

1. Introduction

- 1.1. Supported operating systems
- 1.2. Installation
 - 1.2.1. Linux
- 1.3. Launch

2. Creating and importing project

- 2.1. Creating project
 - 2.1.1. Creating a GD project
 - 2.1.1.1. Graphical Configuration
 - 2.1.1.2. Code Generation
 - 2.1.2. Creating a C project
- 2.2. Importing project
 - 2.2.1. Importing a GD project
 - 2.2.2. Importing a C project
 - 2.2.3. Importing a GNU project
 - 2.2.4. Importing a MDK5 project
 - 2.2.5. Importing a IAR project

3. Building project

- 3.1. Framework of Project
- 3.2. Build Configuration
- 3.3. Toolchain Configuration
- 3.4. Modify Toolchain
- 3.5. Headless build
 - 3.5.1. Windows platform
 - 3.5.2. Linux platform

4. Debugging and Running Project

- 4.1. Creating a Debug/Run Configuration
- 4.2. Editing the Debug/Run Configuration
- 4.3. Debugging/Running
- 4.4. Resetting during Debugging
- 4.5. Single-step Debugging of Assembly Code
- 4.6. Download & Erase
- 4.7. Peripherals View
- 4.8. Live Expressions View
- 4.9. RTT Console View

5. FAQs

- 5.1 Building
- 5.2 Debugging
- 5.3 Platform

6. Revision history

1. Introduction

GD32 Embedded Builder stands as an intricately crafted C-language integrated development environment (IDE) rooted in the Eclipse platform. It serves as an exceptional embedded software development platform based on the RISC-V and ARM® Cortex processor architectures.

The objective of this document is to provide guidance on the utilization of GD32 Embedded Builder. It offers functionalities encompassing project creation, graphical MCU configuration, and project debugging.

1.1. Supported operating systems

Component	Windows	Linux
Platform	Windows 10、 Windows 11	Ubuntu 22.04.2 LTS Ubuntu 24.04.3 LTS
Minimum Memory	4GB	8GB

Note:

- 1.Linux systems require at least 8GB of memory.
- 2.When the Linux system display is configured as Wayland, the GD32EmbeddedBuilder_supportWayland script file needs to be used to start GD32EmbeddedBuilder. For details, refer to section 1.2.1.

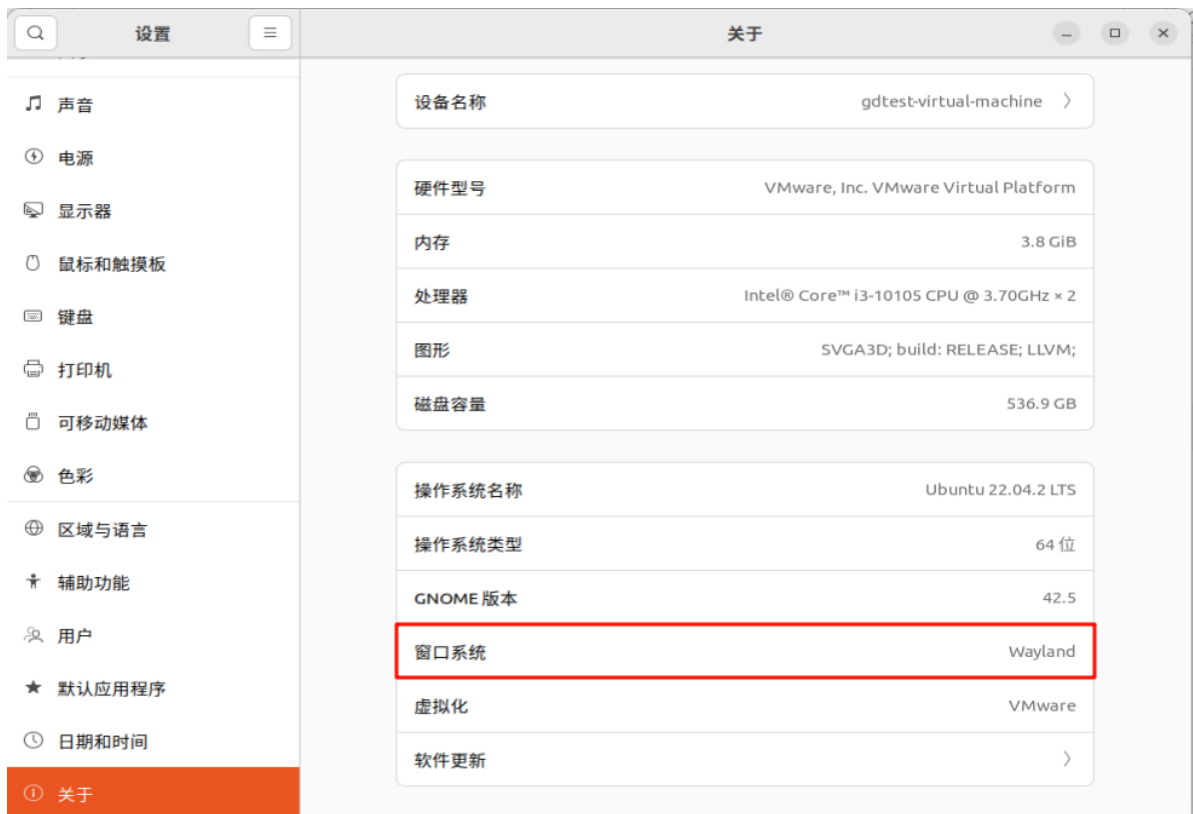
1.2. Installation

GD32 Embedded Builder is a software based on Eclipse and Java platform. To facilitate its utilization, if GD32 Embedded Builder version is earlier than 1.4.1.23782 (excluding 1.4.1.23782), please ensure that the Java Development Kit (JDK) version 1.8 is successfully installed and configured on your computer in advance.

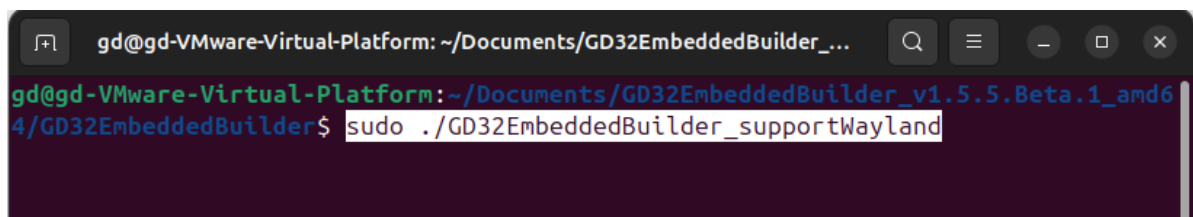
GD32 Embedded Builder is a green and portable software that operates without the need for installation.

1.2.1. Linux

If your display configuration is Wayland, you need to start GD32EmbeddedBuilder through the script file,otherwise, the GD project cannot function properly.



In the path where GD32EmbeddedBuilder is located, execute the command "sudo ./GD32EmbeddedBuilder_supportWayland" in the terminal to start GD32EmbeddedBuilder.



No need to pay attention to the command line output, as it does not affect the software functionality. For example, if the command line displays information like "(GD32EmbeddedBuilder:30457): Gtk-CRITICAL **: 15:45:01.274: gtk_box_gadget_distribute: assertion 'size >= 0' failed in GtkScrollbar", simply maximize the GD32EmbeddedBuilder interface.



1.3. Launch

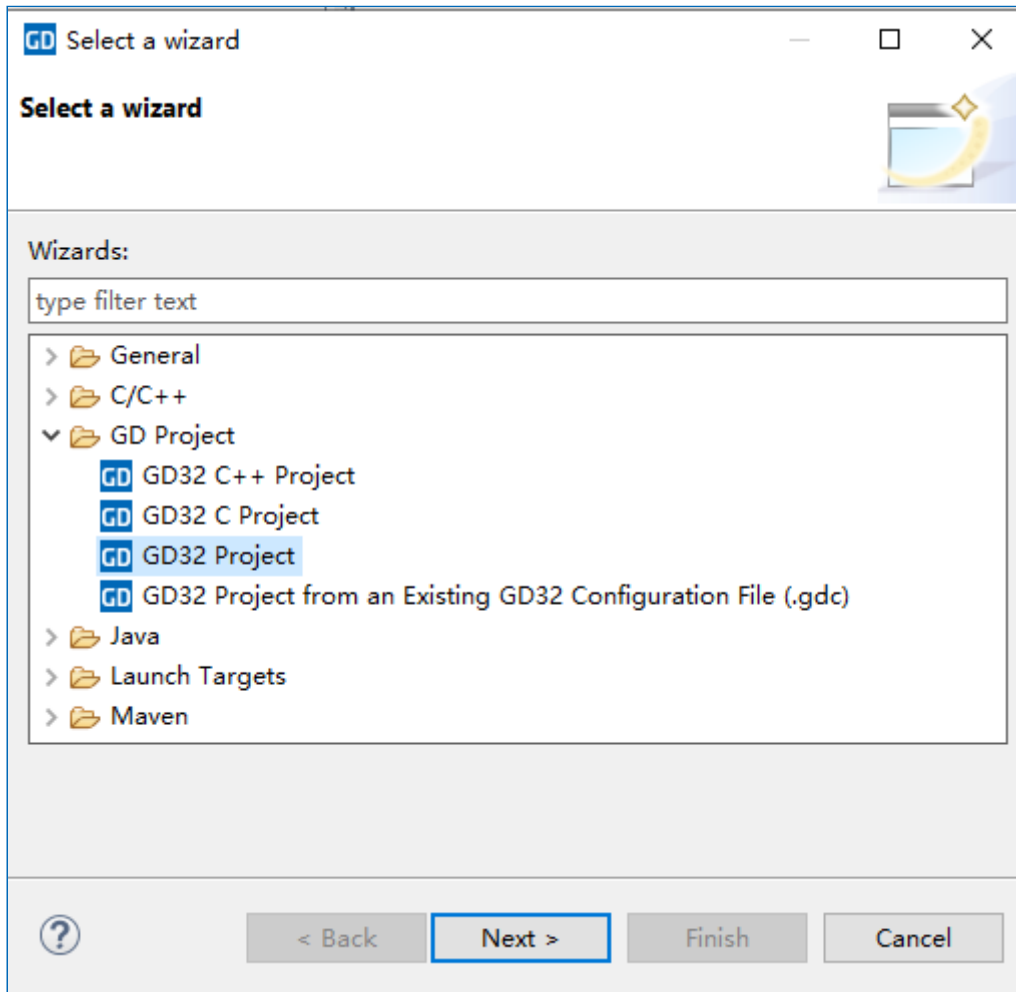
Click the **GD32EmbeddedBuilder.exe** executable file in the Eclipse package provided by GigaDevice to start the software.

2. Creating and importing project

2.1. Creating project

2.1.1. Creating a GD project

1) Navigate to **File > New > Other > GD Project > GD32 Project** within the menu commands, or **Create GD32 Project** on the **Welcome** page to start the new project wizard. Click the **Next** button to continue.



2) Input a designated name into the **Project Name** field, then proceed by clicking the **Next** button.

NEW GD Project

New GD Project
Please input GD Project Name

Project Name

☒ Use default location

Project Path

3) Select a required device from the **Device** list, then click **Finish** button.

NEW GD Project

New GD Project
Please select the target

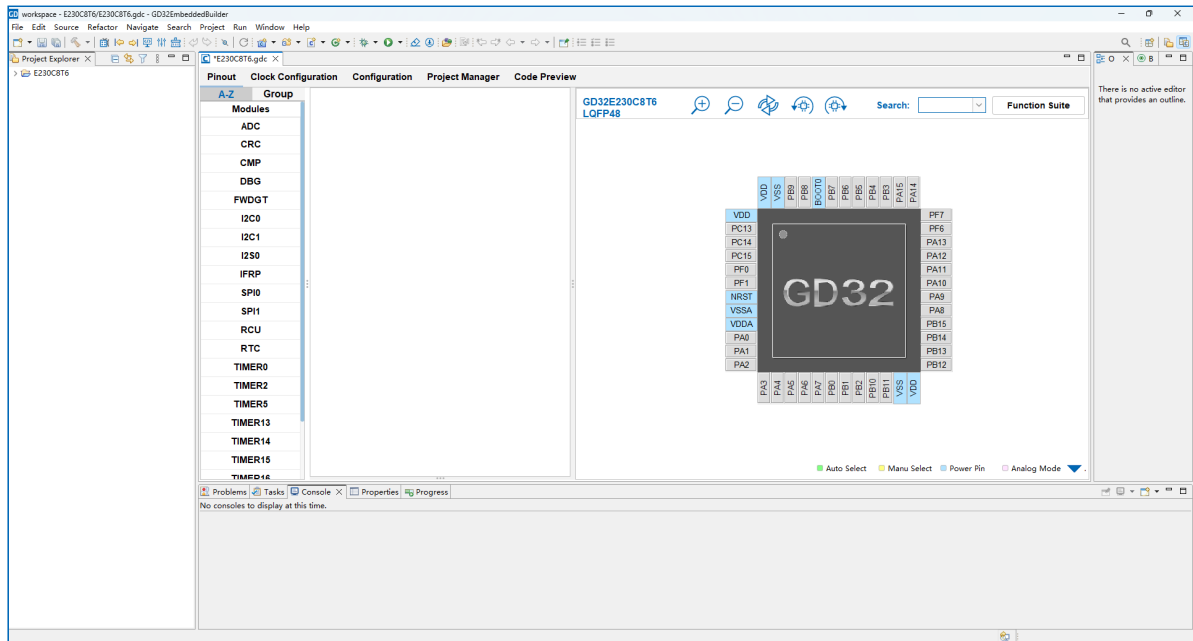
Series: Package: Core:

Flash >= RAM >= IO >=

MCU	Series	Package	Core	Flash	Ram	IO
GD32E230F4P6	GD32E230	TSSOP20	cortex-m23	16	4	15
GD32E230F6P6	GD32E230	TSSOP20	cortex-m23	32	6	15
GD32E230F8P6	GD32E230	TSSOP20	cortex-m23	64	8	15
GD32E230F4V6	GD32E230	LGA20	cortex-m23	16	4	15
GD32E230F6V6	GD32E230	LGA20	cortex-m23	32	6	15
GD32E230F8V6	GD32E230	LGA20	cortex-m23	64	8	15
GD32E230G4U6	GD32E230	QFN28	cortex-m23	16	4	23
GD32E230G6U6	GD32E230	QFN28	cortex-m23	32	6	23
GD32E230G8U6	GD32E230	QFN28	cortex-m23	64	8	23
GD32E230K4U6	GD32E230	QFN32	cortex-m23	16	4	27
GD32E230K6U6	GD32E230	QFN32	cortex-m23	32	6	27
GD32E230K8U6	GD32E230	QFN32	cortex-m23	64	8	27
GD32E230K4T6	GD32E230	LQFP32	cortex-m23	16	4	25
GD32E230K6T6	GD32E230	LQFP32	cortex-m23	32	6	25
GD32E230K8T6	GD32E230	LQFP32	cortex-m23	64	8	25
GD32E230C4T6	GD32E230	LQFP48	cortex-m23	16	4	39
GD32E230C6T6	GD32E230	LQFP48	cortex-m23	32	6	39
GD32E230C8T6	GD32E230	LQFP48	cortex-m23	64	8	39
GD32F310F4P6	GD32F310	TSSOP20	cortex-m4	16	4	15
GD32F310F6P6	GD32F310	TSSOP20	cortex-m4	32	6	15
GD32F310F8P6	GD32F310	TSSOP20	cortex-m4	64	8	15
GD32F310G8U6	GD32F310	QFN28	cortex-m4	64	8	23
GD32F310K8U6	GD32F310	QFN32	cortex-m4	64	8	27

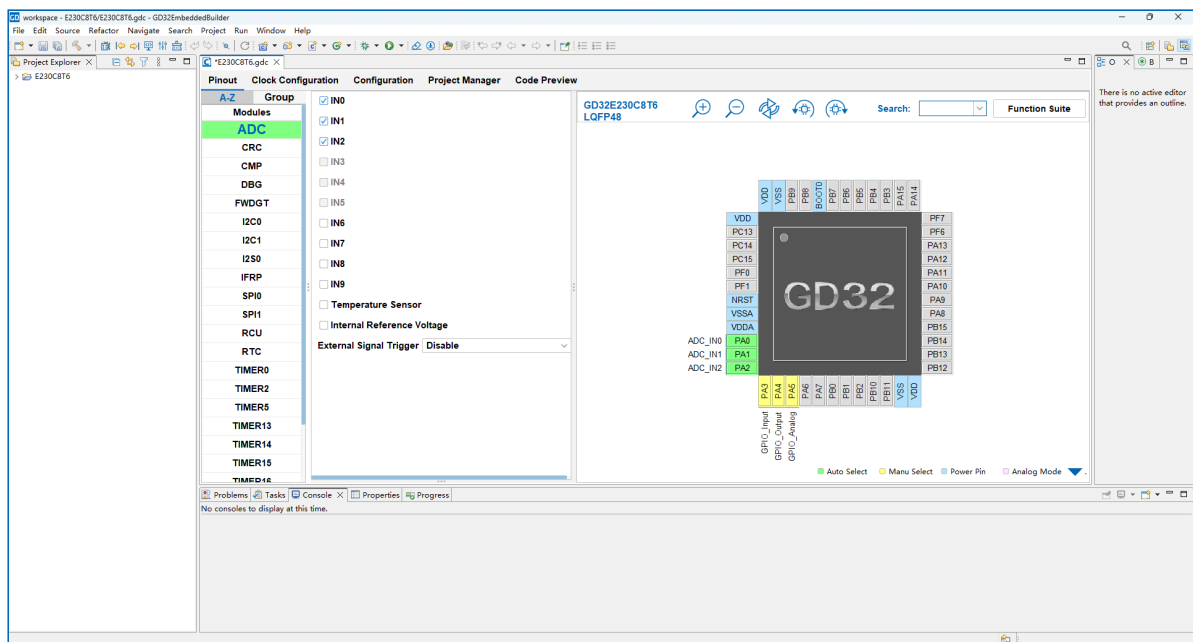
2.1.1.1. Graphical Configuration

GD32 Embedded Builder provides automatic pin function assignment, clock parameter configuration, peripheral information configuration, code preview and project manager configuration.

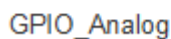


Users can customize the peripheral functions according to your specific requirements. The pin assignment function can be operated intelligently, providing both automatic and manual pin assignment support.

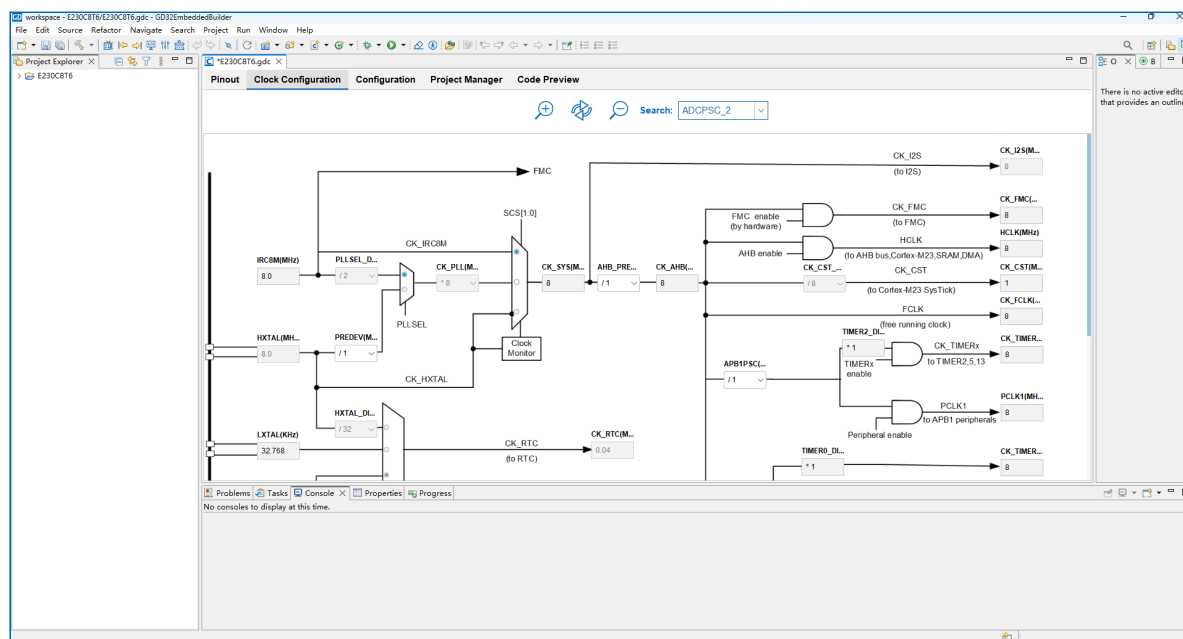
The color of pins represents different meanings. **Blue** represents non-configurable pins. **Gray** represents configurable pins. **Green** represents the pins associated with the pinout function, which support automatic and manual assignment. **Yellow** represents the pins that are not associated with the pinout function and only support manual assignment.



