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。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，4个七段数码管，外设接口电路如图1所示。

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。通过多种IO数据传输控制方式同时实现以下功能：

1）16个LED灯走马灯式轮流循环亮灭。且循环速度可通过两个独立按键步进控制，其中一个按键每按一次步进增速，另一个按键每按一次步进减速。

2）4个七段数码管实时显示16位独立开关表示的十六进制数。

IO数据传输控制方式分为以下几种：

1）程序控制方式（要求不能与教材或视频设计方案完全一致）

2）普通中断方式（要求不能与教材或视频设计方案完全一致）

3）快速中断方式（要求不能与教材或视频设计方案完全一致）

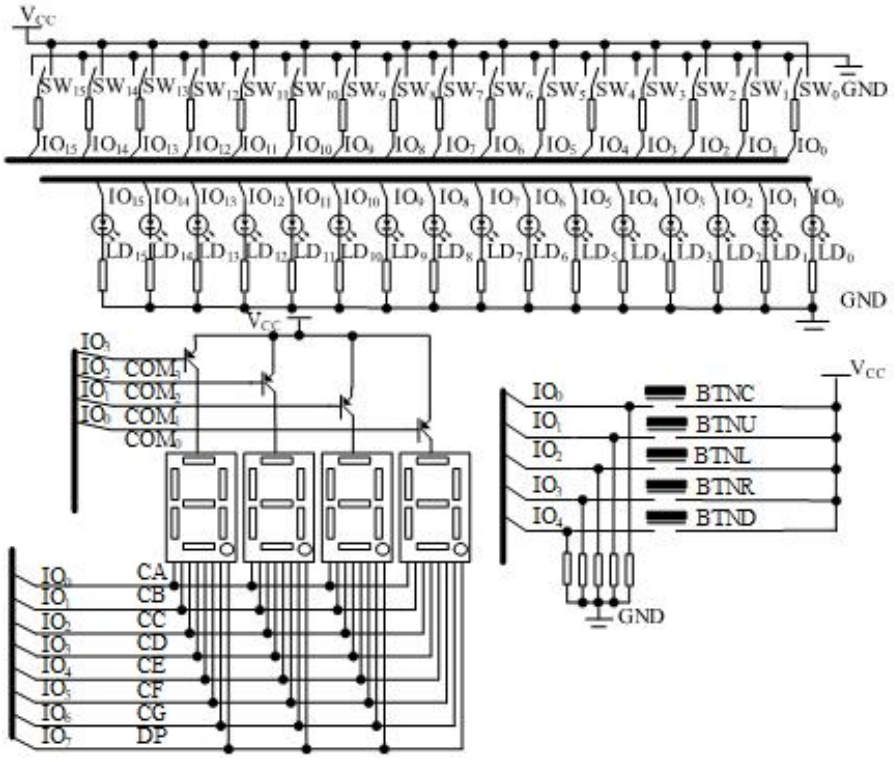

图 1 外设接口电路

分别设计接口电路和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），并讨论和比较各种控制方式控制程序设计的优缺点和可能出现的实验现象和成因分析。（如在什么情况下可能出现对按键或开关状态不能及时响应，原因是什么？如何改进？）

二.实验目的

1. 掌握GPIO IP 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. 软件设计平台：Code::Blocks 17.12

四.设计思路

1.题目分析:

(1) LED 灯流水灯:

LED 灯流水灯需要硬件定时中断，可用一个定时器来实现（设为 T0)。由题意得，流水灯定时时间间隔要求可调，且可长可短，所以 T0 的中断程序为，中断时：中断事务处理为点亮当前位置的 LED 灯，并且将需要点亮的 LED 灯位置修改为下一位。实现循环速度改变功能，可由按键步进调节 T0 的定时间隔（初始时设定为默认值)。

回顾教材中的 Timer 中断如图 2 所示:

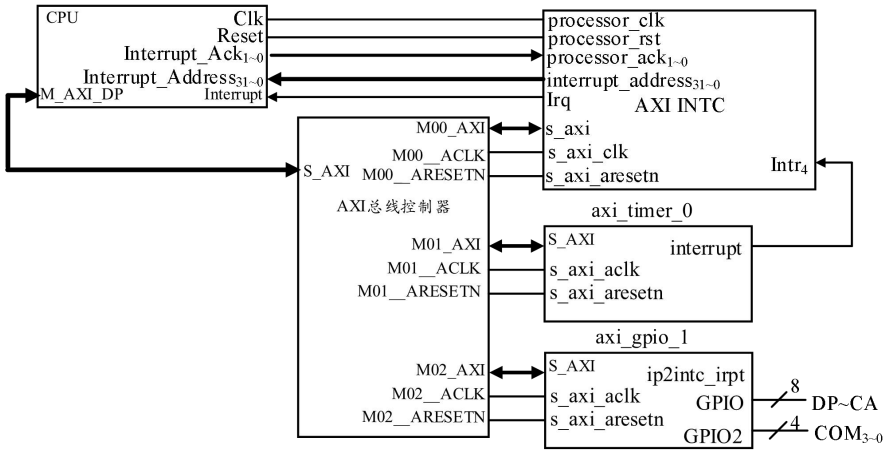

图 1 回顾 Timer 中断

(2) 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，所以不能继续使用 T0，故采用另一定时器（设为 T1）实现定时控制。由题意得，T1 的中断程序为：TI 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，TI 的定时间隔需满足视觉暂留效应要求即可。

(3) 开关与按键状态输入：

开关状态输入以及按键状态输入都由 GPIO 中断事务处理实现。其中开关对应的 GPIO 中断事务处理除了读入开关状态，还需根据开关状态更新七段数码管显示缓冲区。按键对应的 GPIO 中断事务处理除了读入按键状态,还需根据所按按键修改 T0 的装载值，即写 TLR0 寄存器，以便调节 T0 的定时间隔。

