HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULITY OF COMPUTER SCIENCE AND ENGINEERING

LOGIC DESIGN WITH HDL LAB

STUDENT REPORT

Class: CC01		Group: 04
Student's name:	Trịnh Sơn Lâm	1852502
	Nguyễn Trọng Tính	1752545
	Lê Bá Thành	1852739
	Hoàng Nhật Quang	1852691
Lecturer	Ph.D Phạm Quốc Cường	

APRIL 1, 2022

WEEK 1 Introduction & Structural Model

Exercise:

Exercise 1

a. Design a 1-to-2 decoder using structure model as the following circuit:

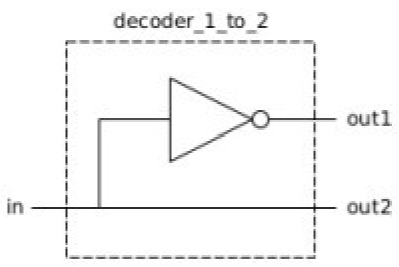
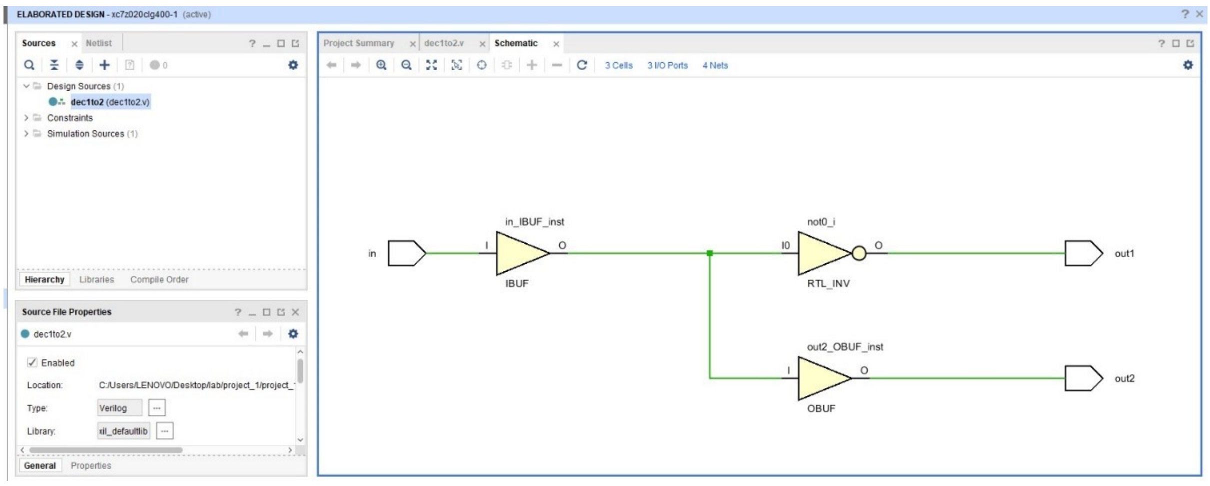


Figure 1: 1-to-2 decoder

We have file code is **dec1to2.v**

And then we have RTL schemetic circuit:


ELABORATED DESIGN - xc7z020clg400-1 (active)

Sources | Netlist

Design Sources (1)
dec1to2 (dec1to2.v)
Constraints
Simulation Sources (1)

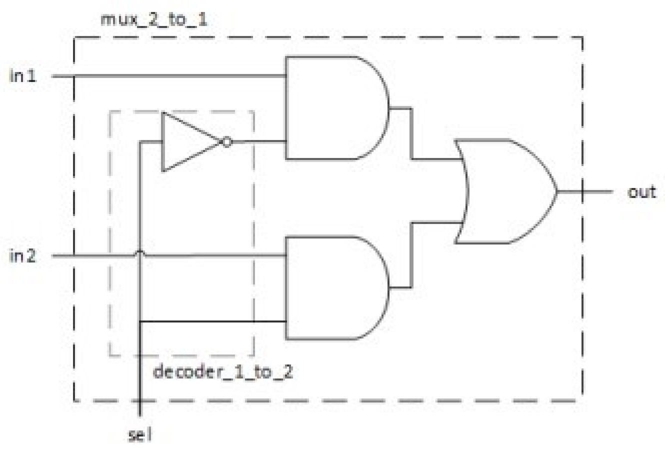
Hierarchy | Libraries | Compile Order

Source File Properties
dec1to2.v
✓ Enabled
Location: C:/Users/LENOVO/Desktop/lab/project_1/project_
Type: Verilog
Library: xil_defaultlib

General | Properties

Figure 2: 1-to-2 decoder in RTL schemetic circuit

b. Design a 2-to-1 multiplexer using structure model and hierarchical design (reuse the module decoder 1 to 2) as following circuit:



Firgure 3: 2-to-1 multiplexer

We have file code is **mux2to1.v**

And then we have RTL schemetic circuit:

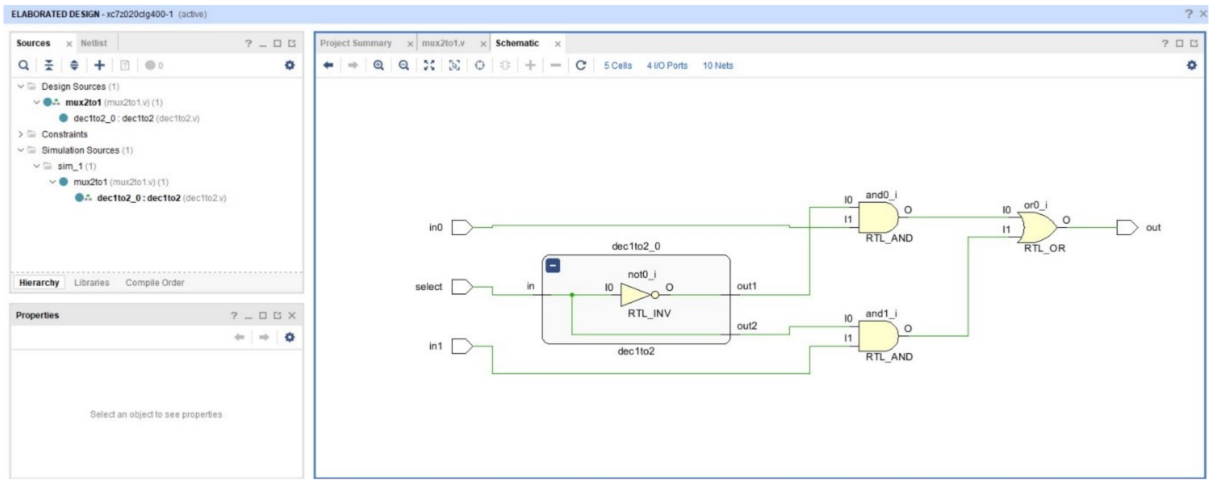


Figure 4: 2-to-1 multiplexer in RTL schemetic circuit

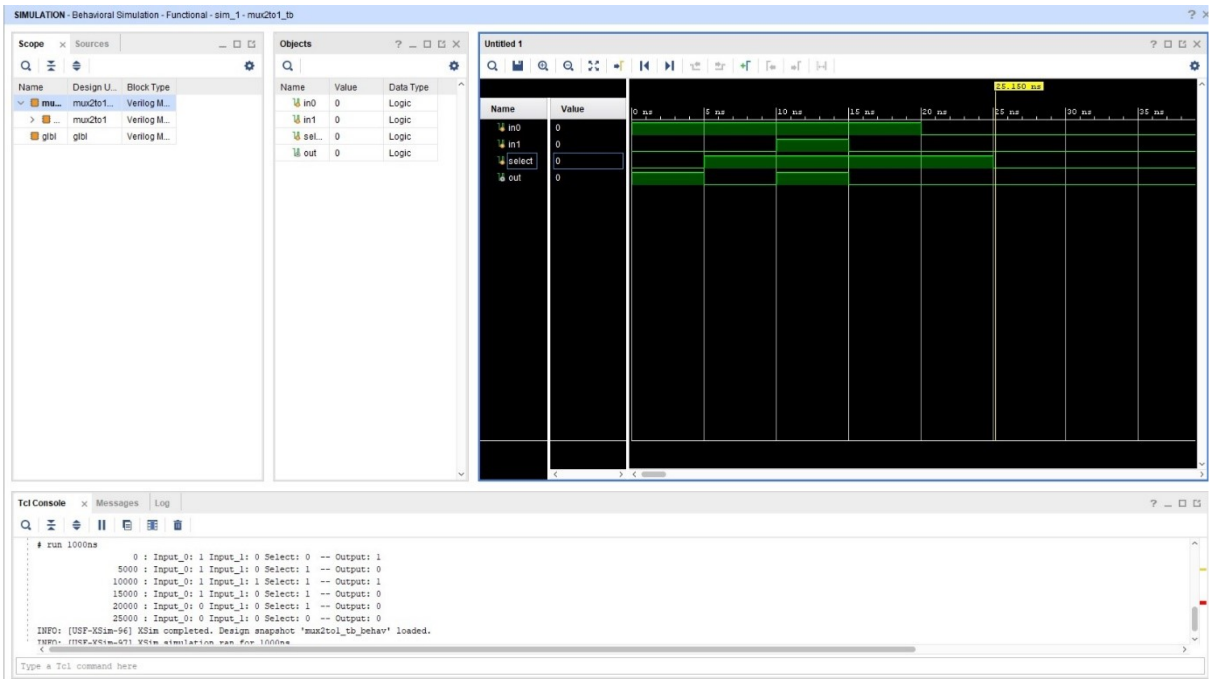
Exercise 2

a. Write a test bench for the 2-to-1 Multiplexer in Exercise 1 then use Vivado Simulator to simulate the design, students can use the given example source code.

Let’s analyse the structure of a test bench then point out the differences between an RTL code and a test bench code.

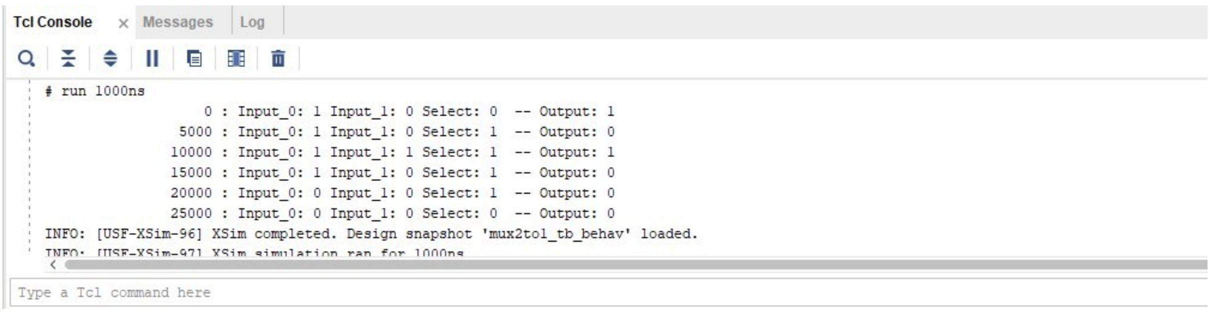
We have file code is **mux2to1_tb.v**

And we can see that, RTL code is using templates that will synthesise to digital logic components while testbench is used for simulation purpose only (not for synthesis). In other words, RTL code don't have any specific timing info on input signal High/ Low.


SIMULATION - Behavioral Simulation - Functional - sim_1 - mux2to1_tb
Scope
Sources
Objects
Untitled 1
in0
in1
select
out
Tcl Console
Messages
Log

```
# run 1000ns
            0 : Input_0: 1 Input_1: 0 Select: 0  -- Output: 1
         5000 : Input_0: 1 Input_1: 0 Select: 1  -- Output: 0
        10000 : Input_0: 1 Input_1: 1 Select: 1  -- Output: 1
        15000 : Input_0: 1 Input_1: 0 Select: 1  -- Output: 0
        20000 : Input_0: 0 Input_1: 0 Select: 1  -- Output: 0
        25000 : Input_0: 0 Input_1: 0 Select: 0  -- Output: 0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'mux2to1_tb_behav' loaded.
```

