GigaDevice Semiconductor Inc.

GD32VW553 Quick Development Guide

Application Note

AN154

Revision 1.3a

(May 2025)

Table of Contents

List of Figures

List of Tables

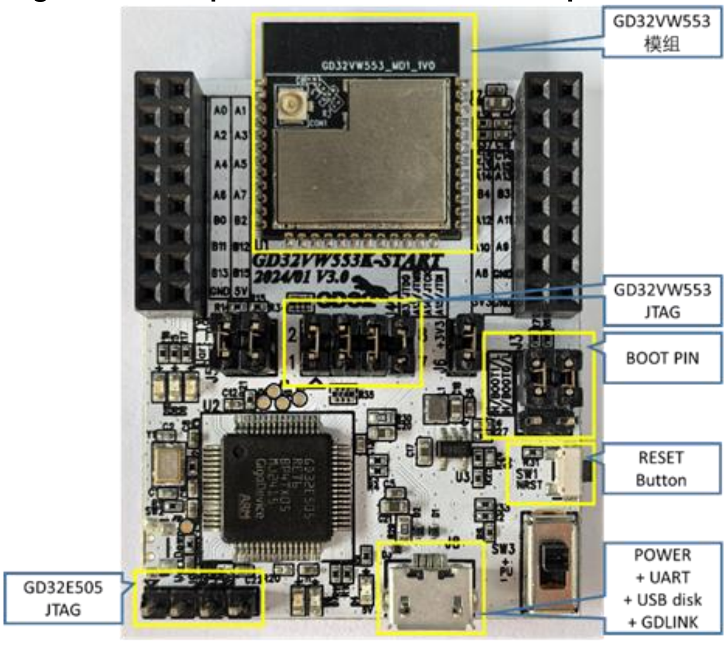
1. Introduction to development board

1.1. Picture of real development board

1.1.1. The START development board

The START development board consists of a baseboard and a module equipped with the GD32VW55x Wi-Fi+BLE chip.

Figure 1-1. The picture of the START development board



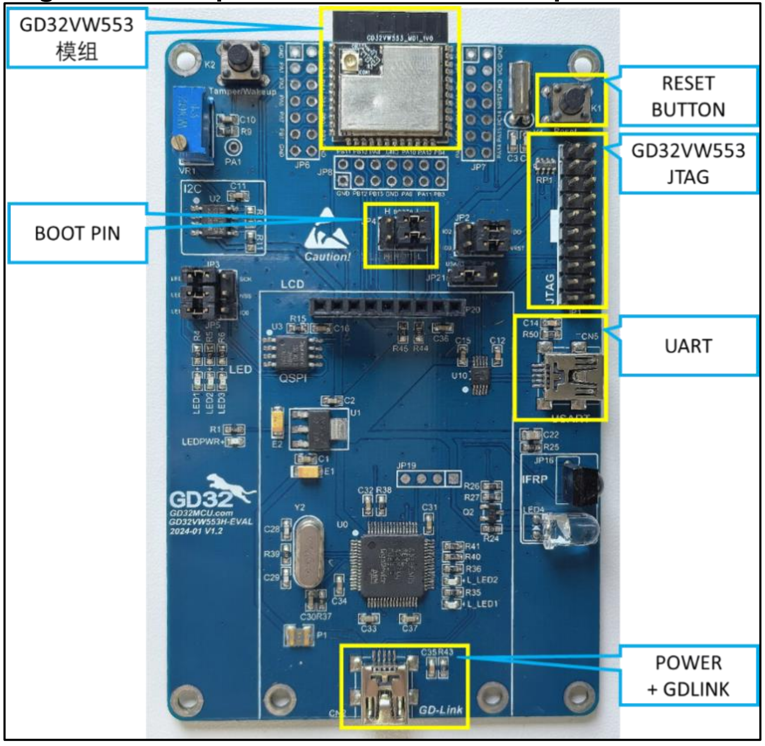
Mainly focus on the following parts of the development board, which have been marked in the *Figure 1-1. The picture of the START development board*.

- Boot mode (Boot PIN);
- Power supply port (POWER);
- View log (UART);
- Debugger interface (JLink, or GDLink);
- Reboot (Reset Button).

1.1.2. The EVAL development board

The EVAL development board consists of a baseboard and a module equipped with the GD32VW55x Wi-Fi+BLE chip. The baseboard lead out many peripheral test ports, such as I2C, IFRP, ADC and so on.

Figure 1-2. The picture of the EVAL development board

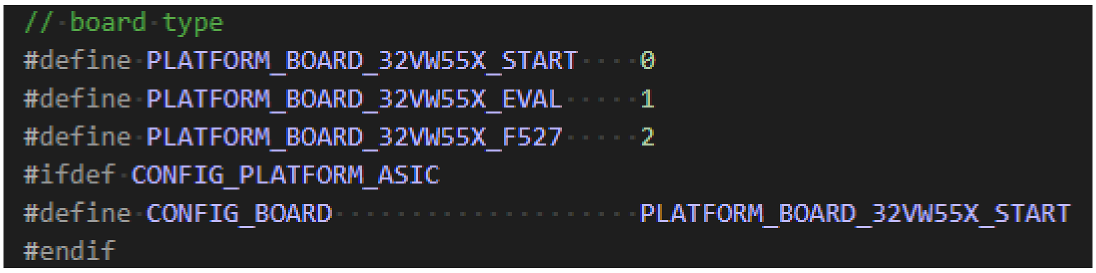


Developers mainly focus on the following parts of the development board, which have been marked in the [***Figure 1-2. The picture of the EVAL development board***].

- Boot mode (Boot PIN);
- Power supply port (power supply);
- View log (UART);
- Debugger interface (JLink, or GDLink);
- Reboot (Reset Button).

For the START development board and the EVAL development board, the SDK configuration is different and different macros need to be selected to enable them. As shown in ***Figure 1-3. Development Board Type Configuration***, the SDK selects the START development board configuration as the default. The configuration file is GD32VW55x_RELEASE/config/ platform_def.h.

Figure 1-3. Development Board Type Configuration



```
// board type
#define PLATFORM_BOARD_32VW55X_START    0
#define PLATFORM_BOARD_32VW55X_EVAL     1
#define PLATFORM_BOARD_32VW55X_F527     2
#ifdef CONFIG_PLATFORM_ASIC
#define CONFIG_BOARD                    PLATFORM_BOARD_32VW55X_START
#endif
```

1.2. Boot mode

GD32VW55x can boot from ROM, FLASH, or SRAM.

The level selection of the two pins BOOT0 and BOOT1 in the BOOT SWD box of the development board determines the boot mode. See ***Table 1-1. Boot mode***. For more instructions on the boot mode, please refer to the document "GD32VW55x_User_Manual".

Table 1-1. Boot mode

EFBOOTLK	BOOT0	BOOT1	EFSB	Boot address	Boot area
0	0	-	0	0x08000000	SIP Flash
0	0	-	1	0x0BF46000	secure boot
0	1	0	-	0x0BF40000	Bootloader/ROM
0	1	1	-	0x20000000	SRAM
1	0	-	0	0x08000000	SIP Flash
1	0	-	1	0x0BF46000	Secure boot
1	1	-	-	0x0BF40000	Bootloader/ROM

1.3. Debugger interface

For START development board, it comes with a GDLink(GD32E505) debugger that can be used with OpenOCD. Can also use an external debugger (GDLink or JLink) at the JTAG interface of the board for debugging and download. The GD32E505 chip also integrates the UART function, so only one USB cable is required to supply power, debug, and view the log. Connect the pins JCLK, JTWS, JTDO and JTDI to the middle four pins through jumper caps, and then download and debug the code through DAPLINK. ***Figure 1-1. The picture of the START development board*** shows how to debug through DAPLINK.

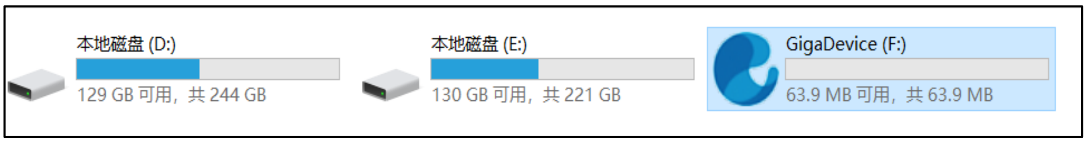
For EVAL development board, GDLink or JLink debugger can be used for debugging and download.

It should also be noted that the GD32VW55x supports cJTAG and JTAG but does not support the SWD debugging interface.

1.4. Download interface

For the START development board, in addition to using the GDLink debugger or JLink debugger mentioned in the previous section for firmware downloading, if debugging functionality is not required and only firmware downloading is needed, the firmware can also be downloaded using a USB drive copy method. Connect the development board to a computer via a USB cable, as shown in ***Figure 1-4. List of devices and drivers***, under the devices and drives list as the GigaDevice drive. Copy the "image-all.bin" file (refer to subsequent sections) into the GigaDevice drive to complete the FLASH programming of the GD32VW55x chip.

Figure 1-4. List of devices and drivers



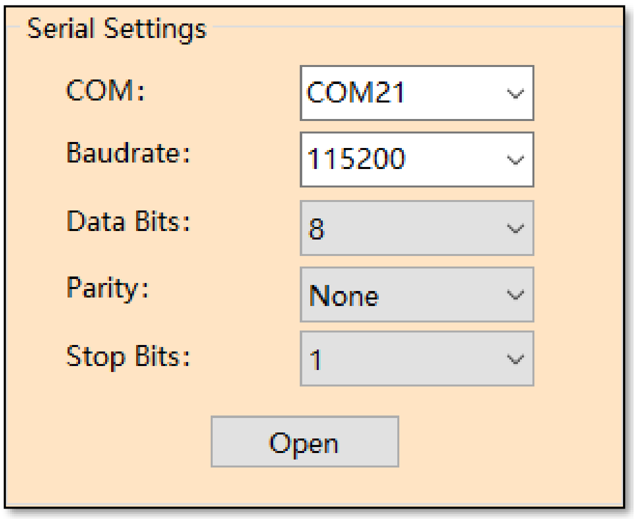
本地磁盘 (D:)
129 GB 可用，共 244 GB
本地磁盘 (E:)
130 GB 可用，共 221 GB
GigaDevice (F:)
63.9 MB 可用，共 63.9 MB

For EVAL development board, GDLink or JLink debugger can be used for download. Dragging into the USB disk is not supported.

1.5. Viewing log

Connect a MicroUSB cable to the START development board, use a serial port tool on the PC, and configure it according to the parameters in ***Figure 1-5. Configuration of serial port*** and connect to the board. After that, use the serial port to output logs.

Figure 1-5. Configuration of serial port



Serial Settings
COM: COM21
Baudrate: 115200
Data Bits: 8
Parity: None
Stop Bits: 1

2. Building development environment

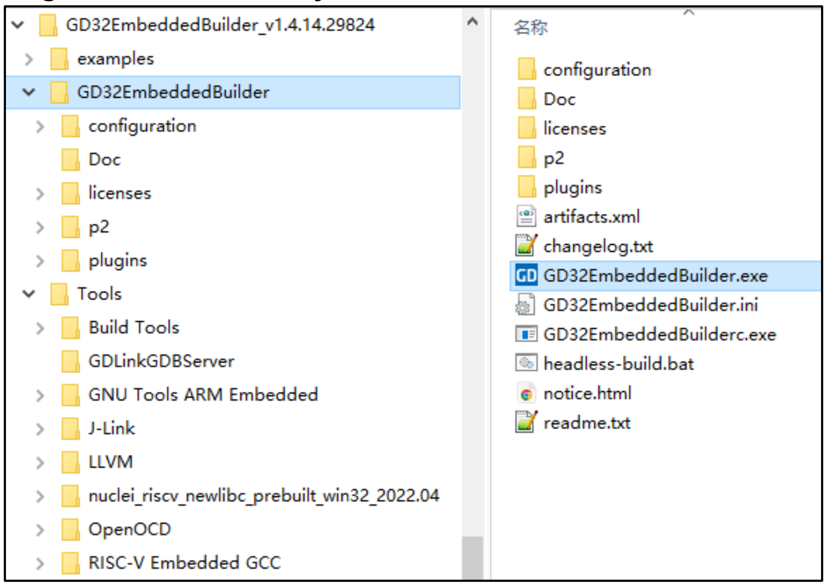
Build a development environment before compiling and downloading the firmware.

The development tool currently used is GD32 Embedded Builder and SEGGER Embedded Studio IDE.

2.1. Installation of GD32 Embedded Builder

The GD32 Embedded Builder can select GD32VW5 at website: ***https://gd32mcu.com/cn/download*** to download. The uncompress downloaded files is as ***Figure 2-1 The Directory Structure of GD32 Embedded Builder*** shows. The build tool, tool chain, openocd, jlink, and other related tools have all been placed in the Tools directory.

