Application Note

Document No.: AN1166

G32R430 DDL SDK Quick Start Guide

——Based on G32R430TinyBoard

Version: V1.2

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:

1. Windows 10 or above operating system
2. G32R430 DDL SDK Vx.x.x
3. Keil MDK 5.40 or above version
4. IAR for ARM 9.60.2 or above version
5. Eclipse 4.35 or above version
6. xpack-windows-build-tools 4.4.1
7. LLVM-ET-Arm-19.1.1-Windows-x86_64
8. arm-gnu-toolchain-14.2.rel1-mingw-w64-x86_64-arm-none-eabi
9. PyOCD 0.36
10. G32R430TinyBoard V1.2
11. 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

1. 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.

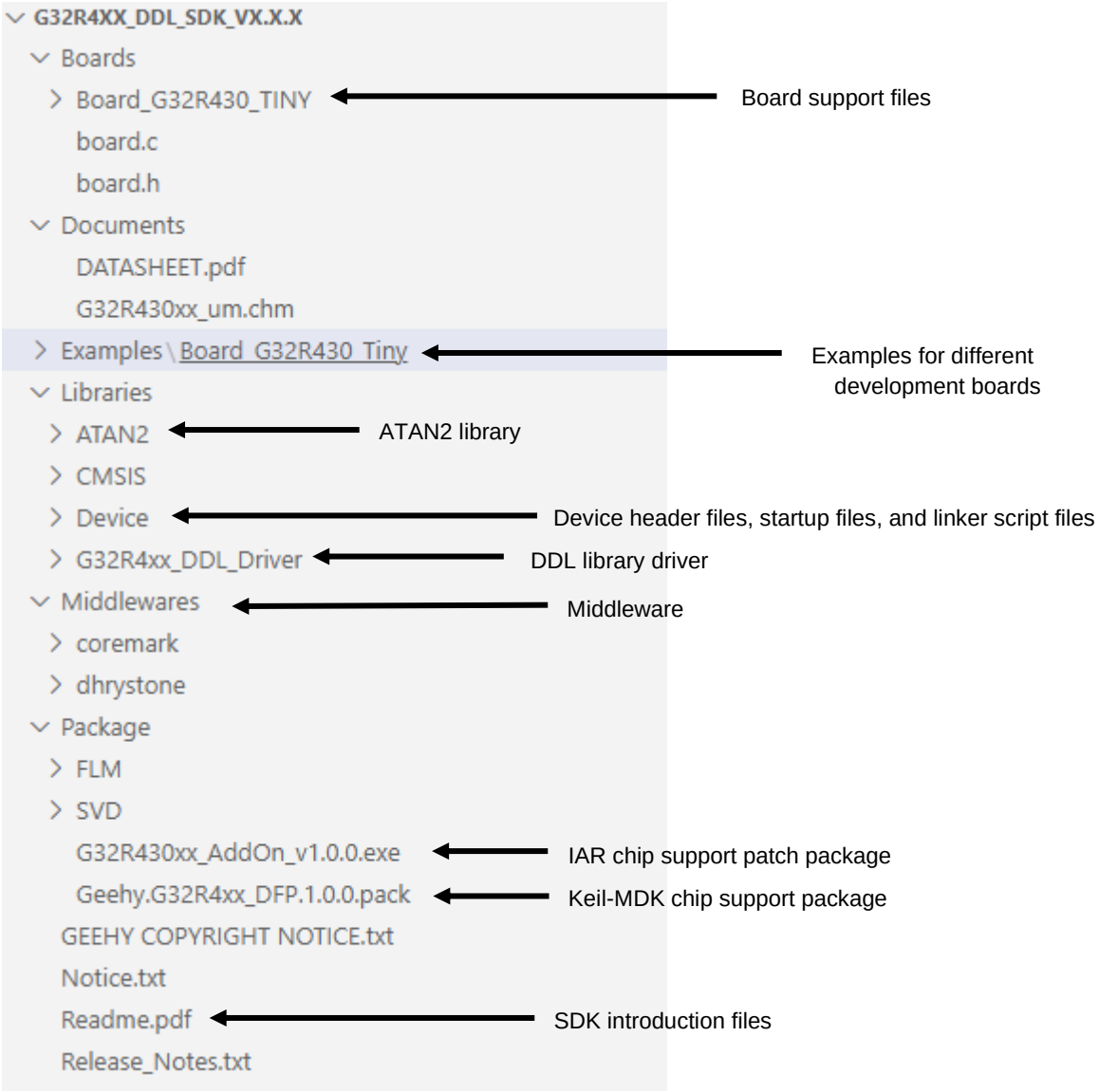
Figure 2 From Default LED1, LED2, KEY Resources to A1, A2, A5

2. 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 shown in Figure 3.

Figure 3 Introduction to G32R430 DDL SDK Directory Structure



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

4 How to Run Examples

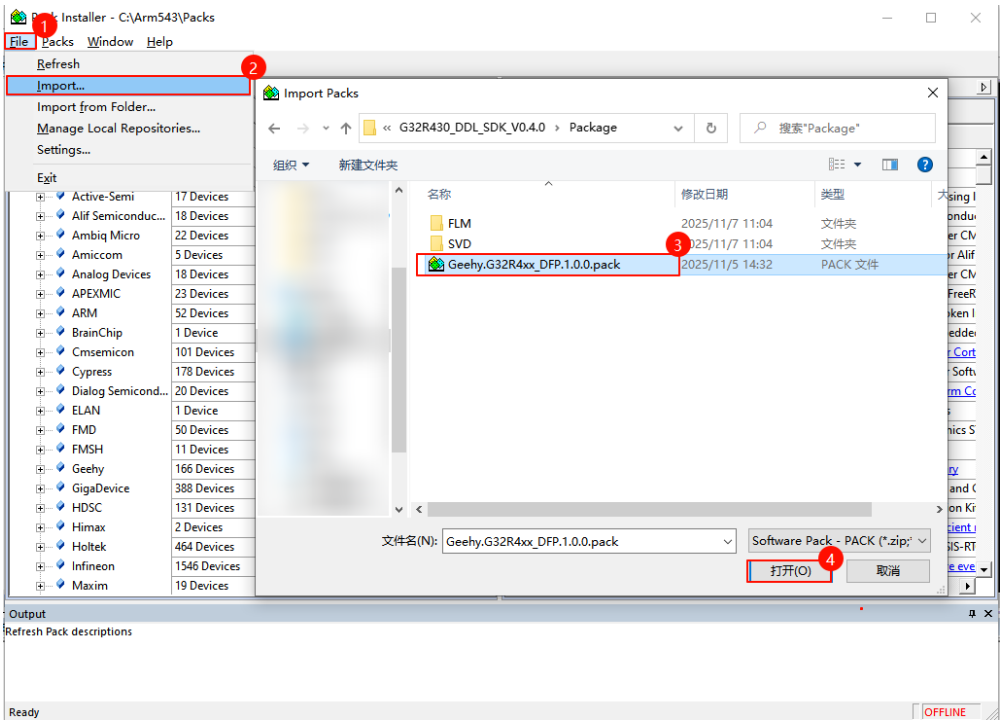
To help users get started quickly and verify the board and SDK functions, the SDK provides example projects in both the MDK, IAR and Eclipse environments. The following will respectively introduce the preparatory work that needs to be done in these development environments, as well as how to compile, 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 4 Using PackInstaller to Install Geehy.G32R4xx_DFP.x.x.x.pack



How to open PackInstaller.exe:

- Method 1: Click the "Pack Installer" icon on the MDK status bar.
- Method 2: Run the PackInstaller.exe from the MDK installation directory, e.g., C:\Users\Geehy\AppData\Local\Keil_v543a\UV4\PackInstaller.exe.

Note (1): MDK version must be ≥ v5.40. It is recommended to use MDK v5.43a to ensure compatibility and stability.

Note (2): For MDK v5.43a, double-click to install the Package\Geehy.G32R4xx_DFP.x.x.x.pack chip support package in the SDK root directory. Abnormal lag may occur, which indicates MDK is abnormal

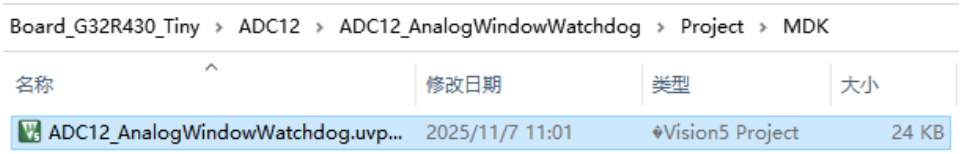
4.1.2 Use an Example Program

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.

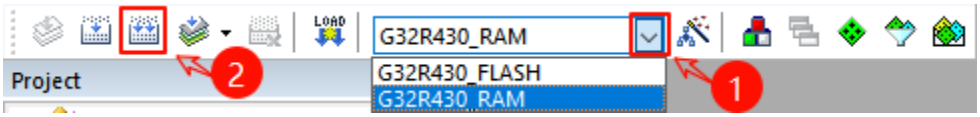
Figure 5 ADC12_AnalogWindowWatchdog MDK Project Diagram



4.1.3 Compile the Program

After opening the project, check if the project target configuration is the desired one. Click the Compile button and wait for completion. If there are no errors, MDK will display "0 error, 0 warning" in the output box.

