上海交通大學

《工程实践与科技创新Ⅱ-A》课程

学生实验报告

实验名称：I2C GPIO 扩展及 SYSTICK 中断实验

姓　　名：彭俊杰

学　　号：521021910404

班　　级：ICE0501-5

任课老师：朱兰娟、邵群

2023 年 5 月 28 日

目录

1 实验目的

（1）了解 I2C 总线标准及在 TM4C1294 芯片的调用方法；
（2）掌握用 I2C 总线扩展 GPIO 芯片 PCA9557 及 TCA6424 的方法，能够通过扩展 GPIO 来输出点亮 LED 及动态数码管；
（3）进一步熟悉 SYSTICK 定时中断的使用，掌握利用软定时器模拟多任务切换的方法。

2 实验原理

2.1 硬件原理分析

2.1.1 IIC 串行总线（Inter-Integrated Circuit）

IIC（或 I2C、I2C）总线接口是 Philips 推出的一种简单的双向两线制串行总线方式，用于 IC 器件之间的通信。主要特点有以下 3 点：I2C 总线仅使用双向串行数据线 SDA 和串行时钟线 SCL 两个信号，数据线上的信号与时钟同步；多个符合 I2C 总线标准的器件都可以通过同一条 I2C 总线进行通信，器件地址采用硬件设置方法，扩展简单灵活；通过软件寻址识别每个器件，不需要额外的地址译码器。

TM4C1294NCPDT I2C 模块如下图所示。两组 8 个 FIFO 分别用于接收和发送数据，I2C I/O Select 可设置主机或从机，一般作主机用。

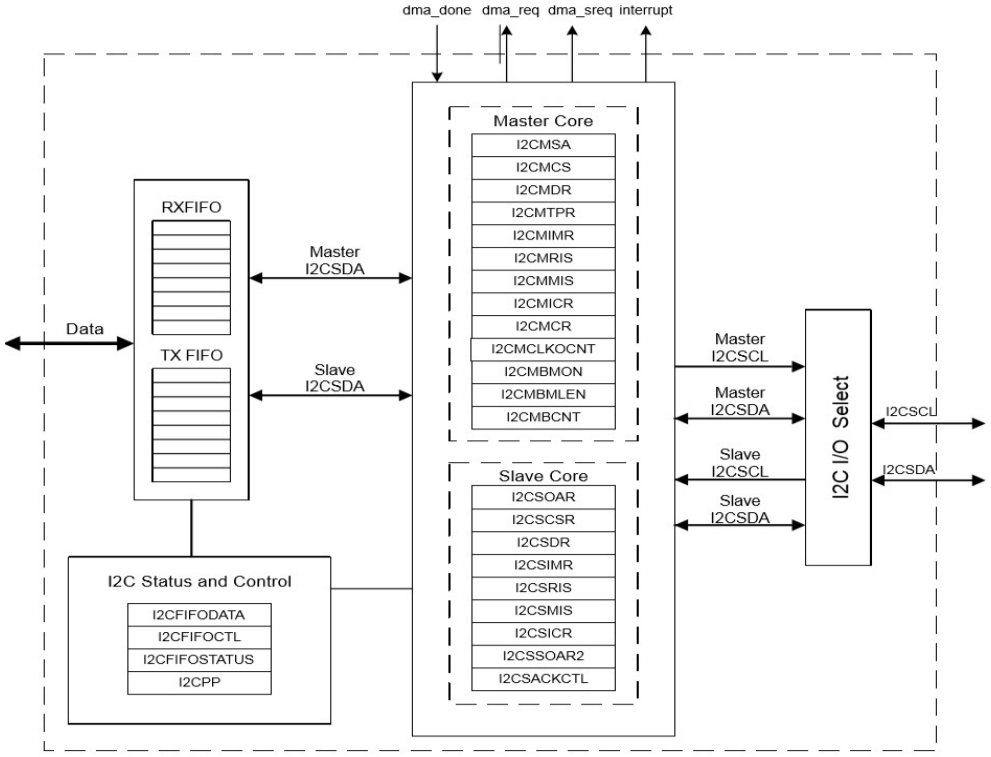
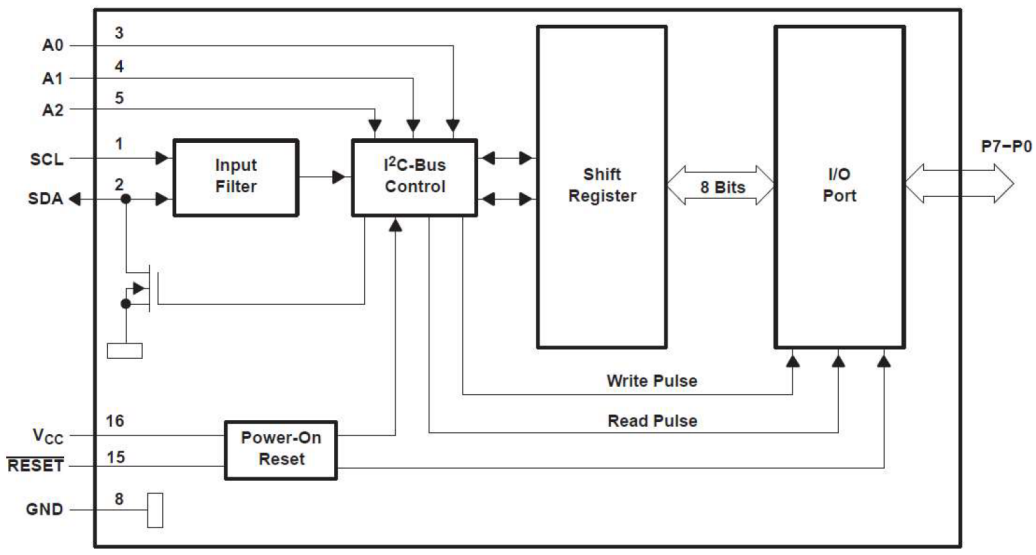


