Digital System Design Verilog Assignment

Semester III

UNIVERSITY OF DELHI

FACULTY OF TECHNOLOGY

Submitted To: Dr. Khushwant Sehra (Assistant Professor, ECE Department)

Submitted By: Manav Bhatia

Branch: Computer Science and Engineering (CSE-A)

Roll No. 24293916122

Enrollment No. 24DOCEBTCS000122

Ques. Writing Source, Testbench Codes and Running Simulations for the following:

1. Basic Gates
2. Boolean Expressions
3. Multiplexers (4 x 1 and 8 x 1)
4. Priority Encoder
5. Decoder
6. Half Adder and Full Adder
7. Half Subtractor and Full Subtractor
8. Universal Adder / Subtractor with Overflow Check
9. SR Latch
10. SR and JK - Flip Flop
11. D and T - Flip Flop
12. Counter Design

Ans.

1) Basic Gates

a) AND Gate

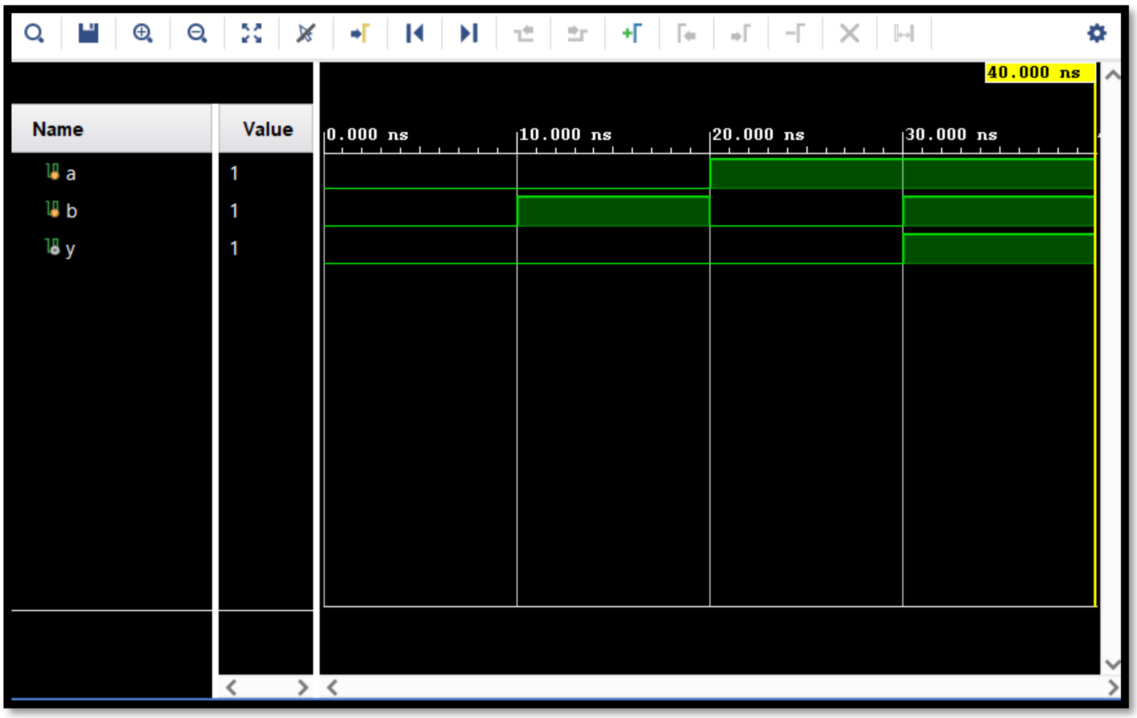
Source Code:

```
`timescale 1ns/1ps
module and_gate (
    input  wire a,
    input  wire b,
    output wire y
);
    assign y = a & b;
endmodule
```

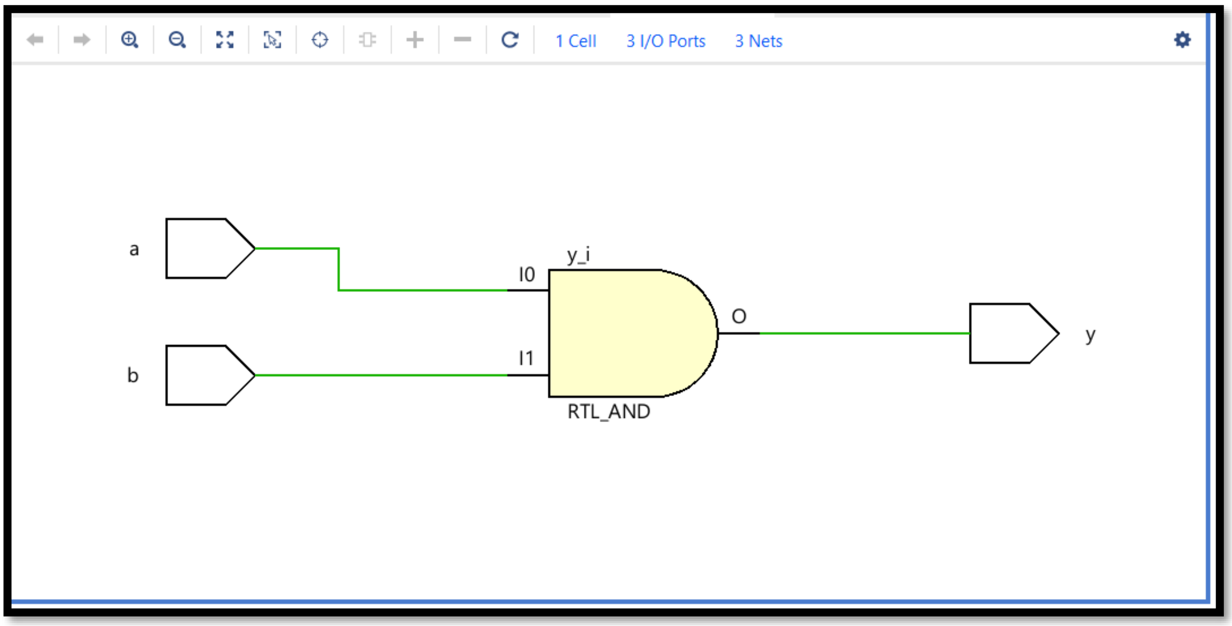
Testbench Code:

```
`timescale 1ns/1ps
module tb_and_gate;
reg a, b;
wire y;
// Instantiate AND gate module
and_gate uut (
    .a(a),
    .b(b),
    .y(y)
);
initial begin
    a = 0; b = 0; #10;
    a = 0; b = 1; #10;
    a = 1; b = 0; #10;
    a = 1; b = 1; #10;
    $finish;
end
endmodule
```

Simulation:



RTL Synthesis:



b) OR Gate

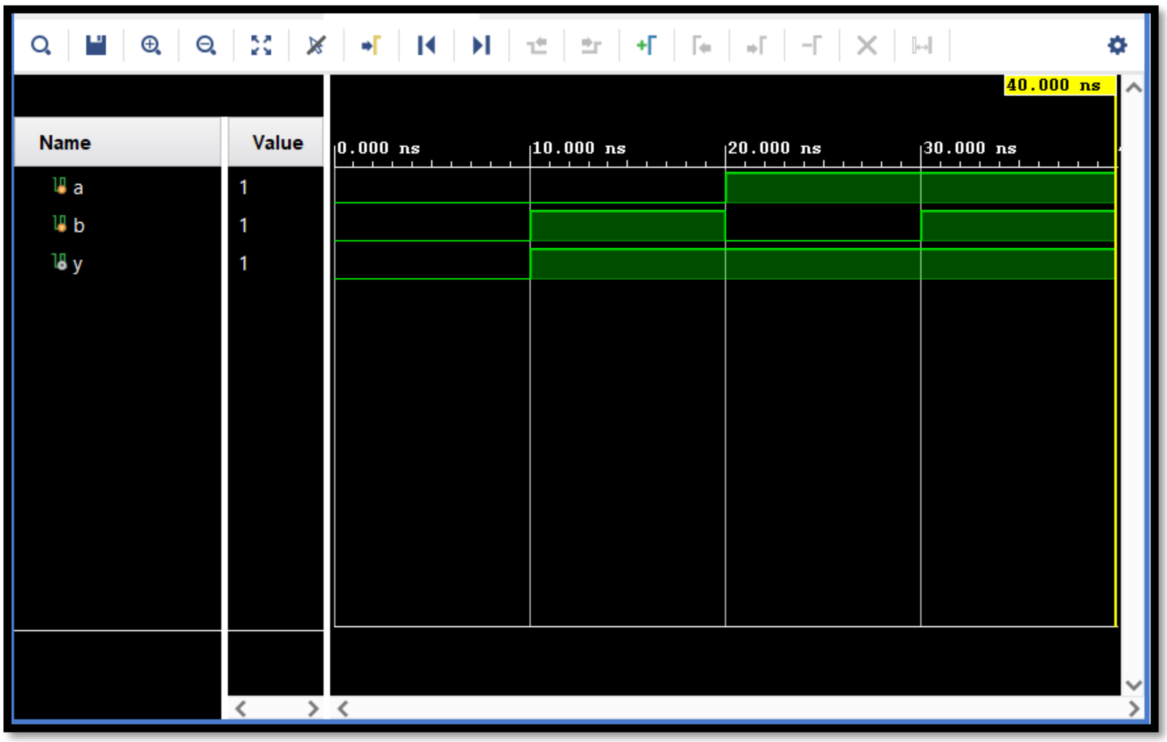
Source Code:

```
`timescale 1ns/1ps
module or_gate (
    input  wire a,
    input  wire b,
    output wire y
);
    assign y = a | b;
endmodule
```

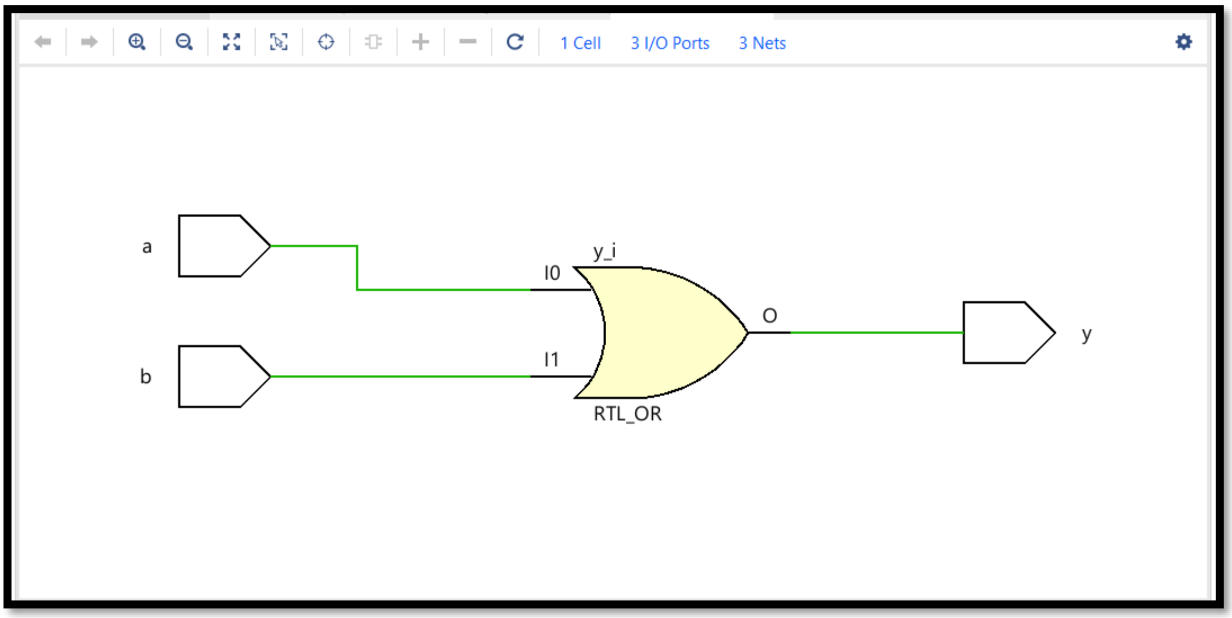
Testbench Code:

```
`timescale 1ns / 1ps

module tb_or_gate;

reg a, b;
wire y;

// Instantiate OR gate module
or_gate uut (
    .a(a),
    .b(b),
    .y(y)
);

initial begin
    a = 0; b = 0; #10;
    a = 0; b = 1; #10;
    a = 1; b = 0; #10;
    a = 1; b = 1; #10;

    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



c) NOT Gate

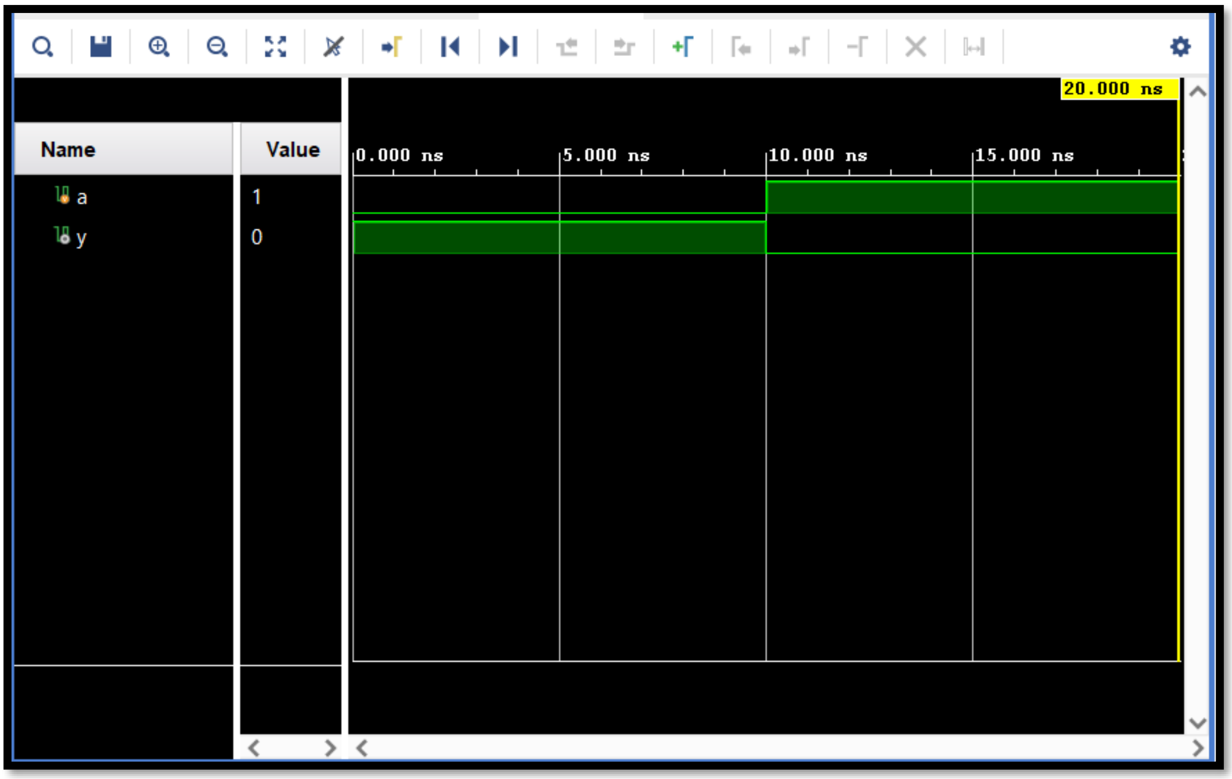
Source Code:

```
`timescale 1ns/1ps
module not_gate (
    input  wire a,
    output wire y
);
    assign y = ~a;
endmodule
```

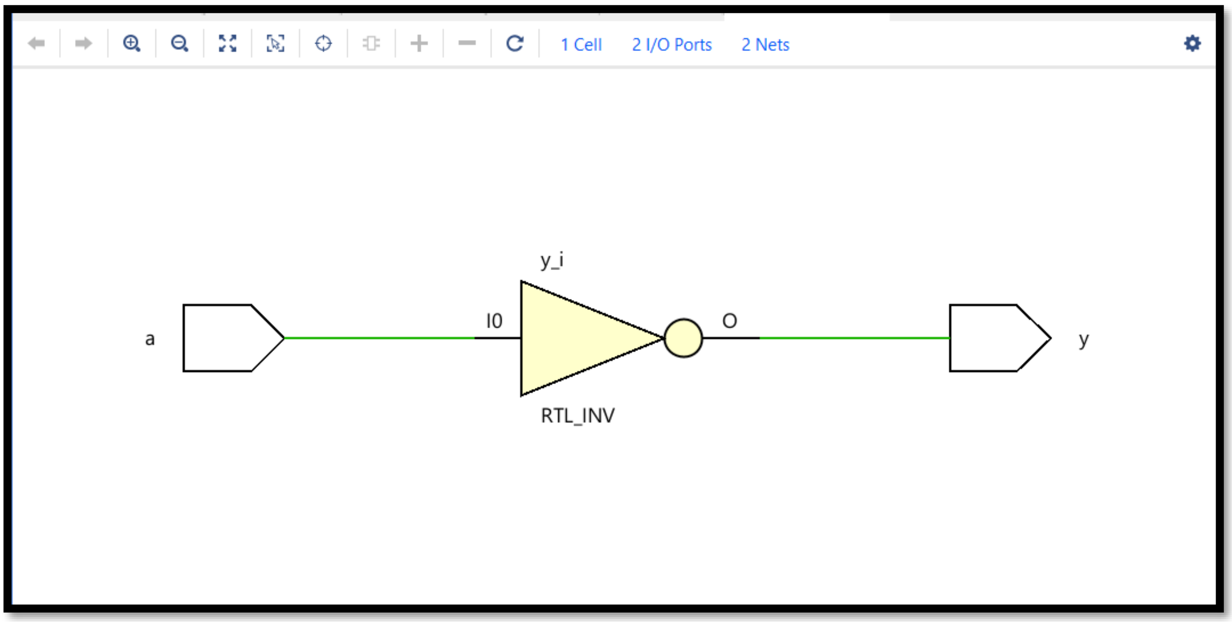
Testbench Code:

```
`timescale 1ns / 1ps

module tb_not_gate;

reg a;
wire y;

// Instantiate NOT gate module
not_gate uut (
    .a(a),
    .y(y)
);

initial begin
   a = 0; #10;
   a = 1; #10;

   $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



d) NAND Gate

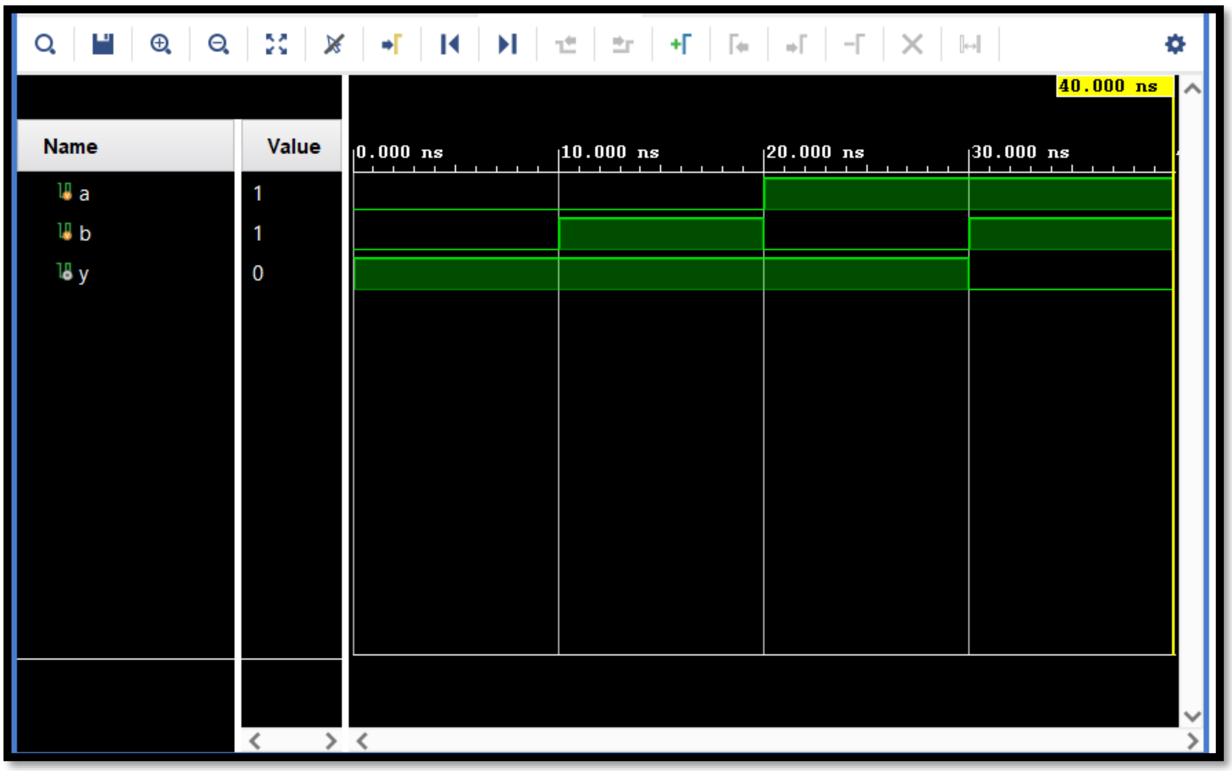
Source Code:

```
`timescale 1ns/1ps
module nand_gate (
    input  wire a,
    input  wire b,
    output wire y
);
    assign y = ~(a & b);
endmodule
```

