一. 实验任务

基于嵌入式MicroBlaze微处理器设计一个同时支持多种并行IO设备工作的嵌入式MIMO系统，采用FPGA开发板Nexys4。该系统的基本输入输出设备有：16个独立LED灯，16个独立开关、5个独立按键，8个七段数码管，外设接口电路如图1所示：

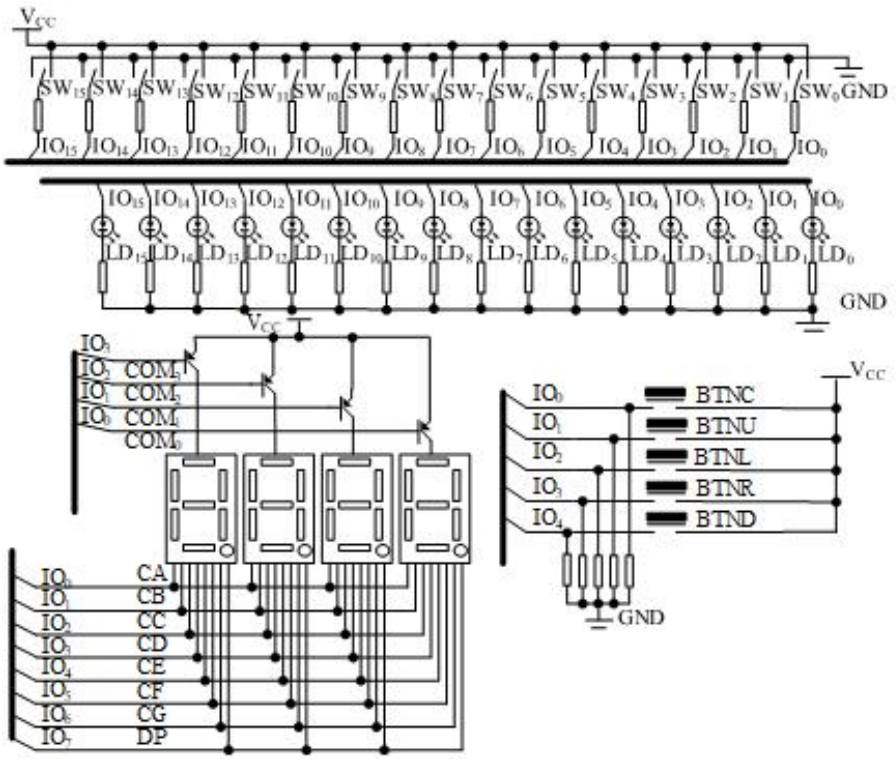

图1 外设接口电路

要求所有外设都通过GPIO连接到MicroBlaze微处理器构成的计算机系统的同一总线上。分别通过程序控制方式和并行IO接口中断方式实现不同的功能：

1. 程序控制方式实现：

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，实现以下功能：

(1) 按下BTNC 按键时，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BTNR 按键时，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BTNU 按键时，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 BTND 按键时，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。

（程序控制方式提示：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。）

2. **并行 IO 接口中断方式实现：**

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，七段数码管作为输出设备。实现以下功能：

(1) 按下 BTNC 按键时，计算机读入一组 16 位独立开关状态作为一个二进制数据，并将该二进制数的低 8 位对应的二进制数值 0 或 1 显示到 8 个七段数码管上。

(2) 按下 BTNU 按键时，计算机读入一组 16 位独立开关状态作为一个二进制数据，并将该 16 进制数据各位数字对应的字符 0~F 显示到低 4 位七段数码管上（高 4 位七段数码管不显示）。

(3) 按下 BTND 按键时，计算机读入一组 16 位独立开关状态作为一个二进制数据，并将该数据表示的无符号十进制数各位数字对应的字符 0~9 显示到低 5 位七段数码管上（高 3 位七段数码管不显示）。

设计各种控制方式下的控制程序（要求设计程序框架结构、各个函数之间的关联关系、各个函数的执行流程图并说明原因、写出程序源代码）。

二. 实验目的

1. 掌握 GPIOIP 核的工作原理;
2. 掌握 IO 接口程序控制方法;
3. 掌握中断控制方式的 IO 接口设计原理;
4. 掌握中断程序设计方法。

三. 实验环境

1. Windows 10 操作系统 ；
2. 嵌入式软件开发平台：Vivado 2018.1
3. 硬件平台开发板：Xilinx Nexys4。

四． 设计方案

(1) 程序控制方式实现任务 1：

任务 1 中共有两种输入设备，一种输出设备：分别是独立按键以及独立开关作为输入设备，LED 灯作为输出设备。

采用程序控制方式实现时，只需要一个主程序来实现，在主程序中设置 GPIO 各个通道的工作模式，并开始循环读取按键状态，如果按键状态改变，则根据按键的不同值，进入不同的 if 语句，来实现相应的功能：

① 当按键值为 0x10，即 BTNC 被按下时，读取此时对应开关状态，存为 csw1，并将其显示在 LED 灯上；

② 当按键值为 0x10，即 BTNR 被按下时，读取此时对应开关状态，存为 csw2，并将其显示在 LED 灯上；

③ 当按键值为 0x10，即 BTNU 被按下时，结果取 r1 = csw1 + csw2，并将结果 r1 显示在 LED 灯上；

④ 当按键值为 0x10，即 BTND 被按下时，结果取 r2 = csw1 *csw2，并将结果 r2 显示在 LED 灯上.

