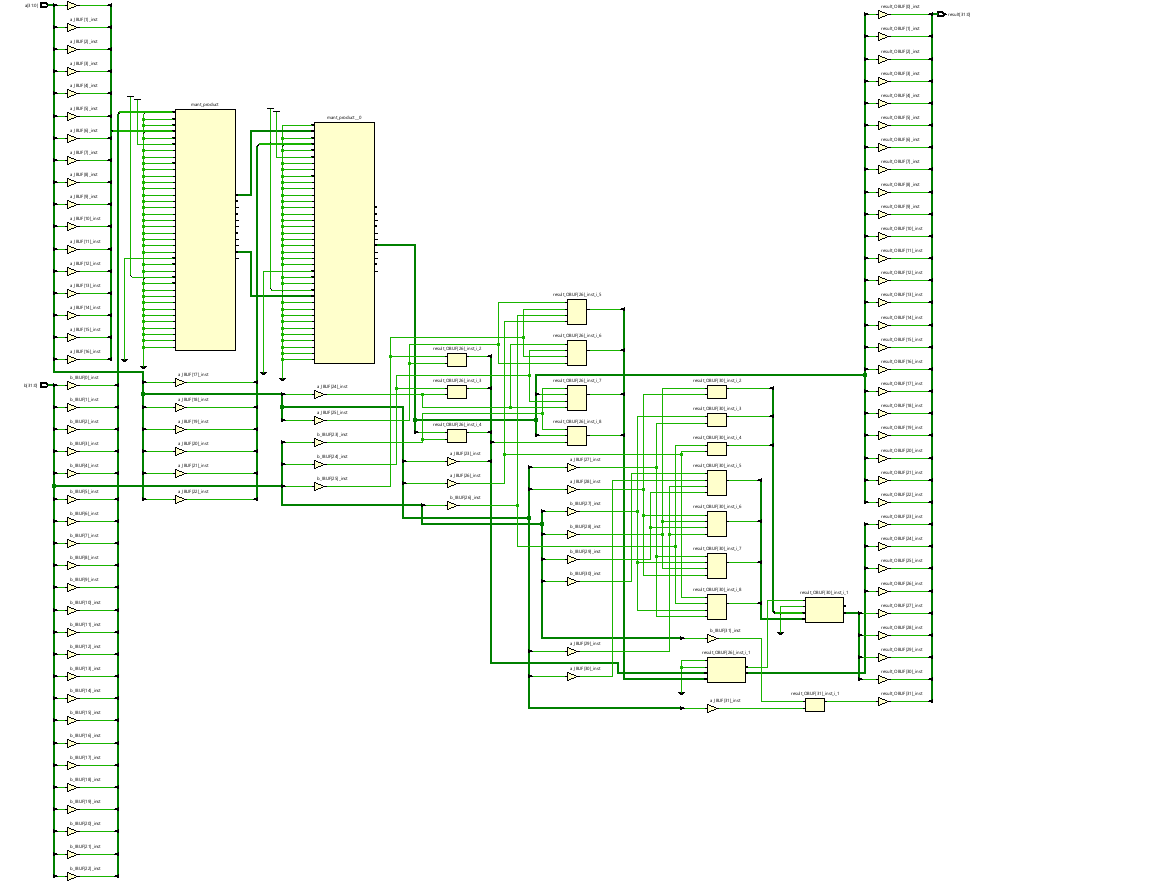
end

endmodule

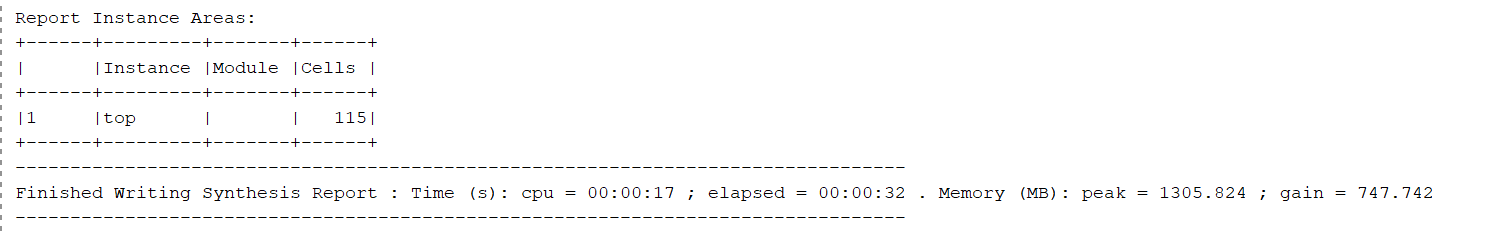
