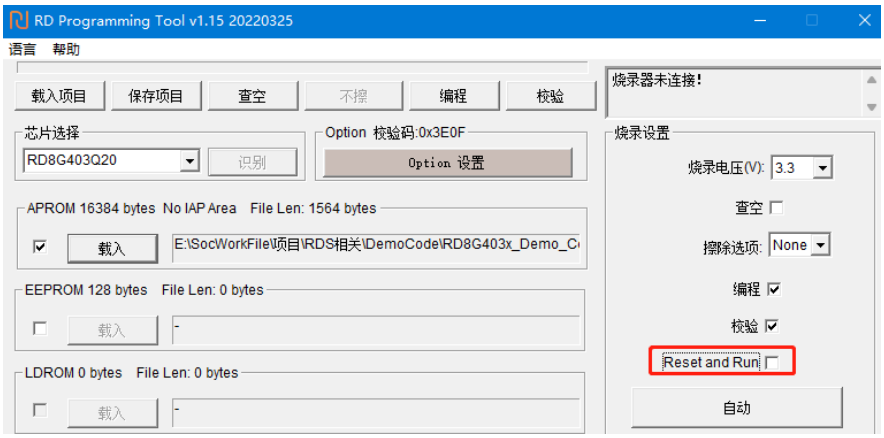
目录

1 新定义 RD8G403 系列 MCU 电气参数注意项

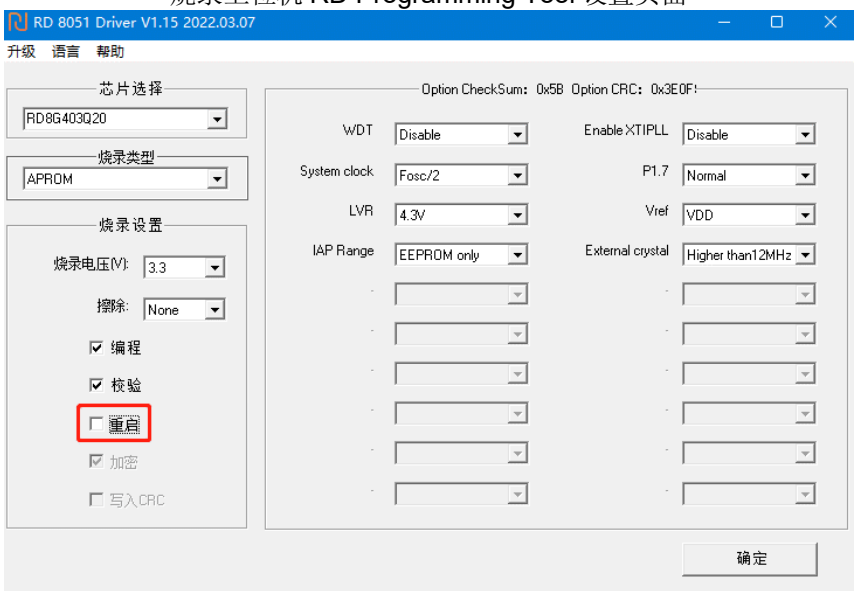
宽电压工作范围：2.4V~5.5V
工作温度：-40 ~ 85℃
内核：超高速 1T 8051
Flash ROM：16 Kbytes Flash ROM（MOVC 禁止寻址 0000H~00FFH）可重复写入 1 万次
EEPROM：独立的 128 bytes，可重复写入 10 万次，10 年以上保存寿命
系统时钟：内建高频振荡器频率误差： 跨越 (2.9V~5.5V) 及 (-20 ~ 85℃) 应用环境，不超过 ±1%

2 新定义 RD8G403 系列 MCU 烧写注意事项

1. 新定义 RD8G403 系列芯片的 CLK 或 DIO 管脚对 GND 的电容不得超过 100pF，VDD 对 GND 的电容不可超过 1000uF。
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 Pro 和 SC LINK 的烧录排线最长不可超过 60cm。
5. RD8G403 系列芯片脱机烧录时，如果程序上电有 IAP 写的操作，则会校验失败，原因为 RD8G403 系列脱机校验位 CRC 校验，程序跑起来导致 ROM 数据改变，从而校验失败，解决方法两种：
   （1） 在初始化之前加一段时间(建议 100ms)的延时才能烧录成功。
   （2） 在烧录上位机和 Keil 插件的配置页面中，取消勾选“Reset and Run” 或者在 KEIL 设置界面取消勾选“重启”：



烧录上位机 RD Programming Tool 设置页面



KEIL 插件的设置页面

3 新定义 RD8G403 系列 MCU 有关 MOVC 指令的应用注意

新定义 RD8G403 系列 MCU Flash ROM 的起始 256B ROM 区间，即 0x0000-0x00FF，禁止 MOVC 寻址。因此，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，即对 Flash ROM 进行访问。

1. 全局变量的初始化
2. 不可变类型数量（code 类型数据）
3. 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

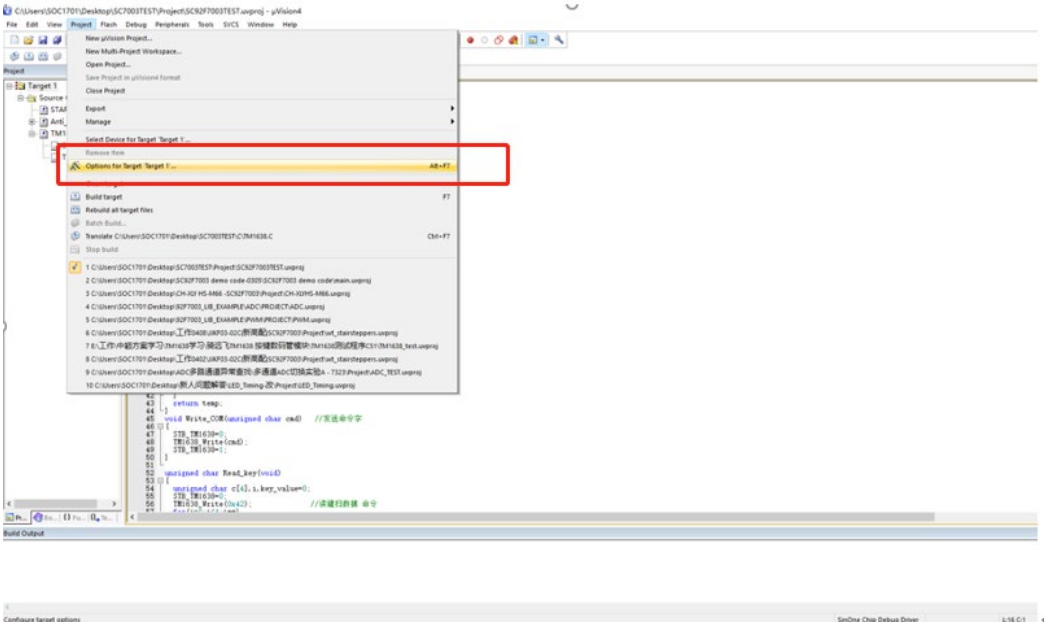
注意：?C?LIB_CODE 标识符只是表明某个函数调用的库文件进行查表运算，通常情况下，客户开发产品不需用调用库文件 Math 函数。（库文件占用较大的 ROM 空间,譬如 sinx 函数）。