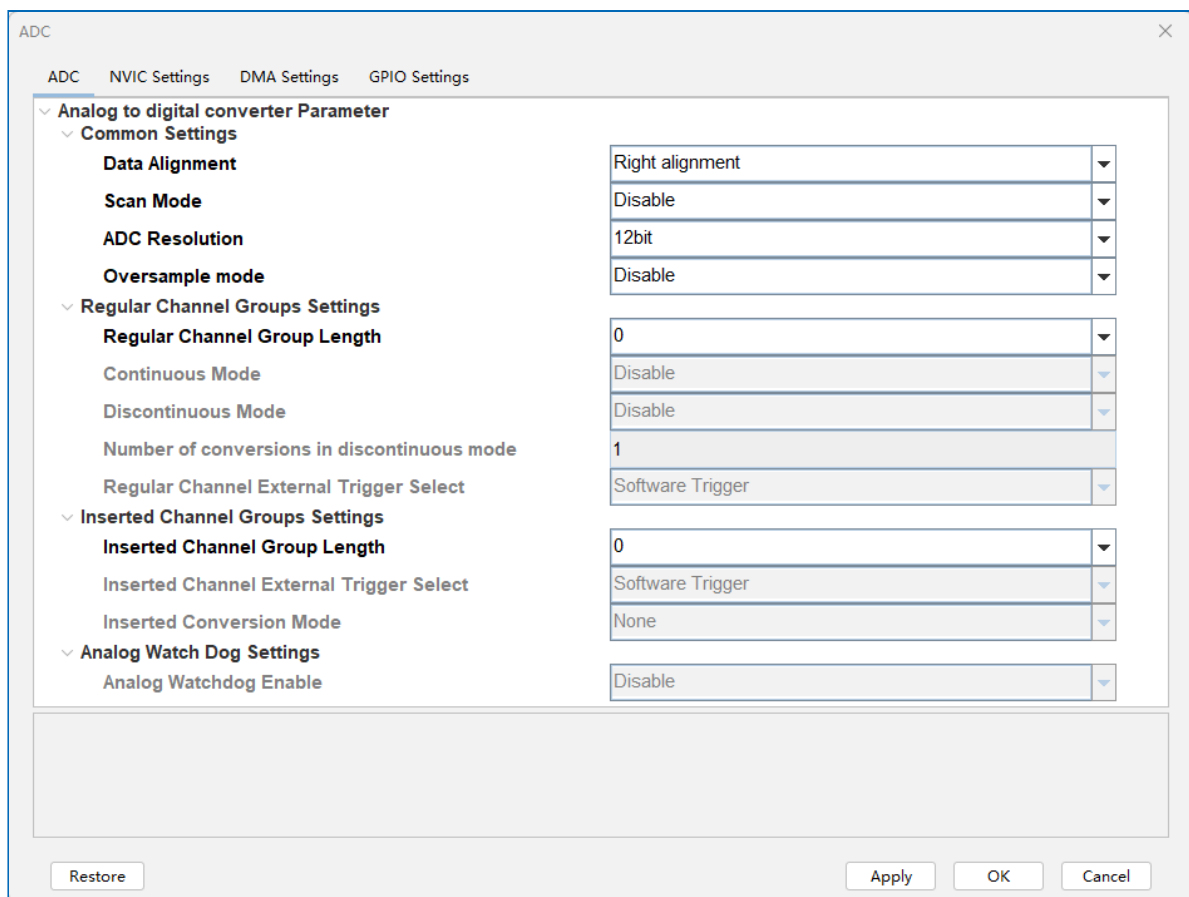
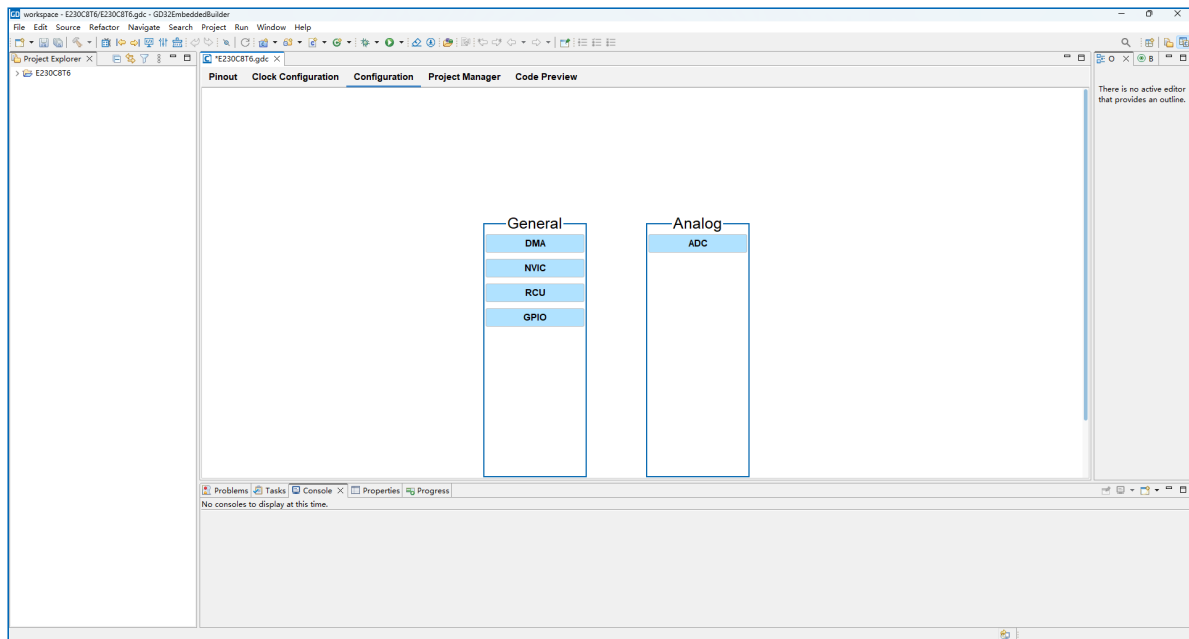
Pink indicates that the pin can be configured for multiple functions at the same time.

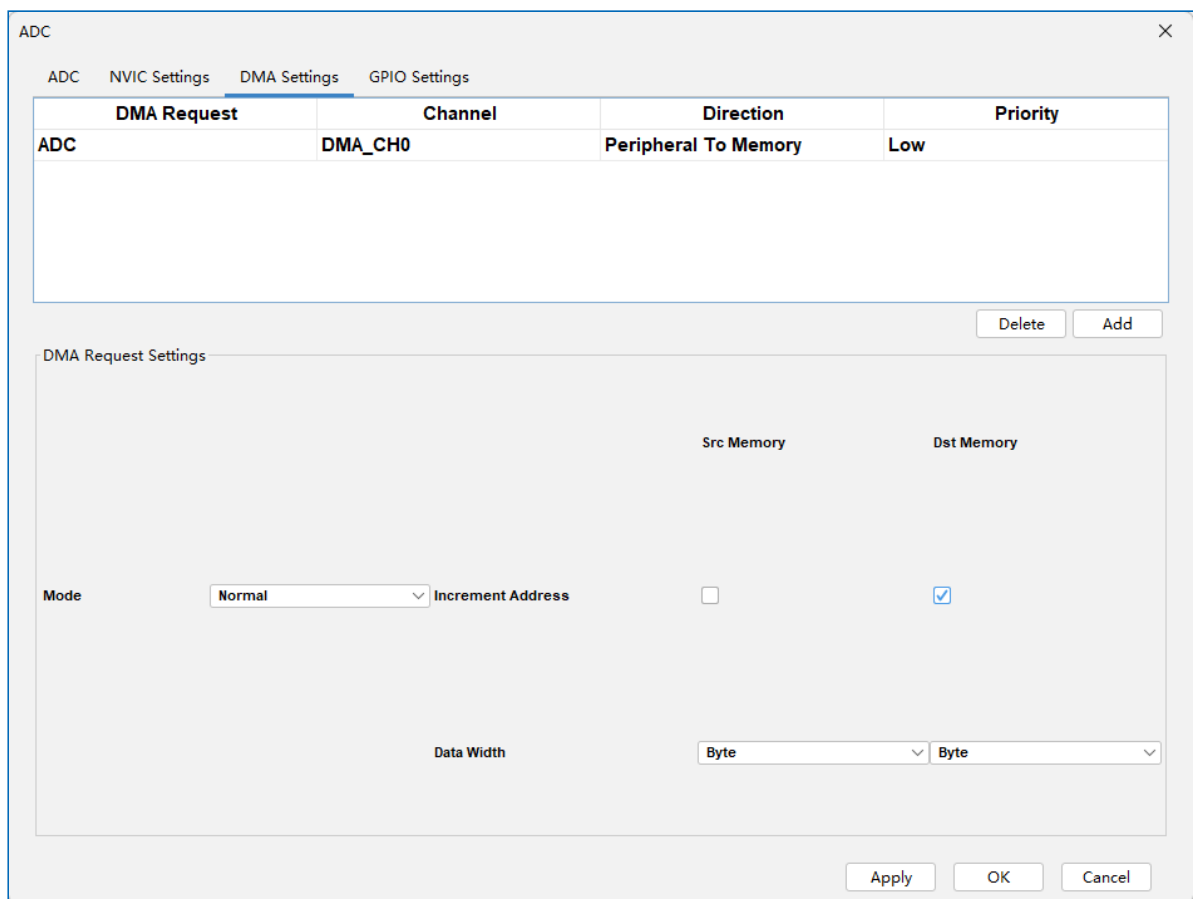


GD32 Embedded Builder presents the initial clock configuration, enabling users to manually adjust the clock parameters as needed.

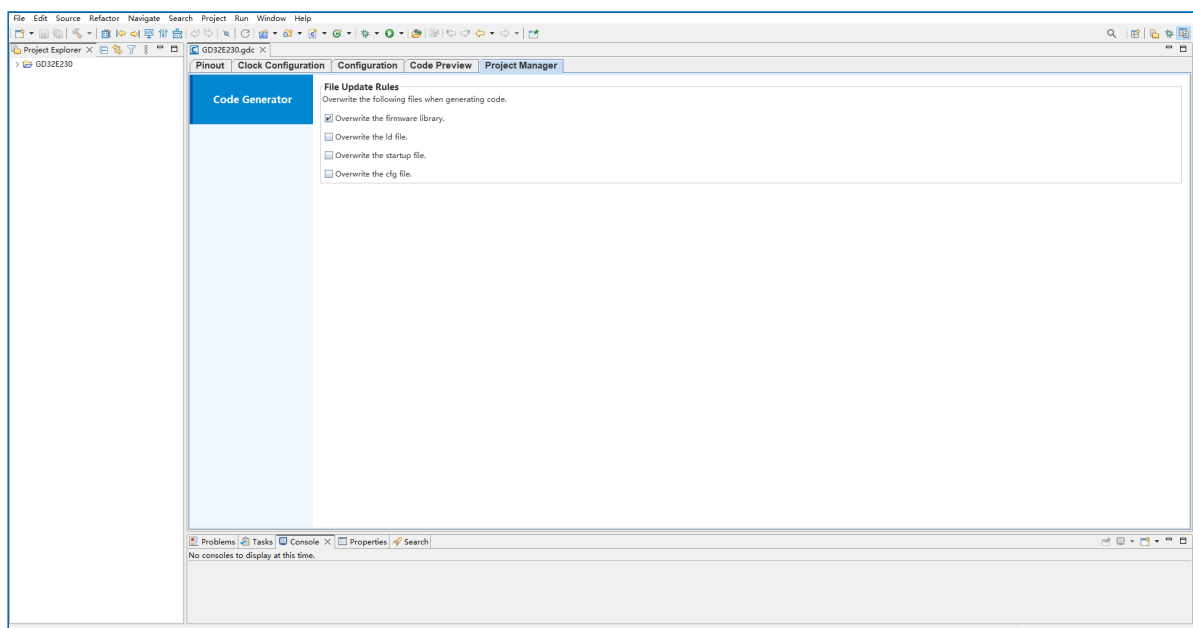


GD32 Embedded Builder facilitates configuration of peripheral modules, allowing users to customize peripheral parameters. It encompasses the setup of NVIC, DMA and GPIO parameters.

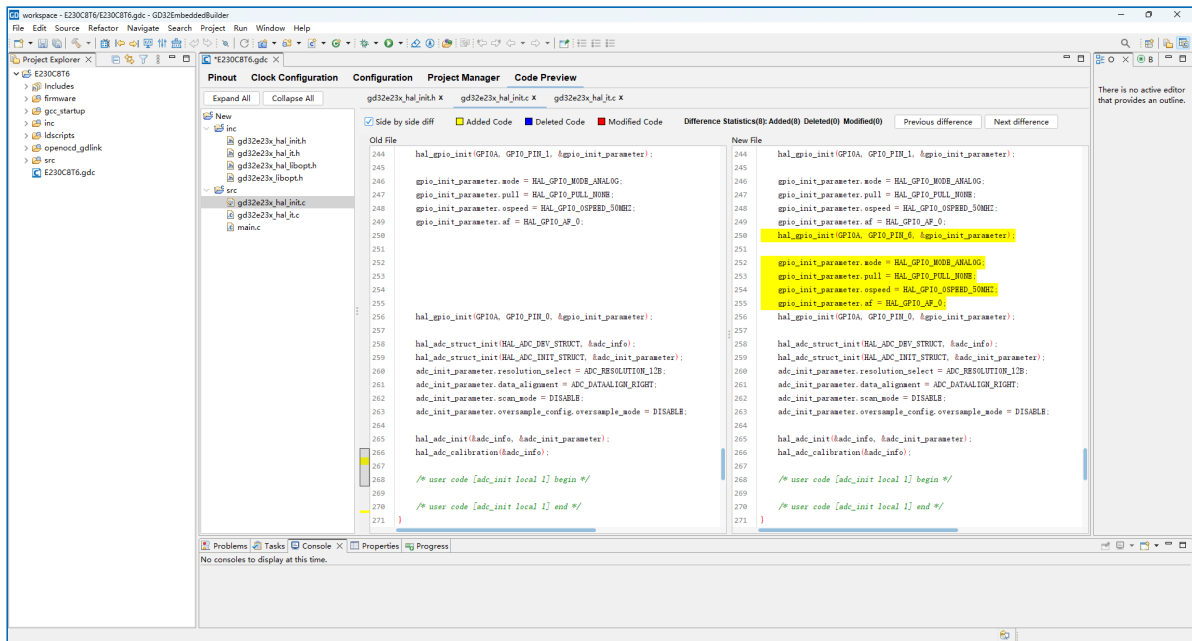




GD32 Embedded Builder provides code generation management functionality, allowing users to decide, based on their needs, which files are overwritten as initial files during code generation.

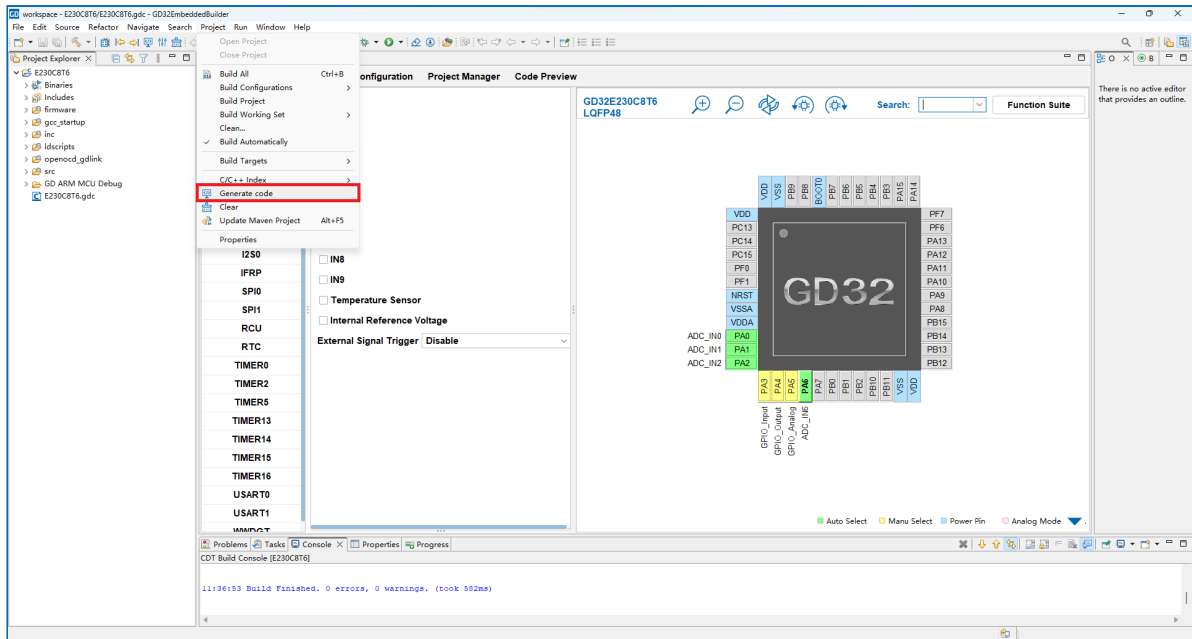


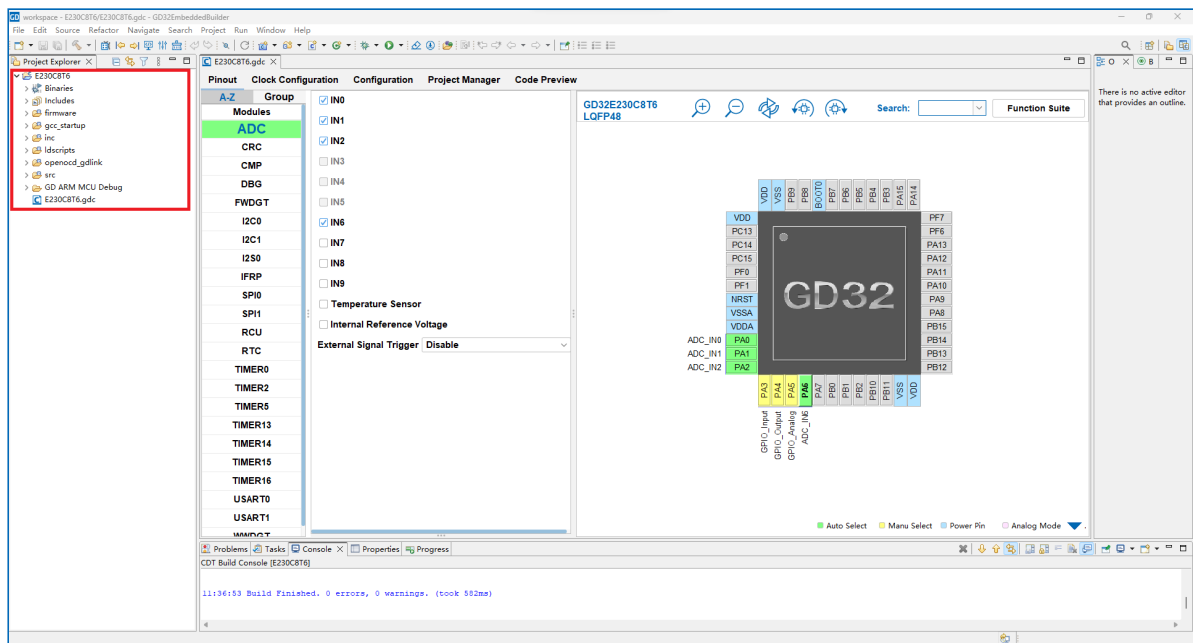
GD32 Embedded Builder provides a code preview feature, allowing users to view the differences between pre-generated code based on current project configurations and existing project code.



2.1.1.2. Code Generation

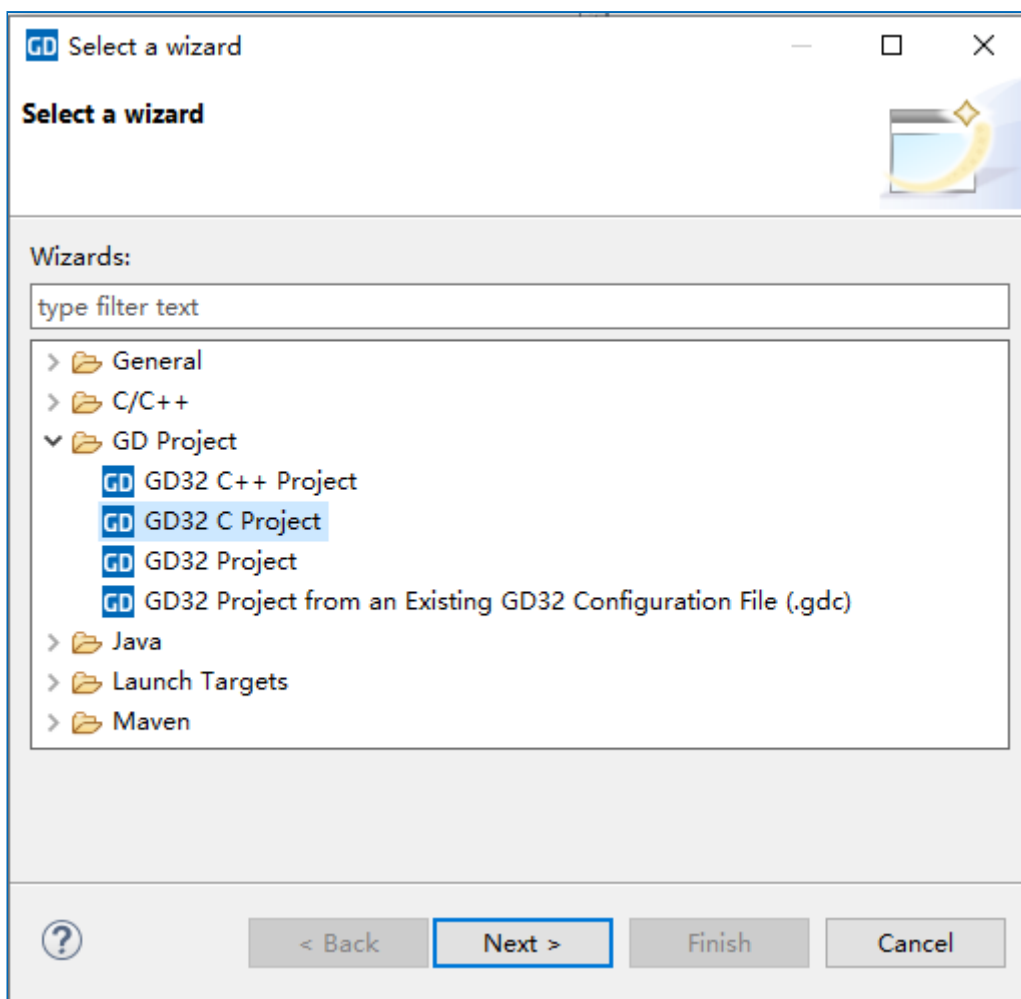
To initiate code generation, double-click the **gdc** file and navigate to **Project > Generate Code** within the menu. Upon successful code generation, GD32 Embedded Builder will display a confirmation prompt.



