



杭州晶华微电子股份有限公司
Hangzhou SDIC Microelectronics Inc.

基于晶华 SD8Link_VSCode 工程实操指南

文件编号:

编写人: 吴华桥

批准人:

版本号: v0.2

审核人: 陈纪彬

编写日期: 2025-06-03

目录

1.概述	4
2.软件安装.....	4
3.创建工程.....	4
4.编译工程.....	5
5.下载工程.....	6
6.调试工程.....	8
7. C 语言应用程序	9
7.1 .c 文件目录层级.....	9
7.2 CODE BANKING 分区	9
7.2.1 分区说明.....	9
7.2.2 分区应用.....	9
7.2.3 写 FLASH 数据	11
7.3 库函数使用	13
7.4 MAKEFILE 文件配置.....	13
7.5 中断函数.....	14
7.6 关键字与 KEIL 编译器的区别	15
7.6.1 关键字 sfr	15
7.6.2 关键字 sbit 和 bit.....	15
7.6.3 关键字 xdata.....	16
7.6.4 关键字 code	16
7.6.5 关键字 data.....	16
7.6.6 关键字 idata.....	16
7.6.7 关键字 pdata.....	17
7.6.8 关键字 interrupt.....	17
7.6.9 关键字 reentrant	17
7.6.10 关键字 at.....	17
7.7 C 语言程序如何嵌入汇编代码	18
8.汇编语言应用程序.....	18
8.1 创建汇编工程.....	18
8.2 配置汇编程序调试.....	20
8.3 汇编语法说明.....	21
8.3.1 文件结构.....	21
8.3.2 表达式.....	22
8.3.3 数字.....	23
8.3.4 预处理器.....	23
8.3.5 处理器头文件.....	23
8.4 汇编指令/伪指令说明	25
8.4.1 代码生成.....	25
8.4.2 条件汇编.....	26

8.4.3 宏定义.....	27
8.4.4 area 定义内存区域指令	27
8.4.5 ORG 汇编起始指令	28
8.4.6 include 预处理指令	28
8.4.7 equ 赋值指令	29
8.4.8 macro 和 endm 宏指令	29
8.4.9 DB 字节定义指令	30
8.4.10 DW 字定义指令	30
8.4.11 DS 存储空间预留指令	30
8.4.12 汇编位操作指令	30
8.4.13 VSCODE 中无法识别的汇编指令	31
8.5 跨文件汇编源程序	31
8.6 VSCODE 符号替换功能	31
8.7 错误(常见的错误)	31
8.7.1 重定位错误	31
8.7.2 汇编指令错误	32
8.7.3 操作、终止、分隔符错误	32
8.7.4 未定义错误	32
8.7.5 未定义内存区域错误	32
9.修改记录	33

1.概述

本文档以 SD82F466_C_Demo 工程为例,说明如何在 VSCode 中创建工程、编译、下载、调试等功能,以及使用注意事项。使用的工具主要包括: VisualStudioCode(以下简称 VSCode)、SD8Link_VSCODE、SD8Link 调试器。

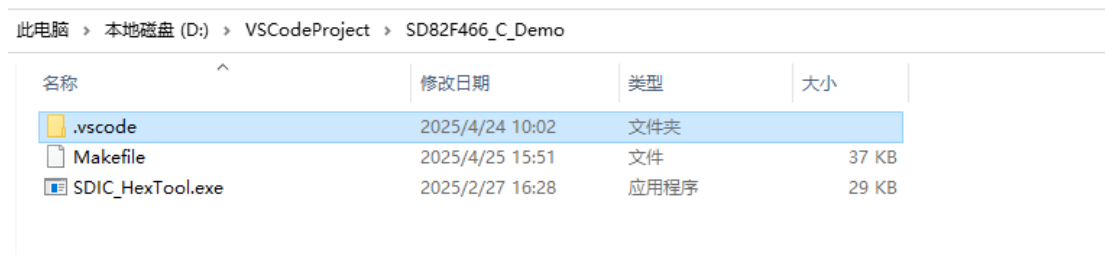
2.软件安装

步骤 1: 安装 VSCode 软件(推荐版本: VSCodeSetup-x64-1.97.2,如需下载 VSCode 软件安装, VSCode 软件要求大于 1.66 版本)。

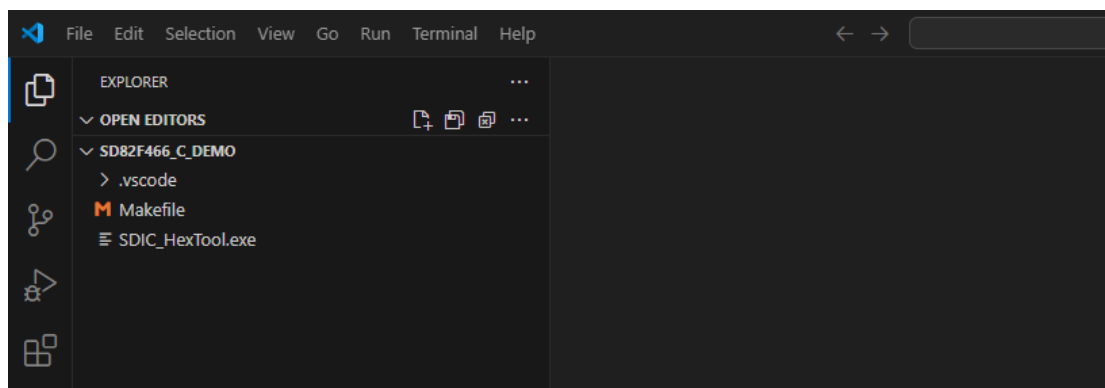
步骤 2: 安装晶华“SD8Link_VSCode_Setup_v0.exe”工具(管理员权限安装)。


3.创建工程

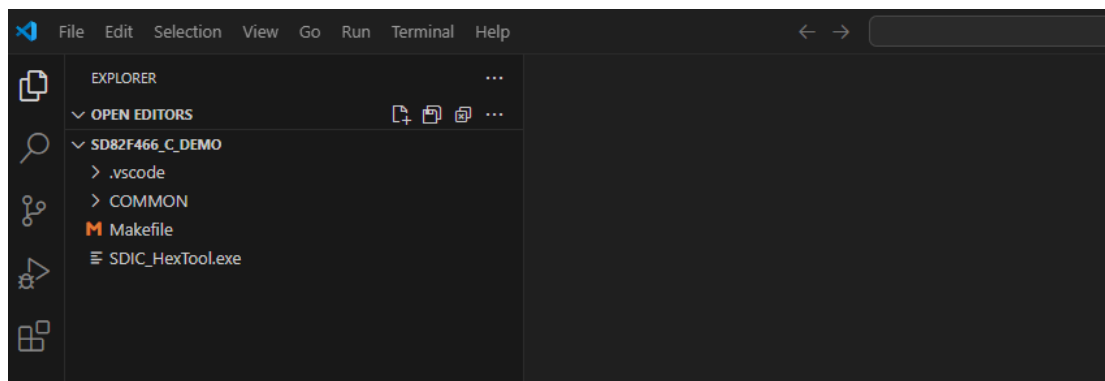
步骤 1: 在电脑指定位置创建工程目录文件夹,此处创建名为 SD82F466_C_Demo 工程目录文件夹(路径不能有中文)。将晶华提供的模板工程内的“.vscode”文件夹、makefile 文件、SDIC_HexTool.exe 拷贝进文件夹内,如下图所示。




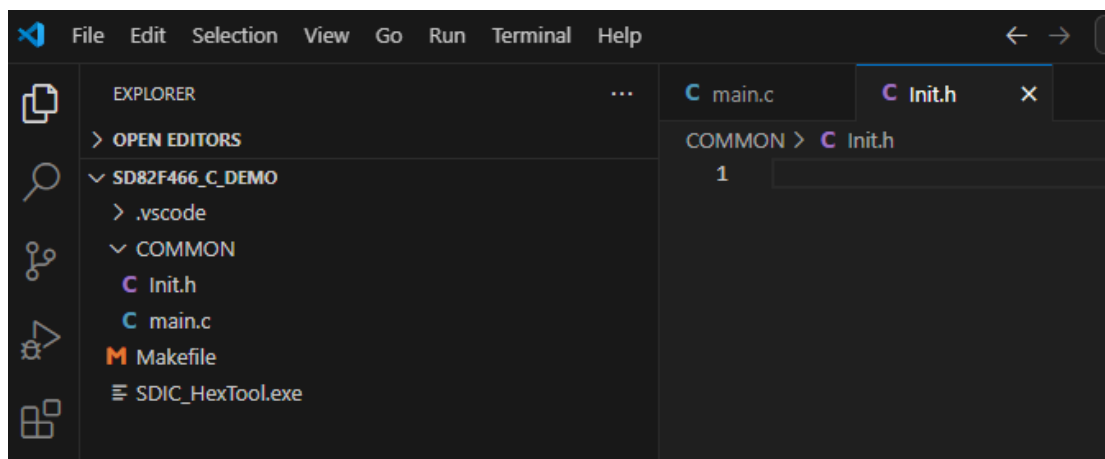
步骤 2: 打开 VSCode 软件,在菜单栏点击 File-Open Folder 找到步骤 1 创建的工程目录文件夹,打开文件夹后,点击选择文件夹,将文件夹加入 VSCode,结果如下图所示。



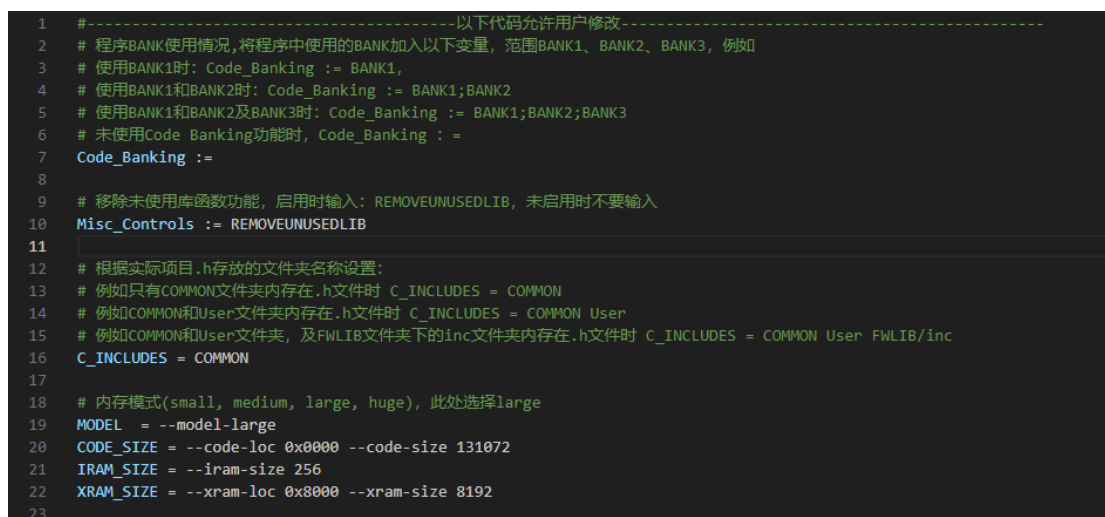
步骤 3: 如上图所示将鼠标移动进工程窗口内时,SD82F466_C_Demo 右边出现的几个图标可用于在鼠标选择的文件夹内创建文件及文件夹。点击  图标创建文件夹,此处输入 COMMON 创建名称为 COMMON 的文件夹,如下图所示。



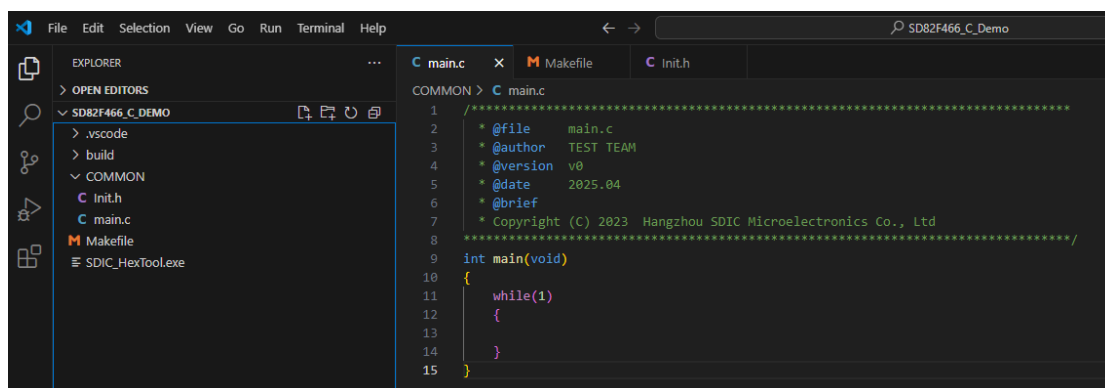
步骤 4: 鼠标点击 **COMMON**, 然后点击  图标创建文件, 此处输入 **main.c** 创建名称为 **main** 类型为 **.c** 的文件, 然后创建 **Init.h** 文件。结果如下图所示。



步骤 5: 点击 **makefile** 文件, 根据工程需求修改如下图所示的参数, 此处目前还未使用 **BANK** 分区功能, 且只有 **COMMON** 存在 **.h** 文件, 芯片是 **SD82F466**(**CODE** 的起始地址是 **0x0000** 大小是 **128kB**, **XDATA** 的起始地址是 **0x8000** 大小是 **8kB**), 所以配置修改如下。



