

GigaDevice Semiconductor Inc.

Arm[®] Cortex[®]- M3/M4/M23/M33 32-bit MCU

应用笔记

AN016

目录

目录.....	2
图索引	3
表索引	4
1.简介.....	5
2.开发环境搭建.....	6
2.1. 安装交叉编译工具.....	6
2.2. 安装 C/C++ MinGW 编译器	9
2.3. 安装 Cmake 工具	16
2.4. 安装 Vscode 及插件	18
2.5. 安装 Openocd.....	20
3.CmakeLists 文件编写	21
3.1. 根目录下 CMakeLists.txt 及 Cortex-M3.cmake 文件.....	22
3.2. gd_libs 文件夹中 CMakeLists.txt 文件.....	23
3.3. src 文件夹中 CMakeLists.txt 文件.....	23
4.编译、下载与调试	26
4.1. 编译与下载	26
4.2. 调试.....	30
5.版本历史	33

图索引

图 2-1. GUN Arm Embedded Toolchain 选择下载.....	6
图 2-2. GUN Arm Embedded Toolchain 安装过程 1.....	6
图 2-3. GUN Arm Embedded Toolchain 安装过程 2.....	7
图 2-4. GUN Arm Embedded Toolchain 安装过程 3.....	7
图 2-5. GUN Arm Embedded Toolchain 安装过程 4.....	8
图 2-6. GUN Arm Embedded Toolchain 安装过程 5.....	8
图 2-7. 测试 GUN Arm Embedded Toolchain 是否安装成功	9
图 2-8. MinGW-W64 选择下载安装.....	9
图 2-9. MinGW-W64 安装过程 1	10
图 2-10. MinGW-W64 安装过程 2	10
图 2-11. MinGW-W64 安装过程 3	11
图 2-12. MinGW-W64 安装过程 4	11
图 2-13. MinGW-W64 安装过程 5	12
图 2-14. MinGW-W64 安装过程 6	12
图 2-15. 测试 MinGW-W64 是否安装成功 1.....	13
图 2-16. 复制 MinGW-W64 bin 文件夹路径	13
图 2-17. 添加 MinGW-W64 环境变量 1	14
图 2-18. 添加 MinGW-W64 环境变量 2	14
图 2-19. 添加 MinGW-W64 环境变量 3	15
图 2-20. 测试 MinGW-W64 是否安装成功 2.....	15
图 2-21. 修改 MinGW-W64 mingw32-make 命令为 make 命令	16
图 2-22. Cmake 安装过程 1	16
图 2-23. Cmake 安装过程 2	17
图 2-24. Cmake 安装过程 3	17
图 2-25. 测试 Cmake 是否安装成功	17
图 2-26. Vscode 选择下载安装.....	18
图 2-27. Vscode 安装过程 1	18
图 2-28. Vscode 安装过程 2	19
图 2-29. Vscode 安装过程 3	19
图 2-30. Vscode 安装过程 4	19
图 2-31. Vscode 插件安装搜索界面.....	20
图 3-1. 文件组织结构图	21
图 4-1. Vscode 中运行任务.....	28
图 4-2. Cmake 生成 makefile 的构建过程.....	28
图 4-3. makefile 生成可执行文件的构建过程	28
图 4-4. build 目录中文件组织结构	29
图 4-5. 一键编译和下载过程.....	29
图 4-6. Vscode 下调式界面.....	30
图 4-7. 调试界面中添加断点并运行	31
图 4-8. 查看外设寄存器及变量值.....	31

表索引

表 3-1. 根目录下 CMakeLists.txt 代码	22
表 3-2. 根目录下 Cortex-M3.cmake 代码	22
表 3-3. gd_libs 中 CMakeLists.txt 代码	23
表 3-4. src 中 CMakeLists.txt 代码	24
表 4-1. tasks.json 文件中代码	26
表 4-2. launch.json 文件中代码	27
表 5-1. 版本历史	33

1. 简介

在进行工程编译过程中,大多都会用到 `make` 工具,常用的 `make` 工具包括 `GNU Make`、`qmake`、`MS nmake` 和 `Makepp` 等,这些 `make` 工具在不同的平台上编写 `makefile` 时都遵循着不同的规范、标准和格式,在开发过程中将工程移植到不同平台进行编译将会造成编译失败,使用 `CMAKE` 工具生成 `Makefile` 将可以有效解决上述问题,本应用手册基于 `GD32F10x SDK` 开发包,使用 `CMAKE` 工具搭建编译环境。

2. 开发环境搭建

开发环境介绍主要如下：

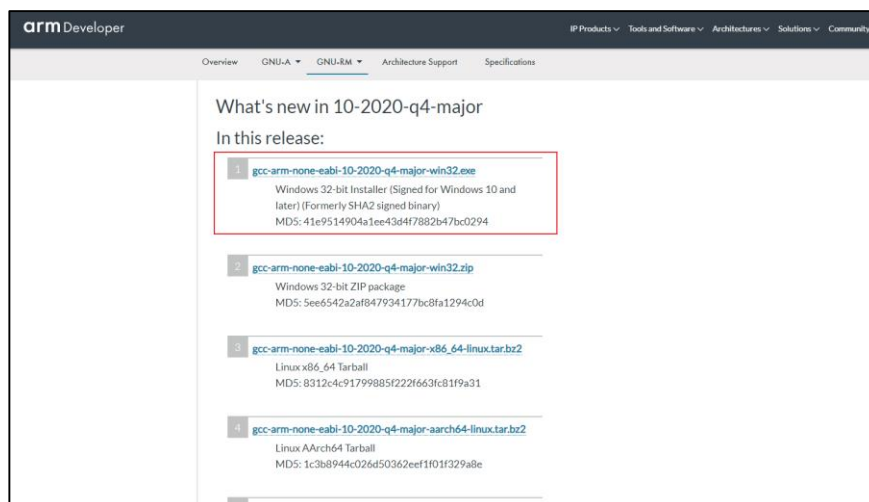
- 硬件开发板：GD32F103C-EVAL-V1.0 开发板
- Cortex-M3: GD32F103C
- 操作系统：Win10-64 位
- 交叉编译工具链：gcc-arm-none-eabi
- C/C++编译器：MinGW
- 开发环境：VSCODE+CMAKE
- 调试下载工具：OPENOCD

2.1. 安装交叉编译工具

GNU Tools for Arm Embedded Processors 下载安装地址：<https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads>

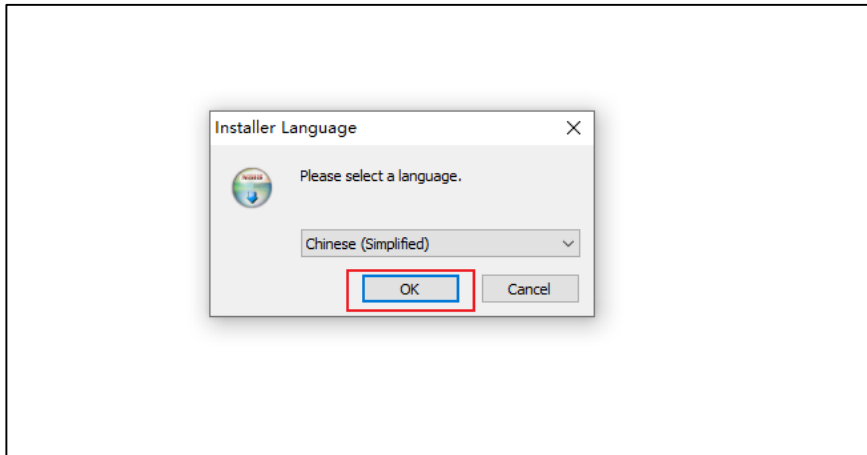
页面中有多个版本的 GUN Arm Embedded Toolchain 供选择下载，在本应用手册中选择下载安装 gcc-arm-none-eabi-10-2020-q4-major-win32.exe，如 [图 2-1. GUN Arm Embedded Toolchain 选择下载](#)所示。

