应用笔记

Application Note

文档编号：**AN1126**

G32R501 IDE 与工具链使用说明

版本：**V1.1**

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详尽的指南，帮助他们在使用 G32R501 微控制器进行开发时能够顺利配置和使用这两种流行的开发环境。

为了充分利用本手册中的信息，用户应熟悉 G32R501 系列微控制器的特性。可以参考以下相关的技术文档：

- G32R501 系列用户手册、数据手册和编程手册、迁移手册
- G32R501 系列内核相关文档，包括 ARMv8.1-M 架构参考手册等

本文档详细介绍了在 MDK-ARM 和 IAR EW for Arm 中配置项目的步骤，涵盖了从项目创建、编译链接到调试的全过程。并通过对比不同开发环境的特性和设置，读者将能够更好地理解如何在不同的开发环境中高效地进行嵌入式项目开发。

本文档将引导读者熟练掌握在 MDK-ARM 和 IAR EW for Arm 环境下开发 G32R501 的技巧，从而为项目的成功实施奠定坚实的基础。

1.1. 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

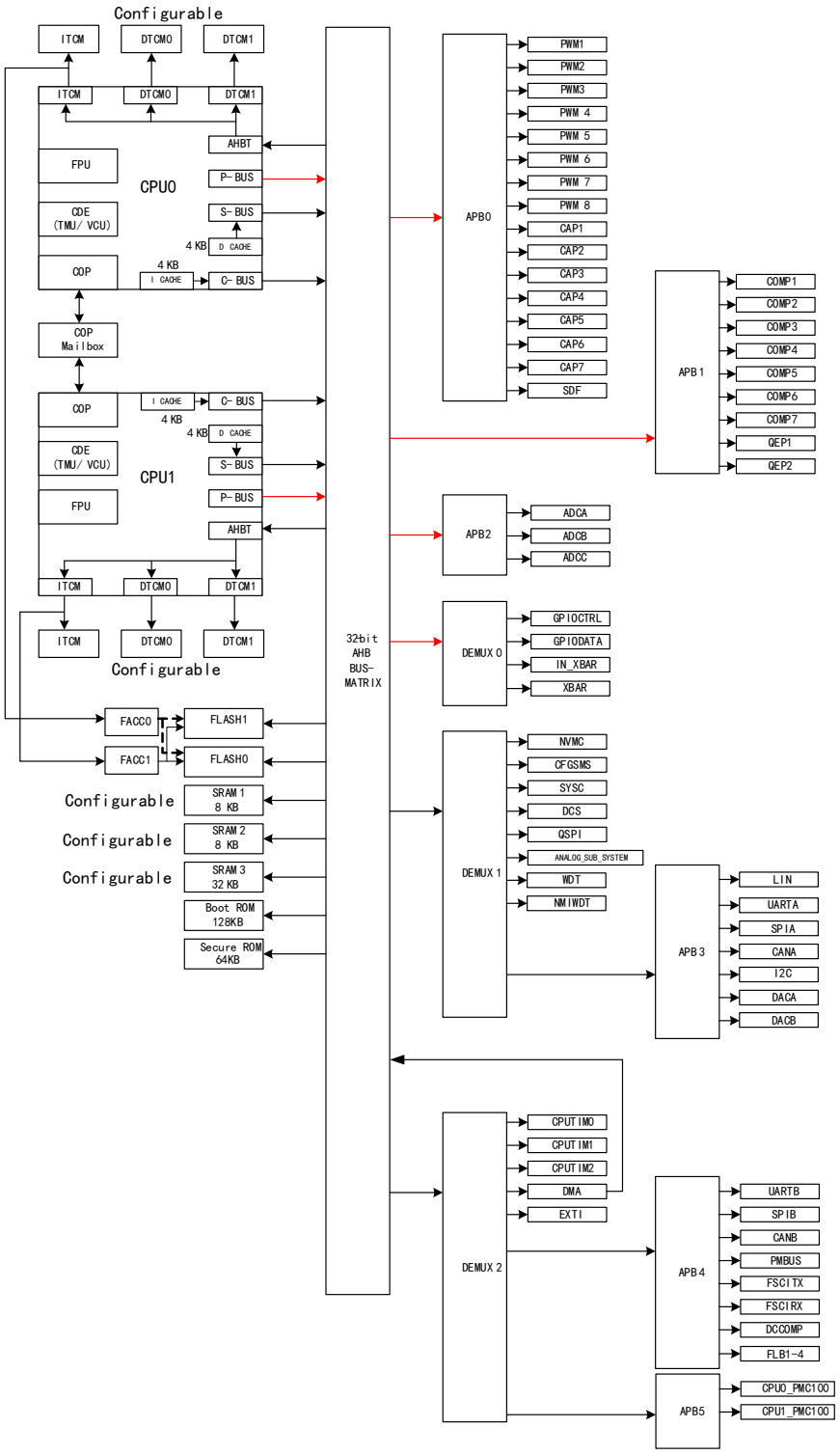
中文全称	英文全称	英文缩写
可编程静态存储子系统	Configurable Static Memory Subsystem	CFGSMS
软件	Software.	SW
硬件	Hardware.	HW
接口	Interface	IF
AHB 从接口	AHB slave interface.	ahbs_if

1.2. R5xx SoC 简介

R5xx SoC 是一款基于 Cortex-M52 双核心的系统级芯片（SoC）。该 SoC 采用了广泛应用的 AMBA 2.0 总线来集成外设 IP 组件：需要大带宽的组件连接到 AMBA AHB 总线上，而低速组件连接到 AMBA APB 总线上。DMA 不能访问与 FLB 相关的寄存器。

R5xx SoC 的架构如图 1 所示。

图 1 R5xx SoC 架构



注意：

1) 图上红色线为总线桥。
2) ITCM/DTCM/SRAM 大小通过 CFGSMS 进行配置.。

目录

2. MDK-ARM 开发工具链

在从 Txx320F28004x 系列微控制器迁移到 G32R501 系列微控制器的过程中，开发工具的迁移是一个重要环节。Code Composer Studio（CCStudio）集成开发环境（IDE）和 MDK-ARM 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 MDK-ARM 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。

2.1. 仿真器支持

G32R501 仿真器支持情况：

- 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

2.2. IDE 版本

请确保使用 MDK-ARM V5.40 或更高版本的 IDE。

注意：MDK 5.40/5.41 的已知问题，代码编辑过程中的函数跳转（按下 F12，会跳转至函数定义）等功能无法正常使用。

2.3. 工程操作

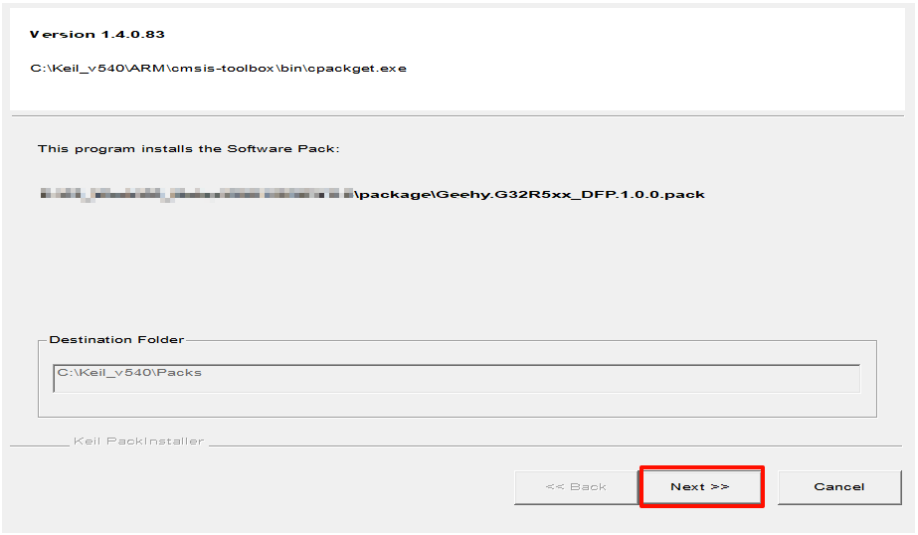
注意：以下操作均在 MDK-ARM v5.40 上进行。

2.3.1. 安装 Pack 支持

你可以选择以下任一方法进行安装：

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

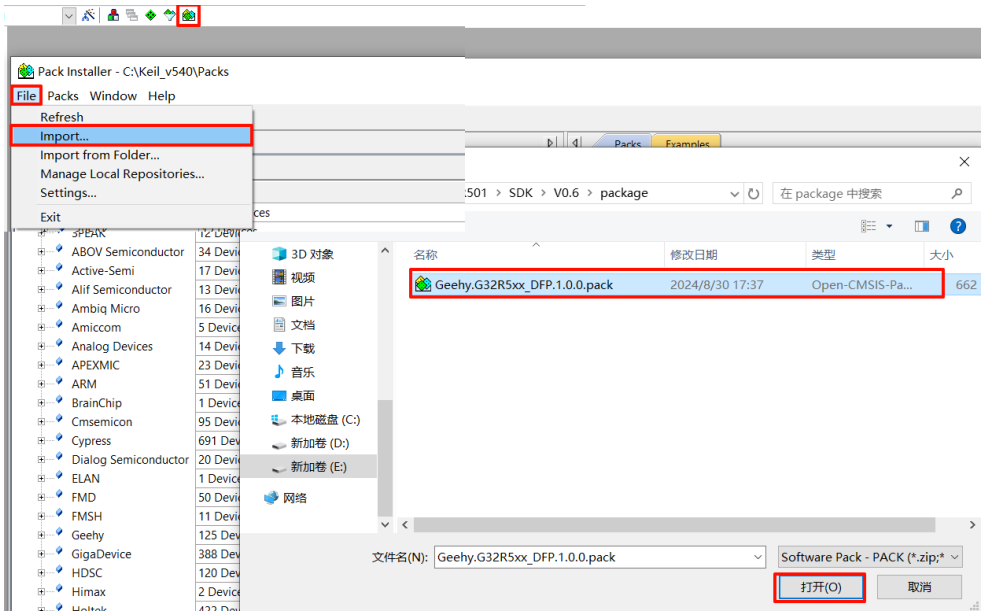
图 2 Geehy.G32R5xx_DFP.x.x.x.pack 点击安装



2. 使用 Pack Installer（如图 3）

- 打开 MDK-ARM v5.40。
- 在菜单栏中，点击“Project”->“Manage”->“Pack Installer”。
- 这将打开一个新的窗口，显示可用的支持包。
- 在新窗口中，选择“File”->“Import…”。
- 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。
- 点击“Open”进行安装。

图 3 Pack Installer 导入 Geehy.G32R5xx_DFP.x.x.x.pack



2.3.2. 打开示例工程

1. 启动 MDK-ARM 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”。

或是安装 MDK-ARM v5.40 后，可直接点击工程文件“.uvprojx 文件”打开即可。

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

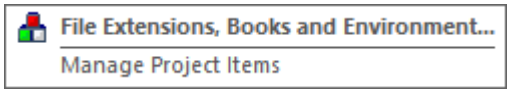
2.3.3. 工程建立

1. 启动 MDK-ARM：
   - 打开 MDK-ARM v5.40。
2. 创建新工程：
   - 点击菜单栏中的 “Project”->“New uVision Project”。
   - 选择工程保存的路径，输入工程名称，点击“Save”。
3. 选择微控制器：
   - 在弹出的对话框中，选择合适的 G32R501 系列微控制器型号，然后点击“OK”。

2.3.4. 文件导入

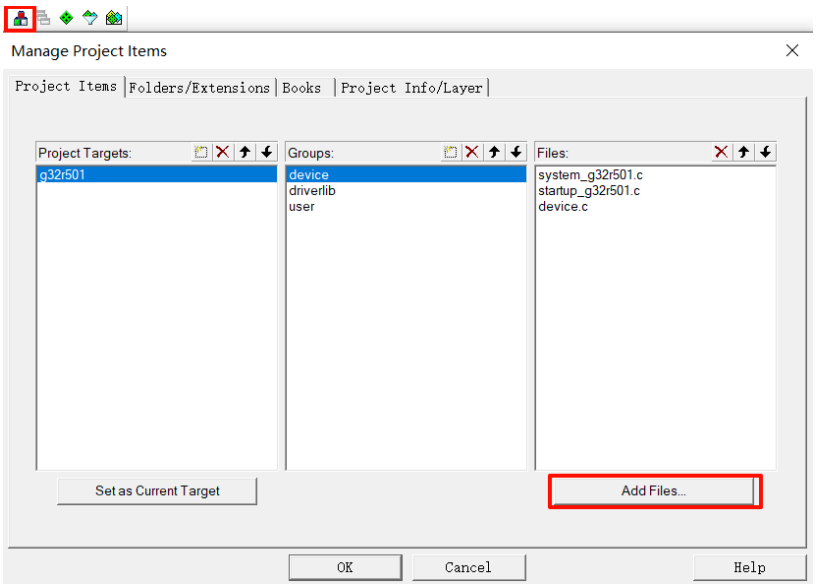
1. 打开工程视图：
   - 确保你的新工程已经在 Project 窗口中打开。
2. 添加现有文件：
   - 点击“File Extensions，Books and Environment…”

图 4 File Extensions



   - 点击“Source Group 1”或你希望添加文件的文件夹，选择“Add Files…”。
   - 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



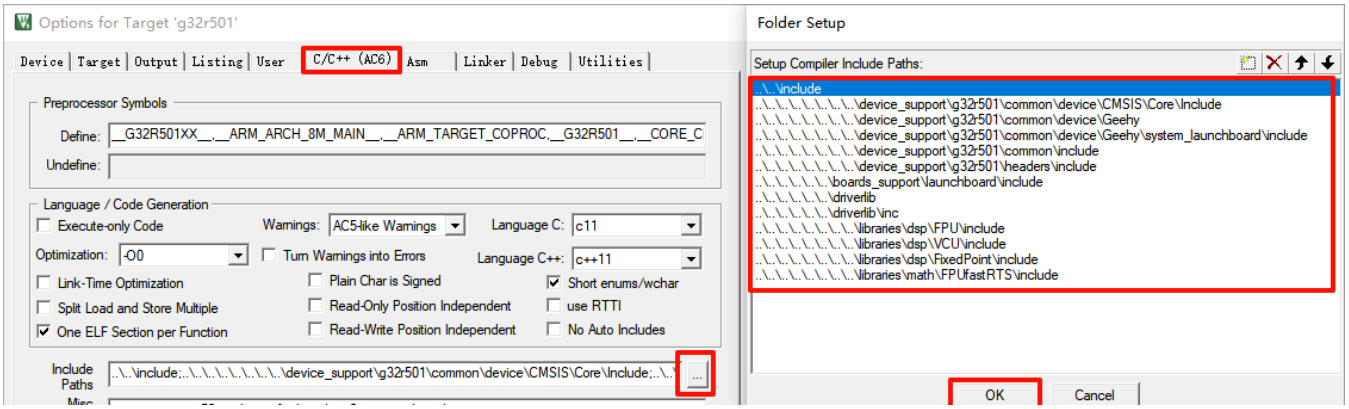
3. 新建文件（如需要）：

   - 如果需要创建新的源文件，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”。

4. 配置头文件路径（如图 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。

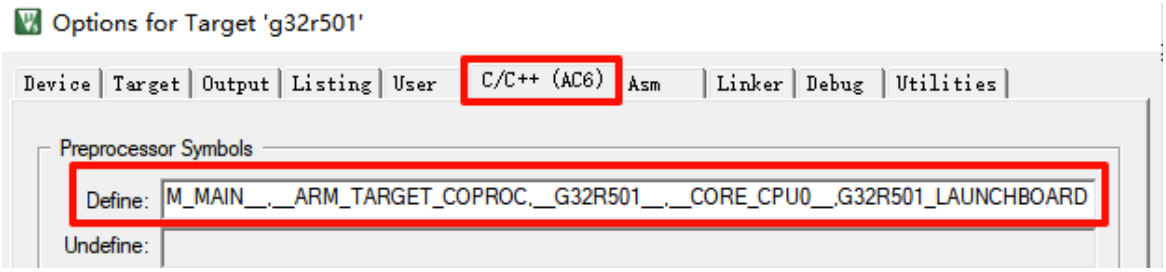
图 6 添加头文件路径



2.3.5. 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



2.3.6. 编译命令控制

在 MDK-ARM 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 3 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
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

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+cdecp0	Not included	Not included	Not included	Not included	Not included	Included