.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

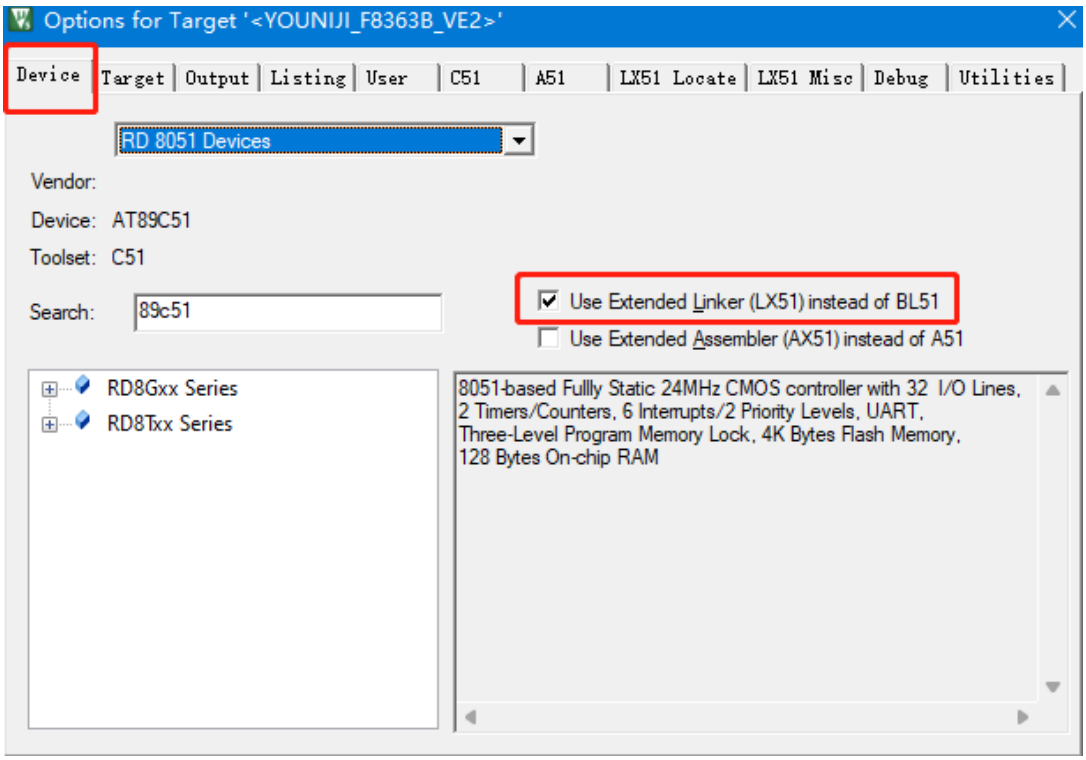
根据 3.1.1 的描述可知，用户在采用 C 语言开发过程当中，需要把全局变量，不可变类型数据（code 类型数据）定义在 Flash ROM 起始的 256B 地址之后。因此，在开发调试时，可以先采用屏蔽该区域的 Flash ROM 来进行开发，待调试完毕后，再做调整，生成最终的程序。使用不同的链接器操作方法不同，具体方法见下：

在 LX51 链接器上实现调整的方法（推荐方法）：



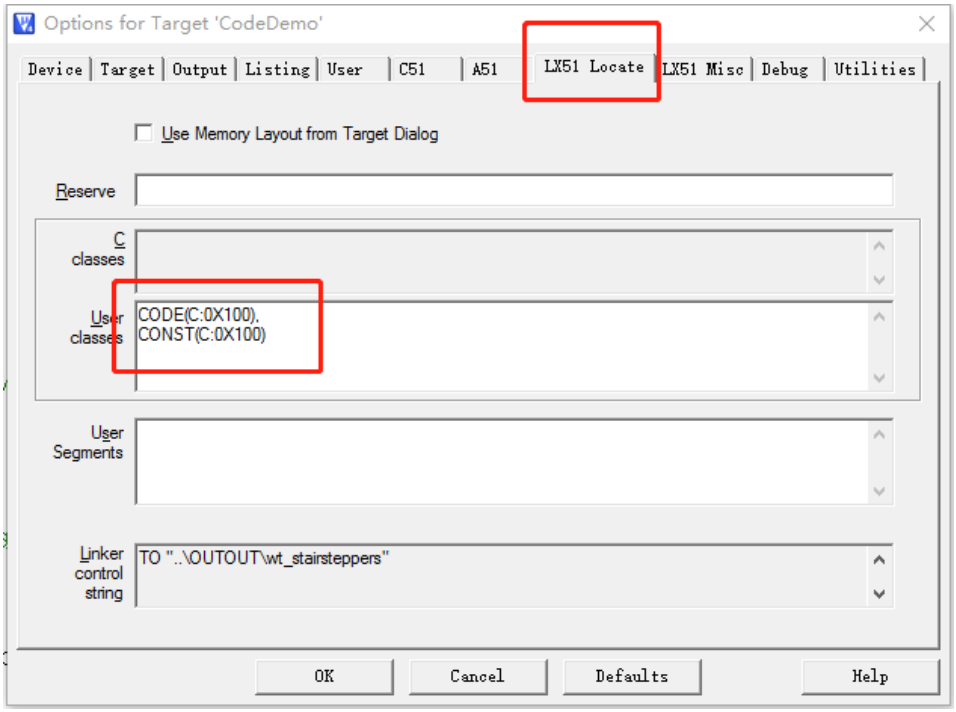
- **选择使用 LX51 链接器。**

  1. 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。**

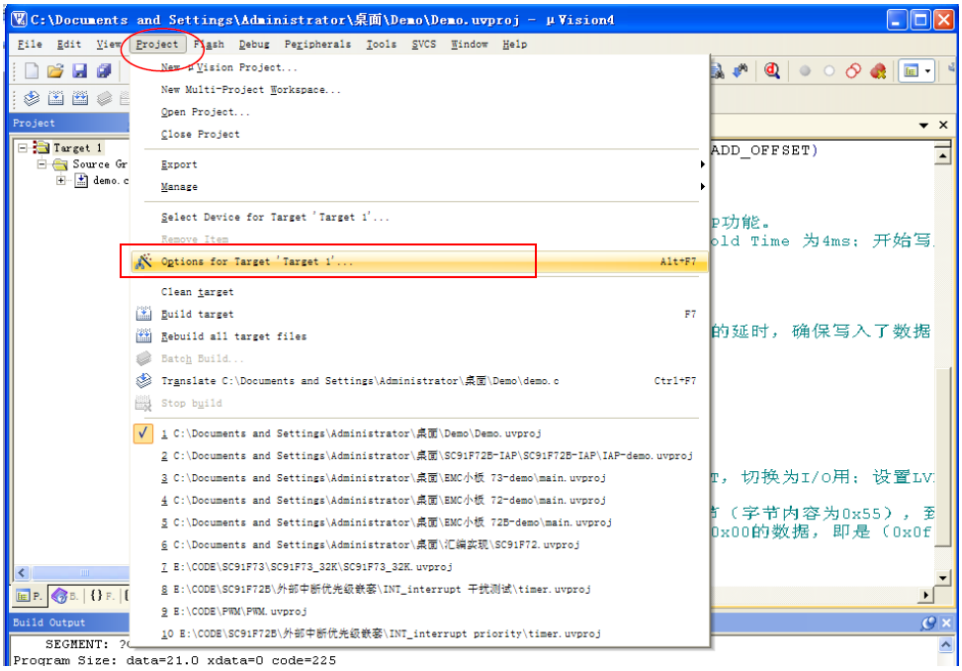
打开项目选项中的“LX51 Locate”属性页，在“User classes”的输入框中填入“CODE(C:0X100), CONST(C:0X100)”指令，点击 OK 完成配置，见下图：