流程图如图 2 所示：

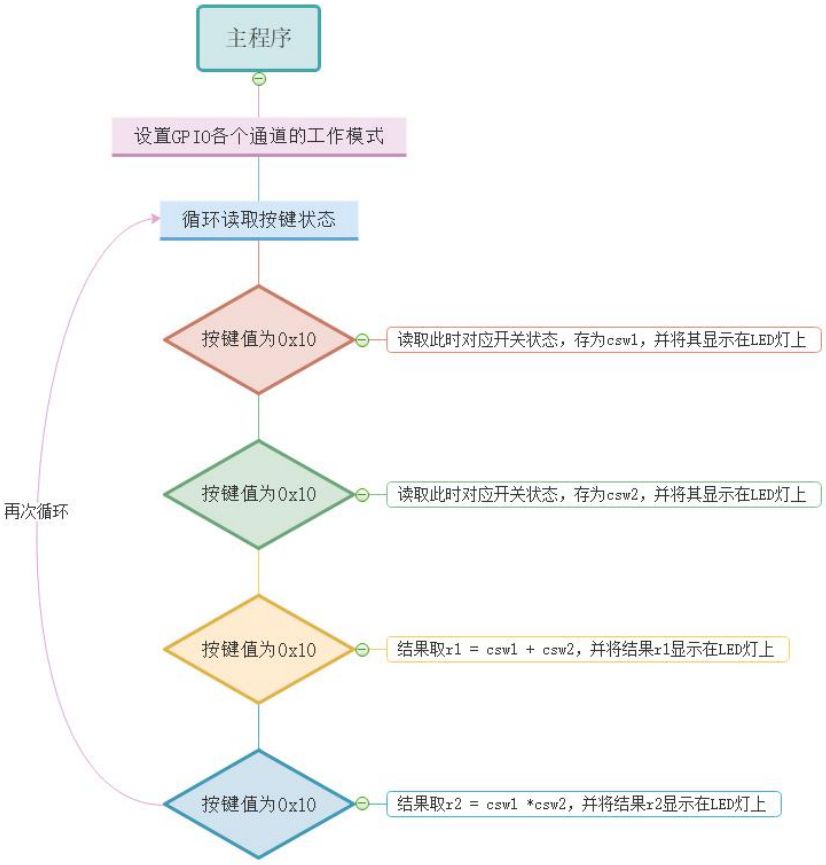

图 2　程序控制方式主程序流程图

(2) 并行 IO 接口中断控制方式实现任务 2：

分析题目可知，主要需要实现 2 个任务：

① 七段数码管动态显示：

七段数码管动态显示电路也需要硬件定时中断，动态显示利用视觉暂留效应，定时间隔不能被人察觉，使用 Timer_0，中断程序为：Timer_0 中断时，中断事务处理为点亮当前位置的七段数码管，即输出当前位置的七段数码管段码和位码，并且将七段数码管点亮位置修改为下一位。其中，Timer_0 的定时间隔需满足视觉暂留效应要求即可。

② 开关与按键状态输入：

开关状态输入以及按键状态输入都在按键的 GPIO 中断事务处理实现。按键的 GPIO 中断事务处理除了读入开关状态的功能外，还需根据开关状态更新七段数码管显示缓冲区。

