流水灯设计与实现实验报告

实验名称	流水灯的设计与实现			课程名	数字逻辑
姓　　名		学　　号		班　　级	
实验地点		实验日期	2018-5-1	组长电话	
指导老师	黄永刚	同组其他成员		Email	

一、实验内容（含实验原理介绍）：

1 实验环境

1.1 实验器材:

EES-338 是依元素科技基于 Xilinx Artix-7 FPGA 研发的便携式数模混合基础教学平台。EES-338 配备的 FPGA (XC7A35TCSG324-1C)具有大容量高性能等特点，能实现较复杂的数字逻辑设计；在 FPGA 内可以构建 MicroBlaze 处理器系 统，可进行 SoC 设计。该平台拥有丰富的外设，以及灵活的通用扩展接口。提供 8 个 LED 灯与 USB 接口

1.2 实现语言:

Verilog HDL 是目前应用最为广泛的硬件描述语言，可以用来进行数字电路的建模、仿真验证、时序分析、逻辑综合。可形式化地表示电路的行为和结构

2 实验原理

2.1 实验介绍:

流水灯即使得开发板上 8 个 LED 灯依次被点亮并来回滚动。

2.2 原理介绍:

2.3.2 时序电路原理

时序电路的行为是由输入、输出和电路当前状态决定的。输出和下一状态是输入和当前状态的函数。通过对时序电路进行分析，可以得到关于输入、输出和状态三者的时序的一个合理描述。

2.3.2 分频器原理

把输入的信号作为计数脉冲，由于计数器的输出端口是按一定规律输出脉冲的，所以对不同的端口输出的信号脉冲，就可以看作是对输入信号的”分频“。

2.3.3 显示原理

LED 灯处于高电平时灯亮，低电平灯不亮。通过在时钟信号下对 8 个流水灯进行不同的赋值，可以实现流水灯的显示功能。

二、实验目的：

1、熟悉一次完整的 FPGA 开发流程，从新建工程，代码设计，综合实现，管脚约束，下载 FPGA 程序。熟悉 Verilog HDL 语言并能建立基本的逻辑部件，在 Vivado 平台进行输入、编辑、调试、行为仿真。

2、熟悉 Vivado 开发环境，Verilog 语言编程基本框架。

3、熟悉 Xilinx EES-338 口袋计算机硬件平台。

4、运用 Xilinx Vivado 设计套件将设计验证后的代码下载到开发板上，并在开发板上验证

5、在开发板上实现流水灯。即上电以后，LED0~LDE7 逐次点亮，然后熄灭，向右或者向左滚动，按一下 rst 复位键，LED0~LED7 全亮，按住 run 键，则再次滚动。流水灯从左向右滚动，滚到最右，自动向左滚，滚到左侧，再向右滚动。

三、涉及实验的相关情况介绍（包含使用软件或实验设备、设计思路等情况）：

1、使用软件 vivado2017.2 设计套件

2、实现语言 Verilog

3、VMware horizon client 云平台

4、Xilinx EES-338 口袋计算机硬件平台

5、设计思路

采用自上而下的设计思路，将流水灯实验分为 4 个模块：分频模块，显示模块，控制模块，总模块--集成模块

5.1 分频模块 divider

首先进行分频操作，将原开发板 100MHz 的时钟转换成更低的频率。

5.2 控制器模块 controller

当刚通电或按下 reset 按键后，当前状态 state 为 stop(0)，按下 run 按键后偶遇改变当前状态为 runing(1)

5.3 展示模块 show

根据分频后的时钟对 LED 灯进行操作，其中高电平时灯亮，低电平灯不亮。若处于初始状态则灯全亮。否则每过一个时钟首先判断是否在向左滚动情况下到达最左端以及是否向右滚动到达最右端。若没有则则当前亮 LED 灯的下一个点亮，并且熄灭原亮灯。否则改变当前滚动方向。

