```
    SPIEXCHANGE(count);
    WAIT_SPI();
    return;
}
// MCP2510 芯片请求发送程序
 void RTS2510(RTSn)
 int RTSn;
{
    a[0]=RTS+RTSn;
    count   1;
    SPIEXCHANGE(count);                    // 发送 MCP2510 芯片,请求发送指令
    WAIT_SPI();
    return;
}
// 读取 MCP2510 芯片的状态
 int GETS2510()
{
    a[0]=STA2510;
    a[1]=0;
    count=2;
    SPIEXCHANGE(count);                    // 读取 MCP2510 芯片状态
    WAIT_SPI();
    b[0]=a[1];                             // 状态存到数组 b[]中
    return;
}
// 对 MCP2510 芯片进行位修改子程序
 void BM2510(adress,mask,data)
{
 int   adress;
 int   mask;
 int   data;
{
    a[0]=BITMOD;                           // 位修改指令
    a[1]=adress;                           // 位修改寄存器地址
    a[2]=mask;                             // 位修改屏蔽位
    a[3]=data;                             // 位修改数据
    count=4;
    SPIEXCHANGE(count);
```

```
    WAIT_SPI();
    return;
}
// 设置 MCP2510 芯片为正常操作模式
void  SETNORMAL()
{
    int  k=1;
    BM2510(CANCTRL,0xe0,0x00);      // 设置为正常操作模式
    do    {
      RD2510(CANSTAT,1);
      k=b[0]&0xe0;
    }while(k);                      // 确认已进入正常操作模式
    return;
}
// 对 MCP2510 进行初始化
void INIT2510()
{
    RESET2510();                    // 使芯片复位
    b[0]=0x02;
    b[1]=0x90;
    b[2]=0x07;
    WR2510(CNF3,3);                 // 波特率为 125 kbps
    b[0]=0x00;
    b[1]=0x00;
    WR2510(RXM0SIDH,2);
    b[0]=0x00;
    b[1]=0x00;
    WR2510(RXF0SIDH,2);             // RX0 接收,屏蔽位为 0,过滤器为 0
    b[0]=0x00;
    WR2510(CANINTE,1);              // CAN 中断不使能
    SETNORMAL();                    // 设置为正常操作模式
    return;
}
// MCP2510 芯片发送完成与否判断,邮箱号为 adress
void TXCOMPLETE(adress)
int adress;
{
```

```
    int k=1;
    do      {
      RD2510(adress,1);
      k=b[0]&0x08;
    }while(k);                              // 确认是否已发送完毕 to add CLRWDT
    return;
}
// 初始化 PIC16F877 芯片
void INIT877()
{
    PORTA=0;
    PORTB=0;
    PORTC=0;
    PORTD=0;
    PORTE=0;
    TRISA=0xff;
    TRISB=0xfd;
    TRISC=0xd7;                             // SCK,SDO 输出,SDI 输入
    TRISD=0;
    TRISE=0x03;                             // 片选 CS 信号输出
    PORTA=0xff;
    PORTB=0x03;                             // RST=1
    PORTC=0;
    PORTD=0xff;
    PORTE=0x04;
    return;
}
// 初始化 SPI 接口
 void INITSPI()
{
    SSPCON=0x11;
    SSPEN=1;                                // SSP 使能
    SSPSTAT=0;
    return;
}
// 发送数据子程序
 void   TXMSG(int DLC)
```

```
    ;
    for(i=0;i<DLC;i++)   b[i]=c[i];
    WR2510(TXB0D0,DLC);
    b[0]=DLC;
    WR2510(TXB0DLC,1);
    b[0]=0x03;
    b[1]=RecID_H;
    b[2]=RecID_L;
    WR2510(TXB0CTRL,3);
    RTS2510(0x01);                    // 请求发送
    TXCOMPLETE(TXB0CTRL);             //等待发送完毕
    return;
}
// 接收数据子程序
 int RXMSG()
{
    int k;
    RD2510(CANINTF,1);
    k=b[0]&0x01;
    if(k==1)      {
        BM2510(CANINTF,0x01,0x00);
        RD2510(RXB0SIDH,2);
        RecID_H=b[0];
        RecID_L=b[1]&0xe0;
        RD2510(RXB0DLC,1);
        DLC=b[0]&0x0f;
        RD2510(RXB0D0,DLC);
        for(i=0;i<DLC;i++) c[i]=b[1];
        return 1;
    }
    return 0;
}
```

第 10 章　利用 CCP 模块设计频率计

PIC16F877 单片机内部集成有捕捉/比较/脉宽调制 PWM(CCP)模块。当 CCP1 工作在捕捉(capture)方式时，可捕捉外部输入脉冲的上升沿或下降沿，并产生相应的中断。利用 CCP 模块的该功能，便可方便地完成一组数字信号的频率、周期、脉宽、占空比等参数的测量。该应用实例就是依据这个原理实现的。

10.1　CCP 模块的捕捉工作方式简介

对 CCP1 的操作和对 CCP2 的操作完全一样，只是各自的特殊触发器不同；因此下面仅以 CCP1 的操作为例，说明 CCP 模块的 capture 工作方式。图 10.1 为捕捉(capture)工作方式结构示意图。

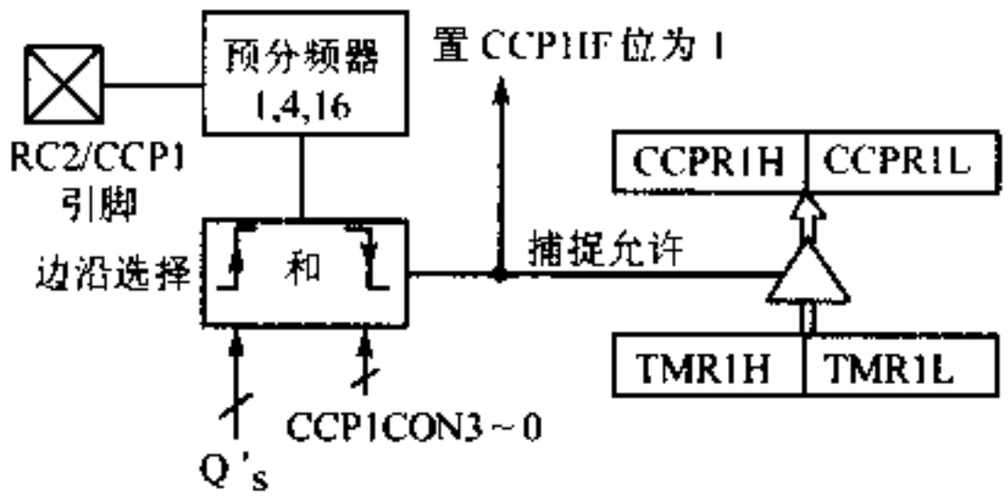


图 10.1　捕捉工作方式结构示意图

当 CCP1 工作在捕捉方式时，一旦有下列事件在 RC2/CCP1 引脚上发生，CCP1 寄存器的 CCPR1H 和 CCPR1L 即捕捉记录下该时刻 TMR1 寄存器中 16 bit 的值。

- 每个脉冲下降沿；
- 每个脉冲上升沿；
- 每 4 个脉冲上升沿；
- 每 16 个脉冲上升沿。

由控制位 CCP1M3～CCP1M0（即 CCP1CON 中的 bit 3～bit 0）设置决定 4 种不同触发控制方式。当捕捉事件发生后，中断请求标志位 CCP1IF（PIR1 寄存器的 *bit 2*）置为 *1*，*该位必须由软件清 0*。如果寄存器 CCPR1 中的值被 CPU 读出之前，又发生另一个新的捕捉事件，那么原来捕捉的值就会被丢失(即被新的值所覆盖)。

在捕捉工作方式下，要先设置 RC2/CCP1 引脚为输入状态，靠设置 TRISC 的 bit 2 为 1 实现。

当需要 CCP 工作在捕捉工作方式时，TMR1 必须设置在定时或同步计数方式下工作；如果 TMR1 工作在异步计数方式下，CCP 则不能工作在捕捉工作方式。

CCP1CON 寄存器控制 CCP1 的操作，其各位的定义如下。

CCP1CON 控制寄存器(地址 17h)和 CCP2CON 控制寄存器(地址 1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
--	--	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7～bit 6——未用，读出时为 0。

bit 5～bit 4——CCPxX～CCPxY——PWM 工作循环周期的低 2 bit。

捕捉工作方式：未用；

比较工作方式：未用；

PWM 工作方式：作为其工作循环周期的低 2 bit，高 8 bit 在 CCPRxL 中。

bit 3～bit 0-- CCPxM3～ CCPxM0——-CCPx 工作方式选择位。

0000 = 关闭捕捉/比较/脉宽调制模块(即 CCPx 复位)。

0100 = 捕捉工作方式，捕捉每个脉冲下降沿。

0101 = 捕捉工作方式，捕捉每个脉冲上升沿。

0110 = 捕捉工作方式，捕捉每 4 个脉冲上升沿。

0111 = 捕捉工作方式，捕捉每 16 个脉冲上升沿。

1000 = 比较工作方式，输出匹配使 RC2/CCPx 引脚为高电平(CCPxIF 置 1)。

1001 = 比较工作方式，输出匹配使 RC2/CCPx 引脚为低电平(CCPxIF 置 1)。

1010 = 比较工作方式，输出匹配产生软件中断(CCPxIF 置 1，CCPx 引脚不受影响)。

1011 = 比较工作方式，特殊事件触发(CCPxIF 置 1；CCP1 将 TMR1 复位；CCP2 将 TMR1 复位，并且启动模/数转换电路(如果 A/D 模块是使能的))。

11xx = 脉宽调制 PWM 工作方式。

10.2 设计要求

测试对象为 100～1 000 Hz 的 TTL 电平信号，要求对其以下的参数进行测量：

① 频率测量，测量误差小于 0.1%；

② 周期测量，测量误差小于 0.1%；

③ 脉冲宽度(高电平持续时间)测量，脉冲宽度大于 100 μs，测量误差小于 1%；

④ 占空比测量，占空比的变化范围为 10%～90%，要求测量误差小于 1%。

10.3　硬件原理图

本例的硬件电路很简单。外部的频率信号从 PIC16F877 的 CCP1 脚输入；人机对话所需的键盘和 LED 显示器都可以在实验模板上直接得到。请读者参考本书相关章节。

10.4　设计与测试原理

测试原理如图 10.2 所示。在 t1 时刻以前，把 CCP1 设置成捕捉脉冲的上升沿；当信号上升沿来到时，发生 CCP 中断，在中断服务程序中捕捉记下此时 TMR1 寄存器中 16 bit 的值 TMR11，把 CCP1 设置成捕捉脉冲的下降沿；当该信号下降沿来时，又发生 CCP 中断，又在中断服务程序中记下此时 TMR1 寄存器中 16 bit 的值 TMR12，然后又把 CCP1 设置成捕捉脉冲的上升沿；当该信号的又一上升沿来时，又发生 CCP 中断，在中断服务程序中记下此时 TMR1 寄存器中 16 bit 的值 TMR13；则信号的周期 T=(TMR13－ TMR11)μs，信号的频率 f= 1/T，脉冲的宽度 Tp=(TMR12－ TMR11) μs，占空比 D= Tp/T * 100%。因为 CCP 捕捉方式可能发生的最大误差为±1 μs，考虑到需要满足的误差要求，可把以上过程多进行几次，再把各次测试的平均值作为最后的测量值。

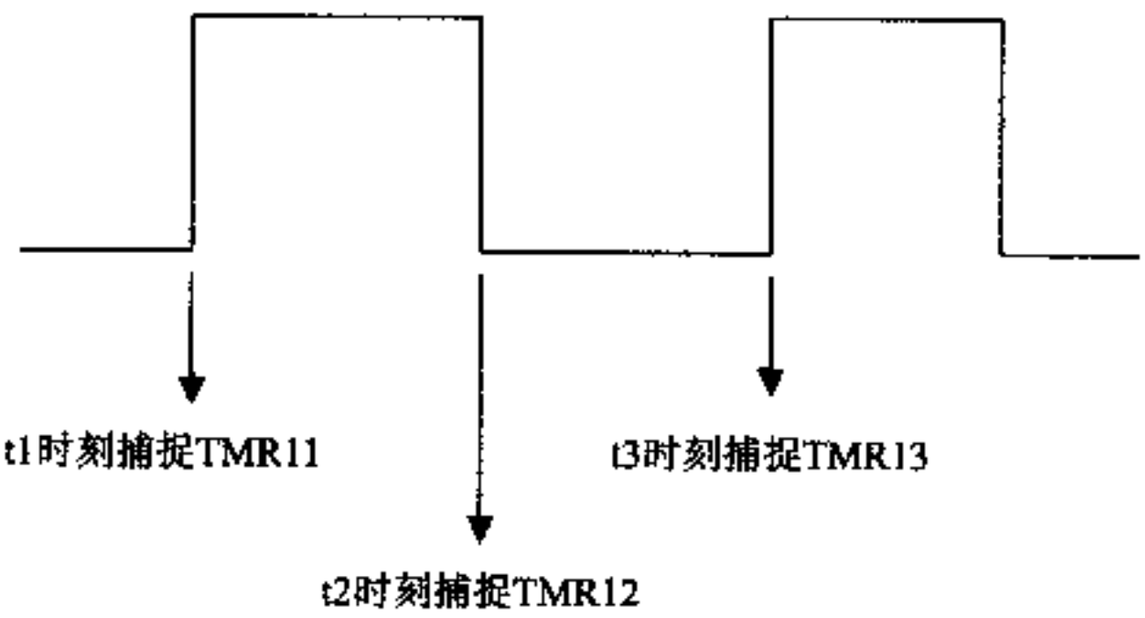


图 10.2　CCP 模块测试原理图

特别需要注意的一点是，2 次中断的时间间隔必须大于 1 次中断服务的执行时间；否则如果在中断服务程序执行时又发生 CCP 中断，就不能正常工作。本题需要测量的最高频率为 1 000 Hz，周期只有 1 000 μs，且占空比的变化范围为 10%～90%，则高低电平持续的最短时间都为 100 μs，可以有充分的时间执行中断服务程序。如果实际应用中发现 2 次捕捉中断的时间间隔小于 1 次中断服务时间，则可以通过适当设置寄存器 CCP1CON 的值，使 CCP 模块每 4 个脉冲上升沿捕捉 1 次或每 16 个脉冲上升沿捕捉 1 次，这样 2 次中断的时间间隔就增大

了；如果还不能达到要求，则可用分频器对输入频率信号分频处理后，再由 CCP1 引脚输入。

在工程实际中，经常会遇到测量一个旋转机械转速的情况。如果用适当的传感器（如红外线发射接收管）把转速信号转化为电脉冲信号，则利用此法也可以方便地实现转速的测量；也可以与光栅编码器直接接口，以测量转速。

10.5 程序设计

10.5.1 人机对话的设计

因为本例要测试 4 种参数，并且可以利用键盘选择显示其中的任何 1 种；因此编程设定 S9 为加 1 键、S11 为减 1 键、S10 为确定键、S12 为功能键。当按下 S9 键时，程序中的一个寄存器 COUNTER 可以从 1 连续或点动步进地加到 4；同理按下减 1 键时，COUNTER 从 4 减到 1，并且把 COUNTER 值显示在 LED 上；而不同的 COUNTER 值对应不同的测试参数。程序根据不同的 COUNTER 值把相应的参数（已转换成 BCD 码）送到显示缓冲区，此时按下确定键，则 LED 上显示出该 COUNTER 值对应的测试参数；按下功能键时，LED 上又显示出当前的 COUNTER 值，又可以对其进行加减操作，从而选择显示不同的参数。COUNTER 值与其对应的测试参数关系表如下：

COUNTER 的值	1	2	3	4
对应的测试参数	频　率	周　期	占 空 比	脉　宽

10.5.2 显示缓冲区的应用

如人机对话设计所述，该程序中有 5 个不同的显示任务（显示 COUNTER、频率、周期、占空比、脉宽）。如果编制 5 个不同的显示子程序，会使程序复杂化。在本例中，引入一个“显示缓冲区”：如果没有按下确定键，则把 COUNTER 的值送到显示缓冲区；如果按下了确定键，则根据 COUNTER 的实际值，把相应的测试参数的结果送到显示缓冲区。利用此方法，可以只编制一个显示程序，该程序只执行从显示缓冲区取数显示的任务。至于是显示 COUNTER，还是显示参数测试结果，显示程序不再理会，只需将显示缓冲区的内容显示出来即可。

10.5.3 程序流程图

图 10.3 为主程序及中断服务程序流程图。

中断服务程序中“把 CCP1 模块变成捕捉相反的脉冲边沿”含义为：若在本次中断以前 CCP1 捕捉脉冲的上升沿，则把它改成捕捉脉冲下降沿；若在本次中断以前 CCP1 捕捉脉冲的下降沿，则把它改成捕捉脉冲上升沿。实际编制程序时，为了充分提高 CPU 的工作效率，利用等待 CCP1 中断的时间，进行键盘的消抖动延时（为了使测试较为精确，测试 10 个脉冲的参

