

# 数字逻辑实验报告（2）

|  |  |  |
| --- | --- | --- |
| **数字逻辑实验2** | | |
| **一、Verilog HDL设计数字逻辑电路50%** | **二、Verilog HDL设计较复杂数字逻辑电路50%** | **总成绩** |
|  |  |  |

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：

**姓 名： 刘逸帆**

**学 号： U201610504**

**班 级： 校交1601班**

**指 导 教 师： 徐有青**

**计算机科学与技术学院**

**2018 年 6月 14日**



**数字逻辑实验报告**

Verilog HDL设计数字逻辑电路预习报告

一、Verilog HDL设计数字逻辑电路

1、实验名称

Verilog HDL设计数字逻辑电路。

2、实验目的

要求同学用Verilog HDL设计数字逻辑电路，通过3个逻辑电路实验，并利用“Vivado 2015.2”软件进行“前、后”仿真检查电路设计，然后在“Xilinx NEXYS 4开发板”上操作、记录实验结果，最后验证设计是否达到要求。

通过以上设计、仿真、验证3个训练过程使同学们掌握Verilog HDL设计数字逻辑电路的基本方法，同时掌握如何避免锁存器的产生以及电路设计中的一致性问题的处理方法。

3、实验所用设备

Xilinx NEXYS 4开发板（芯片为XC7A100TCSG324-1，封装为CSG324，软件为Vivado 2015.2）1套。

4、实验内容

**（1）组合、时序逻辑电路的“always”设计**

**（A）用“always块”设计纯组合逻辑电路**

组合电路的一个特性是它的输出永远受输入变化的影响。也就是说组合电路绝不会保持它们以前的值，即输出不会出现锁存。

在使用always块中的case，if-else等语句设计纯组合逻辑电路时，要保证所有输入条件，其输出均有输出值，否则有可能会产生锁存器，导致“综合”出错。

**例如：**某题目要求使用Verilog设计实现一个纯组合逻辑电路的选择器，某同学设计了一个带“flag”标识的4选1的多路选择器，参见程序1-1所示，但是在“综合”时，报3个错误，即：“Place 30-574、Place 30-99、Common 17-69”。

具体要求：

（a）验证程序1-1在“综合”时，是否会出现上述问题；

（b）如果存在上述问题，请更正程序1-1，帮这位同学完成设计。

程序1-1 带“flag”标识的4选1的多路选择器

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'b00 : begin if(flag) valid\_data = data[0];end

2'b01 : begin if(flag) valid\_data = data[1];end

2'b10 : begin if(flag) valid\_data = data[2];end

2'b11 : begin if(flag) valid\_data = data[3];end

endcase

end

endmodule

///////////////////////////\*.xdc文件//////////////////////////

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）用“always块”设计一个同步时序逻辑电路**

参见图1-1所示电路。



图1-1同步时序逻辑电路

具体要求：

利用Verilog HDL的“always块”设计该电路并进行仿真。

**（注意：阻塞与非阻塞赋值的不同，这种组合和时序的混合电路建议使用非阻塞赋值）**

**（2）脉冲异步计数器的设计**

分析如图1-2所示的脉冲异步计数器电路，完成如下内容：

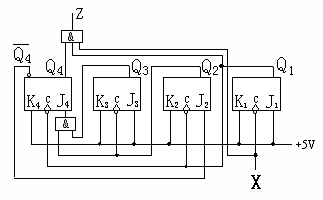


图1-2 脉冲异步计数器电路

（A）说明该计数器的模是多少？

（B）用Verilog DHL实现该电路，并通过仿真和在开发板上验证设计。

**（3）Verilog数字电路设计中一致性问题**

分析如图1-3所示的电路，完成如下内容：

（A）程序1-2是对图1-3所示电路的描述，请用“**Behavior Simulation-它可以称为前仿真**”和“**非Behavior Simulation-它可以称为后仿真**”对程序1-2分别进行仿真，如果有错误，请更正程序1-2，并给出修正后的仿真结果。

（这就是：所谓前仿真和后仿真是否一致的问题）

q1

q2

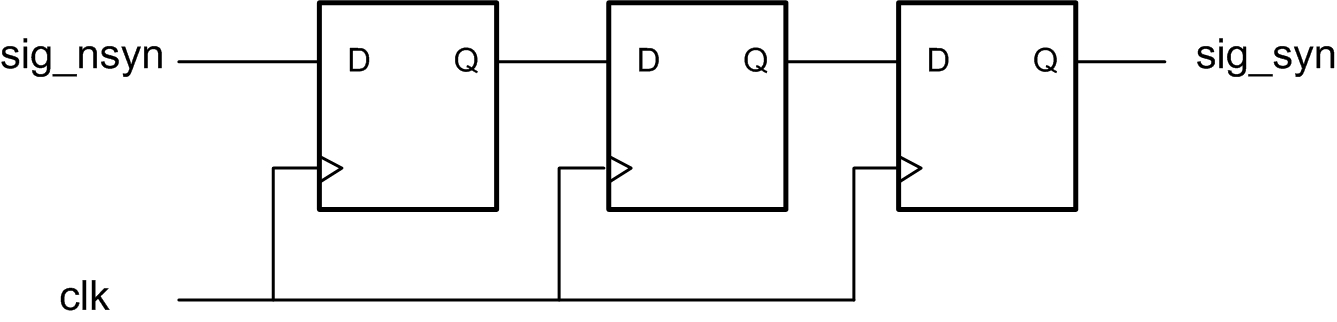


图1-3 3位移位寄存器电路

（B）NEXYS 4开发板提供了一个100Mhz的同步时钟，引脚绑定为“E3”，在更正后的程序1-2中添加一个时钟分频部分，并将降频后的时钟信号接到图1-3中的“clk”，编译成功后再下载到开发板上测试它。

结论：在今后的设计中要保持：“前仿真和后仿真以及下载验证都正确”哟。

程序1-2 3位移位寄存器电路

module pipe3b(

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、实验方案设计

**（1）组合、时序逻辑电路的“always”设计方案**

（A）用“always块”设计纯组合逻辑电路

源程序在综合时会出现问题，是由于行为描述时使用的方式不正确，产生了锁存器。

if语句后对应的else语句与case语句的缺省项必须写，防止产生锁存器。修改后的组合逻辑电路源程序如下所示。

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'b00 : begin

if(flag) valid\_data = data[0];

else valid\_data = 1'b0;

end

2'b01 : begin

if(flag) valid\_data = data[1];

else valid\_data = 1'b0;

end

2'b10 : begin

if(flag) valid\_data = data[2];

else valid\_data = 1'b0;

end

2'b11 : begin

if(flag) valid\_data = data[3];

else valid\_data = 1'b0;

end

default : valid\_data = 1'b0;

endcase

end

endmodule

（B）用“always块”设计实现一个同步时序逻辑电路

（a）源程序

用于实现同步时序逻辑电路的verilog语言程序如下。

module lab3(

input X,

input clk,

output Q0,

output Q1,

output Y

);

reg Q0;

reg Q1;

initial begin

Q0=1'b0;

