CADD -- JOB ASSIGNMENT

Experiential Learning: Level 1- Banana Problem statement

1. Problem Statement: 8-bit BCD Adder

Design hardware to add two 8-bit unsigned BCD numbers. Sketch a schematic for your design, and write an HDL module for the BCD adder. The inputs are A, B, and Cin, and the outputs are S and Cout. Cin and Cout are 1-bit carries, and A, B, and S are 8-bit BCD numbers.

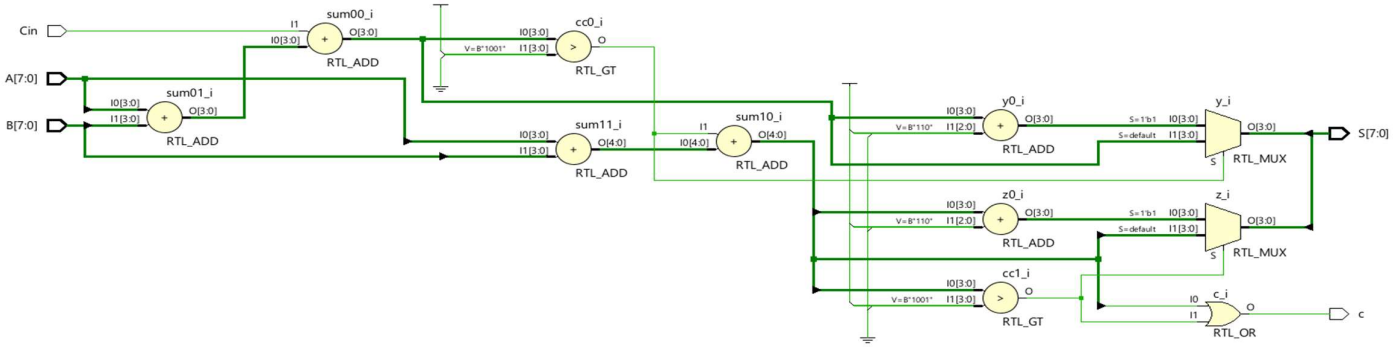
CODE

```
module banana1 (
    input logic  [7:0] A, B,
    input Cin,
    output [7:0] S,
    output Carry
);
    wire [3:0] lower_sum, upper_sum;
    wire lower_carry, upper_carry;
    wire [3:0] lower_corrected, upper_corrected;
    wire lower_overflow, upper_overflow;

    assign {lower_carry, lower_sum} = A[3:0] + B[3:0] + Cin;
    assign lower_overflow = (lower_sum > 4'd9) ? 1'b1 : 1'b0;
    assign lower_corrected = lower_overflow ? lower_sum + 4'd6 : lower_sum;
    assign {upper_carry, upper_sum} = A[7:4] + B[7:4] + lower_overflow;
    assign upper_overflow = (upper_sum > 4'd9) ? 1'b1 : 1'b0;
    assign upper_corrected = upper_overflow ? upper_sum + 4'd6 : upper_sum;
    assign S = {upper_corrected, lower_corrected};
    assign Carry = upper_carry | upper_overflow;
endmodule
```

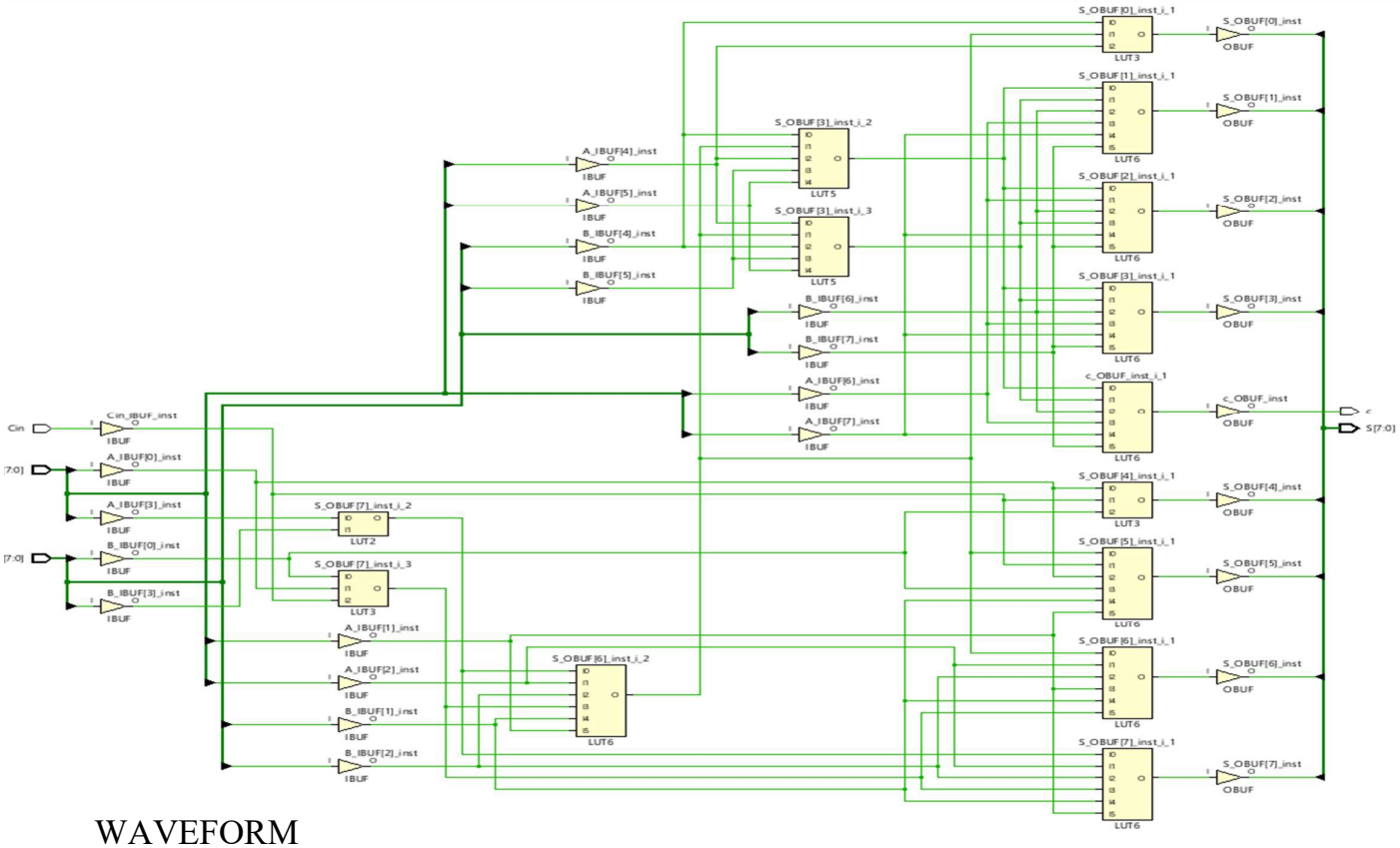
TESTBENCH

```
module banana1tb;
    reg [7:0] A, B;
    reg Carry_in;
    wire [7:0] Sum;
    wire Carry_out;

    banana1 dut (
        A,
        B,
        Carry_in,
        Sum,
        Carry_out
    );
    initial begin
        A = 8'b00000010; B = 8'b00000011; Carry_in = 0; #10;
        A = 8'b00001001; B = 8'b00000010; Carry_in = 0; #10;
        A = 8'b01011000; B = 8'b01001001; Carry_in = 0; #10;
        A = 8'b00000101; B = 8'b00000101; Carry_in = 1; #10;
        A = 8'b10011001; B = 8'b10011001; Carry_in = 0; #10;
        A = 8'b00000000; B = 8'b00000000; Carry_in = 0; #10;
        A = 8'b00100110; B = 8'b00010101; Carry_in = 1; #10;
    end
endmodule
```