6、时序电路设计流程

6.1 规格说明

6.2 形式化 - 得到状态表或状态图

6.3 状态分配- 给状态分配二进制码

6.4 决定触发输入方程 - 选择触发器类型 并且根据状态表中的下一状态推导出触发器方程

6.5 决定输出方程 - 根据状态表中的输出推导出输出方程

6.6 优化 - 对方程进行优化

6.7 工艺映射 - 从方程中得到电路并且映射到触发器和门工艺

6.8 验证 -验证最终设计的正确性

四、实验步骤和测试方法

1、安装 VMWARE horizon client

2、分析问题，规范化

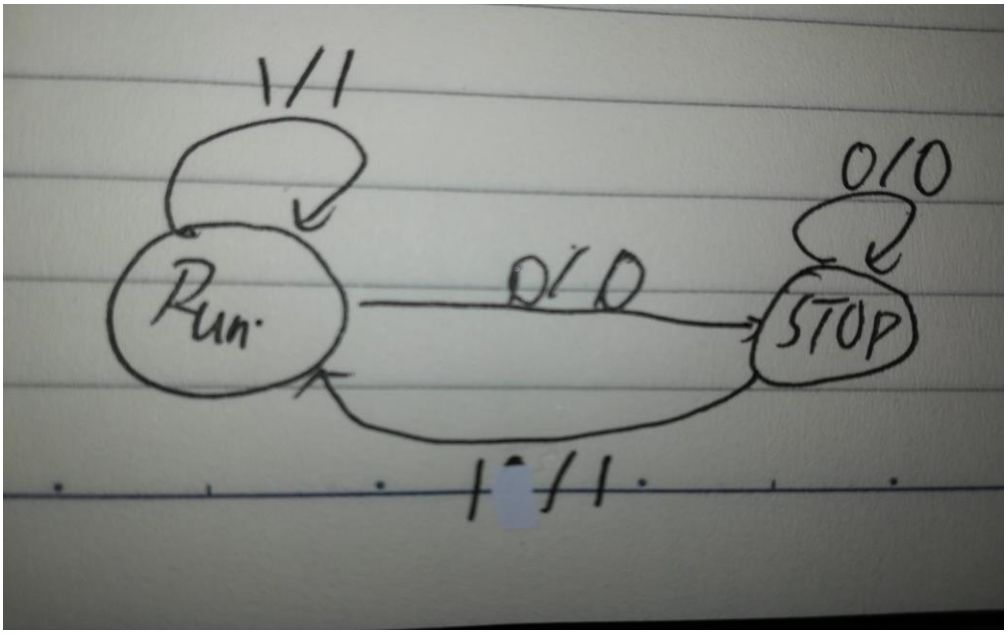
2.1 一个 reset 开关控制是否清零

2.2 一个 run 开关控制是否开始

2.3 run 条件下流水灯到达首端与末端后会折返

3、形式化

3.1 状态图绘制



3.2 状态表绘制

present state	next state		output	
	x=0	x=1	x=0	x=1
Run	STOP	Run	0	1
STOP	STOP	Run	0	1

3.3 状态分配

格雷码 Run=1, STOP=0

present state	next state		output	
	x=0	x=1	x=0	x=1
0	0	1	0	1
1	0	1	0	1

4、确定触发方程

D=X

5、确定输出方程

Y=X

6、优化

无需要优化部分

7、设计源代码

采用自顶向下的设计方式，从顶层模块逐个分割成显示模块 show，控制模块 controller，时钟分频模块 divider 以及集成模块 lightcontrol