所以，控制程序可以分为四个函数，他们的层级结构如图 3 所示：

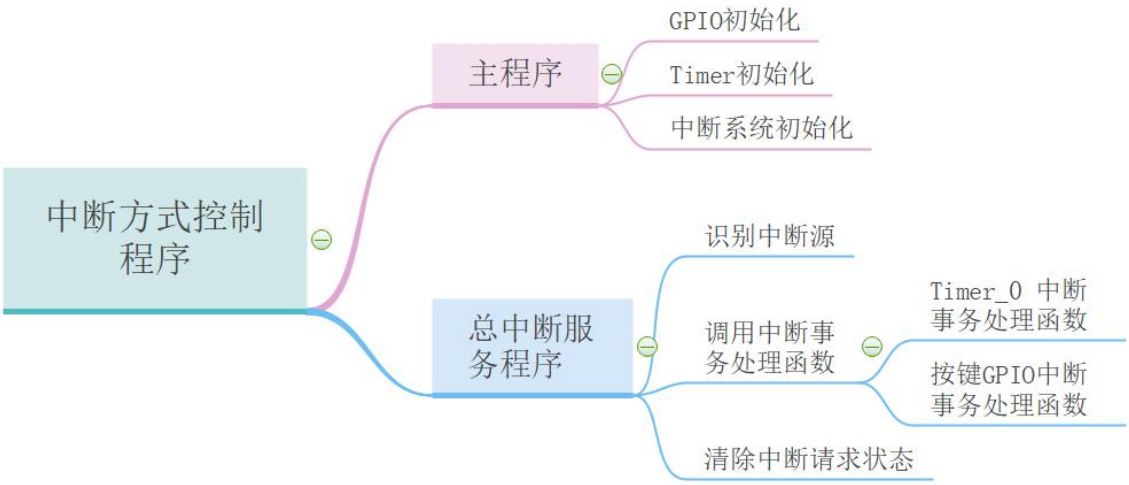

图 3 中断控制方式程序结构图

其中，各个函数的功能与流程如下所示：

①中断初始化程序：

功能与流程图如图 4 所示：

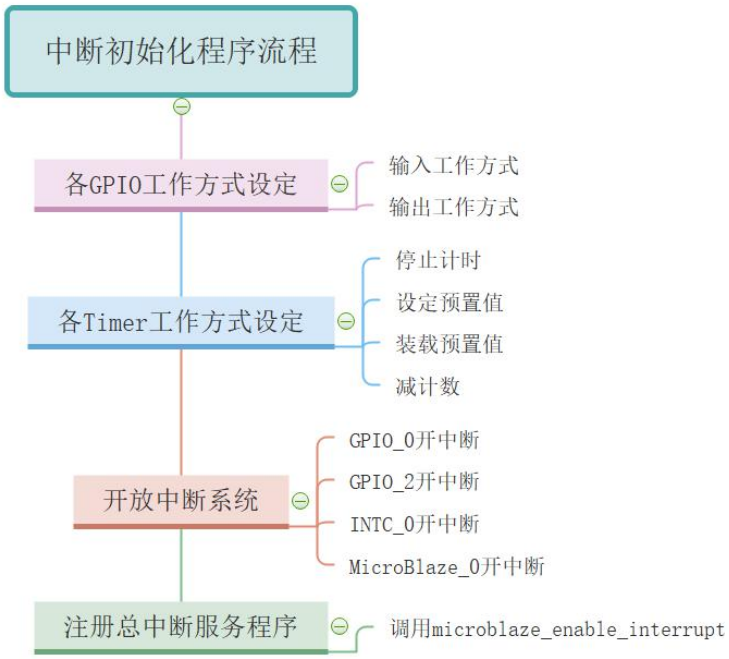

图 4 中断初始化程序流程图

②总中断服务程序：

功能与流程图如图 5 所示：

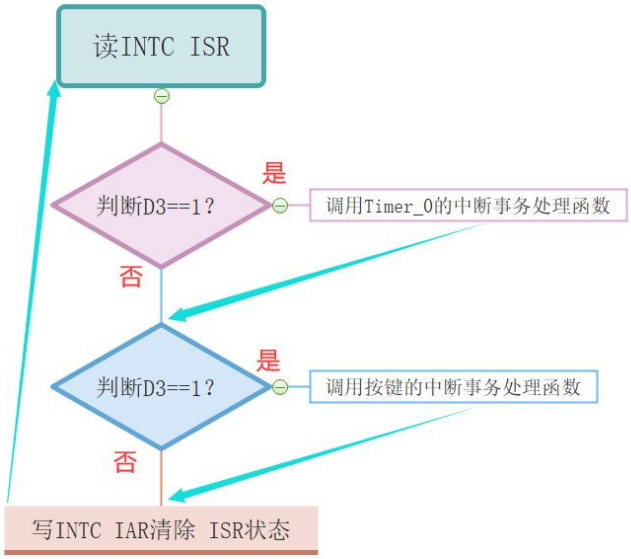

图 5 总中断服务程序流程图

③Timer_0 的中断事务处理函数：

功能与流程图如图 6 所示：

图 6 Timer_0 的中断事务处理函数流程图

④按键的中断事务处理函数：

功能与流程图如图 7 所示：

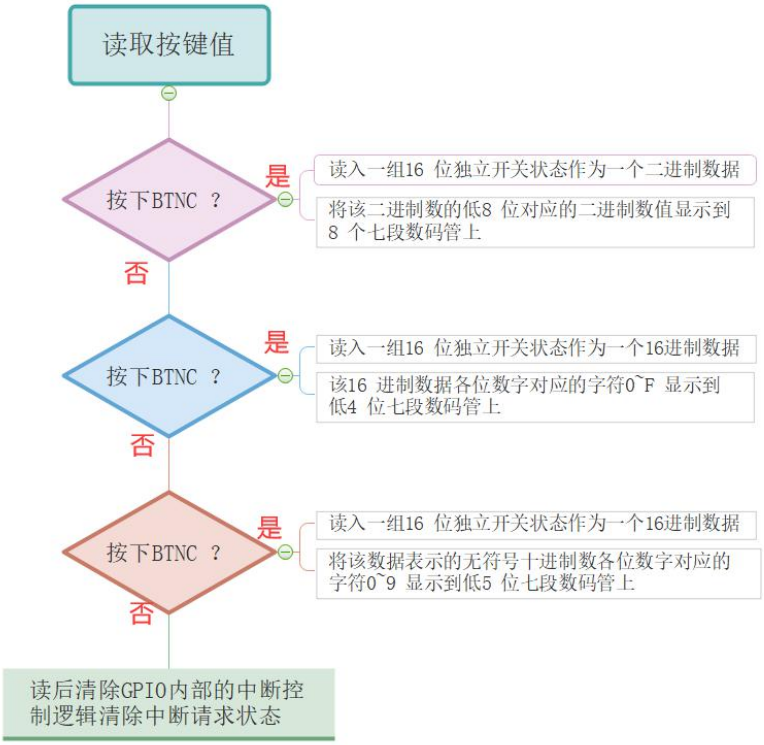

图 7 按键的中断事务处理函数流程图

五. 实现过程

(1) 硬件平台搭建：

在 Vivado 2018.1 中，使用 Xilinx Nexys4 开发板，搭建基于 MicroBlaze 软核的

嵌入式系统硬件平台如下图 8 所示：

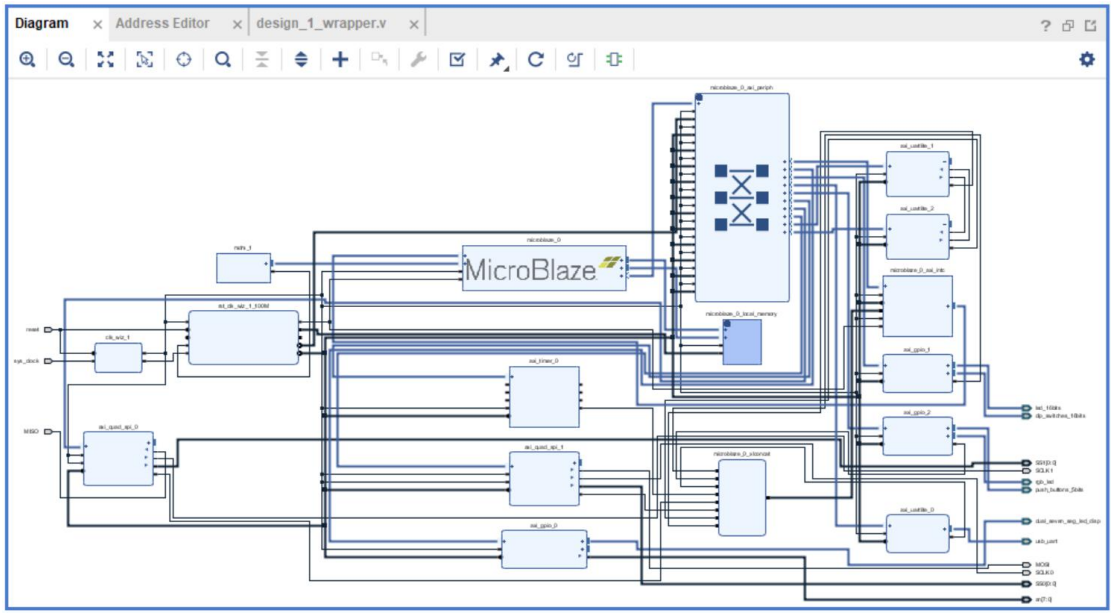

图 8 嵌入式系统硬件平台总框图

中断方式使用一个中断控制器：其中 GPIO_0 中断输出连接到 Intr0，GPIO_2 中断输出连接到 Intr1，Timer_0 中断输出连接到 Intr2；中断控制器的中断向量输出连接到 MicroBlaze 微处理器的中断输入总线上。

其中对常用并行 IO 外设 GPIO 接口进行了配置（如图 9）：

16 位开关和 16 位 LED 灯共用一个 GPIO IP 核（设为 GPIO_1），其中，开关使用 GPIO 通道，LED 灯使用 GPI2 通道；四位七段数码管的位码与段码共用另一个 GPIO IP 核（设为 GPIO_0），其中，位码使用 GPIO 通道，段码使用 GPIO2 通道；两位按键使用另一个 GPIO IP 核（设为 GPIO_2）的 GPIO 通道；延时 T0 与 T1 使用 Timer IP 核（设为 Timer_0）。

