GigaDevice Semiconductor Inc.

Arm® Cortex®-M3/4/23/33 32-bit MCU

Application Note
AN016

Table of Contents

List of Figures

List of Tables

1. Introduction

In the process of project compilation, most of the make tools are used. The make tools include GNU Make, qmake, MS nmake and Makeepp, etc. These make tools follow different specifications, standards and formate when writing makefiles on different platforms. Transplanting the project to a different platform for compilation during the development process will cause compilation failure. Using the CMAKE tool to generate Makefile will effectively solve the above problems. This application manual is based on the GD32F10x SDK development kit, and the CMAKE tool is used to build the compilation environment

2. Development environment constructione

The development environment is mainly introduced as follows:

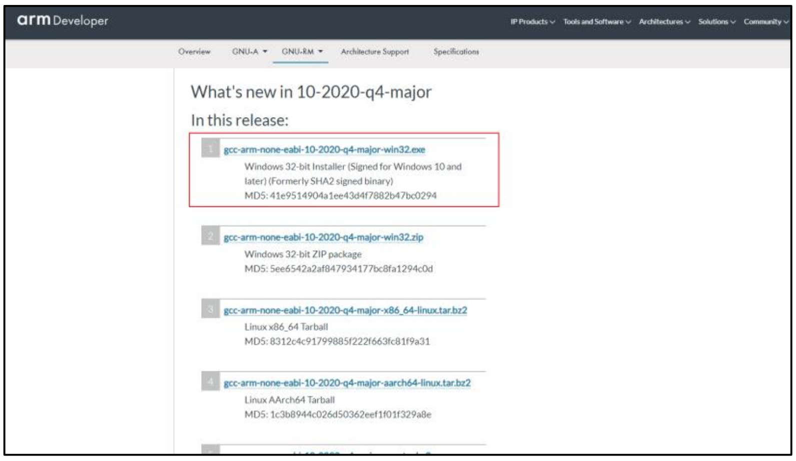
- Hardware development board: GD32F103C-EVAL-V1.0 development board
- Cortex-M3: GD32F103C
- Operating system: Win10-64 bit
- Cross compilation tool chain: gcc-arm-none-eabi
- C/C++ compiler: MinGW
- Development environment: VSCODE+CMAKE
- Debug download tool: OPENOCD

2.1. Install cross-compilation tool

GNU Tools for Arm Embedded Processors download and installation address: https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads.

There are multiple version of GUN Arm Embedded Toolchain is available for downloading on the page. In this application manual, choose to download and install gcc-arm-none-eabi-10-2020-q4-major-win32.exe, as shown in ***Figure 2-1. Select download of GUN Arm Embedded Toolchain***.

Figure 2-1. Select download of GUN Arm Embedded Toolchain



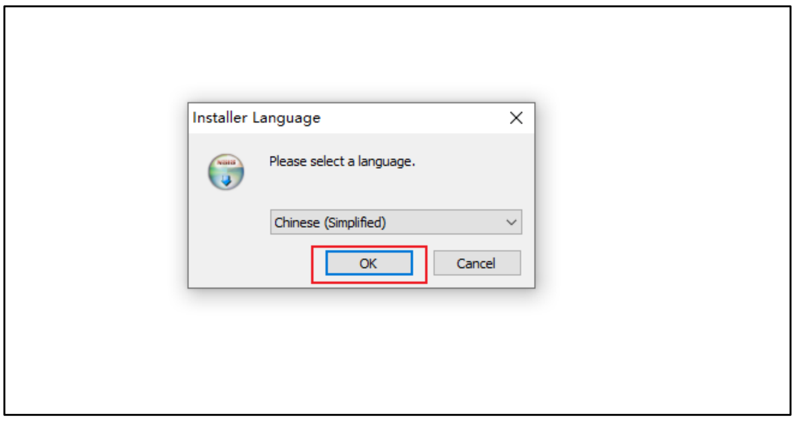
What's new in 10-2020-q4-major

In this release:

1. gcc-arm-none-eabi-10-2020-q4-major-win32.exe
   Windows 32-bit Installer (Signed for Windows 10 and later) (Formerly SHA2 signed binary)
   MD5: 41e9514904a1ee43d4f7882b47bc0294
2. gcc-arm-none-eabi-10-2020-q4-major-win32.zip
   Windows 32-bit ZIP package
   MD5: 5ee6542a2af847934177bc8fa1294c0d
3. gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2
   Linux x86_64 Tarball
   MD5: 8312c4c91799885f222f663fc81f9a31
4. gcc-arm-none-eabi-10-2020-q4-major-aarch64-linux.tar.bz2
   Linux AArch64 Tarball
   MD5: 1c3b8944c026d50362eef1f01f329a8e

After the download is complete, double-click to install and select “OK”.

Figure 2-2. GUN Arm Embedded Toolchain installation process 1

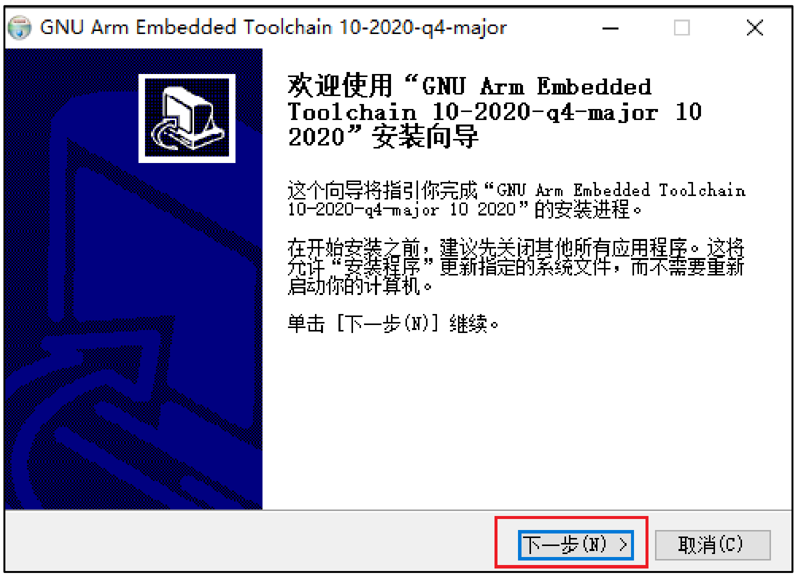

Installer Language

Please select a language.

Chinese (Simplified)

Click “Next”.

Figure 2-3. GUN Arm Embedded Toolchain installation process 2


GNU Arm Embedded Toolchain 10-2020-q4-major

欢迎使用 “GNU Arm Embedded Toolchain 10-2020-q4-major 10 2020” 安装向导

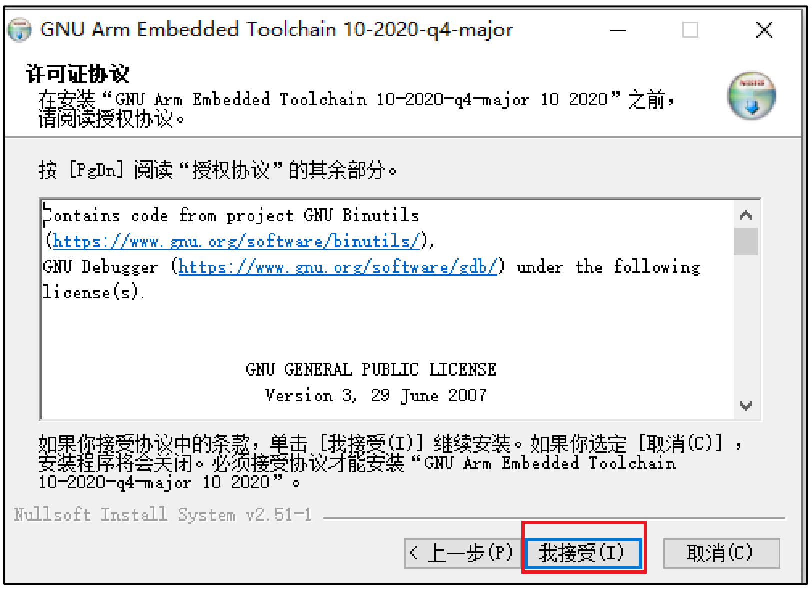
这个向导将指引你完成 “GNU Arm Embedded Toolchain 10-2020-q4-major 10 2020” 的安装进程。

在开始安装之前，建议先关闭其他所有应用程序。这将允许 “安装程序” 更新指定的系统文件，而不需要重新启动你的计算机。

单击 [下一步(N)] 继续。

Click “I accept”, select the installation path, select the default path and click install.

Figure 2-4. GUN Arm Embedded Toolchain installation process 3


GNU Arm Embedded Toolchain 10-2020-q4-major

许可证协议

在安装“GNU Arm Embedded Toolchain 10-2020-q4-major 10 2020”之前，请阅读授权协议。

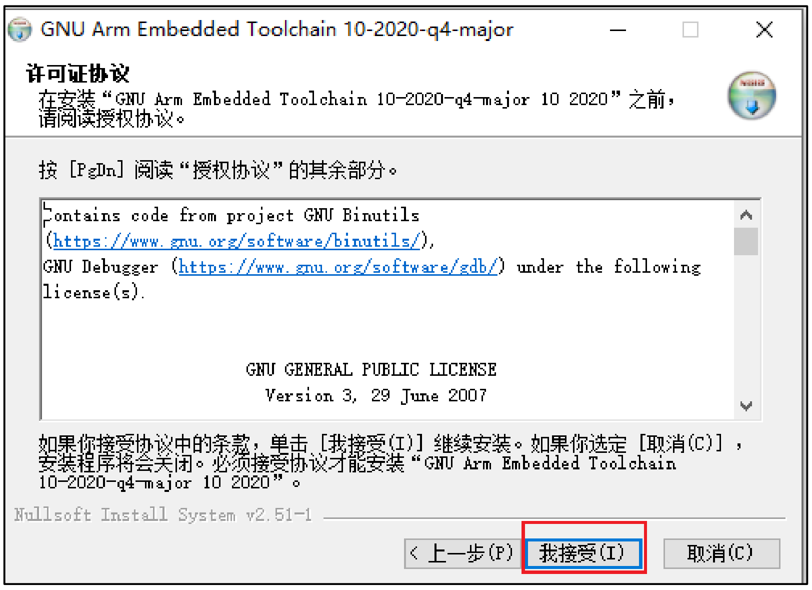
按 [PgDn] 阅读“授权协议”的其余部分。

```
Contains code from project GNU Binutils
(https://www.gnu.org/software/binutils/),
GNU Debugger (https://www.gnu.org/software/gdb/) under the following
license(s).


                    GNU GENERAL PUBLIC LICENSE
                      Version 3, 29 June 2007
```

如果你接受协议中的条款，单击 [我接受(I)] 继续安装。如果你选定 [取消(C)]，安装程序将会关闭。必须接受协议才能安装“GNU Arm Embedded Toolchain 10-2020-q4-major 10 2020”。

Nullsoft Install System v2.51-1

Figure 2-5. GUN Arm Embedded Toolchain installation process 4


GNU Arm Embedded Toolchain 10-2020-q4-major

许可证协议

在安装“GNU Arm Embedded Toolchain 10-2020-q4-major 10 2020”之前，请阅读授权协议。

按 [PgDn] 阅读“授权协议”的其余部分。

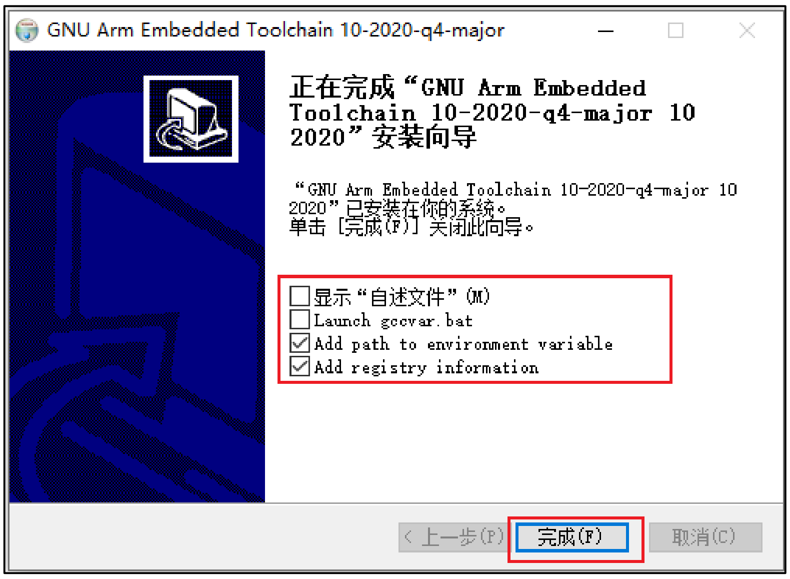
```
Contains code from project GNU Binutils
(https://www.gnu.org/software/binutils/),
GNU Debugger (https://www.gnu.org/software/gdb/) under the following
license(s).


                    GNU GENERAL PUBLIC LICENSE
                      Version 3, 29 June 2007
```

如果你接受协议中的条款，单击 [我接受(I)] 继续安装。如果你选定 [取消(C)]，安装程序将会关闭。必须接受协议才能安装“GNU Arm Embedded Toolchain 10-2020-q4-major 10 2020”。

Nullsoft Install System v2.51-1

When the installation is complete, select the add path to environment variable option and click “Finish”.