图 1 TM4C1294NCPDT I2C 模块图

S800 实验板上的 I2C 资源包括：I2C 转 8 位 I/O 扩展芯片 PCA9557PWR，以及 I2C 转 24 位 I/O 扩展芯片 TCA6424

2.1.2 I2C 转 I/O 扩展芯片 PCA9557PWR

PCA9557PWR 功能框图如下



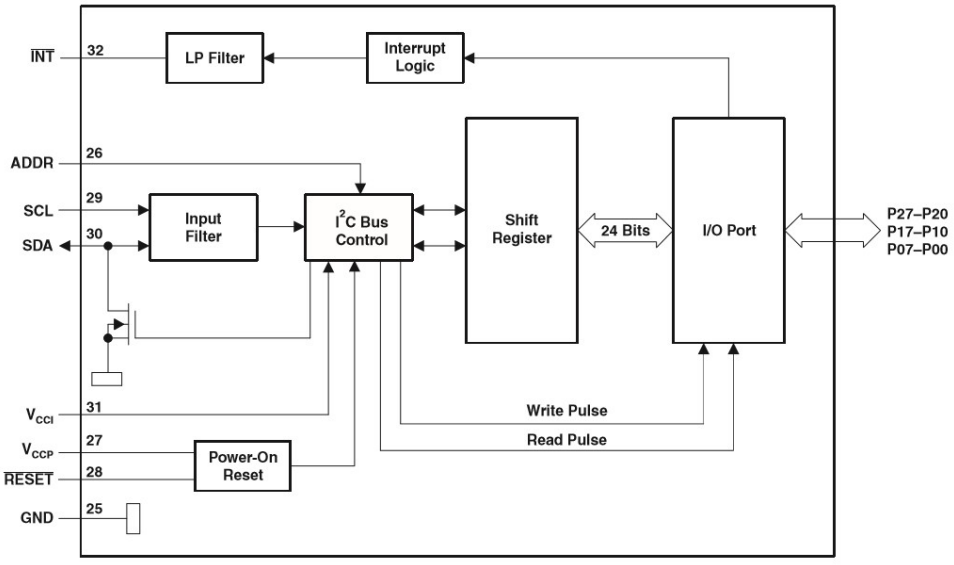
A. Pin numbers shown are for the D, DB, DGV, PW, and RGY packages.
B. All I/Os are set to inputs at reset.