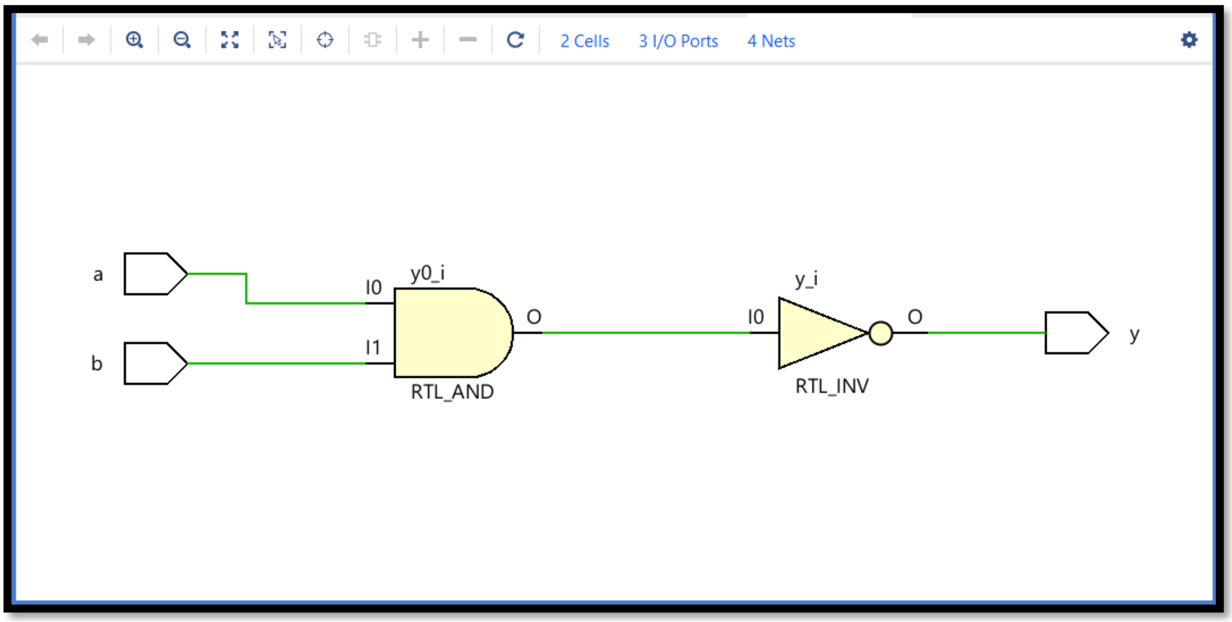
Testbench Code:

```
`timescale 1ns / 1ps

module tb_nand_gate;

reg a, b;
wire y;

// Instantiate NAND gate module
nand_gate uut (
    .a(a),
    .b(b),
    .y(y)
);

initial begin
    a = 0; b = 0; #10;
    a = 0; b = 1; #10;
    a = 1; b = 0; #10;
    a = 1; b = 1; #10;

    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



e) NOR Gate

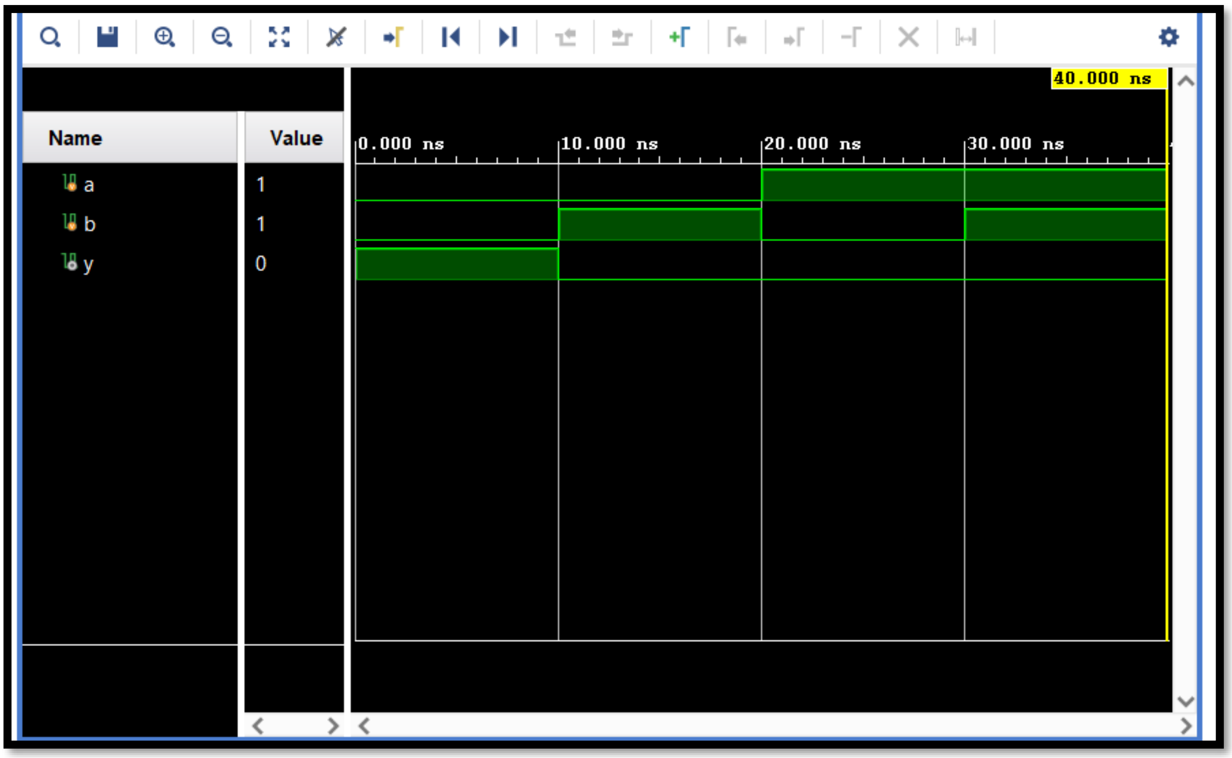
Source Code:

```
`timescale 1ns/1ps
module nor_gate (
    input  wire a,
    input  wire b,
    output wire y
);
    assign y = ~(a | b);
endmodule
```

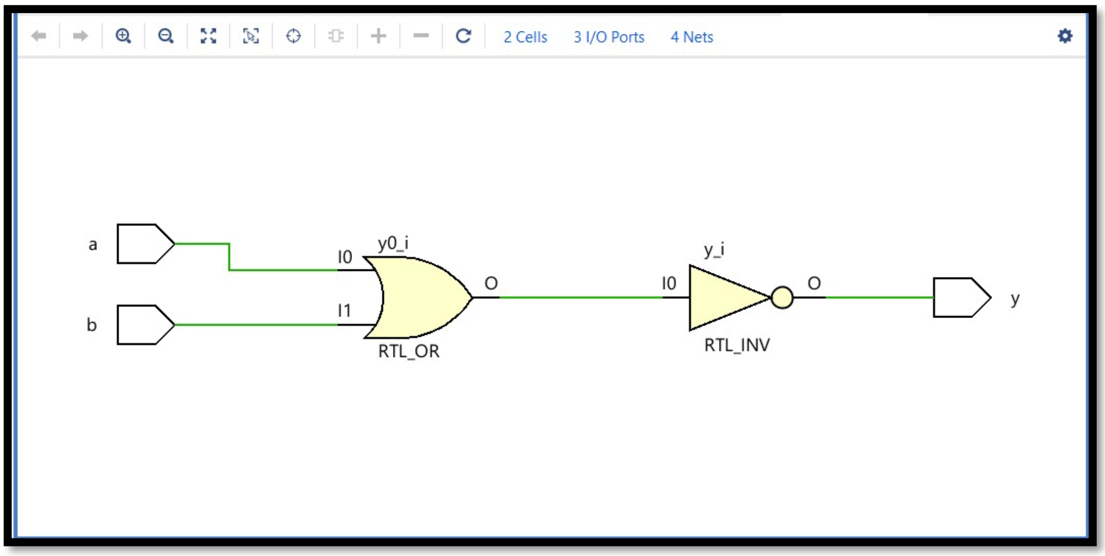
Testbench Code:

```
`timescale 1ns / 1ps

module tb_nor_gate;

reg a, b;
wire y;

// Instantiate NOR gate module
nor_gate uut (
    .a(a),
    .b(b),
    .y(y)
);

initial begin
    a = 0; b = 0; #10;
    a = 0; b = 1; #10;
    a = 1; b = 0; #10;
    a = 1; b = 1; #10;

    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



f) XOR Gate

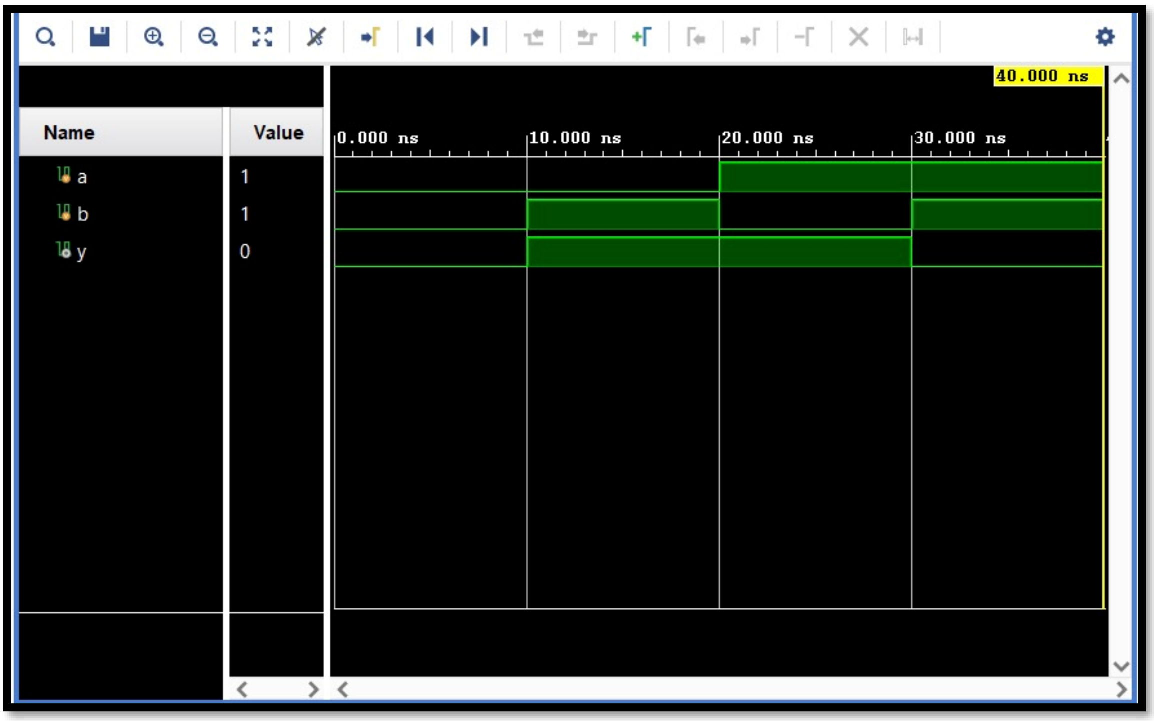
Source Code:

```
`timescale 1ns/1ps
module xor_gate (
    input  wire a,
    input  wire b,
    output wire y
);
    assign y = a ^ b;
endmodule
```

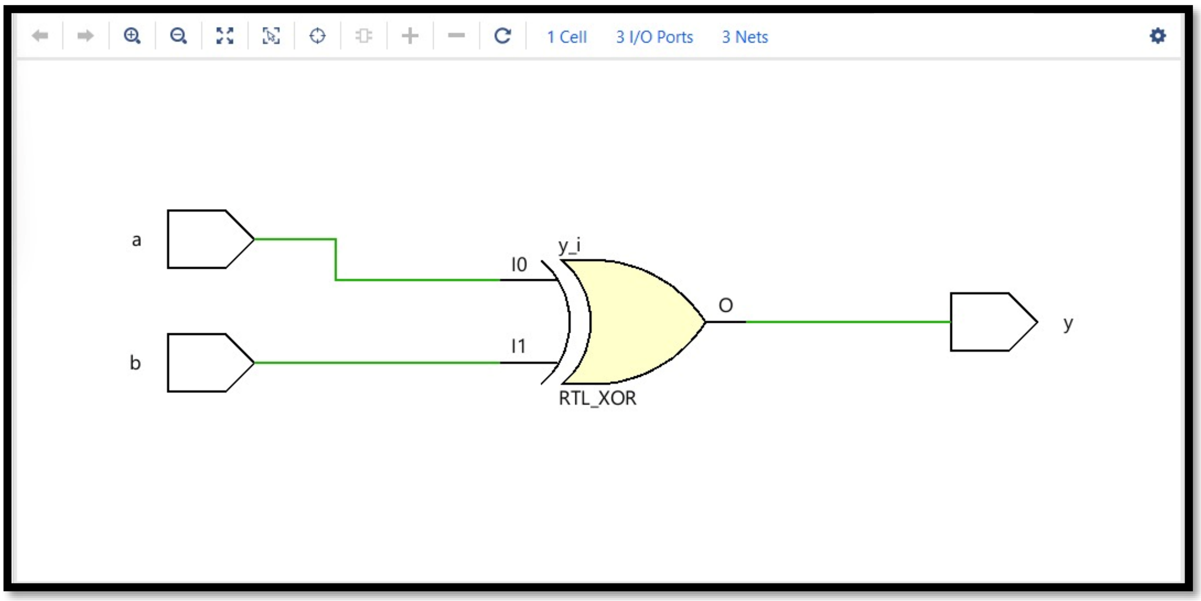
Testbench Code:

```
`timescale 1ns / 1ps

module tb_xor_gate;

reg a, b;
wire y;

// Instantiate XOR gate module
xor_gate uut (
    .a(a),
    .b(b),
    .y(y)
);

initial begin
    a = 0; b = 0; #10;
    a = 0; b = 1; #10;
    a = 1; b = 0; #10;
    a = 1; b = 1; #10;

    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



g) XNOR Gate

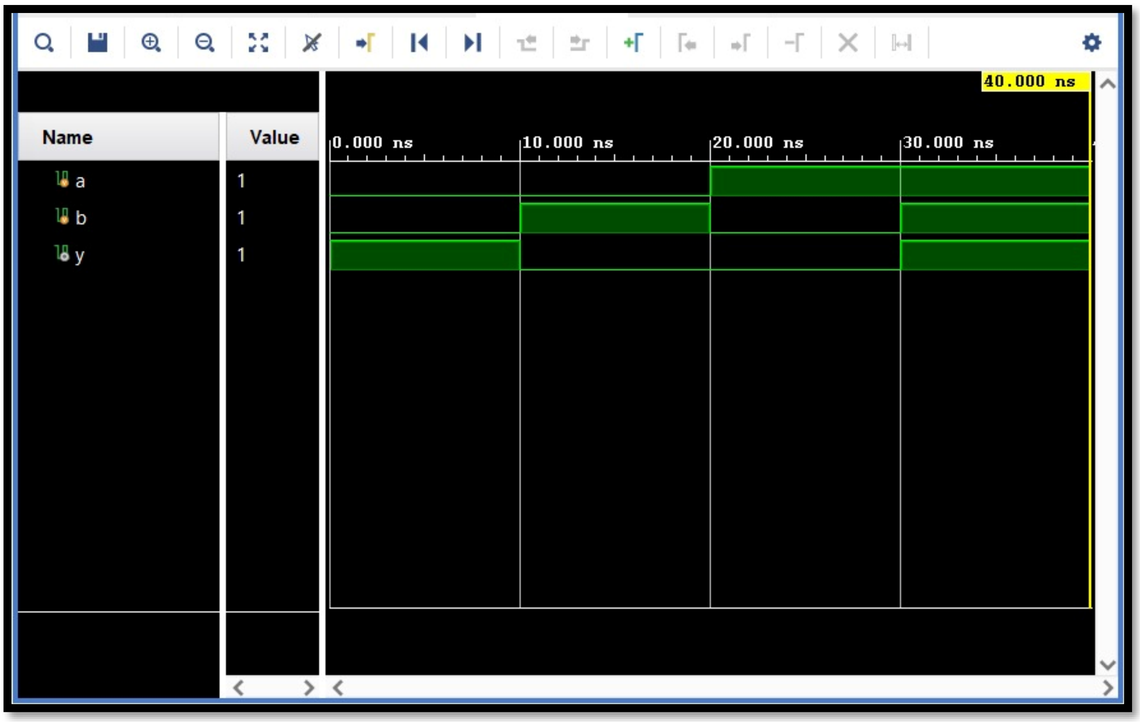
Source Code:

```
`timescale 1ns/1ps
module xnor_gate (
    input  wire a,
    input  wire b,
    output wire y
);
    assign y = ~(a ^ b);
endmodule
```

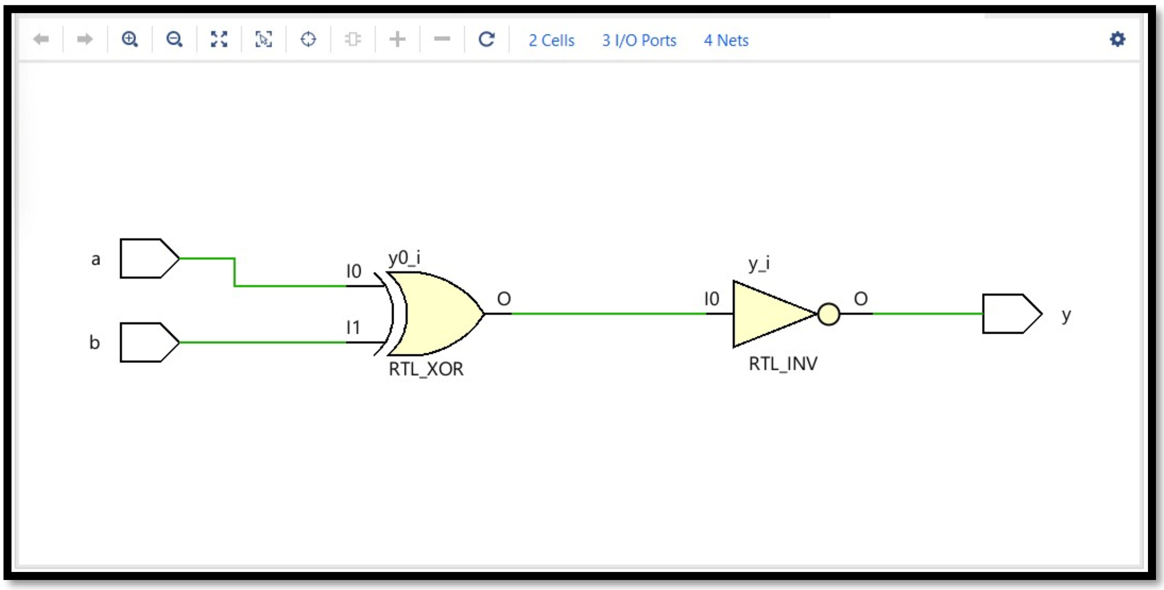
Testbench Code:

```
`timescale 1ns / 1ps

module tb_xnor_gate;

reg a, b;
wire y;

// Instantiate XNOR gate module
xnor_gate uut (
    .a(a),
    .b(b),
    .y(y)
);

initial begin
    a = 0; b = 0; #10;
    a = 0; b = 1; #10;
    a = 1; b = 0; #10;
    a = 1; b = 1; #10;