图 2-1. GUN Arm Embedded Toolchain 选择下载



下载完成后双击安装，选择“OK”。

图 2-2. GUN Arm Embedded Toolchain 安装过程 1



点击下一步。

图 2-3. GUN Arm Embedded Toolchain 安装过程 2



点击“我接受”，选择安装路径，选择默认路径即可点击安装。

图 2-4. GUN Arm Embedded Toolchain 安装过程 3

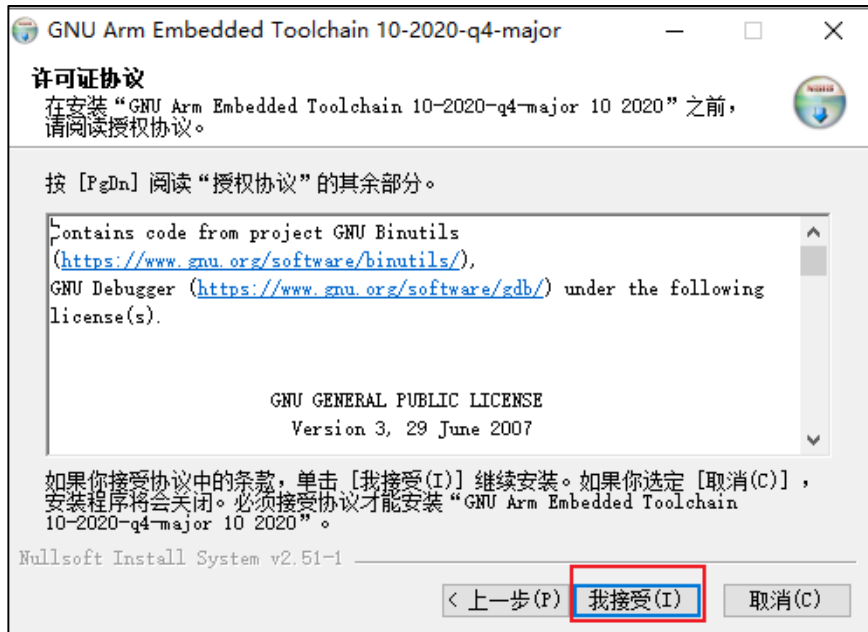
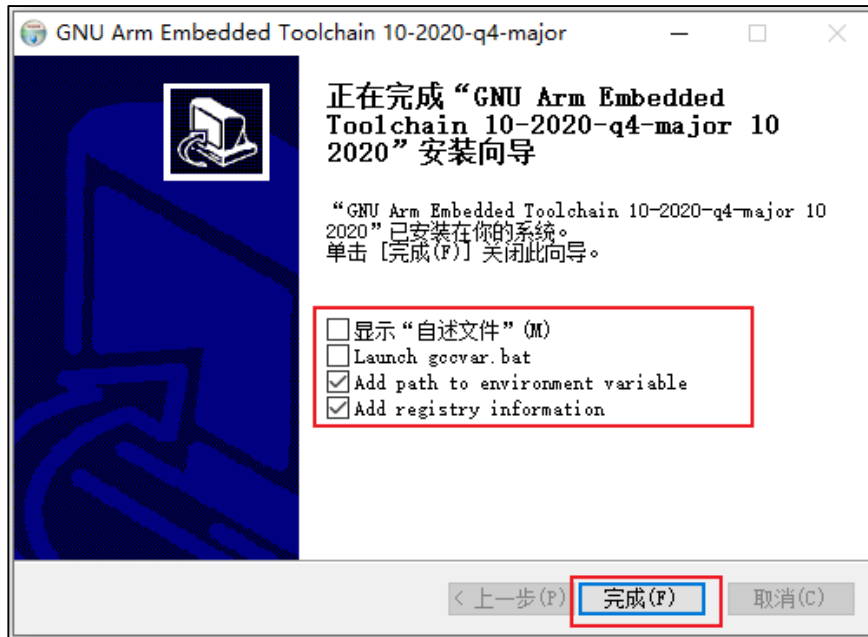


图 2-5. GUN Arm Embedded Toolchain 安装过程 4



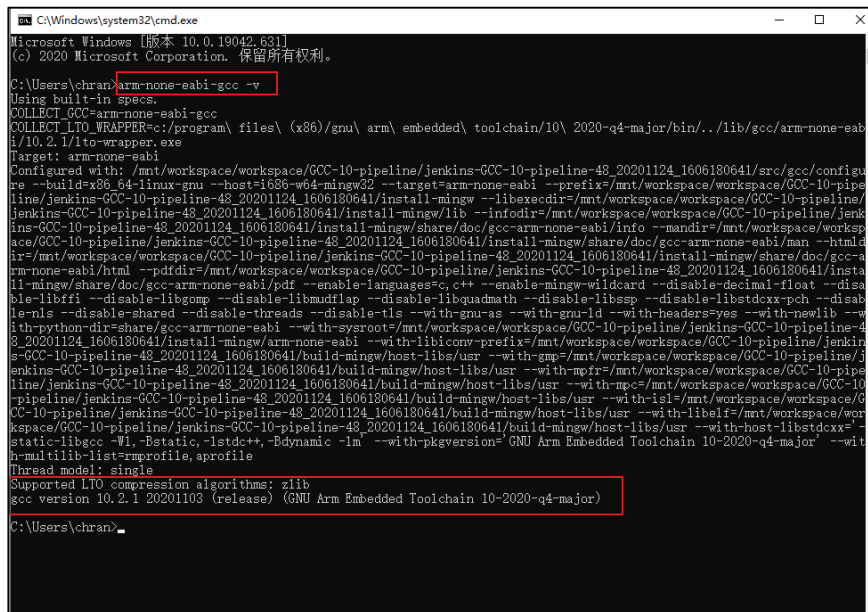
在安装完成时，勾选“Add path to environment variable”选项，点击“完成”。

图 2-6. GUN Arm Embedded Toolchain 安装过程 5



检查是否安装成功，在运行中输入 cmd，点击确定，在命令行中输入 `arm-none-eabi-gcc -v`，返回结果如 [图 2-7. 测试 GUN Arm Embedded Toolchain 是否安装成功](#) 所示代表安装成功。

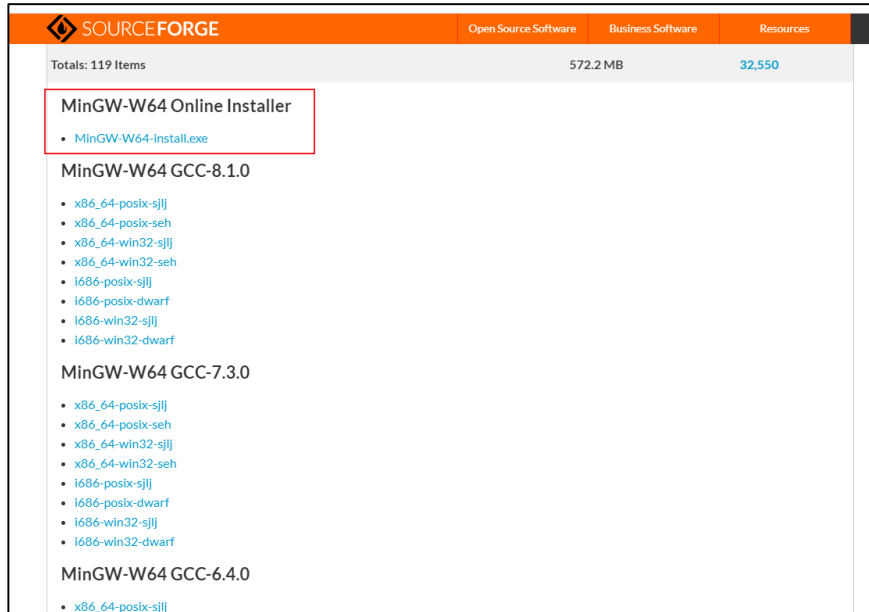
图 2-7. 测试 GUN Arm Embedded Toolchain 是否安装成功



2.2. 安装 C/C++ MinGW 编译器

MinGW 安装包下载地址：<https://sourceforge.net/projects/mingw-w64/files/mingw-w64/mingw-w64-release/>，可以选择在线安装和离线安装两种方式，本手册选择在线安装，下载 MinGW-W64-install.exe。

