目录

1 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。

1.1 C 语言编程有关 MOVC 指令的应用注意

1.1.1 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

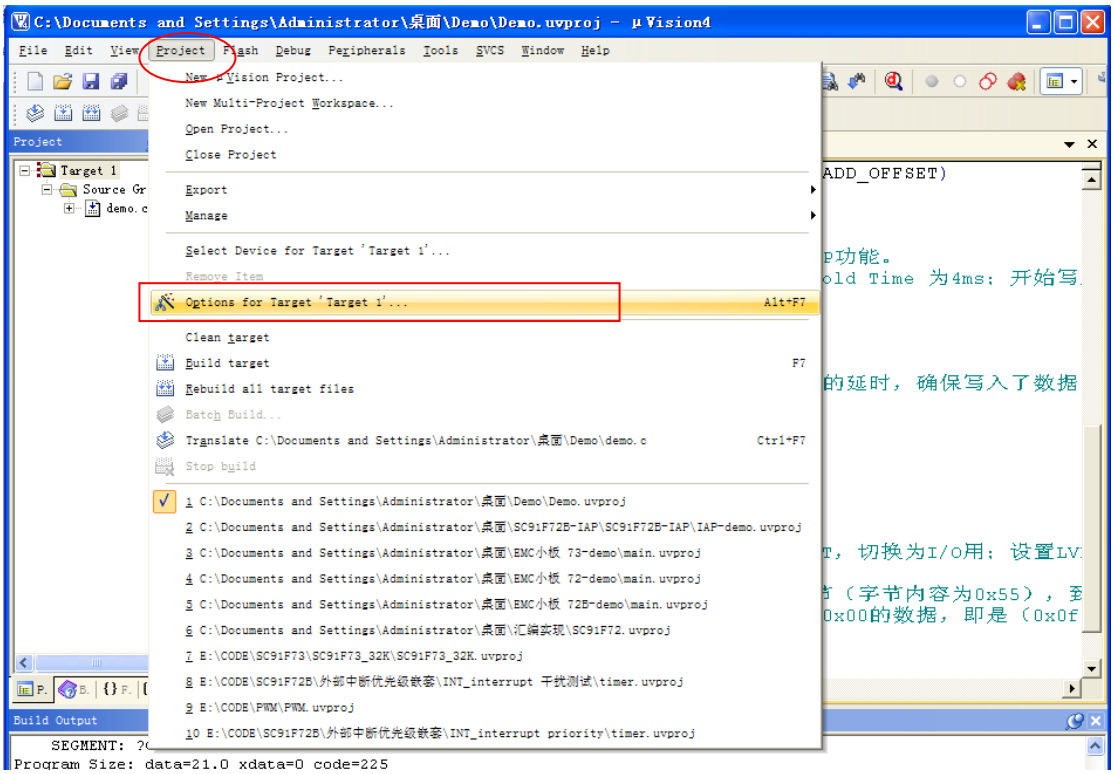
注意：?C?LIB_CODE 标识符只是表明某个函数调用的库文件进行查表运算，通常情况下，客户开发产品不需用调用库文件 Math 函数。（库文件占用较大的 ROM 空间,譬如 sinx 函数）。

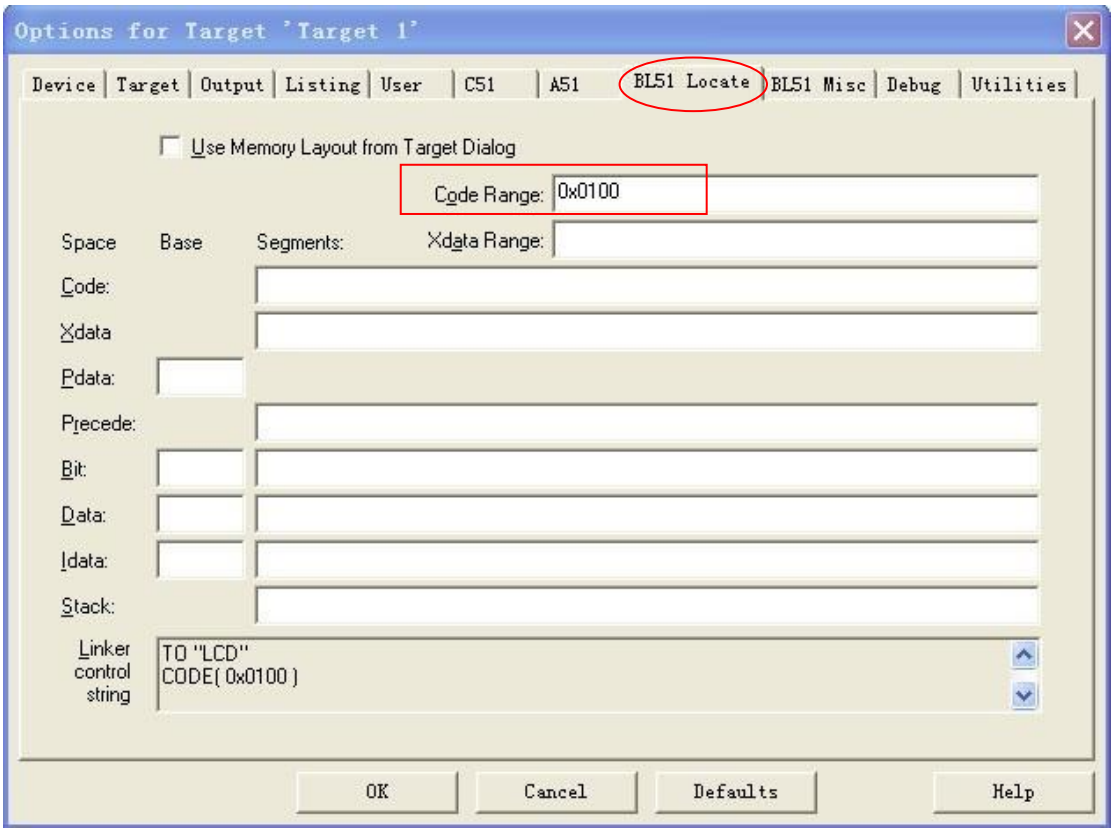
.M51 文件详细记录了上表中各代码段的使用情况，包括 Code 的起始地址、长度等。用户只需要查看?C_INITSEG、?CO?Project_name、调用库文件的函数（如果有?C?LIB_CODE 的话）的起始地址是否在禁止访问区，如果在禁止访问区，可参考后续操作改变起始地址。