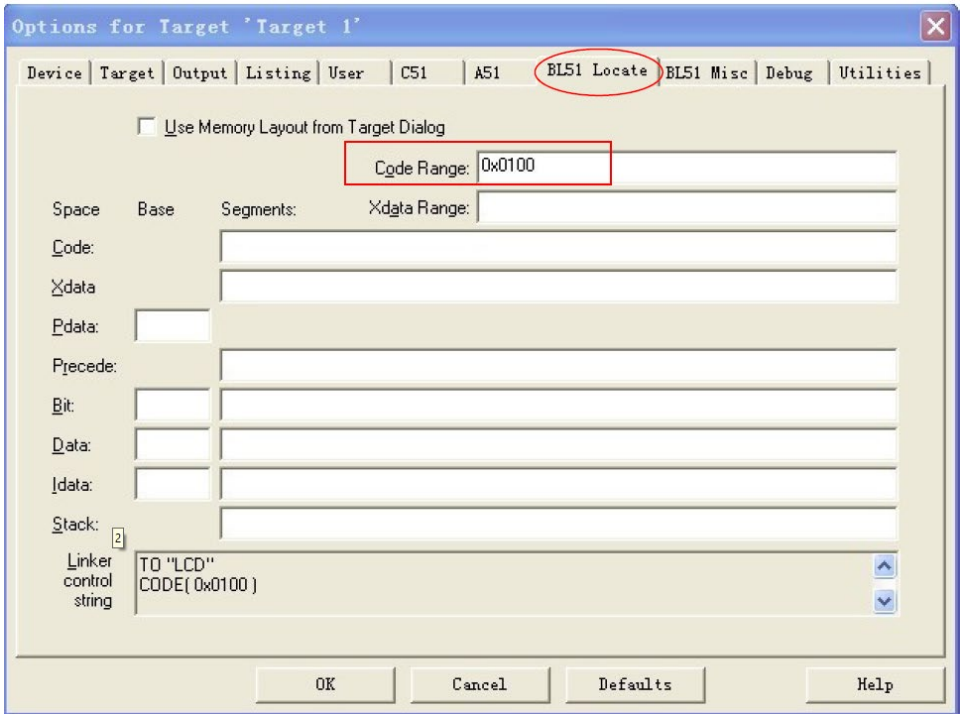


在 BL51 链接器上实现调整的方法：

- **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：



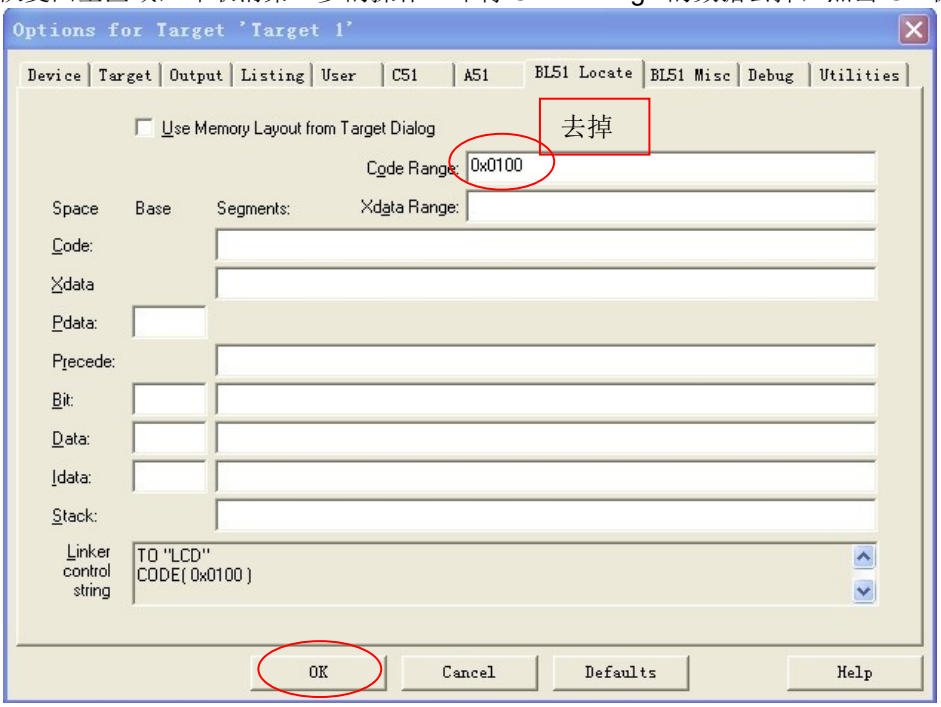


- **调试完成后，生成最终程序；**

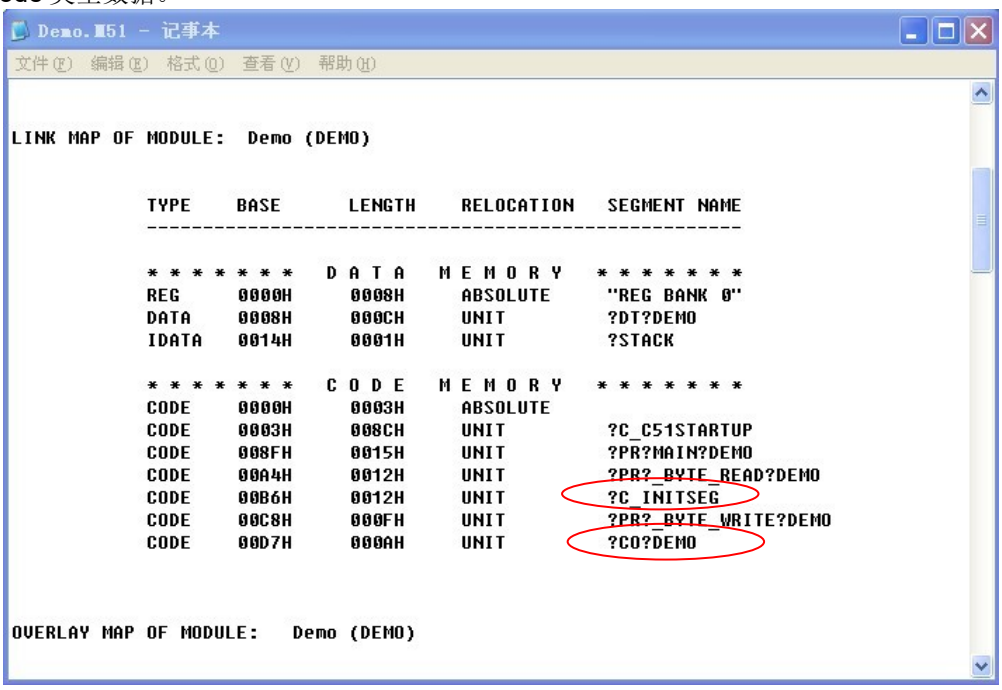
当用户需要在 0X100 前存储代码时，由于 0x100 前的数据不可进行读操作，所以不可在此区域存储全局变量、不可变类型数量（code 类型数据），若编程代码中存在 code 类型的全局变量，需要将这些数据类型存放到 0x100 地址之后。