图 2-8. MinGW-W64 选择下载安装



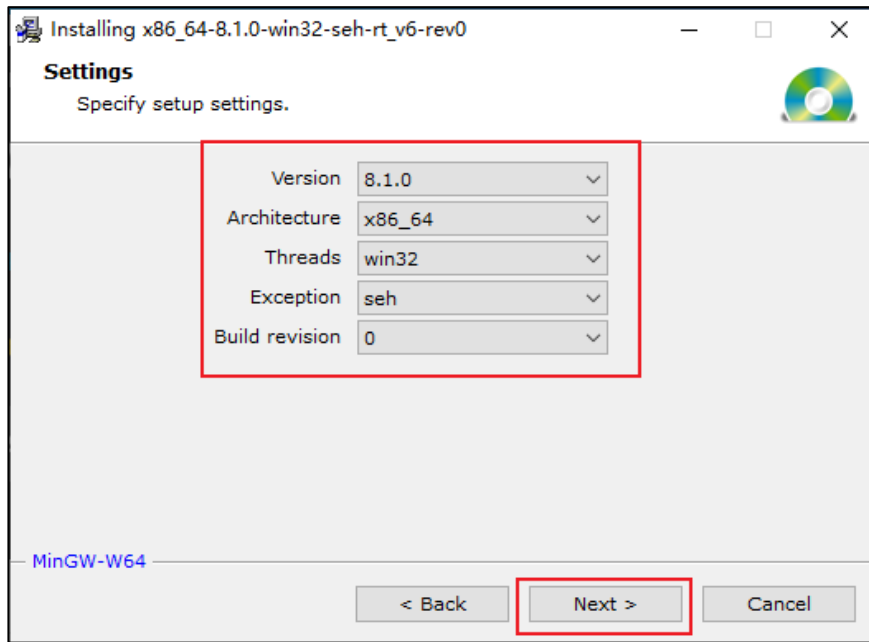
下载“完成”后，双击安装。选择“下一步”。

图 2-9. MinGW-W64 安装过程 1



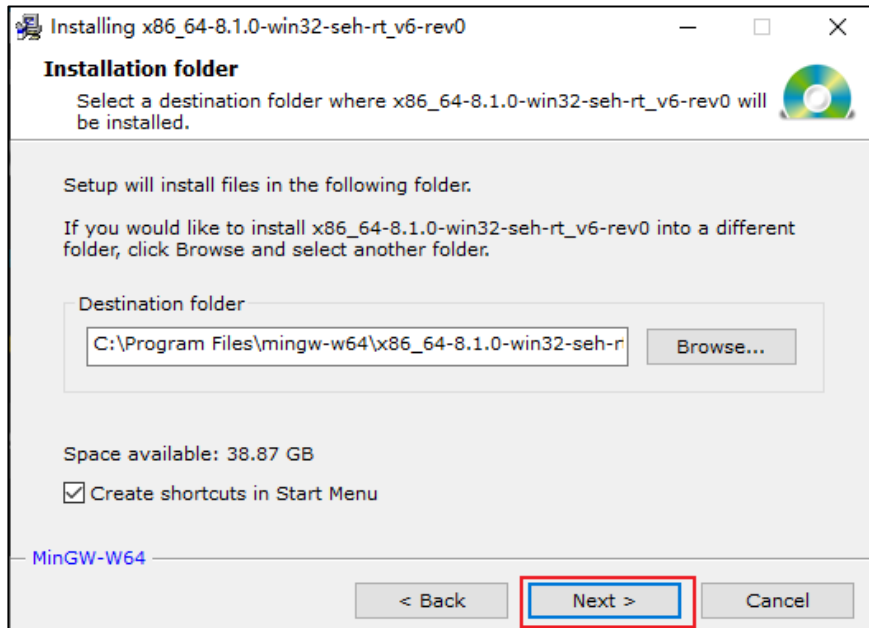
配置选择如下，点击“下一步”。

图 2-10. MinGW-W64 安装过程 2



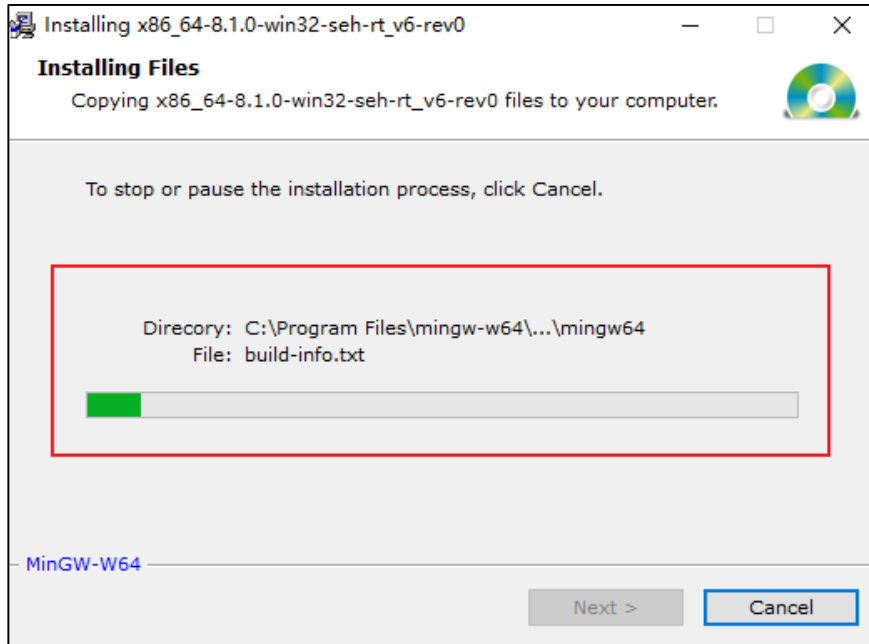
选择安装路径，选择默认路径。

图 2-11. MinGW-W64 安装过程 3



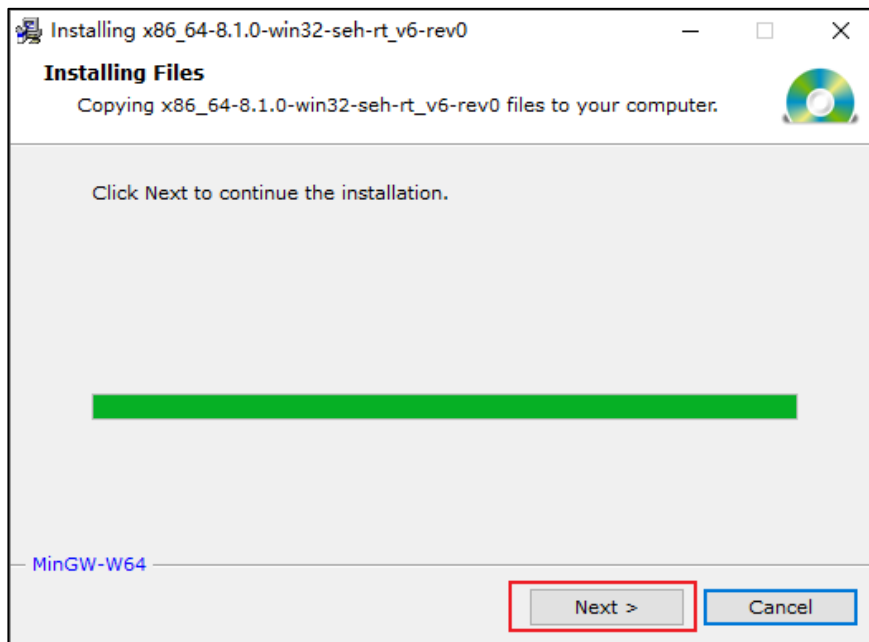
等待下载文件过程。

图 2-12. MinGW-W64 安装过程 4



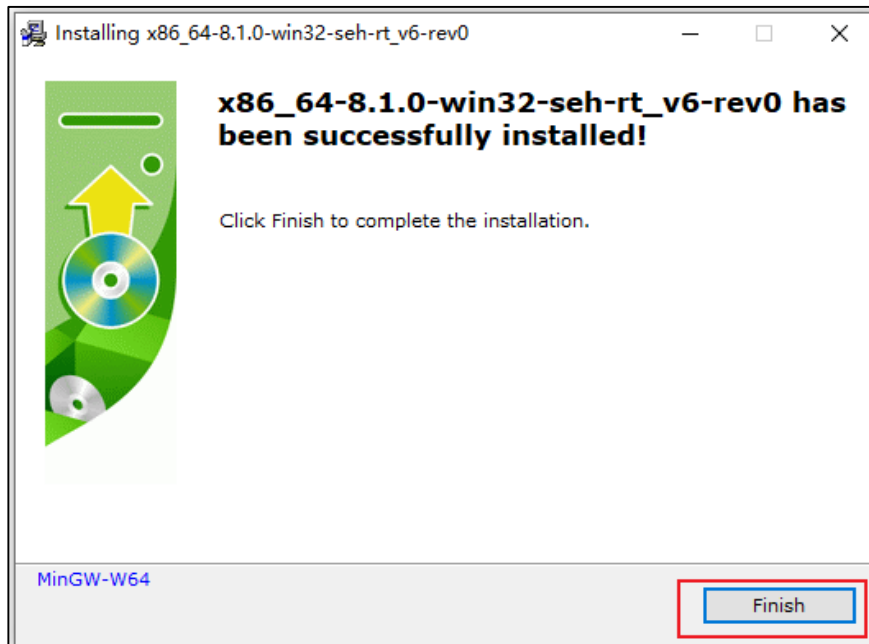
点击“Next”。

图 2-13. MinGW-W64 安装过程 5



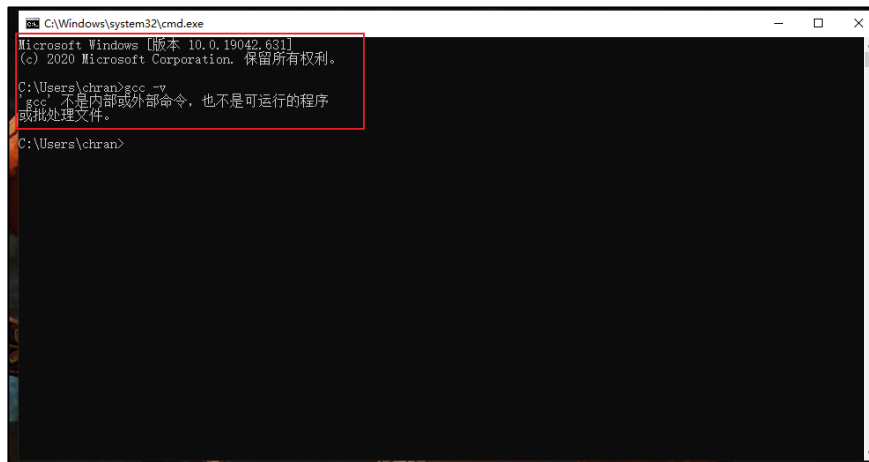
点击“Finish”，完成“安装”。

图 2-14. MinGW-W64 安装过程 6



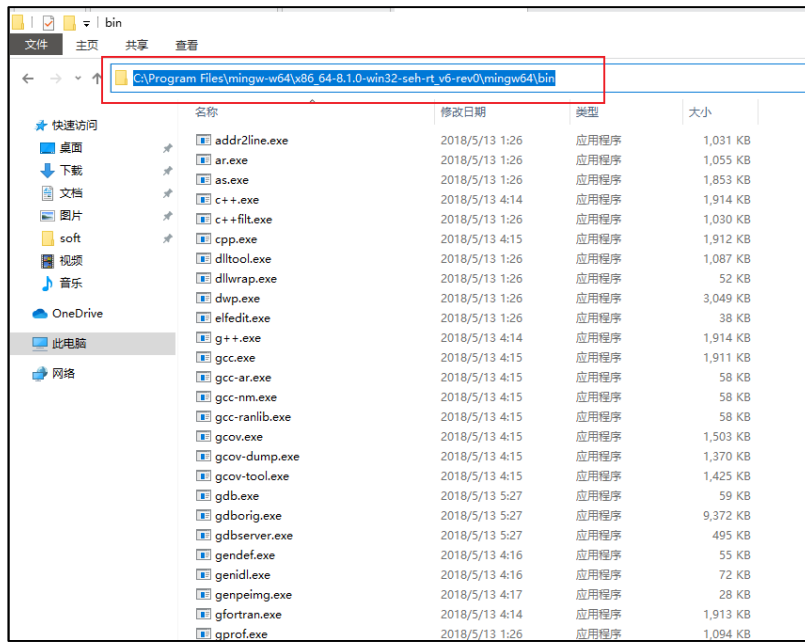
在 cmd 命令行中输入 `gcc -v`，输出不可运行程序，需要将下载的文件加入到系统环境变量中。