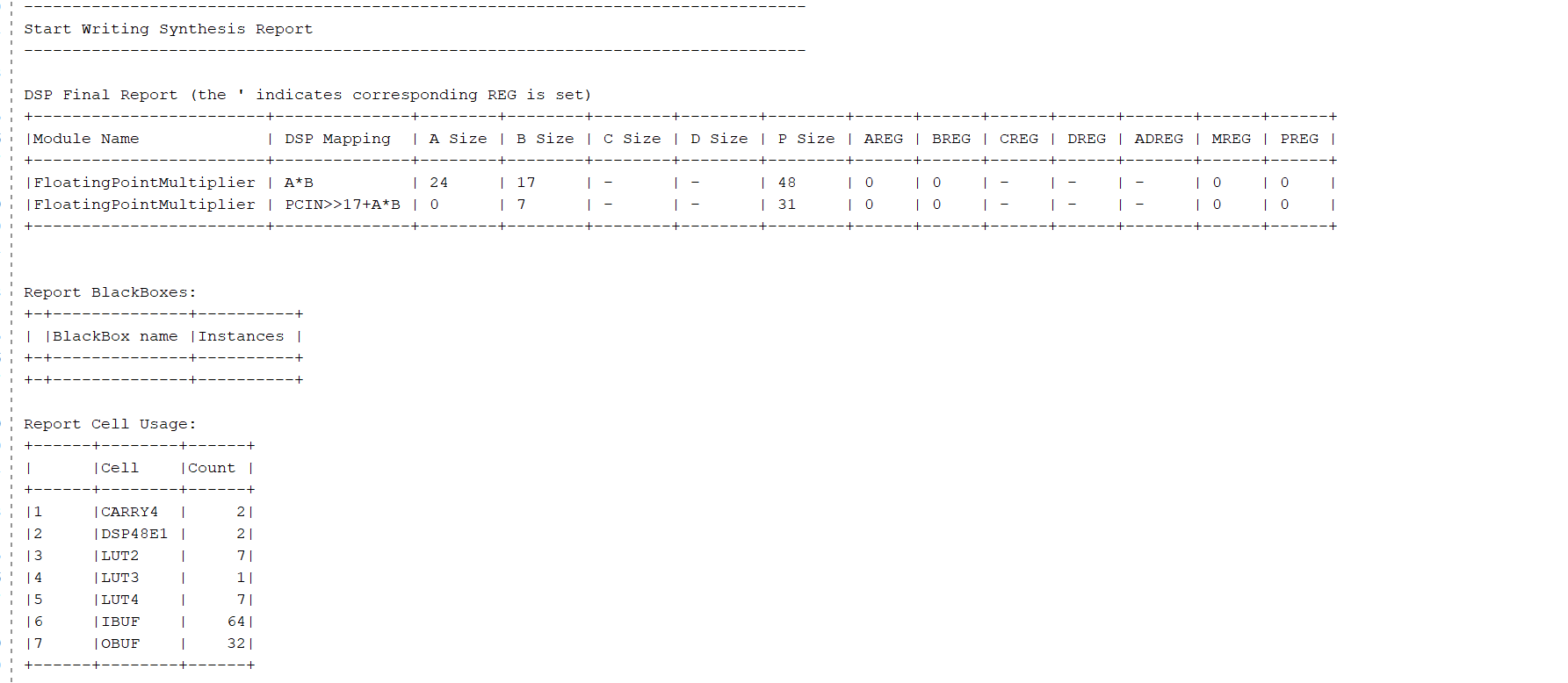
waveform:



Technology:

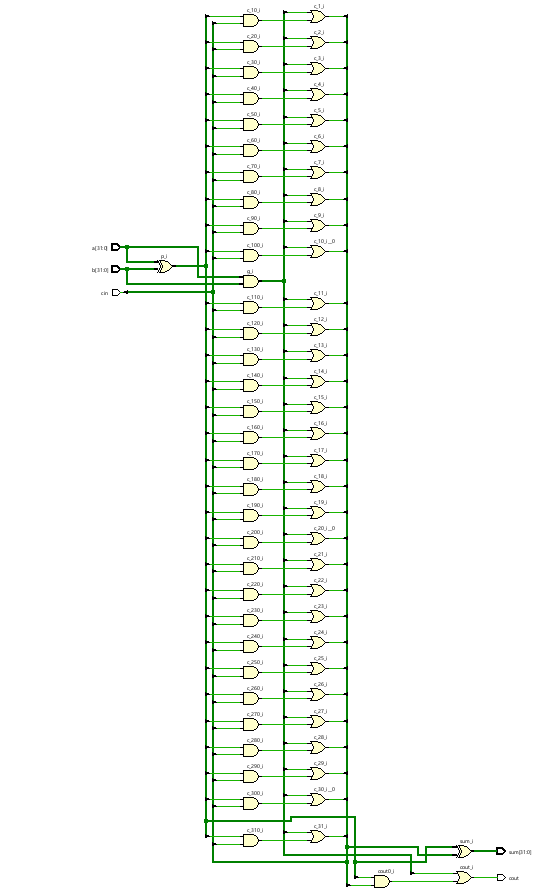


**Synthesis report:**

****

**32-bit Prefix Adder**

1. **Schematic Diagram**

****

**(b) Designcode**

module PrefixAdder (  
 input [31:0] a, b,  
 input cin,  
 output [31:0] sum,  
 output cout  
);  
 wire [31:0] p, g, c;  
  
 assign p = a ^ b;  
 assign g = a & b;  
  
 assign c[0] = cin;  
 genvar i;  
 generate  
 for (i = 1; i < 32; i = i + 1) begin  
 assign c[i] = g[i-1] | (p[i-1] & c[i-1]);  
 end  
 endgenerate  
  
 assign sum = p ^ c;  
 assign cout = g[31] | (p[31] & c[31]);  
endmodule

TEST BENCH:

module PrefixAdder\_tb;

reg [31:0] a, b;

reg cin;

wire [31:0] sum;

wire cout;

PrefixAdder uut (

.a(a),

.b(b),

.cin(cin),

.sum(sum),

.cout(cout)

);

initial begin

$display("Starting Prefix Adder Testbench");

a = 32'h00000001;

b = 32'h00000001;

cin = 0;

#10;

$display("a = %h, b = %h, cin = %b, sum = %h, cout = %b", a, b, cin, sum, cout);

a = 32'hFFFFFFFF;

b = 32'h00000001;

cin = 0;

#10;

$display("a = %h, b = %h, cin = %b, sum = %h, cout = %b", a, b, cin, sum, cout);

a = 32'h00000001;

b = 32'h00000001;

cin = 1;

#10;

$display("a = %h, b = %h, cin = %b, sum = %h, cout = %b", a, b, cin, sum, cout);

a = 32'h80000000;

b = 32'h7FFFFFFF;

cin = 0;