图 2 PCA9557PWR 功能框图

从机地址格式： #define PCA9557_I2CADDR 0x18

2.1.3 I2C 转 I/O 扩展芯片 TCA6424

TCA6424 功能框图如下：



A. All I/Os are set to inputs at reset.
B. Pin numbers shown are for the RGJ package.

图 3 TCA6424 功能框图

从机地址格式： #define PCA6424_I2CADDR 0x22

2.2 软件算法分析（含程序流程图和源代码）

2.2.1 程序流程图

以实验例程 3 为例，程序的流程图如下：

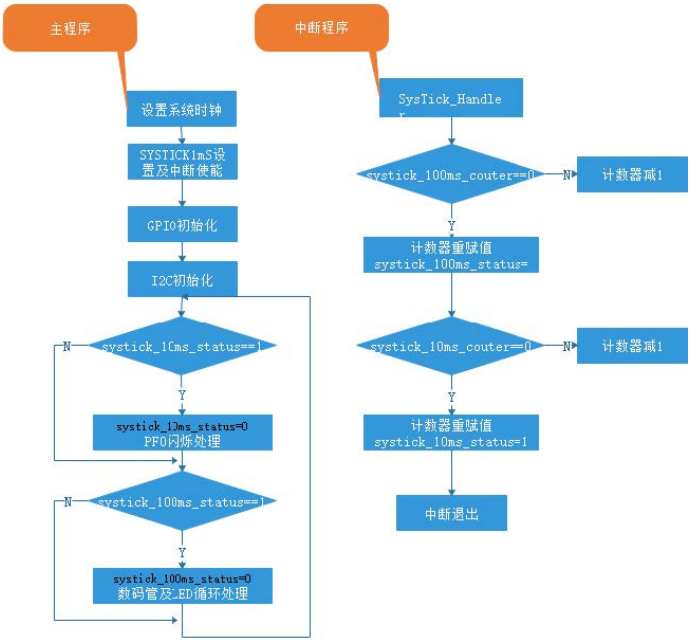


图 4 实验例程 3 程序流程图

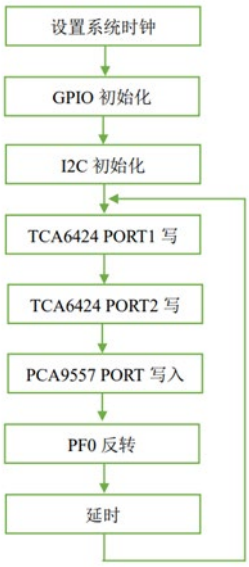


图 5 基本功能配置流程图

在本次实验中，主要是是通过 I2C 模式来实现 7 段数码管和 LED 的控制，具体过程如下所示。图 4 加入了 sistick 时间控制，图 5 为基本的功能配置。

2.2.2 部分源代码分析

```
1.  void S800_I2C0_Init(void)
2.  {
3.      uint8_t result;
4.
5.    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
6.    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
7.      GPIOPinConfigure(GPIO_PB2_I2C0SCL);
8.    GPIOPinConfigure(GPIO_PB3_I2C0SDA);
9.    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
10.   GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
11.
12.     I2CMasterInitExpClk(I2C0_BASE,ui32SysClock, true);
                              //config I2C0 400k
```

```
13.     I2CMasterEnable(I2C0_BASE);
14.
15.     result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_CONFIG_PORT0,0x0f
    f);         //config port 0 as input
16.     result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_CONFIG_PORT1,0x0)
    ;           //config port 1 as output
17.     result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_CONFIG_PORT2,0x0)
    ;           //config port 2 as output
18.
19.     result = I2C0_WriteByte(PCA9557_I2CADDR,PCA9557_CONFIG,0x00);
                    //config port as output
20.     result = I2C0_WriteByte(PCA9557_I2CADDR,PCA9557_OUTPUT,0x0ff);
                //turn off the LED1-8
21.
22. }
```