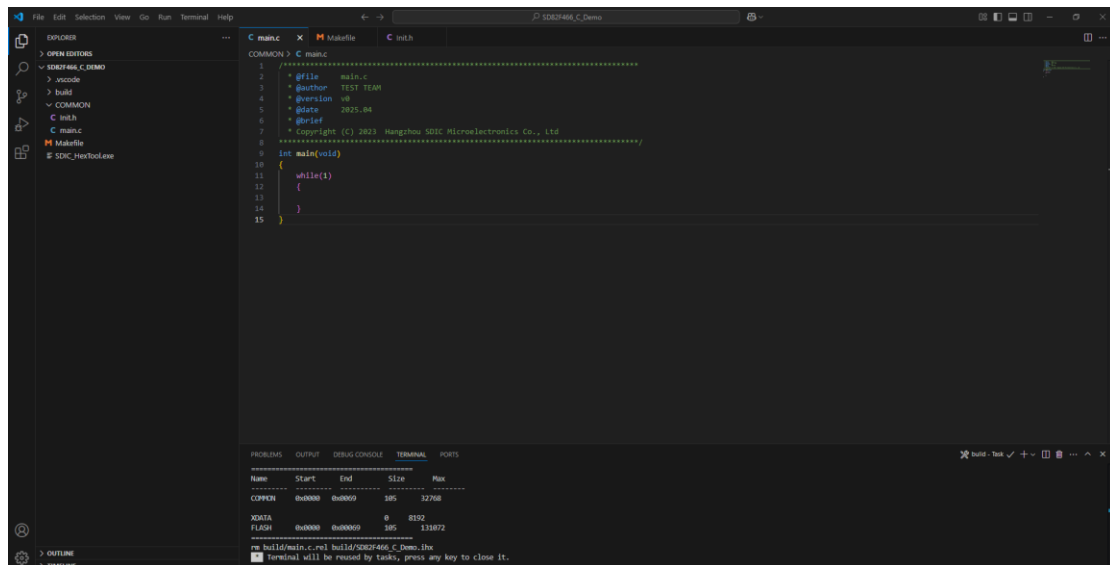
步骤 6: 在 **main.c** 内添加 **main** 主函数代码, 如下图所示。



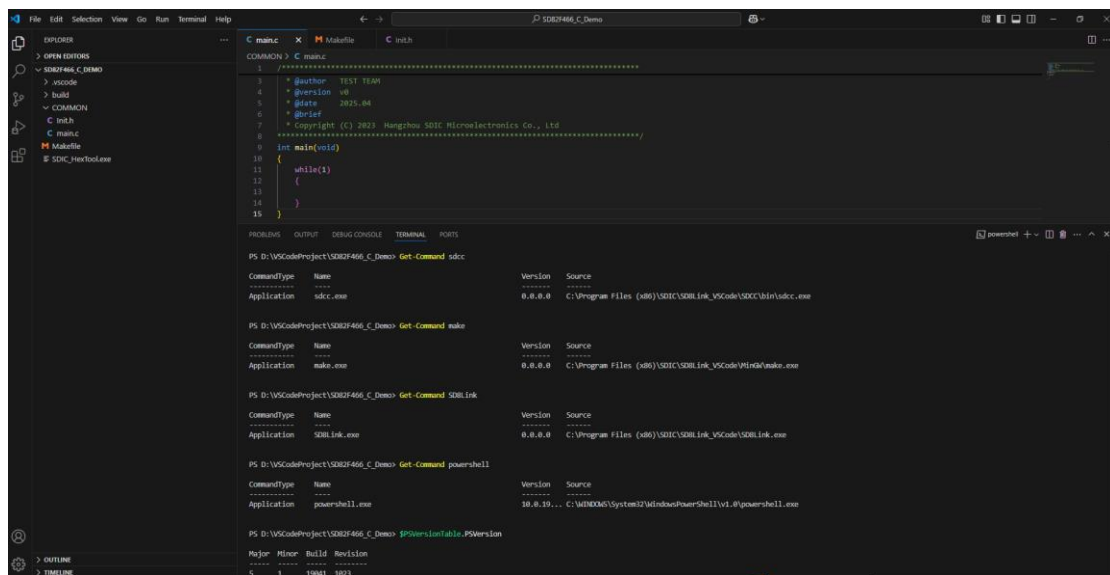
4.编译工程

步骤 1: 鼠标点击界面左边任意位置后, 按下快捷键 “**Ctrl+Shift+B**” 后, 选择 “**bulid**” 鼠标点击或者按下回车开始执行编译, 自动创建 **build** 文件夹, 用于存放编译过程产生的文



件，其中就包括 hex 文件，编译成功时，结果如下图。

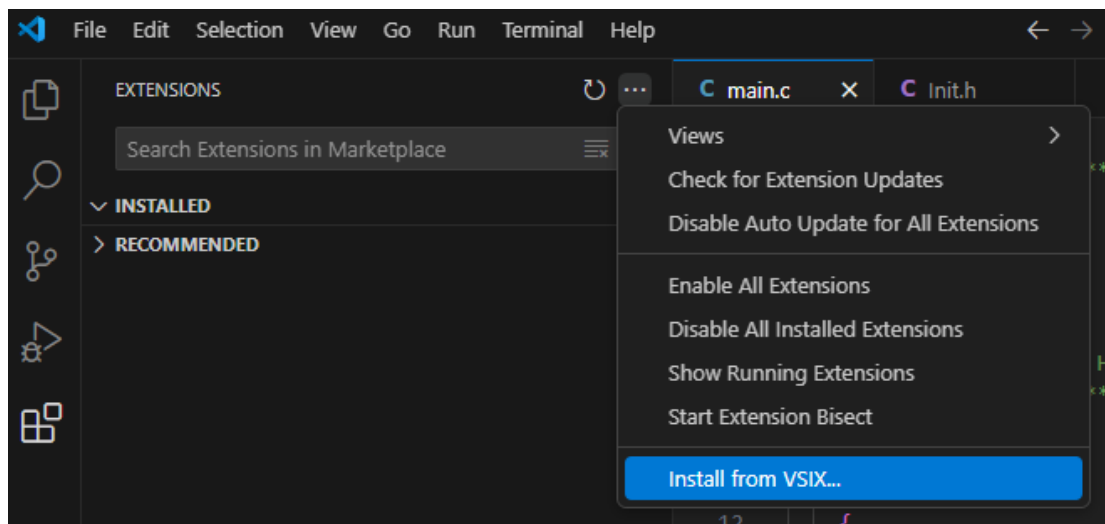


步骤 2: 如果编译失败，按下快捷键“Ctrl+Shift+`”或者菜单栏点击 Terminal-New Terminal，然后分别输入 Get-Command sdcc、Get-Command make、Get-Command SD8Link、Get-Command powershell 查看软件是否支持完整，如下图所示，如果查询结果没有信息或路径异常，需要排除异常再编译。

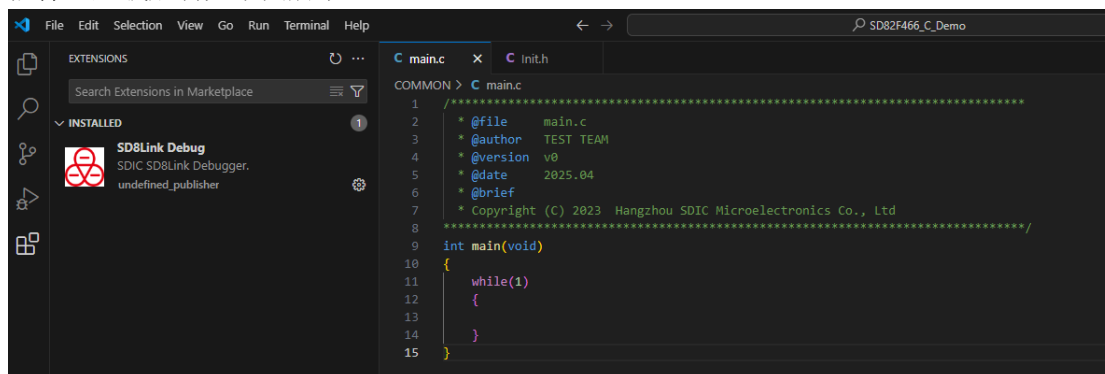



5. 下载工程

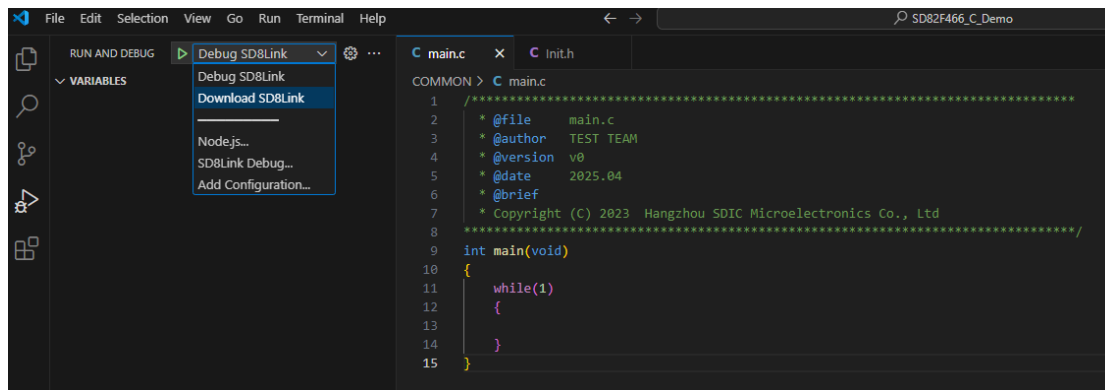
步骤 1: 加载 SD8Link-Debug 插件，此插件主要就是用于下载和调试程序使用，点击左边工具栏图标，然后右上角点击3 个点图标，在点击“Install from VSIX...”，如下图所示。



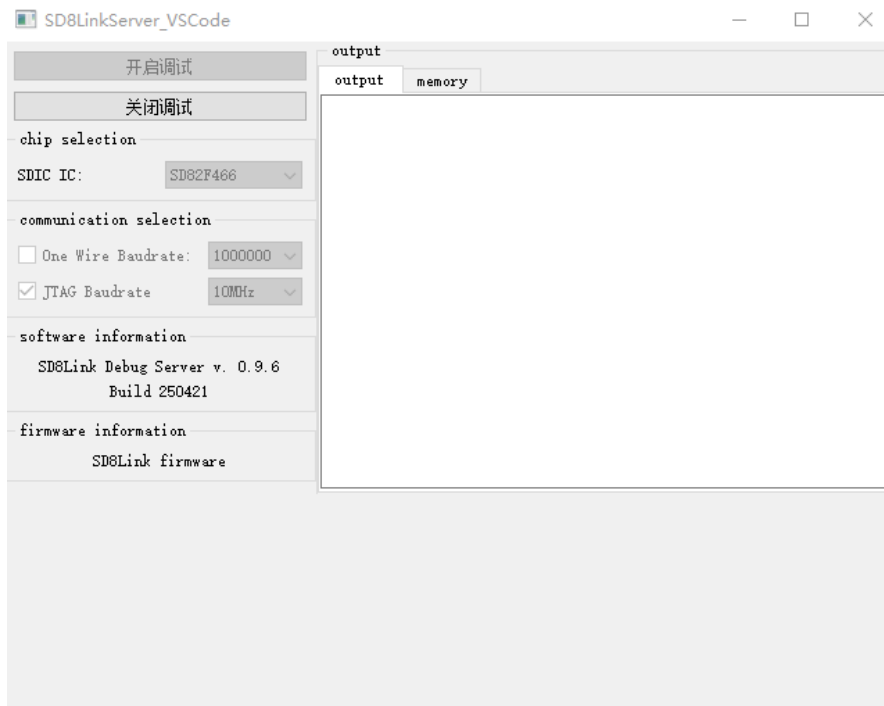
找到“SD8Link_VSCode_Setup_v0.exe”工具安装文件夹下的“sd8link-debug-1.0.0.vsix”插件，加载成功如下图所示。



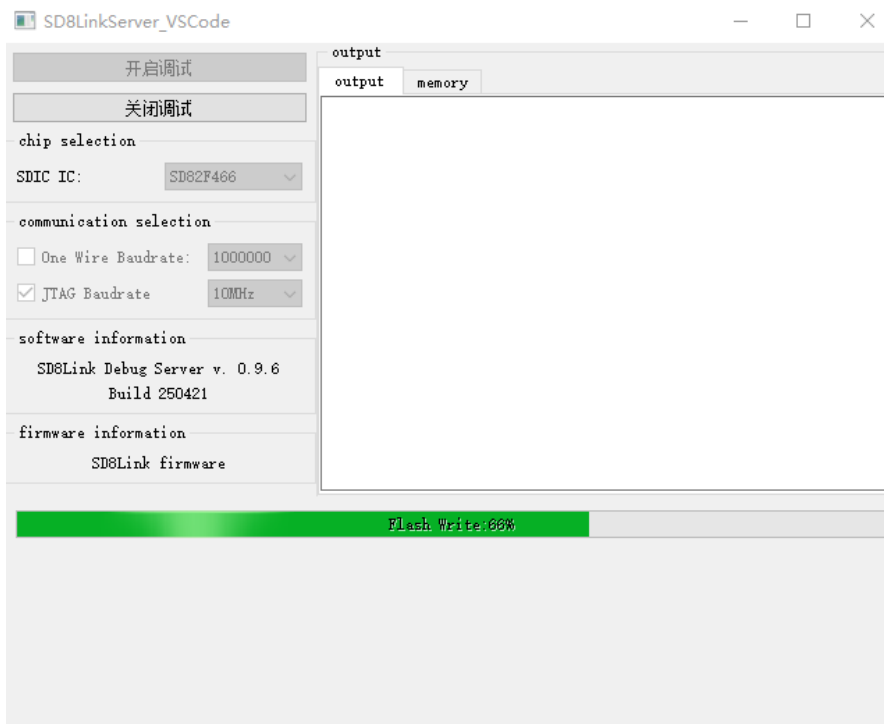
步骤 2: 左边工具栏点击图标，上方绿色三角处选择 Download SD8Link，如下图所示。



步骤 3: 按下快捷键“Ctrl+Shift+B”后，鼠标点击“Debug”自动打开调试所需的调试服务器“SD8LinkServer_VSCode.exe”，路径也是在“SD8Link_VSCode_Setup_v0.exe”工具安装文件夹下，界面如下图所示。

