目录

1 赛元 SC92F7003_8003 电气参数

- **工作电压：** 2.4V~5.5V
- **工作温度：** -40 ~ 85℃
- **内核：**高速 1T 8051
- **IO：**18 个双向可独立控制的 I/O 口， 全部 IO 具有大灌电流驱动能力（70mA），SC92F7003_8003 IO 没有开漏输出模式，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为强推挽输出模式，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
- **Flash ROM：**（MOVC 禁止寻址 0000H~00FFH）可重复写入 1 万次
- **IAP：**可 code option 成 0K、0.5K、1K 或 8K
- **EEPROM：**独立的 128Byte，可重复写入 10 万次，10 年以上保存寿命
- **内建高频 16MHz 振荡器（f_{HRC}）：**作为系统时钟源时，f_{SYS} 可通过编程器选择设定为 16/8/4/1.33MHz
- **内置高频晶体振荡器电路：**可外接 2~16MHz 振荡器，需在外接晶振输入输出引脚 OSCI、 OSCO 对地各加约 15pF 的起振电容。当外振失效时，振荡器自动切换为 f_{HRC}，直至重新上电振荡器才恢复为外振。外振作为系统时钟源时，f_{SYS} 可通过编程器选择使用外接晶振 /1 /2 /4 /12 这四种分频中的一种
- **IC 系统时钟（f_{SYS}）对应的工作电压范围：**
  - >12MHz @2.9~5.5V
  - ≤12MHz @2.4~5.5V
- **内建低频 128kHz LRC 振荡器：**
  - 可作为 BaseTimer 的时钟源，并唤醒 STOP
  - 可作为 WDT 的时钟源
- **低电压复位（LVR）：**复位电压有 4 级可选，分别是：4.3V、3.7V、2.9V、2.3V（LVR 必须低于 VDD）
- **省电模式：**
  - IDLE Mode，可由任何中断唤醒
  - STOP Mode，可由 INT0、1、2 和 BaseTimer 唤醒

2 赛元 SC92F7003_8003 烧写注意事项

1. 赛元 SC92F7003_8003 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 1000μF 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 SC92F7003_8003 软硬件开发平台介绍

3.1 开发平台 KEIL C

赛元 MCU，采用 Keil C 即 KEIL uVISION 平台来开发，支持汇编语言以及 C 语言编写。

KEIL uVISION ，是众多单片机应用开发软件中最优秀的软件之一，它支持众多不同公司的 MCS51 架构的芯片甚至 ARM，它集编辑，编译，仿真等于一体，界面与常用的微软 VC++界面相似，界面简洁，易学易用，在调试程序，软件仿真方面也有很强大的功能。

有关 KEIL C 的使用，请参考赛元官网资料“赛元 MCU 工具使用说明”文档的第二部分“赛元 MCU 的开发平台——Keil C”，有 Keil C 的安装及新建工程等使用说明。

3.2 烧录仿真工具

赛元目前使用的烧录工具有 SC-LINK，DPT52，PRO52。烧录工具使用前请安装赛元仿真插件。SC-LINK 适用于赛元 92F/93F 系列 IC 的脱机烧录、在线烧写、仿真以及 92F/93F 系列触控 IC 的 Touch 调试。在线工具 DPT52 适用于赛元所有系列 IC 的在线编程、触控系列 IC 的调试以及部分系列 IC 的仿真。量产编程工具 PRO52 适用于赛元所有系列 IC 的量产烧写。有关赛元烧录仿真工具的使用与仿真插件的安装，请参考赛元官网资料“赛元烧录仿真工具 SC-LINK 使用说明”、 “赛元在线开发工具 DPT52 使用说明”、 “赛元量产编程工具 PRO52 使用说明”。

3.3 PC 端烧录软件 SOC PRO51

赛元 PC 端烧录上位机 SOC PRO51，支持 SC-LINK、DPT52、PRO52 等全系列烧录仿真工具。关于 SOC PRO51 的安装步骤与使用说明请参考赛元官网资料“赛元 MCU 工具使用说明”文档“1.3 软件安装步骤”与“1.4 开发工具功能说明与操作流程”（文档中烧录软件界面为旧版 V3.05 界面）。