S800_I2CO_Init 是 I2C 模式的初始化过程，主要完成以下四个功能：

（1）使能 I2C 模块；

（2）使能提供 I2C 接口信号的 GPIO 端口；

（3）将 GPIO 引脚设置为 I2C 复用功能；

（4）配置并使能 I2C 主模式。

2.2.3 I2C 读写过程

读写过程流程图如图 6、图 7

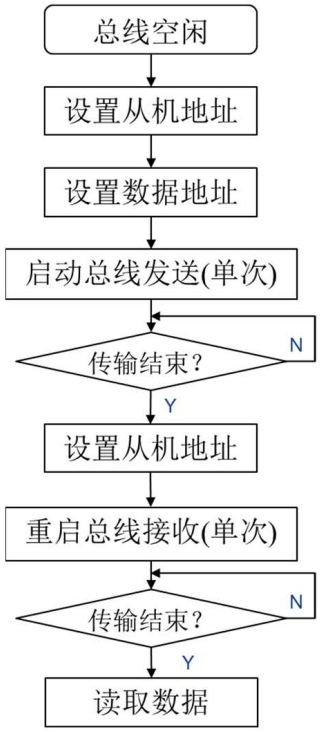


图 6　I2C 读过程流程图

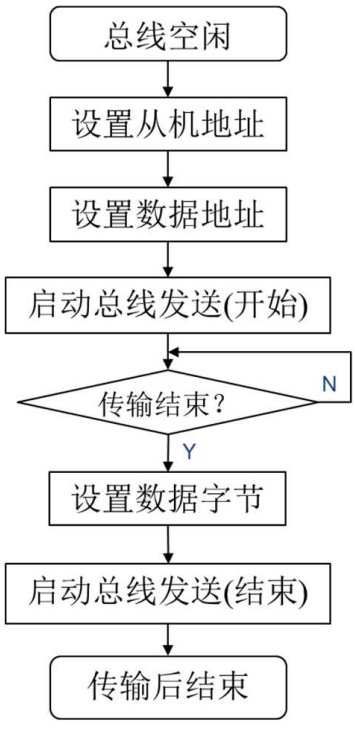


图 7　I2C 写过程流程图

I2C 读过程：

```
uint8_t I2C0_ReadByte(uint8_t DevAddr, uint8_t RegAddr) //读操作函数
{
uint8_t value, rop;
while(I2CMasterBusy(I2C0_BASE)){}; //遇忙等待
I2CMasterSlaveAddrSet(I2C0_BASE, DevAddr, false); //设置从机地址，写
I2CMasterDataPut(I2C0_BASE, RegAddr); //设置数据地址
//启动”总线发送”
I2CMasterControl(I2C0_BASE,I2C_MASTER_CMD_SINGLE_SEND);
while(I2CMasterBusBusy(I2C0_BASE)) {};
rop = (uint8_t)I2CMasterErr(I2C0_BASE);
I2CMasterSlaveAddrSet(I2C0_BASE, DevAddr, true); //设置从机地址，读
//重启“总线接收”
I2CMasterControl(I2C0_BASE,I2C_MASTER_CMD_SINGLE_RECEIVE);
while(I2CMasterBusBusy(I2C0_BASE)) {};
value = I2CMasterDataGet(I2C0_BASE); //读取数据
return value;
}
```

I2C 写过程：

```
uint8_t I2C0_WriteByte(uint8_t DevAddr, uint8_t RegAddr, uint8_t Writ
eData) //写操作函数
{
uint8_t rop;
while (I2CMasterBusy(I2C0_BASE)){}; //遇忙等待
I2CMasterSlaveAddrSet(I2C0_BASE, DevAddr, false); //设置从机地址，写
I2CMasterDataPut(I2C0_BASE, RegAddr); //设置数据地址（命令字节）
//启动”总线发送开始”
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
while(I2CMasterBusy(I2C0_BASE)){}; //等待传输结束
rop = (uint8_t)I2CMasterErr(I2C0_BASE); //检测错误，0-无错
I2CMasterDataPut(I2C0_BASE, WriteData); //设置数据字节
//启动”总线发送后结束”
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
while(I2CMasterBusy(I2C0_BASE)){};
rop = (uint8_t)I2CMasterErr(I2C0_BASE); //检测错误，0-无错
return rop;
}
```

3 实验内容和步骤

3.1 实验要求 1

3.1.1 任务要求

编程实现在数码管上同时显示 8 个字符，如稳定地显示日期，如“20210601”。

3.1.2 任务实现

由于数码管一次只能亮一位，因此要同时显示八个字符只能利用人眼的视觉暂留效应。容易想到分别对数码管八个位进行使能与输入字模，再进行延时，使每个数码管亮 1ms，这样视觉上每一位都不会闪烁，亮度也不是很低。

