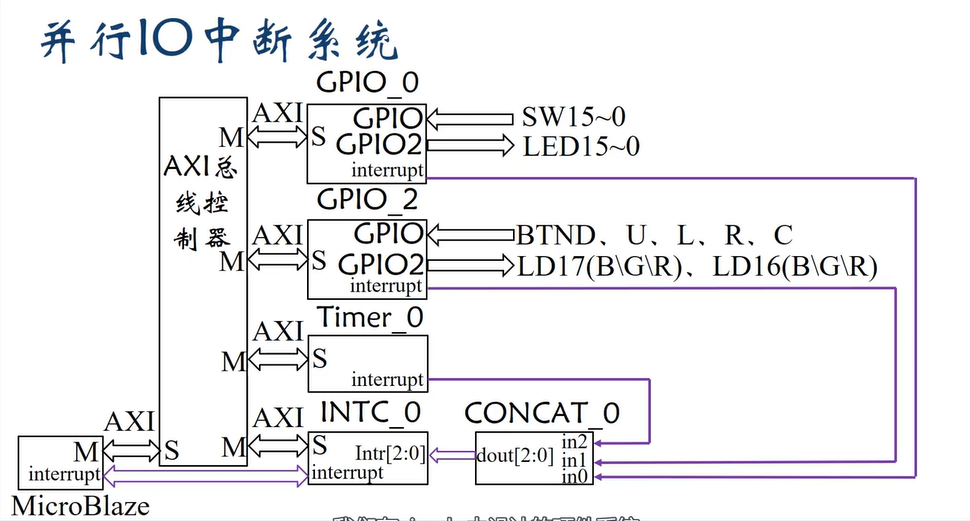
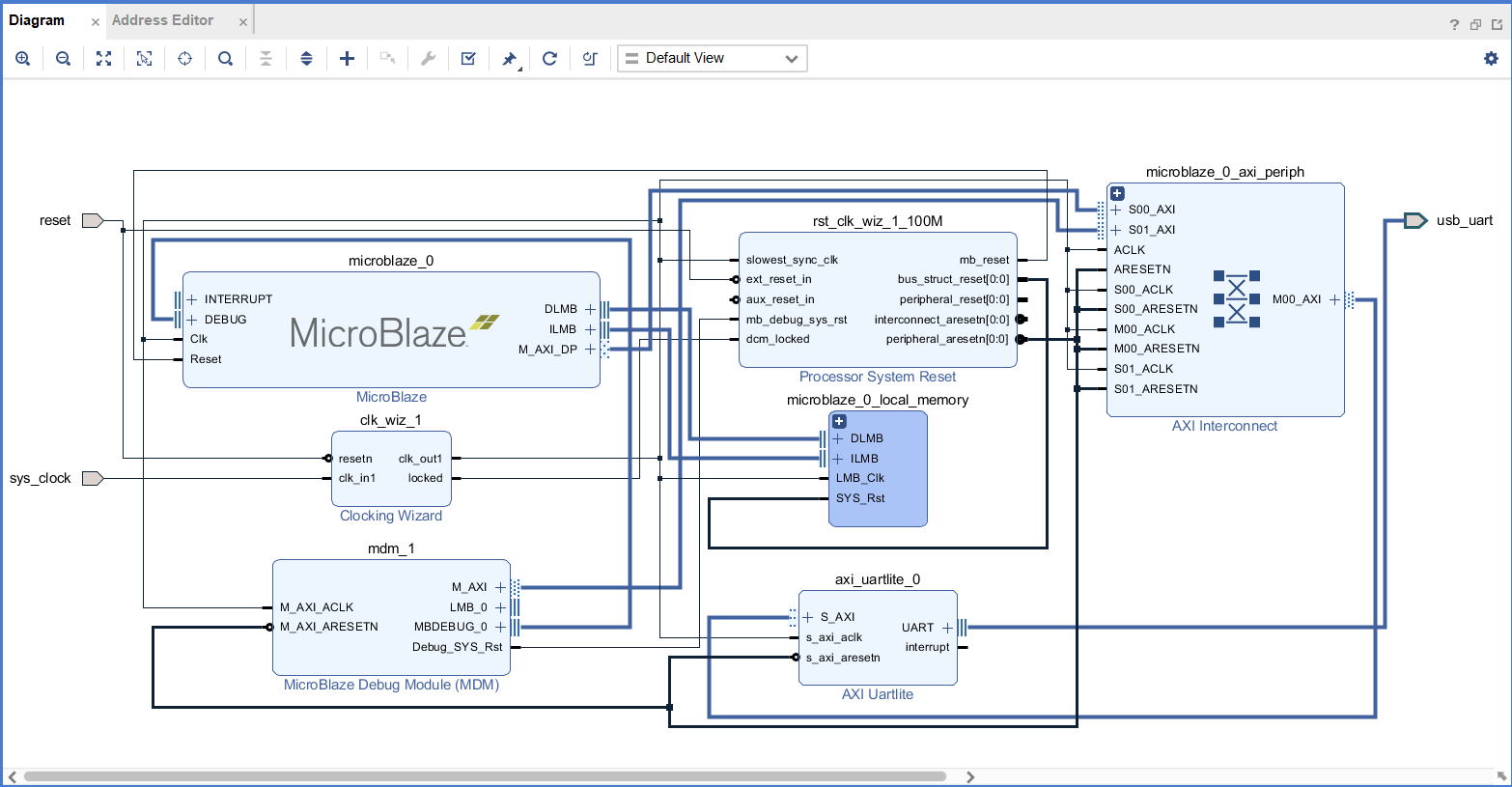