回顾教材中的 GPIO 中断如图 3 所示：

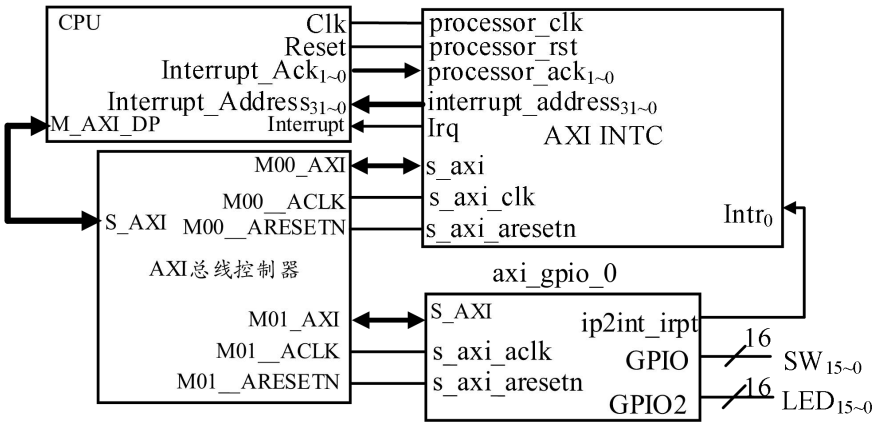

图 3 回顾 GPIO 中断

(4) GPIO 通道：

可知，控制 16 个独立的 LED 灯需要一个 16 位的 GPIO 通道，16 个独立开关需要一个 16 位的 GPIO 通道，2 个独立按键需要一个 2 位的 GPIO 通道，4 位七段数码管需要一个 4 位的 GPIO 通道和一个 8 位的 GPIO 通道。可得，该电路需要用到 5 个 GPIO 通道。

(5) 中断源：

由上述分析可知，该应用示例共 4 个中断源，其中又因为 T0、T1 由于共用一个定时器 IP

核，因此从 INTC 的角度来看，可视为有 3 个中断源。因此，如果采用快速中断方式，控制程序需编写 3 个中断服务程序，并向 INTC 的中断向量表填写这三个中断服务程序的中断向量。定时器的中断服务程序需进一步采用查询方式，判断是 T0 还是 T1 产生的中断，并调用相应的中断事务处理函数。

2. IO 接口电路设计：

由上述题目分析得，该题所需的硬件接口电路需利用到 GRIO IP 核 Timer IP 核以及中断控制器 IP 核。

简化 IO 接口电路硬件电路框图如图 4 所示：

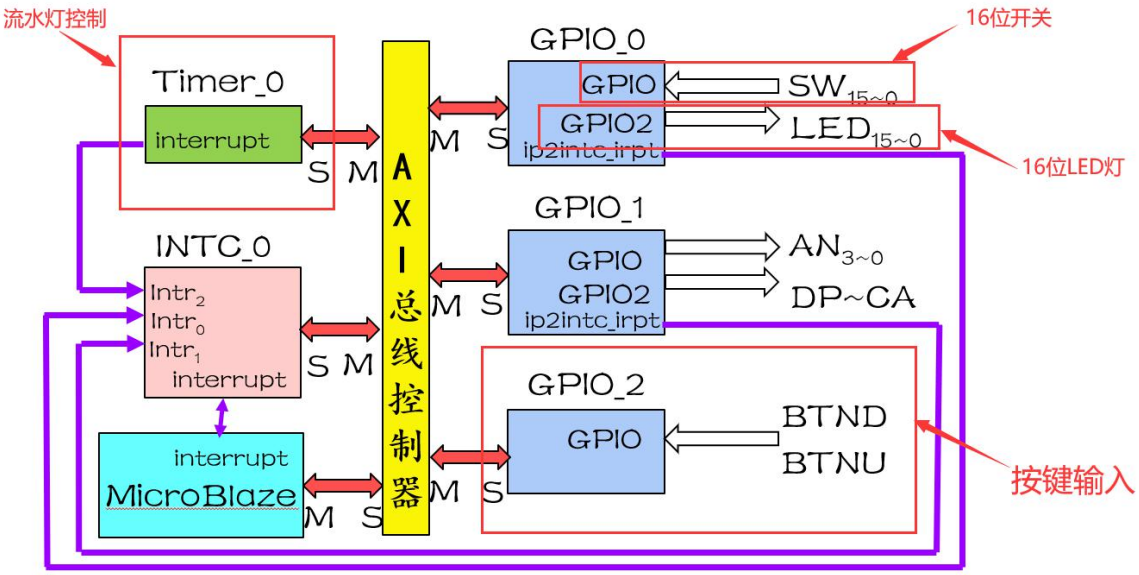

图 4 简化 IO 接口硬件电路图

由题目分析可知，共 4 个中断源，但可知， T0、T1 共用一个定时器 IP 核，因此如果从 INTC 的角度来看，可视为有只有 3 个中断源。

16 位开关和 16 位 LED 灯共用一个 GPIO IP 核（设为 GPIO_0），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_1），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）；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。

五. 实现过程

1. 宏定义：

根据图 1 电路，使用宏定义设定各个 IO 接口的 IP 核基地址、中断类型码与中断掩码，设定代码如图 5 所示（因为此次实验没有建立硬件平台，所以具体地址我用中文注释代替，方便以后建立硬件平台后添加与修改）：

```
#define XPAR_AXI_GPIO_0_BASEADDR GPIO_0基地址
#define XPAR_AXI_GPIO_1_BASEADDR GPIO_1基地址
#define XPAR_AXI_GPIO_2_BASEADDR GPIO_2基地址
#define XPAR_AXI_ TIMER_ 0_BASEADDR Timer_0基地址
#define XPAR_INTC_0_BASEADDR INTC基地址
#define XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK GPIO_0中断向量
#define XPAR_MICROBLAZE_0_AXI_INTC_AXI_GPIO_0_IP2INTC_IRPT_INTR 微处理器偏移
#define XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK GPIO_2中断向量
#define XPAR_MICROBLAZE_0_AXI_INTC_AXI_GPIO_2_IP2INTC_IRPT_INTR 微处理器偏移
#define XPAR_AXI_TIMER_0_INTERRUPT_MASK_Timer_0中断向量
#define XPAR_MICROBLAZE_0_AXI_INTC_AXI_TIMER_0_INTERRUPT_INTR 微处理器偏移
#define XPAR_INTC_0_GPIO_VEC_ID\
 XPAR_MICROBLAZE_0_AXT_INTC_AXI_GPIO_0_IP2INTC_IRPT_INTR
#define XPAR_INTC_0_GPIO_2_VEC_ID\
 XPAR_MICROBLAZE_0_AXI_INTC_AXI_GPIO_2_IP2INTC_IRPT_INTR
#define_XPAR_INTC_0_TMRCTR_0_VEC_ID\
 XPAR_MICROBLAZE_0_AXI_INTC_AXT_TIMER_0_INTERRUPT_INTR
```