图 8 MDK 帮助文档

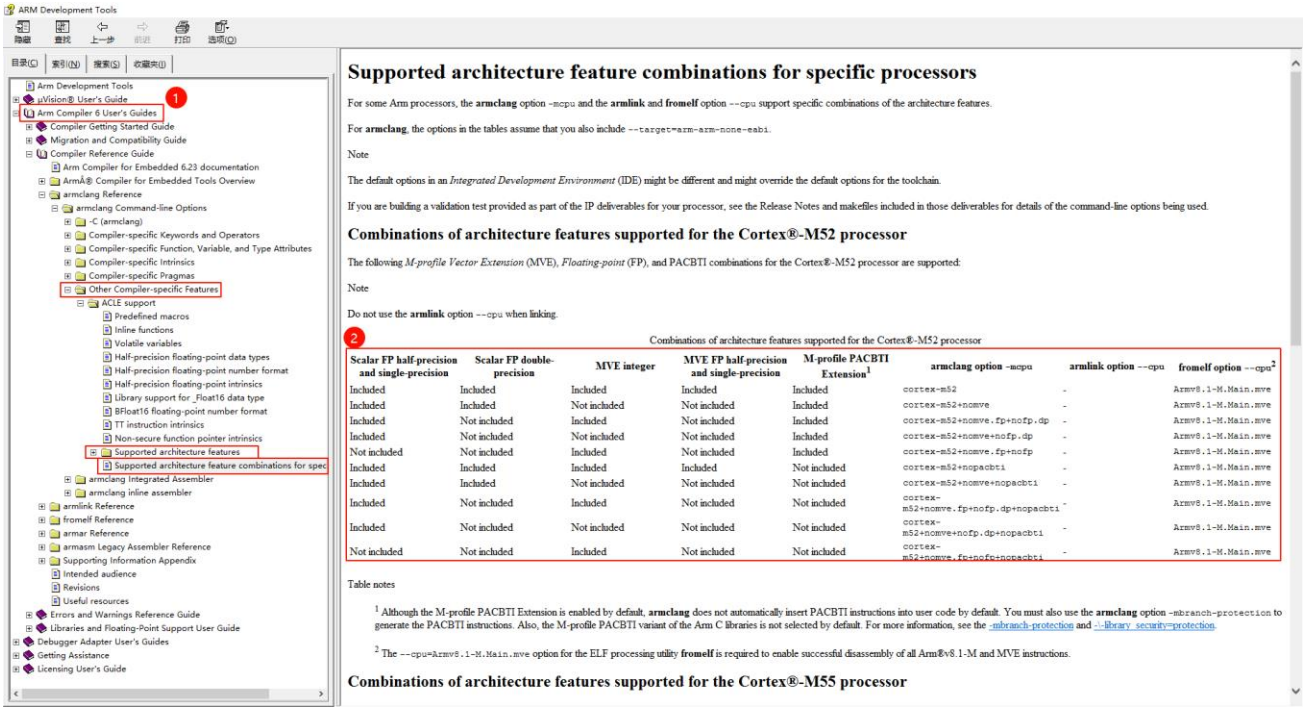
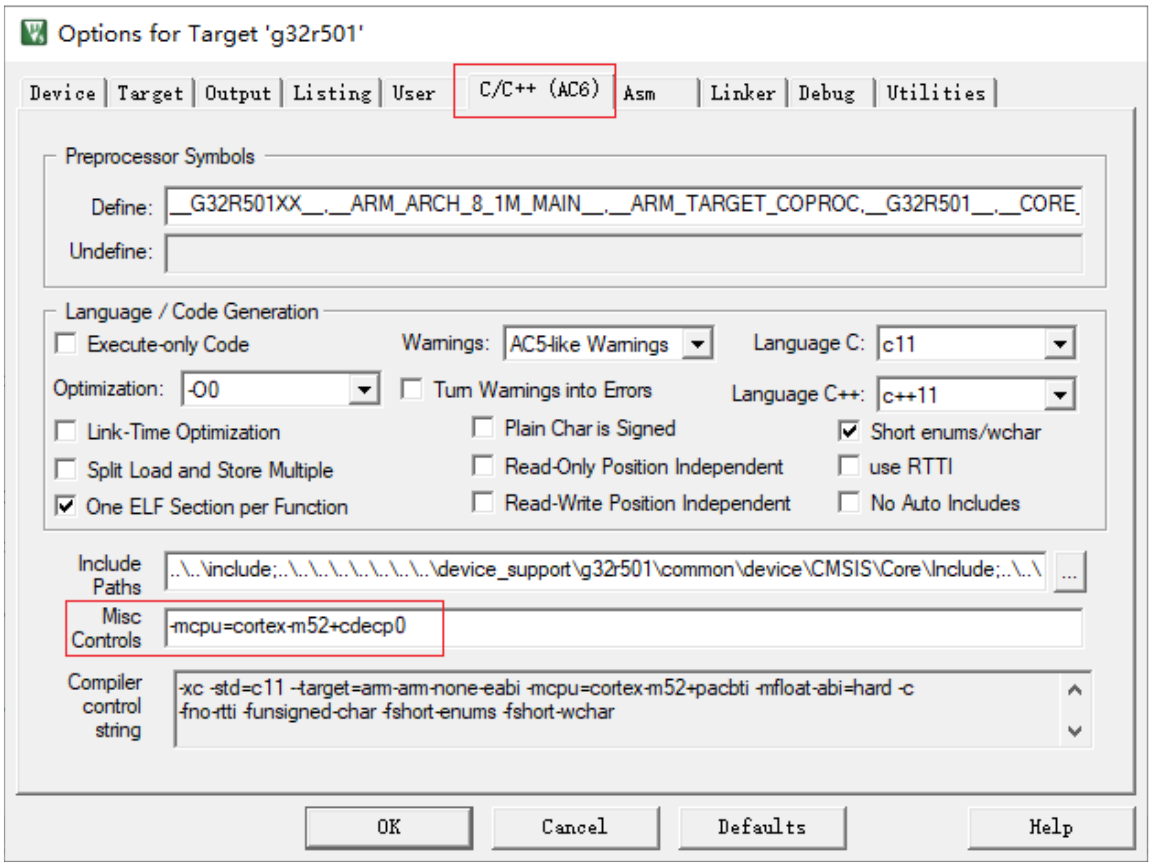


图 9 编译命令控制

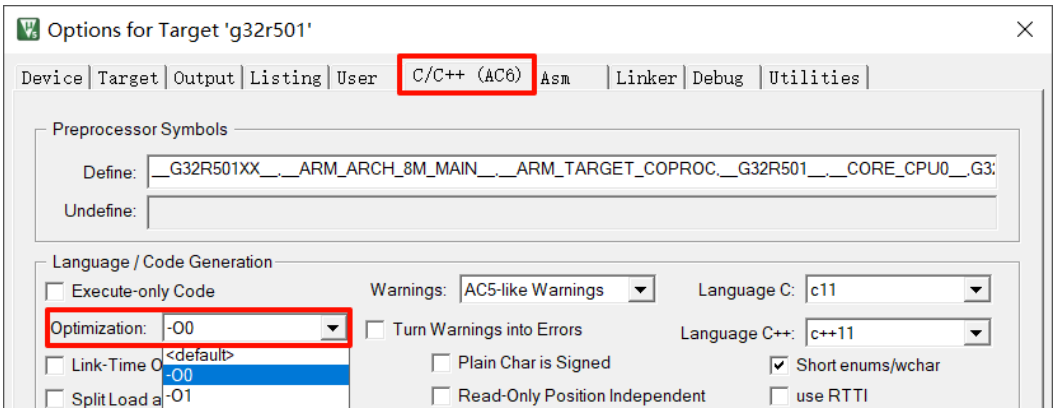


2.3.7. 编译优化等级设置

MDK-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：

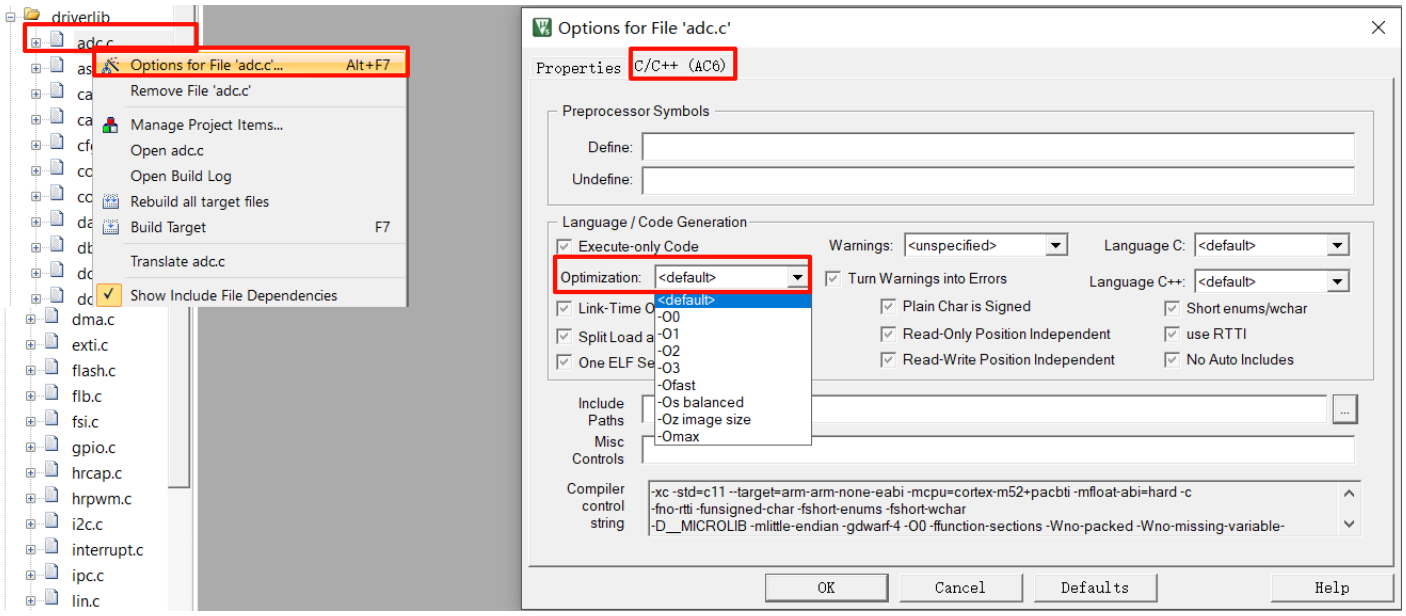
1. 全局优化等级设置：在“Options for Target”窗口中，选择“C/C++”选项卡，然后在“Optimization”下拉菜单中选择合适的优化等级。

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。

2.3.8. 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，或直接按下“F7”按钮）。

图 12 编译程序

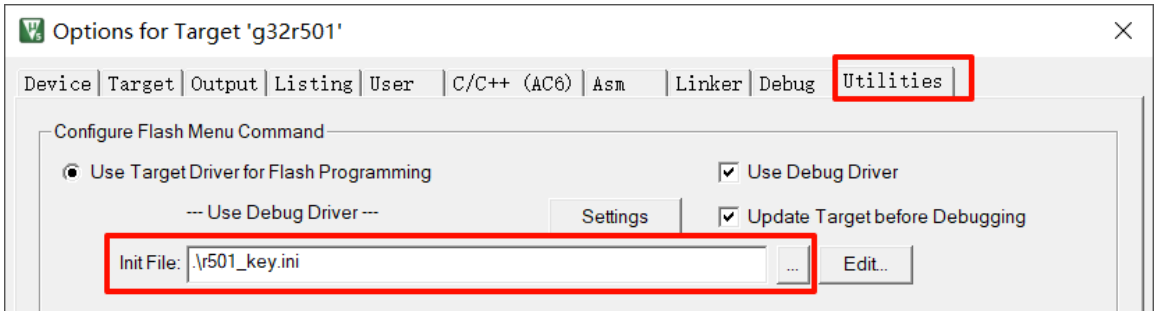
2. 等待编译过程完成，查看输出窗口中是否有错误或警告信息。如果有错误，根据提示进行相应的修改

2.3.9. 程序下载

1. 选择仿真器：
   - 点击菜单栏中的“Project”->“Options”。
   - 在弹出的对话框中，选择“Debug" 标签。
   - 在“Use”下拉菜单中，选择合适的调试仿真器（例如 Geehy-Link、J-Link（J-Link 的下载配置可参考**错误!未找到引用源。**小结）等）。
2. 配置下载脚本（仿真器选择为 Geehy-Link 时）：

- 打开项目选项：选择“Options for Target”。
- 选择下载工具：在“Utilities”选项卡中，选择合适的下载工具。
- 加载下载脚本：点击“Settings”按钮，加载特定的下载脚本文件。

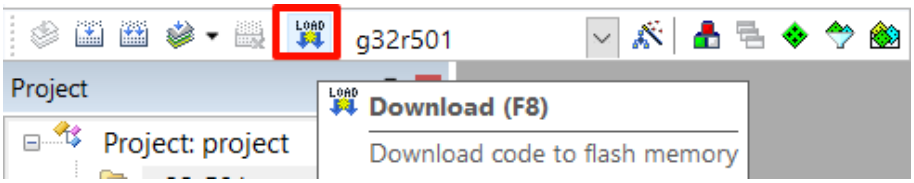
图 13 选择下载脚本



3. 下载程序：

- 在工具栏上，点击“Download”按钮（小箭头图标）或在菜单中选择“Flash”->“Download”（或直接按下“F8”）。

图 14 下载程序



- 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。

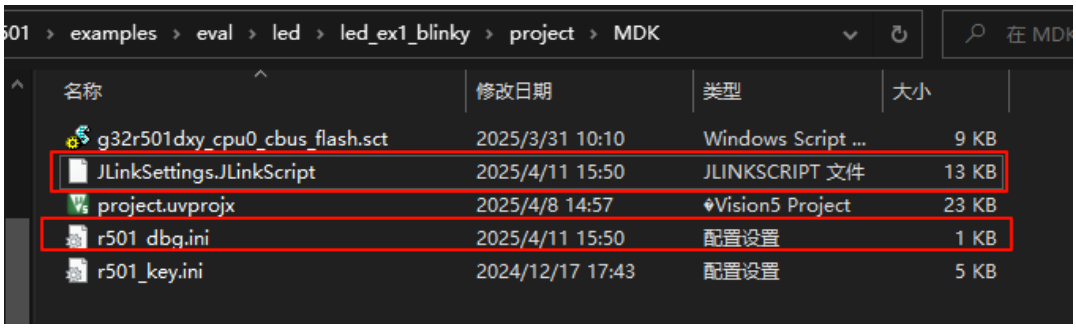
2.3.10. 程序仿真

2.3.10.1. J-Link 仿真

1. 打开项目选项：选择“Options for Target”。
2. 选择调试器：在“Debug”选项卡中，选择 J-Link 调试器。
3. 在目标工程文件夹下添加 r501_dbg.ini 与 JLinkSettings.JLinkScript 文件。

- JLinkSettings.JLinkScript 文件是一种类 C 的脚本语言，用于定制 J-Link 调试器的操作，JLinkScript 文件包括基本语法、自定义操作、API 函数、DLL 全局常量（变量），其语法与 C 语言类似。
- r501_dbg.ini 与 JLinkSettings.JLinkScript 文件位于 SDK/device_support/g32r501/common/Jlink/。

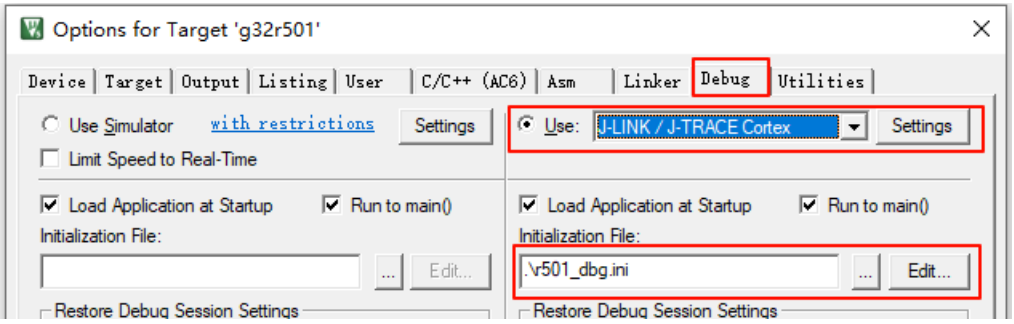
图 15 添加 J-Link 仿真文件



注意：r501_dbg.ini 文件与使用 GEEHY LINK 仿真时使用的 r501_dbg.ini 文件不同，使用时请注意区分。

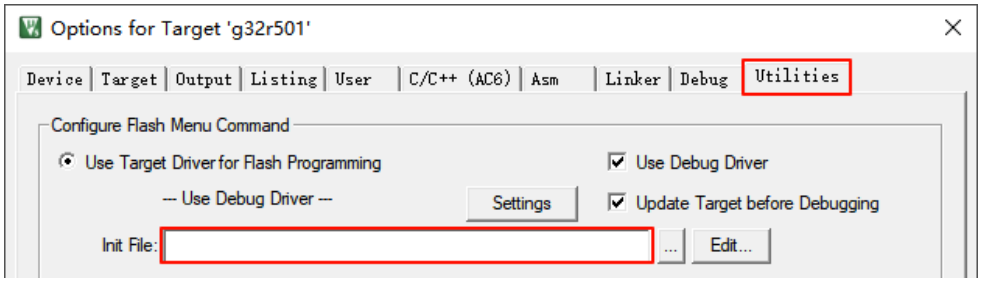
4. 加载仿真脚本：在“Settings”菜单中，点击“Load”按钮，加载特定的仿真脚本文件。

图 16 J-Link 仿真器及仿真脚本



5. 去除下载脚本：使用 J-Link 仿真时无需额外的下载脚本，请删除 Utilities 下载脚本“Init File”的设置。

图 17 去除 Utilities 下载脚本



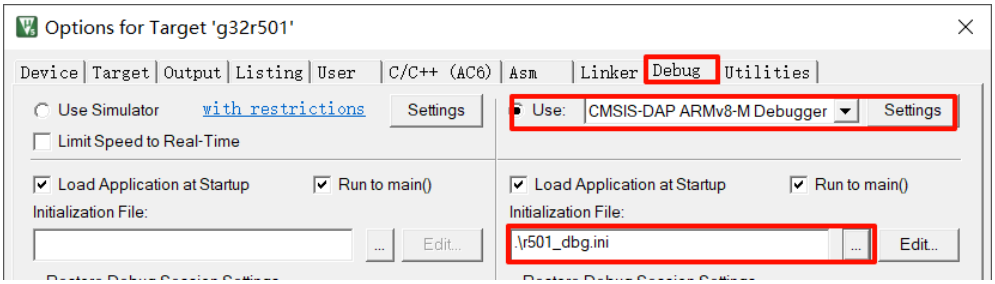
6. 启动调试：

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

2.3.10.2. GEEHY LINK 仿真

1. 打开项目选项：选择“Options for Target”。
2. 选择调试器：在“Debug”选项卡中，选择合适的调试器。
3. 加载仿真脚本：在“Settings”菜单中，点击“Load”按钮，加载特定的仿真脚本文件。