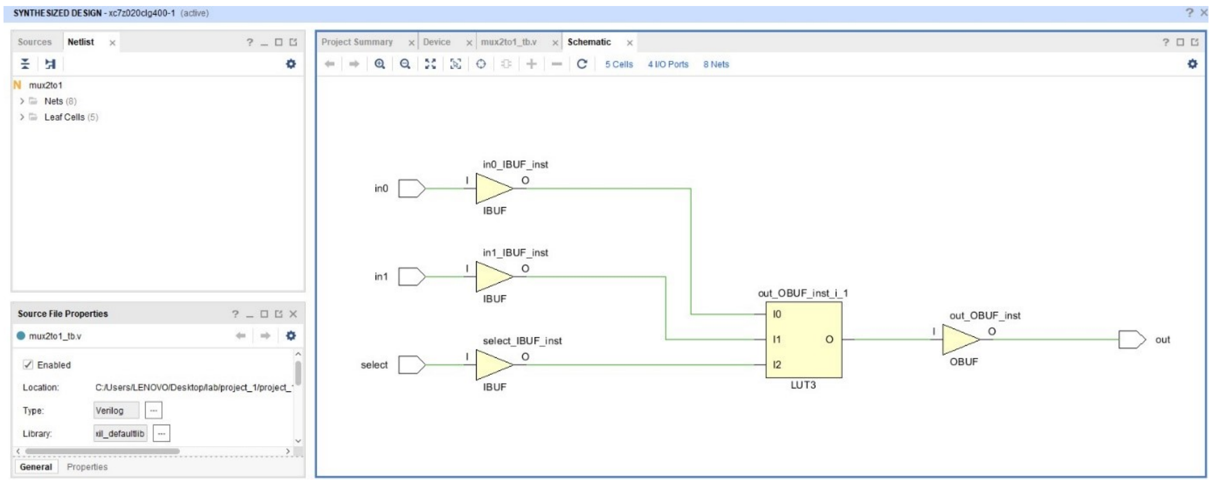
Figure 5: The waveform of test bench for 2-1 multiplexer



```
# run 1000ns
            0 : Input_0: 1 Input_1: 0 Select: 0  -- Output: 1
         5000 : Input_0: 1 Input_1: 0 Select: 1  -- Output: 0
        10000 : Input_0: 1 Input_1: 1 Select: 1  -- Output: 1
        15000 : Input_0: 1 Input_1: 0 Select: 1  -- Output: 0
        20000 : Input_0: 0 Input_1: 0 Select: 1  -- Output: 0
        25000 : Input_0: 0 Input_1: 0 Select: 0  -- Output: 0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'mux2to1_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Figure 6: Tcl console window to see output of $monitor command.

b. Then, perform the Synthesis, compare the Synthesis’s Schematic and the RTL Analysis’s schematic

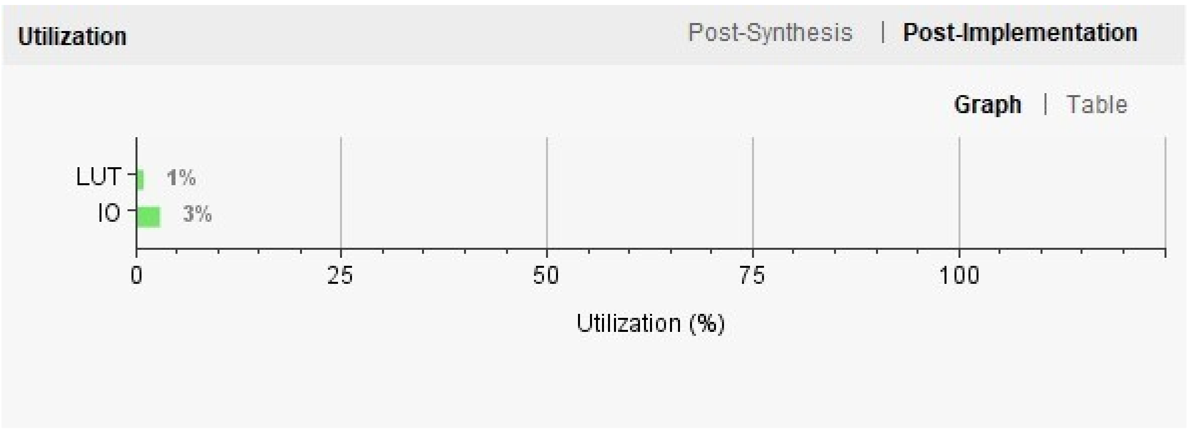


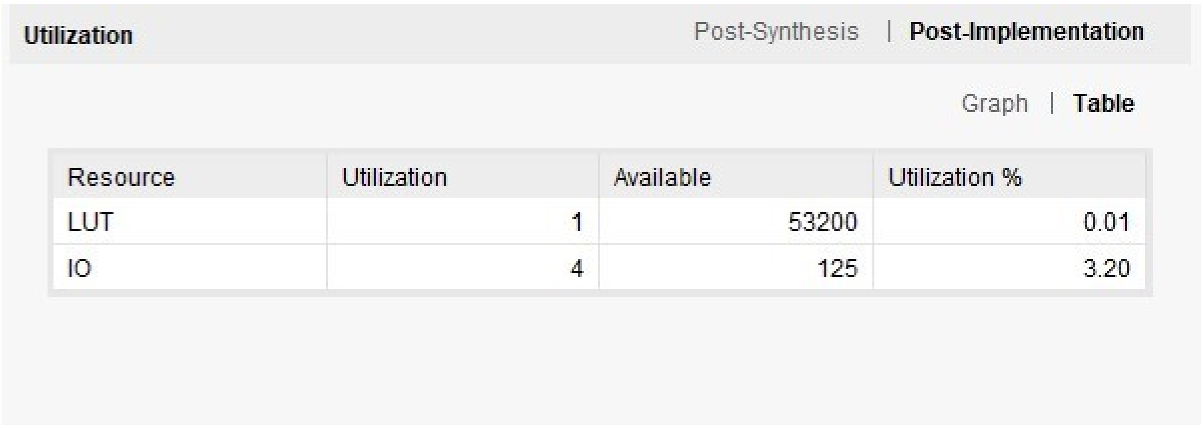
✓ Enabled
Location: C:/Users/LENOVO/Desktop/lab/project_1/project_
Type: Verilog
Library: xil_defaultlib

Figure 7: 2-to-1 multiplexer in Synthesis's schemetic circuit

We can see that RTL schemetic can be viewed as a gate-level schematic.It shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, AND gates, and OR gates, that are independent of the targeted Xilinx device. While Synthesis schemetic an be viewed as an architecture-specific schematic. This schematic is generated after the optimization and technology targeting phase of the synthesis process. It shows a representation of the design in terms of logic elements optimized to the target Xilinx device or "technology"; for example, in terms of of LUTs, carry logic, I/O buffers, and other technology-specific components. Viewing this schematic allows you to see a technology-level representation of your HDL optimized for a specific Xilinx architecture, which might help you discover design issues early in the design process

c. After that, run the Implementation, check the Utilization report in Project Summary for used resources





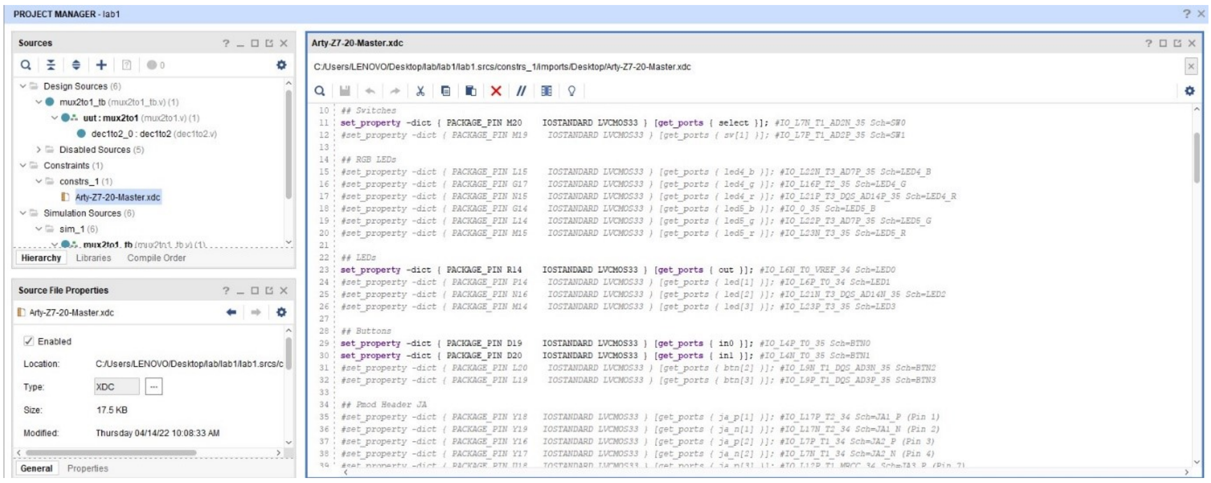
Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	1	53200	0.01
IO	4	125	3.20

Figure 8: The utilization report

d. . Add the Arty-Z7 constraint file to the project, assign pin for the design as follow:

We change some lines in Arty-Z7 constraint as follow:



Arty-Z7-20-Master.xdc

C:/Users/LENOVO/Desktop/lab/lab1/lab1.srcs/constrs_1/imports/Desktop/Arty-Z7-20-Master.xdc

```
## Switches
set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS33 } [get_ports { select }]; #IO_L7N_T1_AD2N_35 Sch=SW0
#set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; #IO_L7P_T1_AD2P_35 Sch=SW1

## RGB LEDs
#set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { led4_b }]; #IO_L22N_T3_AD7P_35 Sch=LED4_B
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led4_g }]; #IO_L16P_T2_35 Sch=LED4_G
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { led4_r }]; #IO_L21P_T3_DQS_AD14P_35 Sch=LED4_R
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led5_b }]; #IO_0_35 Sch=LED5_B
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { led5_g }]; #IO_L22P_T3_AD7P_35 Sch=LED5_G
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led5_r }]; #IO_L23N_T3_35 Sch=LED5_R

## LEDs
set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { out }]; #IO_L6N_T0_VREF_34 Sch=LED0
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L6P_T0_34 Sch=LED1
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_L21N_T3_DQS_AD14N_35 Sch=LED2
#set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L23P_T3_35 Sch=LED3

## Buttons
set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports { in0 }]; #IO_L4P_T0_35 Sch=BTN0
set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { in1 }]; #IO_L4N_T0_35 Sch=BTN1
#set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L9N_T1_DQS_AD3N_35 Sch=BTN2
#set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L9P_T1_DQS_AD3P_35 Sch=BTN3

