使用说明书

User Manual

文档编号：**AN1126**

G32R5xx IDE 与工具链使用说明

版本：**V1.2**

© 珠海极海半导体有限公司

1 引言

在现代嵌入式系统开发中，选择合适的开发工具和环境对于成功实现项目目标至关重要。无论是新项目的开发还是现有项目的移植，开发人员都需要灵活应对不同的开发环境和芯片架构。本文旨在为开发人员提供一份指南，帮助在使用 G32R5xx 系列微控制器（含 G32R501、G32R502）进行开发时顺利配置和使用常用开发环境。

为了充分利用本手册中的信息，用户应熟悉所用器件的特性。可以参考以下相关技术文档：

- G32R5xx / G32R501 / G32R502 用户手册、数据手册与编程手册、迁移手册（AN1138）
- 内核相关文档，包括 Armv8.1-M 架构参考手册等
- SDK 快速上手指南（AN1125）

本文档详细介绍在 Keil MDK、IAR Embedded Workbench for Arm 与 Eclipse 中配置工程的步骤，涵盖从工程创建、编译链接到调试的全过程。示例工程在 SDK 中按器件分路径，例如 driverlib/g32r501/examples/ 与 driverlib/g32r502/examples/。

本文档将引导读者掌握在上述环境下开发 G32R5xx 的基本技巧。

目录

2 术语全称、缩写描述

关于本文档所使用的一些关键词及描述如表格 1 所示。

表格 1 缩写描述

中文全称	英文全称	英文缩写
可编程静态存储子系统	Configurable Static Memory Subsystem	CFGSMS
软件	Software.	SW
硬件	Hardware.	HW
接口	Interface	IF
AHB从接口	AHB slave interface.	ahbs_if

3 R5xx 系列 MCU 简介

G32R5xx（R5xx）系列 MCU 基于 Arm Cortex-M52 内核。其中：

- G32R501 为双/单 Cortex-M52 核心器件；
- G32R502 为单 Cortex-M52 核心器件。

片上存储与外设规模因订货型号而异；ITCM / DTCM / SRAM 等容量可通过 CFGSMS 配置。本手册后续章节按 Keil MDK、IAR Embedded Workbench for Arm、Eclipse 与 pyOCD 说明工程与调试配置；例程路径按器件区分（如 driverlib/g32r501 与 driverlib/g32r502）。

4 MDK-ARM 开发工具链

在从 Txx320F28004x 系列微控制器迁移到 G32R5xx 系列微控制器（含 G32R501、G32R502）的过程中，开发工具的迁移是一个重要环节。Code Composer StudioTM（CCStudio）集成开发环境（IDE）和 Keil MDK 都是嵌入式开发中的常用工具。

本章节将介绍如何将现有的 CCStudio 工程迁移到 Keil MDK 环境中，并详细讨论 C 语言和汇编代码的兼容性，链接脚本文件的配置等内容。

4.1 调试器支持

G32R5xx（含 G32R501 / G32R502）调试器支持情况：

- Geehy-Link（WinUSB）、DAP Link（固件版本为 CMSIS-DAP V2 及以上）
- J-Link V12（J-Link V7.94g 及以上）
- Ulink Pro

4.2 IDE 版本

请确保使用 Keil MDK V5.40 或更高版本的 IDE。

注意：在 Device 选用 Cortex-M52 时，MDK 5.43 以下可能无法使用 F12（转到定义）；需要源码跳转时请使用 MDK 5.43 或更高版本。MDK 5.43.1 上部分 16 位寄存器字段显示可能异常，可暂用 MDK 5.40 或改用 Memory 窗口核对。

4.3 工程操作

本节说明在 Keil MDK 中安装 Pack、打开与建立工程、编译下载及调试等相关操作。

注意：

（1） 推荐使用 Keil MDK**5.43** 或更高版本（Cortex-M52 / F12 等能力）。

（2） 下文截图与逐步操作以 v5.40 界面为例，菜单路径与 5.43 等价。

（3） 5.43.1 上部分 16 位寄存器字段显示可能异常，可暂用 5.40 或改用 Memory 窗口核对。

4.4 安装 Pack 支持

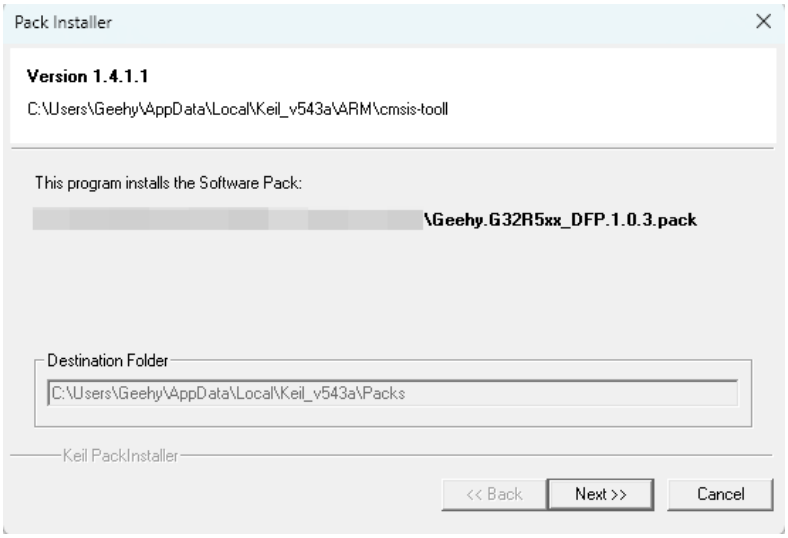
你可以选择以下任一方法进行安装：

直接安装

1. 找到 SDK 中 package 目录下的文件 “Geehy.G32R5xx_DFP.x.x.x.pack”。
2. 双击该文件，在弹出的安装界面（如图 1）中按照指示进行安装。

说明：以下界面以 G32R501 例程（目标名常见为 g32r501）为例。开发 G32R502 时，请在 Device / Target 中选择对应 G32R502 型号，将路径中的 g32r501 换为 g32r502，MDK 调试初始化文件使用 r502_dbg.ini。

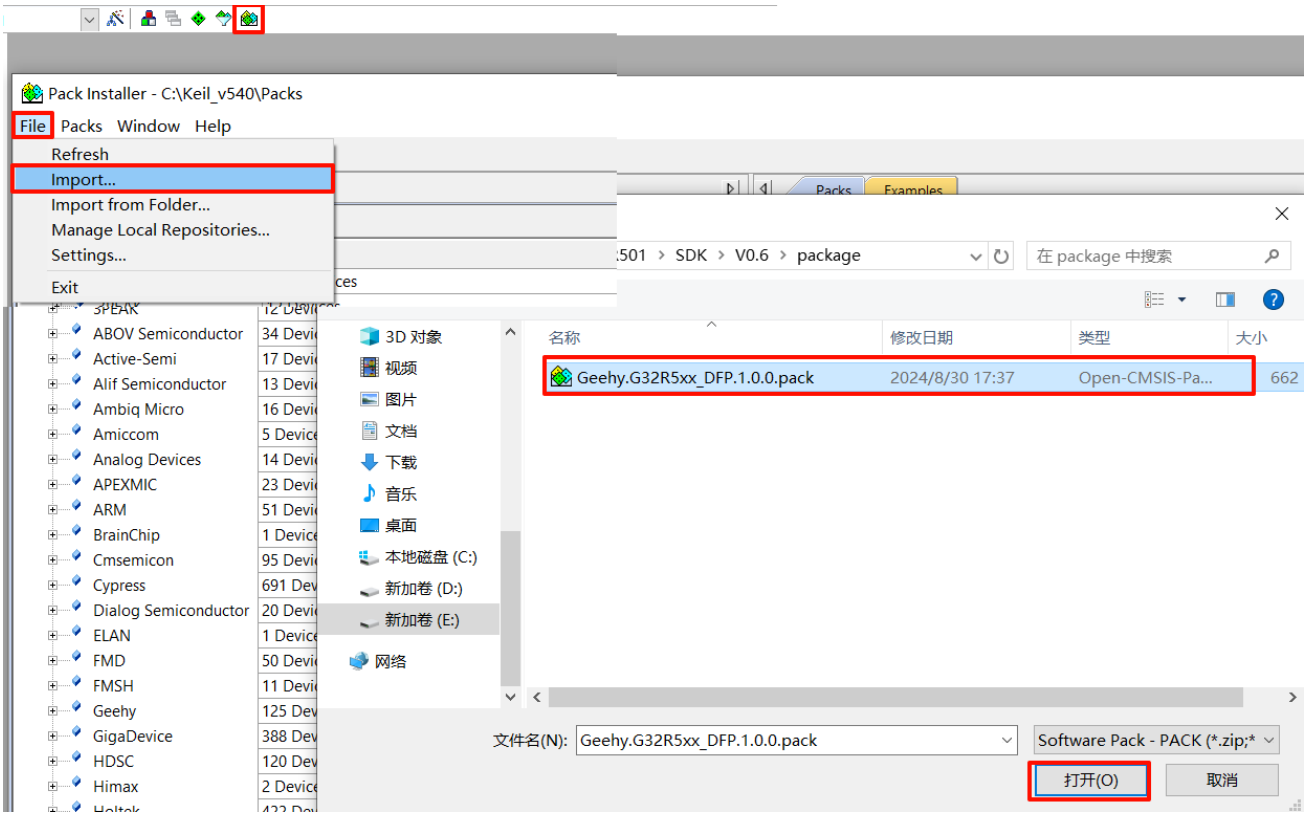
图 1 Geehy.G32R5xxDFP.x.x.x.pack 点击安装



使用 Pack Installer（如图 2）

1. 打开 Keil MDK v5.40。
2. 在菜单栏中，点击 “Project” → “Manage” → “Pack Installer”。
3. 在新窗口中，选择 “File” → “Import…”.
4. 在文件浏览器中，找到 SDK 中 package 目录下的文件 “Geehy.G32R5xx_DFP.x.x.x.pack”，然后进行选择。
5. 点击 “Open” 进行安装。

图 2 Pack Installer 导入 Geehy.G32R5xxDFP.x.x.x.pack



4.5 打开示例工程

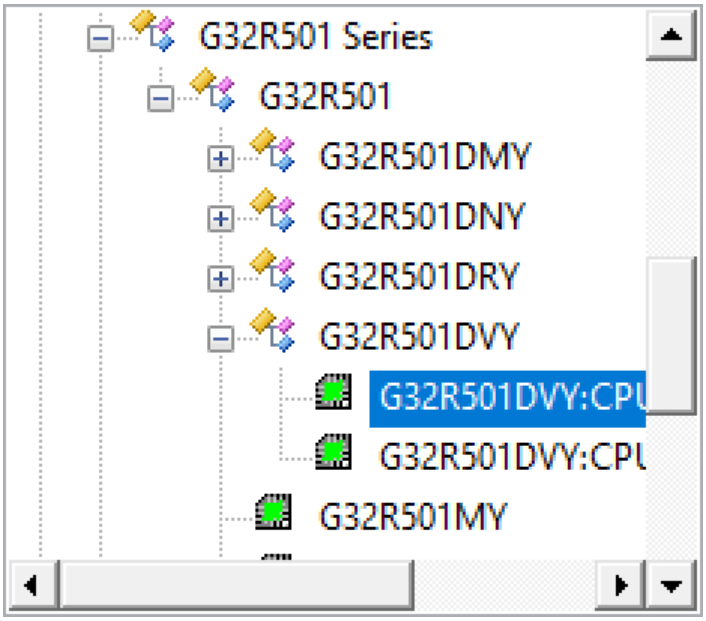
1. 启动 Keil MDK v5.40。
2. 点击菜单栏中的 “Project” → “Open Project”。
3. 浏览到你提供的 SDK 工程文件的路径，选择对应的 “.uvprojx” 文件，然后点击 “Open”。
4. 或是安装 Keil MDK v5.40 后，可直接点击工程文件 “.uvprojx” 打开即可。

注意：以上步骤请在完成 Pack 支持安装后进行，否则 Keil MDK 将提示芯片无法找到。

4.6 工程建立

1. 启动 Keil MDK v5.40。
2. 点击菜单栏中的 “Project” → “New uVision Project”。
3. 选择工程保存的路径，输入工程名称，点击 “Save”。
4. 在弹出的对话框中，选择合适的 G32R5xx 系列微控制器型号，然后点击 “OK”。

图 3 选择微控制器



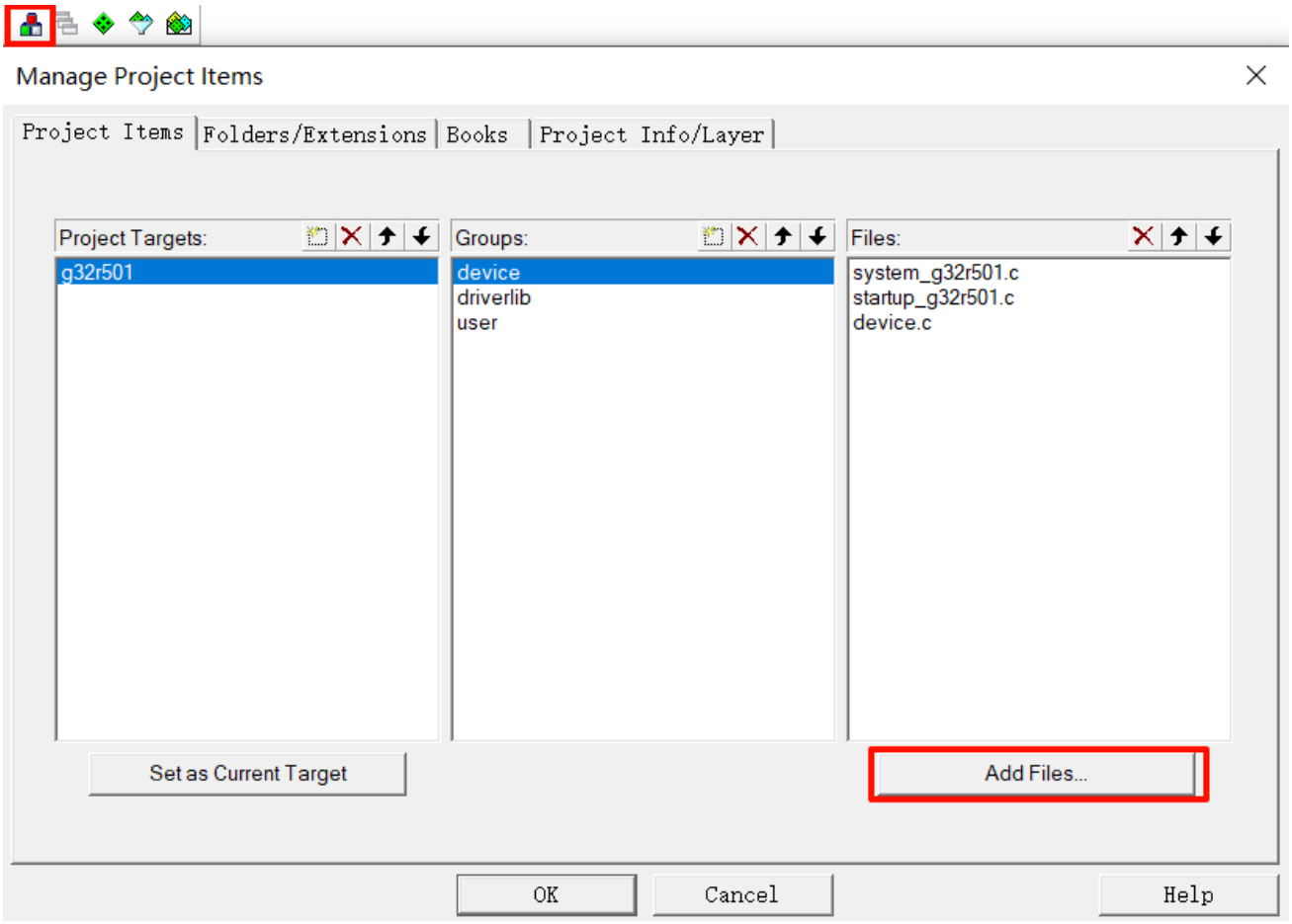
4.7 文件导入

1. 确保你的新工程已经在 Project 窗口中打开。
2. 点击“File Extensions，Books and Environment…”。

图 4 File Extensions

3. 点击“Source Group 1”或你希望添加文件的文件夹，选择“Add Files…”。
4. 在弹出的文件浏览器中，找到并选择要导入的源文件（如 .c 和 .h 文件），然后点击“Add”。

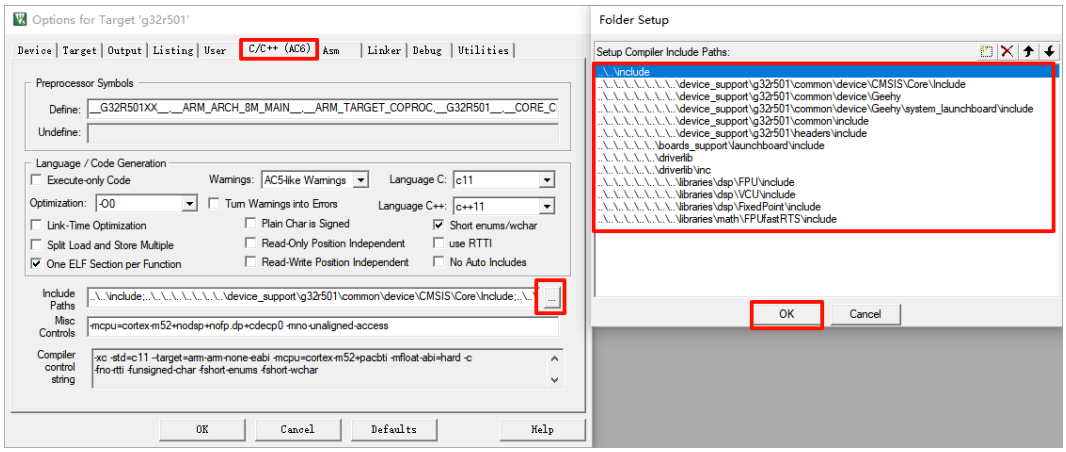
图 5 添加源文件



5. 如需新建源文件，右键点击 “Source Group 1”，选择 “Add New Item”；输入文件名，选择文件类型（如 C 文件或汇编文件），然后点击 “Add”。

6. 在菜单中选择 “Project” → “Options”，检查 “C/C++” 选项卡下的 “Include Paths”，确保添加了所有必要的库文件路径（如图 6）。

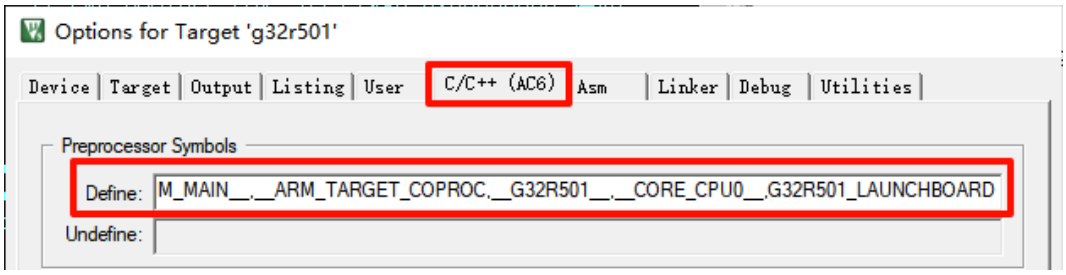
图 6 添加头文件路径



4.8 配置宏定义

在“Project”->“Options”中，选择“C/C++”选项卡，添加所需的宏定义，并进行相应的编译控制配置。

图 7 配置宏定义



4.9 编译命令控制

在 Keil MDK 中，可以通过项目属性中的“Options for Target”窗口下的“C/C++（AC6）”选项卡下的“Misc Controls”来控制编译命令。在“C/C++（AC6）”选项卡下，用户可以设置预处理器指令、编译器标志、不同的精度编译控制等内容时，可参考表格 2 的编译器命令支持情况，更多相关内容可参考 MDK 中的帮助文档。

表格 2 编译器命令

特性/选项	Scalar FP half-precision	Scalar FP single-precision	Scalar FP double-precision	MVE integer	MVE FP half-precision	Custom Datapath Extension（CDE）cp0
cortex-m52	Included	Included	Not included	Not included	Included	Not included
cortex-m52+ nomve	Included	Not included	Included	Not included	Included	Not included
cortex-m52+ nomve.fp+ nofp.dp	Included	Not included	Included	Not included	Included	Not included
cortex-m52+ nomve+ nofp.dp	Not included	Not included	Included	Not included	Included	Not included
cortex-m52+ nomve.fp+ nofp	Included	Not included	Not included	Not included	Included	Not included
cortex-m52+ nopacbti	Included	Included	Not included	Not included	Not included	Not included

特性/选项	Scalar FP half-precision	Scalar FP single-precision	Scalar FP double-precision	MVE integer	MVE FP half-precision	Custom Datapath Extension（CDE）cp0
cortex-m52+ nomve+ nopacbti	Included	Not included	Not included	Not included	Not included	Not included
cortex-m52+ nomve.fp+ nofp.dp+ nopacbti	Included	Not included	Not included	Not included	Not included	Not included
cortex-m52+ nomve+ nofp.dp+ nopacbti	Not included	Not included	Not included	Not included	Not included	Not included
cortex-m52+ nomve.fp+ nofp+ nopacbti	Not included	Not included	Included	Not included	Not included	Not included
cortex-m52+ cdecp0	Not included	Not included	Not included	Not included	Not included	Included

图 8 MDK 帮助文档

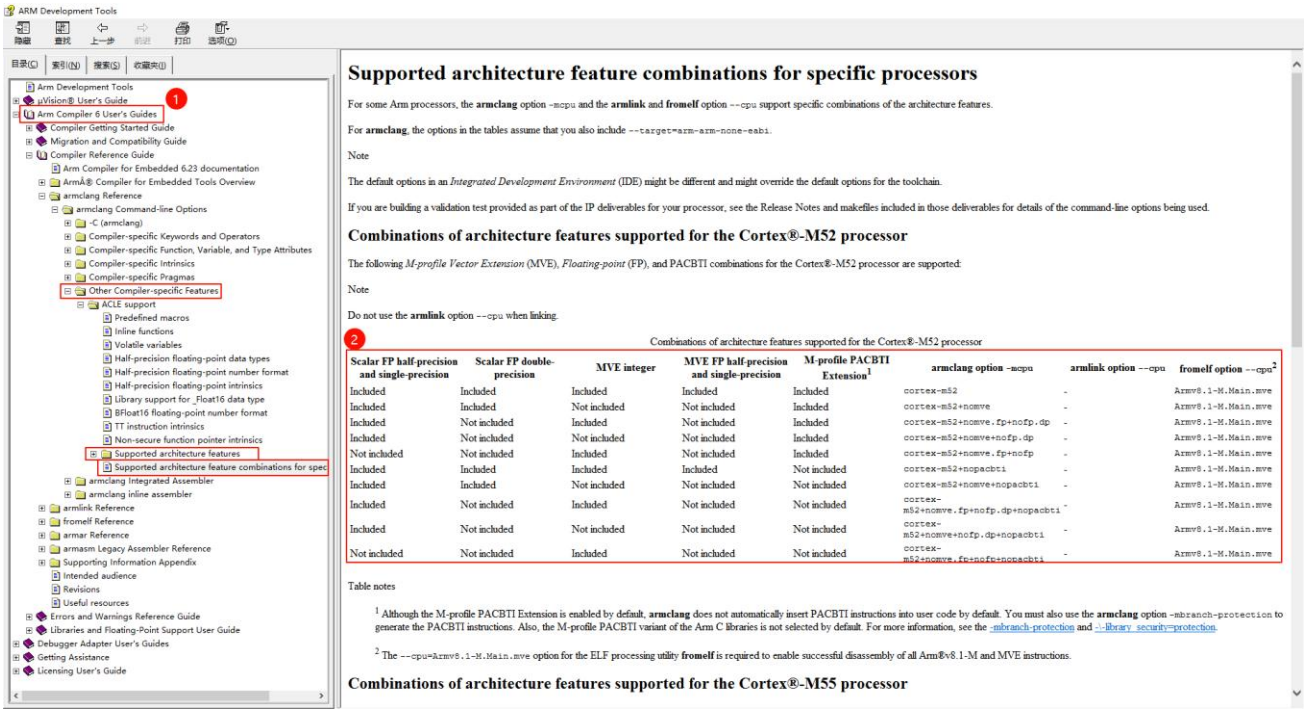
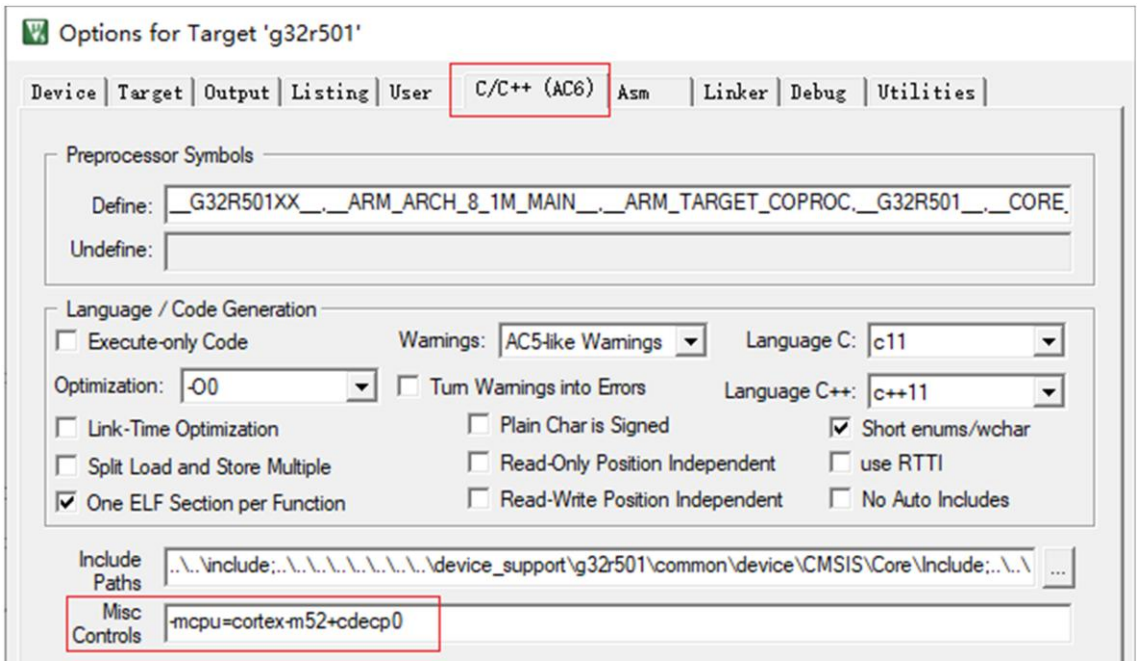