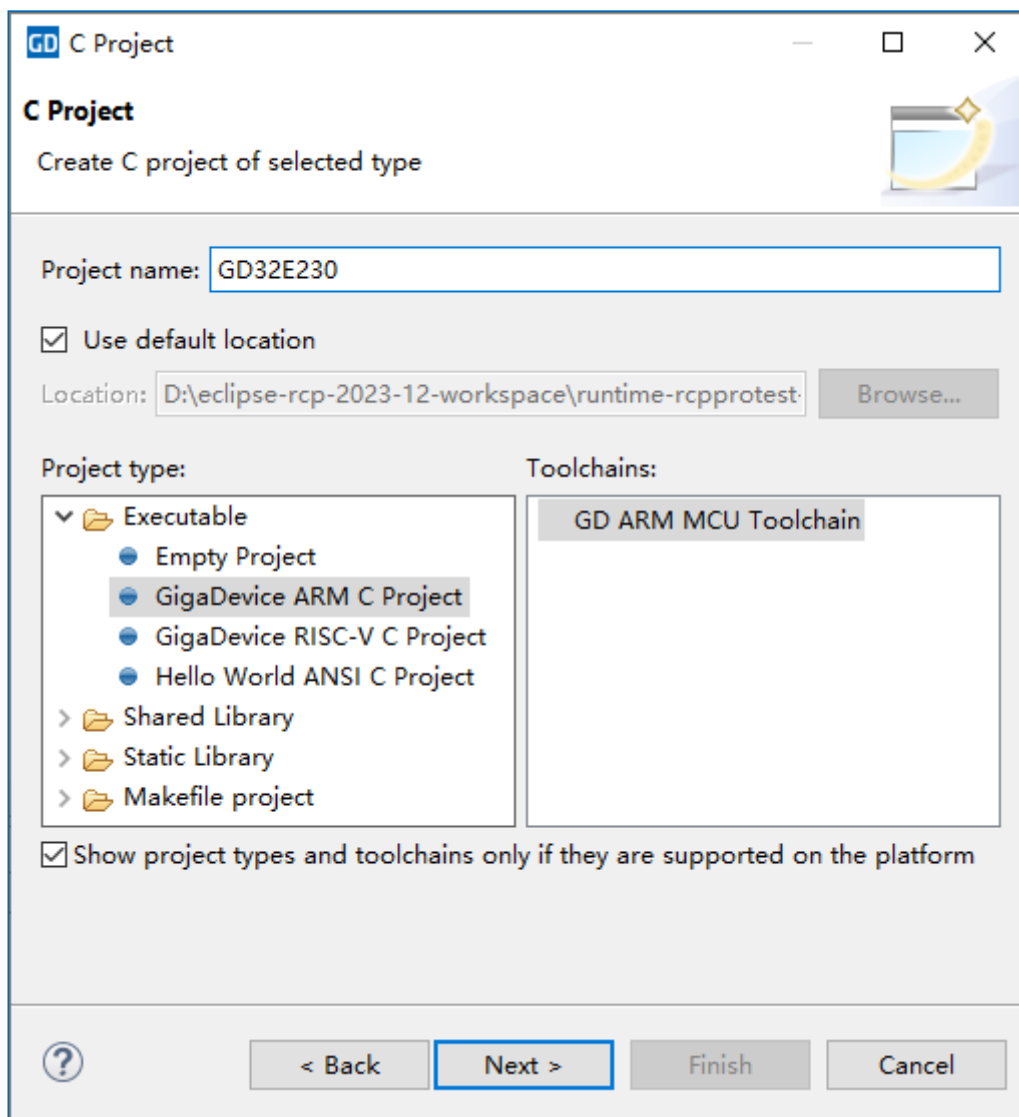


2.1.2. Creating a C project


1) Navigate to **File > New > Other > GD Project > GD32 C Project** menu command to start the new project wizard, Click **Next**.



2) Select to create **GigaDevice ARM C Project** or **GigaDevice RISC-V C Project**, and the compilation chain on the right will automatically associate with **GD ARM MCU Toolchain** or **GD RISC-V MCU Toolchain**. Then, enter a name in **Project name** field, Click **Next**.



- 3) Select a required device from the **Device** list, click **Finish**.

 C Project

Target processor settings
Select the target processor family and define flash and RAM sizes.

Device	Description
GD32E230K4U6	Description
GD32E230K6U6	Description
GD32E230K8U6	Description
GD32E230K4T6	Description
GD32E230K6T6	Description
GD32E230K8T6	Description
GD32E230C4T6	Description
GD32E230C6T6	Description

Flash size(kB)

16

RAM size(kB)

4

Code location

FLASH

Floating point unit

☐

Check some warnings

☐

Check most warnings

☐

Enable -Werror

☐

Use -Og on debug

☒

Use newlib nano

☒

Exclude unused

☒

Use link optimization

☐

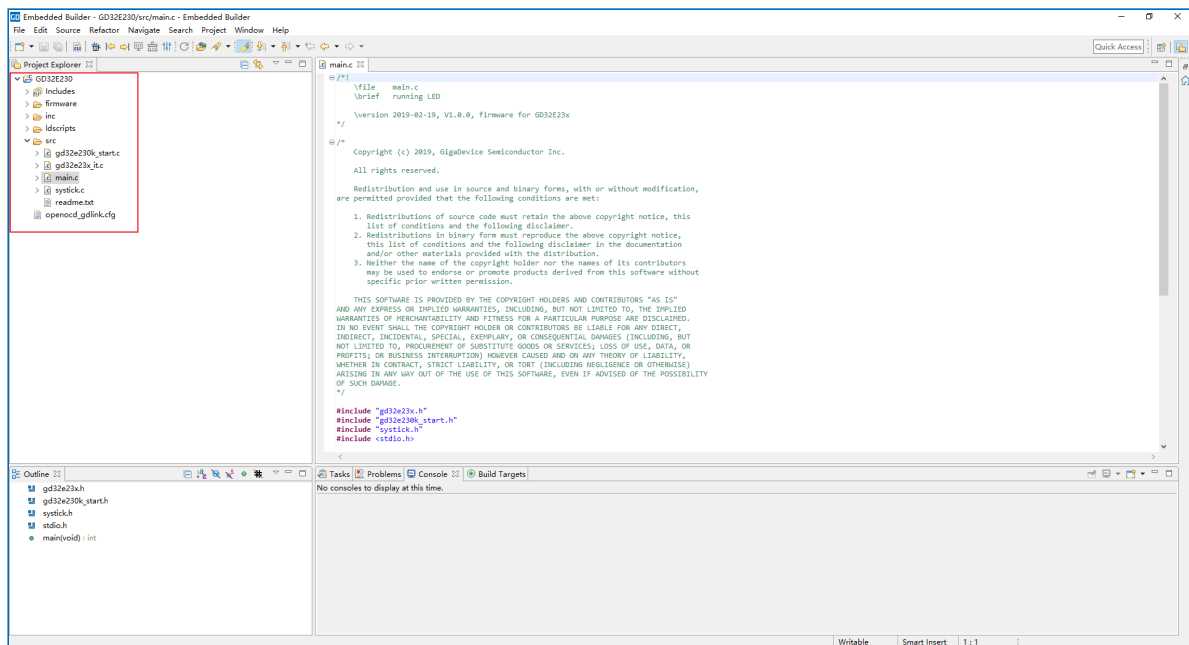
?

< Back

Next >

Finish

Cancel

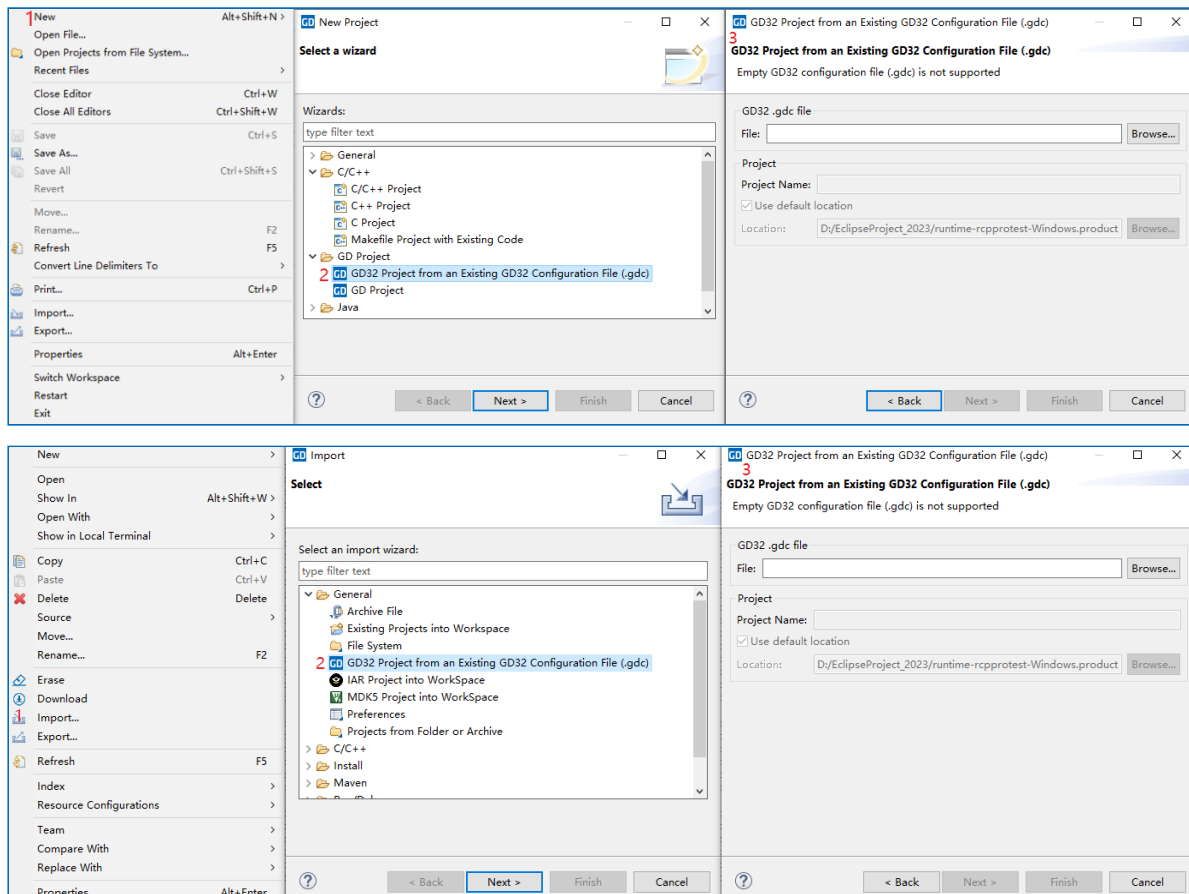


2.2. Importing project

2.2.1. Importing a GD project

1) There are two ways to select this feature.

Navigate to **File > New > Other > GD32 Project from an Existing GD32 Configuration File(.gdc)** within the menu commands, or Select **File > Import**, In the popup window, choose **General > GD32 Project from an Existing GD32 Configuration File(.gdc)** and click Next. Click the **Next** button to continue.



2) Select the **GD** project file. Click the "**Browse**" button and choose the **GD** project file with the **.gdc** extension. By default, the project name from the **GD** file will be used, but you can change the project name as needed. (There cannot be projects with duplicate names in the workspace; if there are duplicate names, you will need to change the project name.) In addition, you can modify the save path for the **GD** project file. Click **Finish** to proceed with the project conversion.

GD32 Project from an Existing GD32 Configuration File (.gdc)

GD32 Project from an Existing GD32 Configuration File (.gdc)

⚠ Directory with specified name already exists

GD32 .gdc file

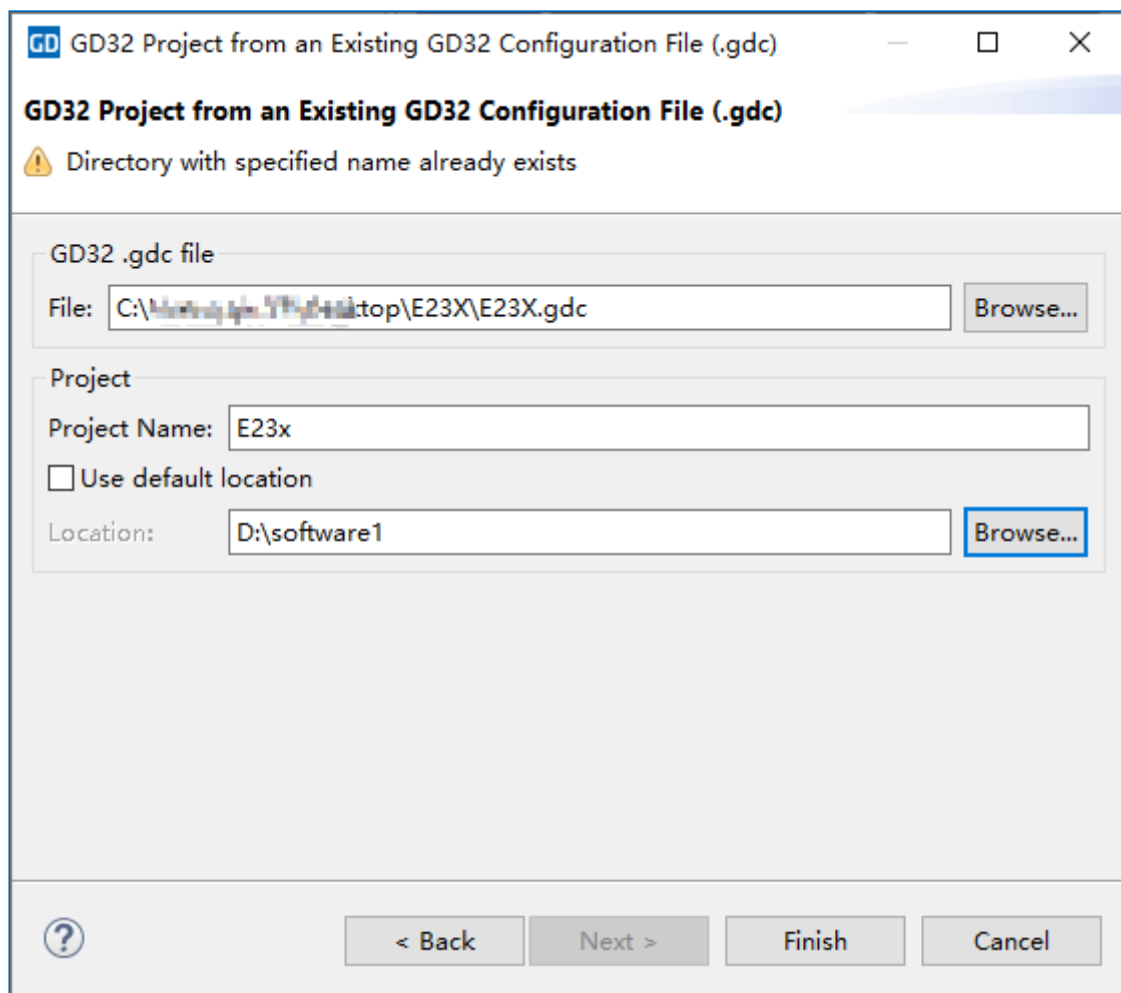
File:

Project

Project Name:

☒ Use default location

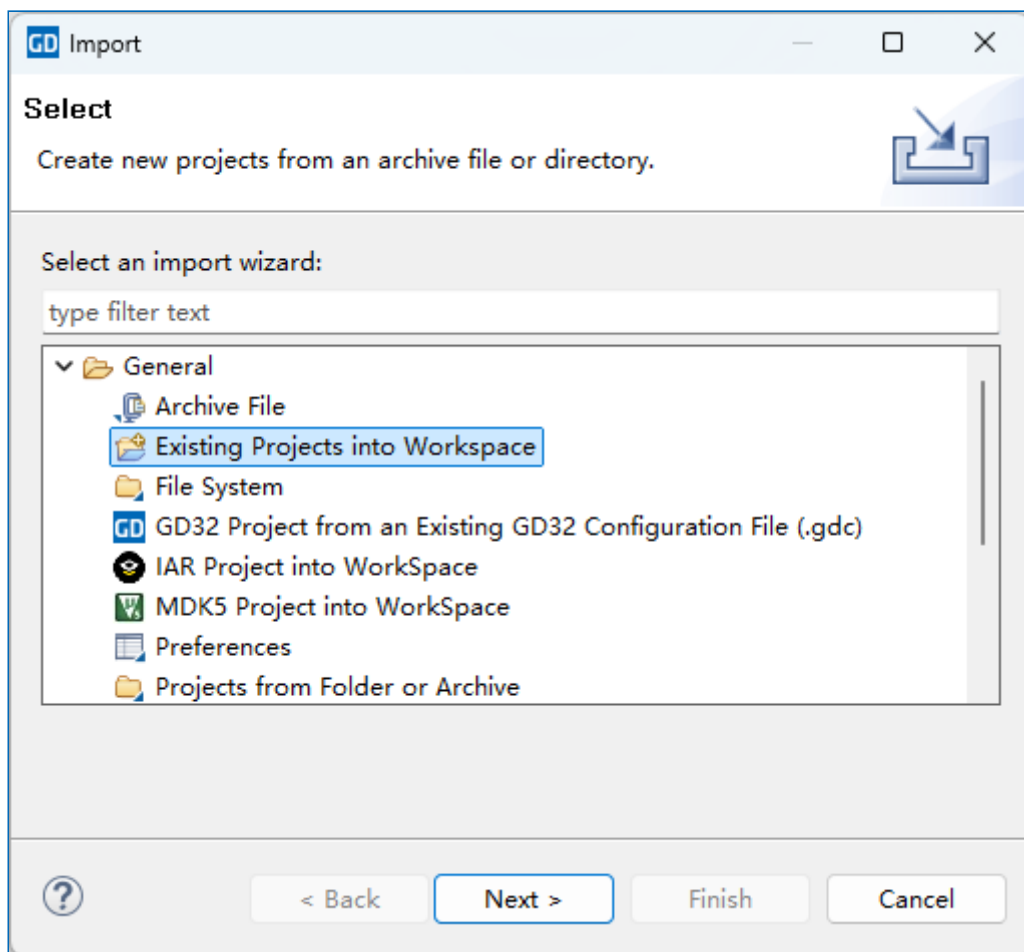
Location:

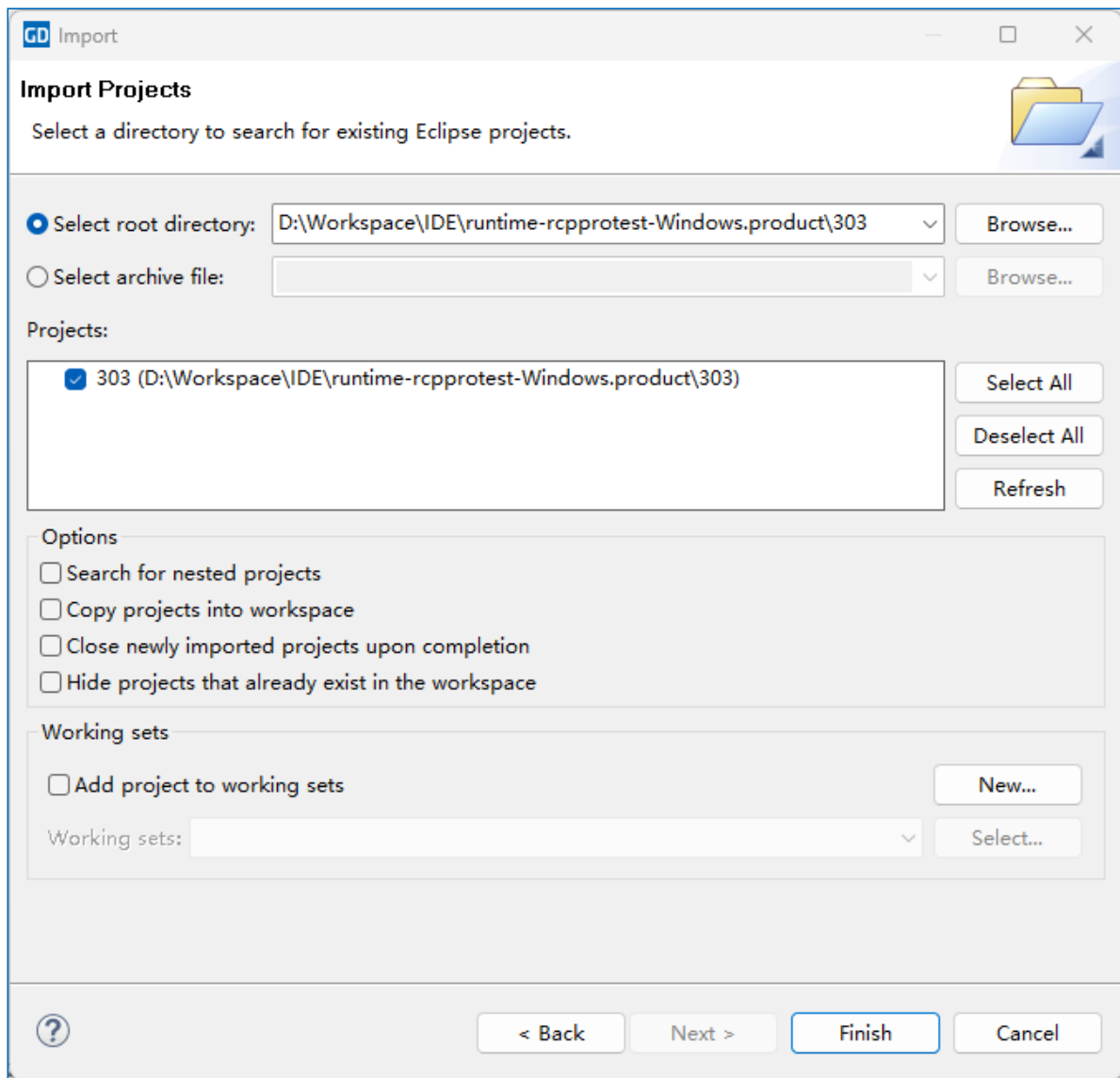


2.2.2. Importing a C project

There are two ways to select this feature.