Figure 2-6. GUN Arm Embedded Toolchain installation process 5


GNU Arm Embedded Toolchain 10-2020-q4-major

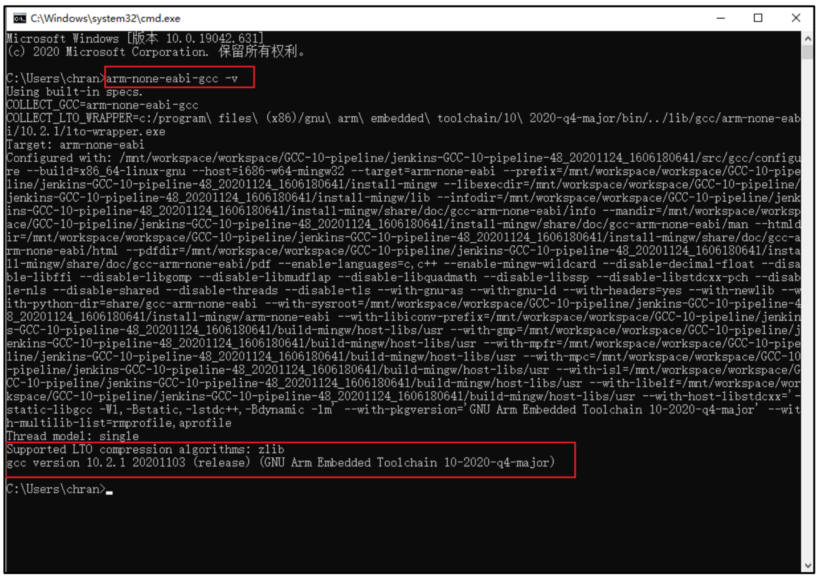
正在完成“GNU Arm Embedded Toolchain 10-2020-q4-major 10 2020”安装向导

“GNU Arm Embedded Toolchain 10-2020-q4-major 10 2020”已安装在你的系统。
单击［完成(F)］关闭此向导。

- [ ] 显示“自述文件”(M)
- [ ] Launch gccvar.bat
- [x] Add path to environment variable
- [x] Add registry information

Check whether the installation is successful, enter cmd in the operation, click “OK”, enter “arm-none-eabi-gcc –v” in the command line, and the return result as shown in ***Figure 2-7. Test whether the GUN Arm Embedded Toolchain is installed successfully*** indicates that the installation is successful.

Figure 2-7. Test whether the GUN Arm Embedded Toolchain is installed successfully


C:\Windows\system32\cmd.exe

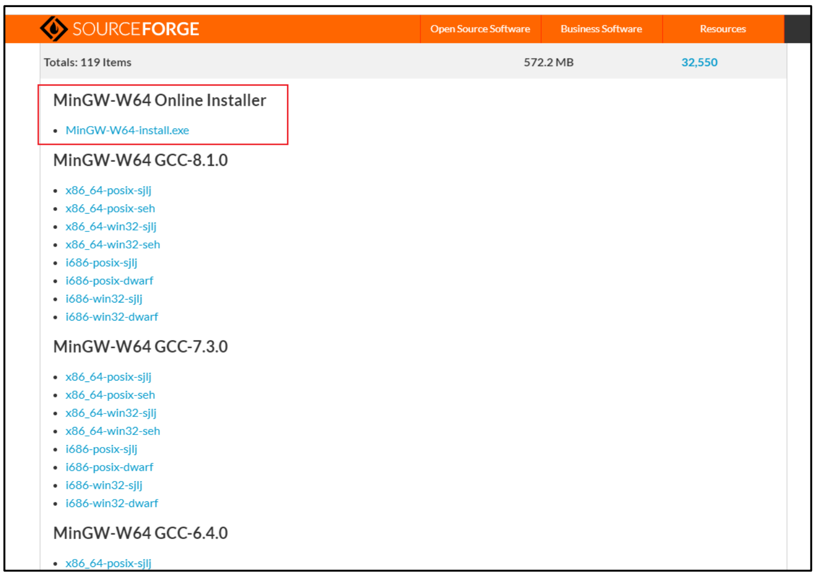
```
Microsoft Windows [版本 10.0.19042.631]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\chran>arm-none-eabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-none-eabi-gcc
COLLECT_LTO_WRAPPER=c:/program\ files\ (x86)/gnu\ arm\ embedded\ toolchain/10\ 2020-q4-major/bin/../lib/gcc/arm-none-eab
i/10.2.1/lto-wrapper.exe
Target: arm-none-eabi
Configured with: /mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/src/gcc/configu
re --build=x86_64-linux-gnu --host=i686-w64-mingw32 --target=arm-none-eabi --prefix=/mnt/workspace/workspace/GCC-10-pipe
line/jenkins-GCC-10-pipeline-48_20201124_1606180641/install-mingw --libexecdir=/mnt/workspace/workspace/GCC-10-pipeline/
jenkins-GCC-10-pipeline-48_20201124_1606180641/install-mingw/lib --infodir=/mnt/workspace/workspace/GCC-10-pipeline/jenk
ins-GCC-10-pipeline-48_20201124_1606180641/install-mingw/share/doc/gcc-arm-none-eabi/info --mandir=/mnt/workspace/worksp
ace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/install-mingw/share/doc/gcc-arm-none-eabi/man --htmld
ir=/mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/install-mingw/share/doc/gcc-a
rm-none-eabi/html --pdfdir=/mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/insta
ll-mingw/share/doc/gcc-arm-none-eabi/pdf --enable-languages=c,c++ --enable-mingw-wildcard --disable-decimal-float --disa
ble-libffi --disable-libgomp --disable-libmudflap --disable-libquadmath --disable-libssp --disable-libstdcxx-pch --disab
le-nls --disable-shared --disable-threads --disable-tls --with-gnu-as --with-gnu-ld --with-headers=yes --with-newlib --w
ith-python-dir=share/gcc-arm-none-eabi --with-sysroot=/mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-4
8_20201124_1606180641/install-mingw/arm-none-eabi --with-libiconv-prefix=/mnt/workspace/workspace/GCC-10-pipeline/jenkin
s-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --with-gmp=/mnt/workspace/workspace/GCC-10-pipeline/j
enkins-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --with-mpfr=/mnt/workspace/workspace/GCC-10-pipe
line/jenkins-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --with-mpc=/mnt/workspace/workspace/GCC-10
-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --with-isl=/mnt/workspace/workspace/G
CC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --with-libelf=/mnt/workspace/wor
kspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --with-host-libstdcxx='-
static-libgcc -Wl,-Bstatic,-lstdc++,-Bdynamic -lm' --with-pkgversion='GNU Arm Embedded Toolchain 10-2020-q4-major' --wit
h-multilib-list=rmprofile,aprofile
Thread model: single
Supported LTO compression algorithms: zlib
gcc version 10.2.1 20201103 (release) (GNU Arm Embedded Toolchain 10-2020-q4-major)

C:\Users\chran>_
```

2.2. Install C/C++ MinGW compiler

MinGW installation package download address: https://sourceforge.net/projects/mingw-w64/files/mingw-w64/mingw-w64-release/.

There are two installation methods, online installation and offline installation, this manual chooses online installation. Download MinGW-W64-install.exe.

Figure 2-8. MinGW-W64 choose to download and install



Totals: 119 Items 572.2 MB 32,550

MinGW-W64 Online Installer

- MinGW-W64-install.exe

MinGW-W64 GCC-8.1.0

- x86_64-posix-sjlj
- x86_64-posix-seh
- x86_64-win32-sjlj
- x86_64-win32-seh
- i686-posix-sjlj
- i686-posix-dwarf
- i686-win32-sjlj
- i686-win32-dwarf

MinGW-W64 GCC-7.3.0

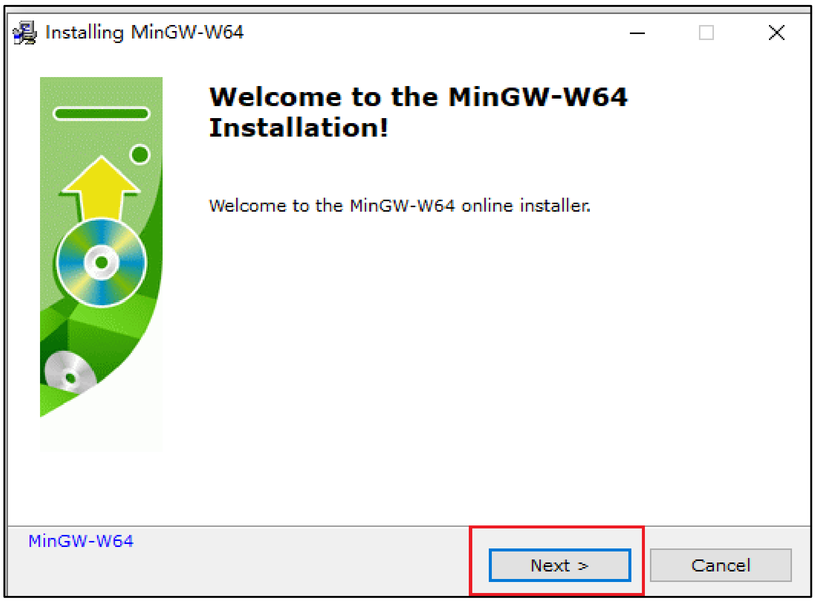
- x86_64-posix-sjlj
- x86_64-posix-seh
- x86_64-win32-sjlj
- x86_64-win32-seh
- i686-posix-sjlj
- i686-posix-dwarf
- i686-win32-sjlj
- i686-win32-dwarf

MinGW-W64 GCC-6.4.0

- x86_64-posix-sjlj

After the download is complete, double-click to install. Select “Next”.

Figure 2-9. MinGW-W64 installation process 1

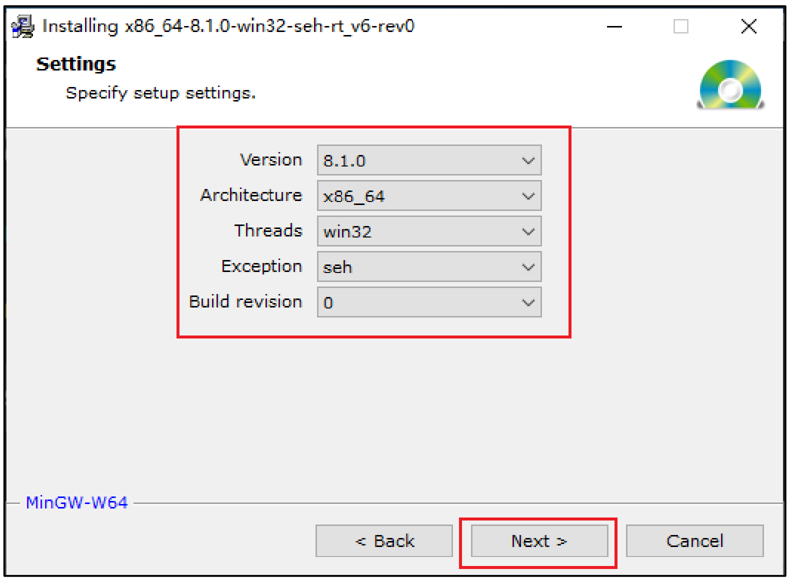


Welcome to the MinGW-W64 Installation!

Welcome to the MinGW-W64 online installer.

The configuration options are as follows, click “Next”.

Figure 2-10. MinGW-W64 installation process 2



Installing x86_64-8.1.0-win32-seh-rt_v6-rev0

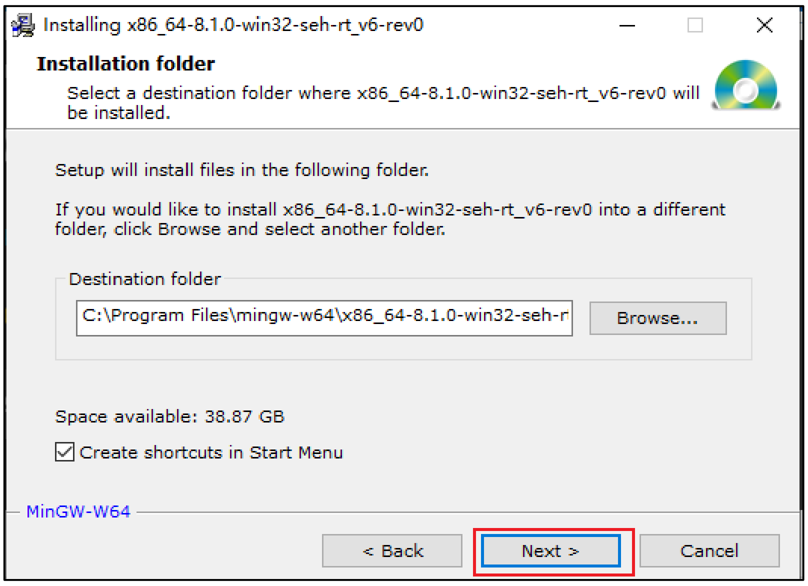
Settings

Specify setup settings.

Version	8.1.0
Architecture	x86_64
Threads	win32
Exception	seh
Build revision	0

Select the installation path, select the default path.

Figure 2-11. MinGW-W64 installation process 3



Installing x86_64-8.1.0-win32-seh-rt_v6-rev0

Installation folder

Select a destination folder where x86_64-8.1.0-win32-seh-rt_v6-rev0 will be installed.

Setup will install files in the following folder.

If you would like to install x86_64-8.1.0-win32-seh-rt_v6-rev0 into a different folder, click Browse and select another folder.