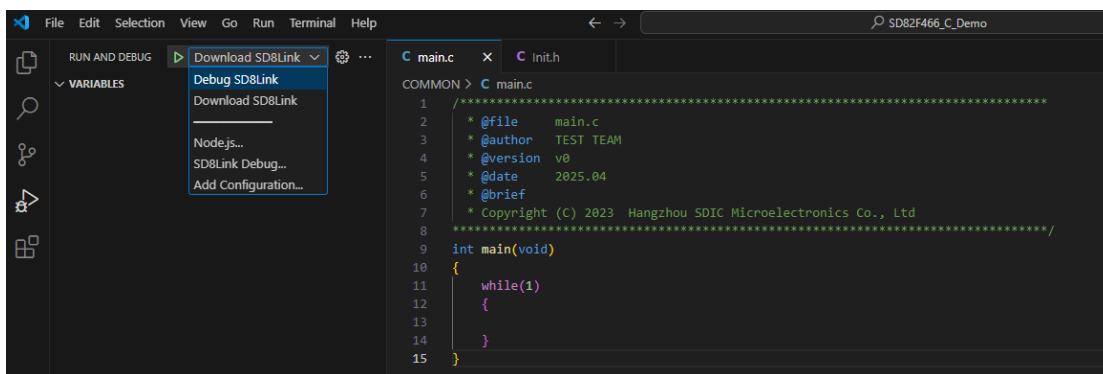


步骤 4: 在步骤 2 的基础上选择 Download SD8Link, 点击绿色三角, 开始下载程序, 下载过程进度条会在步骤 3 打开的界面显示, 如下图所示。

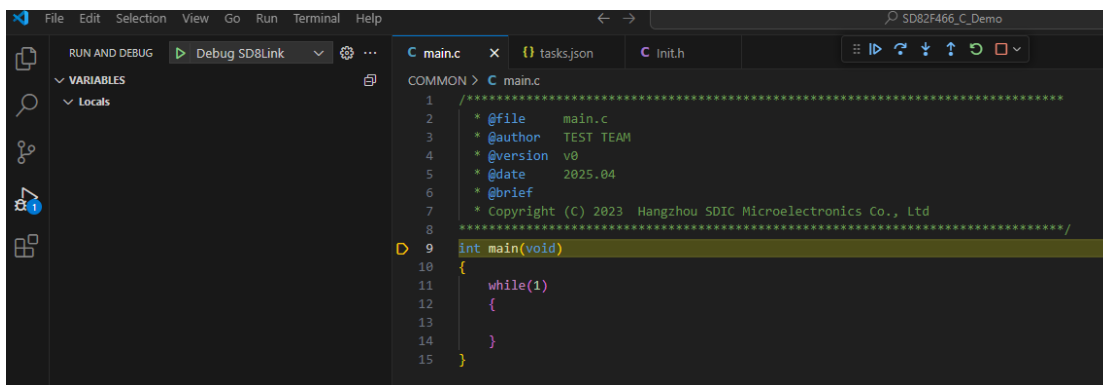


6. 调试工程

步骤 1: 程序下载结束后, 左边工具栏点击  图标, 上方绿色三角处选择 Debug SD8Link, 如下图所示。



步骤 2: 点击绿色三角, 开始进入调试状态, 上方显示调试按钮, 且程序停止在 main 函数的第一条语句, 如下图所示。



7. C 语言应用程序

7.1 .c 文件目录层级

目前允许用户在工程目录下创建最多 20 个自定义的文件夹目录。目前.c 文件目录层级最多允许在工程目录下创建的自定义文件夹目录内, 如果在自定义文件夹目录内继续创建文件夹目录存放.c 文件时, 将导致无法识别.c 文件, 例如: 我们前面在工程目录下创建 COMMON 文件夹目录, 用于存放 main.c 文件, 如果再在 COMMON 内创建子文件夹目录, COMMON 文件夹目录下的子目录内的.c 文件将无法被识别。

7.2 Code Banking 分区

7.2.1 分区说明

未使用 Codebanking 分区功能时, SD82F466 芯片仅支持 64kB 程序空间, 使用 Codebanking 分区功能后, 可扩展到 128kB。目前支持 Bank1、Bank2、Bank3 等三个 Bank 区扩展, 外加一个 Common area 区, 每个区 32kB, 共 4 个区一起构成 128kB 程序空间。

Common area 因为不需要重复加载, 所以效率最高。追求效率的代码都可放在 Common area 中。以下代码部分必须分配在 Common area 内:

- 复位和中断向量;
- 中断函数;
- 定时器;
- 代码常量 (字符串、表);
- Bank 切换程序;
- 库函数。

Bank1 可以放置一上电就需要调用, 调用频繁的代码, 其它调用不频繁的代码可以放在 Bank2 或者 Bank3。

7.2.2 分区应用

此处为了方便区分 Bank 区，所以在工程目录下，再分别新建名称为 BANK1、BANK2、BANK3 的三个文件夹，分别存放 Bank1、Bank2、Bank3 分区的程序。实际应用时文件夹名称可以根据程序需求命名。

例如：在 BANK1 文件夹下，创建硬件初始化“HarWareInit.c”，为了表示接下来的程序代码存放在 Bank1 区，需要在.c 文件开头处添加“#pragma codeseg BANK1”代码进行声明，如下图 7.2.2-1 所示。

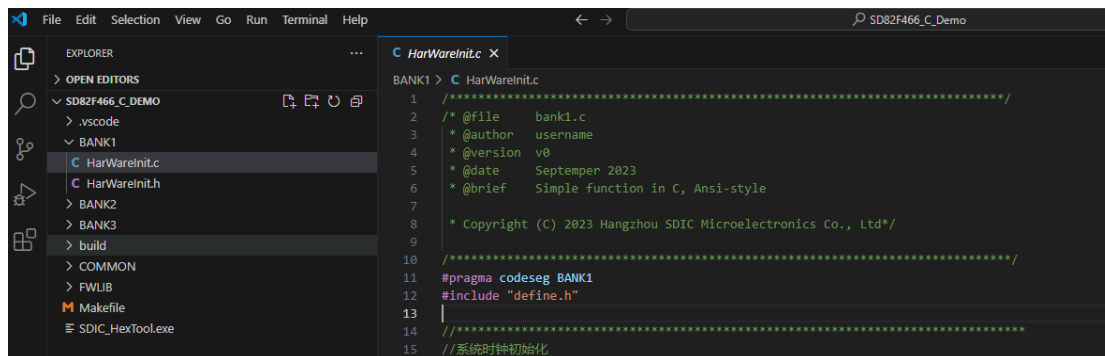


图 7.2.2-1 Bank1 区代码声明

函数定义需要增加“__banked”用于属性声明，编译器通过识别此属性标记函数需支持跨 bank 调用，编译器自动将函数分配到指定 bank，并记录其编号，调用改函数时，编译器自动插入 bank 切换指令(如修改 PSBANK 寄存器值)，确保执行流程正确跳转。通过该关键字实现 bank 切换自动管理，降低开发复杂度，如图 7.2.2-2 所示。

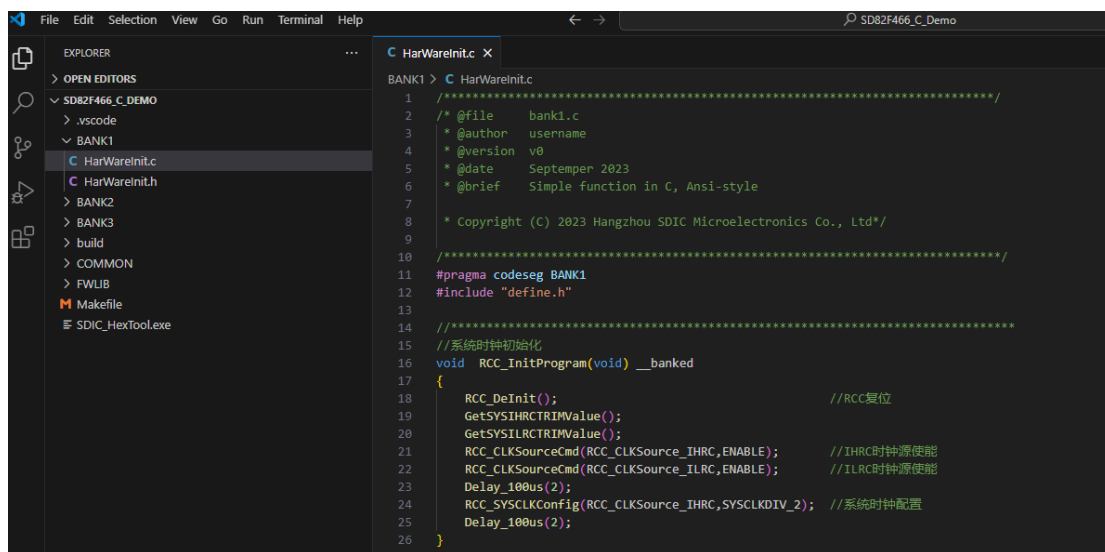


图 7.2.2-2 函数定义添加__banked 属性

在 HarWareInit.h 文件内声明“void RCC_InitProgram(void) __banked”，编译器编译时直接识别函数的“__banked”属性，如图 7.2.2-3 所示。

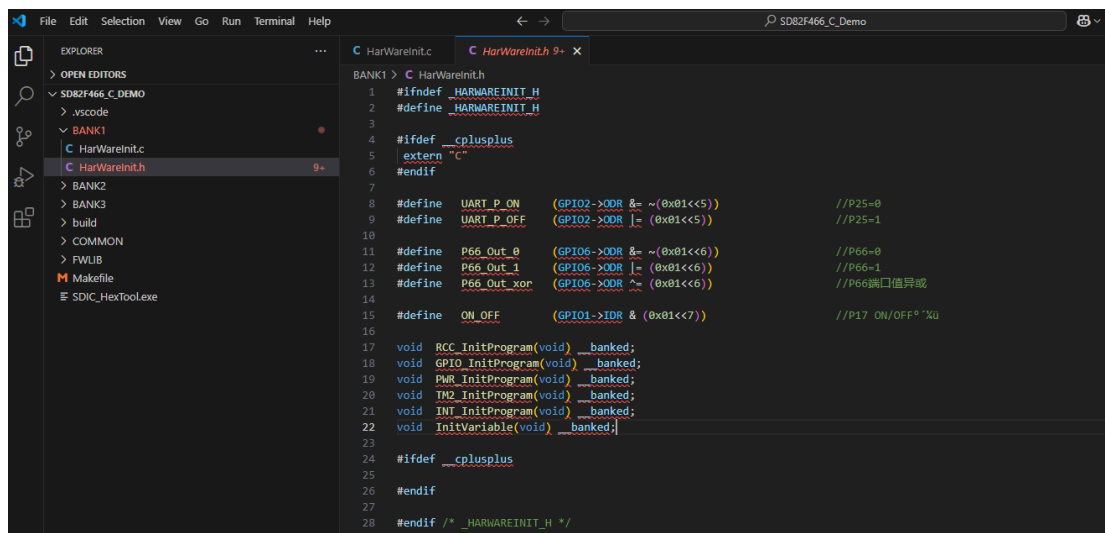


图 7.2.2-3 .h 文件内声明 bank 函数

通过以上步骤，我们就将“void RCC_InitProgram(void)”函数定义结束，接下来需要编译工程。然后通过编译生成的“build”文件夹内的“.map”文件查看函数是否真的在bank1 区地址内，查看结果如图 7.2.2-4 所示。BANK1 内 670 字节与编译结果相同。

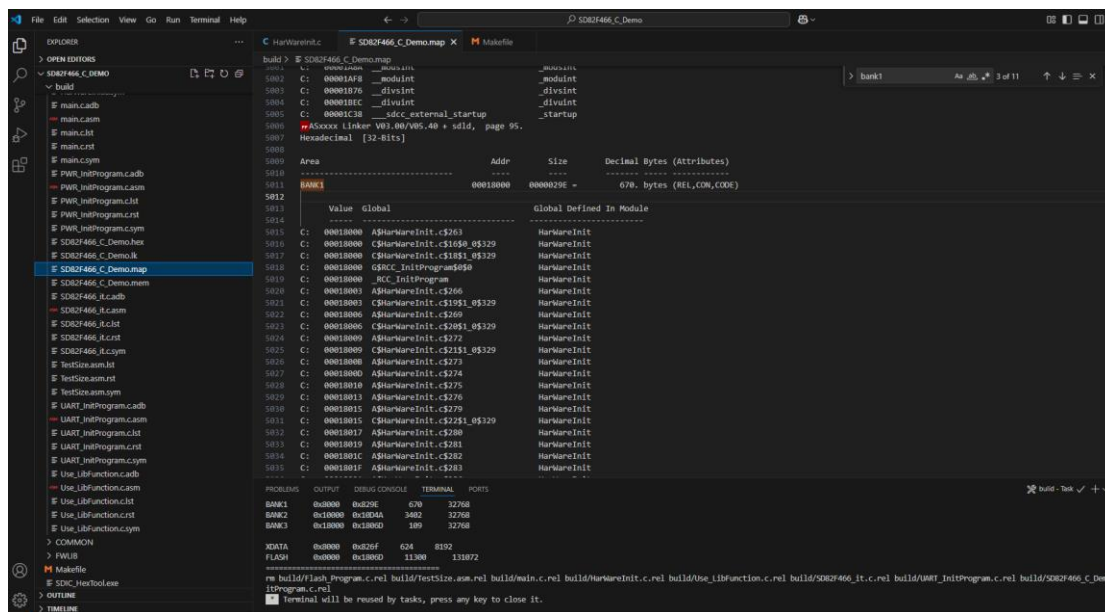


图 7.2.2-4 Bank 分区编译结果

通过以上步骤即可将代码写入指定的 Bank1 区，其它 Bank 区添加代码按照上述 Bank1 区类似，只是需要将.c 文件开头处添加“`#pragma codeseg BANK1`”BANK1 修改为 BANK2 或者 BANK3 即可。

7.2.3 写 FLASH 数据

程序使用分区功能后，因为 Bank 分区后，只有最前面的 0x0000~0x7FFF 范围内的 Common area 区的 32kB 地址保持不变，0x8000~0xFFFF 范围地址属于 Bank1、Bank2、Bank3 共同使用地址，地址内当前存放的是哪个区的代码，取决于目前芯片执行的是哪个 Bank 区的程序。例如向 0xF200 地址内写入的数据，因为这个地址属于 0x8000~0xFFFF 范围地址。因此数据写入哪个 Bank 区主要取决于，执行 Common area 区程序前，最后执行的程序属于哪个编号的 Bank 区，Flash 数据就会写入对应编号的 Bank 区内。