图 9 编译命令控制

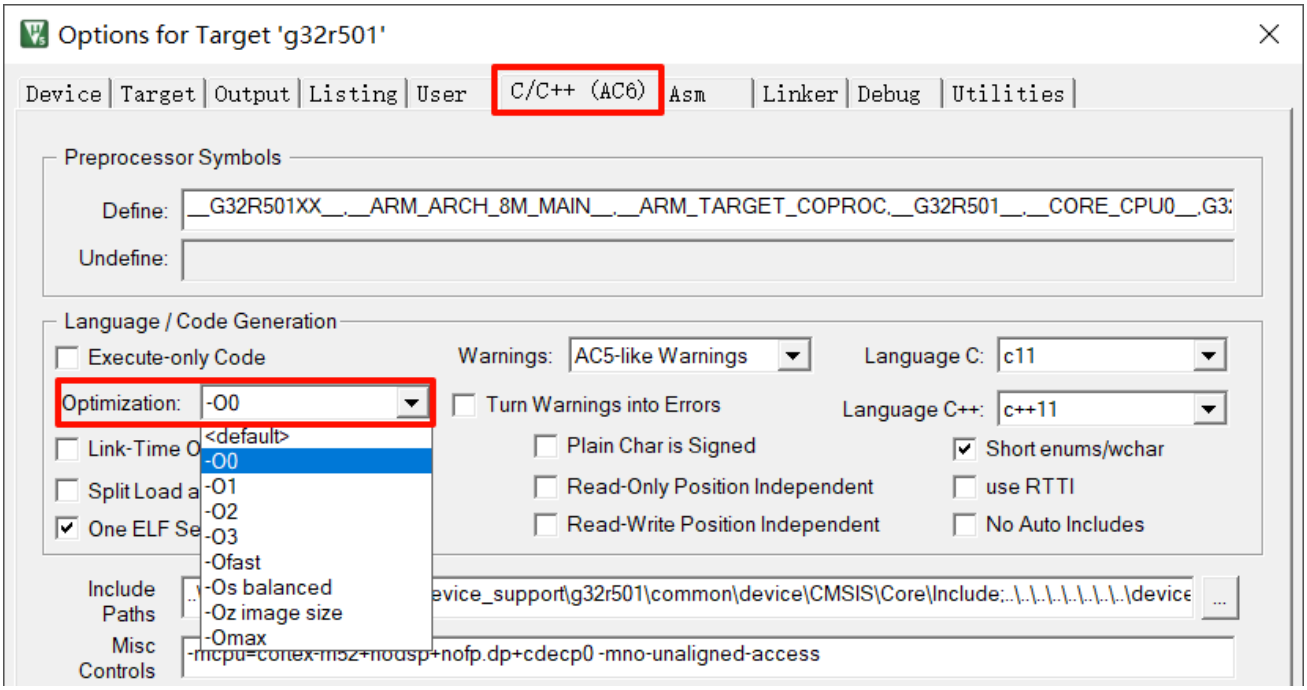


4.10 编译优化等级设置

Keil MDK 提供了多种优化等级设置，可以在全局、单文件和单函数级别进行调整：

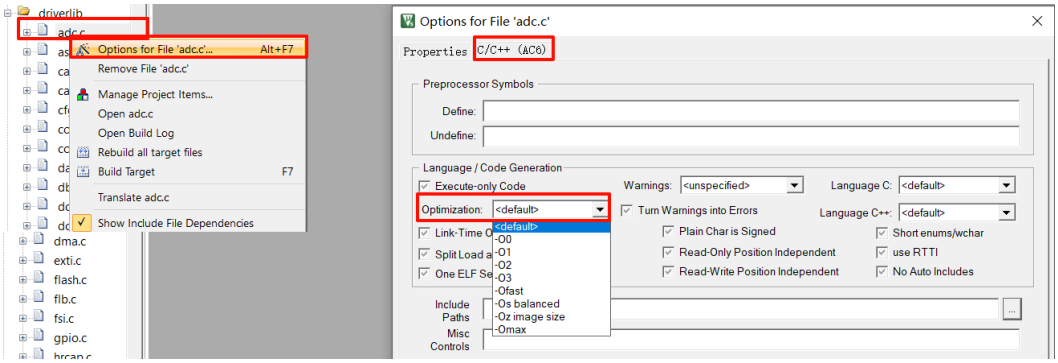
1. 全局优化等级设置：在“Options for Target”窗口中，选择“C/C++”选项卡，然后在“Optimization”下拉菜单中选择合适的优化等级。

图 10 全局优化等级设置



2. 单文件优化等级设置：右键点击特定的源文件，选择“Options for File...”，然后在“C/C++”选项卡中设置优化等级。

图 11 单文件优化等级设置



3. 单函数优化等级设置：可以在函数前使用特定的编译器指令进行优化设置，例如使用__attribute__（（optnone））声明不优化或使用__attribute__（（optimize（"O2"））））进行优化。

4.11 程序编译

1. 在菜单栏中，点击 “Project” → “Build Target”（或直接点击工具栏上的 “Build” 按钮，或按下 “F7”）。

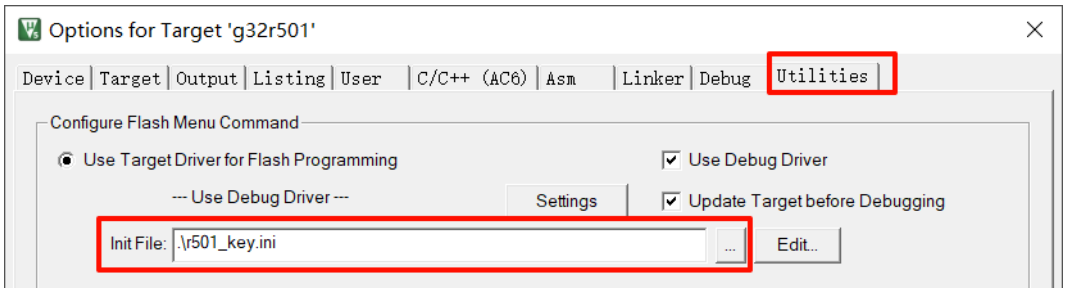
图 12 编译程序

2. 等待编译过程完成，查看输出窗口中是否有错误或警告信息；如有错误，根据提示进行相应的修改。

4.12 程序下载

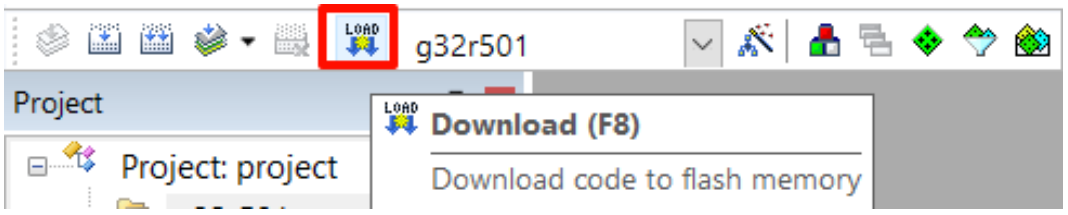
1. 点击菜单栏中的 “Project” → “Options”。
2. 在弹出的对话框中，选择 “Debug” 标签。
3. 在 “Use” 下拉菜单中，选择合适的调试器（例如 Geehy-Link、J-Link 等；J-Link 的下载与 Devices 注册见本章后续相关小节）。
4. 打开 **Options for Target**；在 “Utilities” 选项卡中，选择合适的下载工具。
5. 点击 “Settings” 按钮，加载特定的下载脚本文件（调试器选择为 Geehy-Link 时）。

图 13 选择下载脚本



6. 在工具栏上，点击 “Download” 按钮（小箭头图标）或在菜单中选择 “Flash” → “Download”（或按下 “F8”）。

图 14 下载程序



7. 确保连接了目标设备，等待下载完成，查看输出窗口确认下载状态。

4.13 程序 Debug

4.13.1 J-Link Debug

4.13.1.1 安装 J-Link 器件支持（须先完成）

SEGGER J-Link 默认不认识 G32R501 / G32R502。使用 J-Link 调试前，必须把 SDK 中的器件定义、连接脚本和 Flash 算法安装到本机 JLinkDevices 目录。安装完成后，J-Link 工具才能在器件列表中选到对应型号，一般**不必**再把 “.jlinkscript” 拷进每个工程。

软件与硬件要求

- SEGGER J-Link 软件须支持 Cortex-M52（参考版本：J-Link V7.94g 或更高；建议使用更新版本）。
- 建议使用 J-Link 探头 V12 或更高版本。

需要准备的文件

- G32R501：从 “device_support/g32r501/common/Jlink/” 取 “G32R501.xml”、“Geehy_G32R501_ConnectCore.jlinkscript”；从 SDK “package/FLM/” 取 “G32R5xx_Program_Algorithm.FLM”（OTP 调试另取 “G32R5xx_OTP.FLM”）。
- G32R502：从 “device_support/g32r502/common/jlink/” 取 “G32R502.xml”、“Geehy_G32R502_ConnectCore.jlinkscript”；从 SDK “package/FLM/” 取 “G32R502_Program_Algorithm.FLM”（OTP 调试另取 “G32R502_OTP.FLM”）。

以上 FLM 以 SDK “package/FLM” 为准；若 DFP pack 内含同名算法文件，内容应与之保持一致。将对应三个文件（xml、jlinkscript、主 Flash FLM）复制到同一 J-Link Devices 安装目录。

Windows 安装步骤

1. 在资源管理器地址栏输入 “%APPDATA%\SEGGER\JLinkDevices\” 并回车（若不存在则新建）。
2. 在其下新建目录 “Geehy\G32R501\” 或 “Geehy\G32R502\”（按实际器件二选一或两者都装）。
3. 将对应三个文件复制到该目录。完整路径示例：

- G32R501：“C:\Users\<你的用户名>\AppData\Roaming\SEGGER\JLinkDevices\Geehy\G32R501\”
- G32R502：“C:\Users\<你的用户名>\AppData\Roaming\SEGGER\JLinkDevices\Geehy\G32R502\”

4. 关闭并重新打开所有 J-Link 相关程序（J-Flash、Ozone、IDE 调试会话、JLink.exe 等），以便重新加载器件库。

Linux / macOS 安装步骤

将同样的三个文件复制到：

- G32R501："$HOME/.config/SEGGER/JLinkDevices/Geehy/G32R501/"
- G32R502："$HOME/.config/SEGGER/JLinkDevices/Geehy/G32R502/"

安装后如何确认

1. 打开 J-Link Commander（"JLink.exe"）或 J-Flash。
2. 输入 "device ?" 查看器件列表。
3. 确认厂商 Geehy 下出现对应器件名。常见名称如下：

- G32R501 单核示例："G32R501VY" / "MY" / "RY" / "NY" / "RC"
- G32R501 双核示例："G32R501DVY" / "DMY" / "DRY" / "DNY"
- G32R502："G32R502MC" / "KC" / "CC" / "RC"

4. 连接目标板，做一次擦除或编程，确认 DCS 解锁与 Flash 编程正常。

注意：若 OTP 中 Zone1 / Zone2 的 CSM 密码已改为非出厂默认值，须在复制到 JLinkDevices **之前**，编辑对应 "Geehy_G32R501_ConnectCore.jlinkscript" 或 "Geehy_G32R502_ConnectCore.jlinkscript" 文件顶部的 "Z1_CSM_Buff" / "Z2_CSM_Buff" 数组，再安装。

4.13.1.2 在 Keil MDK 中配置 J-Link

完成上一节安装并重启工具后，按下列步骤配置工程：

1. 打开项目选项：选择 **Options for Target**。
2. 在 **Device** 中选择与板卡一致的 G32R501 / G32R502 器件型号（名称须与已安装的 J-Link Devices 一致，例如 G32R501DVY、G32R502MC）。
3. 在 **Debug** 选项卡中选择 **J-Link / Cortex-M** 调试器。
4. 确认工程 MDK 目录下已有器件对应的调试初始化文件，并在 **Debug → Initialization File** 中引用：

- G32R501："r501_dbg.ini"
- G32R502："r502_dbg.ini"

该 ini 用于 Keil 侧 MSP/PC、DCS 等初始化，与 JLinkDevices 中的连接脚本相互独立，两者都需要正确。

1. 本机已按上一节安装器件包后，工程内**通常不必**再单独放置 ".jlinkscript"；"G32R501.xml" /

“G32R502.xml” 会自动绑定 ConnectCore 脚本。

2. 使用 J-Link Debug 时，请清除 **Utilities** 选项卡中额外的下载脚本 Init File（若有旧配置残留）。

3. 编译并下载完成后，点击 **Debug** 开始调试。可设置断点、查看变量、单步执行。

说明：旧版手册要求把 “JLinkSettings.JLinkScript” 拷进工程的做法已过时，现由本机 JLinkDevices 器件包统一提供连接脚本。

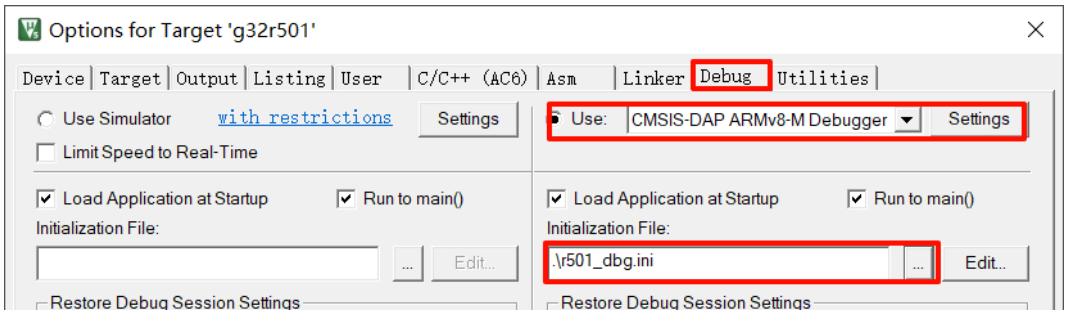
4.13.2 GEEHY LINK Debug

1. 打开项目选项：选择 “Options for Target”。

2. 在 “Debug” 选项卡中，选择合适的调试器。

3. 在 “Settings” 菜单中，点击 “Load” 按钮，加载特定的 Debug 脚本文件。

图 15 选择 CMSIS-DAP 调试器及 Debug 脚本



4. 下载完成后，点击 “Debug” 按钮开始调试程序；在调试模式下，可以设置断点、查看变量、单步执行等操作。

4.14 C 语言兼容性

在进行工具链迁移时，C 语言代码的兼容性是需要重点关注的问题。不同编译器在文件格式支持、内置函数、内存操作和数据类型等方面可能存在差异。为了确保代码在新的编译环境中能够正确编译和运行，需要对现有代码进行必要的调整和优化。

4.14.1 文件格式支持

支持.c/.h 文件。

4.14.2 浮点类型数据

根据 ISO/IEC C 标准规范中的规定：未加后缀的浮点常量类型为 double，后缀为 f 或 F 的浮点常量类型为 float，后缀为 l 或 L 的浮点常量类型为 long double。建议有单精度需求的浮点常量后缀加 f。

4.14.3 sizeof 使用

不同编译器和架构对数据类型的大小可能存在差异，因此需要特别注意 sizeof 操作符在不同平台上的结果。常见数据类型在 G32R501 和 Txx320F28004x 上的大小对比见表格 3。

表格 3 不同数据类型在 G32R501 和 Txx320F28004x 上的大小对比

数据类型	G32R501	Txx320F28004x
char	1	1
short	2	1
int	4	1
long	4	2
long long	8	4
float	4	2
double	8	4
long double	8	4
void	4	2
int8_t	1	1
uint8_t	1	1
int16_t	2	1
uint16_t	2	1
int32_t	4	2
uint32_t	4	2
int64_t	8	4
uint64_t	8	4

4.15 汇编兼容性

不同目标平台的指令集、汇编语法可能存在显著差异，需要对现有的汇编代码进行重新编写和适配，以确保在新的平台上能够正确运行。

4.15.1 文件格式支持

AC6 基于 LLVM 和 Clang 技术，主要使用 GNU 风格的汇编语法。AC6 支持的汇编文件格式：

.s 文件：GNU 风格的汇编文件格式。

与此同时，AC6 支持在 C 函数中使用内联汇编。

4.15.2 汇编编码格式要求

单一汇编文件：这里是一个简单的汇编代码示例，定义了一个汇编函数 add，用于将两个整数相加。文件“add.s”的内容如下：

```
.syntax unified
.global add
.type add, %function
add:
@ Function entry
@ Parameters: r0 and r1
@ Return value: r0
adds r0, r0, r1   @ Add r0 and r1, store result in r0
bx lr             @ Return to calling function
.end
```

C 函数中使用内联汇编：以下是一个在 C 函数中使用内联汇编的示例，定义了一个内联汇编函数 add_inline，用于将两个整数相加：

```
// Inline assembly function
static inline int add_inline(int a, int b){
int result;
__asm volatile(
"adds %0, %1, %2\n"
: "=r"(result)       // Output operand
: "r"(a), "r"(b)    // Input operands
: "cc"                 // Clobbered registers
);
return result;
}
```

4.16 链接脚本文件

链接脚本文件用于定义程序的内存布局和段分配。在迁移过程中，需要根据目标平台和开发环境的不同，使用相应格式的链接脚本文件。G32R5 系列 MCU 在 MDK 开发环境下使用“.sct”格式的链接脚本文件，遵循 Arm 公司的规范。

链接脚本文件的差异

文件格式：

- G32R5 系列 MCU 使用“.sct”格式的链接脚本文件，遵循 Arm 公司的规范。
- Txx320F28004x 使用“.CMD”格式的链接脚本文件，遵循其公司的规范。

内存布局和段分配：

- G32R5 系列 MCU 的“.sct”文件通过定义加载区域（Load Region）和执行区域（Execution Region）来安排内存分配。
- Txx320F28004x 的“.CMD”文件通过定义内存段（MEMORY）和段分配（SECTIONS）来安排内存分配。

4.17 RAM 运行

Txx320F28004x 编译器支持 Progma 语法，告诉编译器如果修改一个特定函数、目标文件或者一段代码的属性，例如 CODE_SECTION，就是为某一个函数分配 Section，使用方式如下：

```
# pragma CODE_SECTION(funcA, "codeA")
```

- G32R5 系列 MCU 实现同样的功能，可使用下面的语句：

```
__attribute__((section("xxx")))
```

其中，xxx 表示将某一函数或者全局变量指定到的 SECTION 名。使用的时候，可以参考如下格式：

```
__attribute__((section("itcm.ramfunc"))) void SysCtl_delay(uint32_t count){}
```

只需要在函数和变量定义的时候进行属性指定即可。

注意：使用 “__attribute__((section("itcm.ramfunc")))” 时需要 .sct（链接脚本文件）中有 “itcm.ramfunc” 字段的声明：“.ANY (itcm.ramfunc)”

5 IAR EW for Arm 开发工具链

5.1 调试器支持

请参考章节 4.1。

5.2 IDE 版本

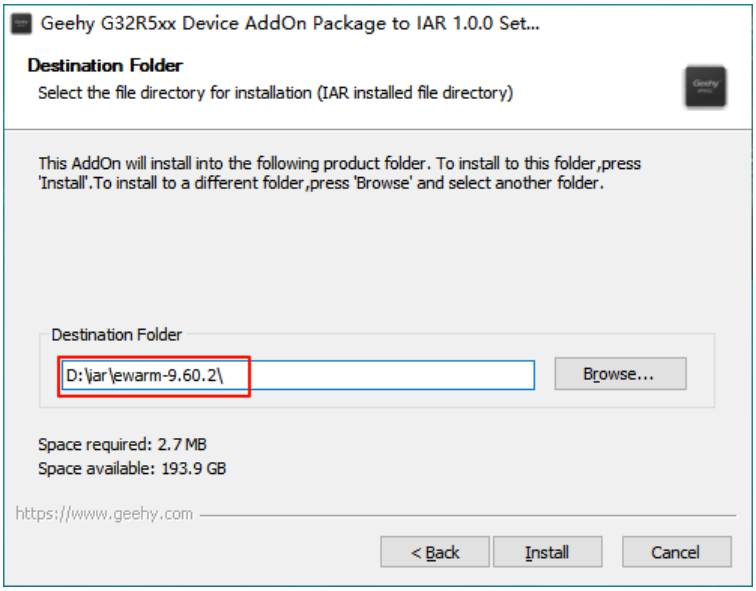
- G32R501：请使用 IAR Embedded Workbench for Arm**9.60.2 或更高**版本。
- **G32R502 系列：必须使用 IAR Embedded Workbench for Arm 9.70.4 或以上版本**；低于该版本不满足本系列开发要求。

5.3 安装芯片支持

在正式使用 IAR Embedded Workbench for Arm 开发 G32R5 系列 MCU 前请先进行芯片支持包的安装。芯片支持所在路径为：..\utilities\G32R5xx_AddOn\G32R5xx_AddOn_vx.x.x.exe

使用管理员权限打开 G32R5xx_AddOn_vx.x.x.exe，来到选择安装芯片支持的路径的界面，该路径为 IAR Embedded Workbench for Arm 的安装路径，如示例为：D:\iar\ewarm-9.60.2\。

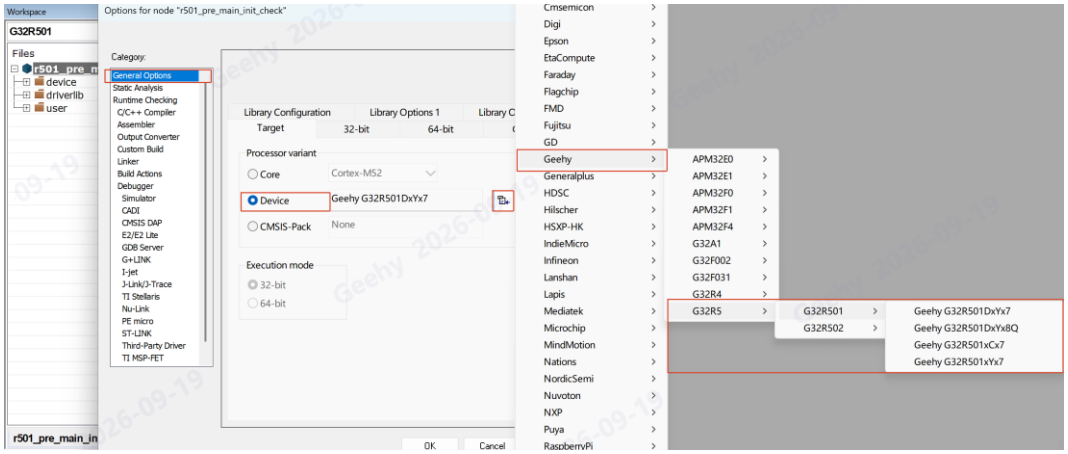
图 16 G32R5xxAddOnvx.x.x.exe 安装芯片支持



若软件无法获取电脑上的 IAR Embedded Workbench for Arm 安装路径，请手动选择**安装根目录**（不要选择安装目录下的 config 等子目录）。

选择正确的路径并添加完毕芯片支持后，打开 IAR Embedded Workbench for Arm，选择“新建工程”，在芯片选型选项卡即可看到 G32R5 系列 MCU 列表。