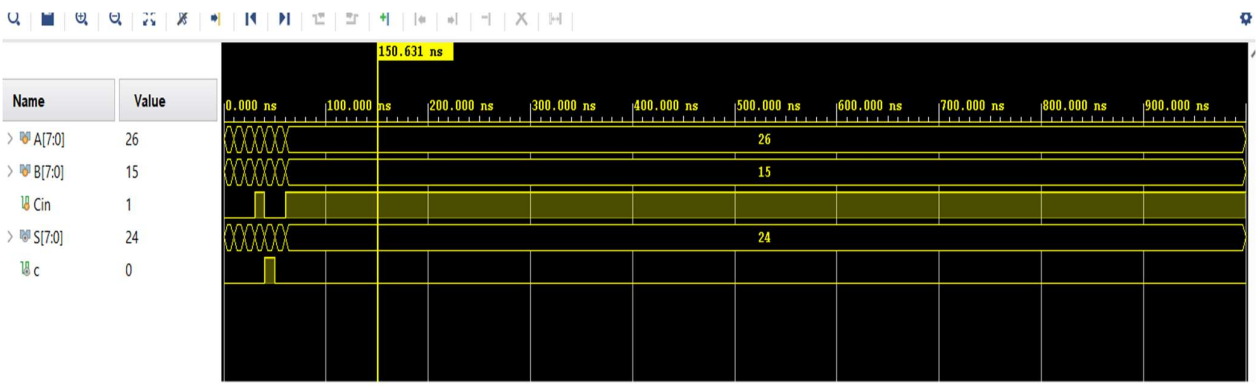
RTl

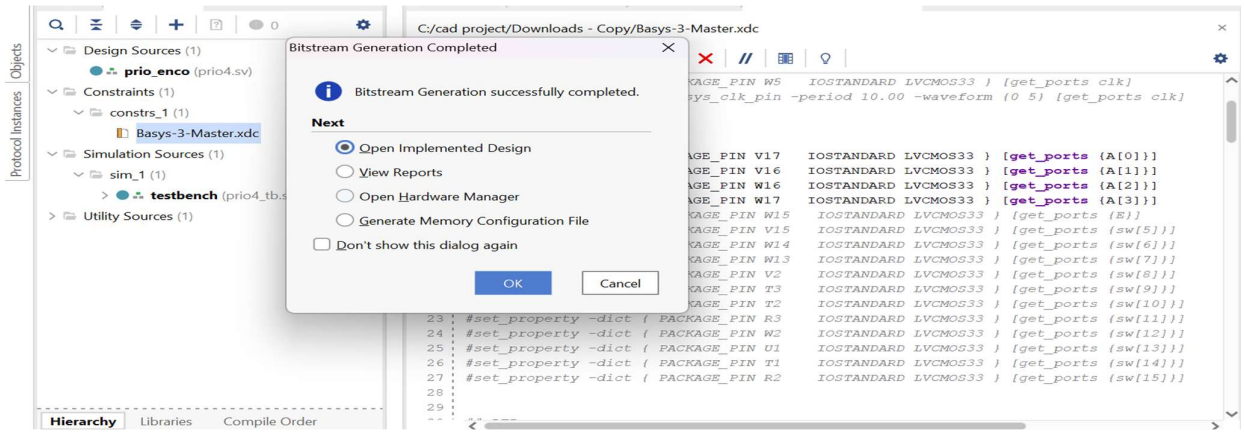


Technology schematic



WAVEFORM





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

How many ARTY 100T FPGA CLBs/recourses are required to perform each of the following functions? Show how to configure one or more CLBs to perform the function.
(i) You should be able to do this by inspection, without performing logic synthesis.
(ii) Write HDL module, perform synthesis, and tabulate the synthesis results

(a) The combinational function

$$Y = \overline{A}\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C} + A\overline{B}C\overline{D} + ABD + \overline{A}\overline{B}CD + B\overline{C}D + \overline{A}$$

CODE

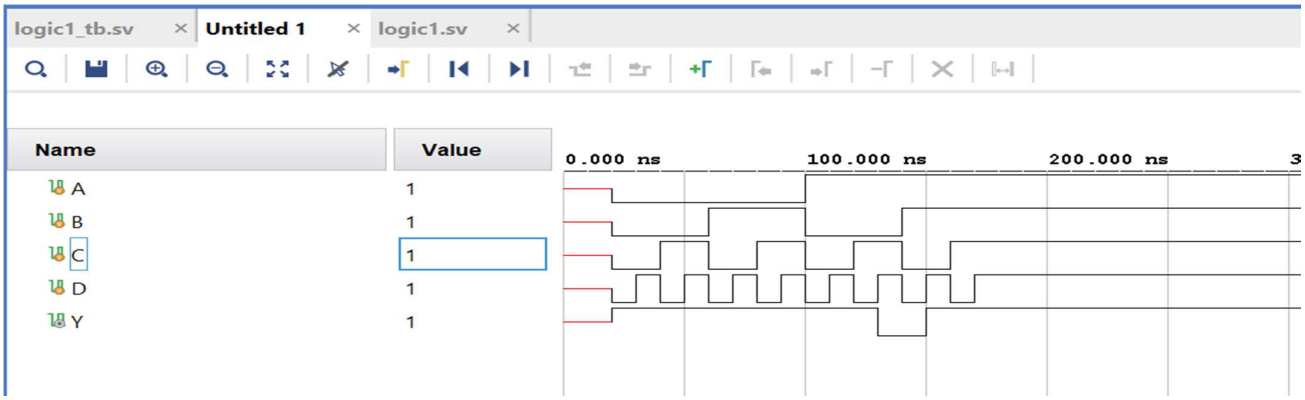
```
module banana_2 (
   input logic InputA, InputB, InputC, InputD,
   output logic OutputY
);
   assign OutputY = ~InputA |
                    (InputA & InputB & InputD) |
                    (InputA & InputC & ~InputD) |
                    (InputA & ~InputB & ~InputC);
Endmodule
```

TESTBENCH

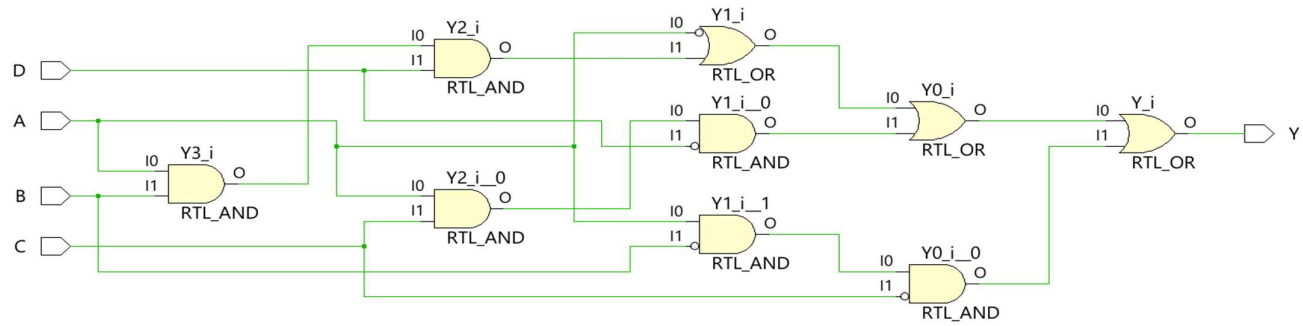
```
module banana_21tb;
   reg InputA, InputB, InputC, InputD;
   wire OutputY;

   banana_21 dut (
      InputA,
      InputB,
      InputC,
      InputD,
      OutputY
   );
   initial begin
      #20; InputA = 0; InputB = 0; InputC = 0; InputD = 0;
      #10; InputA = 0; InputB = 0; InputC = 0; InputD = 1;
      #10; InputA = 0; InputB = 0; InputC = 1; InputD = 0;
      #10; InputA = 0; InputB = 0; InputC = 1; InputD = 1;
      #10; InputA = 0; InputB = 1; InputC = 0; InputD = 0;
      #10; InputA = 0; InputB = 1; InputC = 0; InputD = 1;
      #10; InputA = 0; InputB = 1; InputC = 1; InputD = 0;
      #10; InputA = 0; InputB = 1; InputC = 1; InputD = 1;
      #10; InputA = 1; InputB = 0; InputC = 0; InputD = 0;
      #10; InputA = 1; InputB = 0; InputC = 0; InputD = 1;
      #10; InputA = 1; InputB = 0; InputC = 1; InputD = 0;
      #10; InputA = 1; InputB = 0; InputC = 1; InputD = 1;
```

```
        #10; InputA = 1; InputB = 1; InputC = 0; InputD = 0;
        #10; InputA = 1; InputB = 1; InputC = 0; InputD = 1;
        #10; InputA = 1; InputB = 1; InputC = 1; InputD = 0;
        #10; InputA = 1; InputB = 1; InputC = 1; InputD = 1;
    end
endmodule
```

