

Digital System Design Applications

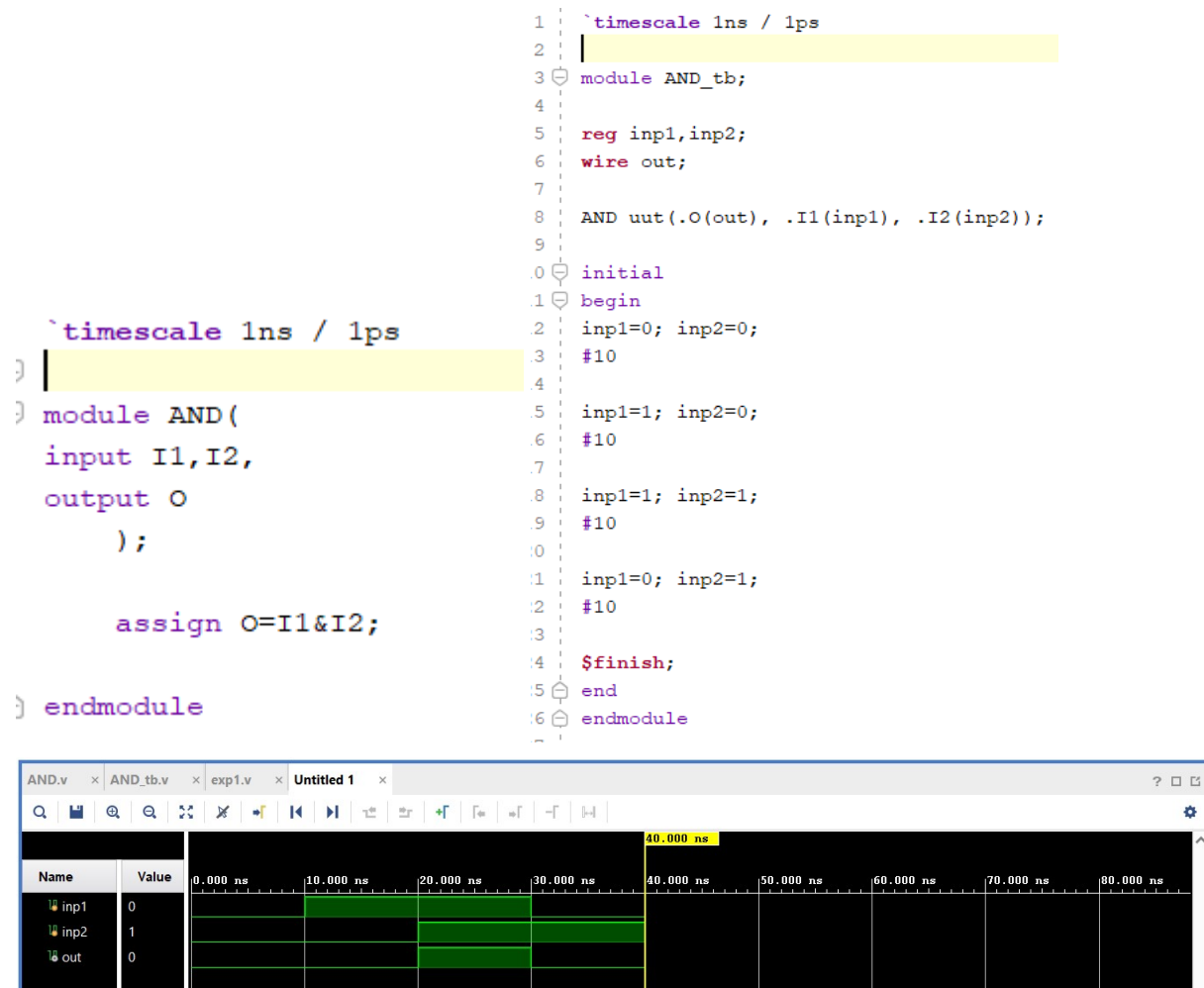
Experiment 1 Report

EHB436E
CRN: 10345

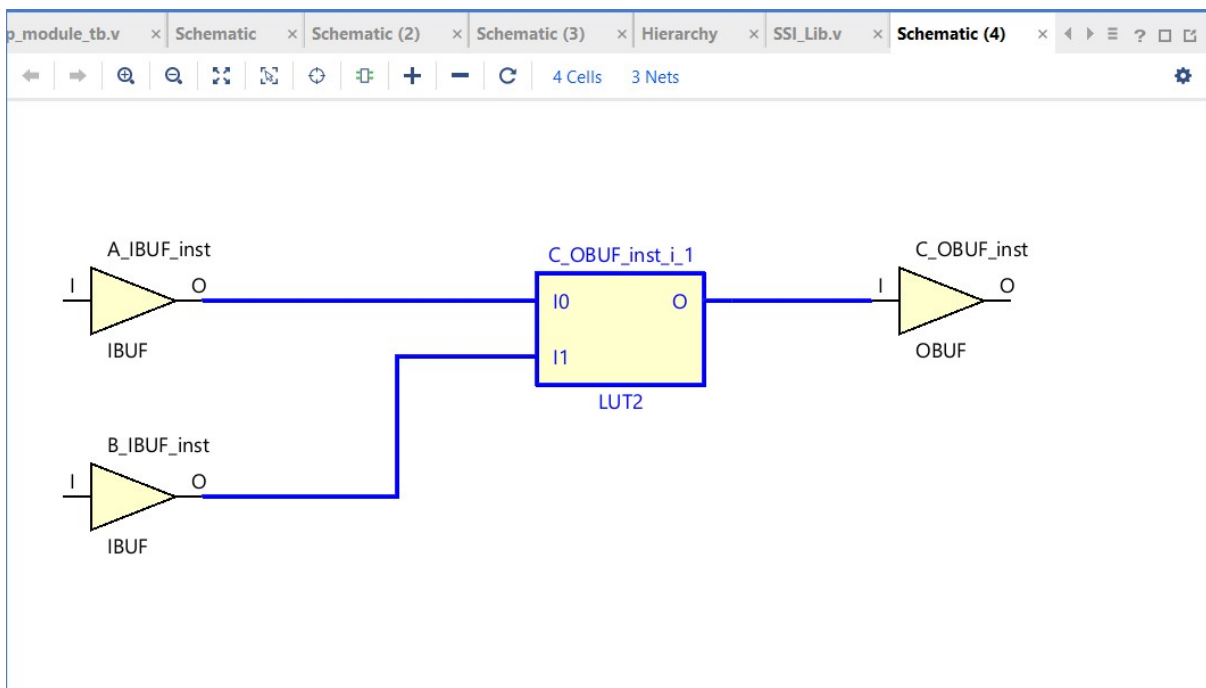
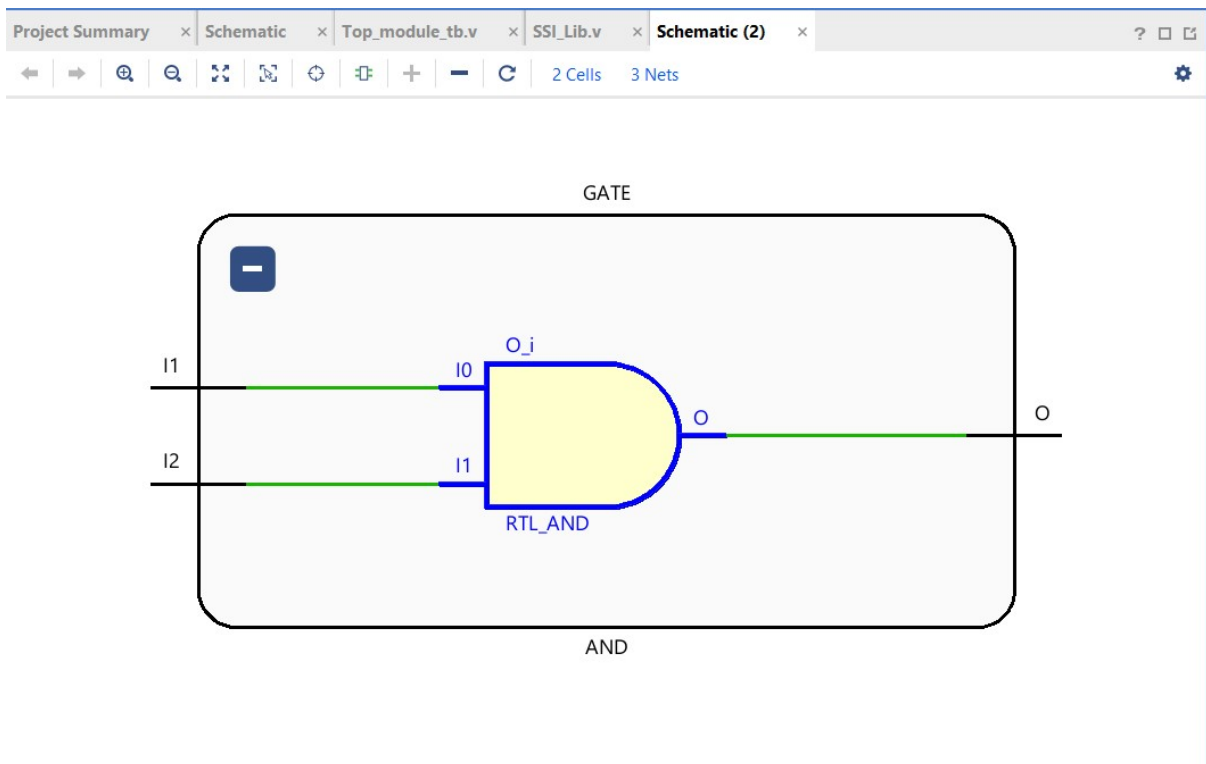
Hatice Nur Andı
040200203