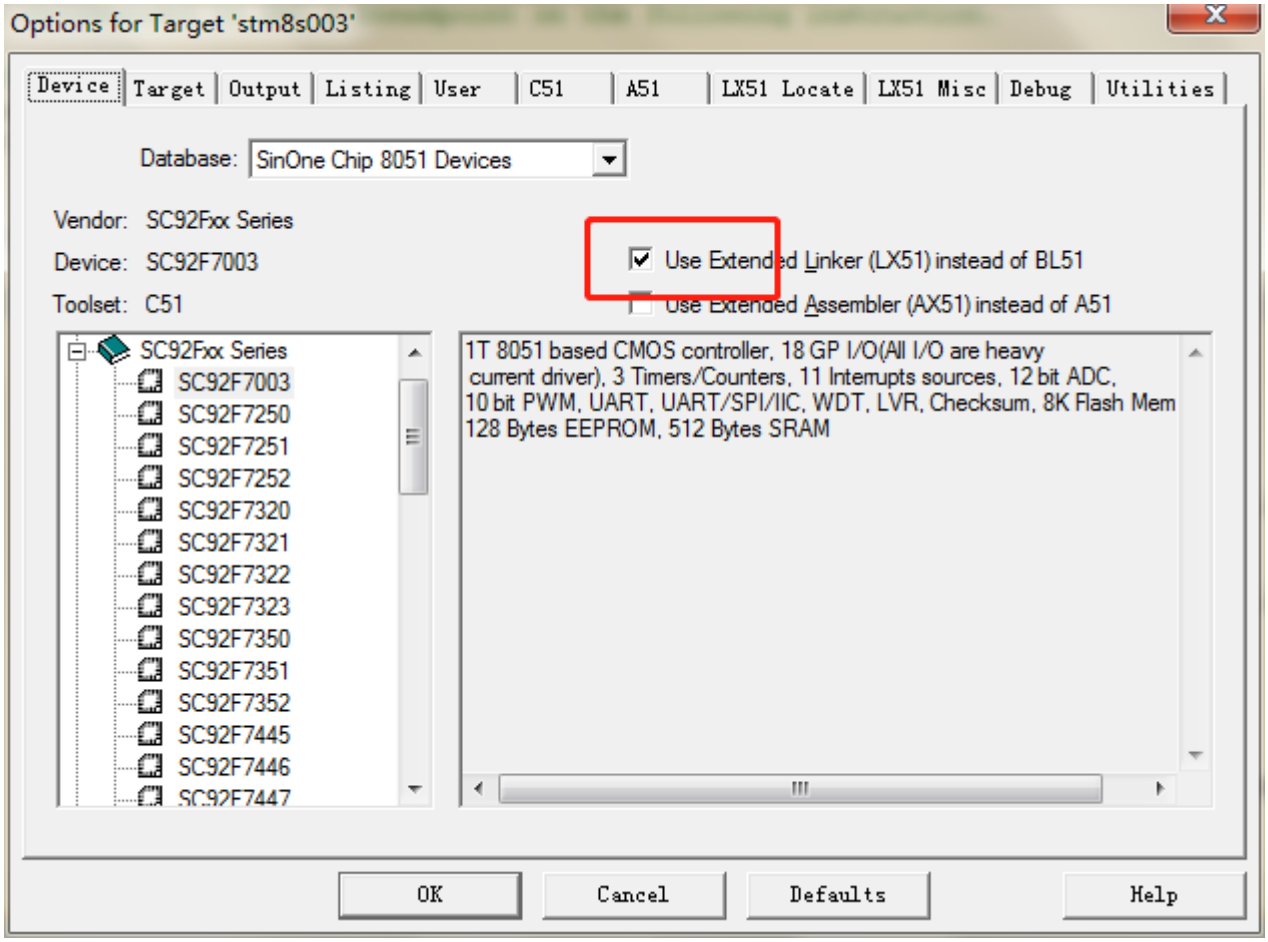
4 KEIL 中优化未调用函数

4.1 未调用函数优化说明

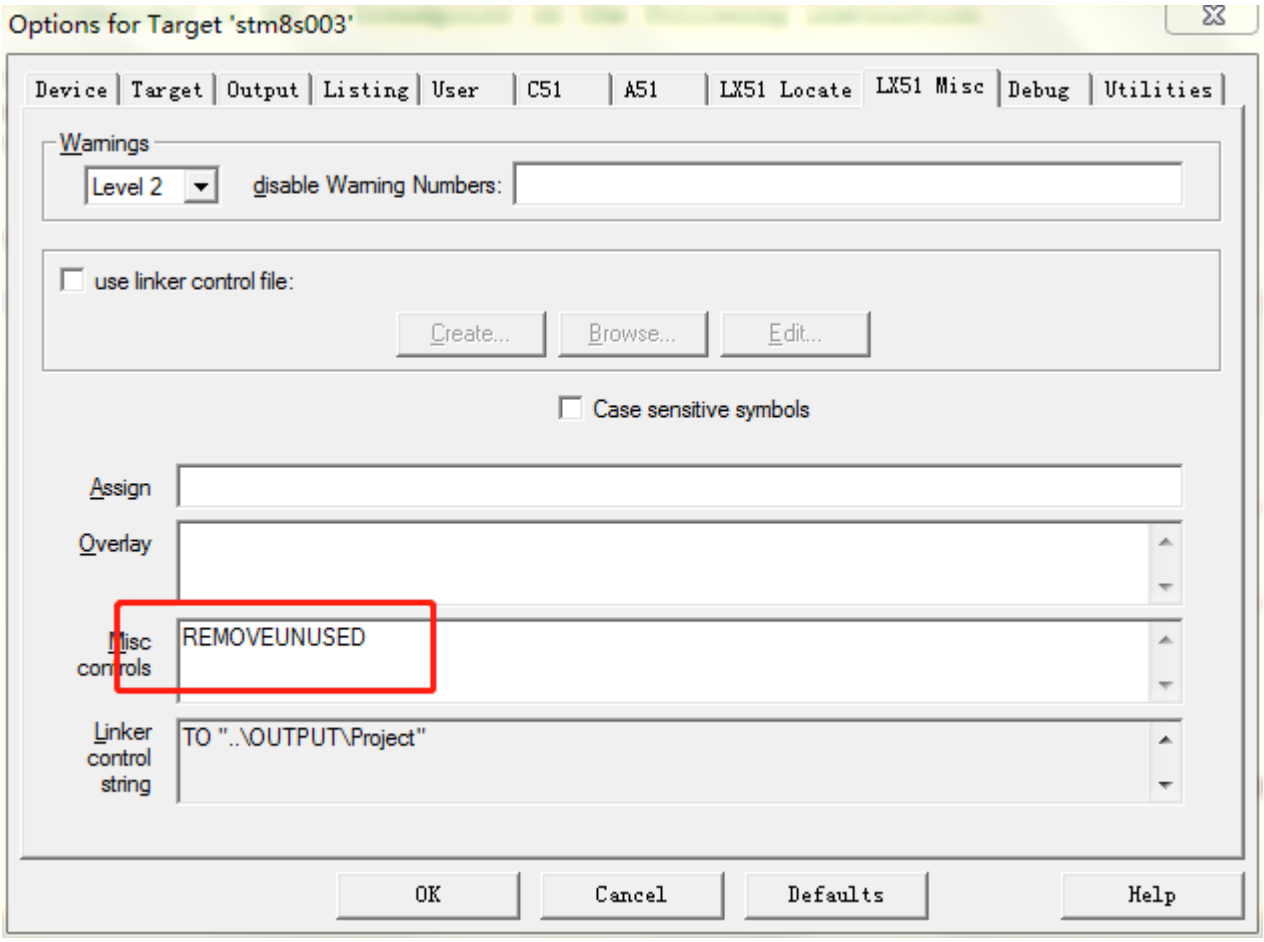
使用 Keil 为 C51 单片机编程时，在程序中常常有一些我们当前未使用的函数，特别是在调用函数库的时候。编译这样的代码时，总会弹出一些警告“UNCALLED SEGMENT，IGNORED FOR OVERLAY PROCESS”，如何在不删去这些函数的同时，又使这部分代码不被编译和链接进最终的程序，浪费单片机的存储空间？这里提供一个方法供参考。

4.2 未调用函数优化方法

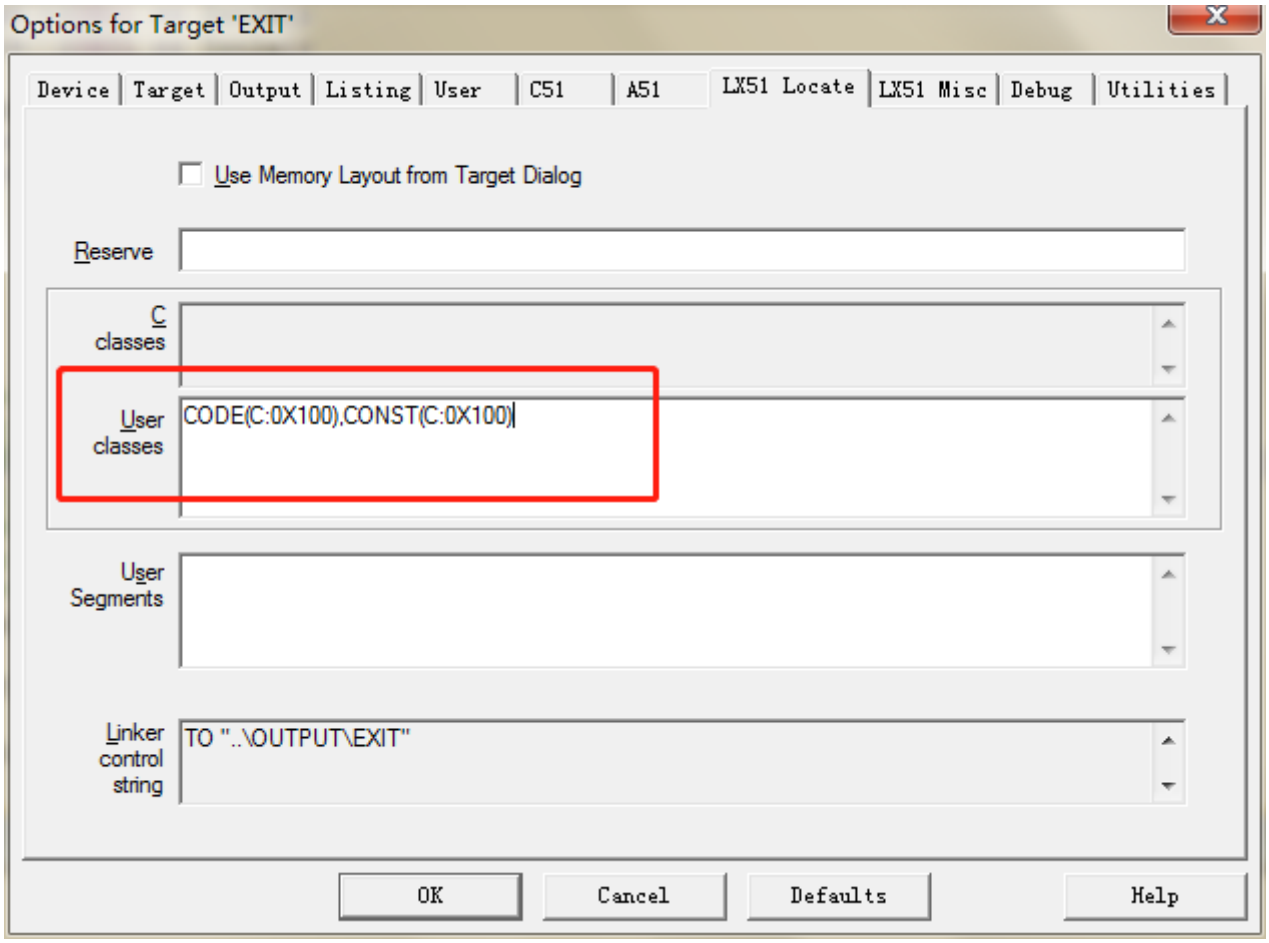
第一步，在 target options 的 Device 页中勾选“Use Exended Linker(LX51) instead of BL51” 。



