VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

LAB_1 REPORT

CLASS CC01 – GROUP 1

SUBJECT:

LOGIC DESIGN

Instructor: Nguyễn Thiên Ân

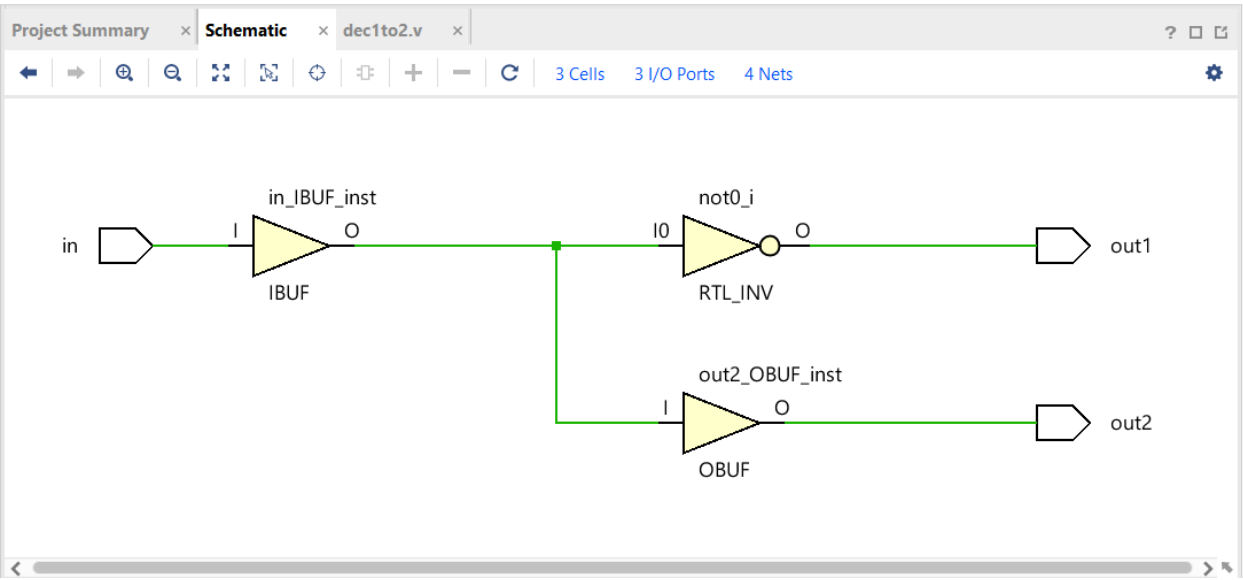
Student	ID
Bùi Thái An	2252001
Phạm Nguyễn Hải Khánh	2252333
Nguyễn Đăng Duy	2252116

2 Exercises

2.1 Exercise 1

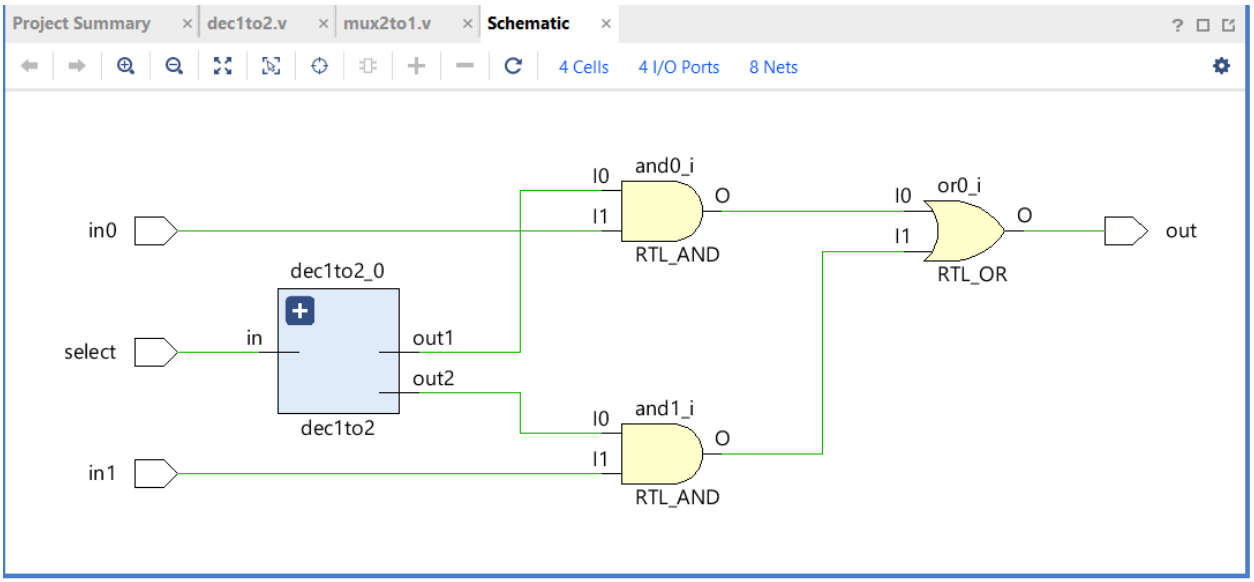
a. Design a 1-to-2 decoder using structure model as the following circuit:

```
module dec1to2(out1, out2, in);

input  wire in;

output wire out1;

output wire out2;

not not0(out1, in);

assign out2 = in;

endmodule
```