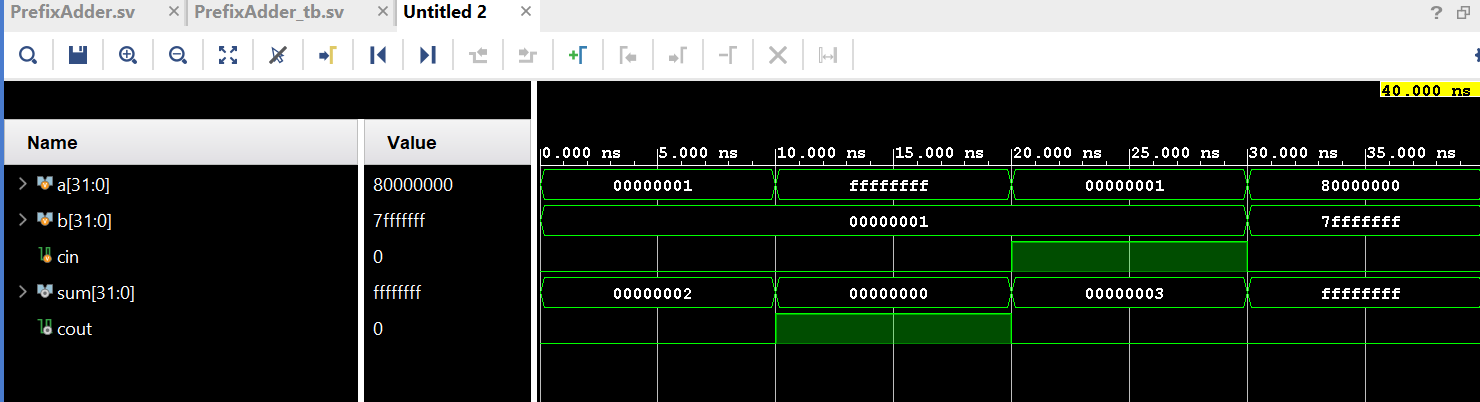
#10;

$display("a = %h, b = %h, cin = %b, sum = %h, cout = %b", a, b, cin, sum, cout);

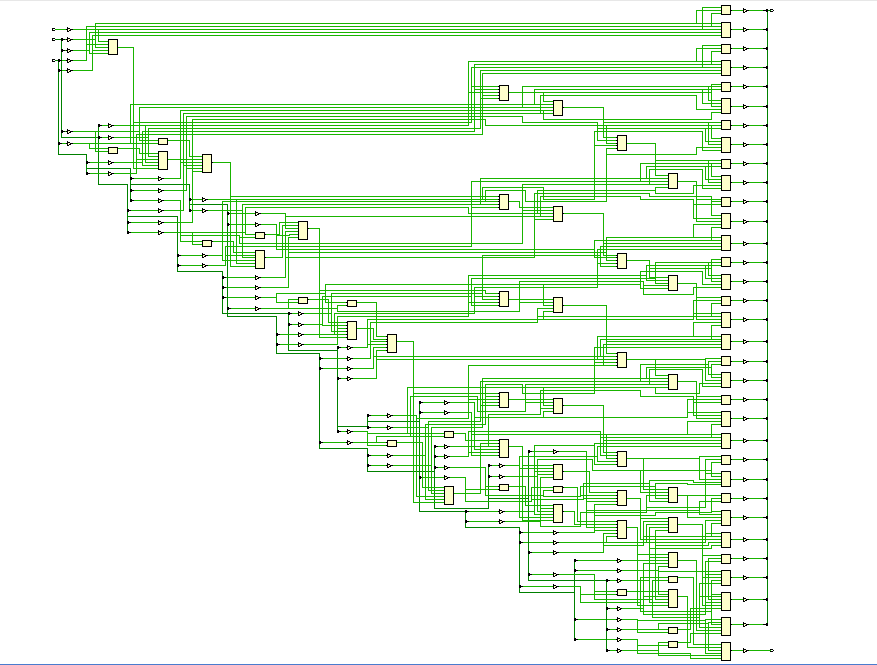
$finish;

