目　录

实验一 I/O与时钟

一、实验目的

1、熟悉实验板和MSP430开发环境

2、掌握MSP430 I/0口线的操作

3、掌握MSP430时钟模块的设置

二、实验原理

1、I/O 口线操作

使用实验板上的 P5.1 、LED3 来完成口线操作，相关电路如图 1-1 所示：

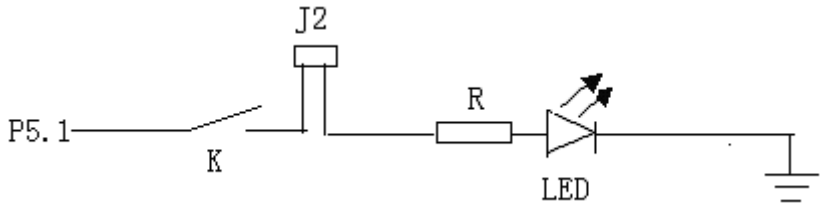


图 1-1 实验电路

图中开关 K 是 DIP 开关 P11 的 SW6，P5.1 通过 DIP 开关 P11 的 SW6、跳线 J2、电阻 R 连接 LED3，P5.1 可输出高电平点亮 LED3 或者输出低电平使 LED3 熄灭。

2、时钟模块

MSP430 系列芯片的时钟模块有高速晶振、低速晶振、DCO、FLL 和 FLL+,其中 MSPX44X 系列包含高速晶振、低速晶振、DCO 和 FLL+,可根据不同需求产生、输出不同时钟频率。MSP430X44X 系列由于使用了增强型锁频环技术 FLL+，消除了 RC 振荡器因电压、温度等不稳定因素带来的不稳定性，可以在低频情况下得到较高的稳定频率，DCO 的输出频率由倍频因子和 DCO+共同决定：

当 DCO+ = 0 时，FDCOCLK＝Faclk * (N+1)

当 DCO+ = 1 时，FDCOCLK＝Faclk *(N+1) *D

系统时钟和子系统时钟除了可以使用 DCO 产生的时钟频率外，还可以使用 XT2，XT2 的典型设置为 8MHZ 或者 4MHz。端口 P1 的 P1.1、P1.4、P1.5 在实验中用于输出 MCLK、SMCLK、ACLK。

三、实验内容

1、 熟悉实验板和 EW430 开发环境

要求：

能够连接仿真器、电源到实验板

能够创建工程、文件，熟练进行 C 程序和汇编程序的编译、连接

2、I/0 端口实验

要求：通过口线 P5.1 控制 LED3

3、设置、观测时钟频率

要求：设置时钟，使得 P1.1,P1.4 输出时钟频率为 ACLK*10,P1.5 输出为 ACLK

四、实验步骤

1、设置 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON

置 DIP 开关 P10 的 SW1,SW2 为 ON

2、打开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、连通 J2，进行 I/O 实验

4、置DIP开关P6的SW6、DIP开关P5的SW1、开关P5的SW2为ON，观测时钟频率

五、分析与思考

1、 修改程序，使 MCLK = 4MHZ。

2、 在不使用 XT2 的情况下，MCLK 的最大频率可为多少？

3、 ACLK 有没有可能比 MCLK 的频率还要低？请说明原因。

六、实验参考代码

1.时钟实验参考代码（C 语言）

```
/*******************************************************************
*   文件名称：
*           clock.c
* 文件说明：
*       通过实验，熟悉如何设置系统主时钟(MCLK)、辅助时钟(ACLK)、
*        子系统时钟(SMCLK)
*        程序运行后可以得到 ACLK＝32768、MCLK＝SMCLK＝32768*10
*
*******************************************************************/

#include <MSP430x44x.h>

/*******************************************************************
*       main（）函数
*******************************************************************/

void main(void)
{
    WDTCTL     = WDTPW + WDTHOLD; //  关闭看门狗
    P1DIR      = 0x32;                        //  设置  P1.1,P1.4,P1.5  方向
    P1SEL      = 0x32;                       // P1.1,P1.4,P1.5  为外围模块
    FLL_CTL1 = FLL_DIV0;                     //  设置  p1.5  输出频率
                                             //  设置  FLL_CTL1 = 0x32;
    SCFQCTL   =0x09;                         //  设置  SMCLK = ACLK * 10
                                             //  设置  MCLK = ACLK * 10
    while(1);                                //  空循环，供用户检测其输出频率

}
```

2.IO 时钟参考程序

```
/*******************************************
*     文件名：
```

```
*           LED3.c
*    功能：
*       主程序
*       运行后可以看到 LED3 每过大约 1 秒闪一次
*******************************************/
#include <msp430x44x.h>

/*******************************************/
void main(void)
{
      unsigned long tmp;

       WDTCTL = WDTHOLD + WDTPW;        //关闭看门狗
       P5OUT = 0x02;                    //设置 P5.1 输出为 1
       while(1)
       {                                //循环
           P5DIR = 0x02;                //  使能 P5.1 为输出
           P5OUT ^= 0x02;               //  对输出置反
           for(tmp=0;tmp<67500;tmp++);//  延时
       }

}
```

实验二 键盘与LED

一、实验目的

1、 熟悉键盘的设计方法，掌握键盘工作原理和编程设计

2、 掌握 LED 静态显示和动态扫描显示的原理和编程设计

二、实验原理

1、键盘原理

在单片机实验中，键盘的设计主要有三种方式：独立按键式键盘、行列扫描式键盘、N 线 N*(N-1)键盘，下面逐一介绍。

（1） 独立按键式键盘

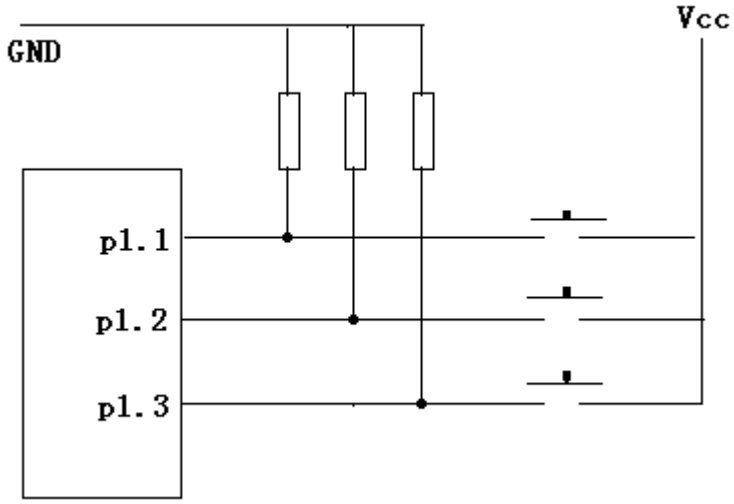


图 2-1 独立按键原理图

这种键盘使用单片机的 I/O 口线直接连接，每个按键对应一根口线，如图 2-1 所示。键盘的工作原理：设置各个口线为输入模式，通过中断方式或者软件查询方式获取各个口线的是否有键按下，在图 2-1 的键盘中，有键按下则口线端电平为高，否则为低电平。独立按键键盘使用简单，但是每根口线只对应一个按键，如果系统需要较多按键，往往不能满足需求，因此这种方式适用于需要按键较少的应用。

（2） 行列扫描式键盘

图 2-2 所示键盘使用，P3.0 ～ P3.7 8 根口线接 16 个按键。键盘为 4*4 格局，P3.4，P3.5，P3.6，P3.7 为行线，P3.0、P3.1、P3.2、P3.3 为列线。列线分别由上拉电阻上拉到 VCC，在行线与列线的每一个交界处有个按键，按键的两端分别接在行线和列线上。

◆ 判断是否有键按下：

如果有键按下，则与之相连的行线与列线被连通。在检测是否有键按下时，先使 4 条行线 P3.4—P3.7 输出低电平，读列线 P3.0—P3.3。如果有按键按下，则列线读进来的数据非全 1。如果没有按键按下，则因所有列线被上拉，读入的数据为全 1。由此，可判断是否有按键被按下。由于在按键的前后会有抖动，因此在判断键值前要消除抖动。一般消除抖动的方法有软件延时法和定时中断法两种方法。

◆ 键值识别：

首先根据读出的 P3 端口 P3.0、P3.1、P3.2、P3.3 哪个口线输入的值不为 1 确定按键位于那一列，然后依次检测在 P3.4、 p3.5 、p3.6，P3.7 的值为 1000、0100、0010、0001 的情况下 P3.0 ～ P3.3 的值是否为全 1，如果是全为 1 说明按键位于在的值为 1 的口线所在的行，从而确定按键的位置。 例如，当“1”键按下时，键值判别过程为：首先 P3.4～P3.7 输出为 0，读取 P3.0～P3.3 的输入（此时输入应该为“1110”），由输入值可知 P3.3 所在的列有按键按下，然后判断行。置 P3.4～P3.7 的输出为“1000”，再读取 P1IN 的值，判断 P3.0～P3.3 是否为全 1，此时显然不是全 1，置 P3.4～P3.7 的输出为“0100”，再进行判别，此时输入依然不是全 1。当 P3.4～P3.7 的输出置位“0001”时，P3.0～P3.3 的输入为全 1，于是得到按键所在的行是 P3.7。获得行列之后，就确定了按键的位置。

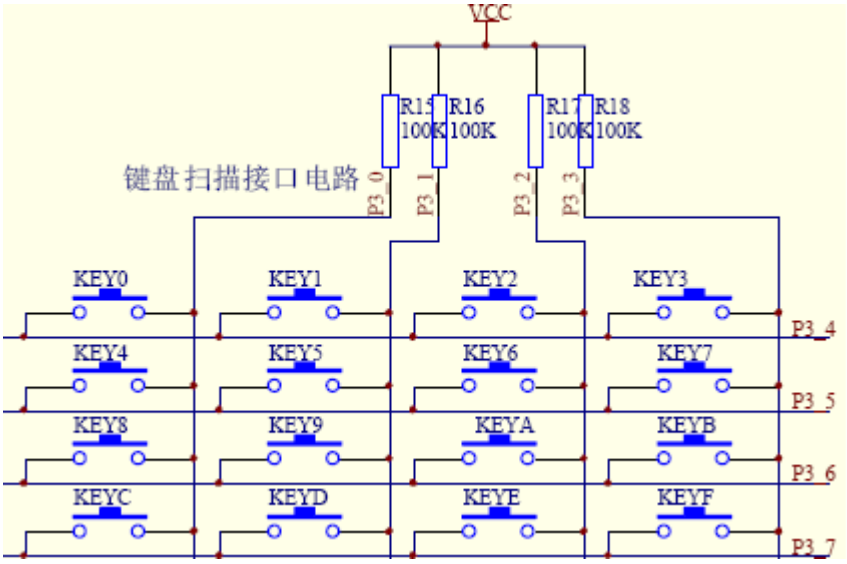


图 2-2 行列式键盘

（3） N 线 N*(N-1)按键键盘

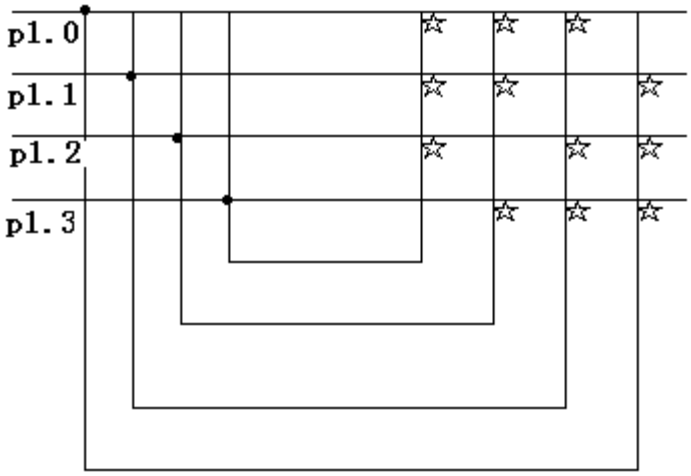


图 2-3 N*(N-1)键盘

此种设计可以用 N 根口线来连接 N*(N-1)个按键，在图 2-3 中，使用 4 根线来

连接 12 个按键，星号代表按键。键值获取的过程如下：

P1.0 输出为 1，P1.1、P1.2、P1.3 为输入模式，检测 P1.1、P1.2、P1.3 的输入是否为全 0，如果不是全 0，比如 P1.1、P1.2、P1.3 的值为（001），说明 p1.0 与 p1.3 对应的交叉处有按键按下。如果为全 0，说明没有按键按下，继续使用该方法判别 P1.1、p1.2、p1.3 行上是否有键按下。

当然，判别过程中间也要消除抖动，为了便于叙述没有在上面指出。这种键盘设计一般用于严格节省口线的应用下。

2、LED 工作原理

LED 显示器通常有 a、b、c、d、e、f、g 和 dp 8 个发光二极管组成，每个二极管成为一个段。a、b、c、d、e、f、g 为用于显示数字或者字母的段，dp 为小数点。根据 LED 的结构可以分为共阴极和共阳极两种，如图 2-4、2-5 所示：

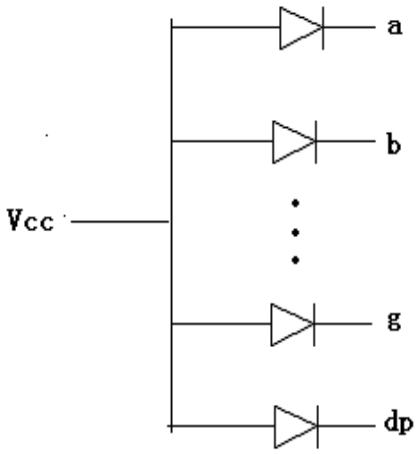


图 2-4 共阳极 LED

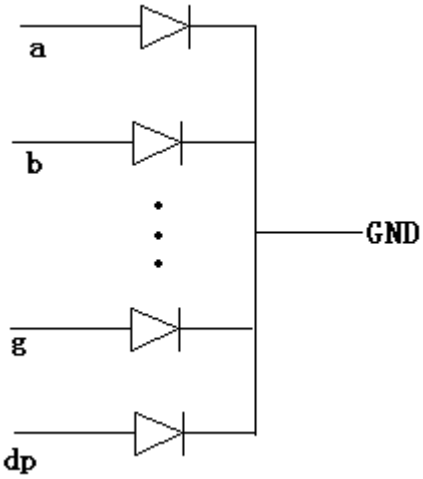


图 2-5 共阴极 LED

LED 的使用方法可以分为静态显示和动态扫描显示两种。直接连接显示就是使用 8 个口线控制 LED 的八个段，这种方式简单易用，只是占用口线较多，故一般采用扫描显示方法。扫描显示是基于人眼的“视觉暂留”来实现的，通过不断的显示不同的 LED 使人感觉几个不同的 LED 在一直显示。

3、 实验所用电路说明

LED 电路

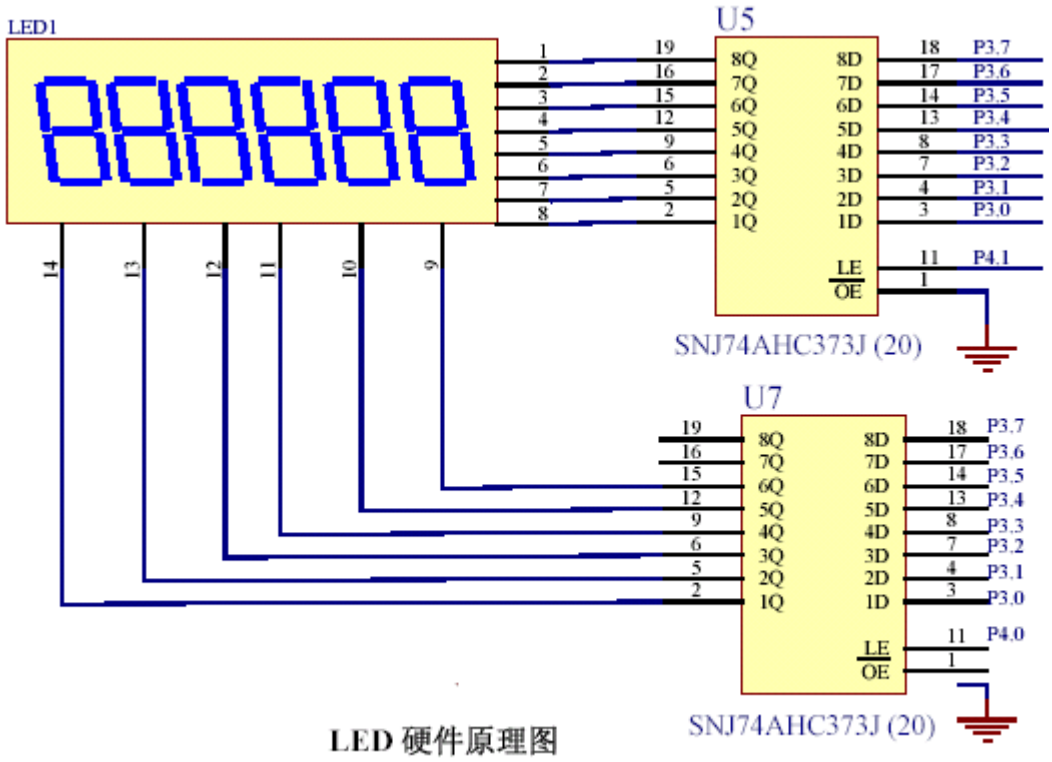


图 2-7 LED 显示电路

LED 显示电路包含友六只LED，共阴极，从左到右依次为D5～D0，其中，D5 为最高位。P3.0～P3.7 既为LED 的显示代码输入，又为LED 的位选控制，通过SNJ74AHC373J 锁存与P4.0、P4.1选择来实现。其工作方式为：当P4.1 为高电平，P3.0～P3.7 为LED 的显示代码输入，依次对应LED 的a、f、b、g、c、dp、d、e 当P4.0 为低电平，P3.0～P3.5 为LED 位选控制输入，依次对应D5、D4、D3、D2、D1、D0。

LED 的显示代码：（0～9）

0——7BH　1——12H　2——4FH　3——7FH　4——36H

5——3DH　6——7DH　7——1BH　8——7FH　9——3FH

三、实验内容

1、 分别在静态模式和动态模式下，用 LED 显示数据

2、 使用独立按键式键盘，用 LED 显示键值

3、 使用扫描式键盘并用 LED 显示用户的按键，显示方式使用扫描显示

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON

置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、设置下列开关为ON，做独立按键实验

DIP 开关 P17 的 SW3、SW4、SW5，DIP 开关 P6 的 SW6、SW7、SW8

DIP 开关 P2 的 SW1、SW2、SW3、SW4、SW5，DIP 开关 P4 的 SW6

DIP 开关 P3 的 SW1、SW2、SW3、SW4

4、置下列开关如下状态，做行列键盘实验

DIP 开关 P5 的 SW1、SW2、Sw3、SW4 为 ON
DIP 开关 P6 的 SW6、SW7、SW8 为 ON
DIP 开关 P2 的 SW1、SW2、SW3、SW4、SW5，DIP 开关 P4 的 SW6
DIP 开关 P3 的 SW1、SW2、SW3、SW4
DIP开关P17的SW2、SW3、SW4、SW5、SW6为OFF

五、分析与思考

1、 如果在 LED 扫描显示时，至少要以多高的频率扫描才能使 LED 没有闪烁感？
2、 如果在键盘键值识别中，不进行消除抖动处理，会出现什么后果？
3、 P1SEL 置位全 1 ，键盘实验会有什么不同？
4、 把键盘的按键处理过程使用一个有限自动机表示出来

六、实验参考代码

行列式键盘实验源程序

```
/*******************************************************************************
*   文件名称：
*             main.c
*   文件说明：
*         程序运行后，行列扫描键盘工作,有按键按下时按键的键值显示到 LED。行列键盘
*    的“＃”对应键值为 A,“* ”对应键值为 B
*******************************************************************************/

#define MSP430F449_H 0
#include <msp430x44x.h>

#ifndef LED_IN_USE
#include "..//..//led//led.c"
#endif

#ifndef KEY_BOARD
#include "keyboard12.c"
#endif

/*******************************************************************************
*   main()函数
*******************************************************************************/
void main(void)
{
      unsigned char tmp;
      WDTCTL = WDTHOLD + WDTPW ;       // 停止看门狗
      init_LED();                      // 初始化 LED
      init_Keyboard();                 // 初始化键盘
      while(1)
      {
            key_Event();               //检测按键事件
```

```
            if (key_Flag == 1)                  // 检测 key_val 里是否有键值可以读取
            {
              for(tmp=LED_IN_USE-1; tmp>0;tmp--)
              {
                led_Buf[tmp]=led_Buf[tmp-1];// 键值左移
              }
              led_Buf[0]=key_val;               // 取出当前键值
              key_Flag = 0;                     // 恢复键盘按键标识
            }
          led_Display();                        // 使用 LED 键盘数据
    }
}

/*****************************************************************************
*       文件名称：
*                keyboard12.c
*        程序功能描述：
*                行列式键盘检测
*    输入：
*                用户的按键事件
*    输出：
*                存放用户输入的键值
*
*
*****************************************************************************/

#ifndef MSP430F449_H
#include <msp430x44x.h>
#endif

#define KEY_BOARD 1

unsigned char key_Pressed,          // 是否有键值按下
              key_val,              // 存放键值
              key_Flag;             // 是否一个按下的按键已经松开，
                                    // 即是按键的键值可以读取

unsigned char key_Map[16] ={      //设置键盘逻辑键值与程序计算键值的映射
                            0x0C,0x0D,0x0E,0x0F,
                            0x08,0x09,0x0A,0x0B,
                            0x04,0x05,0x06,0x07,
                            0x00,0x01,0x02,0x03
                            };
```

```
/*****************************************************************************
*   初始化键盘设备
*****************************************************************************/
void init_Keyboard(void){

        P3DIR |= 0x0F;   // P3.0~P3.3  设置为输入模式
        P3DIR &= 0x0F;   // set p3.4~p3.7  设置为输出模式
        P3OUT |= 0xF0;   // p3.4~p3.7  输出值清零
        key_Flag       = 0;//  初始化 key_Flag
        key_Pressed = 0;//  初始化 key_Pressed
        key_val = 0xFF;

}
/*****************************************************************************
* Check_Key(),检查按键，确认键值
*****************************************************************************/
void check_Key(void){

    unsigned char row ,col,tmp1,tmp2;
    // tmp1 用来设置 P3OUT 的值，使 P1.1~P1.3 中有一个为 0
    tmp1 = 0x80;
    for(row=0;row<4;row++){
        P3OUT |= 0xF0;                                  // p3.4~P3.7=1
        P3OUT &= ~tmp1;                                   // P3.4~p3.7 中有一个为 0
        tmp1 >>= 1;                                    // tmp1  右移一位
        if ((P3IN & 0x0F) < 0x0F){                      //  是否 P3IN 的 P3.0~P3.3 中有一位为 0
            tmp2 = 0x01;                                  // tmp2 用于检测出那一位为 0
            for(col =0;col<0x04;col++){                //  列检测
                if((P3IN & tmp2)==0x00){                //  是否是该列
                   key_val =key_Map[ row*4 +col] ; //  获取键值
                   return;                                //  退出循环
                }
                tmp2 <<= 1;                                 // tmp2 左移
            }
        }
        if(key_val==0xFF)
        {
            key_Pressed = 0;
        }
    }

}
```

```
/*****************************************************************************
* 延迟，用于消除抖动
*****************************************************************************/
void delay(){
        unsigned char tmp;
         for(tmp=0xff;tmp>0;tmp--);
}
/*****************************************************************************
*   key_Event()，检测键盘是否有键按下，如果有获取键值
*****************************************************************************/
void key_Event(void){

        unsigned char tmp;
        P3OUT &= 0x0F;                  // 设置 P3OUT 输出值
        P3OUT = 0xFF;
        P3DIR &= 0xF0;    // P3.0~P3.3  设置为输入模式
        P3DIR |= 0xF0;    // set p3.4~p3.7  设置为输出模式
        P3OUT &= 0x0F;                  // 设置 P3OUT 输出值
         tmp = P3IN;                    // 获取  p3IN
        if ((key_Pressed ==0x00)&&((tmp & 0x0F) < 0x0F))
        {                               //是否有键按下
              key_Pressed = 1;      // 如果有按键按下，设置 key_Pressed 标识
              delay();                  //消除抖动
              delay();
              delay();
              check_Key();              // 调用 check_Key(),获取键值
        }else if ((key_Pressed ==1)&&((tmp & 0x0F) == 0x0F))
        {                               //是否按键已经释放
              key_Pressed = 0;      // 清除 key_Pressed 标识
              key_Flag       = 1;     // 设置 key_Flag 标识
        }

}
/*****************************************************************************
*   文件名称：
*                 led.c
*   文件说明:
*               显示的时候首先设置要显示的内容，然后使能相应的 LED
*
*****************************************************************************/

#ifndef MSP430F449_H
#include <msp430x44x.h>
#endif
```

```
#define LED_IN_USE 6
/*****************************************************************************/
/*     数据定义                                                              */
/*****************************************************************************/
const unsigned char NUM_LED[17]=
                              {0xd7,0x14,0xcd,0x5d,0x1E,   // 0 ~ 4
                              0x5b,0xdb,0x15,0xdf,0x5f,     // 5 ~ 9
                              0x9f,0xda,0xc3,0xcc,0xcf,     // a ~ e
                              0x8b,0x00};                   //f,0x00 使 LED 不显示


unsigned char led_Buf[LED_IN_USE];   // LED 显示缓冲区 ，
                                     // 存放要显示数据
unsigned char led_Ctrl;
/*****************************************************************************
*       模块初始化
*****************************************************************************/
void init_LED(void){
      char tmpv;
      P3DIR   = 0xff;          // 设置 p3 输出
      P3OUT   = 0x00;       // 设置 初始值为 0
      P4DIR |= 0x03;          // 设置 p4.0,p4.1 输出
      P4OUT &= 0xfc;       // 设置初始值
      led_Ctrl = 0;             // led_Ctrl 用于控制那个 LED 可显示
      for(tmpv=0;tmpv<LED_IN_USE;tmpv++)
      {                         // 初始化缓冲区
        led_Buf[tmpv] = 0;
      }

}
/*****************************************************************************
*       LED 显示  ,该函数可以放到定时器中断中
*****************************************************************************/
void led_Display(){
      unsigned tmp ;
      P3DIR   = 0xff;       // 设置 p3 输出
      P3OUT   = 0x00;       // 设置 初始值为 0
      P4DIR |= 0x03;       // 设置 p4.0,p4.1 输出
      P4OUT &= 0xfc;       // 设置初始值
        tmp       = 0x01;
        P3OUT   = NUM_LED[led_Buf[led_Ctrl]];   // 设置显示值
        P4OUT |= 0x02;                          // 打开数据锁存器
        P4OUT &= 0XFD;                          // 关闭数据锁存
```

```
        P3OUT    = ~(tmp<<led_Ctrl);                  // 设置那只 LED 显示
        P4OUT    |= 0x01;                              // 打开控制锁存
        P4OUT    &= 0XFE;                              // 关闭控制锁存
        led_Ctrl= (led_Ctrl +1) % LED_IN_USE; // 设置下一个要显示的 LED

}
```

实验三 定时器实验

一、实验目的

1．掌握看门狗定时器的两种工作模式

2．熟练设置、使用基本定时器

3．熟练设置、使用 Timer_B

二、实验原理

1．看门狗定时器

MSP430 系列芯片都有看门狗定时器，看门狗定时器其实就是一个 16 位定时器，但是其操作需要口令，看门狗定时器具有看门狗和定时器两种工作模式。

看门狗定时器在看门狗模式下，其作用在于发现程序跑飞，原理为：看门狗定时器设置一定时间，这个时间是所有用户程序一定能在此时间内执行完的时间，设置好该时间后，所有用户程序必须在此时间间隔内把看门狗定时器清零。如果 CPU 执行正确，则看门狗始终在规定时间内被用户程序清零，如果 CPU 执行程序跑飞，看门狗定时器得不到用户程序的清零产生溢出，导致 CPU 复位，这样 CPU 又重新运行用户程序。在定时器模式下，看门狗定时器可以在设置时间到后，产生中断，同其他定时器工作方式相同。

2．基本定时器（Basic Timer 1）

Basic Timer 1 是 MSP430X3XX、MSP430F4XX 系列器件中的模块，用于向其他外围模块提供低频控制信号。Basic Timer 1 有两个计数单元：BTCN1 和 BTCN2，通过适当设置 Basic Timer 1 可以是两个 8 位定时器也可以是一个 16 为定时器。

3．Timer_B

Timer_B 是 MSP430 系列芯片都有的一个 16 为定时器/计数器，使用极为广泛。Timer_B 可支持同时进行多种时序控制、多路捕获、比较以及多种输出波形，每个捕获/比较模块均可独立编程。Timer_B 有多种可选的计数器时钟源，八种输出模式。Timer_B 有两种工作模式：定时器模式和捕获模式。在定时器模式下有四种计数模式：停止模式、增计数模式、连续计数模式、增/减计数模式，在捕获/比较模式下,又可以进一步分为捕获模式和比较模式。

