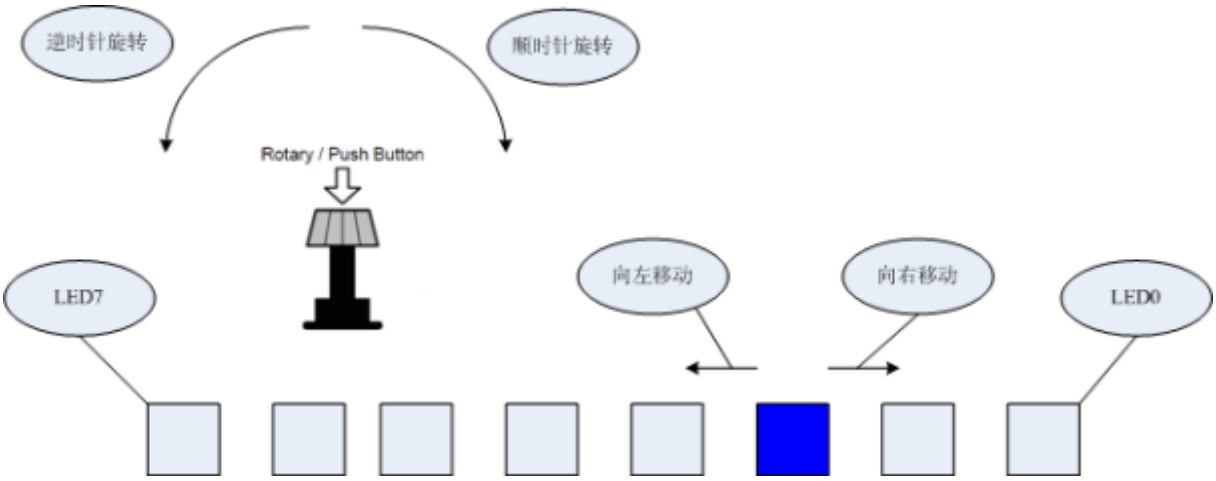
20049200057-薛宇翔-实验一

实验环境

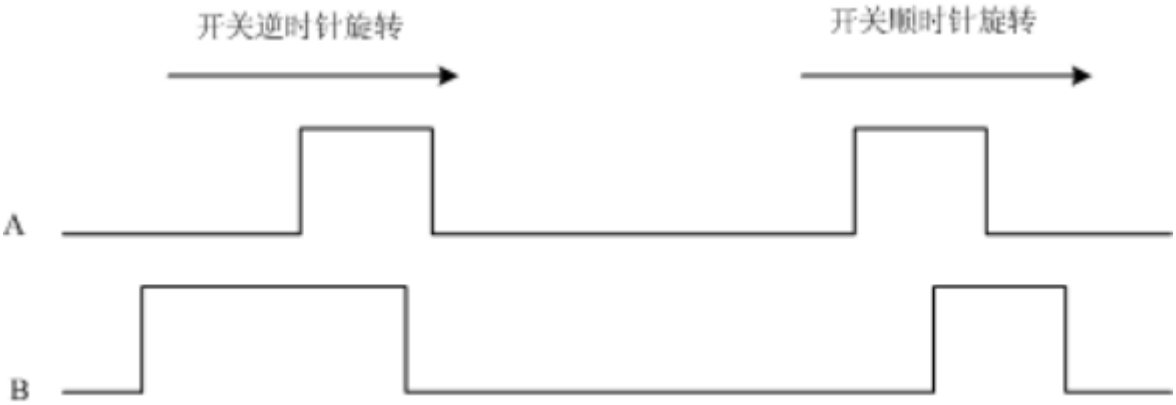
Linux操作系统
ISE Design Suite 14.7
Spartan 3E开发板