AND GATE

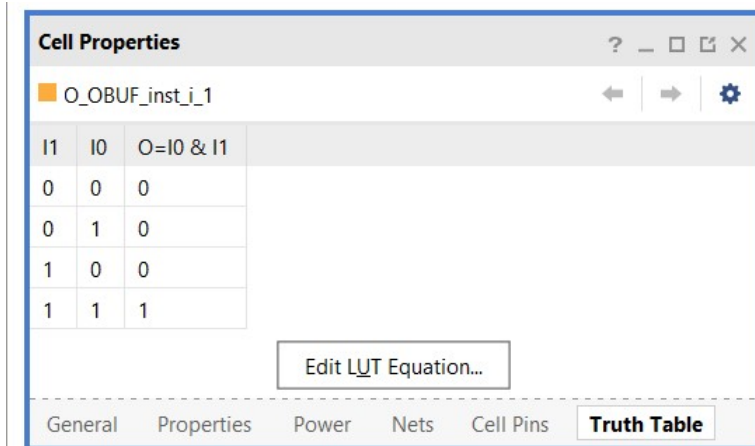
The first part of my report includes the verilog code, testbench code, behavioral simulation waves, RTL schematic, Technology Schematic, truth table, utilization summaries, combinational delays before and after implementation one by one.



Verilog code, tesbench code and simulation wave



RTL and Technology schematic



Cell Properties

O_OBUF_inst_i_1

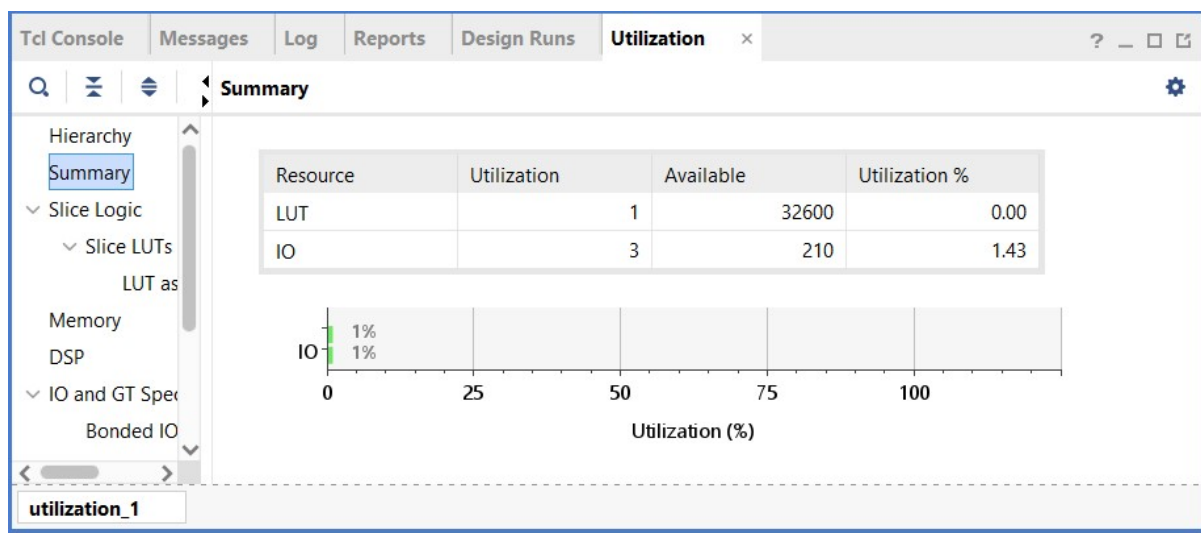
I1	I0	O=I0 & I1
0	0	0
0	1	0
1	0	0
1	1	1

Edit LUT Equation...

