

应用笔记

Application Note

文档编号: **AN1166**

G32R430 DDL SDK 快速上手指南

——基于 G32R430TinyBoard

版本: **V1.2**

1 引言

本应用笔记旨在帮助您快速了解和应用 G32R430 DDL SDK, 以及基于 G32R430TinyBoard 的开发流程。通过阅读本笔记, 您将熟悉常用硬件资源与 SDK 目录结构, 并掌握如何在 MDK、IAR 与 Eclipse 环境下运行示例程序, 帮助您快速完成对 G32R430 系列芯片的初步评估与应用开发。

在开始前, 请准备以下环境与工具:

1. Windows 10 及以上操作系统
2. G32R430 DDL SDK Vx.x.x
3. Keil MDK 5.40 及以上版本
4. IAR for ARM 9.60.2 及以上版本
5. Eclipse 4.35 及以上版本
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 数据线

目录

1	引言	1
2	G32R430TinyBoard 基本情况.....	3
2.1	开发板资源介绍	3
2.2	注意事项.....	4
3	SDK 目录结构介绍	5
4	如何运行例程.....	6
4.1	Keil MDK 下运行例程	6
4.2	IAR for Arm 下运行例程	9
4.3	Eclipse 下运行例程	11
5	pyOCD 安装.....	20
5.1	Windows.....	20
5.2	Ubuntu.....	21
5.3	替换修改项内容	23
5.4	命令行使用	23
5.5	集成至 Eclipse	24
6	关于链接脚本.....	26
6.1	链接脚本基本情况	26
6.2	字段说明	26
6.3	如何指定变量/函数存放区域.....	27
7	ATAN2 Libraries.....	28
7.1	ATAN2 库文件结构	28
7.2	函数原型与说明	28
7.3	函数存储与执行位置	28
7.4	使用方法.....	29
8	版本历史	30

2 G32R430TinyBoard 基本情况

本节内容将对 G32R430TinyBoard 基本情况进行说明, 用户可以快速了解并正确使用 G32R430TinyBoard 的板载功能和资源, 为后续在 MDK、IAR 与 Eclipse 环境中的开发实践做好准备。

2.1 开发板资源介绍

G32R430TinyBoard 是一款基于 G32R430 系列 MCU 的开发板, 板载资源丰富, 可满足初学者到有一定开发经验的用户快速评估和开发的需求。以下为本板常用的资源配置及接口说明:

图 1 G32R430TinyBoard



- 2×LED: LED1 (PA1)、LED2 (PA2)
- 2×按键: 用户 Key1 (PA5)、复位按键 Reset
- 1×USART: USART2_TX (PD0) / USART2_RX (PC12)
- 1×I²C EEPROM: SCL1 (PD5) / SDA1 (PD9)、ZD24C64A-XGMT 芯片
- 1×RS-485 接口: USART1_TX (PB9) / USART1_RX (PB6) / USART1_DE (PB11)
- 1×RS-422 接口: CLK (PC9) / MISO (PD2)
- 1×板载 GEEHY-LINK 仿真器:
 - 可通过板载仿真器的 TX、RX 与 G32R430 芯片进行串口通信

- 支持下载与调试

2.2 注意事项

1. 如果需要使用 J12 接口的 A1、A2、A5 等端口，则需要将 J10 上的跳线帽从默认的 LED1、LED2、KEY 位置分别转移到 A1、A2、A5 上，确保实际可用引脚资源与目标功能对应。

图 2 从默认 LED1、LED2、KEY 资源到 A1、A2、A5

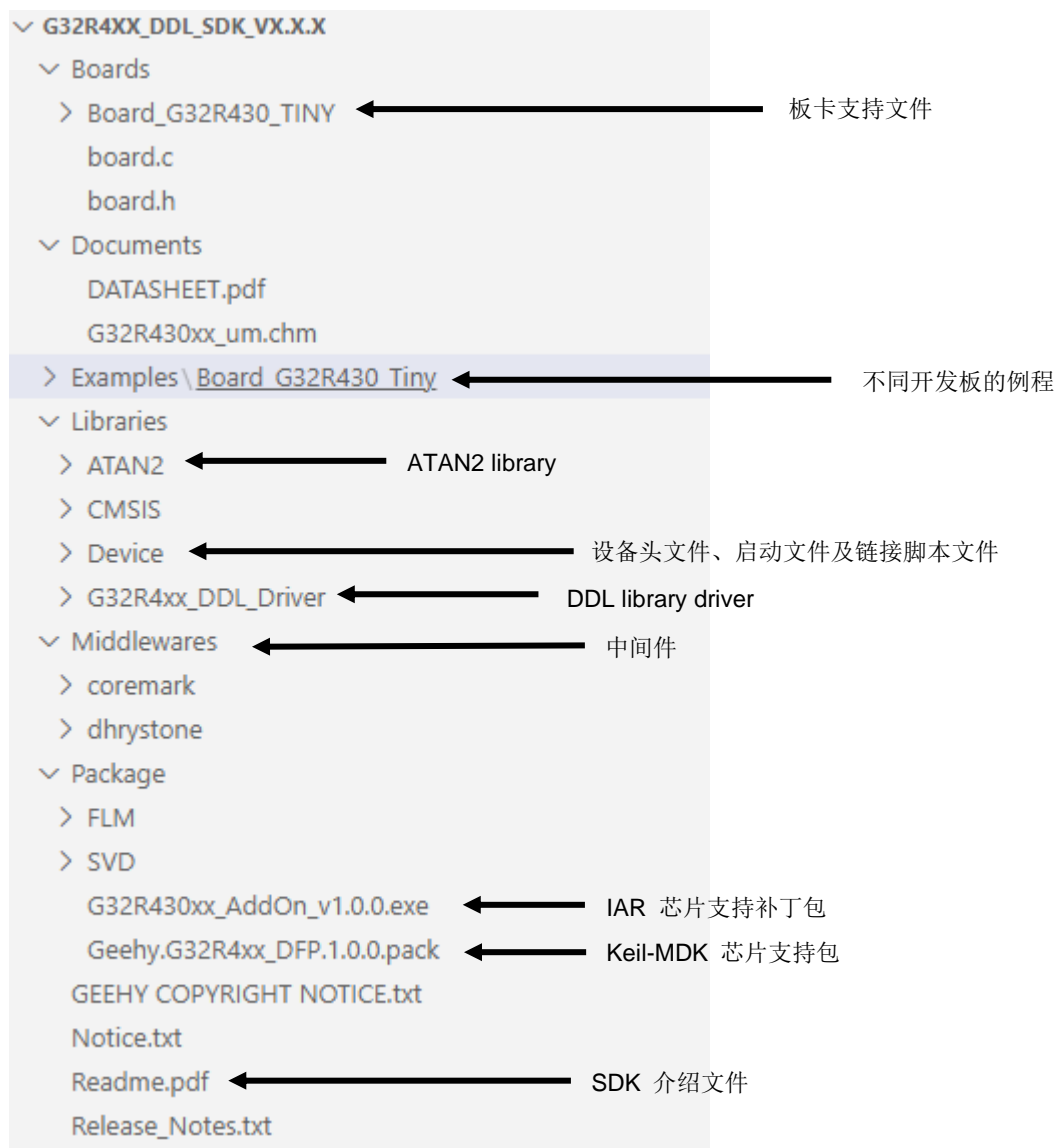


2. 如果需要连接其他外部仿真器，请先断开板载 GEEHY-LINK 仿真器与板卡的电气连接（例如将仿真器与板卡物理掰断）。

3 SDK 目录结构介绍

G32R430 DDL SDK 提供了从驱动、核心库到示例工程等较为完善的开发支持，方便用户快速上手和二次开发。其大致目录结构如图 3 所示。

图 3 G32R430 DDL SDK 目录结构介绍



注：关于 SDK 的更多详细内容请审阅 SDK 根目录下的 Readme.pdf 文件。

4 如何运行例程

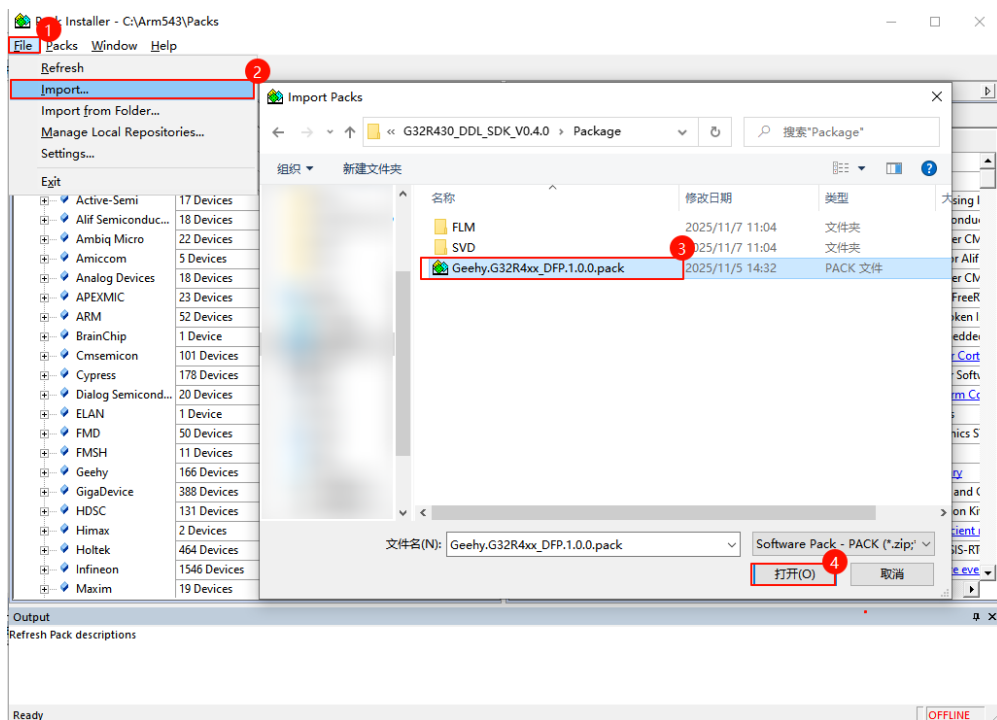
为了让用户快速上手并验证板卡及 SDK 功能, SDK 提供了在 MDK、IAR 与 Eclipse 环境下的示例工程。以下将分别介绍在不同的开发环境中需要进行的准备工作, 以及如何编译、下载、调试示例工程。