Figure 2-1 The Directory Structure of GD32 Embedded Builder



```
GD32EmbeddedBuilder_v1.4.14.29824
  examples
  GD32EmbeddedBuilder
    configuration
    Doc
    licenses
    p2
    plugins
  Tools
    Build Tools
    GDLinkGDBServer
    GNU Tools ARM Embedded
    J-Link
    LLVM
    nuclei_riscv_newlibc_prebuilt_win32_2022.04
    OpenOCD
    RISC-V Embedded GCC
```

```
名称
configuration
Doc
licenses
p2
plugins
artifacts.xml
changelog.txt
GD32EmbeddedBuilder.exe
GD32EmbeddedBuilder.ini
GD32EmbeddedBuilderc.exe
headless-build.bat
notice.html
readme.txt
```

2.2. Installation of SEGGER Embedded Studio IDE

Please visit the website: https://wiki.segger.com/GD32V for how to get the SEGGER Embedded Studio IDE and License Activation Key.

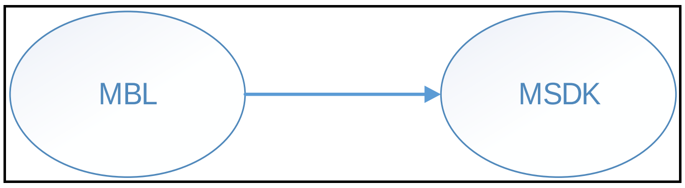
3. What developers must know

Before getting started with development, first understand the members of the SDK execution program group, how to correctly configure the SDK.

3.1. SDK execution program group

SDK will finally generate two main execution programs: MBL (Main Bootloader) and MSDK (Main SDK), which will eventually be downloaded to FLASH to run. After power-on, the programs will boot from Reset_Handler of MBL, and then jump to the MSDK main program to run, as shown in ***Figure 3-1. Boot process***.

Figure 3-1. Boot process

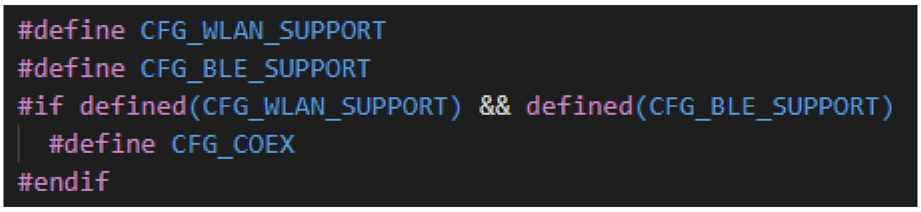


3.2. SDK configuration

3.2.1. Configuration of wireless module

The configuration file is GD32VW55x_RELEASE/config/platform_def.h, whose main content is as shown in ***Figure 3-2. Configuration of wireless module***.

Figure 3-2. Configuration of wireless module



```
#define CFG_WLAN_SUPPORT
#define CFG_BLE_SUPPORT
#if defined(CFG_WLAN_SUPPORT) && defined(CFG_BLE_SUPPORT)
  #define CFG_COEX
#endif
```

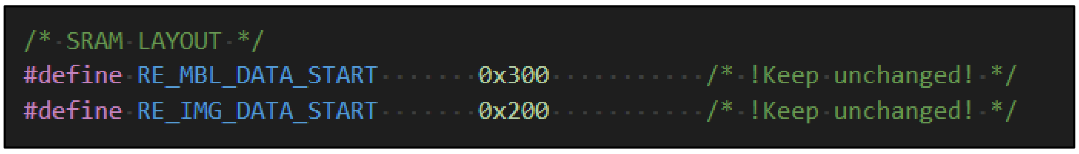
- In the case of BLE/ WiFi combo mode, please enable:
  - #define CFG_WLAN_SUPPORT
  - #define CFG_BLE_SUPPORT
- In the case of BLE only, please only enable:
  - #define CFG_BLE_SUPPORT
- In the case of WiFi only, please only enable:
  - #define CFG_WLAN_SUPPORT
- To disable the wireless module, please disable all

3.2.2. SRAM layout

The configuration file is GD32VW55x_RELEASE\config\config_gdm32.h. Modify the following macro definition (as ***Figure 3-3. SRAM layout*** shows) values to plan the SRAM space occupied by the executable program segments MBL and IMG. These values are offset addresses, and the base address is defined at the beginning of the file.

The line marked "!Keep unchanged!" cannot be modified; otherwise, the operation of the MbedTLS code in the ROM will be affected.

Figure 3-3. SRAM layout



```
/* SRAM LAYOUT */
#define RE_MBL_DATA_START       0x300           /* !Keep unchanged! */
#define RE_IMG_DATA_START       0x200           /* !Keep unchanged! */
```

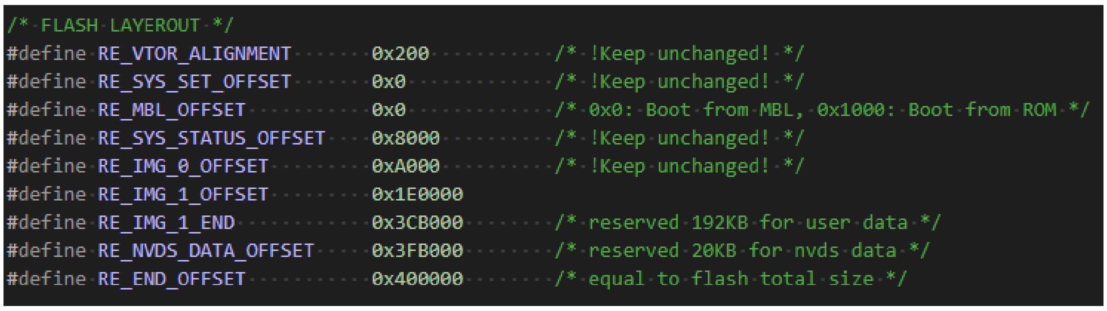
For the planning of SRAM space in each executable program segment, refer to the .ld file under the corresponding project, such as MBL\project\eclipse\mbl.ld and MSDK\plf\riscv\env\gd32w55x.ld.

3.2.3. FLASH layout

The configuration file is GD32VW55x_RELEASE\config\config_gdm32.h. Modify the following macro definition(as ***Figure 3-4. FLASH layout*** shows) values to plan the FLASH space occupied by the executable program segments MBL and MSDK. These values are offset addresses, and the base address is defined at the beginning of the file.

The line marked "!Keep unchanged!" cannot be modified; otherwise, the operation of the project will be affected.

Figure 3-4. FLASH layout



```
/* FLASH LAYEROUT */
#define RE_VTOR_ALIGNMENT      0x200           /* !Keep unchanged! */
#define RE_SYS_SET_OFFSET      0x0             /* !Keep unchanged! */
#define RE_MBL_OFFSET          0x0             /* 0x0: Boot from MBL, 0x1000: Boot from ROM */
#define RE_SYS_STATUS_OFFSET   0x8000          /* !Keep unchanged! */
#define RE_IMG_0_OFFSET        0xA000          /* !Keep unchanged! */
#define RE_IMG_1_OFFSET        0x1E0000
#define RE_IMG_1_END           0x3CB000        /* reserved 192KB for user data */
#define RE_NVDS_DATA_OFFSET    0x3FB000        /* reserved 20KB for nvds data */
#define RE_END_OFFSET          0x400000        /* equal to flash total size */
```

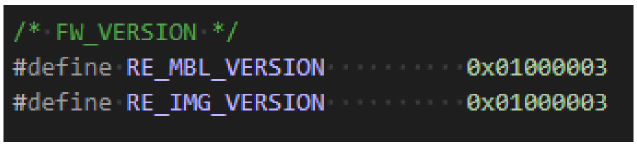
For the planning of FLASH space in each executable program segment, refer to the .ld file under the corresponding project, such as MBL\project\eclipse\mbl.ld and MSDK\plf\riscv\env\gd32w55x.ld.

3.2.4. Firmware version No.

The configuration file is GD32VW55x_RELEASE\config\config_gdm32.h. Modify the following macro definition values showed in *Figure 3-5. Firmware version No.* to specify the version No. In addition, the macro RE_IMG_VERSION is used in Securt Boot to determine the firmware version.

MBL only supports local upgrade, while IMG supports online upgrade. The version No. released by the SDK is consistent with RE_IMG_VERSION.

Figure 3-5. Firmware version No.



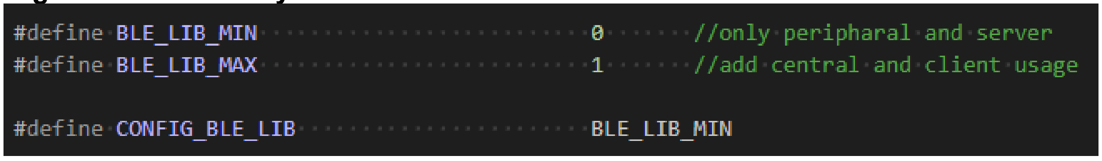
```
/* FW_VERSION */
#define RE_MBL_VERSION          0x01000003
#define RE_IMG_VERSION          0x01000003
```

3.2.5. APP configuration

The configuration file is GD32VW55x_RELEASE\MSDK\app\app_cfg.h. Choose whether to enable some applications, such as ATCMD, Alibaba Cloud, MQTT, COAP and so on.

By modifying the macro CONFIG_BLE_LIB in app_cfg.h, the BLE library can be switched. When CONFIG_BLE_LIB is set to BLE_LIB_MIN (as shown in *Figure 3-6 BLE library selection*), the project compilation will use libble.a, and the header file will include ble_config_min.h. When CONFIG_BLE_LIB is set to BLE_LIB_MAX, the project compilation will use libble_max.a, and the header file will include ble_config_max.h.

Figure 3-6 BLE library selection



```
#define BLE_LIB_MIN                         0      //only peripharal and server
#define BLE_LIB_MAX                         1      //add central and client usage

#define CONFIG_BLE_LIB                      BLE_LIB_MIN
```

The features supported by libble.a are as follows:

1. Supports peripheral
2. Supports a single connection link
3. Supports server
4. Supports host and controller
5. Supports EATT
6. Supports WeChat applet WiFi provisioning

Based on libble.a, libble_max.a additionally supports the following features:

1. Supports central
2. Supports four connection links
3. Supports client

4. Supports periodic advertising
5. Supports PHY updates
6. Supports power control
7. Supports BLE ping
8. Supports secure connection

3.2.6. Configuration Selection

The main project-MSDK, supports multiple configurations, with msdk selected by default. Additional options include msdk_ffd, msdk_mbedtls_2.17.0, msdk_rtthread, and msdk_threadx.

The main difference between msdk_ffd and msdk lies in the WiFi connection management library included in the project. The msdk includes libwpas, which is more streamlined and consumes fewer memory resources. The msdk_ffd includes wpa_supplicant, which is more comprehensive and general-purpose but has a larger codebase and consumes more memory resources. Additionally, msdk_ffd includes libble_max.a by default, enabling more BLE features. Of course, msdk can switch between libble.a and libble_max.a by modifying the configuration.

The main difference between msdk_mbedtls_2.17.0 and msdk lies in the version of the MbedTLS library included in the project. The msdk includes MbedTLS 3.6.2, which runs in FLASH. The msdk_mbedtls_2.17.0 includes MbedTLS 2.17.0, with most of its content running in ROM. If strict security requirements are needed, it is recommended to choose msdk. If FLASH space is limited, it is recommended to choose msdk_mbedtls_2.17.0

The main difference between msdk_rtthread and msdk lies in the RTOS used in the project. The msdk uses FreeRTOS. The msdk_rtthread uses RT-Thread.

The main difference between msdk_threadx and msdk also lies in the RTOS used in the project. The msdk uses FreeRTOS. The msdk_threadx uses ThreadX.

- msdk

  FreeRTOS + Libwpas.a + libble.a + MbesTLS 3.6.2

- msdk_ffd

  FreeRTOS + wpa_supplicant.a + libble_max.a + MbesTLS 3.6.2

- msdk_mbedtls_2.17.0

  FreeRTOS + Libwpas.a + libble.a + ROM MbesTLS 2.17.0

- msdk_rtthread

  RTThread + Libwpas.a + libble.a + MbesTLS 3.6.2

- msdk_threadx

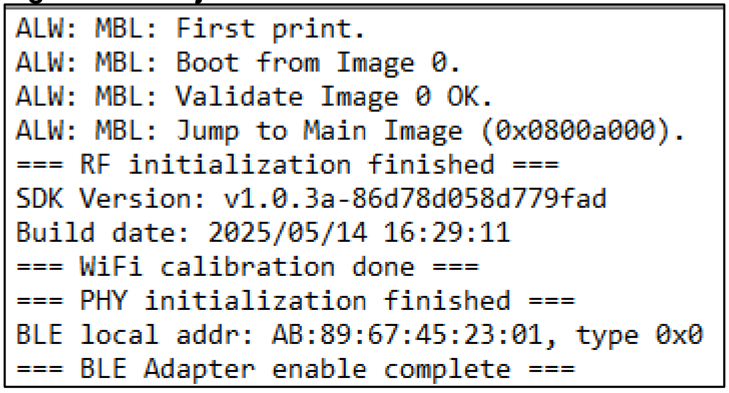
  Threadx + Libwpas.a + libble.a + MbesTLS 3.6.2

For details on how to make configuration selection for actual use, see the Compiling MSDK Projects section in subsection ***4.2Compilation***.

