

EHB436E

DIGITAL SYSTEM DESIGN APPLICATIONS

Muhammed Erkmen

040170049

EXPERIMENT-1 REPORT

AND GATE

This gate created to implement logical “AND” operation. Modules of all gates are coded in SSI_Library.v file. Source code of and gate:

```
//AND GATE
module and_gate(
    input I1,
    input I2,
    output O
);
assign O = I1&I2;
endmodule
```

I1 and I2 represents the inputs, O represents the output of the gate.

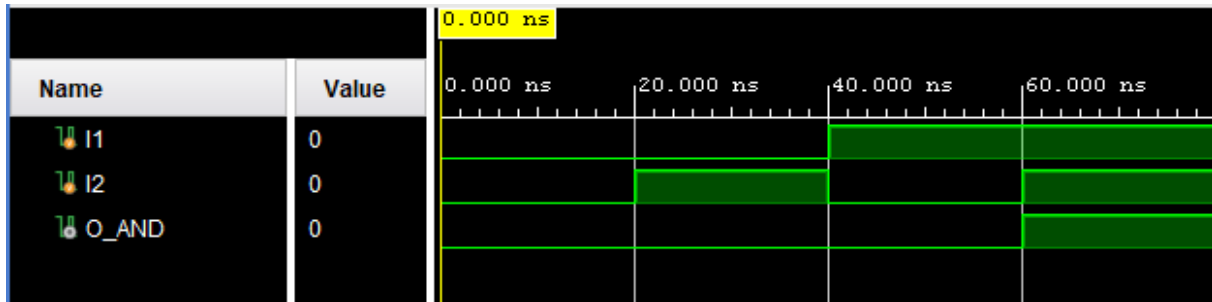
Testbench code of and gate:

```
module SSI_Library_tb;
reg I1;
reg I2;
wire O_AND;

and_gate AND_GATE(
    .I1(I1),
    .I2(I2),
    .O(O_AND));

initial begin
I1=0;
I2=0;
#20;
I1=0;
I2=1;
#20;
I1=1;
I2=0;
#20;
I1=1;
I2=1;
#20;
end
endmodule
```

Behaviour of the gate in simulation:



As expected the output O_AND is 0 in cases:

$I1 = 0 \ \& \ I2 = 0$,

$I1 = 1 \ \& \ I2 = 0$,

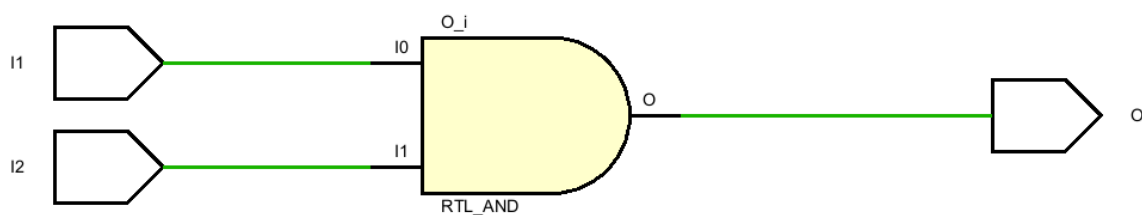
$I1 = 0 \ \& \ I2 = 1$

and the output O_AND is 1 in case:

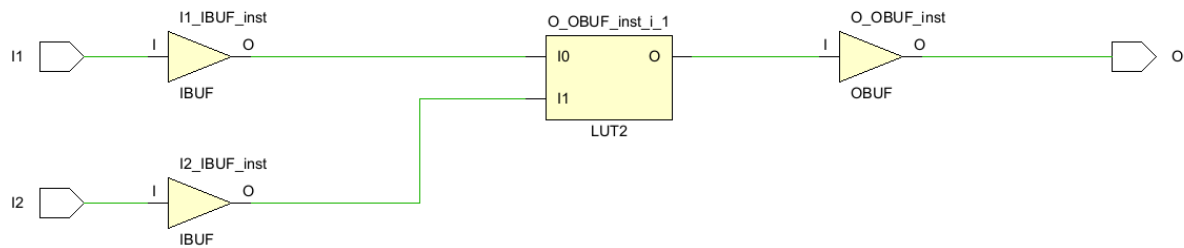
$I1=1 \ \& \ I2 =1$.

Delay is 0 at behavioral simulation.

RTL Schematic of and_gate:



Synthesized technology schematic of and_gate:



Truth table of and_gate:

Cell Properties		
O_OBUF_inst_i_1		
I1	I0	O=I0 & I1
0	0	0
0	1	0
1	0	0
1	1	1

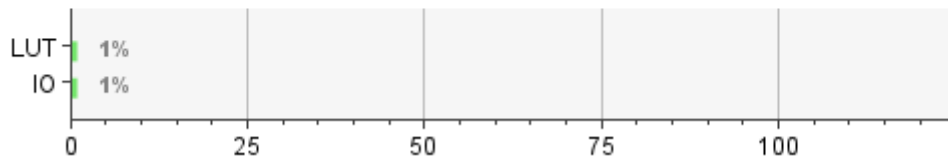
Edit LUT Equation...

Ver Nets Cell Pins **Truth Table**

We can verify that synthesized schematic and RTL schematic works same as their truth tables are same. In another way to explain, LUT block of the technology schematic has a truth table that is the same as AND operation truth table.

Utilization summary:

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



Timing Report:

Setup Time Delay:

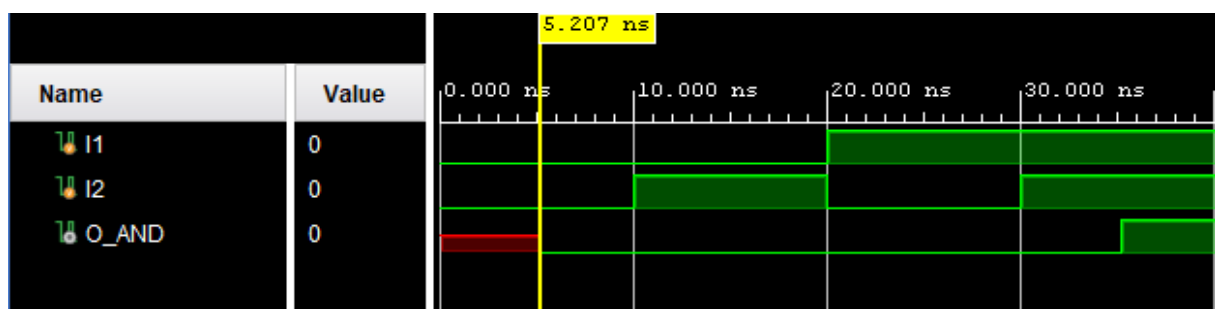
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	3	4	1	I2	O	5.333	3.733	1.599

Hold delay:

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 2	∞	3	4	1	I2	O	2.074	1.400	0.674

Timing report summary shows that maximum delay is setup time delay.