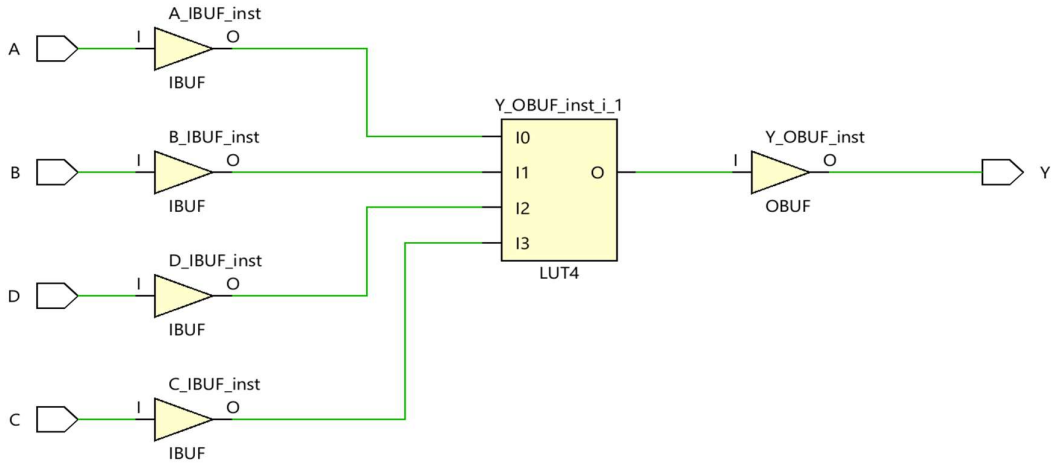
WAVEFORM

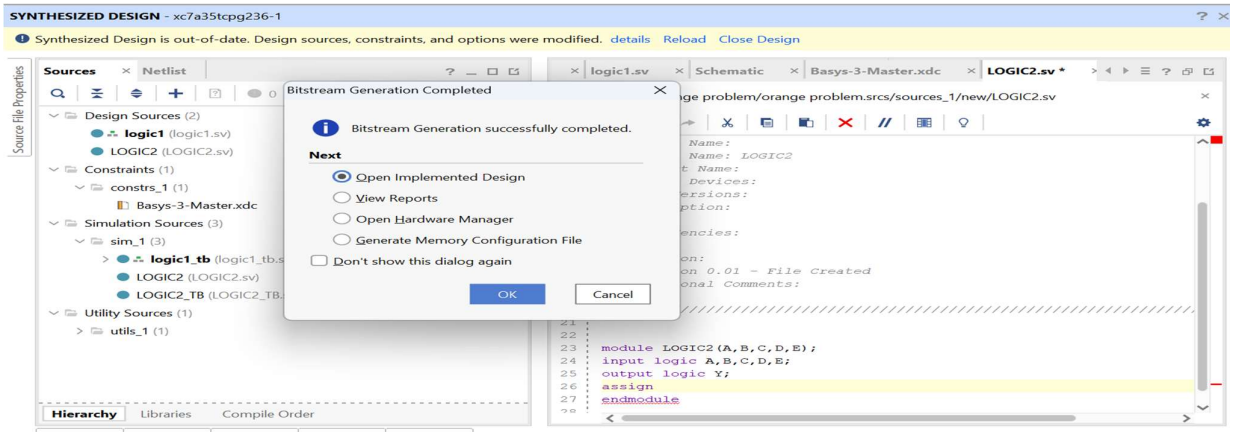


RTL



Technology schematic





(b) The combinational function

$$Y = ABC + ABD + ABE + ACD + ACE + (\overline{A + D + E}) + \overline{B}\,\overline{C}D + \overline{B}\,\overline{C}E + \overline{B}\,\overline{D}E + \overline{C}\,\overline{D}E$$

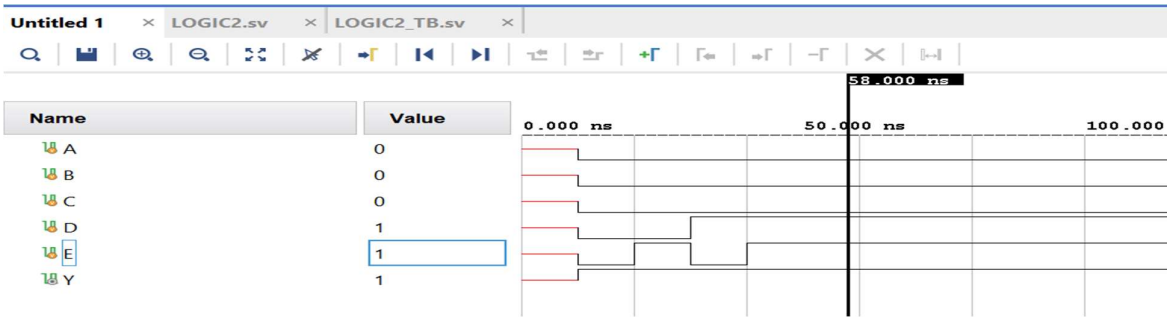
CODE

```
module banana_22(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
```

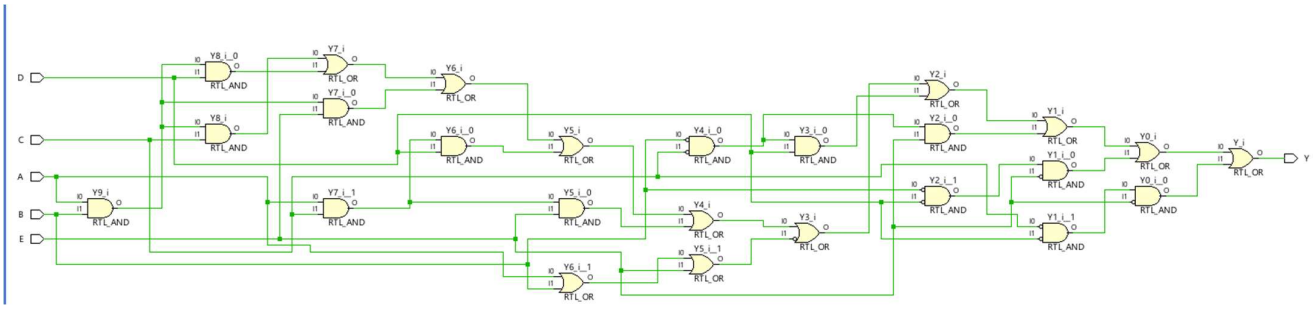
TESTBENCH

```
module banana_22tb;
reg A,B,C,D,E;
wire Y;
banana_22  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
```

