SCMCU-ICE 使用手册
V1.0

注意

使用手册中所出现的说明在出版当时相信是正确的，然而芯联发公司对于说明书的使用不负任何责任。文中提到的应用目的仅仅是用来说明，芯联发公司不保证或表示这些没有进一步修改的应用是适当的，也不推荐它的产品使用在会由于故障或其他原因可能会造成人身危害的应用。芯联发公司产品不授权使用于救生、维生器件等系統中作为元件。芯联发公司拥有不事先通知而修改产品的权利。

1. 概述

SCMCU-ICE 是芯联发公司开发的用来对芯联发单片机进行仿真的工具。其主要特点如下：

硬件

- 外观小巧，便于携带和存放
- 经由 USB 接口与电脑连接，采用 USB 自带协议，无需安装其它 USB 驱动
- 使用“在线仿真”模式，仿真时需要连接芯片管脚，仿真结果就是芯片运行的结果，可以仿真触摸按键等微量变化的功能

软件

- 可视化窗体结构，友好的操作界面
- 需要建立工程文件编译，方便程序的管理
- 语法着色及自动缩进风格使编辑操作更加简单
- 直接生成包含配置选项的烧录文件（.scx），生产时无需重复设置

编程

- 共 49 条指令结构，所有指令为 1-2 个指令周期
- 支持头文件定义(.H 文件)，支持 INCLUDE 文件功能，支持宏定义等功能
- 可自动分配寄存器(定义时用 “?” 来代替地址)
- 支持二种位定义

