Nuclei Development Tool Guide

Release 2024.06

Nuclei

Nov 04, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION

This user guide mainly talked about how to use Nuclei Development Tools, including Nuclei Studio IDE, Nuclei RISC-V Toolchain, Nuclei OpenOCD, Nuclei QEMU and Nuclei Model.

Nuclei Studio IDE is built on Eclipse Embedded CDT plugins, mainly optimized for Nuclei RISC-V Processor to improve user experience in IDE.

Nuclei RISC-V Toolchain is built on RISC-V GNU and LLVM toolchain(gcc/llvm/binutils/gdb/newlib) and also include Nuclei C Runtime Library, it provide good support for Nuclei RISC-V Processor.

Nuclei OpenOCD is built on RISC-V OpenOCD, adding nuspi flash support, cjtag support, customized csr support, nuclei openocd flashloader support.

Nuclei QEMU is built on QEMU project, adding Nuclei N/NX/UX RISC-V processor support, which works with Nuclei SDK and Nuclei Linux SDK.

Nuclei Model uses spike as the RISC-V ISA simulator and adds support for Nuclei ' s N/NX/UX RISC-V processors, it supports near cycle-level simulation and SystemC TLM 2.0 Nuclei EvalSoC modeling.

Note: To get a pdf version of this documentation, please click Nuclei Development Tool User Guide

If you have issues in this user guide, please send us an pull request in https://github.com/Nuclei-Software/nuclei-tool-guide repo to help us improve it.

If you have issues in our Nuclei Tools, please send us an issue in this repo or related tool repo to help us improve it.

- Nuclei Studio: https://github.com/Nuclei-Software/nuclei-studio
- Nuclei RISC-V Toolchain: https://github.com/riscv-mcu/riscv-gnu-toolchain
- Nuclei OpenOCD: https://github.com/riscv-mcu/riscv-openocd
- Nuclei Qemu: https://github.com/riscv-mcu/qemu
- Nuclei DLink: https://github.com/nuclei-Software/nuclei-dlink

CHAPTER

TWO

NUCLEI STUDIO IDE

2.1 Nuclei Studio IDE 简介

Note:

- Nuclei Studio 出视频教程啦，相关内容在 芯来科技视频号中持续更新中，您可以在微信中搜索 芯来科技视频号并关注，以便获取到我们最新的更新内容。
- 在使用 Nuclei Studio, Nuclei Tools 过程中，如查有问题，可以查阅 https://github.com/Nuclei-Software/nuclei-studio内容，也可以向我们提交相关 Issue。

一款高效易用的集成开发环境（Integrated Development Environment，IDE）对于任何 MCU 都显得非常重要，软件开发人员需要借助 IDE 进行实际的项目开发与调试。ARM 的商业 IDE 软件 Keil，在中国大陆很多嵌入式软件工程师均对其非常熟悉。但是商业 IDE 软件（譬如 Keil）存在着授权以及收费的问题，各大 MCU 厂商也会推出自己的免费 IDE 供用户使用，譬如瑞萨的 e2studio 和 NXP 的 LPCXpresso 等，这些 IDE 均是基于开源的 Eclipse 框架，Eclipse 几乎成了开源免费 MCU IDE 的主流选择。

Nuclei Studio IDE 正是芯来公司，基于 Eclipse IDE 开发的一款针对芯来 RISC-V 处理器 IP 核产品的集成开发环境工具。

Eclipse 平台采用开放式源代码模式运作，并提供公共许可证（提供免费源代码）以及全球发布权利。Eclipse 本身只是一个框架平台，除了 Eclipse 平台的运行时内核之外，其所有功能均位于不同的插件中。开发人员既可通过 Eclipse 项目的不同插件来扩展平台功能，也可利用其他开发人员提供的插件。一个插件可以插入另一个插件，从而实现最大程度的集成。

由于 Eclipse IDE 已经在社区被大量使用，一些常见的使用方法在 Eclipse IDE 里面，如果没有和硬件或者 CPU 绑定，一般情况下是可以借鉴其他人写的关于 Eclipse IDE 的使用教程，这里推荐几个常用的教程网站：

- https://help.eclipse.org/latest/index.jsp
- https://eclipse-embed-cdt.github.io/
- https://mcuoneclipse.com/

Eclipse IDE 平台具备以下几方面的优势。

- 社区规模大

  Eclipse 自 2001 年推出以来，已形成大规模社区，这为设计人员提供了许多资源，包括图书、教程和网站等，以帮助他们利用 Eclipse 平台与工具提高工作效率。Eclipse 平台和相关项目、插件等都能直接从 eclipse.org 网站下载获得。

- 持续改进

  Eclipse 的开放式源代码平台帮助开发人员持续充分发挥大规模资源的优势。Eclipse 在以下多个项目上不断改进。

  - 平台项目——侧重于 Eclipse 本身。
  - CDT 项目——侧重于 C/C++ 开发工具。

  - PDE 项目——侧重于插件开发环境。

- 源码开源

  设计人员始终能获得源代码，总能修正工具的错误，它能帮助设计人员节省时间，自主控制开发工作。

- 兼容性

  Eclipse 平台采用 Java 语言编写，可在 Windows 与 Linux 等多种开发工作站上使用。开放式源代码工具支持多种语言、多种平台以及多种厂商环境。

- 可扩展性

  Eclipse 采用开放式、可扩展架构，它能够与 ClearCase、SlickEdit、Rational Rose 以及其他统一建模语言（UML）套件等第三方扩展协同工作。此外，它还能与各种图形用户接口（GUI）编辑器协同工作，并支持各种插件。

Nuclei Studio IDE 充分利用上述 Eclipse IDE 优势，结合社区成熟的 Eclipse embedded CDT, Linux Tools 等插件，并研发自有插件满足 RISC-V 嵌入式开发的日常需求:

- 工程创建，管理，编译和调试
- 支持多种调试方案，例如 OpenOCD，JLink，Nuclei DLink，Nuclei Qemu 等
- 支持扩展调试方案，方便扩展支持更多调试器
- 支持多种编译器，包括 gcc, clang, zcc
- 提供快捷的工程常用设置工具 **Nuclei Settings**
- 支持基于 Nuclei ETrace 软硬件方案的 Onchip Trace
- 支持基于 gprof、gcov 的大幅增强 profiling 和 code coverage 方案
- 支持 Nuclei PacKage(NPK) 软件包方案，可以便捷的扩展支持软件开发包和 Nuclei Studio 解耦，实现 软件包导入 -> **Project Wizard** 的便捷方案, 这种方案已经得到广泛的应用，例如芯来科技自有的 **Nuclei SDK**.
- 更快捷的工程和工作空间的打开方式
- 更好用的 Launchbar 功能

2.2 Nuclei Studio 更新说明

2.2.1 2024.06 版更新说明

本版本是一次比较重大的版本升级，2024.06 版本升级了 CDT 版本到 Eclipse CDT 2024-06，升级了芯来科技的工具版本至 2024.06，优化了部分原有功能，新增了调试及代码性能分析等功能，以及解决了 2024.02 版中存在的缺陷。

升级 Eclipse CDT 版本

在 Nuclei Studio 2024.06 版本中基础的 CDT 版本，升级到了 11.6.0，并基于 Eclipse CDT 2024-06 版本开发此版本。

升级 RISC-V Toolchain、OpenOCD、QEMU 版本

在 Nuclei Studio 2024.06 版本中集成了 Nuclei RISC-V Toolchain 2024.06 版，具体信息可以查看：https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2024.06 。

在 Nuclei Studio 2024.06 版本中集成了 OpenOCD 2024.06 版，具体信息可以查看：https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2024.06 。

在 Nuclei Studio 2024.06 版本中集成了 Nuclei Qemu 2024.06 版，具体信息可以查看：https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2024.06 。

新增对 U600 和 UX1000 的支持

配合 U600 和 UX1000 核的发布，同步增加了对 U600 和 UX1000 核配套支持。

优化 NPK 软件包管理

优化 Nuclei Package Management 中对 NPK 包依赖的管理，使其更易使用；优化了部分 NPK 包安装的提示信息及日志，提高 NPK 包管理的使用体验。具体参见*NPK 软件包管理* (page 73) 。

Note: 注意：本次版本升级，变更了 NPK 包管理的配置，在 2024.02 版及之前版本中安装的 NPK 包在 2024.06 版 NucleiStudio 无法识别，用户需重新下载安装 NPK 包。

增加和优化部分编译选项

在 Properties 和 Nuclei Settings 页面内，在 Optimization Level 中新增 -Oz 选项；在 GNU RISC-V Cross C++ Linker 的 Libraries 页新增对 group libraries 的支持, 参见工程编译链接 *C* 库找不到符号报错 (page 250) 。

优化调式模式切换

NucleiStudio 支持多种调试模式，如 OpenOCD、Jlink、Dlink、Custom 等，同时还有 Qemu 等仿真器，为了方便用户在多工程多种模式之间切换，优化了调式模式的切换，具体内容参见调试模式管理 (page 134) 。

优化和完善 DLink Debug 调试

Nuclei DLink 是芯来科技基于 RV Link，并在 RV Link 的基础上做了许多功能增加后，所研发的 RISC-V 调试器，使之更适应于 Nuclei Studio 的应用场景。具体内容可以查看使用 *DLink* 调试运行项目 (page 170) 。

集成 Terapines ZCC Lite 编译器

Terapines ZCC 是兆松科技研发的高性能 RISC-V 编译器。Nuclei Studio 2024.06 版中对 Terapines ZCC 进行支持，用户可以在 Nuclei Studio 中直接使用。具体参见*Nuclei Studio 中编译 Hello World 项目* (page 133) 。

新增 LST View 工具

LST View 是一个 lst 文件查看器，可以方便用户查看 lst 格式的文件，并实现 *.lst 文件与源代码的联动，具体请参见*LST View* (page 184) 。

优化和完善 Gprof 功能

Gprof 是一个强大的性能分析工具，可以帮助开发者理解 C/C++ 程序的运行情况，通过 Gprof 可以获取到程序中各个函数的调用信息、调用次数、执行时间等，对优化程序、提升程序运行效率具有重要的意义。具体请参见*Code Coverage 和 Profiling 功能* (page 187) 。

优化和完善 Gcov 功能

Gcov 是一个测试 C/C++ 代码覆盖率的工具，伴随 GCC 发布，配合 GCC 共同实现对 C/C++ 文件的语句覆盖、功能函数覆盖和分支覆盖测试。具体请参见*Code Coverage 和 Profiling 功能* (page 187) 。

新增 Call Graph 功能

Call Graph 是分析函数调用关系图的工具，结合 Gprof 使用，便于开发者快速了解程序执行的过程及调用关系。具体请参见*Code Coverage 和 Profiling 功能* (page 187) 。

新增 Nuclei Near Cycle Model 支持

Nuclei Near Cycle Model，它是由芯来科技自主研发的仿真测试和性能分析工具，可以帮助研发人员在项目初期进行一些必要的仿真测试和程序性能分析，具体请参见使用 *Nuclei Near Cycle Model* 仿真性能分析 (page 229) 。

2.2.2 2024.02.dev 版更新说明

本版本是开发版本（您下载到的链接内容随时可能会变更），本版本解决了 Nuclei Studio 2023.10 版中存在的缺陷，并优化了部分原有功能如 ETrace 特性，新增了一些功能如对 N100 的支持、Dlink 的支持等，更好为满足客户评估和更新使用。

升级 Eclipse CDT 版本

在 Nuclei Studio 2024.02.dev 版本中基础的 Eclipse CDT 版本，升级到了 Eclipse CDT 2023.12 版。

新增对 N100 的支持

配合 N100 核的发布，同步增加了对 N100 的配套支持。

新增批量转换 Gcc13 工程工具

在 2023.10 版 Nuclei Studio 中，升级 GCC 13 后，当有大量工程需要转换时，单个转换效率低，为方便开发者，提供了一个批量转换 GCC 13 工具。具体内容参见批量将工程转换成支持 *gcc 13* 的工程 (page 178)。

优化和完善 Trace 功能

Nuclei Studio 中 Trace 功能升级，实现了在 OpenOCD 模式下对单核应用、SMP 多核应用、AMP 多核应用的支持，具体内容参见*Trace* 功能的使用 (page 208)；在 Dlink 模式下，仅对单核应用支持。Trace 功能需要有对应 CPU IP 的支持，如需体验此功能，请与我们联系。

优化和完善 RVProf 功能

RVProf 是芯来科技基于 CPU cycle model 开发的性能分析工具，具体内容参见第*RVProf* 功能的使用 (page 221)。此功能需要有相应的 NPK 软件包支持，如需体验此功能，请与我们联系。

新增对 DLlink Debug 的支持

Dlink 是芯来自主研发的调试解决方案，在本次版本中得到支持。此功能需要有相应的 Dlink 调试器的支持，如需体验此功能，请与我们联系。

2.2.3 2023.10 版更新说明

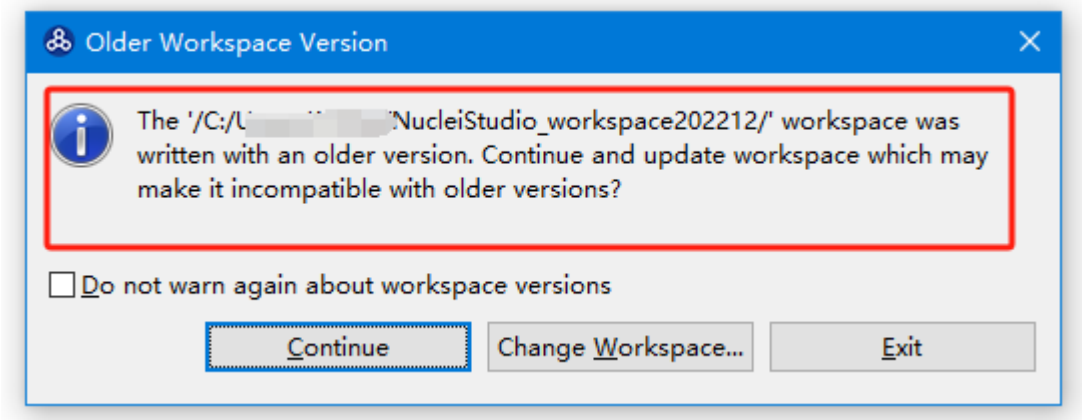
本版本是一次比较重大的版本升级，集成了 Nuclei 2023.10 版本的 Toolchain, QEMU, OpenOCD, 且 Eclipse CDT 版本进行了升级，GCC 版本也做了重大迭代，升级到了 GCC 13, NPK 部分也做了大量的新功能的增加以支持 GCC 或者 CLANG 的工程创建，并且增加很多新的 Configuration 字段类型，方便在 Project Wizard 中更灵活的进行工程配置。

升级 Eclipse CDT 版本

在 Nuclei Studio 2023.10 版本中基础的 Eclipse CDT 版本，升级到了 Eclipse CDT 2023.06 版; Eclipse CDT 2023-06 版本是 Eclipse 基金会 2023 年第二个季度同步版本，有 64 个参与项目，于 2023 年 6 月 14 日发布。

参考地址：Eclipse IDE for C/C++ Developers[1]

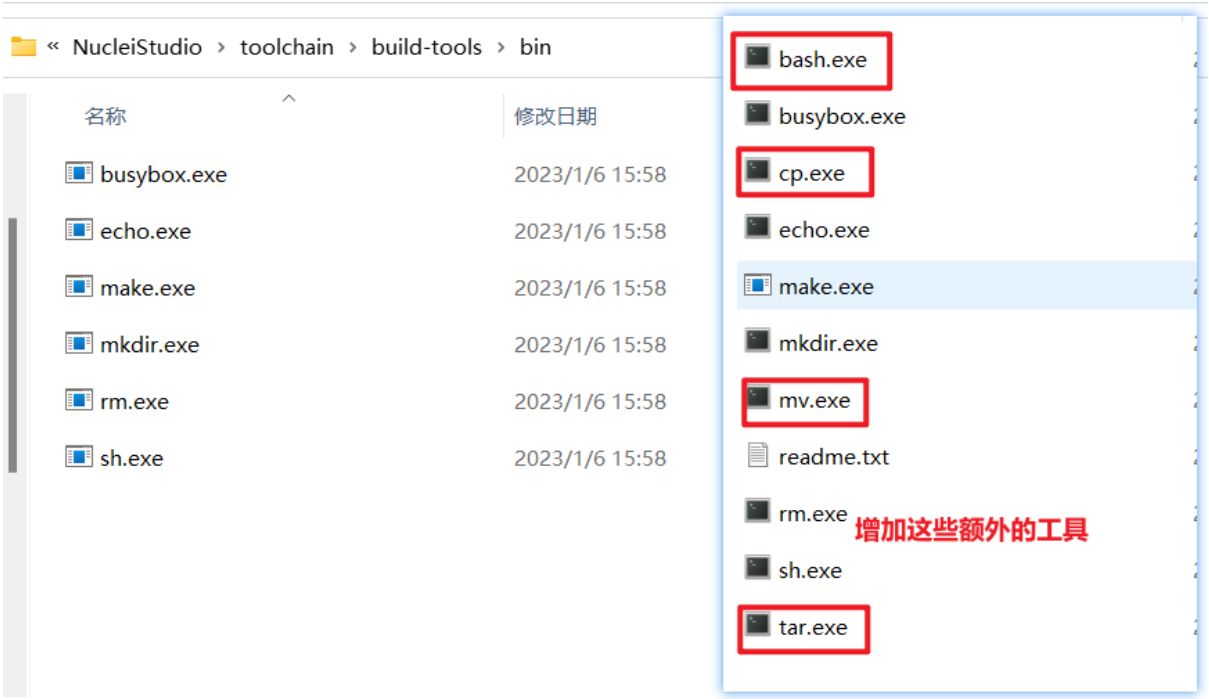
升级后打开之前版本创建的 workspace, 会弹出不兼容的警告，使用时可能会有异常，建议更换新的 workspace 目录。



[1] https://www.eclipse.org/downloads/packages/release/2023-06/r/eclipse-ide-cc-developers

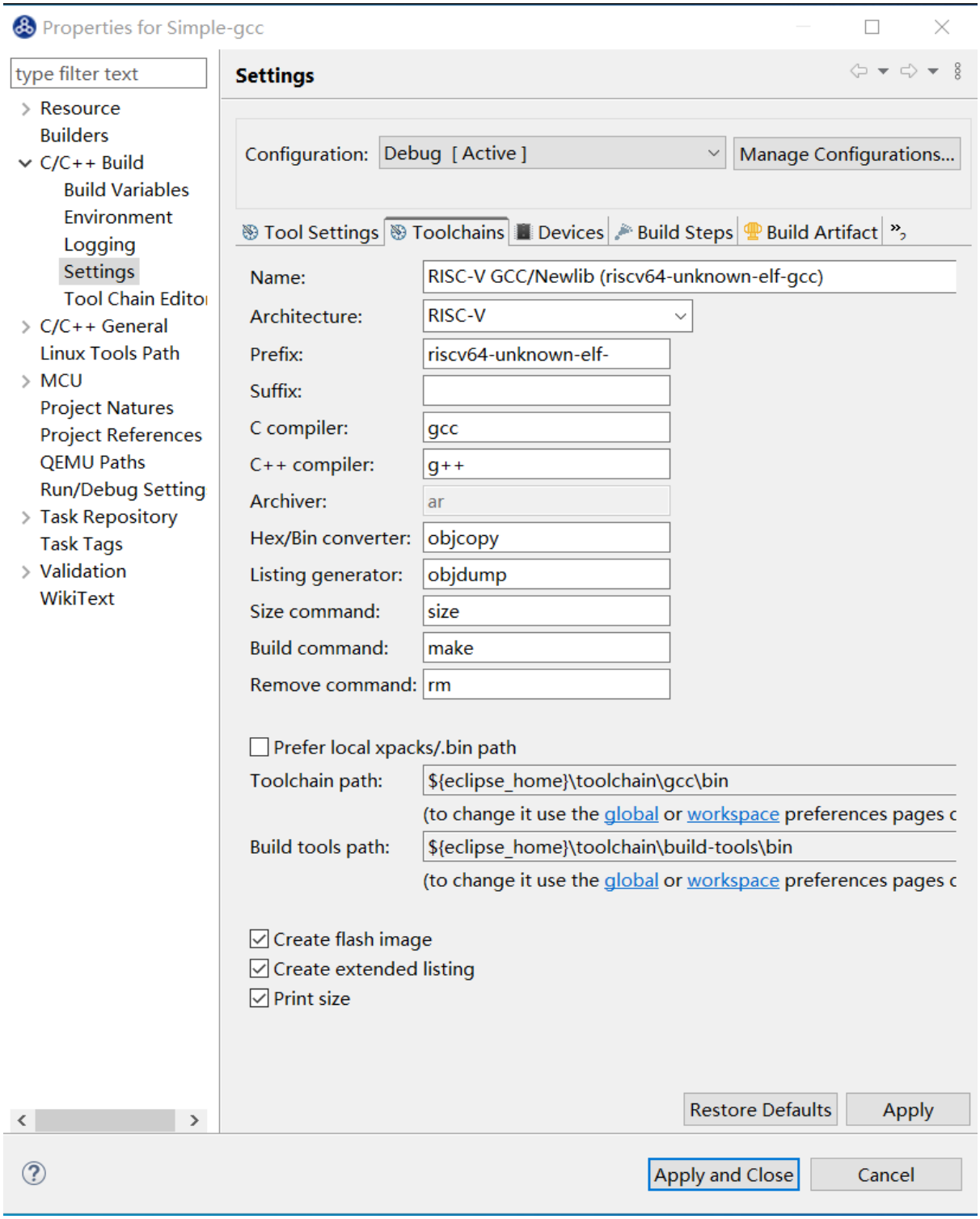
升级 build-tools 版本

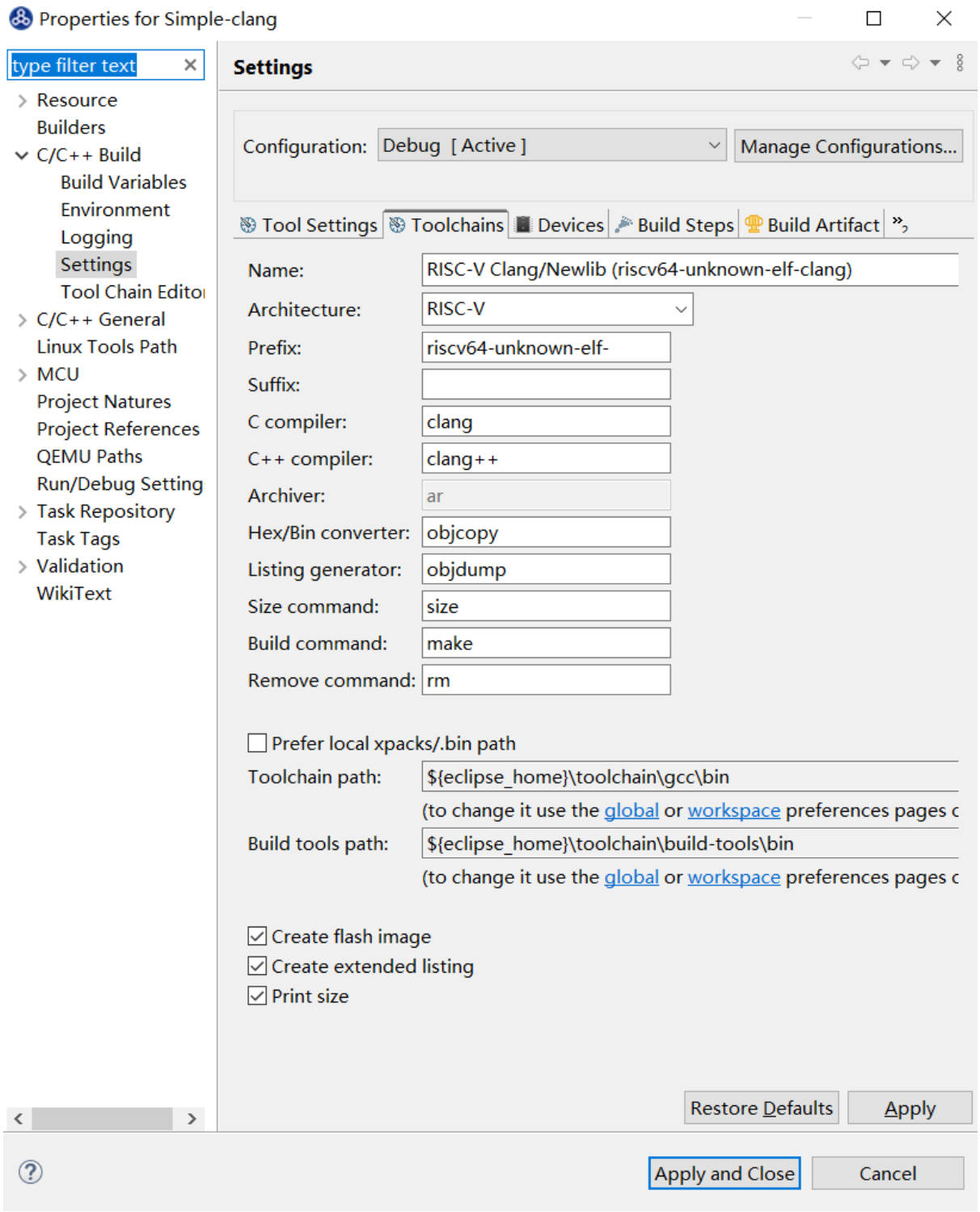
在 Nuclei Studio 2023.10 版将 toolchain 中的 build-tools 更新到 4.4 版本，并额外增加了 bash.exe、cp.exe、mv.exe、tar.exe 工具。



支持 GCC 13 和 Clang 17

在 Nuclei Studio 2023.10 版本中实现了对 GCC 13 的支持，相对于之前的 gcc10 版本 GCC 13 在对 RISC-V 指令扩展的支持更加完备，且在我们维护的版本中，支持完整的 RVV Intrinsic API v0.12 版本。同时 Nuclei Studio 2023.10 版本中也实现了对 Clang 17 的支持（参考地址：https://releases.llvm.org/17.0.1/docs/RISCVUsage.html）。当然，如果有用户依然想使用 GCC 10 时行项目开发，我们也保留了相关的配置，但是工具链并没有集成到 IDE 中，用户需要自行下载并放置在 gcc10 目录中，参见里面的 README.txt，并且我们也提供了老版本采用 gcc10 的 Nuclei Studio 创建的工程升级到 gcc13 工具链上，具体使用可以参考导入旧版本 *Nuclei Studio* 创建的工程 (page 176)。Nuclei RISC-V Toolchain 2023.10 更详细的说明，请参阅: https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2023.10


Properties for Simple-gcc
type filter text
Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
Tool Chain Editor
C/C++ General
Linux Tools Path
MCU
Project Natures
Project References
QEMU Paths
Run/Debug Setting
Task Repository
Task Tags
Validation
WikiText
Settings
Configuration: Debug [Active]
Manage Configurations...
Tool Settings
Toolchains
Devices
Build Steps
Build Artifact
Name: RISC-V GCC/Newlib (riscv64-unknown-elf-gcc)
Architecture: RISC-V
Prefix: riscv64-unknown-elf-
Suffix:
C compiler: gcc
C++ compiler: g++
Archiver: ar
Hex/Bin converter: objcopy
Listing generator: objdump
Size command: size
Build command: make
Remove command: rm
Prefer local xpacks/.bin path
Toolchain path: ${eclipse_home}\toolchain\gcc\bin
(to change it use the global or workspace preferences pages c
Build tools path: ${eclipse_home}\toolchain\build-tools\bin
(to change it use the global or workspace preferences pages c
Create flash image
Create extended listing
Print size
Restore Defaults
Apply
Apply and Close
Cancel

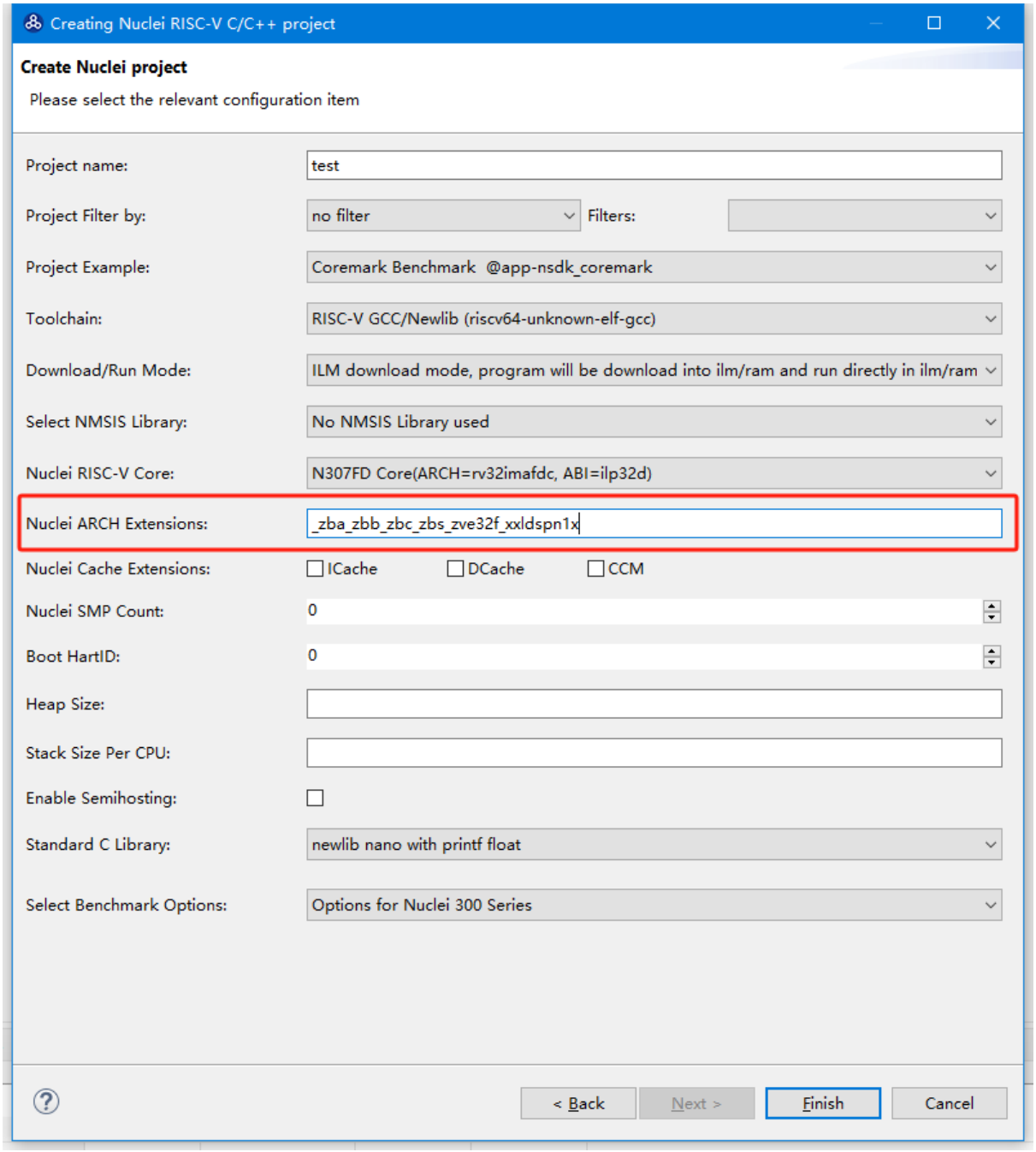

Properties for Simple-clang
type filter text
Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
Tool Chain Editor
C/C++ General
Linux Tools Path
MCU
Project Natures
Project References
QEMU Paths
Run/Debug Setting
Task Repository
Task Tags
Validation
WikiText
Settings
Configuration: Debug [Active]
Manage Configurations...
Tool Settings
Toolchains
Devices
Build Steps
Build Artifact
Name: RISC-V Clang/Newlib (riscv64-unknown-elf-clang)
Architecture: RISC-V
Prefix: riscv64-unknown-elf-
Suffix:
C compiler: clang
C++ compiler: clang++
Archiver: ar
Hex/Bin converter: objcopy
Listing generator: objdump
Size command: size
Build command: make
Remove command: rm
Prefer local xpacks/.bin path
Toolchain path: ${eclipse_home}\toolchain\gcc\bin
(to change it use the global or workspace preferences pages c
Build tools path: ${eclipse_home}\toolchain\build-tools\bin
(to change it use the global or workspace preferences pages c
Create flash image
Create extended listing
Print size
Restore Defaults
Apply
Apply and Close
Cancel

RISC-V 指令扩展使用变更

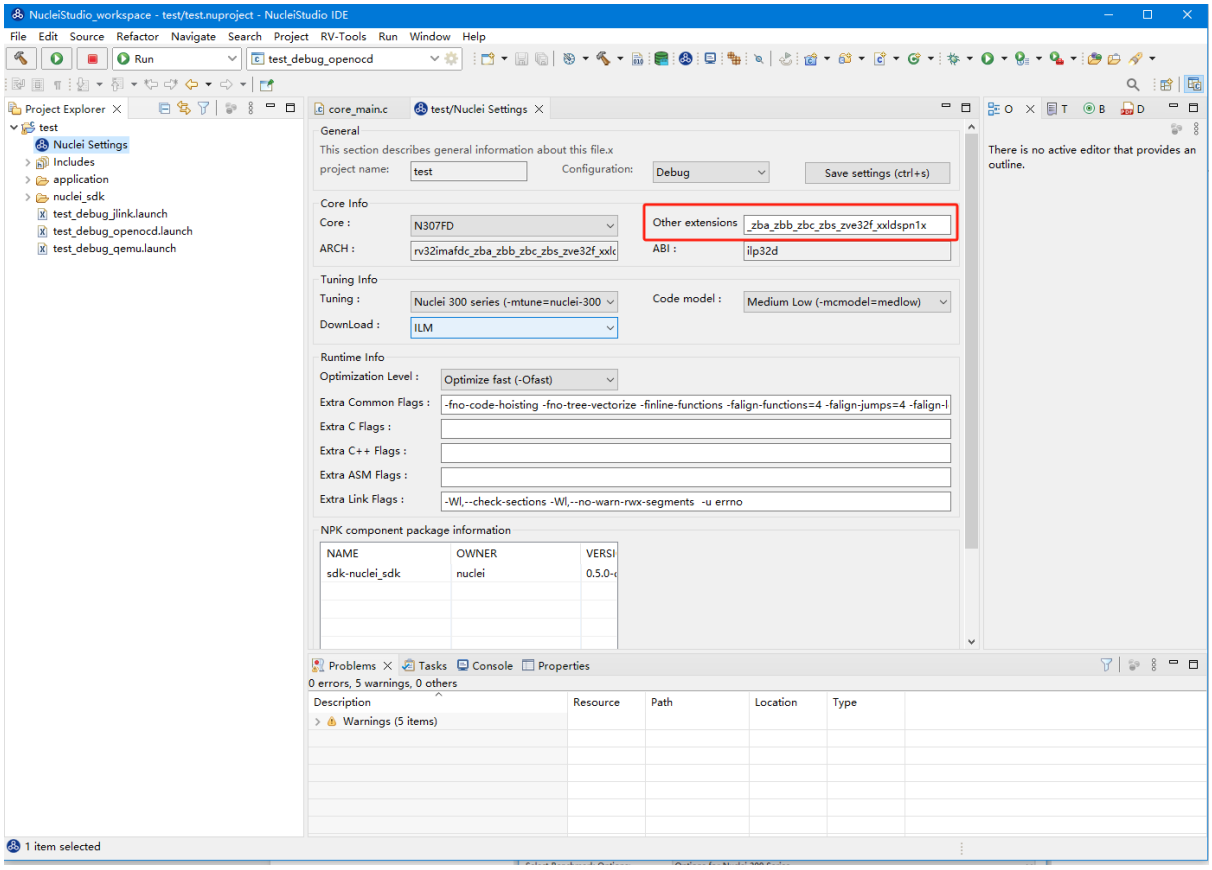
因 gcc 和 clang 的变更，在扩展的使用上，有了较大的变化。原来的 bpkv 扩展与新的规则对应关系如下，更详细的说明，请参阅https://doc.nucleisys.com/nuclei_sdk/develop/buildsystem.html#arch-ext

- `b` -> `_zba_zbb_zbc_zbs`
- `p` -> rv64: `_xxldsp`, rv32: `_xxldspn3x` for n300, `_xxldspn1x` for n900
- `k` -> `_zk_zks`
- `v` -> rv32f/d : `_zve32f`, rv64f: `_zve64f`, rv64fd: `v`

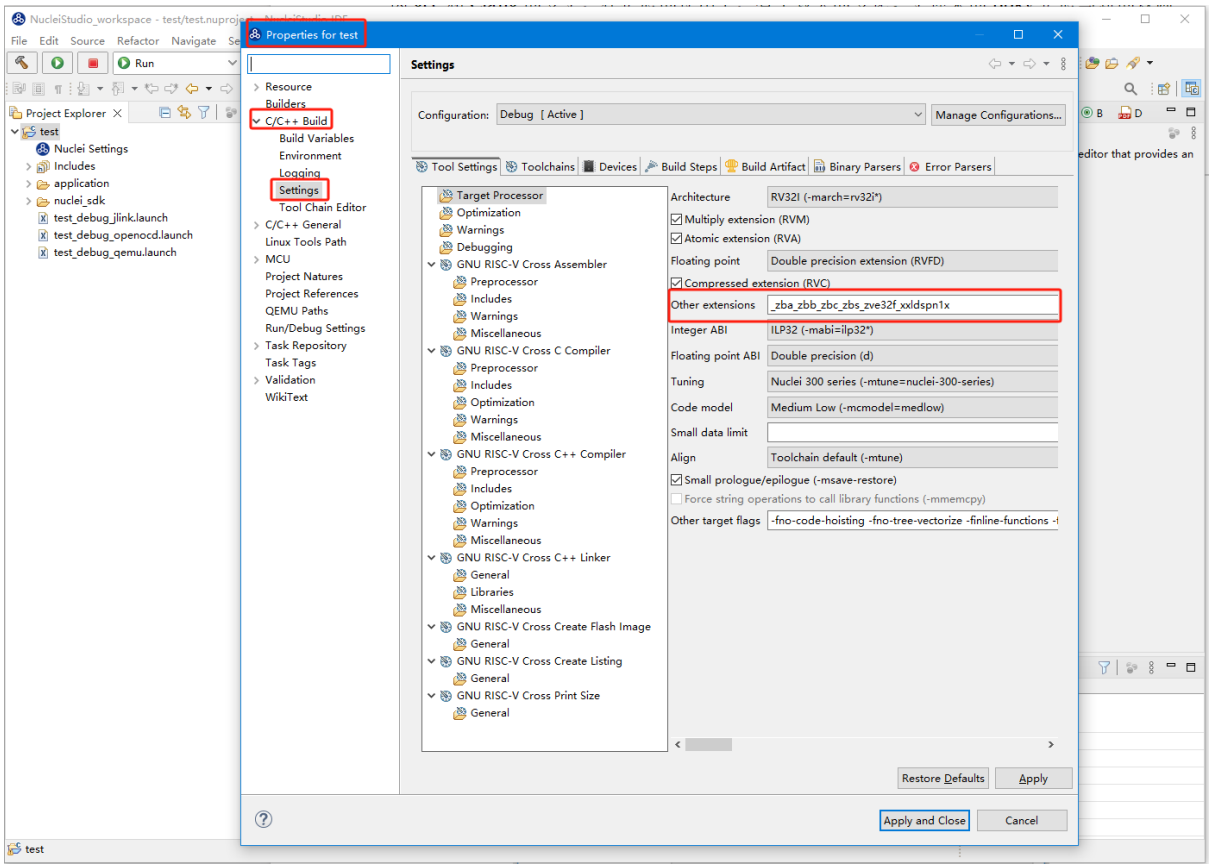
以 N307FD + B + V + Nuclei DSP with N1 extension 为例，创建一个使用扩展的应用，在创建工程的引导中，需要 Nuclei ARCH Extensions 中填入对的扩展字段，如需要使用 bpv 扩展，根据以上规则，需要填入 `_zba_zbb_zbc_zbs_zve32f_xxldspn1x` 。



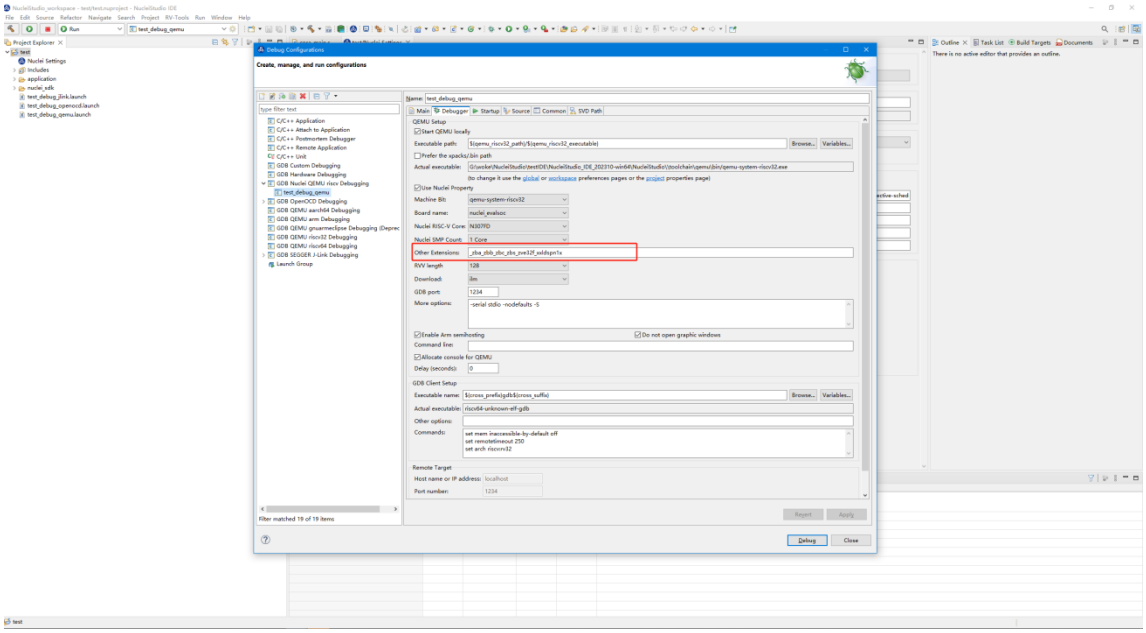
生成的工程中，可以看到在工程的 **Nuclei Settings** 。



同样的查看工程的属性，在 C/C++ Build->Settings->Target Processor 中也是有关于 RISC-V 指令扩展的配置项。

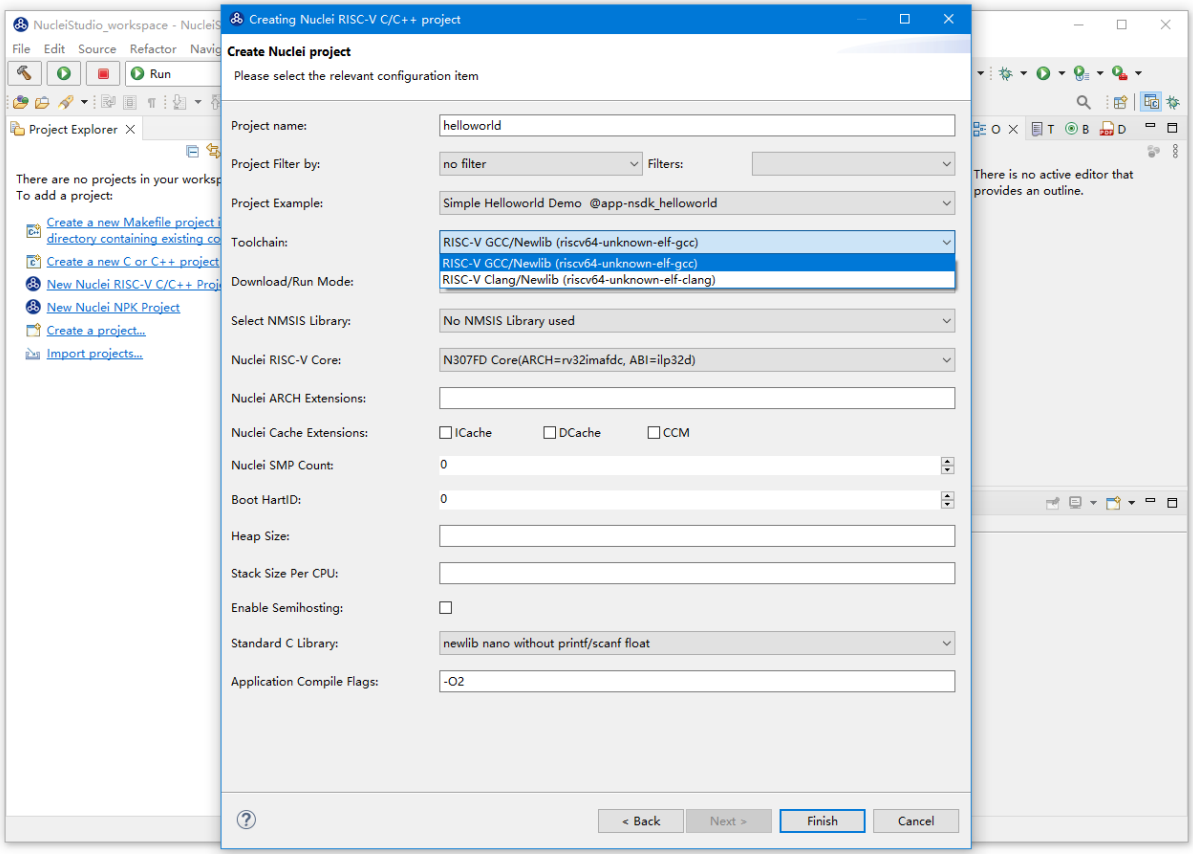


同时在 QEMU 的配置中也会有相对应的 RISC-V 指令扩展的配置项。



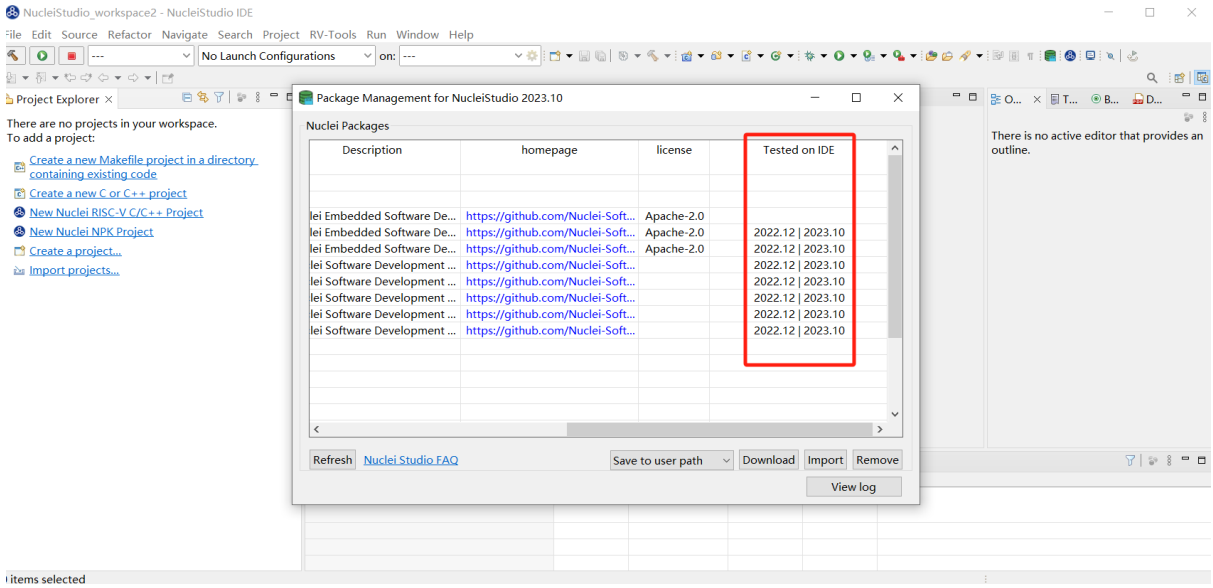
NPK 包的使用变更

为了支持 GCC 13 和 Clang 17，Nuclei SDK 包升级到了 0.5.0 版本，使用 SDK 包创建工程时，用户可以根据需要，选择创建一个 GCC 13 或者 Clang 17 的工程。因为版本变动较大，0.5.0 之前的 sdk 可能有部分功能在 Nuclei Studio 2023.10 版中使用异常，所我们提供了工具帮助您快速进行工程迁移和升级，请自行备份老版本的工程，具体可能参考导入旧版本 *Nuclei Studio* 创建的工程 (page 176)。



另外 Nuclei Studio 2023.10 中会对 npk 在线组件包做适配版本的校验（上传阶段需要填写测在什么版

本的 Nuclei Studio 上测试使用），不同的组件包所适配的 Nuclei Studio 版本号会在 Package Management 页面展示，在下载安装的时候如果版本不匹配，会给与提示，但是导入离线包不会有任何提示，请自行甄别是否被所使用的 Nuclei Studio IDE 版本所支持，具体如下。



升级 OpenOCD

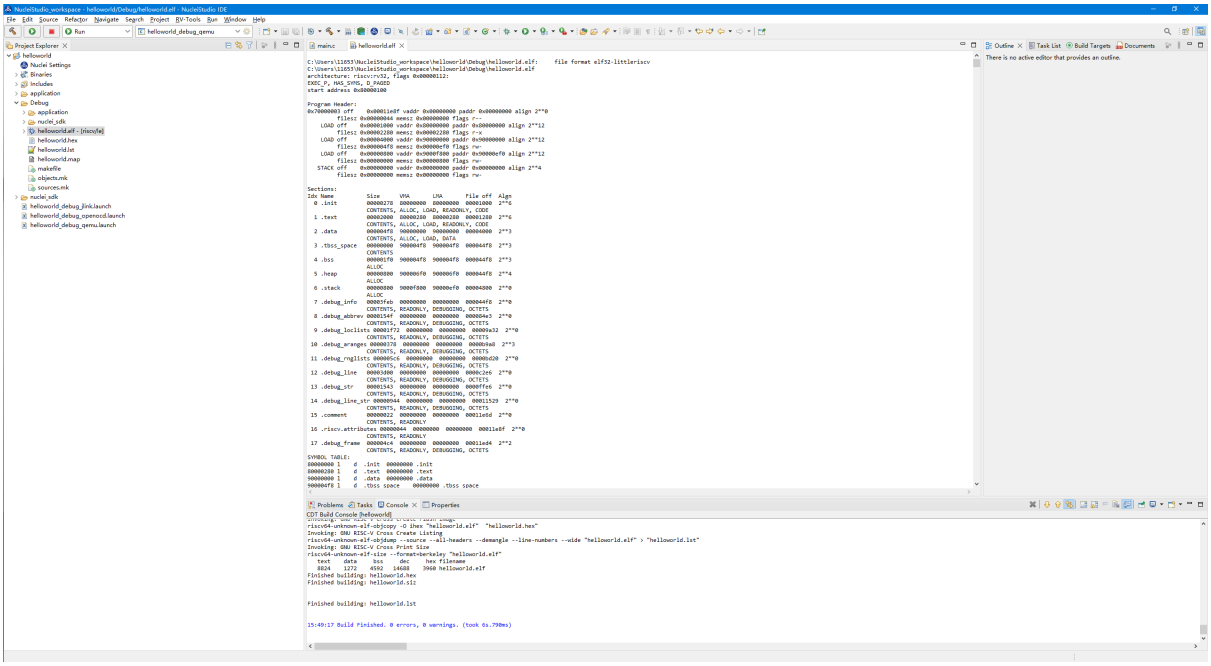
OpenOCD 版本升级至 2023.10 版，增加了一些额外的调试特性，例如查看 cpu 信息，etrace 实验性的支持。关于 OpenOCD 变更更详细的说明，请参阅：https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2023.10

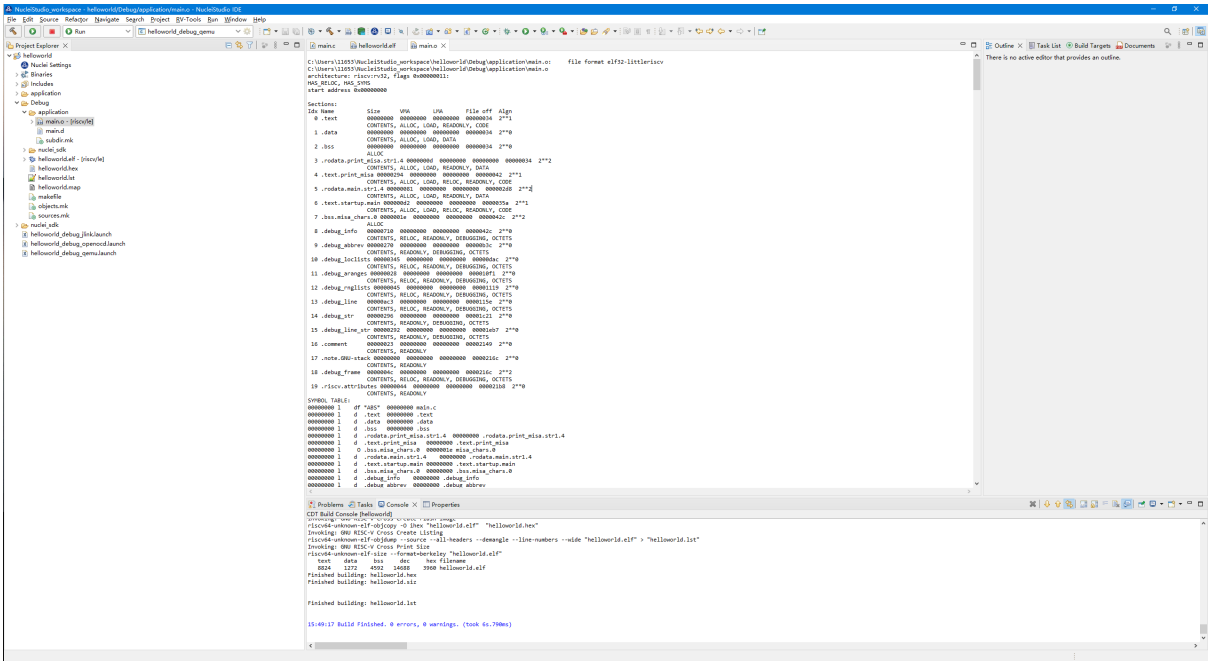
升级 QEMU

在 Nuclei Studio 2023.10 中集成 Nuclei QEMU 2023.10 版本，而 Nuclei QEMU 2023.10 基于 QEMU 8.0 进行二次开发（参考地址：https://wiki.qemu.org/ChangeLog/8.0）。本版本的 QEMU 和 2022.10 版本使用方面有比较大的变化，不再支持 gd32vf103_rvstar 这块开发板，转而只支持 Nuclei EvalSoC, 可以配置 Nuclei SDK/Nuclei Linux SDK 无缝使用。且支持的 machine 由 nuclei_n/nuclei_u 转而统一变为 nuclei_evalsoc。关于详细 Nuclei QEMU 更详细的说明，请参阅：https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2023.10

新增了 elf 文件查看器

在 Nuclei Studio 2023.10 新增 elf 文件编辑器，方便用户查看编译后产生 .elf 、.o 文件。





新增 Code Coverage 和 Profiling 功能

在 Nuclei Studio 2023.10 新增了对 Code Coverage 和 Profiling 功能的支持，具体参考*Code Coverage 和 Profiling 功能* (page 187)。

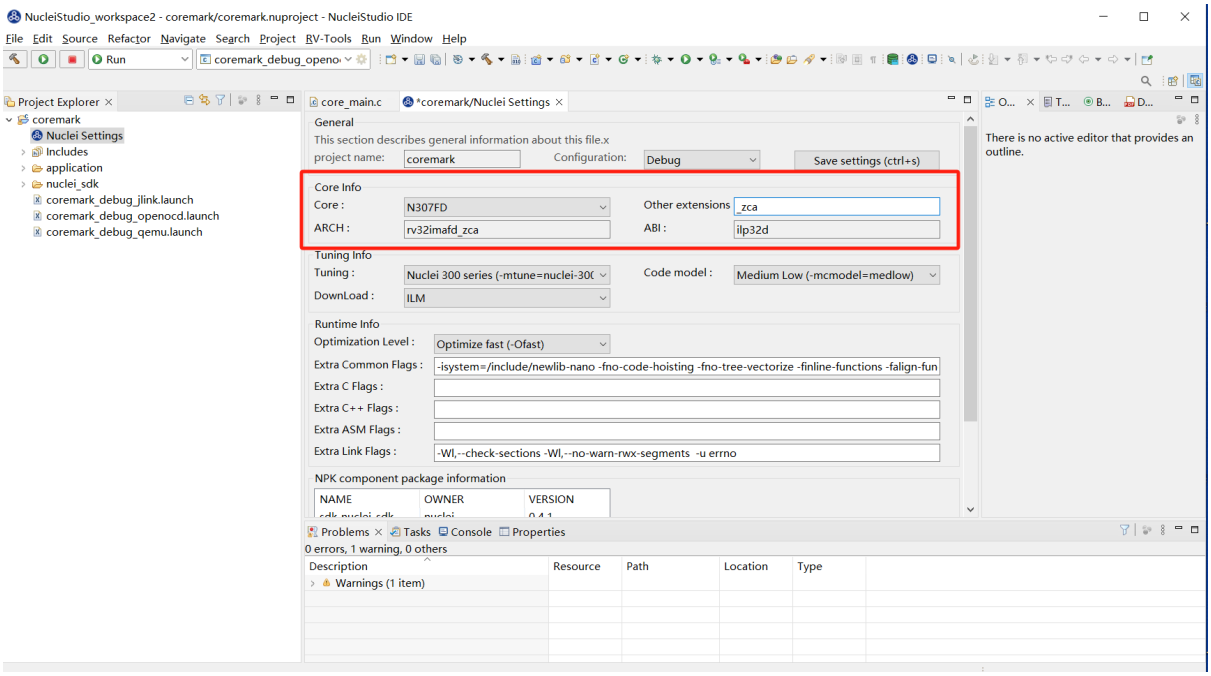
新增 trace 功能

在 Nuclei Studio 2023.10 实验性新增了 trace 功能，因使用此功能需要带有 Nuclei Trace IP 的 CPU，如需体验此功能，请与我们联系。

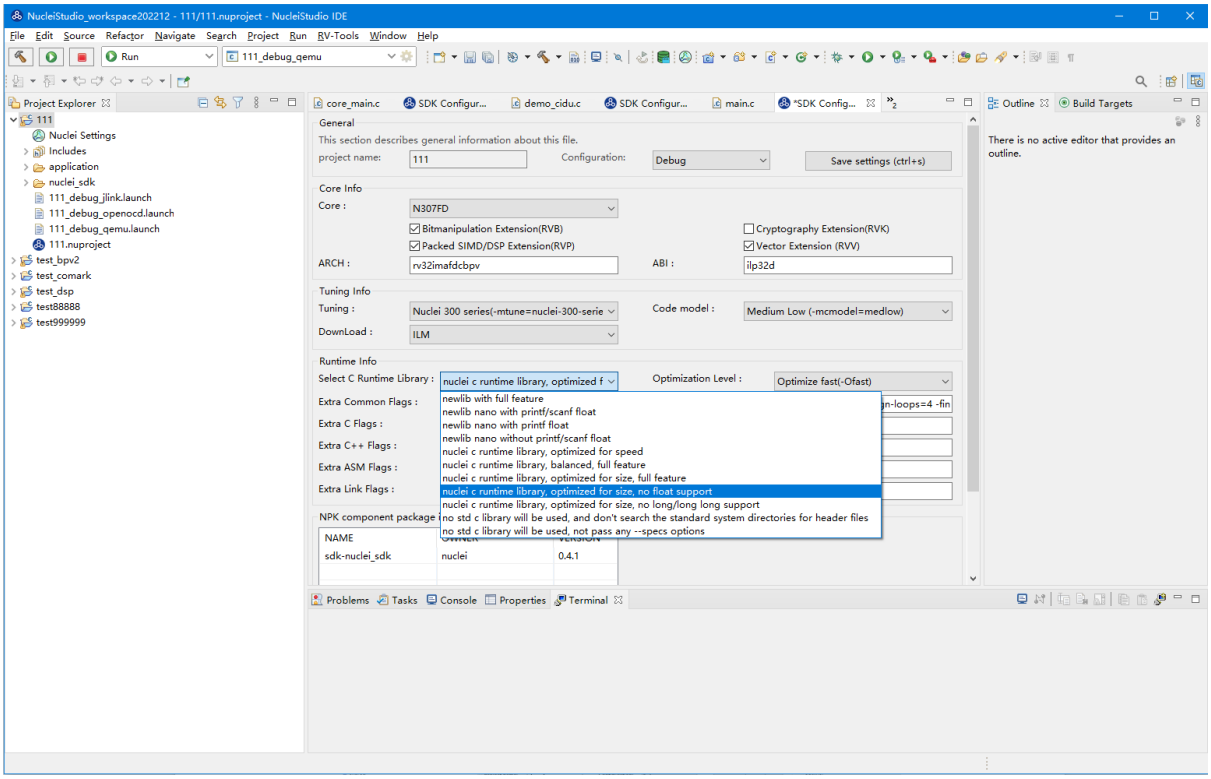
Nuclei Settings 功能优化

为了应对更个性化的配置，我们修改了 Nuclei Settings 部分功能。

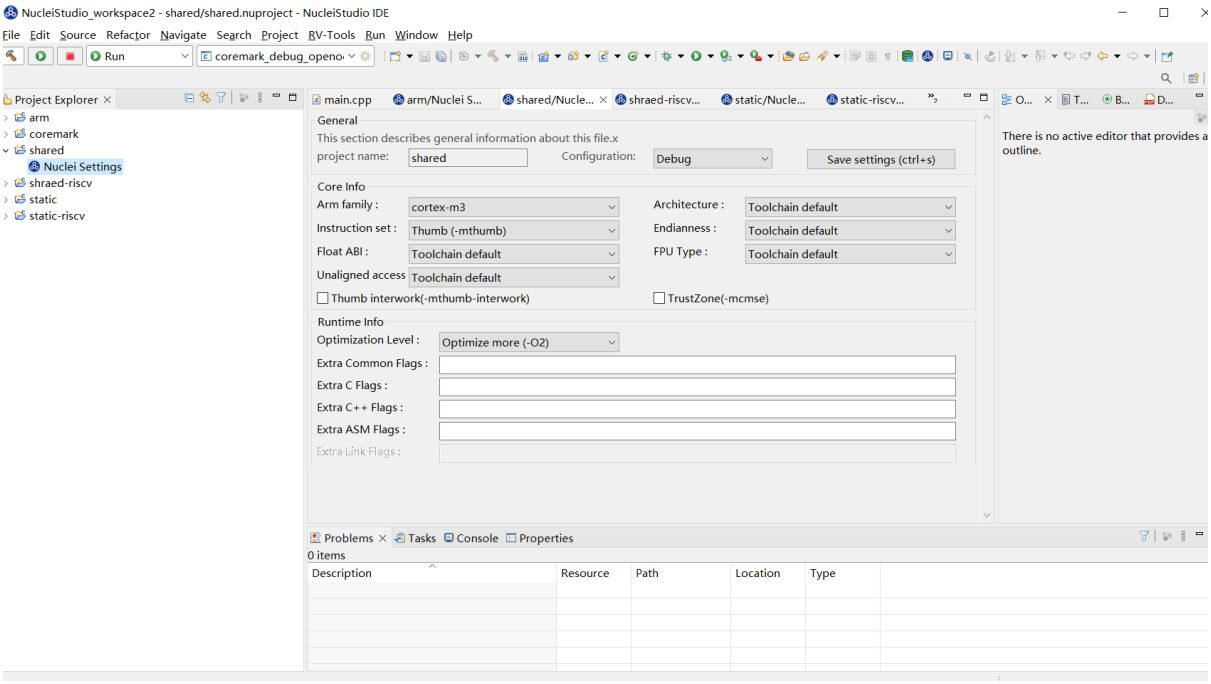
Nuclei Studio 2023.10 去掉了原来的 B/P/K/V 的单选框，换成 Other Extensions 输入框，用户可以根据自己的需求自定义填写。而关于 B/P/K/V 的使用，可以参考*RISC-V 指令扩展使用变更* (page 11)。

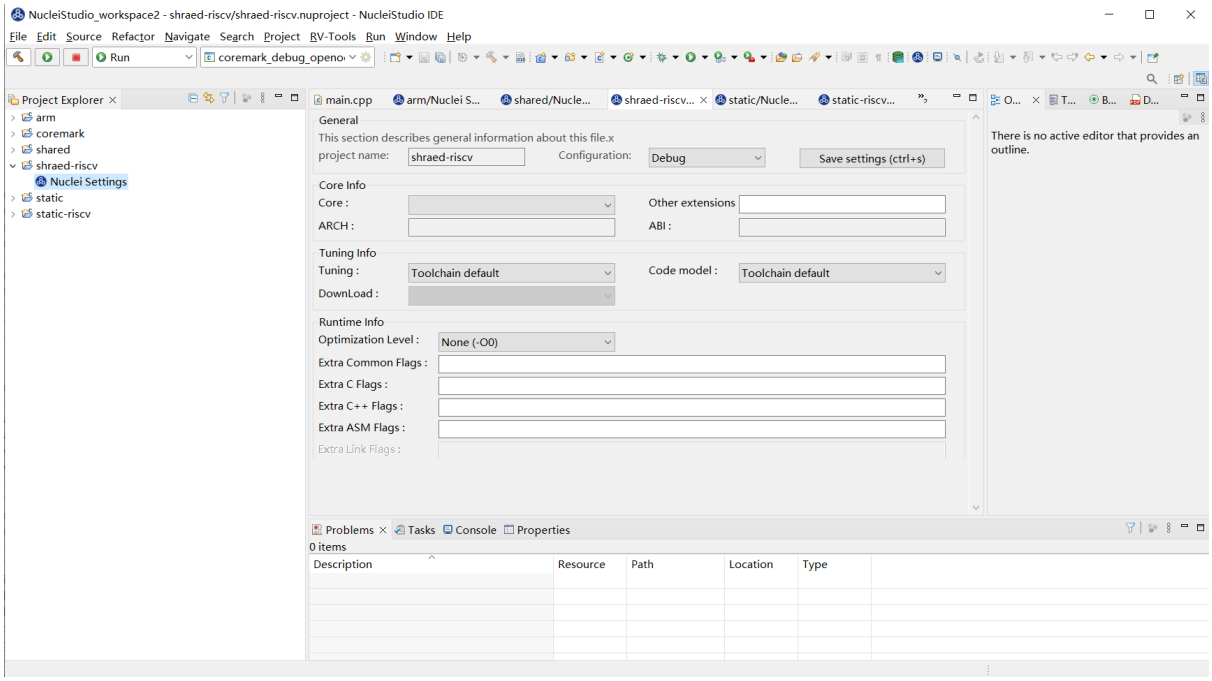


Nuclei Studio 2023.10 去掉了原来的 Select C Runtime Library 单选框，在项目中如果需要使用，可能过项目配置传入的 `--specs=` 选项，或者 Libraries 选项, 来实现。



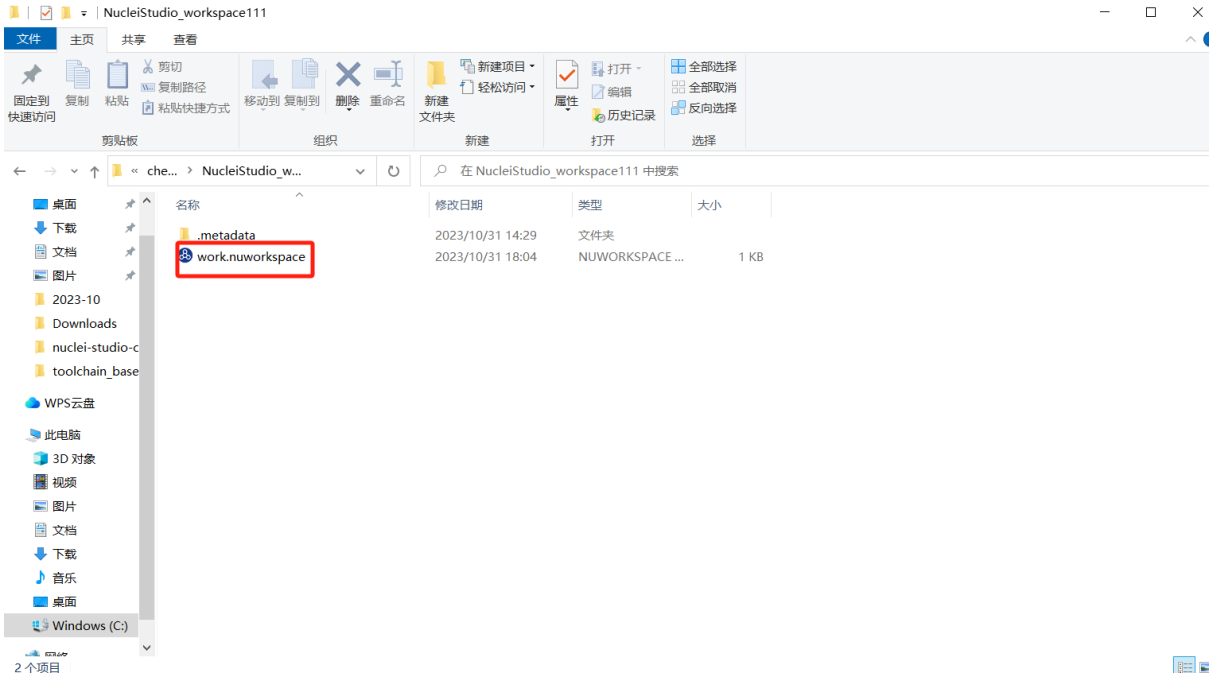
Nuclei Settings 增强了其通用性，使它不仅仅能对 Nuclei 的工程进行快速修改，也新增以对通用 riscv 和 arm 创建的 static 和 shared 的 library 工程的支持。下面为 shared 对应示图。





新增指定工作空间快速打开

类似双击项目下的 `*.nuproject` 文件可快速打开 Nuclei Studio 并导入该项目，现在 Nuclei Studio 会在使用过的工作空间目录下创建 `work.nuworkspace` 文件，双击该文件可以直接打开 Nuclei Studio，但该功能暂时只支持 windows 版本。这个功能需要解压 IDE 后，在 windows 上执行 `install.bat` 来设置文件关联。



2.2.4 2022.12 版更新说明

Nuclei Studio 自 2021.09 版后，将 IDE 与 SDK 完全分离，将采用全新的 Nuclei Package(NPK) 的包管理的方式进行模板工程的管理和使用，方便用户进行不同 SDK 的导入并且在 IDE 上创建示例工程并使用，针对 Nuclei SDK 和 HBird SDK 以及我们公司的 SoC IP 产品提供的 SDK，均可以打包成 Zip 包的方式以通过 Nuclei Package Management 方式进行导入使用。

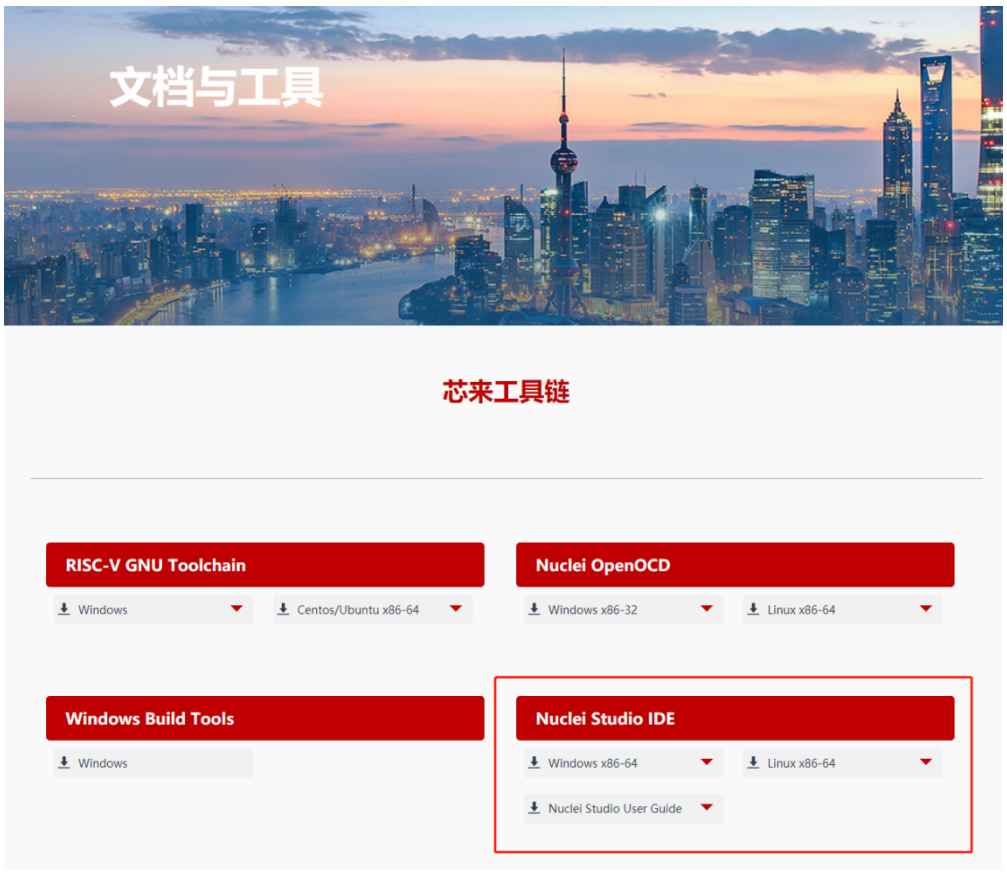
2.3 Nuclei Studio 下载与安装

2.3.1 Nuclei Studio IDE 下载

为了方便用户快速上手使用，本文档推荐使用预先整理好的 Nuclei Studio IDE 软件压缩包。芯来公司已经将该软件压缩包上传至公司网站，具体地址为https://www.nucleisys.com/download.php。

用户可以在芯来科技公司网站的“下载中心”，根据用户开发环境，下载对应 Windows 或 Linux 的 Nuclei Studio 压缩包（注意：芯来科技公司网站的下载中心，其内容会不断更新，用户请自行选择使用最新版本或继续使用当前版本）。