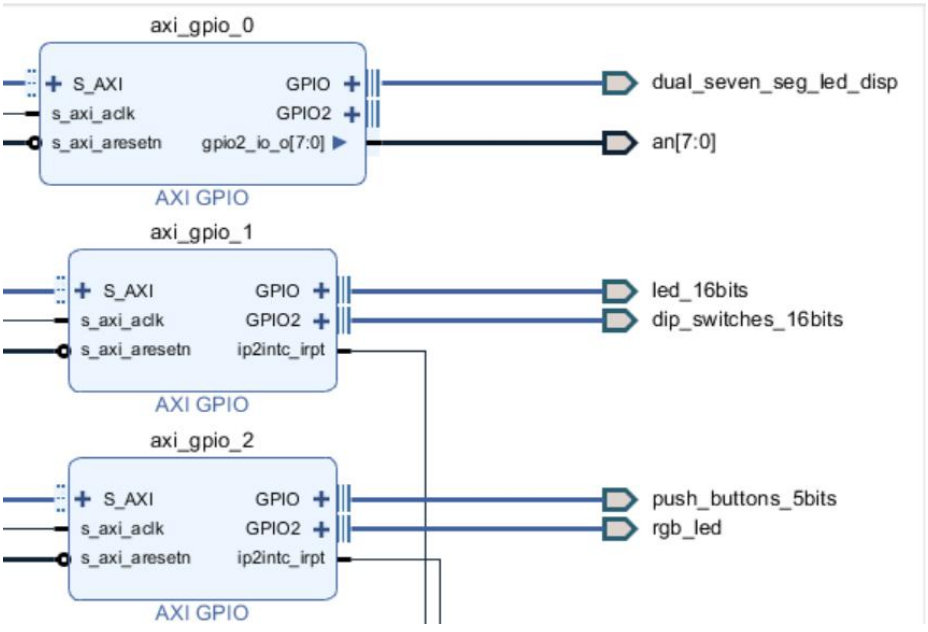

图 9 外设 GPIO 接口配置

对并行 IO 中断系统进行了配置（如图 10）：
中断方式使用一个中断控制器：其中 GPIO_1 中断输出连接到 Intr0;GPIO_2 中断输出连接到 Intr1；UART_0 中断输出连接到 Intr2；Timer_0 中断输出连接到 Intr3；SPI_0 中断输出连接到 Intr4； SPI_1 中断输出连接到 Intr5;；UART_1 中断输出连接到 Intr6；UART_2 中断输出连接到 Intr7.
中断控制器的中断向量输出连接到 MicroBlaze 微处理器的中断输入总线上。
如图所示：

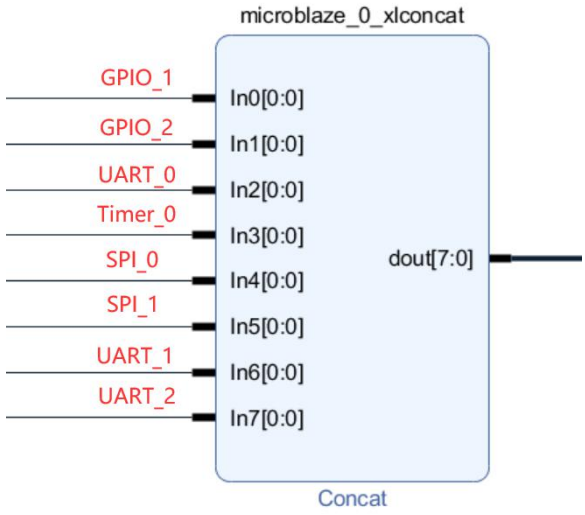

图 10 并行 IO 中断系统配置

对串行 IO 接口外设 UART、SPI 进行了配置（如图 11）：

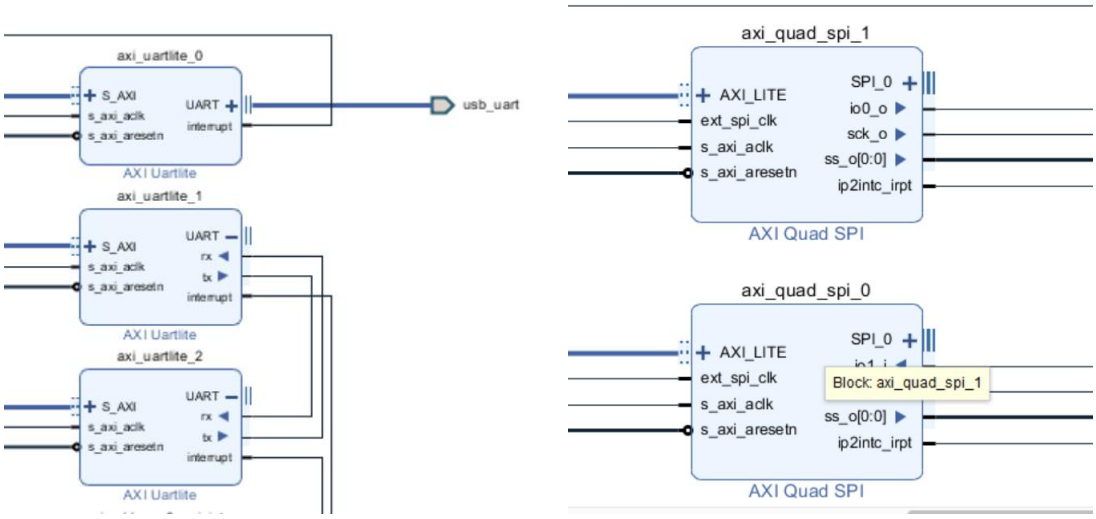

图 11 串行 IO 接口外设 UART、SPI 进行了配置

生成 HDL 封装，查看硬件平台存储空间布局如下（如图 12）：

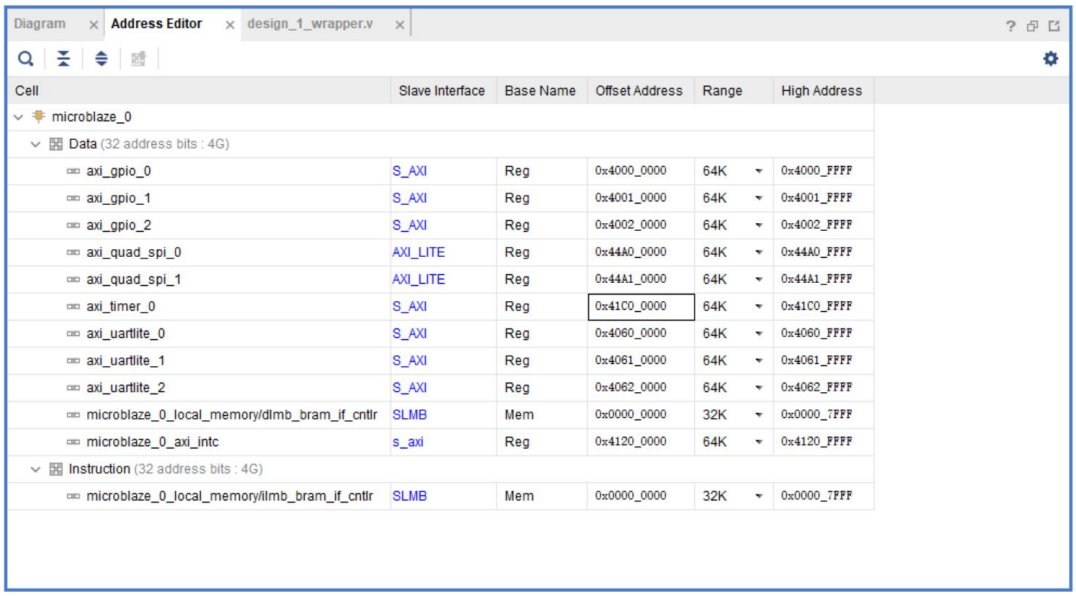

图 12 按硬件平台存储空间布局

(2) 程序控制方式实现任务 1：

根据上面的分析，在主程序中设置按键对应 GPIO 通道工作在输入模式，开关对应的 GPIO 通道工作在输入模式，LED 灯对应的 GPIO 通道工作在输出模式，设置并开始循环读取按键状态，如果按键状态改变，则根据按键的不同值，进入不同的 if 语句，来实现相应的功能：

① 当按键值为 0x10，即 BTNC 被按下时，读取此时对应开关状态，存为 csw1，并将其显示在 LED 灯上；

② 当按键值为 0x10，即 BTNR 被按下时，读取此时对应开关状态，存为 csw2，并将其显示在 LED 灯上；

③ 当按键值为 0x10，即 BTNU 被按下时，结果取 r1 = csw1 + csw2，并将结果 r1 显示在 LED 灯上；

④ 当按键值为 0x10，即 BTND 被按下时，结果取 r2 = csw1 *csw2，并将结果 r2 显示在 LED 灯上.