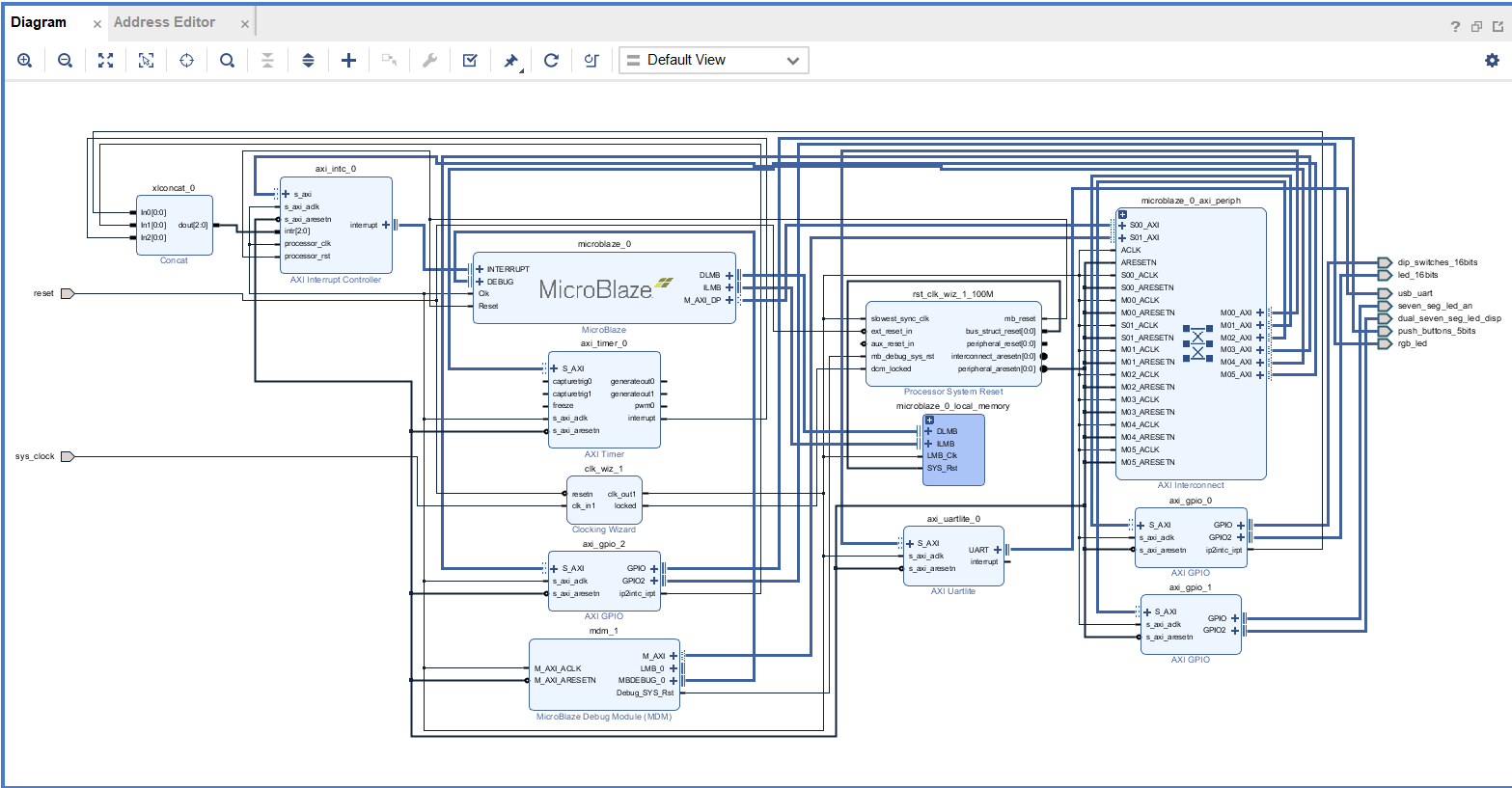
**并行IO接口程序设计**