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

```
module mux2to1(out, in0, in1, select);

input  wire in0, in1;

input  wire select;

output wire out;

wire dec_out0, dec_out1;

wire and_out0, and_out1;

dec1to2 dec1to2_0(dec_out0, dec_out1, select);

and and0(and_out0, dec_out0, in0);

and and1(and_out1, dec_out1, in1);

or  or0 (out, and_out0, and_out1);

endmodule
```

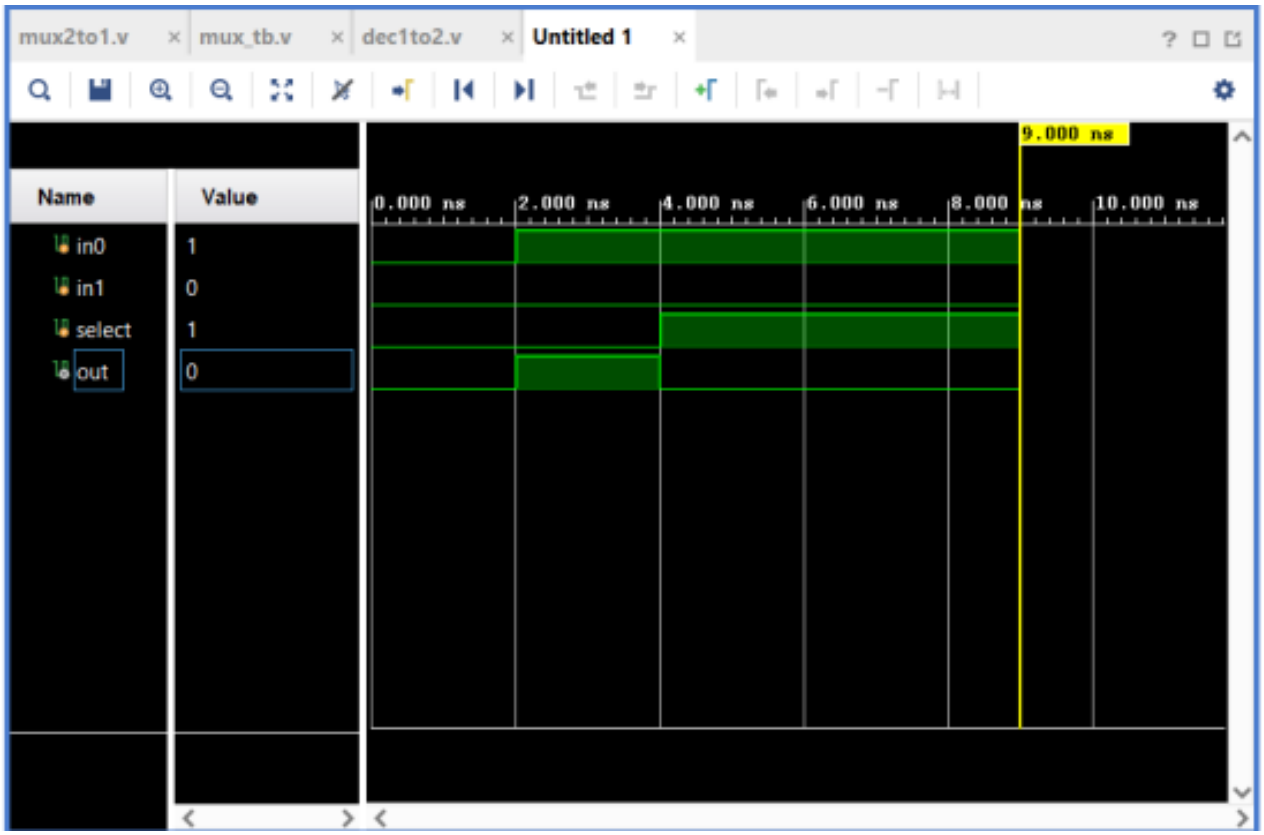


2.2 Exercise 2

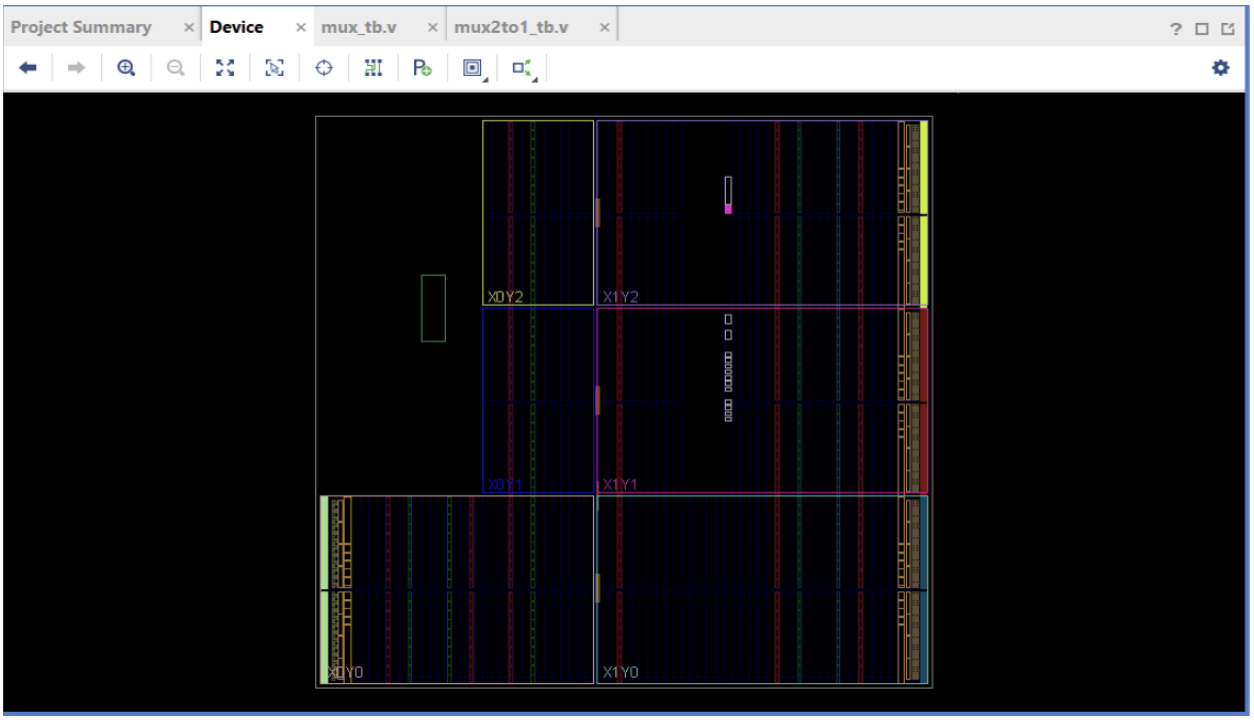
a. Write a test bench for the 2-to-1 Multiplexer in Exercise 1 then use Vivado Simulator to simulate the design, students can use the given example source code. Let's analyze the structure of a test bench then point out the differences between an RTL code