Destination folder

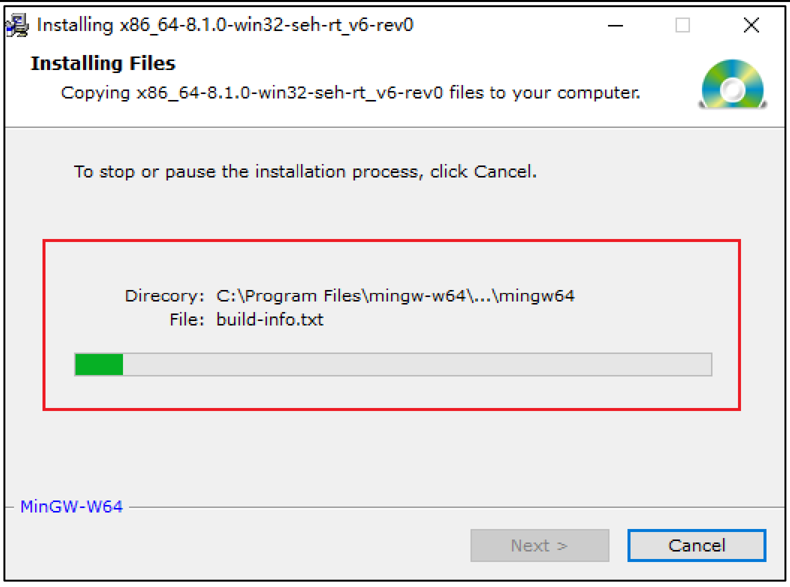
C:\Program Files\mingw-w64\x86_64-8.1.0-win32-seh-r

Space available: 38.87 GB

☑ Create shortcuts in Start Menu

Wait for the file download process.

Figure 2-12. MinGW-W64 installation process 4



Installing x86_64-8.1.0-win32-seh-rt_v6-rev0

Installing Files

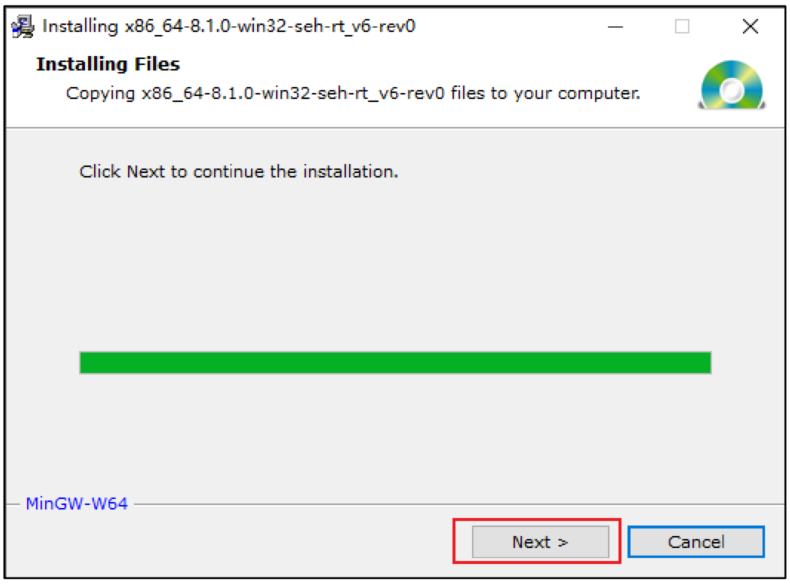
Copying x86_64-8.1.0-win32-seh-rt_v6-rev0 files to your computer.

To stop or pause the installation process, click Cancel.

Direcory: C:\Program Files\mingw-w64\...\mingw64
File: build-info.txt

Click “Next”.

Figure 2-13. MinGW-W64 installation process 5



Installing x86_64-8.1.0-win32-seh-rt_v6-rev0

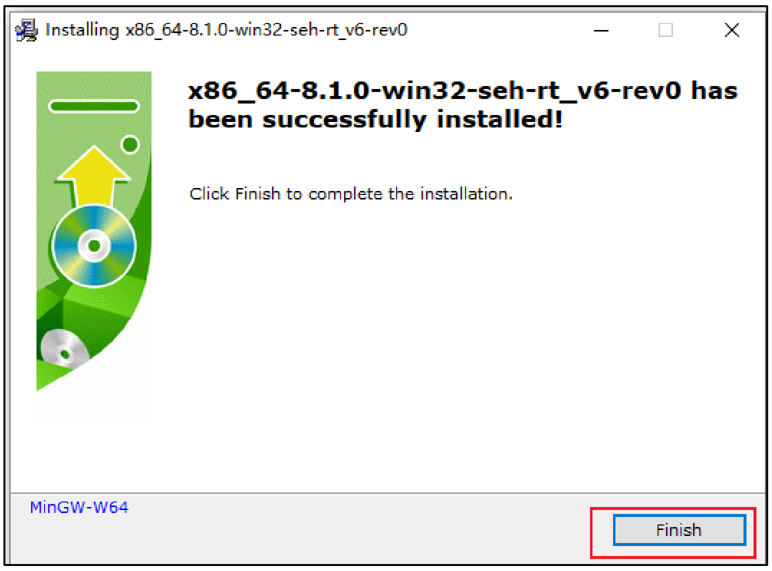
Installing Files

Copying x86_64-8.1.0-win32-seh-rt_v6-rev0 files to your computer.

Click Next to continue the installation.

Click “Finish” to complete the installation.

Figure 2-14. MinGW-W64 installation process 6


Installing x86_64-8.1.0-win32-seh-rt_v6-rev0

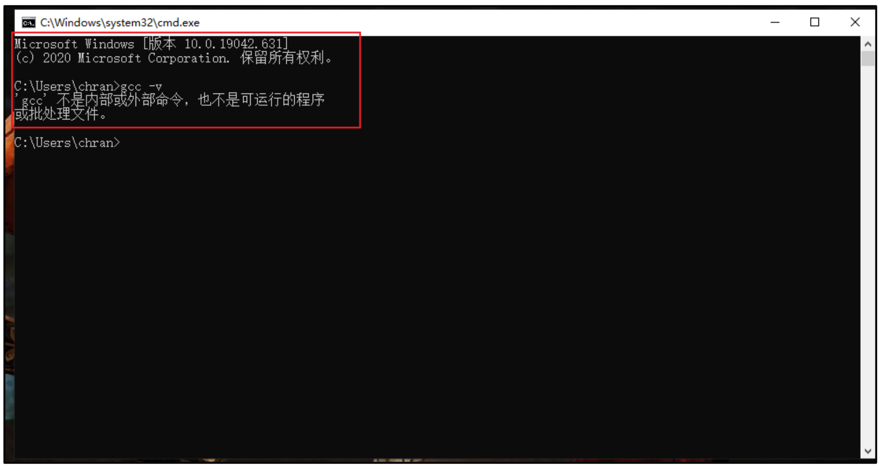
x86_64-8.1.0-win32-seh-rt_v6-rev0 has been successfully installed!

Click Finish to complete the installation.

MinGW-W64

Enter “gcc –v” in the cmd command line to output non-runnable programs. The solution to this problem is to manually add the downloaded file to the system environment variable.

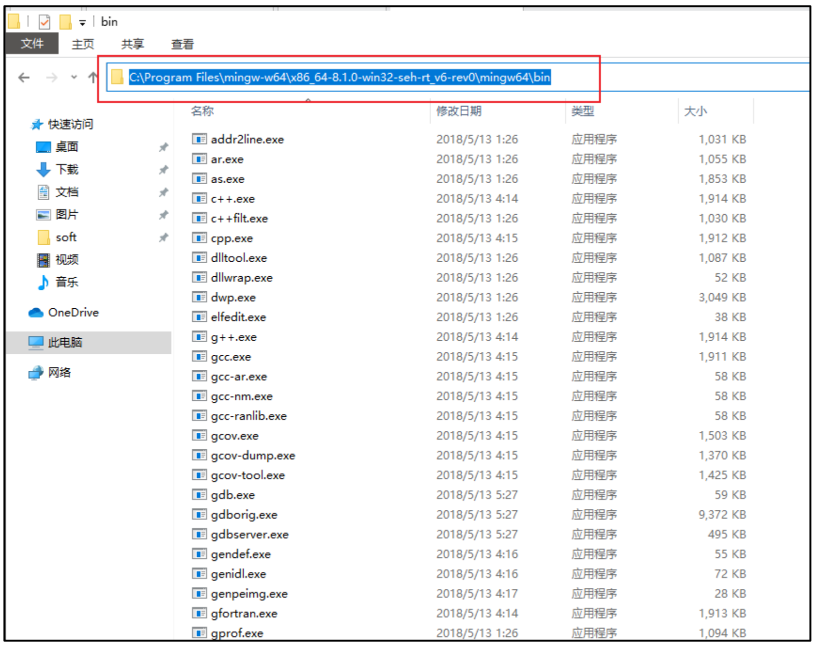
Figure 2-15. Test whether MinGW-W64 is installed successfully 1


C:\Windows\system32\cmd.exe

```
Microsoft Windows [版本 10.0.19042.631]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\chran>gcc -v
'gcc' 不是内部或外部命令，也不是可运行的程序
或批处理文件。

C:\Users\chran>
```

Open the MinGW installation path, copy the path in the /bin folder, and add the path to the system environment variables.

Figure 2-16. Copy MinGW-W64 bin folder path

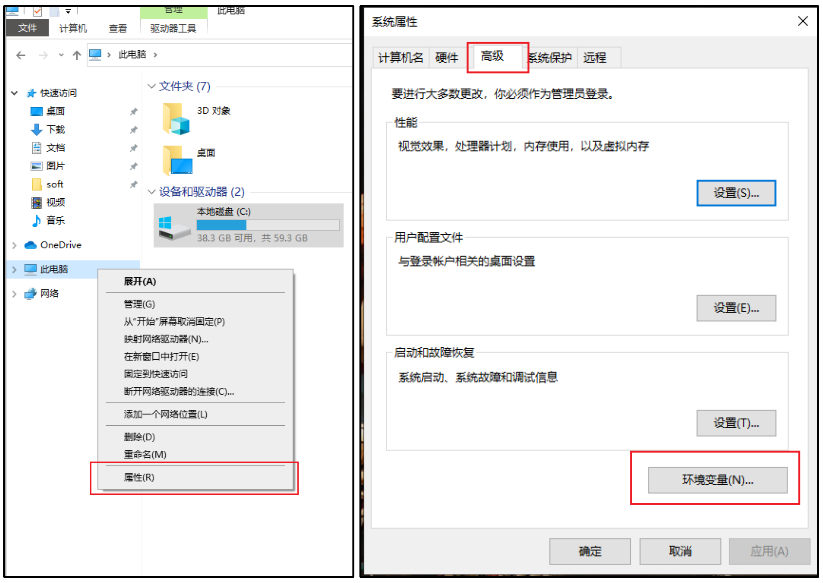


C:\Program Files\mingw-w64\x86_64-8.1.0-win32-seh-rt_v6-rev0\mingw64\bin

名称	修改日期	类型	大小
addr2line.exe	2018/5/13 1:26	应用程序	1,031 KB
ar.exe	2018/5/13 1:26	应用程序	1,055 KB
as.exe	2018/5/13 1:26	应用程序	1,853 KB
c++.exe	2018/5/13 4:14	应用程序	1,914 KB
c++filt.exe	2018/5/13 1:26	应用程序	1,030 KB
cpp.exe	2018/5/13 4:15	应用程序	1,912 KB
dlltool.exe	2018/5/13 1:26	应用程序	1,087 KB
dllwrap.exe	2018/5/13 1:26	应用程序	52 KB
dwp.exe	2018/5/13 1:26	应用程序	3,049 KB
elfedit.exe	2018/5/13 1:26	应用程序	38 KB
g++.exe	2018/5/13 4:14	应用程序	1,914 KB
gcc.exe	2018/5/13 4:15	应用程序	1,911 KB
gcc-ar.exe	2018/5/13 4:15	应用程序	58 KB
gcc-nm.exe	2018/5/13 4:15	应用程序	58 KB
gcc-ranlib.exe	2018/5/13 4:15	应用程序	58 KB
gcov.exe	2018/5/13 4:15	应用程序	1,503 KB
gcov-dump.exe	2018/5/13 4:15	应用程序	1,370 KB
gcov-tool.exe	2018/5/13 4:15	应用程序	1,425 KB
gdb.exe	2018/5/13 5:27	应用程序	59 KB
gdborig.exe	2018/5/13 5:27	应用程序	9,372 KB
gdbserver.exe	2018/5/13 5:27	应用程序	495 KB
gendef.exe	2018/5/13 4:16	应用程序	55 KB
genidl.exe	2018/5/13 4:16	应用程序	72 KB
genpeimg.exe	2018/5/13 4:17	应用程序	28 KB
gfortran.exe	2018/5/13 4:14	应用程序	1,913 KB
gprof.exe	2018/5/13 1:26	应用程序	1,094 KB

Right-click "this computer" and select properties, click "advanced" to select environment variables.