4.1 Keil MDK 下运行例程

4.1.1 安装芯片支持

使用 MDK 配置界面中的“PackInstaller.exe”通过导入 Pack 的方式完成安装, 避免双击安装可能导致的卡顿或异常。

图 4 使用 PackInstaller 安装 Geehy.G32R4xx_DFP.x.x.x.pack



PackInstaller.exe 打开方式:

- 方法 1: 在 MDK 状态栏, 点击“Pack Installer”图标。
- 方法 2: 在 MDK 安装目录运行 PackInstaller.exe, 例如:
C:\Users\Geehy\AppData\Local\Keil_v543a\UV4\PackInstaller.exe。

注 (1): MDK 版本需 \geq v5.40。建议使用 MDK v5.43a 以确保兼容性和稳定性。

注 (2): MDK v5.43a 使用双击安装 SDK 根目录下的 Package\Geehy.G32R4xx_DFP.x.x.x.pack 芯片支持包。会异常卡顿, 为 MDK 异常。


4.1.2 使用例程

打开对应板卡的 MDK 工程目录。以 ADC12 模块为例，路径如下：

Board_G32R430_Tiny\ADC12\ADC12_AnalogWindowWatchdog\Project\MDK

在该目录下找到 .uvprojx 文件，双击打开即可加载示例工程。

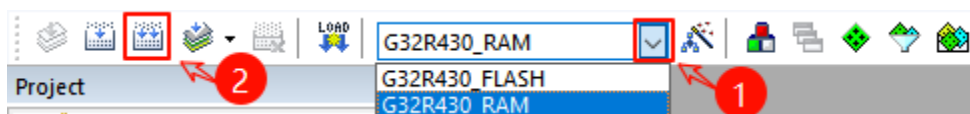
图 5 ADC12_AnalogWindowWatchdog 例程 MDK 工程示意

Board_G32R430_Tiny > ADC12 > ADC12_AnalogWindowWatchdog > Project > MDK			
名称	修改日期	类型	大小
 ADC12_AnalogWindowWatchdog.uvp...	2025/11/7 11:01	◆Vision5 Project	24 KB

4.1.3 编译例程

打开工程后，检查工程目标(Target)配置是否为自己所需要的。点击编译按钮，等待编译完成，若编译无误，MDK 将在输出框中显示“0 error, 0 warning”。

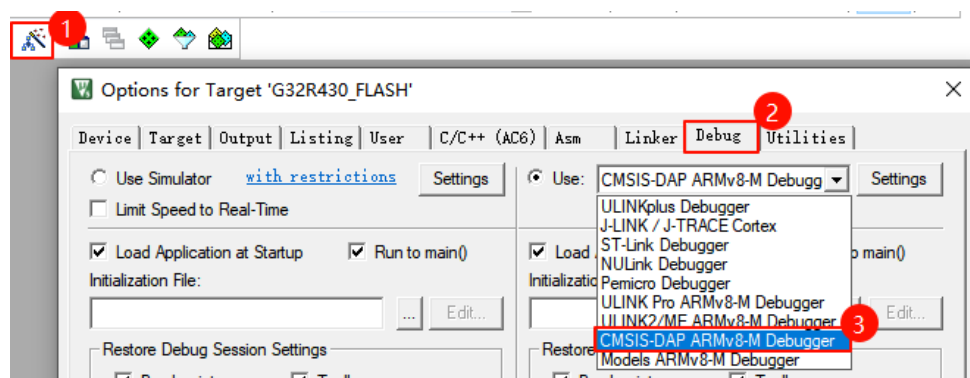
图 6 MDK 下的多 Target 选择与编译



4.1.4 下载程序

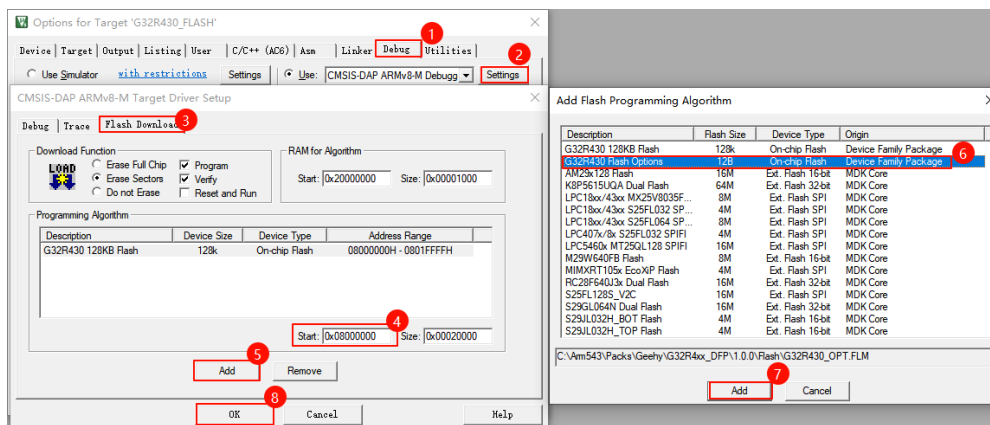
选择仿真器类型为“CMSIS-DAP ARMv8-M Debugger”。

图 7 MDK 选择仿真器



根据需要在 MDK 的“Flash Download”中配置下载相关内容。

图 8 配置 Flash Download 内容



包括:

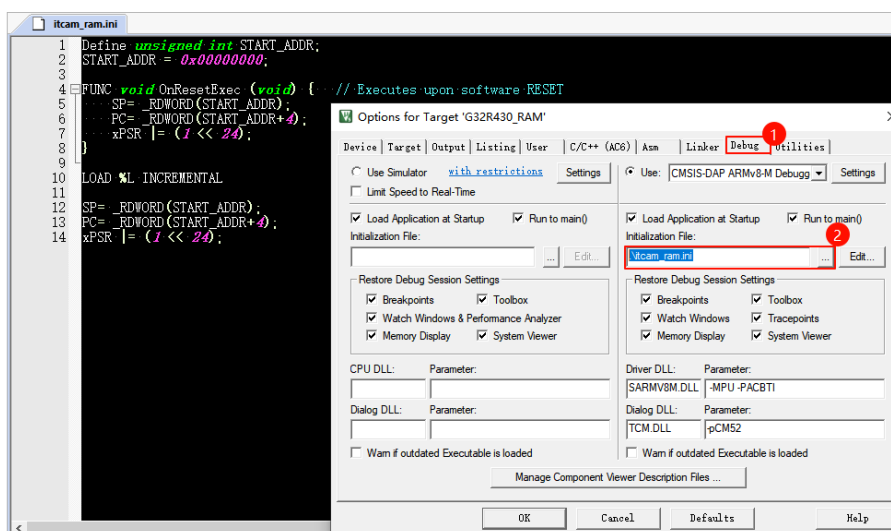
- OPT 代码下载设置(如需写入模拟 OTP/配置区域), 需要额外执行图 8 中的步骤 5、6、7 以添加 OPT 下载算法。
- RAM 程序下载设置(若需要在 RAM 中运行程序), 则需在图 8 中的步骤 4 将编程区域改为要下载的区域, 跳过执行的步骤 5、6、7。

注: 下载区域与下载算法执行所使用的 RAM 区域不能重叠, 否则会导致下载算法无法正常运行。

4.1.5 调试程序

在调试配置中设置 .ini 脚本文件(如果程序运行地址与芯片启动地址不一致), 确保 PC 程序计数器被初始化到正确的启动地址。 .ini 脚本文件可参考: G32R430_DDL_SDK_Vx.x.x \Examples\Board_G32R430_Tiny\ATAN2\ATAN2_Math\Project\MDK\litcam_ram.ini

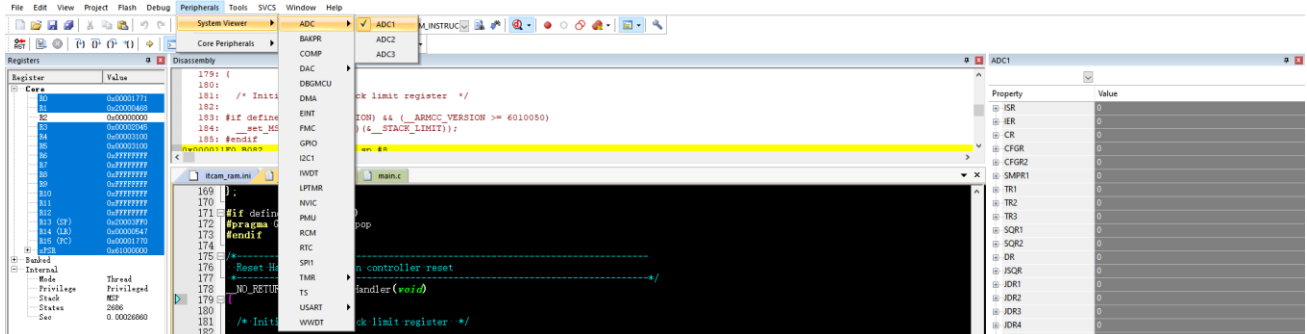
图 9 MDK 配置 .ini 文件



点击 “Debug” 进入调试界面, 可查看:

- 通用寄存器(通用 CPU 寄存器内容)。
- 内核、外设寄存器(各外设模块的寄存器配置及状态)。

图 10 MDK Debug 界面查看外设寄存器

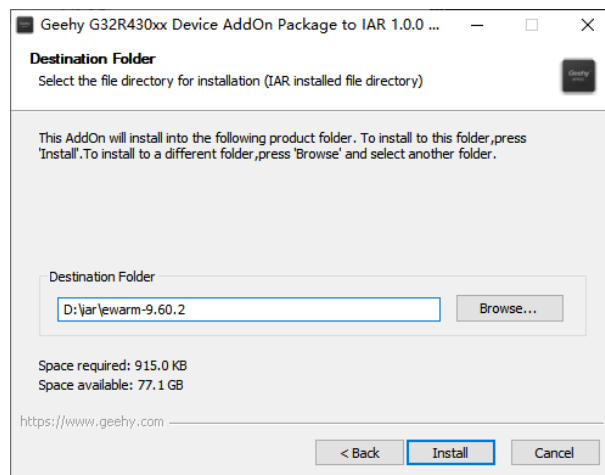


4.2 IAR for Arm 下运行例程

4.2.1 安装芯片支持

运行 Package\G32R430xx_AddOn_vx.x.x.exe 安装程序。安装路径需与本地已安装的 IAR 路径匹配。例如：演示电脑的 IAR 启动程序位于 D:\iar\ewarm-9.60.2\common\bin\iarIdePm.exe，则将安装路径设置为 D:\iar\ewarm-9.60.2。

图 11 G32R430xx_AddOn_vx.x.x.exe 安装程序



4.2.2 使用例程

打开示例工程同一目录下的 IAR 工程。例如：

Board_G32R430_Tiny\ADC12\ADC12_AnalogWindowWatchdog\Project\IAR

在该目录下，双击 .eww 文件即可加载示例工程。

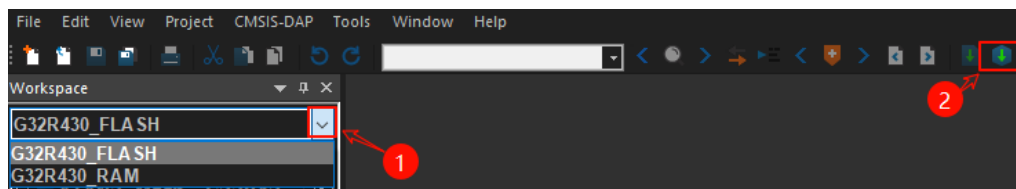
图 12 ADC12_AnalogWindowWatchdog 例程 IAR 工程示意

Board_G32R430_Tiny > ADC12 > ADC12_AnalogWindowWatchdog > Project > IAR			
名称	修改日期	类型	大小
ADC12_AnalogWindowWatchdog.ewp	2025/11/7 11:01	EWP 文件	48 KB
ADC12_AnalogWindowWatchdog.eww	2025/11/7 11:01	IAR IDE Worksp...	8 KB

4.2.3 编译例程

打开工程后, 可在 IAR 的 Workspace 中查看项目结构。检查工程目标(Target)配置是否为自己所需要, 点击 “Make” 或相应的编译按钮, 等待工程编译完成。若编译正常, Console 会显示无错误和警告。

图 13 IAR 下的多 Target 选择与编译



4.2.4 下载程序

在调试器选项中, 选择使用 “CMSIS-DAP ARMv8-M Debugger”。然后在点击状态栏的 “Project” 按钮选择 “Download” 下的 “Download active application” 进行程序下载。

图 14 IAR 配置仿真器与程序下载

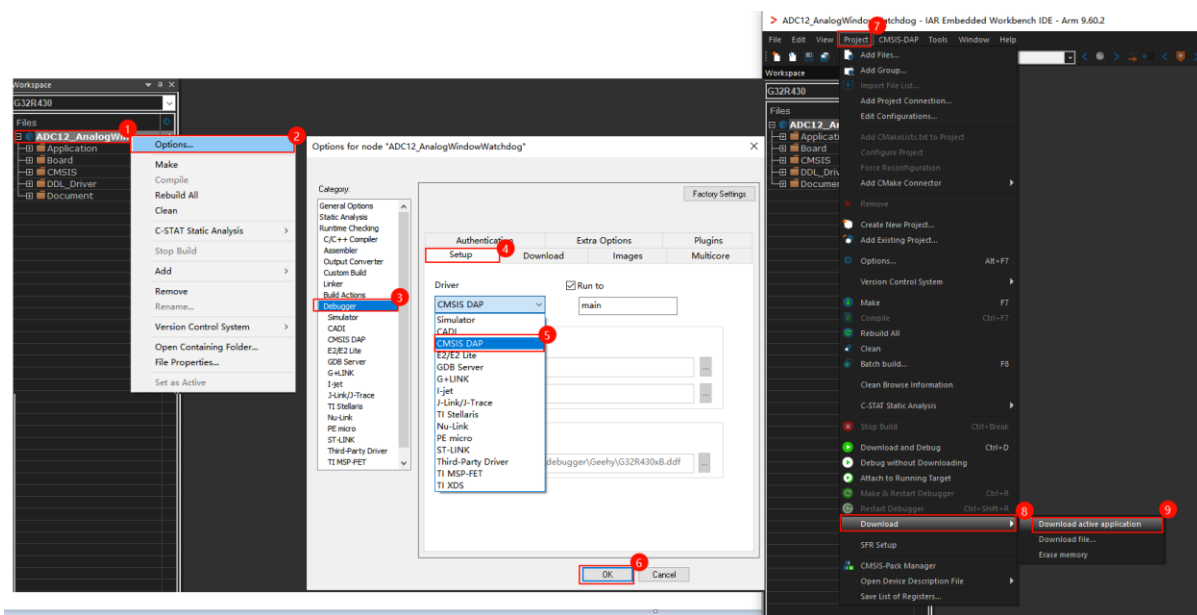
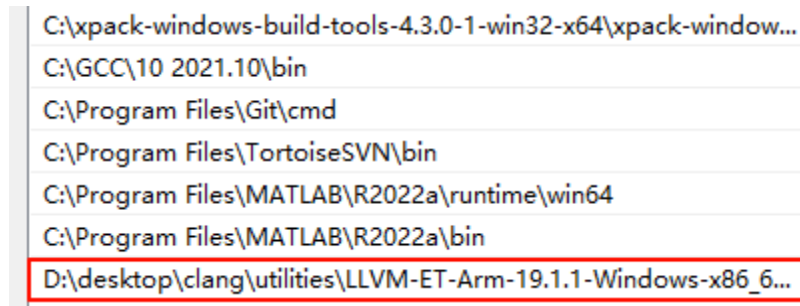