4．实验使用电路

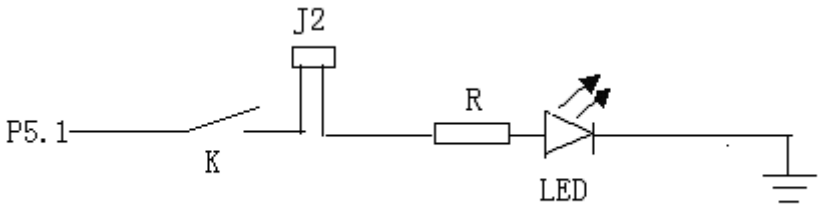


图 3-1 实验电路

基本定时器和看门狗的实验需要控制 LED3，电路如图 3-1，Timer B 由于要输出 PWM 波形，只需要使用到两个端口测量输出波形，没有其他器件的连接。

三、实验内容

1． 看门狗定时器操作

在定时器模式下，使用中断控制 LED3

在看门狗两种模式下使用 RESET 控制 LED3

2． 基本定时器操作

通过定时器中断进行键盘和 LED 的扫描

3． Timer_B 定时器操作

通过设定 Timer B 输出不同的 PWM 波形

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON

置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、设置 P11 的 SW6 为 ON，做看门狗实验

4、设置 DIP 开关做 Basic Timer 实验

DIP 开关 P5 的 SW1、SW2、Sw3、SW4 为 ON

DIP 开关 P6 的 SW6、SW7、SW8 为 ON

DIP 开关 P2 的 SW1、SW2、SW3、SW4、SW5，DIP 开关 P4 的 SW6

DIP 开关 P3 的 SW1、SW2、SW3、SW4

DIP开关P17的SW2、SW3、SW4、SW5、SW6为OFF

5、设置 DIP 开关 P6 的 SW7、P5 的 SW5 用于输出 PWM 波形

五、分析与思考

1． 看门狗和定时器两种工作模式的主要差别是什么？

2． 程序 TB.c 中如果使用模式 5，将输出什么波形？

六、实验参考代码

1.TimerB 输出 PWM 波形

```
/*****************************************************************************
*  文件名称：
*            TB.c
*  文件说明：
*            使用 TB 输出 PWM 波形，可通过 set_TB 设定不同的模式
*****************************************************************************/
#include  "msp430x44x.h"
/*****************************************************************************
* 设置 TimerB 输出 PWM 的工作模式
*****************************************************************************/
void set_TB(int mode)
{
    if (mode==2)
    {                                  // 翻转/复位模式，CCR1＝25％，CCR2＝75％
        TBCCR0   = 11;                   // PWM 周期
        TBCCTL1 = OUTMOD_2;     // PWM toggle/reset
        TBCCR1   = 9;                  // ccr1 Pwm cycle
        TBCCTL2 = OUTMOD_2;     // PWM toggle/reset
        TBCCR2   = 3;                  // ccr2 Pwm cycle
```

```
    P2DIR |= 0x02;                // p2.1  模式设置
    P2SEL |= 0x02;               // p2.1 option select
    P2DIR |= 0x08;               // P2.3 output mode
    P2SEL |= 0x08;               // P2.3 option select
    TBCTL |= MC0;                 //  设置递增模式
}else if (mode==0x03)
{                                  //置位/复位模式，CCR1＝25％，CCR2＝75％

    TBCCR0    = 11;               // PWM  周期
    TBCCTL1 = OUTMOD_3;    // PWM toggle/reset
    TBCCR1    = 9;               // ccr1 Pwm cycle
    TBCCTL2 = OUTMOD_3;    // PWM toggle/reset
    TBCCR2    = 3;               // ccr2 Pwm cycle
    P2DIR |= 0x02;               // p2.1 output mode
    P2SEL |= 0x02;               // p2.1 option select
    P2DIR |= 0x08;               // P2.3 output mode
    P2SEL |= 0x08;               // P2.3 option select
    TBCTL |= MC0;               //  设置递增模式
}else if (mode==0x04)
{                                 //  翻转模式，CCR1＝50％，CCR2＝50％
    TBCCR0    = 11;               // PWM  周期
    TBCCTL1 = OUTMOD_4;    // PWM toggle/reset
    TBCCR1    = 9;               // ccr1 Pwm cycle
    TBCCTL2 = OUTMOD_4;    // PWM toggle/reset
    TBCCR2    = 3;               // ccr2 Pwm cycle
    P2DIR |= 0x02;               // p1.2 output mode
    P2SEL |= 0x02               // p1.2 option select
    P2DIR |= 0x08;               // P2.3 output mode
    P2SEL |= 0x08;               // P2.3 option select
    TBCTL |= MC0;               //  设置递增模式
}else if (mode==0x06)
{                                  //翻转/置位模式，CCR1＝75％，CCR2＝25％

    TBCCR0    = 11;                // PWM  周期
    TBCCTL1 = OUTMOD_6;     // PWM toggle/reset
    TBCCR1    = 9;               // ccr1 Pwm cycle
    TBCCTL2 = OUTMOD_6;     // PWM toggle/reset
    TBCCR2    = 3;               // ccr2 Pwm cycle
    P2DIR |= 0x02;               // p2.1 output mode
    P2SEL |= 0x02;               // p2.1 option select
    P2DIR |= 0x08;               // P2.3 output mode
    P2SEL |= 0x08;               // P2.3 option select
  TBCTL |= MC0;                //  设置递增模式
}else if (mode==0x07)
```

```
        {                                   //  复位/置位模式

            TBCCR0    = 11;                  // P2.1--> CCR1 - 75% PWM
                                             // P2.3--> CCR2 - 25% PWM
                                             // PWM  周期
            TBCCTL1 = OUTMOD_7;    // ccrl reset/set
            TBCCR1    = 9;                 // ccr1 Pwm cycle
            TBCCTL2 = OUTMOD_7;      // ccr1 reset/set
            TBCCR2    = 3;                  // ccr2 Pwm cycle
            P2DIR |= 0x02;                  // p2.1 output mode
            P2SEL |= 0x02;                  // p2.1 option select
            P2DIR |= 0x08;                  // P2.3 output mode
            P2SEL |= 0x08;                  // P2.3 option select
            TBCTL |= MC0;                   //  设置递增模式
        }
}
/*******************    在 Timer B 的不同工作模式下输出 PWM 波形    ***************/
void main()
{
        unsigned int pwm_Delay;
        char tmpv,tmp[5] = {2,3,4,6,7};

        WDTCTL = WDTHOLD + WDTPW;    //关闭看门狗
          FLL_CTL0 |= XCAP14PF;
        TBCTL = TBSSEL0 + TBCLR;    //ACLK,清除 TAR
        tmpv=0;
        while(1)
        {
            set_TB(tmp[tmpv]);                                      //选择  PWM  模式
            tmpv = (tmpv + 1) % 5;                                  //改变模式
            for(pwm_Delay=0;pwm_Delay<0xffff;pwm_Delay++);//delay
        };
}
```

2. Basic Timer 实验

```
/*****************************************************************************
*     文件名称：main_c.c
*     文件说明：测试  basic Timer1  定时器功能的 C 程序
*     实验使用定时器中断方式，重做键盘与 LED 的实验
*****************************************************************************/
#include <msp430x44x.h>
#include "keyboard12.c"
#include "led.c"
```

```
/*******************************************************************************
*       初始化  basic timer 1
*******************************************************************************/
void    init_BT(void)
{
    BTCTL   = 0x06;                         //   Basic Timer 1  中断频率设置
    IE2     |= 0x80;                        //  使能  basic timer  中断
}

/*******************************************************************************
*   main（）
*******************************************************************************/
void main()
{
      WDTCTL = WDTHOLD + WDTPW;   //关闭看门狗
      init_BT();                              //初始化 Basic Timer 1
      init_Keyboard();              //初始化键盘
      init_LED();                   //初始化 LED
      _EINT();                      //使能中断
}

/*******************************************************************************
*   中断处理函数
*******************************************************************************/
#pragma vector = BASICTIMER_VECTOR
__interrupt void    BT_Interrupt(void)
{
      char tmp;
      key_Event();                  //检测按键事件
      if (key_Flag == 1)            //  检测 key_val 里是否有键值可以读取
      {
          for(tmp=LED_IN_USE-1; tmp>0;tmp--)
          {
              led_Buf[tmp]=led_Buf[tmp-1];//  键值左移
          }
          led_Buf[0]=key_val;       //  取出当前键值
          key_Flag = 0;             //  恢复键盘按键标识
      }
      led_Display();                //  使用 LED 键盘数据
}
```

实验四 外围模块操作

一、实验目的

1、了解硬件乘法器的原理，掌握其操作编程

2、熟练使用比较器，掌握其操作编程

3、掌握 MSP430 片内 Flash 的读写操作

二、实验原理

1、乘法器

MSP430 的 MSP430X3XX、MSP430F14X、MSP430F44X 系列均带有硬件乘法器，可以实现 8×8、16×8、8×16、16×16 运算，支持无符号乘法（MPY）、有符号乘法（MPYS）、无符号乘加（MAC）、有符号乘加(MACS)，使用硬件乘法器可以较大的提高运算速度。从一个编程人员的角度，硬件乘法器的结构图如图 4-1：

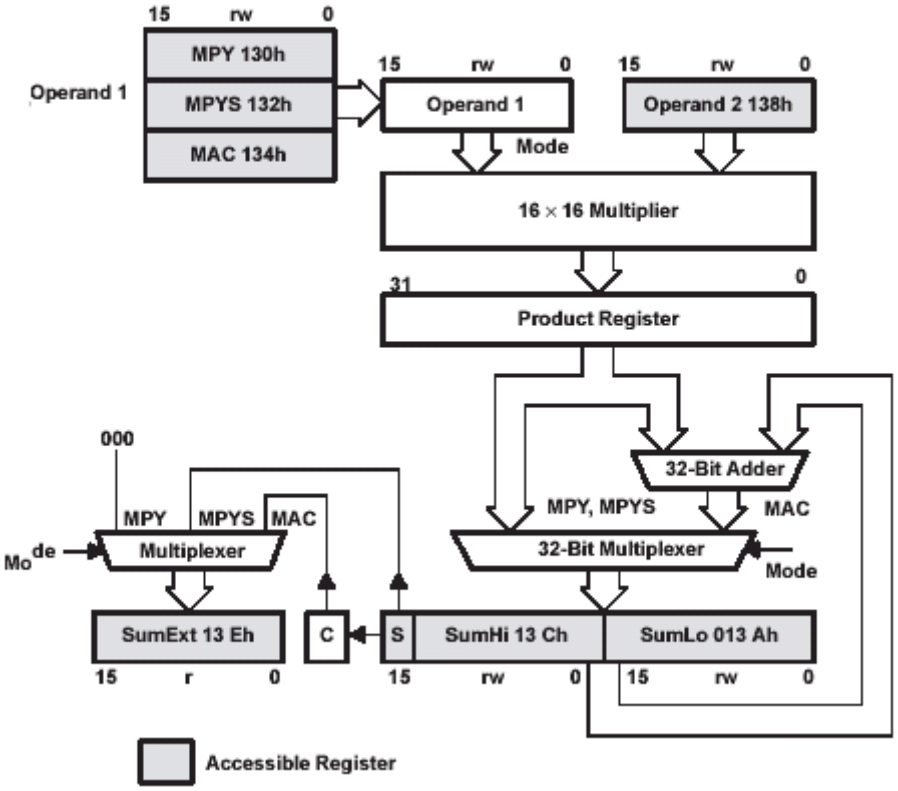


图 4-1 硬件乘法器结构图

硬件乘法器属于MSP430的片内外设，不受CPU的影响，即使是在PUC或者POR事件发生时，硬件乘法器的寄存器也不会发生变化。硬件乘法器的操作类型是由乘法器操作数1使用的地址决定的：地址 130h说明要执行无符号乘法(MPY)，

地址 132h说明要执行有符号乘法(MPYS)，

地址 134h执行无符号乘加（MAC）。

硬件乘法器一般用在以下场合：

✧ 多于16位的乘法，这种乘法可以通过多做几次16位的乘法完成。比如45的无符号

乘法可以把乘数和被乘数分别放到3个寄存器中R5、R6、R7、R8、R9、R10，存放的顺序是高16位、中间16位、低16位。结果可以这样得到：Result＝R7*R10+(2^16)*(R6*R10+R7*R9)+(2^32)*(R5*R9 ＋ R6*R8)+(2^64)*R5*R8 。

✧ 傅立叶级数变化

由于需要大量的计算，所以没有硬件来实现计算时间将是不可容忍的。

✧ 有限冲击相应（FIR）过滤

简单的N级有限冲击相应过滤式如下：

$yn = a_0x_n + a_1x_{n-1} + a_2x_{n-2} \ldots + a_kx_{n-k}$

✧ 数字滤波

实验中通过在使用硬件乘法器与不使用硬件乘法器的情况下，测试在一个大的循环中做乘法所使用的时间，然后通过控制LED的明灭体现执行乘法所用时间的多少。

2、比较器

实验使用电路如 图 4-2：

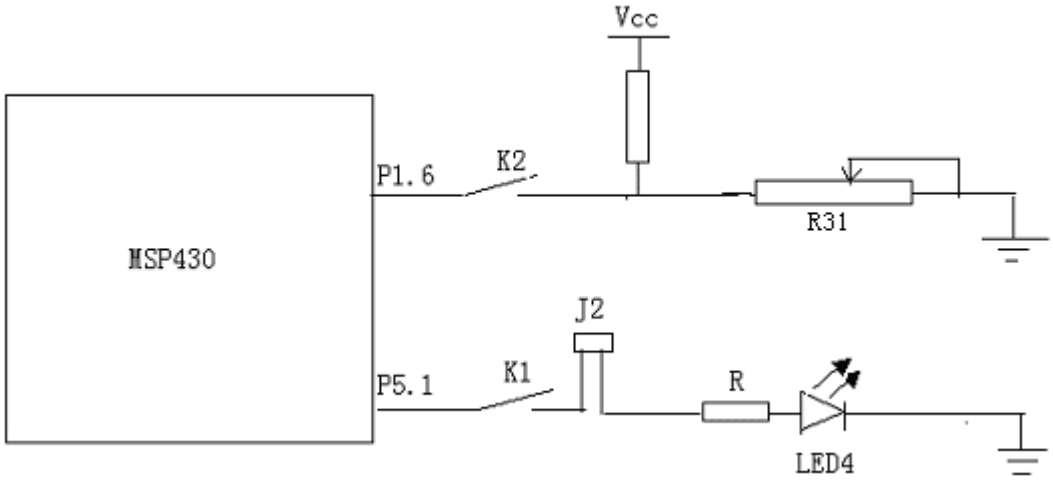


图 4-2 实验电路

其中 P17 的 SW1，P5 的 SW3 设为 ON。PR2 为可变电阻。

MSP430 内部有比较器 A，比较器 A 有两个模拟量输入端 CA0、CA1，一个模拟比较器，一个参考电压发生器和一些控制单元。模拟比较器有正负两个输入端，可以接收 6 种信号：CA0、CA1、0.5Vcc、0.25Vcc、三极管阀值电压、外部参考电源，并可以进行多种组合。比较器 A 多用于模拟量的比较和测试。

实验通过 PR2 的改变，引起 P1.6 端口电压的变化，比较器把 P1.6 端的电压与 MSP430 的内部参考电压 0.25*Vcc 相比较，根据比较结果控制 LED3 的点亮与熄灭，如果 P1.6 端口的电压大于 0.25*Vcc 则点亮 LED，否则熄灭 LED。

（注：该实验完成后，要及时把 P17 的 SW1 设为 OFF）

3、Flash 操作

MSP430 各个系列的芯片都有 Flash，只是容量大小不同，所在的地址空间也不同，但是它们都是有两个信息存储器和 N 个信息存储器，每个信息存储器的大小为 128Byte，主存储器每段 512Byte，信息存储器的地址都是从 1000H 到 10FFH，主存储器的地址都是第零段源于 0FFFFH。Flash 信息存储器 A 段的起始地址是 1080H，结束在 10FFH，信息存储器 B 的起始地址在 1000H，结束地址在 107FH，主存储器第 N 段的起始地址位：起始地址：FE00H－N×512

结束地址：FFFFH－N×512

MSP430F449 系列的 FLASH 大小为 60K，对应地址空间为如图 4-3：

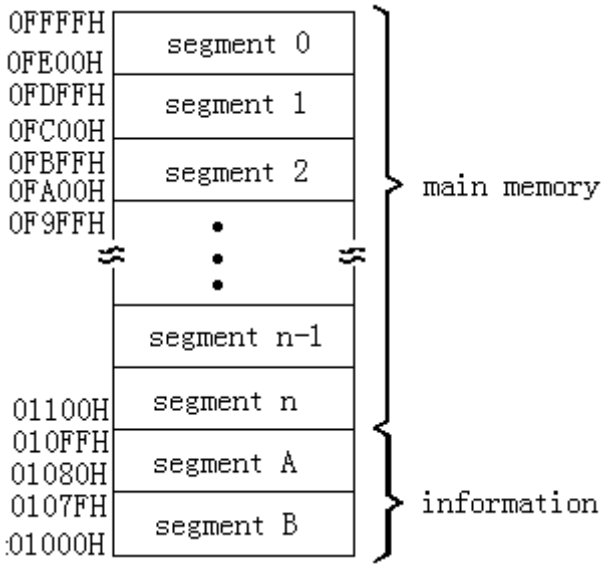


图 4-3 Flash 空间组织

为了正确的读取 FLASH，有必要对 MSP430F449 系列的内存组织做一个了解，图 4-4 为 MSPF449 的内存的组织结构。

MSP430F449 内存组织		
Memory Main: interrupt vector Main: code memory	Size Flash Flash	60KB 0FFFFh – 0FFE0h 0FFFFh – 01100h
Information memory	Size Flash	256 Byte 010FFh – 01000h
Boot memory	Size ROM	1KB 0FFFh – 0C00h
RAM	Size	2KB 09FFh – 0200h
Peripherals	16-bit 8-bit 8-bit SFR	01FFh – 0100h 0FFh – 010h 0Fh – 00h

图 4-4 内存组织

对 Flash 的操作主要有读、写、擦除三种操作，MSP430 的 Flash 操作是通过对控制寄存器 FCTL0、FCTL1、FCTL2 的操作来实现的，控制结构图见图 4-5：

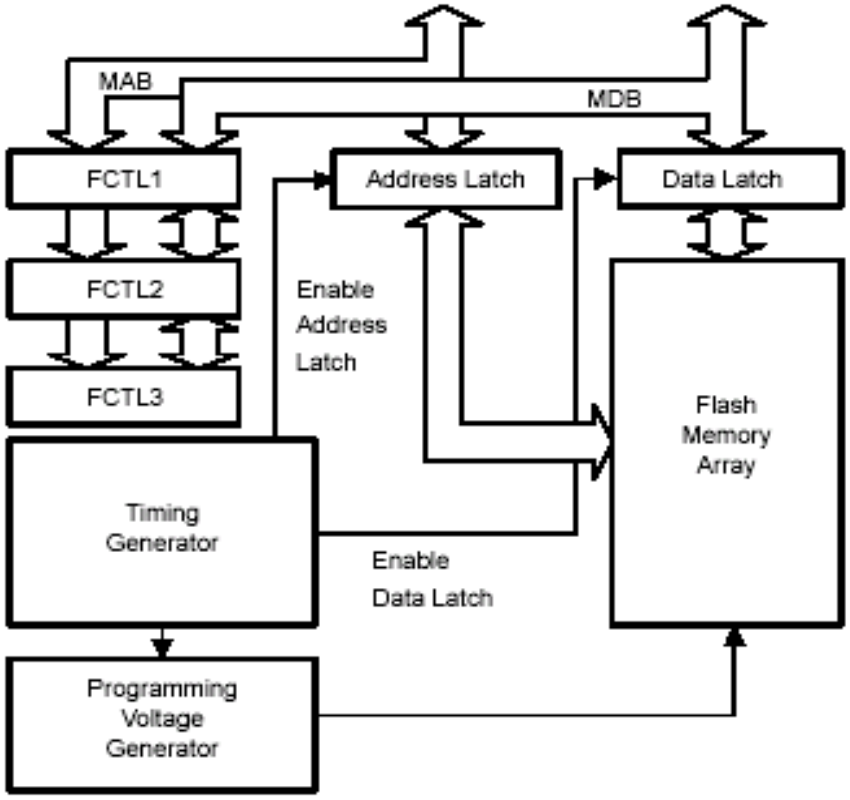


图 4-5 控制结构图

其中，FCTL0 主要控制对 Flash 的编程与擦除，FCTL1 控制 Flash 的时钟发生器的时钟进行定义，FCTL2 是 Flash 操作对应的标识位。

三、实验内容

1、使用硬件乘法器做有符号乘法、无符号乘法

2、比较在执行多次乘法运算时，不使用硬件乘法器与使用硬件乘法器的区别（工程文件”option->General options->target->hardware multiplier ”）

3、利用比较器，比较 P1.6 端口电压与 0.25*Vcc 的大小，根据结构控制 LED4

4、对 Flash 进行读写操作，通过查看寄存器，查看结果是否正确

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON

置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、设置 P11 的 SW6 为 ON，进行乘法器操作

4、设置 DIP 开关 P5 的 SW3、DIP 开关 P17 的 SW1、DIP 开关 P11 的 SW6 为 ON 做比较器实验

5、进行 Flash 读写实验

五、Flash 操作一般流程

1、 Flash 读操作

读操作相对简单，只要判断 FCTL3 的 BUSY 是否为 0 即可，如果为 0 就可进行读操作

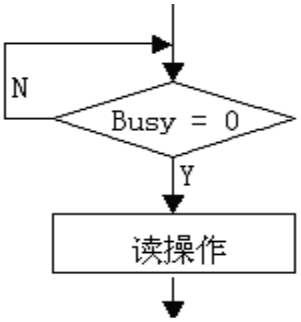


图 4-6　Flash 的读操作

操作流程见图 4-6，为了避免程序在对 flash 擦除过程中，由于中断或者看门狗导致错误，因此开始时关中断和看门狗。Flash 擦除根据操作段数划分为单段擦除和整个擦除两种，在擦除时根据擦除需要设置相应的擦除模式，然后进行盲写（即在要擦除的段进行任意的写操作即可）。操作流程见图 4-7：

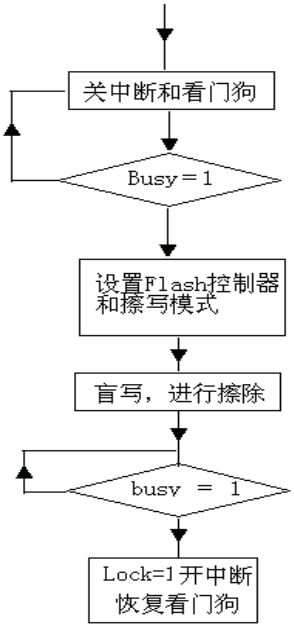


图 4-7 Flash 擦除操作

Flash 编程操作可以分为单个字节写入、多个字节写入、块写入，Flash 编程的一般顺序为：（1）选择时钟和分频因子（2）如果 lock＝1，将它复位（3）中断和看门狗（4）监视 busy 直到 busy＝0　（5）设置 flash 控制寄存器　（6）写操作（7）等待 busy＝1 （8）设置 Lock＝1，开中断、恢复看门狗

如图 4-8：

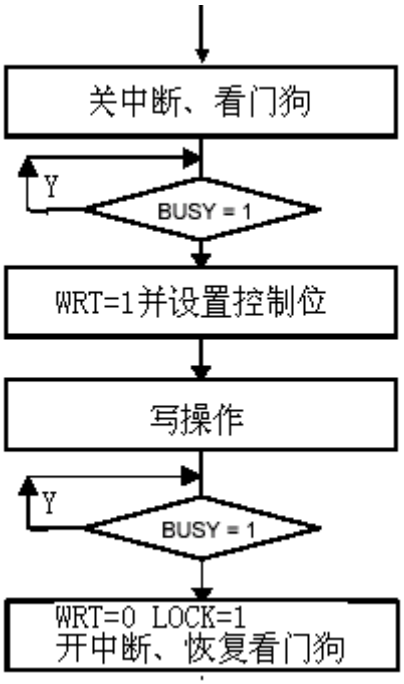


图 4-8 Flash 编程操作

六、分析与思考

1、 在进行 Flash 写操作的时候，很多情况下都要先进行擦除操作，为什么？

2、 试画出操作比较器的流程图

3、 在用循环测试有无硬件乘法器的区别时，把循环写成以下形式，结果会是怎样？实验会测试出差别吗？

```
Int x,y,z;
X = 1000;
Y = 35667;
For(I=0;I<0xFFFF;I++){
   Z=x*y;
}
```

七、实验参考源代码

1. 比较器 A

```
/*****************************************************************************
*  文件名称：
*            cpa.c
* 文件说明；
*          程序使用比较器的 CA0 端接内部电源 Vcc/4，根据 P1.6 口线接收的欠压
*    的电路的电压控制 LED3。程序运行时调节 PR2，观察 LED4
*****************************************************************************/
#include <msp430x44x.h>

/*****************************************************************************
*  light led3
```

```
*******************************************************************************/
void light_LED3(void)
{

  P5DIR |= 0x02;                        //P5.1  输出模式
  P5OUT |= 0x02;                        //P5.1=1
}
/*******************************************************************************
*    熄灭 LED3
*******************************************************************************/
void quench_LED3(void)
{
  P5DIR |= 0x02;                         //P5.1  输出模式
  P5OUT &= 0xfd;                        //P5.1 =0
}
/*******************************************************************************
*   main()  函数
*******************************************************************************/
void main(void)
{
    WDTCTL = WDTHOLD + WDTPW;         //  关看门狗
    CACTL1 = CARSEL + CAREF0 + CAON ; //    Vcc/4 = - cmp
    CACTL2 = P2CA0;                          //  使用  CA0
    quench_LED3();                           //  熄灭 LED
    while(1){

        if((CACTL2 | 0xfe) ==0xff)
        {                                   //  比较电压是否超过 0.25Vcc
          light_LED3();                     //  点亮 LED3
          CACTL1 &= 0xfe;                   // CAIFG = 0
        }else
        {
          quench_LED3();                   //  熄灭 LED3
        }
    }

}
```

2. 片内 Flash 操作

```
/*******************************************************************************
*   文件说明：
*       文件名：main.c
*     程序初始化 Flash 后，首先擦除 flash 的内容，
*     然后写入 write_Buf 的内容，接着读出内容到 read_Buf
```

```
*     程序的执行结果可通过查看 read_Buf 的值确定是否操作正确
*******************************************************************************/

#include <msp430x44x.h>
#define _MSP430F449_ _F449_

#ifndef FLASH_C
#include "Flash.c"
#endif
/*******************************************************************************
*       Main()  函数
*******************************************************************************/
void main(void)
{
   char write_Buf[10] ={                              // Data Buffer to write
          0x00,0x01,0x02,0x03,0x04,
          0x05,0x06,0x07,0x08,0x09
          };
   char read_Buf[10];                                  // Data Buffer be read into
   WDTCTL = WDTHOLD + WDTPW;                           // Stop watchDog
   init_Flash();                                       //  初始化 Flash
   while(1)
   {
       erase_Flash((char*)0x01080);                   //擦除指定段的 Flash 内容
       write_Flash((char*)0x01080,write_Buf,10);      //写入 Write_Buf 中的内容到 Flash
       read_Flash((char*)0x01080,read_Buf,10);        //读出指定位置的内容
   };
}
/*******************************************************************************
*   文件名：
*            flash.c
*   文件说明：
*         对 MSP430 自带 Flash 进行操作
*******************************************************************************/

#define FLASH_C 0
#ifndef MSP430F449_H
#include <msp430x44x.h>
#endif

/*******************************************************************************
*   初始化  Flash
*******************************************************************************/
void init_Flash(void)
```

```
{
    FCTL2 = FWKEY + FSSEL0 + FN0;                    // 设置时钟频率为 ACLK
}
/*******************************************************************************
*    读 Flash
*     输入参数说明:
*              addr:       读地址
*              length:    要读取的字节数
*              readBuf:   用以存储读取内容的缓存区地址
*******************************************************************************/
void read_Flash(char* addr,char * rbuf,int len)
{
    unsigned int cnt;

    while((FCTL3 & 0x01) == 0x0001);              // 等待 flash 空闲
    for(cnt=0;cnt<len;cnt++)
    {
        *(rbuf+ cnt) = *(addr + cnt);             // 读数据
    }
    FCTL3    = FWKEY + LOCK;                      // Lock
}
/*******************************************************************************
*
*   函数功能：写数据到 Flash
*   输入参数:
*              addr:      地址
*              buf:       要写数据的首地址
*              len:       写入的字节数
*******************************************************************************/

void write_Flash(char*addr,char*buf,int len)
{
    unsigned int cnt;
    while((FCTL3 & 0x01) == 0x0001);              // 等待 Flash 空闲
    FCTL3    = FWKEY;                             // 清除“LOCK”标识
    FCTL1    = FWKEY + WRT;                       // 准备写
    for(cnt=0;cnt<len;cnt++)
    {
        *(addr+cnt) = *(buf + cnt);               // 写数据
    }
    FCTL3    = FWKEY + LOCK;                      // Lock

}
/*******************************************************************************
```

```
*
*     Erase Flash
*     intput:
*               add: address that sepecify a Segment
*
*******************************************************************************/
void erase_Flash(char* add)
{
      while((FCTL3 & 0x01) == 0x0001);                  // 等待空闲
      FCTL3    = FWKEY;                                  // 清除 "Lock"
      FCTL1    = FWKEY + ERASE;                          // 准备擦除
      *add     = 2;                                      // 擦除，写任意数均可
      FCTL3    = FWKEY + LOCK;                           // 置“LOCK”
}
```

