

# Digital Logic Experiment Report (2)

|  |  |  |
| --- | --- | --- |
| **Digital Logic Experiment 2** | | |
| **1. Verilog HDL design digital logic circuit 50%** | **2. Verilog HDL design is 50% more complex digital logic circuits** | **Overall result** |
|  |  |  |

评语：（包含：预习报告内容、实验过程、实验结果及分析）

教师签名：

**Name: Chen Haorui**

**student ID: U201612696**

**class: School cross 1601**

**refer to guide teach Teacher: Xu Youqing**

**School of Computer Science and Technology**

**June 14 , 2018 \_ \_ \_**



**Digital Logic Lab Report**

Verilog HDL design digital logic circuit preview report

1. Verilog HDL design digital logic circuit

1. Experiment name

Verilog HDL designs digital logic circuits .

2. Purpose of the experiment

Require students to use Verilog HDL to design digital logic circuits , pass 3 logic circuit experiments , and use " Vivado 2015.2 " software to perform "before and after " simulation to check the circuit design , and then operate and record the experimental results on the " Xilinx NEXYS 4 Development Board " , and finally verify whether the design meets the requirements.

Through the above three training processes of design , simulation, and verification, students can master the basic methods of designing digital logic circuits in Verilog HDL, and at the same time master how to avoid the generation of latches and how to deal with consistency problems in circuit design .

3. Equipment used in the experiment

Xilinx NEXYS 4 development board (chip is XC7A100TCSG324-1 , package is CSG324 , software is Vivado 2015.2 ) 1 set.

4. Experimental content

**(1) "always" design of combinational and sequential logic circuits**

**Design pure combinational logic circuits with " always blocks "**

A property of a combinational circuit is that its output is always affected by changes in the input. This means that combinational circuits never retain their previous values, i.e. outputs do not latch.

When using the case, if-else and other statements in the always block to design pure combinational logic circuits , it is necessary to ensure that all input conditions have output values , otherwise latches may be generated , resulting in "synthesis" errors.

**For example :** a topic requires the use of Verilog to design a selector for a pure combinational logic circuit . A classmate designed a 4-to-1 multiplexer with the " fla a g" logo, as shown in Program 1-1 , but in When " integrating " , three errors are reported, namely: "Place 30-574 , Place 30-99, Common 17-69 " .

Specific requirements:

( a ) Verify whether the above problems will occur when program 1-1 is " synthesized " ;

( b ) If the above problems exist, please correct Program 1 -1 and help this student complete the design.

Program 1-1 4 to 1 multiplexer with "fla a g" logo

module mux\_latch ( \_

input [ 3:0] data,

input [ 1:0] valid,

input flag,

output reg valid\_data );

initial begin

valid\_data = 1'b0;

end

always @ (\*)

begin

case(valid)

2'b 00 : begin if(flag) valid\_data = data[0];end

2'b 01 : begin if(flag) valid\_data = data[1];end

2'b 10 : begin if(flag) valid\_data = data[2];end

2'b 11 : begin if(flag) valid\_data = data[3];end

end case

end

endmodule

////////////////////////////\*.xdc files ///////////////////// \_ ///////

set\_property PACKAGE\_PIN T16 [ get\_ports flag]

set\_property IOSTANDARD LVCMOS33 [ get\_ports flag]

set\_property PACKAGE\_PIN U8 [ get\_ports { valid[ 0] }]

set\_property IOSTANDARD LVCMOS33 [ get\_ports { valid[ 0] }]

set\_property PACKAGE\_PIN R7 [ get\_ports { valid[ 1] }]

set\_property IOSTANDARD LVCMOS33 [ get\_ports { valid[ 1] }]

set\_property PACKAGE\_PIN V7 [ get\_ports { data[ 0] }]

set\_property IOSTANDARD LVCMOS33 [ get\_ports { data[ 0] }]

set\_property PACKAGE\_PIN V6 [ get\_ports { data[ 1] }]

set\_property IOSTANDARD LVCMOS33 [ get\_ports { data[ 1] }]

set\_property PACKAGE\_PIN V5 [ get\_ports { data[ 2] }]

set\_property IOSTANDARD LVCMOS33 [ get\_ports { data[ 2] }]

set\_property PACKAGE\_PIN U4 [ get\_ports { data[ 3] }]

set\_property IOSTANDARD LVCMOS33 [ get\_ports { data[ 3] }]

set\_property PACKAGE\_PIN T8 [ get\_ports valid\_data ]

set\_property IOSTANDARD LVCMOS33 [ get\_ports valid\_data ]

**( B ) Design a synchronous sequential logic circuit with " always block "**

See the circuit shown in Figure 1-1 .



Figure 1-1 Synchronous sequential logic circuit

Specific requirements:

The circuit is designed and simulated using the " always block " of Verilog HDL .

**( Note : The difference between blocking and non-blocking assignments , this combination and timing hybrid circuit recommends using non-blocking assignments )**

**(2) Design of pulse asynchronous counter**

Analyze the pulse asynchronous counter circuit shown in Figure 1-2 , and complete the following:

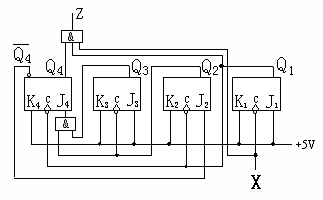


Figure 1-2 pulse asynchronous counter circuit

( A ) What is the modulus of the counter? Answer: The modulo of the counter is 1 0 .

( B ) Implement the circuit with Verilog DHL , and verify the design by simulation and on the development board .

**Consistency issues in Verilog digital circuit design**

Analyze the circuit shown in Figure 1-3 and complete the following :

( A ) Program 1-2 is a description of the circuit shown in Figure 1-3 , please use " **Behavior Simulation-it can be called pre-simulation** " and " **Non-Behavior Simulation-it can be called post** - **simulation** " to program 1-2 Carry out the simulation separately , if there is an error, please correct the program 1-2 , and give the corrected simulation result .