实验介绍

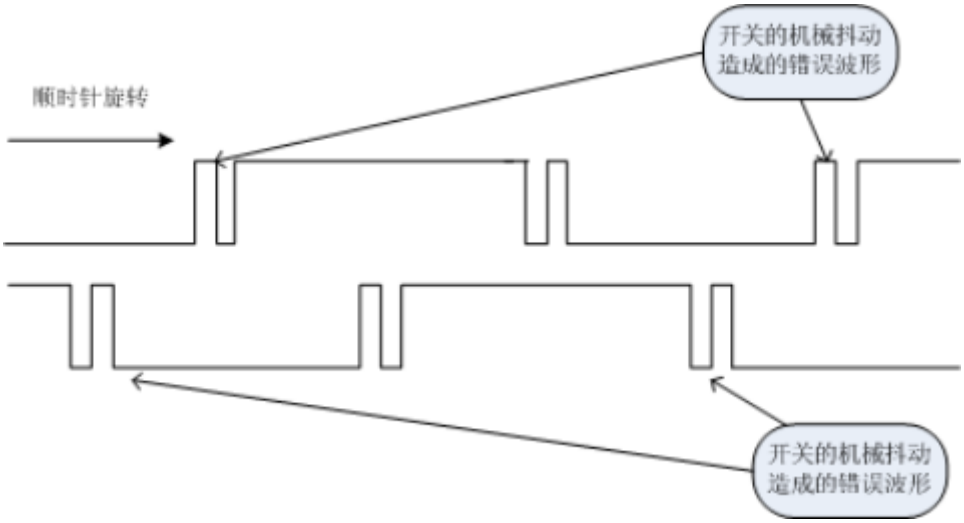
Spartan 3E开发板上有8个并排放置的发光二极管LED7~LED0，以及一个旋转开关，实验要求使用旋转开关控制二极管轮流发光，旋转开关顺时针或逆时针转动控制发光二极管右移或左移