3.3. Correct log example

After the firmware group (MBL+MSDK) is successfully downloaded, open the serial port tool, and press the Reset button on the development board. The startup information is shown in ***Figure 3-7. Project boot information***. If an exception occurs, please check ***6FAQ***for help.

Figure 3-7. Project boot information



```
ALW: MBL: First print.
ALW: MBL: Boot from Image 0.
ALW: MBL: Validate Image 0 OK.
ALW: MBL: Jump to Main Image (0x0800a000).
=== RF initialization finished ===
SDK Version: v1.0.3a-86d78d058d779fad
Build date: 2025/05/14 16:29:11
=== WiFi calibration done ===
=== PHY initialization finished ===
BLE local addr: AB:89:67:45:23:01, type 0x0
=== BLE Adapter enable complete ===
```

4. GD32 Embedded Builder IDE project

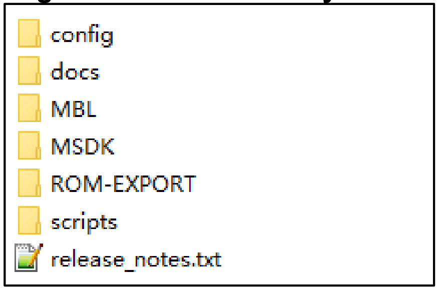
This chapter introduces how to compile and debug the SDK under Embedded Builder IDE.

The project group consists of two projects: MBL/MSDK. MSDK includes Wi-Fi protocol stack, BLE protocol stack, peripheral drivers, applications, etc. The MBL is mainly responsible for selecting the correct MSDK firmware from the two (current firmware and OTA firmware) to run.

4.1. Opening the project group

Check the SDK directory GD32VW55x_RELEASE, as shown in ***Figure 4-1. SDK directory***.

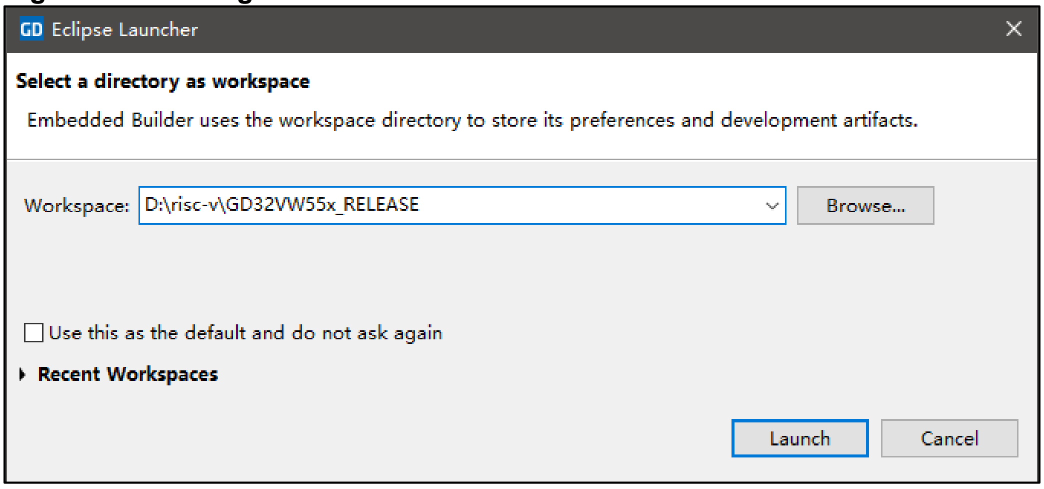
Figure 4-1. SDK directory



config
docs
MBL
MSDK
ROM-EXPORT
scripts
release_notes.txt

To start the IDE, double-click Embedded Builder.exe in the Embedded Builder directory, and select the SDK directory GD32VW55x_RELEASE as the workspace, and then click the launch button, as shown in ***Figure 4-2. Starting GD32 Embedded Builder IDE***.

Figure 4-2. Starting GD32 Embedded Builder IDE



Eclipse Launcher

Select a directory as workspace

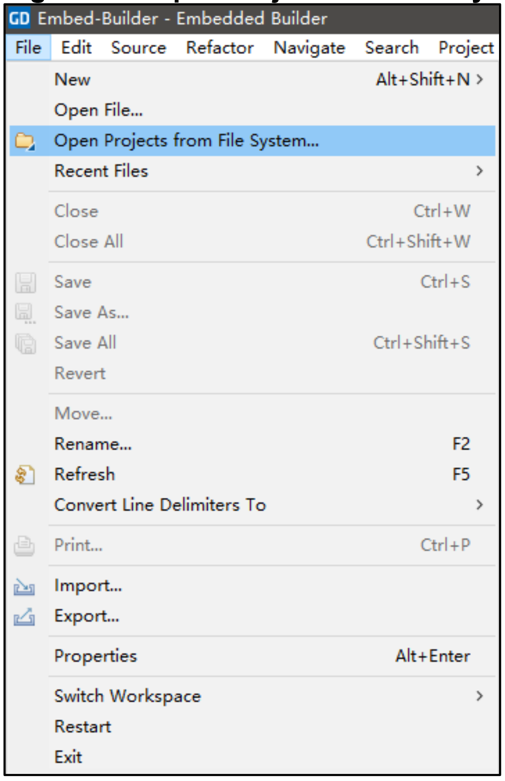
Embedded Builder uses the workspace directory to store its preferences and development artifacts.

Workspace: D:\risc-v\GD32VW55x_RELEASE

- Import the MBL project

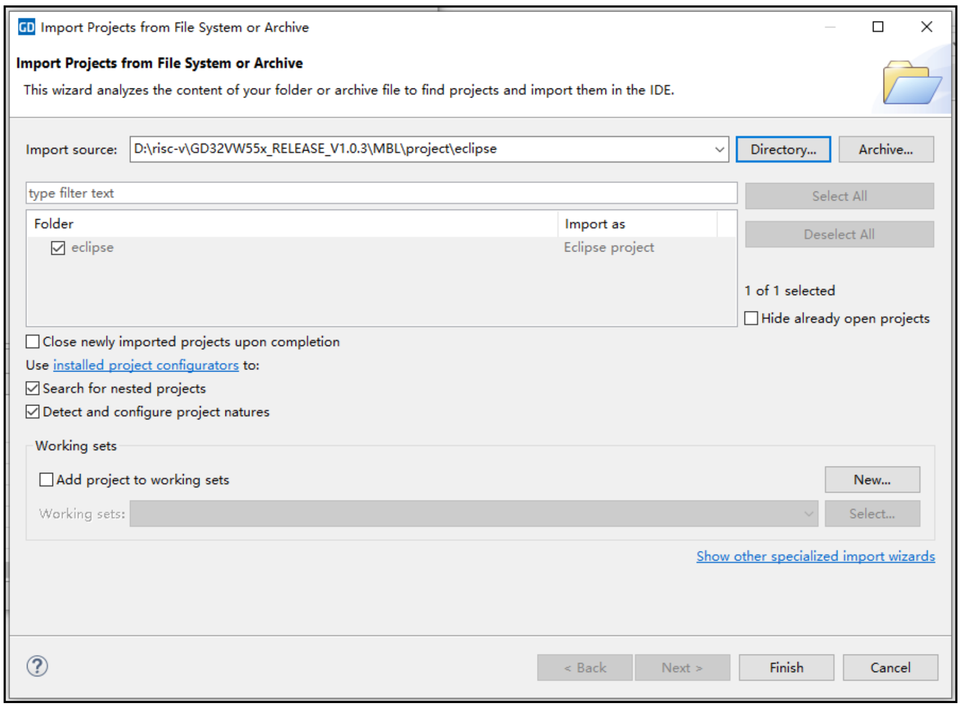
In the File menu, click Open Projects from file System, as shown in ***Figure 4-3. Open Projects from file System***.

Figure 4-3. Open Projects from file System



Select the project path GD32VW55x_RELEASE\MBL\project\eclipse, as shown in ***Figure 4-4. Selecting MBL project path***, and click Finish.

Figure 4-4. Selecting MBL project path


Import Projects from File System or Archive

Import Projects from File System or Archive

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: D:\risc-v\GD32VW55x_RELEASE_V1.0.3\MBL\project\eclipse

Folder	Import as
☑ eclipse	Eclipse project

1 of 1 selected

☐ Hide already open projects

☐ Close newly imported projects upon completion

Use installed project configurators to:

☑ Search for nested projects

☑ Detect and configure project natures

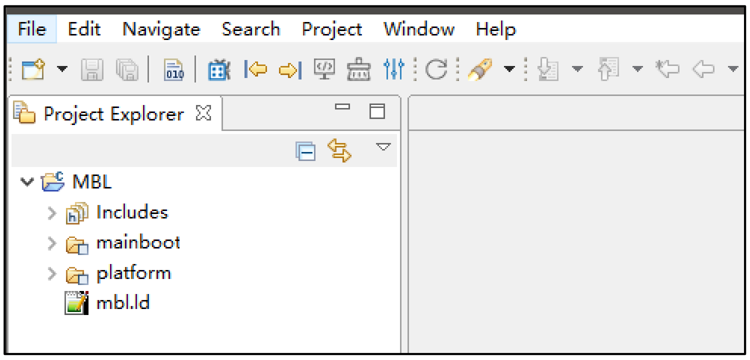
Working sets

☐ Add project to working sets

Working sets:

Close the welcome interface, and the MBL project is shown as ***Figure 4-5. MBL project interface*** shows.

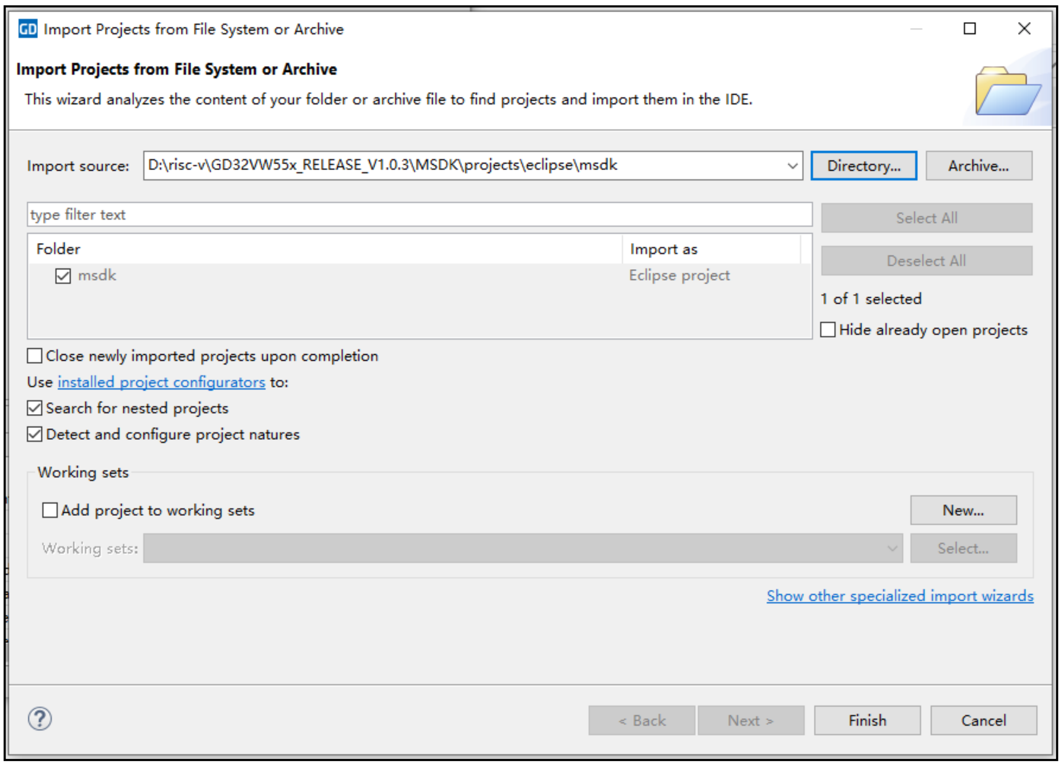
Figure 4-5. MBL project interface



- Import the MSDK project

In the File menu, click Open Projects from file System, Select the project path GD32VW55x_RELEASE\MSDK\projects\eclipse\msdk, as shown in ***Figure 4-6. Selecting MSDK project path***, and click Finish.