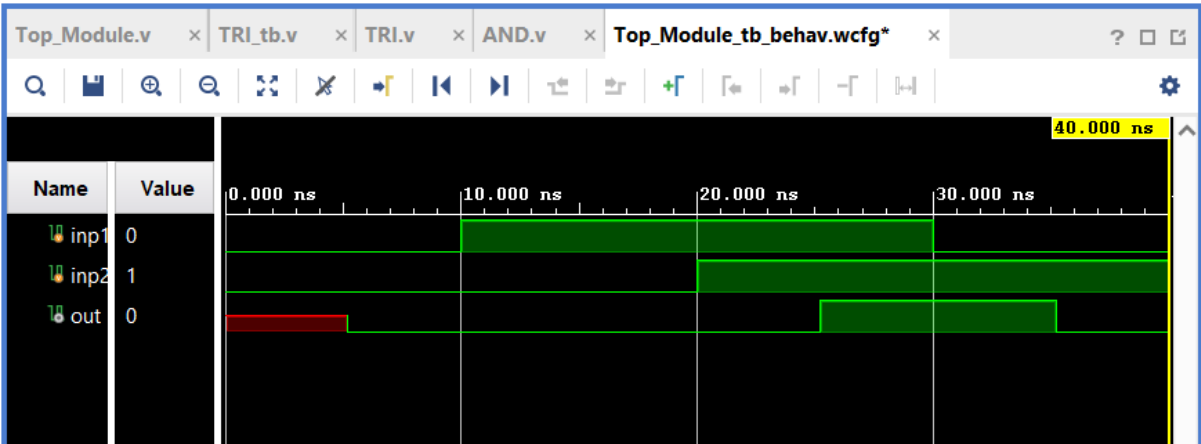
General Properties Power Nets Cell Pins **Truth Table**

Truth Table

We can understand that RTL schematic and technology schematic have the same functionality from the truth table of LUT2. Truth table of LUT2 has the same functionality with the RTL_AND in the RTL schematic.



-Utilization Summary

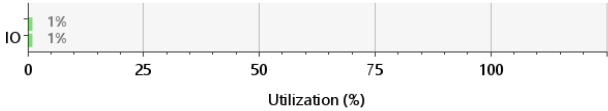


From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
I1	O	5.333	SLOW	2.074	FAST
I2	O	5.333	SLOW	2.074	FAST

-Timing Report

Summary

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	3	210	1.43



Name	Slice LUTs (32600)	Slice (8150)	LUT as Logic (32600)	Bonded IOB (210)
AND	1	1	1	3

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
I1	O	6.378	SLOW	2.105	FAST
I2	O	6.470	SLOW	2.125	FAST

-Implementation Report for AND GATE

The maximum and minimum path delays increased. After the implementation, the physical placement of my LUTs on the board, their proximity to one another, and other factors result in delays.

The screenshots of codes, testbench codes, behavioral simulation wave, RTL schematic and technology schematic screenshots of gates and modules are represented after this part

OR GATE

```

1  `timescale 1ns / 1ps
2  |
3  module OR(
4  input I1,I2,
5  output O
6  );
7
8  assign O=I1||I2;
9
10 endmodule
11
4  module OR_tb;
5
6  reg inp1,inp2;
7  wire out;
8
9  OR uut(.O(out), .I1(inp1), .I2(inp2));
10
11 initial
12 begin
13
14     inp1=0; inp2=0;
15     #10
16
17     inp1=1; inp2=0;
18     #10
19
20     inp1=1; inp2=1;
21     #10
22
23     inp1=0; inp2=1;
24     #10
25
26     $finish;
27
28 end
29 endmodule
30

```

The screenshot shows the Verilog code editor with the OR gate module and its testbench. The code defines an OR gate module 'OR' and a testbench module 'OR_tb' that initializes inputs and checks the output over time.

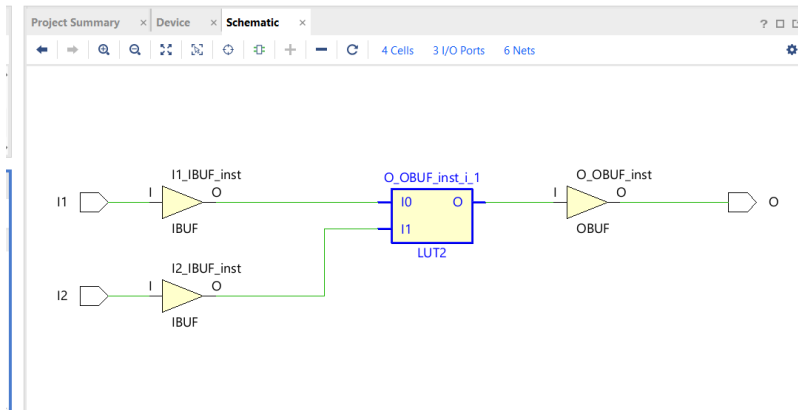
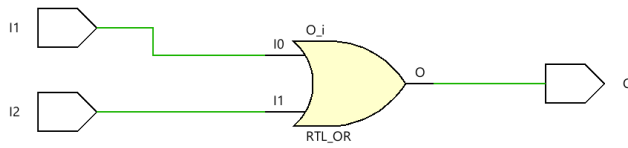
Name	Design...	Block T...
OR_tb	Verilog M	
OR	Verilog M	
glbl	glbl	Verilog M

Name	Value	Data
inp1	0	Logic
inp2	1	Logic
out	1	Logic

The screenshot shows the simulation wave for the OR gate. The wave displays the timing of the inputs and the output over a 40.000 ns period.

Name	Value	0.000 ns	10.000 ns	20.000 ns	30.000 ns	40.000 ns
inp1	0	0	1	1	1	1
inp2	1	1	1	1	1	1
out	1	1	1	1	1	1