and a test bench code. Change the Radix, Format of signals and use zoom tool to evaluate the waveform. Check the Tcl console window to see output of $monitor command:

```
`timescale 1 ns/10 ps

module mux2to1_tb;

reg in0, in1, select;

wire out;

    mux2to1 uut(out, in0, in1, select);

    initial begin

    $monitor("%t : Input_0: %b Input_1: %b Select: %b  -- Output: %b", $time, in0, in1, select, out);

end

initial begin

    in0 <= 1'b1;

    in1 <= 1'b0;

    select <= 1'b0;

    #5

    select <= 1'b1;
```

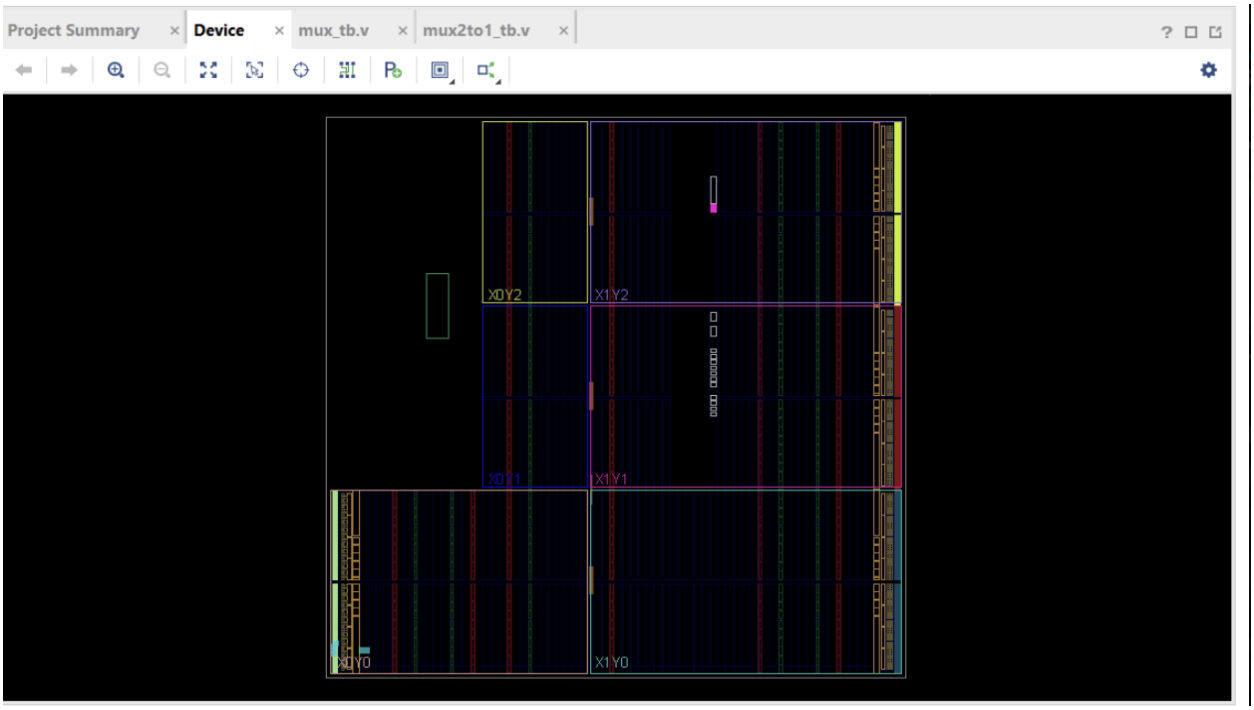
```
        #5
        in1 <= 1'b1;
        #5
        in1 <= 1'b0;
        #5
        in0 <= 1'b0;
        #5
        select <= 1'b0;
        #5;
        end
endmodule
```