    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



2) Boolean Expressions

a) Simple Behavioural

Source Code:

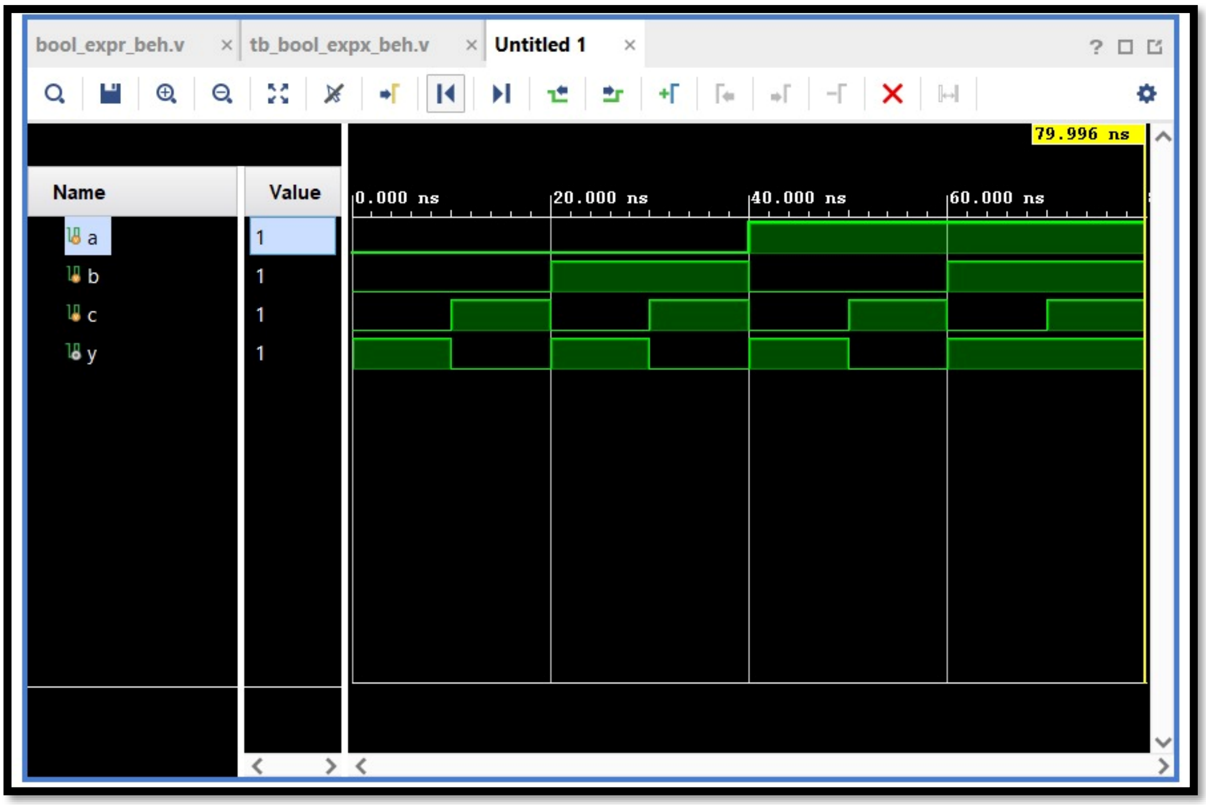
```
module bool_expr_beh(input a, input b, input c, output y);
   assign y = (a & b) | (~c);
endmodule
```

Testbench Code:

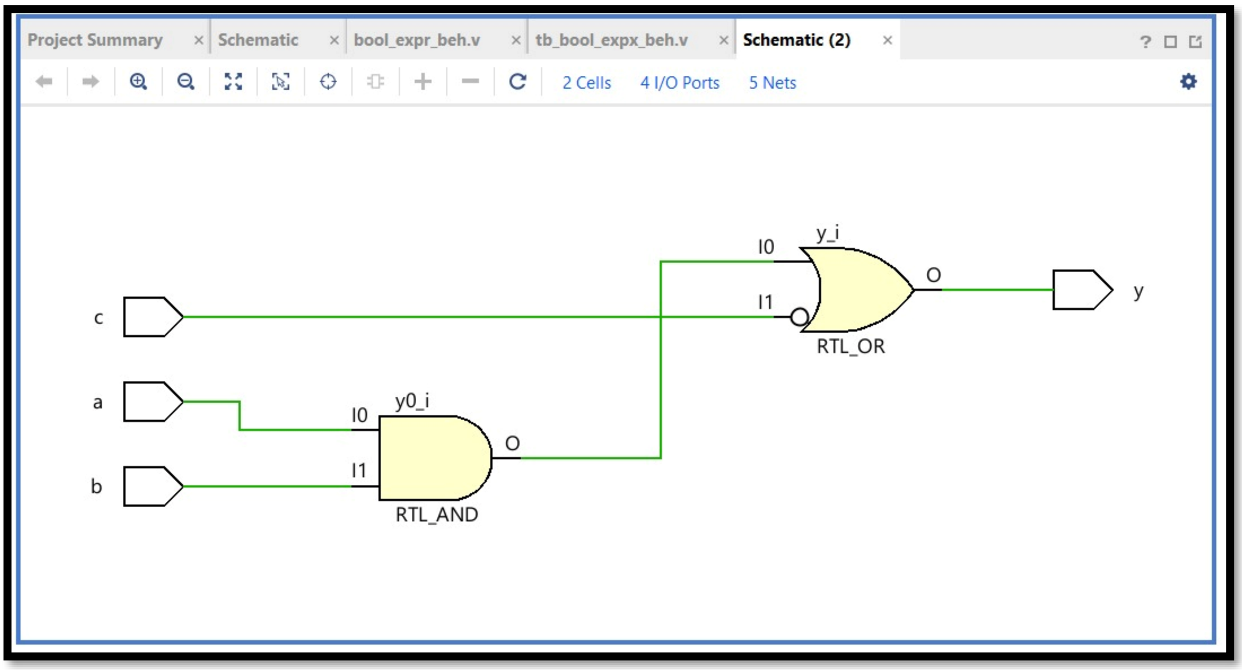
```
`timescale 1ns/1ps


module tb_bool_expr_beh;


reg a, b, c;
wire y;


bool_expr_beh uut(
   .a(a),
   .b(b),
   .c(c),
   .y(y)
);
```

```
initial begin
    a = 0; b = 0; c = 0; #10;
    a = 0; b = 0; c = 1; #10;
    a = 0; b = 1; c = 0; #10;
    a = 0; b = 1; c = 1; #10;
    a = 1; b = 0; c = 0; #10;
    a = 1; b = 0; c = 1; #10;
    a = 1; b = 1; c = 0; #10;
    a = 1; b = 1; c = 1; #10;

    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



b) Complex Behavioural

Source Code:

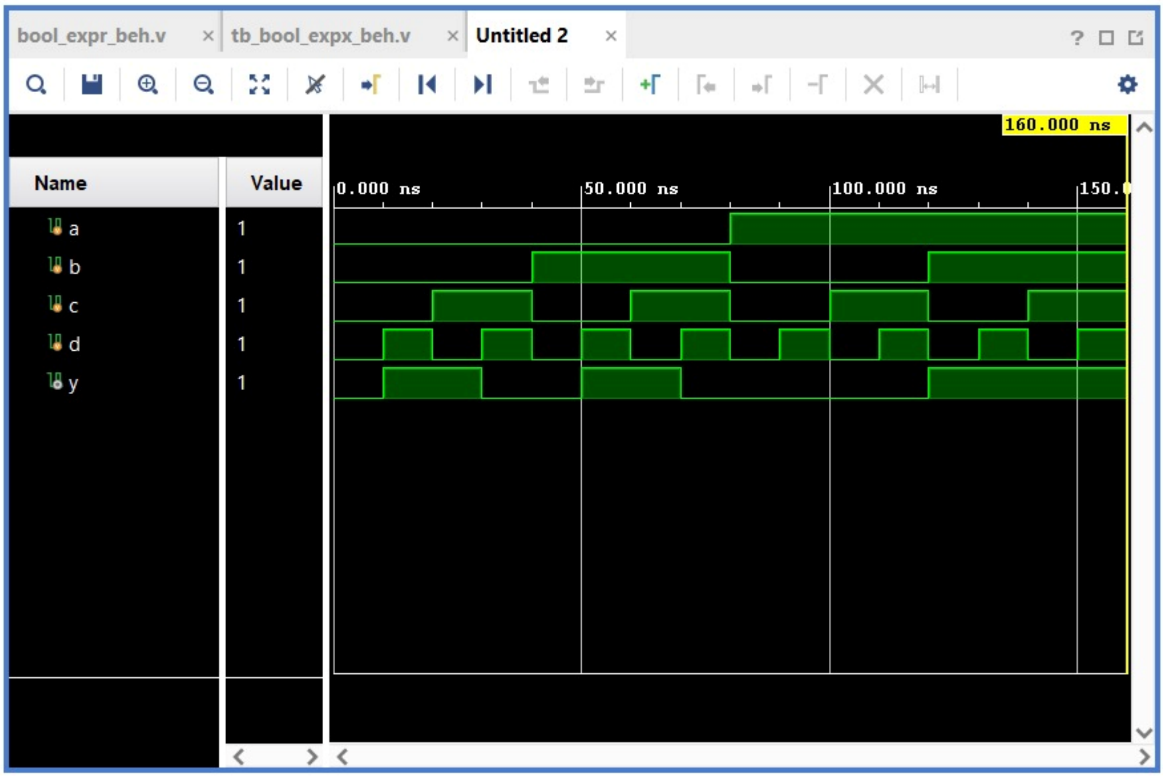
```
module bool_expr_complex_beh(input a, input b, input c, input d, output y);
    assign y = (a & b) | (c ^ d) & (~a | b);
endmodule
```

Testbench Code:

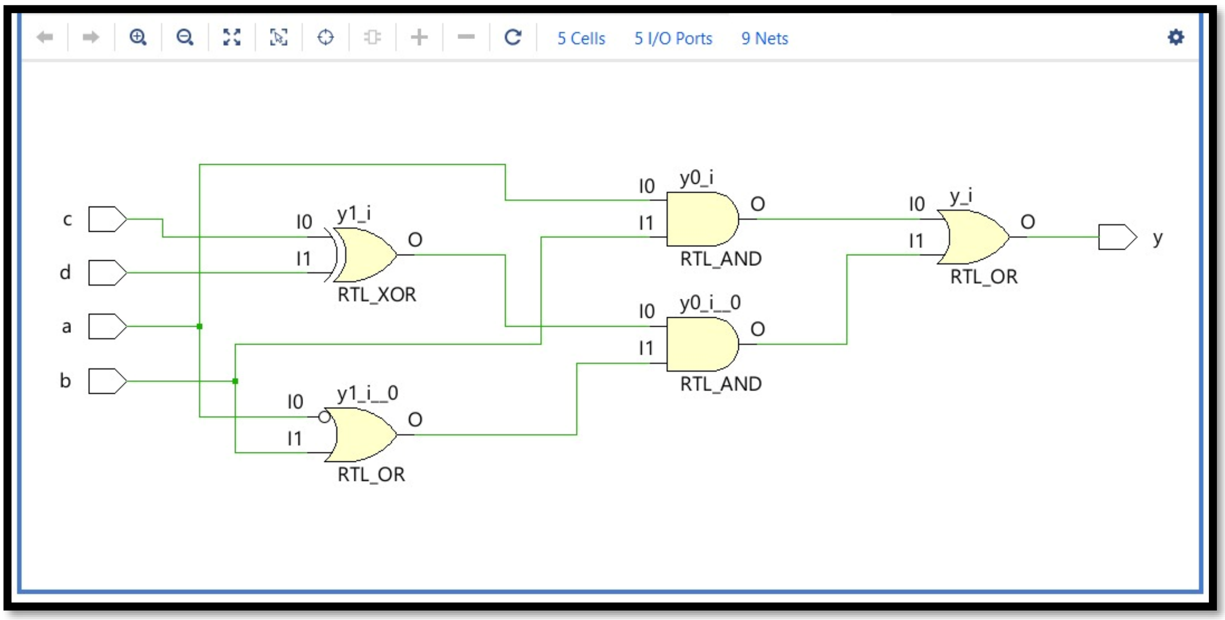
```
`timescale 1ns/1ps


module tb_bool_expr_complex_beh;


reg a, b, c, d;
wire y;


bool_expr_complex_beh uut(
    .a(a),
    .b(b),
    .c(c),
    .d(d),
    .y(y)
);
```

```
initial begin
    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; #10;

    $finish;
end


endmodule
```

Simulation:



RTL Synthesis:



c) Simple Structural

Source Code:

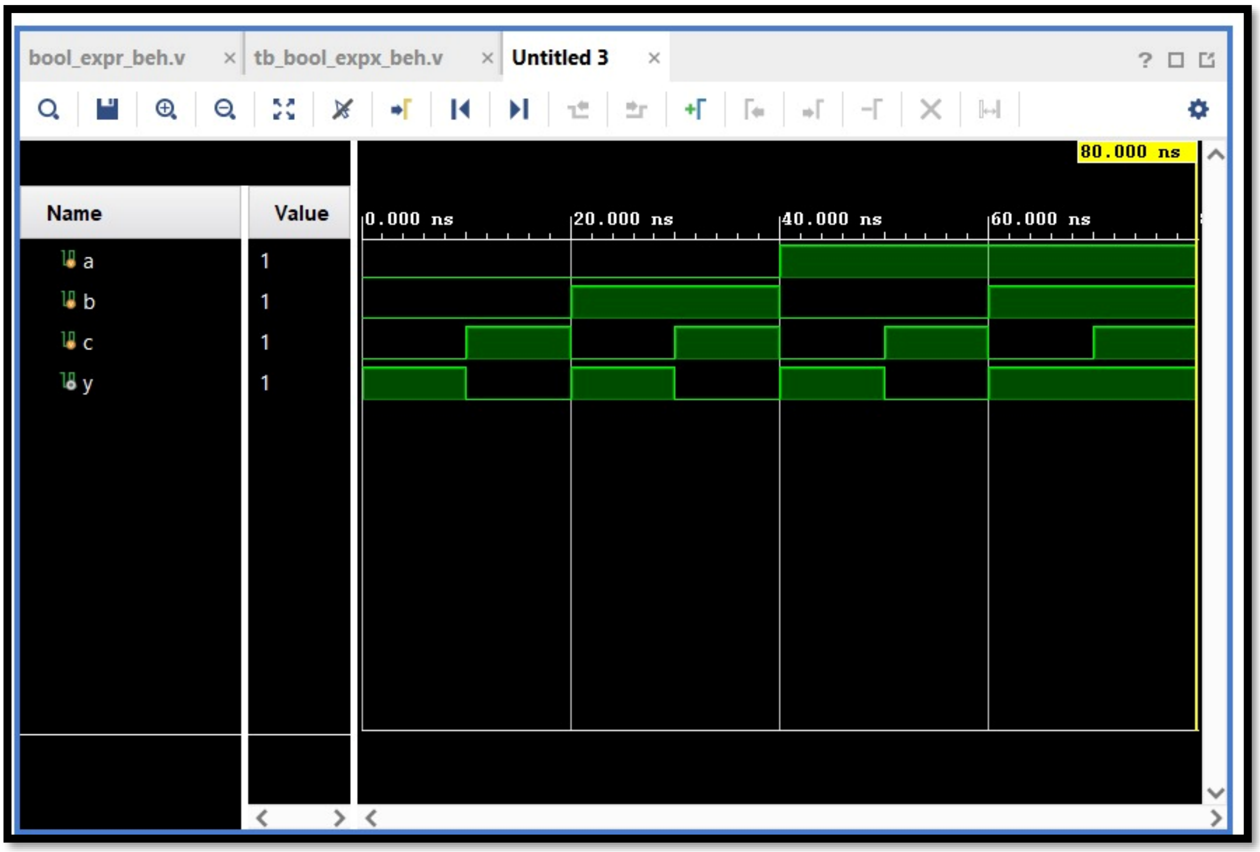
```
`timescale 1ns / 1ps
module bool_expr_struct(input a, input b, input c, output y);
    wire w1, w2;


    and (w1, a, b);
    not (w2, c);
    or (y, w1, w2);
endmodule
```

Testbench Code:

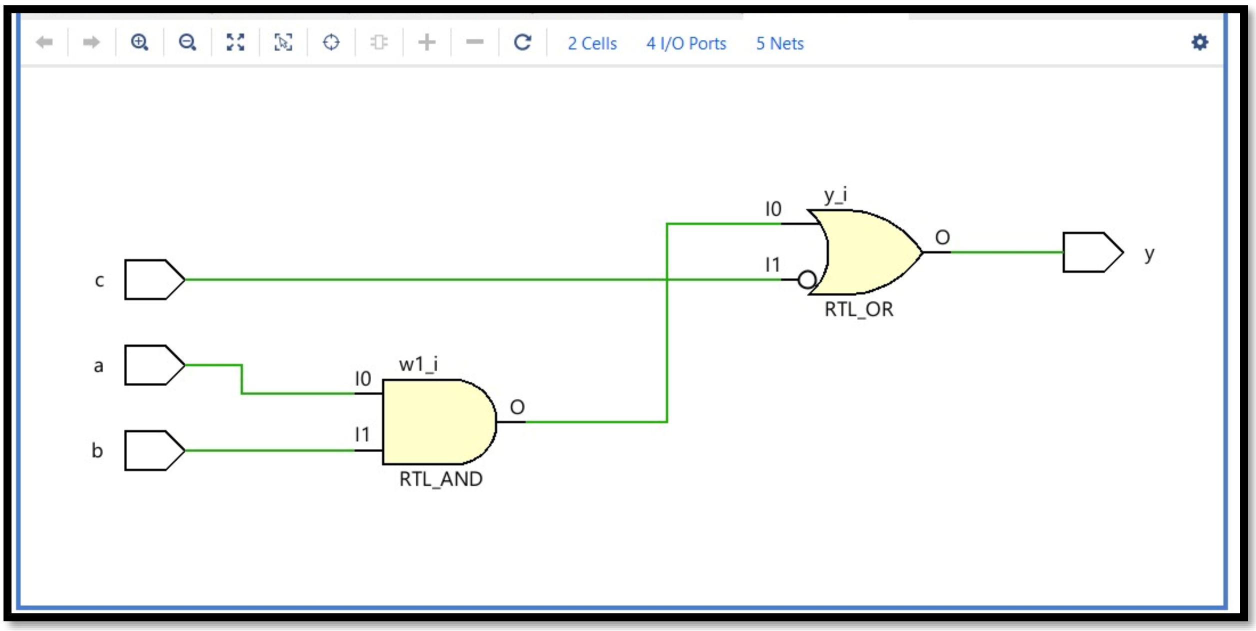
```
`timescale 1ns/1ps

module tb_bool_expr_struct;

reg a, b, c;
wire y;

bool_expr_struct uut(
    .a(a),
    .b(b),
    .c(c),
    .y(y)
);

initial begin
    a=0; b=0; c=0; #10;
    a=0; b=0; c=1; #10;
    a=0; b=1; c=0; #10;
    a=0; b=1; c=1; #10;
    a=1; b=0; c=0; #10;
    a=1; b=0; c=1; #10;
    a=1; b=1; c=0; #10;
    a=1; b=1; c=1; #10;