第二步，在 LX51 Misc 页的 Misc controls 中填入 REMOVEUNUSED。



第三步，由于赛元 MCU Flash ROM 的起始 256B ROM 区间，禁止 MOVC 寻址。故在 LX51 Locate 页的 User classes 中填入 CODE(C:0X100),CONST(C:0X100)。



至此，编译后产生的警告项消失，code 占用量亦减少。

5 赛元 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。

5.1 C 语言编程有关 MOVC 指令的应用注意

5.2 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

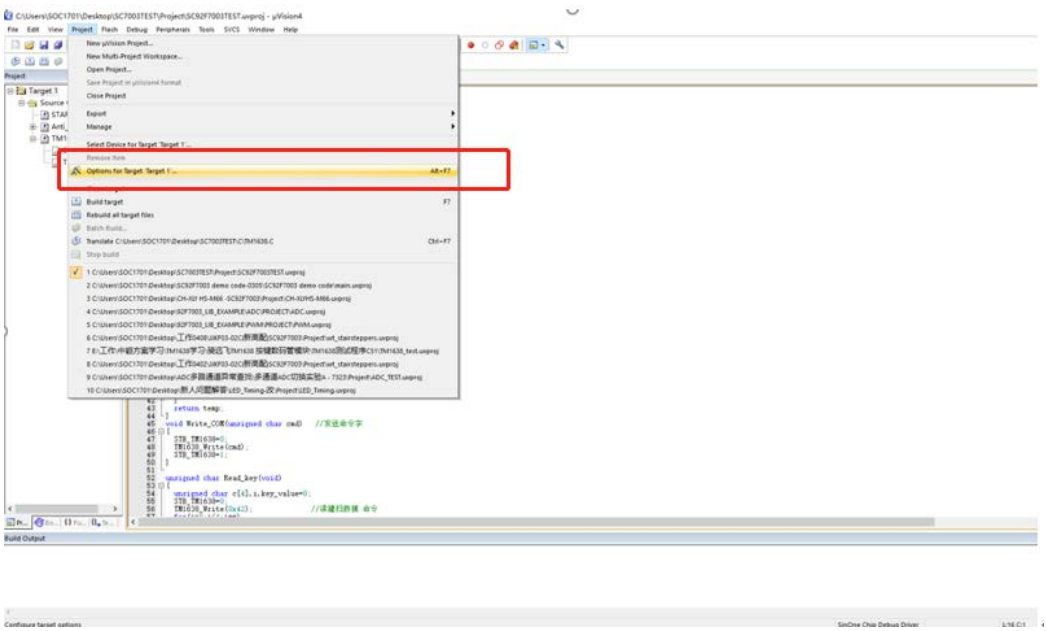
注意：?C?LIB_CODE 标识符只是表明某个函数调用的库文件进行查表运算，通常情况下，客户开发产品不需用调用库文件 Math 函数。（库文件占用较大的 ROM 空间，譬如 sinx 函数）。

.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。

5.2.1 C 语言开发具体操作

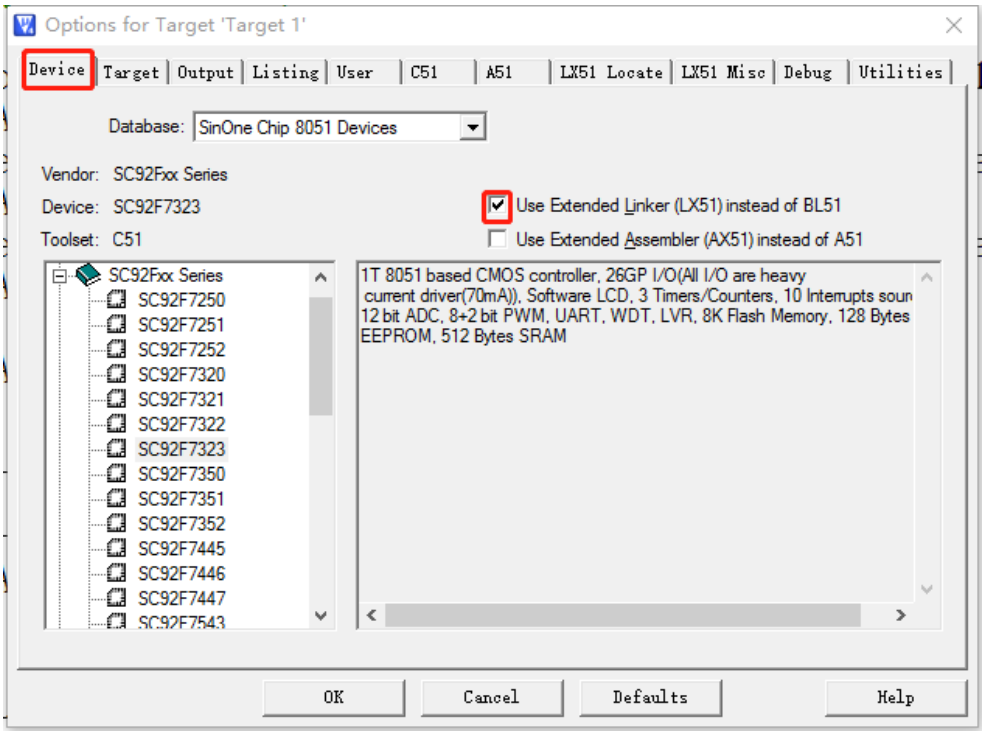
由上描述，用户在采用 C 语言开发过程当中，需要把全局变量，不可变类型数据（code 类型数据）定义在 Flash ROM 起始的 256B 地址之后。因此，在开发调试时，可以先采用屏蔽该区域的 Flash Rom 来进行开发，待调试完毕后，再做调整，生成最终的程序。使用不同的链接器操作方法不同，具体方法见下：

