VSCode 使用指南

概述

Visual Studio Code 是一个非常流畅、快速且现代化的集成开发环境，具有许多酷炫的插件和辅助工具。它并不太常用于嵌入式开发，导致使用 Visual Studio Code 和嵌入式目标进行起步可能会有一些困难，但希望通过这份手册的帮助，能够帮助用户迅速上手。

在本文中，将使用 SPC1169 作为示例进行说明，**其它产品有类似步骤**。整份文档将涵盖安装、第一个简单项目、代码下载，以及代码调试。

SPINTROL

目录

1 工具安装 7
1.1 基础环境配置 7
1.2 VSCode 安装 8
1.3 安装扩展包 9
1.4 CMake 11
1.5 Make 12
1.6 Arm GNU Embedded Toolchain 13
1.7 J-LINK 安装和配置 14
1.8 硬件连接 15
2 编译和下载 16
2.1 打开开发工程（VSCode） 16
2.2 修改工具链 CMake 文件 17
2.3 编译及下载 18
2.4 编译及下载补充 19
2.5 GCC 下 printf 警告问题（不影响使用） 20
2.6 VSCode 与 CMake 配合问题（不影响使用） 21
2.6.1 VSCode 高亮显示错误问题（不影响使用） 21
2.6.2 VSCode 下无法识别 GCC 库文件问题（不影响使用） 21
2.7 终端不会自动关闭 21
3 调试程序（J-LINK） 22
3.1 Launch JSON File 22
3.2 DEBUG 23
3.3 JLinkARM.dll 暗桩 26
3.3.1 J-Link 超时调试异常 26
3.3.2 J-Flash 超时下载异常 27
3.3.3 J-Link 超时下载异常 28
3.4 DEBUG 补充 30
3.5 查看外设寄存器 32
3.6 查看 Memory 32

SPINTROL

©2023 旋智电子科技（上海）有限公司

图片列表

图 1-6：添加 data 文件夹 8
图 1-6：添加到 PATH 8
图 1-2：使用命令行安装扩展 9
图 1-3：扩展包图示 10
图 1-4：勾选添加到 PATH 11
图 1-5：查看 CMake 11
图 1-6：添加到 PATH 12
图 1-7：查看 Make 12
图 1-8：添加到 PATH 13
图 1-9：查看 GNU 13
图 1-8：卸载 J-LINK 14
图 1-9：添加到 PATH 14
图 1-10：查看 jlink 是否配置成功 14
图 1-11：配置 Flash 编程算法 15
图 1-12：JLinkDevices.xml 文件中添加 SPC1169 产品信息 15
图 2-1：添加文件夹到 VSCode 16
图 2-2：VSCode 文件列表 16
图 2-3：.cmake 文件 17
图 2-4：cd 到预编译工程 GCC 目录 18
图 2-5：卸载 J-LINK 18
图 2-6：使用简化指令 .... **错误!未定义书签。**
图 2-7：清除冗余进程 21
图 3-1：GDB 调试图示 22
图 3-2：打断点 25
图 3-3：弹窗 26
图 3-4：同步更新 JLinkARM.dll 26
图 3-5：调试异常 27
图 3-6：弹窗 27
图 3-7：超过 30s J-Link 指令下载异常 28
图 3-8：Debug 任务界面 30
图 3-9：单步调试 30
图 3-10：全速运行到断点 30
图 3-11：关闭冗余断点 31
图 3-12：查看外设寄存器 32
图 3-13：View Memory 32
图 3-14：Memory 数据 33

表格列表

SPINTROL

©2023 旋智电子科技（上海）有限公司

版本历史

版本	日期	作者	状态	变更
A/0	2023-9-15	Hang Su	Outdated	首次发布。
A/1	2023-9-19	Hang Su	Outdated	更新章节 1，2，3
A/2	2023-10-11	Hang Su	Released	更新章节 1，2，3，4

SPINTROL

术语或缩写

术语或缩写	描述

SPINTROL

©2023 旋智电子科技（上海）有限公司

1 工具安装

1.1 基础环境配置

Vscode: VSCode-win32-x64-1.82.2.zip（推荐）

CMake: cmake-3.27.3-windows-x86_64

Make: xpack-windows-build-tools-4.4.0-1-win32-x64

Arm GNU: arm-gnu-toolchain-12.2.mpacbti-rel1-mingw-w64-i686-arm-none-eabi

J-Link: JLink_Windows_V614b(V6.14b**)** （推荐）

SPINTROL

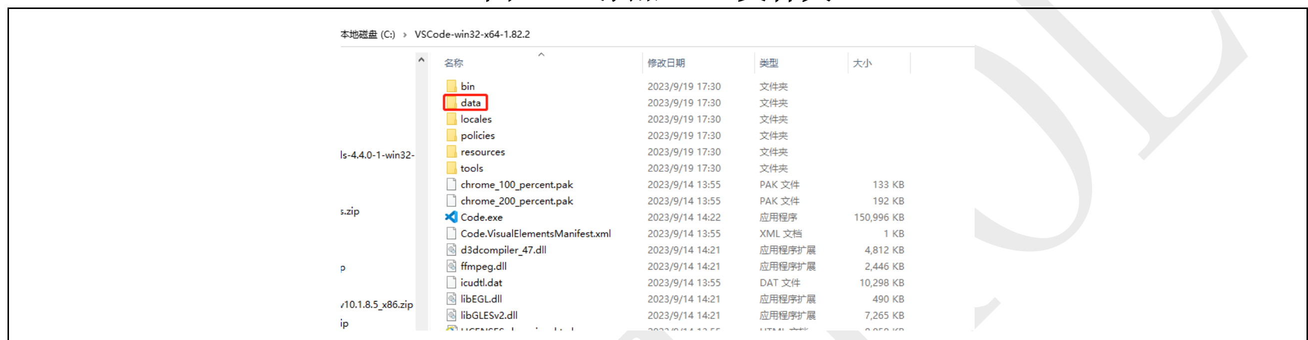
1.2 VSCode 安装

VSCode 共有 User Installer，System Installer，.zip 三种版本，下载链接见 https://code.visualstudio.com/Download。

其中.zip 安装下，插件目录控制最简单，其他版本插件会默认装到 C:\Users\XXX\.vscode（XXX 为当前用户名），如果当前用户名包含中文，插件将运行失败，修改插件目录的方法比较冗余，此处不演示。

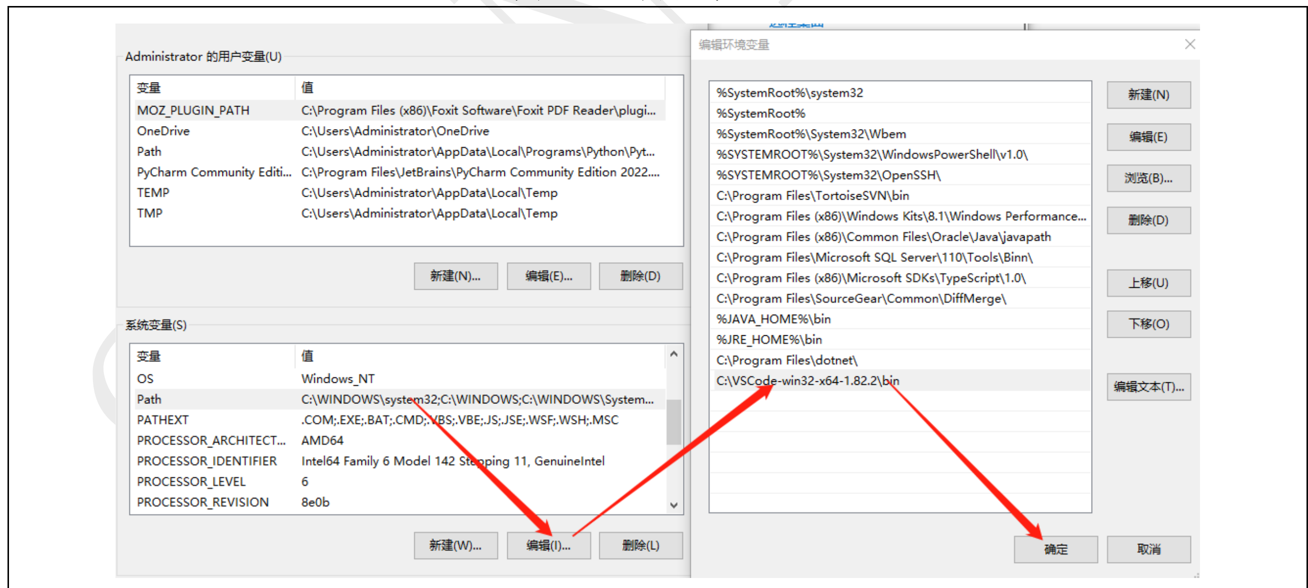
- 解压 VSCode-win32-x64-1.82.2.zip；
- 在 VSCode-win32-x64-1.82.2 下添加 data 文件夹（之后插件会自动安装到该目录）；

图 1-1：添加 data 文件夹



- **添加 bin 目录至环境变量 PATH（例如：C:\VSCode-win32-x64-1.82.2\bin）；**

图 1-2：添加到 PATH



- **重启，使得 PATH 生效。**

©2023 旋智电子科技（上海）有限公司

1.3 安装扩展包

如图 1-3 所示，在命令行中输入以下指令，安装扩展包。

```
code --install-extension ms-vscode.cpptools
code --install-extension marus25.cortex-debug
code --install-extension twxs.cmake
code --install-extension zchrissirhcz.cmake-highlight
code --install-extension dan-c-underwood.arm
code --install-extension ZixuanWang.linkerscript
```