Figure 2-17. Add MinGW-W64 environment variable 1



要进行大多数更改，你必须作为管理员登录。

性能

视觉效果，处理器计划，内存使用，以及虚拟内存

用户配置文件

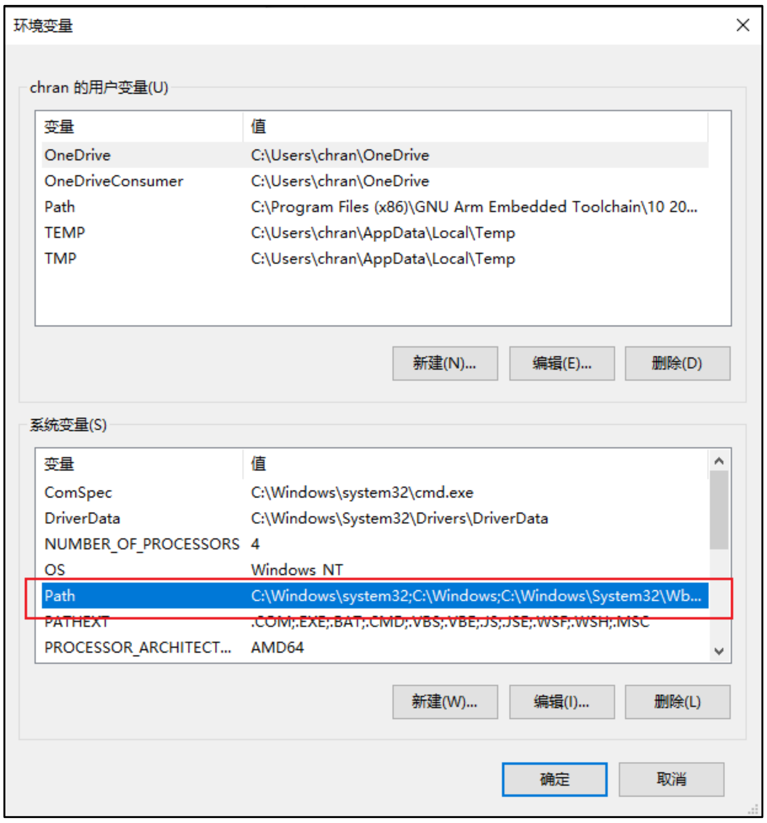
与登录帐户相关的桌面设置

启动和故障恢复

系统启动、系统故障和调试信息

Select the system variable and click Path.

Figure 2-18. Add MinGW-W64 environment variable 2

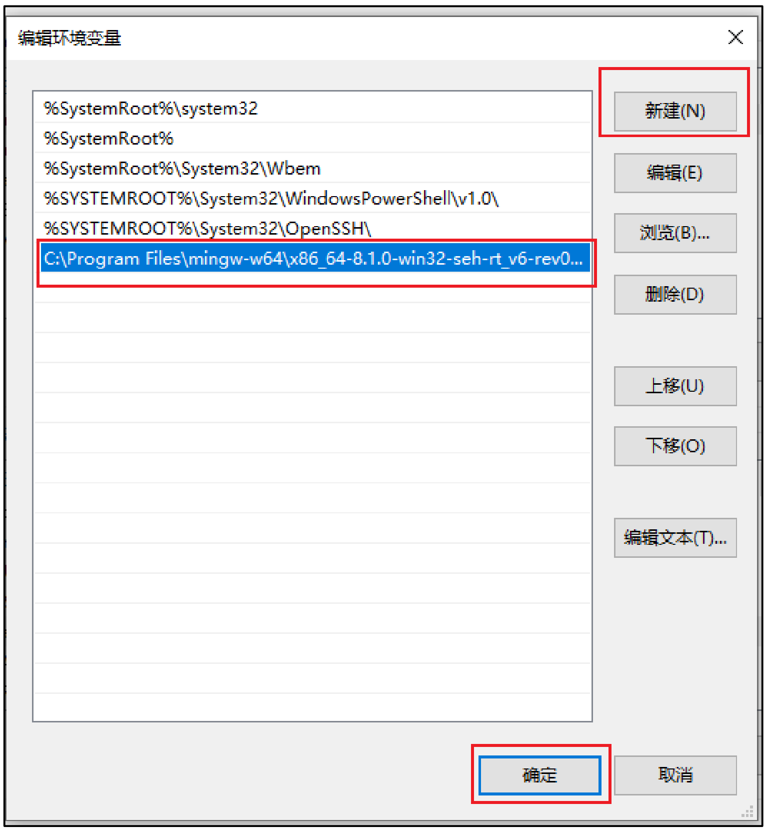


变量	值
OneDrive	C:\Users\chran\OneDrive
OneDriveConsumer	C:\Users\chran\OneDrive
Path	C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 20...
TEMP	C:\Users\chran\AppData\Local\Temp
TMP	C:\Users\chran\AppData\Local\Temp

变量	值
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wb...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECT...	AMD64

Click "New", paste the copied path, and click "OK".

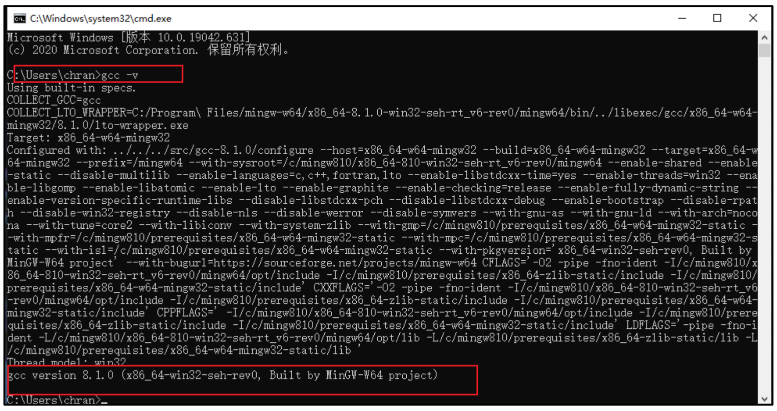
Figure 2-19. Add MinGW-W64 environment variable 3



```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\mingw-w64\x86_64-8.1.0-win32-seh-rt_v6-rev0...
```

Enter "gcc –v" in cmd and output the gcc -v version number. The installation is successful.

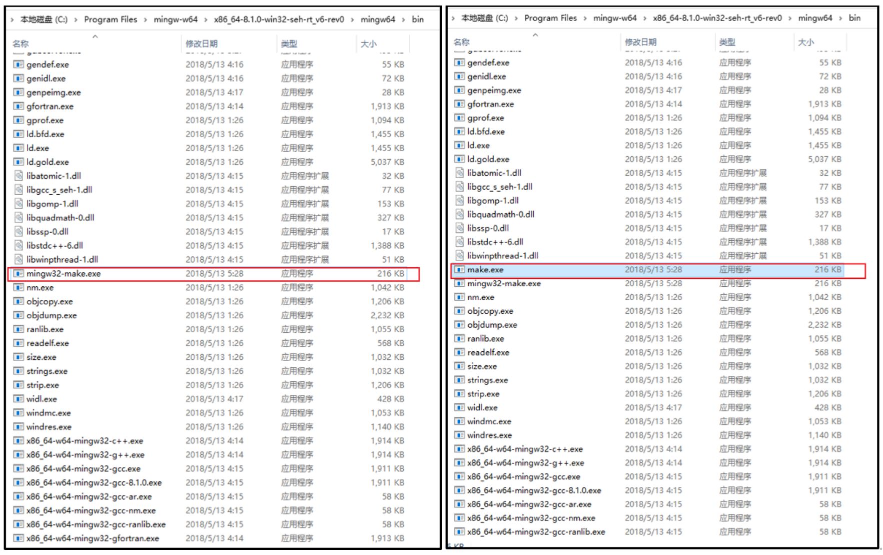
Figure 2-20. Test whether MinGW-W64 is installed successfully 2



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19042.631]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\chran>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=C:/Program\ Files/mingw-w64/x86_64-8.1.0-win32-seh-rt_v6-rev0/mingw64/bin/../libexec/gcc/x86_64-w64-
mingw32/8.1.0/lto-wrapper.exe
Target: x86_64-w64-mingw32
Configured with: ../../../src/gcc-8.1.0/configure --host=x86_64-w64-mingw32 --build=x86_64-w64-mingw32 --target=x86_64-w
64-mingw32 --prefix=/mingw64 --with-sysroot=/c/mingw810/x86_64-810-win32-seh-rt_v6-rev0/mingw64 --enable-shared --enable
-static --disable-multilib --enable-languages=c,c++,fortran,lto --enable-libstdcxx-time=yes --enable-threads=win32 --ena
ble-libgomp --enable-libatomic --enable-lto --enable-graphite --enable-checking=release --enable-fully-dynamic-string --
enable-version-specific-runtime-libs --disable-libstdcxx-pch --disable-libstdcxx-debug --enable-bootstrap --disable-rpat
h --disable-win32-registry --disable-nls --disable-werror --disable-symvers --with-gnu-as --with-gnu-ld --with-arch=noco
na --with-tune=core2 --with-libiconv --with-system-zlib --with-gmp=/c/mingw810/prerequisites/x86_64-w64-mingw32-static -
-with-mpfr=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-mpc=/c/mingw810/prerequisites/x86_64-w64-mingw32-s
tatic --with-isl=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-pkgversion='x86_64-win32-seh-rev0, Built by
MinGW-W64 project' --with-bugurl=https://sourceforge.net/projects/mingw-w64 CFLAGS='-O2 -pipe -fno-ident -I/c/mingw810/x
86_64-810-win32-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/
prerequisites/x86_64-w64-mingw32-static/include' CXXFLAGS='-O2 -pipe -fno-ident -I/c/mingw810/x86_64-810-win32-seh-rt_v6
-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-
mingw32-static/include' CPPFLAGS=' -I/c/mingw810/x86_64-810-win32-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prere
quisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' LDFLAGS='-pipe -fno-i
dent -L/c/mingw810/x86_64-810-win32-seh-rt_v6-rev0/mingw64/opt/lib -L/c/mingw810/prerequisites/x86_64-zlib-static/lib -L
/c/mingw810/prerequisites/x86_64-w64-mingw32-static/lib '
Thread model: win32
gcc version 8.1.0 (x86_64-win32-seh-rev0, Built by MinGW-W64 project)

