西安电子科技大学
XIDIAN UNIVERSITY

基于 FPGA 的数字系统设计

实验报告

实验名称：LAB2_3

任课教师：沈沛意老师

学号姓名：

提交日期：

一、实验介绍

Spartan 3E 开发板实验:

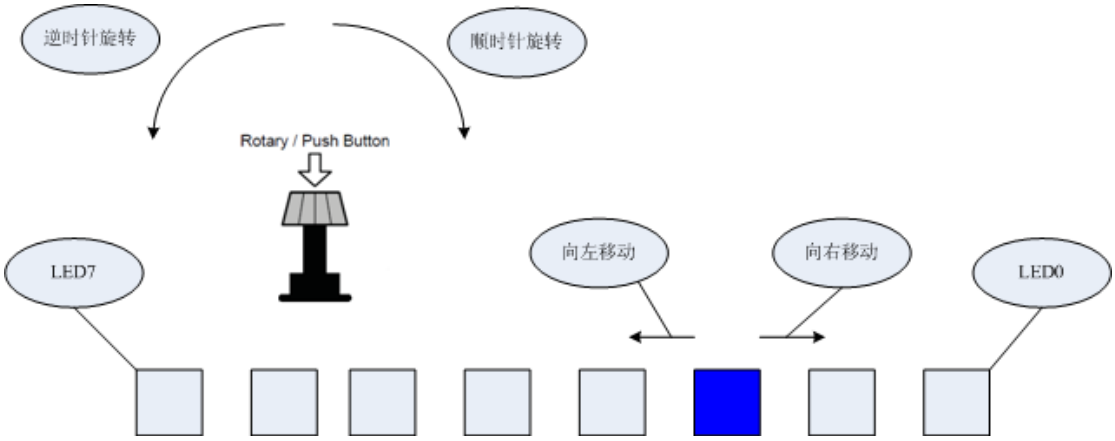
旋转开关控制二极管轮流发光

二、实验目标

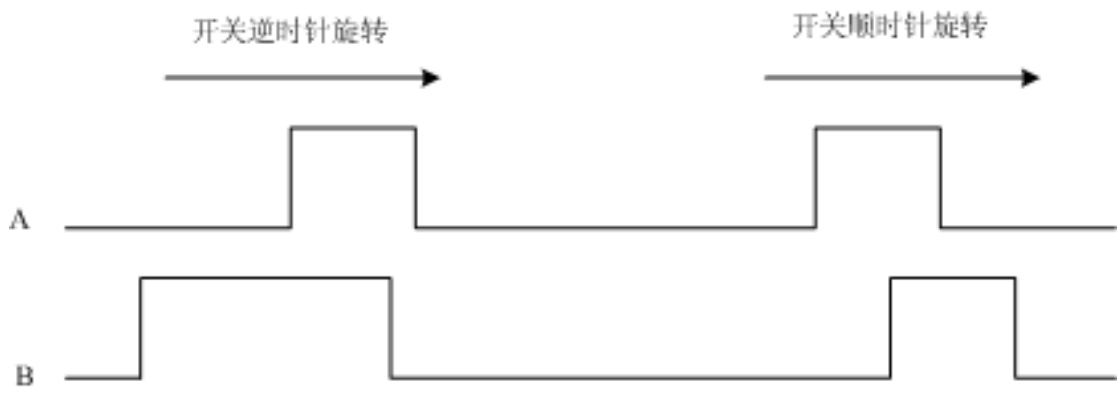
- 熟悉 ISE 软件，会使用 ISE 软件进行设计和仿真
- 掌握 Spartan3E 开发板的配置流程