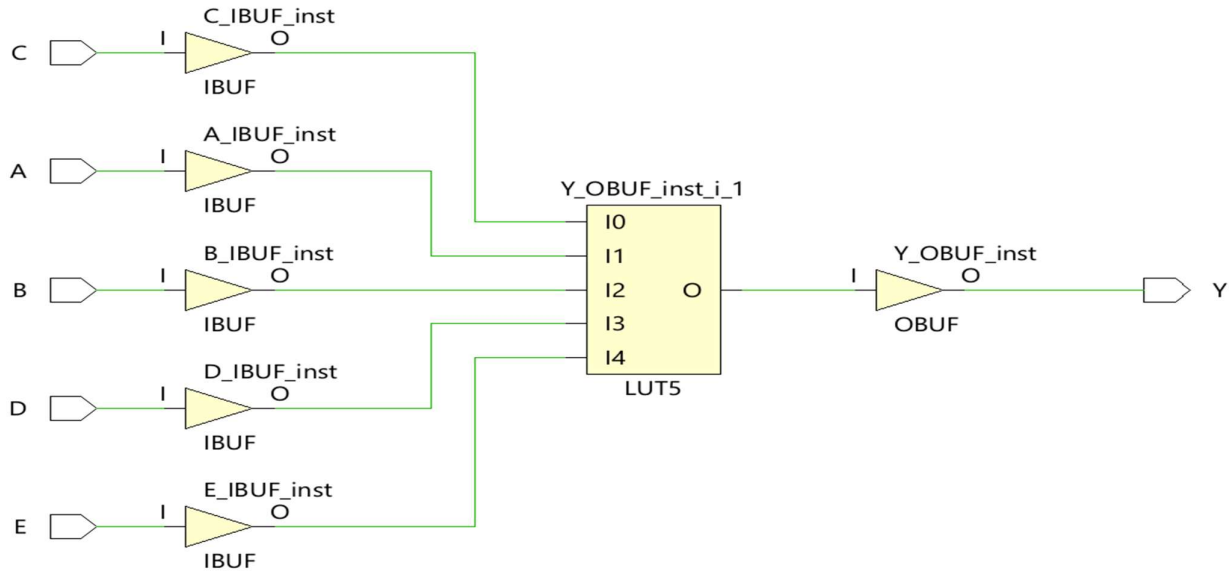
WAVEFORM

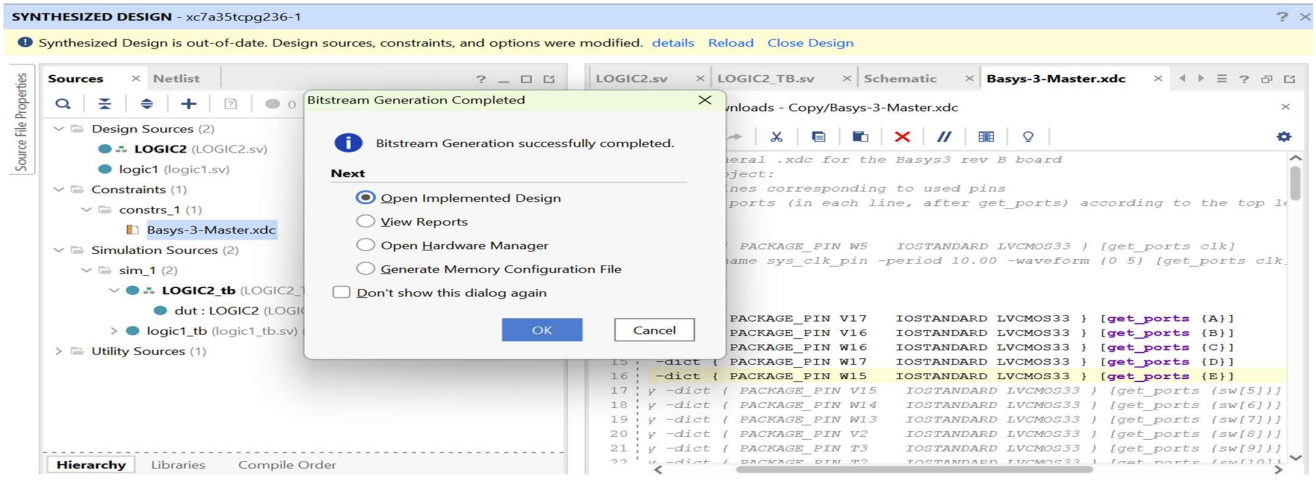


RTL

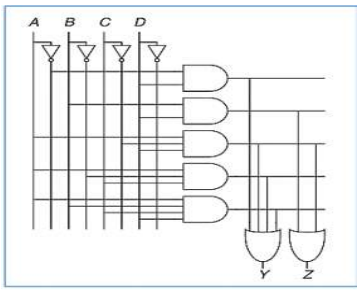


Technology schematic





(c) The two-output function



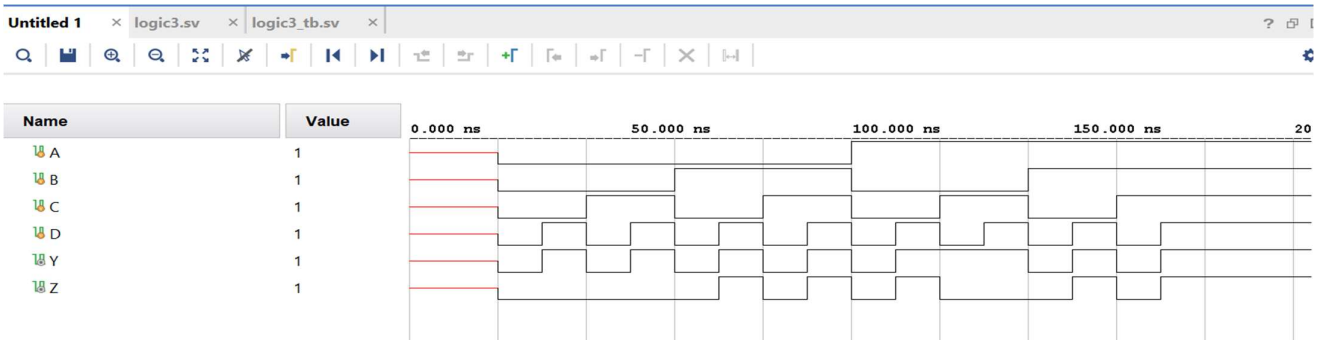
CODE

```
module banana_23(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
```

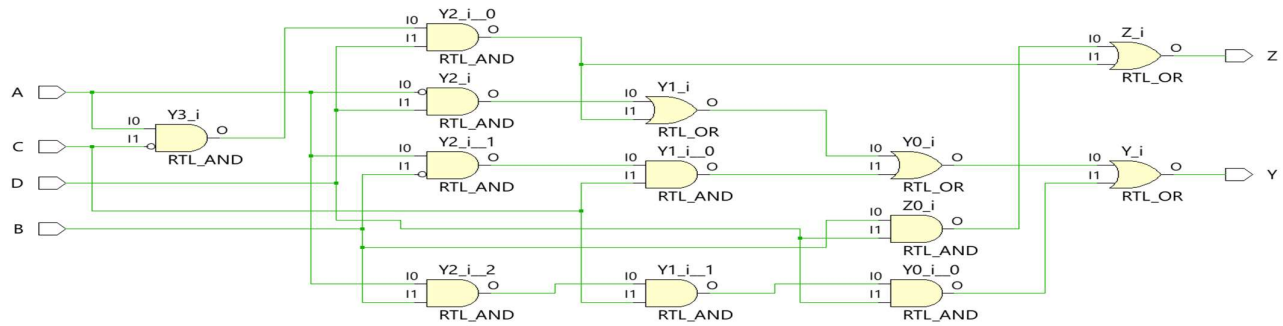
TESTBENCH

```
module banana_23tb;
reg A,B,C,D;
wire Y,Z;
banana_23  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
```

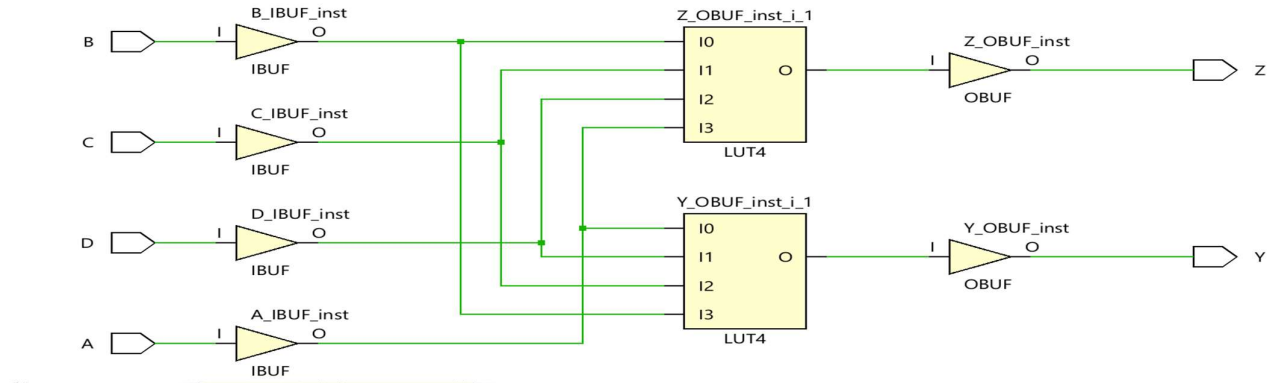
WAVEFORM

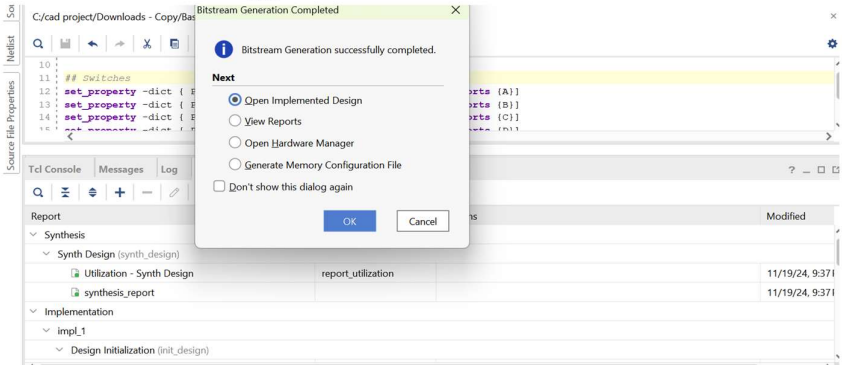


RTL



Technology schematic





(d) The function from –
A circuit has four inputs and two outputs. The inputs, A3:0, represent a number from 0 to 15. Output P should be TRUE if the number is prime (0 and 1 are not prime, but 2, 3, 5, and so on, are prime). Output D should be TRUE if the number is divisible by 3. Give simplified Boolean equations for each output and sketch a circuit.