实验五 使用口线模拟IIC操作

一、实验目的

1. 熟悉 IIC 协议
2. 了解 EEPROM 24LC01B 的操作
3. 掌握口线模拟 IIC 读写 EEPROM 的方法

二、实验原理

1. IIC 协议

IIC（Intel－Integrated Circuit bus）总线是一种由飞利浦公司开发的串行总线，产生于 80 年代，由于其简单有效的特点以及便于对设备进行集中管理，I2C 使用十分广泛。标准 I2C 总线传输速率可以到 100Kbit/s，通过使用了 7 位地址码，就能支持 128 个设备。加强型 I2C 总线用了 10 位地址码（能够支持 1024 个设备），快速模式（400Kbit/s）和高速模式（最高有 3.4Mbit/s）。I2C 总线使用两根信号线来进行数据传输，一根是串行数据线(SDA)，另一根是串行时钟线(SCL)。它允许若干兼容器件共享总线。总线上所有器件要依靠 SDA 发送的地址信号寻址，不需要片选线。任何时刻总线只能由一个主器件控制，各从器件在总线空闲时启动数据传送，由 I2C 总线仲裁来决定哪个主器件控制总线。

I2C 对 SCL 和 SDA 的电平信号进行了定义：

总线空闲：

SCL 和 SDA 都保持高电平。

开始信号：

SCL 保持高电平的状态下，SDA 出现下降沿。出现开始信号以后，总线被认为"忙"。

停止信号：

SCL 保持高电平的状态下，SDA 出现上升沿。停止信号过后，总线被认为"空闲"。

总线忙：

在数据传送开始以后，SCL 为高电平的时候，SDA 的数据必须保持稳定，只有当 SCL 为低电平的时候才允许 SDA 上的数据改变。

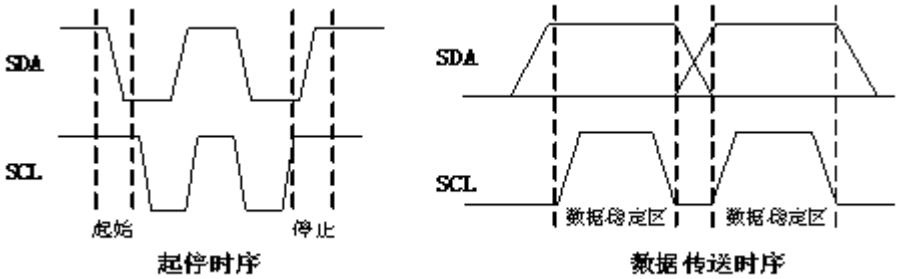


图 5-1 IIC 时序图

I2C 总线的传送格式为主从式，对系统中的某一器件来说有四种可能的工作方式：主发送方式，从发送方式，主接收方式，从接收方式。

⑴ 主发送从接收

主器件产生开始信号以后，发送的第一个字节为控制字节。前七位为从器件的地址片选信号。最低位为数据传送方向位 (高电平表示读从器件，低电平代表写从器件)，然后发送一个选择从器件片内地址的字节，来决定开始读写数据的起始地址。接着再发送数据字节，可以是单字节数据，也可以是一组数据，由主器件来决定。从器件每接收到一个字节以后，都要返回一个应答信号(ASK=0)。主器件在应答时钟周期高电平期间释放 SDA 线，转由从器件控制，从器件在这个时钟周期的高电平期间必须拉低 SDA 线，并使之为稳定的低电平，作为有效的应答信号。

⑵ 从接收主发送

在开始信号以后，主器件向从器件发送控制字节。如果从器件接收到主器件发送来的控制字节中的从地址片选信号与该器件相对应，并且方向位为高电平(R/W=1),就表示从器件将要发送数据。从器件先发送一个应答信号(ASK=0)回应主器件，接着由从器件发送数据到主器件。如果，在这个过程之前，主器件发给从器件一个片内地址选择信号，那么从器件发送的数据就从该地址开始发送；如果在从器件接收到请求发送的控制信号以前，没有收到这个地址选择信号，从器件就从最后一次发送数据的地址开始发送数据。发送数据过程中，主器件每接收到一个字节都要返回一个应答信号 ACK。若 ACK=0(有效应答信号)，那么从器件继续发送；若 ACK=1(停止应答信号)，停止发送。主器件可以控制从器件从什么地址开始发送，发送多少字节。

在向那些只有一个主设备（典型的是主微控制器）的基本系统中不会有仲裁的。然而，更多的复杂系统能够有多个主控设备，因此，就有必要用某种形式的仲裁来避免总线冲突和数据丢失。通过用线与（开路基极）连接 I2C 总线的两路信号（数据和时钟）可以实现仲裁。所有的主设备必须监视 I2C 的数据和时钟线，如果主设备发现已经有传输正在进行，它就不会开始传输了。有很小的几率会产生一下情况：有两个或更多的设备同时发出“开始”信号。在这种情况下，相互竞争的设备自动使它们的时钟保持同步，然后像平常一样继续发射信号。

第一个检测到自己发送的数据和总线上的数据不匹配的设备要失去仲裁能力。这种情况会在这时发生：当前述设备发送一个高电平时，而同时另一个主控设备也正在发送一个低电平。也许直到相互竞争的设备已经传输了许多字节后，仲裁才会完成。

因为没有数据丢失，仲裁处理是不需要一种特殊的仲裁相位的。获得主控权的设备从本质上来说，是不知道它为了总线而和其它设备竞争的

2．24LC01B 的操作

24LC01B芯片是一片EEPROM芯片，使用SDA、SCL两线控制，数据以8位组织。一般SDA打开需要一个上拉电阻。其操作如下：

- 启动与停止

  当SCL 高 ，SDA由高变低时，总线启动

  当SCL 高 ，SDA由低变高时，总线停止

  启动与停止的时序图如图5-2：

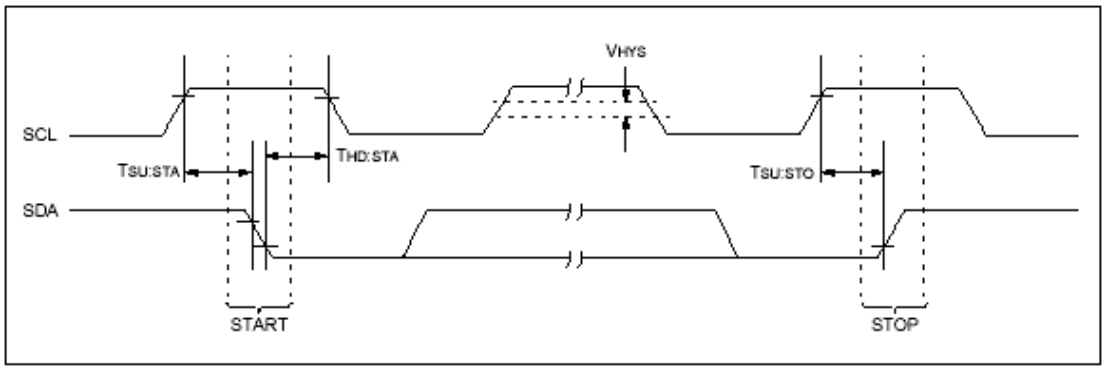


图 5-2　24LC01B 起停时序

- 读操作

  允许访问任何位置的数据，操作过程为：首先发送控制位0xA0，然后发送地址，然后重新启动总线，接着发送控制位0xA1，再接受数据。读取一位的过程见图5-3：

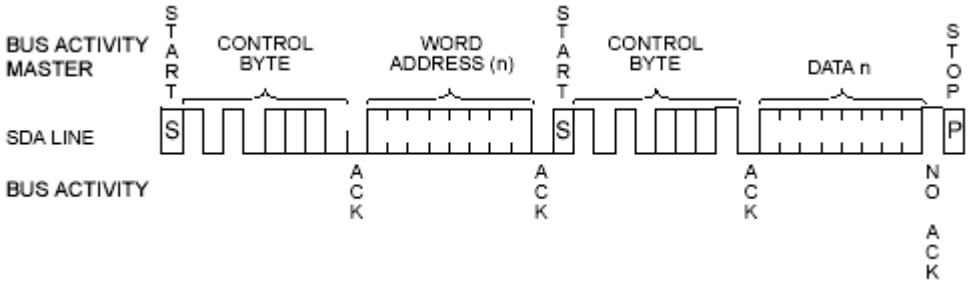


图 5-3 读操作

- 写操作

  操作控制为0xA0，首先开启总线，发送控制位0xA0，然后发送地址，最后写入数据，停止总线。操作过程见图5-4：

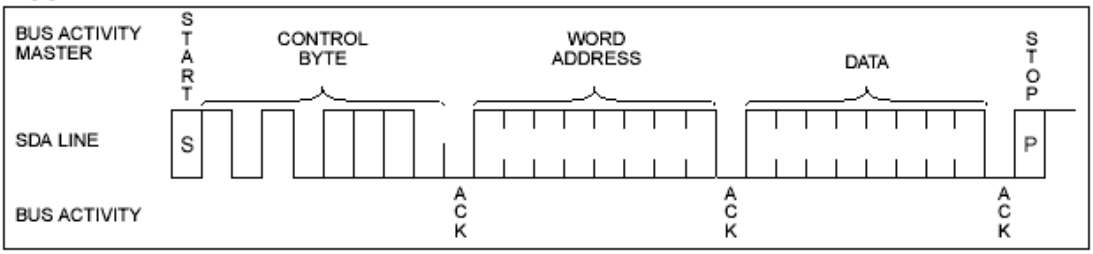


图 5-4 写操作

3. 实验电路

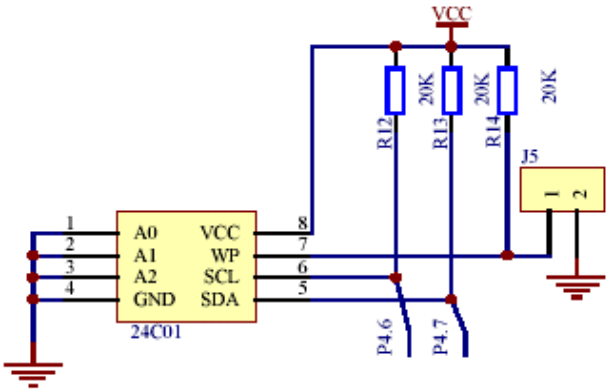


图 5-5 实验电路

三、实验内容

使用口线模拟IIC对EEPROM进行读写操作

要求：

- ✓ 写入数据到EEPROM，并显示该字符到LED
- ✓ 读出写入的数据然后把它显示到LED，比较是否相同

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON

置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、设置下列开关为 ON 做 EEPROM 实验

DIP 开关 P16 的 SW4、SW5，DIP 开关 P3 的 SW1、SW2、SW3、SW4

DIP 开关 P2 的 SW1、SW2、SW3、SW4、SW5，DIP 开关 P4 的 SW6

五、分析与思考

1、如果程序中不使用控制口线方向的方法改变 SDA、SCL 的电平，而是采用口线的输出值来控制，那么程序应该做如何改动？

2、如果程序中在读数据的时候，不是读一个字节结束，而是连续读，那么读出的数据还是你指定地址的数据吗？为什么？

六、实验参考程序

```
/*******************************************************************************
*   文件名称：
*           main.c
*   文件说明：
*           程序通过口线模拟IIC总线操作EEPROM，实验中在
*       EEPROM的地址0x02写数据0x06，然后显示到LED[0],
*       读出的数据显示到LED[1]
*******************************************************************************/

#define MSP430F449_H 0
#include <msp430x44x.h>

#include "IIC.c"
```

```
#ifndef LED_IN_USE
#include "led.c"
#endif

void main()
{
   WDTCTL = 0x5A80;          // 关闭看门狗

   init_LED();               // 初始化LED
   write_Buf = 0x06;         // 设置写的内容
   addr_Buf  = 0x02;         // 设置写地址
   write_EEPROM();           // 写数据到EEPROM
   read_EEPROM();            // 从EEPROM中读去刚才写的 数据

   led_Buf[1] = write_Buf;   // 设置写的数据到LED缓冲
   led_Buf[0] = read_Buf;    // 设置读的内容到LED缓冲
   write_EEPROM();           // 写数据到EEPROM
   read_EEPROM();            // 从EEPROM中读去刚才写的 数据
   while(1)
   {
      led_Display();         // 显示LED
   }
}
/*****************************************************************************
*   文件名称：
*              IIC.c
*       文件说明：
*       使用口线模拟IIC
*****************************************************************************/

#define MSP_IIC 0

#ifndef MSP430F449_H
#include <msp430x44x.h>
#endif

/*****************************************************************************
*   数据定义
*****************************************************************************/
unsigned char read_Buf,  //读缓冲区
          write_Buf,     //写缓冲区
          ctrl_Buf,     //用于存放控制指令等的中间缓冲区
          addr_Buf,  //高地址缓冲
```

```
          ack_Flag;  //应答标识
/*****************************************************************************
*   延迟
*****************************************************************************/
void iic_Delay(void)
{
   _NOP();
   _NOP();
   _NOP();
}
/*****************************************************************************
*   启动IIC
*****************************************************************************/
void start_IIC(void)
{
     P4OUT &= 0x3f; //设置P4OUT
     P4DIR &= 0x7f; //SDA = 1
     iic_Delay();
     P4DIR &= 0xbf; //SCL = 1
     iic_Delay();
     P4DIR |= 0x80; //SDA = 0
     iic_Delay();
     P4DIR |= 0x40; //SCL = 0
     iic_Delay();

}
/*****************************************************************************
*   停止IIC
*****************************************************************************/
void stop_IIC(void)
{
     P4DIR |= 0x80; //SDA = 0
     iic_Delay();
     P4DIR &= 0xbf; //SCL = 1
     iic_Delay();
     P4DIR &= 0x7f; //SDA = 1
     iic_Delay();
     P4DIR |= 0x80; //SDA = 0
     iic_Delay();
     P4DIR |= 0x40; //SCL = 0
}

/*****************************************************************************
*   发送 0
```

```
*******************************************************************************/
void send_Zero(void)
{
     P4DIR |= 0x80; //SDA = 0
     iic_Delay();
     P4DIR &= 0xbf; //SCL = 1
     iic_Delay();
     P4DIR |= 0x40; //SCL = 0
     iic_Delay();

}
/*******************************************************************************
*   发送 1
*******************************************************************************/
void send_One(void)
{
     P4DIR &= 0x7f; //SDA = 1
     iic_Delay();
     P4DIR &= 0xbf; //SCL = 1
     iic_Delay();
     P4DIR |= 0x40; //SCL = 0
     iic_Delay();
     P4DIR |= 0x80; //SDA = 0
     iic_Delay();
}
/*******************************************************************************
*   发送一个字节数据
*******************************************************************************/
void send_Char(void)
{
     unsigned char cnt,tmp=0x80;
     for(cnt=0;cnt<8;cnt++)
     {
        if((ctrl_Buf & tmp )> 0)
        {
             send_One();        // 发送1
        }else
        {
             send_Zero();   // 发送0
        }
        tmp /= 2;       // tmp右移一位
     }
}
```

```
/*******************************************************************************
*   读一个byte数据
*******************************************************************************/
void read_Char(void)
{
     unsigned char cnt,tmp=0x80;
     read_Buf = 0x00;
     for(cnt=0;cnt<8;cnt++)
     {
        P4DIR &= 0x7f;  //SDA = 1
        iic_Delay();
        P4DIR &= 0xbf;  //SCL = 1
        iic_Delay();
        if((P4IN & 0x80) > 0x00)
        {               // 收到 1
            read_Buf |= tmp;
        }
        P4DIR |= 0x40;  //SCL = 0
        iic_Delay();
        tmp = tmp/2;

     }
}

/*******************************************************************************
*   应答信号
*******************************************************************************/
void iic_ACK(void)
{
     ack_Flag = 0x00;
     P4DIR   &= 0x7f;   //SDA = 1
     iic_Delay();
     P4DIR   &= 0xbf;   //SCL = 1
     iic_Delay();
     if ((P4IN & 0x80) == 0x80 )
     {
        ack_Flag = 0x01;
     }
     P4DIR   |=  0x40;  //SCL = 0
     iic_Delay();

}
/*******************************************************************************
*  iic_NACK
```

```
*******************************************************************************/
void iic_NACK(void)
{
     P4DIR   &= 0x7f;   //SDA = 1
     iic_Delay();
     P4DIR   &= 0xbf;   //SCL = 0
     iic_Delay();

     P4DIR   |= 0x40;   //SCL = 0
     iic_Delay();
     P4DIR   |= 0x80;   //SDA = 0
     iic_Delay();
}

/*******************************************************************************
*   写一个数据到EEPROM
*******************************************************************************/
void write_EEPROM(void)
{
     unsigned char step_Flag=0x00;
     while(step_Flag < 0x03)
     {
         if(step_Flag == 0x00)
         {
            start_IIC();       // 启动 I2c
            ctrl_Buf = 0xA0;   // 设置控制位
            send_Char();       // 发送控制位
            iic_ACK();         // 确认
            if (ack_Flag == 0) step_Flag += 1;
     }else if (step_Flag==1)
     {
         ctrl_Buf = addr_Buf;  // 设置地址
         send_Char();          // 发送地址
         iic_ACK();            // 读 确认
         if (ack_Flag == 0)
         {  // 检测是否地址发送成功
             step_Flag += 1;
         }else{
             step_Flag = 0;
         }
      }else{
         ctrl_Buf = write_Buf; // 设置写内容
         send_Char();          // 写
         iic_ACK();            // 读响应信息
```

```
        if (ack_Flag == 0)
        {                       // 检测是否写成功
            step_Flag += 1;
        }else{
            step_Flag = 0;
        }
      }
      }

      stop_IIC();                 // 停止 IIC

}
/*******************************************************************************
*   从EEPROM读数据
*******************************************************************************/
void read_EEPROM()
{
     unsigned char step_Flag=0;
     while(step_Flag < 0x03)
     {
         if(step_Flag == 0x00)
         {
            start_IIC();         //启动 I2c
            ctrl_Buf = 0xa0;    //设置控制位
            send_Char();        //发送控制位
            iic_ACK();          //读取应答
            if (ack_Flag == 0) step_Flag += 1;
     }else if (step_Flag==1){    // 如果控制位发送成功
         ctrl_Buf = addr_Buf;    // 设置读地址
         send_Char();            // 发送地址
         iic_ACK();              // 读响应
         if (ack_Flag == 0)
         {   // 是否地址已经发送
             step_Flag += 1;
         }else{
             step_Flag = 0;
         }
      }else{
         start_IIC();           // 启动 IIC
         ctrl_Buf = 0xa1;       // 设置读模式
         send_Char();           // 发送控制位
         iic_ACK();             // 获取响应
         if (ack_Flag == 0)
         {                      // 是否发送成功
```

```
            step_Flag += 1;
            read_Char();      // 读字符
            iic_NACK();       // 发响应
        }else{
            step_Flag = 0;
        }
    }
    }
    stop_IIC();                 // 停止 IIC
}
```

实验六 同步通讯模块与扩展FLASH

一、实验目的

1. 了解USART 模块同步模式的结构与原理
2. 熟练使用SPI同步通讯模式
3. 掌握扩展串行接口 Data Flash的使用方法

二、实验原理

1、 同步模式（SPI）

在同步模式（SPI）下，主机提供时钟和数据，从机根据主机的时钟保持同步，允许 7 位或者 8 位数据流以内部或者外部确定的速率移入或移出 MSP430，支持 3 线模式和 4 线模式，SPI 可以工作在主机模式也可以工作在从机模式。在主机模式下，USART 模块通过 UCLK 引脚上的 UCLK 信号控制串行通讯，在第一个时钟周期，数据从 SIMO 引脚移出，并在周期中间锁存 SOMI 数据。3 线模式时 STE 输入信号与控制无关，在 4 线模式时，STE 信号被主机用于避免与别的主机发生总线冲突。

实验板上的 SPI 结构，其电路如如图 6-1：

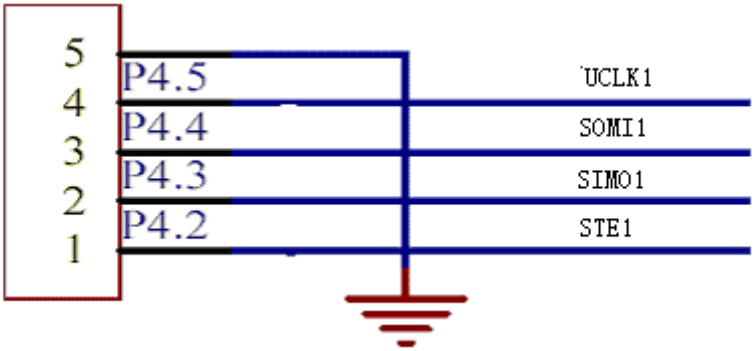
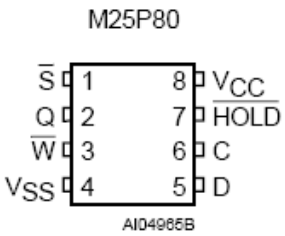


图 6-1 实验板上的 SPI 接口电路

2、扩展 Flash M25P80

M25P80是意法半导体公司推出的8Mbit（1M x 8)大容量串行接口Flash器件，具有先进的保护机制，采用2.7V-3.6V单电源供电。支持使用页编程（Page Program）指令写flash，每次可以写入1-256字节的内容，并可以使用块擦除（Bulk Erase）指令或扇区擦除（Sector Erase）指令对flash的内容进行擦除。

M25P80引脚图以及引脚定义见图6-2：



C	串行时钟
D	串行输入
Q	串行输出
S	芯片片选
$\overline{W}$	写保护
$\overline{HOLD}$	保持
V_{CC}	电源
V_{SS}	接地

图 6-2 M25P80 的芯片图与引脚功能定义

Flash 内存由 16 个扇区，每个扇区包含 256 页。每页 256 字节宽。

M25P80的内存组织结构如图6-3：

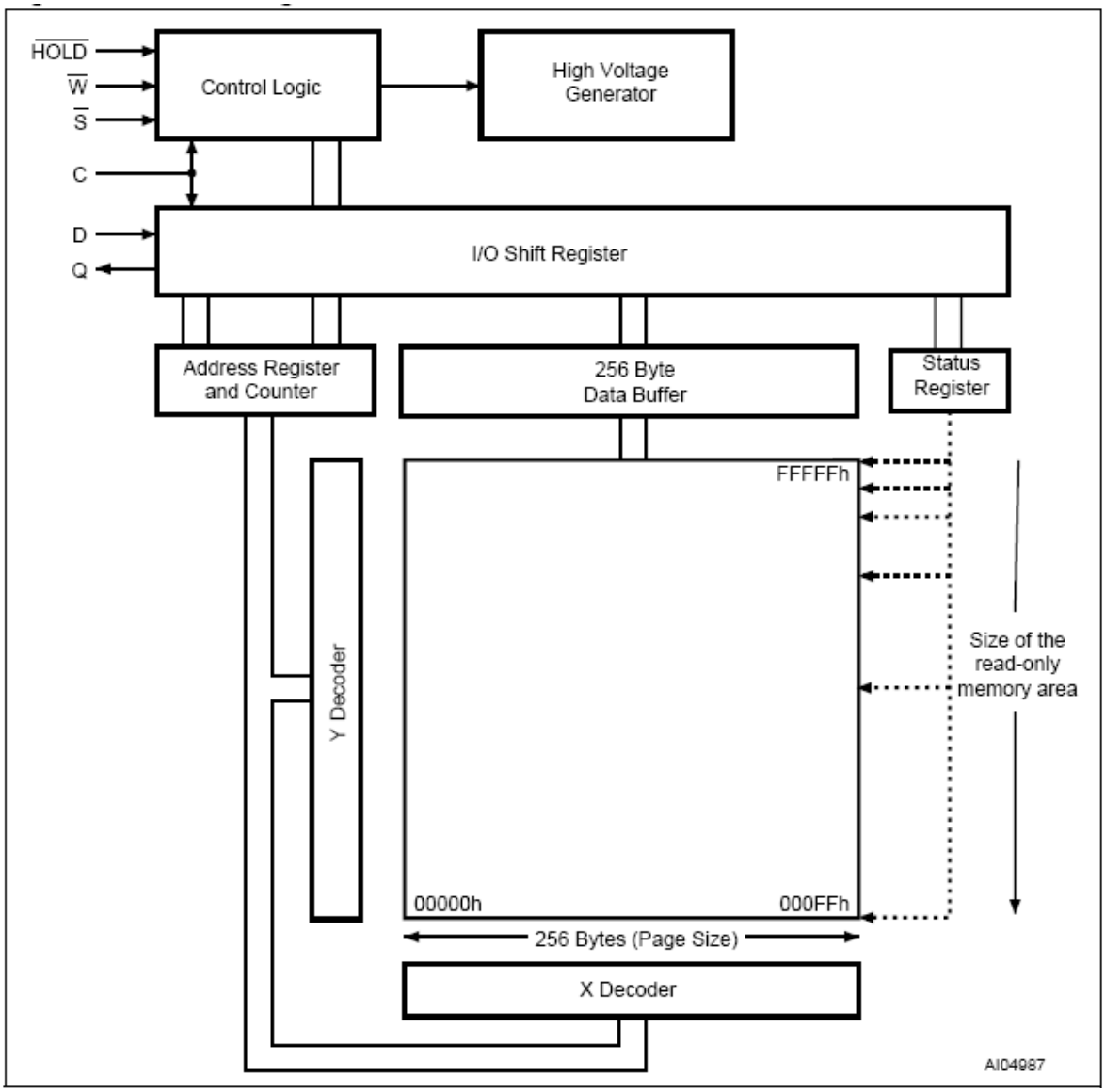


图 6-3 内存组织

M25P80的操作：

所有的指令序列都开始于1个字节的指令码。根据不同的指令，接着可以输入地址字节或者数据，或者不再输入任何数据。芯片片选（/S）被拉低后，串行数据输入引脚在时钟（C）的上升沿时采样数据。数据输出引脚（Q）在时钟的下降沿时从flash中移出数据。芯片操作指令码见表6-1所示。

表6-1 指令码描述

指令	描述	指令码
WREN	写使能	06h
WRDI	写禁止	04h
RDID	读取id信息	9Fh
RDSR	读取状态寄存器信息	05h
WRSR	写状态寄存器	01h
READ	读取数据	03h
FAST_READ	快速读取数据	0Bh
PP	页编程	02h
SE	扇区擦除	D8h
BE	块擦除	C7h
DP	电源关闭	B9h
RES	从电源关闭中释放,并读取电信号	ABh
	从电源关闭中释放	

基本指令操作：

✧ 内存写使能指令（WREN）：

写使能指令设置了状态寄存器中的写使能锁存位(WEL).写使能锁存位必须先于页编程（PP），扇区擦除（SE），块擦除（BE）和写状态寄存器（WRSR）指令而别设定。同时执行该指令前必须把芯片片选位拉低，再发送指令码，最后结束时把片选位拉高。

✧ 写禁止指令（WRDI）：

写禁止指令复位了状态寄存器中的写使能锁存位（WEL）。执行该指令前必须把芯片片选位拉低，再发送指令码，最后结束时把片选位拉高。

在下面情形下写使能锁存位（WEL）会被复位：

i. 上电
ii. 写禁止指令（WRDI）指令执行完毕
iii. 写状态寄存器指令（WRSRM）执行完毕
iv. 页编程指令（PP）执行完毕
v. 扇区擦除指令（SE）执行完毕
vi. 块擦除指令执行（BE）完毕

✧ 读取状态寄存器指令（RDSR）：

读取状态寄存器指令可以任何时间读取状态寄存器信息，甚至是在编程、擦除、写状态寄存器过程中。当这些操作正在进行时，建议在发送新指令至flash设备前检查写进行中（WIP）标志位。状态寄存器格式见图6-4：

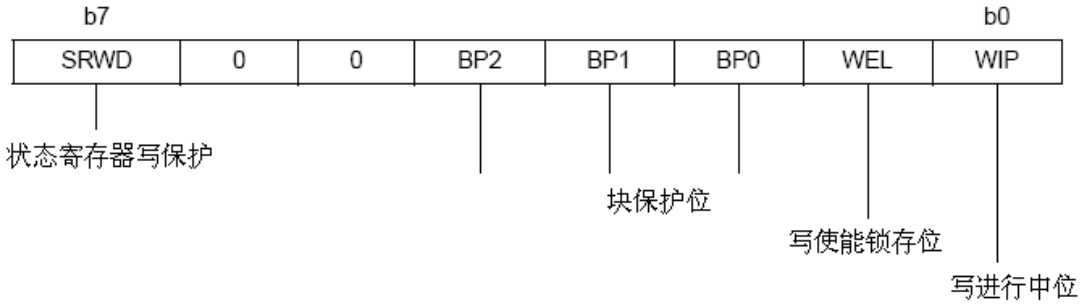


图6-4 状态寄存器格式

✧ 读取数据指令（READ）：

首先把片选位拉低使能芯片。读取数据由指令码并加上3字节地址（A23-A0）的指令序列来完成，且每一位会在时钟的上升沿时被锁存。发送完指令序列后，flash存储体中按地址的设定的内容会在时钟下降沿时被移出至串行数据输出引脚（Q）。

第一个地址可以是任何区域，在数据被读取完后地址会自动增长至下一高地址。整个存储体可以通过一条READ指令被读取。当到达最高地址时，地址会回复至000000h继续读取数据。