C:\Users\chran>_
```

Copy the mingw32-make.exe file under the MinGW installation path/bin folder and rename it as make. Use this command to directly enter "make" to achieve.

Figure 2-21. Modify MinGW-W64 mingw32-make command to make command



本地磁盘 (C:) › Program Files › mingw-w64 › x86_64-8.1.0-win32-seh-rt_v6-rev0 › mingw64 › bin

名称	修改日期	类型	大小
gendef.exe	2018/5/13 4:16	应用程序	55 KB
genidl.exe	2018/5/13 4:16	应用程序	72 KB
genpeimg.exe	2018/5/13 4:17	应用程序	28 KB
gfortran.exe	2018/5/13 4:14	应用程序	1,913 KB
gprof.exe	2018/5/13 1:26	应用程序	1,094 KB
ld.bfd.exe	2018/5/13 1:26	应用程序	1,455 KB
ld.exe	2018/5/13 1:26	应用程序	1,455 KB
ld.gold.exe	2018/5/13 1:26	应用程序	5,037 KB
libatomic-1.dll	2018/5/13 4:15	应用程序扩展	32 KB
libgcc_s_seh-1.dll	2018/5/13 4:15	应用程序扩展	77 KB
libgomp-1.dll	2018/5/13 4:15	应用程序扩展	153 KB
libquadmath-0.dll	2018/5/13 4:15	应用程序扩展	327 KB
libssp-0.dll	2018/5/13 4:15	应用程序扩展	17 KB
libstdc++-6.dll	2018/5/13 4:15	应用程序扩展	1,388 KB
libwinpthread-1.dll	2018/5/13 4:15	应用程序扩展	51 KB
mingw32-make.exe	2018/5/13 5:28	应用程序	216 KB
nm.exe	2018/5/13 1:26	应用程序	1,042 KB
objcopy.exe	2018/5/13 1:26	应用程序	1,206 KB
objdump.exe	2018/5/13 1:26	应用程序	2,232 KB
ranlib.exe	2018/5/13 1:26	应用程序	1,055 KB
readelf.exe	2018/5/13 1:26	应用程序	568 KB
size.exe	2018/5/13 1:26	应用程序	1,032 KB
strings.exe	2018/5/13 1:26	应用程序	1,032 KB
strip.exe	2018/5/13 1:26	应用程序	1,206 KB
widl.exe	2018/5/13 4:17	应用程序	428 KB
windmc.exe	2018/5/13 1:26	应用程序	1,053 KB
windres.exe	2018/5/13 1:26	应用程序	1,140 KB
x86_64-w64-mingw32-c++.exe	2018/5/13 4:14	应用程序	1,914 KB
x86_64-w64-mingw32-g++.exe	2018/5/13 4:14	应用程序	1,914 KB
x86_64-w64-mingw32-gcc.exe	2018/5/13 4:15	应用程序	1,911 KB
x86_64-w64-mingw32-gcc-8.1.0.exe	2018/5/13 4:15	应用程序	1,911 KB
x86_64-w64-mingw32-gcc-ar.exe	2018/5/13 4:15	应用程序	58 KB
x86_64-w64-mingw32-gcc-nm.exe	2018/5/13 4:15	应用程序	58 KB
x86_64-w64-mingw32-gcc-ranlib.exe	2018/5/13 4:15	应用程序	58 KB
x86_64-w64-mingw32-gfortran.exe	2018/5/13 4:14	应用程序	1,913 KB

本地磁盘 (C:) › Program Files › mingw-w64 › x86_64-8.1.0-win32-seh-rt_v6-rev0 › mingw64 › bin

名称	修改日期	类型	大小
gendef.exe	2018/5/13 4:16	应用程序	55 KB
genidl.exe	2018/5/13 4:16	应用程序	72 KB
genpeimg.exe	2018/5/13 4:17	应用程序	28 KB
gfortran.exe	2018/5/13 4:14	应用程序	1,913 KB
gprof.exe	2018/5/13 1:26	应用程序	1,094 KB
ld.bfd.exe	2018/5/13 1:26	应用程序	1,455 KB
ld.exe	2018/5/13 1:26	应用程序	1,455 KB
ld.gold.exe	2018/5/13 1:26	应用程序	5,037 KB
libatomic-1.dll	2018/5/13 4:15	应用程序扩展	32 KB
libgcc_s_seh-1.dll	2018/5/13 4:15	应用程序扩展	77 KB
libgomp-1.dll	2018/5/13 4:15	应用程序扩展	153 KB
libquadmath-0.dll	2018/5/13 4:15	应用程序扩展	327 KB
libssp-0.dll	2018/5/13 4:15	应用程序扩展	17 KB
libstdc++-6.dll	2018/5/13 4:15	应用程序扩展	1,388 KB
libwinpthread-1.dll	2018/5/13 4:15	应用程序扩展	51 KB
make.exe	2018/5/13 5:28	应用程序	216 KB
mingw32-make.exe	2018/5/13 5:28	应用程序	216 KB
nm.exe	2018/5/13 1:26	应用程序	1,042 KB
objcopy.exe	2018/5/13 1:26	应用程序	1,206 KB
objdump.exe	2018/5/13 1:26	应用程序	2,232 KB
ranlib.exe	2018/5/13 1:26	应用程序	1,055 KB
readelf.exe	2018/5/13 1:26	应用程序	568 KB
size.exe	2018/5/13 1:26	应用程序	1,032 KB
strings.exe	2018/5/13 1:26	应用程序	1,032 KB
strip.exe	2018/5/13 1:26	应用程序	1,206 KB
widl.exe	2018/5/13 4:17	应用程序	428 KB
windmc.exe	2018/5/13 1:26	应用程序	1,053 KB
windres.exe	2018/5/13 1:26	应用程序	1,140 KB
x86_64-w64-mingw32-c++.exe	2018/5/13 4:14	应用程序	1,914 KB
x86_64-w64-mingw32-g++.exe	2018/5/13 4:14	应用程序	1,914 KB
x86_64-w64-mingw32-gcc.exe	2018/5/13 4:15	应用程序	1,911 KB
x86_64-w64-mingw32-gcc-8.1.0.exe	2018/5/13 4:15	应用程序	1,911 KB
x86_64-w64-mingw32-gcc-ar.exe	2018/5/13 4:15	应用程序	58 KB
x86_64-w64-mingw32-gcc-nm.exe	2018/5/13 4:15	应用程序	58 KB
x86_64-w64-mingw32-gcc-ranlib.exe	2018/5/13 4:15	应用程序	58 KB

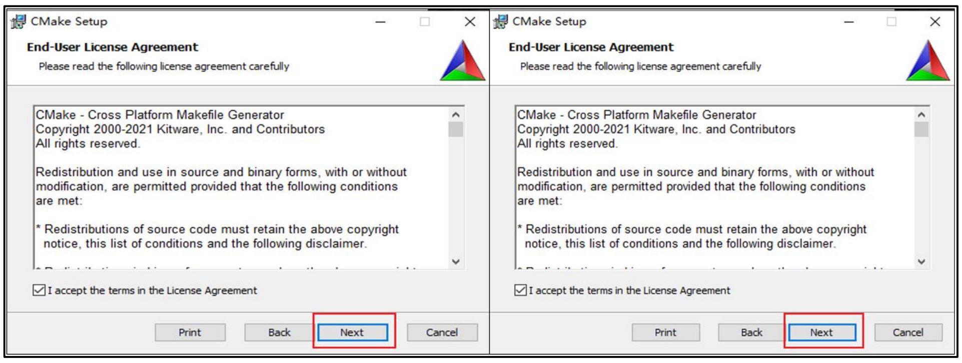
2.3. Install Cmake tool

Cmake tool download and installation address: https://cmake.org/download/

In this application manual, download and install version 3.20.1, and the installation process is as follows.

After the download is complete, double-click to install. Select "Next".

Figure 2-22. Cmake installation process 1


CMake Setup

End-User License Agreement

Please read the following license agreement carefully

CMake - Cross Platform Makefile Generator
Copyright 2000-2021 Kitware, Inc. and Contributors
All rights reserved.

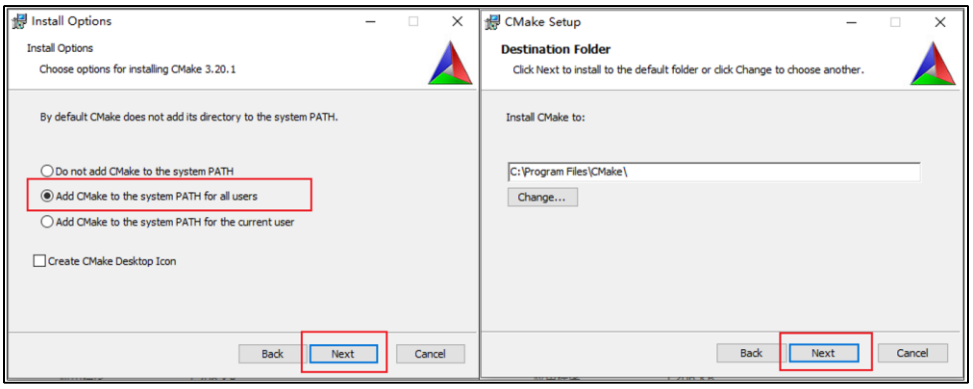
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

[✓] I accept the terms in the License Agreement

Select "Add Cmake to system PATH for all users", click “Next”, select the default path, and click “Next”.

Figure 2-23. Cmake installation process 2


Install Options

Install Options

Choose options for installing CMake 3.20.1

By default CMake does not add its directory to the system PATH.

- ○ Do not add CMake to the system PATH
- ● Add CMake to the system PATH for all users
- ○ Add CMake to the system PATH for the current user

☐ Create CMake Desktop Icon

CMake Setup

Destination Folder

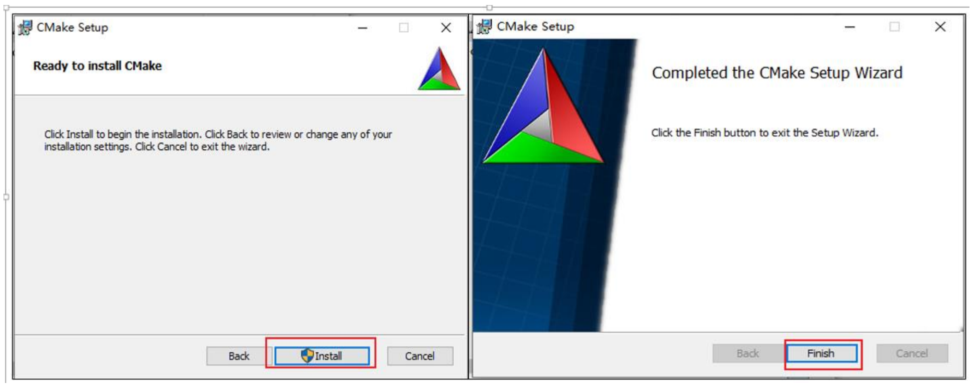
Click Next to install to the default folder or click Change to choose another.

Install CMake to:

C:\Program Files\CMake\

Click “Install”, wait for the installation to complete, and click Finish to complete the installation.

Figure 2-24. Cmake installation process 3


CMake Setup

Ready to install CMake

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

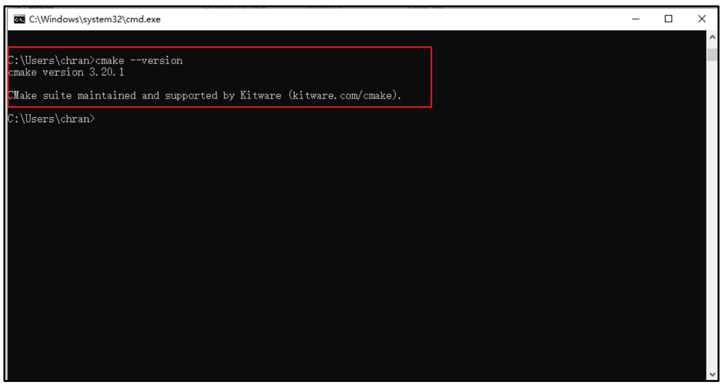
CMake Setup

Completed the CMake Setup Wizard

Click the Finish button to exit the Setup Wizard.

Check whether the installation is successful, enter cmd in the run, click OK, enter cmake --version in the command line, the return result as shown in ***Figure 2-25. Test whether Cmake is installed successfully*** represents the successful installation.

Figure 2-25. Test whether Cmake is installed successfully



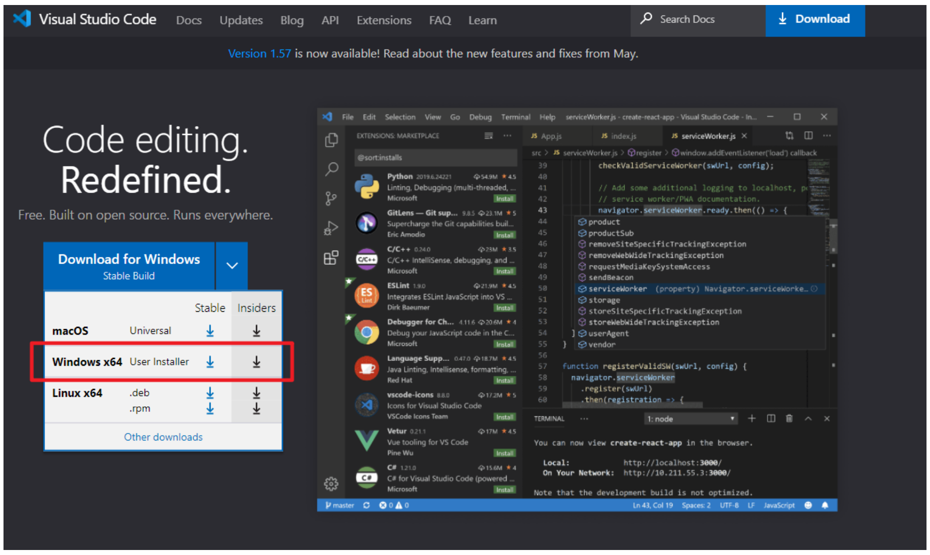
```
C:\Users\chran>cmake --version
cmake version 3.20.1

CMake suite maintained and supported by Kitware (kitware.com/cmake).