1.1.2 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**。具体方法见下：

- **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：



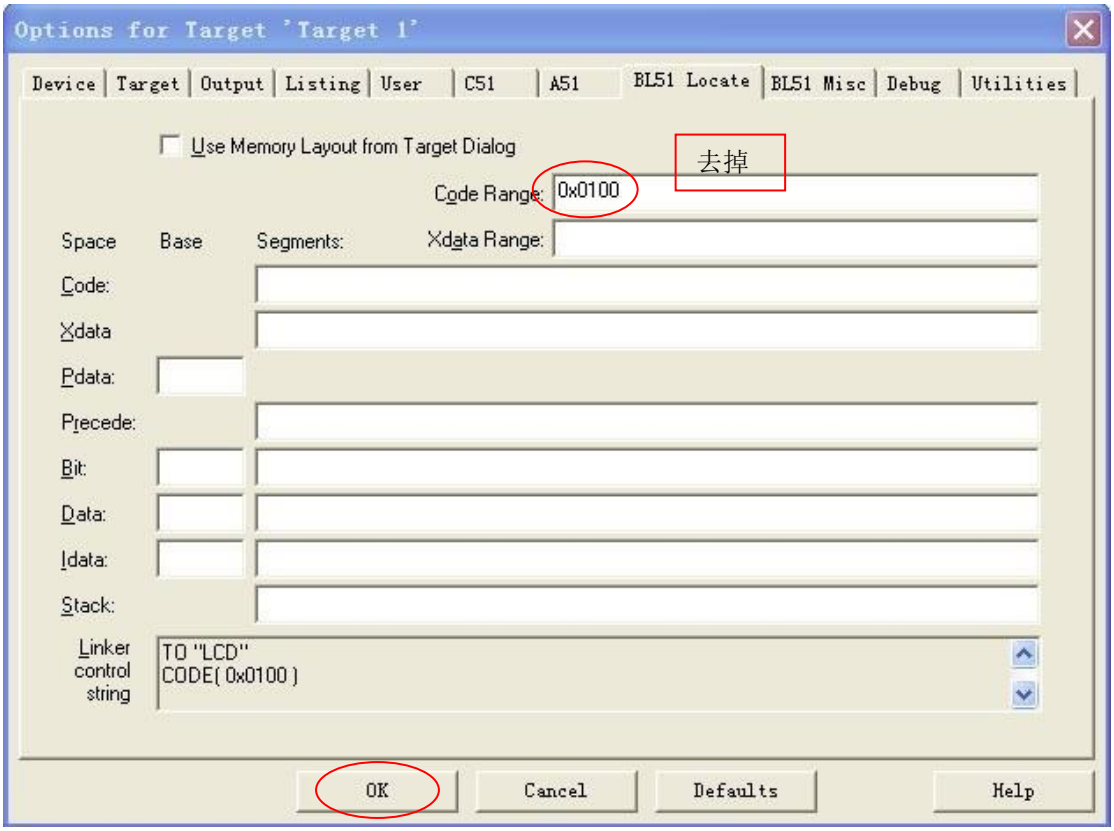


◆ **调试完成后，生成最终程序；**

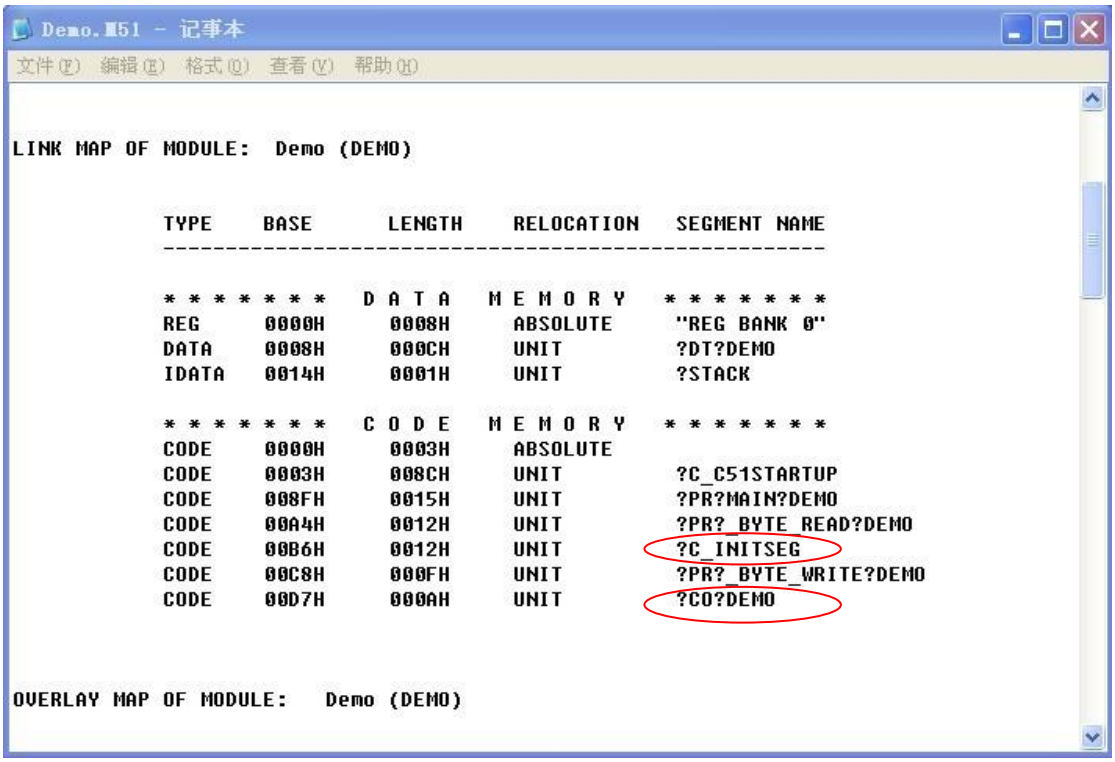
若编程代码中存在全局变量，code 类型数据（不可变），需要将这些数据类型存放到 0x0100 地址之后。