例如：将写数据的程序定义在 Bank2 区，然后在 main 内调用，数据将写入 Bnak2 区

内，如下图 7.2.3-1 所示，在 Bank2 区定义“Write_FlashData”函数。

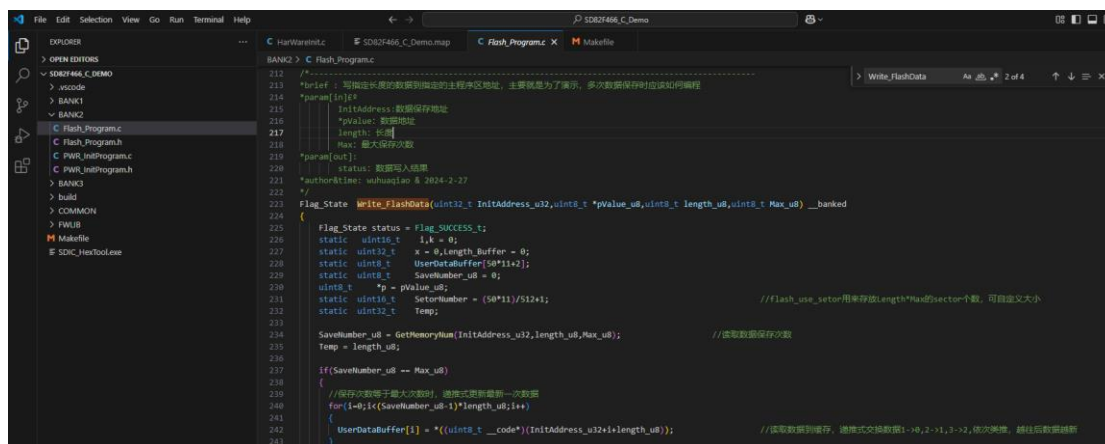


图 7.2.3-1 Bank2 区定义写 Flash 数据函数

然后在主函数内调用 Bank2 区定义的“Write_FlashData”函数，如图 7.2.3-2 所示。

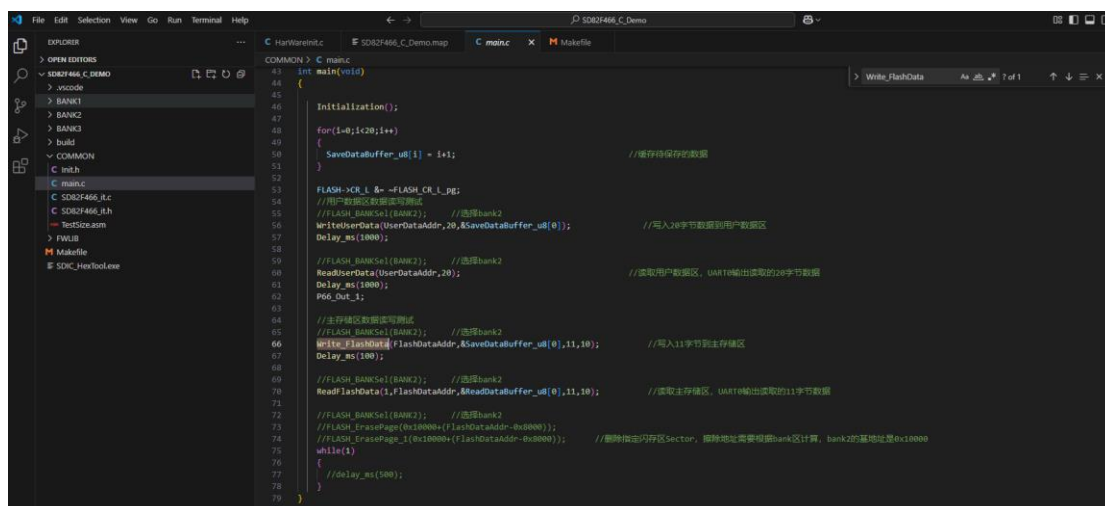


图 7.2.3-2 main 函数内调用 Bank2 区定义的函数

虽然在 Bank2 区定义的“Write_FlashData”函数内，会调用库函数内的函数写 Flash 数据，但是库函数属于 Common area 区程序，并不会改变芯片 Bank 编号设置，因为执行库函数程序前，执行的最后一个函数属于 Bank2 区，所以数据将会写入 Bank2 区。如果在执行库函数程序前，执行的是 Bank1 或者 Bank3 内定义的程序，数据将写入 Bank1 或者 Bank3。

执行程序后，可以通过烧录器查看实际物理地址内的数据，验证 Bank 分区写入数据逻辑，Bank1 物理地址范围 0x8000~0xFFFF，Bank2 物理地址范围 0x10000~0x18000，Bank3 物理地址范围 0x18000~0x1FFFF。或者进入调试后 memory 窗口查看，memory 窗口查看 code 区数据范围是 0x0000~0xFFFF 地址，其中 0x0000~0x7FFF 地址范围内的数据始终保持不变，0x8000~0xFFFF 地址范围内的数据，根据目前执行程序所在 Bank 变化。如果需要查看 Bank2 区内的代码数据，在 Bank2 区程序内加断点，停止在 Bank2 程序内，此时通过 memory 窗口，输入 0xF200 地址，查看的就是 Bank2 区的代码数据，如图 7.2.3-3 所示。

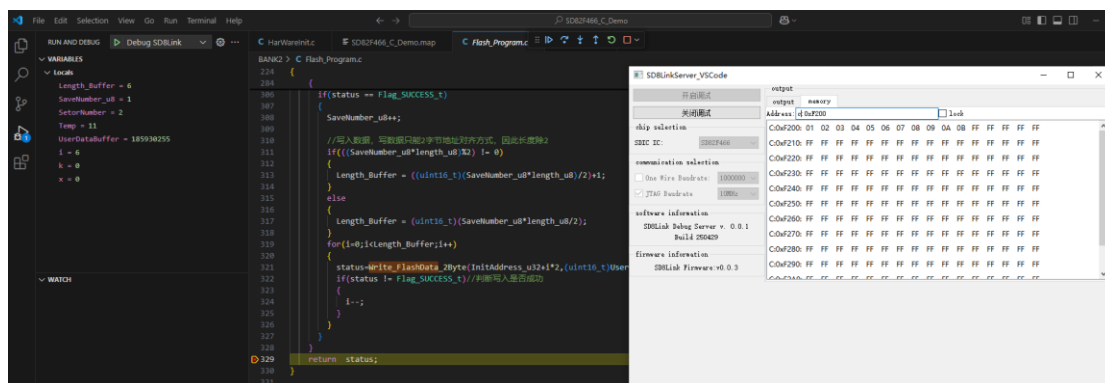


图 7.2.3-3 memory 查看 FLASH 数据

7.3 库函数使用

使用晶华提供的库函数时，为了能够对库函数使用优化选项，存放库函数的文件夹需要严格命名为“FWLIB”，且文件夹下只能创建 inc 和 src 两个文件夹，只能分别用于存放库函数使用的.h 文件和.c 文件。如果在 makefile 内，将“Misc_Controls := ”设置为“REMOVEUNUSEDLIB”时，将启用库函数优化功能，编译前将会优化库函数，将其它非库函数内调用的所有库函数整合，未调用的库函数排除在外，然后在“FWLIB”目录下自动生成“Use_LibFunction.c”文件，此过程会占用一段时间，目的就是优化程序代码大小，如图 7.3-1 所示，**进入调试时将直接使用“Use_LibFunction.c”内的函数。**

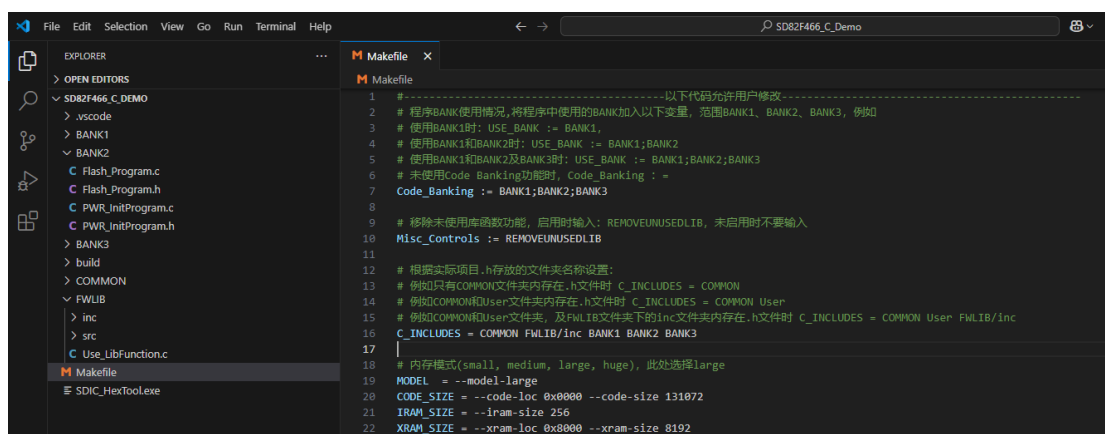


图 7.3-1 makefile 配置图

7.4 makefile 文件配置

makefile 文件需要用户根据项目需求进行简单的配置，主要涉及到 Codebanking 分区、库函数优化、.h 文件目录包含、芯片参数设置等功能需要配置。此处以 SD82F466 芯片为例，CODE 起始地址为 0x0000，大小为 128kB，XDATA 起始地址为 0x8000，大小为 8kB。另外程序中已经使用 BANK1、BANK2、BANK3 等三个分区，且启用库函数优化功能，需要将所有存在.h 文件的目录包含，makefile 配置后如图 7.4-1 所示。

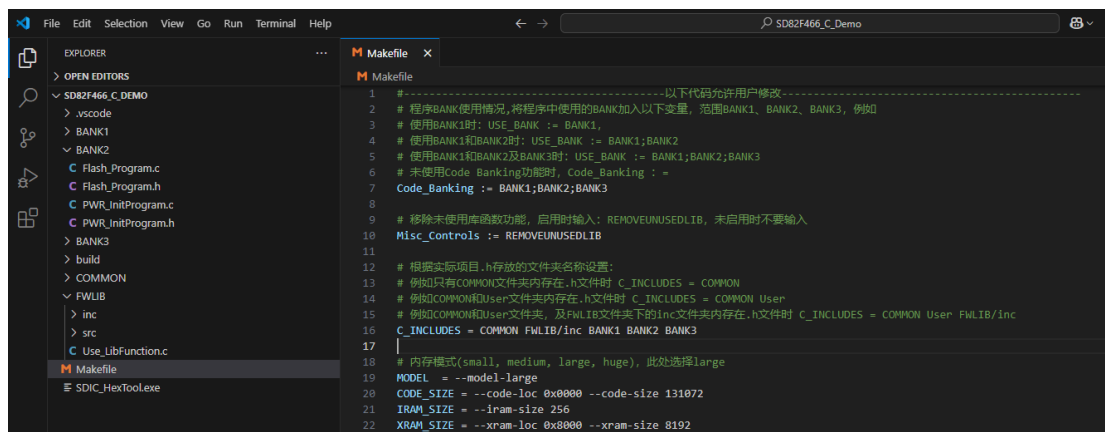


图 7.4-1 makefile 配置 1

如果程序中仅使用 BANK1 和 BANK2 区,且未使用库函数优化功能, makefile 文件配置如图 7.4-2 所示。

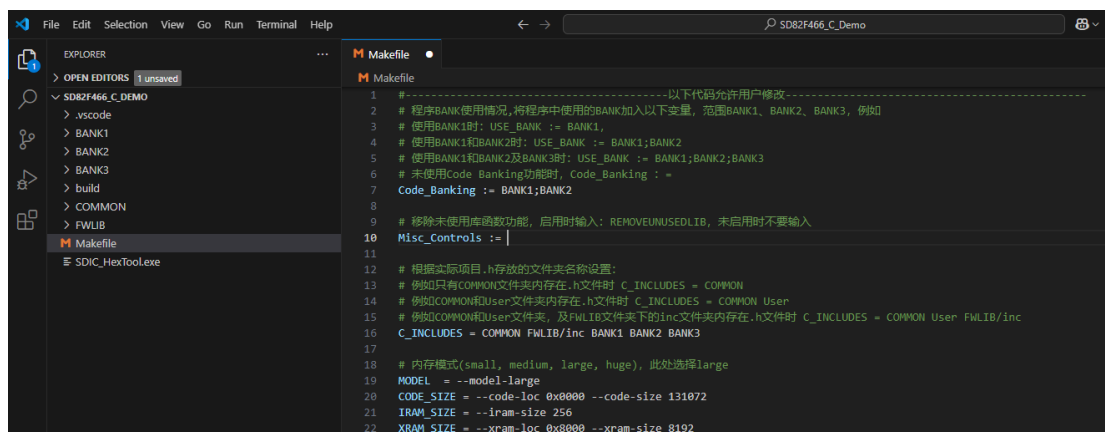


图 7.4-2 makefile 配置 2

以上通过两个举例说明用户如何配置 makefile, 实际项目开发, 请根据实际情况配置 makefile 即可。

7.5 中断函数

中断函数的定义与 Keil 存在差异, 主要就是使用“interrupt”关键字时, 需要修改为“__interrupt”, 且中断通道编号需要使用括号括起来, 如图 7.5-1 所示。

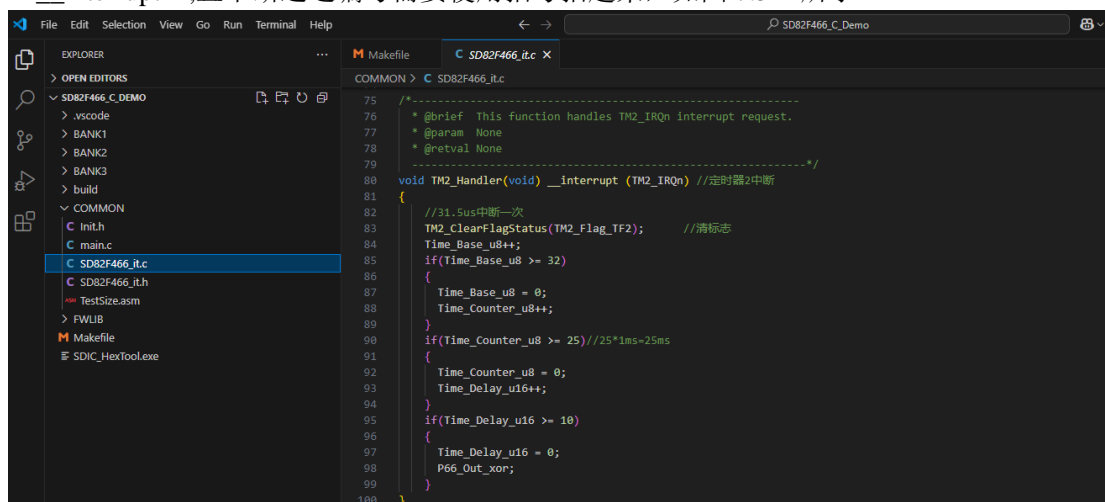


图 7.5-1 中断函数定义

中断函数的声明与 Keil 存在差异，Keil 中不需要在.h 中单独声明中断函数，但是 VS Code 需要在.h 文件内声明中断函数，否则中断函数将无效，不会触发中断，通常中断函数在晶华提供的库文件内已经定义和声明，如果没有，需要用户自己按照要求声明和定义中断函数，确保中断功能正常，如图 7.5-2 所示在.h 文件内声明中断函数。**如果在 makefile 开启了库优化功能，中断函数.c 和.h 文件，请不要放置在库文件夹下，否则会被优化。**因此建议不管是否开启优化库函数功能，中断函数.c 和.h 文件都不要放在库文件夹下。