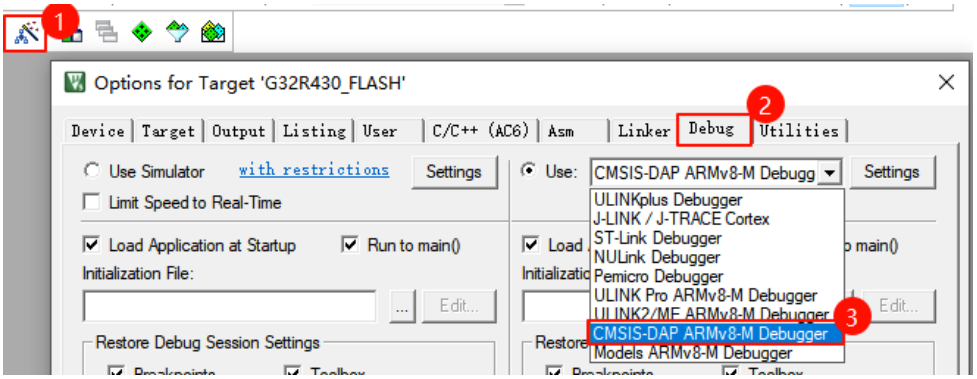
Figure 6 Multi-Target Selection and Compilation in MDK



4.1.4 Download the Program

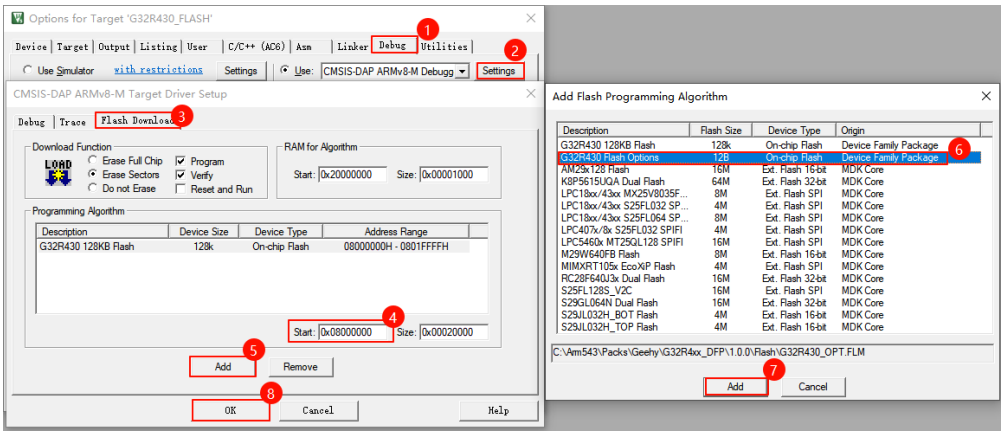
Select the emulator type as "CMSIS-DAP ARMv8-M Debugger".

Figure 7 Selecting an Emulator in MDK



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

Figure 8 Configuring Flash Download Content



Including:

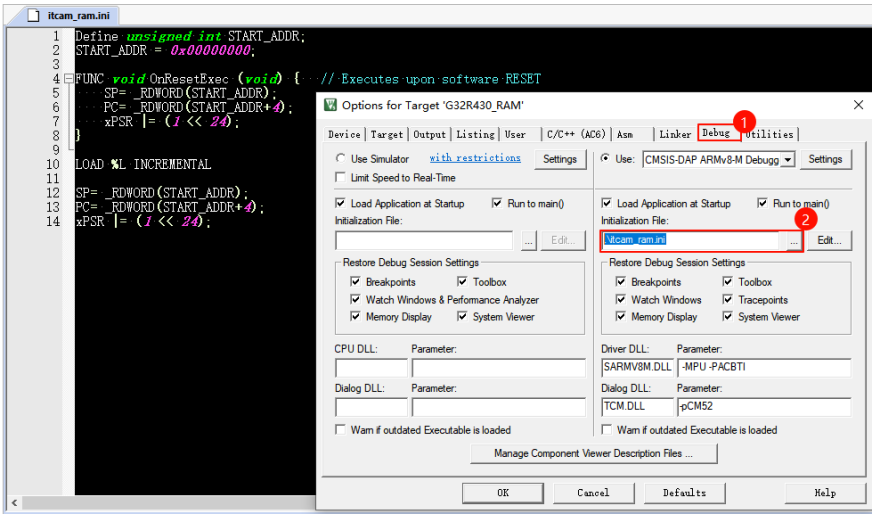
- OPT code download settings (if it is necessary to write to the emulated OTP/configuration area). Steps 5, 6, 7 in Figure 8 need to be additionally executed to add the OPT download algorithm.
- RAM program download settings (if it is necessary to run the program in RAM). Then, in Step 4 of Figure 8, change the programming area to the download area and skip Steps 5, 6, 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's run address differs from the chip's boot address) to ensure the PC Program counter is initialized to the correct start address. For .ini script file, see G32R430_DDL_SDK_Vx.x.x \Examples\Board_G32R430_Tiny\ ATAN2\ATAN2_Math\Project\MDK\itcam_ram.ini

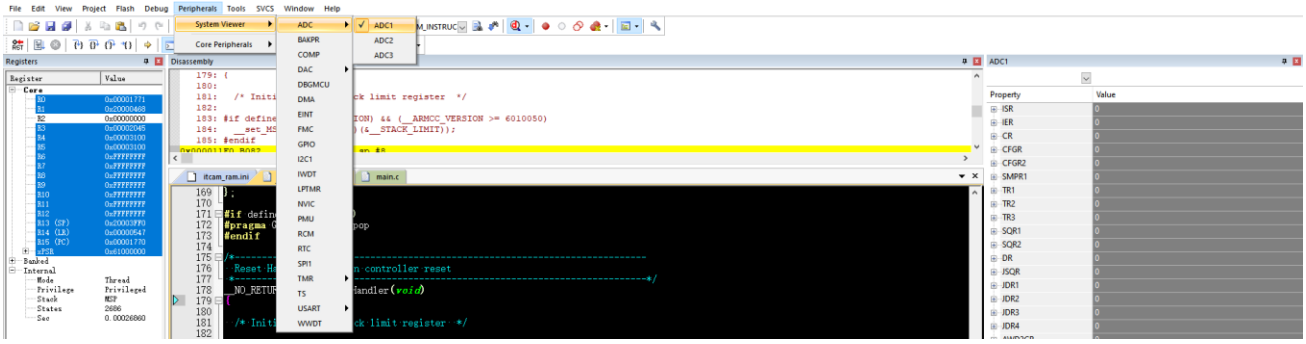
Figure 9 Configuring .ini File in MDK



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 10 Viewing Peripheral Registers in MDK Debug Interface

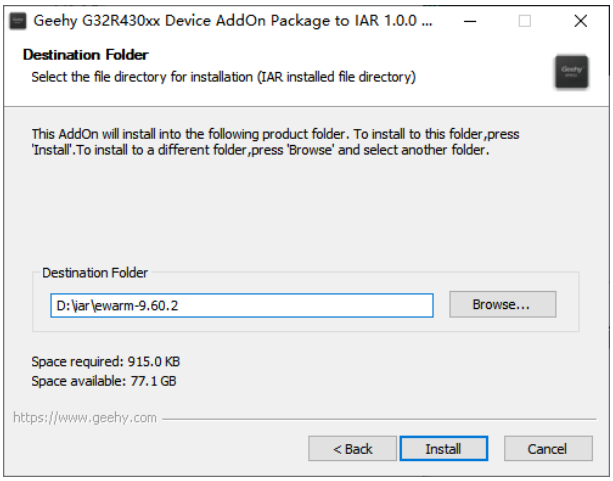


4.2 Running Examples in IAR for Arm

4.2.1 Install the Chip Support

Run the Package\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 11 G32R430xx_AddOn_vx.x.x.exe Installation Program



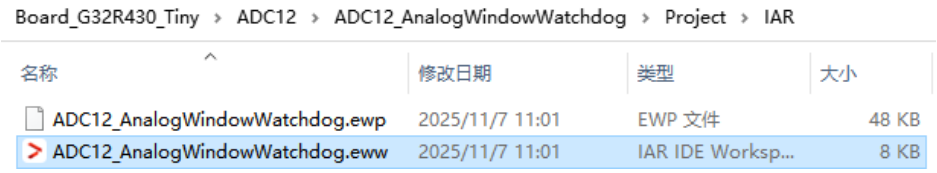
4.2.2 Use an Example Program

Open the IAR project located 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.

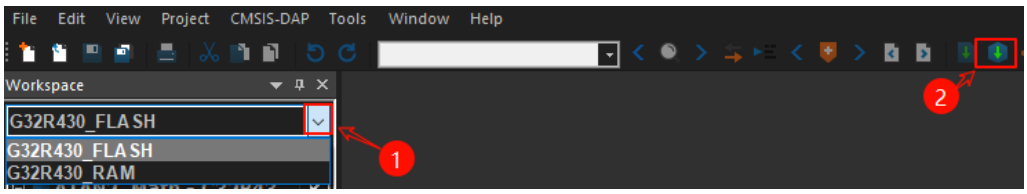
Figure 12 ADC12_AnalogWindowWatchdog IAR Project Diagram



4.2.3 Compile the Program

After opening the project, the project structure can be viewed in the IAR's Workspace. Check if the project target configuration is the desired one, then click "Make" or the corresponding compile button and wait for the project to complete compilation. If compilation is successful, the Console will display no errors or warnings.