(This is : the question of whether the so-called pre- simulation and post-simulation are consistent )

q1

q2

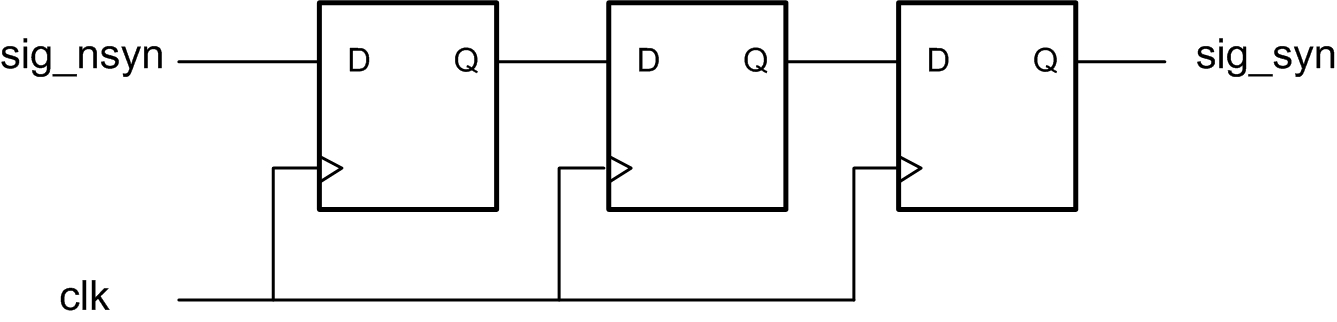


Figure 1-3 3- bit shift register circuit

( B ) The NEXYS 4 development board provides a 100 Mhz synchronous clock , the pin is bound as "E3" , add a clock frequency division part in the corrected program 1-2 , and connect the clock signal after the frequency reduction Go to " clk " in Figure 1-3 , download it to the development board and test it after the compilation is successful .

Conclusion : In the future design, it is necessary to maintain : "The pre-simulation and post-simulation and download verification are all correct".

Program 1-2 3- bit shift register circuit

module pipe3 b(

input sig\_nsyn ,

input clk ,

output q1,

output q2,

output sig\_syn

);

reg q1;

reg q2;

reg sig\_syn ;

always @( posedge clk ) q2=q1;

always @( posedge clk ) sig\_syn =q2;

always @( posedge clk ) q1 = sig\_nsyn ;

endmodule

5. Experimental program design

**(1) The "always" design scheme of combinational and sequential logic circuits**

(A) Design pure combinational logic circuits with " always blocks "

`timescale 1ns / 1ps

module mux\_latch ( \_

input [ 3:0] data,

input [ 1:0] valid,

input flag,

output reg valid\_data

);

initial valid\_data = 1'b0;

always @ (\*)

begin

case(valid)

2'b 00 : if(flag) begin if(flag) valid\_data = data[0]; else valid\_data =0; end

2'b 01 : if(flag) begin if(flag) valid\_data = data[1];end else valid\_data =0;end

2'b 10 : if(flag) begin if(flag) valid\_data = data[2];end else valid\_data =0;end

2'b 11 : if(flag) begin if(flag) valid\_data = data[3];end else valid\_data =0;end

end case

end

endmodule

( B ) Implementation of a synchronous sequential logic circuit with an " always block " design

Note : The waveform situation when x=0 was not given during the inspection , and it has been corrected now.

(a) source program

`timescale 1ns / 1ps

module a\_b ( input x,CP output reg y,Q0,W1,Q1);

initial

begin

Q0<=1'b0;

Q1<=1'b1;

W1=1'b0;

y=1'b0;

end

always @( posedge CP)

begin

Q0 <= ~Q0;

W1 <= Q0^x;

case (W1)

1'b0: Q1<=Q1;

1'b1: Q1<=~Q1;

endcase ;

y <= x & (~Q1);

end

endmodule

(b) Simulation program

`timescale 1ns / 1ps

module a\_B\_sim ;

reg X ;

wire Y ,q1,w1,q0;

reg clk = 0;

initial

begin X <= 1; # 500 X<=0; end

a\_b \_ real\_a\_B (.x(X),.CP ( clk ),.Q0(q0),.W1(w1),.Q1(q1),.y(Y));

always #50 clk = ~ clk ;

endmodule

**(2) Analysis and design scheme of pulse asynchronous counter**

(A) source program

`timescale 1ns / 1ps

module b( input in, output out,q1,q2,q3,q4);

wire j1, k1, ck1, j2, k2, ck2, j3, k3, ck3, j4, k4, ck4, nq4;

assign j1 = 1; assign k1 = 1; assign ck1 = in;

assign j2 = nq 4; assign k2 = 1; assign ck2 = q1;

assign j3 = 1; assign k3 = 1; assign ck3 = q2;

assign j4 = q2 & q 3; assign k4 = 1; assign ck4 = q1;

assign out = in & q1 & q4;

jk uut1 ( .clk (ck1),.j(j1),.k(k1),.q(q1),. qb (nq1));

jk uut2 ( .clk (ck2),.j(j2),.k(k2),.q(q2),. qb (nq2));

jk uut3 ( .clk (ck3),.j(j3),.k(k3),.q(q3),. qb (nq3));

jk uut4 ( .clk (ck4),.j(j4),.k(k4),.q(q4),. qb (nq4));

endmodule

module jk ( input clk,j,k , output q,qb );

reg q;

assign qb =~q;

initial q=1;

always @( negedge clk )

begin

case({ j,k })

2'b 00:q <=q; 2'b01:q<=0;

2'b 10:q <=1; 2'b11:q<=~q;

end case

end

endmodule

(B) Simulation program

`timescale 1ns / 1ps

module b\_sim ;

reg X; wire Z,q 1,q2,q3,q4;

b uut (.in(X),.out(Z) ,.q 1(q1),.q2(q2),.q3(q3),.q4(q4));

initial X = 0;

always #10 X=~X;

endmodule

( C ) Pin Constraint (Binding) Procedure

###############################

# On-board Slide Switches #

###############################

set\_property PACKAGE\_PIN U9 [ get\_ports x]

set\_property IOSTANDARD LVCMOS33 [ get\_ports x]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [ get\_nets x\_IBUF ]

###############################

# On-board led #

###############################

set\_property PACKAGE\_PIN T5 [ get\_ports z]

set\_property IOSTANDARD LVCMOS33 [ get\_ports z]

set\_property PACKAGE\_PIN T8 [ get\_ports q1]

set\_property IOSTANDARD LVCMOS33 [ get\_ports q1]

set\_property PACKAGE\_PIN V9 [ get\_ports q2]

set\_property IOSTANDARD LVCMOS33 [ get\_ports q2]

set\_property PACKAGE\_PIN R8 [ get\_ports q3]

set\_property IOSTANDARD LVCMOS33 [ get\_ports q3]

set\_property PACKAGE\_PIN T6 [ get\_ports q4]

set\_property IOSTANDARD LVCMOS33 [ get\_ports q4]

**(3) Solutions to consistency problems in Verilog design**

(A) source program

`timescale 1ns / 1ps

module c( input sig\_nsyn,cp,output q1,q2,sig\_syn);

wire clk ;

reg q 1, q 2, sig\_syn;

div dv1( cp,clk );

initial

begin q1=0; q2 =0; sig\_syn =0; end

always @( posedge clk ) q1<= sig\_nsyn ;

always @( posedge clk ) q2<=q1;

always @( posedge clk ) sig\_syn <=q2;

endmodule

module div( input cp, output reg clk );

parameter t=10000000;

integer count;

initial begin count <= 0; clk <= 0; end

always @( posedge cp ) begin

count = count + 1;

if( count == t)

begin clk = ~ clk ; count = 0; end

end

endmodule

(B) Simulation program

`timescale 1ns / 1ps

module c\_fz ;

reg IN,CP ;

wire Q 1, Q 2, OUT;

initial

begin IN <=1; CP<=0; end

always # 30000000 IN=~IN;

always #4 CP =~CP;

c uut (.cp ( CP) ,. sig\_nsyn (IN),.q1(Q1),.q2(Q2),. sig\_syn (OUT ) );

endmodule



**Digital Logic Lab Report**

Verilog HDL design more complex digital logic circuit

2. Verilog HDL design is more complex digital logic circuit

1. Experiment name

Verilog HDL designs more complex digital logic circuits.