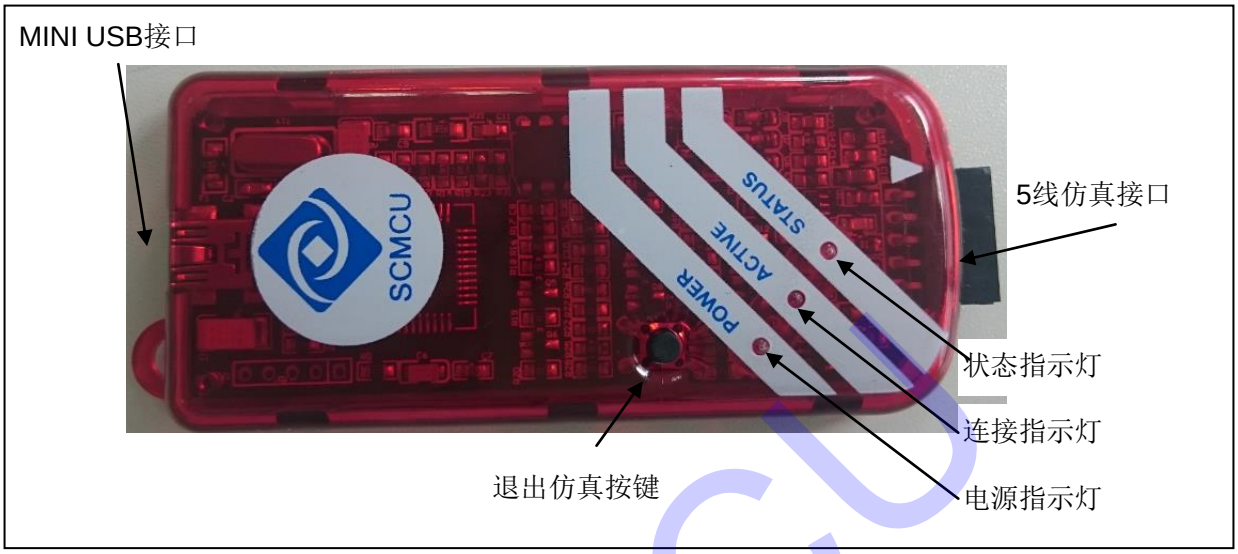
关于生成烧写文件的说明

- 仿真软件生成的”.scx”文件保存了 ROM 数据、芯片型号跟 Config 选项，烧写软件可直接打开该文件进行烧录。

2. 仿真器硬件

2.1 硬件说明

SCMCU-ICE的硬件外观如下图：



说明：

MINI USB接口：通过MINI 接口的USB线连接到电脑；

退出仿真按键：在仿真状态时，按下按键，仿真器退出仿真态；

电源指示灯：该指示灯为绿色，当仿真器通电时，指示灯常亮；

连接指示灯：该指示灯为蓝色，当仿真器连接到电脑时，指示灯常亮，在连接的过程中指示灯闪烁；

状态指示灯：该指示灯为红绿双色灯，在下载程序的过程中，绿灯闪烁，下载完成后绿灯常亮；执行连续运行时红灯常亮，在执行单步命令时红灯闪烁一次。

5线仿真接口：通过排线连接至芯片，5根线的功能从上（有个箭头标记）到下依次为：VDD、GND、VPP、DAT、CLK。

2.2 典型的在线编程连接方式

一般情况下，在使用SCMCU-ICE进行在线编程时，只需要将5跟编程线连接至芯片，而不用增加任何其它元件，如图2-1和图2-2所示：

图2-1 SC8F281X在线仿真连接图

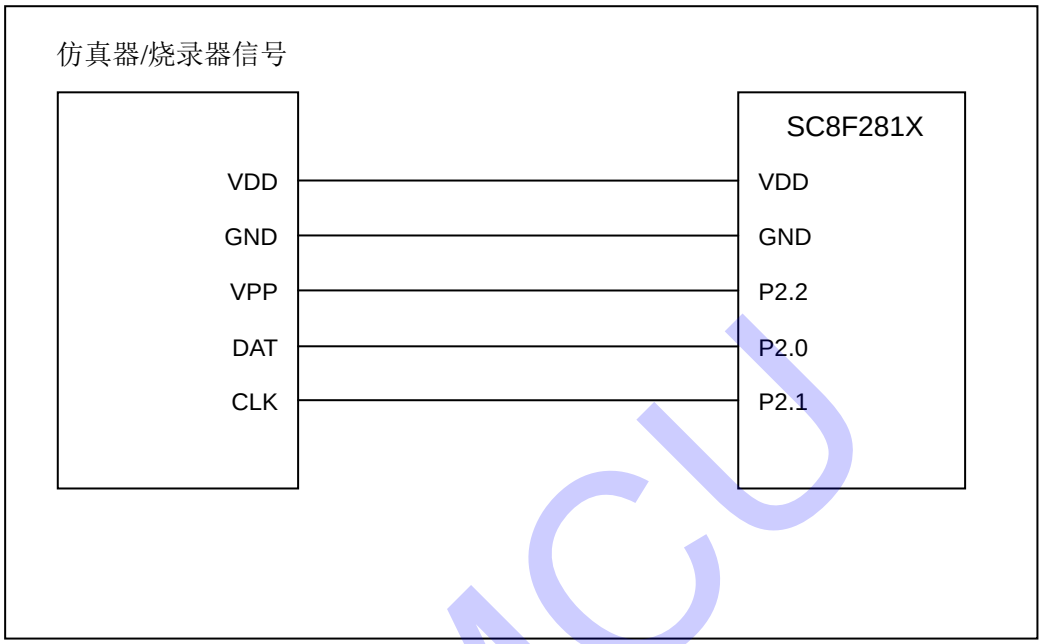
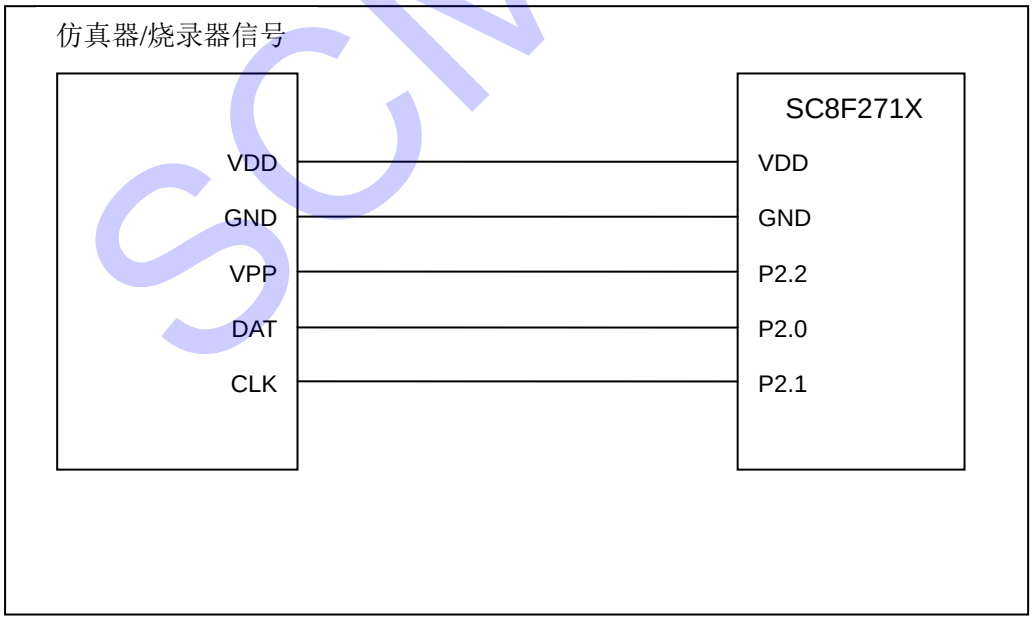


