Application Note

Document No.: AN1166

G32R430 DDL SDK Quick Start Guide——

Based on G32R430TinyBoard

Version: V1.3

1 Introduction

This application note aims to help you quickly understand and apply the G32R430 DDL SDK, as well as the development process based on the G32R430TinyBoard. By reading this note, you will become familiar with common hardware resources and the SDK directory structure, and learn how to run sample programs in the MDK, IAR and Eclipse environments. This will help you quickly complete preliminary evaluation and application development for the G32R430 series chips.

Before starting, please prepare the following environments and tools:

- Windows 10 or above operating system
- G32R430 DDL SDK Vx.x.x
- Keil MDK 5.40 or above version
- IAR for Arm 9.60.2 or above version
- Eclipse 4.35 or above version
- xpack-windows-build-tools 4.4.1
- LLVM-ET-Arm-19.1.1-Windows-x86_64
- arm-gnu-toolchain-14.2.rel1-mingw-w64-x86_64-arm-none-eabi
- PyOCD 0.44 or above
- G32R430TinyBoard V1.3
- USB Type-C data cable

Contents

2 Basic Information of G32R430TinyBoard

This section provides an overview of the G32R430TinyBoard, so that users can quickly understand and correctly use the G32R430TinyBoard's on-board functions and resources, thereby preparing for subsequent development in the MDK, IAR and Eclipse environments.

2.1 Introduction to Development Board Resources

The G32R430TinyBoard is a development board based on the G32R430 series MCU. It features rich on-board resources, and meets the needs of users from beginners to those with certain development experience for rapid evaluation and development. The following are the commonly used resource configurations and interface description for this board:

Figure 1 G32R430TinyBoard

- 2×LED: LED1 (PA1), LED2 (PA2)
- 2×keys: User Key1 (PA5), Reset key
- 1×USART: USART2_TX (PD0) / USART2_RX (PC12)
- 1×I²C EEPROM: SCL1 (PD5) / SDA1 (PD9), ZD24C64A-XGMT chip
- 1×RS-485 interface: USART1_TX (PB9) / USART1_RX (PB6) / USART1_DE (PB11)
- 1×RS-422 interface: CLK (PC9) / MISO (PD2)

- 1×on-board GEEHY-LINK emulator:
- Implements serial communication with the G32R430 chip via the on-board emulator's TX and RX.
- Supports download and debugging.

2.2 Precautions

If ports such as A1, A2, and A5 on the J12 interface need to be used, the jumper cap on J10 needs to be transferred from the default LED1, LED2, KEY positions to A1, A2, A5 respectively. This ensures that the actual available pin resources correspond to the target functions.

Figure 2 From Default LED1, LED2, KEY Resources to A1, A2, A5

If an external emulator needs to be connected, first disconnect the electrical connection between the on-board GEEHY-LINK emulator and the board (e.g., by physically separating the emulator from the board).

3 Introduction to SDK Directory Structure

The G32R430 DDL SDK provides comprehensive development support from drivers and core libraries to example projects, making it easy for users to get started quickly and make secondary development. Its general directory structure is as follows (expanded to the second level; Package contains only the DFP install package; FLM / SVD / pyOCD / IAR AddOn are under Utilities).

```
G32R430_DDL_SDK/
├── Boards/
│   └── Board_G32R430_TINY/
├── Documents/
│   ├── DDL_Driver_API/
│   ├── AN1166_*.pdf
│   ├── AN1185_*.pdf
│   └── DATASHEET.pdf
├── Examples/
│   └── Board_G32R430_Tiny/
├── Libraries/
│   ├── ATAN2/
│   ├── CMSIS/
│   ├── Device/
│   └── G32R4xx_DDL_Driver/
├── Middlewares/
│   └── Coremark/
├── Package/
│   └── Geehy.G32R4xx_DFP.x.y.z.pack
└── Utilities/
    ├── FLM/
    ├── IAR_AddOn/
    ├── pyOCD/
    └── SVD/
```

Note: For more details about the SDK, please review the Readme.pdf file located in the SDK root directory.

4 How to Run Examples

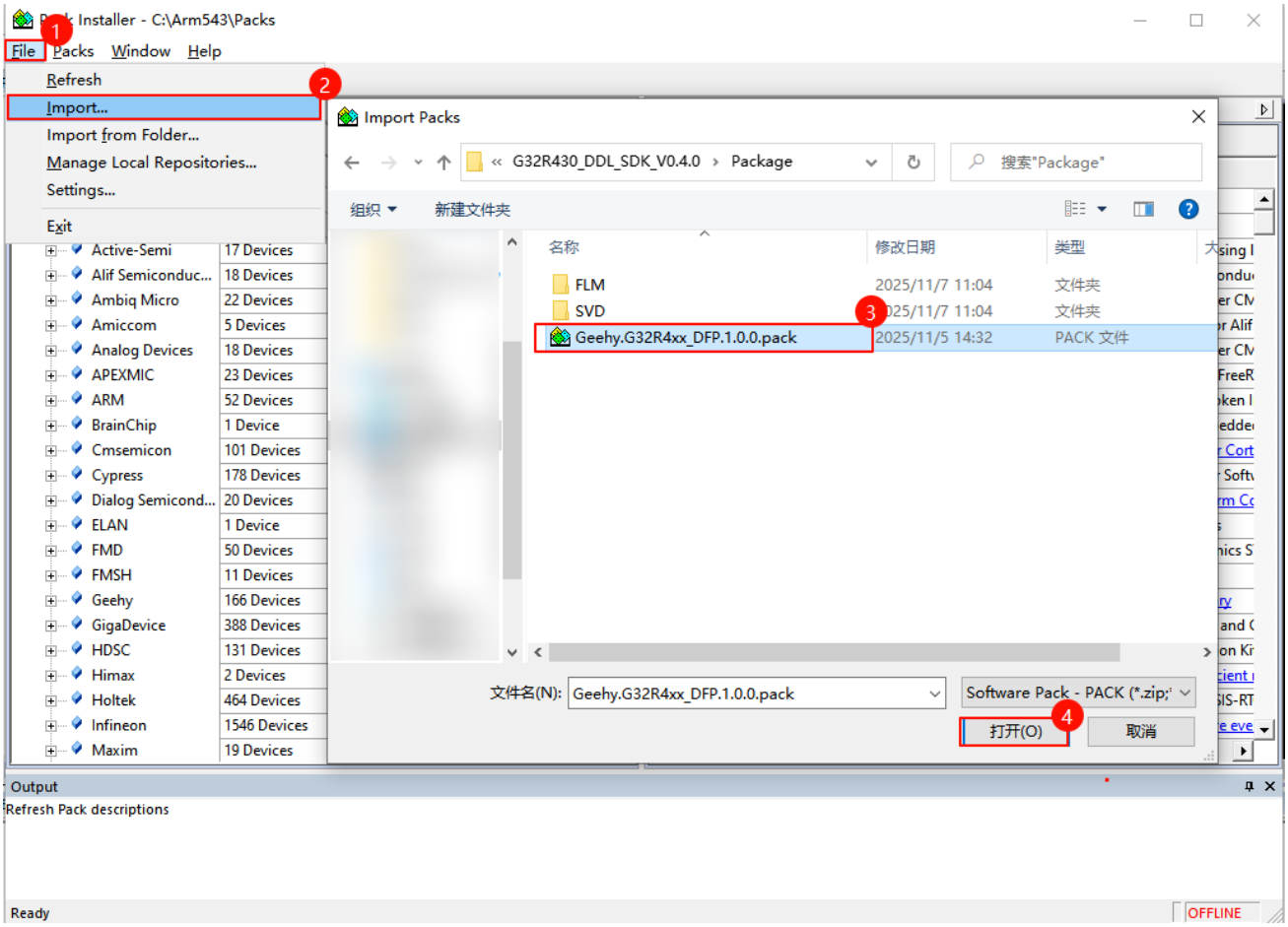
To help users get started quickly and verify the board and SDK functions, the SDK provides example projects in both the MDK, IAR and Eclipse environments. The following will respectively introduce the preparatory work that needs to be done in these development environments, as well as how to compile, download and debug the example projects.

4.1 Running Examples in Keil MDK

4.1.1 Install the Chip Support

Use the "PackInstaller.exe" on the MDK configuration interface to install by importing the Pack, which avoids lags or exceptions that may occur during double-click installation.

Figure 3 Using PackInstaller to Install Geehy.G32R4xxDFP.x.x.x.pack



How to open PackInstaller.exe:

1. Click the "Pack Installer" icon on the MDK status bar.
2. Or run PackInstaller.exe from the MDK installation directory, for example: “C:\Users\Geehy\AppData\Local\Keil_v543a\UV4\PackInstaller.exe”.

Note:

（1） The MDK version must be ≥ v5.40. MDK v5.43a is recommended for compatibility and stability.

（2） On MDK v5.43a, double-clicking to install “Package\Geehy.G32R4xx_DFP.x.x.x.pack” from the SDK root may hang; this is an MDK anomaly.

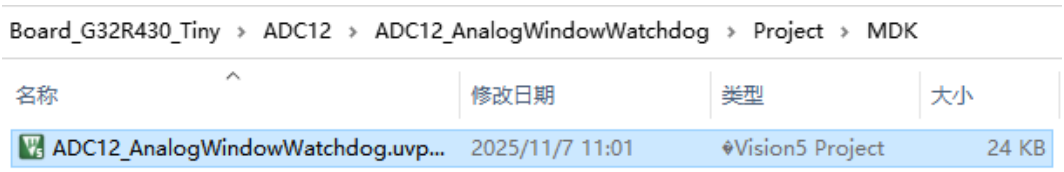
4.1.2 Use an Example Program

Open the MDK project directory for the corresponding board. Taking the ADC12 module as an example, the path is as follows:

```
Board_G32R430_Tiny\ADC12\ADC12_AnalogWindowWatchdog\Project\MDK
```

In this directory, find the “.uvprojx” file and double-click it to load the example project.

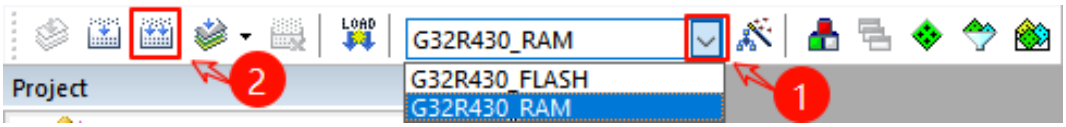
Figure 4 ADC12AnalogWindowWatchdog MDK Project Diagram



4.1.3 Compile the Program

1. After opening the project, check whether the project Target configuration is the desired one.
2. Click the Compile button and wait for completion. If there are no errors, MDK displays "0 error, 0 warning" in the output box.