图 5 宏定义代码

2. 普通中断方式：

中断控制器支持快速中断以及普通中断方式，我们首先分析普通中断方式。

根据题目分析多中断应用系统的功能需求，设计工作在普通中断方式下的控制程序框图，如图 6 所示：

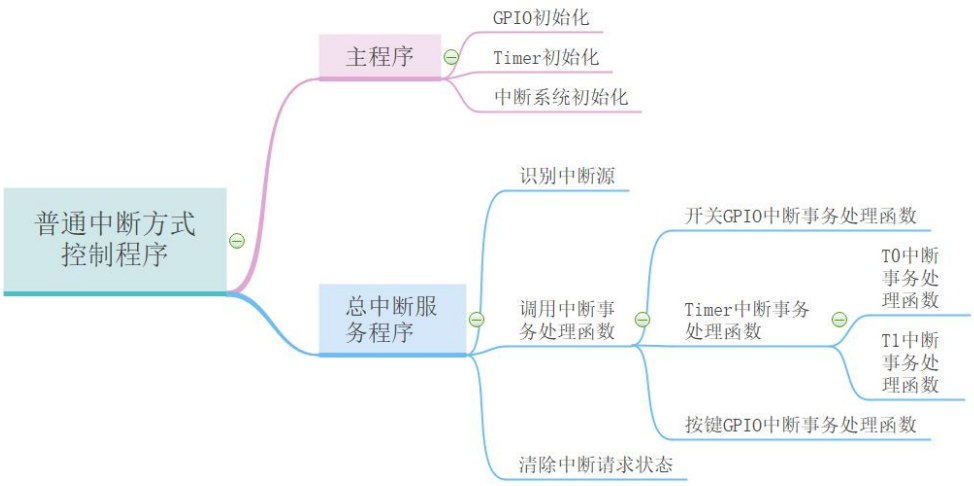

图 6 普通中断方式控制程序框图

由图可知，在主程序中实现启用中断系统、对 GPIO 各个通道的输入输出工作方式

进行设定并设定各个定时器的计时时间以及对中断系统进行初始化。启用中断系统后，中断服务程序能被微处理器硬件中断调用。总中断服务程序识别中断源、调用中断事务处理函数并清除中断控制器的中断请求状态，由开关对应的 GPIO 中断事务处理函数实现 16 位开关中断方式输入并更改四位七段数码管所显示的数字；由按键对应的 GPIO 中断事务处理函数实现两个独立按键中断方式输入并更改 T0 的计时时间；由 Timer 的中断事务处理函数调用两个中断事务处理函数，其中，T0 定时器的中断事务处理函数实现控制 LED 的点亮位置与循环速度， T1 定时器的中断事务处理函数实现四个七段数码管动态扫描。

综上，控制程序可以分为七个函数：①一个主函数、②一个总中断服务程序、③一个开关的中断事务处理函数、④一个按键的中断事务处理函数、⑤一个 Timer 的中断事务处理函数、⑥一个走马灯 T0 的中断事务处理函数、⑦一个动态扫描数码管的 T1 中断事务处理函数。

下面，再来分析上述各个函数应该实现的具体功能、操作流程和具体代码：

（1） 中断初始化程序：

具体功能：

中断初始化程序主要包括各类 IO 接口的初始化和中断系统的初始化。

其中：

① IO 接口的初始化包括三个 GPIO 各个通道输入输出工作方式的设定（通过写控制寄存器实现）；

② Timer 的初始化需初始化 Timer 中的两个定时器，可知他们的操作流程是一致：首先停止计时（通过写各自的 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（通过写各自的 TLR 寄存器实现），紧接着装载预置值（通过写各自的 TCSR 寄存器使装载这一位为 1 实现）然后再是写各自的 TCSR 寄存器，控制各个定时器使能。清除原中断状态、使能中断、使能自动装载、减计数；

③ 中断系统的初始化包含开放中断和注册中断服务程序两部分，其中 Timer 已经在初始化阶段开放中断，之后就是对开关连接的 GPIO IP 核的 GPIO 通道开中断：首先清除原中断状态，使能 GPIO 通道中断、使能中断输出、对按键连接的 GPIO IP 核的 GPIO 通道开中断。

④ 对中断控制器开中断：首先清除原中断状态，然后使能 intr0、intr1 和 intr2 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）；