2. Purpose of the experiment

Students are required to use Verilog HDL to design more complex digital logic circuits , pass 3 logic circuit experiments , and use " Vivado 2015.2 " software to perform "before and after " simulation to check the circuit design , and then operate on the " Xilinx NEXYS 4 Development Board " , Record the experimental results , and finally verify whether the design meets the requirements .

Through the above three training processes of design , simulation, and verification, students can master the basic methods of designing complex digital logic circuits in Verilog HDL , and at the same time master the use of " circuit instantiation", " modularization ", and synchronous processing of asynchronous sequential logic circuits And design the control circuit with the state machine.

3. Components used in the experiment

Xilinx NEXYS 4 development board (chip is XC7A100TCSG324-1 , package is **CSG324** , software is Vivado 2015.2 ) 1 set.

4. Experimental content

**(1) Design of 4 -bit binary addition / subtraction counter**

4 -bit binary counter that adds 1/ subtracts 1 and can clear, set, and carry / borrow output . Its structural block diagram is shown in Figure 2-1 .

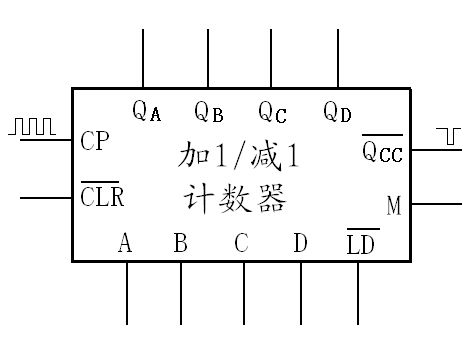


Figure 2-1 4 -bit binary addition / subtraction counter

The circuit input is counting pulse CP , working mode selection M , prefabricated initial value D , C , B , A (where D is high, A is low) and prefabricated control , clearing terminal ;

The output is the count value Q D , Q C , Q B , Q A ( Q D is the high bit, Q A is the low bit) and carry / borrow output ;

When it is 0 , the circuit output is cleared;

When prefabrication control =0 , send the input values of D , C , B , A to the counter, and immediately output them in Q D , Q C , Q B , Q A ;

the mode selection terminal M=1, add 1 to count;

When M=0 , count down by 1 ;

When the CP terminal inputs a rising edge signal, a count is performed;

The terminal outputs a negative pulse when there is a carry / borrow in the count.

**Note :** When designing circuits with Verilog , you often encounter such problems, such as:

( A ) Use two always blocks to assign a value to a register. No matter what conditions are judged, the final result is undoubtedly that two independent trigger signals are connected to the CLK terminal of the register, and one port is connected to Two signals, so such a statement cannot be synthesized into a circuit.

( B ) If a circuit has multiple inputs , it may cause changes in the output value . When using "always" , if the trigger condition is " multiple inputs" of the circuit , if the " concurrency " of the statement is not handled well , it will cause the system to compile successfully and the " behavior simulation" to be successful , but the system cannot generate the "bit" file.

Solution : Use the so-called " synchronization processing of asynchronous sequential logic circuits" , that is , reduce the "always" trigger conditions.

**Specific requirements:**

( A ) Implement the counter with Verilog HDL , and download it to the development board for verification;

( B ) Use the realized " 4 -bit binary counter", adopt " circuit instantiation" or " modularization " to realize a modulo 8 counter with an initial value of 2 , and download it to the development board for verification;

( C ) Give the resource situation of the FPGA chip used by the design (hope the less the better ).

**(2) Using a finite state machine (FSM) to implement a sequence detector**

Design a simple state machine, its function is to detect a serial 5 -bit non- overlapping "10110" binary sequence detector, when the input value appears "10110" , the output flag is given .

The specific requirements are as follows :

( A ) State transition diagram for non- overlapping "10110" binary sequences;

( B ) Use the finite state machine "standard template" to design the "10110" binary sequence detector, and download it to the development board for verification after the simulation is correct ;

( C ) Give the resource situation of the FPGA chip used by the design (hope the less the better ).

**(For the so-called finite state machine "standard template", please refer to the textbook : Xia Yuwen . Verilog Digital System Design Course 3rd Edition. Beijing: Beijing University of Aeronautics and Astronautics Press, 2013. )**

**Design of 3-bit binary numerical comparator**

Design a 3-bit binary numerical comparator. When A>B , F1=1 , F2=F3=0 ; when A=B , F2=1 , F1=F3=0 ; when A<B , F3=1 , F1=F2=0 .

Specific requirements:

( A ) Design a one-bit binary numerical comparator with Verilog HDL ;

( B ) Using the implemented one-bit binary value comparator, implement a 3-bit binary value comparator using "circuit instantiation" or " modularization ";

( C) Download the designed circuit to the development board for verification;

( D ) Give the resource situation of the FPGA chip used by the design (hope the less the better ).

5. Experimental program design

**(1) Design scheme of 4 -bit binary addition / subtraction counter**

(A) Modulo 16 plus 1/subtract 1 counter

(a) source program

`timescale 1ns / 1ps

module a( in,CP ,CLR,Ld,M,out,Qcc );

input[ 3:0] in;

input CP,CLR ,Ld,M ;

output[ 3:0] out;

output Qcc ;

reg [ 3:0] out;

reg Qcc ;

initial out=4'b0000;

initial Qcc =1;

always @( posedge CP or negedge Ld or negedge CLR)

begin

if (~CLR)

begin out<=4'b 0000; Qcc <=1; end

else if(~ Ld )

begin out<= in; Qcc <=1; end

else if((out==4'b 1111)& &M)

begin out<=4'b 0000; Qcc <=0; end

else if((out==4'b 0000)& &~M)

begin out<=4'b 1111; Qcc <=0; end

else if(~M)

begin out<=out-4'b 0001; Qcc <=1; end

else if(M)

begin out<=out+4'b 0001; Qcc <=1; end

end

endmodule

(b) Simulation program

`timescale 1ns / 1ps

module a\_sim ;

reg [3:0] in;

reg CP,CLR ,Ld,M ;

wire [3:0] out;

wire Qcc ;

a uut (.in(in),.CP (CP),.CLR(CLR),. Ld ( Ld ),.M(M),.out(out),. Qcc ( Qcc ));

always #10 CP=~CP;

initial begin

in = 4'b0010;