Q1=1'b0;

end

assign Y = ~(X&(~Q1));

always @(negedge clk) begin

Q0 <= ~Q0;

if(X^Q0) Q1<=~Q1;

end

endmodule

（b）仿真程序

用于仿真测试同步时序逻辑电路的verilog语言程序如下。

module lab3\_2\_sml(

);

reg X;

reg clk\_sim;

wire Q0;

wire Q1;

wire Y;

lab3 simlab3(X,clk\_sim,Q0,Q1,Y);

// Clock Procedure

always begin

clk\_sim <= 0;

#10;

clk\_sim <= 1;

#10;

end // Note: Procedure repeats

// Vector Procedure

initial begin

X <= 0;

@(posedge clk\_sim);

@(posedge clk\_sim);

#5 X <= 1;

@(posedge clk\_sim);

#5 X <= 0;

@(posedge clk\_sim);

$stop;

end

endmodule

**（2）脉冲异步计数器的分析和设计方案**

（A）源程序

用于实现脉冲异步计数器电路的verilog语言程序如下。

module lab3\_3(

X,Q4,Q3,Q2,Q1,Z

);

input X;

output reg Q4;

output reg Q3;

output reg Q2;

output reg Q1;

output Z;

wire K4,K3,K2,K1;

wire J4,J3,J2,J1;

initial begin

Q1<=0;

Q2<=0;

Q3<=0;

Q4<=0;

end

assign K4=1'b1;

assign J4=Q3&Q2;

assign K3=1'b1;

assign J3=1'b1;

assign K2=1'b1;

assign J2=(~Q4);

assign K1=1'b1;

assign J1=1'b1;

assign Z=Q4&Q1&X;

always @(negedge X) begin

if((!K1)&(!J1)) ;

else if((!K1)&J1) Q1=1;

else if(K1&(!J1)) Q1=0;

else Q1=(~Q1);

end

always @(negedge Q1) begin

if((!K2)&(!J2)) ;

else if((!K2)&J2) Q2=1;

else if(K2&(!J2)) Q2=0;

else Q2=(~Q2);

end

always @(negedge Q2) begin

if((!K3)&(!J3)) ;

else if((!K3)&J3) Q3=1;

else if(K3&(!J3)) Q3=0;

else Q3=(~Q3);

end

always @(negedge Q1) begin

if((!K4)&(!J4)) ;

else if((!K4)&J4) Q4=1;

else if(K4&(!J4)) Q4=0;

else Q4=(~Q4);

end

endmodule

（B）仿真程序

用于仿真测试脉冲异步计数器电路的verilog语言程序如下。

module lab3\_3\_sim(

);

reg X;

wire Q4,Q3,Q2,Q1,Z;

lab3\_3 simlab3\_3(X,Q4,Q3,Q2,Q1,Z);

// Clock Procedure

always begin

X <= 0;

#10;

X <= 1;

#10;

end // Note: Procedure repeats

endmodule

（C）引脚约束（绑定）程序

用于将脉冲异步计数器电路下载到开发板上的引脚约束文件如下。

# Switches

#Bank = 34, Pin name = IO\_L21P\_T3\_DQS\_34, Sch name = SW0

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

set\_property IOSTANDARD LVCMOS33 [get\_ports X]

# LEDs

#Bank = 34, Pin name = IO\_L24N\_T3\_34, Sch name = LED0

set\_property PACKAGE\_PIN T8 [get\_ports {Z}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Z}]

#Bank = 34, Pin name = IO\_L21N\_T3\_DQS\_34, Sch name = LED1

set\_property PACKAGE\_PIN V9 [get\_ports {Q1}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Q1}]

#Bank = 34, Pin name = IO\_L24P\_T3\_34, Sch name = LED2

set\_property PACKAGE\_PIN R8 [get\_ports {Q2}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Q2}]

#Bank = 34, Pin name = IO\_L23N\_T3\_34, Sch name = LED3

set\_property PACKAGE\_PIN T6 [get\_ports {Q3}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Q3}]

#Bank = 34, Pin name = IO\_L12P\_T1\_MRCC\_34, Sch name = LED4

set\_property PACKAGE\_PIN T5 [get\_ports {Q4}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Q4}]

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

**（3）Verilog设计中一致性问题的解决方案**

（A）源程序

更正程序1-2后的源程序如下,此时前仿真和后仿真以及下载验证都正确。

module pipe3b(

input sig\_nsyn,

input clk,

output q1,

output q2,

output sig\_syn

);

reg q1;

reg q2;

reg sig\_syn;

always@(posedge clk) begin

sig\_syn=q2;

q2=q1;

q1=sig\_nsyn;

end

（B）仿真程序

用于仿真测试电路的verilog语言程序如下。

module lab3\_4\_sim(

);

reg clk\_sim;

reg sig\_nsyn;

wire q1;

wire q2;

wire sig\_syn;

pipe3b mypipe(sig\_nsyn,clk\_sim,q1,q2,sig\_syn);

always begin

clk\_sim<=0;

#10;

clk\_sim<=1;

#10;

end

always begin

sig\_nsyn <= 0;

@(posedge clk\_sim);

@(posedge clk\_sim);

#5 sig\_nsyn <= 1;

@(posedge clk\_sim);

#5 sig\_nsyn <= 0;

@(posedge clk\_sim);

@(posedge clk\_sim);

@(posedge clk\_sim);

@(posedge clk\_sim);

@(posedge clk\_sim);

// $stop;

end

endmodule



**数字逻辑实验报告**

Verilog HDL设计较复杂数字逻辑电路

二、Verilog HDL设计较复杂数字逻辑电路

1、实验名称

Verilog HDL设计较复杂数字逻辑电路。

2、实验目的

要求同学用Verilog HDL设计较复杂的数字逻辑电路，通过3个逻辑电路实验，并利用“Vivado 2015.2”软件进行“前、后”仿真检查电路设计，然后在“Xilinx NEXYS 4开发板”上操作、记录实验结果，最后验证设计是否达到要求。

通过以上设计、仿真、验证3个训练过程使同学们掌握Verilog HDL设计较复杂数字逻辑电路的基本方法，同时掌握“电路例化”、“模块化”的使用、异步时序逻辑电路的同步化处理以及用状态机设计控制电路。

3、实验所用组件

Xilinx NEXYS 4开发板（芯片为XC7A100TCSG324-1，封装为**CSG324**，软件为Vivado 2015.2）1套。

4、实验内容

**（1）4位二进制加法/减法计数器的设计**

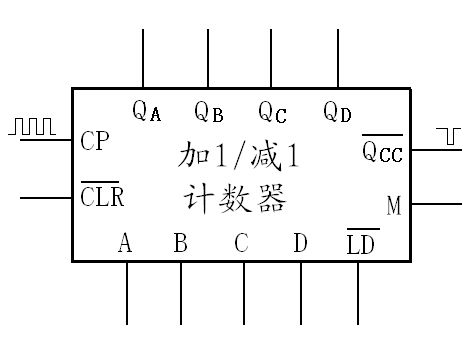
设计一个能清零、置数和进位/借位输出的加1/减1的4位二进制计数器，其结构框图如图2-1所示。

