華中科技大學

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）

姓　　名：吕鹏泽

学　　号：U201614532

班　　级：CS1601

指 导 教 师：赵贻竹

计算机科学与技术学院

2018 年 6 月 20 日

華中科技大學

数字逻辑实验报告

Verilog HDL 设计数字逻辑电路预习报告

《数字电路与逻辑设计》实验报告

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。

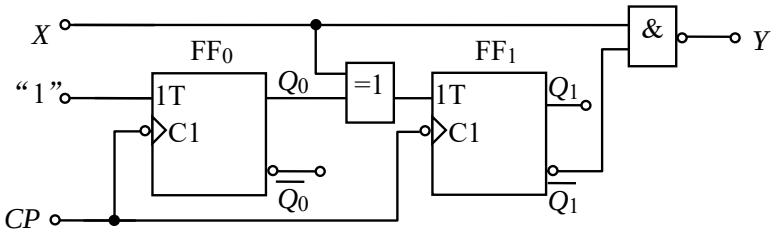


图 1-1 同步时序逻辑电路

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：

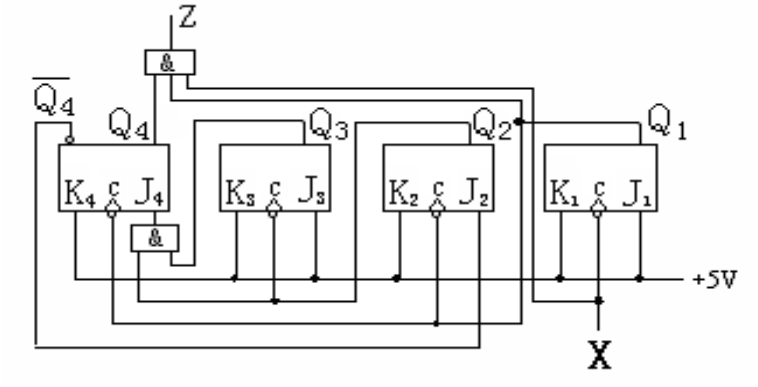


图 1-2 脉冲异步计数器电路

（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）

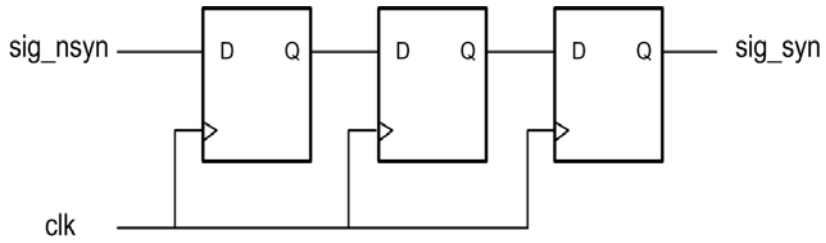


图 1-3　3 位移位寄存器电路

（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

程序 1-1 在综合时，会报 3 个错误，验证结果如图 1-4 所示

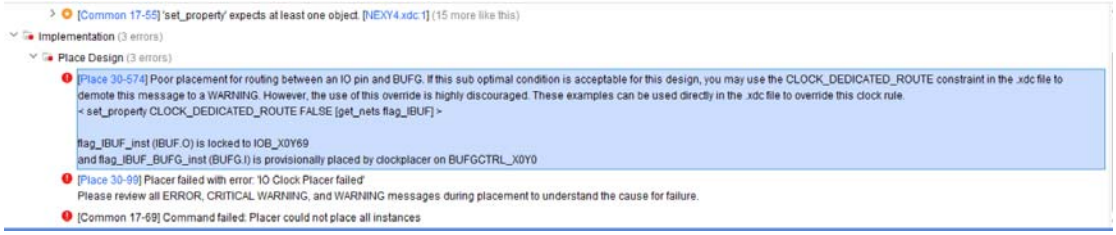


图 1-4 报错信息

这是因为程序的 case 语句中使用了 if，但是却只指出了 flat==1 时候的 valid_data 的值，却没有指明 flat==0 时 valid_data 的值，因此修改后时要在程序的 if 语句后加上 else 语句为 flat==0 时 valid_data 赋值为 0.修改后的程序如程序 1-3 所示。

（A）用“always 块”设计纯组合逻辑电路

程序 1-3 修改后的带“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);

always @ (*)
begin
    case(valid)
        2'b00 : begin if(flag) valid_data = data[0];else valid_data
= 0;end
        2'b01 : begin if(flag) valid_data = data[1];else valid_data
= 0;end
        2'b10 : begin if(flag) valid_data = data[2];else valid_data
= 0;end
        2'b11 : begin if(flag) valid_data = data[3];else valid_data
= 0;end
     default: valid_data=0;
     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

先使用 always 语句设计出 T 触发器模块，然后在主模块中调用 T 触发器的模块实现总的电路，设计程序如程序 1-4 所示。

程序 1-4 “always 块”实现的同步时序逻辑电路

```
//T 触发器模块
module T(
    input IT,
```

```
    input CI,
    output reg Q,
    output wire QB
    );
initial begin Q<=0; end
assign QB=!Q;
always @ (negedge CI)  begin
   Q=Q^IT;
end
endmodule

//主模块
`timescale 1ns / 1ps
module M1(
    input X,
    input CP,
    output reg Y,
    output Q0,
    output Q1
);
wire QB1;
 T T1(.IT(1),.CI(CP),.Q(Q0));
 T T2(.IT(X^Q0),.CI(CP),.Q(Q1),.QB(QB1));
always @ (*) begin
   Y<=!(QB1&X);
end
endmodule
```

（b）仿真程序

```
`timescale 1ns / 1ps
module M1_test( );
reg clk_in;
reg X_in;
wire Y_out,Q0_out,Q1_out;
wire IT1;
M1 test(.X(X_in),.CP(clk_in),.Y(Y_out),.Q0(Q0_out),.Q1(Q1_out));
assign IT1=X_in^Q0_out;
initial begin
    X_in=0;
    clk_in=0;
end
```

```
always begin
    #5 clk_in<=!clk_in;
end
always begin
    #9.3 X_in=!X_in;
end
endmodule
```