Post Synthesis Timing Simulation



Post Synthesis Simulation File:

```

`timescale 1 ps / 1 ps

`define XIL_TIMING

(* NotValidForBitStream *)
module and_gate
    (I1,
     I2,
     O);
    input I1;
    input I2;
    output O;

    wire I1;
    wire I1_IBUF;
    wire I2;
    wire I2_IBUF;
    wire O;
    wire O_OBUF;

initial begin
    $sdf_annotate("and_gate_tb_time_synth.sdf",,,, "tool_control");
end
    IBUF I1_IBUF_inst
        (.I(I1),
         .O(I1_IBUF));
    IBUF I2_IBUF_inst
        (.I(I2),
         .O(I2_IBUF));
    OBUF O_OBUF_inst
        (.I(O_OBUF),
         .O(O));
    LUT2 #(
        .INIT(4'h8))
        O_OBUF_inst_i_1
        (.I0(I1_IBUF),
         .I1(I2_IBUF),
         .O(O_OBUF));
endmodule

`ifndef GLBL
`define GLBL
`timescale 1 ps / 1 ps

module glbl ();

    parameter ROC_WIDTH = 100000;
    parameter TOC_WIDTH = 0;
    parameter GRES_WIDTH = 10000;
    parameter GRES_START = 10000;

//----- STARTUP Globals -----
    wire GSR;
    wire GTS;
    wire GWE;
    wire PRLD;
    wire GRESTORE;
    tri1 p_up_tmp;
    tri (weak1, strong0) PLL_LOCKG = p_up_tmp;

```

```

wire PROGB_GLBL;
wire CCLKO_GLBL;
wire FCSBO_GLBL;
wire [3:0] DO_GLBL;
wire [3:0] DI_GLBL;

reg GSR_int;
reg GTS_int;
reg PRLD_int;
reg GRESTORE_int;

//----- JTAG Globals -----
wire JTAG_TDO_GLBL;
wire JTAG_TCK_GLBL;
wire JTAG_TDI_GLBL;
wire JTAG_TMS_GLBL;
wire JTAG_TRST_GLBL;

reg JTAG_CAPTURE_GLBL;
reg JTAG_RESET_GLBL;
reg JTAG_SHIFT_GLBL;
reg JTAG_UPDATE_GLBL;
reg JTAG_RUNTEST_GLBL;

reg JTAG_SEL1_GLBL = 0;
reg JTAG_SEL2_GLBL = 0;
reg JTAG_SEL3_GLBL = 0;
reg JTAG_SEL4_GLBL = 0;

reg JTAG_USER_TDO1_GLBL = 1'bz;
reg JTAG_USER_TDO2_GLBL = 1'bz;
reg JTAG_USER_TDO3_GLBL = 1'bz;
reg JTAG_USER_TDO4_GLBL = 1'bz;

assign (strong1, weak0) GSR = GSR_int;
assign (strong1, weak0) GTS = GTS_int;
assign (weak1, weak0) PRLD = PRLD_int;
assign (strong1, weak0) GRESTORE = GRESTORE_int;

initial begin
GSR_int = 1'b1;
PRLD_int = 1'b1;
#(ROC_WIDTH)
GSR_int = 1'b0;
PRLD_int = 1'b0;
end
initial begin
GTS_int = 1'b1;
#(TOC_WIDTH)
GTS_int = 1'b0;
end
initial begin
GRESTORE_int = 1'b0;
#(GRES_START);
GRESTORE_int = 1'b1;
#(GRES_WIDTH);
GRESTORE_int = 1'b0;
end

endmodule
`endif

```

Implementation of AND_GATE with top module:

```
`timescale 1ns / 1ps

module top_module(
input [15:0] IN,
output [7:0] OUT
);

and_gate AND_GATE (.I1(IN[0]),.I2(IN[1]),.O(OUT));

endmodule
```

Setup Delay after implementation:

Name	Slack ^{^1}	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
↳ Path 1	∞	3	2	1	IN[1]	OUT[0]	6.470	3.733	2.737

Hold Delay after implementation:

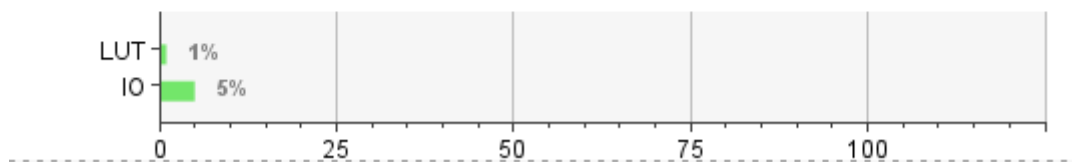
Name	Slack ^{^1}	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
↳ Path 2	∞	3	2	1	IN[0]	OUT[0]	2.105	1.395	0.709

Comparison: Implementation delays are higher than the synthesized model. Because implementation delays are calculated more realistic for in-physical environment.

Utilization Report:

Summary

Resource	Utilization	Available	Utilization %
LUT	1	32600	0.00
IO	10	210	4.76



OR GATE

This gate created to implement logical “OR” operation. Modules of all gates are coded in SSI_Library.v file. Source code of and gate:

```
// OR GATE
module or_gate(
    input I1,
    input I2,
    output O);
assign O = I1|I2;
endmodule
```

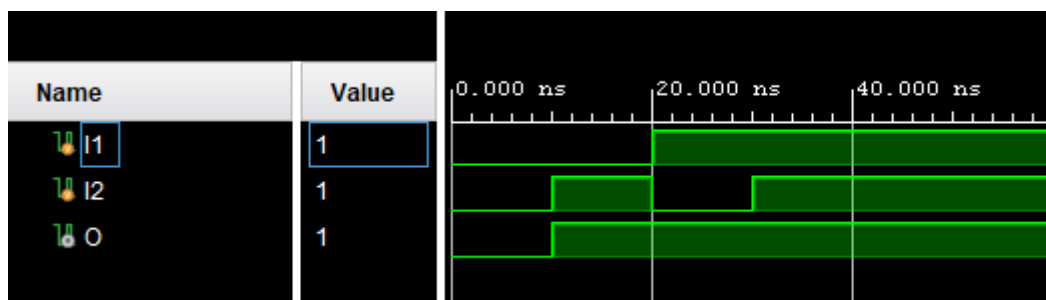
Testbench code:

```
module SSI_Library_tb;
reg I1;
reg I2;
wire O;
or_gate OR_GATE(
    .I1(I1),
    .I2(I2),
    .O(O));

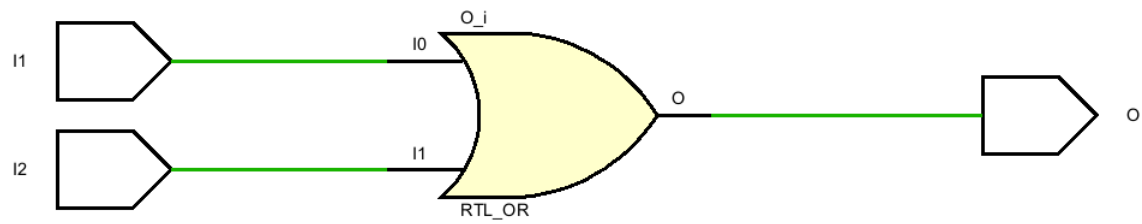
initial begin
I1=0;
I2=0;
#10;
I1=0;
I2=1;
#10;
I1=1;
I2=0;
#10;
I1=1;
I2=1;
#10;
end
```

I1 and I2 represents input and O represents output.