可以通过拉高片选位（/S）来终止读取数据指令的执行。

地址位A23-20无意义。

✧ 页编程（PP）

页编程指令允许数据被写入flash存储体中。在执行PP指令前，必须先执行写使能(WREN)指令。指令执行前先拉低片选位，接着输入指令码、3字节地址和至少1个字节的数据。如果8个最低位地址不全为0，在一页被写满后数据会被写入相同页的起始地址处（最低位全为0的地址）。在指令序列执行中片选信号必须始终被拉低。如果写入数据大于256字节（1页的大小），则会覆盖先前已存入flash的数据，而少于256字节则不会对旧数据有任何影响。

片选信号必须在最后一个字节的数据被锁存后拉高，否则页编程指令不会被执行。

✧ 扇区擦除指令（SE）：

扇区擦除指令会把设定的扇区的所有位设置为1（FFh）。在执行SE指令前，必须先执行写使能（WREN）指令。

指令执行前先拉低片选位，接着输入指令码、3字节地址。在扇区内的任何地址都是合法的地址。在指令序列执行中片选信号必须始终被拉低。片选信号必须在最后一个字节的地址被锁存后拉高，否则SE指令不会被执行。

✧ 块擦除（BE）

快擦除指令会把flash中所有位设置为1（FFh）。在执行BE指令前，必须先执行写使能（WREN）指令。指令执行前先拉低片选位，接着输入指令码。在指令序列执行中片选信号必须始终被拉低。片选信号必须在最后一位的指令码被锁存后拉高，否则BE指令不会被执行。

3、扩展Flash实验使用的电路

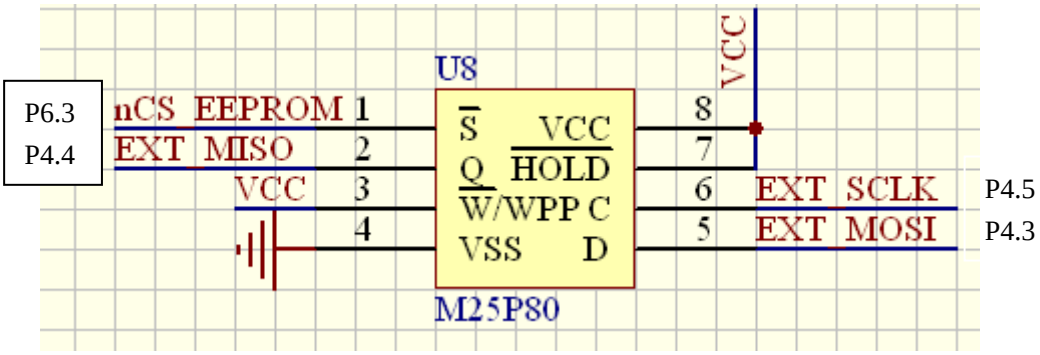


图 6-5 扩展 Flash 电路

三、实验内容

1. SPI通讯

   在两块实验板之间进行数据通讯

2. 扩展Flash操作

   要求：

   - ✓ 写一个数字到M25P80，并把写入的数字显示到LED
   - ✓ 读出写入的数字，并把读出的数字显示到LED

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON

置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、设置 P4 的 SW6，P3 的 SW1、SW2、SW3 为 ON，做 SPI 实验

4、设置 DIP 开关 P18 的全部，DIP 开关 P9 的 SW1、SW2、SW3、SW4 和 p7 的 SW1，P8 的 SW5、SW6，DIP 开关 P2 的 SW1、SW2、SW3、SW4、SW5，DIP 开关 P3 的 SW1、SW2、SW3、SW4，DIP 开关 P4 的 SW6 为 ON，操作扩展 Flash（ M25P80）

五、分析与思考

1、修改 Flash 操作的实验程序，使程序能一次写入一批数据

2、 使用键盘，根据键盘的输入写入数据

六、实验参考代码

1、SPI 实验

```
/*******************************************************************************
*    文件名称：
*
*   文件说明：
*             模块操作 MSP430 的 USART 的 SPI 通讯模块，
*         本实验需要使用外围芯片 HC164、HC165
*         实验从 HC165 读取数据，向 HC164 写数据
*         HC164、HC165 的有关知识请参考 datasheet
*
*******************************************************************************/
```

```
#include   <msp430x44x.h>

/*******************************************************************************
*
*                                  MSP430F449
*                               -----------------
*                           /|\|                 XIN|-
*                            | |                    |       ^          HC164
*               HC165    --|RST               XOUT|-     |   -------------
*             ----------      |                    |       |-|/CLR,B          |   8
*      8   |          /LD|<---|P3.0     SIMO0/P3.1|------>|A               Qx|--\->
*     -\->|A-H      CLK|<---|P3.3/UCLK0 - P3.3|------>|CLK                |
*         |-|INH       QH|--->|P3.2/SOMI0          |       |                   |
*         |-|SER          |      |                    |       |                   |
*         - |              |      |                    |         |                   |
*
*******************************************************************************/

/*****************************    程序初始化    *********************************/
void init_SPI()
{
      P3SEL |= 0x0E;                                  //P1.1~3  置位外围模块
      P3DIR |= 0x01;                                  //P3.0  输出模式
      ME1 |= USPIE0;                                  //  使能  USART0 SPI  模式
      UTCTL0 = CKPH+SSEL1+SSEL0+STC          //  设置  SMCLK  和  3-pin  模式;
      UCTL0    = CHAR + SYNC + MM;              //设置  8-bit  字符模式
      UBR00 = 0x02;                                  //设置波特率
      UBR10 = 0x00;
      UMCTL0 = 0x00;

}

/*******************************************************************************
*
*                转发数据
*
*******************************************************************************/
void forward_Data()
{
       while((IFG1 & UTXIFG0)!= UTXIFG0);        //检测是否 TX 发送缓存 Ready
       P3OUT &= 0xFE;                                  //锁存 HC165 的数据
       P3OUT |= 0x01;
       TXBUF0 = RXBUF0;                                //把 HC165 的数据发到 HC164
}
```

```
void main()
{
      unsigned int tmp;                          //延时变量
      WDTCTL = WDTHOLD + WDTPW;                  //关看门狗
      init_SPI();                                //初始化 SPI
      while(1)
      {
        forward_Data();                          // 交换数据

        for(tmp=0;tmp<0xffff;tmp++);             // 延迟，此间可以设置断点
     }
}
```

2、扩展 Flash 实验

```
/*****************************************************************************
*    文件名称：
*              main.c
*    文件说明：
*              对扩展 FlashM25P80 进行操作，在 Flash 的 0x08
*      位置写 0x08,写入数据显示到 LED[0],读出的数据显示到 LED[1]
*****************************************************************************/
#define MSP430F449_H 0
#include <msp430x44x.h>

#ifndef LED_IN_USE
#include "led.c"
#endif

#include "exflash.c"

/*****************************************************************************
*      main 函数
*****************************************************************************/
void main(void)
{
       char wData=0x03;                          //存放要写的内容

       /****   初始化        ****/
       WDTCTL = WDTHOLD + WDTPW;                 //关闭看门狗
       init_LED();                               //初始化 LED
       init_EXFlash();                           //初始化 Flash
       init_SPI();
       sector_erase(0X00,0X88,0X03);             //擦除扇区
       FLASH_CS(ENABLE);
       Send_Byte(WREN);
```

```
    FLASH_CS(DISABLE);
    FLASH_WaitForLastTask();
    FLASH_CS(ENABLE);
    Send_Byte(PP);                              //页编程
    Send_Byte(0x00);
    Send_Byte(0x88);
    Send_Byte(0x03);
    Send_Byte(wData);
    FLASH_CS(DISABLE);
    FLASH_WaitForLastTask();
    FLASH_CS(ENABLE);
    Send_Byte(READ);                            //读取数据
    Send_Byte(0x00);
    Send_Byte(0x88);
    Send_Byte(0x03);
    read_Buf = Send_Byte(Dummy_Byte);
    FLASH_CS(DISABLE);

    /*************** 把写的内容和读出的内容显示到 LED ***********************/
     while(1)
     {
              led_Buf[0]= wData;
              led_Buf[1] =read_Buf;
              led_Display();                        // 显示到 LED
     };

}
/*****************************************************************************
*   文件名称：
*             exflash.c
*   文件说明：
*           对扩展 FlashAT45DB041 进行读写操作
*
*****************************************************************************
*                    MSP430F449
*                 -----------------
*               |                   |
*               |                   |
*               |                   |    _________
*               |                   |   |         |
*               |              P6.3|-->|    M     |
*               |              P4.3| .  |    2     |
*               |              P4.4| .  |    5     |
*               |              P4.5| ->.|    P     |
```

```
*                |                .        |    8     |
*                |                .        |    0     |
*                |                         |          |
*                                          |_________|
*
*******************************************************************************/

#ifndef MSP430F449_H
#include <msp430x44x.h>
#endif

unsigned char write_Buf,//发送数据的缓存
                read_Buf; //  接收数据的缓存
#define CS 0x08
#define WREN 0X06
#define READ 0X03
#define PP     0x02
#define SE     0XD8
#define BE     0xc7
#define RDSR 0X05
#define Dummy_Byte      0xA5
#define WIP_Flag                0x01        /* Write In Progress (WIP)  标志位  */
#define Write_In_Progress       0x01
#define ENABLE 0
#define DISABLE 1

/*******************************************************************************
*      初始化 AT45DB041B
*******************************************************************************/
void init_EXFlash()
{
  FLL_CTL1 |= SELS + FLL_DIV_1;
  P6DIR|=0X08; // /S
  P6OUT&=0XF7;
}
/*****************************************************************
*   读写期间的时延
*****************************************************************/
void flash_Delay()
{
    _NOP();
    _NOP();
    _NOP();
}
```

```
/*****************************************************************
*    flash enable
*****************************************************************/
 void FLASH_CS(int NewState)
{
   if(NewState == 0)
   {P6DIR |= CS;
            P6OUT &= ~CS;
   }                                                    //cs=0
   else
            P6OUT |= CS;                                //cs=1,去除片选
}

/*************************************************************
*    从 AT45DB041 读一个 Byte
*************************************************************/
unsigned char Read_Byte()
{
   flash_Delay();
   while (!(IFG2 & URXIFG1));                           //等待接收完毕
   read_Buf = RXBUF1;
   return read_Buf;
}
/*****************************************************************
*  发送一个 Byte 到 AT45DB041
*****************************************************************/
unsigned char Send_Byte(unsigned char byte)
{
   while (!(IFG2 & UTXIFG1));                           //等待是否有未完成的发送任务
     flash_Delay();
     TXBUF1 = byte;
     read_Buf = Read_Byte();
     return read_Buf;
}

/*************************************************************
* read status
*************************************************************/
unsigned char FLASH_ReadStatus()
{
   unsigned char FLASH_Status = 0;
   FLASH_CS(ENABLE);

   Send_Byte(RDSR);                                     /*发送读状态寄存器指令  */
```

```
  FLASH_Status = Send_Byte(Dummy_Byte);   /* 发送一个无意义的字符来取得寄存器状态
                                          */
  FLASH_CS(DISABLE);
    return FLASH_Status;
}
/*************************************************************
*   等待任务结束
*************************************************************/
static void FLASH_WaitForLastTask()
{
  unsigned char FLASH_Status = 0;
  /* loop as long as the memory is busy with a write cycle */
  Do                                          /*若 flash 忙碌的话则等待*/
  {
    FLASH_Status = FLASH_ReadStatus();

  } while((FLASH_Status & WIP_Flag) == Write_In_Progress);
}
/*************************************************************
* 块擦除
*************************************************************/
void sector_erase(unsigned char addr1,unsigned char addr2,unsigned char addr3)
{
  FLASH_CS(ENABLE);
 Send_Byte(WREN);                              //发送写使能命令
  FLASH_CS(DISABLE);
  FLASH_WaitForLastTask();                     //等待 flash 空闲
  FLASH_CS(ENABLE);
  Send_Byte(SE);                               //发送扇区擦除指令
  Send_Byte(addr1);                           //发送需要擦除的地址
  Send_Byte(addr2);
  Send_Byte(addr3);
  FLASH_CS(DISABLE);
  FLASH_WaitForLastTask();
}
/*************************************************************
* 全擦除
*************************************************************/
void FLASH_BulkErase(void)
{
  FLASH_CS(ENABLE);
  Send_Byte(WREN);
  FLASH_CS(DISABLE);
  FLASH_WaitForLastTask();
```

```
  FLASH_CS(ENABLE);

  Send_Byte(BE);                                    /* 发送块擦除命令 */
  FLASH_CS(DISABLE);
  FLASH_WaitForLastTask();
}
```

实验七 异步通讯模块

一、实验目的

1．了解USART 模块的结构、原理、功能

2．掌握异步模式的设置，了解地址位多机模式、线路空闲多机模式

3．掌握波特率的设置

4．了解 M_Bus、RS485、RS232 规范

二、实验原理

串行通信只需较少的端口就可以实现单片机和 PC 机的通信，具有无可比拟的优势。串行通信有两种方式：异步模式和同步模式。MSP430F44X 系列都有 USART 模块来实现串行通讯，用户可以根据需要选用同步或者异步通讯模式。下面就 USART 模块的异步模式进行介绍。

在异步模式（UART）下，接受部分自身实现帧的同步，通信双方只需使用相同的波特率即可。异步模式的帧格式由 1 位起始位、7 位或 8 位数据位、校验位、一位地址位、1 或 2 位停止位。在异步模式下，支持两种多机模式：线路空闲多机模式和地址位多机模式。在线路空闲多机模式下，数据块被一段空闲的时间分割。在字符的第一个停止位之后收到 10 个以上的 1，则表示检测到线路空闲，如果采用两个停止位，则第二个停止位被认为是空闲周期的第一个信号。在使用地址位多机模式时，字符包含一个附加的位作为地址标识，数据块的第一个字符带有一个置位的地址位，用以表明该字符是一个地址。

实验使用 MSP430 USART0 模块通过 RS-232 串行口、RS485 串行口、M-bus 来接收或发送数据。

1、RS232 串口通讯

EIA-RS-232 标准最初是为连接调制解调器（Modem）设计的接口标准，由美国电子工业协会（EIA）制定，RS232 连线的方式可以有多种，有三线、六线、八线、两线等，可以根据需要选用。当通讯速率较低时，可以采用简单的三线对接法，只需要使用发送线、接收线、地线，结构十分简单，接线图如图 7-1 所示。RS232 使用串行方式进行通讯，信号电平采用负逻辑，逻辑“1”的电平是 -3V ～ -15V,逻辑 0 的电平为+3V～+15V。

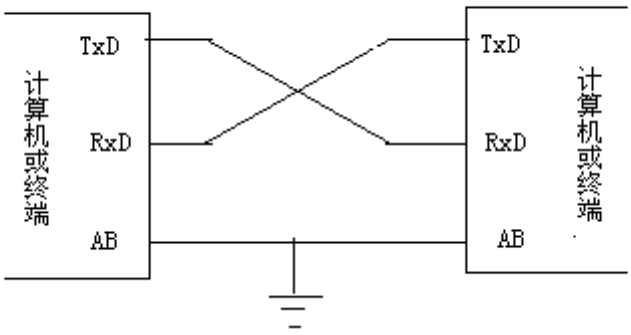


图 7-1 RS－232 电缆连接图

在RS-232实验中，采用一块SN65C3232PWR芯片把从UART0过来的信号进行电平

转换然后输出到计算机或终端，把从计算机或者终端发来的数据发送给UART，实验中使用的RS-232接口的电路图如图7-2所示：

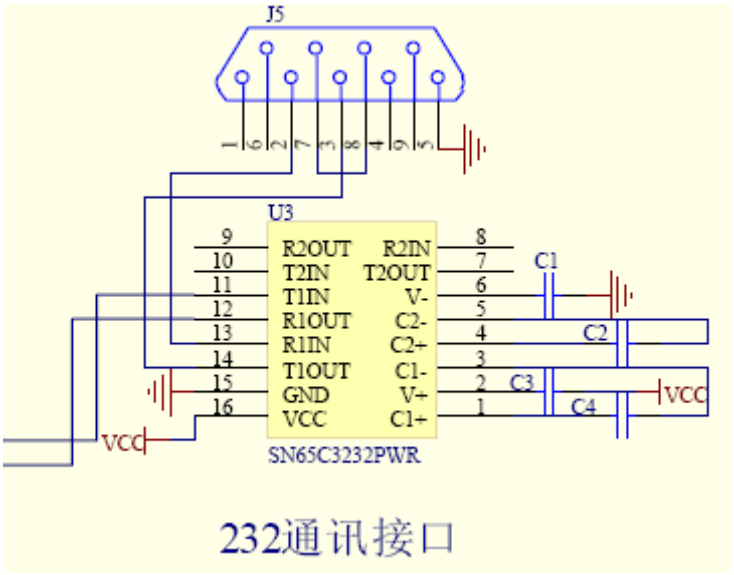


图 7-2 实验使用的 RS-232 接口电路图

在图中我们可以看到，USART0 的TXD 脚与SN65C3232PWR的11 脚（T1IN）相连，USART0 的RXD 脚与SN65C3232PWR的12脚（R1OUT）相连；输入T1IN 的信号转换为RS-232 电平后，经SN65C3232PWR的13 脚（R1IN）输出到接口J5（DB9）的2 脚（DB9 的2 脚为串口的RXD 脚），接口J5（DB9）的3 脚（DB9 的3 脚为串口的TXD 脚）与SN65C3232PWR的14 脚（T1OUT）相连，可以直接通过串口延长线与PC 机相连。

2、RS485通讯

与RS232不同的是，EIA-RS-485是一个使用平衡方式工作的总线，而RS232使用的是非平衡方式。所谓平衡是说RS485总线发送的时候使用两根线（一般为一根双绞线），两根线一个成为A线，一个成为B线。两根线上电压相反，当A线上的电压高于B线时为逻辑1，当B线高于A线时，为逻辑0。

在RS-485实验中,使用MAX485芯片把信号转变成RS485标准需要的电平发送到计算机或者终端，从计算机或终端接收数据然后发给UART0。串口USART0 经过MAX3485，实现RS-485 电平转换的原理图如7-3：

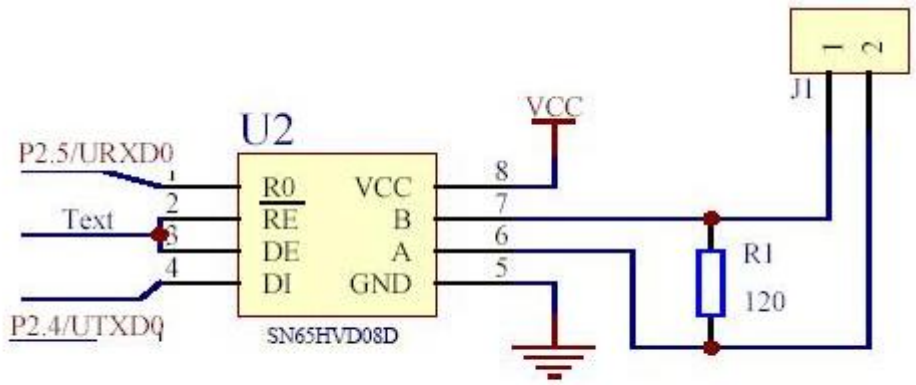


图 7-3 实验中使用的 RS485 接口电路图

图中，MAX3485 的R0（1 脚）与单片机的P2.5/TRXD0 相连，作为通讯电路的数据接收。DI（4 脚）与P2.4/TTXD0相连，作为通讯电路的数据输出。RE 与DE 短接并一同接入P1.1由P4.2 作为MAX3485 的使能端。

MAX3485的内部电路很简单，如图7-4所示：

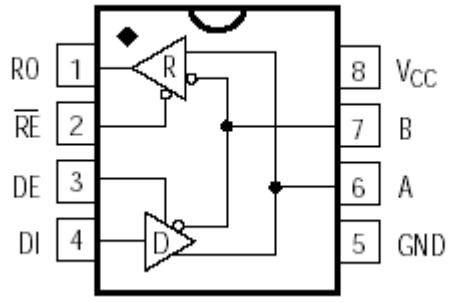


图 7-4 Max3485 内部电路图

图7-5是一个MAX3485的典型操作电路：

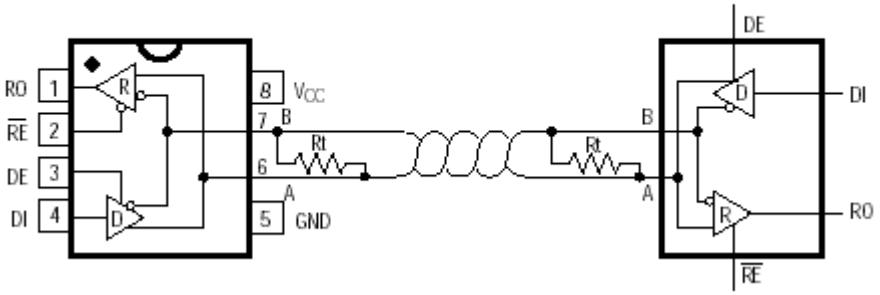


图 7-5 Max3485 的典型操作电路

3、M-Bus 通讯

（1） M_Bus 协议规范

M-bus(Meter Bus) 是由 M-Bus 一个叫做 M-Bus Usergroup 的国际组织指定的，主要用于需要网络和远程读取仪表，这种总线可以远程驱动设备（如用户的仪表等）。另外，在报警系统、加热控制等应用领域也经常用到。

M-Bus 是由 Horst Ziegler 教授与德州仪器基于 ISO-OSI 参考模型共同开发的，以冀望实现一个能够利用其他协议的开放系统。由于 M-Bus 不是一个网络，所以它没有运输层、会话层和表示层，只有物理层、数据链路层、网络层和应用层。另外，根据 OSI 参考模型，高层不允许修改地层的参数，比如波特率等，因此 M-Bus 模型增加了一个管理层（Mangement Layer），其图示表示如图 7-6：

MANAGEMENT LAYER	
Application Layer	
Network Layer (if address = 253)	Address 253 / Enable Disable CI=$52/$56
Data Link Layer	
Physical Layer	Address 254 (255)

图 7-6 M-Bus 协议栈示意图

从图中，我们可以看出，地址 254、255 是为管理物理层保留的，地址 253 为管理网络层保留。由于增加了管理层，它可以管理直接管理 M-Bus 的每一层，所以这一点与 OSI 模型是不一致的。由于我们所做的实验只关系到物理层，没有对其他的高层进行操作，所以我们只是对物理层的知识进行简单介绍。

M-Bus 是一个分级系统，包括一个 Master（Central Allocation Logic）和若干 Slave，由 2 线连接，Master 控制通讯，Slaves 并行地连接在总线上，参见图 7-7：

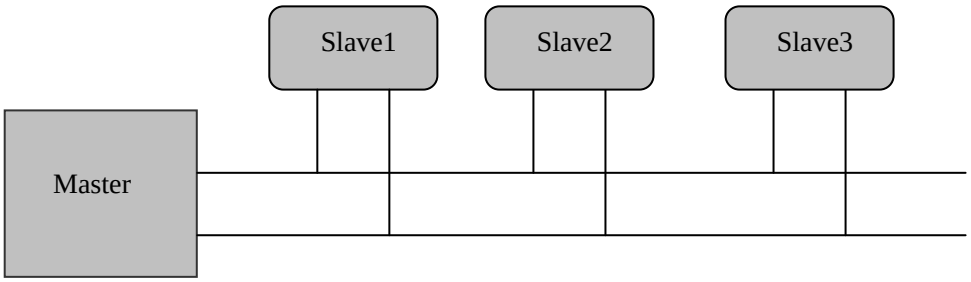


图 7-7 M-Bus 连接

M-Bus 采用串行方式传输数据，。从 Master 发往 Slave 的数据中，利用传送电压地变化来发送“0”和“1”。一个逻辑的“1”（Mark）对应于总线上的一个正常电压（一般为+36V），而逻辑的“0”对应于总线上的+12～24V 的电压。如图 7-8 所示。

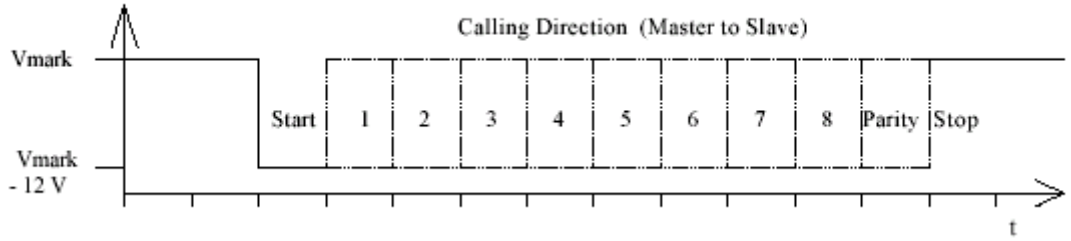


图 7-8 从 Master 到 Slave 的数据传输

从 Slave 发送到 Master 的数据中，采用调制电流的方法来传送数据。一个逻辑的“1”对应于一个恒定的最多 1.5mA 的电流，“0”用增加的电流 11～20mA 来表示。逻辑 1 状态可以用来驱动自身或者接口电路。图 7-9 是数据位发送的图示。

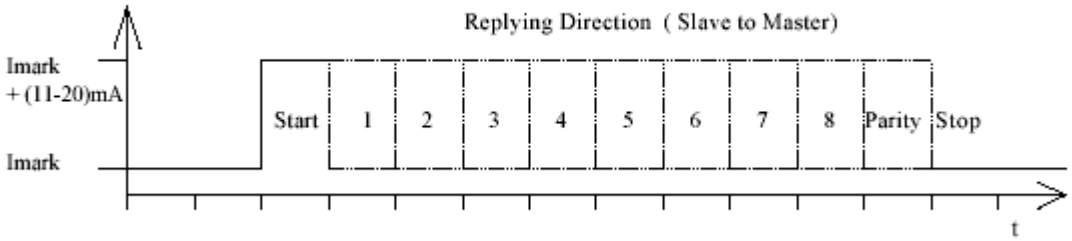


图 7-9 Slave 到 Master 端的数据传输

由于 Master 与 Slave 之间的距离、传输线上的电阻等原因，导致在 slave 端的 MARK 电压小于 36V，因此 Slave 就不是检测＋36V 的电压而是检测一个 12V 的电压衰减。同样在 Master 端只是检测电流是否大于 11mA。为了达到这些要求，M-Bus 在 Slave 端使用 TSS721 这种接口电路，TSS721 的电路图如图 7-10：

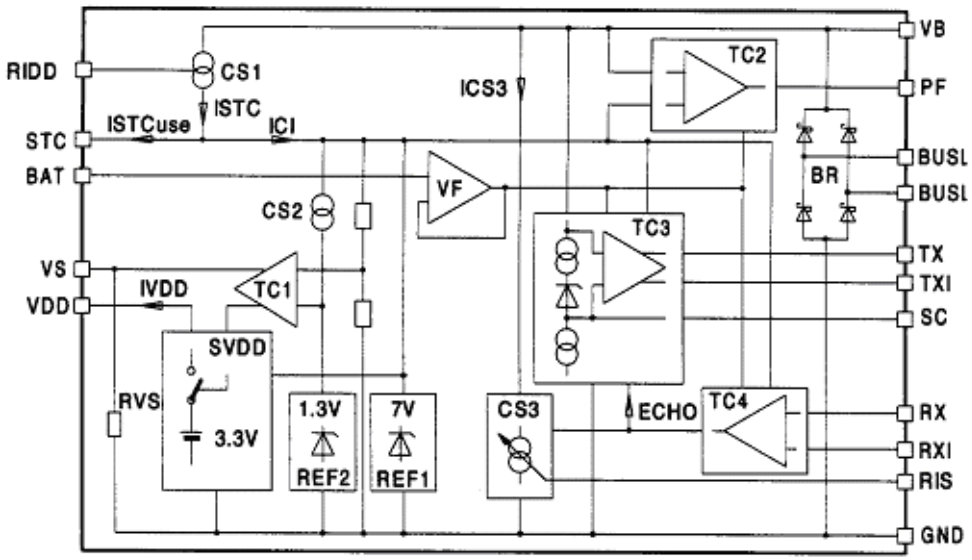


图 7-10　TSS721 模块图

TSS721 这种电路的使用，不但降低了 M-Bus 系统的复杂性，也减少了 Slave 端的开销。除了传输和接收数据之外，TSS721 还用来提供 Slave 端处理器的操作电压。下面就 TSS721 接收数据、发送数据、供电的实现进行说明：

- 从 M-Bus 总线上接收数据

  比较电路TC3检测从主机来的信号，通过电容SC它调整自身到Mark电平（Mark 的定义参见前面介绍），电容SC在Mark状态时充电，电压可以达到8.6V，在Space状态时SC发电。 放电与充电的比率大于30，这样就可保证任何UART协议独立于传输内容正常工作，SC两端的电压可以使比较器动态匹配Mark电平。从放电电流和充电电流的关系，我们可以看出为什么协议要求第十一位必须为逻辑1（Mark），因为这样可以保证SC不会放电太多而导致Mark 电平失效。在Mark电平时，TSS721如果检测到一个7.9V的电压，就输出一个逻辑0到TX端，发送一个翻转电压到TXI。