（2）脉冲异步计数器的分析和设计方案

首先由电路图写出输出函数和激励函数表达式：

$J_1=K_1=1$, $C_1=X$

$J_2=\overline{Q}_4$, $K_2=1$, $C_2=Q_1$

$J_3=K_3=1$, $C_3=Q_2$

$J_4=Q_3Q_2$, $K_4=1$, $C_4=Q_1$

$Z=Q_4Q_1X$

列出次态真值表，结果如表 1-1 所示

表 1-1 次态真值表

输入 X	现态 $Q_4Q_3Q_2Q_1$	激励函数												次态 $Q_4^{n+1}Q_3^{n+1}Q_2^{n+1}Q_1^{n+1}$	输出 Z
		J_4	K_4	C_4	J_3	K_3	C_3	J_2	K_2	C_2	J_1	K_1	C_1		
1	0000	0	1		1	1		1	1		1	1	↓	0001	
1	0001	0	1	↓	1	1		1	1	↓	1	1	↓	0010	
1	0010	0	1		1	1		1	1		1	1	↓	0011	
1	0011	0	1	↓	1	1	↓	1	1	↓	1	1	↓	0100	
1	0100	0	1		1	1		1	1		1	1	↓	0101	
1	0101	0	1	↓	1	1		1	1	↓	1	1	↓	0110	
1	0110	1	1		1	1		1	1		1	1	↓	0111	
1	0111	1	1	↓	1	1	↓	1	1	↓	1	1	↓	1000	
1	1000	0	1		1	1		0	1		1	1	↓	1001	
1	1001	0	1	↓	1	1		0	1	↓	1	1	↓	0000	1
1	1010	0	1		1	1		0	1		1	1	↓	1011	
1	1011	0	1	↓	1	1	↓	0	1	↓	1	1	↓	0100	1
1	1100	0	1		1	1		0	1		1	1	↓	1101	
1	1101	0	1	↓	1	1		0	1	↓	1	1	↓	0100	1
1	1110	1	1		1	1		0	1		1	1	↓	1111	

1	1111	1	1	↓	1	1	↓	0	1	↓	1	1	↓	0000	1

做出状态图，结果如图 1-5 所示，由状态图可知该计数器的模是 10.

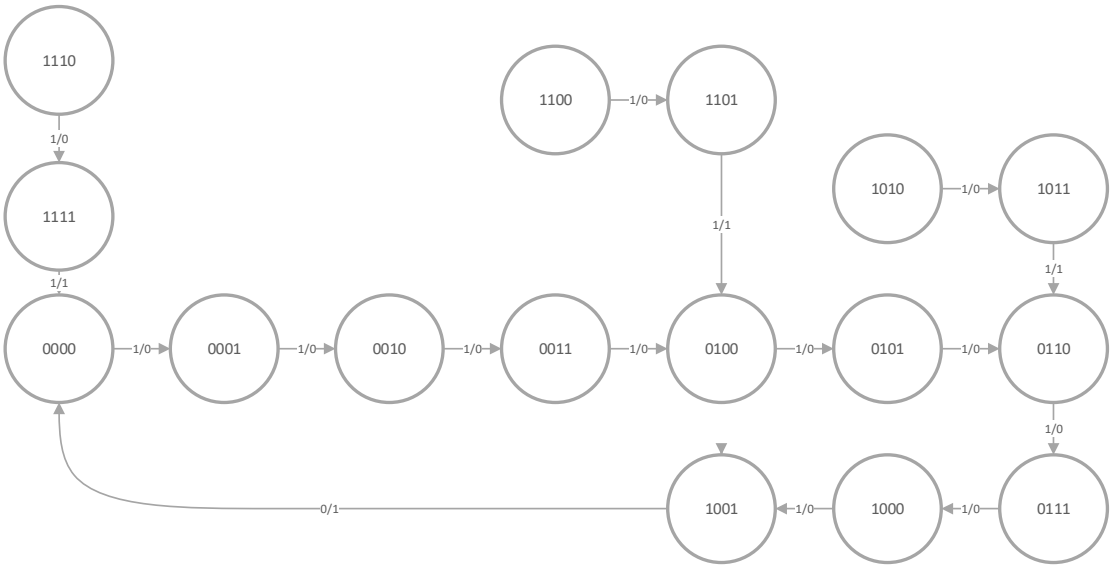


图 1-5 状态图

（A）源程序

程序 1-5 脉冲异步计数器

```
//JK 触发器模块
module JK(
    input IJ,
    input IK,
    input CI,
    output reg Q,
    output wire QB
    );
initial begin Q=0;end
always @(negedge CI) begin
    case({IJ,IK})
        2'b00:Q<=Q;
        2'b01:Q<=0;
        2'b10:Q<=1;
        2'b11:Q<=!Q;
        default:Q<=0;
    endcase
end
assign QB=!Q;
endmodule
```

```
//主模块
`timescale 1ns / 1ps
module M2(
    input X,
    output reg Z=0,
    output Q1,Q2,Q3,Q4
    );
wire QB4;
reg temp1;
JK JK1(.IJ(1),.IK(1),.CI(X),.Q(Q1));
JK JK2(.IJ(QB4),.IK(1),.CI(Q1),.Q(Q2));
JK JK3(.IJ(1),.IK(1),.CI(Q2),.Q(Q3));
JK JK4(.IJ(temp1),.IK(1),.CI(Q1),.Q(Q4),.QB(QB4));
always @ (*) begin
    temp1=Q2&Q3;
    Z=Q4&Q1&X;
end
endmodule
```

（B）仿真程序

程序 1-6　仿真程序

```
`timescale 1ns / 1ps
module M2_test();
reg X;
wire Z;
wire Q1,Q2,Q3,Q4;
M2 test(X,Z,Q1,Q2,Q3,Q4);
initial begin
    X=0;
end
always begin
    #5 X=!X;
end
endmodule
```