图 1-3：使用命令行安装扩展

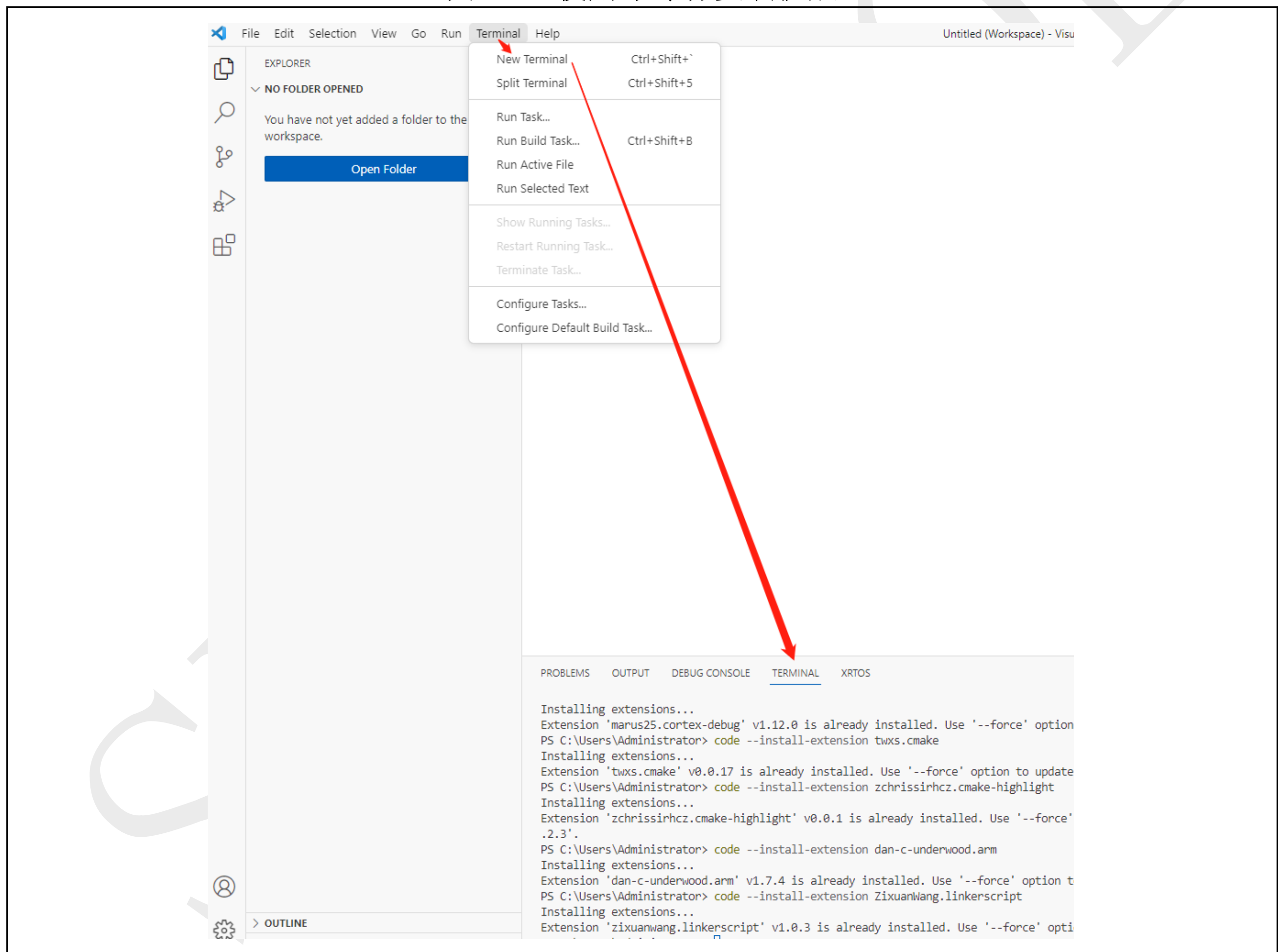
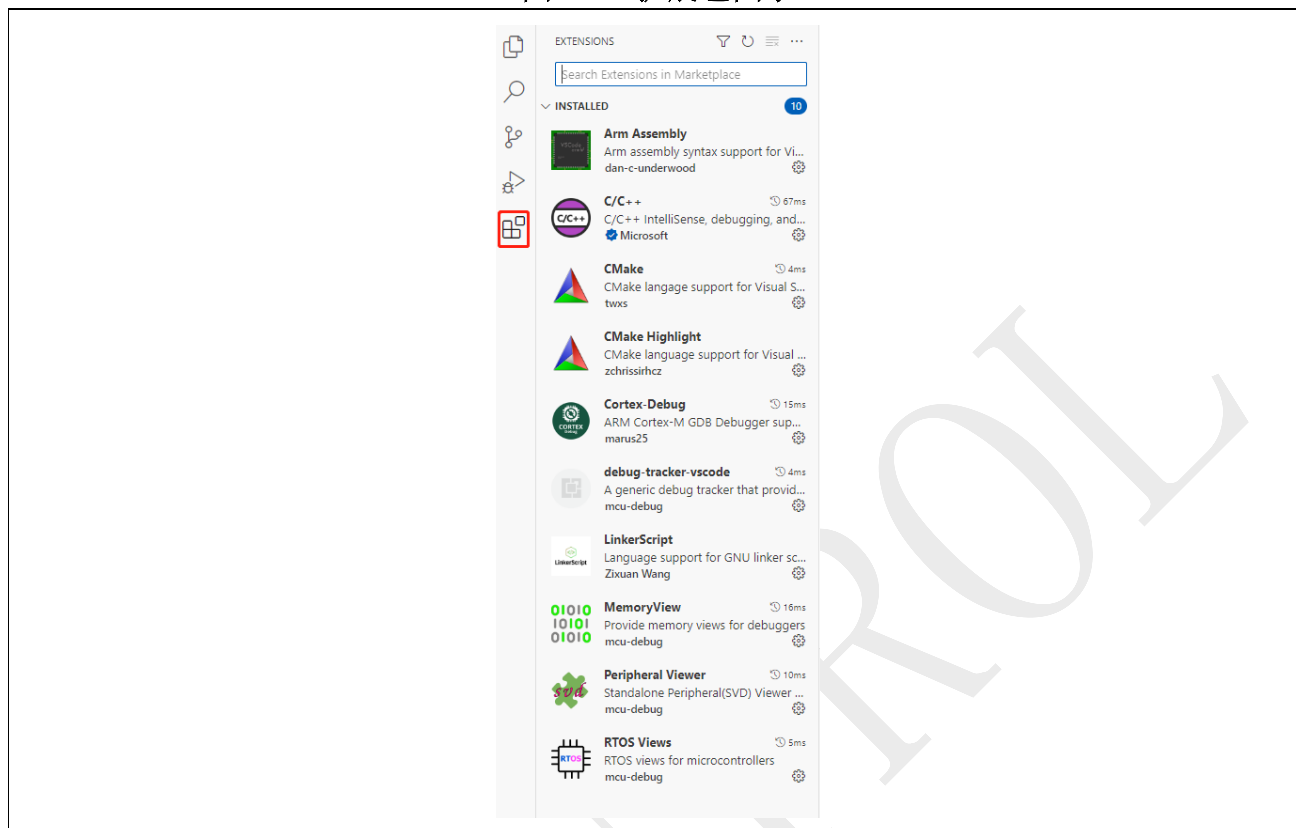


图 1-4 所示是安装扩展包后的视图。

图 1-4：扩展包图示



©2023 旋智电子科技（上海）有限公司

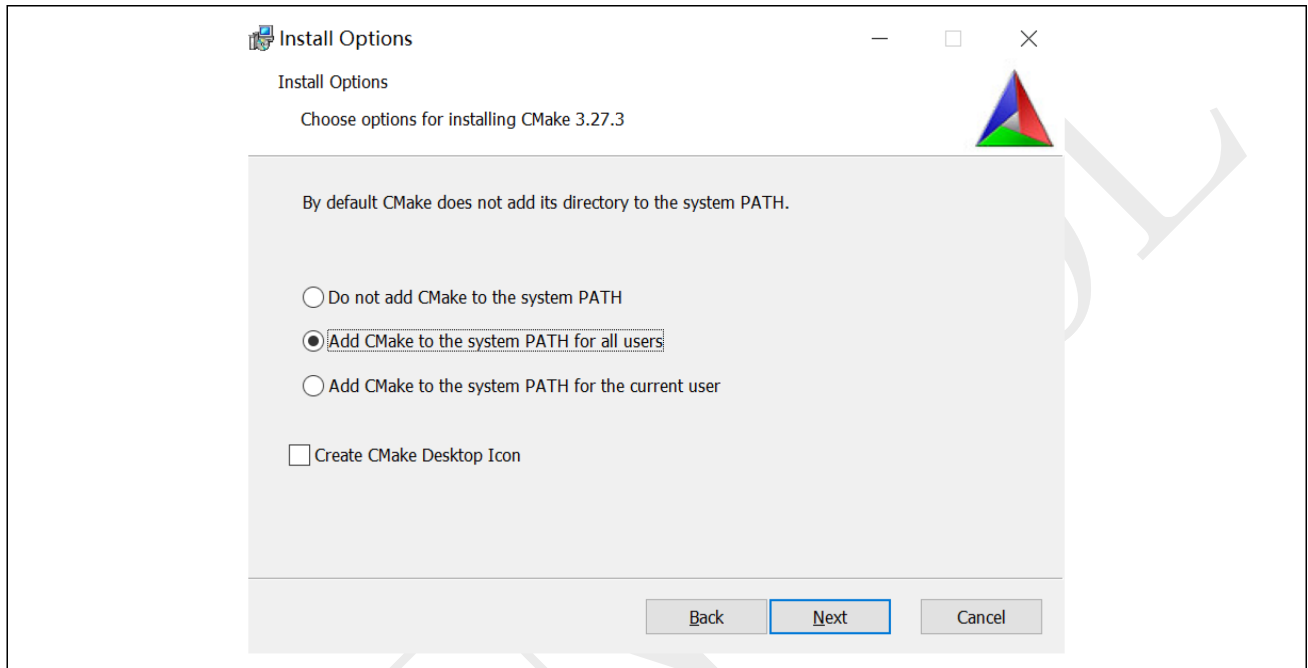
1.4 CMake

由于 VSCode 仅仅是一个编辑器 IDE，并不集成编译工具连的环境，所以为了能够在 VSCode 中编译代码，首先需要使用 Makefile 来组织编译工作。但编写 Makefile 组织一个大工程的编译是一件艰巨的工作，因此需要一个工具来辅助，这个工具就是 CMake。

CMake 可以通过 https://cmake.org/download/进行下载。

安装过程勾选添加到 PATH。

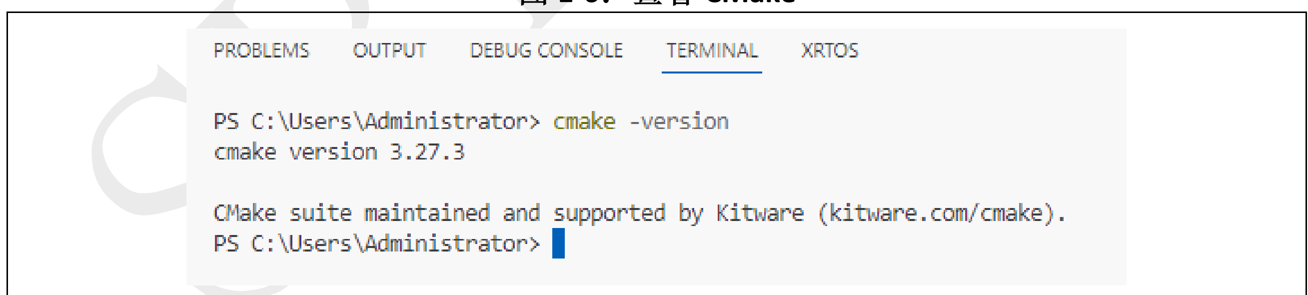
图 1-5：勾选添加到 PATH



安装后重启，使得 PATH 生效。

之后可查看是否添加成功。

图 1-6：查看 CMake



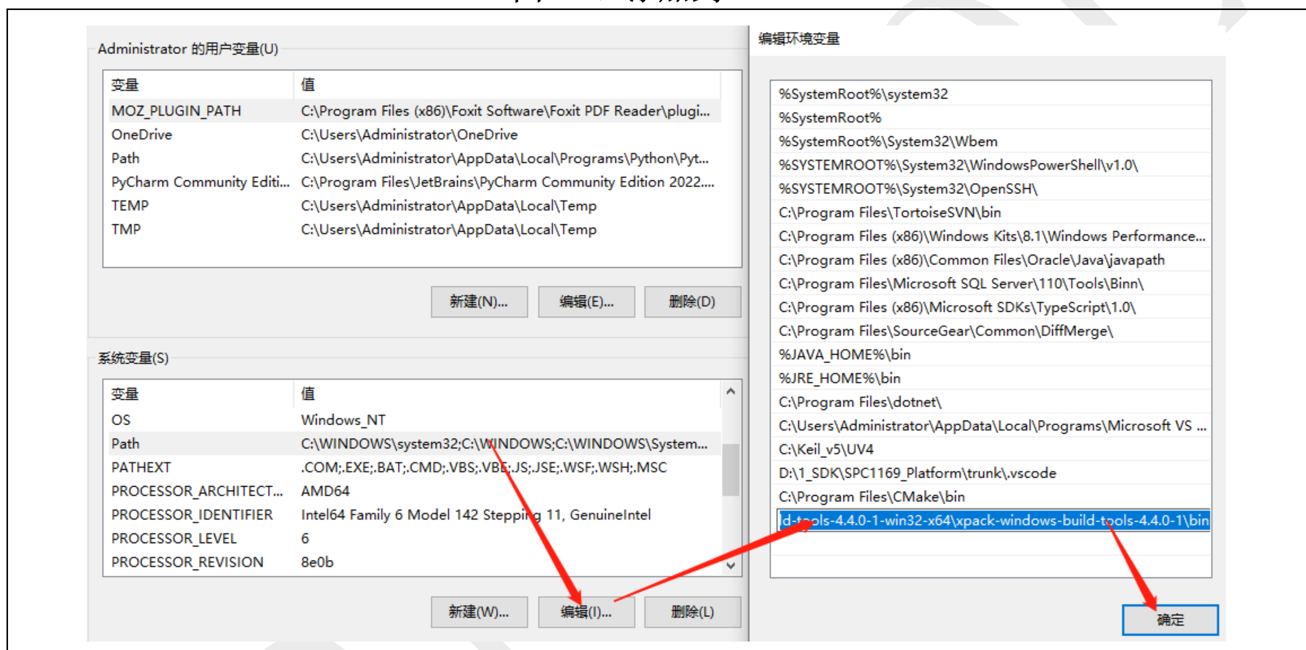
1.5 Make

VSCode IDE 并不集成任何编译工具链，仅仅是一个编辑器。在安装完 CMake 之后，我们就能够在它的帮助下产生一个工程的 Makefile 文件，但除此之外还需要一个能够识别 Makefile，并能按照其内容执行对应的动作的工具，这个工具就是 Make。

