目录

目录.....1

1 赛元 SC92F 系列 MCU 电气参数注意项.....3

2 赛元 SC92F 系列 MCU 烧写注意事项.....3

3 赛元 SC92F 系列 MCU 有关 MOVC 指令的应用注意.....4

3.1 C 语言编程有关 MOVC 指令的应用注意.....4
3.1.1 C 语言开发，MOVC 指令.....4
3.1.2 C 语言开发具体操作.....4

3.2 汇编语言编程有关 MOVC 指令的应用注意.....9

4 赛元 SC92F 系列 MCU 的 EEPROM 及算法解说.....10

4.1 内部 EEPROM 的操作及 CODE 的 IAP 操作.....10

4.2 EEPROM 操作及 CODE 的 IAP 操作代码.....11

4.3 EEPROM 的使用算法.....12

5 电路设计的注意事项.....18

5.1 电路设计实例.....18
5.1.1 LED 的使用以及接法.....18
5.1.2 1 位共阴极数码管的使用.....18
5.1.3 RST 管脚电路.....19
5.1.4 ADC 采样管脚电路.....20
5.1.5 外部晶振电路.....21
5.1.6 TouchKey 电路.....21

5.2 IO 口各模式设置注意事项.....21
5.2.1 I/O 设为高阻，实现电路设计.....21
5.2.2 带上拉输入模式.....21
5.2.3 带上拉输入模式检测按键.....22
5.2.4 I/O 开漏输出模式的实现方式.....22

6 软件编写的注意事项.....23

6.1 PWM 设置及使用注意事项.....23

6.2 PCON 寄存器设置注意事项.....23

6.3 UART0 设置及使用注意事项.....23

6.4 SSI 设置及使用注意事项.....24
6.5 ADC 多通道切换采集注意事项.....24
6.6 使用定时器时外部中断服务函数编写注意事项.....25
6.7 外部中断设置注意事项.....25
6.8 LCD/LED 显示 RAM 使用注意事项.....25
6.9 软件操作 Code Option 的注意事项.....26
6.10 TouchKey 设置注意事项.....26
7 版本更改记录.....27

1 赛元 SC92F 系列 MCU 电气参数注意项

工作电压：2.4V~5.5V

工作温度：-40 ~ 85℃

内核：高速 1T 8051

Flash ROM：（MOVC 禁止寻址 0000H~00FFH）可重复写入 1 万次

EEPROM： 独立的 128Byte，可重复写入 10 万次，10 年以上保存寿命

系统时钟：内建高频振荡器频率误差： 跨越 (4.0V~5.5V) 及 (-20 ~ 85℃) 应用环境，不超过 ±1%

注意：LVR 设置电压必须低于电源电压，防止芯片一直复位！

2 赛元 SC92F 系列 MCU 烧写注意事项

1. 赛元 SC92F 系列 芯片的 CLK 或 DIO 管脚对 GND 不得超过 100pF 的电容，VDD 对 GND 的电容不可超过 1000uF 的电容。
2. 烧录引出点与芯片之前尽量不要串电阻，如无法避免，应保证串接电阻的阻值不超过 100R，且烧录时要尽量缩短烧录排线。
3. 电路设计时应避免将芯片的 CLK 和 DIO 连到同一个数码管上。
4. SC-LINK 的烧录排线最长不可超过 60cm。

3 赛元 SC92F 系列 MCU 有关 MOVC 指令的应用注意

赛元 MCU Flash ROM 的起始 256B ROM 区间，即 0x0000-0x00FF，禁止 MOVC 寻址。因此说，用户自定义的数据不能存放在该区域。譬如说，在 C 语言编程当中，初始化的全局变量，不可变类型数据（code 类型数据），不能存放在该地址区域。

以下主要是针对这个特性，说明在编程当中有关 MOVC 指令的应用注意事项。

3.1 C 语言编程有关 MOVC 指令的应用注意

3.1.1 C 语言开发，MOVC 指令

C 语言开发中，通常有 3 种情况使用到 MOVC 指令，即是对 Flash ROM 进行访问。

a) 全局变量的初始化
b) 不可变类型数量（code 类型数据）
c) 函数调用库文件的查表运算

C 语言编译完成后，用户可打开工程中的.M51 文件查看 Code Memory 部分，通过查看 Code 标识符，就可以确认自己是否有以上 3 种情况的操作。参见下表：

标识符	说明	备注
?C_INITSEG	全局变量初始化	进入 MAIN 之前会调用
?CO?Project_name	放到 Code 区的常量或指针	“Project_name”工程名称
?C?LIB_CODE	库文件	Math 函数或者浮点运算会用到

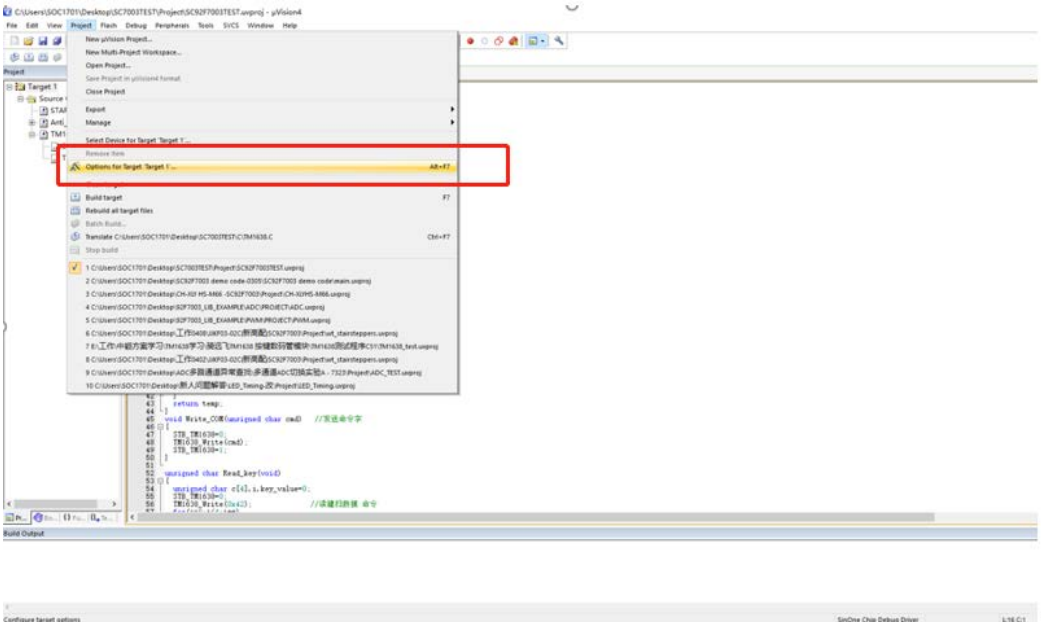
注意：?C?LIB_CODE 标识符只是表明某个函数调用的库文件进行查表运算，通常情况下，客户开发产品不需用调用库文件 Math 函数。（库文件占用较大的 ROM 空间,譬如 sinx 函数）。

.M51 文件详细记录了上表中各代码段的使用情况，包括 Code 的起始地址、长度等。用户只需要查看?C_INITSEG、?CO?Project_name、调用库文件的函数（如果有?C?LIB_CODE 的话）的起始地址是否在禁止访问区，如果在禁止访问区，可参考后续操作改变起始地址。

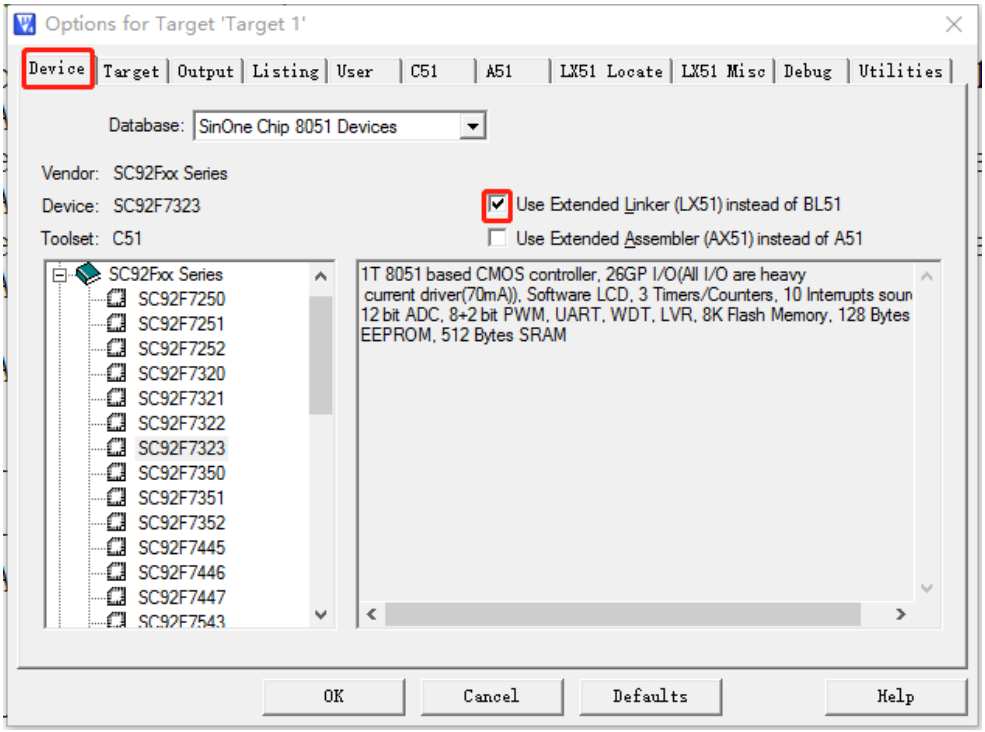
3.1.2 C 语言开发具体操作

根据 3.1.1 的描述可知，用户在采用 C 语言开发过程当中，需要把全局变量，不可变类型数据（code 类型数据）定义在 Flash ROM 起始的 256B 地址之后。因此，在开发调试时，可以先采用屏蔽该区域的 Flash Rom 来进行开发，待调试完毕后，再做调整，生成最终的程序。使用不同的链接器操作方法不同，具体方法见下：