（C）引脚约束（绑定）程序

程序 1-7　约束程序

```
#Bank = 34, Pin name = IO_L24N_T3_34,          Sch name = LED0
set_property PACKAGE_PIN T8 [get_ports {Q1}]
set_property IOSTANDARD LVCMOS33 [get_ports {Q1}]
#Bank = 34, Pin name = IO_L21N_T3_DQS_34,      Sch name = LED1
set_property PACKAGE_PIN V9 [get_ports {Q2}]
set_property IOSTANDARD LVCMOS33 [get_ports {Q2}]
#Bank = 34, Pin name = IO_L24P_T3_34,          Sch name = LED2
set_property PACKAGE_PIN R8 [get_ports {Q3}]
set_property IOSTANDARD LVCMOS33 [get_ports {Q3}]
#Bank = 34, Pin name = IO_L23N_T3_34,          Sch name = LED3
set_property PACKAGE_PIN T6 [get_ports {Q4}]
set_property IOSTANDARD LVCMOS33 [get_ports {Q4}]
#Bank = 34, Pin name = IO_L12P_T1_MRCC_34,     Sch name = LED4
set_property PACKAGE_PIN T5 [get_ports {Z}]
set_property IOSTANDARD LVCMOS33 [get_ports {Z}]
#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}]
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

对 pipe3b 分别进行前仿真和后仿真,得到仿真图像如图 1-6 和图 1-7 所示。

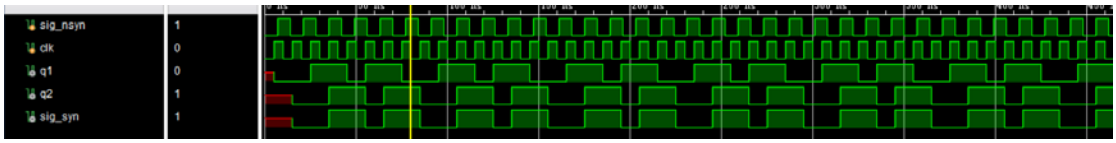


图 1-6 pipe3b 前仿真

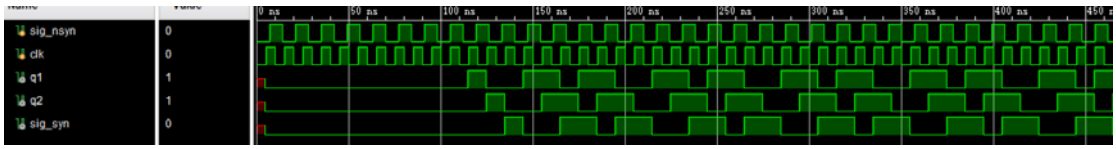


图 1-7 pipe3b 后仿真

可以发现初始一段的输出出现了问题且前仿真和后仿真的结果不一致。初始一段输出错误是因为没有给寄存器赋初始值，仿真结果不一致是因为 assign 使用了阻塞性赋值，修改为非阻塞行赋值即可。修改后的仿真结果如图 1-8 和图 1-9 所示，发现仿真结果一致。

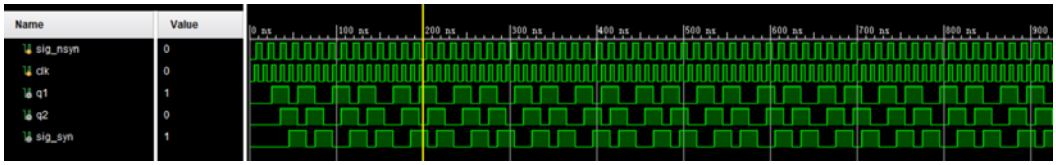


图 1-8 修改后的前仿真

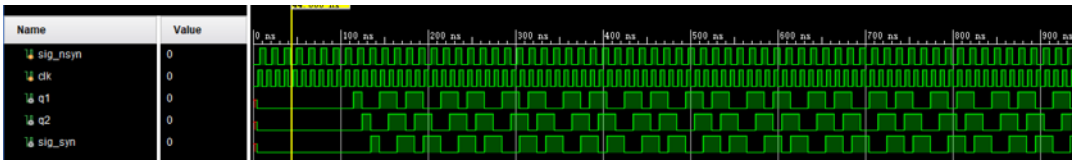


图 1-9 修改后的后仿真

（A）源程序

程序 1-8 修改后的 pipe3b

```
//修改后的程序
module pipe3b(
    input sig_nsyn,
    input clk,
    output reg q1=0,
    output reg q2=0,
    output reg sig_syn=0
      );

   always@(posedge clk) q2<=q1;
   always@(posedge clk) sig_syn<=q2;
   always@(posedge clk) q1<=sig_nsyn;
endmodule
//添加分频器后的程序
module pipe3b(
    input sig_nsyn,
    input clk,
    output reg q1=0,
    output reg q2=0,
    output reg sig_syn=0
      );
   parameter N=2;
      reg clk_N=0;
      reg [31:0] counter=0;
   always @(posedge clk)  begin    // 时钟上升沿
       counter<=counter+1;
       if(counter==N/2-1) begin
           counter<=0;
           clk_N<=~clk_N;
       end
   end
   always@(posedge clk_N) q2<=q1;
```

```
    always@(posedge clk_N) sig_syn<=q2;
    always@(posedge clk_N) q1<=sig_nsyn;
endmodule
```

（B）仿真程序

程序 1-9　仿真程序

```
`timescale 1ns / 1ps
module pipe3b_test();
reg sig_nsyn;
reg clk;
wire q1;
wire q2;
wire sig_syn;
pipe3b test(sig_nsyn,clk,q1,q2,sig_syn);
initial begin sig_nsyn=0;clk=0; end
always begin
    #5 clk<=!clk;
end
always begin
    #7 sig_nsyn<=!sig_nsyn;
end
endmodule
```

華中科技大學

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。

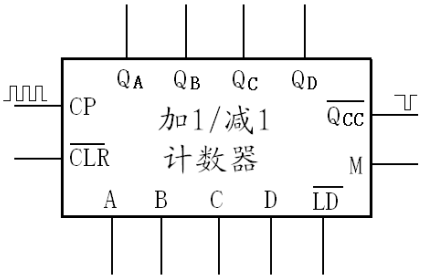


图2-1　4位二进制加法/减法计数器

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 $\overline{LD}$，清零端 $\overline{CLR}$；

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 $\overline{Qcc}$；

当 $\overline{CLR}$ 为 0 时，电路输出清零；

预制控制 $\overline{LD}$ =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 $\overline{Qcc}$ 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

```
`timescale 1ns / 1ps
module counter(CP,CLR,M,LD,D,C,B,A,QD,QC,QB,QA,Qcc);
input CP,CLR,M,LD;//CLR,LD 低电平有效
input D,C,B,A;
output reg QD,QC,QB,QA;
output reg Qcc=0;
initial begin {QA,QB,QC,QD}=0;end
always @ (posedge CP) begin
    Qcc<=!(CLR&LD&((QA&QB&QC&QD&M)|(~QA&~QB&~QC&~QD&~M)));
end
always @ (posedge CP) begin
    if(LD==0)
            {QD,QC,QB,QA}<={D,C,B,A};
    else if(CLR==0)
            {QD,QC,QB,QA}<=0;
    else begin
        if(M==1)
            {QD,QC,QB,QA}= {QD,QC,QB,QA}+1;
        else if(M==0)
            {QD,QC,QB,QA}= {QD,QC,QB,QA}-1;
```