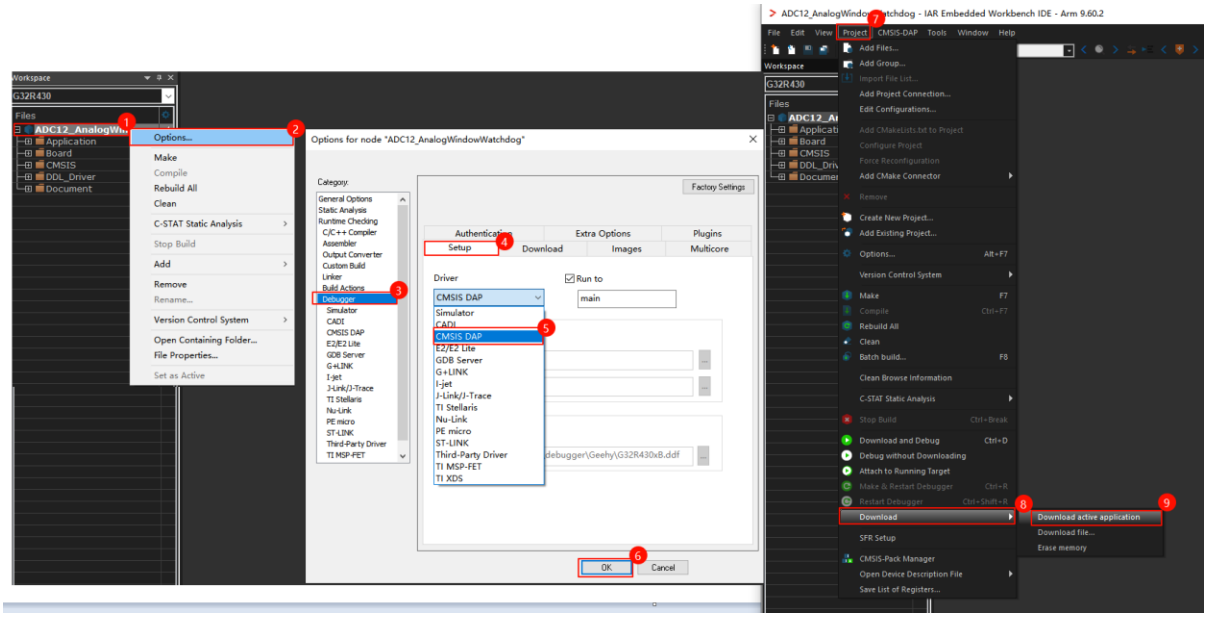
Figure 13 Multi-Target Selection and Compilation in IAR



4.2.4 Download the Program

In the debugger option, select to use "CMSIS-DAP ARMv8-M Debugger". Then, click the "Project" button in the status bar and select "Download active application" under "Download" to download the program.

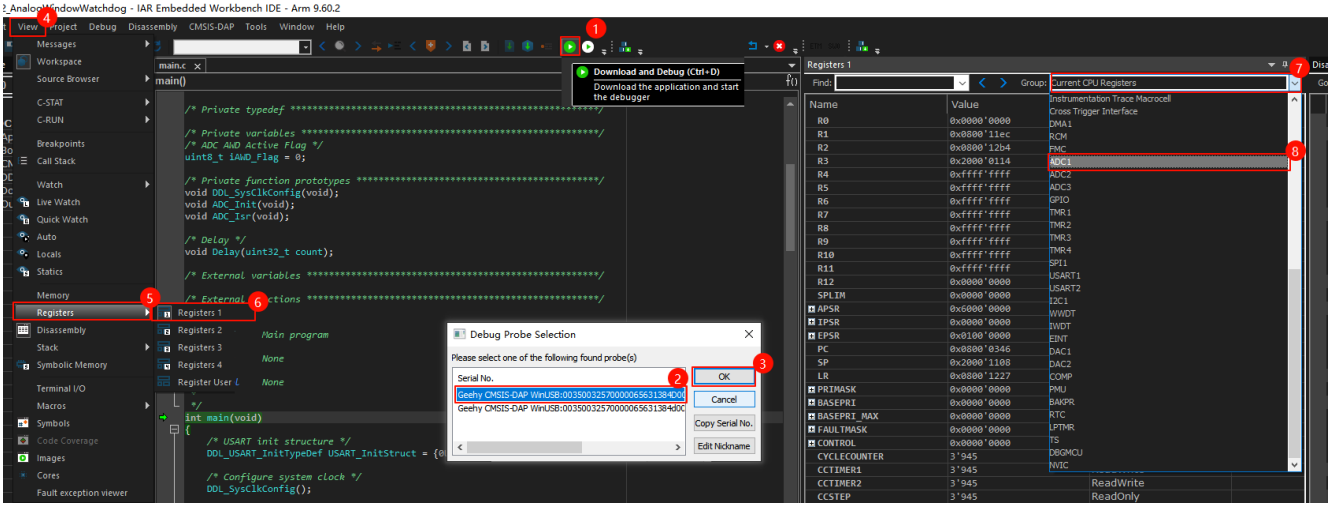
Figure 14 Configuring Emulator and Downloading Program in IAR



4.2.5 Debug the Program

Click "Download and Debug" to enter the emulation and debugging environment. At the first time, an emulator selection interface will pop up; then select the first one. For more operations, please refer to Figure 15.

Figure 15 Viewing Peripheral Registers in IAR Debug Interface



The followings can be viewed in the debug windows:

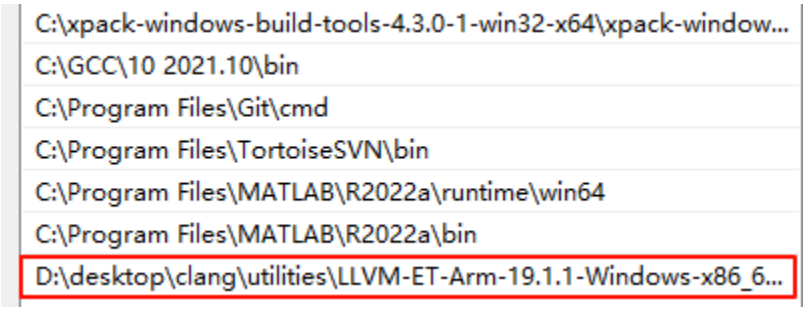
- General-purpose registers (e.g., R0, R1 … R12, SP, LR, etc.).
- Core and peripheral registers (used to view peripheral initialization and status of the chip).

4.3 Running Examples in Eclipse

4.3.1 Toolchain Installation

1. Eclipse version requirements: Ensure you are using Eclipse 4.35 or a later version of the IDE.
2. LLVM_For_ARM_Toolchain
   a) Download LLVM_For_ARM_Toolchain compiler from the LLVM_For_ARM_Toolchain repository: https://github.com/ARM-software/LLVM-embedded-toolchain-for-Arm. Download LLVM-ET-Arm-19.1.1-Windows-x86_64 compiler.
   b) Add environment variables.
      - Decompress the downloaded package (example decompression path: D:\desktop\clang\utilities\LLVM-ET-Arm-19.1.1-Windows-x86_64)
      - Add the bin folder in the folder to the system environment variables.

Figure 16 Add system environment variables

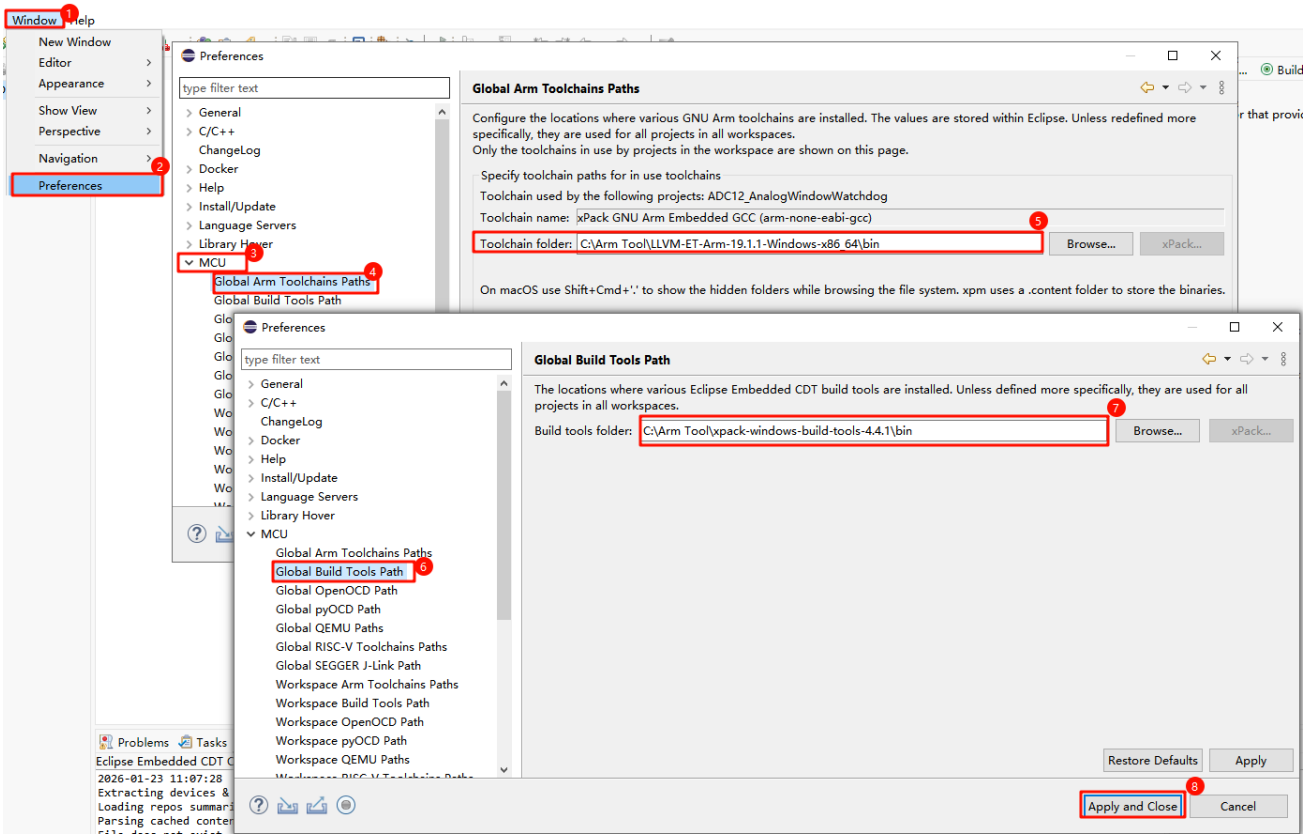


3. GDB Service. It is recommended to use arm-none-eabi-gdb.exe provided by Arm. The arm-none-eabi-gdb.exe version used in the example is 14.2.

