杭州晶华微电子有限公司
Hangzhou SDIC Microelectronics Co.,Ltd.
地址：浙江省杭州市高新之江科技工业园滨康路790号
电话：0571 8667 3071，0571 8667 3068 传真：0571 8667 3072
电邮：info@SDICmicro.cn 网址：www.SDICmicro.cn

SD3003,SD3004 芯片程序开发说明

文件编号：
编写人：朱东升
批准人：

版本号：v0.1
审核人：谢建斌
编写日期： 2011-1-26

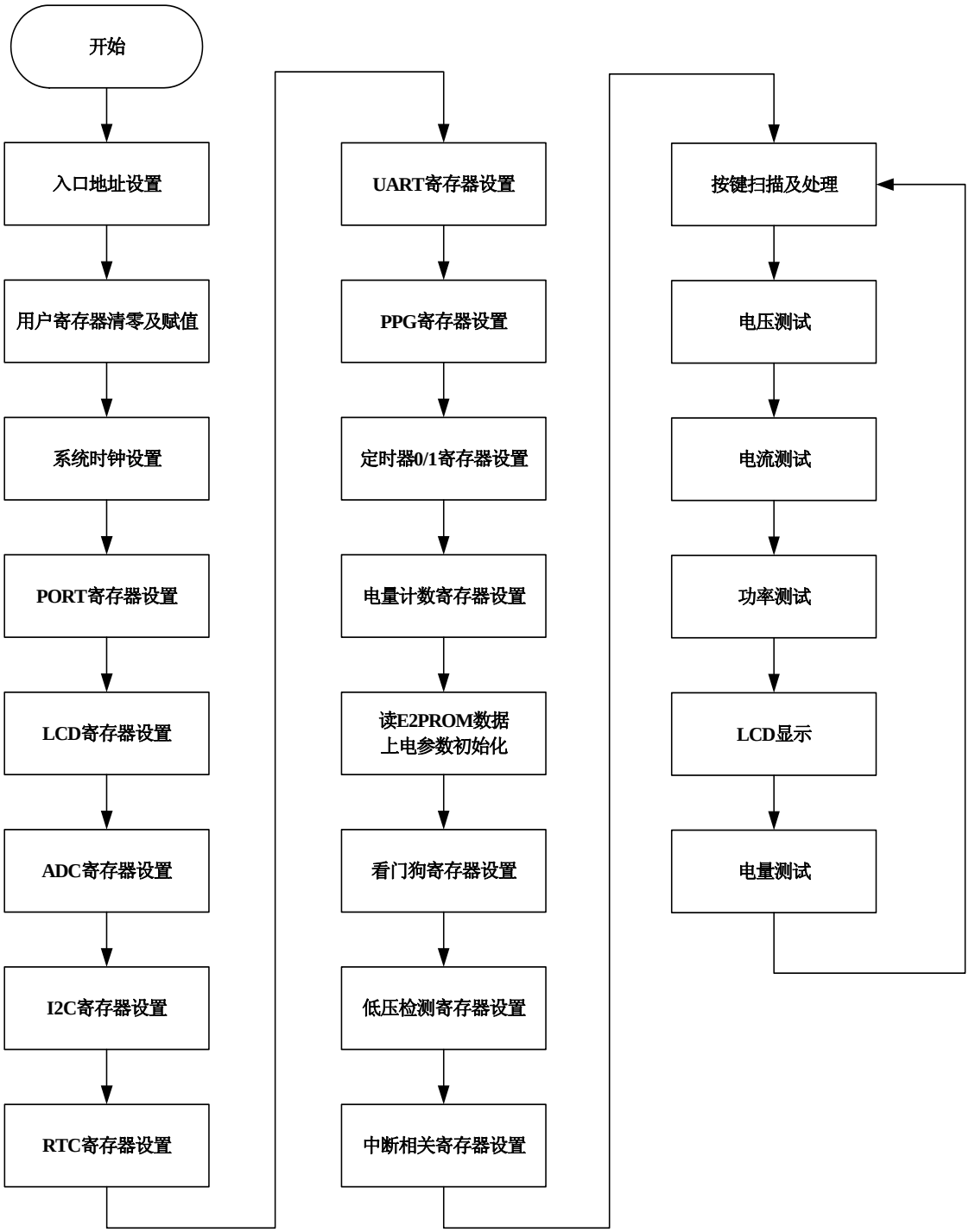
说明

本文件介绍了 SD3003,SD3004 芯片用于计量插座功能的例子程序，为用户开发 SD3003,SD3004 芯片程序提供一个参考，下面逐个介绍程序中的每个程序子模块的实现过程和一些常用子函数的应用及说明。

详细的参考程序源码和参考硬件电路见后面的附件。

如对该文件有任何疑问，请联系杭州晶华微电子有限公司，我们将竭力为用户提供服务与帮助。

1、程序框架：


开始
入口地址设置
用户寄存器清零及赋值
系统时钟设置
PORT寄存器设置
LCD寄存器设置
ADC寄存器设置
I2C寄存器设置
RTC寄存器设置
UART寄存器设置
PPG寄存器设置
定时器0/1寄存器设置
电量计数寄存器设置
读E2PROM数据
上电参数初始化
看门狗寄存器设置
低压检测寄存器设置
中断相关寄存器设置
按键扫描及处理
电压测试
电流测试
功率测试
LCD显示
电量测试

程序框图

2、程序子模块

◆ 2.1 入口地址设置

2.1.1 示例：

```
    org         0x0002              ;可更换为 org  0x0000（加密烧录方式）
    bra         start
    org         0x0008              ;0x0008 为中断入口地址
    bra         int
    org         0x0020
start:
    movlb       0x01                ;选择 BANK
```