分频模块:Input 为 clk，Output 为 clk_out。负责时钟分频。

显示模块:Input 为:state；Output 为 led。 负责 8 位 LED 灯的显示

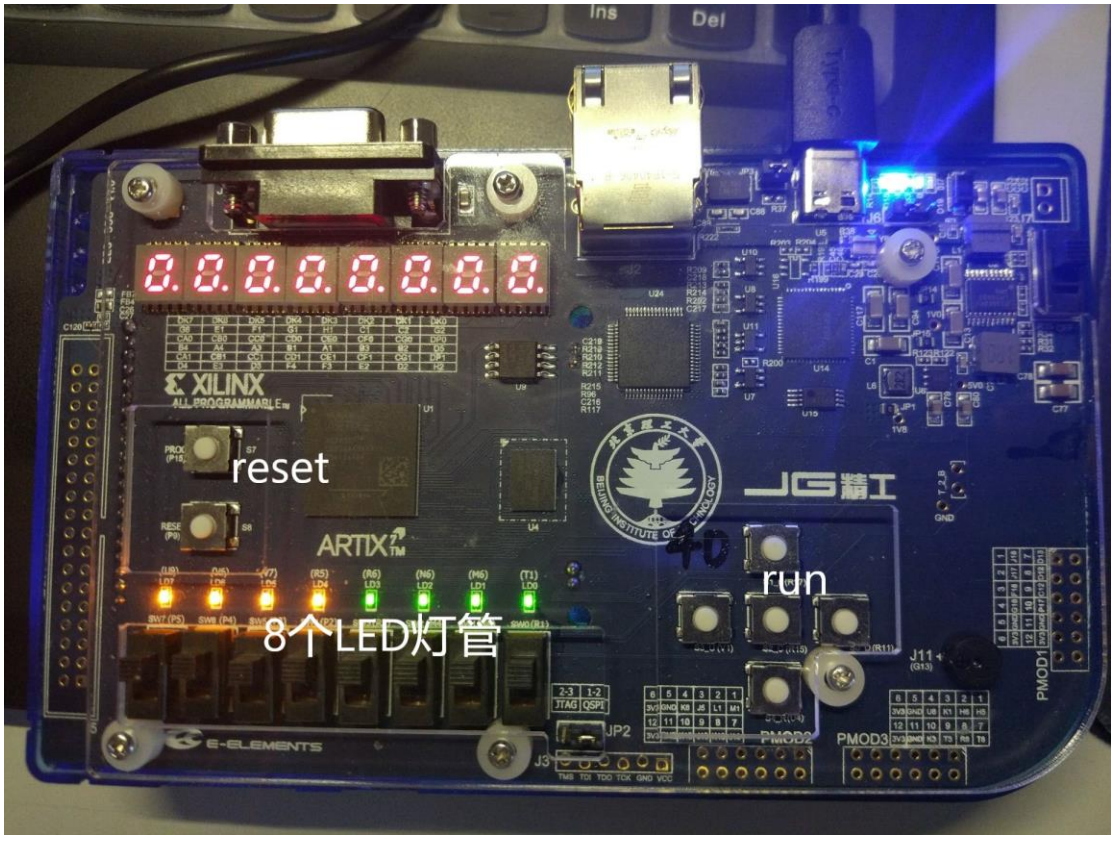
控制模块:Input 为 reset,run, clk;Outputw 为 state。负责状态转移。