以控制第一个数码管为例，第 4 行代码的功能是防止拖影，第 5 行代码设置码段值，第 6 行代码配置对应的数码管口。

代码如下（能稳定显示 20230522）：

```
while (1)
    {
        //点数码管：20230522
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(0));
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, seg7[2]);              //write port 1（‘2’的码值）
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(1));     //write port 2（点亮第 1 个数码管，高点亮）
        SysCtlDelay(ui32SysClock/3000);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(0));
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, seg7[0]);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(2));
        SysCtlDelay(ui32SysClock/3000);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(0));
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, seg7[2]);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(4));
        SysCtlDelay(ui32SysClock/3000);
```

```
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(0));
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, seg7[3]);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(8));
        SysCtlDelay(ui32SysClock/3000);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(0));
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, seg7[0]);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(16));
        SysCtlDelay(ui32SysClock/3000);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(0));
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, seg7[5]);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(32));
        SysCtlDelay(ui32SysClock/3000);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(0));
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, seg7[2]);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(64));
        SysCtlDelay(ui32SysClock/3000);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(0));
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1, seg7[2]);
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2, (uint8_t)(128));
        SysCtlDelay(ui32SysClock/3000);

    }
```

3.2 实验要求 2

3.2.1 任务要求

修改例程 2-3，实现 2 位跑马。如：当显示为 1 时，跑马灯点亮 LED8，LED1，当显示为 2 时，跑马灯点亮 LED1，LED2，如此循环。

3.2.2 任务实现

由于跑马灯需要一个频率，那不可避免会用到 systick 定时中断功能。如图设置好初始化和中断服务程序，引出 100ms 和 10ms 两个定时。10ms 作为 led_M0 的闪烁半周期，100ms 用来设置跑马灯频率。

代码如下：

```
1.  void S800_SysTick_Init(void)
2.  {
3.      SysTickPeriodSet(ui32SysClock/SYSTICK_FREQUENCY); //定时 1ms
4.      SysTickEnable();
5.      SysTickIntEnable();
6.  }
7.
8.  /*
9.      Corresponding to the startup_TM4C129.s vector table systick inter
    rupt program name
10. */
11. void SysTick_Handler(void)
12. {
13.     if (systick_100ms_couter == 0) //利用 1ms 的 SysTick 产生 100ms 的定时
    器
14.     {
15.         systick_100ms_couter = 100;
16.         systick_100ms_status = 1;
17.     }
18.     else
19.         systick_100ms_couter--;
20.
21.     if (systick_10ms_couter == 0) //利用 1ms 的 SysTick 产生 10ms 的定时器
22.     {
23.         systick_10ms_couter  = 10;
24.         systick_10ms_status  = 1;
25.     }
26.     else
27.         systick_10ms_couter--;
28. }
```

在下述代码 main 函数 while 循环内，有着和实验任务 1 中 systick 定时相同的操作，实现了每 10ms 和 100ms 进入一次对应的 if 语句。控制对应的数码管口和 LED 灯可通过对当前 cnt 的值进行移位、按位取反等操作。代码 25、26 两行实现数码管两位跑马。另外，由于显示 1 的时候 led 需要亮 LED8 和 LED1，有必要做分类讨论。

代码如下：

```
1.  while (1)
2.      {
3.          if (systick_10ms_status) //10ms 定时到
4.          {
5.              systick_10ms_status = 0; //重置 10ms 定时状态
6.
7.              if (++gpio_flash_cnt >= GPIO_FLASHTIME/10)   //5*10ms=50m
    s
8.              {
9.                  gpio_flash_cnt = 0;
10.
11.                 //PF0 灯反转
12.                 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, ~GPIOPinRea
    d(GPIO_PORTF_BASE, GPIO_PIN_0 ));
13.             }
14.         }
15.
16.         if (systick_100ms_status) //100ms 定时到
17.         {
18.             systick_100ms_status = 0; //重置 100ms 定时状态
19.
20.             if (++i2c_flash_cnt >= I2C_FLASHTIME/100)  //5*100ms=500m
    s
21.             {
22.                 i2c_flash_cnt = 0;
23.
24.                 //数码管跑马灯
25.                 result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPU
    T_PORT1,seg7[cnt+1]);  //write port 1
26.                 result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPU
    T_PORT2,1<<cnt);        //write port 2
27.
28.                 //LED 跑马灯
29.                 if (cnt == 7){
30.                     //点亮 LED8、LED1
31.                     result = I2C0_WriteByte(PCA9557_I2CADDR,PCA9557_O
    UTPUT,~(129));
32.                 }
33.                 else{
```

```
                    //其它两个 LED 相邻的情况
                    result = I2C0_WriteByte(PCA9557_I2CADDR,PCA9557_OUTPUT,~(3<<cnt));
                }
                cnt = (cnt+1) % 8;
            }
        }
    }
```