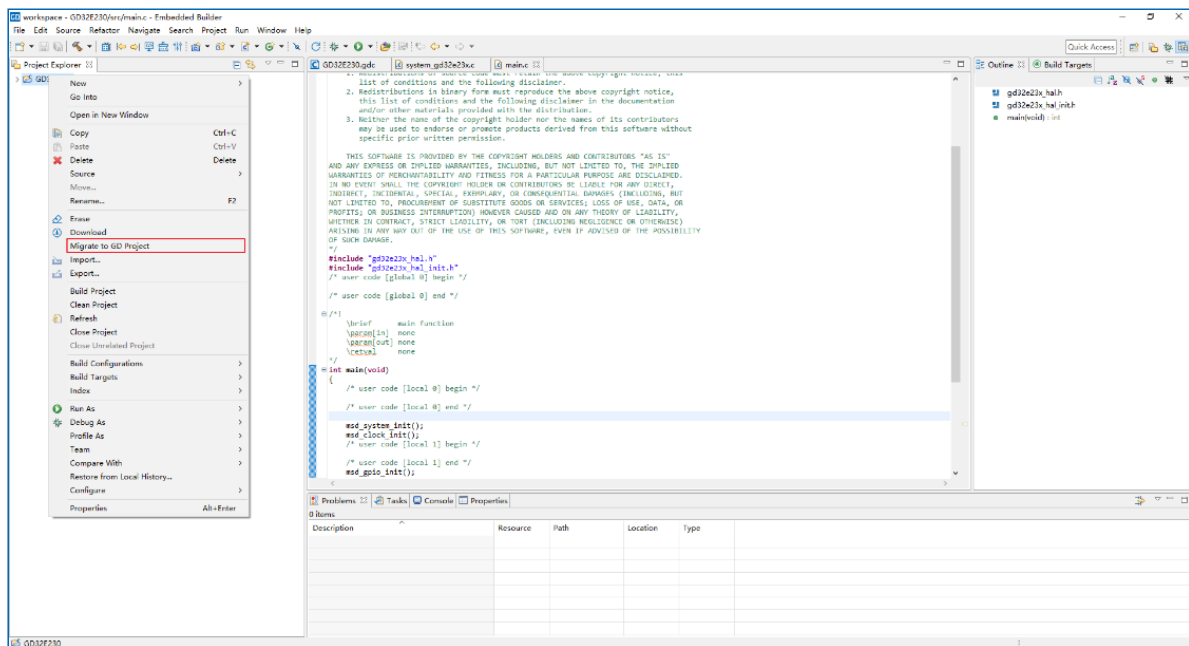
- One way to open the Import dialog is to use the menu [File]>[Import...]
- Another way is to right-click the Project Explorer view and select [Import...]



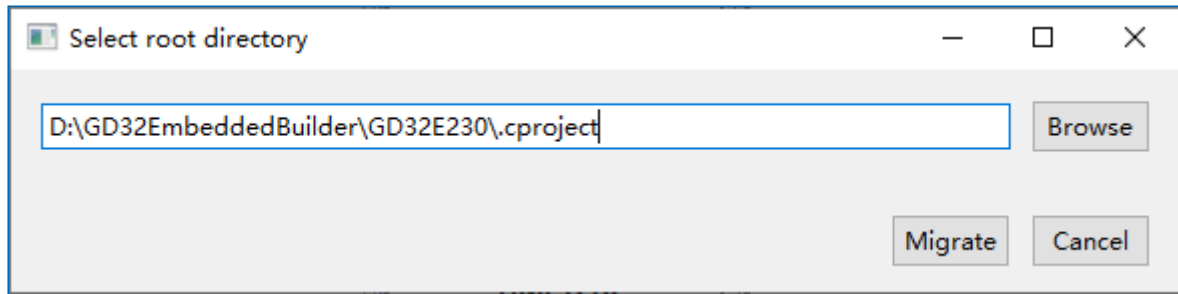


2.2.3. Importing a GNU project

- 1) Right-click in the **Project Explorer** view of **C/C++** perspective to select **Migrate to GD Project** wizard.

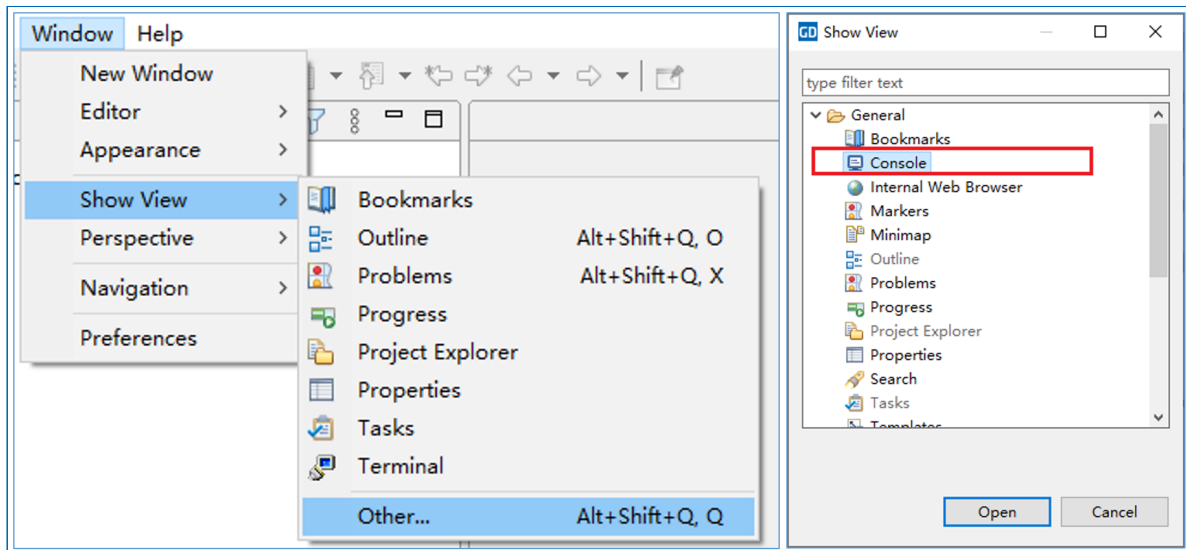


- 2) Select the file with the suffix “cproject” of the GNU project which you want to convert. Click **Finish** button to start the conversion process. If the conversion is successful, you can import and build the GNU project as usual.

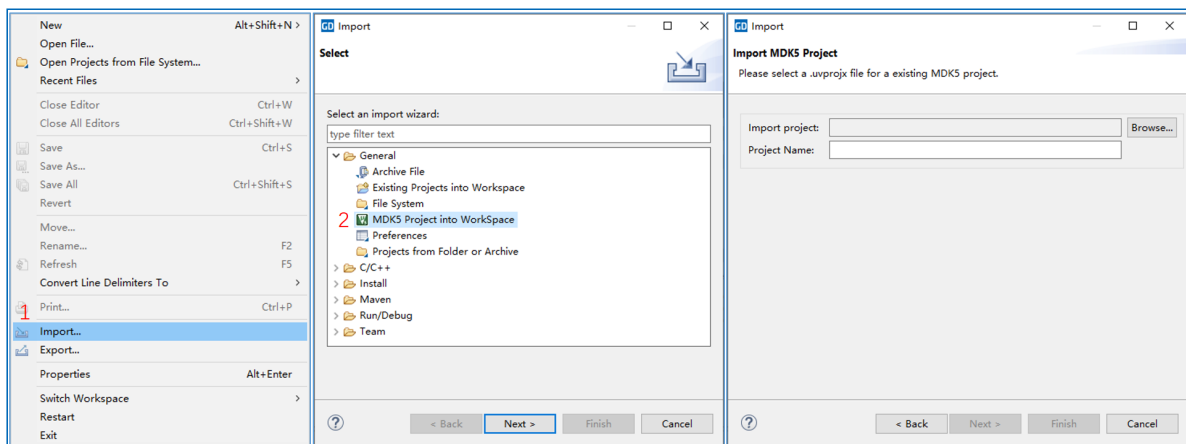


2.2.4. Importing a MDK5 project

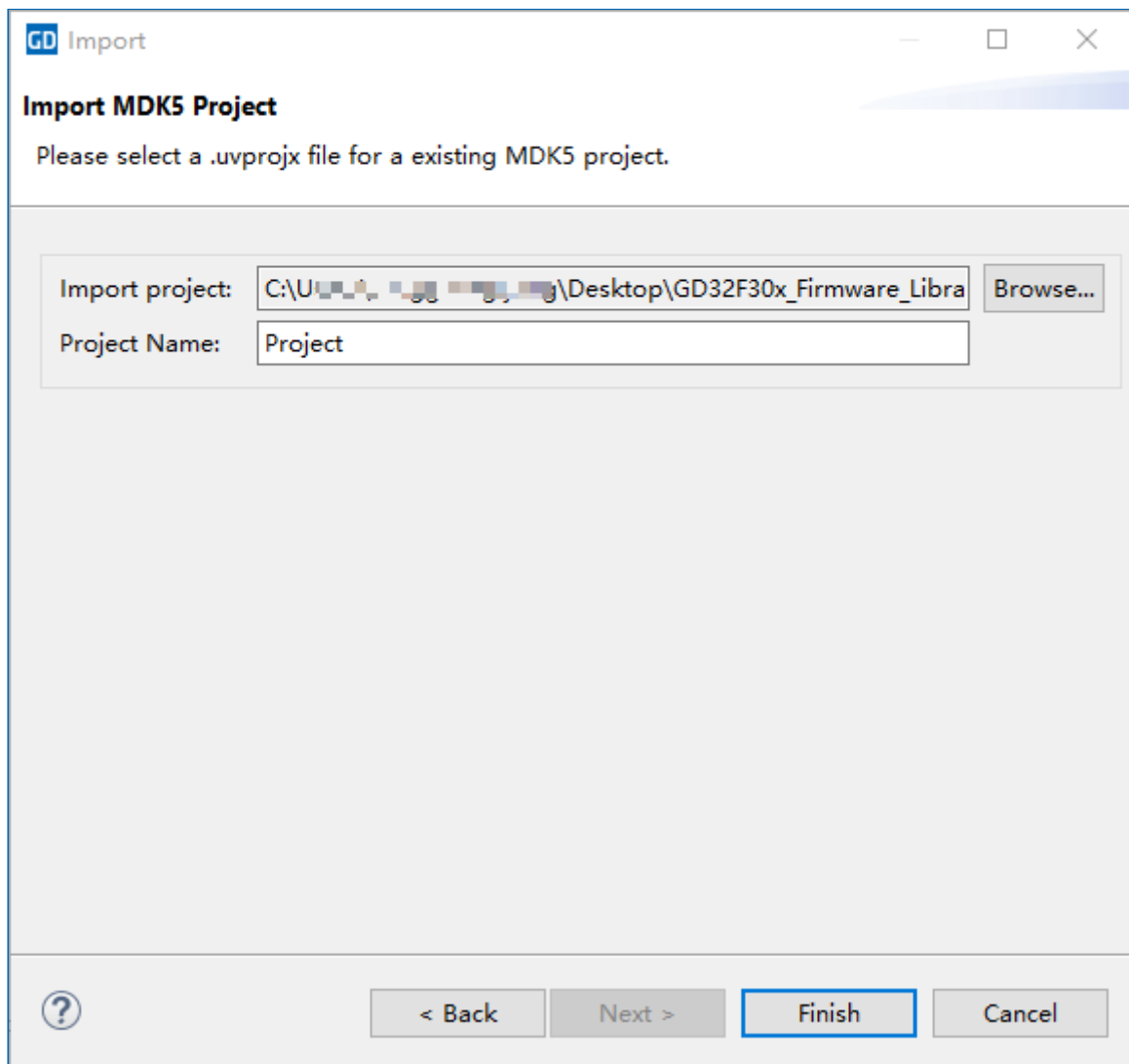
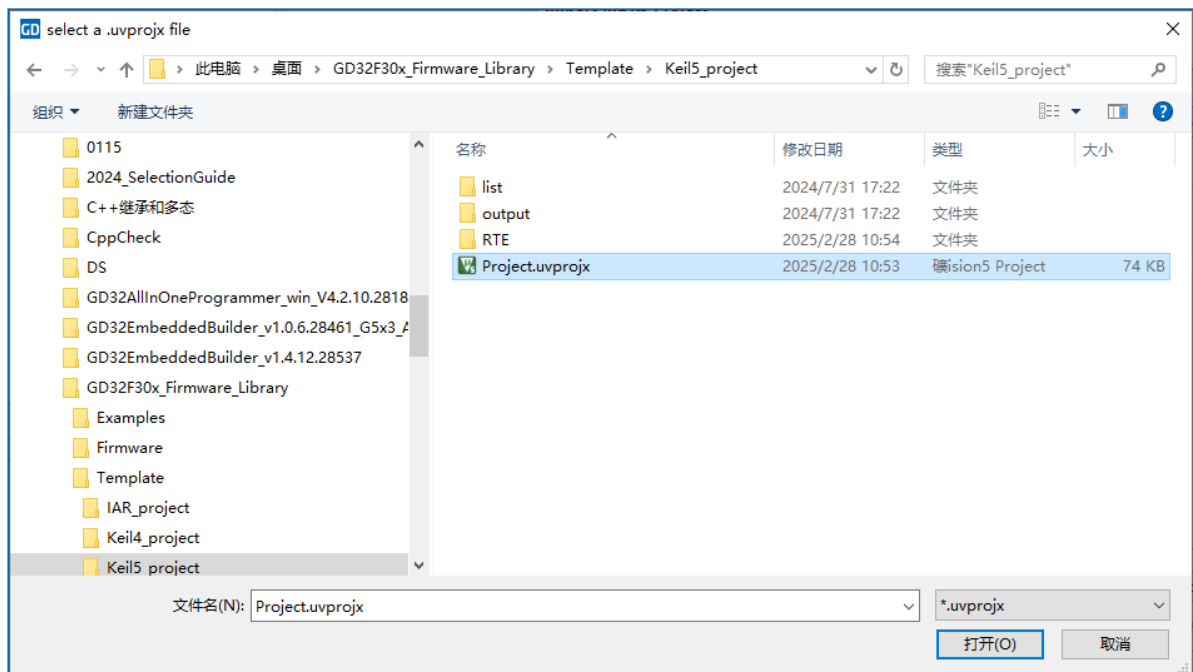
- 1) Select **Window > Show View > Other**. In the popup window, choose **Console**. This will bring up the console to display outputs during the conversion process. (If the console is already open, you can skip this step.)



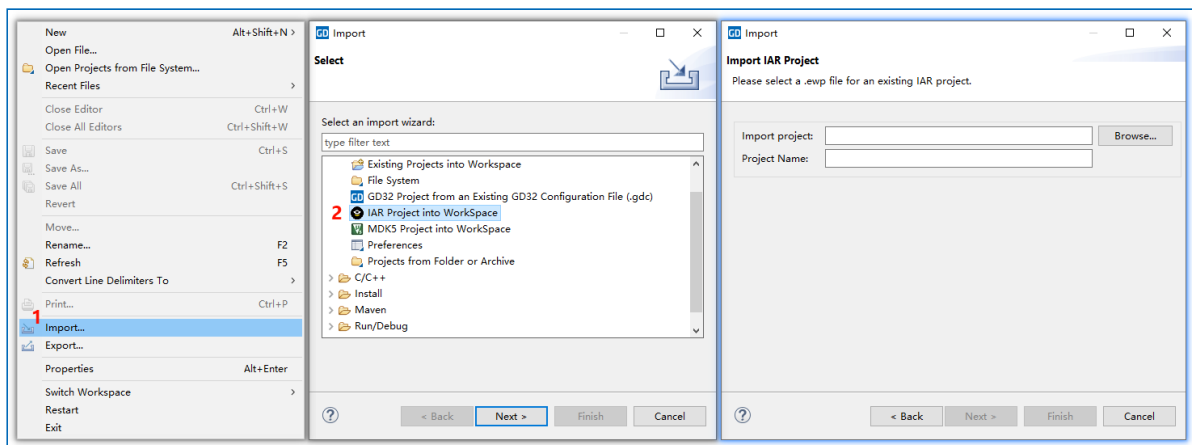
- 2) Select **File > Import**, In the popup window, choose **General > MDK5 Project into Workspace(1)** and click **Next**. Then, select the MDK5 project file that you need to convert.



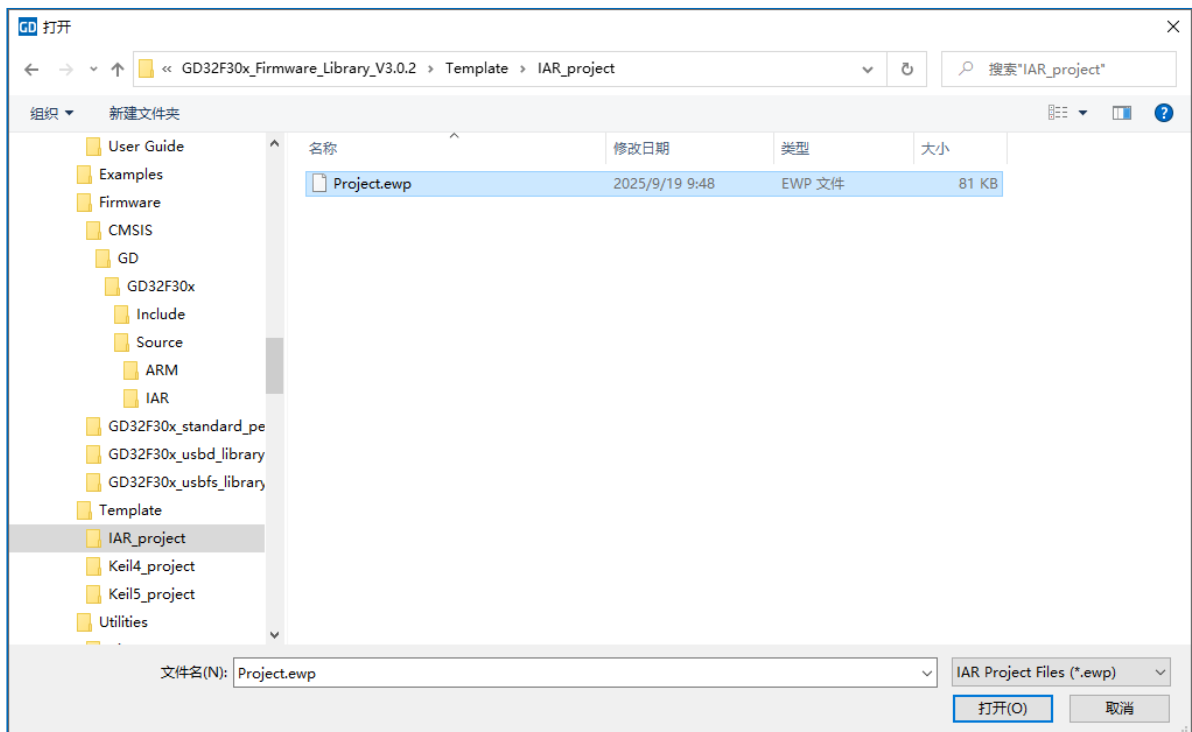
- 3) Select the MDK5 project file. Click the **Browse** button and choose the MDK5 project file with the extension **.uvprojx**. By default, the project name from MDK5 will be used, but you can change the project name as needed. (There cannot be projects with duplicate names in the workspace; projects with duplicate names will need a name change.) Click **Finish** to proceed with the project conversion.

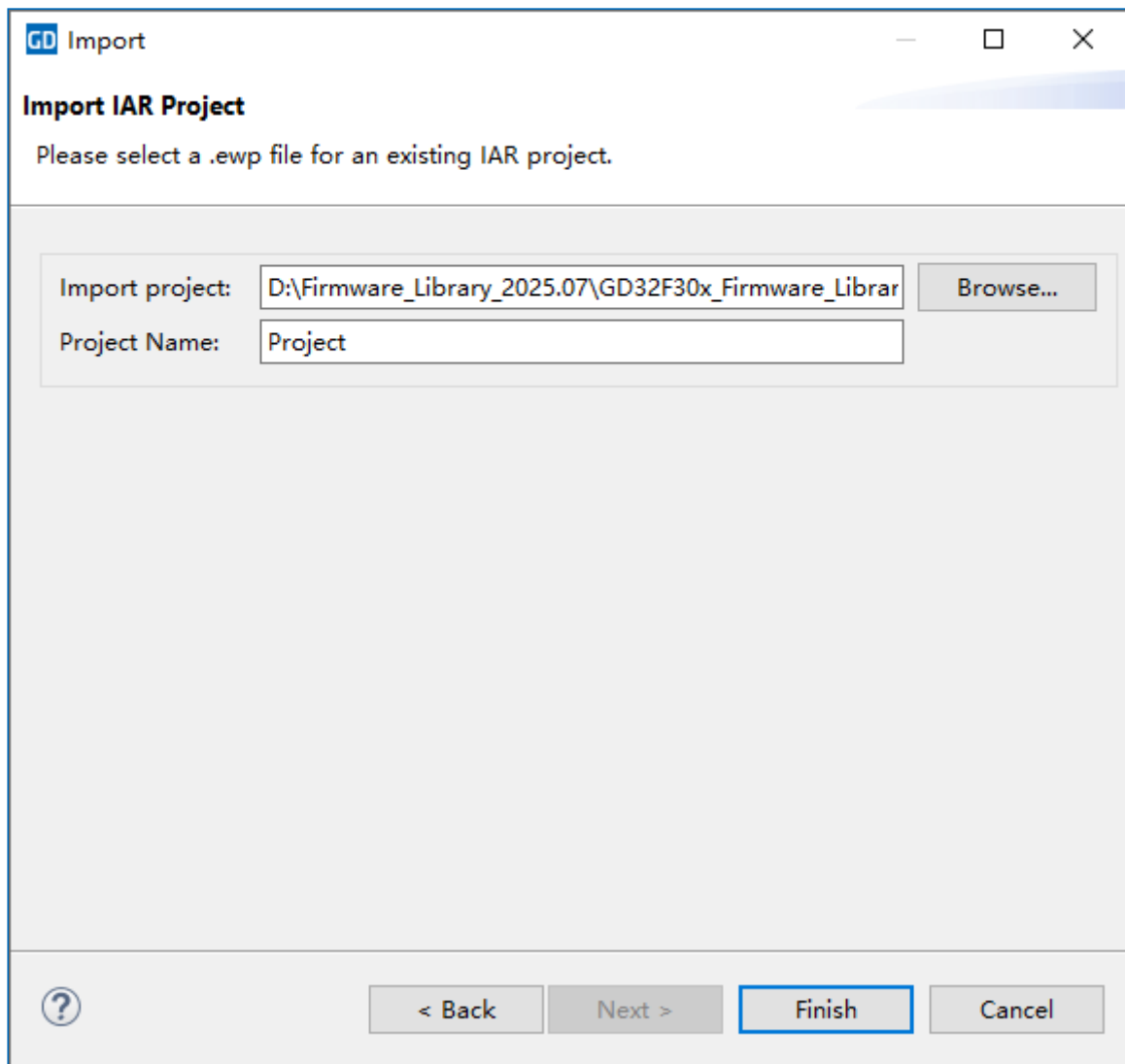


- 4) After completing the conversion, create a directory named **GD32_Eclipse_project** one level above the *.uvprojx project. This will organize the converted project. The files in the project are linked to the files pointed by the *.uvprojx.