M=1;

CLR = 1;

CP = 0;

Ld = 0;

#50; Ld = 1;

#50; Ld = 0;

#50; Ld = 1;

# 500; CLR=0;

# 50; CLR=1;

# 1000; M=0;

# 1000; M=1;

end

endmodule

(c) Pin constraint (binding) procedure

###############################

# On-board Slide Switches #

###############################

set\_property PACKAGE\_PIN U9 [ get\_ports in[ 0]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports in[ 0]]

set\_property PACKAGE\_PIN U8 [ get\_ports in[ 1]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports in[ 1]]

set\_property PACKAGE\_PIN R7 [ get\_ports in[ 2]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports in[ 2]]

set\_property PACKAGE\_PIN R6 [ get\_ports in[ 3]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports in[ 3]]

set\_property PACKAGE\_PIN R5 [ get\_ports M]

set\_property IOSTANDARD LVCMOS33 [ get\_ports M]

set\_property PACKAGE\_PIN R10 [ get\_ports CP]

set\_property IOSTANDARD LVCMOS33 [ get\_ports CP]

set\_property PACKAGE\_PIN V6 [ get\_ports CLR]

set\_property IOSTANDARD LVCMOS33 [ get\_ports CLR]

set\_property PACKAGE\_PIN V5 [ get\_ports Ld ]

set\_property IOSTANDARD LVCMOS33 [ get\_ports Ld ]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [ get\_nets CP\_IBUF]

###############################

# On-board led #

###############################

set\_property PACKAGE\_PIN T8 [ get\_ports out[ 0]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports out[ 0]]

set\_property PACKAGE\_PIN V9 [ get\_ports out[ 1]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports out[ 1]]

set\_property PACKAGE\_PIN R8 [ get\_ports out[ 2]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports out[ 2]]

set\_property PACKAGE\_PIN T6 [ get\_ports out[ 3]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports out[ 3]]

set\_property PACKAGE\_PIN T5 [ get\_ports Qcc ]

set\_property IOSTANDARD LVCMOS33 [ get\_ports Qcc ]

**(B) The modulo 8 counter with an initial value of 2 requires that it be realized by instantiating A [the counter realized by calling (A)].**

(a) source program

`timescale 1ns / 1ps

module a\_ b ( Qcc ,out,M,CP );

input M,CP ;

output reg [ 3:0] out;

output reg Qcc ;

reg CLR;

reg Ld ;

wire [3:0]in;

wire [ 3:0]put ;

assign in=4'b1001;

initial

begin

out <= 4'b0010;

Qcc =1;

Ld <=1;

CLR<=1;

end

a rlz ( in,CP ,CLR,Ld,M,put,Qoc );

always @( posedge CP)

begin

CLR<=~(put[ 3]& (~put[2])&(~put[1])&put[0]&M);

#5 CLR<=1;

Ld <=~((~put[3 ])& (~put[2])&put[1]&(~put[0])&(~M));

#5 Ld <=1;

if((put==4'b0010& M)| (put==4'b1001&(~M))) Qcc <=0;

else Qcc <=1;

out=put;

end

endmodule

(b) Simulation program

`timescale 1ns / 1ps

module a\_b\_sim ;

reg M,CP ;

wire Qcc ;

wire [3:0] out;

a\_b uut ( .Qcc ( Qcc ), .out(out), .M(M), .CP(CP));

always #10 CP=~CP;

initial begin

M = 1;

CP = 0;

# 400; M=0;

# 400; M=1;

end

endmodule

(c) Pin constraint (binding) procedure

###############################

# On-board Slide Switches #

###############################

set\_property PACKAGE\_PIN R5 [ get\_ports M]

set\_property IOSTANDARD LVCMOS33 [ get\_ports M]

set\_property PACKAGE\_PIN R10 [ get\_ports CP]

set\_property IOSTANDARD LVCMOS33 [ get\_ports CP]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [ get\_nets CP\_IBUF]

###############################

# On-board led #

###############################

set\_property PACKAGE\_PIN T8 [ get\_ports out[ 0]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports out[ 0]]

set\_property PACKAGE\_PIN V9 [ get\_ports out[ 1]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports out[ 1]]

set\_property PACKAGE\_PIN R8 [ get\_ports out[ 2]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports out[ 2]]

set\_property PACKAGE\_PIN T6 [ get\_ports out[ 3]]

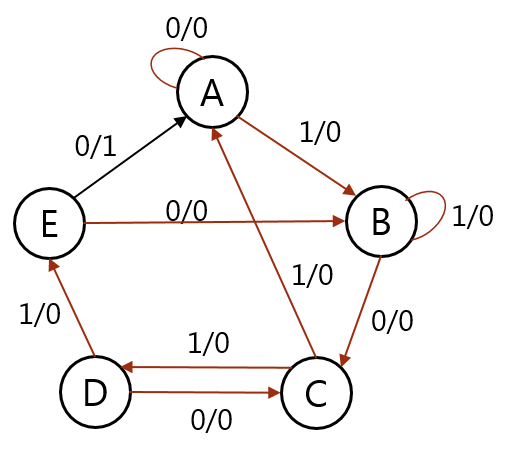
set\_property IOSTANDARD LVCMOS33 [ get\_ports out[ 3]]

set\_property PACKAGE\_PIN T5 [ get\_ports Qcc ]

set\_property IOSTANDARD LVCMOS33 [ get\_ports Qcc ]

**(2) The design scheme of sequence detector realized by finite state machine (FSM)**

(A) State diagram of a serial 5-bit non- overlapping "10110" binary sequence detector



( B ) source program

`timescale 1ns / 1ps

module b( input in, clk, output reg out , output reg [1:3] p,n );

parameter A= 0,B =1,C=2,D=3,E=4;

initial p=A;

always @( negedge clk ) p<=n;

always @( p or in)

begin

case(p)

A: begin out<=0; n <= in?B:A ; end

B: begin out<= 0; n<= in?B:C ; end

C: begin out<= 0; n<= in?D:A ; end

D: begin out<= 0; n<= in?E:C ; end

E: begin out<= in?0:1 ; n<= in?B:A ; end

end case

end

endmodule

(C) Simulation program

`timescale 1ns / 1ps

module b\_sim ;

reg in,clk ;

wire out;

wire p[ 1:3];

b mealy( in,clk ,out,p [1:3]);

always #10 clk = ~ clk ;

initial

begin

in <= 0;

clk = 1;

#12 in<= 1; #20 in<=0; #20 in<=1; #20 in<=1; #20 in<=0;