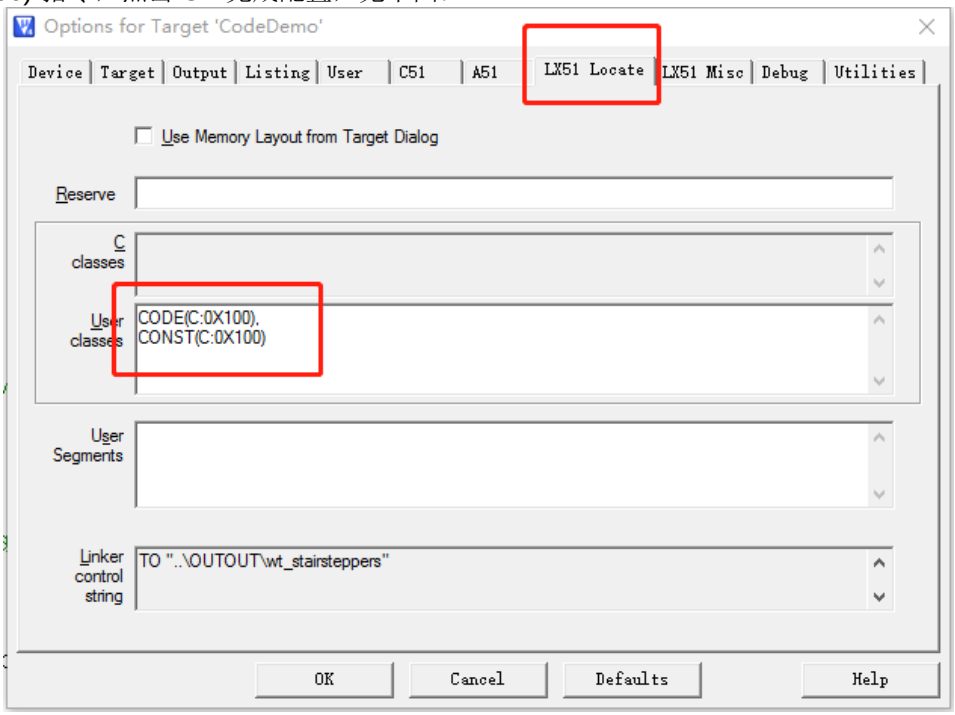
在 LX51 链接器上实现调整的方法（推荐方法）：



- **选择使用 LX51 链接器。**
  打开项目选项中的“Device” 属性页，勾选“Use Extended Linker(LX51)instead of BL51”见下图：

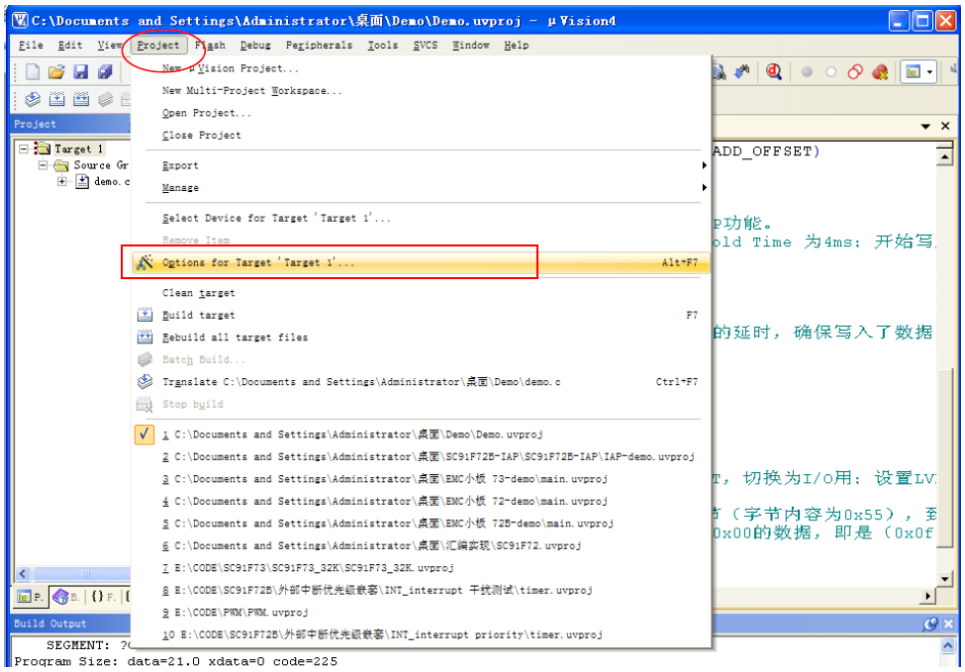


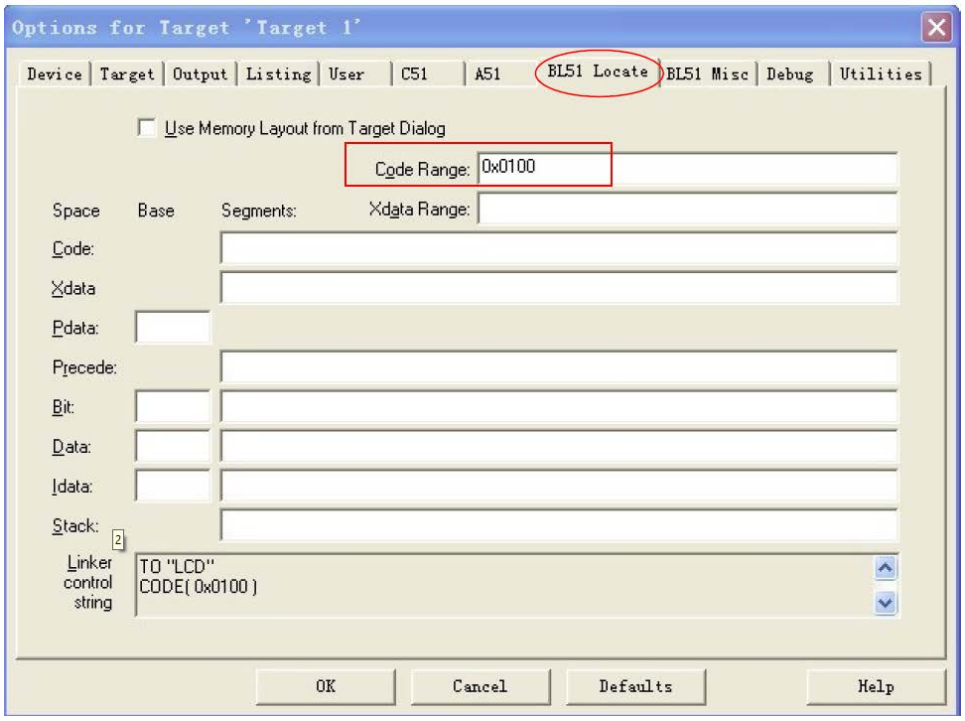
- **在 LX51 设置项中添加代码规定存储范围。**
  打开项目选项中的“LX51 Locate”属性页，在“User classes”的输入框中填入“CODE(C:0X100), CONST(C:0X100)”指令，点击 OK 完成配置，见下图：



在 BL51 链接器上实现调整的方法：

- **设置代码存放区域，便于调试。将代码区设置在 0x0100 之后。**
  打开项目选项中的“BL51Locate”属性页，在 Code Range 处输入“0x0100”保存，重新编译，进行调试等。见下图：



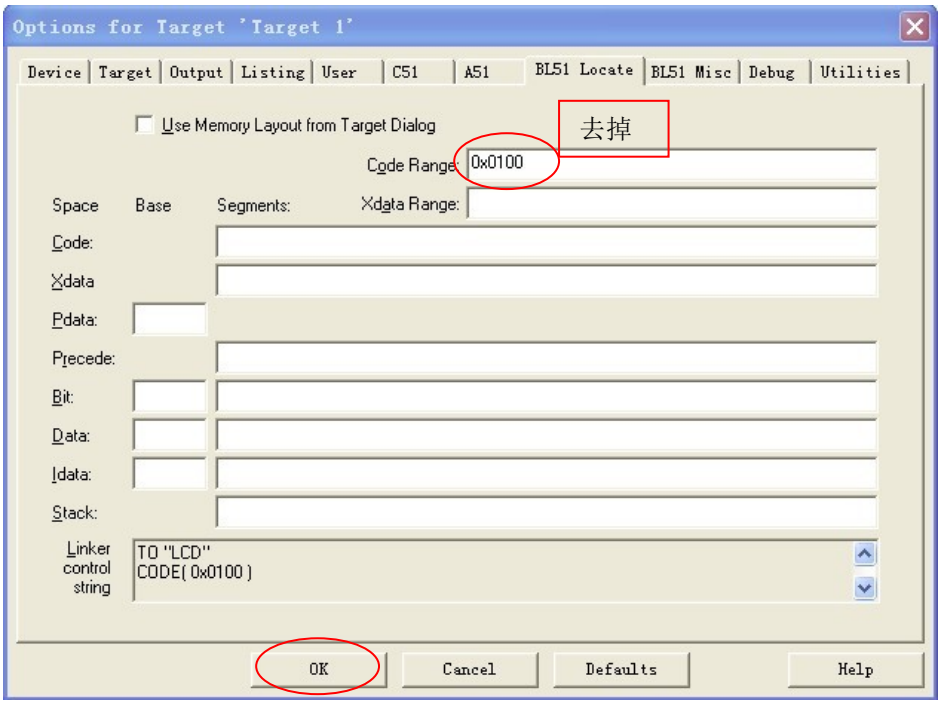


- **调试完成后，生成最终程序；**

当用户需要在 0X100 前存储代码时，由于 0x100 前的数据不可进行读操作，所以不可在此区域存储全局变量、不可变类型数量（code 类型数据），若编程代码中存在 code 类型的全局变量，需要将这些数据类型存放到 0x100 地址之后。