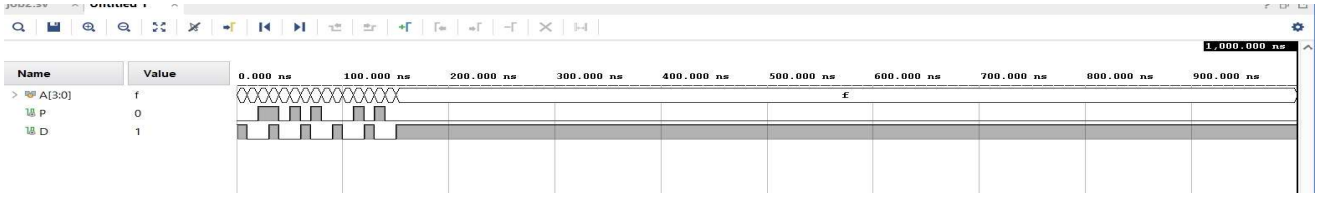
CODE

```
module banana_24(
    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
```

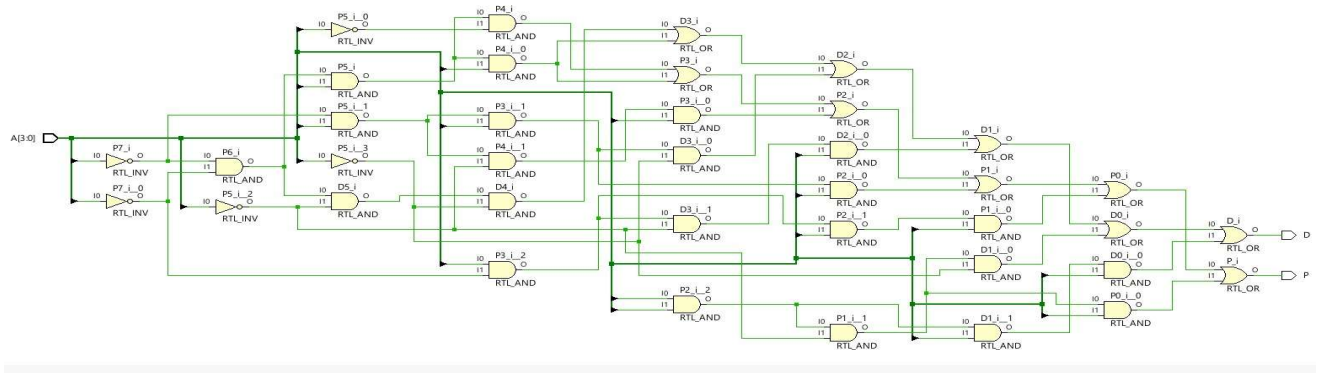
TESTBENCH

```
module banana_24tb;
    reg [3:0] A;
    wire P, D;
   banana_24  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
```

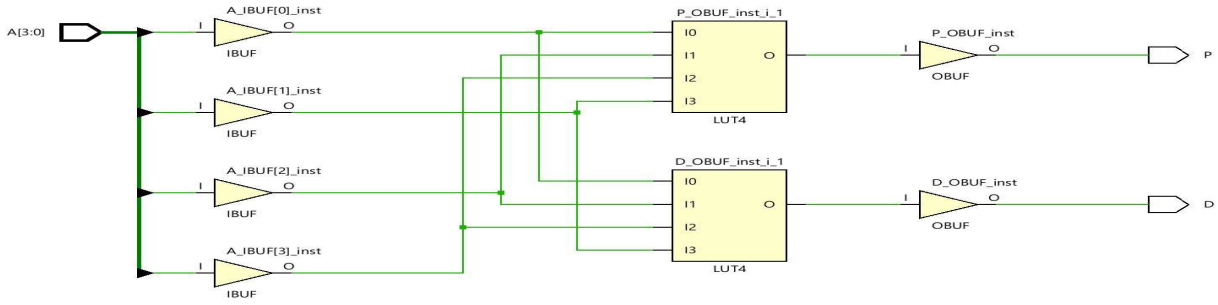
WAVEFORM

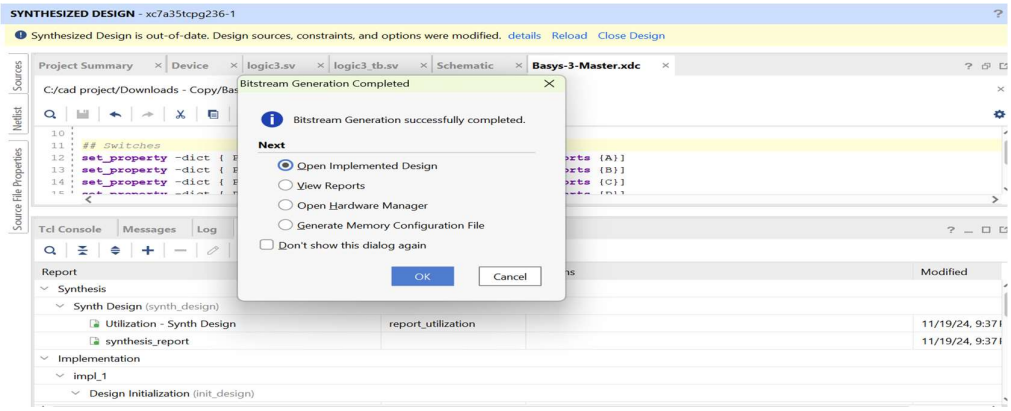


RTL



Technology schematic





(e) A four-input priority encoder

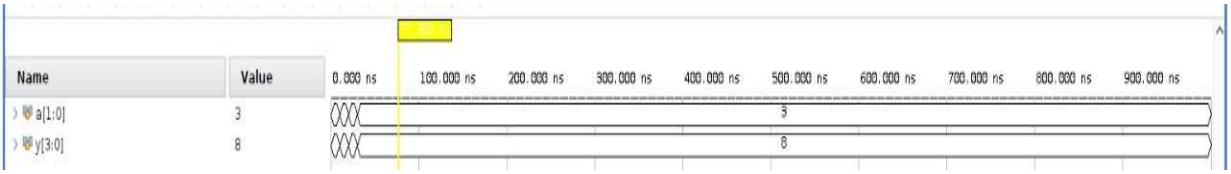
CODE

```
module banana_25(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
```

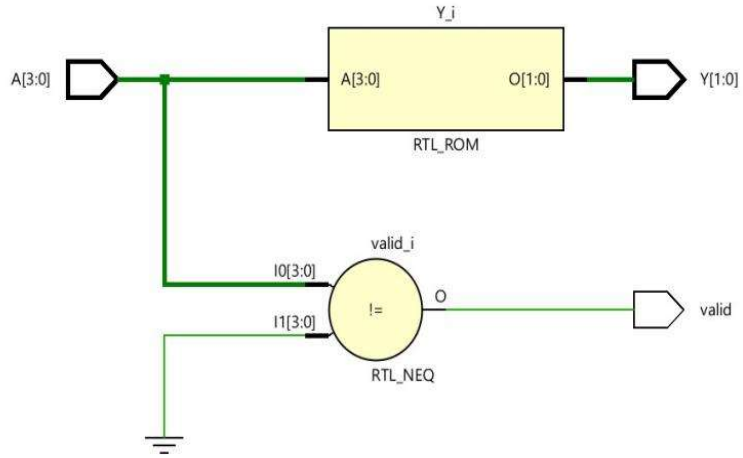
TESTBENCH

```
module job_q_2.5_tb;
  reg [3:0] A;
  wire [1:0] Y;
  wire valid;

  job_q_2.5 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
```