完整代码如下：

```
#include "stdio.h"
#include "xil_io.h"
#include "xgpio.h"

int main()
{
    short button;
    unsigned short csw1,csw2,r1,r2;
    Xil_Out8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_TRI_OFFSET,0x1f);
```

```
    Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI2_OFFSET,0xffff);
    Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI_OFFSET,0x0);

    while(1)
    while((Xil_In8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_DATA_OFFSET)&0x1f)!=0)
    {
            button = Xil_In8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_DATA_OFFSET)&0x1f;
            while((Xil_In8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_DATA_OFFSET)&0x1f)!=0);
            xil_printf("The pushed button's code is %d\n" , button);
            if(button==0x10 ){
                csw1 = Xil_In16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA2_OFFSET)&0xffff;
                Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA_OFFSET,csw1);
            }
            if(button ==0x8){
                csw2 = Xil_In16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA2_OFFSET)&0xffff;
                Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA_OFFSET,csw2);
            }
            if(button ==0x1){
                r1 = csw1 + csw2;
                Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA_OFFSET,r1);
            }
            if(button ==0x4){
                r2 = csw1 * csw2;
                Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA_OFFSET,r2);
            }
    }

}
```

(3) 并行 IO 接口中断控制方式实现任务 2：

a) GPIO 初始化：

对于程序控制方式，依据 GPIO 表格，写 GPIO 的 TRI 寄存器设置输入输出模式；

寄存器名称	偏移地址	初始值	含　义
GPI/O_DATA	0x0	0	32 位宽，通道 GPI/O_I/O 数据寄存器，b_i 对应 GPI/O_I/O$_i$
GPI/O_TRI	0x4	0xffffffff	32 位宽，通道 GPI/O_I/O 控制寄存器，b_i 对应 GPI/O_I/O$_i$ 的传输方向控制。1-输入；0-输出
GPI/O2_DATA	0x8	0	32 位宽，通道 GPI/O2_I/O 数据寄存器，b_i 对应 GPI/O2_I/O$_i$
GPI/O2_TRI	0xc	0xffffffff	32 位宽，通道 GPI/O2_I/O 控制寄存器，b_i 对应 GPI/O2_I/O$_i$ 的传输方向控制。1-输入；0-输出

对于中断控制方式，还需要写 IER 寄存器以及 GIER 寄存器

名称	偏移地址	含义
GIER	0x11c	全局中断使能寄存器，D_{31}=1 使能中断信号 ip2intc_irpt 输出,其余位无意义
IPIER	0x128	中断使能寄存器，控制各个通道是否允许产生中断 D_0=1 使能通道 GPIO 中断；D_1=1 使能通道 GPIO2 中断，其余位无意义
IPISR	0x120	中断状态寄存器，读：获取通道中断状态，写：清除中断状态 读：D_0=1 通道 GPIO 产生了中断；D_1=1 通道 GPIO2 产生了中断 写：D_0=1 清除通道 GPIO 中断状态；D_1=1 清除通道 GPIO2 中断状态

实现代码如下：

```
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI_OFFSET,0x0);//设段码输出方式
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI2_OFFSET,0x0);//设位码为输出方式
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_TRI_OFFSET,0x1f);//设地 BUTTON 方输入方式
Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI2_OFFSET,0xffff);//设地 Switch 方输入方式
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//允许中断
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//GPI
O 中断输出
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,
```

b） Timer_0 初始化：

Timer_0 初始化的操作流程为：首先停止计时（通过写 TCSR 寄存器使使能定时器这一位为 0 实现），然后再是写预置值（通过写 TLR 寄存器实现），紧接着装载预置值（通过写 TCSR 寄存器使装载这一位为 1 实现）然后再是写 TCSR 寄存器，控制定时器使能。清除源中断状态、使能中断、使能自动装载、减计数。

实现代码如下：

```
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，停止计数
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE);//TLR,预置计数值
Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,
    Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)|XTC_CSR_LOAD_MASK);//
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_C
SR_DOWN_COUNT_MASK);
```

c） INTC 初始化与微处理器开中断：

INTC 内部寄存器如下：

寄存器名称	偏移地址	含义
ISR	0x0	中断状态寄存器
IPR*	0x4	中断悬挂寄存器
IER	0x8	中断使能寄存器

IAR	0xC	中断响应寄存器
SIE*	0x10	中断使能设置寄存器
CIE*	0x14	中断使能清除寄存器
IVR*	0x18	中断类型码寄存器
MER	0x1C	主中断使能寄存器
IMR*	0x20	中断模式寄存器
ILR*	0x24	中断级别寄存器
IVAR*	0x100~0x170	中断向量表寄存器

对中断控制器开中断：首先清除原中断状态，然后使能 intr1 和 intr3 对应的中断输入并使能中断输出（分别通过写 IAR、IER 以及 MER 寄存器实现）；
MicroBlaze 微处理器开中断：要实现微处理器开中断，我们可通过调用 microblaze_enable_interrupt 实现：
实现代码如下：

```
    //INTC 初始化
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IER_OFFSET,XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPA
R_AXI_TIMER_0_INTERRUPT_MASK);//
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_MER_OFFSET,0x3);

    //微处理器开中断
    microblaze_enable_interrupts();
```

d） 总中断服务程序：

总中断服务程序，首先读取中断控制器的中断状态寄存器；然后判断 D3 是否为 1，如果是 1 则表示按键状态发生了变化，因此调用按键对应的 GPIO 中断事务处理函数，返回之后，判断 D3 是否为 1 如果是 1 则表示 Timer_0 定时器计时时间到，因此调用 Timer_0 对应的中断事务处理函数。
实现代码如下：

```
void My_ISR()
{
    int status;
    status=Xil_In32(XPAR_INTC_0_BASEADDR+XIN_ISR_OFFSET);//续敢 ISR
    if((status&0x8)==0x8)
        Seg_TimerCounterHandler();//l 溉用用户中断服务狂序
    else if((status&0x2)==0x2)
        BtnHandler();//l 遴用按痉中断
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IAR_OFFSET,status);//写 IAR
}
```