C:\Users\chran>
```

2.4. Install Vscode and plug-ins

Vscode download and installation address: https://code.visualstudio.com/

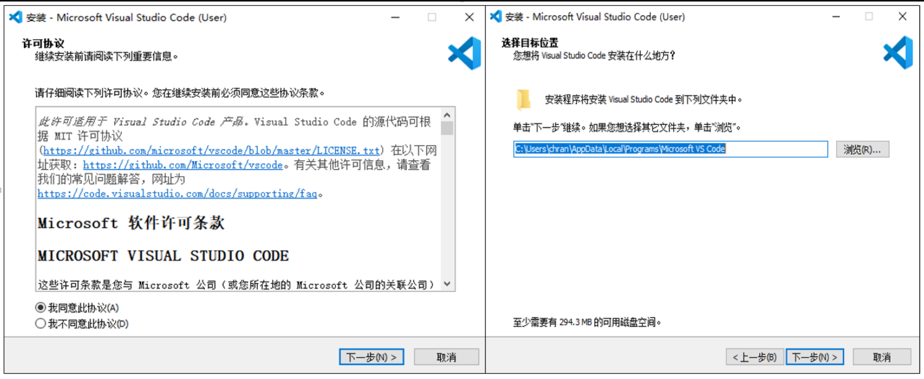
In this application manual, download and install version 1.53.0, the download and installation process is as follows.

Figure 2-26. Vscode choose to download and install



After the download is complete, double-click to install. Select "Next", select the installation path, and click "Next".

Figure 2-27. Vscode installation process 1



安装 - Microsoft Visual Studio Code (User)

许可协议
继续安装前请阅读下列重要信息。

请仔细阅读下列许可协议。您在继续安装前必须同意这些协议条款。

*此许可适用于 Visual Studio Code 产品。*Visual Studio Code 的源代码可根据 MIT 许可协议 (https://github.com/microsoft/vscode/blob/master/LICENSE.txt) 在以下网址获取：https://github.com/Microsoft/vscode。有关其他许可信息，请查看我们的常见问题解答，网址为 https://code.visualstudio.com/docs/supporting/faq。

Microsoft 软件许可条款

MICROSOFT VISUAL STUDIO CODE

这些许可条款是您与 Microsoft 公司（或您所在地的 Microsoft 公司的关联公司）

[✓ 我同意此协议(A)]
我不同意此协议(D)

安装 - Microsoft Visual Studio Code (User)

选择目标位置
您想将 Visual Studio Code 安装在什么地方？

安装程序将安装 Visual Studio Code 到下列文件夹中。

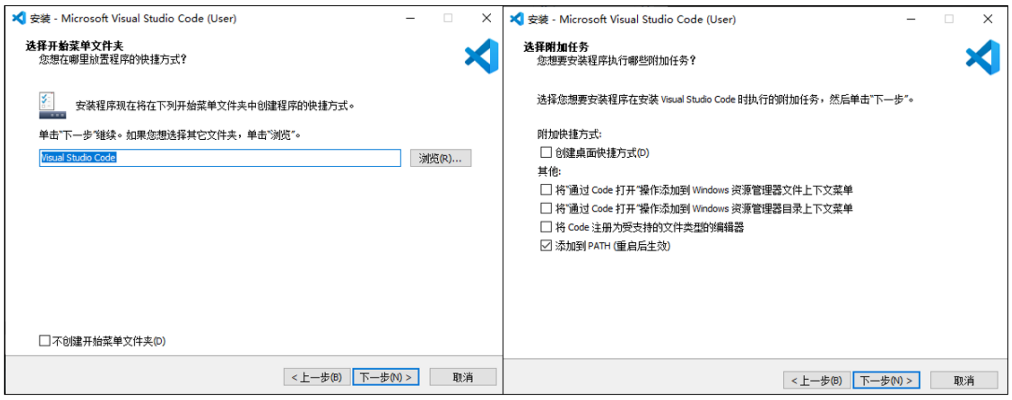
单击"下一步"继续。如果您想选择其它文件夹，单击"浏览"。

C:\Users\chran\AppData\Local\Programs\Microsoft VS Code

至少需要有 294.3 MB 的可用磁盘空间。

Keep clicking “Next” to complete the installation.

Figure 2-28. Vscode installation process 2



安装 - Microsoft Visual Studio Code (User)

选择开始菜单文件夹
您想在哪里放置程序的快捷方式？

安装程序现在将在下列开始菜单文件夹中创建程序的快捷方式。

单击"下一步"继续。如果您想选择其它文件夹，单击"浏览"。

Visual Studio Code

☐ 不创建开始菜单文件夹(D)

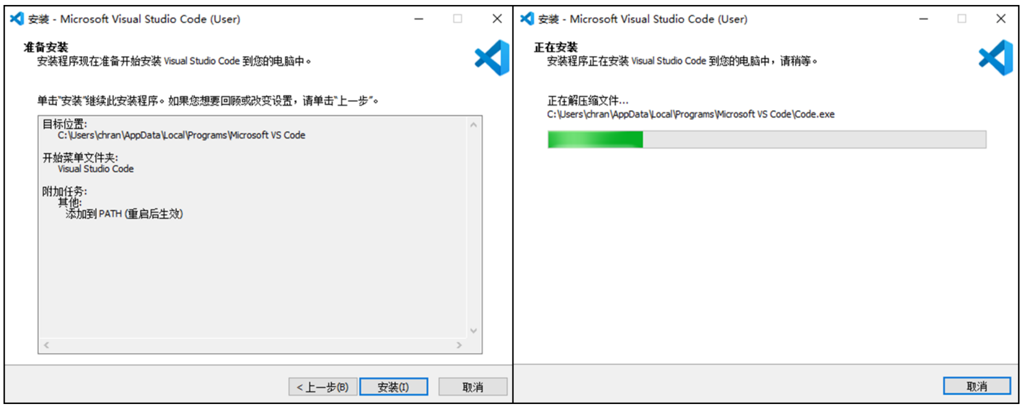
安装 - Microsoft Visual Studio Code (User)

选择附加任务
您想要安装程序执行哪些附加任务？

选择您想要安装程序在安装 Visual Studio Code 时执行的附加任务，然后单击"下一步"。

附加快捷方式:
☐ 创建桌面快捷方式(D)
其他:
☐ 将"通过 Code 打开"操作添加到 Windows 资源管理器文件上下文菜单
☐ 将"通过 Code 打开"操作添加到 Windows 资源管理器目录上下文菜单
☐ 将 Code 注册为受支持的文件类型的编辑器
☑ 添加到 PATH (重启后生效)

Figure 2-29. Vscode installation process 3



安装 - Microsoft Visual Studio Code (User)

准备安装
安装程序现在准备开始安装 Visual Studio Code 到您的电脑中。

单击"安装"继续此安装程序。如果您想要回顾或改变设置，请单击"上一步"。

目标位置:
C:\Users\chran\AppData\Local\Programs\Microsoft VS Code

开始菜单文件夹:
Visual Studio Code

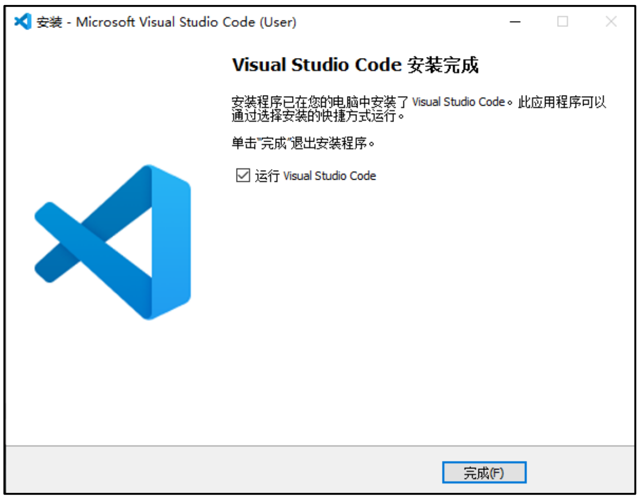
附加任务:
其他:
添加到 PATH (重启后生效)

安装 - Microsoft Visual Studio Code (User)

正在安装
安装程序正在安装 Visual Studio Code 到您的电脑中，请稍等。

正在解压缩文件...
C:\Users\chran\AppData\Local\Programs\Microsoft VS Code\Code.exe

Figure 2-30. Vscode installation process 4


安装 - Microsoft Visual Studio Code (User)

Visual Studio Code 安装完成

安装程序已在您的电脑中安装了 Visual Studio Code。此应用程序可以通过选择安装的快捷方式运行。

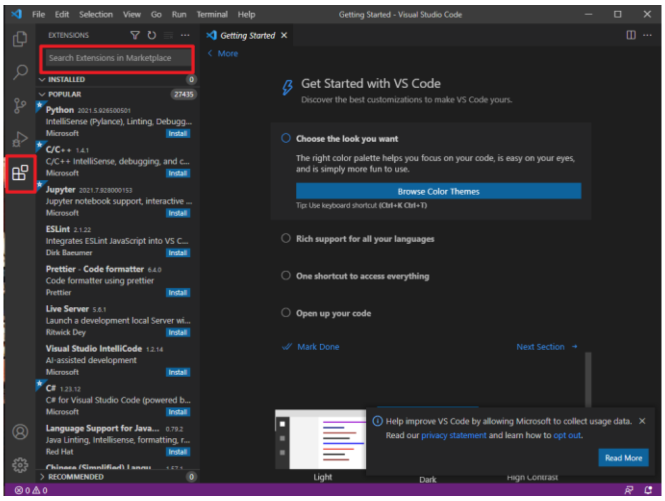
单击"完成"退出安装程序。

☑ 运行 Visual Studio Code

In order to meet the development requirements, some plugins should also be installed, open the installer Vscode, select the plug-in center on the left, search for and install the following plug-ins respectively, and click to install. The installation plug-ins and version numbers are as follows:

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

Figure 2-31. Vscode plugin installation search interface



Python 2021.5.926500501
IntelliSense (Pylance), Linting, Debugg...
Microsoft Install

C/C++ 1.4.1
C/C++ IntelliSense, debugging, and c...
Microsoft Install

Jupyter 2021.7.928000153
Jupyter notebook support, interactive ...
Microsoft Install

ESLint 2.1.22
Integrates ESLint JavaScript into VS C...
Dirk Baeumer Install

Prettier - Code formatter 6.4.0
Code formatter using prettier
Prettier Install

Live Server 5.6.1
Launch a development local Server wi...
Ritwick Dey Install

Visual Studio IntelliCode 1.2.14
AI-assisted development
Microsoft Install

C# 1.23.12
C# for Visual Studio Code (powered b...
Microsoft Install

Language Support for Java... 0.79.2
Java Linting, Intellisense, formatting, r...
Red Hat Install

Chinese (Simplified) Langu... 1.57.1

Get Started with VS Code
Discover the best customizations to make VS Code yours.

Choose the look you want
The right color palette helps you focus on your code, is easy on your eyes, and is simply more fun to use.
Tip: Use keyboard shortcut (Ctrl+K Ctrl+T)

Rich support for all your languages
One shortcut to access everything
Open up your code

Help improve VS Code by allowing Microsoft to collect usage data.
Read our privacy statement and learn how to opt out.

2.5. Install Openocd

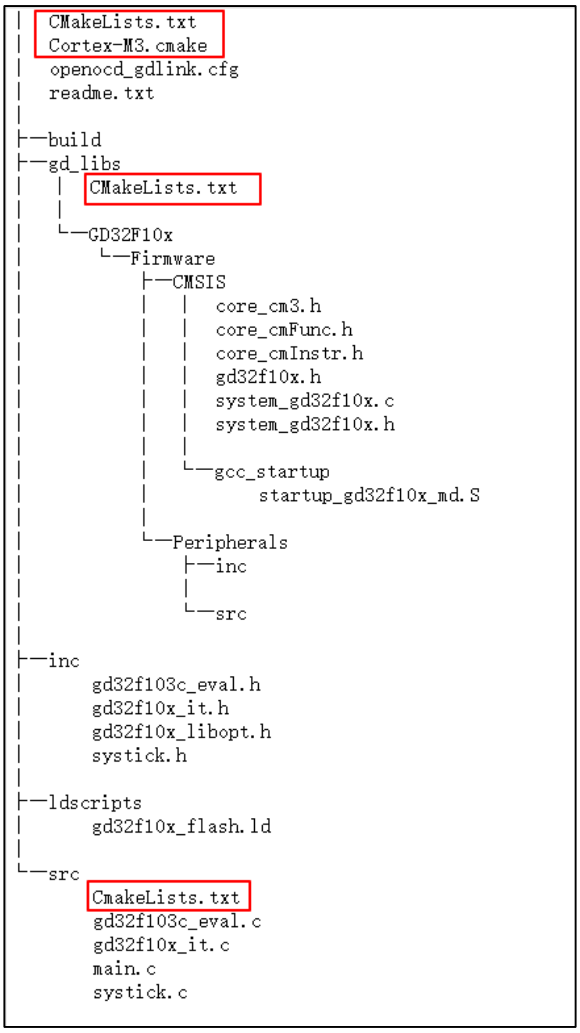
Download link of Openocd that supports GD32MCU:[https://github.com/GigaDevice-Semiconductor/openocd.](https://github.com/GigaDevice-Semiconductor/openocd)

After the user completes the compilation of Openocd according to the requirements, the executable file openocd.exe is generated, and the file is added to the environment variable. For the method of adding the environment variable, please refer to ***Install C/C++ MinGW compiler.***

3. CmakeLists file writing

The file organization structure of this application manual is shown in ***Figure 3-1. File organization chart***, and the contents of each folder are introduced below:

Figure 3-1. File organization chart



```
│  CMakeLists.txt
│  Cortex-M3.cmake
│  openocd_gdlink.cfg
│  readme.txt
│
├─build
├─gd_libs
│  │  CMakeLists.txt
│  │
│  └─GD32F10x
│      └─Firmware
│          ├─CMSIS
│          │  │  core_cm3.h
│          │  │  core_cmFunc.h
│          │  │  core_cmInstr.h
│          │  │  gd32f10x.h
│          │  │  system_gd32f10x.c
│          │  │  system_gd32f10x.h
│          │  │
│          │  └─gcc_startup
│          │          startup_gd32f10x_md.S
│          │
│          └─Peripherals
│              ├─inc
│              │
│              └─src
│
├─inc
│      gd32f103c_eval.h
│      gd32f10x_it.h
│      gd32f10x_libopt.h
│      systick.h
│
├─ldscripts
│      gd32f10x_flash.ld
│
└─src
        CmakeLists.txt
        gd32f103c_eval.c
        gd32f10x_it.c
        main.c
        systick.c
```

In the build folder, the intermediate files, library files and final executable files generated during the compilation process are stored;

In the gd_libs folder, store GD library files, mainly including peripheral library files, startup files and some header files;

In the inc folder, store the header files of the code written by the user and the header files related to the development board;

In the ldscripts folder, store the link script file;

In the src folder, the .c files written by the user and the .c files related to the development board are stored.

Among them, the CMakeLists.txt file is included in the root directory, gd_libs and src folders, and the Cortex-M3.cmake file is included in the root directory. The content of CmakeLists.txt in each file is introduced below.

3.1. CMakeLists.txt and Cortex-M3.cmake files in the root directory

The content of the CMakeLists.txt file in the root directory is shown in ***Table 3-1. CMakeLists.txt code in the root directory***.

Table 3-1. CMakeLists.txt code in the root directory

```
# set the minimum supported version of CMake
cmake_minimum_required(VERSION 3.17)
SET(PRJ_NAME "GD32F10x")
# define project name
project(${PRJ_NAME})
#include Cortex-M3.cmake
include(Cortex-M3.cmake)
# add directories gd_libs and src
add_subdirectory(gd_libs)
add_subdirectory(src)
```

The content of the Cortex-M3.cmake file is shown in ***Table 3-2. Cortex-M3.cmake code in the root directory***. Users can modify the compilation options and add related macro definitions by modifying the parameters in the file.

Table 3-2. Cortex-M3.cmake code in the root directory

```
# set the minimum supported version of CMake
cmake_minimum_required(VERSION 3.17)
#Cmake cross compilation configuration
SET(CMAKE_SYSTEM_NAME Generic)
#setup supportASM
ENABLE_LANGUAGE(ASM)
#debug mode
SET(CMAKE_BUILD_TYPE "Debug")
#release mode
#SET(CMAKE_BUILD_TYPE "Release")
# set up C compilation tools
SET(CMAKE_C_COMPILER arm-none-eabi-gcc)
# ELF to bin and hex file tool
SET(CMAKE_OBJCOPY arm-none-eabi-objcopy)
# file size tool
SET(CMAKE_SIZE arm-none-eabi-size)
# set floating point options
```

```
SET(MCU_FLAGS "-mcpu=cortex-m3 -mfloat-abi=softfp -mfpu=fpv4-sp-d16")
# set warning related information
SET(CMAKE_C_FLAGS "${MCU_FLAGS} -w -Wno-unknown-pragmas")
# set debugging options
SET(CMAKE_C_FLAGS_DEBUG "-O0 -g2 -ggdb")
SET(CMAKE_C_FLAGS_RELEASE "-O3")
# add macro definition
ADD_DEFINITIONS(
        -DGD32F10X_HD
        -DUSE_STDPERIPH_DRIVER