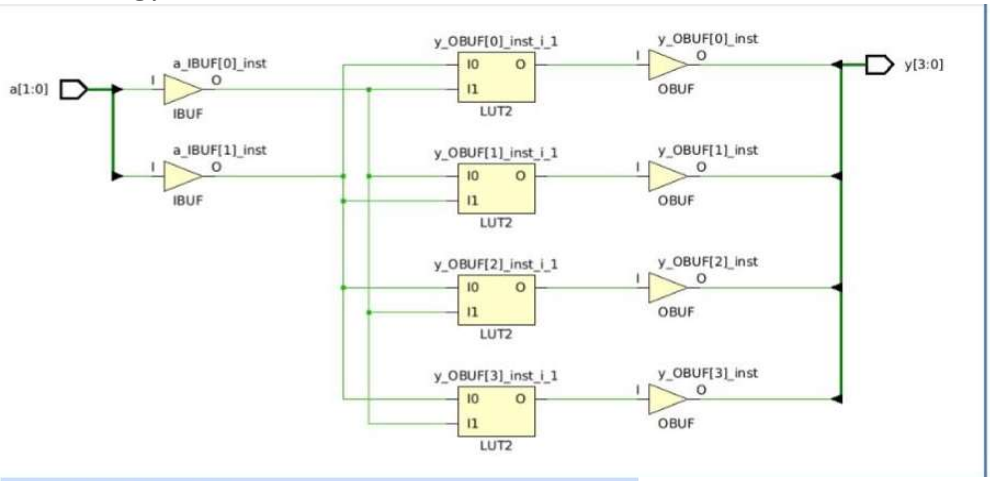
WAVEFORM

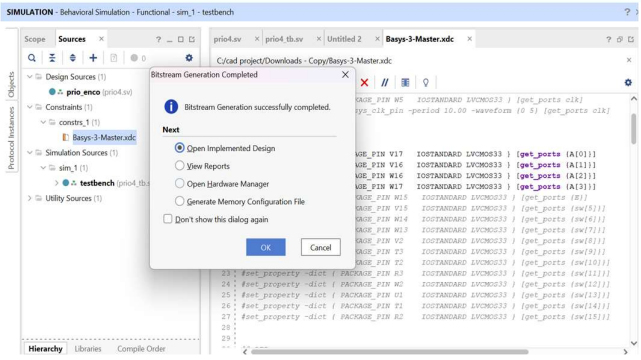


RTL



Technology schematic





(f) An eight-input priority encoder

CODE

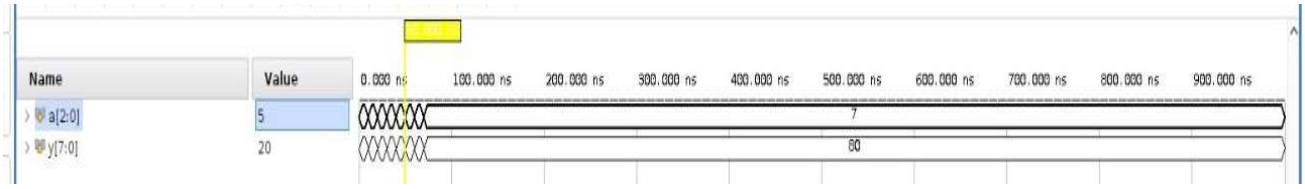
```
module banana_26(
  input logic [7:0] A,
  output logic [2:0] Y
);
  always_comb begin
```

```
    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
```

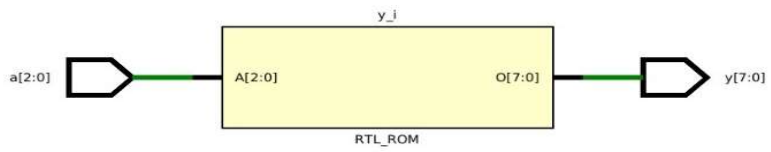
TESTBENCH

```
module  banana_26_tb;
  reg [7:0] A;
  wire [2:0] Y;
  wire valid;
  banana_26 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
```

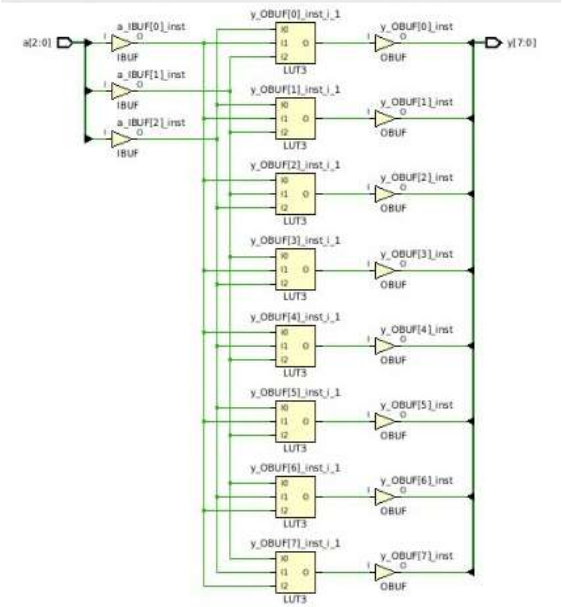
WAVEFORM

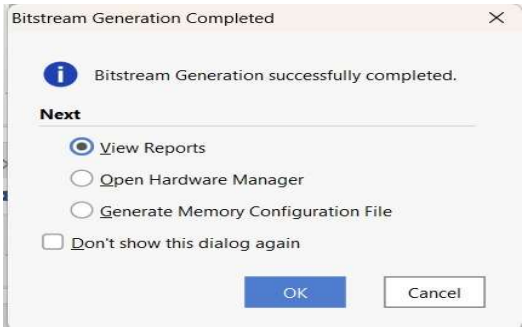


RTL



Technology schematic





(g) A 3:8 decoder.

CODE

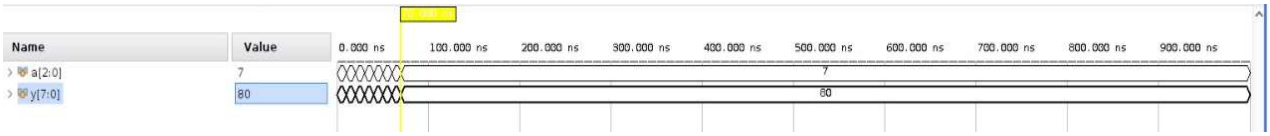
```
module job_q_2.7(input logic [2:0] a, output logic [7:0] y);
  always_comb
    case(a)
      3'b000 : y = 8'b00000001;
      3'b001 : y = 8'b00000010;
      3'b010 : y = 8'b00000100;
      3'b011 : y = 8'b00001000;
      3'b100 : y = 8'b00010000;
      3'b101 : y = 8'b00100000;
      3'b110 : y = 8'b01000000;
      3'b111 : y = 8'b10000000;
      default: y = 8'bxxxxxxxx;
    endcase
endmodule
```

TESTBENCH

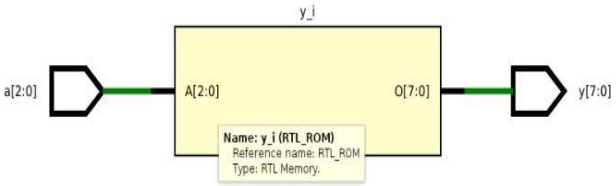
```
  module job_q_2.7_tb;

  logic [2:0] a;
  logic [7:0] y;

  job_q_2.7 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
```