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。



但是在旋转开关输出的过程中可能存在机械抖动现象，在设计时需要加入消除抖动现象的措施。



实验目标

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

实验过程

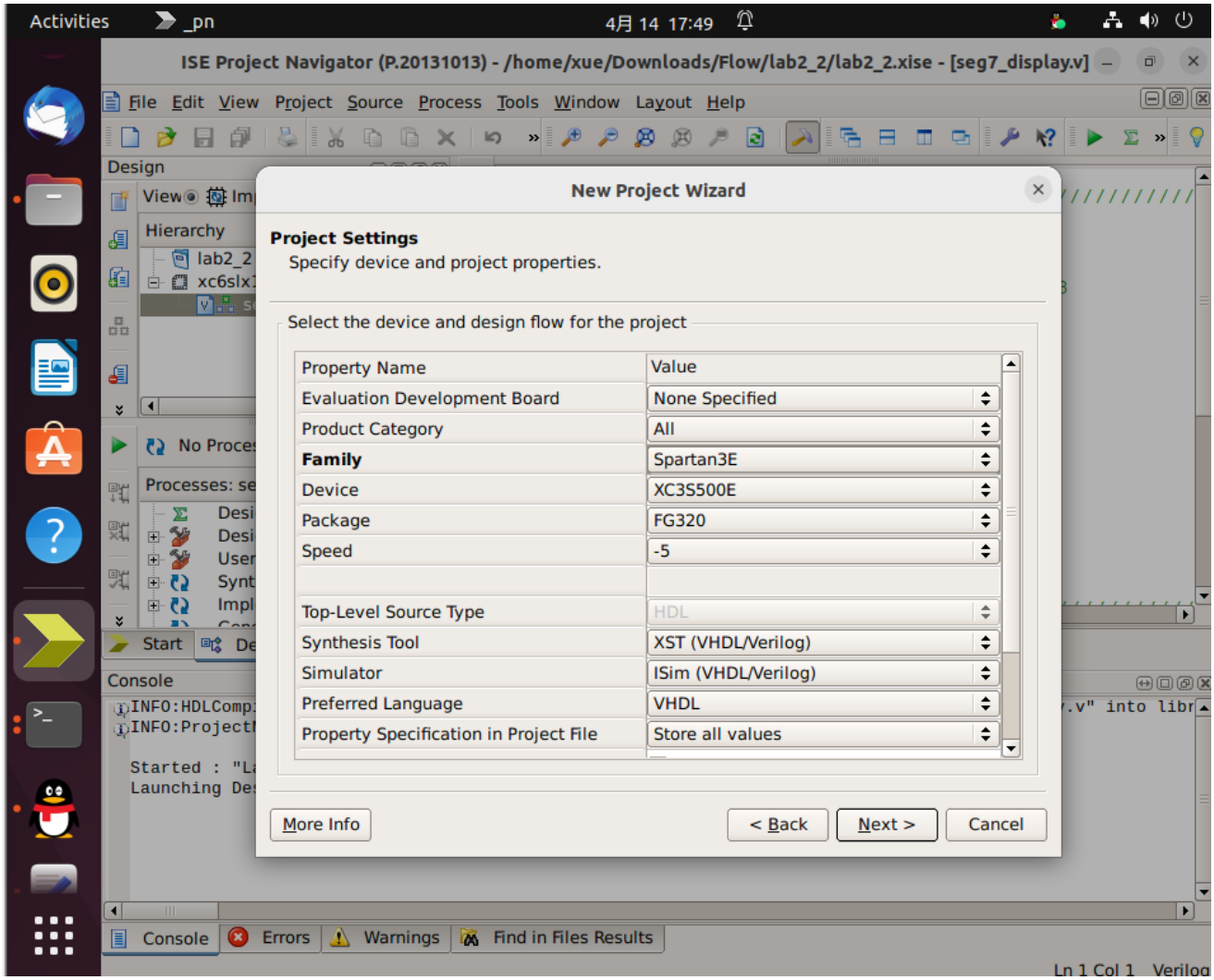