**通信2002班 涂增基 U202013990**

1. **实验任务**

**因为程序控制方式和中断方式选取的实验任务不同，因此实验任务在报告“软件设计”部分展示。**

**二、硬件电路框图**

  
**程序控制方式：**  
**快速中断模式：**  


**三、软件设计**

**（一）程序控制方式**

**1、实验任务**

嵌入式计算机系统将独立按键以及独立开关作为输入设备，LED 灯作为输出设备。修改实验示例程序代码，实现以下功能：

1） 按下C键时，计算机读入一组16 位独立开关状态作为第一个输入的二进制数据，并即时显示输入的二进制数到16 位LED 灯上。（没有按下BTNC按键时，开关拨动不读入数据）

2） 按下R键时，计算机读入另一组16 位独立开关状态作为第二个输入的二进制数据，并即时显示输入的二进制数到16 位LED 灯上。（没有按下BTNR按键时，开关拨动不读入数据）

3） 按下U键时，将保存的2 组二进制数据做无符号加法运算，并将运算结果输出到LED 灯对应位。

4） 按下D键时，将保存的2 组二进制数据做无符号乘法运算，并将运算结果输出到LED 灯对应位。

**2.软件流程图**

****

**图1 程序控制方式流程图**

程序控制方式采用循环读取按键键值，根据按键的值读取开关状态，并做相应处理。

**3.软件源代码加注释**

**int** **main**()

{

**char** button = 0x0;

**int** a,b;

//GPIO输入输出配置

Xil\_Out16(XPAR\_GPIO\_0\_BASEADDR + XGPIO\_TRI\_OFFSET, 0xffff); //配置GPIO\_0通道1的16位开关为输入

Xil\_Out16(XPAR\_GPIO\_0\_BASEADDR+XGPIO\_TRI2\_OFFSET,0X0); //配置GPIO\_0通道2的16位LED灯为输出

**while**(1)

{

**while**((Xil\_In8(XPAR\_GPIO\_2\_BASEADDR + XGPIO\_DATA\_OFFSET)&0x1f)!=0) //循环检测按键是否按下

{

button = Xil\_In8(XPAR\_GPIO\_2\_BASEADDR + XGPIO\_DATA\_OFFSET)&0x1f; //读入按键值

**while**((Xil\_In8(XPAR\_GPIO\_2\_BASEADDR + XGPIO\_DATA\_OFFSET)&0x1f)!=0); //等到无按键输入

**switch**(button)

{

**case** 0x1: //BTNC 读入开关

a=Xil\_In16(XPAR\_AXI\_GPIO\_0\_BASEADDR+XGPIO\_DATA\_OFFSET);

Xil\_Out16(XPAR\_AXI\_GPIO\_0\_BASEADDR+XGPIO\_DATA2\_OFFSET,a);

**break**;

**case** 0x8: //BTNR 读入另一组开关

b=Xil\_In16(XPAR\_AXI\_GPIO\_0\_BASEADDR+XGPIO\_DATA\_OFFSET);

Xil\_Out32(XPAR\_AXI\_GPIO\_0\_BASEADDR+XGPIO\_DATA2\_OFFSET,b);

**break**;

**case** 0x2: //BTNU 加法

Xil\_Out16(XPAR\_AXI\_GPIO\_0\_BASEADDR+XGPIO\_DATA2\_OFFSET,a+b);

**break**;

**case** 0x10: //BTND 乘法

Xil\_Out16(XPAR\_AXI\_GPIO\_0\_BASEADDR+XGPIO\_DATA2\_OFFSET,a\*b);