图2-2 SC8F271X在线仿真连接图

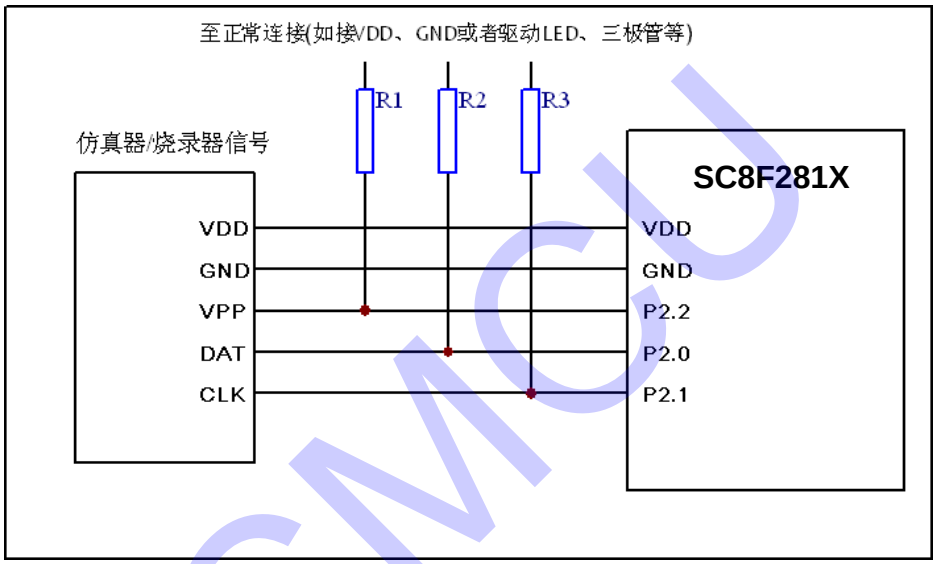


2.3 烧写管脚做其它用途时的连接方式

如果在应用中将DAT、CLK、VPP用作其它用途，那么需要在这些口线和相应的电路之间进行电气隔离。常用的方法是：

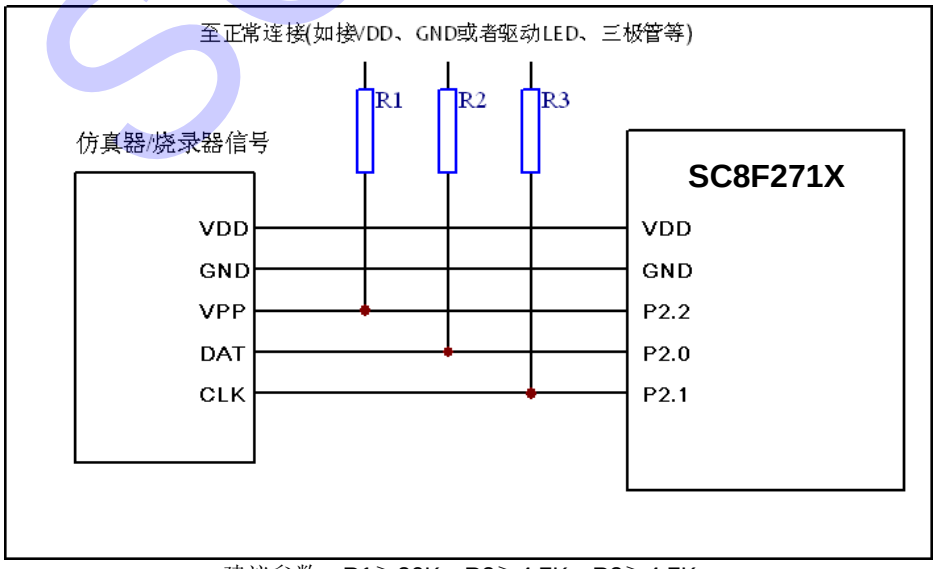
- DAT、CLK口线上串联一个大于4.7K的电阻。
- DAT、CLK口线上不能有大于100PF的电容。
- VPP口线上串联一个大于47K的电阻。
- 如果仿真器和芯片间的连线比较长，可以在VPP管脚上,靠近芯片的地方，加一个对地的电容，建议参数为104。