- 发送数据到 M-Bus 总线

  来自处理器端的数据在 RX 或 RXI 被 TC4、恒流源 CS3 转化成电流，如果在输入端 RX（或者 RXI）是 Mark，TSS721 将从总线上取得静止电流（概念见前面），如果处理器发送的是一个 Space，TC4 切换到恒流源 CS3 上，产生附加电流。静止电流可以通过电阻 Ridd 来调节。

- 对 Slave 端处理器供电

  为了对处理器供电，TSS721 在它的 VDD 端提供一个 3.3V 的电压，通过使用电容 STC 来满足脉冲电流的的输出。当在总线上有连接建立时，STC 充电，VDD 端在 V_{STC}=6V 的时候开始供电。TSS721 在存储数据时将在 PF 端产生电源失效信号，在保存数据期间，有 STC 上存储的电来驱动。为了防止总线失效，TSS721 允许在 VDD 端连接一个电池。图 7-11 是三中可能的 TSS721 操作模式：

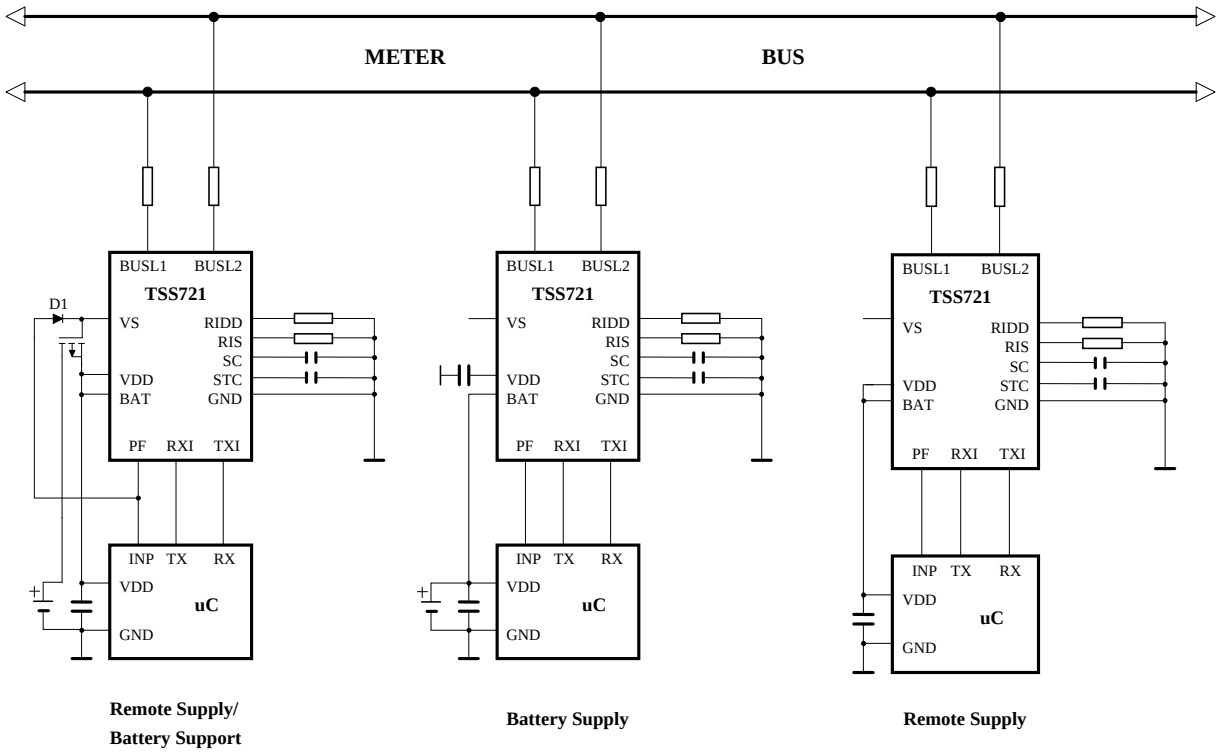


图 7-11　TSS721 的三种连接

图中，处理器可以单独的使用远程供电，可以只使用电池供电，也可以使用二者的结合。图 7-12 是一个应用的典型电路图：

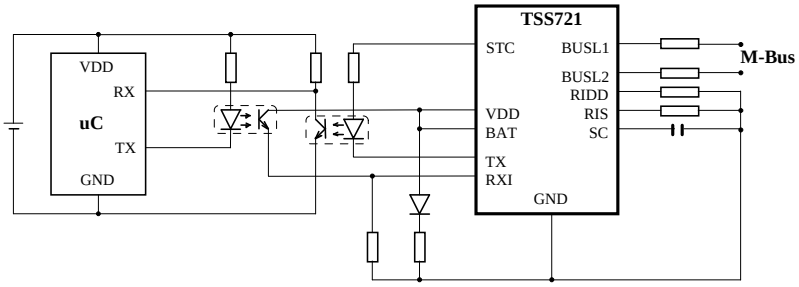


图 7-12　TSS721 典型电路

M-Bus 要求工作在 300～9600 波特，通过使用起始位和停止位实现数据同步。在一个数据报内部不能有暂停状态，即使是在停止位之后，因为在传输线上没有变化对应于“Mark”，起始位必须是 Space，停止位是 Mark。在 8 个数据位传输后是一个偶校验位，然后是停止位。

Master 与 Slave 的通讯如图 7-13：

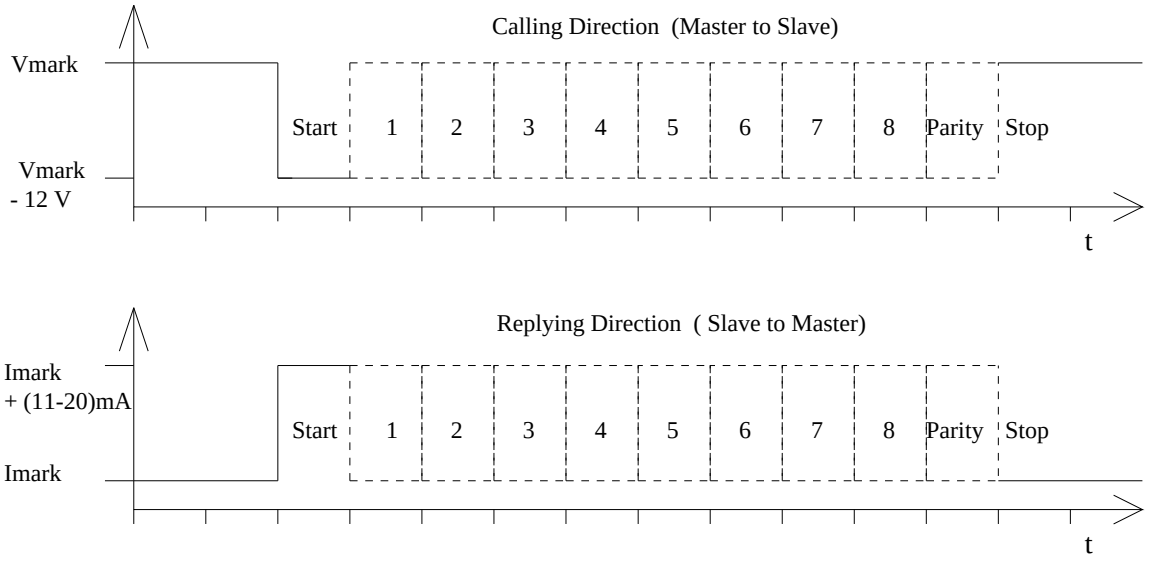


图 7-13　M-Bus 数据传输

（2） 实验使用电路

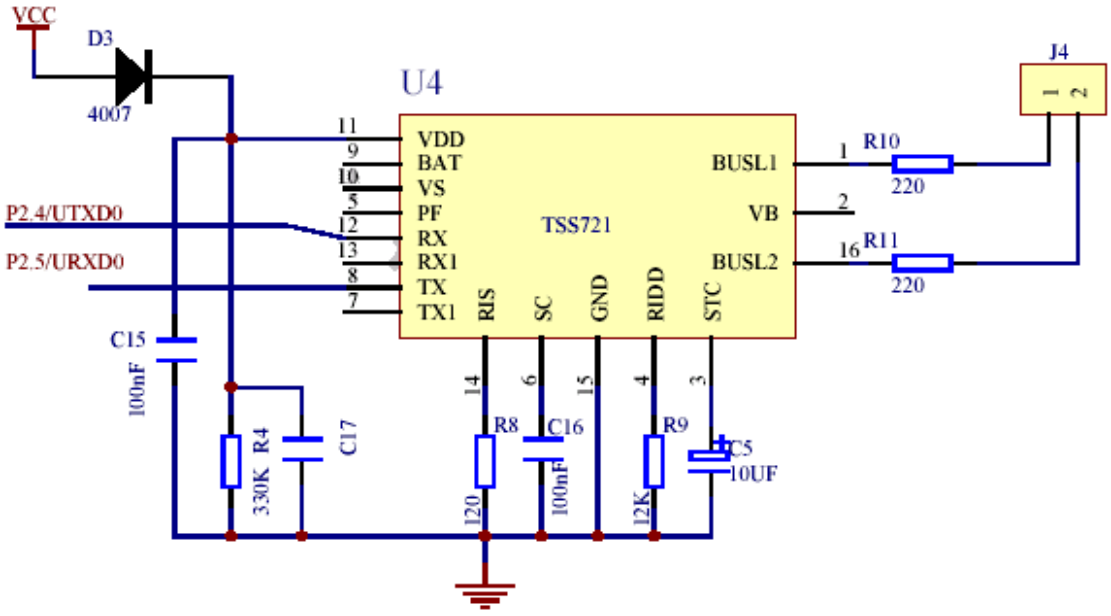


图 7-14　M-Bus 通讯电路

由于实验板上只有 M-Bus 系统的 Slave 端接口，所以要做 M-Bus 实验还需要一个 Master 端。Master 端的要求如下：

- M-Bus 的 Mark 电压：（24V ＋ U_r ）～42V，其中 U_r 为最大压降，定义为最小的 Space 电压减去 12V
- 最低波特率 300bps
- 完成电平转换，能够进行位传输

实验需要程序和 RS232 实验使用程序是相同的。

三、实验内容

1. 使用 RS232 串口与 PC 机进行通讯

要求：

- ✓ 把 PC 机发送过来的 ASC 码字符，显示到 LED
- ✓ 通过行列扫描键盘发送数字到 PC

2. M-Bus 通讯

要求：

- ✓ 实现 M-Bus 的 Master 端
- ✓ 通过 M-Bus 总线，发送、接收数据

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2、SW3、SW4 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON，置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、232 串口通讯

置 DIP 开关 P4 的 SW2、SW3，DIP 开关 J_USART 的 SW2、SW5 为 ON

4、M-Bus 串口通讯

置 DIP 开关 P4 的 SW2、SW3，DIP 开关 DIP，DIP 开关 SW_USART 的 SW3、SW6 为 ON

五、分析与思考

1. 在使用 XT2 的情况下，设置波特率为 19200，重做 RS232 通讯实验
2. 目前 RS232 程序不支持发送字母按键（key A~key F）,试着改写程序，使 LED 能显示字母。
3. 在不使用 XT2 的情况下，设置波特率为 2400，做 RS485 通讯实验

六、实验参考代码

RS232 通讯实验（一些模块的代码在前面实验中已经给出）

```
/*******************************************************************************
*    文件名称：main.c
*    文件说明：
*              使用 LED 显示接收到的数据，使用 LCD 显示发送的字符
*              程序使用波特率为 9600，程序运行时需要在 pc 机上使用一个串口
*            接收发送程序，发送字符应为 asc 码，接收的字符为十六进制
*******************************************************************************/

#define MSP430F449_H 0
#include <msp430x44x.h>

#ifndef LED_IN_USE
#include "led.c"
#endif

#ifndef UART_BUF_SIZE
#include "uart.c"
#endif
```

```
#ifndef KEY_BOARD
#include "keyboard12.c"
#endif
void init_BT(void);
/*****************************************************************************
*     main()  函数
*****************************************************************************/

void main()
{
        char tmp,tt;
        WDTCTL = WDTHOLD + WDTPW;           //关闭看门狗
        init_Keyboard();                    //初始化键盘
        init_LED();                         //初始化 LED
        init_BT();                          //初始化 Basic Timer
        init_UART();                        //初始化 UART0
        _EINT();                            //开中断，允许接收中断
        while(1)
        {
            key_Event();                    //检测键盘事件
            if (key_Flag==1)
            {
                if (uart_TNum == 0)         //有按键
                {
                    uart_TBuf[0] = key_val ;        //如果 UART 空闲，发送数据
                }
                uart_TNum = 1;              // 设定发送数量
                uart_Start();               // 开启 uart 发送
                key_Flag=0;                 //清除按键标识
           }
         if(uart_RFlag>0)
         {
            for(tmp=0;tmp < LED_IN_USE;tmp++)
            {
                tt = uart_RDataPos + tmp - LED_IN_USE + UART_BUF_SIZE;
                // 填充数据到 LED 缓冲区
                if(tt >= UART_BUF_SIZE)
                {
                    tt-= UART_BUF_SIZE;
                }
                led_Buf[tmp] = uart_RBuf[tt];
             }
            uart_RFlag = 0;//清除标识
        }
```

```
    }
}
/*******************************************************************************
*       Initiate basic timer 1
*
*******************************************************************************/
 void    init_BT(void)
 {
     BTCTL    =0x16;                                    //    Basic Timer 1  中断频率设置
     IE2      |= 0x80;                                  //  使能  basic timer  中断
 }
 /*******************************************************************************
 * Basic Timer  中断向量
 *******************************************************************************/
 #pragma vector = BASICTIMER_VECTOR
 __interrupt void    BT_Interrupt(void)
 {
        led_Display();                                 //更新 LED 内容
 }


/*******************************************************************************
*  文件名称：uart.c
*  文件说明：
*               RS232 通迅,使用的 UART0  模块
*******************************************************************************/
#ifndef MSP430XF449_H
#include <msp430x44x.h>
#endif

#define UART_BUF_SIZE 6
/*******************************************************************************/
/*     数据定义                                                                */
/*******************************************************************************/
char uart_RBuf[UART_BUF_SIZE];     //接收缓冲区
char uart_TBuf[UART_BUF_SIZE];     //发送缓冲区
unsigned char uart_RDataPos,        //用于指示下一个存放接收数据的缓冲区位置
                    uart_RFlag,             // 接收缓冲区缓存的数据数目（单位字符）
                    uart_TNum,              // 发送缓冲区缓存的数据数目（单位字符）
                    uart_TPos;              // 标识  uart  下一个要发送的数据的位置
/*******************************************************************************
*       模块初始化
```

```
*******************************************************************************/
void init_UART(void)
{
    unsigned char tmpv;
    FLL_CTL0 &= 0xbf;
    UCTL0 |=SWRST;
    UCTL0 |=CHAR;                                   // 8-bit 字符
  /*  UTCTL0= 0x10;                                 // UCLK=ACLK
    UBR00 = 0x0d;                                   // 在 32768 下进行 2400 波特率通信
    UBR10 = 0x00;                                   // 在 32768 下进行 2400 波特率通信
    UMCTL0= 0x57;                                   // 调整寄存器
  */

    FLL_CTL1|=SELS+XT2OFF+SELM_XT2;             //开启第二个振荡器
    do
    {
      IFG1 &= ~OFIFG;                               // 清除 OSCFault 标志
      for(tmpv = 0xff;tmpv > 0;tmpv--);       //
    }while ((FLL_CTL0&XT2OF) == XT2OF );      // 第二个振荡器是否正常工作
    UCTL0|=SWRST;
    UCTL0|=CHAR;                                    // 8-bit 字符
    UTCTL0=SSEL0+SSEL1;                             // UCLK=SMCLK
    //UBR00=0xa0;                                   // 在 4MHz 下进行 9600 波特率通信
    //UBR10=0x01;                                   // 在 4MHz 下进行 9600 波特率通信
    //UMCTL0=0x5e;                                  // 调整寄存器
    UBR00=0x87;                                     // 在 6MHz 下进行 9600 波特率通信
    UBR10=0x02;                                     // 在 6MHz 下进行 9600 波特率通信
    UMCTL0=0x03;                                    // 调整寄存器
    UCTL0&=~SWRST;

    ME1|= (UTXE0 + URXE0);                          // 使能 USART0 TXD/RXD
    IE1|= URXIE0 ;
    IFG1 = 0x00;
    P2SEL |= 0x30;                                  // P2.4,P2.5 = USART0 TXD/RXD
    P2DIR |= 0x10;
    uart_RDataPos = 0;
    uart_TNum =0 ;
    for(tmpv=0;tmpv<UART_BUF_SIZE;tmpv++)
    {
        uart_RBuf[tmpv] = 0;
    }
}/*******************************************************************************
*
*       数据发送
```

```
*******************************************************************************/
void uart_Start(void)
{
    IE1 |= UTXIE0 ;
    while((UTCTL0 & 0x01 )!=0x01);          //等待直到没有数据发送
    TXBUF0 = uart_TBuf[0];                   //发送数据
    uart_TPos = 1;
}

/*******************************************************************************
*       数据接收中断
*******************************************************************************/
#pragma vector = UART0RX_VECTOR
__interrupt void data_Receive(void)          // UART 接收中断
{
    uart_RBuf[uart_RDataPos]=RXBUF0-48;   //从 asc 码转变到单片机键码索引
                                          //从 asc 码转变到单片机键码索引
    uart_RDataPos = (uart_RDataPos + 1);    //移动接收缓冲区指针
    if (uart_RDataPos >= UART_BUF_SIZE)
    {
        uart_RDataPos = 0;
    }
    uart_RFlag += 1;                        //接收数据计数器加 1
}
#pragma vector = UART0TX_VECTOR
__interrupt void __uart_Send(void)
{
    uart_TNum -= 1;
    if (uart_TNum >0)
    {
        TXBUF0 = uart_TBuf[uart_TPos];
        uart_TPos +=1;
    }else {
       IE1 &= 0x7f;                         //disable UTXIE0
    }
}
/*************************************************************
*   keyboard12.c 文件
*   key_Event()，检测键盘是否有键按下，如果有获取键值
*************************************************************/
void key_Event(void){
      IE2     &= ~0x80;          /*必须关闭定时器 BT，否则 led 的对 P3 口线的操作会产生
                                   uart 发送无意义的数据*/
      unsigned char tmp;
```

```
    P3OUT &= 0x0F;                  // 设置 P3OUT 输出值
    P3OUT = 0xFF;
    P3DIR &= 0xF0;   // P3.0~P3.3 设置为输入模式
    P3DIR |= 0xF0;   // set p3.4~p3.7 设置为输出模式
    P3OUT &= 0x0F;                  // 设置 P3OUT 输出值

    tmp = P3IN;                     // 获取 p3IN
    if ((key_Pressed ==0x00)&&((tmp & 0x0F) < 0x0F))
    {                               //是否有键按下
        key_Pressed = 1;    // 如果有按键按下，设置 key_Pressed 标识
        delay();                    //消除抖动
        delay();
        delay();
        check_Key();            // 调用 check_Key(),获取键值
    }else if ((key_Pressed ==1)&&((tmp & 0x0F) == 0x0F))
    {                               //是否按键已经释放
        key_Pressed = 0;    // 清除 key_Pressed 标识
        key_Flag      = 1;    // 设置 key_Flag 标识
    }
IE2    |= 0x80;                                 // 使能 basic timer 中断，使 led 刷新操作有效
}
```

实验八 ADC与LCD

一、实验目的

1、掌握段式 LCD 的工作原理和显示编程

2、 掌握 ADC 工作的原理，熟练应用 ADC12 的四种工作模式

3、 熟练运用 MSP430 的 LCD 和 ADC 模块

二、实验原理

1．段式 LCD 的基本结构和原理

液晶是一种具有规则性分子排列的有机化合物，它即不是固体也不是液体，它是介于固态和液态之间的物质。液晶具有电光效应和偏光的特性，这是它能用于显示的主要原因。目前前液晶显示器可分成三大种类，分别是扭曲向列型（Twisted Nematic；简称 TN）、超扭曲向列型（Super Twisted Nematic 简称 STN）和彩色薄膜型（Thin Film Transistors；简称 TFT）。限于篇幅这里主要介绍扭曲向列性 LCD 的显示原理，其结构如图 8-11：

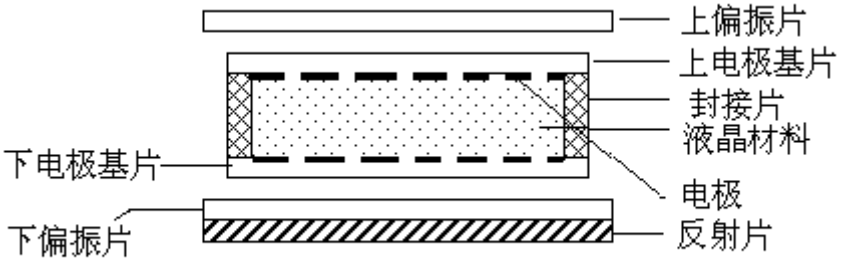


图 8-1 LCD 结构

图中，密封盒中注有扭曲向列型液晶材料，无册透明的玻璃电极排布在上下电极基板的内侧面上。当电板不加电压时液晶材料的内部分子呈 90 度扭曲状态，线形偏振光透过时由液晶分子形成的偏振面也会旋转 90 度，LCD 不会产生显示。当电极两端加上电压时，液晶的扭曲结构在电场作用下消失，线形偏振光透过液晶投射在反射面上，使 LCD 显示。

2．LCD 驱动器的基本原理

液晶在电场作用下很容易分解和失效，因此 LCD 采用交流方波驱动。电路图 8-2 说明了一段的驱动电路：

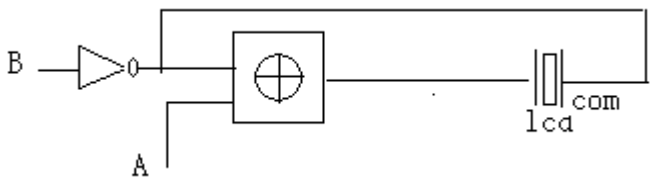


图 8-2 一段的LCD驱动电路

图中 A 为电极信号输入端，B 为交流方波信号输入端，Com 为公共背极信号

显然，如果 A 为 0，A,￢B 异或后，与￢B 相同，LCD 不显示；如果 A 为 1 ，则 LCD 两端产生电压而显示。

段式 LCD 一般用七段或者八段来显示字符，一个七段 LCD 的电路如图 8-3：

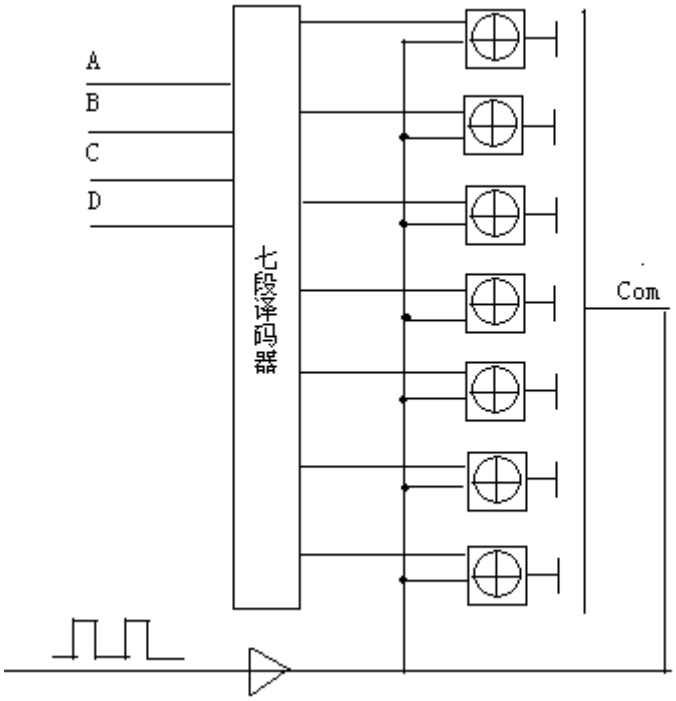


图 8-3 七段LCD

显示电路为静态连接方法，确定是引线较多、电路复杂，故在实际的应用中一般使用动态扫描交流驱动。在多路寻址的动态扫描驱动电路中，液晶字段常采用三分割或者四分割分割方法，驱动电压也有 1/2 、1/3 偏压之分。这种方案实际上是用拓扑方法把液晶字段分割成行列阵列，每位字段的背极分为若干组（如四分法分为 4 组），各组字段的背极对应相连而成为行；字段也划分为若干组，各组相连而成列，同一列上的字段分属不同的背极。行列在显示时分别施加不同的周期性扫描驱动信号，对 1/3 偏压法每拍（半个时钟周期）的周期性扫描驱动信号的电压等级有 Vp，2/3Vp,1/3Vp,0 四组，这样只有扫描到行列交叉处的字段或点阵才会获得 Vp 或者 0 幅度的驱动信号，其余电压小于 LCD 阀值，不会显示。

LCD 模块是 MSP430F4XX 系列自带的模块，根据型号的不同各种芯片驱动能力也不相同，MSP430F449 系列最多可以驱动 160 段。

3． MSP430 温度传感器所测电压和实际温度的关系

V(电压) = 0.00355 * Temp(温度) + 0.986

4．ADC

在计算机应用中，常常需要把模拟量转换为离散的数字信号，以对外部信息进行分析和控制，一般使用一个称为数据采集系统的接口电路完成数据采集转换。数据采集系统由传感器、多路模拟开关 (MUX) 、可编程放大器 (PGA) 、采样－保持单元、模数/数模转换器和数据缓冲与接口电路组成，如图 8-4 所示：

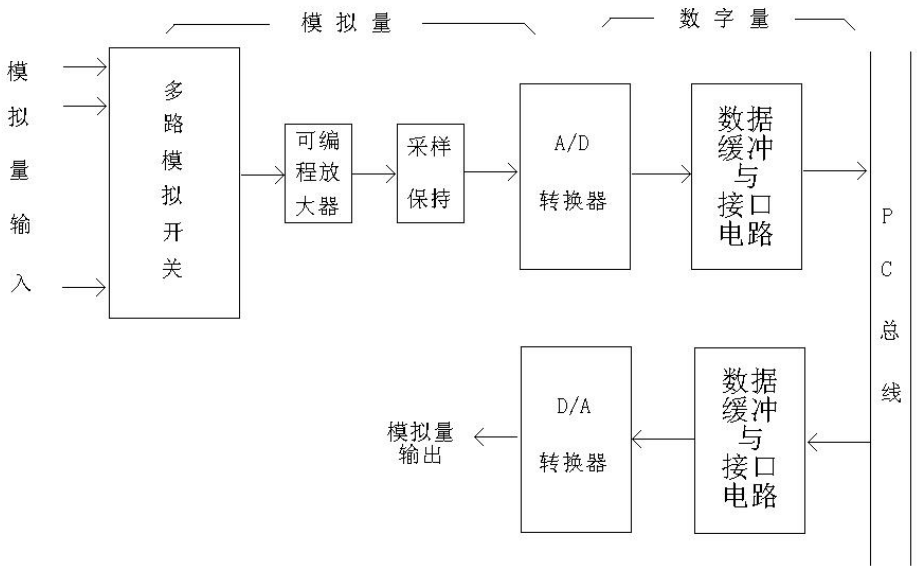


图 8-4 传统数据采集系统

数据采集系统中包含传感器、多路模拟开关 (MUX)、采样－保持电路和模数/数模转换器。模数/数模转换器是数据采样系统的核心部件，担负数字信号和模拟信号的转换任务。A/D 转换器按内部电路的工作原理可分为：计数器 A/D 、逐次比较 A/D 、双积分 A/D 和并行转换 A/D 。

一般数模/模数转换的工作方式有两种：查询方式和中断方式。查询方式就是不断检测，如果有信号输入就进行采样，其实现方法可以是用定时器定时检测，也可以使用循环进行检测。中断方式使用中断进行数据采样。

MSP430 模数转换模块提供四种模式：单通道单次转换、序列通道单次转换、单通道多次转换、序列通道多次转换，这四种方式可以根据需求灵活运用。

单通道单次实现对单一通道的一次采样与转换，转换完成后把采样结果写入选定的寄存器，并把相应的中断标志位置位，如果允许中断则发生中断；若不允许中断可通过软件方法判定标志位是否置位来确定是否要读取指定寄存器的数据。

序列通道单次采样的工作模式和单通道单次模式基本相同，不同的地方是序列通道单次模式可以对多个通道进行一次数据采集。

单通道多次模式实现对一个通道进行多次采样转换，这种方式适用于声音采集之类的需求，可配合中断完成数据的采集。

序列通道多次模式对若干通道进行数据采集转换，直到关闭该功能，适合于一些较为复杂或者要求较高的转换情形。

5．实验电路

实验中是对内部信号进行采样，只是在结果显示的时候才用到外部 LCD 电路，LCD 电路这里不在给出，请参考前面实验中使用的电路。

三、实验内容

1． 使用 LCD 显示数字串“012345”

2． 用 ADC12 对温度进行采样，并把温度用十进制显示到 LCD

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON
置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关
置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

4、 设置下列开关为 ON，进行实验
DIP 开关 P11 的 SW1、SW2、SW3、SW4、SW5、SW6，
DIP 开关 P12 的 SW1、SW2、SW3、Sw4、SW5、SW6、SW7、SW8，
DIP 开关 P13 的 SW5、Sw4
DIP 开关 P1 的 SW4、Sw5、SW6、SW7（com0～com3）