图 17 G32R5 系列 MCU 列表



5.4 工程操作

5.5 打开示例工程

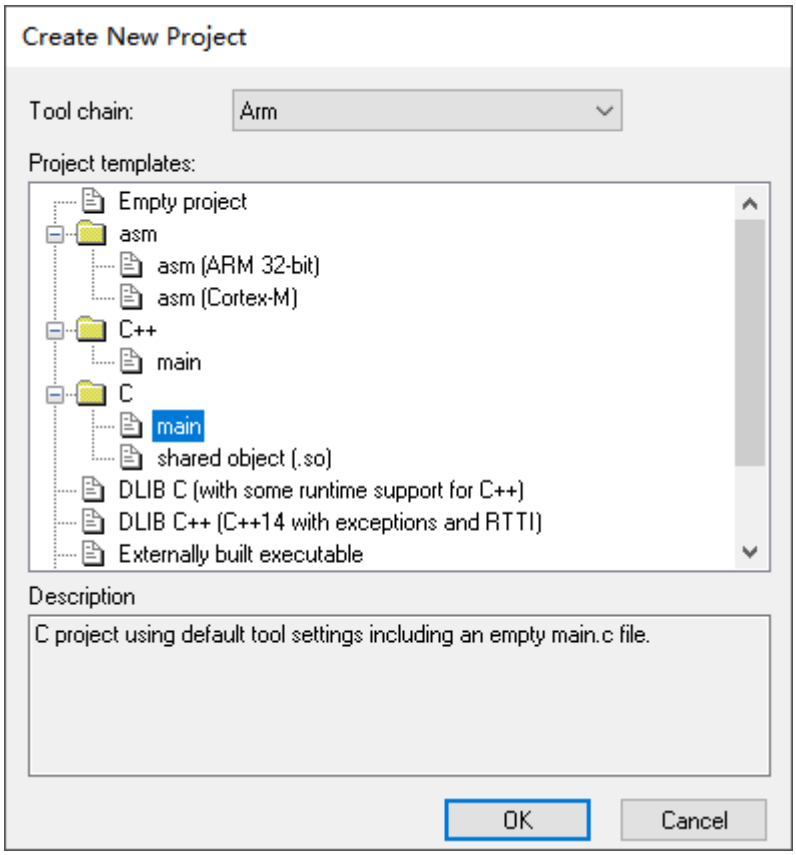
1. 启动 IAR Embedded Workbench for Arm 9.60.2。
2. 点击菜单栏中的 “File” → “Open Workspace …”。
3. 浏览到你提供的 SDK 工程文件的路径，选择对应的 .eww 文件，然后点击 “Open”。
4. 或是安装 IAR Embedded Workbench for Arm 9.60.2 后，可直接点击工程文件 “.eww” 打开即可。

注意：以上步骤请在完成章节 5.3 安装芯片支持后进行，否则 IAR Embedded Workbench for Arm 将提示芯片无法找到。

5.6 工程建立

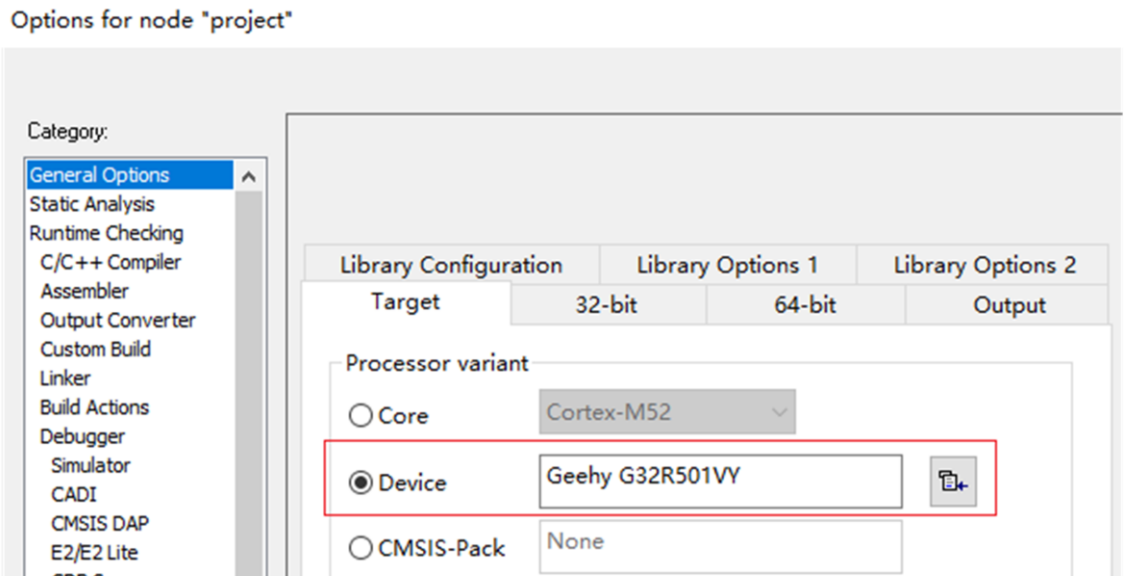
1. 启动 IAR Embedded Workbench for Arm 9.60.2。
2. 点击菜单栏中的 “Project” → “Create New Project”。
3. 选择 “Tool chain” 为 “Arm”。
4. 选择 “C” 下的 “main” 后点击 “OK”。
5. 选择工程保存的路径，输入工程名称，点击 “Save”。

图 18 Create New Project



6. 右键工程名，选择“Options…”。
7. 在“General Options”下的“Target”中选择“Device”。
8. 选择合适的 G32R5xx 系列微控制器型号，然后点击“OK”。

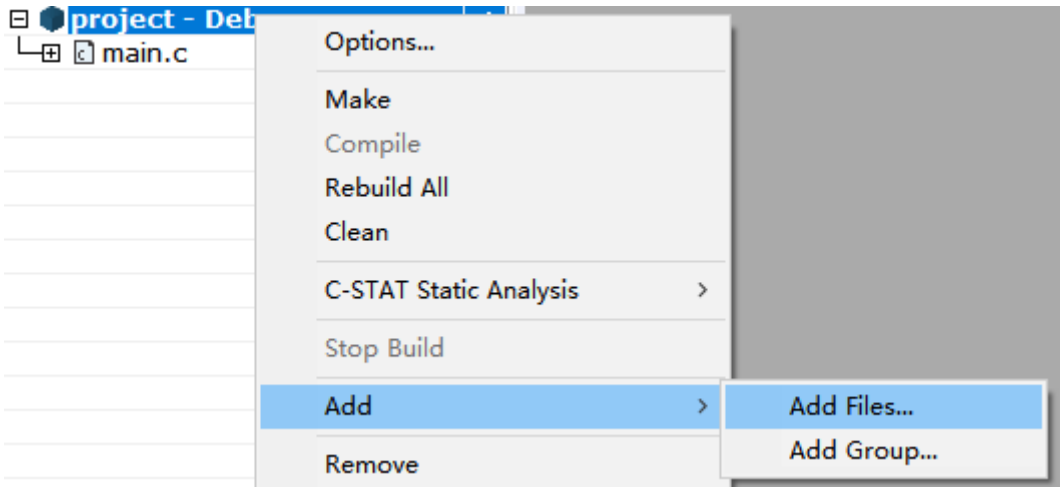
图 19 选择微控制器



5.7 文件导入

1. 确保你的新工程已经在 Project 窗口中打开。
2. 右键工程名，选择 “Add”，然后选择 “Add Files”。

图 20 添加源文件

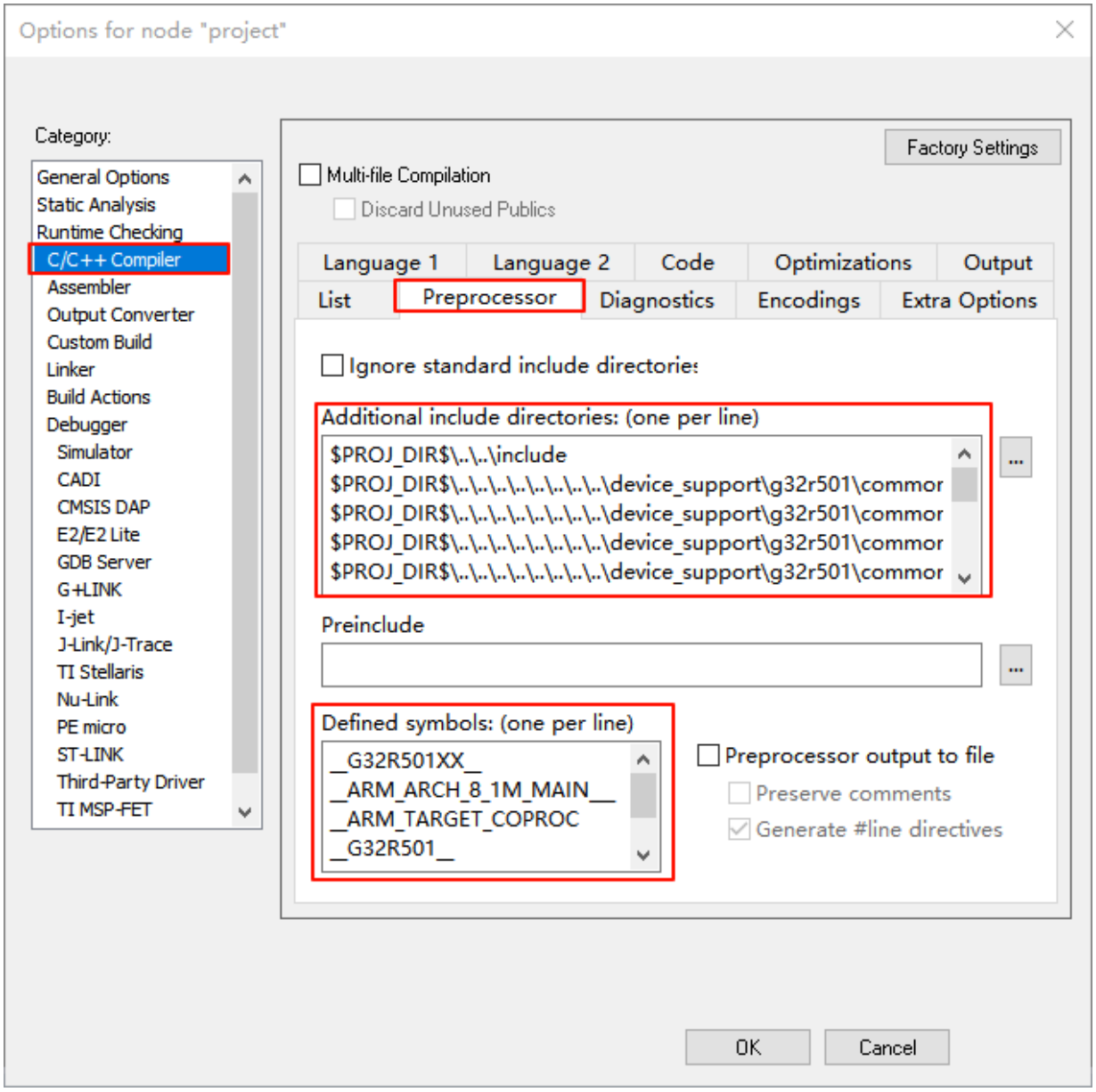


3. 在弹出的文件浏览器中，找到并选择要导入的源文件（如 .c 和 .h 文件），然后点击 “Add”。
4. 如需新建源文件，点击工具栏上的 “File”，选择 “New File”；在 IDE 中按下 “Ctrl+S” 保存文件，输入文件名并选择文件类型（如 C 文件或汇编文件），然后按上述步骤添加。

5.8 配置头文件路径与宏定义

1. 右键工程名，选择“Options…”。
2. 在“C/C++ Compiler”选项卡下选择“Preprocessor”。

图 21 配置头文件路径与宏定义



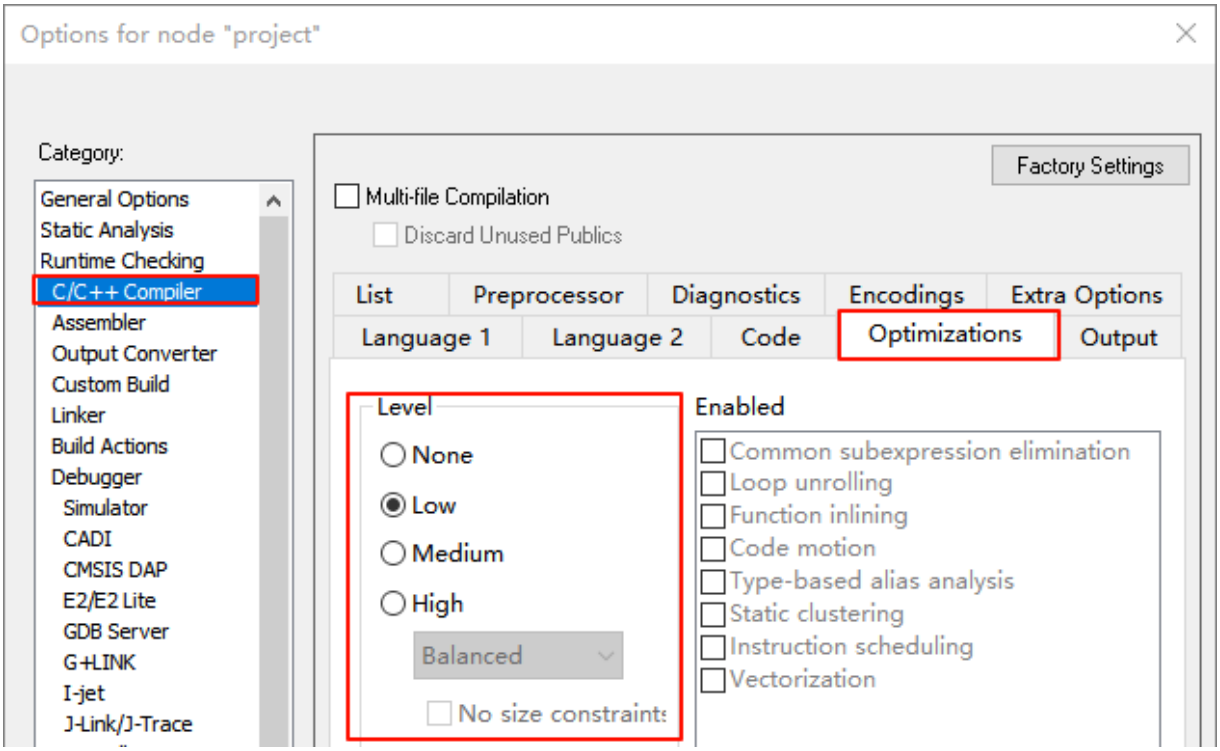
3. 在“Additional include directories:（one per line）”中添加所有必要的库文件路径。
4. 在 Defined symbols:（one per line）中添加所需的宏定义。

5.9 编译优化等级设置

IAR Embedded Workbench for Arm 提供了多种优化等级设置，可以在全局、单文件和单函数级别进行调整：

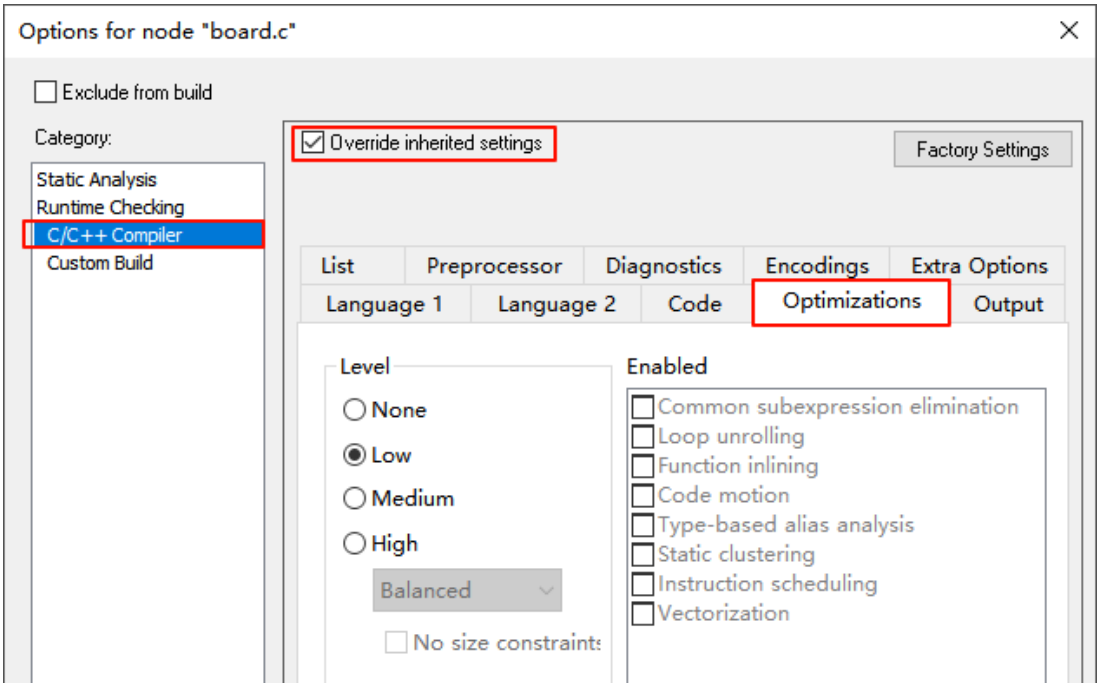
1. 全局优化等级设置：打开“C/C++ Compiler”选项卡，然后在“Optimizations”中选择合适的优化等级。

图 22 全局优化等级设置



2. 单文件优化等级设置：右键点击特定的源文件，选择“Options...”，勾选“Override inherited settings”后“Optimizations”中选择合适的优化等级。

图 23 单文件优化等级设置

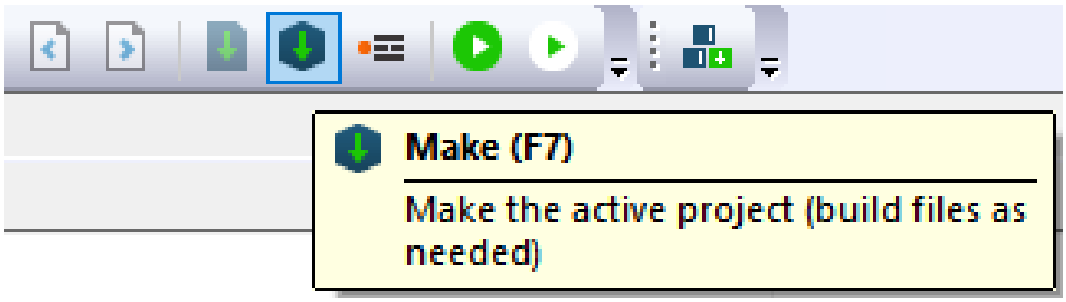


3. 单函数优化等级设置：可以在函数前使用特定的编译器指令进行优化设置，例如使用 _Pragma（"optimize=none"）声明不优化。

5.10 程序编译

1. 在菜单栏中，点击 “Project” → “Make”（或直接点击工具栏上的 “Make” 按钮，或按下 “F7”）。

图 24 编译程序



2. 等待编译过程完成，查看输出窗口中是否有错误或警告信息；如有错误，根据提示进行相应的修改。

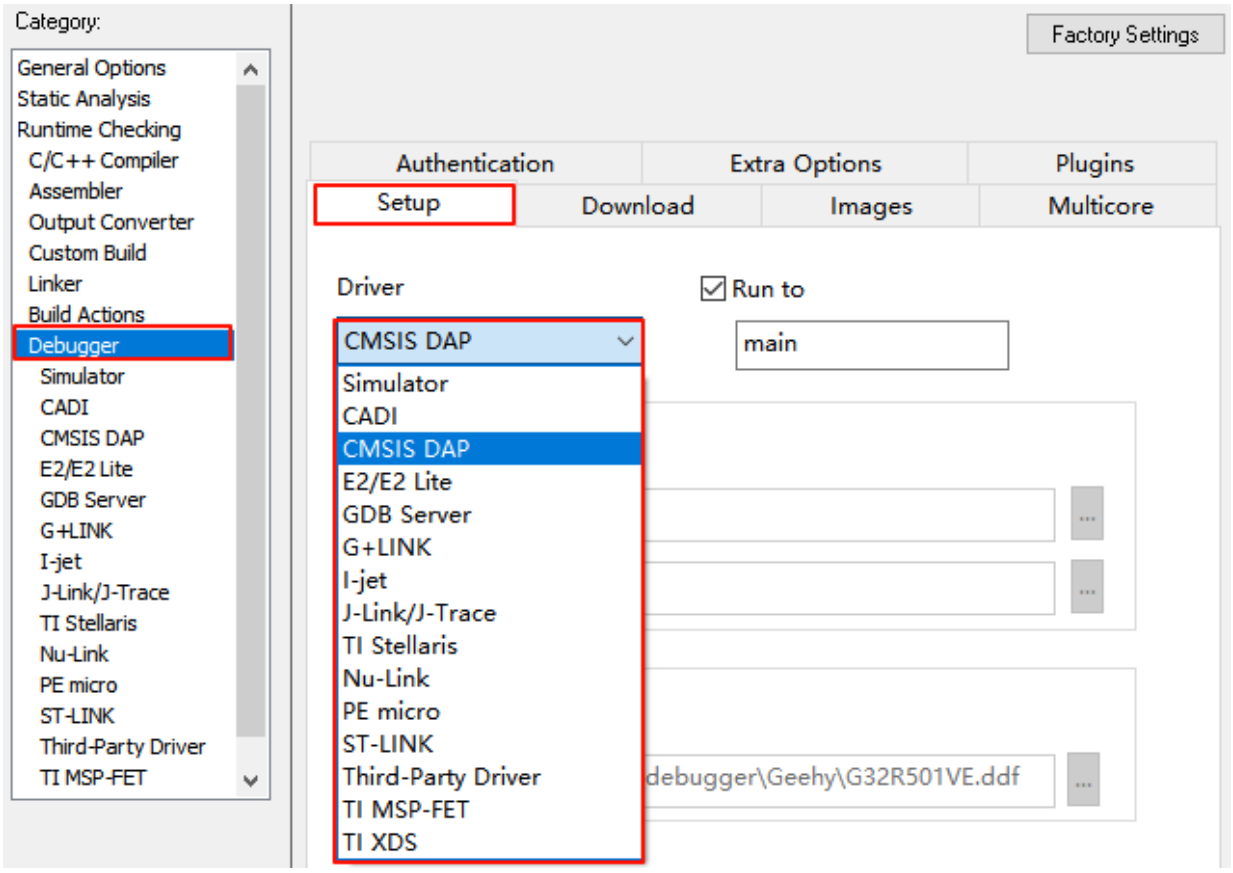
5.11 程序 Debug 与下载

1. 右键工程名，选择 “Options…”。

2. 在弹出对话框中，选择“Debugger”选项卡。

3. 在“Setup”标签下的“Driver”下拉菜单中，选择合适的调试器（例如 Geehy-Link（CMSIS DAP）、J-Link 等）。

图 25 选择调试器



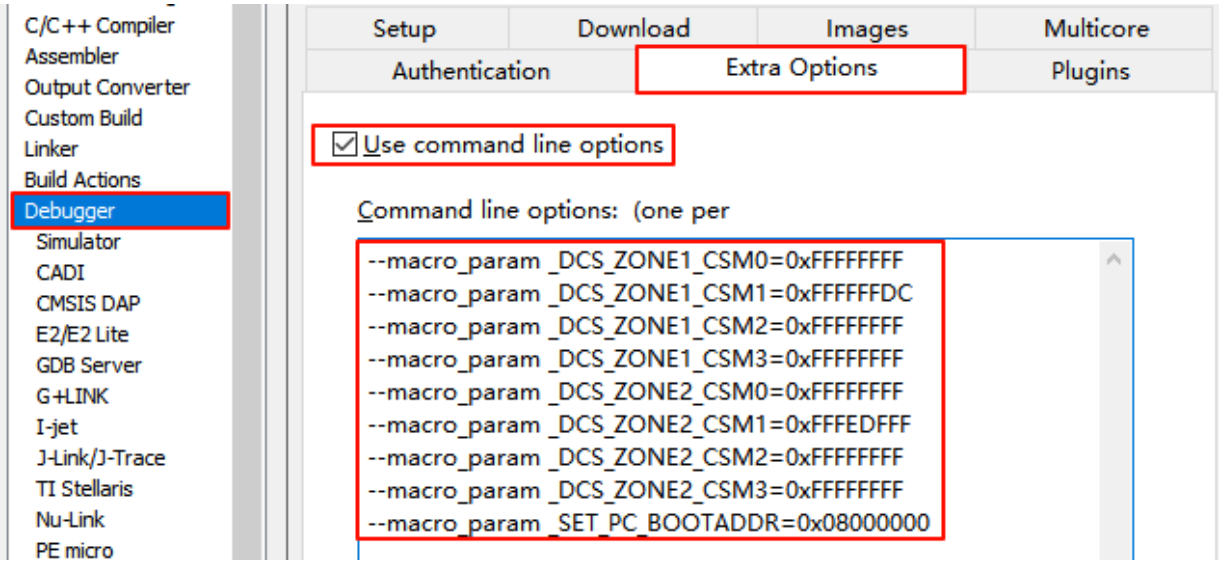
4. 再次右键工程名，选择“Options…”，进入“Debugger”选项卡。

5. 在“Extra Options”标签下勾选“Use command line options”，并在“Command line options”中添加 DCS 密钥与启动地址相关指令，例如：

```
--cdecp=0
--macro_param _DCS_ZONE1_CSM0=0xFFFFFFFF
--macro_param _DCS_ZONE1_CSM1=0xFFFFFFDC
--macro_param _DCS_ZONE1_CSM2=0xFFFFFFFF
--macro_param _DCS_ZONE1_CSM3=0xFFFFFFFF
--macro_param _DCS_ZONE2_CSM0=0xFFFFFFFF
--macro_param _DCS_ZONE2_CSM1=0xFFFEDFFF
--macro_param _DCS_ZONE2_CSM2=0xFFFFFFFF
--macro_param _DCS_ZONE2_CSM3=0xFFFFFFFF
--macro_param _SET_PC_BOOTADDR=0x08000000
```