三、实验原理

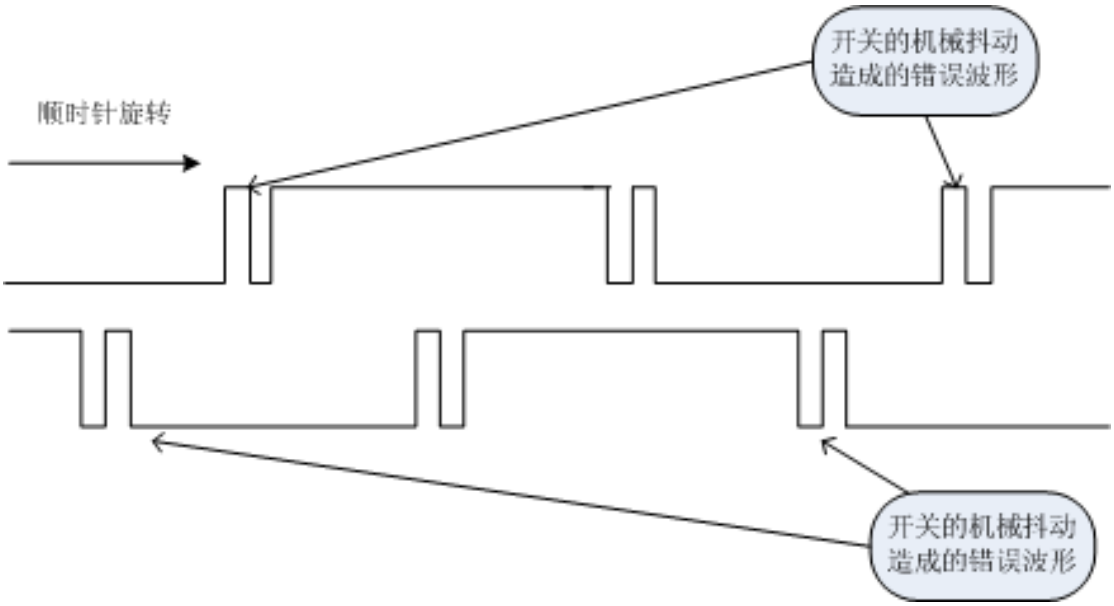
Spartan 3E 开发板上有 8 个并排放置的发光二极管 LED7 ~ LED0，以及一个旋转开关。实验要求使用旋转开关控制二极管轮流发光，旋转开关顺时针或逆时针转动控制发光二极管右移或左移。