图 2-15. 测试 MinGW-W64 是否安装成功 1



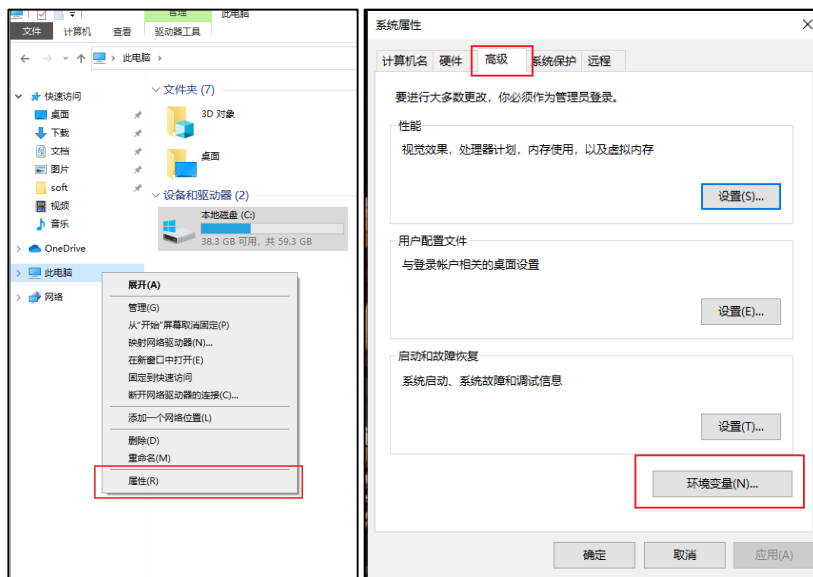
打开 MinGW 安装路径，复制/bin 文件夹下路径，将该路径添加到系统环境变量中。

图 2-16. 复制 MinGW-W64 bin 文件夹路径



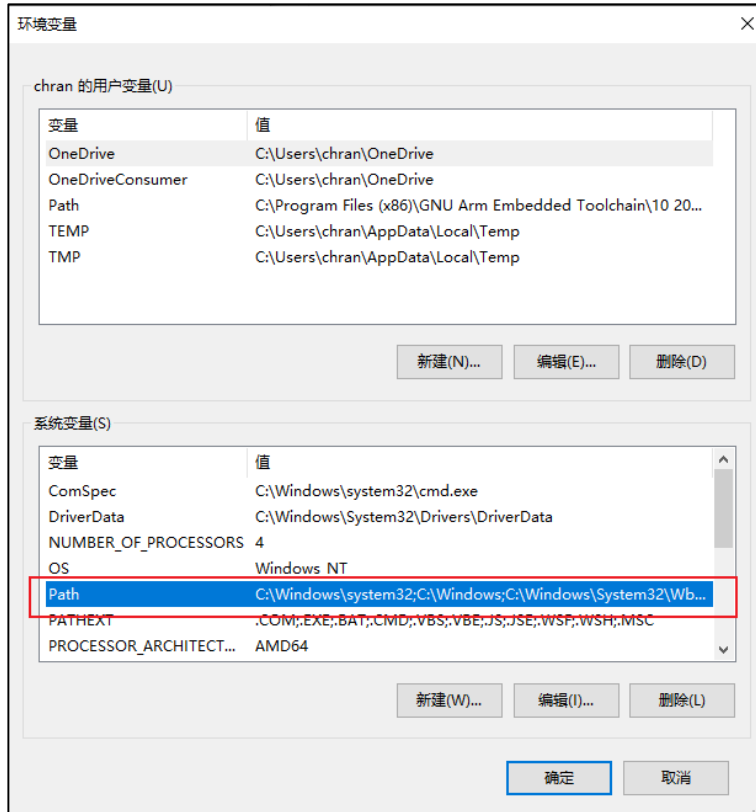
此电脑右键选择属性，点击高级选择环境变量。

图 2-17. 添加 MinGW-W64 环境变量 1



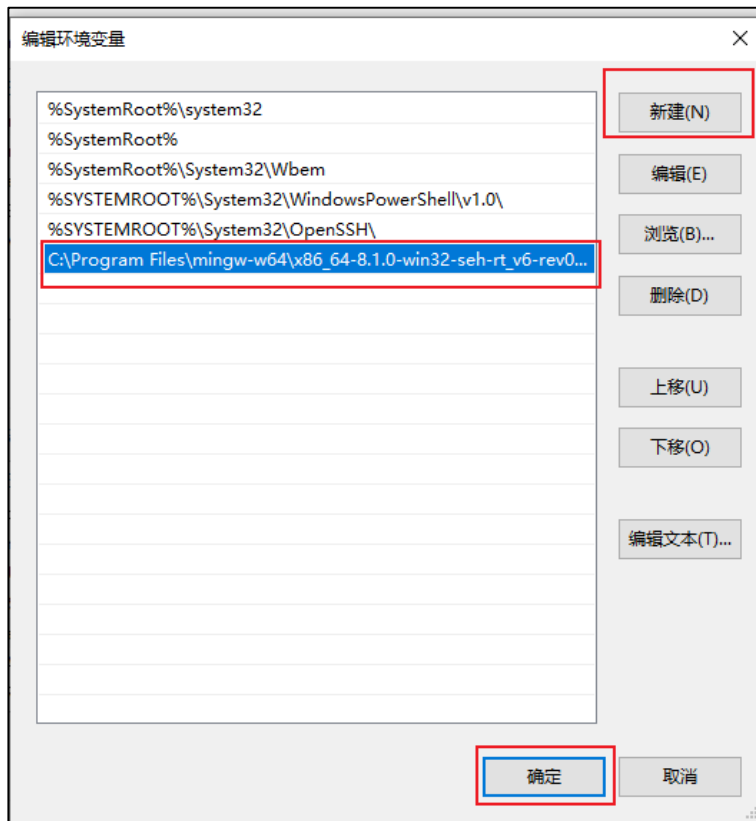
选择系统变量，点击“Path”。

图 2-18. 添加 MinGW-W64 环境变量 2



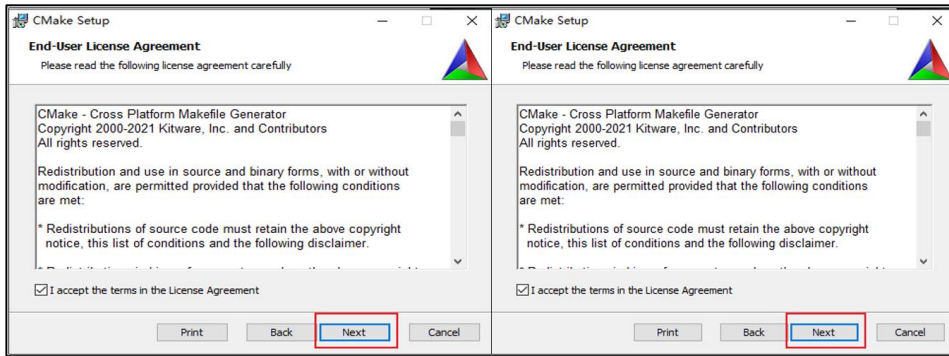
点击“新建”，粘贴复制的路径，点击“确定”。

图 2-19. 添加 MinGW-W64 环境变量 3



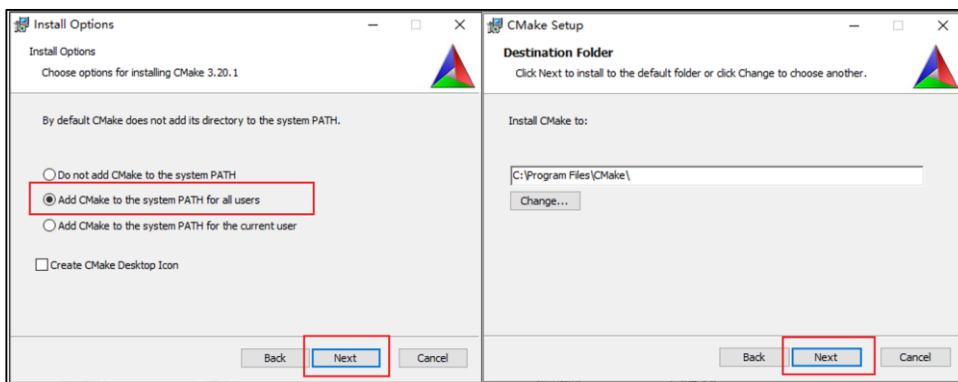
在 cmd 中输入 `gcc -v`，输出 `gcc -v` 版本号，安装成功。

图 2-20. 测试 MinGW-W64 是否安装成功 2



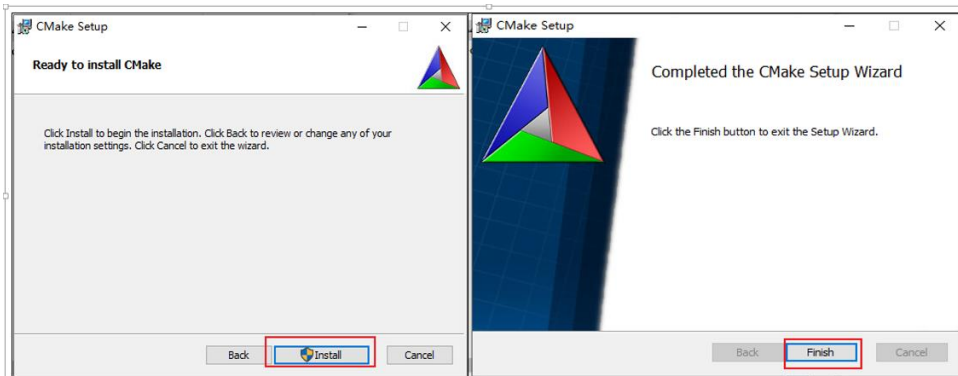
选择“Add Cmake to system PATH for all users”，点击“Next”，选择默认路径，点击“Next”。