⑤ 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；

综上，我们通过中断初始化程序完成了中断系统以及 IO 接口的初始化，根据上述分析我们可画出相关流程框图与具体代码。

流程框图：

根据具体功能要求,设计流程框图如图 7 所示,并与课件上给的参考流程图(如图 8 所示）进行比较：

可见，我们开 GPIO IP 核的中断与 Timer 的初始化的顺序与参考保持一致，我认为这是最佳流程。

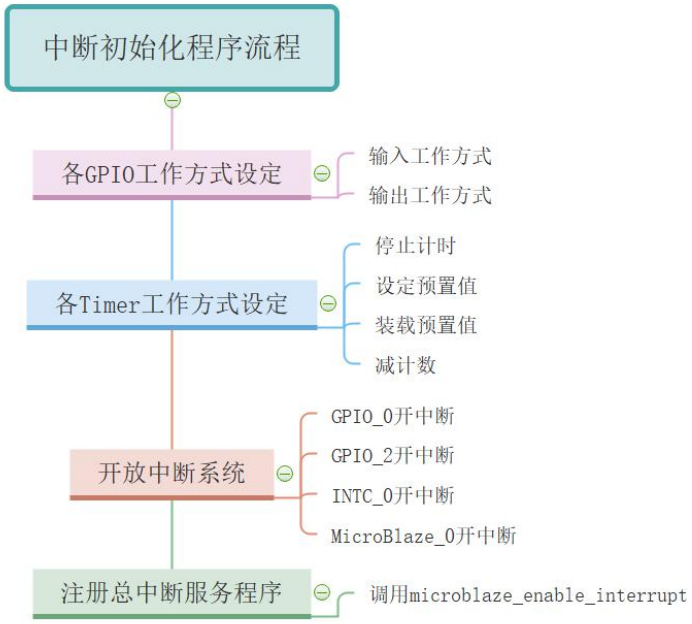

图 7 中断初始化程序流程图（普通中断）

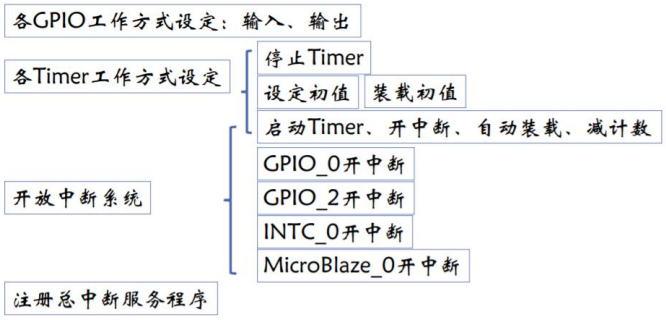

图 8 教材中参考初始化流程图（普通中断）

实现代码：

实现代码见图 9 与图 10 所示：

```
int main()
{
    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));
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,RESET_VALUE0);
    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_CSR_DOWN_COUNT_MASK|XTC_CSR_INT_OCCURED_MASK);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
        Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET)
                &~XTC_CSR_ENABLE_TMR_MASK);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TLR_OFFSET,RESET_VALUE1);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
        Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET)
                |XTC_CSR_LOAD_MASK);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET,
        (Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+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|XTC_CSR_INT_OCCURED_MASK);
```

图 9 中断初始化程序代码 1（普通中断）

```
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI_OFFSET,0xffff);
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_TRI2_OFFSET,0x0);
Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI_OFFSET,0x0);
Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_TRI2_OFFSET,0x0);
Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA2_OFFSET,0x1);
Xil_Out8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_TRI_OFFSET,0x1f);

Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_ISR_OFFSET,XGPIO_IR_CH1_MASK);
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);
Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_ISR_OFFSET,XGPIO_IR_CH1_MASK);
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_IER_OFFSET,XGPIO_IR_CH1_MASK);
Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_GIE_OFFSET,XGPIO_GIE_GINTR_ENABLE_MASK);

Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IAR_OFFSET,XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK
            |XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPAR_AXI_TIMER_0_INTERRUPT_MASK);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IER_OFFSET,XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK
            |XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPAR_AXI_TIMER_0_INTERRUPT_MASK);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK|
           XIN_INT_HARDWARE_ENABLE_MASK);
microblaze_enable_interrupts();
}
```

图 10　中断初始化程序代码 2（普通中断）

（2）　总中断服务程序：

具体功能：

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 D0 是否为 1，如果是 1 则表示拨动了开关，因此调用开关对应的 GPIO 中断事务处理函数。返回之后再判断 D1 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。再返回之后，判断 D2 是否为 1 如果是 1 则表示 Timer 中有定时器计时时间到，因此调用 Timer 对应的中断事务处理函数。

由于系统中没有其他的中断源，因此返回之后写中断控制器的 IAR 寄存器来清除中断请求状态之后就退出

流程框图：

根据具体功能要求，设计流程框图如图 11 所示：

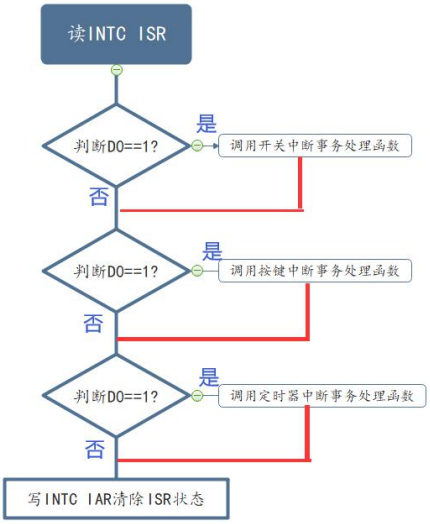

图 11 总中断服务程序流程图（普通中断）

实现代码：

实现代码如图 12 所示：

```
void My_ISR()
{
    int status;
    status=Xil_In32(XPAR_AXI_INTC_0_BASEADDR+XIN_ISR_OFFSET);
    if((status&XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)==XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK)
        switch_handle();
    if((status&XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK)==XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK)
        button_handle();
    if ((status&XPAR_AXI_TIMER_0_INTERRUPT_MASK)==XPAR_AXI_TIMER_0_INTERRUPT_MASK)
        timer_handle();
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IAR_OFFSET,status);
}
```

图 12 总中断服务程序代码（普通中断）

（3） 开关的中断事务处理函数：

具体功能：

开关的中断输入处理则是读取开关状态，更新数码管的显示缓冲区的段码，七段数码管显示开关对应的 16 进制数值。 另外由于 GPIO 内部也集成了一个简单的中断控制器，因此也须清除该中断控制器的中断请求状态，这通过读后写 IPISR 实现。再将 16 位开关表示的 2 进制数转换为四位七段数码管对应的 16 进制数字的段码（如图 13 所示）。