◆ 2.2 用户寄存器清零

2.2.1 示例：

```
    btfss       STATUS,5,0          ;判断是正常上电还是看门狗异常复位
    call        Clear_ram           ;清零地址为 0x100～0x17f 数据寄存器
    clrwdt
```

2.2.2 清 SRAM 子函数：

```
Clear_ram:
    movlw       0x01
    movwf       FSR0H,0             ;SRAM 寄存器高位地址
    movlw       0x00
    movwf       FSR0L,0             ;SRAM 寄存器低位地址
Loop_ram:
    movlw       0x00
    movwf       INDF0,0             ;对应 SRAM 地址里的数据
    incf        FSR0L,1,0
    movlw       0x80
    cpfseq      FSR0L,0
    bra         Loop_ram
    return
```

◆ 2.3 系统时钟设置

2.3.1 示例：

```
    movlw       0x33
    movwf       OSCCON,0
    movlw       0x31
    movwf       OSCCON,0            ;选择 PLL 作为系统时钟（3,604,480Hz）
                                    ;PLL 的频率为低频晶振频率 * 110
```

◆ 2.4 PORT 寄存器设置

2.4.1 示例：

```
clrf        PT1PU,0             ;PT1 上拉使能
bsf         PT1EN,0,0           ;设置 PT1.0 口为输出口
bcf         PT1EN,1,0           ;设置 PT1.1 口为输入口
bsf         PT1,0,0             ;PT1.0 口输出高电平
bcf         PT1,0,0             ;PT1.0 口输出低电平
```

◆ 2.5 LCD 寄存器设置

2.5.1 示例:

```
movlw       0x9c                ;系统时钟除以 8192 作为 LCD 时钟
movwf       LCDCON,0
setf        LCDCOM,0            ;使能 LCD 的 COM 和 SEG 引脚
setf        LCDSE0,0
setf        LCDSE1,0
setf        LCDSE2,0
```

◆ 2.6 ADC 寄存器设置

2.6.1 示例：

```
movlw       0xce                ;G=8
movwf       ADCON,0

bsf         ADC_CON2,1,0        ;SEL_IN=1,测试电压值
bcf         ADC_CON2,0,0        ;SEL_CHANNEL=0,选择电压通道进行计算
movlw       0x03                ;PLL 时钟下的累加次数
movwf       ADD_COUNT1,0        ;FOSC/4/512/f0 * N - 1
movlw       0x6f                ;f0 为信号频率，N 为市电周期数
movwf       ADD_COUNT0,0        ;例如信号频率为 50Hz，取 25 个周期
```

◆ 2.7 I2C 寄存器设置

2.7.1 示例：

```
movlw       0x08
movwf       SSPADD,0
movlw       0x28
movwf       SSPCON1,0           ;99.4KHz 波特率，使能 I2C
```

◆ 2.8 RTC 寄存器设置

2.8.1 示例：

```
clrf        SECONDS,0           ;RTC 初始值设置为 00:00:00
clrf        MINUTES,0
clrf        HOURS,0
bsf         HOURS,7,0           ;24 小时制
```

◆ 2.9 UART 寄存器设置

2.9.1 示例：

```
#define     OSC      .3604480     ;系统时钟为 3.604480MHz
#define     BOUND    .9600        ;波特率为 9600
bsf         RCSTA,7,0             ;串口使能
bsf         RCSTA,4,0             ;串口接收使能
bsf         TXSTA,5,0             ;发送使能
bcf         TXSTA,4,0             ;异步模式时钟,UART 模异步模式
movlw       ((OSC*.10)/(BOUND*.64)+5)/.10-1
                                  ;四舍五入
movwf       SPBRG,0
```

◆ 2.10 PPG 寄存器设置

2.10.1 示例：

```
movlw       0x80
movwf       PPG_CON,0             ;使能 8 位 PPG 功能，时钟选择为 OSC/4
movwf       PPG2CON0              ;使能 16 位 PPG 功能，时钟选择为 OSC/4

movlw       0x00
movwf       PPG_SETH,0            ;赋值 8 位 PPG 高电平时间
movwf       PPG_SETL,0            ;赋值 8 位 PPG 周期时间

movwf       PPG2H_H,0             ;赋值 16 位 PPG 高电平时间
movwf       PPG2H_L,0

movwf       PPG2L_H,0             ;赋值 16 位 PPG 周期时间
movwf       PPG2L_L,0
```

◆ 2.11 定时器 0/1 寄存器设置

2.11.1 示例：

```
clrf        TMR0L,0               ;定时器 0 重载值赋值

movlw       0x85                  ;定时器 0 时钟选择及使能
movwf       T0CON,0

clrf        TMR1L,0               ;定时器 1 重载值赋值
clrf        TMR1H,0
clrf        TMR1U,0
movlw       0x03
movwf       T1CON,0               ;定时器 1 时钟选择及使能
```

◆ 2.12 电量计数寄存器设置

2.12.1 示例：

```
bcf         COUNTER_CON,3,0       ;正向溢出标志位清零
bcf         COUNTER_CON,2,0       ;反向溢出标志位清零
```

```
bsf         COUNTER_CON,1,0      ;正向计数器使能
bsf         COUNTER_CON,0,0      ;反向计数器使能
```

◆ 2.13 E2PROM 上电初始化

2.13.1 主要用于一些校准参数的初始化，一般读 E2PROM 里的值赋值到芯片内部相应的用户寄存器里。

◆ 2.14 看门狗设置

2.14.1 示例：

```
bsf         WDTCON,0,0           ;看门狗时钟的选择及使能
bsf         WDTCON,1,0
bsf         WDTCON,2,0
```

◆ 2.15 低压检测寄存器设置

2.15.1 示例：

```
bcf         LVDCON,2,0           ;清 2.2V 低压检测标志位
bcf         LVDCON,3,0           ;清 4.0V 低压检测标志位
bsf         LVDCON,0,0           ;2.2V 低压检测使能
bsf         LVDCON,4,0           ;2.0V 低压检测复位使能
```