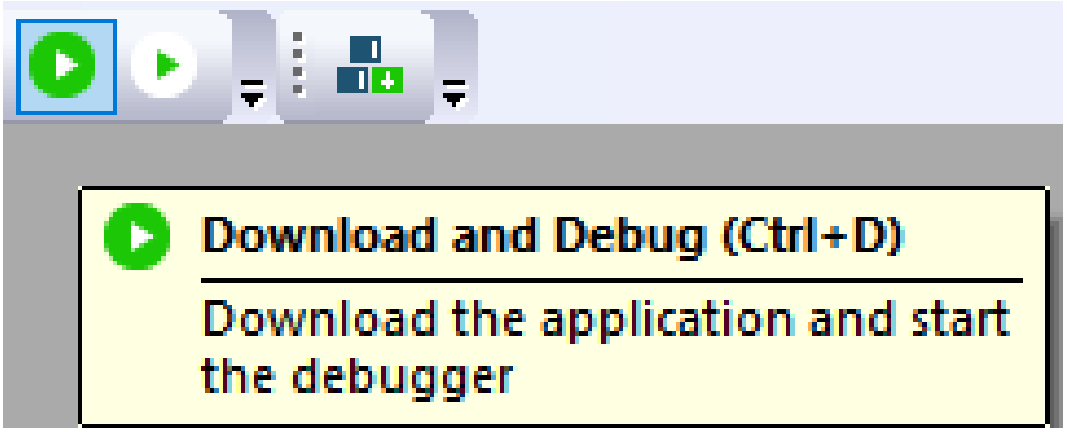
- “--cdecp=0”：开启 CDE 支持
- “--macro_param _DCS_ZONEx_CSMx”：DCS 密钥支持；若为默认密钥，可不设置
- “--macro_param _SET_PC_BOOTADDR”：设置启动地址；若为默认 0x08000000，可不设置

图 26 配置 Debug 指令



6. 在工具栏点击 “Project” → “Download”，按需选择 “Erase memory”、“Download active application” 或 “Download file…”。
7. 点击工具栏的 “Download and Debug” 按钮或按下 “Ctrl+D” 启动 Debug。

图 27 IAR Download and Debug



8. 在调试模式下，可以设置断点、查看变量、单步执行等操作。

调试问题

调试时 Memory 窗口中 Flash 内容不更新

原因：Flash 对应 Memory 区域在 ddf（device description file）文件中的 AccType 是 R，表示调试器对 Flash 只读，不能修改 Flash 的内容，所以调试器在调试过程中不会更新对应 Memory 区域的值。

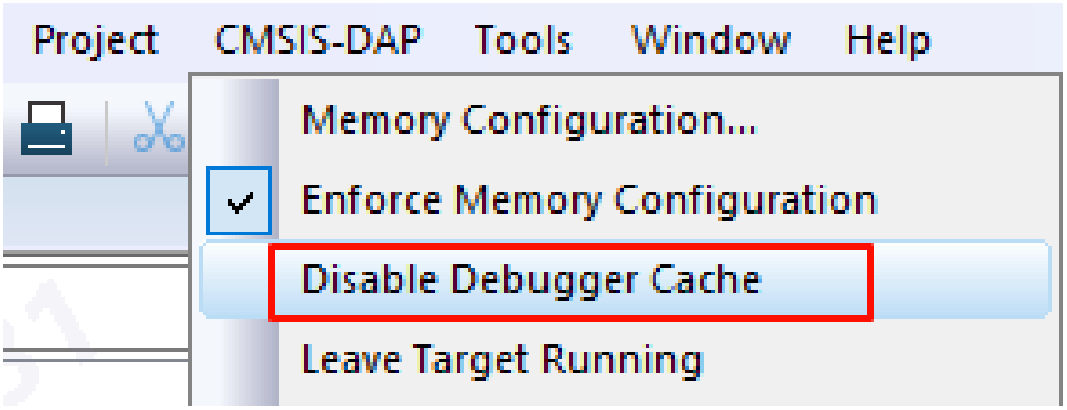
图 28 ddf 文件（节选）

```
[Memory]
;;          Name                 AdrSpace    StartAdr     EndAdr       AccType   Width
Memory =   Boot_ROM             Memory      0x10000000   0x1001FFFF   R
Memory =   Secure_ROM           Memory      0x10020000   0x1002FFFF   R
Memory =   ITCM_Flash           Memory      0x00100000   0x0019FFFF   R
Memory =   BusMatrix_Flash      Memory      0x08000000   0x0809FFFF   R
Memory =   CPU0_ITCM            Memory      0x00000000   0x0000BFFF   RW
Memory =   CPU1_ITCM            Memory      0x00000000   0x00001FFF   RW
Memory =   CPU0_DTCM            Memory      0x20000000   0x20003FFF   RW
```

解决方案：

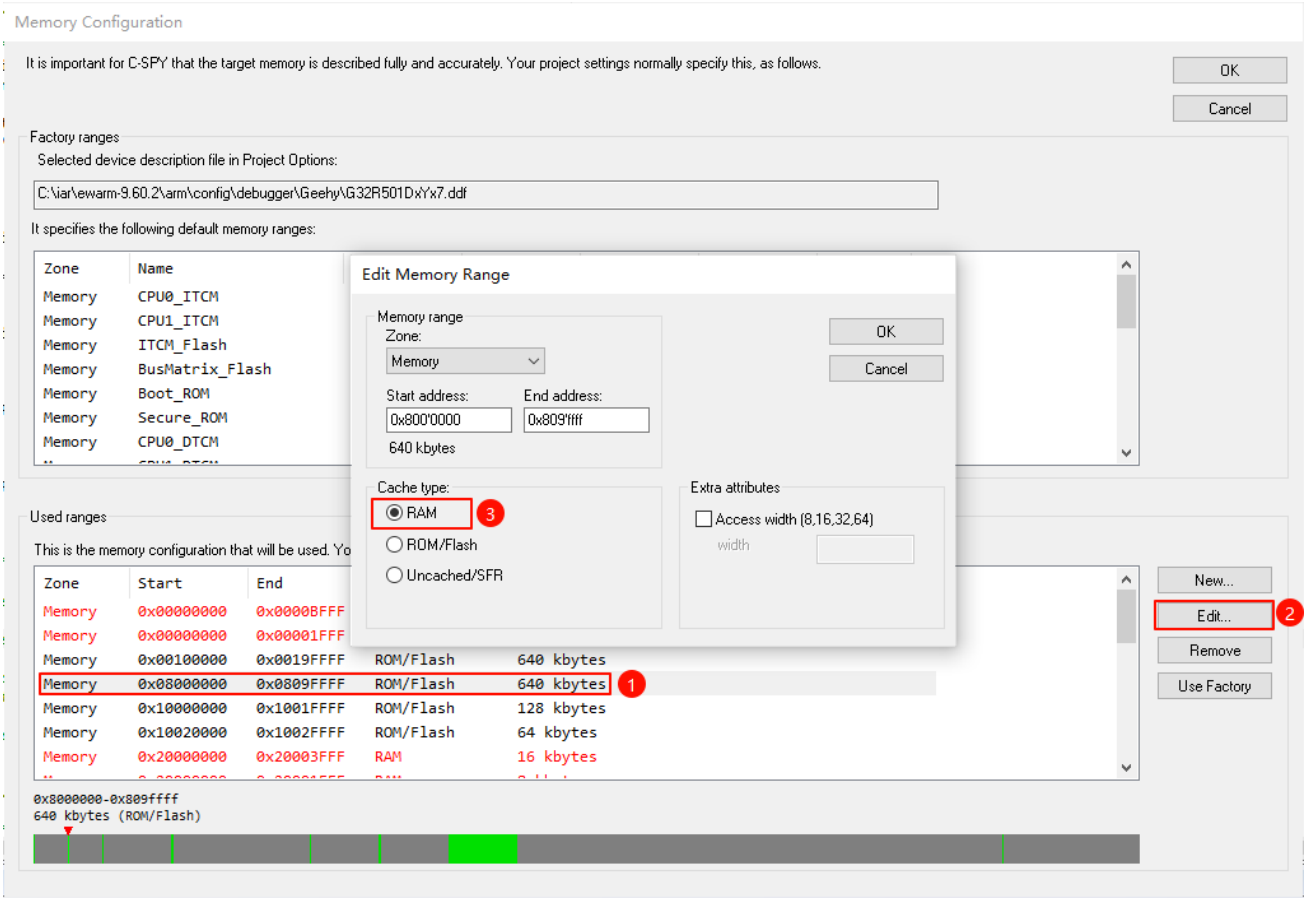
- 使能 Disable Debugger Cache

图 29 Disable Debugger Cache



- 修改 Flash 对应 Memory 区域的 Cache Type 为 RAM。点击 Memory Confoguration，进入 Memory Confoguration 窗口，修改 Cache Type。

图 30 修改 Cache Type



- 修改 ddf 文件中 Flash 对应 Memory 区域的 AccType 为 RW

图 31 修改 ddf 文件 AccType

```
[Memory]
;;          Name                AdrSpace    StartAdr     EndAdr       AccType   Width
Memory =   Boot_ROM            Memory      0x10000000   0x1001FFFF   R
Memory =   Secure_ROM          Memory      0x10020000   0x1002FFFF   R
Memory =   ITCM_Flash          Memory      0x00100000   0x0019FFFF   R
Memory =   BusMatrix_Flash     Memory      0x08000000   0x0809FFFF   RW
Memory =   CPU0_ITCM           Memory      0x00000000   0x0000BFFF   RW
Memory =   CPU1_ITCM           Memory      0x00000000   0x00001FFF   RW
Memory =   CPU0_DTCM           Memory      0x20000000   0x20003FFF   RW
Memory =   CPU1_DTCM           Memory      0x20000000   0x20001FFF   RW
```

5.11.1 J-Link Debug

5.11.1.1 安装 J-Link 器件支持（须先完成）

SEGGER J-Link 默认不认识 G32R501 / G32R502。使用前必须先安装器件包，否则 IAR 无法正确选到芯片。

准备文件

- G32R501：从 “device_support/g32r501/common/Jlink/” 取 “G32R501.xml”、“Geehy_G32R501_ConnectCore.jlinkscript”；从 SDK “package/FLM/” 取 “G32R5xx_Program_Algorithm.FLM”（OTP 调试另取 “G32R5xx_OTP.FLM”）。
- G32R502：从 “device_support/g32r502/common/jlink/” 取 “G32R502.xml”、“Geehy_G32R502_ConnectCore.jlinkscript”；从 SDK “package/FLM/” 取 “G32R502_Program_Algorithm.FLM”（OTP 调试另取 “G32R502_OTP.FLM”）。

以上 FLM 以 SDK “package/FLM” 为准；若 DFP pack 内含同名算法文件，内容应与之保持一致。

Windows

1. 在资源管理器地址栏输入 “%APPDATA%\SEGGER\JLinkDevices\” 并回车（没有则新建）。
2. 新建 “Geehy\G32R501\” 或 “Geehy\G32R502\”，把对应三个文件全部复制进去。
3. 示例：“C:\Users\<你的用户名>\AppData\Roaming\SEGGER\JLinkDevices\Geehy\G32R501\”
4. 关闭并重新打开 IAR、J-Link 相关工具。

Linux / macOS

复制到 “$HOME/.config/SEGGER/JLinkDevices/Geehy/G32R501/” 或 “.../G32R502/”。

确认

打开 J-Link Commander，输入 “device ?”，在 Geehy 下应能看到对应器件名（如 “G32R501DVY”、“G32R502MC”）。

若 OTP 密钥已改，须先编辑 ConnectCore 脚本顶部的 “Z1_CSM_Buff” / “Z2_CSM_Buff”，再复制到上述目录。

5.11.1.2 在 IAR 中配置 J-Link

1. 右键工程名，选择 **Options…**。
2. 打开 **Debugger** 选项卡，在 **Setup → Driver** 中选择 **J-Link**。
3. 选择与板卡一致的 G32R501 / G32R502 型号（须与已安装的 J-Link Devices 名称一致）。
4. 使用上述 J-Link Flash 算法时，请**关闭** IAR 的 **Use flash loader(s)**（避免与 J-Link 侧算法冲突）。
5. 确认下载与复位选项无误后，连接目标板，执行下载并进入调试。

说明：连接 / 复位时的 DCS 解锁由已安装的 ConnectCore 脚本自动完成，一般无需再向工程拷贝独立的

“.jlinkscript”。

5.12 汇编兼容性

不同目标平台的指令集、汇编语法可能存在显著差异，需要对现有的汇编代码进行重新编写和适配，以确保在新的平台上能够正确运行。

5.12.1 文件格式支持

IAR Embedded Workbench 使用的是特定的 IAR 汇编语法。其支持的汇编文件格式：

.s 文件：IAR 风格的汇编文件格式。

与此同时，其支持在 C 函数中使用内联汇编。

5.12.2 汇编编码格式要求

单一汇编文件：这里是一个简单的汇编代码示例，定义了一个汇编函数 add，用于将两个整数相加。文件“add.s”的内容如下：

```
SECTION .text:CODE   ; Define code section
PUBLIC add           ; Declare global symbol
add:
; Function entry
; Parameters: r0 and r1
; Return value: r0
ADD r0, r0, r1       ; Add r0 and r1, store result in r0
BX lr                ; Return to calling function
END
```

C 函数中使用内联汇编：以下是一个在 C 函数中使用内联汇编的示例，定义了一个内联汇编函数 add_inline，用于将两个整数相加：

```
// Inline assembly function
static inline int add_inline(int a, int b){
int result;
__asm volatile(
"adds %0, %1, %2\n"
: "=r"(result)       // Output operand
: "r"(a), "r"(b)    // Input operands
: "cc"                 // Clobbered registers
);
return result;
}
```

5.13 链接脚本文件

链接脚本文件用于定义程序的内存布局和段分配。在迁移过程中，需要根据目标平台和开发环境的不同，使用相应格式的链接脚本文件。G32R5 系列 MCU 在 IAR Embedded Workbench for Arm 开发环境下使用“.icf”格式的链接脚本文件，遵循 IAR 公司的规范。

链接脚本文件的差异

文件格式：

- G32R5 系列 MCU 使用“.icf”格式的链接脚本文件，遵循 IAR 公司的规范。
- Txx320F28004x 使用“.CMD”格式的链接脚本文件，遵循其公司的规范。

内存布局和段分配：

- G32R5 系列 MCU 的“.icf”文件通过定义不同的“region”及其属性来对内存属性进行分配。
- Txx320F28004x 的“.CMD”文件通过定义内存段（MEMORY）和段分配（SECTIONS）来安排内存分配。

5.14 RAM 运行

请参考第 4 章 RAM 运行。

6 Eclipse

6.1 调试器支持

- Geehy-Link（WinUSB）、DAP Link（固件版本为 CMSIS-DAP V2 及以上）
- J-Link V12（J-Link V7.94g 及以上）

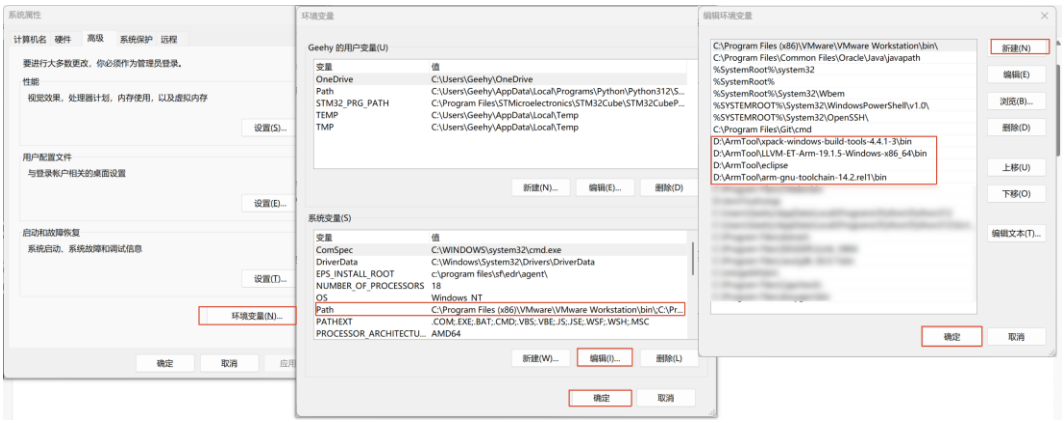
6.2 IDE 版本

请确保使用 Eclipse 4.35 或更高版本的 IDE。

6.3 LLVM-ET-Arm 工具链

1. 下载 LLVM-ET-Arm 编译器。从官方发布渠道下载 LLVM-ET-Arm-19.1.1-Windows-x86_64 编译器。
2. 将下载后的压缩包解压（示例的解压路径为：D:\desktop\clang\utilities\LLVM-ET-Arm-19.1.1-Windows-x86_64）。
3. 将文件夹中的 bin 添加到系统环境变量中。

图 32 添加系统环境变量



6.4 GDB 服务

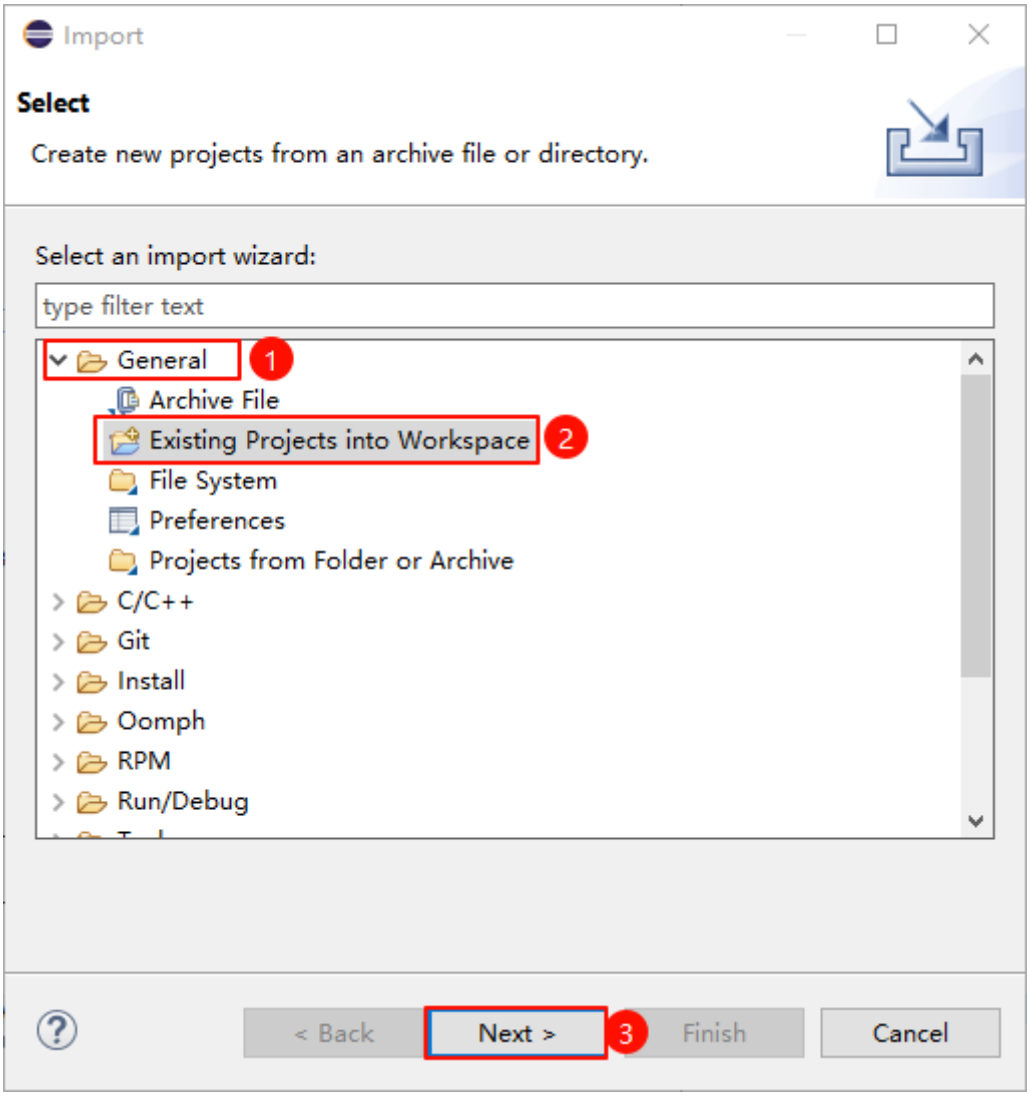
这里建议使用 Arm 提供的 arm-none-eabi-gdb.exe。示例使用的 arm-none-eabi-gdb.exe 版本是 14.2。

6.5 工程操作

6.6 打开示例工程

1. 启动 Eclipse 4.35。
2. 点击菜单栏中的 “File” → “Import …” → “General” → “Existing Project into Workspace”。

图 33 Import project



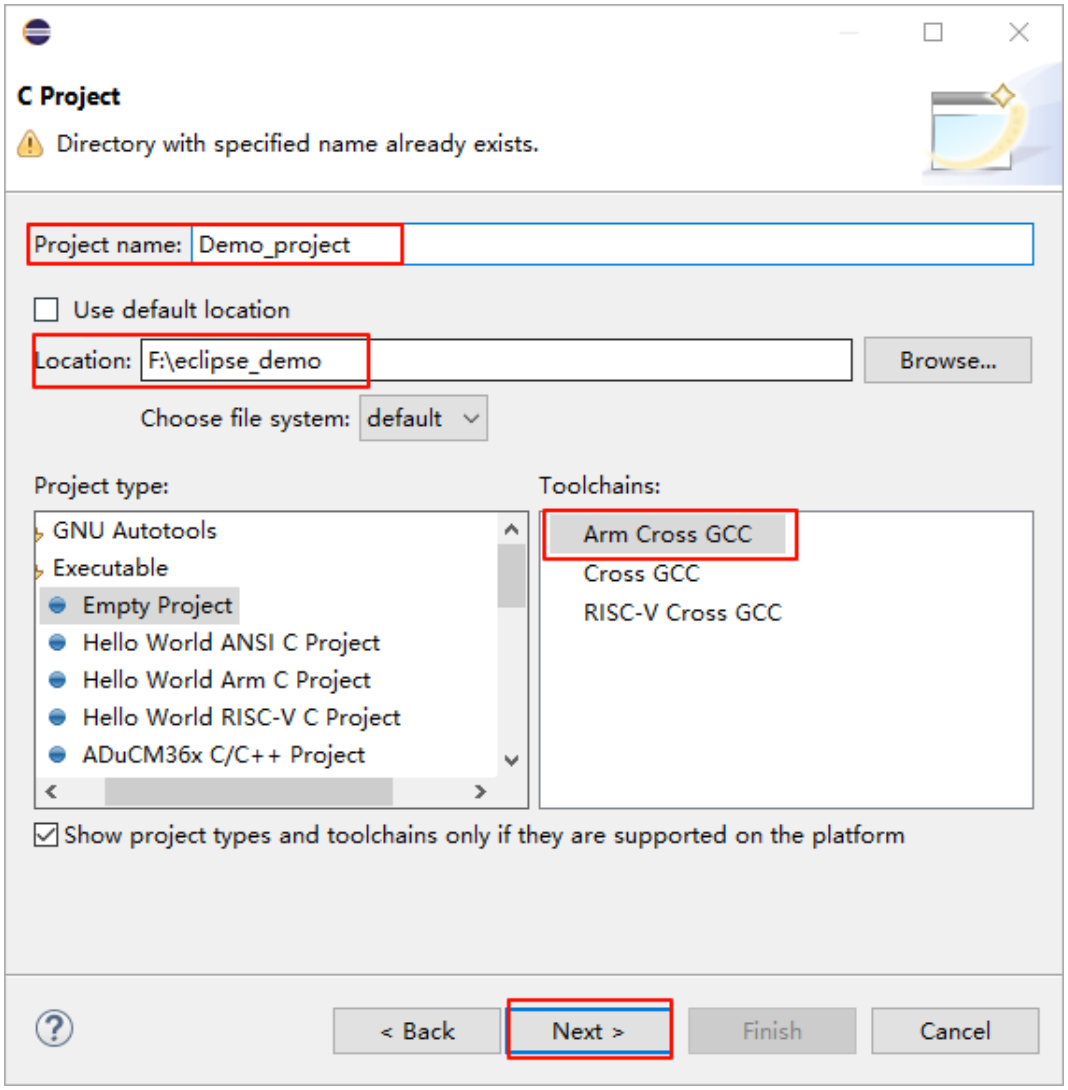
3. 点击 “Select root directory”，浏览到你提供的 SDK 工程文件的路径，选择对应的工程文件夹，然后点击 “Finish”。