```
        else {QD,QC,QB,QA}= {QD,QC,QB,QA};
    end
end
endmodule
```

（b）仿真程序

```
`timescale 10ns / 10ps
module counter_test();
reg CP,CLR,M,LD,D,C,B,A;
wire QD,QC,QB,QA,Qcc;
counter test(CP,CLR,M,LD,D,C,B,A,QD,QC,QB,QA,Qcc);
initial begin
{D,C,B,A}=4'b1101;
LD=1;
CLR=1;
CP=0;
M=1;
#20 CLR=0;
#2 CLR=1;
#20 LD=0;
#2 LD=1;
#2 M=0;
end
always begin
#0.5 CP=!CP;
end
endmodule
```

（c）引脚约束（绑定）程序

```
# Clock signal
#Bank = 35, Pin name = IO_L12P_T1_MRCC_35,    Sch name = CLK100MHZ
set_property PACKAGE_PIN E3 [get_ports CP]
set_property IOSTANDARD LVCMOS33 [get_ports CP]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports CP]
#Bank = 34, Pin name = IO_L24N_T3_34,            Sch name = LED0
set_property PACKAGE_PIN T8 [get_ports {QA}]
set_property IOSTANDARD LVCMOS33 [get_ports {QA}]
#Bank = 34, Pin name = IO_L21N_T3_DQS_34,        Sch name = LED1
set_property PACKAGE_PIN V9 [get_ports {QB}]
set_property IOSTANDARD LVCMOS33 [get_ports {QB}]
#Bank = 34, Pin name = IO_L24P_T3_34,            Sch name = LED2
set_property PACKAGE_PIN R8 [get_ports {QC}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {QC}]
#Bank = 34, Pin name = IO_L23N_T3_34,            Sch name = LED3
set_property PACKAGE_PIN T6 [get_ports {QD}]
set_property IOSTANDARD LVCMOS33 [get_ports {QD}]
#Bank = 34, Pin name = IO_L12P_T1_MRCC_34,       Sch name = LED4
set_property PACKAGE_PIN T5 [get_ports {Qcc}]
set_property IOSTANDARD LVCMOS33 [get_ports {Qcc}]
# Switches
#Bank = 34, Pin name = IO_25_34,                  Sch name = SW1
set_property PACKAGE_PIN U8 [get_ports {CLR}]
set_property IOSTANDARD LVCMOS33 [get_ports {CLR}]
#Bank = 34, Pin name = IO_L23P_T3_34,             Sch name = SW2
set_property PACKAGE_PIN R7 [get_ports {M}]
set_property IOSTANDARD LVCMOS33 [get_ports {M}]
#Bank = 34, Pin name = IO_L19P_T3_34,             Sch name = SW3
set_property PACKAGE_PIN R6 [get_ports {LD}]
set_property IOSTANDARD LVCMOS33 [get_ports {LD}]
#Bank = 34, Pin name = IO_L19N_T3_VREF_34,        Sch name = SW4
set_property PACKAGE_PIN R5 [get_ports {A}]
set_property IOSTANDARD LVCMOS33 [get_ports {A}]
#Bank = 34, Pin name = IO_L20P_T3_34,             Sch name = SW5
set_property PACKAGE_PIN V7 [get_ports {B}]
set_property IOSTANDARD LVCMOS33 [get_ports {B}]
#Bank = 34, Pin name = IO_L20N_T3_34,             Sch name = SW6
set_property PACKAGE_PIN V6 [get_ports {C}]
set_property IOSTANDARD LVCMOS33 [get_ports {C}]
#Bank = 34, Pin name = IO_L10P_T1_34,             Sch name = SW7
set_property PACKAGE_PIN V5 [get_ports {D}]
set_property IOSTANDARD LVCMOS33 [get_ports {D}]
```

（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

```
`timescale 1ns / 1ps
module counter_mod_8(CP,M,QD,QC,QB,QA,Qcc);
input CP,M;
output QD,QC,QB,QA;
output Qcc;
```

```
wire T;
wire LD;
wire [3:0] rst;
wire CP_N;
divider d(CP,CP_N);
assign rst=M?4'b0010:4'b1001;
assign LD=M?(QD&!QC&!QB&QA):(!QD&!QC&QB&!QA);
assign Qcc=M?(!QD&!QC&QB&!QA):(QD&!QC&!QB&QA);
counter
c1(CP_N,1,M,!LD,rst[3],rst[2],rst[1],rst[0],QD,QC,QB,QA,T);
endmodule
```

（b）仿真程序

```
`timescale 1ns / 1ps
module counter_mod_8_test();
reg CP,M;
wire QD,QC,QB,QA,Qcc;
counter_mod_8 test(CP,M,QD,QC,QB,QA,Qcc);
initial begin
CP=0;
M=1;
#1100 M=0;
end
always begin
    #10 CP=!CP;
end
endmodule
```

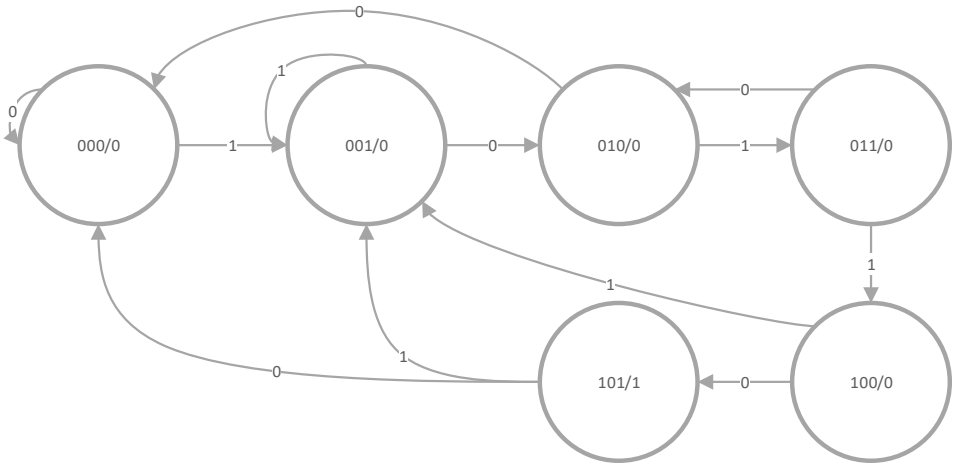
（c）引脚约束（绑定）程序