#20 in<= 1; #20 in<=0; #20 in<=1; #20 in<=1; #20 in<=1;

#20 in<= 0; #20 in<=1; #20 in<=1; #20 in<=0; #20 in<=1;

#20 in<= 0; #20 in<=1; #20 in<=0; #20 in<=1; #20 in<=1;

#20 in<= 0; #20 in<=1; #20 in<=1; #20 in<=0;

end

endmodule

( D ) Pin Constraint (Binding) Procedure

set\_property PACKAGE\_PIN R3 [ get\_ports clk ]

set\_property IOSTANDARD LVCMOS33 [ get\_ports clk ]

set\_property PACKAGE\_PIN P4 [ get\_ports in]

set\_property IOSTANDARD LVCMOS33 [ get\_ports in]

set\_property PACKAGE\_PIN T8 [ get\_ports out]

set\_property IOSTANDARD LVCMOS33 [ get\_ports out]

**Design scheme of 3-bit binary numerical comparator**

**(A) One bit binary value comparator**

(a) source program

`timescale 1ns / 1ps

module c\_A ( a,b ,F1,F2,F3);

input a,b ;

output F 1,F 2,F3;

reg F 1,F 2,F3;

always@( a,b )

if(a>b)

begin F 1=1; F2=0; F3=0; end

else if(a==b)

begin F 1=0; F2=1; F3=0; end

else if(a<b)

begin F 1=0; F2=0; F3=1; end

endmodule

(b) Simulation program

module c\_A\_sim ;

reg a,b ;

wire F 1,F 2,F3;

c\_A uut (.a (a), .b(b), .F1(F1), .F2(F2), .F3(F3));

initial begin

a = 1;

b = 0;

# 100 a =0; b=0;

# 100 a =0; b=1;

# 100 a =1; b=1;

end

endmodule

(c) Pin constraint (binding) procedure

###############################

# On-board Slide Switches #

###############################

set\_property PACKAGE\_PIN U9 [ get\_ports a]

set\_property IOSTANDARD LVCMOS33 [ get\_ports a]

set\_property PACKAGE\_PIN U8 [ get\_ports b]

set\_property IOSTANDARD LVCMOS33 [ get\_ports b]

set\_property PACKAGE\_PIN R7 [ get\_ports b]

###############################

# On-board led #

###############################

set\_property PACKAGE\_PIN T4 [ get\_ports F1]

set\_property IOSTANDARD LVCMOS33 [ get\_ports F1]

set\_property PACKAGE\_PIN U7 [ get\_ports F2]

set\_property IOSTANDARD LVCMOS33 [ get\_ports F2]

set\_property PACKAGE\_PIN U6 [ get\_ports F3]

set\_property IOSTANDARD LVCMOS33 [ get\_ports F3]

**(B) 3-bit binary value comparator, requirement: Realize by instantiating A [calling (A) implemented 1-bit binary value comparator].**

(a) source program

`timescale 1ns / 1ps

module c\_ b ( input [2:0]a,[2:0]b, output F1,F2,F3 );

wire [ 1:3] y ,n,s ;

c C3 (.A (a[2]),.B(b[2]),.F1(s[1]),.F2(s[2]),.F3(s[3]));

c C2 (.A (a[1]),.B(b[1]),.F1(n[1]),.F2(n[2]),.F3(n[3]));

c C1 (.A (a[0]),.B(b[0]),.F1(y[1]),.F2(y[2]),.F3(y[3]));

assign F1 = s[ 1] | (s[2]&n[1]) | (s[2]&n[2]&y[1]);

assign F2 = s[ 2] & n[2] & y[2];

assign F3 = ~(F1|F2);

endmodule

(b) Simulation program

`timescale 1ns / 1ps

module c\_b\_sim ;

reg [ 2:0] a , b ;

wire F 1,F 2,F3;

c\_b comp (.a (a),.b(b),.F1(F1),.F2(F2),.F3(F3));

initial

begin

a=3'b010; b=3'b010;

#50

a=3'b010; b=3'b110;

#50

a=3'b 010; b=3'b100;

#50

a=3'b 100; b=3'b111;

#50

a=3'b 001; b=3'b000;

end

endmodule

(c) Pin constraint (binding) procedure

set\_property PACKAGE\_PIN P4 [ get\_ports a[ 2]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports a[ 2]]

set\_property PACKAGE\_PIN P3 [ get\_ports a[ 1]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports a[ 1]]

set\_property PACKAGE\_PIN R3 [ get\_ports a[ 0]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports a[ 0]]

set\_property PACKAGE\_PIN R6 [ get\_ports b[ 2]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports b[ 2]]

set\_property PACKAGE\_PIN R7 [ get\_ports b[ 1]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports b[ 1]]

set\_property PACKAGE\_PIN U8 [ get\_ports b[ 0]]

set\_property IOSTANDARD LVCMOS33 [ get\_ports b[ 0]]

set\_property PACKAGE\_PIN P5 [ get\_ports F1]

set\_property IOSTANDARD LVCMOS33 [ get\_ports F1]

set\_property PACKAGE\_PIN R1 [ get\_ports F2]

set\_property IOSTANDARD LVCMOS33 [ get\_ports F2]

set\_property PACKAGE\_PIN V1 [ get\_ports F3]

set\_property IOSTANDARD LVCMOS33 [ get\_ports F3]

6. Experimental result record

**(1) Experimental result record of 4 -bit binary addition / subtraction counter**

**( A ) Give the circuit diagram of the modulo 16 plus 1/minus 1 counter designed by Verilog (** " Schematic" screenshot under RTL Analysis **)**

Figure 2-2 is a screenshot of " Schematic" under RTL Analysis of the 4 -bit binary addition / subtraction counter .