图 2-23. Cmake 安装过程 2



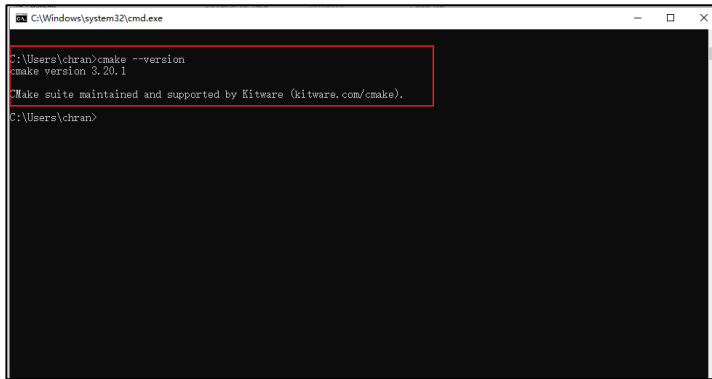
点击“Install”，等待安装完成，点击“Finish”完成安装。

图 2-24. Cmake 安装过程 3



检查是否安装成功，在运行中输入 cmd，点击确定，在命令行中输入 `cmake --version`，返回结果如 [图 2-25. 测试 Cmake 是否安装成功](#) 所示代表安装成功。

图 2-25. 测试 Cmake 是否安装成功

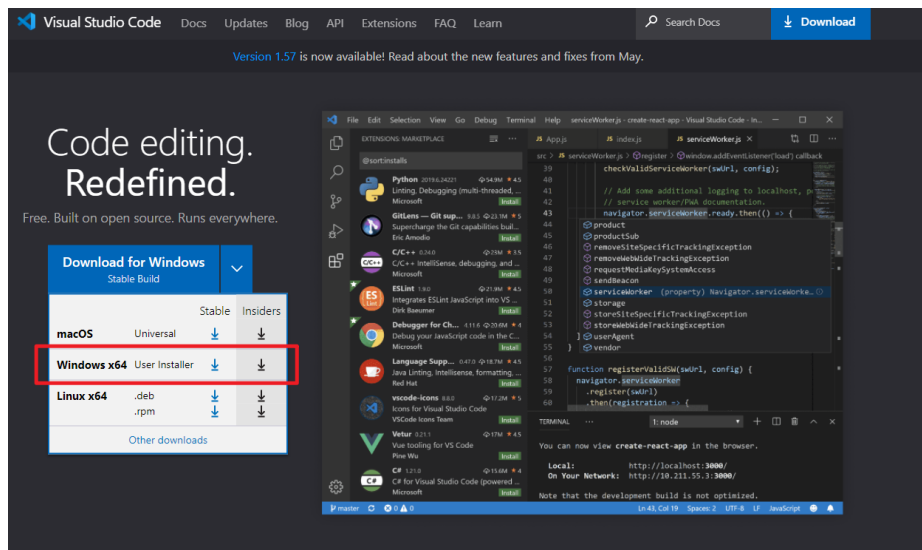


2.4. 安装 Vscode 及插件

Vscode 下载安装地址: <https://code.visualstudio.com/>

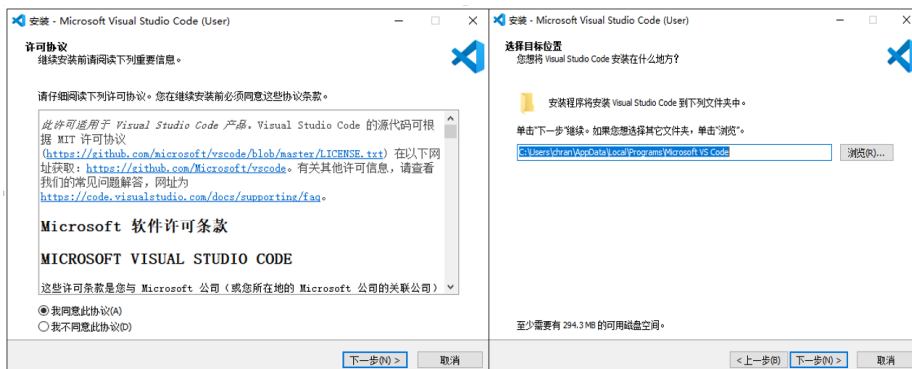
在本应用手册中选择下载安装版本为 1.53.0, 下载安装过程如下所示。

图 2-26. Vscode 选择下载安装



下载完成后, 双击安装。选择“下一步”, 选择安装路径, 点击“下一步”。

图 2-27. Vscode 安装过程 1



一直点击“下一步”，即可完成安装。

图 2-28. Vscode 安装过程 2

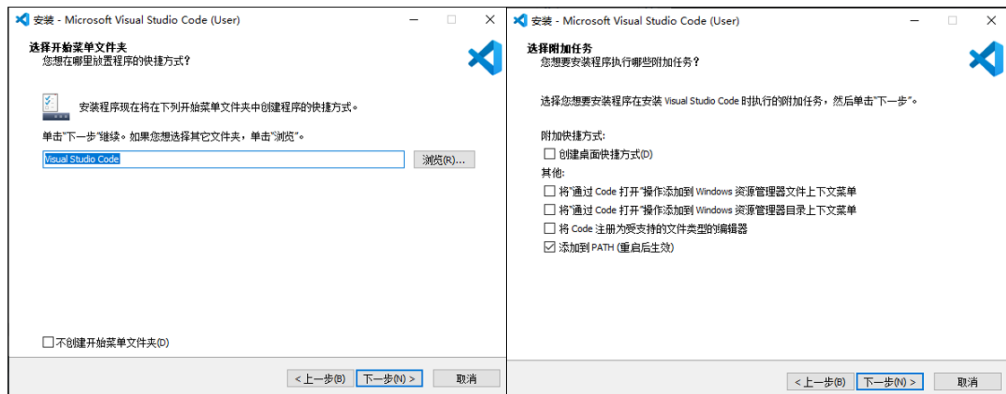


图 2-29. Vscode 安装过程 3

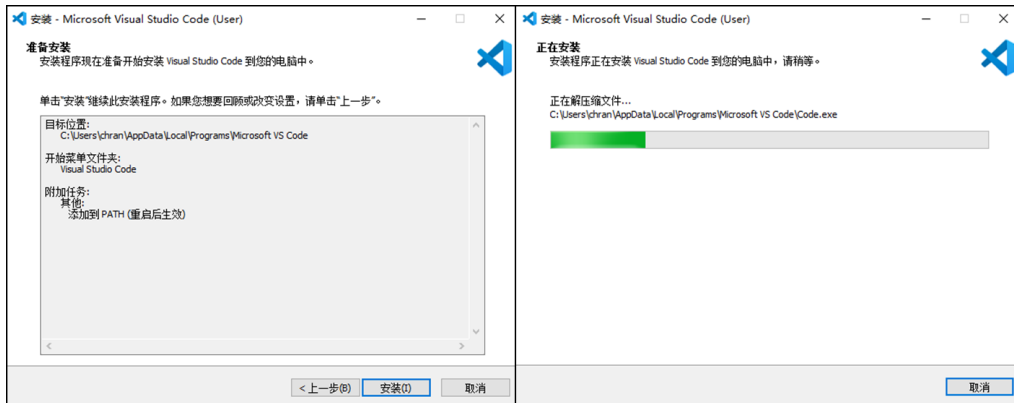
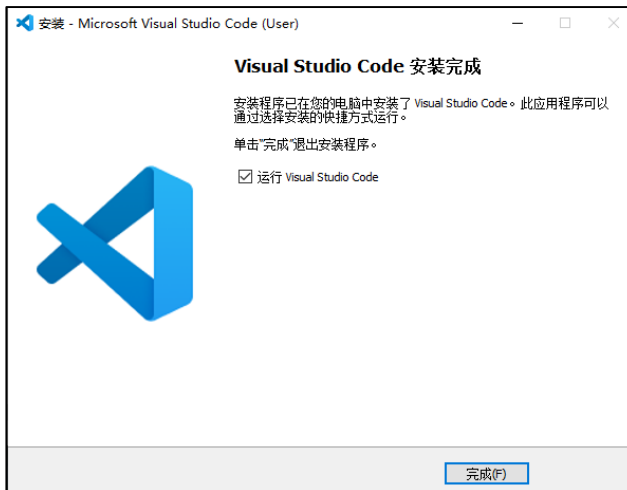


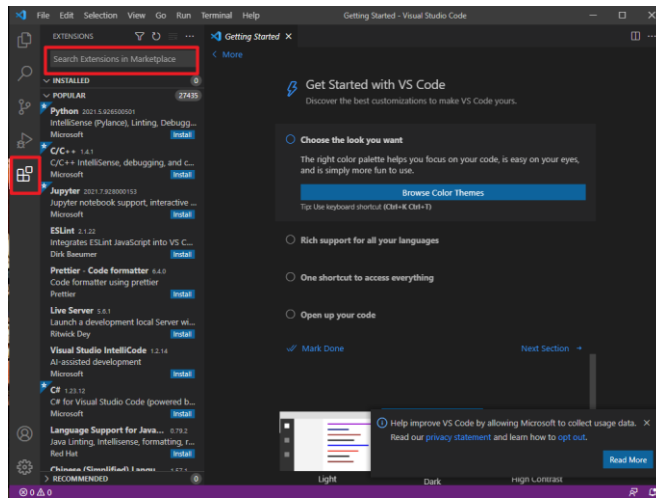
图 2-30. Vscode 安装过程 4



为了满足开发要求，还需要安装部分插件模块，打开安装程序 Vscode,选择左侧插件中心，分别搜索和安装如下插件，搜索到插件点击安装即可。本应用手册安装插件及版本号如下：

- C/C++: V1.2.2
- Cortex-Debug: V0.3.12
- Chinese (simplified) Language for visual studio: V1.52.2