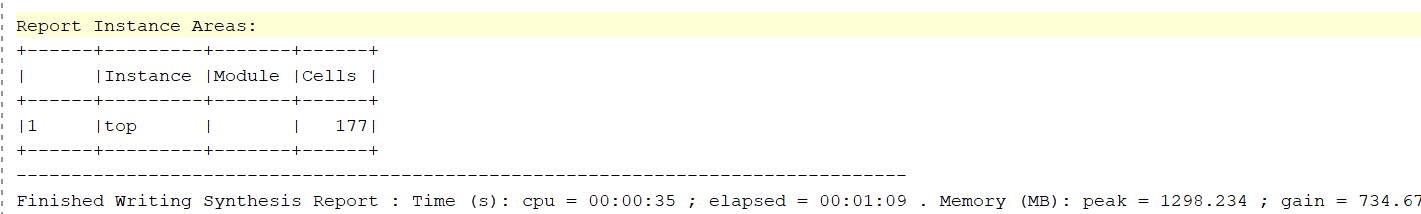
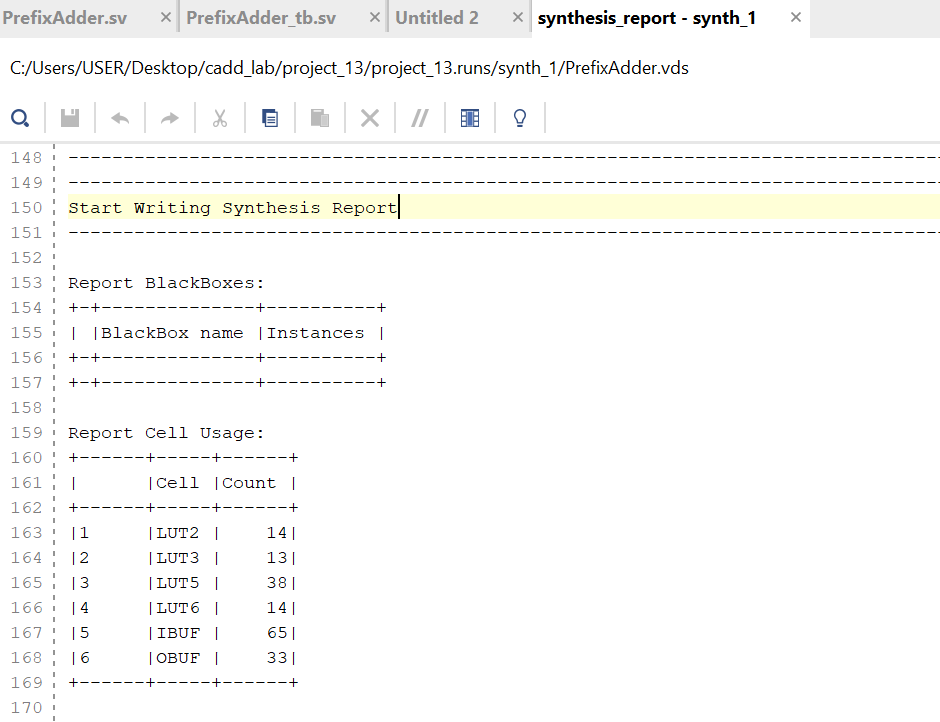
end

endmodule

waveform:

technology:



synthesis report:

**(c) Delay Calculation**

The delay of the prefix adder is calculated as follows:  
- Each stage of the prefix tree has a delay of 100 ps.  
- For a 32-bit adder, the depth of the tree is log2(32) = 5.  
- Total delay = 5 \* 100 ps = 500 ps.

**(d) Pipelined 32-bit Prefix Adder**

* **Maximum speed**:
  + Pipeline stage delay = 100 ps
  + Maximum clock speed = 1/(100 ps)=10 GHz

DESIGN CODE:  
module PipelinedPrefixAdder (

input [31:0] A,

input [31:0] B,

input clk,

output reg [31:0] S

);

reg [31:0] G1, P1;

reg [31:0] C2;

reg [31:0] P2;

always @(posedge clk) begin

G1 <= A & B;

P1 <= A ^ B;

end

always @(posedge clk) begin

C2[0] <= G1[0];

P2 <= P1;

for (int i = 1; i < 32; i = i + 1) begin

C2[i] <= G1[i] | (P1[i] & C2[i-1]);

end

end

always @(posedge clk) begin

for (int i = 0; i < 32; i = i + 1) begin

S[i] <= P2[i] ^ C2[i-1];

end

end

endmodule

TESTBENCH:

module tb\_PipelinedPrefixAdder;

reg [31:0] A;

reg [31:0] B;

reg clk;

wire [31:0] S;

PipelinedPrefixAdder uut (

.A(A),

.B(B),

.clk(clk),

.S(S));

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

A = 0; B = 0;

#10; A = 32'h00000001; B = 32'h00000001;

#10; A = 32'h0000000F; B = 32'h00000001;