图2-1 4位二进制加法/减法计数器

电路输入为计数脉冲CP、工作模式选择M、预制初值D，C，B，A（其中D为高位，A为低位）和预制控制，清零端；

输出为计数值QD，QC，QB，QA（QD为高位，QA为低位）和进位/借位输出；

当为0时，电路输出清零；

预制控制=0时，将D、C、B、A的输入值送到计数器中，并立即在QD，QC，QB，QA中输出；

模式选择端M=1时加1计数；

当M=0时减1计数；

当CP端输入一个上升沿信号时进行一次计数；

计数有进位/借位时端输出一个负脉冲。

**注意：**用Verilog设计电路时，经常会遇到这样一些问题，例如：

（A）用两个always块对一个寄存器进行赋值，无论其中经过了怎样的条件判断，最终结果毫无疑问是将两个相独立的触发信号连在了寄存器的CLK端上，一个端口接入两信号，所以这样的语句是无法被综合成电路的。

（B）某电路如果有多个输入都可能引起输出值的改变，在使用“always”时，如果其触发条件为电路的“多个输入”时，如果语句的“并发性”处理不好，会造成系统编译成功、“行为仿真”也成功，但是系统生成不了“bit”文件。

解决方法：采用所谓“异步时序逻辑电路的同步化处理”，即：减少“always”的触发条件。

**具体要求：**

（A）用Verilog HDL实现该计数器，将之下载到开发板中进行验证；

（B）用已实现了的“4位二进制计数器”，采用“电路例化”或者“模块化”实现一个初值为2的模8计数器，并下载到开发板中进行验证；

（C）给出设计占用FPGA芯片的资源情况（希望越少越好）。

**（2）采用有限状态机(FSM) 实现序列检测器**

设计一个简单的状态机，其功能是检测一个串行的5位不可重叠的“10110”二进制序列检测器，当输入值出现“10110”时，给出输出标志。

具体要求如下：

（A）给出不可重叠的“10110”二进制序列的状态转移图；

（B）采用有限状态机“标准模板”来设计“10110”二进制序列检测器，在仿真正确后再下载到开发板中进行验证；

（C）给出设计占用FPGA芯片的资源情况（希望越少越好）。

**（所谓有限状态机“标准模板”请参考教材：夏雨闻。Verilog数字系统设计教程第3版。北京:北京航空航天大学出版社,2013.）**

**（3）3位二进制数值比较器的设计**

设计一个3位二进制数值比较器。当A>B时，F1=1，F2=F3=0；当A=B时， F2=1，F1=F3=0；当A<B时，F3=1，F1=F2=0。

具体要求：

（A）用Verilog HDL设计一个一位二进制数值比较器；

（B）用已实现了的一位二进制数值比较器，采用“电路例化”或者“模块化”实现一个3位二进制数值比较器；

（C）将所设计的电路下载到开发板上进行验证；

（D）给出设计占用FPGA芯片的资源情况（希望越少越好）。

5、实验方案设计

**（1）4位二进制加法/减法计数器的设计方案**

（A）模16加1/减1计数器

（a）源程序

module counter\_16 #(parameter LOW=0, parameter HIGH=15)(

input clk, //clock

input rst, //reset

input en, //choose inc or dec

input [3:0]data,//load data

input load, //choose load

output [3:0] dout,//4bits output

output reg cout //co output

);

reg [3:0]Q1;

assign dout=Q1;

always @(posedge clk or negedge rst) begin

if(rst) Q1<=LOW;

else if (en) begin//add counter

if(load) Q1<=data;

else if(Q1<HIGH) Q1<=Q1+1;

else Q1<=LOW;

end

else if(!en) begin//dec counter

if(load) Q1<=data;

else if(Q1>LOW) Q1<=Q1-1;

else Q1<=HIGH;

end

end

always @(Q1) begin

if(en) begin

if(Q1==HIGH) cout=1'b1;

else cout=1'b0;

end

else if(!en) begin

if(Q1==LOW)cout=1'b1;

else cout=1'b0;

end

end

endmodule

（b）仿真程序

module counter\_sim(

);

reg clk\_sim; //clock

reg rst; //reset

reg en; //choose inc or dec

reg [3:0]data;//load data

reg load; //choose load

wire [3:0] dout;//4bits output

wire cout; //co output

counter\_16 #(0,15) mycounter(clk\_sim,rst,en,data,load,dout,cout);

always begin

clk\_sim<=0;

#10;

clk\_sim<=1;

#10;

end

always begin

en=1;

#1000;

en=0;

#1000;

end

initial begin

clk\_sim=0;

rst=0;

data=4'b0010;

load=0;

#7;

load=1;

#20;

load=~load;

#50;

rst=1;

#20;

rst=0;

end

endmodule

（c）引脚约束（绑定）程序

# Switches

#Bank = 34, Pin name = IO\_L21P\_T3\_DQS\_34, Sch name = SW0

set\_property PACKAGE\_PIN U9 [get\_ports {clk\_count}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {clk\_count}]

#Bank = 34, Pin name = IO\_25\_34, Sch name = SW1

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

set\_property IOSTANDARD LVCMOS33 [get\_ports {rst}]

#Bank = 34, Pin name = IO\_L23P\_T3\_34, Sch name = SW2

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

set\_property IOSTANDARD LVCMOS33 [get\_ports {en}]

# LEDs

#Bank = 34, Pin name = IO\_L24N\_T3\_34, Sch name = LED0

set\_property PACKAGE\_PIN T8 [get\_ports {dout[0]}]

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

#Bank = 34, Pin name = IO\_L21N\_T3\_DQS\_34, Sch name = LED1

set\_property PACKAGE\_PIN V9 [get\_ports {dout[1]}]

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

#Bank = 34, Pin name = IO\_L24P\_T3\_34, Sch name = LED2

set\_property PACKAGE\_PIN R8 [get\_ports {dout[2]}]

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

#Bank = 34, Pin name = IO\_L23N\_T3\_34, Sch name = LED3

set\_property PACKAGE\_PIN T6 [get\_ports {dout[3]}]

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

#Bank = 34, Pin name = IO\_L12P\_T1\_MRCC\_34, Sch name = LED4

set\_property PACKAGE\_PIN T5 [get\_ports {cout}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {cout}]

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

**（B）初值为2的模8计数器，要求：通过例化A【调用（A）实现的计数器】来实现。**

（a）源程序

module counter\_2to9(

input clk,

input rst,

input en,

input [3:0]data,

input load,

output [3:0]dout,

output cout

);

counter\_16 #(2,9) mycounter(clk,rst,en,data,load,dout,cout);

endmodule

（b）仿真程序

module counter\_sim(

);

reg clk\_sim; //clock

reg rst; //reset

reg en; //choose inc or dec

reg [3:0]data;//load data

reg load; //choose load

wire [3:0] dout;//4bits output

wire cout; //co output

counter\_2to9 mycounter(clk\_sim,rst,en,data,load,dout,cout);

always begin

clk\_sim<=0;

#10;

clk\_sim<=1;

#10;

end

always begin

en=1;

#1000;

en=0;

#1000;

end

initial begin

clk\_sim=0;

rst=0;

data=4'b0010;

load=0;

#7;

load=1;

#20;

load=~load;

#50;

rst=1;

#20;

rst=0;

end

endmodule

（c）引脚约束（绑定）程序

# Switches

#Bank = 34, Pin name = IO\_L21P\_T3\_DQS\_34, Sch name = SW0

set\_property PACKAGE\_PIN U9 [get\_ports {clk\_count}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {clk\_count}]

#Bank = 34, Pin name = IO\_25\_34, Sch name = SW1

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

set\_property IOSTANDARD LVCMOS33 [get\_ports {rst}]