本实验包含四个主要的部分：创建工程，设计输入，综合实现，进行硬件配置。

实验步骤

1.创建工程

打开ISE，在菜单栏中选择File → New Project，打开创建工程界面，选择适当的文件夹作为工程路径，输入工程名。

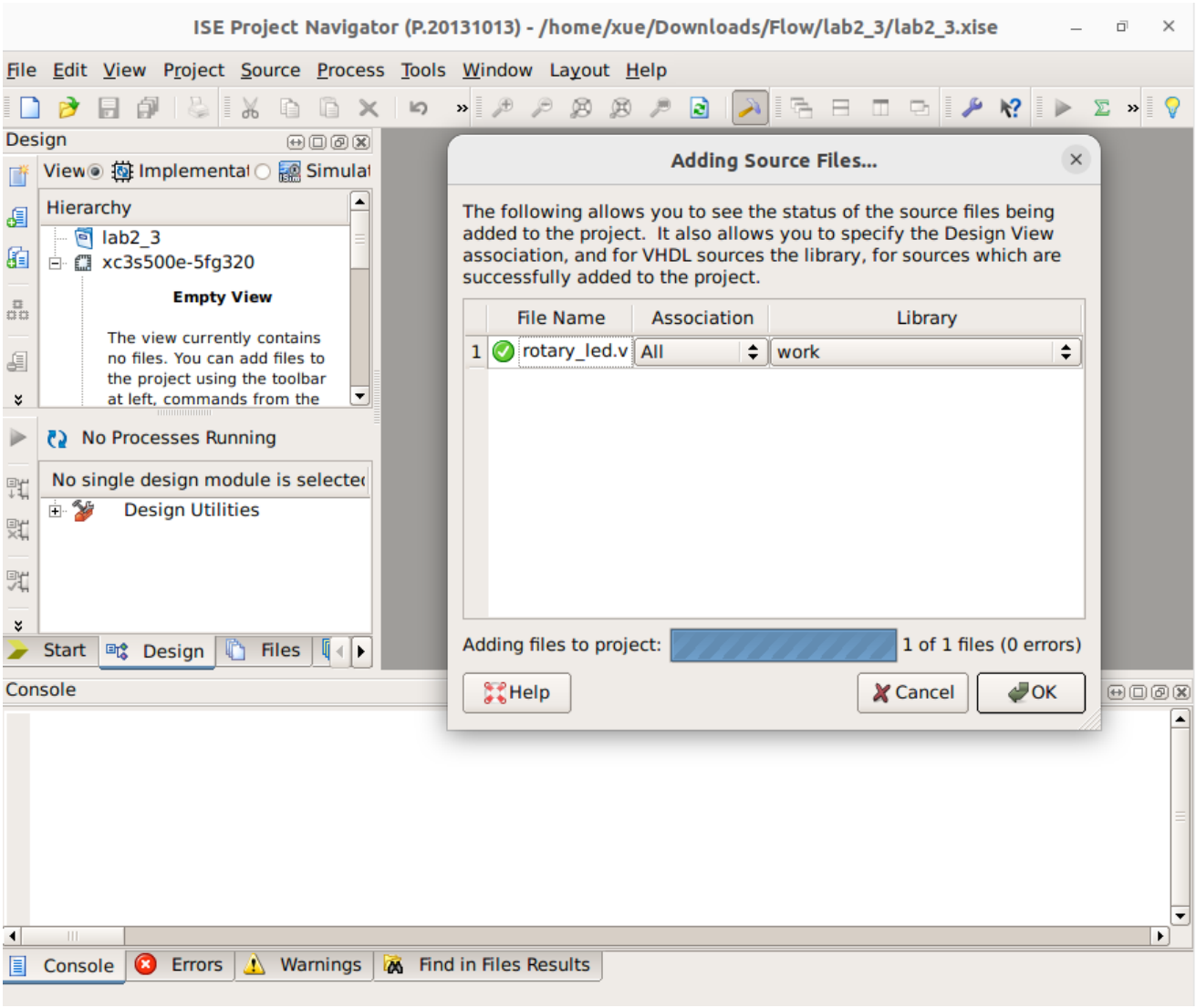
点击Next按钮，进入器件和设计工具选择，对开发板芯片参数设置如下所示：