#10; A = 32'hFFFFFFFF; B = 32'h00000001;

#10; A = 32'h12345678; B = 32'h87654321;

#10; A = 32'hAAAAAAAA; B = 32'h55555555;

#10; A = 32'h80000000; B = 32'h7FFFFFFF;

#10; A = 32'hFFFFFFFF; B = 32'hFFFFFFFF;

#10; $finish;

end

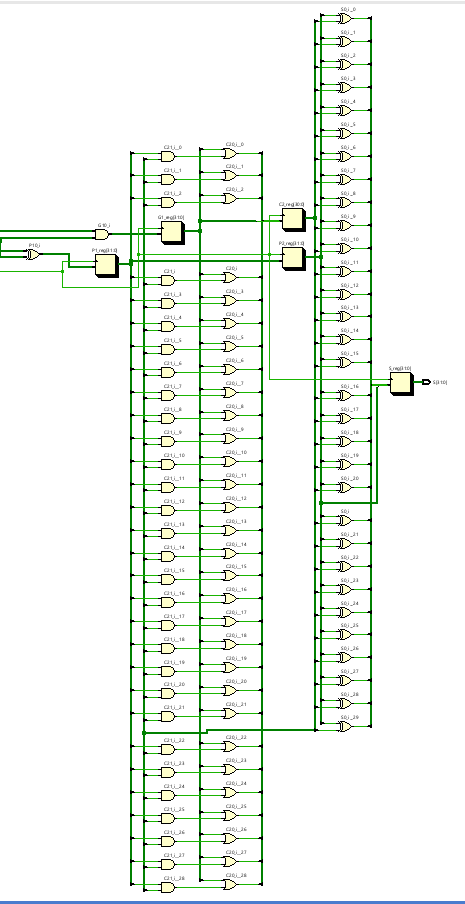
initial begin

$monitor("Time = %0t | A = %h | B = %h | S = %h", $time, A, B, S);