https://gitcode.net/mirrors/gnu-mcu-eclipse/windows-build-tools/-/releases

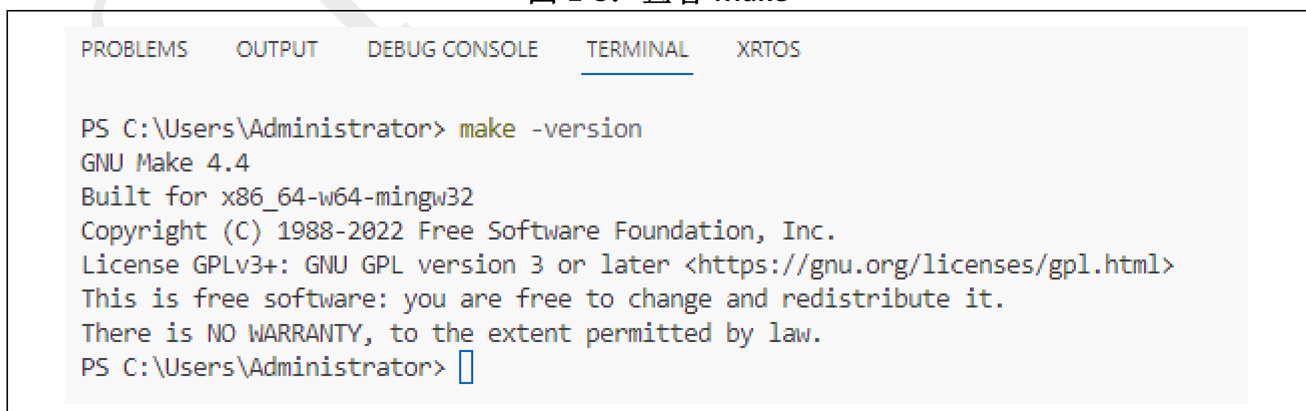
- 解压 xpack-windows-build-tools-4.4.0-1-win32-x64.zip；
- **添加 bin 目录至环境变量 PATH（例如：C:\xpack-windows-build-tools-4.4.0-1-win32-x64\xpack-windows-build-tools-4.4.0-1\bin）；**

图 1-7：添加到 PATH



重开终端可查看是否添加成功(新添加路径对旧终端无效), 如图 1-8 所示。

图 1-8：查看 Make



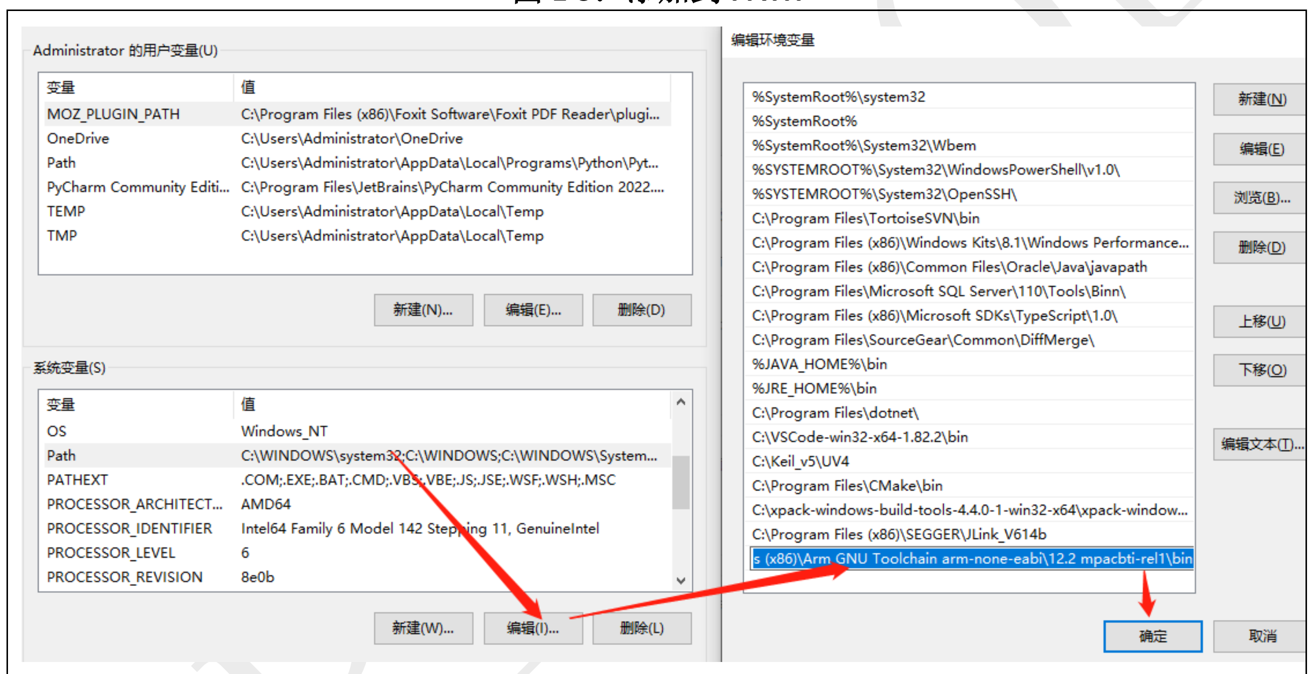
©2023 旋智电子科技（上海）有限公司

1.6 Arm GNU Embedded Toolchain

在有了 Make 之后，就能够安装 Makefile 中描述的编译流程，对源码工程进行编译，但要能正确编译出可执行文件还缺少最后一个步骤：需要使用正确的编译工具，对于需要运行在 ARM Cortex-M4 平台的源码文件来说，当然不能使用 Intel 平台下的 GCC 来编译，因为两个平台对应的指令集不一样。对于 ARM Cortex-M4 平台来说，可以在 Windows 下运行的交叉编译工具链 GNU toolchain for ARM 可以在 https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads 下载，本文作者安装的是 arm-gnu-toolchain-12.2.mpacbti-rel1-mingw-w64-i686-arm-none-eabi.exe。

- **添加 GNU 路径至环境变量 PATH（例如：C:\Program Files (x86)\Arm GNU Toolchain arm-none-eabi\12.2 mpacbti-rel1\bin）；**

图 1-9：添加到 PATH



重开终端可查看是否添加成功(新添加路径对旧终端无效), 如图 1-10 所示。

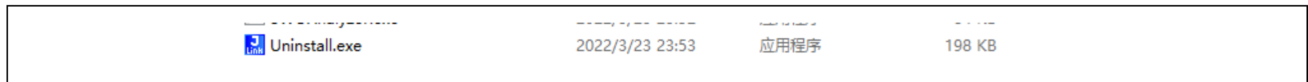
图 1-10：查看 GNU

```
PS C:\Users\Administrator> arm-none-eabi-gcc --help
Usage: arm-none-eabi-gcc.exe [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase.
  --help                   Display this information.
  --target-help            Display target specific command line options (including assembler and linker options).
  --help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[,...].
                           Display specific types of command line options.
  (Use '-v --help' to display command line options of sub-processes).
  --version                Display compiler version information.
  -dumpspecs               Display all of the built in spec strings.
  -dumpversion             Display the version of the compiler.
  -dumpmachine             Display the compiler's target processor.
```

1.7 J-LINK 安装和配置

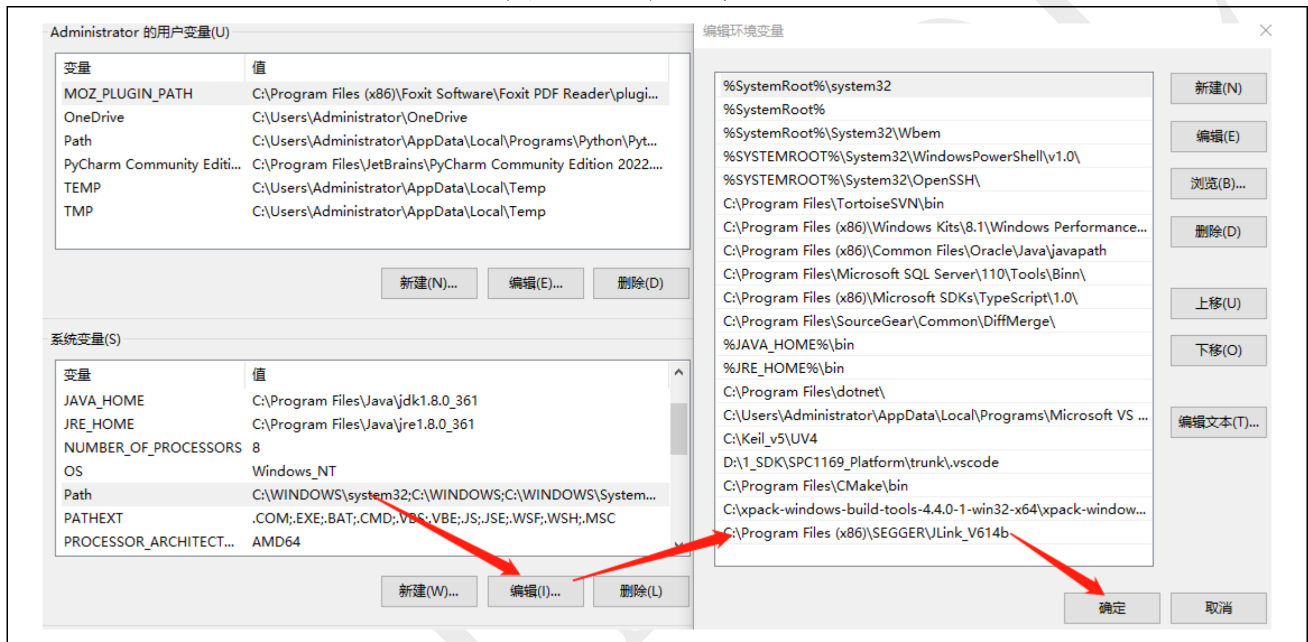
如已经安装 J-LINK，卸载后重装。

图 1-11：卸载 J-LINK



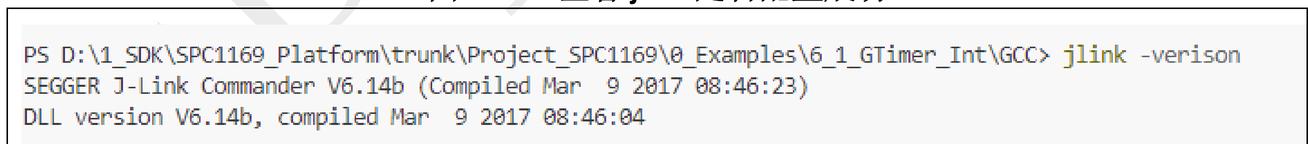
J-LINK 的各版本驱动可在 https://www.segger.com/downloads/jlink#J-LinkSoftwareAndDocumentationPack 进行下载，本文作者使用的是 V6.14b 版本。安装过程中保持默认配置即可，无需任何更改。**将 jlink.exe 路径添加到环境变量 PATH。例如：C:\Program Files (x86)\SEGGER\JLink_V614b**