◆ 2.16 中断相关寄存器设置

2.16.1 示例：

```
bcf         PIR2,2,0             ;清低压 4v 中断标志位
bcf         PIR2,1,0             ;清定时器 1 中断标志位
bcf         PIR2,0,0             ;清 cap 中断标志位
bcf         PIR1,4,0             ;清定时器 0 中断标志位
bcf         PIR1,3,0             ;清半秒中断标志位

bsf         PIE2,2,0             ;低压4v中断使能
bsf         PIE2,1,0             ;定时器1中断使能
bsf         PIE2,0,0             ;cap中断使能
bsf         T1CON,1,0            ;cap捕捉cf使能
bsf         INTCON,4,0           ;定时器 0 中断使能
bsf         INTCON,3,0           ;半秒中断使能

bsf         INTCON,7,0           ;总中断使能
```

3、一些常用子函数的应用及说明

◆ 3.1 电压计算

3.1.1 示例：

电压测试寄存器设置：

```
bsf         ADC_CON2,1,0
```

```
bcf         ADC_CON2,0,0
movlw       0x03
movwf       ADD_COUNT1,0
movlw       0x6f
movwf       ADD_COUNT0,0
```

计算原理：RLT_ADJ4~ RLT_ADJ1 里的值（这里简称 RLT_ADJ），即为信号 25 个周期瞬时值的平方累加和值，对其[RLT_ADJ/（879）]开方即为信号的有效值，所以计算任意时刻信号有效值计算如下：RLT_ADJ_X 为需要测量的信号平方累加和值，RLT_ADJ_220v 为 220V 下的基准值，V_data 为所需的信号有效值，如式 3-1 所示。

$$V\_data=\frac{\sqrt{RLT\_ADJ\_X}}{\sqrt{RLT\_ADJ\_220v}}*220 \qquad (3\text{-}1)$$

◆ 3.2 电流计算

3.2.1 示例：

电流测试寄存器设置：

```
bcf         ADC_CON2,1,0        ;SEL_IN=1,测试电流值
bcf         ADC_CON2,0,0        ;SEL_CHANNEL=0,选择电流通道
movlw       0x03
movwf       ADD_COUNT1,0
movlw       0x6f
movwf       ADD_COUNT0,0
```

计算原理同电压测试，只不过是将式 3-1 中的 220V 电压基准值更换为 10A 下的电流基准值，如式 3-2 所示。

$$I\_data=\frac{\sqrt{RLT\_ADJ\_X}}{\sqrt{RLT\_ADJ\_10A}}*10000 \qquad (3\text{-}2)$$

◆ 3.3 功率计算

3.3.1 示例：

功率测试寄存器设置：

```
movlw       0x03
movwf       T1CON,0             ;OSC/4,捕捉 CF 使能,定时器 1 开

bcf         PIR2,0,0            ;cap 中断标志
bsf         PIE2,0,0            ;cap 中断使能
bsf         T1CON,1,0           ;cap 捕捉 cf 使能
bsf         INTCON,7,0          ;开总中断
```

捕捉中断子程序：

```
btfsc       PIR2,0,0
```

```
    bra         INT_CF_CAP
    ……
```

捕捉中断处理

```
INT_CF_CAP:
    bcf         PIR2,0,0
    ……
```

计算原理：捕捉中断里记录捕捉到的 CF 上升沿时刻，取两次时刻的差值记录为 CF 的一个周期，称为 T1_CF_CP。设 T1_CF_CP_X 为某次记录的一个 CF 周期，W_data 为显示的功率值，把功率为 2200W 时的 CF 周期作为基准值，称为 T1_CF_CP_2200W，计算方法如式 3-3 所示。

$$W\_data=\frac{T1\_CF\_CP\_2200W}{T1\_CF\_CP\_X}*2200 \qquad (3\text{-}3)$$

◆ 3.4 电量计算

3.4.1 示例：

电量相关寄存器设置：

```
    movlw       0xce
    movwf       ADCON,0             ;G=8,可选择增益倍数和输出高、低频率

    bcf         COUNTER_CON,3,0     ;正向溢出标志位清零
    bcf         COUNTER_CON,2,0     ;反向溢出标志位清零
    bsf         COUNTER_CON,1,0     ;正向计数器使能
    bsf         COUNTER_CON,0,0     ;反向计数器使能

ELC_COUNT_HF:
    movlw       0x01                ;高频模式下取电量计数器值
    movwf       DL_NUM,1            ;128

    btfss       COUNTER_CON,4,0     ;1 则为 DOWN
    bra         UP_COUNT_HF

DOWN_COUNT_HF:
    movf        COUNT_DOWN_L,0,0    ;反向计数
    movwf       DL_SECOND_L,1
    movf        COUNT_DOWN_H,0,0
    movwf       DL_SECOND_H,1
    bra         JUDGE_DL
UP_COUNT_HF:
    movf        COUNT_UP_L,0,0      ;正向计数
    movwf       DL_SECOND_L,1
    movf        COUNT_UP_H,0,0
    movwf       DL_SECOND_H,1
```

```
JUDGE_DL:
    call        DL_CAL
```