图 2-31. Vscode 插件安装搜索界面



2.5. 安装 Openocd

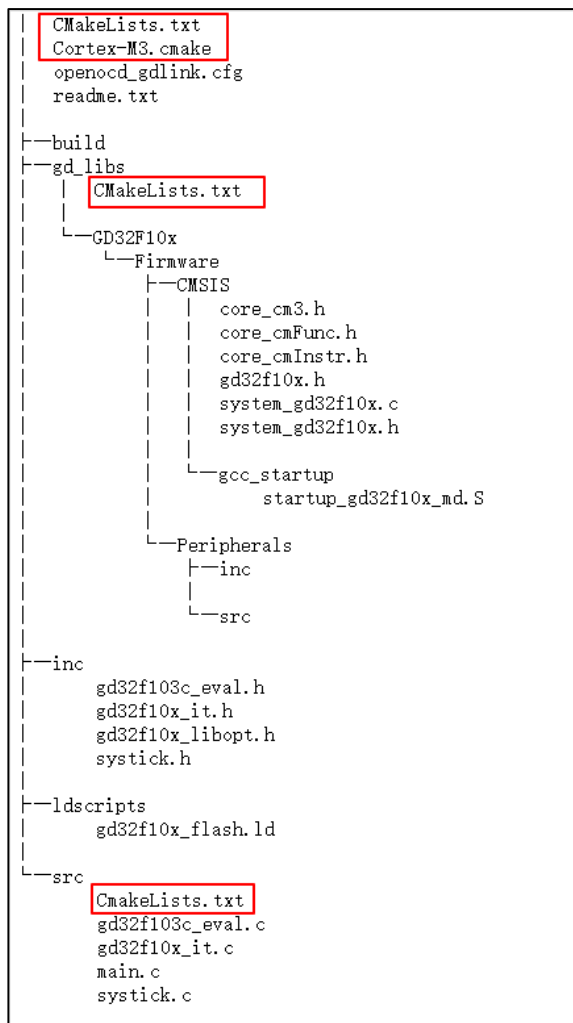
支持 GD32MCU 的 Openocd 地址为: <https://github.com/GigaDevice-Semiconductor/openocd>。

用户根据要求完成 Openocd 的编译后生成可执行文件 openocd.exe,并将该文件加入环境变量中,加入环境变量的方法可参考 [2.2 安装 C/C++ MinGW 编译器](#)。

3. CmakeLists 文件编写

本应用手册文件组织结构如 [图 3-1. 文件组织结构图](#) 所示，下面分别介绍各文件夹中的内容：

图 3-1. 文件组织结构图



在 `build` 文件夹中存放编译过程中所产生的中间文件、库文件和最终的可执行文件；

`gd_libs` 文件夹中存放 GD 库文件，主要包括外设库文件、启动文件以及部分头文件；

`inc` 文件夹中存放用户编写代码的头文件以及开发板相关的头文件；

`ldscripts` 文件夹中存放链接脚本文件；

`src` 文件夹中存放用户编写代码的.c 文件以及开发板相关的.c 文件。

其中在根目录、`gd_libs` 和 `src` 文件夹下分别包含 `CMakeLists.txt` 文件，以及在根目录下包含 `Cortex-M3.cmake` 文件，下面分别介绍各文件中 `CMakeLists.txt` 的内容。

3.1. 根目录下 CMakeLists.txt 及 Cortex-M3.cmake 文件

根目录下 CMakeLists.txt 文件的内容如[表 3-1. 根目录下 CMakeLists.txt 代码](#)。

表 3-1. 根目录下 CMakeLists.txt 代码

```
#设置 CMake 最低支持版本
cmake_minimum_required(VERSION 3.17)
SET(PRJ_NAME "GD32F10x")
#定义工程名称
project(${PRJ_NAME})
#引用 Cortex-M3.cmake
include(Cortex-M3.cmake)
#添加目录 gd_libs 和 src
add_subdirectory(gd_libs)
add_subdirectory(src)
```

Cortex-M3.cmake 文件的内容如[表 3-2. 根目录下 Cortex-M3.cmake 代码](#)，用户可通过修改该文件中的参数来修改编译选项，以及添加相关宏定义：

表 3-2. 根目录下 Cortex-M3.cmake 代码

```
#设置 CMake 最低支持版本
cmake_minimum_required(VERSION 3.17)
#Cmake 交叉编译配置
SET(CMAKE_SYSTEM_NAME Generic)
#设置支持 ASM
ENABLE_LANGUAGE(ASM)
#Debug 模式
SET(CMAKE_BUILD_TYPE "Debug")
#Release 模式
SET(CMAKE_BUILD_TYPE "Release")
#设置 C 编译工具
SET(CMAKE_C_COMPILER arm-none-eabi-gcc)
#ELF 转换为 bin 和 hex 文件工具
SET(CMAKE_OBJCOPY arm-none-eabi-objcopy)
#查看文件大小工具
SET(CMAKE_SIZE arm-none-eabi-size)
#设置浮点选项
SET(MCU_FLAGS "-mcpu=cortex-m3 -mfloat-abi=softfp -mfpu=fpv4-sp-d16")
#设置警告相关信息
SET(CMAKE_C_FLAGS "${MCU_FLAGS} -w -Wno-unknown-pragmas")
#设置调试选项
SET(CMAKE_C_FLAGS_DEBUG "-O0 -g2 -ggdb")
SET(CMAKE_C_FLAGS_RELEASE "-O3")
#添加宏定义
ADD_DEFINITIONS(
    -DGD32F10X_HD
```

```
-DUSE_STDPERIPH_DRIVER
)
```

3.2. gd_libs 文件夹中 CMakeLists.txt 文件

gd_libs 中 CMakeLists.txt 文件的内容如[表 3-3. gd_libs 中 CMakeLists.txt 代码](#)所示，该文件主要实现将 GD 外设固件库文件以及 CMSIS 相关文件和启动文件生成 gd32_lib 库，并指定生成位置：

表 3-3. gd_libs 中 CMakeLists.txt 代码

```
#设置相关路径变量
SET(START_UP_DIR
${PROJECT_SOURCE_DIR}/gd_libs/GD32F10x/Firmware/CMSIS/gcc_startup)
SET(CORE_SUPPORT_DIR ${PROJECT_SOURCE_DIR}/gd_libs/GD32F10x/Firmware/CMSIS)
SET(PERIPHERALS_DIR ${PROJECT_SOURCE_DIR}/gd_libs/GD32F10x/Firmware/Peripherals)
#添加头文件搜索路径
include_directories(
    ${CORE_SUPPORT_DIR}
    ${PERIPHERALS_DIR}/inc
    ${PROJECT_SOURCE_DIR}/inc
)
#设置启动文件变量
SET(START_UP_ASM startup_gd32f10x_md.S)
#设置启动文件 C 属性
set_property(SOURCE ${START_UP_DIR}/${START_UP_ASM} PROPERTY LANGUAGE C)
# GLOB 选项将会为所有匹配查询表达式的文件生成一个文件 list，并将该 list 存储进变量所定义的
STD_LIB, SRC_CORE 里
file(GLOB STD_LIB ${PERIPHERALS_DIR}/src/*.c)
file(GLOB SRC_CORE ${CORE_SUPPORT_DIR}/*.c)
#生成库目标 gd32_lib
add_library(gd32_lib
    ${STD_LIB}
    ${SRC_CORE}
    ${START_UP_DIR}/${START_UP_ASM}
)
#设置库输出的名称
set_target_properties(gd32_lib PROPERTIES OUTPUT_NAME "gd32_lib")
#设置库文件的默认输出路径
SET(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/build/lib)
```

3.3. src 文件夹中 CMakeLists.txt 文件

src 文件夹中 CMakeLists.txt 文件的内容如[表 3-4. src 中 CMakeLists.txt 代码](#)所示，该文件

主要实现将用户编写的.c 文件和 gd32_lib 库文件进行链接编译，生成可执行文件，并指定生成可执行文件的位置。

表 3-4. src 中 CMakeLists.txt 代码

```

#该路径下所有.c 文件定义为 SRC_LIST
aux_source_directory(. SRC_LIST)
#添加头文件搜索路径
include_directories(
    ${PROJECT_SOURCE_DIR}/gd_libs/GD32F10x/Firmware/CMSIS
    ${PROJECT_SOURCE_DIR}/gd_libs/GD32F10x/Firmware/Peripherals/inc
    ${PROJECT_SOURCE_DIR}/inc
)
#添加非标准的共享库搜索路径
link_directories(${PROJECT_SOURCE_DIR}/build/lib)
#设置链接文件相关路径和变量
SET(FLASH_LD_DIR ${PROJECT_SOURCE_DIR}/ldscripts)
SET(FLASH_LD_FILE gd32f10x_flash.ld)
SET(LINKER_SCRIPT ${FLASH_LD_DIR}/${FLASH_LD_FILE})
#设置链接选项
SET(CMAKE_EXE_LINKER_FLAGS
"--specs=nano.specs          -specs=nosys.specs          -T${LINKER_SCRIPT}          -Wl,-
Map=${PROJECT_BINARY_DIR}/${PRJ_NAME}.map,--cref -Wl,--gc-sections")
#生成目标文件
add_executable(${PRJ_NAME}.elf ${SRC_LIST})
#把目标文件与库文件进行链接
target_link_libraries(${PRJ_NAME}.elf gd32_lib)
#设置可执行文件输出路径
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/build/bin)
#设置 ELF 转换路径
SET(ELF_FILE ${PROJECT_SOURCE_DIR}/build/bin/${PRJ_NAME}.elf)
SET(HEX_FILE ${PROJECT_SOURCE_DIR}/build/bin/${PRJ_NAME}.hex)
SET(BIN_FILE ${PROJECT_SOURCE_DIR}/build/bin/${PRJ_NAME}.bin)
#添加自定义命令实现 ELF 转换 hex 和 bin 文件
add_custom_command(TARGET "${PRJ_NAME}.elf" POST_BUILD
    COMMAND ${CMAKE_OBJCOPY} -Obinary ${ELF_FILE} ${BIN_FILE}
    COMMAND ${CMAKE_OBJCOPY} -Oihex  ${ELF_FILE} ${HEX_FILE}
    COMMENT "Building ${PRJ_NAME}.bin and ${PRJ_NAME}.hex"
    COMMAND          ${CMAKE_COMMAND}          -E          copy          ${HEX_FILE}
"${CMAKE_CURRENT_BINARY_DIR}/${PRJ_NAME}.hex"
    COMMAND          ${CMAKE_COMMAND}          -E          copy          ${BIN_FILE}
"${CMAKE_CURRENT_BINARY_DIR}/${PRJ_NAME}.bin"

    COMMAND ${CMAKE_SIZE} --format=berkeley ${ELF_FILE} ${HEX_FILE}
    COMMENT "Invoking: Cross ARM GNU Print Size"