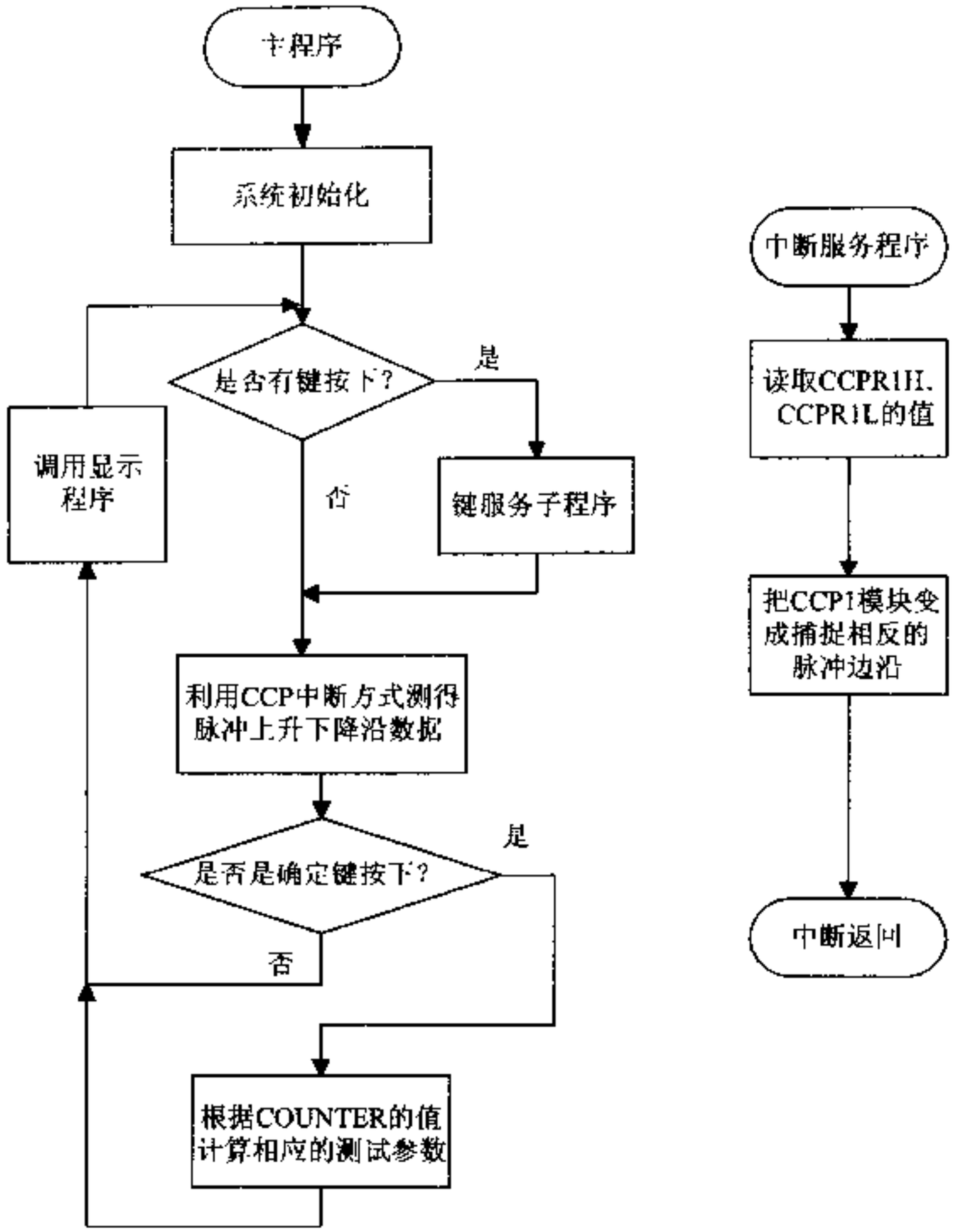


图 10.3　主程序及中断服务程序

数，再取平均值，则等待 CCP1 中断的时间为 0.01～0.1 s)。键盘服务子程序流程图见图 10.4。

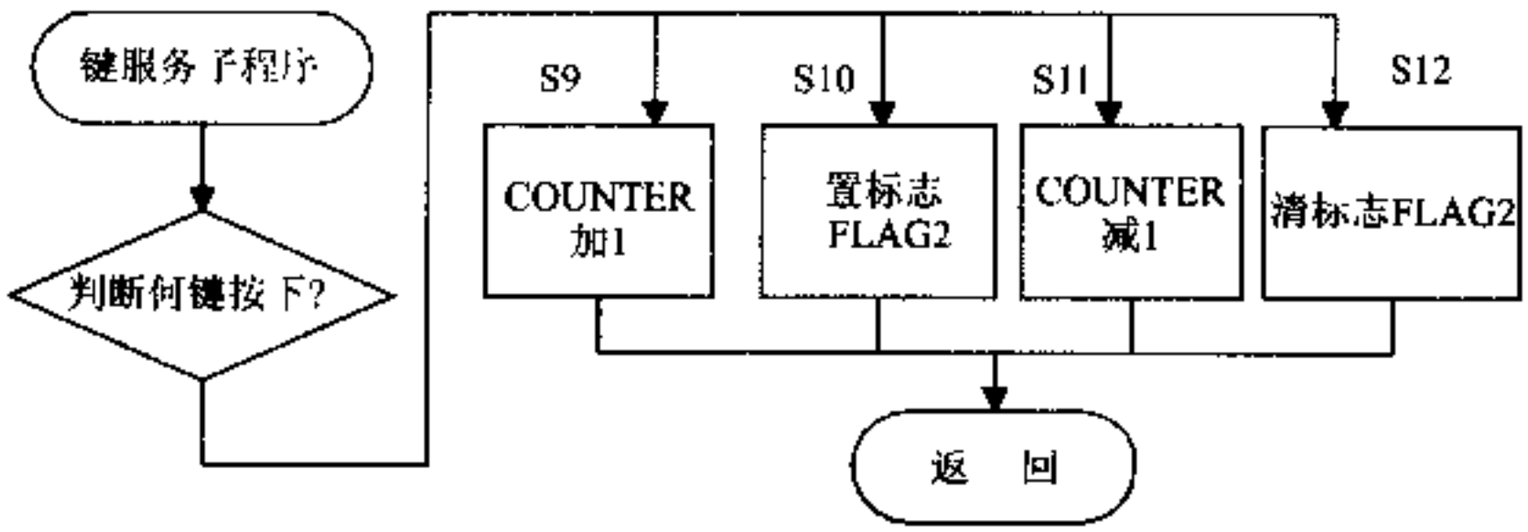


图 10.4　键服务子程序

10.5.4 程序清单

```
#include        <pic.h>
#include        <stdio.h>
#include        <math.h>
//本程序利用 CCP1 模块实现 一个"简易数字频率计"的功能
const  char     table[11]  {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XI)8,0x80,0x90,0xFF};
//不带小数点的显示段码表
const  char     table0[11]= {0X40,0X79,0X24,0X30,0X19,0X12,0X02,0X78,0X00,0X10,0xFF};
//带小数点的显示段码表
bank3    int    cp1z[11];                    //定义一个数组,用于存放各次的捕捉值
union    cp1
{int        y1;
   unsigned  char    cple[2];
}cp1u;                                       //定义 一个共用体
unsigned    char    COUNTW,COUNT;            //测量脉冲个数寄存器
unsigned    char    COUNTER,data,k;
unsigned    char    FLAG @ 0XEF;
#define FLAGIT(adr,bit)  ((unsigned)(&adr) * 8+(bit))     //绝对寻址位操作指令
static    bit FLAG1   @ FLAGIT(FLAG,0);
static    bit FLAG2   @ FLAGIT(FLAG,1);
static    bit FLAG3   @ FLAGIT(FLAG,2);
unsigned    char    s[4];                    //定义一个显示缓冲数组
int         T5 ,uo;
double    RE5;
double    puad5;
//spi 方式显示初始化子程序
void SPIINIT()
{
    PIR1=0;
    SSPCON=0x30;
    SSPSTAT=0xC0;
//设置 spi 的控制方式,允许 SSP 方式,并且时钟下降沿发送,与"74HC595,当其
//"SCLK 从低到高跳变时,串行输入寄存器"的特点相对应
    TRISC  0xD7;                             //SDO 引脚为输出,SCK 引脚为输出
    TRISA5=0;                                //RA5 引脚设置为输出,以输出显示锁存信号
    FLAG1=0;
```

```
    FLAG2=0;
    FLAG3=0;
    COUNTER=0X01;
}
//CCP 模块工作于捕捉方式初始化子程序
void    ccpint( )
{
    CCP1CON=0X05;                    //首先设置 CCP1 捕捉每个脉冲的上升沿
    T1CON=0X00;                      //关闭 TMR1 振荡器
    PEIE=1;                          //外围中断允许(此时总中断关闭)
    CCP1IE=1;                        //允许 CCP1 中断
    TRISC2=1;                        //设置 RC2 为输入
}
//系统其他部分初始化子程序
void    initial( )
{
    COUNT=0X0B;                      //为保证测试精度,测试 5 个脉冲的参数后
                                     //求平均值,每个脉冲都要捕捉其上升、下降沿,
                                     //故需要有 11 次中断
    TRISB1=0;
    TRISB2=0;
    TRISB4=1;
    TRISB5=1;                        //设置与键盘有关的各口的输入、输出方式
    RB1=0;
    RB2=0;                           //建立键盘扫描的初始条件
}
//SPI 传输数据子程序
void            SPILED(data)
{
    SSPBUF=data;                                    //启动发送
    do      {
            ;
    }while(SSPIF==0);
    SSPIF=0;
}
//显示子程序,显示 4 bit 数
void    display( )
```

```
{
    RA5=0;                                          //准备锁存
    for(COUNTW=0;COUNTW<4;COUNTW++){
        data=s[COUNTW];
        data=data&0x0F;
        if(COUNTW==k)      data=table0[data];       //第 2 位需要显示小数点
        else     data=table[data];
        SPILED(data);                               //发送显示段码
    }
    for(COUNTW=0;COUNTW<4;COUNTW++){
        data=0xFF;
        SPILED(data);                               //连续发送 4 个 DARK,使显示好看 些
    }
    RA5=1;                                          //最后给一个锁存信号,代表显示任务完成
}
//键盘扫描子程序
void    keyscan( )
{
    if((RB4==0)||(RB5==0))     FLAG1=1;             //若有键按下,则建立标志 FLAG1
    else     FLAG1=0;                               //若无键按下,则清除标志 FLAG1
}
//键服务子程序
void          keyserve( )
{
    PORTB=0XFD;
    if(RB5==0)     data=0X01;
    if(RB4==0)     data=0X03;
    PORTB=0XFB;
    if(RB5==0)     data=0X02;
    if(RB4==0)     data=0X04;                       //以上确定是哪个键按下
    PORTB=0X00;                                     //恢复 PORTB 的值
    if(data==0x01)     {
        COUNTER=COUNTER+1;                          //若按下 S9 键,则 COUNTER 加 1
        if(COUNTER>4)     COUNTER=0x01;             //若 COUNTER 超过 4,则又从 1 计起
    }
    if(data==0x02)     {
        COUNTER=COUNTER-1;                          //若按下 S11 键,则 COUNTER 减 1
```

```
        if(COUNTER<1)      COUNTER=0x04;                    //若 COUNTER 小于 1,则又循环从 4 计起
        }
        if(data==0x03)     FLAG2=1;                         //若按下 S10 键,则建立标志 FLAG2
        if(data==0x04)     FLAG2=0;                         //若按下 S12 键,则清除标志 FLAG2
}
//中断服务程序
void      interrupt      cp1int(void)
{
        CCP1IF=0;                                           //清除中断标志
        cp1u.cp1e[0]=CCPR1L;
        cp1u.cp1e[1]=CCPR1H;
        cp1z[data]=cp1u.y1;                                 //存储一次捕捉值
        CCP1CON=CCP1CON^0X01;                               //把 CCP1 模块改变成捕捉相反的脉冲沿
        data++;
        COUNT--;
}
//周期处理子程序
void        PERIOD( )
{
        T5=cp1z[10]-cp1z[0];                                //求得 5 个周期的值
        RE5=(double)T5;                                     //强制转换成双精度数
        RE5=RE5/5;                                          //求得平均周期,单位为 μs
}
//频率处理子程序
void        FREQUENCY( )
{
        PERIOD( );                                          //先求周期
        RE5=1000000/RE5;                                    //求周期值倒数,再乘以 1 000 000,得频率,
                                                            //单位为 Hz
}
//脉宽处理子程序
void          PULSE( )
{
      int      pu;
      for(data=0,puad5=0;data<=9;data++)      {
            pu=cp1z[data+1]-cp1z[data];
            puad5=(double)pu+puad5;
```

```
        data=data+2;
    }                                          //求得5个脉宽的和值
    RE5=puad5/5;                               //求得平均脉宽
}
//占空比处理子程序
void        OCCUPATIONAL( )
{
    PULSE( );                                  //先求脉宽
    puad5=RE5;                                 //暂存脉宽值
    PERIOD();                                  //再求周期
    RE5=puad5/RE5;                             //求得占空比
}
//主程序
main( )
{
    SPIINIT( );                                //SPI方式显示初始化
    while(1) {
        ccpint();                              //CCP模块工作于捕捉方式初始化
        initial();                             //系统其他部分初始化
        if(FLAG2==0)     {
            s[0]=COUNTER;                      //第1个存储COUNTER的值
            s[1]=0X0A;
            s[2]=0X0A;
            s[3]=0X0A;                         //后面的LED将显示"DARK"
        }
    display( );                                //调用显示子程序
    keyscan();                                 //键盘扫描
    data=0x00;                                 //存储数组指针赋初值
    TMR1H=0;
    TMR1L=0;                                   //定时器1清0
    CCP1IF=0;                                  //清除CCP1的中断标志,以免中断--打开就进入
                                               //中断
    ei( );                                     //中断允许
    TMR1ON=1;                                  //定时器1开
    while(1){
        if(COUNT==0)break;
    }                                          //等待中断次数结束
```

```
    di();                                   //禁止中断
    TMR1ON=0;                               //关闭定时器
    keyscan();                              //键盘扫描
    if(FLAG1==1)    keyserve();             //若确实有键按下,则调用键服务程序
    if(FLAG2--=0)   continue;               //如果没有按下确定键,则终止此次循环,
                                            //继续进行测量
//如果按下了确定键,则进行下面的数值转换和显示工作
if(COUNTER==0x01) FREQUENCY();              //COUNTER=1,需要进行频率处理
if(COUNTER==0x02) PERIOD();                 //COUNTER=2,需要进行周期处理
if(COUNTER==0x03) OCCUPATIONAL();           //COUNTER=3,需要进行占空比处理
if(COUNTER==0x04) PULSE();                  //COUNTER=4,需要进行脉宽处理
    k=5;
    if(RE5<1){
    RE5=RE5*1000;                           //若 RE5<1,乘以 1 000,保证小数点的精度
    k=0x00;
}
else    if(RE5<10){
    RE5=RE5*1000;                           //若 RE5<10,则乘以 1 000,保证小数点的精度
    k=0x00;
}
else    if(RE5<100){
    RE5=RE5*100;                            //若 RE5<100,则乘以 100,保证小数点的精度
    k=0x01;
}
else    if(RE5<1000){
        RE5=RE5*10;                         //若 RE5<1 000,则乘以 10,保证小数点的精度
        k=0x02;
    }
    else    RE5=RE5;
        uo=(int)RE5;
        sprintf(s,"%4d",uo);                //把需要显示的数据转换成 4 bit ASII 码,且放入
                                            //数组 S 中
        display();
    }
}
```

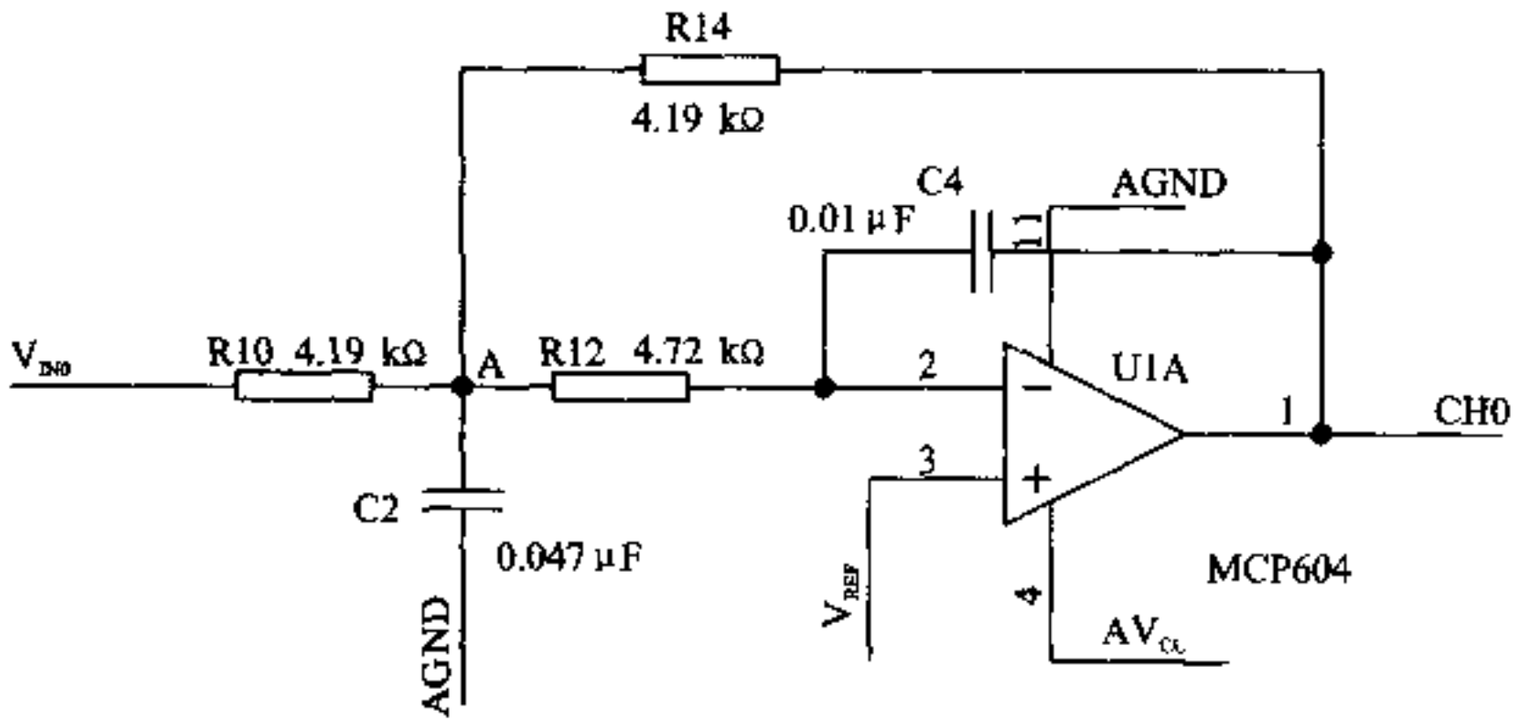
第 11 章　交流电压测量

在工程实际中，常常需要直接测试一些交流信号。该模板设计了交流信号输入电路。

11.1　模拟输入电路

11.1.1　输入电路

A/D转换模块工作时，一般用芯片的工作电压作为 A/D 转换的参考电源（可以为 +5 V）；因此对交流信号而言，需要把双极性输入电压经过提升变成单极性电压，输入调理电路如图 11.1 所示。



截止频率 1 650 Hz

图 11.1　输入调理电路

实际测出，输出的信号 CH0 在输入信号 V_{IN0} 的基础上叠加了一个直流分量，调节 V_{REF} 的值，该直流分量大小可以改变。如果适当调整 V_{REF}，使直流分量为 2 V，则输入为幅值 2 V 的交流正弦信号，输出就为最大值为 4 V、最小值为 0 V 的正弦单极性信号；从而得到了提升的效果，使一般的双极性交流信号变成了适合单片机处理的单极性信号。

11.1.2　系统硬件部分

该系统总的硬件电路如图 11.2 所示。

在实验板上，可通过电位器 R92 调节 V_{REF} 的值，使提升电压为 +2 V，幅值为 2 V 的交流

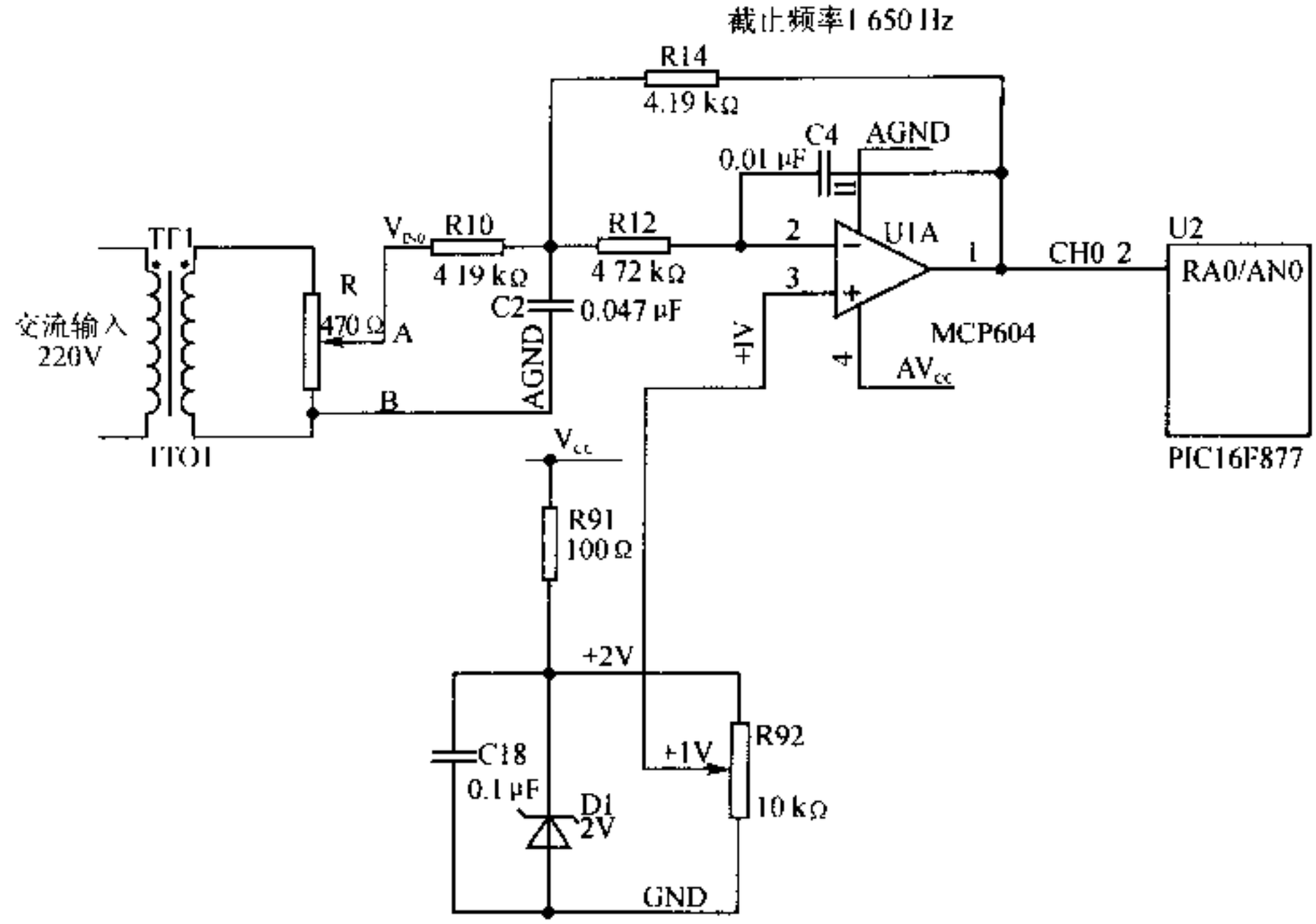


图 11.2　输入电路

输入电压经过提升后，输出 0～4 V 的单极性电压。为保证采样信号能充分地反映模拟信号，必须在一个工频周期内采样足够多的点数，采样点数根据采样定理和所要考虑的谐波次数而定。本例中每个周期各采样 40 个点。以一个 220 V/8 V 的变压器，把 220 V 左右的电网电压转换成 8 V 左右的交流电压，再接上一个 470 Ω 的分压电位器，便可调到幅值为 2 V 的交流电压，经过提升电路变成 0～4 V 的单极性电压信号。在单片机内部，可以简单地编程，减去 +2 V的直流分量；再通过比列运算，使最后的输出为实际电网的电压值。