```
# Clock signal
#Bank = 35, Pin name = IO_L12P_T1_MRCC_35     Schname=CLK100MHZ
set_property PACKAGE_PIN E3 [get_ports CP]
set_property IOSTANDARD LVCMOS33 [get_ports CP]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports CP]
#Bank = 34, Pin name = IO_L24N_T3_34,            Sch name = LED0
set_property PACKAGE_PIN T8 [get_ports {QA}]
set_property IOSTANDARD LVCMOS33 [get_ports {QA}]
#Bank = 34, Pin name = IO_L21N_T3_DQS_34,        Sch name = LED1
set_property PACKAGE_PIN V9 [get_ports {QB}]
set_property IOSTANDARD LVCMOS33 [get_ports {QB}]
#Bank = 34, Pin name = IO_L24P_T3_34,            Sch name = LED2
set_property PACKAGE_PIN R8 [get_ports {QC}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {QC}]
#Bank = 34, Pin name = IO_L23N_T3_34,            Sch name = LED3
set_property PACKAGE_PIN T6 [get_ports {QD}]
set_property IOSTANDARD LVCMOS33 [get_ports {QD}]
#Bank = 34, Pin name = IO_L12P_T1_MRCC_34,       Sch name = LED4
set_property PACKAGE_PIN T5 [get_ports {Qcc}]
set_property IOSTANDARD LVCMOS33 [get_ports {Qcc}]
#Bank = 34, Pin name = IO_25_34,                 Sch name = SW1
set_property PACKAGE_PIN U8 [get_ports M]
set_property IOSTANDARD LVCMOS33 [get_ports M]
```

（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



（B）源程序

```
`timescale 1ns / 1ps
module sequential_detector(CP,RST,X,Z,c_state);
input CP,X,RST;
output reg Z;
output reg [2:0] c_state;
parameter ST0=0,ST1=1,ST2=2,ST3=3,ST4=4,ST5=5;
reg [2:0] n_state;
always @(posedge CP) begin
    if(RST)
        c_state<=ST0;
    else
        c_state<=n_state;
end
```

```
always @(c_state, X) begin
    case(c_state)
    ST0:begin
    Z<=0;
    if(X)
        n_state<=ST1;
    else n_state<=ST0;
    end
    ST1:begin
    Z<=0;
    if(X)
        n_state<=ST1;
    else n_state<=ST2;
    end
    ST2:begin
    Z<=0;
    if(X)
        n_state<=ST3;
    else n_state<=ST0;
    end
    ST3:begin
    Z<=0;
    if(X)
        n_state<=ST4;
    else n_state<=ST2;
    end
    ST4:begin
    Z<=0;
    if(X)
        n_state<=ST1;
    else n_state<=ST5;
    end
    ST5:begin
    Z<=1;
    if(X)
        n_state<=ST1;
    else n_state<=ST0;
    end
    default:begin
    Z<=0;
    n_state<=ST0;
```

```
    end
    endcase
end
endmodule
```

（C）仿真程序

```
`timescale 1ns / 1ps
module sequential_detector_test();
reg CP,X,RST;
wire Z;
wire [2:0] c_state;
sequential_detector test(CP,RST,X,Z,c_state);
initial begin
CP=0;
X=0;
RST=0;
#10 RST=1;
#10 RST=0;
#2
#20 X=1;//
#20 X=0;
#20 X=1;
#20 X=1;
#20 X=0;//独立得 10110 序列 1
#20 X=0;
#20 X=0;
#20 X=1;
#20 X=1;
#20 X=1;//
#20 X=0;
#20 X=1;
#20 X=1;
#20 X=0;//独立得 10110 序列 2
#20 X=1;
#20 X=1;
#20 X=0;//连续的 10110 序列
end
always begin
#10 CP=!CP;
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 {CP}]
set_property IOSTANDARD LVCMOS33 [get_ports {CP}]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets CP_IBUF]
#Bank = 34, Pin name = IO_L14P_T2_SRCC_34,       Sch name= SW15
set_property PACKAGE_PIN P4 [get_ports X]
set_property IOSTANDARD LVCMOS33 [get_ports X]
#Bank = 34, Pin name = IO_L23P_T3_34,            Sch name = SW2
set_property PACKAGE_PIN R7 [get_ports {RST}]
set_property IOSTANDARD LVCMOS33 [get_ports {RST}]
#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_L7P_T1_34,             Sch name = LED13
set_property PACKAGE_PIN U1 [get_ports {c_state[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {c_state[0]}]
#Bank = 34, Pin name = IO_L15N_T2_DQS_34,        Sch name = LED14
set_property PACKAGE_PIN R2 [get_ports {c_state[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {c_state[1]}]
#Bank = 34, Pin name = IO_L15P_T2_DQS_34,        Sch name = LED15
set_property PACKAGE_PIN P2 [get_ports {c_state[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {c_state[2]}]
```

（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

```
`timescale 1ns / 1ps
module comparator(a,b,c);
input a;
input b;
output c;
assign c=a^b;
endmodule
```

（b）仿真程序

```
`timescale 1ns / 1ps
module comparator_test();
reg a,b;
wire c;
comparator test(a,b,c);
```

```
initial begin
a=0;
b=0;
#50 a=1;
#50 b=1;
#50 a=0;b=1;
end
endmodule
```

（c）引脚约束（绑定）程序

```
#Bank = 34, Pin name = IO_L24N_T3_34,          Sch name = LED0
set_property PACKAGE_PIN T8 [get_ports c]
set_property IOSTANDARD LVCMOS33 [get_ports c]
# Switches
 #Bank = 34, Pin name = IO_L21P_T3_DQS_34,   Sch name = SW0
set_property PACKAGE_PIN U9 [get_ports a]
set_property IOSTANDARD LVCMOS33 [get_ports a]
#Bank = 34, Pin name = IO_25_34,              Sch name = SW1
set_property PACKAGE_PIN U8 [get_ports b]
set_property IOSTANDARD LVCMOS33 [get_ports b]
```

（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