图2-3 SC8F281X的烧写口作为其它用途时的连接图



建议参数：R1≥30K，R2≥4.7K，R3≥4.7K

图2-4 SC8F271X的烧写口作为其它用途时的连接图

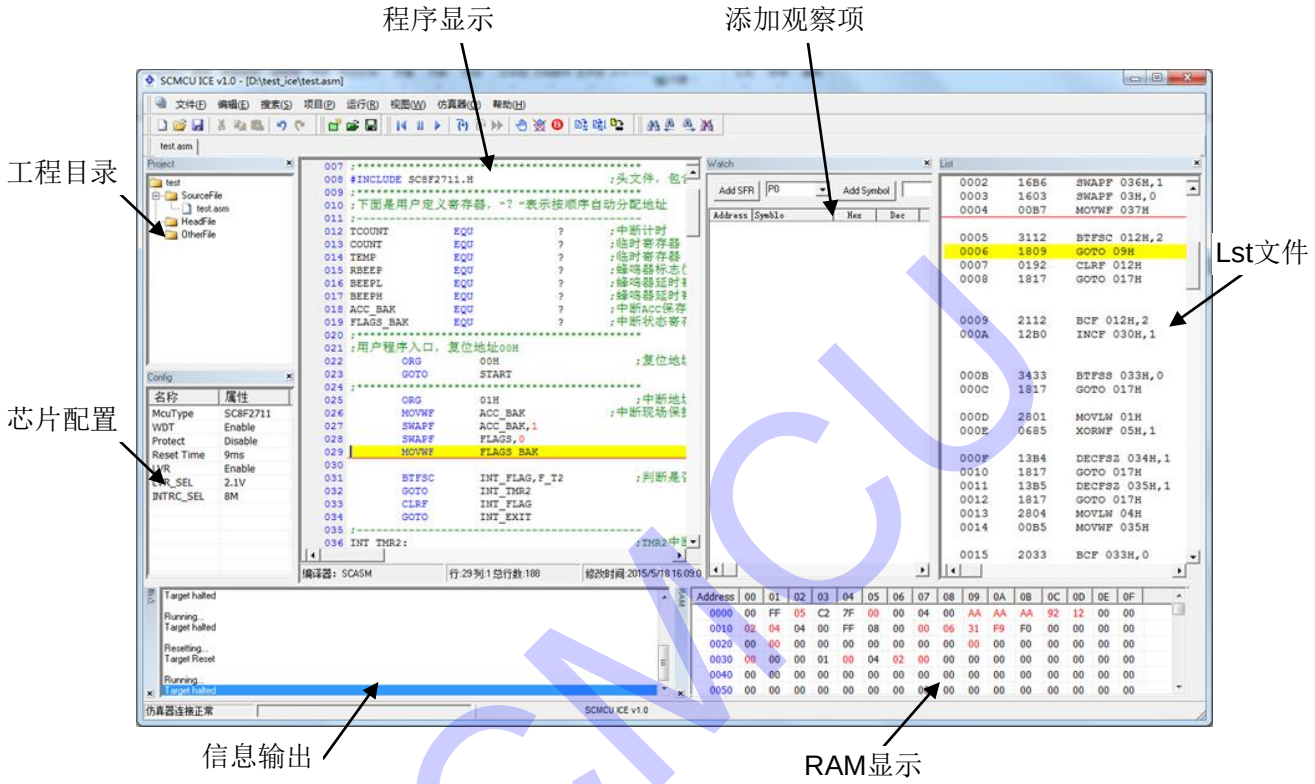


建议参数：R1≥30K，R2≥4.7K，R3≥4.7K。

3. 仿真器软件

3.1 仿真软件界面说明

图3.1 仿真软件界面图



界面说明：

工程目录：显示当前工程所包含的文件，其中源程序文件放在目录 SourceFile 里面，头文件(.H)放在目录 HeadFile 里面，其它文件放在目录 OtherFile 里面。

芯片配置：显示当前所选择的芯片型号及 Config 配置。

程序显示：显示程序内容，双击工程目录中的文件名，可直接打开该文件，并在该窗口显示。

添加观察项：显示用户定义的观察项，包括系统寄存器可用户定义寄存器。

Lst 文件：显示编译完成后的 lst 文件，可查看机器码及程序所存放的 ROM、RAM 地址。

信息输出：显示操作结果，包括 USB 连接、编译结果等内容。

RAM 显示：显示所有 RAM 值。

3.2 快速开始

步骤 1：建立工程并添加文件

- 点击菜单中的“项目”选项并选择“新建工程”命令
- 输入工程名和工程路径，
- 选择芯片型号及 Config 配置
- 点击菜单中的“项目”选项并选择“添加文件至工程”命令
- 将需要的文件依次添加到工程中

步骤 2：设置编译语言及供电方式

- 点击菜单中的“运行”选项并选择“选择电源”命令，选择内部电源或者外部电源

步骤 3：编译调试

- 点击菜单中的“项目”选项并选择“编译”命令
- 当编译完成时，观察信息输出窗口显示编译成功(Compile Successful)或者编译失败(Compile Failed)
- 如果编译失败，双击失败语句，查找失败原因并修正错误，直至编译成功
- 点击菜单中的“项目”选项并选择“编译并下载”或者选择“下载到仿真器”命令，将代码下载至芯片，开始仿真
- 点击菜单中的“运行”选项并选择相应的命令，如“运行”、“单步”、“复位”等命令进行调试

3.3 菜单说明

文件（F）

文件 | 新建文件
新建一个空白文件
文件 | 添加新文件至工程
新建一个空白文件，并添加至当前工程中
文件 | 添加文件至工程
添加一个已经存在的文件至当前工程中
文件 | 打开文件
打开一个用户程序，进行编辑，
文件 | 关闭文件
关闭当前焦点文件
文件 | 保存文件
保存当前焦点文件
文件 | 文件另存为
重新保存文件为其他文件，原来文件不变
文件 | 保存所有文件
保存当前打开的所有文件
文件 | 最近打开文档
这里保存用户最近打开的 5 个文件列表
文件 | 最近打开工程
这里保存用户最近打开的 5 个工程列表
文件 | 退出
关闭仿真软件

编辑（E）

编辑 | 撤销
取消上一次操作
编辑 | 重做
恢复被取消的操作
编辑 | 剪切
删除选定的文本，删除内容赋值到剪贴板上
编辑 | 复制
将选定的内容，复制到剪贴板上
编辑 | 粘贴
将剪贴板上内容插入光标位置
编辑 | 删除
删除选定文本
编辑 | 全选
选定当前窗口所有内容

搜索（S）

搜索 | 查找
在当前窗口文件中查找字符串
搜索 | 下一个
查找下一个字符
搜索 | 上一个
查找上一个字符
搜索 | 清除查找结果
查找结束后，清除查找标记
搜索 | 替换
在当前窗口文件查找字符，并替换成指定字符
搜索 | 转到指定行
将光标跳转到程序某一行
搜索 | 转到指定地址
将光标跳转到程序指定地址
搜索 | 转到当前 PC 所在行
将光标跳转到当前 PC 所在程序位置