3.3 实验要求 3

3.3.1 任务要求

修改例程 2-3，在要求 2 的基础上，实现同时 2 位 LED 跑马和 2 位数码管移动显示。如

第 1 步　数码管第 1，2 码位显示 1，2
　　跑马灯显示 LED1,2
第 2 步　数码管第 2，3 码位显示 2，3
　　跑马灯显示 LED2，3
……
第 8 步　数码管第 8，1 码位显示 8，1
　　跑马灯显示 LED8，1
第 9 步　回到第 1 步

3.3.2 任务实现

在实验要求 2 的基础上，针对数码管一次只能亮一位的特性，让两个数码管轮流亮，利用人眼的视觉暂留效应即可实现同时亮两位数码管。

在 while 循环中设置 bool 变量 flag 代替全局变量 cnt 对数码管的计数功能，每次进入 while 循环改变 flag 的 bool 值，根据不同的 bool 值区别点亮前一个还是后一个数码管，从而实现让两个数码管轮流亮。cnt＝7 时点亮的是第一个和最后一个数码管，特殊情况分类讨论即可。其他代码与实验要求 2 类似。

代码如下：

```
while (1)
    {

        flag = !flag;

        if (systick_10ms_status) //10ms 定时到
        {
            systick_10ms_status = 0; //重置 10ms 定时状态

            if (++gpio_flash_cnt >= GPIO_FLASHTIME/10)  //5*10ms=50ms
            {
```

```
                gpio_flash_cnt = 0;

                //PF0 灯反转
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, ~GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0 ));
            }
        }

        if (systick_100ms_status) //100ms 定时到
        {
            systick_100ms_status = 0; //重置 100ms 定时状态

            if (++i2c_flash_cnt >= I2C_FLASHTIME/100)  //5*100ms=500ms
            {
                i2c_flash_cnt = 0;
                cnt = (cnt+1) % 8;
            }
        }

        //数码管跑马灯
        //两数码管轮流亮
        if (flag){
            //点亮前一个数码管
            result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0);                    //write port 2
            result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,seg7[cnt+1]);  //write port 1
            result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,1<<cnt);             //write port 2
        }

        else{
            //点亮后一个数码管
            if (cnt == 7){
                //此时后一个数码管回到第一位数码管的位置，显示 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0);                    //write port 2
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,seg7[1]);      //write port 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,1);                    //write port 2
            }
            else{
```

```
                //后一个数码管数字比前一个数码管的大 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0);                    //write port 2
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,seg7[cnt+2]);  //write port 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,2<<cnt);             //write port 2
            }
        }

        //LED 跑马灯
        if (cnt == 7){
            //点亮 LED8、LED1
            result = I2C0_WriteByte(PCA9557_I2CADDR,PCA9557_OUTPUT,~(129));
        }
        else{
            //其它两个 LED 相邻的情况
            result = I2C0_WriteByte(PCA9557_I2CADDR,PCA9557_OUTPUT,~(3<<cnt));
        }
    }
```

3.4 实验要求 4

3.4.1 任务要求

修改例程 2-3，在要求 3 的基础上，当蓝板上按键 SW1 按下时，停止跑马灯，但 LED 及数码管显示维持不变；当按键松开后，继续跑马灯。

3.4.2 任务实现

利用蓝板按键暂停跑马灯，这个操作不需要对数码管显示模块做任何改动，这是因为在本代码结构中，数码管显示模块和决定显示什么字型的模块是独立的。因此，只需要在 while 循环中改控制 cnt 的语句即可。第 1 行读取了按键状态，在 3 行的 if 语句块中，判断使得在没有按下按键的时候，cnt 正常运行，在按下按键时 cnt 值不变，从而实现了暂停功能。

代码如下：

```
key = ~I2C0_ReadByte(TCA6424_I2CADDR,TCA6424_INPUT_PORT0);  //读入按键信息

        if (key & 0x01) //判断蓝板上按键 SW1 按下
        {
```

```
            cnt = cnt;  //维持 LED 及数码管显示
        }

        else{
            if (systick_100ms_status) //100ms 定时到
            {
                systick_100ms_status    = 0; //重置 100ms 定时状态

                if (++i2c_flash_cnt     >= I2C_FLASHTIME/100)  //5*10
0ms=500ms
                {
                    i2c_flash_cnt               = 0;
                    cnt = (cnt+1) % 8;
                }
            }
        }
```