图 1-12：添加到 PATH



重开终端可查看是否添加成功(新添加路径对旧终端无效)。

图 1-13：查看 jlink 是否配置成功



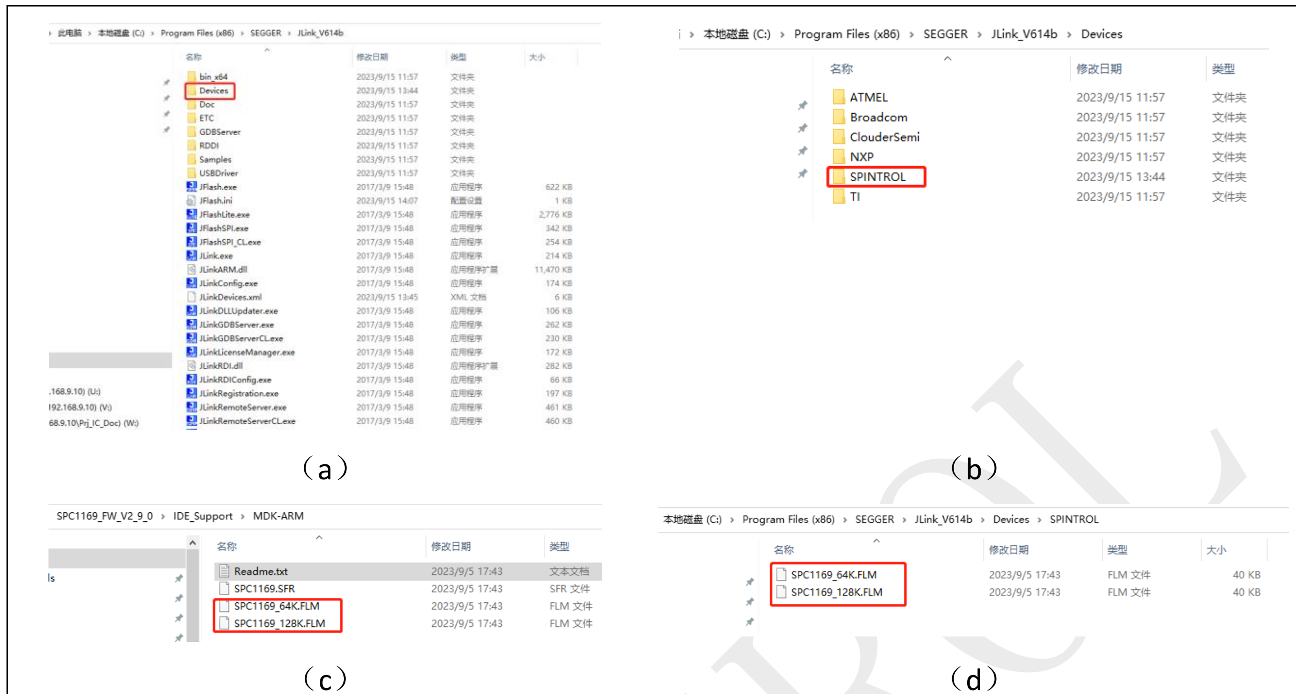
```
PS D:\1_SDK\SPC1169_Platform\trunk\Project_SPC1169\0_Examples\6_1_GTimer_Int\GCC> jlink -verison
SEGGER J-Link Commander V6.14b (Compiled Mar  9 2017 08:46:23)
DLL version V6.14b, compiled Mar  9 2017 08:46:04
```

以 SPC1169 产品为例，如图 1-14 所示，首先打开 J-LINK 软件所在的安装目录，在 Devices 文件夹下新建文件夹 SPINTROL，并打开 SPC1169 芯片的 SDK，将 IDE_Support\MDK-ARM 目录下的 FLM 文件复制到 J-Link 驱动安装路径下的 Devices\SPINTROL 文件夹中。

– Devices\SPINTROL 文件夹若不存在，则需要手动新建 SPINTROL 文件夹。

©2023 旋智电子科技（上海）有限公司

图 1-14：配置 Flash 编程算法



（a） （b）

（c） （d）

在 J-LINK 软件所在的安装目录下，找到 JLinkDevices.xml 文件，然后将 SPC1169 产品信息添加到 JLinkDevices.xml 文件中，添加完成后如图 1-15 所示。具体要添加的 SPC1169 产品信息如下：

图 1-15：JLinkDevices.xml 文件中添加 SPC1169 产品信息

```
<Device>
   <ChipInfo Vendor="Spintrol" Name="SPC1169_128K" WorkRAMAddr="0x1fffc000"
WorkRAMSize="0x4000" Core="JLINK_CORE_CORTEX_M4" />
   <FlashBankInfo Name="FLASH (Main)" BaseAddr="0x10000000"
MaxSize="0x00020000" Loader="Devices/SPINTROL/SPC1169_128K.FLM"
LoaderType="FLASH_ALGO_TYPE_OPEN" AlwaysPresent="1" />
 </Device>
```

注：1.在添加设备信息时，请根据具体产品设置 Flash（Main）的起始地址（BaseAddr），最大大小（MaxSize），详细步骤请参见相关产品《J-Flash 软件烧录使用指南》。

1.8 硬件连接

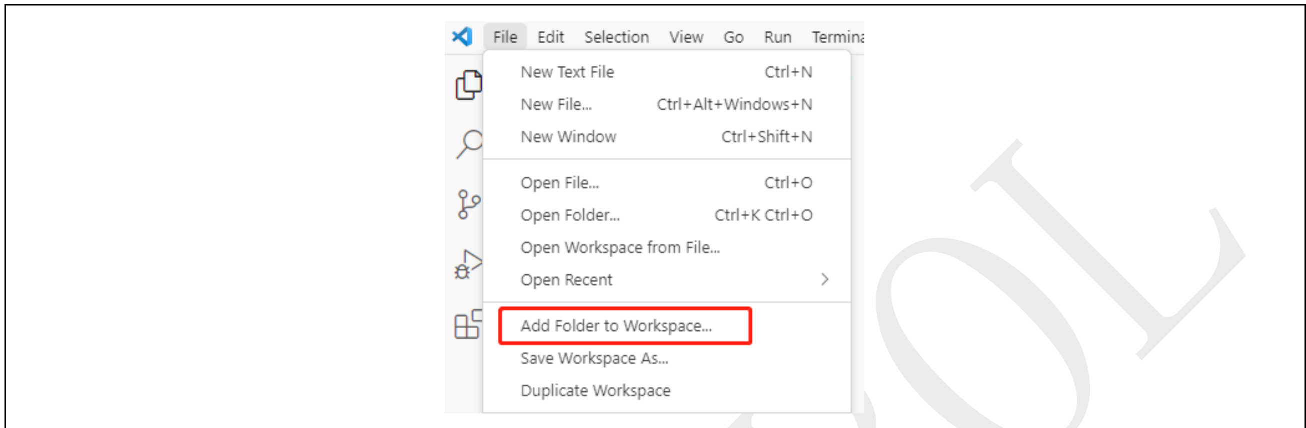
将 J-Link 设备和目标芯片的 SWD 接口连接，然后将 J-Link 设备通过 USB 线连接到电脑。在烧录之前，确保目标芯片正常上电工作。

2 编译和下载

2.1 打开开发工程（VSCode）

打开 VSCode 之后，如图 2-1 所示，选择添文件夹到 VSCode 中。

图 2-1：添加文件夹到 VSCode

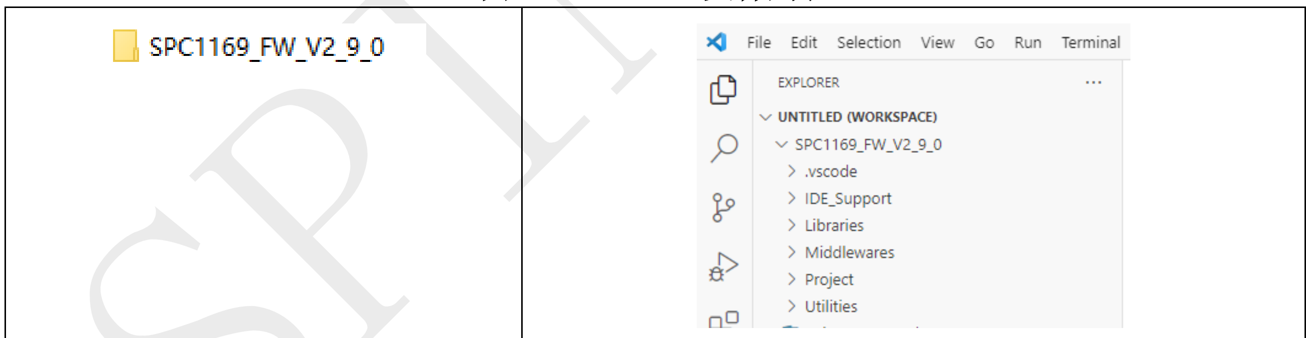


选择根目录文件夹。

选择之后，能够在 VSCode 的文件列表中看到所选择的文件夹，及其子文件夹，如图 2-2 所示。

注意：必须选择根目录，否则仿真会因找不到文件报错。

图 2-2：VSCode 文件列表

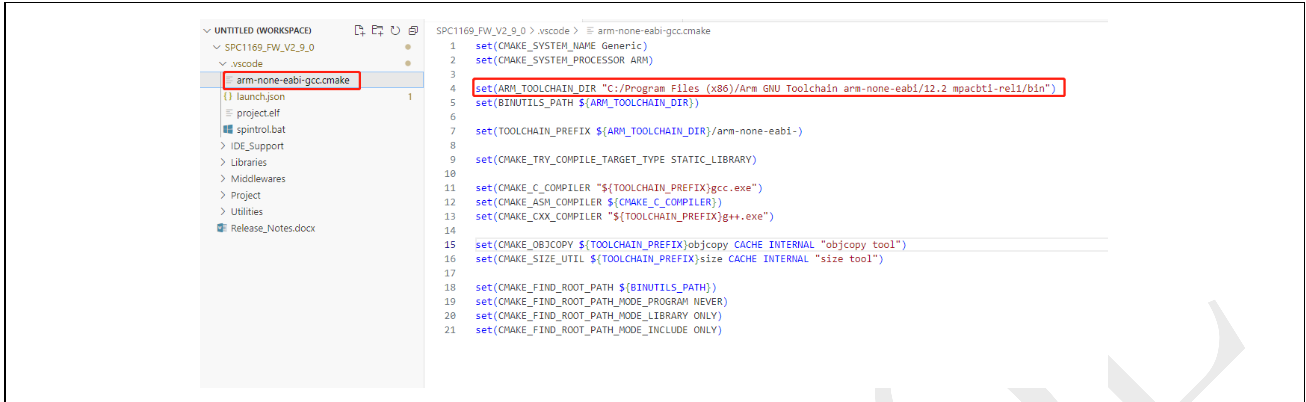


©2023 旋智电子科技（上海）有限公司

2.2 修改工具链 CMake 文件

需要替换成如 1.6 小节所述用户安装 ARM 交叉工具链的路径，如图 2-3 所示。

图 2-3：.cmake 文件

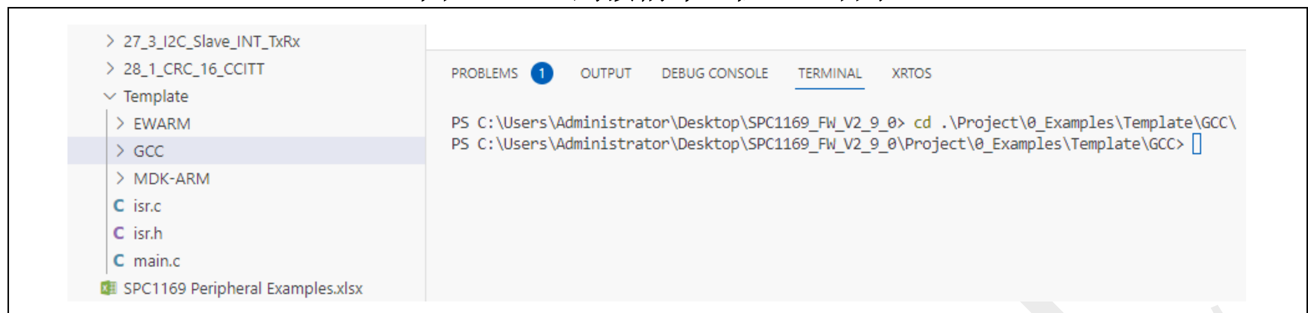


SPINTROL

2.3 编译及下载

在终端中通过 cd 指令进入预编译工程 GCC 目录，如图 2-4 所示。

图 2-4：cd 到预编译工程 GCC 目录



在终端中输入如下命令，即可完成源码编译。

```
cmake -G "Unix Makefiles" ./; make
```

如果想清除编译产生的文件，可以执行如下指令：

```
make clean-all
```

如果想快速下载，可以执行如下指令（前提 J-LINK 已经配置完毕，详见章节 1.7）：

```
jlink.exe -device SPC1169_128K -if SWD -speed 10000 -autoconnect 1 -CommandFile .\download.jlink
```