11.2　数据处理原理

由于 PIC 的 A/D 输入前端加上了变压器、电位器和电压提升电路，故 A/D 采样得到的初始数据需要经过调整变换后，才能得到实际电压值，这可以由简单的数据运算实现。以下的分析与计算皆认为数据已经过了变换。

对交流工频信号的采集，一般以其有效值进行计算，其计算公式为：

$$U = \sqrt{(1/T)\int_0^T u^2(t)\,dt}$$

式中 T 为信号周期。

由于在计算机采集系统中 U(t)是离散值，故应该用下面的计算公式

$$U = \sqrt{\frac{u(1)^2 + u(2)^2 + \cdots + u(n)^2}{n}}$$

式中：u(i)为各次瞬时采样值，i=1,2,…,n；n 为采样次数。

为了能够在 1 个工频周期内采足 40 个点，需要每间隔 500 μs 启动 1 次 A/D 转换。此例中采用 CCP2 的特殊事件触发方式定时达到要求。当 CCP2 工作在比较工作方式时，不断地用 16 bit 的 CCP2 寄存器中的值与 TMR1 寄存器中的值做比较，如果二者相等，CCP2 的特殊事件触发输出将对 TMR1 寄存器对复位，并且启动模/数转换；因此只需在程序的初始化部分对 16 bit 的 CCP2 寄存器赋值为 01F4H(相当于 500 μs)。

11.3 程序流程图及程序清单

11.3.1 程序流程图

图 11.3 为电网电压测试程序流程图。

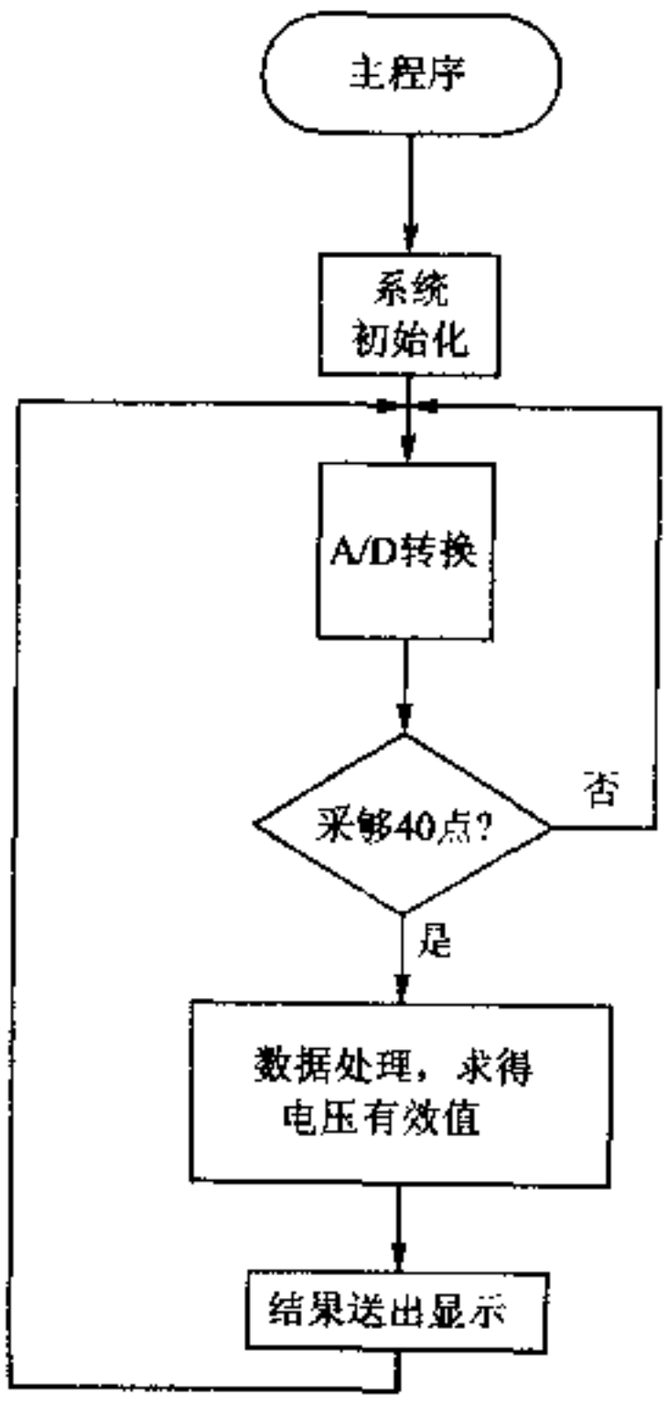


图 11.3 电网电压测试程序流程图

11.3.2　程序清单

该程序已在模板上调试通过，可作为参考。有关显示部分请参考本书相关章节，有关A/D转换的详细设置请参考前面章节。

```
#include        <pic.h>
#include        <math.h>
#include        <stdio.h>
//该程序用于测电网的交流电压有效值，最后的结果将在 4 个 LED 上显示，保留
//1 位小数。
//为了保证调试时数据运算的精确性，需要将 PICC 的 double 型数据选成 32 bit
union     adres
{
    int     yl;
    unsigned     char     adre[2];
}adresult;                              //定义--个共用体
bank3     int     re[40];               //定义存放 A/D 转换结果的数组，在 bank 3 中
unsigned     char     k,data;           //定义几个通用寄存器
double     squ ,squad;                 //平方寄存器和平方和寄存器，squ 又通用为存储其
                                       //他数值
int     uo;
bank1     unsigned  char     s[4];      //此数组用于存储需要显示的字符的 ASII 码
const     char     table[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90};
//不带小数点的显示段码表
const     char  table0[10]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10};
//带小数点的显示段码表
//A/D 转换初始化子程序
void          adinitial()
{
    ADCON0=0x41;                       //选择 A/D 通道为 RA0，且打开 A/D 转换器
                                       //在工作状态，使 A/D 转换时钟为 8$t_{OSC}$
    ADCON1=0X8E;                       //转换结果右移，ADRESH 寄存器的高 6 bit 为"0"
                                       //把 RA0 口设置为模拟量输入方式
    ADIE=1;                            //A/D 转换中断允许
    PEIE=1;                            //外围中断允许
    TRISA0=1;                          //设置 RA0 为输入方式
}
//spi 方式显示初始化子程序
```

```
void                SPIINIT()
{
    PIR1=0;
    SSPCON=0x30;
    SSPSTAT=0xC0;
 //设置 SPI 的控制方式,允许 SSP 方式,并且时钟下降沿发送,与"74HC595,当其
 //SCLK 从低到高跳变时,串行输入寄存器"的特点相对应
    TRISC=0xD7;                          //SDO 引脚为输出,SCK 引脚为输出
    TRISA5=0;                            //RA5 引脚设置为输出,以输出显示锁存信号
}
//系统其他初始化子程序
void    initial()
{
    CCP2IE=0;                            //禁止 CCP 中断
    SSPIE=0;                             //禁止 SSP 中断
    CCP2CON=0X0B;                        //初始化 CCP2CON,CCP2 为特别事件触发方式
    CCPR2H=0X01;
    CCPR2L=0XF4;                         //初始化 CCPR2 寄存器,设置采样间隔 500 μs,
                                         //一个周期内电压采 40 个点
}
//中断服务程序
void          interrupt          adint(void)
{
    CCP2IF=0;
    ADIF=0;                              //清除中断标志
    adresult.adre[0]=ADRESL;
    adresult.adre[1]=ADRESH;             //读取并存储 A/D 转换结果,A/D 转换的结果
                                         //通过共用体的形式放入 f 变量 y1 中
    re[k]=adresult.y1;                   //1 次 A/D 转换的结果存入数组
    k++;                                 //数组访问指针加 1
}
//SPI 传输数据子程序
void        SPILED(data)
{
    SSPBUF=data;                         //启动发送
    do{
        ;
```

```
        }while(SSP1F=·=0);
        SSP1F=0;
    }
    //主程序
    main( )
    {
        adinitial();                    //A/D转换初始化
        SPIINIT();                      //spi 方式显示初始化
        initial();                      //系统其他初始化
        while(1){
        k=0;                            //数组访问指针赋初值
        TMR1H=0X00;
        TMR1L=0X00;                     //定时器 1 清 0
        ei();                           //中断允许
        T1CON=0X01;                     //打开定时器 1
        while(1){
            if(k==40)     break;        //A/D 转换次数达到 40,则终止
        }
        di();                           //禁止中断
    for(k=0;k<40;k++)re[k]=re[k]-0X199;  //假设提升电压为 2 V,对应十六进制数 199H.
                                        //则需在采样值的基础上减去该值
    for(k=0,squad=0;k<40;k++)     {
        uo=re[k];
        squ=(double)uo;                 //强制把采得的数据量转换成双精度数,以便运算
        squ=squ*5/1023;                 //把每点的数据转换成实际数据
        squ=squ*squ;                    //求一点电压的平方
        squad=squad+squ;
    }                                   //以上求得 40 点电压的平方和,存于寄存器 squad 中
    squ=squad/40;                       //求得平均值
    squ=sqrt(squ);                      //开平方,求得最后的电压值
    squ=squ*154.054;                    //通过变压器的变比和分压电阻分配确定该系数
                                        //以上得到了实际电网的电压值
    squ=squ*10;                         //为保证显示的小数点的精度,先对电压值乘以 10
    uo=(int)squ;                        //强制把 U 转换成有符号整型量
    sprintf(s,"%4d",uo);                //通过 sprintf 函数把需要显示的电压数据转换成
                                        //ASII 码,并存于数组 S 中
    RA5=0;                              //准备锁存
```

```
    for(k=0;k<4;k++){
        data=s[k];
        data=data&0X0F;                    //通过按位相与的形式把 ASII 码转换成 BCD 码
        if(k==2)      data=table0[data]; //因为 squ 已乘以 10,则需在第 2 位打小数点
        else      data=table[data];        // table0 存储带小数点的显示段码.
                                           //table 存储不带小数点的显示段码
        SPILED(data);                      //发送显示段码
    }
    for(k=0;k<4;k++)      {
        data=0xFF;
        SPILED(data);                      //连续发送 4 个 DARK,使显示看起来好看一些,这点与
                                           //该实验板的 LED 分布结构有关
    }
    RA5=1;                                 //最后给一个锁存信号,代表显示任务完成
    }
}
```

第 12 章　与 PLC 接口的 4 位 LED 数字显示表

在工业控制应用中，有时需用 LED 数码管显示 PLC 的信息。可使用 PIC 单片机实现一种智能的、带有并行输入接口的数码显示表头。有一种情况就是用 4 个 4 位的 LED 数字显示表头分别显示 PLC 的 4 个参数。

本章就以这种显示表头为例，讲解 PIC 单片机的应用。

PLC 通过 8 点 NPN 输出通道与 4 个 LED 数字显示表头相连。PLC 在 20 ms 或更长一点的时间内更新一次输出数据。在硬件上，每个 LED 数字显示表头都对应一个地址，地址由 2 位跳针设定。要显示的参数为 12 bit 的数据，分 2 帧送到 PIC 单片机。传输的数据包括高位数据帧、低位数据帧标志、地址标志、数据信息，PLC 将 2 帧分 2 次传输，才能传输完 1 个参数。

传输协议为：

第 1 帧	1	A1	A0	B11	B10	B9	B8	B7
第 2 帧	0	B6	B5	B4	B3	B2	B1	B0

PLC 每帧的高位为识别位。每个参数的第 1 帧的高位为 1，第 2 帧的高位为 0。如第 1 帧的 A1A0 地址与该显示表的 2 位地址相同，则更新显示表头所显示的数据。PLC 提供输出 +24 V 电源，4 个表头通过 1 根 16 芯扁平电缆与 1 台 PLC 相连，并要求数字显示表头与 PLC 之间电气隔离。设计时采用了光电隔离方式，光电隔离电路如图 12.1 所示。

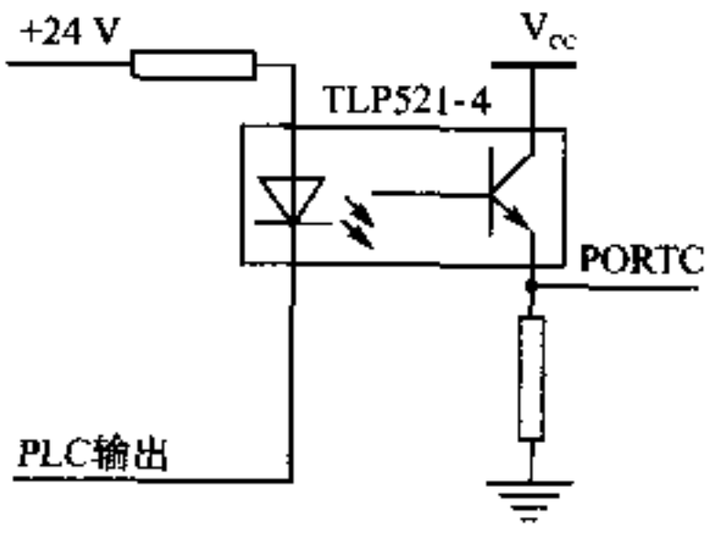


图 12.1　光电隔离电路

本章以用 PIC16F877 加少量外围电路构成的 LED 数字显示表头为例，介绍 PIC 单片机智能表头的硬件电路原理和软件编程方法，并给出全部的程序清单，从硬件设计和软件编程上给读者一些有益的启发和帮助。

12.1　数显表头硬件电路原理

图 12.2 是一个 4 位的 LED 数字显示表头的电路原理图。

PLC 的 8 点 NPN 输出端通过 2 个 TLP521－4 光耦与 PIC16F877 的 PORTC 端口进行信号的隔离传输。如图 12.1 所示，当 PLC 输出高电平时，发光二极管截止，光耦输出端 PORTC 为低电平；当PLC输出低电平时，发光二极管导通，光耦输出端变为高电平。对

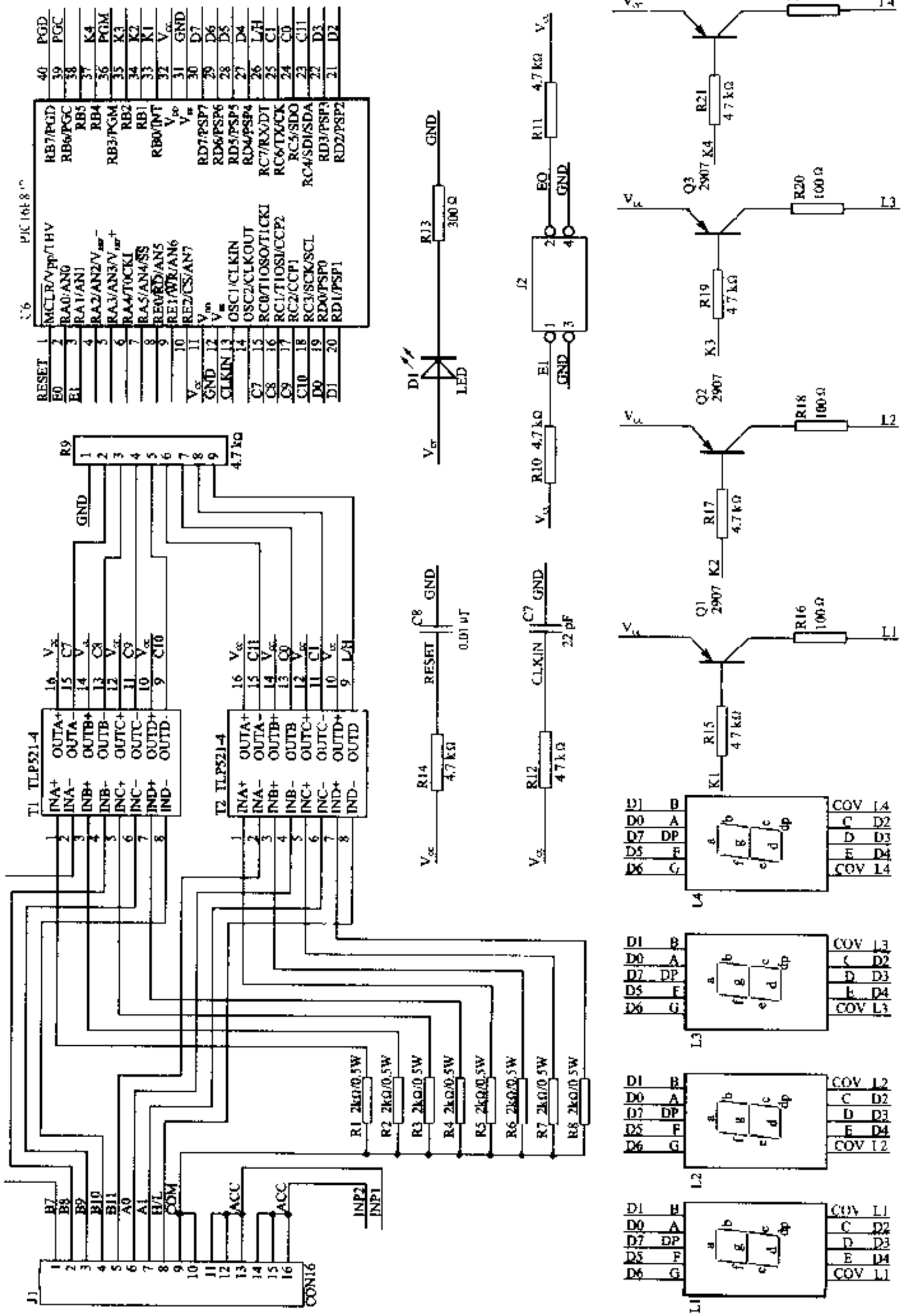


图 12.2 4位 LED 数显表原理

PORTC 端口进行读操作，即可实现数据的输入。在软件中对读到的 PORTC 端口的数据取反，就可以实现信号与 PLC 输出的逻辑一致。数值显示采用 4 只 25.4 mm(1 in)共阳 LED 7 段数码管，由 PIC16F877 以动态扫描方式驱动。

在动态扫描方式下，可以保证有足够的显示亮度和较长的使用寿命。数字显示表的地址跳针设置电路引出的 2 个输出端接到 PIC16F877 的 RA0，RA1 引脚。对这 2 个引脚进行读操作，就可以确定当前数显表设置的表头地址。

PIC16F877 使用 RC 振荡器工作。

12.2　数显表头软件设计思路

由于工业现场的干扰，不能确保 PLC 每 20 ms 发送的数据能够可靠地接收到；所以应用程序中使用循环方式对 PORTC 端口不断地读数据，并进行判断、转换及显示。

考虑到 PLC 输出存在的干扰，编制软件时必须采用抗干扰措施。

在读数据子程序中，一共读 5 次 PORTC 端口，2 次读端口间隔 1 ms，读到的 5 个数据有 4 个相等，即认为读到的数据有效。读到有效数据之后，首先判断读到的数据是第几帧数据。如果读到的是第 1 帧数据，则判断地址位是否与数显表的设置地址相同；地址相同时，才认为读到本表头的第 1 帧数据。读到第 1 帧数据后，对标志寄存器 FLAG 的 F1 位置 1；若地址不同，F1 位清 0。