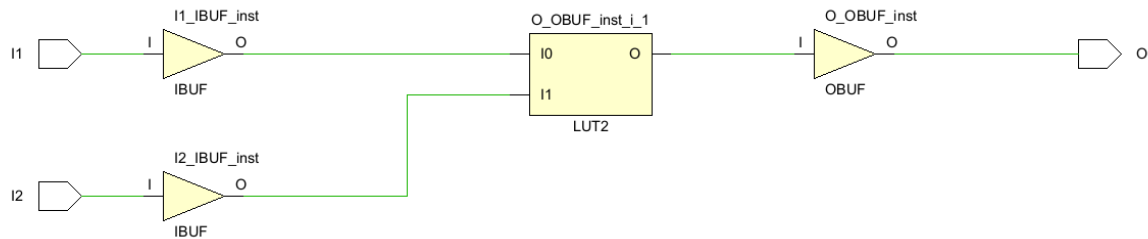
Behavioural Simulation Results of OR gate:



RTL Schematic of OR_gate:



Technology schematic of OR_gate:



NOT GATE

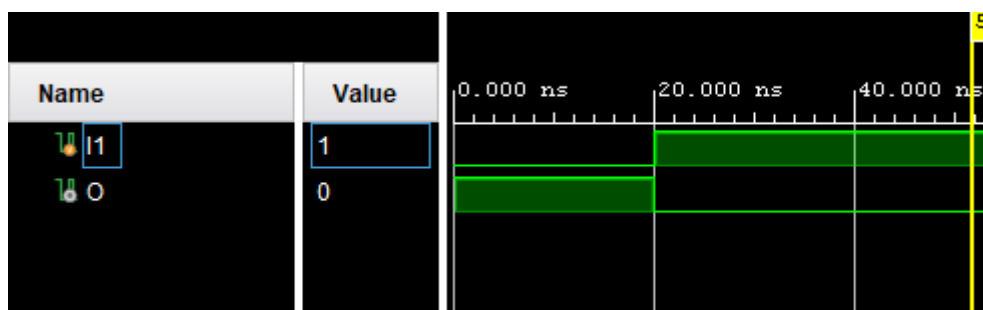
Source codes:

```
//NOT GATE
module not_gate(
    input I,
    output O);
assign O = ~I;
endmodule
```

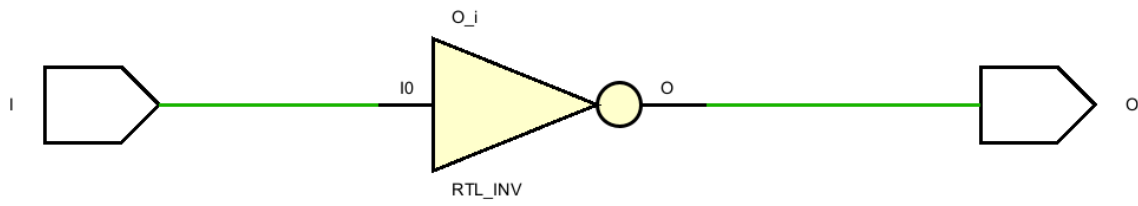
Testbench Codes:

```
module SSI_Library_tb;
reg I1;
reg I2;
wire O;
not_gate NOT_GATE(
    .I(I1),
    .O(O));
initial begin
    I1=0;
    I2=0;
    #10;
    I1=0;
    I2=1;
    #10;
    I1=1;
    I2=0;
    #10;
    I1=1;
    I2=1;
    #10;
end
endmodule
```

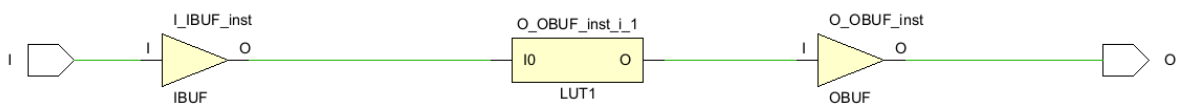
Behavioural simulation results of NOT_gate:



RTL Schematic of NOT_gate:



Technology schematic of NOT_gate:



NAND GATE

This gate is written inside of the always begin end block as wanted.

Source codes:

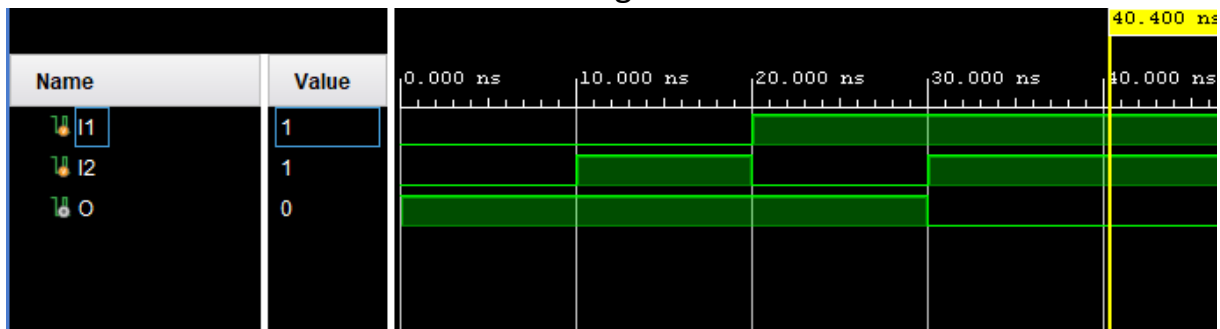
```
//NAND
module nand_gate(
    input I1,
    input I2,
    output reg O

);
always @(I1,I2)
begin
O = ~(I1&I2);
end
endmodule
```

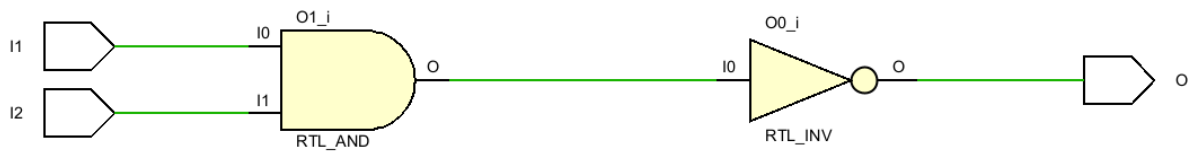
Testbench codes:

```
module SSI_Library_tb;
reg I1;
reg I2;
wire O;
nand_gate NAND_GATE(
    .I1(I1),
    .I2(I2),
    .O(O));
initial begin
I1=0;
I2=0;
#10;
I1=0;
I2=1;
#10;
I1=1;
I2=0;
#10;
I1=1;
I2=1;
#10;
end
endmodule
```

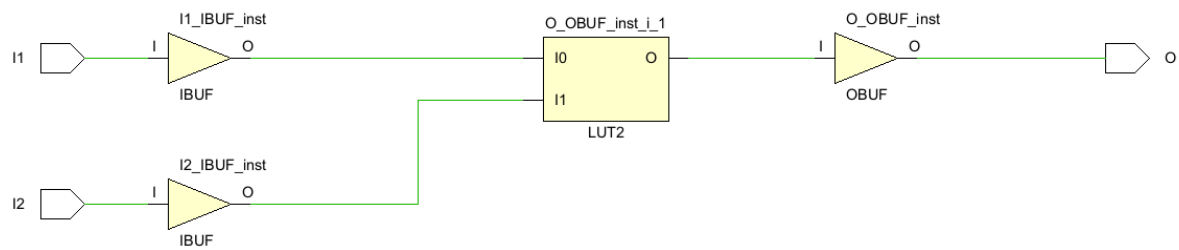
Behavioural simulation results of NAND gate:



RTL schematic of NAND gate:



Technology schematic of NAND gate:



NOR GATE

This NOR gate implemented in always block as same as NAND gate.