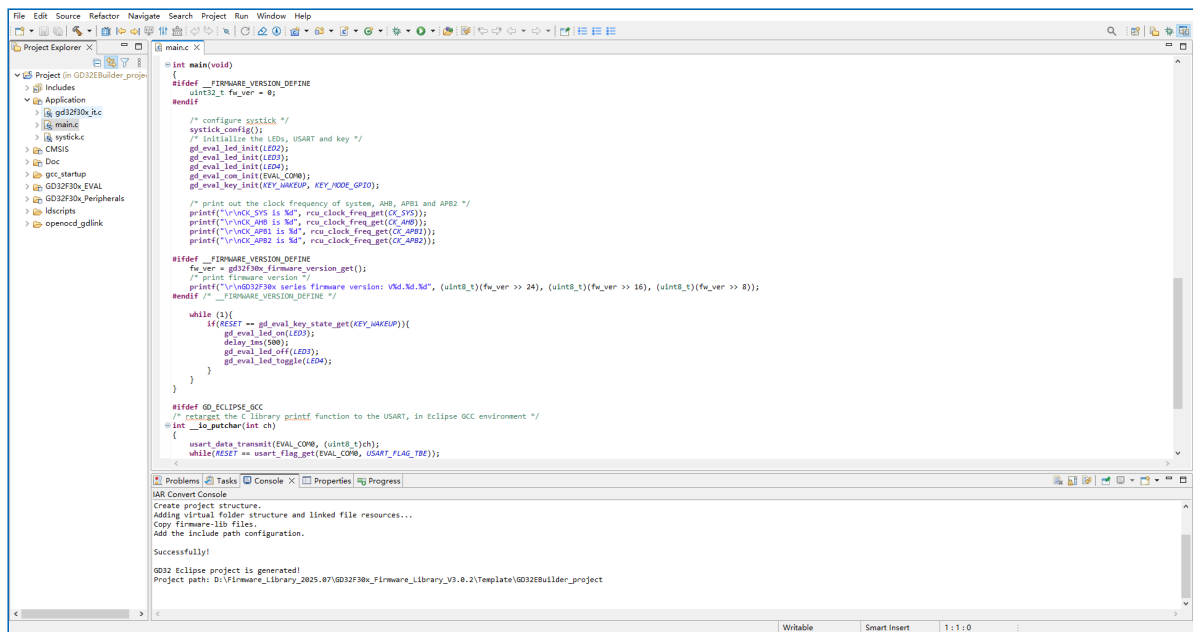


3) Select the IAR project file. Click the **Browse** button and choose the IAR project file with the extension **.ewp**. By default, the original IAR project name will be used, but you can change the project name as needed. (There cannot be projects with duplicate names in the workspace; projects with duplicate names will need a name change.) Click **Finish** to proceed with the project conversion.





- 4) After completing the conversion, create a directory named **GD32_Eclipse_project** one level above the *.ewp project. This will organize the converted project. The files in the Eclipse project are linked to the original source files referenced by the *.ewp configuration.



Note:

This import feature is only supported on Windows operating systems.

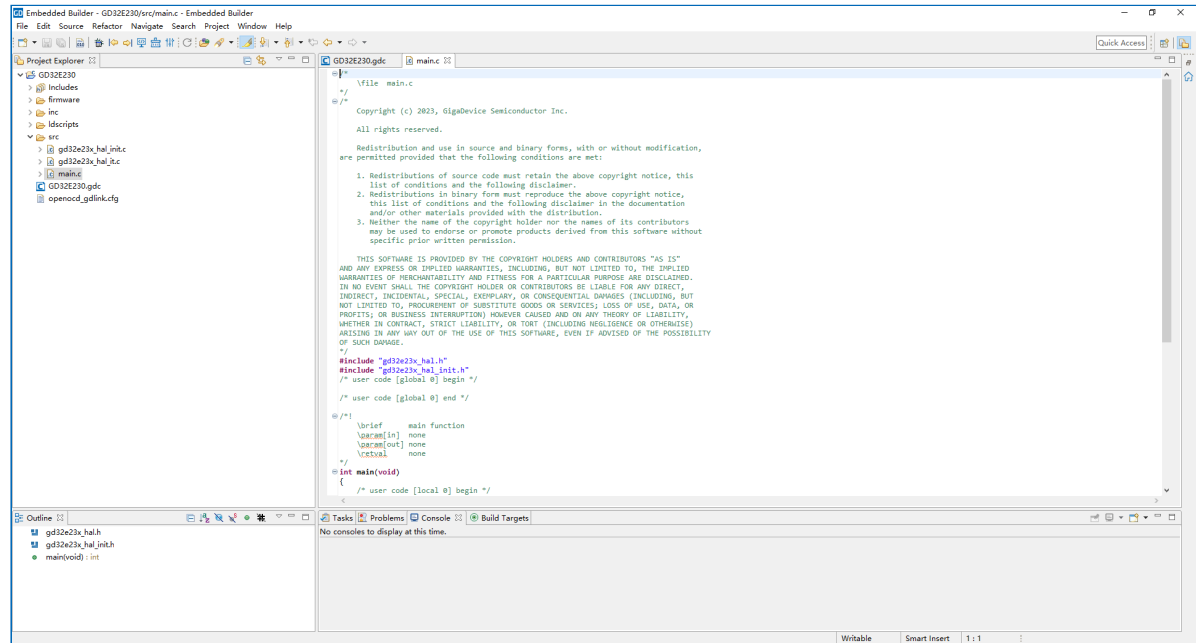
If the IAR workspace (*.eww) contains multiple projects, repeat the import for each required *.ewp or import one and then add others as needed. Relative include paths and preprocessor macros defined in IAR will be translated where possible; verify them in the project properties after import.

3. Building project

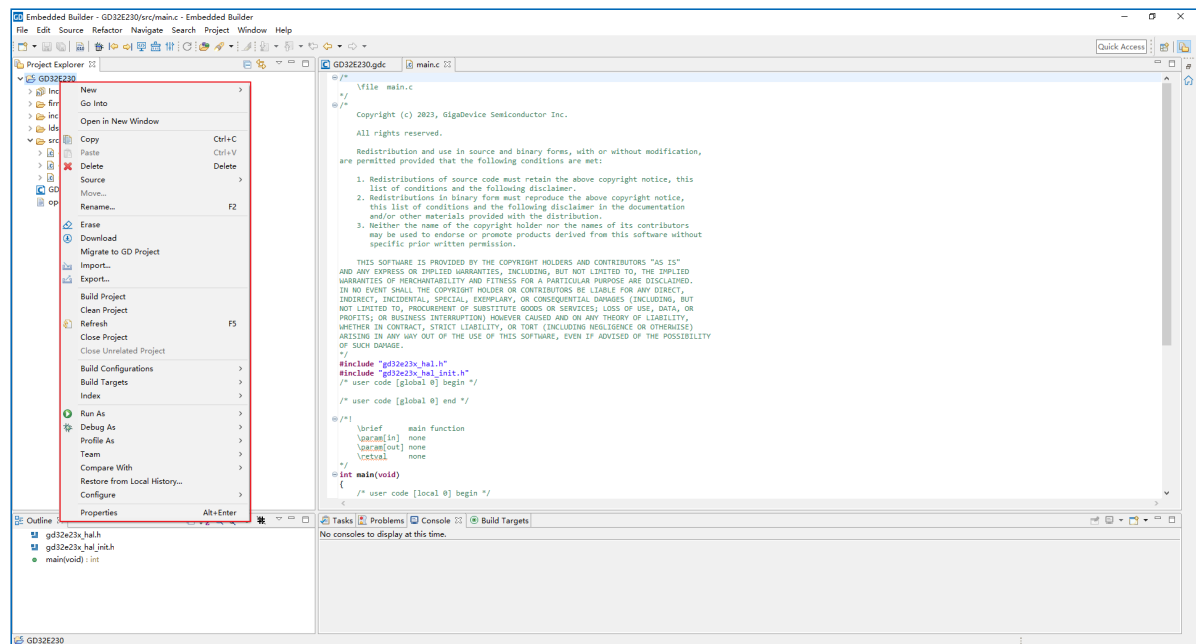
3.1. Framework of Project

A project is created with the default settings and a full set of configurations based on the project type and the device you selected. **C/C++** perspective is opened by default after the project is created.

Expand the project folder to display the project framework, which includes the GD firmware, linker file and peripheral configuration code.



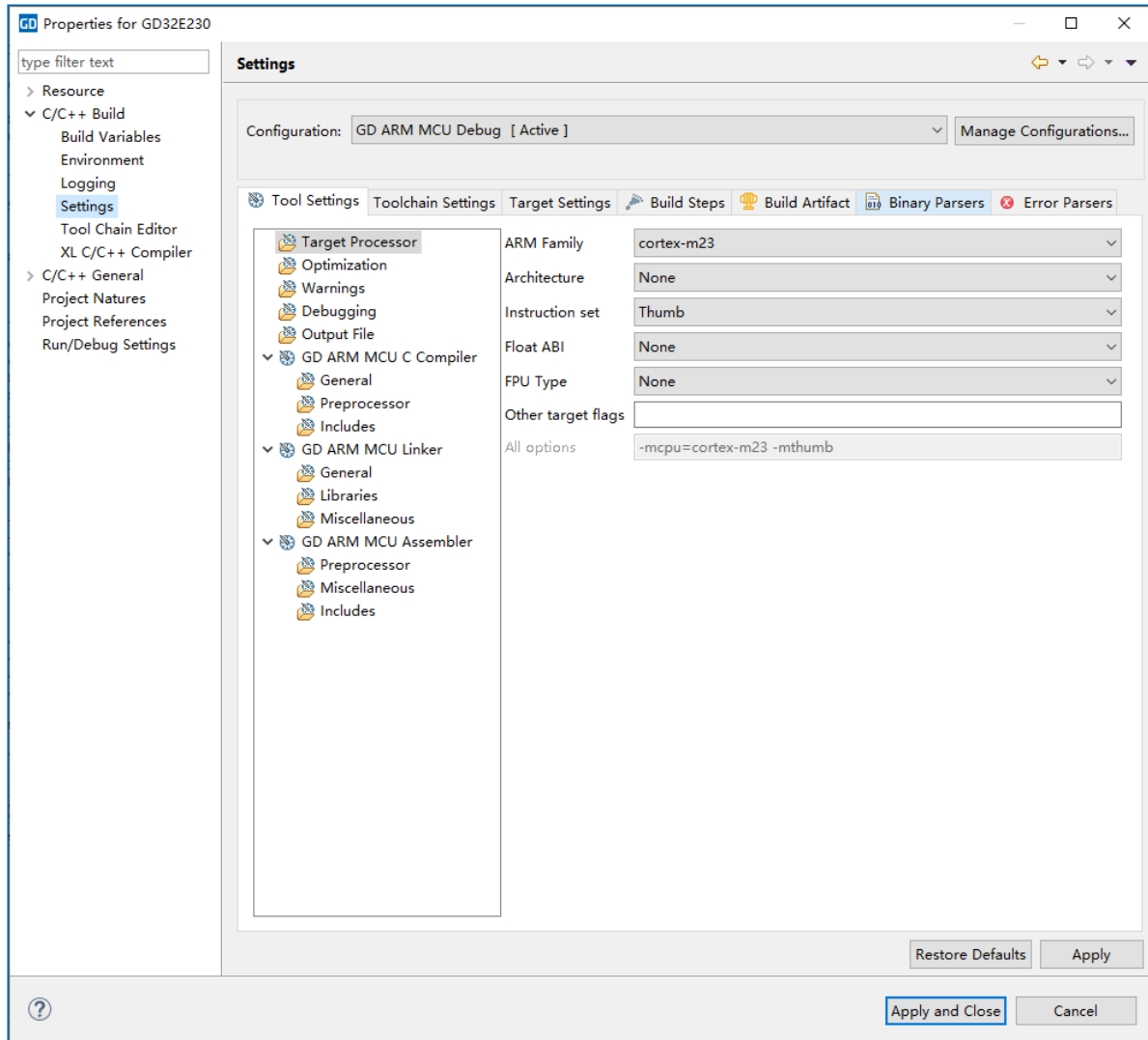
In the **Project Explorer** view, right-click the menu of the target project which includes functions of creating a new file, building a project, opening the project properties dialog and so on.



3.2. Build Configuration

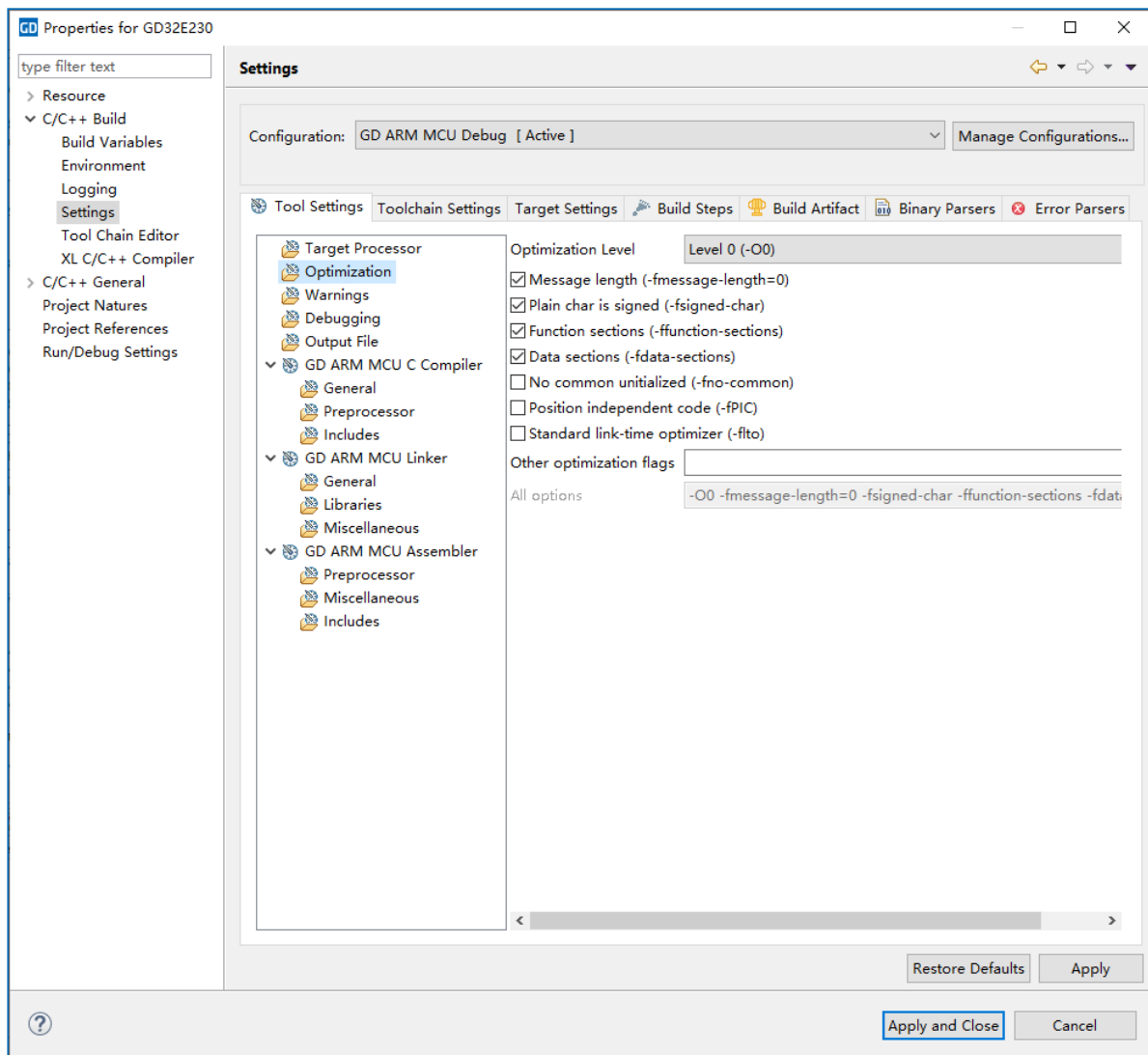
In the project properties dialog, you can see the default settings and configurations, such as include paths, defined symbols, the linker script file and the toolchain path. Depending on the default settings and configurations, the project can be compiled and linked correctly.

Use the **Tool Settings** tab to configure the compiler and linker options.

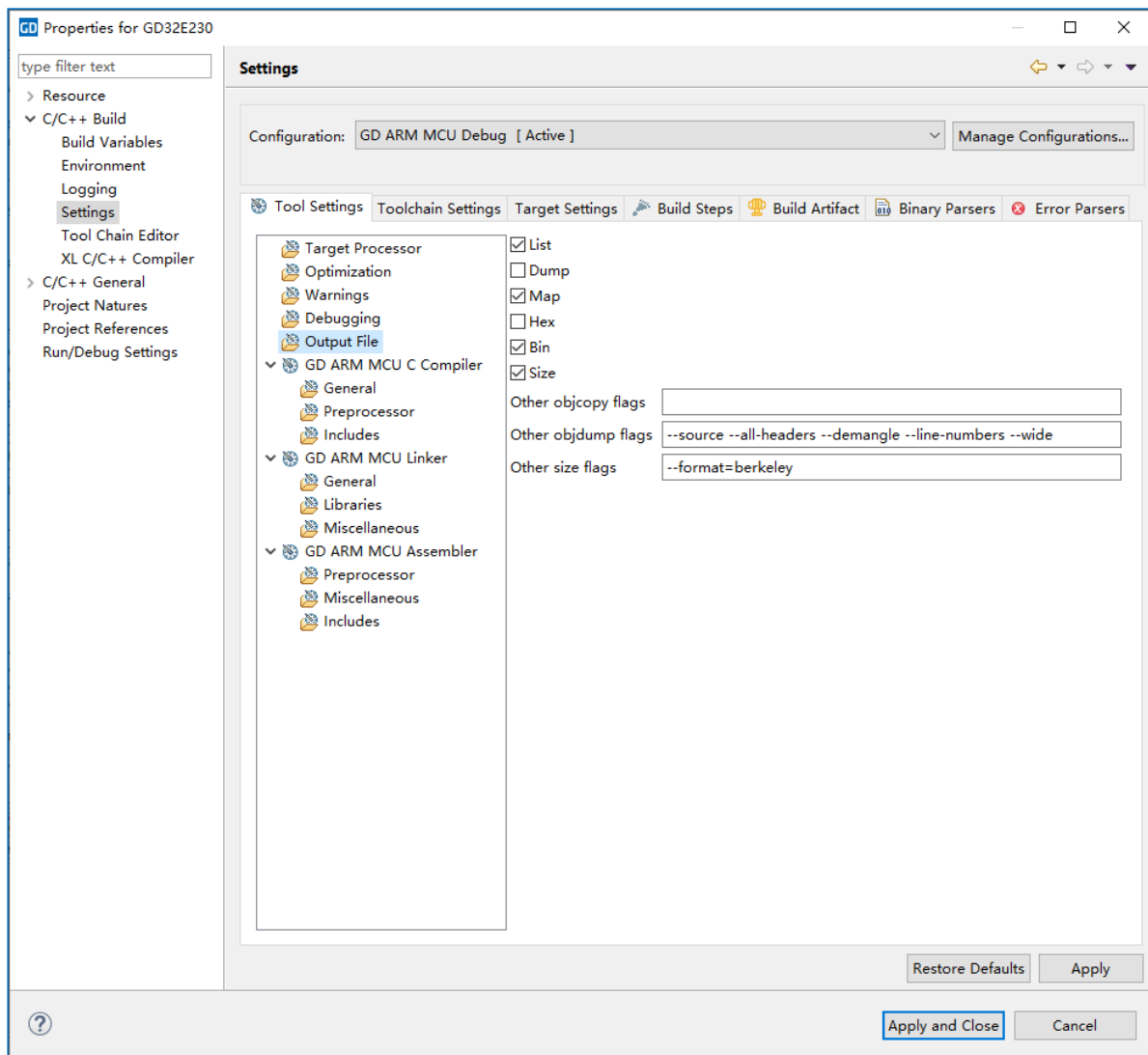


The **Target Processor** category is used to configure the compiler options related to the target processor.

The **Optimization** category is used to configure various sorts of optimization options. The **Warnings** category is used to configure the options to request or suppress warnings. The **Debugging** category is used to configure various special options which are used for debugging.



The **Output File** category is used to output files in other formats, such as list, dump, map, hex, bin and size file. You can enter some options in **Other objcopy flags** edit box to control the hex or bin file. The **Other objdump flags** edit box is used to configure the list options. The **Other size flags** edit box is used to enter some size options.