**break**;

**default**:

**break**;

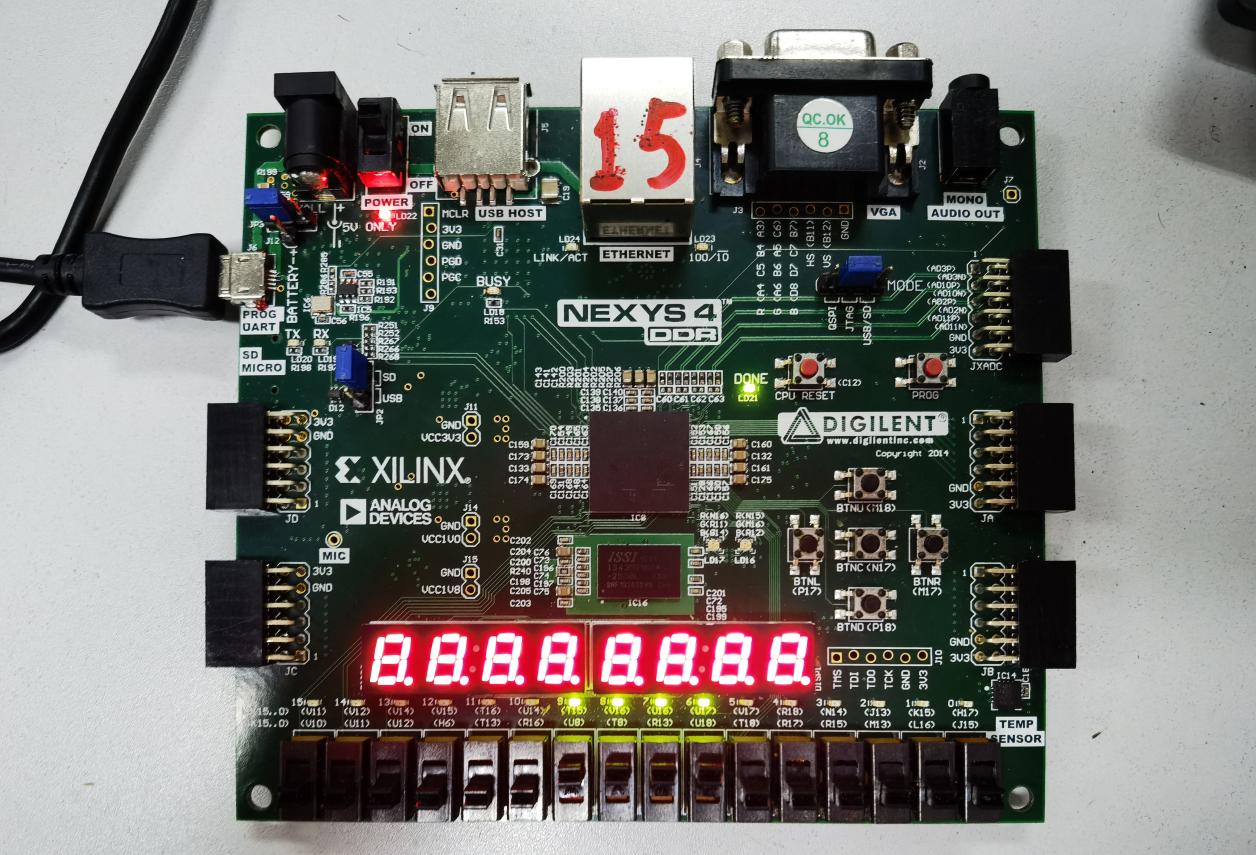