当旋转开关输出的开关 B 信号为高电平时，A 信号的上升沿表示逆时针旋转；当开关 B 信号为低电平时，开关 A 信号的上升沿表示顺时针旋转。



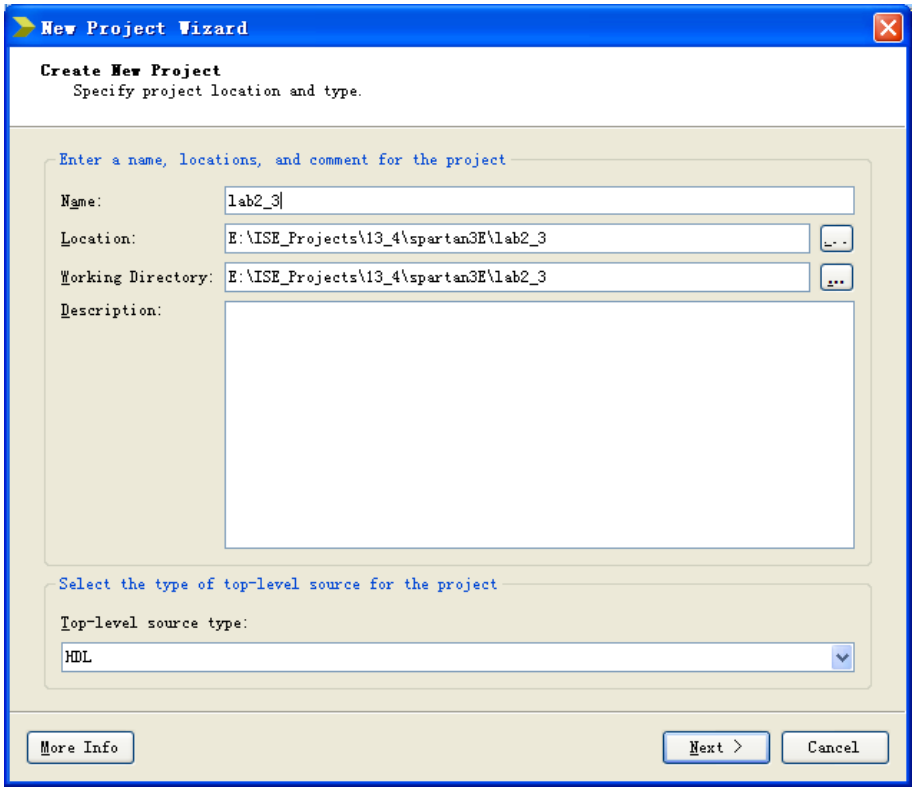
但是旋转开关的输出可能存在机械抖动现象，在设计时必须加入消除抖动的措施。



四、实验过程

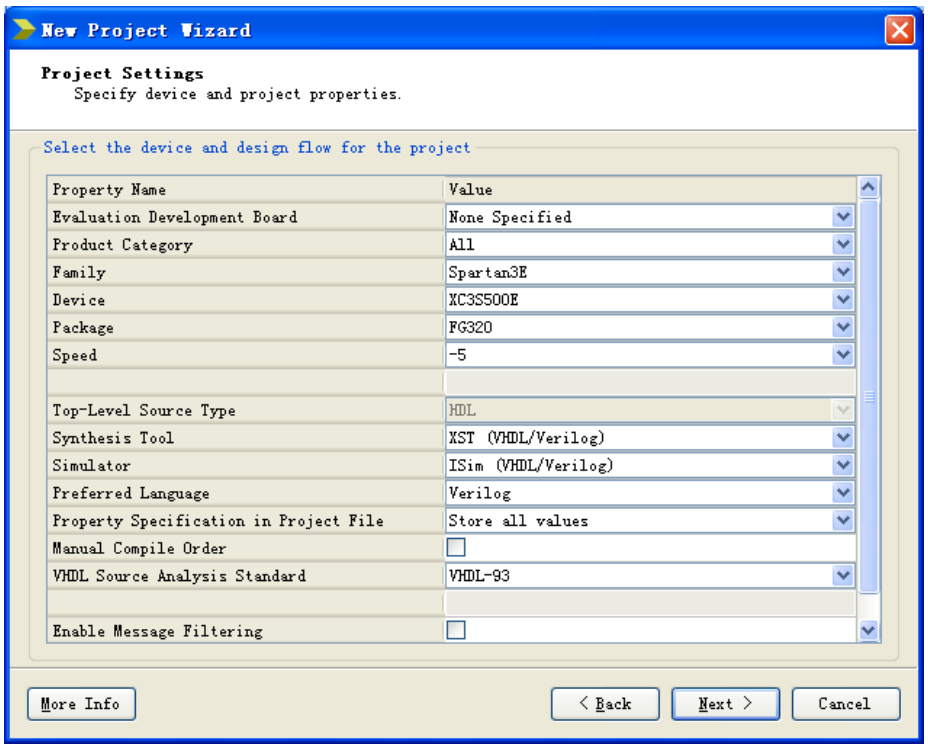
1. 启动 ISE 创建一个新的工程

- 打开 ISE 集成环境
- 打开创建新工程界面



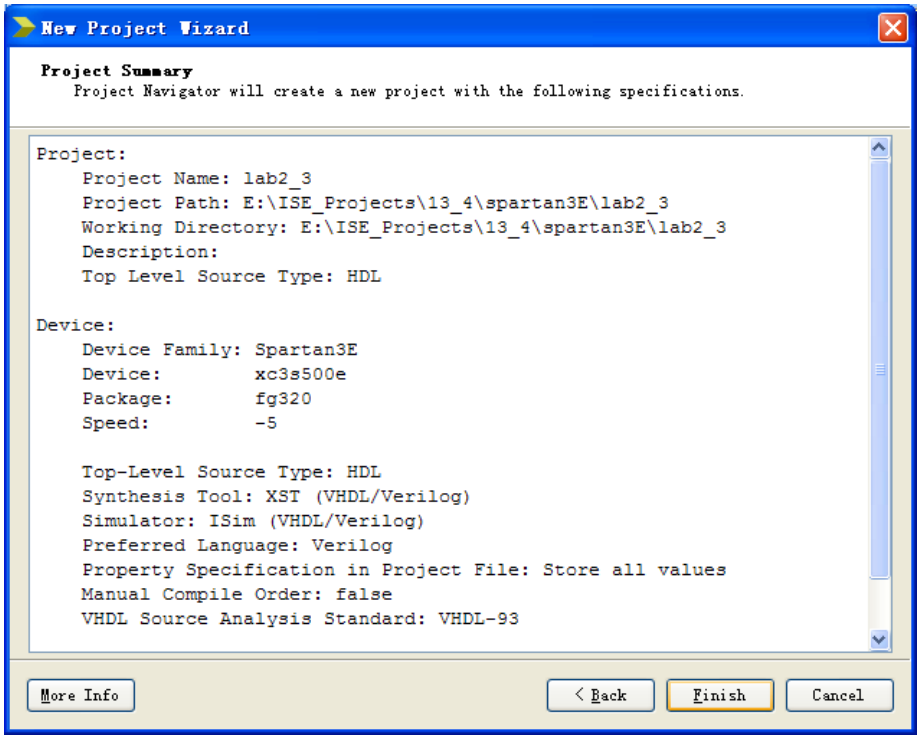
命名为 lab2_3

- 设置参数: 芯片型号为 Spartan3E XC3S500E 芯片，FG320 封装



- 单击 Next，进入工程信息页面，确认无误后，点击 Finish 完成工程

的创建



2. 设计输入

- 选择 Project->New Source,在左侧文件类型中选择 Verilog Module，并输入 Verilog 文件名 rotary_led
- 单击 Next 进入模块定义窗口，在其中填入模块端口定义。
- 单击 Next 进入下一步，点击 Finish 完成创建。后续将代码编写完整

3. 综合与实现

- 在工程管理区的 view 中选择 Implementation，然后在过程管理区双击 Synthesize-XST，就可以开始综合过程
- 添加用户约束文件（UCF)：约束文件的作用是将模块的输入输出端口绑定到 FPGA 相应的外部引脚上。
- 选择 Project->New Source，在弹出的对话框中，左侧文件类型选中 Implementation Constraints File，右侧填写文件名 rotary_led