五、分析与思考

1. 实验中使用单通道单次转换方式，如果使用单通道多次采样程序应该如何修改？
2. 为了使 LCD 不闪烁，Basic Timer 的 BTCTL 的 FRFQ1、FRFQ1 是否可以任意设置？说明其原因。

六、实验参考代码

```
/*****************************************************************************
*  文件名称：
*            main.c
*  文件说明：
*       对 MSP430 片内温度传感器进行采样，输出温度到 LCD
*       程序使用单通道单次转换，温度显示的格式是华氏温度,
*        数据带两位小数
*****************************************************************************/
#define MSP430F449_H  0
#include <msp430x44x.h>

#include "adc12.c"

void main(void)
{
    unsigned int tmpv;

    WDTCTL = WDTHOLD + WDTPW;      //停止看门狗
    init_LCD();                    //初始化 LCD
    init_ADC12();                  //初始化 ADC12
    _EINT();                       //使能中断
    while(1)
    {
        start_ADC12();             //启动 ADC12
        while(adc_Flag == 0);      //等待转换完称
        lcd_Display();             //显示数据到 LCD
        lcd_SetRP();               //显示小数点
        tmpv = 0;
        while(tmpv < 0xffff) tmpv++;//延时
   }
```

```
}
/************************************************************
*   文件名称：adc12.c
*   文件说明：adc12 操作
*
************************************************************/
#ifndef MSP430F449_H
#include <msp430x44x.h>
#endif

#ifndef LCD_IN_USE
#include "lcd.c"
#endif

#define REFVOL 2.5                                          //vcc 参考设为 2.5

unsigned char adc_Flag;
/************************************************************************
*
*   初始化 ADC12
************************************************************************/
void init_ADC12(void)
{
     ADC12CTL0   = ADC12ON + REFON + REF2_5V + SHT0_6;// 设置 ADC12 的内部参考电
压 2.5 伏
     ADC12CTL1   = SHP;                                     // 设置使用采样时钟
     ADC12MCTL0  = INCH_10 + SREF_1;                        // 选择通道 A10，即片内温
度传感器输出
     ADC12IE    |= 0x01;                                    // 使能中断
     ADC12CTL0  |= ENC;                                     // 使能转换

}
/************************************************************************
*   启动  ADC12
************************************************************************/
void start_ADC12()
{
     ADC12CTL0 |= ADC12SC;
     adc_Flag   = 0;
}
/************************************************************************
*   把数据编程要显示的格式，然后写到 lcd_Buf 中去
************************************************************************/
void format_Data()
```

```
{
     int result;
     unsigned char tmp;
     result = ADC12MEM0;

     result = ( int)(((REFVOL * result )/ 4096 - 0.986) /0.0000355) //得到对应的温度值*10,以包括小数两位 ;
     for(tmp=0;tmp<7;tmp++)
     {
           lcd_Buf[tmp] = result % 10;                              //把结果转换成十进制,并存放在 LCD 缓冲区中
           result = result /10;                                      //
     }

}
/*****************************************************************************
*   中断向量
*****************************************************************************/
#pragma vector = ADC_VECTOR
 __interrupt void  ADC_Interrupt(void)
{
    format_Data();              //格式化数据并显示到 LCD
    adc_Flag = 1;               //指示有数据要显示
}
/*****************************************************************************
*  文件名称：
*           LCD.c
*   文件说明：LCD 模块
*****************************************************************************/
#ifndef MSP430F449_H
#include <msp430x44x.h>
#endif

#define LCD_IN_USE 8
#define RADIX_POINT 0x08
/*****************************************************************************
   数据定义
*****************************************************************************/
const unsigned char NUM_LCD_ABCD[10]={ 0xf0, 0x60, 0xb0, 0xf0, 0x60,  //'0'~'4',驱动 abcd 段
                                       0xd0, 0xd0, 0x70, 0xf0, 0xf0  //'5' ~ '9'
                                                                    };
const unsigned char NUM_LCD_FGE[10]={ 0x05, 0x00, 0x06, 0x02, 0x03,  //'0'~'4',驱动 efg 段
```

```
                                0x03, 0x07, 0x00, 0x07,0x03  //'5' ~ '9'
                                                        };
unsigned char lcd_Buf[LCD_IN_USE];          // 自定义显示缓冲区，用于
                                            // 外部设定要显示的数据
/*******************************************************
*     模块初始化
*******************************************************/
void init_LCD(void){
    char tmpv;
    BTCTL |= 0x10;                          // set LCD 时钟
    P3DIR  = 0xff;
    P5SEL  = 0xfc;
    LCDCTL = LCDON+LCD4MUX+LCDP2+LCDP0;
     for (tmpv = 0;tmpv<LCD_IN_USE;tmpv++){
        LCDMEM[tmpv+5] |= NUM_LCD_ABCD[0];
        LCDMEM[tmpv+6] = NUM_LCD_FGE[0];    //初始时 LCD 显示"0000000",
    }
   for(tmpv=0;tmpv<5;tmpv++)                //清除 lcd 上其余不需要显示的字符
    LCDMEM[tmpv] = 0x00;
   for(tmpv=13;tmpv<29;tmpv++)
      LCDMEM[tmpv]=0x00;


}
/*****************************************************
*  set Radix point
*****************************************************/
void lcd_SetRP(){
    LCDMEM[0] |= 0x04;
}

/*****************************************************
*
*    LCD 显示
*
*****************************************************/
void lcd_Display(){

    char tmpv;
    LCDMEM[12]=0x00;
    for(tmpv=LCD_IN_USE;tmpv>0;tmpv--){
        LCDMEM[tmpv+5] |= NUM_LCD_FGE[lcd_Buf[tmpv-1]];
        LCDMEM[tmpv+4] = NUM_LCD_ABCD[lcd_Buf[tmpv-1]];
    }
```

```
}
```

实验九 图形点阵LCD

一、实验目的

1、了解图形点阵LCD的结构和原理

2、掌握在MSP430上如何使用外设

3、掌握图形点阵LCD的编程使用方法

二、实验原理

本实验使用MSP430 外接图形点阵LCD ZJM12864BSBD。ZJM12864BSBD 是一款低功耗的点阵图形式LCD ，显示格式为128 点（列） × 64 点（行），具有多功能指令。ZJM12864BSBD图形点阵LCD的工作电压为＋5.0V ± 0.5V，很容易与8、16 位的MPU 相连。其原理图如图9-1：

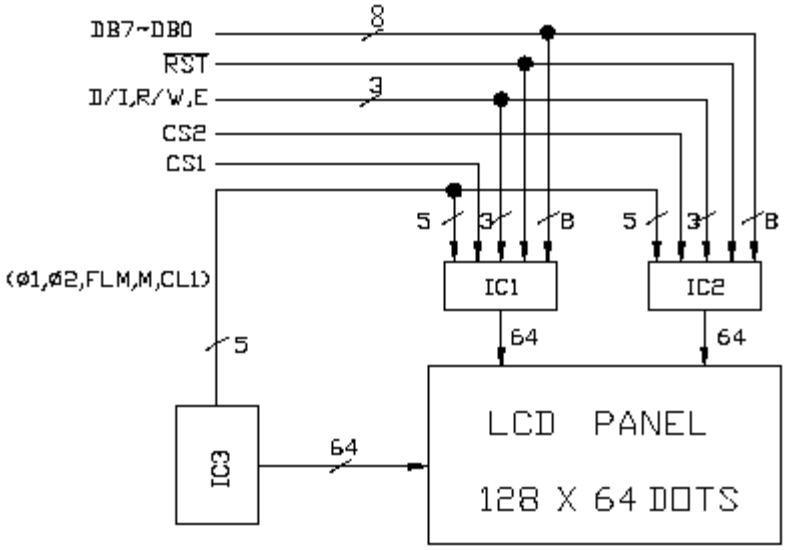


图 9-1 ZJM12864BSBD 原理图

引脚定义见表9-2：

引脚	符号	电平	功能
1	Vss	0	地
2	Vcc	5V	逻辑电压
3 4	Vo RS	-- H/L	LCD驱动电压调节 H：数据输入 L：指令输入
5	R/W	H/L	H：数据读出 L：数据写入
6	E	H,L -->L	使能信号
7	DB0	H/L	数据总线
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	

11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15 16	CS1 CS2	H H	片选信号1 片选信号2
17	RST	L	复位信号
18	Vee	--	10v输出端
19 120	NC NC	-- --	

表 9-2 引脚定义

点阵LCD ZJM12864BSBD使用自己的控制指令来进行数据的显示、清屏等工作，指令及其格式见表9-3：

指令 RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	功能
显示开关 0	0	0	0	1	1	1	1	1	1/0	0：显示关 1：显示关
设定显示 0 起始地址	0	0	1	段地址 0～ 63						在段地址中设定Y显示地址
页地址设定 0	0	1	0	1	1	1	页(0～7)			设定页地址计数器中RAM地址
显示起始行 0	0	1	1	显示起始地址:0～63						设定屏幕显示的起始行
写显示数据 1	0	写数据								向RAM中写数据,Y自动加1
读状态 0	1	BUSY	0	ON/OFF	RESET	0	0	0	0	读状态： BASY 0:允许输入指令 1:忙 ON/OFF: 0显示开 1显示关 RESET: 0:正常 1:复位
读显示数据 1	1	读书据								RAM中的数据读到数据总线上

表 9-3 指令格式

结合控制和指令表，得出在点阵中的某一个点上让其显示的步骤如下：

1) 先选则JC1 或JC2 或两个都选(进行片选)
2) 选择数据类型（数据OR 指令、输入OR 输出）
3) 显示起始行地址
4) 显示开关
5) 设定显示中Y 起始地址
6) 设定显示中页地址

另外，实验所用的LCD是逆向显示，就是说一个byte的数据在显示的时候，高位显示在下面，低位显示在上面，图9-4为八进制数据0x52显示的情况：

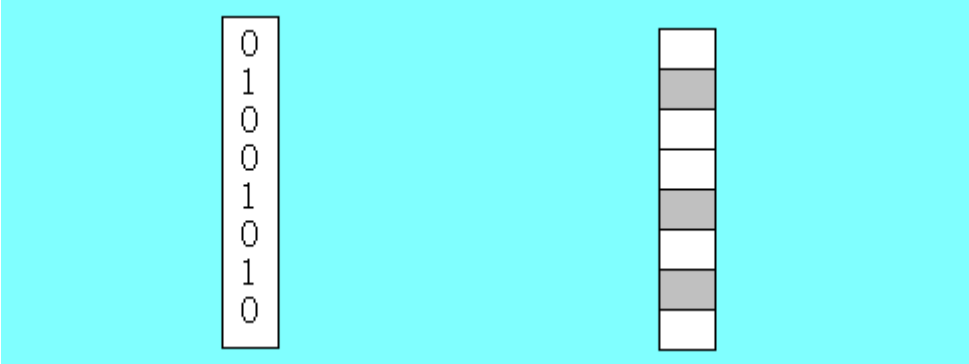


图 9-4　0x52 在 LCD 中的显示

图形点阵 LCD 需要显示的字符串对应的数据可以通过字模软件生成，在图形点阵 LCD 实验目录下附加了一个汉字模型软件。

以上介绍了实验使用的 LCD,下面介绍实验与该 LCD 的接口电路,电路图见图 9-5。

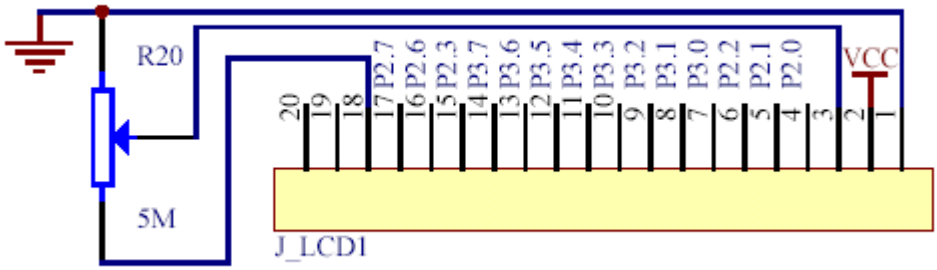


图 9-5 接口电路

从图9-5中我们可以看到449 的P2 口接点阵的控制信号，P3 口接点阵的数据信号。具体的口线连接为:

P2.0——RS；　P2.1——R/W；　P2.2——EN；　P2.3——CS1；
P2.6——CS2；　P2.7——RST；　P3.0——DB0；　P3.1——DB1；
P3.2——DB2；　P3.3——DB3；　P3.4——DB4；　P3.5——DB5；

三、实验内容

1、显示字符串Hello到LCD
2、使用键盘控制字符Hello或者中文字符串在LCD上左右、上下移动

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON
置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、设置与图形点阵LCD相关的开关

以下开关为 ON

DIP 开关 P2 的 SW1、SW2、SW3

DIP 开关 P3 的 SW1、SW2、SW3、SW4

DIP 开关 P4 的 SW4、SW5、SW6

DIP 开关 P5 的 SW5、SW6、SW7、SW8

DIP 开关 P5 的 SW1、SW2、Sw3、SW4

DIP 开关 P6 的 SW6、SW7、SW8

并且要求开关 P17 的 SW1、SW3、SW4、SW5 为 OFF

4、显示特定字符到LCD

在实验过程中，对图形点阵 LCD 的触动可能导致点阵 LCD 不显示而 LED、段式 LCD 显示的现象。出现这种现象可断电后重新安装一次即可。图形点阵 LCD 需要显示的字符串对应的数据可以通过字模软件生成，在图形点阵 LCD 实验目录下附加了一个汉字模型软件。

5、键盘控制LCD上字符的移动

五、思考与分析

1、 程序中控制字符移动时，没有实现字符的循环移动，即当字符串移动到边界时仍然可以移动，显示不完的部分从另一边显示出来。试修改程序完成以上功能。

2、修改程序，使字符在按键没有松开时连续移动。

六、实验参考代码

```
/*
*******************************************************************************
* 文件名称：
*           main.c
* 文件说明：
*           根据键盘的控制，移动字符串"Hello"或者"华东师范大学",行列键盘的"7"用
*       于字符串左移,"2"用于字符串下移，"5"用于字符串右移，"A"
*       用于字符串上移。按键每按一下向各自的方向移动一个点阵的
*       位置。
*******************************************************************************
*/
#define MSP430F449_H 0
#include <msp430x44x.h>

#ifndef LCD_IN_USE
#include "lcd.c"
#endif

#ifndef KEY_BOARD
#include "keyboard12.c"
#endif
/*******************************************************************************/
```

```
#define TOP       0              // 竖直方向零点
#define BOTTOM 64                // 竖直方向坐标最大值
#define LEFT      0              // 水平方向坐标最小值
#define RIGHT    128             // 水平方向坐标最大值
#define EN_WIDTH 40              // Hello 字符串的长度
#define CH_WIDTH 96              // 中文字符串的长度
#define OP_CHINESE       1   // 选择显示中文字符
#define OP_ENGLISH       2   // 选择显示英文字符
void drawStr(unsigned char x,unsigned char y);
void op_Select(char OP);
unsigned char showData[]={//16*8 点阵的“Hello”

                0x08,0x20,0xF8,0x3F,0x08,0x21,0x00,0x01,
                0x00,0x01,0x08,0x21,0xF8,0x3F,0x08,0x20,//H (0)

                0x00,0x00,0x00,0x1F,0x80,0x22,0x80,0x22,
                0x80,0x22,0x80,0x22,0x00,0x13,0x00,0x00,//e (1)

                0x00,0x00,0x08,0x20,0x08,0x20,0xF8,0x3F,
                0x00,0x20,0x00,0x20,0x00,0x00,0x00,0x00,//l (2)

                0x00,0x00,0x08,0x20,0x08,0x20,0xF8,0x3F,
                0x00,0x20,0x00,0x20,0x00,0x00,0x00,0x00,//l (3)

                0x00,0x00,0x00,0x1F,0x80,0x20,0x80,0x20,
                0x80,0x20,0x80,0x20,0x00,0x1F,0x00,0x00,//o (4)

};
unsigned char showData_1[]={ //16*16 点点阵的"华东师大"

                0x20,0x00,0x10,0x04,0x08,0x04,0xFC,0x05,
                0x03,0x04,0x02,0x04,0x10,0x04,0x10,0xFF,
                0x7F,0x04,0x88,0x04,0x88,0x04,0x84,0x04,
                0x86,0x04,0xE4,0x04,0x00,0x04,0x00,0x00,//华  (0)

                0x00,0x00,0x04,0x00,0x04,0x20,0xC4,0x18,
                0xB4,0x0E,0x8C,0x04,0x87,0x20,0x84,0x40,
                0xF4,0xFF,0x84,0x00,0x84,0x02,0x84,0x04,
                0x84,0x18,0x04,0x30,0x00,0x00,0x00,0x00,//东  (1)

                0x00,0x40,0xFC,0x27,0x00,0x10,0x00,0x0E,
                0xFF,0x01,0x00,0x00,0xF2,0x0F,0x12,0x00,
                0x12,0x00,0x12,0x00,0xFE,0xFF,0x12,0x00,
                0x12,0x04,0x12,0x08,0xF2,0x07,0x00,0x00,//师  (2)
```

```
        0x44,0x08,0x94,0x09,0xA4,0xF8,0x64,0x04,
        0x04,0x03,0x0F,0x00,0x04,0x00,0xE4,0x3F,
        0x24,0x40,0x2C,0x40,0x2F,0x42,0x24,0x46,
        0xE4,0x43,0x04,0x70,0x04,0x00,0x00,0x00,//范  (3)

        0x20,0x00,0x20,0x80,0x20,0x40,0x20,0x20,
        0x20,0x10,0x20,0x0C,0xA0,0x03,0x7F,0x00,
        0xA0,0x01,0x20,0x06,0x20,0x08,0x20,0x30,
        0x20,0x60,0x20,0xC0,0x20,0x40,0x00,0x00,//大  (4)

        0x40,0x00,0x30,0x02,0x10,0x02,0x12,0x02,
        0x5C,0x02,0x54,0x02,0x50,0x42,0x51,0x82,
        0x5E,0x7F,0xD4,0x02,0x50,0x02,0x18,0x02,
        0x57,0x02,0x32,0x02,0x10,0x02,0x00,0x00 //学  (5)

};
/*****************************************************************************
*
*   用于取得一个字节的低 N 位的数组
*****************************************************************************/
const unsigned char mapTbl[]=
{
        0x01,0x03,0x07,0x0f,
        0x1f,0x3f,0x7f,0xff
};

unsigned char px,                               // 显示字符串的 X 位置
              py,                               // 显示字符串的 Y 位置
              width;                            // 字符串的宽度
unsigned char * showBuf;                        // 显示数据缓冲区
/*****************************************************************************
*
*   main 函数
*****************************************************************************/
void main()
{
    WDTCTL = WDTHOLD + WDTPW;                   // 关闭看门狗
    init_LCD();                                 // 初始化图形点阵 LCD
    init_Keyboard();                            // 初始化键盘
    op_Select(OP_ENGLISH);                      // 选择显示字符类型
    drawStr(px,py);                             //显示字符串
    while(1)
    {
```

```
            key_Event();                                    //检测按键事件
            if(key_Flag== 0x01)                             //有按键
            {
                key_Flag=0x00;                              //清除按键标识
                if(key_val==0x0A)                           //上移
                {
                    if(px>TOP)                              //如果可以上移
                    {
                        clear_Rect(px/8,0,3,64);        //清除 chip1 可能有数据的三个区域
                        clear_Rect(px/8,64,3,64);       //清除 chip2 可能有数据的三个区域
                        drawStr(px-1,py);               //显示字符
                        px = px-1;
                    }
                }else if (key_val==0x07)                    //左移
                {
                    if(py>LEFT)                             //如果可以左移
                    {
                        clear_Rect(px/8,0,3,64);
                        clear_Rect(px/8,64,3,64);
                        drawStr(px,py-1);
                        py=py-1;
                    }
                }else if (key_val==0x02)                    //下移
                {
                    if(px<BOTTOM-16)                        //如果可以下移
                    {
                        clear_Rect(px/8,0,3,64);
                        clear_Rect(px/8,64,3,64);
                        drawStr(px+1,py);
                        px = px+1;
                    }
                }else if (key_val==0x05)                    //右移
                {
                    if (py<RIGHT - CH_WIDTH)                //如果可以右移
                    {
                        clear_Rect(px/8,0,3,64);
                        clear_Rect(px/8,64,3,64);
                        drawStr(px,py+1);
                        py = py+1;
                    }
                }
            }
    }
}
```

```
/*****************************************************************************
*   在指定的位置显示字符串"Hello" 或者 "华东师范大学"
*   x 是行坐标，y 是列坐标，0=<x<=RIGHT - 字符串长度
*       0=<y<=BOTTOM-字符串高度
*****************************************************************************/
void drawStr(unsigned char x,unsigned char y)
{
      unsigned char tmpv,t1,t2,t3;

      if((x&0x07)==0x00)                              //如果刚好位于某一页的开始
      {
       for(tmpv=y;tmpv<y+width;tmpv++)
       {
            move_To(x/8,tmpv);                        //移动到指定页
            write_Data(showBuf[2*(tmpv-y)]);          //填写数据
//由于显示的字符是 16*N 点阵，占用两页，现在移动到下页
            move_To(x/8+1,tmpv);
            write_Data(showBuf[2*(tmpv-y)+1]);        //填写数据
        }
      }else{
          t1 = (x&0x07);
          for(tmpv=y;tmpv<y+width;tmpv++)
          {

/*****************************************************************************
* 实验使用的图形点阵 LCD 是逆向显示的，关于逆向显示的含义请参看实验中的说明
*****************************************************************************/
            move_To(x/8,tmpv);                        //移动位置
            t2 = showBuf[2*(tmpv-y)];                 //取得要显示的数据
            t2 = t2 & mapTbl[8-t1];                   //获取数据的低（8-t1)位
            t2= t2<<t1;                               //左移 t1 位

            write_Data(t2);                           //填写数据
            move_To(x/8+1,tmpv);                      //移动
            t2 = showBuf[2*(tmpv-y)];                 //取得数据
            t2= t2>>(8-t1);                           //右移(8-t1)
            t3 = showBuf[2*(tmpv-y)+1];
            t3 = t3 & mapTbl[8-t1];                   //取得低(8-t1)位
            t3=t3<<t1;                                //右移 t1 位
            t2=(t2+t3);                               //组合成要显示的数据
            write_Data(t2);                           //显示数据
            move_To(x/8+2,tmpv);                      //移动到下一页
            t2 = showBuf[2*(tmpv-y)+1];
```

```
            t2=t2>>(8-t1);                          //右移(8-t1)位
            write_Data(t2);                         //填写数据
          }
      }
}
/*************************************************************************
*
*   根据选择的是显示中文还是英文，设置不同的显示变量
*
*************************************************************************/
void op_Select(char OP)
{
      if(OP==OP_CHINESE)
      {
          width = CH_WIDTH;                          //   设定字符串长度
          px=24;                                     // 初始化显示位置,X 坐标
          py=16;                                     // Y 坐标
          showBuf = showData_1;
      }
      else if (OP==OP_ENGLISH)
      {
          width = EN_WIDTH;                          //   设定字符串长度
          px=24;                                     // 初始化显示位置,X 坐标
          py=40;                                     // Y 坐标
          showBuf = showData;
      }
}
/*******************************************************
*    文件名称：
*              lcd.c
*   文件说明：对点阵 lcd 的基本操作初始化、移动显示数据的位置、
*              写数据、对指定区域进行清除进行封装
*******************************************************/

#include <msp430x44x.h>

/*************************************************
*   write_Command(),用于写命令到 LCD，
* CS(片选)在此函数外设置
*************************************************/
void write_Command(char cmd)
{
      //P1OUT &= ~0x40;                   //禁止  EN
```

```
    P3DIR =    0xff;
    P3OUT    = cmd;
    P1OUT &= ~0x30;                    //rs=0 ,r/w=0
    P6OUT |= 0x40;
    P6OUT &= ~0x40;
    P1OUT |= 0x40;                    //使能
    P1OUT &= ~0x40;                    //禁止

}
/*****************************************************
*  设置显示的位置,片选在函数外部设置
*****************************************************/
void move_To(char x,char y)
{
    unsigned char tmp;
    tmp=(y&0x7f);
    if(tmp<64)                              //如果位置在 Chip1
    {
        P2OUT &= ~0x80 ;              //cs2=0
        P6OUT |= 0x10;                //cs1=1

        write_Command(0xb8 + x);          //设置  x
        write_Command(0x40 + tmp);       //设置  y
    }else if(tmp>63)                       //在  chip2
    {
        P2OUT |= 0x80 ;               //cs2=1
        P6OUT &= ~0x10;                //cs1=0

        write_Command(0xb8 + x);          //设置  x
        write_Command(0x40 + tmp-64); //设置  y
    }

}
/*****************************************************
* write data
*****************************************************/
void write_Data(char content)
{
    P3DIR =    0x00;                    //  置位输入模式
   P2OUT |= 0x02;                    // r/w=1  读状态
    P2OUT |= 0x04;                    // chip  使能
    while((P3IN & 0x80)==0x80);//检测 LCD 是否忙
    P1OUT &= ~0x20;               //r/w=0;
```

```
        P1OUT &= ~0x40;                    //禁止 EN
        P3DIR =    0xff;
        P1OUT &= ~0x30;                    //rs=0 ,r/w=0
        P1OUT |= 0x10;                   //rs=1,data
        P3OUT    = content;
        P6OUT |= 0x40;
        P6OUT &= ~0x40;
        P1OUT |= 0x40;                    //使能
        P1OUT &= ~0x40;                    //禁止
}


/*****************************************************
*    清除 chip1 或者 chip2 上的某一区域，选择的区域只是在
*    同一 chip 上,x 是页地址,Y 是列地址,h 是多少行,w 是指多少
*    列.区域可以是两个片组成的区域的任何位置
*****************************************************/
void clear_Rect(unsigned char x,unsigned char y,unsigned char h,unsigned char w)
{
        unsigned char tmpv,tmp;

            for(tmpv=x;tmpv<(x+h);tmpv++)
            {
                for(tmp=y;tmp<(y+w);tmp++)
                {
                   move_To(tmpv,tmp);//移动
                   write_Data(0x00); //在当前位置写 0，即清除当前位置的显示内容
                 }
            }
}
/*****************************************************
*   初始化 LCD
*****************************************************/
void init_LCD()
{
        P3DIR    = 0xff;                 //  设置 P3 输出模式
        P3OUT    = 0x00;                   //  初始值为 0
        P1DIR |= 0x70;                  // P1.4,P1.5,P1.6 置为输出模式
        P1OUT &= ~0x70;
        P2DIR |= 0xC0;
        P2OUT &= ~0xC0;                 //reset=0,cs2=0
        P6DIR |= 0x50;
        P6OUT &= ~0x50;
```

```
    P2OUT |= 0x80 ;                 //cs2=1
    P6OUT |= 0x10;                  //cs1=1

    P1OUT |= 0x40;                  // en=1
    P2OUT |= 0x40;                  //reset=1
    write_Command(0xc0);            //
    write_Command(0x3f);            //显示开

    clear_Rect(0,0,8,64);       //清除第一块
    clear_Rect(0,64,8,64);      //清除第二块

}
```

实验十　超低功耗实验

一、实验目的

1、了解 MSP430 系列芯片的几种低功耗模式
2、掌握 MSP430 系列芯片的超低功耗设置方法
3、掌握 C 与汇编混合编程的方法

二、实验原理

TI 的 MSP430 系列芯片具有低功耗特性的单片机系列，适合应用于采用电池供电的长时间工作场合。MSP430 通过使用不同的时钟信号：ACLK、MCLK 和 SMCLK，并且可以在不同工作模式进行切换，更合理地利用系统的电源，实现整个系统的超低功耗。

通过设置 CPU 内状态寄存器 SR 中的 SCG1、SCG2、OscOff 和 CPUOff 可由软件配置成 6 种不同工作模式：1 种活动模式和 5 种低功耗模式。通过设置控制位 MSP430 可以从活动模式进入到相应的低功耗模式；而各种低功耗模式又可通过中断方式回到活动模式。10-1 图显示了各种模式之间的关系。