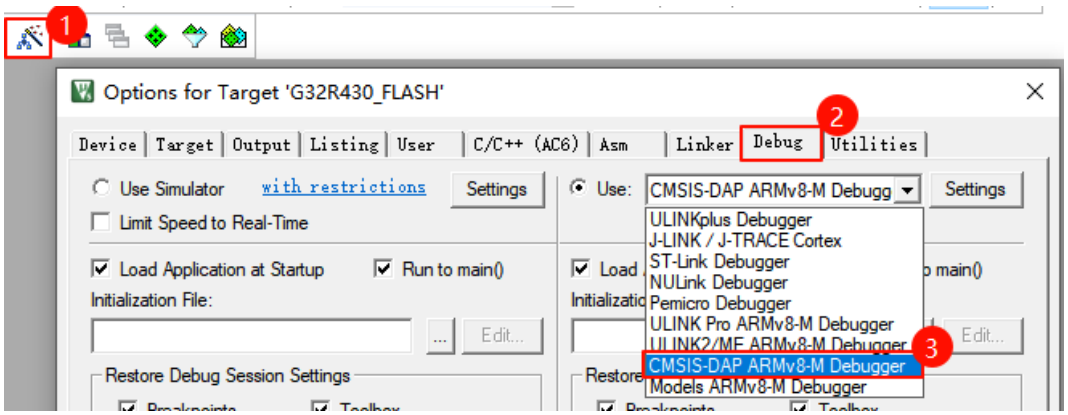
Figure 5 Multi-Target Selection and Compilation in MDK



4.1.4 Download the Program

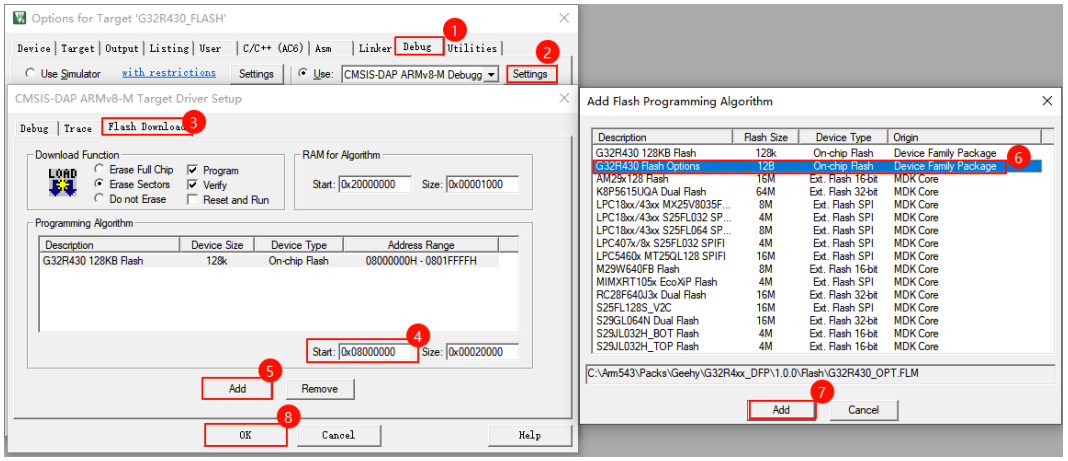
Select the emulator type as "CMSIS-DAP ARMv8-M Debugger".

Figure 6 Selecting an Emulator in MDK



Configure the download content in MDK's "Flash Download" as needed.

Figure 7 Configuring Flash Download Content



Configuration notes:

- OPT code download settings (if writing to the emulated OTP/configuration area is required): additionally perform steps 5, 6, and 7 in Figure 7 to add the OPT download algorithm.
- RAM program download settings (if the program must run in RAM): in step 4 of Figure 7, change the programming area to the download area and skip steps 5, 6, and 7.

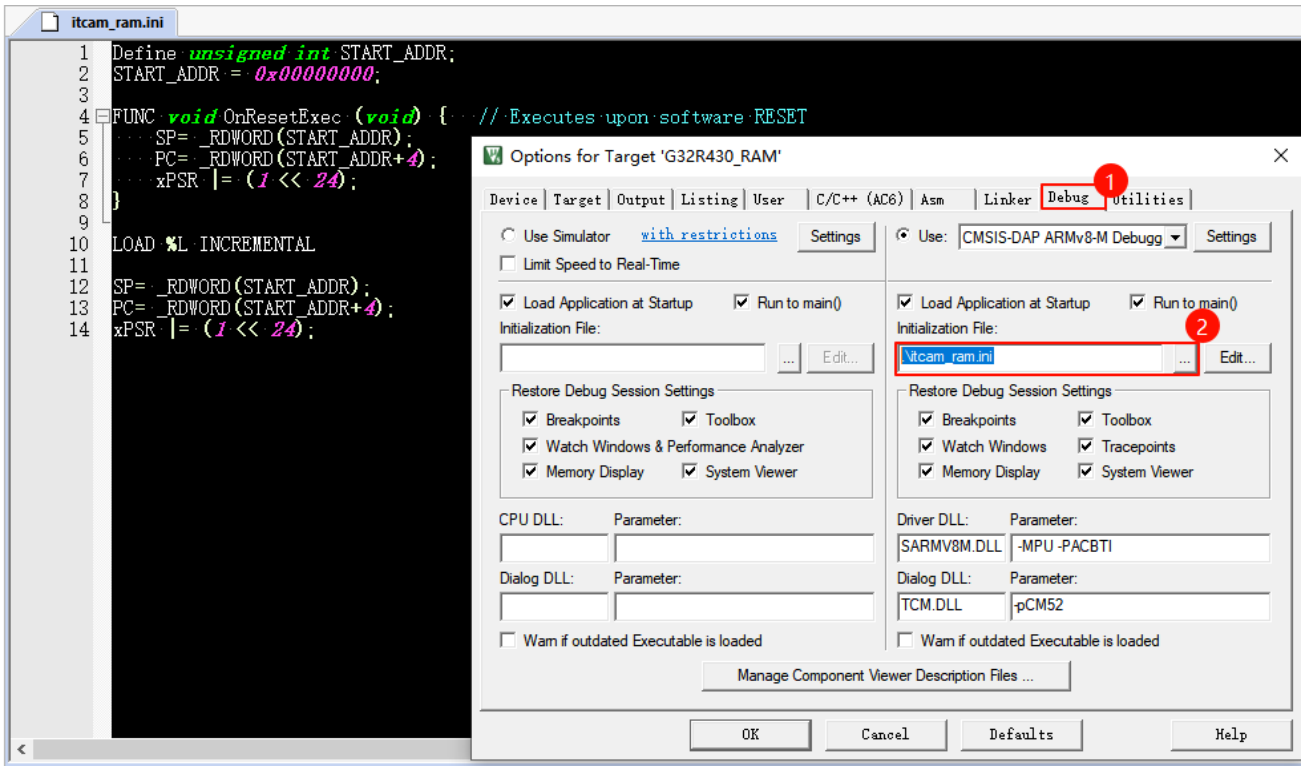
Note: The download area must not overlap with the RAM area used to execute the download algorithm; otherwise the download algorithm will fail to run properly.

4.1.5 Debug the Program

In the debugging configuration, set the .ini script file (if the program run address differs from the chip boot address) so that the PC is initialized to the correct start address. For the .ini script, see

G32R430_DDL_SDK_Vx.x.x\Examples\Board_G32R430_Tiny\ATAN2\ATAN2_Math\Project\MDK\itcam_ram.ini.

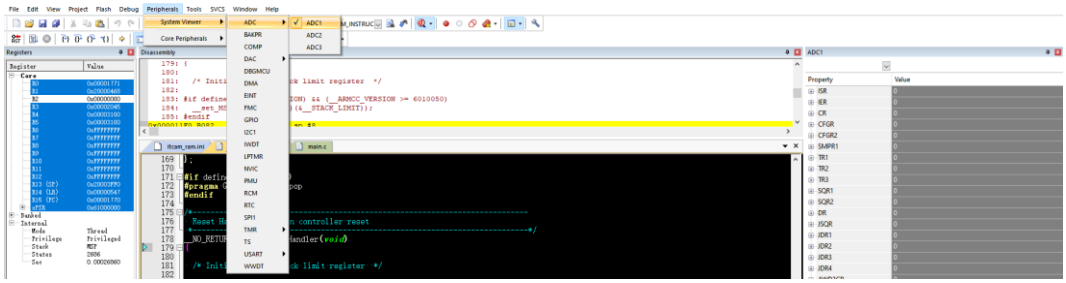
Figure 8 Configuring .ini File in MDK



Click "Debug" to enter the debugging interface, where you can view:

- General-purpose registers (content of general-purpose CPU registers).
- Core and peripheral registers (register configuration and status of each peripheral module).

Figure 9 Viewing Peripheral Registers in MDK Debug Interface



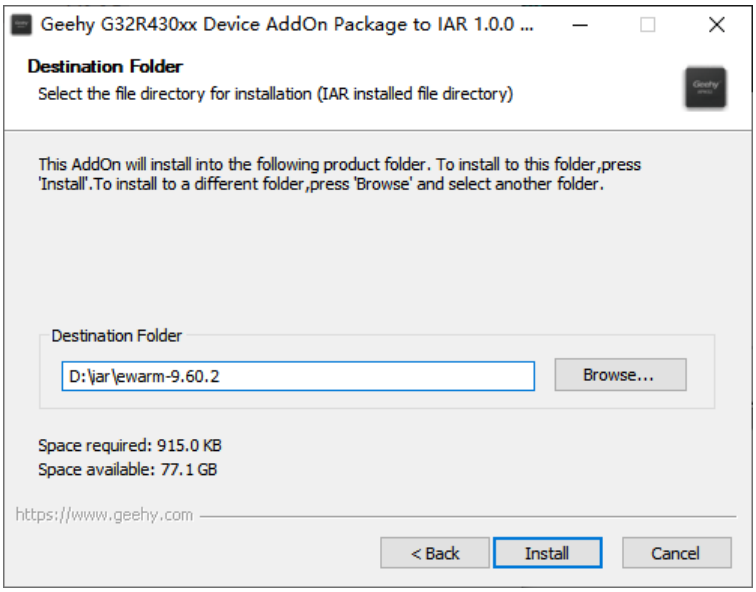
4.2 Running Examples in IAR for Arm

4.2.1 Install the Chip Support

Run the Utilities\IAR_AddOn\G32R430xx_AddOn_vx.x.x.exe installation program. The

installation path should match the path of the locally installed IAR. For example, if the IAR startup program on the demonstration PC is located at D:\iar\ewarm-9.60.2\common\bin\IarIdePm.exe, set the installation path to D:\iar\ewarm-9.60.2.

Figure 10 G32R430xxAddOnvx.x.x.exe Installation Program



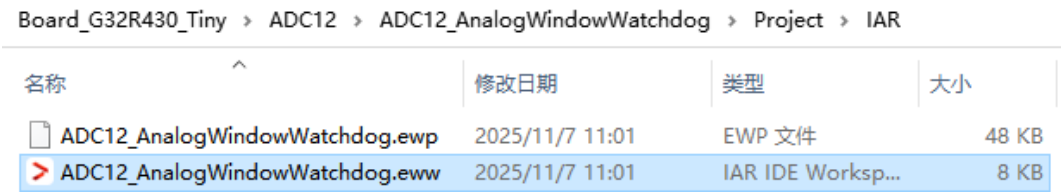
4.2.2 Use an Example Program

Open the IAR project in the same directory as the example project. For example:

```
Board_G32R430_Tiny\ADC12\ADC12_AnalogWindowWatchdog\Project\IAR
```

In this directory, double-click the ".eww" file to load the example project.

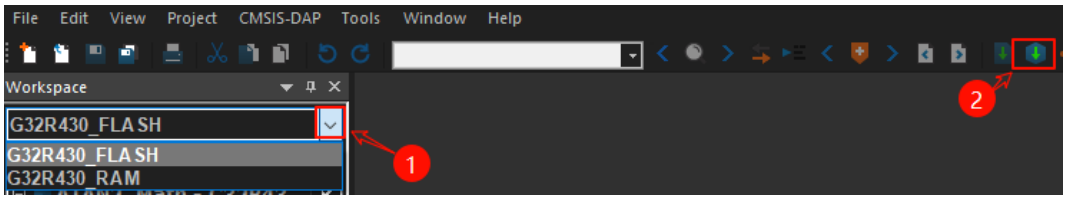
Figure 11 ADC12AnalogWindowWatchdog IAR Project Diagram



4.2.3 Compile the Program

1. After opening the project, view the project structure in the IAR Workspace.
2. Check whether the project Target configuration is the desired one.
3. Click "Make" or the corresponding compile button and wait for compilation to finish. If compilation succeeds, the Console shows no errors or warnings.