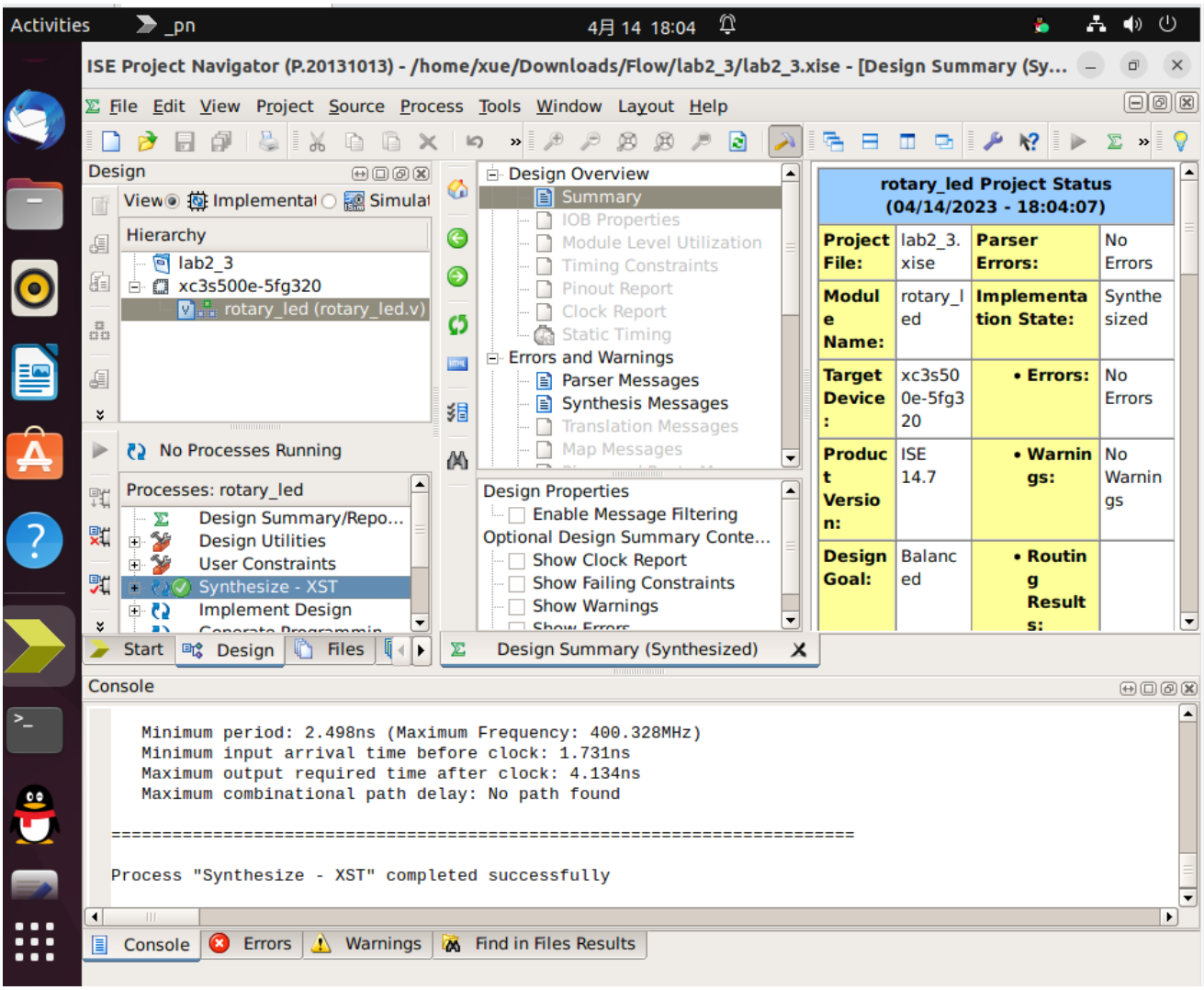
配置好参数后点击Next，然后点击Finish即可完成工程的创建。

2.设计输入和综合实现

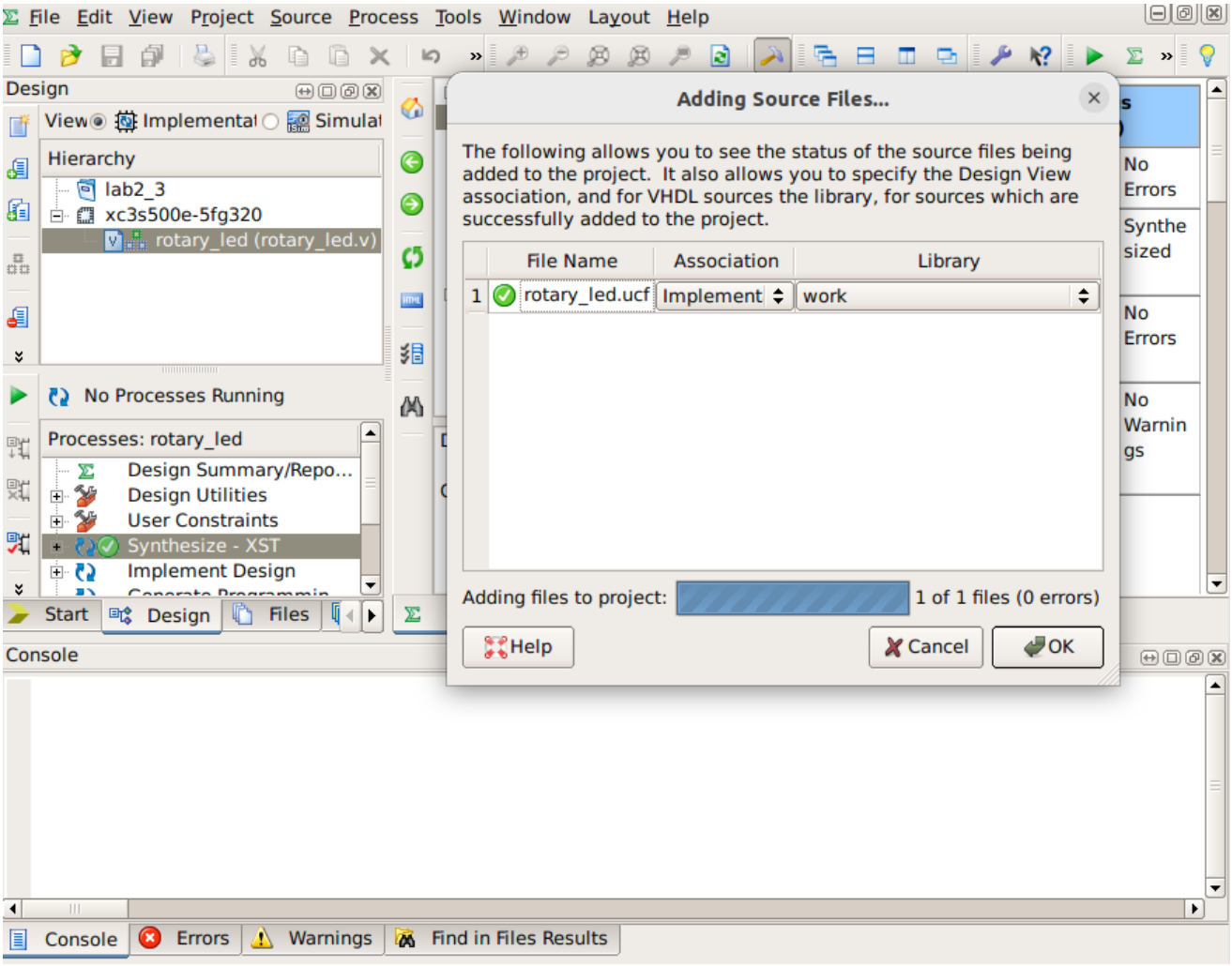
选择菜单栏中的Project→Add Source，将试验资料中的rotary_led.v文件添加到当前项目中，如下图所示：