项目（P）

项目 | 新建工程
建立一个新的工程
项目 | 打开工程
打开一个已经存在的工程
项目 | 关闭工程
关闭当前打开的工程
项目 | 保存工程
保存当前打开的工程
项目 | 工程另存为
重新保存工程为其它工程，原工程不变
项目 | 添加文件至工程
添加一个已经存在的文件至当前工程中
项目 | 添加新文件至工程
建议一个新的文件，并添加至当前工程中
项目 | 最近打开工程
这里保存用户最近打开的 5 个工程列表
项目 | 编译
编译当前打开的工程并保存，如果有错误，系统会提示错误所在位置
项目 | 编译并下载
编译当前打开的工程并保存，如果有错误，系统会提示错误所在位置，如果没有错误，系统会将代码下载到仿真器
项目 | 下载到仿真器
将编译完成的代码下载到仿真器

运行（R）

运行 | 复位

终止调试过程，程序将被复位

运行 | 暂停

暂停正在全速运行的程序

运行 | 运行

运行程序，如果程序运行到断点位置，将自动停止运行

运行 | 单步运行

单步执行程序的每一步，观察程序运行状态

运行 | 忽略断点运行

全速执行程序且不响应断点

运行 | 设置/取消断点

将光标所在行设为断点，如果该行已经是断点，则取消该断点

运行 | 清除所有断点

清除程序中所有断点

运行 | 选择电源

选择仿真目标板供电方式，仿真器内部供电或者外部供电

视图（W）

视图 | 工程文件

打开/关闭工程目录窗口

视图 | 配置选项

打开/关闭芯片配置显示窗口

视图 | 信息

打开/关闭信息输出窗口

视图 | 观察变量

打开/关闭观察变量窗口

视图 | 局部变量

打开/关闭局部变量窗口(只有 C 语言有效)

视图 | List 文件

打开/关闭 List 文件窗口

视图 | 寄存器

打开/关闭寄存器显示窗口

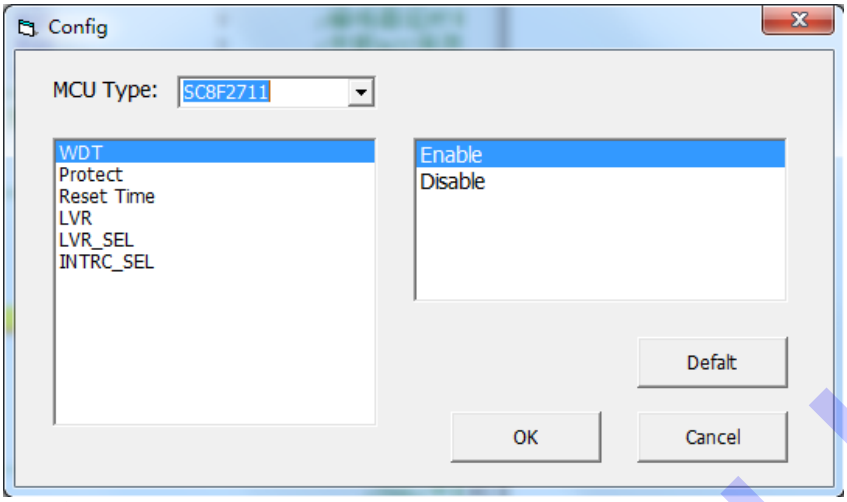
视图 | 工具栏

打开/关闭各工具栏。包括 文件编辑、运行调试、文件查找、文件列表等工具栏。

仿真器（O）

仿真器 | 仿真器设置

打开仿真器设置对话框进行 Config 设置



常用选项说明：

芯片型号：仿真器所仿真的芯片型号选择

WDT：看门口使能：Enable 看门狗打开; Disable 看门狗关闭。

Protect：芯片加密选择：Disable 不加密; Enable 加密。

Reset Time：复位时间选择，表示芯片的起振时间。

LVR：低压侦测使能选择：Enable 打开低压侦测,复位口可作为普通输入口; Disable 关闭低压侦测，复位口在低电平的时候芯片复位。

LVR_SEL：LVR 电压选择。

RES_OUT：复位口开漏输出功能选择。

帮助（H）

帮助 | 关于 SC-ICE

打开关于 SCMCU-ICE 对话框

帮助 | 程序范例

打开程序范例

帮助 | 文档

打开文档

4. 程序调试

4.1 定义

SCMCU-ICE 编程定义包括：

- 寄存器（RAM）定义
- 常数定义
- 位（BIT）定义
- 地址（ROM）定义

标签符号只能由字母、数字、下划线组成且第一个字符不能是数字，标签定义不能重复。

4.1.1 寄存器定义

寄存器包含专用寄存器和通用寄存器。专用寄存器的定义必须使用关键字“EQU”，并指明地址，如：

```
IAR         EQU     00H
PORTA       EQU     05H
```

请在源程序中尽量用 SCMCU-ICE 标准头文件定义的寄存器符号。一来这些被定义的寄存器符号和芯片数据手册上的描述一一对应，理解起来即直观又容易；二来如果用你自己定义符号就缺乏一个大家能一起交流的标准平台，其他人要解读你的代码时将费时费力。

如果全部使用标准头文件定义，用户可在程序中用 “#INCLUDE”来载入头文件，如:

```
#INCLUDE SC8F2711.H
```

通用寄存器的定义必须使用关键字“EQU”，可以自己指明地址或者由系统自动分配地址。例如：

- 用户指定地址： WORK EQU 30H
- 系统自动分配地址： WORK EQU ?