因为一个完整的数据需分 2 帧传输，为了确保每个数据的第 1 帧数据与第 2 帧数据正确组合，在读到正确的第 1 帧数据之后紧接着又读到第 2 帧数据，才认为收到一个完整的数据。

读到第 2 帧数据后，对标志寄存器 FLAG 的 F0 位置 1。当 F1，F0 同时为 1，即 FLAG 等于 3 时，说明读到了 2 帧数据，先执行数据转换子程序，再执行显示子程序；FLAG 不等于 3 时，说明 2 帧数据没读完或根本没读到数据，跳过数据转换子程序，直接执行显示子程序，显示表头继续显示上一次数据转换子程序中转换好的数据。数据转换子程序中对标志寄存器 FLAG 清 0。

显示子程序中对 FLAG 的 F0 位清 0，既可以避免把先读到的第 2 帧数据和接着又读到的地址相同的第 1 帧数据进行组合转换而发生错误，又不会影响按正确顺序读到的数据的组合转换。

只有按照先读到第 1 帧数据、后读到第 2 帧数据的顺序读到的数据，才会在数据转换程序中转换为所要显示的数据，而被显示出来。

12.3　程序流程图

图 12.3 是程序流程图。

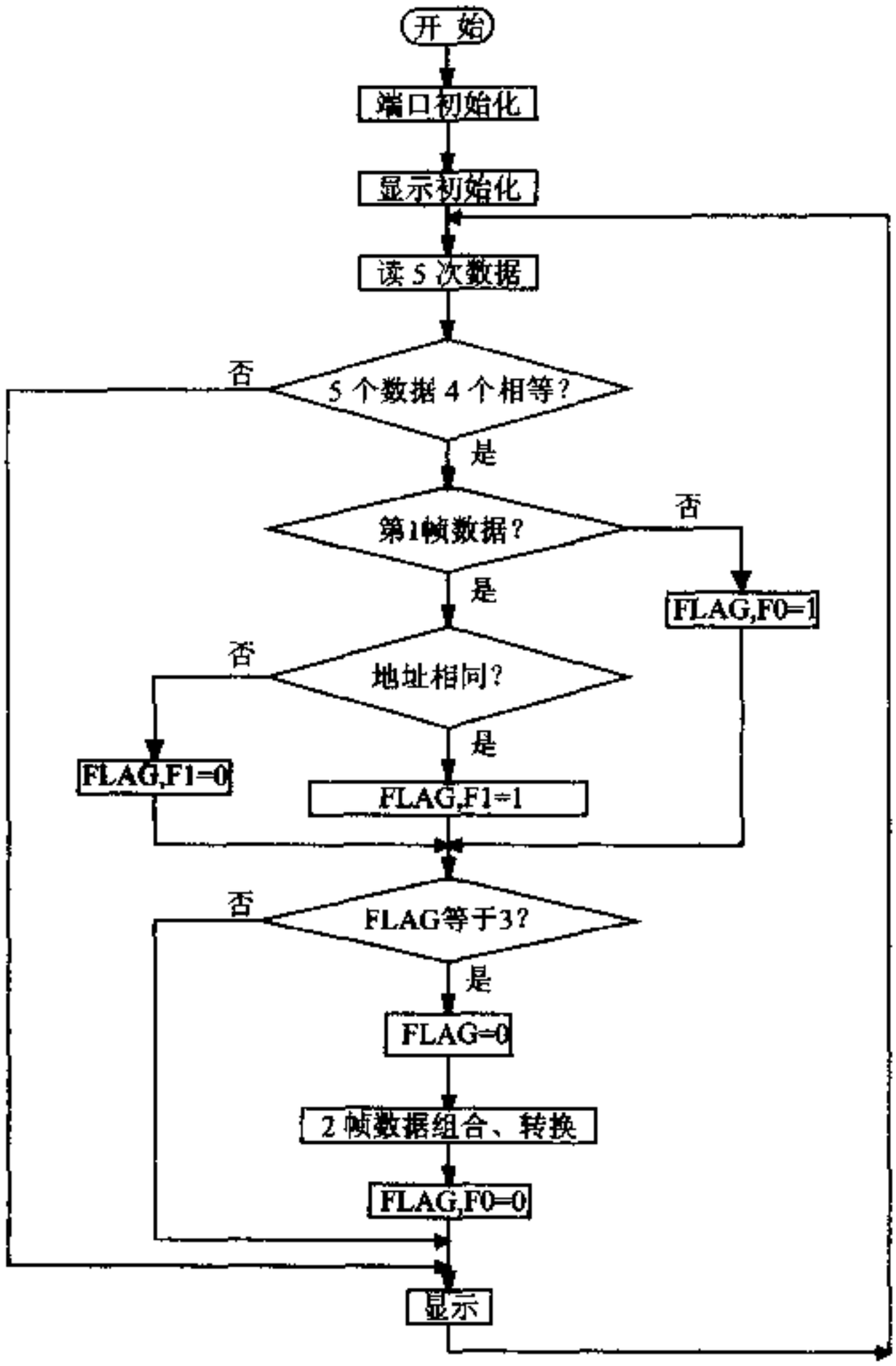


图 12.3 程序流程图

12.4 程序清单

```
#include <pic16F87x.h>
#include "mydefine.h"
#include <pic.h>
static int flag,flag0,flag1,flag3,led_d;
static int data1[5],data2[5];
static int data,data0,data_1,data_2,sdata;
```

```
//=========子程序==========
//端口初始化子程序
void initport( )
{
    PORTA=0;
    PORTB=0;
    PORTC=0;
    PORTD=0;
    ADCON1=0x07;
    TRISA=0x03;                    //设 RA0,RA1 为输入
    TRISB=0xE8;                    //设 RB0,RB1,RB2,RB4 为输出
    TRISC=0xFF;                    //设 C 口为输入
    TRISD=0;                       //设 D 口为输出
}
//判断地址是否相同子程序
int adr_jud(int x)
{
    int adress,y;
    adress=PORTA&0x03;
    x&=0x60;
    adress=adress<<5;
    if (adress==x) y=1;
    else     y=0;
    CLRWDT();
    return(y);
}
//显示初始化子程序
void initdis( )
{
    PORTB=0xFE;                    //选通数码管 1
    PORTD=0xC0;
    PORTB=0xFD;                    //选通数码管 2
    PORTD=0xC0;
    PORTB=0xFB;                    //选通数码管 3
    PORTD&=0x7F;                   //选通小数位
    PORTD=0xC0;
    PORTB=0xEF;                    //选通数码管 4
```

```
        PORTD=0xC0;
    }
    //读 5 次数据判是否有 4 次相等
    int judge(arry)
    int arry[5];
    {
        int i,j,k;
        for(i=0;i<=4;i++){
            k=0;
            for(j=0;j<=4;j++)
              { if(arry[i]==arry[j])  k++;
                if(k>=4) {
                    flag1=1;
                    data0=arry[i];
                    return(flag1);
                }
                else flag1=0;
             }
           }
        return(flag1);
    }
    //数据转换子程序
    int convert(int d1,int d2)
    {
        auto int dd1,dd2;
        int i1,j1,k1,i2,j2,m;
        dd1=d1;
        dd2=d2;
        j1=0x10;
        k1=2048;
        d1=0;
        for(i1=1;i1<=5;i1++) {
            if(j1==(dd1&j1)) m=1;
            else  m=0;
            d1=d1+m*k1;
            j1=j1/2;
            k1=k1/2;
```

```
    }
j2=0x40;
d2=0;
for(i2=1;i2<=7;i2++) {
        if(j2==(dd2&j2))    m=1;
        else    m=0;
        d2=d2+m*k1;
        j2=j2/2;
        k1=k1/2;
    }
    data=d1+d2;
    return(data);
}
//显示子程序
int display(int x)
{     int l1,l2,l3,l4;
    l1=x/1000;
    PORTB=0xFE;                          //选通数码管 1
    PORTD=led[l1];
    l2=(x-l1*1000)/100;
    PORTB=0xFD;                          //选通数码管 2
    PORTD=led[l2];
    l3=(x-l1*1000-l2*100)/10;
    PORTB=0xFB;                          //选通数码管 3
    PORTD=0x7F;
    PORTD=led[l3];
    l4=x-l1*1000-l2*100-l3*10;
    PORTB=0xEF;                          //选通数码管 4
    PORTD=led[l4];
}
//中断服务子程序
void interrupt int_serve( )
{
    PIR1=0;
    TMR1L=0xE5;
    TMR1H=0xBE;
    di( );
```

```
        sdata=PORTC&0x80;
        ei( );
    }
    //开中断子程序
    void int_open( )
    {
        inportc=PORTC&0x80;
        if(inportc==1) return;
        else  data1[0]=~PORTC;
        flag=adr_jud(data1[0]);
        if(flag==0)     return;                    //地址不同返回
        else  data1[1]=~PORTC;
        data1[2]=~PORTC;
        if(data1[0]==data1[1])
          if(data1[0]==data1[2])     {
            flag3=1;
            PIR1=0;                                //开通总中断前,清所有中断标志位
            TMR1IE=1;                              //TMR1 溢出中断使能
            PEIE=1;
           ei( );
            TMR1L=0xE5;
            TMR1H=0xBE;                            //20 ms 中断 1 次
            T1CON=0x01;                            //设 TMR1 为 1 分频,计数器方式工作
           }
        else  return;
    }
    //读第 1 帧子程序
    voidread_1( )
    {     int j0;
        for(j0=1;j0<=4;j0++)      data1[j0]=~PORTC;
        flag1=judge(data1);
        if (flag1==1) {
            data_1=data0;
            flag0=1;
            count1++;
        }
        flag=adr_jud(data1[0]);
```

```
        if(flag==1)   {
            for(j0=1;j0<=4;j0++) data1[j0]=~PORTC;
            flag1=judge(data1);
            if (flag1==1){
                    data_1=data0;
                    flag0=1;
                   count1++;
            }
        }
}
// 主程序
main( )
{     int i0,ii,i;
      flag0=0;                          //帧标志位
      flag1=0;                          //读 5 次数据判有 4 次相等标志位
      flag3=1;                          //开中断标志位
      count1=0;                         //读第 1 帧计数单元
      count2=0;                         //读第 2 帧计数单元
      data_1=0;
      data_2=0;
      led_d=0;
      led[0]=0xc0;                      //0
      led[1]=0xf9;
      led[2]=0xa4;
      led[3]=0xb0;
      led[4]=0x99;
      led[5]=0x92;
      led[6]=0x82;
      led[7]=0xf8;
      led[8]=0x80;
      led[9]=0x90;                      //9
initport( );
OPTION=0xFF;                            //开看门狗
initdis( );
while(1)      {
 if(flag3==0)   int_open();
 else{
```

```
        if(sdata==0x80){                    //第 2 帧数据到
            if(flag0==1){
                for(i0=0;i0<=4;i0++)   data2[i0]=~PORTC;
                flag1=judge(data2);
                if (flag1==1)      {
                    data_2=data0;
                    flag0=0;
                    count2++;
                }
            }
        }
        else if(sdata==0) {                 //第 1 帧数据到
            if(flag0==0)      {
                data1[0]=~PORTC;
                flag=adr_jud(data1[0]);
                if(flag==1) {
                    for(j0=1;j0<=4;j0++)          data1[j0]=~PORTC;
                    flag1=judge(data1);
                    if (flag1==1)      {
                        data_1=data0;
                        flag0=1;
                        count1++;
                    }
                }
            }
            CLRWDT();
                if(count1==count2)      led_d=convert(data_1,data_2 );
            }
            display(led_d);
        }
    }
```

第 13 章　数控步进直流稳压电源

D/A 转换是将数字量转换为模拟量的过程。在计算机控制系统中，这一技术应用十分广泛。灵活掌握这方面的技术，对于单片机开发应用者是十分重要的。本应用实例通过介绍“数控步进直流稳压电源”，阐明了如何灵活、有效地把 PIC16F877 单片机和一些常规外围电路组合，并充分利用软件编程的灵活性，实现所要求的功能。

13.1　电路原理图

本实例介绍的“数控步进直流稳压电源”实际上是由 PIC16F877 单片机控制的直流输出电源。该电源的输出电压能在 0～12 V 的范围内、按照 0.1 V 的步进量连续可调(可通过编程设定为自动或手动步进)。电路原理图如图 13.1 所示(为表示方便，该图部分线路采用网络标

图 13.1　步进电源电路图

号的形式，只需把具有相同网络标号的线路连接在一起，如凡是标有 GND 的地方都表示接地。该图的单片机及 MAX518 部分可在实验板上直接得到）。

图中变压器从电网中取出一电压信号，经过桥式整流器后得到一直流电压；该电压接到 3 端可调稳压器 LM317 的输入端，作为供电电压；MAX518 的 D/A 输出端 D/A1 经过运算放大器组的运算后，接到 LM317 的电压调整端。图中所示的电阻值为用仿真软件得到的精确值，实际制作电路时，可用可调电阻得到某些特殊的阻值。

13.2 系统工作原理

本应用实例的原理为：PIC16F877 单片机送出一 8 bit 数据 D_n 给数/模转换器 MAX518，由后者输出一对应模拟量 $D/A1=5*D_n/255$ V（MAX518 的参考电压为 5 V）；该模拟量经过 LM324 组电路以及 LM317 稳压电路变换后，得到对应的输出量 V_{OUT}；当 PIC16F877 送出的 8 bit数据 D_n 按照预定的规律变化时，输出量 V_{OUT} 也按照预定规律变化；同时为了人机交付方便，把 V_{OUT} 的值显示在 LED 上，并通过键盘选择步进加或步进减。

13.2.1 数/模输出部分

该部分直接由 MAX518 输出一模拟量 $D/A1=5*D_n/255$ V。

13.2.2 LM324 电路

为减小后级电路对前级的影响，D/A1 先经过一电压跟随器 U1：1 从引脚 1 输出，大小不变，仍为 $D/A1=5*D_n/255$ V；然后 D/A1 输入一个加法器 U1：2 和一个增益为 1 的反相比例放大器，得到输出：

$$V_{OUT2}=0.05D_n-1.25$$

单位为 V，此即为 LM324 组电路的最后输出。

13.2.3 LM317 稳压电路

为使该数控电源具有承受一定负载的能力，采用 LM317 稳压电路输出。V_{OUT2} 接在 LM317 的 1 引脚上，由于 LM317 的特性决定了其引脚 2 电压比引脚 1 电压恒定高出 1.25 V，故引脚 2 的输出即所要的电压：

$$V_{OUT}=V_{OUT2}+1.25=0.05\ D_n$$

单位为 V。到此为止，已推出 PIC16F877 的输出 8 bit 数据 D_n 与需要的输出 V_{OUT} 之间的关系式，即：$V_{OUT}=0.05\ D_n$，单位 V。不难看出，当单片机输出的数字量 D_n 为 0～240（*十六进制时*为 00H～F0H）以步进为 2 变化时，V_{OUT} 为 0～12 V 以步进为 0.1 V（也可以轻易实现步进量为 0.05 V）变化。

13.3　程序设计

13.3.1　编程思想

本程序设定 S9 键为增加键，当按住 S9 键不松开时，输出 V_{OUT} 以 0.1 V 连续步进，直至 S9 键松开；当以一定的时间间隔点动 S9 键时，输出 V_{OUT} 也为点动步进。递减键 S11 的功能与 S9 基本相同；同时，输出电压的值显示在 3 个 LED 上。通过这种人机交互设置，可以方便地对电压源输出进行控制。

源程序的工作过程为：系统上电复位以后，默认输出 0 V 电压，此时 3 个 LED 显示 00.0 V；然后扫描 S9，S11 键，当 S9 或 S11 有键按下时，程序跳转至相应的按键处理子程序，经过按键处理子程序处理后，置相应的标志位，并处理相应的寄存器的值；再回到主程序中，依据不同的标志送出相应的数字量 D_n 给 MAX518，并把相应的数据送入显示缓冲区，最后显示电源输出的电压值；程序继续扫描 S9 和 S11，再循环执行前面的步骤。在程序编制过程中，R3，R2，R1 寄存器分别存放 3 个 LED 上显示的数字；TXDATA 寄存器内存储待转换的数字量 D_n；BJF 为增减标志(为 1 时步进增，为 0 时步进减)。

13.3.2　程序流程框图

图 13.2 和图 13.3 为部分程序框图。

13.3.3　关于程序的说明

在初始化模块中，应设置相应口的输入、输出方式，并令 R3＝R2＝R1＝0，TXDATA＝0，LEDF＝1，BJF＝1(默认为步进加方式)。

此程序当 V_{OUT} 增加到 12.0 V 时，返回 0.00 V；当 V_{OUT} 减少到 00.0 V 时，返回 12.0 V。有关 MAX518 的工作原理，请参考本书相关章节。

13.3.4　源程序代码

```
#include <pic.h>
//此程序实现"数控步进直流稳压电源"的功能，调试时为了避免资源冲突，应使实验板上的拨码开关 S8 拨向高电平
//本程序设定 S9 键为增加键，当按住 S9 键不松开时，输出 VOUT 以 0.1 V 连续步进，
//直至 S9 键松开，当以一定的时间间隔点动 S9 键时，输出 VOUT 也为点动步进
//递减键 S11 的功能与 S9 基本相同时，输出电压的值显示在 3 个 LED 上
unsigned  char  R1,R2,R3,TXDATA,LEDF,BJF,COUNTW,data;
```

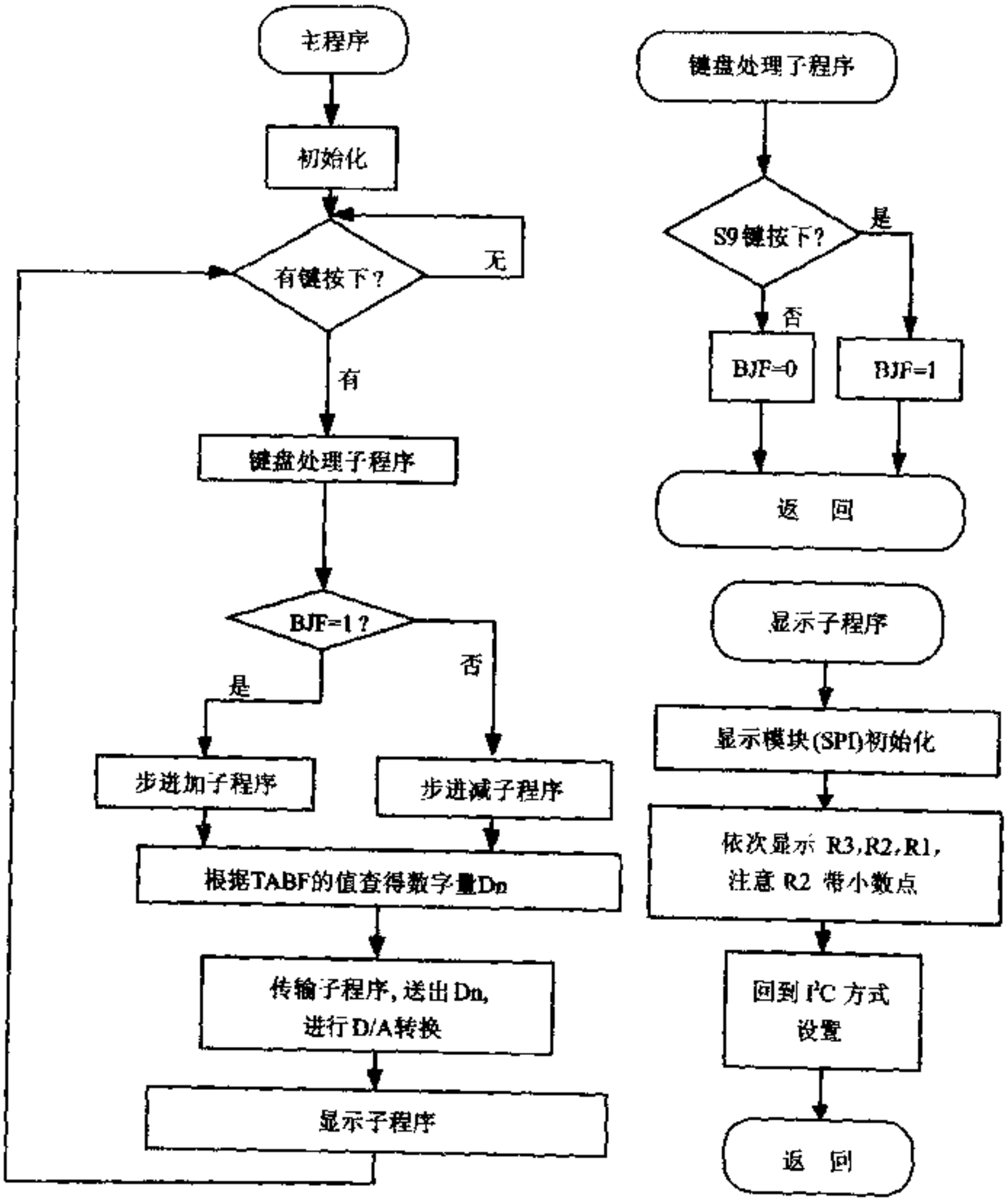


图 13.2 部分程序框图

```
unsigned    int     i;
const   char   table[11]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0XD8,0x80,0x90,0xFF};
//不带小数点的显示段码表
const     char          table0[11]={0X40,0X79,0X24,0X30,0X19,0X12,0X02,0X78,0X00,0X10,
0xFF};
//带小数点的显示段码表
unsigned     char      s[4];                    //定义一个显示缓冲数组
```