Verilog Code, Testbench Code and Simulation Wave



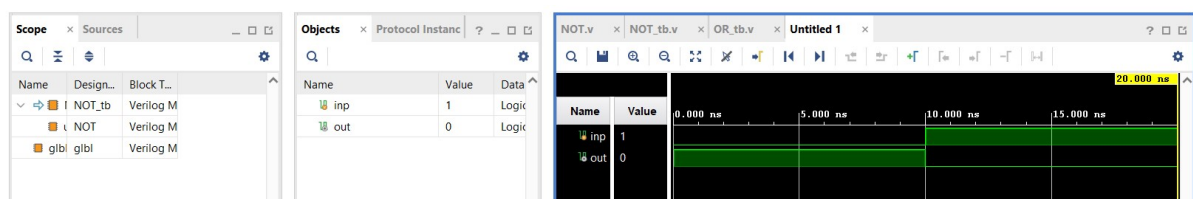
RTL and Technology Schematic

NOT GATE

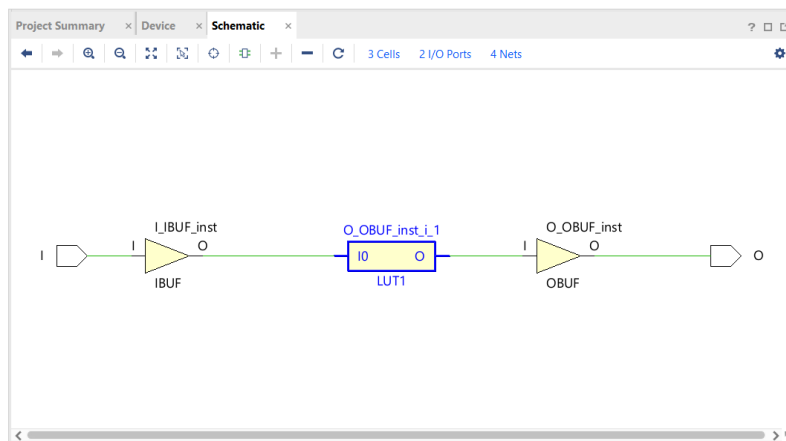
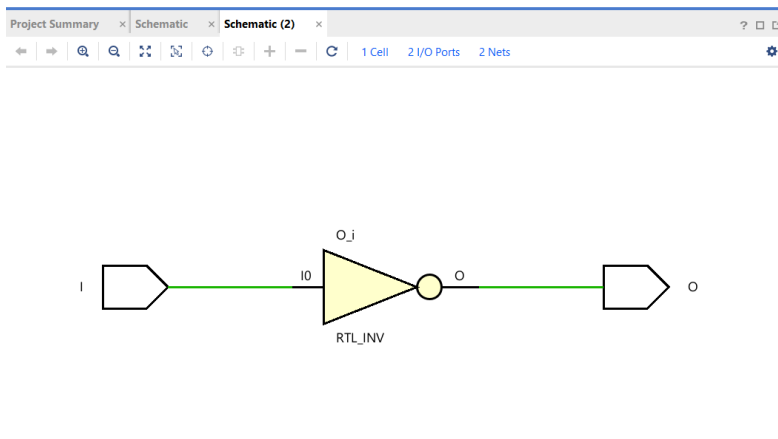
```

1  `timescale 1ns / 1ps
2
3  module NOT(
4
5      input I,
6      output O
7
8      );
9
10     assign O=~I;
11 endmodule
12
13 `timescale 1ns / 1ps
14 module NOT_tb;
15     reg inp;
16     wire out;
17
18     NOT uut(.I(inp), .O(out));
19
20     initial
21     begin
22         inp=0;
23         #10
24         inp=1;
25         #10
26         $finish;
27     end
28 endmodule

```



Verilog Code, Testbench Code and Simulation Wave



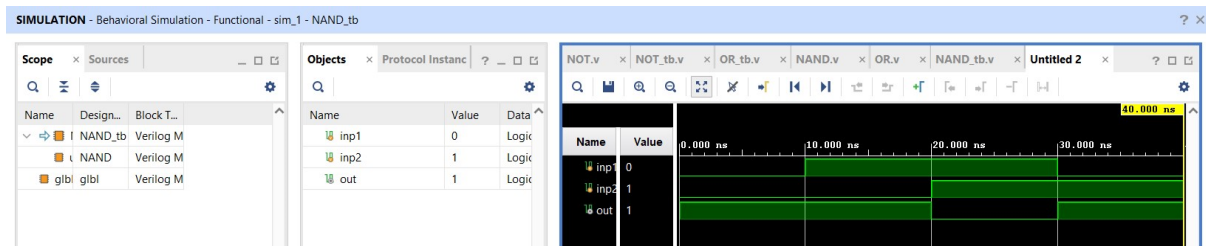
RTL and Technology Schematic

NAND GATE

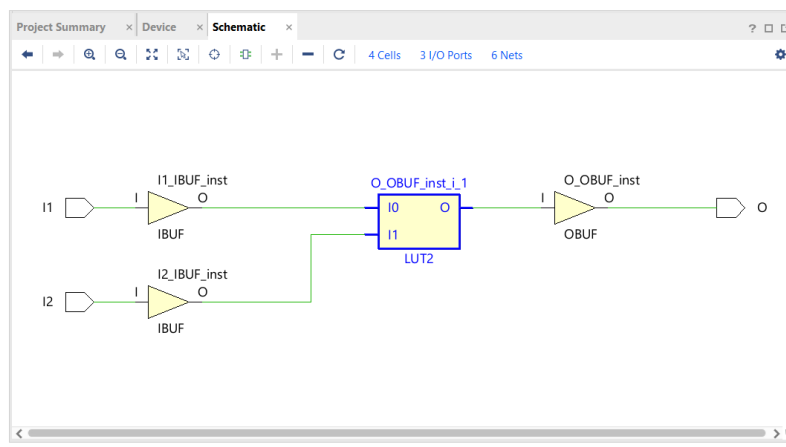
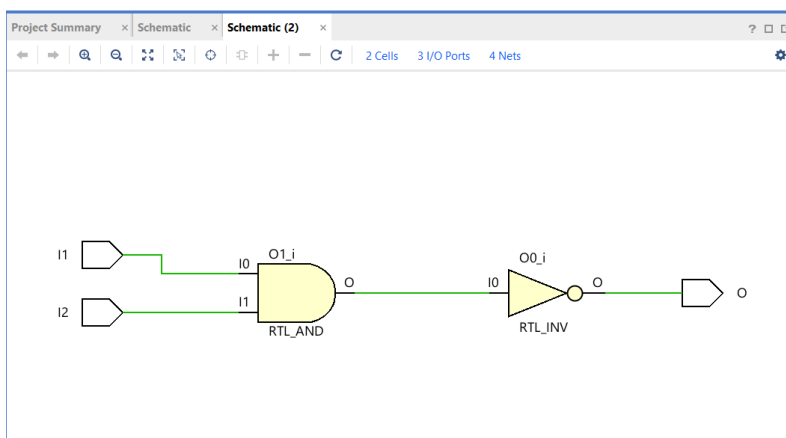
```

1  `timescale 1ns / 1ps
2
3  module NAND(
4      input I1,I2,
5      output reg O
6  );
7
8  always @ (I1 or I2)
9  begin
10     O =~(I1&I2);
11 end
12 endmodule
13
14 `timescale 1ns / 1ps
15
16 module NAND_tb;
17
18     reg inp1,inp2;
19     wire out;
20
21     NAND uut(.O(out), .I1(inp1), .I2(inp2));
22
23     initial
24     begin
25
26         inp1=0; inp2=0;
27         #10
28
29         inp1=1; inp2=0;
30         #10
31
32         inp1=1; inp2=1;
33         #10
34
35         inp1=0; inp2=1;
36         #10
37         $finish;
38     end
39 endmodule

```

Verilog Code, Testbench Code and Simulation Wave



RTL and Technology Schematic

NOR GATE

The image displays the Verilog code for a NOR gate and its testbench, along with the corresponding simulation waveforms.

Verilog Code (Left):

```

`timescale 1ns / 1ps
module NOR(
    input I1,I2,
    output reg O
);

always @ (I1 or I2)
begin
    O =~(I1||I2);
end
endmodule