目前已在 Win 10 64 位系统，Ubuntu 18.04/20.04 和 Redhat7.6 64 位版本上验证测试，推荐使用以上版本的系统。



2.3.2 Nuclei Studio IDE 安装

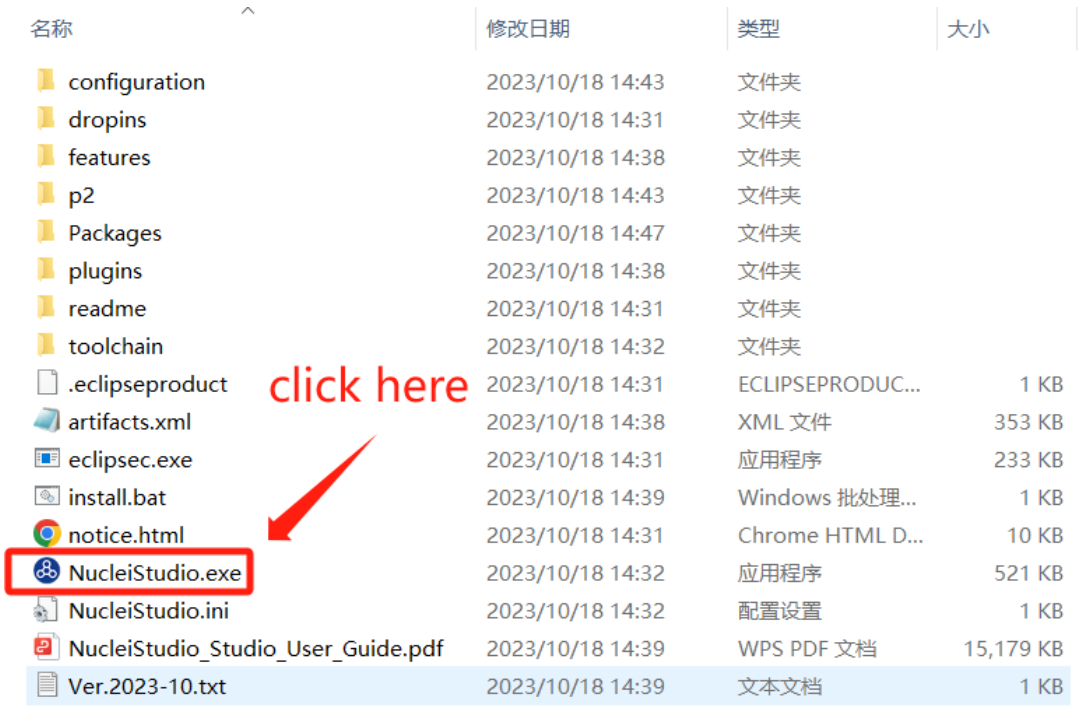
当完成 Nuclei Studio IDE 压缩软件包下载，解压后包含若干文件，分别介绍如下。

- Nuclei Studio 软件包
  - 该软件包中包含了 Nuclei Studio IDE 的软件。注意：具体版本以及文件名可能会不断更新。
- HBird_Driver.exe（2021.02 版本起不再提供）
  - 仅 **Windows** 版提供，此文件为芯来蜂鸟调试器的 USB 驱动安装文件。
  - 当在 Windows 环境下，使用该调试器时，需要安装此驱动使该 USB 设备能够被系统识别。
  - 由于 2021.02 版本中更新的 openocd 引入了免驱功能。
- SerialDebugging_Tool（2021.02 版本起不再提供）
  - 仅 **Windows** 版提供，此文件为“串口调试助手”软件。此软件可以用于后续软件示例调试时通过串口打印信息。

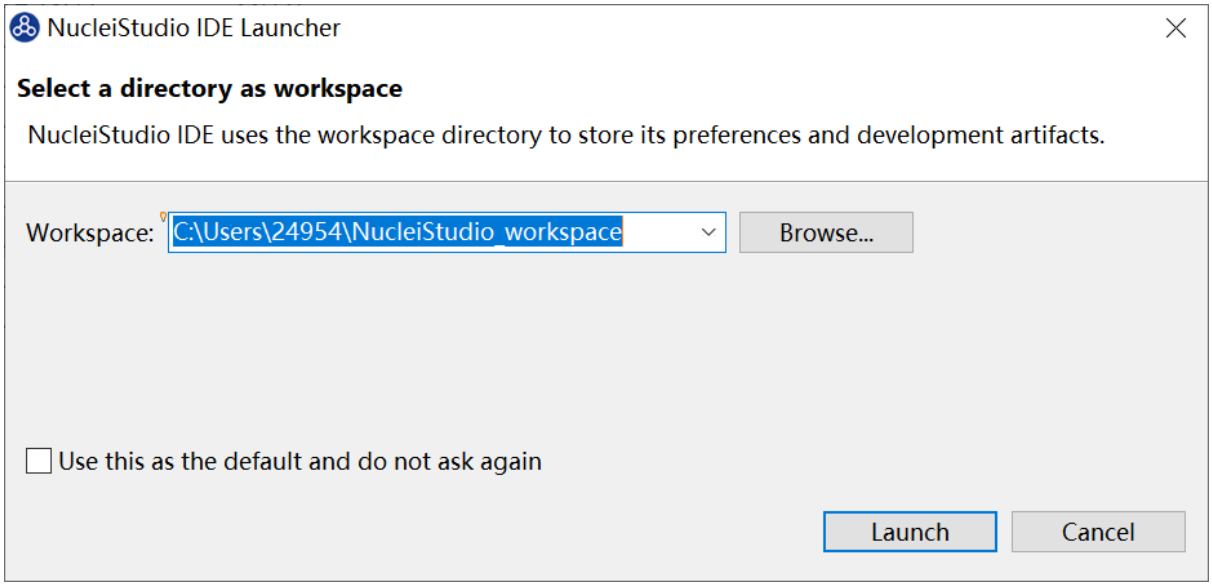
2.3.3 Nuclei Studio IDE 启动

启动 Nuclei Studio 的要点如下（windows 和 linux 均按照如下操作）：

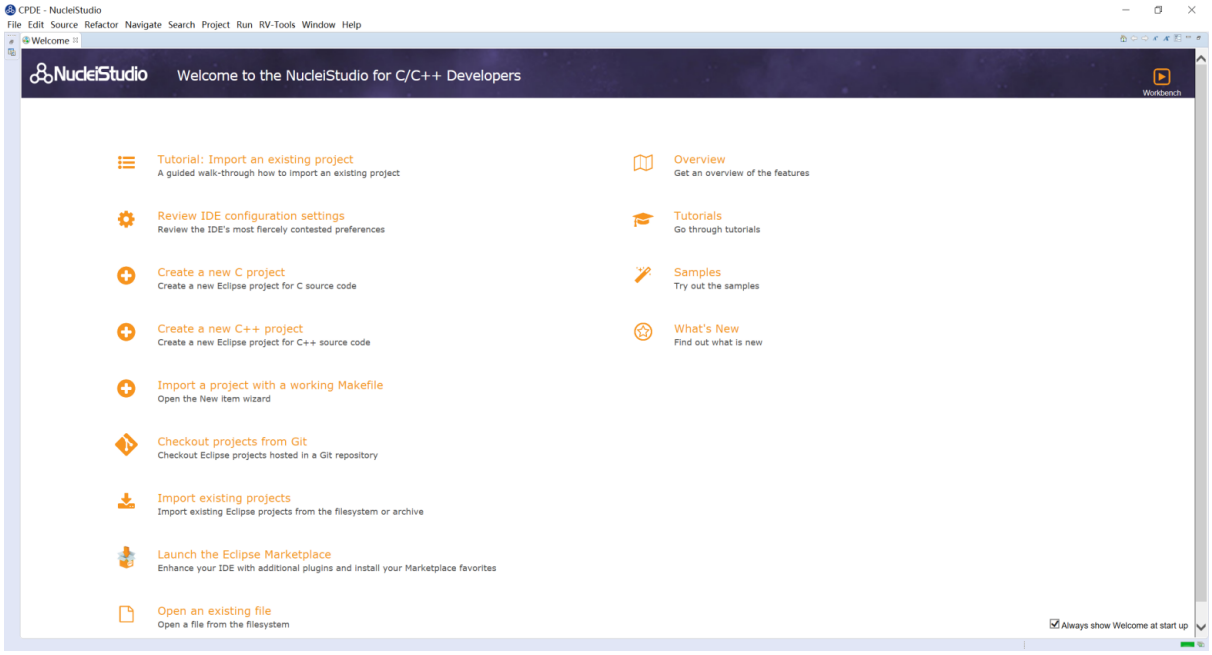
直接双击 Nuclei Studio IDE 文件包中 Nuclei Studio 文件夹下面的可执行文件，即可启动 Nuclei Studio。



第一次启动 Nuclei Studio 后，将会弹出对话框要求设置 Workspace 目录路径，该目录将用于存放后续创建的项目工程文件。



设置好 Workspace 目录之后，单击“Launch”按钮，将会启动 Nuclei Studio。第一次启动后的 Nuclei Studio。



Note: 2021.02 版本 Nuclei Studio 默认关闭了 Launch Bar，请参照 10.10.1 开启 Nuclei Studio 中的 Launch Bar 功能，方便快速编译调试和下载。

2.4 Nuclei Studio NPK 介绍

在芯来 RISC-V 嵌入式软件开发里面引入组件 (Package) 的理念，主要是借鉴于 npm 包管理，开发出方便开发者开发，使用，分发设计好的软件组件，可以大大加快软件的迭代速度。

组件包主要由以下文件组成：

- 组件描述文件 (基于 YAML 语言)：`npk.yml`
- 组件相关代码以及说明文档

组件包会以 zip 包形式存在，组件包在导入使用时，需要被 IDE 或者其他工具进行包完整性以及可用性检查，才予以导入。

Note:

- 我们提供了一个开源的 NPK 软件包检查工具，参见 https://github.com/Nuclei-Software/npk-checker
- 最好的 NPK 软件包参考项目可以参见 Nuclei SDK 以及其他线上的软件包 (搜索 npk.yml 文件) https://www.rvmcu.com/nucleistudio-npk.html
- 组件包后期会提供签名机制，在发布前先签名，然后导入软件包会验证签名，确保导入的开发包是合法身份发布，不导入不可信的开发包。

2.4.1 组件描述文件 (npk.yml)

组件主要分为以下几大类型，分别是：

- **csp:** Core Support Package, 例如 NMSIS
- **ssp:** SoC Support Package, 例如 gd32vf103 的 SoC 的支持包
- **bsp:** Board Support Package，例如 rvstar 开发板板级支持的代码包
- **osp:** RTOS Support Package，例如各类 RTOS 支持包
- **app:** Application Package，例如各类上层应用
- **mwp:** Middleware Package，各类第三方中间件，例如语音识别，算法库之类的
- **sdk:** Software Development Kit，一组预设定好的软件开发包，一般情况下里面会包含了 CSP, SSP, BSP, APP 类型的包，OSP 和 MWP 类型的包可选加入
- **bdp:** Bundle Package, 一组 package，目前仅对 app 有效

以下是比较特殊的类型，主要用于工具包和模版包

- **tool:** Tool Package，各类工具组件包，可放入其他需要引用或参考的文件
- **tpp:** Template Package, 模板类型，可以用于创建 csp/ssp/bsp/osp/app/mwp/sdk 的模板工程, 该类型比较特殊，描述文件名称为 **npk_template.yml**

描述文件通用定义

组件描述文件示例以及详细说明参见如下：

```
Package Base Information
--------------------------
name: your_package_name                    # <MUST> 组件包名称 ID，不要有空格，符合 C 语言命
名规范，英文名称，唯一名称 ID，用于 dependency 管理
                                           # name 命名规则：唯一性，全英文，不支持特殊字符
                                           # csp 类型 package: csp-xxxx
                                           # ssp 类型 package: ssp-xxxx
```

(continues on next page)

(continued from previous page)

```
                                        # bsp 类型 package: bsp-xxxx
                                        # osp 类型 package: osp-xxxx
                                        # app 类型 package: app-xxxx
                                        # mwp 类型 package: mwp-xxxx
                                        # sdk 类型 package: sdk-xxxx
                                        # tpp 类型 package: tpp-xxxx
                                        # tool 类型 package: tool-xxxx
description: 简要描述软件包的功能          # <MUST> 一句话描述清楚包的主要特性与功能, English␣
↪only, 20 字符以内
details: 详细描述软件包的功能              # <MUST> 描述清楚包的主要特性与功能, English only
version: 1.2.3                          # <OPTIONAL> 可选，如不填，默认为空，建议采用
SEMVER2.0 版本号管理，只能数字打头，例如 1.2.3
                                        # <MUST> 针对 sdk 类型的 package 必须包含一个
version tag
type: ssp                               # <MUST> 包类型， 可选类型有 csp, ssp, bsp,␣
↪osp, app, mwp, sdk, tpp, tool
os: win64                               # <MUST> OS 类型，例如 win32、win64、lin64、
lin32, 但目前组件包上传页面只支持 win64 和 lin64, 该字段只存在 tool 类型 package
category:                               # <OPTIONAL> 包分类，按照上面的包类型进行二次分类，
主要是针对 package 进行分类
keywords:                               # <MUST> 包关键字，主要用于索引与搜索
  - soc
  - risc-v
  - nuclei
license: Apache-2.0                     # <OPTIONAL> 对应包代码所用的许可证，可选列表␣
↪GPL, GPLv2, MIT, Apache License v2, BSP 等
owner: nuclei                           # <MUST>    组件包的拥有者，便于后期进行权限查找匹
配
                                        # 后期在整合到网站以后，用户在发布新的开发包的时候，
需要先注册一个唯一的 owner 名称
                                        # 然后申请一个 package name，确保 name 可用的情
况下才可以发布新的该 name 的包，同时限制单用户发布的包个数。
contributors:                           # <OPTIONAL> 组件包的代码贡献者列表
    - author_name, author_email
homepage:                               # <OPTIONAL> 组件包的主页，如果有的话，必须是链接

## Package Information
-----------------------

packinfo:                               # <MUST> Only for bsp/ssp type, optional␣
↪for other types 用于描述 SoC 层面的一些信息
  core_vendor: Nuclei                   # <MUST> only for ssp, 处理器内核的供应商英文名
称，如果是基于芯来的处理器内核，这里填写 Nuclei
  vendor: Nuclei                        # <MUST> For ssp/bsp, <OPTIONAL> for others
                                        # For ssp, SoC 或者芯片的供应商英文名称，填写对应
的厂商的名称，例如 GigaDevice
                                        # For bsp, 表示板子的提供商英文名称
                                        # For others, OPTIONAL, 表示该 package 提供商
  name: demosoc                         # <MUST> 这里 name 和 package name 作用不一样,
↪ 用来显示具体的包名称
                                        # For ssp, 表示本组件包中 SoC 的具体英文名称，可
以是一个系列名称，例如 GD32VF103
                                        # For bsp, 板子的名称
                                        # For others, 表示该 package 显示名称，中英文均
可，不写的话，默认使用根字段 name
  doc:                                  # <OPTIONAL>, 不填的话，这里为空，不做展示
    website:                            # <OPTIONAL>, Package 对应的网站
    sch:                                # <OPTIONAL> only for bsp, 表示板子的电路图
    datasheet:                          # <OPTIONAL>, package datasheet 本地地址或者是
网址
    usermanual:                         # <OPTIONAL>, Package User Manual
```

(continues on next page)

(continued from previous page)

```
    extra:                                  # <OPTIONAL>, 额外的一些资料，列表形式 (URI +␣
↪DESCRIPTION)
      - uri:                                # <OPTIONAL>, file path or web link
        description:                        # <MUST> when uri is defined, description␣
↪for file


## Package Dependency
----------------------

dependencies:                               # <OPTIONAL> 列出依赖的组件包列表 owner/
↪name:version
  - name: csp-nsdk_nmsis                    # <MUST> when defined        # 1. 针对 sdk
类型的包，内部有一个隐形的依赖，依赖有且仅有一个 bsp 类型的包，
    owner: nuclei # <OPTIONAL> if not defined, it will use the owner definiton␣
↪above. 用于依赖特定所有者的 package， owner/name:version， 最终查找的包按照这样来找的
      version: # <OPTIONAL> when defined empty, use default as version number #  ␣
↪并且会自动查找当前路径下所有的 npk.yml 文件，并作为 sdk 包中一部分
                                            # 2. 除了 sdk 类型的包之外的其他的包，如果不是放在
sdk 包下面的目录，则依赖于
                                            #    sdk，则需要显式加上 dependency
                                            # 3. bsp 类型的 package 肯定会依赖一个 ssp 类型
的 package
                                            # 4. 依赖包也可以带上版本号，支持版本号条件比对，如
果不带版本号，则优先选择不带版本号的包，其次最新的包
                                            # 参考 https://docs.platformio.org/en/
↪latest/librarymanager/config.html#version
                                            # https://docs.npmjs.com/about-semantic-
↪versioning
                                            # 1.2.3 - an exact version number. Use␣
↪only this exact version
                                            # ^1.2.3 - any compatible version (exact␣
↪version for 1.x.x versions
                                            # ~1.2.3 - any version with the same major␣
↪and minor versions, and an equal or greater patch version
                                            # >1.2.3 - any version greater than 1.2.3.␣
↪>=, <, and <= are also possible
                                            # >0.1.0,!=0.2.0,<0.3.0 - any version␣
↪greater than 0.1.0, not equal to 0.2.0 and less than 0.3.0
                                            #    例如 version: master, version: 1.2.3,␣
↪version: >0.1.0
                                            # 依赖关系处理规则
                                            # 1. sdk，app 类型的包可以依赖多个 ssp、bsp 类型
的包，但是最终只会根据 project wizard 选择具体使用到 package
                                            # 2. app/bsp/ssp/csp/osp/mwp 类型的包可以依赖
sdk，如果依赖了 sdk 类型的包，则表述该包隶属于 sdk 下
                                            # 3. bsp 类型的包只能依赖一个 ssp/csp 类型的包，
ssp 类型的包只能依赖一个 csp 类型的包
                                            # 4. sdk 类型的包可以依赖多个 app/bsp/ssp/csp/
↪osp/mwp 类型，但是这种依赖只是建立从属关系，表示该 sdk 包含了这些包
                                            # 5. 一个 sdk 类型的包不可以依赖另一个 sdk 包，但
是 app/bsp/ssp/csp/osp/mwp 却可以依赖多个 sdk 类型的包

## Package Configurations
--------------------------

configuration:                              # <OPTIONAL> 关于包配置的一些选项，用于 Project␣
↪Wizard 创建以及内部参数设置
                                            # 其中 sdk 类型暂不支持 configuration 参数定义
                                            # configuration 定义的配置可以互相覆盖
                                            # 覆盖规则为 app > mwp  > osp > bsp > ssp >␣
```

(continues on next page)

(continued from previous page)

```
↪csp
  nuclei_core:                                # <OPTIONAL> 一个配置选项，类型为 choice
    default: n307fd                           # <MUST> 如果这个配置定义了，针对 choice 类型，默
认值选择必须是 choices 里面列举的
    type: choice                              # <MUST> 配置类型，可选有 choice, list,␣
↪checkbox, multicheckbox，text
    global: true                              # <OPTIONAL> 可选为 true 或者 false，默认为
true
    description: Nuclei RISC-V Core           # <MUST> 该配置项的描述，20 字以内
    choices:                                  # <MUST> 当配置项 type == choice 时
      - name: n201                            # <MUST> item 中必须包含 name 和 description
        arch: rv32iac                         # <OPTIONAL> 仅用于表示 RISC-V CORE 中的 ARCH
信息，不建议随意使用
        abi: ilp32                            # <OPTIONAL> 仅用于表示 RISC-V CORE 中的 ABI
信息，不建议随意使用
        tune:                                 # <OPTIONAL> 仅用于表示 RISC-V CORE 中的 TUNE
信息，不建议随意使用
        info:                                 # <OPTIONAL> 用于自定义 key-value 数据的访问，例
如 ${nuclei_core.info.key1} 返回的是 value1
          - name: key1                        # <MUST> key in pair with value, 字符串类型，
不包含任意空格
            value: value1                     # <MUST> value in pair with key，字符串类型
          - name: key2                        # <MUST>
            value: value2                     # <MUST>
        description: N201 Core(ARCH=rv32iac, ABI=ilp32)     # <MUST> 描述这个 item 具
体含义
                                              # 除了 name, description 之外，可能会定义其他
items, 名称不定
                                              # 例如在这里就定义了 arch 和 abi
      - name: n201e                           # 另一个 item
        arch: rv32eac
        abi: ilp32e
        description: N201E Core(ARCH=rv32eac, ABI=ilp32e)
  extra_flags:                                # <OPTIONAL> 一个配置选项，当前这个为
text 类型
    value:                                    # <MUST>, 仅接受英文字符串
    description: Extra compiler flags         # <MUST> 该配置项的描述，30 字以内
  dsp_present:                                # <OPTIONAL> 一个配置选项，当前这个为
checkbox 类型
    default: 0                                # <MUST>, 默认为 0，可选 0 或者 1
    type: checkbox                            # <MUST>, 配置类型，当前为 checkbox
    global: true                              # <OPTIONAL> 可选为 true 或者 false，默
认为 true
    description: P-Extension(DSP) Present     # <MUST> 该配置项的描述，30 字以内
  libraries:                                  # <OPTIONAL> 一个配置选项，当前这个为
multicheckbox 类型
    default: [dsp, nn]                        # <MUST> 默认值，为 choices 里面的组合
    type: multicheckbox                       # <MUST>, 配置类型，当前为
multicheckbox
    global: true                              # <OPTIONAL> 可选为 true 或者 false，默
认为 true
    description: Libraries Used               # <MUST> 该配置项的描述，30 字以内
    choices:                                  # <MUST> 当配置项 type ==␣
↪multicheckbox 时
      - name: dsp                             # <MUST> item 中必须包含 name 和
description
        description: DSP Library              # <MUST> 描述这个 item 具体含义
                                              # 除了 name, description 之外，可能会定
义其他 items, 名称不定
      - name: nn
        description: NN Library
```

(continues on next page)

(continued from previous page)

```
      - name: ai
        description: AI Library

## Source Code Management
-------------------------

codemanage:                                     # <MUST> 这个为必选项
  installdir: demosoc                           # <MUST> 希望代码安装的目录名称，仅限英文，满
足 C 语言命名格式
                                                # 针对 sdk 类型的 package, 会被安装到<sdk_
↪installdir>, 如果 installdir 未定义，默认为 SDK，如果没有任何 sdk 类型的 package 被引用，
sdk_installdir 也被默认设置为 SDK
                                                # 针对 csp 类型的 package，会被安装到<sdk_
↪installdir>/<csp_installdir> 目录下，TBD
                                                # 针对 ssp 类型的 package, 会被安装到<sdk_
↪installdir>/SoC/<ssp_installdir>/Common 下面
                                                # 针对 bsp 类型的 package, 会被安装到<sdk_
↪installdir>/SoC/<ssp_installdir>/Board/<bsp_installdir> 下面，如果不依赖于任何 ssp 类
型，则安装到<sdk_installdir>/BSP/<bsp_installdir>
                                                # 针对 osp 类型的 package, 会被安装到<sdk_
↪installdir>/OS/<osp_installdir> 下面
                                                # 针对 app 类型的 package, 会被安装到<app_
↪installdir>/目录下， 如果 installdir 未定义，默认为 application
                                                # 针对 mwp 类型的 package, 会被安装到<sdk_
↪installdir>/Components/<mwp_installdir> 目录下
  copyfiles:                                    # <MUST> 待拷贝的文件或者文件夹，支持 glob␣
↪pattern 匹配，这里是指所有的目录或者文件
    - path: ["Source/", "Include/", "demosoc.svd"]          # <MUST> 待拷贝的文件或者文件
夹的路径列表，支持 glob pattern 匹配
    - path: ["DSP_Source", "DSP_Include"]
      condition: $( ${dsp_present} == 1 )                   # <OPTIONAL> 这里的 if 是一
个固定的标识符，如果出现则表示要做判定，判定的方式如下
                                                          # 如 dsp_present 是在
configuration 里面定义的，根据 wizard 或者其他 package 选定而定
  incdirs:                                                  # <OPTIONAL> 需要加入头文件
目录列表
    - path: ["Include/"]                                    # <OPTIONAL> 需要加入头文件
的目录
  libdirs:                                                  # <OPTIONAL> 可选的 lib 库
所在目录
      - path:
  ldlibs:                                                   # <OPTIONAL> 可选的需要链接
的库
      - libs:

## Set Configuration for other packages
---------------------------------------

setconfig:                                                  # <OPTIONAL> 这个用于设置其
他 Package 的选项

# 以下选项是覆盖关系，规则 app > mwp  > osp > bsp > ssp > csp
  - config: nmsislibarch
    value : ${nuclei_core.arch}p                            # <OPTIONAL> 直接设置
Configuration 里面的选项
    condition: $( ${dsp_present} == 1 )                     # <OPTIONAL> 根据这里 dsp_
↪present 来判断是否设置 nmsislibarch 值
  - config: nmsislibarch
    value: ${nuclei_core.arch}
    condition: $( ${dsp_present} == 0 )
```

(continues on next page)

(continued from previous page)

```
## Build Configuration
-----------------------

buildconfig:                                    # <OPTIONAL> 编译选项的配置
                                                # 目前编译选项会将 package 中定义的所有的拼
接在一起或者覆盖
                                                # 以下选项是覆盖方式，app > mwp  > osp >␣
↪bsp > ssp > csp
                                                # type 是一个特殊字段，用于标识特定的编译器,␣
↪目前支持 gcc
                                                # cross_prefix, prebuild_steps,␣
↪postbuild_steps, description
                                                # 其余选项是拼接的
  - type: gcc                                   # <OPTIONAL> 目前只有 gcc，预留其他接口
    description: Nuclei GNU Toolchain           # <MUST> For ssp
      cross_prefix: riscv-nuclei-elf-           # <OPTIONAL> 如果不写或者留空，就自动按照系
统里面提供的工具链来定
    common_flags:                               # <OPTIONAL> 通用的编译选项，将会添加到
cflags, asmflags, cxxflags 上
      - flags: -g -fno-common -ffunction-sections -fdata-sections
      - flags: -march=${nuclei_core.arch} -mabi=${nuclei_core.abi} -mcmodel=medany
    ldflags:                                    # <OPTIONAL> 链接选项列表，留空表示没有任何
选项
      - flags: -nostartfiles --specs=nosys.specs
      - flags: --specs=nano.specs
        condition: $(${newlib} != "normal")
      - flags: -u _printf_float
        condition: $(${newlib} != "nano_with_printfloat")
      - flags: -u _isatty -u _write -u _sbrk -u _read -u _close -u _fstat -u _lseek
    linkscript:                                 # <MUST> 链接脚本的定义，必须在 bsp/ssp 中
定义
      - script: "Source/GCC/gcc_demosoc_${.download}.ld"
          condition: $(check pattern)           # <OPTIONAL> 进行条件判断
    cflags:                                     # <OPTIONAL> C 编译选项，留空表示没有任何选
项
      - flags: -O3
    asmflags:                                   # <OPTIONAL> ASM 编译选项，留空表示没有任何
选项
      - flags: -O2
    cxxflags:                                   # <OPTIONAL> CXX 编译选项，留空表示没有任何
选项
      - flags: -O1
    common_defines:                             # <OPTIONAL> 通用的宏定义
      - defines: __RISCV_FEATURE_DSP=1
        condition: $(${dsp_present} == 1)
      - defines: DOWNLOAD_MODE_STRING=\"flashxip\"
    cdefines:                                   # <OPTIONAL> C 的宏定义
      - defines: __RISCV_FEATURE_DSP=1
        condition: $(${dsp_present} == 1)
      - defines: DOWNLOAD_MODE_STRING=\"flashxip\"
    asmdefines:                                 # <OPTIONAL> ASM 的宏定义
      - defines: __RISCV_FEATURE_DSP=1
        condition: $(${dsp_present} == 1)
      - defines: DOWNLOAD_MODE_STRING=\"flashxip\"
    cxxdefines:                                 # <OPTIONAL> CXX 的宏定义
      - defines: __RISCV_FEATURE_DSP=1
        condition: $(${dsp_present} == 1)
      - defines: DOWNLOAD_MODE_STRING=\"flashxip\"
    prebuild_steps:                             # <OPTIONAL> 编译前执行的命令
      command:                                  # <OPTIONAL> 执行的命令行
      description:                              # <OPTIONAL> 执行的命令的描述
```

(continues on next page)

(continued from previous page)

```
    postbuild_steps:                              # <OPTIONAL> 编译完成后执行的命令
      command:                                    # <OPTIONAL> 执行的命令行
      description:                                # <OPTIONAL> 执行的命令的描述

## Debug Configuration
------------------------

debugconfig:                                  # <MUST> For bsp type, optional for app/
↪ssp type
# 目前 Debug 选项会将 package 中定义的所有的拼接在一起或者覆盖
# type 是一个特殊的字段，用于描述特定的调试器，目前支持 openocd, qemu
# 以下字段是覆盖关系：app > mwp  > osp > bsp > ssp > csp
# description, svd
# configs 字段下面的 key, value 是合并关系，如果对应的 key 存在就覆盖，覆盖规则同上，如果不存在
就合并
  - type: openocd   # <MUST> 选择的工具
    description: Nuclei OpenOCD # <MUST> For bsp type
    svd: gd32vf103.svd  # <OPTIONAL> 可选的 SVD 文件
    configs:
      - key: config  # openocd 配置文件
        value: "openocd_gd32vf103.cfg"

  - type: qemu
    description: Nuclei QEMU
    svd:
    configs:
       - key: nuclei_core    # Nuclei RISC-V Core
         value: ${nuclei_core}
         condition:  # condition set nuclei_core key
       - key: download_mode  # Download mode
         value: ${download_mode}
       - key: riscv_arch     # RISCV ARCH
         value: ${nuclei_core.arch}
       - key: riscv_abi      # RISCV ABI
         value: ${nuclei_core.abi}
       - key: machine    # QEMU Machine
         value: gd32vf103v_rvstar

##Extended variable
## Only works on tool 类型
## 每个包存在一个包路径，引用为 npk 名称-版本号，例如${tool-cmlink-1.0.0}，
## 其他变量的引用为 npk 名称-版本号-变量名，例如 ${tool-cmlink-1.0.0-proxy}
environment:              # 扩展变量
  - key: proxy              # 变量名，
    value: bin/cmlink_gdbserver.exe         # 实际引用结果为 npk 文件父路径 +value，例如
C:\Users\jj\nuclei-pack-npk\NPKs\XinShengTech\Tool_Package\tool-cmlink\1.0.0\
↪cmlink\bin\cmlink_gdbserver.exe
    description: proxy location
    system: true     # 默认为 fasle，当 system 为 true 时，该变量引用时直接使用变量名，例如$
↪{proxy}

## Template File Management
## Only works on tpp 类型，该类型比较特殊，描述文件为 npk_template.yml，是基于 npk.yml 做的
扩展
templatemanage:
  installdir: ${soc}
  files:
    build.mk.ftl: build.mk
    Common/npk.yml.ftl: Common/npk.yml
    Common/demosoc.svd.ftl: Common/${soc}.svd
    Common/Source/demosoc_common.c: Common/Source/${soc}_common.c
```

(continues on next page)

(continued from previous page)

```
    Common/Source/system_demosoc.c.ftl: Common/Source/system_${soc}.c
    Common/Source/Drivers/demosoc_uart.c.ftl: Common/Source/Drivers/${soc}_uart.c
    Common/Source/GCC/intexc_demosoc.S.ftl: Common/Source/GCC/intexc_${soc}.S
    Common/Source/GCC/startup_demosoc.S.ftl: Common/Source/GCC/startup_${soc}.S
    Common/Source/Stubs: Common/Source/Stubs
    Common/Include/demosoc.h.ftl: Common/Include/${soc}.h
    Common/Include/demosoc_uart.h.ftl: Common/Include/${soc}_uart.h
    Common/Include/nuclei_sdk_soc.h.ftl: Common/Include/nuclei_sdk_soc.h
    Common/Include/system_demosoc.h.ftl: Common/Include/system_${soc}.h
    Board/nuclei_fpga_eval/openocd_demosoc.cfg: Board/${board}/openocd_${soc}.cfg
    Board/nuclei_fpga_eval/npk.yml.ftl: Board/${board}/npk.yml
    Board/nuclei_fpga_eval/Source/GCC/gcc_demosoc_ilm.ld.ftl.ftl: Board/${board}/
↪Source/GCC/gcc_${soc}_ilm.ld
    Board/nuclei_fpga_eval/Source/GCC/gcc_demosoc_flash.ld.ftl: Board/${board}/
↪Source/GCC/gcc_${soc}_flash.ld
    Board/nuclei_fpga_eval/Source/GCC/gcc_demosoc_flashxip.ld.ftl: Board/${board}/
↪Source/GCC/gcc_${soc}_flashxip.ld
    Board/nuclei_fpga_eval/Include/board_nuclei_fpga_eval.h.ftl: Board/${board}/
↪Include/board_${board}.h
    Board/nuclei_fpga_eval/Include/nuclei_sdk_hal.h.ftl: Board/${board}/Include/
↪nuclei_sdk_hal.h
```

内容约定

为了保证 npk.yml 文件的可读性与简约性，对 npk.yml 文件的存储制定如下约定：

- 各字段的存储顺序请保持与模板一致，数据与 DICT 按读入时顺序保存
- 各字段建议适当加上注释，尤其是那种需要解释的地方
- **MUST** 类型的字段需要按照上述注释描述的规则进行检查，如果不合规请报错提示，并不予导入
- 缩进建议采用 2 个空格字符
- 针对一些 **OPTIONAL** 的字段可以留空或者不写该字段
- 一级字段之间增加一行空行，二级及以下字段不使用空行，第一部分基础信息一级字段间不使用空行
- 字符串建议不使用引号，除特殊语法需要
- 所有的 `description`字段建议控制在 20 字符以内，方便排版展示，仅限英文
- 关于 yaml 里面多行的约定如下：https://yaml-multiline.info/

包导入规则

下面定义合法的包导入规则：

- 如果存在导入包中存在一些依赖的包 (带版本匹配)，并没有被导入，则不允许导入，并提示缺乏依赖的包，请导入该包。
  - 后续包管理联网了，则可以提示是否从网上下载依赖的包，或者手动导入 zip 包
- 如果删除包，并且该包被其他包依赖，则提示哪些包依赖于该包，询问是否删除，如果删除以后，则在包管理中显示缺少的包
  - 后续包管理联网了，支持点击按钮下载缺失的包，或者手动导入 zip 包
- 如果导入相同版本的包，则提示该包已经存在，是否替换
- 如果导入不同版本的包，则提示已经存在其他版本的包，是否继续导入
- 导入的包，按照定义的类型分类显示，显示包的版本，包的 name，owner，description, homepage, license

zip 包内容规范

下面定义合法的 zip 包的内容规则：

- 一个 zip 包中必须包含至少 1 个 npk 文件
- 包类型的判定：如果包内存在多个类型的 **npk** 文件，**npk** 类型判定条件如下
  - sdk > ssp > bsp > osp > mwp > csp > app
  - 如果判定出包的类型存在多个相同的 npk，则该包不合法，不允许导入，并提示
- 该类型的包不允许存在多个该类型的 npk 文件
- 如果是 `sdk` 类型的包，则必须包含至少一个 `ssp` 和依赖于该 `ssp` 的 `bsp` 文件，以及至少一个 `app` 类型的文件，允许存在其他类型的包
- 如果是其他类型的包，则里面包含的其他 npk，必须显式依赖于该包

包依赖关系处理

包依赖关系的处理涉及到如何能够将包拆分并形成合理的依赖关系，便于包的独立维护。这里对不同类型的包的依赖处理进行详细的分析。

依赖通过 `dependencies` 字段下的依赖列表来控制，支持依赖特定 owner 的某个 name，某个 version 版本的包，查找规则为 `owner/name:version`，如果 owner 未定义，则默认为该 npk 文件中定义的 owner，如果 `version` 未定义，则优先在同组件包查找，否则取最新的包。

csp Core Support Package 依赖

csp 类型的包是处理器内核 CORE 支持的软件包，目前针对 Nuclei RISC-V 内核，我们主要推广 NMSIS 这样的开源软件支持包。

一般情况下，csp 类型的包是非常底层的包，这里不支持依赖 `ssp/bsp/mwp/rtos/app` 这样的类型的包。但是可以依赖 sdk 类型的包，表示该包属于依赖的 sdk 包的环境中。

ssp SoC Support Package 依赖

ssp 类型的包是 SoC 或者芯片的支持的软件包，例如 `gd32vf103` , `demosoc` 这样 SoC 的支持软件包。

ssp 软件包仅可以依赖 `csp/mwp/osp` 这样的软件包，如果依赖了这三种类型的软件包，则表示在工程创建的时候或者是代码引入的时候，这三类软件包需要导入代码。而如果依赖 **sdk** 类型的软件包，则表示该 **ssp** 类型的包属于依赖的 **sdk** 类型的软件包的环境。

理论上用户可以创建一个 ssp 软件包，不依赖任何 `csp/mwp/rtos` 的软件包，也不属于 sdk 类型的软件包。**osp** 类型软件包仅可以依赖一个。

bsp Board Support Package 依赖

bsp 类型的包是针对基于某款 SoC/芯片做的开发板而推出的软件支持包，例如 `gd32vf103-rvstar` 这款开发板的 bsp 软件包。

bsp 软件包仅可以依赖 `ssp/csp/mwp/osp` 这样的软件包, 如果依赖了这几种类型的软件包，则表示在工程创建的时候或者是代码引入的时候，这类软件包需要导入代码。而如果依赖 **sdk** 类型的软件包，则表示该 **ssp** 类型的包属于依赖的 **sdk** 类型的软件包的环境。

理论上用户可以创建一个 bsp 软件包，不依赖任何软件包。**osp** 类型软件包仅可以依赖一个。

osp OS Support Package 依赖

osp 类型的包是指特定的 RTOS 的软件支持包，例如 freertos，ucosii 之类的。

osp 类型的包仅可以依赖 ssp/csp/mwp 类型的软件包，如果依赖了这几种类型的软件包，则表示在工程创建的时候或者是代码引入的时候，这类软件包需要导入代码。而如果依赖 **sdk** 类型的软件包，则表示该 **ssp** 类型的包属于依赖的 **sdk** 类型的软件包的环境。

mwp Middleware Support Package 依赖

mwp 类型的软件包是指中间件类型的软件包，例如某个语音算法的库，某种物联网连接库如 mqtt，coap 之类。

mwp 类型的包仅可以依赖 bsp/ssp/csp/mwp/osp 类型的软件包，但是不建议直接依赖 bsp/ssp，在创建 middleware 的时候尽量保证其通用性，可以很好被集成到其他的软件中。

sdk Software Development Kit Package 依赖

sdk 类型的软件包是一类非常特殊的软件包，本身并不会有额外的代码引入，而是通过依赖其他类型的软件包而组织的一个特殊的包。如果是 sdk 类型的软件包，导入是会强制检查软件包目录下所有的 npk.yml 文件以查找其他的软件包并引入该 SDK 依赖中，无需 npk.yml 文件显示进行依赖，这种依赖关系并不会直接导致创建工程时的代码的导入，更多的是软件包的集合。

一个 sdk 类型的软件包可能会依赖多个 ssp，多个 bsp，多个 csp，多个 app，多个 mwp 和多个 osp。更详细的内容请参见 [构建 SDK 开发包](# 构建 SDK 开发包)

对于属于 sdk 类型的软件包的其他软件包里面的依赖，优先使用都属于统一 sdk 软件包内部的软件包。例如：

- sdk-nuclei-sdk 是一个 sdk 类型的软件包，内部包含了 csp-nsdk_nmsis, bsp-nsdk_nuclei_fpga_eval, ssp-nsdk_demosoc, ssp-nsdk_gd32vf103, osp-nsdk_freertos, osp-nsdk_ucosii 这些软件包
- 而外部也有 csp-nsdk_nmsis，osp-nsdk_freertos 这类的软件包，在工程创建阶段，在版本匹配满足的情况下，优先使用内部的 csp-nsdk_nmsis 和 osp-nsdk_freertos，只有在版本匹配不满足要求的情况下，才会使用
- 在工程创建完成后，用户可以手动升级特定的包到其他版本。

2.4.2 模块说明

从上述描述文件中可以看出，一个标准的 npk.yml 实际是上由几个大块组成的，而在实现应用中，我们并不一定会完全用到，一个合规的 npk.yml 文件，只要拥有基本的信息，就是可以正常给 Nuclei Studio 使用。

Package Base Information

这一块分信息，是 NPK 的基础的信息，很多关键的信息在这部分内容中需要描述清楚。其中着重说明几个字段。

- **name**

  必填，NPK 的名称 ID，不要有空格，符合 C 语言命名规范，英文名称，是唯一名称 ID。

- **version**

  选填，如不填，默认为空，建议采用 SEMVER2.0 版本号管理，只能数字打头, 例如 1.2.3

- **type**

  必填，可选类型值有 csp, ssp, bsp, osp, app, mwp, sdk, tpp, tool

- **os**

选填，标明该 NPK 适用于什么类型的 Nuclei Studio，目前我们发行的 Nuclei Studio 有 win64 和 lin64 两个版本。OS 类型可以填 win32、win64、lin64、lin32, 但目前组件包上传页面只支持 win64 和 lin64, 该字段只存在 tool 类型 package

- **owner**

必填，组件包的拥有者，该 ID 一般为认证开发者 ID，便于后期进行权限查找匹配。如果该 NPK 仅作本地测试，可以随意。

Package Information

packinfo 这一块分信息，主要是对 NPK 做一些说明，包括一些文档等信息，最终在 Nuclei Studio 中使用该 NPK 时，这部分信息，会在 New Project 的导引中显示。

Package Dependency

dependencies 描述的是 NPK 的依赖关系，为了实现 NPK 的复用性，减少 NPK 的维护成本，我们在设计时，是允许 NPK 实现依赖关系，一个 NPK 可以依赖 0 个以上的 NPK，所以在这里 dependencies 是以组对象出现，每个依赖对象内需要明确 NPK 的 **name owner version**

Package Configurations

Configuration 字段是个非常特殊的字段，主要用于提供一些可配置项，以满足在工程创建时的交互场景。

不同包里面的 configuration 字段的下的二级字段名称可以一样，如果使用一样的名称则具备一样的含义，如果定义了一样的名称则按照如下的规则进行覆盖。

覆盖规则为：app > mwp > osp > bsp > ssp > csp

Configuration 对象组会包含多个对象，而每个对象有固定的结构。

- **XXX**(变量名)

变量名可以随意，简合 c++ 的命名规范即可。在后面部分会以 ${XXX} 或 ${XXX.XX} 的方式引用。

- **default**

默认值，可选项。

- **type**

这个变量的类型，为了支持更丰富的 UI 体验，我们在 NPK 中定义了很多的 UI 组件类型，具体请参看后面章节。

- **global**

标明此字段是否在工程创建时显示在引导页面中。

- **tips**

对该变量的说明信息，主要用于 UI 的 tips 事件。

- **hints**

对该变量的说明信息，如值的示例等，主要用于 UI 的 hints 事件。

- **description**

此变量的 NPK 中的说明描述。

- **UI** 组件信息

支持的类型有 choice, list, checkbox, multicheckbox, text 等，具体信息参见

Source Code Management

codemanage 描述的是跟模板工程有关的内容，大多的时候，我们的 NPK 会包括很多复杂的功能，需要创建某个一个具体的工程的时候，我们又只需要一些具体的文件，同时需要配置这些文件的信息。**codemanage** 就是将这些信息描述出来，它包含以下关键字：

- **custom**

  默认为 false, 当为 true 时，这个 installdir 就表示直接安装的目录

- **srcroot**

  默认为 `.`，表示当前 `npk.yml` 所在目录，可以是相对路径, 例如 `../`，`../bsp` 等；需要注意的是，设置了这个以后，对应的 copyfiles/incdirs/libdirs 的路径的根目录均受到影响，就会使用新设置的和这个路径

- **installdir**

  希望代码安装的目录名称，仅限英文，满足 C 语言命名格式

- 针对 sdk 类型的 package, 会被安装到 `<sdk_installdir>` , 如果 installdir 未定义，默认为 SDK，如果没有任何 sdk 类型的 package 被引用，sdk_installdir 也被默认设置为 SDK
- 针对 csp 类型的 package, 会被安装到 `<sdk_installdir>/<csp_installdir>` 目录下，TBD
- 针对 ssp 类型的 package, 会被安装到 `<sdk_installdir>/SoC/<ssp_installdir>/Common` 下面
- 针对 bsp 类型的 package, 会被安装到 `<sdk_installdir>/SoC/<ssp_installdir>/Board/<bsp_installdir>` 下面，如果不依赖于任何 ssp 类型，则安装到 `<sdk_installdir>/BSP/<bsp_installdir>`
- 针对 osp 类型的 package, 会被安装到 `<sdk_installdir>/OS/<osp_installdir>` 下面
- 针对 app 类型的 package, 会被安装到 `<app_installdir>/` 目录下，如果 installdir 未定义，默认为 application
- 针对 mwp 类型的 package, 会被安装到 `<sdk_installdir>/Components/<mwp_installdir>` 目录下

Note: 2023.05.26 新增 `copyfiles/incdirs/libdirs` 均支持 `../../` 这样的相对上级目录，但是安装或者设置路径的时候，均设置到 `<installdir>` 下面

例如: `path: ["../common/"]` 就拷贝上一级目录的 `common` , 并放在 `<installdir>/common` 下面，

如果下面有 `common` 这个目录，则创建 `R1L_common` , 如果是 `../../common` , 则创建 `R2L_common` ，

这种方案不考虑了，直接创建同名目录，同名文件直接覆盖，建议采用 `srcroot: ..` 来解决问题对应的 `incdirs/libdirs`

如果遇到这种相对路径，也需要以最终安装到路径以及文件名为准

- **copyfiles**

  待拷贝的文件或者文件夹，这里是指所有的目录或者文件，支持 `../` 、`*` 、`*.*` ，给结合 srcroot 一起使用，

- **incdirs**

  必填，加入头文件目录列表，Nuclei Studio 会进行补全，最终的路径是相对于工程根目录的路径。

- **libdirs**

  可选，lib 库所在目录，Nuclei Studio 会进行补全，最终的路径是相对于工程根目录的路径。

- **ldlibs**

可选的需要链接的库，Nuclei Studio 会进行补全，最终的路径是相对于工程根目录的路径。

Set Configuration for other packages

setconfig 用来设置 NPK 的其他选项，遵循覆盖规则 app > mwp > osp > bsp > ssp > csp。

setconfig 是一个对象组，可以无限扩展，每个对象中有三个字段来描述一个对象。

- **config**

变量名，遵循 C++ 命名规范，一般变量 XXX, 在其它部分以 ${XXX} 的方式引用

该变量名不唯一，可以通过条件进行判定生效，也遵循覆盖规则 app > mwp > osp > bsp > ssp > csp，进行自动覆盖。

- **value**

变量的值

- **condition**

变量的条件，只有条件生效时，该变量的该值才会生效

Build Configuration

设置工程的编译工具和编译选项的配置，它的关键字包含以下几个固定字段。

- **type**

支持的编译工具的类型，值一般为 gcc、clang、common。目前只支持 gcc、clang 两种，因为编译选项的配置有一些是相同的，为了提高代码的复用性，我们又添加 common 类型。

- **description**

对此编译工具的说明。

- **toolchain_name**

重要字段，编译工具名字。

- **cross_prefix**

重要字段，编译工具的前缀。

- **unflags**

在 buildconfig section 中的 `common_flags/cflags/asmflags/ldflags/cxxflags` 中生效，用于删掉之前已经定义的 flags。

- **undefines**

在 buildconfig section 中的 `common_defines/cdefines/asmdefines/cxxdefines` 中生效，用于删掉之前已经定义的 defines。（字符串完全匹配，则生效）

- **common_flags**

用的编译选项，将会添加到 cflags, asmflags, cxxflags 上，留空表示没有任何选项。

- **ldflags**

链接选项列表，留空表示没有任何选项。

- **linkscript**

链接脚本的定义，必须在 bsp/ssp 中定义，留空表示没有任何选项。

- **cflags**

C 编译选项，留空表示没有任何选项。

- **asmflags**

ASM 编译选项，留空表示没有任何选项。

- **cxxflags**

CXX 编译选项，留空表示没有任何选项。

- **common_defines**

通用的宏定义，留空表示没有任何选项。

- **cdefines**

C 的宏定义，留空表示没有任何选项。

- **asmdefines**

ASM 的宏定义，留空表示没有任何选项。

- **cxxdefines**

CXX 的宏定义，留空表示没有任何选项。

- **prebuild_steps**
  - **command**

    编译前执行的命令，留空表示没有任何选项。

  - **description**

    编译前执行的命令的说明，留空表示没有任何选项。

- **postbuild_steps**
  - **command**

    编译后执行的命令，留空表示没有任何选项。

  - **description**

    编译后执行的命令的说明，留空表示没有任何选项。

Debug Configuration

设置工程的 Debug 类型及相关参数的配置，它的关键字包含以下几个固定字段，可以不用配，如果配置了，在工程生成的时候，Nuclei Studio 会根据这里面的内容，生成了个 launch 文件，同时可以根据相关内容进行工程的 Debug。

- **type**

Debug 类型, 目前支持 GDB Custom 、GDB SEGGER J-Link、GDB OpenOCD、GDB Nuclei QEMU、Nuclei RVProf

- **description**

对支持的 Custom Jlink OpenOCD Qemu RVProf 的说明

- **configs**

对应的 Debug 类型的参数，所有的参数，都是以 key-value 的方式出现，因为每中 Debug 类型所需参数不同，对应的情况也不同，更详细的说明如下。

```
debugconfig:
  - type: openocd
    description: Nuclei OpenOCD
    configs:
      - key: XXXX
        value: xxxx
```

GDB Custom 的 Debug 参数

Table 2.1: Arguments of GDB Custom Debug

Name	Reset Value	Description
doStartGdbCLient	true	Start locally
doStartGdbServer	true	Start GDB session
gdbClientOtherCommands		gdb Client Other Commands
gdbClientOtherOptions		gdb Client Other Options
gdbMode	Commands	支持的类型，目前支持 Commands、Generi
gdbServerConnectionAddress		gdb Server Connection Address
gdbServerExecutable		gdb Server Executable
serverCheckFlag	Started by GNU MCU Eclipse	server Check Flag
gdbServerGdbPortNumber	3333	gdb Server Gdb Port Number
gdbServerOther		Config options
DEBUG_NAME	${cross_prefix}gdb${cross_suffix}	Executable path
ipAddress	localhost	Host name or lP address
portNumber	3333	
UPDATE_THREADLIST_ON_SUSPEND	false	Force thread list update on suspend
otherInitCommands		Initialization Commands
loadImage	true	Load executable
imageFileName		use File For lmage name
imageOffset		Executable offset (hex):
useFileForImage	false	Use file for Image
useProjBinaryForImage	true	
loadSymbols	true	Load symbols
symbolsFileName		
symbolsOffset		Symbols offset (hex)
useProjBinaryForSymbols	true	Use project binany
useFileForSymbols	false	Use file for Symbols
doDebugInRam	true	Debug in RAM
otherRunCommands		Run/Restart Commands
setPcRegister	false	Set program counter at (hex)
pcRegister		
setResume	false	
setStopAt	true	Set breakpoint at
stopAt	main	
doContinue	true	Continue
svdPath		svd file path

GDB SEGGER J-Link 的 Debug 参数

Table 2.2: Arguments of GDB SEGGER J-Link Debug

Name	Reset Value	Description
doStartGdbServer	true	Start the j-Link GDB server locally
doConnectToRunning	false	Connect to running target
gdbServerExecutable		Executable path
doGdbServerAllocateConsole	true	Allocate console for the GDB server
doGdbServerInitRegs	true	do Gdb Server Init Regs
doGdbServerLocalOnly	true	do Gdb Server Local Only
doGdbServerSilent	false	do Gdb Server Silent
doGdbServerVerifyDownload	true	do Gdb Server Verify Download
doStartGdbServer	true	Start the j-Link GDB server locally
gdbClientOtherCommands	set mem inaccessible-by-default off	gdb Client Other Commands

continues on next page

Table 2.2 – continued from previous page

gdbServerConnection	usb	gdb Server Connection
gdbServerConnectionAddress		gdb Server Connection Address
gdbServerDebugInterface	jtag	gdb Server Debug Interface
gdbServerDeviceEndianness	little	gdb Server Device Endianness
gdbServerDeviceName		gdb Server Device Name
gdbServerLog		gdb Server Log path
gdbServerGdbPortNumber	2331	gdb Server Gdb Port Number
gdbServerSwoPortNumber	2332	gdb Server SwoPort Number
gdbServerTelnetPortNumber	2333	gdb Server Telnet PortNumber
gdbServerOther		gdb ServerO ther
DEBUG_NAME	${cross_prefix}gdb${cross_suffix}	Executable path
gdbClientOtherOptions		gdb Client Other Options
ipAddress	localhost	
portNumber	2331	
gdbServerDeviceSpeed	auto	gdb Server Device Speed
doFirstReset	false	Initial Reset and Halt
firstResetType		
firstResetSpeed	1000	
enableFlashBreakpoints	true	Enable flash breakpoints
doGdbServerAllocateSemihostingConsole	true	Allocate console for semihosting and SWO
enableSemihosting	true	Enable semihosting console routed to
enableSemihostingIoclientTelnet	true	Telnet
enableSemihostingIoclientGdbClient	false	GDB client
enableSwo	true	Enable SWO
swoEnableTargetCpuFreq	0	SWO Cpu freq
swoEnableTargetSwoFreq	0	SWO freq
swoEnableTargetPortMask	0x1	SWO Port mask
otherInitCommands		other Init Commands
jtagDevice	GNU MCU J-Link	
loadImage	true	Load executable
imageFileName		
imageOffset		
useFileForImage	false	
useProjBinaryForImage	true	
loadSymbols	true	Load symbols
symbolsFileName		
symbolsOffset		
useFileForSymbols	false	
useProjBinaryForSymbols	true	
doDebugInRam	true	Debug in RAM
doSecondReset	true	Pre-run/Restart reset
secondResetType		Type (always executed at Restart)
otherRunCommands		
setPcRegister	false	Set program counter
pcRegister		Set program counter at (hex)
setStopAt	true	Set breakpoint
stopAt	main	Set breakpoint at
doContinue	true	Continue
svdPath		svd file path

GDB OpenOCD 的 Debug 参数

Table 2.3: Arguments of GDB OpenOCD Debug

Name	Reset Value	Description
doStartGdbServer	true	Start OpenocD locally
gdbServerExecutable		Executable path:
gdbServerGdbPortNumber	3333	GDB port
gdbServerTelnetPortNumber	4444	Telnet port
gdbServerTclPortNumber	6666	Tcl port
gdbClientOtherOptions		gdb Client Other Options
gdbServerOther		Config options
doGdbServerAllocateConsole	true	do GdbServer Allocate Console
doGdbServerAllocateTelnetConsole	false	do Gdb Server Allocate Telnet Console
gdbServerConnectionAddress		gdb Server Connection Address
doStartGdbCLient	true	do Start Gdb CLient
DEBUG_NAME	${cross_prefix}gdb${cross_suffix}	
gdbClientOtherCommands		Commands
ipAddress	localhost	Host name or lp address
portNumber	3333	Port number
UPDATE_THREADLIST_ON_SUSPEND	false	Force thread list update on suspend
doFirstReset	false	Initial Reset and Halt
firstResetType	init	
otherInitCommands		
enableSemihosting	false	Enable semihosting console routed to
loadImage	true	Load executable
useFileForImage	false	
imageFileName		
imageOffset		
symbolsFileName		
symbolsOffset		
loadSymbols	true	Load symbols
useFileForSymbols	false	
useProjBinaryForImage	true	
useProjBinaryForSymbols	true	
useRemoteTarget	true	
doDebugInRam	true	Debug in RAM
doSecondReset	true	Pre-run/Restart reset
secondResetType	halt	Type (always executed at Restart)
otherRunCommands		
setPcRegister	false	Set program counter
pcRegister		Set program counter at (hex)
setStopAt	true	Set breakpoint
stopAt	main	Set breakpoint at
doContinue	true	Continue
svdPath		svd file path

GDB Nuclei QEMU 的参数

Table 2.4: Arguments of GDB Nuclei QEMU Debug

Name	Reset Value	Description
doStartGdbServer	true	Start OpenocD locally
gdbServerExecutable		Executable path:
gdbMachineBit		Machine Bit:
gdbServerBoardName		Board name;
gdbCoreName		Nuclei RisC-V Core:
gdbServerSMPCount	1	Nuclei SMP Count:
gdbDownloadName		Download:
gdbServerOther	-serial stdio -nodefaults -S	More options:
otherExtensions		other Extensions
gdbServerGdbPortNumber	1234	GDB port:
isGdbServerVerbose	false	Extra verbose
enableSemihosting	true	Enable Arm semihosting
disableGraphics	true	Do not open graphic windows
doGdbServerAllocateConsole	true	Allocate console for QEMU
DEBUG_NAME	${cross_prefix}gdb${cross_suffix}	Executable name
gdbClientOtherOptions		Other options
gdbClientOtherCommands		Commands
ipAddress	localhost	Host name or lp address
portNumber	1234	Port number
doFirstReset	false	Initial Reset and Halt
otherInitCommands		
loadSymbols	true	Load symbols
symbolsFileName		
symbolsOffset		
useFileForSymbols	false	
useProjBinaryForSymbols	true	
useRemoteTarget	true	
loadImage	true	Load executable
useFileForImage	false	
imageFileName		
imageOffset		
useProjBinaryForImage	true	
doDebugInRam	false	Debug in RAM
otherRunCommands		
doSecondReset	true	Pre-run/Restart reset
setPcRegister	false	Set program counter
pcRegister		Set program counter at (hex)
setStopAt	true	Set breakpoint
stopAt	main	Set breakpoint at
doContinue	true	Continue
svdPath		svd file path

Nulcei RVProf 的参数

Table 2.5: Arguments of Nuclei RVProf

Name	Reset Value	Description
cycleModelExecutable	${cycelmodel_path}/${cycelmodel_executable}	cycleModel Executable
cycleModelExecutableTimeOut	20	cycleModelExecutable TimeOut
cycleModelExecutableProcessorCores	4	cycleModel Executable Processor Cores
cycleModelOther		cycleModel Other
docycleModelAllocateConsole	true	
docycleModelAllocateTelnetConsole	false	
RVProfExecutable	${rvprof_path}/${rvprof_executable}	RVProf Executable
RVProfExecutableTimeOut	20	RVProf Executable TimeOut
RVProfOther		RVProf Other
RVProfPortNumber	5000	RVProfPort Number
doRVProfAllocateConsole	true	
doRVProfAllocateTelnetConsole	false	

Extended variable

environment 是应用于 tool 类型的 NPK 包中的配置，当用户想要通过 NPK 来共享一个 tools 如 cycleModel，可以使用。当定义了 **environment**，Nuclei Studio 会自动产生几个全局，这个变量可以在其他的 NPK 中以 `${xxx-1.0.0-XXX}` 的方式使用。

Note:

- 每个包存在一个包路径，引用为 npk 名称-版本号，例如 `${tool-cyclemodel-1.0.0}`
- 其他变量的引用为 npk 名称-版本号-变量名，例如 `${tool-cyclemodel-1.0.0-cycelmodel_path}`,`${tool-cyclemodel-1.0.0-cycelmodel_executable}`
- 当变量的 system 值为 true 时，额外新增一个不带版本号的变量，取最高版本的该变量，例如 `${tool-cyclemodel-cycelmodel_executable}`

```
name: tool-cyclemodel
owner: nuclei
os:
version: 1.0.0
description: Nuclei Tools cyclemodel
details: Nuclei Tools cyclemodel
type: tool
keywords:
  - tool
  - cyclemodel
license: Apache-2.0
homepage:

## 扩展变量  tool-cyclemodel-1.0.0 与 tool-cyclemodel-1.0.0-proxy
environment:
  - key: cycelmodel_path
    value: bin
    description: cyclemodel location
    system: true
```

(continues on next page)

(continued from previous page)

```
  - key: cycelmodel_executable
    value: bin/n300_best_config_cymodel_latest
    description: cyclemodel location
    system: true


## 这是另一个 NPK 中的代码，演示了如何使用 tool-cyclemodel
debugconfig:
  - type: rvprof
    description: Nuclei RVProf
    configs:
      - key: ncycm_path
        value: ${tool-cyclemodel-1.0.0-cycelmodel_executable}
      - key: rvprof_path
        value: ${tool-rvprof-1.0.0-rvprof_executable}
```

templatemanage

内部使用的配置，这里不做详细说明。

2.4.3 NPK 中的 UI 组件

NPK 中提供了丰富的 UI 组件，这些组件的字段里面都会有 default, description, global 这些子字段，这些字段均具备含义。

default 表示默认值，description 表示该选项的含义，global 表示这个选项是否在工程创建时显示 (true)，或者仅仅内部传参使用 (false)。

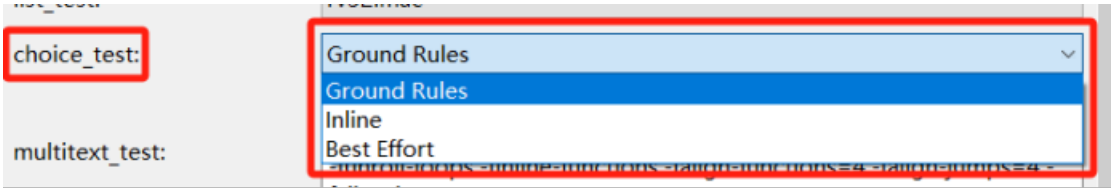
- **Choice** 单项选择框

```
choice_test:
  default_value: ground
  type: choice
  description: choice_test
  choices:
    - name: ground
      description: Ground Rules
      info:
        - name: app_commonflags
          value: >-
            -O3 -flto -fno-inline -funroll-loops -Wno-implicit -mexplicit-relocs
            -fno-builtin-printf -fno-common -falign-functions=4 -falign-jumps=4 -
↪falign-loops=4
    - name: inline
      description: Inline
      info:
        - name: app_commonflags
          value: >-
            -O3 -flto -finline -funroll-loops -Wno-implicit -mexplicit-relocs -fno-
↪builtin-printf
            -fno-common -falign-functions=4 -falign-jumps=4 -falign-loops=4 -
↪finline-functions
    - name: best
      description: Best Effort
      info:
        - name: app_commonflags
          value: >-
            -Ofast -flto -fwhole-program -finline -funroll-loops -Wno-implicit -
↪mexplicit-relocs
```

(continues on next page)

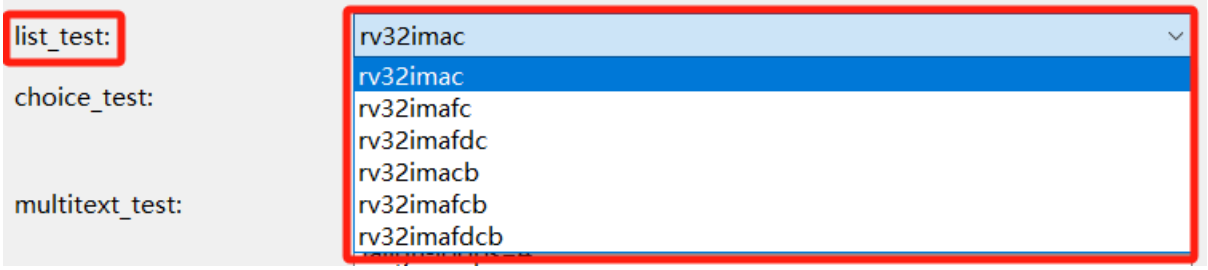
(continued from previous page)

```
            -fno-builtin-printf -fno-common -falign-functions=4 -falign-jumps=4 -
↪falign-loops=4
            -finline-functions
```



- **list** 单项选择框

```
list_test:
  default_value: rv32imac
  type: list
  global: true
  description: list_test
  value: >-
    [rv32imac,rv32imafc,rv32imafdc,rv32imacb,rv32imafcb,rv32imafdcb]
```



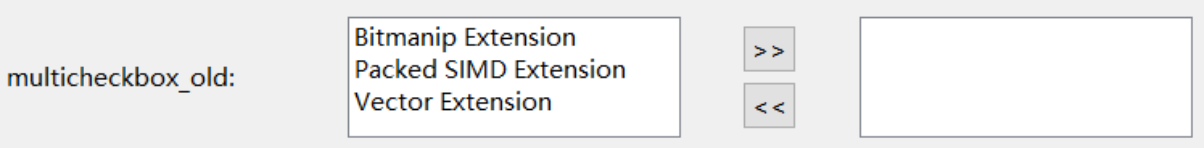
- **checkbox** 单项勾选框

```
checkbox_test:
  default_value: 0
  type: checkbox
  global: true
  description: checkbox_test
```

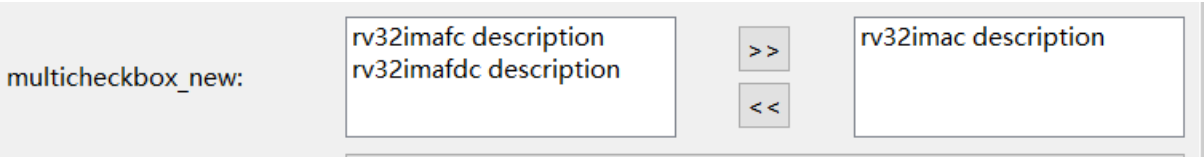
- **multicheckbox** 穿梭选择框

  下面提供 2 种写法

```
multicheckbox_old:
  default_value: []
  type: multicheckbox
  global: true
  description: multicheckbox_old
  choices:
    - name: b
      description: Bitmanip Extension
    - name: p
      description: Packed SIMD Extension
    - name: v
      description: Vector Extension
```

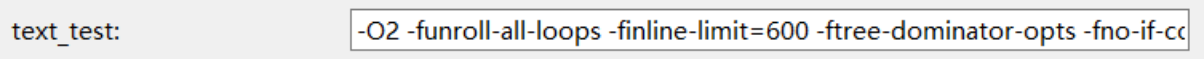



```
multicheckbox_new:
  default_value: rv32imac
  type: multicheckbox
  global: true
  description: multicheckbox_new
  param:
    name: ["rv32imac","rv32imafc","rv32imafdc"]
    description: ["${name} description","${name} description","${name} description
↪"]
```



- **text** 单行文本框

```
text_test:
  value: >-
    -O2 -funroll-all-loops -finline-limit=600 -ftree-dominator-opts
    -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting
    -fno-common -funroll-loops -finline-functions -falign-functions=4
    -falign-jumps=4 -falign-loops=4
  type: text
  description: text_test
```



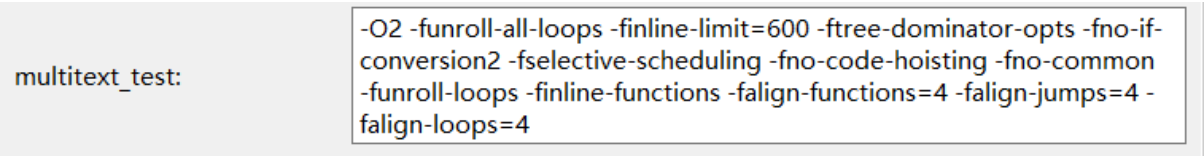
- **multitext** 多行文本框

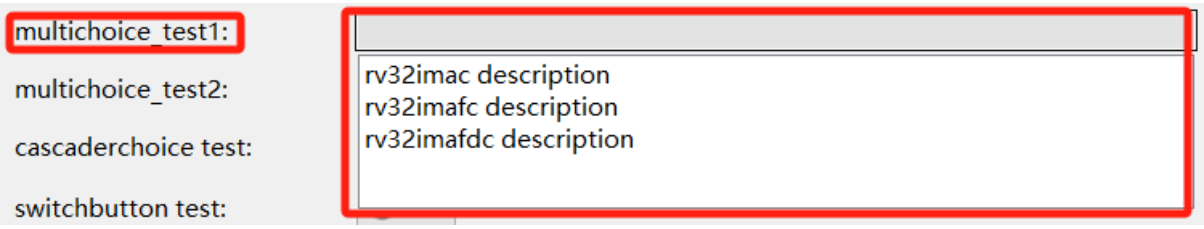
```
multitext_test:
  value: >-
    -O2 -funroll-all-loops -finline-limit=600 -ftree-dominator-opts
    -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting
    -fno-common -funroll-loops -finline-functions -falign-functions=4
    -falign-jumps=4 -falign-loops=4
  type: multitext
  description: multitext_test
```

- **multichoice** 多选下拉框

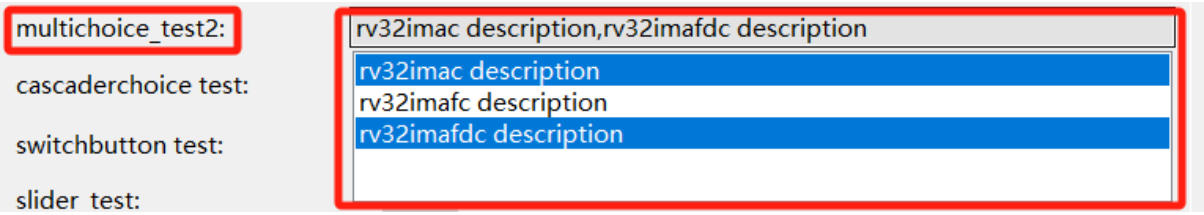
  下面提供 2 种写法

```
multichoice_test1:
  default_value: []
  type: multichoice
  global: true
  description: multichoice_test1
  param:
    name: ["rv32imac","rv32imafc","rv32imafdc"]
    description: ["${name} description","${name} description","${name} description
↪"]
```






```
multichoice_test2:
  default_value: >-
    [rv32imac,rv32imafdc]
  type: multichoice
  global: true
  description: multichoice_test2
  choices:
    - name: rv32imac      #
      description: ${name} description
    - name: rv32imafc
      description: ${name} description
    - name: rv32imafdc
      description: ${name} description
```



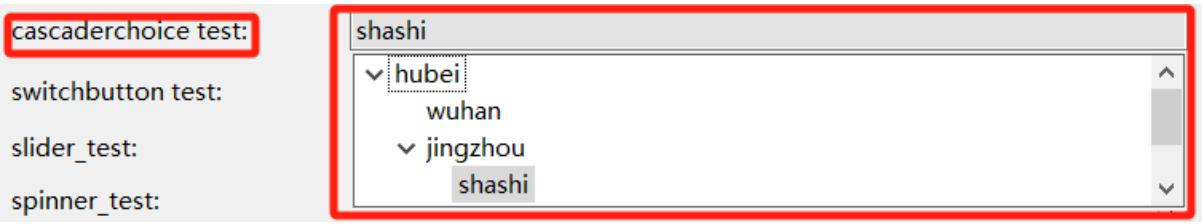
- **cascaderchoice** 级联选择框

```
cascaderchoice_test:
  default_value: >-
    [hubei,jingzhou,shashi]
  type: cascaderchoice
  global: true
  description: cascaderchoice test
  cascader_param:
    - hubei:
        - wuhan
        - jingzhou:
            - shashi
            - jianli
    - hunan:
        - changsha
        - guangdong
```

- **switchbutton** 开关

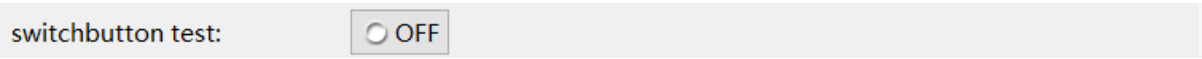
```
switchbutton_test:
  default_value: 0
  type: switchbutton
```

(continues on next page)



(continued from previous page)

```
  global: true
  description: switchbutton test
```



- **slider** 数字选择框

```
slider_test:
  default_value: 0
  type: slider
  description: slider_test
  param:
    range: >-
      [0,100,1]
```

- **spinner** 数字选择框

```
spinner_test:
  default_value: 10
  type: spinner
  description: spinner_test
  param:
    range: >-
      [-100,100,2]
```

- **multispinner** 多数字选择框

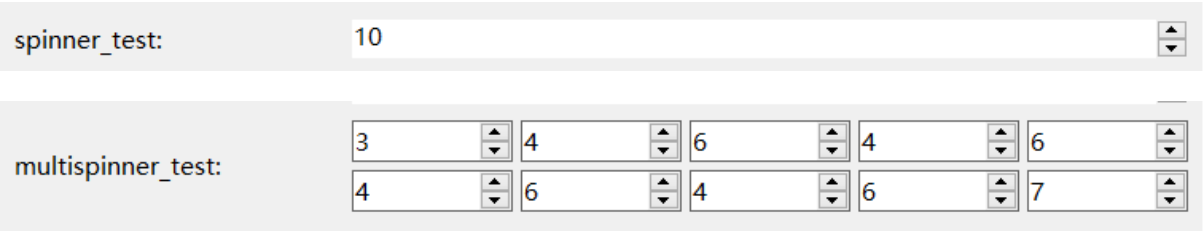
```
multispinner_test:
  default_value: >-
    [3,4,6,4,6,4,6,4,6,7]
  type: multispinner
  global: true
  description: multispinner_test
  param:
    range: >-
      [-100,100,1],[-100,100,2],[-100,100,3],[-100,100,3],[-100,100,3],[-100,100,3],
↪[-100,100,3],[-100,100,3],[-100,100,3],[-100,100,4]
```

- **multicheckbox_v2** 多项勾选框

  下面提供 2 种写法

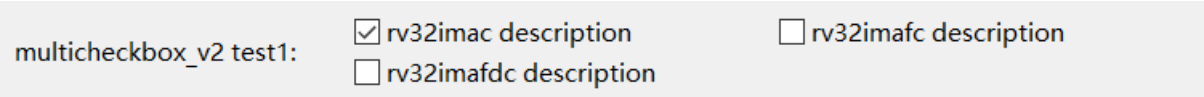
```
multicheckbox_v2_test1:
default_value: >-
 [rv32imac]
```

(continues on next page)



(continued from previous page)

```
type: multicheckbox_v2
global: true
description: multicheckbox_v2 test1
param:
  name: ["rv32imac","rv32imafc","rv32imafdc"]
  description: ["${name} description","${name} description","${name} description"]
```



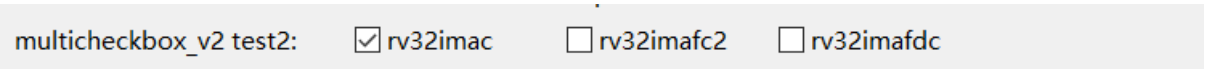

```
multicheckbox_v2_test2:
  default_value: >-
    [rv32imac]
  type: multicheckbox_v2
  global: true
  description: multicheckbox_v2 test2
  choices:
    - name: rv32imac
      description: rv32imac
    - name: rv32imafc
      description: rv32imafc2
    - name: rv32imafdc
      description: rv32imafdc
```