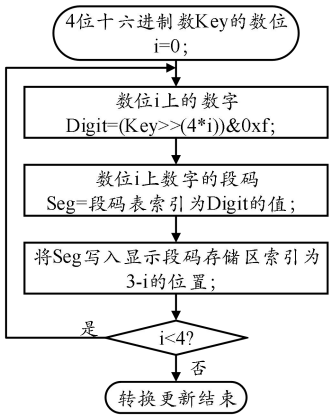

图 12 七段数码管显示电路流程

实现代码：

实现代码如图 13 所示：

```
void switch_handle()//开关的中断事务处理函数
{
    short hex=Xil_In16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA_OFFSET);
    int segcode_index=3;
    for(int digit_index=0;digit_index<4;digit_index++)//读取开关值、更新数码管显示缓冲区

    {
        segcode[segcode_index]=segtable[(hex>>(4*digit_index))&0xf];
        segcode_index--;
    }
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_ISR_OFFSET,
            Xil_In32(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_ISR_OFFSET));//读后写GPIO_0 IPISR清除IPISR状态

}
```

图 13 开关中断事务处理函数代码

（4） 按键中断事务处理函数：

具体功能：

按键控制走马灯的循环速度，通过修改定时器的预置值来实现。

首先读取按键的状态：无论是按下还是释放，按键都会产生中断。（即按一次按键 实际上会产生两次中断）2.250224 00:12:35.750 --> 00:12:38.470 因此需判断按键是否按下 225 00:12:39.990 --> 00:12:42.060 如果增速按键按下，则将定时器 T0 的预置值减少一固定值（通过读修改写 TLR0 寄存器的方式来实现：首先读取 TLR0 寄存器的值，然后减去某一固定值后再写入 TLR0 计算器），预置值变小，中断频率就会提高 ，从而实现增速。如果是减速按键按下，则将定时器 T0 的预置值增加一固定值，实现方式同理。最后对 GPIO 内部的中断控制逻辑清除中断请求状态。

流程框图：

如图 14 所示：

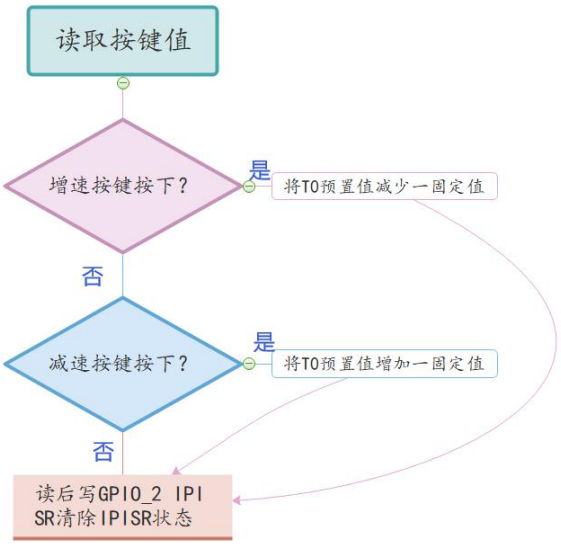

图 14 按键中断事务处理函数流程图

实现代码：

实现代码和注释如图 15 所示：

```
void button_handle()
{
    char button;
    button = Xil_In8(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_DATA_OFFSET)&0x1f;//首先读取按键的状态
    if(button==0x2)//如果增速按键按下
        Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,
                Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET)-STEP_PACE);
    //将定时器T0的预置值减少一固定值
    else if (button==0x10)//如果是减速按键按下
        Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET,
                Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TLR_OFFSET)+STEP_PACE);
    //将定时器T0的预置值增加一固定值
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_ISR_OFFSET,//对GPIO内部的中断控制逻辑清除中断请求状态
                Xil_In32(XPAR_AXI_GPIO_2_BASEADDR+XGPIO_ISR_OFFSET));
}
```

图 15 按键中断事务处理函数实现代码

(5) Timer 的中断事务处理函数

具体功能：

首先，读取各个定时器的控制状态寄存器来判断是哪个定时器产生了中断（因为 Timer IP 核包含两个定时器，但是共用一个中断请求输出），然后再调用相应的中断事务处理函数，并清除中断请求状态。

基本流程为：首先读取 TCSR0，然后判断 T0 是否产生了中断，如果产生了中断则调用 T0 的中断事务处理函数，返回之后清除 T0 的中断请求状态，之后再读取的 TCSR1，然后判断 T1 是否产生了中断。如果产生了中断 ， 则调用 T1 的中断事务处理函数并在返回之后清除 T1 的中断请求状态，最后退出。

流程框图：

如图 16 所示：

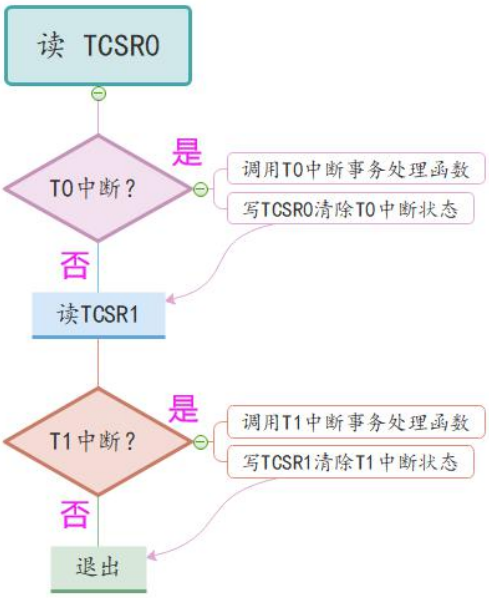

图 16 Timer 中断事务处理函数流程图

实现代码：

实现代码及注释见图 17 所示：

```
void timer_handle()
{
    int status;
    status=Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET);//首先读取TCSR0
    if((status&XTC_CSR_INT_OCCURED_MASK)==XTC_CSR_INT_OCCURED_MASK)//判断T0是否产生了中断
        {
            timer0_handle();//调用T0的中断事务处理函数
            Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET,status);
    }
    status=Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET);
    //清除T0的中断请求状态
    if((status&XTC_CSR_INT_OCCURED_MASK)==XTC_CSR_INT_OCCURED_MASK)//判断T1是否产生了中断
        {
            timer1_handle();//调用T1的中断事务处理函数
        Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TIMER_COUNTER_OFFSET+XTC_TCSR_OFFSET, status);
    // 返回之后清除T1的中断请求状态
    }
}
```