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



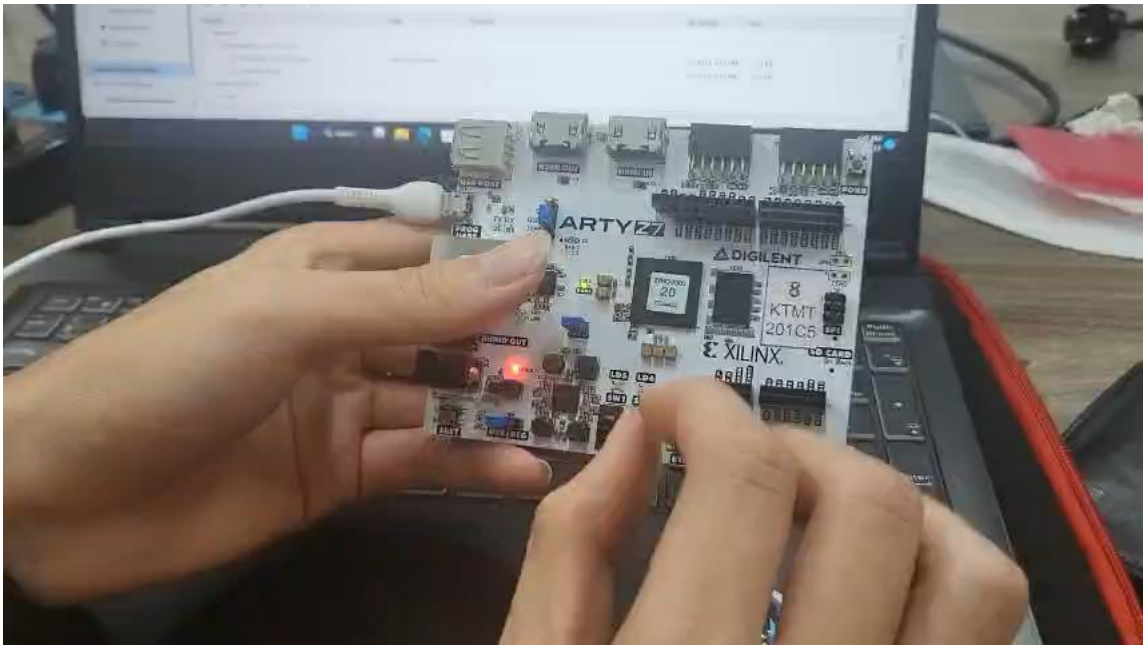
c. After that, run the Implementation, check the Utilization report in Project Summary for used resources.



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

- ***in1: btn[0], in2: btn[1]***
- ***sel: sw[0]***
- ***out: led[0]***

then, generate bitstream file and program the FPGA to test the implemented circuit on board:



2.3 Exercise 3

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

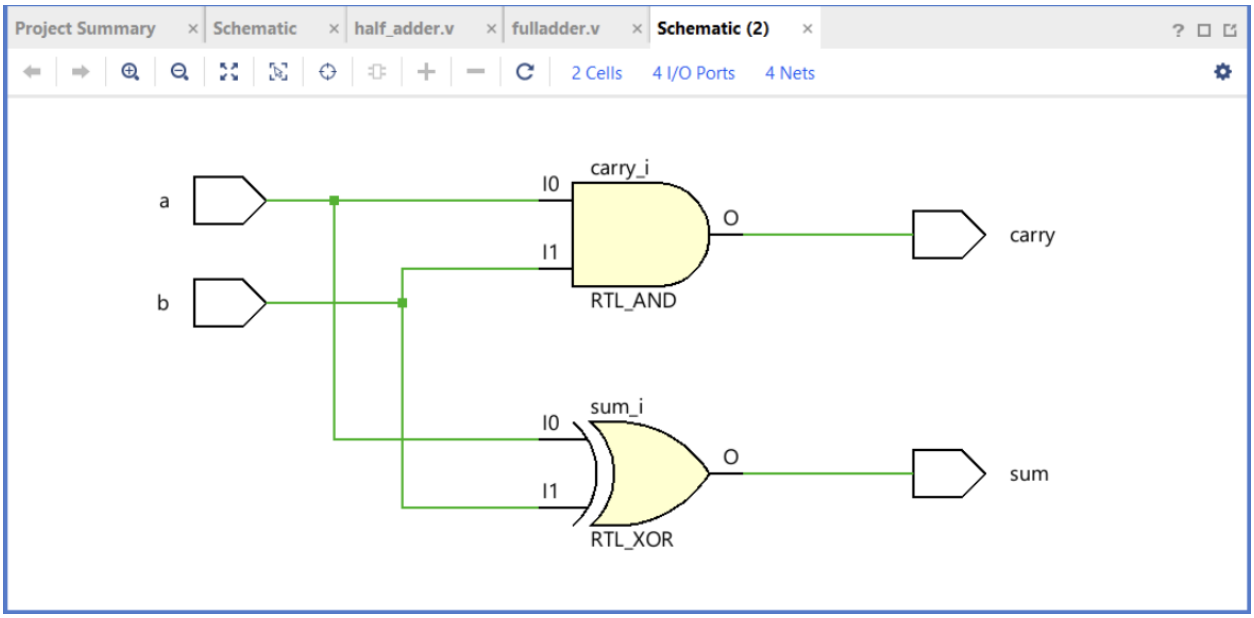
```
`timescale 1ns / 1ps

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

input a,b;
```

output sum,carry;

xor (sum,a,b);

and (carry,a,b);

endmodule



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

`timescale 1ns / 1ps

module fulladder(a,b,c_in,c_out,sum);

input a,b,c_in;

output c_out, sum;

wire sumha1,carryha1;