e） Timer_0 的中断事务处理函数：

T0 通过 gpio_0 的 GPIO2 通道控制一位 LED 灯点亮，并准备下一位的输出后直接退出。
实现代码如下：

```
void Seg_TimerCounterHandler()
{
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA_OFFSET,segcode[j]);
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA2_OFFSET,pos);
    pos=pos>>1;
    j++;
    if(j==8)
    {
        j=0;
        pos=0xff7f;
    }
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,Xil_In32(XPAR_AXI_TIMER_0_BASEA
DDR+XTC_TCSR_OFFSET));//满总中断
}
```

f） 按键的中断事务处理函数：

首先读取按键的状态，根据按键的值来实现相应功能：

实现代码如下：

```
void BtnHandler()
{
    int button;
    unsigned short sw;
    int i;
    int temp;
    int q,w,e,r,t;
        sw = Xil_In16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA2_OFFSET)&0xffff;
        button = Xil_In8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_DATA_OFFSET)&0x1f;

        if(button == 0x10)  //(1)
        {
            short pos1 = 0x0080;
            for(i=0;i<8;i++)
            {
                if((sw&pos1) != 0)
                    temp = 1;
                else
                    temp = 0;
                segcode[i] =  segtable[temp];
                pos1=pos1>>1;
            }

        }
        else if(button == 0x1)  //(2)
```

```
        {
            for(i=0;i<8;i++)
            {

                if(i>3){
                   temp = ( sw >>(4*(7-i)))&0xf;
                   segcode[i]=segtable[temp];
                }
                else
                    segcode[i] = 0xff;

            }
        }else if(button == 0x4) //(3)
        {
            q = sw/10000;
            w = (sw-10000*q)/1000;
            e = (sw-10000*q-1000*w)/100;
            r = (sw-10000*q-1000*w-100*e)/10;
            t = (sw-10000*q-1000*w-100*e-10*r)/1;
            xil_printf("q=%d,w=%d,e=%d,r=%d,t=%d,",q,w,e,r,t);
                for(i=0;i<8;i++)
                {
                    if(i>2){
                        switch(i)
                        {
                            case 3:temp =q;break;
                            case 4:temp =w;break;
                            case 5:temp =e;break;
                            case 6:temp =r;break;
                            case 7:temp =t;break;
                            default: break;
                        }
                        segcode[i] =  segtable[temp];
                    }
                    else
                        segcode[i] = 0xff;
                }

        }
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_ISR_OFFSET,
        Xil_In32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_ISR_OFFSET));
}
```

g） 完整代码：

最终的实现代码如下（上面的中断事务处理函数已折叠）：

```
#include "xil_io.h"
#include "stdio.h"
#include "xgpio_l.h"
#include "xintc_l.h"
#include "xtmrctr_l.h"
#define RESET_VALUE 100000
void Seg_TimerCounterHandler();
void BtnHandler();
void My_ISR()__attribute__((interrupt_handler));//总中断服务雅序

char segtable[16] = { 0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,
        0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e };

char segcode[8]={0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};//缓冲区

short pos=0xff7f;
int j = 0;
int main()
{

    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI_OFFSET,0x0);//设段码输出方式
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI2_OFFSET,0x0);//设位码为输出方式
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_TRI_OFFSET,0x1f);//设地 BUTTON 方输入方式
    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI2_OFFSET,0xffff);//设地 Switch 方输入方式
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);//允许中断
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);//
GPIO 中断输出
    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);//
琛 TCSR，停止计数
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE);//TLR,预置计数值
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,
        Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET)|XTC_CSR_LOAD_MASK);//
    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_C
SR_DOWN_COUNT_MASK);


    //INTC 初始化
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IER_OFFSET,XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPA
R_AXI_TIMER_0_INTERRUPT_MASK);//
    Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_MER_OFFSET,0x3);
```

```
    //微处理器开中断
    microblaze_enable_interrupts();
    return 0;
}

void My_ISR()……

void Seg_TimerCounterHandler()……

void BtnHandler()……
```

六． 实验结果

(1) 程序控制方式实现任务 1：

a) 按下 BTNC 按键时，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。(如图 13）
输入为： 0000000000101101（十进制：45）

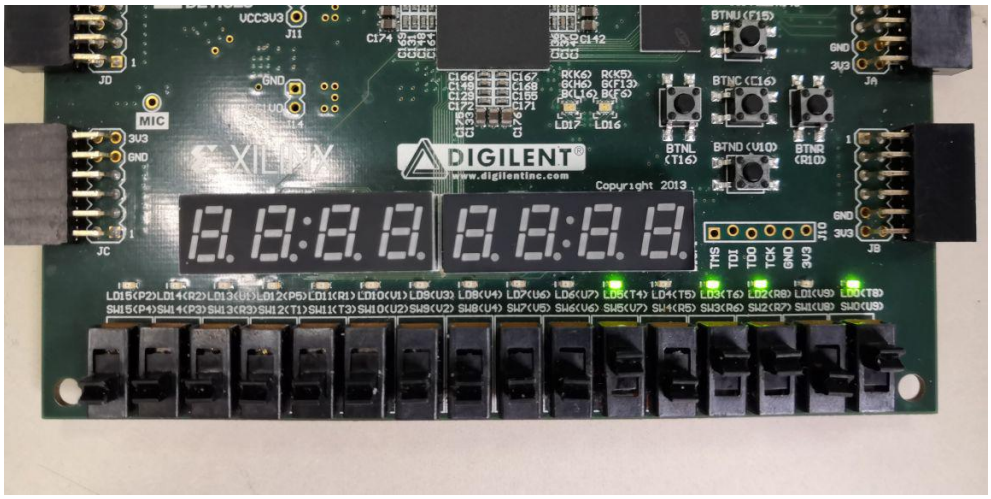

图 13 按下按键 BTNC

b) 按下 BTNR 按键时，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。(如图 14）
输入为： 0000000011010010（十进制：210）

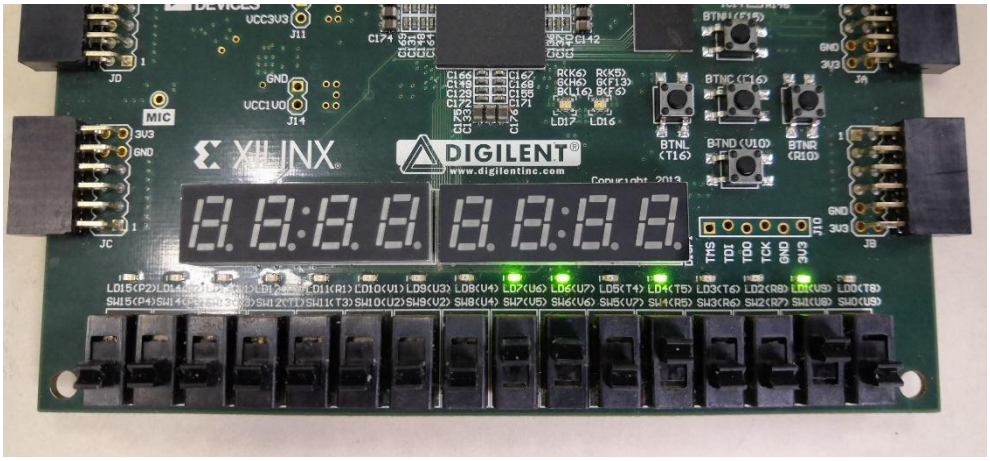

图 14 按下按键 BTNR

c) 按下 BTNU 按键时，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。(如图 15)