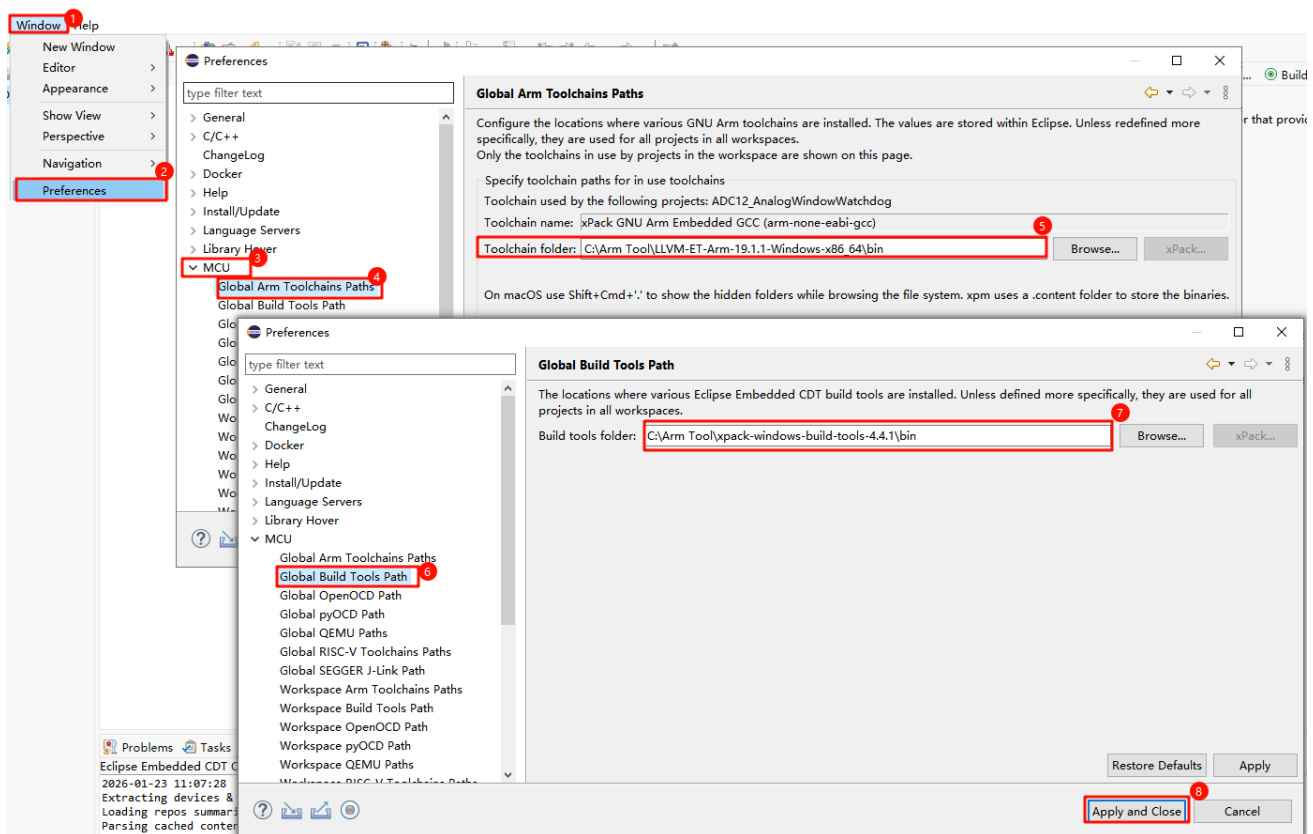
图 16 添加系统环境变量



3. GDB 服务。建议使用 Arm 提供的 arm-none-eabi-gdb.exe。示例使用的 arm-none-eabi-gdb.exe 版本是 14.2。
4. Make 工具。建议使用 xpack-windows-build-tools，示例使用的 4.4.1 版本的。

以上工具链可配置至 Eclipse 的全局 Path 中。可参考图示意。

图 17 Eclipse 添加 MCU Global 工具链配置



4.3.2 pyOCD 适配

为便于用户能够在开源环境下对 G32R430 进行程序下载仿真等操作，G32R430 系列 MCU 需要对 pyocd 进行支持。

目前 SDK 中的 Eclipse 例程在下载过程中使用的是 pyOCD 0.36

(<https://github.com/pyocd/pyOCD/releases/tag/v0.36.0>)，该版本不支持 M52 内核芯片以及 G32R430 芯片，需要修改其源码以完成支持。

1. 在 pyOCD 中添加 M52 内核支持，主要修改的文件是：

a) pyocd\coresight\component_ids.py，在 class CmpInfo(NamedTuple):下添加：

# 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),

图 18 修改后 component_ids.py

```

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

```

b) pyocd\coresight\core_ids.py:

1) 在# CPUID PARTNO values 下添加内核 ID

```
ARM_China_StarMC2 = 0xD24
```

2) 在 CORE_TYPE_NAME: Dict[Tuple[int, int], str]下添加内核名称

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

图 19 修改后 core_ids.py

```

39 ARM_CortexM55 = 0xD22
40 ARM_CortexM85 = 0xD23
41 ARM_China_StarMC1 = 0x132
42 ARM_China_StarMC2 = 0xD24
43
44 # pylint: enable=invalid_name
45
46 ## @brief User-friendly names for core types.
47 CORE_TYPE_NAME: Dict[Tuple[int, int], str] = {
48     (CPUID_ARM, ARM_SC000): "SecurCore SC000",
49     (CPUID_ARM, ARM_SC300): "SecurCore SC300",
50     (CPUID_ARM, ARM_CortexM0): "Cortex-M0",
51     (CPUID_ARM, ARM_CortexM1): "Cortex-M1",
52     (CPUID_ARM, ARM_CortexM3): "Cortex-M3",
53     (CPUID_ARM, ARM_CortexM4): "Cortex-M4",
54     (CPUID_ARM, ARM_CortexM7): "Cortex-M7",
55     (CPUID_ARM, ARM_CortexM0p): "Cortex-M0+",
56     (CPUID_ARM, ARM_CortexM23): "Cortex-M23",
57     (CPUID_ARM, ARM_CortexM33): "Cortex-M33",
58     (CPUID_ARM, ARM_CortexM35P): "Cortex-M35P",
59     (CPUID_ARM, ARM_CortexM55): "Cortex-M55",
60     (CPUID_ARM, ARM_CortexM85): "Cortex-M85",
61     (CPUID_ARM_CHINA, ARM_China_StarMC1): "Star-MC1",
62     (CPUID_ARM_CHINA, ARM_China_StarMC2): "Star-MC2",
63 }

```

2. 添加 G32R430 芯片支持。

- 在 pyocd\target\builtin 中添加 g32r430 下载算法支持文件: target_G32R430xx.py。此文件已添加在 SDK/Package/pyOCD/target_G32R430xx.py。
- 添加 g32r430 支持, 在 pyocd\target\builtin__init__.py 中加入:

```

from . import target_G32R430xx

'g32r430xb': target_G32R430xx.G32R430xB,

```

图 20 修改后 __init__.py

```

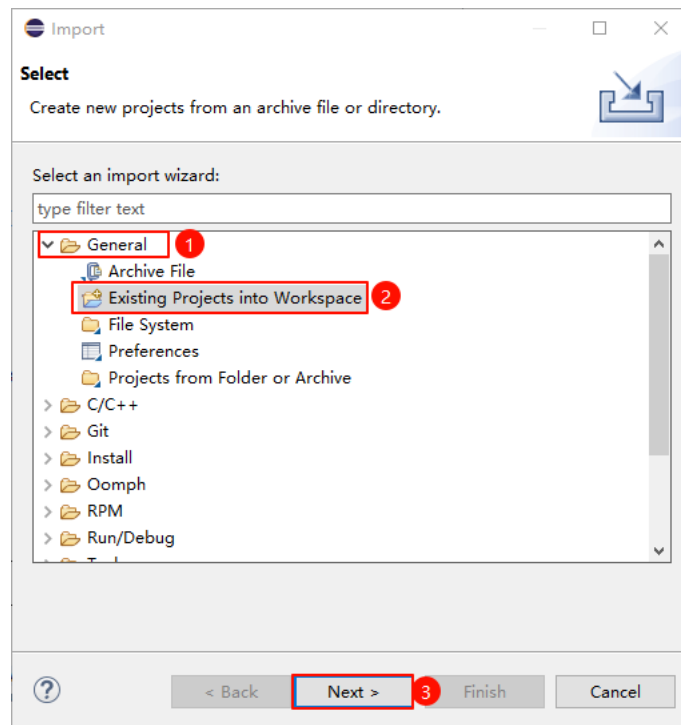
137 from .. import target_Air001
138 from .. import target_Air32F103xx
139 from .. import target_G32R501xx
140 from .. import target_G32R430xx
141
142 .....: 'air001': target_Air001.Air001,
143 .....: 'air32f103xb': target_Air32F103xx.Air32F103xB,
144 .....: 'air32f103xc': target_Air32F103xx.Air32F103xC,
145 .....: 'air32f103xp': target_Air32F103xx.Air32F103xP,
146 .....: 'air32f103xe': target_Air32F103xx.Air32F103xE,
147 .....: 'air32f103xg': target_Air32F103xx.Air32F103xG,
148 .....: 'g32r501dxx': target_G32R501xx.G32R501Dxx,
149 .....: 'g32r501xx': target_G32R501xx.G32R501xx,
150 .....: 'g32r430xb': target_G32R430xx.G32R430xB,

```

4.3.3 使用例程

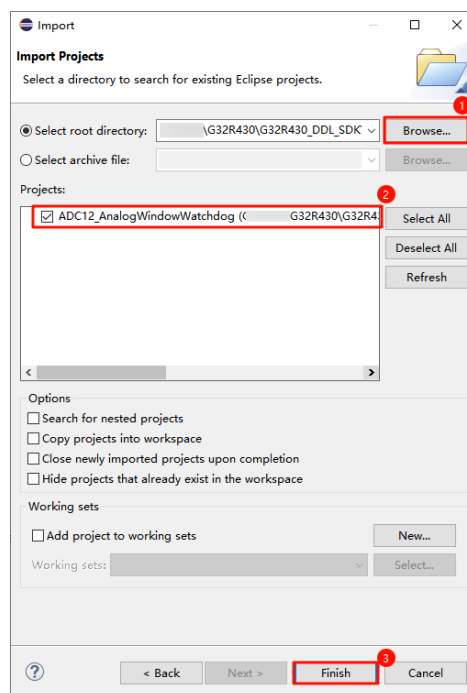
- 启动 Eclipse 4.35。
- 点击菜单栏中的 “File” -> “import ...” -> “General” -> “Existing Project into Workspace”

图 21 Import project 1



3. 点击“Select root directory”，浏览到你保存的 SDK 工程文件的路径，选择对应的工程文件夹，然后点击“Finish”。

图 22 Import project 2



注意：以上步骤请先完成章节 4.3.1 的 LLVM 编译配置。

4.3.4 编译例程

打开工程后, 可在 Eclipse 的 Project Explorer 中查看项目结构。检查工程目标(Target)配置是否为自己所需要, 点击 “Build” 编译按钮, 等待工程编译完成。若编译正常, Console 会显示无错误和警告。