4. Make tool. It is recommended to use xpack-windows-build-tools, and the example uses version 4.4.1.

The above toolchain can be configured into the global Path of Eclipse. Refer to the diagram for illustration.

Figure 17 Eclipse Adds MCU Global Toolchain Configuration



4.3.2 pyOCD Adaptation

In order to facilitate users to perform program download and simulation operations on G32R430

in an open source environment, the G32R430 series MCU needs to support pyocd.

Currently, the Eclipse example in the SDK uses pyOCD 0.36 （https://github.com/pyocd/pyOCD/releases/tag/v0.36.0）during the download process. This version does not support M52 core chips and G32R430 chips. It is necessary to modify its source code to complete support.

1. The main file to be modified to add M52 core support in pyOCD is:

   a) pyocd\coresight\component_ids.py, add the following under the class CmpInfo(NamedTuple):

```
# Designer        |Component Class |Part   |Type |Archid              |Name          |Product      |Factory

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x31, 0x0a31) : CmpInfo('MTB',         'Star-MC2', None                ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x43, 0x1a01) : CmpInfo('ITM',         'Star-MC2', ITM.factory         ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x1a02) : CmpInfo('DWT',         'Star-MC2', DWTv2.factory       ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x1a03) : CmpInfo('BPU',         'Star-MC2', FPB.factory         ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x14, 0x1a14) : CmpInfo('CTI',         'Star-MC2', None                ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x2a04) : CmpInfo('SCS',         'Star-MC2', CortexM_v8M.factory ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x13, 0x4a13) : CmpInfo('ETM',         'Star-MC2', None                ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0x132, 0x11, 0)      : CmpInfo('TPIU',        'Star-MC2', TPIU.factory        ),
```

Figure 18 Modified component_ids.py

```
## Map from (designer, class, part, devtype, archid) to component name, product name, and factory.
COMPONENT_MAP: Dict[Tuple[int, int, Optional[int], Optional[int], int], CmpInfo] = {
  # Archid-only entries
     # Designer       |Component Class |Part   |Type |Archid              |Name          |Product      |Factory
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x31, 0x0a31) : CmpInfo('MTB',         'Star-MC2', None                ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x43, 0x1a01) : CmpInfo('ITM',         'Star-MC2', ITM.factory         ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x1a02) : CmpInfo('DWT',         'Star-MC2', DWTv2.factory       ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x1a03) : CmpInfo('BPU',         'Star-MC2', FPB.factory         ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x14, 0x1a14) : CmpInfo('CTI',         'Star-MC2', None                ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x2a04) : CmpInfo('SCS',         'Star-MC2', CortexM_v8M.factory ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x13, 0x4a13) : CmpInfo('ETM',         'Star-MC2', None                ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0x132, 0x11, 0)      : CmpInfo('TPIU',        'Star-MC2', TPIU.factory        ),
  # Designer|Component Class |Part   |Type |Archid              |Name          |Product      |Factory
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x0a00) : CmpInfo('RASv1',         None,       None                ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x1a01) : CmpInfo('ITMv2',         None,       ITM.factory         ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x1a02) : CmpInfo('DWTv2',         None,       DWTv2.factory       ),
```

   b) pyocd\coresight\core_ids.py：

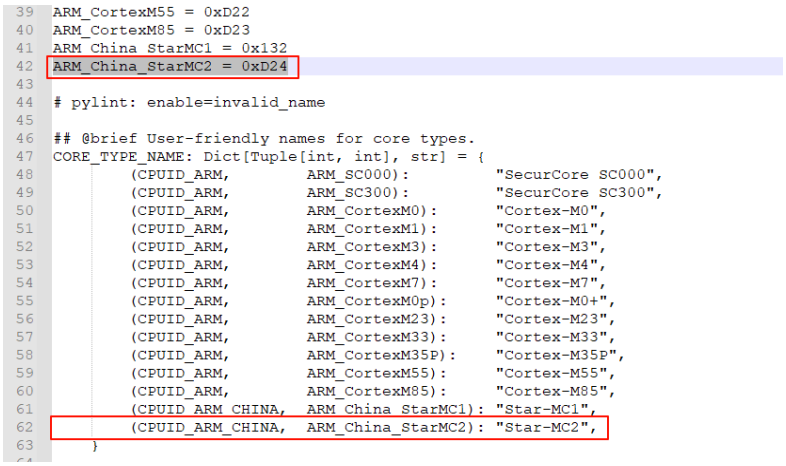
      1) Add core ID under # CPUID PARTNO values

```
ARM_China_StarMC2 = 0xD24
```

      2) Add core name under CORE_TYPE_NAME: Dict[Tuple[int, int], str]

(CPUID_ARM_CHINA, ARM_China_StarMC2): "Star-MC2",

Figure 19 Modified core_ids.py



```
ARM_CortexM55 = 0xD22
ARM_CortexM85 = 0xD23
ARM_China_StarMC1 = 0x132
ARM_China_StarMC2 = 0xD24

# pylint: enable=invalid_name

## @brief User-friendly names for core types.
CORE_TYPE_NAME: Dict[Tuple[int, int], str] = {
        (CPUID_ARM,         ARM_SC000):         "SecurCore SC000",
        (CPUID_ARM,         ARM_SC300):         "SecurCore SC300",
        (CPUID_ARM,         ARM_CortexM0):      "Cortex-M0",
        (CPUID_ARM,         ARM_CortexM1):      "Cortex-M1",
        (CPUID_ARM,         ARM_CortexM3):      "Cortex-M3",
        (CPUID_ARM,         ARM_CortexM4):      "Cortex-M4",
        (CPUID_ARM,         ARM_CortexM7):      "Cortex-M7",
        (CPUID_ARM,         ARM_CortexM0p):     "Cortex-M0+",
        (CPUID_ARM,         ARM_CortexM23):     "Cortex-M23",
        (CPUID_ARM,         ARM_CortexM33):     "Cortex-M33",
        (CPUID_ARM,         ARM_CortexM35P):    "Cortex-M35P",
        (CPUID_ARM,         ARM_CortexM55):     "Cortex-M55",
        (CPUID_ARM,         ARM_CortexM85):     "Cortex-M85",
        (CPUID_ARM_CHINA,   ARM_China_StarMC1): "Star-MC1",
        (CPUID_ARM_CHINA,   ARM_China_StarMC2): "Star-MC2",
    }
```

2. Add G32R430 chip support.

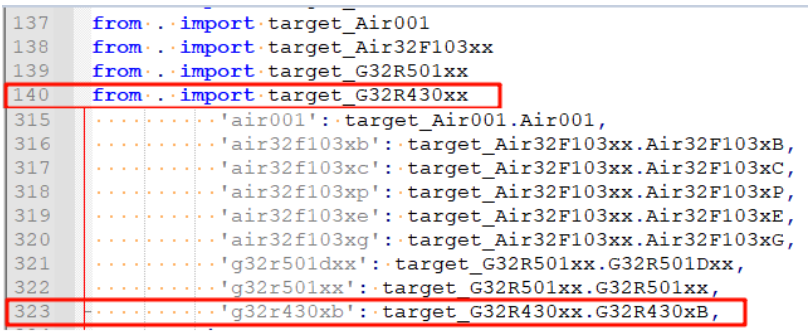
   a) Add g32r430 download algorithm support file: target_G32R430xx.py in pyocd\target\builtin. This file has been added to SDK/Package/pyOCD/target_G32R430xx.py.

   b) Add g32r430 support, and add the following to pyocd\target\builtin__init__.py:

from . import target_G32R430xx

'g32r430xb': target_G32R430xx.G32R430xB,

Figure 20 Modified __init__.py



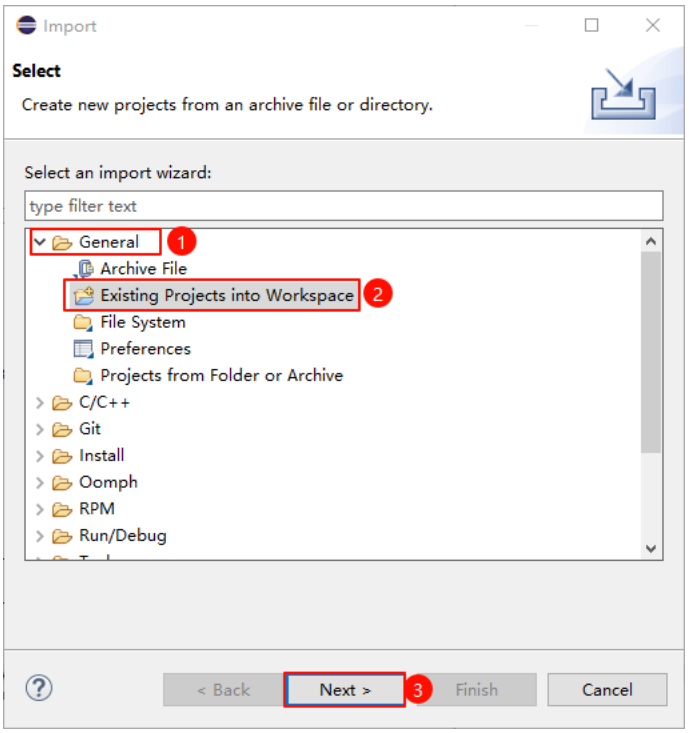
```
from . import target_Air001
from . import target_Air32F103xx
from . import target_G32R501xx
from . import target_G32R430xx
        'air001': target_Air001.Air001,
        'air32f103xb': target_Air32F103xx.Air32F103xB,
        'air32f103xc': target_Air32F103xx.Air32F103xC,
        'air32f103xp': target_Air32F103xx.Air32F103xP,
        'air32f103xe': target_Air32F103xx.Air32F103xE,
        'air32f103xg': target_Air32F103xx.Air32F103xG,
        'g32r501dxx': target_G32R501xx.G32R501Dxx,
        'g32r501xx': target_G32R501xx.G32R501xx,
        'g32r430xb': target_G32R430xx.G32R430xB,
```