```


)

4. 编译、下载与调试

4.1. 编译与下载

使用 `vscode` 打开该工程目录，在该目录下会产生 `.vscode` 文件夹，在文件夹中新建 `tasks.json` 文件 `launch.json` 文件，`tasks.json` 文件主要通过不用通过命令行而通过按钮的方式来实现编译和下载功能，`launch.json` 文件主要是调试配置文件，具体文件内容和注释分别如 [表 4-1. tasks.json 文件中代码](#)和 [表 4-2. launch.json 文件中代码](#)所示。

表 4-1. `tasks.json` 文件中代码

```
{
  "version": "2.0.0",
  //指定命令执行所在路径
  "options": {
    "cwd": "${workspaceRoot}/build"
  }
  "tasks": [
    {
      //执行 cmake 命令，生成 makefile 文件
      "type": "shell",
      "label": "cmake",
      "command": "cmake",
      "args": [
        "-G",
        "MinGW Makefiles",
        ".."
      ]
    },
    {
      //执行 make 命令，生成可执行文件
      "label": "make",
      "type": "shell",
      "command": "make",
      "args": [],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "dependsOn": [
        "cmake"
      ],
      "problemMatcher": []
    }
  ],
}
```

```

{
    //执行 openocd 命令，下载可执行文件到目标 MCU
    "type": "shell",
    "label": "Build & Updatde",
    "command": "openocd",
    "args": [
        "-f",
        //配置文件绝对路径
        "E:/Work_Code/10.cmake/Cmake/Code/Example-
windows/gd32f103C_example/openocd_gdlink_gd32f10x.cfg",
        "-c",
        //编译生成的可执行文件绝对路径
        "program E:/Work_Code/10.cmake/Cmake/Code/Example-
windows/gd32f103C_example/build/bin/GD32F10x.elf verify reset exit"
    ],
    "group": "build",
    "dependsOn": "make"
}
]
}

```

表 4-2. launch.json 文件中代码

```

{
    // 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": [
        {
            //指定命令执行所在路径
            "cwd": "${workspaceRoot}",
            //可执行文件所在路径
            "executable": "build/bin/GD32F10x.elf",
            "name": "Debug Microcontroller",
            "request": "launch",
            "type": "cortex-debug",
            //svd 文件所在路径
            "svdFile": "./GD32F10x_MD.svd",
            //openocd 命令
            "serverType": "openocd",
            //openocd 配置文件所在相对路径
            "configFiles": [
                "./openocd_gdlink_gd32f10x.cfg "
            ]
        }
    ]
}

```

```

    }

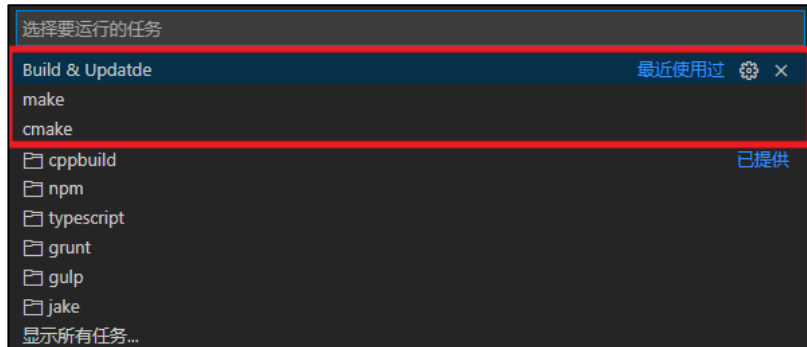
]

}

```

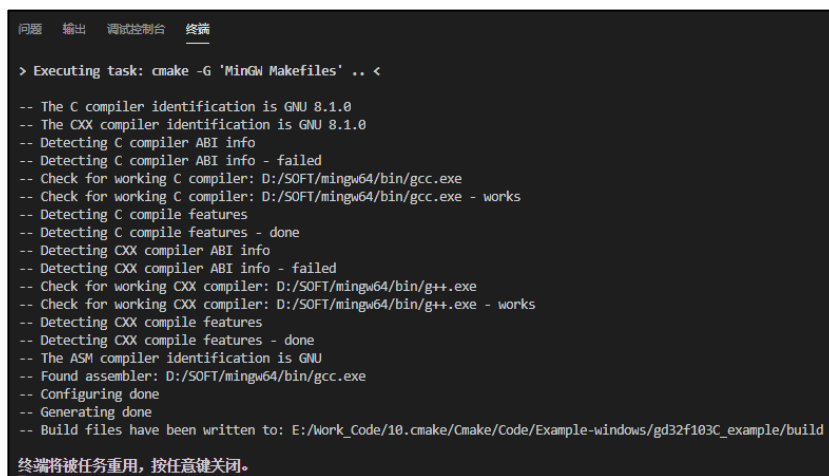
在 VScode 中点击“终端->运行任务”，可以看到[图 4-1. Vscode 中运行任务](#)选项。

图 4-1. Vscode 中运行任务



选择 `cmake` 选项，执行 `cmake` 命令并生成 `makefile` 文件，从 `vscode` 终端打印信息可以看到 `makefile` 构建过程。

图 4-2. Cmake 生成 makefile 的构建过程



选择 `make` 选项，执行 `make` 命令，实现编译功能，并生成可执行文件 `.elf`、`.bin` 和 `.hex` 文件，从 `vscode` 终端打印信息可以看到可执行文件生成过程。同时从 `build` 目录下可以看出生成的 `elf`、`hex` 和 `bin` 文件存放在 `bin` 目录下。

图 4-3. makefile 生成可执行文件的构建过程

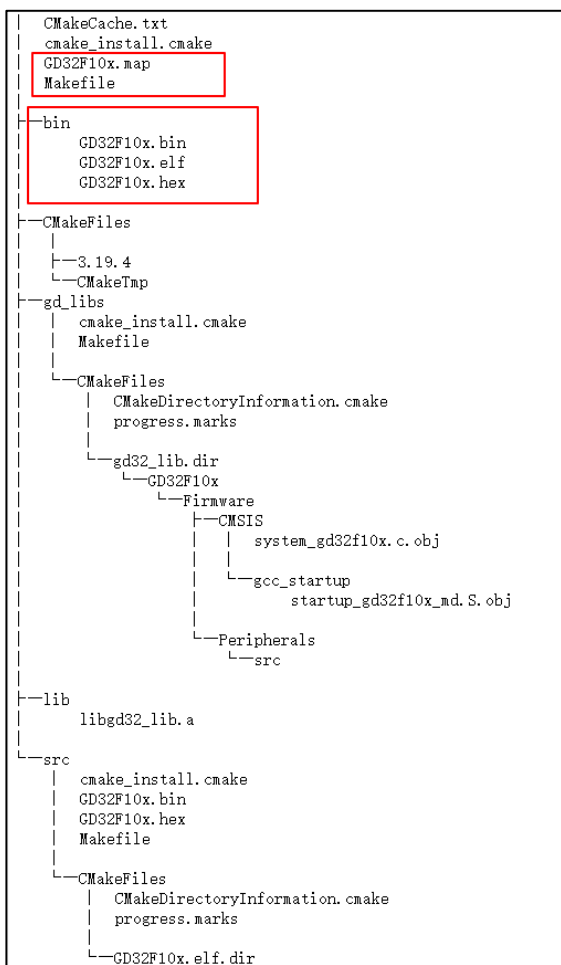
```

> Executing task: make <

Scanning dependencies of target gd32_lib
[ 3%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_adc.c.obj
[ 6%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_bkp.c.obj
[ 9%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_can.c.obj
[ 12%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_crc.c.obj
[ 16%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_dac.c.obj
[ 19%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_dbg.c.obj
[ 22%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_dma.c.obj
[ 25%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_enet.c.obj
[ 29%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_exti.c.obj
[ 32%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_exti.c.obj
[ 35%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_exti.c.obj
[ 38%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_exti.c.obj
[ 41%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_gpio.c.obj
[ 45%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_i2c.c.obj
[ 48%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_misc.c.obj
[ 51%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_pmu.c.obj
[ 54%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_rcu.c.obj
[ 58%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_rtc.c.obj
[ 61%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_sdio.c.obj
[ 64%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_spi.c.obj
[ 67%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_timer.c.obj
[ 70%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_usart.c.obj
[ 74%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_wdgt.c.obj
[ 77%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/CMSIS/system_gd32f10x.c.obj
[ 80%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/CMSIS/gcc_startup/startup_gd32f10x_md.S.obj
[ 83%] Linking C static library ../lib/libgd32_lib.a
[ 83%] Built target gd32_lib
Scanning dependencies of target GD32F10x.elf
[ 87%] Building C object src/CMakeFiles/GD32F10x.elf.dir/gd32f103c_eval.c.obj
[ 90%] Building C object src/CMakeFiles/GD32F10x.elf.dir/gd32f10x_it.c.obj
[ 93%] Building C object src/CMakeFiles/GD32F10x.elf.dir/main.c.obj
[ 96%] Building C object src/CMakeFiles/GD32F10x.elf.dir/systick.c.obj
[100%] Linking C executable ../bin/GD32F10x.elf
Invoking: Cross ARM GNU Print Size
text    data    bss     dec     hex filename
18552   140    2080    12772   31e4 E:/Work_Code/10_cmake/cmake/Code/Example-windows/gd32f103c_example/build/bin/GD32F10x.elf
0      10692   0      10692   29c4 E:/Work_Code/10_cmake/cmake/Code/Example-windows/gd32f103c_example/build/bin/GD32F10x.hex
[100%] Built target GD32F10x.elf