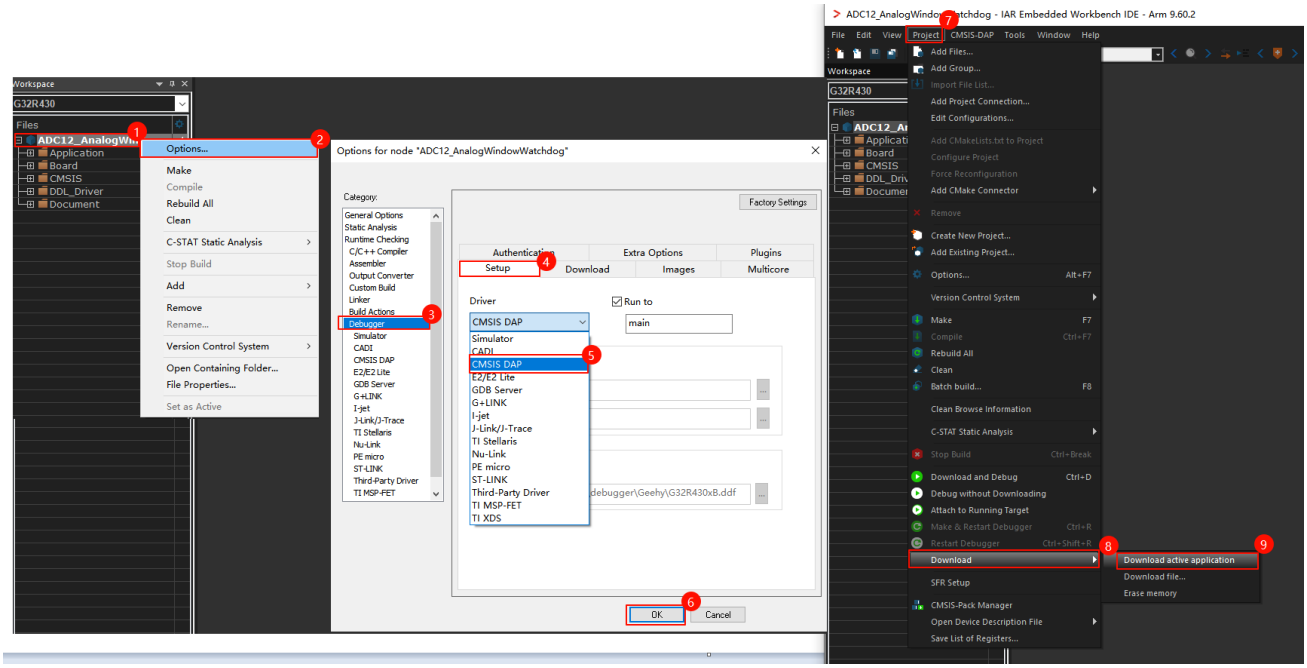
Figure 12 Multi-Target Selection and Compilation in IAR



4.2.4 Download the Program

1. In the debugger options, select "CMSIS-DAP ARMv8-M Debugger".
2. Click "Project" on the status bar, then select "Download" → "Download active application" to download the program.

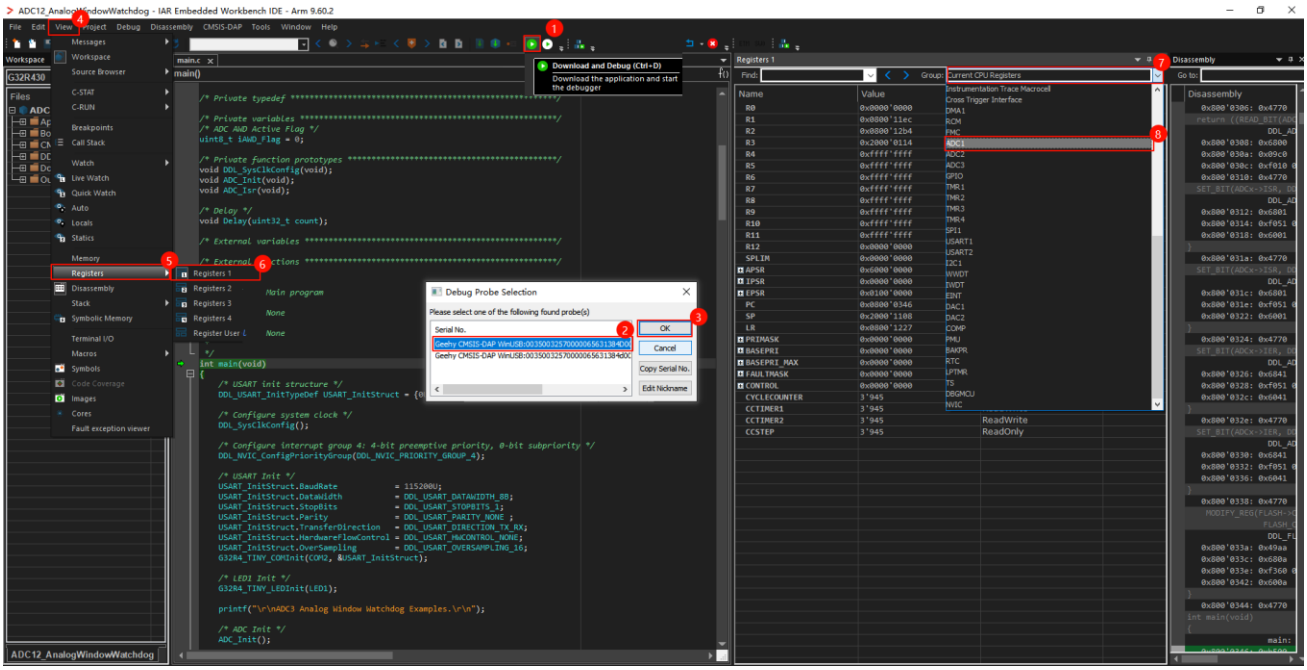
Figure 13 Configuring Emulator and Downloading Program in IAR



4.2.5 Debug the Program

Click "Download and Debug" to enter the emulation and debugging environment. The first time, an emulator selection dialog appears; select the first entry. For more operations, see Figure 14.

Figure 14 Viewing Peripheral Registers in IAR Debug Interface



In the debug windows, you can view:

- General-purpose registers (for example R0, R1 … R12, SP, LR).
- Core and peripheral registers (peripheral initialization and status).

4.3 Running Examples in Eclipse

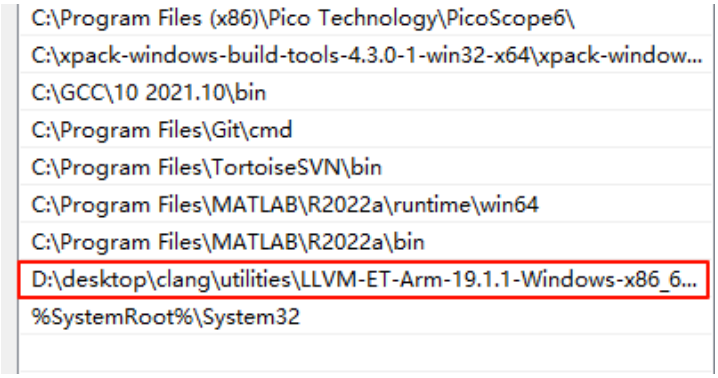
4.3.1 Toolchain Installation

Ensure you are using Eclipse 4.35 or a later IDE version.

LLVM Embedded Toolchain for Arm

1. Download “LLVM-ET-Arm-19.1.1-Windows-x86_64” from https://github.com/ARM-software/LLVM-embedded-toolchain-for-Arm.
2. Extract the downloaded package (example path: “D:\desktop\clang\utilities\LLVM-ET-Arm-19.1.1-Windows-x86_64”).
3. Add the “bin” folder to the system environment variables.

Figure 15 Add system environment variables

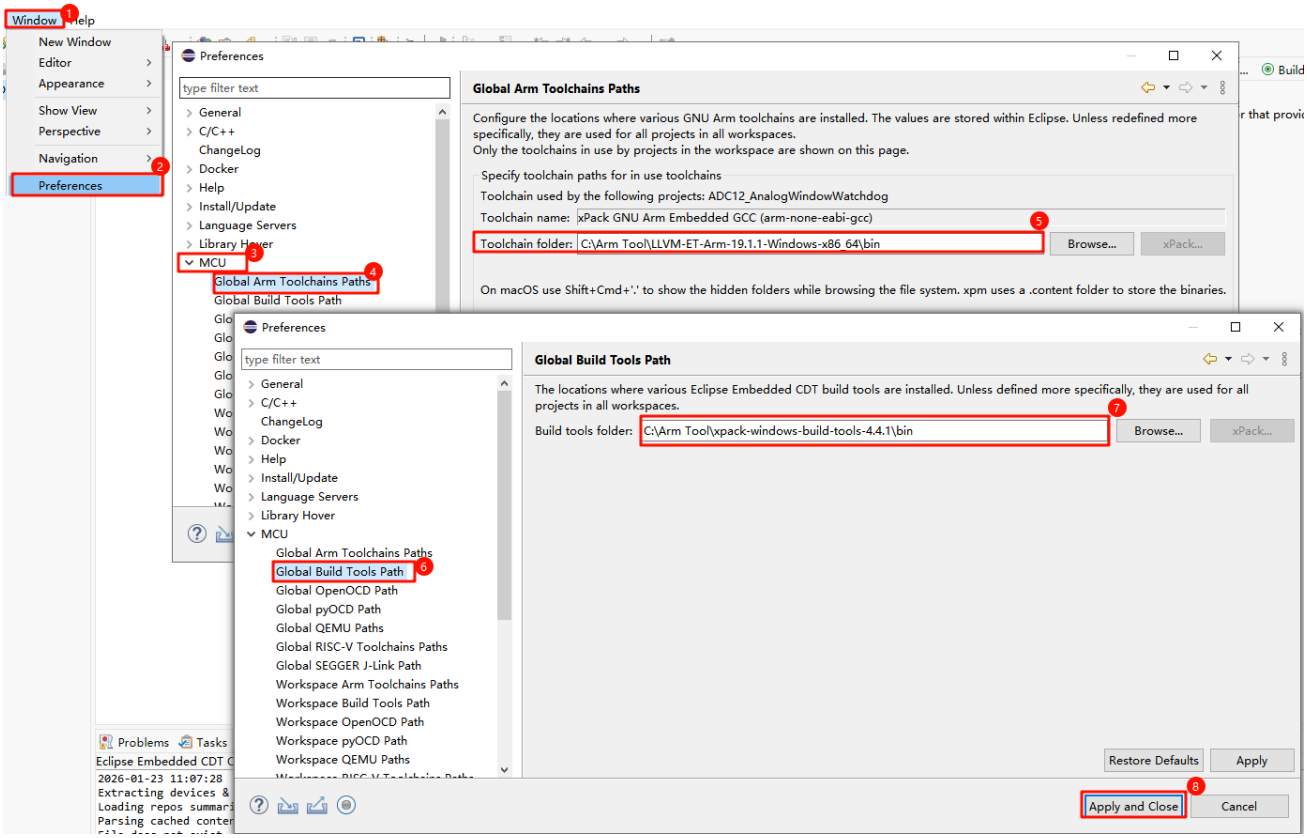


GDB and Make

- GDB service: It is recommended to use “arm-none-eabi-gdb.exe” provided by Arm. The example uses version 14.2.
- Make tool: It is recommended to use xpack-windows-build-tools; the example uses version 4.4.1.