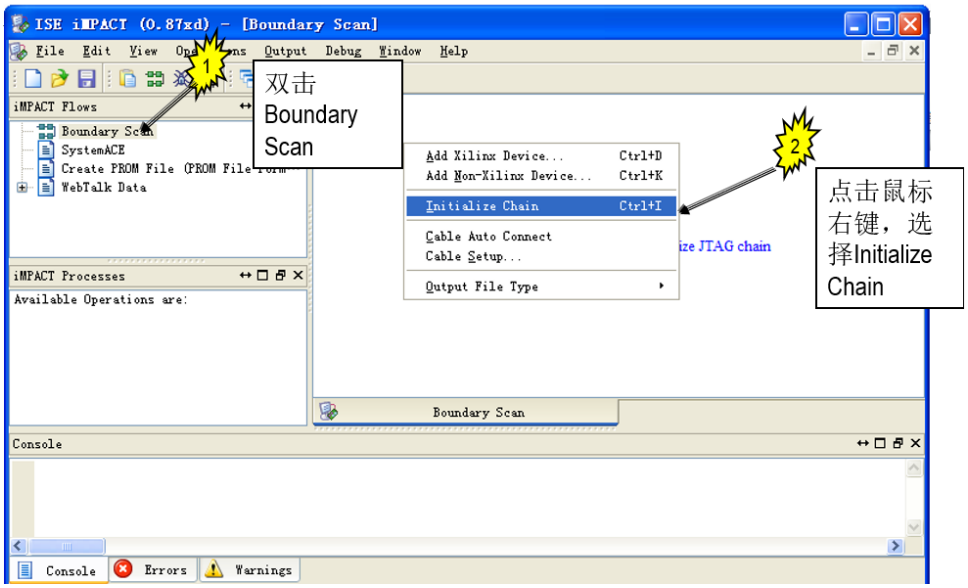
- 在工程管理区内选中 UCF 文件，再点击 Edit Constraints 编辑约束文件
- 综合完成后，下一个步骤就是实现(Implementation) 。实现主要分为 3 个步骤：翻译(Translate) 、映射(Map) 与布局布线(place & Route)。
- 在 ISE 中，执行实现过程，会自动执行翻译、映射和布局布线过程：也可单独执行。在过程管理区双击 Implement Design 选项，就可以自动完成实现的 3 个步骤

4. 器件配置

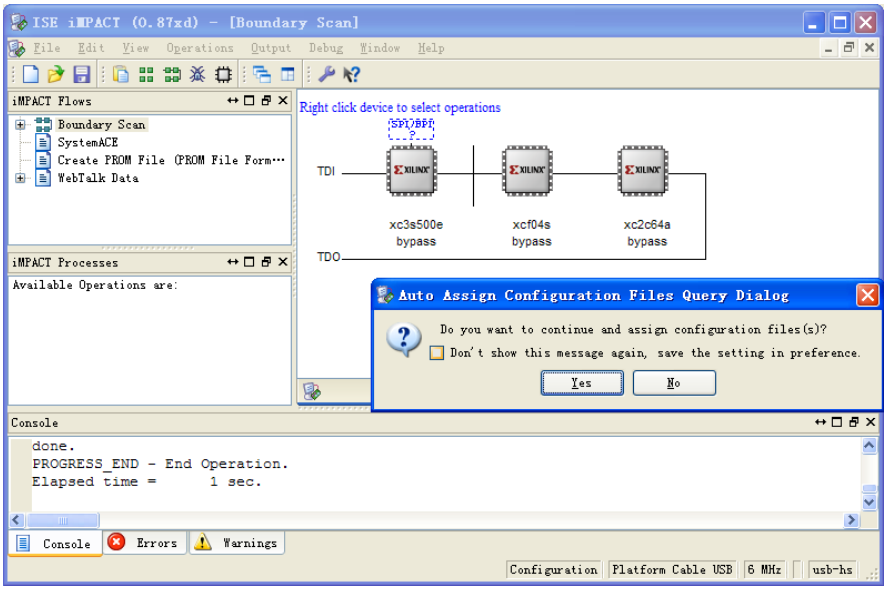
器件配置是 FPGA 开发最关键的一步，只有将 HDL 代码下载到 FPGA 芯片中，才能进行调试并最终实现相应的功能。首先我们必须生成能下载到硬件中的二进制比特文件。

对 Spartan3E 开发板进行配置时，不需要使用 Digilent Adept 软件，使用 ISE 自带的 iMPACT 即可完成配置。

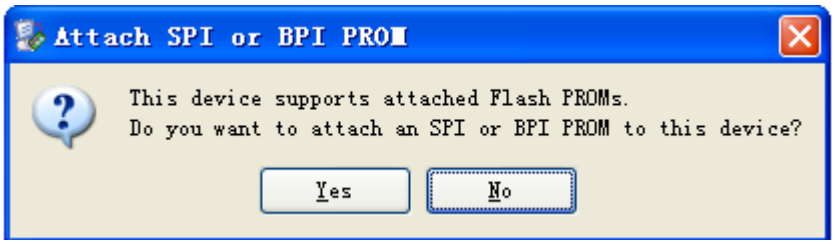
- 双击过程管理区的 Generate Programming File，ISE 就会为设计生成相应的二进制比特文件。
- 首先连接并启动开发板
- 在工程管理区域内选择 Configure Target Device
- 当出现弹出对话框时，选择 OK，启动 iMPACT GUI