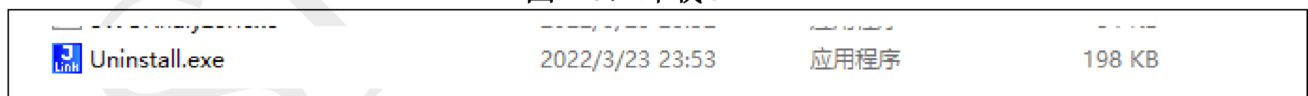
如果想编译并下载，可以执行如下指令（前提 J-LINK 已经配置完毕，详见章节 1.7）：

```
cmake -G "Unix Makefiles" ./; make;jlink.exe -device SPC1169_128K -if SWD -speed 10000 -autoconnect 1 -CommandFile .\download.jlink
```

指令中的-device，保持和 J-LINK 中用到的设备号一致，如图 1-15 所示。

如果执行不成功，请卸载 J-LINK，重新按章节 1.7 安装和配置。

图 2-5：卸载 J-LINK



©2023 旋智电子科技（上海）有限公司

2.4 编译及下载补充

GCC 工程下的 bat 脚本提供了简化下载指令，相对快速下载，其指令长度更短。

```
PS D:\1_SDK\SPC1169_Platform\trunk\Project_SPC1169\0_Examples\Template\GCC> .\download.bat
```

SPINTROL

2.5 GCC 下 printf 警告问题（不影响使用）

ARMCC 将 uint32_t 定义为 unsigned int，GCC 将 uint32_t 的定义为 long unsigned int。在 32 位编译器下，int 和 long int 都是占 4 个字节，长度完全等价。

GCC 中，printf(“%d\n”,u32cnt)会报警，因为其认为%d 为 int，u32cnt 为 long unsigned int，并不知道 int 和 long unsigned int 实际等长。

可通过显性声明解决编译警告。例如：

```
printf("%" PRIu32 "\n", u32cnt);//无符号 32bit
printf("ECAP->CAP0 = %" PRIu32 "\n", TStamp0);//无符号 32bit
printf("ECAP->CAP0 = %" PRIx32 "\n", TStamp0);//16 进制 32bit
```

2.6 GCC 下局部变量必须赋初值

如果局部变量未赋初值，Keil 和 GCC 的处理方式不同，Keil 会默认其是 0，GCC 不会。

GCC 下必须对局部变量赋初值，消除“is used uninitialized”警告，否则其值是任意的。

```
int main(void)
{
    uint16_t u16a;

    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);
    Delay_Init();

    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("u16a is %d\n", u16a);

    while (1)
    {
    }
```

LEMS 3   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   MEMORY   XRTOS

```
MAKE_TOOLCHAIN_FILE is: D:/1_SDK/SPC1169_Platform/trunk/.vscode/arm-none-eabi-gcc.cmake
onfiguring done (3.9s)
enerating done (0.0s)
uild files have been written to: D:/1_SDK/SPC1169_Platform/trunk/Project_SPC1169/0_Examples/Template/GCC
%] Building C object CMakeFiles/project.elf.dir/D_/1_SDK/SPC1169_Platform/trunk/Project_SPC1169/0_Examples/Template/main.c.obj
_SDK/SPC1169_Platform/trunk/Project_SPC1169/0_Examples/Template/main.c: In function 'main':
_SDK/SPC1169_Platform/trunk/Project_SPC1169/0_Examples/Template/main.c:40:5: warning: 'u16a' is used uninitialized [-Wuninitialized]
0 |     printf("u16a is %d\n", u16a);
  |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~
_SDK/SPC1169_Platform/trunk/Project_SPC1169/0_Examples/Template/main.c:31:14: note: 'u16a' was declared here
1 |     uint16_t u16a;
  |              ^~~~
```


©2023 旋智电子科技（上海）有限公司

2.7 VSCode 与 CMake 配合问题（不影响使用）

2.7.1 VSCode 高亮显示错误问题（不影响使用）

虽然 CMake 编译分支与 CMakeLists.txt 中宏定义一致，但 VSCode 编辑器无法识别 CMakeLists.txt 中-D 配置，#else 分支永远高亮显示。

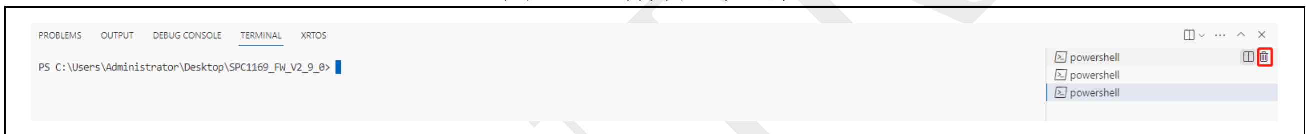
2.7.2 VSCode 下无法识别 GCC 库文件问题（不影响使用）

虽然 CMake 编译可以找到#include <XXX.h>并编译成功，但 VSCode 编辑器仅可以识别#include "XXX.h"，无法识别#include <XXX.h>。通过右键菜单中 Go to Definition 无法完成跳转。

2.8 终端不会自动关闭

终端不会自动关闭，平时开多了要手动关下，避免占用资源。

图 2-6：清除冗余进程

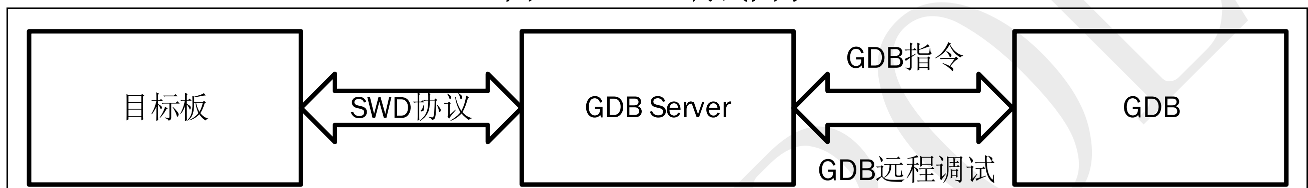


3 调试程序（J-LINK）

本文将展示如何使用 Microsoft Visual Studio Code 调试 ARM Cortex-M4 微控制器。为此，请务必安装好第一章所述的扩展包。本文描述的调试内容所使用的调试工具是 J-Link，且使用 SWD 模式，但也可以使用 JTAG。

因为在安装 J-LINK 时，也会一起安装 J-LINK GDB Server，所以可以先启动这个 GDB Server，然后使用 1.5 小节安装的 ARM 交叉编译工具链中的 GDB 链接这个 GDB Server，从而达到调试程序的目的，如图 3-1 所示。但这个过程中所需要使用的 GDB 命令非常多，会使得调试效率变慢，且过程让人抓狂，所幸，可以通过配置 VSCode，让 VSCode 帮助我们做这些事情，以下内容将详细描述配置及使用步骤。

图 3-1：GDB 调试图示



3.1 Launch JSON File

在 Visual Studio Code (VSCode)中，launch.json 文件用于配置和定义调试器的启动配置。它是一种 JSON 格式的文件，用于指定调试会话的各种设置，如启动程序、传递命令行参数、设置断点等。

以下是 launch.json 文件中需要根据本地替换的路径：

– **"device"**：J-LINK 所需的设备 ID，保持和 J-LINK 中用到的设备号一致，如图 1-15 所示；

launch.json 文件

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit:
https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "cwd": "${workspaceFolder}",
            "executable": "${workspaceFolder}/.vscode/project.elf",
//POST_BUILD COMMAND already copy project.elf to .vscode/
            "name": "cortex-debug",
            "request": "launch",
            "type": "cortex-debug",
            "servertype": "jlink",
            "serverpath": "C:/Program Files
(x86)/SEGGER/JLink_V614b/JLinkGDBServerCL.exe",
            "armToolchainPath": "C:/Program Files (x86)/Arm GNU Toolchain arm-
none-eabi/12.2 mpacbti-rel1/bin",
            "device": "SPC1169_128K",
            "interface": "swd",
            "serialNumber": "", //If you have more than one J-Link probe, add the
serial number here.
```


©2023 旋智电子科技（上海）有限公司

```
            "runToEntryPoint": "main",
            "svdFile": "${workspaceFolder}/IDE_Support/EWARM/SPC1169.svd",
        }
    ]
}
```

3.2 DEBUG

– 修改 launch.json 文件中"executable"与被编译工程 project.elf 一致，默认这一步可省略，因为 CMakeLists.txt 中的 POST_BUILD 部分在编译完成后自动将被编译工程 project.elf 移动到.vscode 目录下；（**当被编译工程目录深度改变时，必须同步修改 CMakeLists.txt 文件中 arm-none-eabi-gcc.cmake， .c, .h, .vscode 的相对路径，否则编译仿真必然不符合预期**）

CMakeLists.txt 文件

```
cmake_minimum_required(VERSION 3.15.3)

# Optional: print out extra messages to see what is going on. Comment it to
have less verbose messages
set(CMAKE_VERBOSE_MAKEFILE ON)

# Path to toolchain file. This one has to be before 'project()' below
set(CMAKE_TOOLCHAIN_FILE ${CMAKE_SOURCE_DIR}/../../../../.vscode/arm-none-eabi-
gcc.cmake)

# Setup project, output and linker file
project(project)
set(EXECUTABLE ${PROJECT_NAME}.elf)
set(LINKER_FILE ${CMAKE_SOURCE_DIR}/project.ld)

enable_language(C ASM)
set(CMAKE_C_STANDARD 99)
set(CMAKE_C_STANDARD_REQUIRED ON)
set(CMAKE_C_EXTENSIONS OFF)

# Optional: issue a message to be sure it uses the correct toolchain file.
message(STATUS "CMAKE_TOOLCHAIN_FILE is: ${CMAKE_TOOLCHAIN_FILE}")

# List of source files
set(SRC_FILES
        ${CMAKE_SOURCE_DIR}/../main.c
        ${CMAKE_SOURCE_DIR}/../isr.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/adc.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/can.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/clock.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/comp.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/crc.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/dma.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/ecap.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/gpio.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/hwlib.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/i2c.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/pga.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/phcomp.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/pin.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/power.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/pwm.c
        ${CMAKE_SOURCE_DIR}/../../../../Libraries/drivers/src/spi.c
```

```
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/drivers/src/system.c
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/drivers/src/timer.c
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/drivers/src/uart.c
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/drivers/src/wdt.c
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/CMSIS/device/system_spc1169.c

${CMAKE_SOURCE_DIR}/../../../../../Libraries/CMSIS/device/startup/gcc/startup_spc1
169.S
        ${CMAKE_SOURCE_DIR}/../../../../../Utilities/delay.c
        ${CMAKE_SOURCE_DIR}/../../../../../Utilities/retarget.c
        )

# Build the executable based on the source files
add_executable(${EXECUTABLE} ${SRC_FILES})

# List of compiler defines, prefix with -D compiler option
target_compile_definitions(${EXECUTABLE} PRIVATE

        )

# List of includ directories
target_include_directories(${EXECUTABLE} PRIVATE
        ${CMAKE_SOURCE_DIR}/../
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/drivers/inc
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/drivers/inc/reg
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/CMSIS/core
        ${CMAKE_SOURCE_DIR}/../../../../../Libraries/CMSIS/device
        ${CMAKE_SOURCE_DIR}/../../../../../Utilities
        )

# Compiler options
target_compile_options(${EXECUTABLE} PRIVATE
        -mcpu=cortex-m4
        -mthumb
        -mfpu=fpv4-sp-d16
        -mfloat-abi=hard
        -fdata-sections
        -ffunction-sections
        -Wall
        -O0
        -g3
        -DSPC1169
        )