    $finish;
end
```

endmodule

Simulation:



RTL Synthesis:



d) Complex Structural

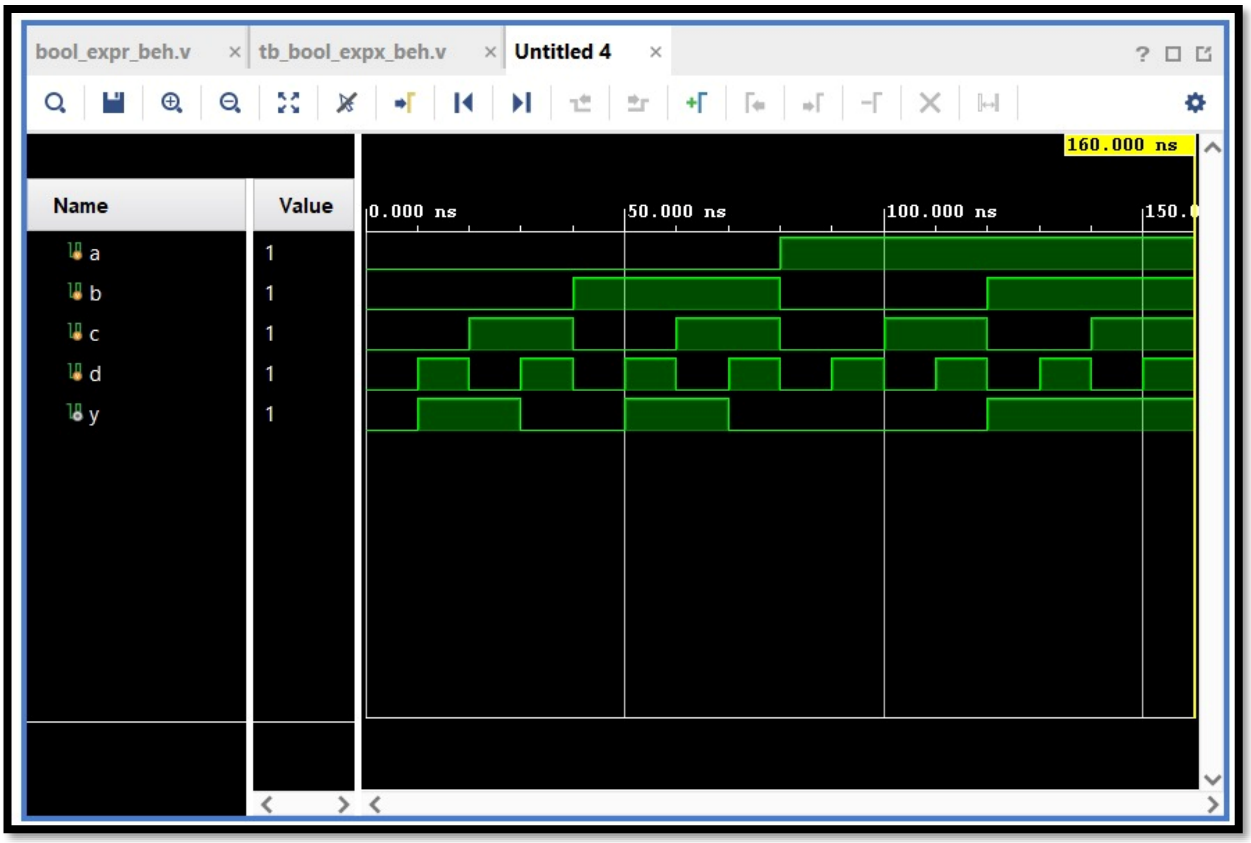
Source Code:

```
module bool_expr_complex_struct(input a, input b, input c, input d, output y);
    wire w1, w2, w3, w4, w5;
    and (w1, a, b);
    xor (w2, c, d);
    or (w3, w1, w2);
    not (w4, a);
    or (w5, w4, b);
    and (y, w5, w3);
endmodule
```

Testbench Code:

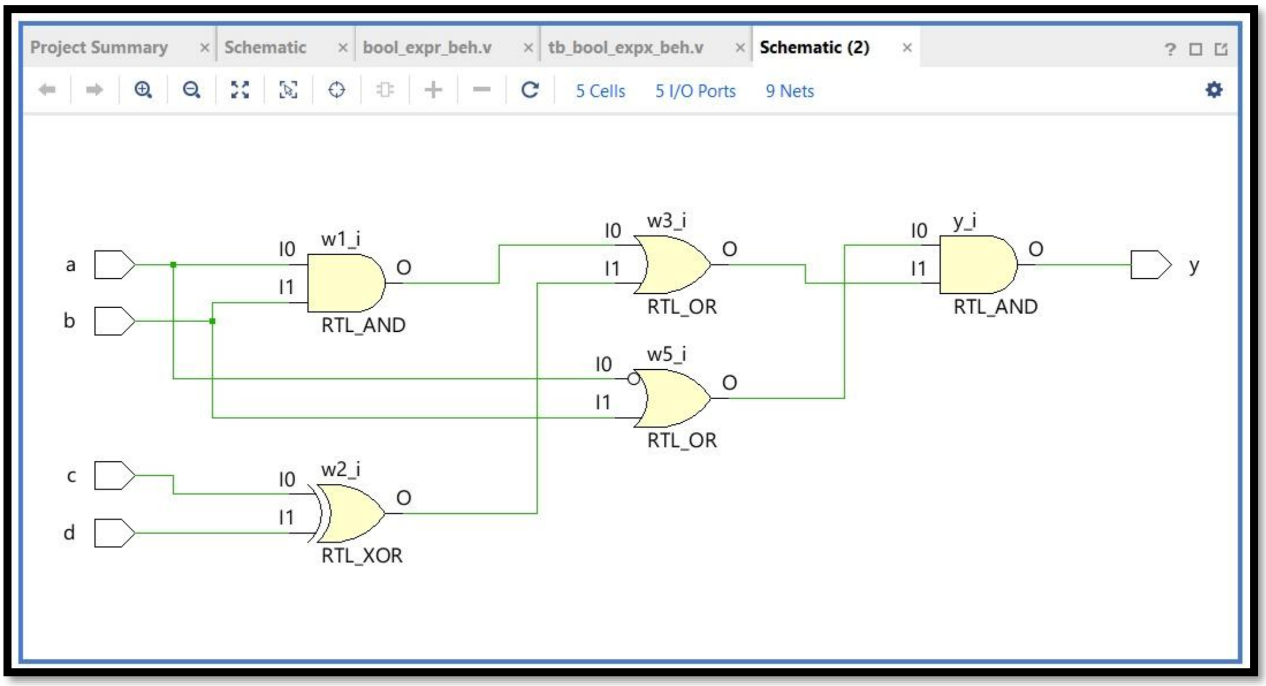
```
`timescale 1ns/1ps

module tb_bool_expr_complex_struct;

reg a, b, c, d;
wire y;

bool_expr_complex_struct uut(
    .a(a),
    .b(b),
    .c(c),
    .d(d),
    .y(y)
);

initial begin
    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; #10;


    $finish;
end


endmodule
```

Simulation:



Name	Value
a	1
b	1
c	1
d	1
y	1

RTL Synthesis:



3) Multiplexers (4 x 1 and 8 x 1)

a) Mux 4x1

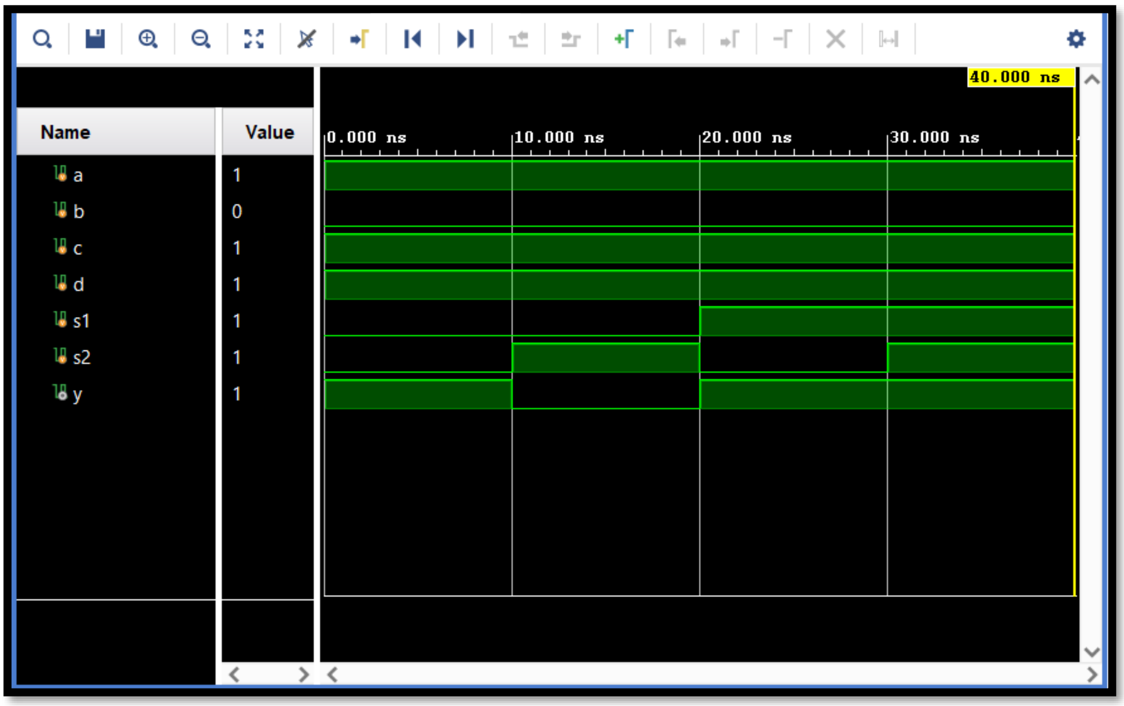
Source Code:

```
`timescale 1ns/1ps
module mux4to1 (
    input wire [3:0] I,    // 4 data inputs: I[3:0]
    input wire [1:0] S,    // 2 select lines: S[1:0]
    output wire Y
);


    assign Y = (S == 2'b00) ? I[0] :
               (S == 2'b01) ? I[1] :
               (S == 2'b10) ? I[2] :
                              I[3];


endmodule
```

Testbench Code:

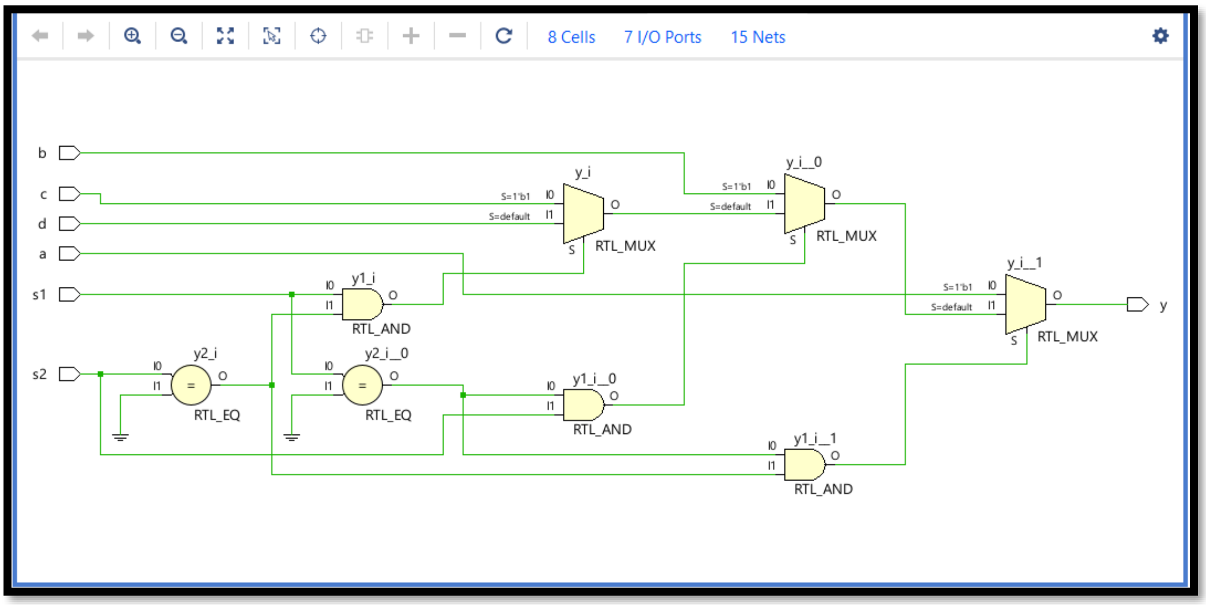
```
module tb_mux4x1;


reg  [3:0] I;
reg  [1:0] S;
wire Y;

mux4x1 dut(I, S, Y);

initial begin
    // Test all combinations
    I = 4'b1010;  // arbitrary pattern

    S = 2'b00; #10;
    S = 2'b01; #10;
    S = 2'b10; #10;
    S = 2'b11; #10;

    $finish;
end

endmodule
```

Simulation:



Name	Value
a	1
b	0
c	1
d	1
s1	1
s2	1
y	1

RTL Synthesis:



b) Mux 8x1

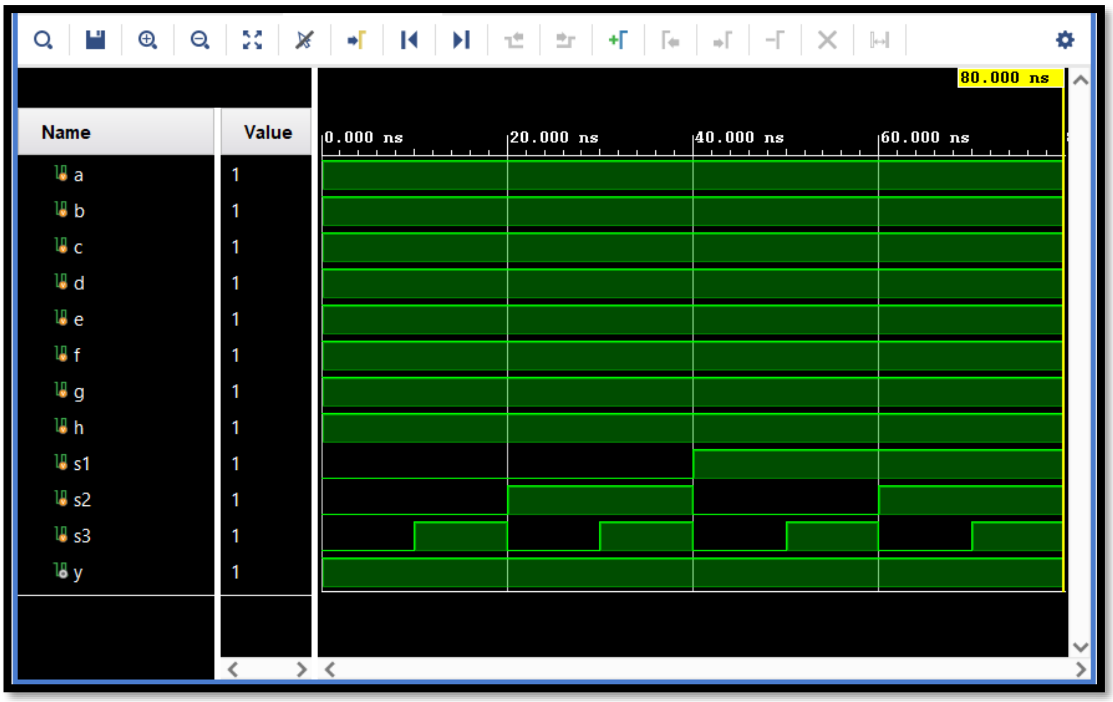
Source Code:

```
`timescale 1ns / 1ps
module mux8to1(
    input a, b, c, d, e, f, g, h,
    input s1, s2, s3,
    output reg y
);
always @(*) begin
    if (s1 == 0 && s2 == 0 && s3 == 0) y = a;
    else if (s1 == 0 && s2 == 0 && s3 == 1) y = b;
    else if (s1 == 0 && s2 == 1 && s3 == 0) y = c;
    else if (s1 == 0 && s2 == 1 && s3 == 1) y = d;
    else if (s1 == 1 && s2 == 0 && s3 == 0) y = e;
    else if (s1 == 1 && s2 == 0 && s3 == 1) y = f;
    else if (s1 == 1 && s2 == 1 && s3 == 0) y = g;
    else y = h;
end
endmodule
```

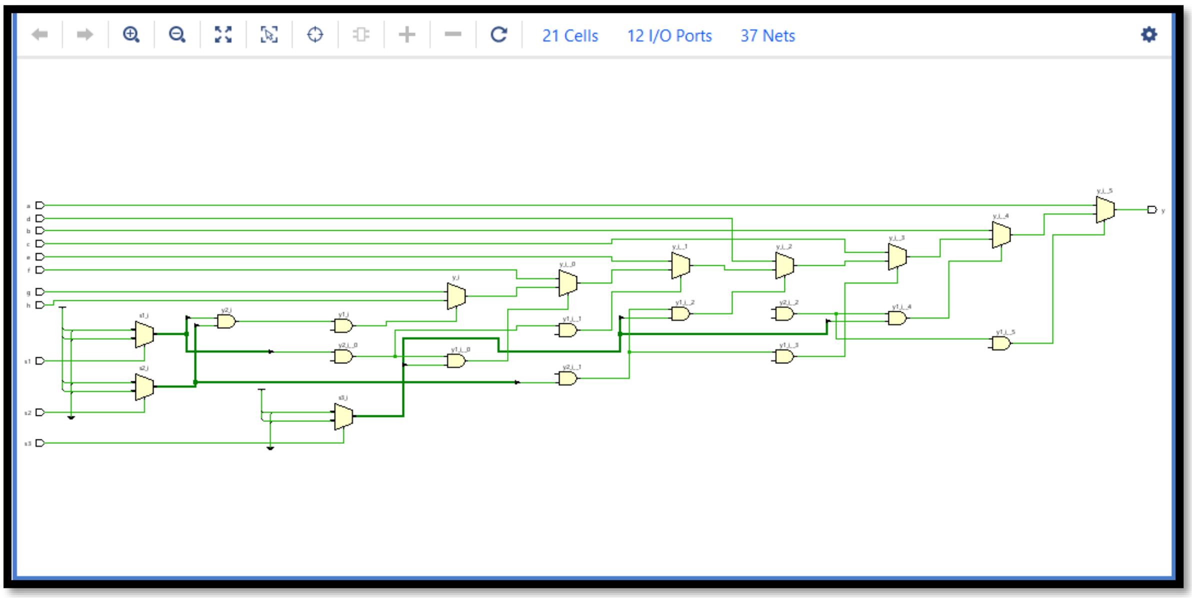
Testbench Code:

```
`timescale 1ns / 1ps
module tb_mux8to1();
reg a, b, c, d, e, f, g, h;
reg s1, s2, s3;
wire y;
mux8to1 uut(
    .a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g), .h(h),
    .s1(s1), .s2(s2), .s3(s3),
    .y(y)
);
initial begin
    a = 1; b = 1; c = 1; d = 1;
    e = 1; f = 1; g = 1; h = 1;
    s1 = 0; s2 = 0; s3 = 0; #10;
    s1 = 0; s2 = 0; s3 = 1; #10;
    s1 = 0; s2 = 1; s3 = 0; #10;
    s1 = 0; s2 = 1; s3 = 1; #10;
    s1 = 1; s2 = 0; s3 = 0; #10;
    s1 = 1; s2 = 0; s3 = 1; #10;
    s1 = 1; s2 = 1; s3 = 0; #10;
    s1 = 1; s2 = 1; s3 = 1; #10;
    $finish;
end
endmodule
```

Simulation:



RTL Synthesis:



4) Priority Encoder

a) 4 to 2 Priority Encoder

Source Code:

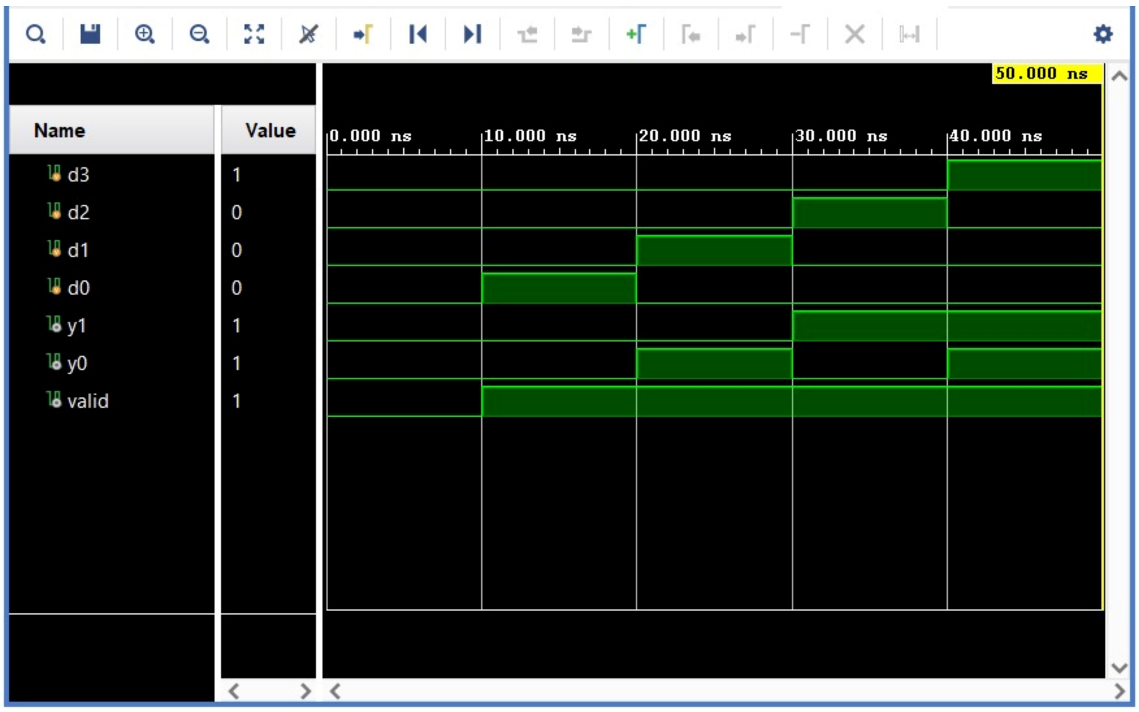
```
`timescale 1ns / 1ps
module priority_encoder(
input d0,d1,d2,d3,
output a,b,v
   );
   assign a = d2|d3;
   assign b=( d1 &(~d2))|d3;
   assign v = d3|d2|d1|d0;
endmodule
```

Testbench Code:

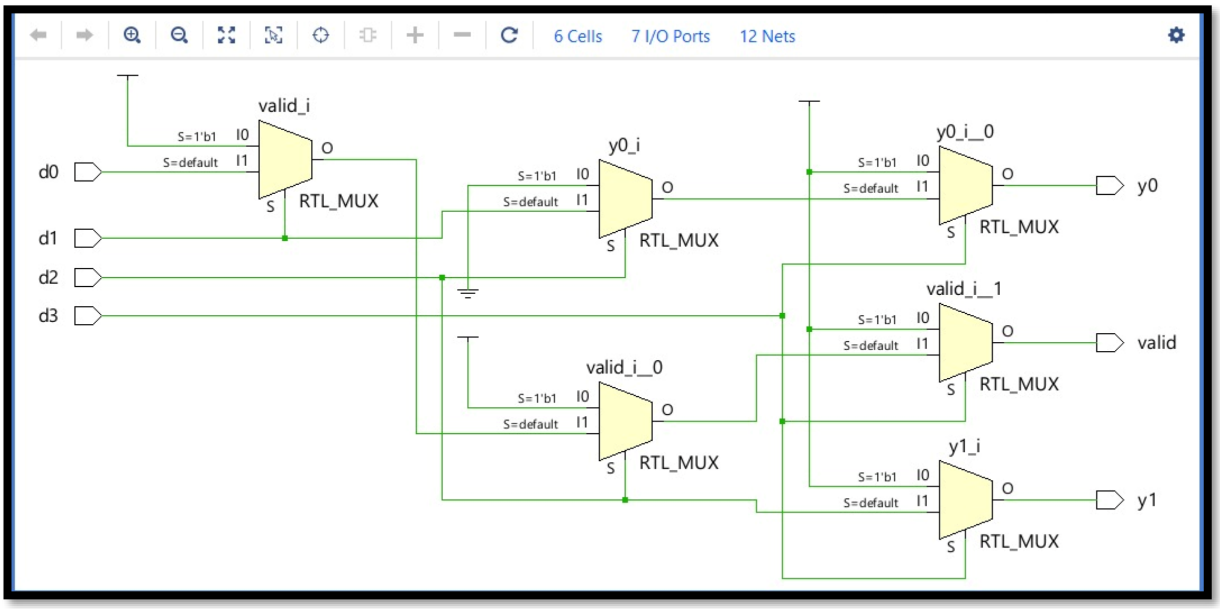
```
`timescale 1ns / 1ps

module tb_priority_encoder(
   );
   reg d0,d1,d2,d3;
   wire a,b ,v;

   priority_encoder uut(d0,d1,d2,d3,a,b,v);

   initial
   begin
   d0=0; d1=0;d2=0;d3=0;
   #10
   d0=1; d1=0;d2=0;d3=0;
   #10
   d0=0; d1=1;d2=0;d3=0;
   #10
```

```
  d0=0; d1=0;d2=1;d3=0;

  #10

  d0=0; d1=0;d2=0;d3=1;

  #10

  d0=1; d1=1;d2=0;d3=0;

  #10

  d0=1; d1=1;d2=1;d3=0;

  #10

  d0=1; d1=1;d2=1;d3=1;

  #10

  d0=1; d1=0;d2=1;d3=0;

  #10

  d0=1; d1=0;d2=0;d3=1;

  #10

  d0=1; d1=0;d2=1;d3=1;

  #10

  d0=0; d1=0;d2=1;d3=1;

  #10

  d0=0; d1=1;d2=1;d3=1;

  #10

  d0=1; d1=1;d2=0;d3=1;

  #10

  d0=0; d1=1;d2=1;d3=0;

  #10

  d0=0; d1=1;d2=0;d3=1;

  #10


 $finish;

  end

endmodule
```

Simulation:



RTL Synthesis:



5) Decoder (2 to 4)

Source Code:

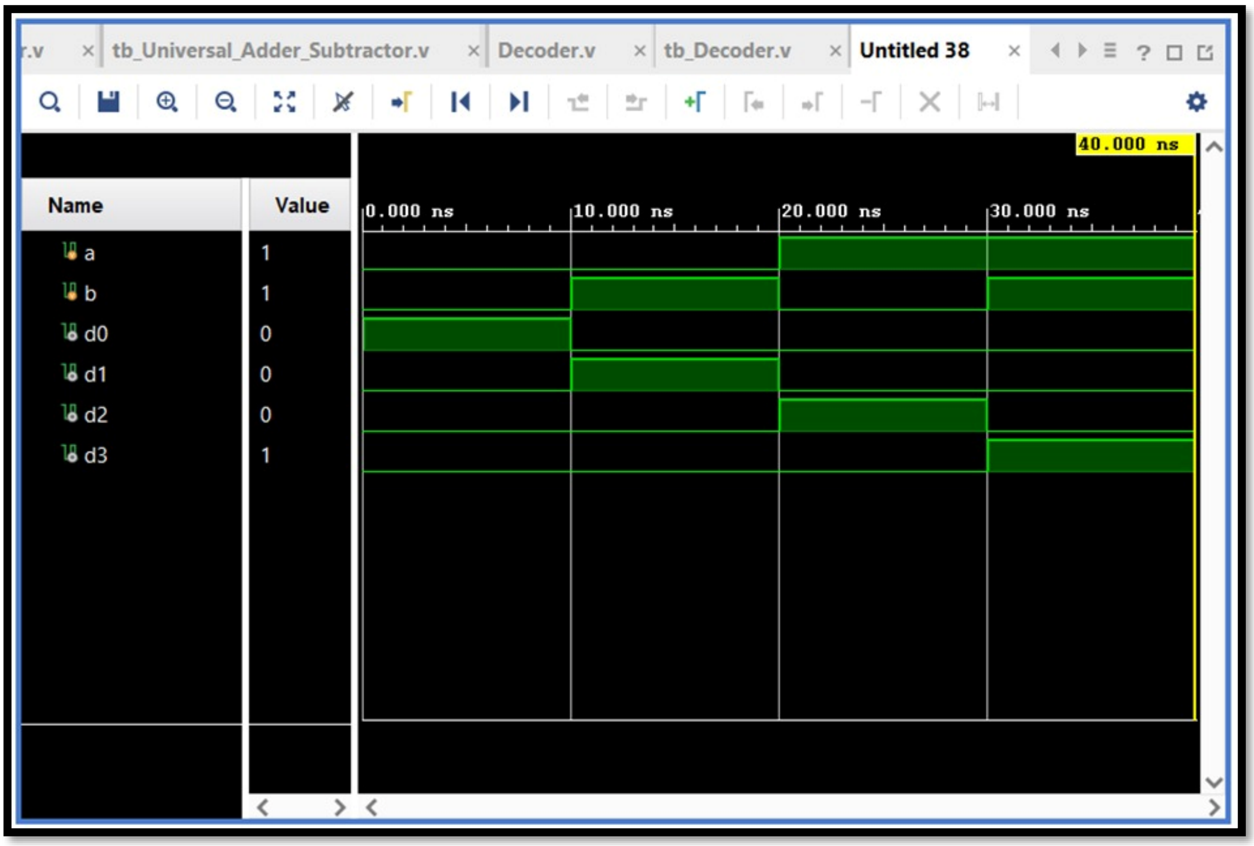
```
`timescale 1ns / 1ps
module decoder(
input a,b,
output d0,d1,d2,d3
   );
   assign d0 = ~a&(~b);
```

```
    assign d1 = ~a&b;
    assign d2 = a&(~b);
    assign d3 = a&b;
endmodule
```

Testbench Code:

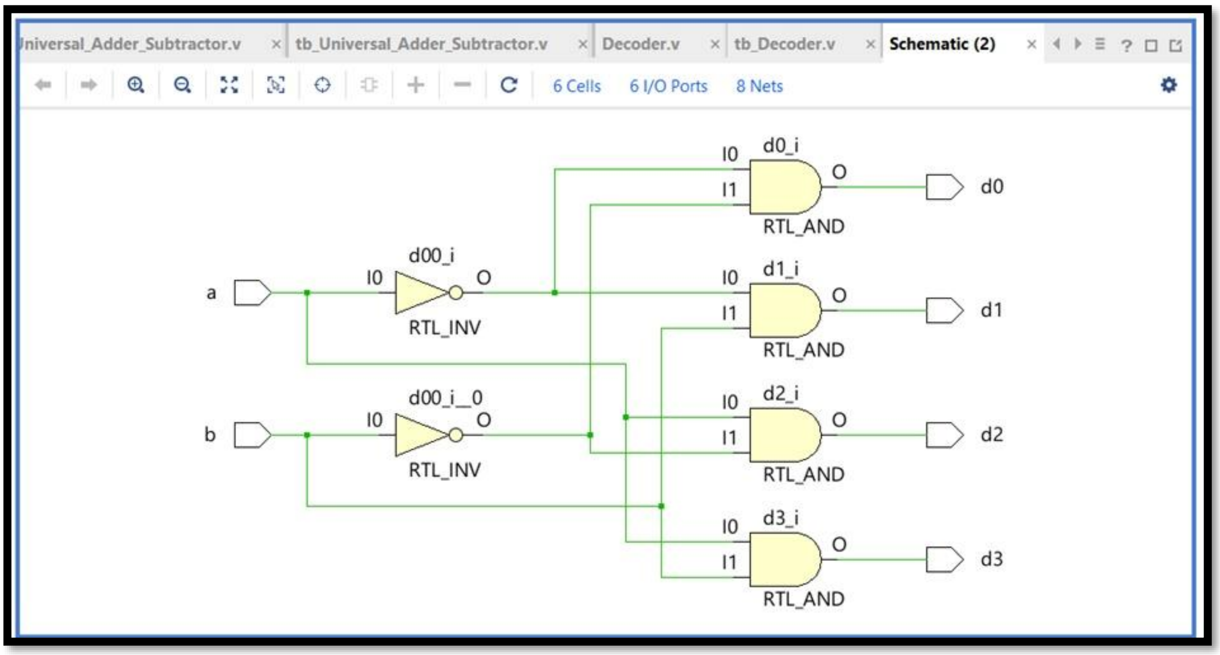
```
`timescale 1ns / 1ps

module tb_decoder(
    );
    reg a,b;
    wire d0,d1,d2,d3;

    decoder uut(a,b,d0,d1,d2,d3);

    initial
    begin
    a =0; b=0;
    #10
     a =0; b=1;
    #10
     a =1; b=0;
    #10
     a =1; b=1;
    #10
    $finish;
    end
endmodule
```

Simulation:



Name	Value
a	1
b	1
d0	0
d1	0
d2	0
d3	1

RTL Synthesis:



6) Half Adder

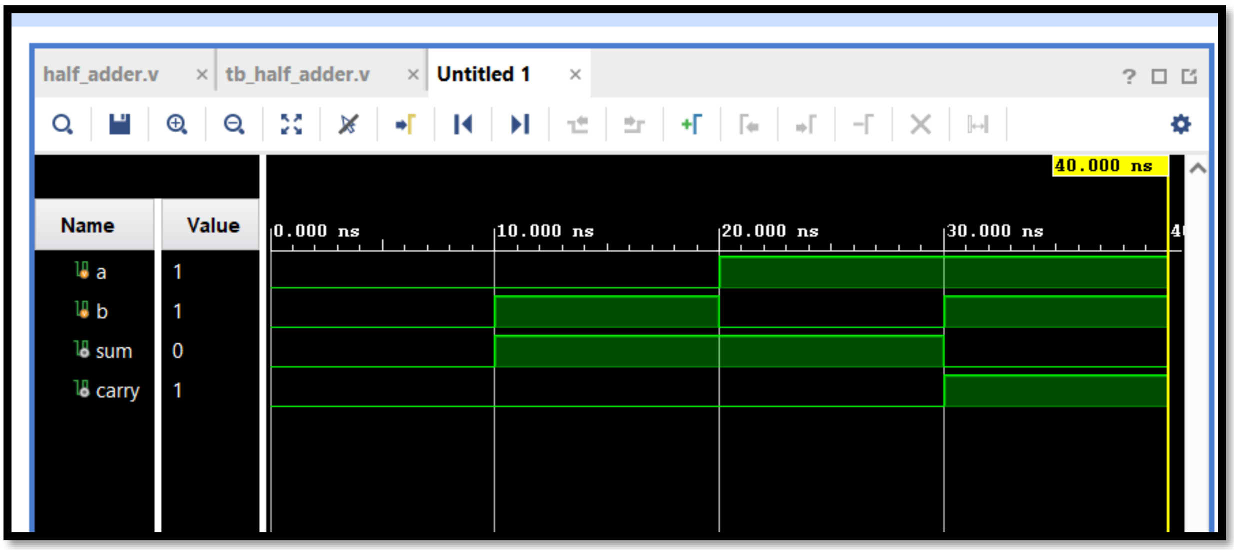
Source Code:

```
`timescale 1ns / 1ps
module half_adder(
    input a, b,
    output sum, carry
);
    assign sum = a ^ b;
    assign carry = a & b;
endmodule
```

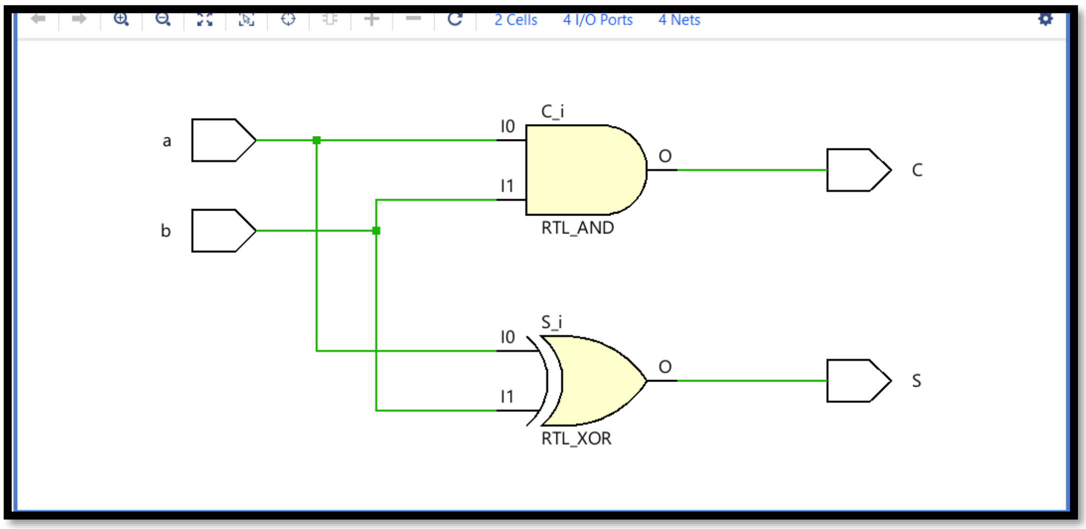
Testbench Code:

```
`timescale 1ns / 1ps
module tb_half_adder;
reg a, b;
wire sum, carry;

half_adder dut(a, b, sum, carry);

initial begin
    $display("A B | SUM CARRY");
    a = 0; b = 0; #10;
    a = 0; b = 1; #10;
    a = 1; b = 0; #10;
    a = 1; b = 1; #10;
    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



7) Full Adder

Source Code:

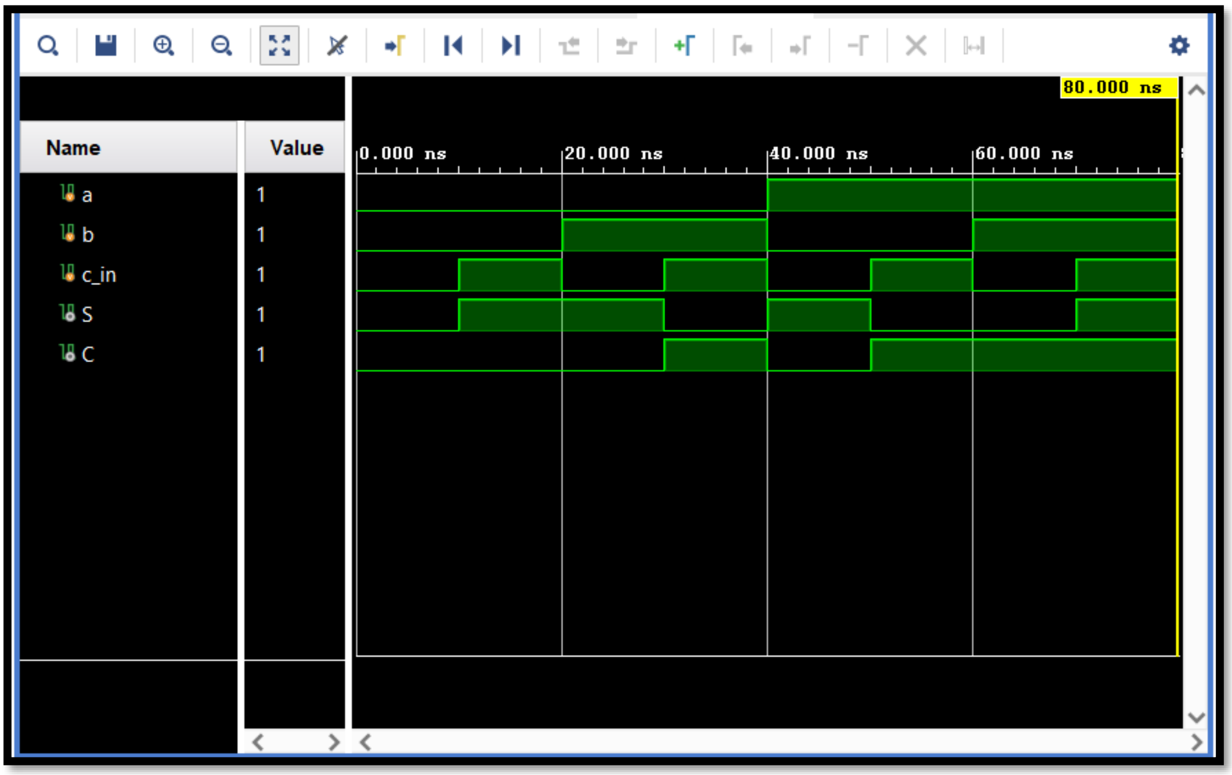
```
`timescale 1ns / 1ps
module full_adder(
    input a, b, cin,
    output sum, cout
);
    assign sum  = a ^ b ^ cin;
    assign cout = (a & b) | (b & cin) | (a & cin);
endmodule
```