设置方法如下：

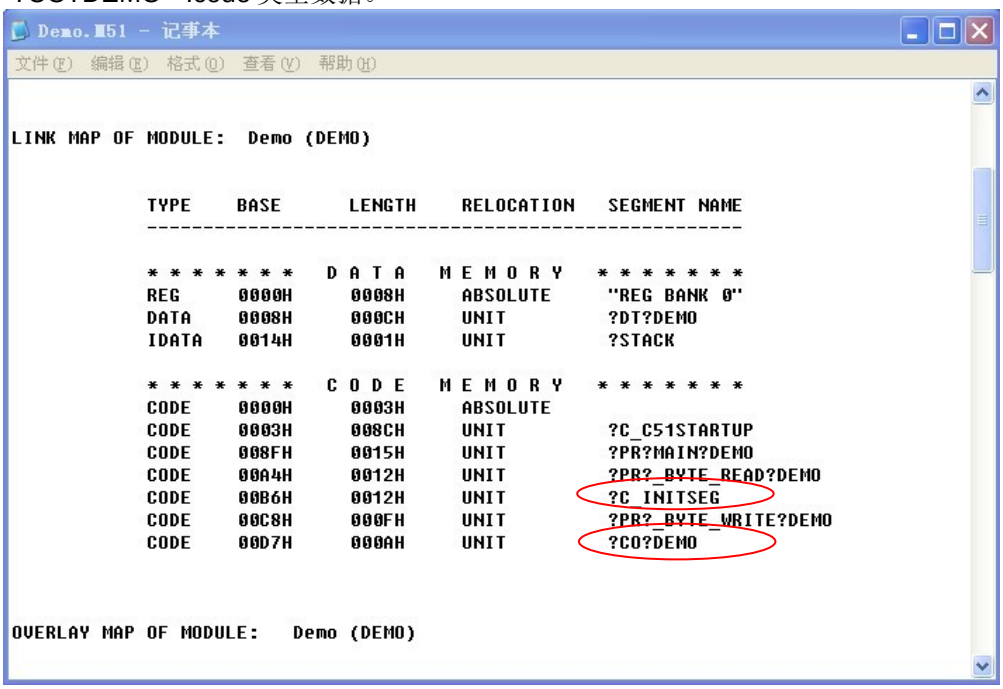
1）将代码存放区恢复回全区域，即取消第一步的操作。即将 Code Range 的数据去掉，点击 OK 保存。



2）重新编译后，在建立的工程目录下，找到并打开 .M51 文件，在 CODE MEMORY 会出现：

“?C_INITSEG” :全局变量初始化数据。

“?CO?DEMO” :code 类型数据。



说明：从以上 M51 文件的“CODE MEMORY”信息中，可以看到“?C_INITSEG”，链接地址为 00B6H，长度为 0012H 字节；“?CO?DEMO”，链接地址为 00D7H，长度为 000AH 字节。

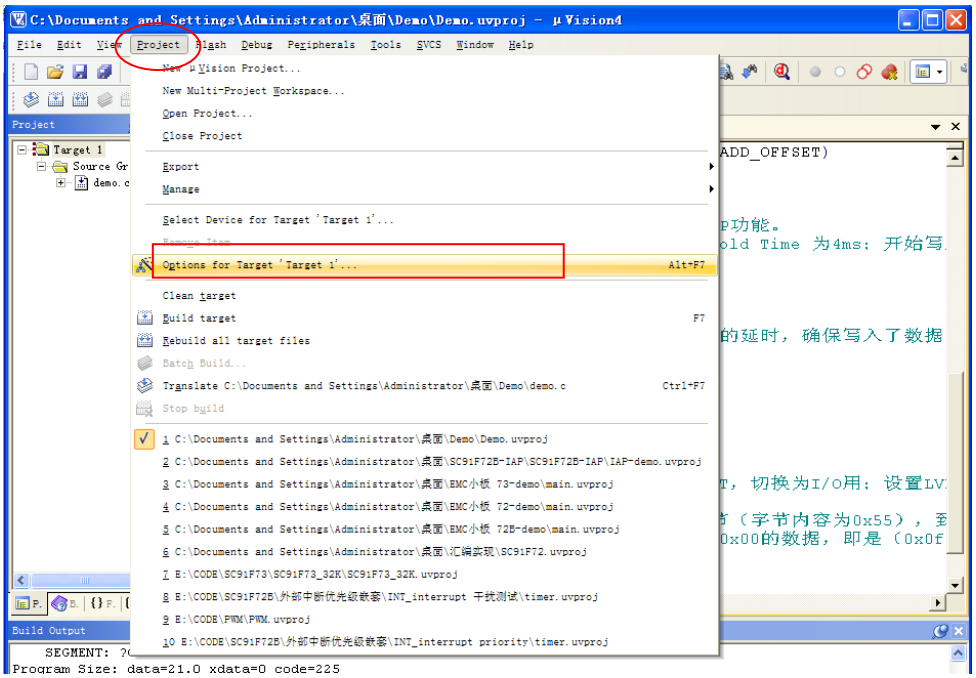
1）根据“?C_INITSEG”以及“?CO?DEMO”的长度信息计算出各自的重定位的地址：

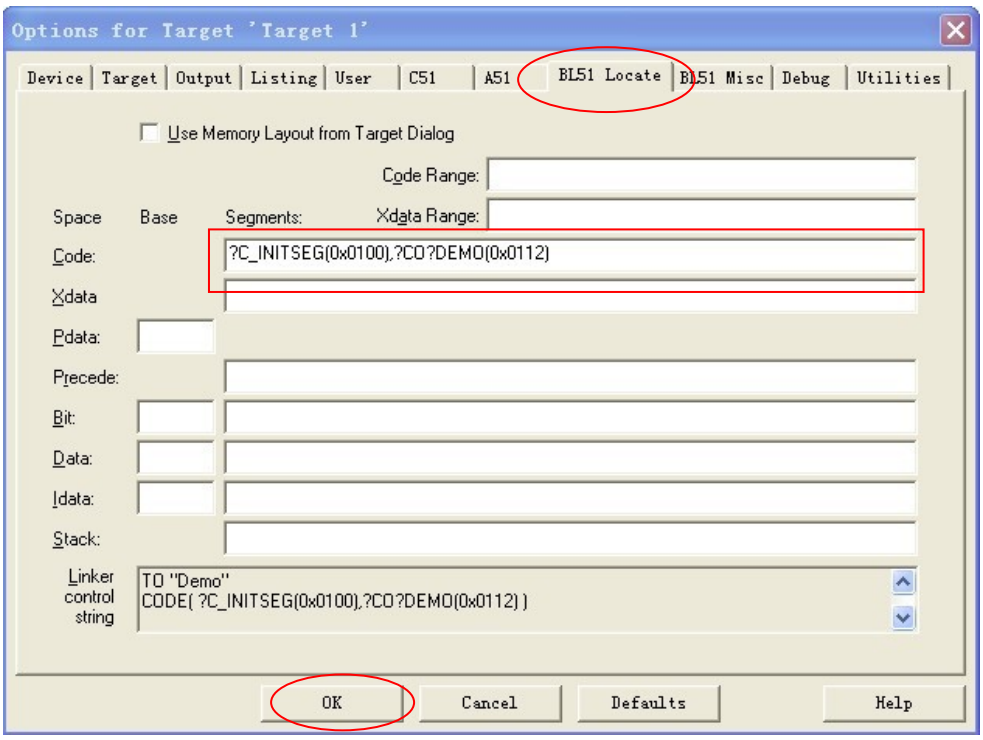
“?C_INITSEG” 的重定位地址为 0x0100

“?CO?DEMO” 的重定位地址为 0x0112

2）打开项目选项中的“BL51Locate 属性页，在“Code”域中输入下列语句：

“?C_INITSEG(0x0100),?CO?DEMO(0x0112)”





3）点击 OK 按钮，并重新编译即可生成了最终程序。

- **设置清 RAM 范围。**

赛元 SC92F 系列 MCU 内有 256B 的内部 RAM 和 256B 的外部 RAM（不同型号的外部 RAM 大小不同，具体请参照对应型号的规格书），在单片机复位后，如果需要对所有的 RAM 进行清 RAM 操作，则需要修改 STARTUP.A51 中对应的值，打开 STARTUP.A51，根据芯片 RAM 的各区域的大小进行相关设置，如图，是将 RAM 的 IDATA 区的 256Byte 和 XDATA 区的 256Byte 的数据清零。

```
;
;     Lx51 your object file list, STARTUP.OBJ  controls
;
;------------------------------------------------------------------------------
;
;  User-defined <h> Power-On Initialization of Memory
;
;  With the following EQU statements the initialization of memory
;  at processor reset can be defined:
;
; <o> IDATALEN: IDATA memory size <0x0-0x100>
;     <i> Note: The absolute start-address of IDATA memory is always 0
;     <i>       The IDATA space overlaps physically the DATA and BIT areas.
IDATALEN        EQU     100H
;
; <o> XDATASTART: XDATA memory start address <0x0-0xFFFF>
;     <i> The absolute start address of XDATA memory
XDATASTART      EQU     0
;
; <o> XDATALEN: XDATA memory size <0x0-0xFFFF>
;     <i> The length of XDATA memory in bytes.
XDATALEN        EQU     100H
;
; <o> PDATASTART: PDATA memory start address <0x0-0xFFFF>
;     <i> The absolute start address of PDATA memory
PDATASTART      EQU     0H
;
; <o> PDATALEN: PDATA memory size <0x0-0xFF>
;     <i> The length of PDATA memory in bytes.
PDATALEN        EQU     0H
;
;</h>
;------------------------------------------------------------------------------
```