Figure 4-6. Selecting MSDK project path


Import Projects from File System or Archive

Import Projects from File System or Archive

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: D:\risc-v\GD32VW55x_RELEASE_V1.0.3\MSDK\projects\eclipse\msdk

Folder	Import as
☑ msdk	Eclipse project

1 of 1 selected

☐ Hide already open projects

☐ Close newly imported projects upon completion

Use installed project configurators to:

☑ Search for nested projects

☑ Detect and configure project natures

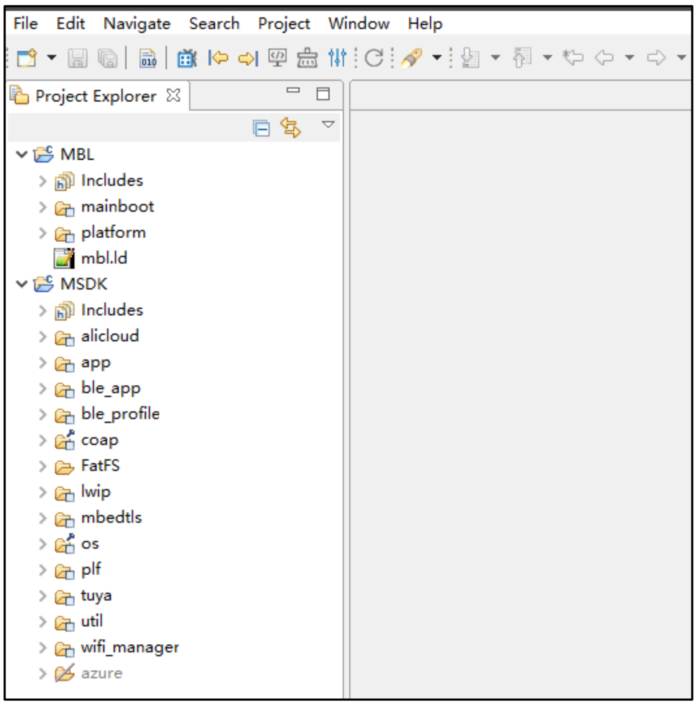
Working sets

☐ Add project to working sets

Working sets:

View the MSDK and MBL project interfaces, as shown in ***Figure 4-7. MSDK and MBL project interfaces***.

Figure 4-7. MSDK and MBL project interfaces

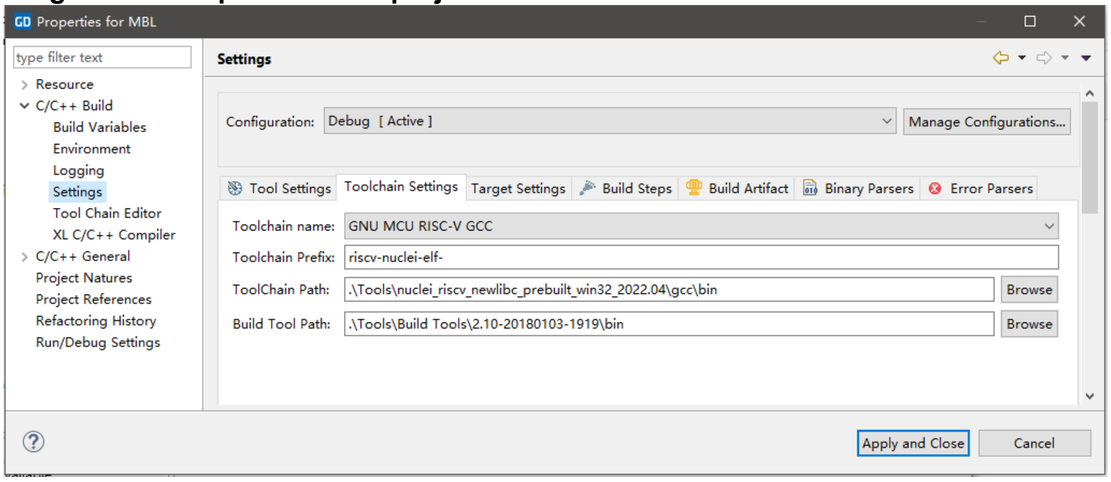


4.2. Compilation

- Check the configuration of the project compilation tool

Right-click on the project, click on properties, select C/C++ Build -> Settings in order, and on the tab click on toolchain settings., as shown in ***Figure 4-8. Properties of the project***.

Figure 4-8. Properties of the project



Configuration: Debug [Active]

Toolchain name: GNU MCU RISC-V GCC

Toolchain Prefix: riscv-nuclei-elf-

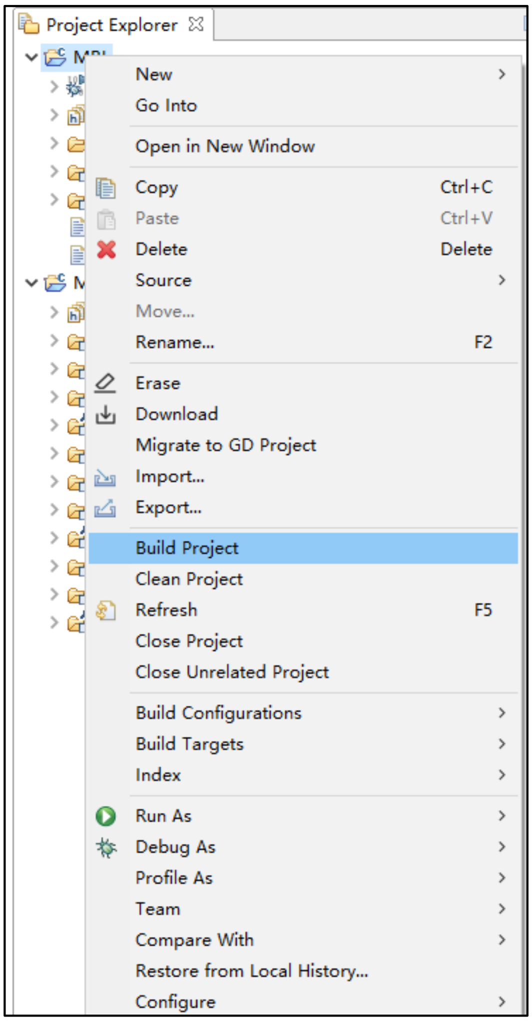
ToolChain Path: .\Tools\nuclei_riscv_newlibc_prebuilt_win32_2022.04\gcc\bin

Build Tool Path: .\Tools\Build Tools\2.10-20180103-1919\bin

- Compile the MBL project

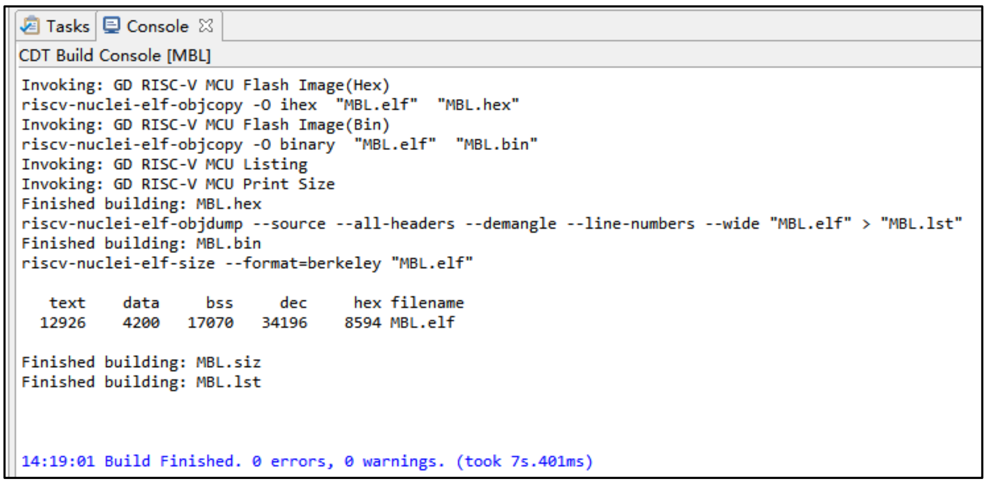
Right-click the project, and click Build Project, as shown in ***Figure 4-9. Compiling the MBL project***.

Figure 4-9. Compiling the MBL project



The compilation result is as shown in ***Figure 4-10. MBL compilation result***.

Figure 4-10. MBL compilation result



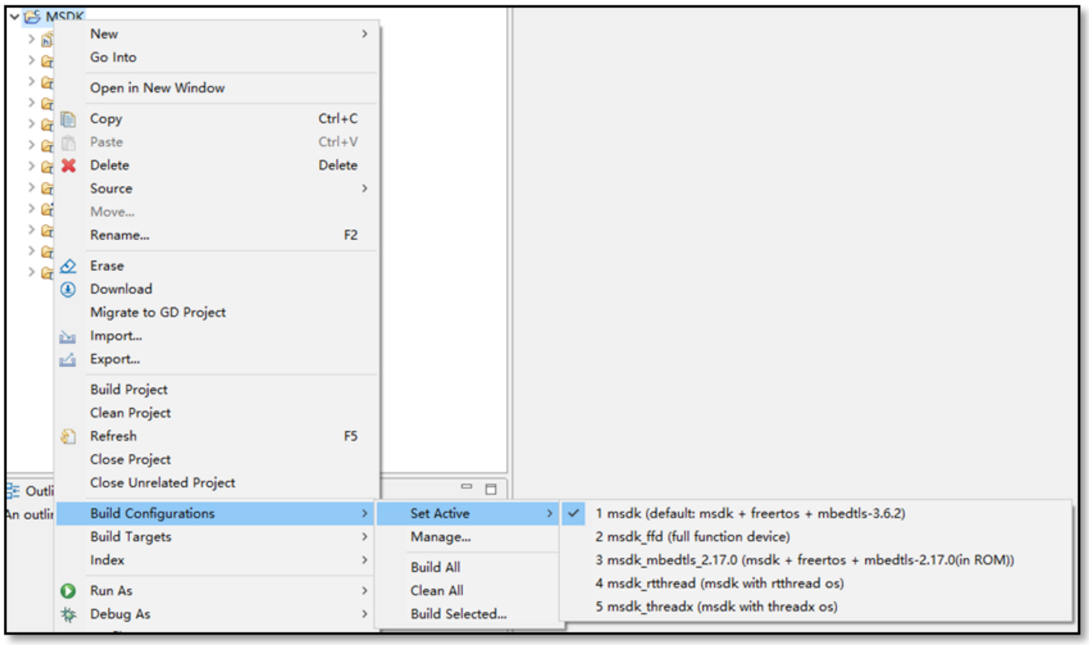
```
Invoking: GD RISC-V MCU Flash Image(Hex)
riscv-nuclei-elf-objcopy -O ihex  "MBL.elf"  "MBL.hex"
Invoking: GD RISC-V MCU Flash Image(Bin)
riscv-nuclei-elf-objcopy -O binary  "MBL.elf"  "MBL.bin"
Invoking: GD RISC-V MCU Listing
Invoking: GD RISC-V MCU Print Size
Finished building: MBL.hex
riscv-nuclei-elf-objdump --source --all-headers --demangle --line-numbers --wide "MBL.elf" > "MBL.lst"
Finished building: MBL.bin
riscv-nuclei-elf-size --format=berkeley "MBL.elf"

   text    data     bss     dec     hex filename
  12926    4200   17070   34196    8594 MBL.elf

Finished building: MBL.siz
Finished building: MBL.lst


14:19:01 Build Finished. 0 errors, 0 warnings. (took 7s.401ms)
```

After the compilation is complete, the script MBL\project\mbl_afterbuild.bat will be automatically called to generate mbl.bin and copied to the directory \scripts\images.

- Compile the MSDK project

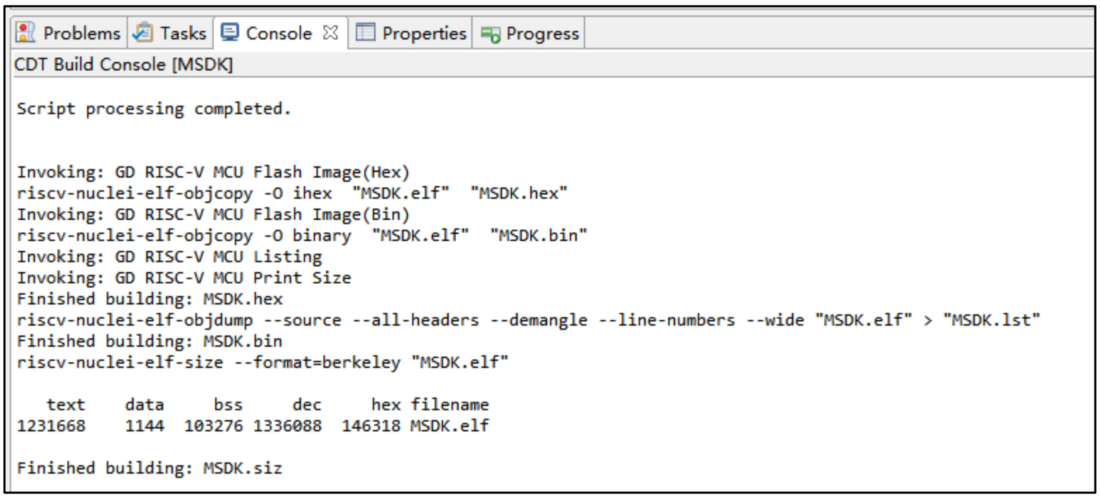
Right-click the project, and click Build Configurations—>Set Active—><target configuration> in order. as shown in *Figure 4-11. target configuration selection*, the default target project is msdk.

Figure 4-11. target configuration selection



Right-click the project again, and click Build Project, The compilation result is as shown in *Figure 4-12. MSDK compilation result*.