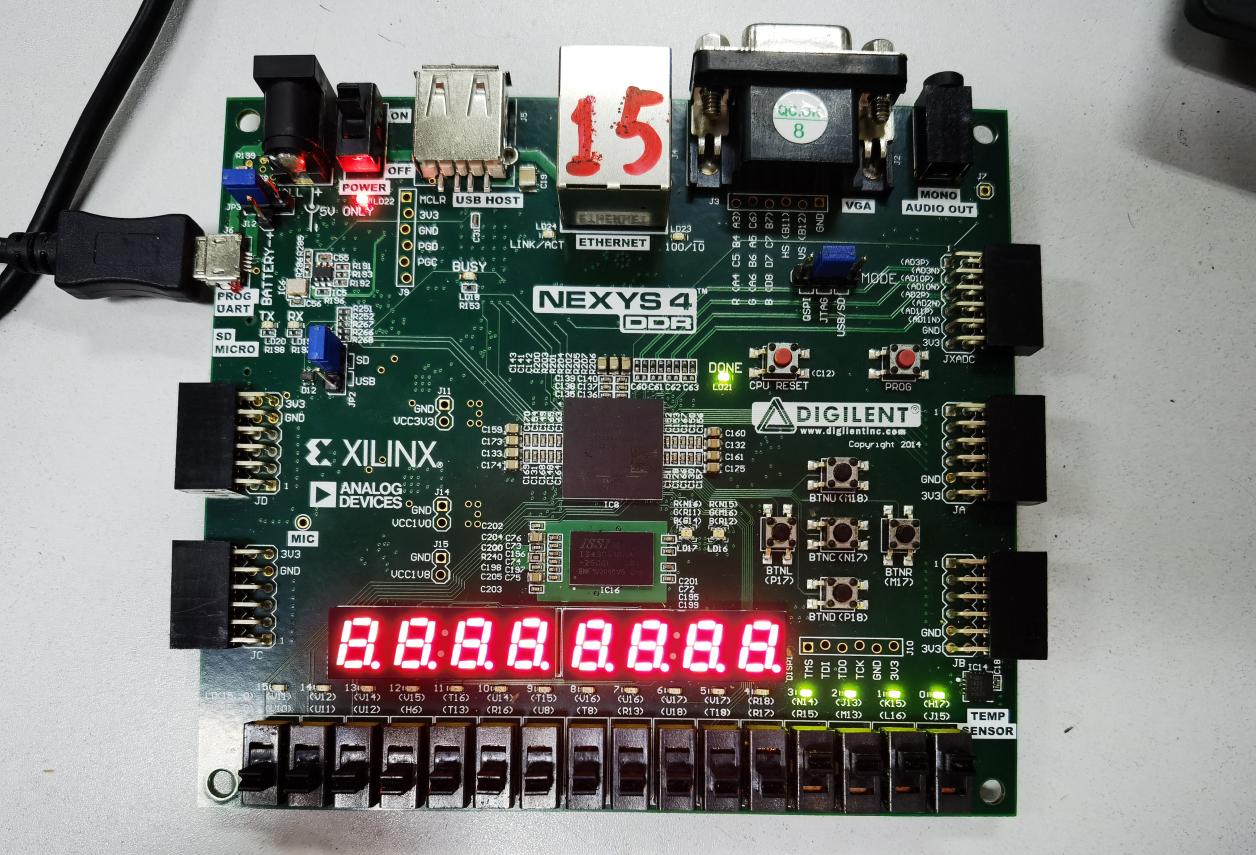
}