)
```

3.2. CMakeLists.txt file in the gd_libs folder

The content of the CMakeLists.txt file in gd_libs is shown in ***Table 3-3. CMakeLists.txt code in gd_libs***. This file mainly generates of the gd32_lib library from the GD peripheral firmware library file and CMSIS related files and startup files, and specifies the location of the generated file.

Table 3-3. CMakeLists.txt code in gd_libs

```
# set relevant path variables
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)
# add header file search path
include_directories(
        ${CORE_SUPPORT_DIR}
        ${PERIPHERALS_DIR}/inc
        ${PROJECT_SOURCE_DIR}/inc
)
# set startup file variables
SET(START_UP_ASM startup_gd32f10x_md.S)
# set the C properties of the startup file
set_property(SOURCE ${START_UP_DIR}/${START_UP_ASM} PROPERTY LANGUAGE C)
# The GLOB option will generate a file list for all files matching the query expression, and store the list
in the STD_LIB, SRC_CORE defined by the variable
file(GLOB STD_LIB ${PERIPHERALS_DIR}/src/*.c)
file(GLOB SRC_CORE ${CORE_SUPPORT_DIR}/*.c)
# generate library target gd32_lib
add_library(gd32_lib
        ${STD_LIB}
        ${SRC_CORE}
        ${START_UP_DIR}/${START_UP_ASM}
)
# set the name of the library output
set_target_properties(gd32_lib PROPERTIES OUTPUT_NAME "gd32_lib")
# set the default output path of library files
SET(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/build/lib)
```

3.3. CMakeLists.txt file in the src folder

The content of the CMakeLists.txt file in the src folder is shown in ***Table 3-4. CMakeLists.txt code in src***. This file mainly realizes linking and compiling the .c file and gd32_lib library file written by the user to generate an executable file and specify the location of the generated executable file.

Table 3-4. CMakeLists.txt code in src

```
# all .c files in this path are defined as SRC_LIST
aux_source_directory(. SRC_LIST)
# add header file search path
include_directories(
        ${PROJECT_SOURCE_DIR}/gd_libs/GD32F10x/Firmware/CMSIS
        ${PROJECT_SOURCE_DIR}/gd_libs/GD32F10x/Firmware/Peripherals/inc
        ${PROJECT_SOURCE_DIR}/inc
)
# add non-standard shared library search path
link_directories(${PROJECT_SOURCE_DIR}/build/lib)
# set the relative path and variables of the linked file
SET(FLASH_LD_DIR ${PROJECT_SOURCE_DIR}/ldscripts)
SET(FLASH_LD_FILE gd32f10x_flash.ld)
SET(LINKER_SCRIPT ${FLASH_LD_DIR}/${FLASH_LD_FILE})
# set link options
SET(CMAKE_EXE_LINKER_FLAGS
"--specs=nano.specs          -specs=nosys.specs          -T${LINKER_SCRIPT}          -Wl,-
Map=${PROJECT_BINARY_DIR}/${PRJ_NAME}.map,--cref -Wl,--gc-sections")
# generate object file
add_executable(${PRJ_NAME}.elf ${SRC_LIST})
# link the object file with the library file
target_link_libraries(${PRJ_NAME}.elf gd32_lib)
# set executable file output path
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/build/bin)
# set ELF conversion path
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)
```

```
# add custom commands to realize ELF conversion of hex and bin files
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. Compile, download and debug

4.1. Compile and download

Use vscode to open the project directory, a .vscode folder will be generated in the directory, and create tasks.json file launch.json files in the folder. The tasks.json file mainly realizes compilation and download function through buttons instead of command lines. The launch.json file is mainly debugging configuration file. The specific file content and comments of the two files are shown in ***Table 4-1. tasks.json code*** and ***Table 4-2. launch.json code***.

Table 4-1. tasks.json code

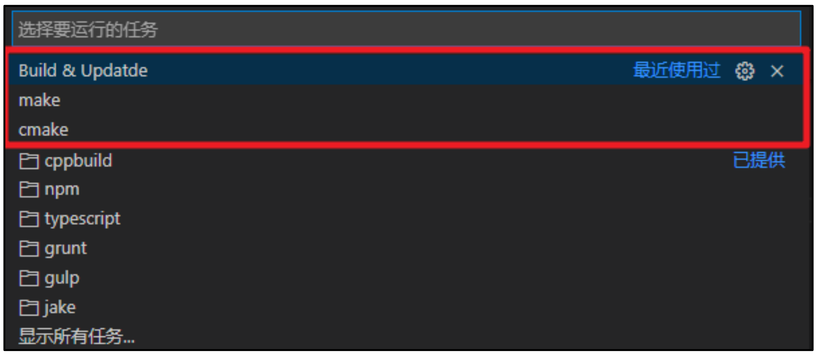
```
{
    "version": "2.0.0",
      // specify the path where the command is executed
    "options": {
        "cwd": "${workspaceRoot}/build"
    }
    "tasks": [
        {
        // execute cmake command to generate makefile
            "type": "shell",
            "label": "cmake",
            "command": "cmake",
            "args": [
                "-G",
                "MinGW Makefiles",
                ".."
            ]
        },
        {
        // execute the make command to generate an executable file
            "label": "make",
            "type": "shell",
            "command": "make",
            "args": [],
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "dependsOn": [
              "cmake"
            ],
                        "problemMatcher": []
                },
                {
                        // execute the openocd command to download the executable file to the target MCU
                        "type": "shell",
                        "label": "Build & Updatde",
                        "command": "openocd",
                        "args": [
                                "-f",
                        //absolute path of configuration file
                                "E:/Work_Code/10.cmake/Cmake/Code/Example-
windows/gd32f103C_example/openocd_gdlink_gd32f10x.cfg",
                                "-c",
                        // the absolute path of the compiled executable file
                                "program E:/Work_Code/10.cmake/Cmake/Code/Example-
windows/gd32f103C_example/build/bin/GD32F10x.elf verify reset exit"
                        ],
                        "group": "build",
                        "dependsOn": "make"
                }
        ]

}
```

Table 4-2. launch.json code

```
{
        // 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": [
                {
                        // Specify the path where the command is executed
                        "cwd": "${workspaceRoot}",
                        // Path to executable file
                        "executable": "build/bin/GD32F10x.elf",
                        "name": "Debug Microcontroller",
                        "request": "launch",
                        "type": "cortex-debug",
                        // the path where the svd file is located
                        "svdFile": "./GD32F10x_MD.svd",
                        // openocd command
                        "servertype": "openocd",
                        // the relative path where the openocd configuration file is located
                "configFiles": [
                    "./openocd_gdlink_gd32f10x.cfg ",
                ]
            }

    ]
}
```

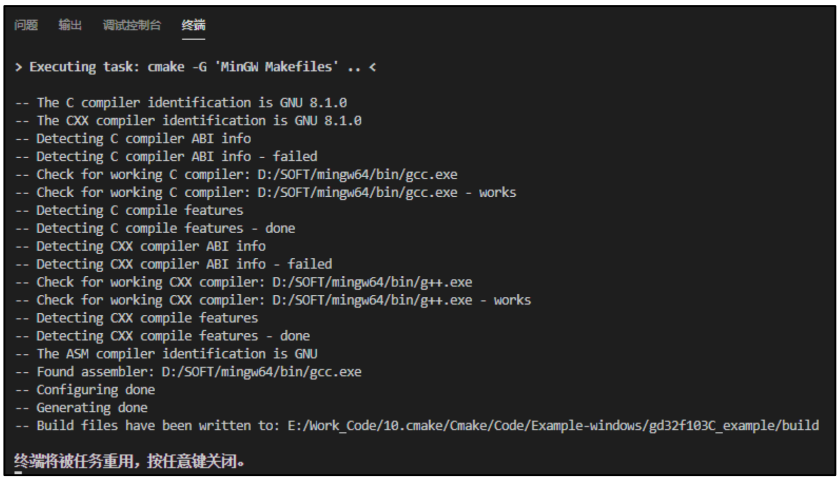
Click "Terminal -> Run Task" in VScode, you can see the following options

Figure 4-1. Running tasks in Vscode



Select the cmake option, execute the cmake command and generate the makefile file. Vscode terminal will prints makefile build process information.

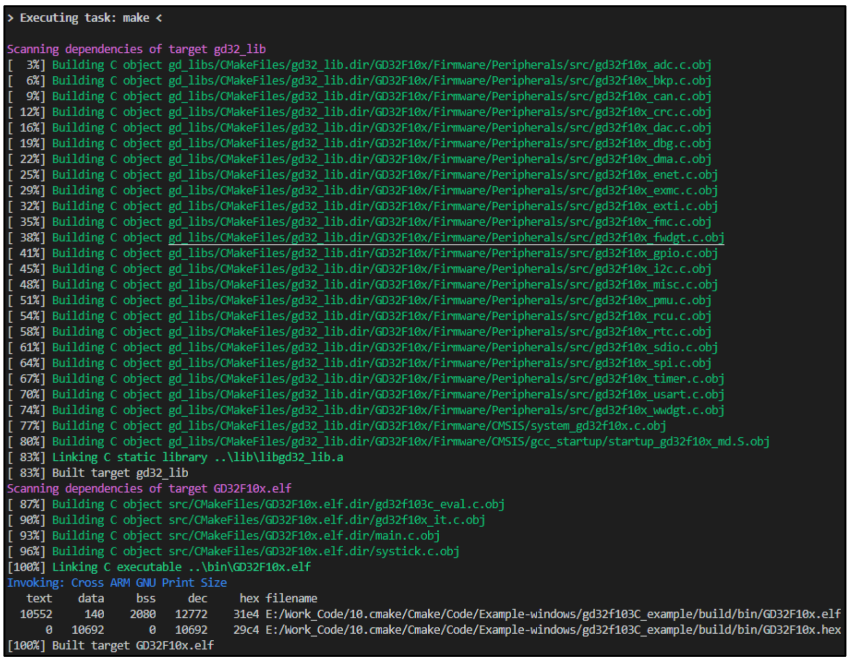
Figure 4-2. The build process of Cmake to generate makefile



```
> Executing task: cmake -G 'MinGW Makefiles' .. <

-- The C compiler identification is GNU 8.1.0
-- The CXX compiler identification is GNU 8.1.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - failed
-- Check for working C compiler: D:/SOFT/mingw64/bin/gcc.exe
-- Check for working C compiler: D:/SOFT/mingw64/bin/gcc.exe - works
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - failed
-- Check for working CXX compiler: D:/SOFT/mingw64/bin/g++.exe
-- Check for working CXX compiler: D:/SOFT/mingw64/bin/g++.exe - works
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- The ASM compiler identification is GNU
-- Found assembler: D:/SOFT/mingw64/bin/gcc.exe
-- Configuring done
-- Generating done
-- Build files have been written to: E:/Work_Code/10.cmake/Cmake/Code/Example-windows/gd32f103C_example/build

终端将被任务重用，按任意键关闭。
```

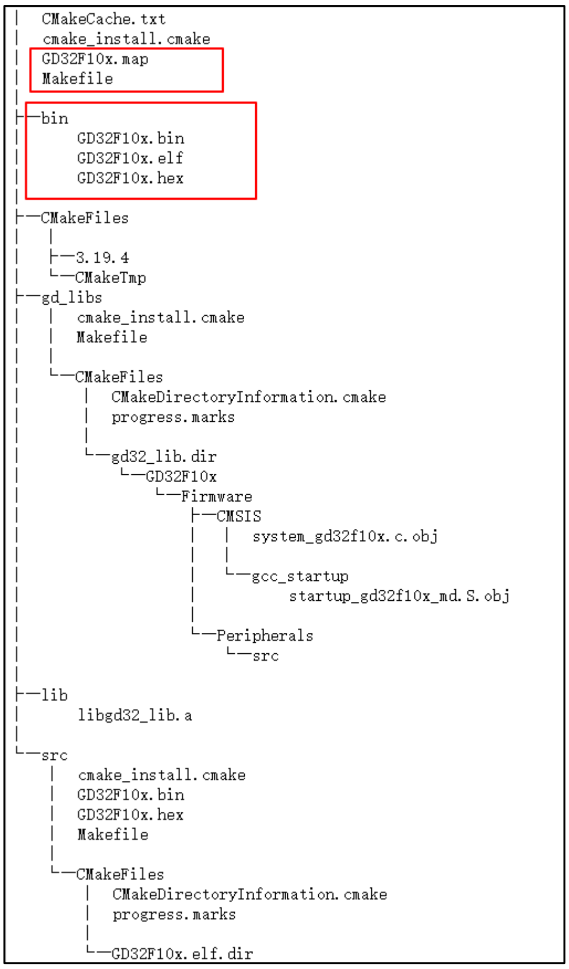
Select the make option, execute the make command, realize the compilation function, and generate executable files .elf, .bin and .hex files. The vscode terminal will print the executable file generation process information. At the same time, it can be seen from the build directory that the generated elf, hex and bin files are stored in the bin directory.