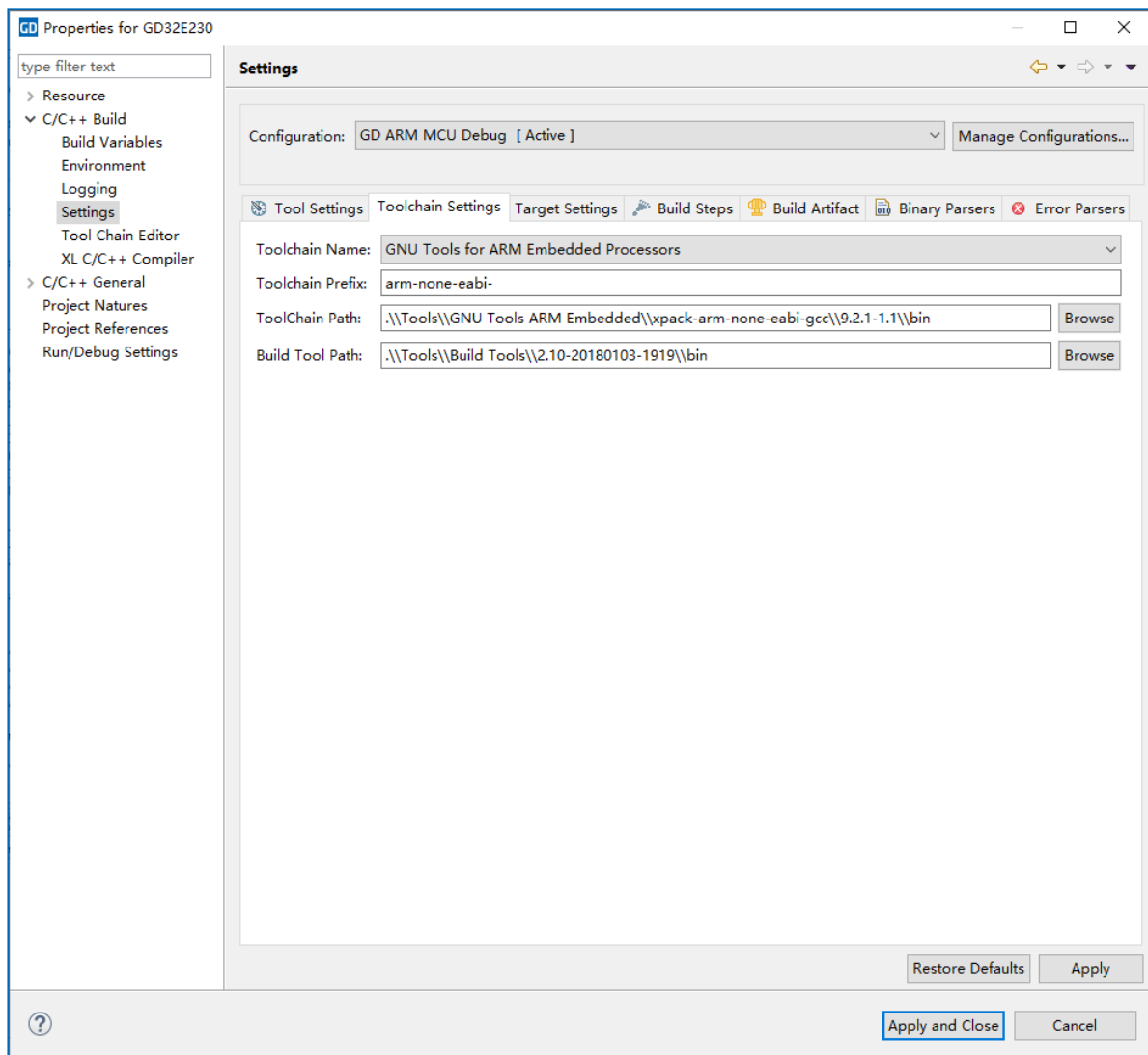
The **GD ARM MCU Assembler** tool is used to configure the options for the assembler.

The **GD ARM MCU C Compiler** tool is used to configure the options for C compiler. In the **General** category, you can configure the start address and length attributes of Flash and RAM, which are correspond to the memory configuration of the linker script file in the valid format.

3.3. Toolchain Configuration

Paths of the toolchain and build tool is the default configuration.

Select **Project > Properties > C/C++ Build > Settings**. In the **Toolchain Settings** tab, click **Browse** button to choose required toolchain path and the build tool path. Then, click **Apply** or **Apply and Close** button.



3.4. Modify Toolchain

Right click the project name, click **Properties**, then click **Properties > C/C++ Build > Settings > Toolchain Settings**, modify the content in the **Toolchain Prefix** to the new toolchain prefix, then modify the **ToolChain Path** to the **new toolchain path**, then click **Apply**. When debugging, please open **Debug Configuration** of the debugging project, then click **Apply** and **Debug**.

3.5. Headless build

3.5.1. Windows platform

The "headless-build.bat" file under the project encapsulates some commands used for executing the headless compilation function, allowing you to compile a specified project from the command line without opening the graphical interface. A simple example is shown below.

```

C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19041.508]
(c) 2020 Microsoft Corporation. 保留所有权利。

D:\Software-Test\EmbeddedBuilder\EmbeddedBuilder_v1.4.7.Test\ eclipse>headless-build.bat -workspace D:\Software-Test\Embe
ddedBuilder\EmbeddedBuilder_v1.4.7.Test\ eclipse\workspace -project ARM -build
GD32EmbeddedBuilder.exe --launcher.suppressErrors -nosplash -application org.eclipse.cdt.managedbuilder.core.headlessbu
ild -data D:\Software-Test\EmbeddedBuilder\EmbeddedBuilder_v1.4.7.Test\ eclipse\workspace -build ARM
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
11:52:02 **** Build of configuration GD ARM MCU Release for project ARM ****
make -j8 all
Building file: ../src/gd32a490_it.c
Building file: ../src/gd32a490i_eval.c
Invoking: GD ARM MCU C Compiler
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -O0 -fmessage-length=0 -fsigned-char -ffunction-sections -fdata-sections -g -
std=gnu11 -DGD32A490 -DGD_ECLIPSE_GCC -DUSE_STDPERIPH_DRIVER -I../inc -I../Firmware/CMSIS -I../Firmware/CMSIS/GD/gd
32a490/Include -I../Firmware/gd32a490_standard_peripheral/Include -MMD -MP -MF"src/gd32a490_it.d" -MT"src/gd32a490_it
.o" -Wa,-adhlns=src/gd32a490_it.o.lst -c -o "src/gd32a490_it.o" "../src/gd32a490_it.c"
...
../Firmware/gd32a490_standard_peripheral/Source/gd32a490_usart.o ../Firmware/gd32a490_standard_peripheral/Source/gd32a490_
wwdgt.o ../Firmware/CMSIS/GD/gd32a490/Source/syscalls.o ../Firmware/CMSIS/GD/gd32a490/Source/system_gd32a490.o
Finished building target: ARM.elf

Invoking: GD ARM MCU Flash Image(Bin)
arm-none-eabi-objcopy -O binary "ARM.elf" "ARM.bin"
Invoking: GD ARM MCU Listing
arm-none-eabi-objdump --source --all-headers --demangle --line-numbers --wide "ARM.elf" > "ARM.lst"
Invoking: GD ARM MCU Print Size
arm-none-eabi-size --format=berkeley "ARM.elf"
Finished building: ARM.bin
Finished building: ARM.lst
   text      data       bss         dec         hex filename
   6776      148       2108      9032      2348 ARM.elf

Finished building: ARM.siz

11:52:24 Build Finished. 0 errors, 0 warnings. (took 11s.821ms)

```

If you need to use more features, you can use the "GD32EmbeddedBuilder.exe --launcher.suppressErrors -nosplash -application org.eclipse.cdt.managedbuilder.core.headlessbuild -help" command to view more parameter information.

```

D:\Software-Test\EmbeddedBuilder\EmbeddedBuilder_v1.4.7.Test\ eclipse>GD32EmbeddedBuilder.exe --launcher.suppressErrors -nosplash -application
org.eclipse.cdt.managedbuilder.core.headlessbuild -help
Usage: D:\Software-Test\EmbeddedBuilder\EmbeddedBuilder_v1.4.7.Test\ eclipse\GD32EmbeddedBuilder.exe -data <workspace> -application org.eclips
e.cdt.managedbuilder.core.headlessbuild [ OPTIONS ]

-data      {/path/to/workspace}
-remove    {uri:/path/to/project}
-removeAll {uri:/path/to/projectTreeURI} Remove all projects under URI
-import    {uri:/path/to/project}
-importAll {uri:/path/to/projectTreeURI} Import all projects under URI
-build     {project_name_reg_ex /config_reg_ex} | all
-cleanBuild {project_name_reg_ex /config_reg_ex} | all
-markerType Marker types to fail build on (all | cdt | marker_id)
-no-indexer Disable indexer
-verbose   Verbose progress monitor updates
-printErrorMarkers Print all error markers
-I         {include_path} additional include_path to add to tools
-include   {include_file} additional include_file to pass to tools
-D         {preproc_define} additional preprocessor defines to pass to the tools
-E         {var=value} replace/add value to environment variable when running all tools
-Ea        {var=value} append value to environment variable when running all tools
-Ep        {var=value} prepend value to environment variable when running all tools
-Er        {var} remove/unset the given environment variable
-T         {toolid} {optionid=value} replace a tool option value in each configuration build
-Ta        {toolid} {optionid=value} append to a tool option value in each configuration build
-Tp        {toolid} {optionid=value} prepend to a tool option value in each configuration build
-Tr        {toolid} {optionid=value} remove a tool option value in each configuration build
Tool option values are parsed as a string, comma separated list of strings or a boolean based on the options type

```

3.5.2. Linux platform

In the same directory as the executable file, run the following command in the terminal interface:
 "./headless-build.sh -workspace /root/eclipse/patch/headlessbuild1/eclipse/workspace -project ARM -build"

```

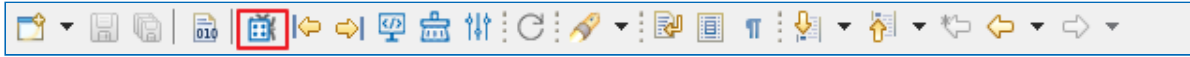
root@gdtest-virtual-machine: ~/eclipse/patch/headlessbuild...
root@gdtest-virtual-machine:~/eclipse/patch/headlessbuild1/eclipse# ./headless-build.sh -workspace /root/eclipse/patch/headlessbuild1/eclipse/workspace -project ARM -build

```

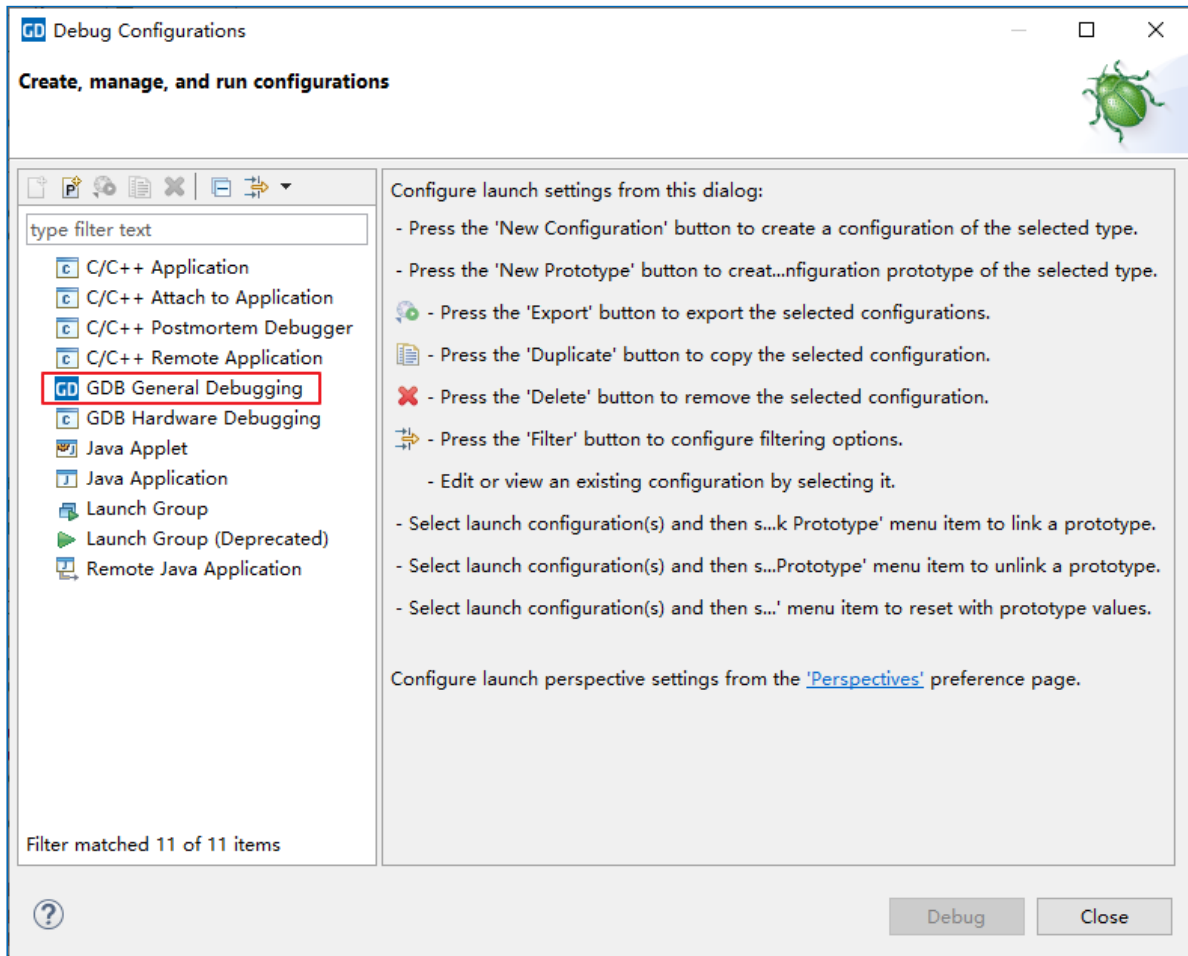
4. Debugging and Running Project

4.1. Creating a Debug/Run Configuration

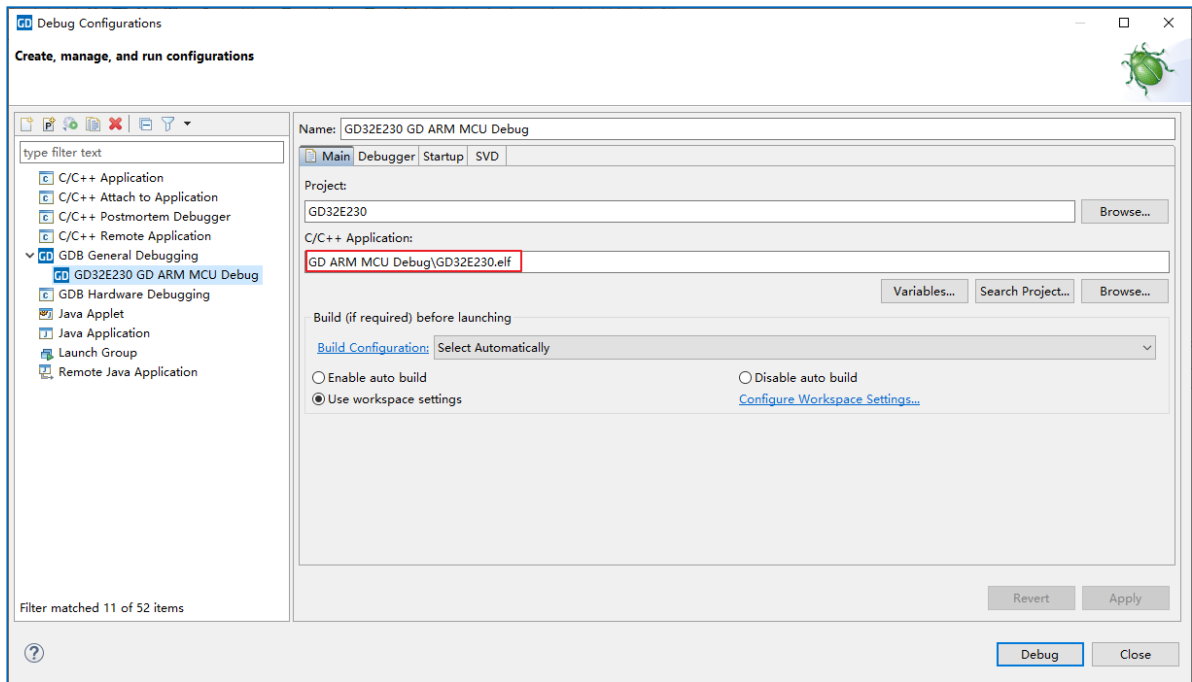
Select your target project, then click the debug configuration icon on the toolbar. Then, the **Debug Configurations** dialog window will be opened.



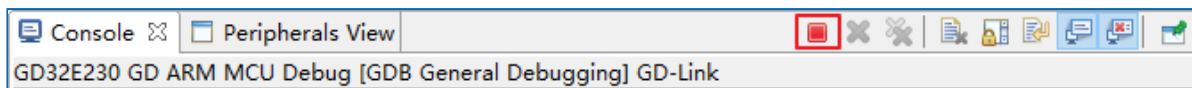
Double-click **GDB General Debugging** to create a new debug launch configuration. If the configuration is created successfully, the debugging project name will be in the **Name** field.



Please make sure that the project has been built successfully.

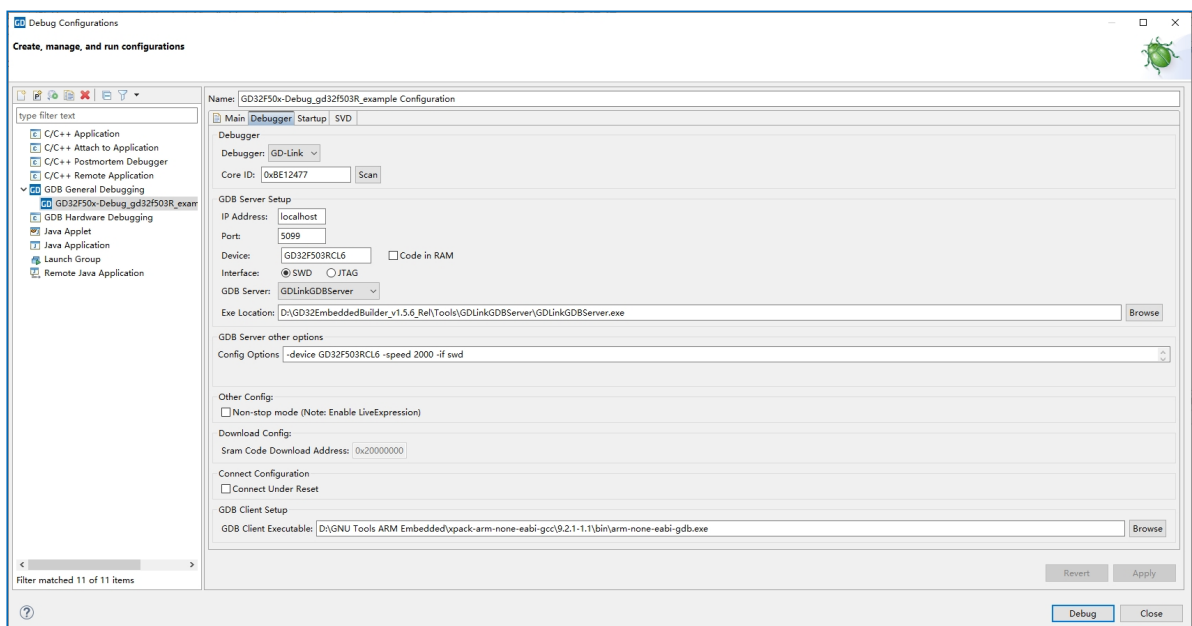


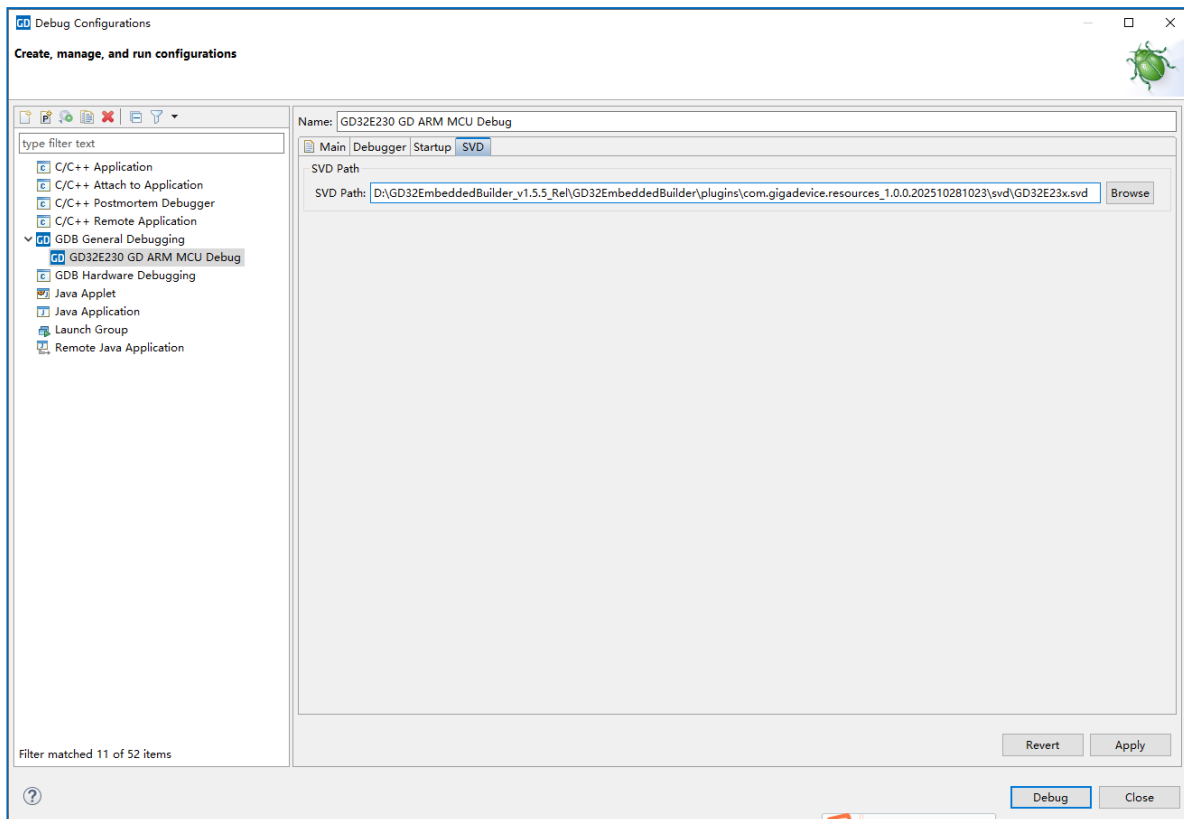
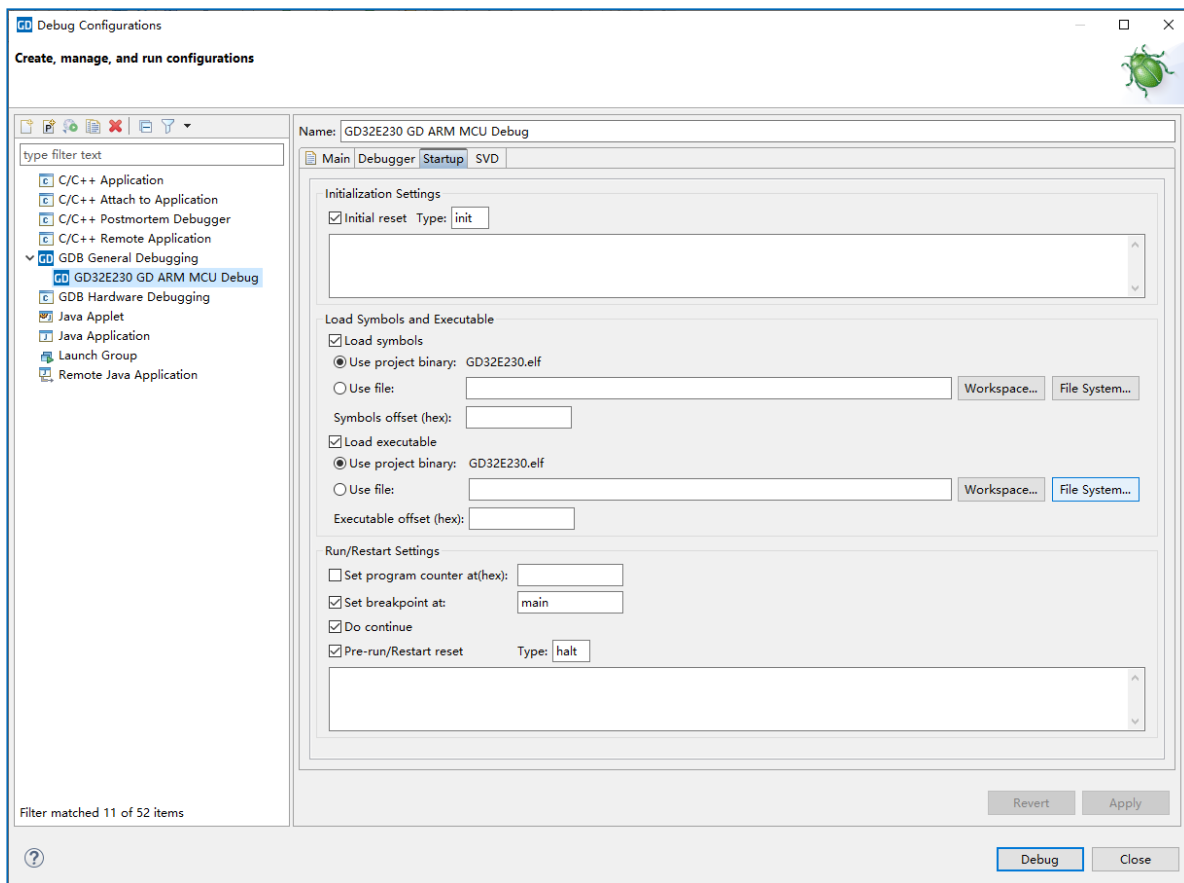
Notice it should be **terminated** by clicking the red block when you stop debugging. If you don't click it, it will have influence on the next debugging.