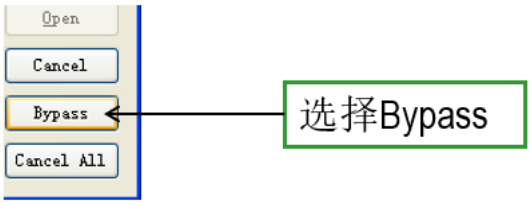
- 在弹出的对话框中选择 Yes，并指定之前生成的比特流文件



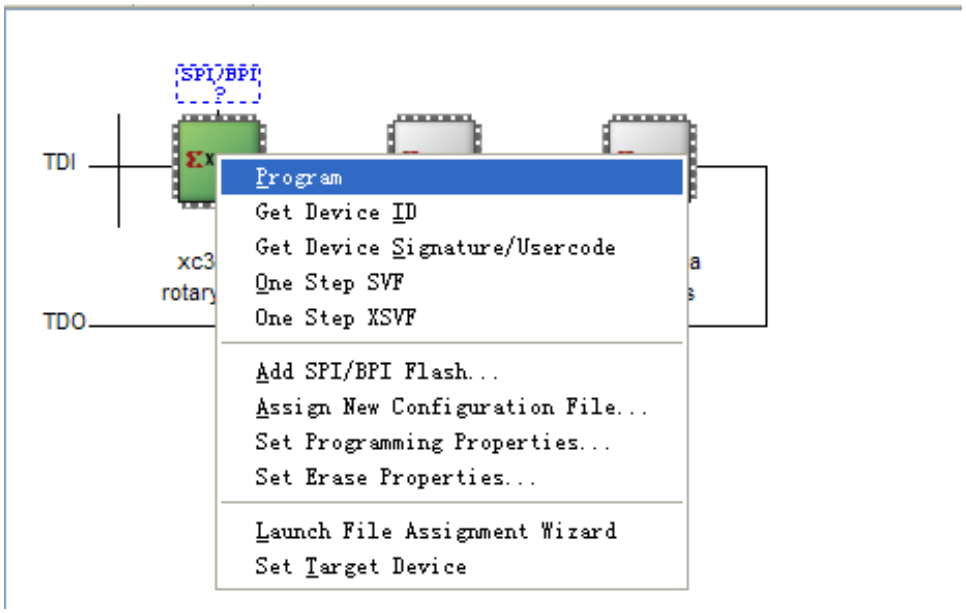
- 之后弹出的对话框询问是否添加 Flash PROM 配置文件，在这里选择 No