在 LX51 链接器上实现调整的方法（推荐方法）：



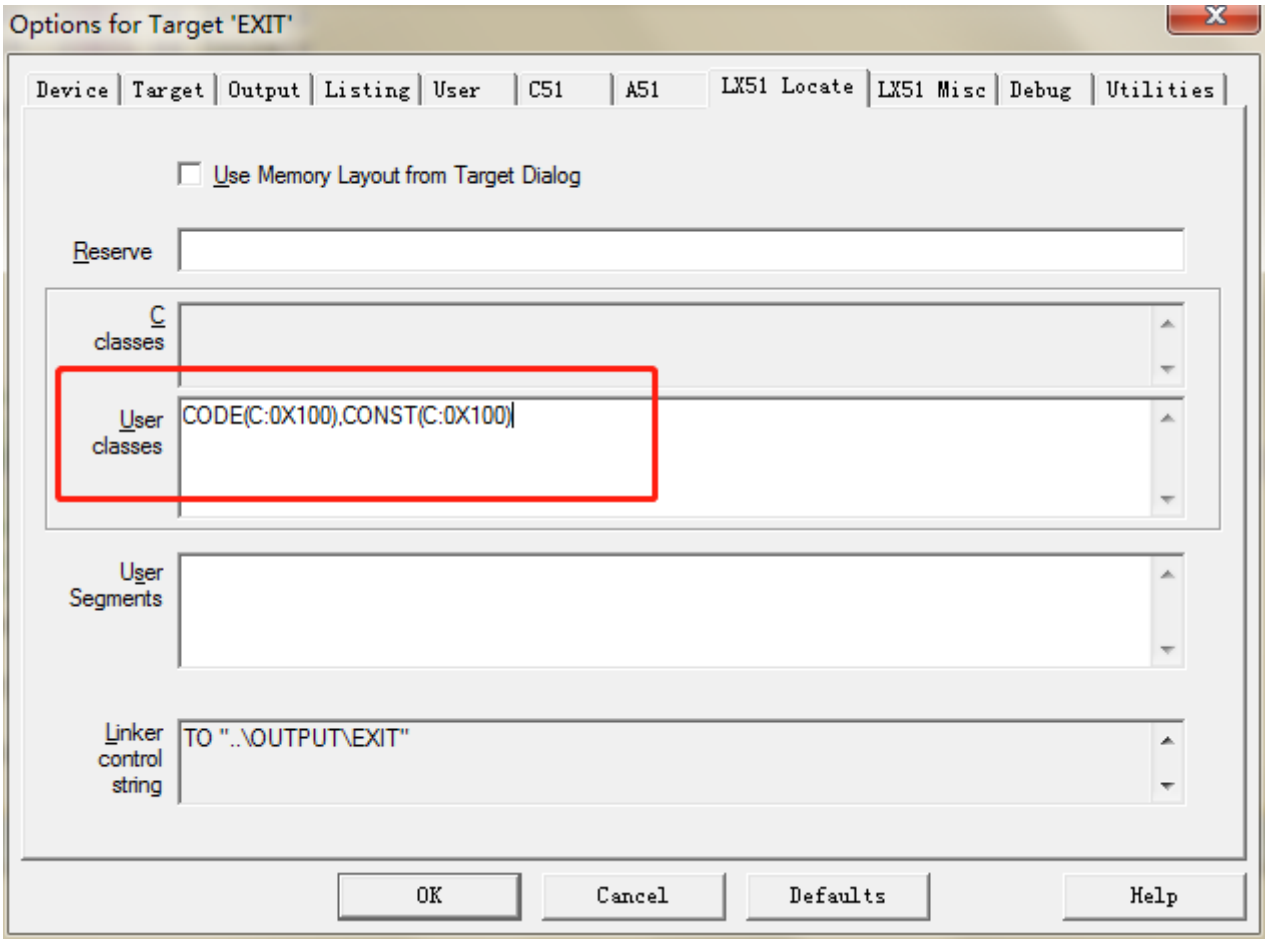
◆ **选择使用 LX51 链接器。**

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。**

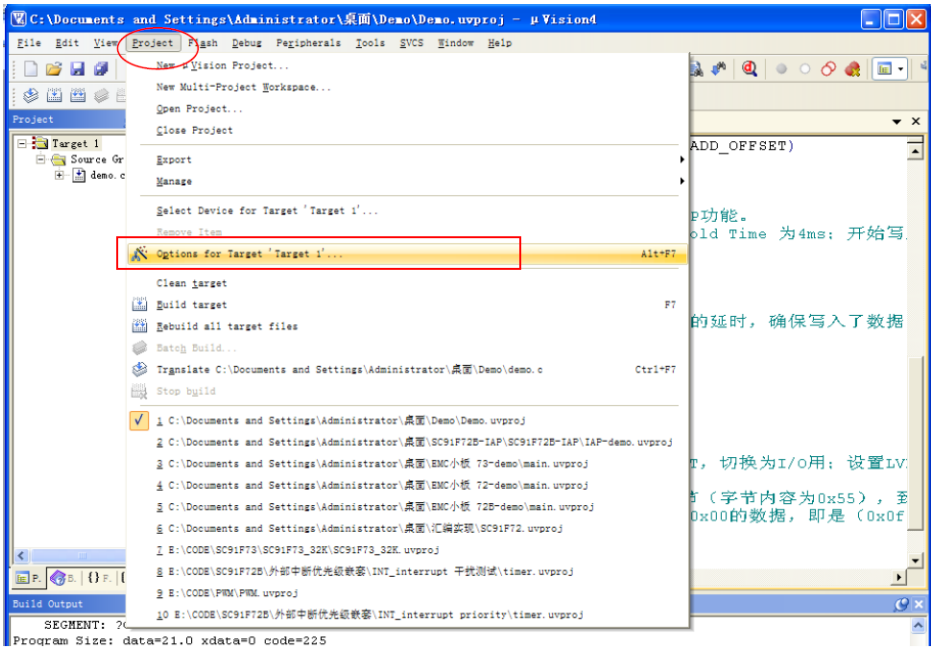
打开项目选项中的“LX51 Locate”属性页，在“User classes”的输入框中填入“CODE(C:0X100), CONST(C:0X100)”指令，点击 OK 完成配置，见下图：