图 17 Timer 中断事务处理函数具体代码

（6） 定时器中断事务处理程序

具体功能：

定时器 T0 和 T1 的中断事务处理流程基本一致，都是控制 GPIO 输出。

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。

T1 控制七段数码管扫描通过 gpio_1 控制一位数码管点亮 （数码管动态扫描接口包括段码和位码的输出），并准备下一位的位置之后直接退出。

流程框图：

如图 18 所示：

图 18 定时器中断事务处理函数流程图（左 T0，右 T1）

实现代码：

实现代码及注释见图 19 所示：

```
void timer0_handle()
{
    Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA2_OFFSET,1<<ledbits);
    //写gpio_0的GPIO2通道点亮当前位置LED灯
    ledbits++;//将点亮位置修改为下一位

    if(ledbits==16)
        ledbits=0;
}

void timer1_handle()
{
    Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA2_OFFSET,segcode[pos]);
    Xil_Out16(XPAR_AXI_GPIO_1_BASEADDR+XGPIO_DATA_OFFSET,poscode[pos]);
    //写gpio_1的 GPIO\GPIO2输出当前位置数码管位码\段码
    pos++;//将扫描位置修改为下一位
    if(pos==4)
        pos=0;
}
```

图 18 定时器中断事务处理函数流程图

（7） 变量及函数声明

具体功能：

① 写了宏定义所在的头文件；
② 设定了 T0 与 T1 的预置值；
③ 设定了 T1 每次更改的步长；
④ 填写了七段数码管显示缓冲区以及位码数组；；
⑤ 设定了 LED 位置的全局变量。

实现代码：

实现代码见图 19 所示：

```
#include "xil_io.h"
#include "stdio.h"
#include "xintc_l.h"
#include "xtmrctr_l.h"
#include "xgpio_l.h"
#define RESET_VALUE0     100000000-2
#define RESET_VALUE1     100000-2
#define STEP_PACE 10000000
void My_ISR() __attribute__((interrupt_handler));
void switch_handle();
void button_handle();
void timer_handle();
void timer0_handle();
void timer1_handle();
char segtable[16]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x98,0x88,0x83,0xc6,0xa1,0x86,0x8e};
char segcode[4]={0xc0,0xc0,0xc0,0xc0};
short poscode[4]={0xf7,0xfb,0xfd,0xfe};
int ledbits=0;
int pos=0;
```

图 19 变量及函数声明

2. 快速中断方式：

使用快速中断方式，则无总中断服务程序，但多出了几个独立寄存器。它对其中的每一个中断源，都提供独立的中断服务程序。所以，相比上面的普通中断模式，我们只需修改中断初始化程序与函数声明部分。

（1） 中断初始化程序修改：

功能需求：

快速中断方式下的中断初始化程序与普通中断方式相同，但是要增加对中断控制器控制的代码：要设置 INTC 工作在快速中断模式，并且要填写 INTC IVAR 中断向量表。

实现代码：

相比普通中断方式，增加代码和相关注释如图 20 所示：

```
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IMR_OFFSET,XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK
|XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK|XPAR_AXI_TIMER_0_INTERRUPT_MASK);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IVAR_OFFSET+
          4*XPAR_AXI_INTC_0_AXI_GPIO_0_IP2INTC_IRPT_INTR,(int)switch_handle);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IVAR_OFFSET+
          4*XPAR_AXI_INTC_0_AXI_GPIO_2_IP2INTC_IRPT_INTR,(int)button_handle);
Xil_Out32(XPAR_AXI_INTC_0_BASEADDR+XIN_IVAR_OFFSET+
          4*XPAR_AXI_INTC_0_AXI_TIMER_0_INTERRUPT_INTR,(int)timer_handle);
```

设置快速中断模式

填写中断向量表

图 20 快速中断模式初始化程序增加代码

（2） 函数声明修改：

功能需求：

由于开关、按键以及 Timer 的主断事务处理函数不再是任何程序的子函数，. 因此需将它们的属性定义为 fast_interrupt。

修改代码：

相比普通中断方式，修改代码和相关注释如图 21 所示：

```
#include "xil_io.h"
#include "stdio.h"
#include "xintc_l.h"
#include "xtmrctr_l.h"
#include "xgpio_l.h"
#define RESET_VALUE0      100000000-2
#define RESET_VALUE1      100000-2
#define STEP_PACE 10000000
//void My_ISR() __attribute__((interrupt handler));
void switch_handle() __attribute__ ((fast_interrupt));
void button_handle() __attribute__ ((fast_interrupt));
void timer_handle() __attribute__ (fast_interrupt));
void timer0_handle();
void timer1_handle();
char segtable[16]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x8
char segcode[4]={0xc0,0xc0,0xc0,0xc0};
short poscode[4]={0xf7,0xfb,0xfd,0xfe};
int ledbits=0;
int pos=0;
```


图 20 函数声明修改代码

3. 完全程序控制方式

GPIO IP 核有两个通道，且每个通道都可以达到 32 位，各个引脚可分别独立控制在输人或输出工作状态，因此 16 位独立开关以及 16 位 LED 灯可以通过 GPIO IP 核同一个通道. 不同 I/O 引脚控制，也可以通过两个不同通道单独控制。

4 位七段数码管共 4 位位选信号以及 8 位段选信号，共 12 位输出，可采用 GPIO IP 核. 同一个通道控制，也可以通过两个不同通道单独控制位选和段选信号。若采用不同 GPIO 通道分别控制 16 位独立开关输人、16 位独立 LED 输出、4 位七段数码管位选以及 8 位七段数码管段选，共需两个 GPIO IP 核，接口电路如图 21 所示：