- 随后会弹出两个对话框，询问是否向开发板上的 CPLD 芯片和 Flash 芯片添加配置文件，这里不需要添加，选择 Bypass 即可



- 在 Spartan3E FPGA 芯片上点击右键，选择 Program



- 配置完成后，画面上会出现 Program Succeeded 字样。

- 旋转开发板上的旋转开关，观察 LED 灯的发光情况，验证设计的正确性。

五、实验结果分析

(一)代码及注释

rotary_led.v:

```
    `timescale 1ns / 1ps// 定义仿真时间单位 1ns/精度 1ps
    ////////////////////////////////////////////////////////////////////
    // 旋转编码器控制 LED 模块
    // 功能：通过旋转编码器控制 8 个 LED 的流水灯效果，支持左右旋转方向检测
    // 特性：包含按键消抖、旋转方向检测、LED 移位控制逻辑
    ////////////////////////////////////////////////////////////////////
    module rotary_led(
       clk,
       reset,
       R_A,
       R_B,
       LED_DATA
        );

       input clk; // 系统时钟输入
       input reset; // 异步复位信号（高有效）
       input R_A; // 旋转编码器 A 相信号
       input R_B; // 旋转编码器 B 相信号

       output [7:0] LED_DATA; // LED 输出控制（1 亮 0 灭）

    // 内部寄存器定义
       reg [7:0] shift_d;   //LED 移位寄存器（控制 8 个 LED 状态）
       reg rotary_q1;           // A 相消抖后信号
       reg rotary_q2;           // B 相消抖后信号
       reg rotary_q1_d;     // rotary_q1 的延迟版本（用于边沿检测）

       reg rotary_event;    // 旋转事件触发标志（高有效）
       reg rotary_left;     // 旋转方向标志（1=左旋，0=右旋）

       reg [3:0] clk_cnt;   // 时钟分频计数器（16 分频）
       reg R_A_d;           // A 相输入同步寄存器
       reg R_B_d;           // B 相输入同步寄存器

    // 连线定义
       wire [7:0] LED_DATA;
```

```
    wire rotary_press_in = 1'b0;

    //frequency divider 时钟分频模块
  //降低采样频率，配合机械开关的消抖需求
    always @ (posedge clk or posedge reset) begin
        if(reset) begin
            clk_cnt <= 4'b0;
        end
        else begin
            clk_cnt <= clk_cnt + 1; // 4 位计数器，每 16 个时钟
周期循环
        end
    end

  //输入信号同步与消抖预处理
  // 同步旋转编码器输入信号（在分频后的时钟域）
    //register the signals of switch A and switch B
    always @ (posedge clk_cnt[3] or posedge reset) begin
        if(reset) begin
            R_A_d <= 1'b0;
            R_B_d <= 1'b0;
        end
        else begin
            R_A_d <= R_A; // 同步 A 相信号
            R_B_d <= R_B; // 同步 B 相信号
        end
    end

    //viberation reduce for switch A,A 相信号消抖处理
  // 当 AB 相同时为高时置位，同时为低时复位.
    always @ (posedge clk or posedge reset) begin
        if(reset) begin
            rotary_q1 <= 1'b0;
        end
        else if(R_A_d && R_B_d) begin
            rotary_q1 <= 1'b1; // 相位同步高电平
        end
        else if( !(R_A_d || R_B_d) ) begin
            rotary_q1 <= 1'b0; // 相位同步低电平
        end
    end

    //viberation reduce for switch B, B 相信号消抖处理
  // 根据 AB 相不同状态切换
```

```
        always @ (posedge clk or posedge reset) begin
            if(reset) begin
                rotary_q2 <= 1'b0;//复位
            end
            else if(!R_A_d && R_B_d) begin
                rotary_q2 <= 1'b1; // A低B高时置位
            end
            else if(R_A_d && !R_B_d) begin
                rotary_q2 <= 1'b0;// A高B低时复位
            end
        end

        //旋转事件边沿检测模块
        //register the rotary_q1 signal
    //检测rotary_q1的上升沿（表示有效旋转动作）
        always @ (posedge clk or posedge reset) begin
            if(reset) begin
                rotary_q1_d <= 1'b0;
            end
            else begin
                rotary_q1_d <= rotary_q1; // 延迟一个时钟周期
            end
        end

    // 生成旋转事件脉冲（rotary_q1上升沿触发）
        //decide whether the rotary event occured
        //rising edge of rotary_q1 means rotary event occured
        always @ (posedge clk or posedge reset) begin
            if(reset) begin
                rotary_event <= 1'b0;
            end
            else begin
                rotary_event <= !rotary_q1_d && rotary_q1; // 上升
沿检测
            end
        end


        //get the rotary direction 旋转方向判断模块
// 在旋转事件发生时，根据rotary_q2状态判断方向
        always @ (posedge clk or posedge reset) begin
            if(reset) begin
                rotary_left <= 1'b0;
            end
```

```
            // 仅在旋转事件发生时更新
            else if( !rotary_q1_d && rotary_q1) begin
                rotary_left <= rotary_q2; // rotary_q2 状态决定方向
            end
        end

        //control the leds 根据旋转方向控制 LED 流水灯效果
        always @ (posedge clk or posedge reset) begin
            if(reset) begin
                shift_d <= 8'h01; // 复位时点亮第一个 LED
            end
            else if(rotary_event) begin
// 左旋转：循环左移（高位向低位移动）
// 右旋转：循环右移（低位向高位移动）

                shift_d <= rotary_left ? {shift_d[6:0],
shift_d[7]} : {shift_d[0], shift_d[7:1]};
            end
        end

    //将移位寄存器值输出，与保留信号异或（当前保留信号固定为 0）
       assign LED_DATA = {8{rotary_press_in}} ^ shift_d; // 异或
操作保留原值

endmodule
```