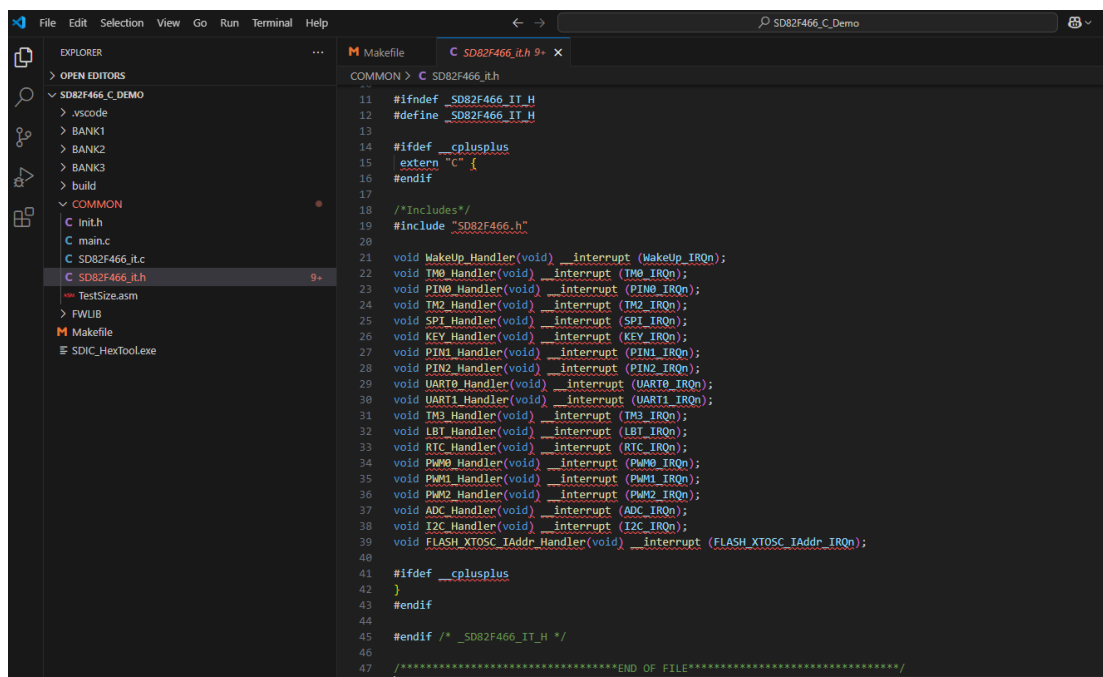


图 7.5-2 中断函数的声明

7.6 关键字与 Keil 编译器的区别

以下表格仅说明使用晶华设计的 VSCode 工具开发程序时，与 Keil 工具中，关键字的区别。

7.6.1 关键字 sfr

sfr 关键字用于定义 8051 系列单片机的特殊功能寄存器，直接映射到硬件的物理地址，从而实现对寄存器的直接操作，在 keil 中用“sfr”表示，SDCC 中用“__sfr”表示，以下是举例说明。

Keil 格式：

sfr 寄存器名 = 寄存器地址，例：sfr P2 = 0x00A0;

SDCC 格式：

__sfr __at 地址 寄存器名，例：__sfr __at 0x00A0 P2;

7.6.2 关键字 sbit 和 bit

Sbit 和 bit 关键字均用于位寻址，在 keil 中用“sbit”表示，SDCC 中用“__sbit”表示，以下是举例说明。另外因为在 Keil 中和 SDCC 中使用格式不同，需要说明下 SDCC 中位寻址要求，芯片内部 IDATA 区的 0x20~0x2F 允许进行为寻址，共有 16 字节地址可位寻址，因此 16*8=128 位，位地址对应 0x00~0x7F，只允许使用 sbit 和 bit 进行为寻址操作，无法同 Keil 定义寄存器位操作，如果需要对寄存器位操作，需要使用宏定义+掩码方式。

Keil 格式：sbit P2_1 = P2^1;

SDCC 宏定义+掩码方操作：

unsigned char flags;//变量声明

```
#define Flag0 (flags & 0x01)//变量第 0 位
```

SDCC 格式: `__sbit __at 0x07 Flag1` 或者 `__bit __at 0x07 Flag1`, 位地址 0x07, 表示访问的是 0x20 地址内的位 7。 `__sbit __at 0x0F Flag1` 或者 `__bit __at 0x0F Flag1`, 表示访问的是 0x21 地址内的位 7, 可通过调试窗口 memory I:0x20 或者 0x21 查看操作结果。

7.6.3 关键字 xdata

xdata 关键字用于访问外部扩展 RAM, 在 keil 中用 “xdata” 表示, SDCC 中用 “__xdata” 表示, 以下举例说明。

Keil 格式:

外部 RAM 变量: `unsigned char xdata buffer[10];`

指针定义: `xdata unsigned char *ptr;`

寄存器基地址定义: `#define FLASH ((FLASH_TypeDef xdata) FLASH_BASE)`

读 xdata 数据: `Data_u8 = *((uint8_t xdata*)Address_u16)`

SDCC 格式:

外部 RAM 变量: `__xdata unsigned char buffer[10];` //也可以不显式写 __xdata 或 __data 关键, 编译器自动分配。

指针定义: `__xdata unsigned char __data *ptr;` //也可以不显式写 __xdata 或 __data 关键, 编译器自动分配。

寄存器基地址定义: `#define FLASH ((FLASH_TypeDef __xdata) FLASH_BASE)`

读 xdata 数据: `Data_u8 = *((uint8_t __xdata*)Address_u16);`

7.6.4 关键字 code

code 关键字用于将数据存储在 ROM(程序存储区), 或者读取 ROM 数据, 在 keil 中用 “code” 表示, SDCC 中用 “__code” 表示, 以下举例说明。

Keil 格式:

存储数据: `unsigned char code table[] = {0x01, 0x02};`

指定地址存储数据 `uint8_t code at(0x600) Data[10]={0x00};`

读 code 数据: `Data_u8 = *((uint8_t code*)Address_u16);`

SDCC 格式:

存储数据: `__code unsigned char table[] = {0x01, 0x02};`

读 code 数据: `Data_u8 = *((uint8_t __code*)Address_u16);`

指定地址存储数据: `__code __at(0x600) uint8_t Data[10]={0x00};`

7.6.5 关键字 data

data 关键字在 8051 单片机中, 用于指定变量存储在内部 RAM 区, 可直接寻址的低 128 字节区域(地址范围 0x00~0x7F), 在 keil 中用 “data” 表示, SDCC 中用 “__data” 表示, 以下举例说明。

Keil 格式:

变量定义: `unsigned char data var;`

指针定义: `unsigned char data *ptr;`

SDCC 格式:

变量定义: `__data unsigned char var;`

指针定义: `__data unsigned char *__data ptr;`

7.6.6 关键字 idata

idata 关键字在 8051 单片机中, 用于指定变量存储在整个内部 RAM 的 256 字节区域(地址范围 0x00~0xFF), 包含与 data 区重叠的低 128 字节(0x00~0x7F), 在 keil 中用 “idata” 表示, SDCC 中用 “__idata” 表示, 以下举例说明。

Keil 格式:

变量定义: unsigned char idata var;

指针定义: unsigned char idata *ptr;

SDCC 格式:

变量定义: __idata unsigned char var;

指针定义: __idata unsigned char *__data ptr;

7.6.7 关键字 pdata

pdata 关键字在 8051 单片机中, 用于指定变量存储在外部扩展 RAM 低 256 字节区域 (0x00~0xFF), 在 keil 中用 “pdata” 表示, SDCC 中用 “__pdata” 表示, 以下举例说明。

Keil 格式:

变量定义: unsigned char pdata var;

指针定义: unsigned char pdata *ptr;

SDCC 格式:

变量定义: __pdata unsigned char var;

指针定义: __pdata unsigned char *__data ptr;

7.6.8 关键字 interrupt

interrupt 关键字, 用于定义中断服务函数(ISR), 在 keil 中用 “interrupt” 表示, SDCC 中用 “__interrupt” 表示, 以下举例说明。

Keil 格式:

中断服务函数:

```
void TM2_Handler(void) interrupt TM2_IRQn
{
}
```

SDCC 格式:

中断服务函数:

```
void TM2_Handler(void) __interrupt (TM2_IRQn)
{
}
```

7.6.9 关键字 reentrant

reentrant 关键字, 用于声明可重入函数, 在 keil 中用 “reentrant” 表示, SDCC 中用 “__reentrant” 表示, 以下举例说明。

Keil 格式:

可重入函数定义:

```
void Func(int a) reentrant
{
}
```

SDCC 格式:

```
void Func(int a) __reentrant
{
}
```

注意: 默认将局部变量和参数分配至固定地址, 需显示声明 __reentrant 或在编译时 --stack-auto 选项启用堆栈模拟。因此可重入函数的局部变量不能指定绝对地址或存储类型(如 __data), 避免与堆栈冲突。

7.6.10 关键字 at

at 关键字，用于绝对地址分配，在 keil 中用 “_at_” 表示，SDCC 中用 “__at” 表示，以下举例说明。

Keil 格式：需结合 data\xdata\code 等存储类型声明，且使用_at_定义的变量不能直接初始化，需要在代码中手动赋值。

变量定义：uint8_t xdata text _at_ 0x8100;

SDCC 格式：需结合 __data__xdata__code 等存储类型声明，SDCC 不跟踪__at 定义的变量地址，使用时需要注意，避免和其它变量冲突。

变量定义：__xdata __at(0x8100) uint8_t text;

7.7 C 语言程序如何嵌入汇编代码

在编写 C 语言时，为了控制代码大小，通常我们可以嵌入汇编代码，C 语言程序中嵌入汇编代码的格式如下所示，使用 “__asm 和 __endasm;” 嵌入汇编代码。

```
int main(void)
{
    Initialization();

    //插入汇编程序
    __asm
    NOP;空指令
    NOP;空指令
    __endasm;

    flag1 = 0x01;

    for(i=0;i<20;i++)
    {
        SaveDataBuffer_u8[i] = i+1;           //缓存待保存的数据
    }
}
```

如果我们想在 C 语言程序中，创建一个单独的汇编源文件，即.asm 文件，目前不建议使用这个方法，因为跨文件汇编源文件，在编译时自动分配的地址会与其它代码段地址重合，因此建议使用嵌入汇编代码的方式。

8. 汇编语言应用程序

8.1 创建汇编工程

本小节主要作用是创建一个简单的可进行编译的汇编工程，在此基础上继续后续内容的说明，会更容易理解，具体汇编语法在后续章节进行详细说明。

步骤 1：按照“3.创建工程”小节的说明，在工程目录下创建一个自定义的子目录文件夹，用于存放汇编源文件(.asm 后缀)，此处创建一个“User”名称的子目录文件夹，结果如图 8.1-1 所示。