#Bank = 34, Pin name = IO\_L23P\_T3\_34, Sch name = SW2

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

set\_property IOSTANDARD LVCMOS33 [get\_ports {en}]

# LEDs

#Bank = 34, Pin name = IO\_L24N\_T3\_34, Sch name = LED0

set\_property PACKAGE\_PIN T8 [get\_ports {dout[0]}]

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

#Bank = 34, Pin name = IO\_L21N\_T3\_DQS\_34, Sch name = LED1

set\_property PACKAGE\_PIN V9 [get\_ports {dout[1]}]

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

#Bank = 34, Pin name = IO\_L24P\_T3\_34, Sch name = LED2

set\_property PACKAGE\_PIN R8 [get\_ports {dout[2]}]

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

#Bank = 34, Pin name = IO\_L23N\_T3\_34, Sch name = LED3

set\_property PACKAGE\_PIN T6 [get\_ports {dout[3]}]

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

#Bank = 34, Pin name = IO\_L12P\_T1\_MRCC\_34, Sch name = LED4

set\_property PACKAGE\_PIN T5 [get\_ports {cout}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {cout}]

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

**（2）采用有限状态机(FSM) 实现序列检测器的设计方案**

（A）串行5位不可重叠的“10110”二进制序列检测器的状态图

为保证状态机仅在时钟上升沿处判断输入序列，添加第六个状态LIGHT。以此防止序列检测器在没有时钟信号时对输入进行判断，给出错误输出信号。状态图如图2-2所示。

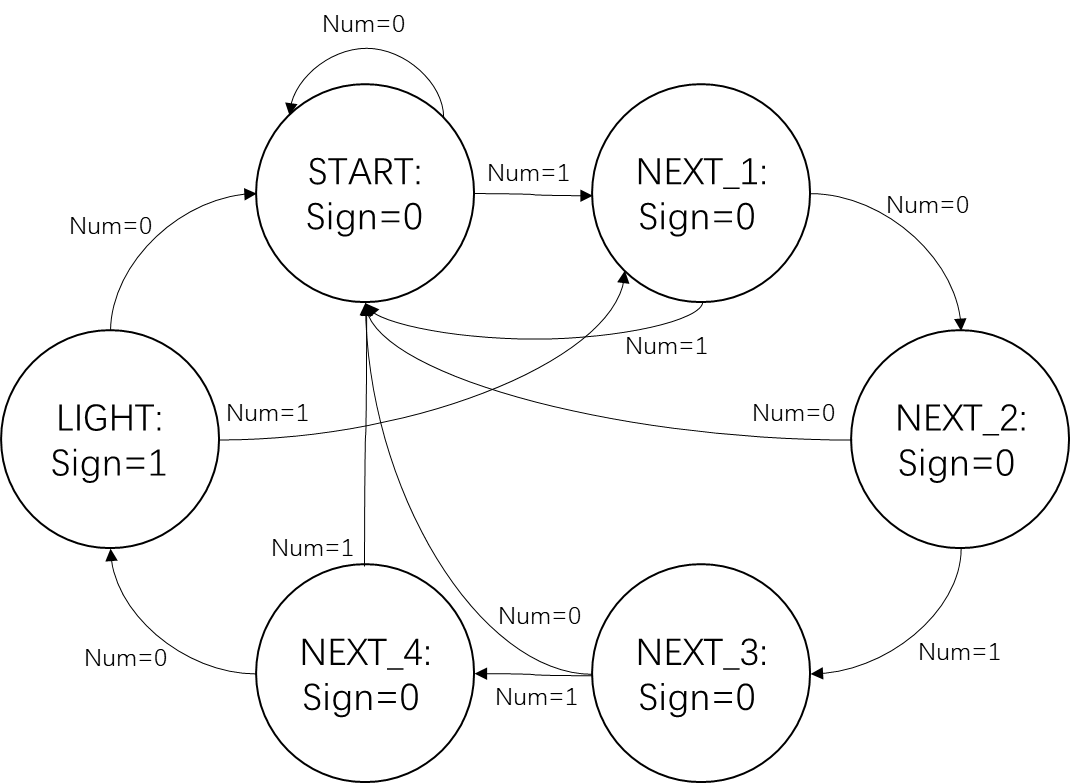


图2-2 有限状态机的状态图

（B）源程序

module FSM(

input clk\_x,

input rst,

input num\_x,

output reg sign,

output reg [2:0]state

);

parameter START = 3'b000, NEXT\_1 = 3'b001, NEXT\_2 = 3'b010, NEXT\_3 = 3'b011, END = 3'b100, LIGHT = 3'b101;

reg [2:0] next\_state;

always @(posedge clk\_x) begin

if(rst==1'b1)

state <= START;

else

state <= next\_state;

end

always @(\*) begin

case(state)

START: begin

sign=1'b0;

if(num\_x==1'b1) next\_state = NEXT\_1; //check first sign '1'

else next\_state = START;

end

NEXT\_1: begin

sign=1'b0;

if(num\_x==1'b0) next\_state = NEXT\_2; //check second sign '2'

else next\_state = START;

end

NEXT\_2: begin

sign=1'b0;

if(num\_x==1'b1) next\_state = NEXT\_3; //check third sign '1'

else next\_state = START;

end

NEXT\_3: begin

sign=1'b0;

if(num\_x==1'b1) next\_state = END; //check fourth sign '1'

else next\_state = START;

end

END: begin

sign=1'b0;

if(num\_x==1'b0) next\_state = LIGHT; //check the last sign '1'

else next\_state = START;

end

LIGHT: begin

sign=1'b1;

if(num\_x==1'b1) next\_state = NEXT\_1; //check the last sign '1'

else next\_state = START;

end

default: next\_state = START;

endcase

end

endmodule

（C）仿真程序

module lab4\_2\_sim(

);

reg clk;

reg rst;

reg x;

wire sign;

wire [2:0]state;

FSM myfsm(clk,rst,x,sign,state);

always begin

clk<=0;

#10;

clk<=1;

#10;

end

initial begin

rst<=0;

clk<=0;

x<=0;

#15;

rst=1;

#20;

rst=0;

#20;

x<=0;

#20;

x<=1;

#20;

x<=1;

#20;

x<=0;

#20;

x<=1;

#20;

x<=0;

#20;

x<=1;

#20;

x<=1;

#20;

x<=0;

#20;

x<=1;

#20;

x<=1;

#20;

x<=0;

end

endmodule

（D）引脚约束（绑定）程序

# Switches

#Bank = 34, Pin name = IO\_L21P\_T3\_DQS\_34, Sch name = SW0

set\_property PACKAGE\_PIN U9 [get\_ports {clk\_x}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {clk\_x}]

#Bank = 34, Pin name = IO\_25\_34, Sch name = SW1

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

set\_property IOSTANDARD LVCMOS33 [get\_ports {rst}]

#Bank = 34, Pin name = IO\_L23P\_T3\_34, Sch name = SW2

set\_property PACKAGE\_PIN R7 [get\_ports {num\_x}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num\_x}]

# LEDs

#Bank = 34, Pin name = IO\_L24N\_T3\_34, Sch name = LED0

set\_property PACKAGE\_PIN T8 [get\_ports {sign}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sign}]