Testbench Code:

```
`timescale 1ns / 1ps
```

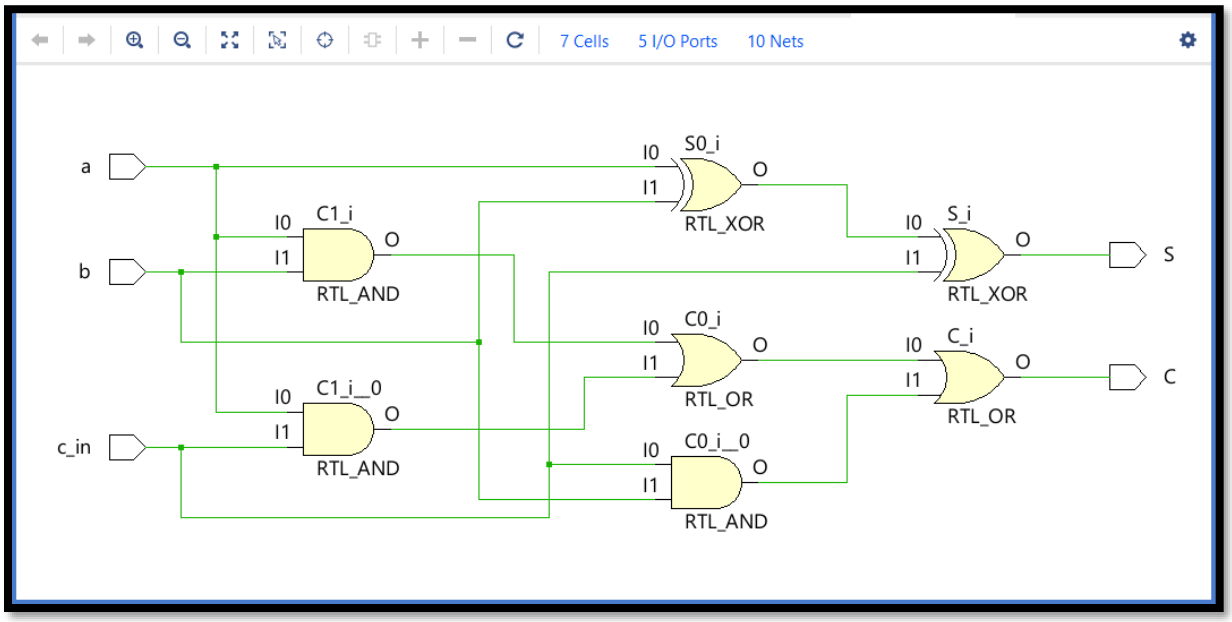
```
module tb_full_adder;


reg a, b, cin;
wire sum, cout;


full_adder dut(a, b, cin, sum, cout);


initial begin
   a=0; b=0; cin=0; #10;
   a=1; b=0; cin=1; #10;
   a=1; b=1; cin=0; #10;
   a=1; b=1; cin=1; #10;
   $finish;
end


endmodule
```

Simulation:



Name	Value
a	1
b	1
c_in	1
S	1
C	1

RTL Synthesis:



8) Half Subtractor

Source Code:

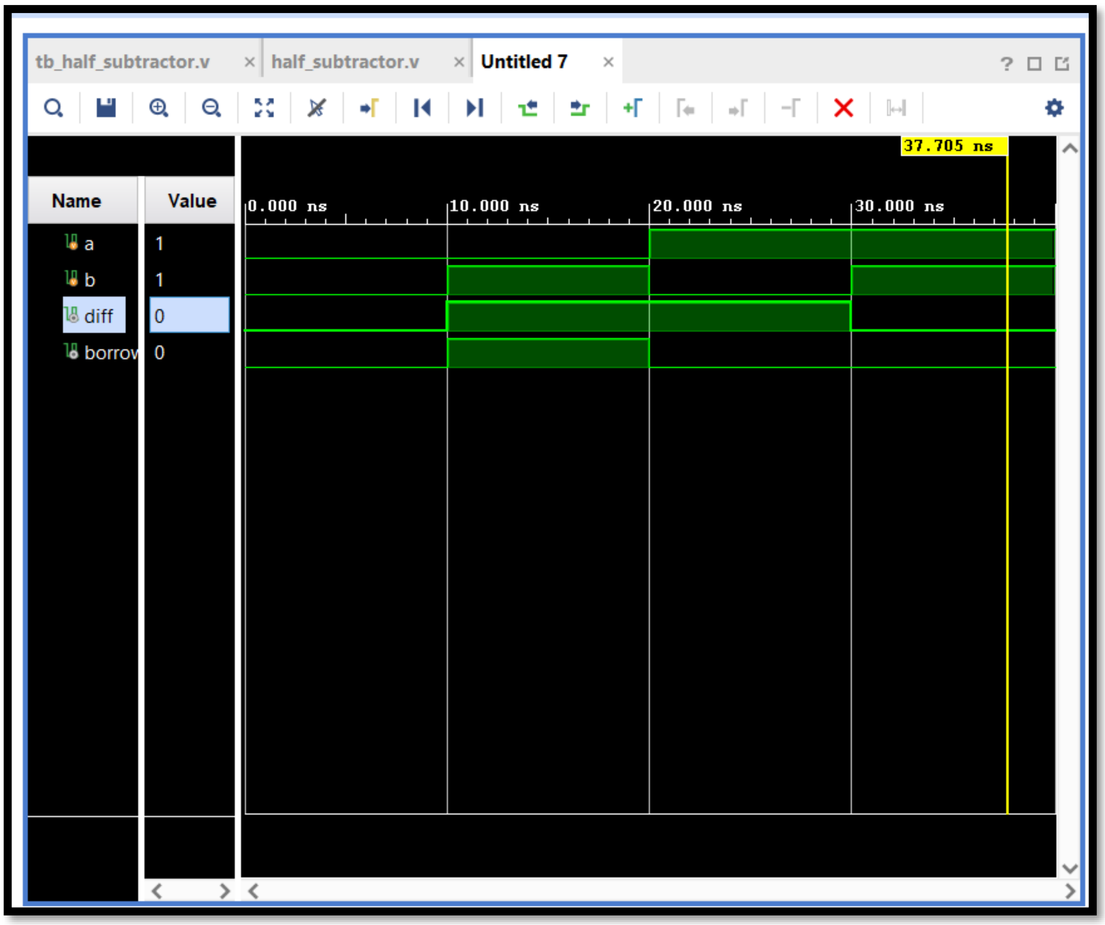
```
`timescale 1ns / 1ps
module half_subtractor(
    input a, b,
    output diff, borrow
);
    assign diff   = a ^ b;
    assign borrow = (~a) & b;
endmodule
```

Testbench Code:

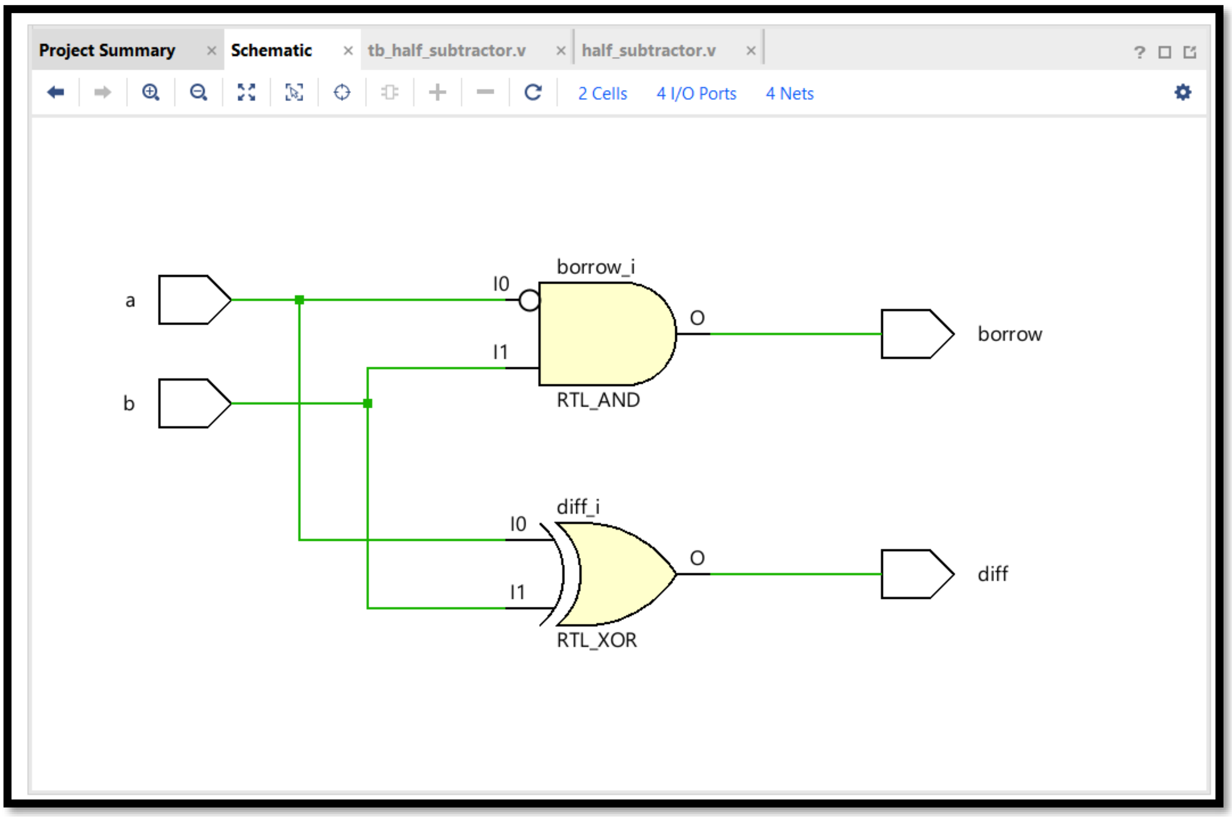
```
`timescale 1ns / 1ps
module tb_half_subtractor;

reg a, b;
wire diff, borrow;

half_subtractor dut(a, b, diff, borrow);

initial begin
```

```
    a=0; b=0; #10;
    a=0; b=1; #10;
    a=1; b=0; #10;
    a=1; b=1; #10;
    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



9) **Full Subtractor**

Source Code:

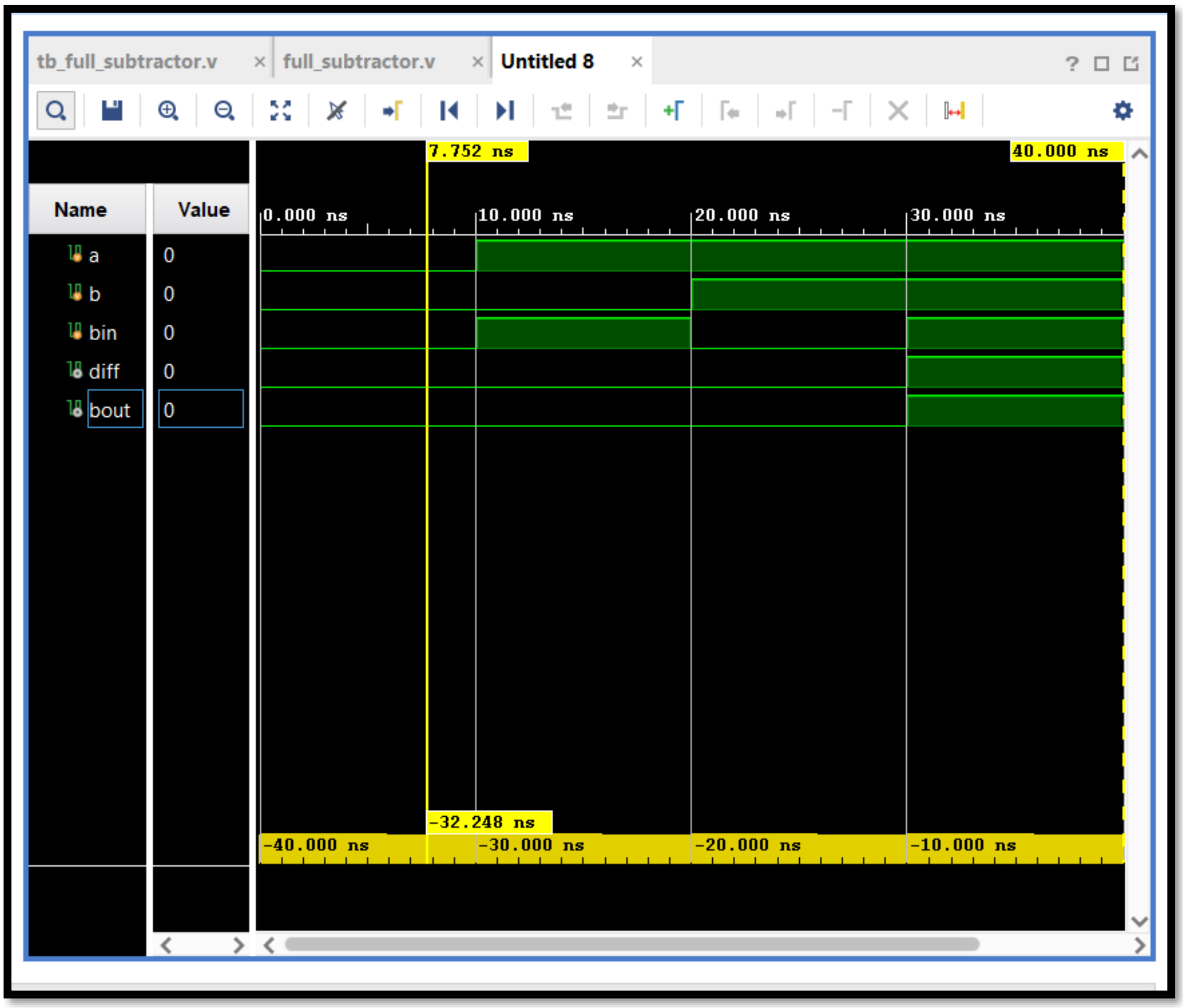
```
`timescale 1ns / 1ps
module full_subtractor(
    input a, b, bin,
    output diff, bout
);
    assign diff = a ^ b ^ bin;
    assign bout = (~a & b) | (b & bin) | (~a & bin);
endmodule
```

Testbench Code:

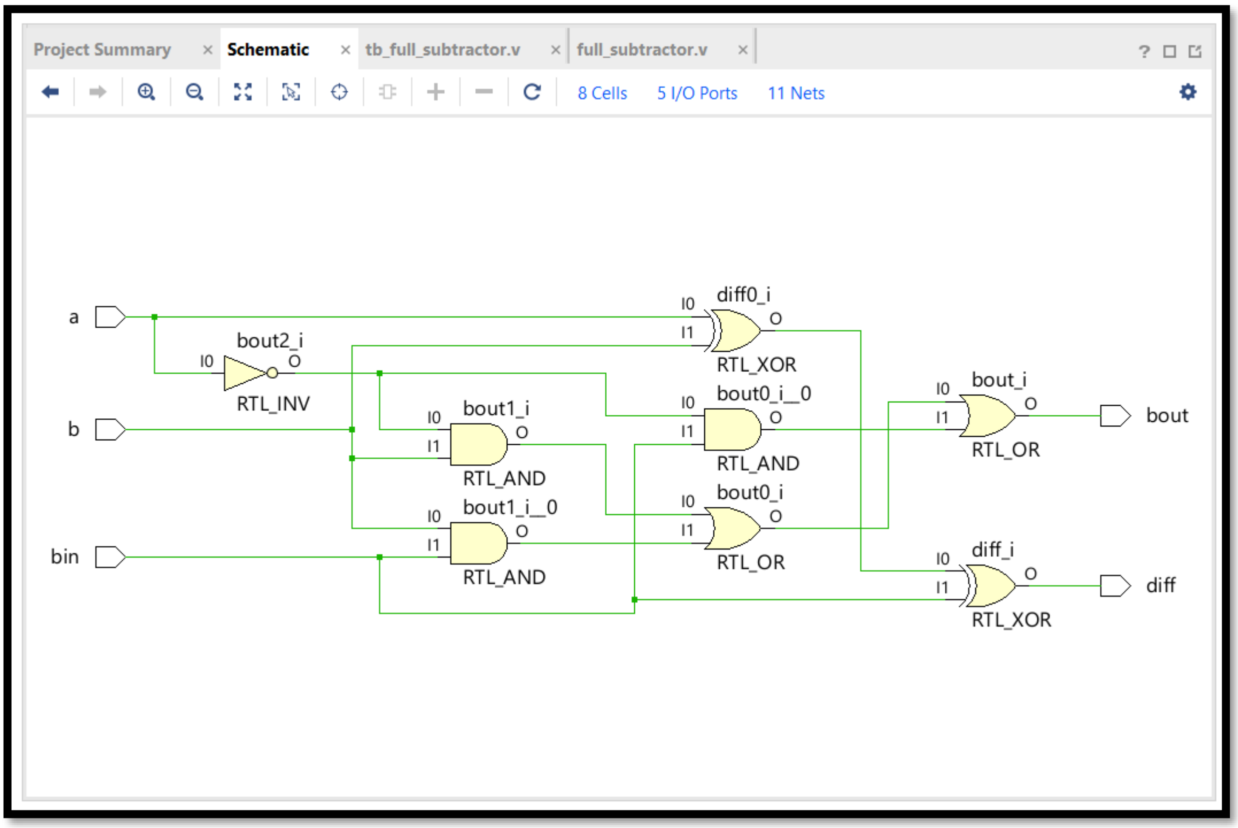
```
`timescale 1ns / 1ps
module tb_full_subtractor;

reg a, b, bin;
wire diff, bout;

full_subtractor dut(a, b, bin, diff, bout);
```

```
initial begin
    a=0; b=0; bin=0; #10;
    a=1; b=0; bin=1; #10;
    a=1; b=1; bin=0; #10;
    a=1; b=1; bin=1; #10;
    $finish;
end


endmodule
```

Simulation:



RTL Synthesis:



10) Universal Adder / Subtractor with Overflow Check

Source Code:

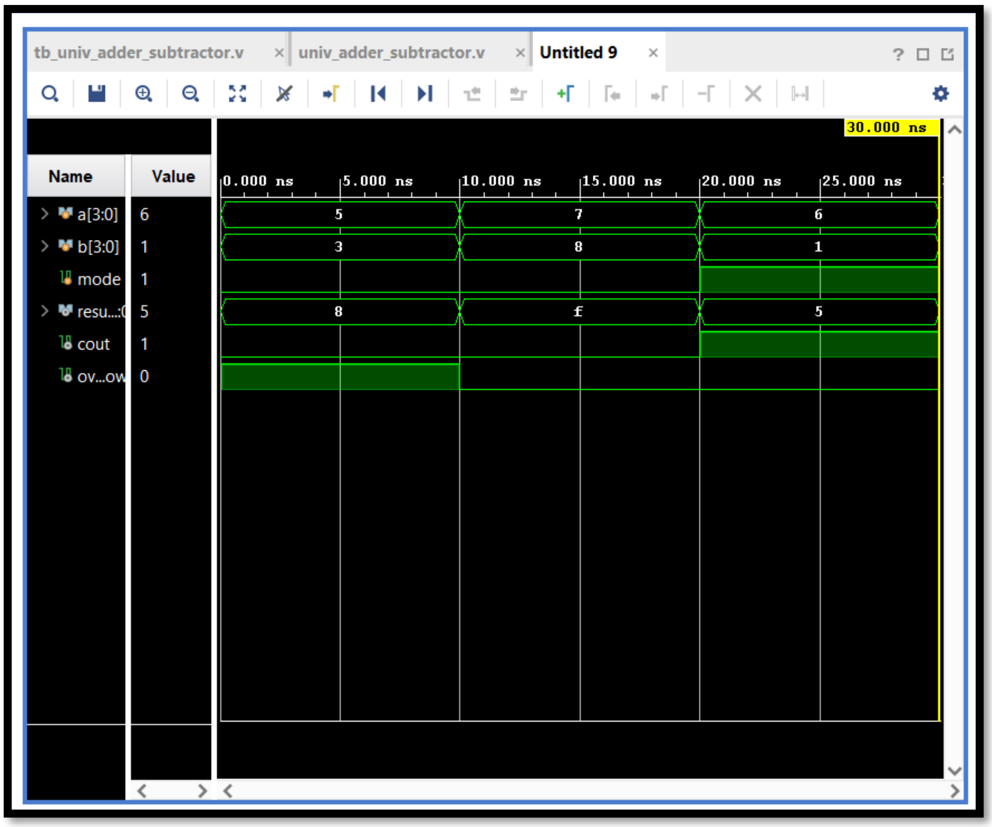
```
`timescale 1ns / 1ps
module univ_adder_subtractor(
    input  [3:0] a, b,
    input  mode,              // 0 = add, 1 = subtract
    output [3:0] result,
    output cout, overflow
);
    wire [3:0] b_mod = mode ? ~b : b;
    wire cin = mode;

    assign {cout, result} = a + b_mod + cin;

    assign overflow = (a[3] ~^ b_mod[3]) & (result[3] ^ a[3]);
endmodule
```

Testbench Code:

```
`timescale 1ns / 1ps
```

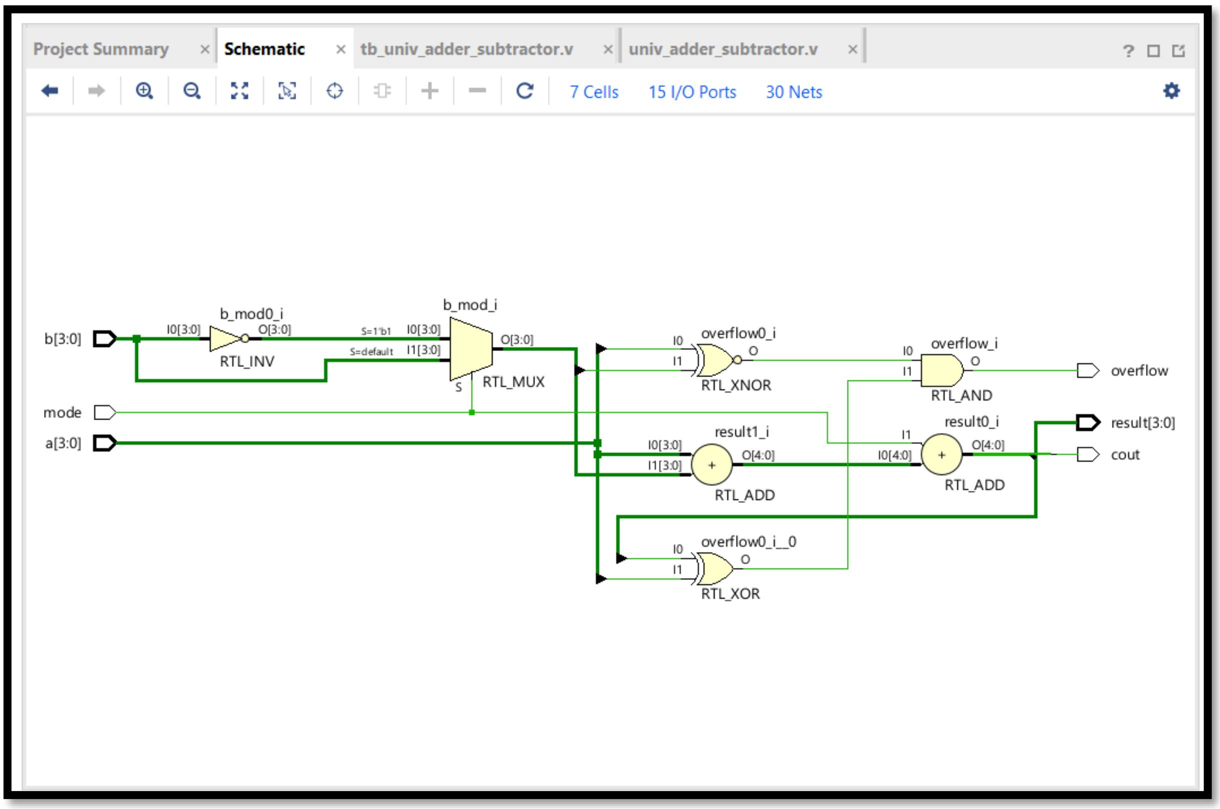
```
module tb_univ_adder_subtractor;

reg [3:0] a, b;
reg mode;
wire [3:0] result;
wire cout, overflow;

univ_adder_subtractor dut(a, b, mode, result, cout, overflow);

initial begin
   a=4'b0101; b=4'b0011; mode=0; #10;
   a=4'b0111; b=4'b1000; mode=0; #10;
   a=4'b0110; b=4'b0001; mode=1; #10;
   $finish;
end

endmodule
```

Simulation:



Name	Value	0–10 ns	10–20 ns	20–30 ns
a[3:0]	6	5	7	6
b[3:0]	1	3	8	1
mode	1	0	0	1
resu...:0	5	8	f	5
cout	1	0	0	1
ov...ow	0	1	0	0

RTL Synthesis:



11) SR Latch

Source Code:

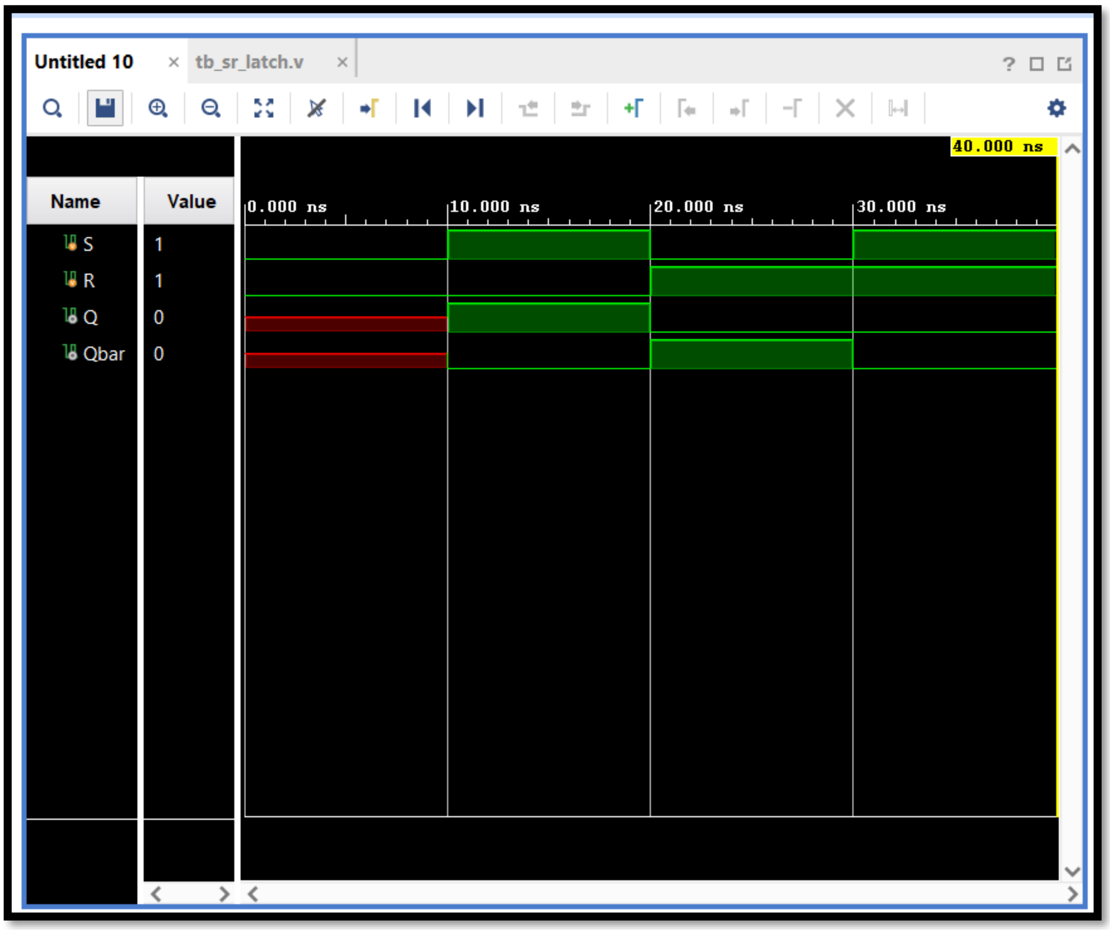
```
`timescale 1ns / 1ps
module sr_latch(
    input S, R,
    output Q, Qbar
);
    assign Q    = ~(R | Qbar);
    assign Qbar = ~(S | Q);
endmodule
```

Testbench Code:

```
`timescale 1ns / 1ps
module tb_sr_latch;

reg S, R;
wire Q, Qbar;
```

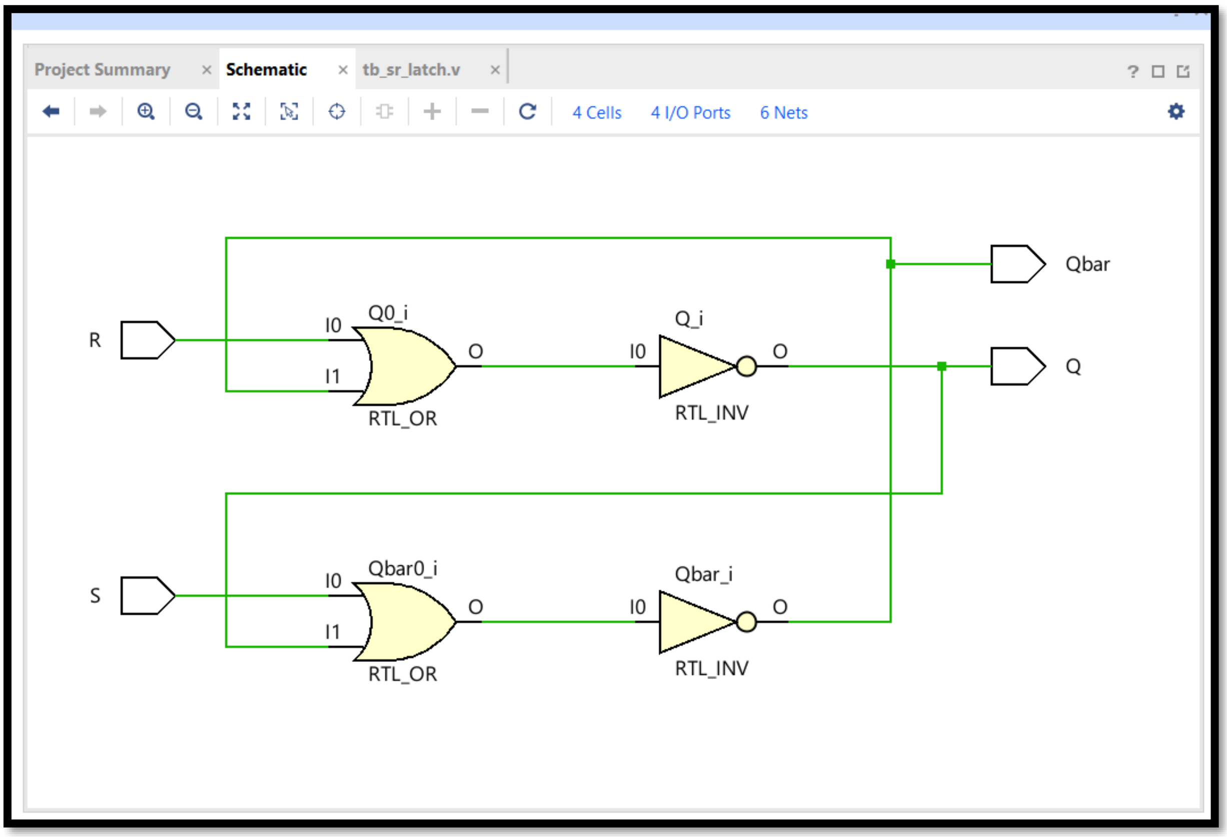
```
sr_latch dut(S, R, Q, Qbar);


initial begin
    S=0; R=0; #10;
    S=1; R=0; #10;
    S=0; R=1; #10;
    S=1; R=1; #10; // invalid
    $finish;
end


endmodule
```

Simulation:



Name	Value
S	1
R	1
Q	0
Qbar	0

RTL Synthesis:



12) SR - Flip Flop

Source Code:

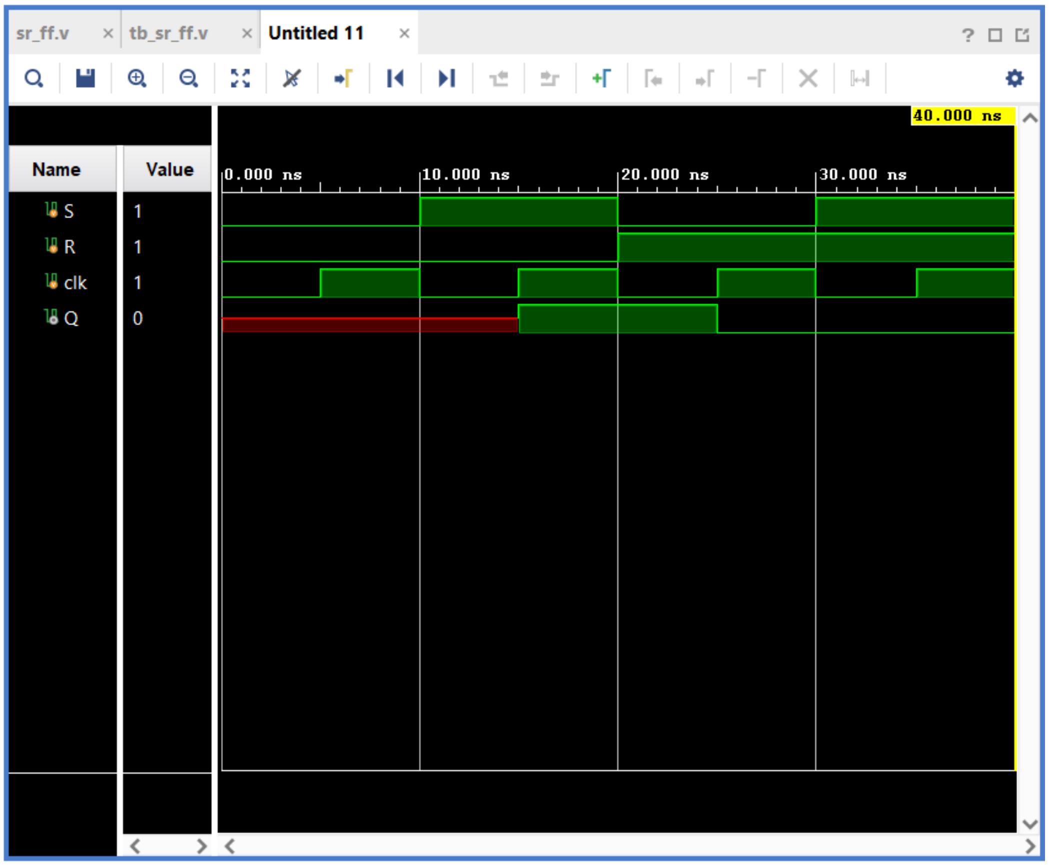
```
`timescale 1ns / 1ps
module sr_ff(
    input S, R, clk,
    output reg Q
);
always @(posedge clk) begin
    if (S && ~R)      Q <= 1;
    else if (R && ~S) Q <= 0;
    else              Q <= Q;
end
endmodule
```

Testbench Code:

```
`timescale 1ns / 1ps
module tb_sr_ff;
```

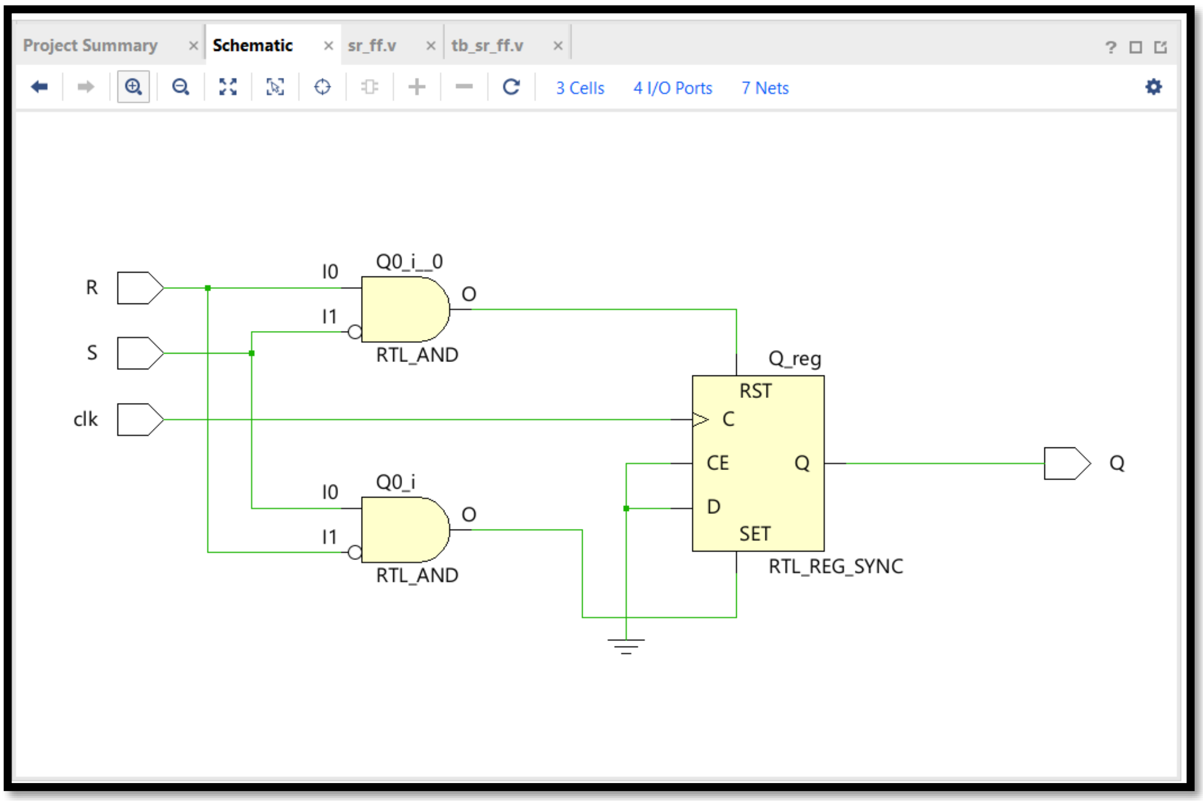
```
reg S, R, clk;
wire Q;

sr_ff dut(S, R, clk, Q);

initial clk = 0;
always #5 clk = ~clk;

initial begin
   S=0; R=0; #10;
   S=1; R=0; #10;
   S=0; R=1; #10;
   S=1; R=1; #10;
   $finish;
end
endmodule
```

Simulation:



Name	Value
S	1
R	1
clk	1
Q	0

RTL Synthesis:



13) JK - Flip Flop

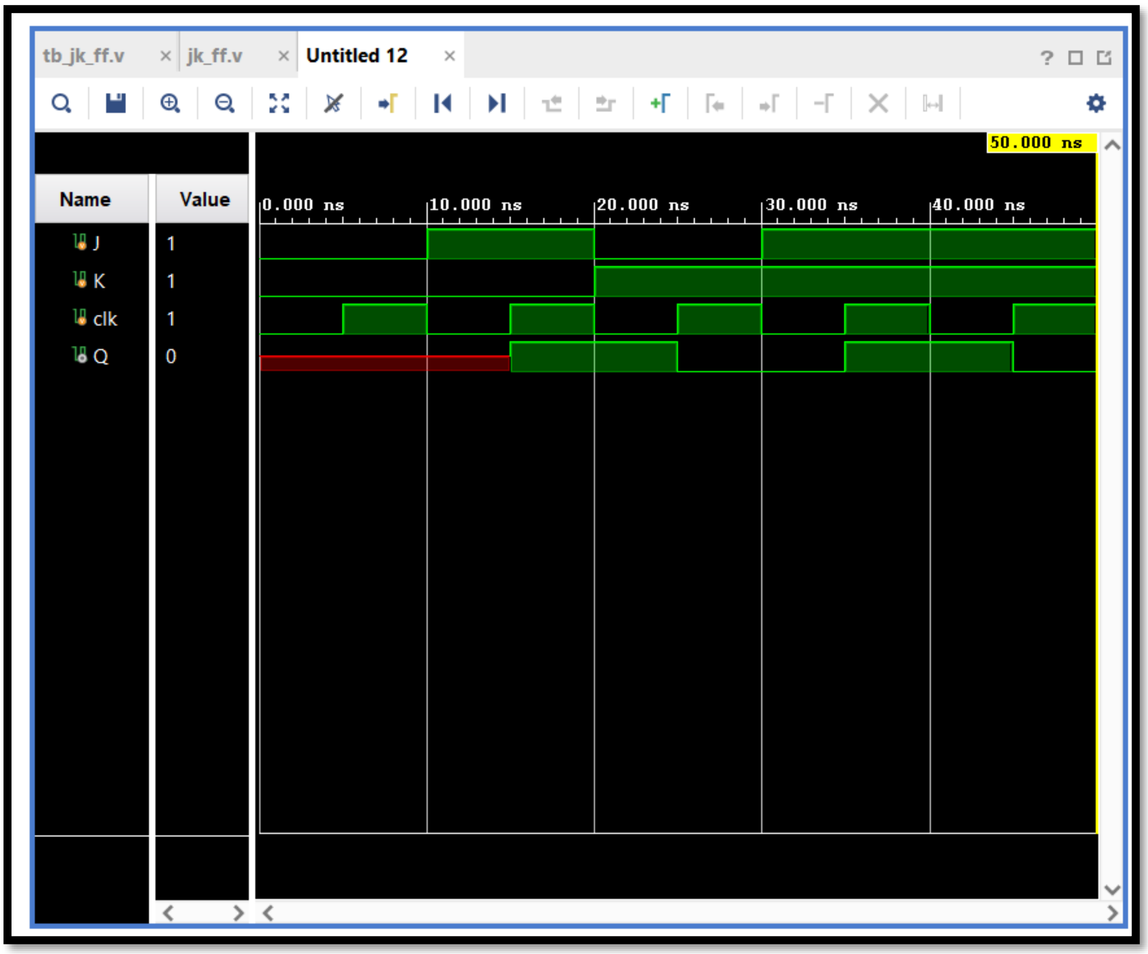
Source Code:

```
`timescale 1ns / 1ps
module jk_ff(
   input J, K, clk,
   output reg Q
);
always @(posedge clk) begin
   case ({J, K})
      2'b00: Q <= Q;
      2'b01: Q <= 0;
      2'b10: Q <= 1;
      2'b11: Q <= ~Q;
   endcase
end
endmodule
```

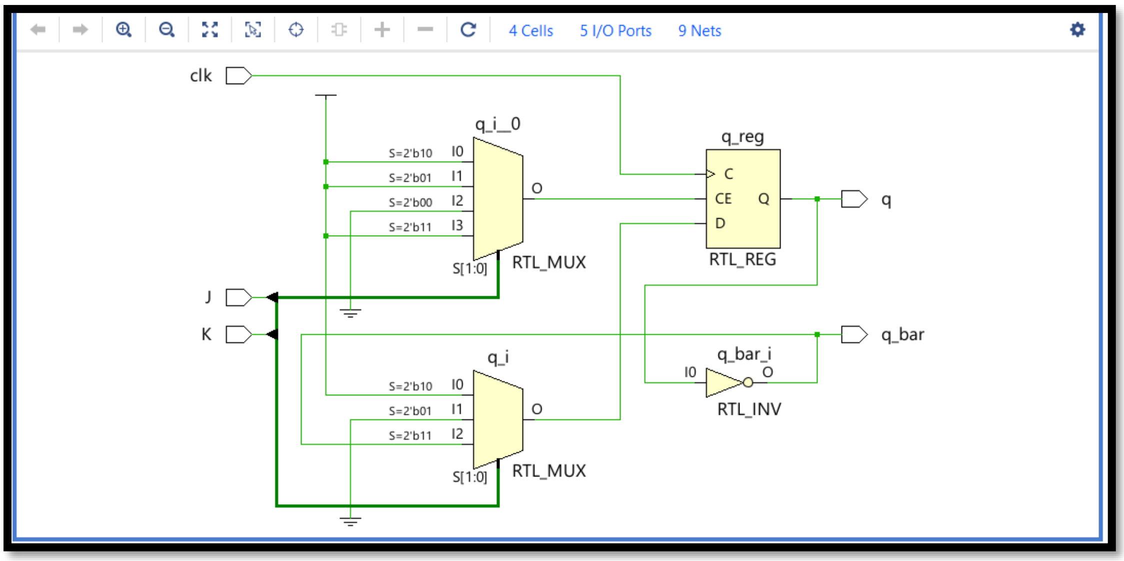
Testbench Code:

```
`timescale 1ns / 1ps
module tb_jk_ff;


reg J, K, clk;
wire Q;


jk_ff dut(J, K, clk, Q);


initial clk=0;
always #5 clk = ~clk;


initial begin
   J=0; K=0; #10;
   J=1; K=0; #10;
   J=0; K=1; #10;
   J=1; K=1; #20;
   $finish;
end


endmodule
```

Simulation:



RTL Synthesis:



14) D Flip Flop

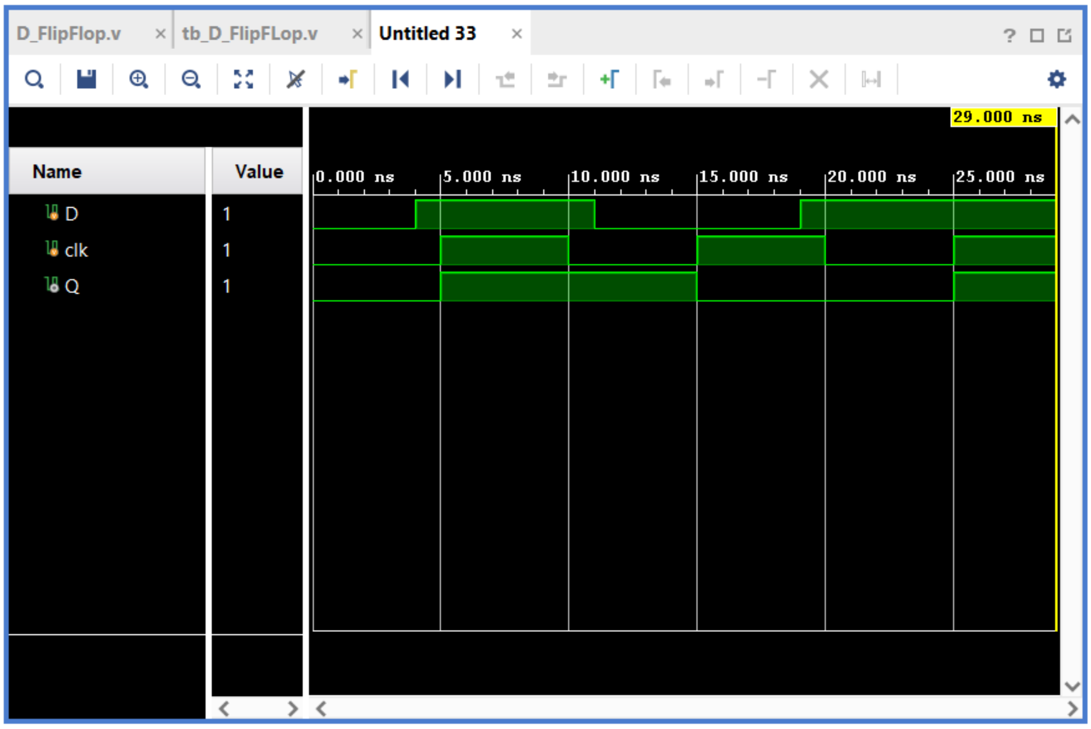
Source Code:

```
`timescale 1ns / 1ps
module d_ff(
    input D, clk,
    output reg Q
);
always @(posedge clk)
    Q <= D;
endmodule
```

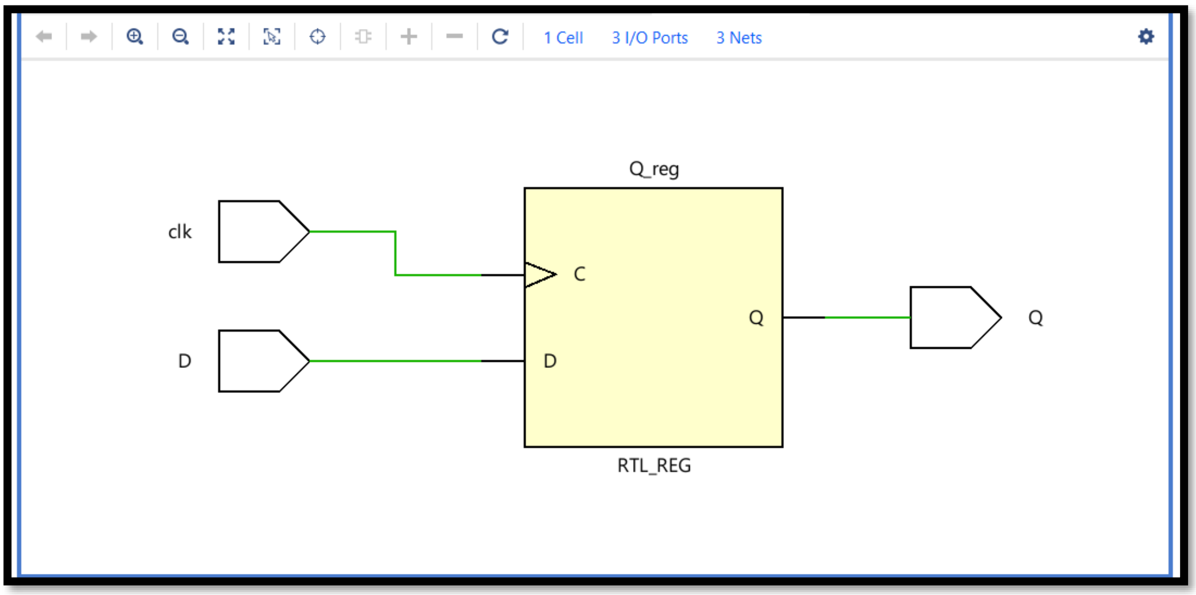
Testbench Code:

```
`timescale 1ns / 1ps
module tb_d_ff;

reg D, clk;
wire Q;

d_ff dut(D, clk, Q);

initial clk = 0;
always #5 clk = ~clk;

initial begin
    D=0; #10;
    D=1; #10;
    D=0; #10;
    D=1; #10;
    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



15) T Flip Flop

Source Code:

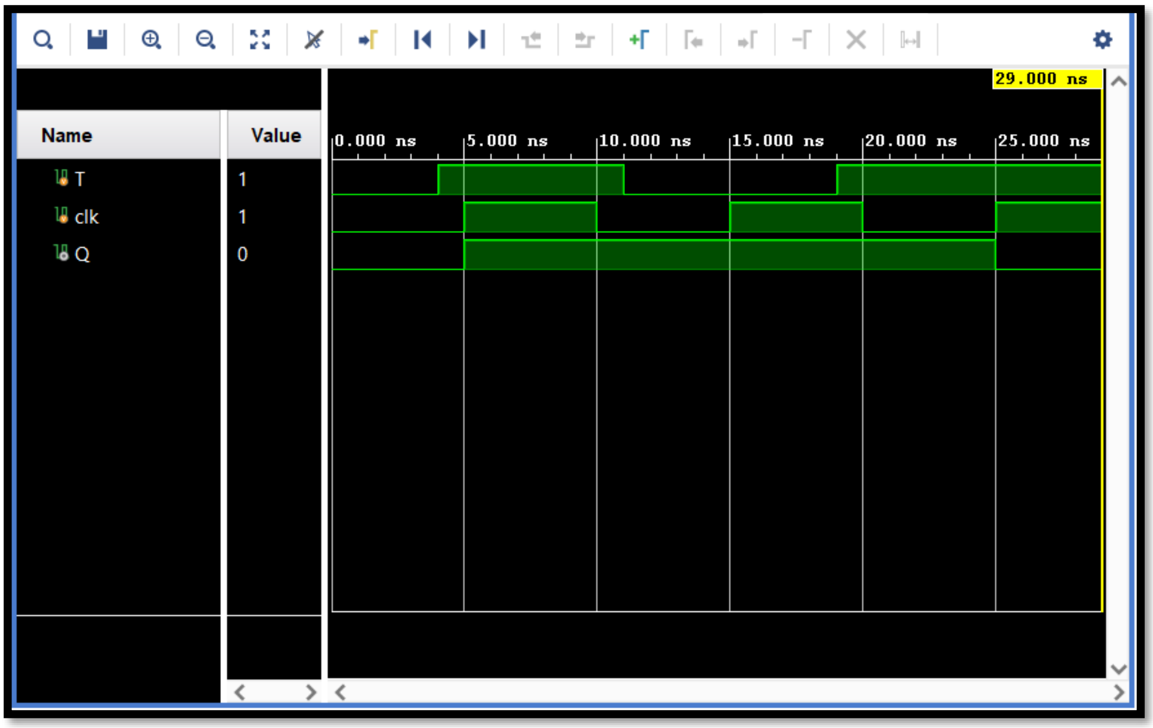
```
`timescale 1ns / 1ps
module t_ff(
   input T, clk,
   output reg Q
);
always @(posedge clk) begin
   if (T) Q <= ~Q;
```

```
    else   Q <= Q;
end
endmodule
```

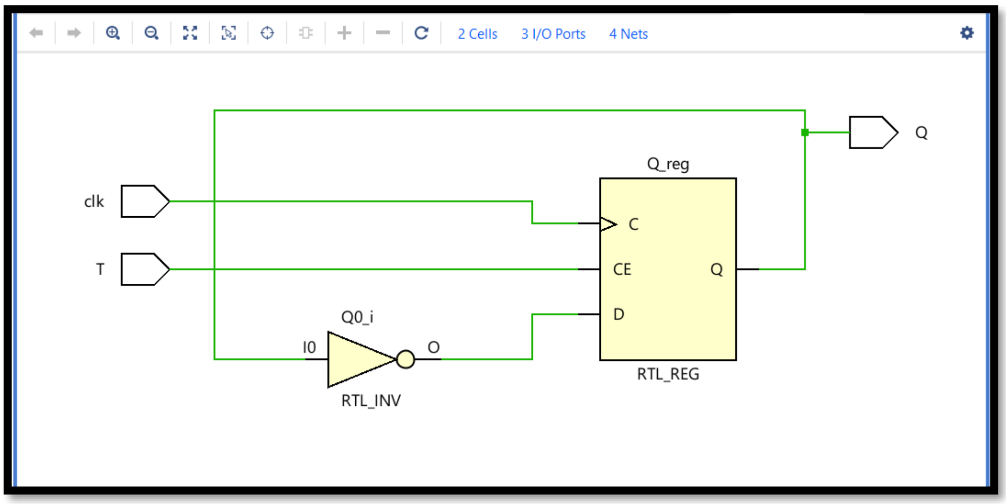
Testbench Code:

```
`timescale 1ns / 1ps
module tb_t_ff;

reg T, clk;
wire Q;

t_ff dut(T, clk, Q);

initial clk = 0;
always #5 clk = ~clk;

initial begin
    T=0; #10;
    T=1; #10;
    $finish;
end

endmodule
```

Simulation:



RTL Synthesis:



16) Counter Design

Source Code:

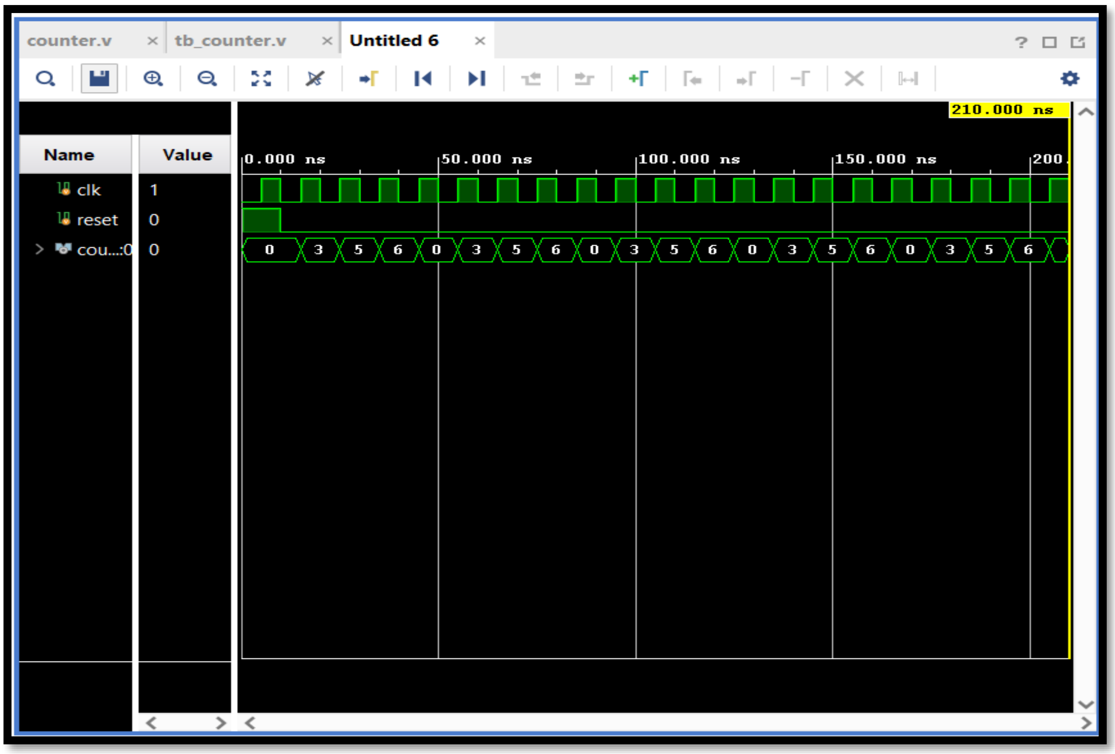
```
`timescale 1ns / 1ps
module counter(
    input clk,
    input reset,
    output reg [2:0] count
);

always @(posedge clk or posedge reset) begin
```

```
    if (reset)
        count <= 3'b000;                    // Start at 0
    else begin
        case (count)
            3'b000: count <= 3'b011;          // 0 → 3
            3'b011: count <= 3'b101;          // 3 → 5
            3'b101: count <= 3'b110;          // 5 → 6
            3'b110: count <= 3'b000;          // 6 → 0 (loop)

            default: count <= 3'b000;         // Lockout correction
        endcase
    end
end

endmodule
```

Testbench Code:

```
`timescale 1ns / 1ps
module tb_counter;

reg clk, reset;
wire [2:0] count;

counter dut(clk, reset, count);

initial clk = 0;
always #5 clk = ~clk;

initial begin
    reset = 1; #10;
    reset = 0;
    // cycles - taken 200 ns
    #200;


    $finish;
end


endmodule
```

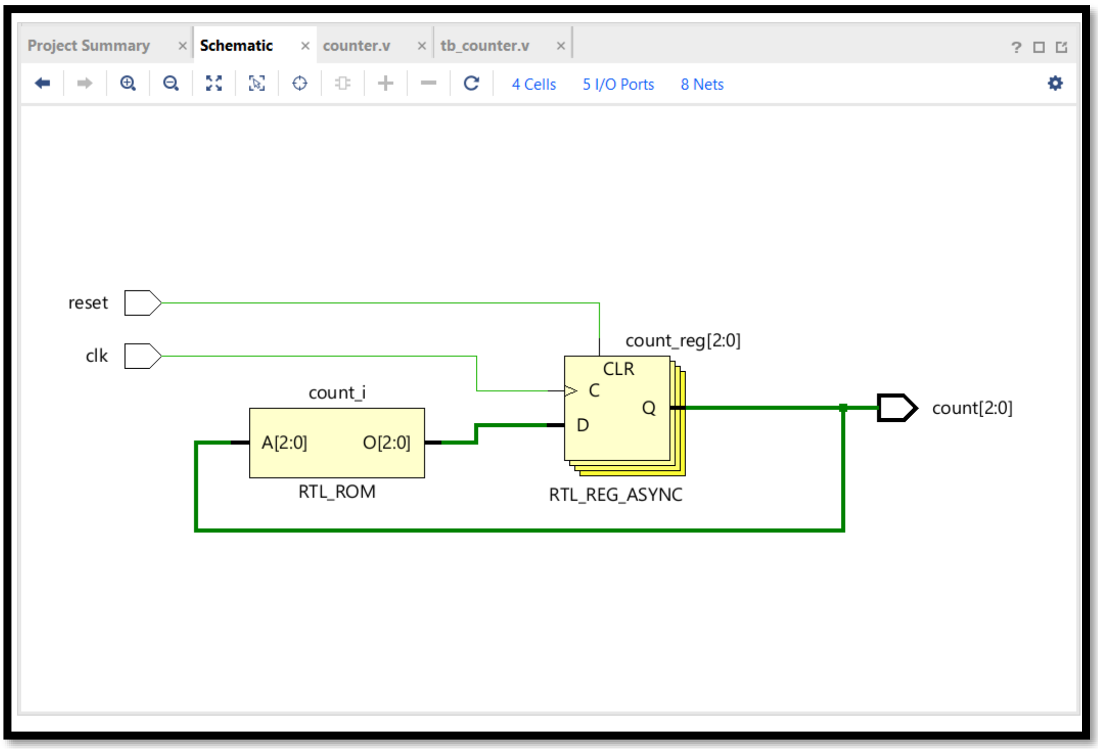
Simulation:



clk 1
reset 0
cou...:0 0

0 3 5 6 0 3 5 6 0 3 5 6 0 3 5 6 0 3 5 6

RTL Synthesis:



Project Summary
Schematic
counter.v
tb_counter.v
4 Cells
5 I/O Ports
8 Nets
reset
clk
count_reg[2:0]
CLR
C
Q
D
count_i
A[2:0]
O[2:0]
RTL_ROM
RTL_REG_ASYNC
count[2:0]