图 23 Eclipse 下编译例程操作

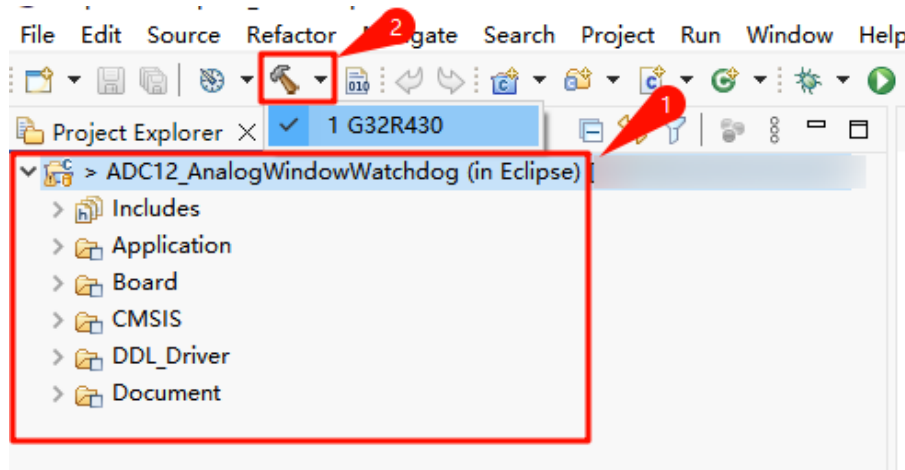


图 24 Eclipse 下成功编译例程



4.3.5 下载与调试程序

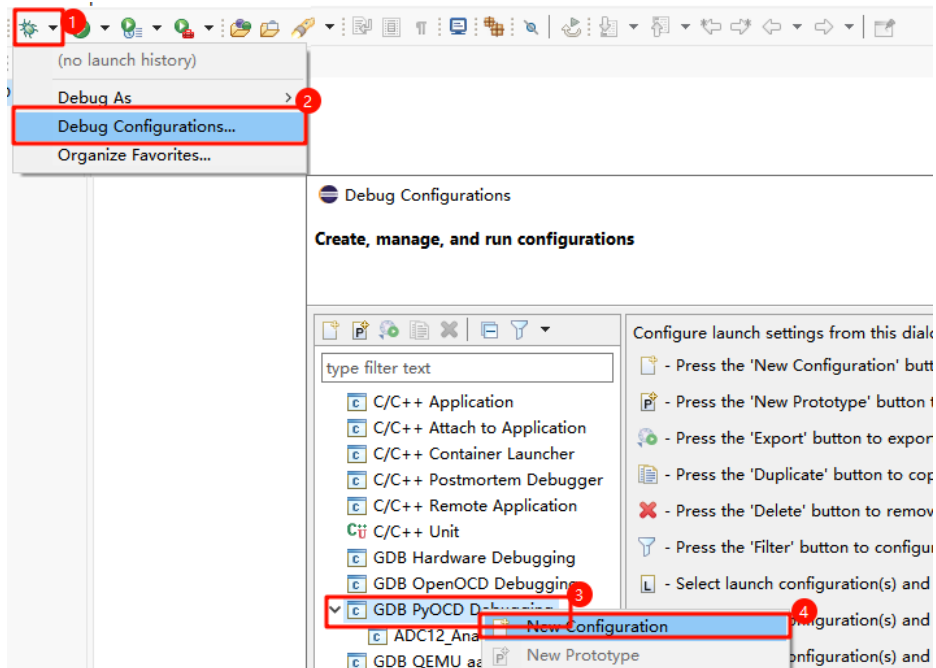
在 Eclipse 下程序的下载与调试需要配置 GDB 服务, 请按照如下步骤配置调试选项卡。

注: 如下步骤需完成 python、pyOCD 以及 pyOCD 的 G32R430 芯片支持后进行。可参考章节 4.3.2 与章节 5 进行环境部署工作。

1. 新增 pyOCD Debugging 配置

- 1) 在 Debug 图标处左键, 以显示 Debug 配置。
- 2) 选择显示出来的 “Debug Configurations...”。
- 3) 在新窗口选择 “GDB PyOCD Debugging”, 右键。
- 4) 选择 “New Configuration” 以进行仿真配置。

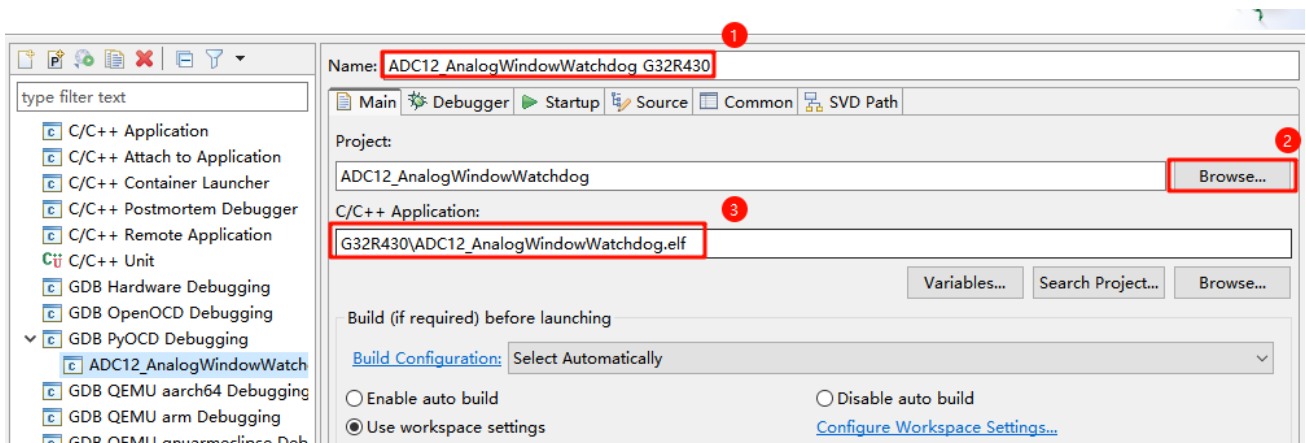
图 25 新增 pyOCD Debugging 配置



2. 配置 Main 选项卡

- 1) 在最上端命名当前仿真配置名称。
- 2) 选择“Browse...”，选择当前仿真配置对应的工程。
- 3) 选择对应的仿真 elf 文件，例如：G32R430\ADC12_AnalogWindowWatchdog.elf。示例使用的是相对于工程文件的相对路径，可支持绝对路径。

图 26 配置 Main

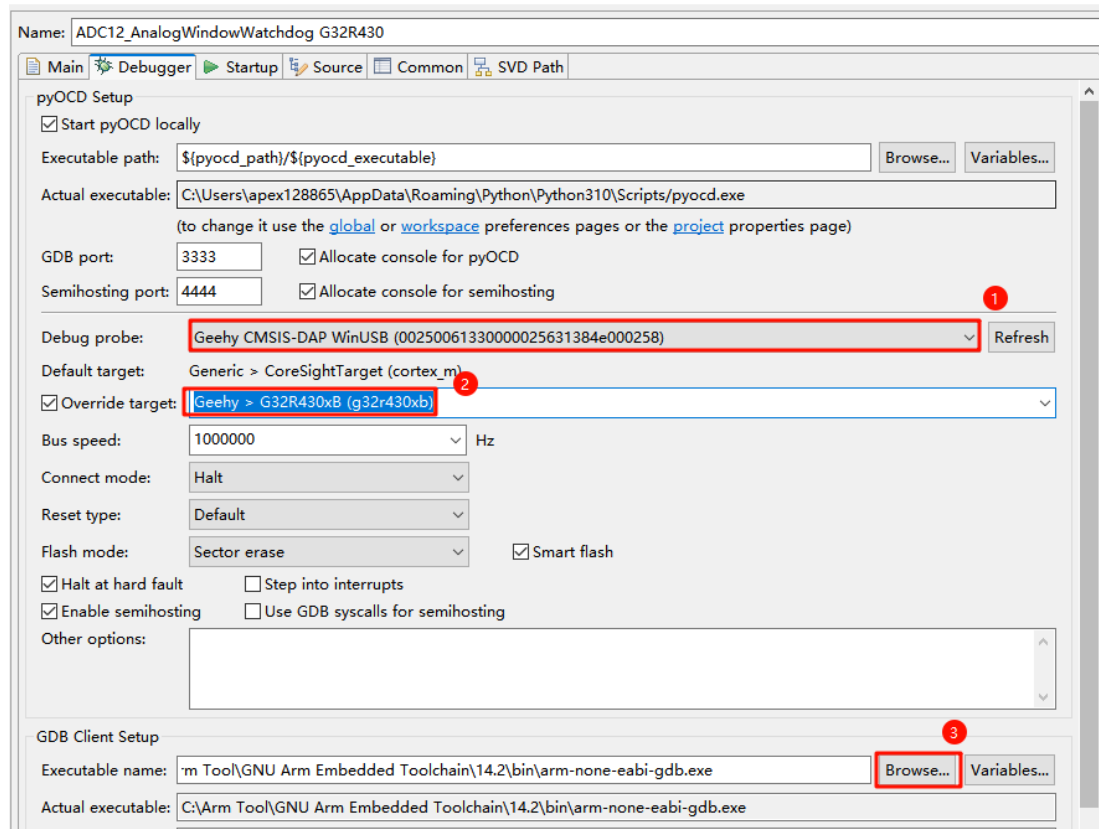


3. 配置 Debugger 选项卡

- 1) 选择使用到的 Geehy-Link 仿真器，括号中的字符为仿真器序列号

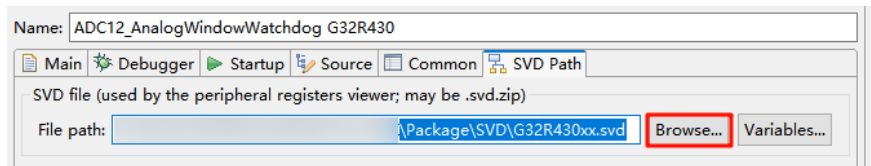
- 2) 选择对应的仿真芯片，例如：Geehy > G32R430xB(g32r430xb)
- 3) 选择 GDB 服务，这里建议使用 Arm 提供的 arm-gnu-toolchain-14.2.rel1\bin\arm-none-eabi-gdb.exe。