#Bank = 34, Pin name = IO\_L7P\_T1\_34, Sch name = LED13

set\_property PACKAGE\_PIN U1 [get\_ports {state[0]}]

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

#Bank = 34, Pin name = IO\_L15N\_T2\_DQS\_34, Sch name = LED14

set\_property PACKAGE\_PIN R2 [get\_ports {state[1]}]

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

#Bank = 34, Pin name = IO\_L15P\_T2\_DQS\_34, Sch name = LED15

set\_property PACKAGE\_PIN P2 [get\_ports {state[2]}]

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

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

**（3）3位二进制数值比较器的设计方案**

**（A）一位二进制数值比较器**

（a）源程序

如下为位数可变的一位二进制数值比较器。

module compare\_1#(parameter WIDTH=1)(

input [WIDTH-1:0]num1,

input [WIDTH-1:0]num2,

output reg [2:0]sign

);

initial begin

sign=0;

end

always @(\*) begin

if(num1>num2) sign=3'b100;

else if(num1==num2) sign=3'b010;

else sign=3'b001;

end

endmodule

（b）仿真程序

module compare\_1\_sim(

);

reg [2:0]num1;

reg [2:0]num2;

wire [2:0]sign;

compare\_1 #(1) mycom(num1,num2,sign);

initial begin

num1=100;

num2=011;

#20;

num2=100;

#20;

num2=101;

end

endmodule

（c）引脚约束（绑定）程序

# Switches

#Bank = 34, Pin name = IO\_L21P\_T3\_DQS\_34, Sch name = SW0

set\_property PACKAGE\_PIN U9 [get\_ports {num2[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num2[0]}]

#Bank = 34, Pin name = IO\_25\_34, Sch name = SW1

set\_property PACKAGE\_PIN U8 [get\_ports {num2[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num2[1]}]

#Bank = 34, Pin name = IO\_L23P\_T3\_34, Sch name = SW2

set\_property PACKAGE\_PIN R7 [get\_ports {num2[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num2[2]}]

#Bank = 34, Pin name = IO\_L11P\_T1\_SRCC\_34, Sch name = SW13

set\_property PACKAGE\_PIN R3 [get\_ports {num1[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num1[0]}]

#Bank = 34, Pin name = IO\_L14N\_T2\_SRCC\_34, Sch name = SW14

set\_property PACKAGE\_PIN P3 [get\_ports {num1[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num1[1]}]

#Bank = 34, Pin name = IO\_L14P\_T2\_SRCC\_34, Sch name = SW15

set\_property PACKAGE\_PIN P4 [get\_ports {num1[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num1[2]}]

# LEDs

#Bank = 34, Pin name = IO\_L24N\_T3\_34, Sch name = LED0

set\_property PACKAGE\_PIN T8 [get\_ports {sign[0]}]

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

#Bank = 34, Pin name = IO\_L21N\_T3\_DQS\_34, Sch name = LED1

set\_property PACKAGE\_PIN V9 [get\_ports {sign[1]}]

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

#Bank = 34, Pin name = IO\_L24P\_T3\_34, Sch name = LED2

set\_property PACKAGE\_PIN R8 [get\_ports {sign[2]}]

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

**（B）3位二进制数值比较器，要求：通过例化A【调用（A）实现的一位二进制数值比较器】来实现。**

（a）源程序

module compare\_3(

input [2:0]num1,

input [2:0]num2,

output [2:0]sign

);

compare\_1 #(3) mycom(num1,num2,sign);

endmodule

（b）仿真程序

module compare\_3\_sim(

);

reg [2:0]num1;

reg [2:0]num2;

wire [2:0]sign;

compare\_3 mycom(num1,num2,sign);

initial begin

num1=100;

num2=011;

#20;

num2=100;

#20;

num2=101;

end

endmodule

（c）引脚约束（绑定）程序

# Switches

#Bank = 34, Pin name = IO\_L21P\_T3\_DQS\_34, Sch name = SW0

set\_property PACKAGE\_PIN U9 [get\_ports {num2[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num2[0]}]

#Bank = 34, Pin name = IO\_25\_34, Sch name = SW1

set\_property PACKAGE\_PIN U8 [get\_ports {num2[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num2[1]}]

#Bank = 34, Pin name = IO\_L23P\_T3\_34, Sch name = SW2

set\_property PACKAGE\_PIN R7 [get\_ports {num2[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num2[2]}]

#Bank = 34, Pin name = IO\_L11P\_T1\_SRCC\_34, Sch name = SW13

set\_property PACKAGE\_PIN R3 [get\_ports {num1[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num1[0]}]

#Bank = 34, Pin name = IO\_L14N\_T2\_SRCC\_34, Sch name = SW14

set\_property PACKAGE\_PIN P3 [get\_ports {num1[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num1[1]}]

#Bank = 34, Pin name = IO\_L14P\_T2\_SRCC\_34, Sch name = SW15

set\_property PACKAGE\_PIN P4 [get\_ports {num1[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {num1[2]}]

# LEDs

#Bank = 34, Pin name = IO\_L24N\_T3\_34, Sch name = LED0

set\_property PACKAGE\_PIN T8 [get\_ports {sign[0]}]

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

#Bank = 34, Pin name = IO\_L21N\_T3\_DQS\_34, Sch name = LED1

set\_property PACKAGE\_PIN V9 [get\_ports {sign[1]}]

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

#Bank = 34, Pin name = IO\_L24P\_T3\_34, Sch name = LED2

set\_property PACKAGE\_PIN R8 [get\_ports {sign[2]}]