图 18 选择 DAP LINK 仿真器及仿真脚本



4. 启动调试：
   - 下载完成后，可以点击“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。

2.4. 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。

2.4.1. 文件格式支持

支持.c/.h 文件。

2.4.2. 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。

2.4.3. 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。以下是常见数据类型在 G32R501 和 Txx320F28004x 上的大小对比：

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

2.5. 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

2.5.1. 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。

2.5.2. 汇编编码格式要求

1. 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
```

2. 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;

}
```

2.6. 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。G32R501 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：
  - G32R501 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：
  - G32R501 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。

2.7. 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")
```

G32R501 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)*

3. IAR EW for Arm 开发工具链

3.1. 仿真器支持

请参考章节 2.1。

3.2. IDE 版本

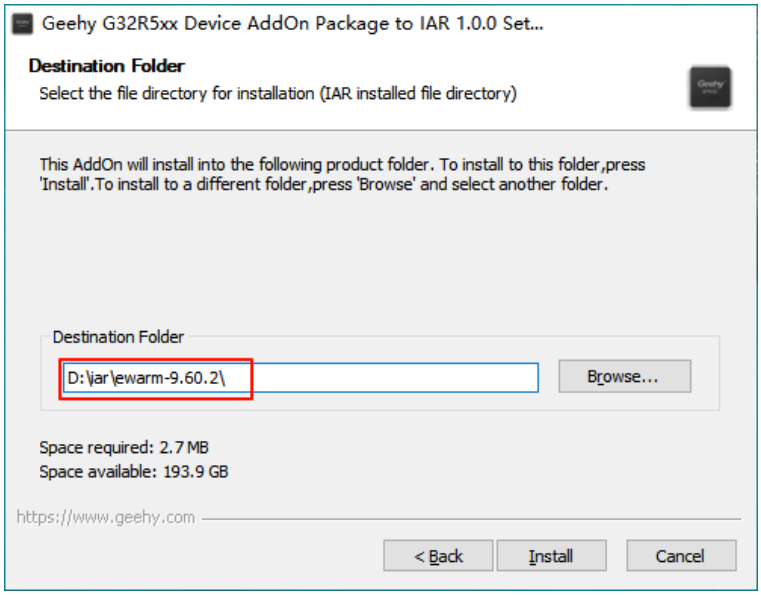
请确保使用 IAR EW for Arm 9.60.2 或更高版本的 IDE。

3.3. 安装芯片支持

在正式使用 IAR EW 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

1. 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 EW for Arm 的安装路径，如示例为：D:\iar\ewarm-9.60.2\。

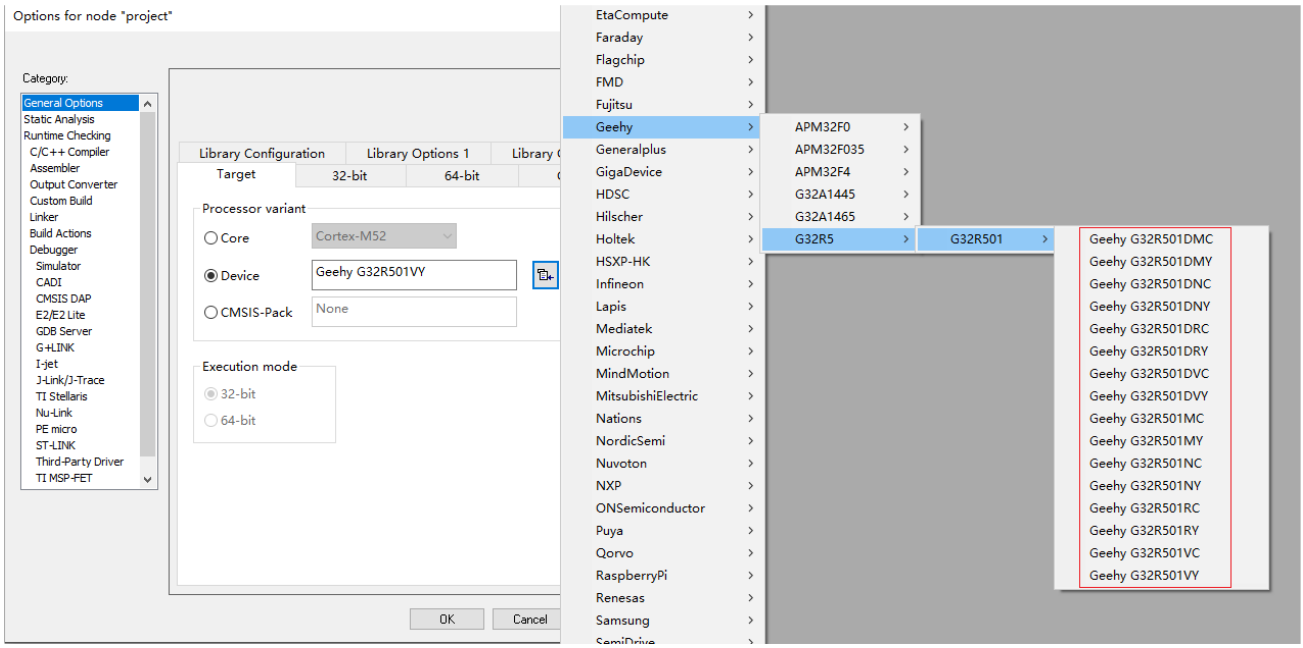
图 19 G32R5xx_AddOn_vx.x.x.exe 安装芯片支持



若软件无法获取电脑上的 IAR EW for Arm 安装路径，请手动进行选择。

2. 选择正确的路径并添加完毕芯片支持后，打开 IAR EW 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 20 G32R5 系列 MCU 列表



3.4. 工程操作

3.4.1. 打开示例工程

1. 启动 IAR EW 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”。

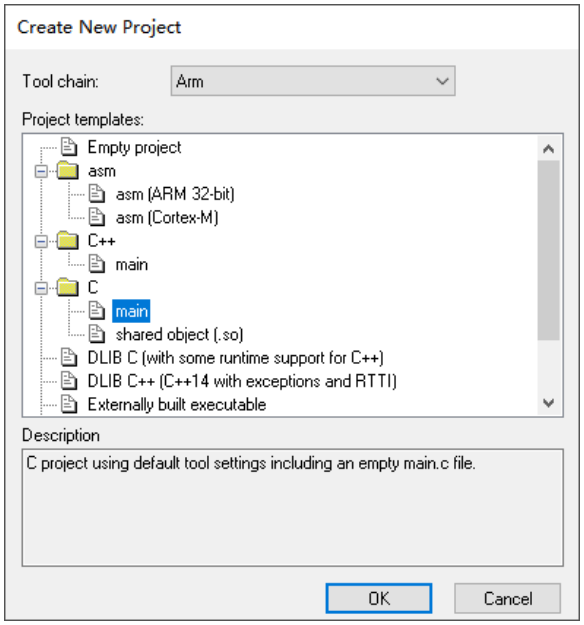
或是安装 IAR EW for Arm 9.60.2 后，可直接点击工程文件“.eww 文件”打开即可。

注意：以上步骤请在完成章节 3.3 安装芯片支持，否则 IAR EW for Arm 将提示芯片无法找到。

3.4.2. 工程建立

1. 启动 IAR EW for Arm：
   - 打开 IAR EW for Arm 9.60.2。
2. 创建新工程：
   - 点击菜单栏中的 “Project”->“Create New Project”。
   - 选择“Tool chain”为“Arm”。
   - 选择“C”下的“main”后点击“OK”。
   - 选择工程保存的路径，输入工程名称，点击“Save”。

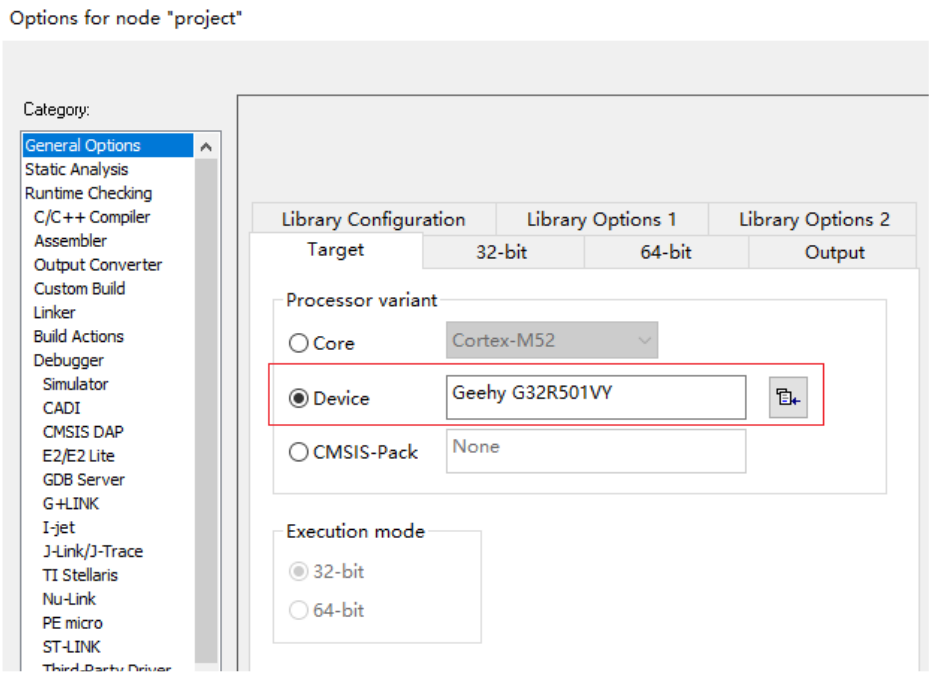
图 21 Create New Project



3. 选择微控制器：

- 在右键工程名，选择“Options…”。
- 在“General Options”下的“Target”下选择“Device”。
- 选择合适的 G32R501 系列微控制器型号，然后点击“OK”。

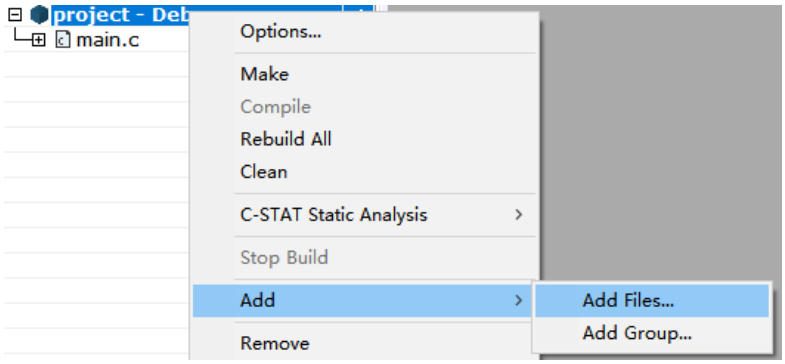
图 22 选择微控制器



3.4.3. 文件导入

1. 打开工程视图：
   - 确保你的新工程已经在 Project 窗口中打开。
2. 添加现有文件：
   - 在右键工程名，选择“Add”，然后选择“Add Files”。

图 23 File Extensions

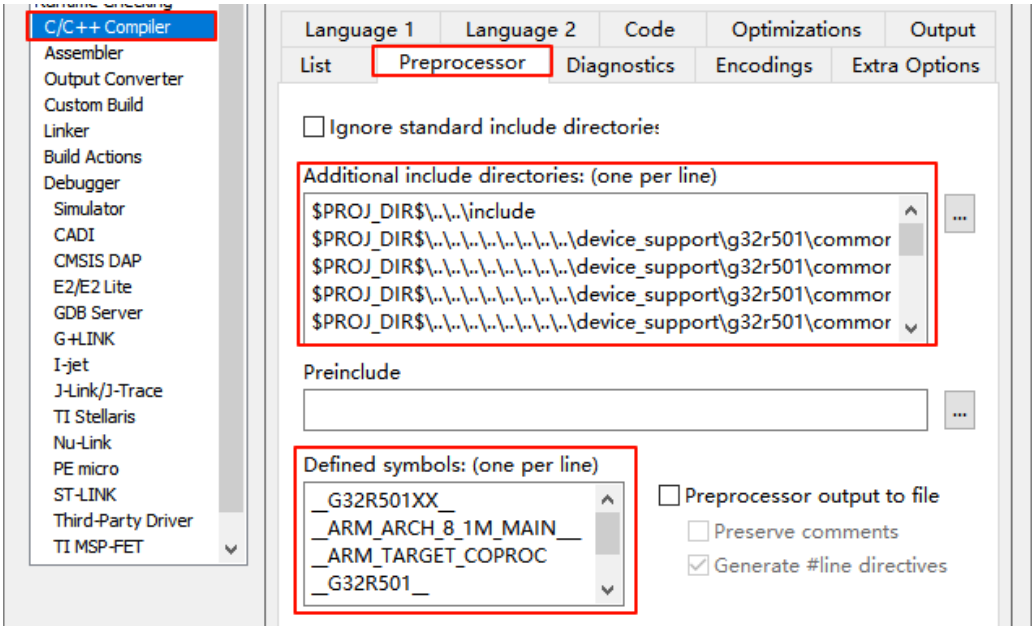


   - 在弹出的文件浏览器中，找到并选择要导入的源文件（如.c 和.h 文件），然后点击“Add”。
3. 新建文件（如需要）：
   - 如果需要创建新的源文件，点击工具栏上的“File”，选择“New File”。
   - 在 IDE 中将出现一个文件，按下“CTRL + S”保存文件。
   - 输入文件名，选择文件类型（如 C 文件或汇编文件），然后参考“添加文件”的操作步骤。

3.4.4. 配置头文件路径与宏定义

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

图 24 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。

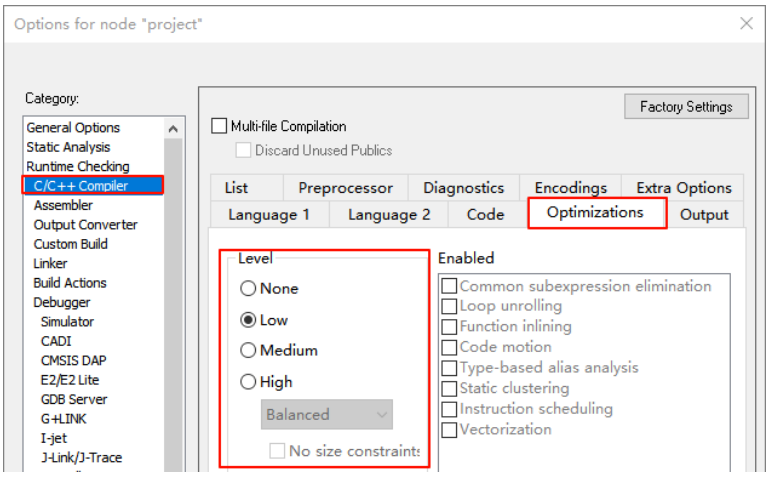
4. 在 Defined symbols: (one per line)所需的宏定义。

3.4.5. 编译优化等级设置

IAR EW 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：

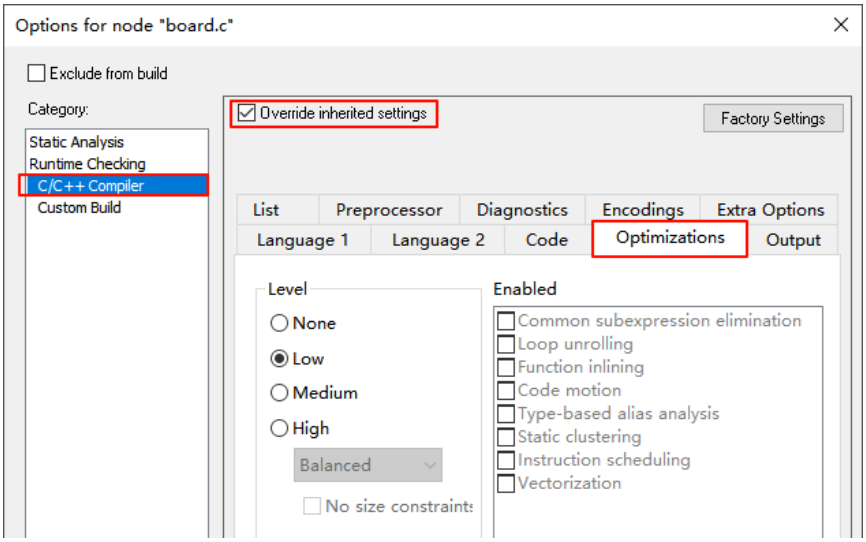
1. 全局优化等级设置：打开“C/C++ Compiler”选项卡，然后在“Optimizations”中选择合适的优化等级。

图 25 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 26 单文件优化等级设置

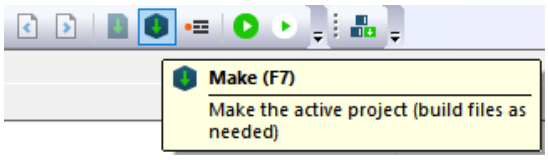


3. 单函数优化等级设置：可以在函数前使用特定的编译器指令进行优化设置，例如使用_Pragma("optimize=none")声明不优化。

3.4.6. 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，或直接按下“F7”按钮）。

图 27 编译程序

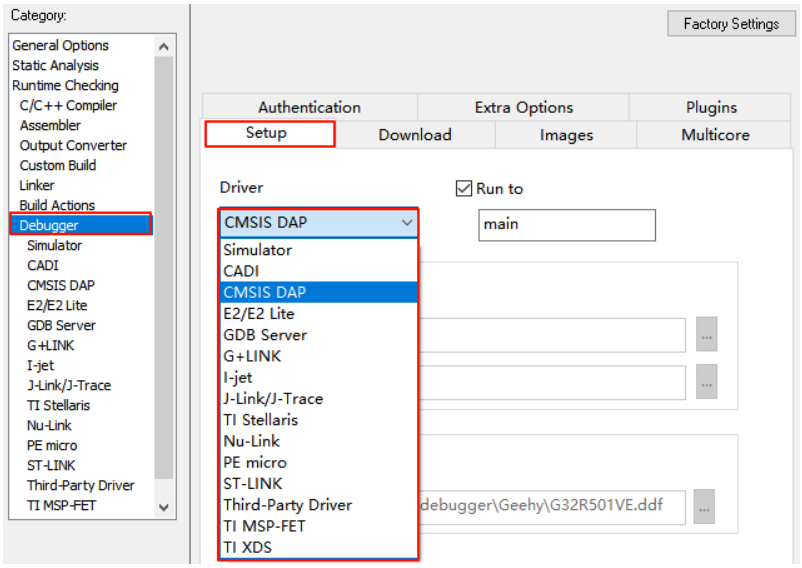


2. 等待编译过程完成，查看输出窗口中是否有错误或警告信息。如果有错误，根据提示进行相应的修改

3.4.7. 程序仿真与下载

1. 选择仿真器：
   - 在右键工程名，选择“Options…”。
   - 在弹出的对话框中，选择“Debugger" 选项卡。
   - 在“Setup”标签下的“Driver”下拉菜单中，选择合适的调试仿真器（例如 Geehy-Link（CMSIS DAP）、J-Link 等）。

图 28 选择仿真器

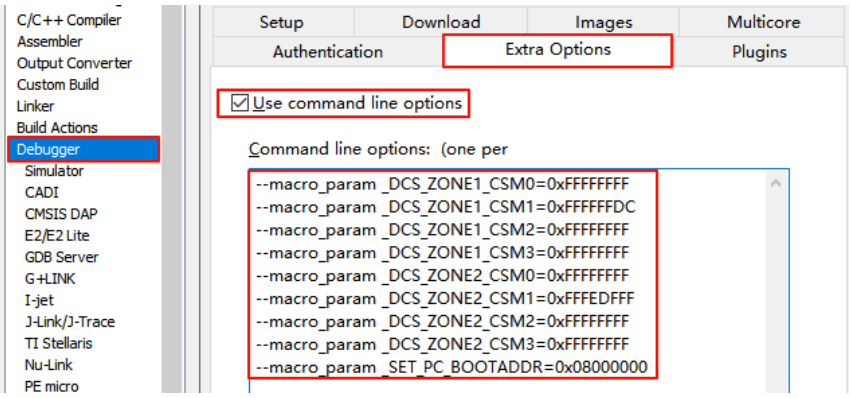


2. 配置仿真指令：

由于 G32R5 系列 MCU 支持 DCS 加密，所以需要配置相应的指令，才能正常仿真。

- 在右键工程名，选择“Options…”。
- 在弹出的对话框中，选择“Debugger" 选项卡。
- 在“Extra Options”标签下勾选“Use command line options”，并在“Command line options”添加指令
  - “--macro_param _DCS_ZONE1_CSM0=0xFFFFFFFF”等，设置 DCS 秘钥。
  - “--macro_param _SET_PC_BOOTADDR=0x08000000”，设置启动地址。

图 29 配置仿真指令



3. 擦除和下载程序：

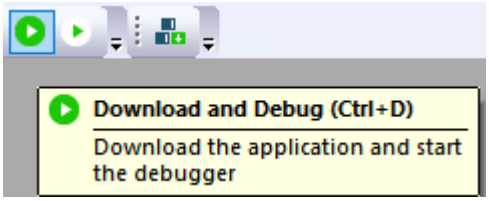
- 在工具栏上，点击“Project”在菜单中选择“Download”
  - “Erase memory”擦除芯片 Flash。

- “Download active application”下载本工程的程序至芯片。
- “Download file...”下载其他程序指芯片

4. 程序仿真：

- 点击工具栏的“Download and Debug”按钮或按下“Ctrl+D”即可启动仿真。

图 30 IAR Download and Debug



- 在调试模式下，可以设置断点、查看变量、单步执行等操作。

5. 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。

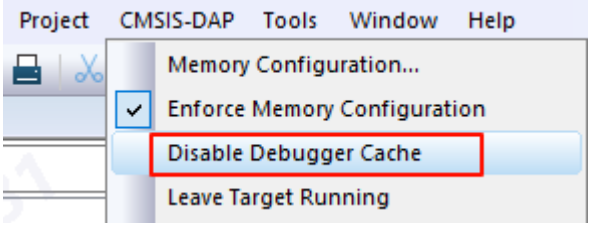
图 31 ddf 文件(节选)