作用：开启电表正接反接电量计量，上例高频模式下电量正反接情况下取值及计算。

3.4.2 电量计算子程序：

```
DL_CAL:
    movlw       0x07
    movwf       divdend_Reg_HL,1     ;乘法,移位次数 7

    movf        DL_SECOND_H,0,1      ;当前电量计数器的值
    movwf       divdend_Reg_LH,1
    movf        DL_SECOND_L,0,1
    movwf       divdend_Reg_LL,1

    movf        DL_FIRST_L,0,1       ;当前电量值 - 上次电量值
    subwf       divdend_Reg_LL,1,1
    movf        DL_FIRST_H,0,1
    subwfb      divdend_Reg_LH,1,1

    movlw       0x01
    cpfseq      DL_NUM,1             ;DL_NUM 为 1，不用移位
    bra         Muti_L128            ;DL_NUM 为 128, 左移 7 次
    bra         ADD_DL_REMAIND

Muti_L128:                           ;乘以 128
    bcf         STATUS,0,0
    rlcf        divdend_Reg_LL,1,1
    rlcf        divdend_Reg_LH,1,1
    decfsz      divdend_Reg_HL,1,1
    bra         Muti_L128

ADD_DL_REMAIND:
    movf        divdend_Reg_LL,0,1   ;(当前电量值-上次电量值)*n+剩余电量
    addwf       DL_remaindL,1,1
    movf        divdend_Reg_LH,0,1
    addwfc      DL_remaindH,1,1

    movf        C_DAT_L,0,1          ;C 为 0.01 度电表脉冲常数
                                     ;每计满 C 时电量加 0.01 度
    subwf       DL_remaindL,0,1
    movf        C_DAT_H,0,1
    subwfb      DL_remaindH,0,1
```

```
    btfss       STATUS,C,0   ;【(当前电量值－上次电量值)*n+剩余电量】-C>0?
    bra         END_DL       ;大于 0 则加 0.01 度，小于 0 则退出继续累加

    movf        C_DAT_L,0,1          ;重新计算剩余电量
    subwf       DL_remaindL,1,1
    movf        C_DAT_H,0,1
    subwfb      DL_remaindH,1,1
    incfsz      COUNT_DATA_L,1,1     ;计满 C，加 0.01 度
    bra         END_DL
    incfsz      COUNT_DATA_H,1,1
    bra         END_DL
    incf        COUNT_DATA_U,1,1     ;COUNT_DATA_U/H/L 为总电量寄存器

END_DL:
    movf        DL_SECOND_H,0,1      ;更新上次电量值
    movwf       DL_FIRST_H,1
    movf        DL_SECOND_L,0,1
    movwf       DL_FIRST_L,1
    return
```

◆ 3.5 IO 口输入输出

3.5.1 示例：

```
    PORT 口寄存器相关设置
    movlw       0x00
    movwf       PT1PU,0              ;内部上拉电阻使能

    bsf         PT1EN,0,0            ;PT1.0 口作为输出口
    bcf         PT1EN,1,0            ;PT1.1 口作为输入口

    btfss       PT1,1,0              ;判断 PT1.1 口的电平
    bra         SET_HIGH

    bcf         PT1,0,0
    bra         NEXT

SET_HIGH:
    bsf         PT1,0,0

NEXT:
    ……
```

作用：检测 PT1.0 口的电平，为高电平则 PT1.1 口置低，为低电平则 PT1.1 口置高。

◆ 3.6 掉电保存

3.6.1 示例：

相关寄存器设置：

```
bcf         LVDCON,1,0          ;低压 4v 检测标志位
bcf         PIR2,2,0            ;低压 4v 中断标志位
bsf         LVDCON,3,0          ;低压检测使能位
bsf         PIE2,2,0            ;低压 4v 中断使能位
bsf         INTCON,7,0          ;开总中断

中断服务子程序：
btfsc       PIR2,2,0            ;判断是否掉电，是则跳转保存数据
bra         Save_Data
 ……

掉电保存：
Save_Data:
bcf         PIR2,2,0
……
```

作用：芯片电源低于 4.0v 后，进低压检测中断，保存数据。

◆ 3.7 定时器 0/1 溢出中断

3.7.1 示例：

```
定时器 0 初始化设置:
clrf        TMR0L,0             ;中断周期为 15.625ms

movlw       0x85                ;OSC32K/2 作为定时器 0 时钟
movwf       T0CON,0

bcf         PIR1,4,0            ;清定时器 0 中断标志位
bsf         INTCON,4,0          ;定时器 0 中断

……

中断程序处理：
btfsc       PIR1,4,0            ;标志位为 1 则跳到定时器 0 中断处理
bra         INT_T0              ;定时器 0 处理完成后必须清零标志位

……
定时器 0 中断处理程序
INT_T0:
bcf         PIR1,4,0
……
```

作用：定时器 0/1 溢出中断程序结构，定时器 0 中断时间为 15.625ms，定时器 1 溢出中断程序结构与定时器 0 溢出中断类同。

◆ 3.8 RTC 实时时钟

3.8.1 示例：

半秒中断相关设置：

```
bcf         PIR1,3,0            ;清半秒中断标志位
bsf         INTCON,3,0          ;半秒中断使能

中断程序处理：
btfsc       PIR1,3,0            ;标志位为 1 则跳到半秒中断处理
bra         INT_SEC

……

半秒中断处理程序
INT_SEC:
bcf         PIR1,3,0            ;半秒中断处理完成后必须清零标志位

……                              ;可对用户定义的时/分/秒寄存器累加
```

作用：开启半秒中断，每 0.5s 程序会自动中断一次，每来一次中断时用户可以对自己定义的时/分/秒寄存器进行累加，从而达到计时的效果。

◆ 3.9 UART 通讯

3.9.1 示例：

UART 寄存器设置：

```
#define     OSC         .3604480;系统时钟为 3.604480MHz
#define     BOUND       .9600   ;波特率为 9600

bsf         RCSTA,7,0           ;串口使能
bsf         RCSTA,4,0           ;串口接收使能
bsf         TXSTA,5,0           ;发送使能
bcf         TXSTA,4,0           ;异步模式时钟,UART 模异步模式
movlw       ((OSC*.10)/(BOUND*.64)+5)/.10-1
movwf       SPBRG,0


UART 数据发送：
movlw       0x88
movwf       TXREG,0             ;发送 0x88 数据
btfss       PIR2,4,0
bra         $-1                 ;跳转到上一条指令


UART 数据接收：
btfss       PIR2,5,0
```

```
    bra         $-1
    movf        RCREG,0,0
    movwf       reg_tmp,1           ;接收数据放入 reg_tmp,里
```