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

6、实验结果记录

**（1）4位二进制加法/减法计数器的实验结果记录**

**（A）Verilog设计的模16加1/减1计数器的电路图**

图2-3为verilog设计的4位二进制加减法计数器的电路图。

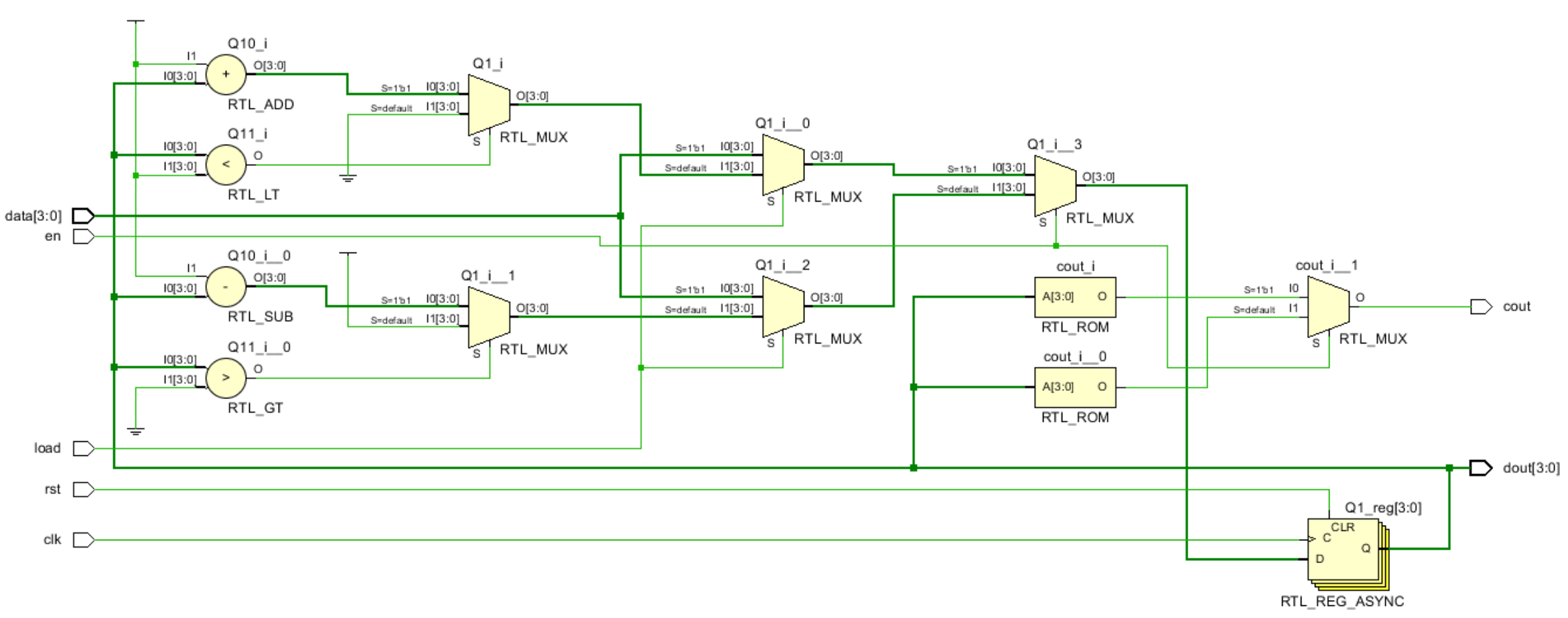


图2-3 4位二进制加法/减法计数器

**（B）初值为2的模8计数器仿真结果截图**

图2-4为初值为2的模8计数器仿真波形结果截图。

输入参数：

clk\_sim为模拟的时钟信号；

rst为置零信号，高电平有效；

en为加减技术状态切换，低电平时计数器进行加技术，高电平时计数器进行减计数；

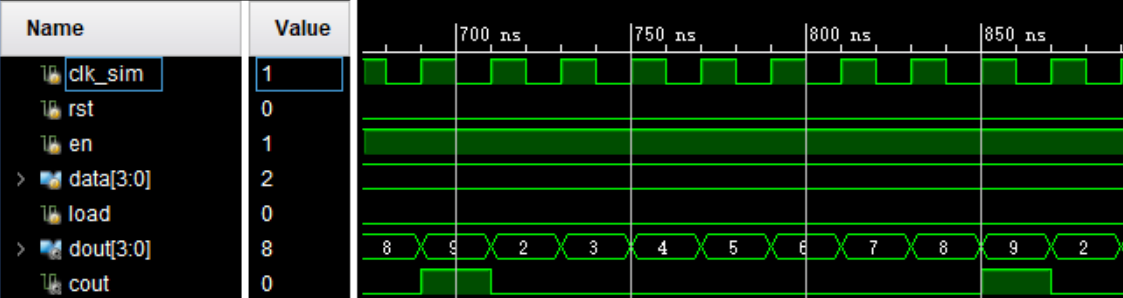
load为同步加载使能信号，高电平有效；

data[3:0]为同步加载信号生效时加载的数据，data[3:0]依次被装载到输出dout[3:0]中。

输出参数：

dout[3:0]为计数器输出信号，输出计数器的技术状态；

cout为进位/借位信号，在加/减计数的一个周期中出现一次。



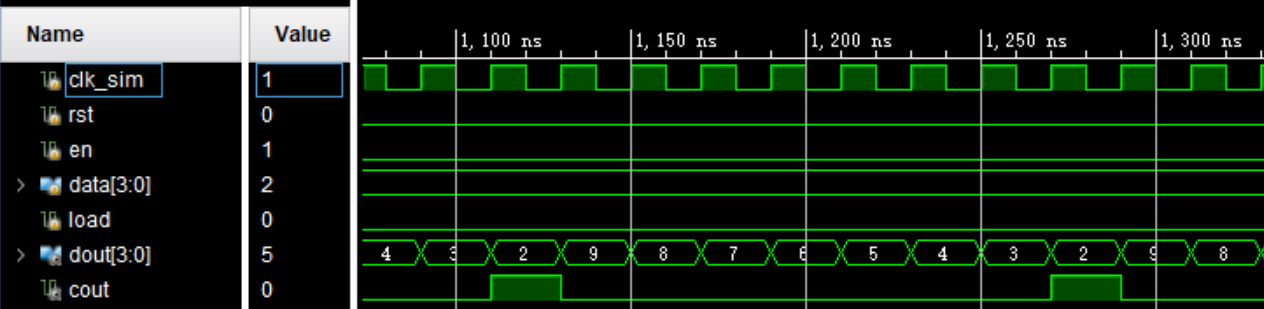


图2-4 初值为2的模8计数器仿真

**（C）开发板上的验证情况**

编写对应的约束文件后将电路下载到开发板上，将计数用的时钟clk、置零信号rst和加减计数使能信号en分别绑定在开发板的拨动开关上。测试后发现减计数正常，但在加计数的时候出现了状态跳变的现象，重新下载一次后问题消失，分析可能是因为在代码中阻塞赋值与非阻塞赋值存在错误使用的情况，导致程序的后仿真与前仿真可能不一致。

最后成功在开发板上进行了验证。

**（D）给出设计占用FPGA芯片的资源情况**

FPGA芯片资源占用情况如图2-5所示。

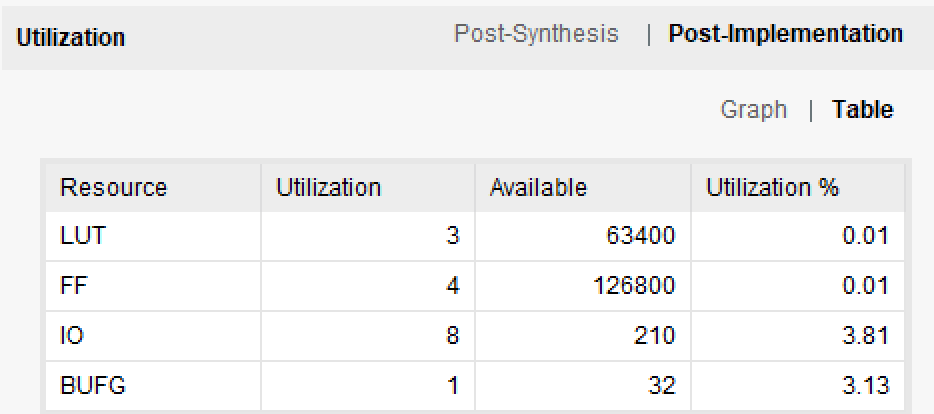
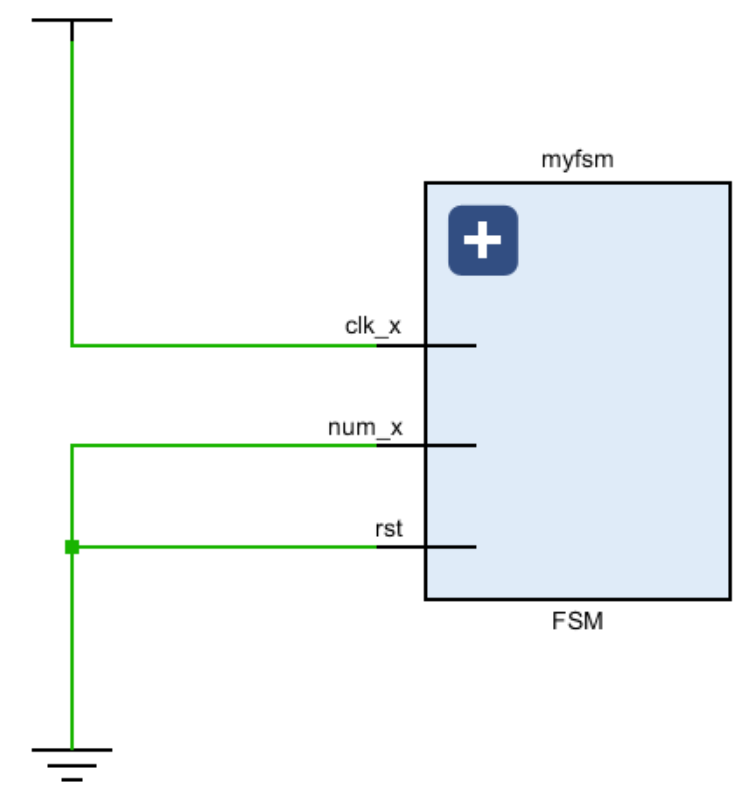