```
[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：

1) 使能 Disable Debugger Cache

图 32 Disable Debugger Cache

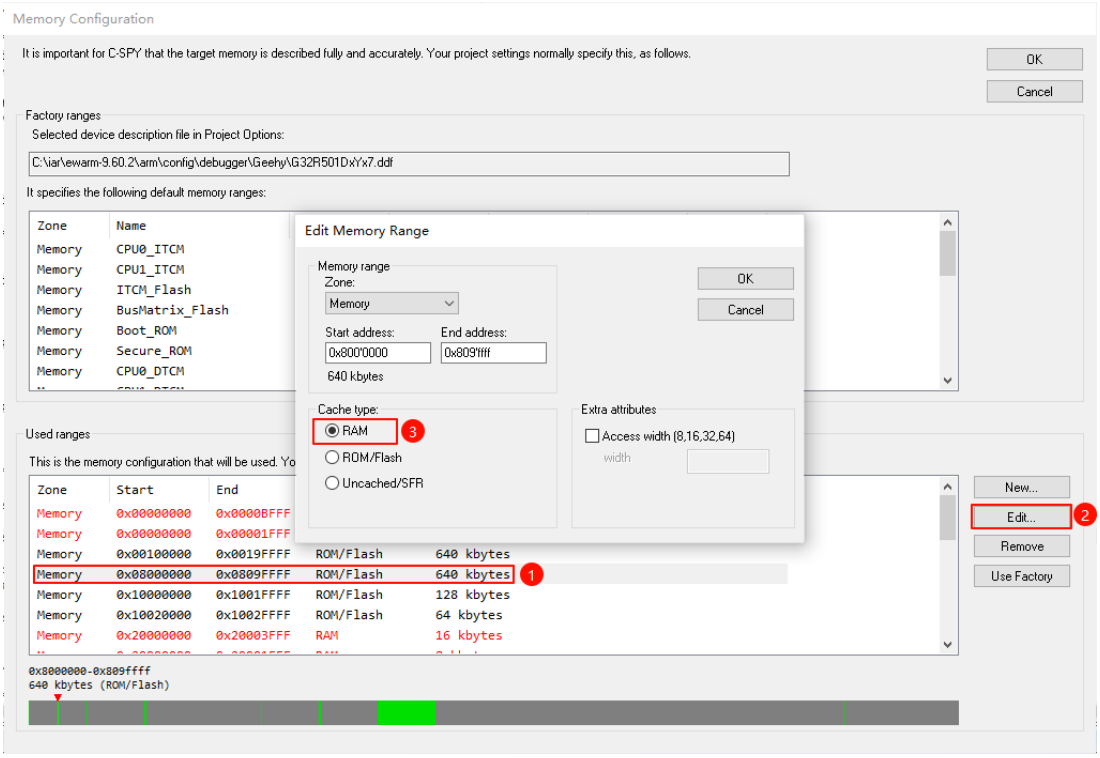


2) 修改 Flash 对应 Memory 区域的 Cache Type 为 RAM

点击 Memory Confoguration，进入 Memory Confoguration 窗口，修改 Cache

Type。

图 33 修改 Cache Type



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

图 34 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
Memory =   SRAM1               Memory      0x20100000   0x2011FFFF   RW
Memory =   SRAM2               Memory      0x20200000   0x2021FFFF   RW
Memory =   SRAM3               Memory      0x20300000   0x2031FFFF   RW
Memory =   APB0_Peripherals    Memory      0x40000000   0x4000FFFF   W
Memory =   APB1_Peripherals    Memory      0x40010000   0x4001FFFF   W
Memory =   APB2_Peripherals    Memory      0x40020000   0x4002FFFF   W
Memory =   DEMUX0_AHB          Memory      0x40030000   0x4003FFFF   W
Memory =   APB3_Peripherals    Memory      0x50000000   0x5000FFFF   W
Memory =   DEMUX1_AHB0         Memory      0x50010000   0x50027FFF   W
Memory =   DEMUX1_AHB1         Memory      0x60000000   0x6FFFFFFF   W
Memory =   APB4_Peripherals    Memory      0x50100000   0x50103FFF   W
Memory =   APB5_Peripherals    Memory      0x50104000   0x5010FFFF   W
Memory =   DEMUX2_AHB          Memory      0x50110000   0x5011FFFF   W
Memory =   PPB                 Memory      0xE0000000   0xE00FFFFF   W
```

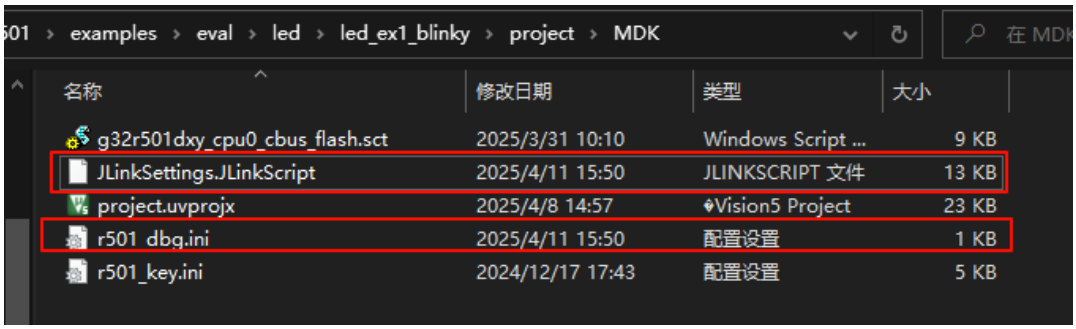
更多内容请参考：调试时 Memory 窗口中 Flash 内容不更新

3.5. C 语言兼容性

3.5.1.1. 请参考章节 2“J-Link 仿真

7. 打开项目选项：选择“Options for Target”。
8. 选择调试器：在“Debug”选项卡中，选择 J-Link 调试器。
9. 在目标工程文件夹下添加 r501_dbg.ini 与 JLinkSettings.JLinkScript 文件。
   - JLinkSettings.JLinkScript 文件是一种类 C 的脚本语言，用于定制 J-Link 调试器的操作，JLinkScript 文件包括基本语法、自定义操作、API 函数、DLL 全局常量（变量），其语法与 C 语言类似。
   - r501_dbg.ini 与 JLinkSettings.JLinkScript 文件位于 SDK/device_support/g32r501/common/Jlink/。

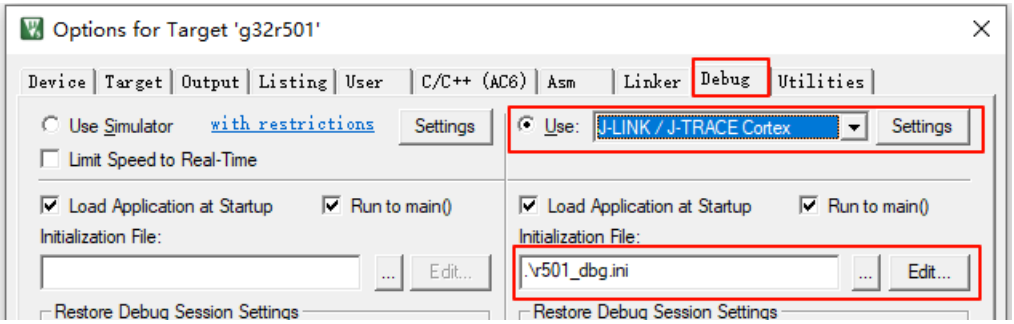
图 15 添加 J-Link 仿真文件



注意：r501_dbg.ini 文件与使用 GEEHY LINK 仿真时使用的 r501_dbg.ini 文件不同，使用时请注意区分。

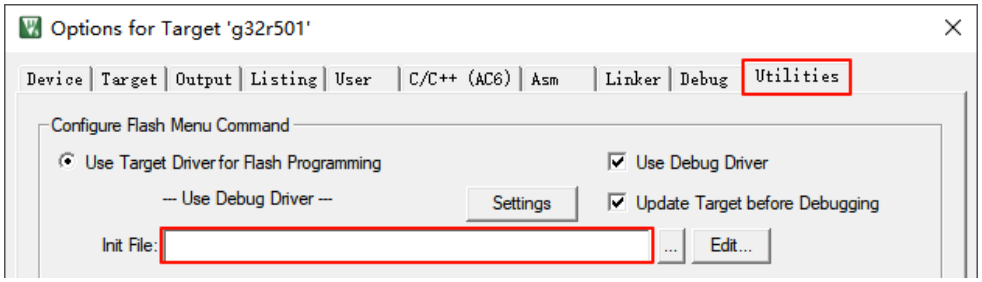
10. 加载仿真脚本：在“Settings”菜单中，点击“Load”按钮，加载特定的仿真脚本文件。

图 16 J-Link 仿真器及仿真脚本



11. 去除下载脚本：使用 J-Link 仿真时无需额外的下载脚本，请删除 Utilities 下载脚本“Init File”的设置。

图 17 去除 Utilities 下载脚本



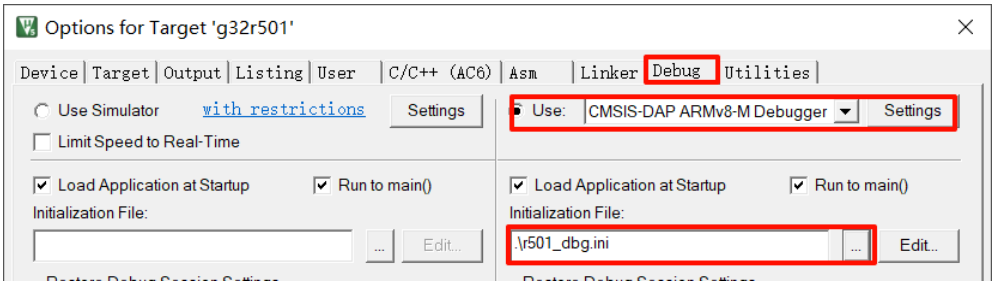
12. 启动调试：

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

3.5.1.2. GEEHY LINK 仿真

5. 打开项目选项：选择“Options for Target”。
6. 选择调试器：在“Debug”选项卡中，选择合适的调试器。
7. 加载仿真脚本：在“Settings”菜单中，点击“Load”按钮，加载特定的仿真脚本文件。

图 18 选择 DAP LINK 仿真器及仿真脚本



8. 启动调试：

- 下载完成后，可以点击“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。

C 语言兼容性”。

3.6. 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

3.6.1. 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。

3.6.2. 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;

}
```

3.7. 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。G32R501 在 IAR EW 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：
  - G32R501 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：
  - G32R501 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。

3.8. RAM 运行

请参考章节 2.7。

4. Eclipse

4.1. 仿真器支持

- 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）

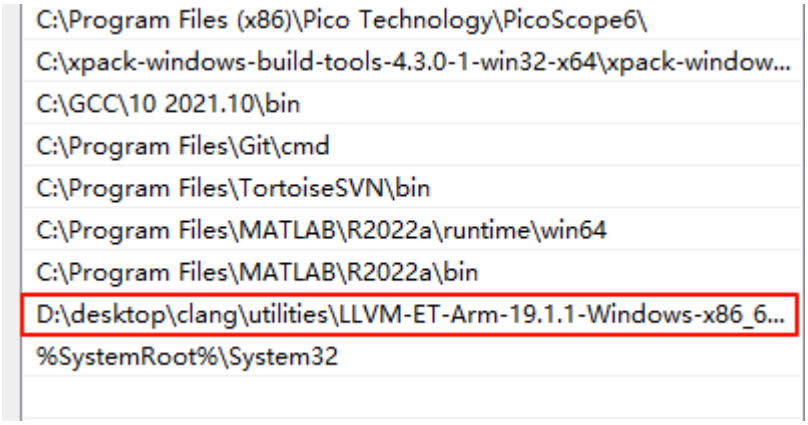
4.2. IDE 版本

请确保使用 Eclipse 4.35 或更高版本的 IDE。

4.3. LLVM_For_ARM_Toolchain

1. 下载 LLVM_For_ARM_Toolchain 编译器
   - 从 LLVM_For_ARM_Toolchain 仓库：https://github.com/ARM-software/LLVM-embedded-toolchain-for-Arm，下载 LLVM-ET-Arm-19.1.1-Windows-x86_64 编译器。
2. 添加环境变量
   - 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）
   - 将文件夹中的 bin 添加到系统环境变量中