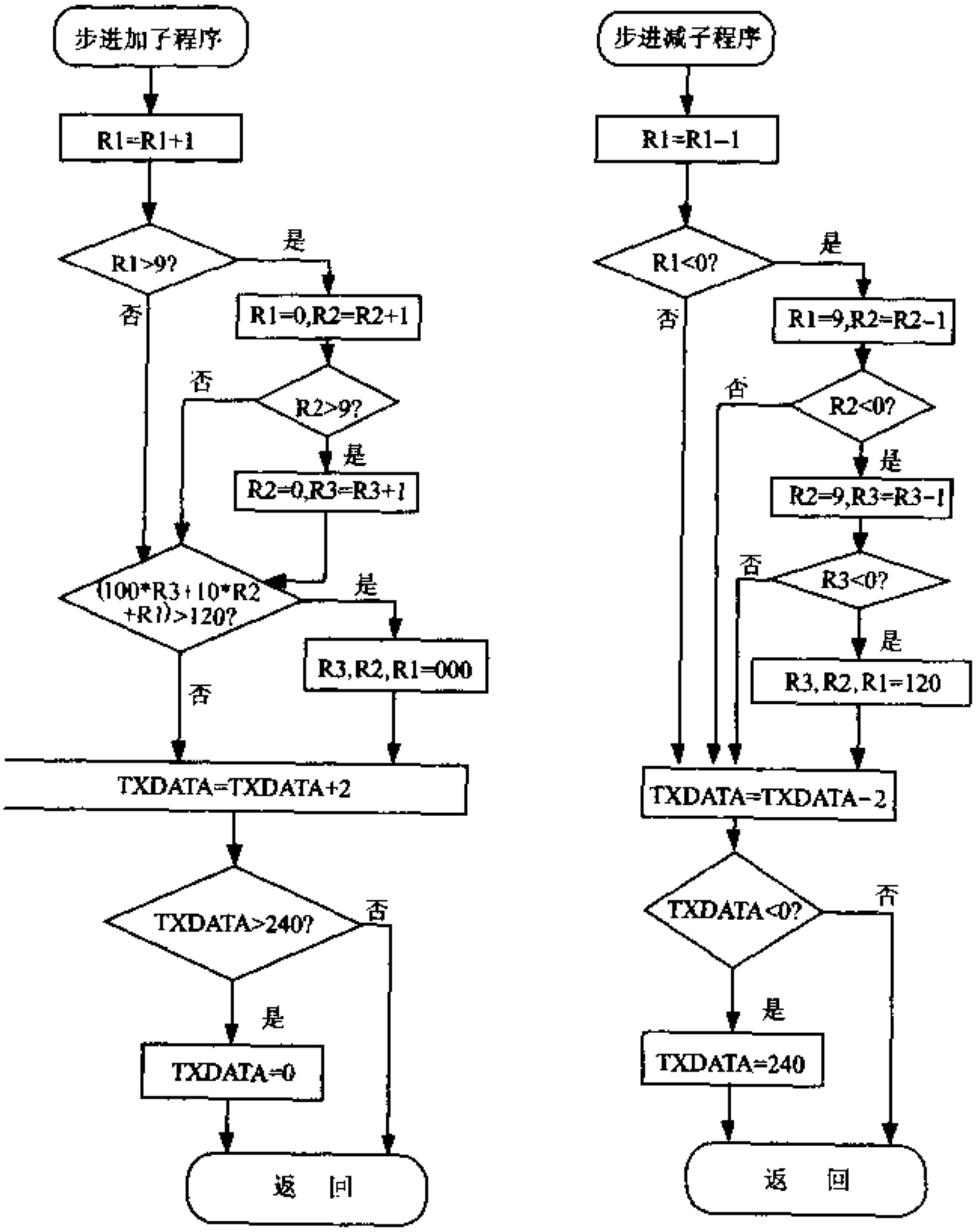


图 13.3　部分程序框图

```
//把需要显示的数字装入显示缓冲数组
void        sfz()
{
    s[0]=R3;
    s[1]=R2;
    s[2]=R1;
    s[3]=0x0A;                          //最后 1 个 LED 显示"DARK"
```

```
}
//系统各寄存器初始化子程序
void        initial()
{
    R1=0X00;
    R2=0X00;
    R3=0X00;
    sfz();                          //把需要显示的数字装入显示缓冲数组
    TXDATA=0X00;
    LEDF=0X01;
    BJF=0X01;
    TRISB1=0;
    TRISB2=0;
    TRISB4=1;
    TRISB5=1;                       //设置与键盘相关的各口的输入、输出方式
    RB1=0;
    RB2=0;                          //设置扫描初始条件
}
//spi 方式显示初始化子程序
void        SPIINIT()
{
    PIR1=0;
    SSPCON=0x30;
    SSPSTAT=0xC0;
//设置 SPI 的控制方式,允许 SSP 方式,并且时钟下降沿发送,与"74HC595,当其
//SCLK 从低到高跳变时,串行输入寄存器"的特点相对应
    TRISC=0xD7;                     //SDO 引脚为输出,SCK 引脚为输出
    TRISA5=0;                       //RA5 引脚设置为输出,以输出显示锁存信号
}
//I²C 初始化子程序
void        i2cint()
{
    SSPCON = 0X08;                  //初始化 SSPCON 寄存器
    TRISC3 =1;                      //设置 SCL 为输入口
    TRISC4 =1;                      //设置 SDA 为输入口
    TRISA4 = 0;
    SSPSTAT=0X80;                   //初始化 SSPSTAT 寄存器
```

```
    SSPADD=0X02;                    //设定 I²C 时钟频率
    SSPCON2=0X00;                   //初始化 SSPCON2 寄存器
    di();                           //关闭总中断
    SSPIF=0;                        //清 SSP 中断标志
    RA4=0;                          //关掉 74HC165 的移位时钟使能,以免 74HC165
                                    //移位数据输出与 I²C 总线的数据线发生冲突
    SSPEN=1;                        //SSP 模块使能
}
//软件延时子程序
void        DELAY()
{
    for(i = 3553; --i ;) continue;
}
//键服务子程序
void        keyserve()
{
    PORTB=0XFD    ;
    if(RB5==0)    BJF=0X01;         //S9 键按下,步进加标志置 1
    PORTB=0XFB    ;
    if(RB5==0)    BJF=0X00;         //S11 键按下,步进加标志清 0
    RB1=0;                          //恢复 PORTB 的值
    RB2=0;
}
//键扫描子程序
void        KEYSCAN()
{
    while(1){
    while(1)    {
            if (RB5==0)    break;
        }
        DELAY();                    //若有键按下,则软件延时
        if (RB5==0)    break;       //若还有键按下,则终止循环扫描,返回
      }
}
//SPI 传输数据子程序
void  SPILED(data)
{
```

```
    SSPBUF=data;                                    // 启动发送
    do      {
        ;
    }while(SSPIF==0);
    SSPIF=0;
}
//显示子程序
void        display()
{
    SPIINIT();                                      //spi 方式显示初始化
    RA5=0;                                          //准备锁存
    for(COUNTW=0;COUNTW<4;COUNTW++)      {
        data=s[COUNTW];
        if(COUNTW==1)      data=table0[data];//第 2 位需要显示小数点
        else      data=table[data];
        SPILED(data);                               //发送显示段码
    }
    for(COUNTW=0;COUNTW<4;COUNTW++){
        data=0xFF;
        SPILED(data);                               //连续发送 4 个"DARK",使显示看起来好看 些
    }
    RA5=1;                                          //最后给 个锁存信号,代表显示任务完成
}
//I²C 总线输出数据子程序
void        i2cout()
{
    i2cint();                                       //因为 SPI 输出和 I²C 输出不能同时工作,则需要
                                                    //不断在 2 种方式间切换
    SEN=1;                                          //产生 I²C 启动信号
    for(i=0x02;--i;)      continue;                 //给予一定的延时,保证启动
    do      {
        RSEN=1;                                     //产生 I²C 启动信号
    }while(SSPIF==0);                               //如果没能启动,则反复启动,直到启动为止
    SSPIF=0;                                        //SSPIF 标志清 0
    SSPBUF=0X58;                                    //I²C 总线发送地址字节
    do      {
        ;
```

```
    }while(SSPIF==0);                    //等待地址发送完毕
    SSPIF=0;                             //SSPIF 标志清 0
    SSPBUF=0X01;                         //I²C 总线发送命令字节
    do      {
        ;
    }while(SSPIF==0);                    //等待命令发送完毕
    SSPIF=0;                             //SSPIF 标志清 0
    SSPBUF=TXDATA;                       //I²C 总线发送数据字节
do      {
        ;
    }while(SSPIF==0);                    //等待数据发送完毕
    SSPIF=0;                             //SSPIF 标志清 0
    PEN=1;                               //产生停止条件
    do      {
        ;
    }while(SSPIF==0);                    //等待停止条件产生
    SSPIF=0;                             //SSPIF 标志清 0
}
//步进加子程序
void        BJADD()
{
    R1++;
    TXDATA=TXDATA+2;
    if(R1>9) {
        R1=0;
        R2++;
        if(R2>9)    {
            R2=0;
            R3++;
        }
}
if((R3==1)&&(R2==2)&&(R1==1)){
    R3=0;
    R2=0;
    R1=0;                                //若(100*R3+10*R2+R1)的值超过 120,则又从 0 计起
    TXDATA=0;
}
```

```
    sfz();                                          //把需要显示的数字装入显示缓冲数组
}
//步进减子程序
void        BJSUB()
{
    R1--;
    TXDATA=TXDATA-2;
    if(R1==0XFF){
        R1=9;
        R2--;
        if(R2==0XFF)    {
            R2=9;
            R3--;
            if(R3==0XFF){
                R3=1;
                R2=2;
                R1=0;                   //若(100*R3+10*R2+R1)的值小于0,则又从120计起
                TXDATA=0XF0;
            }
        }
    }
    sfz();                                          //把需要显示的数字装入显示缓冲数组
}
//主程序
main()
{
    initial();                                      //系统各寄存器初始化
    display();                                      //调用一次显示子程序
    while(1)    {
        i2cout();                                   //调用I²C子程序,启动D/A转换
        KEYSCAN();                                  //键盘扫描
        keyserve()    ;                             //若确实有键按下,则调用键服务程序
        if(BJF==0X01)       BJADD();                //若步进加标志为1,则调用步进加子程序
        else    BJSUB();                            //若步进加标志为0,则调用步进减子程序
        display();                                  //调用一次显示子程序
    }
}
```

第 14 章　单片机控制的电动自行车驱动系统

14.1　单片机控制的电动自行车驱动系统简介

现有的电动自行车一般采用电池(通常为 36 V)供电,其控制对象是无刷直流电机。本章介绍采用 PIC16F877 实现的电动自行车驱动控制系统。

14.2　无刷直流电动机的工作原理

无刷直流电动机既具有交流电机的简单、运行可靠、维护方便等优点,又具有直流电机运行效率高、调速性能好的优点;而且由于不受机械换向限制,易于做到大容量、高转速,在航天、军工、数控、冶金、医疗器械等领域有着广阔的应用前景,正成为研究的热点。本章的电动自行车使用无刷直流电动机。

无刷直流电机的运行原理与步进电动机相似,都是通过给电机的 2 相绕组通电,在电机内部产生一定的磁场。由于磁通具有力图走最小路径的特点,从而使转子和定子的相对位置发生变化。当按照一定的顺序轮流使不同的 2 相绕组导电时,则可使电机内部的磁场旋转起来,从而使电动机转动。电机的通电顺序不同,电机的转动方向也不相同。

如图 14.1 所示,当通电顺序为 BC,AC,AB,CB,CA,BA 时(其中 A,B,C 表示电机的 3 相),电动机正转。此时应使相应的 MOSFET 导通,其顺序为 M3M6,M1M6,M1M4,M5M4,M5M2,M3M2。如果反转,则只需将 MOSFET 的导通顺序倒过来即可。

由于转子从一个位置转动到下一位置受磁场大小的影响,磁场越大,转子转动的转矩也越大,而磁场大小由电机 2 端的电压大小决定;因而改变电机 2 端的直流电压的大小,就可以改变电机转矩的大小,从而达到调节转速的目的。

如图 14.1,将单片机输出的各相上桥臂的触发信号和斩波信号 PWM 相与,即可对转速进行调节。

由于触发模块 IR2102 的输出信号与输入信号极性相反,实际上是先对单片机输出的换向信号和 PWM 信号相或之后,再输入 IR2102 的。IR2102 的功能参考相关资料。

换向信号由电动机内嵌的 3 个数字霍尔元件给出。当电动机转动到相应的位置时,霍尔信号则发生相应的变化,且霍尔信号和转子的位置是一一对应的;因此检测到霍尔信号发生变化时,再触发相应的 MOSFET,即可实现电动机换向控制。

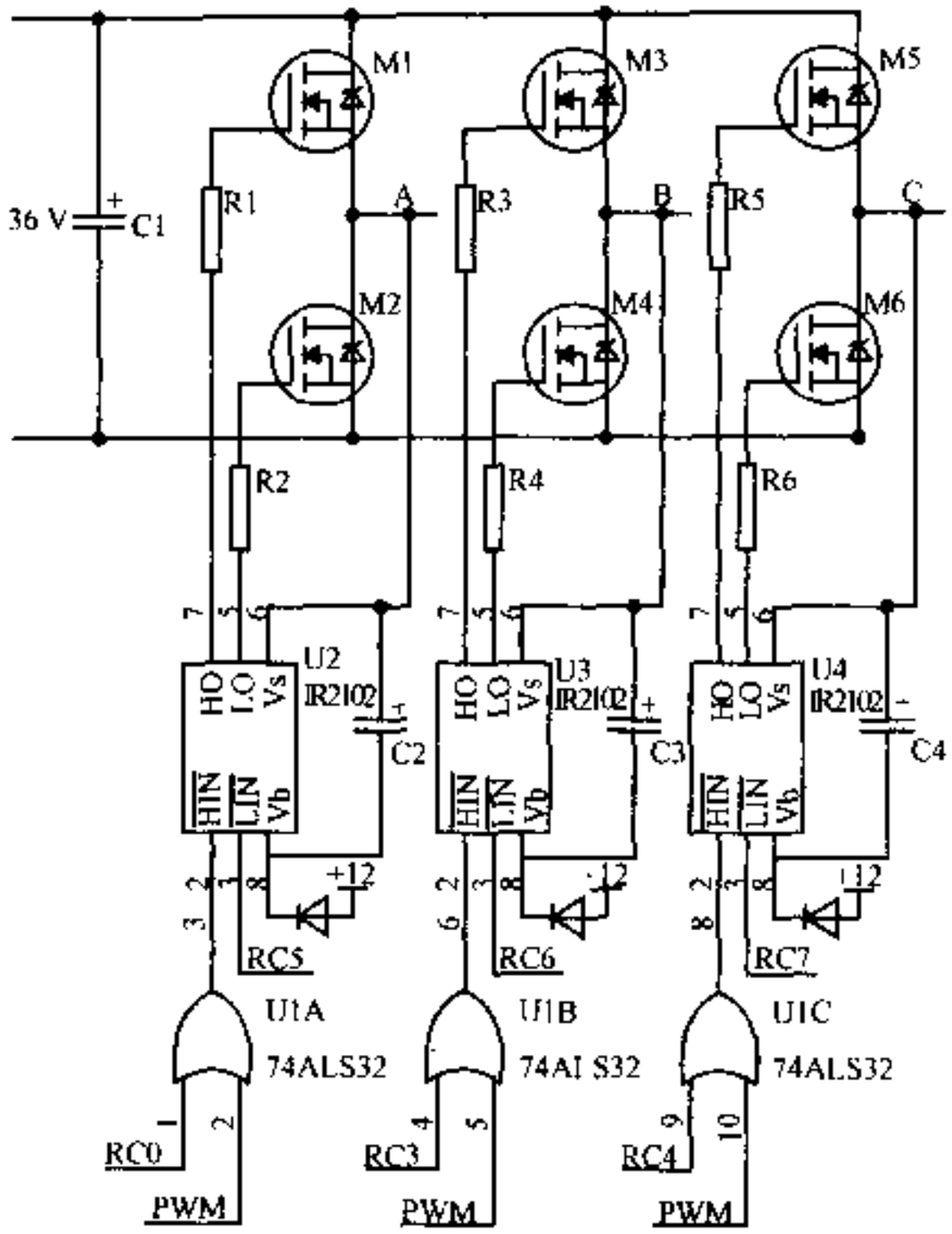


图 14.1 触发及换向驱动电路

14.3 控制系统结构设计

电动车控制系统实现的主要功能是对电动机进行调速、刹车及欠压保护。实现其功能对应的电动车系统结构框图如图 14.2 所示。手柄给定 TS 为一电位器，其范围为 0～5 V，由 PIC 单片机 AN0 通道采样。电池电压经过放大电路后，由 AN2 通道采样。采样值在单片机内部进行比较，当电压较低时进行欠压保护。从电机输出的霍尔信号被分别输入到 B 口的高 3 位 RB5，RB6，RB7。由于 PIC 单片机 B 口高位具有电平变化中断的功能，当霍尔信号发生变化时，单片机采样霍尔信号，再输出相应的触发信号。触发信号由单片机 C 口的 RC0，RC3，RC4，RC5，RC6，RC7 输出，如图 14.1 所示，共 6 路信号由单片机分别控制，互不影响。PWM 信号由 PIC 单片机输出，其作用是对图 14.1 中上桥臂 MOSFET 进行调制，通过 PWM 方式改变输出电压的大小。刹车信号与单片机 RB0/INT 相连。当刹车时，由刹车手柄上的开关给出开关量信号，单片机接受到低电平信号，在下降沿产生中断，并进行相应的处理。