集成模块:把分割成的 3 个模块功能集成至总模块

8、配置管脚

选择 8 个 LED 灯以及专用按键 PROG 键(P15)以及通用按键中的中按键(R15)进行管脚映射

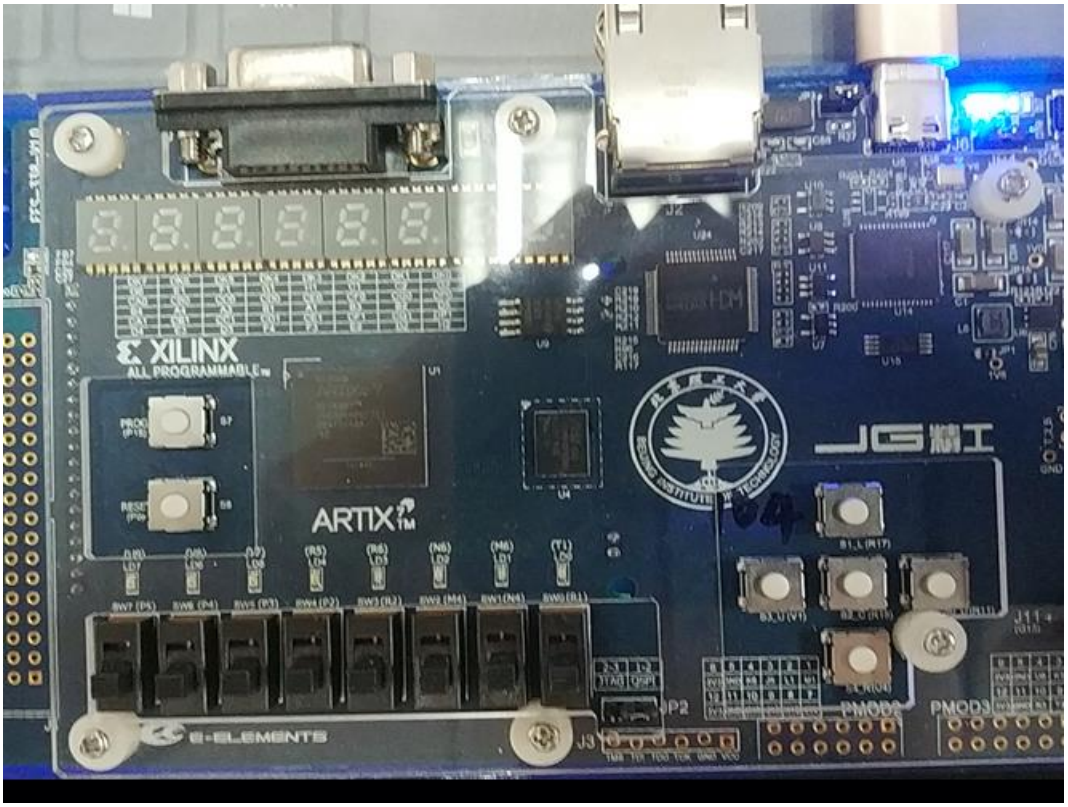
```
set_property PACKAGE_PIN T5 [get_ports clk]
set_property PACKAGE_PIN R15 [get_ports run]
set_property PACKAGE_PIN P15 [get_ports reset]
set_property PACKAGE_PIN U9 [get_ports {led[7]}]
set_property PACKAGE_PIN V6 [get_ports {led[6]}]
set_property PACKAGE_PIN V7 [get_ports {led[5]}]
set_property PACKAGE_PIN R5 [get_ports {led[4]}]
set_property PACKAGE_PIN R6 [get_ports {led[3]}]
set_property PACKAGE_PIN N6 [get_ports {led[2]}]
set_property PACKAGE_PIN M6 [get_ports {led[1]}]
set_property PACKAGE_PIN T1 [get_ports {led[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports run]
```