注意：以上步骤请在完成章节 6.3 的 LLVM 编译配置后进行。

6.7 工程建立

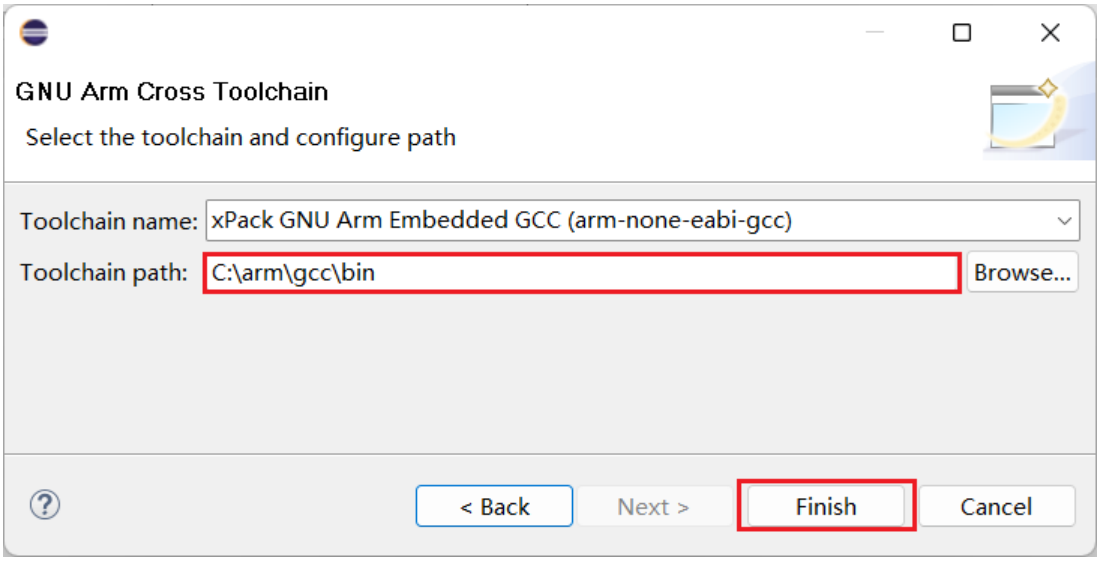
1. 启动 Eclipse 4.35。
2. 在 “File” → “New” 下选择新建 C/C++ Project，选择 C Managed Build。
3. 输入工程名，配置工程类型（建议将该工程放到 Project 目录下），编译链选择 Arm Cross GCC。

图 34 配置工程



4. 若 Eclipse IDE 的 Arm Toolchains 已正确配置，路径将自动选择；否则点击 Browse 选择对应的绝对路径。

图 35 设置 Arm Toolchains Path

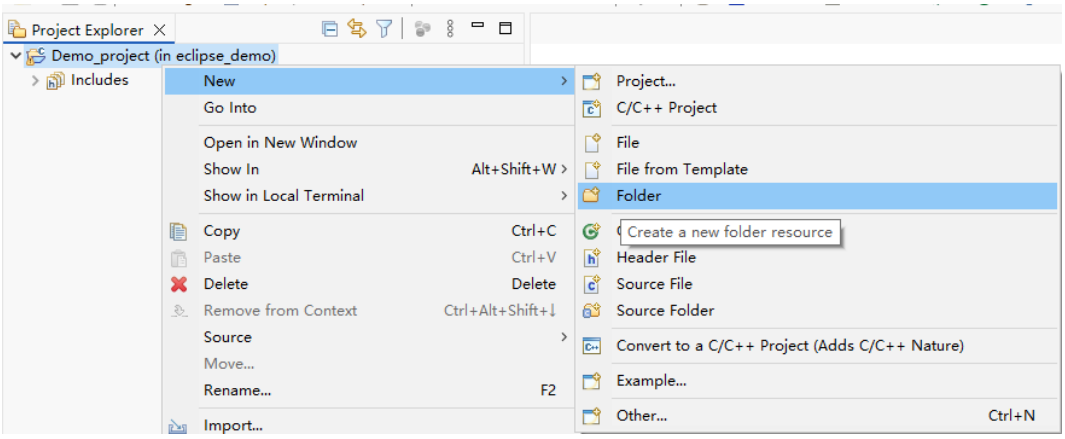


5. 点击“Finish”，完成 Project 的建立。

6.8 文件导入

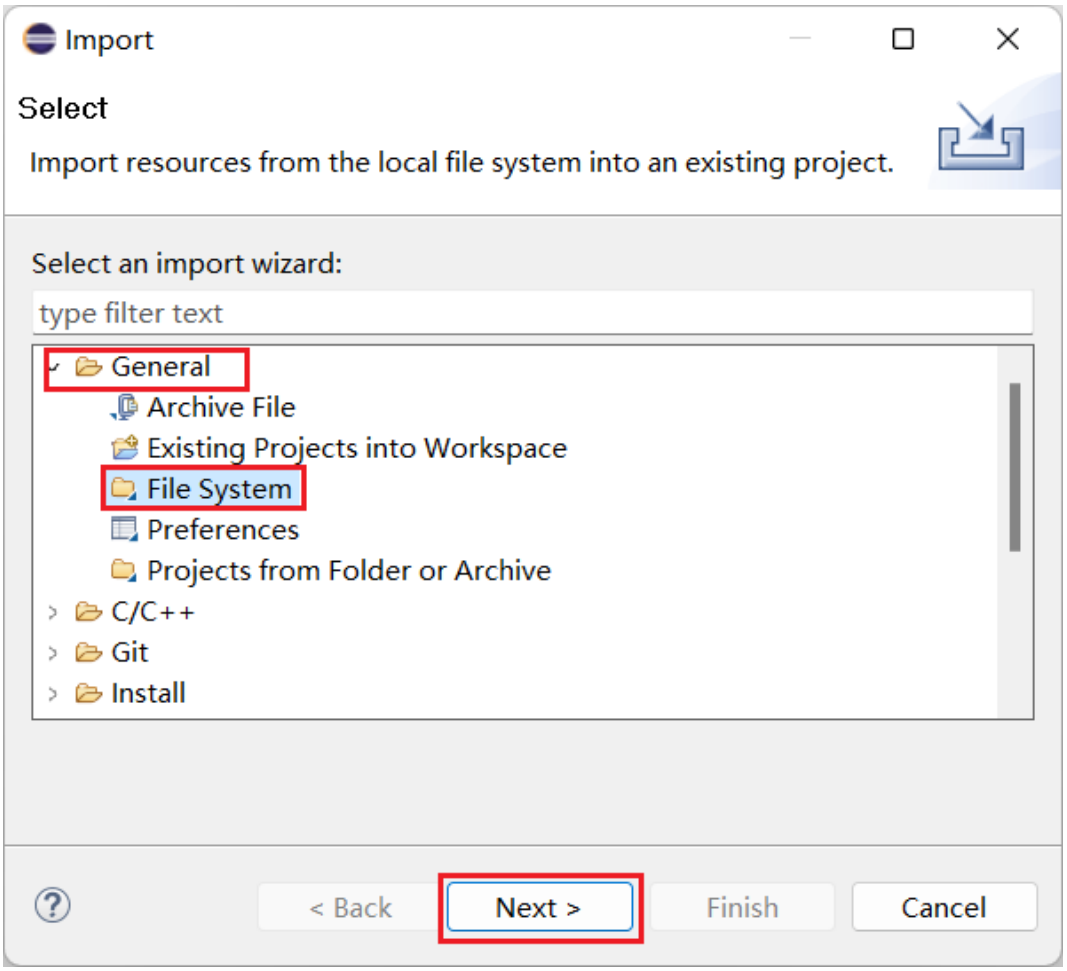
1. 右键所在工程文件夹，建立虚拟文件夹。

图 36 新建文件夹



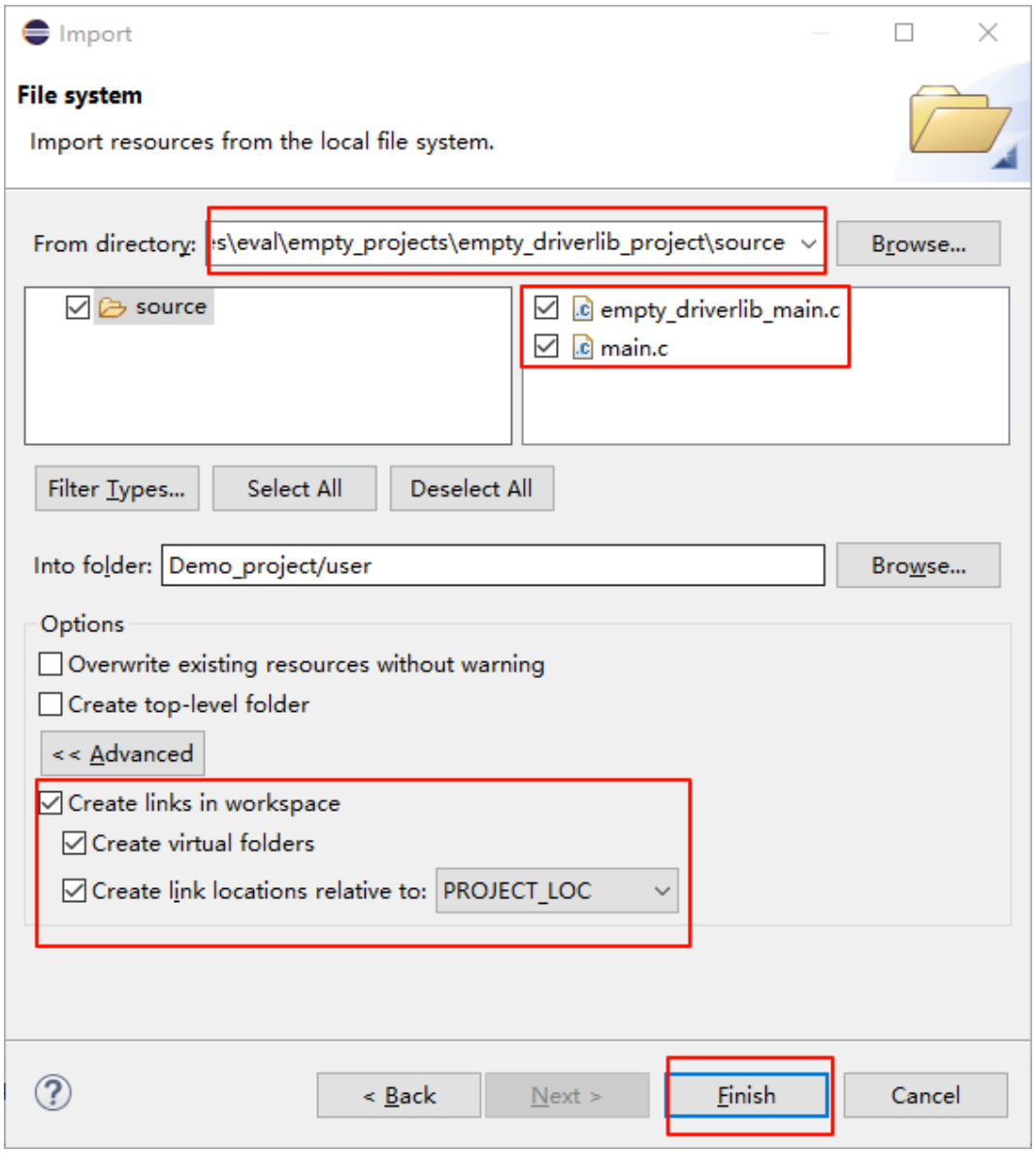
2. 选中文件夹，右键选择 Import，可直接导入文件。

图 37 Import file



3. 导入时选择“File System”作为导入方式，在弹出的文件路径选择对话框中选择路径，并勾选需要导入的文件。

图 38 Select file



4. 如需新建源文件，点击 “File” → “New”，选择要新建的源文件类型，并指定文件夹与文件名。

6.9 配置 Eclipse 全局工具路径（Clang / Make / GDB）

在开始“编译配置”前，建议先把本机工具链路径写入 Eclipse **全局**设置，避免每个工程反复 Browse，也减少部分环境下 Eclipse 读不到系统 PATH 的问题。

本 SDK 的 Eclipse + Clang 工程通常依赖下列工具（版本以 AN1125 环境清单为准，路径请按本机实际解压位置修改）：

- LLVM-ET-Arm（内含 “clang” / “llvm-objcopy” 等），示例：

“D:\desktop\clang\utilities\LLVM-ET-Arm-19.1.1-Windows-x86_64\bin”

- xpack-windows-build-tools（提供 “make”），示例：“C:\Tools\xpack-windows-build-tools-4.4.1-3\bin”
- Arm GNU Toolchain（提供 “arm-none-eabi-gdb” 等调试工具，Debug 配置常用），示例：“C:\Program Files（x86）\Arm GNU Toolchain arm-none-eabi\14.2.rel1\bin”

说明：LLVM-ET-Arm 的下载与解压步骤见前文“LLVM-ET-Arm 工具链”；此处只说明如何写入 Eclipse。

推荐配置步骤：

1. 菜单栏选择 **Window → Preferences**（部分中文界面为“窗口 → 首选项”）。
2. 在左侧展开 **MCU**（或 **C/C++ → Build → Environment**，视 Eclipse / GNU MCU Eclipse 插件版本而定）。
3. 配置全局工具路径（名称可能因插件版本略有差异，按界面实际项选择）：

- **Global LLVM / Clang path**（或 Toolchain path）：指向 LLVM-ET-Arm 的 “bin” 目录。
- **Build tools / Make path**：指向 xpack-windows-build-tools 的 “bin” 目录（保证能找到 “make.exe”）。
- **GDB Client**（若在全局调试页配置）：指向 Arm GNU Toolchain 的 “arm-none-eabi-gdb.exe”。

4. 若插件未提供独立“全局工具路径”页，可在 **C/C++ → Build → Environment** 中新增或编辑变量 “PATH”，把上述三个 “bin” 目录追加到 PATH 最前，并勾选应用到工作区。
5. 点击 **Apply and Close**。

打开任意已导入的 G32R5xx Eclipse 工程，确认工程 **Properties → C/C++ Build → Settings → Toolchains** 中已选择“使用 llvm 内嵌 clang”与“使用 xpack 内置 make”（与后文“配置 ToolChain”一致）；若仍提示找不到工具，回到本步骤检查路径拼写与是否指向 “bin”。

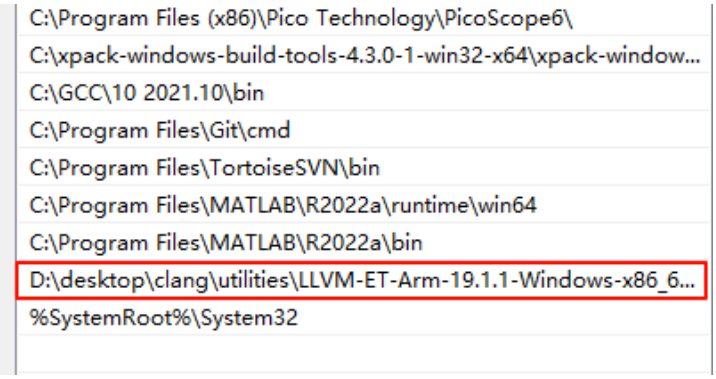
注意：

路径中尽量避免未加引号的空格异常；若必须含空格，在 Environment / 命令行中按 Eclipse 要求加引号。

pyOCD 的全局路径配置见后文“集成至 Eclipse”中的 Global pyOCD Path；与 Clang/Make 路径分开设置。

配置完成后，再进行下一节“编译配置”。

图 39 系统环境变量 Path 中的工具链路径示意



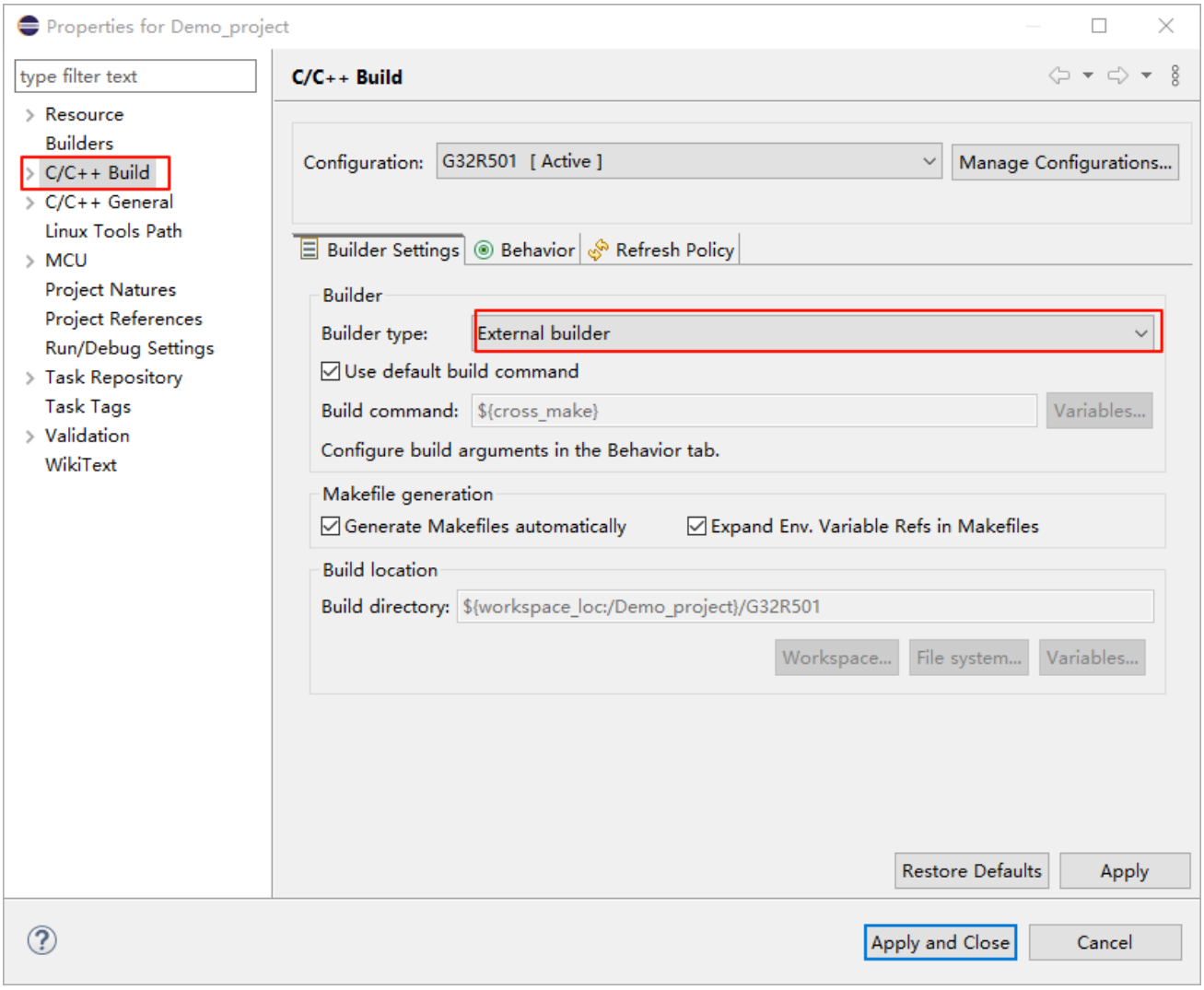
说明：Preferences 中 Clang/Make/GDB 分项全局路径界面若与 Path 示意不同，以本机 Eclipse MCU 插件实际菜单为准；Path 中须能解析到所用工具链可执行文件。

6.10 编译配置

在右键工程名，选择“Properties”。

选择“External builder”配置：

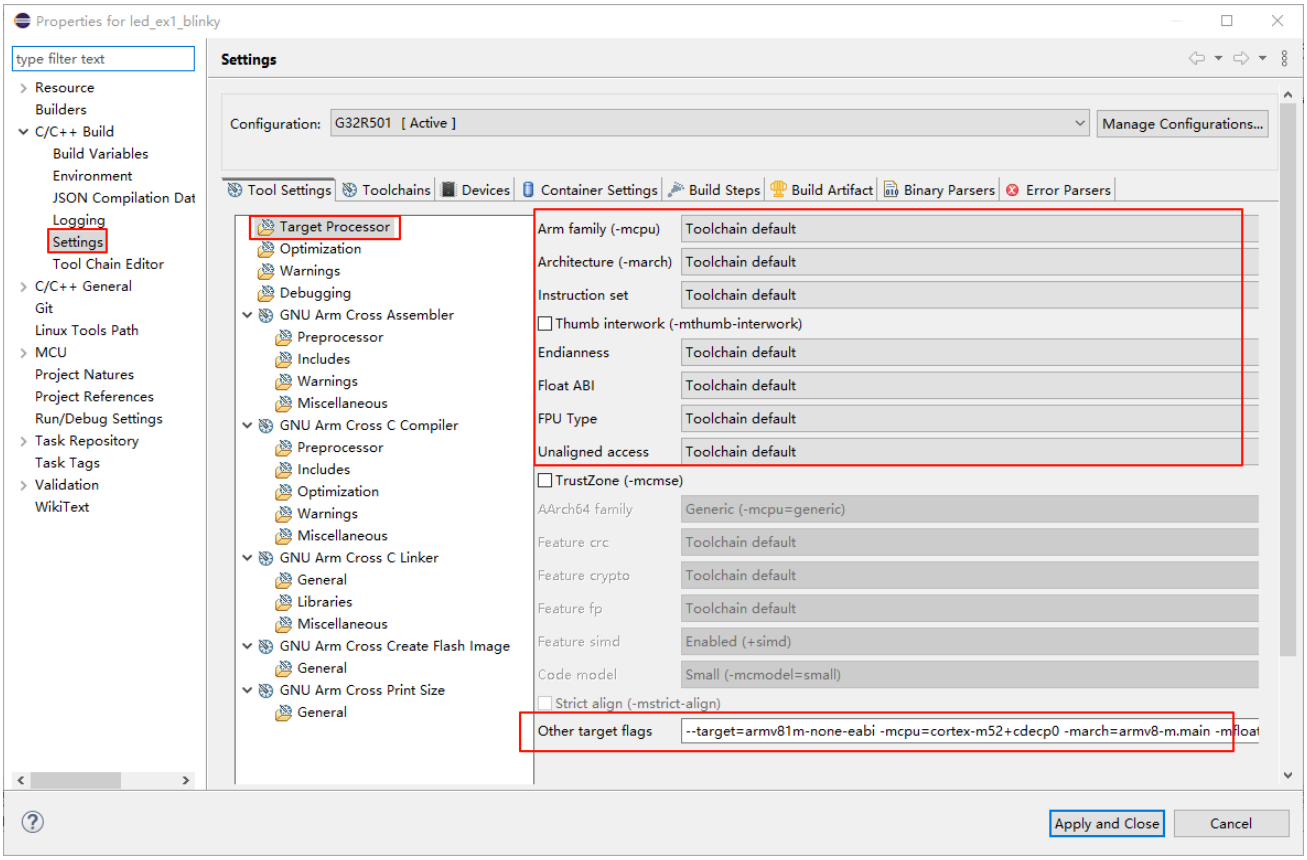
图 40 External builder



在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings” ->“Target Processor”

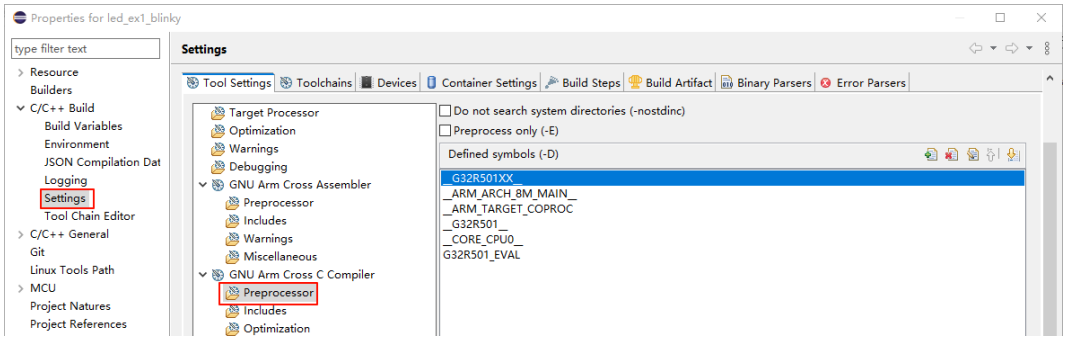
将所有可下拉的配置项选为默认，手动添加命令行：--target=armv81m-none-eabi -mcpu=cortex-m52 -mfpu=none -fno-exceptions -fno-rtti -lcrt0-semihost –lsemihost（可根据芯片实际参数进行修改）