```

图 4-4. build 目录中文件组织结构



选项 Build&Updata, 实现一键 cmake 构建、编译以及 openocd 下载功能, 从 vscode 终端打印信息可以看到构建、编译和程序下载过程。

图 4-5. 一键编译和下载过程

```

> Executing task: cmake -G 'MinGW Makefiles' .. <
-- Configuring done
-- Generating done
-- Build files have been written to: E:/Work_Code/10.cmake/Cmake/Code/Example-windows/g

终端将被任务重用, 按任意键关闭。

> Executing task: make <

[ 83%] Built target gd32_lib
[100%] Built target GD32F10x.elf

终端将被任务重用, 按任意键关闭。

> Executing task: openocd -f E:/Work_Code/10.cmake/Cmake/Code/Example-windows/gd32f103C
n/GD32F10x.elf verify reset exit' <

xPack OpenOCD, x86_64 Open On-Chip Debugger 0.10.0+dev-dirty (2021-03-04-06:52)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
cortex_m_reset_config sysresetreq

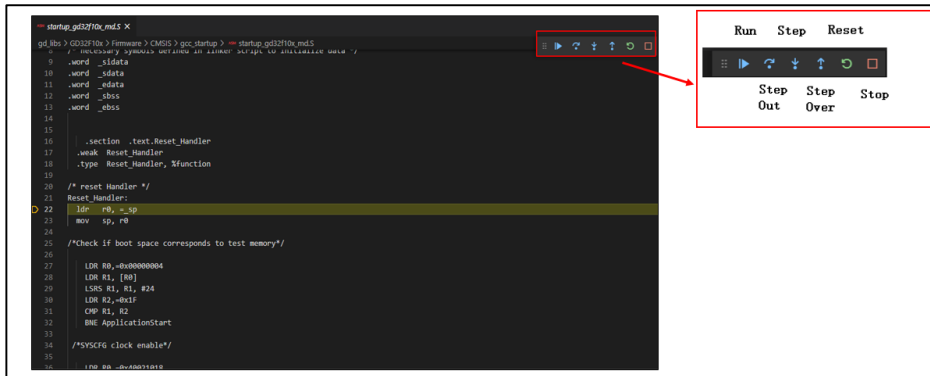
Info : CMSIS-DAP: SWD Supported
Info : CMSIS-DAP: JTAG Supported
Info : CMSIS-DAP: FW Version = 2.0.0
Info : CMSIS-DAP: Interface Initialised (SWD)
Info : SWCLK/TCK = 1 SWDIO/TMS = 1 TDI = 1 TDO = 1 nTRST = 0 nRESET = 1
Info : CMSIS-DAP: Interface ready
Info : clock speed 1000 kHz
Info : SWD DPIDR 0x1ba01477
Info : gd32f10x.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : starting gdb server for gd32f10x.cpu on 3333
Info : Listening on port 3333 for gdb connections
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08002890 msp: 0x2000c000
** Programming Started **
Info : device id = 0x13090414
Info : flash size = 256kbytes
Info : bank->num_sectors = 128
Info : bank->size = 262144
Info : GD32: Flash erasing...
Info : GD32: Flash erase ... sector erase(0 to 5)
Info : GD32: Flash erase ...finished
Info : GD32: Flash writing...
Info : bank=0000000028a0da0 buffer=000000002905580 offset=00000000 count=00003000
Info : GD32: Flash write ... not words to write, padding with 0xff
Info : GD32: Flash write ... words to be programmed = 0x0000c01
** Programming Finished **
** Verify Started **
** Verified OK **
** Resetting Target **
shutdown command invoked

```

4.2. 调试

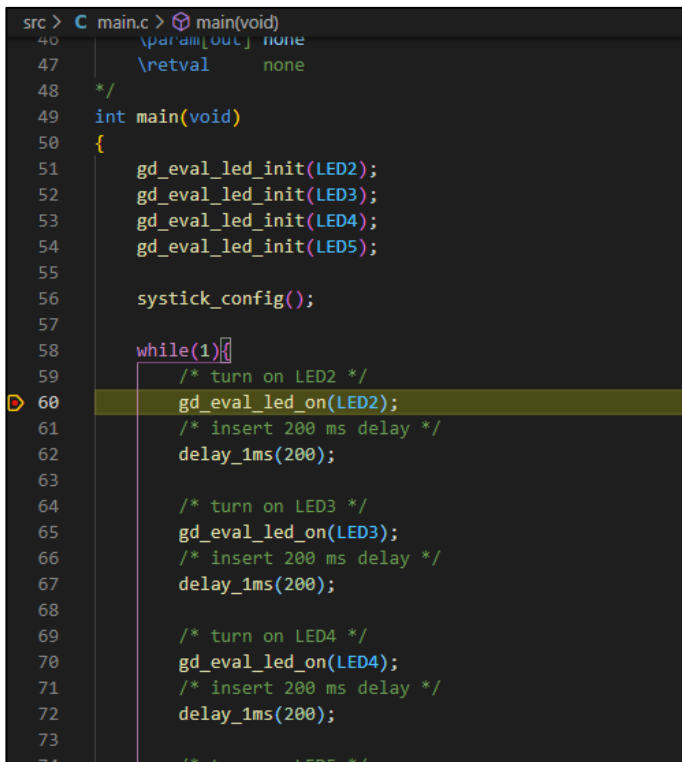
在 VScode 中点击“运行->启动调试”，进入调试界面如[图 4-6. Vscode 下调试界面](#)所示，可通过右上角对程序调试进行运行、单步、复位以及终止操作。

图 4-6. Vscode 下调试界面



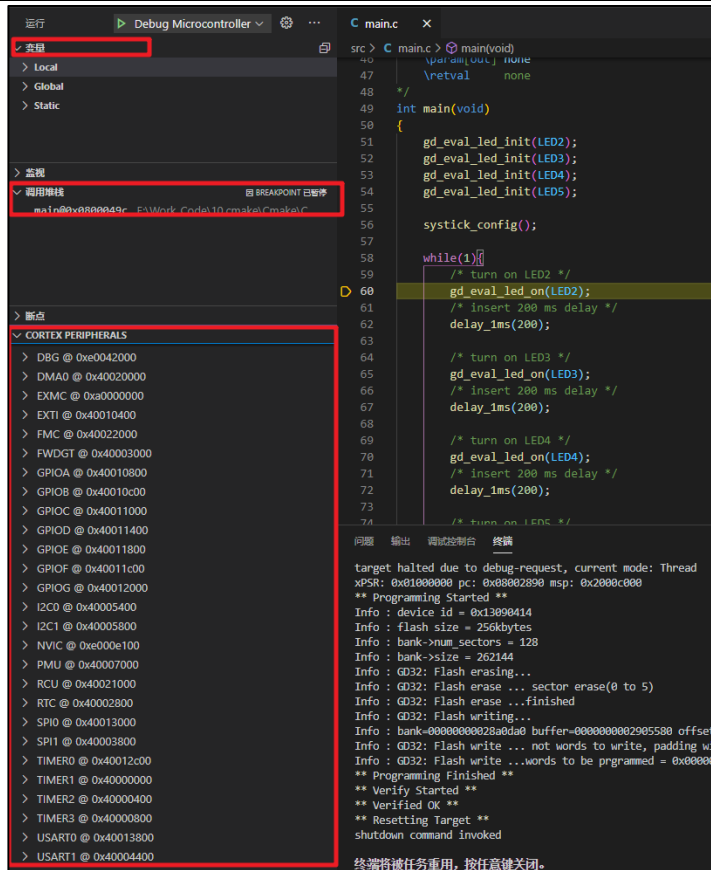
可以通过在代码左侧加入断点，并执行 Run 运行到断点处。

图 4-7. 调试界面中添加断点并运行



在左边界面可以查看外设寄存器值，可添加变量查看等调试操作。

图 4-8. 查看外设寄存器及变量值



The screenshot displays an IDE window with the following components:

- Variable Window (变量):** Shows local, global, and static variables.
- Watch Window (监视):** Shows the current execution point at line 60: `gd_eval_led_on(LED2);`.
- Breakpoint Window (调用断点):** Shows a breakpoint set at `main@0x0000049c`.
- Peripheral Register List (断点):** Lists various peripherals such as DBG, DMA0, EXMC, EXTI, FMC, FWDGT, GPIOA through GPIOI, I2C0 through I2C1, NVIC, PMU, RCU, RTC, SPI0 through SPI1, TIMER0 through TIMER3, USART0, and USART1.
- Code Editor:** Shows the C source code for `main.c`, including initialization of LEDs and a loop that turns on LEDs sequentially with 200ms delays.
- Terminal:** Displays the following output:


```
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x000002890 msp: 0x2000c000
** Programming Started **
Info : device id = 0x13090414
Info : flash size = 256kbytes
Info : bank-xnum_sectors = 128
Info : bank-size = 262144
Info : GD32: Flash erasing...
Info : GD32: Flash erase ... sector erase(0 to 5)
Info : GD32: Flash erase ...finished
Info : GD32: Flash writing...
Info : bank-0000000028a0da0 buffer-000000002905580 offset-
Info : GD32: Flash write ... not words to write, padding wit
Info : GD32: Flash write ...words to be prgrammed = 0x000000
** Programming Finished **
** Verify Started **
** Verified OK **
** Resetting Target **
shutdown command invoked
终端将按任务重用, 按任意键关闭。
```


5. 版本历史

表 5-1. 版本历史

版本号.	描述	日期
1.0	首次发布	2021 年 06 月 30 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.