3.2 汇编语言编程有关 MOVC 指令的应用注意

同理，在汇编编程的过程中，请注意将自定义的 ROM 区数据，定义在 0x0100 之后。操作方法比较简单，通过 ORG 来定位即可。

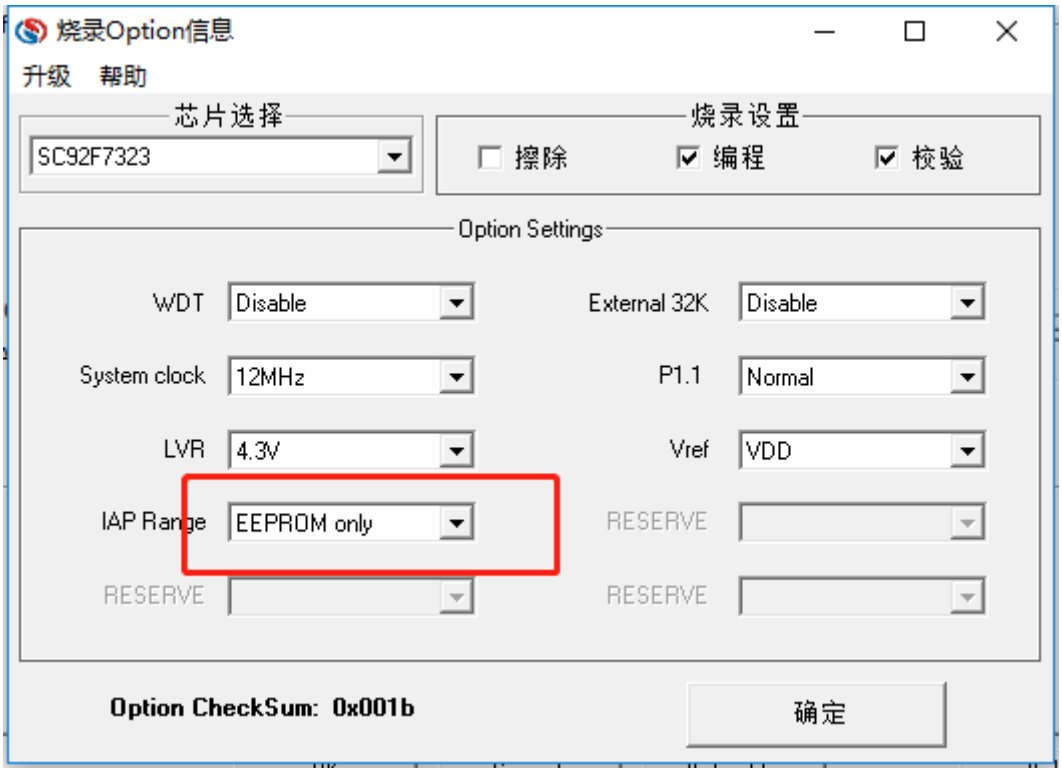
4 赛元 SC92F 系列 MCU 的 EEPROM 及算法解说

4.1 内部 EEPROM 的操作及 CODE 的 IAP 操作

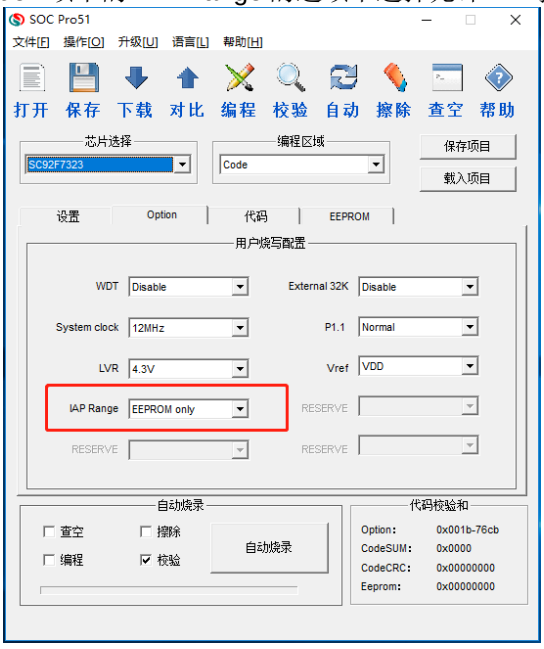
以 SC92F7323 为例，说明赛元 MCU 内部 EEPROM 的使用方法及 CODE 区 IAP 操作的方法。

SC92F7323 内部有 8K Flash 均可以进行 In Application Programming（IAP）操作，即允许用户程序动态的把数据写入内部的 Flash， 并且有独立的 128Byte 的 EEPROM。

用户使用 IAP 时, 需要在 Option 项设置允许 IAP 操作的范围，在 Keil 中设置方法如下，在 IAP Range 的选项中选择允许 IAP 操作的范围。



在上位机烧录时也是在 Option 项中的 IAP Range 的选项中选择允许 IAP 操作的范围。如图。



- **EEPROM 读写特点：**

a) By Byte 操作。即一个字节一个字节写入，读取。
b) 类 RAM 读写，写前不需擦除。

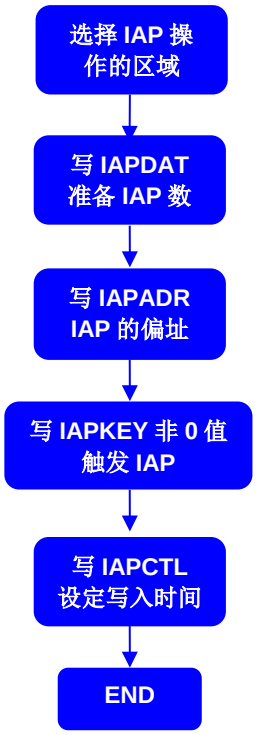
- **EEPROM 的寿命：**10W 次以上。
- **FLASH 读写特点：**

c) By Byte 操作。即一个字节一个字节写入，读取。
d) 类 RAM 读写，写前不需擦除。

- **FLASH 的寿命：**2W 次以上。

IAP 写入流程：

每写入一个字节，需要指定一个地址；具体 IAP 写入流程如下：



4.2 EEPROM 操作及 CODE 的 IAP 操作代码

SC92F 系列芯片在进行 IAP 操作的过程中不允许响应外部中断，因此，在进行相关操作时，需要先把中断总中断关闭，即 EA=0；带完成 IAP 操作后再恢复总中断开关。

使用独立的 EEPROM 进行 IAP 操作时，在操作完成后，务必使 IAPADE 指回 CODE 区，以免程序将跑飞。

独立 EEPROM 操作例程

```
#include ”intrins.h”
unsigned char EE_Add;
unsigned char EE_Data;
unsigned char code * POINT =0x0000;
```

EEPROM 写操作 C 的 Demo 程序：

```
EA = 0;                 //关总中断
IAPADE = 0X02;          //选择 EEPROM 区域
IAPDAT = EE_Data;       //送数据到 EEPROM 数据寄存器
IAPADH = 0x00;          //高地址默认写 0x00
IAPADL = EE_Add;        //写入 EEPROM 目标地址低位值
IAPKEY = 0XF0;          //此值可根据实际调整；需保证本条指令执行后到对 IAPCTL 赋值前，
                        //时间间隔需小于 240（0xf0）个系统时钟，否则 IAP 功能关闭;
```

```
                                    // 开启中断时要特别注意
    IAPCTL = 0X0A;                  //执行 EEPROM 写入操作，1ms@24M/12M/6M/2M;
    _nop_();                        //等待(至少需要 8 个_nop_())
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();

    IAPADE = 0X00;                  //返回 ROM 区域
    EA = 1;                         //开总中断
```

EEPROM 读操作 C 的 Demo 程序：

```
    EA = 0;                         //关总中断
    IAPADE =  0X02;                 //选择 EEPROM 区域
    EE_Data = *( POINT +EE_Add);//读取 IAP_Add 的值到 IAP_Data
    IAPADE =  0X00;                 //返回 ROM 区域, 防止 MOVC 操作到 EEPROM
    EA = 1;                         //开总中断
```

CODE 区域 IAP 操作例程

```
    #include ”intrins.h”
    unsigned int IAP_Add;
    unsigned char IAP_Data;
    unsigned char code * POINT =0x0000;
```

IAP 写操作 C 的 Demo 程序：

```
    IAPADE = 0X00;                          //选择 Code 区域
    IAPDAT = IAP_Data;                      //送数据到 IAP 数据寄存器
    IAPADH = (unsigned char)((IAP_Add >> 8));   //写入 IAP 目标地址高位值
    IAPADL = (unsigned char)IAP_Add;        //写入 IAP 目标地址低位值
    IAPKEY = 0XF0;                  //此值可根据实际调整；需保证本条指令执行后到对 IAPCTL 赋值前，
                                    //时间间隔需小于 240（0xf0）个系统时钟，否则 IAP 功能关闭;
                                    // 开启中断时要特别注意
    IAPCTL = 0X0A;                  //执行 IAP 写入操作，1ms@24M/12M/6M/2M;
    _nop_();                        //等待(至少需要 8 个_nop_())
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
```

IAP 读操作 C 的 Demo 程序：

```
    IAPADE = 0X00;                  //选择 Code 区域
    IAP_Data = *( POINT+IAP_Add);//读取 IAP_Add 的值到 IAP_Data
```

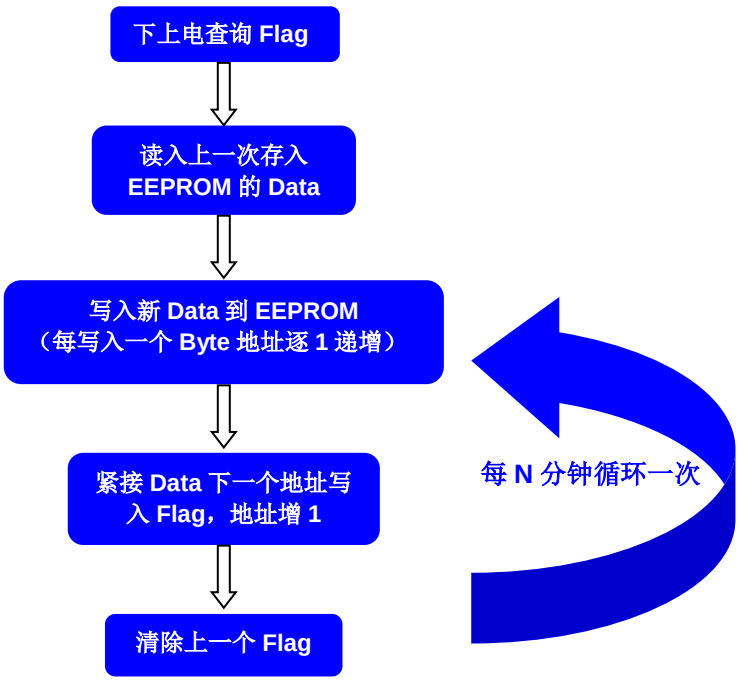
注意： Code 区域内的 IAP 操作有一定的风险，需要用户在软件中做相应的安全处理措施，**如果操作不当可能会造成用户程序被改写！**除非用户必需此功能(比如用于远程程序更新等)，否则不建议用户使用。