◆ 3.10 读E2PROM程序

3.10.1 读 E2PROM 示例：

```
    movlw       0x01
    movwf       FSR0H,0
    movlw       0x02
    movwf       FSR0L,0             ;SRAM 地址
    movlw       0x00
    movwf       multiplier_L,1      ;E2PROM 地址
    movlw       0x0f
    movwf       multiplicand_L,1    ;要写的字数

    call        Read_more_24c02
```

作用：从首地址为 0x00 的 E2PROM 里读取 15 个字节的数据，连续地放入首地址为 0x0102 开始的 15 个数据寄存器里；

3.10.2 读 E2PROM 子函数：

```
Read_more_24c02:
    bsf         SSPCON2,0,0         ;SEN 启动启动条件
    btfsc       SSPCON2,0,0         ;SEN 等待启动条件硬件清零
    bra         $-1
read_rep_read:
    movlw       0xa0
    movwf       SSPBUF,0            ;器件地址（写）
    btfsc       SSPSTAT,2,0         ;判断是否发送完
    bra         $-1
;收应答
    btfsc       SSPCON2,6,0         ;判断是否收到从器件的应答
    bra         repet_read
    movf        multiplier_L,0,1
    movwf       SSPBUF,0            ;要读的字地址
    btfsc       SSPSTAT,2,0         ;判断是否发送完
    bra         $-1
;收应答
    btfsc       SSPCON2,6,0         ;ACKSTAT 判断是否收到从动器件的应答
    bra         repet_read
;启动重复开始条件
    bsf         SSPCON2,1,0         ;RSEN 启动启动条件
    btfsc       SSPCON2,1,0         ;RSEN 等待重复启动条件硬件清零
    bra         $-1
    movlw       0xa1
```

```
    movwf       SSPBUF,0            ;发送器件地址（读）
    btfsc       SSPSTAT,2,0         ;判断是否发送完
    bra         $-1
;收应答
    btfsc       SSPCON2,6,0         ;ACKSTAT 判断是否收到从动器件的应答
    bra         repet_read
;读 e2prom 数据
read_more_byte:
    bsf         SSPCON2,3,0         ;RCEN 使能接收
    btfsc       SSPCON2,3,0         ;RCEN 判断是否接收完
    bra         $-1
    movf        SSPBUF,0,0
    movwf       INDF0,0
    incf        FSR0L,1,0
    dcfsnz      multiplicand_L,1,1
    bra         N_ACK
    bcf         SSPCON2,5,0         ;0 应答
    bsf         SSPCON2,4,0         ;应答，并启动应答
    btfsc       SSPCON2,4,0         ;ACKEN 判断是否应答空闲
    bra         $-1
    bra         read_more_byte
;启动重复开始条件
repet_read:
    bsf         SSPCON2,1,0         ;RSEN 启动启动条件
    btfsc       SSPCON2,1,0         ;RSEN 等待重复启动条件硬件清零
    bra         $-1
    bra         read_rep_read
N_ACK:
    bsf         SSPCON2,5,0         ;1 不应答
    bsf         SSPCON2,4,0         ;应答，并启动应答
    btfsc       SSPCON2,4,0         ;ACKEN 判断是否应答空闲
    bra         $-1
    bsf         SSPCON2,2,0         ;PEN 启动停止条件
    btfsc       SSPCON2,2,0         ;PEN 判断停止是否空闲
    bra         $-1
    return
```

◆ 3.11 写 E2PROM 程序

3.11.1 写 E2PROM 示例：

```
    movlw       0x01
    movwf       FSR0H,0
    movlw       0x02
    movwf       FSR0L,0             ;SRAM 地址
    movlw       0x00
```

```
    movwf       multiplier_L,1      ;E2PROM 地址
    movlw       0x08
    movwf       multiplicand_L,1    ;要写的字数

    call        Write_Page_24C02
```

作用：从首地址为 0x0102 的数据寄存器里读取 8 个字节的数据，连续写入首地址为 0x00 的 E2PROM 的 8 个单元里；

3.11.2 写 E2PROM 子函数：

```
Write_page_24c02:
    bsf         SSPCON2,0,0             ;启动条件使能，硬件自动清零
    btfsc       SSPCON2,0,0             ;等待启动条件空闲
    bra         $-1
send_w_dev:
;发送器件地址
    movlw       0xa0
    movwf       SSPBUF,0                ;24C02 的器件地址（写）
    btfsc       SSPSTAT,2,0             ;判断是否发送完
    bra         $-1
;收应答
    btfsc       SSPCON2,6,0             ;判断是否收到从器件的应答
    bra         repet_w_start
;发送字地址
    movf        multiplier_L,0,1        ;字地址
    movwf       SSPBUF,0
    btfsc       SSPSTAT,2,0             ;判断是否发送完
    bra         $-1
;收应答
    btfsc       SSPCON2,6,0             ;判断是否收到从器件的应答
    bra         repet_w_start
send_more_byte:
;发送数据
    movf        INDF0,0,0               ;要写入的数据
    movwf       SSPBUF,0
    btfsc       SSPSTAT,2,0             ;判断是否发送完
    bra         $-1
;收应答
    btfsc       SSPCON2,6,0             ;判断是否收到从器件的应答
    bra         write_err
    incf        FSR0L,1,0
    decfsz      multiplicand_L,1,1
    bra         send_more_byte
;启动停止条件
    bsf         SSPCON2,2,0             ;启动停止条件，硬件自动清零
```

```
    btfsc       SSPCON2,2,0             ;等待停止条件空闲
    bra         $-1
    return
;启动重复开始条件
repet_w_start:
    bsf         SSPCON2,1,0             ;RSEN 启动启动条件
    btfsc       SSPCON2,1,0             ;RSEN 等待重复启动条件硬件清零
    bra         $-1
    bra         send_w_dev
;写 e2porm 出错
write_err:

    ;置错误标志位
    ;LCD 上指示 err

    return
```