4.2. Editing the Debug/Run Configuration

The **GDB General Debugging** wizard contains four tabs as shown in the figures below. You will use these tabs to customize your run and debug configurations. Click the **Debugger** tab to select a debugger type from GD-Link. If GD-link is selected as the debugger, you should select GDB Server between GdLinkGDBServer and Openocd. In addition, click the **Scan** button to read the core ID of the target device by your selected debugger. When using a project that is not created by the IDE, the GDB Client Executable will be enabled, allowing you to select the local gdb path. Click the **Startup** tab to set debugging options, including reset options, executable file and symbols loading, breakpoint options, etc.





4.3. Debugging/Running

On the toolbar, choose **Debug** from the left drop-down menu. Choose **Debug** from launch mode menu to start debugging.



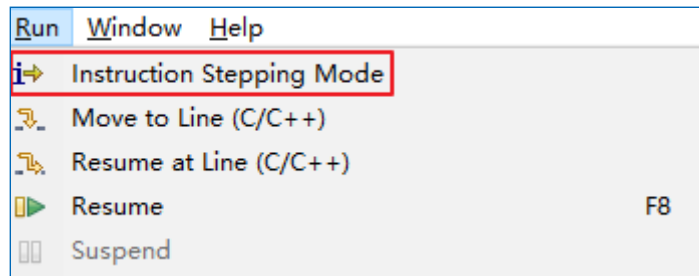
4.4. Resetting during Debugging

You can reset device during debugging by clicking the **reset** button on the toolbar.



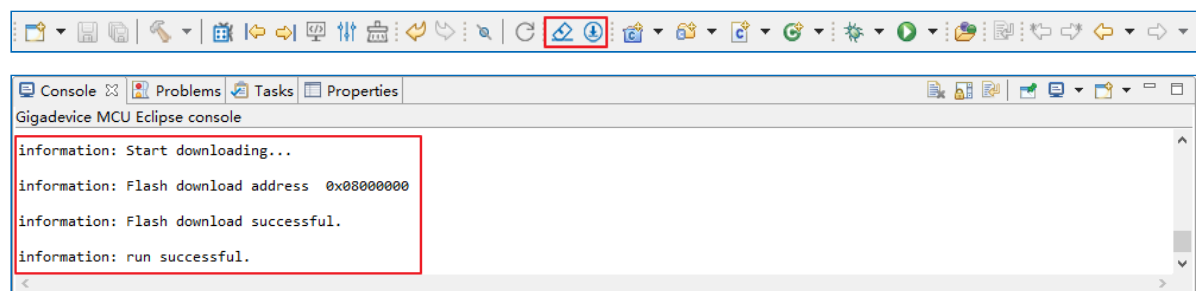
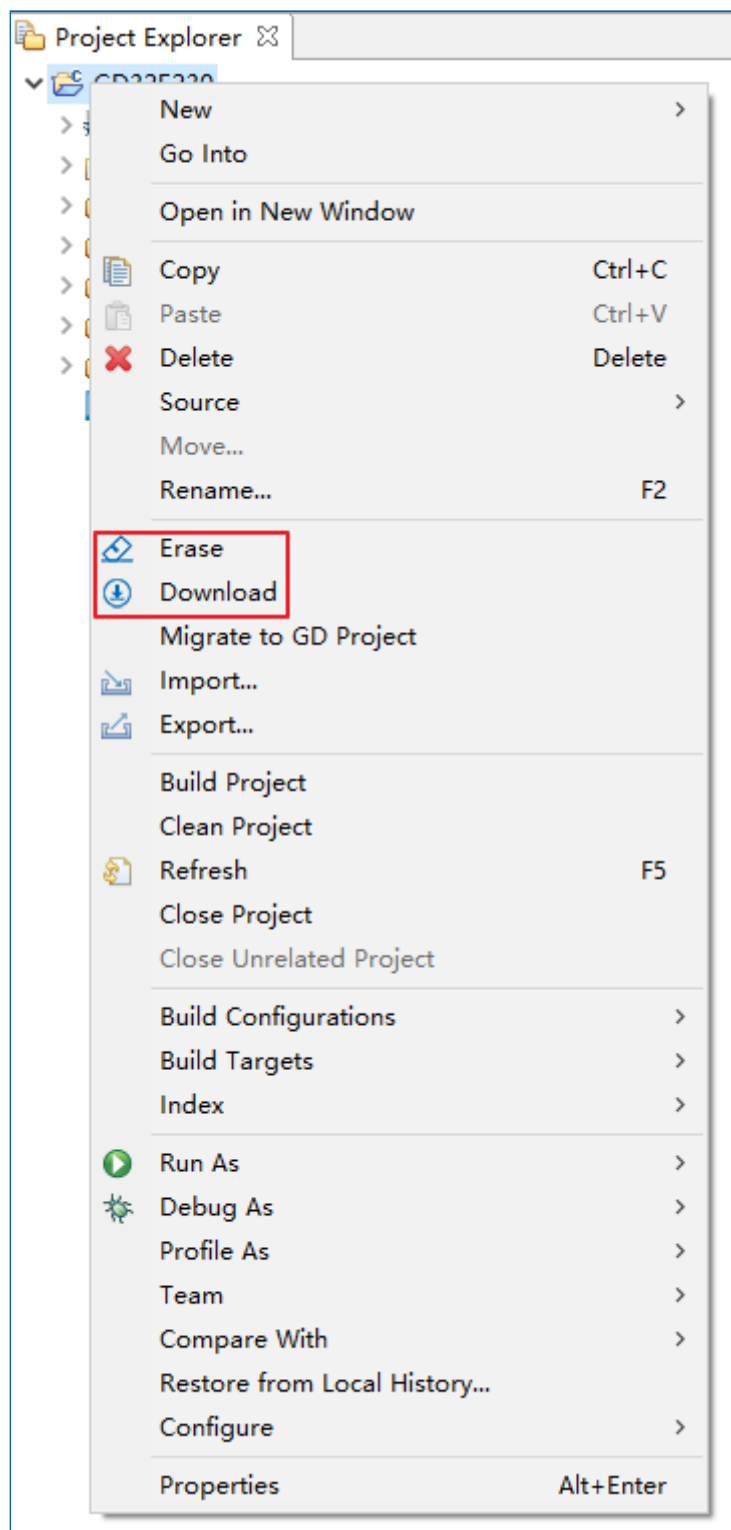
4.5. Single-step Debugging of Assembly Code

You can open or close the function by **Run-Instruction Stepping Mode** on the menu bar.



4.6. Download & Erase

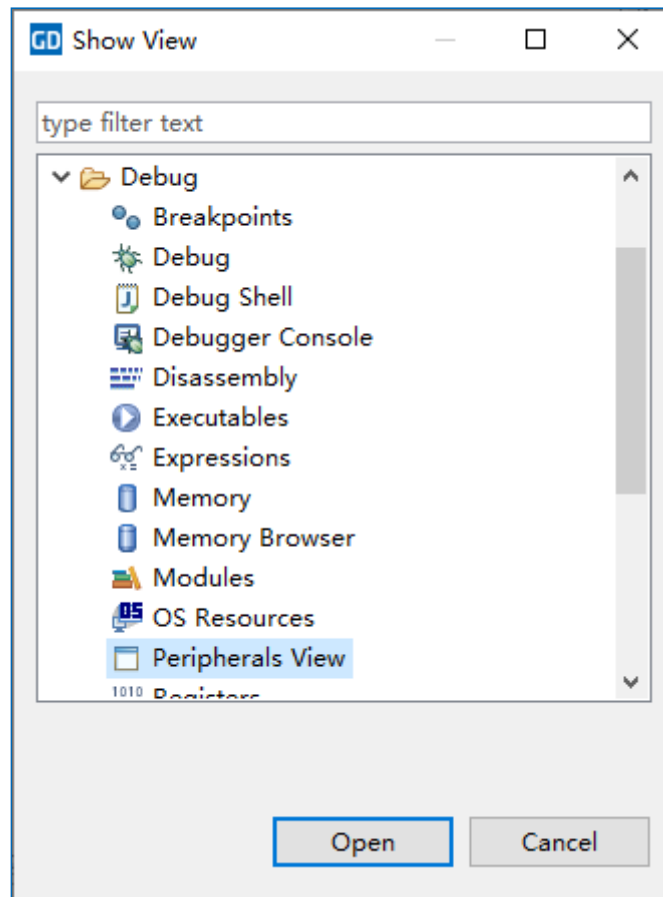
Click the **Erase** menu command to erase chip, and click the **Download** menu command to download. You can see the output and results of the run command in the **Console** view. The **Download** and **Erase** button functions in the two images below are consistent.



4.7. Peripherals View

To monitor registers during debugging, you need to open the **Peripherals View** first. When the MCU is halted, you can watch the peripheral register value.

Select **Window > Show View > Other** menu command to start a new wizard, then expand the **Debug** folder and select **Peripherals View**. Click **Open**.



You can modify registers in the **Peripherals View**. Type a new value in the **HEX Value** column of target register bits row, then press the **Enter** key to start the modification. If the modification is successful, both the register bits and the register value will be refreshed. Otherwise, the modification is invalid, which may be due to the peripheral clock is not enabled.

Name	Address	HEX Value	BINARY Value
PMU			
RCU			
> CTLO	0x 40021000	0x 03037CB3	
> CFG0	0x 40021004	0x 001D000A	
> INT	0x 40021008	0x 00000000	
> APB2RST	0x 4002100C	0x 00000000	
> APB1RST	0x 40021010	0x 00000000	
> AHBEN	0x 40021014	0x 00000014	
PFEN		0x 0	0b 0
PCEN		0x 0	0b 0
PBEN		0x 0	0b 0
PAEN		0x 0	0b 0
CRCEN		0x 0	0b 0
FMCSPEN		0x 1	0b 1
SRAMSPEN		0x 1	0b 1
DMAEN		0x 0	0b 0
> APB2EN	0x 40021018	0x 00000000	
> APB1EN	0x 4002101C	0x 00000000	

Name	Address	HEX Value	BINARY Value
PMU			
RCU			
> CTLO	0x 40021000	0x 03037CB3	
> CFG0	0x 40021004	0x 001D000A	
> INT	0x 40021008	0x 00000000	
> APB2RST	0x 4002100C	0x 00000000	
> APB1RST	0x 40021010	0x 00000000	
> AHBEN	0x 40021014	0x 00400014	
PFEN		0x 1	0b 1
PCEN		0x 0	0b 0
PBEN		0x 0	0b 0
PAEN		0x 0	0b 0
CRCEN		0x 0	0b 0
FMCSPEN		0x 1	0b 1
SRAMSPEN		0x 1	0b 1
DMAEN		0x 0	0b 0
> APB2EN	0x 40021018	0x 00000000	
> APB1EN	0x 4002101C	0x 00000000	

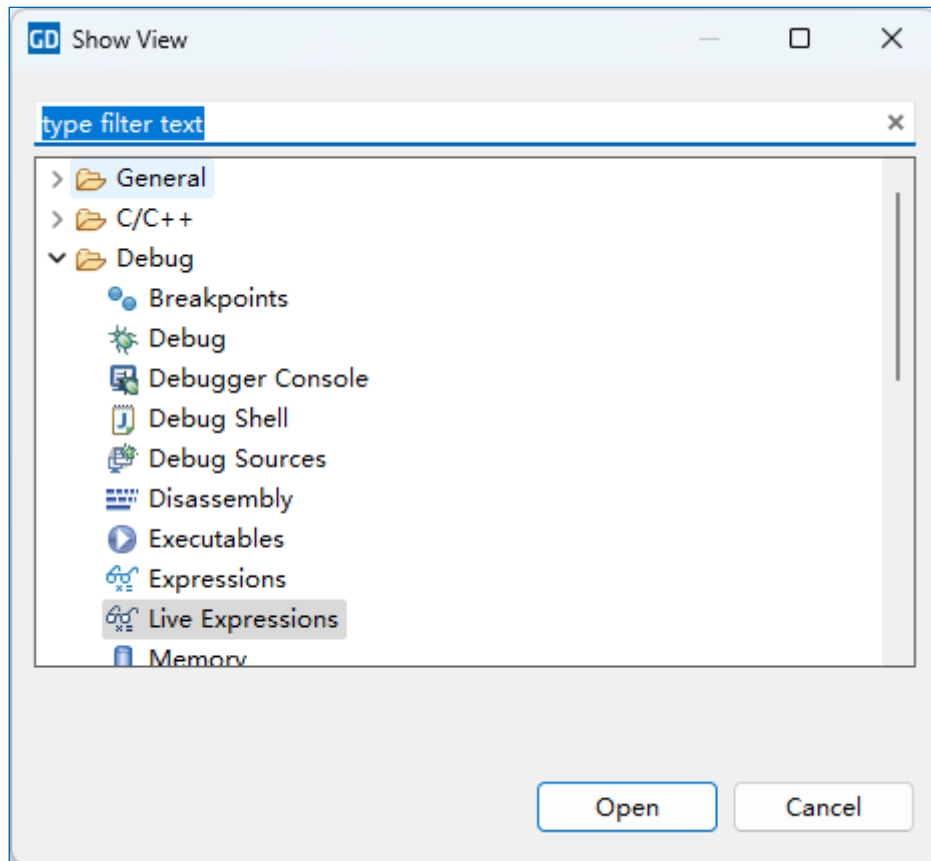
4.8. Live Expressions View

This view supports **real-time monitoring of expressions during debugging**. You can add expressions that need to be monitored in the view and set the monitoring time through the view menu bar.

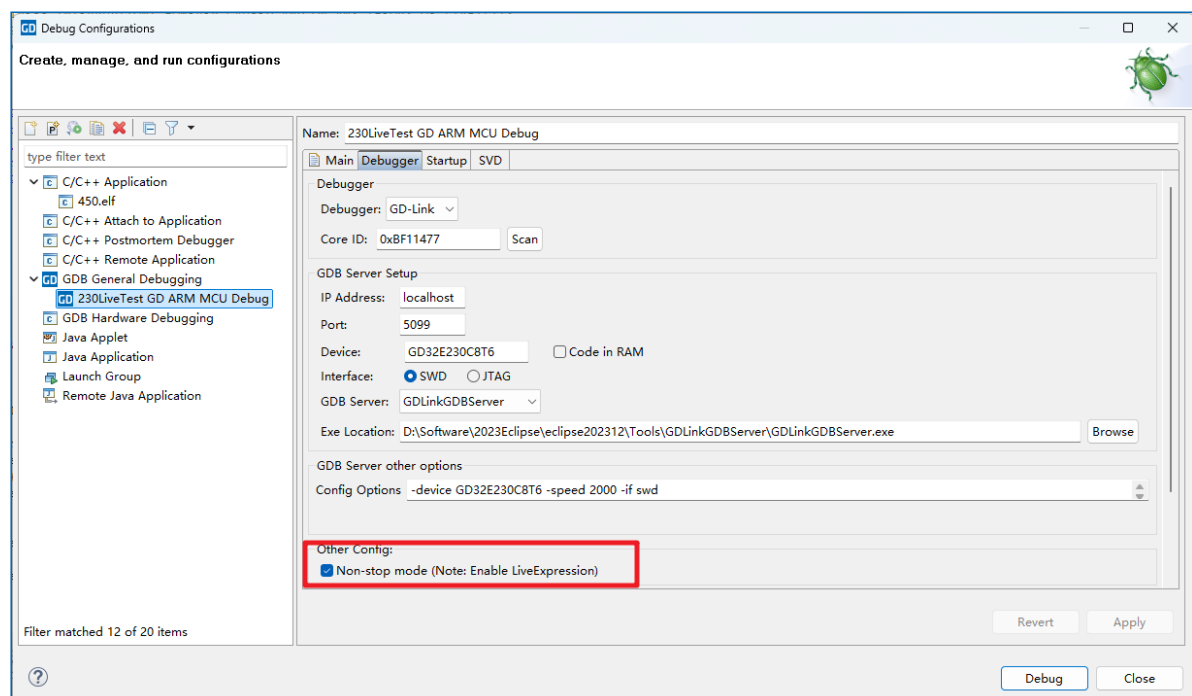
Note:

Only supports Live Expressions viewing during **GD-Link** debugging sessions

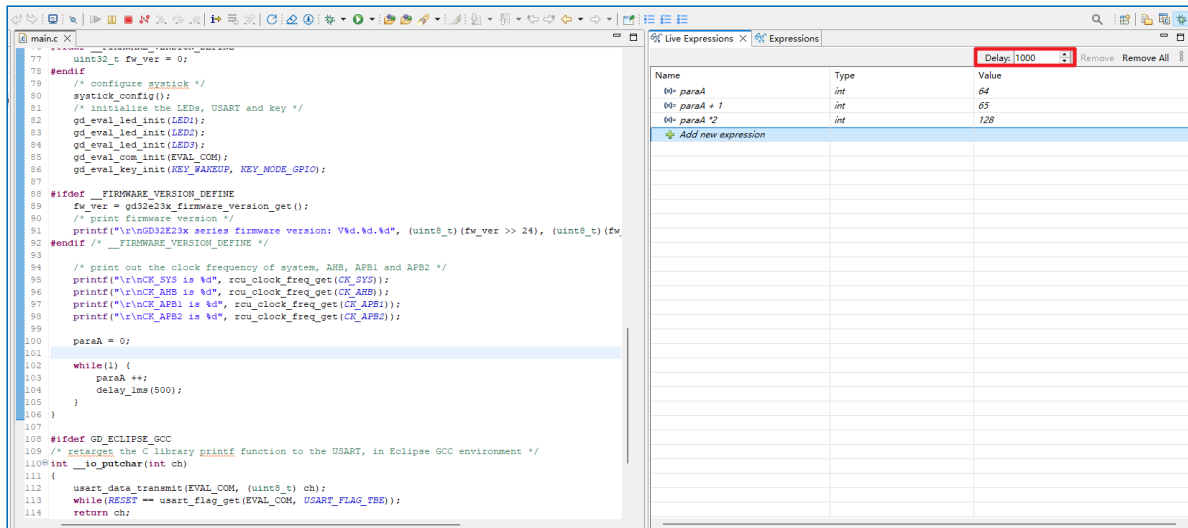
Select **Window > Show View > Other** menu command to start a new wizard, then expand the **Debug** folder and select **Live Expressions**. Click **Open**.



Before using Live Expressions Views, you need to **enable Non-stop mode** in the debug configuration.



Variables or expressions can be added to the view for monitoring, and the sampling time can be changed through the Delay configuration.



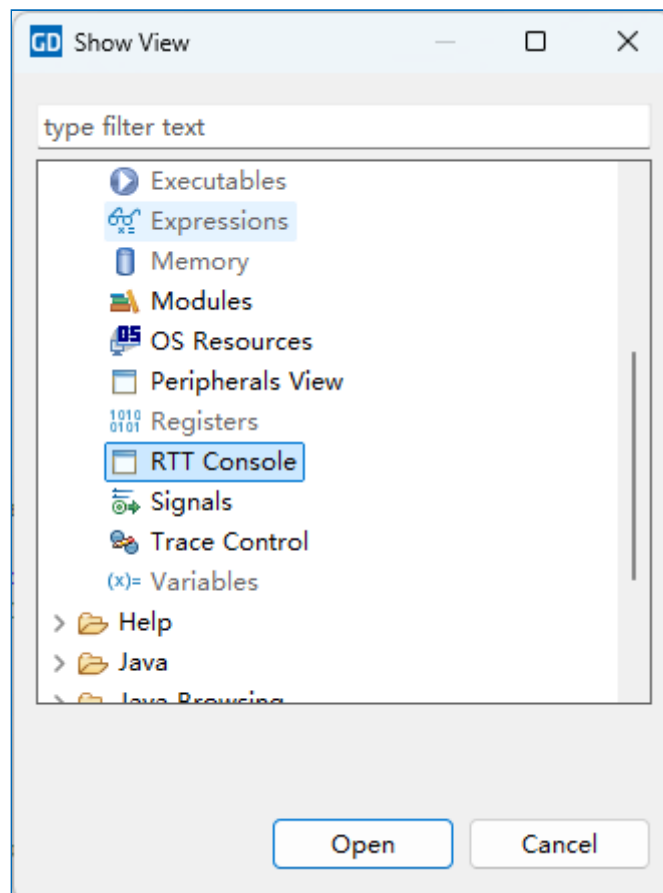
4.9. RTT Console View

This view supports **real-time monitoring of RTT data during JLink debugging**.