## Pmod Header JA
#set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports { ja_p[1] }]; #IO_L17P_T2_34 Sch=JA1_P (Pin 1)
#set_property -dict { PACKAGE_PIN Y19   IOSTANDARD LVCMOS33 } [get_ports { ja_n[1] }]; #IO_L17N_T2_34 Sch=JA1_N (Pin 2)
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { ja_p[2] }]; #IO_L7P_T1_34 Sch=JA2_P (Pin 3)
#set_property -dict { PACKAGE_PIN Y17   IOSTANDARD LVCMOS33 } [get_ports { ja_n[2] }]; #IO_L7N_T1_34 Sch=JA2_N (Pin 4)
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { ja_p[3] }]; #IO_L12P_T1_MRCC_34 Sch=JA3_P (Pin 7)
```

Figure 9: Assign pin for Arty-Z27 before running for FPGA

Then, the generated bitstream file is **mux2to1.bit**

Exercise 3

a. Design a half-adder circuit using structural model.

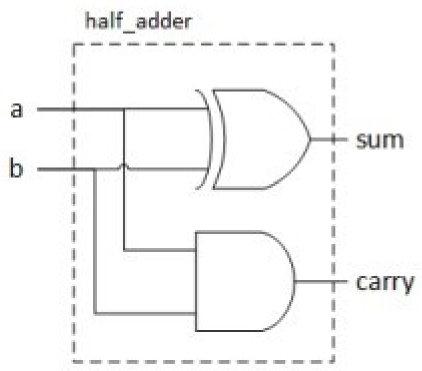


Figure 10: Half adder

We have file code is **half_adder.v**

b. Design a full-adder circuit using structural model. Reuse the half-adder module.

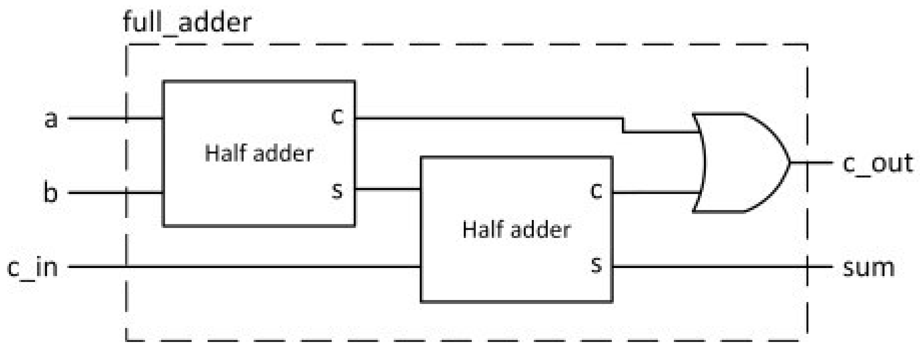


Figure 11: Full adder

We have file code is **full_adder.v**

And then we have the file code for test bench of full adder is **full_adder_tb.v**

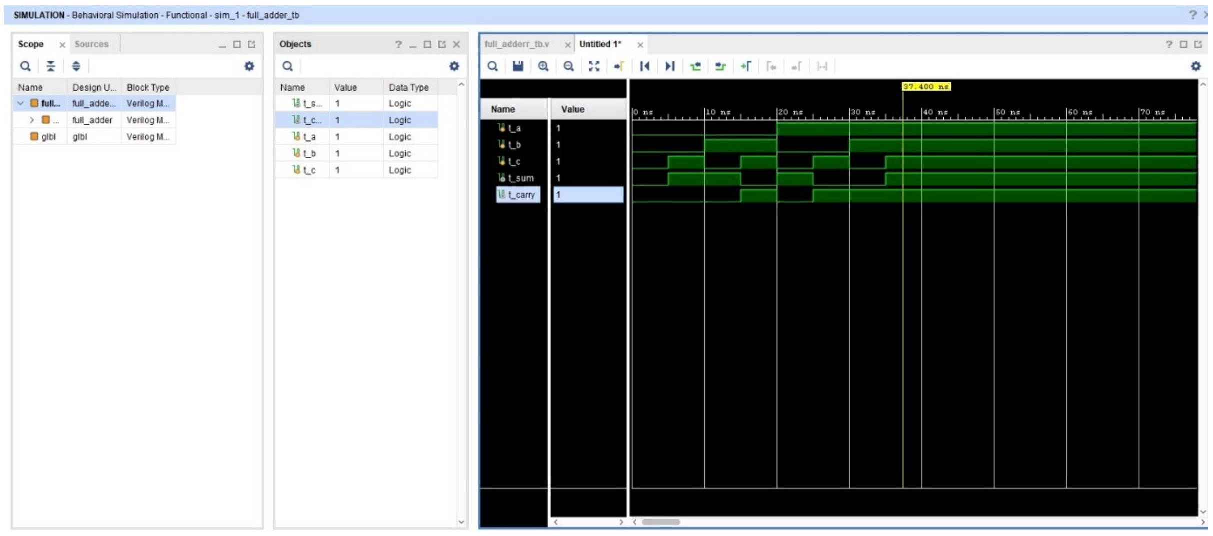
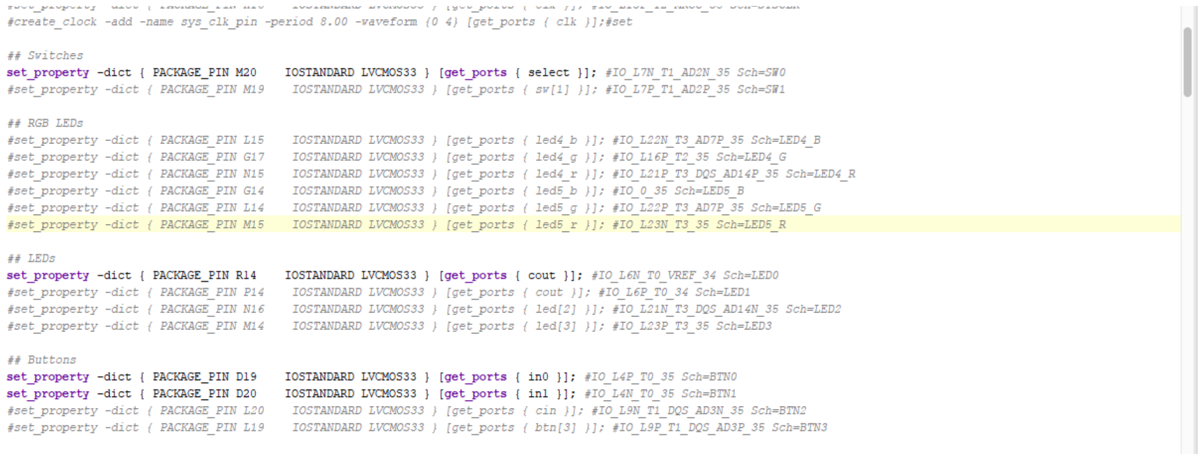


Figure 12: The waveform of test bench for full adder

Test the implemented circuit on Arty-Z7 board using switches/buttons and LEDs. We have some changes in constraint as follow:



```
#create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];#set

## Switches
set_property -dict { PACKAGE_PIN M20    IOSTANDARD LVCMOS33 } [get_ports { select }]; #IO_L7N_T1_AD2N_35 Sch=SW0
#set_property -dict { PACKAGE_PIN M19    IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; #IO_L7P_T1_AD2P_35 Sch=SW1

## RGB LEDs
#set_property -dict { PACKAGE_PIN L15    IOSTANDARD LVCMOS33 } [get_ports { led4_b }]; #IO_L22N_T3_AD7P_35 Sch=LED4_B
#set_property -dict { PACKAGE_PIN G17    IOSTANDARD LVCMOS33 } [get_ports { led4_g }]; #IO_L16P_T2_35 Sch=LED4_G
#set_property -dict { PACKAGE_PIN N15    IOSTANDARD LVCMOS33 } [get_ports { led4_r }]; #IO_L21P_T3_DQS_AD14P_35 Sch=LED4_R
#set_property -dict { PACKAGE_PIN G14    IOSTANDARD LVCMOS33 } [get_ports { led5_b }]; #IO_0_35 Sch=LED5_B
#set_property -dict { PACKAGE_PIN L14    IOSTANDARD LVCMOS33 } [get_ports { led5_g }]; #IO_L22P_T3_AD7P_35 Sch=LED5_G
#set_property -dict { PACKAGE_PIN M15    IOSTANDARD LVCMOS33 } [get_ports { led5_r }]; #IO_L23N_T3_35 Sch=LED5_R