}

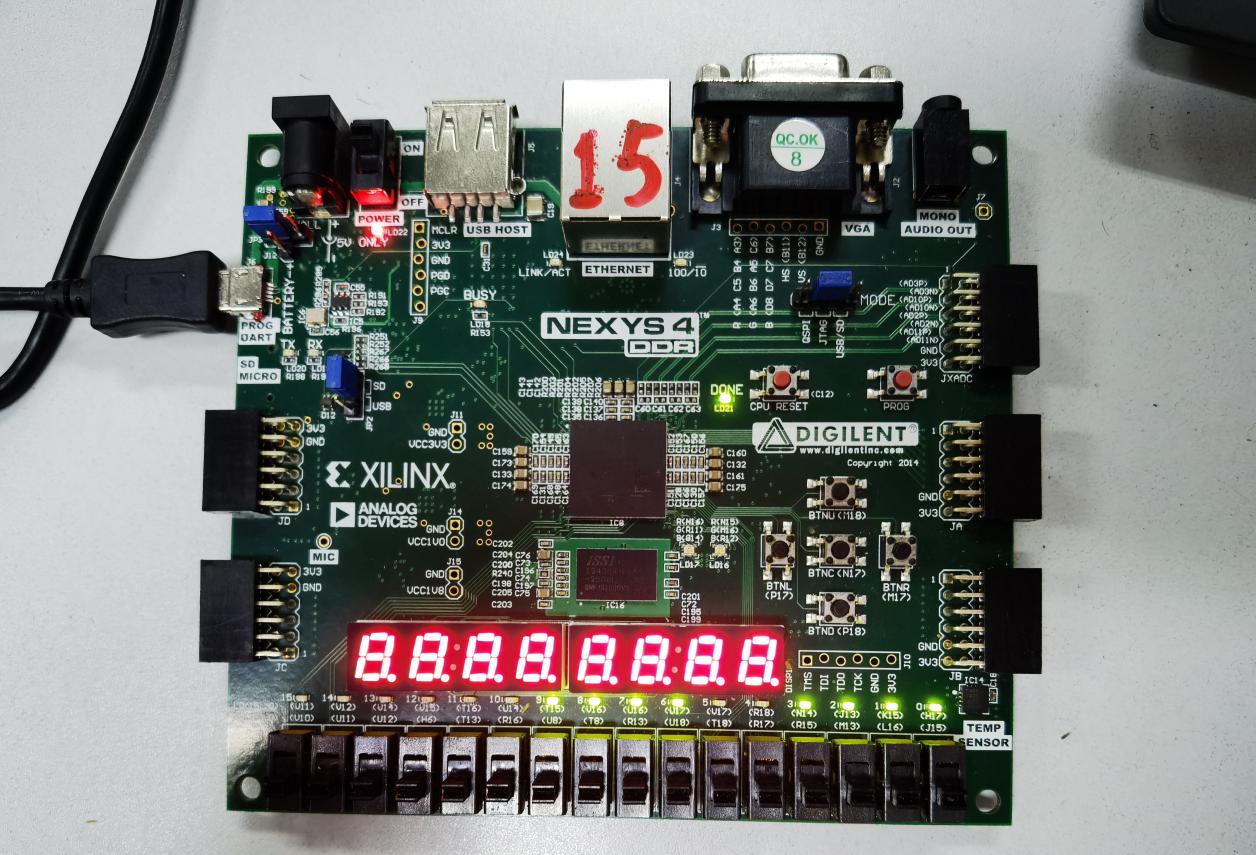
}

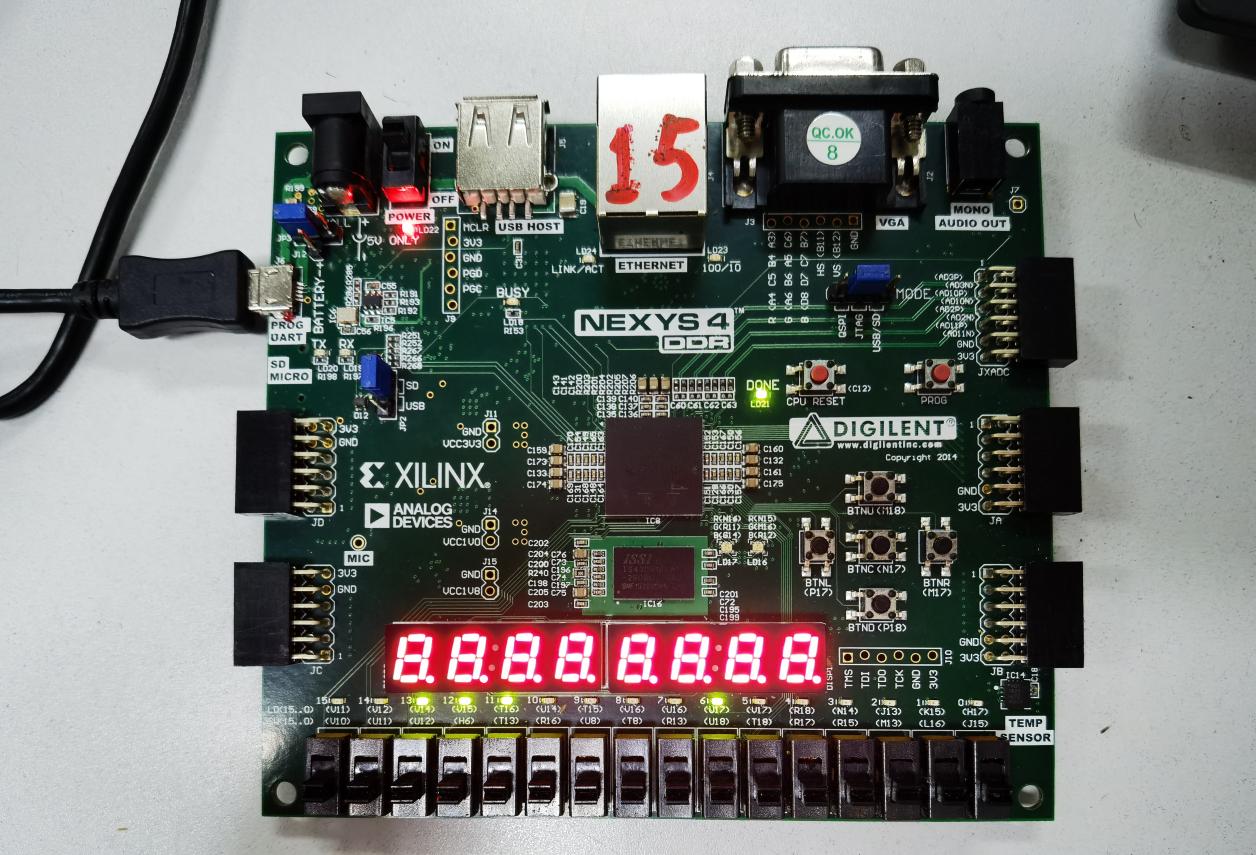
}

**4、测试结果**

**  
  
**

**加：**

****

**乘：  
**

**（二）快速中断方式**

**1.实验任务**采用中断方式，由独立按键控制LED灯实现不同显示：  
按下L键，LED灯向左走马灯显示，移动频率1Hz；  
按下R键，LED灯向右走马灯显示，移动频率1Hz；  
按下C键，LED灯固定当前显示位置；  
按下U键，当前显示位置的LED灯快闪，闪烁频率为3Hz；  
按下D键，当前显示位置的LED灯慢闪，闪烁频率0.5Hz

**2、软件流程图**

****

**3.软件源代码加注释  
#include** "xil\_io.h"

**#include** "stdio.h"

**#include** "xgpio.h"

**#include** "sleep.h"

**#include** "xtmrctr.h"

**#include** "xintc\_l.h"

**#define** RESET\_VALUE0 20000000-2 //1s

**#define** RESET\_VALUE1 40000000-2 //2s

**#define** RESET\_VALUE2 6666666-2 //1/3s

**void** **button\_handle**()**\_\_attribute\_\_**((fast\_interrupt));

**void** **timer\_handle**()**\_\_attribute\_\_**((fast\_interrupt));

**void** **button\_handle**();

**void** **timer\_handle**();

**void** **timer0\_handle**();

**int** mode=0;

**int** counter1=0;

**int** counter2=0;

**int** **main**()

{

Xil\_Out16(XPAR\_AXI\_GPIO\_0\_BASEADDR+XGPIO\_TRI2\_OFFSET,0X0000);//LED方向控制

Xil\_Out8(XPAR\_AXI\_GPIO\_2\_BASEADDR+XGPIO\_TRI\_OFFSET,0X1f);//按键方向控制

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET,

Xil\_In32((XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET)&~XTC\_CSR\_ENABLE\_TMR\_MASK));//写TCSR，停止定时器