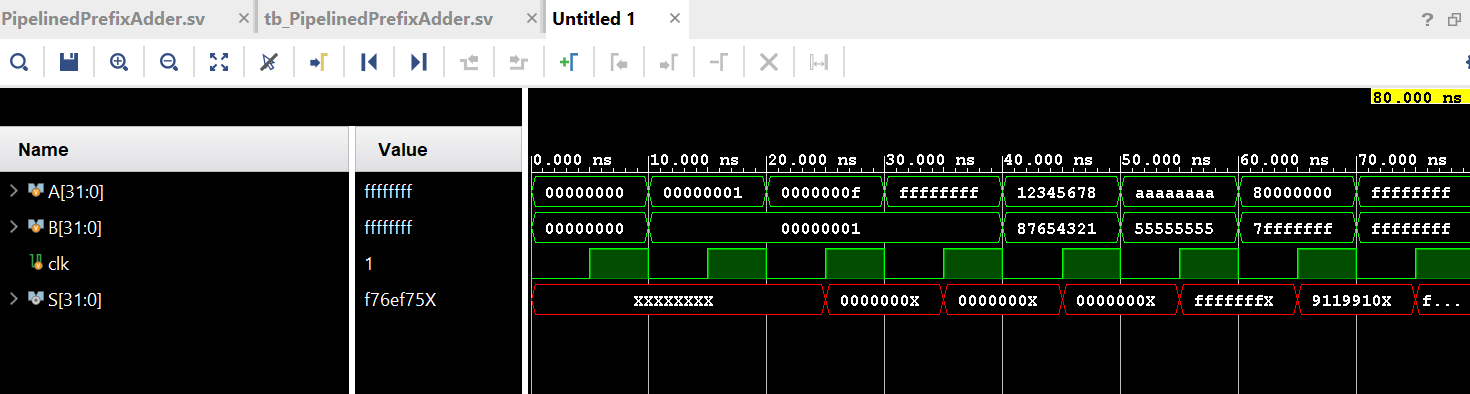
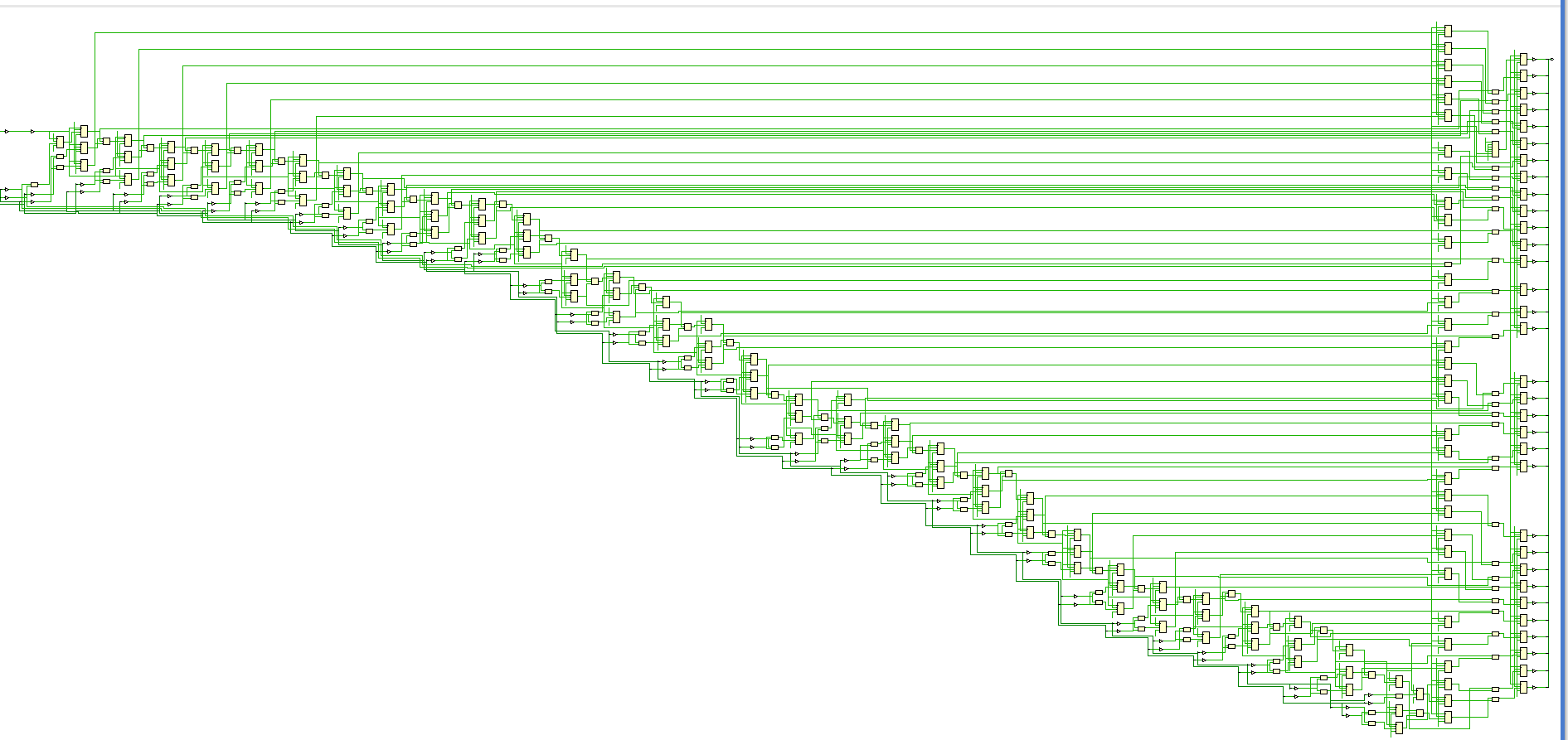
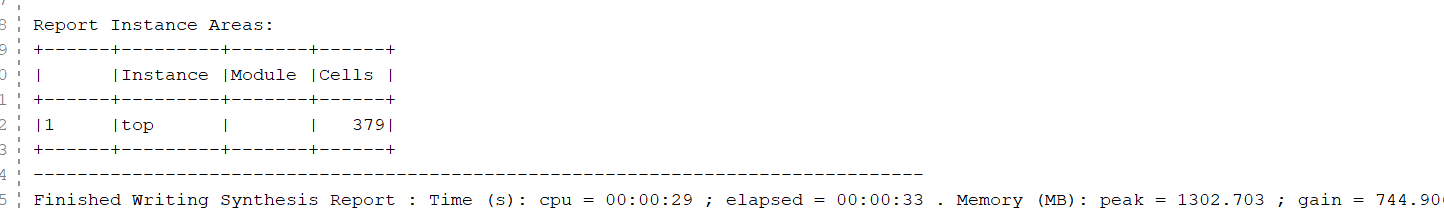