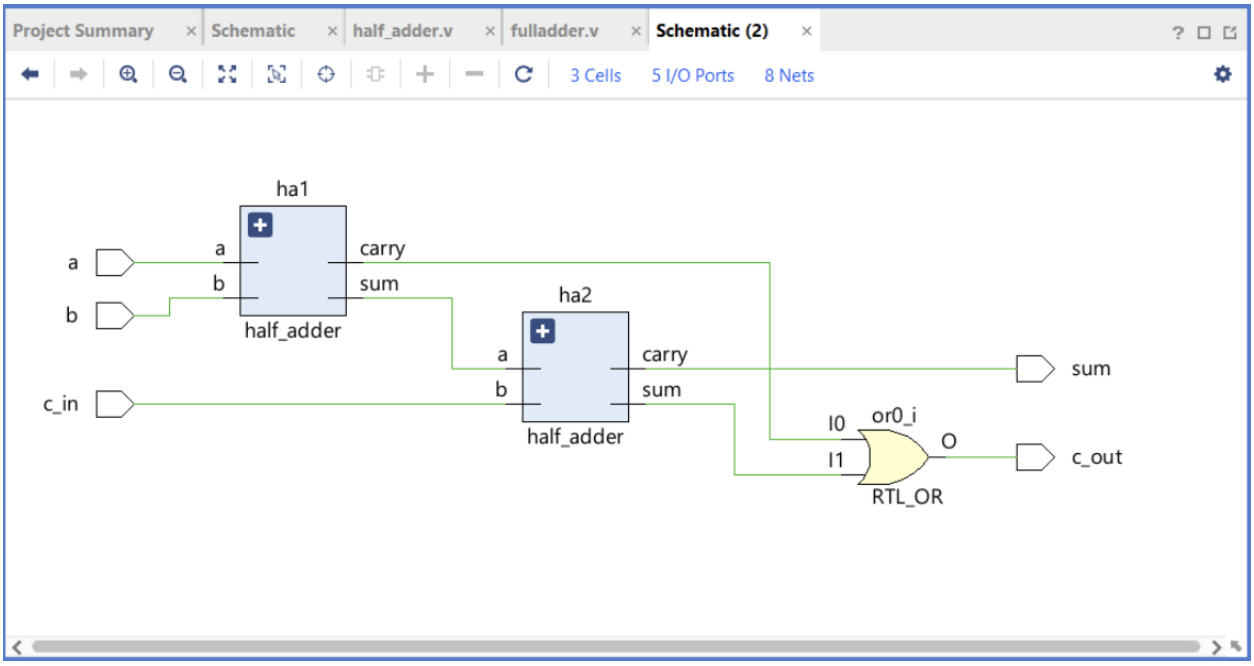
wire carryha2;

half_adder ha1(a,b,sumha1,carryha1);

half_adder ha2(sumha1,c_in,sum,carryha2);

or or0(c_out,carryha1,carryha2);

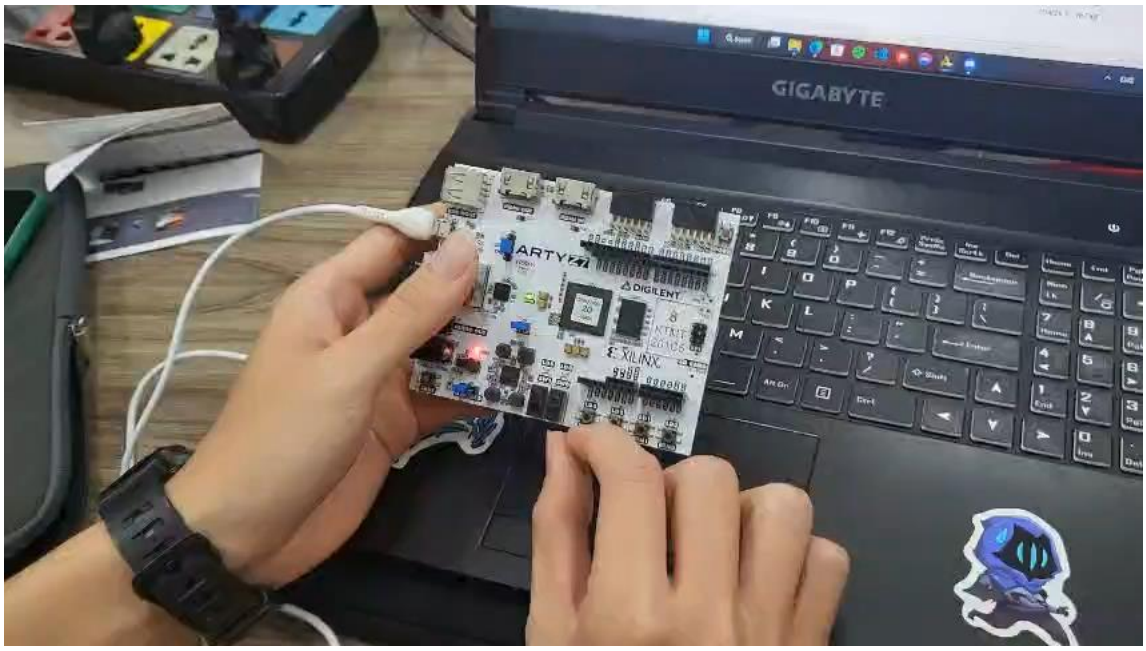
endmodule



TEST BENCH:

```
`timescale 1ns / 1ps
module fulladd_tb;
reg a,b,c_in;
wire c_out,sum;
fulladder fa1(a,b,c_in,c_out,sum);
initial begin
 a = 1'b0;
 b = 1'b0;
 c_in=1'b0;
 #2
 a = 1'b1;
 #2
 b = 1'b1;
 #2
```