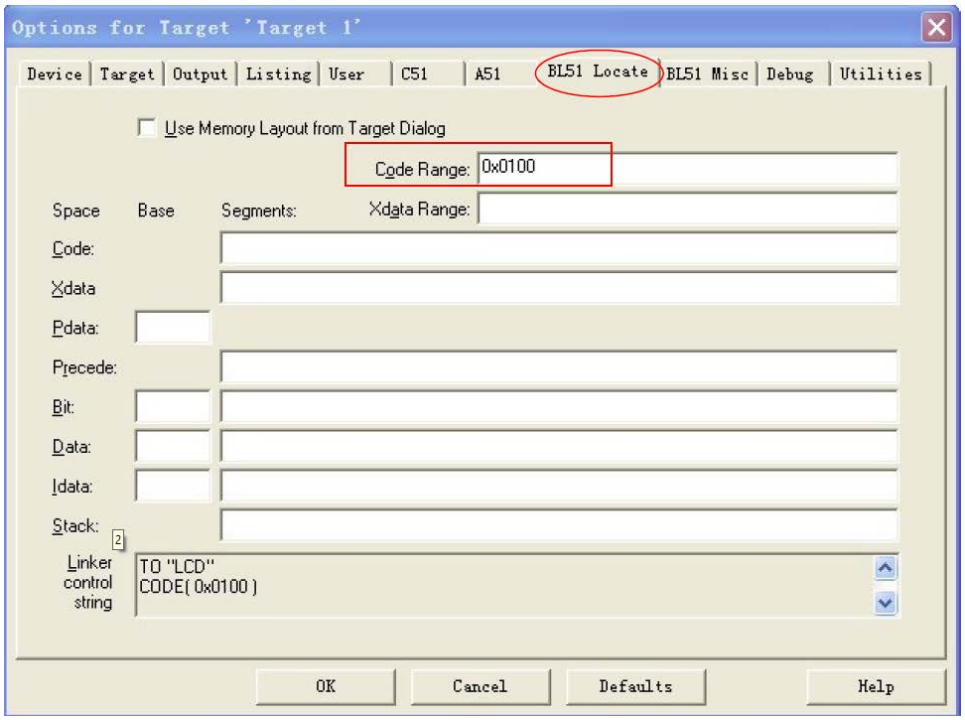


在 BL51 链接器上实现调整的方法：

- **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：



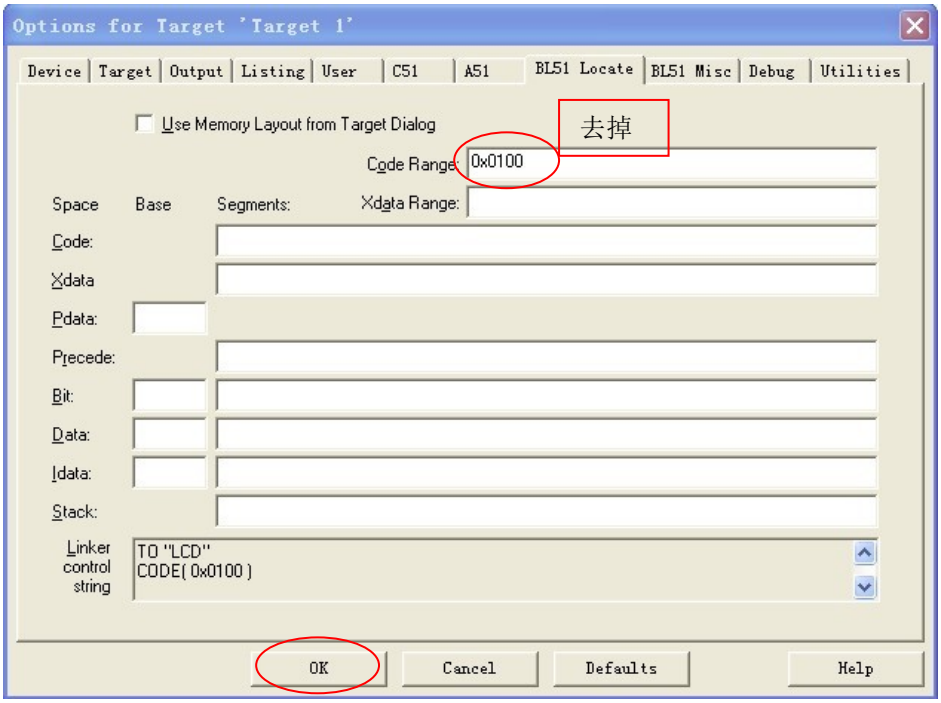


◆ **调试完成后，生成最终程序；**

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：

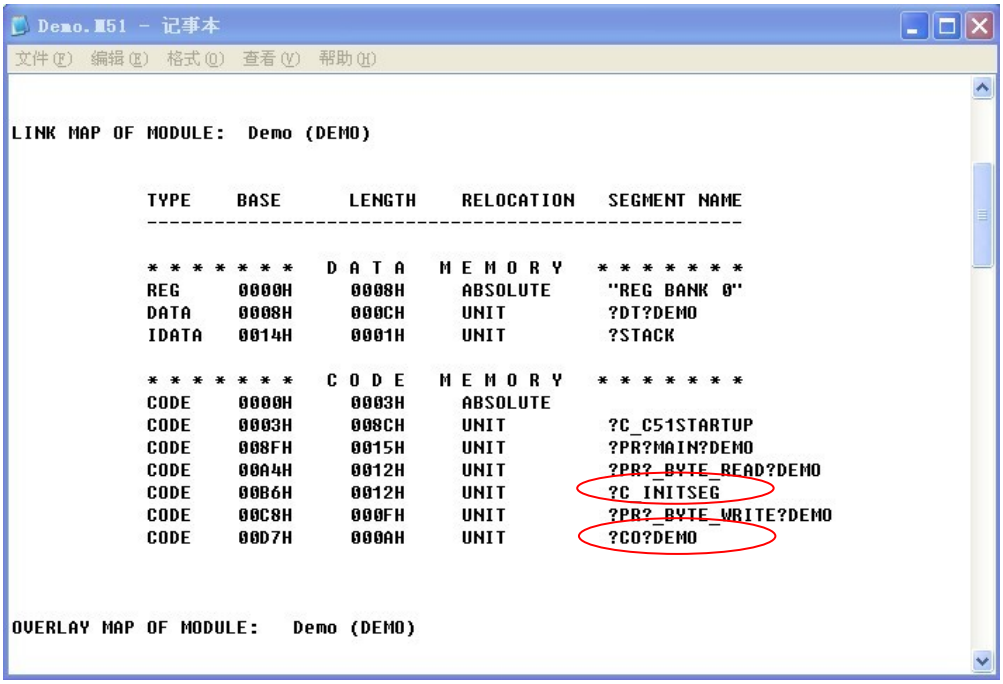
1）将代码存放区恢复回全区域，即取消第一步的操作。即将 Code Range 的数据去掉，点击 OK 保存。



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；

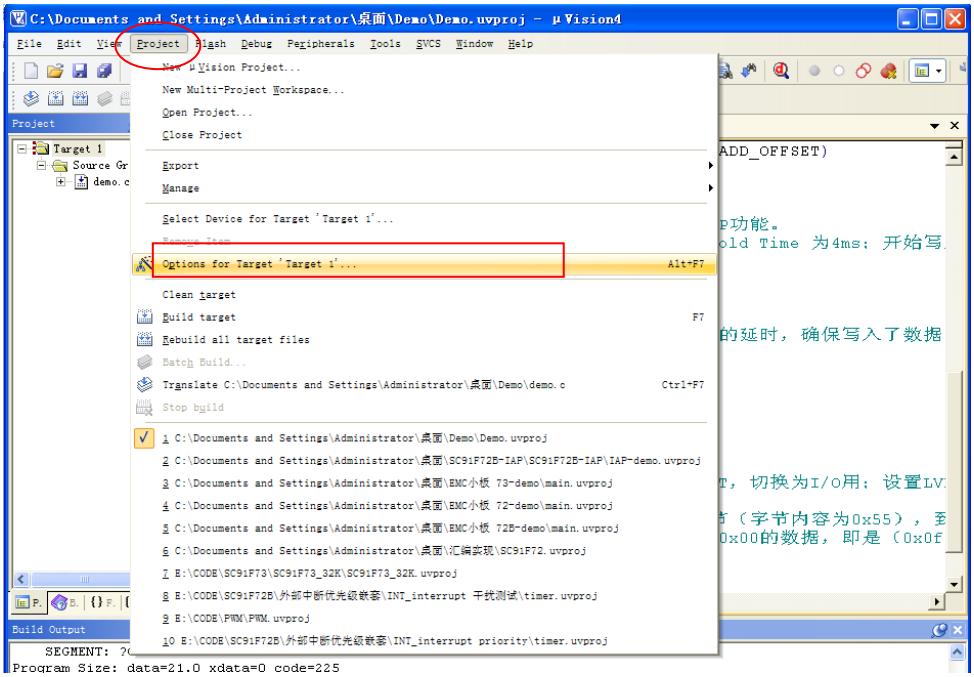
“?CO?DEMO” :code 类型数据。

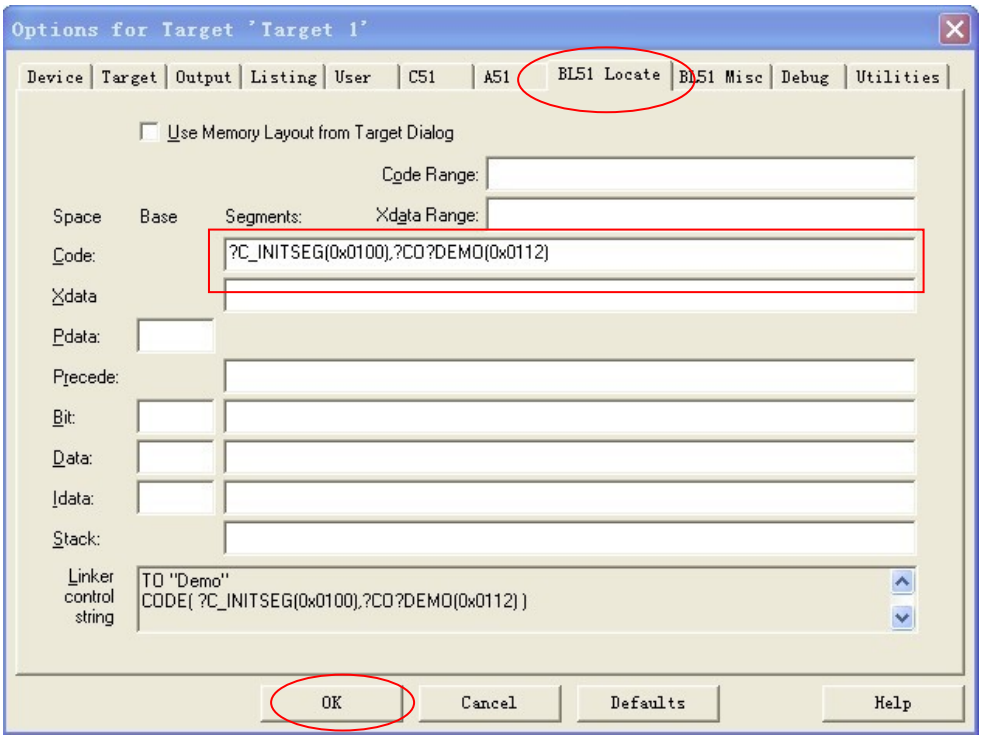


说明:从以上 M51 文件的“CODE MEMORY”信息中，可以看到“?C_INITSEG”，链接地址为 00B6H，长度为 0012H 字节；“?CO?DEMO”，链接地址为 00D7H，长度为 000AH 字节。

a) 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。

b) 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)”





c) 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（或者 2KB）的外部 RAM，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，将如图所示的区域填入 100H，若 RAM 空间为 2K 大小的单片机，则将 XDATALEN 对应值填 0X700。

```
17  ;
18  ;     Lx51 your object file list, STARTUP.OBJ  controls
19  ;
20  ;------------------------------------------------------------------------------
21  ;
22  ;  User-defined <h> Power-On Initialization of Memory
23  ;
24  ;  With the following EQU statements the initialization of memory
25  ;  at processor reset can be defined:
26  ;
27  ; <o> IDATALEN: IDATA memory size <0x0-0x100>
28  ;     <i> Note: The absolute start-address of IDATA memory is always 0
29  ;     <i>       The IDATA space overlaps physically the DATA and BIT areas.
30  IDATALEN        EQU     100H
31  ;
32  ; <o> XDATASTART: XDATA memory start address <0x0-0xFFFF>
33  ;     <i> The absolute start address of XDATA memory
34  XDATASTART      EQU     0
35  ;
36  ; <o> XDATALEN: XDATA memory size <0x0-0xFFFF>
37  ;     <i> The length of XDATA memory in bytes.
38  XDATALEN        EQU     100H
39  ;
40  ; <o> PDATASTART: PDATA memory start address <0x0-0xFFFF>
41  ;     <i> The absolute start address of PDATA memory
42  PDATASTART      EQU     0H
43  ;
44  ; <o> PDATALEN: PDATA memory size <0x0-0xFF>
45  ;     <i> The length of PDATA memory in bytes.
46  PDATALEN        EQU     0H
47  ;
48  ;</h>
49  ;------------------------------------------------------------------------------
```