4.3.3 Example Usage

1. Start Eclipse 4.35.

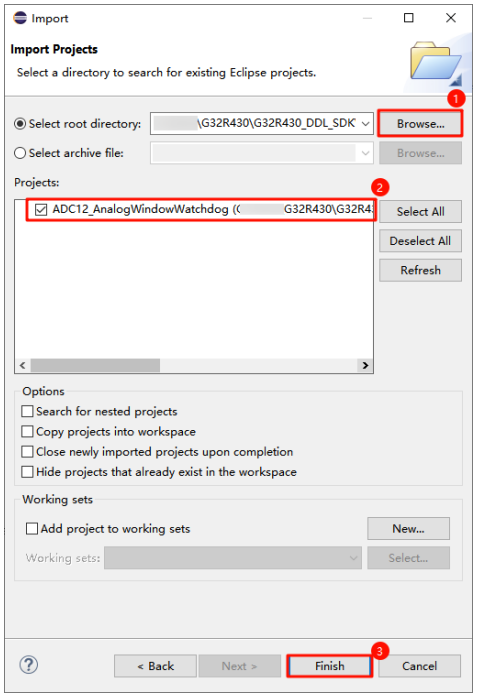
2. Click "File" -> "import ..." -> "General" -> "Existing Project into Workspace" in the menu bar.

Figure 21 Import project 1



3. 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 22 Import project 2



Note: Please complete the LLVM compilation configuration in section 4.3.1 before the above steps.

4.3.4 Compile the Program

After opening the project, you can view the project structure in the Eclipse Project Explorer. Check whether the project target (Target) configuration is what you need. Click the "Build" compilation button and wait for the project compilation to complete. If the compilation is normal, the Console will display no errors and warnings.

Figure 23 Compiling Example Operation under Eclipse

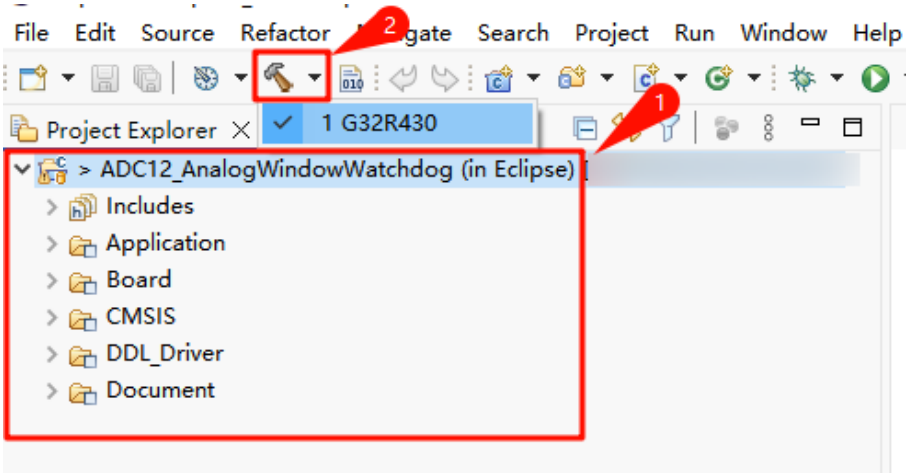
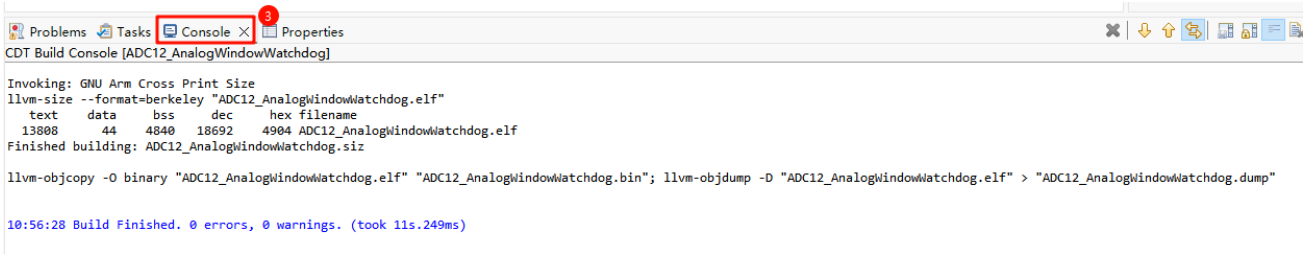


Figure 24 Successfully Compiled Example under Eclipse



4.3.5 Download and Debug the Program

Downloading and debugging programs under Eclipse requires configuring the GDB service. Please configure the debugging tab according to the following steps.

Note: The following steps need to be performed after completing the python, pyOCD and pyOCD's G32R430 chip support. Refer to chapters 4.3.2 and chapter 5 for environment deployment.

1. Add pyOCD Debugging configuration

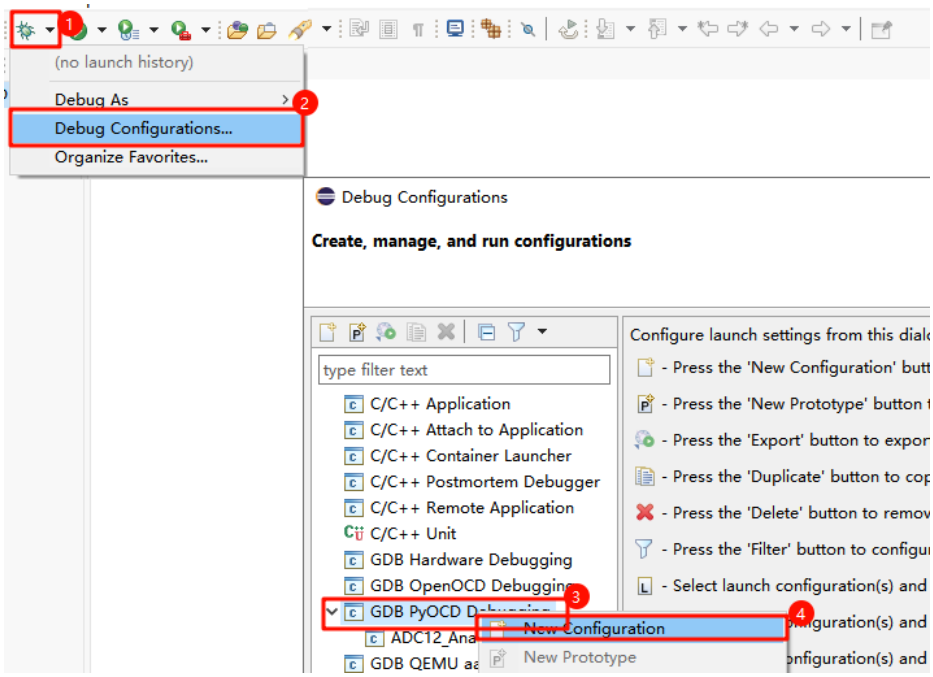
   1) Left-click on the Debug icon to display the Debug configuration.

   2) Select "Debug Configurations..." to display.

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

   4) Select "New Configuration" to configure the simulation.

Figure 25 Add pyOCD Debugging Configuration



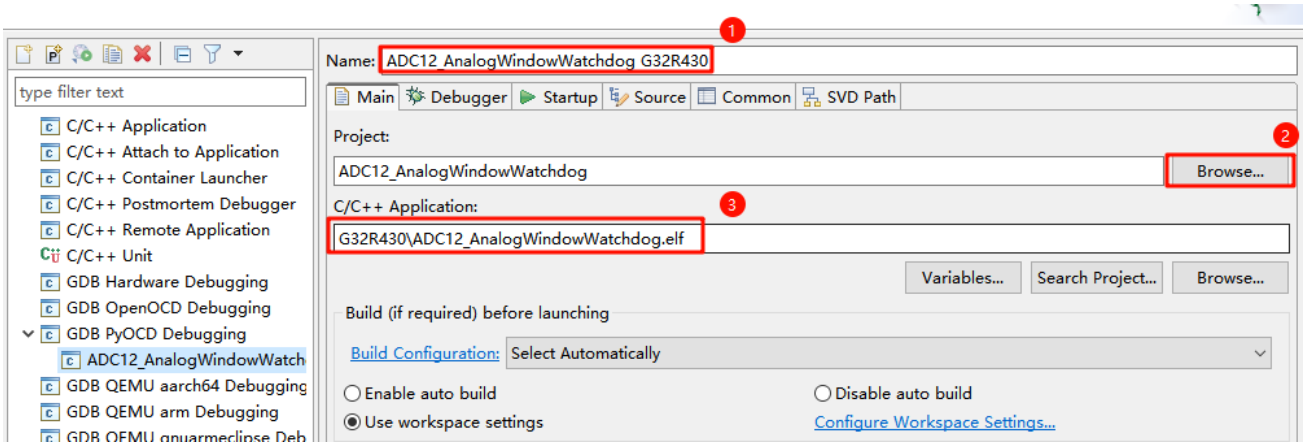
2. Configure the Main tab

   1) Name the current simulation configuration at the top.