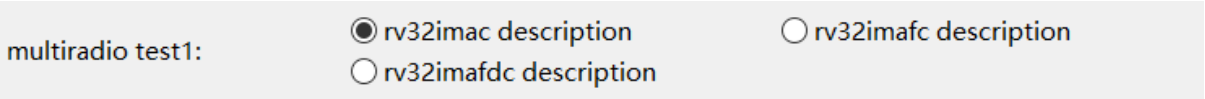
- **multiradio** 单选框

  下面提供 2 种写法

```
multiradio_test1:
  default_value: rv32imac
  type: multiradio
  global: true
  description: multiradio test1
  param:
    name: ["rv32imac","rv32imafc","rv32imafdc"]
    description: ["${name} description","${name} description","${name} description
↪"]
```

```
multiradio_test2:
  default_value: rv32imac
  type: multiradio
  global: true
  description: multiradio test2
  choices:
    - name: rv32imac
      description: rv32imac
    - name: rv32imafc
      description: rv32imafc2
    - name: rv32imafdc
      description: rv32imafdc
```





2.4.4 NPK 的语法

YAML 语言

NPK 的描述文件 npk.yml，是以 YAML 语言来编写，所以它支持标准的 YAML 语法，获取更多 YAML 相关的信息，可以参考：https://yaml.org/

变量定义

NPK 的描述语言中，允许用户自义定一个变量，并在有依赖关系的 NPK 中的任意位置使用

Note:

- 例子 NPK 中定义了一个变量 app_commonflags，在与该 NPK 有依赖关系的任意 npk.yml 文件内的任意位置，我们可以通过 `${app_commonflags}` 来使用 app_commonflags 的值。
- 例子 NPK 中定义了一个 list 对象 nuclei_core, 在与该 NPK 有依赖关系的任意 npk.yml 文件内的任意位置，我们可以通过 `${nuclei_core.arch}` 来使用 nuclei_core 对象中的 arch 值；通过 `${nuclei_core.abi}` 来使用 nuclei_core 对象中的 abi 值；通过 `${nuclei_core.tune}` 来使用 nuclei_core 对象中的 tune 值。

```yaml
configuration:
    app_commonflags:
        value:
        type: text
        description: Application Compile Flags

    nuclei_core:
        default_value: n201
        type: choice
        global: true
        description: Nuclei RISC-V Core
        choices:
          - name: n200
            arch: rv32imc
            abi: ilp32
            cmodel: medlow
            tune: nuclei-200-series
            description: N200 Core(ARCH=rv32imc, ABI=ilp32)
          - name: n201
            arch: rv32iac
            abi: ilp32
            cmodel: medlow
            tune: nuclei-200-series
            description: N201 Core(ARCH=rv32iac, ABI=ilp32)
## Set Configuration for other packages
setconfig:
```

(continues on next page)

(continued from previous page)

```
  - config: nmsislibarch
    value: ${nuclei_core.arch}
## Build Configuration
buildconfig:
  - type: gcc
    common_flags: # flags need to be combined together across all packages
      - flags: ${app_commonflags}
```

关键字

为了更好的描述 NPK，我们定义了一些字段，以描述出各种关系，其中大部分字段如其字面意义，这里重点介绍以下几个关键字。

- **condition**

condition 在 npk.yml 中，使用很频繁，是自定义的一个关键字，用来处理逻辑关系，类似 **if**，具体的使用如下。

```
ldflags:
  - flags: --specs=nosys.specs
    condition: $( ${stdclib} == "newlib_full" )
  - flags: --specs=nano.specs --specs=nosys.specs -u _printf_float -u _scanf_float
    condition: $( ${stdclib} == "newlib_fast" )
  - flags: --specs=nano.specs --specs=nosys.specs -u _printf_float
    condition: $( ${stdclib} == "newlib_small" )
  - flags: --specs=nano.specs --specs=nosys.specs
    condition: $( ${stdclib} == "newlib_nano" )
  - flags: --specs=${stdclib}.specs
    condition: $( startswith(${stdclib}, "libncrt") )

# 上述描术中 flags 的值，由 condition 决定，当在不同的场景时，flags 的值会不同，
# 又因为 flags 是一个数组类型，所以上述例子中 flags 会有多个值，最终使用是，是 flags 的值拼接成的字符串。
```

- **dependencies**

dependencies 在 npk.yml 中，用来描述 NPK 的依赖关系。

在很多的时候，NPK 需要依赖特定 owner 的某个 name，某个 version 版本的包，查找规则为 `owner/name:version`, 如果 owner 未定义，则默认为该 npk 文件中定义的 owner，如果 **version** 未定义，则优先在同组件包查找，否则取最新的包。如果所依赖的包找不到，则该 NPK 将无法使用。

Note: 例子中 NPK 依赖了三个 npk，如下：

- sdk-nuclei_sdk owner、`version` 未定义，则优先在同组件包查找，否则取最新的包
- tool-testmodel 明确了 owner 和 version
- tool-rvprof 明确了 owner 和 version

```
## Package Dependency
dependencies:
  - name: sdk-nuclei_sdk
    version:
    owner:
  - name: tool-testmodel
    version: 1.0.0
    owner: nuclei
  - name: tool-rvprof
```

(continues on next page)

(continued from previous page)

```
    version: 1.0.0
    owner: nuclei
```

自定函数

在 NPK（npk.yml）中，为了更好地满足各种不同的需求，我们特意定义了一些常用的函数。

- **upper**

  将字符串变大写

```
${linker_script} = "test"
$(upper("${linker_script}CD")) => TESTCD
```

- **lower**

  将字符串变小写

```
${linker_script} = "test"
$(lower("${linker_script}cd")) => testcd
```

- **contains**

  判断字符串中是否包含另一个字符串

```
${linker_script} = "test"
$(contains(${linker_script}, nmsis) ) => false
```

- **join**

  将字符串连接数组

```
$(join([a,b,c,v], '') => abcv
```

- **concat**

  连接字符串成为新一字符串

```
${linker_script} = "test"
$(concat(${linker_script}, v) => testv
```

- **strip**

  去掉字符串两端空格

```
${linker_script} = "   test  "
$(strip(${linker_script})) => test
```

- **startswith**

  判断字符串是否以 xxx 开头

```
${linker_script} = "testabcd"
$(startswith(${linker_script}, test)) => true
```

- **endswith**

  判断字符串是否以 xxx 结尾

```
${linker_script} = "testabcd"
$(endswith(${linker_script}, test))  => false
```

- **arithop**

数学运算符, 支持 +、- 、* 、/、% 、?(三元运算) 等常用运算符，不支持 ++、

```
$(arithop(${linker_script}+22) > 1000)
$(arithop(${linker_script}+22))
$(arithop(${linker_script}>22?1:0))
```

- **npack/npack_installdir**

  npack 是否包含指定的 npk

  npack_installdir 包含的 npk 的路径

```
# a.yml
    name: mwp-a
    owner: nuclei
    copyfiles:
        - path: ["common", "abc.ld", "src/openocd.cfg", "inc"]

# b.yml

    name: mwp-b
    owner: nuclei
    copyfiles:
        - path: ["common", "111.ld", "src", "inc"]
    debugconfig:
      - type: openocd
        description: Nuclei OpenOCD
        configs:
          - key: config
            value: "$(npack_installdir(mwp-a))/src/openocd.cfg"
            condition: $( npack(nuclei:mwp-a) )

# 这段描述，是当 b.yml 如果依赖 a.yml 时，就可以将 config 的值，设置为 a.yml 的目录下的/src/
↪openocd.cfg
```

- **list_get**

  获取数组元素中指定脚标的值

```
${nuclei_cache}=[ic,dc,ccm]
$(list_get(${nuclei_cache},0)) -> ic
```

- **list_set**

  修改数组元素中指定脚标的值

```
${nuclei_cache}=[ic,dc,ccm]
$(list_set(${nuclei_cache},1,aa)) -> [ic,aa,ccm]
```

- **list_del**

  删除数组元素中指定脚标的值

```
${nuclei_cache}=[ic,dc,ccm]
$(list_del(${nuclei_cache},1)) -> [ic,ccm]
```

- **list_add**

  在数组元素指定脚标插入值

```
${nuclei_cache}=[ic,dc,ccm]
$(list_add(${nuclei_cache},2,aa)) -> [ic,dc,aa,ccm]
```

- **list_size**

  获取数组元素 list 的长度

```
${nuclei_cache}=[ic,dc,ccm]
$(list_size(${nuclei_cache})) -> 3
```

- **list_sub**

  从 list 中指定的位置开始，截取指定长度的 list

```
${nuclei_cache}=[ic,dc,ccm]
$(list_sub(${nuclei_cache},1,2)) -> [dc]
$(list_sub(${nuclei_cache},0,2)) -> [ic,dc]
$(list_sub(${nuclei_cache},1,)) -> [dc,ccm]
$(list_sub(${nuclei_cache},,2)) -> [ic,dc]
```

- **subst**

  对字符串内部的指定字符串进行替换，第三个参数可为空

```
subst(libncrt_small,lib,) ==> ncrt_small
subst(libncrt_small,lib,ext) ==> extncrt_small
```

2.5 Nuclei Studio NPK 创建与共享

Nuclei Studio 2022.04 版中，提供了一个非常重要的功能，该功能主要通过提供的各种初始模板，方便开发者去创建自己的 NPK 组件包，并可以通过平台将自己的 NPK 组件包贡献出来，供其他开发者使用。

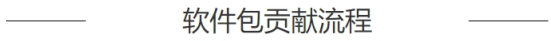
2.5.1 开发 NPK 组件包

认证开发者

在创建 NPK 组件包前，先需要认证一个开发者帐号，获取一个唯一的 owner ID, 这在 owner ID 对应的就是前文中介绍的 owner，它能方便我们对 NPK 进行管理。

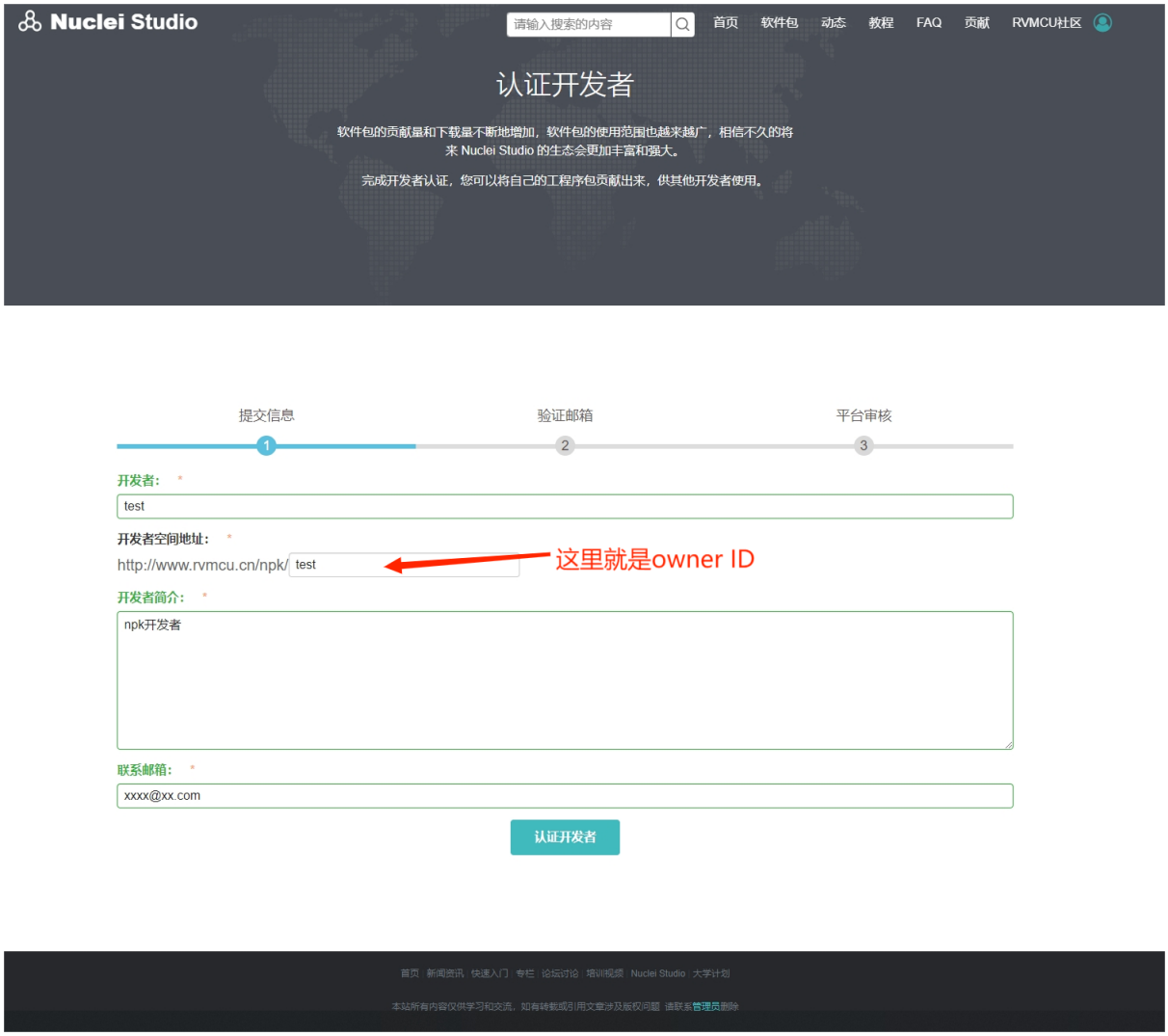
首先，登陆芯来科技的 RVMCU 社区的网站[2]，登陆成功后，依次进入 Nuclei Studio-> 贡献页面，就可以看到认证按钮。

[2] https://www.rvmcu.com/user-login.html



或者直接访问 认证地址[3]，按提示分别填写相关信息，其中 开发者对应的开发者空间地址的后辍将会是您的 owner ID。

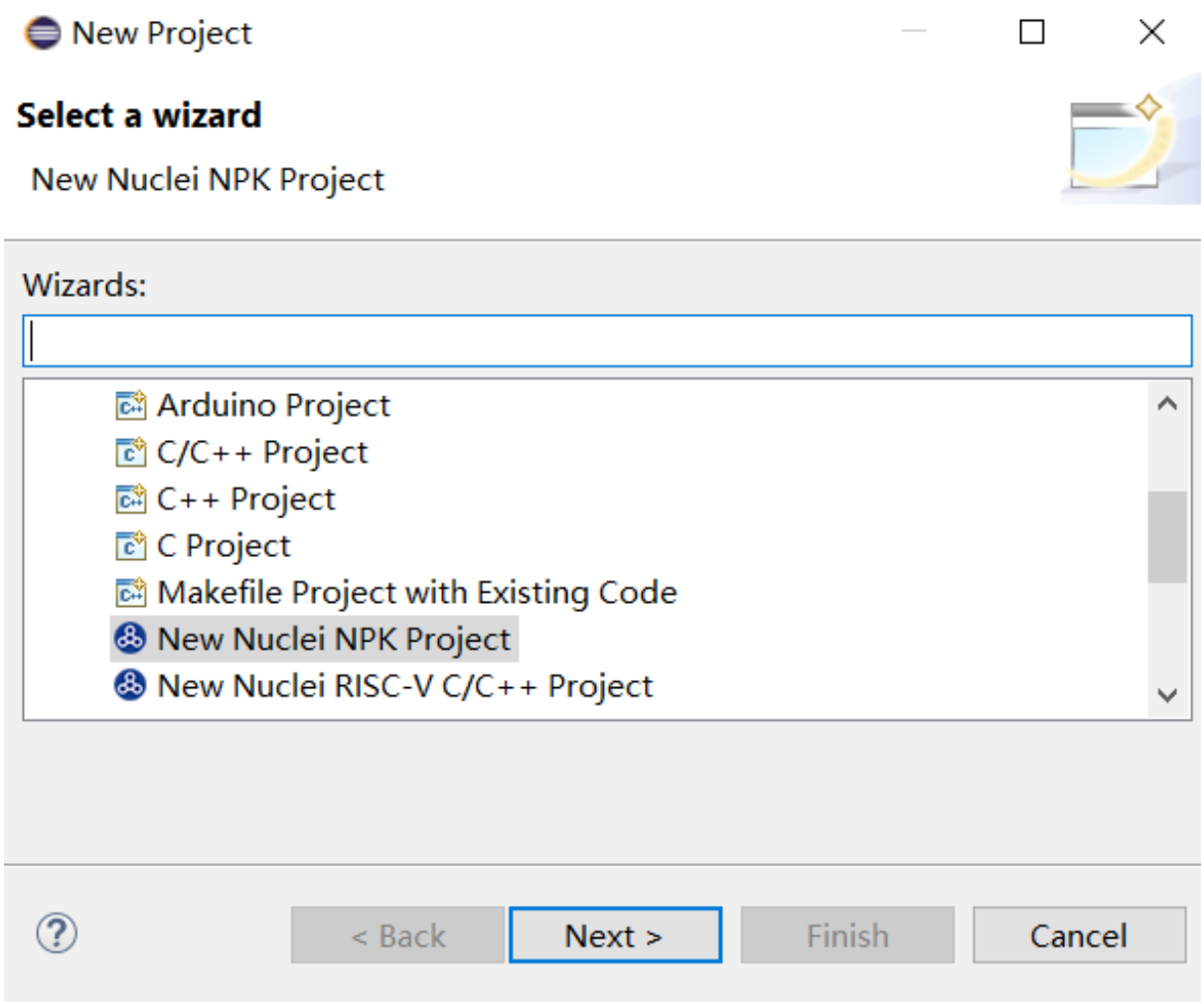
[3] https://www.rvmcu.com/nucleistudio-developer.html



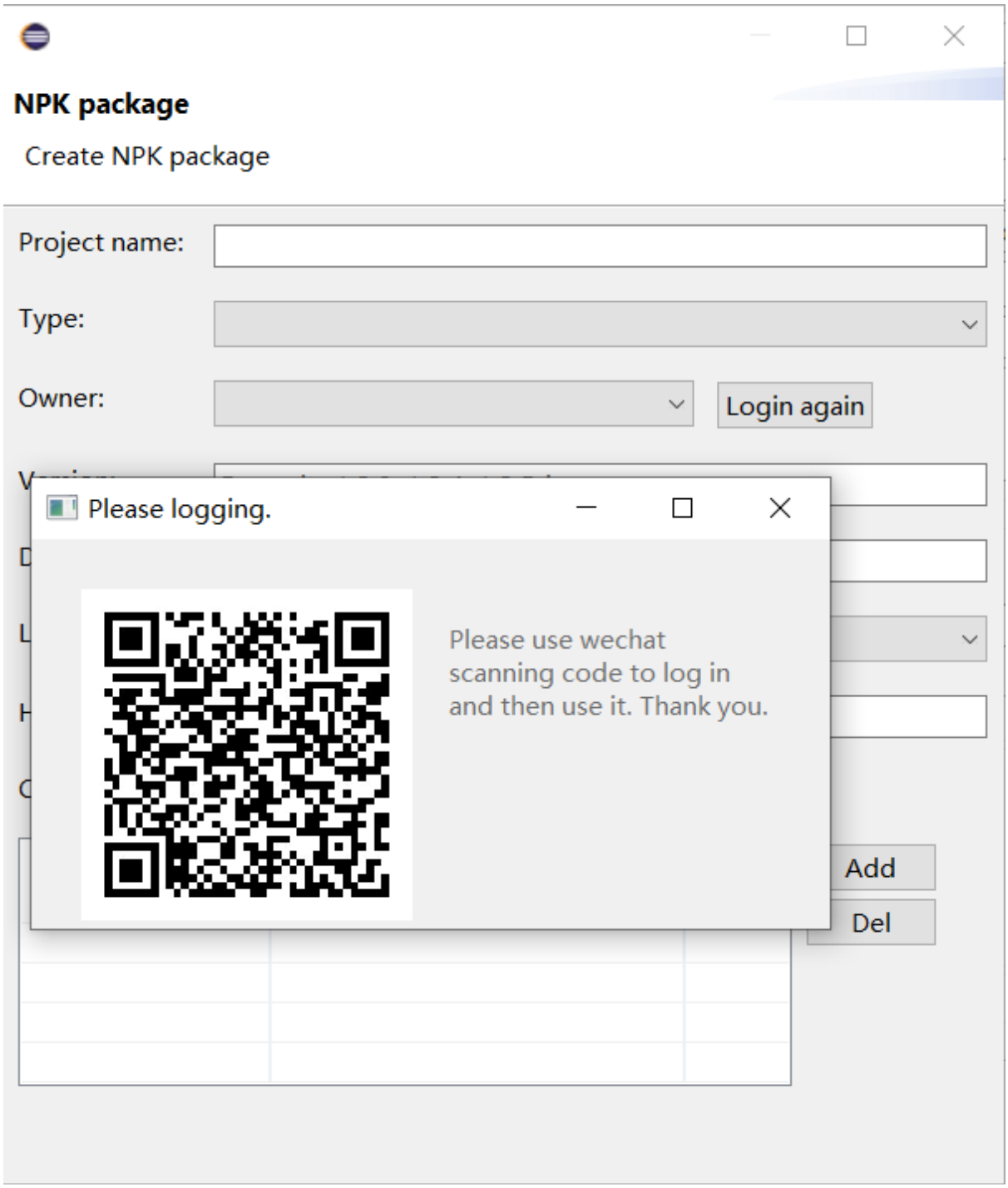
填写完相关内容后提交，系统审核通过后，即可完成认证。

创建 NPK 组件包

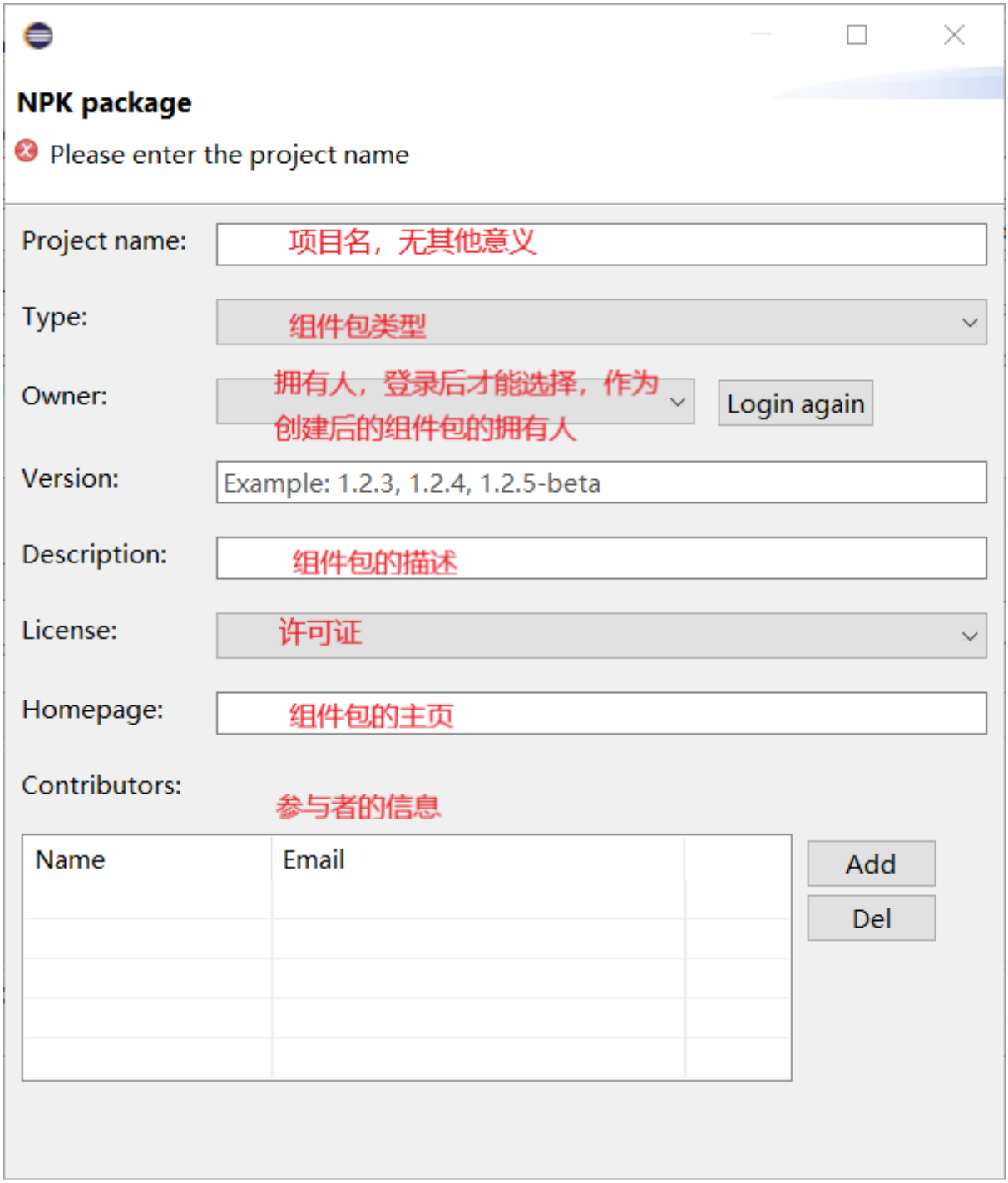
完成开发者认证后，在 Nuclei Studio 中新建一个 Nuclei Studio 组件工程，在菜单栏中，选择 `File->New->Project->New Nuclei NPK Project`。



首次使用需要登录开发者帐号，且登录后 7 天有效，超过时间后需要再次登录，登录需要使用微信扫描二维码，见下图。登录成功后，owner 可选框会列出绑定的相关开发者账号，其他的根据提示输入相关信息。需要变更账号时，可点击 login again 重新登录。



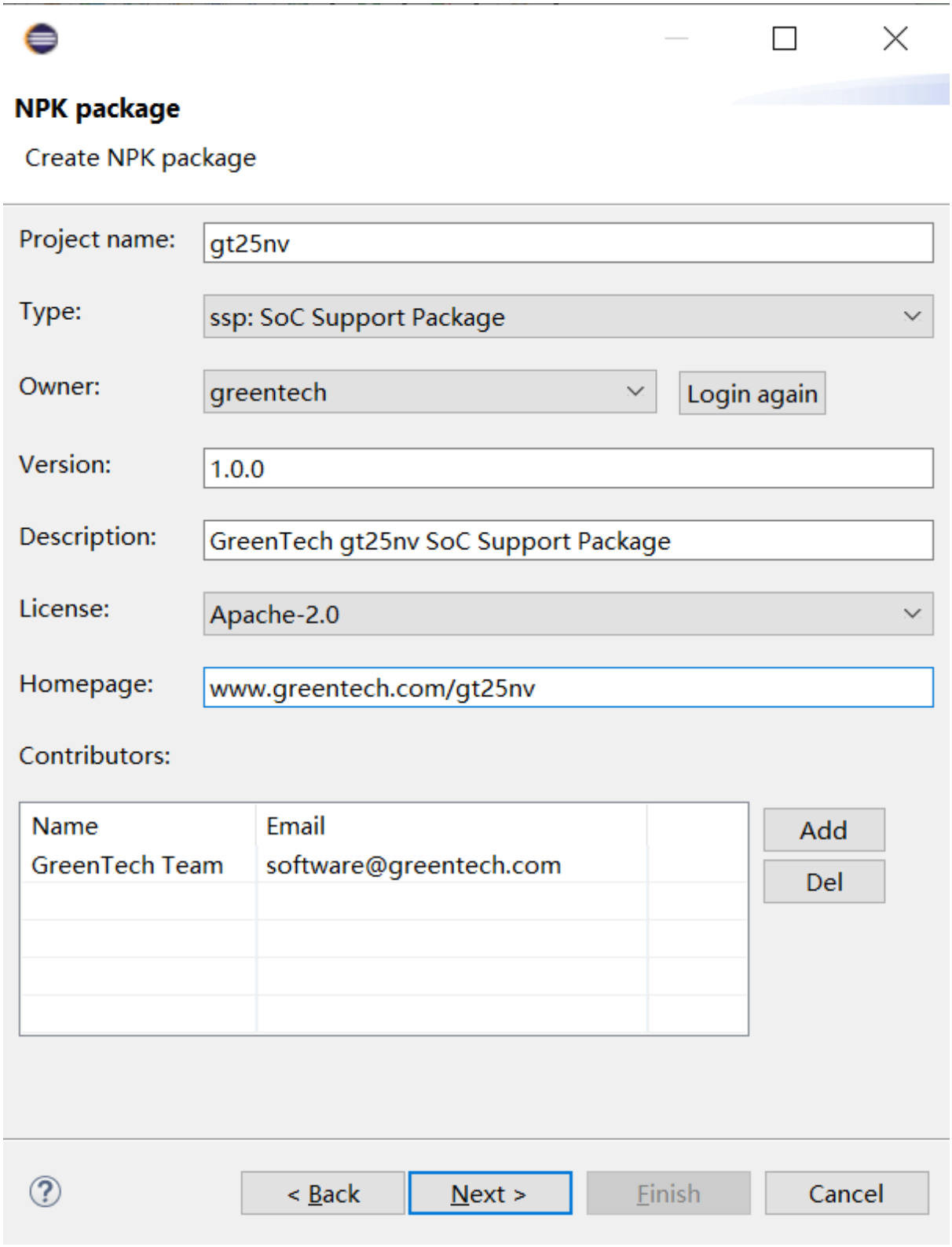
开发者帐号登陆成功后，依据工程向导依次填入工程名、工程类型等等相关信息。



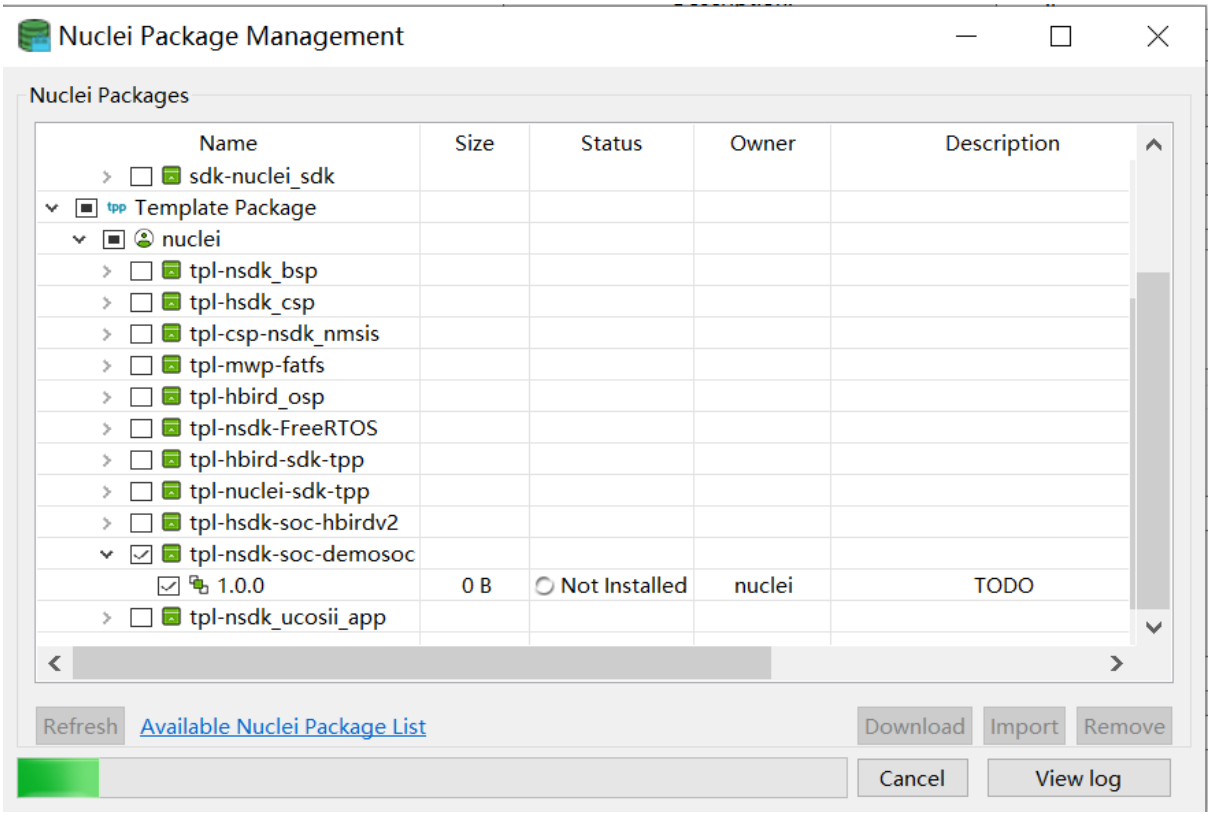
选择 Type 时，无对应模板时，会跳出对应提示，点击确定，进入 Nuclei Package Management 页面，根据需要下载 Template Package 的对应模板，或点击下图右下角 Import 自行导入。

下面以 ssp 类型模板为例，假设你的公司名称为 GreenTech，你的 SoC 名称为 gt25nv，适配的开发板为 gt25nv_devkit，采用了我们的 n307FD 处理器 (rv32imafdc) 配置，并且配置了 dsp 特性，并且提供了 ilm, flash,flashxip 三种下载方式。

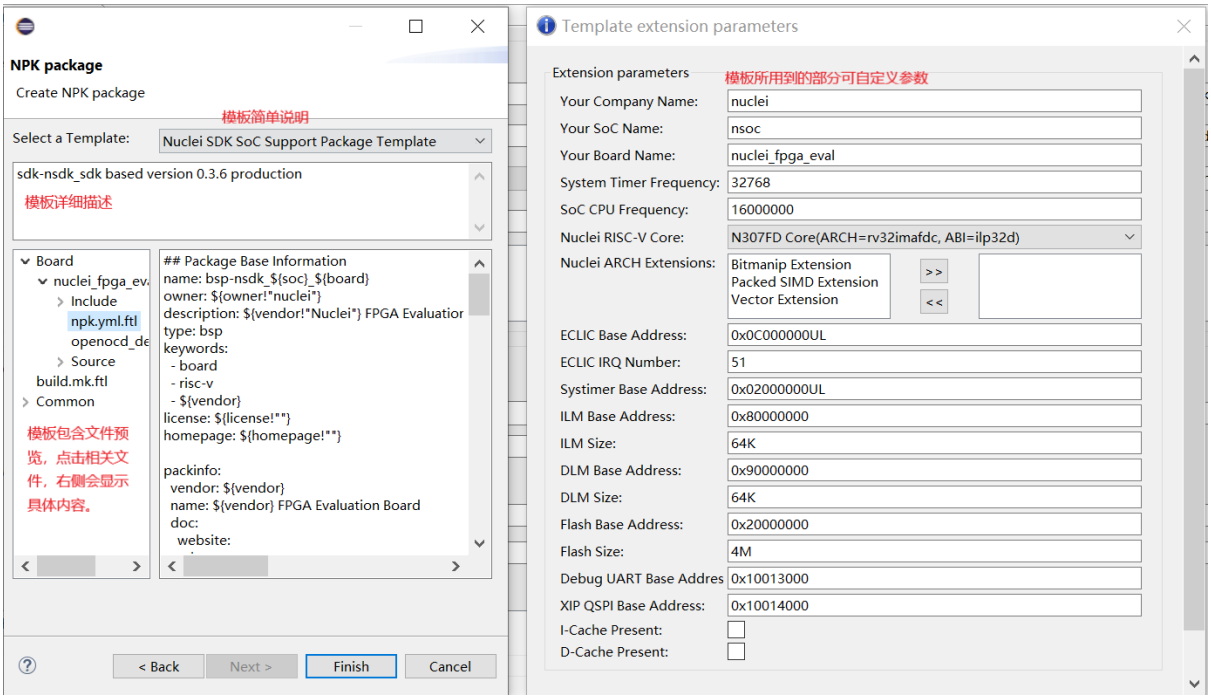
Type 选择 ssp: `Soc Support Package`



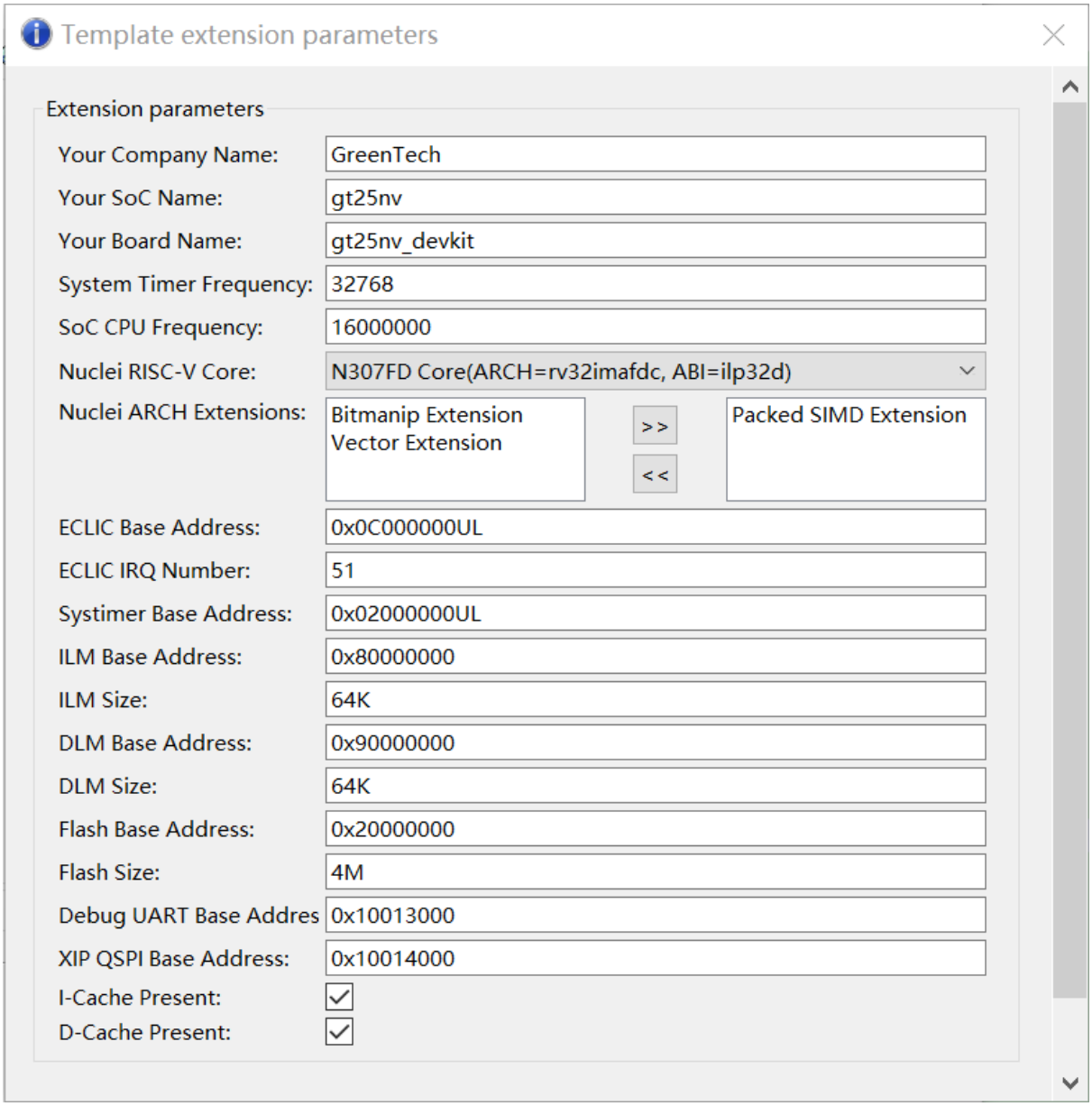
其次进入缺少对应模板，进入 Nuclei Package Management 页面，选择 `tpl-nsdk-soc-demosoc`，点击 `Download` 下载，下载完成后关闭该页面



点击 Next，在 `Select a Template` 中选择刚才下载的模板 `tpl-nsdk-soc-demosoc`，左侧为模板描述和相关的文件预览，右侧为模板中部分可自定义的内容。

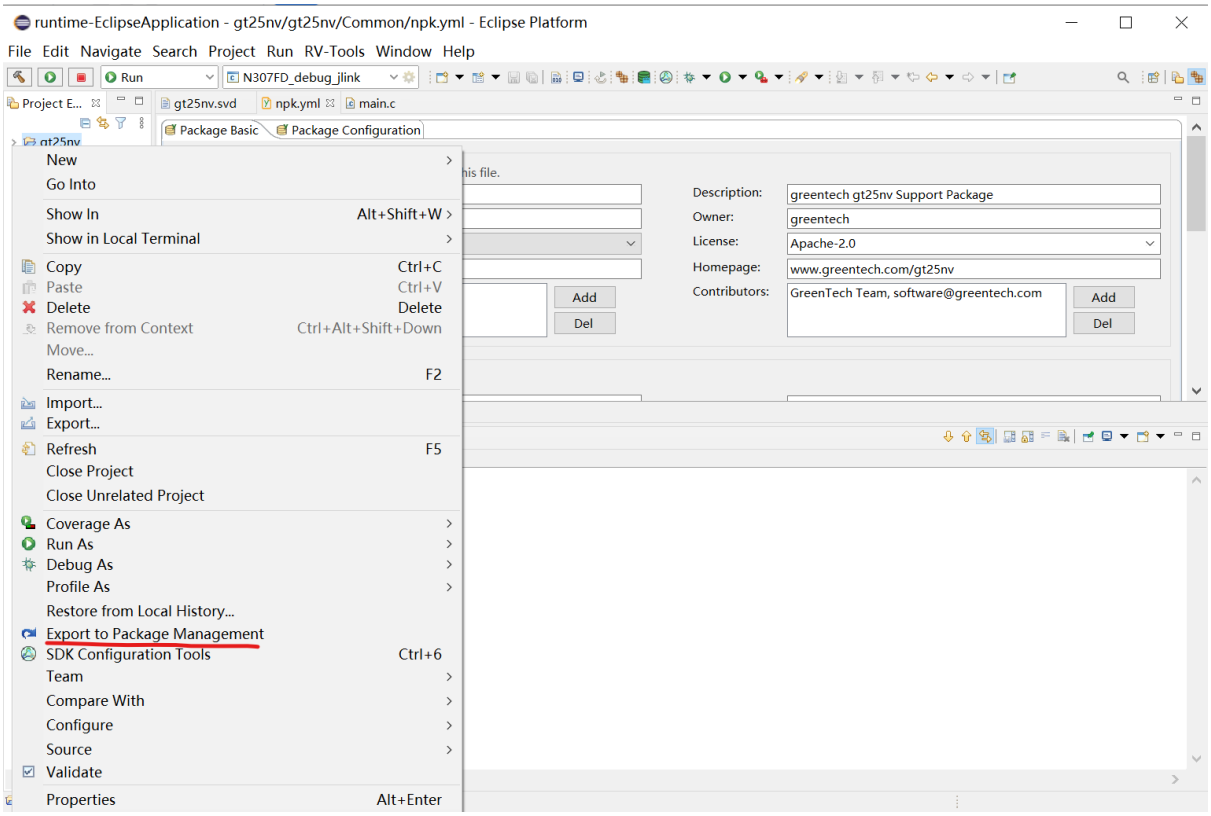


我们这里举例，公司名称为 `GreenTech`，SoC 名称为 `gt25nv`，适配的开发板为 `gt25nv_devkit`，采用了我们的 n307FD 处理器（`rv32imafdc`）配置，并且配置了 dsp 特性，并且提供了 ilm，flash，flashxip 三种下载方式。然后 Nuclei RISC-V Core 选择为 `NX600`，经过修改后如图。

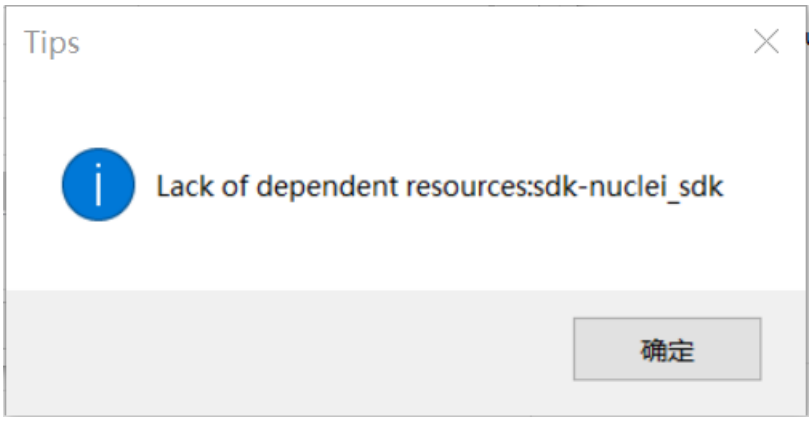


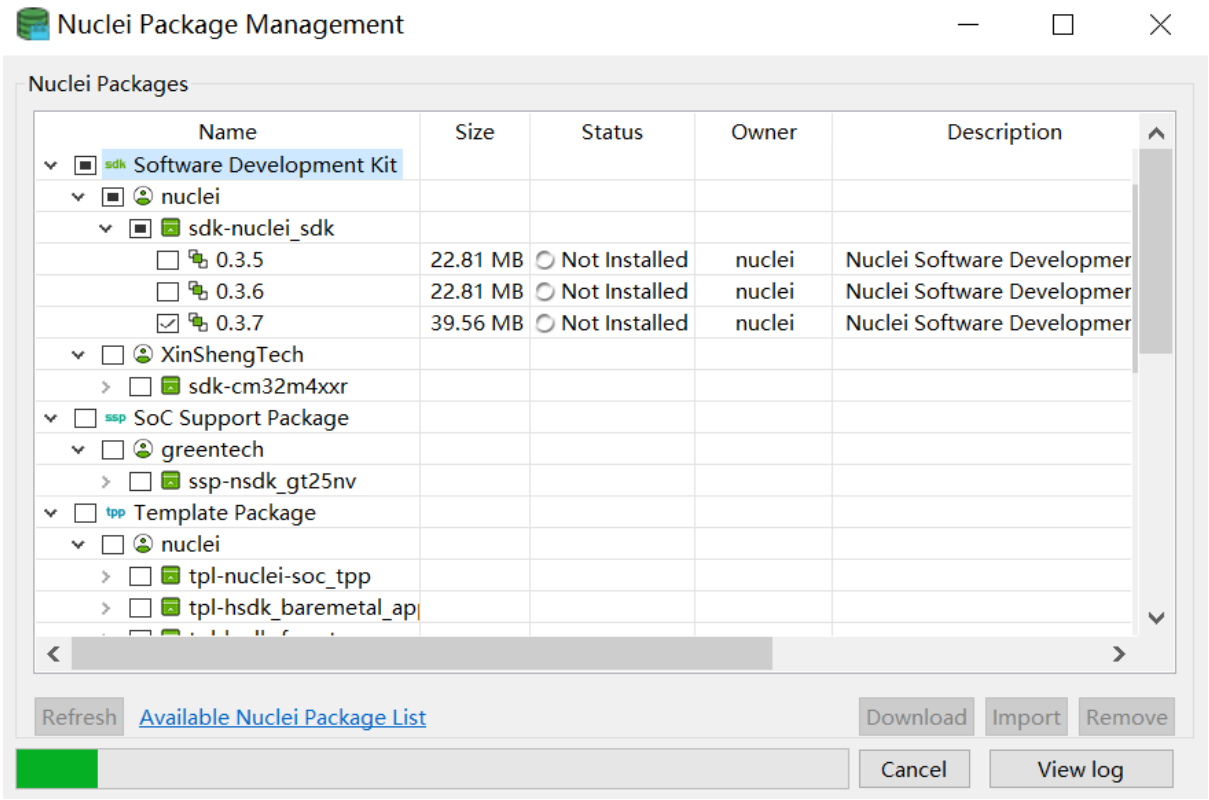
然后点击 Finish 即成功生成对应 NPK 组件包，再根据需要，可以打开目录查看并修改对应的文件和结构，修改完成后再导入该 NPK 组件包。

修改项目，根据需要自行修改。项目修改完成，导入 NPK 组件包。

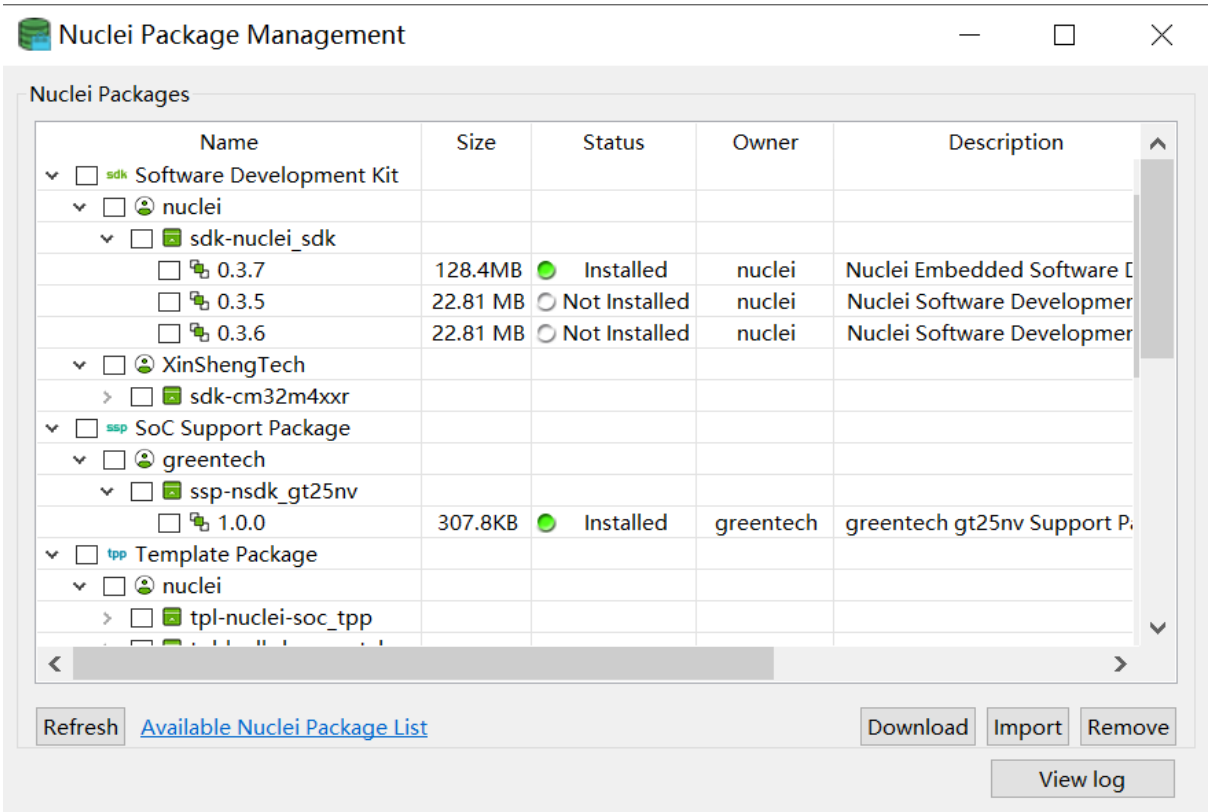


按照示例创建的 soc 在导入时，提示缺少依赖 sdk-nuclei_sdk，点击确定，进入 Nuclei Package Management 页面，根据提示，选中 sdk-nuclei_sdk，点击 Download。





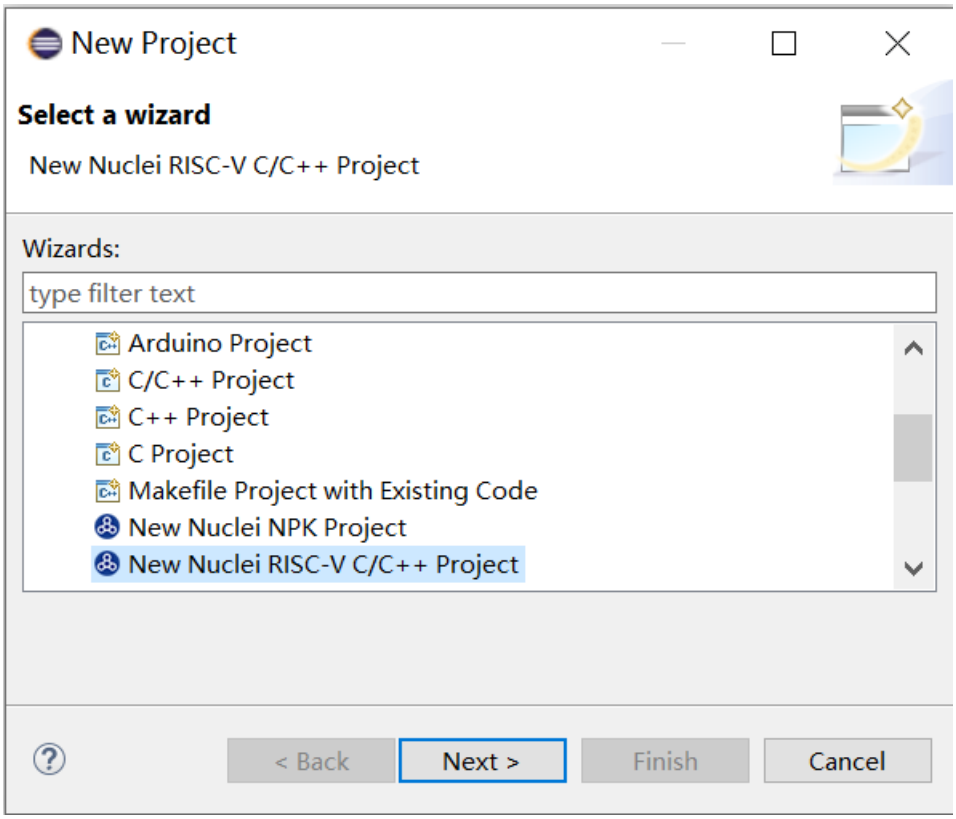
下载完成后，NPK 组件包工程导入成功后结果如下图。



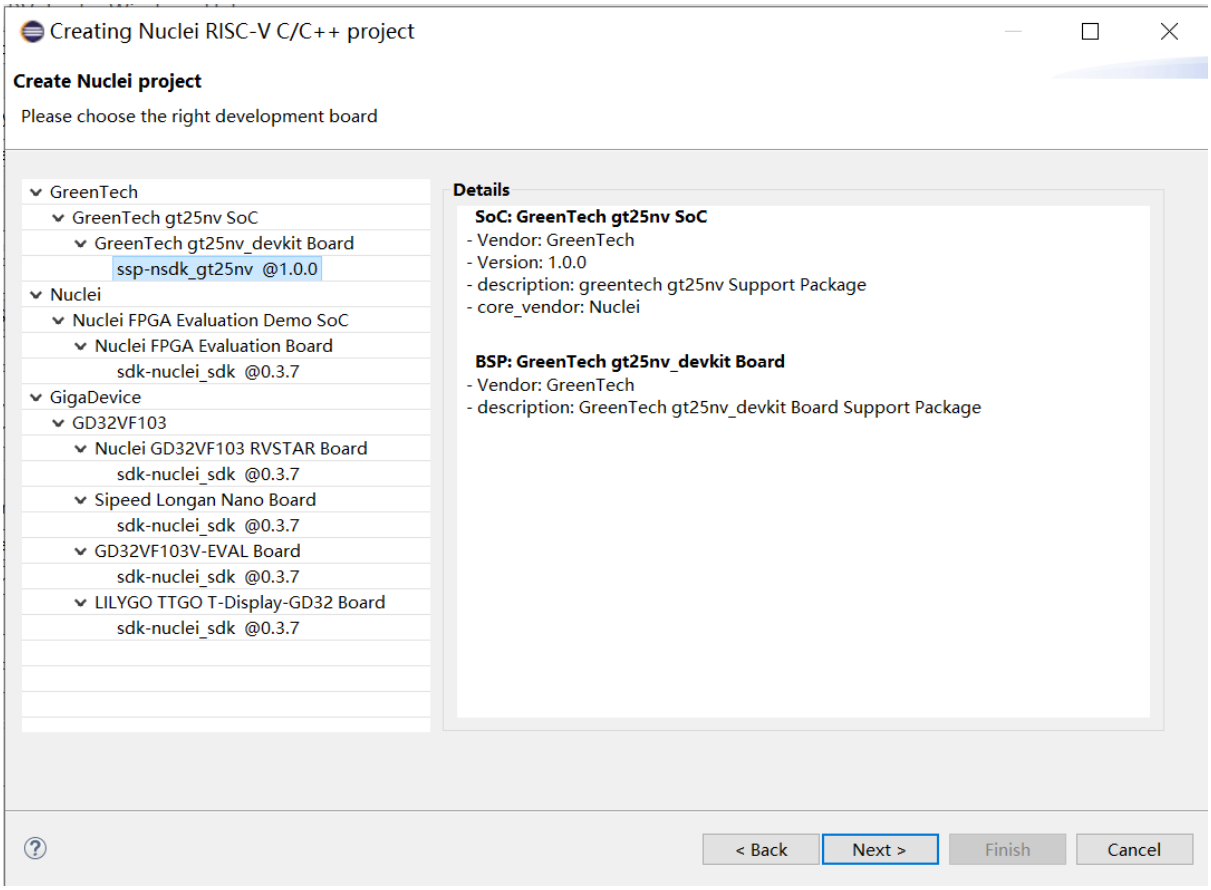
2.5.2 测试 NPK 组件包

创建测试项目

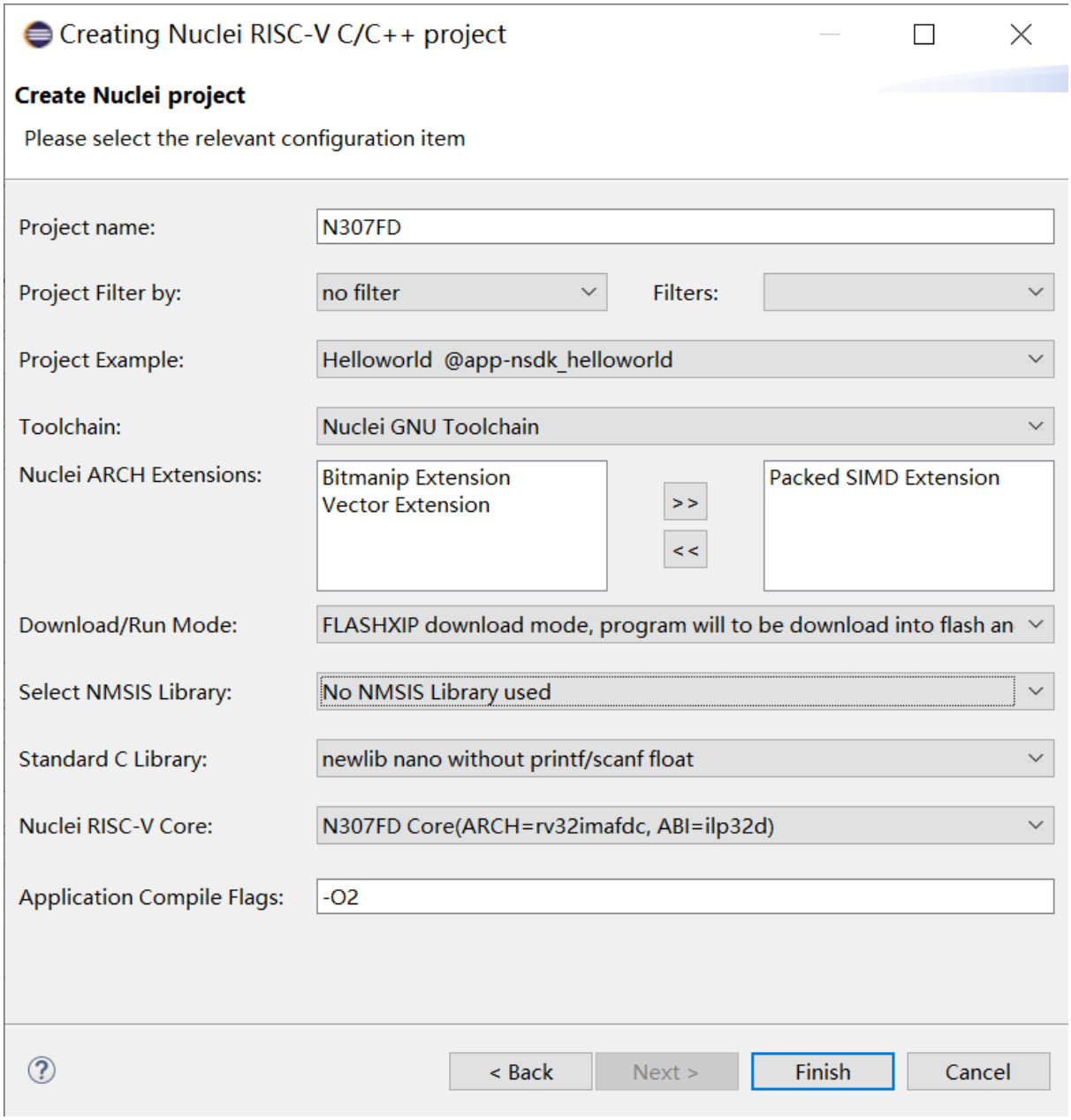
使用导入的 NPK 组件包工程创建一个工程，可以在菜单栏中，选择 File-> New-> Project-> New Nuclei RISC-V C/C++ Project 。



点击 Next，选择上面创建的 soc 内所包含的 soc 和 board



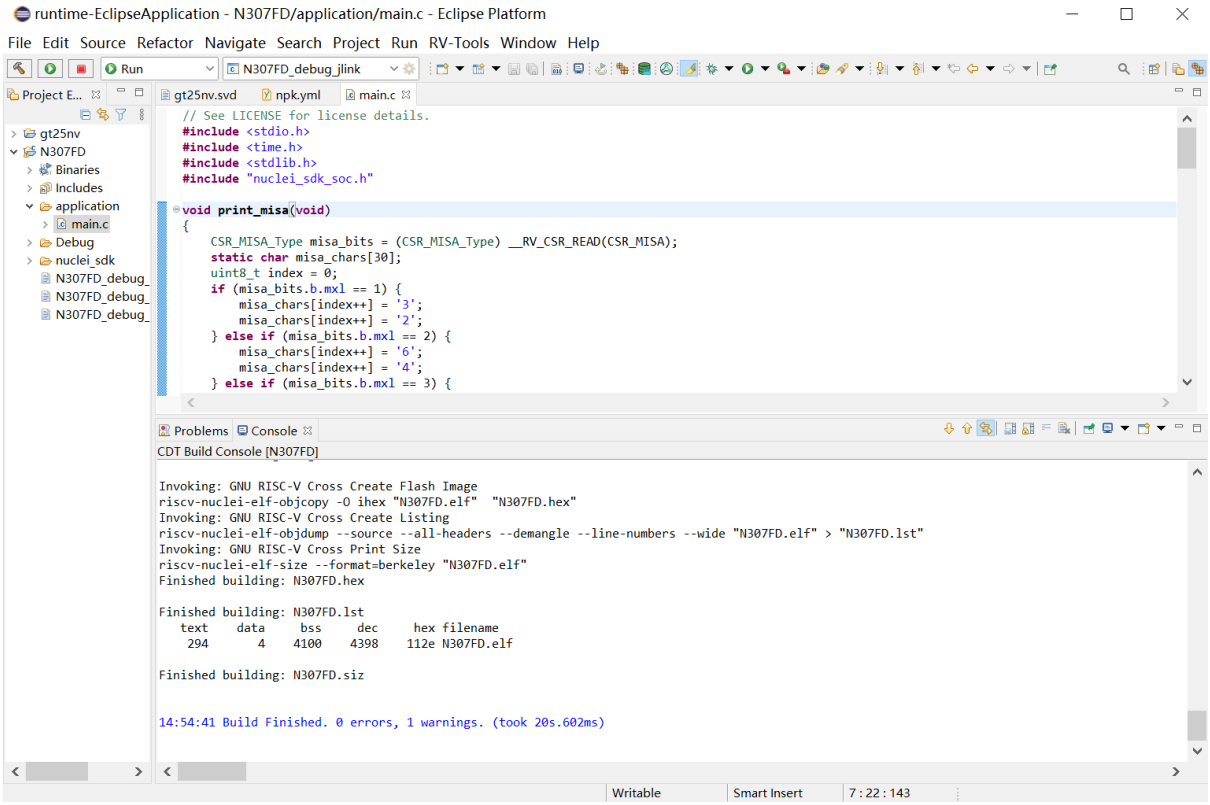
点击 Next，填入 Project Name，并选择 Project Example 为 `Helloworld`，点击 `Finish`，完成测试工程的创建。



编译调试测试工程

上文步骤中创建的一个工程，就是根据开发者的 NPK 组件包创建出来的一个测试工程，开发者可以按一个正常的工程进行对应的编码、调式、运行等操作。

鼠标点击选中上一步生成的项目 N307FD，然后编译成功，后续运行等步骤略去，至此已成功创建了一个 NPK 组件包，并使用此 NPK 组件包进行了导入使用。

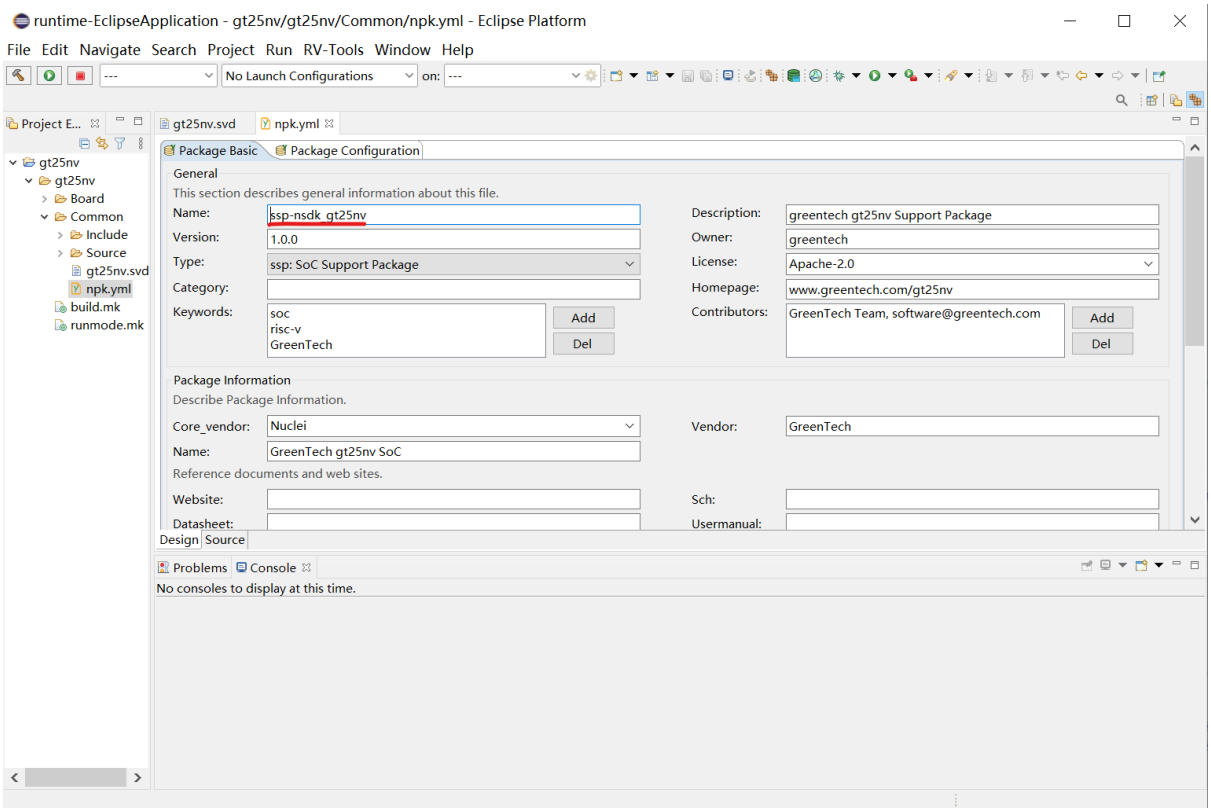


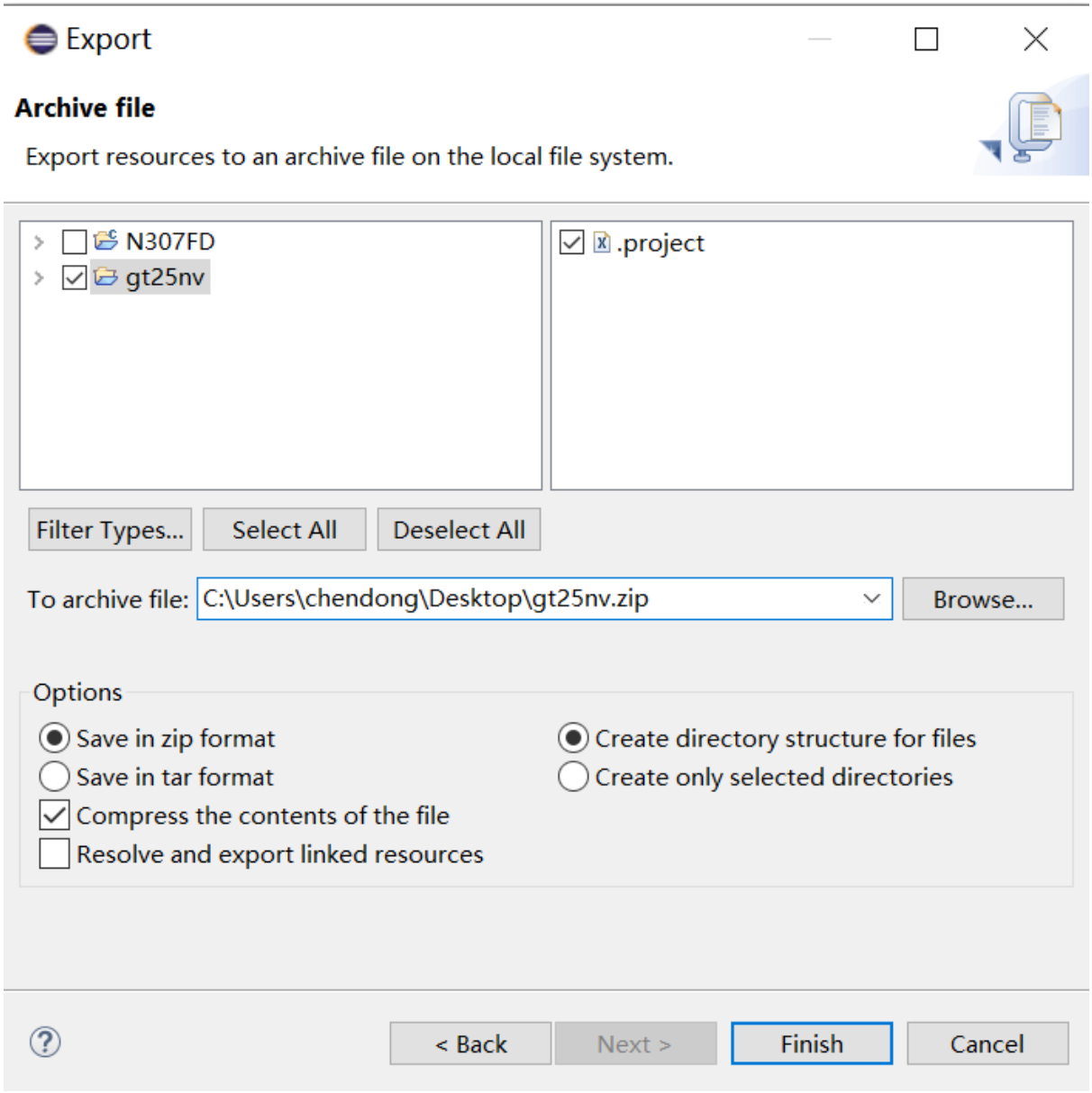
2.5.3 共享 NPK 组件包

NPK 组件包共享

经过测试通过后，可以将您创建的 NPK 组件包分享出去，首先需要将您的 NPK 组件包工程导出为一个 zip 包，具体操作如下。

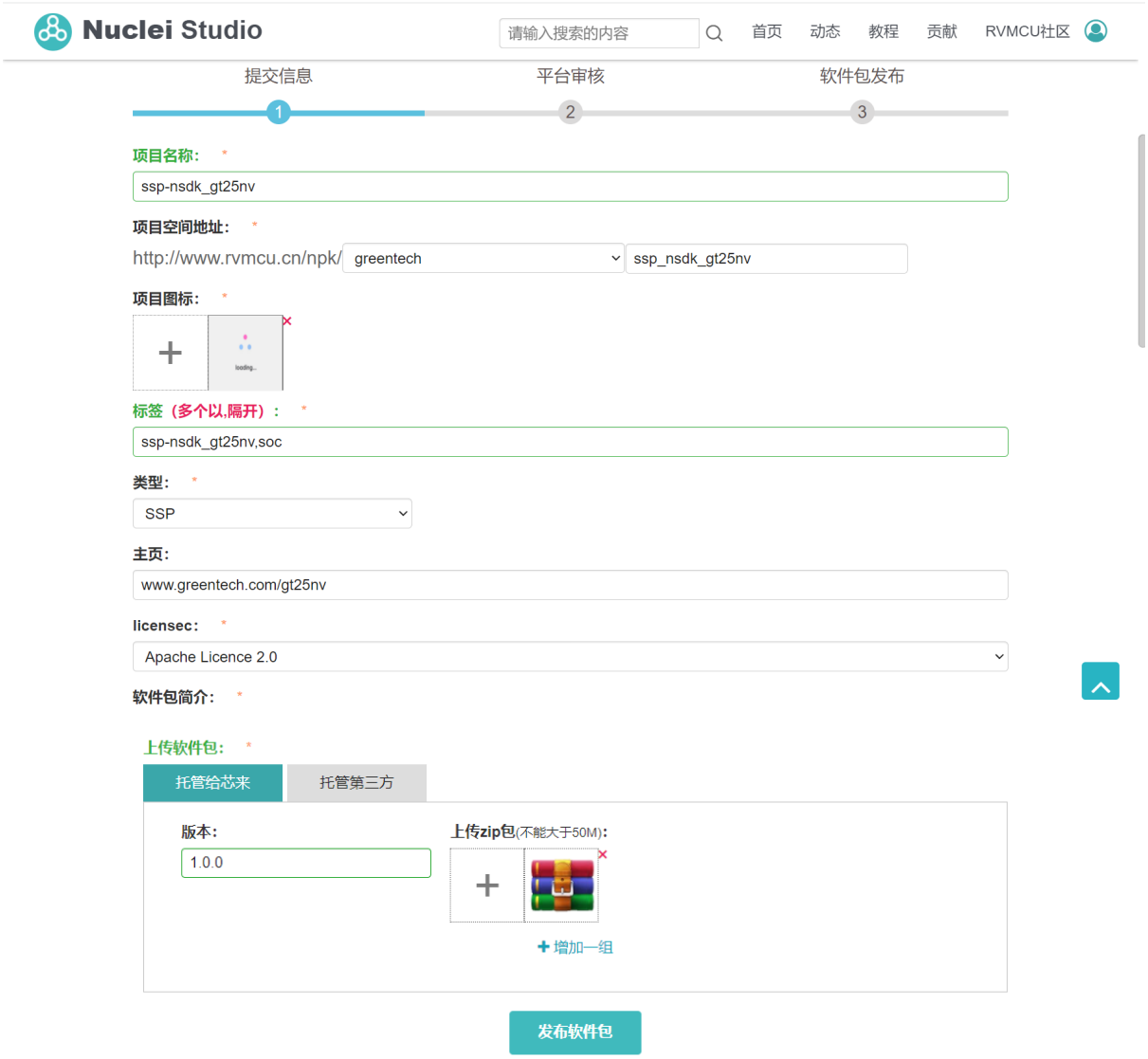
打开 NPK 组件包项目，双击最外层的 `npk.yml`，找到其 Name 为 `ssp-nsdk_gt25nv`，右键点击 NPK 组件包项目，点击 `Export`，选择 `Archive File`，选择需要导出的工程，然后根据提示指定导出 zip 文件存放的位置。


runtime-EclipseApplication - gt25nv/gt25nv/Common/npk.yml - Eclipse Platform
File Edit Navigate Search Project Run RV-Tools Window Help
No Launch Configurations
on:
Project E...
gt25nv
gt25nv
Board
Common
Include
Source
gt25nv.svd
npk.yml
build.mk
runmode.mk
gt25nv.svd
npk.yml
Package Basic
Package Configuration
General
This section describes general information about this file.
Name:
ssp-nsdk_gt25nv
Version:
1.0.0
Type:
ssp: SoC Support Package
Category:
Keywords:
soc
risc-v
GreenTech
Add
Del
Description:
greentech gt25nv Support Package
Owner:
greentech
License:
Apache-2.0
Homepage:
www.greentech.com/gt25nv
Contributors:
GreenTech Team, software@greentech.com
Add
Del
Package Information
Describe Package Information.
Core_vendor:
Nuclei
Name:
GreenTech gt25nv SoC
Vendor:
GreenTech
Reference documents and web sites.
Website:
Datasheet:
Sch:
Usermanual:
Design
Source
Problems
Console
No consoles to display at this time.