图 35 添加系统环境变量



4.4. GDB 服务

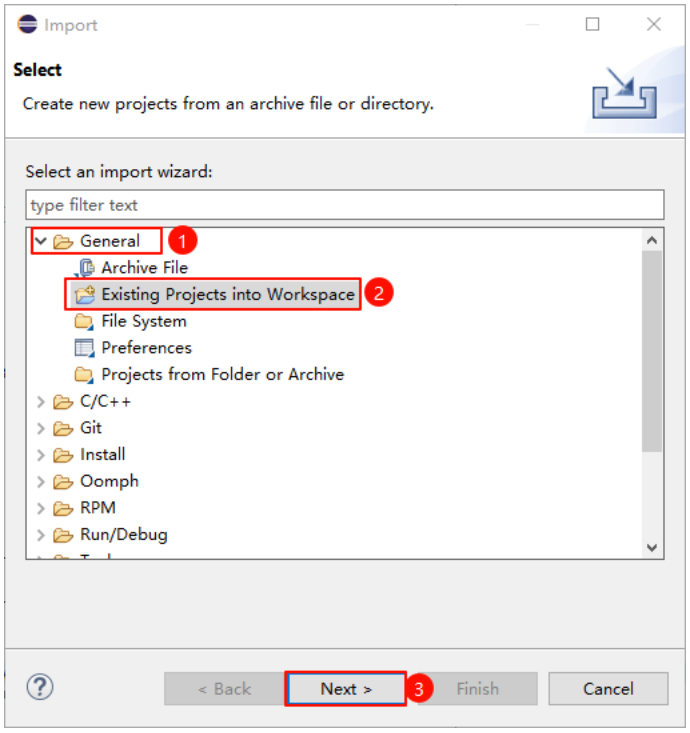
这里建议使用 Arm 提供的 arm-none-eabi-gdb.exe。示例使用的 arm-none-eabi-gdb.exe 版本是 14.2。

4.5. 工程操作

4.5.1. 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 36 Import project



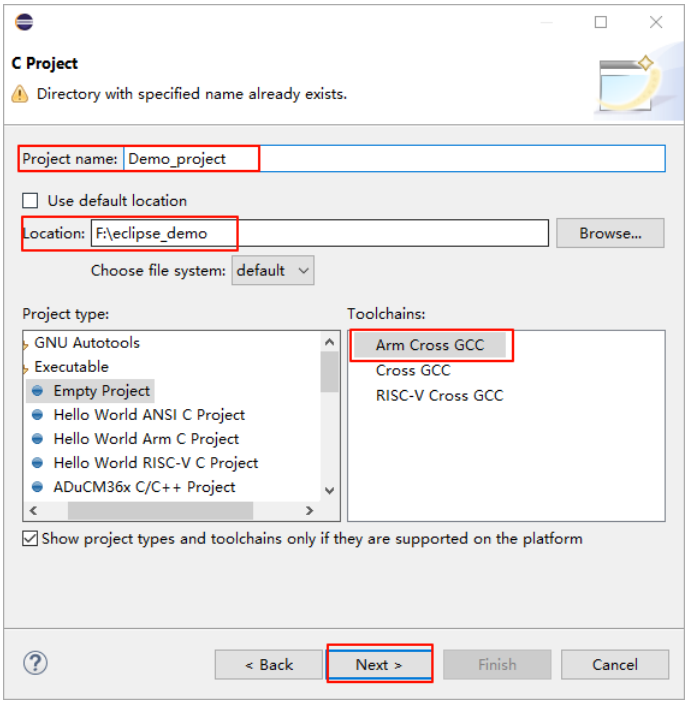
3. 点击“Select root directory”，浏览到你提供的 SDK 工程文件的路径，选择对应的工程文件夹，然后点击“Finish”。

注意：以上步骤请在完成章节 4.3 的 LLVM 编译配置。

4.5.2. 工程建立

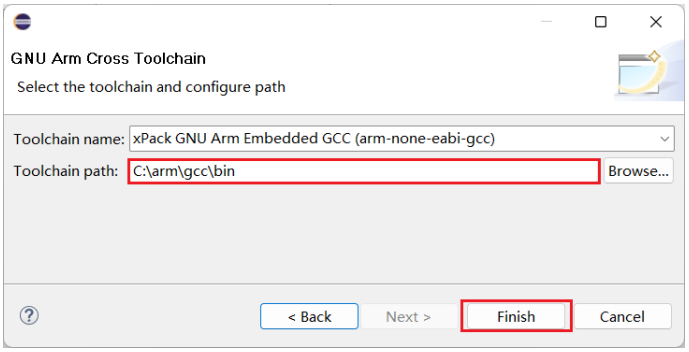
1. 启动 Eclipse：
   - 打开 Eclipse 4.35。
2. 创建新工程：
   - “File->New”下选择新建 C/C++ Project，选择 C Managed Build。
   - 输入工程名，配置工程类型，建议将该工程放到 Project 目录下。配置编译链，编译链选择为 Arm Cross GCC。

图 37 配置工程



- 若 Eclipse IDE 的 Arm Toolchains 已正确配置，这里将会自动选择路径。若未正确配置，可点击 Browse 选择对应的绝对路径。

图 38 设置 Arm Toolchains Path

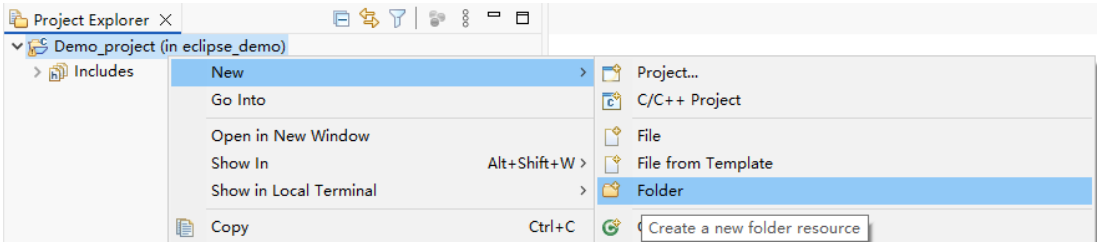


- 点击“Finish”，完成 Project 的建立。

4.5.3. 文件导入

1. 右键所在工程文件夹，建立虚拟文件夹。

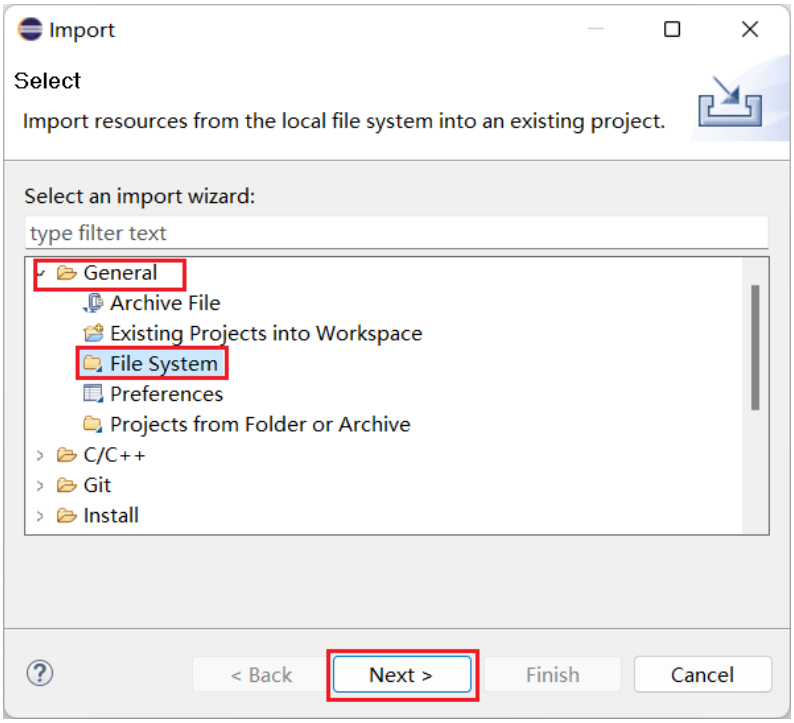
图 39 新建文件夹



2. 添加现有文件：

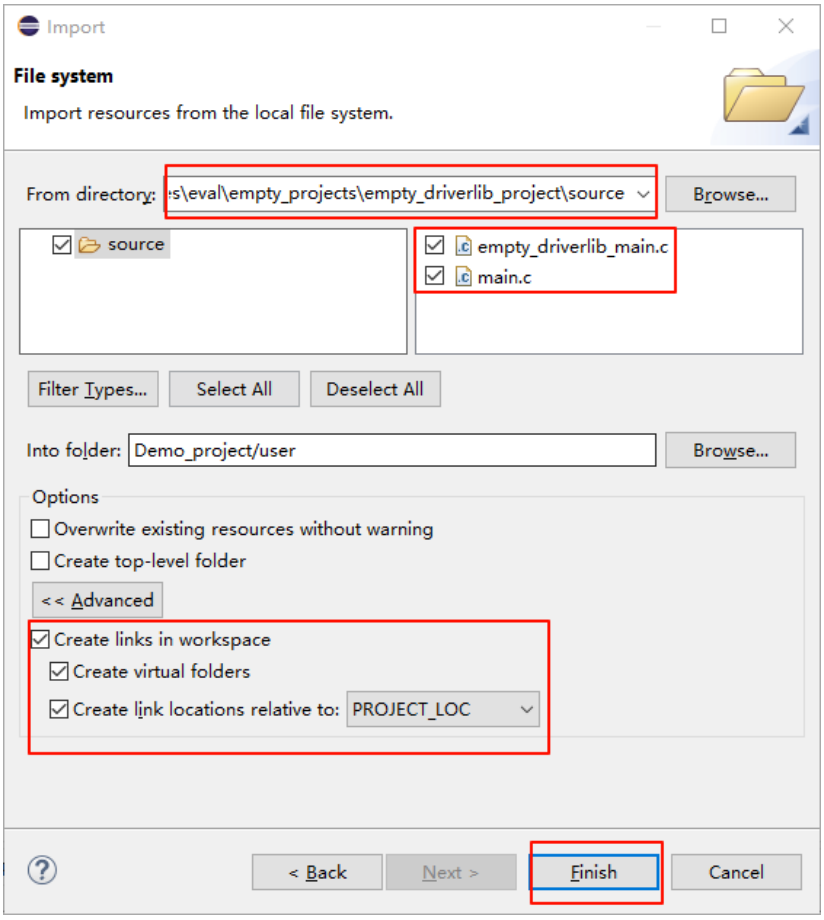
- 选中文件夹，右键选择 Import 选项，可直接导入文件。

图 40 Import file



- 在导入文件时，选择“File System”作为导入方式，在弹出的文件路径选择对话框中，选择需要导入的文件路径，然后勾选需要导入的文件。

图 41 Select file



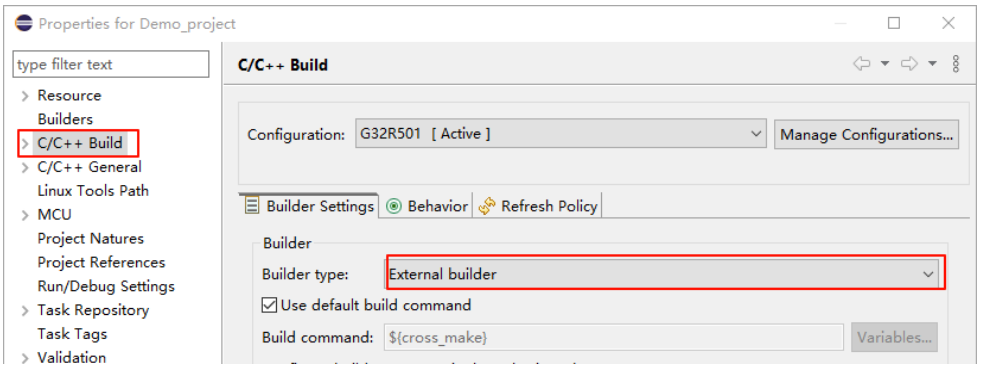
3. 新建文件（如需要）：

   - 如果需要创建新的源文件，点击“File”->“New”，选择要新建的源文件。
   - 选择新建的源文件的文件夹与文件名。

4.5.4. 编译配置

1. 在右键工程名，选择“Properties”。
2. 选择“External builder”配置：

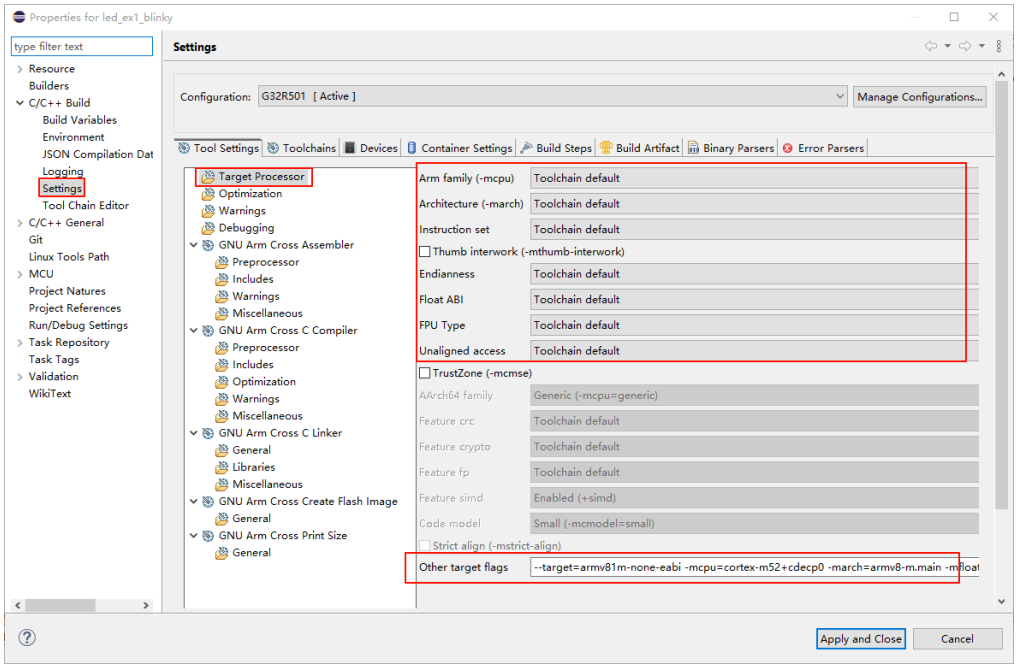
图 42 External builder



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

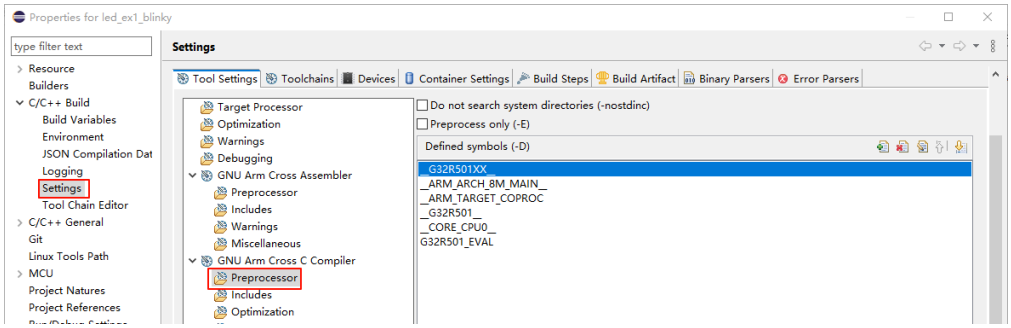
   - 将所有可下拉的配置项选为默认，手动添加命令行：--target=armv81m-none-eabi -mcpu=cortex-m52 -mfpu=none -fno-exceptions -fno-rtti -lcrt0-semihost –lsemihost（可根据芯片实际参数进行修改）

图 43 Target Processor



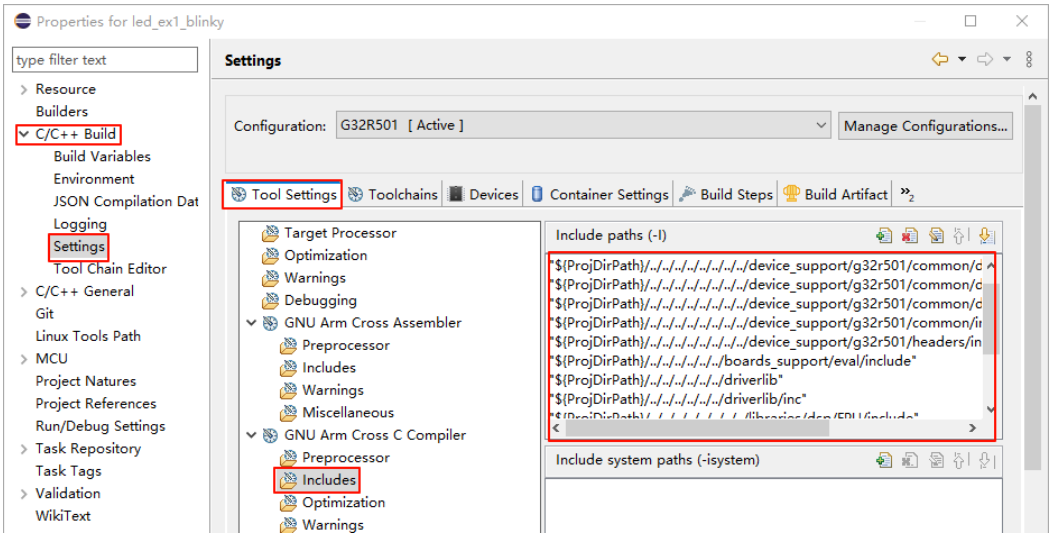
4. 在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“GNU Arm Cross C Compiler”->“Preprocessor”。