# Linker options
target_link_options(${EXECUTABLE} PRIVATE
        -T${LINKER_FILE}
        -mcpu=cortex-m4
        -mthumb
        -mfpu=fpv4-sp-d16
        -mfloat-abi=hard
        -specs=nano.specs
        -specs=nosys.specs
        -lrdimon -u _printf_float
        -lc
        -lm
        -Wl,-Map=${PROJECT_NAME}.map,--cref
        -Wl,--gc-sections
        -Wl,--no-warn-rwx-segments
        -Xlinker -print-memory-usage -Xlinker
        )

# Optional: Print executable size as part of the post build process
```


©2023 旋智电子科技（上海）有限公司

```
add_custom_command(TARGET ${EXECUTABLE}
        POST_BUILD
        COMMAND ${CMAKE_SIZE_UTIL} ${EXECUTABLE})

# Optional: Create hex, bin and S-Record files after the build
add_custom_command(TARGET ${EXECUTABLE}
        POST_BUILD
        COMMAND ${CMAKE_OBJCOPY} -O ihex ${EXECUTABLE} ${PROJECT_NAME}.hex
        # COMMAND ${CMAKE_OBJCOPY} -O binary ${EXECUTABLE} ${PROJECT_NAME}.bin
        COMMAND ${CMAKE_COMMAND} -E copy ${CMAKE_SOURCE_DIR}/project.elf
${CMAKE_SOURCE_DIR}/../../../../.vscode/ )

add_custom_target(clean-all
        rm -rf CMakeFiles
        rm -rf CMakeCache.txt
        rm -rf cmake_install.cmake
        rm -rf Makefile
        rm -rf project.hex
        rm -rf project.map
        rm -rf project.elf
    )
```

- 可以提前在 IDE 中双击行数左侧打断点，如图 3-2 所示；
- 之后按快捷键 F5 开启 Debug 任务，其会自动调用 J-LINK 进行下载，并开启仿真；
- 如果执行不成功，重新按章节 1.6，章节 1.7 配置环境变量。

图 3-2：打断点

```
28
29  int main(void)
30  {
31      CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);
32      Delay_Init();
33
34      PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
35      PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
36      UART_Init(UART0, 38400);
37
38      printf("Just a Sample....\n");
39
40      while (1)
41      {
42      }
43  }
```

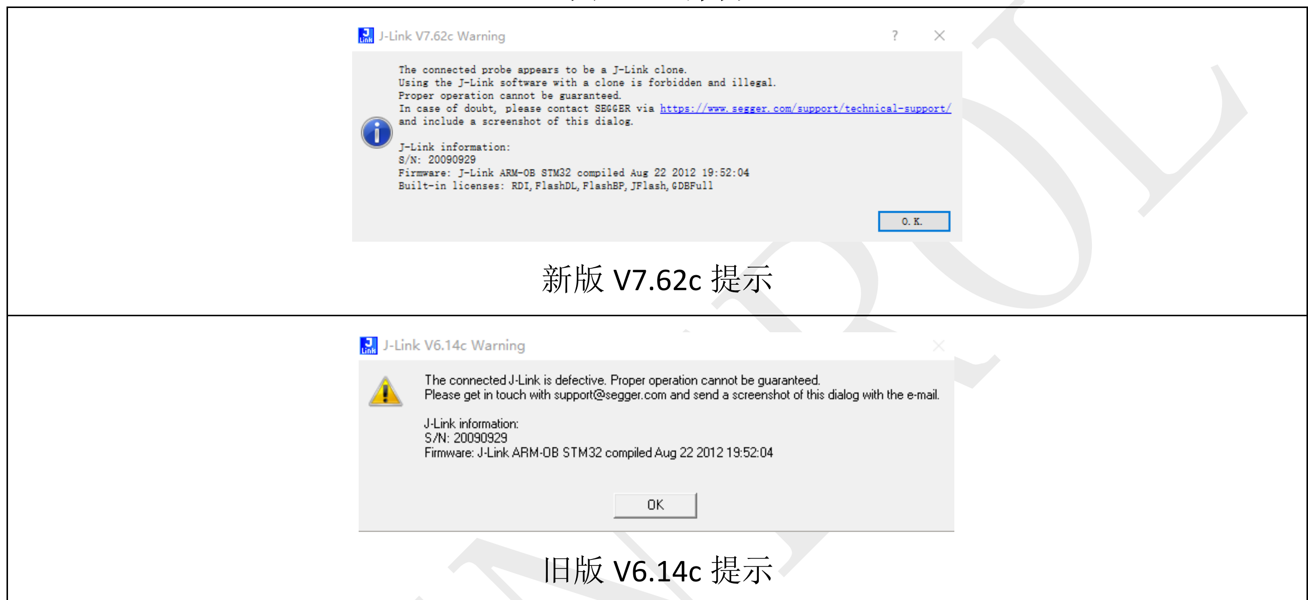
3.3 JLinkARM.dll 暗桩

J-Link 从 **V6.14c[2017-03-31]**开始，加入了暗桩，给下载，仿真，量产烧录过程中的问题定位带来极大不便。

并在之后的版本例如 **V7.62c** 中迭代到 **5s Warning**，**30s** 自动断开的组合逻辑。

如果调试过程中 **J-Link** 有弹窗警告，**" The connected J-Link is defective"**或者**" The connected probe appears to be a j-link clone"**，点 **OK** 忽略没用，因为 **JLinkARM.dll** 中存在暗桩，**30s** 后暗桩会在无任何通知情况下自动断开 **J-Link** 连接。

图 3-3：弹窗



新版 V7.62c 提示

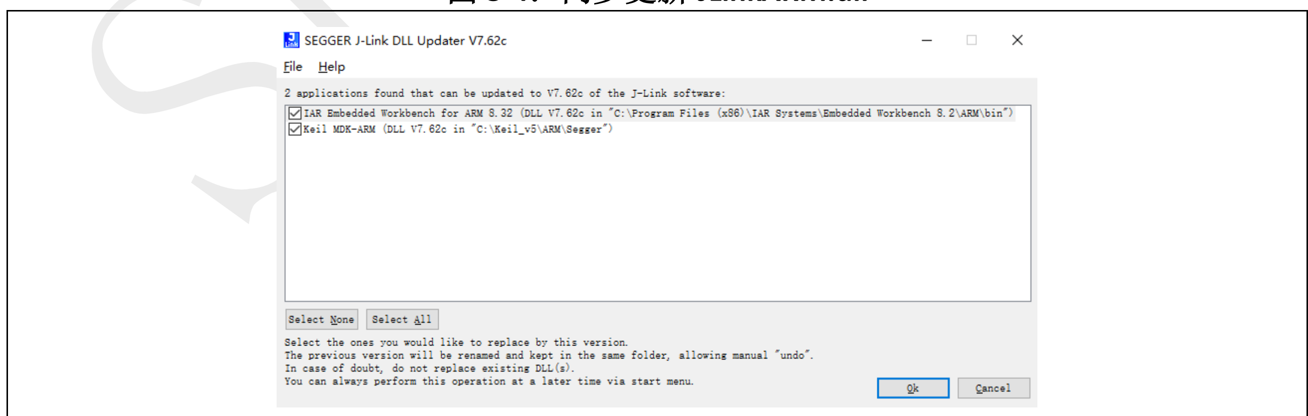
旧版 V6.14c 提示

3.3.1 J-Link 超时调试异常

因为断开事先毫无通知，并且是发生在 **30s** 后，容易误导为芯片有问题或 **GCC** 环境有问题，实际和芯片以及 **GCC** 环境无任何关系。

如果安装过程选择同步更新到 **keil** 或 **IAR**，如图 3-4 所示。

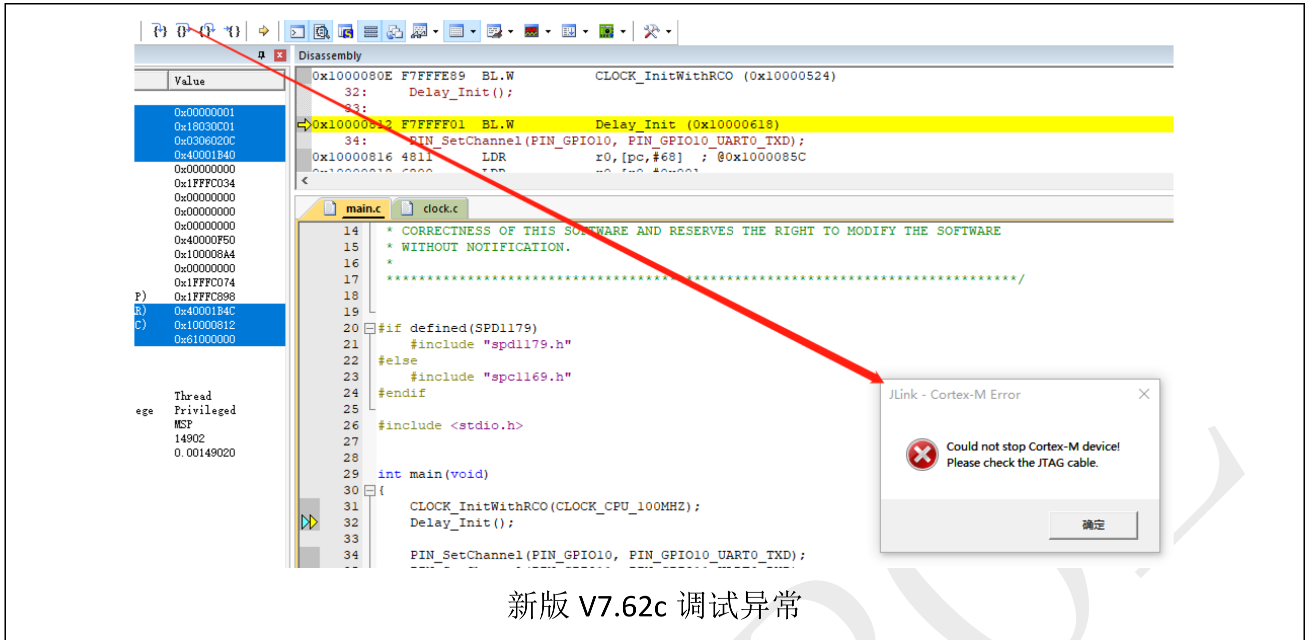
图 3-4：同步更新 JLinkARM.dll



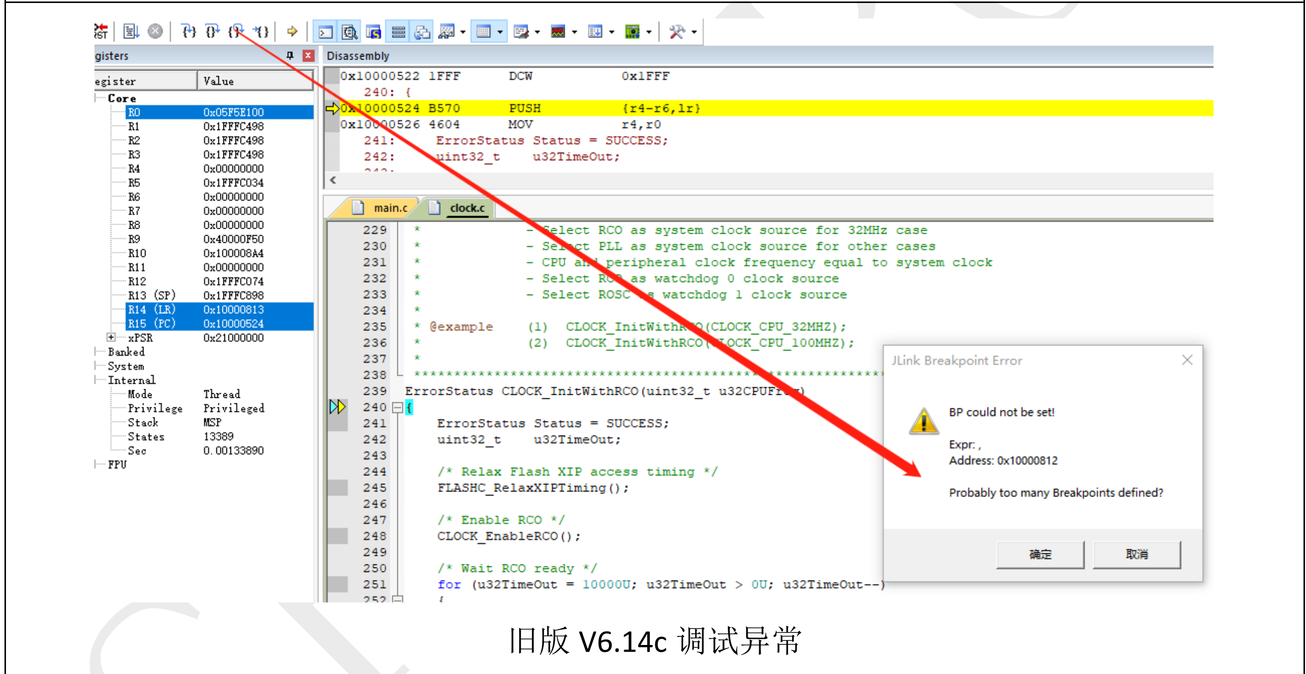
此时其仿真超过 **30s** 也会异常，以 **keil** 为例，如图 3-5 所示。

©2023 旋智电子科技（上海）有限公司

图 3-5：调试异常



新版 V7.62c 调试异常

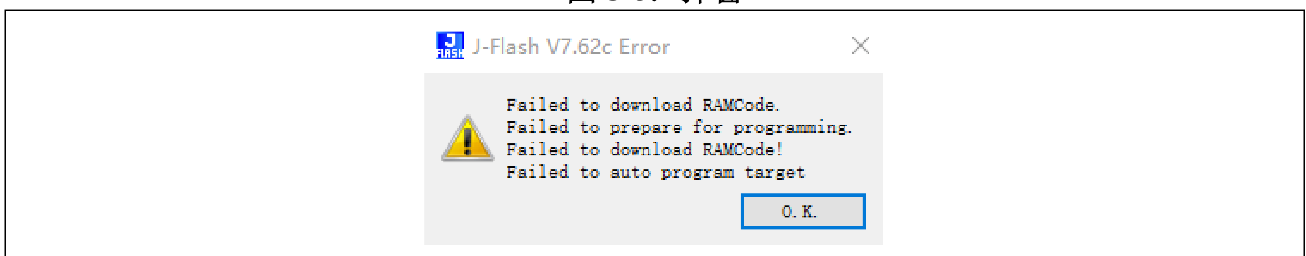


旧版 V6.14c 调试异常

3.3.2 J-Flash 超时下载异常

同时，J-Flash 下载也会异常，5s 时会有警告弹窗，30s 内来不及点下载指令，则下载异常。

图 3-6：弹窗



此时需要重新 Connect，并在 30s 内完成下载。

3.3.3 J-Link 超时下载异常

同时，**J-Link** 指令下载也会异常，**5s** 时会有警告弹窗，**30s** 内来不及输入下载指令，则下载异常。

图 3-7：超过 30s J-Link 指令下载异常