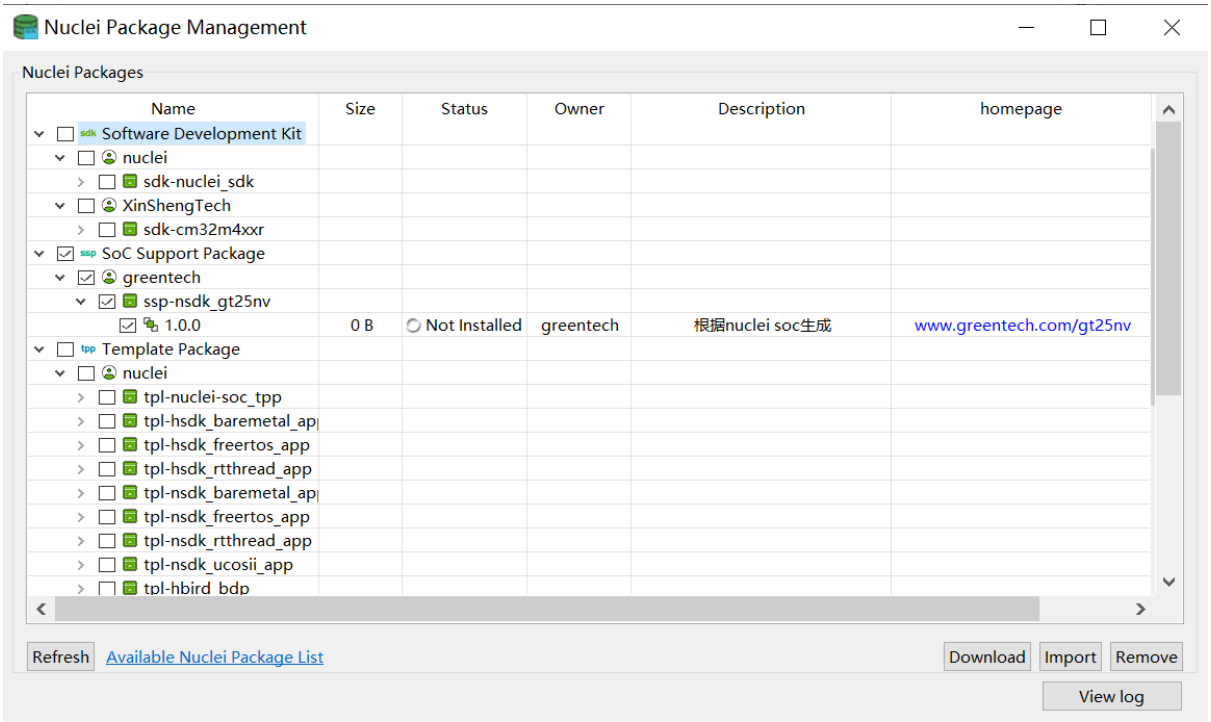


导出的 zip 包，可以通过 rvmcu 社区进行分享贡献。进入 社区分享页面[4]，依据提示信息，依次填写需要分享的 NPK 组件包的名称、所属类型、描述待信息，并上传刚导出的 zip 文件，信息提交后，待管理员审核通过后，该 NPK 组件包就成功贡献了，其他的开发者就可以通过 Nuclei Studio 的 `Nuclei Package Management` 页面找到您的 NPK 组件包，并下载使用。具体操作如下图

[4] https://www.rvmcu.com/nucleistudio-developer.html

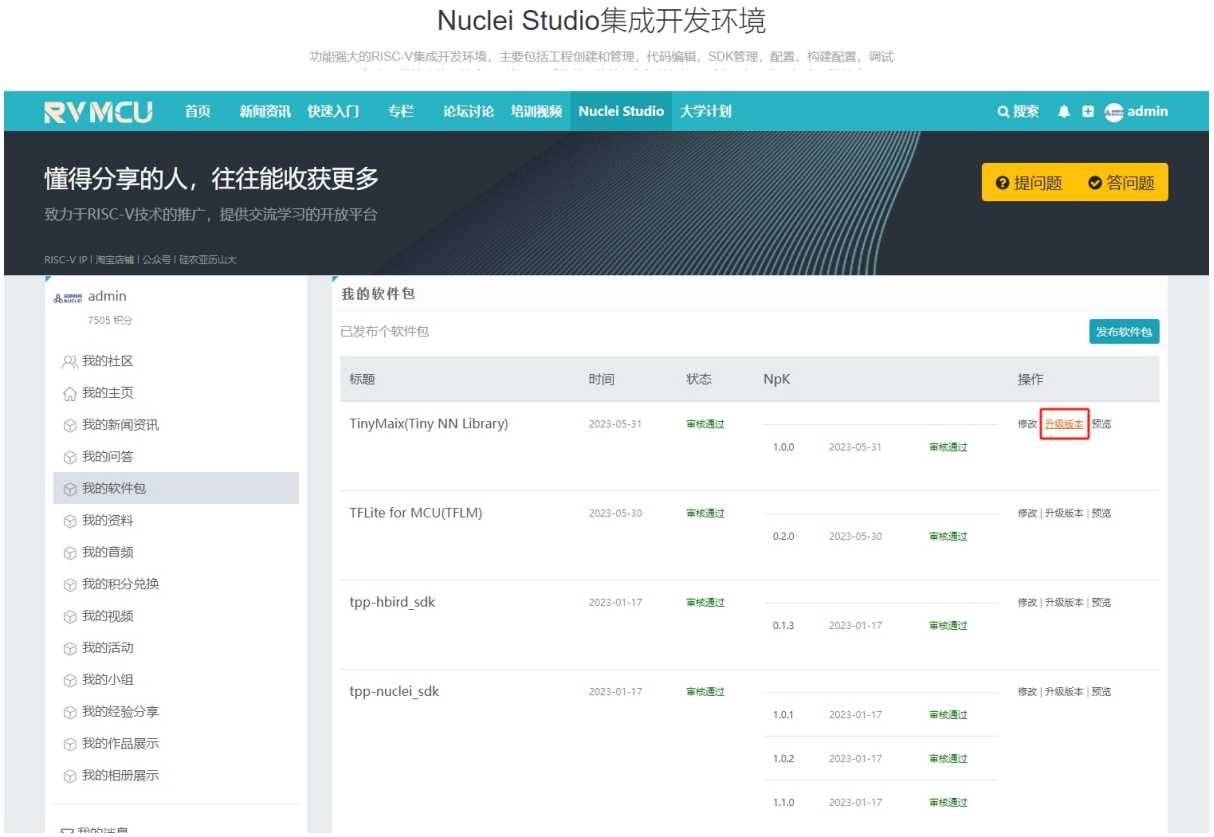

Nuclei Studio
请输入搜索的内容
首页
动态
教程
贡献
RVMCU社区
提交信息
平台审核
软件包发布
1
2
3
项目名称：
ssp-nsdk_gt25nv
项目空间地址：
http://www.rvmcu.cn/npk/
greentech
ssp_nsdk_gt25nv
项目图标：
标签（多个以,隔开）：
ssp-nsdk_gt25nv,soc
类型：
SSP
主页：
www.greentech.com/gt25nv
licensec：
Apache Licence 2.0
软件包简介：
上传软件包：
托管给芯来
托管第三方
版本：
1.0.0
上传zip包(不能大于50M)：
增加一组
发布软件包

分享的 npk 组件包通过审核后，在 Nuclei Studio 中打开 `Nuclei Package Management` 页面，然后点击 `Refresh`，刷新后即可找到刚分享的组件包。

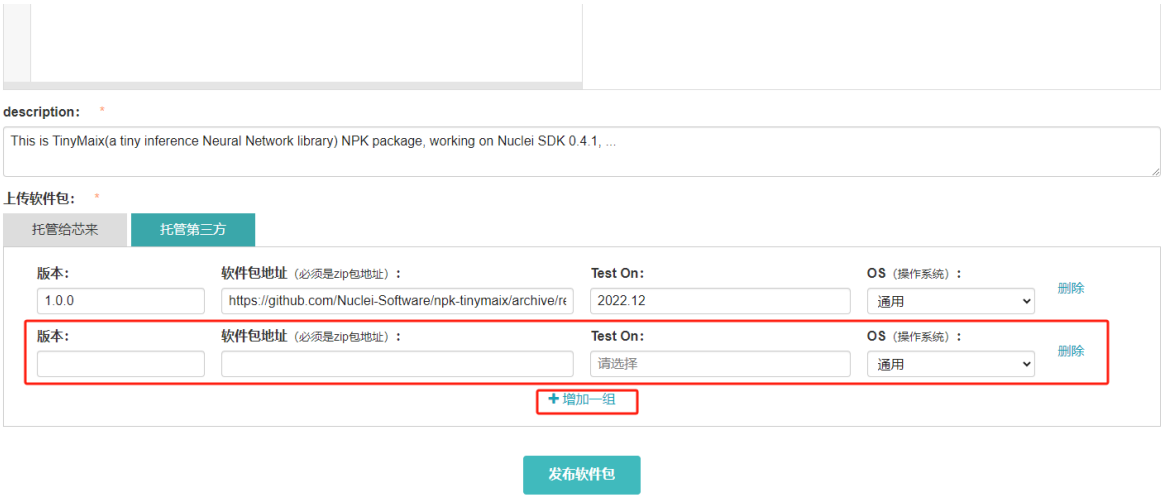


NPK 组件包升级

在 NPK 组件包共享后，如果有新的版本需要维护，在创建测试打包完成后，可以对原有的 NPK 组件包进行升级。共入 Nuclei Studio[5] 页面，找到管理组件包入口，然后进组件包管理页面，点击升级组件包，然后之前的步骤，上传 NPK 组件包，等待审核通过，则组件包升级完成。



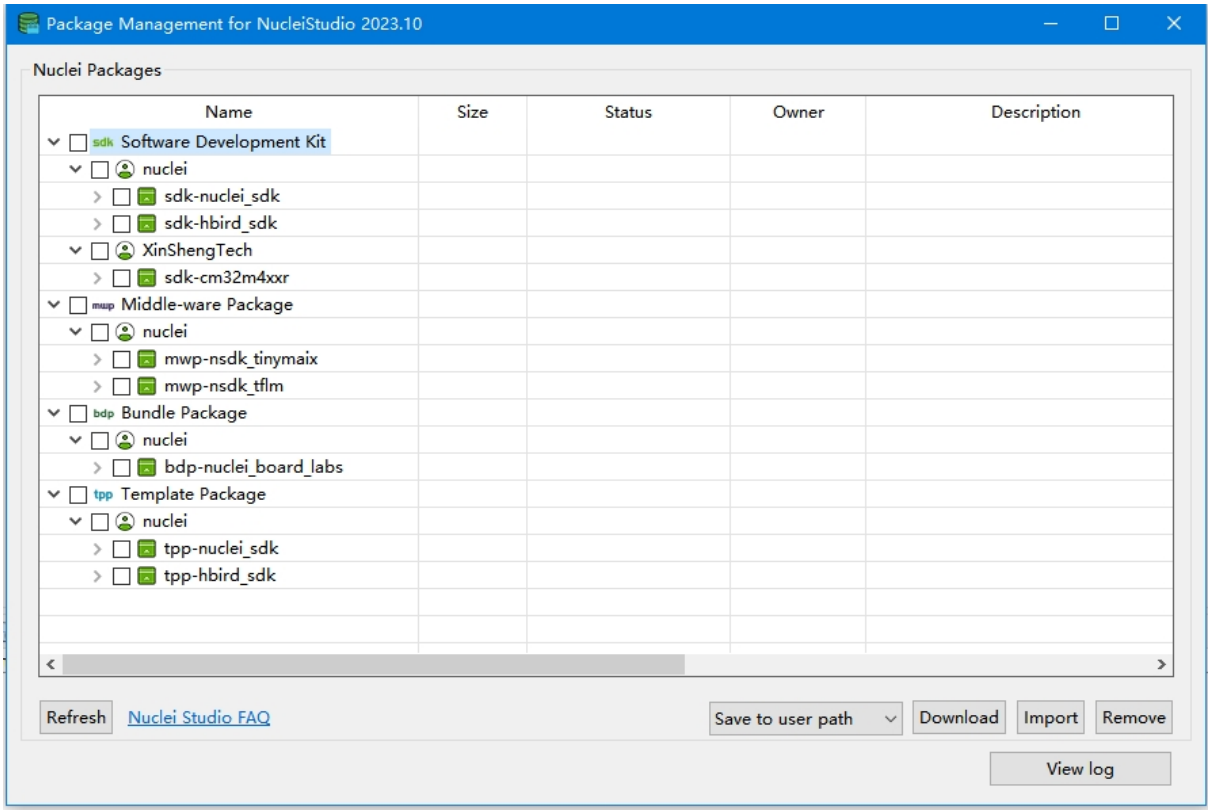
[5] https://www.rvmcu.com/nucleistudio.html



2.5.4 NPK 组件包在 Nuclei Studio 中的使用

NPK 组件包在 Nuclei Studio 中，丰富了其用户体验，通过 NPK 组件包我们可以定义各种不同的创建工程流程，也能很方便的将成熟的工程或者组件共享给其人。

我们所有贡献的 NPK 包，都在 Nuclei Studio 的 `NPK Package Managment` 中进行管理，用户可以在这里进行 NPK 的下载、导入、删除等操作。



2.6 Nuclei Studio NPK 应用

Nuclei Studio 中内建了对 Nuclei Package（NPK）功能的完整支持，方便开发者或者创建不同的软件开发包，并且通过 Nuclei Package Management 方式导入到 Nuclei Studio 中使用，如果是 CPU IP 客户，可以将自己的芯片 SDK 略作改造以支持 NPK，并导入到 Nuclei Studio 中使用，如果是 SoC IP 客户，可以将提供的 SDK 打包成 Zip 包，导入使用，如果是开发者，也可以依赖某个特定的 NPK，创建对应的 BSP 包或者 APP 包并提供给第三方使用，关于 NPK 功能的详细介绍，以及后续更新说明参见 https://github.com/Nuclei-Software/nuclei-sdk/wiki/Nuclei-Studio-NPK-Introduction 。通过在原有的 SDK 中引入 npk 功能，编写 npk.yml 可以达到 Project Wizard 功能的部分定制化。

开发者要使用 Nuclei Studio 进行工程的创建，需先将对应的 SDK NPK Zip 包安装到 IDE 中，方可根据不同的开发板快速新建不同的模板工程，并根据不同的模板添加需要的 SDK 源码，根据选项生成不同的编译链接选项设置。

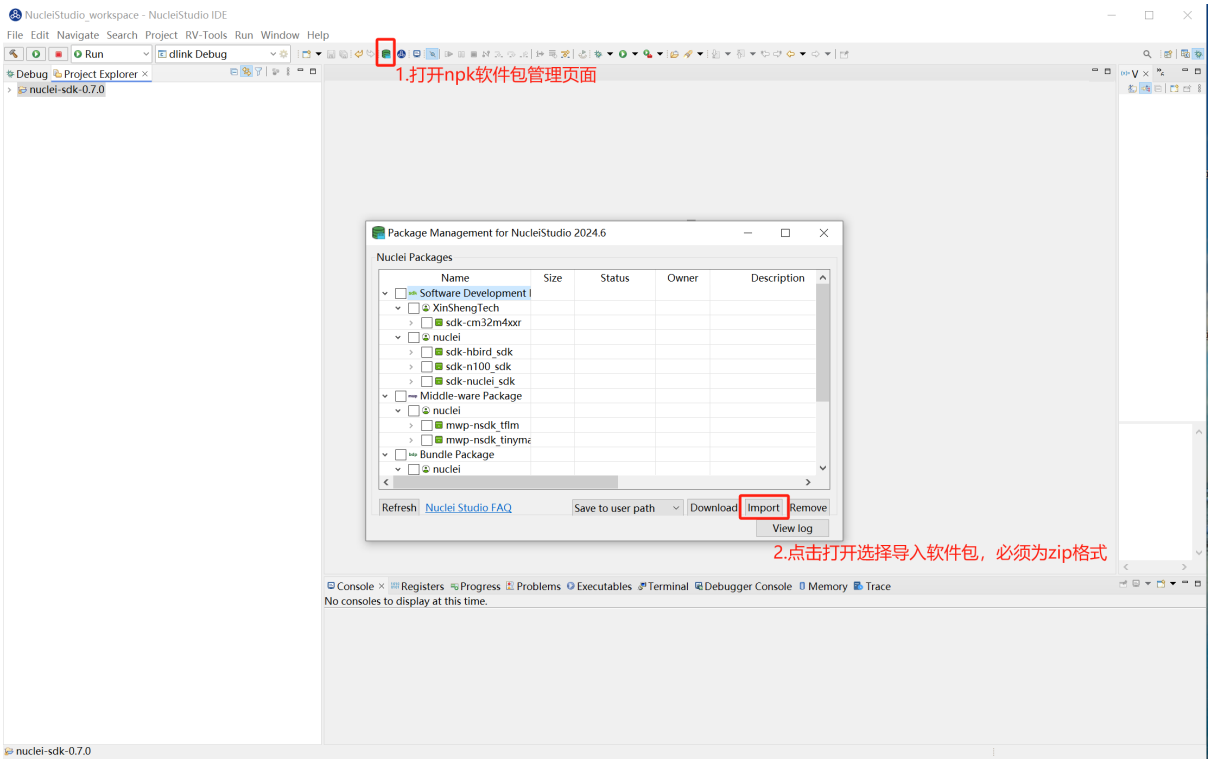
2.6.1 NPK 软件包管理

导入本地 NPK 软件包

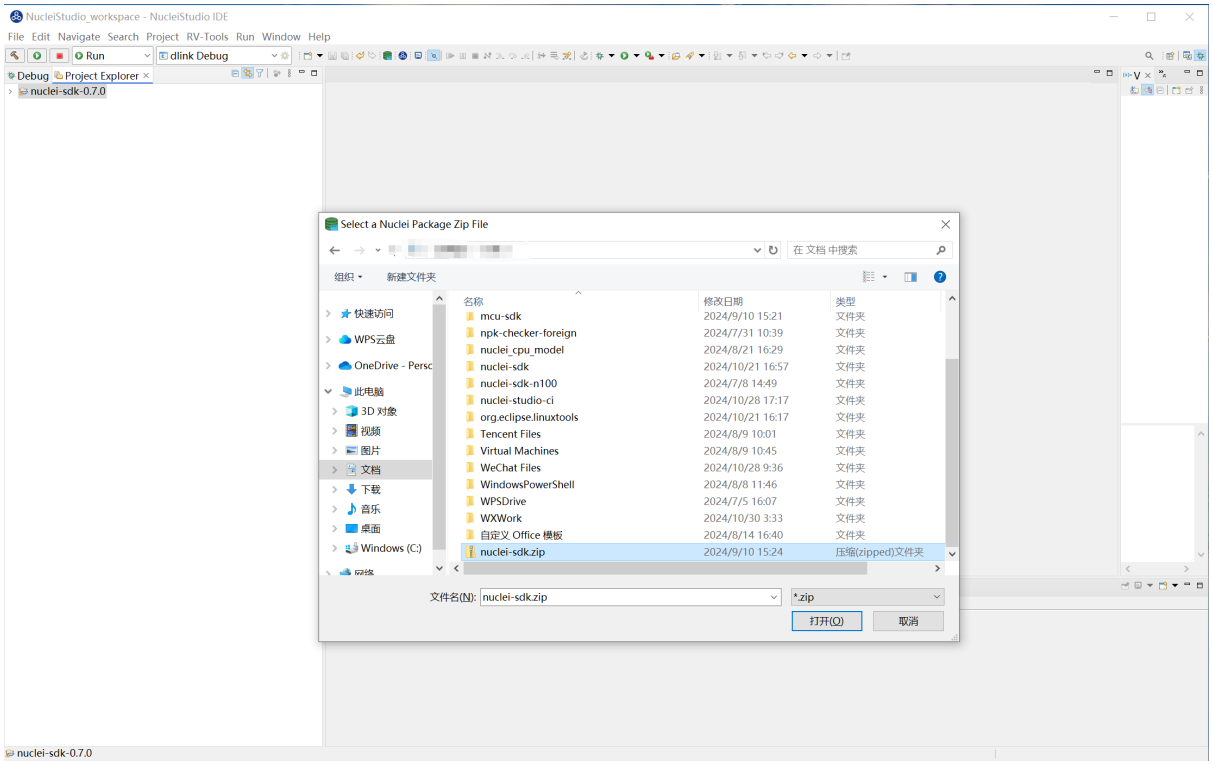
导入 **zip** 软件包

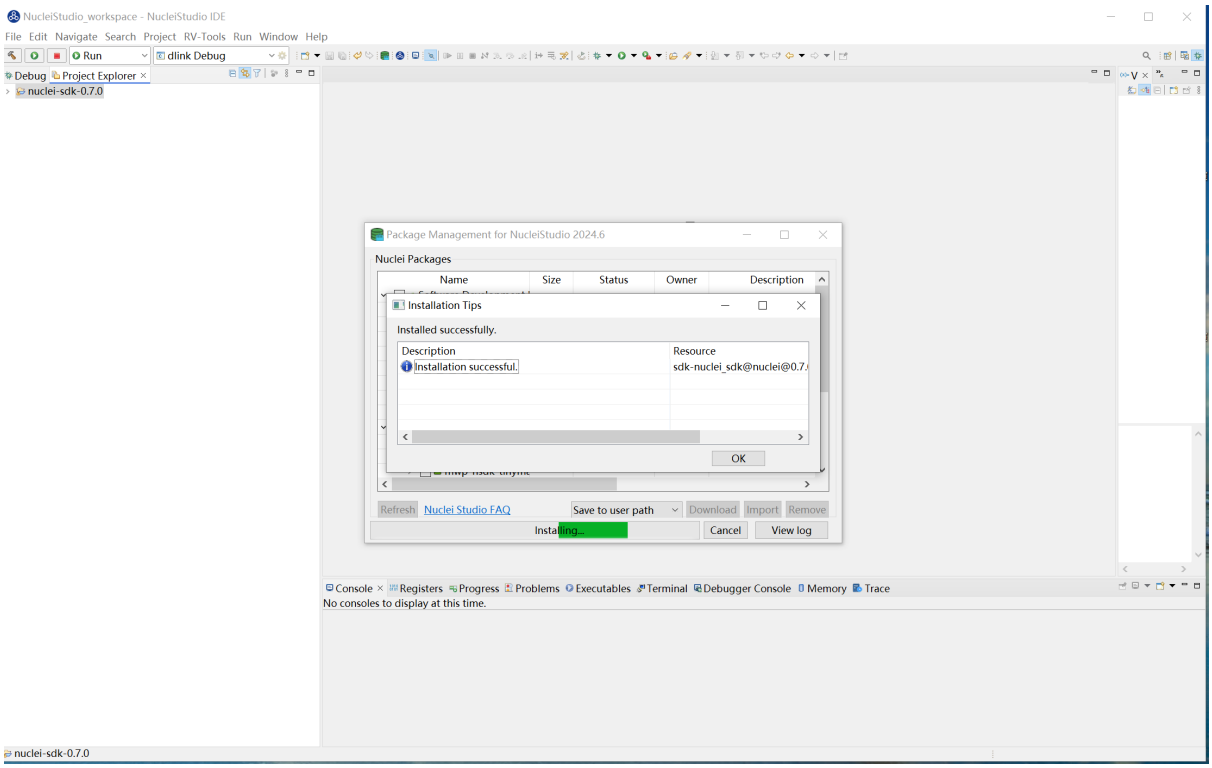
当用户拿到一个 zip 格式的软件包时，可以通过 Nuclei Package Management 导入一个 zip 格式的软件包，你可以按照以下的步骤操作：

首先，点击 Nuclei Studio IDE 工具栏上的 **Nuclei Package Management** 按钮，这将打开 npk 软件包管理页面。在这个页面上，你可以管理和浏览已安装的软件包，以及导入新的软件包。在这里，我们找到并点击 **Import** 选项。这个选项用于从本地文件系统选择 zip 类型的软件包。

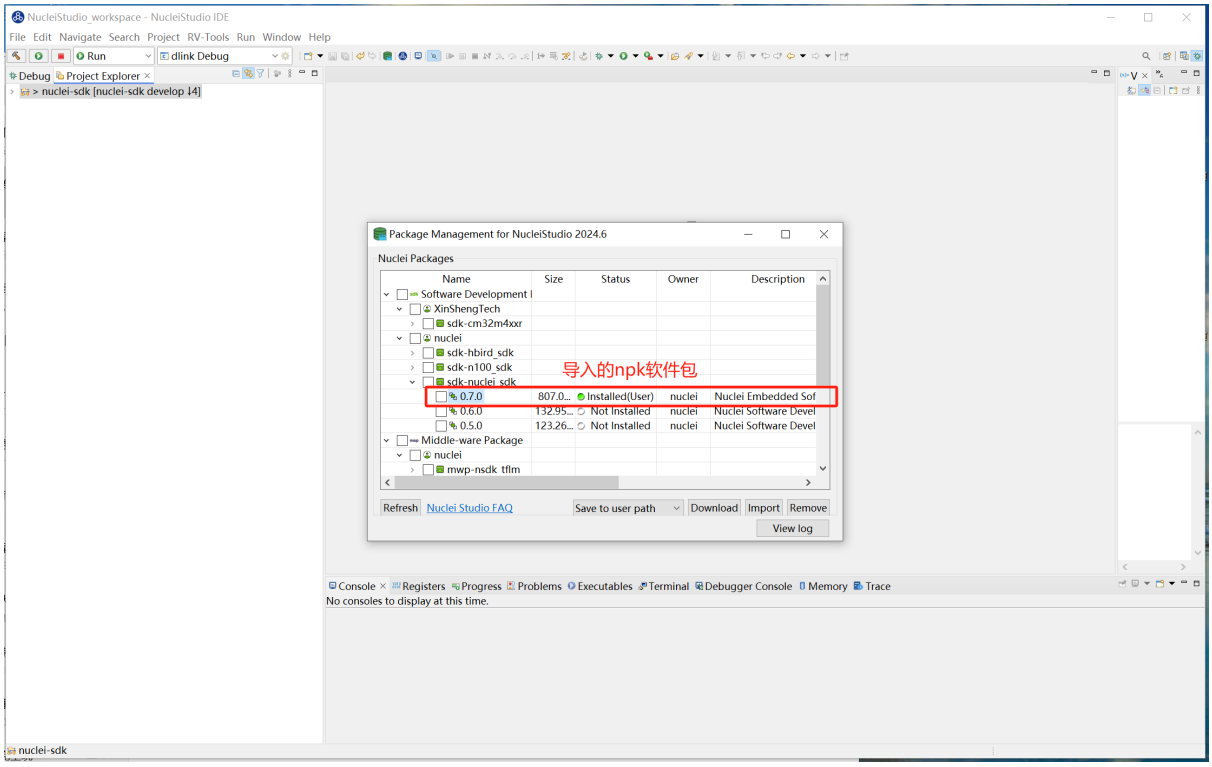


在点击 **Import** 后，系统会提示你选择一个要导入的软件包文件。此时，你需要浏览到你的 zip 格式软件包所在的目录，并选中它。确保选中的是一个有效的 zip 包，点击 打开，系统会开始处理并导入这个软件包。





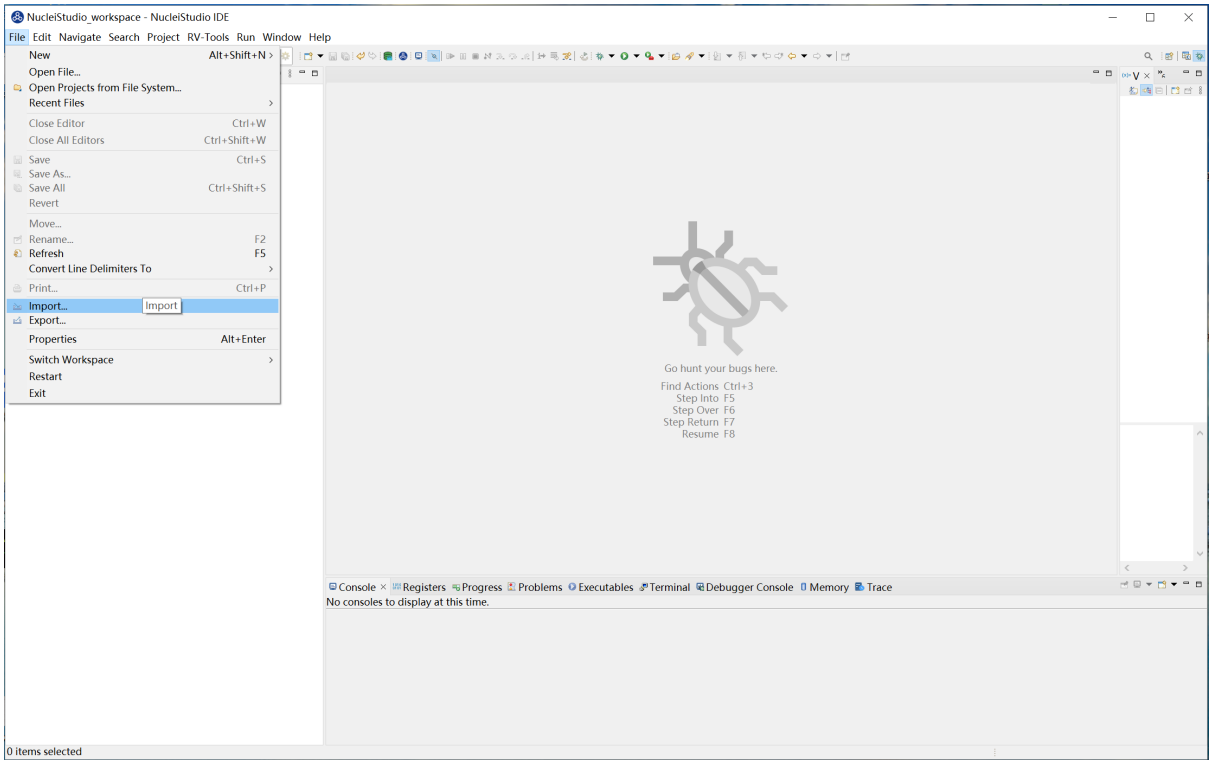
导入完成后，你应该能够在 Nuclei Studio IDE 的 Package Management 页面中看到新导入的软件包。



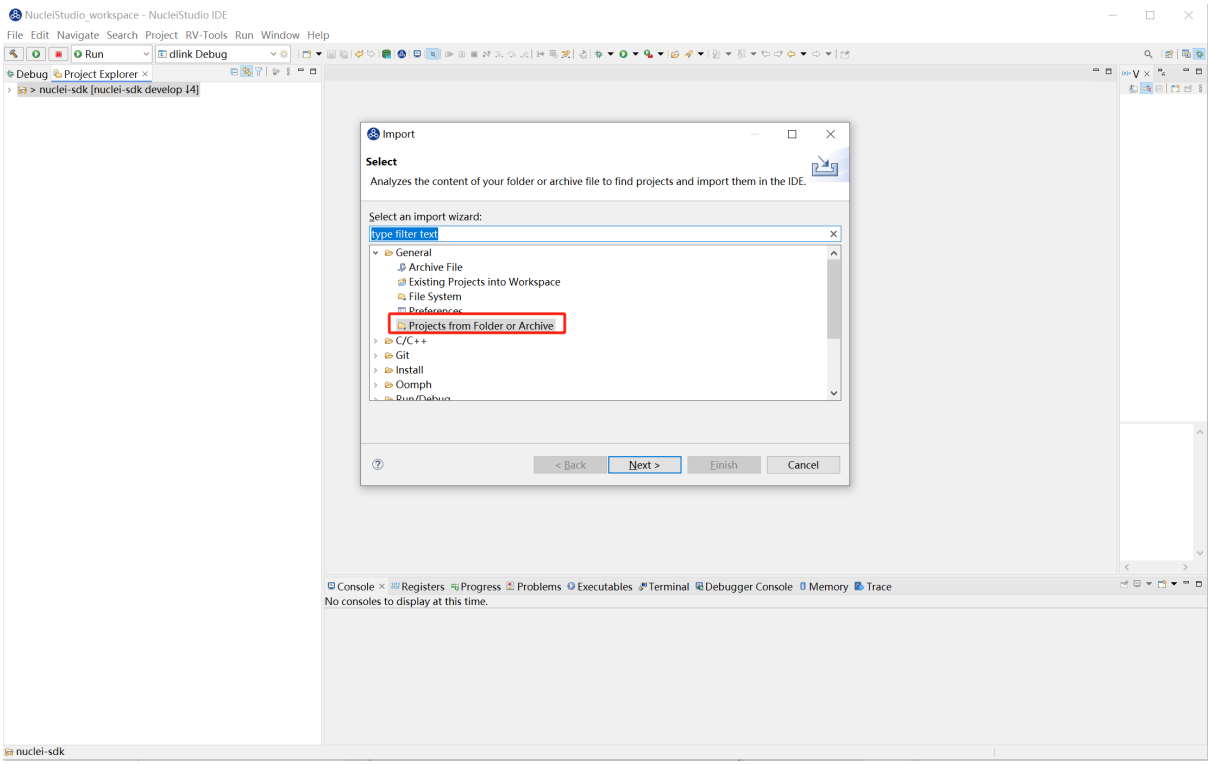
导入未压缩的 **npk** 软件包

当用户拿到一个未压缩的 npk 软件包工程时，如果想将它导入到 NucleiStudio 中，最简单的方式就是通过压缩软件，将其压缩成为一个.zip 压缩包，然后通过上述操作导入到 NucleiStudio 中，如果想一边修改这个 npk 软件包工程，修改完后快速导入到 NucleiStudio 中进行测试，可以通过如下操作实现。

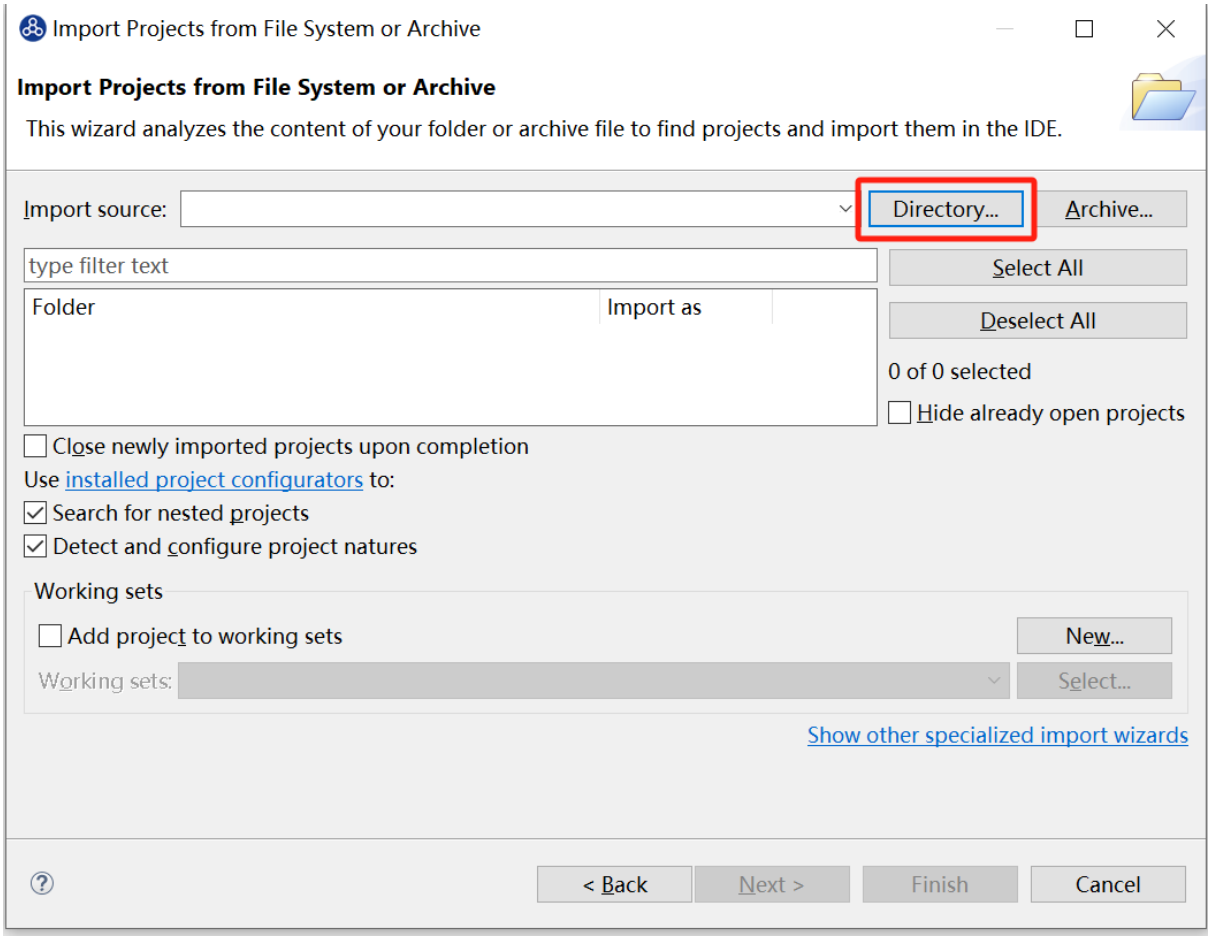
直接导入未压缩的软件包，先要将软件包导入到 Project Exlorer 视图中。点击菜单栏上的 File 选项，在下拉菜单中，找到并点击 **Import**…选项。这将打开导入向导，帮助你从本地文件系统或其他来源导入新的项目或文件。



然后选择 General 下的 **Projects from Folder or Archive** 选项，点击 Next 按钮。

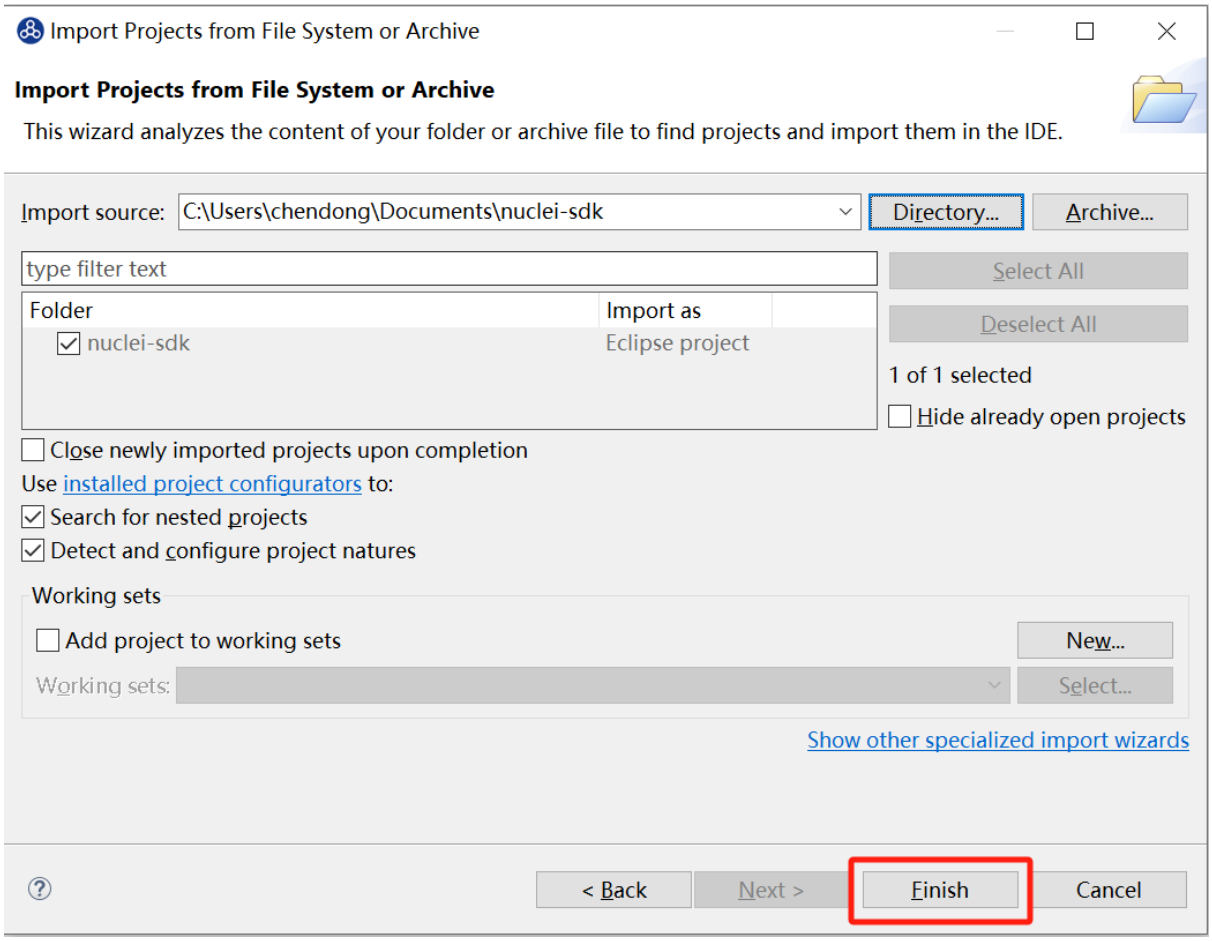


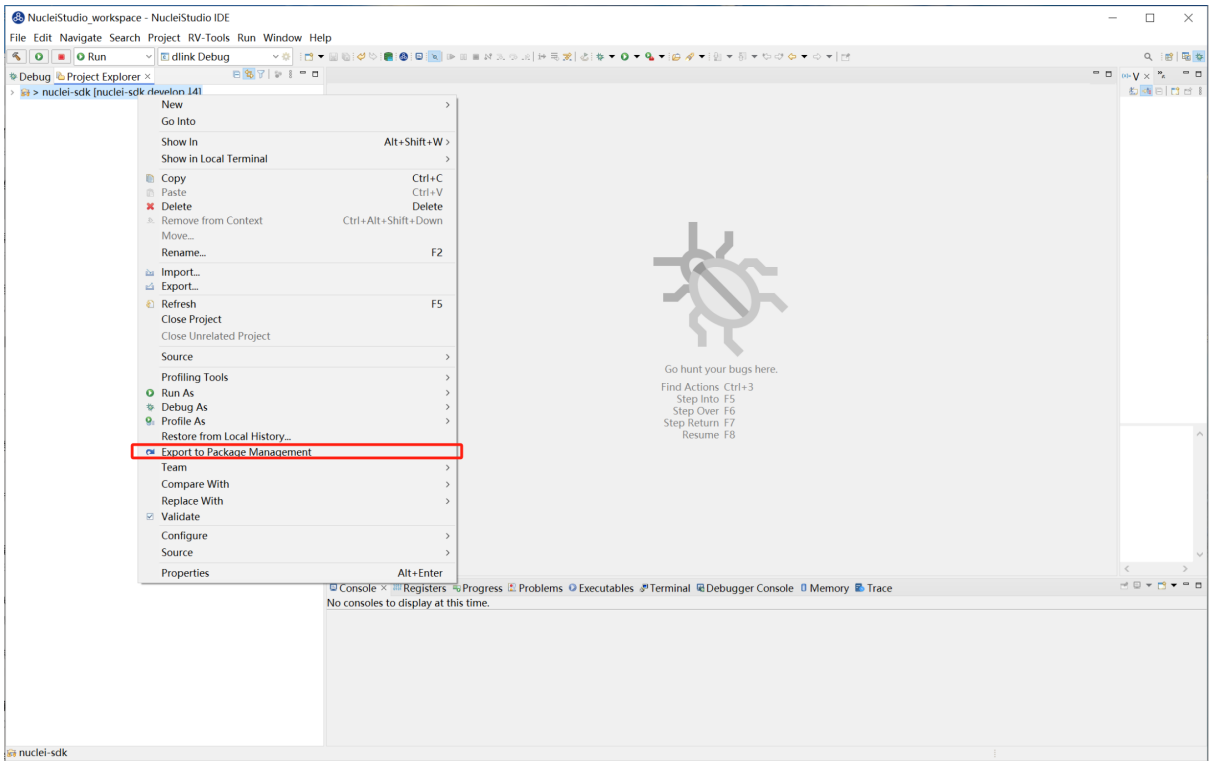
接下来，点击 **Directory**…，这里浏览到你的软件包所在的目录，并选择要导入的文件或文件夹。确保你选择了正确的路径和文件，然后点击 **Finish** 按钮。



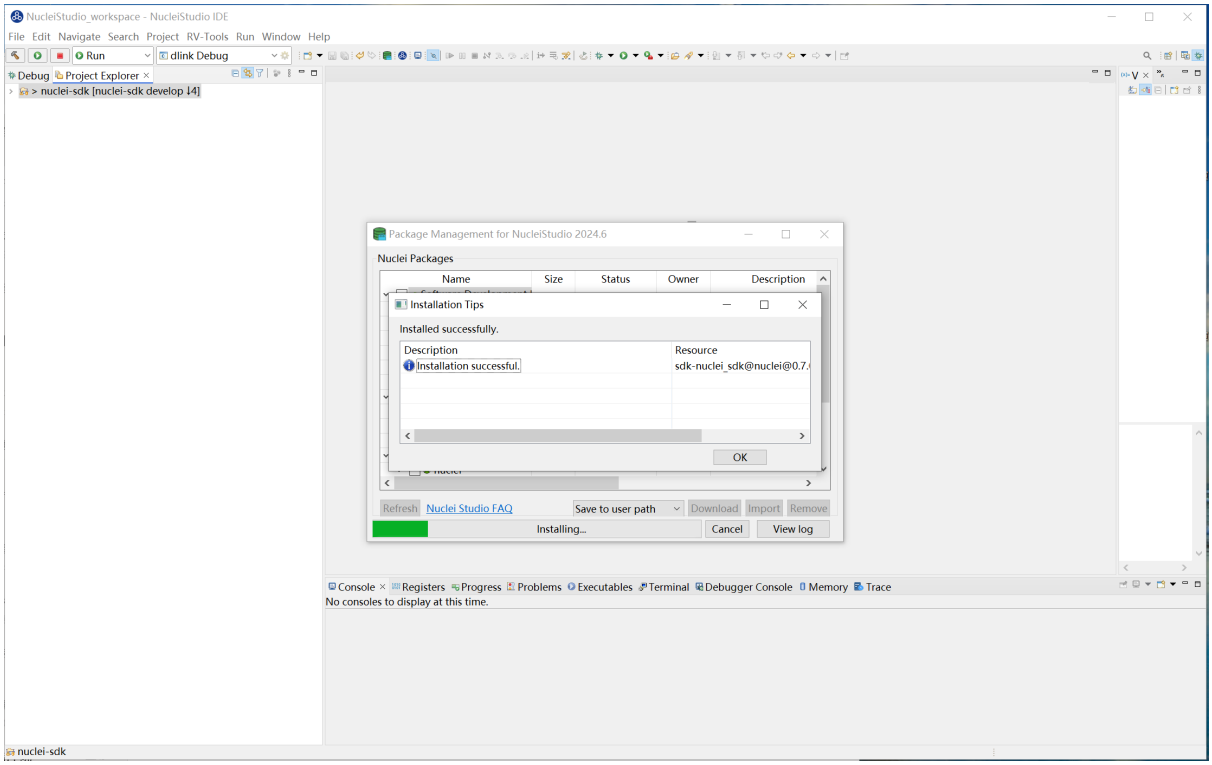
此时软件包已导入到 **Project Explorer** 视图中，右键点击这个文件夹（代表你的软件包），在弹出的

上下文菜单中选择 **Export to Package Management** 并点击，系统会开始处理并导入这个软件包。



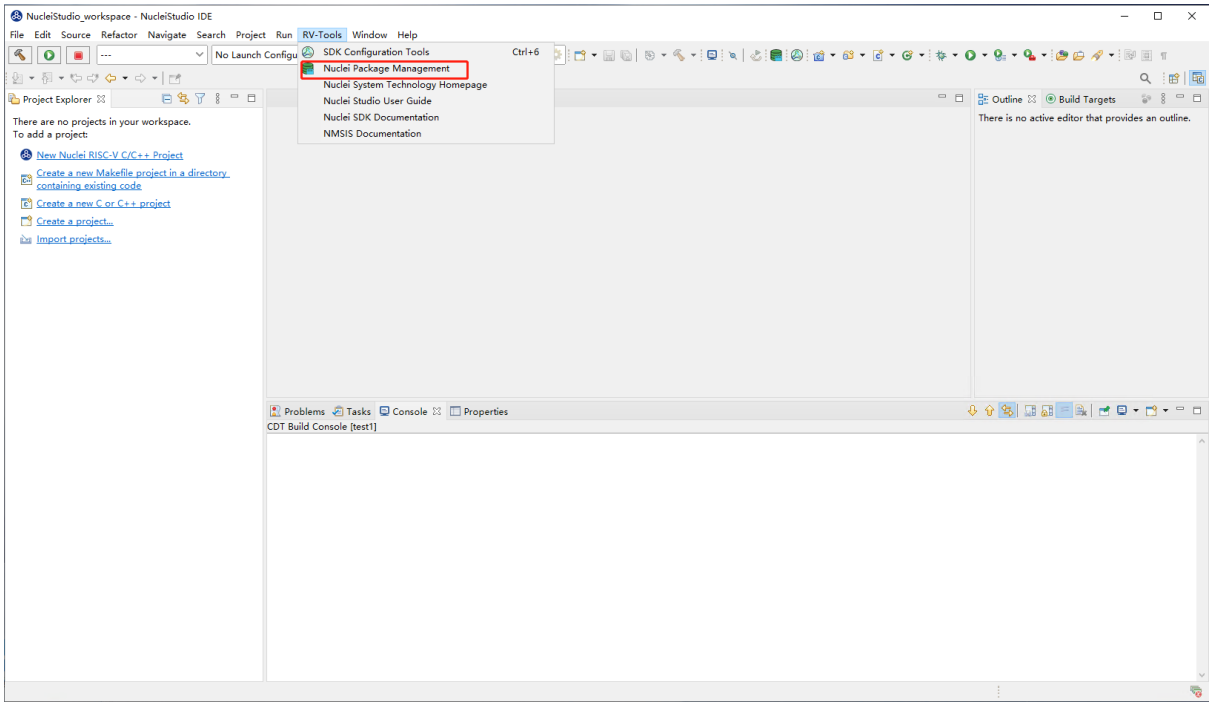


导入过程中，会自动打开 Package Management 页面。待导入完成后，你应该能够在 Nuclei Studio IDE 的 Package Management 页面中看到新导入的软件包。

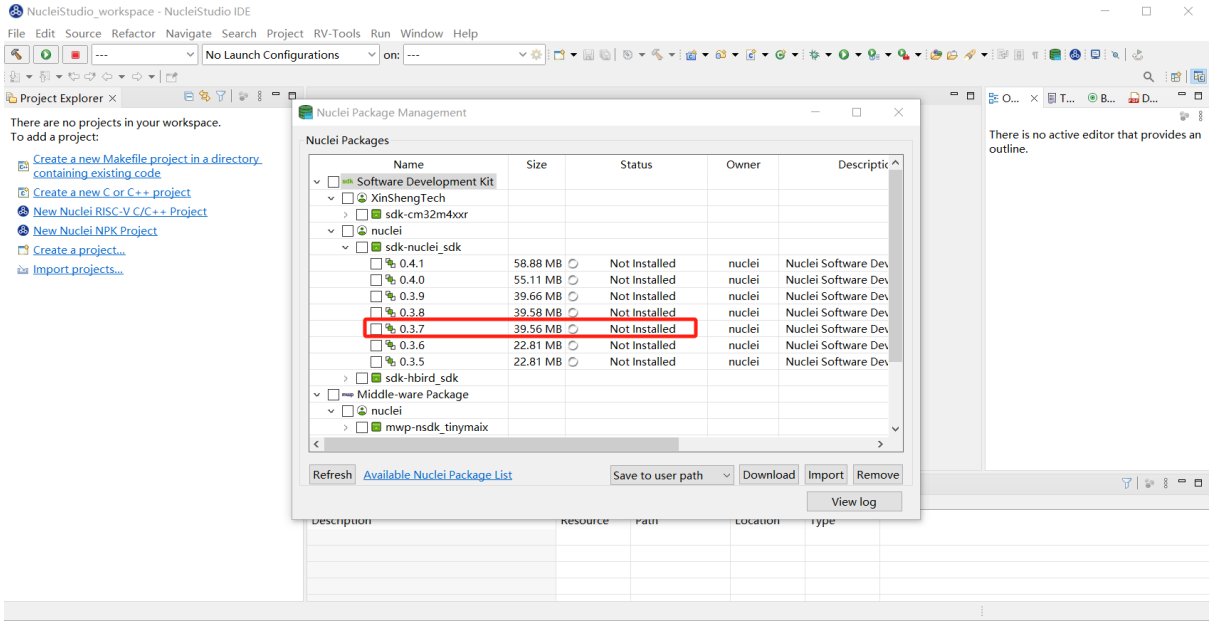


下载云端 NPK 软件包

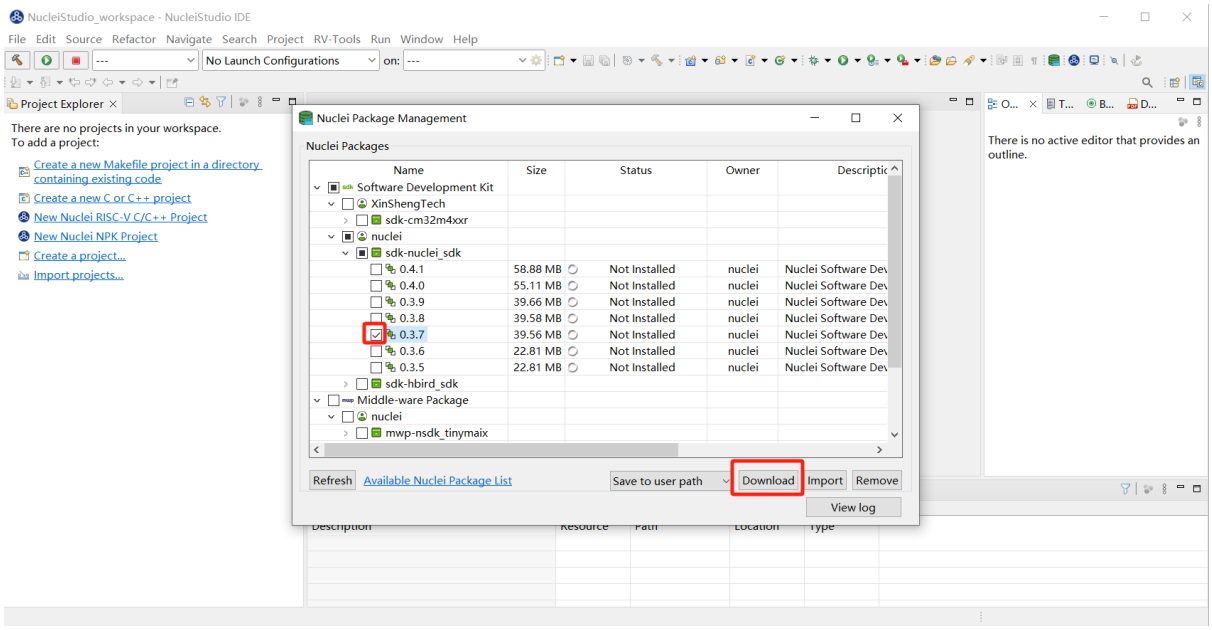
在 Nuclei Studio 中最大的更新，就是将 npk 云端化，用户直接在 Nuclei Studio 中就可以下查看到所有的 npk 并自行安装，在菜单栏选择 `RV-Tools-->Nuclei Package Management` 在弹出的 **Nuclei Package Management** 管理页进行 npk 管理。



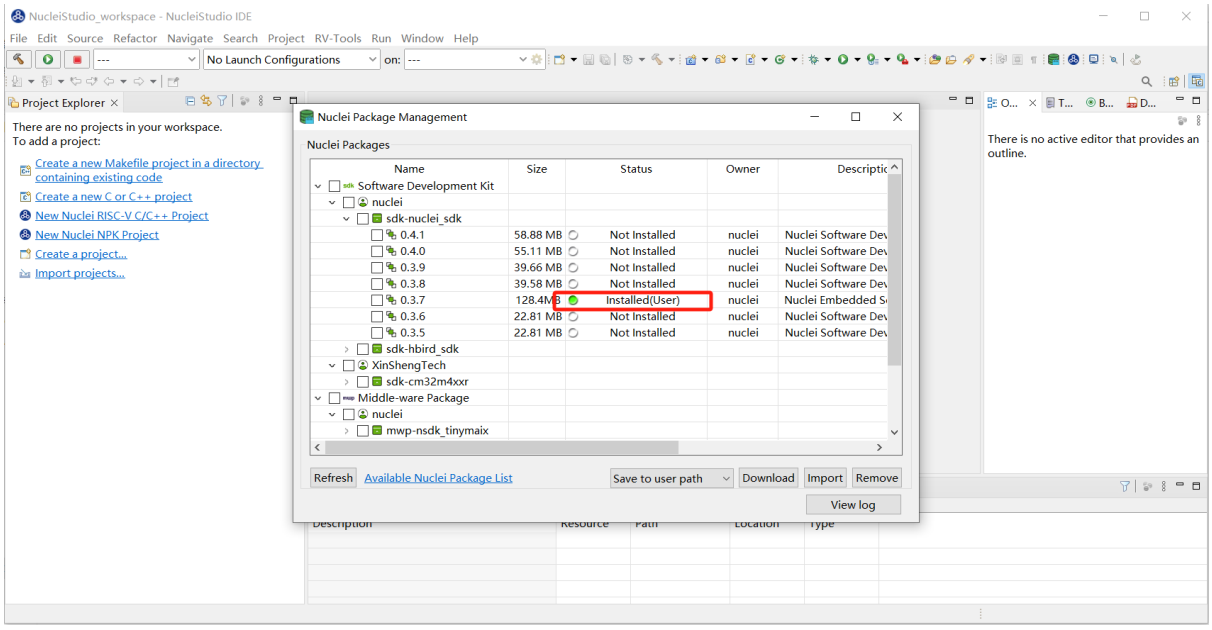
在 Nuclei Package Management 页面，先点击一下 Refresh 获取最新的 npk 信息，npk 安装前状态为 Not Installed。



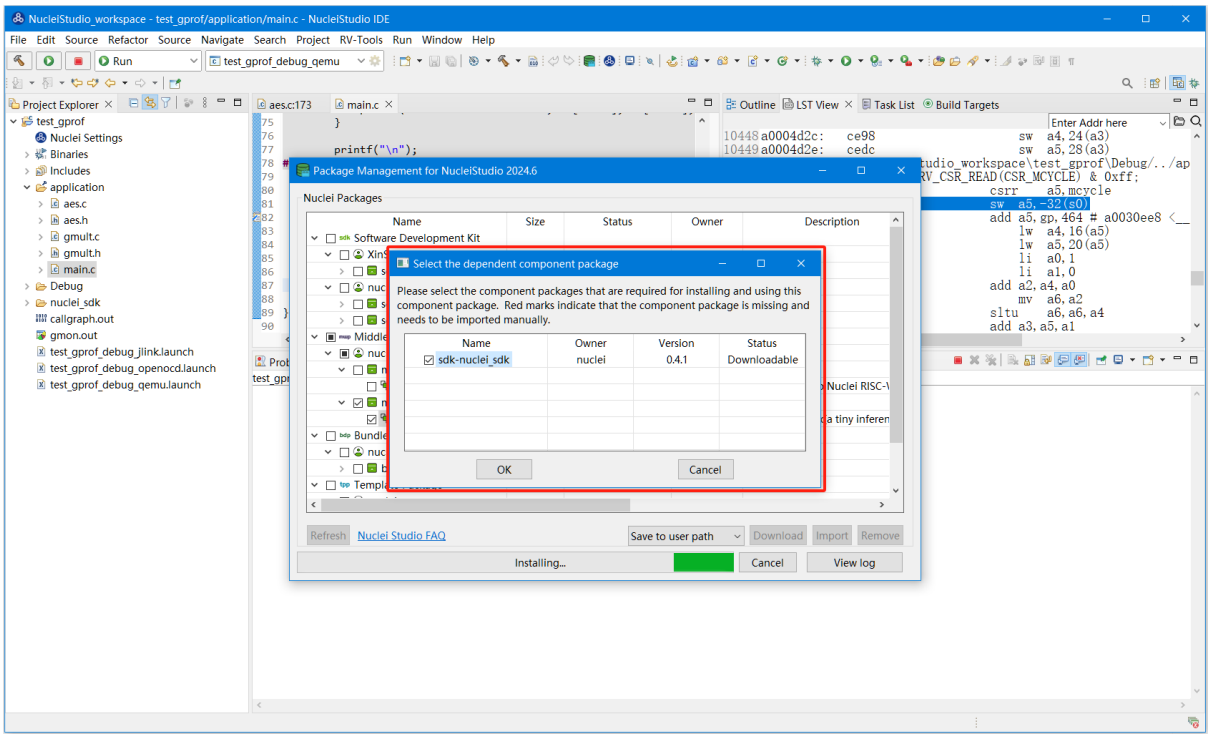
然后选中自己想要安装的 npk, 点击 Download 进行安装，本教程安装 sdk-nuclei_sdk 的 0.3.7 版本，最新版本为 0.5.0，请使用最新版本进行安装，最新版本支持 gcc13 工具链，之前版本支持的是 gcc10。



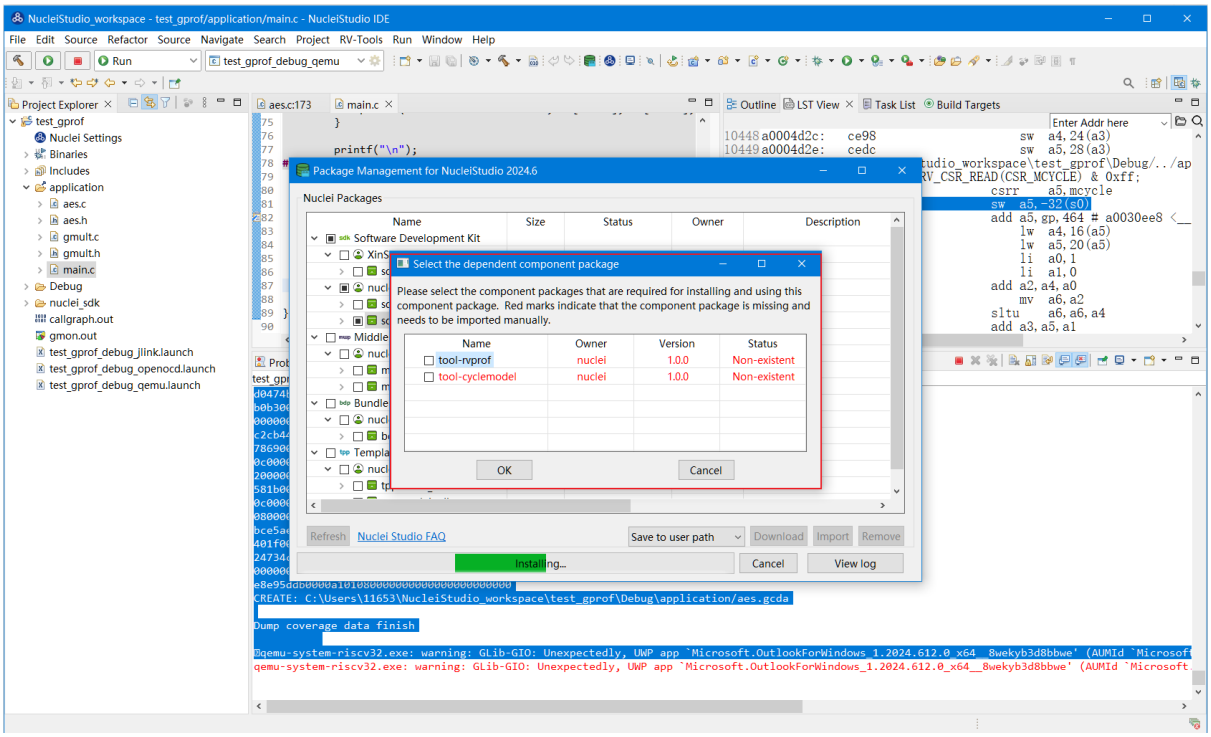
安装完成后 npk 状态变为 Installed。



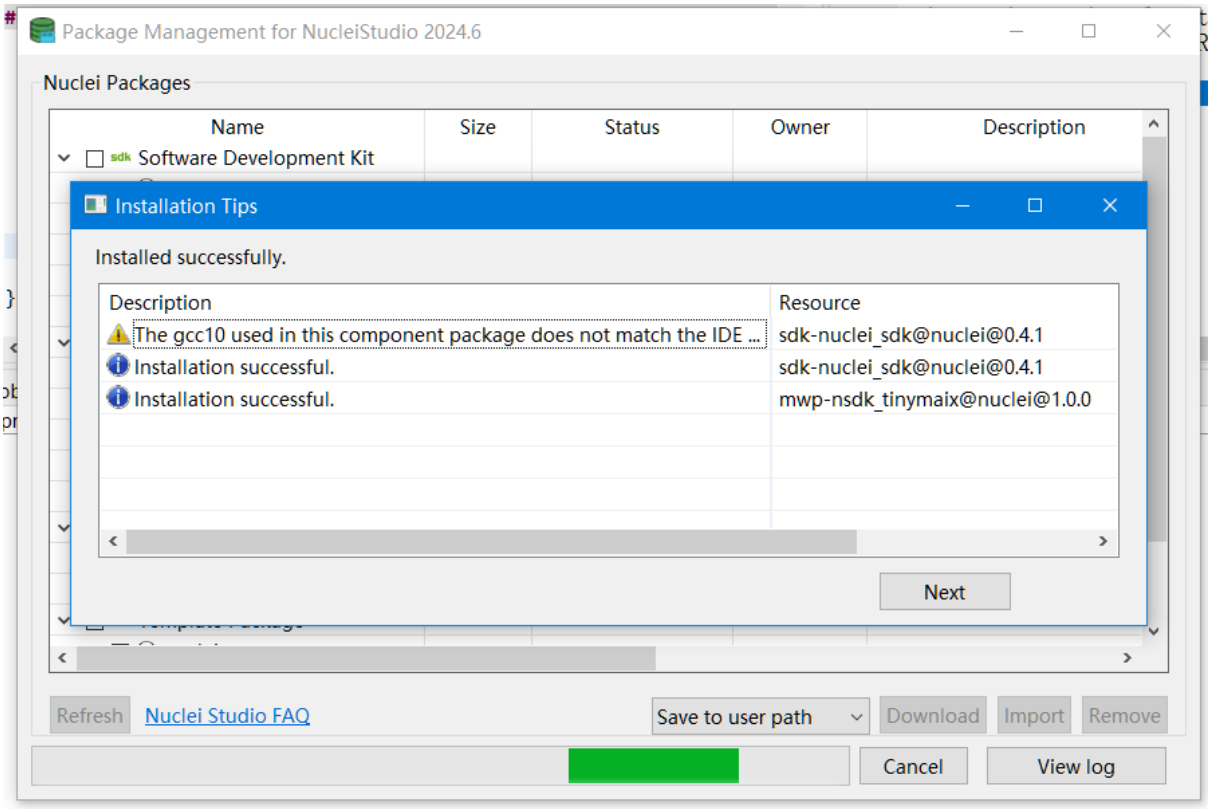
在 NucleiStudio 2024.06 版本中，升级了 Nuclei Package Management 管理页中关于依赖的管理，大大简化了使用者关于 NPK 包依赖的问题。



当用户在下载/安装某个 NPK 包时，NucleiStudio 会根据当前 NPK 包信息的依赖关系，再结合本地已安装的包信息及云端共享的 NPK 信息，给出其的所有依赖，并提示用户进行关联下载安装。



当 NPK 包安装完成后，会提示用户，此安共计安装了多少个 NPK 包，每个 NPK 包安装的后状态以及出错的原因。方便用户排查问题。极大的解决的之前版中用户因 NPK 包依赖的不正确而无法使用的问题。

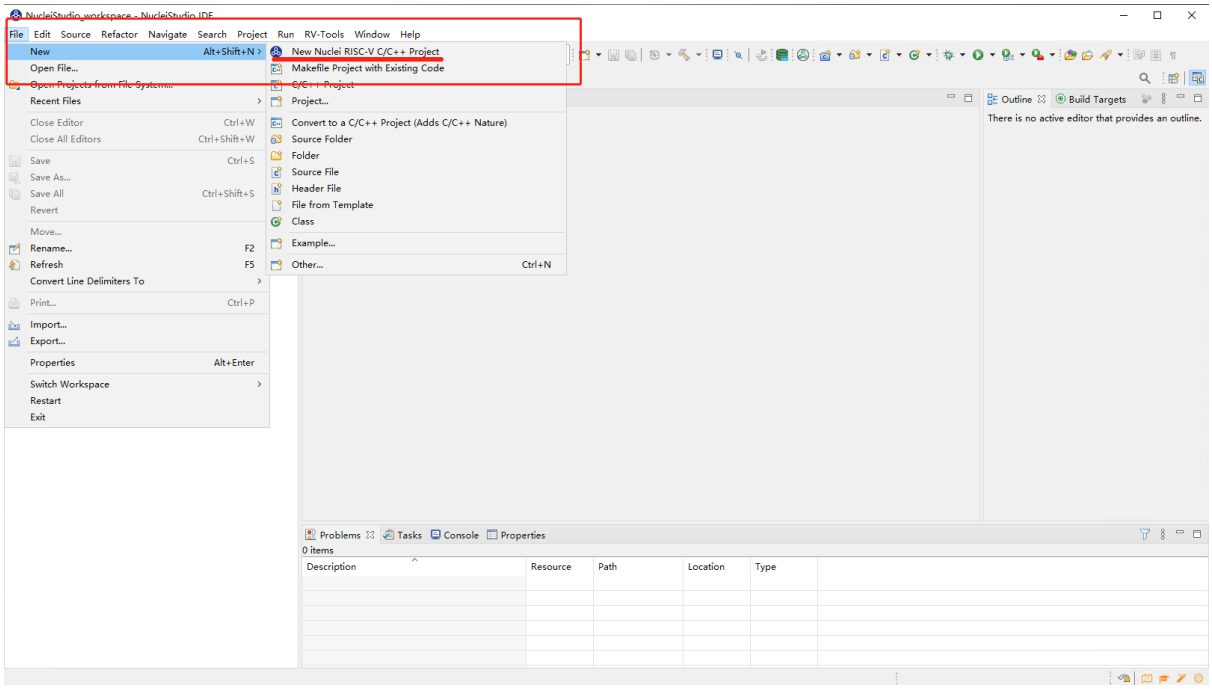


2.6.2 通过 NPK 创建工程

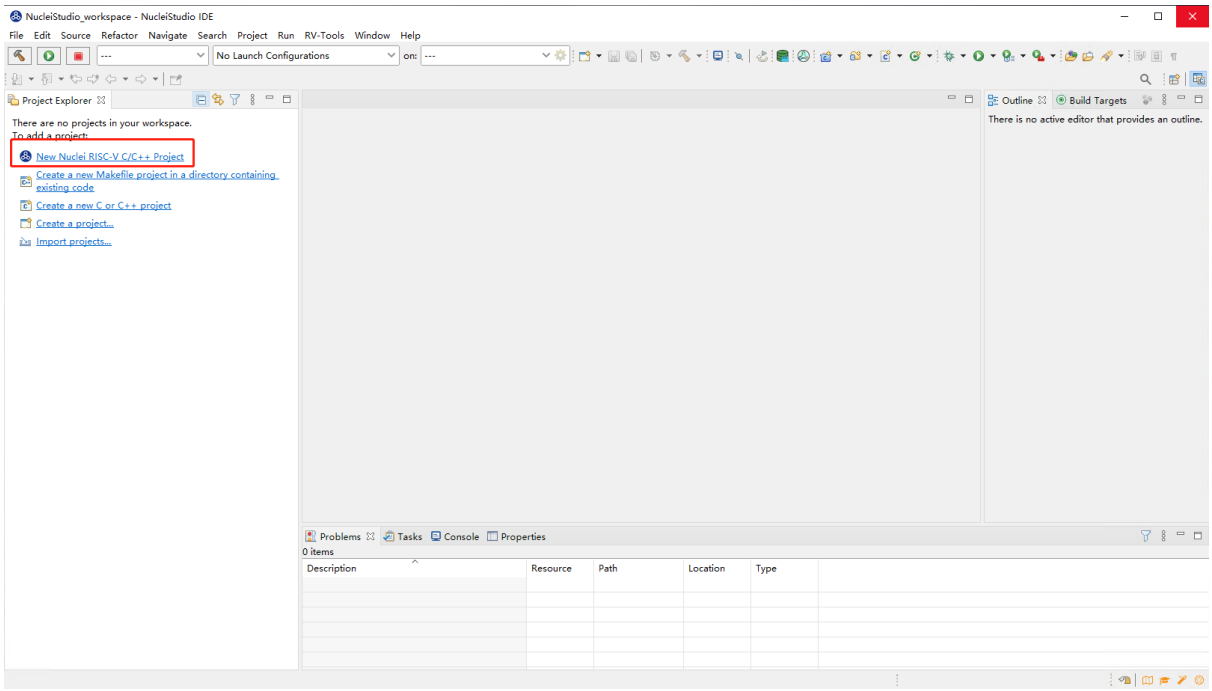
本章将在 RVSTAR 开发板上，以新建和修改 GD32VF103 的工程为例快速介绍 Nuclei Studio 功能，RVSTAR 开发板开发需要使用 nuclei_sdk 的 npk 包，详细的流程请参考之后的章节。