图2-5 FPGA芯片资源占用情况

**（2）采用有限状态机(FSM) 实现序列检测器的实验结果记录**

**（A）Verilog设计的时序逻辑电路图**

图2-6为verilog设计的时序逻辑电路图。



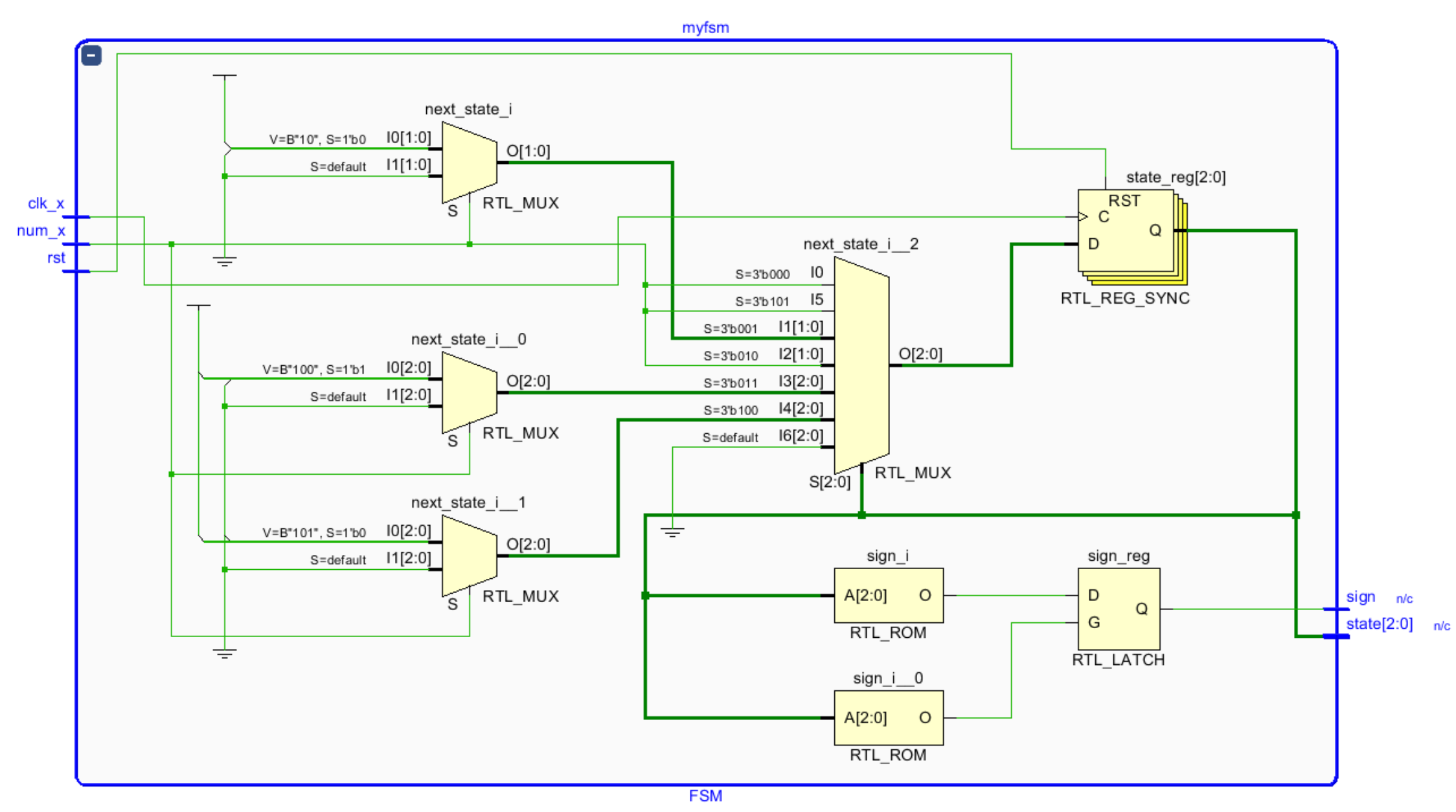


图2-6 10110序列检测器

**（B）仿真结果截图**

图2-7为10110序列检测器的仿真波形结果截图。

输入参数：

clk为模拟的时钟信号；

rst为置零信号，高电平有效；

X为输入信号，其指定的排列即为被检测的序列。

输出参数：

sign为输出的检测信号，在检测到10110序列后输出长度为一个时钟周期的高电平信号；

state[2:0]为状态机状态指示信号，设计的状态机共有0~5六个信号，其中状态5为保证检测信号的输出而设计，将在检测到对应信号后保持sign输出一个周期的高电平。