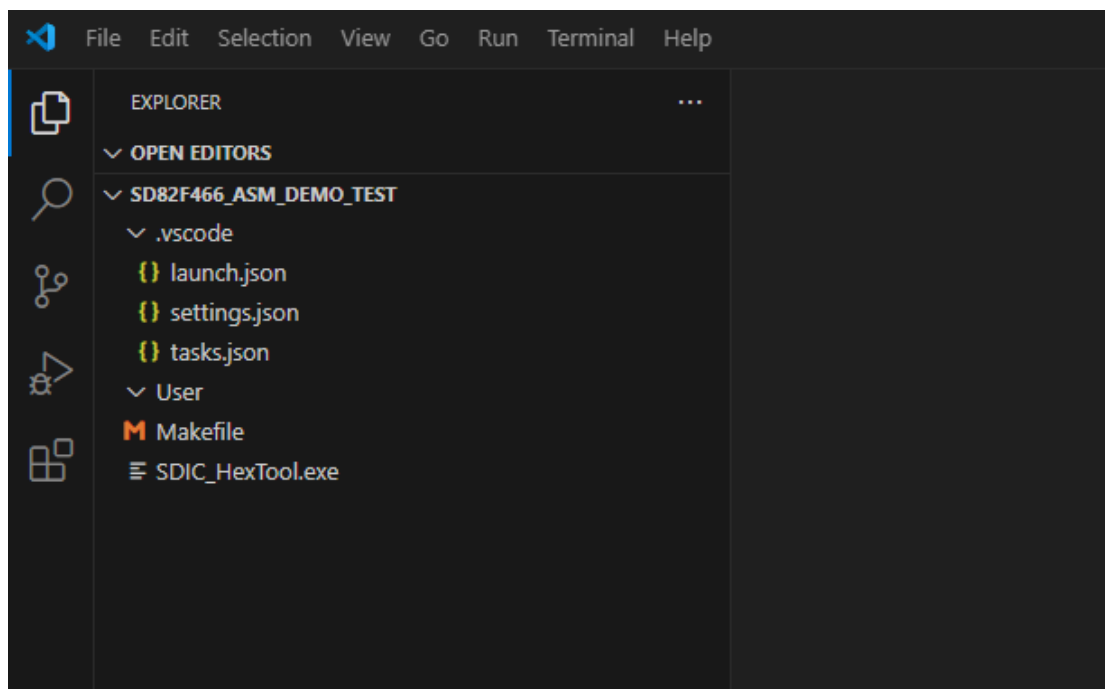


图 8.1-1 创建 User 子目录文件夹

步骤 2: 在“User”子目录文件夹下, 创建自定义名称的源文件, 此处举例创建“main.asm”源文件, 结果如图 8.1-2 所示。

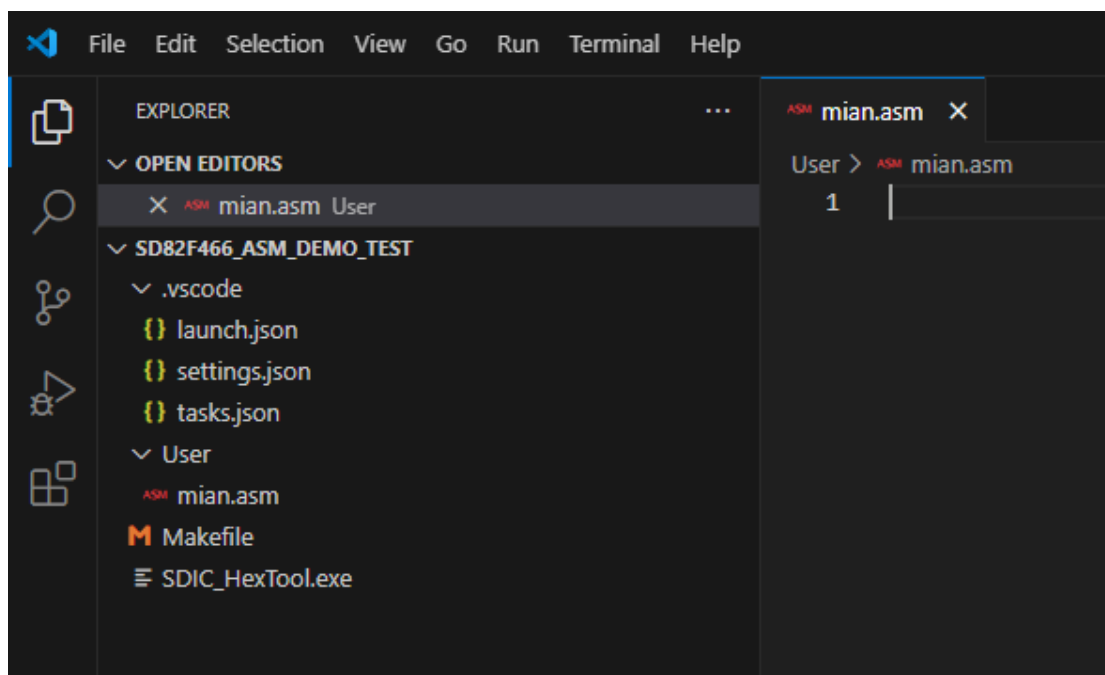


图 8.1-2 创建汇编源文件

步骤 3: 在汇编原文件内添加汇编代码, 为了后续更好的使用变量和寄存器, 再添加两个头文件, 分别是芯片寄存器头文件“SD82F466.INC”及变量或常量定义头文件“RAM.INC”, 头文件内容根据实际使用情况定义, 结果如图 8.1-3 所示。

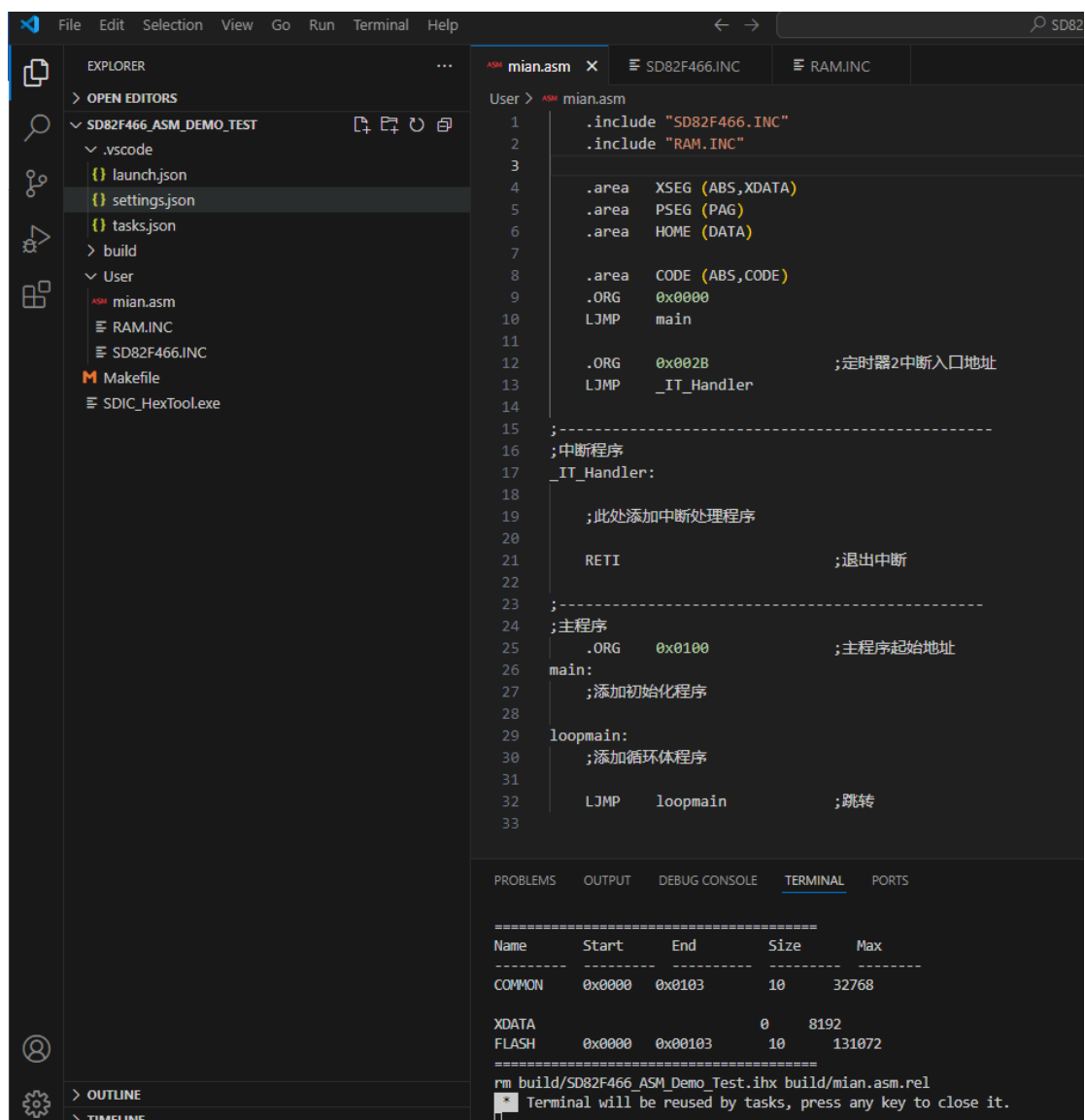


图 8.1-3 源文件添加汇编程序

8.2 配置汇编程序调试

创建工程、编译、下载等功能与上述 C 工程相同，汇编调试功能需要进行简单的配置，具体说明如下：

步骤 1：打开工程目录文件夹“.vscode”下的“launch.json”文件，增加"debugType":"asm"字符串，否则无法进入调试，C 工程时去掉或者改为"debugType":"c"都行，如图 8.2-1 所示。

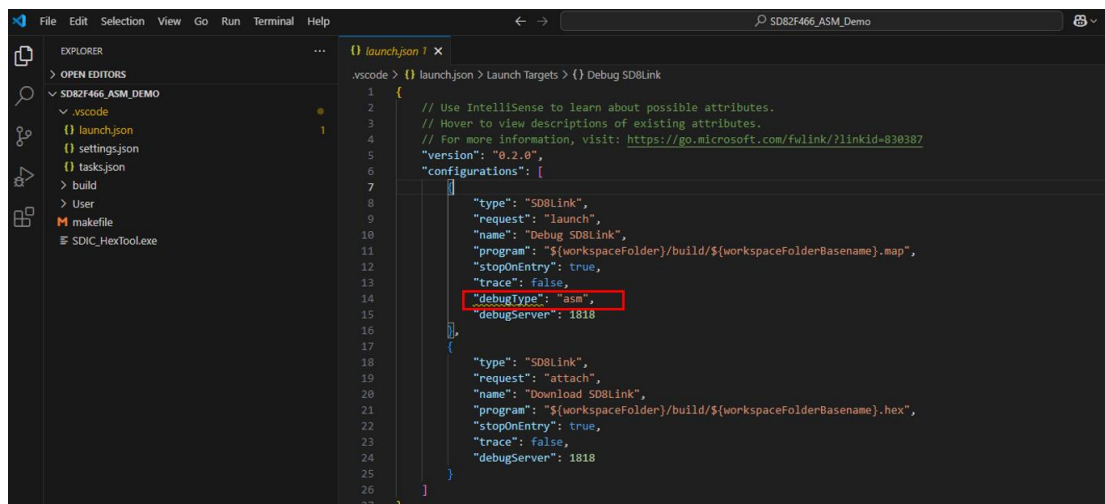



图 8.2-1 增加"debugType":"asm"字符串

步骤 2: 点击左边工具栏左下角  图标, 然后点击“Settings”, 在弹出的界面 User-Features-Debug 勾选 Allow setting breakpoints in any file, 不勾选调试汇编程序时, 无法设置断点, 如图 8.2-2 所示。

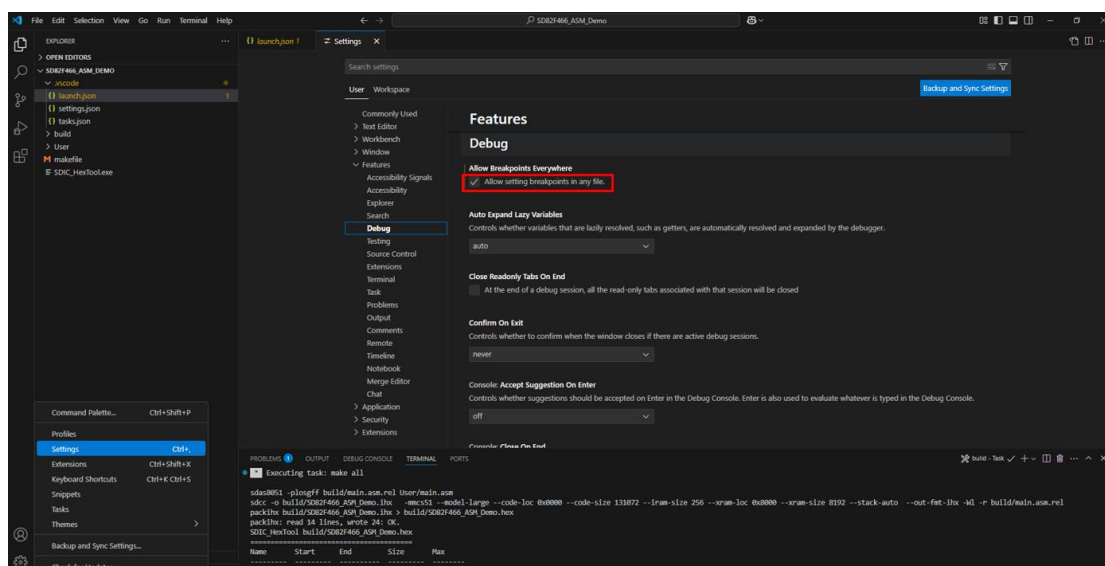


图 8.2-2 勾选 Allow setting breakpoints in any file

8.3 汇编语法说明

8.3.1 文件结构

一个汇编源文件可以包含很多行, 每行可以包含一个标号(必须从第一列开始)或者一条汇编操作(从第一列之后开始)。标号和汇编操作可以单独成行, 也可以存在于同一行中。汇编文件中的**注释用分号“;”表示**, 该行中分号之后的内容汇编器不作处理。

标号只能包含 ‘a’ ~ ‘z’、‘A’ ~ ‘Z’ 的字母及 ‘0’ ~ ‘9’ 的数字和下划线 ‘_’。一个标号不能以数字开头作为第一个字符, 标号后面需要跟冒号 ‘:’, 否则会报错。

可在标号 ‘:’ 后面直接写汇编操作代码。一条汇编操作语句一般包含一个指令助记符和 n 个操作数, 多个操作数之间用 ‘,’ 进行分隔。

例如: 以下是合法的汇编代码

```
loopmain_end: MOV A,#0x01 ;汇编操作码与标号loopmain_end同一行时书写格式
MOV A,#0x55
LJMP loopmain
```

图8.3.1-1 合法的汇编代码

汇编源文件格式只能是“源文件名.asm”，否则编译器无法识别源文件。另外源文件不能直接放在工程目录下面，需要放在工程目录下创建的子目录内，否则编译器无法识别，另外也不能放在子目录下的子目录内。例如下图，将源文件放在自定义的 User 文件夹下，文件夹名称可自定义。

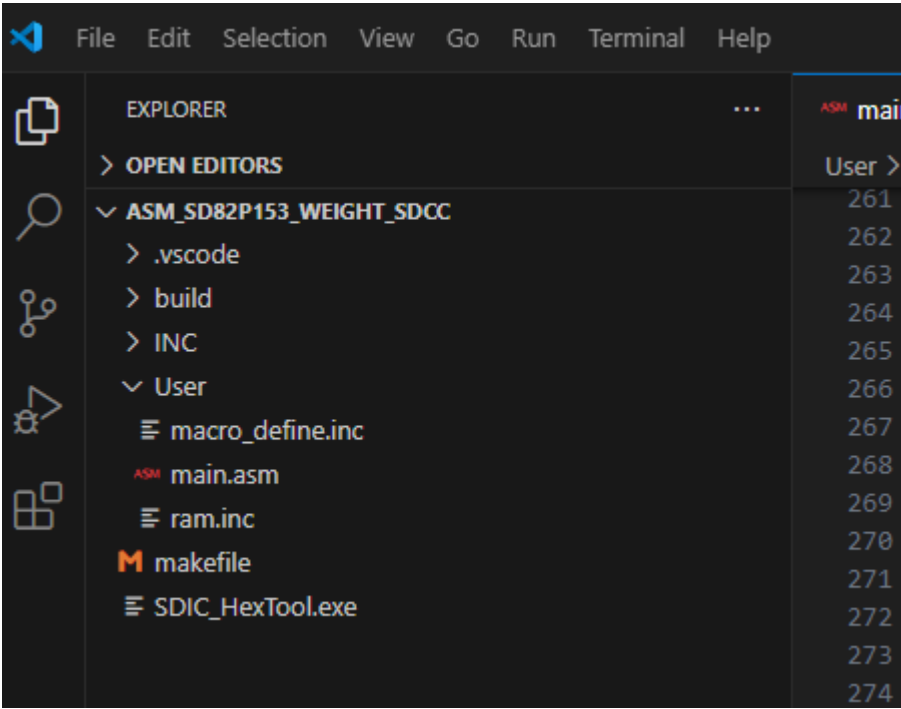


图8.3.1-2 源文件格式及存放目录

VSCODE 中对自定义的变量名或常量名，有大小写要求，程序中使用自定义的**变量名或常量名**时，需要注意**保持大小写一致**。

8.3.2 表达式

VSCODE 的汇编器可以进行表达式计算，下表列出了 VSCODE 的汇编器允许使用的运算操作符，按照优先级顺序从低到高排列，相同优先级的操作符，其计算顺序为从左到右。