创建 NPK 示例工程

新建一个工程，可以在菜单栏中，选择 `File --> New --> New Nuclei RISC-V C/C++ Project` 。



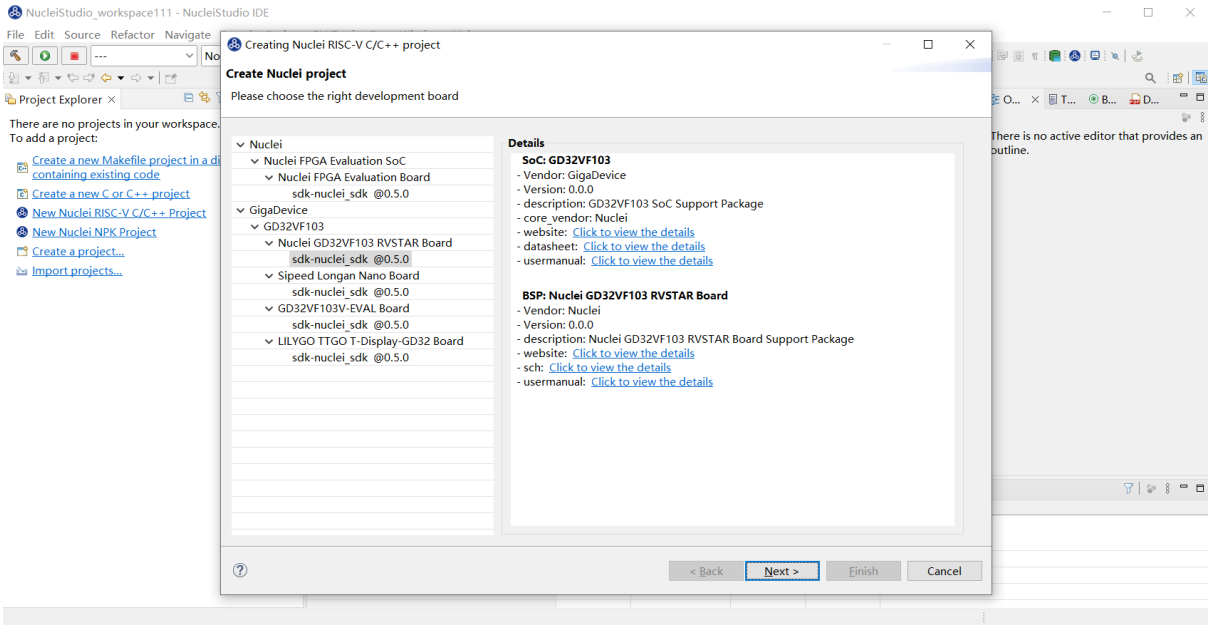
也可以在 Project Explorer 视图中选中 `New Nuclei RISC-V C/C++ Project` 。



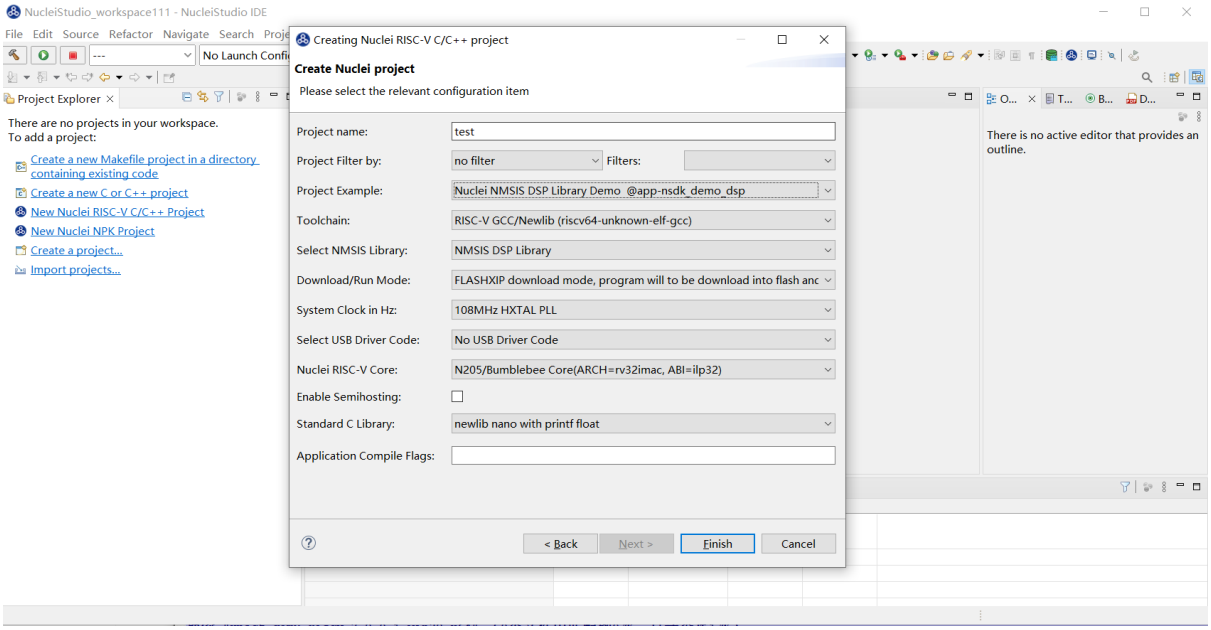
在弹出的窗口中可以不同的厂商提供的不同版本的 SDK，选种某一 Board 下的 SDK，看到相关 SoC 及 Board 的介绍。这里以 RVSTAR 开发板为例，所以这一项选择 `GigaDevice->GD32VF103->Nuclei`

GD32VF103 RVSTAR Board->sdk-nuclei_sdk@0.5.0 。点击 Next 进入下一步。

Note: 注意：这里的 sdk 版本号会随着版本迭代做相应的更新，并且也可能依赖特定版本的 Nuclei Studio 使用



进入具体的项目配置页如图所示，因为 RVSTAR 的内核是固定的 N205，其对应的 arch 和 abi 分别是 rv32imac 和 ilp32，所以 Core 选项不能修改。同样，RVSTAR 开发板仅支持一种 FLASHXIP 下载模式，所以 DOWNLOAD 这一选项也不能修改。点击 Finish 完成工程创建。在 2023.10 版本，增加了对 Arm 项目的支持。



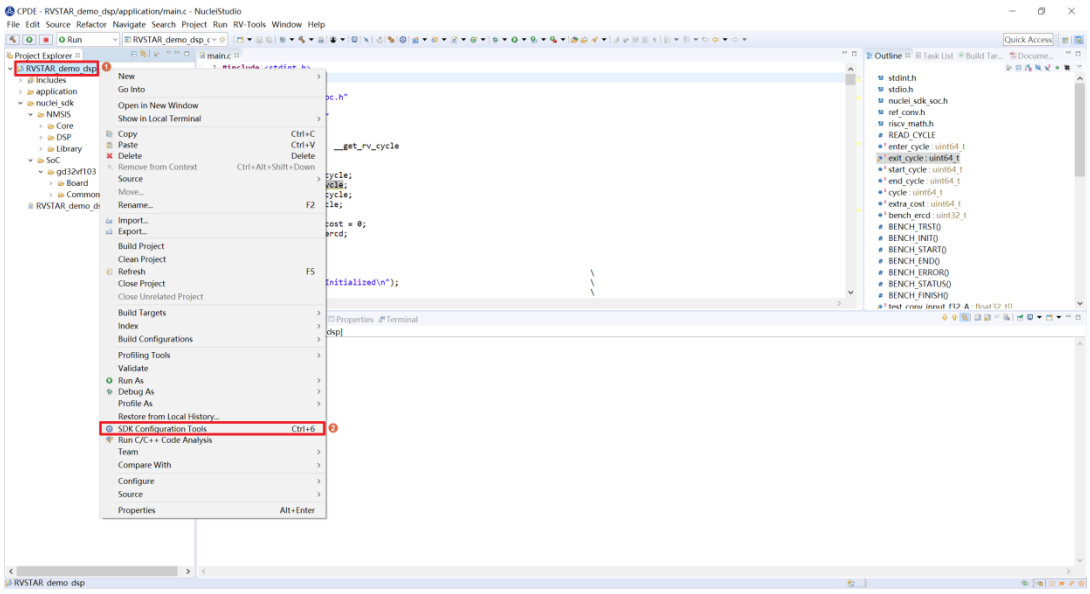
Nuclei Studio 可以根据不同的工程模板添加不同的 SDK 源码，例如 FreeRTOS 模板工程会添加对应的 OS 内容，Demo_dsp 模板工程可以添加 NMSIS 库文件。关于 NMSIS 详细信息请参考（https://doc.nucleisys.com/nmsis/index.html）。这里以 Demo_dsp 为例，Project Example 选择 Nuclei NMSIS DSP Library Demo 。因为使用 dsp 工程，需要添加 NMSIS 库，所以 Libraries 选择 NMSIS DSP Library 。

Nuclei Studio 可以根据新建工程时的选项自动设置工程的选项。这里选择使用浮点打印，所以 NEWLIB 选择 newlib nano with printf float 。之后一直选择 Next 直到 Finish 。

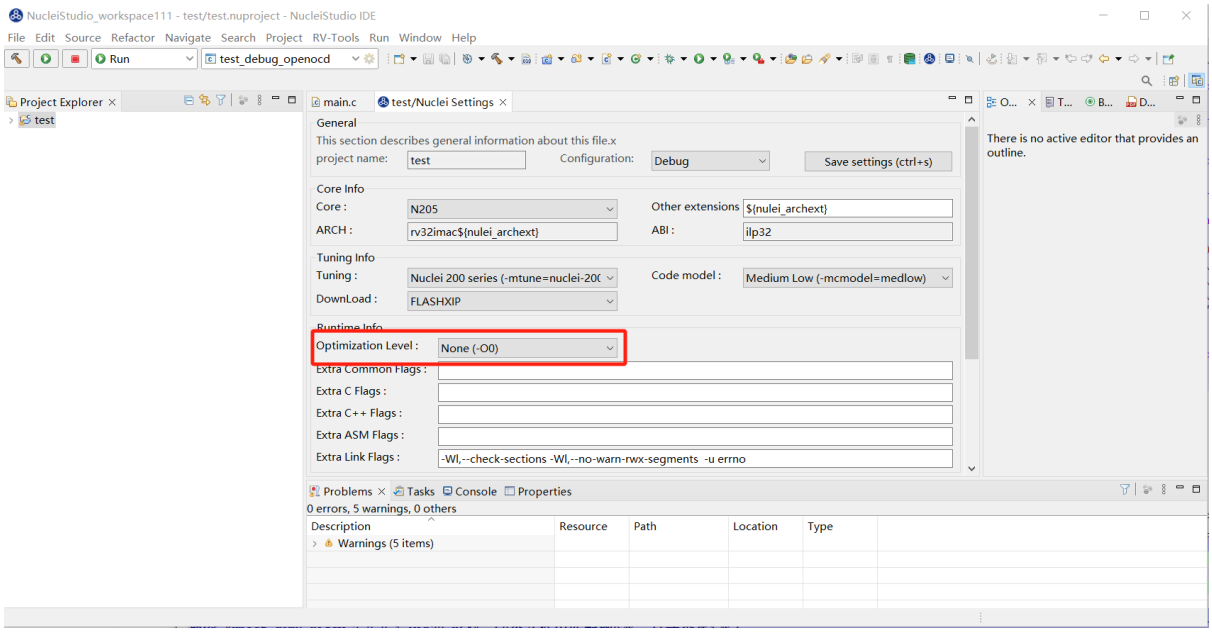
SDK Configuration Tools 更改工程配置

在 Nuclei Studio 可以快速修改工程的设置选项，提供了 SDK Configuration Tools 工具，Nuclei Studio IDE 2022.12 版后，对 SDK Configuration Tools 工具进行了重构，变更为用户体验更好的 Nuclei Settings 菜单。

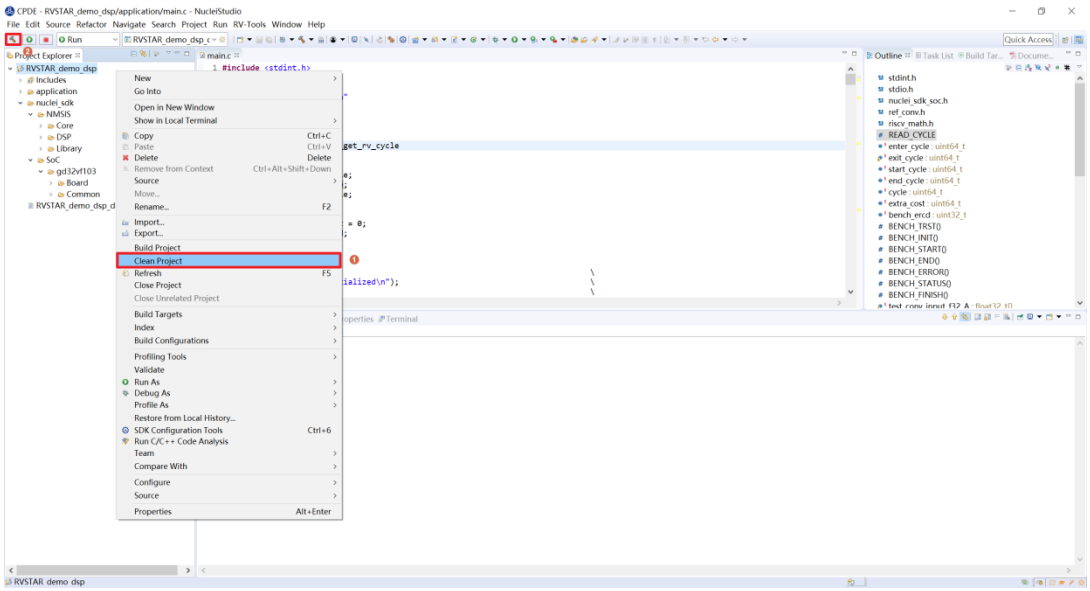
新建好的工程，单击要修改的工程名，右击打开右键菜单，选择 SDK Configuration Tools 打开设置选项工具。



如果要修改编译优化等级，修改 Optimization Level 为 None（-O0），点击 Save 修改选项。



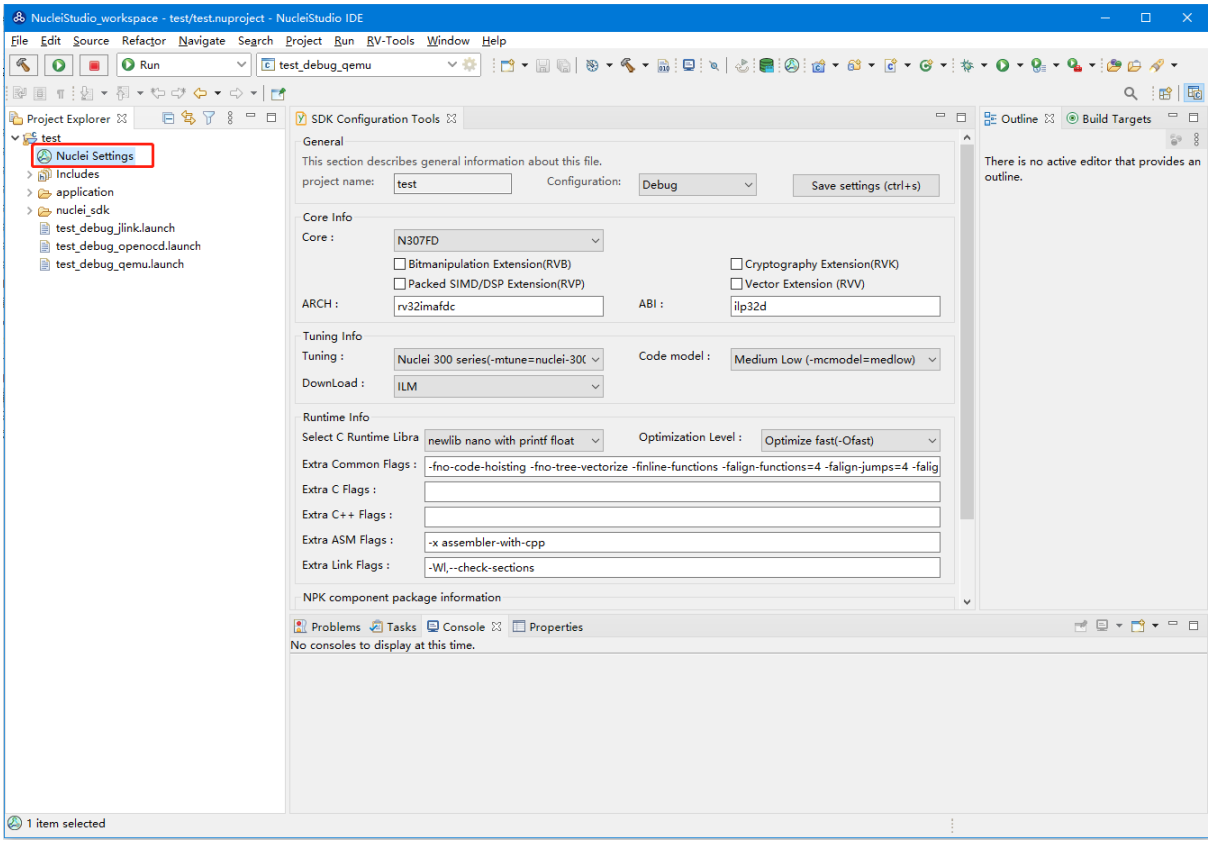
修改成功后在修改后的工程处右击打开右键菜单，选择 clean 清除一下工程，再点击锤子图标编译工程。



Note:

- 注意：SDK Configuration Tools 修改编译配置后对调试配置（Debug Configurations）不生效，请手动修改对应的调试配置。
- 注意：后续版本中，将不再维护 `SDK Configuration Tools` 功能，由 Nuclei Settings 菜单功能替代。

为了更好的用户使用体验，Nuclei Studio IDE 2022.12 版对 `SDK Configuration Tools` 进行了重构，新创建的工程中会多一个 Nuclei Settings 菜单，双击 Nuclei Settings 菜单，将打开工程配置工具其在功能上与 `SDK Configuration Tools` 无异，在 2023.10 版本及其后续版本，SDK Configuration Tools 将直接打开这个 Nuclei Settings 界面。



2.6.3 通过 NPK 导入工具

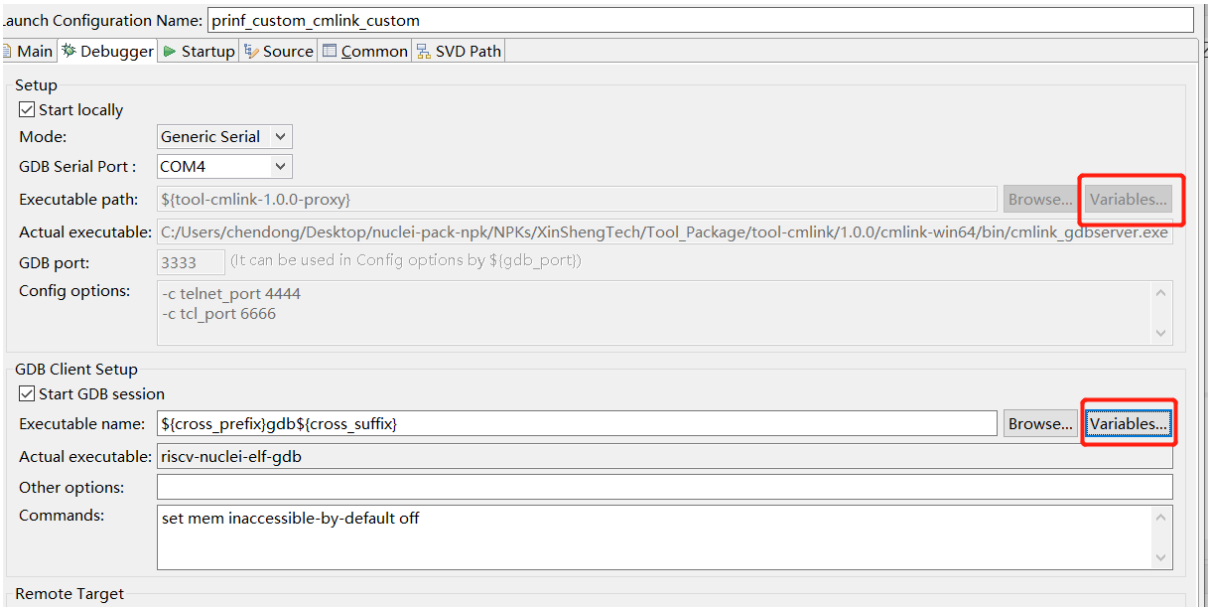
NPK 包除了可以导入 SDK, 还可以方便的导入各种工具包，来扩展 Nuclei Studio 的能力，2022.08 版本的 Nuclei Studio 增加 NPK Tools 的支持，为增加组件包的可扩展性，以及在编译和调试上使用更便捷，增加类型为 tool 的 npk 组件包。tool 组件包可包含 gcc,qemu,cmlink-gdb 等内容，以 zip 包的形式导入到 IDE 去使用。

以 tool-cmlink 包为例，一个工具包中有该工具的执行文件及 npk.yml，开发者在 npk.yml 文件中对该工具做了一些简单的描述，如工具包的开发者、版本、支持的操作系统、可执行文件的路径等，包结构和 npk.yml 内容如下示例。然后将工具包压缩成一个 zip 文件，可以参考导入本地 *NPK* 软件包 (page 73)的内容，将 npk tools 导入到 ide 中，或共享到www.rvmcu.com[6]网站上。

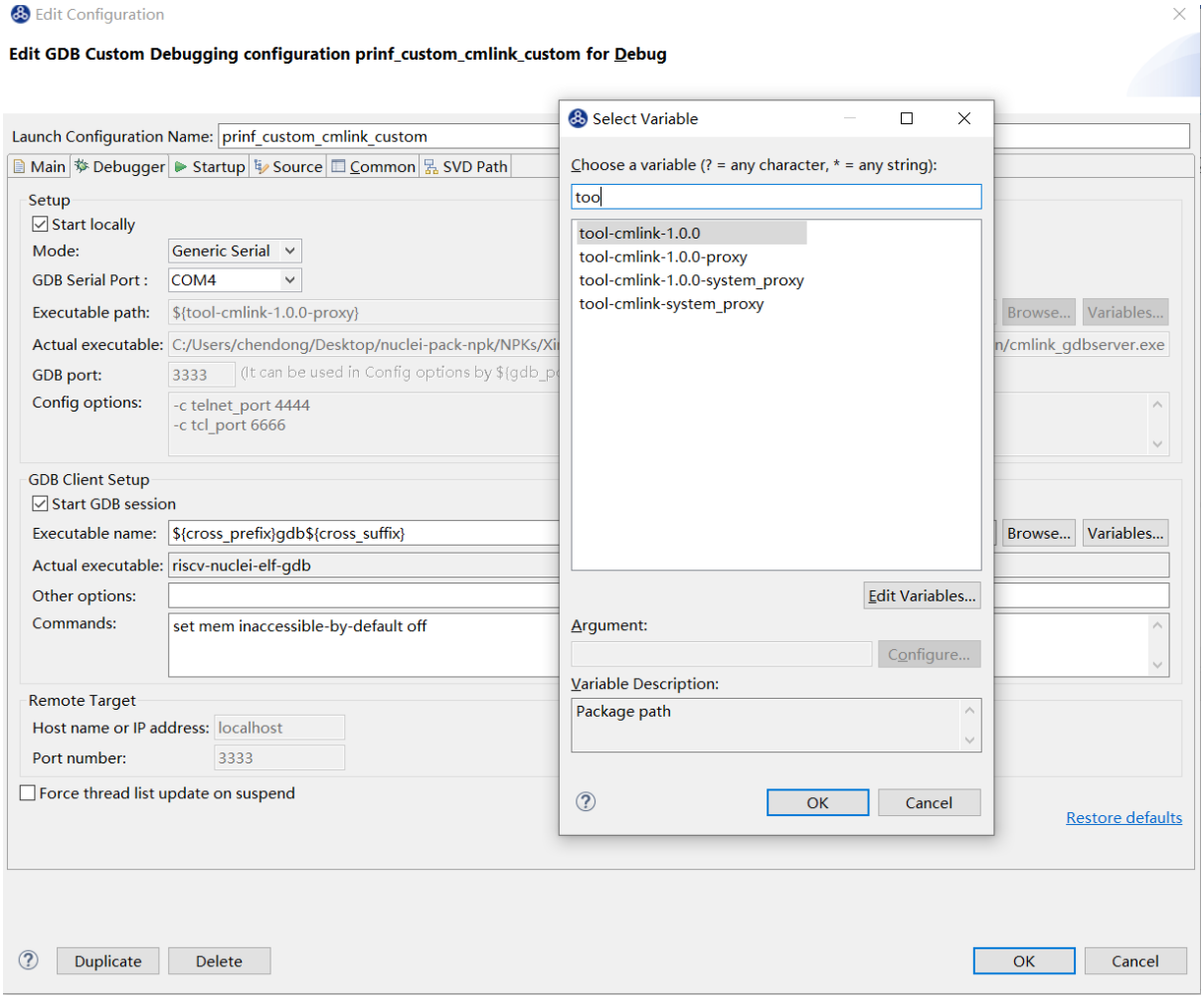
- `bin`
- `bin\cmlink_gdbserver.exe`
- `npk.yml`

在 Nuclei Package Management 管理页中同样可以对 npk tools 进行管理，下载该组件包后，打开任意调试界面，点击 Variables 可以查看到该 npk tools 对应的参数，直接选中对应的参数就可以使用该工具了。

[6] http://www.rvmcu.com



一般我们在 npk tool 中为该组件包扩展变量有 4 个，每个包存在一个包路径，引用为 npk 名称-版本号，例如 `${tool-cmlink-1.0.0}`，其他变量的引用为 npk 名称-版本号-变量名，例如 `${tool-cmlink-1.0.0-proxy}`, `${tool-cmlink-1.0.0-system_proxy}`，当变量的 system 值为 true 时，额外新增一个不带版本号的变量，取最高版本的该变量，例如 `${tool-cmlink-system_proxy}`。



2.7 Nuclei Studio 创建工程

这里以开发板为 Nuclei FPGA Evaluation Board，评估处理器内核为 N307(rv32imafc) 为例，详细介绍 Nuclei Studio 中创建项目的常见方式。

在 Nuclei Studio IDE 创建项目可以有以下几种常见方式：

使用 **NPK** 模板工程自动创建项目：

这是最简单快捷的方式，目前模板项目功能依赖于 Nuclei Studio NPK 功能，在导入对应的 SDK NPK Zip 包，即可在 Nuclei Studio 上进行模板工程的创建。芯来科技提供了 Nuclei SDK、HBird SDK、SoC IP SDK 的 NPK Zip 包，均可导入到 IDE 中进行工程的创建和使用。

从已有项目直接导入创建新项目：

这是最常见的方式，譬如，用户 A 可以将已有项目的文件夹直接进行打包保存，然后进行分享传播，用户 B 可以在另外的电脑上直接导入该项目，从而以此为基础创建新的项目，在此基础上直接使用或者开发修改。

无模板手动创建项目：

这是最繁琐的方式，该方法除了创建项目之外，还需要手动设置各种选项和路径。由于该方式比较繁琐，所以在实际工作中较少使用，但是通过该方式的详细讲解，用户可以详细了解如何配置各中选项和路径。

基于已有的 **Makefile** 创建项目：

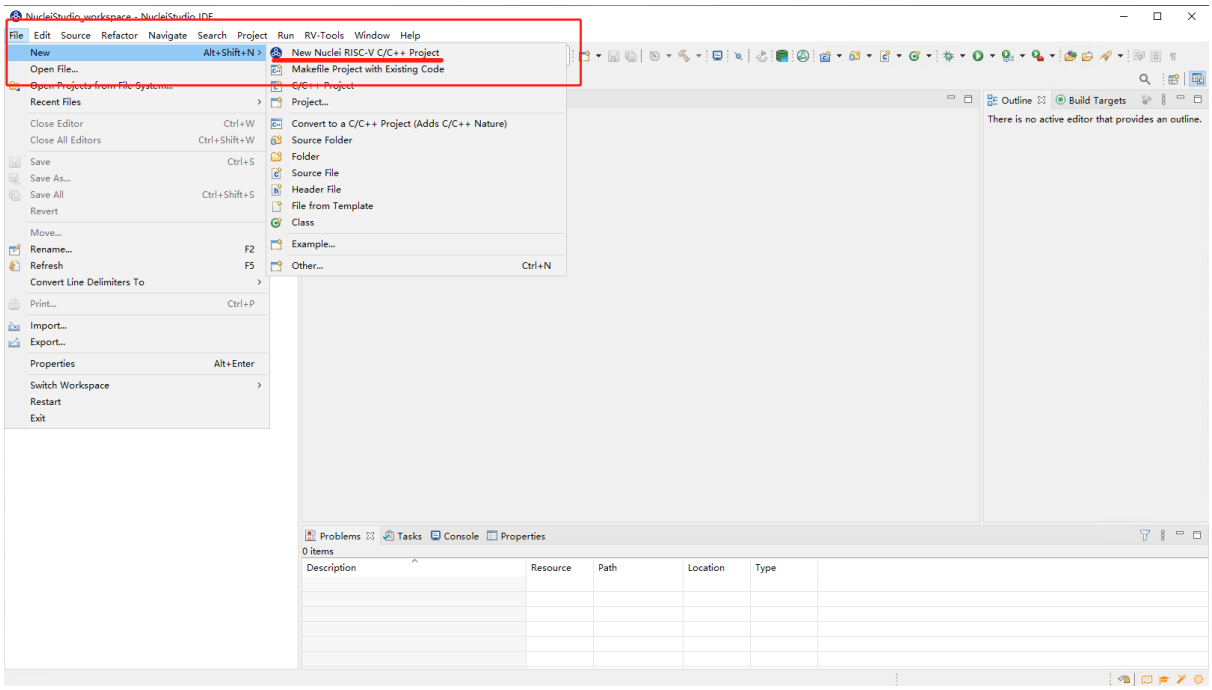
这种方式比较适合于已经采用 Makefile 或者其他编译工具的项目，提供一种在 IDE 中编译工程，清理工程，调试工程的方式。在不修改编译系统的基础上，提供良好的 IDE 调试环境。

下文将对这几种方式分别进行介绍。

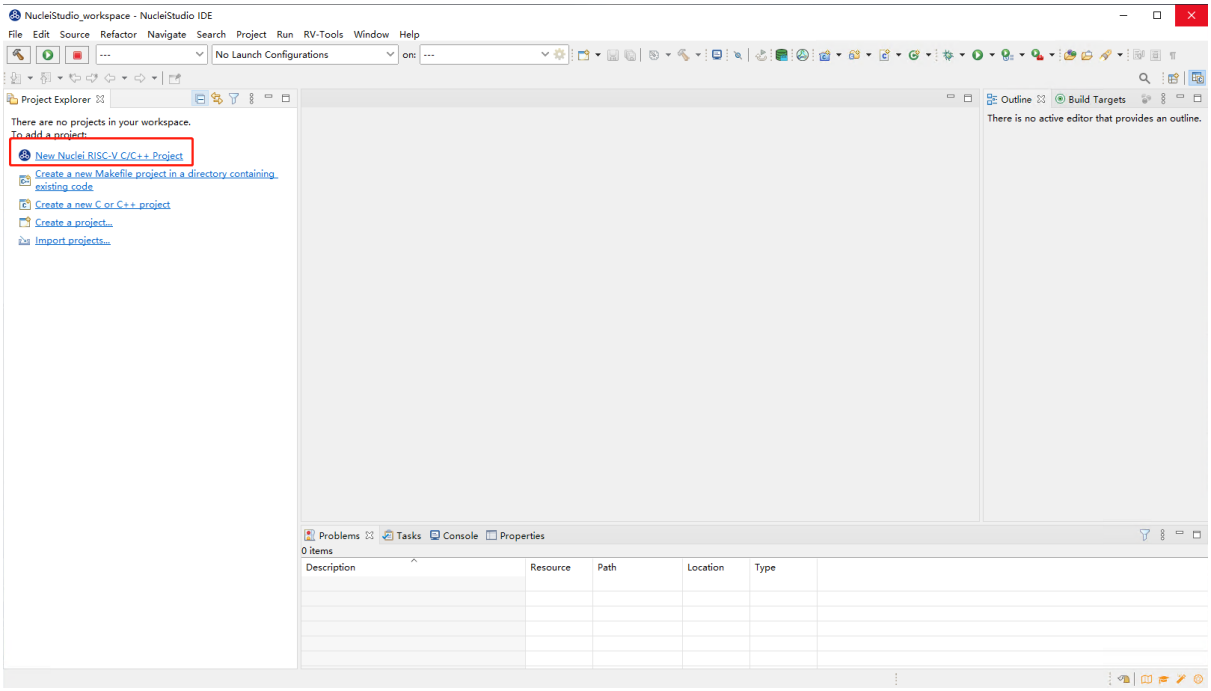
2.7.1 通过 NPK 模板工程自动创建项目

本节将介绍如何使用模板自动创建项目的方式，在 Nuclei Studio IDE 创建一个简单的 Hello World 项目，详细步骤如下。

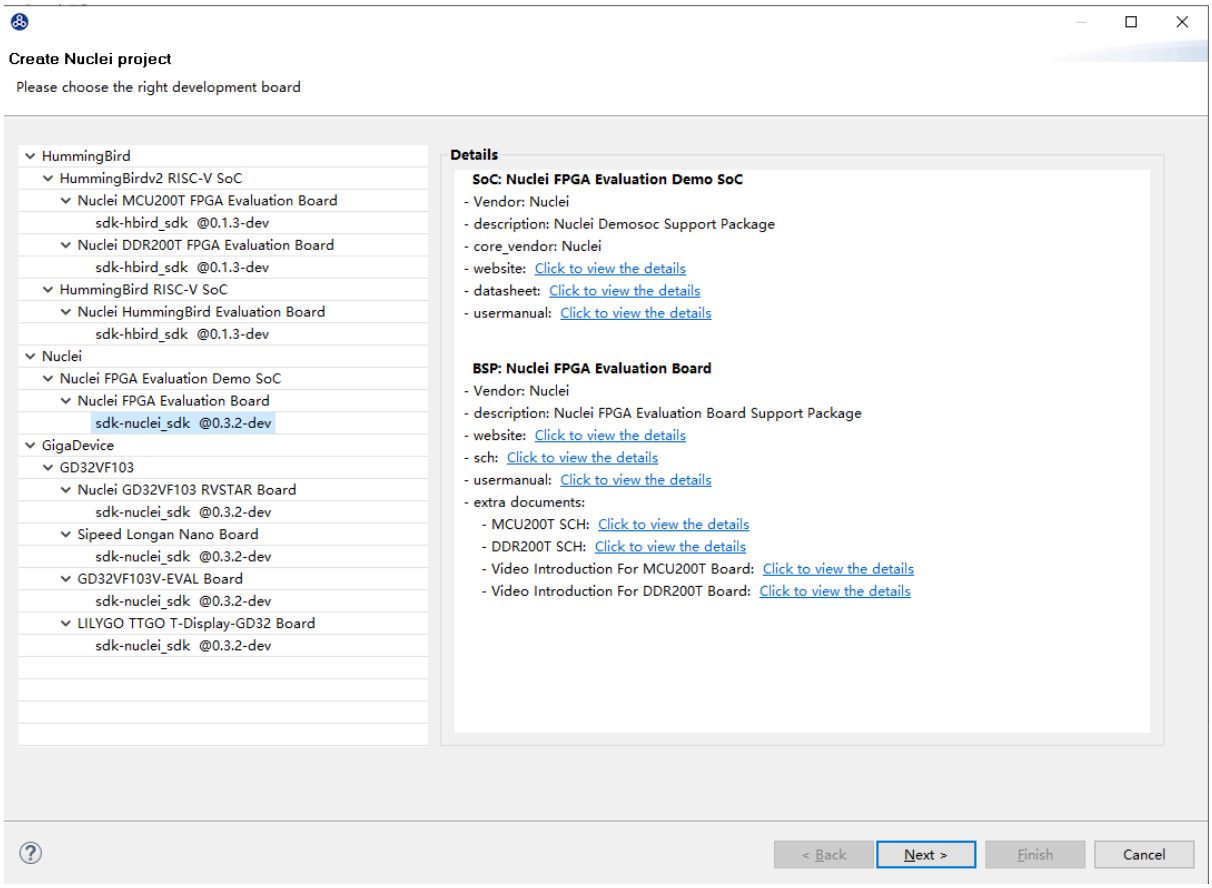
新建一个工程，可以在菜单栏中，选择 `File --> New --> New Nuclei RISC-V C/C++ Project`。



也可以在 Project Explorer 视图中选中 New Nuclei RISC-V C/C++ Project 。



在弹出的窗口中选择项目类型，这里我们在 Nuclei FPGA 板，内核是 N307，SDK 为 nuclei_sdk@0.3.9 版本来做一个测试开发，选对对应的 Board 下的 SDK, 点击 Next 进入下一步。



在弹出的窗口中设定如下参数。

- Project name：项目命名。这里设置为 1_helloworld
- Project Example：选 Helloworld 。

- Toolchains：我们使用 `Nuclei GUN Toolchain` 。

Note: 注意：此页面是通过 NPK Configuration 字段自动解析并生成的页面，不同的 SDK 或者不同的开发板或者不同的例子都可能会有不同的选项页面，请注意。

我们的内核是 N307，所以 `Core` 选择 `N307` 。

蜂鸟开发板支持三种下载模式，以下为每种下载模式的简介，这里我们选择 ILM 模式。

- **ILM**

ILM 下载模式程序将被直接下载在 MCU 的 ILM 中，并从 ILM 开始执行。ILM 由 SRAM 组成，会掉电丢失。

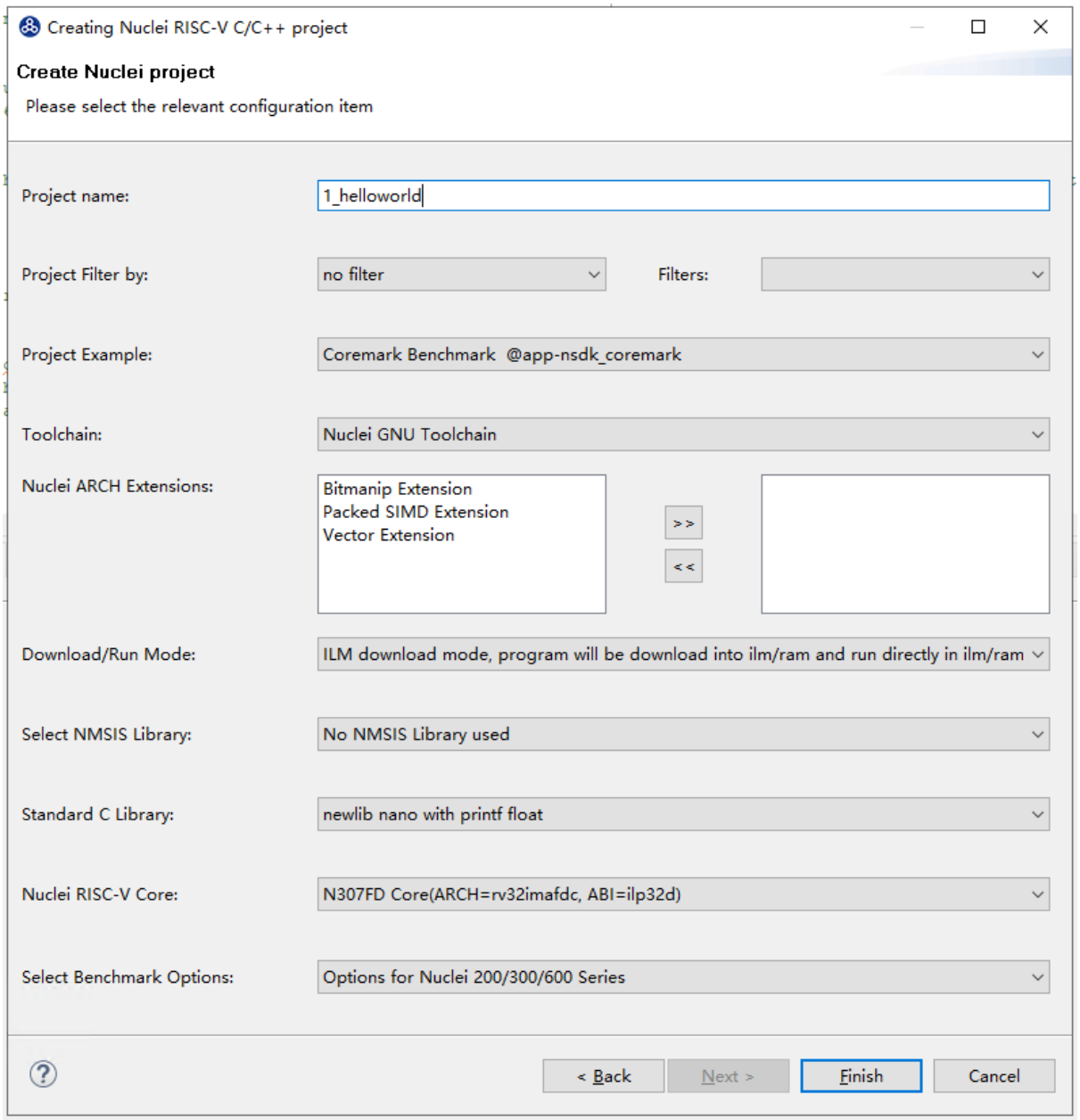
- **FLASH**

FLASH 下载模式程序代码段的物理地址约束 Flash 区间，将代码段的逻辑地址约束在 ILM 的地址区间，意味着程序将被直接下载在 MCU 的 Flash 中，但是上电后要通过引导程序将代码段搬运到 ILM 中，然后从 ILM 中开始执行。程序被烧写在 Flash 中，不会掉电丢失。

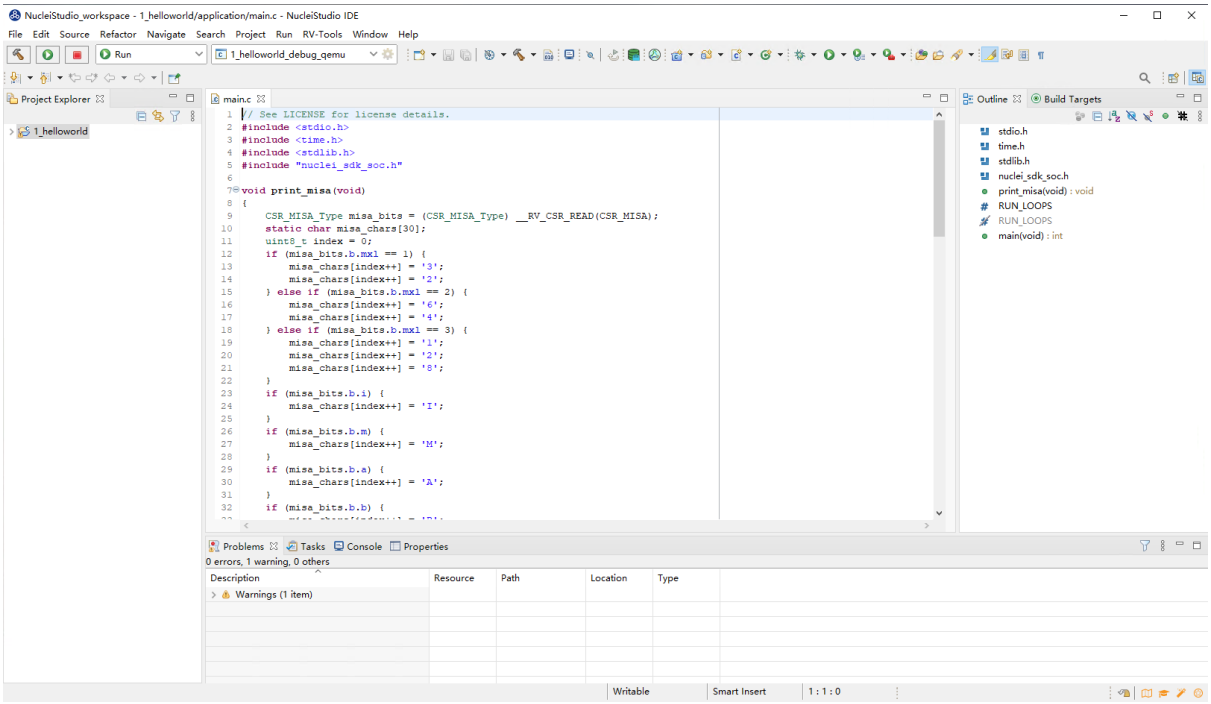
- **FLASHXIP**

FLASHXIP 下载模式程序代码段约束 Flash 区间，意味着程序将被直接下载在 MCU 的 Flash 中，并直接从 Flash 开始执行。程序被烧写在 Flash 中，不会掉电丢失。

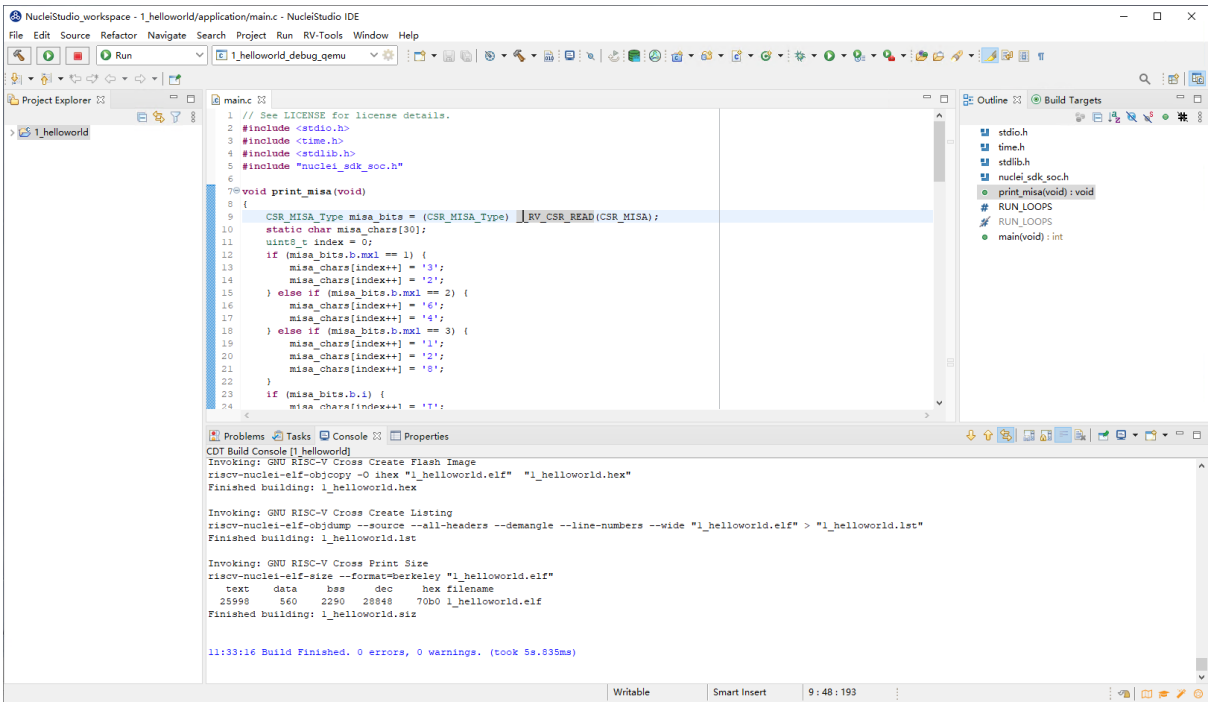
其他各项可以按需进行配置，点击 `Finish` 完成工程创建。



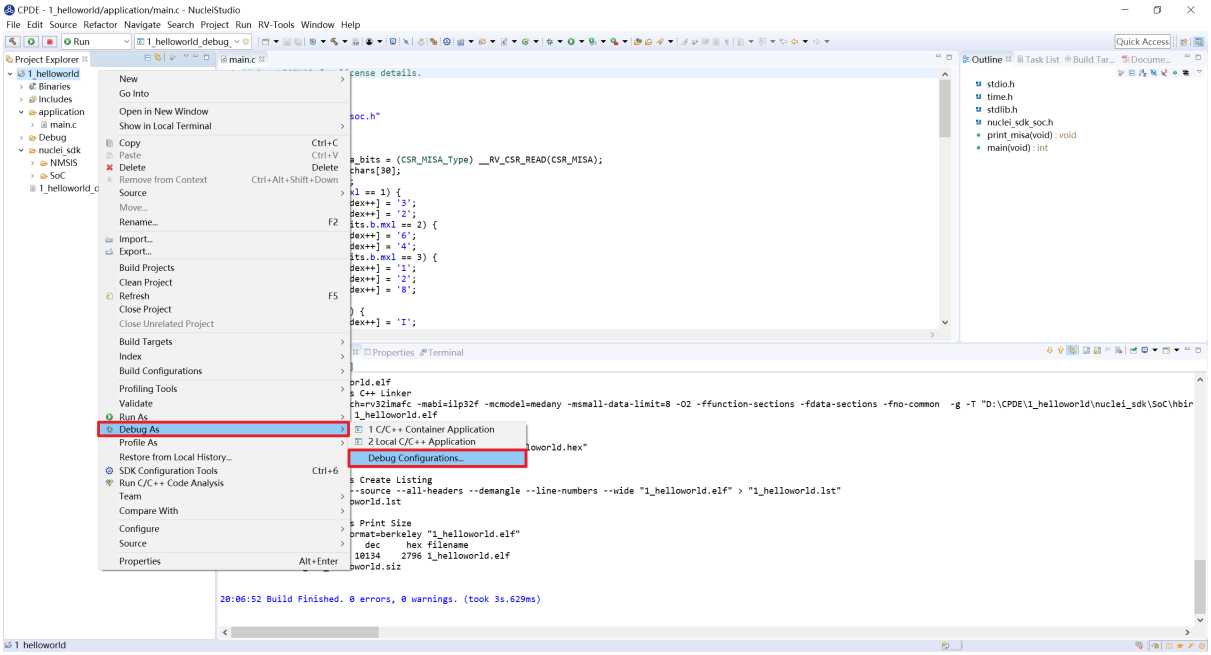
使用模板自动创建 Hello World 项目已经完成。



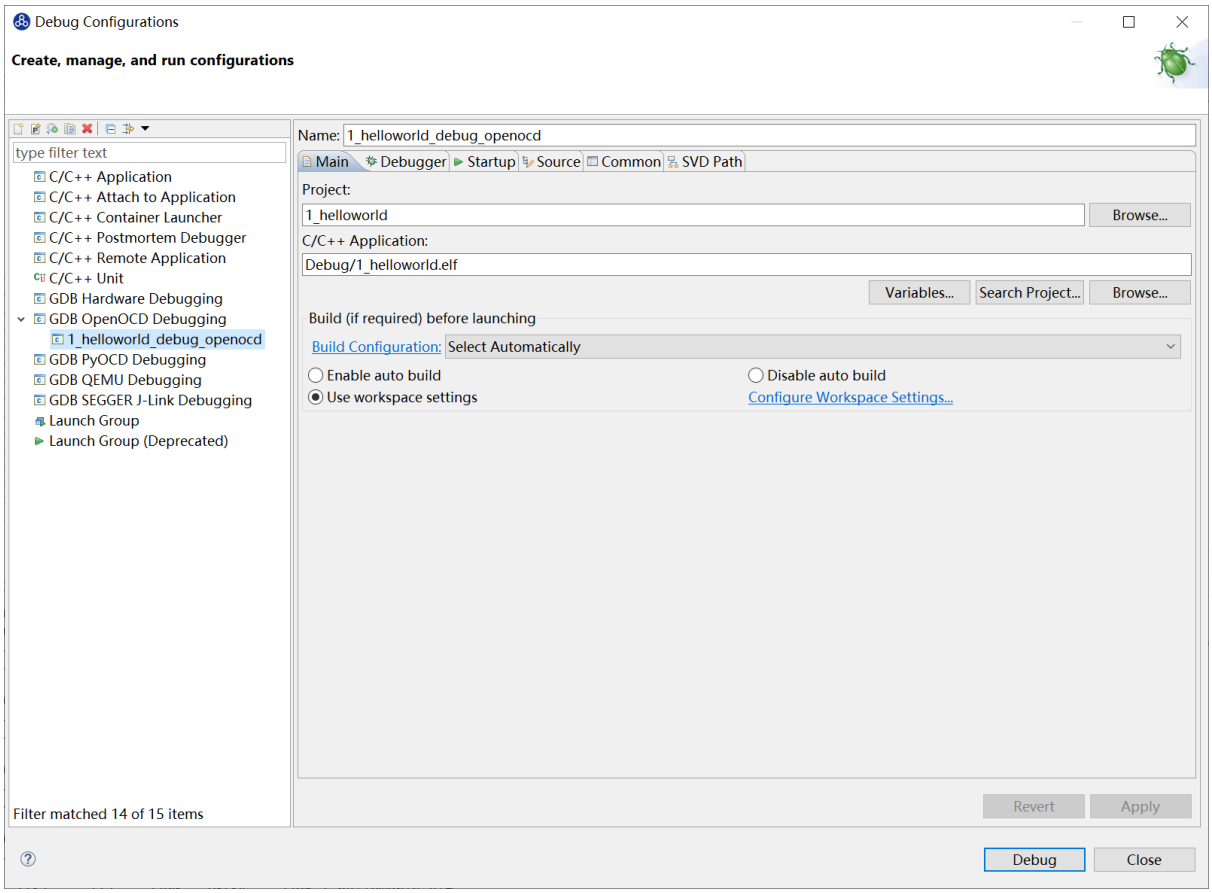
用户可以直接使用菜单栏 Project—> Build Project 或  按钮，来对该项目进行编译。



在 Hello World 项目自动生成过程中，其对应的 OpenOCD 配置已经同步完成。在项目编译完毕后，用户可以右键点击项目列表 Hello World，点击 Debug As —>Debug Configurations 开启调试配置面板进行查看。Debug 与 Run 使用相同的配置文件，所以也可通过 Run As —>Run Configurations 打开。



用于调试使用的配置文件 Hello World_Debug_OpenOCD 已经自动生成。关于使用芯来蜂鸟调试器结合 OpenOCD 进行下载和调试的方法，可以查看使用蜂鸟调试器结合 *OpenOCD* 调试运行项目 (page 137) 进行详细了解。

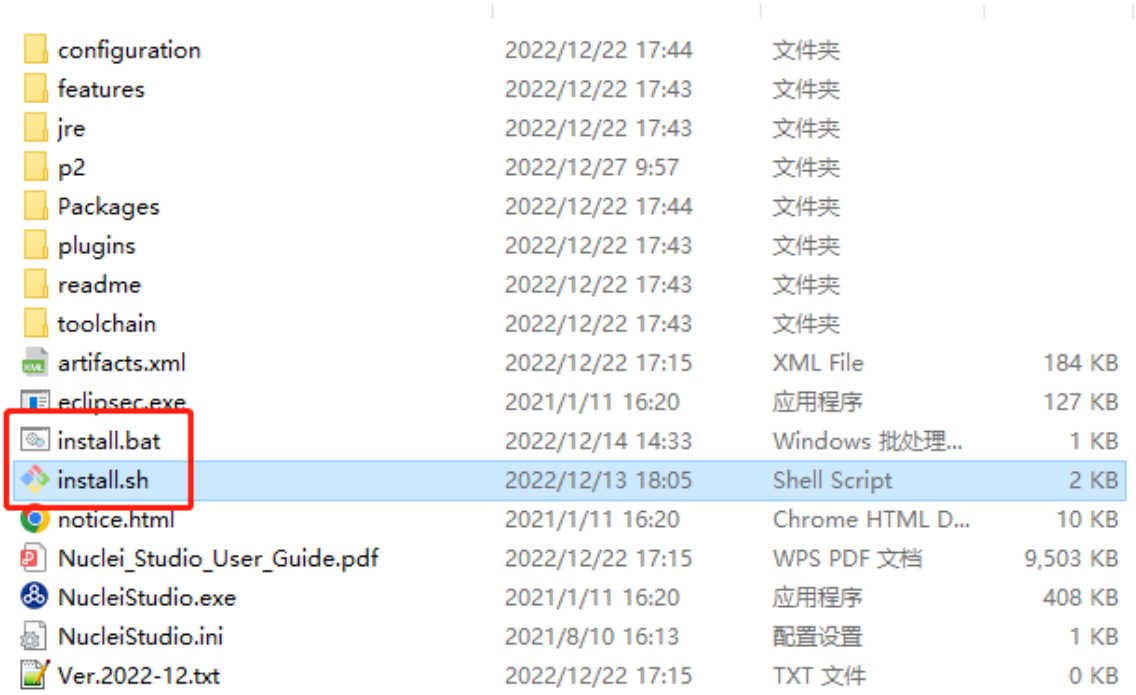


2.7.2 通过应用关联文件导入工程

本节将介绍如何通过 Nuclei Studio IDE 的关联文件，将一个 Hello World 项目导入到 IDE 中。Nuclei Studio IDE 2022.12 版中，新增了应用文件关联的支持，可以将 IDE 的启动路径注册到系统注册表中，然后通过特定的文件，可以实现 IDE 的启动、工程导入等功能，极大的方便了 Nuclei Studio IDE 用户在使用过程中，对工程的分享和快速导入。

将 Nuclei Studio IDE 写入到注册表

下载 Nuclei Studio IDE 2022.12 版，在安装包中多了两个文件 `install.bat/install.sh`，在 windows 系统下，双击 `install.bat`，因为这里需要写入注册表，所以需要一个用户授权，授权后安装成功；在 linux 系统下，需要在 shell 命令下执行 `install.sh` 文件。



`install.sh` 文件在运行后，有一个用户授权的界面，同意授权。

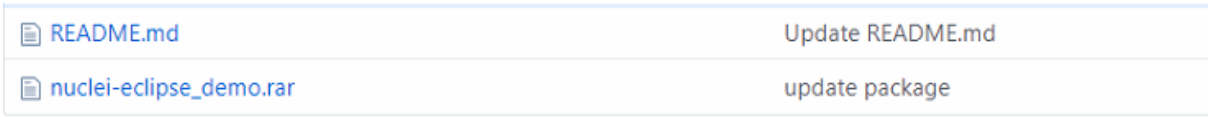
通过应用关联文件导入工程

Nuclei Studio IDE 2022.12 版创建工程 `test`，在工程中会有一应用关联文件 `test.nuproject`，如果 ide 的启动路径已写入注册表，双点 `test.nuproject` 文件，系统会自动启动 Nuclei Studio IDE 并将 test 工程导入到 IDE 中。

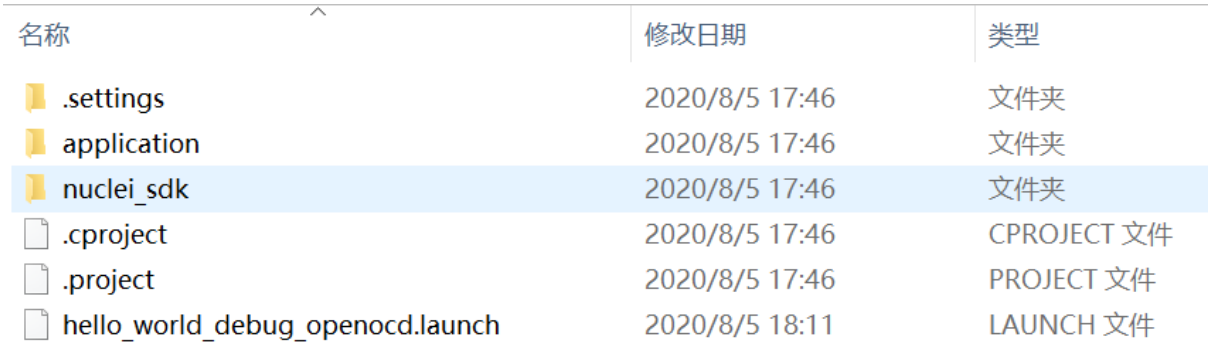
2.7.3 从已有项目直接导入创建新项目

本节将介绍如何使用 IDE 从已有项目直接导入创建新项目，本文以 N307 的项目包为例进行导入，项目包存放在（https://github.com/riscv-mcu/Nuclei-Studio_IDE-Project-Package）。如需其它项目包请与芯来科技联系。

在基于 Windows 的 Nuclei Studio IDE 开发环境中，如果用户使用 无模板手动创建工程，也需要加载此项目包中的 nuclei-sdk 文件夹，相关内容会在无模板手动创建项目 (page 100) 中具体介绍。



将 nuclei-eclipse_demo.rar 压缩包下载解压后，内容分别为：

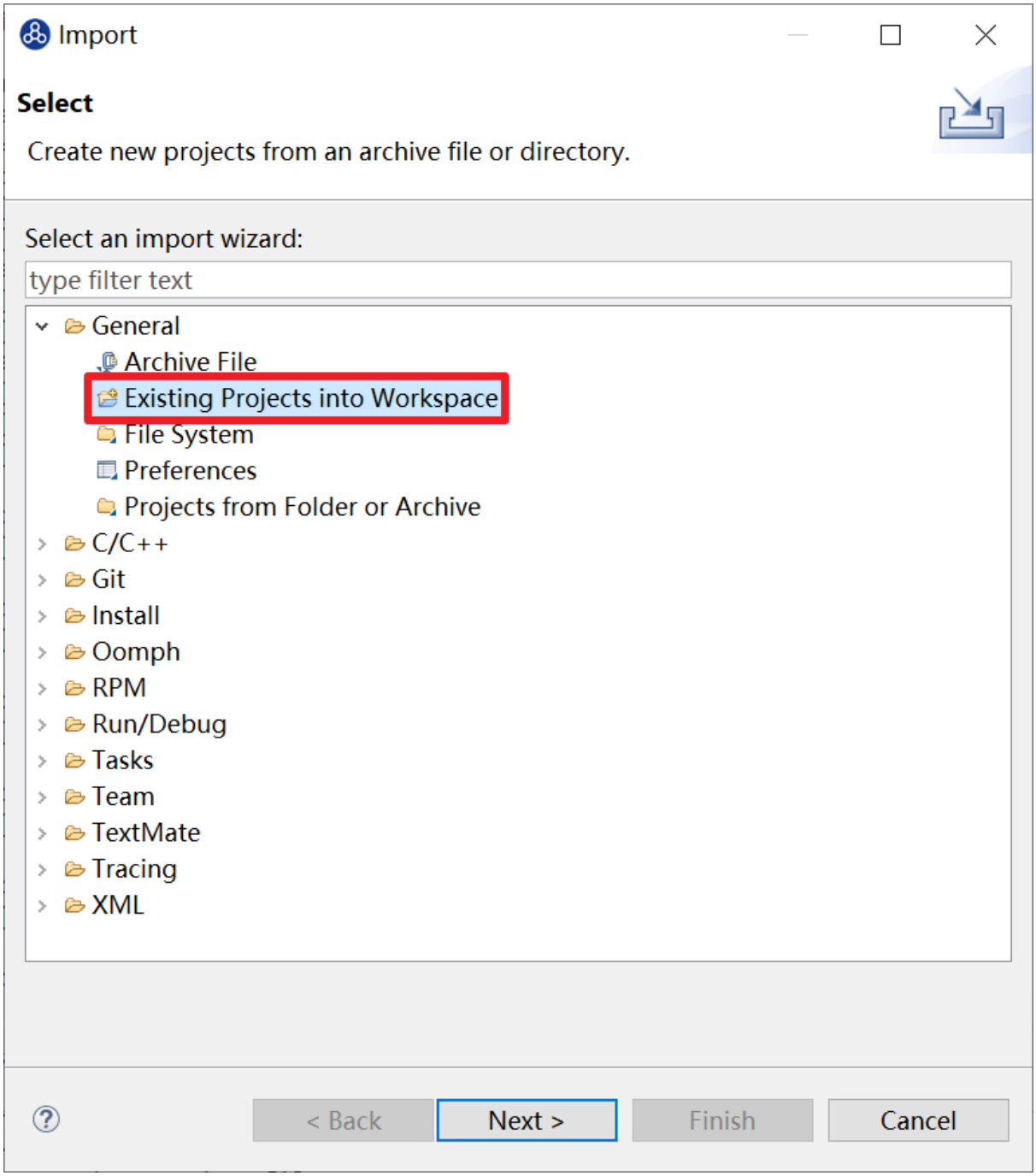


- 项目包的描述文件 `.setting`， `.project` 和 `.cproject`
- 项目包的 Debug 设置文件 `*.launch`
- nuclei_sdk 文件夹

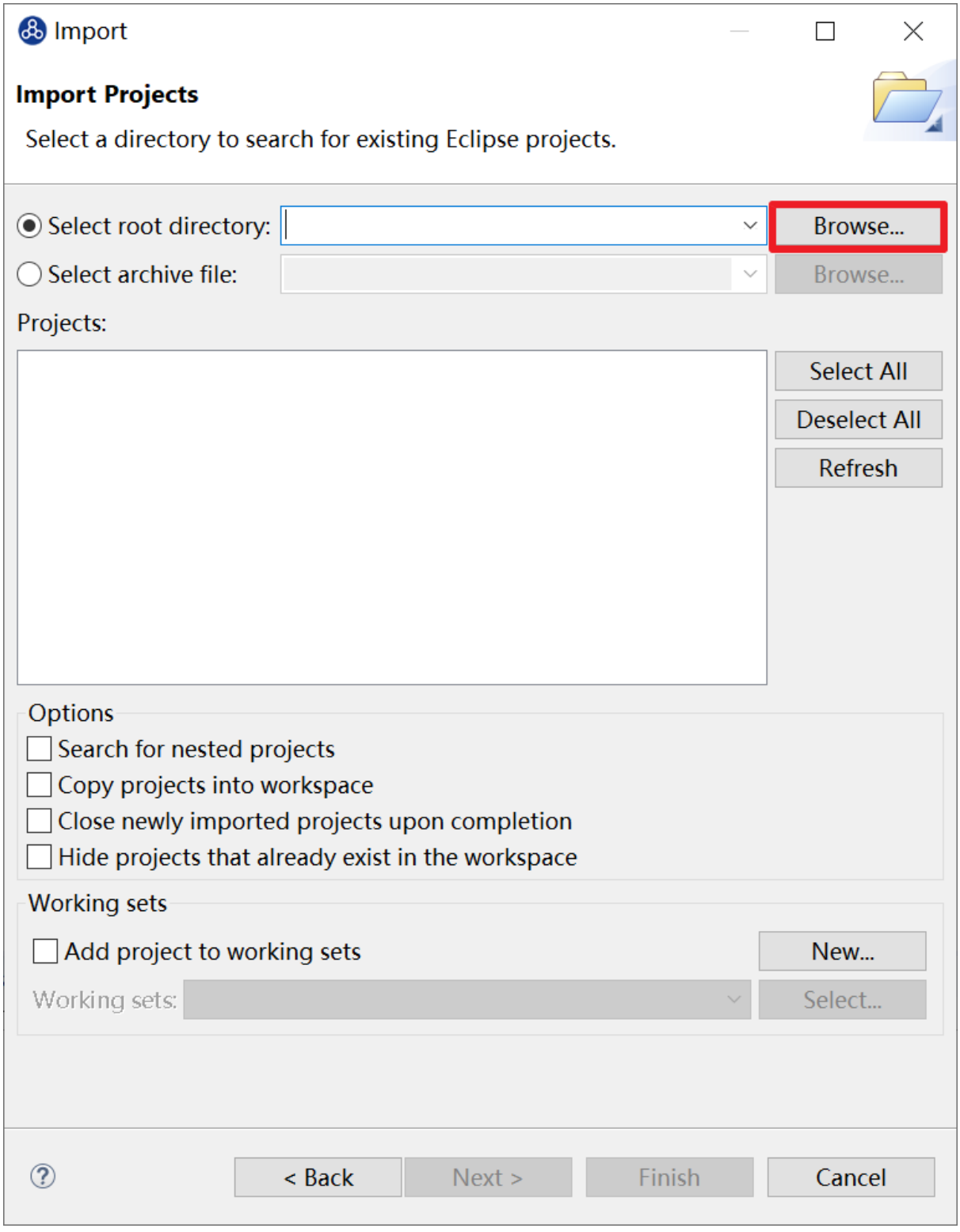
  该文件夹下存放部分 SDK 源代码。
- application 文件夹

  此文件夹包含 hello_world 样例程序的 main 函数源代码。

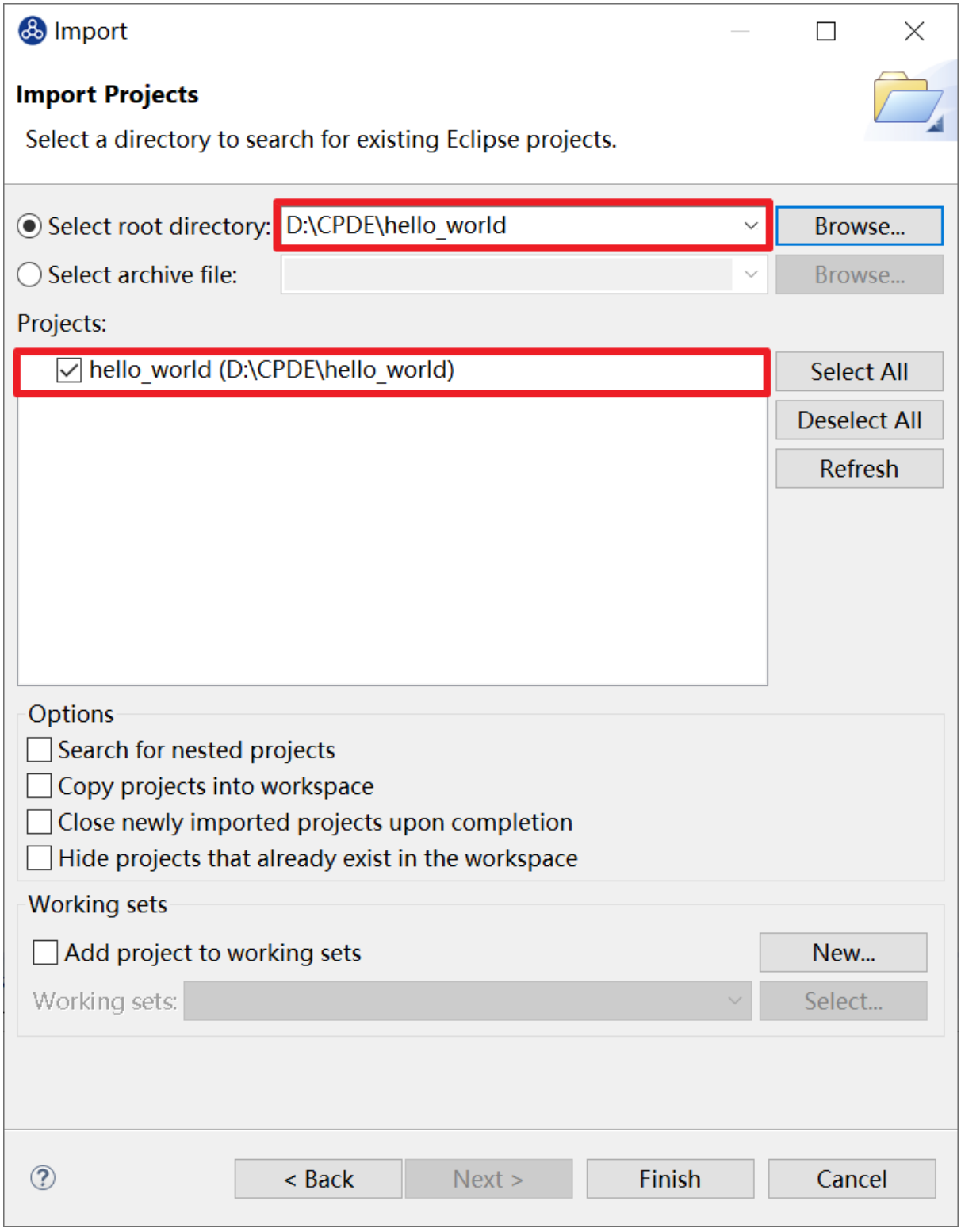
  下一步导入下载好的项目包，导入步骤如下：
- 在菜单栏中选择 `File—>import` 。
- 如图所示，选择 `Existing Project into WorkSpace` 后，点击 `Next` 。