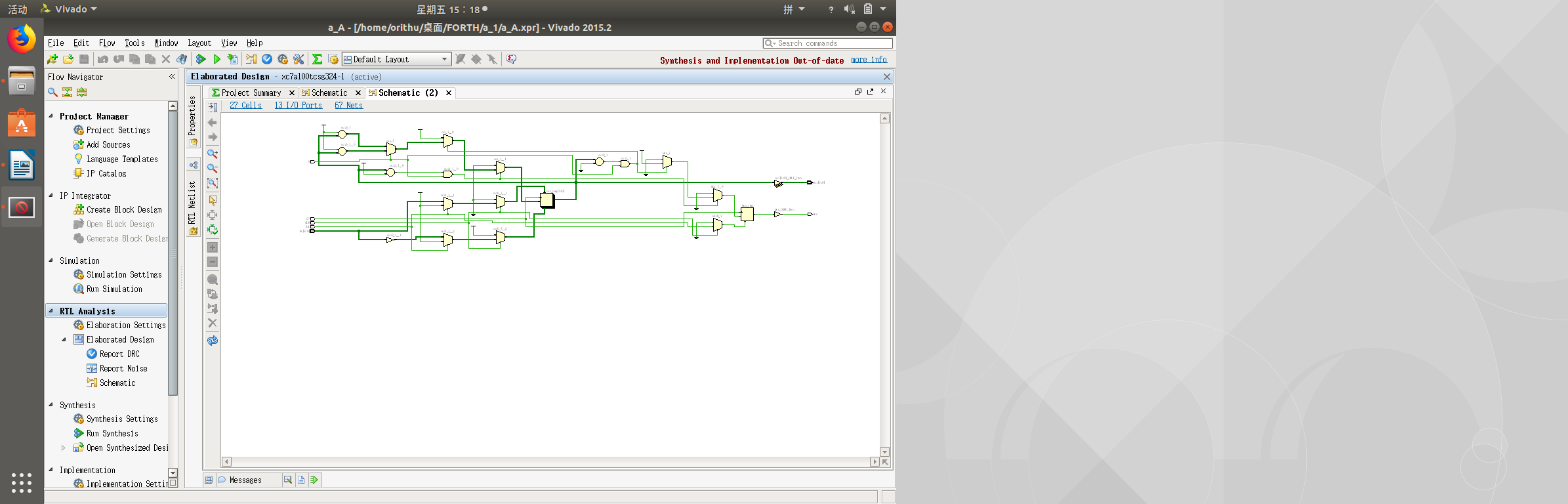


Figure 2 -2 4 -bit binary addition / subtraction counter

**( B ) Screenshot of the simulation result of a modulo 8 counter with an initial value of 2 (waveforms include CP, M, Qa , Qb , Qc, Qd , Q modulo 8 , etc.)**

( The simulation is required to be described as follows: simulation object, input parameters, output parameters, if it is periodic, its "period" should be marked and explained on the simulation diagram)

Note: The waveform situation when M=0 was not given during the inspection , and it has been corrected now.

Figure 2-3 is a simulation waveform diagram of a modulo 8 counter with an initial value of 2. Input parameter : M (counting direction), CP (clock pulse); output parameter : out[3:0] (current count value), period is 8 (2 -9 ).

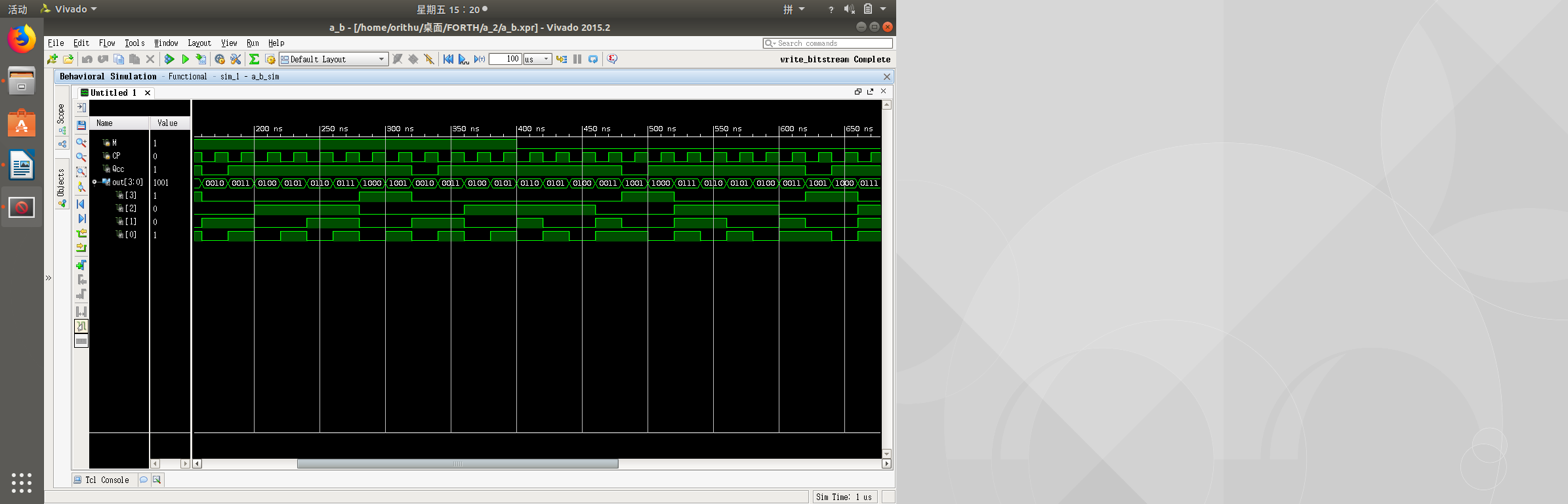


Figure 2-3 Simulation of a modulo 8 counter with an initial value of 2

**( C ) Verification on the development board (main record: verification process and conclusion)**

downloading Bitstream to the board, the initial value is 0010, V9 ( Qcc ) and T5 are on ; M is set to 1, and the CP is toggled back and forth After 8 times, out returns to 0010, and V9 is off ; dial out to 1001, M is set to 0, and after calling back CP 8 times, out returns to 1001, and V9 is off, which proves that the verification is successful.

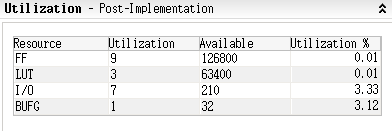


Figure 2 - 4 modulo 16 plus 1/minus 1 counter FPGA resource occupancy

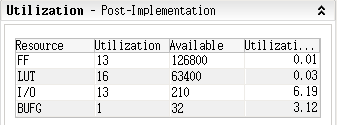


Figure 2-5 Modulo 8 counter FPGA resource occupancy with an initial value of 2

**(2) Using the finite state machine (FSM) to realize the recording of the experimental results of the sequence detector**

**( A ) Give the sequential logic circuit diagram of the Verilog design (** screenshot of " Schematic" under RTL Analysis **)**

Figure 2-6 is a sequential logic circuit diagram of the 10110 sequence detector .