```

Testbench Code (Right):

```

`timescale 1ns / 1ps
module NOR_tb;

    reg inp1,inp2;
    wire out;

    NOR uut(.O(out), .I1(inp1), .I2(inp2));

    initial
    begin
        inp1=0; inp2=0;
        #10
        inp1=1; inp2=0;
        #10
        inp1=1; inp2=1;
        #10
        inp1=0; inp2=1;
        #10
        $finish;
    end
endmodule

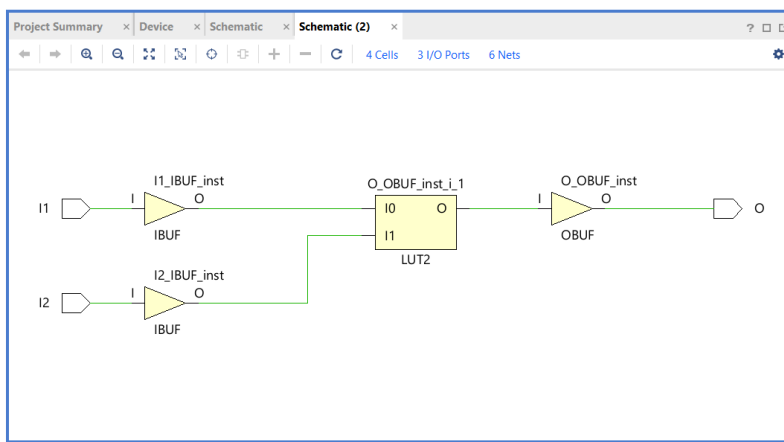
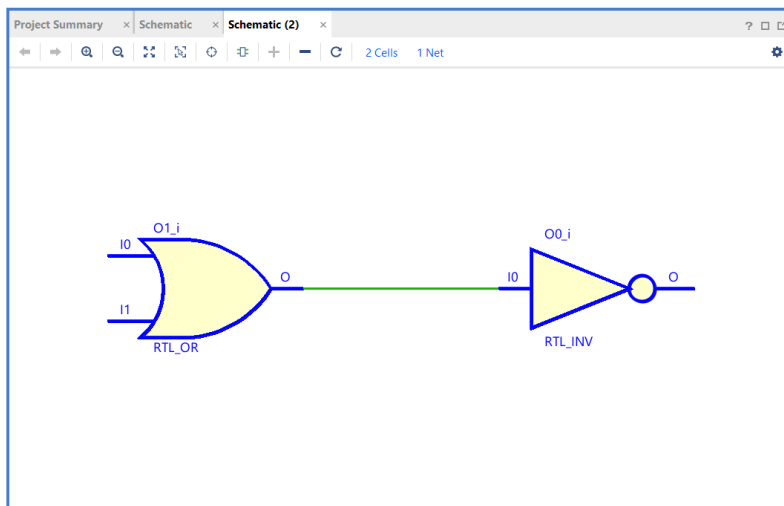
```

Simulation Waveforms (Bottom Right):

The waveforms show the signals `inp1`, `inp2`, and `out` over time. The time scale is 10.000 ns. The signals are as follows:

Signal	0.000 ns	10.000 ns	20.000 ns	30.000 ns
inp1	0	1	1	0
inp2	0	0	1	1
out	0	0	0	0

Verilog Code, Testbench Code and Simulation Wave



RTL and Technology Schematic

XOR

The image displays the Verilog code for an XOR gate and its testbench, along with the simulation results in a waveform viewer.

Verilog Code (Left):

```

`timescale 1ns / 1ps
module XOR(
  input I1,I2,
  output O);

  LUT2# (.INIT(4'b0110)) lut(.O(O),.I0(I1),.I1(I2));

endmodule

```

Testbench Code (Right):

```

`timescale 1ns / 1ps
module XOR_tb;

  reg inp1,inp2;
  wire out;

  XOR uut(.O(out), .I1(inp1), .I2(inp2));

  initial
  begin
    inp1=0; inp2=0;
    #10
    inp1=1; inp2=0;
    #10
    inp1=1; inp2=1;
    #10
    inp1=0; inp2=1;
    #10
    $finish;
  end
endmodule

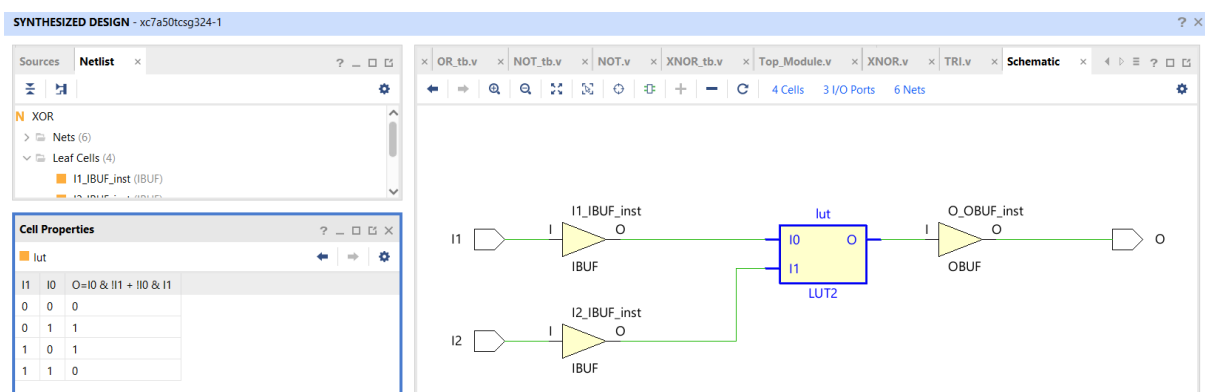
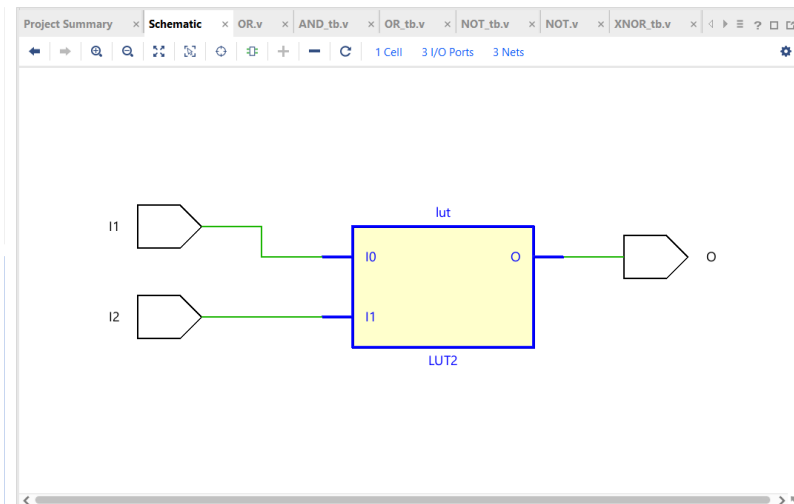
```

Simulation Waveform (Bottom):

The waveform viewer shows the signals `inp1`, `inp2`, and `out` over time. The signals are sampled at 10 ns intervals.