The above toolchain can be configured in the Eclipse global Path. Refer to Figure 16.

Figure 16 Eclipse Adds MCU Global Toolchain Configuration



4.3.2 pyOCD Adaptation

To download and debug G32R430 in an open-source environment, the G32R430 series MCU requires pyOCD support.

Currently, Eclipse examples in the SDK use pyOCD 0.44 during download/debug. This version already supports the required core; users do not need to patch pyOCD source to add core ID / core name entries. However, the G32R430 device target must still be integrated manually: copy the SDK target script into the pyOCD installation tree and register the device in builtin "__init__.py". Install with "pip install pyocd==0.44" (see https://github.com/pyocd/pyOCD/releases/tag/v0.44.0).

Add G32R430 device support using the following steps (open files with Notepad, VS Code, or another text editor):

1. Copy "Utilities\pyOCD\target_G32R430xx.py" from the SDK into the installed pyOCD directory "pyocd\target\builtin" (typically under your Python install path: "Lib\site-packages\pyocd\target\builtin\").
2. Open "__init__.py" in the same directory. Near the existing "from . import target_xxx" lines, add exactly this line:

```
from . import target_G32R430xx
```

3. In the device target mapping dictionary in the same file (usually named "TARGETS", with entries like "'name': module.ClassName"), add exactly this item (keep the trailing comma):

```
'g32r430xb': target_G32R430xx.G32R430xB,
```

4. Save "__init__.py", then open CMD and run:

```
pyocd list --targets
```

5. Confirm that "g32r430xb" appears in the list; if it does, device-target adaptation is complete. Optionally run the following for a connectivity check:

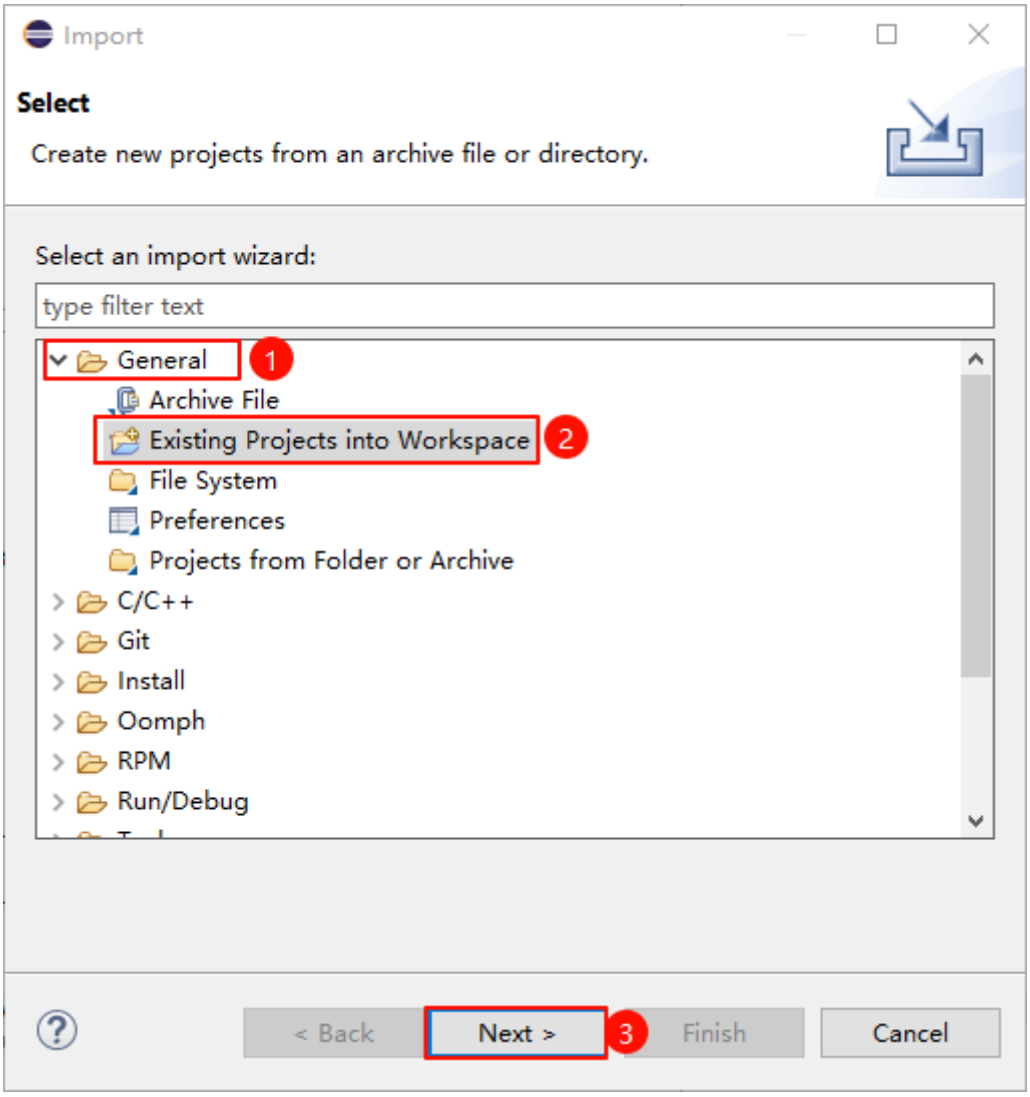
```
pyocd commander -t g32r430xb
```

4.3.3 Example Usage

1. Start Eclipse 4.35.
2. Click "File" → "Import…" → "General" → "Existing Project into Workspace" in the menu

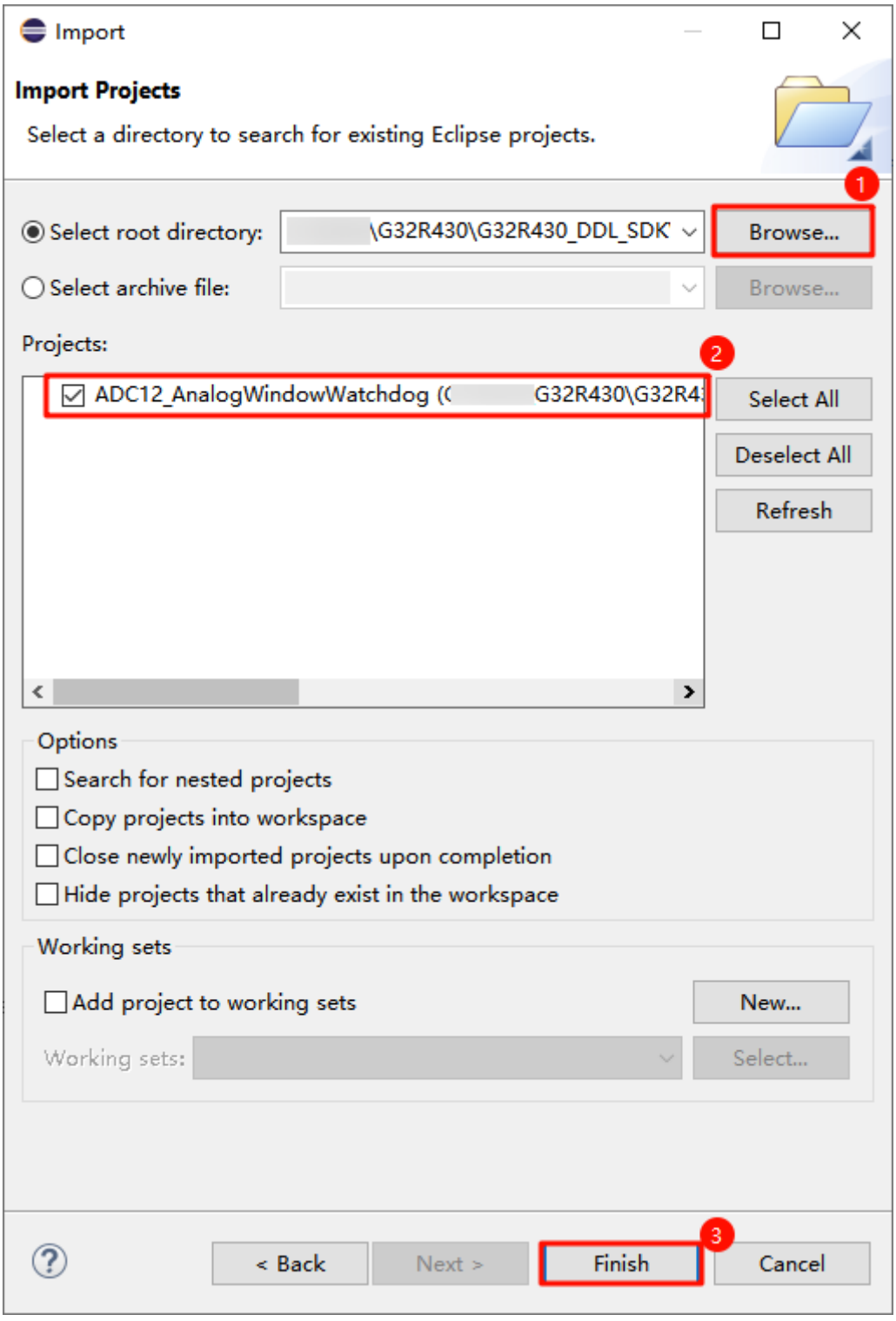
bar.

Figure 17 Import project 1



Click "Select root directory", browse to the path where you saved the SDK project files, select the corresponding project folder, and then click "Finish".

Figure 18 Import project 2



Note: Complete the LLVM toolchain configuration in section 4.3.1 before the steps above.

4.3.4 Compile the Program

1. After opening the project, view the project structure in the Eclipse Project Explorer.

2. Check whether the project Target configuration is what you need.

3. Click the "Build" button and wait for compilation to complete. If compilation succeeds, the Console shows no errors or warnings.

Figure 19 Compiling Example Operation under Eclipse

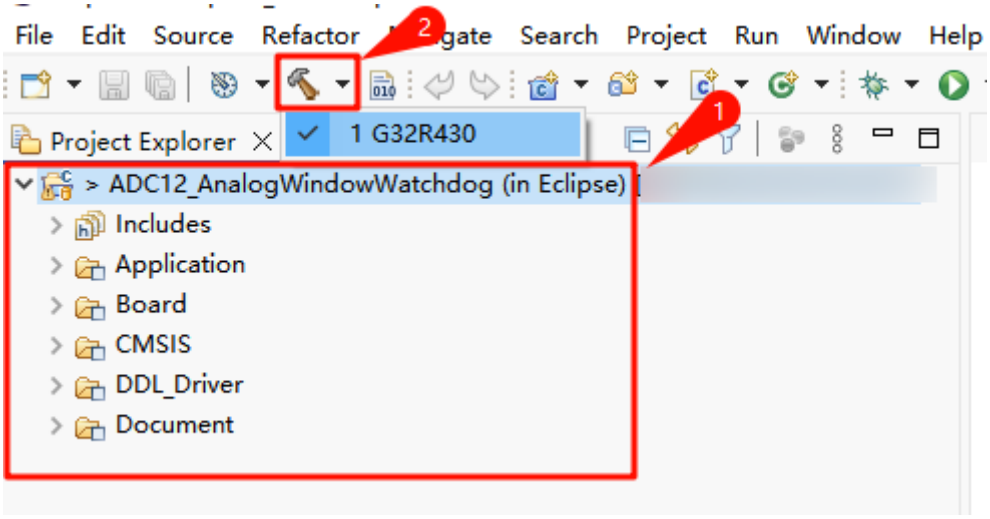


Figure 20 Successfully Compiled Example under Eclipse

4.3.5 Download and Debug the Program

Downloading and debugging programs under Eclipse requires configuring the GDB service. Configure the debug tabs as follows.