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。



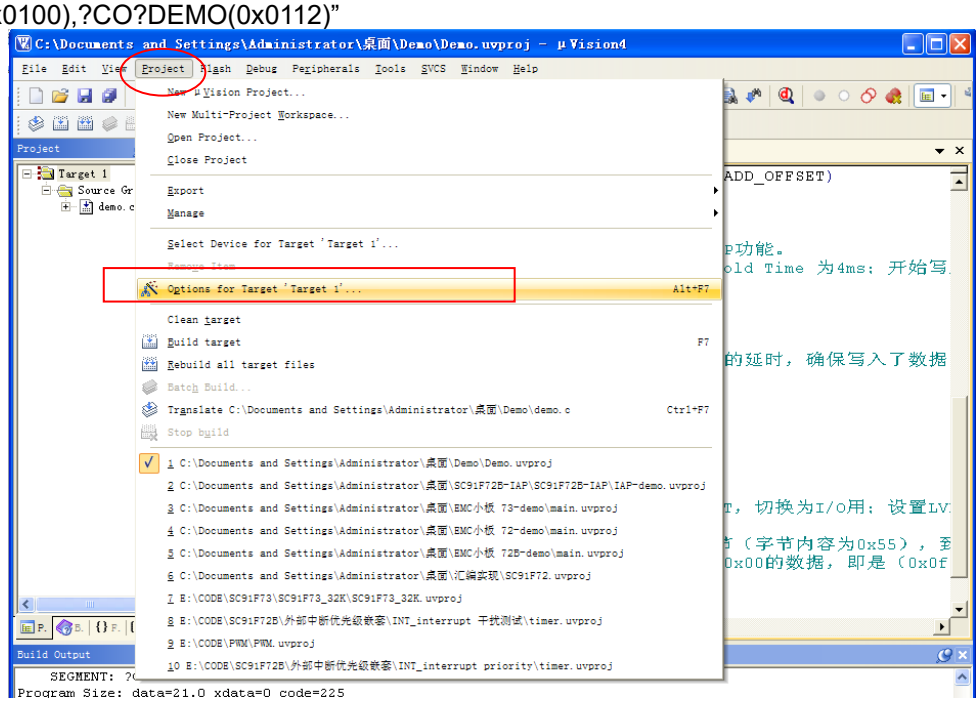
2）重新编译后，在建立的工程目录下，找到并打开 .M51 文件，在 CODE MEMORY 会出现：
“?C_INITSEG” :全局变量初始化数据。
“?CO?DEMO” :code 类型数据。

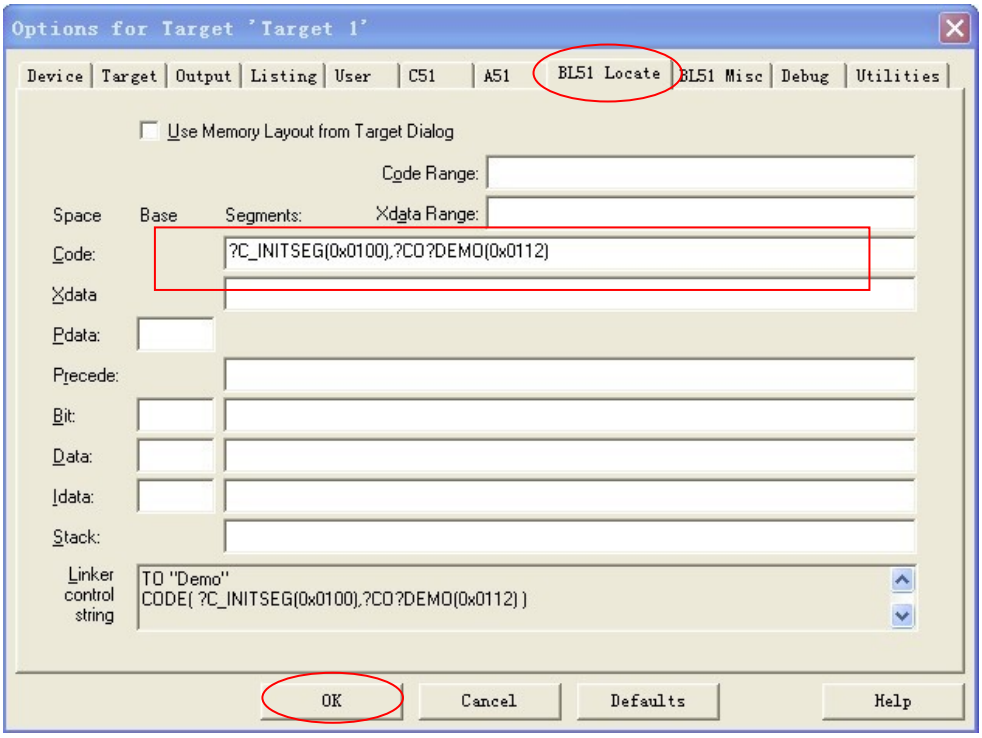


说明：从以上 M51 文件的“CODE MEMORY”信息中，可以看到“?C_INITSEG”，链接地址为 00B6H，长度为 0012H 字节；“?CO?DEMO”，链接地址为 00D7H，长度为 000AH 字节。

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。**

新定义 RD8G403 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。

```
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  ;------------------------------------------------------------------------------
```