图 27 配置 Debugger



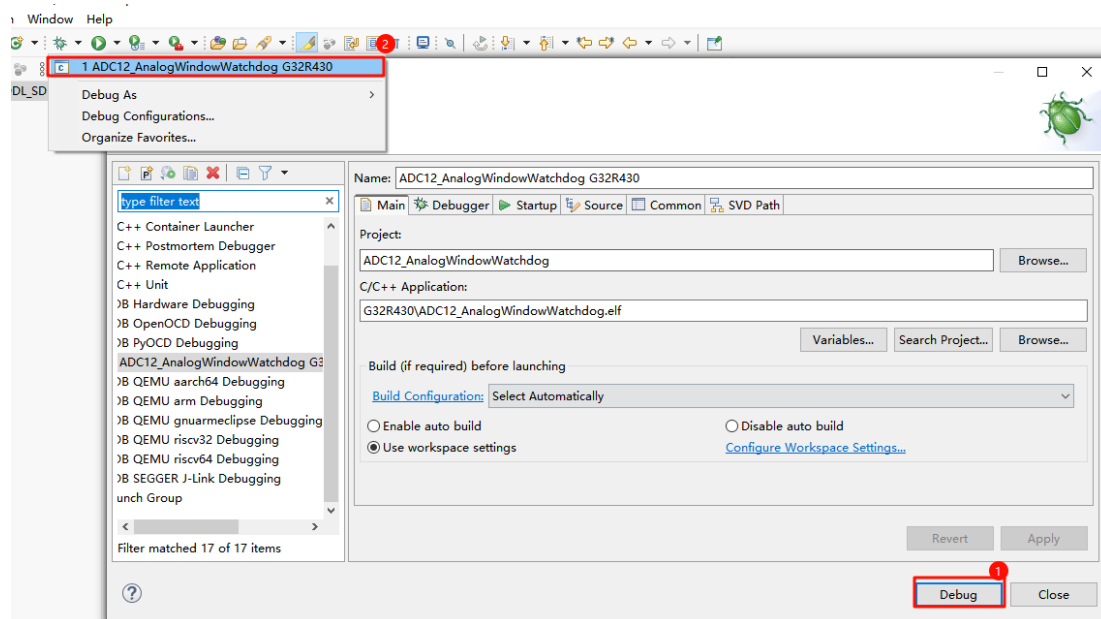
4. 配置 SVD 文件选项卡。SVD 文件提供用户便捷的查看 MCU 的外设寄存器内容。G32R430 请选择其 SDK 中的 Package\SVD\G32R430xx.svd。

图 28 配置 SVD 文件选项卡



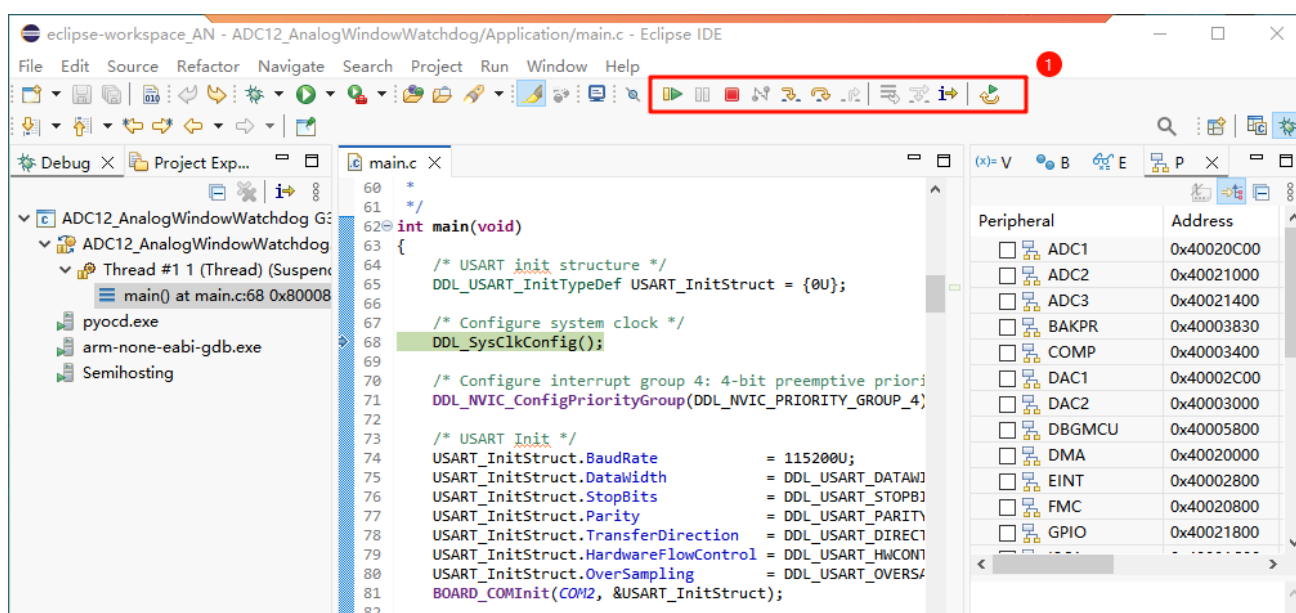
5. 最后点击选项卡的右下角的“Apply”按钮，应用所有的配置项。
6. 启动下载或调试。第一次需要点击选项卡的右下角的“Debug”按钮进行仿真。后续可通过工具栏中的 Debugger 按钮进行。

图 29 启动下载或调试



7. 调试成功。请使用工具栏中的调试按钮控制程序。

图 30 调试成功



5 pyOCD 安装

5.1 Windows

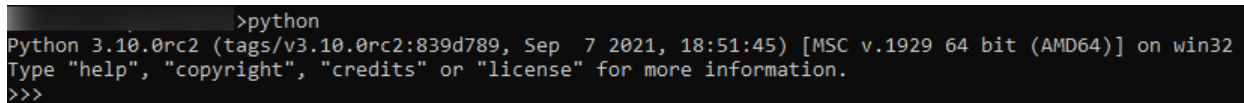
5.1.1 安装 python

pyOCD 支持需要在 python 环境下, 请至 python 官网 (<https://www.python.org>) 下载最新的 python 安装包进行安装。

注意: 安装完成后, 请确保将 Python 添加到系统的 PATH 环境变量中, 以便在命令行中使用。

验证: 使用 Win+R 键, 输入 CMD, 在命令行窗口上输入: python, 然后回车。显示已安装的 Python 版本号等内容, 并进入 Python 的交互式命令行 (REPL) 中。如需退出输入 exit() 或 quit(), 然后按下回车键。

图 31 python 安装验证



```
>python
Python 3.10.0rc2 (tags/v3.10.0rc2:839d789, Sep 7 2021, 18:51:45) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

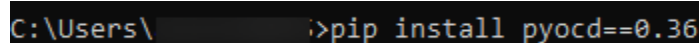
5.1.2 安装 pyOCD

pyOCD 是 python 的一个组件包, 支持在线安装 (建议使用在线安装, 因为有不同的依赖包以便在线安装)。

安装方法参考如下:

1. 使用 pip 命令 “pip install pyocd==0.36” 安装 0.36 版本的 pyOCD:

图 32 安装 pyOCD



```
C:\Users\>pip install pyocd==0.36
```

注意: 安装完成后, 请将 pyOCD 的安装路径添加至系统的 PATH 环境变量中, 以便于在命令行中使用。例如:

C:\Users\Geehy\AppData\Local\Programs\Python\Python313\Scripts

2. 验证安装成果, 使用 Win+R 键, 输入 CMD, 在命令行窗口上输入: pyocd -h, 会显示命令帮助。

图 33 pyOCD 安装验证

```
C:\Users\...>pyocd -h
usage: pyocd [-h] [-V] [--help-options] ...

PyOCD debug tools for Arm Cortex devices

options:
  -h, --help            show this help message and exit
  -V, --version          show program's version number and exit
  --help-options        Display available session options.
```

5.2 Ubuntu

5.2.1 安装 python

1. Ubuntu 一般默认带 Python，可在终端输入命令以下查询 Python 和 pip 版本。

```
python --version
```

2. 如果没有安装 Python 或 pip，请使用以下命令安装：

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

5.2.2 python3-venv

部分 Ubuntu 自带的 Python3 环境是外部管理的，无法在直接使用 pip 在全局环境中安装包。为了避免这个问题，可以直接使用 Python 自带的 venv 模块来创建虚拟环境。

1. 使用以下命令来安装 python3-venv 包：

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

2. 使用以下命令来创建虚拟环境（假设你将虚拟环境命名为 venv）：

```
python3 -m venv venv
```

3. 使用以下命令来激活虚拟环境：

- 激活虚拟环境，以便在其中安装包

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

- 若后续想退出虚拟环境可使用以下命令退出:

```
deactivate
```

5.2.3 安装 pyOCD

1. 在激活的虚拟环境中, 使用 pip 安装 pyOCD:

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

2. 验证安装结果

```
pyocd --version
```

3. 查询 pyOCD 安装位置 (以便后续更改 pyOCD 源码)