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TLR\_OFFSET,RESET\_VALUE0);//写TLR预置定时器初值

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET,

Xil\_In32((XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET)|XTC\_CSR\_LOAD\_MASK));//装载计数器初值

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET,

Xil\_In32((XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET)&~XTC\_CSR\_LOAD\_MASK)\

|XTC\_CSR\_ENABLE\_TMR\_MASK|XTC\_CSR\_AUTO\_RELOAD\_MASK|XTC\_CSR\_ENABLE\_INT\_MASK|XTC\_CSR\_DOWN\_COUNT\_MASK);

//开始计时运行，自动获取，允许中断，减计数

Xil\_Out32(XPAR\_AXI\_GPIO\_2\_BASEADDR+XGPIO\_ISR\_OFFSET,0X1);//按键GPIO中断清除

Xil\_Out32(XPAR\_AXI\_GPIO\_2\_BASEADDR+XGPIO\_IER\_OFFSET,0X1);//使能按键GPIO中断

Xil\_Out32(XPAR\_AXI\_GPIO\_2\_BASEADDR+XGPIO\_GIE\_OFFSET,0X80000000);//写GPIOGIER

Xil\_Out32(XPAR\_AXI\_INTC\_0\_BASEADDR+XIN\_IAR\_OFFSET,XPAR\_AXI\_GPIO\_2\_IP2INTC\_IRPT\_MASK|XPAR\_AXI\_TIMER\_0\_INTERRUPT\_MASK);//中断控制器进行清除

Xil\_Out32(XPAR\_AXI\_INTC\_0\_BASEADDR+XIN\_IER\_OFFSET,XPAR\_AXI\_GPIO\_2\_IP2INTC\_IRPT\_MASK|XPAR\_AXI\_TIMER\_0\_INTERRUPT\_MASK);//中断控制器进行中断使能

Xil\_Out32(XPAR\_AXI\_INTC\_0\_BASEADDR+XIN\_MER\_OFFSET,XIN\_INT\_MASTER\_ENABLE\_MASK|XIN\_INT\_HARDWARE\_ENABLE\_MASK);//中断使能