添加后双击Processes窗口中的Synthesize-XST进行综合，综合结果如下图所示：

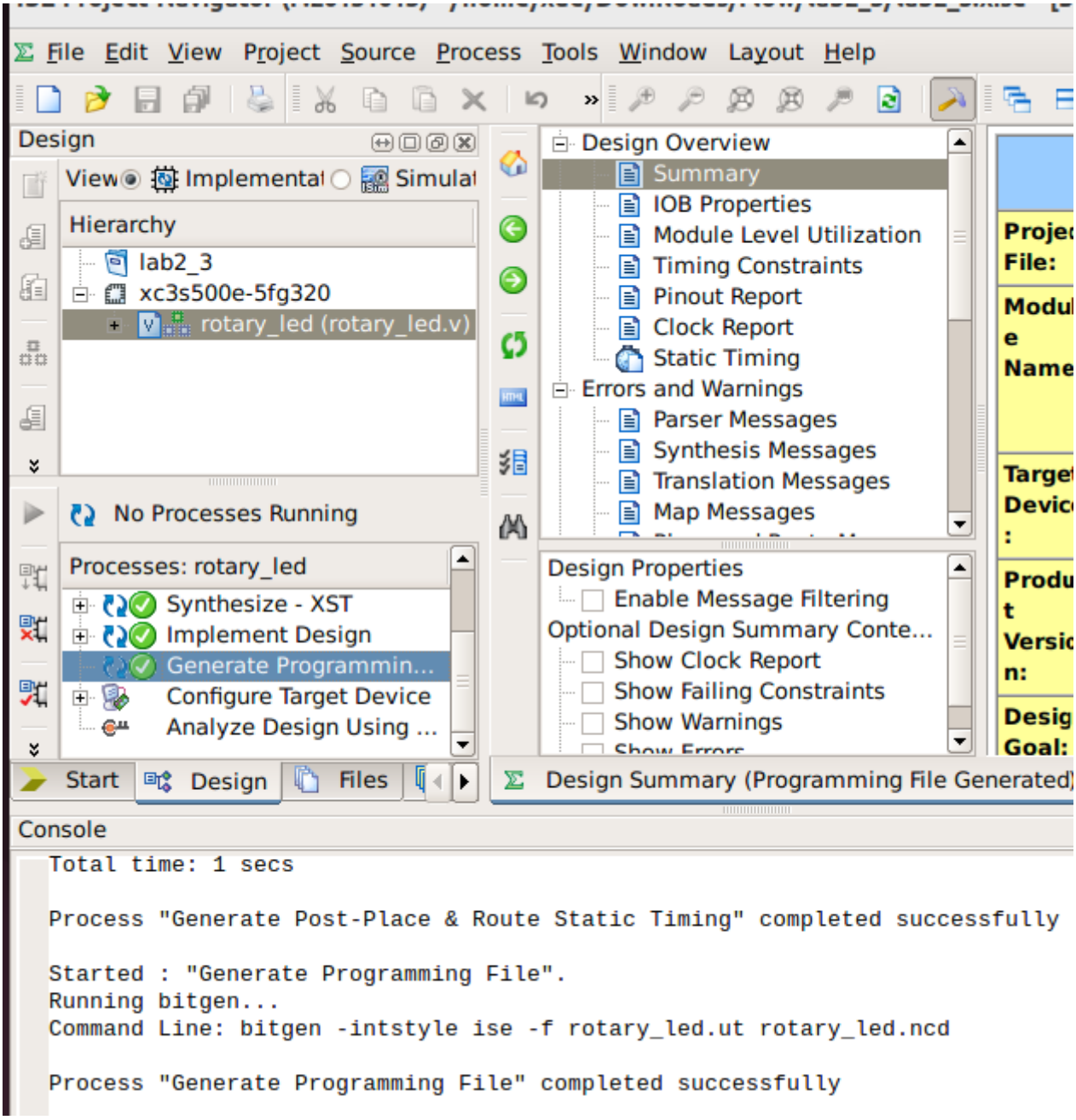


在实验资料中的文件rotary_led.ucf即为该实验的约束文件。点击菜单栏中Project→Add Source，将该约束文件添加到项目中，添加截图如下图所示