```
pip show pyocd
```

5.2.4 pyOCD 的 USB 权限

如果 pyOCD 在普通用户下无法访问调试器, 可以考虑给当前用户添加适当的权限。使用 udev 规则来确保你可以在不使用 sudo 的情况下访问 USB 设备。步骤如下:

1. 创建一个新的规则文件, 在终端中执行以下命令来创建新的 udev 规则文件 (例如命名为 99-pyocd.rules)

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

2. 在文件中添加以下内容 (Geehy-Link 设备 ID 是 314B)

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

在 nano 编辑器中, 按 **Ctrl + O** 保存文件, 然后按 **Enter** 确认。接着按 **Ctrl + X** 退出编辑器。

3. 保存文件后, 重新加载 udev 规则

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

```
sudo udevadm trigger
```

4. 验证 pyOCD 是否可正常识别 Geehy-Link

```
pyocd list
```

图 34 连接至 Ubuntu 的仿真器序列号

```
(venv) kai@Ubuntu:~$ pyocd list
```

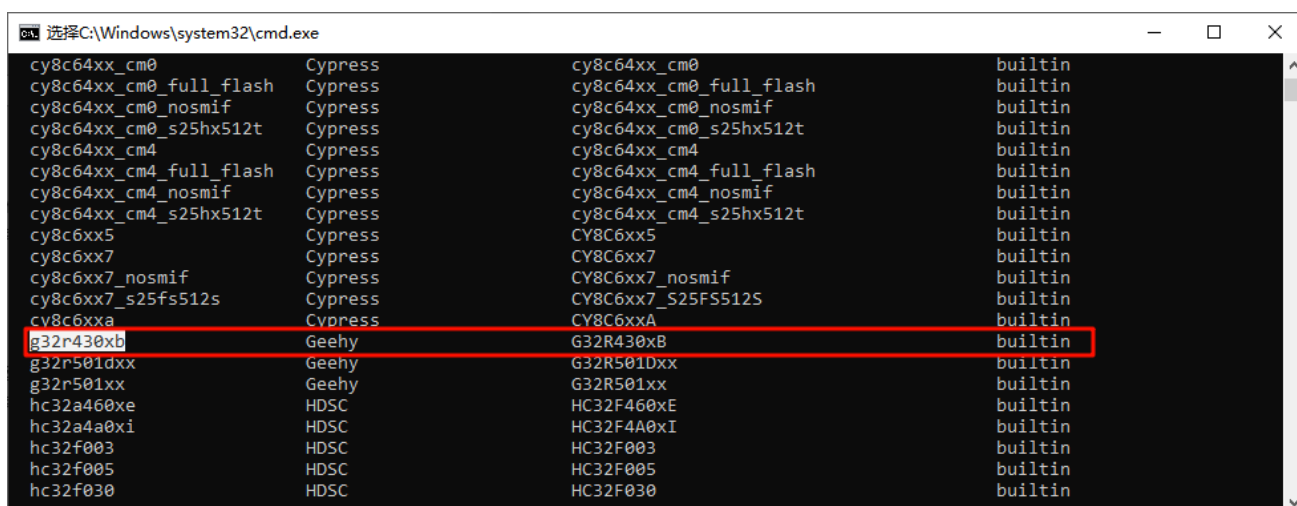
#	Probe/Board	Unique ID	Target
0	Geehy CMSIS-DAP WinUSB	00350043500000144e544859000258	n/a

5.3 替换修改项内容

为支持 G32R430, 请参考 4.3.2 章节内容修改已下载完毕的 pyOCD 内容。

验证 G32R430 支持是否已添加成功, 使用 Win+R 键, 输入 CMD, 在命令行窗口上输入: `pyocd list --targets`, 会显示所有芯片, 在支持的芯片中查找是否有 "g32r430xb".

图 35 支持芯片列表



Target	Manufacturer	Device	Driver
cy8c64xx_cm0	Cypress	cy8c64xx_cm0	builtin
cy8c64xx_cm0_full_flash	Cypress	cy8c64xx_cm0_full_flash	builtin
cy8c64xx_cm0_nosmif	Cypress	cy8c64xx_cm0_nosmif	builtin
cy8c64xx_cm0_s25hx512t	Cypress	cy8c64xx_cm0_s25hx512t	builtin
cy8c64xx_cm4	Cypress	cy8c64xx_cm4	builtin
cy8c64xx_cm4_full_flash	Cypress	cy8c64xx_cm4_full_flash	builtin
cy8c64xx_cm4_nosmif	Cypress	cy8c64xx_cm4_nosmif	builtin
cy8c64xx_cm4_s25hx512t	Cypress	cy8c64xx_cm4_s25hx512t	builtin
cy8c6xx5	Cypress	CY8C6xx5	builtin
cy8c6xx7	Cypress	CY8C6xx7	builtin
cy8c6xx7_nosmif	Cypress	CY8C6xx7_nosmif	builtin
cy8c6xx7_s25fs512s	Cypress	CY8C6xx7_S25FS512S	builtin
cy8c6xxa	Cypress	CY8C6xxA	builtin
g32r430xb	Geehy	G32R430xB	builtin
g32r501dxx	Geehy	G32R501Dxx	builtin
g32r501xx	Geehy	G32R501xx	builtin
hc32a460xe	HDSC	HC32F460xE	builtin
hc32a4a0xi	HDSC	HC32F4A0xI	builtin
hc32f003	HDSC	HC32F003	builtin
hc32f005	HDSC	HC32F005	builtin
hc32f030	HDSC	HC32F030	builtin

5.4 命令行使用

pyOCD 支持在 CMD 命令行使用, 使用方法可参考如下步骤 (使用前需确认 pyOCD 已添加至 PATH)。

1. 板卡或仿真器与芯片连接一起并连接至 PC。
2. 在工作目录下启动 cmd, 并输入: `pyocd commander -t g32r430xb`, 随后可在命令行窗口输入相关指令进行对应的操作。

图 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 slated
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 集成至 Eclipse

目前使用 Eclipse+pyocd 对 Eclipse 的版本有需求, 建议使用 202503 版本的 Eclipse。

此外, 建议将 pyOCD 的路径在 Eclipse 进行全局配置 (部分电脑下发现 eclipse 获取 PATH 可能有异常) 步骤如下:

1. 在菜单栏 “Windows” 处右键, 显示所有配置。
2. 在显示的配置中选择 “Preference”。
3. 在新窗口下, 打开 “MCU” 选项下的子选项。
4. 选择子选项 “Global pyOCD Path”。
5. 在右侧选择对应的 pyocd.exe 所在目录。
6. 最后点击 “Apply and Close”。

图 37 Global pyOCD Path 步骤 1-2

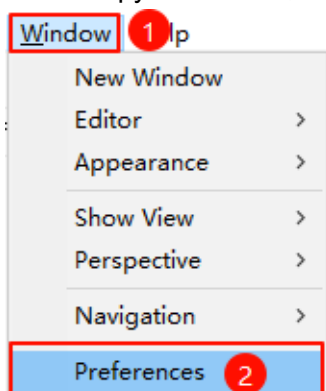
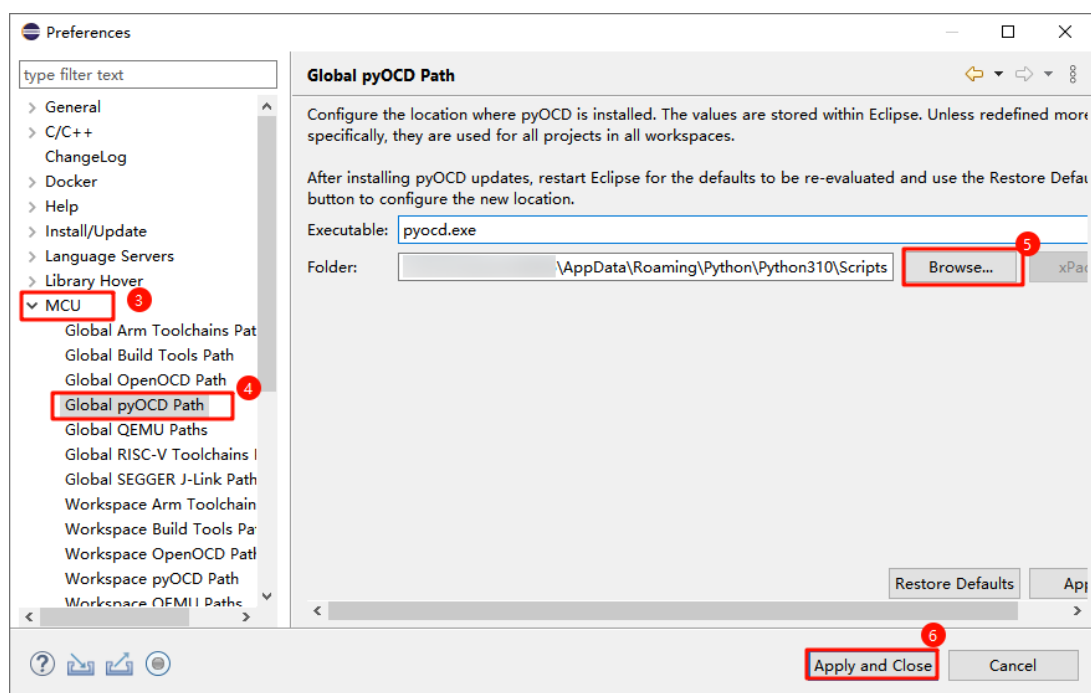


图 38 Global pyOCD Path 步骤 3-6



6 关于链接脚本

链接脚本是一种决定程序各内存段放置位置与结构的配置文件，在嵌入式开发中用于将编译后的对象代码、常量、全局变量以及中断向量等映射到 MCU 的 Flash、RAM 等物理存储区域。本小节将介绍 G32R430 系列在 MDK、IAR 和 Eclipse (gcc) 三种开发环境下链接脚本的存放位置、所对应的内存结构，以及常见的示例内存配置文件，帮助你在工程中选择合适的存放策略并理解后续字段与区段的作用。

6.1 链接脚本基本情况

- 存放位置

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

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

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

- 芯片内存结构(G32R430xB)

Flash: 128 KB

DTCM RAM: 16 KB

ITCM RAM: 32 KB

- 示例内存配置文件说明

g32r430xb_flash.sct/icf/ld: 程序运行于 Flash，堆栈存放在 DTCM RAM，未使用 ITCM RAM 空间。

g32r430xb_itcm_ram.sct/icf/ld: 程序运行于 ITCM RAM，堆栈存放在 DTCM RAM，未使用 Flash 空间。

g32r430xb_flash_option.ld: 程序运行于 Flash，堆栈存放在 DTCM RAM，未使用 ITCM RAM 空间，且包含了 opt 选项字节配置区域。用于

“\Examples\Board_G32R430_Tiny\FLASH\FLASH_OPT” 例程。

注：以上配置用于不同存储区域的性能对比与特定场景优化，请在工程需求与资源约束允许的情况下选择合适脚本。

6.2 字段说明

在 g32r430.h 头文件中定义了若干字段，这些字段在链接脚本中对应具体的段定义。常见字段及其含义如下：