◆ 3.12 3(字节) * 2(字节) = 4(字节)乘法程序

3.12.1 乘法示例：

```
    movlw       0x00                    ;被乘数赋值
    movwf       multiplicand_U,1        ;10000 的十六进制表示
    movlw       0x27
    movwf       multiplicand_H,1
    movlw       0x10
    movwf       multiplicand_L,1

    movlw       0x03                    ;乘数赋值
    movwf       multiplier_H,1          ;1000 的十六进制表示
    movlw       0xe8
    movwf       multiplier_L,1

    Call        Muti_Process            ;调用乘法子函数
```

例如，10000（被乘数）*1000（乘数）=10000000（积）,结果的十六进制为 0x00 0x98 0x96 0x80 分别存放在 product_HH, product_HL, product_LH, product_LL 里。需要注意的是积只有 4 个字节，不能溢出。

3.12.2 乘法子函数：

```
Muti_process:
    movlw       0x10
    movwf       divdend_Reg_LH          ;移位次数 16

    movlw       0x01
    movwf       divdend_Reg_HH,1        ;移到第几位标志位
```

```
    clrf        product_HH,1
    clrf        product_HL,1
    clrf        product_LH,1
    clrf        product_LL,1            ;清零积值

loop_multi:
    bcf         STATUS,C,0
    rrcf        multiplier_H,1,1
    rrcf        multiplier_L,1,1

    btfss       STATUS,C,0
    bra         judge_loop_end

    movf        divdend_Reg_HH,0,1
    movwf       divdend_Reg_HL,1
    movf        multiplicand_L,0,1
    movwf       multiplicand_reg_LL,1
    movf        multiplicand_H,0,1
    movwf       multiplicand_reg_LH,1
    movf        multiplicand_U,0,1
    movwf       multiplicand_reg_HL,1
    clrf        multiplicand_reg_HH,1

loop_rlf:
    dcfsnz      divdend_Reg_HL,1,1
    bra         add_multi               ;完成移位计算
    bcf         STATUS,C,0
    rlcf        multiplicand_reg_LL,1,1
    rlcf        multiplicand_reg_LH,1,1
    rlcf        multiplicand_reg_HL,1,1
    rlcf        multiplicand_reg_HH,1,1
    bra         loop_rlf

add_multi:
    movf        multiplicand_reg_LL,0,1
    addwf       product_LL,1,1
    movf        multiplicand_reg_LH,0,1
    addwfc      product_LH,1,1
    movf        multiplicand_reg_HL,0,1
    addwfc      product_HL,1,1
    movf        multiplicand_reg_HH,0,1
    addwfc      product_HH,1,1

judge_loop_end:
    incf        divdend_Reg_HH,1,1
```

```
decfsz      divdend_Reg_LH,1,1      ;判断是否循环完
bra         loop_multi
nop
return
```

◆ 3.13 4(字节) / 4(字节)= 4(字节)除法程序

3.13.1 除法示例

```
movlw       0x00
movwf       divdend_HH,1
movlw       0x0f
movwf       divdend_HL,1
movlw       0x42
movwf       divdend_LH,1
movlw       0x40                    ;1000 000 的十六进制
movwf       divdend_LL,1            ;被除数赋值

movlw       0x00
movwf       divsor_HH,1
movlw       0x00
movwf       divsor_HL,1
movlw       0x03
movwf       divsor_LH,1
movlw       0xe8                    ;1000 的十六进制
movwf       divsor_LL,1             ;除数赋值

Call        Div_process
```

例如，1000 000（被除数）/ 1000（除数）= 1000(商)，商值十六进制为 0x00 0x00 0x03 0xe8 分别存放在 quotient_HH、quotient_HL、quotient_LH、quotient_LL 里,余数为 0x00 0x00 0x00 0x00 分别存放在 remainder_HH、remainder_HL、remainder_LH、remainder_LL 里。