5.3 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。

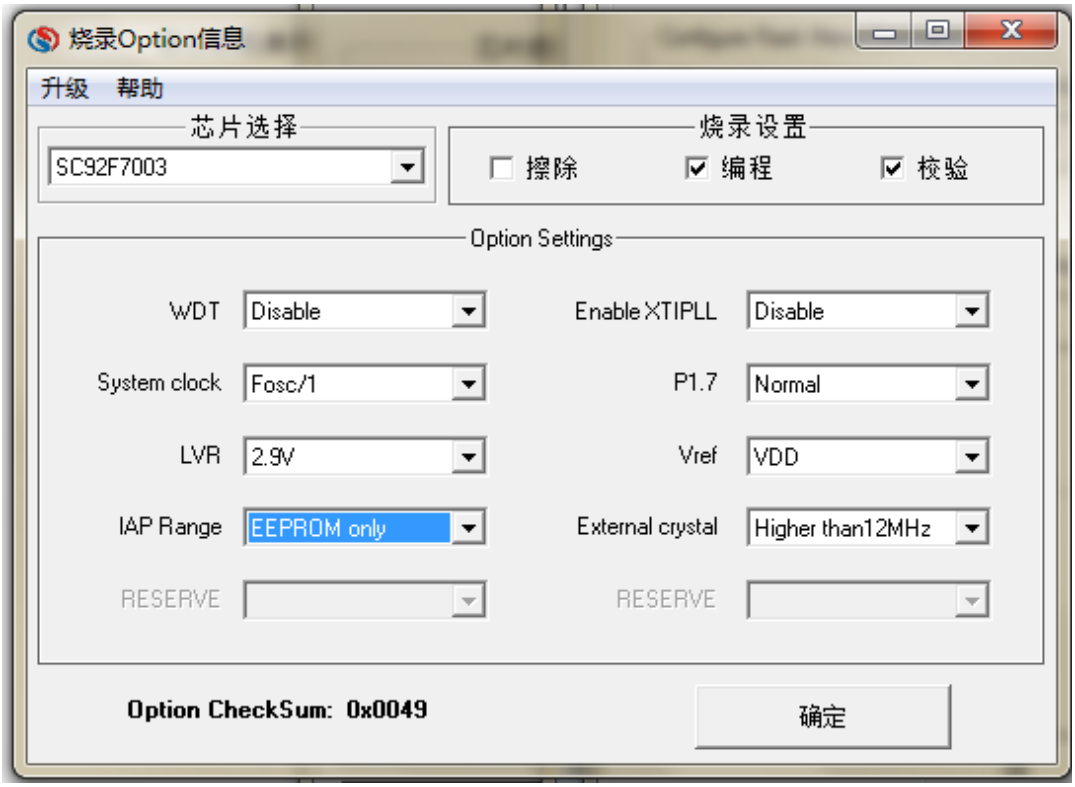
6 赛元 MCU 的 EEPROM 及算法解说

6.1 内部 EEPROM 的操作及 CODE 的 IAP 操作

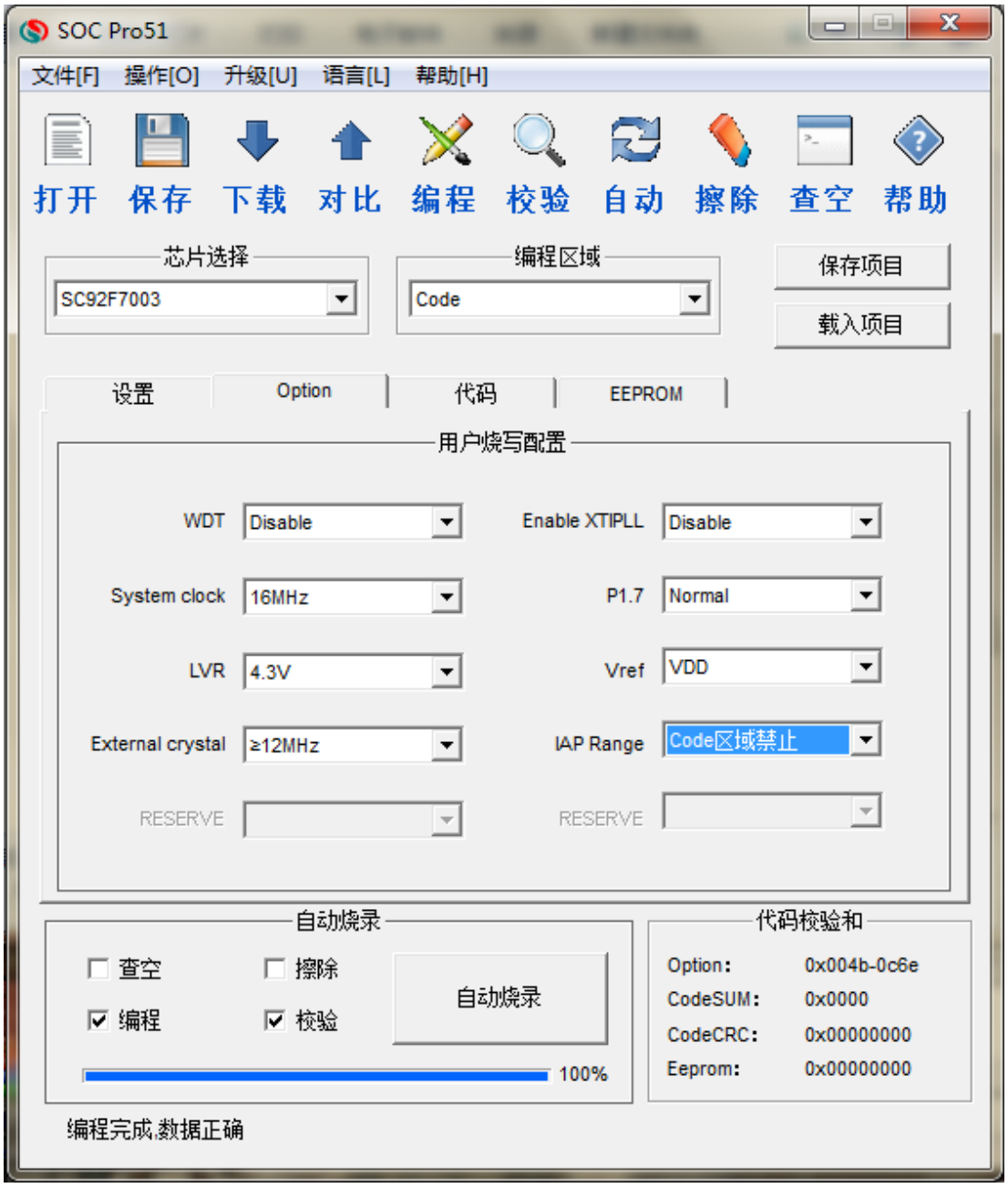
以 SC92F7003 为例，说明赛元 MCU 内部 EEPROM 的使用方法及 CODE 区 IAP 操作的方法。

SC92F7003 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 Ranged 的选项中选择允许 IAP 操作的范围。



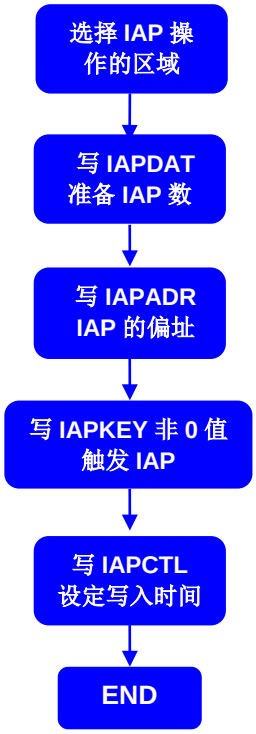
在上位机烧录时也是在 Option 项中的 IAP Ranged 的选项中选择允许 IAP 操作的范围。如图：