Time (ns)	inp1	inp2	out
0.000	0	0	0
10.000	1	0	1
20.000	1	1	0
30.000	0	1	1

Verilog Code, Testbench Code and Simulation Wave



RTL and Technology Schematic

XNOR GATE

```

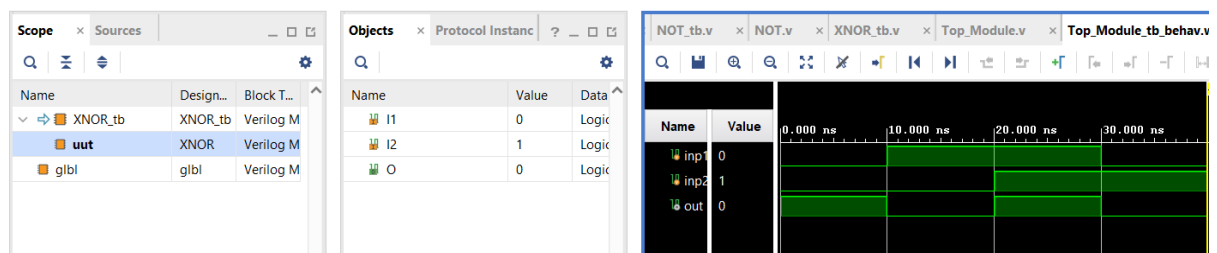
`timescale 1ns / 1ps

module XNOR(
    input I1,I2,
    output O);

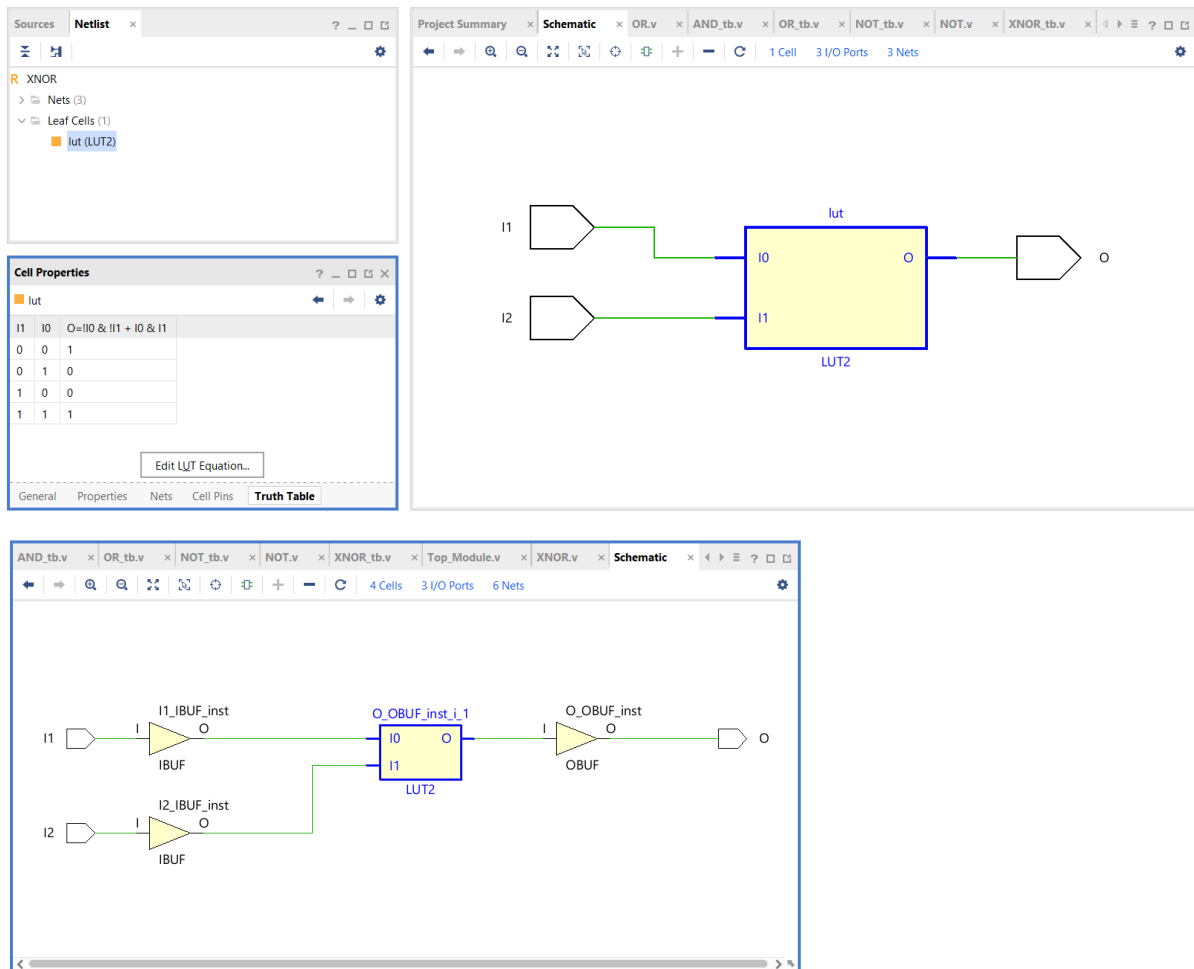
    LUT2# (.INIT(4'b1001)) lut(.O(O),.I0(I1),.I1(I2));
endmodule

1 `timescale 1ns / 1ps
2 module XNOR_tb;
3
4     reg inp1,inp2;
5     wire out;
6
7     XNOR uut(.O(out), .I1(inp1), .I2(inp2));
8
9     initial
10    begin
11        inp1=0; inp2=0;
12        #10
13
14        inp1=1; inp2=0;
15        #10
16
17        inp1=1; inp2=1;
18        #10
19
20        inp1=0; inp2=1;
21        #10
22
23        $finish;
24    end
25 endmodule

```



Verilog Code, Testbench Code and Simulation Wave



RTL and Technology Schematic

TRI GATE

The image displays the Verilog code for a TRI gate and its testbench, along with the simulation waveforms.

Verilog Code (Left):

```

`timescale 1ns / 1ps
module TRI(
  input I,E,
  output O
);
  assign O = E ? I : 1'bZ;
endmodule