rotary_led.ucf:

```
    # PlanAhead Generated physical constraints

    NET "LED_DATA[0]" LOC = F12;
    NET "LED_DATA[1]" LOC = E12;
    NET "LED_DATA[2]" LOC = E11;
    NET "LED_DATA[3]" LOC = F11;
    NET "LED_DATA[4]" LOC = C11;
    NET "LED_DATA[5]" LOC = D11;
    NET "LED_DATA[6]" LOC = E9;
    NET "LED_DATA[7]" LOC = F9;
    NET "R_A" LOC = K18;
    NET "R_B" LOC = G18;
    NET "clk" LOC = C9;
    NET "reset" LOC = V16;
```

```
# PlanAhead Generated IO constraints

NET "LED_DATA[0]" IOSTANDARD = LVTTL;
NET "LED_DATA[1]" IOSTANDARD = LVTTL;
NET "LED_DATA[2]" IOSTANDARD = LVTTL;
NET "LED_DATA[3]" IOSTANDARD = LVTTL;
NET "LED_DATA[4]" IOSTANDARD = LVTTL;
NET "LED_DATA[5]" IOSTANDARD = LVTTL;
NET "LED_DATA[6]" IOSTANDARD = LVTTL;
NET "LED_DATA[7]" IOSTANDARD = LVTTL;
NET "R_A" IOSTANDARD = LVTTL;
NET "R_B" IOSTANDARD = LVTTL;
NET "clk" IOSTANDARD = LVCMOS33;
NET "reset" IOSTANDARD = LVTTL;
NET "LED_DATA[0]" DRIVE = 8;
NET "LED_DATA[1]" DRIVE = 8;
NET "LED_DATA[2]" DRIVE = 8;
NET "LED_DATA[3]" DRIVE = 8;
NET "LED_DATA[4]" DRIVE = 8;
NET "LED_DATA[5]" DRIVE = 8;
NET "LED_DATA[6]" DRIVE = 8;
NET "LED_DATA[7]" DRIVE = 8;
NET "R_A" PULLUP;
NET "R_B" PULLUP;
NET "reset" PULLDOWN;
```

(二)Spartan3E 现象分析

在旋转旋钮时，IED 灯按序发光:

初始状态为 0x01（二进制 00000001）

左旋时执行循环左移

右旋时执行循环右移

符合实验要求

六、实验总结

通过本次试验，我熟悉使用 ISE 集成开发环境进行逻辑设计的基本流程，掌握了 Verilog 硬件描述语言的基本语法，而且学会了利用开发板上的各种资源在 Spartan3E 板上验证实验结果时，我发现灯光闪烁时出现轻微不规则的闪

动：旋转旋钮而灯光按照要求顺序闪动时，会出现不按照规则的闪动。排除了程序的问题后，我判断为实验板子的接触不良。