```
`timescale 1ns / 1ps
module comparator_3_bit(A,B,F1,F2,F3);
input [2:0] A;
input [2:0] B;
output F1,F2,F3;
wire O1,O2,O3;
comparator C3(A[2],B[2],O3);
comparator C2(A[1],B[1],O2);
comparator C1(A[0],B[0],O1);
assign F1=(O3&{A[2]})|(!O3&O2&{A[1]})|(!O3&!O2&O1&{A[0]});
//A>B 包括：第 2 位不同且 A[2]=1 或第 2 位相同，第 1 位不同且 A[1]=1 或第 2、1 位相同，第 0 位不同且 A=1
assign F2=!O3&!O2&!O1;
assign F3=!F1&!F2;
endmodule
```

（b）仿真程序

```
`timescale 1ns / 1ps
module comparator_3_bit_test();
reg [2:0] A;
reg [2:0] B;
wire F3,F2,F1;
comparator_3_bit test(A,B,F1,F2,F3);
initial begin
A=0;B=0;
for(A=0;A<8;A=A+1)begin
    for(B=0;B<7;)
        #5 B=B+1;
    B=7;
    #5;
end
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 {A[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]
#Bank = 34, Pin name = IO_25_34,                 Sch name = SW1
set_property PACKAGE_PIN U8 [get_ports {A[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]
#Bank = 34, Pin name = IO_L23P_T3_34,            Sch name = SW2
set_property PACKAGE_PIN R7 [get_ports {A[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]