## LEDs
set_property -dict { PACKAGE_PIN R14    IOSTANDARD LVCMOS33 } [get_ports { cout }]; #IO_L6N_T0_VREF_34 Sch=LED0
#set_property -dict { PACKAGE_PIN P14    IOSTANDARD LVCMOS33 } [get_ports { cout }]; #IO_L6P_T0_34 Sch=LED1
#set_property -dict { PACKAGE_PIN N16    IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_L21N_T3_DQS_AD14N_35 Sch=LED2
#set_property -dict { PACKAGE_PIN M14    IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L23P_T3_35 Sch=LED3

## Buttons
set_property -dict { PACKAGE_PIN D19    IOSTANDARD LVCMOS33 } [get_ports { in0 }]; #IO_L4P_T0_35 Sch=BTN0
set_property -dict { PACKAGE_PIN D20    IOSTANDARD LVCMOS33 } [get_ports { in1 }]; #IO_L4N_T0_35 Sch=BTN1
#set_property -dict { PACKAGE_PIN L20    IOSTANDARD LVCMOS33 } [get_ports { cin }]; #IO_L9N_T1_DQS_AD3N_35 Sch=BTN2
#set_property -dict { PACKAGE_PIN L19    IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L9P_T1_DQS_AD3P_35 Sch=BTN3
```

And we have the generated bitstream is **full_adder.bit**

c. Design a 4-bit ripple carry adder using structural model. Reuse the implemented full-adder. Write a test bench to simulate the implemented circuit.

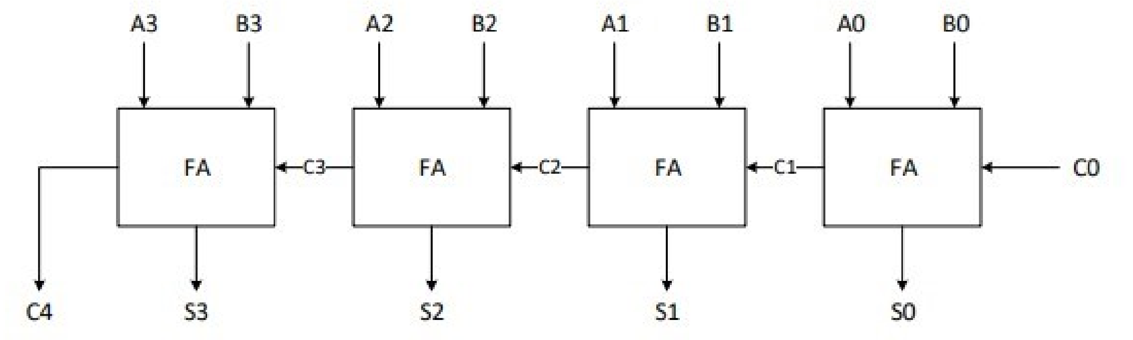


Figure 13: 4-bit ripple carry adder

We have file code is **ripple_adder_4bit.v**

And then we have the file code for test bench of ripple adder 4-bit is **ripple_adder_4bit_tb.v**

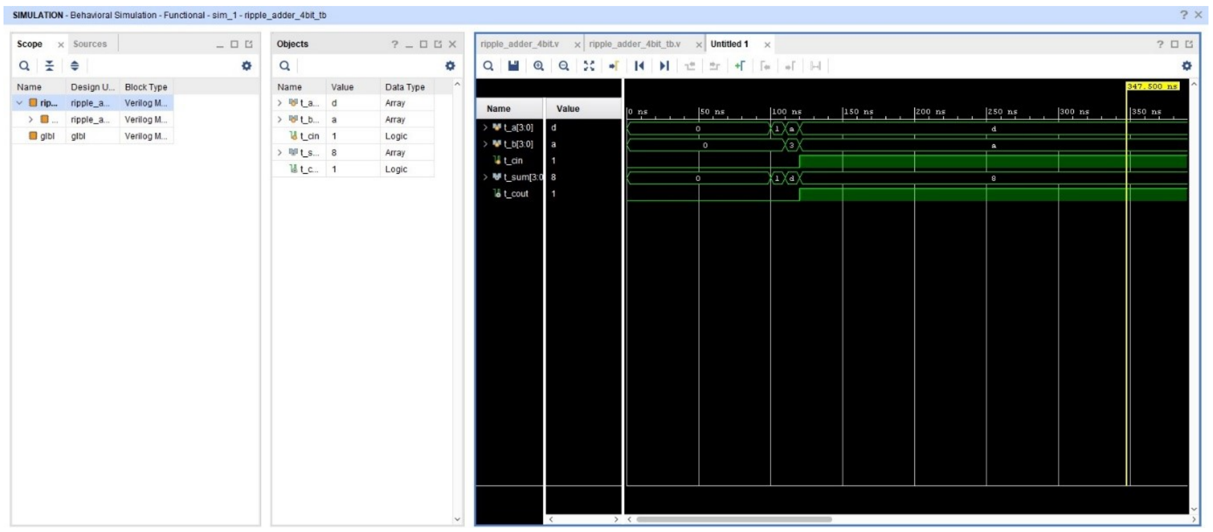
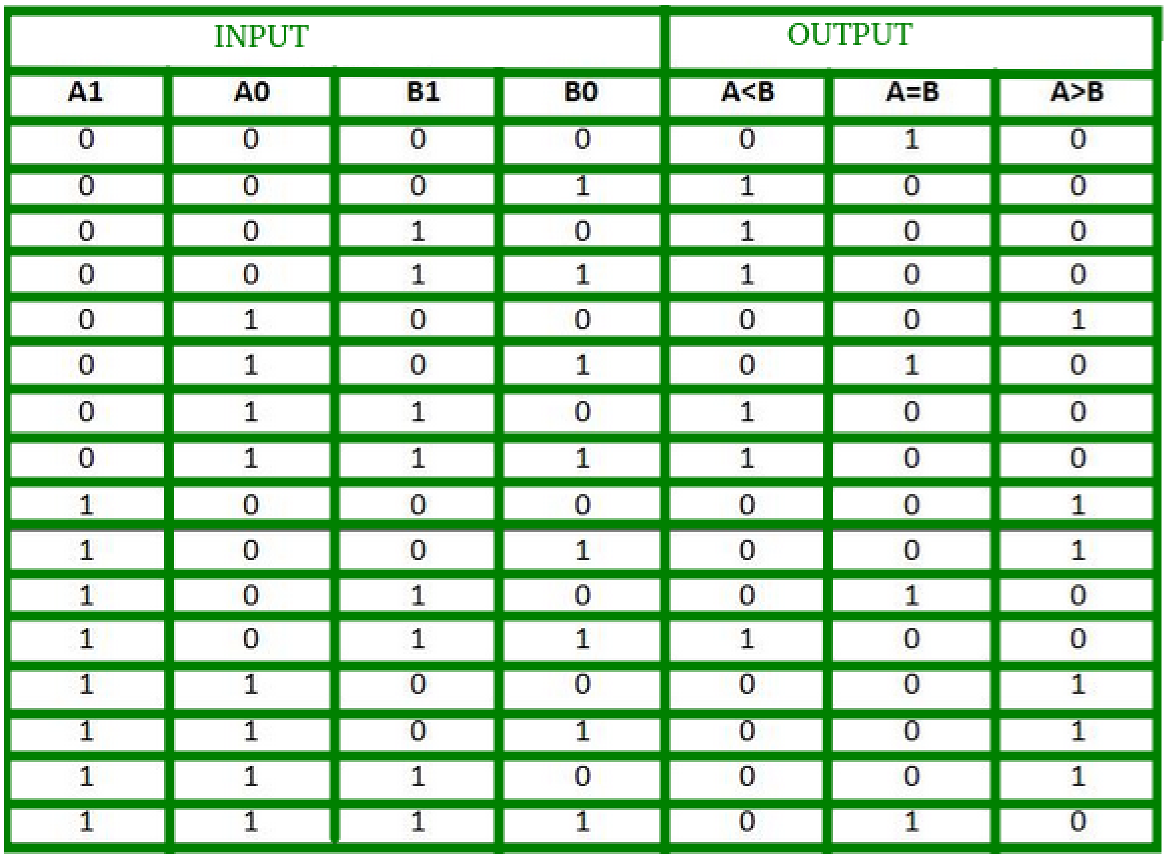