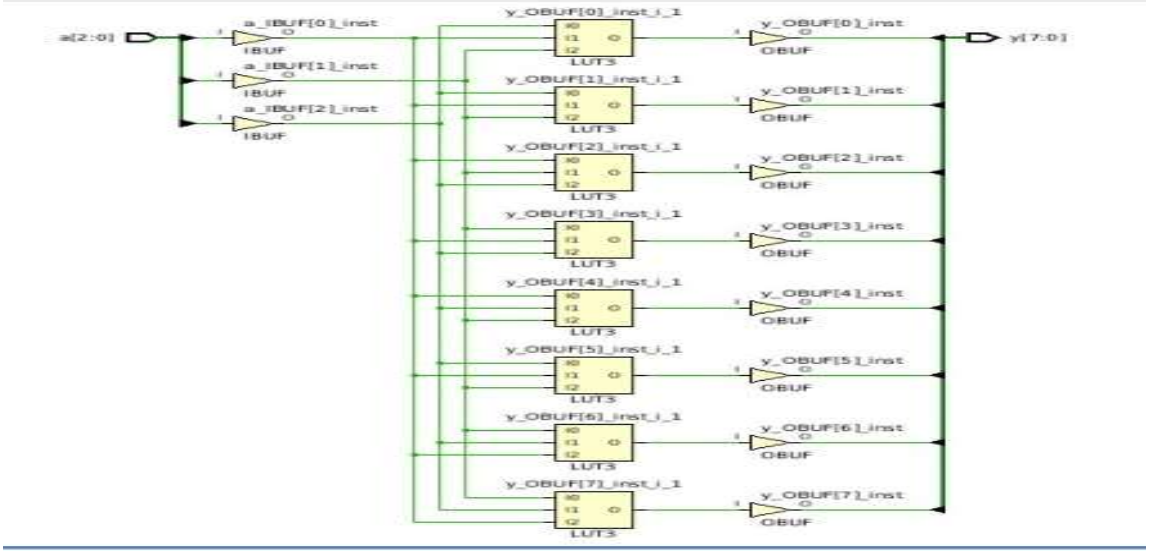
WAVEFORM

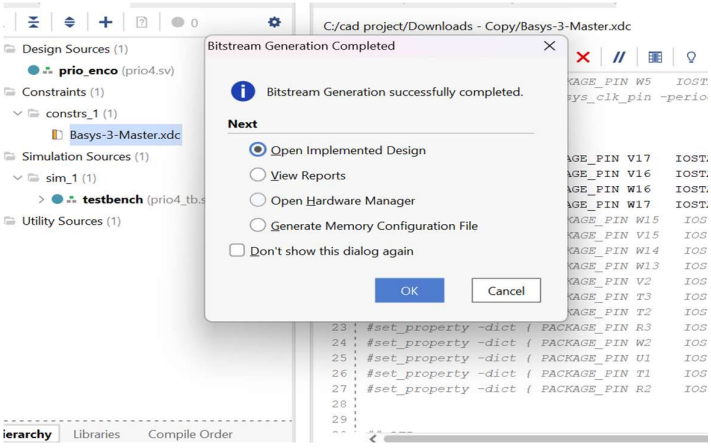


RTL



Technology schematic





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

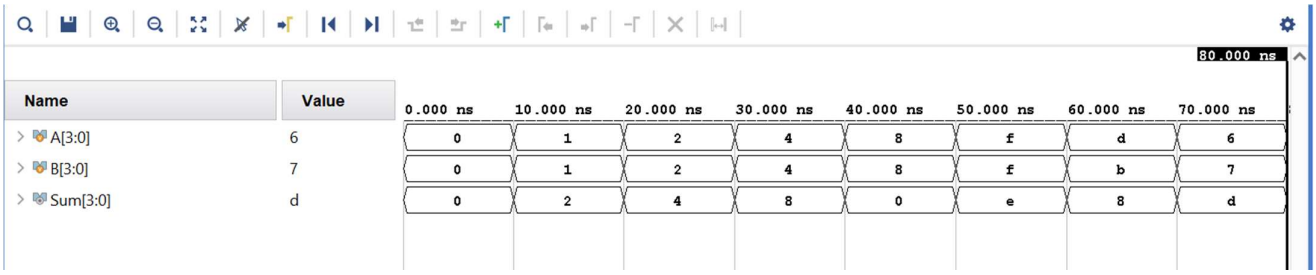
CODE

```
module job_q_2.8(
  input [3:0] A,
  input [3:0] B,
  output [3:0] Sum
);
  assign Sum = A + B;
endmodule
```

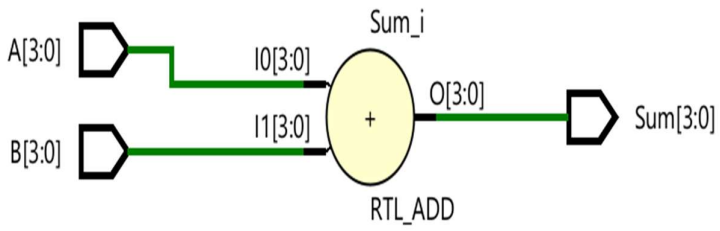
TESTBENCH

```
module job_q_2.8_tb;
 reg [3:0] A, B;
 wire [3:0] Sum;

 carry_ job_q_2.8 dut (
  .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
```

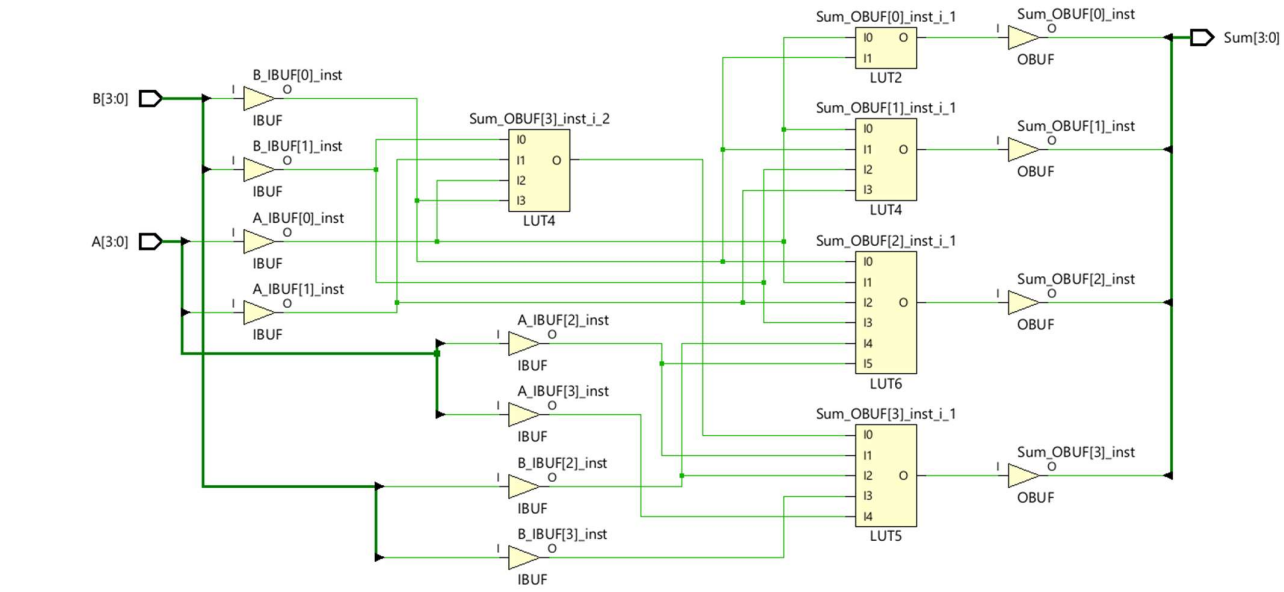
WAVEFORM

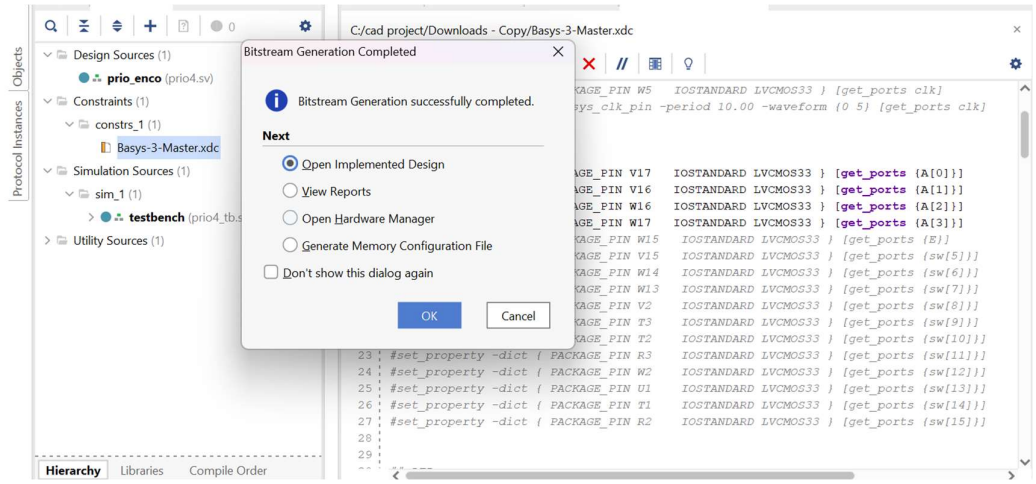


RTL

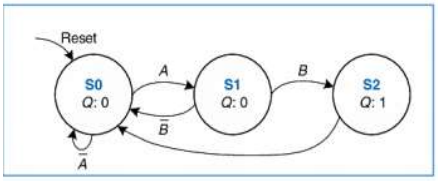


Technology schematic





(i) The FSM



CODE

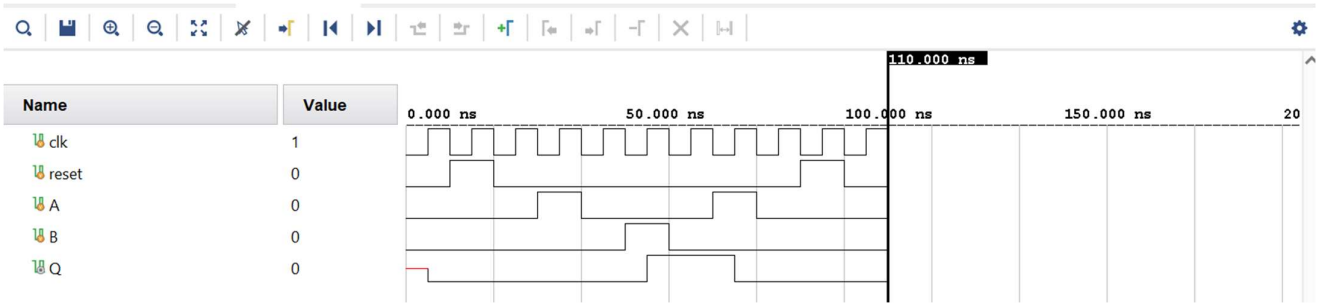
```
module job_q_2.9(
  input logic clk,
  input logic reset,
  input logic A,
  input logic B,
  output logic Q
);
  typedef enum logic [1:0] {S0 = 2'b00, S1 = 2'b01, S2 = 2'b10} state_t;
  state_t state, next_state;
```

```
 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
```