   2) Select "Browse..." and select the project corresponding to the current simulation configuration.

   3) Select the corresponding simulation elf file, for example: G32R430\ADC12_AnalogWindowWatchdog.elf. The example uses a relative path relative to the project file, which supports absolute paths.

Figure 26 Configure Main



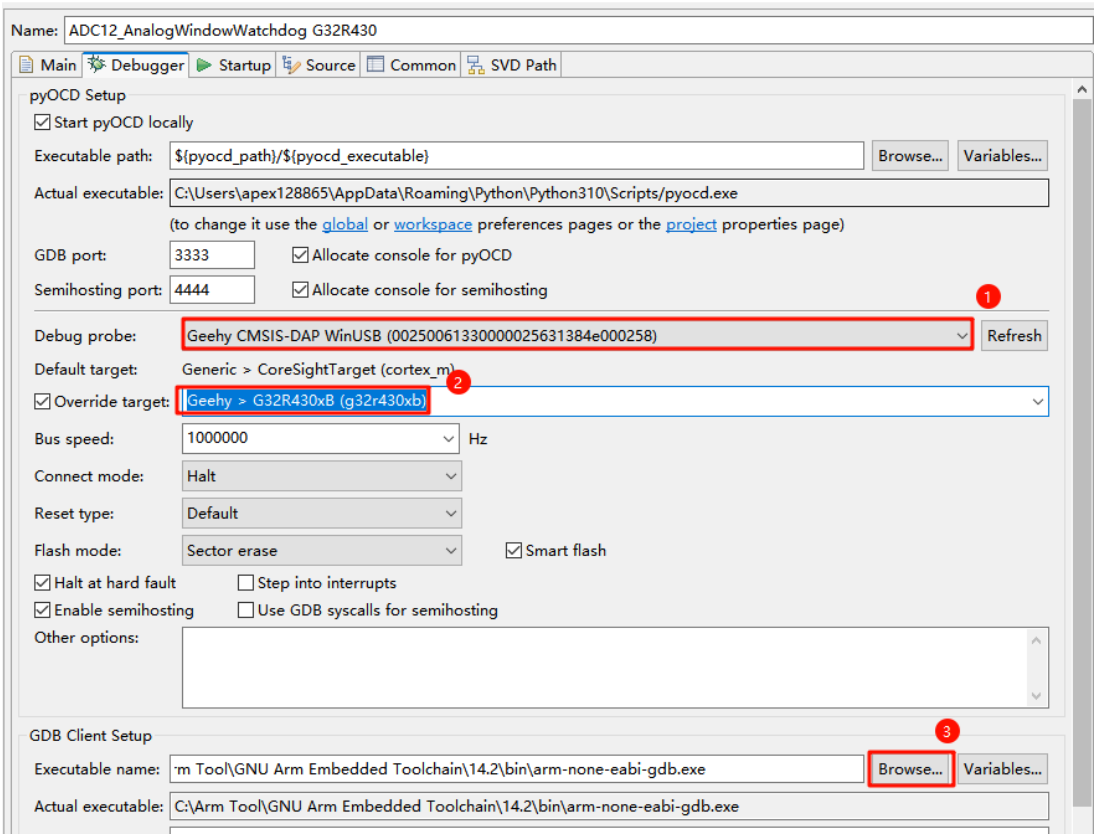
3. Configure Debugger tab

1) Select the Geehy-Link emulator used. The characters in parentheses are the emulator serial number.

2) Select the corresponding chip, for example: Geehy >G32R430xB(g32r430xb)

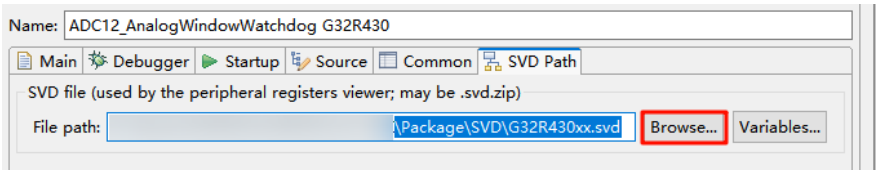
3) Select the GDB service. It is recommended to use arm-gnu-toolchain-14.2.rel1\bin\arm-none-eabi-gdb.exe provided by Arm.

Figure 27 Configure Debugger



4. Configure the SVD file tab. The SVD file provides users with a convenient way to view the peripheral register contents of the MCU. For G32R430, please select Package\SVD\G32R430xx.svd in its SDK.

Figure 28 Configure SVD File Tab

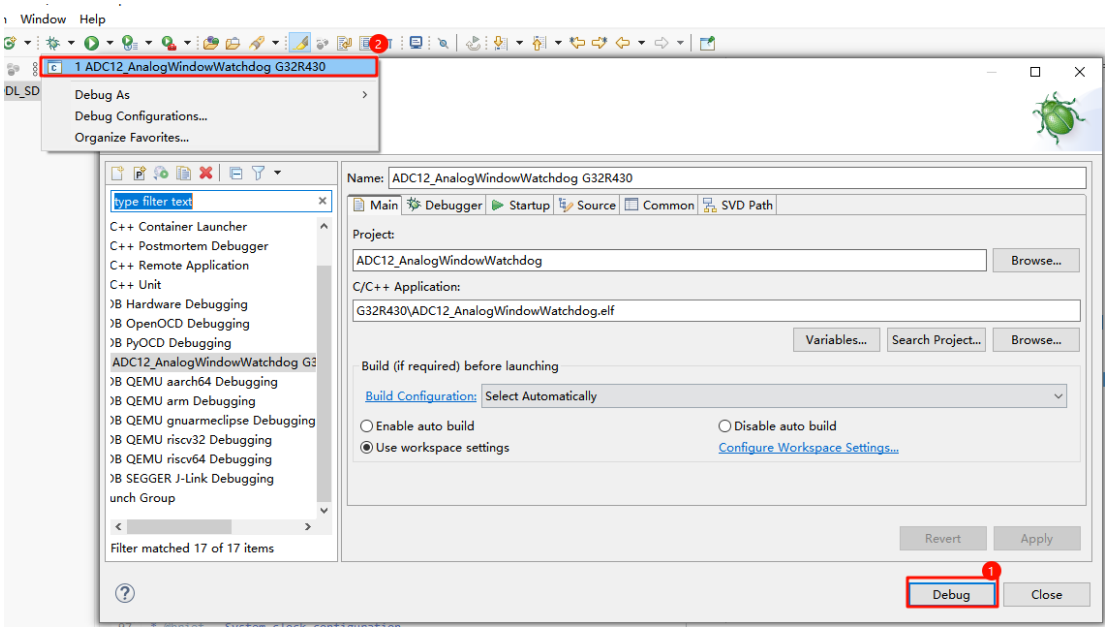


5. Finally, click the "Apply" button in the lower right corner of the tab to apply all configuration items.

6. Launch download or debugging. The first time, you need to click the "Debug" button in the

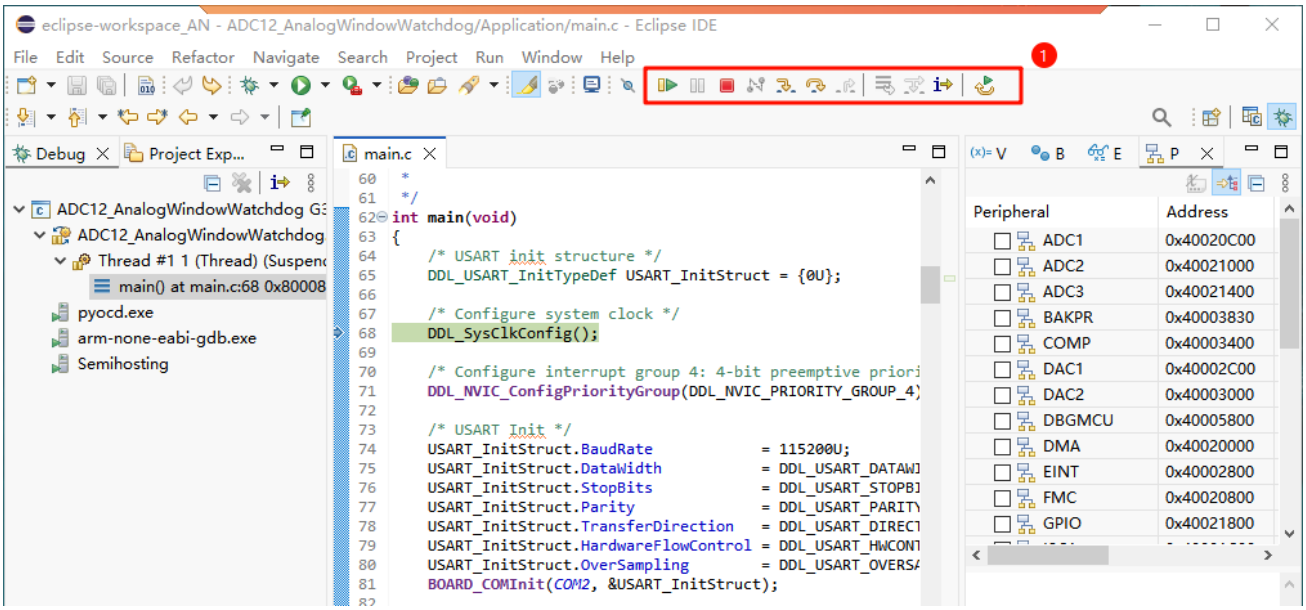
lower right corner of the tab to simulate. Subsequently, it can be done through the Debugger button in the toolbar.