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

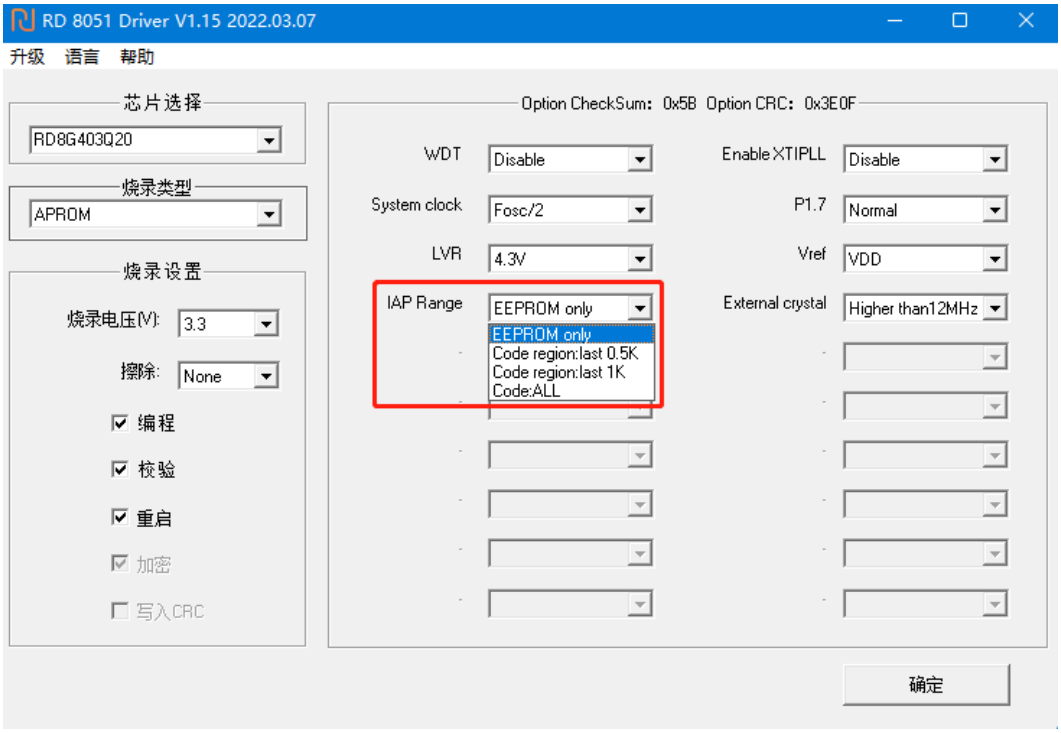
4 新定义 RD8G403 系列 MCU 的 EEPROM 及算法解说

4.1 内部 EEPROM 的操作及 CODE 的 IAP 操作注意事项

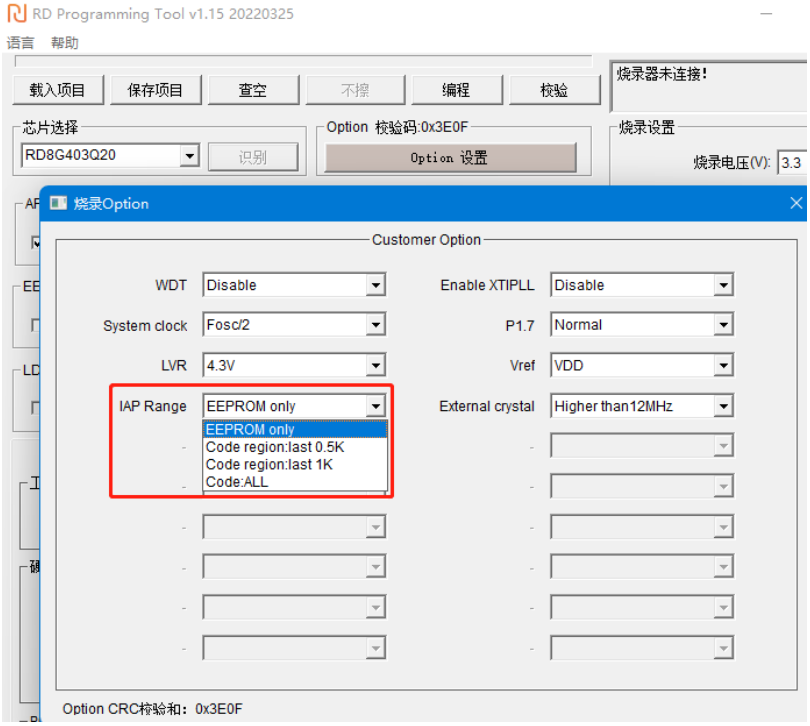
以下是新定义 RD8G403MCU 内部 EEPROM 的使用方法及 CODE 区 IAP 操作的方法。

4 新定义 RD8G403 系列 MCU 内部有 16K 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：**

1. By Byte 操作。即一个字节一个字节写入，读取。
2. 类 RAM 读写：写前不需擦除。

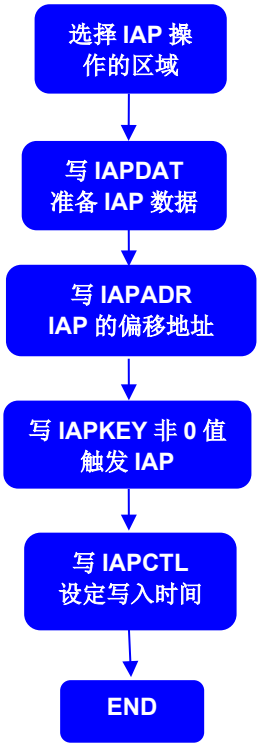
- **EEPROM 的寿命：10 万次以上。**
- **建议用户擦写不要超过 EEPROM 的额定烧写次数，否则会出现异常！**
- **超过寿命再执行 IAP 写操作的话 CPU Hold Time 时间将变为无限长，无法退出 IAP 模式，此时即使 WDT 开启也无法复位！**
- **FLASH 读写特点：**

1. By Byte 操作。即一个字节一个字节写入，读取。
2. 类 RAM 读写，写前不需擦除。

- **FLASH 的寿命：**可重复写入 1 万次。
- **建议用户擦写不要超过 FLASH 的额定烧写次数，否则会出现异常！**
- **超过寿命再执行 IAP 写操作的话 CPU Hold Time 时间将变为无限长，无法退出 IAP 模式，此时即使 WDT 开启也无法复位！**
- **新定义 RD8G403 系列 MCU 越界 IAP 写或擦除操作 LDROM 区域会导致芯片损坏，请勿越界操作 LDROM！**

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

新定义 RD8G403 系列 MCU 在进行 IAP 操作的过程中不允许响应中断，因此，在进行相关操作时，需要先把总中断关闭，即 EA=0；待完成 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：

```
EA = 0;                                  //关总中断
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_();
EA = 1;                         //开总中断
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
```

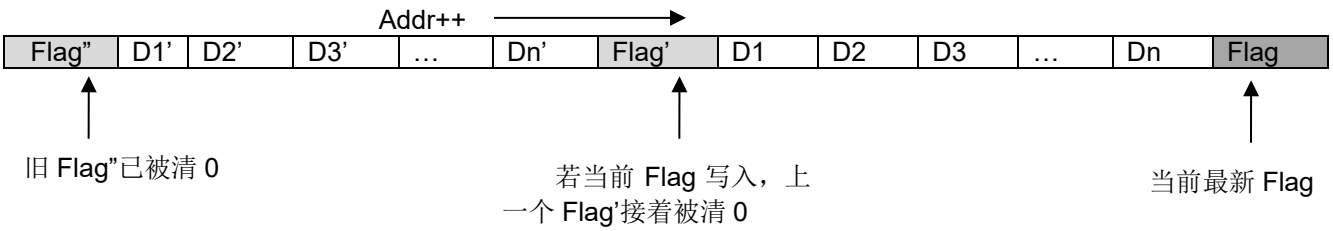
注意：Code 区域内的 IAP 操作有一定的风险，需要用户在软件中做相应的安全处理措施，**如果操作不当可能会造成用户程序被改写！**除非用户必需此功能(比如用于远程程序更新等)，否则不建议用户使用。