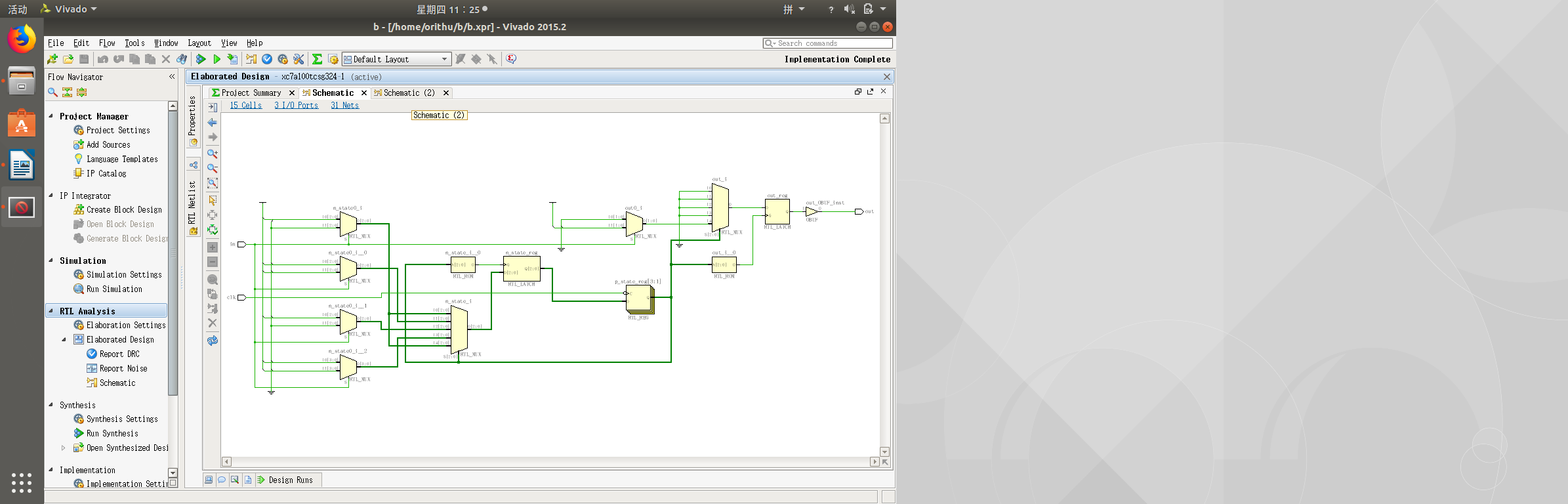


Figure 2-6 10110 sequence detector

**( B ) Screenshot of simulation results (waveform includes clk , input value, output value, etc.)**

( The simulation is required to be described as follows: simulation object, input parameters, output parameters, if it is periodic, its "period" should be marked and explained on the simulation diagram)

Note : Status was not elicited at the time of inspection, currently modified to be correct.

Figure 2-7 is the simulation of 10110 sequence detector . Input parameters: in (input state), clk (clock pulse); output parameters: out (sequence detection), p[1:3] ( current state ).

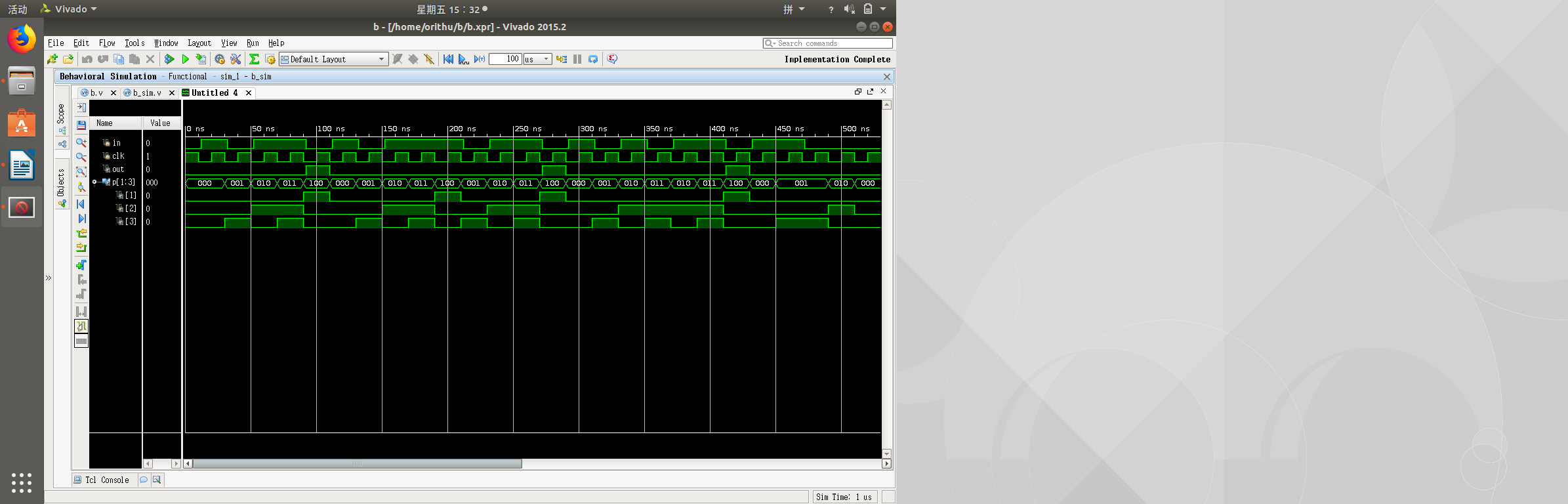


Figure 2-7 10110 sequence detector simulation

**( C ) Verification on the development board (main record: verification process and conclusion)**

After downloading Bitstream to the board, start T8 ( out) off , set R3 (in) to 1, P4 ( clk ) to pulse, R3 to 0, P4 to pulse, R3 to 1, P4 to two pulses, and then set When R3 is turned to 0 , T8 is found to be on, which proves that the verification is successful.

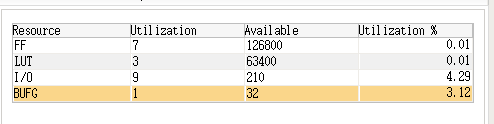


Figure 2-8 10110 Sequence Detector FPGA Resource Occupancy

**Experimental result record of 3-bit binary numerical comparator**

**( A ) The circuit diagram of the 3-bit binary numerical comparator designed by Verilog is given (** screenshot of " Schematic" under RTL Analysis **)**

Figure 2-9 is a circuit diagram of a 3-bit binary value comparator.