Figure 4-12. MSDK compilation result



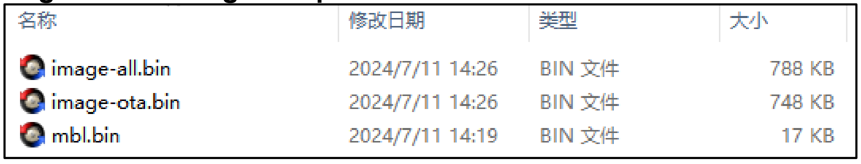
```
Script processing completed.


Invoking: GD RISC-V MCU Flash Image(Hex)
riscv-nuclei-elf-objcopy -O ihex  "MSDK.elf"  "MSDK.hex"
Invoking: GD RISC-V MCU Flash Image(Bin)
riscv-nuclei-elf-objcopy -O binary  "MSDK.elf"  "MSDK.bin"
Invoking: GD RISC-V MCU Listing
Invoking: GD RISC-V MCU Print Size
Finished building: MSDK.hex
riscv-nuclei-elf-objdump --source --all-headers --demangle --line-numbers --wide "MSDK.elf" > "MSDK.lst"
Finished building: MSDK.bin
riscv-nuclei-elf-size --format=berkeley "MSDK.elf"

   text    data     bss     dec     hex filename
1231668    1144  103276 1336088  146318 MSDK.elf

Finished building: MSDK.siz
```

- Images generated by SDK

After MSDK is compiled, it will call MSDK\projects\ image_afterbuild.bat to generate image-ota.bin and image-all.bin, and copy the generated bin files to \scripts\images, as shown in *Figure 4-13. Images output*.

image-ota.bin is the bin file generated by MSDK project, which can be used for OTA upgrade. image-all.bin is the combination of MBL(mbl.bin) and MSDK(image-ota.bin), the firmware can be used for production, download into FLASH and run.

Figure 4-13. Images output



名称	修改日期	类型	大小
image-all.bin	2024/7/11 14:26	BIN 文件	788 KB
image-ota.bin	2024/7/11 14:26	BIN 文件	748 KB
mbl.bin	2024/7/11 14:19	BIN 文件	17 KB

4.3. Download firmware

4.3.1. USB Drive Copy

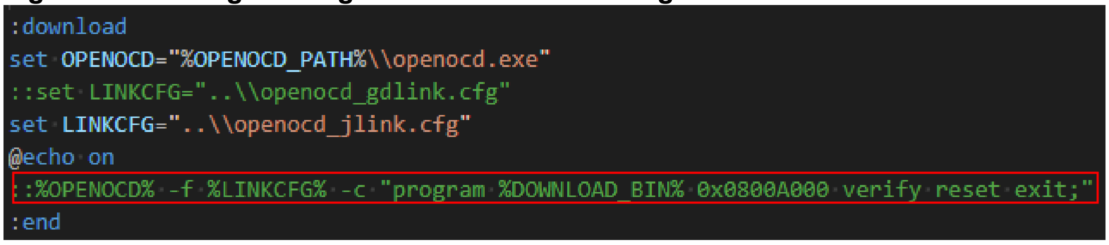
As shown in*1.4Download interface*, copying the image-all.bin file from GD32VW55x_RELEASE\scripts\images to the Gigadevice drive. This functionality is only supported on the START development board when using the onboard GDLink connection.

4.3.2. Use afterbuild.bat for downloading

The project supports automatically downloading the image after compilation by invoking a script through afterbuild. The script file is MSDK\projects\image_afterbuild.bat.

At the end of image_afterbuild.bat, there is a section of code configured for automatic image downloading, as shown in ***Figure 4-14 Configure image automatic downloading***.

Figure 4-14 Configure image automatic downloading



```
:download
set OPENOCD="%OPENOCD_PATH%\\openocd.exe"
::set LINKCFG="..\\openocd_gdlink.cfg"
set LINKCFG="..\\openocd_jlink.cfg"
@echo on
::%OPENOCD% -f %LINKCFG% -c "program %DOWNLOAD_BIN% 0x0800A000 verify reset exit;"
:end
```

This segment of code utilizes OpenOCD with Jlink/GDLink to perform downloading. By configuring OpenOCD to use different .cfg files, users can choose whether to download via Jlink or GDLink.

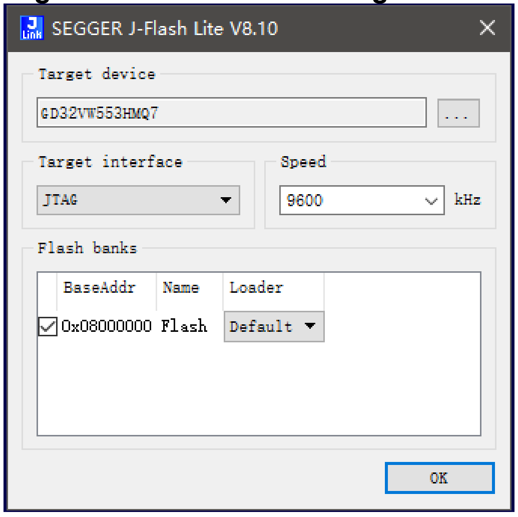
When using GDLink to connect the computer and development board, set LINKCFG to openocd_gdlink.cfg. For connections via Jlink, set LINKCFG to openocd_jlink.cfg.

After compilation, Afterbuild will invoke this script to execute the configured commands and complete the corresponding image download (please uncomment the lines within the red box from the figure; this feature is disabled by default).

4.3.3. Using J-Flash Lite for downloading

When using Jlink for debugging and firmware downloading, you can find JFlashLite.exe in the directory: GD32EmbeddedBuilder_v1.4.14.29824\Tools\J-Link. Double-click to open JFlashLite and configure it as shown in *Figure 4-15 JFlashLite Configuration*.

Figure 4-15 JFlashLite Configuration



SEGGER J-Flash Lite V8.10

Target device: GD32VW553HMQ7

Target interface: JTAG

Speed: 9600 kHz

Flash banks

	BaseAddr	Name	Loader
☑	0x08000000	Flash	Default

Set the Target device to GD32VW553xxxx, choose the Target interface as JTAG (cJTAG can also be selected, but it is slower), and set the Speed to 9600 kHz. Then click OK.

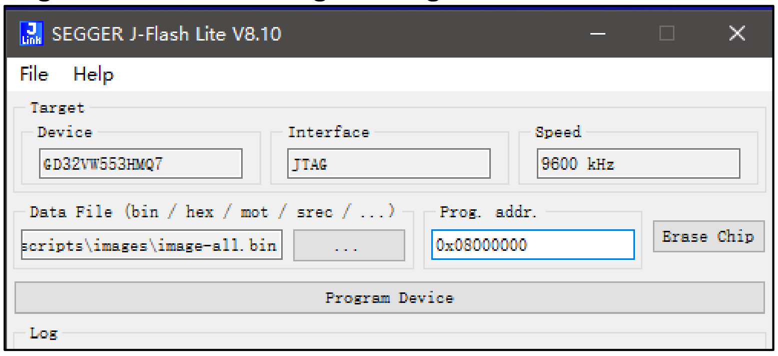
In the opened interface, as shown in *Figure 4-16 J-Flash Programming Interface*, select the Data File as the compiled image-all.bin or image-ota.bin (stored in the directory GD32VW55x_RELEASE_V1.0.xx\scripts\images after compilation).

When selecting image-all.bin, set the Prog. Addr. on the right to 0x08000000.

When selecting image-ota.bin, set the Prog. Addr. on the right to 0x0800A000.

Once the settings are complete, click "Program Device" and wait for the progress bar to finish, which indicates the completion of the programming process.

Figure 4-16 J-Flash Programming Interface



SEGGER J-Flash Lite V8.10

Target

Device: GD32VW553HMQ7

Interface: JTAG

Speed: 9600 kHz

Data File (bin / hex / mot / srec / ...): scripts\images\image-all.bin

Prog. addr.: 0x08000000

4.4. Debugging

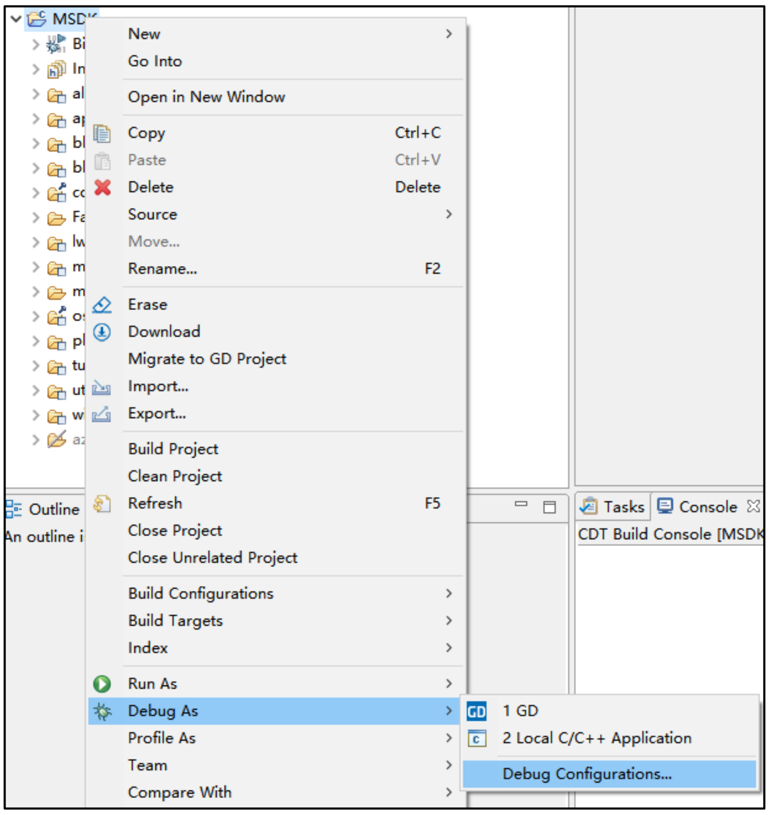
Currently, both the START development board and the EVAL development board feature onboard GDLink, and an external Jlink can also be used for debugging.

The debugging process is described below, and the default project configuration is msdk. If you need to switch to another project configuration, please refer to section ***6.3Select different project configurations during debugging*** for selecting different project configurations for debugging.

4.4.1. Debugging configuration

Right-click on the MSDK project and click Debug As->Debug Configurations, as shown in ***Figure 4-17. Opening the Debug Configuration option***.

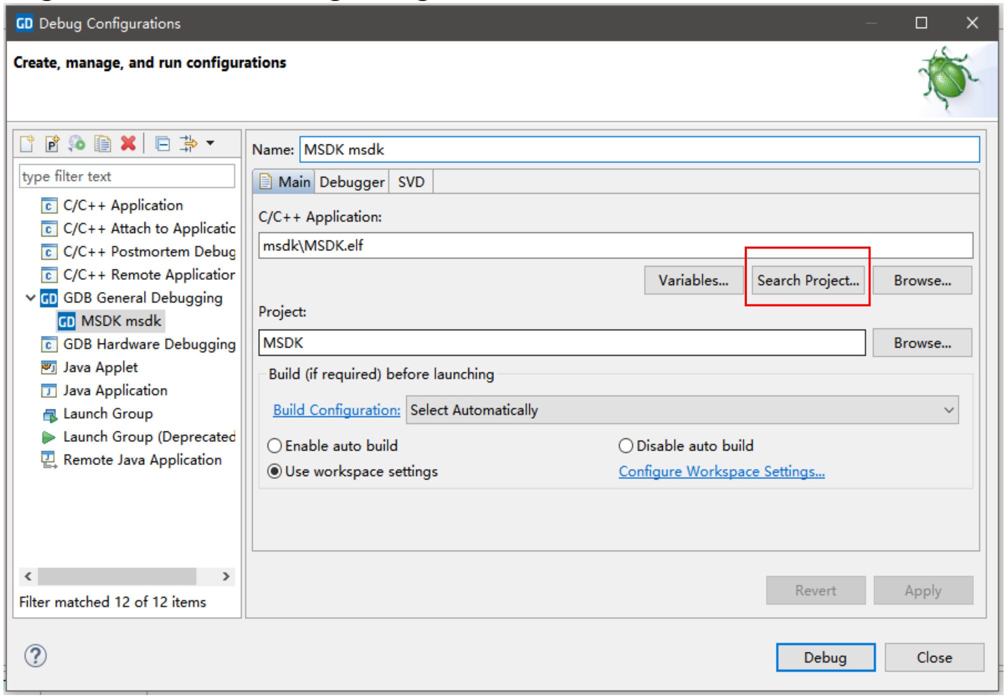
Figure 4-17. Opening the Debug Configuration option



Double-click "GDB General Debugging" on the left, and a Debug configuration will automatically be created, as shown in ***Figure 4-18. MSDK debug configuration***. Here, the c/c++ application field has automatically selected msdk\MSDK.elf, which points to the ELF file generated by the configuration to be debugged.

You can check or uncheck “Enable/Disable auto build” to choose whether or not to compile the project before debugging.