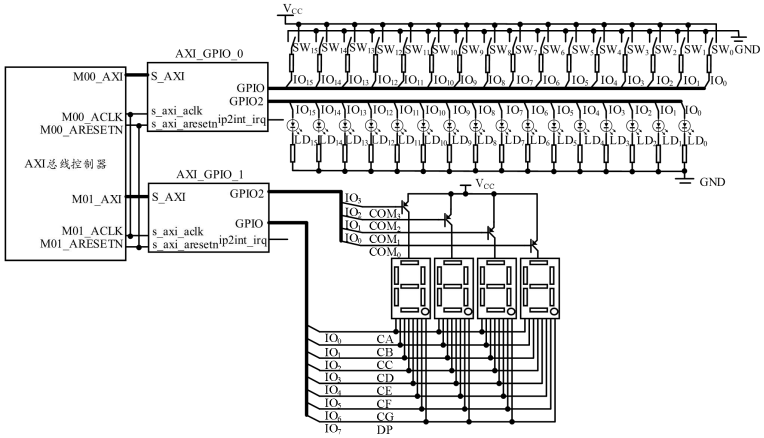

图 21 GPIO 简单并行数字外设接口电路

分析可知：AXI_ GPIO_ 0 每个通道 16 位,通道 GPIO 输人、通道 GPIO2 输出；AXI_ GPIO_ 2 两个通道都为输出,通道 GPIO 宽度为 8 位、通道 GPIO2 宽度为 4 位。