Source codes:

```
//NOR
module nor_gate(
    input I1,
    input I2,
    output reg O
);
always @(I1,I2)
begin
O = ~(I1|I2);
end
endmodule
```

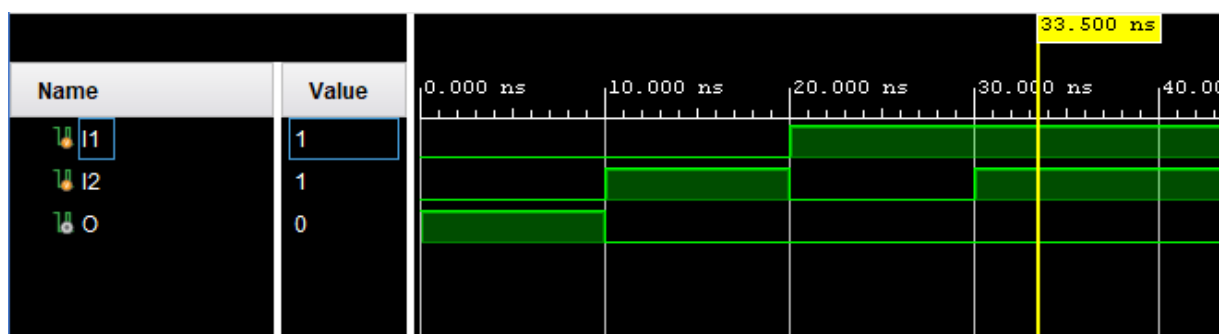
Testbench codes:

```
module SSI_Library_tb;
reg I1;
reg I2;
wire O;

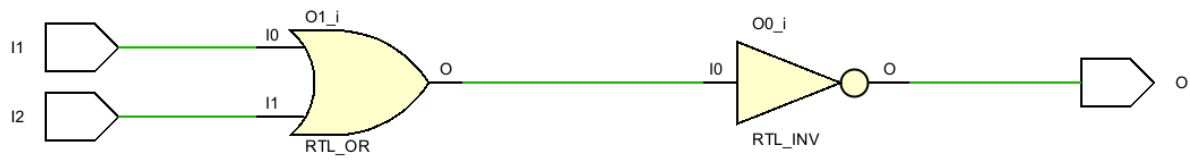
nor_gate NOR_GATE(
    .I1(I1),
    .I2(I2),
    .O(O));

initial begin
I1=0;
I2=0;
#10;
I1=0;
I2=1;
#10;
I1=1;
I2=0;
#10;
I1=1;
I2=1;
#10;
end
endmodule
```

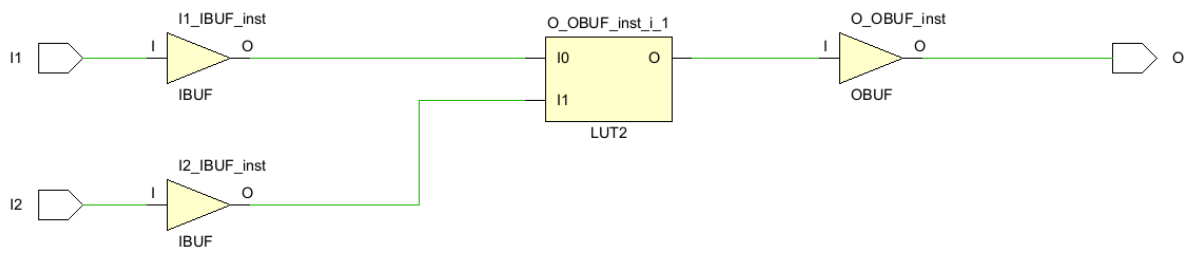
Behavioural simulation results of NOR gate:



RTL schematic of NOR gate:



Technology schematic of NOR gate:



XOR GATE

This XOR gate implemented with using LUT2 primitive.

Source codes:

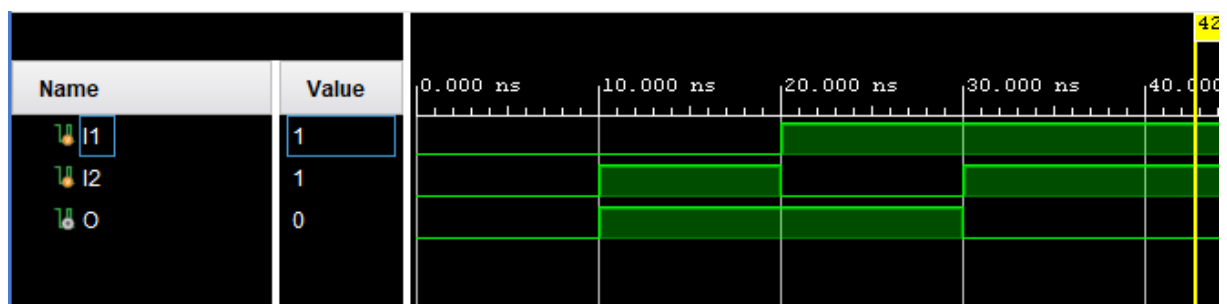
```
//XOR
module xor_gate(
    input I1,
    input I2,
    output O
);
LUT2# (.INIT(4'b0110)) xor_gate(
    .O(O),
    .I0(I1),
    .I1(I2));
endmodule
```

Testbench codes:

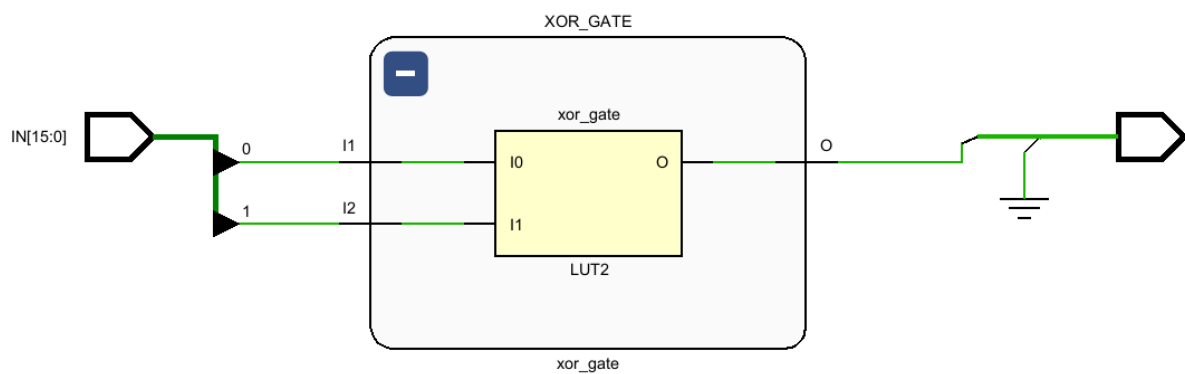
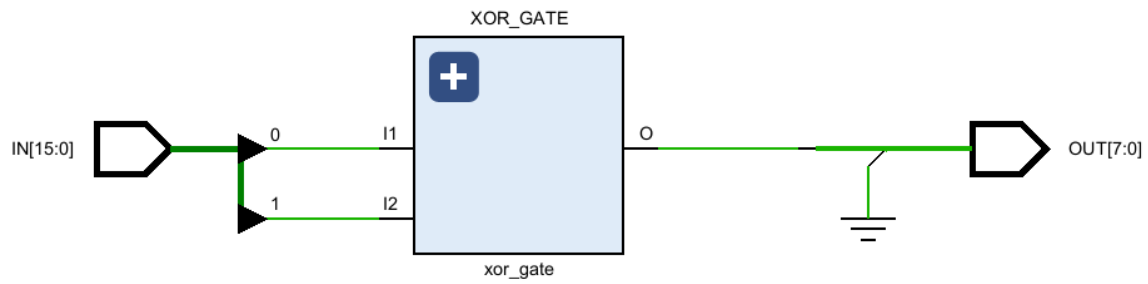
```
module SSI_Library_tb;
reg I1;
reg I2;
wire O;
xor_gate XOR_GATE(
    .I1(I1),
    .I2(I2),
    .O(O));

initial begin
I1=0;
I2=0;
#10;
I1=0;
I2=1;
#10;
I1=1;
I2=0;
#10;
I1=1;
I2=1;
#10;
end
endmodule
```

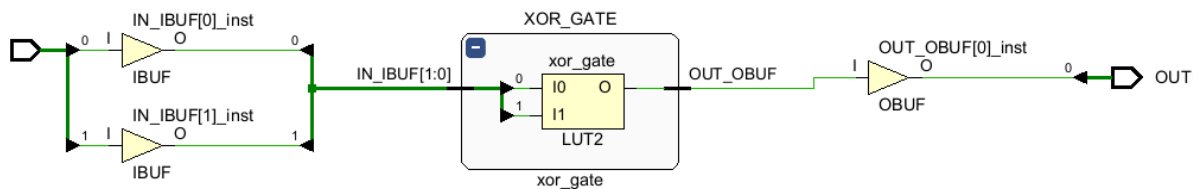
Behavioural simulation results of XOR gate:



RTL Schematic of XOR module:



Technology Schematic of XOR module:



Technology schematic and RTL schematic are same in LUT2 method.