Note: Perform the following steps only after Python, pyOCD, and G32R430 pyOCD device support are ready. See section 4.3.2 and section 5 for environment setup.

Add a pyOCD Debugging configuration

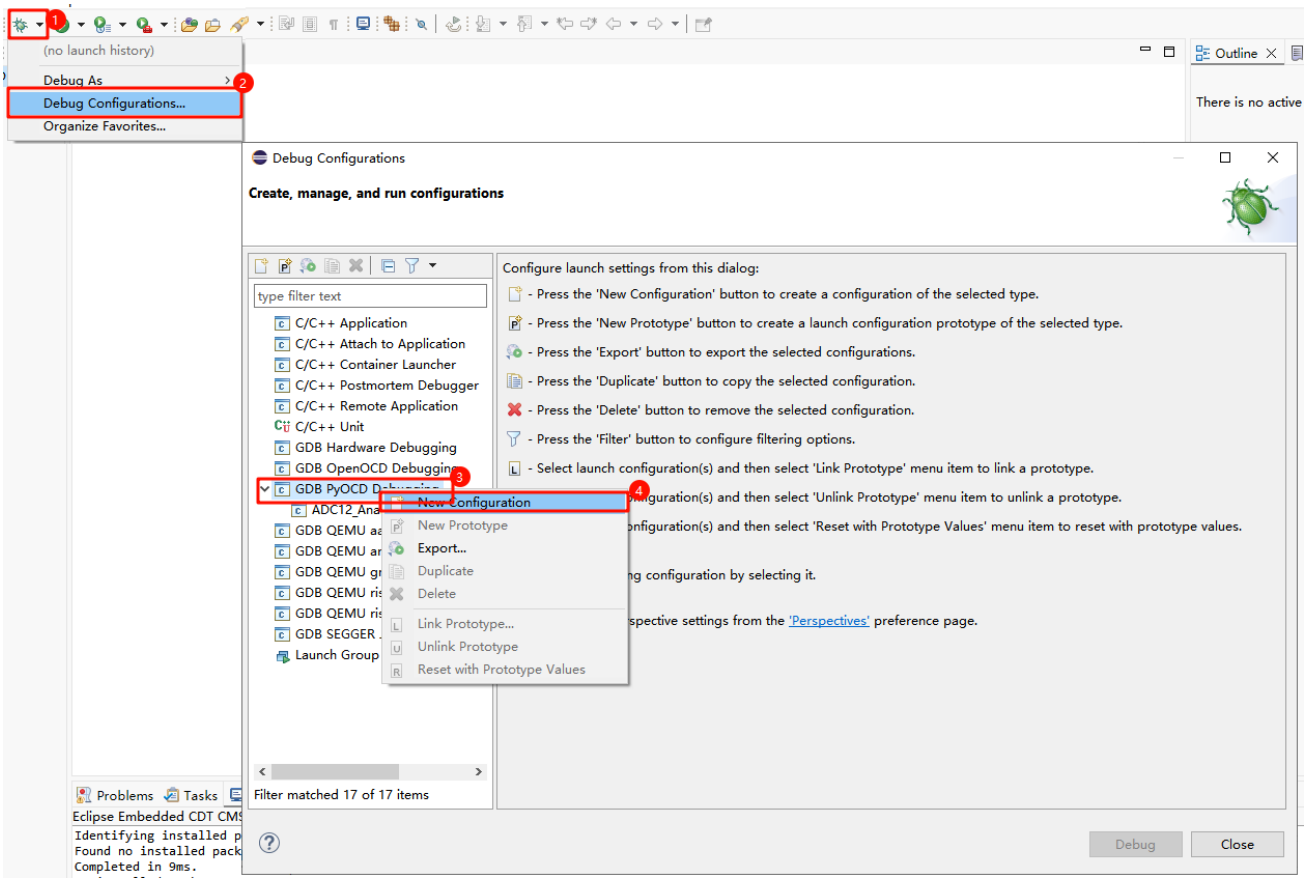
1. Left-click the Debug icon to show Debug configurations.

2. Select "Debug Configurations...".

3. In the new window, select "GDB PyOCD Debugging" and right-click.

4. Select "New Configuration".

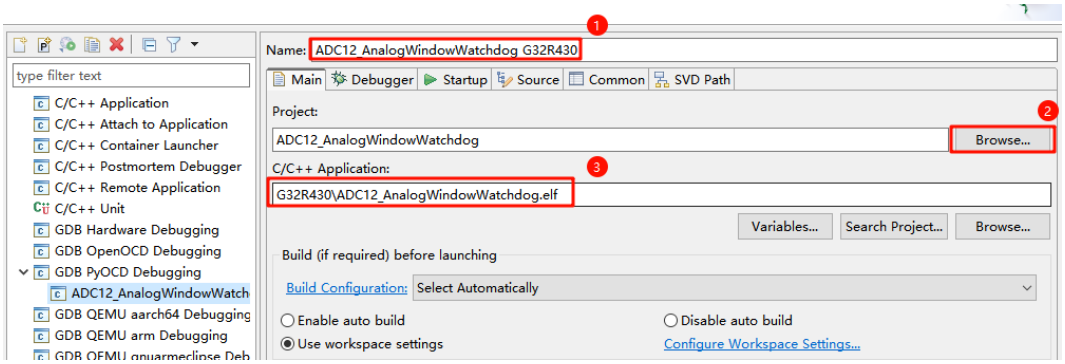
Figure 21 Add pyOCD Debugging Configuration



Configure the Main tab

1. Name the current debug configuration at the top.
2. Click "Browse..." and select the project for this configuration.
3. Select the corresponding ELF file, for example: G32R430\ADC12_AnalogWindowWatchdog.elf. The example uses a path relative to the project file; absolute paths are also supported.

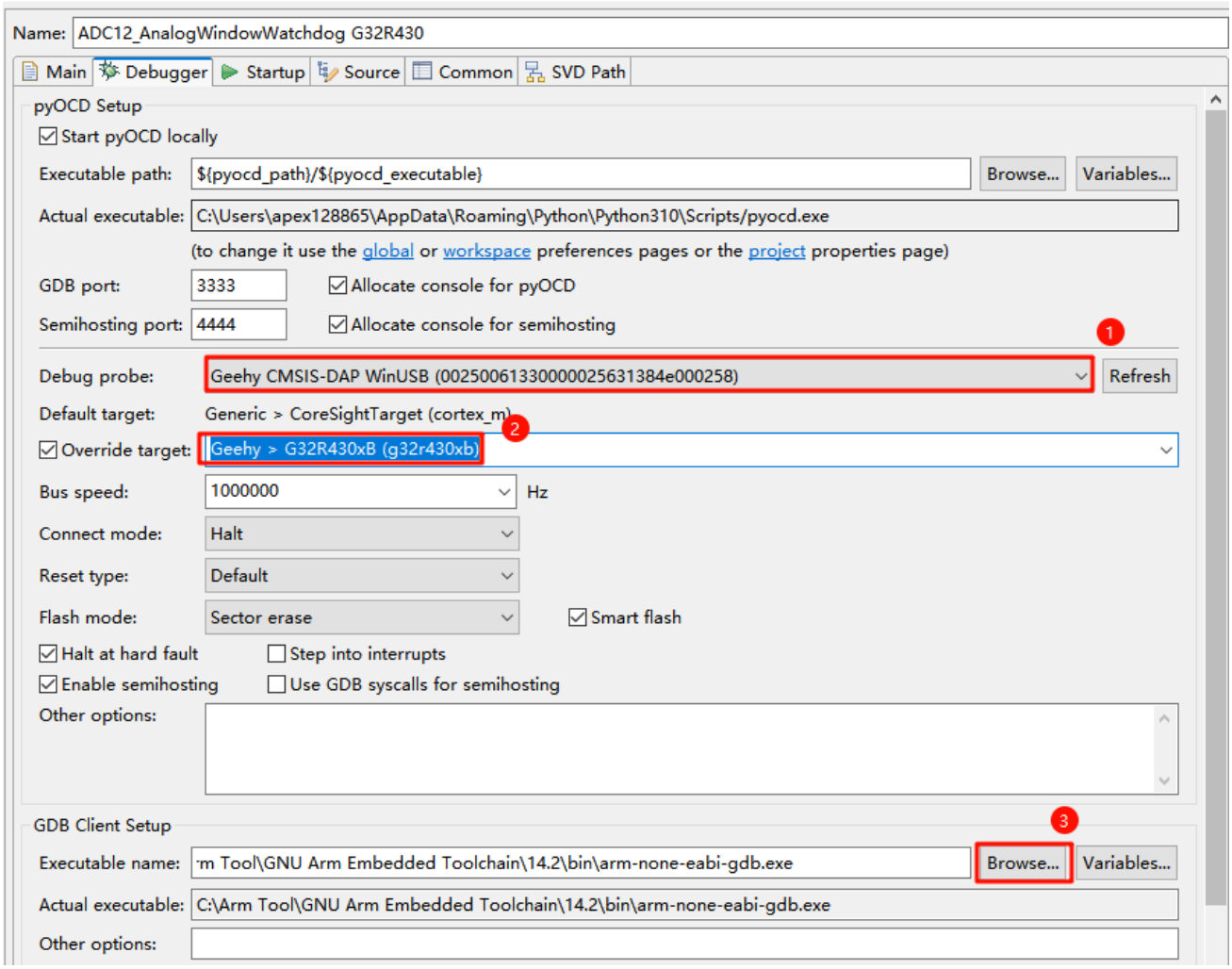
Figure 22 Configure Main



Configure the Debugger tab

1. Select the Geehy-Link probe in use; the characters in parentheses are the probe serial number.
2. Select the corresponding chip, for example: Geehy > G32R430xB (g32r430xb).
3. Select the GDB service. It is recommended to use arm-gnu-toolchain-14.2.rel1\bin\arm-none-eabi-gdb.exe provided by Arm.

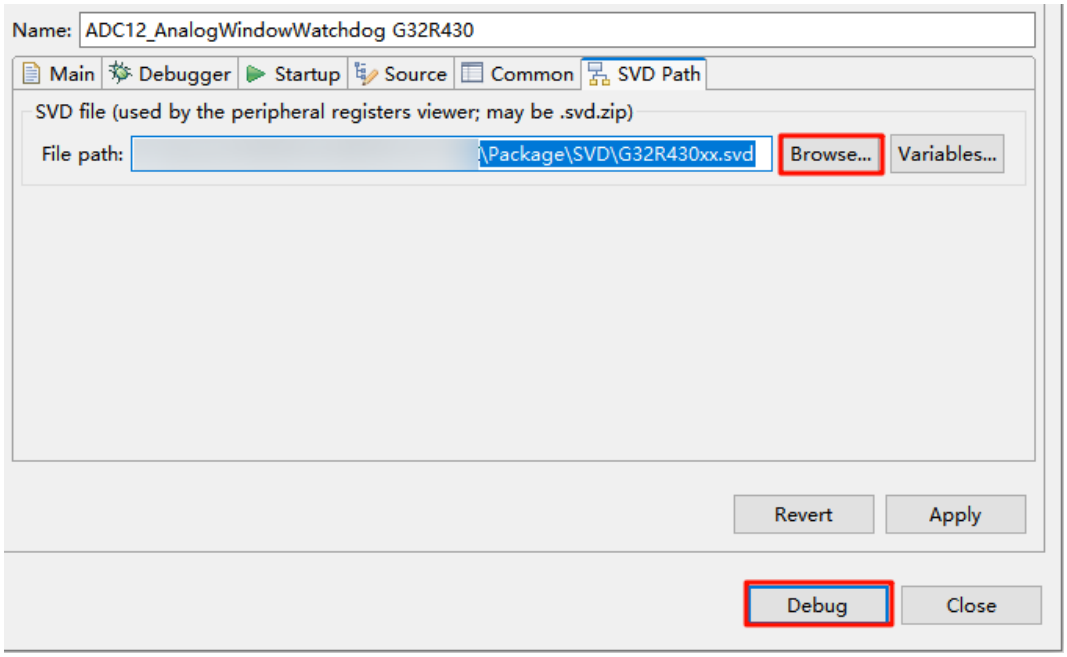
Figure 23 Configure Debugger



Configure the SVD file tab

The SVD file provides a convenient way to view MCU peripheral registers. For G32R430, select Utilities\SVD\G32R430xx.svd in the SDK.