操作符	说明	优先级
=	赋值运算	0
&	位与	1
	位或	
^	位异或	
==	等于	2
!=	不等于	
<<	左移	3
>>	右移	
+	加法	4
-	减法	
*	乘法	5

/	除法	
%	取模	
-	取负数	6
~	位非	
>	取高字节	
<	取低字节	

8.3.3 数字

VSCODE 的汇编器支持通过多种格式来指定一个数字，其语法为：通过一个字符前缀来指定数字基数，如下表所示。

基数	格式
二进制	0B01010101
八进制	0o77 或者 0q77(对应 10 进制 63)
十进制	85(无前缀)
十六进制	0x55(不可使用 55H 表示)
ASCII	'A'(单引号表示，A=0x41(A 的 ASCII 码))

8.3.4 预处理器

预处理器用于预处理命令，一般来说，预处理命令在整个汇编过程中是最先被处理的，例如：.include 和.equ 命令，汇编器分析到.include 指令时，会自动把文件的内容插入到.include 所在行的后面。对于宏定义命令 X.equ Y，汇编器在处理的时候，对后续所有出现的 X 全部使用 Y 进行替换。

8.3.5 处理器头文件

对于处理器，一般需要一个标准的头文件定义，此文件中定义了处理器特定的一些数据，例如寄存器名称、变量名称、RAM 或 ROM 空间等。这样一个标准的头文件定义可以到处重用，用户不必进行新定义，主需要使用.include 命令包含该头文件即可，例如在 SD82F466.INC 内定义寄存器与地址的映射关系，在 RAM.INC 文件内定义程序使用到的变量与地址的映射关系。

```
-----  
;   
GPIO1_DIR      .EQU      0x3300      ;PT1  端口方向设置寄存器  
GPIO1_IDR      .EQU      0x3301      ;PT1  输入数据寄存器  
GPIO1_ODR      .EQU      0x3302      ;PT1  输出数据寄存器  
GPIO1_PUR      .EQU      0x3303      ;PT1  上拉控制寄存器  
GPIO1_DSR      .EQU      0x3304      ;PT1  驱动能力控制寄存器  
  
GPIO2_DIR      .EQU      0x3308      ;PT2  端口方向设置寄存器  
GPIO2_IDR      .EQU      0x3309      ;PT2  输入数据寄存器  
GPIO2_ODR      .EQU      0x330A      ;PT2  输出数据寄存器  
GPIO2_PUR      .EQU      0x330B      ;PT2  上拉控制寄存器  
GPIO2_DSR      .EQU      0x330C      ;PT2  驱动能力控制寄存器  
  
GPIO3_DIR      .EQU      0x3310      ;PT3  端口方向设置寄存器  
GPIO3_IDR      .EQU      0x3311      ;PT3  输入数据寄存器  
GPIO3_ODR      .EQU      0x3312      ;PT3  输出数据寄存器  
GPIO3_PUR      .EQU      0x3313      ;PT3  上拉控制寄存器  
GPIO3_DSR      .EQU      0x3314      ;PT3  驱动能力控制寄存器  
  
GPIO4_DIR      .EQU      0x3318      ;PT4  端口方向设置寄存器  
GPIO4_IDR      .EQU      0x3319      ;PT4  输入数据寄存器  
GPIO4_ODR      .EQU      0x331A      ;PT4  输出数据寄存器  
GPIO4_PUR      .EQU      0x331B      ;PT4  上拉控制寄存器  
GPIO4_DSR      .EQU      0x331C      ;PT4  驱动能力控制寄存器  
  
GPIO5_DIR      .EQU      0x3320      ;PT5  端口方向设置寄存器  
GPIO5_IDR      .EQU      0x3321      ;PT5  输入数据寄存器  
GPIO5_ODR      .EQU      0x3322      ;PT5  输出数据寄存器  
GPIO5_PUR      .EQU      0x3323      ;PT5  上拉控制寄存器  
GPIO5_DSR      .EQU      0x3324      ;PT5  驱动能力控制寄存器  
  
GPIO6_DIR      .EQU      0x3328      ;PT6  端口方向设置寄存器  
GPIO6_IDR      .EQU      0x3329      ;PT6  输入数据寄存器  
GPIO6_ODR      .EQU      0x332A      ;PT6  输出数据寄存器  
GPIO6_PUR      .EQU      0x332B      ;PT6  上拉控制寄存器  
GPIO6_DSR      .EQU      0x332C      ;PT6  驱动能力控制寄存器  
  
GPIO7_DIR      .EQU      0x3330      ;PT7  端口方向设置寄存器  
GPIO7_IDR      .EQU      0x3331      ;PT7  输入数据寄存器  
GPIO7_ODR      .EQU      0x3332      ;PT7  输出数据寄存器  
GPIO7_PUR      .EQU      0x3333      ;PT7  上拉控制寄存器
```

图 8.3.5-1 SD82F466.INC 文件定义寄存器


```
;=====
;RAM定义
;=====
AX .EQU 0x8000
BX .EQU 0x8003
CX .EQU 0x8006
DX .EQU 0x8009

AH .EQU AX+1

BL .EQU BX
BH .EQU BX+1

CL .EQU CX
CH .EQU CX+1

DL .EQU DX
DH .EQU DX+1

AHH .EQU AX+2
BHH .EQU BX+2
CHH .EQU CX+2
DHH .EQU DX+2

ADC_Filter_U1 .EQU 0x800C
R_ADCOut .EQU 0x800F
```

图8.3.5-2 RAM.INC 文件内定义变量

```
User > ASM main.asm

1 | .include "SD82F466.INC"
2 | .include "RAM.INC"
```

图8.3.5-3 asm 汇编源文件内包含 INC 头文件

8.4 汇编指令/伪指令说明

8.4.1 代码生成

在绝对汇编模式下，使用.ORG 伪指令设置汇编代码的起始 PC 位置，如果一个汇编程序没有使用.ORG 伪指令指定起始地址，汇编器默认从 0x0000 地址开始汇编，在可重定位模式下，使用.CODE 伪指令来指定汇编代码的位置，如下图所示。

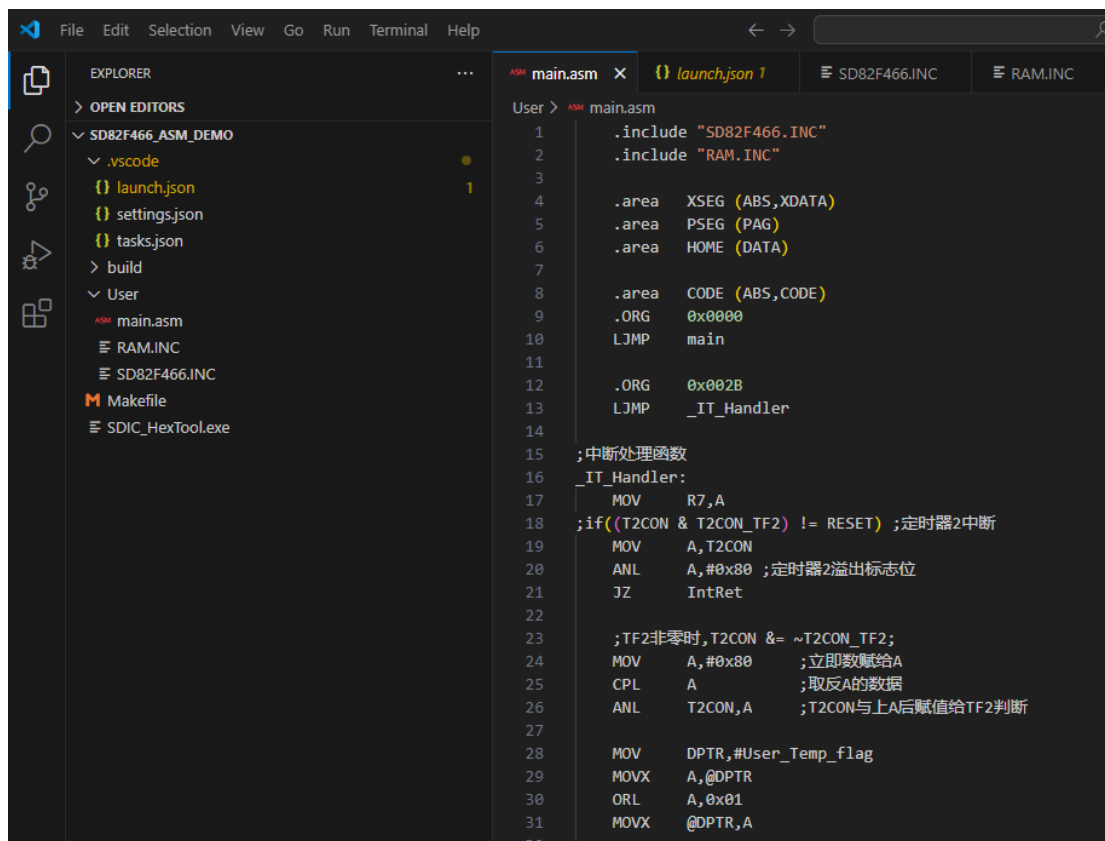


图 8.4.1-1 程序起始地址

8.4.2 条件汇编

VSCODE 的汇编器可以使用 .IF、.IFDEF、.IFNDEF、.ELSE、.ENDIF、.DEFINE 伪指令来进行条件汇编，只有当条件成立时，该部分代码才会被汇编器处理，条件汇编伪指令本身并不会产生实际的代码。

例如需要使用 “.IFDEF、.ENDIF” 进行条件汇编选择，并且使用 “.IF、ELSE、ENDIF” 进行多层代码条件判断处理，代码如下图所示，此代码为了更直观的说明条件汇编，不同条件下编译指令字节数不同，直接看指令编译字节数就能判断程序条件选择结果，另外也可以进入 debug 直接查看 “Mul_Cnt” 变量不同选择的结果。

```

.DEFINE Use_Flag      ;USE_Flag符号定义
DEBUG_MODE = 0
Use_DelayFlag = 1
Delay_Time = 0x10
Delay_Time1 = 0x20
Delay_Time2 = 0x40

;条件选择多层判断
IFDEF Use_Flag
    MOV     DPTR,#Mul_Cnt
    .IF DEBUG_MODE
        MOV     A,#Delay_Time
        MOVX    @DPTR,A
    .ELSE
        .IF Use_DelayFlag
            MOV     A,#Delay_Time1
            MOVX    @DPTR,A
            MOV     A,#Delay_Time1
            MOVX    @DPTR,A
        .ELSE
            MOV     A,#Delay_Time2
            MOVX    @DPTR,A
            MOV     A,#Delay_Time2
            MOVX    @DPTR,A
            MOV     A,#Delay_Time2
            MOVX    @DPTR,A
        .ENDIF
    .ENDIF
ENDIF
ENDIF

```

图8.4.2-1 条件汇编

可使用“.DEFINE”进行符号定义，例如：.DEFINE Use_Flag，仅用于符号定义，不可赋值或表达式操作，例如：.DEFINE Use_Flag 0x11，这样会编译错误，实际操作中，请根据以上使用方式，进行条件汇编处理。

使用“.IFNDEF、.ENDIF”进行条件汇编选择时，除能识别使用“.DEFINE”进行定义的符号，也可使用“.EQU、=”赋值操作定义的符号。

8.4.3 宏定义

汇编器支持简单的宏设计，基本的宏定义和使用的语法如下：

```

.macro _clrf fl
    mov     dptr,#fl
    mov     a,#0
    movx    @dptr,a
.endm

```

对于宏的使用，可以简单的这样调用，完成相应的操作

_clrf tmp ;清除 tmp 内容

8.4.4 area 定义内存区域指令

area 指令在汇编器中用于定义和切换内存区域，指定代码、数据或未初始化数据应该放

置在哪个内存段内。

格式: `.area` 区域名(区域属性)

举例:

```
.area      XSEG(ABS,XDATA)      ;切换到外部 RAM 段,且是绝对地址
.ORG      0x8002                ;指定内存地址
Sensor_Data: .DS 50              ;50 字节的传感器数据缓冲区
```

```
.area      PSEG (PAG)           ;分页外部存储区 XDATA,需硬件支持,因编译
需要必须写
```

```
.area      HOME (DATA)          ;可用于将变量分配到内部 RAM 的低 128 字
节, 支持直接寻址, 因编译需要, 必须写
```

```
.area      CODE (ABS,CODE)      ;切换到 CODE 代码段,且是绝对地址
main:
.ORG      0x0100
MOV       DPTR,# Sensor_Data    ;加载外部 RAM 地址到 DPTR
MOV       A,#0x55               ;A 取值 0x55
MOVBX     @DPTR,#A              ;将 0x55 写入到 Sensor_Data[0]
LJMP      main
```

8.4.5 ORG 汇编起始指令

ORG 是汇编语言中的伪指令, 它用于指定程序或数据在内存中的起始地址。

格式: `.ORG` 16 位地址(例如: 0x0000)

举例:

```
.area  XSEG (XDATA)
.area  PSEG (PAG)
.area  HOME (ABS,CODE)
.ORG  0x0000      ;程序起始地址
LJMP  main        ;跳转至主程序

.ORG  0x002B      ;定时器 2 中断入口地址
LJMP  _IT_Handler ;跳转至中断处理函数
```