XNOR GATE

This XNOR gate implemented with using LUT2 primitive.

Source codes:

```
//XNOR
module xnor_gate(
    input I1,
    input I2,
    output O
);
LUT2# (.INIT(4'b1001)) xnor_gate(
    .O(O),
    .I0(I1),
    .I1(I2));
endmodule
```

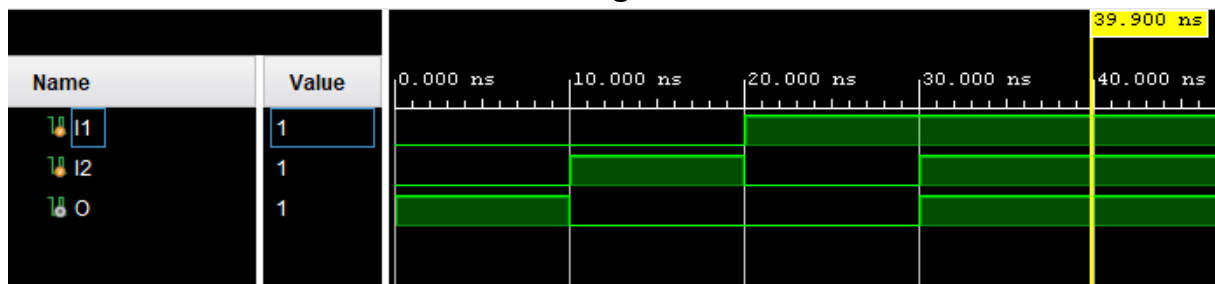
Testbench codes:

```
module SSI_Library_tb;
reg I1;
reg I2;
wire O;

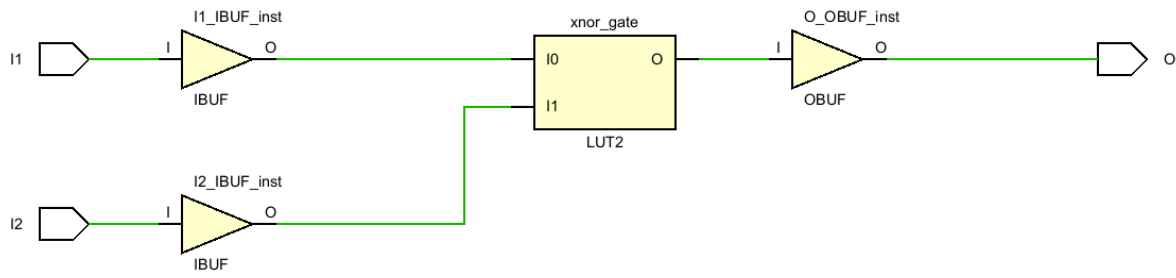
xnor_gate XNOR_GATE(
    .I1(I1),
    .I2(I2),
    .O(O));

initial begin
I1=0;
I2=0;
#10;
I1=0;
I2=1;
#10;
I1=1;
I2=0;
#10;
I1=1;
I2=1;
#10;
end
endmodule
```

Behavioural simulation results of XNOR gate:



RTL schematic and technology schematic of XNOR module is same:



TRI module

This module has 2 1-bit input called I and E, 1-bit output O. Written in = ? () : () format. $E = 1 \Rightarrow O = I$, $E = 0 \Rightarrow O = \text{High impedance (Z)}$

Source code:

```
//TRI
module TRI (
    input I,
    input E,
    output O);

assign O = E?I:1'bz;
endmodule
```

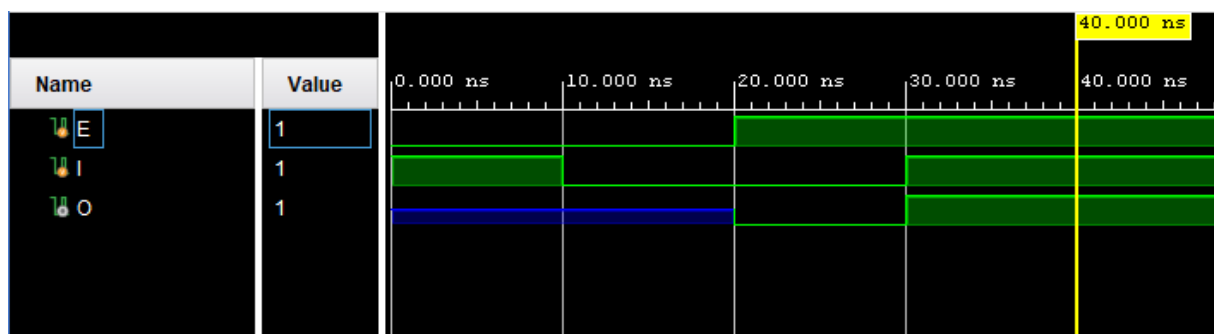
Testbench code:

```
module SSI_Library_tb;
reg E;
reg I;
wire O;

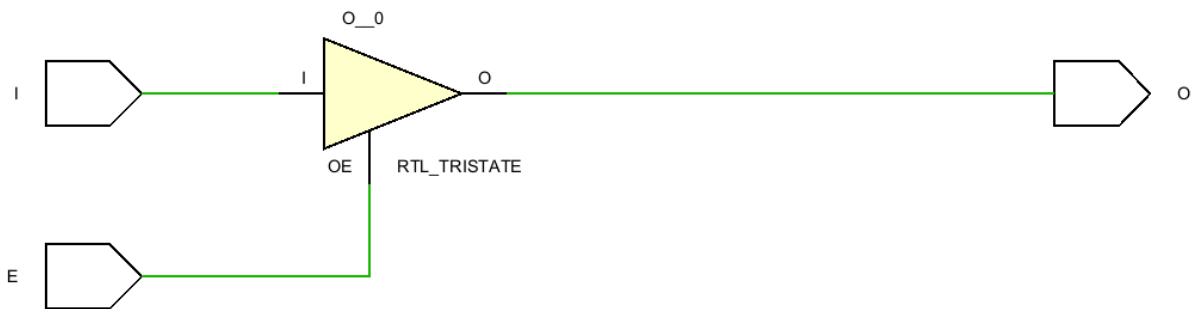
TRI TRI_GATE (
    .I(I),
    .E(E),
    .O(O));

initial begin
    I=1;
    E=0;
    #10;
    I=0;
    E=0;
    #10;
    I=0;
    E=1;
    #10;
    I=1;
    E=1;
    #10;
end
endmodule
```

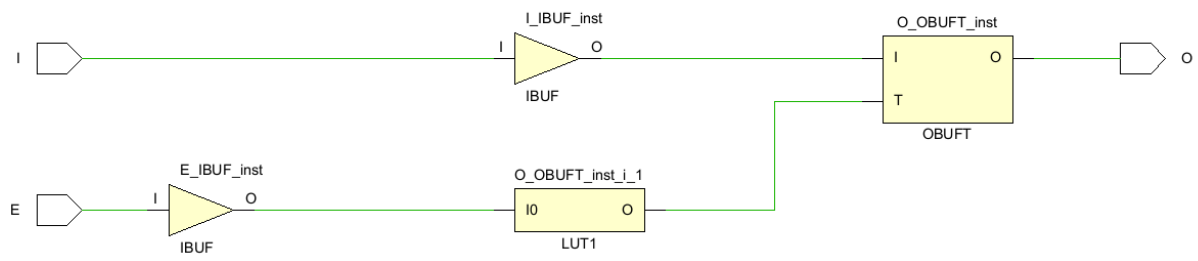
Behavioural simulation results of TRI module:



RTL Schematic of TRI module:



Technology schematic of TRI module:



TOP MODULE

This module has all SSI_Library modules and 15 bit input (because no need 16 bit while not gate input is 1 bit) and 8 bit output.