#Bank = 34, Pin name = IO_L19N_T3_VREF_34,       Sch name = SW4
set_property PACKAGE_PIN R5 [get_ports {B[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]
#Bank = 34, Pin name = IO_L20P_T3_34,            Sch name = SW5
set_property PACKAGE_PIN V7 [get_ports {B[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]
#Bank = 34, Pin name = IO_L20N_T3_34,            Sch name = SW6
set_property PACKAGE_PIN V6 [get_ports {B[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]
# LEDs
#Bank = 34, Pin name = IO_L24N_T3_34,            Sch name = LED0
set_property PACKAGE_PIN T8 [get_ports F1]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports F1]
#Bank = 34, Pin name = IO_L21N_T3_DQS_34,          Sch name = LED1
set_property PACKAGE_PIN V9 [get_ports F2]
set_property IOSTANDARD LVCMOS33 [get_ports F2]
#Bank = 34, Pin name = IO_L24P_T3_34,              Sch name = LED2
set_property PACKAGE_PIN R8 [get_ports F3]
set_property IOSTANDARD LVCMOS33 [get_ports F3]
```

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）给出 Verilog 设计的模 16 加 1/减 1 计数器的电路图（RTL Analysis 下“Schematic”截图）

图 2-2 为模 16 加 1/减 1 计数器的电路图

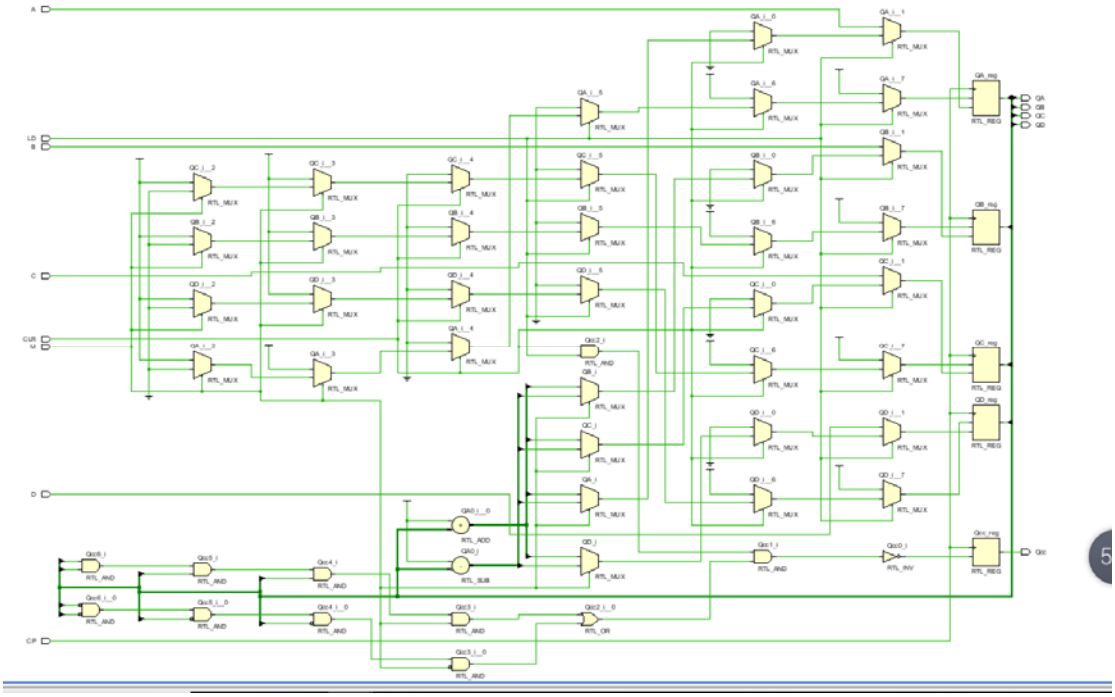


图 2-2 4 位二进制加法/减法计数器

（B）初值为 2 的模 8 计数器仿真结果截图（波形含 CP，M，Qa，Qb，Qc，Qd，Q 模 8 等）

图 2-3 为初值为 2 的模 8 计数器仿真,输入参数为时钟端 CP，加/减模式选择 M，输出端 QD，QC，QB，QA，Qcc。

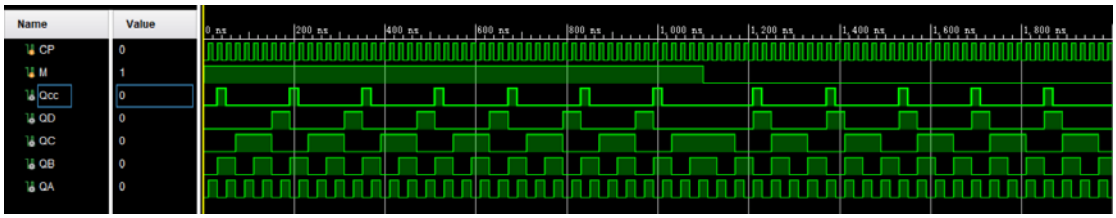


图 2-3 初值为 2 的模 8 计数器仿真

（C）开发板上的验证情况（主要记录：验证过程和结论）

综合实现后查看 FPGA 资源占用情况，如图 2-4 所示。

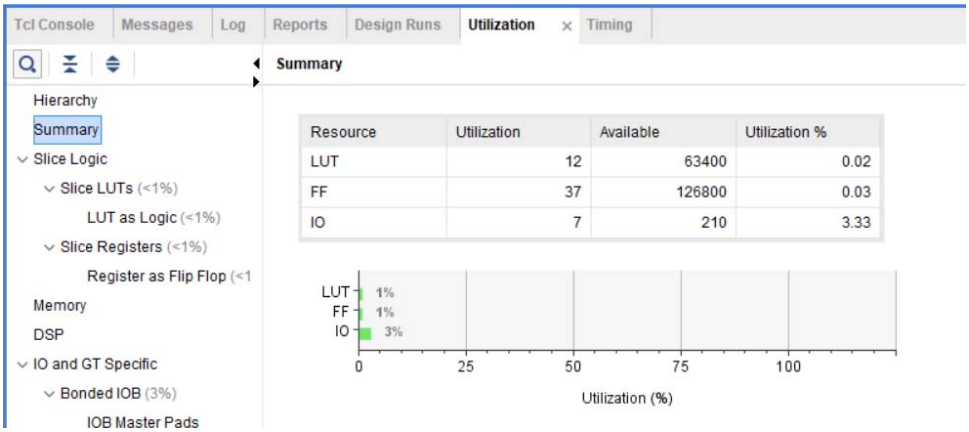


图 2-4 模 8 计数器 FPGA 资源占用情况