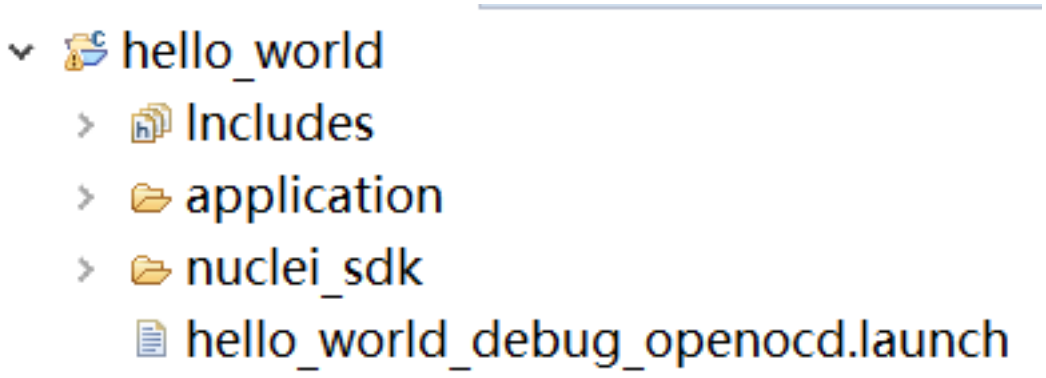
- 点击 Browse，选择需要导入的项目路径，如图所示。



- 需要的导入的项目成功被 IDE 识别，点击 `Finish` 。



- 在 IDE 的项目资源管理器中显示导入项目的目录结构如下图所示。已有项目默认为 N307 的编译选项，Nuclei SDK 仅包含 helloworld 使用到的文件。需要更多的 Nuclei SDK 源码请访问 Github（https://github.com/riscv-mcu/hbird-sdk）获取源码。



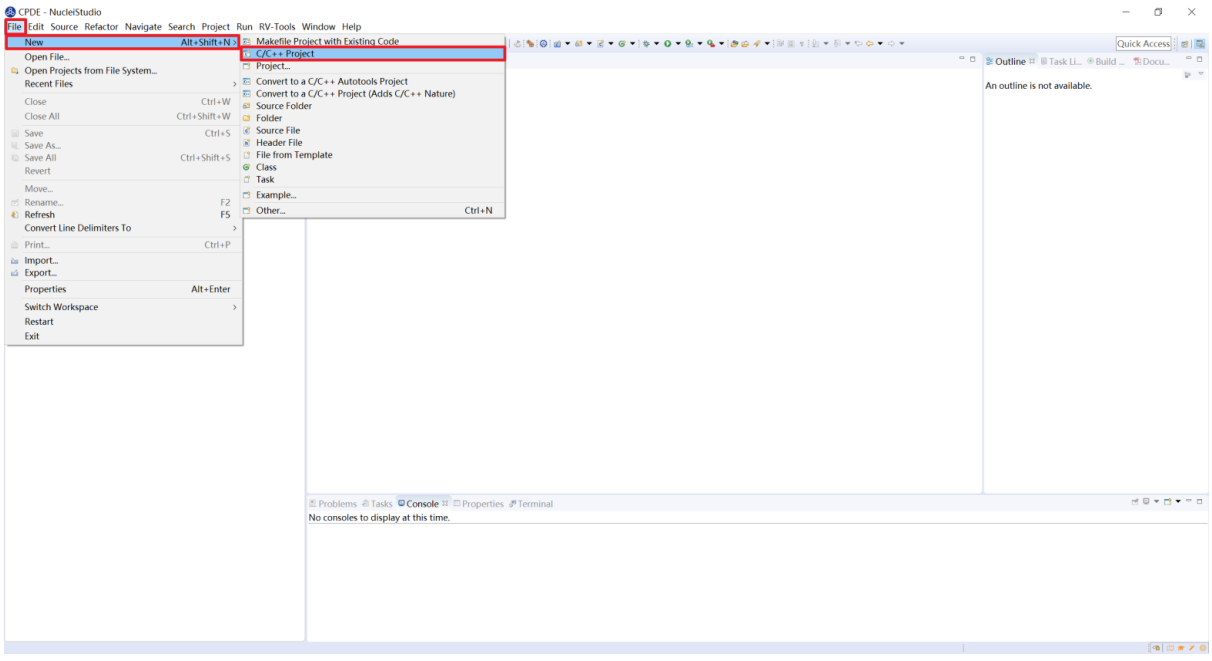
2.7.4 无模板手动创建项目

本节将介绍如何使用手动方式在 Nuclei Studio IDE 创建一个用户自定义的 Hello World 项目。开发板为 Nuclei FPGA Evaluation Board，内核为 N307。该方法除了创建项目之外，还需要手动设置各种选项和路径，详细步骤如下。

Note: 不建议使用，建议使用 NPK 模板的方式创建工程

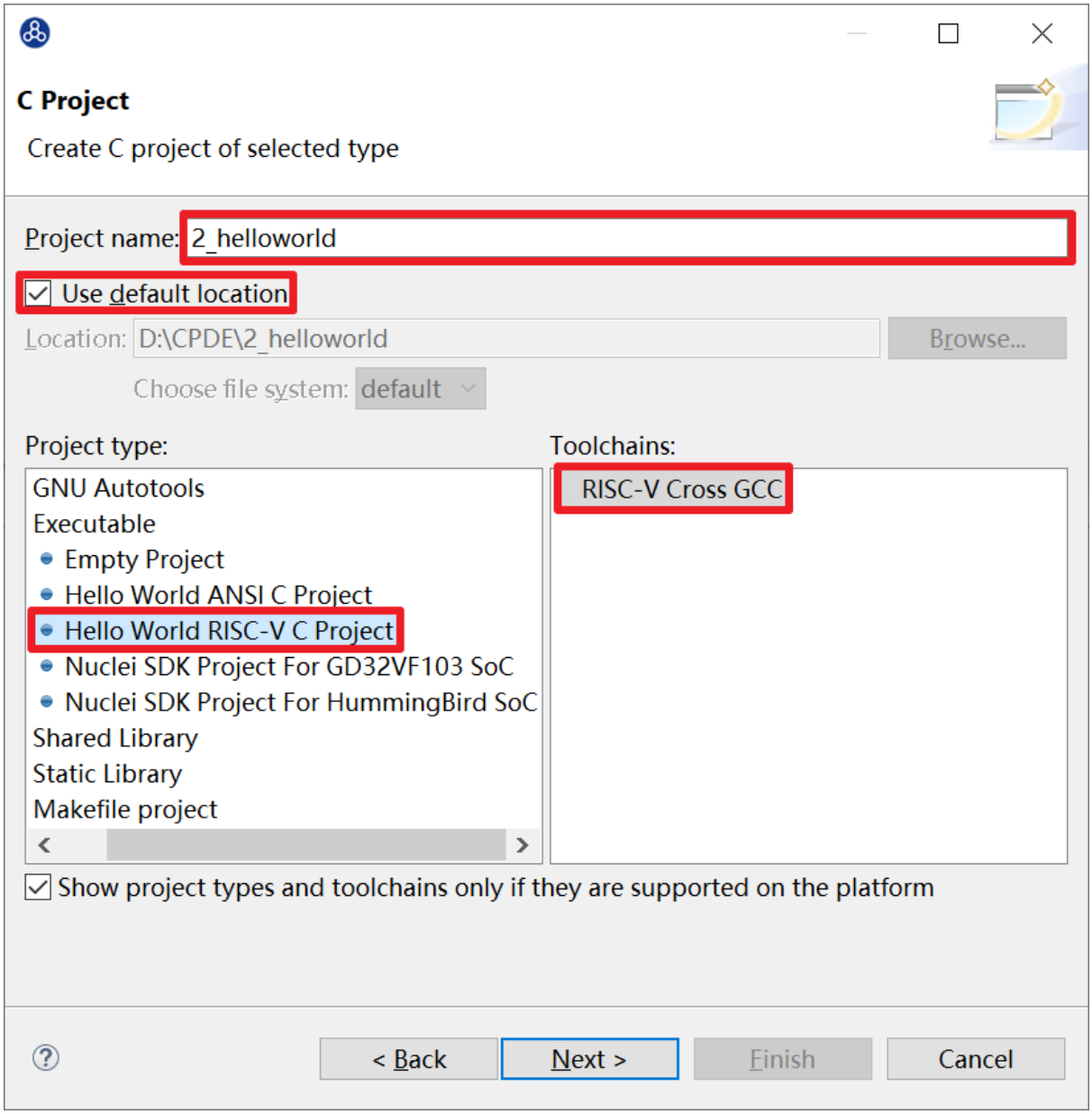
手动创建项目

在 Nuclei Studio 的主菜单栏中，依次选择 `File—> New —> C/C++ Project` 。

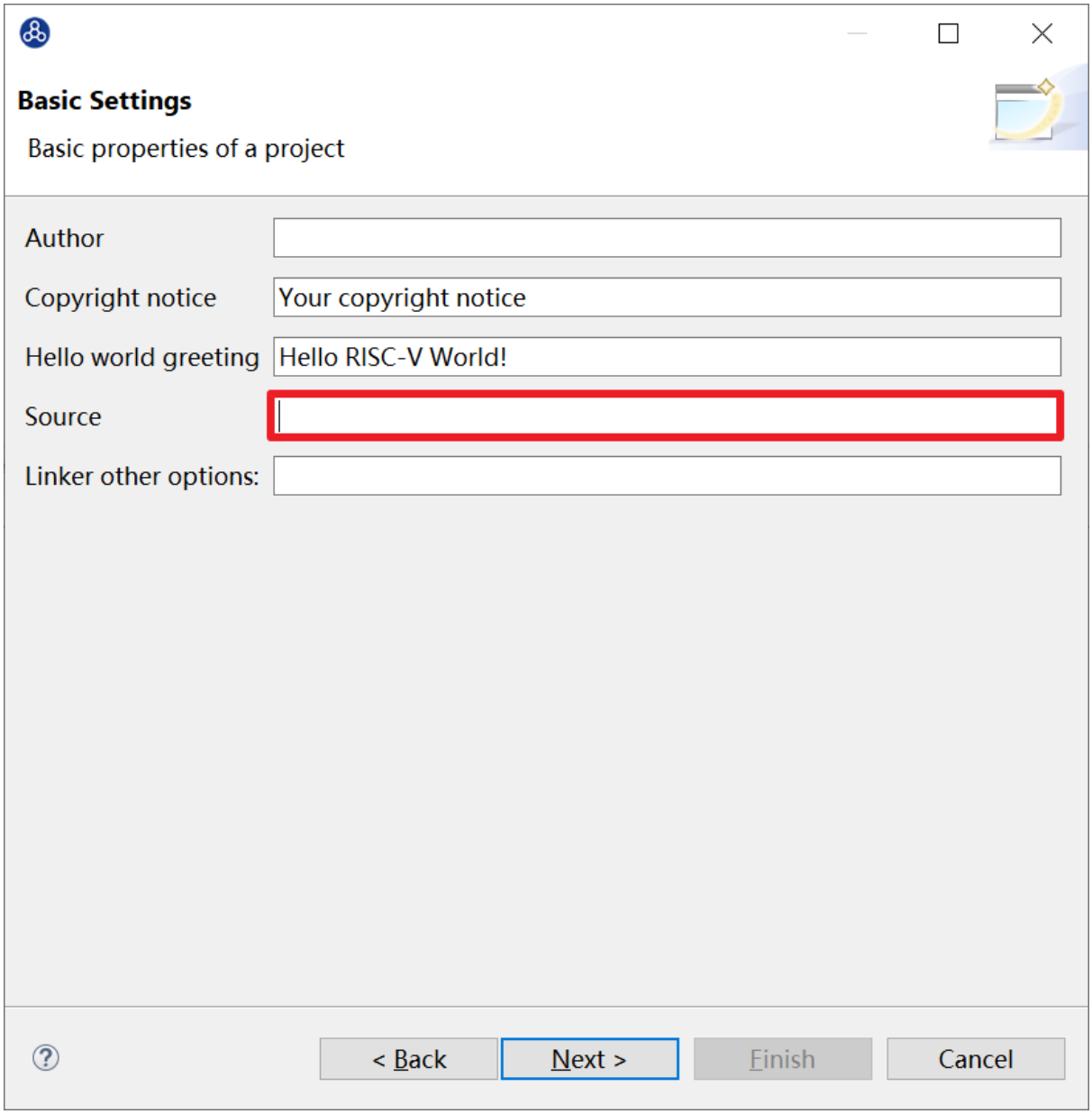


然后在弹出的窗口中设定如下参数。

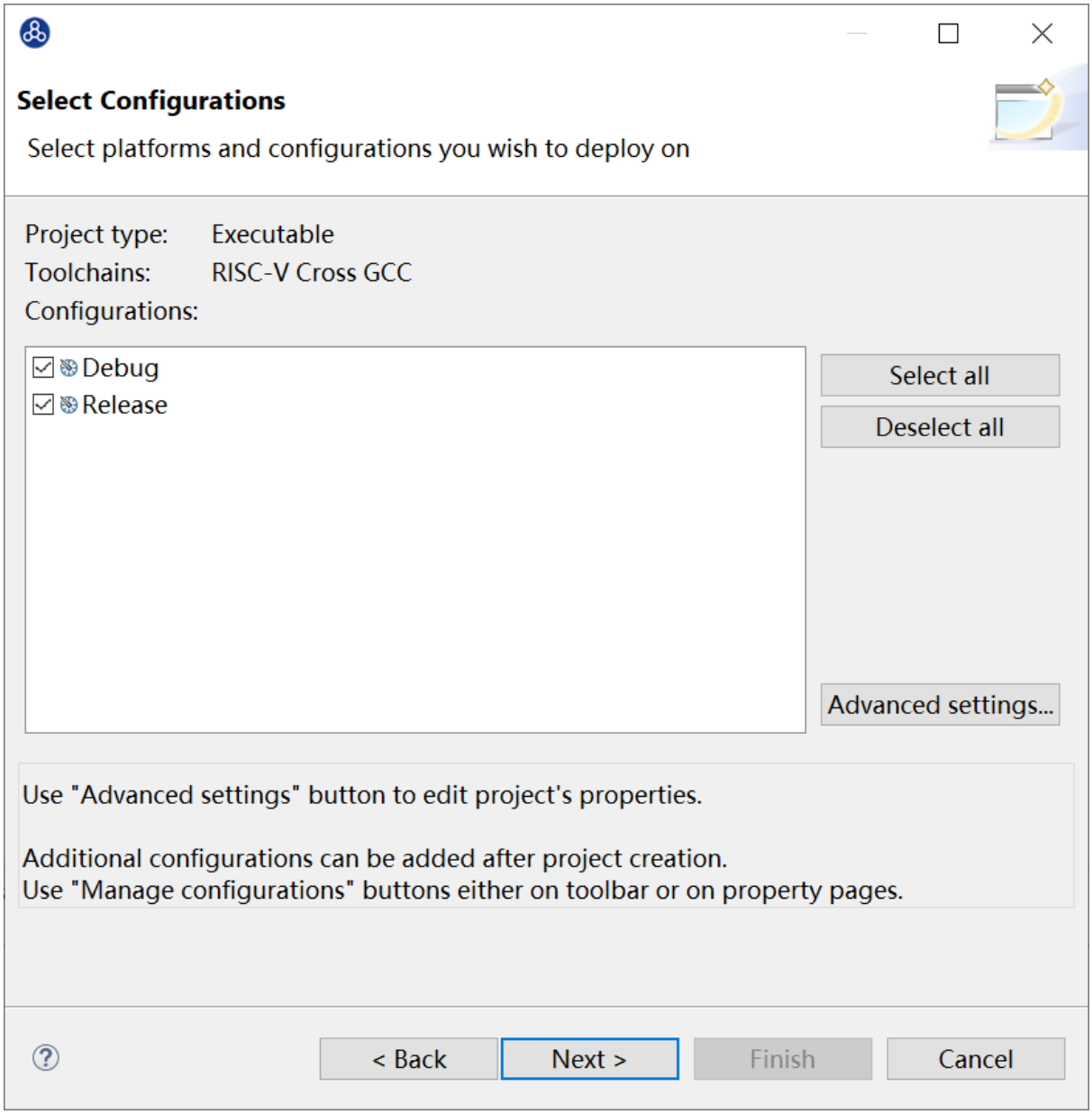
- Project name：项目命名。
- Use default location：如果勾选了此选项，则会使用默认 Workspace 文件夹存放此项目。
- Project type：选择 `Hello World RISC-V C Project` 。



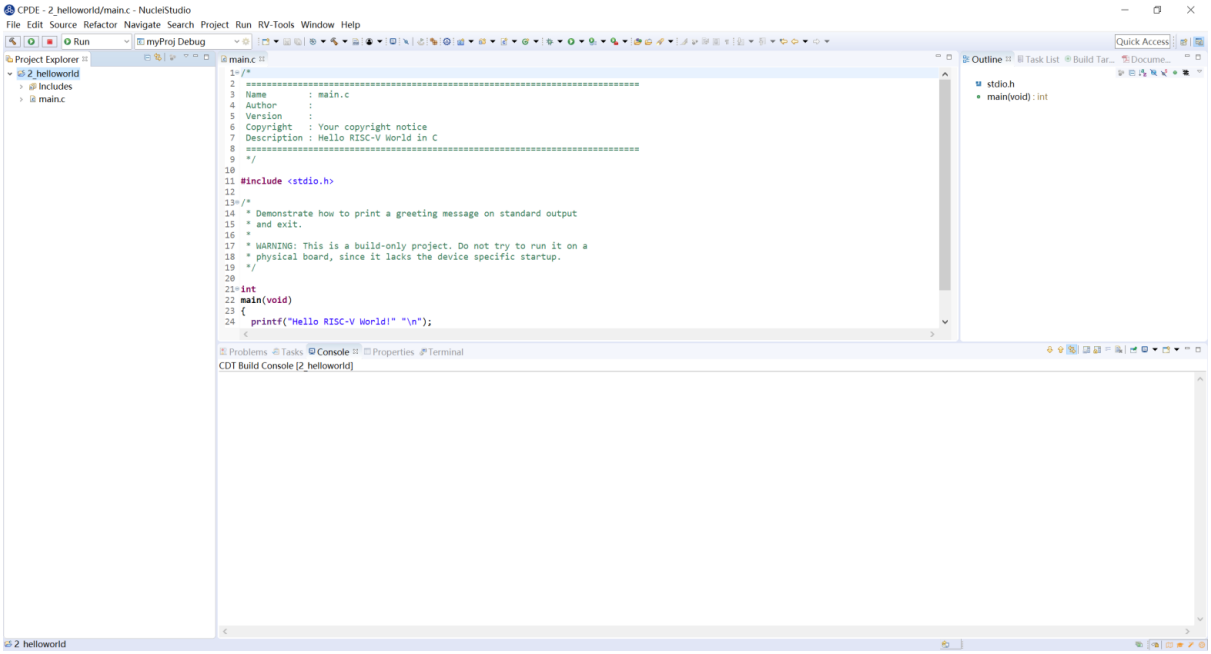
然后点击 Next 进入下一步，在弹出的窗口中设置 Hello World 项目的基本信息。确保 `Source` 选项内容为空，直接单击 `Next` 进入下一步。



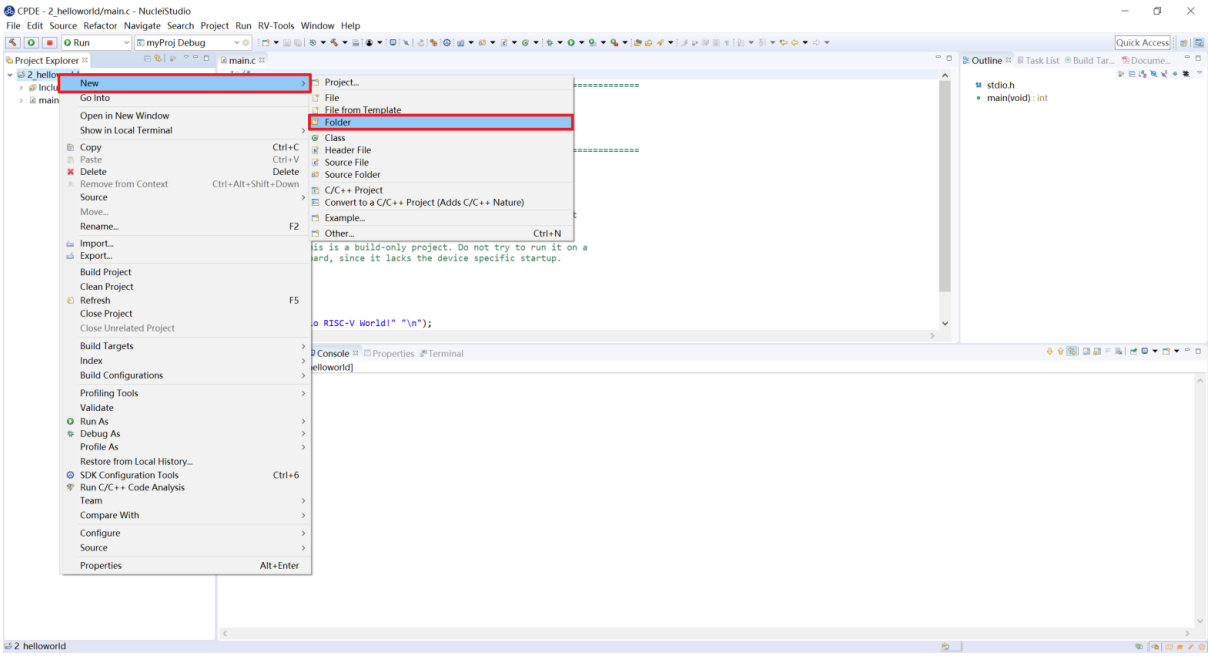
在弹出的窗口中设置项目的调试或者发布属性。该步骤可以使用默认信息不做任何修改，直接单击 Next 进入下一步。



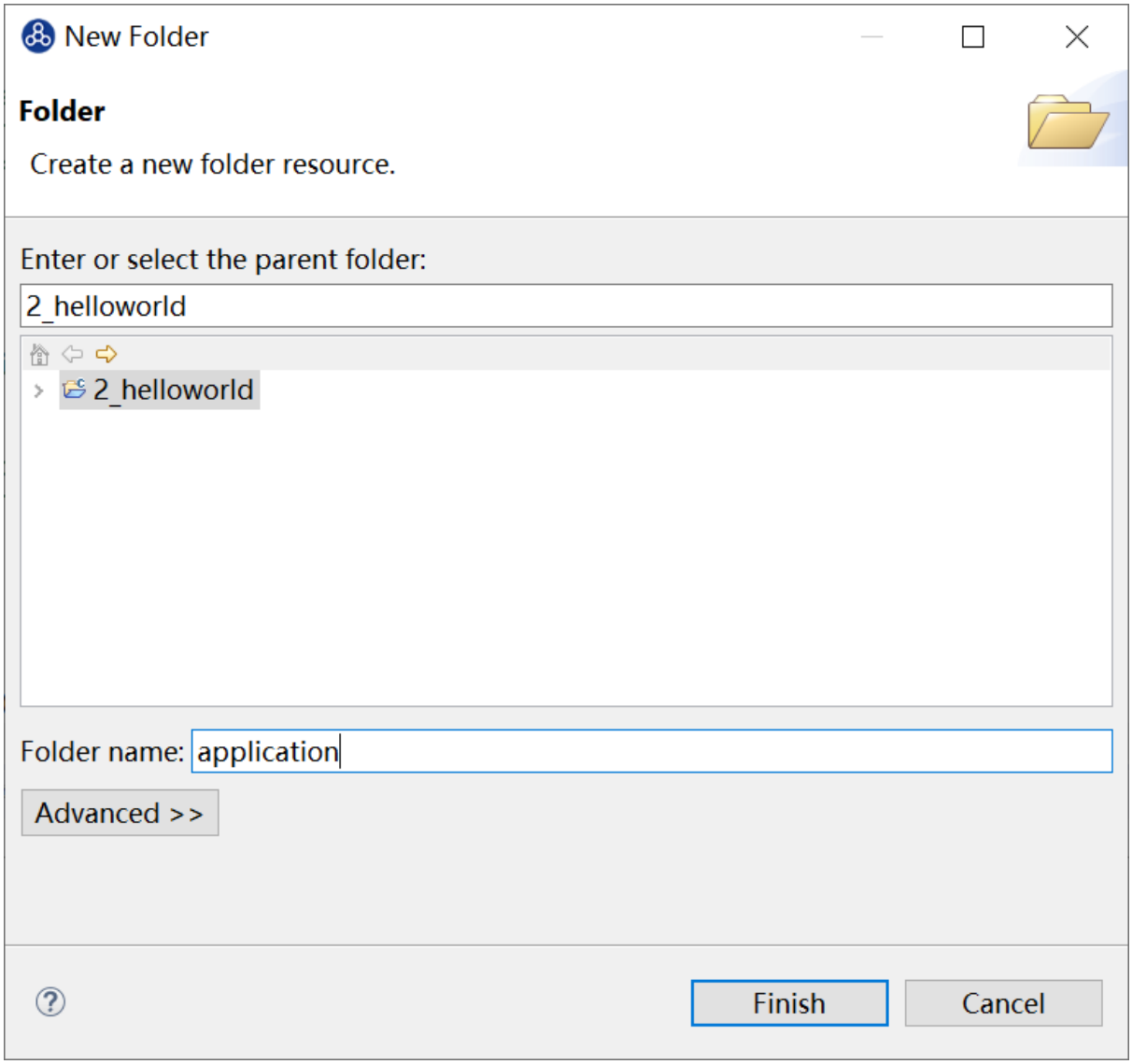
在弹出的窗口中设置项目所使用的 RISC-V 工具链。此处不要配置，直接选择 `Finish`，至此便完成了 HelloWorld 项目的创建。



创建完成，Hello World 项目的展示界面如下。



新建一个 application 文件夹。在工程处右击选择 `New —> Folder`，输入 application，点击 `Finish` 完成新建工程。将 main.c 拖入 application 文件夹完成文件分类。

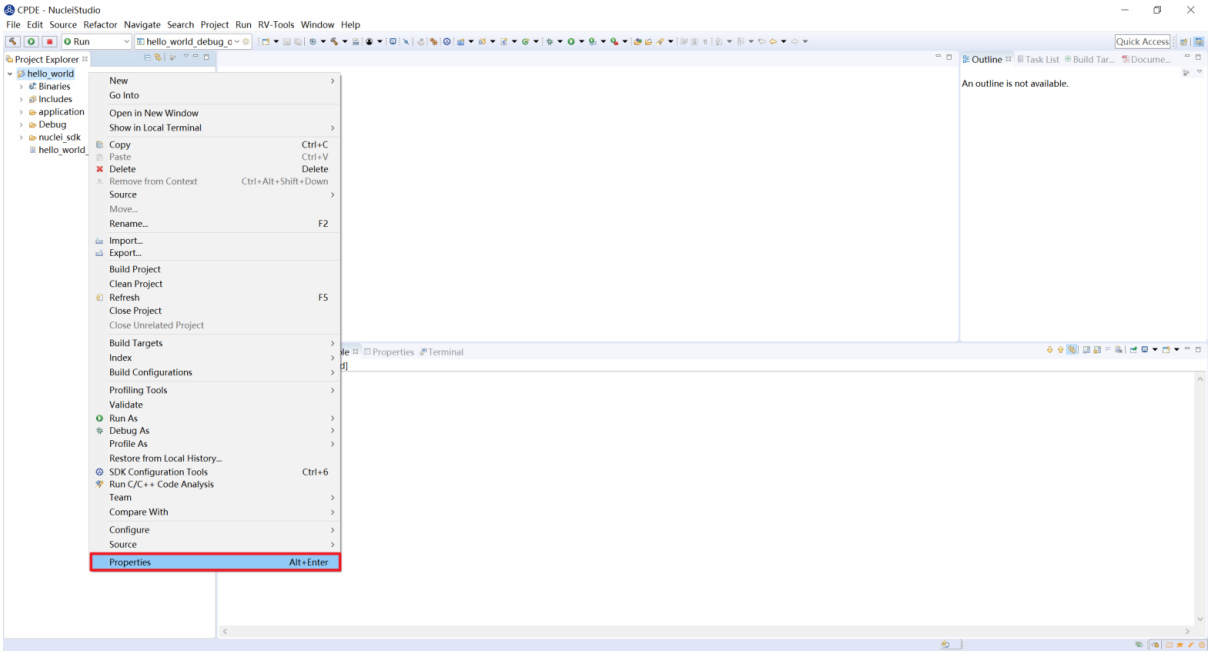


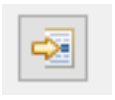
配置项目的 nuclei_sdk

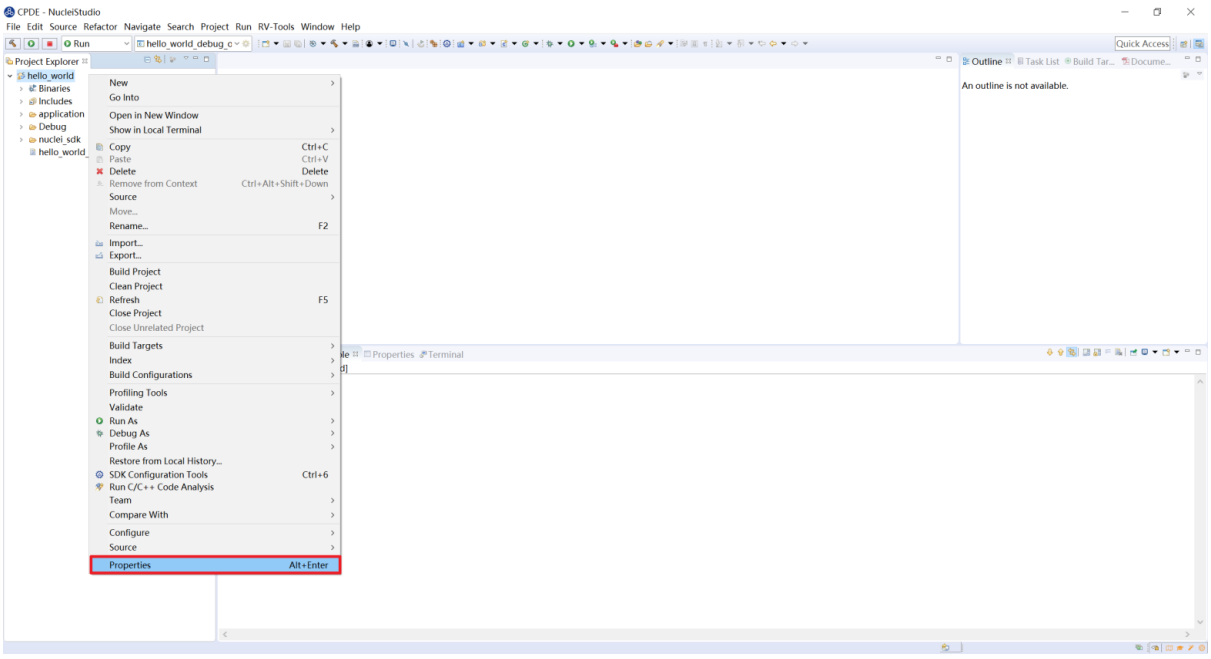
本节介绍如何将 nuclei_sdk 加入到项目中，SDK 的具体内容本文不做详细介绍，可以参考https://doc.nucleisys.com/nuclei_sdk/index.html。如果需要使用 SDK 的其他源文件，请到 Github 获取全部的 Nuclei SDK 源码（这里以 0.3.9 版本为例），链接如下：https://github.com/Nuclei-Software/nuclei-sdk/releases 。本节仅介绍将 nuclei_sdk 中 helloworld 需要的文件加入到项目的步骤，如果使用新版本的 SDK，对应的目录结构可能有所调整，请自行解决，具体步骤如下：

进入 Nuclei Studio 的 `2_helloworld` 项目，按照如下步骤添加 nuclei_sdk 源文件。

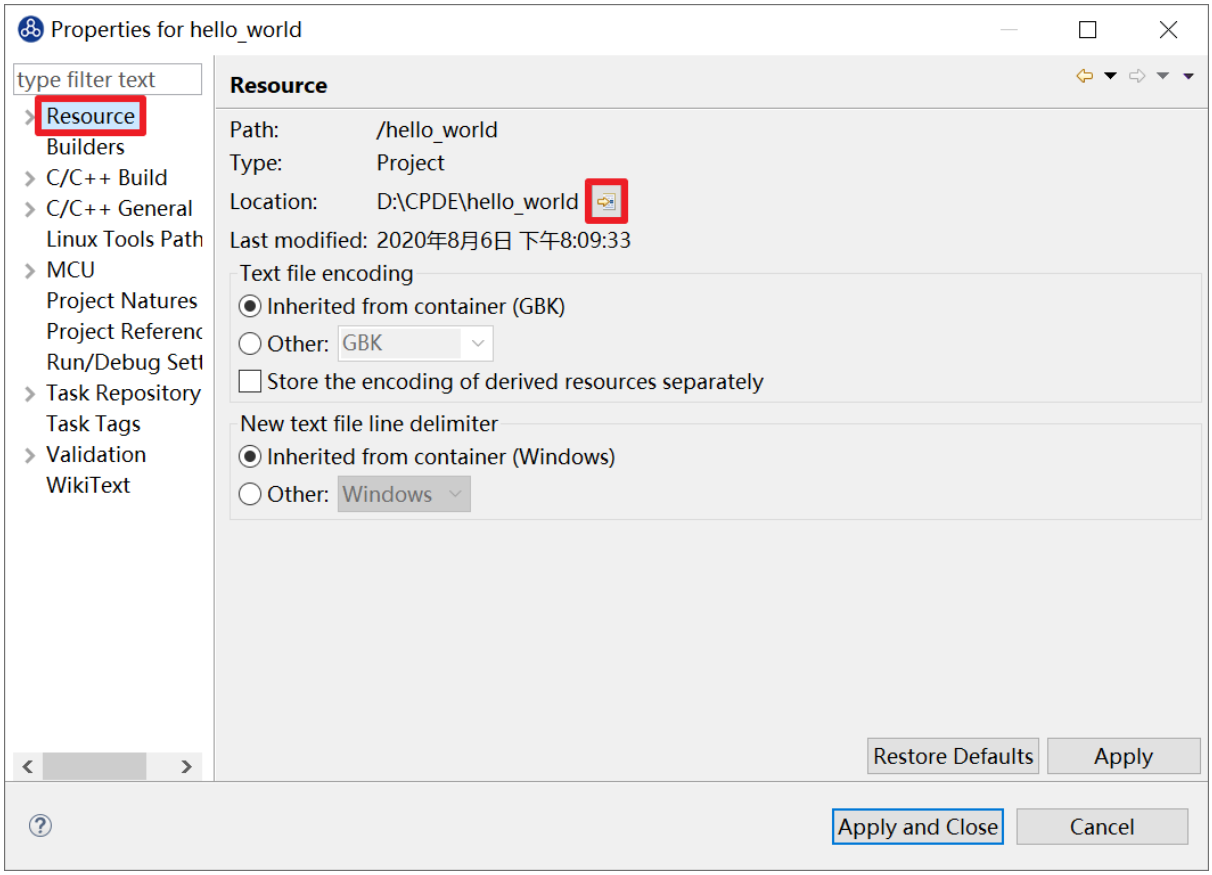
在 Project Explorer 栏中选中 `2_helloworld` 项目，单击鼠标右键，选择 `Properties` 打开工程设置页面。



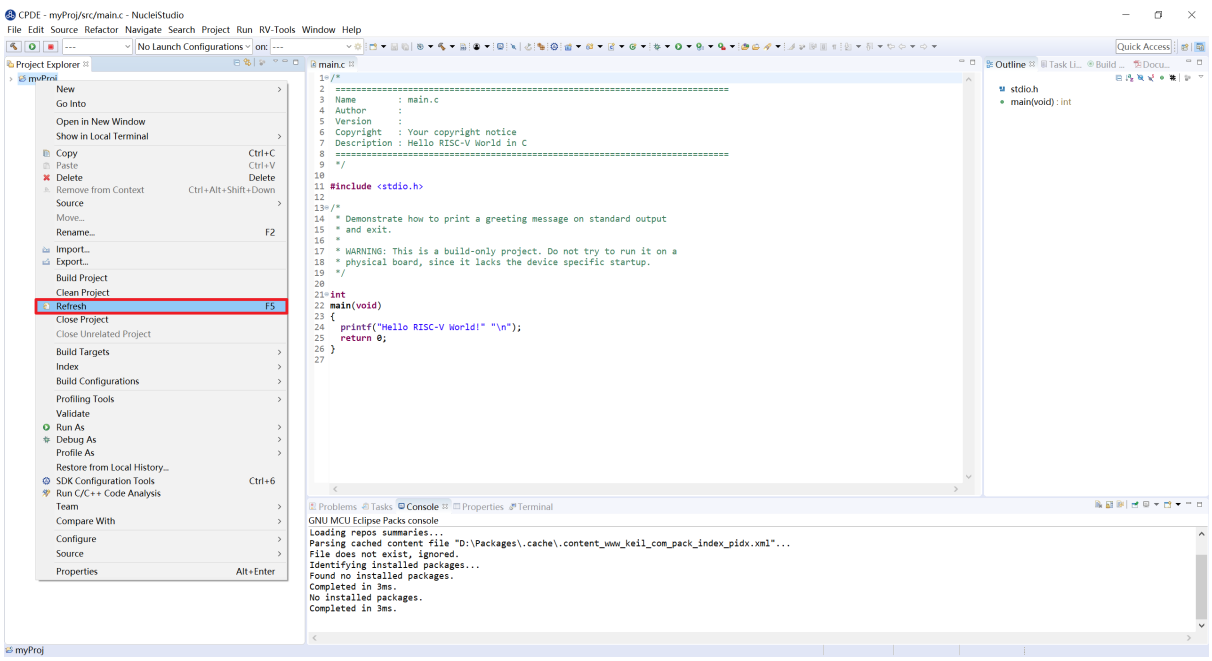
在弹出的窗口中单击 Resource，在右侧的 Location 栏目中单击其最右侧的箭头图标  ，则会弹出文件窗口进入 2_helloworld 项目的文件夹位置。



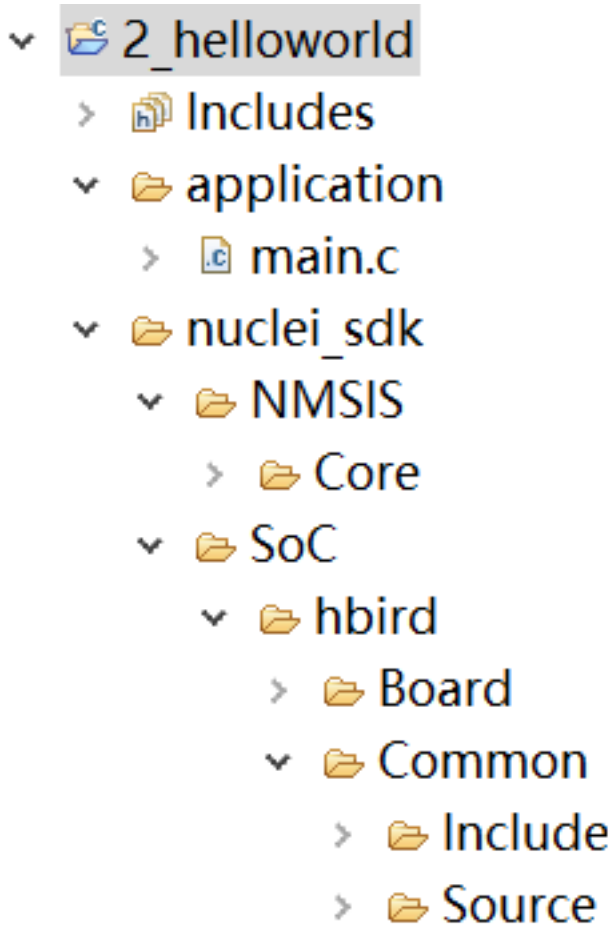
将 nuclei-eclipse_demo.rar 压缩包中的 nuclei_sdk 文件夹复制放于 2_helloworld 项目的目录下。



回到 Nuclei Studio，在 Project Explorer 栏中选中 `2_helloworld` 项目，单击鼠标右键，选择 `Refresh`。



Refresh 之后 `2_helloworld` 项目的下便可以看到 nuclei_sdk 文件夹，至此便完成了 nuclei_sdk 源文件的导入。



配置项目的编译和链接选项

为了使项目源代码能够被正确编译，需要配置编译和链接选项。

Note: 注意：本节中设置的编译与链接选项均为 GCC 工具链的常用选项，与在 Linux 环境中使用时的同名选项含义一致，本节在此不做赘述介绍。

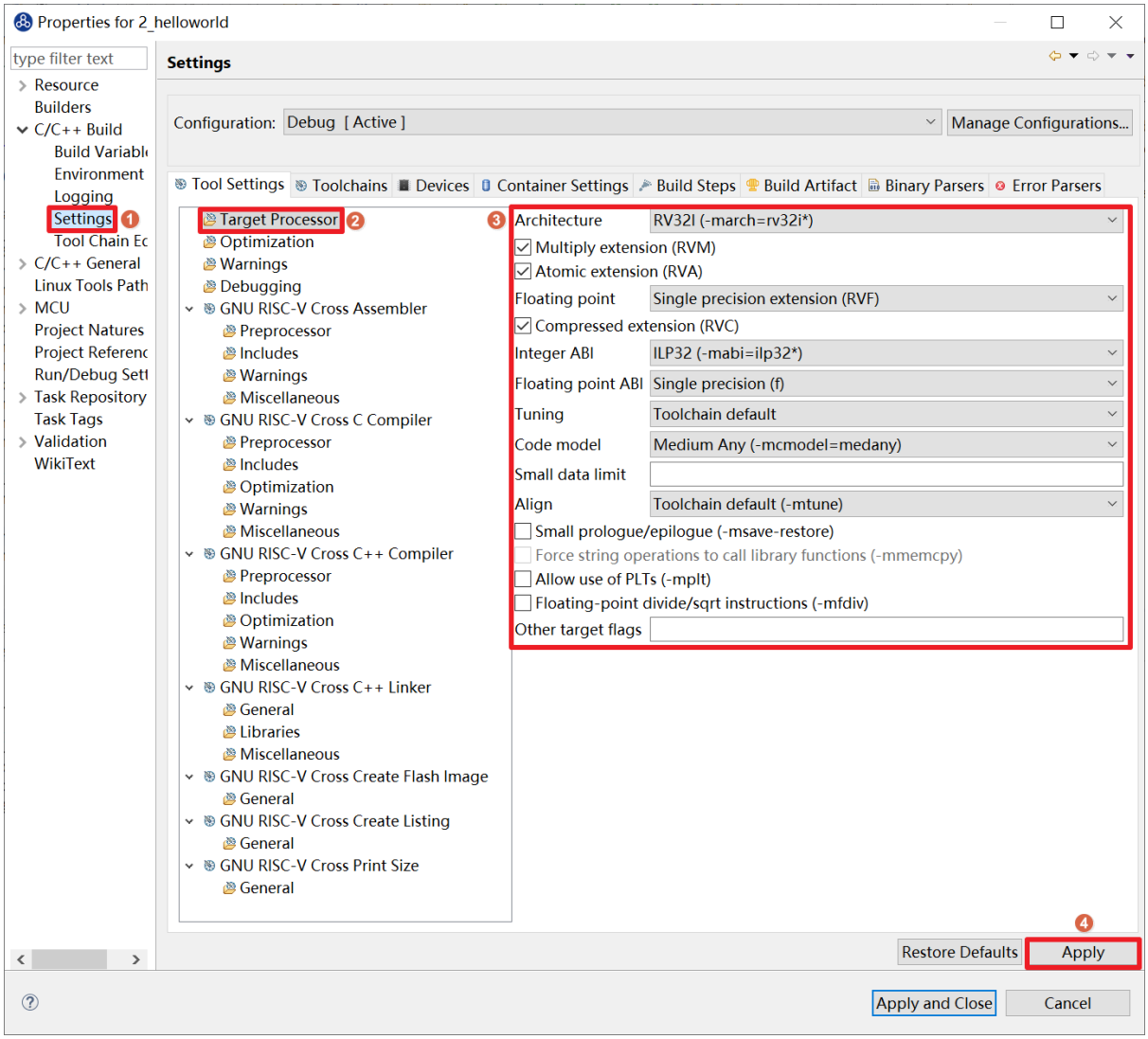
配置编译与连接选项的步骤如下：

在 Project Explorer 栏中选中 hello_world 项目，单击鼠标右键，选择 `Properties` 。

在弹出的窗口中，展开 C/C++ Build 菜单，单击 `Setting` ，在右侧的 Tool Settings 栏目中进行设置。

选中 Target Processor，我们的内核是 N307，因此需要按照图所示勾选配置选项，分别如下。

- Architecture：选择 `RV32I` 。
- Multiply extension（RVM）：需勾选。
- Atomic extension（RVA）：需勾选。
- Compressed extension（RVC）：需勾选。
- Integer API：选择 `ILP32` 。
- Floting Point ABI：选择 `single precision`
- Code model：选择 `Medium Any` 。
- 单击右下角的 `Apply` 按钮。



选中 `Optimization`，按照图所示勾选配置选项。

- Optimization Level：选择 `Optimization Most (-O2)`。

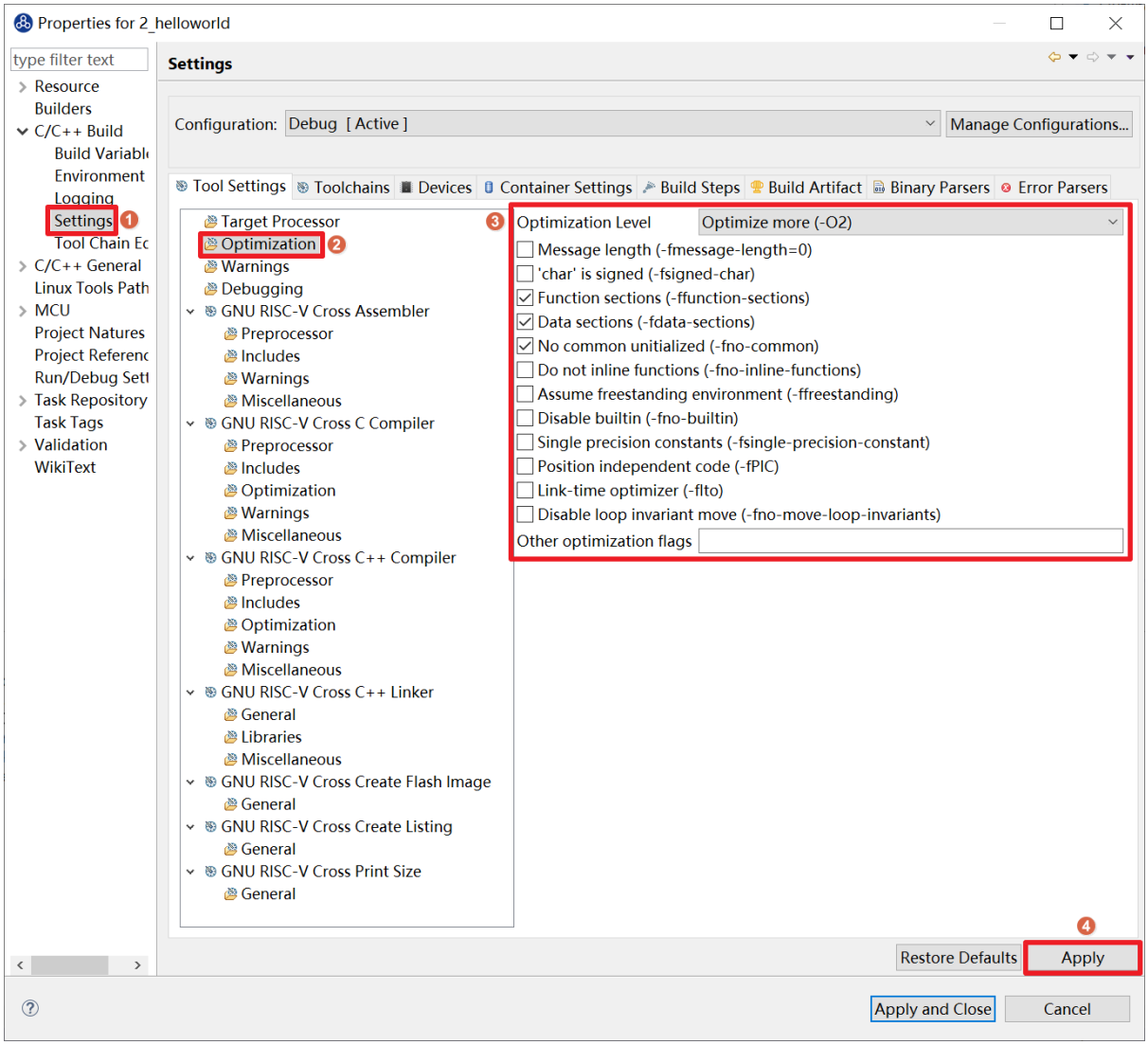
Note: 注意：在 NucleiStudio 2024.06 版本中新增了 `-Oz`，用来优化编译后程序的尺寸。

依次勾选：

- Function Sections (-ffunction-sections)
- Data Sections (-fdata-sections)
- No common unitialized (-fno-common)

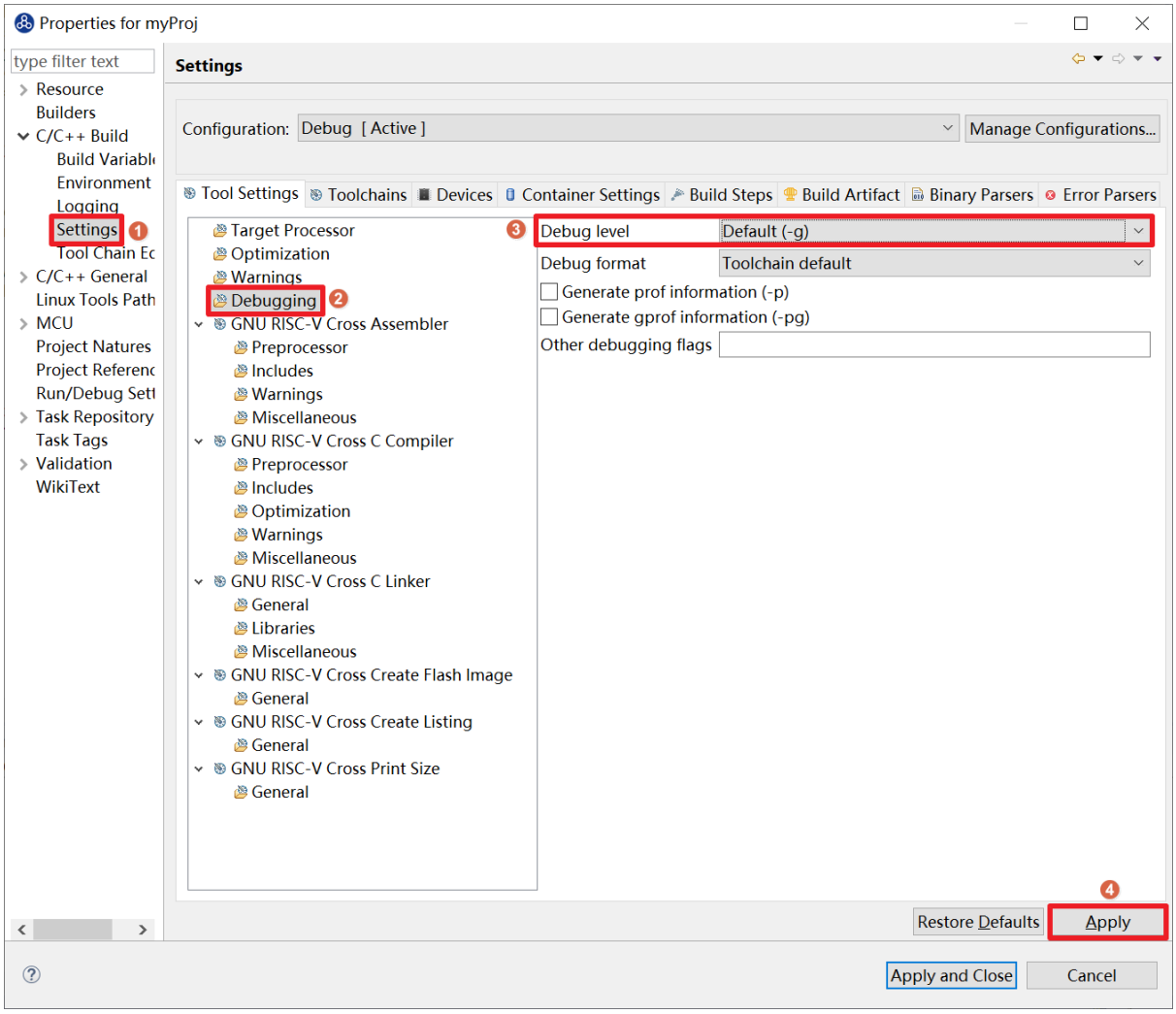
Note: 注意：上述选项均为通用的 GCC 编译优化选项，请用户自行查阅 GCC 手册了解其含义。

单击右下角的 `Apply` 按钮。



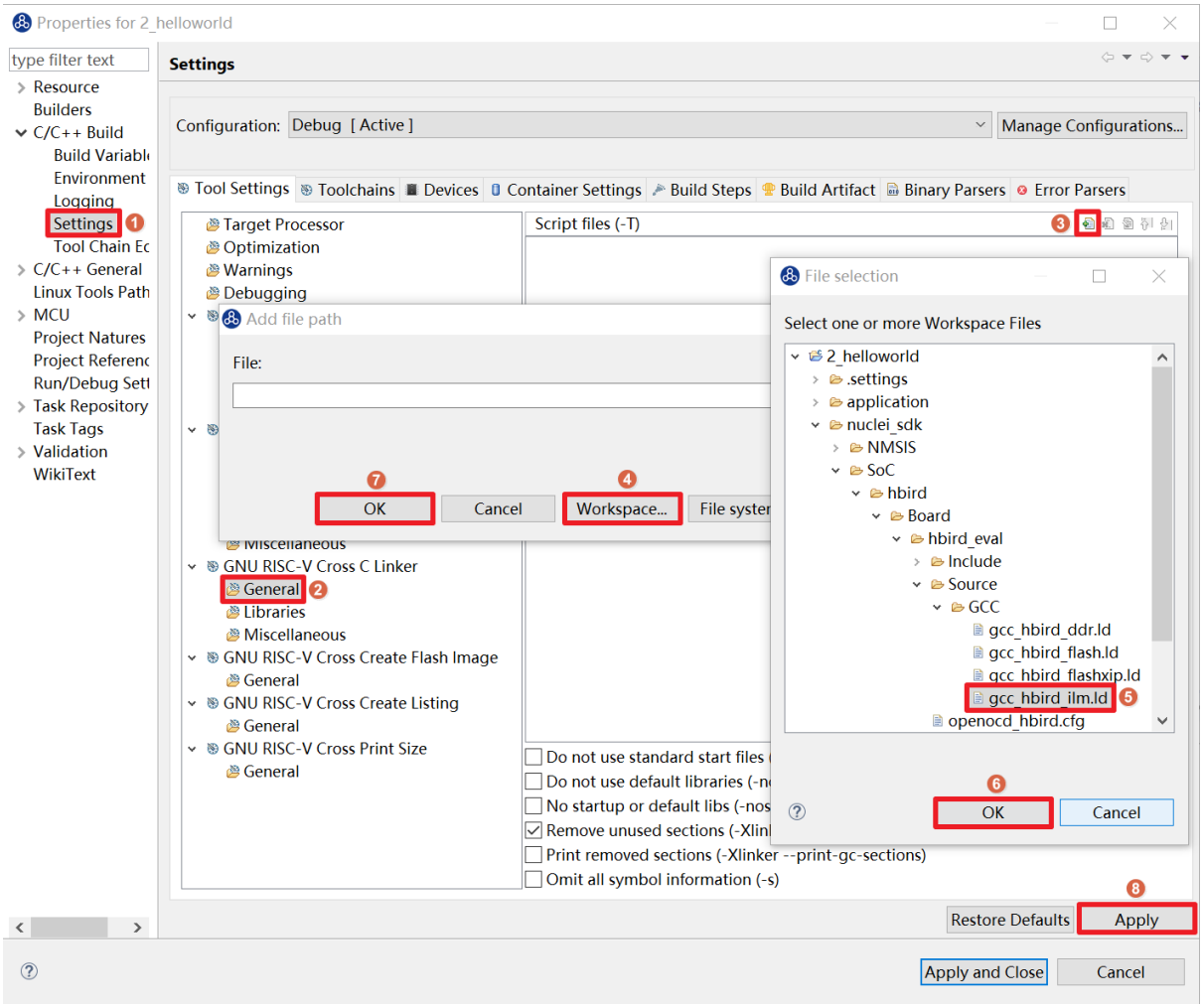
选中 Debugging，按照图中所示勾选配置选项，分别为：

- Debug Level：选择 `Default (-g)`。
- 单击右下角的 `Apply` 按钮。



选中 GNU RISC-V Cross C Linker 的 General。按照如下步骤设置链接器的所需的链接脚本。

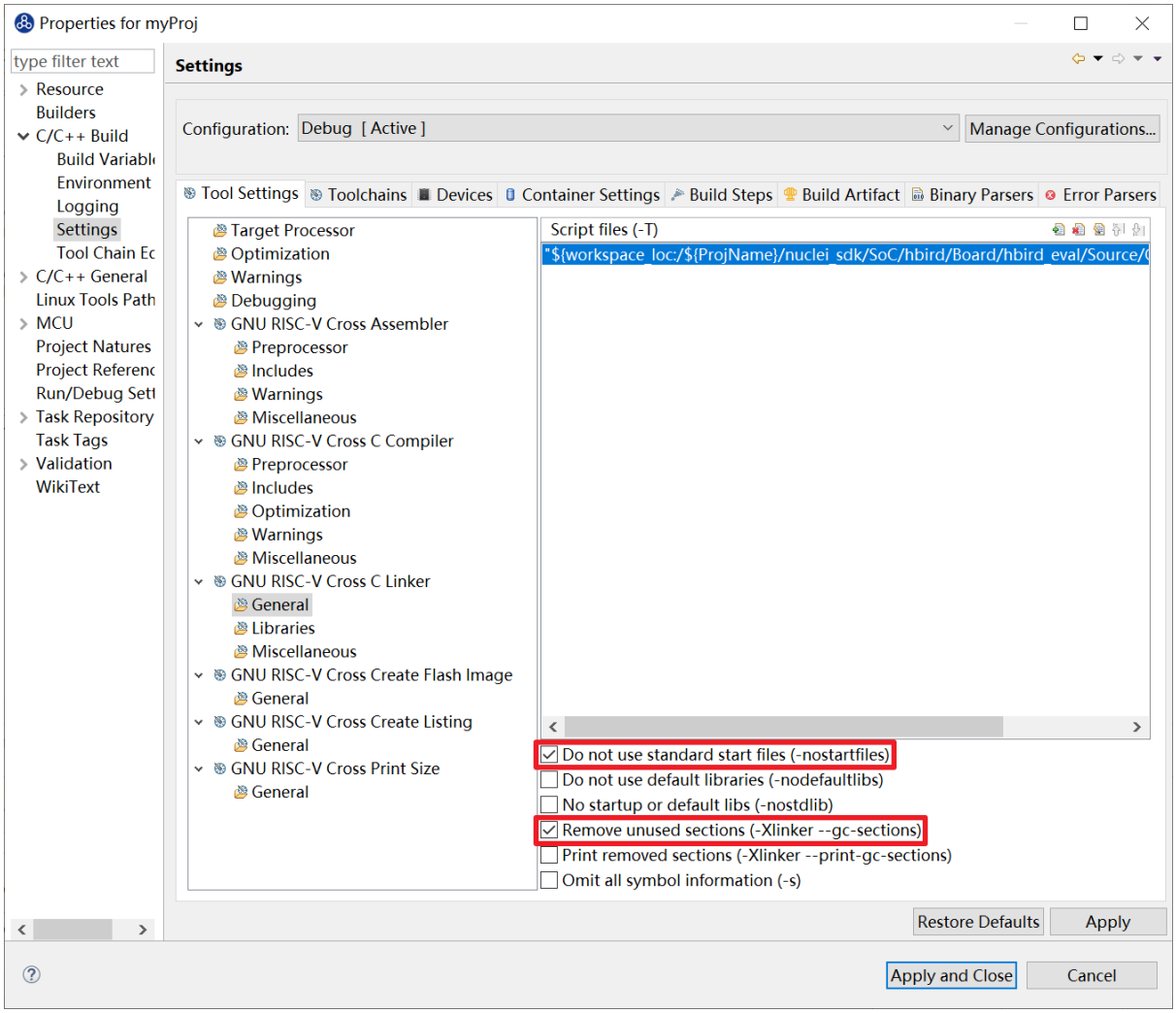
- 选中右上角的加号按键。
- 在弹出的窗口中单击 `Workspace` 按钮。
- 这里我们使用 HummingBird 评估板，所以可以选择 ILM 下载模式对应的 `gcc_hbird_ilm.ld` 文件。在弹出的窗口中选择 Nuclei Studio 文件包中的 `nuclei_sdk/SoC/hbird/Board/hbird_eval/Source/GCC` 文件夹下 `gcc_hbird_ilm.ld` 文件。其他下载模式切换此处文件，各文件详细介绍如下，可根据自己的实际情况选择。
  - `gcc_hbird_ilm.ld` 脚本将程序代码段约束在 ILM 的地址区间，意味着程序将被直接下载在 MCU 的 ILM 中，并从 ILM 开始执行。ILM 由 SRAM 组成，会掉电丢失。
  - `gcc_hbird_flash.ld` 脚本程序代码段的物理地址约束 Flash 区间，将代码段的逻辑地址约束在 ILM 的地址区间，意味着程序将被直接下载在 MCU 的 Flash 中，但是上电后要通过引导程序将代码段搬运到 ILM 中，然后从 ILM 中开始执行。
  - `gcc_hbird_flashxip.ld` 脚本程序代码段约束 Flash 区间，意味着程序将被直接下载在 MCU 的 Flash 中，并直接从 Flash 开始执行。程序被烧写在 Flash 中，不会掉电丢失。
  - 用户可以按照自己的需求选择合适的链接脚本。本节示例选择 `gcc_hbird_ilm.ld` 作为演示。
- 设置完毕请单击右下角的 `Apply` 按钮。



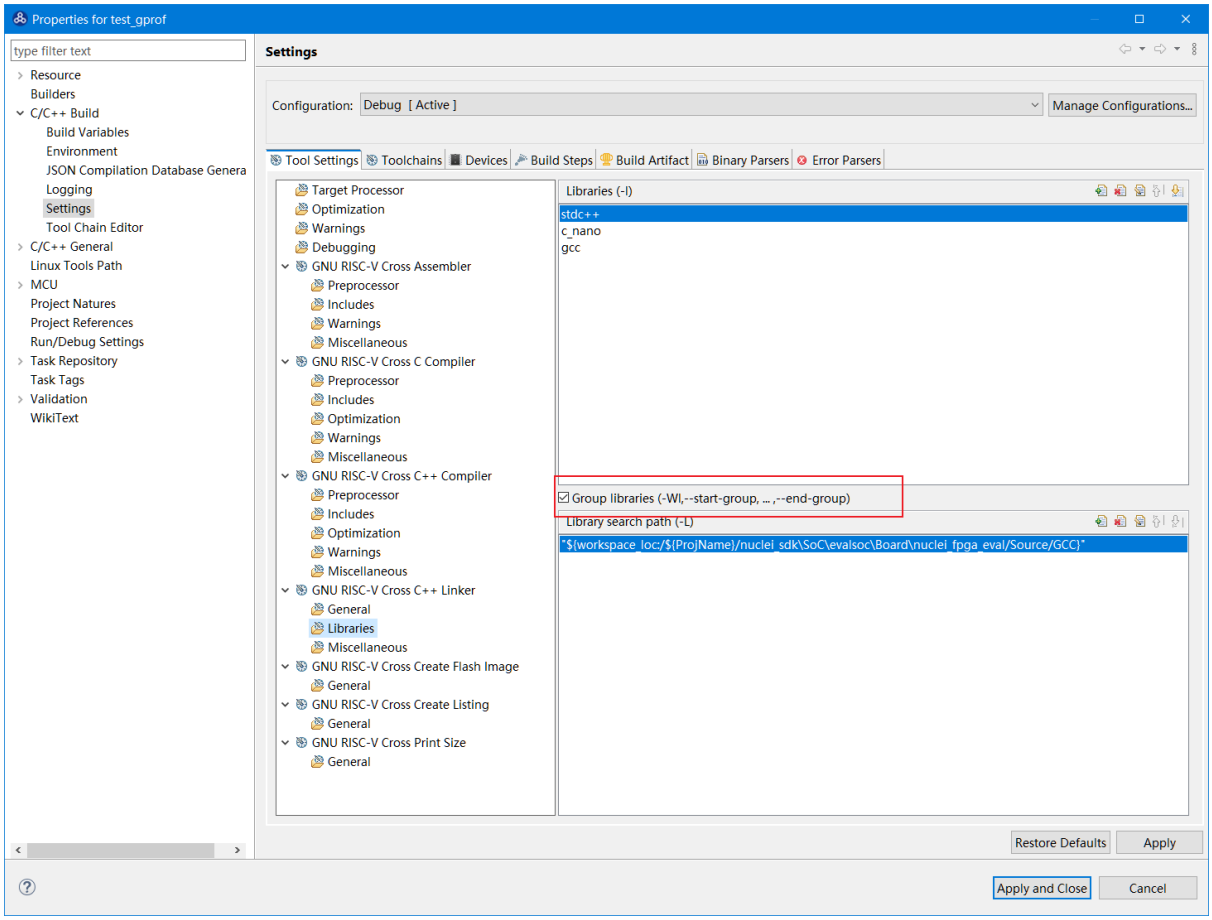
按下图所示勾选配置选项，分别如下。

- Do not use standard start files (-nostartfiles) 。
- Remove unused sections (gc-sections)。
- 单击右下角的 Apply 按钮。

Note: 注意：上述选项均为通用的 GCC 链接选项，请用户自行查阅 GCC 手册了解其含义。

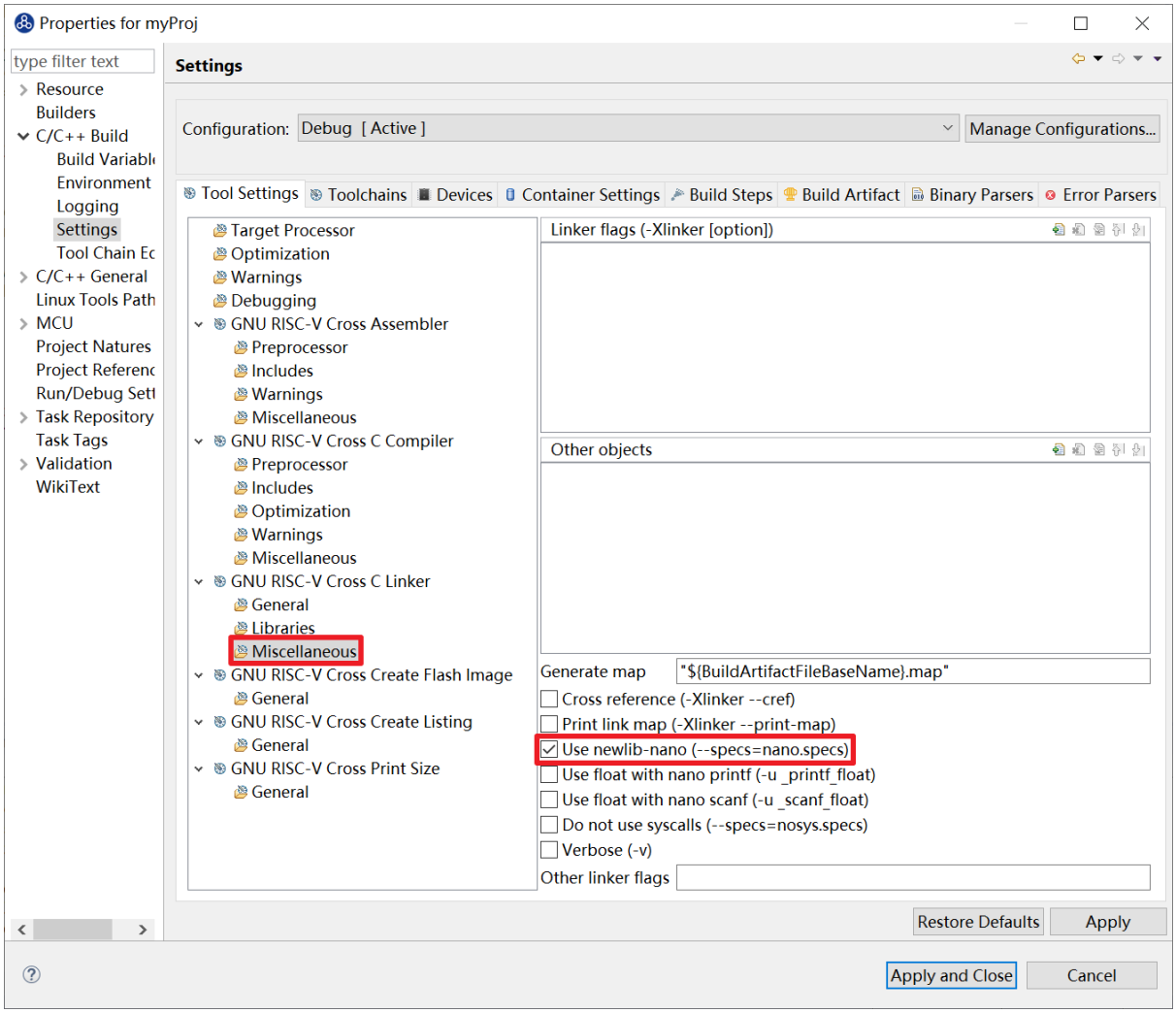


Note: 注意：在 NucleiStudio 2024.06 版本中 Libraries 支持 Group 功能，如果勾选了 Group 功能，所有的 Libraries 在编译时会用 `-Wl,--start-group,……,--end-group,` ，能解决 Libraries 内相互依赖的问题。



选中 GNU RISC-V Cross C Linker 的 Miscellaneous，按照下图所示勾选配置选项。

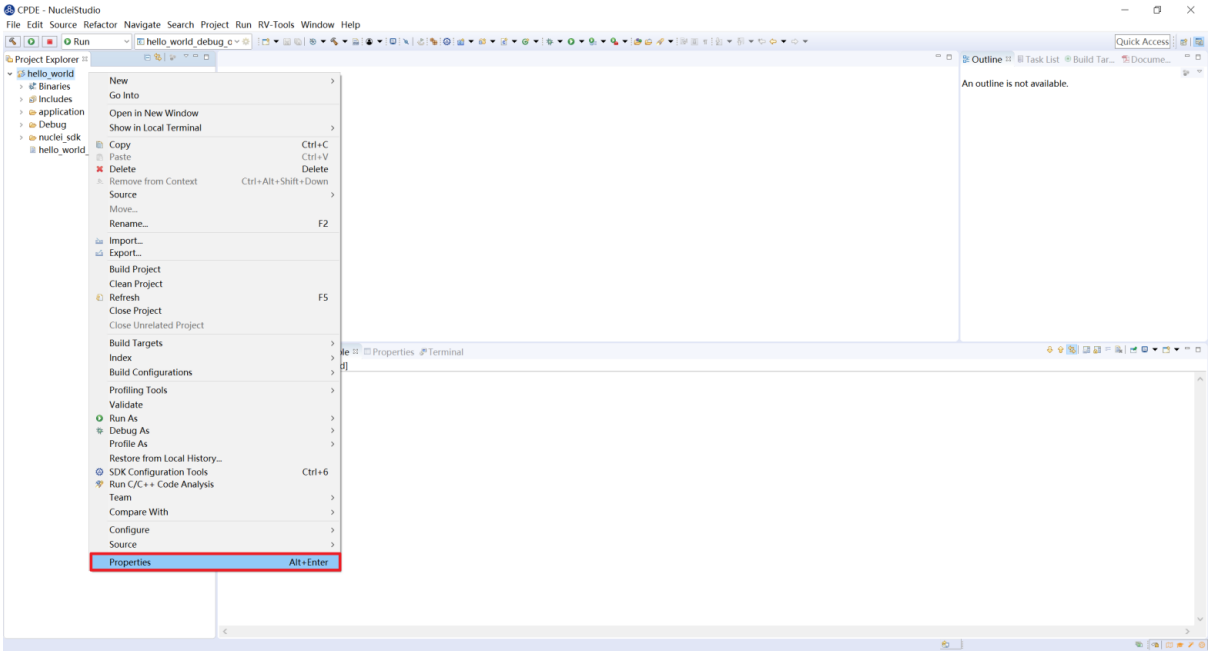
- 勾选 `Use newlib-nano` 。
- 因为 Hello World 程序的 Printf 不需要打印浮点数，所以不要勾选 `Use float with nano printf` 。
- 单击右下角的 `Apply` 按钮。