Source code:

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

);

and_gate AND_GATE (.I1(IN[0]),.I2(IN[1]),.O(OUT[0]));
or_gate OR_GATE (.I1(IN[2]),.I2(IN[3]),.O(OUT[1]));
not_gate NOT_GATE_I1 (.I(IN[4]),.O(OUT[2]));
nand_gate NAND_GATE (.I1(IN[5]),.I2(IN[6]),.O(OUT[3]));
nor_gate NOR_GATE (.I1(IN[7]),.I2(IN[8]),.O(OUT[4]));
xor_gate XOR_GATE (.I1(IN[9]),.I2(IN[10]),.O(OUT[5]));
xnor_gate XNOR_GATE (.I1(IN[11]),.I2(IN[12]),.O(OUT[6]));
TRI TRI_MODULE (.I(IN[13]),.E(IN[14]),.O(OUT[7]));
endmodule
```

Now this module does every operation which is wanted in experiment-1 paper.

In testbench code, i assigned the exact values of the inputs of modules to another wire sets with their operand name. These assignments were not necessary but they are needed to see everything clear and faster.

Testbench code:

```

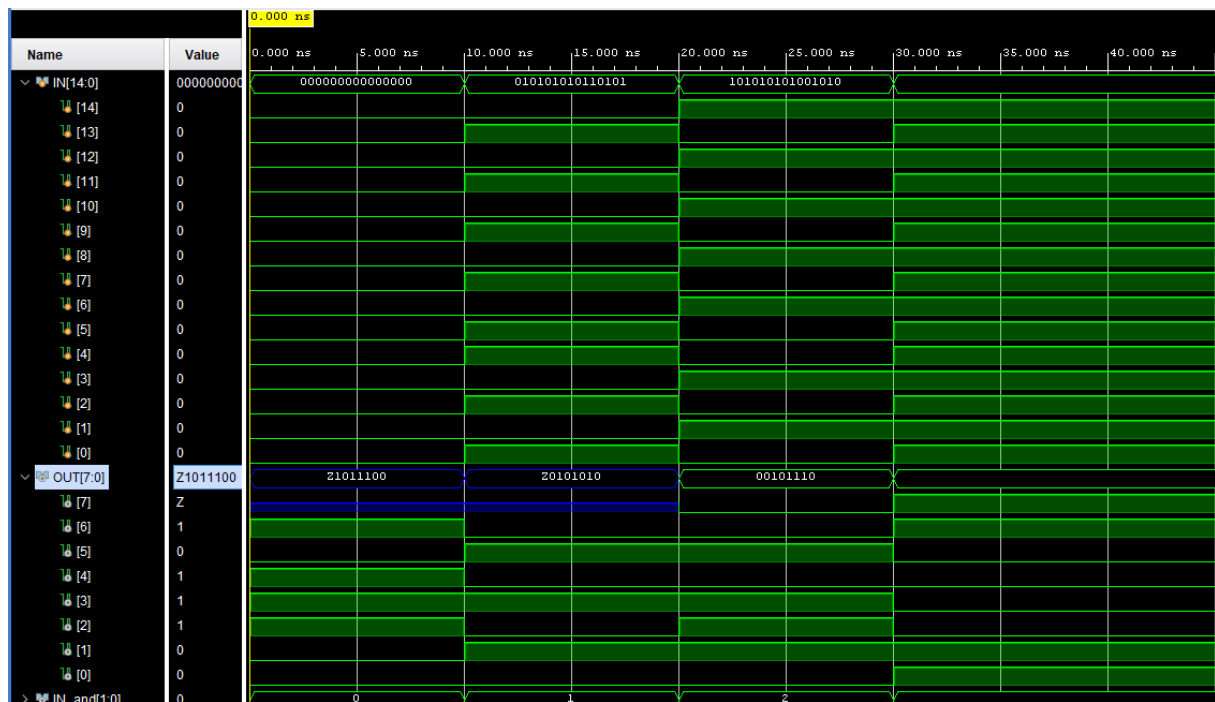
`timescale 1ns / 1ps

module SSI_Library_tb;
reg [14:0] IN;
wire [7:0] OUT;
wire OUT_and;
wire OUT_or;
wire OUT_not;
wire OUT_nand;
wire OUT_nor;
wire OUT_xor;
wire OUT_xnor;
wire OUT_TRI;
wire [1:0] IN_and = IN[1:0];
wire [1:0] IN_or = IN[3:2];
wire IN_not = IN[4];
wire [1:0] IN_nand = IN[6:5];
wire [1:0] IN_nor = IN[8:7];
wire [1:0] IN_xor = IN[10:9];
wire [1:0] IN_xnor = IN[12:11];
wire [1:0] IN_TRI = IN[14:13];
assign OUT_and = OUT[0];
assign OUT_or = OUT[1];
assign OUT_not = OUT[2];
assign OUT_nand = OUT[3];
assign OUT_nor = OUT[4];
assign OUT_xor = OUT[5];
assign OUT_xnor = OUT[6];
assign OUT_TRI = OUT[7];