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。



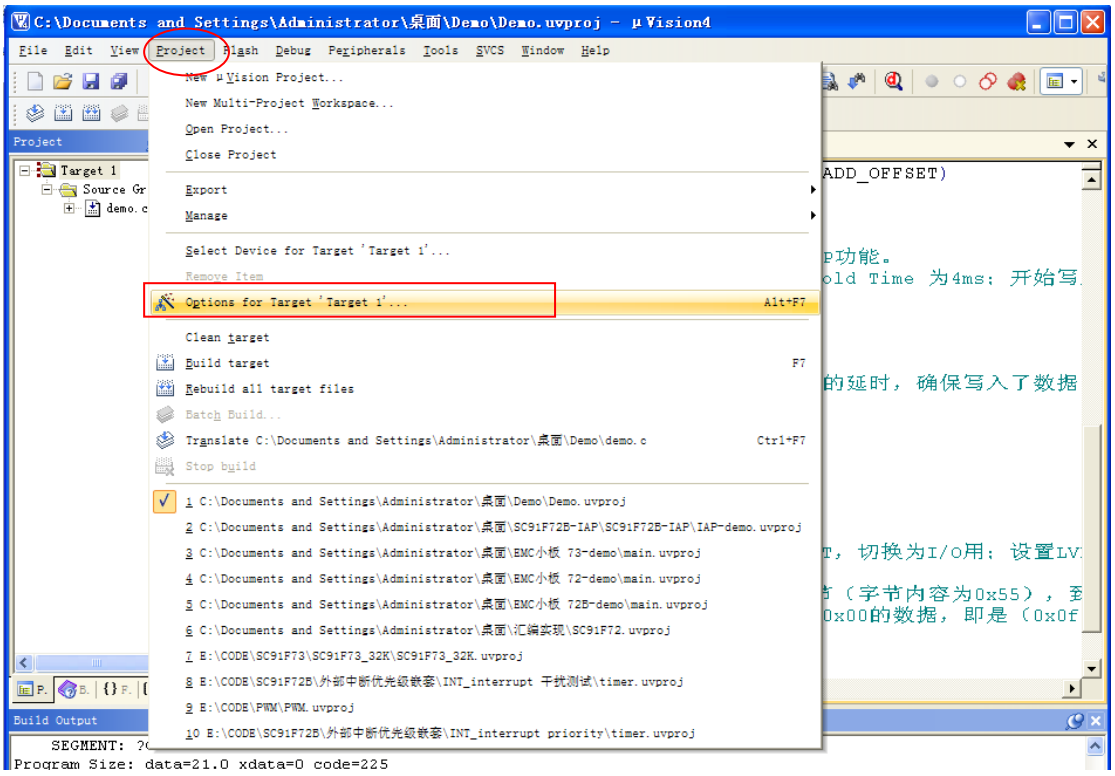
② 重新编译后，在建立的工程目录下，找到并打开 .M51 文件，在 CODE MEMORY 会出现：
“?C_INITSEG” :全局变量初始化数据。
“?CO?DEMO” :code 类型数据。

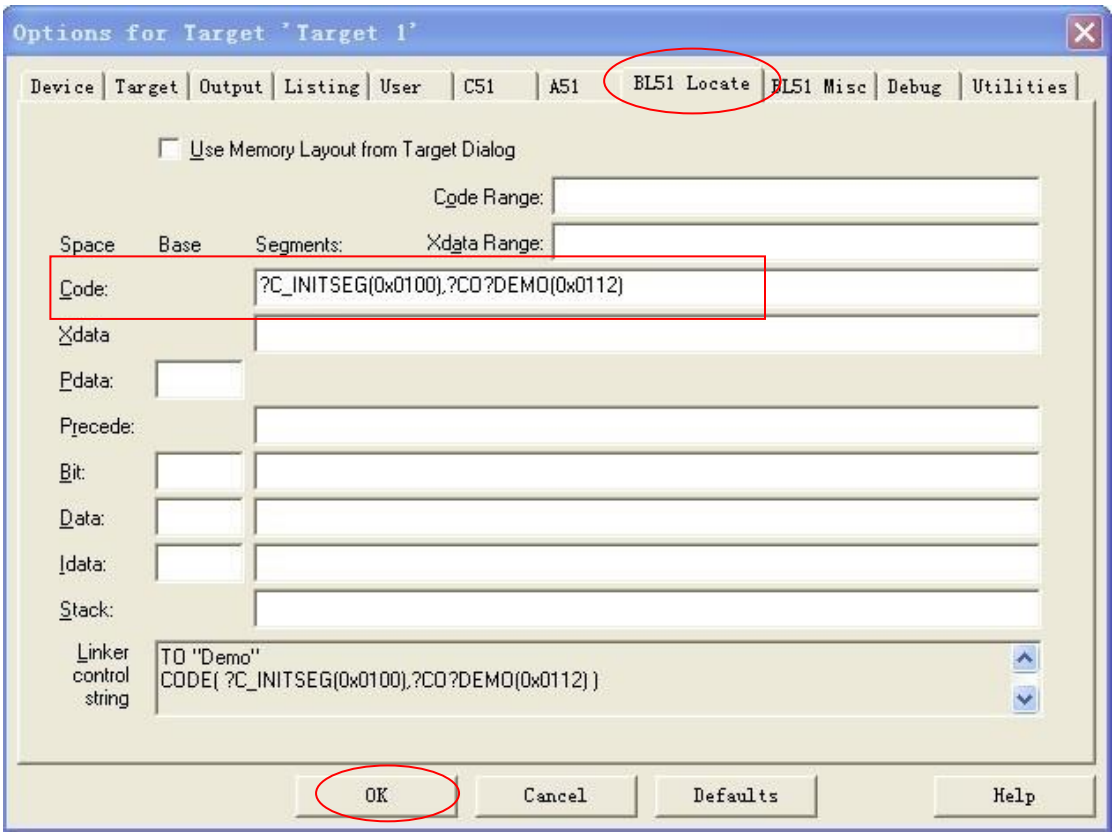


说明:从以上 M51 文件的“CODE MEMORY”信息中，可以看到“?C_INITSEG”，链接地址为 00B6H，长度为 0012H 字节；“?CO?DEMO”，链接地址为 00D7H，长度为 000AH 字节。

③ 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

④ 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)”





⑤ 点击 OK 按钮，并重新编译即可生成了最终程序。

1.2 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。

2 赛元 MCU 的 EEPROM，及算法解说

2.1 内部 EEPROM 的操作——IAP 操作

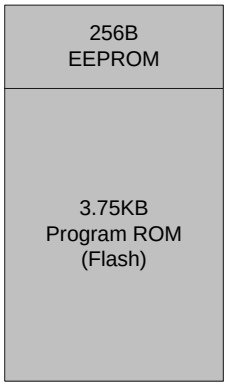
以 SC91F72B 为例，说明赛元 MCU 内部 EEPROM 的使用方法。

SC91F72B 内部有 256B Flash 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， 即作为 EEPROM 使用。

用户使用 IAP 时，只能把数据写入内部 4K Flash ROM 的最后 256 Bytes (0F00H ~ 0FFFH)。

SC91F712/SC91F711 为 ROM 最 128Bytes（780H~7FFH）

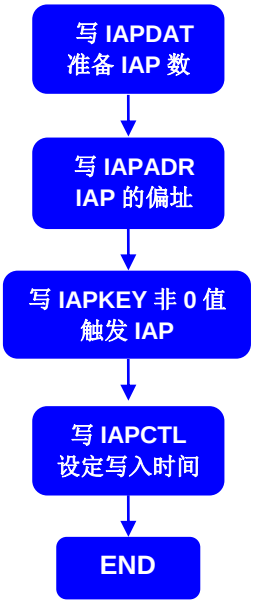
注意：SC91F72B/SC91F729B/SC91F73/SC91F731 的 Flash ROM 最后 4 Bytes 用于保存实际 IRC 相对 16MHz/8MHz 的偏差值及内部实际参考电压相对于 2.4V 的偏差值。如有用到以上数据，请不要对 EEPROM 的最高地址的 4 Bytes 进行 IAP 操作。