Figure 4-18. MSDK debug configuration



Name: MSDK msdk

C/C++ Application: msdk\MSDK.elf

Project: MSDK

Build (if required) before launching

Build Configuration: Select Automatically

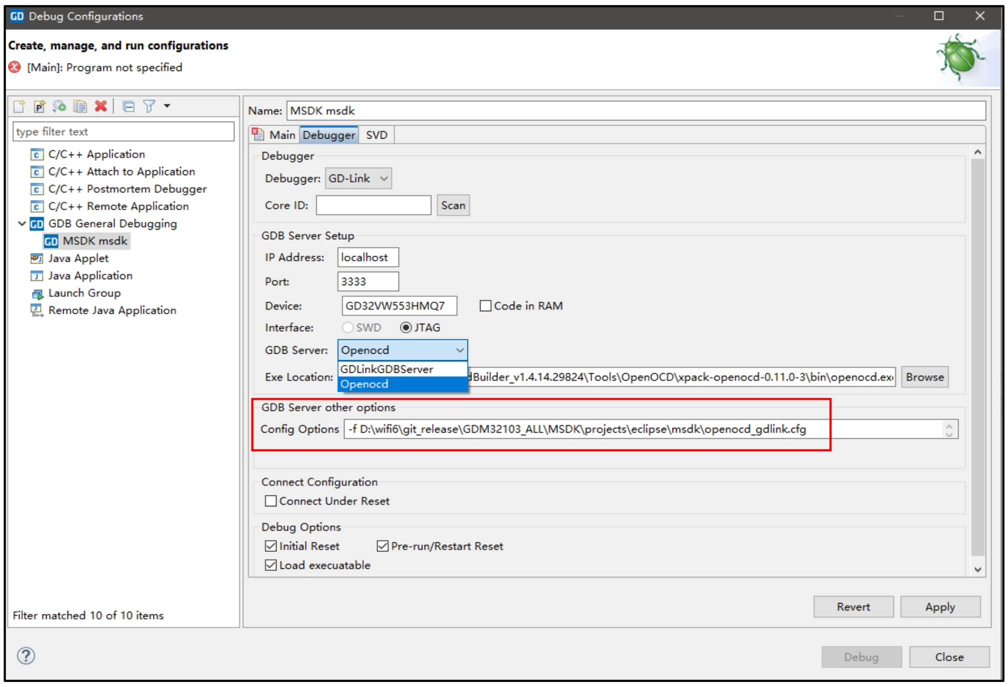
○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

4.4.2. Debugging using GDLink

As shown in *Figure 4-19. MSDK Debugging Configuration Interface with openocd*, switch the GDB Server to OpenOCD in the Debugger interface. Additionally, specify the Config Options within the red box as shown in the figure. Afterward, click "Debug" to start debugging. The debugging interface is displayed in *Figure 4-20. MSDK debug interface*.

Figure 4-19. MSDK Debugging Configuration Interface with openocd



[Main]: Program not specified

Name: MSDK msdk

Debugger

Debugger: GD-Link

Core ID:

GDB Server Setup

IP Address: localhost

Port: 3333

Device: GD32VW553HMQ7 ☐ Code in RAM

Interface: ○ SWD ● JTAG

GDB Server: Openocd

Exe Location: dBuilder_v1.4.14.29824\Tools\OpenOCD\xpack-openocd-0.11.0-3\bin\openocd.ex

GDB Server other options

Config Options -f D:\wifi6\git_release\GDM32103_ALL\MSDK\projects\eclipse\msdk\openocd_gdlink.cfg

Connect Configuration

☐ Connect Under Reset

Debug Options

☑ Initial Reset ☑ Pre-run/Restart Reset

☑ Load executable

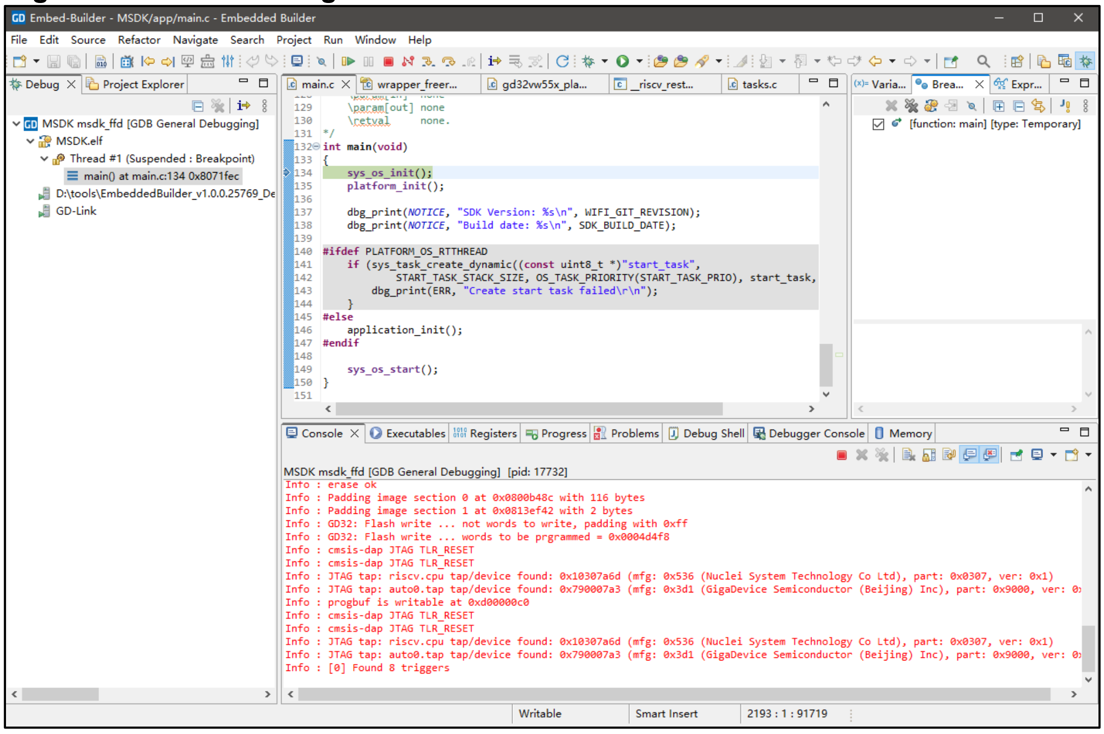
In *Figure 4-19. MSDK Debugging Configuration Interface with openocd*, selecting "Initial

Reset" and "Pre-run/Restart Reset" in the Debug Options will reset the chip at the start of debugging. Choosing "Load executable" will flash the firmware once before debugging begins.

4.4.3. Debugging using Jlink

The GD32VW553 supports JTAG and cJTAG debugging. First, connect the pins of the JLink debugger to the GD32VW553 JTAG pins. Next, replace the cfg file within the red box in the *Figure 4-19. MSDK Debugging Configuration Interface with openocd* with "openocd_jlink.cfg" (this file is located in the directory: MSDK\projects\eclipse\msdk). Then, click "Debug" to enter the debugging process. If driver issues arise during JLink debugging, please refer to *6.4 JLink Driver Replacement*.

Figure 4-20. MSDK debug interface



```
MSDK msdk_ffd [GDB General Debugging]
  MSDK.elf
    Thread #1 (Suspended : Breakpoint)
      main() at main.c:134 0x8071fec
  D:\tools\EmbeddedBuilder_v1.0.0.25769_De
  GD-Link
```

```
[function: main] [type: Temporary]
```

```
    \param[out] none
    \retval     none.
*/
int main(void)
{
    sys_os_init();
    platform_init();

    dbg_print(NOTICE, "SDK Version: %s\n", WIFI_GIT_REVISION);
    dbg_print(NOTICE, "Build date: %s\n", SDK_BUILD_DATE);

#ifdef PLATFORM_OS_RTTHREAD
    if (sys_task_create_dynamic((const uint8_t *)"start_task",
            START_TASK_STACK_SIZE, OS_TASK_PRIORITY(START_TASK_PRIO), start_task,
        dbg_print(ERR, "Create start task failed\r\n");
    }
#else
    application_init();
#endif

    sys_os_start();
}
```

```
MSDK msdk_ffd [GDB General Debugging]  [pid: 17732]
Info : erase ok
Info : Padding image section 0 at 0x0800b48c with 116 bytes
Info : Padding image section 1 at 0x0813ef42 with 2 bytes
Info : GD32: Flash write ... not words to write, padding with 0xff
Info : GD32: Flash write ... words to be prgrammed = 0x0004d4f8
Info : cmsis-dap JTAG TLR_RESET
Info : cmsis-dap JTAG TLR_RESET
Info : JTAG tap: riscv.cpu tap/device found: 0x10307a6d (mfg: 0x536 (Nuclei System Technology Co Ltd), part: 0x0307, ver: 0x1)
Info : JTAG tap: auto0.tap tap/device found: 0x790007a3 (mfg: 0x3d1 (GigaDevice Semiconductor (Beijing) Inc), part: 0x9000, ver: 0
Info : progbuf is writable at 0xd00000c0
Info : cmsis-dap JTAG TLR_RESET
Info : cmsis-dap JTAG TLR_RESET
Info : JTAG tap: riscv.cpu tap/device found: 0x10307a6d (mfg: 0x536 (Nuclei System Technology Co Ltd), part: 0x0307, ver: 0x1)
Info : JTAG tap: auto0.tap tap/device found: 0x790007a3 (mfg: 0x3d1 (GigaDevice Semiconductor (Beijing) Inc), part: 0x9000, ver: 0
Info : [0] Found 8 triggers
```

5. SEGGER Embedded Studio IDE project

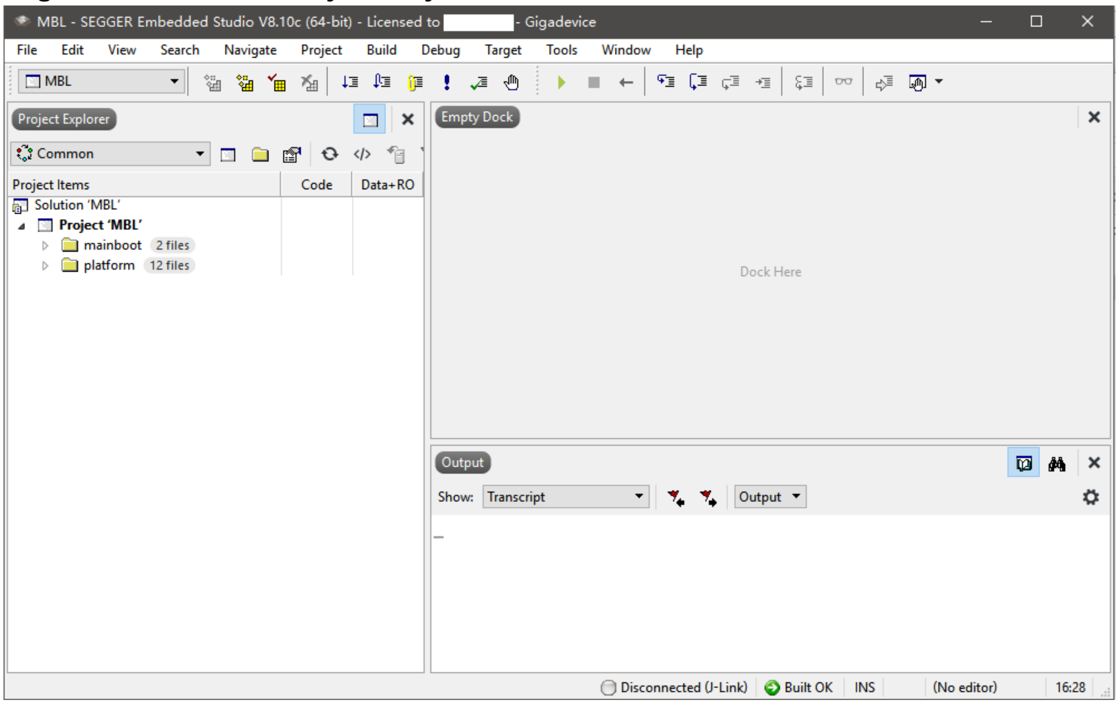
This chapter introduces how to compile and debug the SDK under SEGGER Embedded Studio IDE.

5.1. Open projects

- Open MBL project

Open the directory: GD32VW55x_RELEASE\MBL\project\segger, double click MBL.emProject to open the MBL SES project. The opened project is shown in ***Figure 5-1. MBL SES Project Project Interface***.