```
PS D:\1_SDK\SPC1169_Platform\trunk\Project_SPC1169\0_Examples\6_1_GTimer_Int\GCC> jlink
SEGGER J-Link Commander V7.62c (Compiled Mar 23 2022 16:41:20)
DLL version V7.62c, compiled Mar 23 2022 16:40:05

Connecting to J-Link via USB...O.K.
Firmware: J-Link ARM-OB STM32 compiled Aug 22 2012 19:52:04
Hardware version: V7.00
S/N: 20090929
License(s): RDI,FlashDL,FlashBP,JFlash,GDBFull
VTref=3.300V
Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: SPC1169_128K
Type '?' for selection dialog
Device>SPC1169_128K
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>S
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>4000 kHz
Device "SPC1169_128K" selected.
Connecting to target via SWD
Found SW-DP with ID 0x2BA01477
DPv0 detected
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x24770011)
```


©2023 旋智电子科技（上海）有限公司

```
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x410FC241. Implementer code: 0x41 (ARM)
Found Cortex-M4 r0p1, Little endian.
FPUnit: 6 code (BP) slots and 2 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
[0][0]: E000E000 CID B105E00D PID 000BB00C SCS-M7
[0][1]: E0001000 CID B105E00D PID 003BB002 DWT
[0][2]: E0002000 CID B105E00D PID 002BB003 FPB
[0][3]: E0000000 CID B105E00D PID 003BB001 ITM
[0][4]: E0040000 CID B105900D PID 000BB9A1 TPIU
Cortex-M4 identified.
J-Link>loadfile ./project.hex
Downloading file [./project.hex]...
Unspecified error -1
```

但 **J-Link** 脚本没问题，因为脚本是一次性输入的，其执行时间可以忽略不计，通常 **hex** 也不会过大，下载时间完全可以控制在 **30s** 内。

```
jlink.exe -device SPC1169_128K -if SWD -speed 10000 -autoconnect 1 -CommandFile .\download.jlink
```

下载完提示成功。

```
Downloading file [.\project.hex]...
J-Link: Flash download: Bank 0 @ 0x10000000: Skipped. Contents already match
O.K.
J-Link>exit