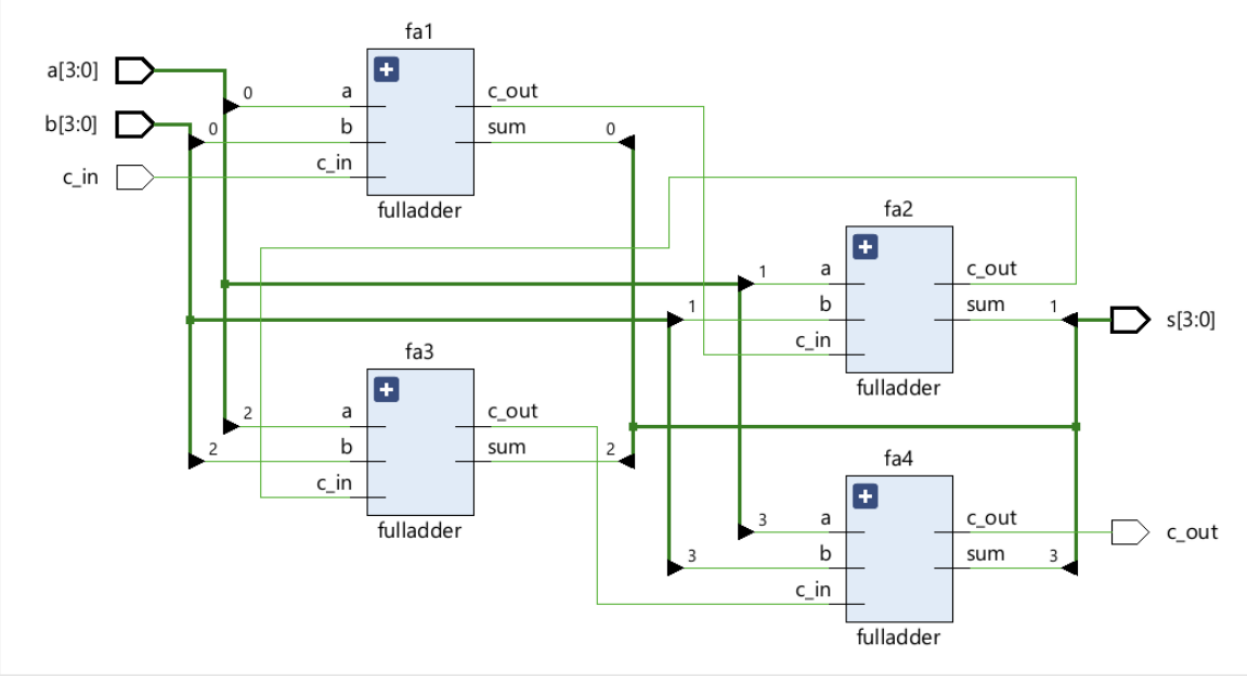
```
 c_in = 1'b1;
 #3 $stop;
 end
endmodule
```



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:

```
`timescale 1ns / 1ps
module bit_ripple_carry(a,b,c_in,s,c_out);
input [3:0] a,b;
input c_in;
wire [3:1] c_wire;
output [3:0] s;
output c_out;
fulladder fa1(a[0],b[0],c_in,c_wire[1],s[0]);
fulladder fa2(a[1],b[1],c_wire[1],c_wire[2],s[1]);
fulladder fa3(a[2],b[2],c_wire[2],c_wire[3],s[2]);
```

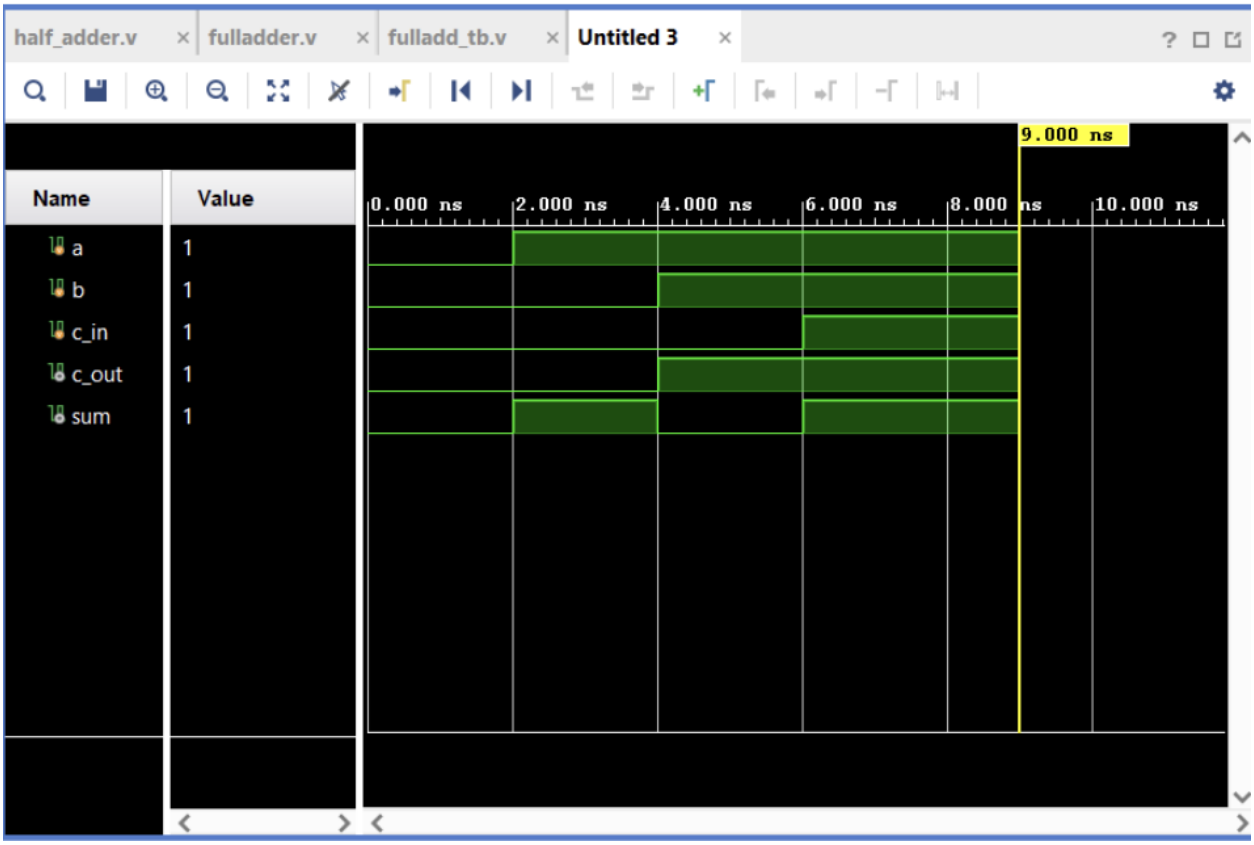
fulladder fa4(a[3],b[3],c_wire[3],c_out,s[3]);
endmodule