3.13.2 除法子函数

```
Div_process:
    movlw       0x20
    movwf       multiplicand_reg_LH,1   ;循环 32 次(4 个字节的除法)
    clrf        divdend_Reg_LL,1        ;被除缓存数清零
    clrf        divdend_Reg_LH,1
    clrf        divdend_Reg_HL,1
    clrf        divdend_Reg_HH,1
    clrf        quotient_LL,1           ;商清零
    clrf        quotient_LH,1
    clrf        quotient_HL,1
    clrf        quotient_HH,1
```

```
    movlw       0x00                    ;如被除数为 0，直接赋值为 0
    cpfseq      divdend_HH,1
    bra         loop_div
    cpfseq      divdend_HL,1
    bra         loop_div
    cpfseq      divdend_LH,1
    bra         loop_div
    cpfseq      divdend_LL,1
    bra         loop_div

    bra         Sezo_Div

loop_div:
    bcf         STATUS,C,0
    rlcf        divdend_LL,1,1          ;将被除数低位左移到 divdend 高位
    rlcf        divdend_LH,1,1
    rlcf        divdend_HL,1,1
    rlcf        divdend_HH,1,1

    rlcf        divdend_Reg_LL,1,1
    rlcf        divdend_Reg_LH,1,1
    rlcf        divdend_Reg_HL,1,1
    rlcf        divdend_Reg_HH,1,1

    movf        divsor_LL,0,1           ;比较 divdend_Reg 和 divsor 的大小
    subwf       divdend_Reg_LL,0,1
    movf        divsor_LH,0,1
    subwfb      divdend_Reg_LH,0,1
    movf        divsor_HL,0,1
    subwfb      divdend_Reg_HL,0,1
    movf        divsor_HH,0,1
    subwfb      divdend_Reg_HH,0,1

    btfss       STATUS,C,0              ;如果被除数小于除数(相减为负,C 为 0)
    bra         Lower

Greater:
    movf        divsor_LL,0,1
    subwf       divdend_Reg_LL,1,1      ;被除数-除数
    movf        divsor_LH,0,1
    subwfb      divdend_Reg_LH,1,1
    movf        divsor_HL,0,1
    subwfb      divdend_Reg_HL,1,1
    movf        divsor_HH,0,1
```

```
    subwfb      divdend_Reg_HH,1,1
    bsf         STATUS,C,0              ;置进位位为 1
    bra         JudgeLoop

Lower:
    bcf         STATUS,C,0              ;置进位位为 0

JudgeLoop:
    rlcf        quotient_LL,1,1         ;将商带进位左移 1 位
    rlcf        quotient_LH,1,1
    rlcf        quotient_HL,1,1
    rlcf        quotient_HH,1,1
    decfsz      multiplicand_reg_LH,1,1 ;判断是否循环完
    bra         loop_div

Sezo_Div:
    movf        divdend_Reg_HH,0,1      ;赋值余数
    movwf       remainder_HH,1
    movf        divdend_Reg_HL,0,1
    movwf       remainder_HL,1
    movf        divdend_Reg_LH,0,1
    movwf       remainder_LH,1
    movf        divdend_Reg_LL,0,1
    movwf       remainder_LL,1
    return

Divd_Dat:

    movf        product_HH,0,1
    movwf       divdend_HH,1
    movf        product_HL,0,1
    movwf       divdend_HL,1
    movf        product_LH,0,1
    movwf       divdend_LH,1
    movf        product_LL,0,1
    movwf       divdend_LL,1
    return
```

◆ 3.14 [4(字节)]$^{1/2}$ = 2(字节) 开方程序

3.14.1 开方示例：

```
    movlw       0x00
    movwf       Sqrt_Prim_HH,1
    movlw       0x0f
    movwf       Sqrt_Prim_HL,1
    movlw       0x45
```

```
    movwf       Sqrt_Prim_LH,1
    movlw       0x68
    movwf       Sqrt_Prim_LL            ;开方数赋值：十六进制为 0xF4568

    Call        Sqrt_process
```

作用：对十六进制数 0xF4568 进行开方，开方后的结果（0x03 0xe8）分别存放在 Sqrt_Result_H、Sqrt_Result_L 里。

3.14.2 开方子程序：

```
sqrt_process:
    clrf        Sqrt_Result_H,1         ;清零开方结果
    clrf        Sqrt_Result_L,1

    ;判断被开方数是否为 0
    movlw       0x00
    cpfseq      Sqrt_Prim_HH,1          ;被开方数
    bra         Check_Start_Sqrt
    cpfseq      Sqrt_Prim_HL,1
    bra         Check_Start_Sqrt
    cpfseq      Sqrt_Prim_LH,1
    bra         Check_Start_Sqrt
    cpfseq      Sqrt_Prim_LL,1
    bra         Check_Start_Sqrt
    bra         Ret_SqrtFunc

;判断第 1 节
Check_Start_Sqrt:
    movlw       0x02
    movwf       temp1,1                 ;开方计数值

    clrf        Sqrt_Remain_LL,1        ;清零余数
    clrf        Sqrt_Remain_LH,1
    clrf        Sqrt_Remain_HL,1
    clrf        Sqrt_Remain_HH,1

Check_One_Sqrt:
    rlcf        Sqrt_Prim_LL,1,1        ;将被开方数高两位移入余数
    rlcf        Sqrt_Prim_LH,1,1
    rlcf        Sqrt_Prim_HL,1,1
    rlcf        Sqrt_Prim_HH,1,1
    rlcf        Sqrt_Remain_LL,1,1

    decfsz      temp1,1,1
    bra         Check_One_Sqrt
```

```
        movlw       0x01
        subwf       Sqrt_Remain_LL,1,1      ;判断第 1 节是否大于等于 1

        bsf         Sqrt_Result_L,0,1       ;大于等于 1 则商置 1
        btfsc       STATUS,C,0
        bra         Check_Next_Sqrt

        clrf        Sqrt_Remain_LL,1        ;为 0 则商置 0
        bcf         Sqrt_Result_L,0,1

    ;判断余下 15 节
    Check_Next_Sqrt:
        movlw       0x0f
        movwf       temp1,1

   Check_15_Sqrt:
        bcf         STATUS,C,0
        rlcf        Sqrt_Result_L,1,1       ;商左移 1 位
        rlcf        Sqrt_Result_H,1,1

        rlcf        Sqrt_Prim_LL,1,1        ;将被开方数高两位移入余数
        rlcf        Sqrt_Prim_LH,1,1
        rlcf        Sqrt_Prim_HL,1,1
        rlcf        Sqrt_Prim_HH,1,1

        rlcf        Sqrt_Remain_LL,1,1
        rlcf        Sqrt_Remain_LH,1,1
        rlcf        Sqrt_Remain_HL,1,1
        rlcf        Sqrt_Remain_HH,1,1

        rlcf        Sqrt_Prim_LL,1,1
        rlcf        Sqrt_Prim_LH,1,1
        rlcf        Sqrt_Prim_HL,1,1
        rlcf        Sqrt_Prim_HH,1,1

        rlcf        Sqrt_Remain_LL,1,1      ;中间暂存变量
        rlcf        Sqrt_Remain_LH,1,1      ;中间暂存变量
        rlcf        Sqrt_Remain_HL,1,1      ;中间暂存变量
        rlcf        Sqrt_Remain_HH,1,1      ;中间暂存变量

        rlcf        Sqrt_Result_L,0,1       ;将商左移 1 位放入试减数中并加 1
        movwf       temp3,1
        rlcf        Sqrt_Result_H,0,1
        movwf       temp2,1
```

```
        movlw       0x01
        addwf       temp3,1,1
        movlw       0x00
        addwfc      temp2,1,1

        movf        temp3,0,1
        subwf       Sqrt_Remain_LL,1,1
        movf        temp2,0,1
        subwfb      Sqrt_Remain_LH,1,1
        movlw       0x00
        subwfb      Sqrt_Remain_HL,1,1
        subwfb      Sqrt_Remain_HH,1,1

        bsf         Sqrt_Result_L,0,1       ;商上 1
        btfsc       STATUS,C,0
        bra         Check_15_Sqrt_Next

        bcf         Sqrt_Result_L,0,1       ;商上 0 并加回来
        movf        temp3,0,1
        addwf       Sqrt_Remain_LL,1,1
        movf        temp2,0,1
        addwfc      Sqrt_Remain_LH,1,1
        movlw       0x00
        addwfc      Sqrt_Remain_HL,1,1
        addwfc      Sqrt_Remain_HH,1,1

    Check_15_Sqrt_Next:
        decfsz      temp1,1,1
        bra         Check_15_Sqrt

    Ret_SqrtFunc:
        return
```