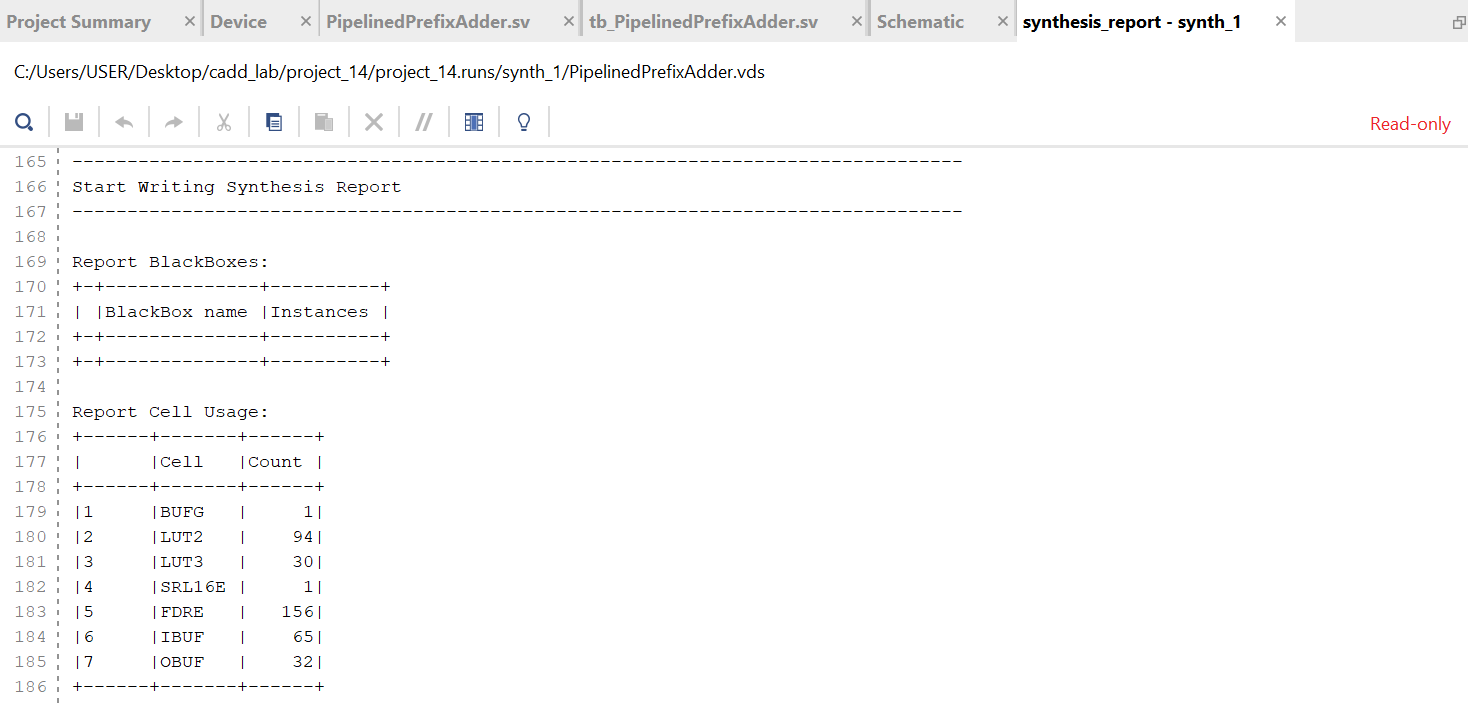
end

endmodule

RTL:



WAVEFORMS:

TECHNOLOGY DIAGRAM:SYNTHESIS REPORT:

THANK YOU