TEST BENCH:

```
`timescale 1ns / 1ps
module rippleadd_tb;
reg [3:0] a,b;
reg c_in;
wire [3:0] s;
wire c_out;
bit_ripple_carry test(a,b,c_in,s,c_out);
initial begin
a = 4'b1010;
b = 4'b1001;
c_in = 1'b0;
#2
a= 4'b0110;
b= 4'b0101;

#2
a = 4'b0010;
b = 4'b1000;
```

```
c_in = 1'b1;
#2
a = 4'b1100;
b = 4'b1000;

#5
$stop;
end
endmodule
```

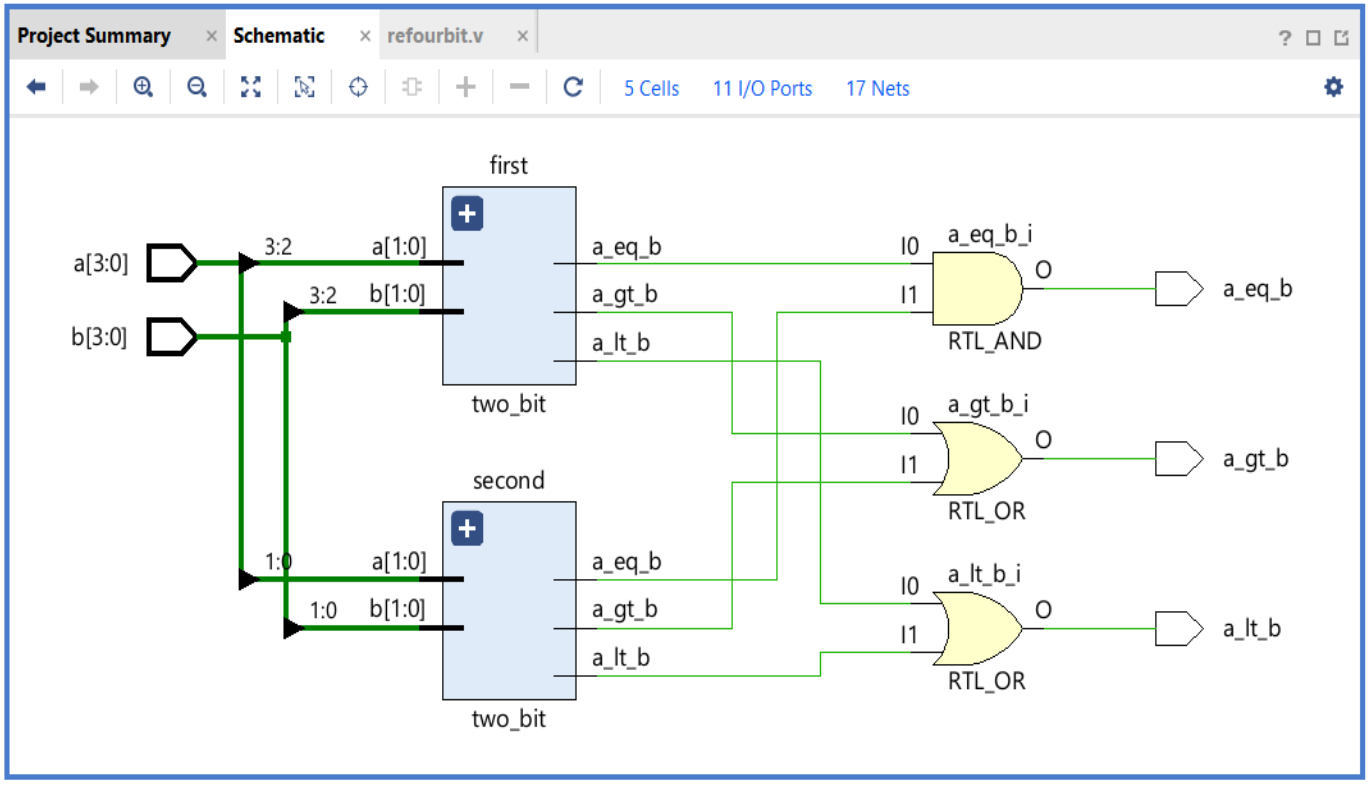


2.4 Exercise 4

Give the 2-bit comparator circuit as Figure 6 with A = {A1, A0} and B = {B1, B0} are 2 2-bit input numbers, A gt B is active if A > B, A lt B is active if A < B and A eq B is active if A = B:

```
`timescale 1ns / 1ps
module twobit_tb;
reg [1:0] a,b;
```