- **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：



6.2 EEPROM 操作代码

SC92F7003_8003 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。

6.2.1 128 BYTE 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;                 //若总中断已使能，需先关总中断
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，1.5mS@16/8/4/1.33MHz;
_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;                          //若总中断已使能，需先关总中断
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
```

6.2.2 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，1.5mS@16/8/4/1.33MHz;
_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：

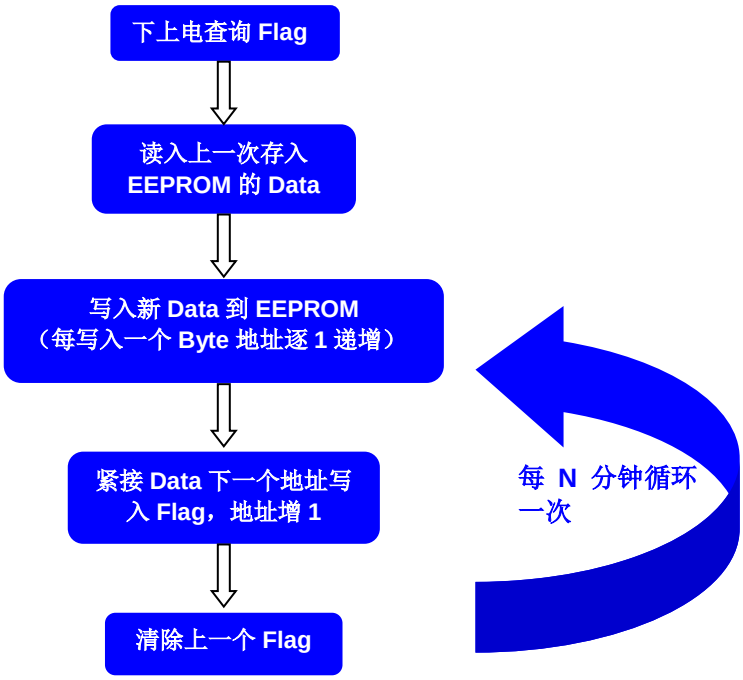
```
IAPADE = 0X00;                  //选择 Code 区域
IAP_Data = *( POINT+IAP_Add);   //读取 IAP_Add 的值到 IAP_Data
```

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)，不建议用户使用。

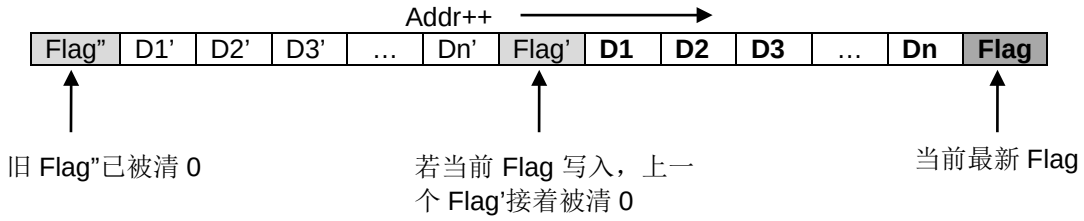
6.3 EEPROM 的使用算法

由于 SC92F7003_8003，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 SC92F7003_8003 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 "SC92F7003_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;
      IAPADE=0x00;
      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)
{
    P1=chZimo[nMinG];              //显示分个位
        P21=1;
    P20=0;
    P07=0;
    soft_delay(800);               //软延时
    P1=chZimo[nMinS];              //显示分十位
    P21=0;
    P20=1;
    P07=0;
    soft_delay(800);               //软延时
    P1=chZimo[nHourG];             //显示小时个位
    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;
                    }
                }
            }
        }

    nSecS=nSec/10;          //取秒钟
    nSecG=nSec%10;

    nMinS=nMin/10;          //取分
    nMinG=nMin%10;

    nHourS=nHour/10;        //取时
    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);
}
```

7 软件编写与电路设计注意事项

赛元单片机内含有丰富的外设，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。

7.1 PWM 占空比及周期的低 2 位设置注意项

SC92F7003_8003 提供了 7 路共用周期、单独可调占空比的 10 位 PWM 输出：PWM0~6。

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

使用示例如下：

```
PWMDTYA = 0x15;          //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
```

7.2 PCON 寄存器设置注意事项

SC92F7003_8003 单片机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 "SC92F7003_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_();
```

7.3 UART0 设置注意事项

SC92F7003_8003 单片机 UART0 需要占用一个定时器资源，因此，**在使用 UART0 时，需将 TXD 设置为输入带上拉模式，保证 TXD 口在空闲时为高电平。**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

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 的值赋值个一个临时变量，再将临变量赋值给 SBUF。

使用示例如下：

```
usigned 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。
```

7.4 ADC 注意事项

7.4.1 ADC 多通道切换

SC927003_8003 单片机拥有多路 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，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：

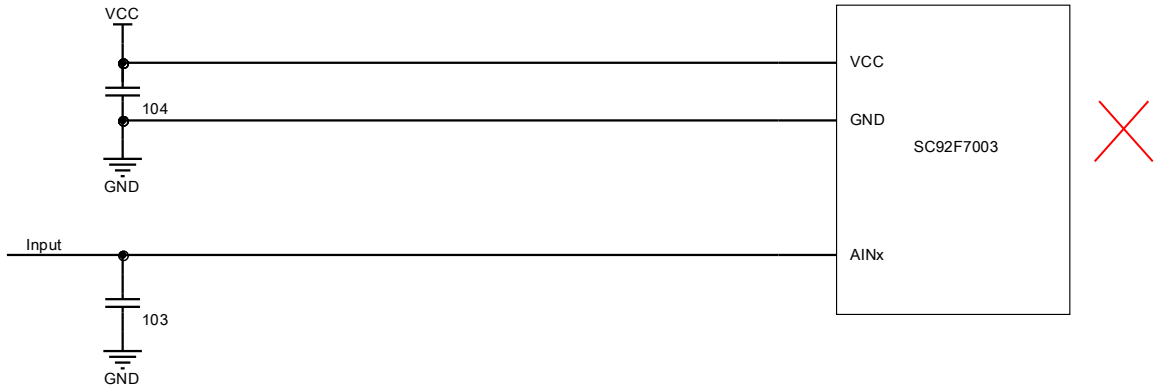
```
unsigned char ADCBUFF[10];
unsigned char ADC_Value0,ADC_Value1,ADC_Value2;
unsigned int Average(unsigned char *ValuePoint,unsigned ArrSize)
{
    unsigned char i;
    unsigned int sum=0;
     unsigned char RemoveCount = 2;          //去掉切换 ADC 口后转换的前两次数据。
    for(i= RemoveCount;i< ArrSize;i++)
    {
        sum += ValuePoint[i];
    }
    return    sum/(Count- RemoveCount;)

}

unsigned int ADC_Change()
{
    ADCCON |= 0X40;                         //开始 ADC 转换
    while(!(ADCCON&0x20));                  //等待 ADC 转换完成
    ADCCON&=~(0X20);                        //清中断标志位
    return ((unsigned int)ADCVH<<4)+(ADCVL>>4); //取得 AD 转换值
}

void ADC_channel(unsigned char channel)
{
      ADCCON &= 0XE0;
     ADCCON |= channel;                         //ADC 输入选择为 ADCchannel 口
```

```
}

void ADC_Multichannel()
{
    unsigned char i;

    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
    for(i=0;i<10;i++)
    {
        ADCBUFF[i]=ADC_Change();       //进行一次 AD 转换，取得转换后的转换值
    }
    ADC_Value0 = Average(ADCBUFF,10);     //平均值滤波取值

    ADC_channel(1);                     //ADC 入口切换至 AIN1 口
    for(i=0;i<10;i++)
    {
        ADCBUFF[i]=ADC_Change();       //进行一次 AD 转换，取得转换后的转换值
    }
    ADC_Value0 = Average(ADCBUFF,10);     //平均值滤波取值