4.3 EEPROM 的使用算法

由于 RD8G403 系列 MCU 有 16kB 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：

1. 见以下流程图：

2. 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 是使用上了新定义 RD8G403 系列 MCU 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 " RD8G403x_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 RD 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)
{
  EA = 0;                                            //关总中断
  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。
  EA = 1;                                 //开总中断
}
/************************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
```

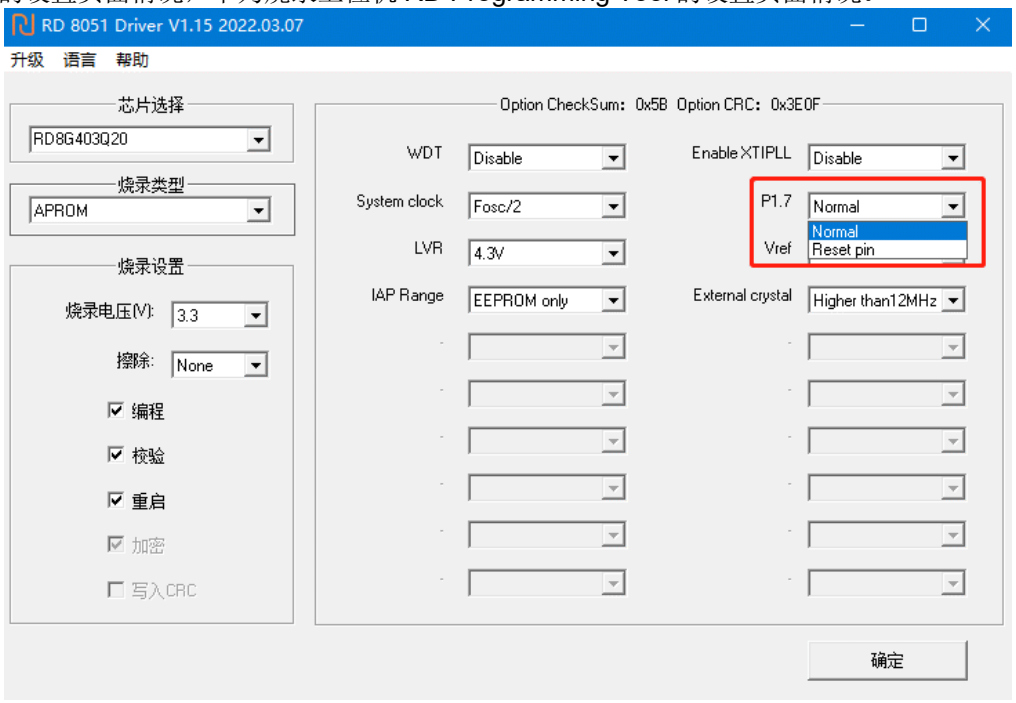
```
    if(Byte_Read(offset) == 255)
      ADD_OFFSET = offset;