另外，为了改善电机的调速性能，单片机还需对电机电流进行采样(图中没有标出)，并根

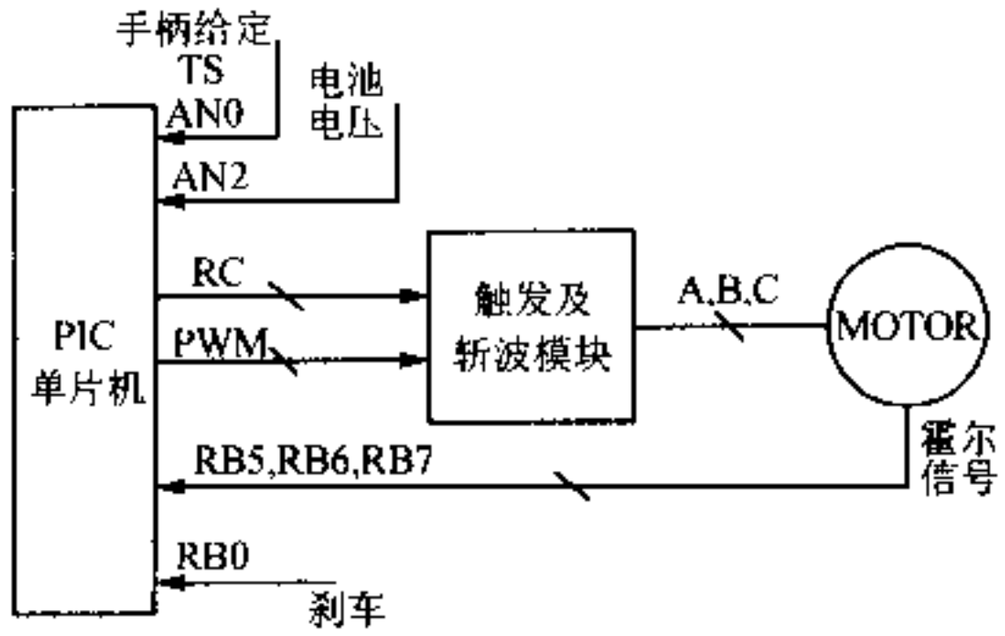


图 14.2　硬件结构示意图

据霍尔信号计算电机转速。

14.4　控制系统软件设计

14.4.1　控制系统方案设计

本系统采用转速双闭环调速方案。其中，内环为电流环，外环为转速环，转速给定由手柄提供。转速环输出 PWM 高电平时间，通过调节 PWM 占空比，调节电机转速。

注意，这里指的转速环输出仅仅为单片机的输出。

14.4.2　单片机内部资源分配

本应用程序共用到 13 个 I/O 接口，其中 RC0，RC3，RC4，RC5，RC6，RC7 输出触发信号；RC2/CCP1 输出 PWM 调压信号；RB7，RB6，RB5 采样霍尔信号；RB0/INT 用于刹车中断；RA0/AN0 采样手柄；RA2/AN2 采样电池电压。

本系统使用了 PIC 单片机的以下特殊功能模块及功能：AD 模块，CCP1 模块的 PWM 功能，CCP2 模块的特殊事件触发功能触发 AD、RB0 中断、RB 口电平变化中断及看门狗定时器。

14.4.3　控制程序设计

各主要部分的流程图如图 14.3～14.7 所示。

其中主程序(图 14.3)模块完成 PWM 输出、欠压处理及刹车处理 3 大功能；PWM 输出直接调节输出电压的大小，进而控制电机的转速；欠压处理(图 14.6)的功能是当电池电压低于保护电压(32 V)时，停止对电机输出电压，但保持正常的换向顺序，如果电池开路电压大于重

开电压(35 V),则跳出保护程序,否则一直处于保护状态;刹车处理(图 14.5)和欠压处理相似,不同的是跳出处理的条件是刹车信号已去掉,且手柄已复位。

为了防止程序跑飞,本系统使用了看门狗定时器 WDT。由于主程序是一个大的循环,因而每个循环要清看门狗 WDT。

中断服务程序(图 14.4)主要包括:B 口电平变化中断、AD 采样中断、刹车中断 3 个中断服务子程序。

其中 B 口电平变化中断服务子程序(图 14.7)根据电机的转子状态输出相应的 MOSFET 换向信号,从而保证电机能够平滑地转动;AD 中断服务子程序完成对电流、手柄电压及电池电压的采样,电流采样由 CCP2 模块触发,手柄电压采样周期为电流采样周期的 256 倍,电池电压采样周期为手柄采样周期的 256 倍,采样通道的转换及触发均在中断服务子程序中完成;刹车中断只是通过建立标志位,通知单片机已有刹车信号输入,以便在主程序中进行刹车处理。