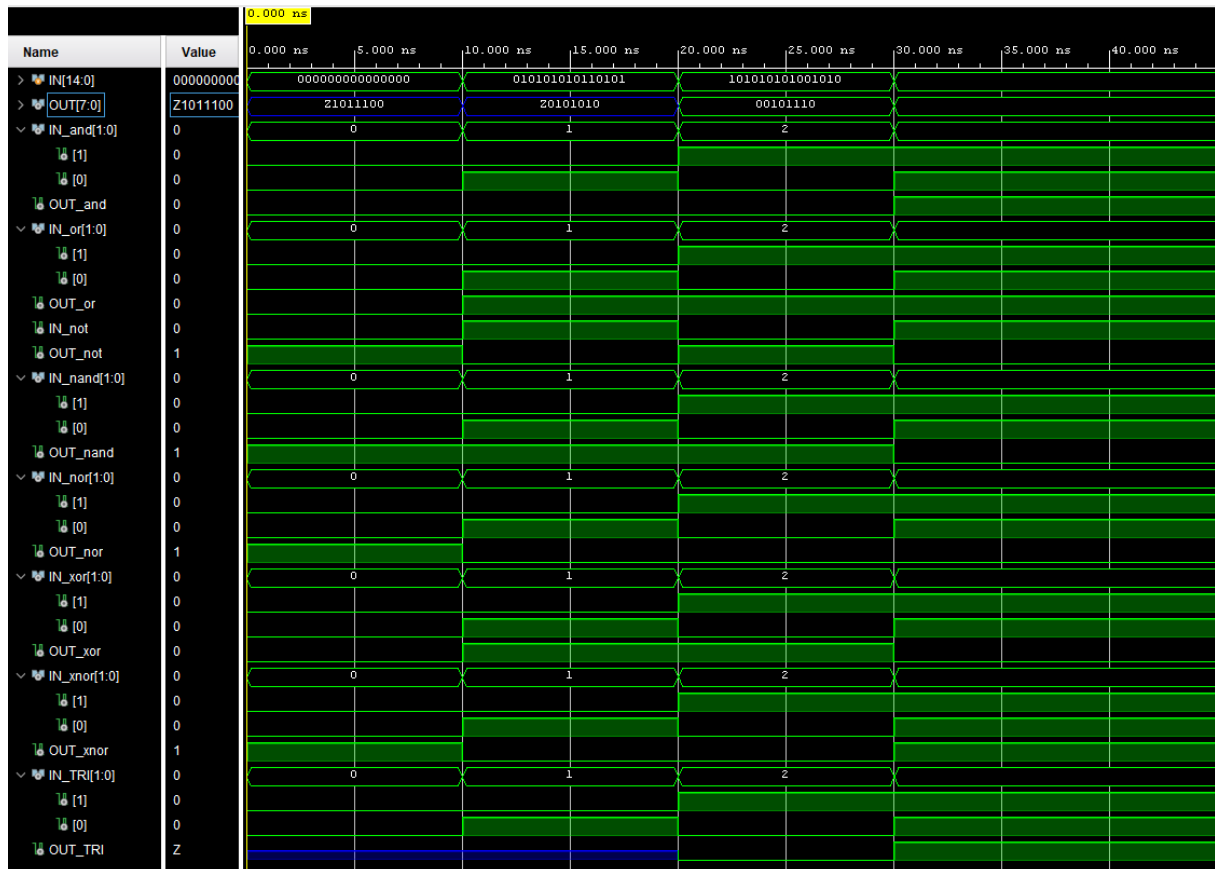
top_module TOP (.IN(IN),.OUT(OUT));
initial begin
IN = 15'b00_00_00_00_00_0_00_00;
#10;
IN = 15'b01_01_01_01_01_1_01_01;
#10;
IN = 15'b10_10_10_10_10_0_10_10;
#10;
IN = 15'b11_11_11_11_11_1_11_11;
end
endmodule

```

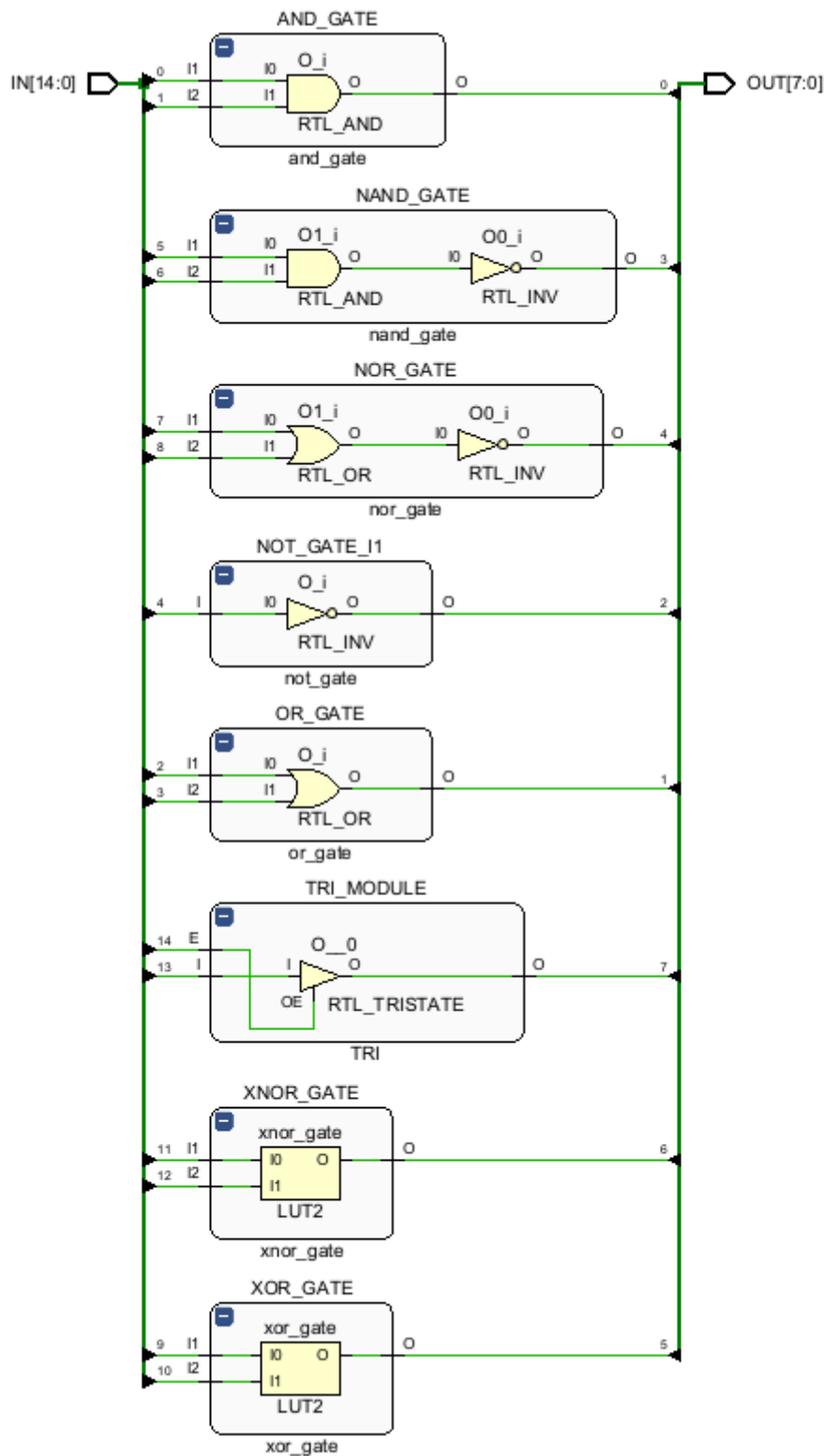

Behavioural simulation results:



I assigned the Inputs with the name of the gate and outputs as them. So here is another perspective to behavioural simulation which is more clear that it works true.

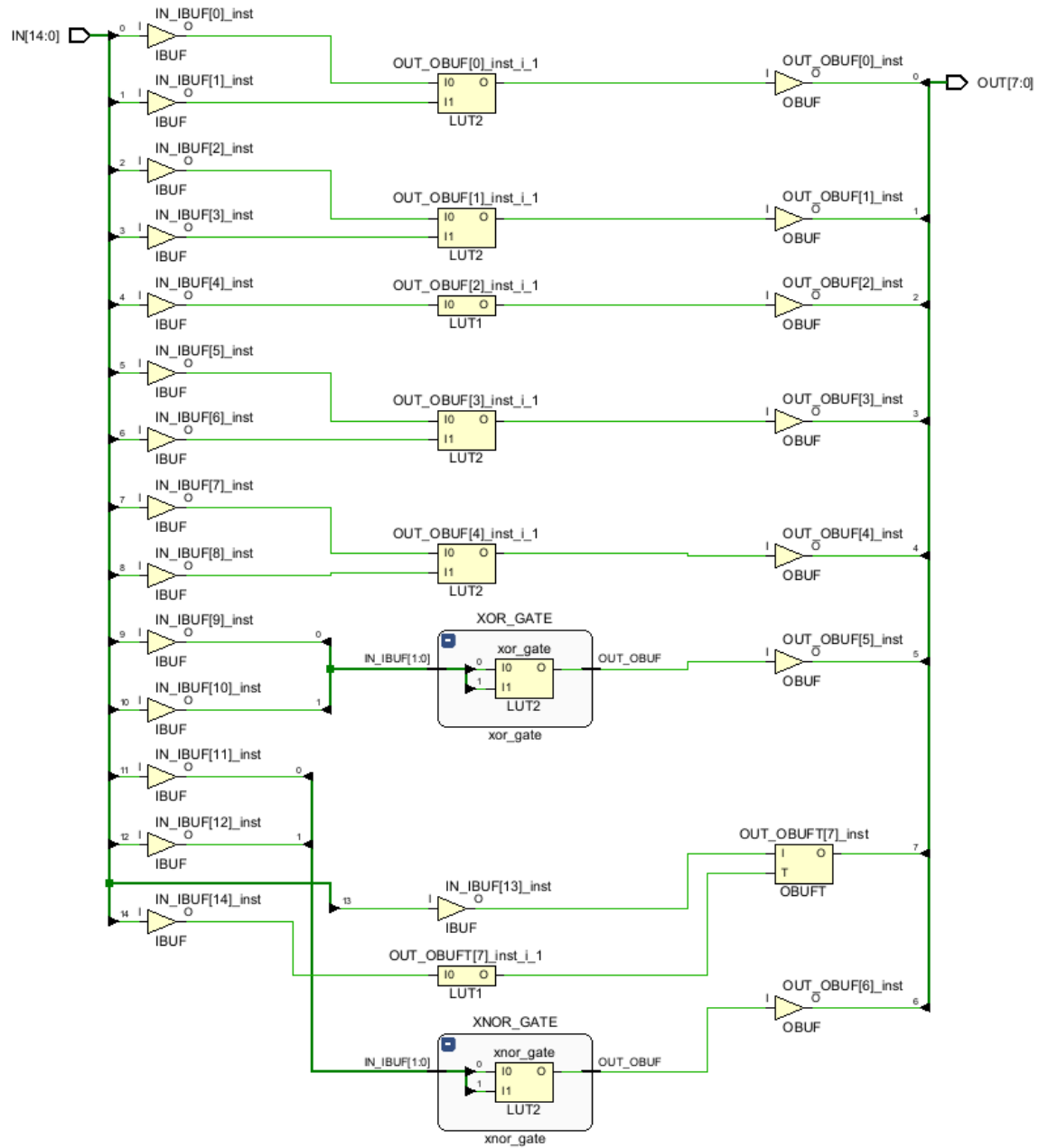


RTL schematic of top module:



















Everything seems clear here.

Technology schematic of top module:



Last timing report of top_module after implementation with constraint file.

Name	Slack	Levels	High Fanout	From	To	Total ...  1	Logic Delay	Net Delay
 Path 16	∞	3	1	IN[4]	OUT[2]	2.958	1.521	1.437
 Path 15	∞	3	1	IN[2]	OUT[1]	2.458	1.527	0.932
 Path 14	∞	3	1	IN[11]	OUT[6]	2.420	1.534	0.886
 Path 13	∞	3	1	IN[0]	OUT[0]	2.420	1.529	0.891
 Path 12	∞	3	1	IN[5]	OUT[3]	2.356	1.531	0.825
 Path 11	∞	3	1	IN[10]	OUT[5]	2.353	1.525	0.828
 Path 10	∞	3	1	IN[7]	OUT[4]	2.336	1.543	0.793
<hr/>								
Name	Slack	Levels	High Fanout	From	To	Total ...  1	Logic Delay	Net Delay
 Path 1	∞	3	1	IN[12]	OUT[6]	11.062	5.131	5.931
 Path 2	∞	3	1	IN[8]	OUT[4]	10.346	4.620	5.726
 Path 3	∞	3	1	IN[4]	OUT[2]	9.540	5.131	4.409
 Path 4	∞	3	1	IN[9]	OUT[5]	9.348	4.603	4.745
 Path 5	∞	3	1	IN[3]	OUT[1]	8.644	5.136	3.509
 Path 6	∞	3	1	IN[6]	OUT[3]	8.486	5.132	3.354
 Path 7	∞	3	1	IN[1]	OUT[0]	8.371	5.132	3.238

PIN Assignment list:

IN[1:0] => AND module inputs	,	O[0] => AND module output
IN[3:2] => OR module inputs	,	O[1] => OR module output
IN[4] => NOT module input	,	O[2] => NOT module output
IN[6:5] => NAND module inputs	,	O[3] => NAND module output