4.3 EEPROM 的使用算法

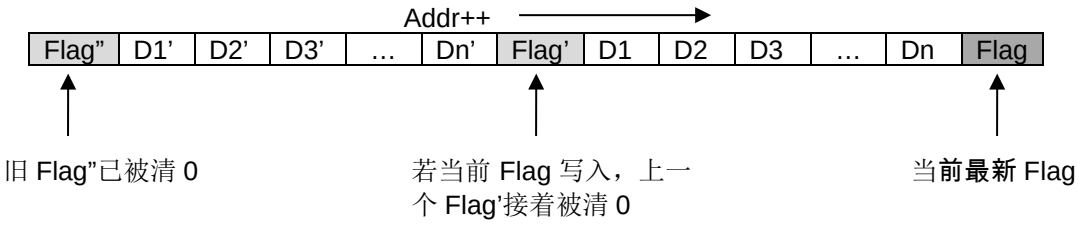
由于 SC92F7323 有 8kB Flash 及 128B 的 EEPROM 可以进行 In Application Programming（IAP）操作，而实际产品的应用中，譬如电压力锅，只是需要把仅几个 Bytes 的数据写到 EEPROM。为了充分利用 MCU 内部所

有的 EERPOM，预防过早达到写 IAP 寿命次数，有以下算法供参考：

a) 见以下流程图：



b) 采用以上算法，写入 EEPROM 的数据，见下：



- **以上算法特点：**
  ① 充分利用了 MCU 内部所有的 EEPROM；
  ② **算法较强健，存入 EEPROM 的 Data 不会因电源因素变化而被破坏；**
  ③ 算法效率较高，256B EEPROM 可以存放 256/(N+1)次数据。N 为要写入内部 EEPROM 的字节数目；
  ④ 若要写入内部 EEPROM 的字节数 N，若 N+1 不能被 256 整除，则 EEPROM 的寿命能发挥到极致；否则，EEPROM 内的固定地址(Flag 地址)会被多写入一次，EEPROM 的寿命会被拉低。因此说，若(N+1)能被 256 整除时，建议多写入一个字节的空数据到 EEPROM；
  ⑤ **确保标志 Flag 的唯一性，**即选取的 Flag 要区别于每一个写入 EEPROM 的 Data。

- **采用以上的算法，实现以下一个 demo 程序，供参考：**

```
/*********************************************************************/
/*****该 Demo 是使用上了 SC92F7323 的 CODE 区域最后 256B 作掉电存储区域（做为 EEPROM）*****/
//该 Demo 是一个时钟演示程序，共有 4 个 Byte 的数据（即是天数，小时数
//分钟数，秒钟数）每 3 分钟写入内部 EEPROM
/*********************************************************************/
#include "SC92F7323_C.H"
#include "Hex2Bin.h"
#include "intrins.h"

void display_shifen(void) ;                    //数码显示时钟分钟
```

```
void Byte_Write(unsigned char DATA,unsigned char ADD_OFFSET); //IAP 写入数据函数
unsigned char Byte_Read(unsigned char AddOffset );//IAP 读取数据函数

#define uchar unsigned char                    //简化无符号字符
#define uint unsigned int                      //简化无符号整数
#define   ADD_BASE   0x1f00                    //定义 IAP 的基址 [根据 SOC 不同型号的 IC 确定基址]
/*****************************************/
/*****************************************/
/***需要存放进 EEPROM 的数据，4 个 Byte*******/
uchar nSec;
uchar nMin;
uchar nHour;
uchar  nday;
/*****************************************/
/*****************************************/
uchar  ADD_OFFSET=0;                       //偏移地址
uchar  code *POINT;                        //定义一个指针
uint offset,min3;

uint TusCounter;
uint nMinG;
uint nMinS;
uint nHourG;
uint nHourS;
uint nSecG;
uint nSecS;

uchar code chZimo [10]={0xc0,0xf9,0x64,0x70,0x59,0x52,0x42,0xf8,0x40,0x50};  //存字模

/***************************IAP 写入数据函数*****************************/
void Byte_Write(unsigned char DATA,unsigned char Add_Offset)
{

        IAPDAT=DATA;                       //送数据 DATA 到 IAP 数据寄存器
        if(Add_Offset>255)
        {
                ADD_OFFSET=0;
                Add_Offset=0;
        }
        IAPADH=0x1f;
        IAPADL =Add_Offset;                //写入偏移地址；
        IAPKEY=0x09;                       //任意写入一个非 0 值，打开 IAP 功能。
        IAPCTL=0x0a;                       //执行 IAP 写入操作，同时 CPU Hold 1ms。
        _nop_();_nop_();_nop_();_nop_();
        _nop_();_nop_();_nop_();_nop_();   //每次写入 IAP 数据需做 8 个 nop 的延时，确保写入了数据。
}
/************************IAP 读取数据函数**********************/
unsigned char Byte_Read(unsigned char AddOffset )
{
        POINT=ADD_BASE+AddOffset;          //指针 POINT 指向偏移地址；
        return (*POINT);                   //返回指针内容，读取成功。
}
//定时器 timer0 工作模式 2——8 位自动重载计数器/定时器；定时 50us
void timer0init()
{
        TMCON=_b00000001;                  //fsys=fosc/4
        TMOD=_b00000010;                   //方式 2
  /*载入初值******************定时 50us
```

```
  200*(1/4us)=50us;初值=（2^8-200)=56
  56=0x1060=_b 0011 1000
  高 8 位  10000011=0x38
  ****************************************/
        TH0=0x38;
        TL0=0x38;
  /*使能并启动 Timer*/
        TR0=0;
        ET0=1;
        TR0=1;
}
/********************软件延时********************/
void soft_delay(unsigned char n)
{
        unsigned char k;
        for(k=0;k<n;k++)
                _nop_();
}
void display_shifen(void)
{
//显示分个位
        P1=chZimo[nMinG];
        P21=1;
        P20=0;
        P07=0;
        soft_delay(800);                        //软延时
//显示分十位
        P1=chZimo[nMinS];
        P21=0;
        P20=1;
        P07=0;
        soft_delay(800);                        //软延时
//显示小时个位
        P1=chZimo[nHourG];
        P21=0;
        P20=0;
        P07=1;
        soft_delay(800);                        //软延时
}

void PRA_Write(void)                    //写数据到 EEPROM
{
        Byte_Write(nSec,ADD_OFFSET++);      //写入一个 Byte 到 EEPROM
        Byte_Write(nMin,ADD_OFFSET++);      //写入一个 Byte 到 EEPROM
        Byte_Write(nHour,ADD_OFFSET++);     //写入一个 Byte 到 EEPROM
        Byte_Write(nday,ADD_OFFSET++);      //写入一个 Byte 到 EEPROM
        Byte_Write(255,ADD_OFFSET++);       //写入标志 0xff；

        if(ADD_OFFSET==0)
                Byte_Write(0,250);              //清除上一次标志为 0；
        if(ADD_OFFSET==1)
                Byte_Write(0,251);              //清除上一次标志为 0；
        if(ADD_OFFSET==2)
                Byte_Write(0,252);              //清除上一次标志为 0；
        if(ADD_OFFSET==3)
                Byte_Write(0,253);              //清除上一次标志为 0；
        if(ADD_OFFSET==4)
                Byte_Write(0,254);              //清除上一次标志为 0；
        if(ADD_OFFSET==5)
```

```
                Byte_Write(0,255);                  //清除上一次标志为 0；
        if(ADD_OFFSET>5)
                Byte_Write(0,(ADD_OFFSET-6));//清除上一次标志为 0；
}

void PRA_Read(void)                                 //读出掉电前写入 EEPROM 的数据
{
        if(ADD_OFFSET==0)
        {
                nSec=Byte_Read(256-4);
                nMin=Byte_Read(256-3);
                nHour=Byte_Read(256-2);
                 nday=Byte_Read(256-1);
        }
        if(ADD_OFFSET==1)
        {
                nSec=Byte_Read(256-4+1);
                nMin=Byte_Read(256-3+1);
                nHour=Byte_Read(256-2+1);
                nday=Byte_Read(0);
        }
        if(ADD_OFFSET==2)
        {
                nSec=Byte_Read(254);
                nMin=Byte_Read(255);
                nHour=Byte_Read(0);
                nday=Byte_Read(1);
        }
        if(ADD_OFFSET==3)
        {
                nSec=Byte_Read(255);
                nMin=Byte_Read(0);
                nHour=Byte_Read(1);
                nday=Byte_Read(2);
        }
        if(ADD_OFFSET>=4)
        {
                nSec=Byte_Read(ADD_OFFSET-4);
                nMin=Byte_Read(ADD_OFFSET-3);
                nHour=Byte_Read(ADD_OFFSET-2);
                nday=Byte_Read(ADD_OFFSET-1);
        }
    }

void timer0()interrupt 1
{
        TH0=0x38;
        TusCounter++;
        if(TusCounter==20000)    //1s
                {
                        TusCounter=0;
                        nSec++;
                        P36=~P36;           //每 1s 闪一次灯
                        if(nSec>59)
                  {
                                nSec=0;
                                nMin++;                                 min3++;     //min3 每跑 1 分钟递增 1
                                if(nMin>59)
                                {
                                        nMin=0;
                                        nHour++;
```

```
                                        if(nHour>23)
                                        {
                                                nHour=0;
                                                nday++;
                                        }
                                        if(nday>9)
                                        {
                                                nday=0;
                                        }
                                }
                        }
                }
        //取秒钟
        nSecS=nSec/10;
        nSecG=nSec%10;
        //取分
        nMinS=nMin/10;
        nMinG=nMin%10;
        //取时
        nHourS=nHour/10;
        nHourG=nHour%10;
}

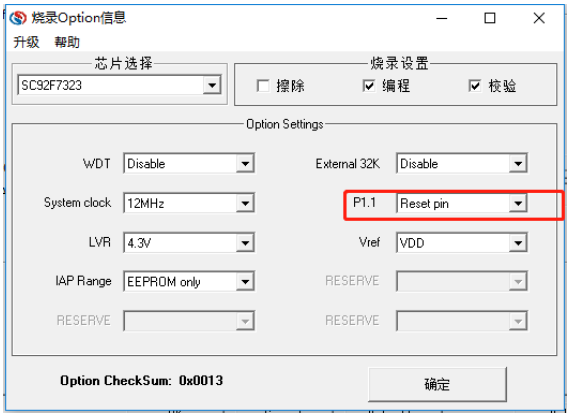
/*********************主程序*********************/
void main()
{
         timer0init();
        EA=1;
        for(offset=0;offset<256;offset++)           //查询标志 0xff
                if(Byte_Read(offset)==255)
                        ADD_OFFSET=offset;

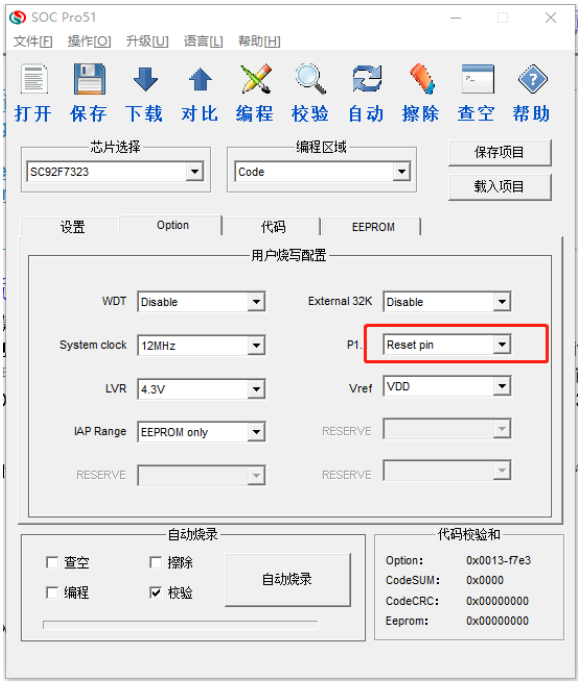
        PRA_Read();                                 //读出掉电前写入 EEPROM 的数据
        do
        {
                display_shifen();                   //显示时钟分钟表
                if(min3>3)
                {
                        min3=0;
                        PRA_Write();          //每 3 分钟写入一次数据。
                }
        }
        while(1);
    }
```