4.1.2 常数定义

常数定义必须使用关键字“EQU”或者“DEF”。例如定义常数 NUMBER，其值为 50H：

```
NUMBER      EQU     50H
NUMBER      DEF     50H
```

4.1.3 位定义

位定义有 2 种定义方式：

1：只定义位，不定义地址，如：

```
BIT0        EQU     0
```

程序中在使用 BIT0 这个位时，可以用以下语句：

```
BCF     30H,BIT0        ;表示将 30H 地址的第 0 位置“0”
BTFSC   50H,BIT0        ;判断 50H 地址的第 0 位，为“0”间跳
```

2：直接定义某寄存器的某一位，如：

```
BIT0        EQU     30H,0
```

程序使用时，可以使用以下语句：

```
BCF     BIT0        ;表示将 30H 地址的第 0 位置“0”
BTFSC   BIT0        ;判断 30H 地址的第 0 位，为“0”间跳
```

4.1.4 地址定义

地址定义必须是单独的一行，标签后面可以加“：”。如

```
        START
或      START:
```

程序使用时，可以直接调用该标签，如：

```
GOTO    START       ;跳转至 START 定义的地址
CALL    START       ;调用 START 子程序
```

如果程序中需要用到所定义地址的高位或低位，比如在使用表格命令时，需要将地址高位赋给表格高位寄存器，将低位地址赋给表格低位寄存器，此时可以再定义的地址后增加“$H”表示该地址的高位，在地址后增加“$L”表示该地址的低 8 位，以下程序表示了如何使用定义表格的高位和低位：

```
TABLE_SEG:                          ;定义表格标号，地址不确定
            DW      1234H           ;表格第 1 行内容
            DW      5678H           ;表格第 2 行内容
            …                       ;表格其它行
            …
      XXXX:
            MOVLW   TABLE_SEG$H     ;表格高位地址给 ACC
            MOVWF   TABLE_SPH       ;表格高位地址给高位指针 TABLE_SPH
            MOVLW   TABLE_SEG$L     ;表格低位地址给 ACC
            MOVWF   TABLE_SPL       ;表格低位首地址给高位指针 TABLE_SPL
            MOVF    R01,0           ;R01 表示需要调用表格第几行
            ADDWF   TABLE_SPL,1     ;实际需要的表格地址
            BTFSC   STATUS,C        ;如果有进位，则高位地址加 1
            INCF    TABLE_SPH,1
            TABLEA                  ;调表格指令
            MOVWF   TEMPL           ;把表格低位值赋给寄存器 TEMPL
            MOVF    TABLE_DATAH,0
            MOVWF   TEMPH           ;把表格高位值赋给寄存器 TEMPH
```

4.2 数字

SCMCU-ICE 可以识别十六进制，十进制、二进制 3 种格式的数字输入。

4.2.1 十六进制

在 SCMCU-ICE 中，默认的数字格式为十六进制，用户可以在数字后加“H”或在数字前面加“0x”用以区分其它格式，如十六进制的 10(10 进制为 16)，可以写成 10、10H 或 0x10。如以下语句：

```
MOVLW   10
MOVLW   10H
MOVLW   0x10
```

都表示的是将十六进制数“10”赋给累加器 A。

若所操作十六进制数最高位值大于 9，则必须在其前面加 0。如将十六进制数“A5”赋给累加器 A

错误的表示方法: MOVLW A5H

正确的表示方法: MOVLW 0A5H

4.2.2 十进制

在 SCMCU-ICE 中，十进制有 2 种表示方法：D‘i’ 或者 .i ，如

```
MOVLW   D‘10’
MOVLW   .10
```

都表示将十进制数“10”赋给累加器 A

4.2.3 二进制

在 SCMCU-ICE 中，二进制只有一种表示方法：B‘i’ 如：

```
MOVLW   B‘00010000’
```

表示将二进制数“00010000”赋给累加器 A。

由于二进制数较长，为避免由于多输入或者少输入而造成编译器难以检测的错误，SCMCU-ICE 规定，所使用的二进制数必须为 8 位或 16 位。

4.3 宏定义

SCMCU-ICE 允许用户使用宏定义，宏定义是定义一个名称来代表一段指令，而在源程序中可以重复使用此名称以取代这段程序，也就是程序中所有用到此段程序的地方皆可用此名称代替之。在编译时，编译器会自动将每一个宏定义的名称用其所定义的程序来取代。