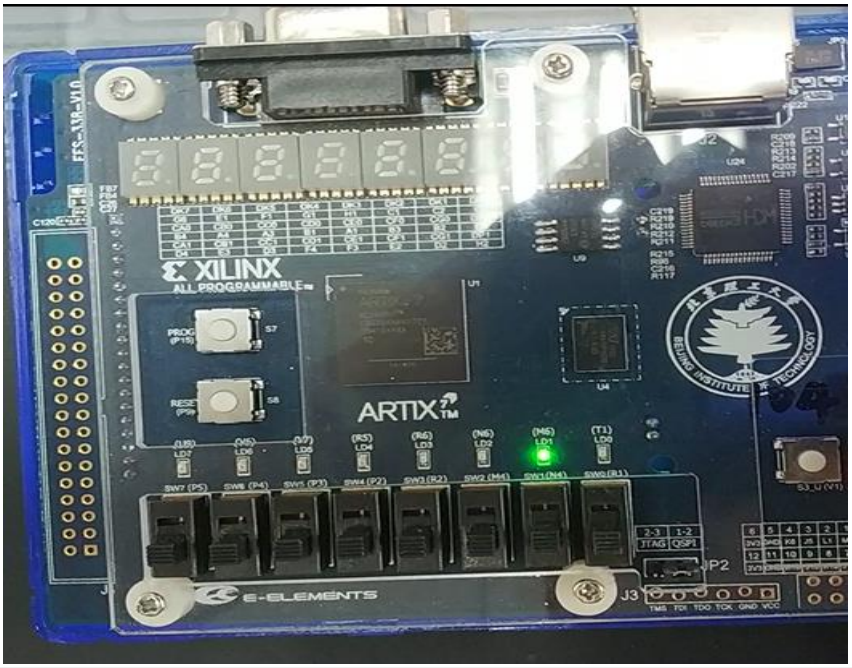
五、实验结果（含程序、数据记录及分析和实验总结等，可附页）：

1 对开发板的操作

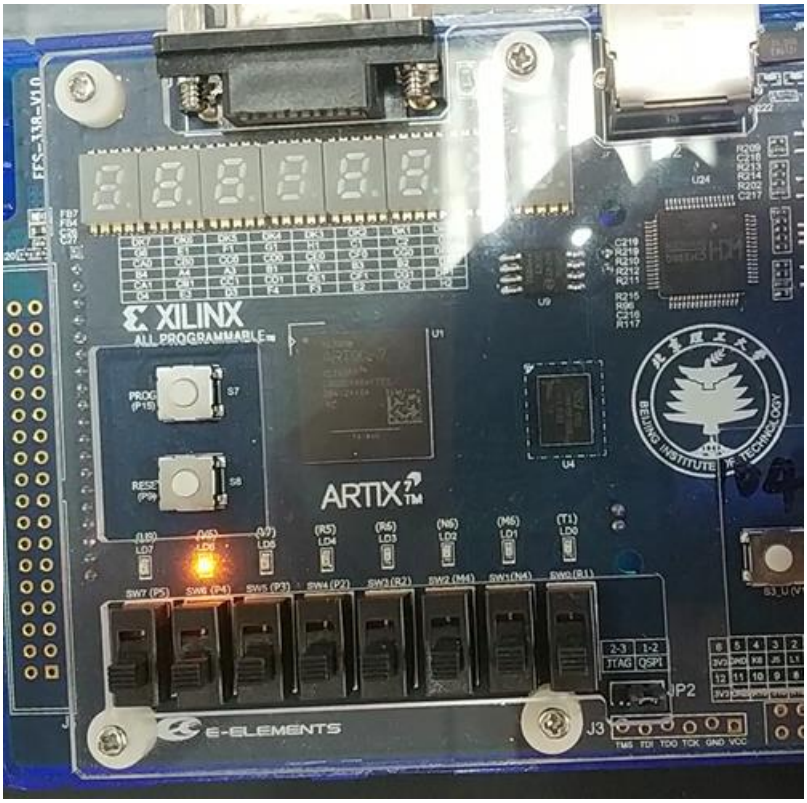
1.1 初始状态灯全灭



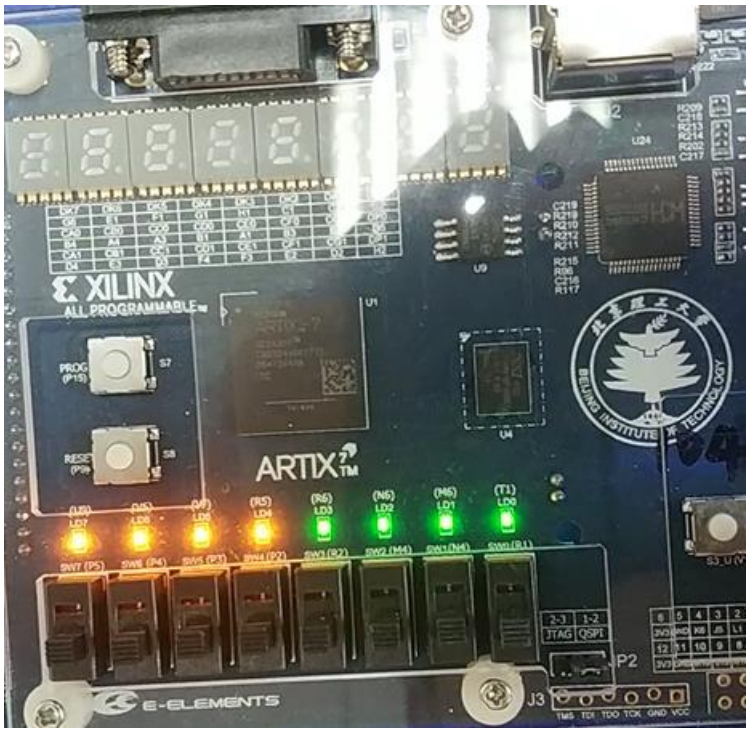
1.2 点击 run 按键后从 LED0 开始开始向左滚动



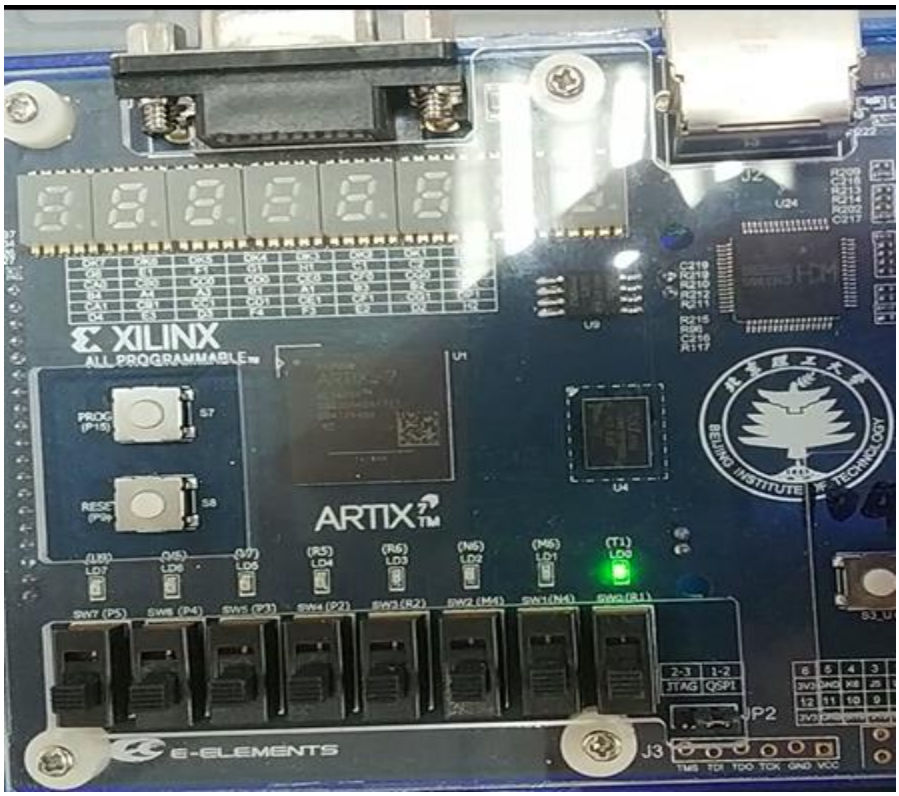
1.3 到达左端后返回并向右滚动



1.4 按下 reset 键后全部灯亮



1.5 再次按下 run 按键后从 LED0 开始向左滚动



2 源程序

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2018/05/14 22:45:12
// Design Name:
// Module Name: lightcontrol
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module  lightcontrol(
        input  clk,
        input  reset,
        input  run,
        output  [7:0]led
        );
        wire  clk_out;
        wire  state;

        clk_divider  divder(
        .clk(clk),
        .reset(reset),
        .clk_out(clk_out)
        );

        controller  cler(
                .run(run),
                .reset(reset),
                .clk(clk),
                .state(state)
        );

        shower  my_shower(
                .clk(clk_out),
                .reset(reset),
.run(run),
                .led(led)
        );

endmodule

module  clk_divider(
        input  clk,
        input  reset,
        output  reg  clk_out
);

parameter  count_limit  =  499999;
reg  [31:0]  count;

always  @(posedge  clk  or  posedge  reset)
begin
        if(reset)begin
                count  <=  0;
```