图 41 Target Processor



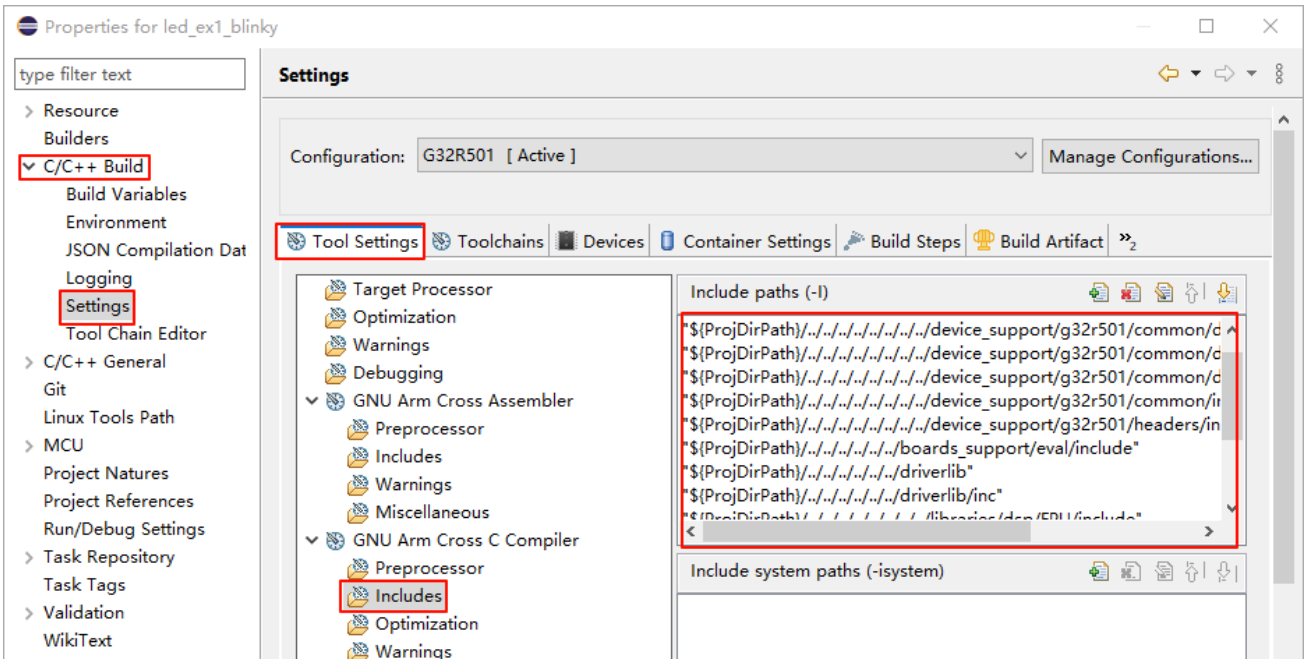
在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“GNU Arm Cross C Compiler”->“Preprocessor”。

图 42 添加宏定义



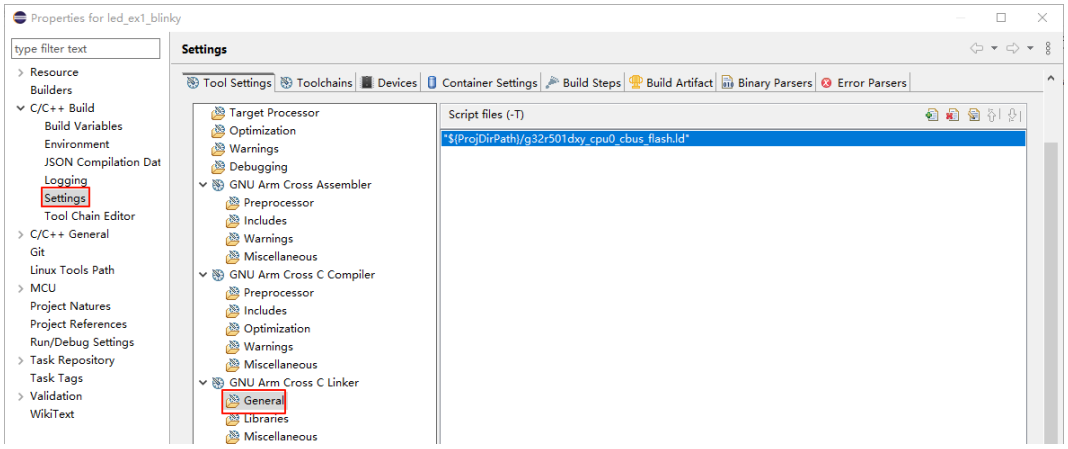
在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“GNU Arm Cross C Compiler”->“Includes”。

图 43 配置头文件路径



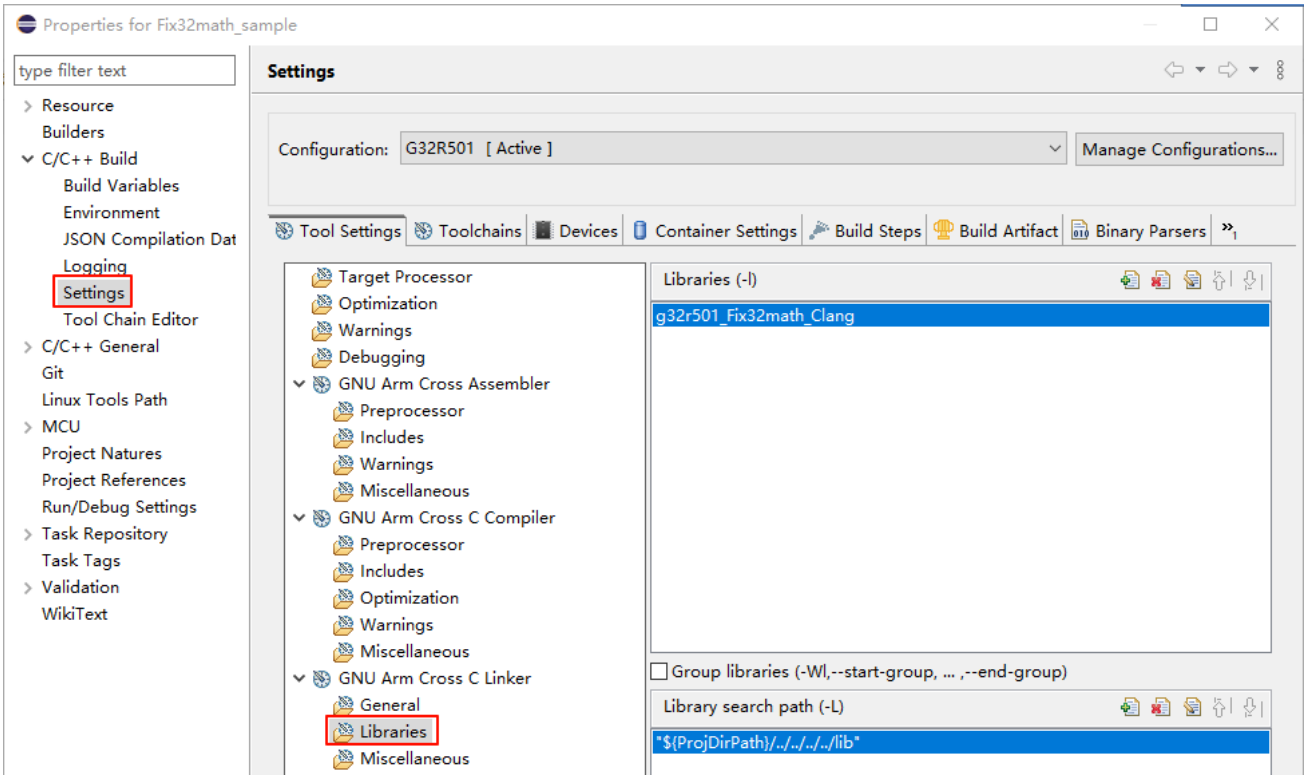
在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“GNU Arm Cross C Linker”->“General”

图 44 添加链接文件



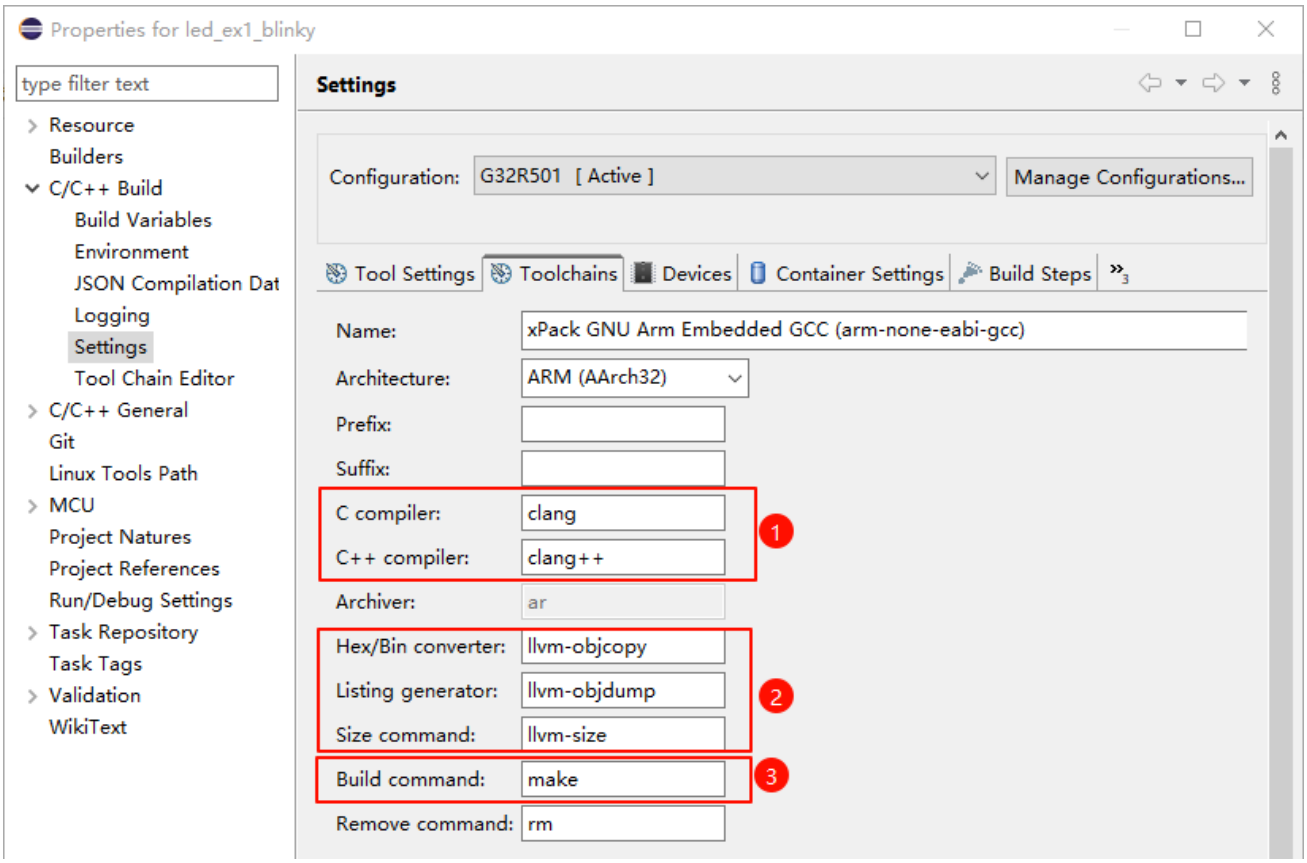
在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“GNU Arm Cross C Linker”->“Libraries”

图 45 添加 libraries



配置“ToolChain”

图 46 配置 ToolChain



使用 llvm 内嵌的 clang 编译器。

llvm 工具链内置工具。

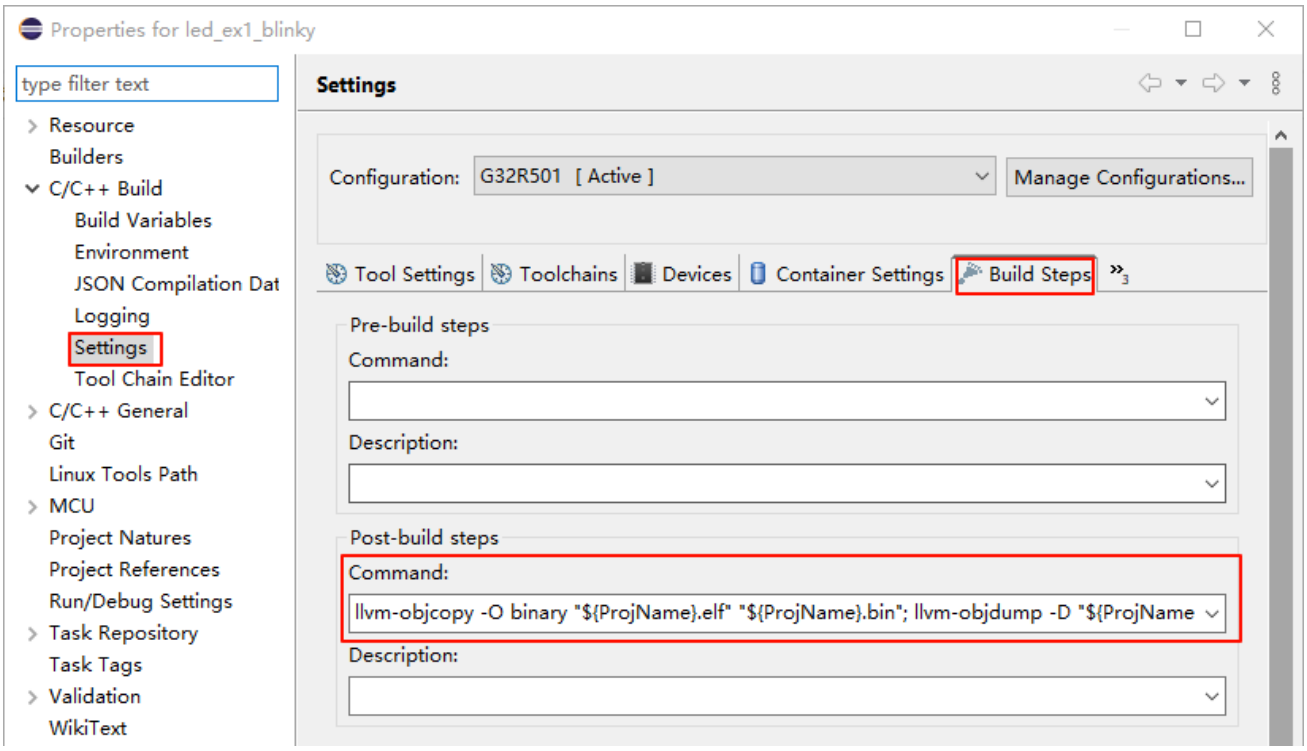
使用 xpack 内置的 make 编译器。

配置“Bulid Step”

在“C/C++ Build”选项卡下选择“Settings”->“Bulid Step”->“Post-build steps”->“Command”

添加命令 llvm-objcopy -O binary "${ProjName}.elf" "${ProjName}.bin"; llvm-objdump -D "${ProjName}.elf" > "${ProjName}.dump"

图 47 Bulid Step

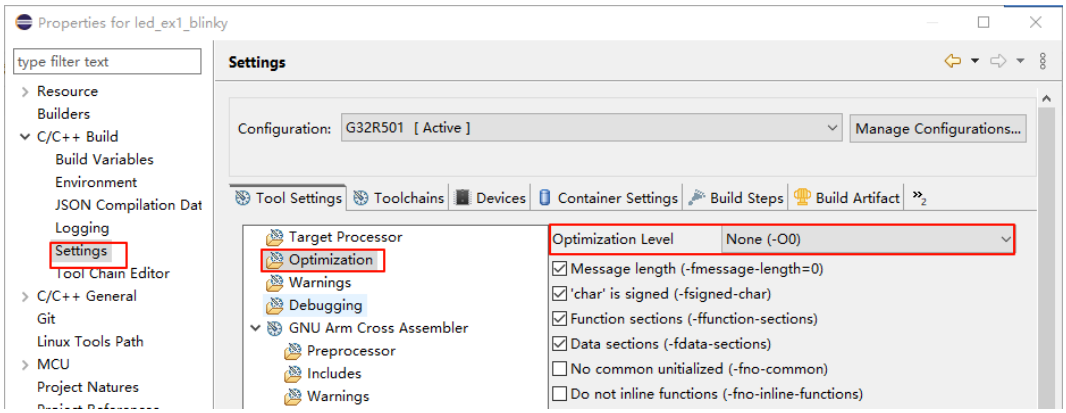


6.11 编译优化等级设置

Eclipse 提供了多种优化等级设置，可以在全局、单文件和单函数级别进行调整：

1. 全局优化等级设置：在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“Optimization”->“Optimization Level”。

图 48 全局优化等级设置



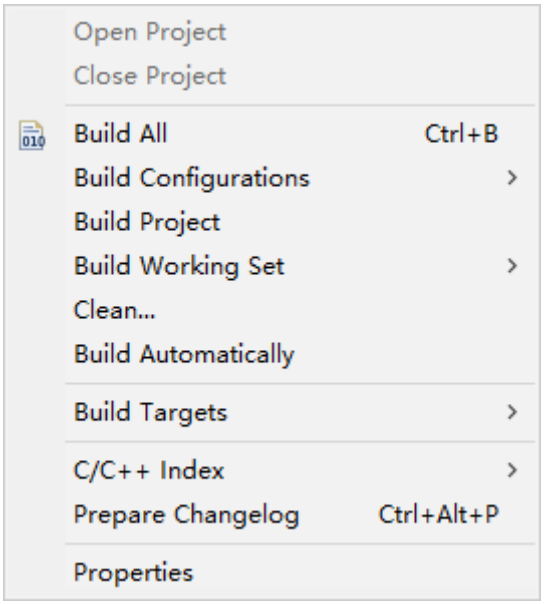
2. 单函数优化等级设置：可以在函数前使用特定的编译器指令进行优化设置，例如使用 _Pragma（"optimize=none"）声明不优化。

6.12 程序编译

1. 点击 Project，选择 Build Project 可对当前工程编译。

注意：使用 Build Project 仅编译当前工程，使用 Build All 则会编译当前 workspace 所有工程。

图 49 编译程序



注意：

（1） 编译前务必先保存当前工程；否则可能编译的是上一次保存的工程，而不是当前修改后的工程。

（2） 修改后须先做工程 Clean，再编译；完成后会生成对应的 elf、hex 及 bin 文件。

（3） 等待编译过程完成，查看输出窗口中是否有错误或警告信息；如有错误，根据提示进行相应的修改。

6.13 程序 Debug 与下载

6.13.1 J-Link Debug

6.13.1.1 安装 J-Link 器件支持（须先完成）

SEGGER J-Link 默认不认识 G32R501 / G32R502。使用 Eclipse 的 J-Link GDB Server 前，必须先安装器件包。

准备文件

- G32R501：从 “device_support/g32r501/common/Jlink/” 取 “G32R501.xml”、“Geehy_G32R501_ConnectCore.jlinkscript”；从 SDK “package/FLM/” 取 “G32R5xx_Program_Algorithm.FLM”（OTP 调试另取 “G32R5xx_OTP.FLM”）。
- G32R502：从 “device_support/g32r502/common/jlink/” 取 “G32R502.xml”、“Geehy_G32R502_ConnectCore.jlinkscript”；从 SDK “package/FLM/” 取

“G32R502_Program_Algorithm.FLM”（OTP 调试另取“G32R502_OTP.FLM”）。

以上 FLM 以 SDK “package/FLM” 为准；若 DFP pack 内含同名算法文件，内容应与之保持一致。

Windows

1. 在资源管理器地址栏输入“%APPDATA%\SEGGER\JLinkDevices\”并回车（没有则新建）。
2. 新建“Geehy\G32R501\”或“Geehy\G32R502\”，把对应三个文件全部复制进去。
3. 示例：“C:\Users\<你的用户名>\AppData\Roaming\SEGGER\JLinkDevices\Geehy\G32R502\”
4. 关闭并重新打开 Eclipse、J-Link 相关工具。

Linux / macOS

复制到“$HOME/.config/SEGGER/JLinkDevices/Geehy/G32R501/”或“.../G32R502/”。

确认

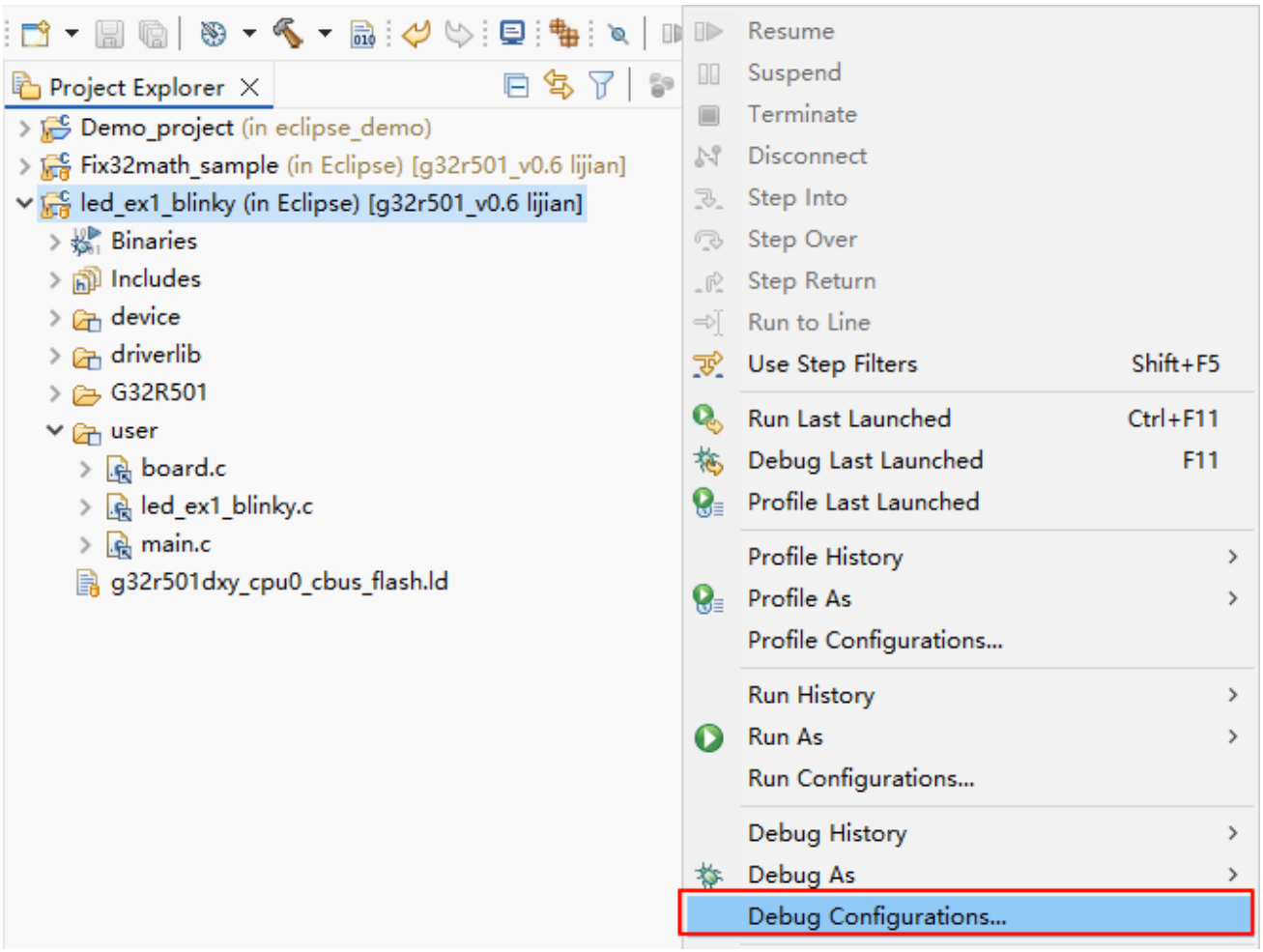
打开 J-Link Commander，输入“device ?”，在 Geehy 下应能看到对应器件名。若 OTP 密钥已改，须先改 ConnectCore 脚本中的“Z1_CSM_Buff” / “Z2_CSM_Buff” 再安装。

6.13.1.2 在 Eclipse 中配置 J-Link

打开 Debug 配置界面

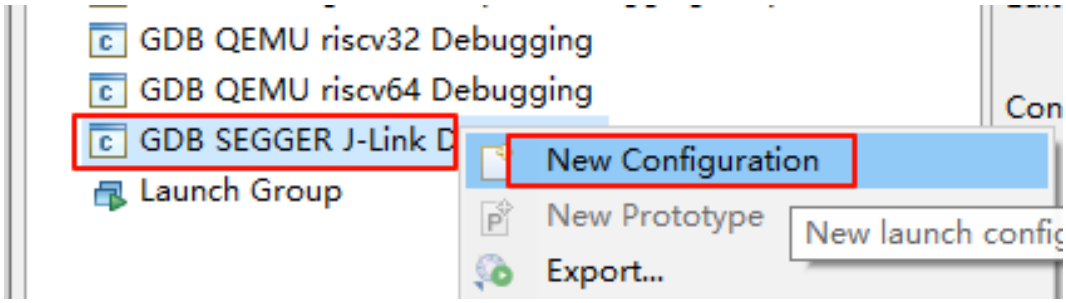
在工程中，点击菜单栏中的 Run，在弹出的选项框中点击 Debug Configurations，进入 Debug 配置界面。