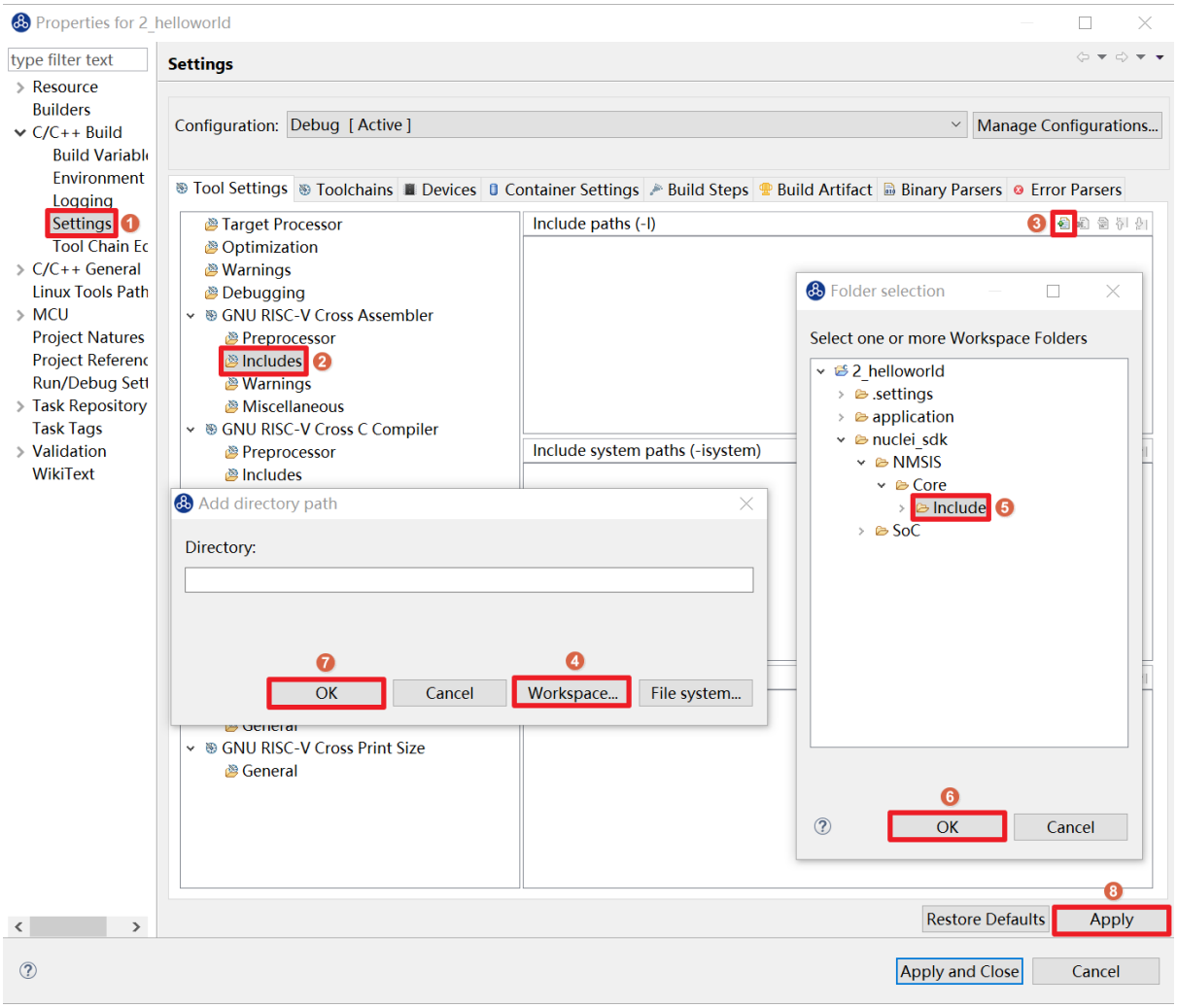
配置项目的包含路径和文件

为了能够正确编译 nuclei_sdk 文件夹中的源文件，需要按照如下步骤配置项目的包含路径和包含文件。

在 Project Explorer 栏中选中 hello_world 项目，点击鼠标右键，选择 Properties 。



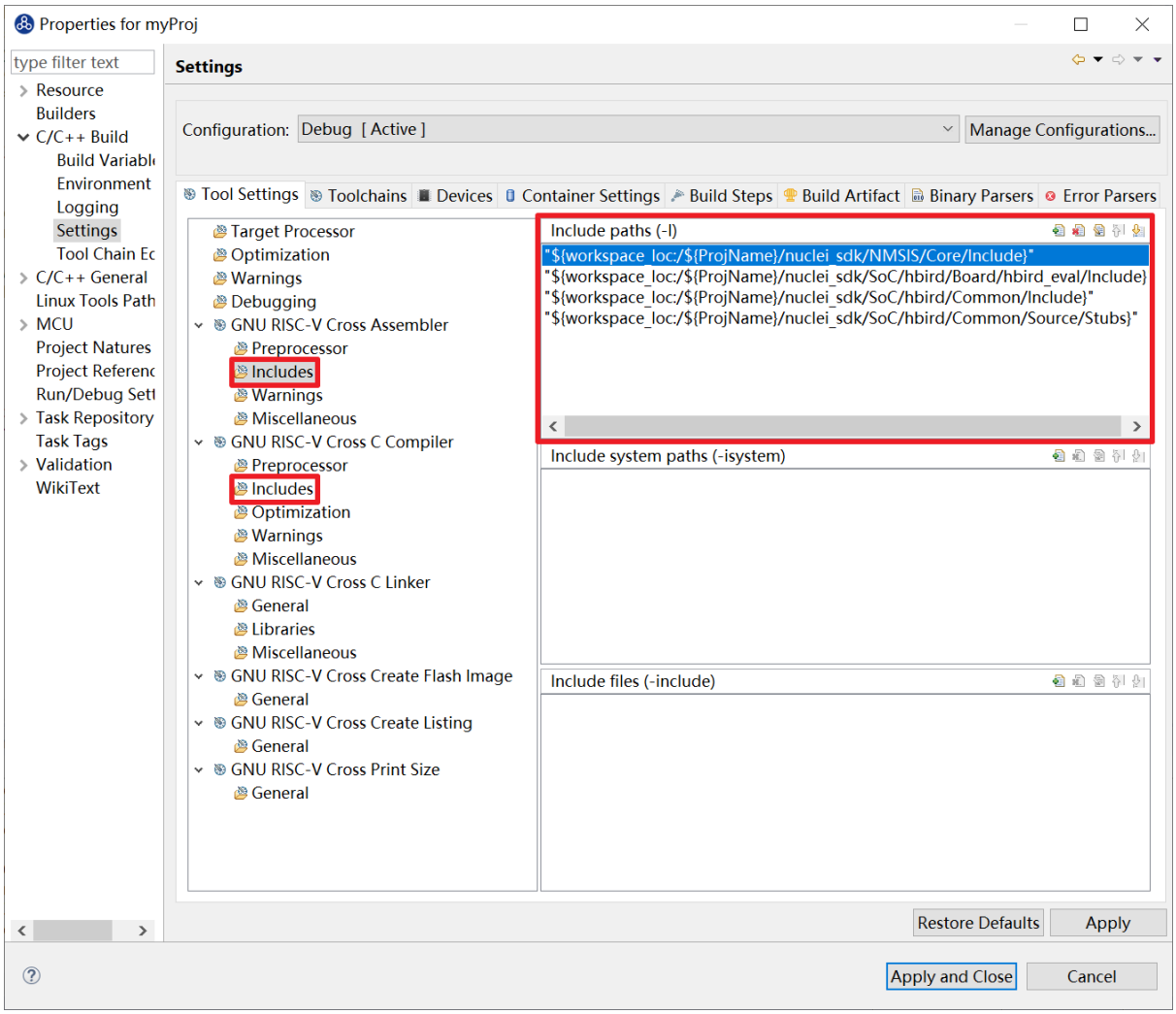
在弹出的窗口中，展开 C/C++ Build 菜单，单击 `Setting`，在右侧的 Tool Settings 栏目中进行设置。选中 GNU RISC-V Cross C Assembler 的 Includes，按照图中所示配置包含文件，步骤如下。



- 在 Include paths 栏目单击加号键。

- 在弹出的窗口中单击 Workspace，弹出 Folder selection 窗口。
- 在 Folder selection 窗口中选择项目的 nuclei_sdk 目录下的 NMSIS>Core>Include 文件夹。
- 在右下角单击 Apply 完成配置。

采用上述方法，依次添加 nuclei_sdk 目录下的 SoC>hbird>Board>hbird_eval>Include，SoC>hbird>Common>Include 和 SoC>hbird>Common>Source>Stubs 文件夹作为包含路径，并采用同样的方法为 GNU RISC-V Cross C Compiler 的 Includes 栏目设置包含路径。设置完成后的界面如下图所示。

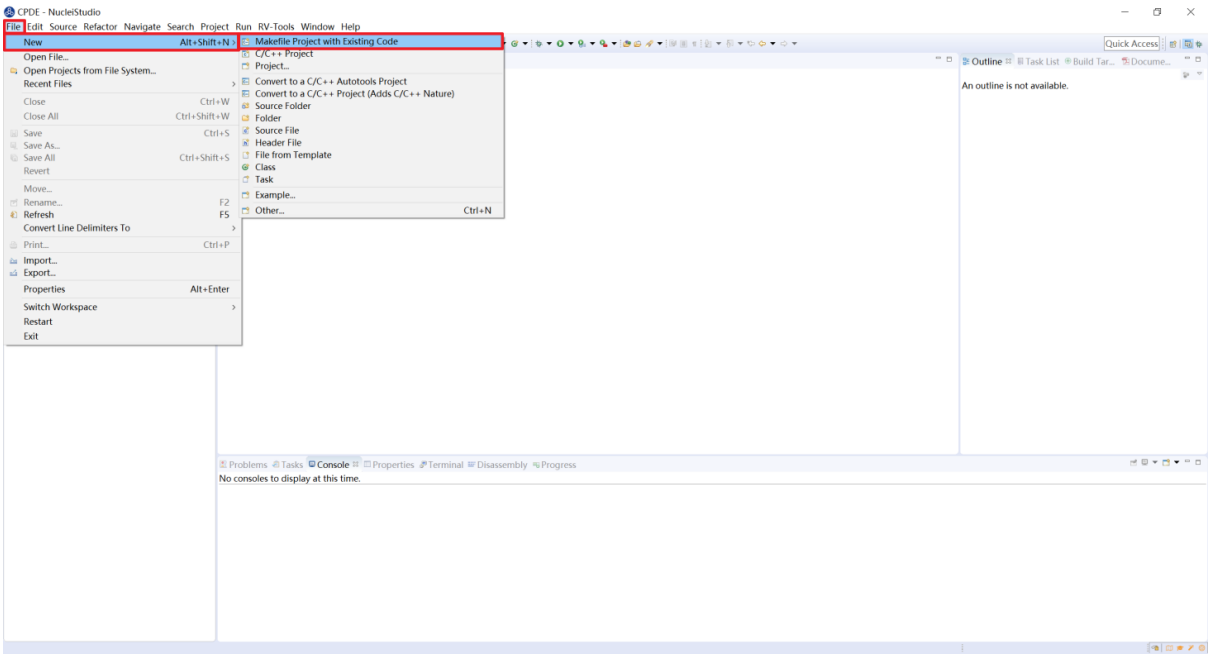


2.7.5 基于已有的 Makefile 创建项目

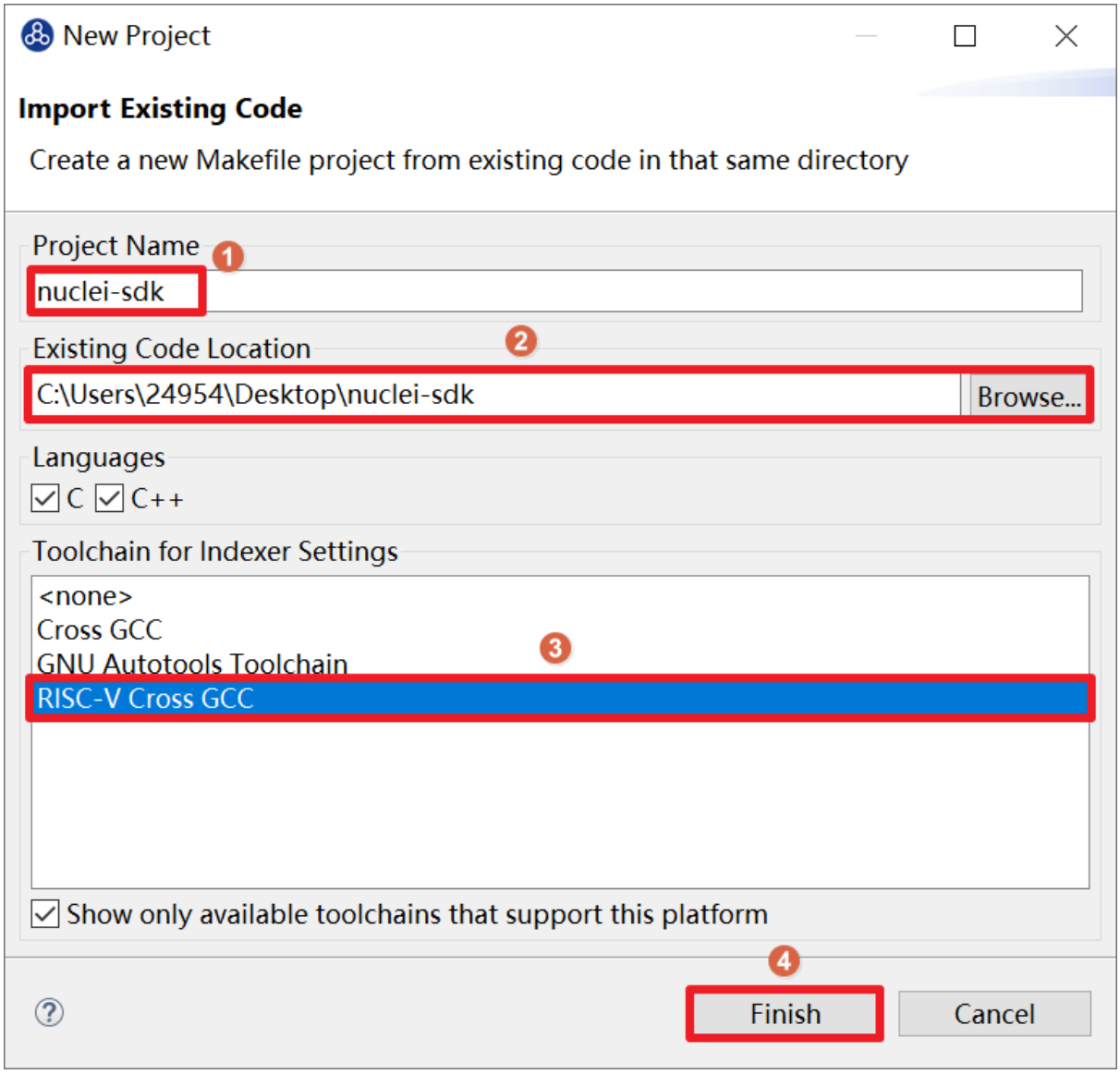
本节将介绍如何使用已有的 Makefile 在 Nuclei Studio IDE 创建一个使用 Makefile 的 Hello World 项目。开发板为 Nuclei FPGA Evaluation Board，内核为 N307。请先下载 Nuclei SDK，Github 链接为：https://github.com/Nuclei-Software/nuclei-sdk。该方法除了创建项目之外，还需要手动设置各种选项和路径，这里以 helloworld 为例，详细步骤如下。

手动新建项目

在菜单栏中选择 `File—> New —> Makefile Project with Existing Code`。

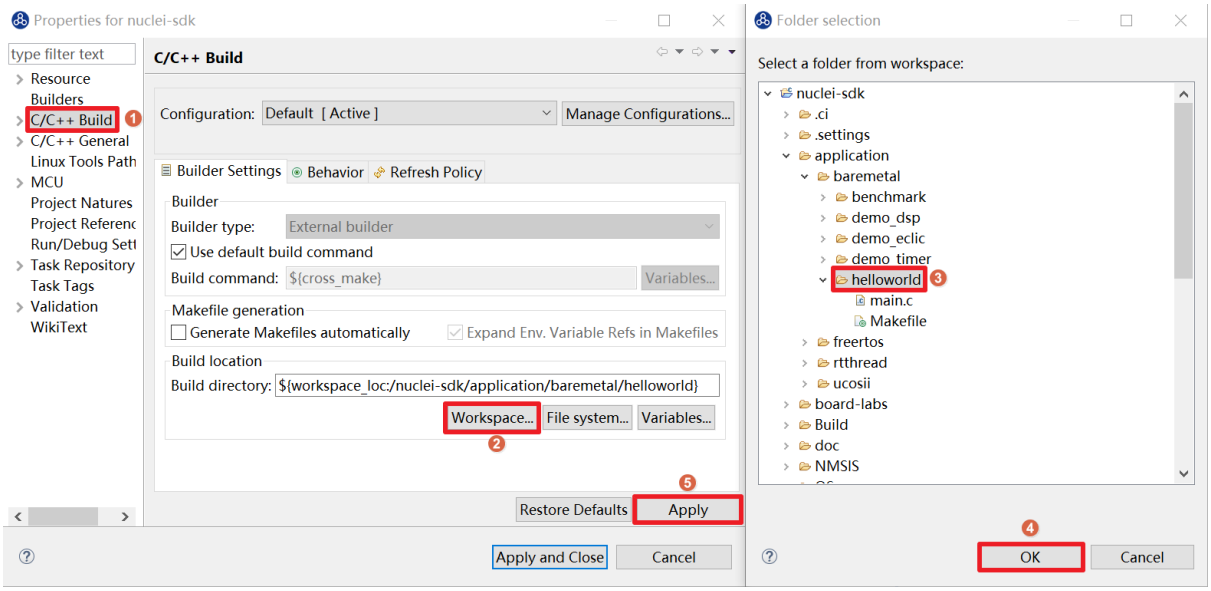


在图标 1 处输入工程名，这里我们命名为 nuclei-sdk。在图标 2 处输入 SDK 的实际路径。在图标 3 处选择 `RISC-V Cross GCC`。点击图标 4 完成新建项目。

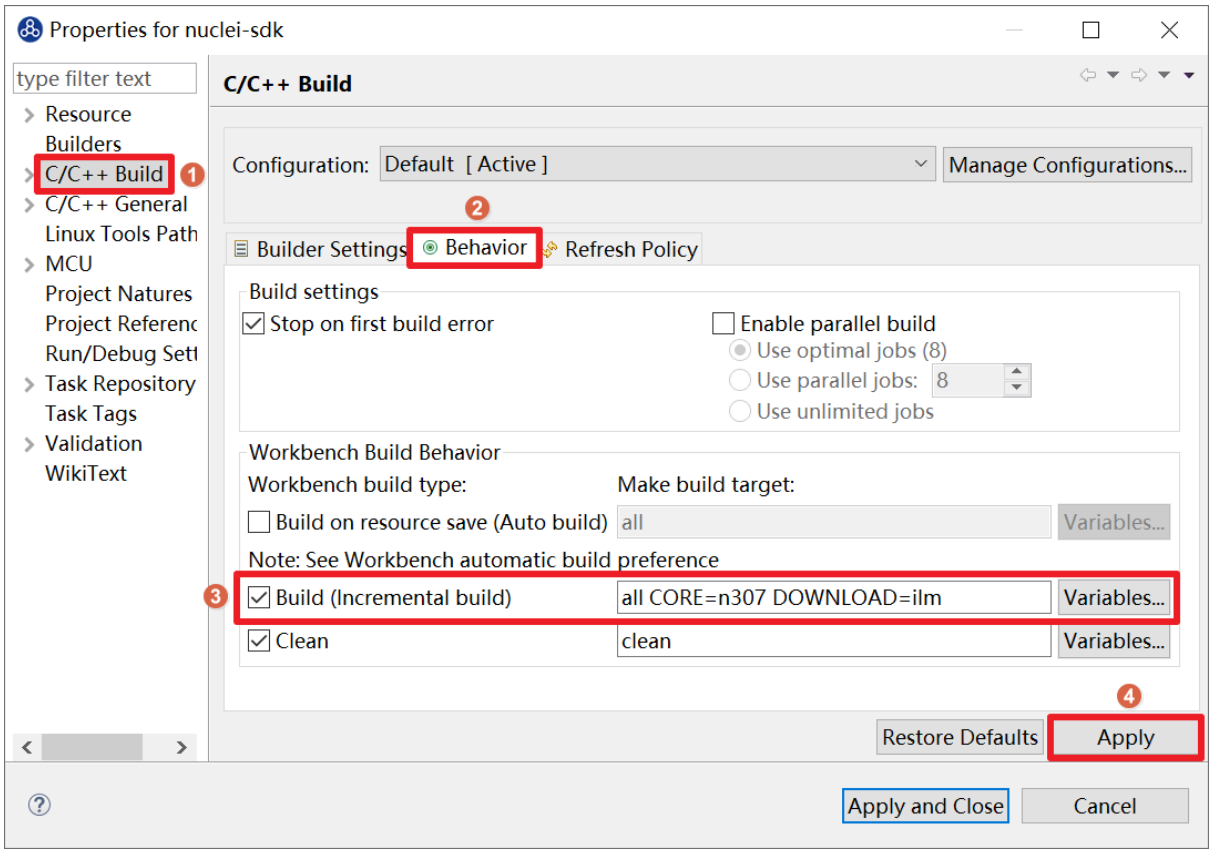


设置 **Makefile** 路径和 **Build** 选项

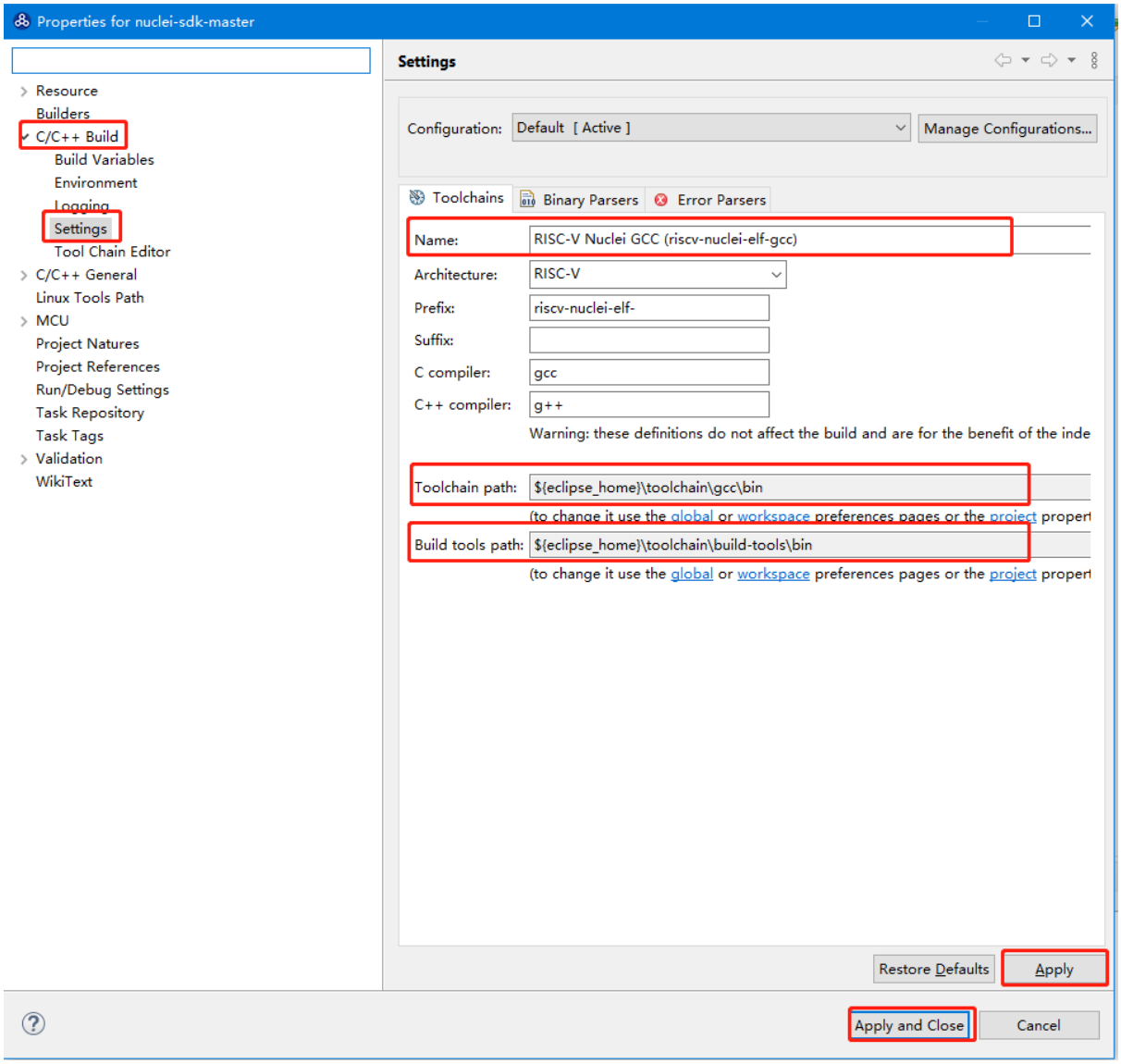
右击新建好的工程，选择 Properties 打开设置页面，选择 C/C++ uild，在 Build Location 中选择 Workspace。在弹出的弹窗中选择 application > baremetal > helloworld 点击 OK 再点击 Apply 保存。



在 C/C++ Build 中选择 Behavior 栏目，确保勾选 Build（Incremental Build）选项并输入 all CORE=n307 DOWNLOAD=ilm 。其中 CORE 选项根据实际的内核变化，这里以 n307 为例。DOWNLOAD 选项可以修改不同的下载模式，详细请参考 5.1 节，这里以 ilm 模式为例。因为例程使用 HummingBird Evaluation Board，所以 SoC 和 Board 都不必修改，如果使用其他开发板，以 RVSTAR 为例，请在此处设置增加 SOC=gd32vf103 BOARD=gd32vf103v_rvstar ，并且由于 RVSTAR 仅支持 FLASHXIP 模式，需要将 DOWNLOAD 设置为 flashxip ，同时 CORE 修改为 n205 。完成后点击 Apply 保存修改。

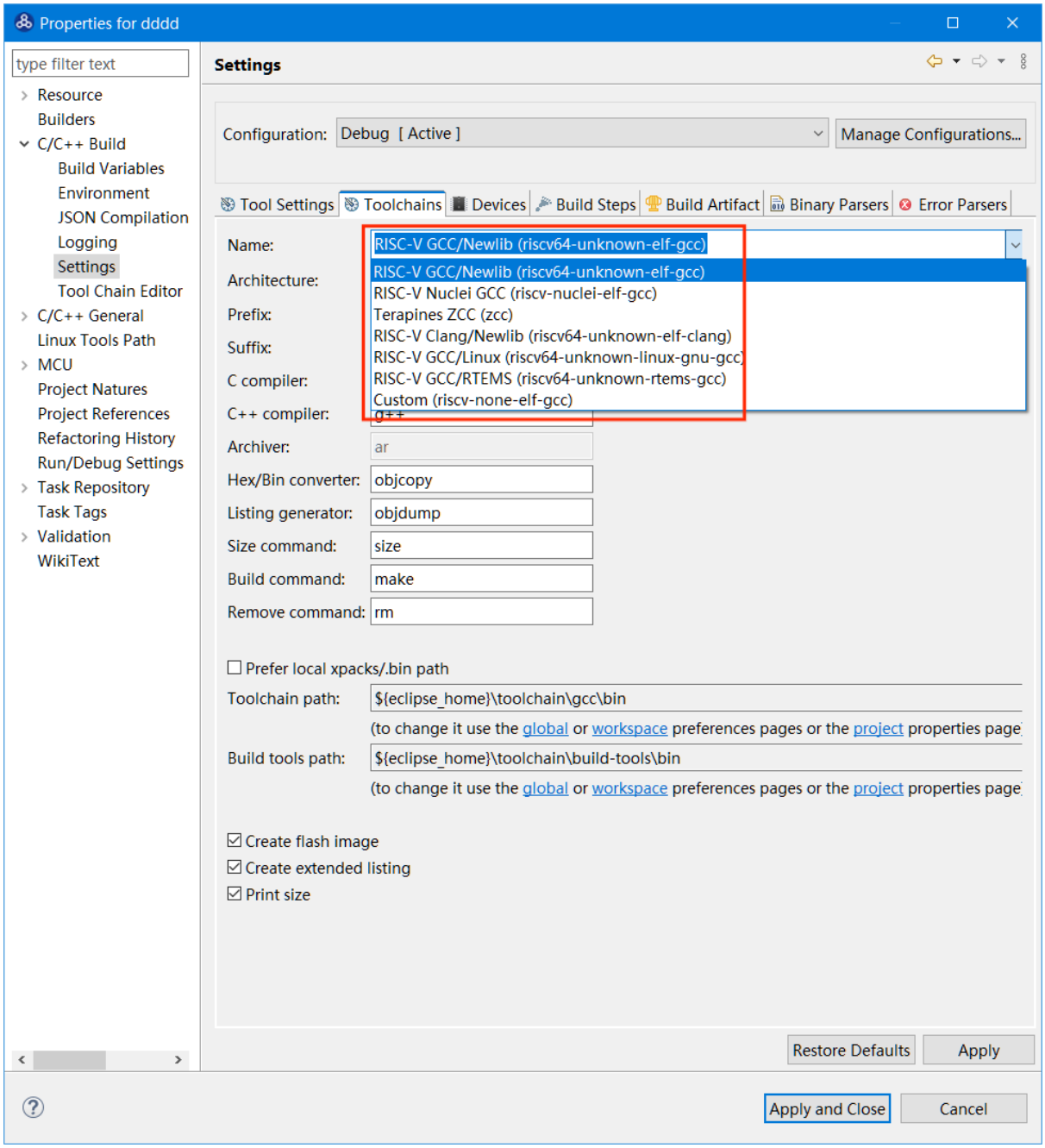


在完成上述操作后，打开工具链配置页，点击 Apply 保存修改。



2.8 Nuclei Studio 编译工程

在 NucleiStudio 中，支持多种 Toolchain，以便用户在创建工程时，可以选择不同的 Toolchain 对工程进行编译。NucleiStudio 中已经集成了 GCC 13、Clang 17、ZCC Lite 三款编译器，用户无需下载即可使用。



- **RISC-V Nuclei GCC (riscv-nuclei-elf-gcc)**

早期 NucleiStudio 对 GCC 10 编译器的支持，Nuclei Studio 2023.10 及之后版本弃用，如果用户需要对 GCC 10 的支持，需要自行下载 Nuclei RISC-V Toolchain 2022.12(gcc10) 并替换 `<NucleiStudio>/toolchain/gcc10` 目录内容。

- **RISC-V GCC/Newlib (riscv64-unknown-elf-gcc)**

Nuclei Studio 2023.10 及之后对 GCC 13 编译器的支持，对应的软件包存放在 `<NucleiStudio>/toolchain/gcc` 目录。

- **RISC-V Clang/Newlib (riscv64-unknown-elf-clang)**

Nuclei Studio 2023.10 及之后对 Clang 17 编译器的支持，对应的软件包存放在 `<NucleiStudio>/toolchain/gcc` 目录。

- **Terapines ZCC (zcc)**

Nuclei Studio 2024.06 版本对 ZCC 编译器的支持，对应的软件包存放在 `<NucleiStudio>/`

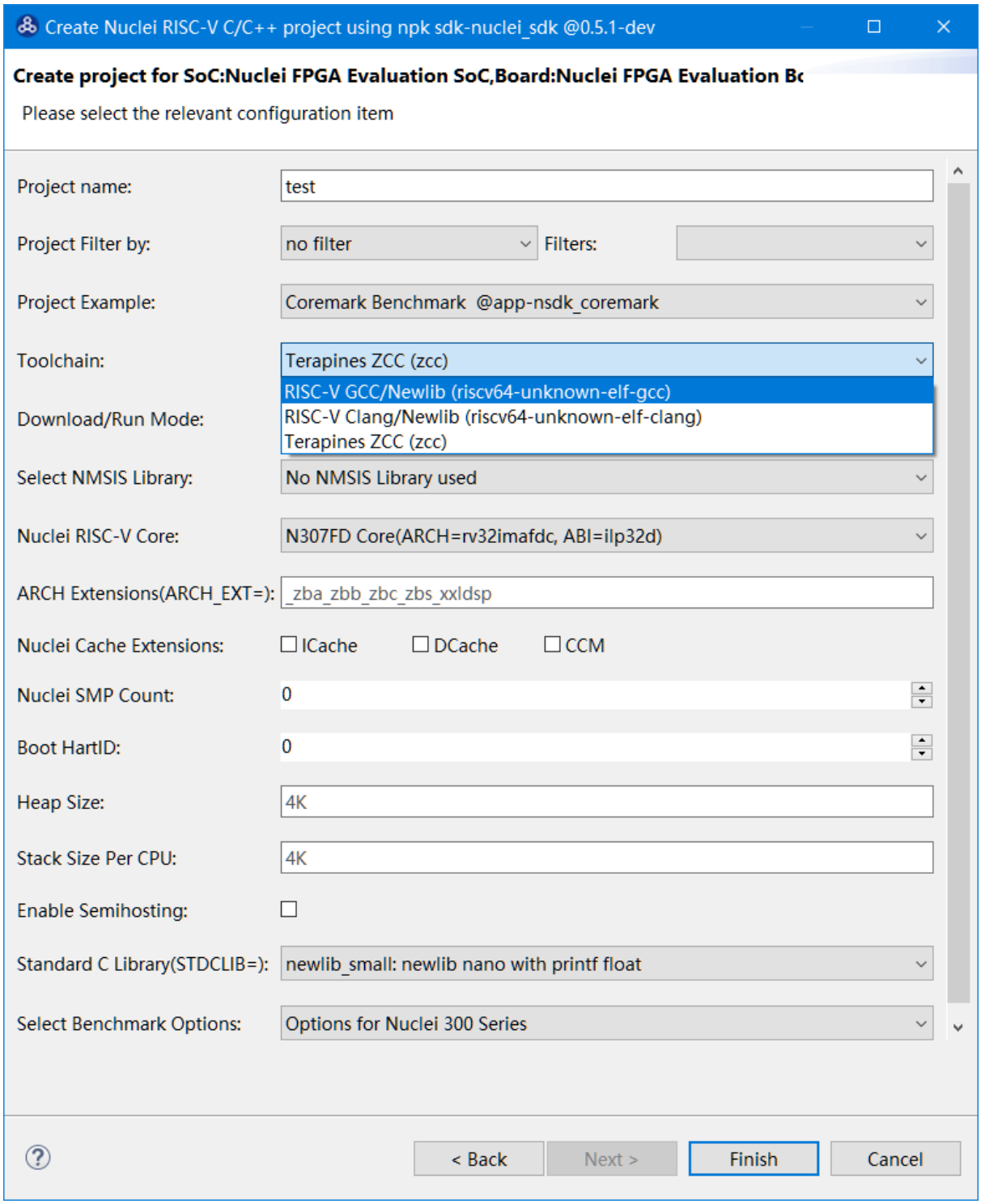
toolchain/zcc 目录。Terapines ZCC 工具链工程的创建需要依赖 Nuclei SDK **> 0.6.0** 之后的版本才可以, 目前可以通过 Nuclei SDK 命令行的方式进行测试。

2.8.1 编译工具

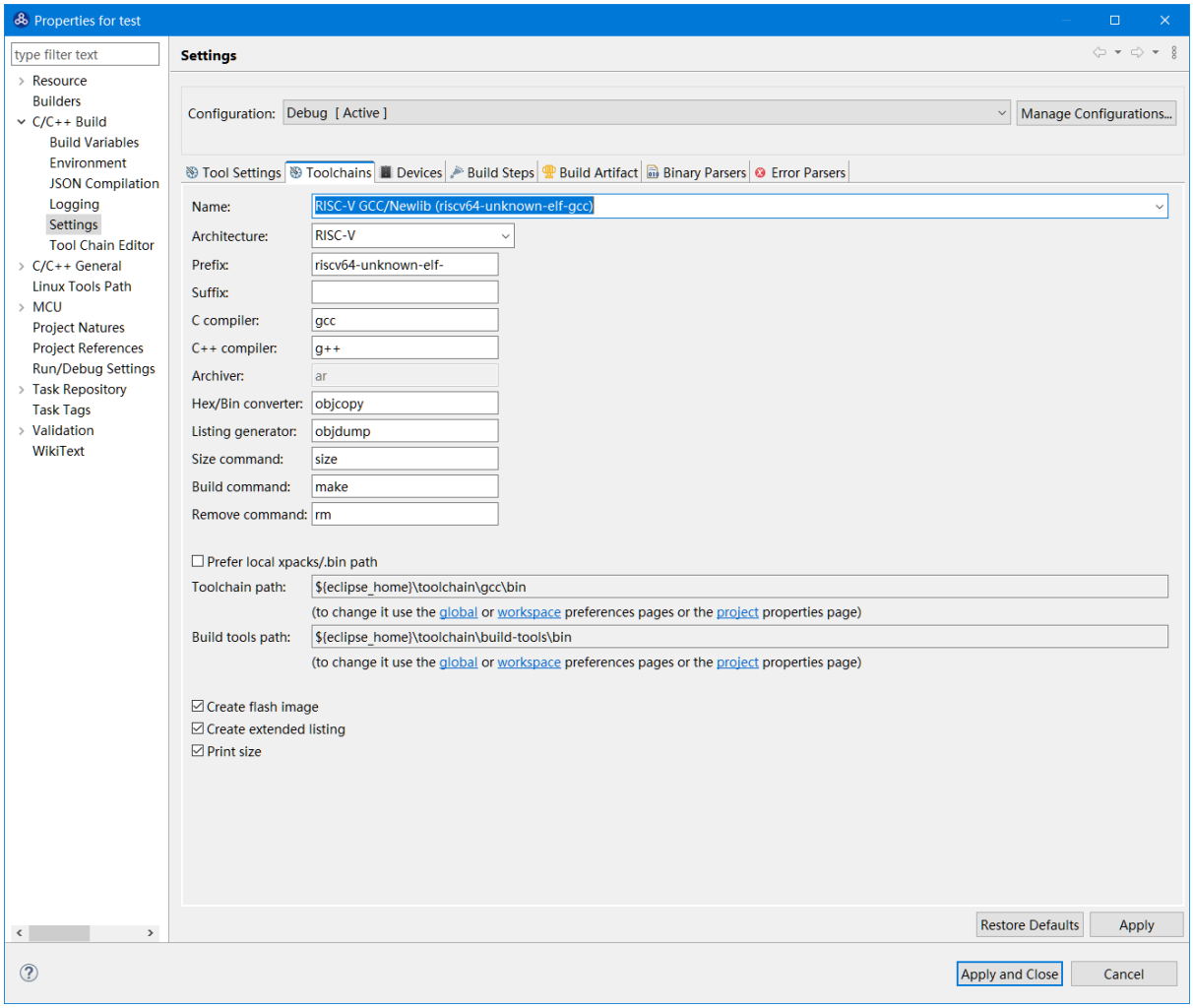
Nuclei GNU Toolchain

GNU Toolchain 是由 GNU 项目提供的一套完整的软件开发工具链，它包括了编译器、调试器、链接器、库文件等一系列用于软件开发和构建的必需工具。GNU Toolchain 以其开源、跨平台、高度可定制和强大的功能特性，成为了全球开发者社区广泛使用的开发工具集。Nuclei Studio 2023.10 之前的版本中集成了 GCC 10 ; Nuclei Studio 2023.10 及之后的版本中，集成了 GCC 13 。

在 nuclei_sdk 0.5.0 之后的版中，在创建工程时，用户可以选择 Toolchain 为 RISC-V GCC/Newlib (riscv64-unknown-elf-gcc) 则可以创建一个支持 GCC 13 编译的工程，NucleiStudio 将默认将相对应的编译选项配置好。关于 GCC 10 与 GCC 13 工程的问题，可以参阅通过工具将工程转换成支持 *gcc 13* 的工程 (page 177) 。



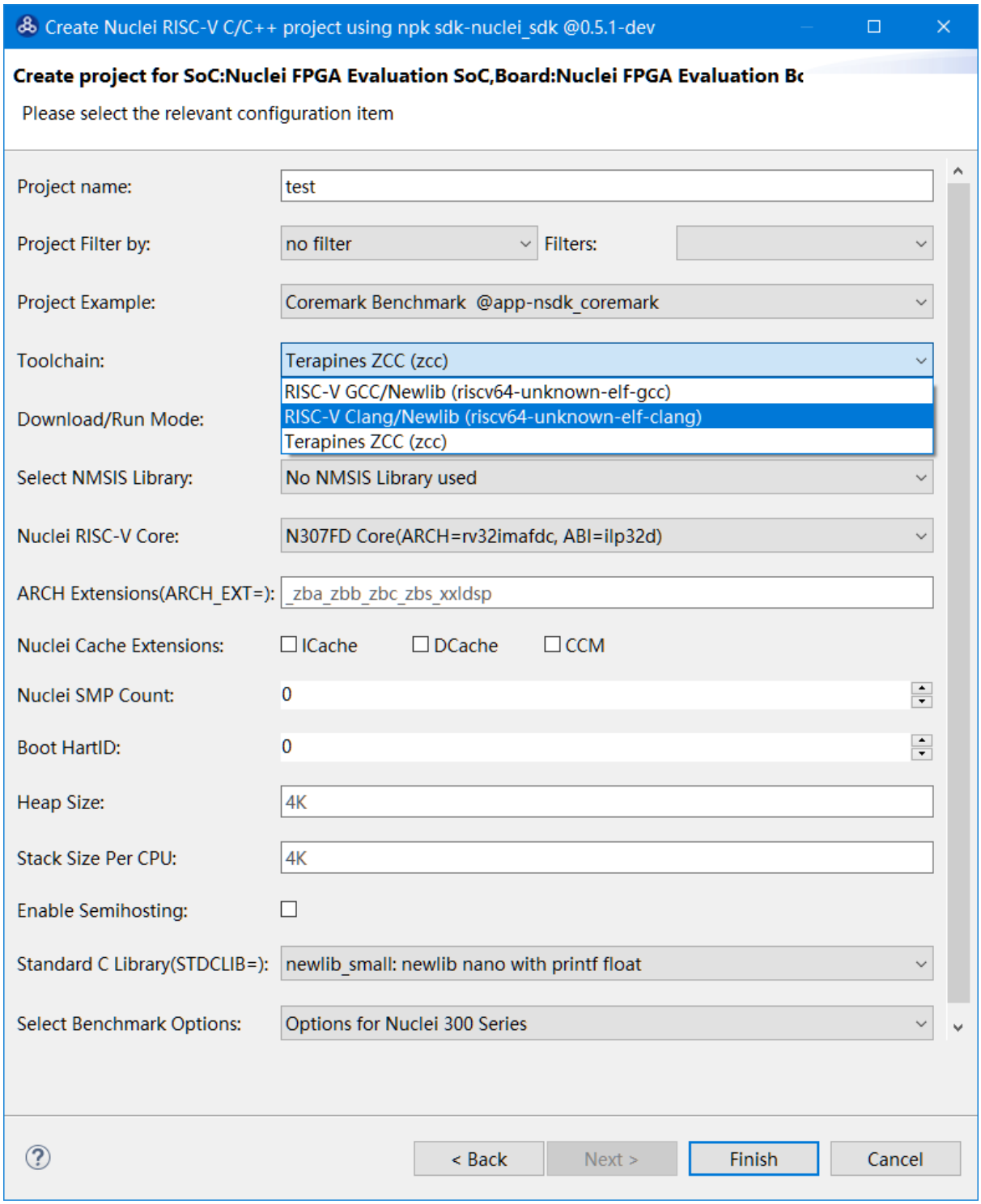
在工程上 右键->Properties->C/C++ Build->Setting->Toolchains 中可以看到工程所用到的 Toolchains 有一些工具，以及 Toolchains 所在的路径。



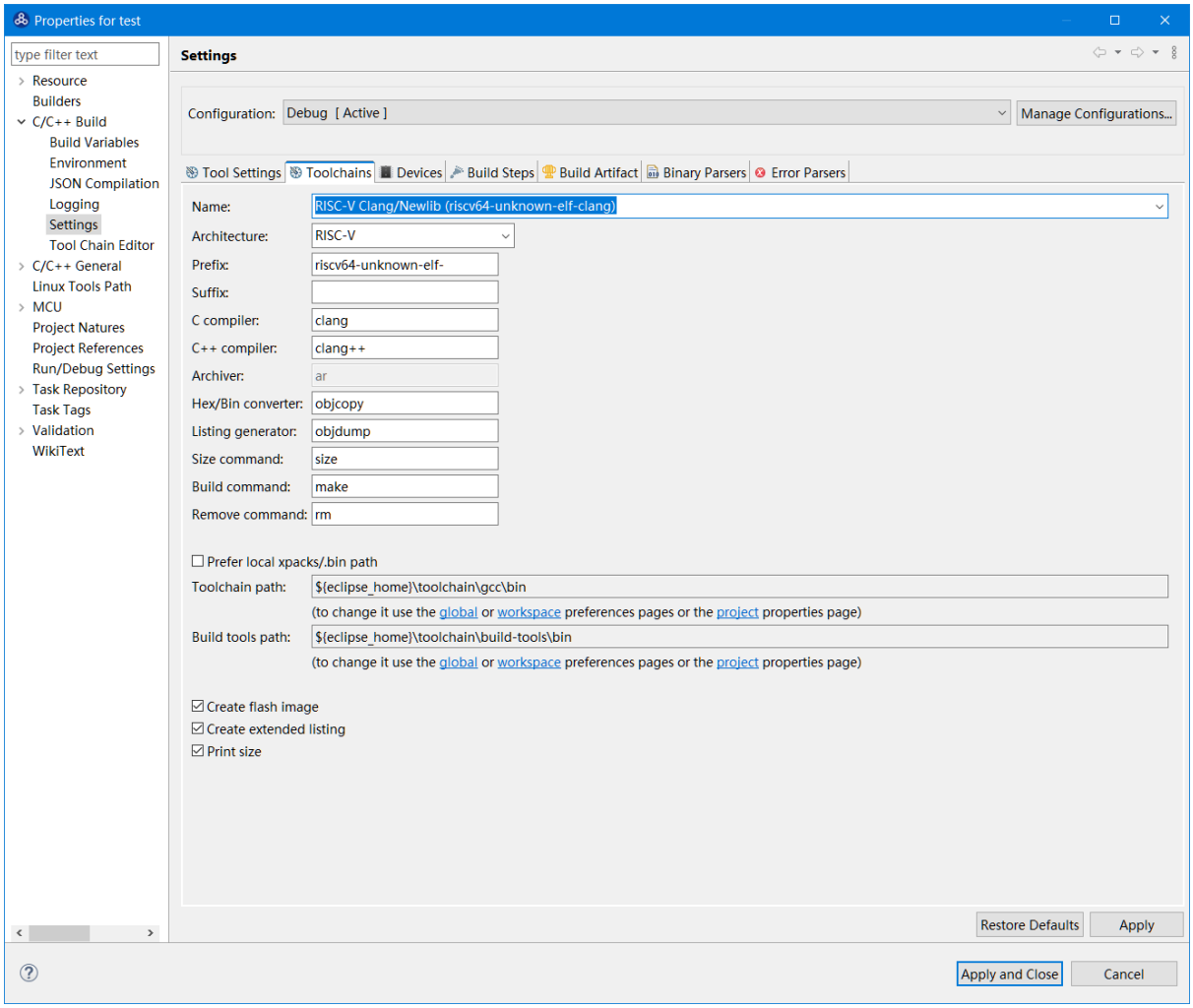
Nuclei LLVM Toolchain

LLVM Toolchain 是一套为 C 系列编程语言设计的完整工具链，旨在提供从源代码到可执行文件的编译和链接过程。LLVM Toolchain 的核心组件包括 Clang 编译器、LLVM 编译器基础设施以及相关的工具和库。LLVM Toolchain 是一套功能强大、灵活可扩展的编译工具链，它采用了先进的编译技术和设计理念，为开发者提供了高效、便捷的编译和构建解决方案。Nuclei Studio 2023.10 及之后的版本中，集成了 LLVM Toolchain。

在 nuclei_sdk 0.5.0 之后的版中，在创建工程时，用户可以选择 Toolchain 为 `RISC-V Clang/Newlib (riscv64-unknown-elf-clang)` 则可以创建一个支持 `Clang 17` 编译的工程，NucleiStudio 将默认将相对应的编译选项配置好。



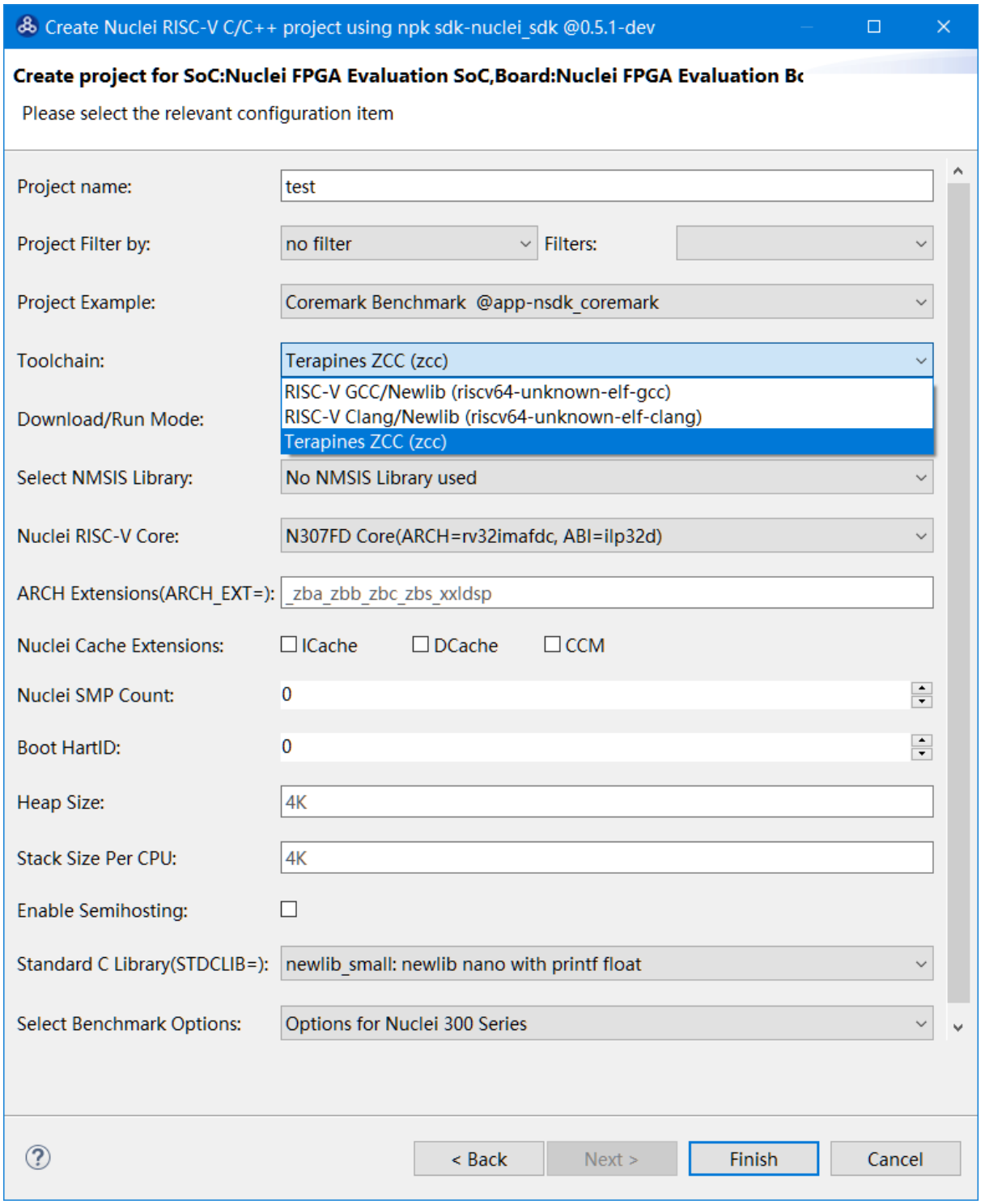
在工程上 右键->Properties->C/C++ Build->Setting->Toolchains 中可以看到工程所用到的 Toolchains 有一些工具，以及 Toolchains 所在的路径。



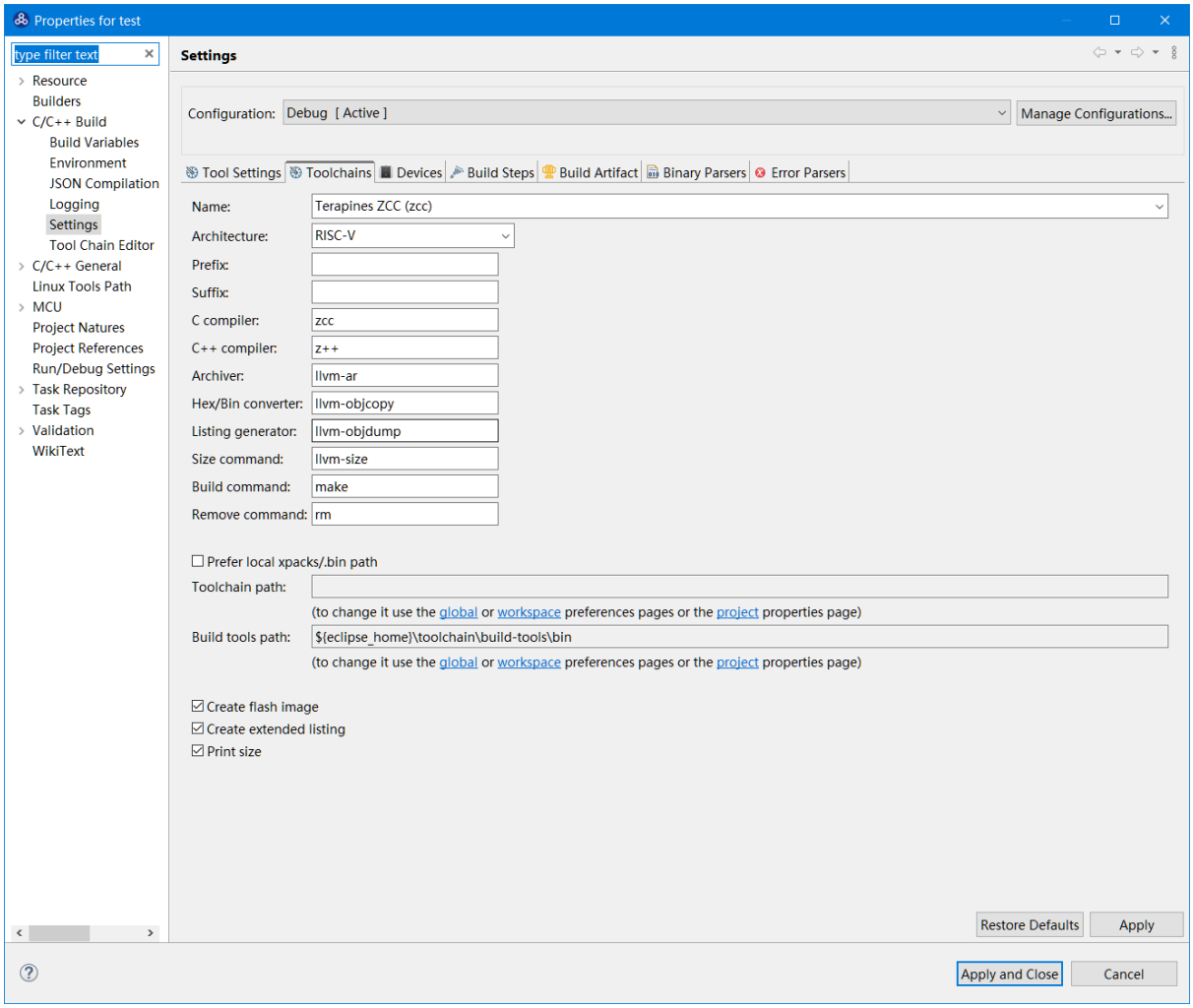
Terapines ZCC

Terapines ZCC 是兆松科技研发的高性能 RISC-V 编译器。Nuclei Studio 2024.06 版中对 Terapines ZCC 进行支持，集成了 **ZCC Lite** 版本的工具链，如果需要更新，可以自行下载好 Terapines ZCC 后，替换 `<NucleiStudio>/toolchain/zcc` 目录下的内容，可以在 NucleiStudio 中直接创建一个支持 Terapines ZCC 的工程，并使用 Terapines ZCC 进行编译。

关于 Terapines ZCC 参见：https://products.terapines.com/downloads



在工程上 右键->Properties->C/C++ Build->Setting->Toolchains 中可以看到工程所用到的 Toolchains 有一些工具，以及 Toolchains 所在的路径。在这里可以配置用户所下载的 ZCC 编译器的路径。

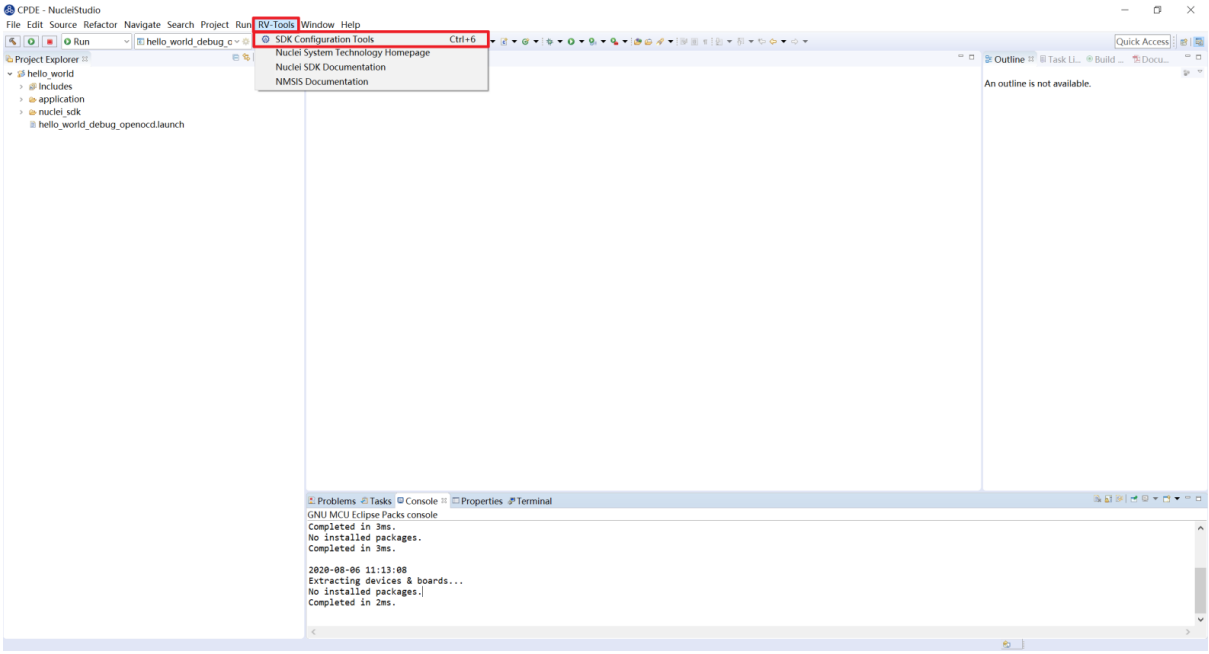


2.8.2 Nuclei SDK 工程设置工具

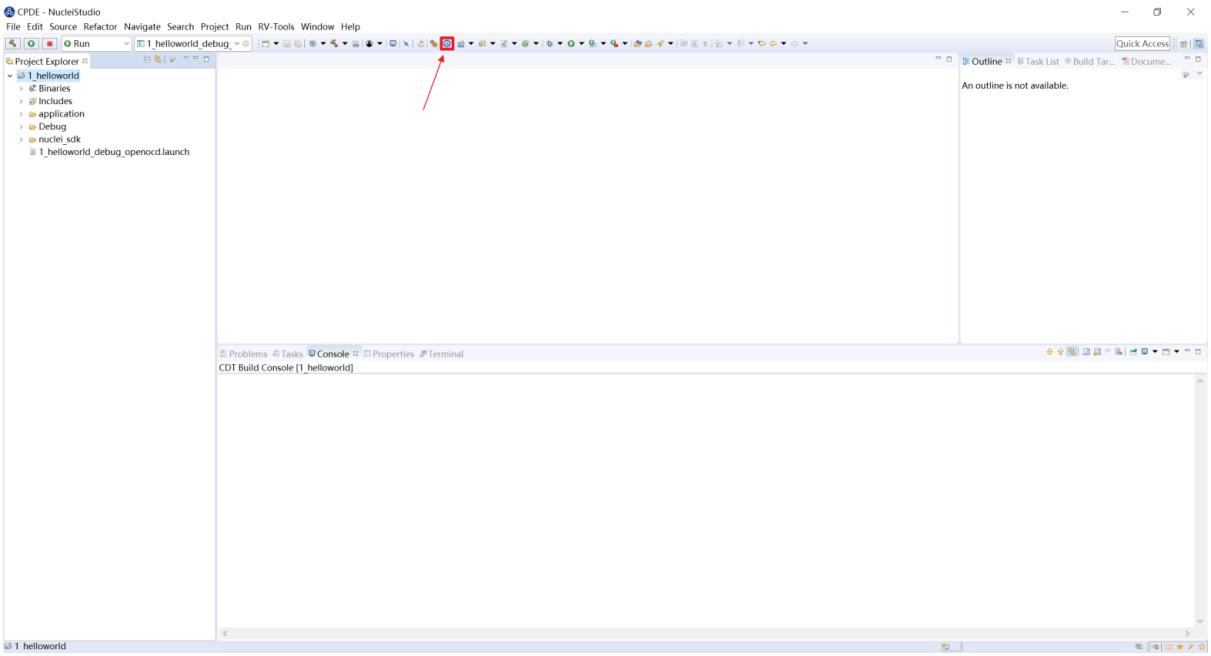
在 Nuclei Studio 中可以通过 Nuclei SDK 工程设置工具一键修改通过 Nuclei SDK Project Wizard 创建的工程的编译链接选项。

Note: 本工具目前仅支持 Nuclei SDK，HBird SDK，Nuclei Subsystem SDK, 不支持无模板手动创建的项目和基于 Makefile 创建的项目, 或者是自行创建维护的项目

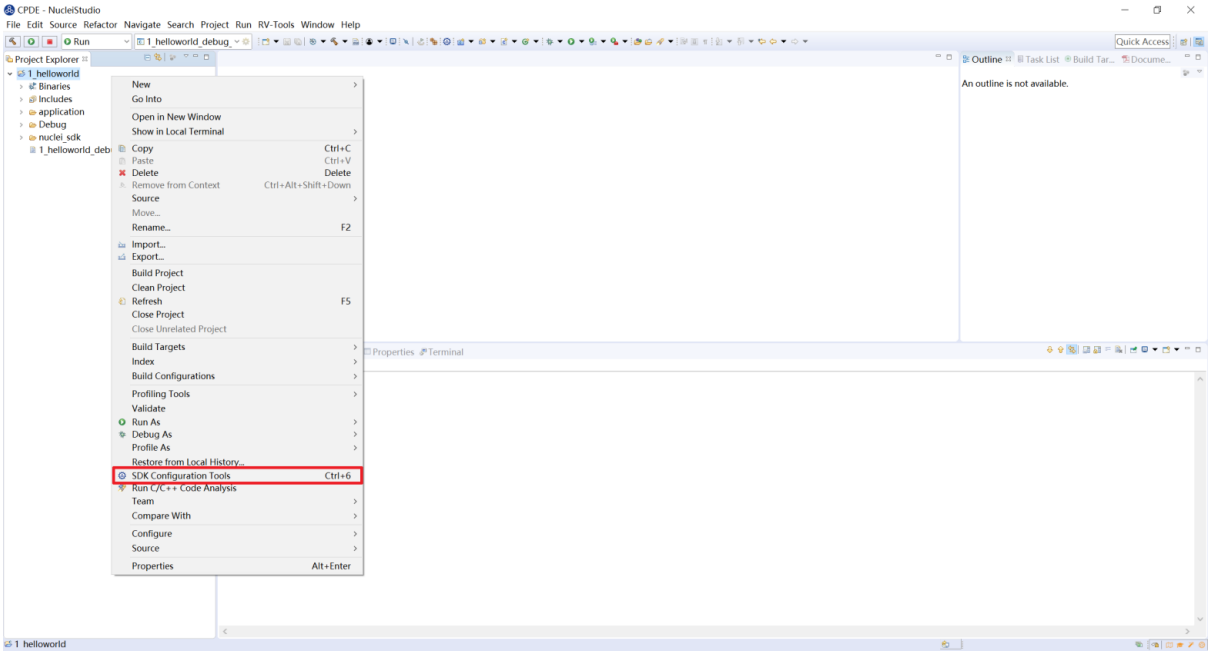
单击选中需要修改的工程，之后如图 6-1 打开 `RV-Tools>SDK Configuration Tools`，可以打开修改编译选项的弹窗，在 **2023.10** 版本以后，将直接打开 **Nuclei Settings** 页面。也可以单击要修改的工程后，点击工具栏的工程设置工具图标。



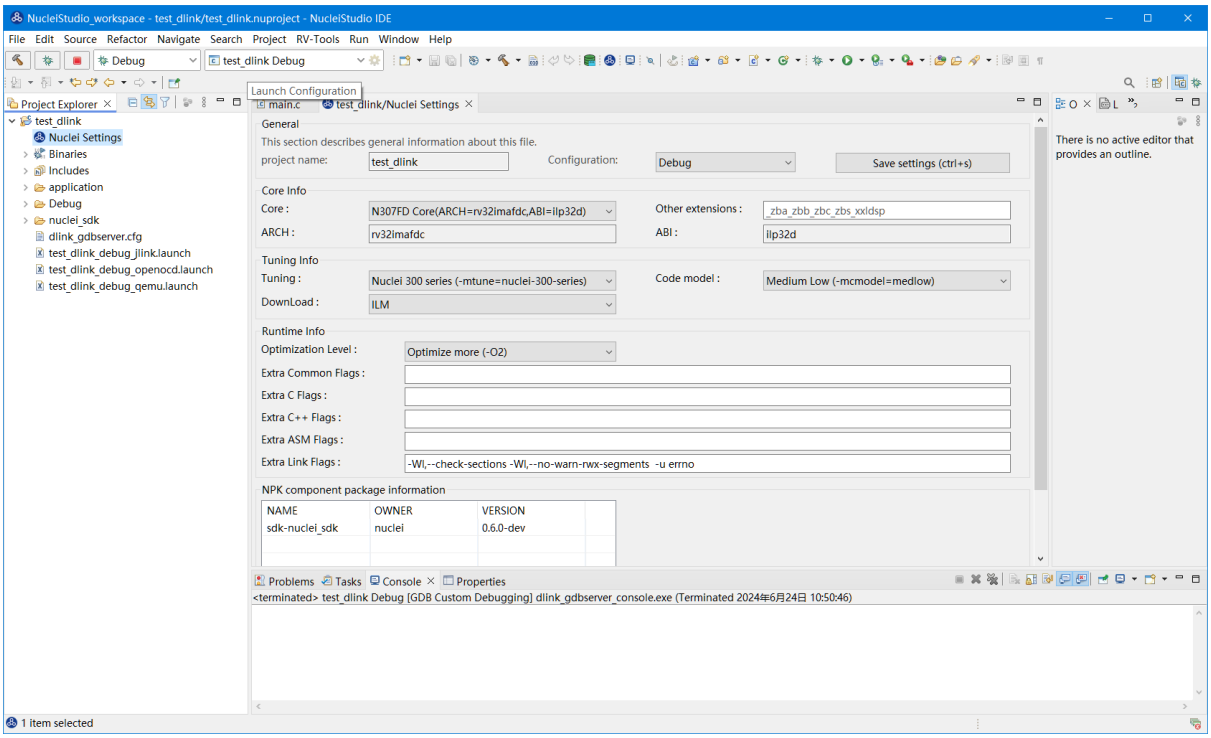
或者单击要修改的工程后，键盘按下 `ctrl+6` 。



也可以单击要修改的工程后，右击打开右键菜单，选择 SDK Configuration Tools。



SDK Configuration Tools 各选项详细功能如下：

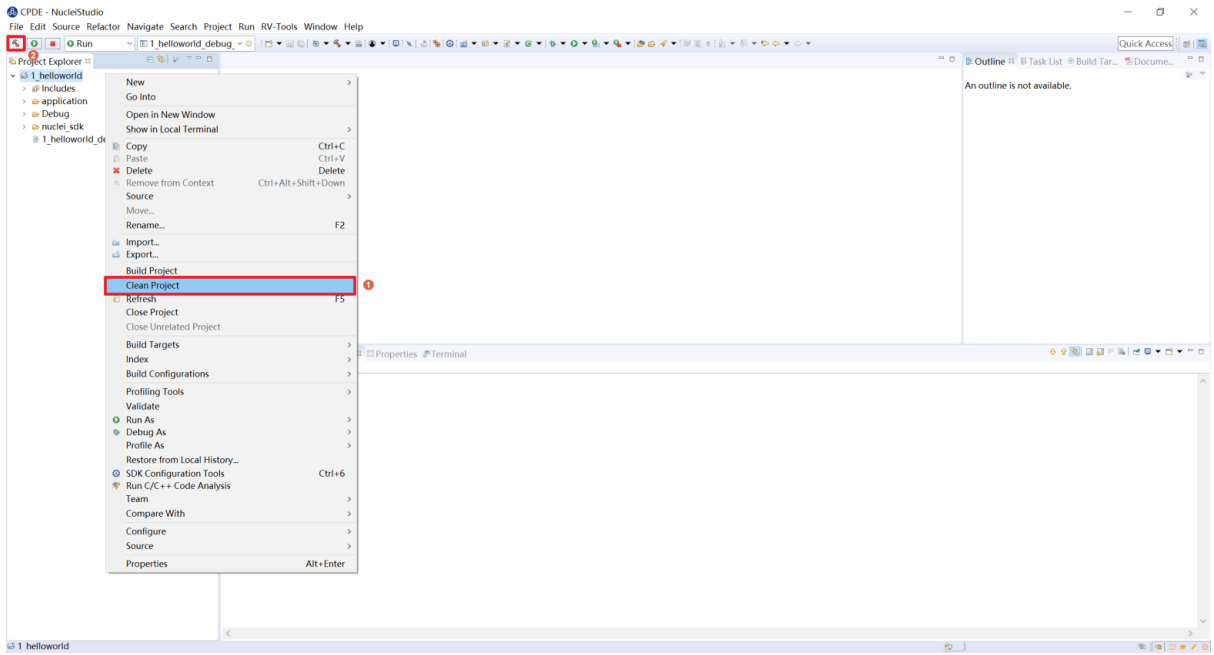


- projectName 为当前选中的工程名。
- Core 为当前工程对应的内核。由于工具根据 ARCH 和 ABI 选项反推出对应的内核，而不同的内核可能有相同的 ARCH 和 ABI 选项，所以显示上可能会有所偏差，只要 ARCH 与 ABI 为正确的选项即可。此选项为方便快速切换内核选项使用。
- 四个勾选项：`Bitmanipulation Extension(RVB)`，`Cryptography Extension(RVK)`，`Packed SIMD/DSP Extension`，`Vector Extension(RVV)` 用于选择对应的扩展指令集 (B/K/P/V)。

Note: 本功能在 2023.10 版本中移除，使用 `Other Extensions` 输入框来制定额外的扩展。

- ARCH 对应的当前工程的 arch 选项, 根据 Core 和勾选项自动组合。
- ABI 对应的当前工程的 abi 选项。
- Tuning 根据不同级别处理器优化的 gcc 选项，选择 Core 会自动选择正确的 Tuning 选项，不建议自己调整。
- Code Model 针对 RV32 处理器，自动选择为 `Medium Low`，而针对 RV64 处理器自动选择为 `Medium High`，选择 Core 以后会自动选择合适的 Code Model，其中 RV64 处理器必须使用 `Medium High`。
- Download 对应当前工程的下载模式，可以切换选择不同的下载模式，目前仅 Nuclei FPGA 评估开发板支持切换下载模式，RVSTAR 仅有 `FLASHXIP` 模式。其中切换到 flash 模式会额外定义 `VECTOR_TABLE_REMAPPED` 宏，其他模式不会定义这个宏
- `Select C Runtime Library` 对应的使用标准 C 库，本功能在 **2023.10** 版本中移除。在工程创建的时候，如果创建的工程采用的是 Newlib，则这里只能进行 newlib 版本的切换，如果创建的工程才用的是 `Nuclei C Runtime Library(libncrt)`，则这里只能进行 libncrt 版本的切换。
- `Optimization Level` 对应编译的优化等级。
- `Extra Common Flags` 对应的是额外的通用编译选项。可以添加额外的通用编译选项。
- `Extra C Flags` 对应的是额外的 C 编译选项。可以添加额外的 C 编译选项。
- `Extra C++ Flags` 对应的是额外的 C++ 编译选项。可以添加额外的 C++ 编译选项。
- `Extra ASM Flags` 对应的是额外的汇编编译选项。可以添加额外的汇编编译选项。
- `Extra Link Flags` 对应的是额外的链接选项。如果此选项已经有默认选项并且需要增加编译选项，可以在编译选项开头或结尾处相隔一个空格字符再增加编译选项。

根据需要修改以上的选项，这里我们修改优化等级为 `-Os` 优化生成可执行文件大小。点击 `save` 一键修改编译选项，`save` 以后一定要先 `clean project`，之后右击修改后的工程打开右键菜单，选择 `Clean Project` 清理一下工程，再点击锤子图标即可完成修改编译选项后重新编译工程。



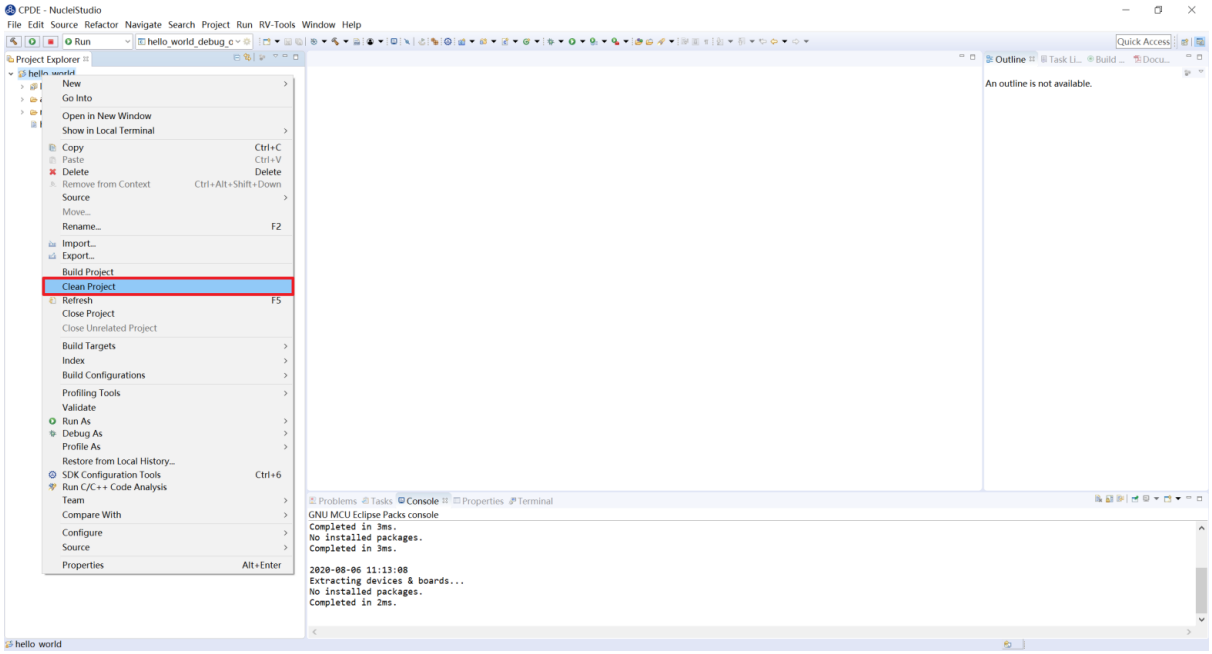
这里的 SDK Configuration Tool 切换不会对 Debug Configuration 选项做任何改动，因此如果切换了 Core 以后，对应的调试配置 (OpenOCD/QEMU/JLink) 也需要手动修改。

需要注意的是如果要切换工程从 32 位变为 64 位，需要打开调试设置页面，修改 `command` 中 `set arch riscv:rv32` 为 `set arch riscv:rv64`，从 64 位切换回 32 位也应当修改这里的参数为对应的数值。