```
                clk_out  <=  0;
        end
        else
                if(count  ==  count_limit)begin
                        count  <=  0;
                        clk_out  <=  !clk_out;
                end
                else  begin
                        count  <=  count  +  1;
                end
end
endmodule

module  controller(
        input  run,
        input  reset,
        input  clk,
        output  reg  state
);

parameter  Running  =  1'b1;
parameter  STOP  =  1'b0;

reg  state,nextstate;

always  @(posedge  clk  or  posedge  reset)
begin
        if(reset)
                state  <=  STOP;
        else
                state  <=  nextstate;
end

always  @(*)
begin
        if(state  ==  STOP)begin
                if(reset)begin
                        nextstate  =  STOP;
                        state  =  STOP;
                end
                else  begin
                if(run)begin
                        nextstate  =  Running;
                        state  =  Running;
```

```
                        end
                    end
                end
        else  begin
                if(run)begin
                        nextstate  =  Running;
                        state  =  Running;
                end
                else  begin
                if(reset)begin
                        nextstate  =  STOP;
                        state  =  STOP;
                        end
                end
        end
end
endmodule


module  shower(
        input  clk,
        input  reset,
        input  run,
        output  reg  [7:0]led
);
reg  flag=1;
reg  [3:0]count;
parameter[3:0]  delay=4'd0100;

always@(posedge  clk  or  posedge  reset)
begin
if(reset)
count<=0;//assign
else  if(count==delay)
count<=0;//assign
else
count<=count+1;
end
always@(posedge  clk  or  posedge  reset)
begin
if(reset)  begin
led<=8'b1111_1111;//from  the  first  one
end
else  if(run)  begin
```

```
led<=8'b0000_0001;
end
else
begin
if(flag==1)  begin
if(led==8'b0111_0000)  begin
flag<=0;
end
else  if(count==delay)  begin
led<={led[6:0],led[7]};  end
else  begin
led<=led;  end
end
else
begin
if(led==8'b0000_1110)  begin
flag<=1;
end
else  if(count==delay)  begin
led<={led[0],led[7:1]};  end
else  begin
led<=led;  end
end
end
end

endmodule
```

六、实验分工、实验过程中问题的处理、讨论和建议，收获和体会

1、实验分工

1.1 王兴峥:分析问题，绘制状态图表，编写代码，验证调试，撰写报告

1.2 谭超:辅助调试

1.3 李昊宸:辅助调试

2、实验中的问题

2.1 问题：配置管脚时 reset 键 P9 无法使用

方法：使用 prog 键代替 reset，再吧 run 映射到通用按钮

2.2 问题：分频后发现流水灯流速过快

方法：分频后的频率依旧过高，使用更大的间隔进行分频，使得得到更低的分频后的频率

3、实验体会

流水灯作为一个时序电路设计的入门实验，为后面我负责实验三的时序电路设计、编程、报告以及改进调试奠定了基础。状态图表，触发函数等也为后面实验三计时器的状态机图、状态表绘制做了铺垫。让我进一步认识到理论学习与实际操作的不同之处，一部分是理论的掌握，并通过规范化形式化等一系列操作对开发带来的便捷性。以及实际需要开发调试的经验，不能紧紧停留于理论学习。

七、提交资料列表和说明

提交资料表:

目录名：

目录下文件：**1、实验一 实验报告.doc**

2、实验一工程源代码.txt

八、参考文献（可选）