图 44 添加宏定义



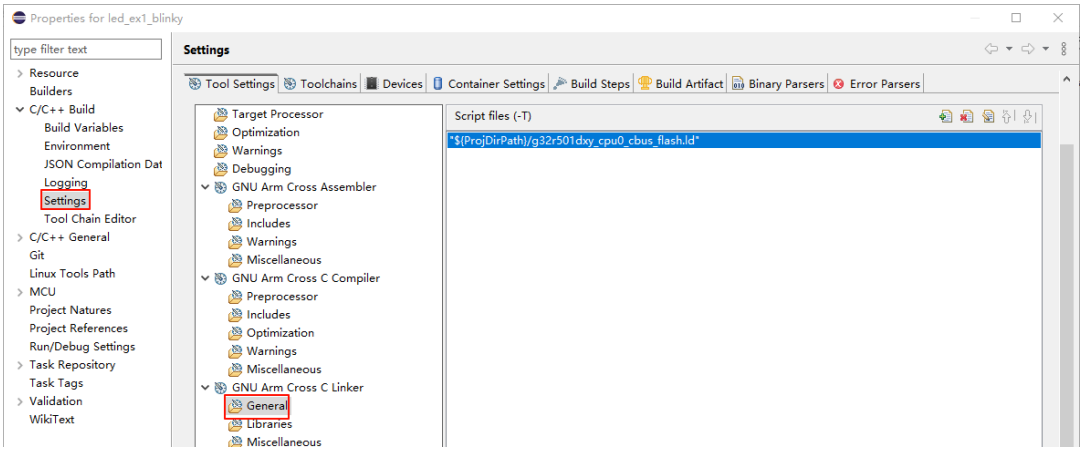
5. 在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“GNU Arm Cross C Compiler”->“Includes”。

图 45 配置头文件路径



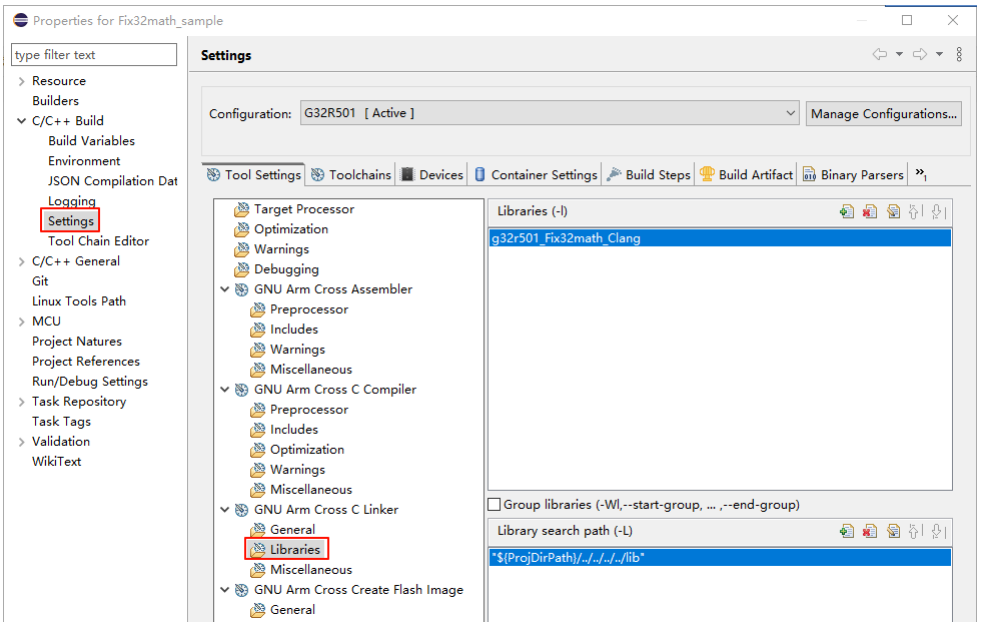
6. 在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“GNU Arm Cross C Linker”->“General”

图 46 添加链接文件



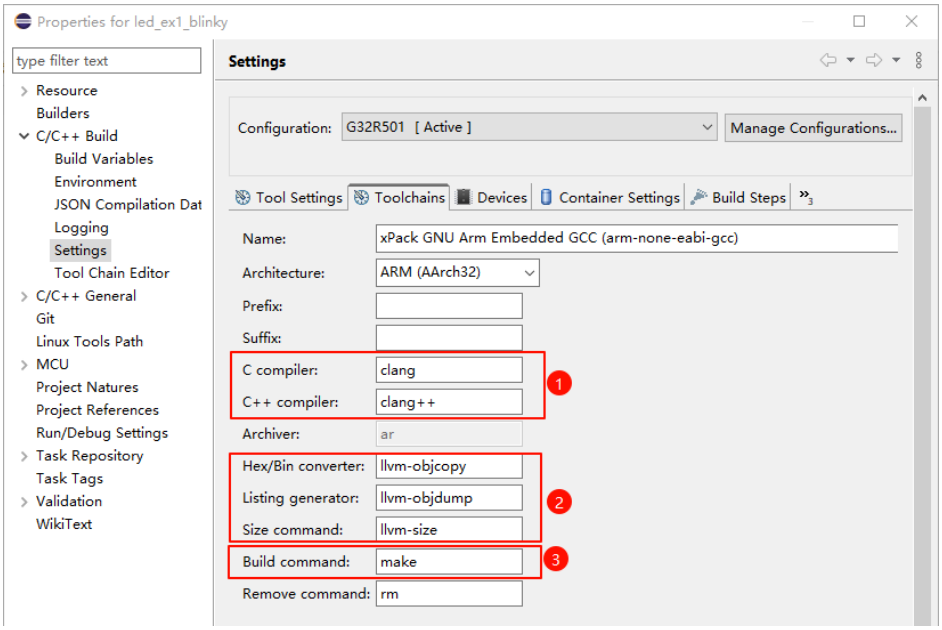
7. 在“C/C++ Build”选项卡下选择“Settings”->“Tool Settings”->“GNU Arm Cross C Linker”->“Libraries”

图 47 添加 libraries



8. 配置“ToolChain”

图 48 配置 ToolChain



1) 使用 llvm 内嵌的 clang 编译器。

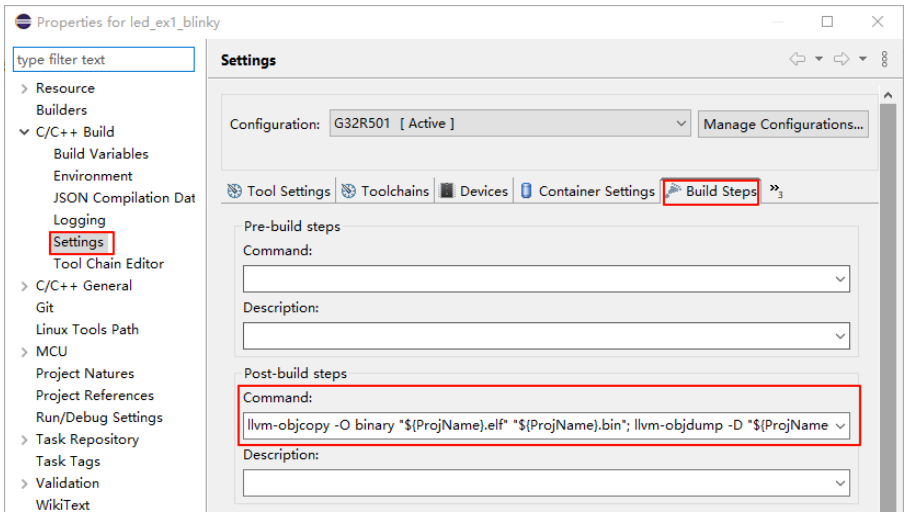
2) llvm 工具链内置工具。

3) 使用 xpack 内置的 make 编译器。

9. 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 49 Bulid Step

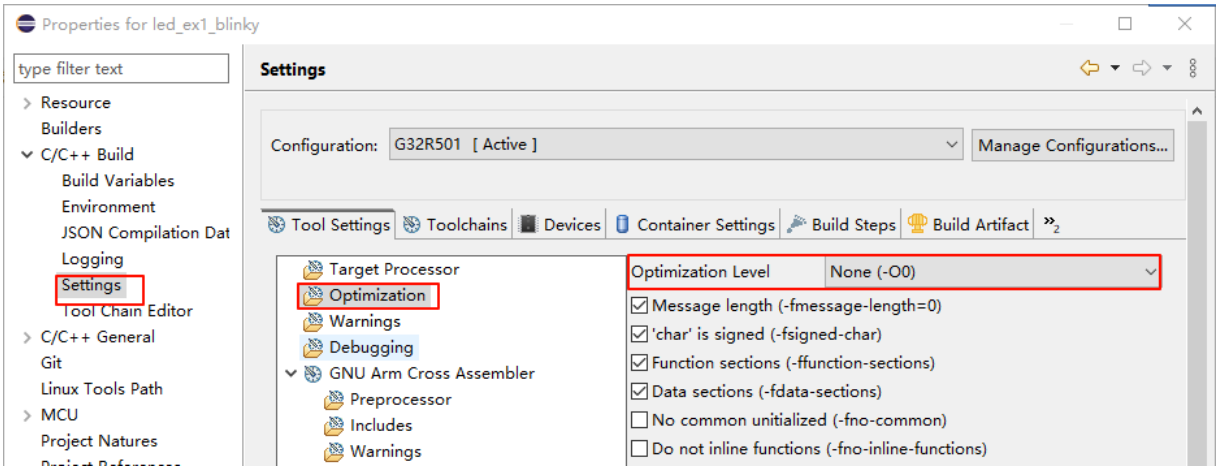


4.5.5. 编译优化等级设置

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 50 全局优化等级设置



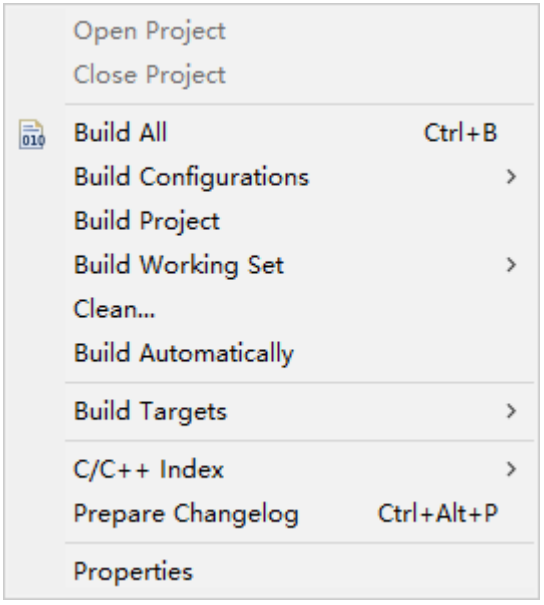
2. 单函数优化等级设置：可以在函数前使用特定的编译器指令进行优化设置，例如使用 _Pragma("optimize=none")声明不优化。

4.5.6. 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 51 编译程序



注意：在进行编译前，务必先保存当前工程。否则，编译的可能是上一次保存的工程，不是当前修改后的工程。为确保编译的正确性，需要在修改后进行工程 clean,然后再进行编译。编译完成后，会生成对应的 elf、hex 及 bin 文件。

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。如果有错误，根据提示进行相应的修改

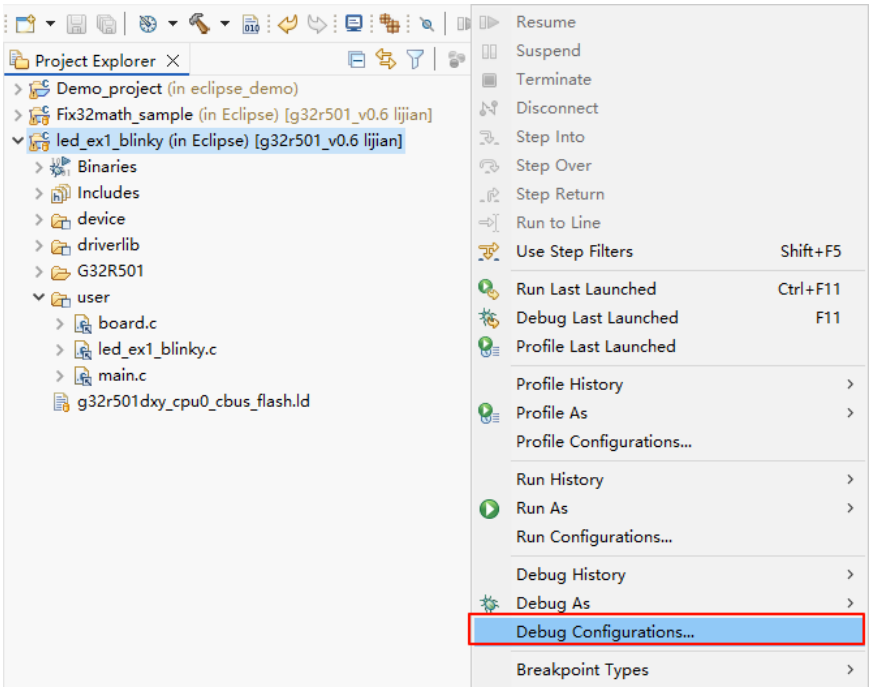
4.5.7. 程序仿真与下载

4.5.7.1. J-Link 仿真

1. 打开 Debug 配置界面

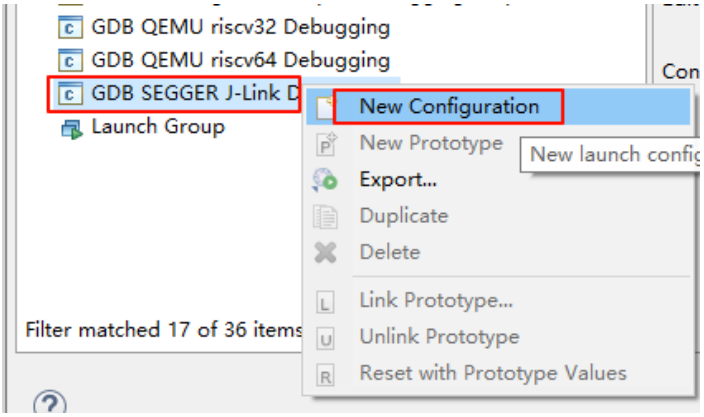
   - 在工程中，点击菜单栏中的 Run，在弹出的选项框中点击 Debug Configurations，进入 Debug 配置界面。

图 52 Debug 配置界面



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

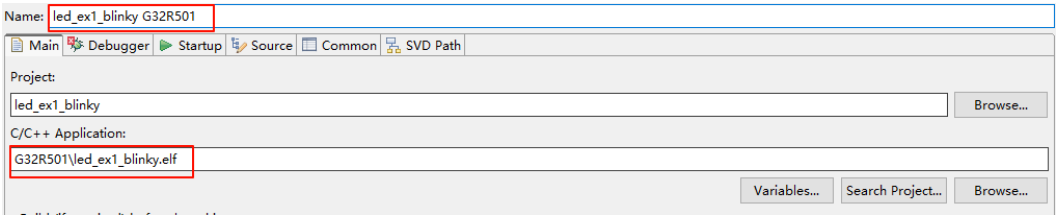
图 53 新建仿真 J-Link 配置



2. 配置 Main 选项卡

- 在最上端命名当前仿真配置名称。
- 选择“Browse...”，选择当前仿真配置对应的工程。
- 选择对应的仿真 elf 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。