5 电路设计的注意事项

赛元 SC92F 系列 MCU 的 GPIO 上电默认模式为高阻输入模式。

赛元 SC92F 系列 MCU 的 RST 管脚，低电平使能，通过 Option 的对应管脚的控制选项可关闭 RST 功能取消 RESET 功能并将此 Pin 设为 GPIO，此后管脚低电平不会产生复位。Option 的设施如图（以 SC92F7323 为例）。左为 KEIL 的设置情况，右为烧录上位机的设置情况。

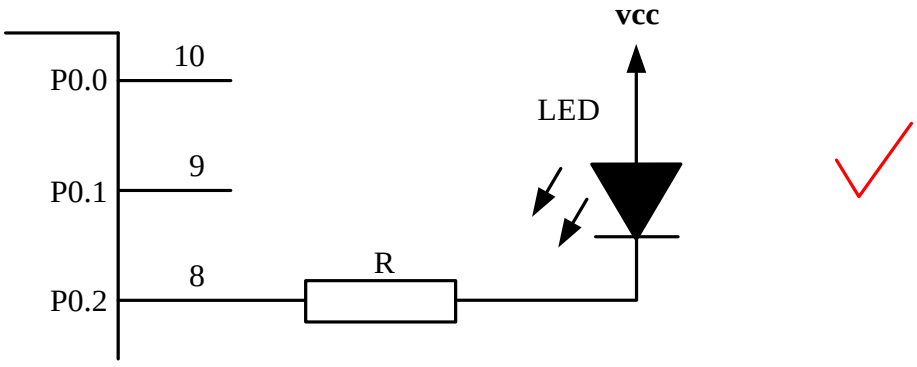




5.1 电路设计实例

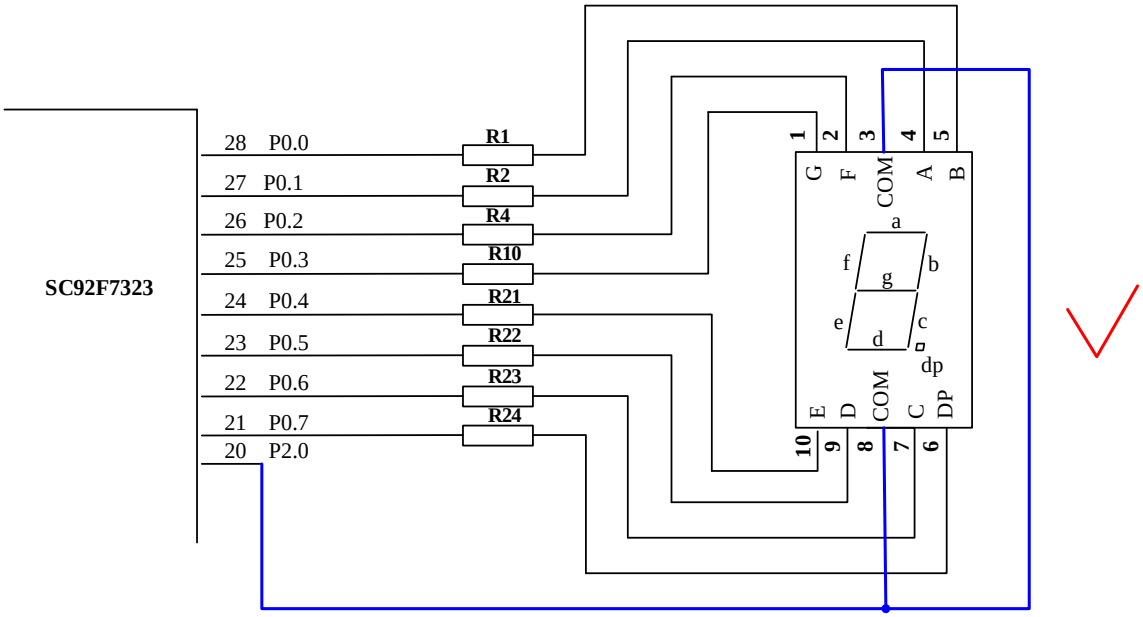
5.1.1 LED 的使用以及接法

- **建议接法：**LED 正向接到 VCC，负向接入 I/O。见下图：



5.1.2 1 位共阴极数码管的使用

- **建议接法：**见下图。

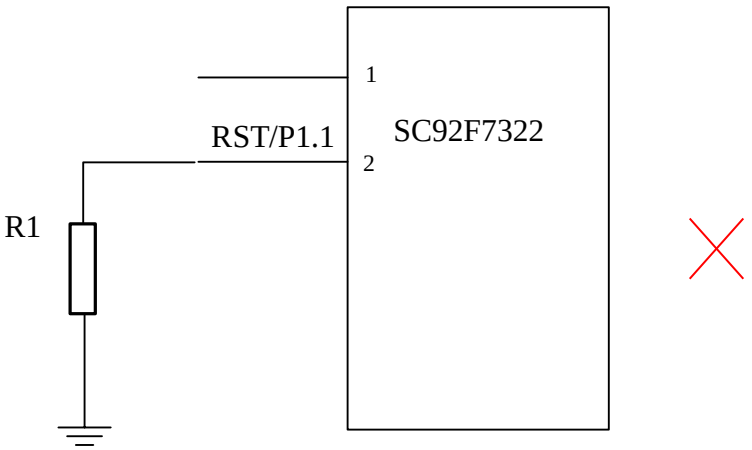


注意：至于其他多位数码管，对连接方法没有特殊要求，按正常接法即可，因为多位数码管对 MCU I/O 上电默认模式无关。

5.1.3 RST 管脚电路

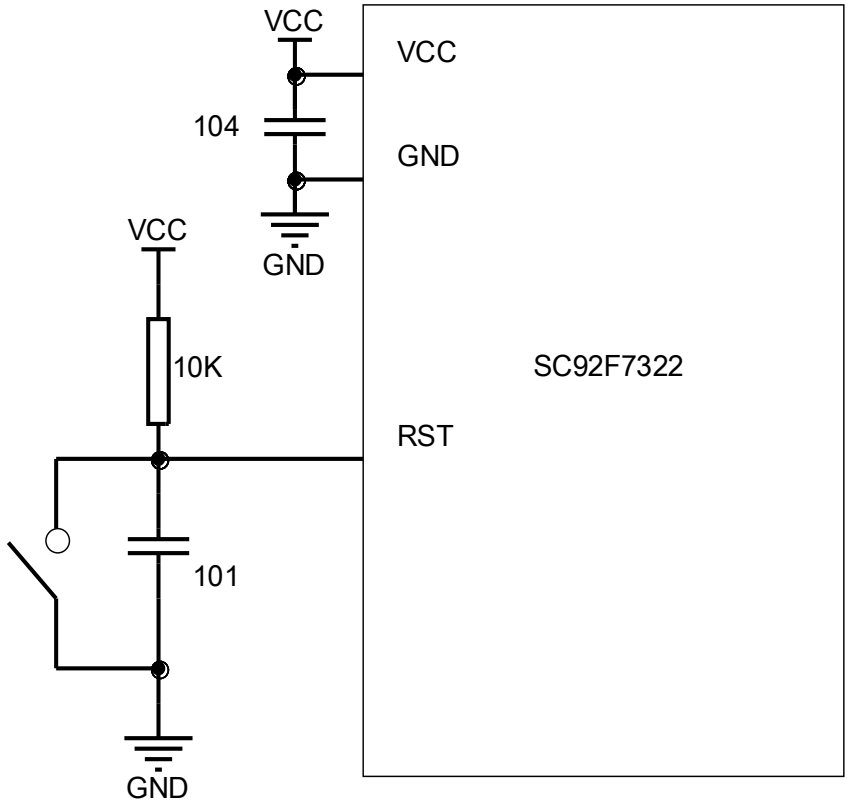
赛元 SC92F 系列 MCU 的 RST 引脚，与 I/O 复用，有别于传统 MCU 的 RST 引脚（传统的 RST 的引脚只能做输入，不能做输出），既可以做输入，又可以做输出。需要注意的是：SC92F7323、SC92F7322、SC92F7321、SC92F7320 这几个型号的芯片无论 Option 项 RST 口是设置为 RST 模式还是 IO 模式，上电后默认先为复位模式，然后才会将 RST 口设置为对应的模式，所以当使用这几个型号的芯片用 RST 口做普通 IO 口时，必须确保上电时 RST 口不为低。当将 IO 口设置为复位口时，上电后，用户电路的 RST 口不能一直为低，否则会一直复位，无法正常工作。因此，用户设计电路时，需要注意：