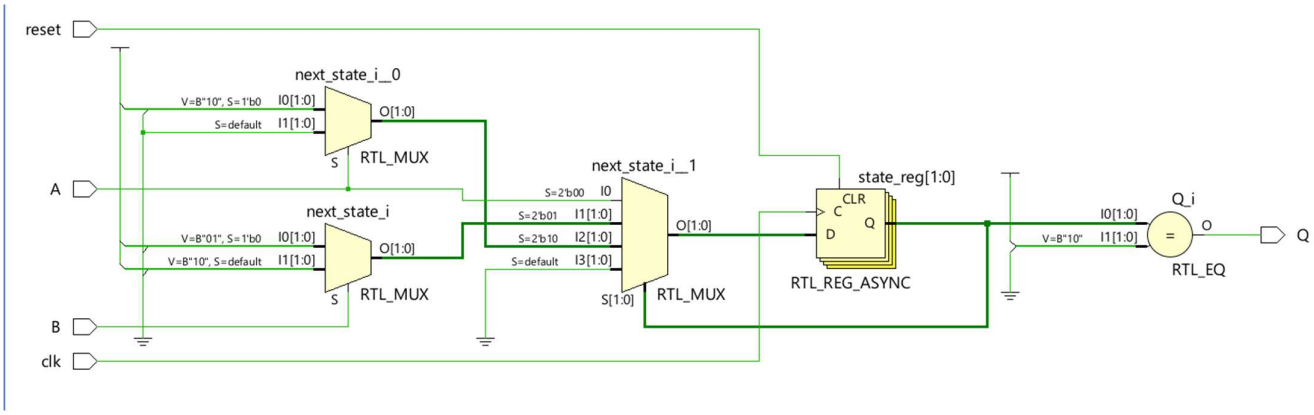
TESTBENCH

```
module job_q_2.9_tb;
reg clk,reset,A,B;
wire Q;
job_q_2.9 dut(.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
```

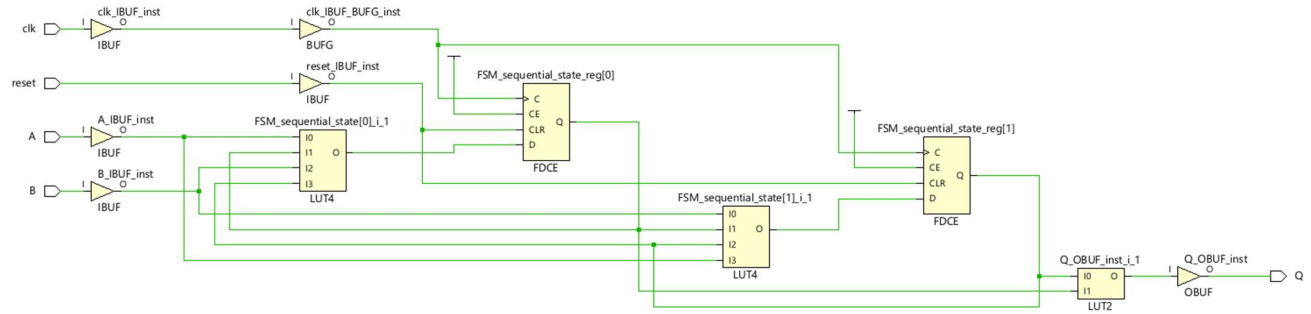
WAVEFORM

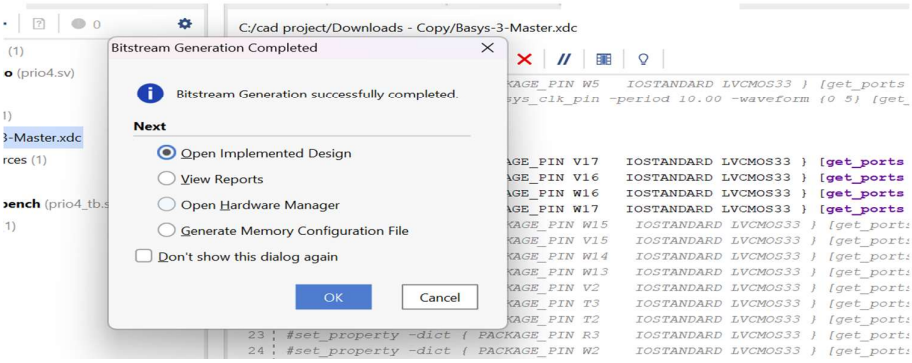


RTL



Technology schematic





(j) The Gray code counter

CODE

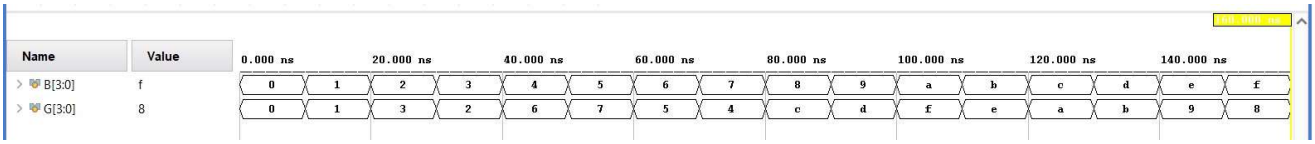
```
module  job_q2_10(
   input logic [3:0] B,
   output logic [3:0] G
);
   assign G[3] = B[3];
   assign G[2] = B[3] ^ B[2];
   assign G[1] = B[2] ^ B[1];
   assign G[0] = B[1] ^ B[0];
endmodule
```

TESTBENCH

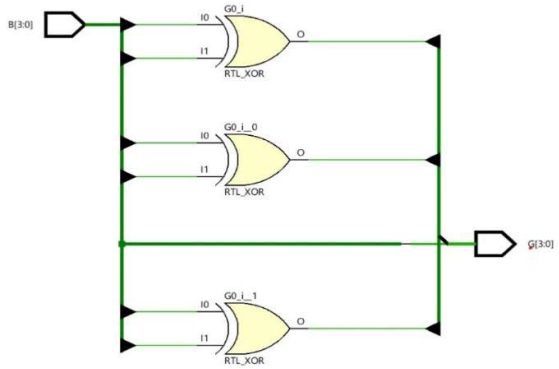
```
module job_q_2.10_tb ();
   logic [3:0] B;
   logic [3:0] G;
   job_q_2.10 dut (
      .B(B),
```

```
        .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
```

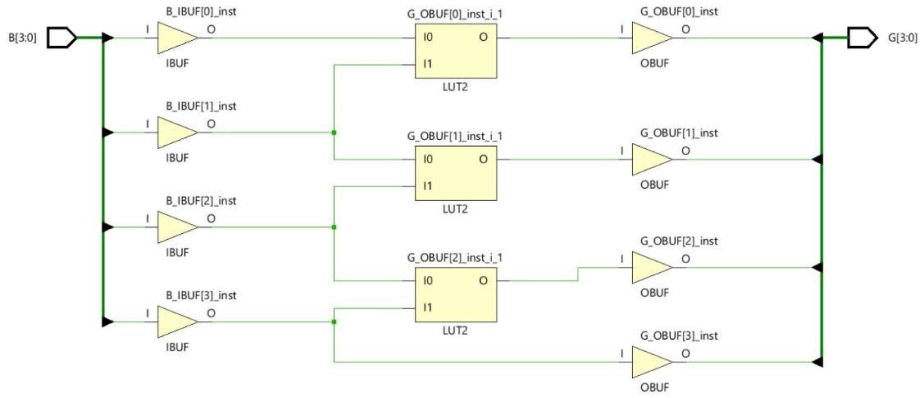
WAVEFORM

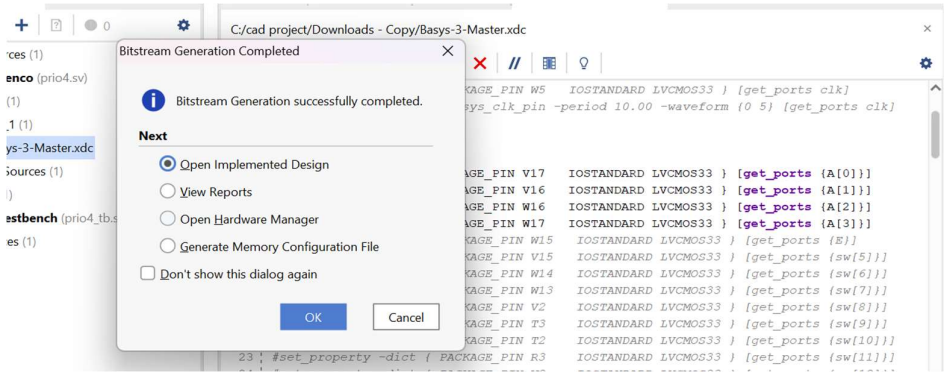


RTL



Technology schematic




C:/cad project/Downloads - Copy/Basys-3-Master.xdc
Bitstream Generation Completed
Bitstream Generation successfully completed.
Next
Open Implemented Design
View Reports
Open Hardware Manager
Generate Memory Configuration File
Don't show this dialog again
OK
Cancel