由题目分析，设计整个控制程序的流程如图 22 所示：

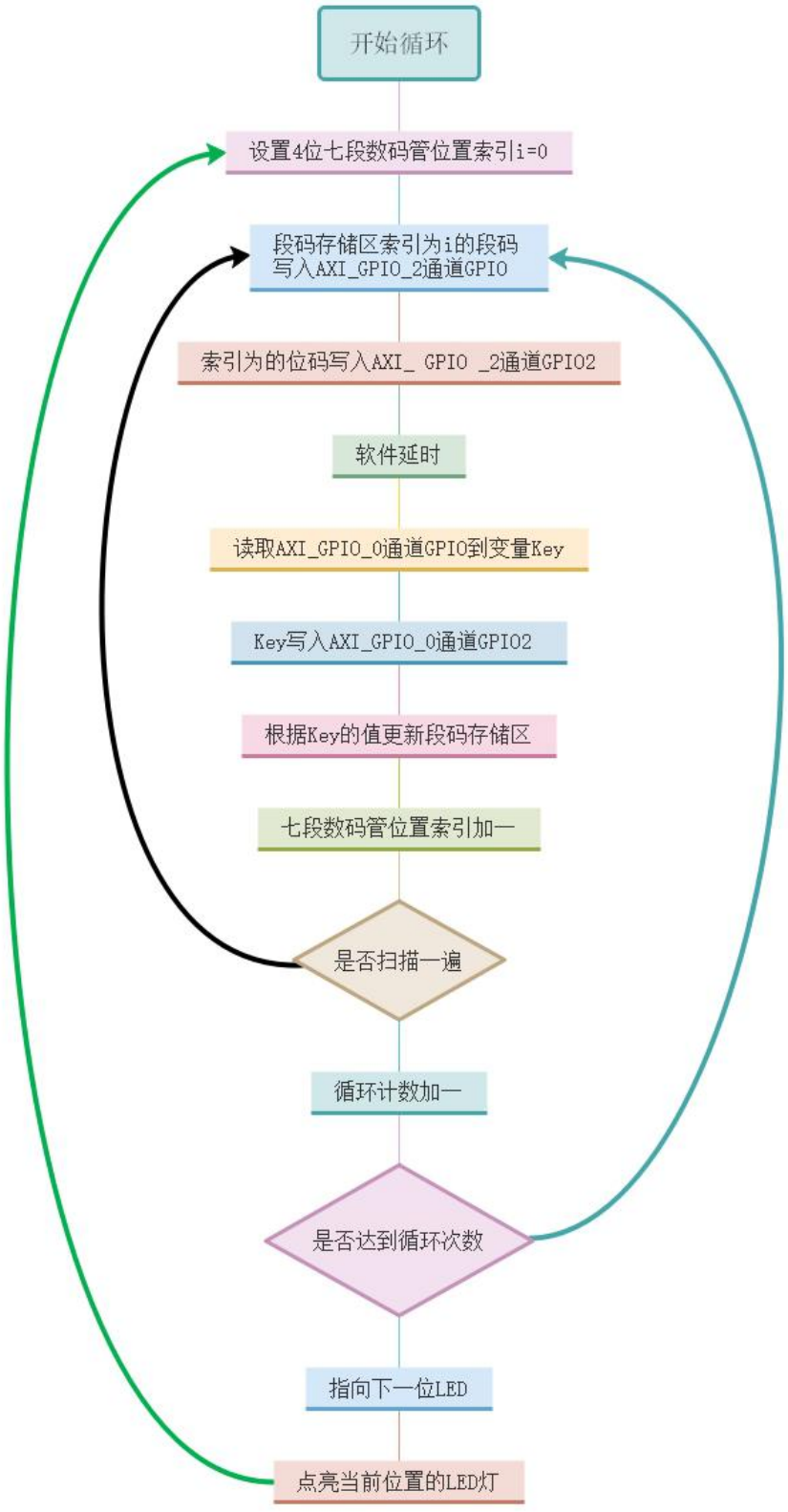

图 22 程序控制方式流程框图

与课件参考相对比（见图 23 所示），两者部分顺序不同，但执行效率相似：

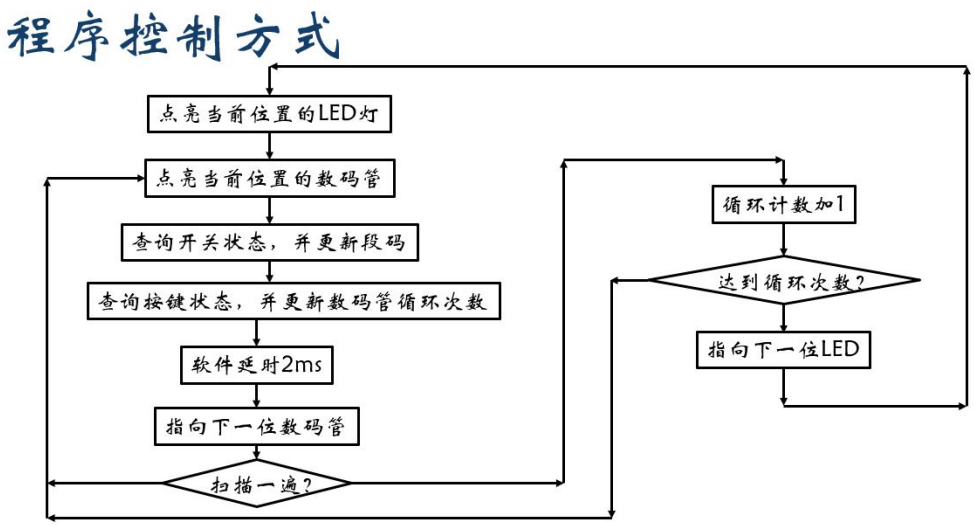

图 23 课件参考程序控制方式流程框图

程序控制方式程序代码：

控制程序代码和相关注释如图 24 所示：

```
int main()
{
    char segtable[16]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,\
    0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e,};//七段数码管十六进制数字段码表
    char segcode[4]={0xc0,0xc0,0xc0,0xc0};//显示段码存储区初始化为0
    short poscode[4]={0xf7,0xfb,0xfd,0xfe};//4位七段数码管位码
    Xil_Out8(XPAR_GPIO_2_BASEADDR+ XGPIO_TRI_OFFSET,0x0);
    Xil_Out8(XPAR_GPIO_2_BASEADDR+ XGPIO_TRI2_OFFSET,0x0);
    Xil_Out16(XPAR_GPIO_0_BASEADDR+ XGPIO_TRI_OFFSET,0xffff);
    Xil_Out16(XPAR_GPIO_0_BASEADDR+ XGPIO_TRI2_OFFSET,0x0);

    while(1){
        for (int i=0;i<4;i++)
        {
            Xil_Out8 XPAR_GPIO_2_BASEADDR+ XGPIO_DATA_OFFSET segcode i
            Xil_Out8 XPAR_GPIO_2_BASEADDR+ XGPIO_DATA2_OFFSET poscode i
            for (int j=0;j<10000;j++){

            }//延时
            shortKey=Xil_In16 XPAR_GP)IO_0_BASEADDR+ XGPIO_DATA_OFFSET
            Xil_Out16 XPAR_GPIO_0_BASEADDR+ XGPIO_DATA2_OFFSET Key
            for (int digit_index=0;digit_index<4 ; digit_index++)// 更新显示段码存储区
            {
                segcode[3-digit_index] =segtable[(Key>>(4*digit_index))&0xf];
            }
        }
    }

    return 0;
}
```

图 24 控制程序代码

六.方式比较

<table>
<tr><th>程序控制方式</th><th>普通中断方式</th><th>快速中断方式</th></tr>
<tr><td rowspan="2">• 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
•</td><td colspan="2">• 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.</td></tr>
<tr><td>• 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</td><td>• 多出了独立寄存器。避免通用寄存器压栈所耗费的时间
• 具有硬件向量表
• 根据中断向量直接转入，不存在函数调用
• 需尽快处理完中断请求并离开</td></tr>
</table>

七.问题分析

1. **使用按键调整循环速度时，可能会出现按键按下后无反应，或者需要长时间一直按着按键才能正常的运行。**
   解：①可能是因为程序中没有设置对按键按下或弹起这两种状态改变都做出反应，而仅仅设置了对其中一种状态改变做出反应；
   ②可能在程序中，错误地设置了下降沿上升沿触发与高电平低电平触发。

2. **可能会出现进行了一次操作后，程序一直进入中断的现象。比如说，按下一次加速键后，一直进行循环加速操作。**
   解：可能在中断服务程序的结尾，没有对中断状态进行清除，使得每一次检测时都处在中断状态，从而一直调用中断服务程序。

3. **可能会出现做出某个动作后，进入“死机”状态，很长一段时间对其它的动作无反应。**
   答：可能是中断服务程序中出现了死循环，导致程序无法返回，则对后面的中断无法做出反应。

4. **可能会出现未进行任何操作而自己进入中断。**
   答：应该是中断初始化程序中的问题，要注意应该先使能，再清除。可能是没有对中断状态进行清除，使得自动进入中断。

5. **可能出现系统对我们的操作反应缓慢，比如说按下按键后，肉眼可见的延迟后才做出改变。**
   答：可能是中断里面做了太多工作，导致主函数丢失任务。我们需要记住，进入中断后

需要尽快离开中断状态。

八. 实验总结

这次实验是微机原理实验中耗时最长，实验要求最高、实现难度最高的一次实验。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，也对独立自主学习与独立完成任务有了进一步的体会，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.**

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。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. **自主学习.**

这学期因为疫情的原因，我们不能像以往一样在学校里统一的课堂上学习，学习资源上也受到这样那样的限制,所以这种情况就更加考验我们的自主学习和独立完成任务的能力。在这次实验中，因为是新的知识，所以不仅需要我们理解了知识点，更需要我们对内容反复的学习与渗透，做到对中断相关的内容熟练的掌握与应用，还要能比较与区分各种不同实现方法的优劣与适用范围,权衡各个方面后结合起来使用。

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，在当下疫情的情况中帮助我调整心态，树立目标。

总之，这是一次记忆深刻、满载而归的实验经历，我从中学到了很多！