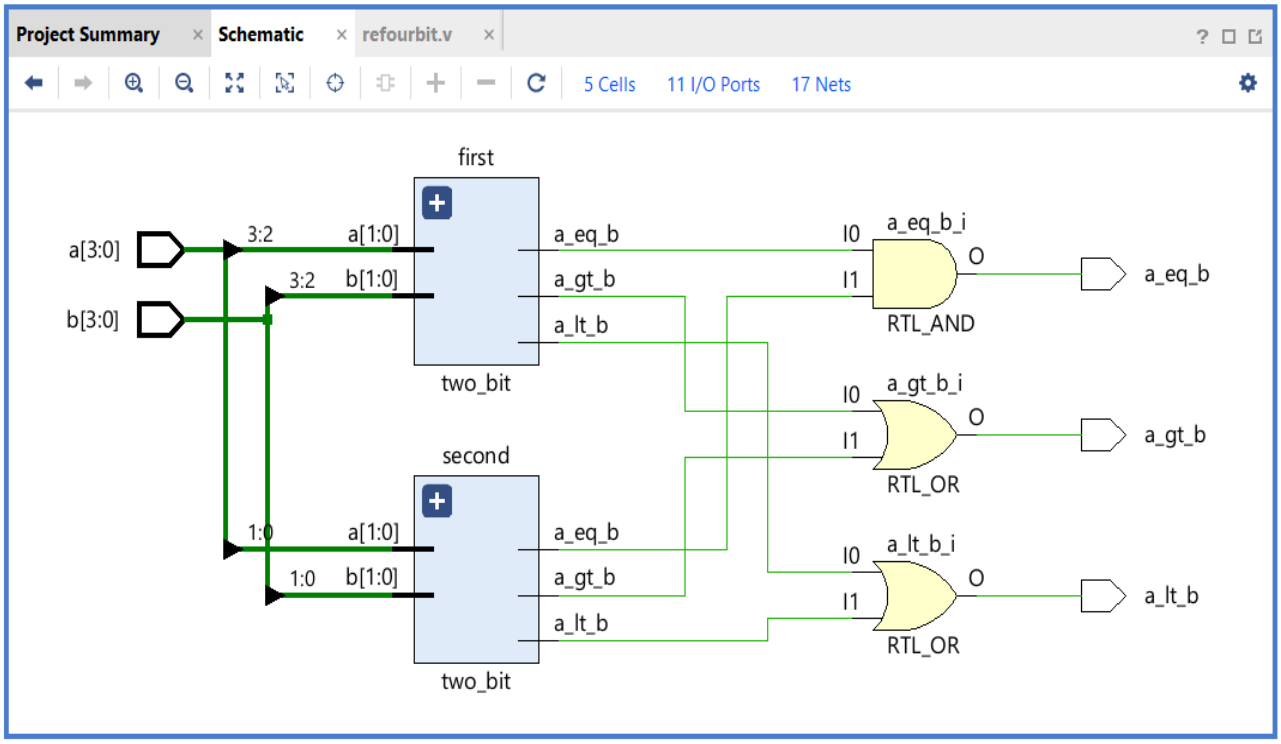
```
wire agtb,aeqb,altb;
two_bit compare(a,b,altb,aeqb,agtb);
initial begin
a = 2'b11;
b = 2'b11;
#2
a = 2'b10;
#2
b = 2'b10;
#2
a = 2'b11;
#2
$stop;
end
endmodule
```



Let's design a 4-bit comparator following below steps:

- ***Analyse the functions of each output of the 2-bit comparator, then determine the functions of 4-bit comparator outputs.***
- ***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.***
- ***Draw a diagram to shows the designed circuit hierarchy.***
- ***Implement the designed circuit using Verilog HDL structural model.***
- ***Write a test bench to simulate the implemented circuit.***

```
`timescale 1ns / 1ps
module refourbit_tb;
reg [3:0] a,b;
wire a_lt_b,a_eq_b,a_gt_b;
refourbit test(a,b,a_lt_b,a_eq_b,a_gt_b);
initial begin
a = 4'b1111;
b = 4'b1111;
#2
a = 4'b1110;
#2
b = 4'b1000;
#2
a = 4'b1000;
#5
$stop;
end
endmodule
```


Project Summary
Schematic
refourbit.v
5 Cells
11 I/O Ports
17 Nets
first
a[3:0]
b[3:0]
3:2
a[1:0]
3:2
b[1:0]
a_eq_b
a_gt_b
a_lt_b
two_bit
second
1:0
a[1:0]
1:0
b[1:0]
a_eq_b
a_gt_b
a_lt_b
two_bit
I0
I1
a_eq_b_i
O
RTL_AND
a_eq_b
I0
I1
a_gt_b_i
O
RTL_OR
a_gt_b
I0
I1
a_lt_b_i
O
RTL_OR
a_lt_b