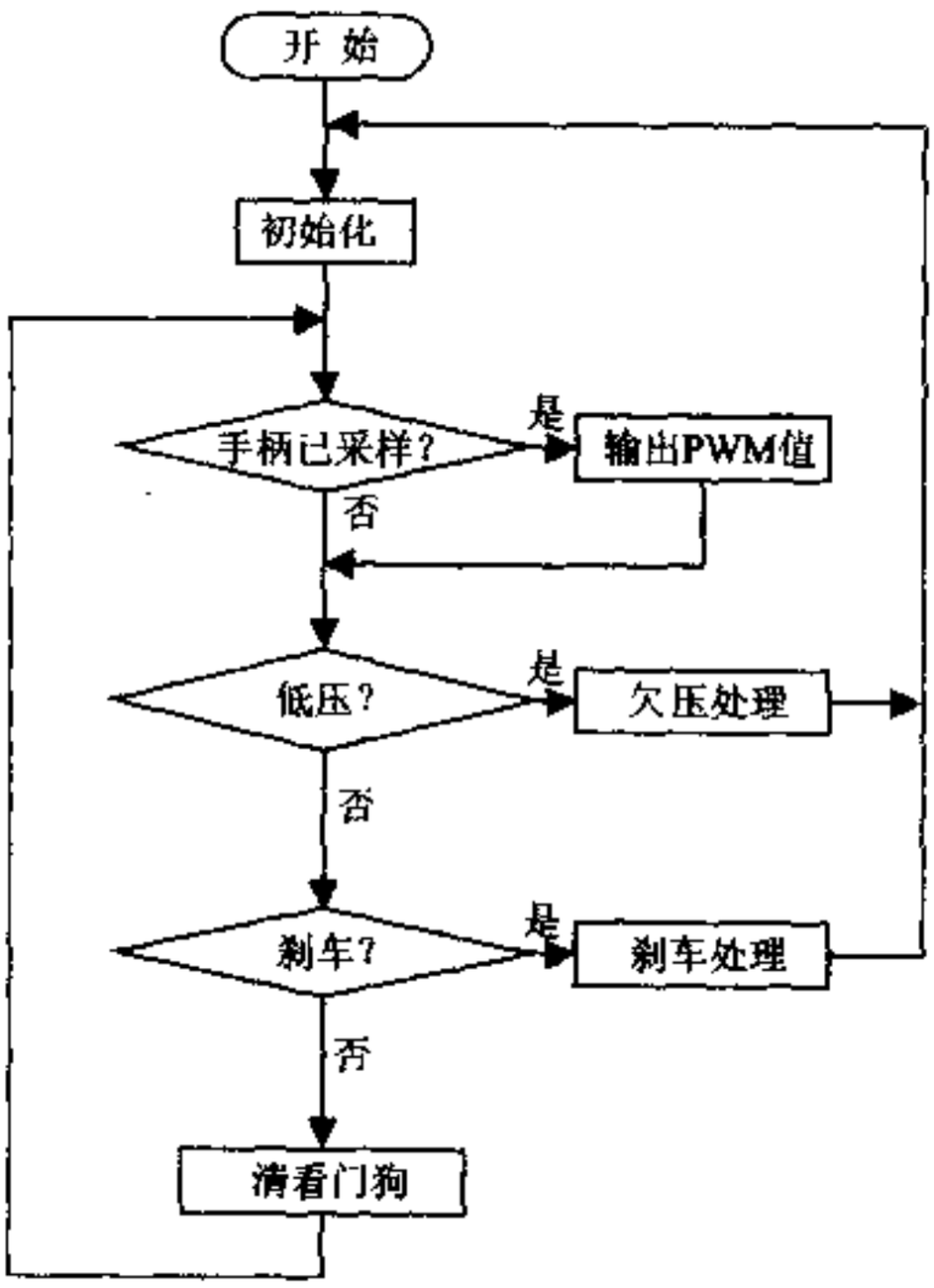


图 14.3 主程序

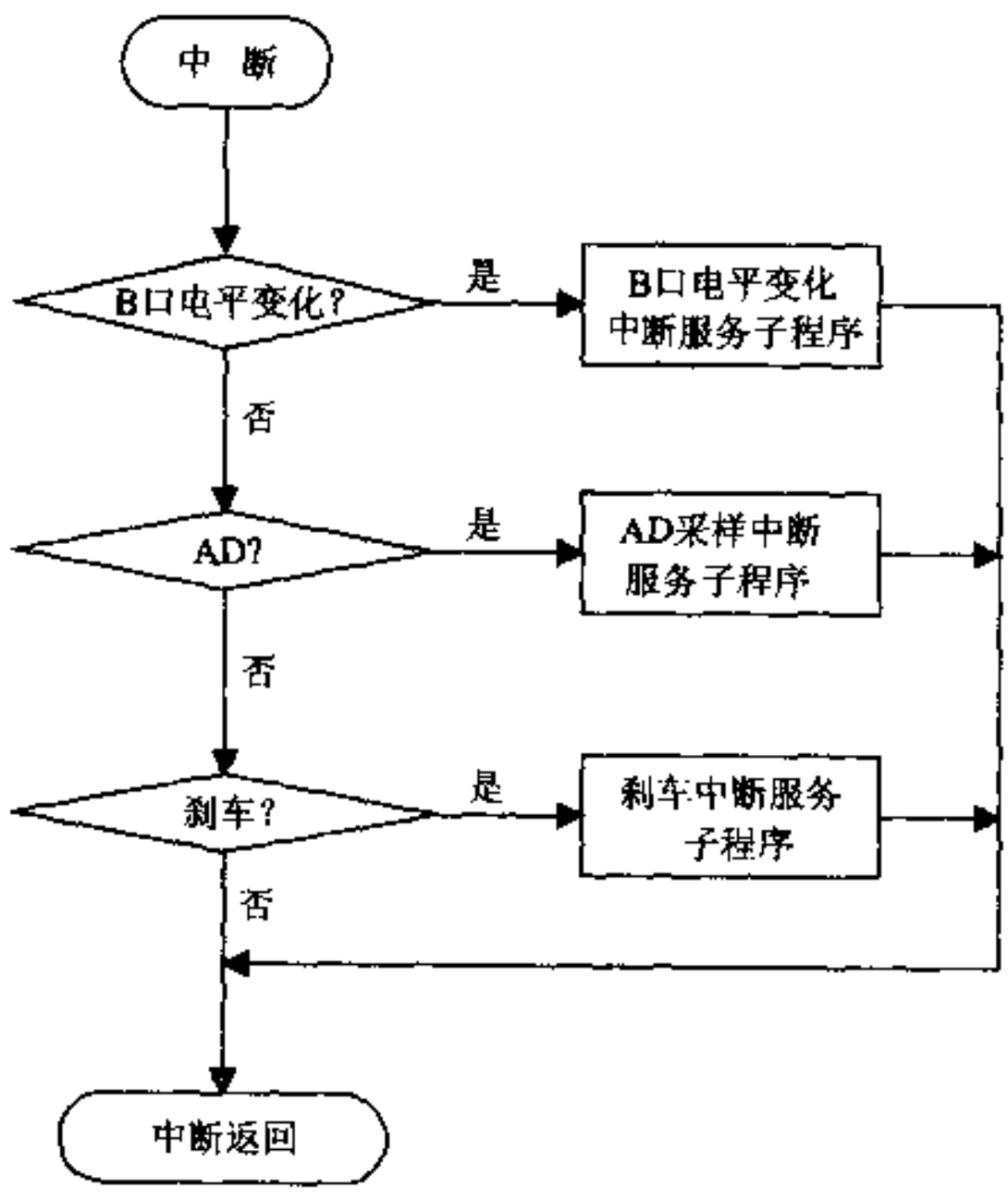


图 14.4 中断服务程序

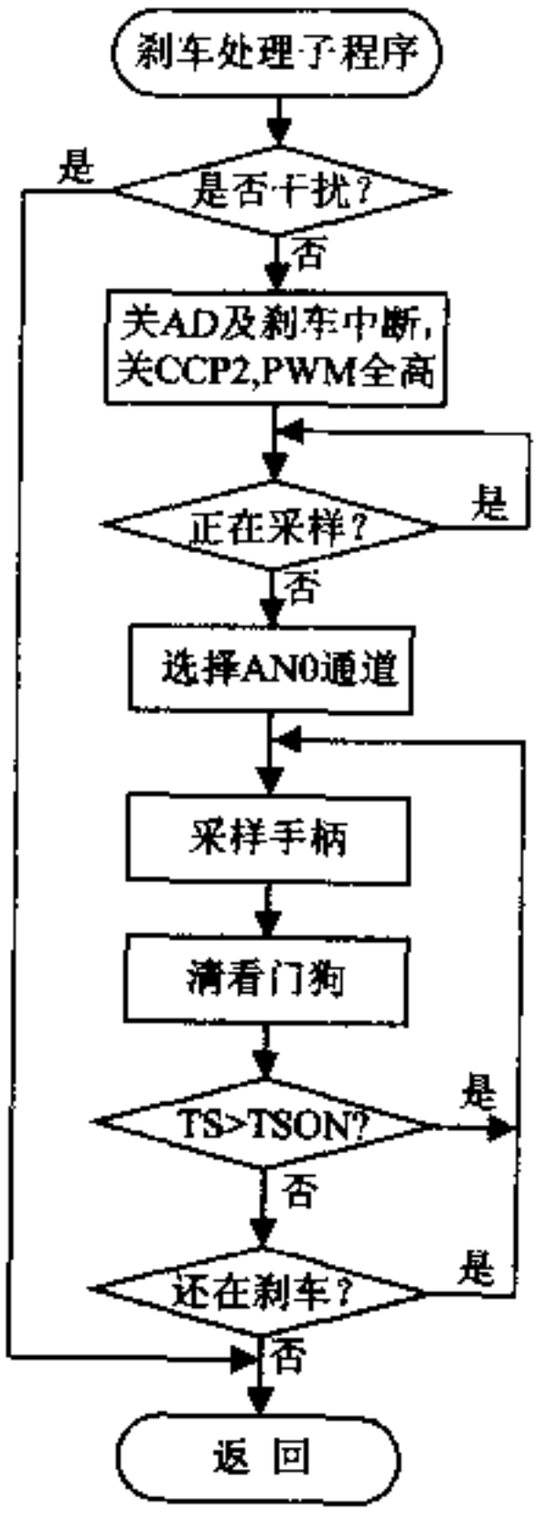


图 14.5　刹车处理子程序

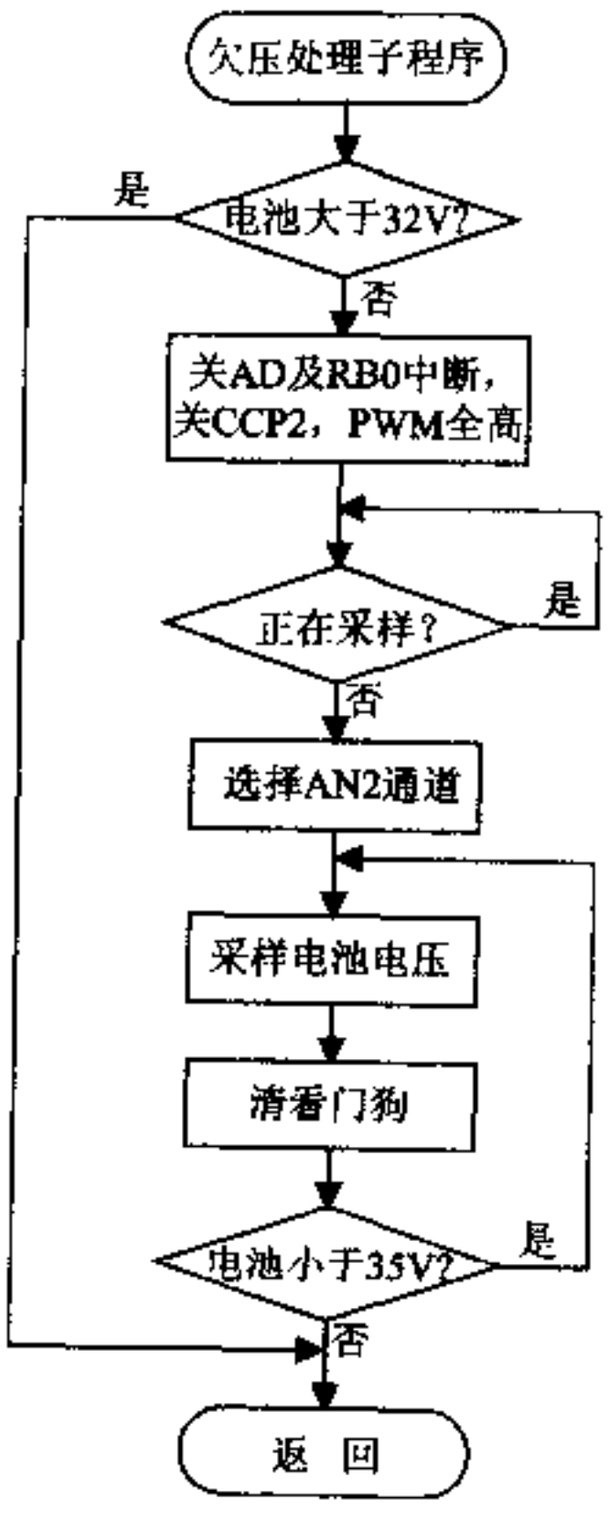


图 14.6　欠压处理子程序

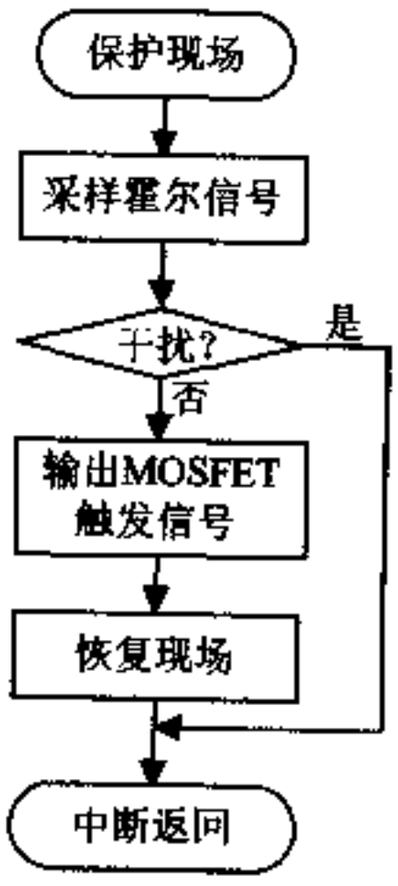


图 14.7　B 口电平变化中断服务子程序

14.4.4 C 语言程序

```
# include <pic. h>
//电动车双闭环程序,采用双闭环方式控制电机,以得到最好的转速性能,并且可以
//限制电机的最大电流。本应用程序用到 2 个 CCP 部件,其中 CCP1 用于 PWM 输出,以控制
//电机电压;CCP2 用于触发 AD,定时器 TMR2、TMR1、INT 中断,RB 口电平变化中断,看
//门狗以及 6 个通用 I/O 口
# define AND 0xe0                    //状态采集 5,6,7 位
# define CURA 0X0a                   //电流环比例和积分系数之和
# define CURB 0X09                   //电流环比例系数
# define THL   0X6400                //电流环最大输出
# define FULLDUTY 0X0FF              //占空比为 1 时的高电平时间
# define SPEA 0X1d                   //转速环比例和积分系数之和
# define SPEB 0X1c                   //转速环比例系数
# define    GCURHILO 0X0330          //转速环最大输出
# define GCURH 0X33                  //最大给定电流
# define GSPEH 0X67                  //最大转速给定
# define TSON 0X38                   //手柄开启电压 1.1 V,TSON × 2 为刹车后手柄开启电压,即 2.2 V
# define VOLON 0X4c                  //低电压保护重开电压 3.0 V 即 33 V
# define VOLOFF 0X49                 //低电压保护关断电压 2.86 V 即 31.5 V
volatile unsigned char DELAYH,DELAYL,oldstate,speed,
    speedcount,tsh,count_ts,count_vol,gcur,currenth,
    voltage;                         //寄存器定义
static bit sp1,spe,ts,volflag,spepid,lowpower,
    off,shutdown,curpid;             //标志位定义
static volatile unsigned char new[10]={0xaf,0xbe,0xff,0x7e,0xcf,
    0xff,0xd7,0x77,0xff,0xff};    //状态寄存器表
//————————PIC16F877 初始化子程序————————
void INIT877()
{
    PORTC=0X0FF;                     //关断所有 MOSFET
    TRISC=0X02;                      //设置 C 口输出
    PIE1=0X00;                       //中断寄存器初始化,关断所有中断
    TRISA=0XCF;                      //设置 RA4,RA5 输出
    TRISB=0XEF;                      //RB 口高 3 bit 输入,采集电机 3 相的霍尔信号
    PORTC=new[(PORTB&AND)>>5];       //采集第 1 次霍尔信号,并输出相应的信号,导通 2
                                     //个 MOS 管
```

```
T2CON=0X01;                    //TMR2 4 分频
CCPR1L=0X0FF;                  //初始时 PWM 输出全高
CCP1CON=0X0FF;                 //CCP1 设置为 PWM 方式
CCP2CON=0X0B;                  //CCP2 设置为特殊方式,以触发 AD
ADCON0= 0X81;                  //AD 时钟为 32 分频,且 AD 使能,选择 AN0 通道采集手
                               //柄电压
TMR2=0X00;                     //TMR2 寄存器初始化
TMR1H=0X00;                    //TMR1 寄存器初始化
TMR1L=0X00;
T1CON=0X00;                    //TMR1 为 1 分频
CCPR2H=0X08;
CCPR2L=0X00;                   //电流采样周期设置为 T_AD=512 μs
PR2=0XC7;                      //PWM 频率设置为 5 kHz
ADCON1=0X02;                   //AD 结果左移
OPTION=0XFB;                   //INT 上升沿触发
TMR2ON=1;                      //PWM 开始工作
INTCON=0XD8;                   //中断设置 GIE=1,PEIE=1,RBIE=1
ADIE=1;                        //AD 中断使能
speedcount=0x00;               //转速计数寄存器
speed=0x7f;                    //转速保持寄存器
spe  1;                        //低速标志位
spl=1;                         //低速标志位
oldstate=0x0ff;                //初始状态设置,区别于其他状态
count_ts=0x08;                 //电流采样 8 次,采集 1 次手柄
count_vol=0x00;                //采样 256 次手柄,采集 1 次电池电压
    ts=1;                      //可以采集手柄值的标志位
    ADGO=1;                    //AD 采样使能
    TMR1ON=1;                  //CCP2 部件开始工作
}
//~~~~~~~~~~延时子程序~~~~~~~~~~
#pragma interrupt_level 1
void DELAY1(x)
char x;
{
    DELAYH  x;                 //延时参数设置
#asm
DELAY2      MOVLW      0X06
```

```
        MOVWF     _DELAYL
DELAY1  DECFSZ    _DELAYL
        GOTO      DELAY1
        DECFSZ    _DELAYH
        GOTO      DELAY2
#endasm
}
//-  ----------状态采集子程序----------
void sample()
{
char state1,state2,state3,x;
do     {
       x=1;
       state1=(PORTB&AND);              //霍尔信号采集
       DELAY1(x);
       state2=(PORTB&AND);
}while(state1-state2);                  //当3次采样结果不相同时,继续采集状态
if(state1-oldstate!  =0)                //看本次采样结果是否与上次相同,不同
                                        //则执行
 {oldstate=state1;                      //将本次状态设置为旧状态
  state1=(oldstate>>5);
  PORTC=new[state1];                    //C口输出相应的信号触发2个MOS管
  if(sp1==1){spe=1;sp1=0;}
  else     {                            //如果转速很低,则spe置1
spe=0;sp1=0;
     speedcount<<=1;
       state3=(TMR1H>>2);               //否则,spe=0,计转速
       speed=speedcount+state3;         //speed寄存器为每256 μs加1
      }
      speedcount=0;
     }
}
//-  ----------AD采样子程序----------
void AD()
{
    char x;
    ADIF=0;                             //清AD中断标志位
```

```
if(ts==1){                                    //如果为手柄采样,则采样手柄值
 CHS0=1;                                      //选择电流采样通道
 count_vol=count_vol+1;                       //电池采样计数寄存器
 spepid=1;                                    //置转速闭环运算标志
 ts=0;tsh=ADRESH;                             //存手柄值
 if(count_vol==0) {                           //如果电池采样时间到,则选择 AN2 通道,采集电池电压
    CHS0=0;CHS1=1;volflag=1;x=1;DELAY1(x);ADGO=1;
  }
}
else if(volflag==1) {                         //电池采样完毕,进行相应的处理
   CHS1=0;CHS0=1;volflag=0;voltage=ADRESH;lowpower=1;
   }
   else     {                                 //否则,中断为采样电流中断
    speedcount=speedcount+1;                  //speedcount 寄存器加 1,作为测量转速用
    if(speedcount>0x3d)   sp1=1;              //如果转速低于 1 000 000 μs/(512 μs * 3eh * 3),则认为
                                              //为低速状态
    currenth=ADRESH;
    curpid=1;
    count_ts=count_ts-1;
    if(count_ts==0) {                         //如果手柄时间到,则转入手柄采样通道
       CHS0=0;count_ts=0x08;ts=1;x=1;DELAY1(x);ADGO=1;
      }
   }
}
//----------刹车处理子程序----------
void BREAKON()
{
    char x;
    off=0;                                    //off 清 0,如果是干扰则不复位
    shutdown=0;
    if(RB0==1)     {                          //如果刹车信号为真,则停止输出电压
        ADIE=0;                               //关 AD 中断
        INTE=0;                               //关刹车中断
        CCPR1L=FULLDUTY;                      //输出电压 0
        TMR1ON=0;                             //关 CCP2,不再触发 AD
        for(;ADGO==1;)     continue;          //如正在采样,则等待采样结束
        ADIF=0;                               //ADIF 位清 0
```

```
        CHS0=0;                                    //选择通道 0 采样手柄
        CHS1=0;
        x=1;
    DELAY1(x);
        do      {
            ADG0=1;
            for(;ADIF==0;)continue;
            ADIF=0;
            CCPR1L=FULLDUTY;
        asm("CLRWDT");
            tsh=(ADRESH>>1);
            }while(tsh>TSON||RB0==1);              //当手柄值大于 2.2 V 或刹车仍旧继续时,执行以上
                                                   //语句
            off=1;                                 //置复位标志
    }
{
//- ----------欠压保护子程序-------------
void POWER()
{
    char x;
    lowpower=0;
    voltage>>=1;                                   //电压值换为 7 bit,以利于单字节运算
    if(voltage<VOLOFF) {                           //电池电压小于 3 * k(V)时保护
        ADIE=0;
        INTE=0;
        TMR1ON=0;
        CCPR1L=FULLDUTY;
        for(;ADG0==1;)continue;
        ADIF=0;
        CHS0=0;CHS1=1;
        x=1;
    DELAY1(x);
        do{ADG0=1;
            for(;ADIF==0;)continue;
            ADIF=0;
            voltage=(ADRESH>>1);
            CCPR1L=FULLDUTY;
```

```
            asm("CLRWDT");
            }while(voltage<VOLON);              //电池电压小于 35 V 时继续保护
            off=1;                              //置复位标志
        }
}
//----------电流环运算子程序----------
void CURPI()
{     static int curep=0x00,curek=0x00,curuk=0x00;
      union data{int pwm;
          char a[2];}b;                         //定义电流环运算寄存器
      curpid=0;                                 //清电流运算标志
      curep=curek * CURB;                       //计算上一次偏差与比例系数的积
      if(currenth<2)currenth=2;                 //如果采样电流为 0,则认为有一个小电流,以利于
                                                //使转速下降
      currenth>>=1;
      curek=gcur-currenth;                      //计算本次偏差
      curuk=curuk+curek * CURA-curep;           //按闭环 PI 运算方式得到本次输出结果,下
                                                //面对结果进行处理
      if(curuk<0x00) {                          //如果输出小于 0,则认为输出为 0
        curuk=0;CCPR1L=FULLDUTY;CCP1X=0;CCP1Y=0;
        }
      else if(curuk-THL>=0)  {                  //如果输出大于限幅值,则输出最大电压
              curuk=THL;CCPR1L=0;CCP1X=0;CCP1Y=0;
          }
      else      {                               //否则,按比例输出相应的高电平时间到 CCPR1 寄存器
                  b.pwm=THL-curuk;
                 b.pwm<<=1;
                  CCPR1L=b.a[1]; file;          //将 PWM 寄存器的高字节送到 CCPR1L 寄存器
                  if(b.pwm&0x80!  =0) CCP1X=1;
                  else CCP1X=0;
                  if(b.pwm&0x40!  =0) CCP1Y=1;
                  else CCP1Y=0;
      }
}
//----------转速环运算子程序----------
void SPEPI()
{     static int speep=0x00,speek=0x00,speuk=0x00;
```

```
int tsh1,speed1;                                    //转速寄存器定义
 spepid=0;                                          //清转速运算标志
if(spe: =1)      speed1=0x00;                       //若转速太低,则认为转速为0
else speed1=0x7f-speed;                             //否则计算实际转速
if(speed1<0) speed1=0;
speep=speek*SPEB;
tsh1=tsh-0x38;                                      //得到计算用的手柄值
speek=tsh1-speed1;
if(tsh1<0) {speuk=0;gcur=0;}                        //当手柄值低于1.1 V时,则认为手柄给定为0
else {                                              //否则,计算相应的转速环输出
      if(tsh1>=GSPEH)                               //限制最大转速
      tsh1=GSPEH;
      speuk=speuk+speek*SPEA-speep;                 //计算得转速环输出
      if(speuk<=0X00) {speuk=0x00;gcur=0x00;}       //转速环输出处理
      else if(speuk>GCURHILO)     {                 //转速环输出限制,即限制最大电流约12 A
          speuk=GCURHILO;gcur=GCURH;}
      else          {                               //调速状态时的输出
          gcur=(speuk>>4)&0x0ff;
      }
   }
}
//----------主程序----------
main()
{
   for(;;){
      INIT877();                                    //单片机复位后,先对其进行初始化
      off=0;                                        //清复位标志
      for(;off==0;)     {                           //复位标志为0,则执行下面程序,否则复位
      if(curpid==1) CURPI();                        //电流PI运算
      else if(spepid==1)  SPEPI();                  //转速PI运算
      else if(lowpower==1)     POWER();
      else if(shutdown==1)     BREAKON();
      asm("CLRWDT");
   }
  }
}
//----------中断服务子程序----------
```

```
#pragma interrupt_level 1
void interrupt INTS(void)
{
    if(RBIF==1)      {RBIF=0;sample();}
    else if(ADIF==1)      AD();
    else if(INTF==1)      {shutdown=1;INTF=0;}   //刹车中断来,置刹车标志
}
```

第 15 章　液晶显示模块编程

由于液晶显示器(LCD)具有功耗低、体积小、质量轻、超薄等诸多其他显示器无法比拟的优点，已广泛应用于各种智能型仪器和低功耗电子产品中。

点阵式(或图形式)LCD 不仅可以显示字符、数字，还可以显示各种图形、曲线及汉字，并且可以实现屏幕上下左右滚动、动画、闪烁、文本特征显示等功能，用途十分广泛。

本章在简介液晶显示器 MG－12232 的驱动器 SED1520F0A 的结构、功能的基础上，介绍 PIC16F877 单片机的 LCD 硬件接口电路和软件编程特点。

15.1　PIC16F877 与 MG－12232 的硬件接口电路

15.1.1　SED1520F0A 的接口信号

SED1520F0A 是行列驱动及控制合一的小规模液晶显示驱动芯片，电路简单，经济实用；它内含振荡器，只需外接振荡电阻即可工作，模块工作的稳定性好。SED1520F0A 与微处理器的接口信号如下。

DB0～DB7 ——数据总线。

A0 ——数据/指令选择信号。A0＝1，表示出现在数据总线上的是数据；A0＝0，表示出现在数据总线上的是指令或读出的状态。

$\overline{RES}$ ——接口时序类型选择。$\overline{RES}$＝1，为 M6800 时序，操作信号是 E，R/W；$\overline{RES}$＝0，为 Intel8080 时序，操作信号是 $\overline{RD}$，$\overline{WR}$。

$\overline{RD}$(E) —— 在 Intel8080 时序时为读，低电平有效；在 M6800 时序时为使能信号，该信号是个正脉冲，在下降沿处为写操作，在高电平时为读操作。

$\overline{WR}$(R/W)——在 Intel8080 时序时为写，低电平有效；在 M6800 时序时为读、写选择信号，R/W＝1 为读，R/W＝0 为写。

SED1520F0A 与 2 种总线的接口信号和时序的详细资料见《液晶显示模块使用手册》。

15.1.2　MG－12232 模块的引脚说明

MG－12232 模块共有 18 个引脚，定义如表 15.1 所列。

表 15.1　MG－12232 模块的引脚定义

序　号	符　号	状　态	功　能
1	V_{cc}	—	逻辑电源正
2	GND	—	逻辑电源地
3	V0		液晶显示驱动电源
4	$\overline{RES}$	—	接口时序类型选择
5	E1	输　入	主工作方式 IC 使能信号
6	E2	输　入	从工作方式 IC 使能信号
7	R/W	输　入	读/写选择信号
8	A0	输　入	寄存器选择信号
9～16	DB0～DB7	3 态	数据总线(低位)
17	SLA		
18	SLK	—	背光灯负电源

15.1.3　PIC16F877 与 MG－12232 模块的接口电路

现以功能强、价格便宜的 PIC16F877 为例，说明 PIC 与 MG－12232 模块的硬件接口电路。其接口控制时序采用了 M6800 操作时序，这样 SED1520F0A 引出的控制信号 R/W，A0，E1 及 E2 由 PIC16F877 的 I/O 端口 PORTB 的 4 个脚控制。接口电路如图 15.1 所示。由于 PIC16F877 的位操作指令丰富，所以用软件不难模拟出 M6800 操作时序(该接口电路可以在实验板上把跳针 J9 短接后直接得到)。

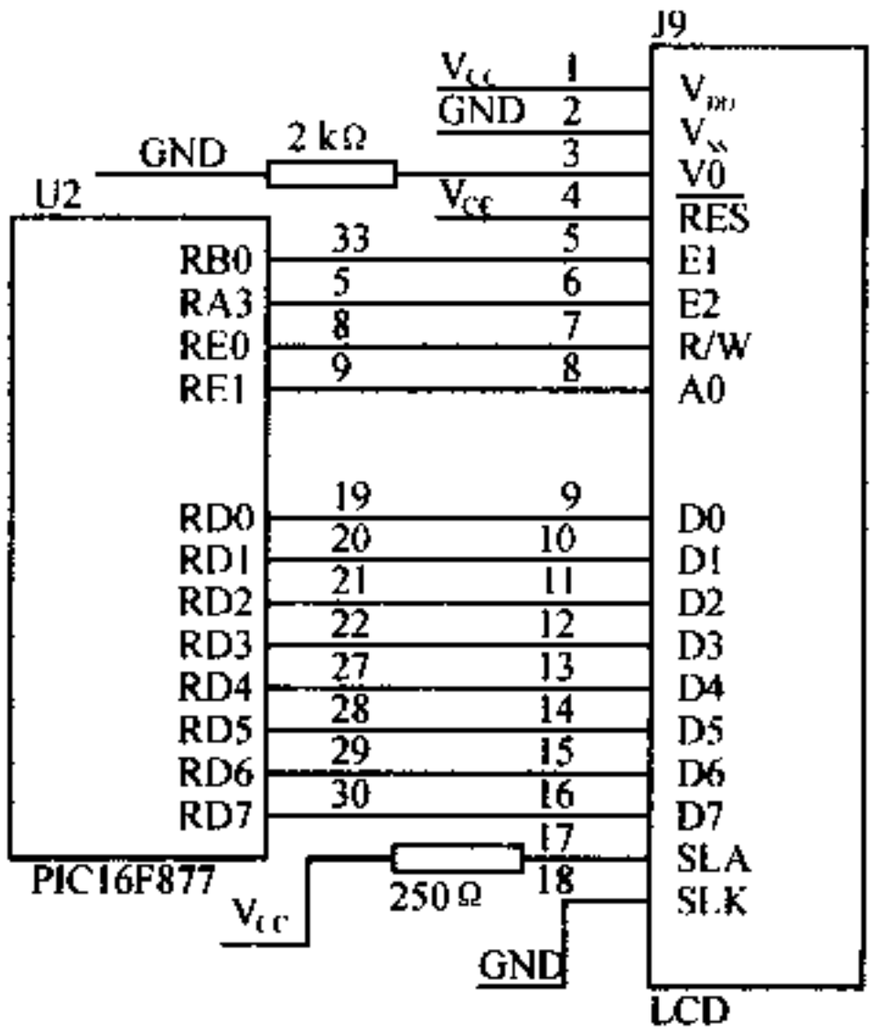


图 15.1　液晶与 PIC16F877 的接口

15.2 软件编程

15.2.1 SED1520F0A的指令集

SED1520F0A液晶显示驱动器有13条指令，下面以与M6800系列MPU接口为例(RES--1)，总结这些指令，如表15.2所列。

表15.2 SED1520F0A的指令集

指令名称	控制信号		控制代码							
	R/W	A0	D7	D6	D5	D4	D3	D2	D1	D0
显示开/关指令	0	0	1	0	1	0	1	1	1	1/0
显示起始行设置	0	0	1	1	0	显示起始行(0～31)				
设置页地址	0	0	1	0	1	1	1	0	页地址(0～3)	
设置列地址	0	0	0	列地址(0～79)						
读状态指令	1	0	BUSY	ADC	OFF/ON	RESET	0	0	0	0
写数据	0	1	显示的数据							
读数据	1	1	显示的数据							
ADC选择指令	0	0	1	0	1	0	0	0	0	0/1
静态驱动开/关	0	0	1	0	1	0	0	1	0	0/1
占空比选择	0	0	1	0	1	0	1	0	0	0/1
改写开始指令	0	0	1	1	1	0	0	0	0	0
改写结束指令	0	0	1	1	1	0	1	1	1	0
复 位	0	0	0	1	0					

15.2.2 MG-12232模块的编程

下面以图15.1的接口电路为例。液晶显示区域分成E1边和E2边，下面只含E1边的程序(表15.1中E1=1，E2=0)，E2边(表15.1中E1=0，E2=1)类推。

在系统程序的初始化部分，应对程序中用到的寄存器和临时变量做说明。

```
unsigned char   TRANS;
unsigned char   PAGEADD;          //存放页地址寄存器
unsigned char   PAGENUM;          //存放总页数寄存器
unsigned char   CLMSUM;           //存放总列数寄存器
unsigned char   CLMADD;           //存放列地址寄存器
unsigned char   WRITE;            //存放显示数据寄存器
unsigned char   row;              //存放显示起始行寄存器
unsigned char   i,k;              //通用寄存器
//系统各口的输入输出状态初始化子程序
void INITIAL()
```

```
{
    ADCON1=0X87;                          //设置 PORTA 口和 PORTE 口为数字 I/O 口
    TRISA3=0;
    TRISB0=0;
    TRISE=0X00;                           //设置液晶的 4 个控制脚为输出
}
//读液晶显示器状态子程序
void    LCDSTA1()
{
    while(1)  {
        TRISD=0XFF;                       //设置 D 口为输入
        RB0=1;                            //E1=1
        RA3=0;                            //E2=0
        RE0=1;                            //R/W=1
        RE1=0;                            //A0=0
        if(RD7==0)     break;             //为忙状态,则继续等待其为空闲
  }
}
//对液晶显示器发指令子程序(指令保存在 TRANS 寄存器中)
void    TRANS1()
{
    LCDSTA1();                            //判断液晶是否为忙
    TRISD=0X00;                           //置 D 口为输出
    RB0=1;                                //E1=1
    RA3=0;                                //E2=0
    RE0=0;                                //R/W=0
    RE1=0;                                //A0=0
    PORTD=TRANS;                          //需要写入的命令字送入数据线
    RB0=0;                                //E1=0 写入指令
    RE0=1;                                //R/W=1
}
//对液晶显示器写数据子程序(数据保存在 WRITE 寄存器中)
void    WRITE1()
{
    TRANS=CLMADI;                         //设置列地址
    TRANS1();
    LCDSTA1();                            //查询液晶是否为空闲
```

```
    TRISD=0X00;                          //D口为输出
    RB0=1;//E1=1
    RA3=0;//E2=0
    RE0=0;//R/W=0
    RE1=1;//A0=1
    PORTD=WRITE;                         //需要写入的数据放入D口
    RB0=0;                               //E1=0,写入数据
    CLMADD++;                            //列地址加1
    RE0=1;                               //R/W=1
}
//开E1显示子程序
void    DISP1()
{
while(1)
  {
    TRANS=0XAF;
    TRANS1();                                    //送出控制命令
    LCDSTA1();                                   //判断液晶是否为空闲
    TRISD=0XFF;                                  //设置D口为输入
    RB0=1;                                       //E1=1
    RA3=0;                                       //E2=0
    RE0=1;                                       //R/W=1
    RE1=0;                                       //A0=0
    if(RD5==0)      break;                       //如果液晶没被关闭,则继续关
    }
}
//E1边清屏子程序
void    CLEAR1()
{
    PAGEADD=0xB8;                                //设置页地址代码
    for(PAGENUM=0X04;PAGENUM>0;PAGENUM--){
        TRANS=PAGEADD;
        TRANS1();
        CLMADD=0x00;                             //设置起始列
     for(CLMSUM=0X50;CLMSUM>0;CLMSUM--){
            LCDSTA1();                           //判断液晶是否为空闲
            WRITE=0X00;
```

```
                WRITE1();                    //写入 00H 以清屏
            }
            PAGEADD++;                       //页号增 1
        }
    }
    //关 E1 显示子程序
    void        DISOFF1()
    {
      while(1)
      {
        TRANS=0XAE;
        TRANS1();                            //发出控制命令
        LCDSTA1();                           //判断液晶是否为空闲
        TRISD=0XFF;                          //D 口设置为输入
        RB0=1;                               //E1=1
        RA3=0;                               //E2=0
        RE0=1;                               //R/W=1
        RE1=0;                               //A0=0
        if(RD5==1)      break;               //如果液晶没被关闭,则继续关
      }
    }
```