  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);
}
```

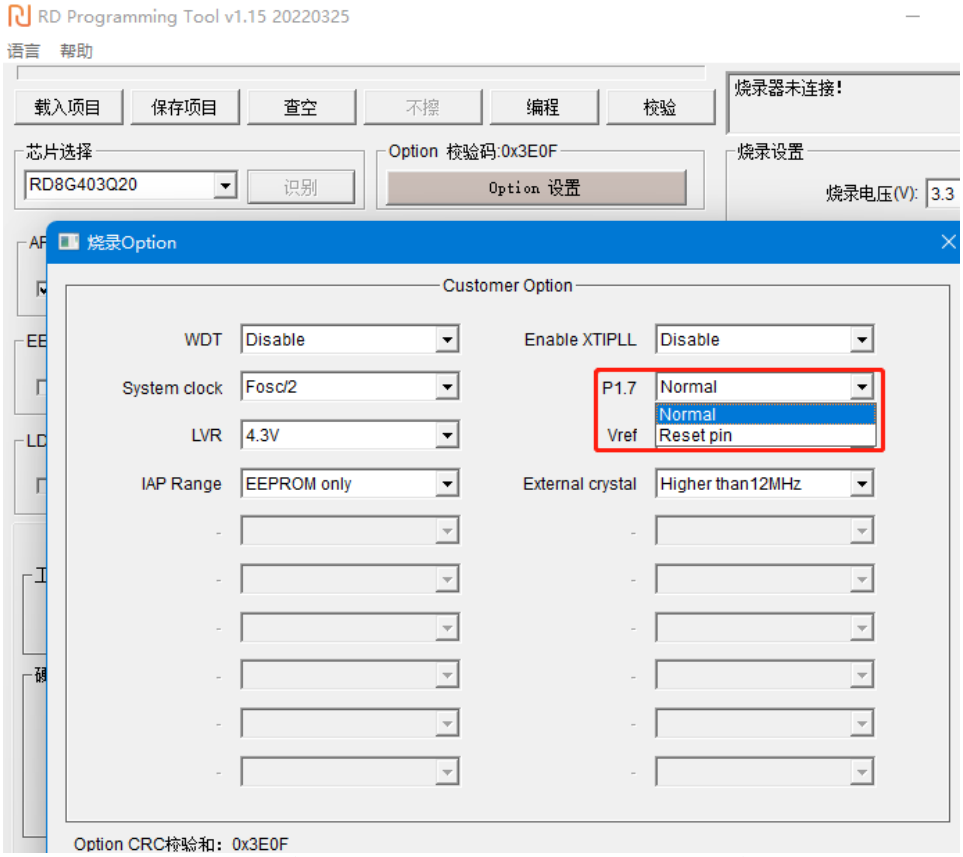
5 电路设计的注意事项

新定义 RD8G403 系列 MCU 的 GPIO 上电默认模式为高阻输入模式。

新定义 RD8G403 系列 MCU 的 RST 管脚，通过 Option 选项可设置为 Reset pin 或 GPIO。RST 管脚作为 Reset pin 时，低电平使能；作为 GPIO 时，管脚低电平不会产生复位。Option 的设置如图（以 RD8G05x 为例）。上为 KEIL 插件的设置页面情况，下为烧录上位机 RD Programming Tool 的设置页面情况。



KEIL 插件的设置页面情况



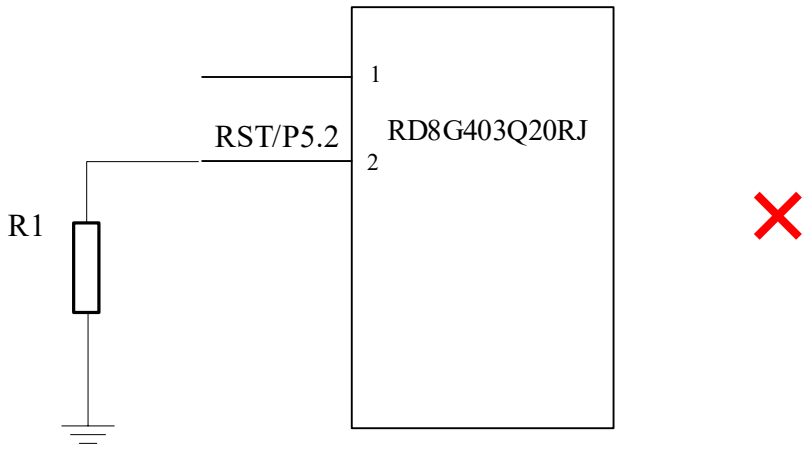
烧录上位机 RD Programming Tool 的设置页面情况

5.1 电路设计实例

5.1.1 RST 管脚电路

新定义 RD8G403 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。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：

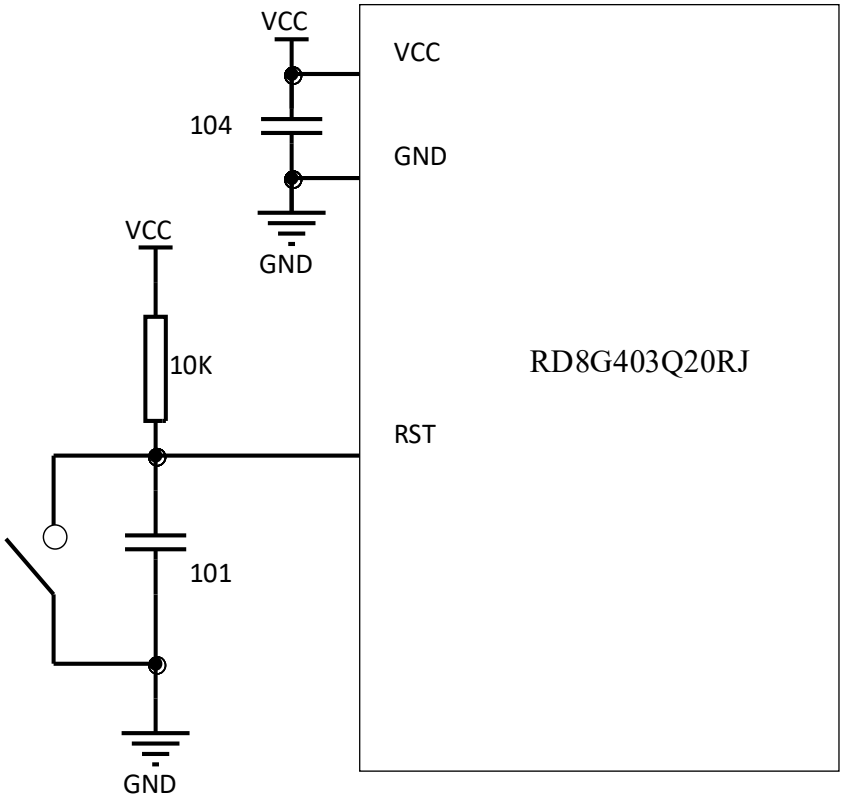
错误接法：



错误接法图示

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：

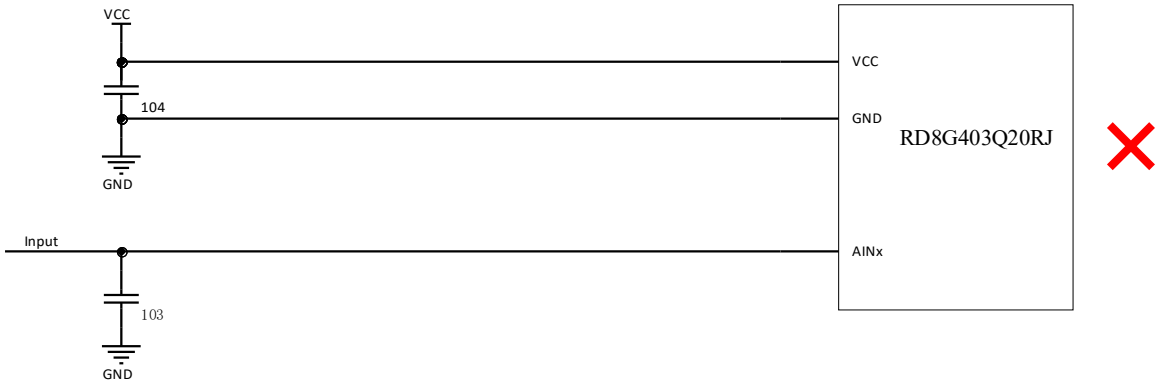


建议接法图示

5.1.2 ADC 采样管脚电路

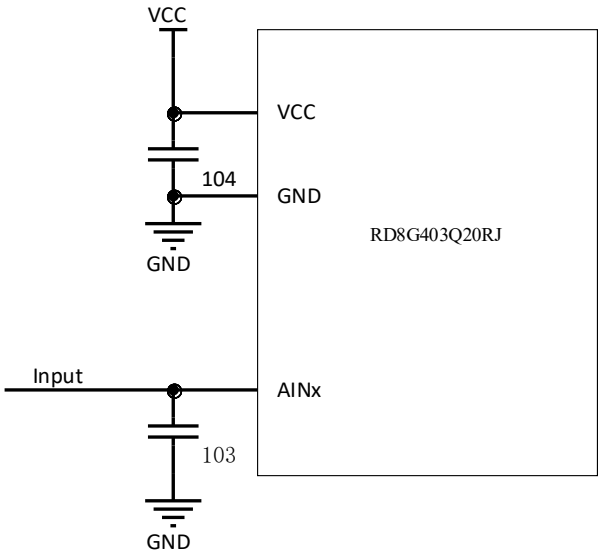
新定义 RD8G403 系列 MCU 的 ADC 采样口需要在靠近管脚处加 103 电容， ADC 转换需要让电源电压稳定，所以在使用 ADC 功能时，请在靠近 IC 的 VCC 和 GND 处加 104 电容，以保证转换结果准确。

错误接法：104 电容离电源脚太远， 103 电容离 ADC 采样口太远。



错误接法图示

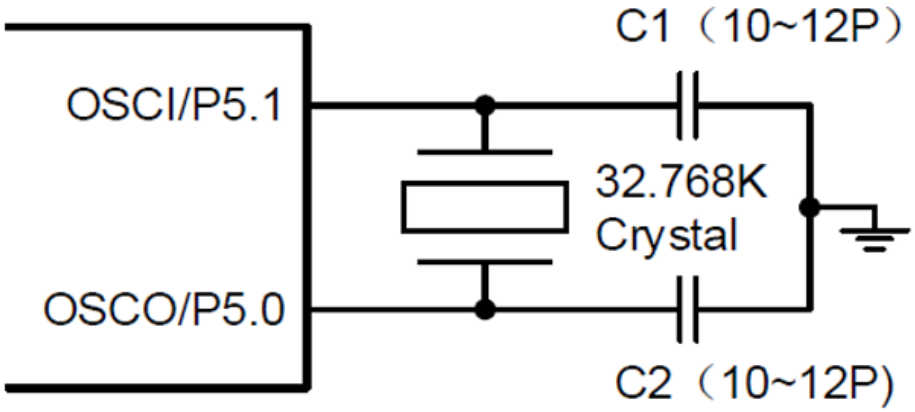
建议接法：104 电容靠近电源脚， 103 电容靠近 ADC 采样口。



建议接法图示

5.1.3 外部晶振电路

新定义 RD8G403 系列 MCU 提供了低频外部接口，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：



32.768K 外部晶振连接图

5.2 IO 口各模式设置注意事项

新定义 RD8G403 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

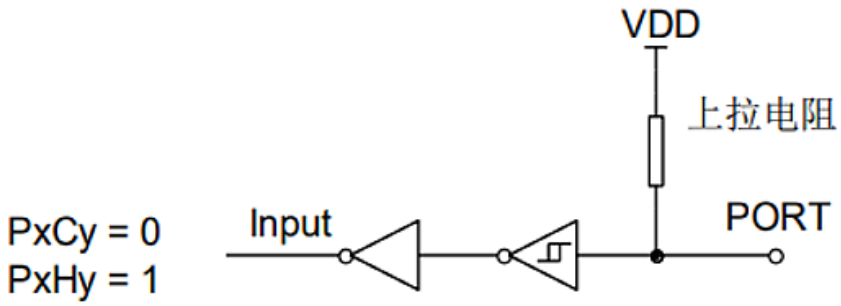
注意：未使用及封装未引出的 IO 口均要设置为强推挽输出模式。

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，用户可以从新定义 MCU 体系中按需选择。

5.2.2 带上拉输入模式

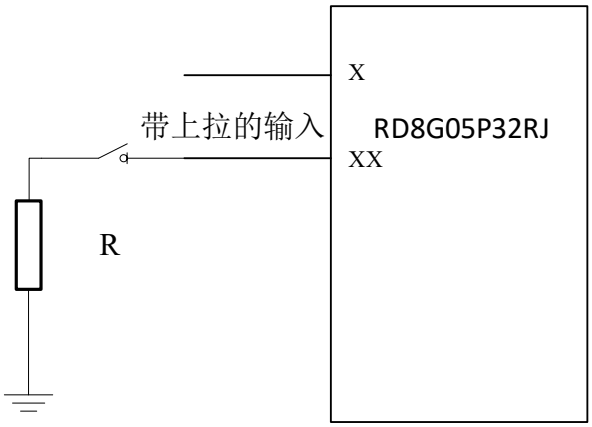
带上拉的输入模式下，输入口恒定接一个上拉电阻，仅当输入口上拉电平被拉低时，才会检测到低电平信号。带上拉的输入模式的端口结构示意图如下：



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 10K。



上拉模式的按键输入接法图示

5.2.4 I/O 开漏输出模式的实现方式

新定义 RD8G403 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。

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
```