Figure 4-3. Build process of makefile to generate executable file



```
> 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_exmc.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_fmc.c.obj
[ 38%] Building C object gd_libs/CMakeFiles/gd32_lib.dir/GD32F10x/Firmware/Peripherals/src/gd32f10x_fwdgt.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_wwdgt.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
  10552     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
```

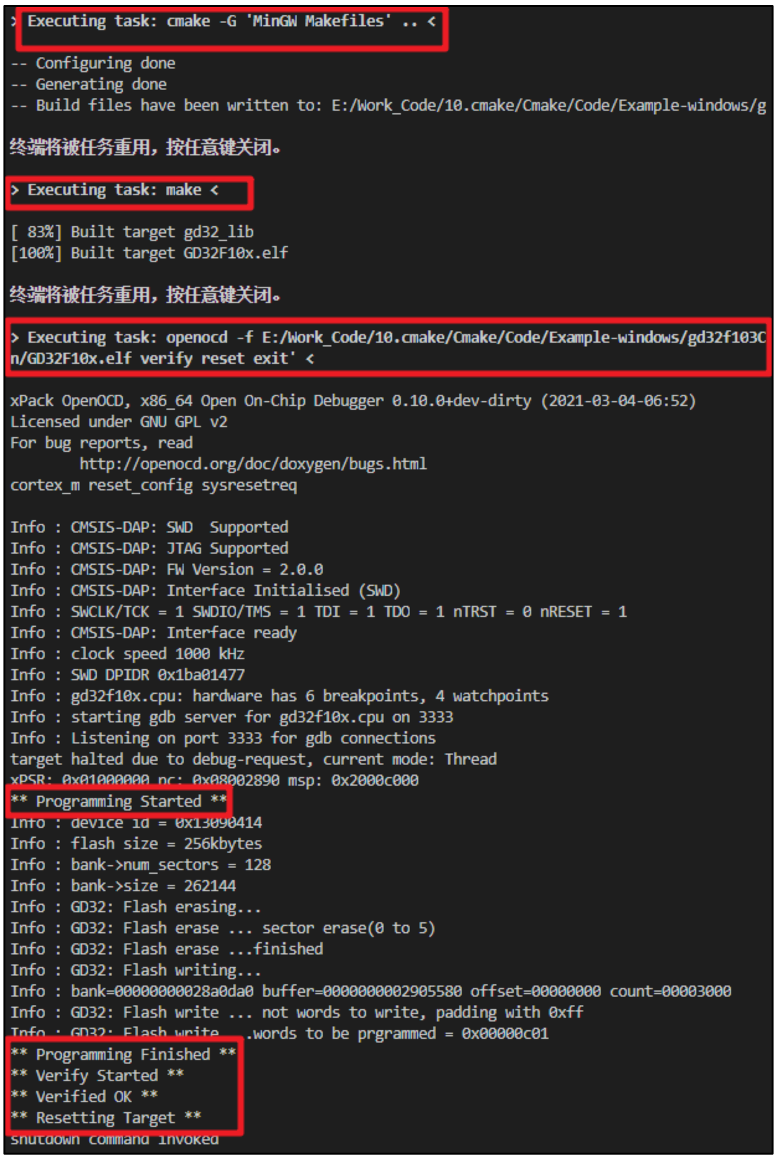
Figure 4-4. File organization structure in the build directory



```
│  CMakeCache.txt
│  cmake_install.cmake
│  GD32F10x.map
│  Makefile
│
├─bin
│      GD32F10x.bin
│      GD32F10x.elf
│      GD32F10x.hex
│
├─CMakeFiles
│  │
│  ├─3.19.4
│  └─CMakeTmp
├─gd_libs
│  │  cmake_install.cmake
│  │  Makefile
│  │
│  └─CMakeFiles
│      │  CMakeDirectoryInformation.cmake
│      │  progress.marks
│      │
│      └─gd32_lib.dir
│          └─GD32F10x
│              └─Firmware
│                  ├─CMSIS
│                  │  │  system_gd32f10x.c.obj
│                  │  │
│                  │  └─gcc_startup
│                  │          startup_gd32f10x_md.S.obj
│                  │
│                  └─Peripherals
│                      └─src
│
├─lib
│      libgd32_lib.a
│
└─src
    │  cmake_install.cmake
    │  GD32F10x.bin
    │  GD32F10x.hex
    │  Makefile
    │
    └─CMakeFiles
        │  CMakeDirectoryInformation.cmake
        │  progress.marks
        │
        └─GD32F10x.elf.dir
```

The option Build&Updata realizes one-click cmake build, compile and openocd download functions. The vscode terminal will print information about the build, compile and program download process.

Figure 4-5. One-click compilation and download process

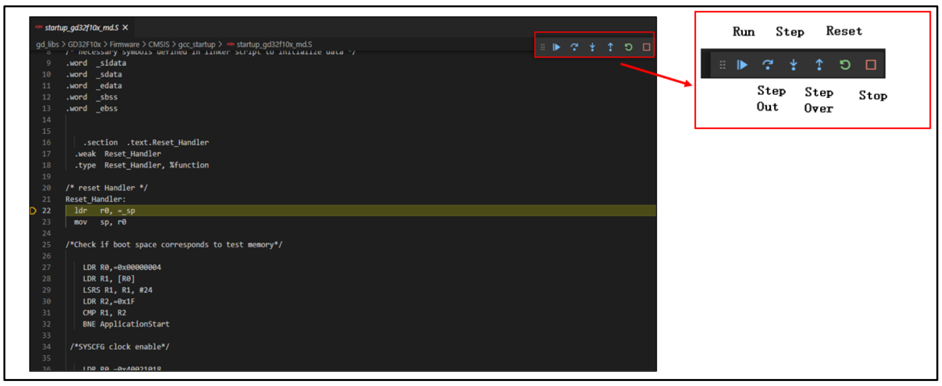


```
> Executing task: cmake -G 'MinGW Makefiles' .. <

-- 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=00000000028a0da0 buffer=0000000002905580 offset=00000000 count=00003000
Info : GD32: Flash write ... not words to write, padding with 0xff
Info : GD32: Flash write ...words to be prgrammed = 0x00000c01
** Programming Finished **
** Verify Started **
** Verified OK **
** Resetting Target **
shutdown command invoked
```

4.2. Debug

Click "Run -> Start Debugging" in VScode to enter the debugging interface as shown in *Figure 4-6. Vscode Debug interface*. Run, single step, reset and terminate the program debugging in the upper right corner of the interface.

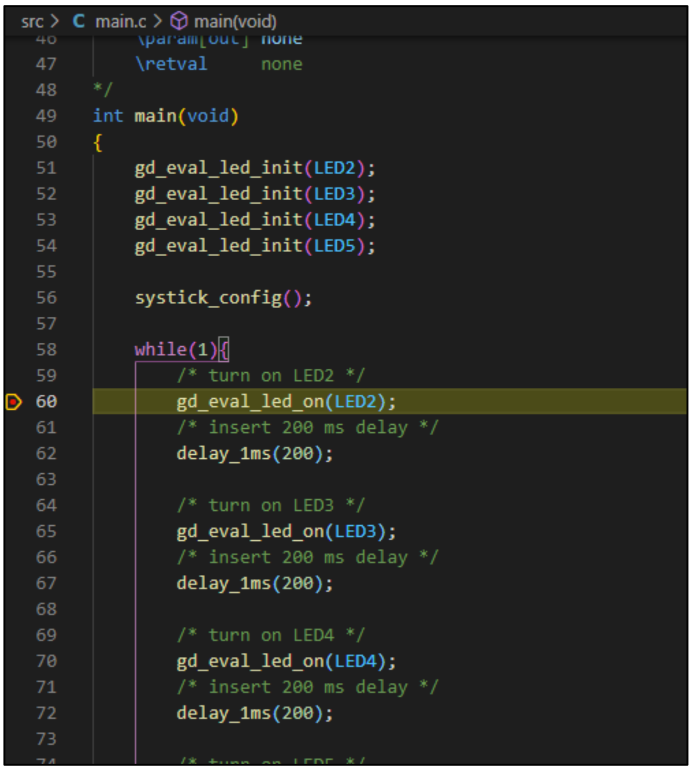
Figure 4-6. Vscode Debug interface


startup_gd32f10x_md.S
gd_libs > GD32F10x > Firmware > CMSIS > gcc_startup > startup_gd32f10x_md.S
Run
Step
Reset
Step Out
Step Over
Stop

```
/* necessary symbols defined in linker script to initialize data */
.word  _sidata
.word  _sdata
.word  _edata
.word  _sbss
.word  _ebss


    .section  .text.Reset_Handler
  .weak  Reset_Handler
  .type  Reset_Handler, %function

/* reset Handler */
Reset_Handler:
  ldr   r0, =_sp
  mov   sp, r0

/*Check if boot space corresponds to test memory*/

    LDR R0,=0x00000004
    LDR R1, [R0]
    LSRS R1, R1, #24
    LDR R2,=0x1F
    CMP R1, R2
    BNE ApplicationStart

 /*SYSCFG clock enable*/

    LDR R0,=0x40021018
```

Add a breakpoint on the left side of the code, and execute Run to run to the breakpoint.

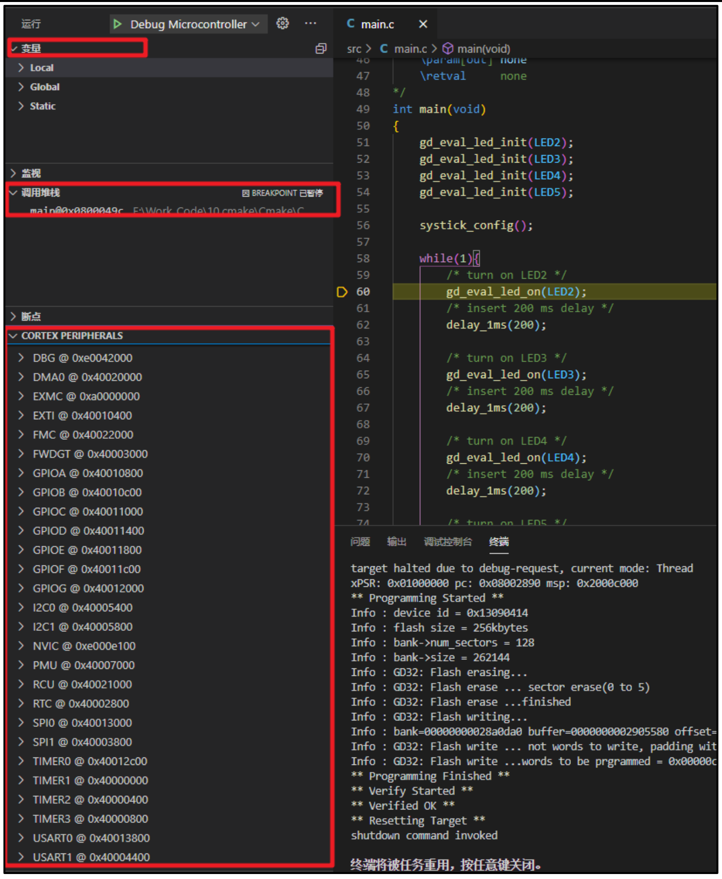
Figure 4-7. Add a breakpoint in the debugging interface and run



```
    \param[out] none
    \retval     none
*/
int main(void)
{
    gd_eval_led_init(LED2);
    gd_eval_led_init(LED3);
    gd_eval_led_init(LED4);
    gd_eval_led_init(LED5);

    systick_config();

    while(1){
        /* turn on LED2 */
        gd_eval_led_on(LED2);
        /* insert 200 ms delay */
        delay_1ms(200);

        /* turn on LED3 */
        gd_eval_led_on(LED3);
        /* insert 200 ms delay */
        delay_1ms(200);

        /* turn on LED4 */
        gd_eval_led_on(LED4);
        /* insert 200 ms delay */
        delay_1ms(200);

        /* turn on LED5 */
```

View the peripheral register value on the left boundary surface, and add variables to view and other debugging operations.

Figure 4-8. View peripheral registers and variable values



```
DBG @ 0xe0042000
DMA0 @ 0x40020000
EXMC @ 0xa0000000
EXTI @ 0x40010400
FMC @ 0x40022000
FWDGT @ 0x40003000
GPIOA @ 0x40010800
GPIOB @ 0x40010c00
GPIOC @ 0x40011000
GPIOD @ 0x40011400
GPIOE @ 0x40011800
GPIOF @ 0x40011c00
GPIOG @ 0x40012000
I2C0 @ 0x40005400
I2C1 @ 0x40005800
NVIC @ 0xe000e100
PMU @ 0x40007000
RCU @ 0x40021000
RTC @ 0x40002800
SPI0 @ 0x40013000
SPI1 @ 0x40003800
TIMER0 @ 0x40012c00
TIMER1 @ 0x40000000
TIMER2 @ 0x40000400
TIMER3 @ 0x40000800
USART0 @ 0x40013800
USART1 @ 0x40004400
```

```c
    \param[out] none
    \retval     none
*/
int main(void)
{
    gd_eval_led_init(LED2);
    gd_eval_led_init(LED3);
    gd_eval_led_init(LED4);
    gd_eval_led_init(LED5);

    systick_config();

    while(1){
        /* turn on LED2 */
        gd_eval_led_on(LED2);
        /* insert 200 ms delay */
        delay_1ms(200);

        /* turn on LED3 */
        gd_eval_led_on(LED3);
        /* insert 200 ms delay */
        delay_1ms(200);

        /* turn on LED4 */
        gd_eval_led_on(LED4);
        /* insert 200 ms delay */
        delay_1ms(200);

        /* turn on LED5 */
```

```
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=00000000028a0da0 buffer=0000000002905580 offset=
Info : GD32: Flash write ... not words to write, padding wit
Info : GD32: Flash write ...words to be prgrammed = 0x00000c
** Programming Finished **
** Verify Started **
** Verified OK **
** Resetting Target **
shutdown command invoked

终端将被任务重用，按任意键关闭。
```

5. Revision history

Table 5-1. Revision history

Revision No	Description	Date
1.0	Initial Release	June.30, 2021