- **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。

EEPROM 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：



2.2 EEPROM 操作代码

```
#include "SC91F72B_C.H"
#include "Hex2Bin.h"
#include "intrins.h"

#define   ADD_BASE   0x0f00                    //定义 IAP 的基址

unsigned char  code *POINT;
unsigned char  DATAEEPROM;                     //定义一个要写入内部 EEPROM 的数据。

/**************************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
        IAPADR=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_();       //每次写入 IAP 数据需做 4 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。
}
/*********************主程序*********************/
void main()
{
        RSTCFG=0x0B;                           //初始化，关闭 RST，切换为 I/O 用；
                                               //设置 LVR 为最低电压 3.5V；
        Byte_Write(0x55,0);                    //IAP 写入一个字节（字节内容为 0x55），
                                               //到偏移地址为 0 的地址，
                                               //即是（0x0f00+0x00）地址；
        DATAEEPROM=Byte_Read(0x00);            //读取偏移地址为 0x00 的数据，即是
```

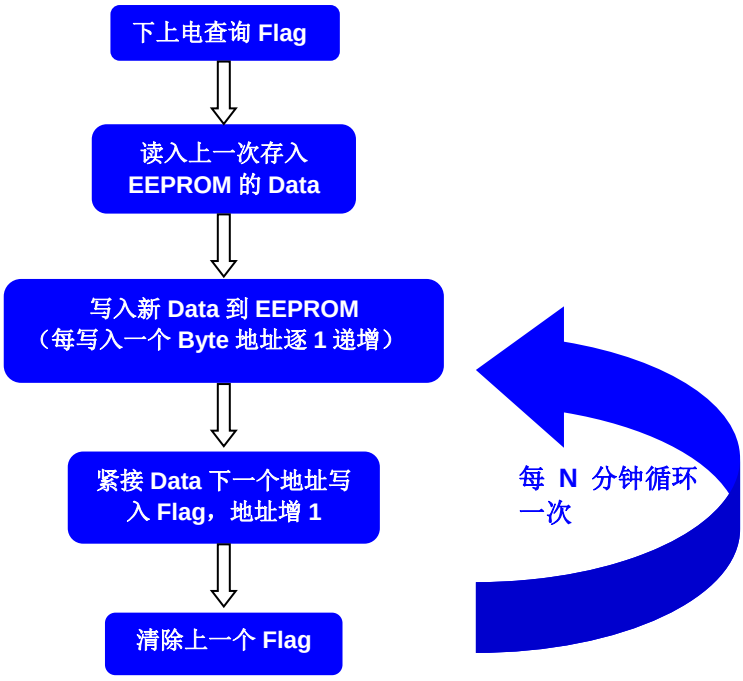
```
                                        //（0x0f00+0x00）的数据。
        while(1);
}
```

2.3 EEPROM 的使用算法

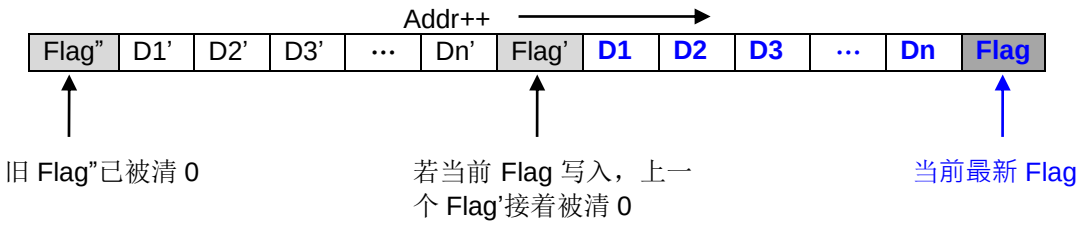
由于 SC91F72B，有 256B Flash 可以进行 In Application Programming（IAP）操作，而实际产品的应用中，譬如电压力锅，只是需要把仅几个 Bytes 的数据写到 EEPROM。为了充分利用 MCU 内部所有的 EERPOM，预防过早达到写 IAP 寿命次数，有以下算法供参考：

1.标志查询法：

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 SC91F72B 内部 256B 的 EEPROM，充分利用了 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 "SC91F72B_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   0x0f00                          //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
/*/
        IAPADR=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 2ms@8M。
        _nop_();_nop_();_nop_();_nop_();             //每次写入 IAP 数据需做 4 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;
        P37=0;
        soft_delay(800);                   //软延时
//显示分十位
        P1=chZimo[nMinS];
        P21=0;
        P20=1;
        P37=0;
        soft_delay(800);                   //软延时
//显示小时个位
        P1=chZimo[nHourG];
        P21=0;
        P20=0;
        P37=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()
{
        RSTCFG=0x0B;                        //初始化，关闭 RST，切换为 I/O 用；设置 LVR 为最
//低电压 3.5V；
         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);
}
```

2.分时写 IAP 法：

在实际电路中，用户会结合显示或者按键处理一起使用 EEPROM，为了保证 EEPROM 的使用不影响显示或者按键处理，需要将写 EEPROM 的过程使用分时处理的方式完成。可查看规格书中（SC91F72B）IAPCTL 中说明的描述：当写 IAP 时，CPU HOLD PC 指针，中断标志会被保存，并在 HOLD 结束后，进入中断，但多次的中断只能保留一次。

基本思路：数码管分时扫描时间片用作处理一次 IAP，如果写 N（N>=1）次，则只需要处理 IAP N 次的扫描即可。通常情况下，选择 IAP 写时间为 2ms@8M，数码管扫描时间>=2ms。

参考例程：

```
/*说明：工作过程中写 4byte 数据，EEWorkDone 用来控制何时开始写 EEPROM 数据，每 1 轮写 EEPROM
的均读后校验，错误则退出该轮写操作，等待下次重新写 1 轮 EE 数据*/
#include “SC91F72B”

unsigned char   code    *PointROm=0x0f00;      //SC91F72B 的 EEPROM 的起始地址
/************要写入 EEPROM 的随机数据*********************************/
unsigned char   TempData_1, TempData_2, TempData_3, TempData_4;
bit  EEWorkDone;                                //完成一轮 EEPROM 写

/************4bit 数码管 Com 扫描,扫描时间 3ms---------------*/
    void  Display_Scan(void)
    {
        static    unsigned char  ComNum;
                  switch(ComNum)
                  {
                  case 0:
                          COM1=0;
                          ComNum++;
                          break;
                  case 1:
                           COM2=0;
                           ComNum++;
                            break;
                  case 2:
                          COM3=0;
                          ComNum++;
                          break;
                  case 3:
                           COM4=0;
                           ComNum++;
                          break;
                  case 4:
                          IAPWrite_4Byte();                    //写一轮 4byte 数据
                          ComNum=0;
                          break;
                  default:
                          ComNum=0;
                          break;
```