Script processing completed.
```

因为 **keil** 集成了 **J-Link** 下载，其过程通常在 **30s** 内完成，一般也不会有超时异常问题。

SPINTROL

3.4 DEBUG 补充

调试过程如下所示。

图 3-8：Debug 任务界面

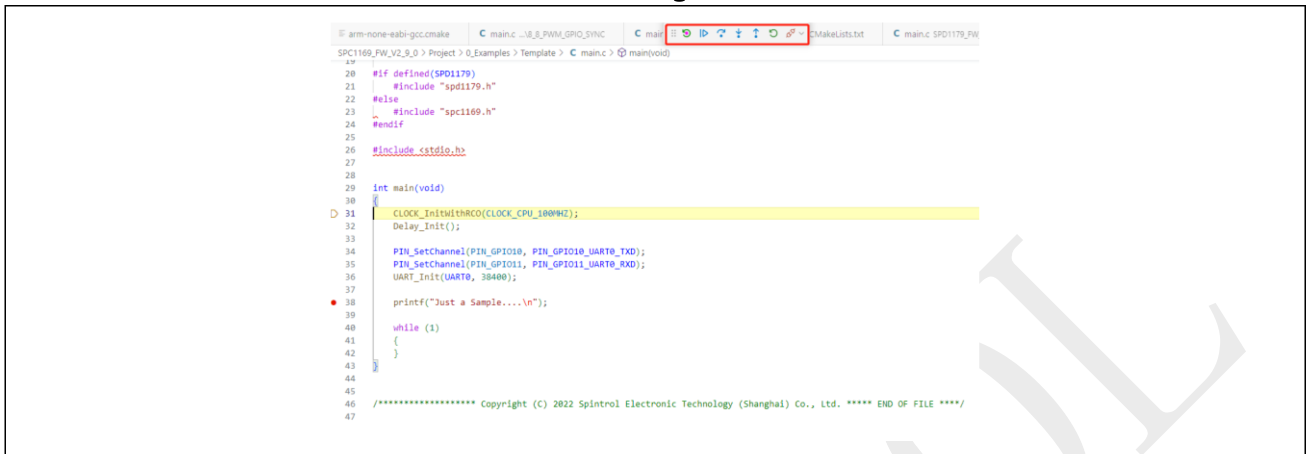


图 3-9：单步调试

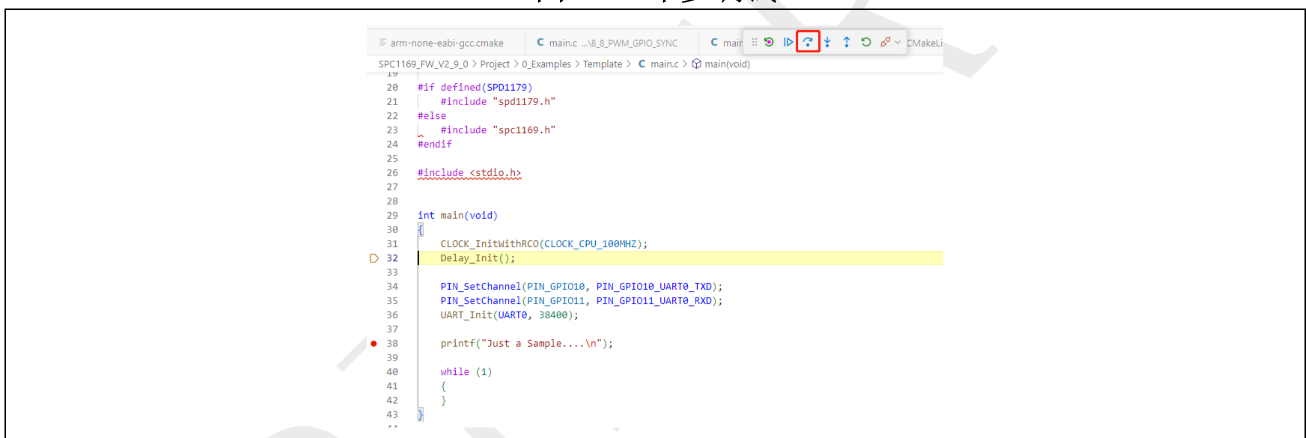
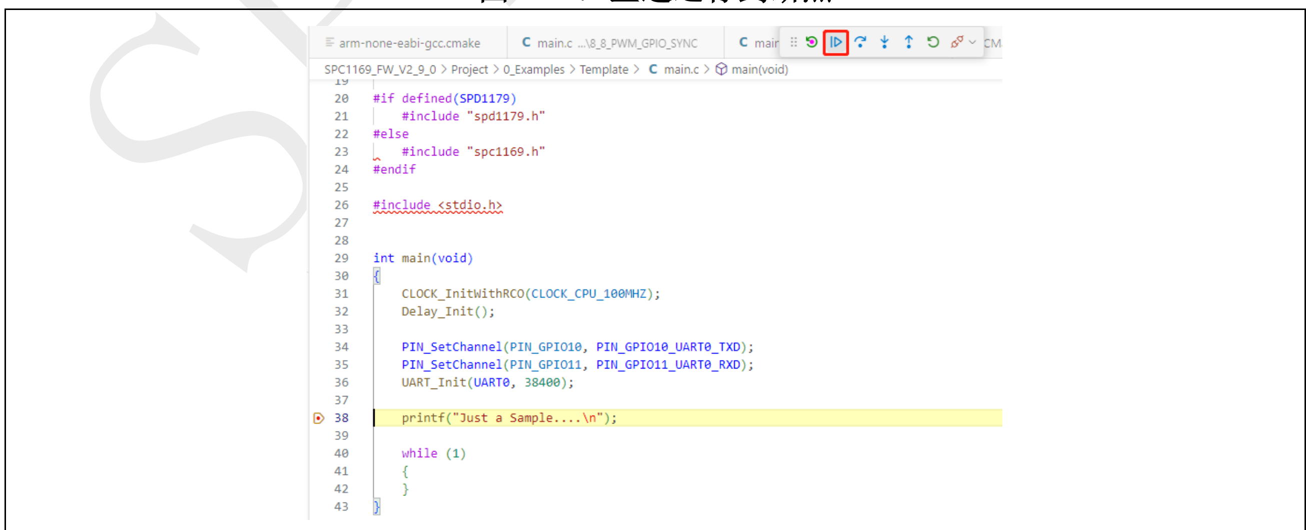


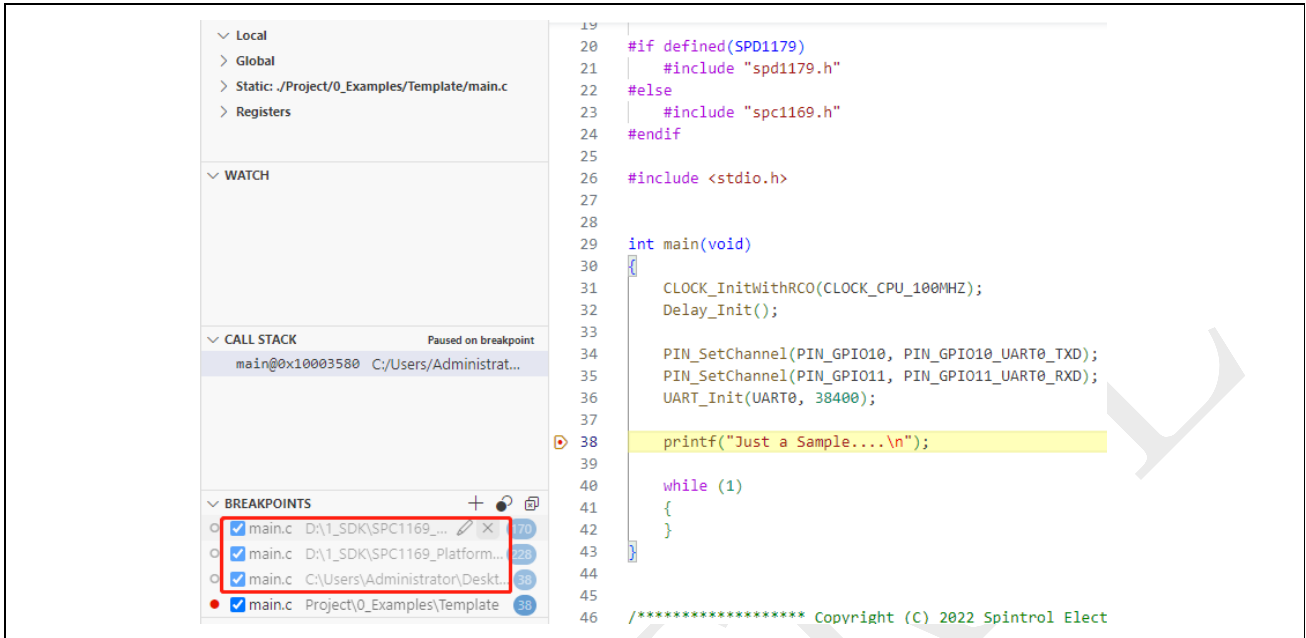
图 3-10：全速运行到断点



©2023 旋智电子科技（上海）有限公司

调试状态下，其他工程的断点会变成灰色，表示不可用。可以手动将其删除，避免视觉干扰。

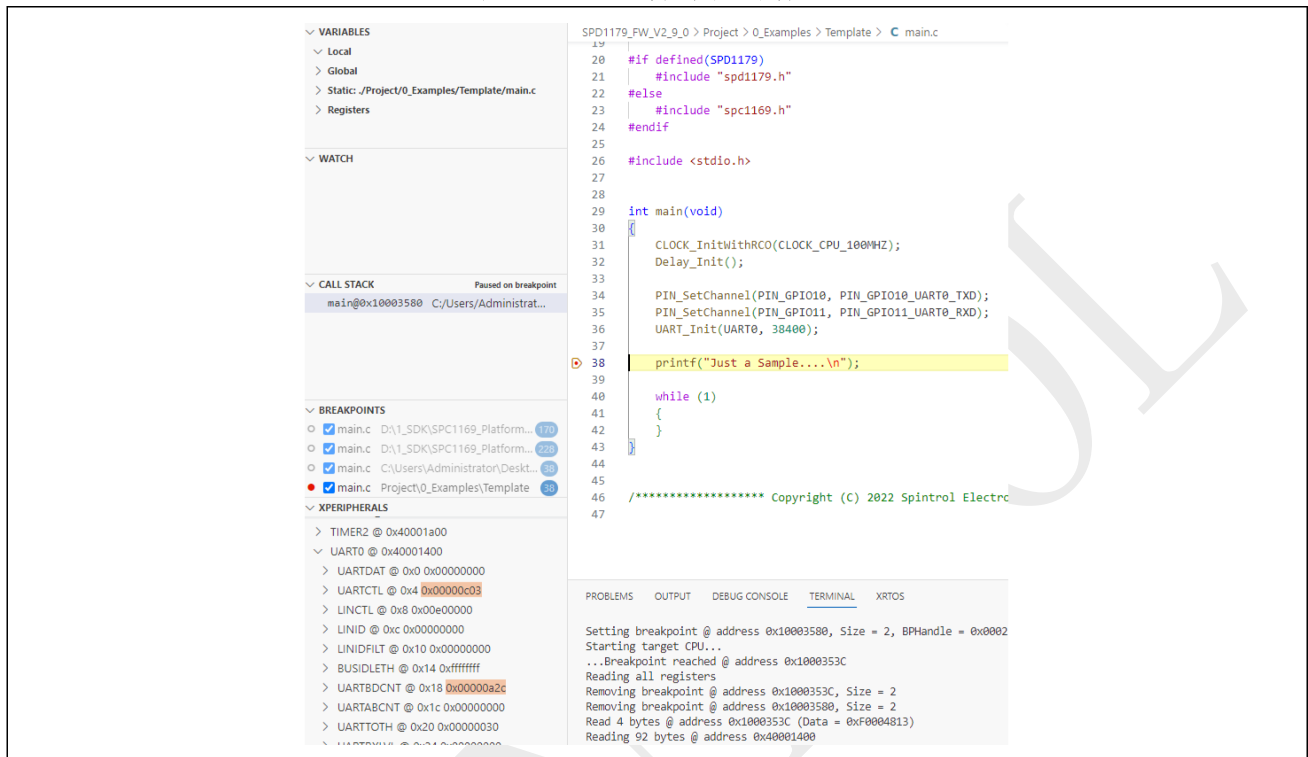
图 3-11：关闭冗余断点



3.5 查看外设寄存器

Svd 文件位置默认在 launch.json 中已经配好，直接在窗口观察即可。**需要事先在窗口点开 UART0，调试过程中在窗口点开 UART0 无效，此时可停止<SHIFT>+F5 后重新运行 F5。**

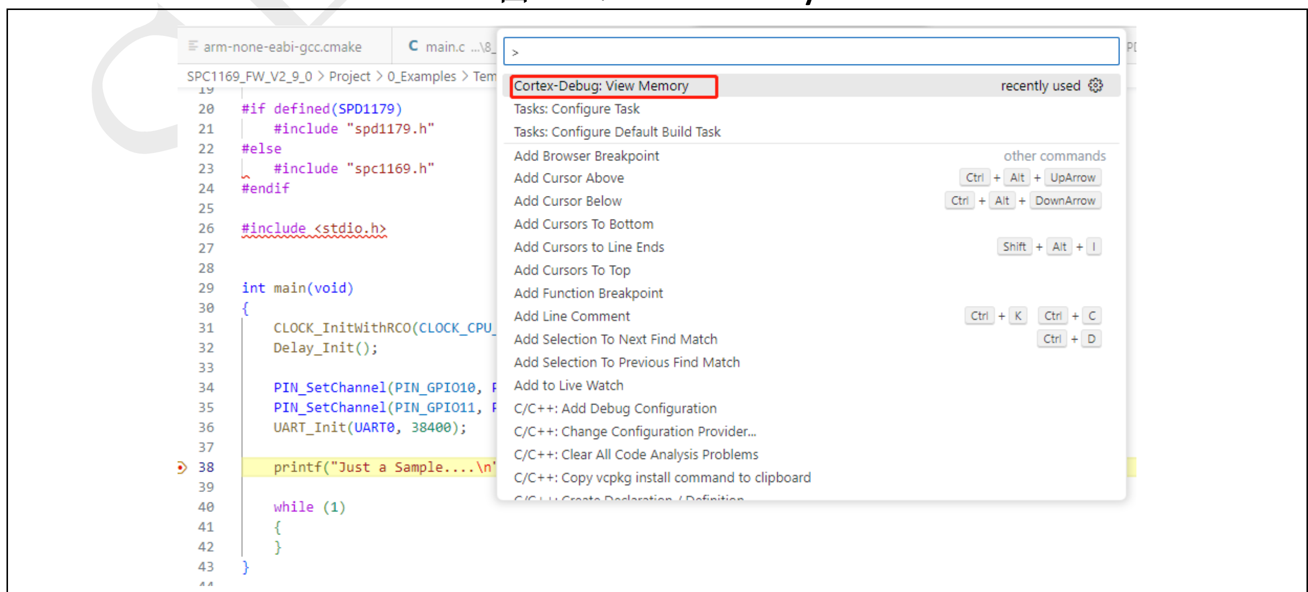
图 3-12：查看外设寄存器



3.6 查看 Memory

按快捷键**<CTRL>+<SHIFT>+P**，在命令框中输入“cortex”，会在下拉框中出现“View Memory”的选项，选择即可，如图 3-13 所示。

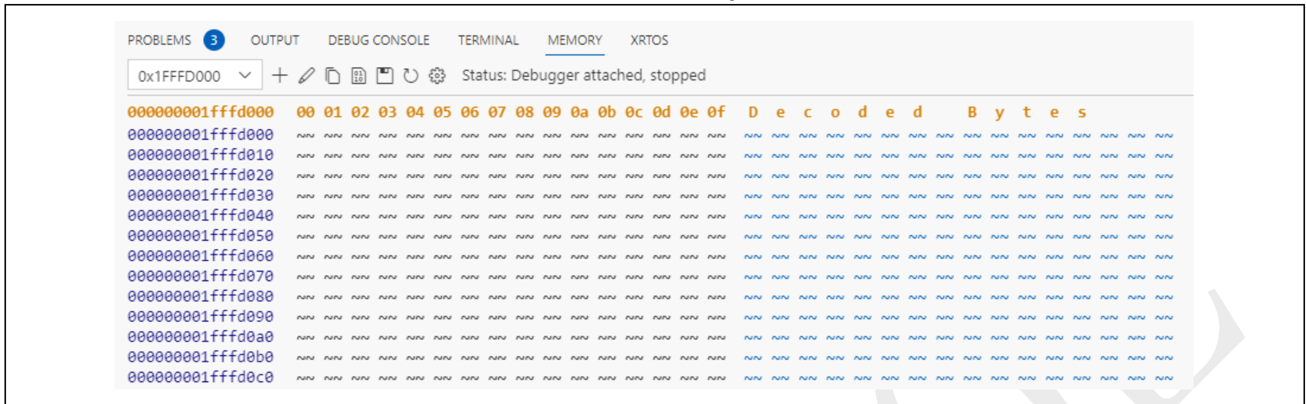
图 3-13：View Memory



©2023 旋智电子科技（上海）有限公司

选择“Debug View Memory”选项之后，需要输入以“0x”开头的地址，回车后，如图 3-14所示。

图 3-14：Memory 数据



SPINTROL

4 代码运行到 RAM

可以通过以下方式将单个函数，或整个文件放到 RAM 中运行。

```
__attribute__ ((section("RAMCODE")))
void MotorDebugBuffer_RecData(struct BUFFER_DEBUG_T *p)
{
  ……这里省略具体内容
}
__attribute__ ((section("RAMCODE")))
void MotorDebugBuffer_Stop(struct BUFFER_DEBUG_T *psDBG_Buffer)
{
  ……这里省略具体内容
}
```

在 ld 文件中，将单个函数，或整个文件分配到 FLASH 中，.S 启动文件负责将其从 FLASH 搬移到 RAM。

```
  /* Initialized data sections goes into RAM, load LMA copy after code */
  .data :
  {
    . = ALIGN(4);
    _sdata = .;        /* create a global symbol at data start */
    *(.data)           /* .data sections */
    *(.data*)          /* .data* sections */
    KEEP (*(RAMCODE))
    KEEP ( *motor_math.o* (.text* .rodata*))
    KEEP ( *f_clarke.o* (.text* .rodata*))
    KEEP ( *f_lpf_1st.o* (.text* .rodata*))
    KEEP ( *f_lpf_2nd.o* (.text* .rodata*))
    KEEP ( *f_park.o* (.text* .rodata*))
    KEEP ( *pmsm_foc_sin_cos_float_tbl.o* (.text* .rodata*))
    KEEP ( *pmsm_foc.o* (.text* .rodata*))
    KEEP ( *pmsm_foc_data.o* (.text* .rodata*))
    KEEP ( *spintrolmotorstateflow.o* (.text* .rodata*))
    . = ALIGN(4);
    _edata = .;        /* define a global symbol at data end */
  } >RAM AT> FLASH
```

 ©2023 旋智电子科技（上海）有限公司