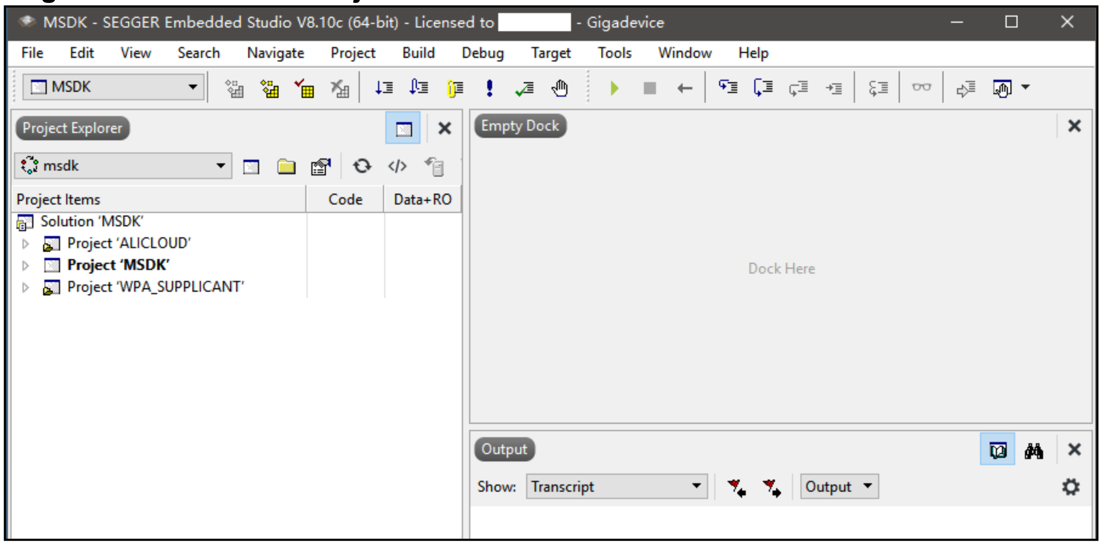
Figure 5-1. MBL SES Project Project Interface



- Open MSDK project

Open the directory: GD32VW55x_RELEASE\MSDK\projects\segger, double-click on the MSDK.emProject to open the MSDK project, open the project as ***Figure 5-2. MSDK SES Project Interface*** shown.

Figure 5-2. MSDK SES Project Interface

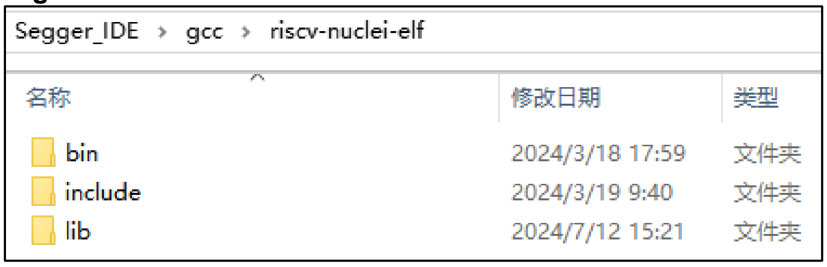


5.2. Compilation

■ SES build tool configuration

SES compiles the GD32VW55x project using the riscv32-none-elf toolchain by default. In order to better support the extended instruction set of riscv, it needs to be compiled using the nuclei toolchain: riscv-nuclei-elf. The compilation tool can be obtained by contacting sales or FAE. The details of the toolchain are shown in ***Figure 5-3. nuclei toolchain content***. Where the Segger_IDE is the SES IDE installation directory.

Figure 5-3. nuclei toolchain content

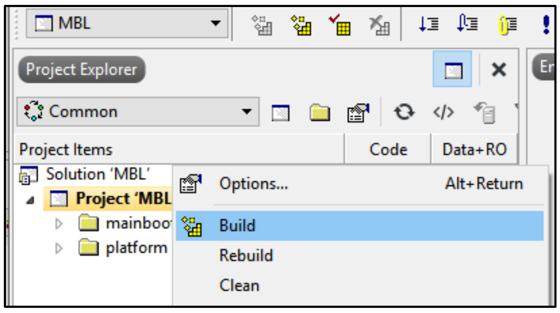


名称	修改日期	类型
bin	2024/3/18 17:59	文件夹
include	2024/3/19 9:40	文件夹
lib	2024/7/12 15:21	文件夹

■ Compile the MBL project

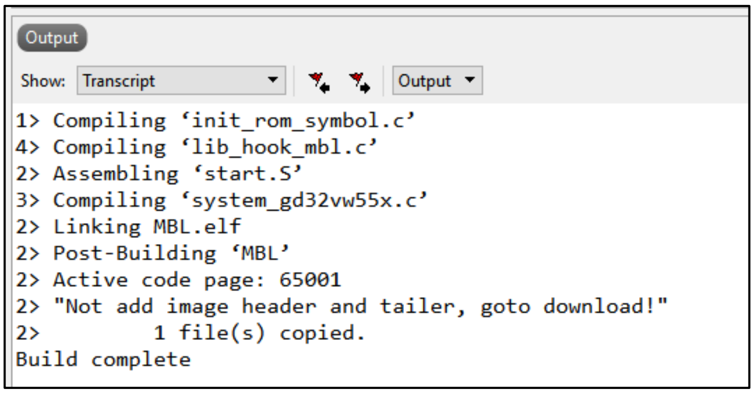
Right-click the project and click build to guild MBL, as shown in ***Figure 5-4. Compiling the MBL project***; or click Build->Build MBL in the menu bar.

Figure 5-4. Compiling the MBL project



The compilation result is as shown in ***Figure 5-5. MBL compilation result***.

Figure 5-5. MBL compilation result



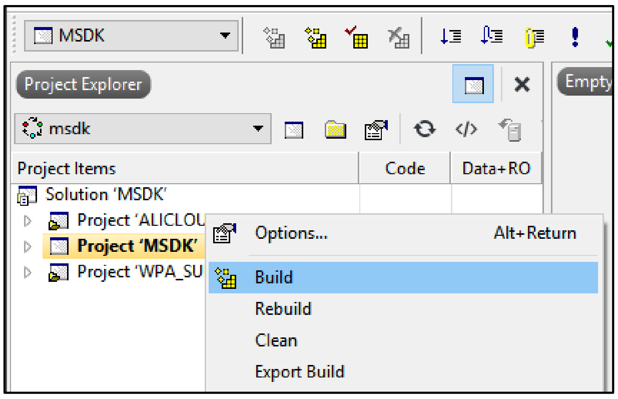
```
1> Compiling 'init_rom_symbol.c'
4> Compiling 'lib_hook_mbl.c'
2> Assembling 'start.S'
3> Compiling 'system_gd32vw55x.c'
2> Linking MBL.elf
2> Post-Building 'MBL'
2> Active code page: 65001
2> "Not add image header and tailer, goto download!"
2>         1 file(s) copied.
Build complete
```

After the compilation is complete, the script MBL\project\mbl_afterbuild.bat will be automatically called to generate mbl.bin and copied to the directory \scripts\images.

■ Compile the MSDK project

Right-click Project 'MSDK' and click Build, as shown in ***Figure 5-6. Compile MSDK project***.

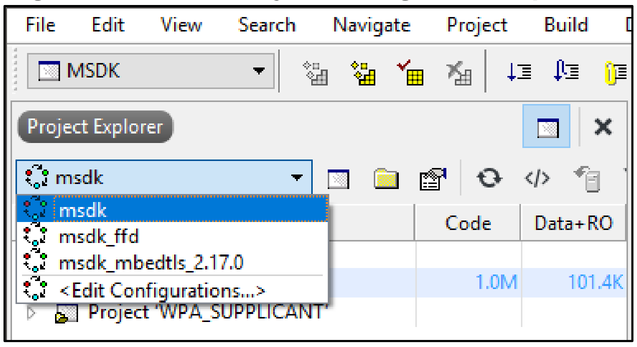
Figure 5-6. Compile MSDK project



■ Configuration selection of MSDK

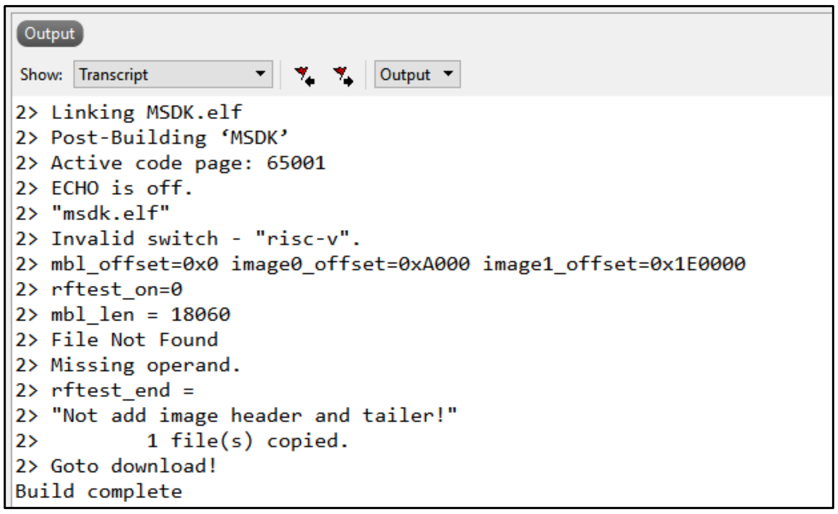
MSDK configuration switch as shown in ***Figure 5-7 MSDK Project Configuration Options***. MSDK SES project only supports msdk, msdk_ffd and msdk_mbedtls_2.17.0; if you need to use the configuration of msdk_threadx, msdk_ffd_threadx please use the GD32 EmbeddedBuilder IDE project or wait for subsequent updates.

Figure 5-7 MSDK Project Configuration Options



After selecting the corresponding configuration, right-click the project and click Build, the compilation result is shown in *Figure 5-8 MSDK compilation result*.

Figure 5-8 MSDK compilation result



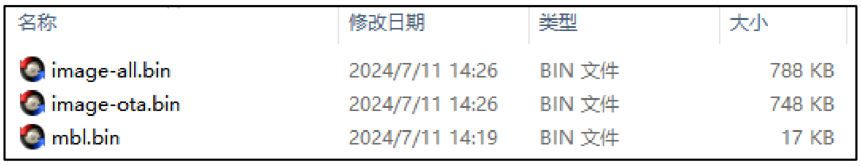
```
2> Linking MSDK.elf
2> Post-Building ‘MSDK’
2> Active code page: 65001
2> ECHO is off.
2> "msdk.elf"
2> Invalid switch - "risc-v".
2> mbl_offset=0x0 image0_offset=0xA000 image1_offset=0x1E0000
2> rftest_on=0
2> mbl_len = 18060
2> File Not Found
2> Missing operand.
2> rftest_end =
2> "Not add image header and tailer!"
2>         1 file(s) copied.
2> Goto download!
Build complete
```

- Image generated by SDK

After MSDK is compiled, it will call MSDK\projects\image_afterbuild.bat to generate image-ota.bin and image-all.bin, and copy the generated bin files to \scripts\images, as shown in *Figure 5-9. Images output*.

image-ota.bin is the bin file generated by MSDK project, which can be used for OTA upgrade. image-all.bin is the combination of MBL(mbl.bin) and MSDK(image-ota.bin), the firmware can be used for production, download into flash and run.

Figure 5-9. Images output

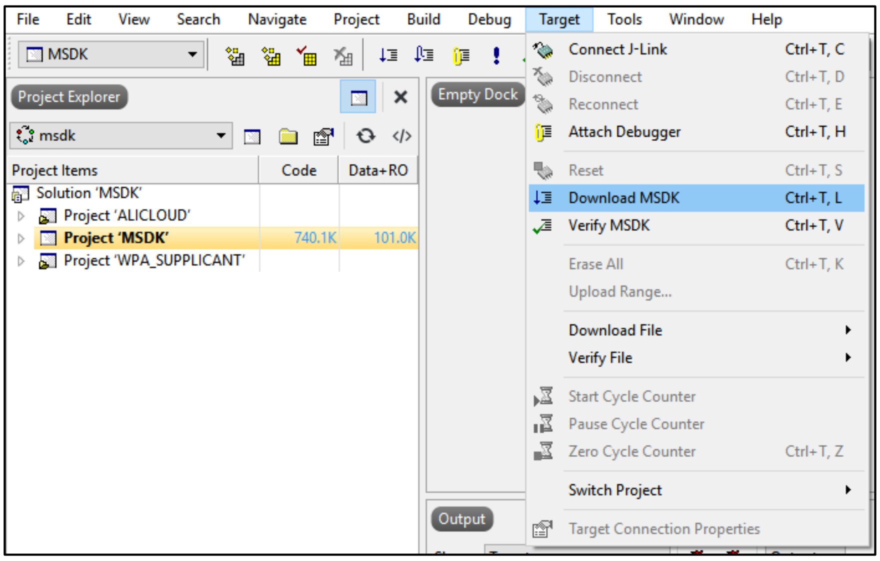


名称	修改日期	类型	大小
image-all.bin	2024/7/11 14:26	BIN 文件	788 KB
image-ota.bin	2024/7/11 14:26	BIN 文件	748 KB
mbl.bin	2024/7/11 14:19	BIN 文件	17 KB

5.3. Download firmware

Refer to *1.4 Download interface*, copy GD32VW55x_RELEASE\scripts\images\image-all.bin to the Gigadevice disc to download it. Or download it by clicking Target->Download MSDK in the menu bar, as shown in *Fogure 5-10 SES IDE image download*.