输出为：0000000011111111（十进制：255）

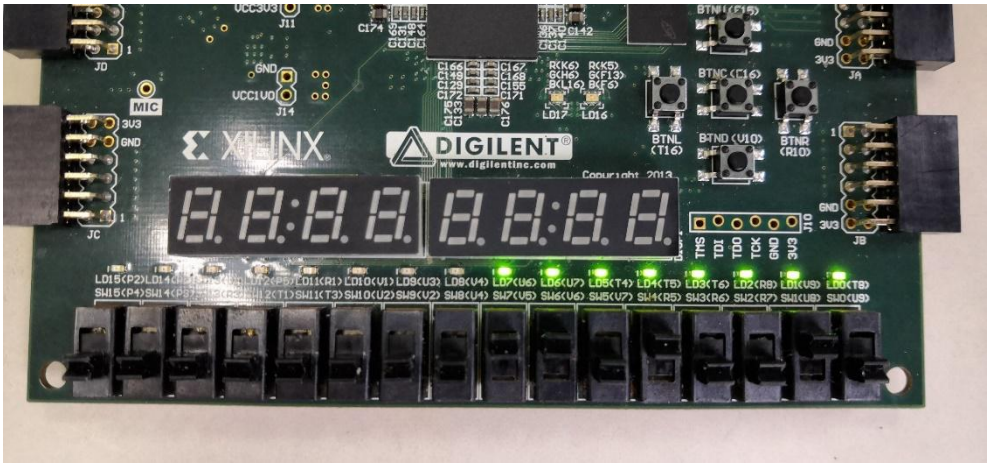

图 15 13 按下按键 BTNU

d) 按下 BTND 按键时，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。(如图 16)

输出为：0010010011101010（十进制：9450）

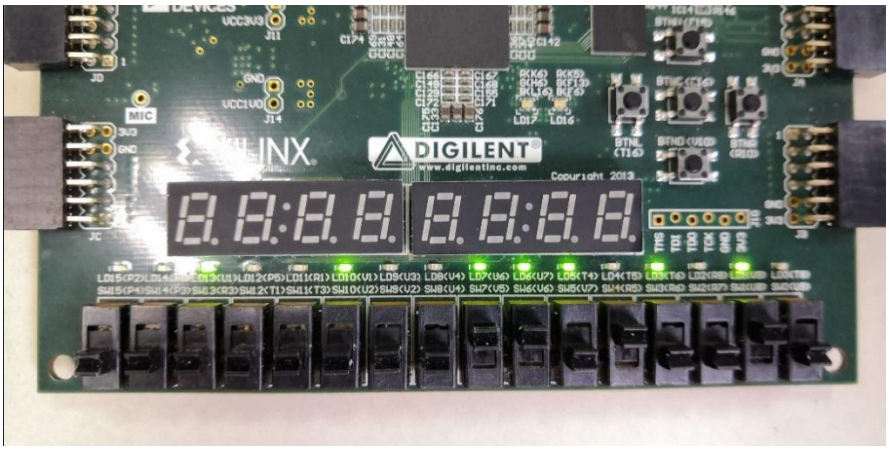

图 16 按下按键 BTND

(2) 并行 IO 接口中断控制方式实现任务 2：

a) 按下 BTNC 按键时，计算机读入一组 16 位独立开关状态作为一个二进制数据，并将该二进制数的低 8 位对应的二进制数值 0 或 1 显示到 8 个七段数码管上。（如图 17）

输入为：0000000010101101

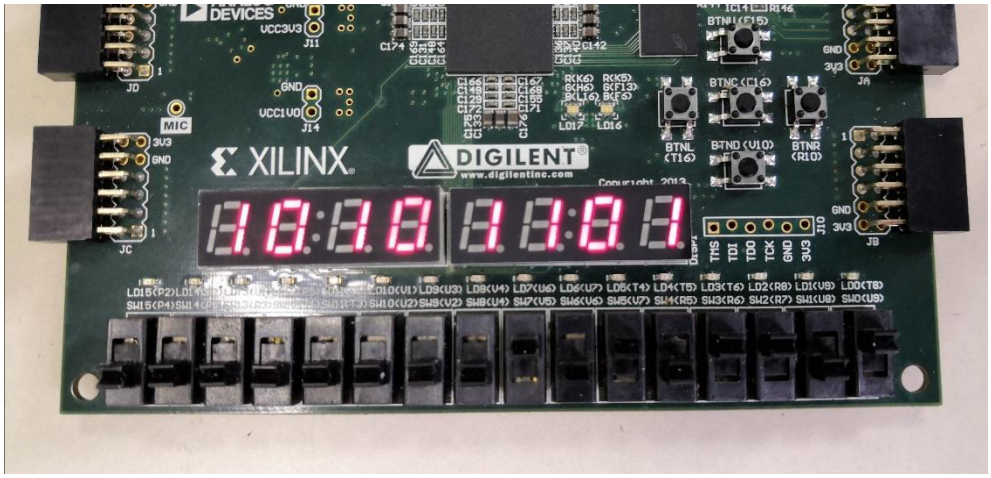

图 17 按下按键 BTNC

b) 按下 BTNU 按键时，计算机读入一组 16 位独立开关状态作为一个二进制数据，并将该 16 进制数据各位数字对应的字符 0~F 显示到低 4 位七段数码管上(高 4 位七段数码管不显示)。(如图 18）