```
            }
    }
/*******************完成一轮写 4byte 数据*************************************/
void IAPWrite_4Byte(void)
{
    static    unsigned char WriteCnt;
    if(EEWorkDone)
    {
            switch(WriteCnt)
            {
             case    0:
                     IAPWrite(WriteCnt,TempData_1);
                     If(TempData_1!= IAPRead(WriteCnt))
                     {
                          EEWorkDone =0;
                          WriteCnt =0;
                     }
                     else
                     { WriteCnt++;    }
                     break;
             case    1:
                     IAPWrite(WriteCnt,TempData_2);
                     If(TempData_2!= IAPRead(WriteCnt))
                     {
                          EEWorkDone =0;
                          WriteCnt =0;
                     }
                     else
                     { WriteCnt++;}
                     break;
             case    2:
                     IAPWrite(WriteCnt,TempData_3);
                     If(TempData_3!= IAPRead(WriteCnt))
                     {
                          EEWorkDone =0;
                          WriteCnt =0;
                     }
                     else
                     { WriteCnt++;}
                     break;
             case    3:
                     IAPWrite(WriteCnt,TempData_4);
                     If(TempData_4!= IAPRead(WriteCnt))
                     {
                          EEWorkDone =0;
                          WriteCnt =0;
                     }
                     else
                     { WriteCnt++;}
                     break;

              default:          WriteCnt=0;break;
            }
    }
}
/*********************IAP 写操作*************************************/
void IAPWrite(unsigned char IAPAdress,unsigned char  IAPDate)
```

```
{
        IAPDAT=IAPDate;          //准备写入数据 dat
        IAPADL=IAPAdress;        //写入地址
        IAPKEY=0x80;             //128CLK 未收到写命令则关闭
        IAPCTL=0x0a;             //写入，Hold time 2ms@8MHZ
        NOP;
        NOP;
        NOP;
        NOP;
}