Fogure 5-10 SES IDE image download



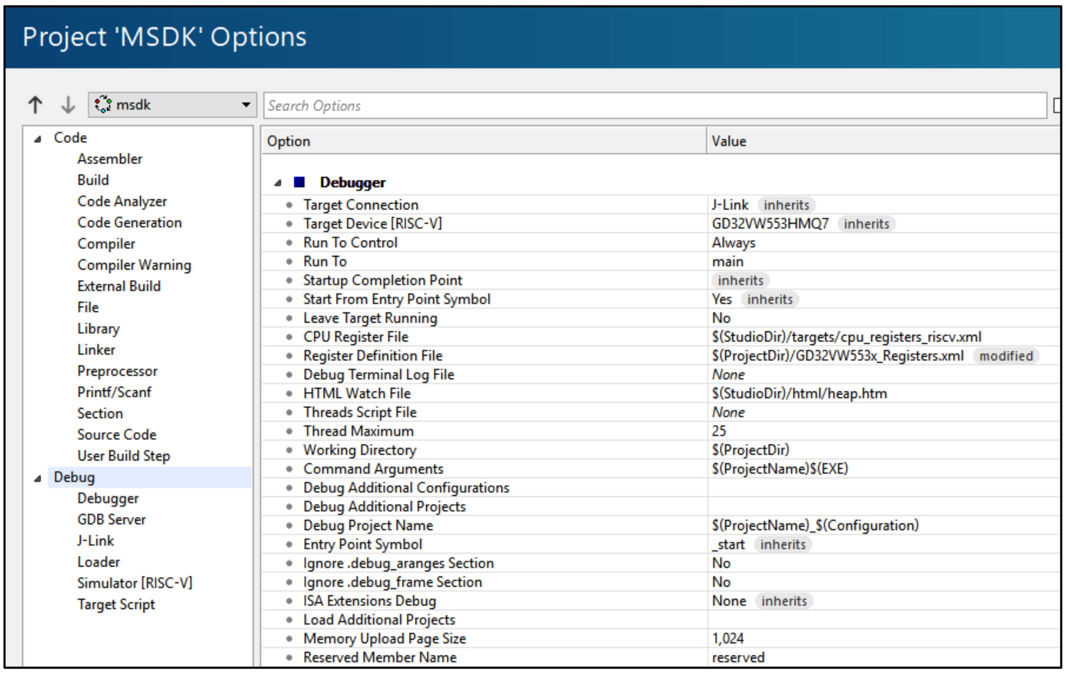
5.4. Debugging

■ Debugging configuration

SES IDE recommends using J-link to debug, and J-link driver version at least V7.92o, this version of J-link driver support GD32VW55x chip.

The project has been configured with Debug information by default, if you need to change it, right-click on the MSDK project, click Options to open the configuration interface, you can modify the Debugger and JLink under the Debug option, as shown in *Figure 5-11. MSDK SES Project Configuration Interface*.

Figure 5-11. MSDK SES Project Configuration Interface


Project 'MSDK' Options

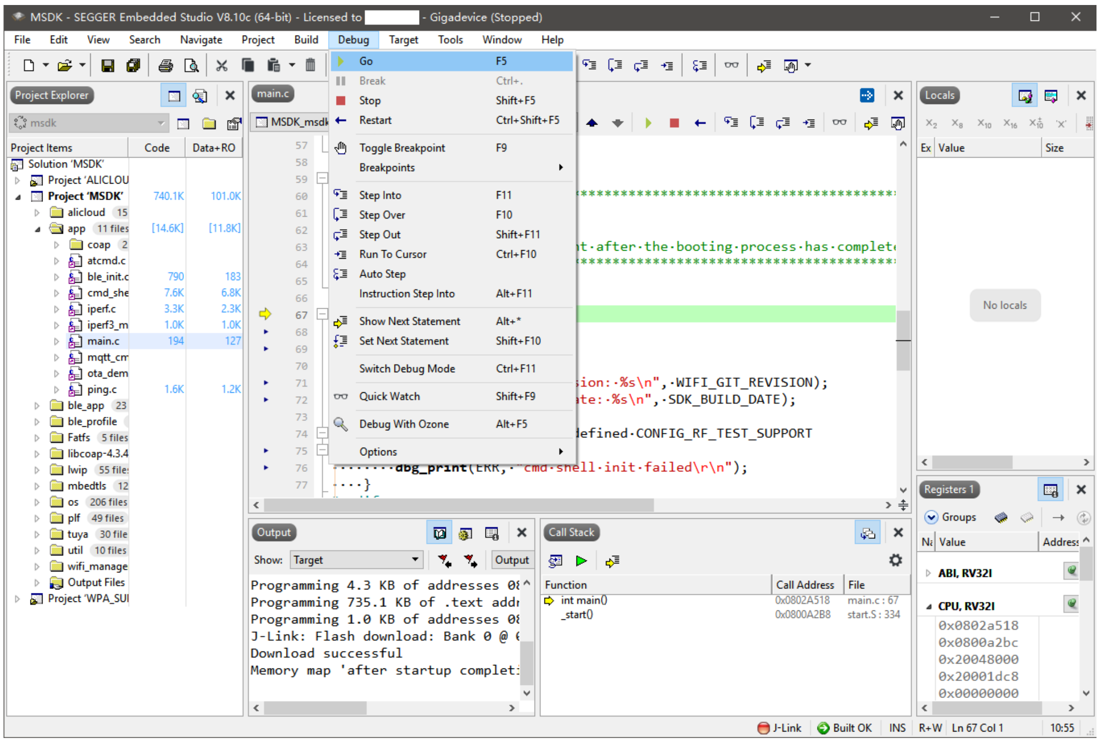
msdk

Option	Value
Debugger	
Target Connection	J-Link inherits
Target Device [RISC-V]	GD32VW553HMQ7 inherits
Run To Control	Always
Run To	main
Startup Completion Point	inherits
Start From Entry Point Symbol	Yes inherits
Leave Target Running	No
CPU Register File	$(StudioDir)/targets/cpu_registers_riscv.xml
Register Definition File	$(ProjectDir)/GD32VW553x_Registers.xml modified
Debug Terminal Log File	*None*
HTML Watch File	$(StudioDir)/html/heap.htm
Threads Script File	*None*
Thread Maximum	25
Working Directory	$(ProjectDir)
Command Arguments	$(ProjectName)$(EXE)
Debug Additional Configurations	
Debug Additional Projects	
Debug Project Name	$(ProjectName)_$(Configuration)
Entry Point Symbol	_start inherits
Ignore .debug_aranges Section	No
Ignore .debug_frame Section	No
ISA Extensions Debug	None inherits
Load Additional Projects	
Memory Upload Page Size	1,024
Reserved Member Name	reserved

■ Start Debugging

Click Debug->GO in the menu bar to debug, click and wait for the image downloading to complete and enter the interface shown in ***Figure 5-12. SES IDE Debug Interface***.

Figure 5-12. SES IDE Debug Interface


MSDK - SEGGER Embedded Studio V8.10c (64-bit) - Licensed to [redacted] - Gigadevice (Stopped)

```
Programming 4.3 KB of addresses 08
Programming 735.1 KB of .text addr
Programming 1.0 KB of addresses 08
J-Link: Flash download: Bank 0 @ 0
Download successful
Memory map 'after startup completi
```

Function	Call Address	File
int main()	0x0802A518	main.c : 67
_start()	0x0800A2B8	start.S : 334

6. FAQ

6.1. No image error

Print ERR: No image to boot (ret = -5).

Reason: An error occurs during the previous boot of WIFI_IOT, and the MBL records operation exception of the IMAGE. If another IMAGE is not downloaded or also has a boot exception, this message will be printed. In other words, the MBL believes that there is no valid IMAGE to jump to, and the boot fails.

Solution: Download the MBL again. After that, the IMAGE status will be cleared.

6.2. Code running in SRAM

To run programs faster to achieve higher performance, move them to the SRAM.

Open GD32VW55x_RELEASE\MSDK\plf\riscv\env\gd32vw55x.ld, and find the line ".code_to_sram:". The code in the braces runs in the SRAM. To add new content, add it at the end of the code. Refer to existing files for the format, for exemple:

KEEP (*port.o* (.text* .rodata*))

It is to put the entire port.c file in the SRAM and run it. For example:

KEEP (*tasks.o* (.text.xTaskIncrementTick))

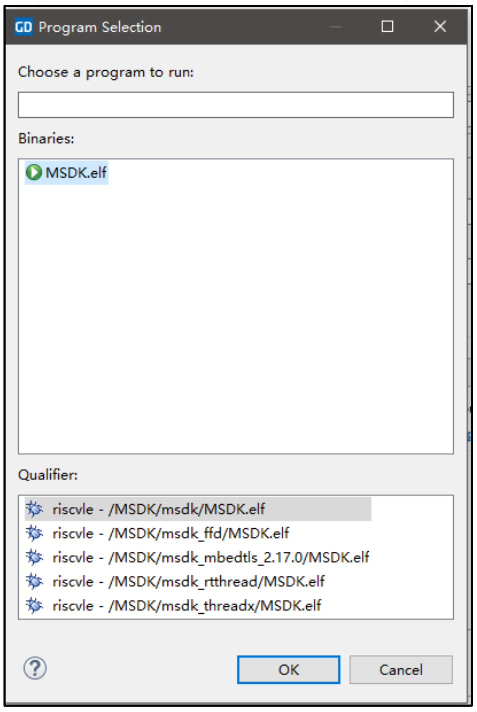
It is to put the xTaskIncrementTick () function in tasks.c in the SRAM and run it.

6.3. Select different project configurations during debugging

The MSDK project supports multiple configurations (refer to ***3.2.6Configuration Selection***). During debugging, you need to select the appropriate configuration.

The specific operation is as follows: Click the "Search Project" button within the red box shown in ***Figure 4-18. MSDK debug configuration***. This opens the interface shown in ***Figure 6-1 Select Project Configuration for Debugging***. Double-click the configuration you want to debug within the Qualifier box below (note that you need to compile the corresponding configuration first for the option to appear).

Figure 6-1 Select Project Configuration for Debugging


Program Selection

Choose a program to run:

Binaries:

MSDK.elf

Qualifier:

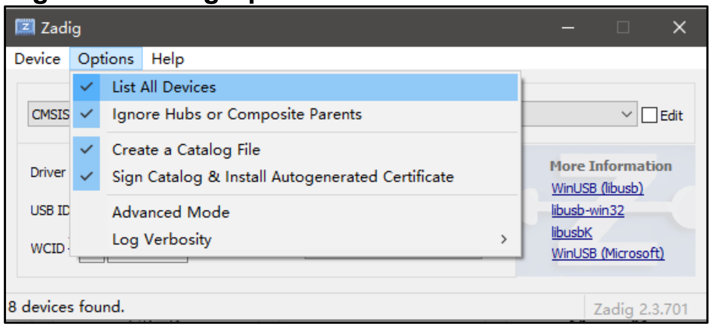
riscvle - /MSDK/msdk/MSDK.elf
riscvle - /MSDK/msdk_ffd/MSDK.elf
riscvle - /MSDK/msdk_mbedtls_2.17.0/MSDK.elf
riscvle - /MSDK/msdk_rtthread/MSDK.elf
riscvle - /MSDK/msdk_threadx/MSDK.elf

6.4. JLink Driver Replacement

When using OpenOCD+JLink debugging in an Embedded Builder project, if encountering the issue "Error: No J-Link device found," JLink drivers need to be replaced. The steps are as follows:

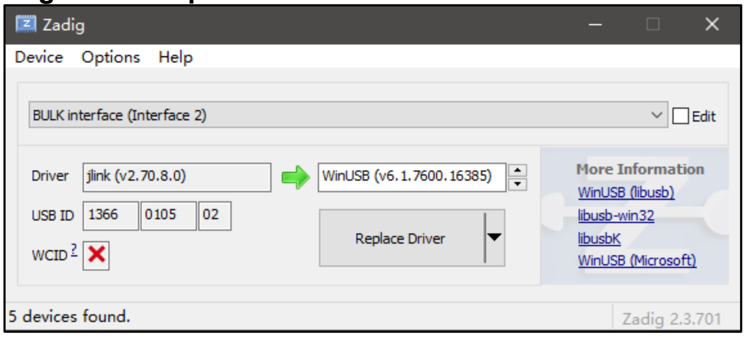
1. Use administrator privileges to open the zadig.exe file (official website: https://zadig.akeo.ie). Click "Options" and check "List All Devices," as shown in ***Figure 6-2 Zadig Options Selection***.

Figure 6-2 Zadig Options Selection



2. Select "JLink devices" from the dropdown menu, as shown in ***Figure 6-3 Replace JLink Driver***, where the BULK interface is displayed. Click "Replace Driver" to replace the JLink driver with WinUSB.

Figure 6-3 Replace JLink Driver


Zadig
Device Options Help
BULK interface (Interface 2) ☐ Edit
Driver jlink (v2.70.8.0) WinUSB (v6.1.7600.16385)
USB ID 1366 0105 02
WCID ✗
Replace Driver
More Information
WinUSB (libusb)
libusb-win32
libusbK
WinUSB (Microsoft)
5 devices found.
Zadig 2.3.701

3. After the replacement is complete, unplug and replug the JLink device. Then use JLink for debugging, and there will be no driver issues.

7. Revision history

Table 7-1. Revision history

Revision No.	Description	Date
1.0	Initial release	Nov.24.2023
1.1	Chapter 2 revision	Jan.26.2024
1.2	SES IDE project added, GD32 Eclipse IDE updated to GD32 Embedded Builder	July.17. 2024
1.3	Update some diagrams to be consistent w ith SDK1.0.3	Apr.8.2025
1.3a	Update the development board images and add firmw are dow nloads	May.15.2025

Important Notice