将生成的比特流文件后下载到板子上，将板子上提供的时钟信号经过分频后接到 CP 端，使用 led 灯作为当前计数状态，当 M 置 1 时，计数器从 2 加到 9 后置 2，对应的 led 灯亮起，每从 9 置 2 时接 Z 的 led 灯会亮。当 M 置 0 时，计数器从 9 减到 2 再置 9，对应的 led 灯亮起，每从 2 置 9 时接 Z 的 led 灯会亮。

（2）采用有限状态机(FSM) 实现序列检测器的实验结果记录

（A）给出 Verilog 设计的时序逻辑电路图（RTL Analysis 下“Schematic”截图）

图 2-5 为 10110 序列检测器电路图

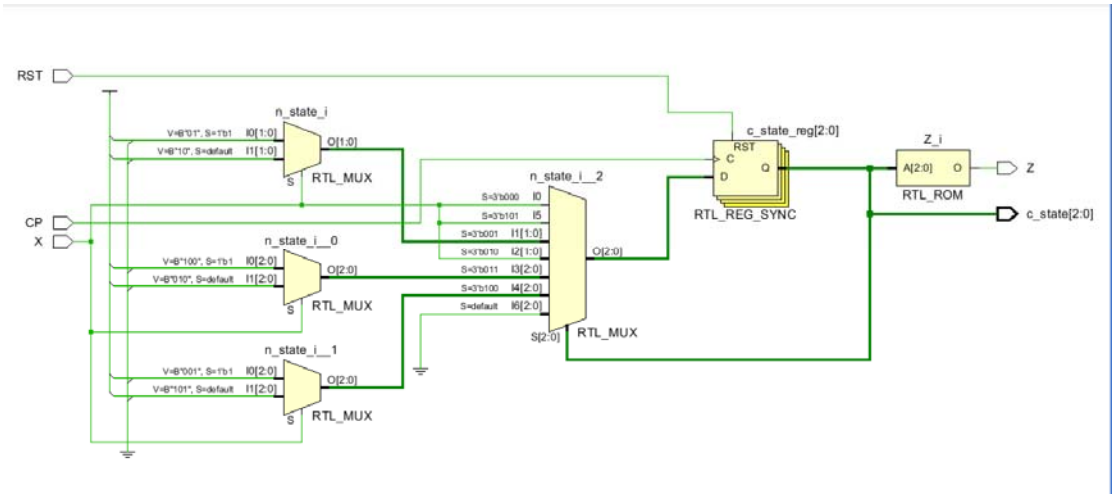


图 2-5　10110 序列检测器电路图

（B）仿真结果截图（波形含 clk，输入值，输出值等）

图 2-6 为 10110 序列检测器仿真，其中输入参数为时钟端 CP，输入序列 X，重置信号 RST；输出参数为检测信号 Z，当前状态 c_state.

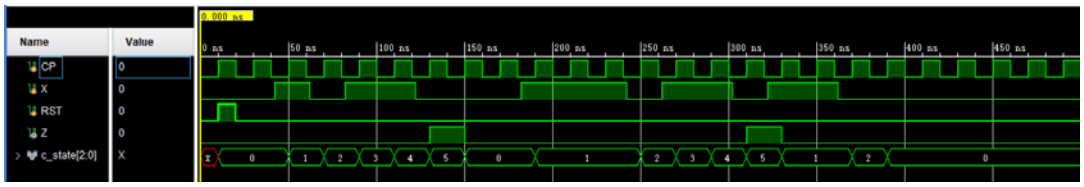


图 2-6　10110 序列检测器仿真

（C）开发板上的验证情况（主要记录：验证过程和结论）

综合实现后查看 FPGA 资源占用情况，如图 2-7 所示

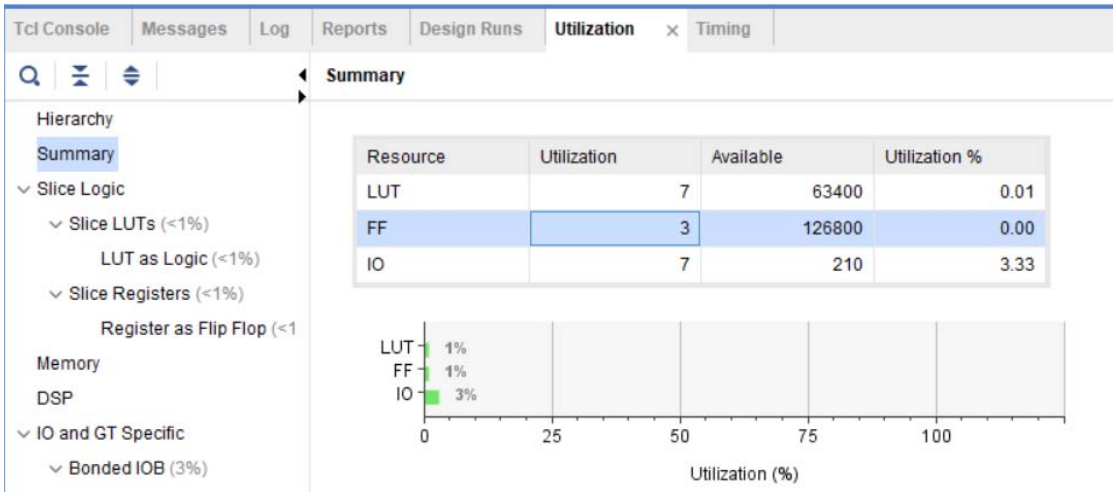


图 2-7 10110 序列检测器 FPGA 资源占用情况

采用开关作为时钟端和输入序列，采用 led 灯作为当前状态显示和输入 Z，当输入独立得 10110 时 Z 会输出 1 个脉冲，对用的 led 灯亮，当输入 10110110 时，即连续的 10110，此时 Z 只输入一个脉冲，led 灯也只亮一次。

（3）3 位二进制数值比较器的实验结果记录

（A）给出 Verilog 设计的 3 位二进制数值比较器的电路图（RTL Analysis 下“Schematic”截图）

图 2-8 为 3 位二进制数值比较器电路图

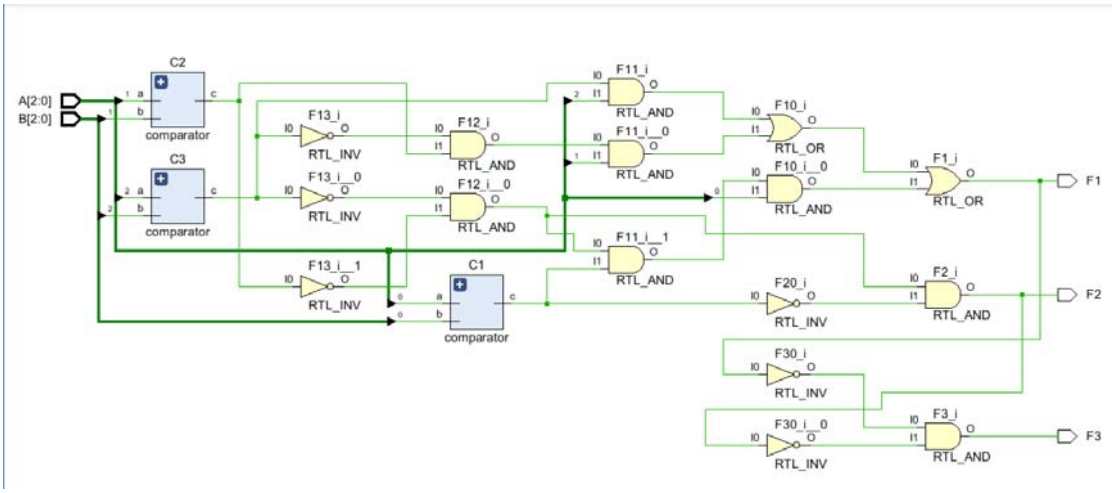


图 2-8　3 位二进制数值比较器电路图

（B）仿真结果截图（含输入值，输出值等）

图 2-9 为 3 位二进制数值比较器仿真，输入参数为参与比较的数 A,B，输出参数为比较结果 F3,F2,F1.

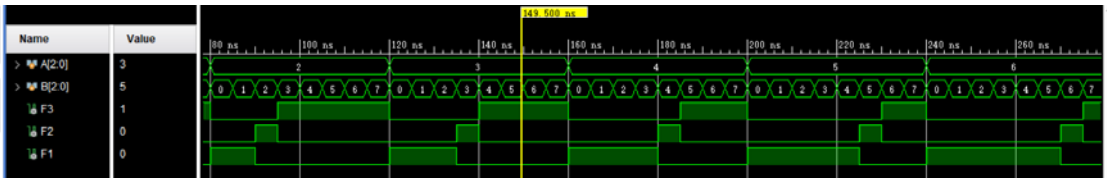


图 2-9　3 位二进制数值比较器仿真

（C）开发板上的验证情况（主要记录：验证过程和结论）

综合实现后查看 FPGA 资源占用情况，如图 2-10 所示

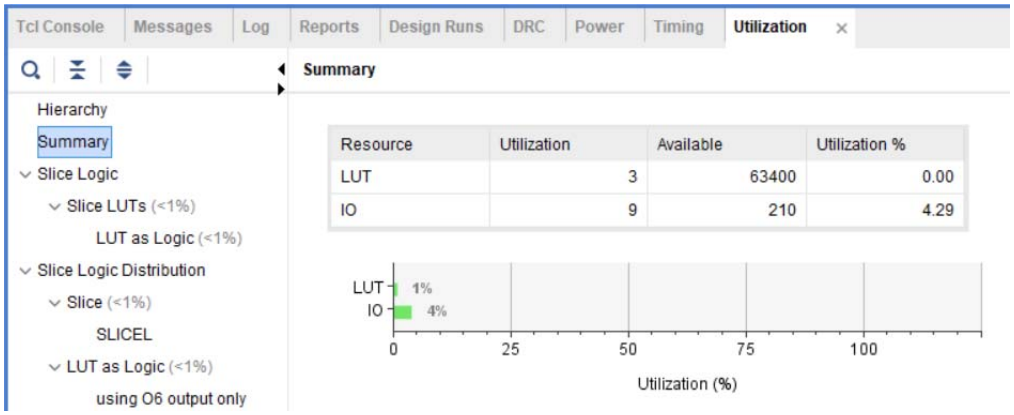


图 2-10　3 位二进制数值比较器 FPGA 资源占用情况

采用两组各 3 个开关作为输入 A,B，采用 3 个 led 灯作为输出 F3F2F1，当 A>B 时，输出为 001，当 A=B 时输出为 010，当 A<B 时输出为 100，下板验证正确。

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。

以 10110 序列检测器来说，在书写仿真程序时，使用 always 语句生成时钟信号 CP，然后使用 initial 语句给出输入序列 X，为了使输入序列和时钟端不同时变化，先使用#2 给出一个延时，使它们错开。然后观察生成的仿真图像，每当到始终上升沿对应的 X 就是输入序列，写出仿真图像上给出的输入序列，和理论的输入序列比对，发现均是 10110 0011 10110 110，匹配成功，然后观察出现不重复的 10110 序列后 Z 是否出现脉冲，经比对也符合，因此电路设计是正确的。

2．意见和建议

无