/**********************IAP 读操作************************************/
unsigned char  IAPRead(unsigned char IAPAdress)
{
        unsigned char    temp;
        temp = *(PointROm+IAPAdress);
        return    temp;
}
```

3 电路设计的注意事项

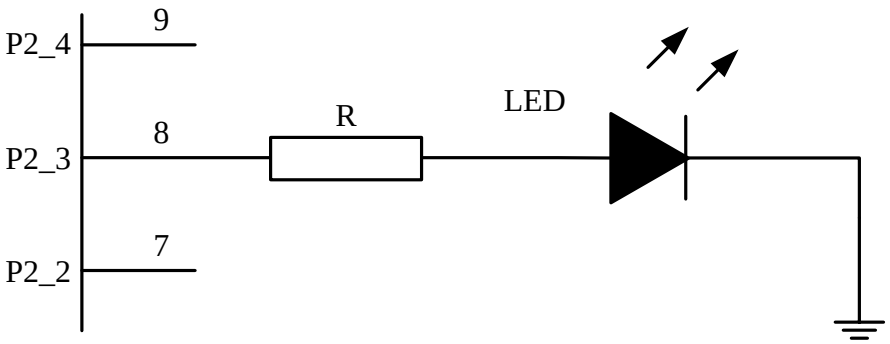
赛元 MCU 的 GPIO 上电默认模式为准双向模式，譬如 SC91F72B、SC91F73。此外，赛元 MCU 的 **RST 管脚，低电平使能，用户设计电路不能在上电时强制拉低**（上电复位时，系统默认为 RST，复位完成后可通过设置 SFR（RSTCFG）取消 RESET 功能并将此 Pin 设为 GPIO，此后管脚低电平不会产生复位）。

由于以上特性，用户设计电路时，需要注意以下几点，譬如 LED 的使用以及接法、1bit 阴极数码管的使用，MCU 的 RST 管脚电路。

3.1 电路设计实例

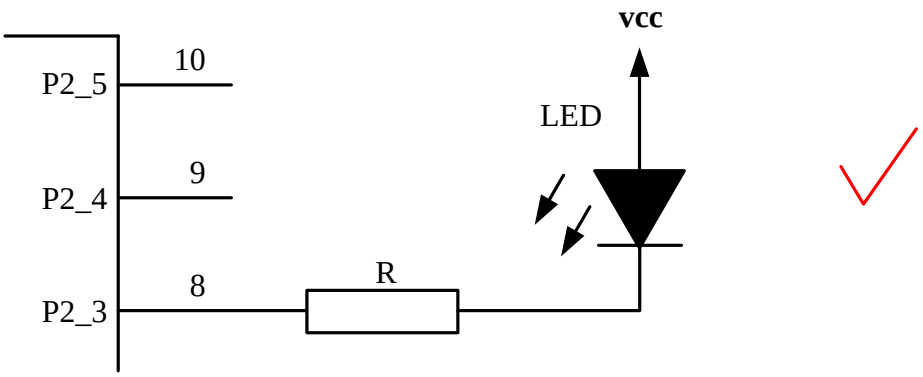
3.1.1 LED 的使用以及接法

- **非建议接法**：LED 正向接入 I/O，负向接 GND，见下图：



说明：以上接法不可取。因为 I/O 上电默认为准双向模式，有 uA 级别的弱输出，又 LED 对电流非常敏感，导致 LED 导通，在 IC 上电到用户程序改变 I/O 默认状态期间会表现为微亮（**时间很短，us 级别**）。

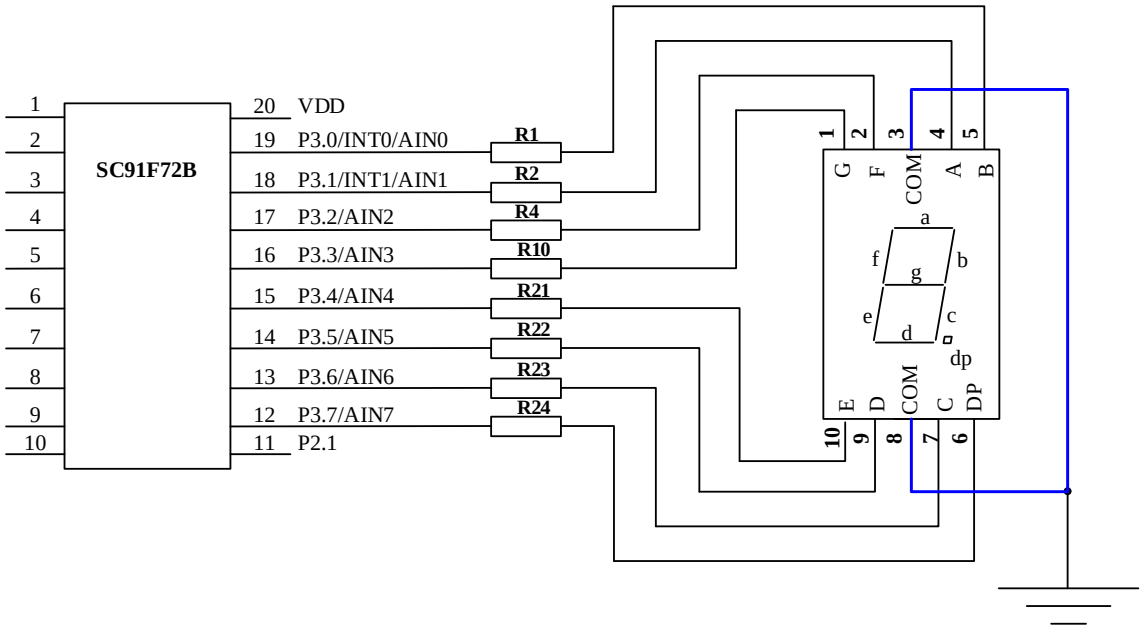
- **建议接法**：LED 正向接到 VCC，负向接入 I/O。见下图：



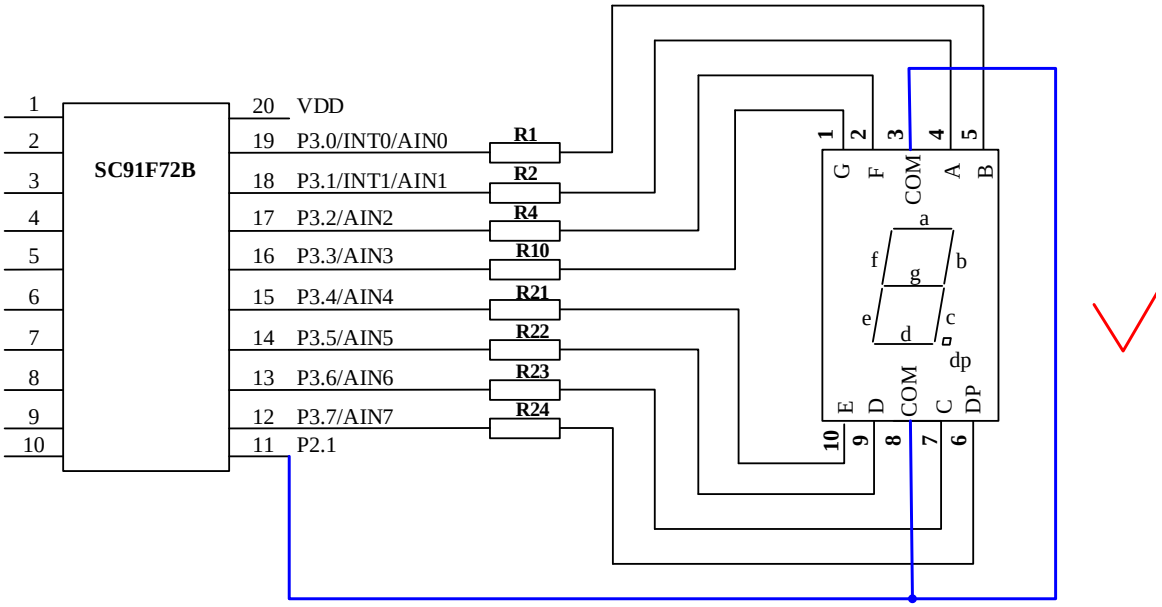
3.1.2 1 位共阴极数码管的使用

数码管是由 LED 构成，控制原理与 LED 一样。1 位共阴极传统接法是 COM 接到 GND，这种接法会导致数码管在 IC 上电到用户程序改变 I/O 默认状态期间出现微亮现象。为避免这种现象，需要采用 COM 接到 I/O 的方法来，这种方法比传统接法多占用一个 I/O，请使用者注意。

◆ **非建议接法：**见下图。



◆ **建议接法：**见下图。

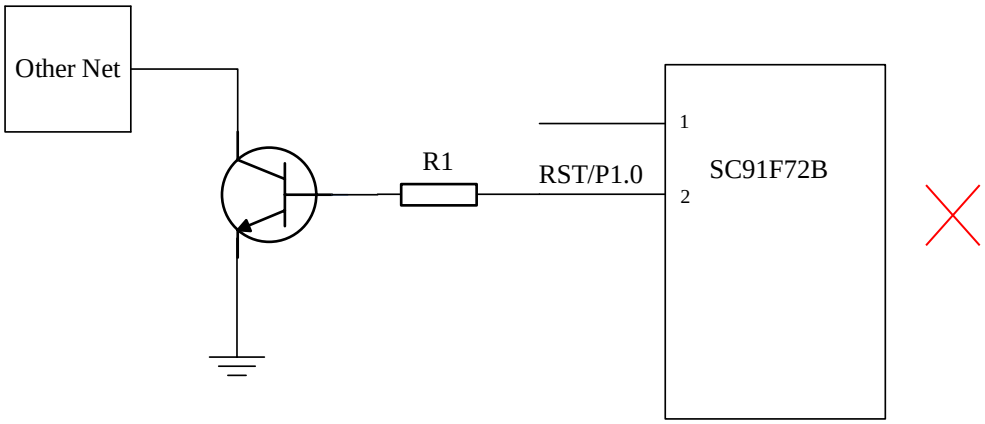


注意：至于其他多位数码管，对连接方法没有特殊要求，按正常接法即可，因为多位数码管对 MCU I/O 上电默认模式无关。

3.1.3 RST 管脚电路

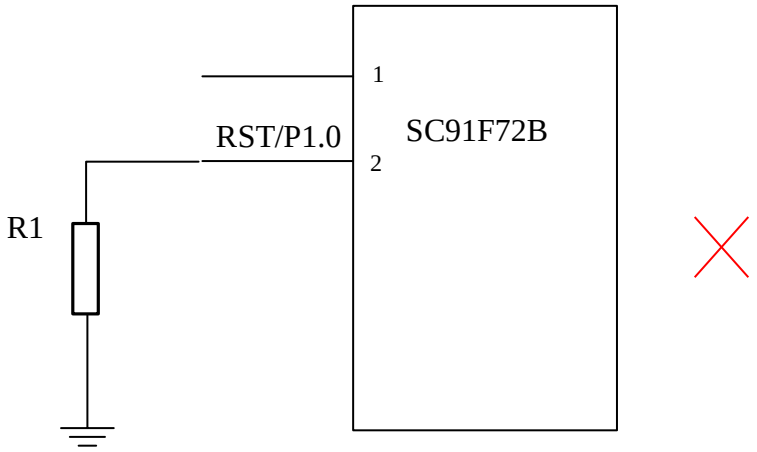
赛元 MCU 的 RST 引脚，与 I/O 复用，有别于传统 MCU 的 RST 引脚（传统的 RST 的引脚只能做输入，不能做输出），既可以做输入，又可以做输出。但是**上电复位时，用户电路不能强制拉低，否则会一直复位，无法正常工作**。因此说，用户设计电路时，需要注意：

◆ 错误接法：



说明：以上电路接法，尽管 RST 是先通过一个电阻 R1，再连接一个三极管到地的，但是，三级管的基极 b 与发射极 e 间的阻抗，即 Rbe（一般来说，Rbe 为几 K 到几十 K），在系统上电时与内部上拉分压还是判为低电平，造成系统一直复位，无法正常工作。

◆ 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。

3.2 实现电路设计的方法

3.2.1 I/O 设为高阻，实现电路设计

赛元 MCU 的 GPIO，有四种工作模式：

①. 准双向工作模式；
②. 强推挽工作模式；
③. 高阻工作模式；
④. open drain 工作模式。

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，譬如 SC91F72B，SC91F731，SC91F73 等等。**

3.2.2 I/O 的准双向模式

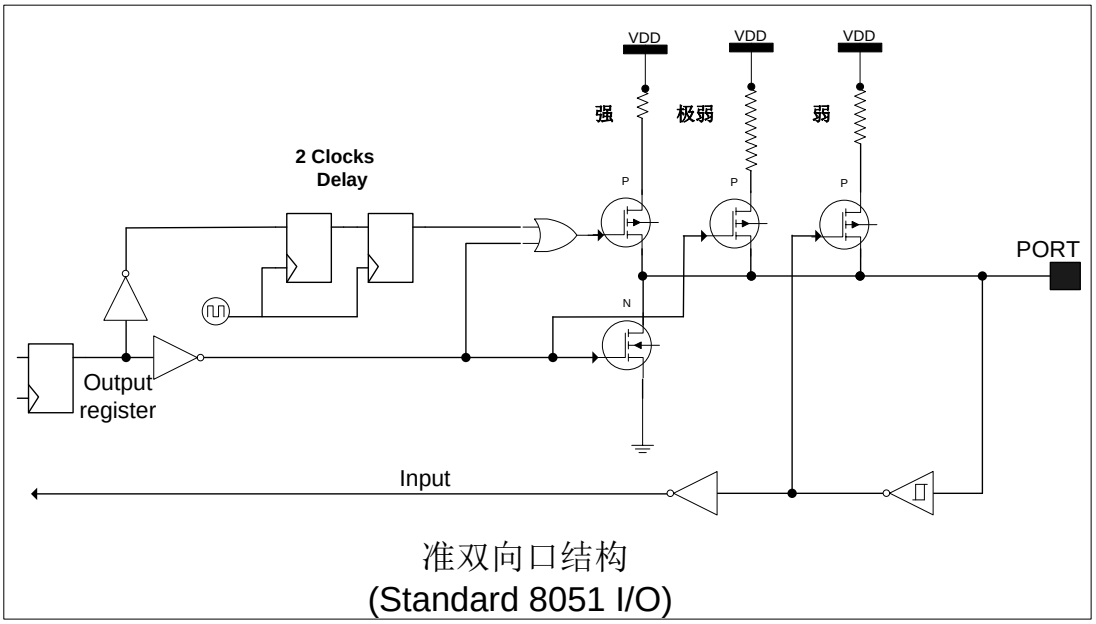
准双向口有 3 个上拉的 MOS 管以适应不同的需要，分别称为“弱（Weak）上拉”、“极弱（Very weak）上拉”和“强（Strong）上拉”。

在 3 个上拉 MOS 管中，有 1 个上拉 MOS 管称为**“弱上拉”，当口线寄存器为 1 且引脚本身也为 1 时打开**。此上拉提供基本驱动电流使准双向口输出为 1.如果 1 个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。

第 2 个上拉 MOS 管称为“极弱上拉”，当口线锁存为 1 时打开。当引脚悬空时，这个极弱的上拉源产生很弱的电流将引脚上拉为高电平。

第 3 个上拉 MOS 管称为**“强上拉”，当口线锁存器由 0 跳变为 1 时，这个上拉用来加快准双向口由逻辑 0 到逻辑 1 转换**。当发生这种情况时，强上拉打开约 2 个机器周期以使引脚能迅速地上拉到高电平。

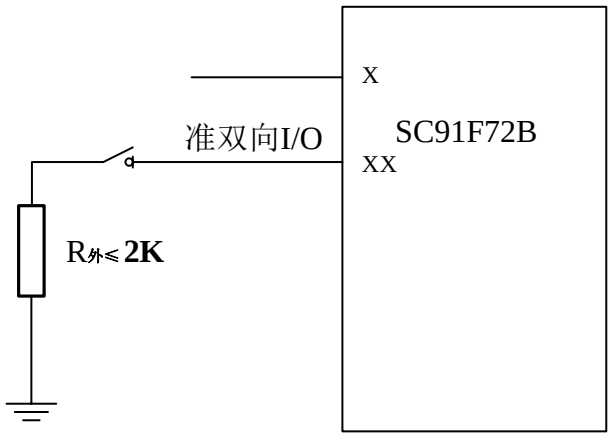
准双向模式的端口结构示意图如下：**（$R_{极弱}$>$R_{弱}$）$R_{强}$）**



准双向口结构
(Standard 8051 I/O)

3.2.3 I/O 准双向模式检测按键

据上 3.2.2 的理论描述，就 I/O 的工作模式在上电默认为准双向模式，进行举例说明。I/O 外接一个下拉电阻 **$R_{外}$**，有以下情况：



XX 切换为准双向 I/O 后，接入 $R_{外}$：

这时，**$R_{弱}$**一直处于为打开状态，除非 **$R_{外}$**电阻足够小，把 I/O 的电压拉到低电平，根据公式计算：

$$U_{I/O}=R_{外}*VDD/（R_{外}+R_{极弱}\,||\,R_{弱}）$$

由于 I/O 内部的 $R_{弱}$较小，并考虑到外围及布板因素，一般来说，$R_{外}$<=2 KΩ（在不影响外部电路功能的情况下，$R_{外}$越小越好），才可把 I/O 的电压拉到低电平。

4 附注：赛元 MCU 的 DEMO 程序

赛元 MCU，使用 KeilC 的开发平台，可参照资料《赛元 MCU 的工具使用说明》。本章节以 SC91F72 为例，对芯片的各个基本功能，做一个 demo 程序，具体见下。

4.1 I/O 的初始化设置

```
/*********************************************************************
                        SC91F72B GPIO 的使用
 文件名：IO.c
 功能说明：SC91F72B GPIO 的输入、输出及其强推挽使用
*********************************************************************/
#include "SC91F72B_C.h"
/*********************************************************************
            IO 初始化函数
函数原型: void GPIO_init(void);
功能:    初始化 IO 状态
*********************************************************************/
void GPIO_init(void)
{
        RSTCFG|=0x08;                   //P1.0 切换为 GPIO,取消其复位功能

        P1CFG0&=0Xf0;                    // P1CFG0=_b11110000，设置 P1.0、P1.1 为准相向模式
        P1      |=0x03;                 //设置 P1.0、P1.1 口为输入(在被设为准双向模式条件下),
                                        //也即输出高，但驱动能力较弱
        P1      &=~0x03;                //设置 P1.0、P1.1 口为输出低

        P2CFG0=0X05;                    //设置 P2.0、P2.1 口为强推挽输出
        P2      =0x03;                  // P2.0、P2.1 强制输出高，驱动能力较强（在被设为强推挽模式
                                        //条件下）
        P2      &=~0x03;                //P2.0、P2.1 口为输出低

}
void main(void)
{
        GPIO_init();
        while(1);
}
```

4.2 ADC 中断

```
/*********************************************************************
                        SC91F72B ADC 中断
 文件名：ADC_INT.c
 功能说明：在 ADC 中断服务子程序中反转 IO,转换完成一次约 89 个 ADC CKL
*********************************************************************/
#include "SC91F72B_C.H"

sbit TEST_PORT=P2^1;                    //ADC Interrupt test PORT

/*********************************************************************
            ADC 初始化函数
函数原型: void AdcInit(void);
功能:    选择参考电压、ADC 输入口；配置 ADC 启动
*********************************************************************/
void ADC_Init(void)
{
        /*选择参考电压*/
        ADCCFG=0x00;                    //VDD 作参考电压;ADCCFG=_b00000000;
        P3ADC=0x80;                     //remove P3.7 Pll-up,使能 ADC 输入 ;P3ADC=_b10000000;
```

```
        /*启动 ADC 电源,Fadc=Fosc/6,ADCS 开启，P3.7 为输入口*/
        ADCCR=0xEF;                     //ADCCR=_b11101111;

        EADC=1;                         //ADC 中断使能
}

/*************************************************************************
            ADC 中断服务子函数
函数原型: void ADC()interrupt 6;
功能:    反转 IO
*************************************************************************/

void ADC()interrupt 6
{
        ADCCR =ADCCR&0xEF;          //ADCIF 清 0
        TEST_PORT= ~ TEST_PORT;     //反转 IO 口
        ADCCR =ADCCR|0x08;          //再次启动 ADC

}

/*************************************************************************
            主函数
函数原型: void main(void);
功能:    测试 ADC 中断
*************************************************************************/

void main(void)
{
        RSTCFG|=0x08;                       //P1.0 切换为 GPIO,取消复位功能
        ADC_Init();
        EA=1;                           //开总中断
}
```

4.3 PWM 周期

```
/*************************************************************
                    SC91F72B PWM        OutPut 800us
 文件名：PWM_800us.c
 功能说明：PWM 时钟选择 128 分频，PWM 输出周期为 800us
*************************************************************/

#include"SC91F72B_C.h"

/*************************************************************
            PWM 初始化函数
函数原型: void PWM_init(void);
功能:    设置 PWM 周期 800us
*************************************************************/
void PWM_init(void)
{
        RSTCFG|=0x08;                   //P1.0 切换 GPIO,取消其复位功能
        PWMCFG=0x06;                    //PWM 输出不反向，设定 128 个系统时钟才对 PWM 算数器+1；
        /*PWM0 输出周期*/
        PWMPRD=99;                      //初始化为 PWMPRD=99;周期=99+1=100 个 PWM CKL
        /*PWM0 High 周期*/
```

```
        PWMDTY0=10;                 //PWM0 初始化为 PWMDTY0=10,duty cycle=PWMPRD/ 周期
=10/100=10%;
        PWMDTY1=10;
        PWMCR=0x85;                 //开启 PWM，关闭 PWM 中断
}
/*********************************************************************
            主函数
函数原型: void main(void);
功能:    PWM 输出周期为 800us
*********************************************************************/
void main(void)
{
        PWM_init();
        while(1);
}
```

4.4 TIMER 定时

```
/************************************************************
                        SC91F72B Timer0 计时
 文件名：Timer0_M2.c
 功能说明：使用 Timer0 定时 50us,并在其中断服务子程序中反转 IO
*********************************************************************/

#include "SC91F72B_C.h"
#define  Test_PORT  P33                 //宏定义


/*********************************************************************
            硬件初始化函数
函数原型: void Hard_init(void);
功能:    切换 P1.0 为 GPIO，并选择 1/4 时钟分频
*********************************************************************/
void Hardware_init(void)
{
        RSTCFG|=0x08;                       //P1.0 切换为 GPIO,取消其复位功能
}
/*********************************************************************
            Timer0 初始化函数
函数原型: void Timer0_init(void);
功能:    载入初始值，并启动 Timer0
*********************************************************************/
void Timer0_init(void)
{

 TMOD=0x02;                         //选择工作方式 2
 TMCON=0x01;                        //fsys=fosc/4
/*载入初始值，定时 50us*/
 TH0=(256-200);
 TL0=(256-200);

 TR0=0;
 ET0=1;                             //使能 Timer0 中断
 TR0=1;                             //启动 Timer0
}
```

```
void Timer0() interrupt 1                    //入口向量表示为"1"
{
  Test_PORT=~Test_PORT;                      //反转 IO
}
/*********************************************************************
                主函数
函数原型: void main(void);
功能:    开启总中断，产生 50us 定时
*********************************************************************/
void main(void)
{
        Hardware_init();
        Timer0_init();
        EA=1;                                //开总中断
}
```

4.5 TIMER 计时

```
/*********************************************************************
                        SC91F72B  Timer0 计数
 文件名：Timer_Count.c
 功能说明：PWM 提供 2us 的计数脉冲，Timer0 计满 4000 溢出。
                P12 为 T0/PWM0 复用，因此，开 PWM0 输出，作为 T0 脉冲。
*********************************************************************/
#include "SC91F72B_C.h"
#define Tst_PORT        P21             //Timer0 计数溢出反转 IO

/*********************************************************************
                PWM 初始化函数
函数原型: void PWM_init(void);
功能:    PWM 周期 4us，脉冲时间为 2us,选择 Fosc/8 时钟源
*********************************************************************/
void PWM_init(void)
{
        PWMCFG=0x03;                    //PWM 时钟源选择 Fosc/8 =2MHZ
        PWMPRD=7;                       //周期设置 8*1/2us=4us
        PWMDTY0=4;                      //High duty=2us
        PWMCR=0x81;                     //打开 PWM 模块；P21 切换 PWM0 输出;b10000001
}


/*********************************************************************
                Timer0 初始化函数
函数原型: void timer0_init(void);
功能:    选择工作方式 1 计数
*********************************************************************/
void timer0_init(void)
{
        TMCON=0x01;                     //fsys=fosc/4
        TMOD=0x05;                      //方式 1,GATE0=0;C/T=1,count pin -->P1.2;b00000101
        TH0=(65536-4000)>>8;            //(65536-4000)/256;
        TL0=(65536-4000)&255;           //(65536-4000)%256;

        TR0=0;
```

```
    ET0=1;                          //Timer0 中断使能
    TR0=1;                          //启动 Timer0
}

/*************************************************************************
        Timer0 中断服务子程序函数
函数原型: void timer0()interrupt 1;
功能:    检测计数
*************************************************************************/
void timer0()interrupt 1
{
    /*再次装入初值*/
    TH0=(65536-4000)>>8;
    TL0=(65536-4000)&255;
    Tst_PORT=~Tst_PORT;          //检测溢出值；是否为 16*2=32ms？

}

/*************************************************************************
        主函数
函数原型: void main(void);
功能:    测试 Timer0 计数
*************************************************************************/
void main(void)
{
    RSTCFG|=0x08;                       //P1.0 切换为 GPIO,取消复位功能
    timer0_init();
    PWM_init();
    EA=1;
    while(1);
}
```