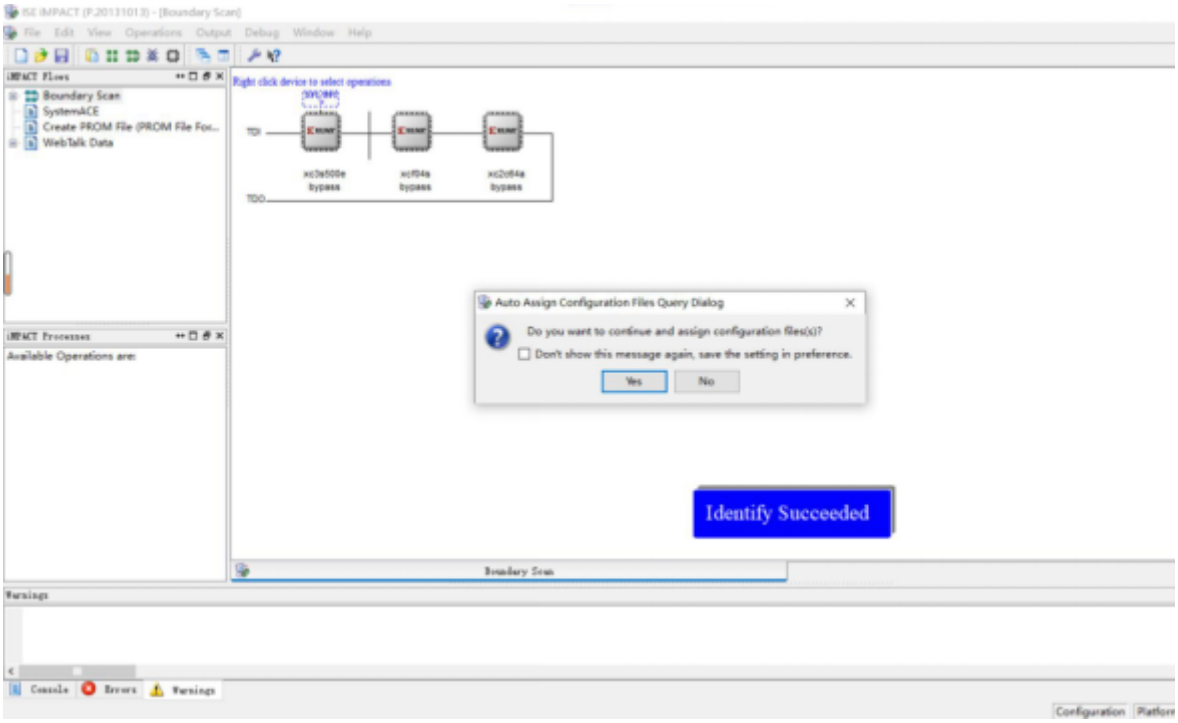
3. 器件配置

使用ISE自带的iMPACT可以完成Spartan 3E开发板的配置。首先双击Processes窗口中的Generate Programming File，生成该项目的二进制比特流文件，用于输入到开发板中，如下图所示：

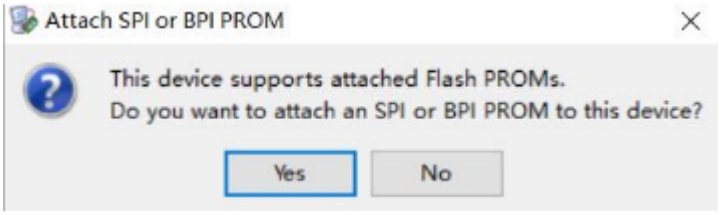


然后双击Configure Target Device，打开iMPACT GUI，在这里进行开发板的配置。点击右边的Boundary Scan，然后在空白区域右键选择Initialize Chain。

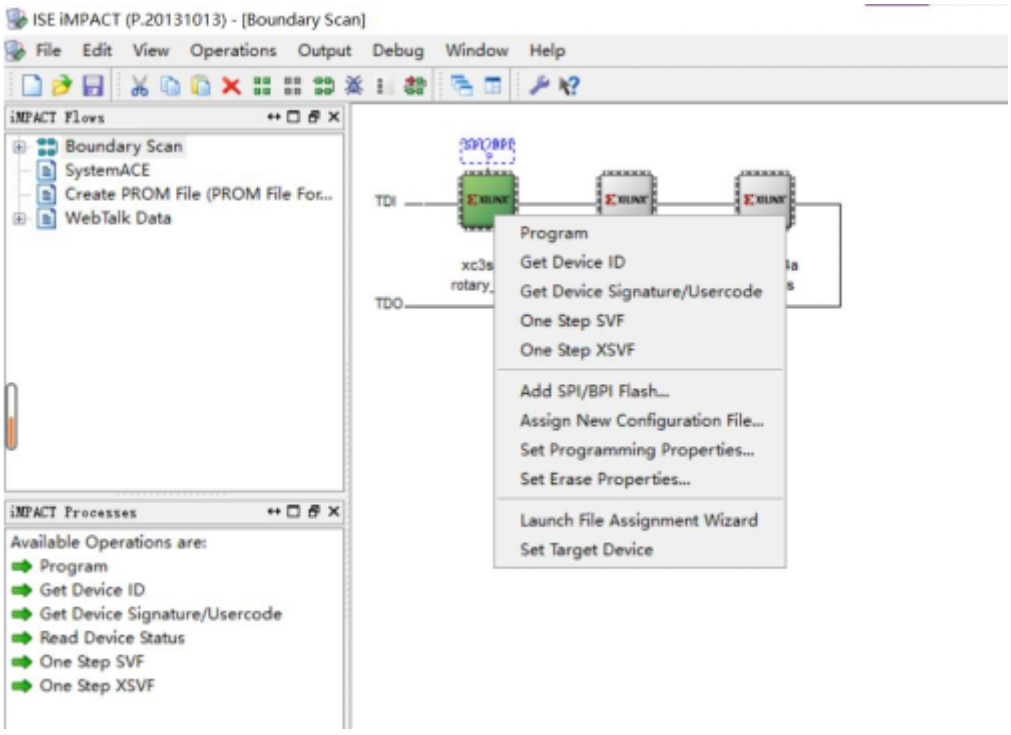
然后在弹出的对话框中选择Yes，并指定之前生成的二进制比特流文件，如下图所示。



弹出如下对话框，选择No



然后右键选择芯片，并选择Program，即可将项目信息导入到开发板中，如下图所示。



导入成功，开发板的配置结束。

实验结果：可以根据旋转按钮来实现LED灯的循环发光。

实验总结

通过本次试验，熟悉使用ISE集成开发环境进行逻辑设计的基本流程，掌握Verilog硬件描述语言的基本语法，并学会连接利用开发板。

实验代码

rotary_led.v

```
//定义model
module rotary_led(

        //定义输入输出端口
        clk,
        reset,
        R_A,
        R_B,
        LED_DATA
    );
        //输入信号，clk表示时钟，reset表示复位，R_A, R_B是两个控制事件的信号
        input clk;
        input reset;
        input R_A;
        input R_B;
        //定义8个LED灯的输出
        output [7:0] LED_DATA;
        //控制8个LED灯的状态
        reg [7:0] shift_d;

        //定义抖动减弱后的信号状态

        //rotary_q1定义的是当前信号的情况，rotary_q2定义的是当前旋钮的旋转方向
        reg rotary_q1;
        reg rotary_q2;

        //一个临时变量，用于记录rotary_q1的数值
        reg rotary_q1_d;


        //定义旋转事件是否发生
        reg rotary_event;

        //定义旋转方向
        reg rotary_left;


        //时钟计数
        reg [3:0] clk_cnt;
```

```
        //定义switch A和switch B的信号状态
        reg R_A_d;
        reg R_B_d;


        //定义物理布线的输出
        wire [7:0] LED_DATA;
        wire rotary_press_in = 1'b0;

        //always@(敏感事件列表) 用于描述时序逻辑用于描述时序逻辑，posedge表示上升沿，在时钟上升沿和复位信号上升沿的时候触发
        always @ (posedge clk or posedge reset) begin
                if(reset) begin
                        clk_cnt <= 4'b0;
                end

                //对时钟信号计数
                else begin
                        clk_cnt <= clk_cnt + 1;
                end
        end

        //检测clk_cnt[3]时的时钟信号，相当于一个延迟检测，将此时R_A和R_B端口的信号记录下来

        //在一个时钟延迟后记录，用于消除按钮的抖动
        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;
                        R_B_d <= R_B;
                end
        end

        //在抖动结束后记录状态，R_A_d和R_B_d端口同时有效时才会使rotary_q1有效
        always @ (posedge clk or posedge reset) begin
                if(reset) begin
                        rotary_q1 <= 1'b0;
                end

                //两个都是高电平，代表信号的开始，从这里开始记录
                else if(R_A_d && R_B_d) begin
                        rotary_q1 <= 1'b1;
                end
```

```
        //两个有一个是低电平，表示这次信号已经结束了
        else if( !(R_A_d || R_B_d) ) begin
                rotary_q1 <= 1'b0;
        end
end


//rotary_q2用于记录旋钮的旋转方向
always @ (posedge clk or posedge reset) begin
        if(reset) begin
                rotary_q2 <= 1'b0;
        end

        //B信号为高电平，A信号为上升沿，表示逆时针旋转
        else if(!R_A_d && R_B_d) begin
                rotary_q2 <= 1'b1;
        end

        //B信号为低电平，A信号为上升沿，表示顺时针旋转
        else if(R_A_d && !R_B_d) begin
                rotary_q2 <= 1'b0;
        end
end

//记录rotary_q1的信号状态
always @ (posedge clk or posedge reset) begin
        if(reset) begin
                rotary_q1_d <= 1'b0;
        end
        else begin
                rotary_q1_d <= rotary_q1;
        end
end

//一次信号的开始与结束，表示一次事件的发生
always @ (posedge clk or posedge reset) begin
        if(reset) begin
                rotary_event <= 1'b0;
        end
        else begin
                rotary_event <= !rotary_q1_d && rotary_q1;
        end
end

//获得当前旋转方向
always @ (posedge clk or posedge reset) begin
        if(reset) begin
                rotary_left <= 1'b0;
        end
```

```
                else if( !rotary_q1_d && rotary_q1) begin
                        rotary_left <= rotary_q2;
                end
        end

        //使能LED灯
        always @ (posedge clk or posedge reset) begin
                if(reset) begin
                        shift_d <= 8'h01;
                end

                //用shift_d记录当前按钮的状态
                else if(rotary_event) begin
                        shift_d <= rotary_left ? {shift_d[6:0], shift_d[7]} :
{shift_d[0], shift_d[7:1]};
                end
        end
        //通过与操作来确定LED灯的状态
        assign LED_DATA = {8{rotary_press_in}} ^ shift_d;

endmodule
```