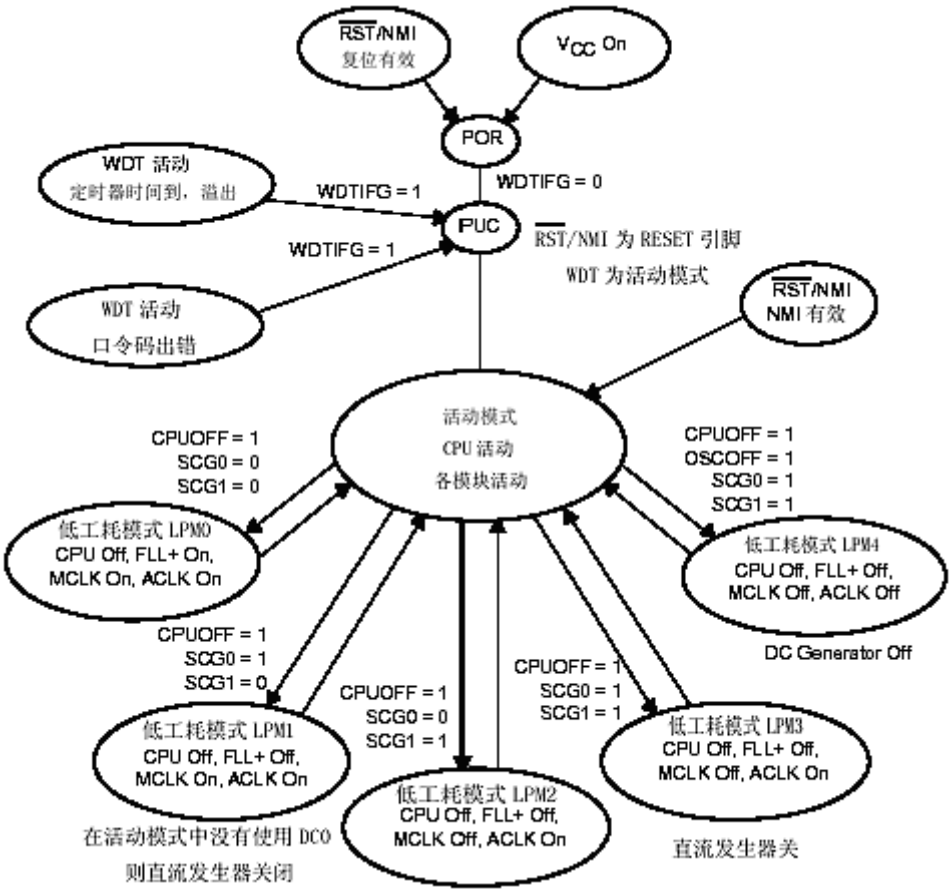


图 10-1 工作模式转换图

各种工作模式、各控制位及 3 种时钟的活动状态之间的相互关系如表 10-1 所示：

工作模式	控制位	CPU 状态、振荡器及时钟
活动模式 AM	SCG1=0 SCG0=0 OscOff=0 CPUOFF=0	CPU 处于活动状态 MCLK 活动 SMCLK 活动 ACLK 活动
低功耗模式 0 LPM0	SCG1=0 SCG0=0 OscOff=0 CPUOFF=1	CPU 处于禁止状态 MCLK 被禁止 SMCLK 活动 ACLK 活动
低功耗模式 1 LPM1	SCG1=0 SCG0=1 OscOff=0 CPUOFF=1	CPU 处于禁止状态 如果 DCO 未用作 MCLK 或 SMCLK，则直流发生器被禁止，否则仍保持活动 MCLK 被禁止 SMCLK 活动 ACLK 活动
低功耗模式 2 LPM2	SCG1=1 SCG0=0 OscOff=0 CPUOFF=1	CPU 处于禁止状态 如果 DCO 未用作 MCLK 或 SMCLK，自动被禁止 MCLK 被禁止 SMCLK 被禁止 ACLK 活动
低功耗模式 3 LPM3	SCG1=1 SCG0=1 OscOff=0 CPUOFF=1	CPU 处于禁止状态 DCO 被禁止，直流发生器被禁止 MCLK 被禁止，SMCLK 被禁止 ACLK 活动
低功耗模式 4 LPM4	SCG1=× SCG0=× OscOff=1 CPUOFF=1	CPU 处于禁止状态 DCO 被禁止，直流发生器被禁止 所有振荡器停止工作 MCLK 被禁止，SMCLK 被禁止 ACLK 被禁止

表 10-1 工作模式与控制位、时钟关系

各种工作模式的功耗见图 10-2：

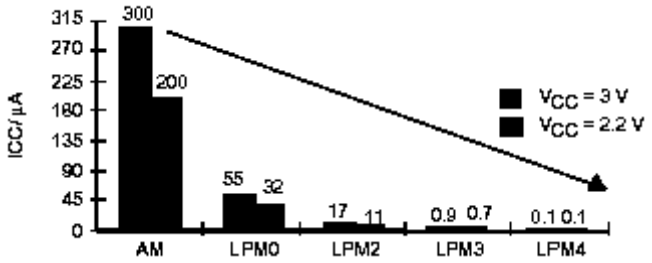


图 10-2 工作模式与功耗图

另外，MSP430 的瞬间响应特性是系统超低功耗事件驱动方式的重要保证。如图 10-3 所示：

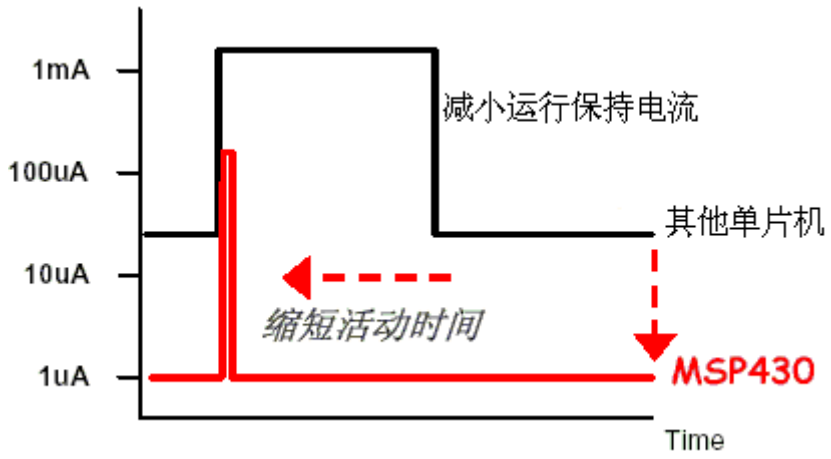


图 10-3 瞬时响应图

三、实验内容

使用行列扫描键盘作一个加法器，把输入和结果显示到段式 LCD 上。程序开始时工作在正常模式，如果用户一段时间没有进行操作（即是没有进行按键），设置单片机进入低功耗模式，关闭 LCD。在低功耗模式下，如果用户有按键操作，返回低功耗前的状态。

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON

置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、打开行列键盘对应的开关

DIP 开关 P5 的 SW1、SW2、Sw3、SW4

DIP 开关 P6 的 SW6、SW7、SW8

并且要求开关 P17 的 SW3、SW4、SW5 为 OFF

4、LCD 对应的开关：

DIP 开关 P11 的 SW1、SW2、SW3、SW4、SW5、SW6，

DIP 开关 P12 的 SW1、SW2、SW3、Sw4、SW5、SW6、SW7、SW8，

DIP 开关 P13 的 SW5、Sw4

DIP 开关 P1 的 SW4、Sw5、SW6、SW7（com0～com3）

五、分析与思考

1、 实验中给出了 C 语言与汇编混合编写的代码，请写出单独的汇编程序

2、 改变程序中低功耗模式的设置方式，重写该程序

六、实验参考代码

```
/*******************************************************************************
*
* 通过键盘、LCD 来进行超低功耗实验
* 在一段时间内，如果用户没有进行按键操作，系统将进入“睡眠”－－低功耗状态
```

```
* 用户按键后（INC），系统从低功耗状态转到正常的工作状态
* 在非低功耗状态下，程序接收键盘按键执行加法器操作(因为键盘和 LCD 限制不能实现复
* 杂的功能,如乘法、减法、除法等）.
*
*******************************************************************************/

#include <msp430x44x.h>

#include "lcd.c"
#include "keyBoard12.c"

/*******************************************************************************
 unsigned int wait_Time;
 unsigned long    num1,num2;
/*******************************************************************************/

extern void sleep_Mode();
void init_BT();
void show_Data(unsigned long);

/*******************************************************************************
*   main()函数
*******************************************************************************/
void main(void)
{
    char len=0;
    WDTCTL = WDTHOLD + WDTPW;   //停止看门狗
    init_BT();                  //初始化 Basic Timer
    init_Keyboard();            //键盘初始化
    init_LCD();                 //LCD 初始化
    _EINT();                    //开中断
    while(1)
    {
        lcd_Display();          // LCD 显示
        key_Event();            // 检测键盘事件
        if(key_Flag==1)
        {
            switch (key_val)
            {
              case 0x0A:                //对应 清零
                    num1=0;
                    num2=0;
                    len=0;
                    show_Data(num2);   // 填写 0 到 LCD 缓冲区
```

```
                    break;
                case 0x0B:                          // 对应 +
                    num1= num1 + num2;
                    show_Data(num1);     // 显示上次运算结果
                    num2=0;
                    len=0;
                    break;
                default:                            // 数字 0 到 9
                    len += 1;
                    if(len>7) break;     // 限制输入数字个数
                    num2=num2*10+key_val;
                    show_Data(num2);     // 显示当前输入数字
                    break;
            }

            key_Flag = 0;
            wait_Time = 0;                  // 重新设置 wait_Time
        }


    }

}
/*******************************************************************************
*
*    按键触发中断后，设置 CPU、LCD、键盘正常工作
*
*******************************************************************************/
#pragma vector = PORT1_VECTOR
__interrupt void active_mode()
{

    P1IFG &= 0x00;     //清除标识

    LCDCTL |= 0x01;     // 重新初始化 LCD,LCDON = 1

    P1IE &= 0x7F; //取消中断使能
    init_Keyboard();

    IE2 |= 0x80;   // 使能 bt 中断

    __bic_SR_register_on_exit(LPM0_bits); // 退出低功耗模式
}
/*******************************************************************************
```

```
*
*     Basic Timer  中断向量
*******************************************************************************/
#pragma vector = BASICTIMER_VECTOR
__interrupt void get_Time()
{
   wait_Time += 1;              //   计数器加 1
   if(wait_Time >= 40000)     // 是否等待时间到 10s
   {                           //
       wait_Time = 0;          // 等待时间计数器清零
       sleep_Mode();           // 处理设备进入适当状态
       __bis_SR_register_on_exit(LPM0_bits);   // cpu 进入低功耗
   }
}
/*******************************************************************
*
*     初始化 Basic Timer
*
*******************************************************************************/
void init_BT()
{
      BTCTL   =BT_MDLY_0_25;                    //   Basic Timer 1  中断频率
      IE2     |= 0x80;                          // 使能 bt 中断
}
/*******************************************************************************
*
*         把十进制数的每一位都填充到 LCD 缓冲
*
*******************************************************************************/
void show_Data(unsigned long num)
{
      char tmp1;
      for(tmp1=0;tmp1<7;tmp1++)
      {
          lcd_Buf[tmp1] = num%10;     // 依次取数
          num = num/10;                // 数字除以 10
      }
}
;-------------------------------------------------------------------------------------------------------------
;          文件名称：
;                    ultra_low_Power.s43
;          文件说明：
;                    主程序中使用的函数：进入低功耗模式，以及退出低功耗模式
;-------------------------------------------------------------------------------------------------------------
```

```
#include   "msp430x44x.h"

            public      sleep_Mode
            RSEG        CODE
sleep_Mode:
            bic.b       #0x80,&IE2              ; 取消 bt 中断
            and.b       #0x1,&P1DIR             ; 输入模式
            and.b       #0x8,&P1IES             ; 设置中断触发方向：从低到高
            bis.b       #0x8,&P1IE              ; 使能中断
            bic.b       #0x1,&LCDCTL            ; LCDON = 0
            mov.b       &P1IFG,R14              ; 清除标识
            clr.b       &P1IFG                  ;
            ret                                 ;
            END
;-------------------------------------------------------------------------------------------------------------------
-
```

实验十一 模拟设定时间和RS-485 通信实验

一、实验目的

1、 熟悉独立键盘的设计方法，掌握独立键盘工作原理和编程设计

2、 利用汇编编程实验，更好地理解 MCU、编译器的原理

二、实验原理

本实验所用到的独立按键、LED、RS-485通信原理已在‘实验二 键盘与LED’和‘实验七 异步通讯模块中讲述，需要了解的可以查看实验二、七中的相关内容，本实验不再赘述原理部分。

三、实验内容

1、 使用独立按键式键盘，用 LED 显示键值

2、 使用 RS485 串口在两块实验板之间进行通讯操作

要求：

使用行列式键盘发送数字到另一块实验板，同时接收从另一块实验板发送来的字符，然后把接收到的数字显示到 LED

四、实验步骤

1、打开 JTAG 与晶振对应的开关

置 DIP 开关 P6 的 SW1、SW2 以及 DIP 开关 P7 的 SW4、SW3、SW2 为 ON

置 DIP 开关 P10 的 SW1,SW2 为 ON

2、开实验板电源对应的开关

置 DIP 开关 P8 的 SW2、SW3、SW4 和 P9 的 SW5 为 ON

3、设置下列开关为ON，做独立按键实验

DIP 开关 P17 的 SW3、SW4、SW5，DIP 开关 P6 的 SW6、SW7、SW8

DIP 开关 P2 的 SW1、SW2、SW3、SW4、SW5，DIP 开关 P4 的 SW6

DIP 开关 P3 的 SW1、SW2、SW3、SW4

4、做 RS485 串口通讯

置 DIP 开关 P4 的 SW2、SW3， DIP 开关 SW_USART 的 SW1、SW4 为 ON

1、 两个 RS485 连接的电路如图 7-15:

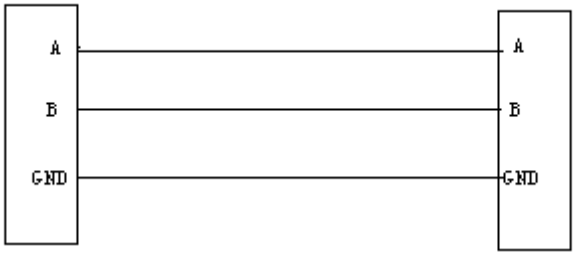


图11-1 RS485 连接

5、建立汇编工程时，需注意在工程option中GeneralOptions->Assembler-only project选中，如图11-2所示：

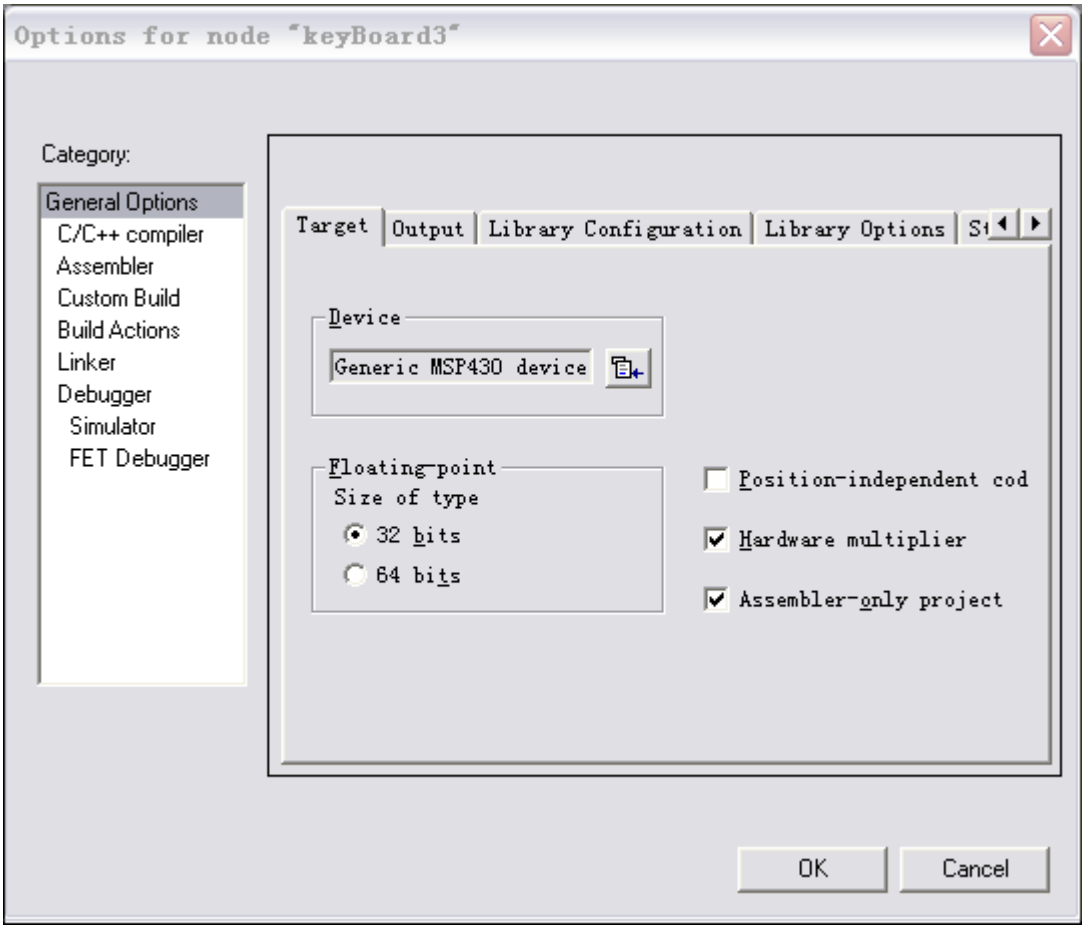


图11-2 工程option选项中的Assembler-only project

五、分析与思考

1、 如何在设定时间时，让 LED 上模拟的时钟不运行，即‘秒’部分停止显示，设置完毕后再次开始计时。

2、 在不使用 XT2 的情况下，设置波特率为 2400，做 RS485 通讯实验

六、实验参考代码

1．独立按键式键盘

```
;*******************************************************************************
; 文件名称：
;             keyBoard3.s43
; 文件说明：
;             使用 LED 和小键盘做一个计时程序,格式为 HHMMSS，按 FUN 键
;       后将选中其中的两位，再按“INC”对选中为进行加操作，按“DEC”
;       进行减操作,按四次退出功能选择。
;*******************************************************************************
#include   "msp430x44x.h "

;*******************************************************************************
;          RAM 单元定义
;*******************************************************************************
hour                equ             201h            ; 时
```

```
min             equ     202h        ; 分
second          equ     203h        ; 秒
flag            equ     204h        ; 标志寄存器
led_disp_data   equ     205h        ; led 显示缓冲
led_disp_bit    equ     206h        ; led 显示位选
counter         equ     207h        ; 显示计数
dip             equ     208h        ; 小数点
inc_dec_buffer  equ     209h        ; inc 缓存;dec 缓存
fun_buffer      equ     20ah        ; fun 缓存
key_flag        equ     20bh        ; 键盘中断标志缓存
fun_flag        equ     20ch        ; 功能键标志缓存
second_flag     equ     200h        ; 0.5S 标志
;*******************************************************************************
;               按键定义
;*******************************************************************************
key_inc         equ     008h        ; P1.3
key_dec         equ     004h        ; P1.2
key_fun         equ     002h        ; P1.1
;*******************************************************************************
; 首先，对系统进行初始化。关看门狗、置端口的输入输出状态、以及中断情况
; 接着，对用到的 ram 区进行初始化、对标志位初始化。
;*******************************************************************************
                ORG     08000h              ; 程序起始位置
;*******************************************************************************
Reset
                mov.w   #0600h,SP                       ; 初始化堆栈
StopWDT         mov.w   #WDTPW+WDTHOLD,&WDTCTL          ; 关看门狗

Setup_P1
                bic.b   #key_inc+key_dec+key_fun,&P1DIR ; 设置 P1DIR
                bis.b   #key_inc+key_dec+key_fun,&P1OUT ;
                bis.b   #key_inc+key_dec+key_fun,&P1IES ; 接收从高到低的跳变
                bis.b   #key_inc+key_dec+key_fun,&P1IE  ; 使能中断
Setup_P3
                mov.b   #0ffh,&P3DIR                    ; P3 为输出模式
                mov.b   #00h,&P3OUT                     ; 输出值为 0
                mov.b   #0ffh,&P4DIR                    ; 设置 P4 为输出模式
                bis.b   #03h,&P4OUT                     ; P4.1,P4.0 为 1
Setup_BT
                mov.b   #BTSSEL+BT_ADLY_500,&BTCTL      ; 中断间隔设置
                mov.b   #0a5h,&BTCTL                    ; 设置 BTCTL
                bis.b   #BTIE,&IE2                      ; 使能 Basic Timer 中断

Setup_TA
```

```
                mov.w    #TASSEL1+TACLR,&TACTL          ; led 刷新一位定时
                mov.w    #CCIE,&CCTL0
                mov.w    #3600,&CCR0                    ; 4.5ms
                bis.w    #MC0,&TACTL

clr_ram
                clr      R8                             ; R8 =0
loop_clr        clr.b    second_flag(R8)                ; 对从 200H 起始的单元设置其值为 0
                inc.b    R8                             ; R8=1
                cmp.b    #0dh,R8                        ; 判断 R8 是否大于 0x0d
                jeq      clr_ram_over                   ;
                jmp      loop_clr                       ; 继续循环
clr_ram_over    clr      R8                             ; 清零 r8
                eint                                    ;
;*******************************************************************************
;                主程序区
;*******************************************************************************
mainloop
                nop
                nop
                bis.w    #LPM0,SR                       ; CPUOFF
                nop
                nop
                bic.b    #key_inc+key_dec+key_fun,&P1IE ; 清除中断使能
                call     #key_scan                      ; 调用 key_scan
ret_jmp         jmp      mainloop                       ;

key_scan
                push     R15                            ; 保存 R15
                call     #delay_10ms                    ; 去抖动和防止干扰
                mov.b    &P1IN,R15                      ;
                inv.b    R15
                bit.b    #key_inc+key_dec+key_fun,R15   ; 判断有无按键按下
                jz       ret_scan

loop_key
                mov.b    &P1IN,R15                      ; 判断按键有无松开
                inv.b    R15
                bit.b    #key_inc+key_dec+key_fun,R15   ;
                jnz      loop_key                       ; 没有按键继续循环
                call     #keycodej3                     ; 判断键值


ret_scan        clr.b    &P1IFG                         ; 清除标志
```

```
                bis.b       #key_inc+key_dec+key_fun,&P1IE   ; 开中断

                jmp         ret_jmp                          ;

;*****************************************************************
;                   PORT 1 INT
;*****************************************************************
PORT1_INT
                bic.w       #LPM0,0(SP)                      ; 退出 LMP0
                mov.b       &P1IFG,&key_flag                 ; 获取按键中断标识
ret_int         clr.b       &P1IFG                           ; 清除 P1IFG
                reti                                         ;
;*****************************************************************
;               延时 10ms 子程序
;*****************************************************************************
delay_10ms
                mov         #2666,R15                        ; R15 ＝ 2666
loop_delay      dec         R15                              ; R15=R15-1
                jnz         loop_delay                       ;
                ret                                          ;
;*****************************************************************************
;              判断按键子程序
;*****************************************************************************
keycodej3
test_inc        bit.b       #key_inc,&key_flag               ; 是 INC 吗？
                jz          test_dec                         ;
                call        #inc_fun                         ;
                jmp         ret_1                            ;
test_dec        bit.b       #key_dec,&key_flag               ; 是 DEC 吗？
                jz          test_fun                         ;
                call        #dec_fun                         ;
                jmp         ret_1                            ;
test_fun        bit.b       #key_fun,&key_flag               ; 是 FUN 吗？
                call        #fun_fun                         ;
                jmp         ret_1                            ;

ret_1
                ret
;*****************************************************************************
;             inc 按键子程序
;*****************************************************************************
inc_fun
                bit.b       #1,&fun_flag                     ; 是否对 FUN 按下过？
                jz          ret_inc                          ;
```

```
                cmp.b       #1,&fun_buffer                  ; 是 HOUR 要增加吗？
                jeq         inc_hour                        ;

                inc.b       &inc_dec_buffer                 ; 操作缓存＋1
                dadc.b      &inc_dec_buffer                 ;
                cmp.b       #60h,&inc_dec_buffer            ; 是否要进位
                jlo         inc_move                        ;
                clr.b       &inc_dec_buffer                 ; 进位，清零操作缓存
                jmp         inc_move                        ;
inc_hour
                inc.b       &inc_dec_buffer                 ;
                dadc.b      &inc_dec_buffer                 ;
                cmp.b       #23h,&hour                      ; 是否要清零小时
                jlo         inc_move                        ;
                clr.b       &inc_dec_buffer                 ;
inc_move        mov.b       &fun_buffer,R15                 ;

                mov.b       &inc_dec_buffer,second_flag(R15)    ;
ret_inc         ret
;*************************************************************************
;               dec 按键子程序
;*************************************************************************
dec_fun
                bit.b       #1,&fun_flag                    ; 是否 FUN 按下过？
                jz          ret_dec                         ;
                cmp.b       #01h,&inc_dec_buffer            ; 大于等于 1 吗？
                jhs         loop_dec                        ; 如果大于等于跳转
                cmp.b       #1,&fun_buffer                  ; 是对小时操作吗？
                jeq         dec_hour                        ;
                mov.b       #60h,&inc_dec_buffer            ; 变为 60H，对分或者秒操作
                jmp         loop_dec                        ;
dec_hour        mov.b       #24h,&inc_dec_buffer            ; 变为 24H
loop_dec        bit.b       #0fh,&inc_dec_buffer            ; 是小时吗？
                jz          set_dec                         ;
unset_dec       dec.b       &inc_dec_buffer                 ; 减去 1
                clrc                                        ;
                dadd.b      #0,&inc_dec_buffer              ;
                jmp         dec_move                        ;
set_dec         sub.b       #06,&inc_dec_buffer             ; 减去 6
                jmp         unset_dec                       ;
dec_move        mov.b       &fun_buffer,R15                 ;
                mov.b       &inc_dec_buffer,second_flag(R15)    ; 保存

ret_dec         ret
```

```
;*************************************************************************
;          fun 按键子程序
;*************************************************************************
fun_fun
              inc.b     &fun_buffer                                  ; fun_buffer  加 1
              cmp.b      #1,&fun_buffer                              ; 是 1 吗？  小时增加
              jeq        fun_star_hour                               ;
              cmp.b      #2,&fun_buffer                              ; 是 2 ？  分增加
              jeq        fun_min                                      ;
              cmp.b      #3,&fun_buffer                              ; 是 3 ？  秒增加
              jeq        fun_second                                  ;
              cmp.b      #4,&fun_buffer                             ; 是 4 ？  fun 按了 4 次
              jeq        fun_stop                                      ;

fun_ret       ret
;*************************************************************************
fun_star_hour
              bis.b      #1,&fun_flag                                ; 设置 fun_flag 标识

              mov.b      &hour,&inc_dec_buffer                      ; hour 如操作缓存

              jmp        fun_ret                                       ;
;*************************************************************************
fun_min                                                               ;
              mov.b      &min,&inc_dec_buffer                       ; min 进操作缓存
              jmp        fun_ret                                     ;
;*************************************************************************
fun_second                                                            ;
              mov.b      &second,&inc_dec_buffer                    ; second 进操作缓存
              jmp        fun_ret                                     ;
;*************************************************************************
fun_stop                                                              ;
              clr.b      &fun_flag                                  ; 清除标识
              clr.b      fun_buffer                                 ;
              jmp        fun_ret                                     ;

;*************************************************************************
;            led 显示子程序
;*************************************************************************
led_chang
                        br          led_take(R8)                    ;
                        EVEN                                          ;
led_take
                        dw          second_table_h                  ;
```

```
                dw          second_table_l                  ;
                dw          min_table_h                     ;
                dw          min_table_l                     ;
                dw          hour_table_h                    ;
                dw          hour_table_l                    ;
                                                            ;
                                                            ;
second_table_h                                              ;
                mov.b       &second,R14         ;R14=秒数
                mov.b       #2fh,&led_disp_bit  ; 控制 LED 的右边第二个显示
                jmp         take_h              ; 取高位
second_table_l                                  ;
                mov.b       &second,R14         ;
                mov.b       #1fh,&led_disp_bit  ;
                mov.b       #00h,&dip           ; 控制 LED 右边第一个显示
                jmp         take_l              ; 取低位
                                                ;
                                                ;
min_table_h     mov.b       &min,R14            ;
                mov.b       #3bh,&led_disp_bit  ; 控制 LED 的右边第四个显示
                jmp         take_h              ; 取高位

min_table_l     mov.b       &min,R14            ;
                mov.b       #37h,&led_disp_bit  ; 控制 LED 右边第三个显示
                mov.b       #20h,&dip           ; 取低位
                jmp         take_l              ;

hour_table_h    mov.b       &hour,R14           ;
                mov.b       #3eh,&led_disp_bit  ; 控制 LED 右边第六个显示
                jmp         take_h              ; 取高位
                                                ;
hour_table_l    mov.b       &hour,R14           ;
                mov.b       #3dh,&led_disp_bit  ; 显示小时的低位
                mov.b       #20h,&dip           ;
                jmp         take_l              ;
                                                ;
take_h          mov.b       R14,R13             ; 取高位
                rra.b       R13                 ;
                rra.b       R13                 ;
                rra.b       R13                 ;
                rra.b       R13                 ;
                and.b       #0fh,R13            ;
                mov.b       led_table(R13),&led_disp_data   ;
```

```
                ret                                             ;
take_l                                                          ; 取低位
                mov.b           R14,R13                         ;
                and.b           #0fh,R13                        ;
                mov.b           led_table(R13),&led_disp_data ;
                add.b          &dip,&led_disp_data              ;
                ret
;*******************************************************************************
;                           led 显示子程序
;*******************************************************************************
led_display
                        mov.b          &led_disp_data,&P3OUT ; 设置显示数据
                        bis.b          #02h,&P4OUT             ; 打开数据锁存
                        bic.b          #02h,&P4OUT             ; 关闭数据锁存


                        mov.b          &led_disp_bit,P3OUT
                        bis.b          #01h,&P4OUT             ; 打开控制锁存
                        bic.b          #01h,&P4OUT             ; 关闭控制锁存
                        ret
;*******************************************************************************
;                           TimerA_int
;*******************************************************************************
Timera_int
                          inc.b         R8                      ;
                          inc.b         R8                      ;
                          cmp.b          #0ch,R8                ;
                          jnz            Timer_int_end          ;
                          clr.b         R8                      ;

Timer_int_end             push.b    &second                     ;
                          push.b    &min                        ;
                          push.b    &hour                       ;
                          call      #led_blink                  ; 如果有键按下，LED 闪烁
                          call      #led_chang                  ; 改变 LED 缓冲
                          call      #led_display                ; 显示数据到 LED
                          pop.b     &hour                       ;
                          pop.b     &min                        ;
                          pop.b     &second                     ;
                          reti                                  ;
;*******************************************************************************
;               led_blink
;*******************************************************************************
led_blink                 cmp.b        #3,&fun_buffer     ;按下功能键三次，秒钟对应的 LED 闪烁
```

```
                    jeq         second_blink
                    cmp.b       #2,&fun_buffer   ;按下功能键两次，分钟对应的 LED 闪烁
                    jeq         min_blink
                    cmp.b       #1,&fun_buffer   ;按下功能键一次，小时对应的 LED 闪烁
                    jeq         hour_blink       ;
                    jmp         blink_ret        ;
second_blink                                     ; 秒对应的 LED 闪烁
                    bit.b       #1,&second_flag  ;
                    jnz         s_ret            ;
                    mov.b       #0aah,&second    ;
s_ret               jmp         blink_ret        ;
min_blink                                        ; 分对应的两个 LED 闪烁
                    bit.b       #1,&second_flag  ;
                    jnz         m_ret            ;
                    mov.b       #0aah,&min       ;
m_ret               jmp         blink_ret        ;
hour_blink                                       ; 小时对应的 LED 闪烁
                    bit.b       #1,&second_flag  ;
                    jnz         blink_ret        ;
                    mov.b       #0aah,&hour      ;

blink_ret           ret                          ;
;*************************************************************************
;               BASIC TIMER int
;*************************************************************************
BASIC_INT
                    xor.b   #01h,&second_flag    ; 标识取反
                    bit.b   #1,&second_flag      ; 是 1 吗?
                    jz      clock_ret            ;

                    xor.b   #01h,&flag           ;
                    setc                         ;
BASIC_END
                    dadc.b   &second             ;
                    cmp.b   #60h,&second         ; 与 60H 比较
                    jlo     clock_end            ;
                    clr.b   &second              ; 清零 second
                    dadc.b   &min                ; 如果大于 60H，分加 1
                    cmp.b   #60h,&min            ;
                    jlo     clock_end            ; 如果分大于 60H
                    dadc.b   &hour               ; 小时加 1
                    clr.b   &min                 ; 如果小时大于 24H
                    cmp.b   #24h,&hour           ;
                    jlo     clock_end            ; 小时边为 0
```

```
                clr.b     &hour                ;
clock_end

clock_ret       reti                           ;

delay     mov.b     #0ffh,R9                   ; 延迟
ttt       nop                                  ;
          nop                                  ;
          dec.b     R9                         ; R9--
          jnz       ttt                        ;
          ret                                  ;

;*************************************************************************
;          led 显示代码
;*************************************************************************

led_table
          db        0d7h          ;0
          db        14h           ;1
          db        0cdh          ;2
          db        5dh           ;3
          db        1Eh           ;4
          db        5bh           ;5
          db        0dbh          ;6
          db        15h           ;7
          db        0dfh          ;8
          db        05fh          ;9
          db        00h           ;no,用于在熄灭 LED
          db        00h           ;no

;*************************************************************************
;          中断向量
;*************************************************************************
          RSEG      INTVEC              ; MSP430 RESET 向量
;*************************************************************************
          DW        BASIC_INT           ;0FFE0h
          DW        Reset               ;0FFE2h
          DW        Reset               ;0FFE4h
          DW        Reset               ;0FFE6h
          DW        PORT1_INT           ;0FFE8h
          DW        Reset               ;0FFEAh
          DW        Timera_int          ;0FFECh
          DW        Reset               ;0FFEEh
          DW        Reset               ;0FFF0h
```

```
            DW      Reset                   ;0FFF2h
            DW      Reset                   ;0FFF4h
            DW      Reset                   ;0FFF6h
            DW      Reset                   ;0FFF8h
            DW      Reset                   ;0FFFAh
            DW      Reset                   ;0FFFCh
            DW      Reset                   ;0FFFEh
            END
```