图 50 Debug 配置界面



在新窗口选择“GDB SEGGER J-Link Debugging”，右键。选择“New Configuration”以新建 Debug 配置。

图 51 新建 DebugJ-Link 配置



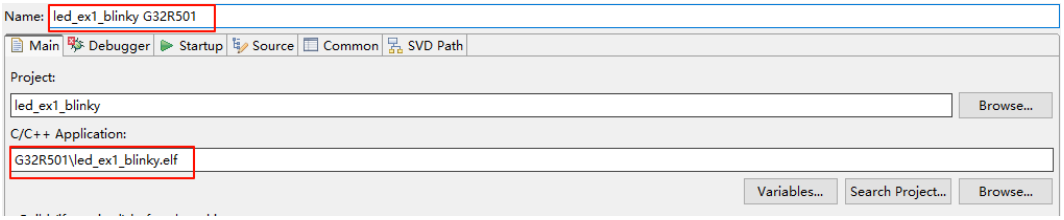
配置 Main 选项卡

在最上端命名当前 Debug 配置名称。

选择“Browse...”，选择当前 Debug 配置对应的工程。

选择对应的 DebugElf 文件，例如：G32R501\led_ex1_blinky.elf。我这里使用的是相对于工程文件的相对路径，可支持绝对路径。

图 52 配置 Main

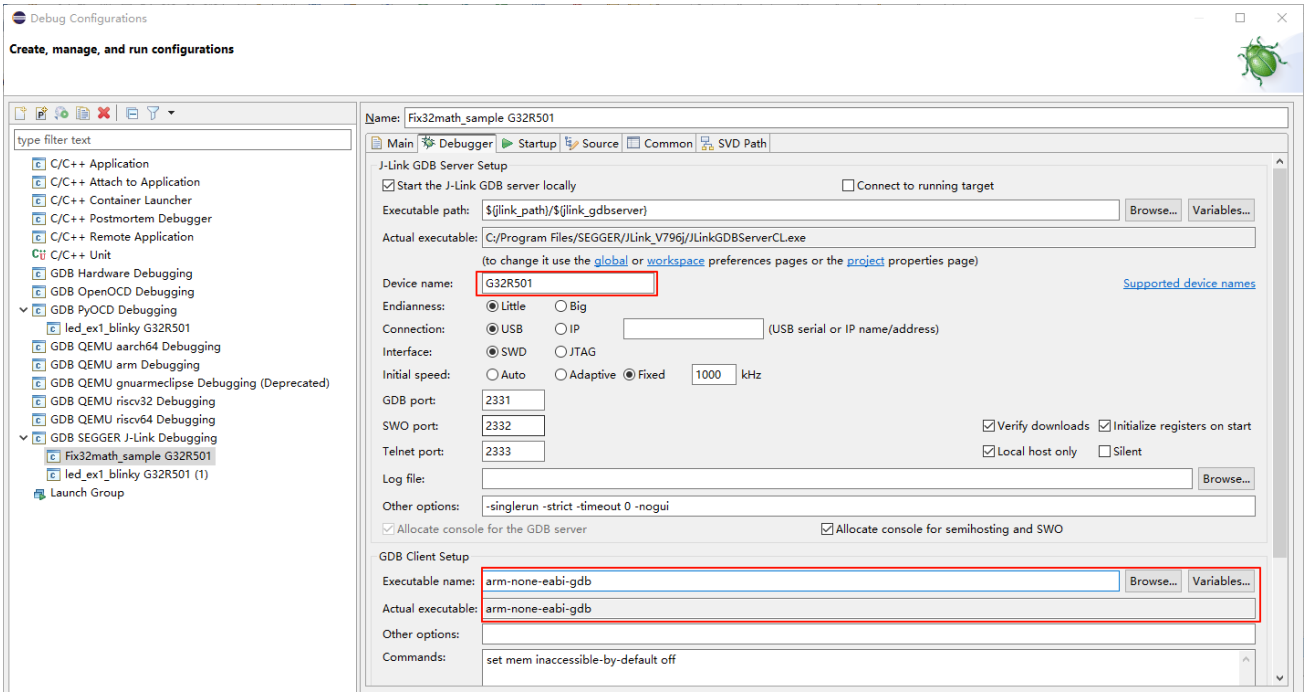


配置 Debugger 选项卡

选择对应的 Debug 芯片，例如：G32R501

选择 GDB 服务，这里建议使用 Arm 提供的 arm-none-eabi-gdb.exe。

图 53 Debugger 选项卡



配置 Startup 选项卡

若芯片使用**出厂默认 DCS 密钥**，且程序从**默认 Flash 启动地址**（CBUS Flash 一般为“0x08000000”）启动，则 **Initialization Commands** 与 **Run/Restart Commands** 中**不必**再手工粘贴 DCS KEY 与向量表脚本。

更推荐的做法（避免在 Startup 里手写一长串内存写命令）：

1. 在资源管理器中打开 SDK，按器件复制下列两个文件到当前 Eclipse 工程目录（与“.elf”同级，或与 Debug 工作目录同级）：

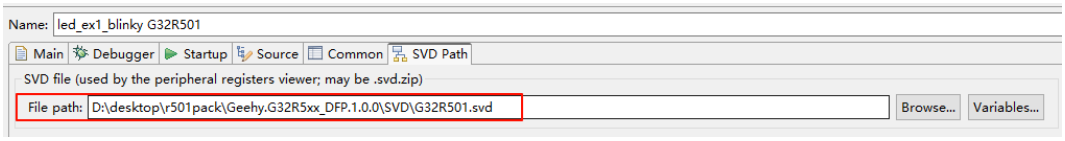
- G32R501：“device_support/g32r501/common/pyOCD/pyocd.yaml” 与 “pyocd_user.py”
- G32R502：“device_support/g32r502/common/pyocd/pyocd.yaml” 与 “pyocd_user.py”

2. 打开 **Debug Configurations**，选择 **GDB PyOCD Debugging**（或工程中已有的 pyOCD 配置）。
3. 将 **Working directory**（工作目录）设为刚才放置了 “pyocd.yaml” / “pyocd_user.py” 的工程目录。
4. 在 pyOCD / 配置文件相关选项中指定使用 “pyocd.yaml”（或命令行等价参数 “--config pyocd.yaml”）。
5. 点击 **Apply**，再启动调试。连接时 “pyocd_user.py” 会自动写入默认 DCS 密钥并设置向量表基址。

仅当 OTP 密钥已改、或启动地址不是默认 Flash（例如改到 ITCM “0x00000000”）时，才需要用文本编辑器打开工程目录中的 “pyocd_user.py”，修改其中的密钥表或向量基址常量后再调试。一般用户使用 SDK 默认模板即可，无需改文件，也无需在 Startup 里添加命令。

配置 SVD 选项卡

图 54 配置 SVD 所在路径目录



最后点击选项卡的右下角的“Apply”按钮，应用所有的配置项。

6.13.2 GEEHY LINK Debug

使用 Geehy-Link（WinUSB）下载及调试工程时，建议配合 pyOCD。请打开本手册第 7 章，按该章正文依次完成：安装 Python 与 pyOCD 0.44 及以上 → 把 “target_G32R501xx.py” / “target_G32R502xx.py” 复制到本机 “pyocd/target/builtin/” 并改 “__init__.py” 注册 → 把对应器件的 “pyocd.yaml” 与 “pyocd_user.py” 复制到工程工作目录 → 再启动调试。第 7 章已写出完整命令与路径，无需另查其它说明文件。

6.14 C 语言兼容性

请参考第 4 章“C 语言兼容性”。

6.15 汇编兼容性

不同目标平台的指令集、汇编语法可能存在显著差异，需要对现有的汇编代码进行重新编写和适

配，以确保在新的平台上能够正确运行。

6.15.1 文件格式支持

Eclipse 使用的是特定的 GCC 汇编语法。GCC 的汇编语法（GNU Assembler，简称 GAS）与 Keil MDK 所用汇编器（armasm）在核心指令集上一致，但伪指令、语法格式和符号定义存在显著差异。以下是详细对比；其支持的汇编文件格式：

.S 文件：GCC 风格的汇编文件格式。

与此同时，其支持在 C 函数中使用内联汇编。

6.15.2 汇编编码格式要求

单一汇编文件：这里是一个简单的汇编代码示例，定义了一个汇编函数 add，用于将两个整数相加。文件“add.s”的内容如下：

```
.section .text   //Define code section
.global add    // Declare global symbol
add:
//Function entry
//Parameters: r0 and r1
//Return value: r0
add r0, r0, r1    // Add r0 and r1, store result in r0
bx  lr          // Return to calling function
```

C 函数中使用内联汇编：以下是一个在 C 函数中使用内联汇编的示例，定义了一个内联汇编函数 add_inline，用于将两个整数相加：

```
// Inline assembly function
static inline int add_inline(int a, int b){
int result;
asm volatile(
"ADD %0, %1, %2"
: "=r"(result)
: "r"(a), "r"(b)
:
);
return result;
}
```

6.16 链接脚本文件

链接脚本文件用于定义程序的内存布局和段分配。在迁移过程中，需要根据目标平台和开发环境的不同，使用相应格式的链接脚本文件。G32R5 系列 MCU 在 Eclipse 开发环境下使用“.ld”格式的链接脚本文件，遵循 Eclipse 的规范。

链接脚本文件的差异

文件格式：

- G32R5 系列 MCU 使用“.ld”格式的链接脚本文件。
- Txx320F28004x 使用“.CMD”格式的链接脚本文件，遵循其公司的规范。

内存布局和段分配：

- G32R5 系列 MCU 的“.ld”文件通过定义不同的“MEMORY”及其“SECTIONS”来对内存属性进行分配。
- Txx320F28004x 的“.CMD”文件通过定义内存段（MEMORY）和段分配（SECTIONS）来安排内存分配。

6.17 RAM 运行

请参考第 4 章 RAM 运行。

7 pyocd 适配 G32R5xx

7.1 背景

为便于用户在开源环境下对 G32R5xx 系列 MCU（含 G32R501、G32R502）进行下载与 Debug，SDK 提供各器件的 pyOCD 目标支持文件与工程模板。

7.2 版本要求

请使用 **pyOCD 0.44 或以上**版本。该版本已支持 Cortex-M52 内核，**无需再手工向 pyOCD 源码中添加 M52 内核支持**。

7.3 器件目标文件（向本机 pyOCD 注册）

SDK 为每个器件提供一个目标模块，必须复制到本机已安装的 pyOCD 目录中的“target/builtin/”，并在同目录“__init__.py”里登记后，命令行与 IDE 才能选用该器件。

源文件位置（SDK 内）

- G32R501：“device_support/g32r501/common/pyOCD/target_G32R501xx.py”
- G32R502：“device_support/g32r502/common/pyocd/target_G32R502xx.py”

目标名称（注册后可选用）

- G32R501 单核：“g32r501xx”
- G32R501 双核：“g32r501dxx”（“pyocd.yaml”模板默认使用双核名）
- G32R502：“g32r502xx”

本机要改动的目录在哪里（先定位，再复制）

最终目标目录相对 pyOCD 安装根均为：

pyocd\target\builtin\

该目录下应能看到已有的“target_.py”与“__init__.py”。Windows 上常见完整路径示例如下（把“<你的用户名>”、“Python312”换成你本机实际用户名与 Python 版本号）：

用 pip 安装到官方 Python（最常见）

C:\Users\<你的用户名>\AppData\Local\Programs\Python\Python312\Lib\site-packages\pyocd\target\builtin\

部分本机/ 打包安装布局（“Scripts_internal”）

C:\Users\<你的用户

名>\AppData\Local\Programs\Python\Python312\Scripts_internal\pyocd\target\builtin\

用源码可编辑安装（“pip install -e .”）

则为你克隆的 pyOCD 源码树，例如：

<你的 pyOCD 源码根>\pyocd\target\builtin\

推荐用命令确认，避免找错目录：

打开 CMD，执行：

pip show pyocd

查看输出中的 **Location** 一行（这是 pyOCD 包所在目录）。

在资源管理器中打开该 Location，再进入子目录 “pyocd\target\builtin\”（若 Location 本身已是 “...\pyocd”，则进入 “target\builtin\”）。

也可在 Python 中打印绝对路径：

```
python -c "import pyocd, pathlib; print(pathlib.Path(pyocd.__file__).resolve().parent /
'target' / 'builtin')"
```

命令打印出的路径，就是你要复制 “target_G32R501xx.py” / “target_G32R502xx.py” 并修改 “__init__.py” 的目录。

注意：若本机装了多套 Python，请确认当前 CMD 里的 “python” / “pip” / “pyocd” 属于同一套；改 A 套的 “builtin”、却用 B 套的 “pyocd” 命令，列表里不会出现新器件。

操作步骤

1. 确认已安装 **pyOCD 0.44 或以上**（见下文“pyocd 安装”），并用上一节方法打开本机 “...\pyocd\target\builtin\” 目录。
2. 将对应的 “target_G32R501xx.py” 或 “target_G32R502xx.py” 从 SDK 复制到该 “builtin” 目录（文件名保持不变）。若两个器件都要用，则两个文件都复制。
3. 用文本编辑器打开同一目录下的 “__init__.py”，做两处修改：

- 在文件中已有的 “from . import ...” 区域增加导入，例如：

```
from . import target_G32R501xx
from . import target_G32R502xx
```

- 在 “BUILTIN_TARGETS” 字典中增加条目，例如：

```
'g32r501xx': target_G32R501xx.G32R501xx,
'g32r501dxx': target_G32R501xx.G32R501Dxx,
'g32r502xx': target_G32R502xx.G32R502xx,
```

4. 保存文件。若 pyOCD 是以 “pip install -e .” 可编辑方式安装的，保存后一般立即生效；若是普通安装，重启一次命令行后再验证即可。

在命令行验证（Windows 示例）：

```
pyocd list -t | findstr g32r50
```

Linux / macOS 示例：

```
pyocd list -t | grep g32r50
```

列表中应能看到 “g32r501xx”、“g32r501dxx”、“g32r502xx”（按你实际注册的条目）。也可用：

```
pyocd list -t -n g32r50
```

说明：pyOCD 0.44 及以上已内置 Cortex-M52 内核支持，**不要**再按旧手册去改 “component_ids” / “core_ids” 手工加内核。只需完成上述目标文件复制与注册。

7.4 推荐工程配置（密钥与启动）

除注册目标外，每次调试还需要把会话配置放进**当前工程工作目录**（与要下载的 “.elf” 或 Debug 配置工作目录相同）。

从 SDK 复制两个文件到工程工作目录（覆盖同名文件前请自行备份）：

- G32R501：“device_support/g32r501/common/pyOCD/pyocd.yaml”、“pyocd_user.py”
- G32R502：“device_support/g32r502/common/pyocd/pyocd.yaml”、“pyocd_user.py”

“pyocd.yaml” 中通过 “user_script: pyocd_user.py” 与 “target_override” 指定器件。示例（G32R502；G32R501 请使用对应模板）：

```
* target_override: g32r502xx
user_script: pyocd_user.py
flash.timeout.init: 30.0
frequency: 8000000
persist: true
```

“target_override” 取值说明：

- G32R501 双核模板默认：“g32r501dxx”
- G32R501 若只用单核：改为 “g32r501xx”，并按模板注释调整 “session” 段

- G32R502：“g32r502xx”

“pyocd_user.py” 内含默认 DCS 密钥表与向量表基址：

- 使用**出厂默认密钥**且从 **Flash “0x08000000”**启动时，**不要改**该文件
- 仅当 OTP 密钥已改，或启动地址改为 ITCM 等非默认地址时，再用文本编辑器修改其中的密钥对或基址常量

在该工作目录打开命令行，执行例如：

```
pyocd gdbserver --config pyocd.yaml
```

或：

```
pyocd cmd --config pyocd.yaml
```

在 Eclipse 中：打开 **Debug Configurations**，选择 pyOCD 相关配置，将工作目录设为放置了上述两个文件的工程目录，并指定使用 “pyocd.yaml”，再启动调试。

命令行自检示例（板卡已连接）

```
pyocd list -p
pyocd erase --config pyocd.yaml -c 0x08000000
pyocd load --config pyocd.yaml <你的程序.elf>
pyocd gdbserver --config pyocd.yaml
```

7.5 pyocd 安装

7.5.1 Windows

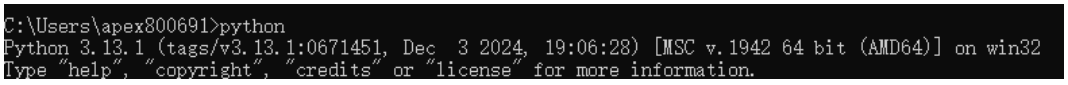
7.5.1.1 安装 python

pyocd 支持需要在 python 环境下，请至 Python 官网（。

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 55 python 安装验证



7.5.1.2 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：

- 使用 pip 命令安装（建议指定版本下限，例如）：

```
pip install "pyocd>=0.44"
```

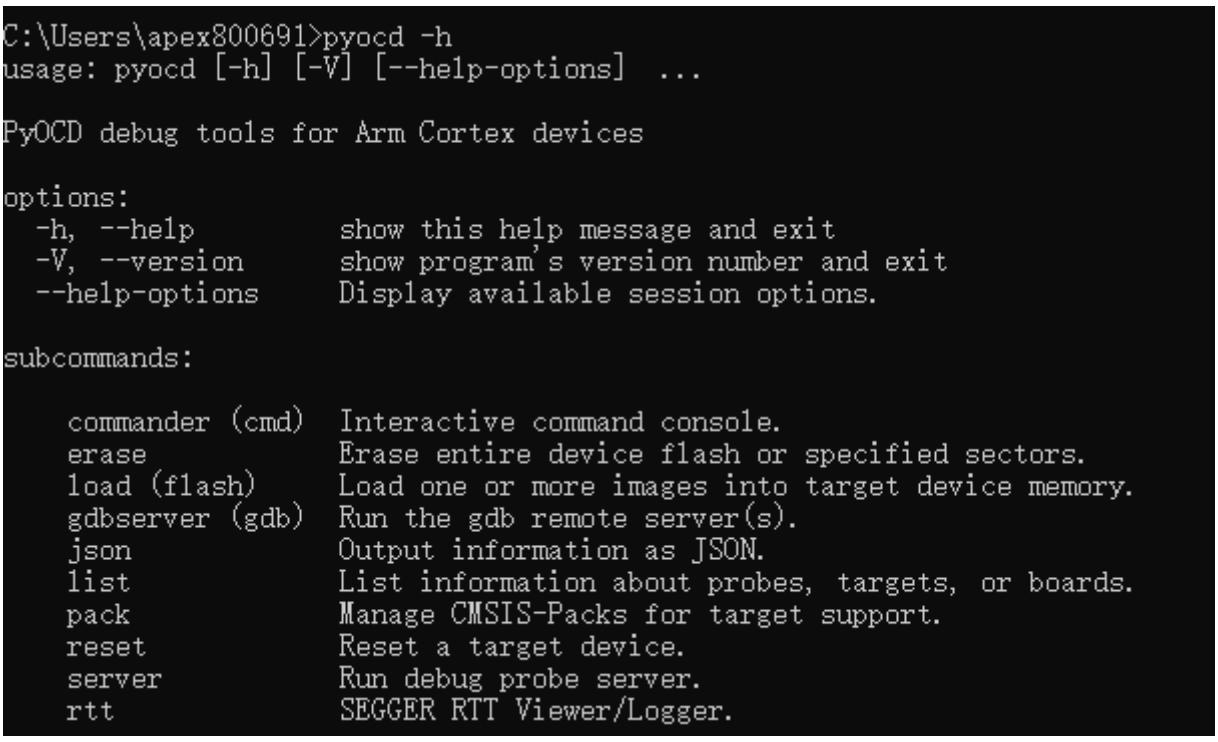
- 也可在图形界面或其它方式安装；安装后务必确认版本不低于 0.44：

图 56 安装 pyocd

注意：安装完成后，请将 pyOCD 可执行文件所在目录加入 PATH。建议安装 **pyOCD 0.44 或以上**。

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 57 pyocd 安装验证



7.5.2 Ubuntu

7.5.2.1 安装 python

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
```

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
```

7.5.2.2 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。

使用以下命令来安装 python3-venv 包：

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

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
```

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
```

7.5.2.3 安装 pyocd

在激活的虚拟环境中，使用 pip 安装 pyOCD：

```
pip install pyocd
```

验证安装结果

```
pyocd --version
```

查询 pyocd 安装位置（以便后续更改 pyocd 源码）

```
pip show pyocd
```

7.5.2.4 pyocd 的 USB 权限

如果 pyOCD 在普通用户下无法访问调试器，可添加 udev 规则（Geehy-Link USB 厂商 ID 为 314B）：

1. 创建新的规则文件：

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

2. 在文件中添加：

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

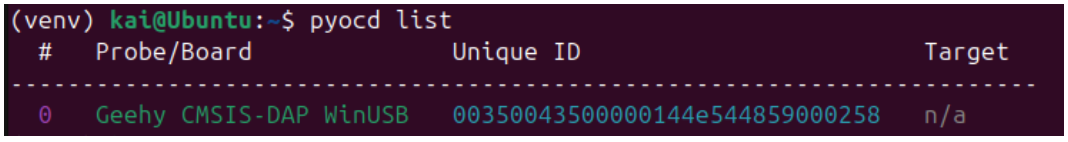
3. 保存（Ctrl+O，Enter）并退出（Ctrl+X），然后重新加载规则：

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

4. 验证调试器识别：

```
pyocd list
```

图 58 连接至 Ubuntu 的调试器序列号

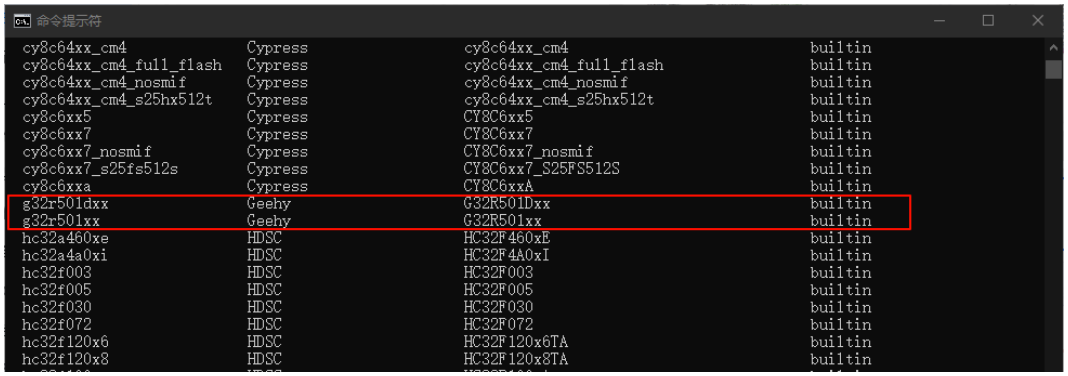


7.5.2.5 确认器件已注册成功

完成上文“器件目标文件（向本机 pyOCD 注册）”全部步骤后，在命令行执行 “pyocd list -t”（可用 “findstr” / “grep” 过滤 “g32r50”），确认列表中出现你已注册的 “g32r501xx” / “g32r501dxx” / “g32r502xx”。若没有出现，请回到注册步骤检查：文件是否拷到 “pyocd/target/builtin/”、

“__init__.py”是否已保存、以及当前命令行用的是否为同一套 Python / pyOCD。

图 59 支持芯片列表



7.6 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. 将板卡或调试器连接至 PC。
2. 在工作目录下准备会话文件（与上文“推荐工程配置”相同）：

- 从 SDK 复制对应器件的“pyocd.yaml”与“pyocd_user.py”到该工作目录（G32R501：“device_support/g32r501/common/pyOCD/”；G32R502：“device_support/g32r502/common/pyocd/”）。
- 确认“pyocd.yaml”中的“target_override”与已注册目标名一致。
- 默认密钥与默认 Flash 启动地址时，不要改“pyocd_user.py”。

3. 在该工作目录启动命令行，输入：

```
pyocd commander --config pyocd.yaml
```

或：

```
pyocd cmd --config pyocd.yaml
```

图 60 pyocd commander

```
#   Probe/Board             Unique ID                       Target
-----------------------------------------------------------------------
0   Geehy CMSIS-DAP WinUSB  00330057500000144e544859000258  n/a
1   Geehy CMSIS-DAP WinUSB  00480051500000054e544859000258  n/a

nter the number of the debug probe or 'q' to quit> 1
010556 W Invalid coresight component, cidr=0x0 [rom_table]
onnected to G32R501Dxx [Halted]: 00480051500000054e544859000258
yocd> erase
yocd> rw 0x08000000 0x10
8000000:  ffffffff ffffffff ffffffff ffffffff    |................|
yocd>
```

7.7 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. 菜单栏选择 **Window → Preferences**（部分中文界面为“窗口 → 首选项”；若需先显示全部配置项，可在菜单栏处右键后再进入 Preferences）。
2. 展开 **MCU**。
3. 选择 **Global pyOCD Path**。
4. 在右侧选择对应的 “pyocd.exe” 所在目录。
5. 点击 **Apply and Close**。