1. SECTION_ITCM_INSTRUCTION: ITCM 指令代码段
2. SECTION_ITCM_RAMFUNC: ITCM RAM 中的函数段(RAM 内执行的函数)
3. SECTION_DTCM_DATA: DTCM 数据段

4. SECTION_DTCM_BSS: DTCM BSS(未初始化全局/静态变量)段
5. SECTION_RAM_VEC: 中断向量表所在段, 通常放置于 DTCM RAM, 用户可以修改链接脚本中的定义位置以修改其存放位置。

6.3 如何指定变量/函数存放区域

通过将变量和函数放置在不同的内存区域, 可以在不同应用场景下获得更优的启动时间、执行效率和内存利用率。本小节给出常用的分区及对应的代码示例, 便于在实际工程中快速实现目标区域的放置。

示例目标:

- 将对启动时间、运行速度敏感数据放入 DTCM RAM, 以获得更低访问延迟。

注: DMA 访问的 RAM 内存地址必须是在 DTCM RAM 中。

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

- 将对启动时间敏感或需要快速执行的函数放到 ITCM 区域(指令区/RAMFUNC 区), 以提升启动和执行效率。

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

或

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

7 ATAN2 Libraries

为满足某些应用场景对角度计算的需求，G32R430 MCU 提供了 ATAN2 库，能够快速计算给定坐标(nX, nY)对应的角度值。该函数在电机控制、轨迹规划等场合都有广泛应用价值。

7.1 ATAN2 库文件结构

在 SDK 的 “Libraries/ATAN2” 目录下提供了不同编译器及精度版本的库文件，主要包括：

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

其中：

- “AC6” 表示库文件适用于 MDK(AC6 编译器)环境
- “ICC” 表示库文件适用于 IAR(ICC 编译器)环境
- “high_resolution” 与 “low_resolution” 则代表了不同的精度实现，用户可根据应用需求来选择合适的库

7.2 函数原型与说明

ATAN2 函数允许用户通过指定精度等级，计算给定坐标在二维平面上的角度。具体原型如下：

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

- 参数说明：
 - nX, nY: 分别表示待计算点的 X、Y 坐标(以 Q32 格式输入);
 - nPrecisionLevel: 计算精度等级，范围为 [1, 8]，推荐使用 6、7 或 8;
- 返回值：
 - 以 Q31 格式返回 $(-\pi, \pi]$ 区间内的角度值。原点(0,0)属于特殊情况，返回结果需用户结合应用场景进行处理或判断。

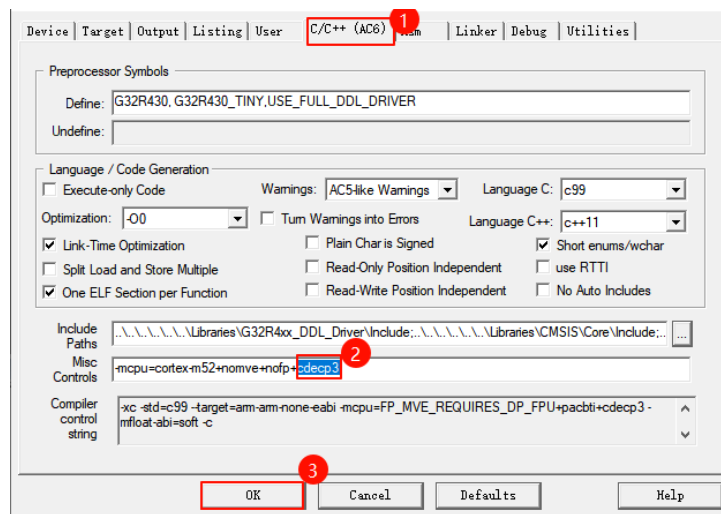
7.3 函数存储与执行位置

ATAN2 函数的代码段默认为 “atan2_instruction”，可在链接脚本(如 .sct 或 .icf 文件)中修改该段的地址或将其映射到需要的存储空间。若有特别的性能或内存需求，可按照工程需求对链接脚本进行相应调整。

7.4 使用方法

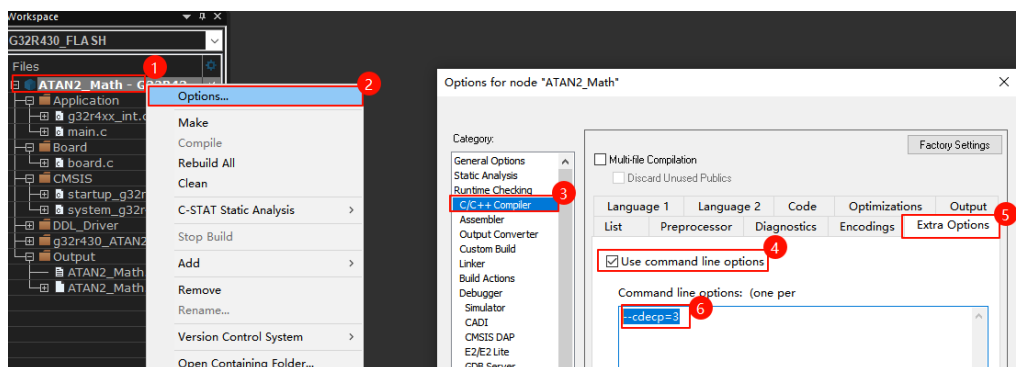
1. 选择合适的库文件: 根据编译器(MDK 或 IAR)以及对计算精度的需求 (high/low_resolution), 在工程的链接选项中包含相应的 “.lib” 或 “.a” 文件。
2. 在工程中开启 CDE3 支持:
 - 对于 MDK, 需要在项目设置中启用对 CDE3 的支持;

图 39 MDK 下为 C 代码开启 CDE3 支持



- 对于 IAR, 需在项目选项的编译设置中启用对应 CDE3 的扩展指令集支持。

图 40 IAR 下为 C 代码开启 CDE3 支持



3. 在工程中包含 ATAN2 头文件:
 - 打开项目设置, 确认已将 “Libraries\ATAN2” 目录添加到 Include 路径;
 - 在需要调用 ATAN2 函数的源文件顶部, 加上相关头文件引用。

注: 示例程序可参考: G32R430_DDL_SDK_Vx.x.x\Examples\Board_G32R430_Tiny\ATAN2\ATAN2_Math\

8 版本历史

表格 1 文件版本历史

日期	版本	变更历史
2025.11	1.0	<ul style="list-style-type: none">新建
2025.12	1.1	<ul style="list-style-type: none">2.1 更新板卡图片为 V1.2, 删除对应 V1.1 版本所需操作内容
2026.01	1.2	<ul style="list-style-type: none">新增章节 4.3 Eclipse 环境运行例程的说明新增章节 5 pyOCD 安装说明及如何添加 G32R430 芯片支持章节 6 新增 gcc 环境下链接脚本说明

声明

本手册由珠海极海半导体有限公司(以下简称“极海”)制订并发布, 所列内容均受商标、著作权、软件著作权相关法律法规保护, 极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册, 一旦使用产品则表明您(以下称“用户”)已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用, 未经极海许可, 任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“™”的“极海”或“Geehy”字样或图形均为极海的商标, 其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权, 不应被视为极海授权用户使用前述第三方产品、服务或知识产权, 也不应被视为极海对第三方产品、服务或知识产权提供任何形式的保证, 包括但不限于任何第三方知识产权的非侵权保证, 除非极海在销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的, 应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户(代表其本身、子公司及关联企业)应同意并保证遵守所有关于取得极海产品及/或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”(as is)提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

极海产品并非设计、授权或担保适合用于军事、生命保障系统、污染控制或有害物质管理系统中的关键部件，亦非设计、授权或担保适合用于在产品失效或故障时可导致人员受伤、死亡、财产或环境损害的应用。

如果产品未标明“汽车级”，则表示不适用于汽车应用。如果用户对产品的应用超出极海提供的规格、应用领域、规范，极海不承担任何责任。

用户应该确保对产品的应用符合相应标准以及功能安全、信息安全、环境标准等要求。用户对极海产品的选择和使用负全部的责任。对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册及产品的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册及产品而产生的直接、间接或附带损害(包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失)，这涵盖了可能导致的人身安全、财产或环境损害等情况，对于这些损害极海概不承担责任。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。

©2026 珠海极海半导体有限公司 – 保留所有权利