3.5 实验要求 5

3.5.1 任务要求

修改例程 2-3，在要求 3 的基础上，用 USR_SW1 控制数码管和 LED 跑马灯的频率，
按第 1 下，间隔为 1s
按第 2 下，间隔为 2s
按第 3 下，间隔为 0.2s
按第 4 下，回到上电初始状态，间隔 0.5s
以 4 为模，循环往复

3.5.2 任务实现

i 用于记录按键次数。key = GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)，读 USR_SW1 信息。由于是四次按键为一个周期，i 余 4。i 初始设为 0。switch 分支设置 time，用于设置不同周期。

代码如下：

```
        //i:按键次数%4；time:周期
         i = i % 4;
         switch (i){
            case 0:
                time = 500; //0.5s
                break;
            case 1:
                time = 1000;    //1s
```

```
                break;
            case 2:
                time = 2000;    //2s
                break;
            case 3:
                time = 200; //0.2s
                break;
        }

        if (systick_100ms_status) //100ms 定时到
        {
            systick_100ms_status    = 0; //重置 100ms 定时状态

            key = GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0); //读
    USR_SW1 信息
            if (key == 0){
                i++;
            }

            if (++i2c_flash_cnt     >= time/100)  //5*100ms=500ms
            {
                i2c_flash_cnt               = 0;
                cnt = (cnt+1) % 8;
            }
        }
```

程序有两个值得优化的地方，一是采用 GPIO 中断方法检测按键按下，二是在检测按键按下时加入消抖程序。但实验过程中发现加入中断方法和消抖程序后数码管显示效果差，因此暂未采用中断方法和消抖程序，等待后续实验进一步探究。

3.6 实验要求 6

3.6.1 任务要求

编程在数码管上实现时钟功能，在数码管上最左端显示分钟+秒数。其中分钟及秒数均为 2 位数字，形如“12-00”，共 5 位。每隔一秒，自动加 1，当秒数到 60 时，自动分钟加 1，秒数回到 00，分钟及秒数显示范围 00~59。

当按下 USR_SW1 时，秒数自动加 1

当按下 USR_SW2 时，分钟自动加 1

当按下以上一个或两个按键不松开时，对应的显示跳变数每隔 200ms 自动加 1。即如下按下 USR_SW1 1s，则显示跳变秒数加 5。

3.6.2 任务实现

为了实现过 1 秒加一的功能，在 systick 中断服务程序中引出 1s 的 systick 定时；为了实现按键不松开时，对应的显示跳变数每隔 200ms 自动加 1，在 systick 中断服务程序中引出 200ms 的 systick 定时。

```
1.  void SysTick_Handler(void)
2.  {
3.      if (systick_200ms_couter == 0) //利用 1ms 的 SysTick 产生 200ms 的定时
    器
4.      {
5.          systick_200ms_couter = 200;
6.          systick_200ms_status = 1;
7.      }
8.      else
9.          systick_200ms_couter--;
10.
11.     if (systick_1s_couter == 0) //利用 1ms 的 SysTick 产生 1s 的定时器
12.     {
13.         systick_1s_couter = 1000;
14.         systick_1s_status = 1;
15.     }
16.     else
17.         systick_1s_couter--;
18. }
```

在 while 循环中，利用全局变量 cnt 计数，在 1s 的 systick 定时到达时自动加 1，min（分钟）、sec（秒钟）均由 cnt 计算得到。

变量 i 控制显示的数码管，在 switch 中分成 5 种情况。其中短横线的实现只需要 result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,64)即可。

按下按键不松开时，每过 200ms 自动加 1 秒的功能和按下按键自动加 1 的功能同时实现。利用 200ms 的 systick 定时，每次检测到按键按下，便 200ms 自动加 1。这种方法的弊端是，如果 1s 内按键按下次数超过 5 次，时钟只会实现 5 次加 1。但是采用这种方法合并了功能，不仅使得程序简单，还同时实现了对按键的延时消抖，因为按下按键 200ms 才会起作用。考虑到按键在 1s 内超过 5 次按下的概率很小，综合考虑下决定使用此方法。