2. RS485 通讯实验

```
;-------------------------------------------------------------------------------------------------------
;文件名称：
;           485_a.s43
;文件说明：
;           用于两个对等的实验板通过 485 接口进行通讯，本方发送的
;      数据现在在对方的 LED 上，实验使用波特率为 9600
;-------------------------------------------------------------------------------------------------------

#include "msp430x44x.h"

;-------------------------------------------------------------------------------------------------------
;  定义数据
;-------------------------------------------------------------------------------------------------------
Receive_Buf   equ   200H                    ;485 接收缓冲区
Send_Buf      equ   201H                    ;485 发送缓冲区
R_Flag        equ   202H                    ;接收到数据标识
LED_Buf       equ   203H                    ;LED 显示数据缓存，共六个字节

LED_CTRL      equ   209H                    ;LED 显示位控制
KEY_Pressed equ     20AH                    ;按键是否按下
KEY_Val       equ   20BH                    ;键值
KEY_Flag      equ   20CH                    ;键值是否可以读取

;-------------------------------------------------------------------------------------------------------
            ORG     08000h                  ; 程序起始位置
;-------------------------------------------------------------------------------------------------------
RESET       mov.w   #0600h,SP               ; 初始化堆栈
main:
            push.w  R10
            mov.w   #0x5A80,&WDTCTL         ;关看门狗
            call    #init_Keyboard          ;初始化键盘
            call    #init_LED               ;初始化 LED
            call    #init_BT                ;初始化 Basic Timer
            call    #init_485               ;初始化 485
```

```
                eint                                ;开中断
MainLoop        call    #key_Event                  ;检测键盘
                cmp.b   #0x1,&KEY_Flag              ;如果有键值可读取
                jne     T12
                mov.b   &KEY_Val,&Send_Buf
                call    #rs485_SendData             ;发送数据
                clr.b   &KEY_Flag                   ;清除键值标识
T12             cmp.b   #0x1,&R_Flag                ;是否收到数据
                jnc     MainLoop
                clr.b   R10
T14             cmp.b   #0x5,R10                    ;LED_Buf 的数据移动
                jc      T13
                mov.b   R10,R14
                and.w   #0xFF,R14
                mov.w   #LED_Buf,R15
                mov.b   R10,R12
                and.w   #0xFF,R12
                add.w   R12,R15
                mov.b   0x1(R15),LED_Buf(R14)
                inc.b   R10
                jmp     T14
T13             push.b  R14
                mov.b   #0x05,R14
                mov.b   &Receive_Buf,LED_Buf(R14) ;把收到的数据放到 LED_Buf[5]
                pop.b   R14
                clr.b   &R_Flag                     ;清除收到数据标识
                jmp     T12
;-------------------------------------------------------------------------------------------------------
;   初始化 Basic Timer
;-------------------------------------------------------------------------------------------------------
init_BT:
                mov.b   #0x16,&BTCTL
                bis.b   #0x80,&IE2
                ret
;-------------------------------------------------------------------------------------------------------
;  Basic timer 的中断函数
;-------------------------------------------------------------------------------------------------------
BT_Interrupt:
                push.w  R13                         ;保存寄存器
                push.w  R12                         ;
                push.w  R15                         ;
                push.w  R14                         ;
                call    #led_Display                ;LED 显示
                pop.w   R14                         ;
```

```
                pop.w   R15                     ;寄存器值出栈
                pop.w   R12
                pop.w   R13
                reti
;-------------------------------------------------------------------------------------------------------
; 移位操作
;-------------------------------------------------------------------------------------------------------
Shift_L:
                tst.b   R14
                jeq     EXIT_S
SHIFT           rla.w   R12
                dec.b   R14
                jne     SHIFT
EXIT_S          ret
;-------------------------------------------------------------------------------------------------------
;  硬件乘法器操作
;-------------------------------------------------------------------------------------------------------
H_Mul:
                push.w  SR
                dint
                nop
                mov.w   R12,&MPY
                mov.w   R14,&OP2
                mov.w   &RESLO,R12
                mov.w   &RESHI,R13
                reti
;-------------------------------------------------------------------------------------------------------
;   初始化 RS485
;-------------------------------------------------------------------------------------------------------
init_485:
        bic.b   #0x40,&FLL_CTL0         ;
        bis.b   #0x1,&U0CTL             ;
        bis.b   #0x10,&U0CTL            ;8-bit 字符
        bis.b   #0x34,&FLL_CTL1         ;开启第二个振荡器
CLEAR_F bic.b   #0x2,&IFG1              ;清除 OSCFault 标识
        mov.b   #0xFF,R14               ;
T1      cmp.b   #0x1,R14                ;
        jnc     T2
        add.b   #0xFF,R14
        jmp     T1
T2      bit.b   #0x8,&FLL_CTL0          ;检测第二个振荡器是否正常工作
        jc      CLEAR_F                 ;
        bis.b   #0x1,&U0CTL             ;
        bis.b   #0x10,&U0CTL            ;
```

```
            mov.b    #0x30,&U0TCTL          ;uclk=smclk
            mov.b    #0x87,&U0BR0           ;在 6MHz 下进行 9600 波特率通讯
            mov.b    #0x02,&U0BR1           ;
            mov.b    #0x03,&U0MCTL          ;
            bic.b    #0x1,&U0CTL            ;
            bis.b    #0xC0,&ME1
            bis.b    #0x40,&IE1             ;使能接收
            clr.b    &IFG1
            bis.b    #0x30,&P2SEL           ;设置 TX,RX
            and.b    #0xCF,&P2DIR
            bis.b    #0x10,&P2DIR
            clr.b    &Receive_Buf           ;清零接收缓冲
            clr.b    &Send_Buf              ;清零发送缓冲
            bis.b    #0x4,&P4DIR            ;
            bic.b    #0x4,&P4SEL            ;
            bic.b    #0x4,&P4OUT            ;
            ret
;-------------------------------------------------------------------------------------------------------
;   发送数据
;-------------------------------------------------------------------------------------------------------
rs485_SendData:
            bis.b    #0x4,&P4OUT            ;使能发送
            mov.b    &Send_Buf,&U0TXBUF
T3          bit.b    #0x1,&U0TCTL
            jnc      T3
            bic.b    #0x4,&P4OUT            ;禁止发送
            ret
;-------------------------------------------------------------------------------------------------------
;    接收数据
;-------------------------------------------------------------------------------------------------------
data_Receive:
            mov.b    &U0RXBUF,&Receive_Buf
            mov.b    #0x1,&R_Flag
            reti
;-------------------------------------------------------------------------------------------------------
;    初始化 LED
;-------------------------------------------------------------------------------------------------------
init_LED:
            mov.b    #0xFF,&P3DIR           ;设置 P3DIR
            clr.b    &P3OUT                 ;
            bis.b    #0x3,&P4DIR            ;
            and.b    #0xFC,&P4OUT           ;
            clr.b    &LED_CTRL              ;
            clr.b    R14                    ;
```

```
T4          cmp.b    #0x6,R14                ;
            jc       EXIT_I                  ;
            mov.b    R14,R15                 ; 初始化  LED_Buf
            and.w    #0xFF,R15               ;
            clr.b    LED_Buf(R15)            ;
            inc.b    R14                     ;
            jmp      T4                      ;
EXIT_I      ret
;-------------------------------------------------------------------------------------------------------------
;    LED 显示数据
;-------------------------------------------------------------------------------------------------------------
led_Display:
                mov.w    #0x1,R15                    ;
                mov.b    &LED_CTRL,R14               ;   设定 LED 控制位
                and.w    #0xFF,R14                   ;
                mov.b    LED_Buf(R14),R14            ;   获取 LED_Buf 的内容
                and.w    #0xFF,R14                   ;
                mov.b    NUM_LED(R14),&P3OUT         ;   设定 LED 显示段码
                bis.b    #0x2,&P4OUT                 ;
                bic.b    #0x2,&P4OUT                 ;
                mov.b    R15,R12                     ;
                mov.b    &LED_CTRL,R14               ;
                call     #Shift_L                    ;   改变控制位
                inv.b    R12                         ;
                mov.b    R12,&P3OUT                  ;
                bis.b    #0x1,&P4OUT                 ;
                bic.b    #0x1,&P4OUT                 ;
                inc.b    &LED_CTRL                   ;
                cmp.b    #0x6,&LED_CTRL              ;   如果 LED_Ctrl >5 ,LED_Ctrl=0
                jnc      EXIT_D                      ;
                clr.b    &LED_CTRL                   ;
EXIT_D          ret
;-------------------------------------------------------------------------------------------------------------
; 初始化 keyboard
;-------------------------------------------------------------------------------------------------------------
init_Keyboard:
                and.b    #0x1,&P1DIR                 ; 设置 P1
                bis.b    #0xE,&P1DIR                 ;
                bis.b    #0xE,&P1OUT                 ;
                clr.b    &KEY_Flag                   ;   清零  key_Flag
                clr.b    &KEY_Pressed                ;   清零  key_Pressed
                ret
;-------------------------------------------------------------------------------------------------------------
;    获取键值
```

```
;-------------------------------------------------------------------------------------------------------------
check_Key:
                push.w    R10                    ; 进栈
                push.w    R11                    ;
                push.w    R8                     ;
                push.w    R9                     ;
                mov.b     #0x8,R8                ;
                clr.b     R10                    ;
T9              cmp.b     #0x3,R10               ;  检测行
                jc        T5                     ;
                bis.b     #0xE,&P1OUT            ;
                sub.b     R8,&P1OUT              ;
                clrc                             ;
                rrc.b     R8                     ;
                mov.b     &P1IN,R14              ;
                and.b     #0xF0,R14              ;
                cmp.b     #0xF0,R14              ;
                jc        T6                     ;
                mov.b     #0x80,R9               ;
                clr.b     R11                    ;
T8              cmp.b     #0x4,R11               ; 检测列
                jc        T6                     ;
                mov.b     &P1IN,R14              ;
                and.b     R9,R14                 ;
                tst.b     R14                    ;
                jne       T7                     ;
                mov.b     R10,R12                ;
                and.w     #0xFF,R12              ;
                mov.w     #0x4,R14               ;
                call      #H_Mul                 ;
                mov.b     R11,R14                ;
                and.w     #0xFF,R14              ;
                add.w     R12,R14                ;
                mov.b     KEY_MAP(R14),&KEY_Val ; 获取键值
                jmp       T5                     ;
T7              clrc                             ;
                rrc.b     R9                     ;
                inc.b     R11                    ;
                jmp       T8                     ;
T6              inc.b     R10
                jmp       T9                     ;
T5              pop.w     R9                     ; 出栈
                pop.w     R8                     ;
                pop.w     R11                    ;
```

```
                    pop.w     R10                   ;
                    ret

delay:
                    mov.w     #0xFF,R14             ;
T10                 tst.w     R14                   ;
                    jeq       EXIT_Delay            ;
                    add.w     #0xFFFF,R14           ;
                    jmp       T10                   ;
EXIT_Delay          ret                             ;
;-------------------------------------------------------------------------------------------------------------
;   检测键盘是否有按键按下
;-------------------------------------------------------------------------------------------------------------
key_Event:
                    push.w    R10                   ;
                    and.b     #0x1,&P1OUT           ;
                    mov.b     &P1IN,R10             ;
                    tst.b     &KEY_Pressed          ; 测试是否有键按下
                    jne       T11                   ;
                    mov.b     R10,R14               ;
                    and.b     #0xF0,R14             ;
                    cmp.b     #0xF0,R14             ;
                    jc        T11                   ;
                    mov.b     #0x1,&KEY_Pressed     ;
                    call      #delay                ; 延时
                    call      #check_Key            ; 获取键值
                    jmp       EXIT_K                ;
T11                 cmp.b     #0x1,&KEY_Pressed     ;
                    jne       EXIT_K                ;
                    and.b     #0xF0,R10             ;
                    cmp.b     #0xF0,R10             ;
                    jne       EXIT_K                ;
                    clr.b     &KEY_Pressed          ;
                    mov.b     #0x1,&KEY_Flag        ; 置位
EXIT_K              pop.w     R10                   ;
                    ret                             ;
;-------------------------------------------------------------------------------------------------------------
;    LED 段码
;-------------------------------------------------------------------------------------------------------------
NUM_LED
                    db    0xd7          ;0
                    db    0x14          ;1
                    db    0xcd          ;2
                    db    0x5d          ;3
```

```
                db      0x1E            ;4
                db      0x5b            ;5
                db      0xdb            ;6
                db      0x15            ;7
                db      0xdf            ;8
                db      0x5f            ;9
                db      0x9f            ;A
                db      0xda            ;B
                db      0xc3            ;C
                db      0xcc            ;D
                db      0xcf            ;E
                db      0x8b            ;F
                db      0x00            ;用于清空 LED 显示
;-------------------------------------------------------------------------------------------------------
;     行列键盘的键值对应表
;-------------------------------------------------------------------------------------------------------
KEY_MAP
                db      0x01            ; 1
                db      0x02            ; 2
                db      0x03            ; 3
                db      0x0A            ; A
                db      0x04            ; 4
                db      0x05            ; 5
                db      0x06            ; 6
                db      0x00            ; 0
                db      0x07            ; 7
                db      0x08            ; 8
                db      0x09            ; 9
                db      0x0b            ; B
;-------------------------------------------------------------------------------------------------------
;               中断向量
;-------------------------------------------------------------------------------------------------------
                ORG     0FFFEh                  ; MSP430 RESET  向量
                DW      RESET                   ;
                ORG     0fff2H                  ; USART  接收中断
                DW      data_Receive            ;
                ORG     0FFE0H                  ; basic Timer 中断
                DW      BT_Interrupt            ;
                END
```

实验十二 SPI接口扩展RF通信

一、实验目的

1、了解RF通信的基本原理

2、了解CC2420射频芯片的工作原理

3、掌握SPI接口扩展RF通信的操作

二、实验原理

1、CC2420 射频芯片介绍

CC2420 是 Chipcon As 公司推出的首款符合 2.4GHz IEEE802.15.4 标准的射频收发器。该器件包括众多额外功能，是第一款适用于 ZigBee 产品的 RF 器件。它基于 Chipcon 公司的 SmartRF 03 技术，以 0.18um CMOS 工艺制成　只需极少外部元器件，性能稳定且功耗极低。CC2420 的选择性和敏感性指数超过了 IEEE802.15.4 标准的要求，可确保短距离通信的有效性和可靠性。利用此芯片开发的无线通信设备支持数据传输率高达 250kbps 可以实现多点对多点的快速组网。

芯片主要性能特点

CC2420 的主要性能参数如下：

●工作频带范围：2.400～2.4835GHz；

●采用 IEEE802.15.4 规范要求的直接序列扩频方式；

●数据速率达 250kbps 码片速率达 2MChip/s；

●采用 o-QPSK 调制方式；

●超低电流消耗（RX:19.7mA,TX:17.4mA）高接收灵敏度（-99dBm）；

●抗邻频道干扰能力强(39dB)；

●内部集成有 VCO、LNA、PA 以及电源整流器　采用低电压供电(2.1～3.6V)；

●输出功率编程可控；

●IEEE802.15.4 ＭＡＣ层硬件可支持自动帧格式生成、同步插入与检测、16bit CRC 校验、电源检测、完全自动 MAC 层安全保护(CTR,CBC－MAC,CCM)；

●与控制微处理器的接口配置容易(4 总线 SPI 接口)；

●开发工具齐全　提供有开发套件和演示套件；

●采用 QLP-48 封装，外形尺寸只有 7 × 7 ｍｍ。

电路描述

CC2420 是以低-中频(low-IF)接收器为特色。接收 RF 信号由低噪声的放大器放大，再正交转换成中频（IF）。在中频，复杂的 I/Q 信号被过滤和放大，再由 ADC 转换成数字信号。自动获得控制、最终通道过滤、解扩、符号相关性和字节同步都是实现数字化的操作。

对于 CC2420 的操作来说只需要少量的外部元件，其外围电路包括晶振时钟电路、射频输入/ 输出匹配电路和微控制器接口电路三个部分。芯片本振信号既可由外部有源晶体提供,也可由内部电路提供。由内部电路提供时需外加晶体振荡器和两个负载电容，电容的大小取决于晶体的频率及输入容抗等参数。射频输入/ 输出匹配电路主要用来匹配芯片的输入输出阻抗。CC2420 可以通过 4 线 SPI 总线(SI、SO、SCLK、CSn)设置芯片的工作模式，并实现读/ 写缓存数据,读/ 写状态寄存器等。标准应用电路可

参考图 12-2，所需外部元件可参考表 12-1：

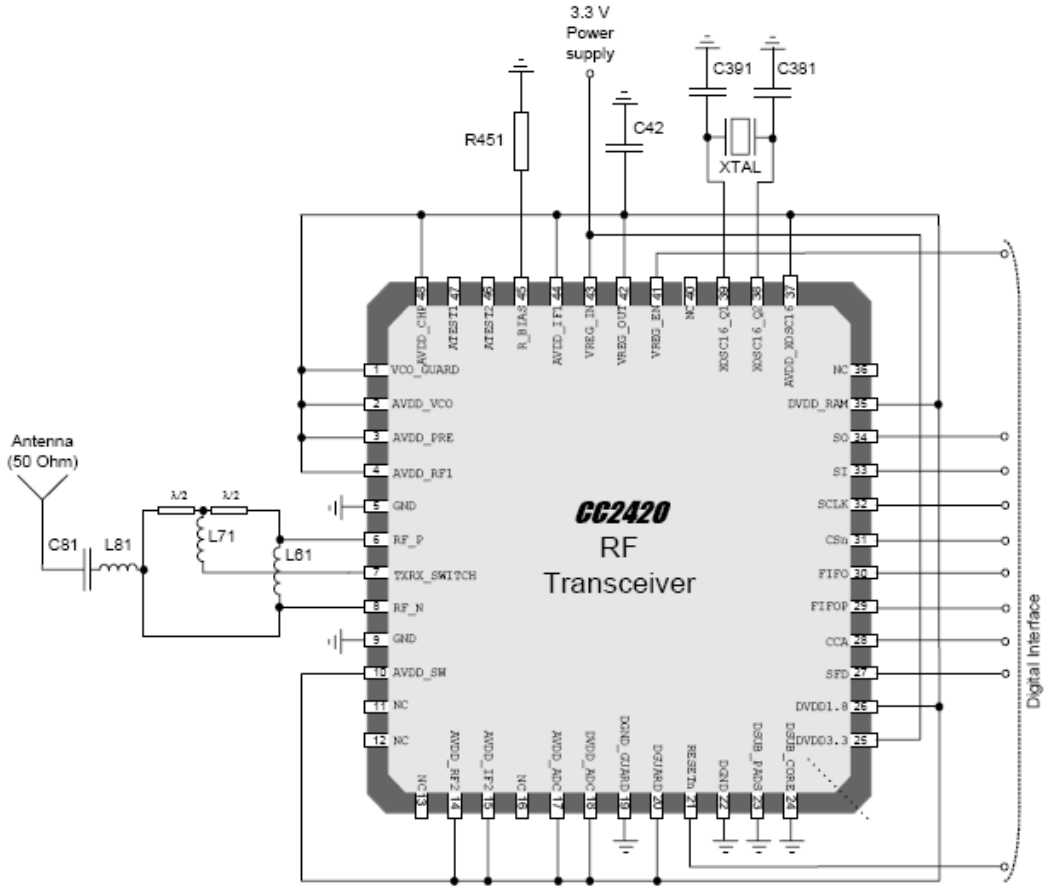


图 12-2 标准应用电路

器件	描述
C42	电压自动调节器 负载电容
C61	不平衡变压器和匹配电路
C62	天线的 DC 和匹配电路
C71	前端偏压比退耦装置和匹配电路
C81	电压自动调节器 负载电容
C381	16MHz 晶振负载电容
C391	16MHz 晶振负载电容
L61	DC 偏压比电路和匹配电路
L62	DC 偏压比电路和匹配电路
L71	DC 偏压比电路和匹配电路
L81	不平衡变压器和匹配电路
R451	电流参考发生器中精确电阻
XTAL	16MHz 晶振

表 12-1 外部元件

2、SPI 接口的 RF 通信操作

实验所用电路示意图见图 12-3

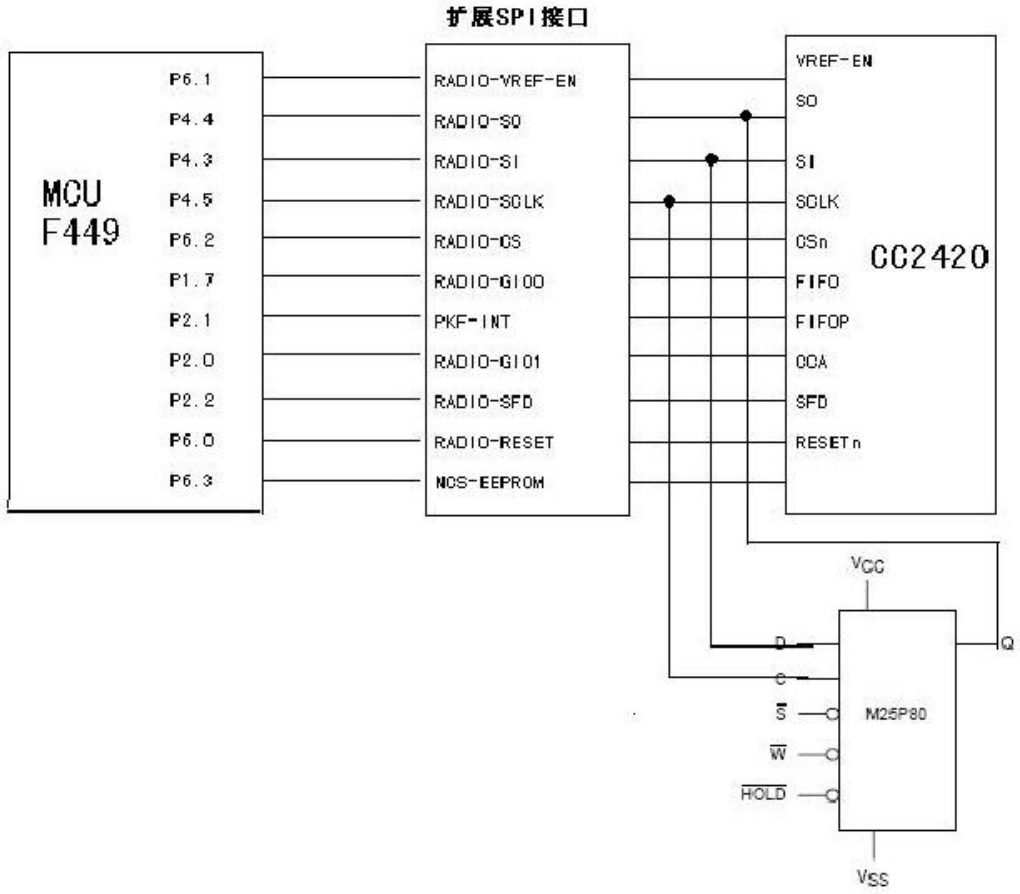


图 12-3 RF 通信电路示意图

CC2420 可以通过配置来为各种不同的应用达到最好的性能。通过可编程配置寄存器可对以下参数进行配置：

- ●接收/发送模式
- ●RF 通道选择
- ●RF 输出功率
- ●电源休眠/唤醒模式
- ●晶振休眠/唤醒
- ●清除通道估计
- ●硬件支持对数据报的处理
- ●加密/解密模式

三、实验内容

利用 SPI 接口进行 CC2420 射频芯片的通信

四、实验步骤

1、烧写发送程序到实验台

2、烧些接收程序到另一实验台，观察是否能进行通信

五、分析与思考

CC2420 遵循的 802.15.4 协议的内容

六、实验参考代码

1．发送程序代码

```
/*********************************************************************
CC2420 RF transmit
*********************************************************************/
void main()
{

tTxPacket *message;
  TOS_Msg *data;
  hardwareinit();
  message = (tTxPacket *)data->Txdata;
  message->val=1;
  message->Data=0x55;
  data->fcflo = CC2420_DEF_FCF_LO;
  data->fcfhi = CC2420_DEF_FCF_HI_ACK;
// destination PAN is broadcast
  data->destpan = TOS_BCAST_ADDR;
  // adjust the destination address to be in the right byte order
  data->addr = 0xffff;
  // adjust the data length to now include the full packet length
  data->length = 19;
  // keep the DSN increasing for ACK recognition
  data->dsn = 1;
  // reset the time field
  data->time = 0;
  sendAMessage(sizeof(message)+18,(char*)data);

}
```

2. 接收程序代码

```
/*********************************************************************
CC2420 RF receive
*********************************************************************/
void main()
{
      WDTCTL = WDTPW + WDTHOLD;                    // Stop WDT
      hardwareinit();
      P2DIR&=~0XFD;
      P2SEL&=~0XFD;
      IE2 &=URXIE1;
      P2IES|=0X02;
      P2IE |= 0x02;
_EINT();
while(1);
```

```
}

#pragma vector = PORT2_VECTOR
__interrupt void Data_Rev()
{
   P2IE &= ~0x02;
   HPLCC2420_readRXFIFO();
   P2IFG &= ~0x02;
   P2IE |= 0x02;

}
```