在源文件的任何地方皆可定义宏，只要调用此宏的地方是在宏定义之后即可。宏不能嵌套，即在宏定义中不能调用其它宏。

宏定义格式：

name **MACRO** *temp1,temp2*

xxx

ENDM

name:表示定义的宏名称

MACRO:宏定义标号，有该标号才表示宏定义开始

temp1,temp2:宏参数，一个宏可以没有宏参数或者多个宏参数

xxx:宏内容

ENDM:宏结束

例：不带参数的宏

```
IO_CLR  MACRO                   ;定义一个名称为 IO_CLR 的宏
        CLRF    PORTA
        CLRF    PORTB           ;宏内容
        CLRF    PORTC
        ENDM                    ;宏结束
        …
        …
        IO_CLR                  ;程序中宏调用语句
        …
        …
```

例：带参数的宏

```
DELAY_M MACRO   TEMP            ;定义一个名称为 DELAY_M 的宏,带一个参数 TEMP
        DECFSZ  TEMP,1
        GOTO    $-2
        ENDM
        …
        MOVLW   .10
        MOVWF   DELAY_TIME
        DELAY_M DELAY_TIME      ;程序中宏调用语句,把寄存器 DELAY_TIME 地址赋给宏参数
        …                       ;TEMP
```

5. 指令

5.1 伪指令

伪指令可以增加源程序的可读性和可维护性。

以下是 SCMCU-ICE 所能识别的伪指令：

● # INCLUDE 或 INCLUDE

INCLUDE 伪指令的作用是把另外一个文件的内容全部包含复制到本伪指令所在的位置。被包含复制的文件可以是寄存器定义文件(“.H”文件)，也可以是原程序文件(“.asm”文件)。最经常被“INCLUDE”的文件是针对单片机内部特殊功能寄存器定义的包含头文件， 在 SCMCU-ICE 安装后它们全部放在安装目录下的“head”文件夹里，每一个型号的单片机都有一个对应的预定义包含头文件，扩展名是“.H”。

例如：

```
#INCLUDE SC8F2711.H
```

根据文件存放位置的不同，INCLUDE 伪指令有以下几种方式：

```
#INCLUDE    SC8F2711.H          表示在安装目录"HEAD"文件夹下的 SC8F2711.H 文件
#INCLUDE    < SC8F2711.H>       表示在安装目录"HEAD"文件夹下的 SC8F2711.H 文件
#INCLUDE    "USER.ASM"          表示与当前文件在同个目录下的 USER.ASM 文件
#INCLUDE    "D:\项目\USER.ASM"  表示在 D 盘项目文件夹里的 USER.ASM 文件
```

● EQU

EQU 顾名思义是“等于”的意思，是用一个符号名字替换其它数字变量，但它只能替换立即数。如果要替换一个符号名字，则此符号名必须事先用 EQU 伪指令已经定义替换了一个立即数。

例如：

```
WORK    EQU    30H        ;定义 WORK 符号替换立即数 30H
TEMP    EQU    WORK       ;符号名 TEMP 等于 WORK(如果 WORK 没有定义则会产生一个错误)
```

在编程模式中 EQU 被经常用于定义用户自己的变量，即用一个符号名代替一个固定的存储单元地址，上例中的 WORK 定义即属于此类。用 EQU 方式定义的符号在汇编后可以生成相关的调试信息，可以通过各种变量观察的方式显示此符号所代表的内存地址处的数据内容。要注意 EQU 伪指令本身并没有限定所定义的一定是一个变量地址，它只是一个简单的符号和数字替换而已，其意义必须和具体的指令结合才能确定，如下例中对符号 WORK 的理解。

```
WORK    EQU    30H        ;符号名 WORK 等于 30H
MOVLW   10H               ;A=10H
MOVWF   WORK              ;把 A 的值送给变量 WORK，（30H 单元内容＝10H）
```

● END

END 伪指令告诉汇编编译器编译工作到此为止，END 后面所有的信息，不管正确与否，一概不管。绝大多数情形下你的程序的最后一行应该是“END”。如果没有，编译器会默认你的程序最后一行为结束。

● DEF

DEF 伪指令可用来做常量的定义。用 EQU 定义的标号会占用 RAM 空间，而用 DEF 定义的标号不会占用 RAM 空间。如下例所示：

```
NUMBER1   DEF     20H     ;定义常量 NUMBER1 为 20H
MOVLW     NUMBER1         ;把 20H 赋给 ACC
```

用 DEF 定义的常量不能作为 RAM 用，如以下语句是错误的：

```
MOVWF       NUMBER        ;错误的语句，寄存器只能用 EQU 定义
```

● ORG

ORG 用以定义程序代码的起始地址，通过此伪指令你可以把程序定位到任何可用的程序空间，它实现的是程序代码绝对定位。

如例：

```
ORG         00H           ;定义复位入口地址，以下指令从地址 0x0000 开始
GOTO        START
ORG         01H           ;定义中断入口地址，以下指令从地址 0x0001 开始
GOTO        INT_START
```

只要你认为代码需要确定放在某一特定地址处，在程序的任何地方都可以用 ORG 伪指令重新定义存放的起始地址，且地址顺序可以任意编排。但要注意的是若干个确定起始地址的代码块不能相互重叠，否则编译器会报错，无法得到正确结果。若用可重定位方式开发指令