- 错误接法



说明：以上电路，若 RST 外接一个电阻 R1，系统在上电时与内部上拉判为低电平，造成系统一直复位，无法正常工作。

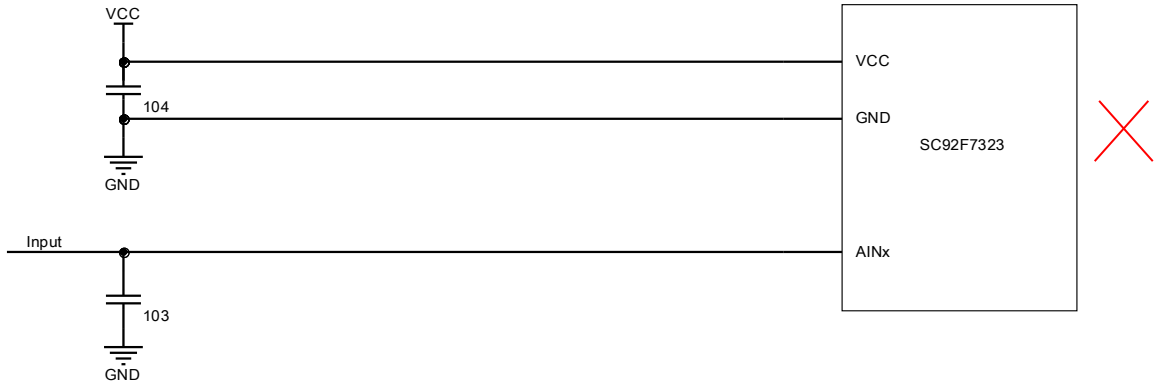
- **建议接法：**见下图。



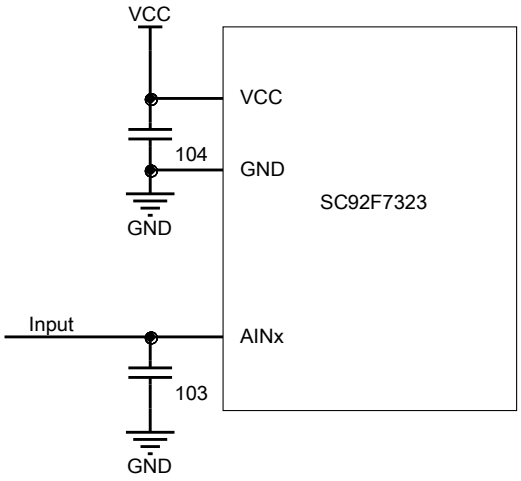
5.1.4 ADC 采样管脚电路

赛元 SC92F 系列 MCU 的 ADC 采样口需要在靠近管脚处加 103 电容， ADC 转换需要让电源电压稳定，所以在使用 ADC 功能时，请在靠近 IC 的 VCC 和 GND 处加 104 电容，以保证转换结果准确。

- 错误接法：104 电容离电源脚太远， 103 电容离 ADC 采样口太远

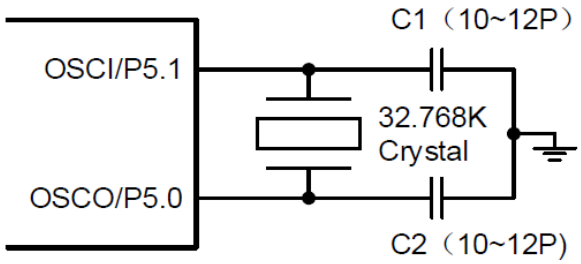


- **建议接法：**104 电容靠近电源脚， 103 电容靠近 ADC 采样口。见下图：



5.1.5 外部晶振电路

赛元 SC92F 系列 MCU 的部分型号提供了高频外部晶振接口或低频外部接口，用户如需使用外部晶振，匹配电容请根据所选晶振的要求进行选择，请将晶振电路靠近芯片引脚处。如图：



32k 外部晶振连接图

5.1.6 TOUCHKEY 电路

赛元触控 MCU 的触控架构分为高灵敏度触控模式和高可靠触控模式。

高灵敏度触控模式外接的 CMOD 电容容值范围为 472~104，推荐使用 103 电容，电容材质无特殊要求。

高可靠触控模式外接的 CMOD 电容容值范围为 332~473，推荐使用 103 电容。建议使用温度系数小精度高的电容，以免造成灵敏度不一致或随温度变化而变化。一般插件电容建议 5%精度涤纶电容，如需贴片电容则建议使用 10%或更高精度的 NPO 材质电容或 X7R 材质电容。

CMOD 电容需要尽量靠近芯片管脚。

5.2 IO 口各模式设置注意事项

赛元 SC92F 系列 MCU 的 GPIO，有三种工作模式：

1. 带上拉输入模式
2. 高阻输入模式
3. 强推挽输出模式

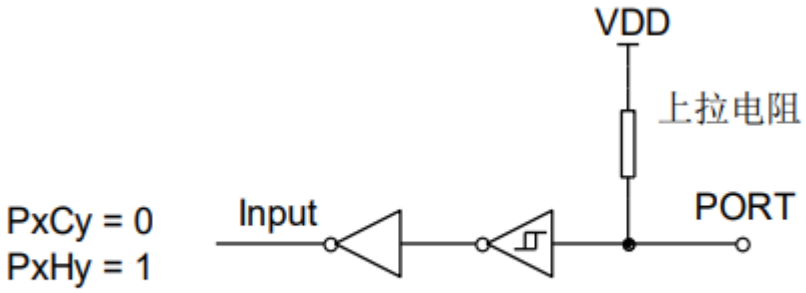
5.2.1 I/O 设为高阻，实现电路设计

通常来说，对于某些特定场合的应用，譬如电压检测，过零检测，LCD 的应用等，都是采用高阻工作模式来实现的。因此说，用户可以从赛元 MCU 体系中按需选择。

5.2.2 带上拉输入模式

带上拉的输入模式下，输入口恒定接一个上拉电阻，仅当输入口上拉电平被拉低时，才会检测到低电平信号。

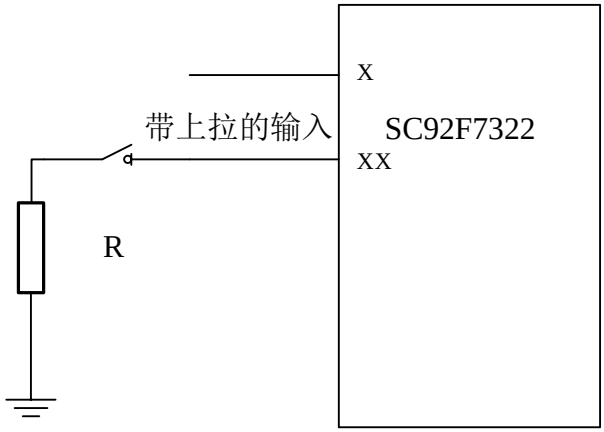
带上拉的输入模式的端口结构示意图如下：



带上拉的输入模式

5.2.3 带上拉输入模式检测按键

I/O 口作按键输入时，串接的下拉电阻 R 需小于 2K。



5.2.4 I/O 开漏输出模式的实现方式

赛元 SC92F 系列 MCU 没有开漏输出设置项，若用户想让 IO 口实现开漏输出功能，需要通过切换模式以达到开漏输出的效果，需要引脚输出低时切换位强推挽输出模式，需要引脚保持悬空状态时，则将 IO 口切换为高阻输入模式即可。

代码示例如下：

```
P0PH &= 0XFE;  //去除 P00 的上拉电阻
P00 = 0;             //将 P00 输出 0
P0CON &= 0xfe;  //P0 设置为输入模式,等效为开漏输出的开漏状态
P0CON |= 0x01;  //P0 设置为输出模式,等效为开漏输出低
```

6 软件编写的注意事项

赛元 SC92F 系列 MCU 内含有丰富的外设，只要配置相应的寄存器即可对其实现操作，但一些操作需要按要求进行，用户编程过程中需要注意以下几点。

6.1 PWM 设置及使用注意事项

当应用中需要关闭 PWM 时，用户需要根据实际应用需求对 PWM 对应 IO 口的输出寄存器进行设置，设置为“1“或者“0“（关闭 PWM，由 PWM 输出变为 GPIO 口，而此时的 IO 口输出为不确定状态）。

赛元 SC92F 系列 MCU 的 PWM 具有三种精度，分别为 12 位、10 位、8 位（具体精度描述请参照对应型号的规格书）。12 位精度的 PWM 周期共用，占空比可独立设置；10 位精度的 PWM 周期共用，占空比可独立设置，但各路 PWM 共用低 2 位的占空比设置。

使用 10 位 PWM 时，为保证 PWM 设置准确，需要先配置低 2 位的，再配置高 8 位。

```
PWMDTYA = 0x00;          //先配置 PWM 的低 2 位
PWMDTY0 = 50;            //再配置 PWM 的高 8 位
PWMDTY1 = 45;
PWMDTY2 = 40;
PWMCON0 = 0x38;          //先配置 PWM 周期的低 2 位
PWMPRD  = 59;            //再配置 PWM 周期的高 8 位
```

6.2 PCON 寄存器设置注意事项

赛元 SC92F 系列 MCU 提供了电源管理功能，可让芯片进入到省电模式，操作 PCON 寄存器的对应项即可，但在操作 PCON 寄存器后，请在其配置指令后接至少 8 个 NOP 指令，否则程序可能出错。

使用示例如下：

```
#include "SC92F7323_C.H"
#include "intrins.H"

PCON |= 0X02;                //进入 STOP 模式，后需接 8 个 NOP
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
```