代码如下：

```
1.  while (1)
2.      {
3.
4.          if (systick_200ms_status)
5.          {
6.              systick_200ms_status     = 0;
7.
```

```
            key_1 = GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0);    //USR_SW1
            key_2 = GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1);    //USR_SW2

            if (key_1 == 0){
                cnt++;  //按下 USR_SW1 时，秒数自动加 1，200ms 加一次
            }

            if (key_2 == 0){
                cnt = cnt + 60; //按下 USR_SW2 时，分钟自动加 1，200ms 加一次
            }
        }

        if (systick_1s_status)
        {
            systick_1s_status   = 0;

            cnt++;
        }

        min = cnt / 60;
        sec = cnt - 60 * min;

        if (sec >= 60){
            sec = sec - 60;
            min++;
        }

        if (min >= 60){
            sec = 0;
            min = 0;
            cnt = 0;
        }

        i++;
        i = i % 5;


        //点亮五位数码管

        switch(i){
            case 0: //分钟十位
```

```
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0);                    //write port 2
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,seg7[min/10]); //write port 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,1);                    //write port 2

            case 1: //分钟个位
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0);                    //write port 2
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,seg7[min%10]); //write port 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,2);                    //write port 2

            case 2: //短横线
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0);                    //write port 2
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,64);              //write port 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,4);                    //write port 2

            case 3: //秒数十位
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0);                    //write port 2
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,seg7[sec/10]); //write port 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,8);                    //write port 2

            case 4: //秒数个位
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,0);                    //write port 2
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,seg7[sec%10]); //write port 1
                result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,16);                   //write port 2

        }

    }
```

4. 讨论题解答

（1）I2C 是同什么类型的总线（同步/异步、串行/并行、单工/双工）？传输速率是多少？

答：是一种同步串行，双向两线制总线。

传输速率：标准 100Kbit/s，快速 400Kbit/s，快速附加 1Mbit/s，高速 3.33Mbit/s。

（2）在同一 I2C 总线上最多可以连接多少 PCA9557 和 TCA6424 芯片？如果 PCA9557 的 A2、A1 管脚连接到高电平，A0 管脚接到地时，它的 I2C 从机地址为多少？

答：最多 112 个。从机地址为 0x1E。

（3）若 I2C 主设备传送的第一个字节为二进制数 01000111B，其表达的含义是什么？

答：表示从机地址和数据方向。含义为主机接收 01000111 地址从机的信息。

（4）若时钟源选用 25M 外部 MOSC，则能否利用 SysTick 直接产生 8s 的定时？请给出设置方法或不能设置的理由。

答：可以。先设置计数初值 12.5M 得到 500ms 的 systick 定时，然后通过软件 16 分频就能够得到 8s 定时。

（5）例程 2-4 中 3 个任务的优先级为多少？程序是如何实现“任务 3 的优先级高于任务 1，2”的要求的？

答：

PF0 闪烁和 LED 跑马灯的优先级一致，按下 USR_SW1 按键时，PN0 常亮的优先级最高。

在任务 3 被触发时，修改其余两个任务的变量，使程序运行到另两个任务时因为不满足条件而不执行。任务 3 要结束时再将变量修改回来，恢复执行另两个任务。在中断触发的模块，当按钮按下时，两个计数器一直为零，即任务 1 和 2 的进程被阻止，以此来实现任务 3 的优先级最高。

5. 感想和建议

在第一次实验的基础上，本次实验中我对 GPIO 的工作原理进行了更深入的了解，能够通过扩展 GPIO 来输出点亮 LED 及动态数码管。

此外，我还了解了 I2C 总线标准，掌握了用 I2C 总线扩展 GPIO 芯片 PCA9557 及 TCA6424 的方法。另外我进一步熟悉了 SYSTICK 定时中断的使用，实现了一些定时功能。

实验要求 6 中，我权衡了自己的现有方案的局限性和优点，并且做出抉择，让我直观地、

循序渐进地掌握嵌入式开发的知识，收获很大。

最后，感谢《工程实践与科技创新Ⅱ-A》课程组准备的高质量课程和嵌入式开发条件，感谢朱兰娟老师为本门课程的精心准备与悉心讲解，深入浅出的为我们讲解嵌入式开发的诸多知识，将复杂的嵌入式知识网络按照对应模块分割开来，逐一细解，帮助我们拓宽视野，增长见识。感谢所有为我答疑解惑，帮助过我的老师、助教和同学。