◆ 3.15 LCD 十六进制转十进制显示程序

3.15.1 显示示例：

```
        movlw       0x00
        movwf       Disp_DataHL,1
        movlw       0x04
        movwf       Disp_DataLH,1
        movlw       0xd2
        movwf       Disp_DataLL,1           ;显示 1234

        call        LCD_HEX2DEC_DISP
```

作用：把 DataHL/LH/LL 里的十六进制数（0x00 04 D2）转换为十进制数(1234)并显示在 LCD 上。

3.15.2 显示子程序

```
LCD_HEX2DEC_DISP：
    ;显示第 1 位
    clrf        divdend_HH,1
    movf        Disp_DataHL,0,1
    movwf       divdend_HL,1
    movf        Disp_DataLH,0,1
    movwf       divdend_LH,1
    movf        Disp_DataLL,0,1
    movwf       divdend_LL,1

    clrf        divsor_HH,1
    clrf        divsor_HL,1
    movlw       0x03
    movwf       divsor_LH,1
    movlw       0xe8
    movwf       divsor_LL,1

    rcall       Div_process

    movf        quotient_LL,0,1
    andlw       0x0f
    movwf       dis_reg,1

    rcall       judge_disreg
    movf        TABLAT,0,0              ;显示 lcd 第 1 位
    movwf       LCDDATA6,1,0            ;寄存器视 LCD 连接引脚而定

    ;显示第 2 位
    clrf        divdend_HH,1
    movf        remainder_HL,0,1
    movwf       divdend_HL,1
    movf        remainder_LH,0,1
    movwf       divdend_LH,1
    movf        remainder_LL,0,1
    movwf       divdend_LL,1

    clrf        divsor_HH,1
    clrf        divsor_HL,1
    movlw       0x00
    movwf       divsor_LH,1
    movlw       0x64
```

```
movwf       divsor_LL,1

rcall       Div_process

movf        quotient_LL,0,1
andlw       0x0f
movwf       dis_reg,1

rcall       judge_disreg
movf        TABLAT,0,0              ;显示 lcd 第 2 位
movwf       LCDDATA2,1,0            ;寄存器视 LCD 连接引脚而定

;显示第 3 位
clrf        divdend_HH,1
movf        remainder_HL,0,1
movwf       divdend_HL,1
movf        remainder_LH,0,1
movwf       divdend_LH,1
movf        remainder_LL,0,1
movwf       divdend_LL,1

clrf        divsor_HH,1
clrf        divsor_HL,1
movlw       0x00
movwf       divsor_LH,1
movlw       0x0a
movwf       divsor_LL,1

rcall       Div_process

movf        quotient_LL,0,1
andlw       0x0f
movwf       dis_reg,1

rcall       judge_disreg
movf        TABLAT,0,0              ;显示 lcd 第 3 位
movwf       LCDDATA1,1,0            ;寄存器视 LCD 连接引脚而定

;显示第 4 位
movf        remainder_LL,0,1
andlw       0x0f
movwf       dis_reg,1

rcall       judge_disreg
movf        TABLAT,0,0              ;显示 lcd 第 4 位
```

```
    movwf       LCDDATA0,1,0            ;寄存器视 LCD 连接引脚而定

    return
```

◆ 3.16 LCD 代码判断程序

3.16.1 示例：

配合 lcd 十六进制转换十进制使用。

3.16.2 子程序：

```
judge_disreg:
    movlw       low(table)              ;根据 dis_reg 值判断 LCD 代码
    movwf       TBLPTRL,0
    movlw       high(table)
    movwf       TBLPTRH,0

    movf        dis_reg,0,1
    addwf       TBLPTRL,1,0
    movlw       0x00
    addwfc      TBLPTRH,1,0

    tblrd *
    nop
    return

;LCD 显示码表，视具体 LCD 型号和管脚连接而定
table:
    db          0xd7,0x06       ;0,1
    db          0xe3,0xa7       ;2,3
    db          0x36,0xb5       ;4,5
    db          0xf5,0x07       ;6,7
    db          0xf7,0xb7       ;8,9
    db          0x77,0xf4       ;a,b
    db          0xd1,0xe6       ;c,d
    db          0xf1,0x71       ;e,f
```

修改记录：

版本号	修改日期	修改人	修改记录
V0.1	2013-11-20	贵焱锋	删除定时器 0/1 初始化的一些指令