图 61 Global pyOCD Path 步骤 1-2

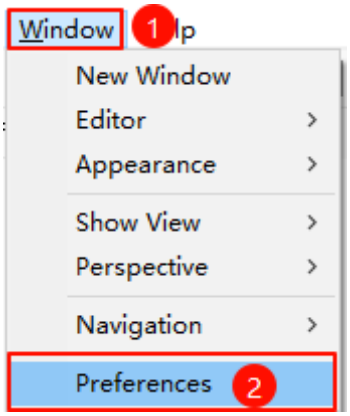
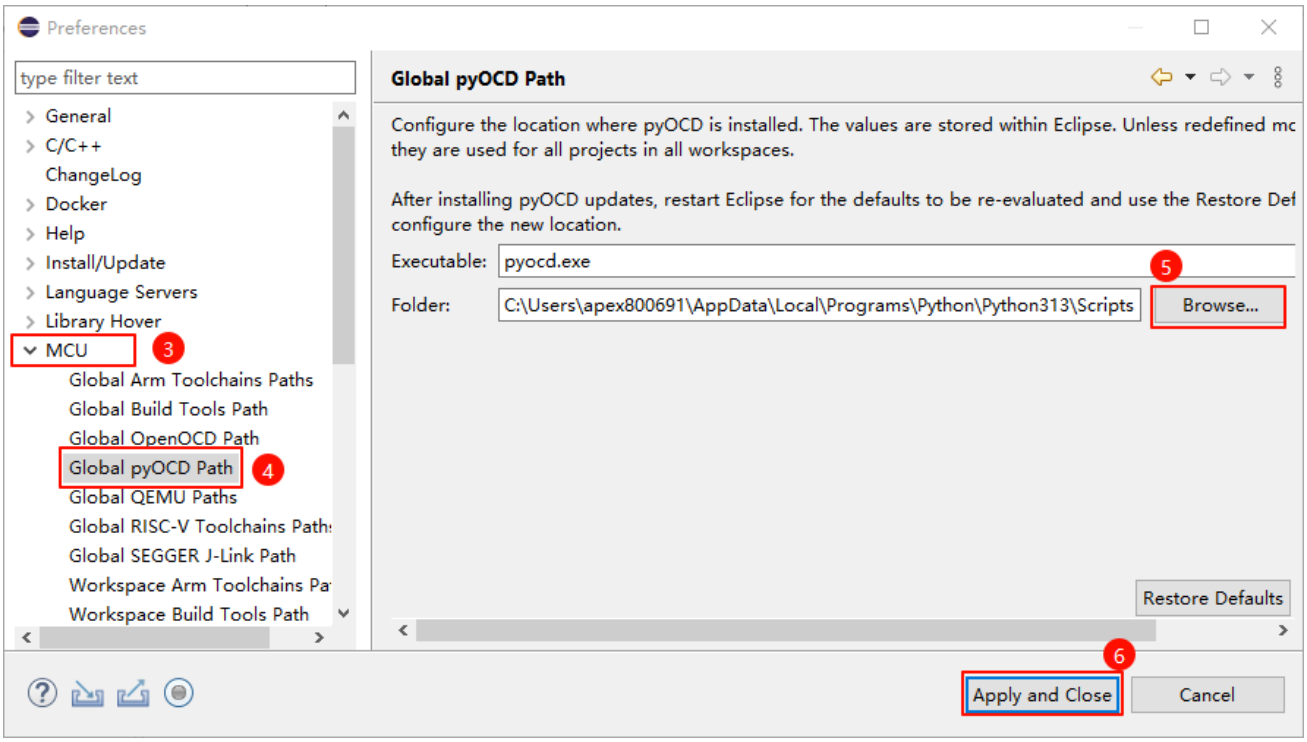


图 62 Global pyOCD Path 步骤 3-6



7.7.1 单核 Debug 配置

Eclipse 导入工程，确保编译无误后，请按照如下步骤配置 Debug 选项卡。

新建 Debug 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 Debug 配置。

图 63 New Configuration 步骤 1-2

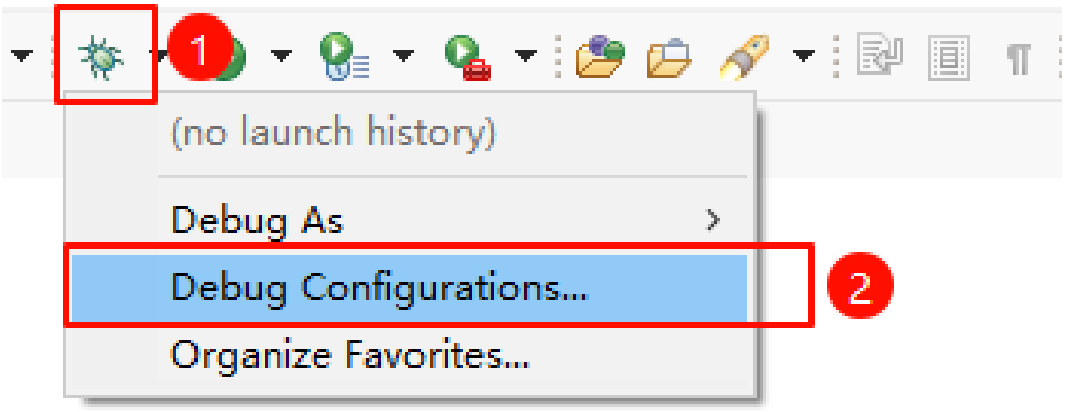
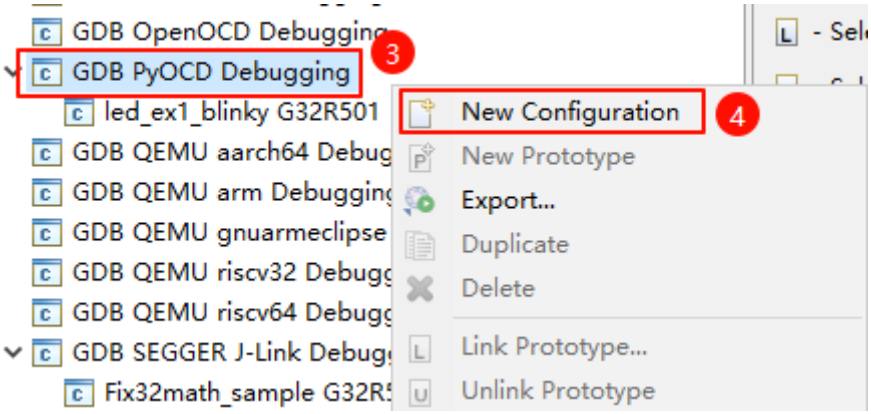


图 64 New Configuration 步骤 3-4



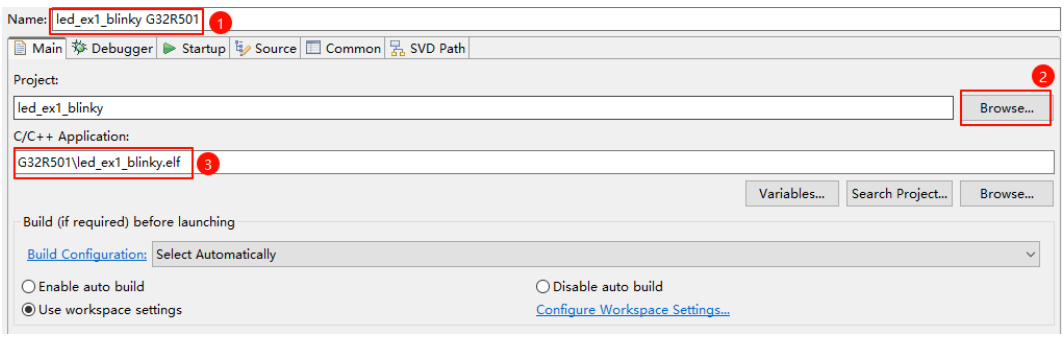
配置 Main 选项卡

在最上端命名当前 Debug 配置名称。

选择“Browse...”，选择当前 Debug 配置对应的工程。

选择对应的 DebugElf 文件，例如：G32R501\led_ex1_blinky.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 65 配置 Main



配置 Debugger 选项卡

选择使用到的 Geehy-Link 调试器，括号中的字符为调试器序列号

选择对应的 Debug 芯片，例如：Geehy >G32R501xx（g32r501xx）

复位模式选择“Software（SYSRESETREQ）”

配置文件建议同时使用 “pyocd.yaml” 与 “pyocd_user.py”（可自 “device_support/g32r501/common/pyocd/” 或 “device_support/g32r502/common/pyocd/” 复制到工程工作目录）。Eclipse Debugger 选项中的配置文件栏优先填写 **pyocd.yaml 的绝对路径**；也可仅填写 “pyocd_user.py”。路径不要有中文或空格。

“pyocd.yaml” 示例（G32R502；G32R501 请改用对应目录模板，双核默认可为 “g32r501dxx”）：

```
* target_override: g32r502xx
user_script: pyocd_user.py
flash.timeout.init: 30.0
frequency: 8000000
persist: true
```

向量表基址在 “pyocd_user.py” 中配置（如 “VECTOR_TABLE_ADDRESS” / “CORE0_SET_ADDR”），不在 yaml 中设置。

命令行示例：

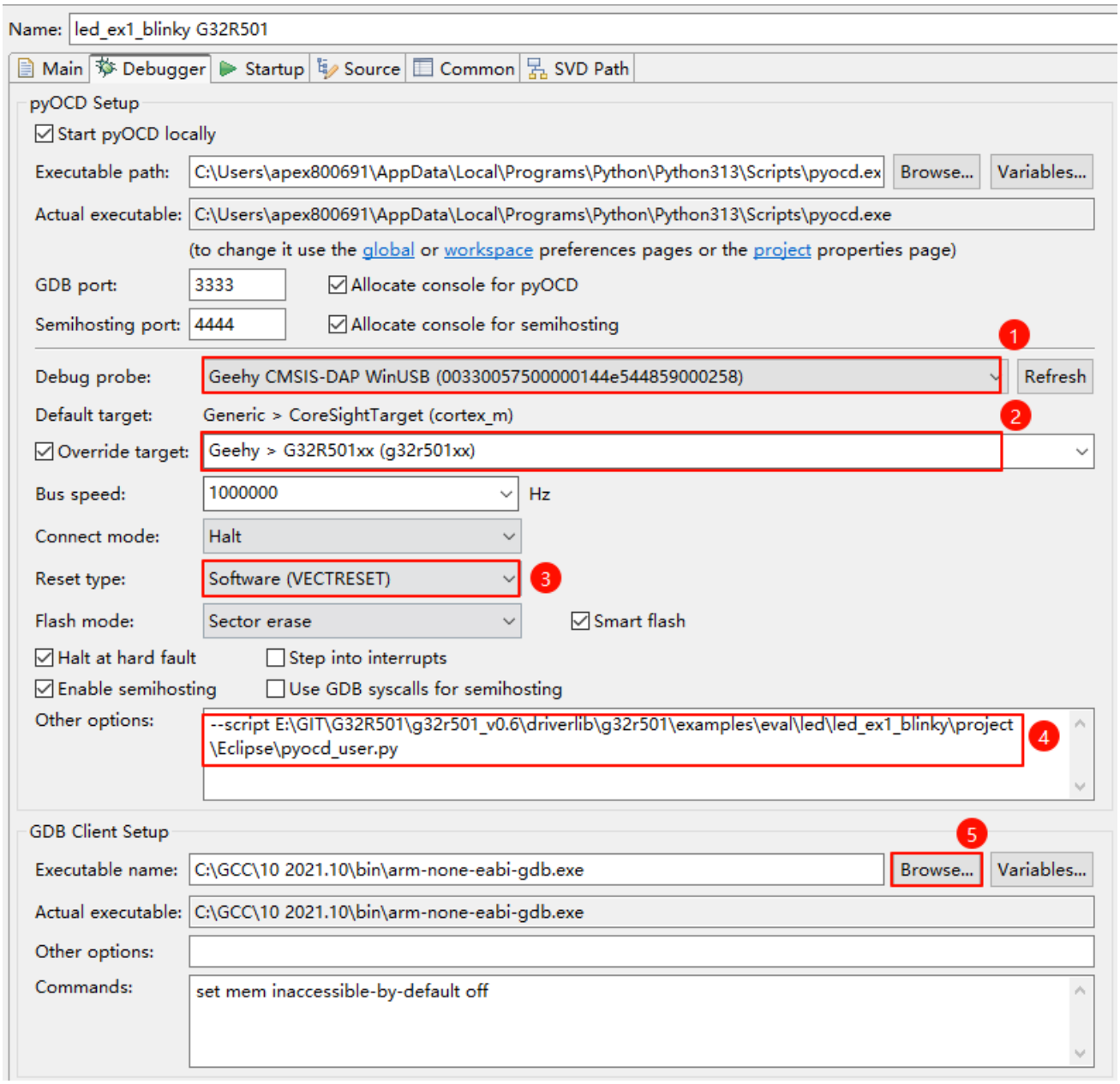
```
pyocd gdbserver --config pyocd.yaml -j <含 yaml 与 pyocd_user.py 的目录>
```

或：

```
pyocd gdbserver --config <绝对路径>/pyocd.yaml --target g32r502xx
```

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 66 配置 Debugger



配置 Startup 选项卡

若芯片使用**出厂默认密钥**，且启动地址 / 向量表与默认配置一致，**可不在** Initialization Commands、Run/Restart Commands 中重复粘贴 DCS KEY 与 SP/PC 设置。

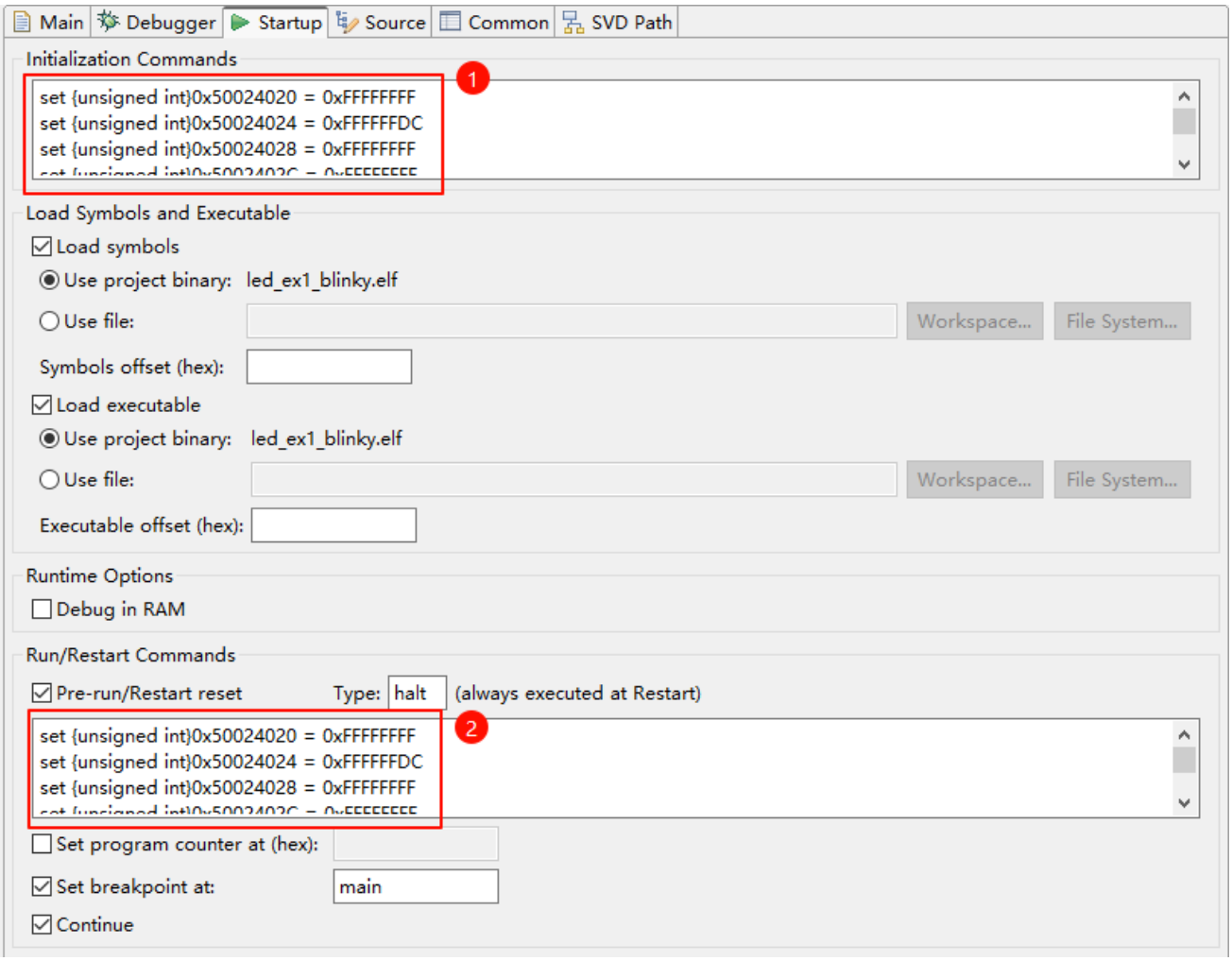
若需要**更新密钥或启动地址**，建议优先按上一节配置 “pyocd.yaml” + “pyocd_user.py” 统一维护，稳定性更好；仅在无法使用用户脚本时，再于 Initialization Commands、Run/Restart Commands 处添加 DCS KEY（须与 “pyocd_user.py”、芯片端一致）。

两处添加的内容一致时，可参考：

```
set {unsigned int}0x50024020 = 0xFFFFFFFF
```

```
set {unsigned int}0x50024024 = 0xFFFFFFDC
set {unsigned int}0x50024028 = 0xFFFFFFFF
set {unsigned int}0x5002402C = 0xFFFFFFFF
set {unsigned int}0x500240A0 = 0xFFFFFFFF
set {unsigned int}0x500240A4 = 0xFFFEDFFF
set {unsigned int}0x500240A8 = 0xFFFFFFFF
set {unsigned int}0x500240AC = 0xFFFFFFFF
set $sp = *(unsigned int*)0x08000000
set $pc = *(unsigned int*)0x08000004
set $xpsr = $xpsr | (1<<24)
```

图 67 配置 Startup



最后点击选项卡的右下角的“Apply”按钮，应用所有的配置项。

7.7.2 双核 Debug 配置

双核 Debug 配置见 AN1128《G32R501 双核 Debug 指导手册》

7.7.3 Ubuntu 下退出 Debug 后程序无法运行

7.7.3.1 原因

Ubuntu 下的 Eclipse 在 arm-none-eabi-gdb 发送 resuming core 0 [cortex_m]前终止了当前线程，导致芯片无法正常复位。

7.7.3.2 解决方案

参考双核的 Debug 配置，在终端启动 pyOCD gdbserver 的解决方案。

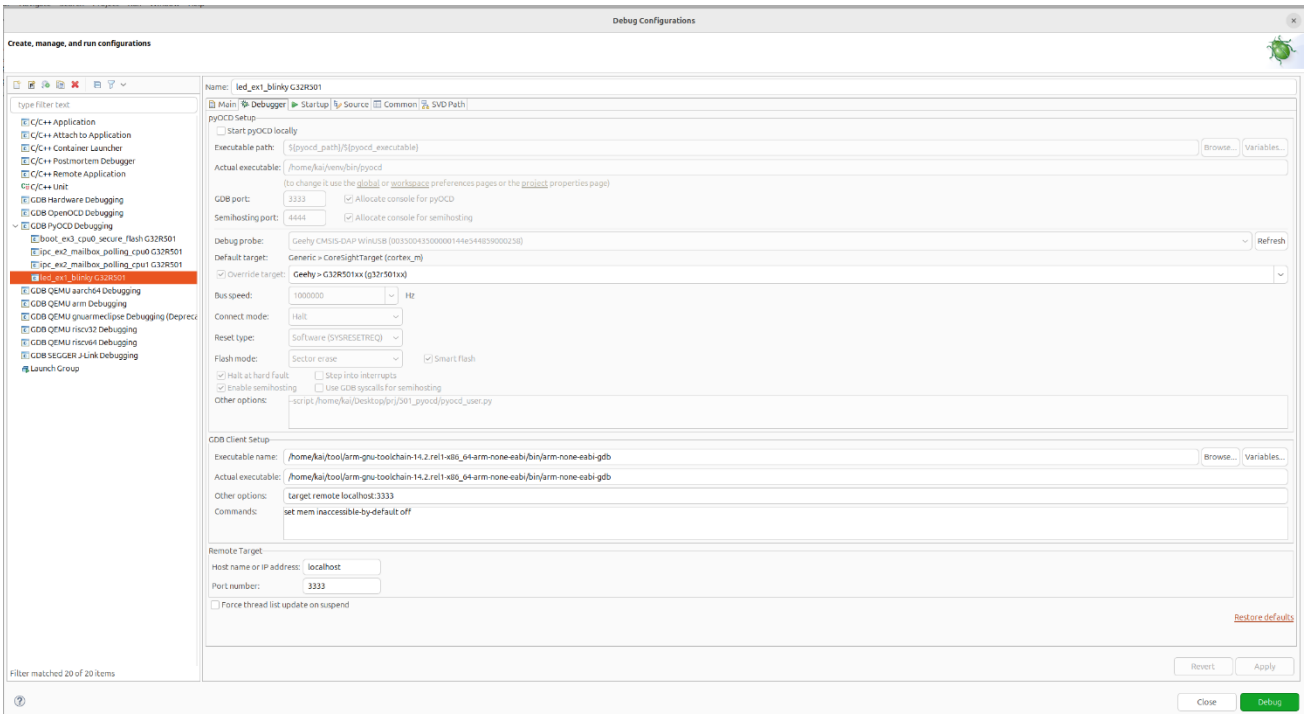
在终端启动 pyOCD gdbserver：

图 68 启动 pyOCD gdbserver

```
(venv) kai@Ubuntu:~/Desktop/prj/501_pyocd$ pyocd gdbserver
0001289 I Patched target_G32R501xx DECRYPT_KEYS via pyocd_user.py! [pyocd_user]
0001327 I Target type is g32r501xx [board]
0001366 I DP IDR = 0x6ba02477 (v2 rev6) [dap]
0001383 I AHB-AP#0 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001389 I AHB-AP#1 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001395 I AHB-AP#2 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001401 I AHB-AP#0 Class 0x1 ROM table #0 @ 0xe00ff000 (designer=a75:Arm China part=4d2) [rom_table]
0001406 I [0]<e000e000:SCS Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=2a04 devid=0:0:0> [rom_table]
0001409 I [1]<e0001000:DWT Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a02 devid=0:0:0> [rom_table]
0001411 I [2]<e0002000:BPU Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a03 devid=0:0:0> [rom_table]
0001414 I [3]<e0000000:ITM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=43 archid=1a01 devid=0:0:0> [rom_table]
0001417 I [5]<e0041000:ETM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=13 archid=4a13 devid=0:0:0> [rom_table]
0001422 I [6]<e0003000:??? class=9 designer=a75:Arm China part=d24 devtype=16 archid=0a06 devid=0:0:0> [rom_table]
0001425 I [7]<e0042000:CTI Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=14 archid=1a14 devid=40800:0:0> [rom_table]
0001428 I [8]<e0046000:PMC-100 class=9 designer=43b:Arm part=9ba devtype=55 archid=0a55 devid=145509d6:c105c04:0> [rom_table]
0001436 I AHB-AP#1 Class 0x1 ROM table #0 @ 0xe00ff000 (designer=a75:Arm China part=4d2) [rom_table]
0001442 I [0]<e000e000:SCS Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=2a04 devid=0:0:0> [rom_table]
0001447 I [1]<e0001000:DWT Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a02 devid=0:0:0> [rom_table]
0001450 I [2]<e0002000:BPU Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a03 devid=0:0:0> [rom_table]
0001455 I [3]<e0000000:ITM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=43 archid=1a01 devid=0:0:0> [rom_table]
0001459 I [5]<e0041000:ETM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=13 archid=4a13 devid=0:0:0> [rom_table]
0001464 I [6]<e0003000:??? class=9 designer=a75:Arm China part=d24 devtype=16 archid=0a06 devid=0:0:0> [rom_table]
0001467 I [7]<e0042000:CTI Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=14 archid=1a14 devid=40800:0:0> [rom_table]
0001470 I [8]<e0046000:PMC-100 class=9 designer=43b:Arm part=9ba devtype=55 archid=0a55 devid=145509d6:c105c04:0> [rom_table]
0001478 I CPU core #0: Star-MC2 r0p1, v7.0-M architecture [cortex_m]
0001478 I   Extensions: [DSP, FPU, FPU_DP, FPU_V5, MPU] [cortex_m]
0001478 I   FPU present: FPv5-D16-M [cortex_m]
0001479 I core is created and initialized. [target_G32R501xx]
0001484 I 4 hardware watchpoints [dwt]
0001486 I 8 hardware breakpoints, 1 literal comparators [fpb]
0001493 I 4 hardware watchpoints [dwt]
0001495 I 8 hardware breakpoints, 1 literal comparators [fpb]
r501 connect in did_connect...
0001550 I Semihost server started on port 4444 (core 0) [server]
0001569 I GDB server started on port 3333 (core 0) [gdbserver]
```

Eclipse Debugger 选项卡配置:

图 69 Eclipse Debugger



注意：终端启动 pyOCD gdbserver 的路径下应当包含单核的“pyocd.yaml”文件。

8 版本历史

表格 4 文件版本历史

日期	版本	变更说明
2025.1	1.0	● 新建
2025.4	1.1	● 新增章节
2026.8	1.2	● 适用产品列为 G32R501、G32R502 ● Keil 工程建立补充选择微控制器截图 ● 程序 Debug 下将 J-Link / GEEHY LINK 调整为子节（IAR / Eclipse 同理） ● 补充 J-Link Devices 与 pyOCD 本机注册步骤 ● 明确 G32R502 使用 IAR 须 9.70.4 及以上 ● FLM 改取自 SDK package/FLM（含 Keil J-Link 准备文件清单） ● Eclipse pyOCD 补充 yaml 与命令行示例 ● 默认密钥可跳过 Startup 长命令 ● 新增 Eclipse 全局工具路径（Clang / Make / GDB）配置说明 ● 同步刷新 IDE/调试相关操作截图，与现行工具链界面一致 ● 修正 Startup 中 SP/PC 须按向量表解引用的写法 ● 补充 Eclipse 章节添加系统环境变量截图。

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