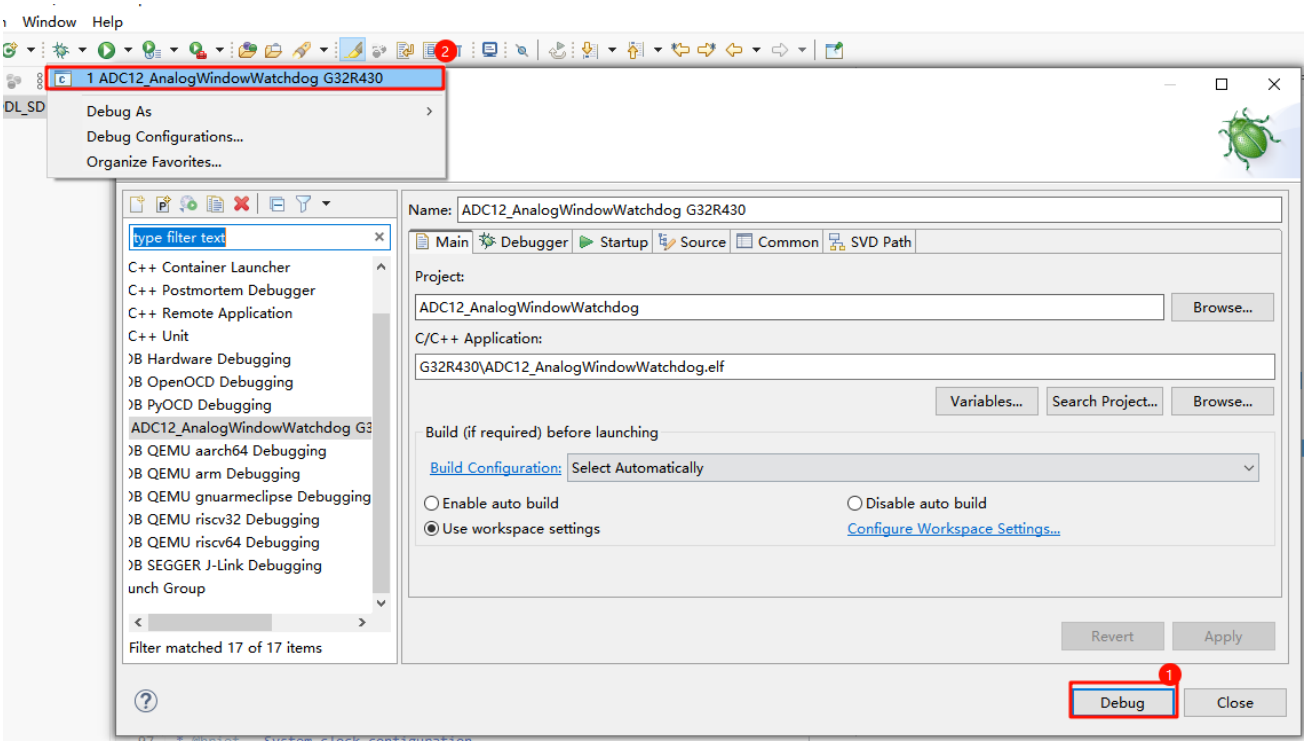
Figure 24 Configure SVD File Tab



Apply the configuration and start debugging

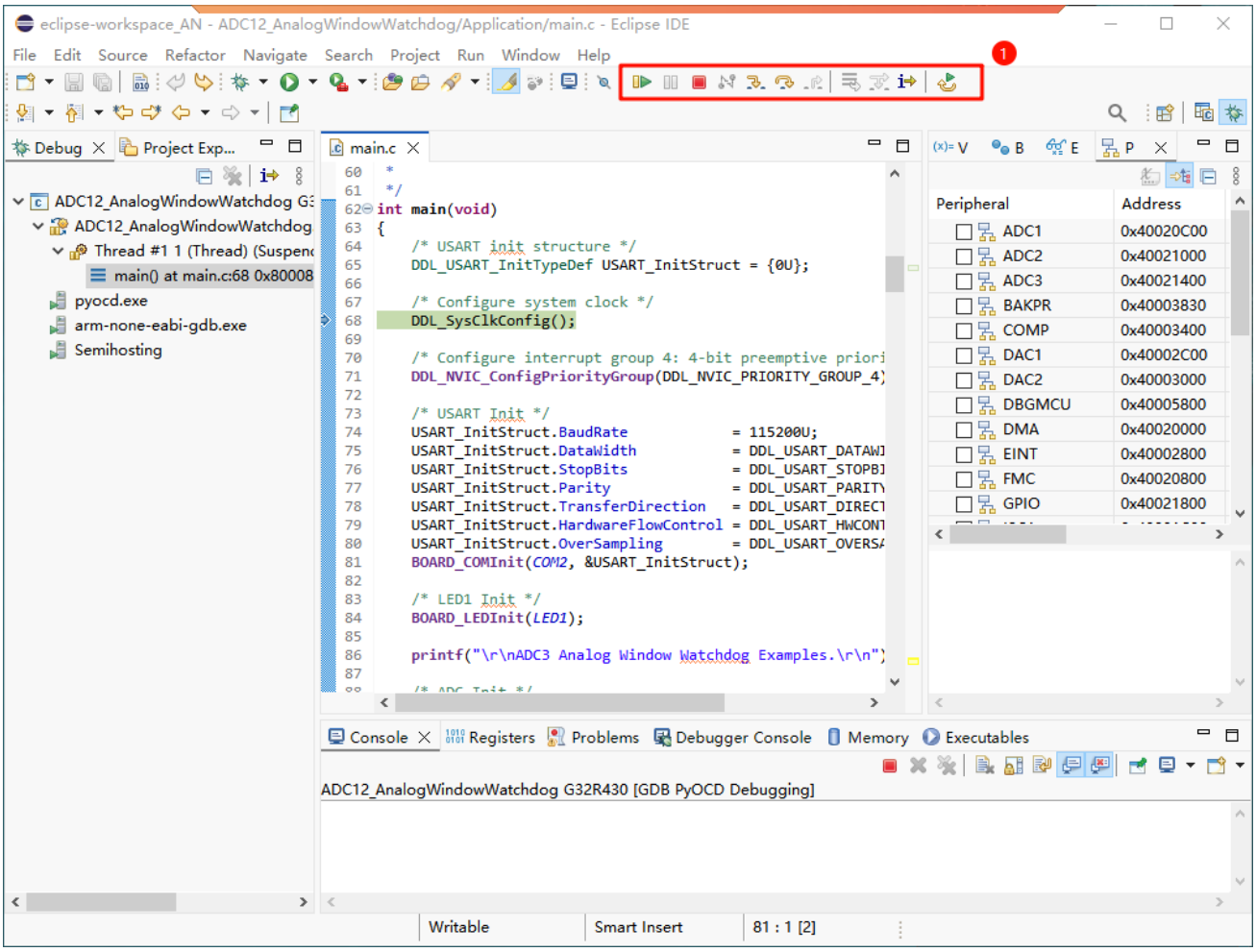
1. Click "Apply" in the lower-right corner of the tab to apply all settings.
2. Launch download or debugging: the first time, click "Debug" in the lower-right corner of the tab; afterwards you can use the Debugger button on the toolbar.

Figure 25 Launch Download or Debugging



After a successful debug session, use the toolbar debug buttons to control the program.

Figure 26 Debugging Successfully



5 pyOCD Installation

5.1 Windows

5.1.1 Install Python

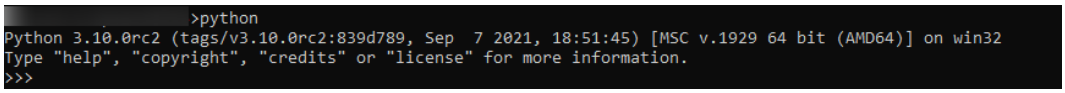
pyOCD requires a Python environment. Download the latest Python installer from https://www.python.org.

Note: After installation, ensure Python is added to the system PATH so it can be used from the command line.

Verification steps:

1. Press Win+R, enter CMD, and press Enter.
2. In the command window, enter “python” and press Enter.
3. Confirm that the installed Python version is shown and that you enter the Python interactive prompt (REPL).
4. To exit, enter “exit()” or “quit()”, then press Enter.

Figure 27 Python Installation Verification



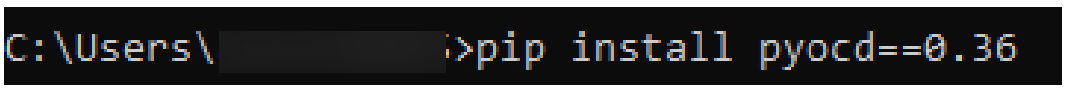
5.1.2 Install pyOCD

pyOCD is a Python package. Online installation is recommended so that dependencies are installed together.

Install and verify as follows:

Install pyOCD 0.44 with pip: pip install pyocd==0.44.

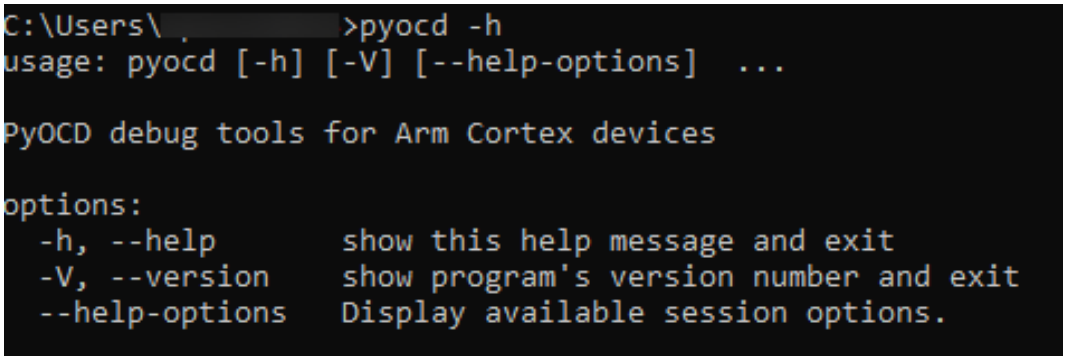
Figure 28 Install pyOCD



After installation:

1. Add the pyOCD Scripts path to the system PATH for command-line use. For example: C:\Users\Geehy\AppData\Local\Programs\Python\Python313\Scripts.
2. Press Win+R, enter CMD, then run pyocd -h and confirm that command help is shown.