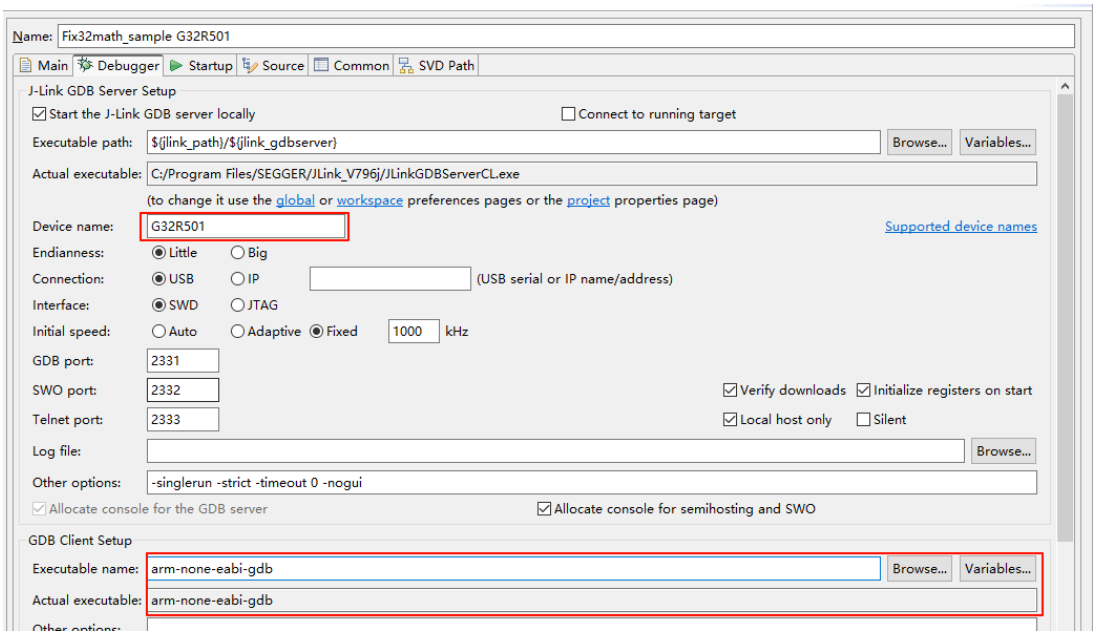
图 54 配置 Main



3. 配置 Debugger 选项卡

- 选择对应的仿真芯片，例如：G32R501
- 选择 GDB 服务，这里建议使用 Arm 提供的 arm-none-eabi-gdb.exe。

图 55 Debugger 选项卡



4. 配置 Startup 选项卡

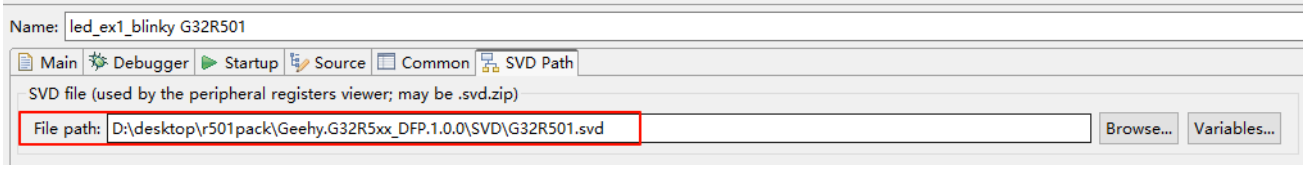
- 在 Initialization Commands、Run/Restart Commands 处添加 DCS KEY（这里需要与芯片端对应）
- 两处添加的内容一致，可参考

```
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 $t0 = *(unsigned int *)0x08000000

set $sp=$t0

set $t1 = *(unsigned int *)0x08000004

set $pc=$t1

set $xpsr=$xpsr|(1<<24)
```

5. 配置 SVD 选项卡

图 56 配置 SVD 所在路径目录



6. 最后点击选项卡的右下角的“Apply”按钮，应用所有的配置项。

4.5.7.2. GEEHY LINK 仿真

使用 Geehy-Link（WinUSB）下载及调试工程，使用 pyocd 对工程进行下载和仿真。具体请参考 pyocd 适配 G32R501

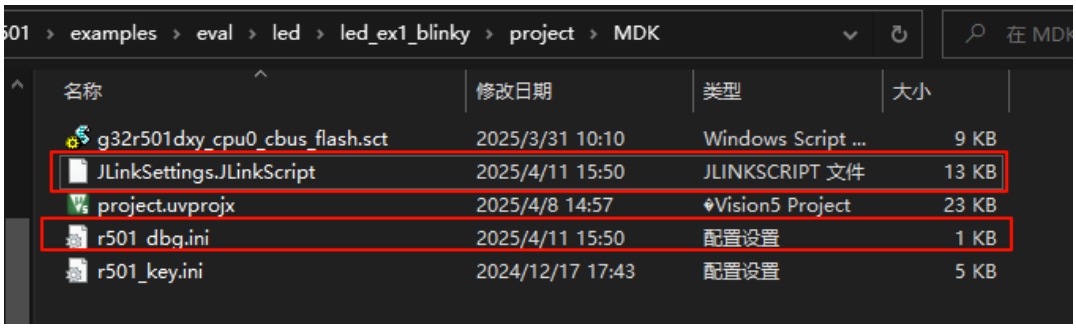
4.6. C 语言兼容性

4.6.1.1. 请参考章节 2“J-Link 仿真

13. 打开项目选项：选择“Options for Target”。
14. 选择调试器：在“Debug”选项卡中，选择 J-Link 调试器。
15. 在目标工程文件夹下添加 r501_dbg.ini 与 JLinkSettings.JLinkScript 文件。

- JLinkSettings.JLinkScript 文件是一种类 C 的脚本语言，用于定制 J-Link 调试器的操作，JLinkScript 文件包括基本语法、自定义操作、API 函数、DLL 全局常量（变量），其语法与 C 语言类似。
- r501_dbg.ini 与 JLinkSettings.JLinkScript 文件位于 SDK/device_support/g32r501/common/Jlink/。

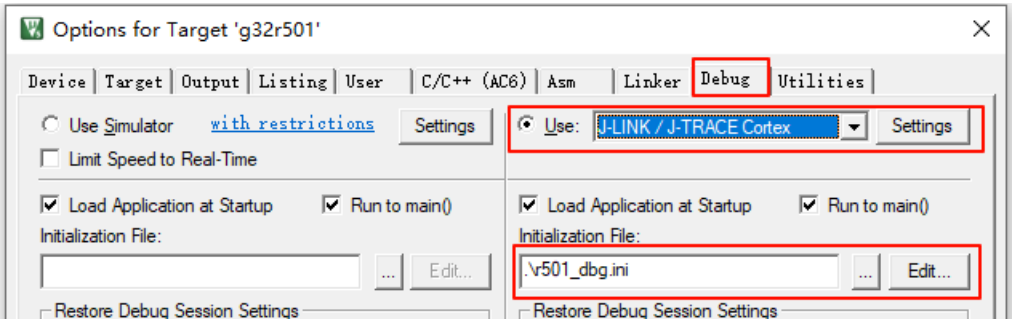
图 15 添加 J-Link 仿真文件



注意：r501_dbg.ini 文件与使用 GEEHY LINK 仿真时使用的 r501_dbg.ini 文件不同，使用时请注意区分。

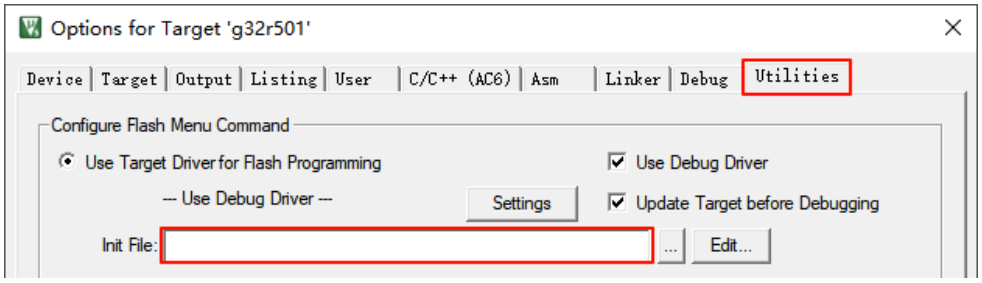
16. 加载仿真脚本：在“Settings”菜单中，点击“Load”按钮，加载特定的仿真脚本文件。

图 16 J-Link 仿真器及仿真脚本



17. 去除下载脚本：使用 J-Link 仿真时无需额外的下载脚本，请删除 Utilities 下载脚本“Init File”的设置。

图 17 去除 Utilities 下载脚本



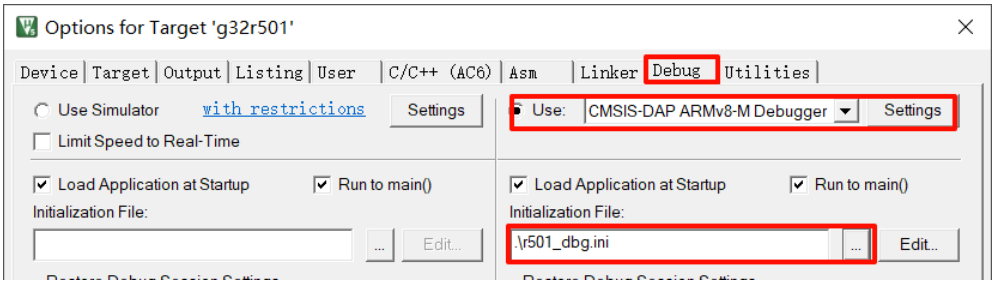
18. 启动调试：

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

4.6.1.2. GEEHY LINK 仿真

9. 打开项目选项：选择“Options for Target”。
10. 选择调试器：在“Debug”选项卡中，选择合适的调试器。
11. 加载仿真脚本：在“Settings”菜单中，点击“Load”按钮，加载特定的仿真脚本文件。

图 18 选择 DAP LINK 仿真器及仿真脚本



12. 启动调试：

- 下载完成后，可以点击“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。

C 语言兼容性”。

4.7. 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

4.7.1. 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 MDK（Keil 的 ARM 汇编器，即 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。

4.7.2. 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;
}
```

4.8. 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。G32R501 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：
  - G32R501 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：
  - G32R501 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。

4.9. RAM 运行

请参考章节 2.7。

5. pyocd 适配 G32R501

5.1. 背景

为便于客户能够在开源环境下对 G32R501 进行程序下载仿真等操作，G32R501 系列 MCU 需要对 pyocd 进行支持。

5.2. pyocd 适配修改内容

目前的 pyocd（https://github.com/pyocd/pyOCD/releases/tag/v0.36.0）发布版本至 0.36，该版本不支持 M52 内核芯片以及 G32R501 芯片，需要修改其源码以完成支持。

5.2.1. 添加 M52 内核支持

1. 在 pyocd 中添加 M52 内核支持，主要修改的文件是：

   - 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 57 修改后 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       ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x1a03) : CmpInfo('FPBv2',          None,       FPB.factory         ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x2a04) : CmpInfo('v8-M Debug',     None,       CortexM_v8M.factory ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x6a05) : CmpInfo('v8-R Debug',     None,       None                ),
```

- 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 58 修改后 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",
    }
```

5.2.2. 添加 G32R501 芯片支持

1. 在 pyocd\target\builtin 中添加 g32r501 下载算法支持文件：target_G32R501xx.py。此文件已添加在 SDK/device_support/g32r501/common/pyOCD/target_G32R501xx.py。
2. 添加 g32r501 支持，在 pyocd\target\builtin__init__.py 中加入：

```
from . import target_G32R501xx



'g32r501dxx': target_G32R501xx.G32R501Dxx,
'g32r501xx': target_G32R501xx.G32R501xx,
```

图 59 修改后__init__.py

```
138  from . import target_Air32F103xx                32B1ME0,
139  from . import target_G32R501xx                  32B1MD1,
140
315          'air32f103xb': target_Air32F103xx.Air32F103xB,
316          'air32f103xc': target_Air32F103xx.Air32F103xC,
317          'air32f103xp': target_Air32F103xx.Air32F103xP,
318          'air32f103xe': target_Air32F103xx.Air32F103xE,
319          'air32f103xg': target_Air32F103xx.Air32F103xG,
320          'g32r501xx': target_G32R501xx.G32R501xx,
321          'g32r501dxx': target_G32R501xx.G32R501Dxx,
322      }
323
```

5.2.3. pyocd_user.py

pyocd 支持使用“--script=<path>”解析相应的脚本以添加自定义命令和一些连接前后的额外操作。

pyocd_user.py 文件已添加在 SDK/device_support/g32r501/common/pyOCD/target_G32R501xx.py。

5.2.3.1. DCS 秘钥

由于 G32R501 特性，芯片带 DCS 功能，需要对芯片在连接过程中传输秘钥至前文提及到的：target_G32R501xx.py 中，以便在下载过程中使用。

如用户需要修改 DCS 秘钥可直接修改 NEW_DECRYPT_KEYS：

```
NEW_DECRYPT_KEYS = [
    (0x50024020, 0xFFFFFFFF),
    (0x50024024, 0xFFFFFFDC),
    (0x50024028, 0xFFFFFFFF),
    (0x5002402C, 0xFFFFFFFF),
    (0x500240A0, 0xFFFFFFFF),
    (0x500240A4, 0xFFFEDFFF),
    (0x500240A8, 0xFFFFFFFF),
    (0x500240AC, 0xFFFFFFFF),
]
```

注意：前面是写入的地址，后面是写入的 DCS KEY 内容。

5.2.3.2. 连接阶段

连接阶段需进行如下步骤：

1. 执行 DCS 解密
2. 初始化 CPU

以上操作实现在'def did_connect(board) -> None:'中。

5.3. pyocd 安装

5.3.1. Windows

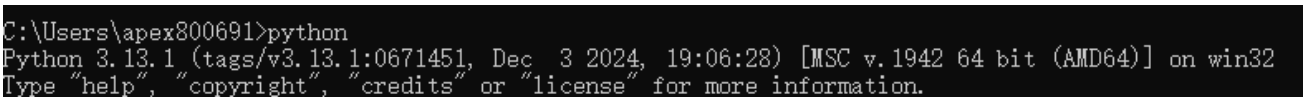
5.3.1.1. 安装 python

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 60 python 安装验证



5.3.1.2. 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：

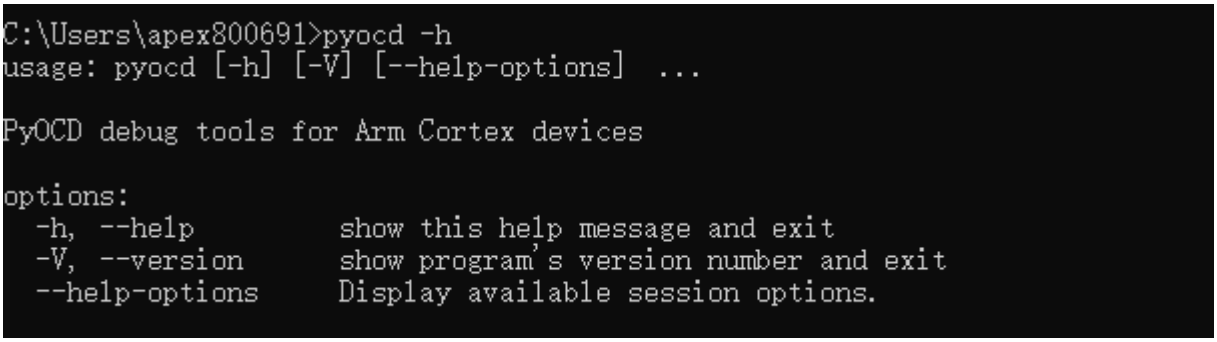
1. 使用 pip 命令安装：

图 61 安装 pyocd

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 62 pyocd 安装验证



5.3.2. Ubuntu

5.3.2.1. 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
```

5.3.2.2. 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
```

5.3.2.3. 安装 pyocd

1. 在激活的虚拟环境中，使用 pip 安装 pyOCD：

```
pip install pyocd
```

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
```

5.3.2.4. 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 63 连接至 Ubuntu 的仿真器序列号

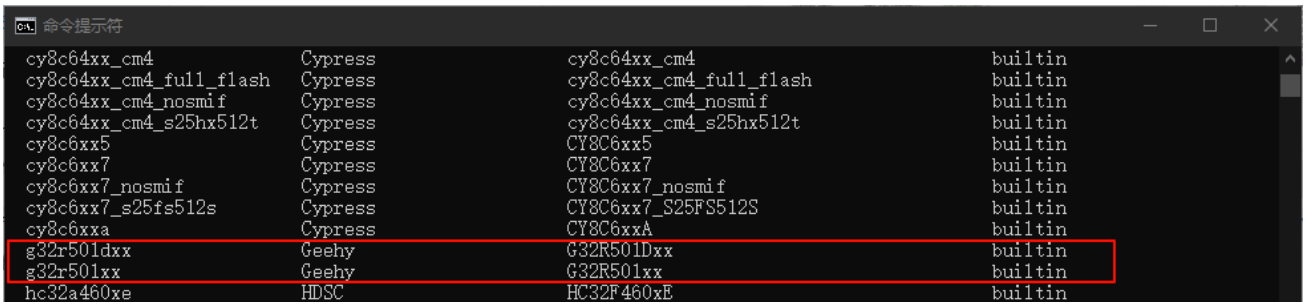
```
(venv) kai@Ubuntu:~$ pyocd list
  #   Probe/Board             Unique ID                        Target
------------------------------------------------------------------------
  0   Geehy CMSIS-DAP WinUSB  00350043500000144e544859000258   n/a
```

5.3.3. 替换修改项内容

为支持 G32R501，请参考 5.2 内容修改已下载完毕的 pyocd 内容。

验证 G32R501 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"g32r501xx"。

图 64 支持芯片列表



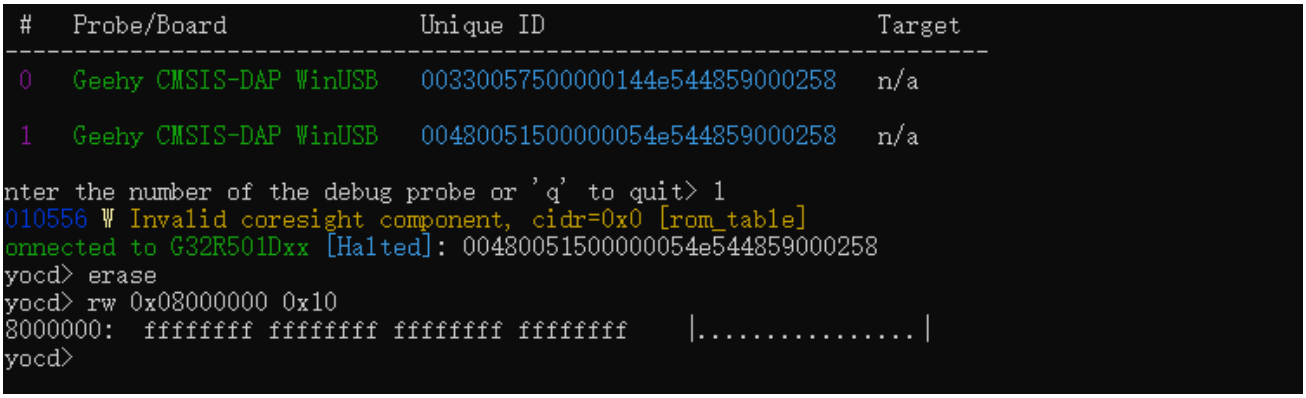
5.4. 命令行使用

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，添加 pyocd.yaml 管理文件，这个文件将指示默认连接的芯片、使用的连接方式等内容。用户可参考官网的说明：https://pyocd.io/docs/configuration.html

   g32r501 的参考 pyocd.yaml 管理文件添加在 SDK/device_support/g32r501/common/pyOCD/target_G32R501xx.py。
3. 在工作目录下添加 pyocd_user.py 文件，文件内容请参考 2.3 小结。
4. 在工作目录下启动 cmd，并输入：pyocd commander，随后可在命令行窗口输入相关指令进行对应的操作。