请不要同时置起 STOP 位和 IDLE 位！

6.3 UART0 设置及使用注意事项

赛元 SC92F 系列的部分 MCU（SC92F735X、SC92F725X、SC92F732X）与传统的 51 单片机的波特率配置模式相似，使用 UART0 时用定时器 1 做波特率发生器时需要将 TR1 置 1，但赛元 SC92F 系列的其他型号的单片机在使用 UART0 时，如果选择 TIMER1 做波特率发生器，定时器 1 必须停止计数，即 TR1 = 0。使用 UART0 时需要将其对应的 TX 口设置为输入带上拉模式，保证 TX 口在空闲时为高电平。

使用示例如下：

```
P1CON &=0XDF;  //将 P15(TX0)设置为输入模式。
P1PH |= 0x02;      //将 P15(TX0)加上拉电阻；
SCON   = 0X50;   //设置通信方式为模式一，允许接收
T2CON &= 0XCF; //选择 T1 做波特率发生器
TR1 = 0;             //用定时器 1 作为波特率发生器，定时器 1 必须停止计数
TH1 = 0x06;        //在 16M 时，波特率为 9600；定时器初值[TH1,TL1] = Fsys/波特率
TL1 = 0x82;        //在 16M 时，波特率为 9600
EUART = 1;         //开启 Uart0 中断
```

赛元 SC92F 系列 MCU 的 UART0 不可直接发送 SFR 寄存器的值，若要通过 UART0 发出 SFR 的值，请先将 SFR 的值赋值给一个临时变量，再将临变量赋值给 SBUF。

使用示例如下：

```
unsigned char BufTemp;
BufTemp=ADCVH; //先将需要发出的 SFR 值存入一个临时变量
SBUF= BufTemp;   //再将临时变量赋值给 SBUF 发出。
```

或者用户可以将发送过程写成一个函数，将需要发送的数据作为入参进行发送。

```
void Uart0Send(usigned char data0)
{
    SBUF= data0;
}
Uart0Send(ADCVH); //通过调用函数来将 SFR 的数据发出。
```

6.4 SSI 设置及使用注意事项

使用赛元 SC92F 系列 MCU 的 SSI 功能时，请将所使用的 SSI 口设置为输入带上拉模式。

赛元 SC92F 系列 MCU 的 SSI 的 TWI 功能只能作为从机，若需要作为主机，请通过软件模拟实现。

当使用 SSI 的 UART1 的模式 3 时， RB8 接受数据只有置 1 功能的，因此使用模式 3 接收数据后，需要对 RB8 进行清零，如下：

```
uint16_t SSI_UART1_ReceiveData9(void)
{
   uint16_t Data9;
   Data9 =  SSDAT + ((uint16_t)(SSCON0&0X04)<<6); //获得接收的数据
   SSCON0 &= 0XFB; //将 RB8 清零。
   return Data9;
}
```

6.5 ADC 多通道切换采集注意事项

赛元 SC92F 系列 MCU 大多数型号拥有多路 ADC 通道，但每次转换只能转换一个通道，若想实现多路通道的 ADC 信号的采集，需要在一路 ADC 通道转换完毕后将转换口切换至另一路 ADC，如此反复以实现多通道的 ADC 转换。若在 ADC 通道切换后马上进行 AD 转换，通道口线上的电压可能存在不稳定的现象，在切换通道后转换的第一个值可能会存在异常，建议用户对某个通道做连续的多次采集和转换后，将切换通道后转换的第一个值或几个值去除或将最大值及最小值去除，再将剩余的 AD 转换值求平均值得到采集结果。

使用示例如下：

```
unsigned int ADC_Value0,ADC_Value1,ADC_Value2;
unsigned int ADC_Convert(void)
{
  unsigned  int Tad=0,MinAd=0x0fff,MaxAd=0x0000,TempAdd=0;
  unsigned char t=0;
  for(t=0;t<10;t++)
  {
          ADCCON |= 0X40;          //开始 ADC 转换
         while(!(ADCCON&0x20)); //等待 ADC 转换完成，不同型号的转换完成标志位位置不同，部分型号
                                                在 Bit5,部分在 Bit4,具体请参照规格。
          ADCCON&=~(0X20);       //清中断标志位
          Tad = ((unsigned int)ADCVH<<4)+(ADCVL>>4);  //取得一次转换值
          if (Tad>MaxAd)
          {
                  MaxAd=Tad      ;//获得当前的最大值
          }
          if (Tad<MinAd)
          {
                  MinAd=Tad;      //获得当前的最小值
          }
          TempAdd+=Tad;           //转换值累加
  }
  TempAdd-=MinAd;         //去掉最小值
  TempAdd-=MaxAd;         //去掉最大值
```

```
    TempAdd>>=3;            //求平均值
    return(TempAdd);
}

void ADC_channel(unsigned char channel)
{
   ADCCON = ADCCON &0xE0| channel;          //ADC 输入选择为 ADCchannel 口
}

void ADC_Multichannel()
{
  ADCCFG0 = 0x07;                 //设置 AIN0、AIN1、AIN2 设置为 ADC 口，并自动将上拉电阻移除。
  ADCCON  |= 0X80;                //开启 ADC 模块电源

  ADC_channel(0);                 //ADC 入口切换至 AIN0 口
  ADC_Value1 = ADC_Convert();  //启动 ADC 转换，获得转换值

  ADC_channel(1);                 //ADC 入口切换至 AIN1 口
  ADC_Value1 = ADC_Convert();  //启动 ADC 转换，获得转换值

  ADC_channel(2);                 //ADC 入口切换至 AIN2 口
  ADC_Value2 = ADC_Convert();  //启动 ADC 转换，获得转换值
}
```

6.6 使用定时器时外部中断服务函数编写注意事项

当使用了定时器功能时，请不要在外部中断服务函数内操作 TCON 的 TR1、TR0、TF1、TF0 位！

6.7 外部中断设置注意事项

赛元 SC92F 系列 MCU 在使用外部中断功能时，请将对应的 IO 口设置为输入模式！同组外部中断共用一个中断向量，用户需要在中断服务函数内读取 IO 口电平，判断中断的来源，再执行对应的操作。不建议将多个双边沿中断设置在同一组外部中断内。

使用示例如下：

```
P1CON &= 0XFC;   //将 INT10（P10）口设置为输入模式
P1PH1|=0X03;     //打开 P10 和 P11 的上拉电阻
INT1F = 0X03;    //使能 INT10、INT1 下降沿触发
EINT1 = 1;       //使能外部中断 1
EA = 1;          //开总中断
void Interrupt_work() interrupt 2
{
    if(P10==0)       //判断外部中断是否来自于 INT10
    {
        //执行代码
    }
    if(P11==0)       //判断外部中断是否来自于 INT11
    {
        //执行代码
    }
 }
```

6.8 LCD/LED 显示 RAM 使用注意事项

赛元 SC92F 系列的 SC92F854X、SC92F754X、SC92F844X、SC92F744X 型号 MCU 具有专用于 LCD/LED 及 PWM 功能的 RAM 区（700H-74FH），该区域的 RAM 只可写，不可读。用户如果有需要获取该区域 RAM 的数据，建议如下操作：

1、 在 RAM 开辟一片缓存区，用户可读取、改写缓存区。
2、 再将缓存区的内容覆盖掉 LCD/PWM RAM。

使用示例如下：

```
unsigned int xdata PWM_Duty[8] _at_ 0x740;
unsigned int PWM_Duty_Buff[8];   //用于存储写入 PWM_Duty 寄存器的值
unsigned int temp;
PWM_Duty_Buff [0] = 0x0000;      //将待写入的值存入缓存
PWM_Duty[0]=PWM_Duty_Buff[0];//将值写入对应的 PWM 控制寄存器
Temp = PWM_Duty_Buff [0];        //通过 PWM_Duty_Buff 获得 PWM_Duty 数组内的值
```

6.9 软件操作 CODE OPTION 的注意事项

赛元 SC92F 系列的 MCU 内部有单独的一块 Flash 区域用于保存客户的上电初始值设置，此区域称为 Code Option 区域。用户在烧写 IC 时将此部分代码写入 IC 内部，IC 在复位初始化时，就会将此设置调入 SFR 作为初始设置。

Option 相关 SFR 的读写操作由 OPINX 和 OPREG 两个寄存器进行控制，各 Option SFR 的具体位置由 OPINX 确定，各 Option SFR 的写入值由 OPREG 确定：

符号	地址	说明		上电初始值
OPINX	FEH	Option 指针	OPINX[7:0]	00000000b
OPREG	FFH	Option 寄存器	OPREG[7:0]	nnnnnnnnb

操作 Option 相关 SFR 时 OPINX 寄存器存放相关 OPTION 寄存器的地址，OPREG 寄存器存放对应的值。
例如：要将 OP_HRCR 配置为 0x01，具体操作方法如下：

C 语言例程：

```
OPINX = 83H;          //将 OP_HRCR 的地址写入 OPINX 寄存器
OPREG = 0x01;         //对 OPREG 寄存器写入 0x01（待写入 OP_HRCR 寄存器的值）
```

汇编例程：

```
MOV OPINX,#83H;       //将 OP_HRCR 的地址写入 OPINX 寄存器
MOV OPREG,#01H;       //对 OPREG 寄存器写入 0x01（待写入 OP_HRCR 寄存器的值）
```

注意：禁止向 OPINX 寄存器写入 Customer Option 区域 SFR 地址之外的数值！否则会造成系统运行异常！

6.10 TOUCHKEY 设置注意事项

应用程序初始化时需要将 TK 对应的 IO 口设置为强推挽输出模式，输出高电平。在 TK 扫描过程中，不能去操作使用 TK 对应的 IO。

更多 TouchKey 的注意事项请参考《赛元 SC92F_93F 系列 TouchKey MCU 应用指南》。

7 版本更改记录

	记录	日期
V1.4	1. PWM 的 IO 口状态设置说明更改 2. 增加了 SSI 的 UART1 的工作模式 3 的使用说明 3. 增加了 TouchKey 的相关注意事项。	2018 年 12 月
V1.3	1. ADC 转换完成标志位不同型号差异说明补充, ADC 代码错误修改，2. 更新了 UART0 使用说明	2018 年 9 月
V1.2	1. 添加了第二功能 IO 口设置为输入带上拉的注意说明 2. UART0 的 TI 和 RI 误清除的描述错误修改	2018 年 7 月
V1.1	增加 UART0 的使用注意事项	2018 年 6 月
V1.0	初版	2018 年 5 月