`_IT_Handler:`

;开始编写中断处理程序

RETI ;退出中断程序

```
.ORG  0x0100 ;主程序起始地址
```

main:

;开始编写主程序内容

Loop_main:

;编写循环处理程序

LJMP Loop_main

8.4.6 include 预处理指令

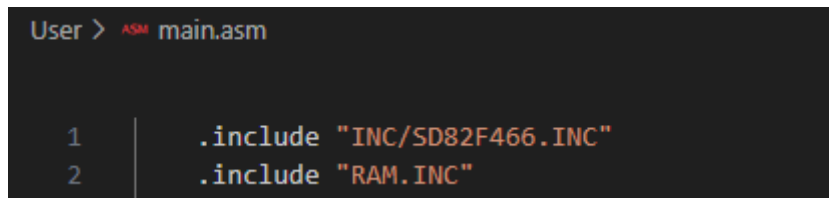
`include` 指令主要起预处理的功能，预处理命令在整个汇编过程中是最先被处理的，在汇编阶段将制定文件插入当前位置。

格式 1: `.include "文件名+后缀"`

举例: `.include "RAM.INC"`(与包含文件的 `asm` 源文件在同一个子目录下)

格式 2: `.include "指定工程目录下的文件名+后缀"`

举例: `.include "INC/SD82F466.INC"`(因为 `SD82F466.INC` 在工程目录下的子目录 `INC` 内，与包含文件的 `asm` 源文件不在同一个子目录下)。



```
User > ASM main.asm

1      .include "INC/SD82F466.INC"
2      .include "RAM.INC"
```

图 8.4.6-1 `include` 预处理指令举例

8.4.7 `equ` 赋值指令

`equ` 指令核心作用是将一个常数或一个特定的符号赋给规定的字符名称，在编译阶段遇到 `EQUQ` 前面的字符名称后，便会用 `EQU` 后面的数(或特定的符号)代替。

格式: `X .EQU Y`

举例: `AX .EQU 0x8000`

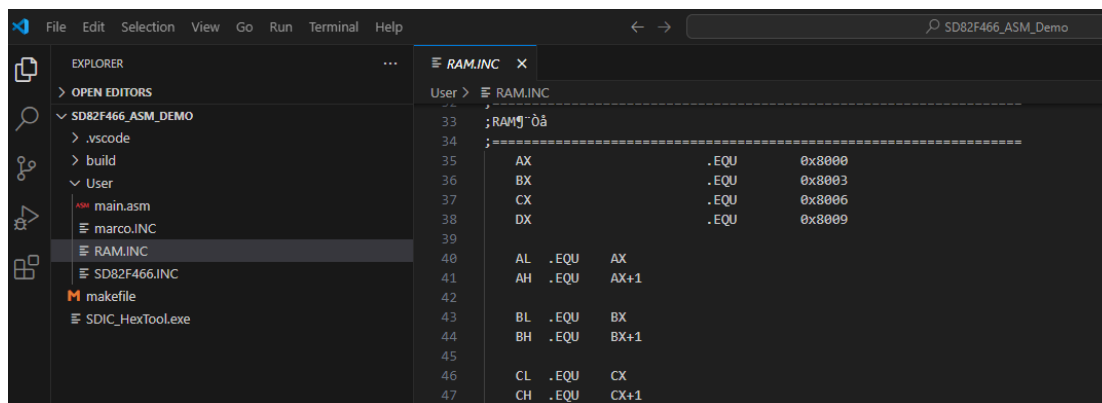


图 8.4.7-1 指令 `EQU` 使用格式

8.4.8 `macro` 和 `endm` 宏指令

`macro` 指令用于标记宏定义的起始位置，支持形参，与 `endm` 标记宏定义的终止位置成对出现，可用于将重复使用的指令序列封装为一个可调用的单元，宏内可包含汇编伪指令、汇编指令或汇编器宏伪指令。

格式: `.macro 字符名 参数`

`;指令代码`

`.endm`

举例:

```
.macro _SETF f1
MOV    DPTR,#f1
MOV    A,#0Xff
MOVX   @DPTR,A
.endm
```

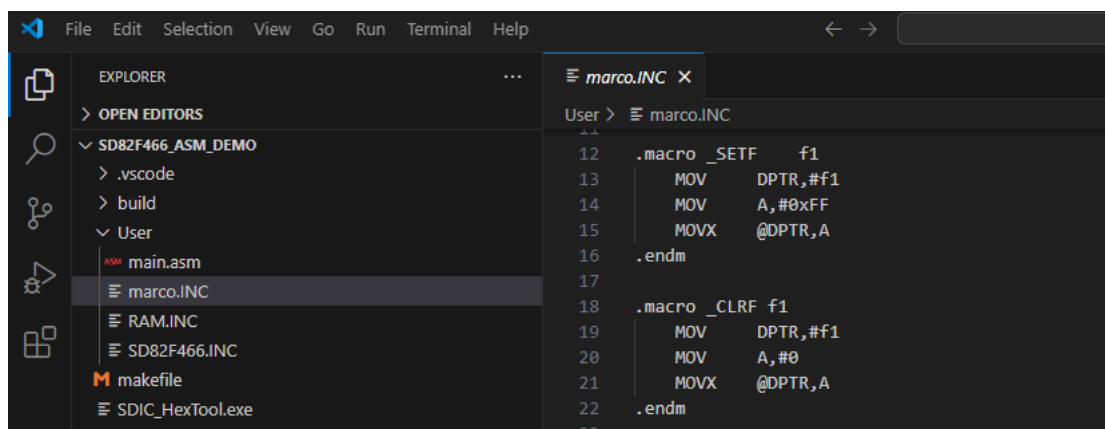


图 8.4.8-1 指令 macro 和 endm 使用格式

8.4.9 DB 字节定义指令

DB 字节定义指令功能是，从该地址开始，在程序存储器中定义一串字节单元，并用数据项进行赋值，该指令通常放在程序的最后，用于开辟表格。

格式：.DB 数据项(数据项之间用“,”号隔开)

举例：

```
.ORG 0x0400
```

```
TAB: .DB 0x80, 0x95, 0x74, 0x55
```

则经过编译后，程序存储器中：(0x0400)=0x80、(0x0401)=0x95、(0x0402)=0x74、(0x0403)=0x55。

8.4.10 DW 字定义指令

DW 字定义指令功能是，从该地址开始，在程序存储器中定义一串字节单元，并用数据项进行赋值，先存高字节，再存低字节，即：高字节放在低地址，低字节放在高地址。

格式：.DW 数据项

举例：

```
.ORG 0x0400
```

```
TAB: .DW 0x8095, 0x7455
```

则经过编译后，程序存储器中：(0x0400)=0x80、(0x0401)=0x95、(0x0402)=0x74、(0x0403)=0x55。

8.4.11 DS 存储空间预留指令

DW 字定义指令功能是，从该地址开始，保留 DS 之后表达式的值所规定的存储单元，以备后用。

格式：.DS 表达式

举例：

```
.ORG 0x0400
```

```
.DS 10
```

```
.DW 0x8095, 0x7455
```

则编译后，程序存储器中，从 0x0400 开始保留 10 个单元，而(0x040A)=0x80、(0x040B)=0x95、(0x040C)=0x74、(0x040D)=0x55。

8.4.12 汇编位操作指令

雪松系列芯片位寻址区位于 idata 区 0x20~0x2F 地址区间，16 个位寻址地址，每个地址包含 8 位，0x20~0x2F 地址，位地址对应 0x00~0x7F，0x20 地址对应的位地址为

0x00~0x07, 0x21 地址对应的位地址为 0x08~0x0F, 因此使用位操作指令时, 需要直接指定位地址就行, 下面以 SETB 置位操作指令举例说明, 其它位操作指令类似要求。

在 keil 中如果声明 “Temp_Flag .EQU 0x20”, 且声明位号 “b_ad_stable .EQU 7”, 在 keil 中可以使用指令 “SETB Temp_Flag.b_ad_stable” 将 Temp_Flag 变量第 7 位置 1, 在 SDCC 中无法使用 “.” 直接位操作, 只能直接对位寻址区进行操作, 在 SDCC 中以上指令需要改为 “SETB b_ad_stable”, 对应将 0x20 地址对应的第 7 位置 1。如果再声明位号 “b_ad_stable .EQU 0x0F”, 指令 “SETB b_ad_stable” 表示对应 0x21 地址对应的第 7 位置 1, 通过上述方法对 0x20~0x2F 地址内的位地址进行操作。

8.4.13 VSCODE 中无法识别的汇编指令

VSCODE 中无法识别 “.end” 指令, 因此不用加 .end 指令结束程序。“LOW” 和 “HIGH” 运算符分别使用 “<” 和 “>” 替代。无法识别 “\$” 字符。

8.5 跨文件汇编源程序

编写汇编程序或在 C 语言程序中跨文件编写汇编程序, 在未指定汇编程序存放的绝对地址时, 将自动分配非主程序内的汇编程序地址, 会与其它地址重合, 因此跨文件编写汇编程序时, 需要根据程序存放空间, 指定代码存放的绝对地址, 可通过 “.area HOME (ABS, CODE)” 及 “.ORG” 指定程序地址。指定的地址不可与其它程序地址重合, 实际应用中可根据 build 文件夹内的 “.rst” 和 “.map” 文件查看程序地址使用情况。

8.6 VSCODE 符号替换功能

遇到需要将一个符号替换为另外一个符号时, 可使用 VSCODE 的替换功能, 通过步骤 “鼠标点击一个文件, 然后 Ctrl+F” 右上角弹出输入框, 将需要查找或替换的字符写入, 如下图所示。

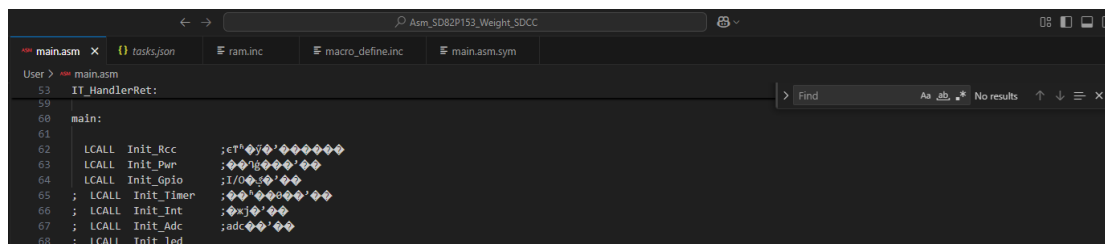


图 8.6-1 字符写入

例如输入 R_zeroAuto 字符串后, 会显示本文件内有多少个相同字符串, 如果想要将 R_zeroAuto 字符串, 替换为 R_zeroauto 字符串, 点击输入框左边的 “>” 符号, 弹出替换字符输入框, 输入 R_zeroauto 字符串, 点击单个替换或者全部替换, 可替换当前文件内的字符串。

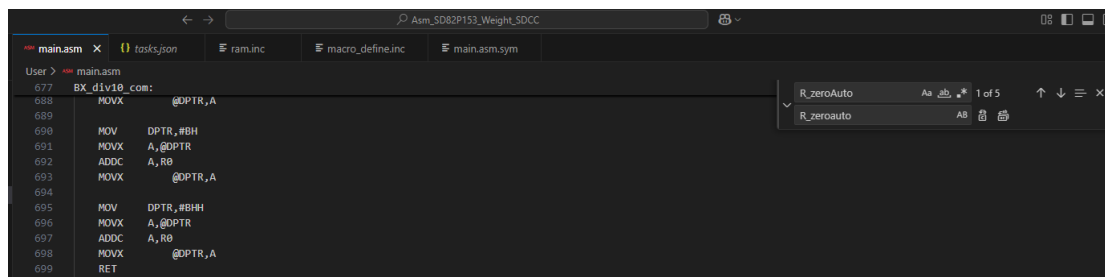


图 8.6-2 替换字符

8.7 错误(常见的错误)

8.7.1 重定位错误

如果在头文件中出现未定义的字符引用, 会出现 < > 提示的错误, 表示重定位错误, 错误格式如下, 错误行号为头文件行号。

build/源文件.sym:错误行号: Error: <r> relocation error

8.7.2 汇编指令错误

如果在头文件中没有正确使用助记符，会出现<o>提示的错误，错误格式如下，错误行号为头文件行号，例如将“.EQU”指令，错误的写为“EQU”字符。

build/源文件.sym:错误行号: Error: <o> .org in REL area or directive / mnemonic error

如果在源文件中错误使用助记符，错误提示格式如下：

源文件目录/源文件.asm:错误行号: Error: <o> .org in REL area or directive / mnemonic error

8.7.3 操作、终止、分隔符错误

如果在源文件内使用 VSCODE 编译器无法识别的符号，会出现<q> 提示的错误，错误格式如下，例如，VSCODE 中不需要使用 end 结束命令，但是程序使用了。

源文件目录/源文件.asm:错误行号:Error: <q> missing or improper operators, terminators, or delimiters

例如错误使用注释符，VSCODE 仅支持“;”号注释，当使用其它注释符时，例如“//”或“/**/”会出现上述错误。

8.7.4 未定义错误

如果在源文件中使用了未定义的符号，会出现未定义错误，格式如下，例如自定义的变量名称，大小写未注意：

?ASlink-Warning-Undefined Global '符号名' referenced by module "

8.7.5 未定义内存区域错误

汇编程序需要指定 XSEG、PSEG、HOME 内存区域，否者会出现未指定内存区域错误，格式如下：

ASlink-Warning-No definition of area 区域名。

9.修改记录

版本	修改日期	作者	修改记录
v0	2025-04-24	吴华桥	初始版本。
v0.1	2025-05-26	吴华桥	1、修改 C 语言应用程序中的关键字 sbit 和 bit 的说明； 2、修改修改 C 语言应用程序中的关键字 xdata 的说明； 3、修改汇编语言应用程序中的位操作指令说明； 4、增加汇编语言应用程序,汇编工程 Keil 转 VSCODE 的说明； 5、增加符号替换功能的使用说明。
v0.2	2025-06-03	吴华桥	1、增加汇编语法说明； 2、增加汇编指令/伪指令说明； 3、增加汇编常见错误说明； 4、增加创建汇编工程说明。