图 65 pyocd commander



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 66 Global pyOCD Path 步骤 1-2

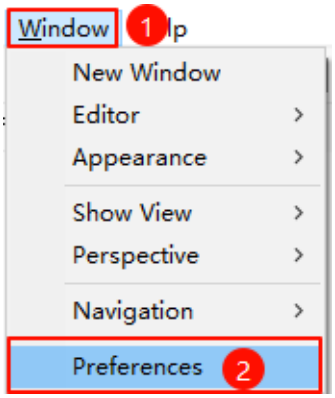
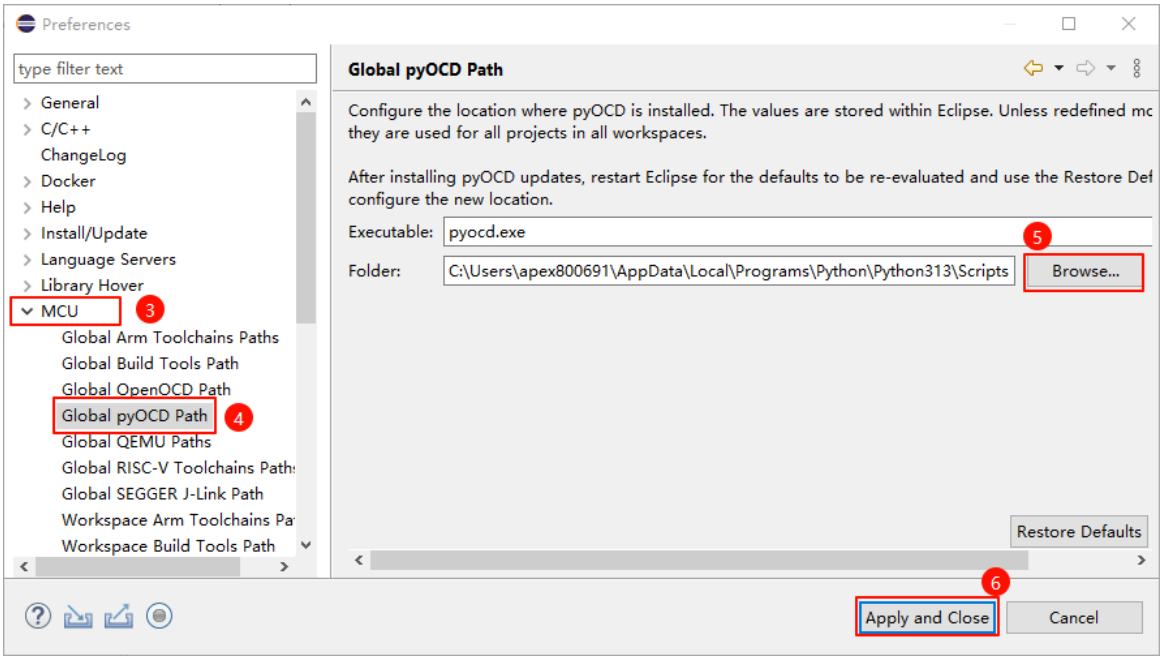


图 67 Global pyOCD Path 步骤 3-6



5.5.1. 单核仿真配置

Eclipse 导入工程，确保编译无误后。请按照如下步骤配置仿真选项卡。

1. 新建仿真配置

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 68 New Configuration 步骤 1-2

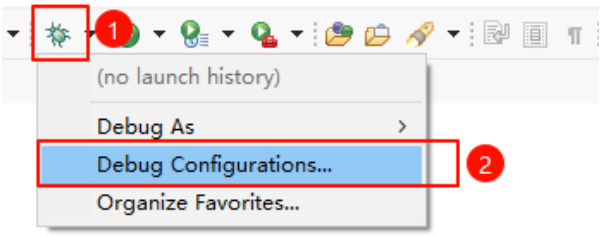
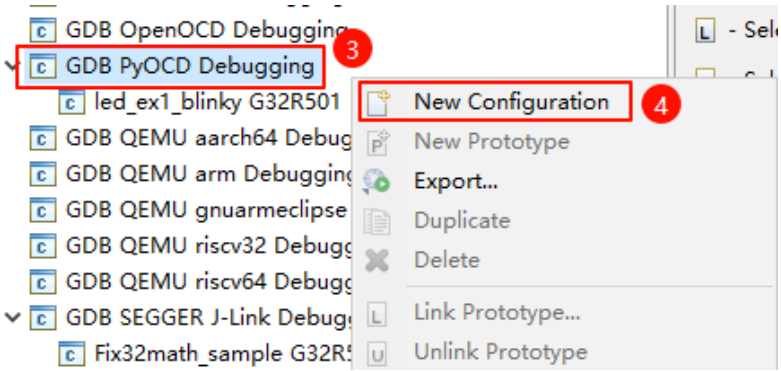


图 69 New Configuration 步骤 3-4



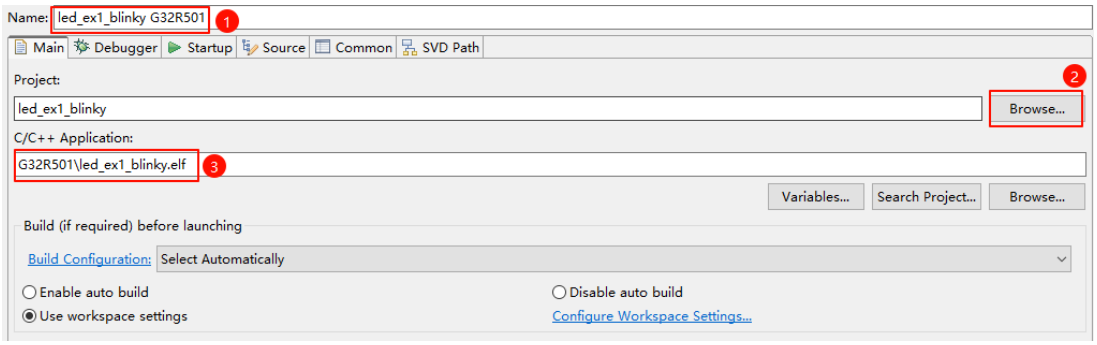
2. 配置 Main 选项卡

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：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 70 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

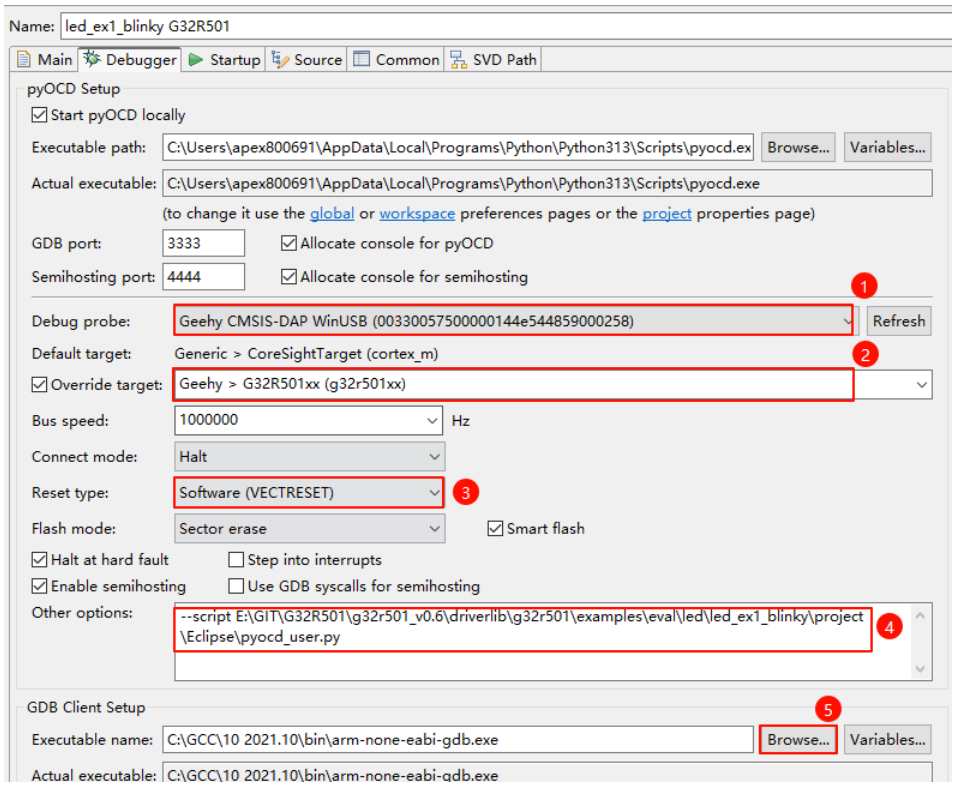
   2) 选择对应的仿真芯片，例如：Geehy >G32R501xx(g32r501xx)

   3) 复位模式选择“Software (SYSRESETREQ)”

   4) 配置文件选择前文写好的"pyocd_user.py"，这里建议使用绝对路径，且路径不要有中文或空格。

   5) 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 71 配置 Debugger



4. 配置 Startup 选项卡

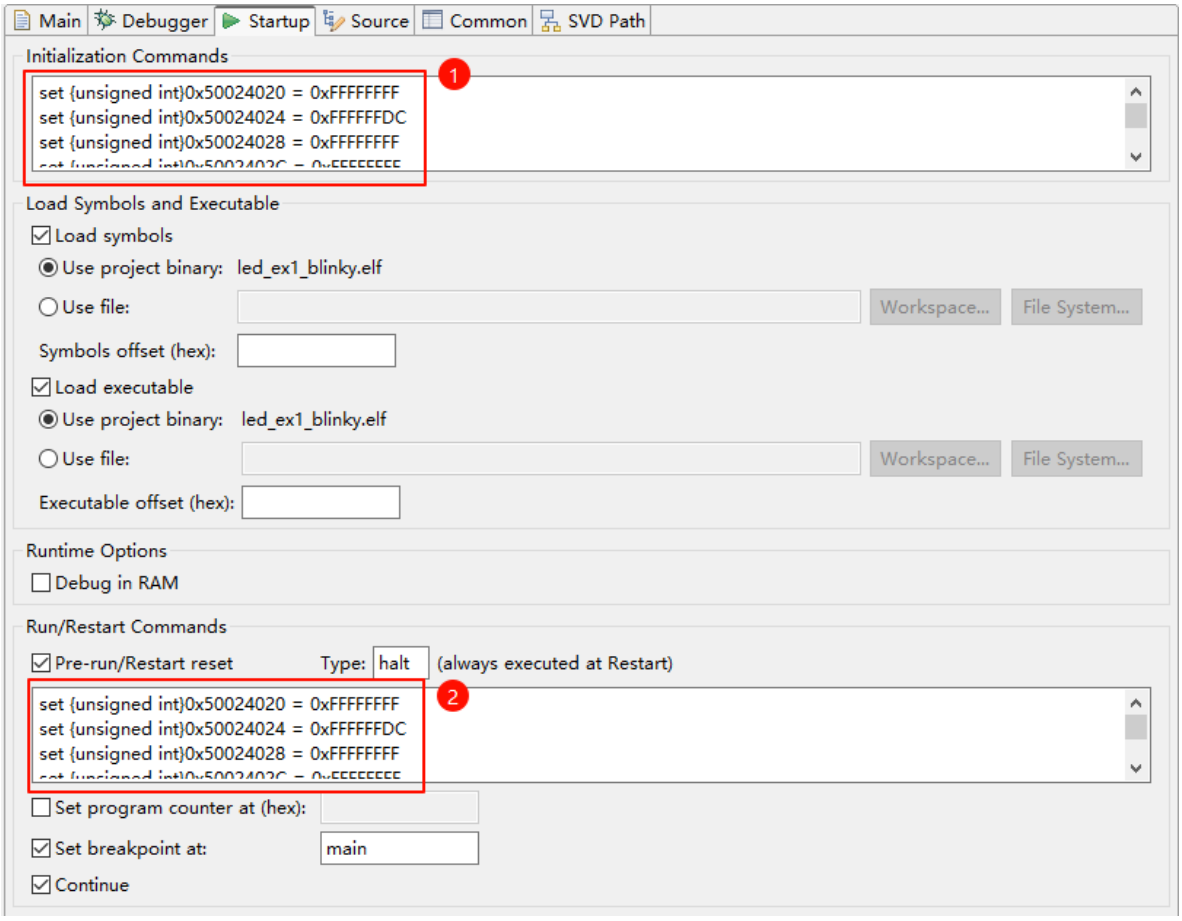
   1) 在 Initialization Commands、Run/Restart Commands 处添加 DCS KEY（这里需要与 pyocd_user.py、芯片端对应）

   2) 两处添加的内容一致，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 $t0 = *(unsigned int *)0x08000000
set $sp=$t0
set $t1 = *(unsigned int *)0x08000004
set $pc=$t1
set $xpsr=$xpsr|(1<<24)
```

图 72 配置 Startup



5. 最后点击选项卡的右下角的“Apply”按钮，应用所有的配置项。

5.5.2. 双核仿真配置

双核仿真配置请查看《AN1128_G32R501_双核仿真指导手册》

5.5.3. Ubuntu 下退出仿真后程序无法运行

5.5.3.1. 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。

5.5.3.2. 解决方案

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

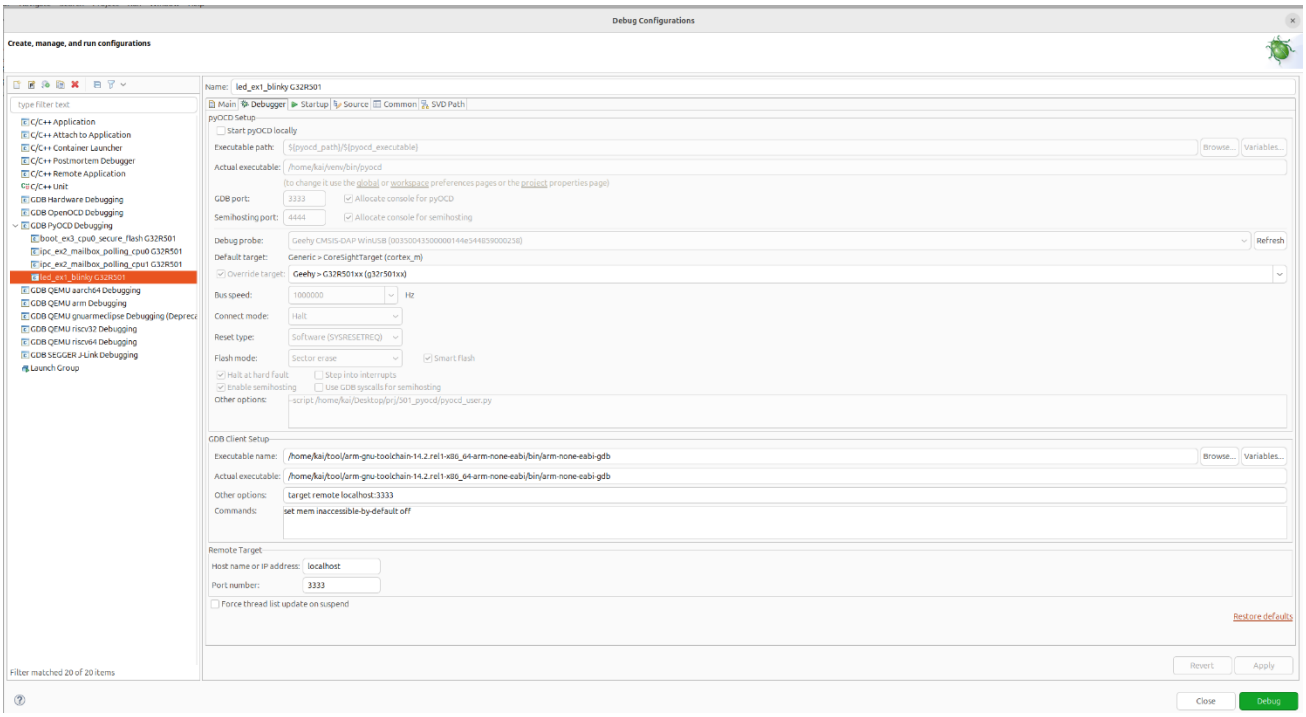
在终端启动 pyOCD gdbserver：

图 73 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 74 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。

6. 版本历史

表格 4 文件版本历史

日期	版本	变更历史
2025.01	1.0	新建
2025.04	1.1	新增章节 2.3.10.1、4、5

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。