Figure 29 Launch Download or Debugging



7. Debugging successful. Please use the debug button in the toolbar to control the program.

Figure 30 Debugging Successfully



5 pyOCD Installation

5.1 Windows

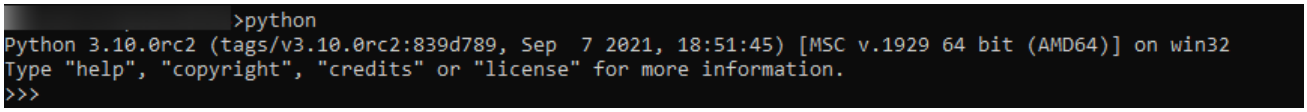
5.1.1 Install Python

pyOCD support requires a Python environment. Please go to the official Python website (https://www.python.org) to download the latest Python installation package.

Note: After installation, ensure that Python is added to the system's PATH environment variable so that it can be used in the command line.

Verification: Use the Win+R keys, enter CMD, and enter "python" in the command line window, then press Enter. The installed Python version number and other information will be displayed, and you will enter the Python interactive command line (REPL). To exit, enter exit() or quit(), and then press Enter.

Figure 31 Python Installation Verification



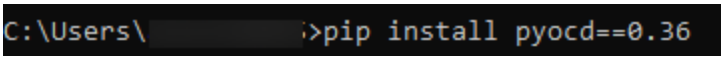
5.1.2 Install pyOCD

pyOCD is a Python component package that supports online installation (online installation is recommended because there are different dependency packages for online installation).

The installation method is as follows:

1. Use the pip command "pip install pyocd==0.36" to install version 0.36 of pyOCD:

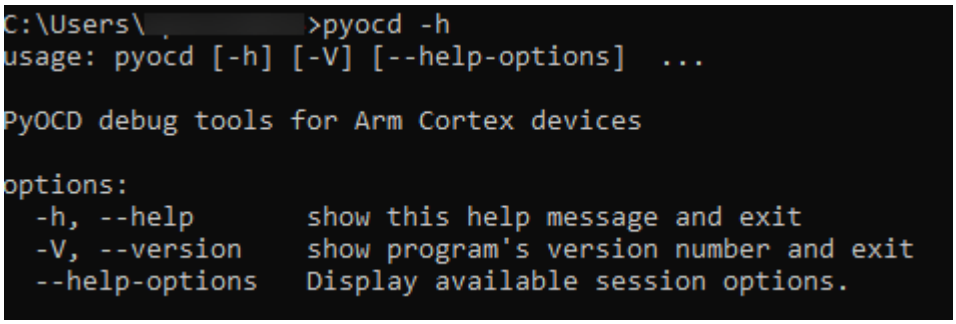
Figure 32 Install pyOCD



Note: After installation, please add the installation path of pyOCD to the system's PATH environment variable for command-line usage. For example: C:\Users\Geehy\AppData\Local\Programs\Python\Python313\Scripts

2. To verify the installation result, use Win+R keys, enter CMD, and enter the following in the command line window: pyocd -h, which will display command help.

Figure 33 pyOCD Installation Verification



5.2 Ubuntu

5.2.1 Install Python

1. 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
```

2. 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 Python3 environments are externally managed, making it impossible to directly use pip to install packages in the global environment. To avoid this issue, you can directly use Python's built-in venv module to create a virtual environment.

1. Use the following command to install the python3-venv package:

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

2. Use the following command to create a virtual environment (assuming you name the virtual environment venv):

```
python3 -m venv venv
```

3. Use the following command to activate the virtual environment:

- Activate the virtual environment to install packages within it.

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

- If you want to exit the virtual environment later, you can use the following command:

```
deactivate
```

5.2.3 Install pyOCD

1. In the activated virtual environment, use pip to install pyOCD:

```
pip install pyocd==0.36
```

2. Verify the installation result

```
pyocd --version
```

3. Query the pyOCD installation location (for subsequent modification of the pyOCD source code)

```
pip show pyocd
```

5.2.4 pyOCD's USB Permissions

If pyOCD cannot access the debugger under a normal user, consider adding appropriate permissions to the current user. Use udev rules to ensure that you can access USB devices without using sudo. The steps are as follows:

1. Create a new rules file. Execute the following command in the terminal to create a new udev rules file (for example, name it 99-pyocd.rules).

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

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

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

In the nano editor, press Ctrl + O to save the file, and then press Enter to confirm. Then press Ctrl + X to exit the editor.

3. After saving the file, reload the udev rules.

```
sudo udevadm control --reload-rules

sudo udevadm trigger
```

4. Verify that pyOCD can correctly recognize Geehy-Link.