```

Testbench Code (Right):

```

`timescale 1ns / 1ps
module TRI_tb;
  reg i,e;
  wire out;

  TRI uut(.O(out), .I(i), .E(e));

  initial
  begin
    i=0; e=0;
    #10
    i=1; e=0;
    #10
    i=1; e=1;
    #10
    i=0; e=1;
    #10
    $finish;
  end
endmodule

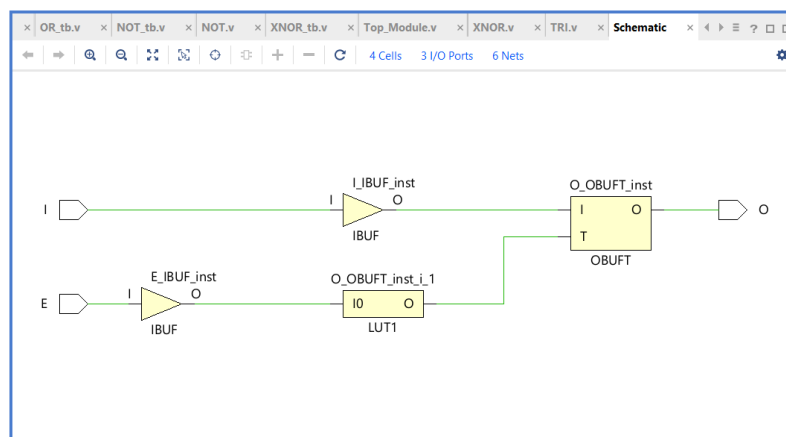
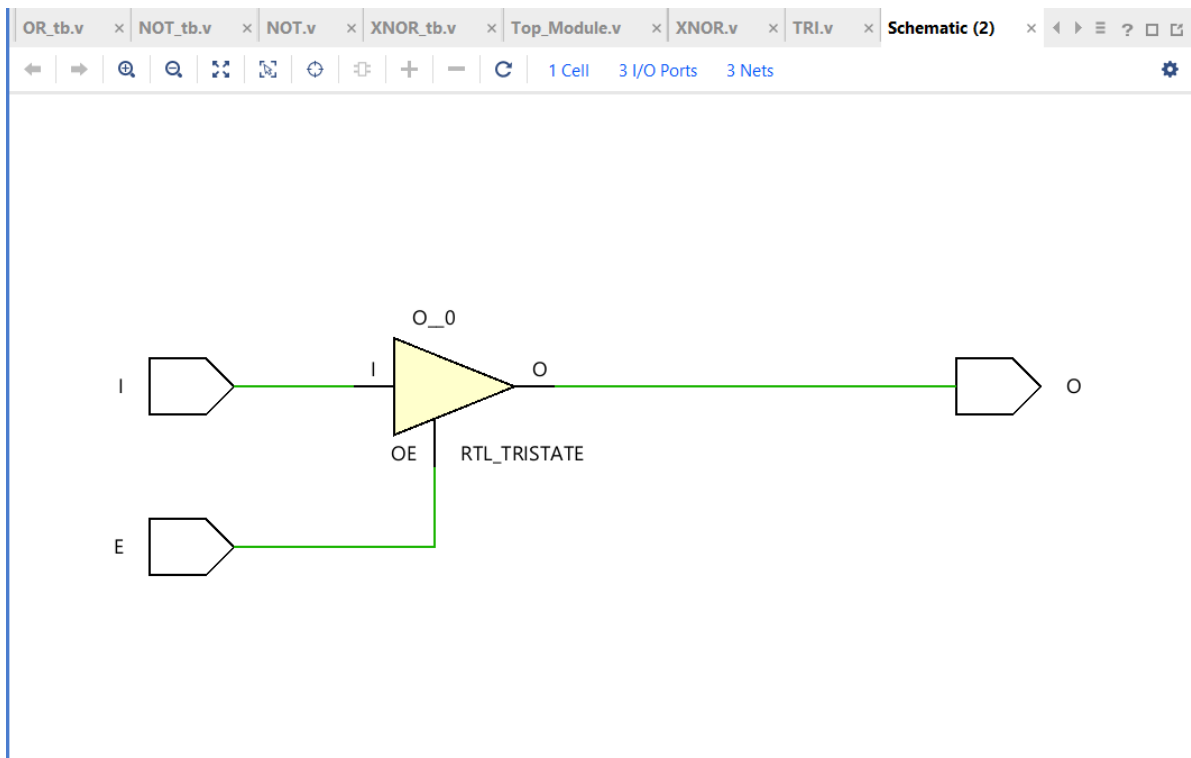
```

Simulation Waveforms (Bottom Right):

The waveforms show the signals `i`, `e`, and `out` over time. The signal `out` is high (1) when both `i` and `e` are high (1), and low (0) otherwise. The signal `out` is high-impedance (Z) when both `i` and `e` are low (0).

Time (ns)	i	e	out
0.000 - 10.000	0	0	0
10.000 - 20.000	1	0	0
20.000 - 30.000	1	1	1
30.000 - 40.000	0	1	0
40.000 - 50.000	0	0	0

Verilog Code, Testbench Code and Simulation Wave



RTL and Technology Schematic

TOP_MODULE

```

module Top_Module(
    input [14:0] IN,
    output [7:0] OUT

    );

    AND aGATE(.O(OUT[0]), .I1(IN[0]), .I2(IN[1]));

    OR oGATE(.O(OUT[1]), .I1(IN[2]), .I2(IN[3]));

    NOT nGATE(.O(OUT[2]), .I(IN[4]));

    NAND naGATE(.O(OUT[3]), .I1(IN[5]), .I2(IN[6]));

    NOR noGATE(.O(OUT[4]), .I1(IN[7]), .I2(IN[8]));

    XOR xGATE(.O(OUT[5]), .I1(IN[9]), .I2(IN[10]));

    XNOR xnGATE(.O(OUT[6]), .I1(IN[11]), .I2(IN[12]));

    TRI tGATE(.O(OUT[7]), .I(IN[13]), .E(IN[14]));

endmodule

```

```

module Top_Module_tb;
reg [14:0] inp;
wire [7:0] out;

Top_Module uut(.OUT(out), .IN(inp));

initial
begin

inp=16'b0000000000000000;
#10

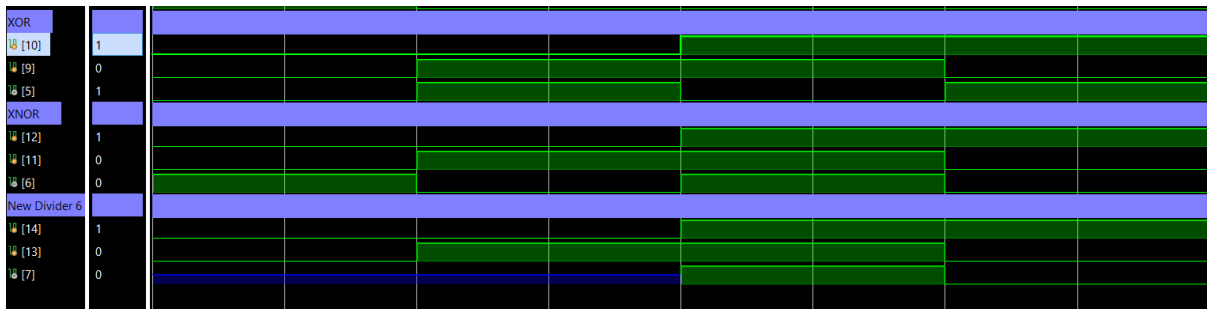
inp=16'b010101010100101;
#10

inp=16'b1111111111111111;
#10

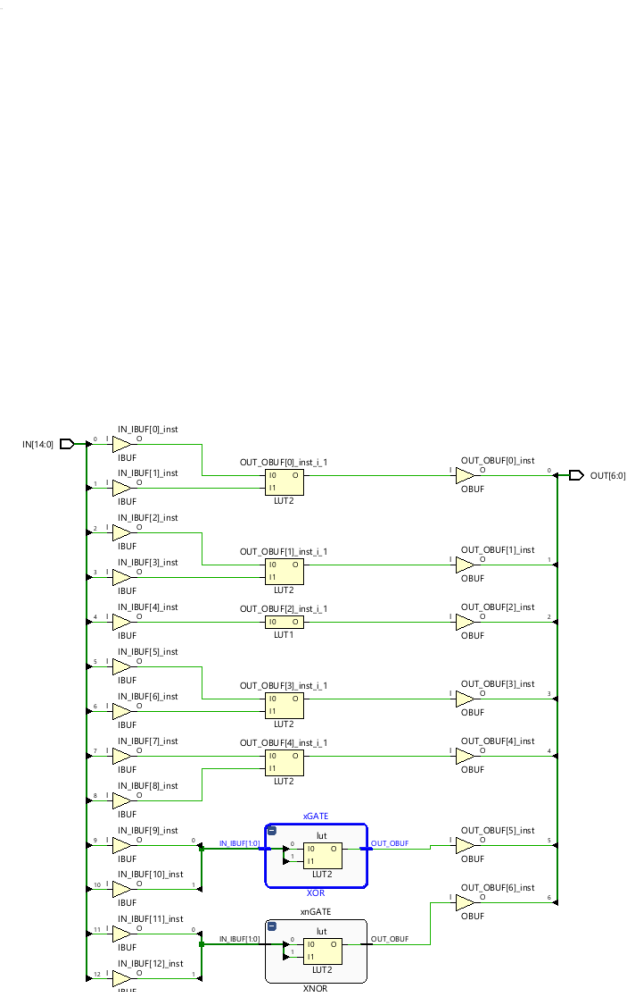
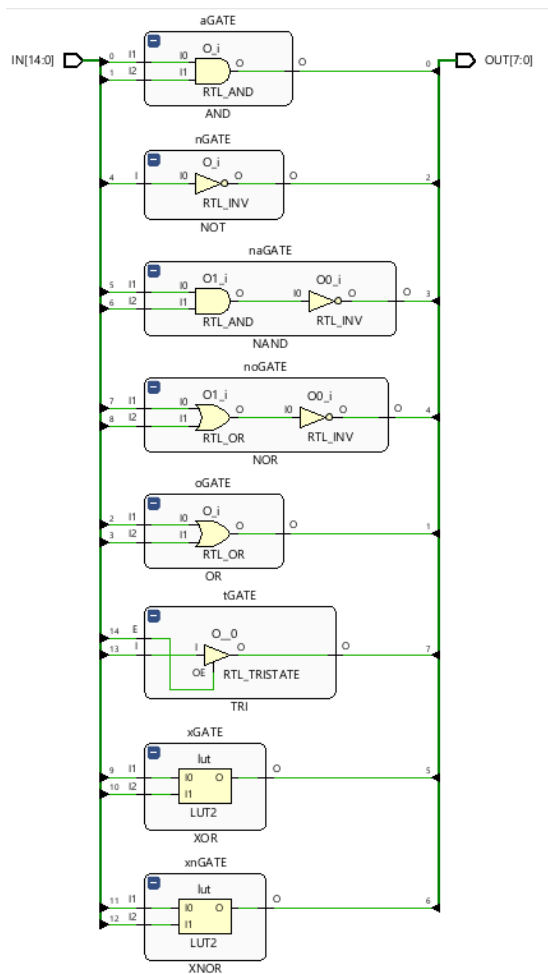
inp=16'b101010101011010;
#10
$finish;
end