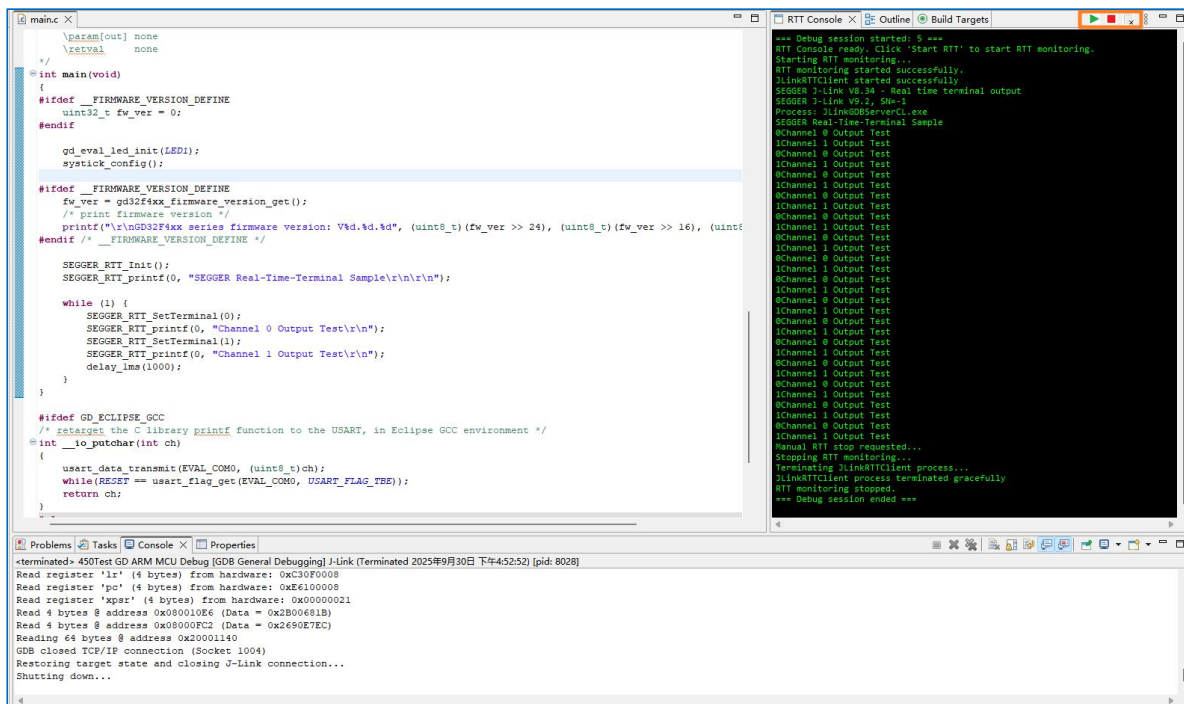
Note:

Only supports real-time RTT data viewing during **JLink** debugging sessions

Select **Window > Show View > Other** menu command to start a new wizard, then expand the **Debug** folder and select **RTT Console**. Click **Open**.



The upper right corner of the view provides '**Start**', '**Stop**', and '**Clear Console**' functions for controlling RTT monitoring and information printing.

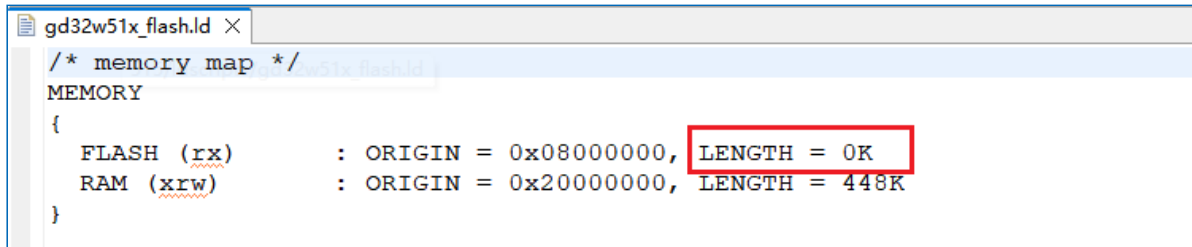


5. FAQs

5.1 Building

Q1: W515 compilation error.

A1. Check if it is a 0 Flash product, manually change the ld size.



Q2: Compilation prompts that .h file cannot be found, but the file exists in the project and can be navigated to.

A2. This is caused by Windows system path length and compiler limitations. The file path may be too long, exceeding the compiler's range. This can be avoided by changing the software folder length or project path length.

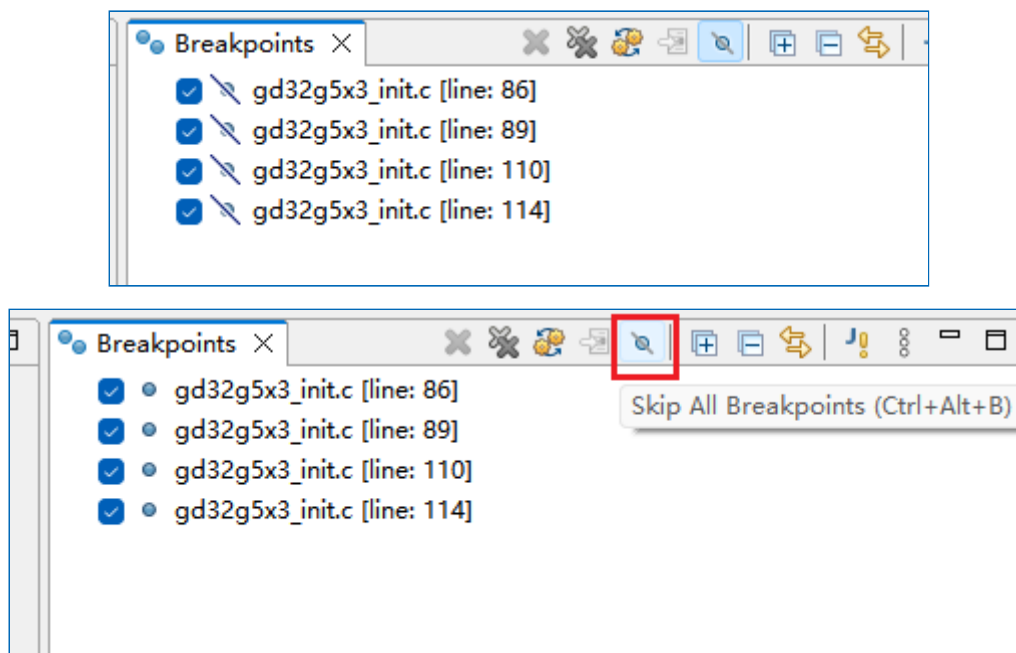
5.2 Debugging

Q1: After connecting GD-Link V2 to the computer, the IDE does not detect GD-Link on Windows 10/Windows 11.

A1. On Windows 10/Windows 11, first uninstall the GD-Link V2 device driver, then reconnect and power cycle the GD-Link V2 device.

Q2: Breakpoints do not stop during debugging

A2. Check if the "Skip All Breakpoints" feature is enabled.



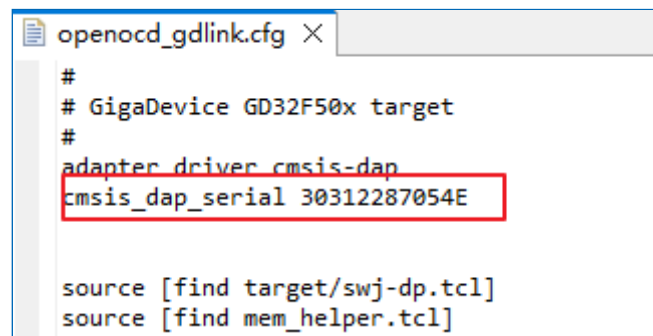
Q3: If the Debugger page prompts "Fail to read core ID"

A3.Try using GD-Link Programmer to check if the GD-Link device serial number is correctly recognized. If not, and your PC is running Windows 10 or 11, uninstall the driver in Device Manager, then reconnect the device. Open GD-Link Programmer again to see if the serial number is recognized. If it is, the issue is resolved. This method usually solves the problem.

Q4: When multiple CMSIS-DAP devices are connected to the computer and using OpenOCD and GD-Link for debugging

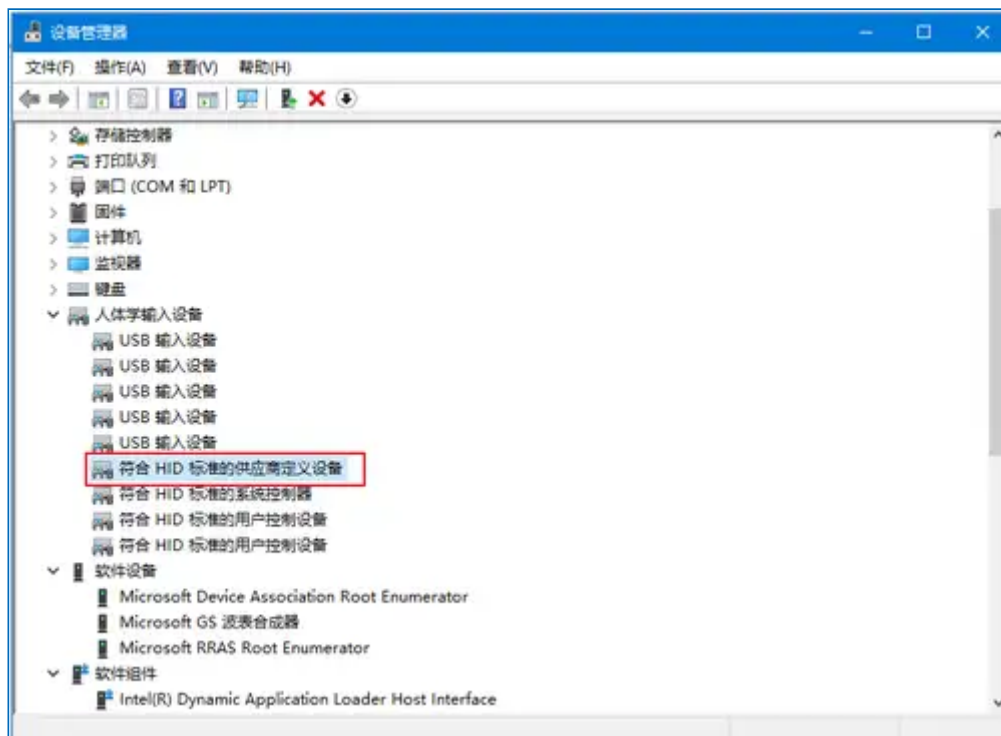
A4.Add the GD-Link device serial number in the cfg file.

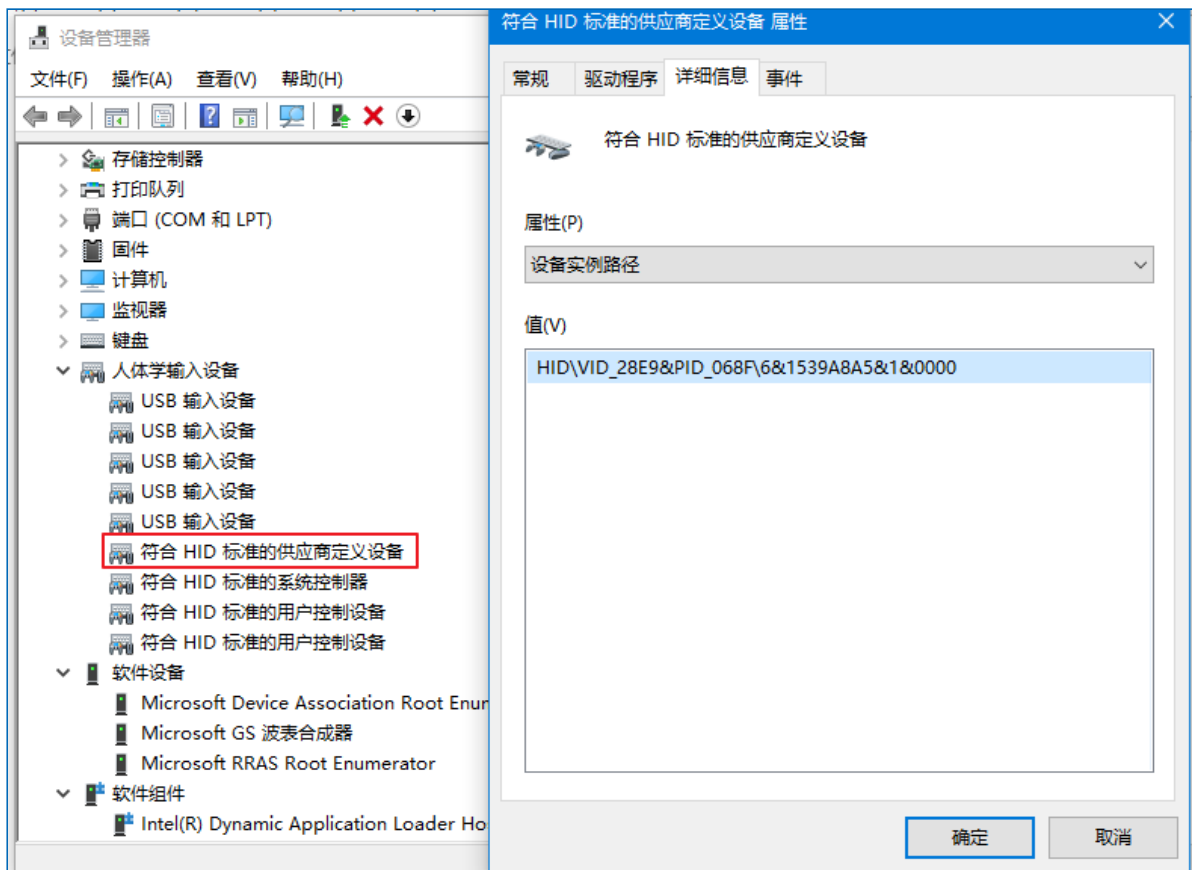
```
GDLink_CLI V1.0.9.33666.
Connected device number: 1
#0 SN 30312287054E
ERROR: Fail to connect GD-Link.
Change USB Device failed, please check SN 30312287054E.
请按任意键继续. . .
```



Q5: GD-Link V1 does not require driver installation

A5.GD-Link V1 is driver-free. If V1 is not recognized by the PC, check if a driver has been installed. In Device Manager, V1 should appear as shown below:

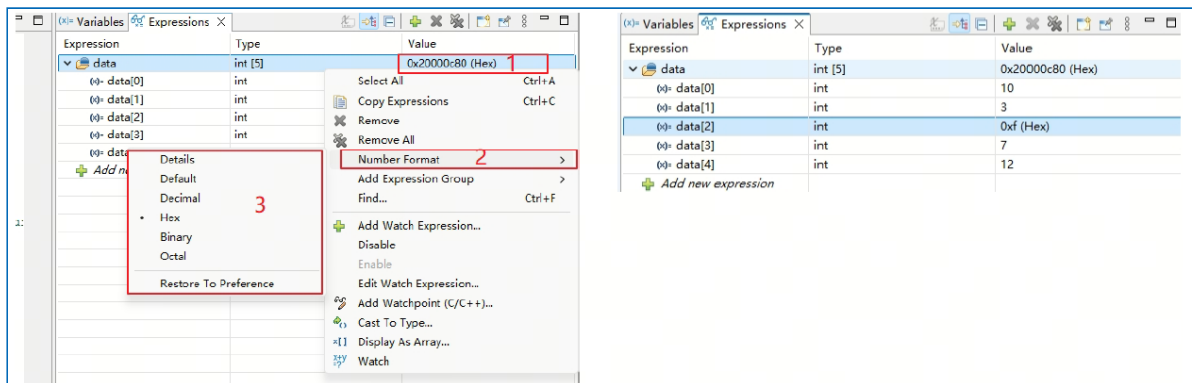




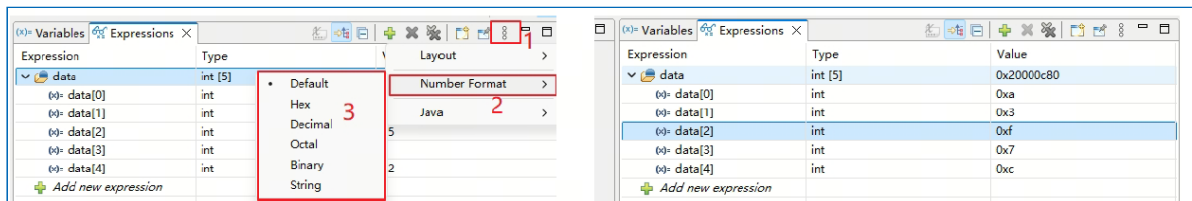
Q6: Expressions View Data Number Format Selection

A6.

1.To select the number format for a single Expression, right-click the Expression > Number Format > choose the desired format.



2.To select the number format for a all Expression, click the View Menu in the upper left corner > Number Format > select the radix to display.

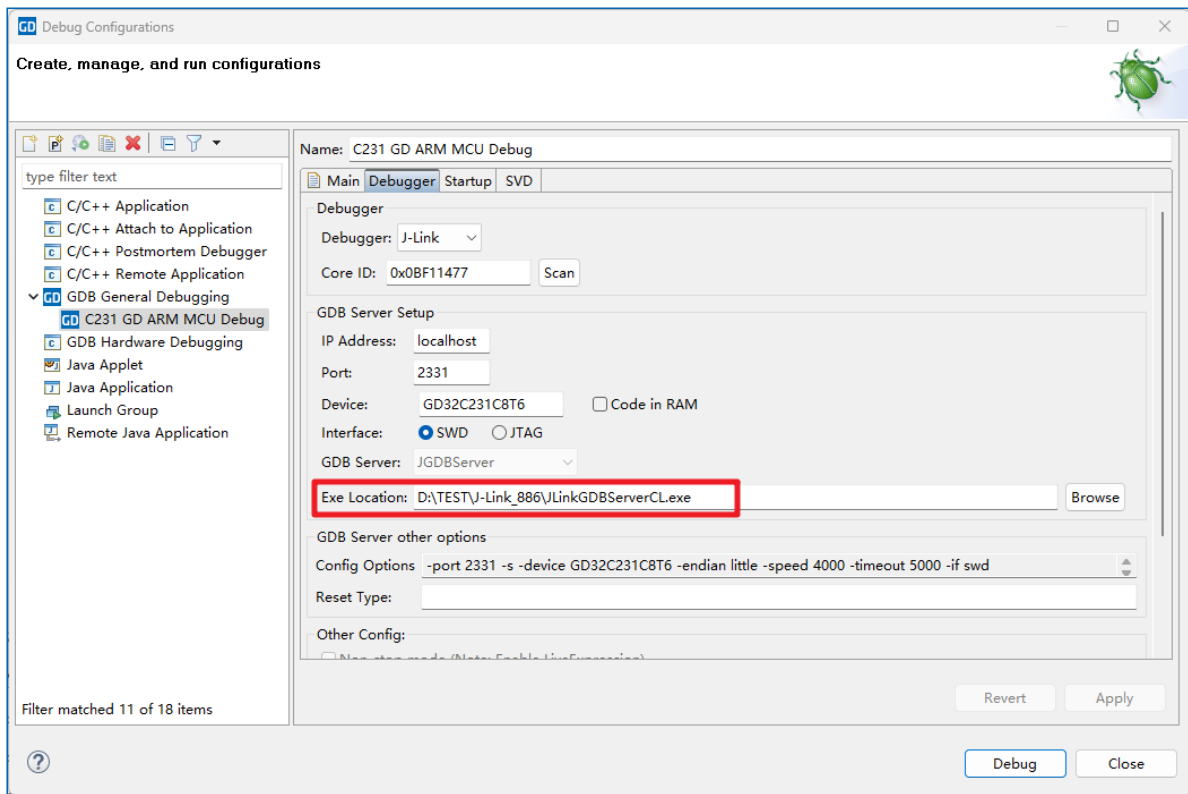


Q7: Need to use the new version of J-Link

A7.

Method 1: Replace the new version in .\Tools\J-Link directory - this configuration applies to all projects.

Method 2: In Project > Debug Configurations > Debugger, update to the local new version JLinkServerCL.exe path - this only applies to the current project.



5.3 Platform

Q1: Linux gdb debugging error.

A1. If the gdb debugging function cannot be used normally and reports the error `"/arm-none-eabi-gdb: error while loading shared libraries: libncurses.so.5: cannot open shared object file: No such file or directory"`, please execute the following two commands to install libncurses5 online.

- `sudo apt-get update`
- `sudo apt-get install libncurses5`

6. Revision history

Version	Changes
v1.5.5_Rel	<ol style="list-style-type: none"> 1. Added the Download and Erase buttons synchronously to the toolbar. 2. Added the "Import file" option to the project's right-click menu for importing folders or files into the project. 3. Updated the firmware library to the mid-2025 version. 4. Added the Startup Tab to the debug configuration for setting up debugging options and commands. 5. Added GD-Link Live Expressions View. 6. Added J-Link RTT Console View. 7. Added support for J-Link JTAG protocol. 8. Template project added the F50x series support. 9. Optimized the GD project interface. 10. Repaired some bugs.
v1.5.6_Rel	<ol style="list-style-type: none"> 1. Added a GDB executable file path configuration feature to the debugger settings page to support debugging projects not created by this software. 2. Modified the GD project architecture design for the G5 and C2x1 series. 3. Updated the G5 series examples. 4. The firmware library version of the F50x template project has been updated to V1.0.2. 5. J-Link version updated to V8.86. 6. Optimized the GD project interface. 7. Repaired some bugs.
v1.5.7_Rel	<ol style="list-style-type: none"> 1. Added Chinese and English language support. 2. Added -Ofast and -Og options to compiler optimization settings. 3. Adjusted G5 firmware library and chip selection in template projects, updated E231 and C103 series chip selections. 4. Automatically display disassembly view when debugging. 5. Removed support for the A503 and A513 series. 6. Optimized GD project-related functions. 7. Updated J-Link to version V9.10. 8. Repaired some bugs.