IN[8:7] => NOR module inputs	,	O[4] => NOR module output
IN[10:9] => XOR module inputs	,	O[5] => XOR module output
IN[12:11] => XNOR module inputs	,	O[6] => XNOR module output
IN[14:13] => TRI module inputs	,	O[7] => TRI module output

SW bit array is defined to the same indexes as IN bit array and LED bit array defined to the same indexes as the OUT bit array.

Bit file generated and will load in class to the FPGA.

Research about Look up tables

Lets say there is a function as $y = f(x)$.

Look up tables differs as instead of calculating y from an x value, it stores the y value which is directly addressable with x . So retrieving a result value with look up tables becomes faster than the computing process, because getting a value from memory is more faster than the computing in input/output calculations.

Look up tables can be defined as a customized truth table which output values saved in a small RAM. Instead of connect many gates as NAND or NAND, designer simply use a look up table to get all 2^n results for n input which designer wanted.

Research about fan-in and fan-out

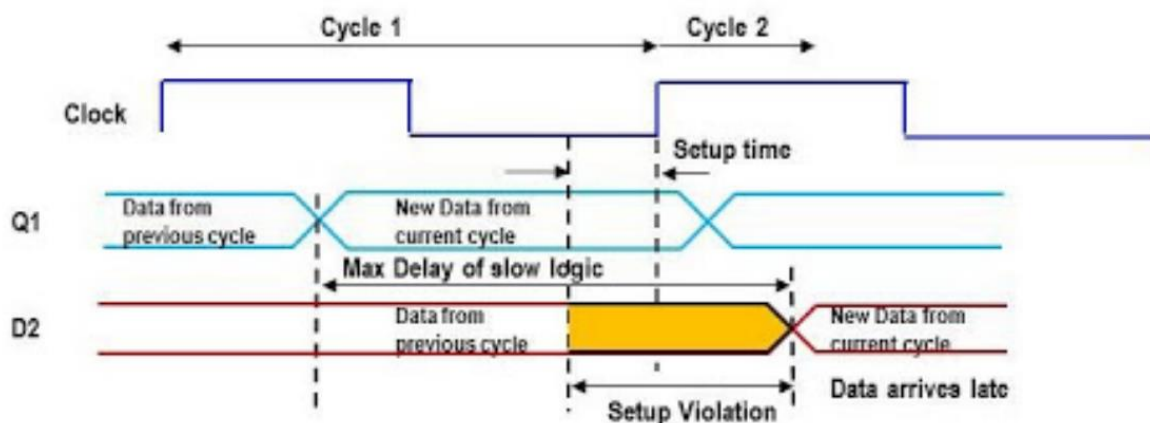
Fan-in is an term to explain of the maximum number of input signals which feed the input equations of a logic cell. Fan-out is another term for the maximum number of output signals that are fed by the output equations of the logic cell.

For example, my AND_module has 2 inputs as IN[0] and IN[1] and 1 output as OUT[0], in reality the AND gate has 2 more inputs as GND and VDD, so fan-in is 4 for AND_module logic cell and fan-out is 1.

Research about setup time and hold time delays

Setup time is the time required for the input to a Flip-Flop to be stable before a clock edge. Hold time is similar 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.

Setup time example:



Hold time example:

