目录

1 赛元MCU有关MOVC指令的应用注意 1

1.1 C语言编程有关MOVC指令的应用注意 1
1.1.1 C语言开发，MOVC指令 1
1.1.2 C语言开发具体操作 1

1.2 汇编语言编程有关MOVC指令的应用注意 5

2 赛元MCU的EEPROM，及算法解说 6

2.1 内部EEPROM的操作——IAP操作 6

2.2 EEPROM操作代码 7

2.3 EEPROM的使用算法 7

3 电路设计的注意事项 13

3.1 电路设计实例 13
3.1.1 LED的使用以及接法 13
3.1.2 1Bit共阴极数码管的使用 13
3.1.3 RST管脚电路 14

3.2 实现电路设计的特殊方法 15
3.2.1 I/O设为高阻，实现电路设计 15
3.2.2 I/O的准双向模式 16
3.2.3 I/O准双向的使用 16
3.2.4 I/O设为准双向，实现电压检测的方法 17
3.2.5 I/O设为准双向，实现过零检测的方法 17
3.2.6 I/O设为准双向，实现LCD应用的方法 21

4 附注：赛元MCU的DEMO程序 22

4.1 I/O的初始化设置 22

4.2 ADC中断 23

4.3 PWM周期 24

4.4 Timer定时 25

4.5 Timer计时 26

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（SC91F72，SC91F719 除外）。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

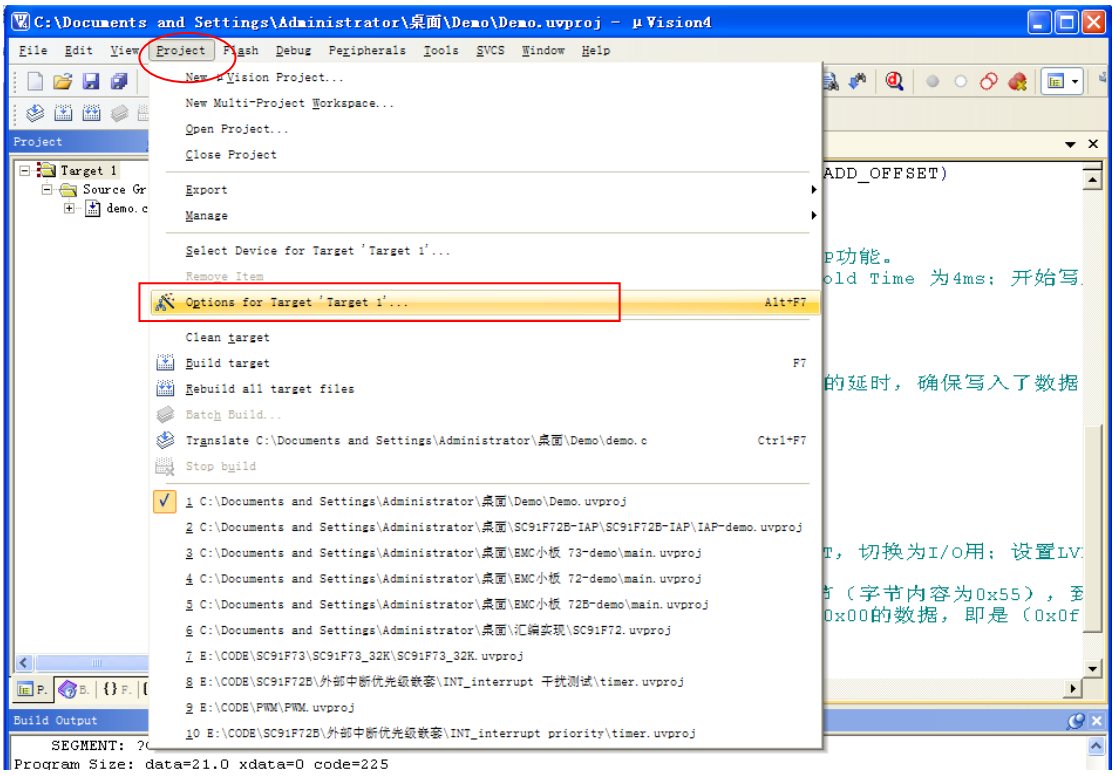
注意：?C?LIB_CODE 标识符只是表明某个函数调用的库文件进行查表运算，通常情况下，客户开发产品不需用调用库文件 Math 函数。（库文件占用较大的 ROM 空间,譬如 sinx 函数）。

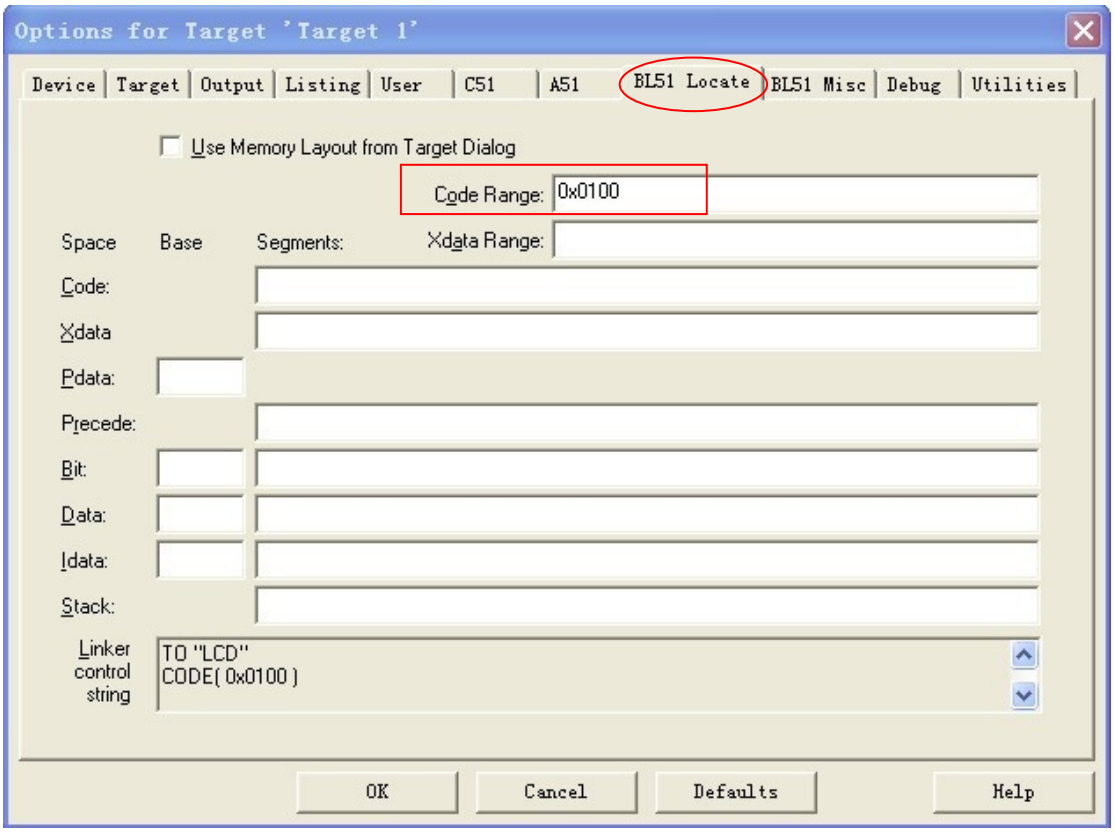
.M51 文件详细记录了上表中各代码段的使用情况，包括 Code 的起始地址、长度等。用户只需要查看?C_INITSEG、?CO?Project_name、调用库文件的函数（如果有?C?LIB_CODE 的话）的起始地址是否在禁止访问区，如果在禁止访问区，可参考后续操作改变起始地址。

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：



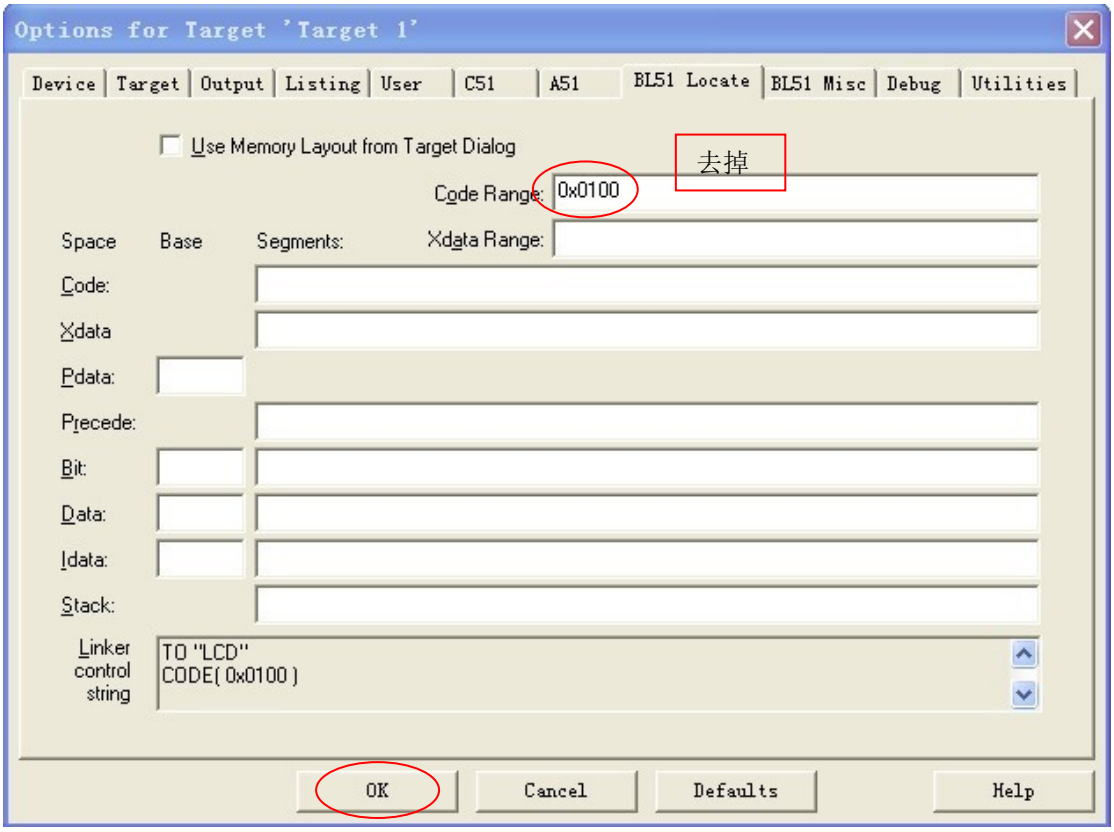


◆ **调试完成后，生成最终程序；**

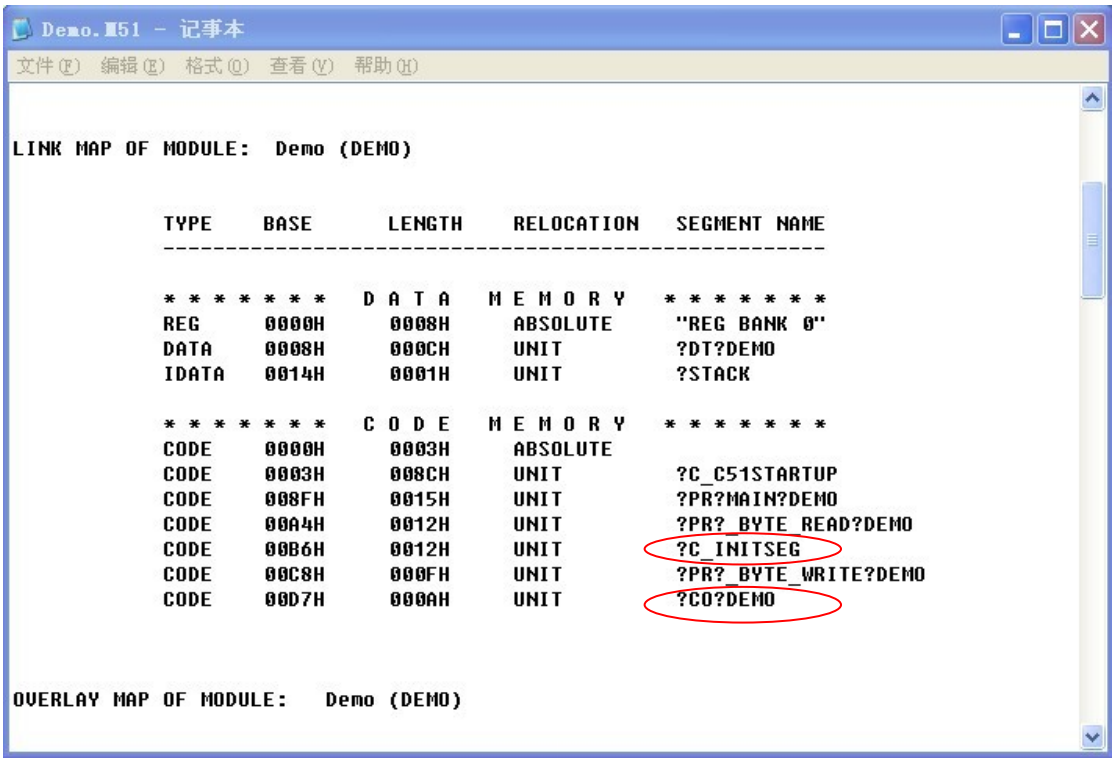
若编程代码中存在全局变量，code 类型数据（不可变），需要将这些数据类型存放到 0x0100 地址之后。

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。



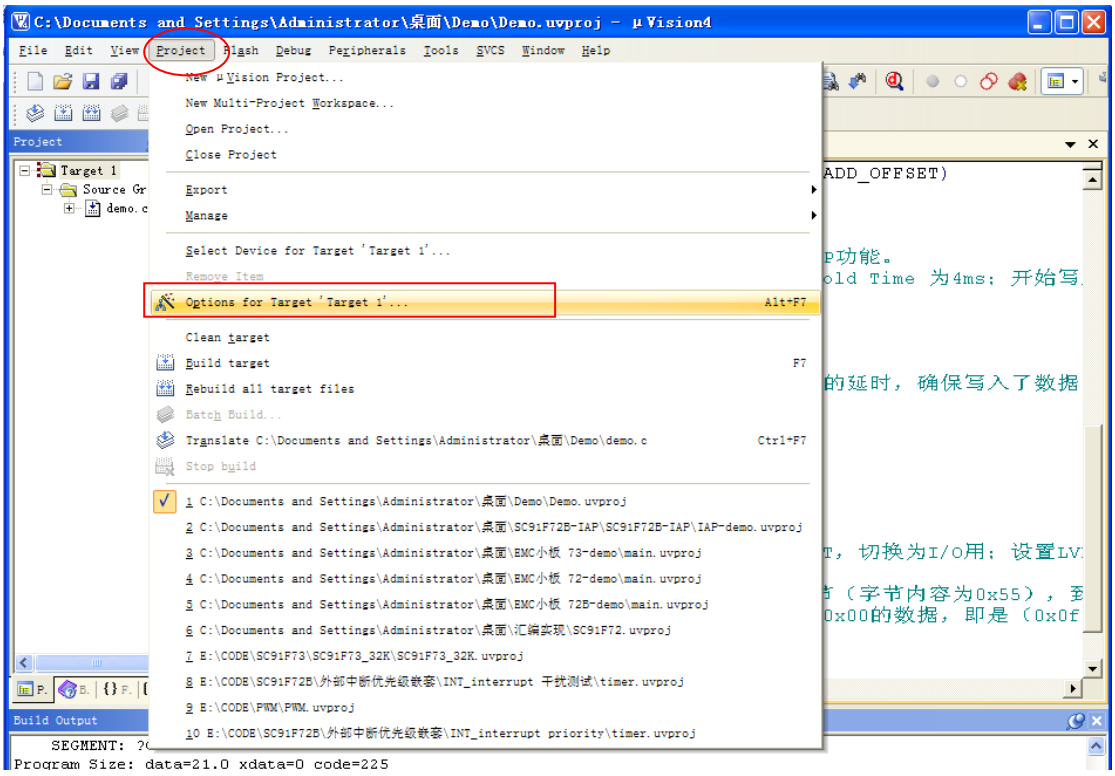
② 重新编译后，在建立的工程目录下，找到并打开 .M51 文件，在 CODE MEMORY 会出现：
“?C_INITSEG” :全局变量初始化数据。
“?CO?DEMO” :code 类型数据。

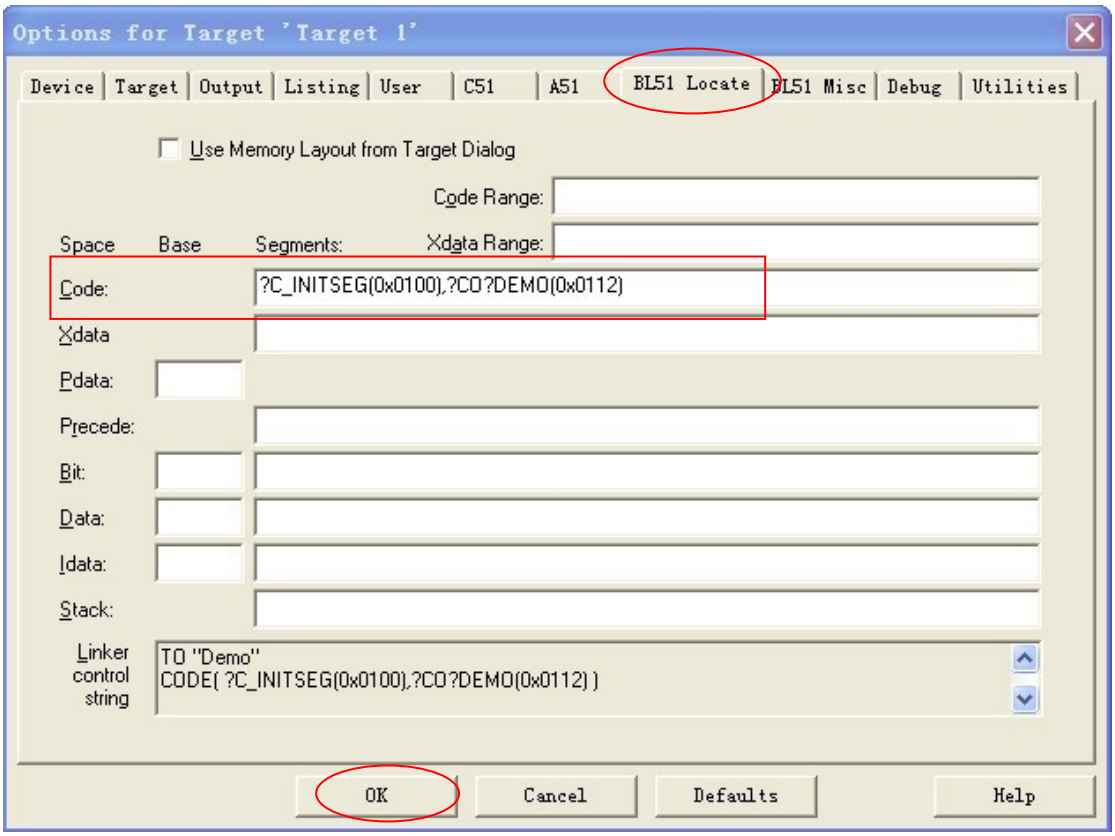


说明:从以上 M51 文件的“CODE MEMORY”信息中，可以看到“?C_INITSEG”，链接地址为 00B6H，长度为 0012H 字节；“?CO?DEMO”，链接地址为 00D7H，长度为 000AH 字节。

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

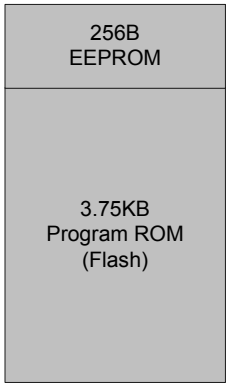
2.1 内部EEPROM的操作——IAP操作

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)。

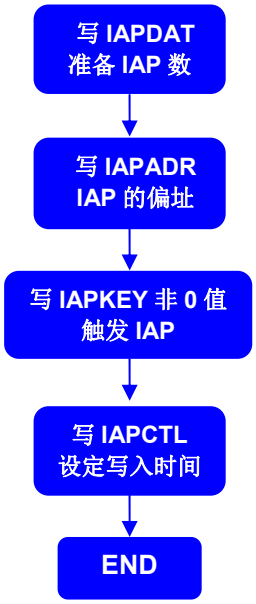
注意：SC91F72B/SC91F729B/SC91F73/SC91F731 的 Flash ROM 最后 4 Bytes 用于保存实际 IRC 相对 16MHz/8MHz 的偏差值及内部实际参考电压相对于 2.4V 的偏差值。如有用到以上数据，请不要对 EEPROM 的最高地址的 4 Bytes 进行 IAP 操作。



- **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：



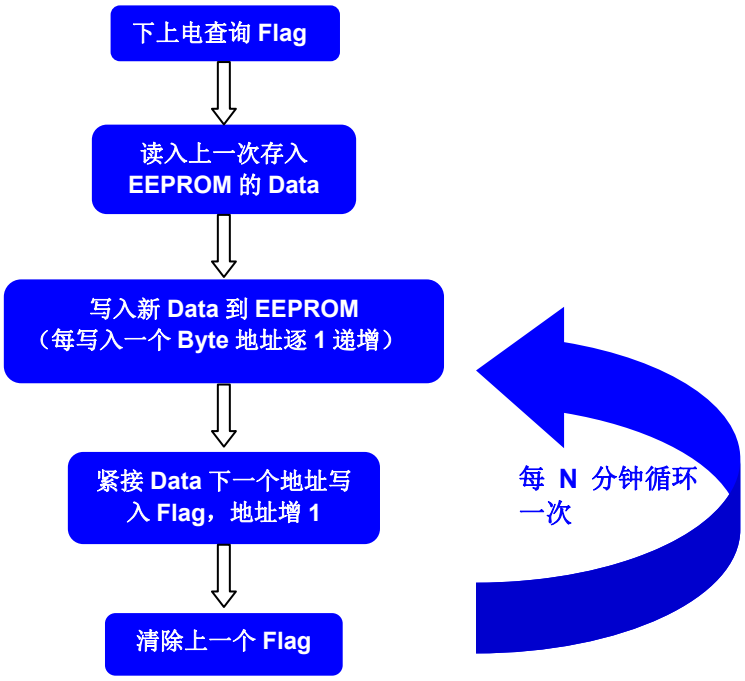
2.2 EEPROM操作代码

```
#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=0x0c;                           //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);
}
```

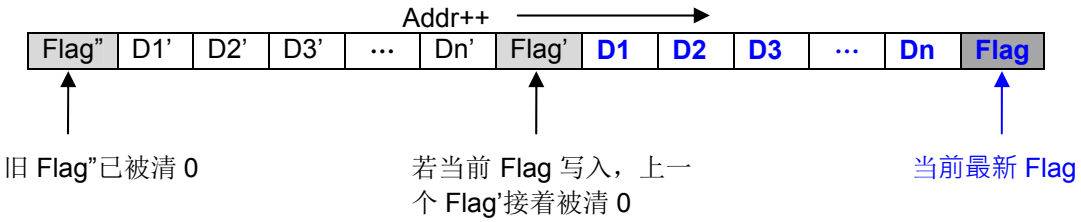
2.3 EEPROM的使用算法

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，提高 EEPROM 的寿命，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 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
/*****************************************/
/*****************************************/
/***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;
        }
        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。
}
//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=0x0c;                    //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);
}
```

3 电路设计的注意事项

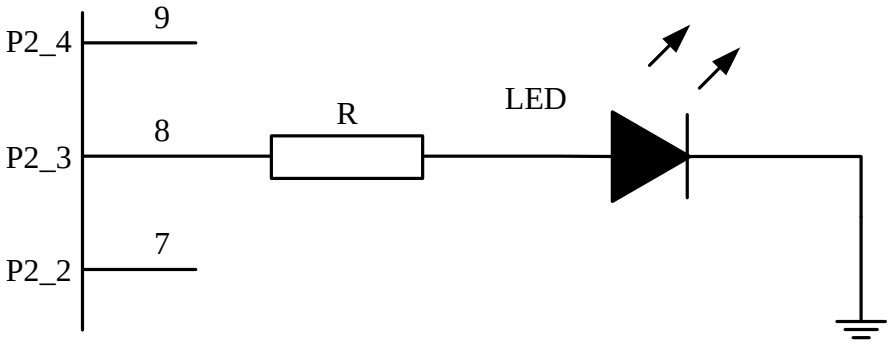
赛元 MCU 的 GPIO 上电默认模式为准双向模式，譬如 SC91F72，SC91F719。此外，赛元 MCU 的 **RST 管脚，低电平使能，用户设计电路不能在上电时强制拉低**（上电复位时，系统默认为 RST，复位完成后可通过设置 SFR（RSTCFG）取消 RESET 功能并将此 Pin 设为 GPIO，此后管脚低电平不会产生复位）。

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

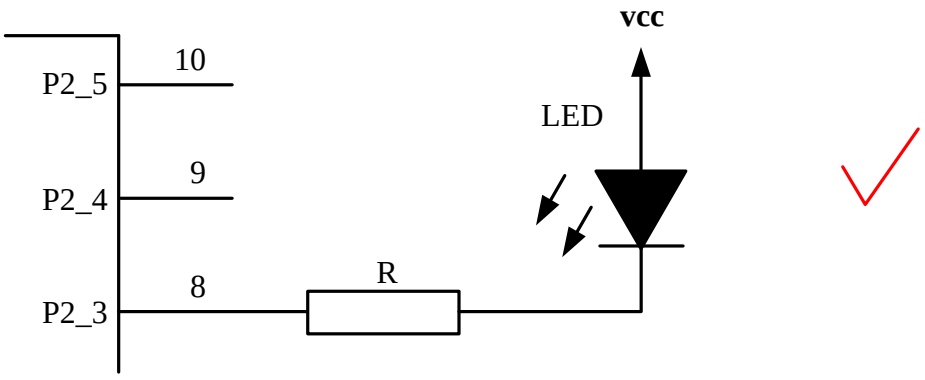
3.1.1 LED的使用以及接法

- **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，在初始化前会表现为微亮（**时间很短，us 级别**）。

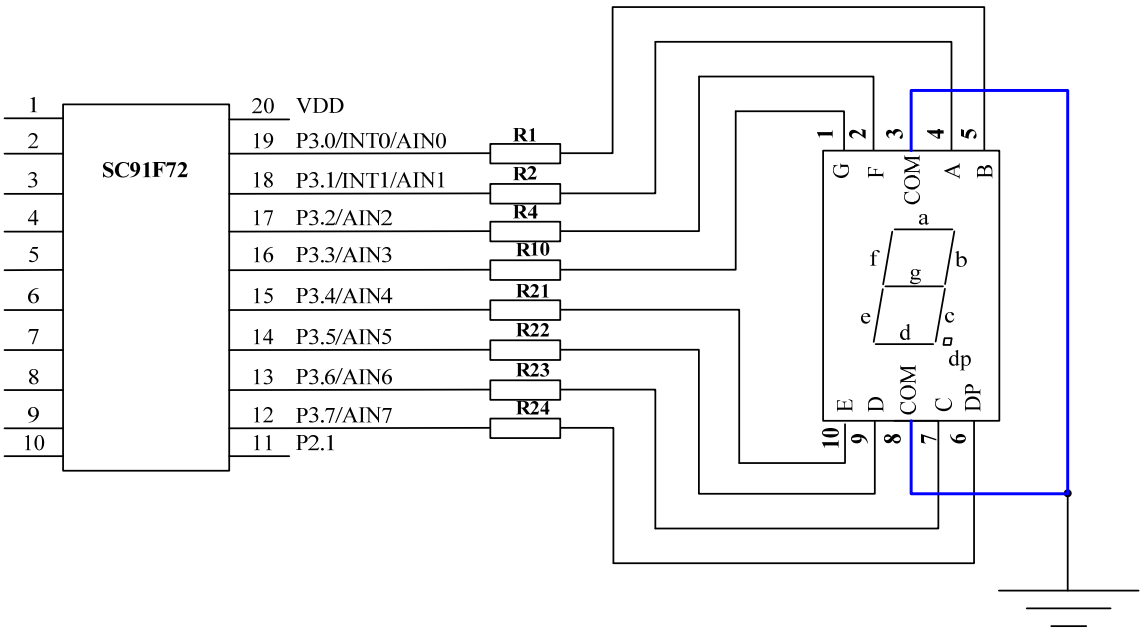
- **建议接法**：LED 正向接到 VCC，负向接入 I/O。见下图：



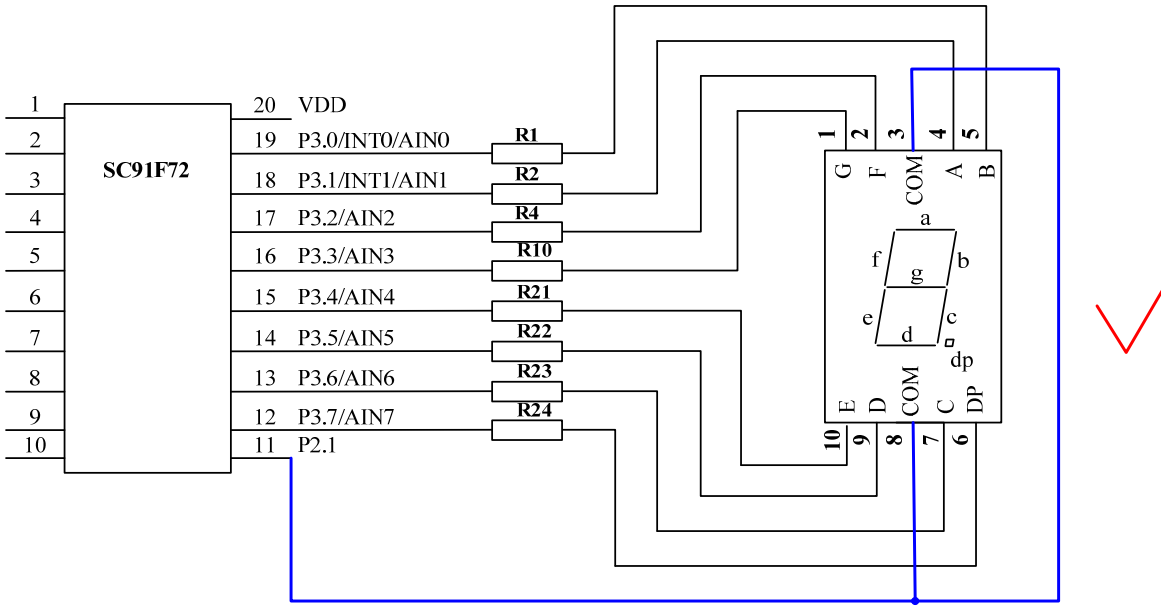
3.1.2 1BIT共阴极数码管的使用

数码管是由 LED 构成，控制原理与 LED 一样。1bit 共阴极传统接法是 COM 接到 GND，这种接法会导致数码管在上电后初始化前，会出现微亮现象。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。

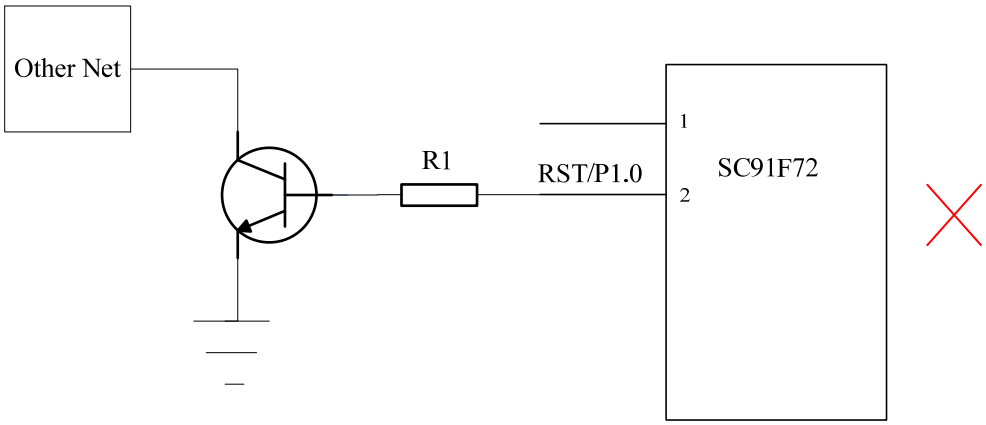


注意：至于其他多位数码管，对连接方法没有特殊要求，按正常接法即可，因为多位数码管对 MCU I/O 上电默认模式无关。

3.1.3 RST管脚电路

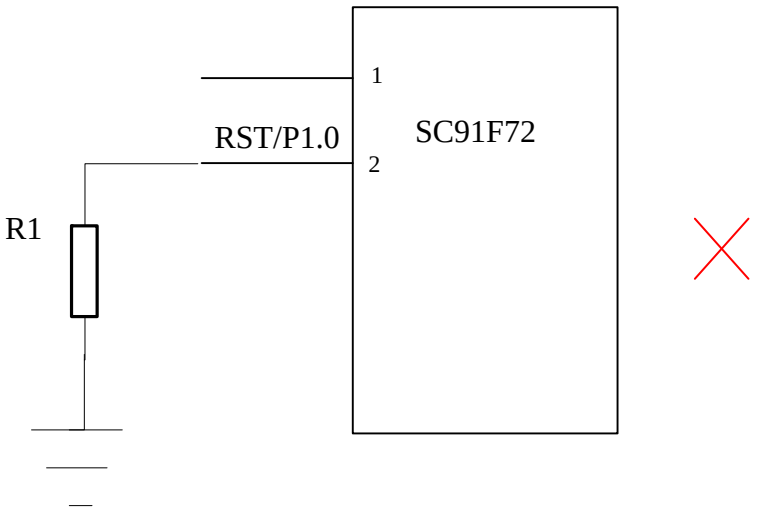
赛元 MCU 的 RST 引脚，与 I/O 复用，有别于传统 MCU 的 RST 引脚（传统的 RST 的引脚只能做输入，不能做输出），既可以做输入，又可以做输出。但是**上电复位时，用户电路不能强制拉低，否则会一直复位，无法正常工作**。因此说，用户设计电路时，需要注意：

◆ 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 外接一个小电阻 R1（通常说来，R1<120KΩ），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 实现电路设计的特殊方法

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。**

不过，赛元 MCU 体系中的 SC91F72，SC91F719（其他 MCU 都有 4 种工作模式），只有准双向工作模式、强推挽工作模式。（一般来说，该两种工作模式也足以满足产品开发需求）

本章节主要是讨论，**利用 SC91F72，SC91F719 采用准双向模式**，来实现以上这些特定场合的应用，即包括**电压检测，过零检测，LCD 的应用**等。

3.2.2 I/O的准双向模式

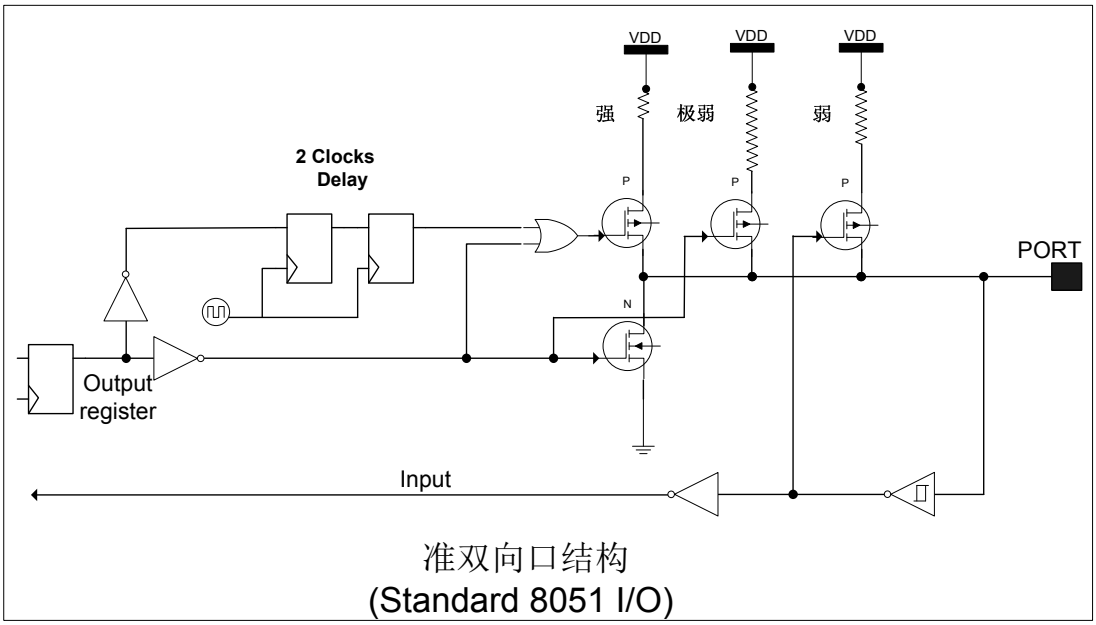
准双向口有 3 个上拉的 MOS 管以适应不同的需要，分别称为“弱（Weak）上拉”、“极弱（Very weak）上拉”和“强（Strong）上拉”。

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准双向的使用

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 外接一个电阻或等效电阻 **$R_{外}$**，有以下情况：

①. **接好了 $R_{外}$重新上电:**

若 **$R_{外}$**分压判高，即

$$U_{I/O}=R_{外}*VDD/(R_{外}+R_{极弱})$$

为高电平，才打开 **$R_{弱}$**，由于 **$R_{弱}$<<$R_{极弱}$**，则此时准双向输出的电流才较大，一般有几十 uA（由 **$R_{外}$**大小决定），(下调 **$R_{外}$**，最大达到 240uA 左右)。

若 **$R_{外}$**分压判低，即

$$U_{I/O}=R_{外}*VDD/(R_{外}+R_{极弱})$$

为低电平，会关闭 **$R_{弱}$**，由于 **$R_{极弱}$较大**，在 200K 多左右，则此时准双向输出的电流较小，在 **20UA 以下**（以 SC91F72 为例）。

以 SC91F72 为例，若 **$R_{外}$**的电阻足够大时，譬如 **$R_{外}$**=200KΩ 时，**$R_{外}$分压判高。（R 外>115KΩ，**一般情况下，**$R_{外}$**的取值要留有一定的余量）

②. **系统上电后，才接入 R 外：**（应用中一般不会出现这种方式，但是为了说明 I/O 的特性，在次赘述一下）

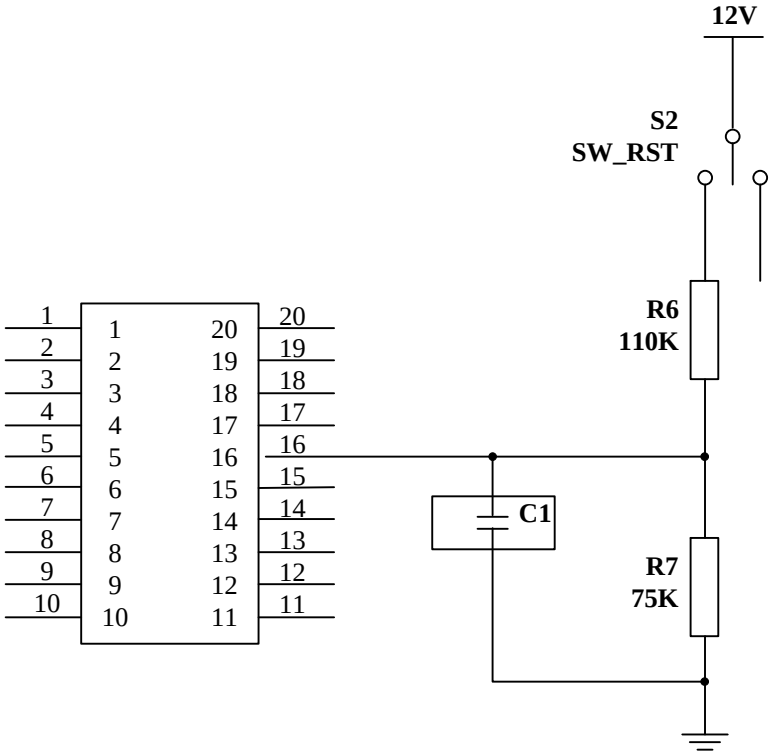
这时，$R_{弱}$一直处于为打开状态，除非 $R_{外}$电阻足够小，把 I/O 的电压拉到低电平，根据公式计算：

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

由于 $R_{弱}$较小，在（6KΩ，14KΩ）区间，因此一般来说 $R_{外}$<7 KΩ，才可把 I/O 的电压拉到低电平。

3.2.4 I/O设为准双向，实现电压检测的方法

见图：



说明：MCU I/O 设为准双向模式，一直设为输入状态。会出现误判现象：不管是否将 12V 电压切走，I/O Pin 可能一直检测为高电平。

分析：出现以上误判现象，是因为（由上 3.2.3 对 I/O 在准双向模式的理论分析可得知），当先将开关 S2 切到 R6 时，I/O 判高，即检测到 12V 电压，这一步判断正确；但是，当把开关 S2 切走时，由于之前 I/O 引脚为高，又将 I/O 设为输入（即口线寄存器为 1），此时 I/O 的上拉很小（R 极弱 || R 弱，6K~14K），而 I/O 外接的电阻 R7=75KΩ，则 R7 分压为高电平，I/O 还是判高，依然检测到 12V 电压，这一步判断错误。

那么，需要解决误判问题，成功正确检测电压的方法是：

第一种方法：

①. 采用小电阻方法，即将外接 R7 电阻阻值选取在 7K 以下，R7<7K（VDD＝5V））；（R6 随 R7 变化）；

②. MCU I/O Pin 一直写“1”，即设为输入口进行判断。

第二种方法：

①. 若外接电阻较大，R7>7K，则 R7 处要并联一个小电容 c=104；

②. 每一次判断电压前，都先对 MCU I/O Pin 写一次“0”；

③. 再对 MCU I/O Pin 写一次“1”，即设为输入口进行判断。

3.2.5 I/O设为准双向，实现过零检测的方法

过零检测，指的是当交流系统中，当波形从正半周向负半周转换时，经过零位时，系统作出的检测。可作开关电路（可控硅触发，继电器保护），频率控制，计时。一般的实现方法，见一下方框图：

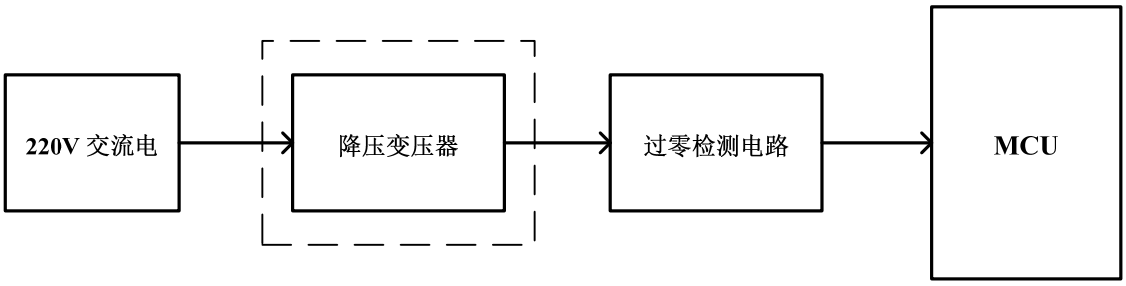


图 1　过零检测方框图

方法通常有两种：

第一种，220V 的交流电，经过一个降压变压器，间接提供交流点信号。举例说明见下：

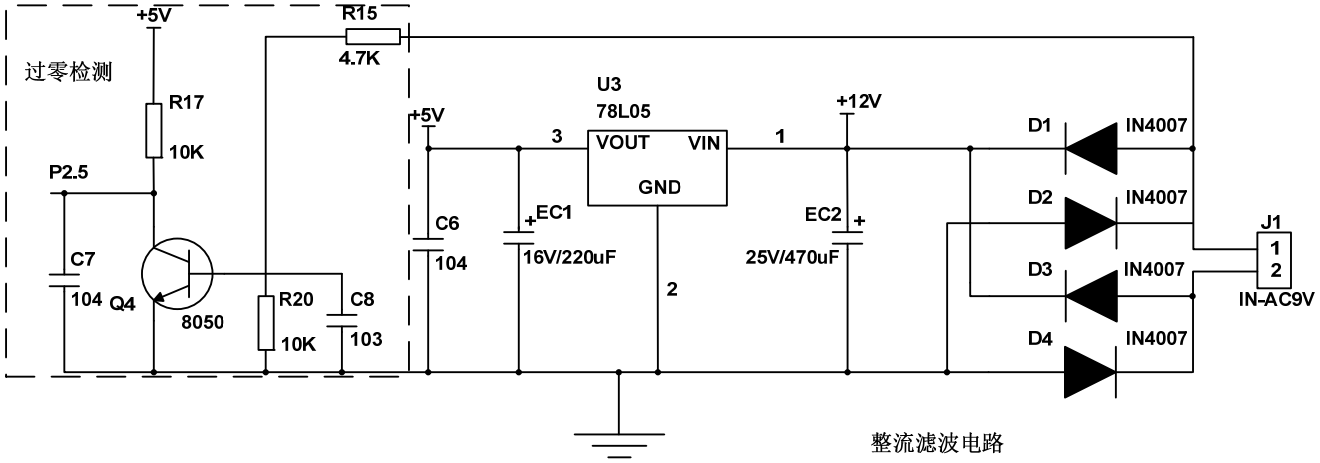


图 2　过零检测电路 1

以上电路，是将 220V 交流电经变压器降压成 9V 的交流电，给过零检测电路提供同步的交流信号。这种方法的特点：相对安全，可靠，但元件多，成本较高。

第二种，220V 交流电直接给过量检测电流提供信号。见下图：

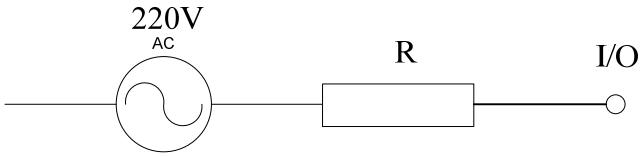


图 3　过零检测电路 2

这种方式比较简单，元件少，成本低。本小节主要是针对这种方式来展开讨论。

◆ **过零检测，个例分析**

采用上图 3 即“过零检测电路 2”的接法，以 SC91F72 做母体，具体做法如下：

第一，　若 R=100KΩ 时，MCU IO 设为准双向模式，IO 设为输入状态。则 I/O 的波形，见下图：

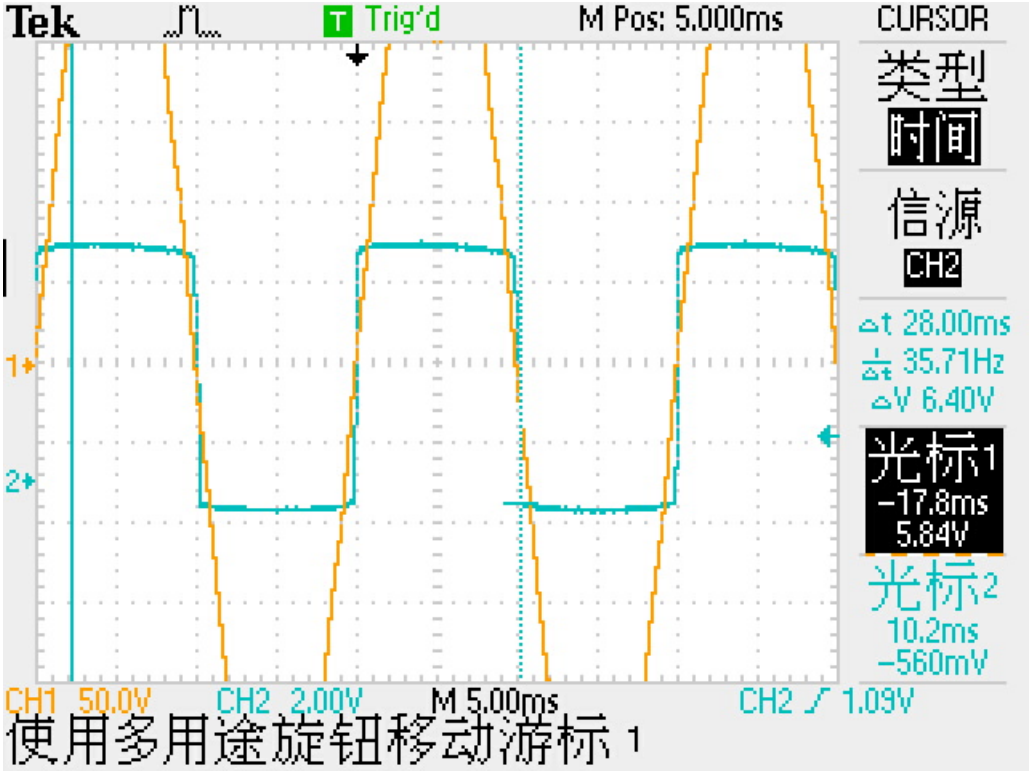


图 4　R=100KΩ，I/O 截取波形

说明：蓝色波形为 I/O 的波形，波形不对称，正半周时间为 10.2ms，负半周时间为 9.8ms；黄色波形为 50Hz，220V 的市电波形，半周时间为 10ms。

由上可知，当采用 R =100KΩ 时，过零检测的时间会有 t=（10.2ms-10.0ms）=200us 的误差，若客户采用该电路，需要注意该误差是否能够满足系统需求。

第二， 若 R=1MΩ 时，MCU IO 设为准双向模式，IO 设为输入状态。则 I/O 的波形，见下图：

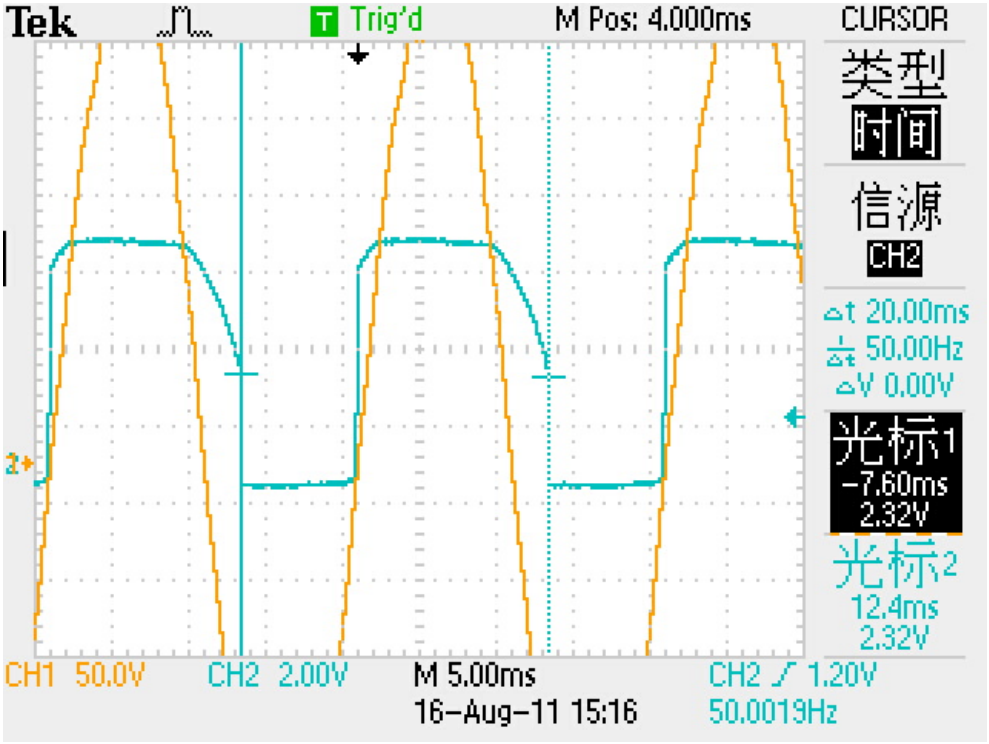


图 5　R=1MΩ，I/O 截取波形

说明：蓝色波形为 I/O 的波形，波形完成不对称，上正半周时间为 12.4ms，下负半周时间为 7.6ms；

由上可知，当采用 R =1MΩ 时，过零检测的时间会有 t=12.4ms-10.0ms=2.4ms 的误差，误差较大，完全不能满足系统要求；

R 取值越大，则 t 会越大，过零检测的误差就越大。

因此说，过零检测，若是采用 I/O 设为准双向的模式来实现，R 的选取不能太大， 否则过零检测的时间误差会很大，根本不可能满足系统要求。对于 SC91F72，通常的选取规则可为：R<=100KΩ，但同时需要考虑误差能否满足系统的要求。

以上使用第一，第二种方法来进行过零检测，**扫描 I/O 的高低电平变化，实现检测每一次的过零点**。这两种方法都存在不足：

a. 无论是第一种方法，还是第二种方法，过零检测的误差都比较明显，尽管将 R=100K Ω，还是存在 200us 的误差；

b. 尽管 R 取值较小，I/O 检测波形的对称性会更好，但是会造成电流过大，以 R=100K Ω 为例，最高的电流 I_{max}=311V/100K=3.1mA，平均电流 $I_{average}$=2.2mA。

综上所述，为避免第一，第二种方法的不足，可以采用检测上升沿（共 GND）或下降沿（共 VDD）的方法，再结合母体 SC91F72 的 IRC 高精度的特点，来实现过零检测的功能。

第三，　**系统共 GND，检测上升沿的方法**，见下图：

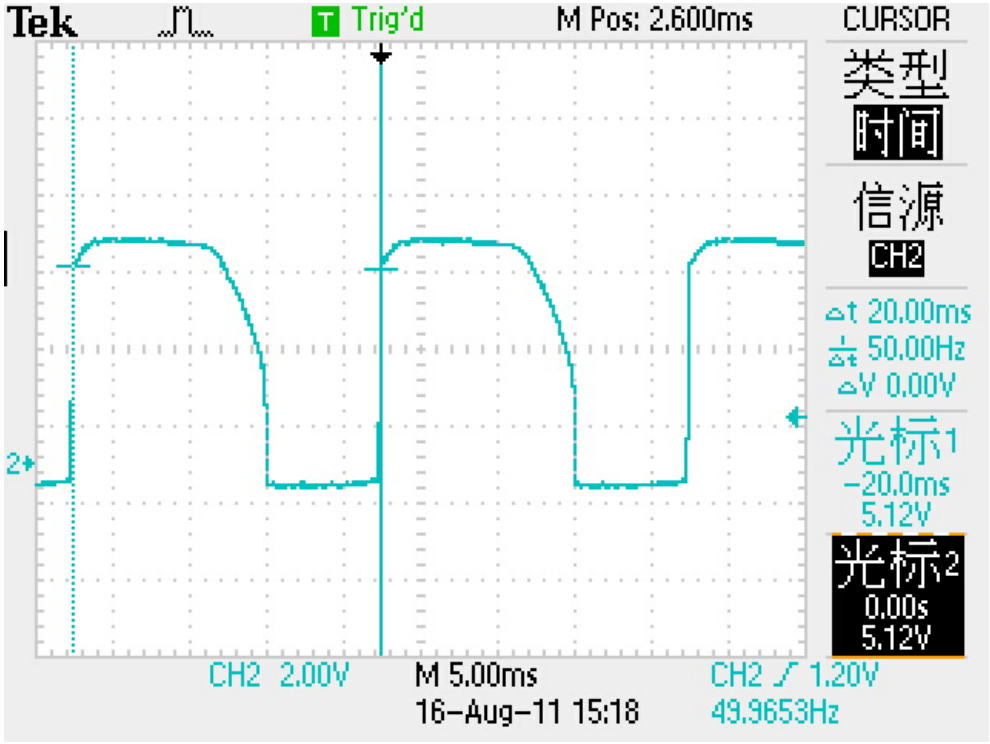


图 6　检测单片（上升沿，共 GND）

说明：见上图 6，每两个上升沿的时间差都是 20ms（交流电信号为 220V，50Hz），与 R 的取值无关，（每个上升沿的时间点与过零点的时间差非常小，几乎为 0）。又 SC91F72 的 IRC 的精度很高，在全温（-40^{0}C ~ 85^{0}C）全电压（3.6V~5.5V）条件下，精度误差在 2%以内，在常温 25^{0}C，5V 的条件下，精度误差只有 0.5%左右，那么计算 10ms 的时间误差在（60us-200us）范围。

具体做法是：每检测到一个上升沿，即为一次过零点，延时 10ms 即为下一个过零点，如此循环。这种做法可以结合 timer 的计时来实现。

这种方法的优点：过零检测的时间误差较小，在 60US-200US 之间；与 R 的取值无关，R 值可取较大值，

譬如 R=2M（过零检测电路的工作电流小）。

注意: 采用图 8 电路来实现过零检测，为了保护 I/O Pin 免受浪涌电压或者 HF-noise 的冲击而造成 EOS 损坏，一般会在 MCU I/O Pin 上分别接 PN 结二极管：一个与 VDD 相接，另一个与 VSS 相接。

此外，过零检测，还可以采用 ADC 的方法来实现，在此不做赘述。

3.2.6 I/O设为准双向，实现LCD应用的方法

本小节主要是针对 1/2BIAS 的 LCD 进行说明。

SC91F72/SC91F719 的 GPIO 可选择准双向模式和强推挽模式。当需要 GPIO 驱动 LCD，实现 LCD 的 1/2BIAS 偏置电压时，可选择准双向模式，通过相应的电阻分压(图 13)实现。

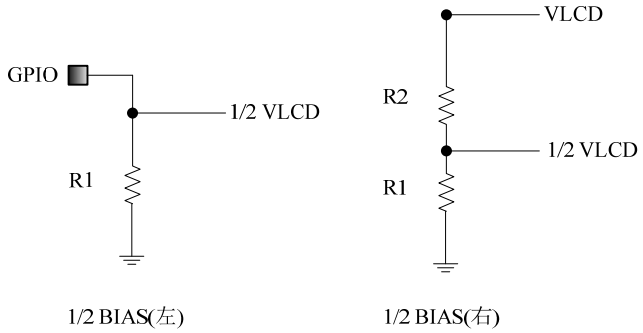


图 7　电阻分压结构图

显然，图 7 中给出 2 种不同方式的电阻分压结构，图(左)，仅接下拉电阻 R1，利用准双向 IO 的内部上拉电阻与外接的下拉电阻进行分压，实现 1/2BIAS 的偏置电压；图(右)，接上拉电阻 R2 和下拉电阻 R1，同样实现 1/2BIAS 的偏置电压。

当采用图 7(右)接法驱动 LCD 时，需要注意：

1，电阻选择：上拉电阻 R2 选择 5.1K，下拉电阻 R1 选择 3.3K。以 VLCD=5.0V 为例，实际测试 1/2BIAS 的偏置电压为:2.48V。特别指出：所接上拉电阻与下拉电阻不能等值；因为准双向 IO 的内部有弱上拉电阻，所以接上拉电阻与下拉电阻不宜过大，否则内部上拉电阻对分压有影响。

2，驱动波形：驱动波形的输出通常有 2 种扫描方式，4 种扫描模式(图 8)：扫描方式一，一个周期内，COM 口连续一次反相；扫描方式二，一个周期内，每半周期反相一次。如果希望驱动波形更加平整，可在被选择作电阻分压的 COM 口与 VSS 之间加 102 电容。

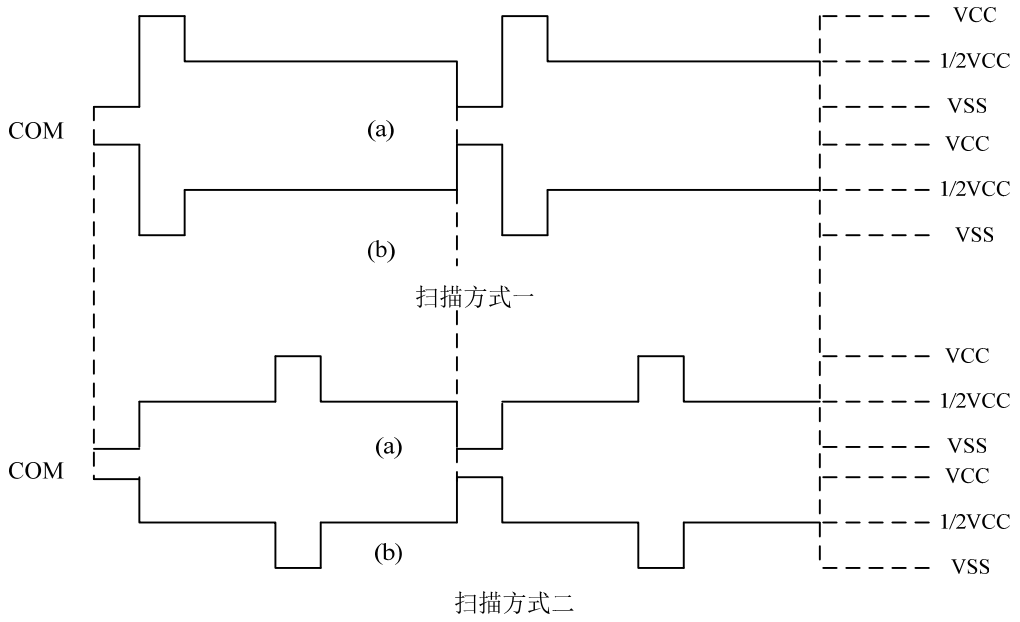


图 8　扫描方式

3，端口设置：按表 1 设置驱动 LCD 端口。

设置模式	操作	Vout
准双向模式	写“1”	1/2VCC
强推挽模式	写“1”	VCC
	写“0”	VSS

表 1　驱动 LCD 端口设置

当采用图 7(左)接法驱动 LCD 时，需要注意：

①. 顺电阻选择：建议实际应用中采用 10K~12K 下拉电阻，以实现最佳效果。以 VLCD=5.0V 为例，实际测试选择 10K、11K、12K 分别对应 1/2BIAS 的偏置电压为:2.48V、2.50V、2.52V。

②. 驱动波形：当采用仅接下拉电阻方式时，只能选择扫描方式一(a)，其余 3 种模式均不能选择。同样，希望驱动波形更加平整，可在 COM 口与 VSS 之间加 102 电容。

③. 端口设置：设置方法见表 1。

比较上述 2 种电阻分压结构，**图 7(左)接法相比图 7(右)的 ss 经济划算**，但是只有一种扫描模式；而图 7(右)花费稍高，但有 4 种扫描模式。两种方式实现的驱动波形效果都是一样的。

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

```
/*****************************************************************************
                        SC91F72 GPIO 的使用
 文件名：IO.c
 功能说明：SC91F72 GPIO 的输入、输出及其强推挽使用
*****************************************************************************/
#include "SC91F72_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      =       0x04;       //P1.0 切换为 GPIO,取消其复位功能

        FP1SO &=    0xFC;               //FP1SO=_b11111100，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

        FP2SO =     0x03;               //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

```
/*********************************************************************
                        SC91F72 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 90 个 ADC CKL
*********************************************************************/
#include "SC91F72_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=0x04;                //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

```
/*****************************************************************
                        SC91F72 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"SC91F72_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=0x04;                //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

```
/*************************************************************
                        SC91F72 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 "SC91F72_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=0x04;                    //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

```
/*********************************************************************
                        SC91F72  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 "SC91F72_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=0x04;                   //P1.0 切换为 GPIO,取消复位功能
        timer0_init();
        PWM_init();
        EA=1;
        while(1);
}
```