输入为：1111010101001101（0xf54d）

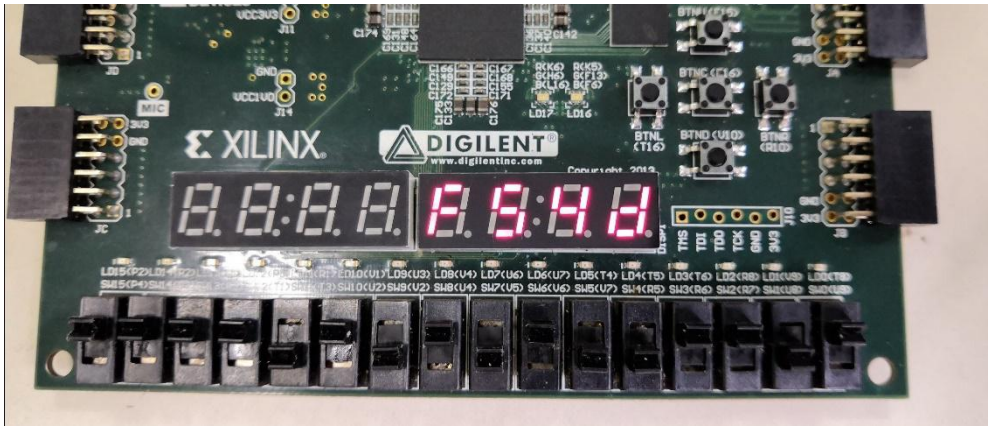

图 18 按下按键 BTNU

c) 按下 BTND 按键时，计算机读入一组 16 位独立开关状态作为一个二进制数据，并将该数据表示的无符号十进制数各位数字对应的字符 0~9 显示到低 5 位七段数码管上（高 3 位七段数码管不显示）。(如图 19）

输入为：0001010101010101（十进制：5461）

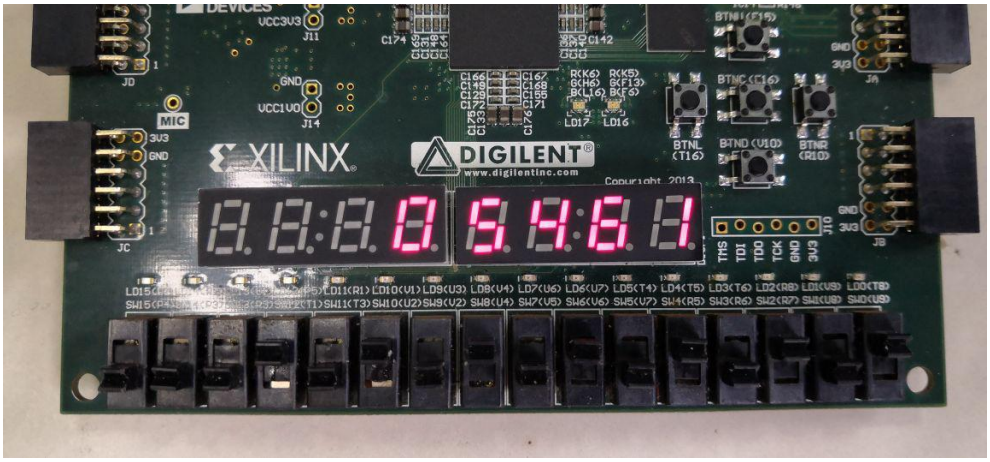

图 19 按下按键 BTND

七. 方式比较

程序控制方式	中断方式
• IO 接口设计简单. • 程序代码简洁明了. • CPU 和外设之间交换信息采用查询方式.，使得快速 CPU 与慢速外设之间矛盾 • 在反复查询上浪费了 CPU 的大部分时间. • 程序配置比较灵活	• CPU 实行分时操作，从而大大提高了计算机的效率. • 能够立即响应与处理. • 由软件维护中断向量表 • 通过一个总中断服务程序查询 ISR 寄存器识别中断源，来调用各个中断服务函数 • 通过软件写 IAR 和清除 ISR

八. 实验总结

这次实验是微机原理实验中最重要的一次实验。通过这次的实验，我学到了很多，也收获了很多。不仅对《微机原理与接口技术》课程的相关知识有了更为深入的了解，对使用中断控制方式实现各种较复杂的任务有了亲身的体会，也对独立自主学习与独立完成任务有了进一步的实践，主要有以下一些感受和认识：

1. **中断控制器在计算机系统实现中断管理中至关重要.**

由中断控制器来实现中断请求信号保持与清除、中断源识别、中断使能控制、中断优先级设置等功能，配合微处理器实现与外设之间的中断方式通信。

2. **工作模式有快速中断与普通中断.**

普通中断还是快速中断可由软件设定，他们之间的区别为：快速中断模式时，INTC 维护硬件中断向量表，在中断响应周期向微处理器提供中断向量，根据微处理器中断响应周期提供的中断响应状态信号自动清除中断状态;普通中断模式时，由软件维护中断向量表以及读取中断状态寄存器识别中断源，并且软件需在中断服务程序中写中断响应寄存器清除 INTC 中断状态。

3. **中断方式程序设计.**

中断方式程序设计包含两个程序:中断系统初始化程序和中断服务程序。它们的功能不同，但都十分重要，缺一不可。中断系统初始化程序主要完成设备初始化和中断系统初始化，实现如使能中断、填写中断向量表等功能；而中断服务程序主要完成中断事务处理以及清除中断状态等。中断服务程序由硬件中断源调用，因此不带参数。但是，当设备中断服务程序由总中断服务程序调用时，则可以带参数，此时设备的中断服务程序并非真正意义上的中断服务程序，而可以看作是总中断服务程序的一个子程序。

4. **中断执行流程.**

微处理器检查是否存在中断，需要在执行完现行指令且设置开中断。若有中断，则进入中断响应周期，流程为：关中断、保护断点；=> 读取中断向量、转入中断服务程序；=> 中断服务；=> 中断返回、开中断。

5. **多中断源.**

当有多个中断源时，中断发生时，不同的中断服务程序各自处理完成自己的任务，不会引起混乱；但每个中断事务要求要尽可能简单，如果过于复杂的话，可能会阻塞其他中断源，造成难以预料的错误。

6. **使用全局变量与公共缓冲区.**

因为中断服务程序相互之间不存在子、主程序关系，所以无法通过参数调用实现信息互传。因此，需要使用全局变量来完成不同程序之间的信息传递，比如说使用多个数码管显示时。同时，为了简化开发，在 C 语言中我们可以把地址用宏定义来代替。

7. **代码纠错.**

因为我们在开始开发前，总可能会没有考虑到实际会遇到的一些问题与困难，很多时候第一次写出来的代码无法实现理想的效果，会出现各种各样的问题。在解决问题时，我们应当学会对整个问题进行细化，学会对每部分进行一个流程的设计，学会先对每个模块进行分析，这样降低了问题的难度，也使得自己写代码更加顺畅，头脑更具有条理性，这样的分析、解决问题的方法，也是本次实验以及报告贯穿始终的。

通过这次实验，我对中断技术有了更加直观的了解，中断方式是计算机系统应用最为广泛的接口通信方式，同时，当许算机系统运行复杂软件系统时，中断方式也是软件各模块之间通信方式之一；对各种不同的实现方法也有了更加深刻地了解，不仅巩固了所学，也提升了各方面的能力，对计算机实现各种各样的更复杂的任务的底层逻辑有了一个初步的了解。

总之，这是一次意义丰富、收获众多的实验经历，我从中学到了很多！