Figure 29 pyOCD Installation Verification



5.2 Ubuntu

5.2.1 Install Python

Ubuntu generally comes with Python by default. You can enter the following commands in the terminal to query the Python and pip versions.

```
python --version
```

If Python or pip is not installed, use the following commands to install them:

```
sudo apt update
sudo apt install python3 python3-pip
```

5.2.2 python3-venv

Some Ubuntu-native Python 3 environments are externally managed, so packages cannot be installed into the global environment with pip. To avoid this, use Python's built-in venv module.

Install the “python3-venv” package:

```
sudo apt install python3-venv
```

Create a virtual environment (named “venv” in this example):

```
python3 -m venv venv
```

Activate the virtual environment:

```
source venv/bin/activate
```

To leave the virtual environment later, run:

```
deactivate
```

5.2.3 Install pyOCD

In the activated virtual environment, install pyOCD with pip:

```
pip install pyocd==0.44
```

Verify the installation:

```
pyocd --version
```

Query the pyOCD install location (for PATH / global configuration):

```
pip show pyocd
```

5.2.4 pyOCD USB Permissions

If pyOCD cannot access the debugger as a normal user, grant appropriate permissions. Use udev rules so USB devices can be accessed without sudo. Steps:

1. Create a new udev rules file in the terminal (for example “99-pyocd.rules”):

```
sudo nano /etc/udev/rules.d/99-pyocd.rules
```

2. Add the following content to the file (Geehy-Link vendor ID is 314B):

```
SUBSYSTEM=="usb", ATTR{idVendor}=="314b", MODE="0666"
```

3. In nano, press Ctrl+O to save, press Enter to confirm, then press Ctrl+X to exit.

4. Reload the udev rules:

```
sudo udevadm control --reload-rules
sudo udevadm trigger
```

5. Verify that pyOCD can recognize Geehy-Link:

```
pyocd list
```

Figure 30 Emulator Serial Number Connected to Ubuntu

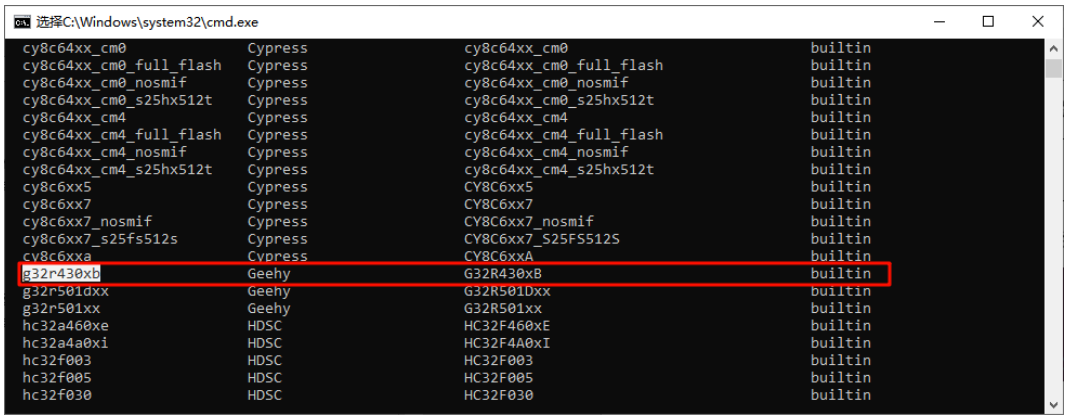
```
(venv) kai@Ubuntu:~$ pyocd list
  #   Probe/Board             Unique ID                        Target
------------------------------------------------------------------------
  0   Geehy CMSIS-DAP WinUSB  00350043500000144e544859000258   n/a
```

5.3 Replace Modified Item Content

After installing pyOCD 0.44 and completing the steps above (copy “target_G32R430xx.py” into “pyocd\target\builtin”; add “from . import target_G32R430xx”; and add “'g32r430xb': target_G32R430xx.G32R430xB,” in “__init__.py”), verify G32R430 support as follows:

1. Press Win+R and enter CMD.
2. Run “pyocd list --targets”.
3. Check whether “g32r430xb” appears in the supported target list.

Figure 31 List of Supported Chips



5.4 Command Line Usage

pyOCD can be used from the CMD command line. Confirm that pyOCD is on PATH before use. Steps:

1. Connect the board or probe to the chip, and connect to the PC.
2. Open CMD in the working directory and run “pyocd commander -t g32r430xb”.
3. Enter further commands in the command window as needed.

Figure 32 pyOCD commander

```
G:\>pyocd commander -t g32r430xb
C:\Users\apex128865\AppData\Roaming\Python\Python310\site-packages\capstone\__init__.py:267: UserWarning: pkg_resources
is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html. The pkg_resources package is slat
d for removal as early as 2025-11-30. Refrain from using this package or pin to Setuptools<81.
  import pkg_resources
Connected to G32R430xB [Running]: 00250061330000025631384e000258
pyocd> erase
pyocd> rw 0x08000000 0x10
08000000:  ffffffff ffffffff ffffffff ffffffff    |................|
pyocd>
```

5.5 Integration with Eclipse

Using Eclipse with pyOCD has IDE version requirements. Eclipse 202503 is recommended.

Also configure the pyOCD path globally in Eclipse (on some PCs, Eclipse may fail to pick up PATH). Steps:

1. Right-click "Window" in the menu bar to show all configurations.
2. Select "Preferences".
3. In the new window, expand the "MCU" options.
4. Select "Global pyOCD Path".
5. On the right, select the directory that contains “pyocd.exe”.
6. Click "Apply and Close".

Figure 33 Global pyOCD Path Steps 1-2

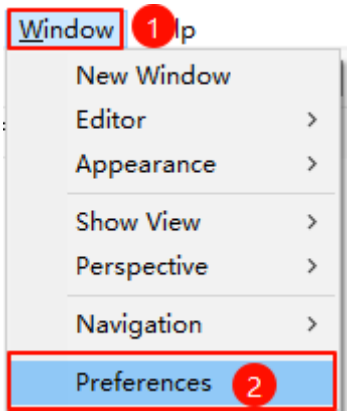
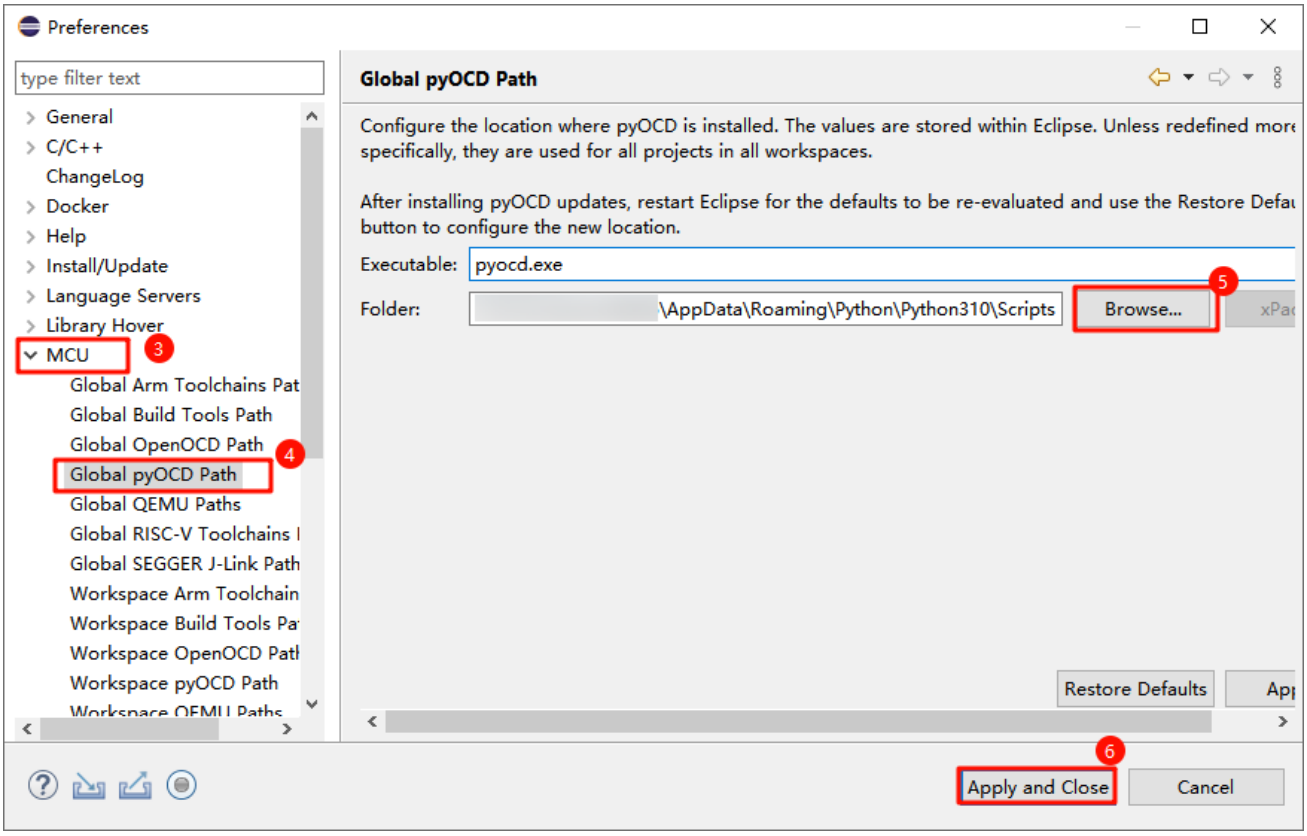


Figure 34 Global pyOCD Path Steps 3-6



6 About Linker Script

A linker script is a configuration file that determines the placement and structure of various memory segments of a program. In embedded development, it is used to map compiled object code, constants, global variables, and interrupt vectors to the MCU's physical memory areas such as Flash and RAM. This section introduces the storage locations, corresponding memory structures, and common example memory configuration files for the G32R430 series across three development environments: MDK, IAR, and Eclipse (gcc). This will help you select an appropriate placement strategy for your project and understand the roles of subsequent fields and sections.

6.1 Basic Information of Linker Script

Storage locations are as follows:

- MDK: Libraries\Device\Geehy\G32R4xx\Source\arm
- IAR: Libraries\Device\Geehy\G32R4xx\Source\iar
- GCC: Libraries\Device\Geehy\G32R4xx\Source\gcc

The chip memory structure (G32R430xB) is shown in Table 1.

Table 1 G32R430xB Memory Structure

Memory Region	Size
Flash	128 KB
DTCM RAM	16 KB
ITCM RAM	32 KB

Example memory configuration files are described in Table 2.

Table 2 Example Memory Configuration Files

Configuration File	Description
g32r430xb_flash.sct/icf/ld	The program runs in Flash, the stack is located in DTCM RAM, and the ITCM RAM space is unused.
g32r430xb_itcm_ram.sct/icf/ld	The program runs in ITCM RAM, the stack is located in DTCM RAM, and the Flash space is unused.

Configuration File	Description
g32r430xb_flash_option.ld	The program runs in Flash, the stack is located in DTCM RAM, and the ITCM RAM space is unused. It also includes the Option Byte configuration area and is used for the "\Examples\Board_G32R430_Tiny\FLASH\FLASH_OPT" program.

Note: The above configurations are used for performance comparison between different storage areas and optimization for specific scenarios. Please select the appropriate script based on the project requirements and resource constraints.

6.2 Field Description

Several fields are defined in the “g32r430.h” header file. These fields correspond to specific section definitions in the linker scripts. Common fields and their meanings are as follows:

- “SECTION_ITCM_INSTRUCTION”: ITCM instruction code section
- “SECTION_ITCM_RAMFUNC”: Function section in ITCM RAM (functions executed in RAM)
- “SECTION_DTCM_DATA”: DTCM data section
- “SECTION_DTCM_BSS”: DTCM BSS (uninitialized global/static variables) section
- “SECTION_RAM_VEC”: The section containing the interrupt vector table, usually placed in DTCM RAM. Users can modify the definition location in the linker script to modify its storage location.