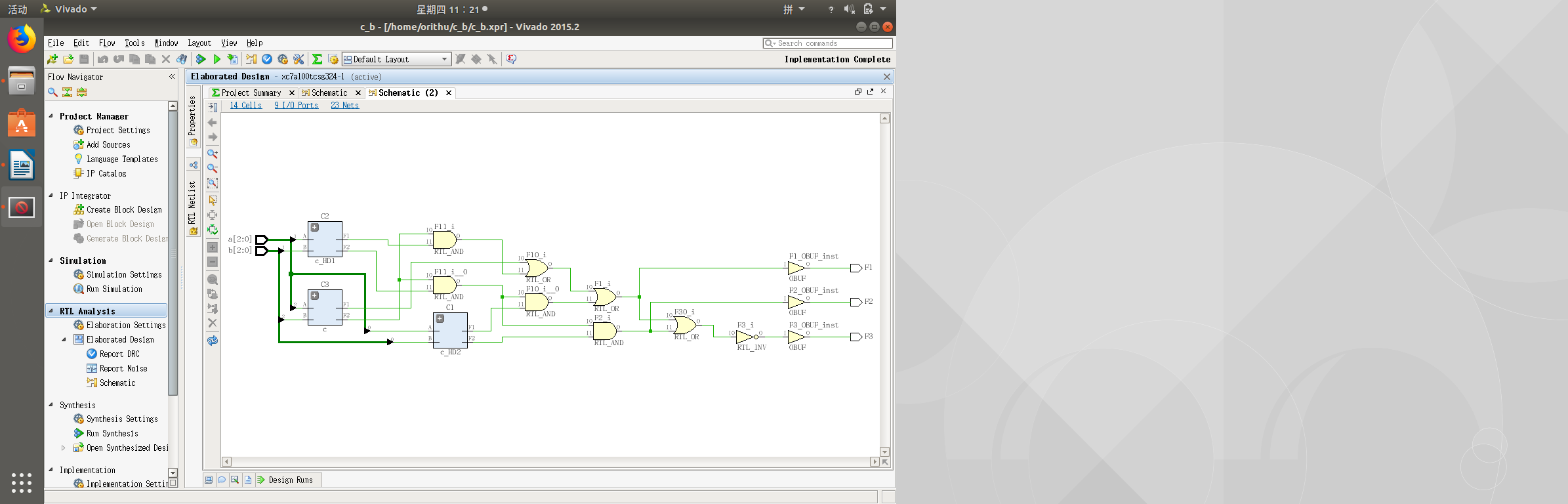


Figure 2-9 3 -bit binary value comparator

**( B ) Screenshot of simulation results (including input values, output values, etc.)**

( The simulation is required to be described as follows: simulation object, input parameters, output parameters, if it is periodic, its "period" should be marked and explained on the simulation diagram)

Figure 2-10 is a simulation diagram of a 3-bit binary value comparator . Input parameters : 3-digit binary numbers a, b ; output parameters: comparison results F1 ( a>b ) , F2 (a==b) , F3 (a<b) .

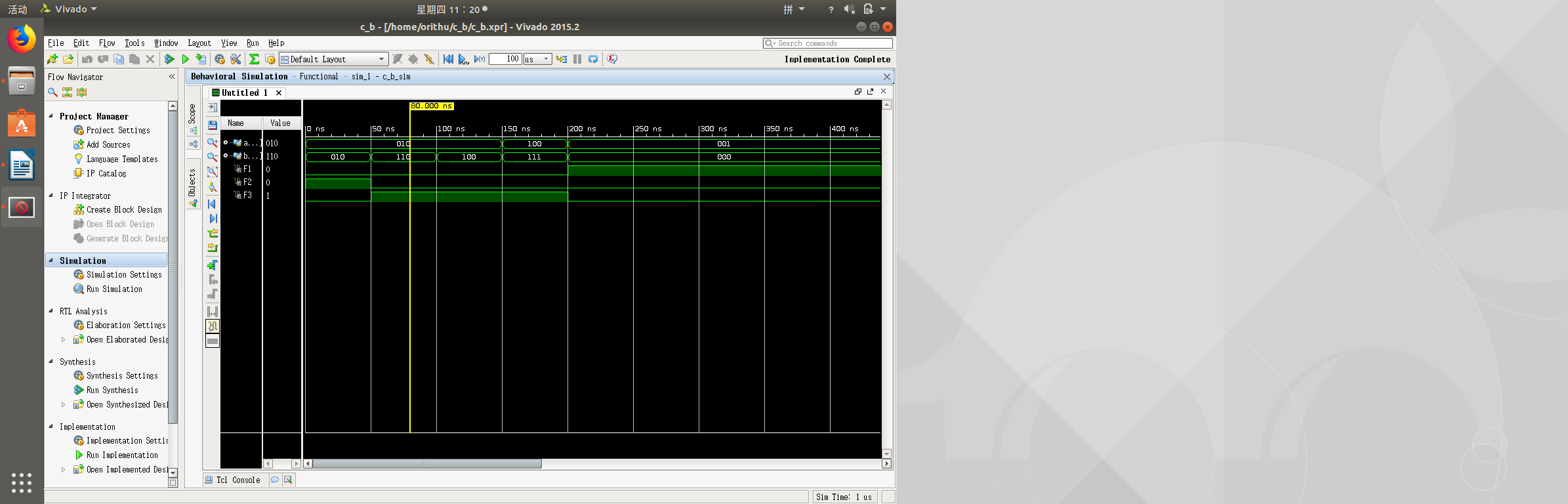


Figure 2-10 Simulation of 3-bit binary numerical comparator behavior

**( C ) Verification on the development board (main record: verification process and conclusion)**

After downloading Bitstream to the development board , at the beginning a and b are both 0, and R1(F2) is on ; when a > b , P5(F1) is on ; when a<b , V1(F3) is on ; both a and b are on When it is 3'b111 , R1 (F2) lights up, which proves that the verification is successful .

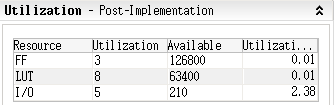


Figure 2-11 1 -bit binary value comparator FPGA resource occupancy

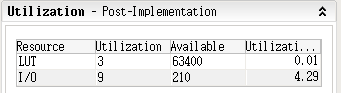


Figure 2-12 3 -bit binary value comparator FPGA resource occupancy

7. Thoughts after the experiment

**1 . Please use a specific example to explain how you use simulation to verify the correctness of your circuit design .**

testing the 3-bit binary number comparator, the behavioral simulation can output the correct result ; after running the integrated project, the post-simulation is the same as the pre-simulation result , as shown in Figure 2-13 , it can be judged that the circuit design is correct .

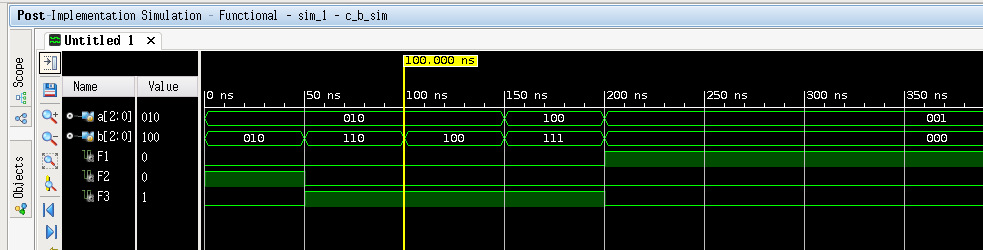
****

Figure 2-13 3-bit Binary Numerical Comparator Timing Simulation

**2. comments and suggestions**

Properly extend the experiment time, and split the tasks of two experiments into three . At the same time , use the doubled experimental class hours to reduce the difficulty gradient of each experiment .