```
pyocd list
```

Figure 34 Emulator Serial Number Connected to Ubuntu

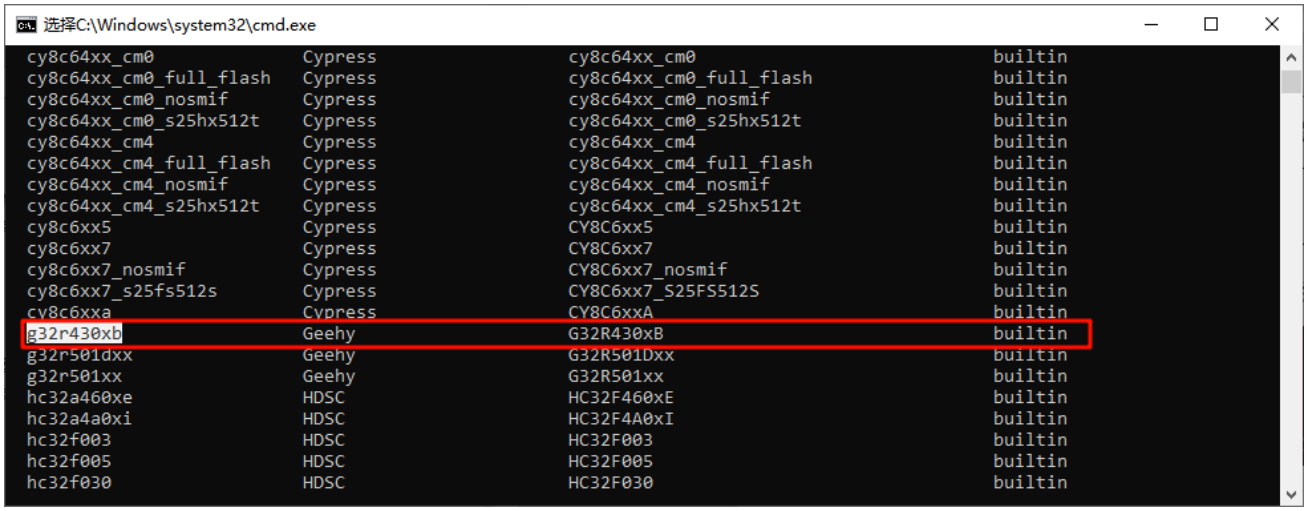
```
(venv) kai@Ubuntu:~$ pyocd list
  #   Probe/Board              Unique ID                                Target
------------------------------------------------------------------------------
  0   Geehy CMSIS-DAP WinUSB   00350043500000144e544859000258   n/a
```

5.3 Replace Modified Item Content

To support G32R430, refer to section 4.3.2 to modify the downloaded pyOCD content.

To verify whether G32R430 support has been successfully added, use Win+R, enter CMD, and enter the following in the command line window: pyocd list --targets. This will display all chips. Check whether "g32r430xb" is in the supported chips.

Figure 35 List of Supported Chips



5.4 Command Line Usage

pyOCD supports using the CMD command line. Refer to the following steps for usage (confirm that pyOCD has been added to PATH before use).

1. Connect the board or emulator to the chip and connect to the PC.
2. Start cmd in the working directory and enter: pyocd commander -t g32r430xb, then you can enter relevant instructions in the command line window to perform the corresponding

operations.

Figure 36 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

Currently, using Eclipse+pyocd has version requirements for Eclipse. It is recommended to use Eclipse version 202503.

In addition, it is recommended to configure the pyOCD path globally in Eclipse (it has been found that eclipse may have exceptions in getting PATH on some computers). The steps are as follows:

1. Right-click on "Windows" in the menu bar to display all configurations.
2. Select "Preference" in the displayed configuration.
3. In the new window, open the sub-options under the "MCU" option.
4. Select the sub-option "Global pyOCD Path".
5. Select the corresponding pyocd.exe directory on the right.
6. Finally, click "Apply and Close".

Figure 37 Global pyOCD Path Steps 1-2

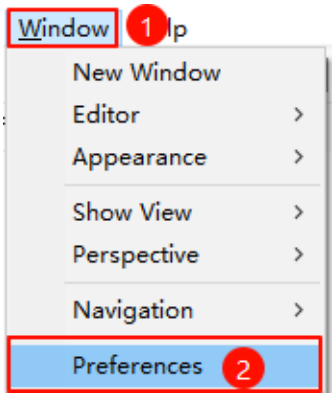
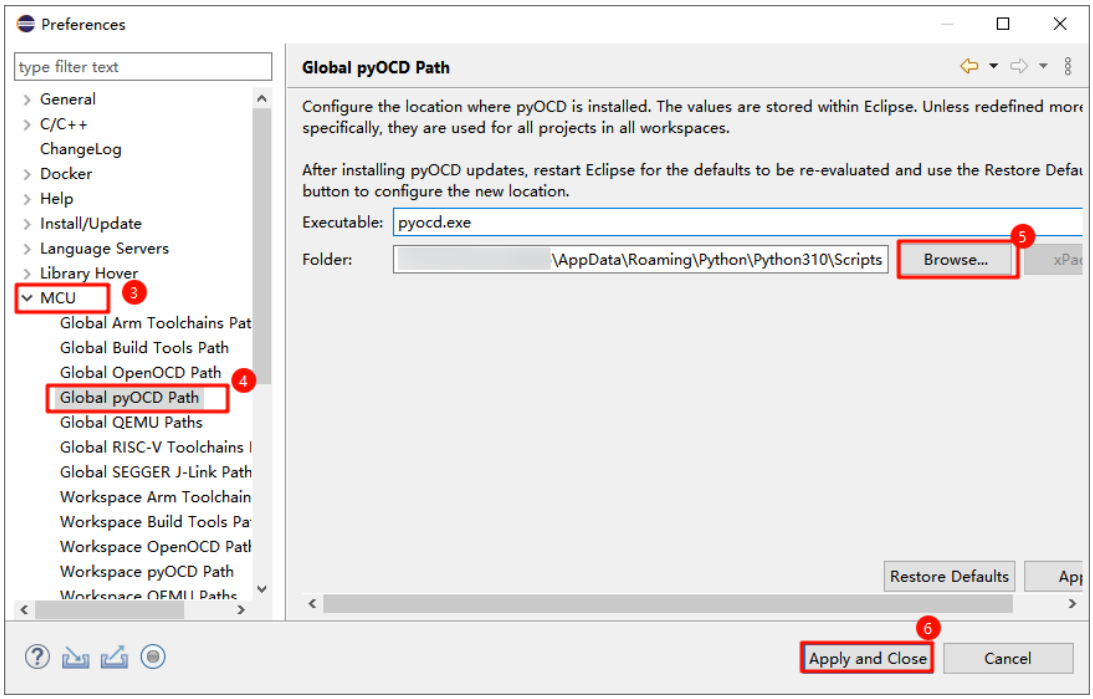


Figure 38 Global pyOCD Path Steps 3-6



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

MDK: Libraries\Device\Geehy\G32R4xx\Source\arm

IAR: Libraries\Device\Geehy\G32R4xx\Source\iar

GCC：Libraries\Device\Geehy\G32R4xx\Source\gcc

- Chip memory structure (G32R430xB)

Flash: 128 KB

DTCM RAM: 16 KB

ITCM RAM: 32 KB

- Description for example memory configuration files

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.

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 field definitions in the linker scripts. Common fields and their meanings are as follows:

1. SECTION_ITCM_INSTRUCTION: ITCM instruction code section
2. SECTION_ITCM_RAMFUNC: Function section in ITCM RAM (functions executed in RAM)
3. SECTION_DTCM_DATA: DTCM data section

4. SECTION_DTCM_BSS: DTCM BSS (uninitialized global/static variables) section

5. 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

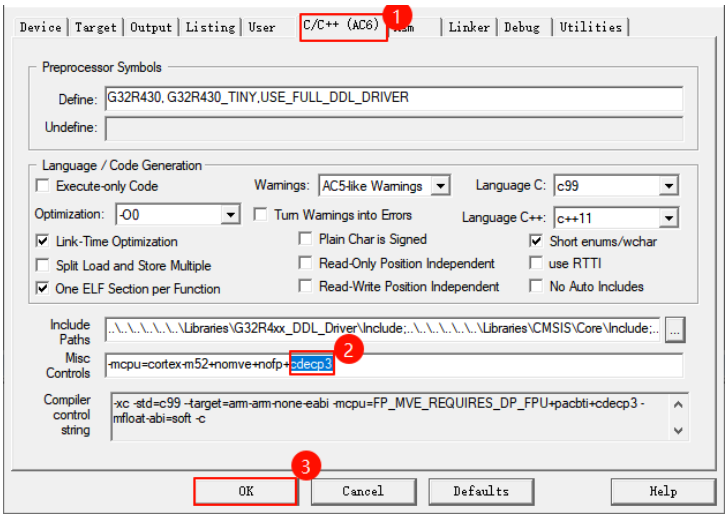
The code section for the ATAN2 function is "atan2_instruction" by default. The address of this section can be modified in the linker script (e.g., .sct or .icf files) or be mapped to the required storage space. If there are specific performance or memory requirements, the linker script can be adjusted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

7.4 Usage

1. Select the appropriate library file: Based on the compiler (MDK or IAR) and the requirements for calculation precision (high/low_resolution), include the corresponding ".lib" or ".a" file in the project's linker options.
2. Enable CDE3 support in the project:

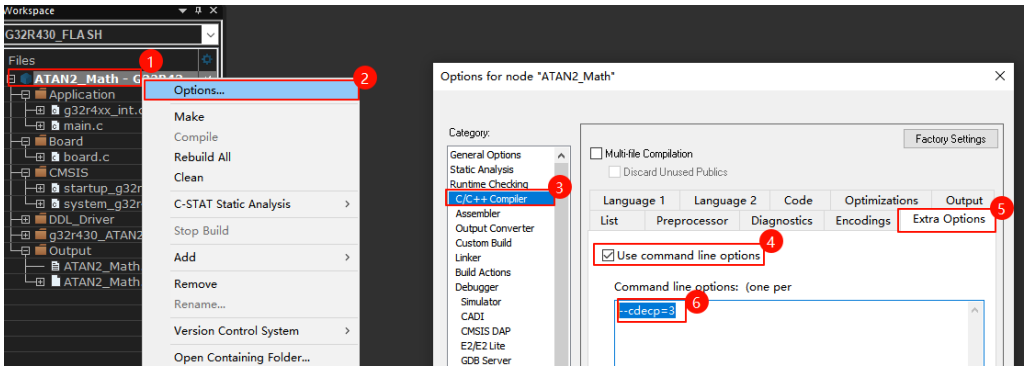
- For MDK, enable support for CDE3 in the project settings.

Figure 39 Enabling CDE3 Support for C Code in MDK



- For IAR, enable the support for corresponding extended instruction set of CDE3 in the project options' compilation settings.

Figure 40 Enabling CDE3 Support for C Code in IAR



3. Include the ATAN2 header file 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 file reference at the top of the source file that needs to call the ATAN2 function.

Note: For example programs, please refer to:
G32R430_DDL_SDK_Vx.x.x\Examples\Board_G32R430_Tiny\ATAN2\ATAN2_Math\

8 Revision History

Table 1 Document Revision History

Date	Version	Revision History
November 2025	1.0	• New
December 2025	1.1	• 2.1 updated the board picture to V1.2 and deleted the operational content required for the V1.1 version.
January 2026	1.2	• Added Section 4.3: Instructions for running example programs in the Eclipse environment. • Added Section 5: pyOCD installation instructions and how to add support for the G32R430 chip. • Section 6: Added descriptions of linker scripts for the gcc environment.

Statement

This manual is prepared and published by Zhuhai Geehy Semiconductor Co., Ltd. (hereinafter referred to as "Geehy"). The contents in this manual are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to correct and modify this manual at any time. Please read this manual carefully before using the product. Once you use the product, it means that you (hereinafter referred to as the "users") have known and accepted all the contents of this manual. Users shall use the product in accordance with relevant laws and regulations and the requirements of this manual.

1. Ownership of rights

This manual can only be used in combination with chip products and software products of corresponding models 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 manual for any reason or in any form.

The "Geehy" or "Geehy" words or graphics with "®" or "TM" in this manual 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 manual.

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

If any third party’s products, services or intellectual property are involved in this manual, Geehy shall not be deemed to authorize users to use the aforesaid third party’s products, services or intellectual property, nor shall it be deemed to provide any form of guarantee for third-party products, services, or intellectual property, including but not limited to any non-infringement guarantee for third-party intellectual property, unless otherwise agreed in sales order or sales contract of Geehy.

3. Version update

Users can obtain the latest manual of the corresponding products when ordering Geehy

products.

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

4. Information reliability

The relevant data in this manual 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 are unavoidable. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this manual. The relevant data in this manual 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 the user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Compliance requirements

Users shall abide by all applicable local laws and regulations when using this manual 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 product suppliers, Geehy, Geehy distributors and users. Users (on behalf of itself, subsidiaries and affiliated enterprises) shall agree and undertake 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

This manual is provided by Geehy on an "as is" basis. To the extent permitted by applicable laws, Geehy does not provide any form of express or implied warranty, including without limitation the warranty of product merchantability and applicability of specific purposes.

Geehy products are not designed, authorized, or guaranteed to be suitable for use as critical components in military, life support, pollution control, or hazardous substance management systems, nor are they designed, authorized, or guaranteed to be suitable for applications that may cause injury, death, property, or environmental damage in case of product failure or malfunction.

If the product is not labeled as "Automotive grade", it means it is not suitable for automotive applications. If the user's application of the product is beyond the specifications, application fields, and standards provided by Geehy, Geehy will assume no responsibility.

Users shall ensure that their application of the product complies with relevant standards, and the requirements of functional safety, information security, and environmental standards. Users are fully responsible for their selection and use of Geehy products. Geehy will bear no responsibility for any disputes arising from the subsequent design and use of Geehy products by users.

7. Limitation of liability

In any case, unless required by applicable laws or agreed in writing, Geehy and/or any third party providing this manual and the products on an "as is" basis shall not be liable for damages, including any general or special direct, indirect or collateral damages arising from the use or no use of this manual and the products (including without limitation data loss or inaccuracy, or losses suffered by users or third parties). This covers damage to personal safety, property, or environment, for which Geehy will not be responsible.

8. Scope of application

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