Figure 14: The waveform of test bench for 4-bit ripple carry adder

Exercise 4

Analyse the functions of each output of the 2-bit comparator, then determine the functions of 4-bit comparator outputs.

Firstly, we have the true table of 2-bit comparator.



INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Figure 15: The true table of 2-bit comparator

And the we have K-map for each output:

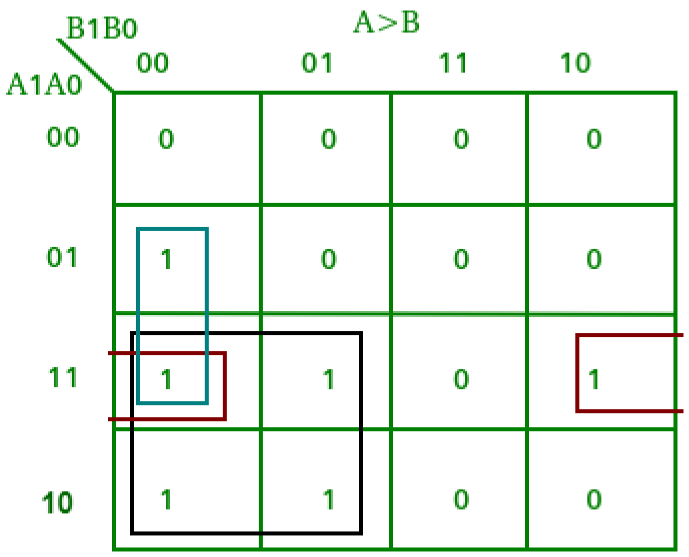


Figure 17: A>B

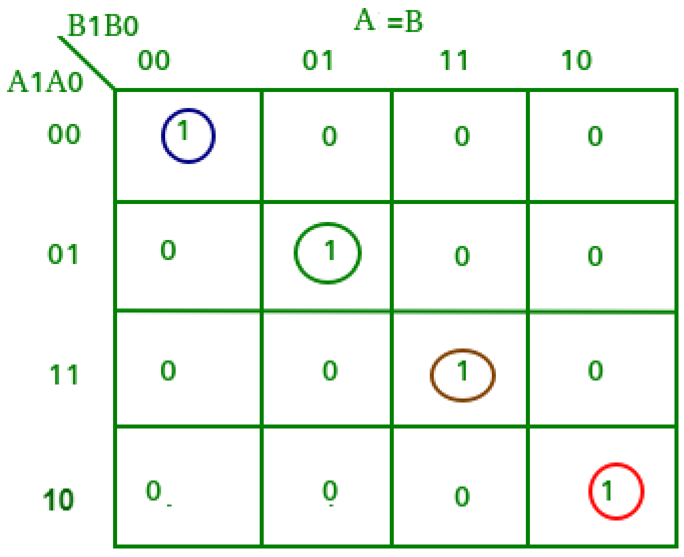


Figure 18: A=B

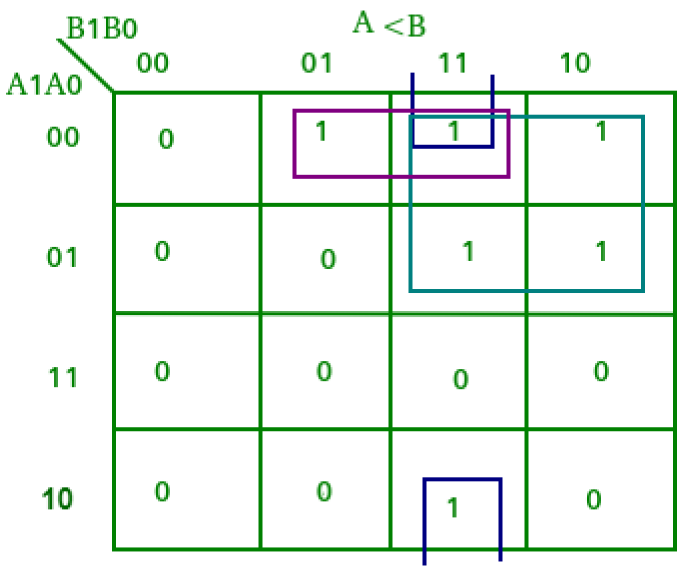


Figure 17: A<B

So, we will have logical expression for each output:

A>B: A1B1' + A0B1' B0’ + A1A0B0’

: A1B1’ + (A1$\odot$B1) A0B0′

A=B: A1’A0’B1’B0’ + A1’A0B1’B0 + A1A0B1B0 + A1A0’B1B0’

: A1’B1’ (A0’B0’ + A0B0) + A1B1 (A0B0 + A0’B0’)

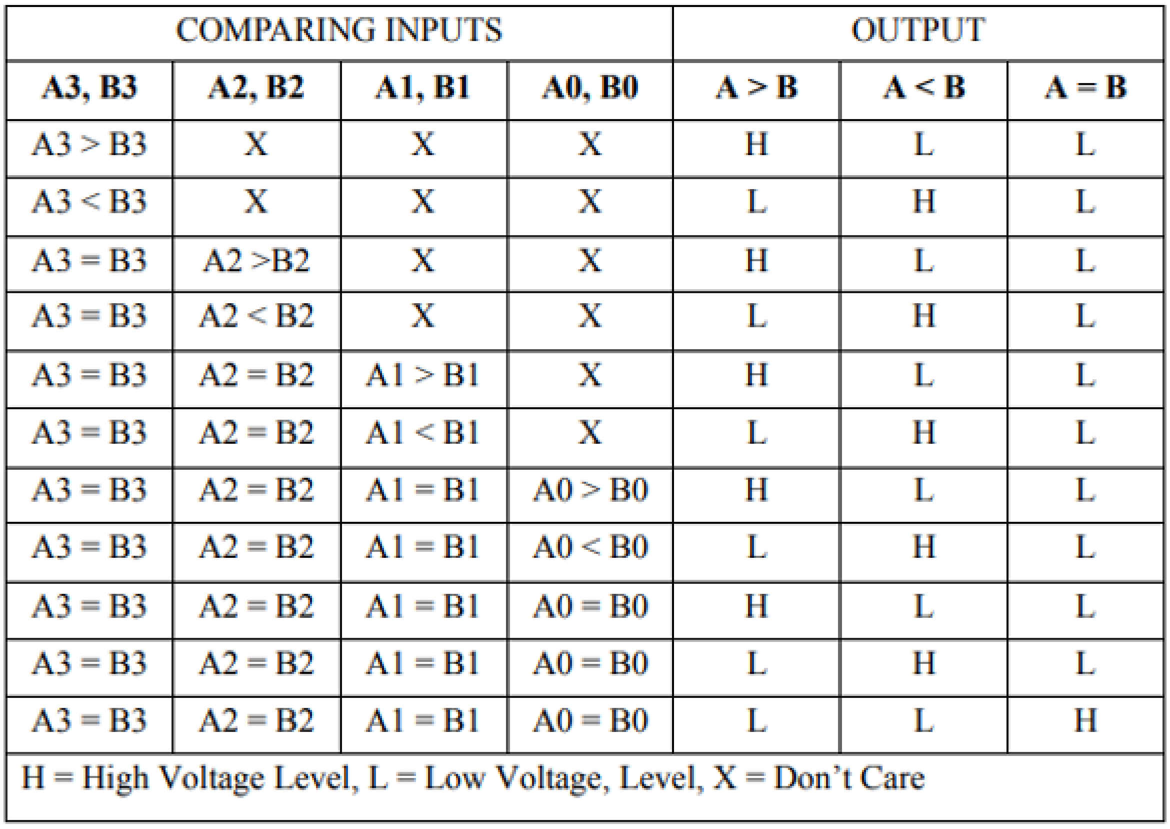
: $(A0B0 + A0'B0')(A1B1 + A1'B1')$

: $(A0 \odot B0)(A1 \odot B1)$

A<B: $A1'B1 + A0'B1B0 + A1'A0'B0$

: $A1'B1 + (A1 \odot B1) A0'B0$

Comparing to the truth table of a 2-bit comparator, a 4-bit comparator will be used 4-bit in input A and 4-bit in input B. Therefore, the truth table of the 4-bit comparator is the following table below:



COMPARING INPUTS				OUTPUT		
A3, B3	**A2, B2**	**A1, B1**	**A0, B0**	**A > B**	**A < B**	**A = B**
A3 > B3	X	X	X	H	L	L
A3 < B3	X	X	X	L	H	L
A3 = B3	A2 >B2	X	X	H	L	L
A3 = B3	A2 < B2	X	X	L	H	L
A3 = B3	A2 = B2	A1 > B1	X	H	L	L