6.3 How to Specify Variable/Function Storage Area

By placing variables and functions in different memory areas, better startup time, execution efficiency, and memory utilization for different application scenarios can be obtained. This section provides commonly used partitions and corresponding code examples, facilitating quick placement of target areas in practical projects.

Example objective:

Place the data sensitive to startup time and execution speed into DTCM RAM to obtain lower access latency.

Note: The RAM memory addresses accessed by DMA must be located in DTCM RAM.

```
SECTION_DTCM_DATA volatile uint32_t g_fastConfigFlag = 0;
```

Place functions sensitive to startup time or requiring fast execution into the ITCM area (instruction area/RAMFUNC area) to improve startup and execution efficiency.

```
SECTION_ITCM_RAMFUNC void fast_filter(uint8_t *data, size_t len)
{
}
```

Or

```
SECTION_ITCM_INSTRUCTION void fast_filter(uint8_t *data, size_t len)
{
}
```

7 ATAN2 Libraries

To meet the requirements for angle calculation in certain application scenarios, the G32R430 MCU provides an ATAN2 library, which can quickly calculate the angle value of the given coordinates (nX, nY). This function has wide application value in such fields as motor control and trajectory planning.

7.1 ATAN2 Library File Structure

Library files for different compilers and precision versions are provided in the "Libraries/ATAN2" directory of SDK, mainly including:

- g32r430_ATAN2_high_resolution_AC6.lib
- g32r430_ATAN2_low_resolution_AC6.lib
- g32r430_ATAN2_high_resolution_ICC.a
- g32r430_ATAN2_low_resolution_ICC.a

Wherein:

"AC6" indicates the library file is suitable for the MDK (AC6 compiler) environment.

"ICC" indicates the library file is suitable for the IAR (ICC compiler) environment.

"high_resolution" and "low_resolution" represent different precision implementations. Users can select the appropriate library based on their application requirements.

7.2 Function Prototype and Description

The ATAN2 function allows users to calculate the angle on a two-dimensional plane for given coordinates by specifying the precision level. The specific prototype is as follows:

int32_t ATAN2(int32_t nX, int32_t nY, int32_t nPrecisionLevel);

Parameter description:

nX, nY: Represent the X and Y coordinates of the point to be calculated (entered in Q32 format), respectively.

nPrecisionLevel: Calculated precision level, within the range of [1, 8]. It is recommended to use 6, 7, or 8;

Returned value:

Returns the angle value within the range of (−π, π] in Q31 format. The origin (0, 0) is a special case. The return result requires the user to handle or judge based on the application scenario.

7.3 Function Storage and Execution Location

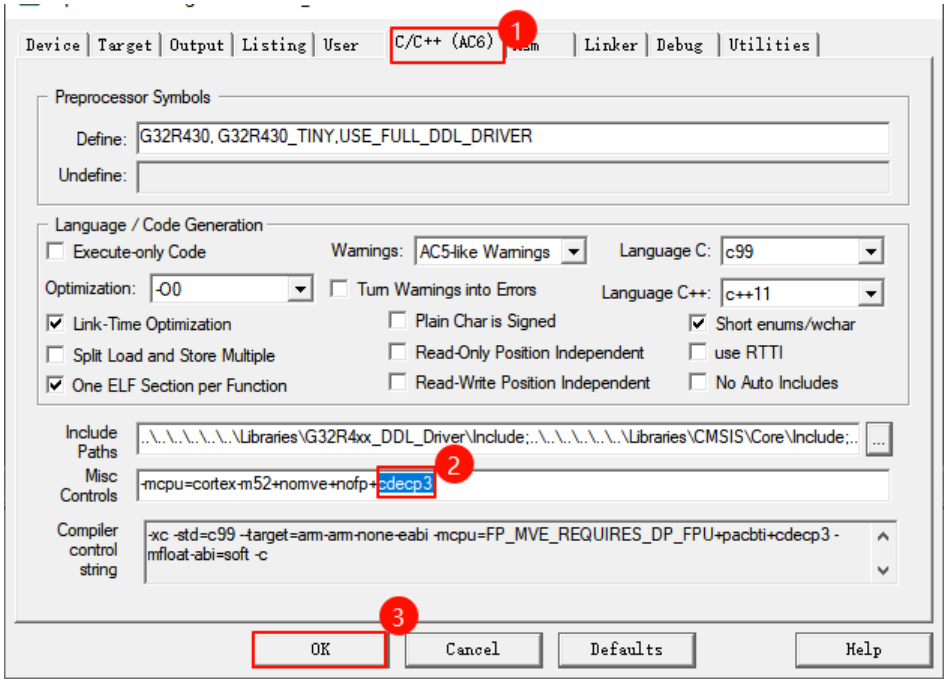
The code section for the ATAN2 function is "atan2_instruction" by default. The address of this section can be modified in the linker script (e.g., .sct or .icf files) or be mapped to the required storage space. If there are specific performance or memory requirements, the linker script can be adjusted according to the project requirements.

7.4 Usage

Select the appropriate library file: based on the compiler (MDK or IAR) and precision requirement (high/low_resolution), include the corresponding .lib or .a file in the project linker options.

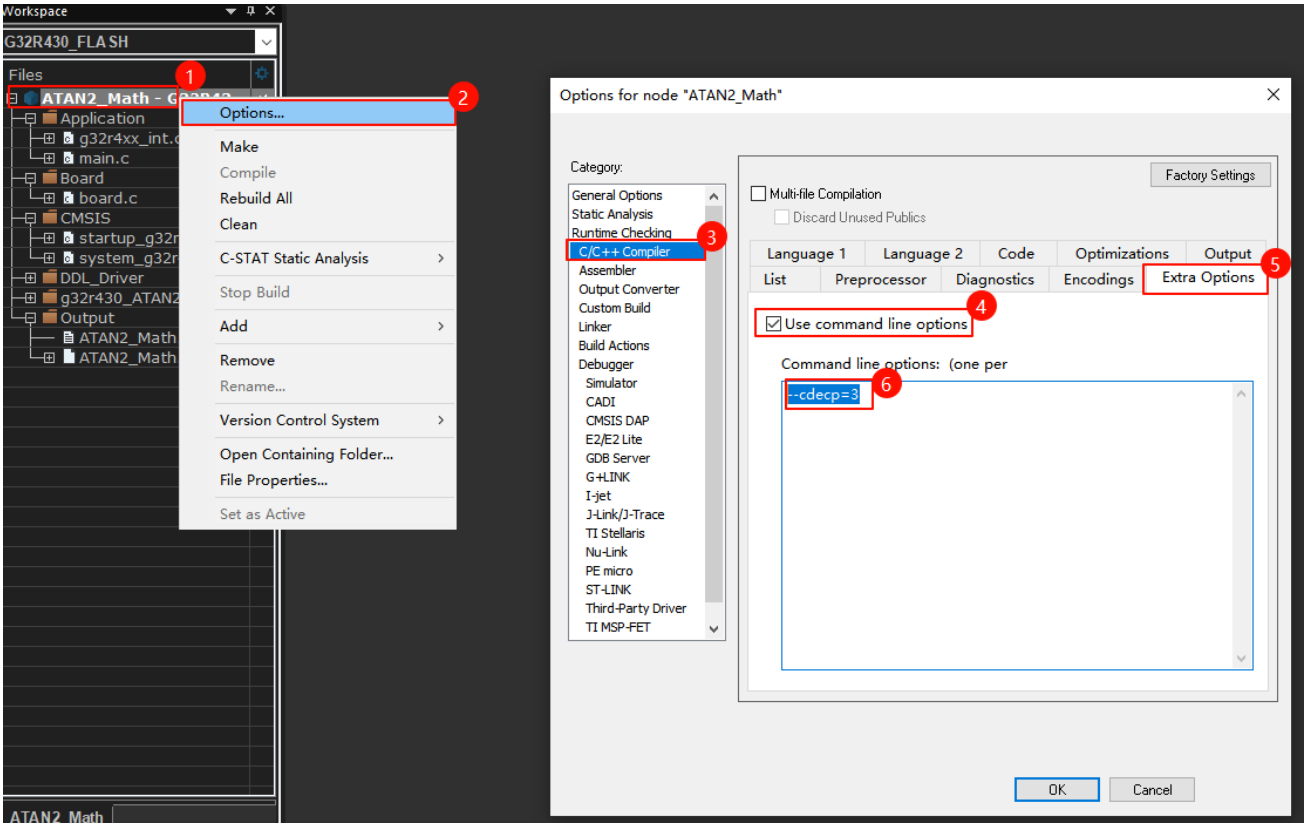
Enable CDE3 support in the project. For MDK, enable CDE3 support in the project settings.

Figure 35 Enabling CDE3 Support for C Code in MDK



For IAR, enable the corresponding CDE3 extended instruction set in the project compiler options.

Figure 36 Enabling CDE3 Support for C Code in IAR



Include the ATAN2 header in the project: open the project settings and ensure that the Libraries\ATAN2 directory has been added to the Include path; add the relevant header include at the top of the source file that calls ATAN2.

Note: For example programs, see G32R430_DDL_SDK_Vx.x.x\Examples\Board_G32R430_Tiny\ATAN2\ATAN2_Math\

8 Revision

Document revision is shown in Table 3.

Table 3 Document revision

Date	Version	Description
November 2025	1.0	● New
December 2025	1.1	● Section 2.1: updated the board picture to V1.2 ● Deleted the operational content required for the V1.1 version
January 2026	1.2	● Added Section 4.3: running examples in Eclipse ● Added Section 5: pyOCD installation and G32R430 device support ● Section 6: added linker-script descriptions for the gcc environment
2026.8	1.3	● Aligned with SDK Utilities layout ● Upgraded pyOCD to 0.44 ● Removed steps for patching pyOCD core support

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “TM” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property. Any information regarding the application of the product, Geehy

hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in the sales order or the sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If losses are caused to users due to user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY

DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY'S PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED FOR USE AS CRITICAL COMPONENTS IN MILITARY, LIFE-SUPPORT, POLLUTION CONTROL, OR HAZARDOUS SUBSTANCES MANAGEMENT SYSTEMS, NOR WHERE FAILURE COULD RESULT IN INJURY, DEATH, PROPERTY OR ENVIRONMENTAL DAMAGE.

IF THE PRODUCT IS NOT LABELED AS "AUTOMOTIVE GRADE," IT SHOULD NOT BE CONSIDERED SUITABLE FOR AUTOMOTIVE APPLICATIONS. GEEHY ASSUMES NO LIABILITY FOR THE USE BEYOND ITS SPECIFICATIONS OR GUIDELINES.

THE USER SHOULD ENSURE THAT THE APPLICATION OF THE PRODUCTS COMPLIES WITH ALL RELEVANT STANDARDS, INCLUDING BUT NOT LIMITED TO SAFETY, INFORMATION SECURITY, AND ENVIRONMENTAL REQUIREMENTS. THE USER ASSUMES FULL RESPONSIBILITY FOR THE SELECTION AND USE OF GEEHY PRODUCTS. GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDES THE DOCUMENT AND PRODUCTS

"AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT AND PRODUCTS (INCLUDING BUT NOT LIMITED TO LOSSES OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES). THIS COVERS POTENTIAL DAMAGES TO PERSONAL SAFETY, PROPERTY, OR THE ENVIRONMENT, FOR WHICH GEEHY WILL NOT BE RESPONSIBLE.

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.