● $

$符号表示获取取当前指令的地址值。

你可以在写程序时给一条指令前加上一个标号，然后直接引用该标而得到此程序字的地址。如果你的程序经常需要用到指令的当前地址或附近的地址值，这样的标号就需要写很多且不能重复。用“$”运算符让汇编器替你计算当前指令所处的位置将有效地减轻你的这份工作量。

如例：

```
GOTO      $+1       ;跳转到下一条地址
```

这条语句表示跳转到下一个地址，它跟 NOP 的区别是 NOP 需要 1 个指令周期，而 它需要 2 个指令周期。

在一个标号后加上“$H”或“$L”来获取该标号的高位或低位，如：

```
MOVLW       TABLE_SEG$H
MOVWF           ADDH
MOVLW       TABLE_SEG$L
MOVWF           ADDL
```

以上程序中，TABLE_SEG 为一个标号，程序运行结果是把 TABLE_SEG 所表示的地址高位赋给 ADDH 寄存器，低位地址赋给 ADDL 寄存器。

5.2 指令表

助记符	操 作	指令周期	标 志
控制类-4			
NOP	空操作	1	None
OPTION	装载 OPTION 寄存器	1	None
SLEEP	进入休眠模式	1	TO,PD
CLRWDT	清零看门狗计数器	1	TO,PD
数据传送-3			
MOVWF f	将 W 内容传送到 f	1	NONE
MOVF f,d	将 f 内容传送到目标寄存器	1	Z
MOVLW k	立即数 k 送给 W	1	NONE
逻辑运算-14			
CLRW	清零 W	1	Z
SETF f	置位数据寄存器 f	1	NONE
CLRF f	清零数据存储器 f	1	Z
TABLE f	读取 FLASH 内容结果放入 TABLE_DATAH 与 f	2	NONE
TABLEA	读取 FLASH 内容结果放入 TABLE_DATAH 与 W	2	NONE
DAW f	将加法运算中放入 W 的值调整为 10 进制数，并将结果存入 f	1	C
IORWF f,d	f 与 W 内容做“或”运算	1	Z
ANDWF f,d	f 与 W 内容做“与”运算	1	Z
XORWF f,d	f 与 W 内容做“异或”运算	1	Z
SWAPF f,d	f 寄存器内容的高低半字节转换	1	NONE
COMF f,d	f 寄存器内容取反	1	Z
XORLW k	W 与立即数 k 做“异或”运算，结果存入 W	1	Z
ANDLW k	W 与立即数 k 做“与”运算，结果存入 W	1	Z
ORLW k	W 与立即数 k 做“或”运算，结果存入 W	1	Z
移位操作，4			
RRF f,d	数据存储器带进位循环右移一位	1	C
RLF f,d	数据存储器带进位循环左移一位	1	C
RRNCF f,d	数据存储器不带进位循环右移一位	1	NONE
RLNCF f,d	数据存储器不带进位循环左移一位	1	NONE
递增递减，2			
INCF f,d	递增数据存储器 f	1	Z
DECF f,d	递减数据存储器 f	1	Z
位操作，2			
BCF f,b	将数据存储器 f 中某位清零	1	NONE
BSF f,b	将数据存储器 f 中某位置一	1	NONE
数学运算，9			
ADDWF f,d	W+f	1	C,DC,Z,OV
ADDLW k	W+k→W	1	Z,C,DC,OV
SUBWF f,d	f-W	1	C,DC,Z,OV
SUBLW k	k-W→W	1	Z,C,DC,OV

ADDWFC f,d	W+f+C→目标寄存器	1	Z,C,DC,OV
SUBWFB f, d	f-W-C→目标寄存器	1	Z,C,DC,OV
HSUBWF f,d	当 d=0 时，W-f→W； 当 d=1 时，W-f-C→W	1	Z,C,DC,OV
HSUBWFB f, d	W-f- $\overline{C}$ →目标寄存器	1	Z,C,DC,OV
HSUBLW k	W-k→W	1	Z,C,DC,OV
无条件转移，5			
RETURN	从子程序返回	2	NONE
RETLW k	从子程序返回，并将立即数 k 存入 W	2	NONE
RETFIE	从中断返回	2	NONE
CALL k	子程序调用	2	NONE
GOTO k	无条件跳转	2	NONE
条件转移，6			
BTFSC f,b	如果数据存储器 f 的 b 位为“0”，则跳过下一条指令	1 or 2	NONE
BTFSS f,b	如果数据存储器 f 的 b 位为“1”，则跳过下一条指令	1 or 2	NONE
INCFSZ f,d	数据存储器 f 加“1”，若结果为”0”，则跳过下一条指令	1 or 2	NONE
DECFSZ f,d	数据存储器 f 减“1”，若结果为”0”，则跳过下一条指令	1 or 2	NONE
TSTFSZW f	数据存储器 f 送至 W，若内容为”0”，则跳过下一条指令	1 or 2	NONE
TSTFSZ f	数据存储器 f 内容为”0”，则跳过下一条指令	1 or 2	NONE