A3 = B3	A2 = B2	A1 < B1	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 > B0	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 < B0	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	H
H = High Voltage Level, L = Low Voltage, Level, X = Don't Care						

Figure 18: The true table of 4-bit comparator

In a 4-bit comparator the condition of A>B can be possible in the following four cases:

1. If A3 = 1 and B3 = 0
2. If A3 = B3 and A2 = 1 and B2 = 0
3. If A3 = B3, A2 = B2 and A1 = 1 and B1 = 0
4. If A3 = B3, A2 = B2, A1 = B1 and A0 = 1 and B0 = 0

Similarly the condition for A<B can be possible in the following four cases:

1. If A3 = 0 and B3 = 1
2. If A3 = B3 and A2 = 0 and B2 = 1
3. If A3 = B3, A2 = B2 and A1 = 0 and B1 = 1
4. If A3 = B3, A2 = B2, A1 = B1 and A0 = 0 and B0 = 1

The condition of A=B is possible only when all the individual bits of one number exactly coincide with corresponding bits of another number.

From the above statements logical expressions for each output can be expressed as follows:

$$A>B = A3B3' + (A3\odot B3)A2B2' + (A3\odot B3)(A2\odot B2)A1B1' \\ (A3\odot B3)(A2\odot B2)(A1\odot B1)A0B0'$$

$$A=B=(A3\odot B3)(A2\odot B2)(A1\odot B1)(A0\odot B0)$$

$$A<B = A3'B3 + (A3\odot B3)A2'B2 + (A3\odot B3)(A2\odot B2)A1'B1 + \\ (A3\odot B3)(A2\odot B2)(A1\odot B1)A0'B0$$

Conceptualize the design of 4-bit comparator from 2-bit comparators (the 2-bit comparator can be partitioned into smaller blocks). Draw a block diagram that describes the idea.

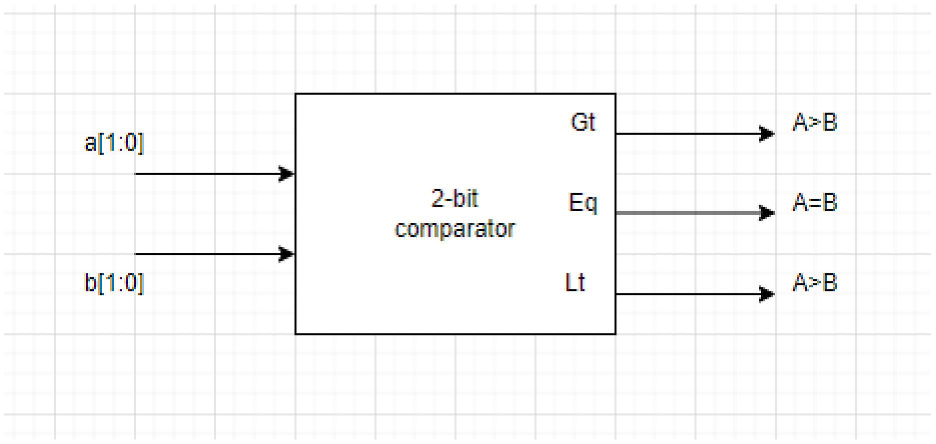


Figure 19: Block diagram of 2-bit comparator