    ADC_channel(2);                    //ADC 入口切换至 AIN2 口
    for(i=0;i<10;i++)
    {
        ADCBUFF[i]=ADC_Change();       //进行一次 AD 转换，取得转换后的转换值
    }
    ADC_Value0 = Average(ADCBUFF,10);     //平均值滤波取值

}
```

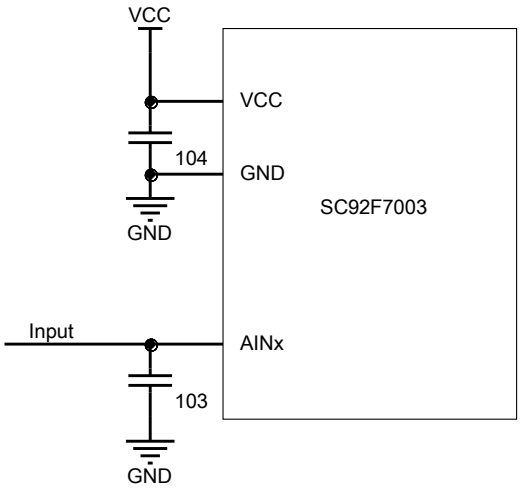
7.4.2 ADC 采样管脚电路

SC92F7003_8003 的 ADC 采样口需要在靠近管脚处加 103 电容，另外 ADC 转换需要以电源电压作参考电压，所以在使用 ADC 功能时，请在靠近 IC 的 VCC 和 GND 处加 104 电容，以保证参考电压稳定，用户设计电路时，需要注意：

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



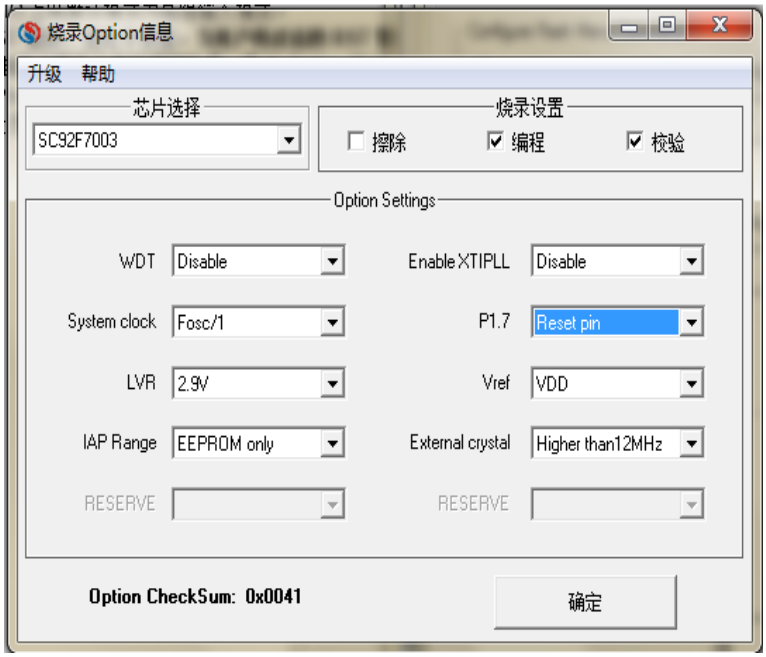
- 建议接法：104 电容靠近电源脚， 103 电容靠近 ADC 采样口。见下图：

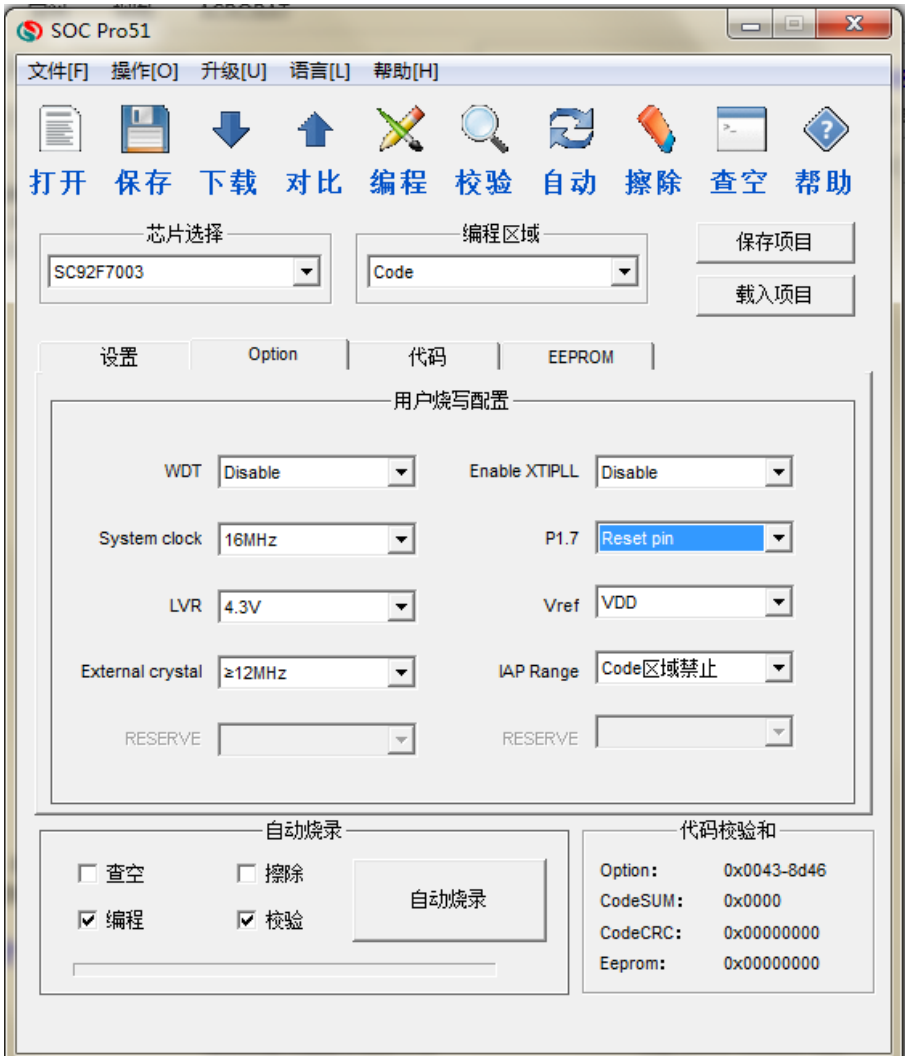


7.5 RST 管脚电路

SC92F7003_8003 的 GPIO 上电默认模式为高阻输入模式。

SC92F7003_8003 的 **RST 管脚，低电平复位，当用户将对应的 RST 管脚的 RST 功能使能（可通过 Opiton 项选择）时，用户设计电路不能在上电时强制拉低**，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 的设置如图（以 SC92F7003 为例）。前为 KEIL 的设置情况，后为烧录上位机的设置情况。





7.6 OPTION 相关 SFR 操作说明

SC92F7003_8003 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 Customer 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。

OPINX 值	OPREG										上电初始值
	符号	说明	7	6	5	4	3	2	1	0	
83H@FFH	OP_HRCR	高频 RC 震荡频率调节(细调)	OP_HRCR[7:0]								nnnnnnnnb
C1H@FFH	OP_CTM0	Customer Option 寄存器 0	ENWDT	ENXTL	SCLKS[1:0]		DISRST	DISLVR	LVRS[1:0]		nnnnnnnnb
C2H@FFH	OP_CTM1	Customer Option 寄存器 1	VREFS	XTLHF	-	-	IAPS[1:0]		-	-	nnxxnnxxb

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！

7.7 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！

7.8 外部中断设置注意事项

赛元 SC92F7003_8003 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
     }

 }
```

7.9 SSI 设置注意事项

使用 SC92F7003_8003 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 SC92F7003_8003 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;
}
```

规格更改记录

版本	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	2018 年 12 月
V1.3	ADC 示例修改，UART0 设置注意事项修改	2018 年 10 月
V1.2	增加 7.1 节 PWM 所在 IO 口初始化为输入带上拉模式 增加 7.3 节 UART0 使用注意事项 增加 7.9 节 SSI 设置注意事项	2018 年 7 月
V1.1	增加 SC92F8003 相关说明	2018 年 6 月
V1.0	初版	2018 年 5 月