5.2.5 I/O READ IO 功能注意事项

新定义 RD8G403 系列 MCU 具有 Read IO 功能，Read IO 功能是默认开启的，即使 GPIO 处于输出模式，读端口数据寄存器，也会以 IO 管脚上的电平为依据返回高低值；若把 Read IO 功能关闭则输出模式下读端口数据寄存器则不会以 IO 管脚上的电平为依据，读取的是寄存器的值。输入模式不受影响。

代码示例如下：

```
OPINX = 0X86;      //关闭 Read IO 功能
OPREG&=0XF7 ;
OPINX = 0X86;      //开启 Read IO 功能
OPREG|=0X08 ;
```

6 软件编写的注意事项

新定义 RD8G403 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 口输出由 IO 口相关配置寄存器（PxCON、PxPH）的值决定）。

新定义 RD8G403 系列 MCU 的 PWM 精度为 10 位，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，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

新定义 RD8G403 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 " RD8G37x_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 CheckSum 设置注意事项

新定义 RD8G403 系列 MCU 内建了 Check Sum 功能，可用来实时生成程序代码的 16 位 Check Sum 值，用户可利用此 Check Sum 和理论值比较，监测程序区的内容是否正确。操作前需要先关闭 EA，执行后再打开 EA,同时在操作 OPERCON 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 " RD8G403X_C.H"
#include "intrins.H"
EA = 0;                        //关总中断
OPERCON|= 0X01;                //执行 check sum 运算后需接 8 个 NOP
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
EA =1;                         //开总中断
```

6.4 UART0 设置及使用注意事项

新定义 RD8G403 系列 MCU 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 |= 0x20;      //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; //选择定时器 1 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
```

新定义 RD8G403 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。
```

6.5 SPI/TWI/UART 三选一通用串行接口 SSI 设置及使用注意事项

使用新定义 RD8G403 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。

新定义 RD8G403 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，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;
}
```

SSI 的 UART 发送中断标志 TI 和接收中断标志 RI 在同一个寄存器上，此寄存器不能位操作，所以在清除 TI 和 RI 时会对整个寄存器进行与操作，这样当 UART 进行全双工通信时发送和接收中断有可能同时产生，或者是两者之间间隔时间很短，会出现 TI 或者 RI 被误清除导致中断丢失的风险，因此在全双工通信应用场景中，通信需要有容错机制，不能因为某次中断丢失就导致通讯崩溃，在发送数据后不能通过死等发送标志来判断发送是否完成，需要加入超时监测保证在一段时间后可以退出等待，如下是建议的写法：

```
SSDAT =0x55；//将发送数据 0X55 推入发送缓存
i=0x8000;  //用做超时处理，通过改变这个变量改变超时时间长度，用户可根据波特率进行调整
while(!SSI0SendFlag)
{
i--;
if(i==0)
{
    break;          //超时退出
}
}
SSI0SendFlag = 0;  //清除发送标志。
```

6.6 多通道切换采集注意事项

新定义 RD8G403 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
    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。
  ADCCFG2 = 0x10;                 //ADCCFG 配置出错会影响 ADC 采集精度
  ADCCON  |= 0X80;                //开启 ADC 模块电源
  ADC_channel(0);                 //ADC 入口切换至 AIN0 口
  ADC_Value0 = 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
}
```

6.7 使用定时器时外部中断 0/1 服务函数编写注意事项

当用户程序在初始化完外部中断 0/1 后，若后续程序中有操作到 TCON 的 TR1、TR0、TF1、TF0 位的场合，需要在外部中断 0/1 服务程序内手动清除外部中断标志，否则可能会导致外部中断标志位无法硬件清除。

使用示例如下：

```
void EX0() interrupt 0
{
   TCON &= 0xFD;
}

void EX1() interrupt 2
{
   TCON &= 0XF7;
}
```

6.8 外部中断设置注意事项

新定义 RD8G403 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！IO 口需要先设置，再设置相应的外部中断配置。反过来操作有可能会误产生一次边沿中断。

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：

```
P4CON &= 0XFC;  //将 INT10（P40）口和 INT11(P41)设置为输入模式
P4PH |= 0X03;   //打开 P40 和 P41 的上拉电阻
INT1F = 0X03;   //使能 INT10、INT11 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(P40==0)      //判断外部中断是否来自于 INT10
    {
                //执行代码
    }
    if(P41==0)  //判断外部中断是否来自于 INT11
    {
                //执行代码
    }
}
```

6.9 软件操作 CODE OPTION 的注意事项

新定义 RD8G403 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：

```
EA = 0;                 //关总中断
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）
EA=1;                   //开总中断
```

汇编例程：

```
CPL EA;
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！

```
SETB EA;
```

6.10 软件安全加密功能注意事项

新定义 RD8G403 系列芯片允许用户是否开启安全加密功能，该功能需要配合 RD LINK PRO 以及 RD Programming Tool 上位机软件使用，具体的操作说明见《新定义 LINK 系列量产开发工具使用手册》安全加密章节，可到新定义网站下载。

6.11 中断关闭注意事项

在实际应用中开启子中断后一般不需要再次关闭子中断，如果用户需要关闭子中断，必须在关闭子中断前先将总中断关闭，然后再关闭子中断，如下是子中断关闭步骤，请用户务必按照此步骤进行子中断的关闭

```
EA = 0;
子中断开关=0;(如 ET0 = 0;)
EA = 1;
```

规格更改记录

版本	记录	日期
V1.0	初版	2022 年 6 月

声明

合肥市新定义电子有限公司（以下简称 RD）保留随时对 RD 产品、文档或服务进行变更、更正、增强、修改和改进的权利，恕不另行通知。RD 认为提供的信息是准确可信的。本文档信息于 2022 年 6 月开始使用。在实际进行生产设计时，请参阅各产品最新的数据手册等相关资料。