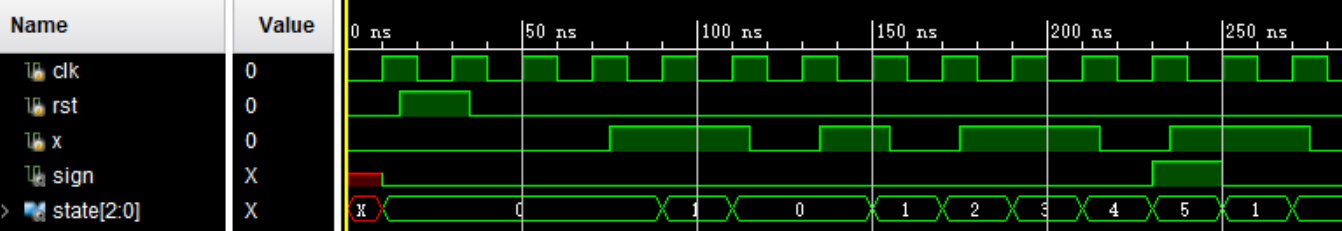


图2-7 10110序列检测器仿真

**（C）开发板上的验证情况**

编写对应的约束文件后将电路下载到开发板上，将时钟信号clk、置零信号rst和输入序列信号X分别绑定在开发板的拨动开关上；输出信号绑定在LED灯上。

开发板上验证电路成功，能够在输入指定序列后输出检测结果。

**（D）给出设计占用FPGA芯片的资源情况**

FPGA芯片资源占用情况如图2-8所示。

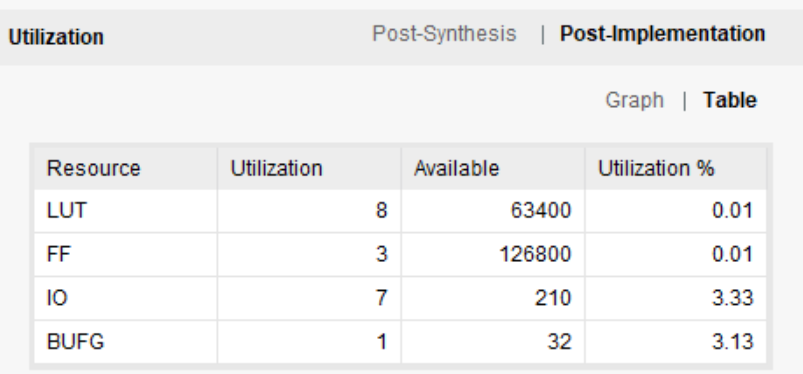


图2-8 FPGA芯片资源占用情况

**（3）3位二进制数值比较器的实验结果记录**

**（A）Verilog设计的3位二进制数值比较器的电路图**

图2-9为verilog设计的3位二进制数值比较器的电路图。

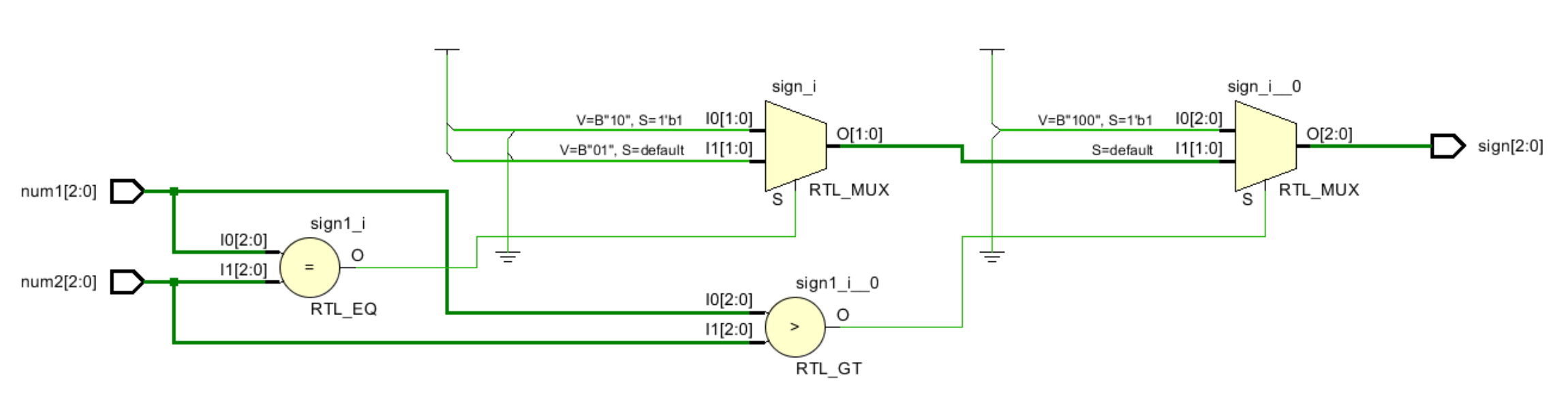


图2-9 3位二进制数值比较器

**（B）仿真结果截图**

图2-10为3位二进制数值比较器的仿真波形结果截图。

输入参数：

num1[2:0]为待比较的三位二进制数之一；

num2[2:0]为带比较的另一个三位二进制数。

输出参数：

sign[2:0]为指示两个3位二进制数大小比较结果的指示信号。

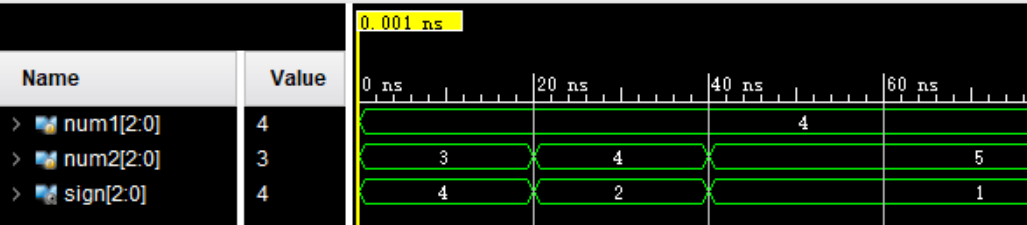


图2-10 3位二进制数值比较器仿真

**（C）开发板上的验证情况（主要记录：验证过程和结论）**

编写对应的约束文件后将电路下载到开发板上，将输入的两个三位二进制数字分别绑定在开发板的拨动开关上；输出信号绑定在LED灯上。

开发板上验证电路成功，能够对不同的二进制数输入指示正确的比较结果。

**（D）给出设计占用FPGA芯片的资源情况**

FPGA芯片资源占用情况如图2-11所示。

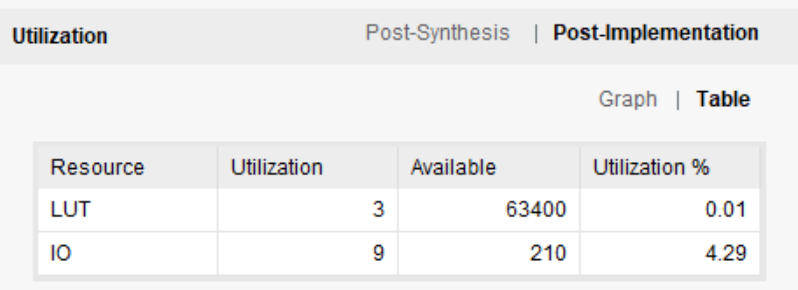


图2-11 FPGA芯片资源占用情况

7、实验后的思考

**1．请通过一个具体的实例来说明你是如何用仿真来验证你电路设计的正确性。**

如实验4中的实验内容2：

首先新建一个仿真文件，通过结构描述引入编写的FSM模块。

然后观察需要提供的输入，有时钟信号clk、置零信号rst、与输入序列信号X，则在仿真文件中对对应信号进行模拟。

对clk使用always语句模拟正常的时钟信号。

通过initial语句将指定序列安排在仿真波形开头处：将rst信号安排在第一个时钟上升沿处保证状态机的初态，随后在连续的时钟上升沿信号处依次安排高电平或低电平的x输入序列。

设计x输入序列时，首先随机设计一段不包含指定序列的信号，测试状态机能否正常工作，然后设计一段指定序列。由于实验内容中的状态机不识别重叠输入，再设计一段重叠的指定序列进行检测。

最后通过vivado的run simulation功能进行前仿真检测，验证电路设计是否正确。

**2．意见和建议**

希望实验内容4中的要求部分再明确一点，与后面的记录部分一致。开始直接做出了初值为2的模8计数器和三位比较器并进行仿真，在看到实验记录部分时才理解具体要求并作出修改。

实验难度适中，起到了锻炼效果。