```

AND									
[1]	1								
[0]	0								
[0]	0								
OR									
[3]	1								
[2]	0								
[1]	1								
NOT									
[4]	1								
[2]	0								
NAND									
[6]	1								
[5]	0								
[3]	1								
NOR									
[8]	1								
[7]	0								
[4]	0								
XOR									
[10]	1								
[9]	0								
[5]	1								



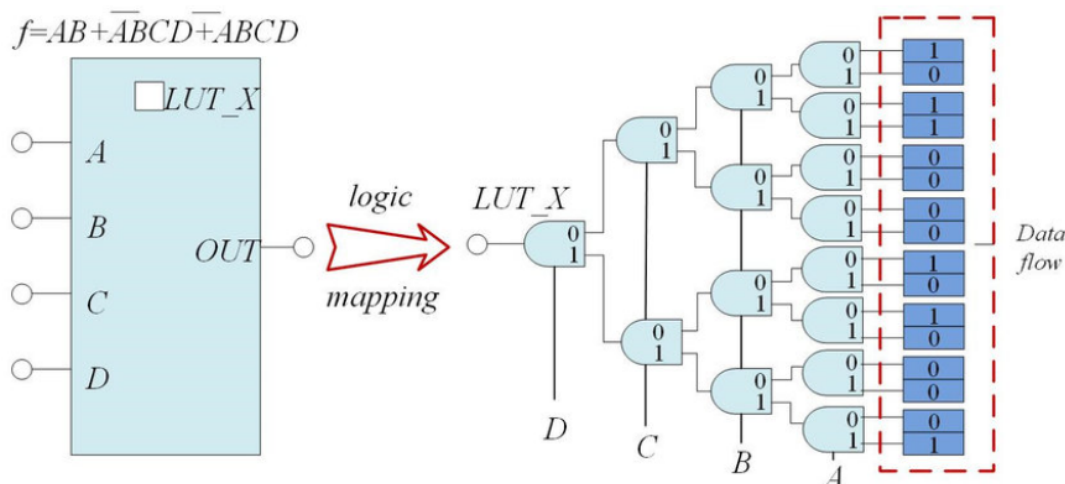
Verilog Code, Testbench Code and Simulation Wave



RTL and Technology Schematic

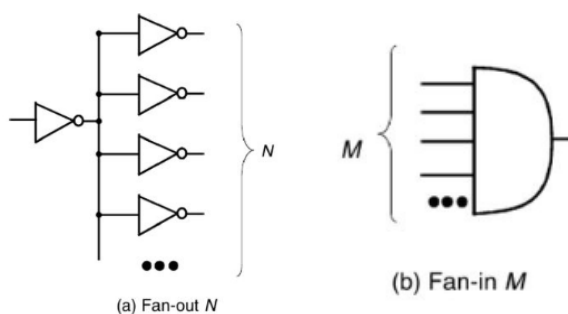
LOOK UP TABLE

A Look-Up Table is a configurable memory element in an FPGA that stores output values for all possible combinations of input signals. LUTs are implemented as arrays of memory cells, where each cell corresponds to a unique input combination. These cells store precomputed output values, enabling rapid signal processing and logic implementation. When an FPGA is programmed, the LUTs are configured to implement specific logic functions. Input signals are used as addresses to access the corresponding output values stored in the LUT. This process allows FPGAs to perform complex logic operations, making them highly adaptable for various applications. LUTs can implement any combinational logic function, making them essential for tasks like arithmetic operations, data manipulation, and conditional branching.



FAN-OUT FAN-IN

The fan-in is defined as the maximum number of inputs that a logic gate can accept. If the number of input exceeds, the output will be undefined or incorrect. It is specified by the manufacturer and is provided in the data sheet. The fan-out is defined as the maximum number of inputs (load) that can be connected to the output of a gate without degrading the normal operation. Fan Out is calculated from the amount of current available in the output of a gate and the amount of current needed in each input of the connecting gate. It is specified by the manufacturer and is provided in the data sheet. Exceeding the specified maximum load may cause a malfunction because the circuit will not be able to supply the demanded power.



SETUP and HOLD TIME DELAY

As a signal travels down a wire, it can change from a 0->1 or 1->0. An input to a Flip-Flop needs to be stable (not changing) in order for an FPGA design to work properly. The input must be stable for some small amount of time prior to being sampled by the clock. This amount of time is called setup time. Setup time is the amount of time required for the input to a Flip-Flop to be stable before a clock edge. Hold time is similar to setup time, but it deals with events after a clock edge occurs. Hold time is the minimum amount of time required for the input to a Flip-Flop to be stable after a clock edge.