有了以上的通用子程序,就可以构造出各种显示程序,如字符、汉字、曲线等。执行这些程序前,必须对液晶进行初始化。初始化的顺序为:关显示→正常显示驱动设置→占空比设置→复位→ADC 选择→清屏→开显示,程序如下。

```
//E1 边初始化
void    lcd1()
{
      DISOFF1();                             //关显示 E1
      TRANS=0XA4;                            //静态显示驱动
      TRANS1();                              //发出控制命令
      TRANS=0XA9;                            //占空比为 1/32
      TRANS1();                              //发出控制命令
      TRANS=0XE2;                            //复位
      TRANS1();                              //发出控制命令
          TRANS=0XA0;                        //ADC 选择正常输出
          TRANS1();                          //发出控制命令
      CLEAR1();                              //清屏
```

```
    LCDSTA1();                          //判断液晶是否为空闲
    DISP1();                            //开显示
}
```

字符、汉字及曲线显示的原理是相似的。它们都是以字节为单位进行显示，关键在于形成字模库时，必须保证每个字节数据的最高位与每一列最下面一个点相对应，最低位与每一列最上面点相对应。这当然可以用专门的软件生成相应的代码，然后再将这些代码逐字节地写到相应的页和列。程序较简单，由于篇幅关系，不再赘述。

15.3 液晶显示屏的结构

该液晶分为E1，E2两边(如右图所示)，每边又分为第0～3页，每页有8行、62列，每列对应D口送出的1个数据(8 bit)。一般而言，显示一个汉字的点阵为16行×16列；显示一个字母或数字的点阵为16行×8列。假如要在液晶的左上角显示一个“电”字，则需在第2页的第0列开始连续送16个数(每个数代表1列)，然后在第3页第0列开始连续送16个数。当然送出的数据应事先用软件生成。

E1边	E2边
第2页	第2页
第3页	第3页
第0页	第0页
第1页	第1页

15.4 程序清单

下面给出一个已经在模板上调试通过的程序。注意在调试该程序时，需把模板上的J9跳针短接。

```
# include          <pic. h>
//该程序用于液晶显示功能的演示
//运行程序后，液晶上显示“电流有效值”和“电压有效值”字样
//系统总的初始化子程序
unsigned    char    TRANS;
unsigned    char    PAGEADD;          //存放页地址寄存器
unsigned    char    PAGENUM;          //存放总页数寄存器
unsigned    char    CLMSUM;           //存放总列数寄存器
unsigned    char    CLMADD;           //存放列地址寄存器
unsigned    char    WRITE;            //存放显示数据寄存器
unsigned    char    row;              //存放显示起始行寄存器
unsigned    char    i,k;              //通用寄存器
const   char   table[192]={0X00,0XF8,0X48,0X48,0X48,0X48,0XFF,0X48,
```

```
0X48,0X48,0X48,0XFC,0X08,0X00,0X00,0X00,
0X00,0X07,0X02,0X02,0X02.0X02,0X3F,0X42,
0X42,0X42.0X42,0X47,0X40.0X70,0X00,0X00,    //"电"
0X00,0X00.0XFE,0X02,0X82,0X82,0X82,0X82,
0XFE,0X82.0X82,0X82,0XC3,0X82.0X00,0X00,
0X40,0X30.0X0F,0X40,0X40.0X40,0X40.0X40,
0X7F,0X40,0X42,0X44,0X4C,0X60.0X40,0X00.    //"压"
0X04,0X04,0X04,0X84,0XE4,0X3C,0X27,0X24,
0X24,0X24,0X24,0XF4,0X24,0X06,0X04 ,0X00,
0X4 ,0X2 ,0X1 ,0X0 .0XFF,0X9,0X9 ,0X9,
 0X9 ,0X49,0X89,0X7F,0X0,0X0,0X0 ,0X0,      //"有"
 0X88,0X48,0XB8,0X9,0XA,0X98,0X2C ,0X48,
 0X20,0XD0,0X1F,0X10,0X10,0XF8,0X10 ,0X0,
 0X40,0X20,0X18,0X5,0X2,0XD,0X30 ,0X80,
 0X80,0X41,0X36,0X8,0X37,0XC0,0X40 ,0X0,    //"效"
 0X80,0X40,0X20,0XF8.0X7,0X4,0XE4.0XA4,
 0XA4,0XBF,0XA4,0XA4,0XF6,0X24,0X0 ,0X0,
 0X0,0X0,0X0,0XFF,0X40,0X40,0X7F,0X4A,
 0X4A,0X4A,0X4A,0X4A,0X7F,0X40,0X40 ,0X0, //"值"
 0X10,0X22,0X64,0XC,0X80,0X44,0X44,0X64,
  0X55,0X4E,0X44,0X54,0X66,0XC4,0X0,0X0,
 0X4,0X4,0XFE,0X1,0X0,0X80,0X40,0X3F,
  0X0,0XFF,0X0,0X3F.0X40,0X40,0X70.0X0      //"流"
};
//系统各口的输入输出状态初始化子程序
void        INITIAL()
{
    ADCON1=0X87;                            //设置 PORTA 口和 PORTE 口为数字 I/O 口
    TRISA3=0;
    TRISB0=0;
    TRISE=0X00;                             //设置液晶的 4 个控制脚为输出
}
//读液晶显示器状态子程序
void        LCDSTA1()
{
 while(1){
        TRISD=0XFF;                         //设置 D 口为输入
```

```
        RB0=1;                                  //E1=1
    RA3=0;                                      //E2=0
    RE0=1;                                      //R/W=1
    RE1=0;                                      //A0=0
    if(RD7==0)      break;                      //为忙状态,则继续等待其为空闲
  }
}
//对液晶显示器发指令子程序(指令保存在 TRANS 寄存器中)
void        TRANS1()
{
    LCDSTA1();                                  //判断液晶是否为忙
    TRISD=0X00;                                 //D 口为输出
    RB0=1;                                      //E1=1
    RA3=0;                                      //E2=0
    RE0=0;                                      //R/W=0
    RE1=0;                                      //A0=0
    PORTD=TRANS;                                //需要写入的命令字送入数据线
    RB0=0;                                      //E1=0 写入指令
    RE0=1;                                      //R/W=1
}
//对液晶显示器写数据子程序(数据保存在 WRITE 寄存器中)
void        WRITE1()
{
    TRANS=CLMADD;                               //设置列地址
    TRANS1();
    LCDSTA1();                                  //查询液晶是否为空闲
    TRISD=0X00;                                 //D 口为输出
    RB0=1;                                      //E1=1
    RA3=0;                                      //E2=0
    RE0=0;                                      //R/W=0
    RE1=1;                                      //A0=1
    PORTD=WRITE;                                //需要写入的数据放入 D 口
    RB0=0;                                      //E1=0,写入数据
    CLMADD++;                                   //列地址加 1
    RE0=1;                                      //R/W=1
}
//开 E1 显示子程序
```

```
void          DISP1()
{
    while(1)  {
    TRANS=0XAF;
    TRANS1();                          //送出控制命令
    LCDSTA1();                         //判断液晶是否为空闲
    TRISD=0XFF;                        //设置 D 口为输入
    RB0=1;                             //E1=1
    RA3=0;                             //E2=0
    RE0=1;                             //R/W=1
    RE1=0;                             //A0=0
    if(RD5==0)     break;              //如果液晶没被关闭,则继续关
    }
}
//E1 边清屏子程序
void          CLEAR1()
{
    PAGEADD=0xB8;                      //设置页地址代码
    for(PAGENUM=0X04;PAGENUM>0;PAGENUM--){
    TRANS=PAGEADD;
    TRANS1();
    CLMADD=0x00;                       //设置起始列
      for(CLMSUM=0X50;CLMSUM>0;CLMSUM--) {
        LCDSTA1();                     //判断液晶是否为空闲
        WRITE=0X00;
        WRITE1();                      //写入 00H 以清屏
      }
      PAGEADD++;                       //页号增 1
   }
}
//关 E1 显示子程序
void          DISOFF1()
{
  while(1) {
    TRANS=0XAE;
    TRANS1();                          //发出控制命令
    LCDSTA1();                         //判断液晶是否为空闲
```

```
    TRISD=0XFF;                          //D 口设置为输入
    RB0=1;                               //E1=1
    RA3=0;                               //E2=0
    RE0=1;                               //R/W=1
    RE1=0;                               //A0=0
    if(RD5==1)      break;               //如果液晶没被关闭,则继续关
  }
}
//E1 边初始化
void        lcd1()
{
    DISOFF1();                           //关显示 E1
    TRANS=0XA4;                          //静态显示驱动
    TRANS1();                            //发出控制命令
    TRANS=0XA9;                          //占空比为 1/32
    TRANS1();                            //发出控制命令
    TRANS=0XE2;                          //复位
    TRANS1();                            //发出控制命令
    TRANS=0XA0;                          //ADC 选择正常输出
    TRANS1();                            //发出控制命令
    CLEAR1();                            //清屏
    LCDSTA1();                           //判断液晶是否为空闲
    DISP1();                             //开显示
}
//E2 边的处理部分
//读液晶显示器状态子程序
void        LCDSTA2()
{
    while(1) {
    TRISD=0XFF;                          //设置 D 口为输入
    RB0=0;                               //E1=0
    RA3=1;                               //E2=1
    RE0=1;                               //R/W=1
    RE1=0;                               //A0=0
    if(RD7==0)      break;               //为忙状态,则继续等待其为空闲
  }
}
```

```
//对液晶显示器发指令子程序,指令保存在 TRANS 寄存器中
void        TRANS2()
{
    LCDSTA2();                          //判断液晶是否为忙
    TRISD=0X00;                         //D 口为输出
    RB0=0;                              //E1=0
    RA3=1;                              //E2=1
    RE0=0;                              //R/W=0
    RE1=0;                              //A0=0
    PORTD=TRANS;                        //需要写入的命令字送入数据线
    RA3=0;                              //E2=0 写入指令
    RE0=1;                              //R/W=1
}
//对液晶显示器写数据子程序(数据保存在 WRITE 寄存器中)
void        WRITE2()
{
    TRANS=CLMADD;                       //设置列地址
    TRANS2();
    LCDSTA2();                          //查询液晶是否为空闲
    TRISD=0X00;                         //D 口为输出
    RB0=0;                              //E1=0
    RA3=1;                              //E2=1
    RE0=0;                              //R/W=0
    RE1=1;                              //A0=1
    PORTD=WRITE;                        //需要写入的数据放入 D 口
    RA3=0;                              //E2=0,写入数据
    CLMADD++;                           //列地址加 1
    RE0=1;                              //R/W=1
}
//开 E2 显示子程序
void        DISP2()
{
    while(1)  {
    TRANS=0XAF;
    TRANS2();                           //送出控制命令
    LCDSTA2();                          //判断液晶是否为空闲
    TRISD=0XFF;                         //设置 D 口为输入
```

```
        RB0=0;                                  //E1=0
        RA3=1;                                  //E2=1
        RE0=1;                                  //R/W=1
        RE1=0;                                  //A0=0
        if(RD5==0)      break;                  //如果液晶没被关闭,则继续关
    }
}
//E2 边清屏子程序
void        CLEAR2()
{
    PAGEADD=0xB8;                               //设置页地址代码
    for(PAGENUM=0X04;PAGENUM>0;PAGENUM--){
        TRANS=PAGEADD;
        TRANS2();
        CLMADD=0x00;                            //设置起始列
        for(CLMSUM=0X50;CLMSUM>0;CLMSUM--){
            LCDSTA2();                          //判断液晶是否为空闲
            WRITE=0X00;
            WRITE2();                           //写入 00H 以清屏
        }
        PAGEADD++;                              //页号增 1
    }
}
//关 E2 显示子程序
void        DISOFF2()
{
    while(1){
    TRANS=0XAE;
    TRANS2();                                   //发出控制命令
    LCDSTA2();                                  //判断液晶是否为空闲
    TRISD=0XFF;                                 //D 口设置为输入
    RB0=0;                                      //E1=0
    RA3=1;                                      //E2=1
    RE0=1;                                      //R/W=1
    RE1=0;                                      //A0=0
    if(RD5==1)      break;                      //如果液晶没被关闭,则继续关
    }
```

```
}
//E2 边初始化
void        lcd2()
{
    DISOFF2();                          //关显示 E1
    TRANS=0XA4;                         //静态显示驱动
    TRANS2();                           //发出控制命令
    TRANS=0XA9;                         //占空比为 1/32
    TRANS2();                           //发出控制命令
    TRANS=0XE2;                         //复位
    TRANS2();                           //发出控制命令
    TRANS=0XA0;                         //ADC 选择正常输出
    TRANS2();                           //发出控制命令
    CLEAR2();                           //清屏
    LCDSTA2();                          //判断液晶是否为空闲
    DISP2();                            //开显示
}
//LCD 的 E1 边显示函数,调用一次该函数,则在相应的位置显示相应的字
void        dis1()
{
     TRANS=row;
     TRANS1();
     TRANS=PAGEADD;
     TRANS1();
     i=i*32;                                    //i 变成数组指示指针
for(k=0;k<16;k++)     {
    WRITE=table[i+k];                           //查得需要显示的字节
    WRITE1();                                   //在 WRITE1 子程序里面,列地址加 1
}
CLMADD=CLMADD-16;                               //恢复列地址
PAGEADD=PAGEADD+1;                              //页地址加 1
TRANS=PAGEADD;
TRANS1();
for(;k<32;k++)     {
    WRITE=table[i+k];                           //查得需要显示的字节
    WRITE1();                                   //在 WRITE1 子程序里面,列地址已经加 1
}
```

```
}
//LCD 的 F2 边显示函数,调用一次该函数,则在相应的位置显示相应的字
void        dis2()

    TRANS=row;
    TRANS2();
    TRANS=PAGEADD;
    TRANS2();
    i=i*32;                                          //i 变成数组指示指针
    for(k=0;k<16;k++)     {
        WRITE=table[i+k];                            //查得需要显示的字节
        WRITE2();                                    //在 WRITE1 子程序里面,列地址已经加 1
        }
    CLMADD=CLMADD-16;                                //恢复列地址
    PAGEADD=PAGEADD+1;                               //页地址加 1
    TRANS=PAGEADD;
    TRANS2();
    for(;k<32;k++)     {
       WRITE=table[i+k];                             //查得需要显示的字节
        WRITE2();                                    //在 WRITE1 子程序里面,列地址已经加 1
    }
}
//主程序
main()
{
    INITIAL();                                       //系统初始化
    lcd1();                                          //E1 边初始化
    lcd2();                                          //E2 边初始化
    row=0XC0;                                        //显示起始列为第 0 行
//以下显示不同的字符
    PAGEADD=0XB8;                                    //显示起始页为第 0 页
    CLMADD=0X00;                                     //起始列为第 0 列
    i=0;                                             //显示数组中对应的第 1 个字
    dis1();                                          //调用显示函数
    PAGEADD=0XB8;                                    //显示起始页为第 0 页
    CLMADD=16;                                       //起始列为第 16 列
    i=1;                                             //显示数组中对应的第 2 个字
```

```
    dis1();                          //调用显示函数
    PAGEADD=0XB8;                    //显示起始页为第 0 页
    CLMADD=32;                       //起始列为第 32 列
    i=2;                             //显示数组中对应的第 3 个字
    dis1();                          //调用显示函数
    PAGEADD=0XB8;                    //显示起始页为第 0 页
    CLMADD=48;                       //起始列为第 48 列
    i=3;                             //显示数组中对应的第 4 个字
    dis1();                          //调用显示函数
    PAGEADD=0XB8;                    //显示起始页为第 0 页
    CLMADD=0;                        //起始列为第 0 列
    i=4;                             //显示数组中对应的第 5 个字
    dis2();                          //调用 E2 边显示函数
    PAGEADD=0XBA;                    //显示起始页为第 2 页
    CLMADD=0X00;                     //起始列为第 0 列
    i=0;                             //显示数组中对应的第 1 个字
    dis1();                          //调用显示函数
    PAGEADD=0XBA;                    //显示起始页为第 2 页
    CLMADD=16;                       //起始列为第 16 列
    i=5;                             //显示数组中对应的第 6 个字
    dis1();                          //调用显示函数
    PAGEADD=0XBA;                    //显示起始页为第 2 页
    CLMADD=32;                       //起始列为第 32 列
    i=2;                             //显示数组中对应的第 3 个字
    dis1();                          //调用显示函数
    PAGEADD=0XBA;                    //显示起始页为第 2 页
    CLMADD=48;                       //起始列为第 48 列
    i=3;                             //显示数组中对应的第 4 个字
    dis1();                          //调用显示函数
    PAGEADD=0XBA;                    //显示起始页为第 2 页
    CLMADD=0;                        //起始列为第 0 列
    i=4;                             //显示数组中对应的第 5 个字
    dis2();                          //调用 E2 边显示函数
    while(1)      {
        ;
    }
}
```

参 考 文 献

1 刘和平，黄开长，严利平 译著. PIC16F87X 数据手册. 北京：北京航空航天大学出版社，2001

2 窦振中，汪立森. PIC 系列单片机的应用设计与实例. 北京：北京航空航天大学出版社，1999

3 王有绪，许杰，李拉成. PIC 系列单片机接口技术及应用系统设计. 北京：北京航空航天大学出版社，2000

4 窦振中. PIC 系列单片机原理和程序设计. 北京：北京航空航天大学出版社，1998

5 邬宽明. CAN 总线原理和应用系统设计. 北京：北京航空航天大学出版社，1996

6 Peatman John B. Design with PIC Microcontrolers

7 Microchip. Stand - Alone CAN Controler with SPI Interface

8 Hi_TECH. PICC_ME Manual. 2001

PIC 系列单片机丛书

《PIC16F87X 单片机实用软件与接口技术
——汇编语言及其应用》 刘和平 等编著 39.00 元

《PIC16F87X 单片机实用软件与接口技术
——C 语言及其应用》 刘和平 等编著 32.00 元

《PIC 系列单片机的开发应用技术》 武 锋 编著 23.00 元

《PIC 系列单片机原理和程序设计》 窦振中 编著 45.00 元

《PIC 系列单片机应用设计与实例》 窦振中 等编著 29.00 元

《PIC 系列单片机接口技术及应用系统设计》 王有绪 等编著 36.00 元

《PIC16F87X 数据手册
——28/40 脚 8 位 FLASH 单片机》 刘和平 等译 22.00 元

《PIC 单片机实用教程——基础篇》 李学海 编著 29.50 元

《PIC 单片机实用教程——提高篇》 李学海 编著 即将出版

ISBN 7-81077-169-8

9 787810 771696 >

ISBN 7-81077-169-8/TP·093

定价：32.00 元(含光盘)