We have a block diagram 2-bit comparator, thanks to it we can describe 4-bit comparator as follow:

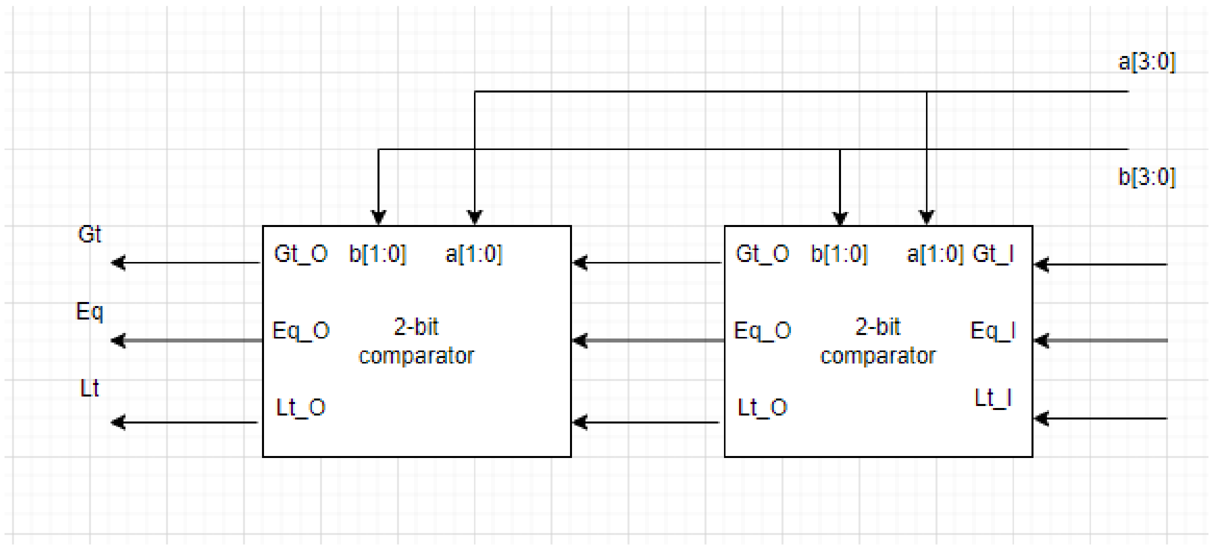


Figure 20: Block diagram of 4-bit comparator

Draw a diagram to shows the designed circuit hierarchy.

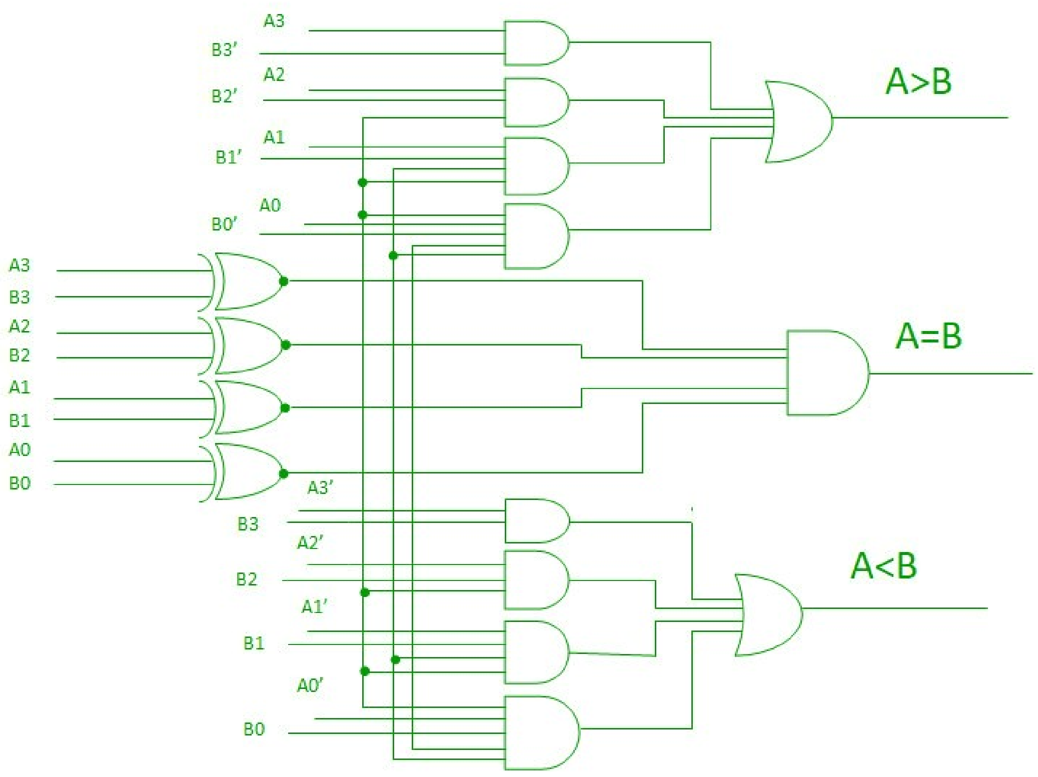


Figure 21: Circuit hierarchy of 4-bit comparator

Implement the designed circuit using Verilog HDL structural model.

We have structural model file code of 2-bit comparator is **comparator_2bit.v**, and then we also have file code of 4-bit comparator is **comparator_4bit.v**

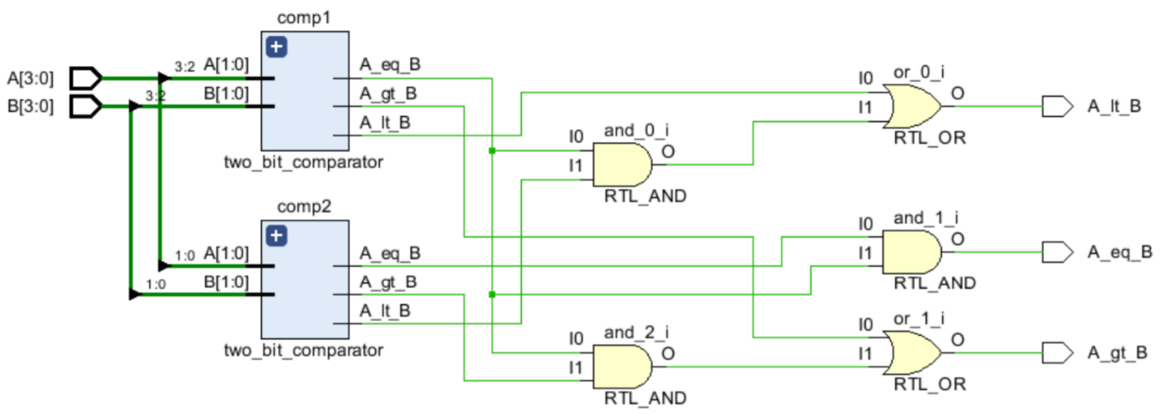


Figure 22: RTL schemetic of 4-bit comparator

Write a test bench to simulate the implemented circuit.

We have file code is **comparator_4bit_tb.v**

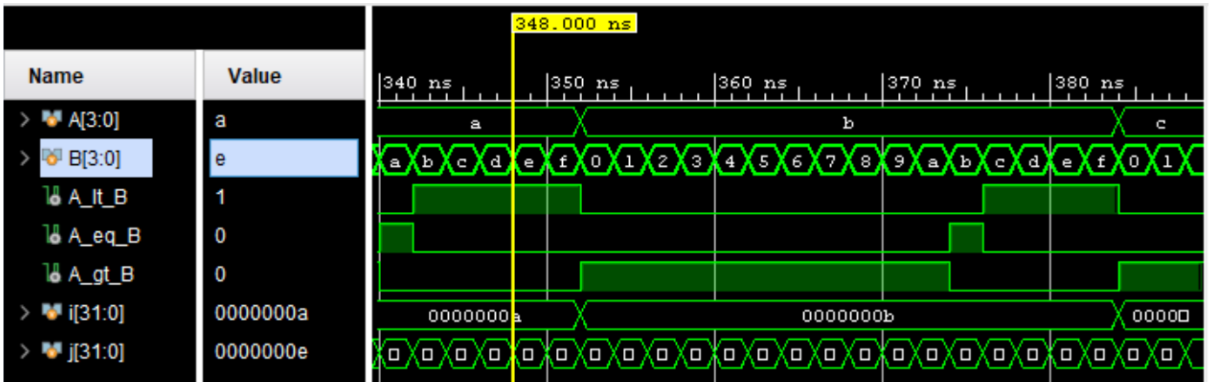
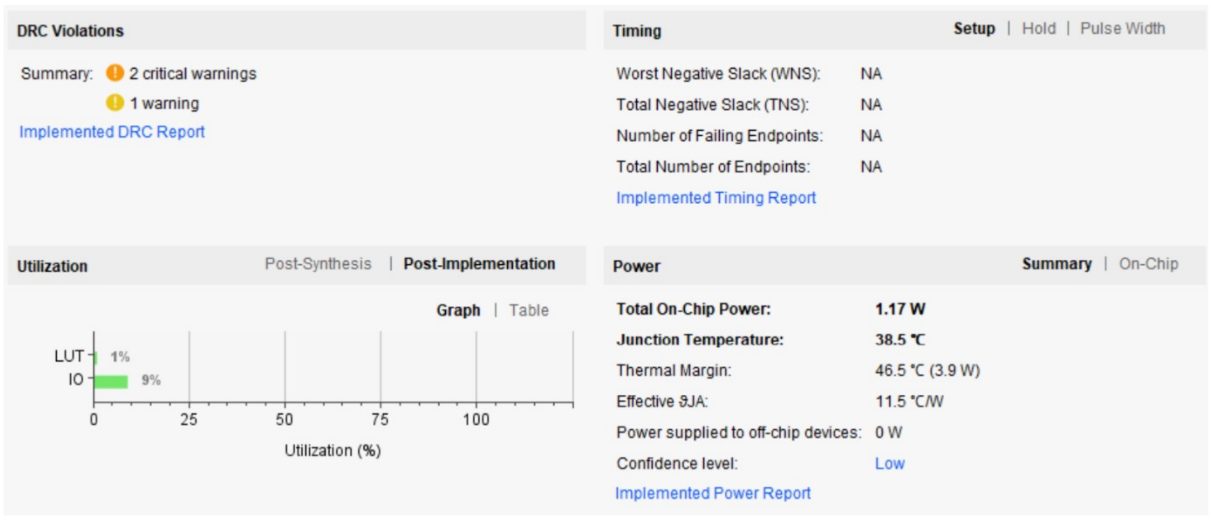


Figure 23: The waveform of 4-bit comparator



DRC Violations

Summary: 2 critical warnings

1 warning

Implemented DRC Report

Timing — **Setup** | Hold | Pulse Width

Worst Negative Slack (WNS):	NA
Total Negative Slack (TNS):	NA
Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA

Implemented Timing Report

Utilization — Post-Synthesis | **Post-Implementation**

Graph | Table

Power — **Summary** | On-Chip

Total On-Chip Power:	**1.17 W**
Junction Temperature:	**38.5 ℃**
Thermal Margin:	46.5 ℃ (3.9 W)
Effective ϑJA:	11.5 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Figure 24: The utilization report