Xil\_Out32((XPAR\_INTC\_0\_BASEADDR)+(XIN\_IMR\_OFFSET), XPAR\_AXI\_GPIO\_2\_IP2INTC\_IRPT\_MASK|XPAR\_AXI\_TIMER\_0\_INTERRUPT\_MASK);//写中断模式寄存器

Xil\_Out32(XPAR\_AXI\_INTC\_0\_BASEADDR+XIN\_IVAR\_OFFSET+4\*(XPAR\_AXI\_INTC\_0\_AXI\_GPIO\_2\_IP2INTC\_IRPT\_INTR),(**unsigned** **int**)button\_handle);

Xil\_Out32(XPAR\_AXI\_INTC\_0\_BASEADDR+XIN\_IVAR\_OFFSET+4\*(XPAR\_AXI\_INTC\_0\_AXI\_TIMER\_0\_INTERRUPT\_INTR),(**unsigned** **int**)timer\_handle);

microblaze\_enable\_interrupts();

**return** 0;

}

**void** **button\_handle**()

{

**short** current\_key\_in;

current\_key\_in=Xil\_In8(XPAR\_AXI\_GPIO\_2\_BASEADDR+XGPIO\_DATA\_OFFSET)|0xe0;

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET,

Xil\_In32((XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET)&~XTC\_CSR\_ENABLE\_TMR\_MASK));//写TCSR，停止定时器

**switch**(current\_key\_in){

**case** 0xe1:

mode=4;///// C

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TLR\_OFFSET,RESET\_VALUE0);**break**;//写TLR预置定时器初值

**case** 0xe2:

mode=2;//// U

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TLR\_OFFSET,RESET\_VALUE2);**break**;//写TLR预置定时器初值

**case** 0xe4:

mode=0;/// L

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TLR\_OFFSET,RESET\_VALUE0);**break**;//写TLR预置定时器初值

**case** 0xe8:

mode=1;///R

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TLR\_OFFSET,RESET\_VALUE0);**break**;//写TLR预置定时器初值

**case** 0xf0: //D

mode=3;

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TLR\_OFFSET,RESET\_VALUE1);**break**;//写TLR预置定时器初值

**default**:**break**;

}

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET,

Xil\_In32((XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET)|XTC\_CSR\_LOAD\_MASK));//装载计数器初值

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET,

Xil\_In32((XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET)&~XTC\_CSR\_LOAD\_MASK)\

|XTC\_CSR\_ENABLE\_TMR\_MASK|XTC\_CSR\_AUTO\_RELOAD\_MASK|XTC\_CSR\_ENABLE\_INT\_MASK|XTC\_CSR\_DOWN\_COUNT\_MASK);

}

**void** **timer\_handle**()

{

**int** status;

status = Xil\_In32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET);

**if**((status&XTC\_CSR\_INT\_OCCURED\_MASK) == XTC\_CSR\_INT\_OCCURED\_MASK)

timer0\_handle();

Xil\_Out32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET, Xil\_In32(XPAR\_AXI\_TIMER\_0\_BASEADDR+XTC\_TCSR\_OFFSET));

}

**void** **timer0\_handle**()

{

**int** data;

**switch**(mode)

{

**case** 0:

counter1++;

**if**(counter1 >= 16)

counter1 = 0;

data = 1<<counter1;

**break**;

**case** 1:

counter1--;

**if**(counter1 < 0)

counter1 = 15;

data = 1<<counter1;

**break**;

**case** 2:

counter2++;

**if**(counter2 >= 2)

counter2 = 0;

data = (counter2 == 0) ? 0 : 1<<counter1;

**break**;

**case** 3:

counter2++;

**if**(counter2 >= 2)

counter2 = 0;

data = (counter2 == 0) ? 0 : 1<<counter1;

**break**;

**case** 4:

data = 1<<counter1;

**default**:

**break**;

}

Xil\_Out16(XPAR\_AXI\_GPIO\_0\_BASEADDR+XGPIO\_DATA2\_OFFSET, data);

}

**四、心得体会**

这次实验，加深了我对程序控制方式、快速中断方式两种实现方式的理解。程序控制方式中没有中断，故需要循环读取按键状态；快速中断方式在中断系统初始化方面有一些区别，快速中断方式需要配置IMR，还需填写IVAR。通过编写程序，我更加理解了中断系统初始化程序由主函数调用，中断服务程序由硬件中断请求调用的含义，加深了我对中断技术的理解。同时C语言熟练程度也得到了很大的提升。

我亲自搭建了一个基于Microblze微处理器的微型系统平台，来运行自己的C语言代码实现一定的功能，非常有成就感。