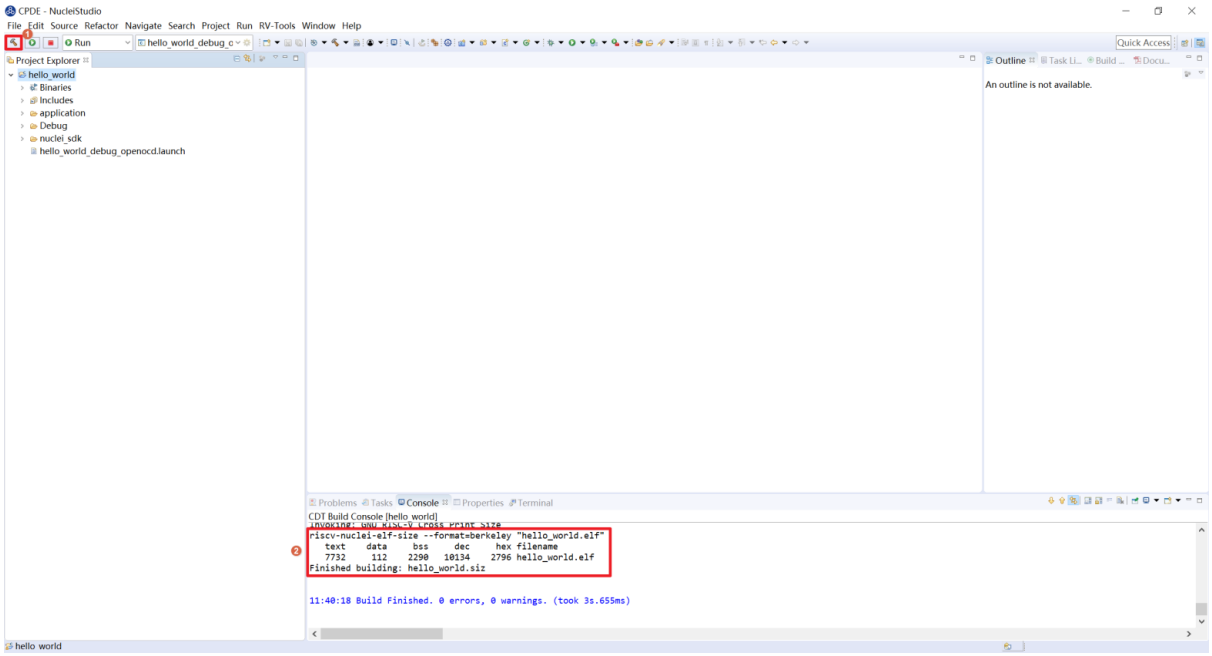
2.8.3 Nuclei Studio 中编译 Hello World 项目

在 Nuclei Studio 中编译 Hello World 项目的步骤如下。

在编译工程前，建议先将项目清理一下。在 Project Explorer 栏中选中 hello_world 项目，单击鼠标右键，选择 Clean Project 。



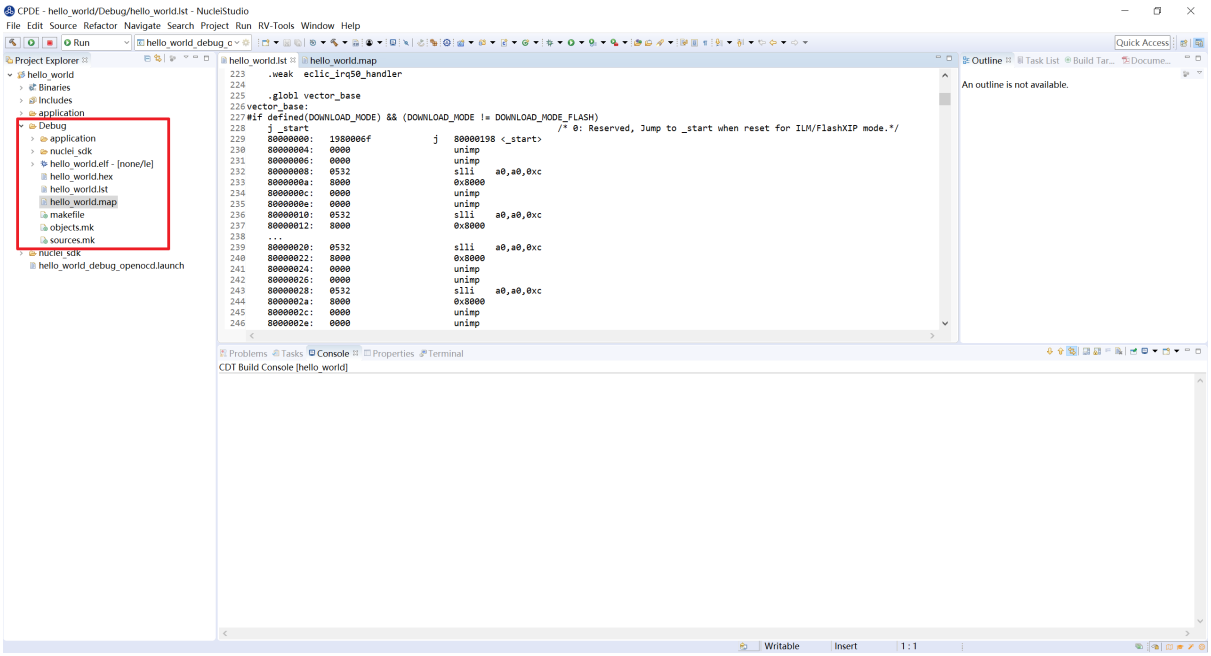
单击 Launch Bar 菜单上的锤子按钮，开始对项目进行编译。如果编译成功，能够看到生成可执行文件的代码体积大小，包括 text 段、data 段和 bss 段，以及总大小的十进制和十六进制数值。使用 Makefile 方式新建的工程需要在右键菜单中选择 Build Project 进行编译。



编译成功后可以看到增加了 Debug 文件夹，各文件作用如下：

- hello_world.elf 是生成的可执行文件。
- hello_world.hex 是生成的 Hex 文件。
- hello_world.lst 是生成的 list 文件，可以看到反汇编和简单的代码分部信息。

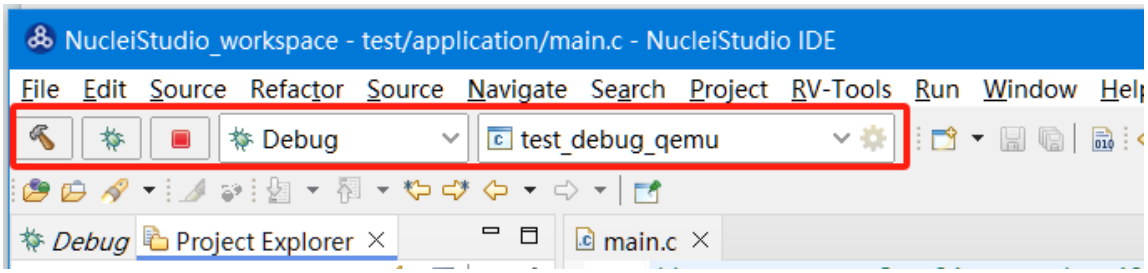
- `hello_world.map` 是生成的 map 文件，可以详细的看到生成的代码分布情况。



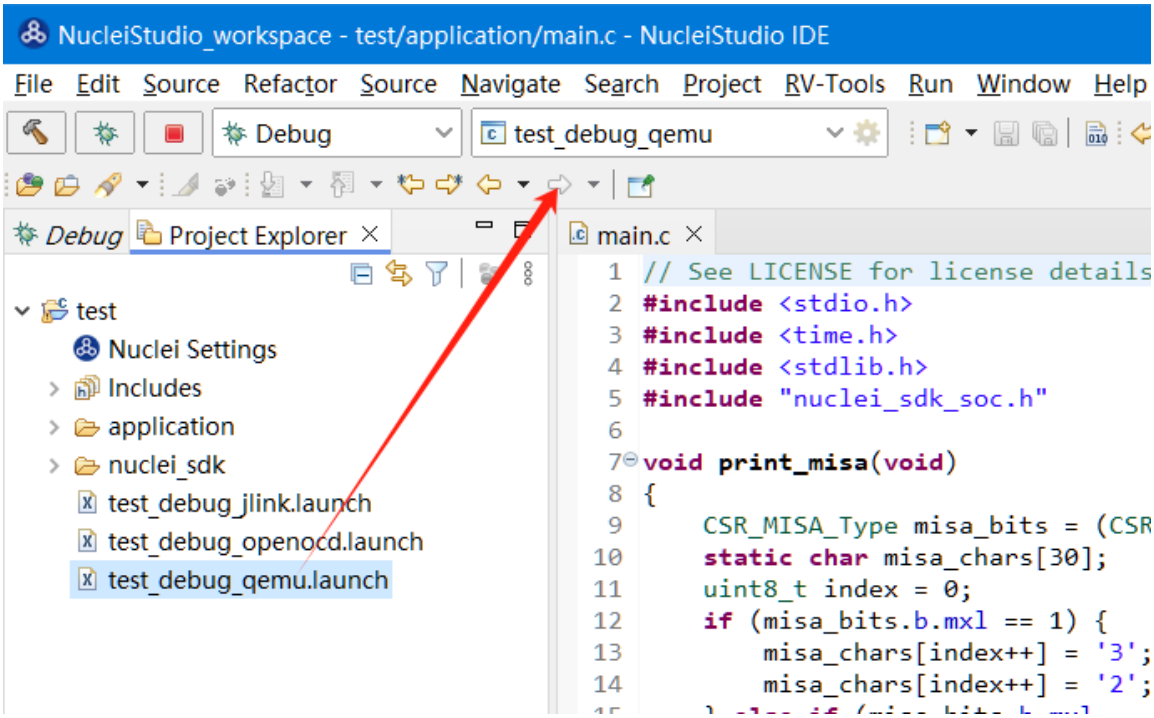
2.9 Nuclei Studio 调试运行工程

2.9.1 调试模式管理

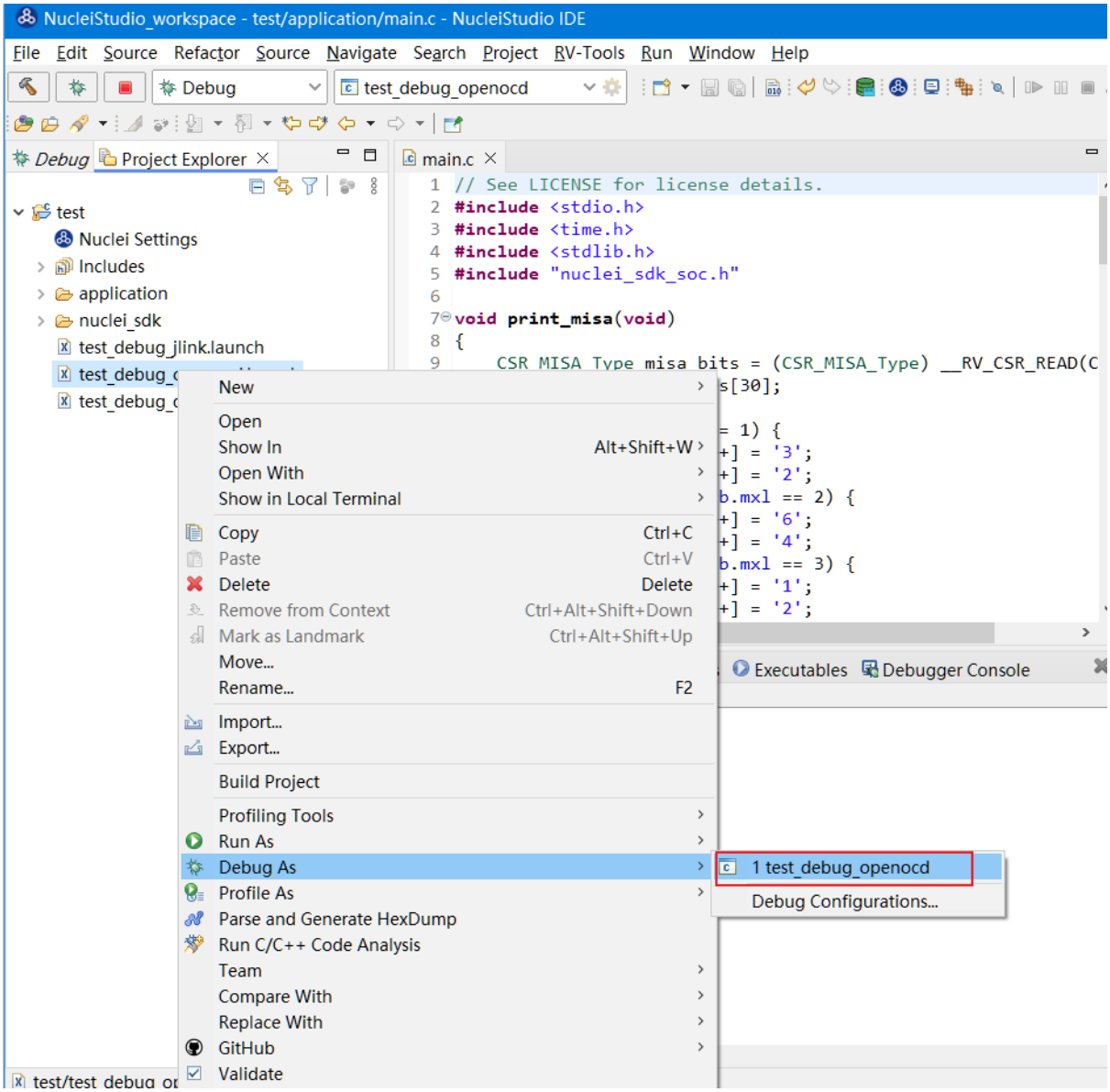
在 NucleiStudio 中，使用 Launch Bar 管理不同的调试器，默认情况下，NucleiStudio 会为 OpenOCD、Jlink、Qemu 生成对应的 `*.launch` 调试文件，NucleiStudio 识别到 `*.launch` 文件后，会将其加入到 Launch Bar 中进行管理，用户可以通过以下三种方式来使用指定的调试模式。



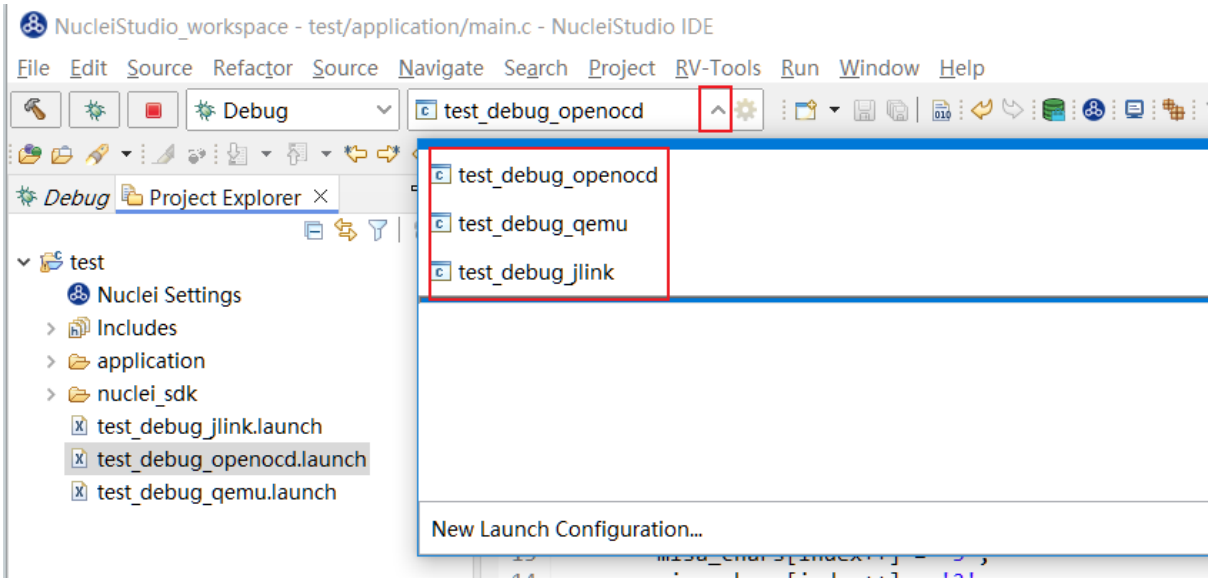
通过点击工程中的 `*.launch` 文件切换不同的调试模式，当用户单击工程中的某一个 `*.launch` 文件时，NucleiStudio 会将该文件设置为 Launch Bar 中的选中文件，然后就可以通过 Launch Bar 执行 Run/Debug 操作。



在工程展开文件，找到 *.launch 文件并在 *.launch 文件上点击鼠标右键，在弹出菜单中选中 Debug As/Run As，在下一级菜单中点对应的模式开始 Run/Debug 操作。



用户可以通过 Launch Bar 中的下接框，来切换成不同的调试模式，在 Launch Bar 中点击展开按钮，然后选中对应的调试模式，并执行 Run/Debug 操作。



2.9.2 使用蜂鸟调试器结合 OpenOCD 调试运行项目

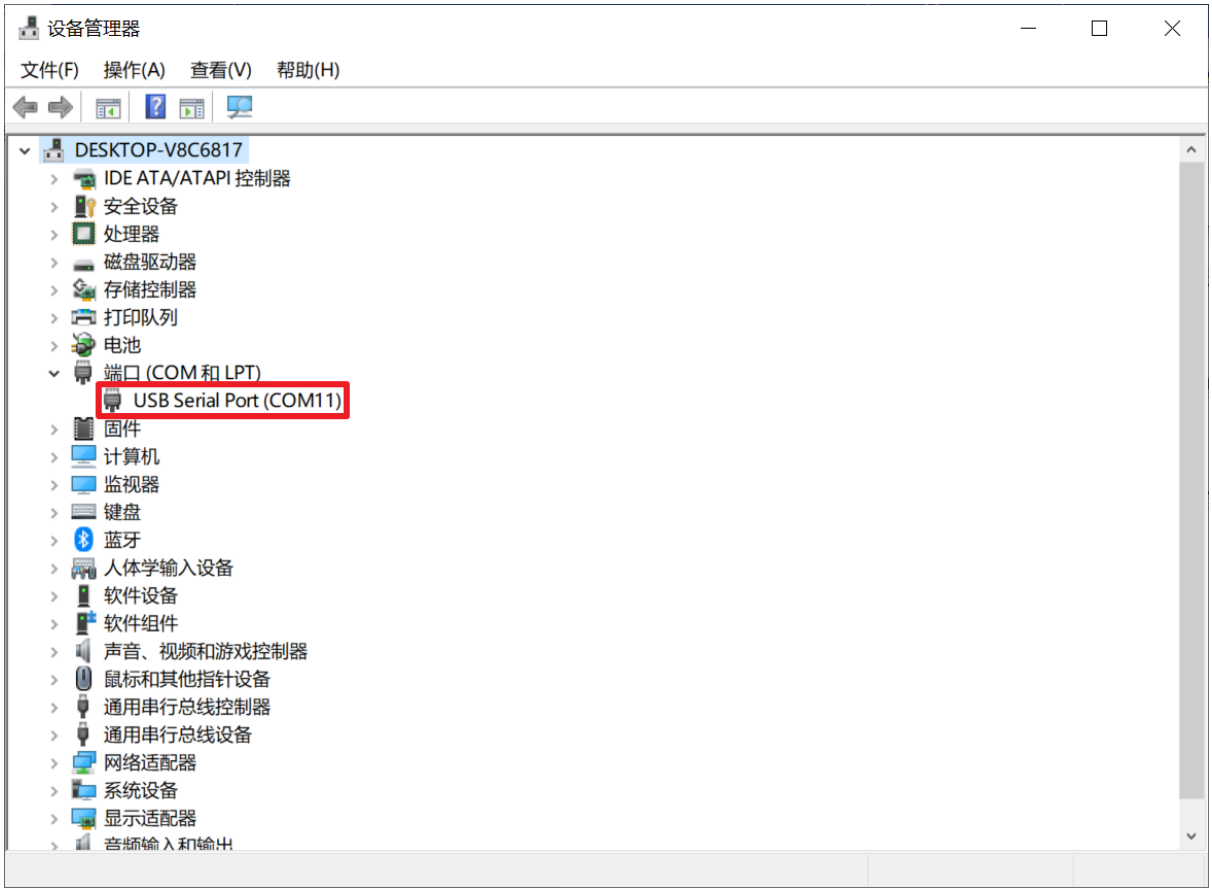
安装蜂鸟调试器驱动

Note: Nuclei Studio 自 2021.02 版本起 openocd 实现 windows 免驱功能，使用此版本及以上 windows 环境的用户可跳过此节，Linux 环境仍需配置驱动。低于此版本的用户需按照以下方法安装驱动。

在 Windows 系统中安装驱动

程序编译成功后，便可以将程序下载到 FPGA 原型开发板运行。首先将原型开发板与主机 PC 进行连接，步骤如下。

将蜂鸟调试器的一端插入主机 PC 的 USB 接口，另一端与原型开发板连接。由于蜂鸟调试器还包含了将原型开发板输出的 UART 转换成 USB 的功能，因此如果蜂鸟调试器被主机 PC 识别成功（且驱动安装成功），那么将能够被主机识别成为一个 COM 串口。在主机 PC 的设备管理器中的端口（COM 和 LPT）栏目中可以查询到该 COM 的串口号（譬如 COM11）。此串口在后续的程序运行过程中将充当原型开发板运行程序的 printf 输出显示接口。



Note: 注意：如果使用蜂鸟调试器，主机 PC 的 Windows 系统不能够识别蜂鸟调试器的 USB，需要安装驱动，可以在 https://www.nucleisys.com/developboard.php#debuggerkit 下载驱动程序并安装。

Note: 注意：在通过蜂鸟调试器下载之前，需要注意蜂鸟调试器被 Windows 正确识别，检验的标准即为本节中所述正确地安装了 HBird-Driver.exe 的驱动，且能够在设备管理器中查询到 COM 的串口号。

在 Linux 系统中安装驱动

在 Linux 环境下安装驱动步骤如下：

- 1：连接开发板到 Linux 中，确保 USB 被 Linux 识别出来。如果使用虚拟机，确保开发板连到了虚拟机当中。

- 2：在控制台中使用 lsusb 指令查看信息，参考的打印信息如下：

```
Bus 001 Device 010: ID 0403:6010 Future Technology Devices International, Ltd␣
↪FT2232xxxx
```

- 3：控制台中输入 sudo vi /etc/udev/rules.d/99-openocd.rules 指令打开 99-openocd.rules 文件，输入如下内容，保存退出。

```
SUBSYSTEM=="usb", ATTR{idVendor}=="0403",

ATTR{idProduct}=="6010", MODE="664", GROUP="plugdev"

SUBSYSTEM=="tty", ATTRS{idVendor}=="0403",

ATTRS{idProduct}=="6010", MODE="664", GROUP="plugdev"
```

- 4：断开调试器再重新连接到 Linux 系统中。
- 5：使用 `ls /dev/ttyUSB*` 命令查看 ttyUSB 信息，参考输出如下：

```
/dev/ttyUSB0

/dev/ttyUSB1
```

- 6：使用 ls -l /dev/ttyUSB1 命令查看分组信息，参考输出如下:

```
crw-rw-r-- 1 root plugdev 188, 1 Nov 28 12:53 /dev/ttyUSB1
```

可以看到 ttyUSB1 已经加入 `plugdev` 组，接下来我们要将自己添加到 plugdev 组（不同环境可能名字不同，请根据实际情况修改）。使用 whoami 命令查看当前用户名，我们将其记录为 `< your_user_name >` 。

- 7：使用 `sudo usermod -a -G plugdev <your_user_name>` 命令将自己添加进 plugdev 组。加入以后一定要重启或者注销操作系统。

- 8：再次确认当前用户名已属于 plugdev 组，使用 groups 命令，可以看到打印信息中有 plugdev 即成功将当前用户添加至 plugdev 组。如果没有可以尝试重启。
- 9：查看 gcc 的依赖是否完整，如果有依赖需要安装，可以执行 sudo apt install libncurses5libtinfo5 进行安装

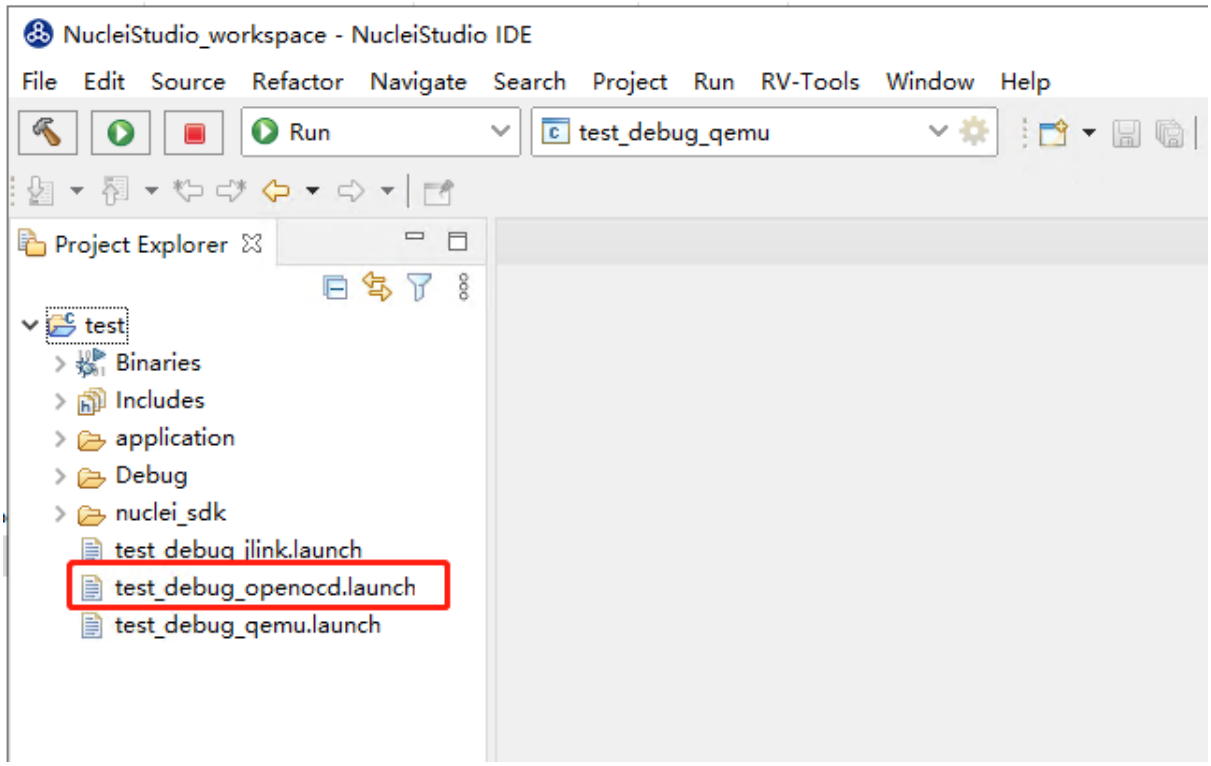
```
cd Nuclei Studio/toolchain/gcc/bin/

ldd ./riscv-nuclei-elf-gdb
```

```
linux-vdso.so.1 (0x00007fffc83f3000)
libtinfo.so.5 => not found
libncursesw.so.5 => not found
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f4df6d08000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f4df6c21000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f4df6c1c000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f4df69f2000)
/lib64/ld-linux-x86-64.so.2 (0x00007f4df6d1e000)
```

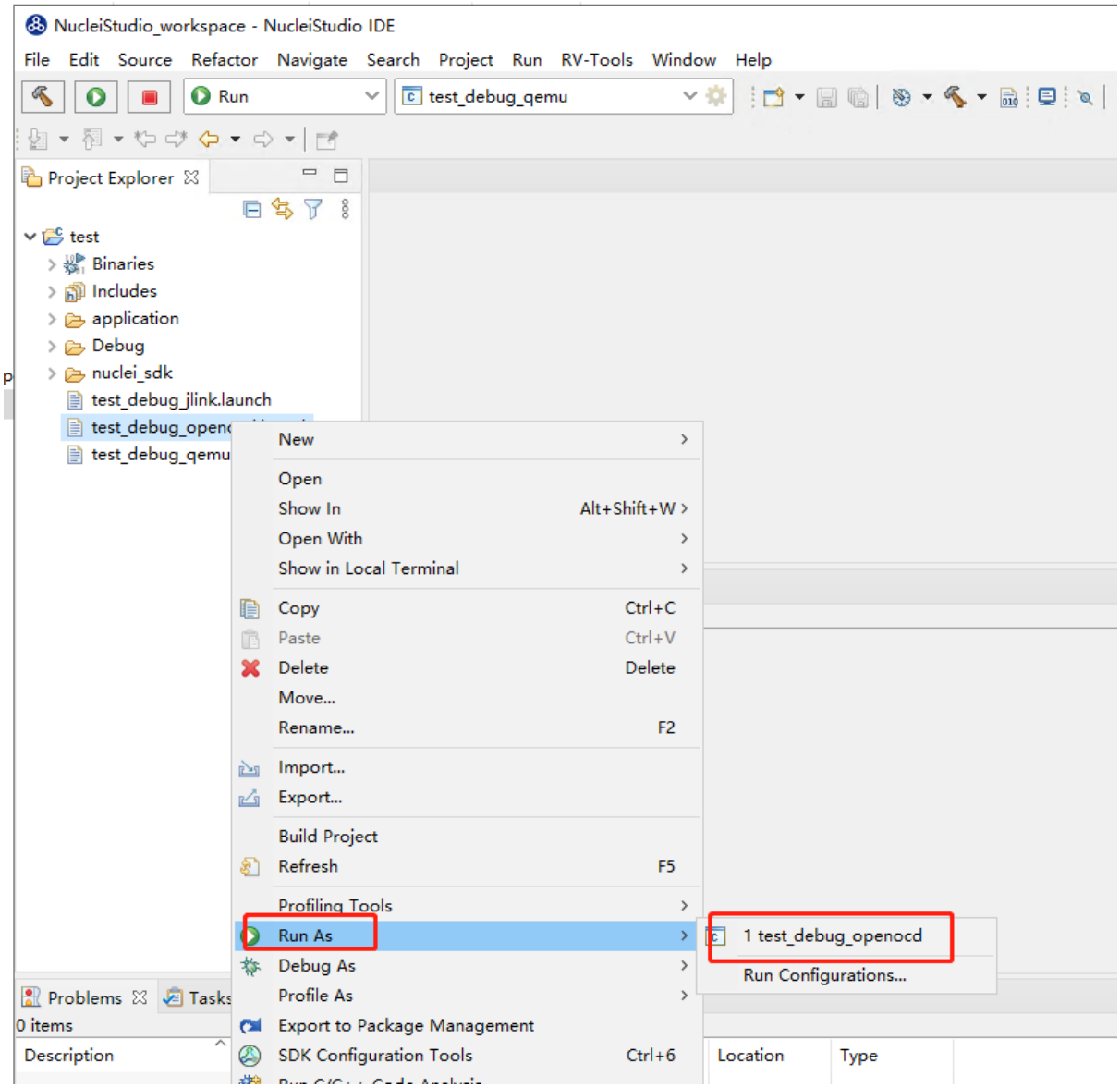
Debug Configuration

使用 Nuclei Studio 生成的 Debug Configuration

为了方便用户调试，Nuclei Studio 在创建工程时，会根据 NPK 的配置，默认的生成 Debug Configurations 的 Launch 文件。

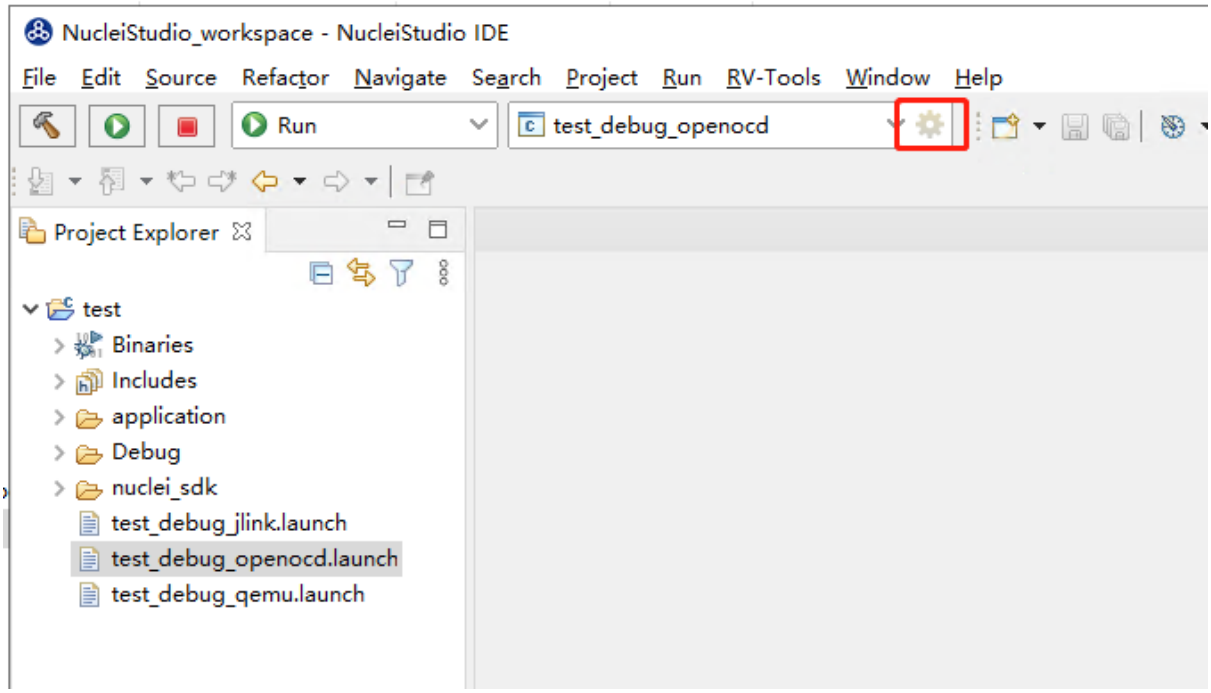


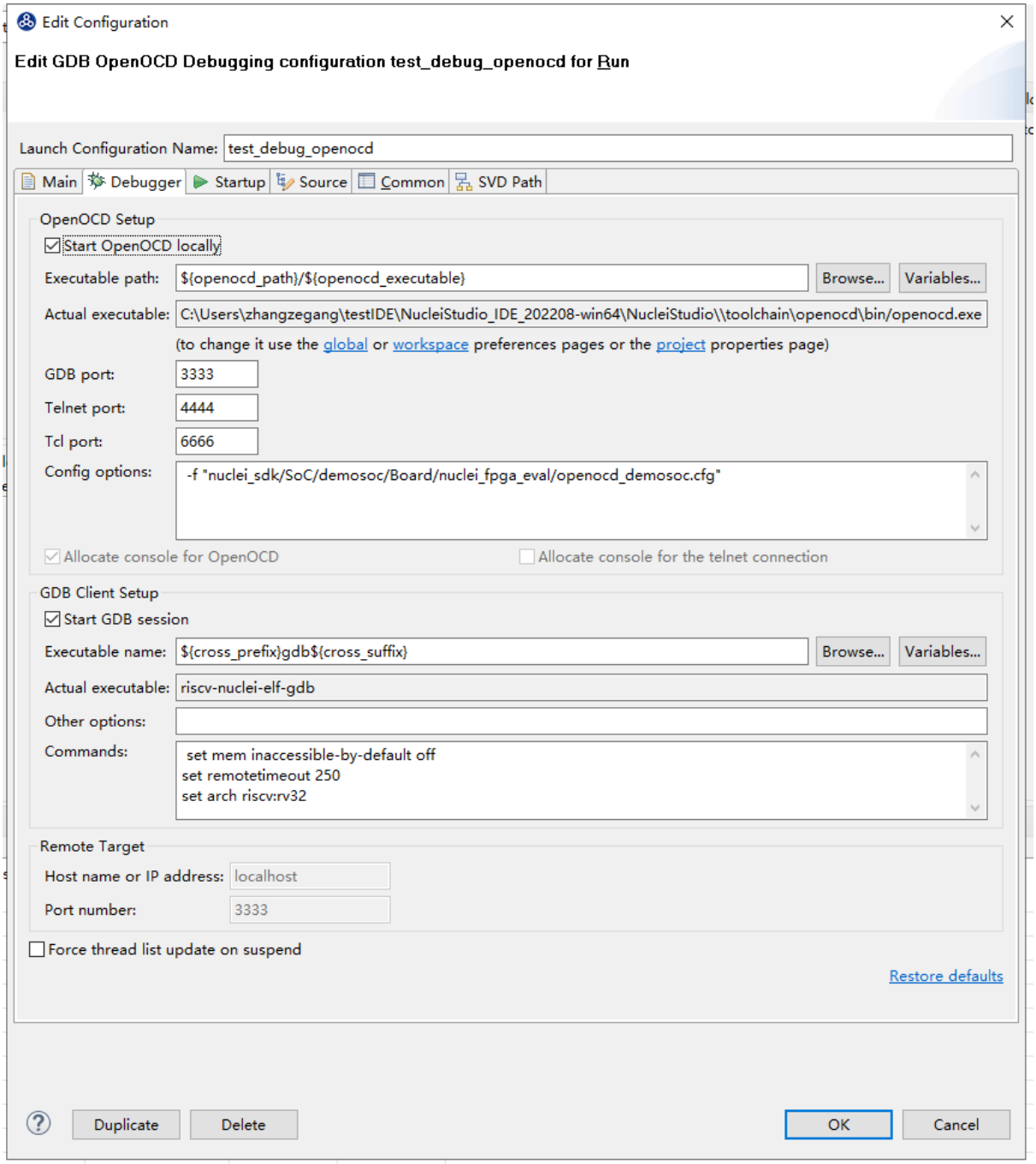
用户可以展开工程，选中对应的 test_debug_openocd.launch 文件，在右键菜单中，可以 Run as/Debug as->test_debug_openocd，就可以按照对应的 Debug Configurations 操作工程程了。



Note: 注意：配图可能没有及时更新，导致图文不一致，以文字为准，结合对应版本进行使用。

具体的 Debug Configurations 的内容可以在 Launch Bar 中进行详情查看。

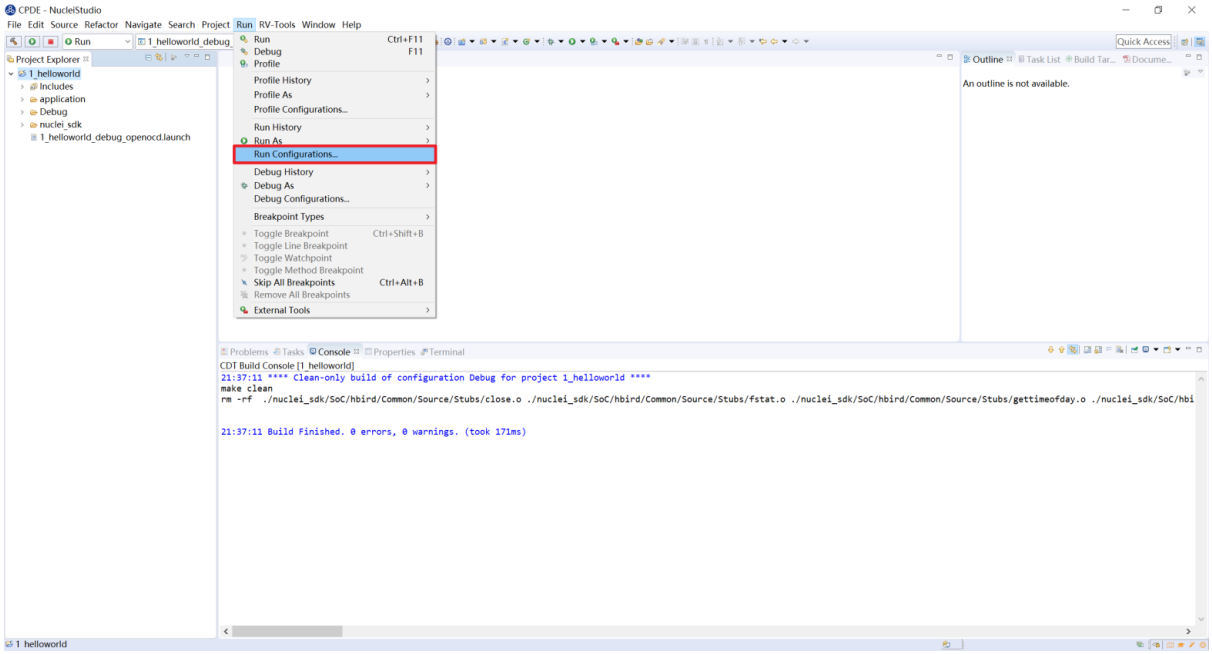

NucleiStudio_workspace - NucleiStudio IDE
File Edit Source Refactor Navigate Search Project Run RV-Tools Window Help
Run
test_debug_openocd
Project Explorer
test
Binaries
Includes
application
Debug
nuclei_sdk
test_debug_jlink.launch
test_debug_openocd.launch
test_debug_qemu.launch



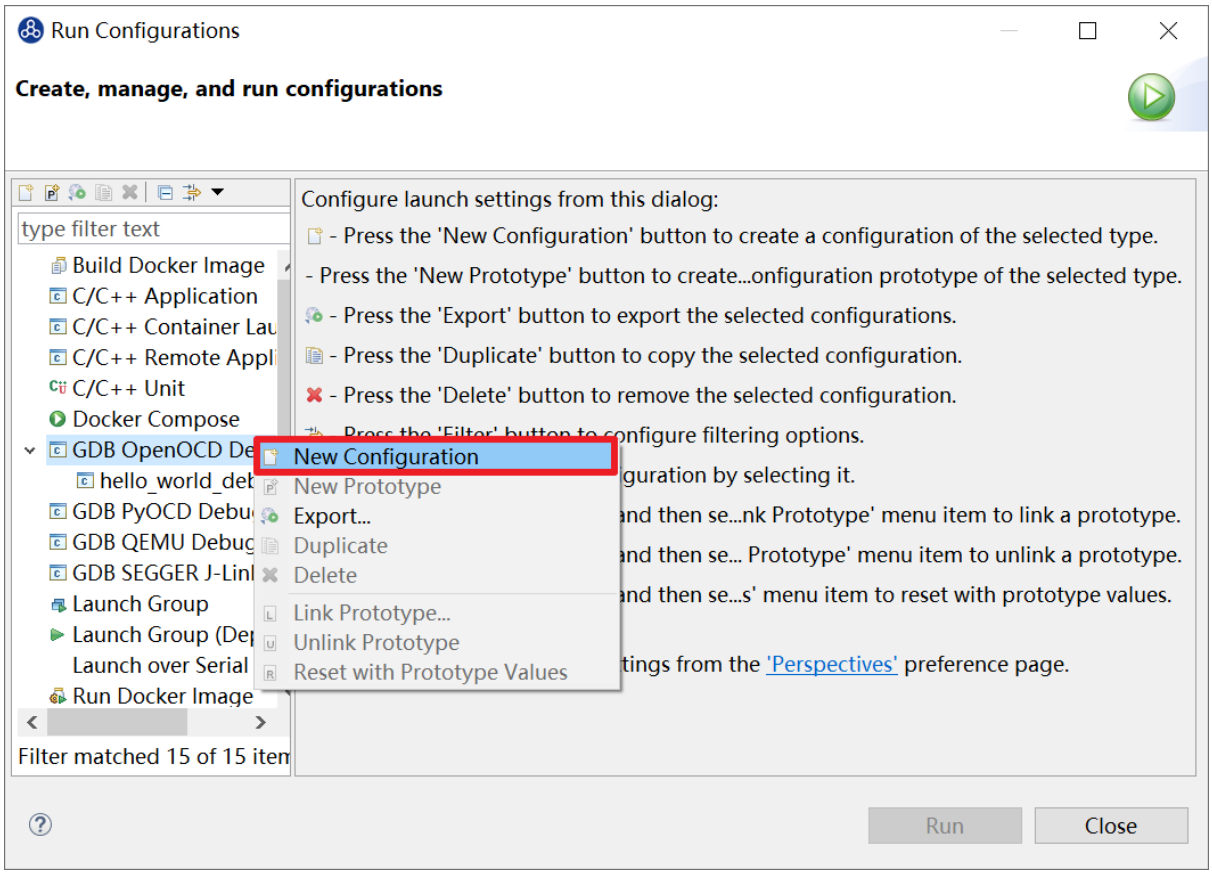
新建并配置 Debug Configuration

通过 Nuclei Studio 新建并配置 Debug Configuration 内容的步骤如下。

在 Nuclei Studio 的主菜单栏中选择 `Run—>Debug Configurations` 。



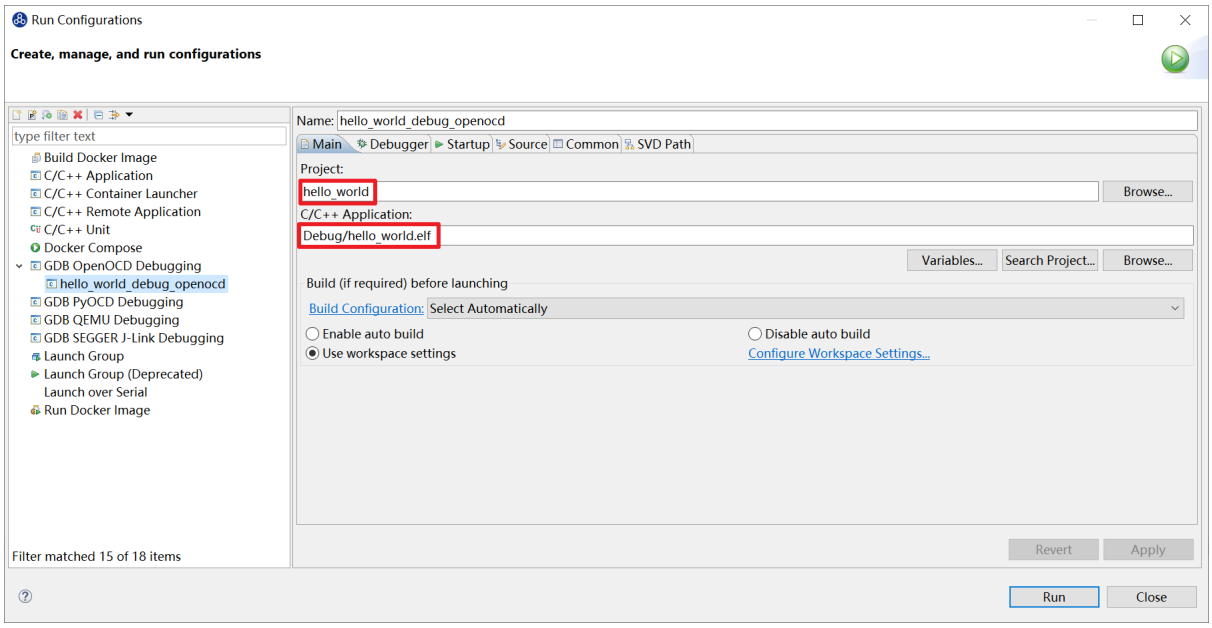
在弹出的窗口中，如果没有当前工程的调试设置内容，右键单击 GDB OpenOCD Debugging，选择 New，将会为本项目新建出一个调试项目 hello_world_demo Debug。确保 Project 是当前需要调试的工程，C/C++ Application 中选择了正确的需要调试的 ELF 文件。



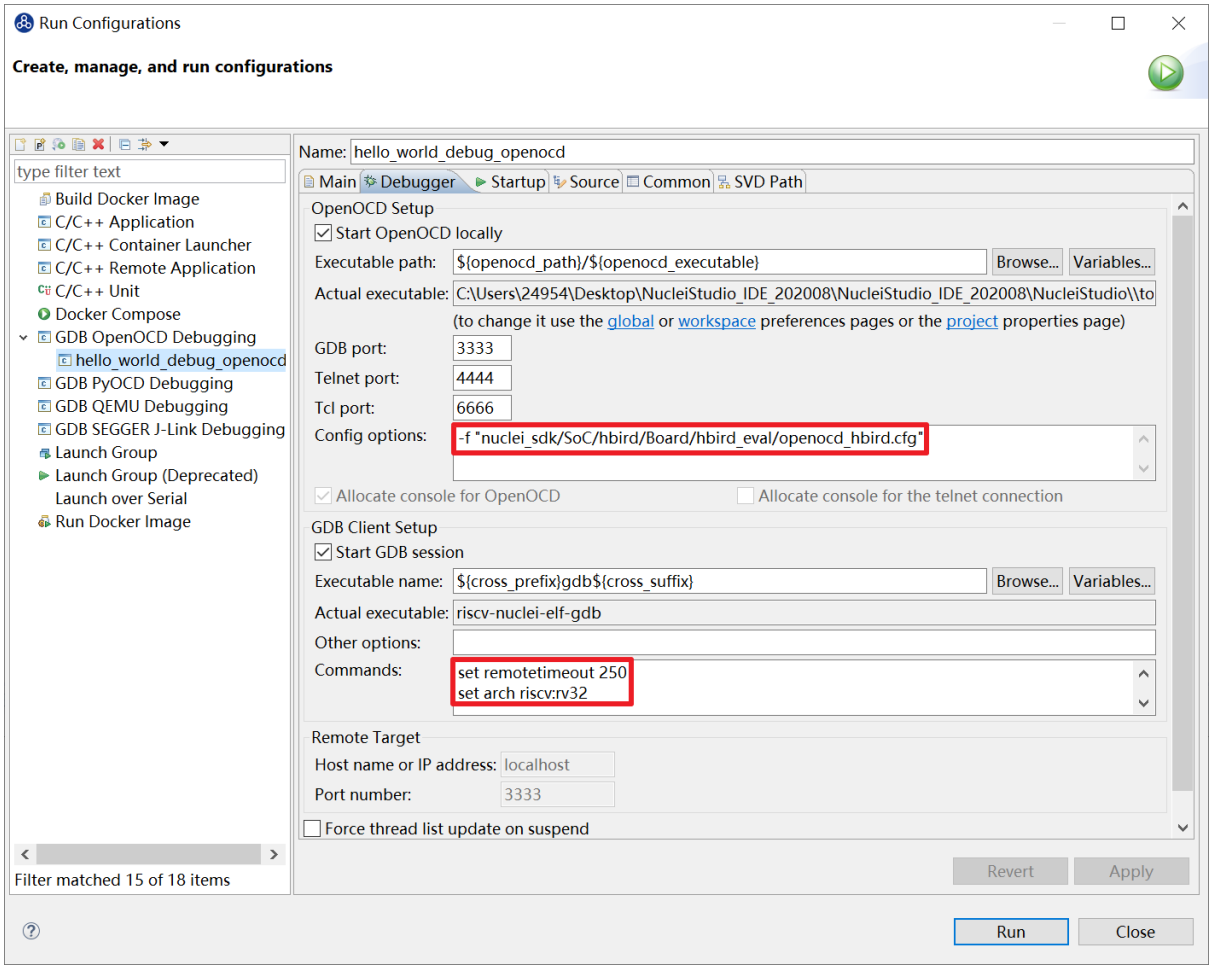
选择调试项目 hello_world_demo Debug 的 Debugger 菜单，在 Config options 栏目中填入 -f "nuclei_sdk/SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg"，以确保 OpenOCD 使用正确的配置文件。这里的配置文件 (*nuclei_sdk/SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg*) 根据实际工程中 openocd 的配置文件路径而定。例如：如果使用 makefile 方式导入工程，修改此处的内容为 -f "SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg"。

如果当前内核是 RISC-V 32 位内核，请确保 Commands 内容包含 `set arch riscv:rv32`

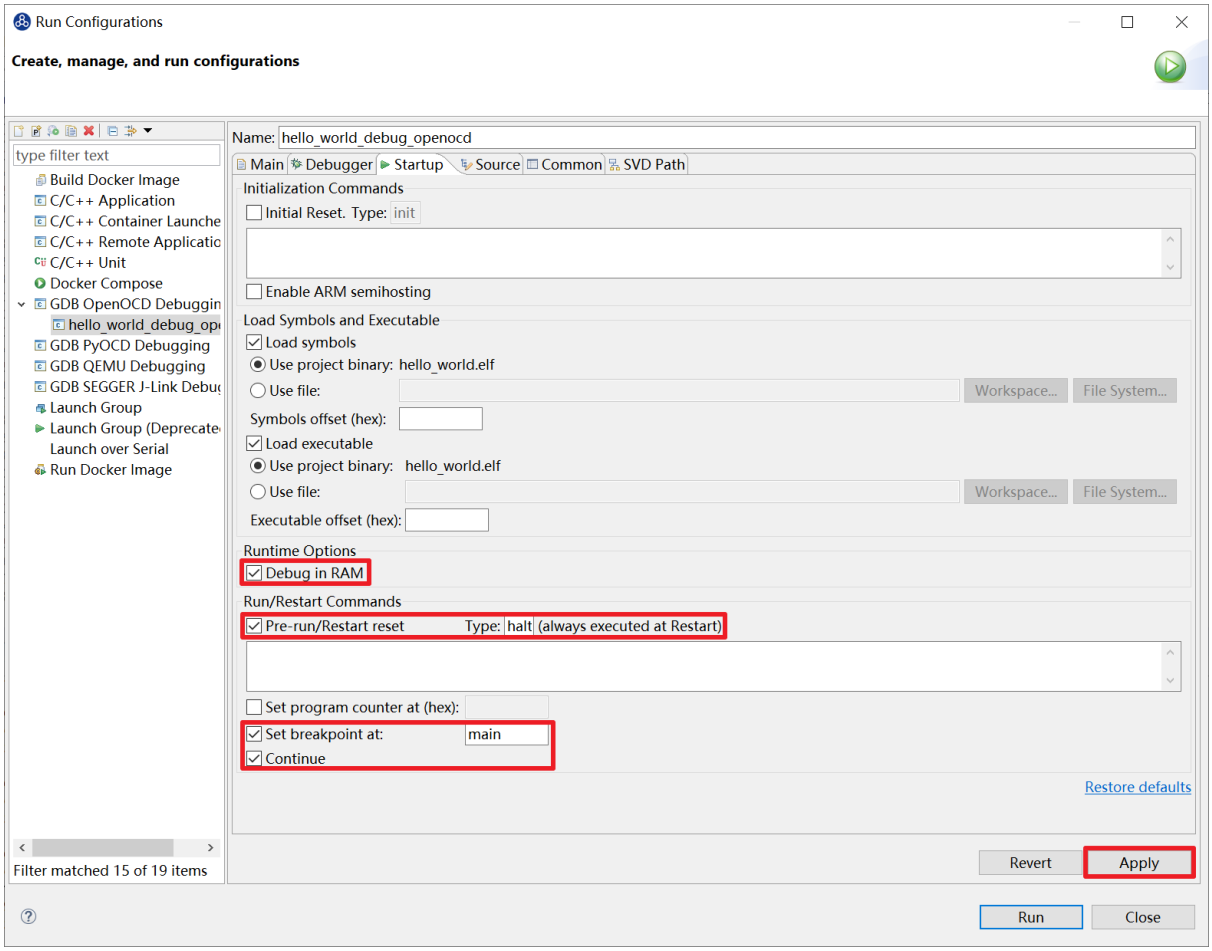
如果当前内核为 64 位，应确保替换为 `set arch riscv:rv64`



选择调试项目 `hello_world_demo Debug` 的 Startup 菜单，确保 `Debug in RAM`，`Pre-run/Restart reset`，`Set Breakpoint at Main` 和 `Continue` 被勾选。



完成配置后点击右下方 `Apply` 保存设置。

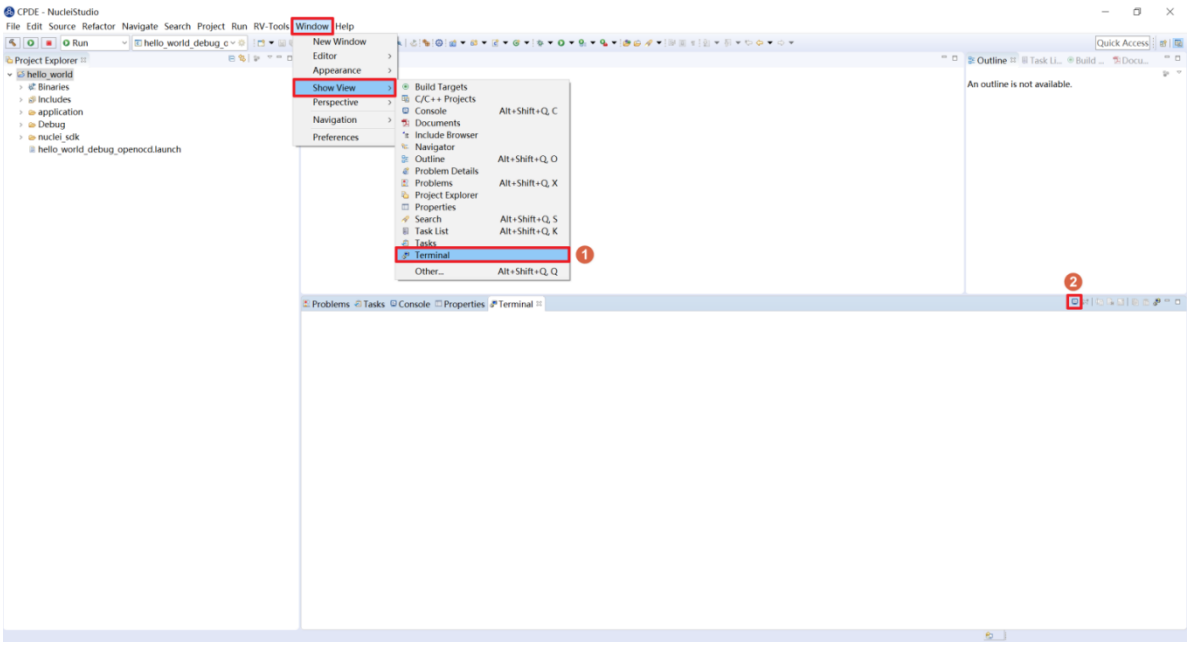


在原型开发板上调试程序

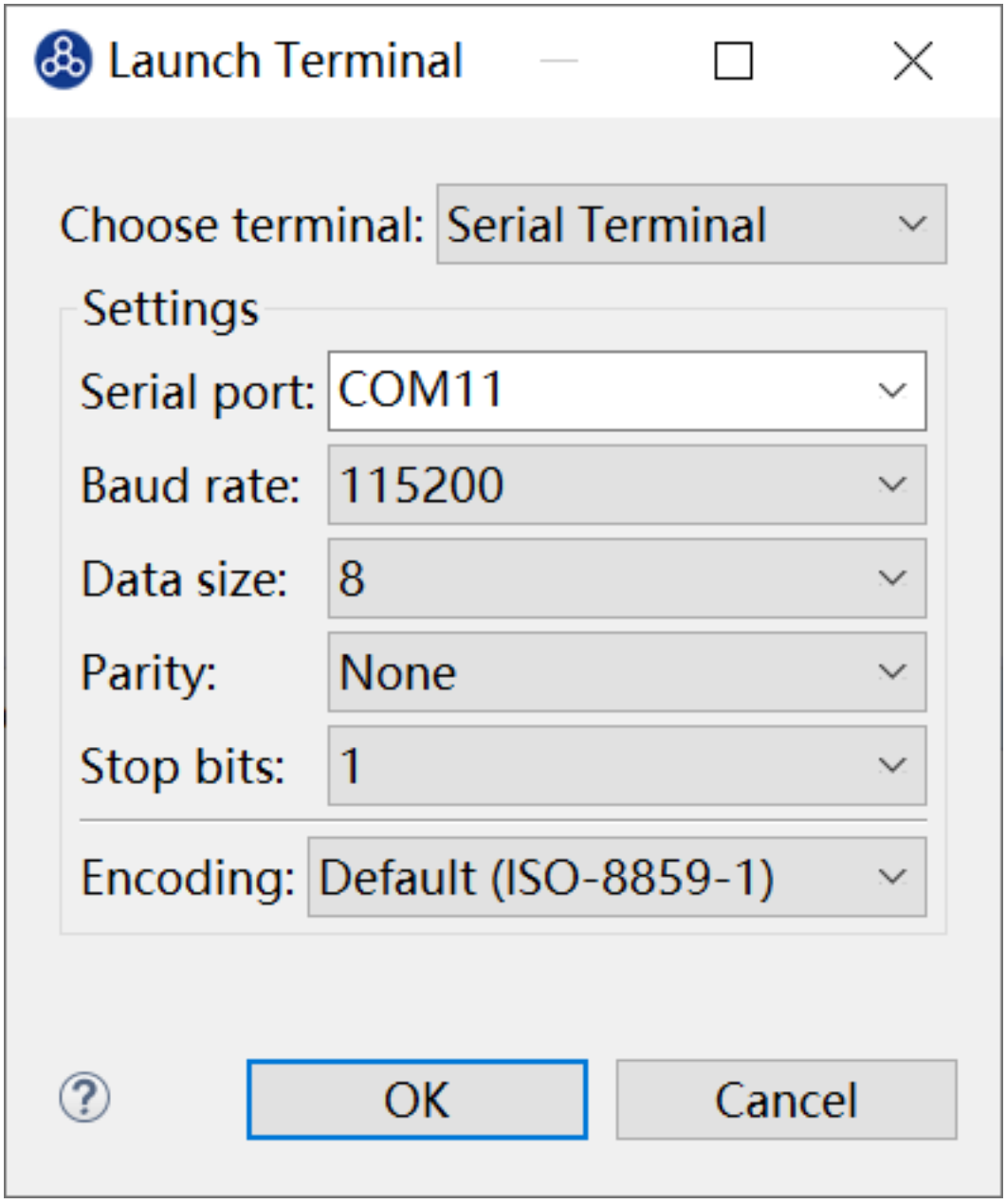
在开发板上调试之前，需要打开串口以便观察 Printf 函数打印信息。

使用 Windows 系统打开串口的方法如下：

打开 Nuclei Studio 自带的串口打印通道，选择 `Window>Show View>Terminal`，如图 7-13 所示，点击显示器图标打开串口设置选项。

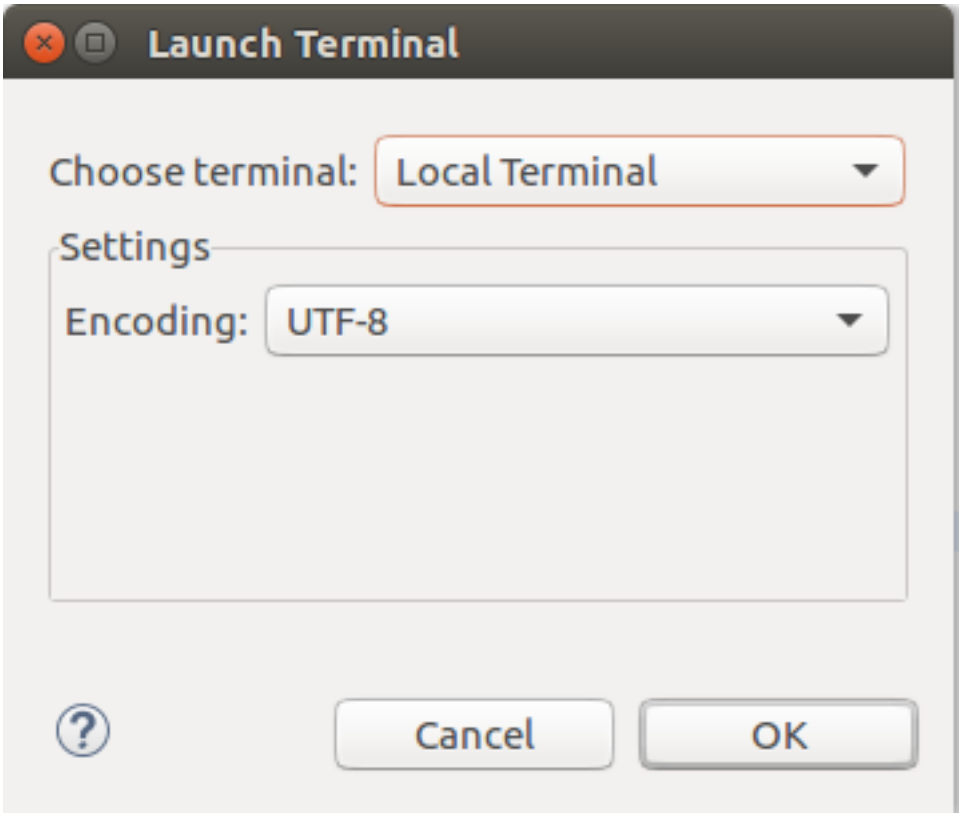


在其窗口中设置 Choose terminal（选择串口，即 Serial Terminal）、串口号（这里以 COM11 为例）、波特率（设置为 115200）等参数后，单击 OK 按钮。

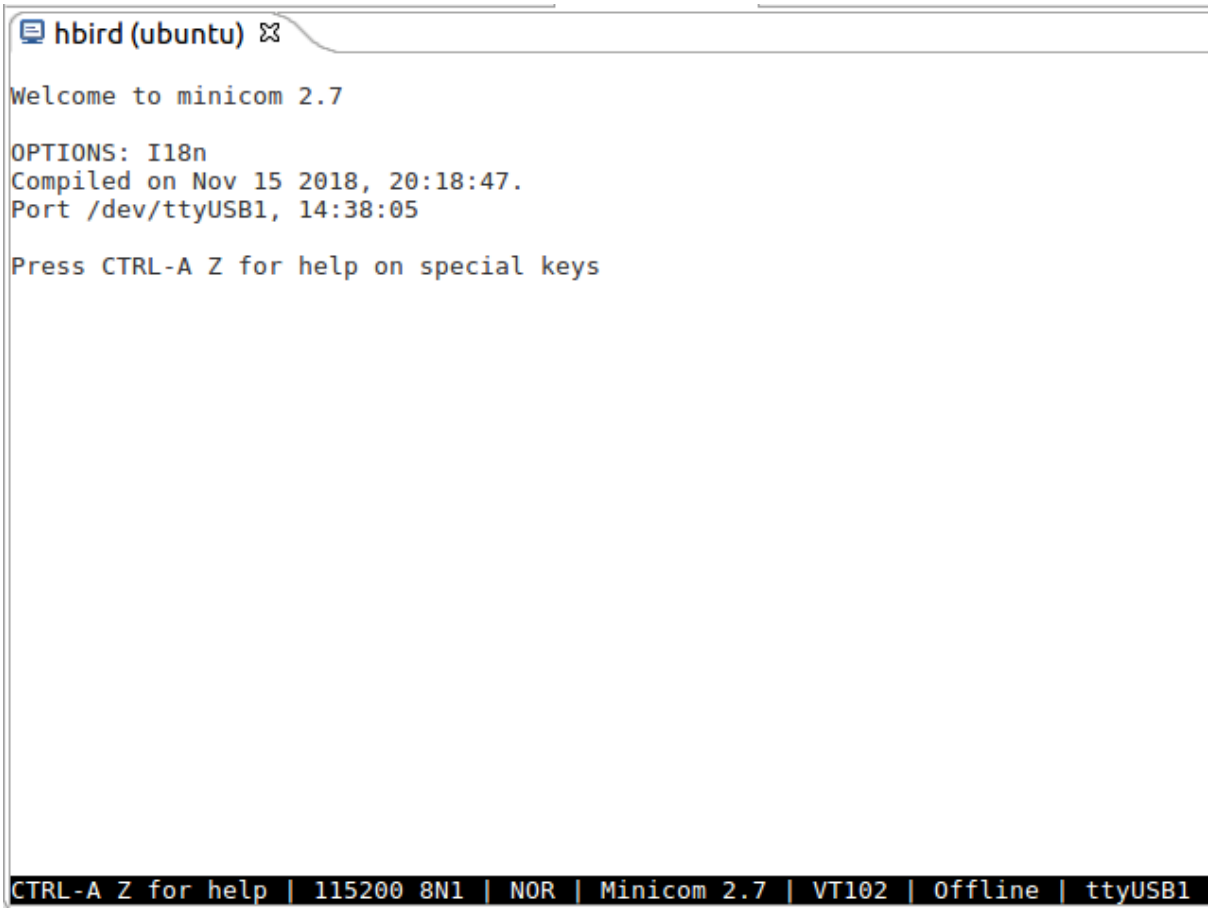


使用 Linux 系统打开串口方法如下：

打开 Nuclei Studio 自带的 Terminal 终端，选择 `Window>Show View>Terminal`，点击显示器图标打开串口设置选项。choose terminal 选择 `Local Terminal`，点击 OK 打开 Terminal 终端。



在框口中输入 `minicom /dev/ttyUSB1 115200` 打开串口，即可在 Nuclei Studio 中查看串口打印信息。



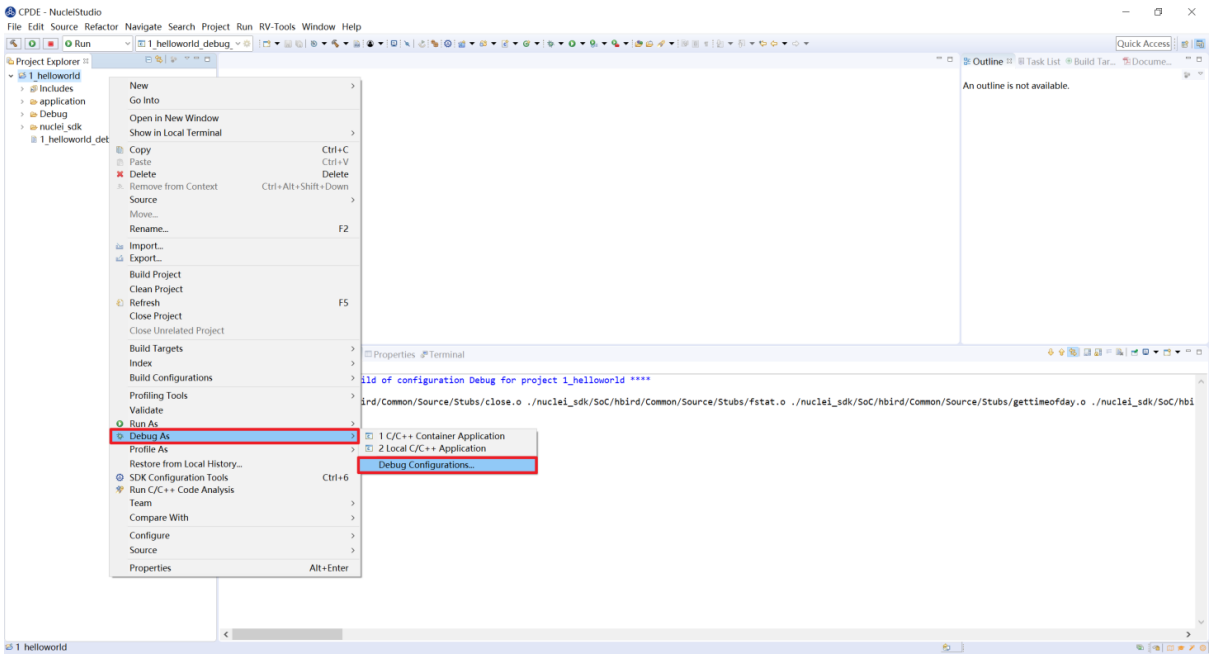
如果程序员希望能够调试运行于原型开发板中程序，可以使用 Nuclei Studio IDE 进行调试。由于 IDE

运行于主机 PC 端，而程序运行于原型开发板上，因此这种调试也称为 在线调试或者 远程调试。

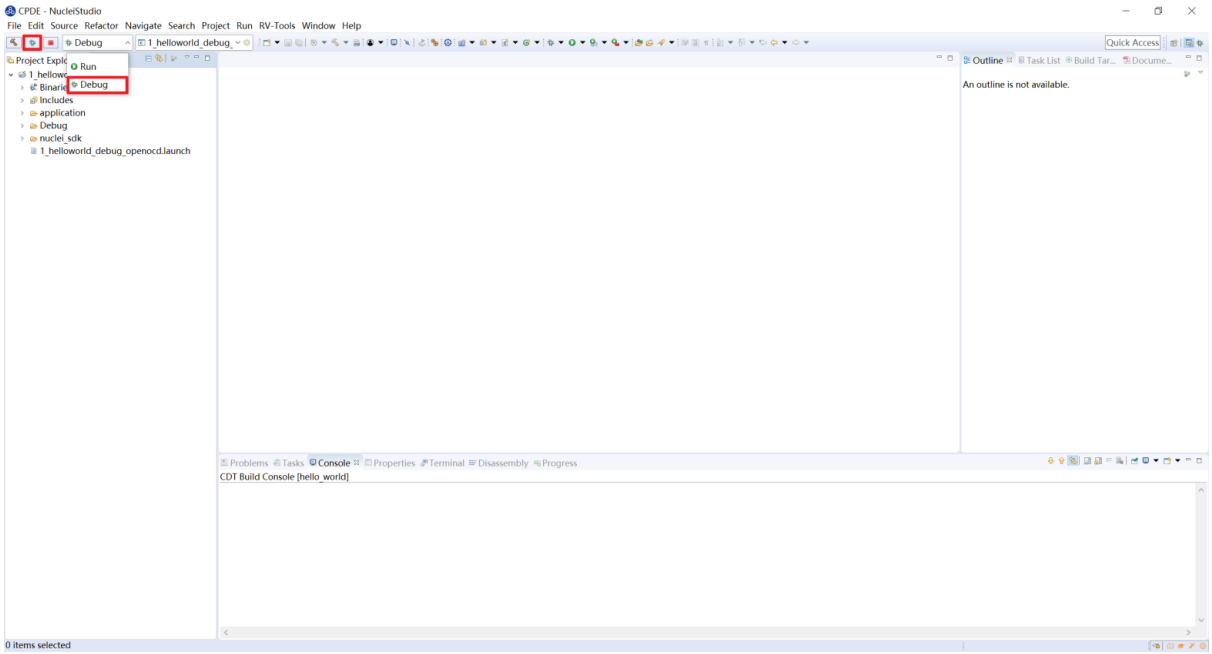
这里以 1_helloworld 为例，使用 Nuclei Studio IDE 对 evalsoc 原型开发板进行在线调试的步骤如下：

Note: 注意 demosoc 在 Nuclei SDK .5.0 中被移除，请使用 evalsoc 作为替代。

确保 Debug 设置内容正确，可以打开 Debug 设置选项确认。在 1_helloworld 工程处右击，选择 `Debug As >Debug Configuration` 打开 Debug 设置页面选择之前新建的设置进行检查。



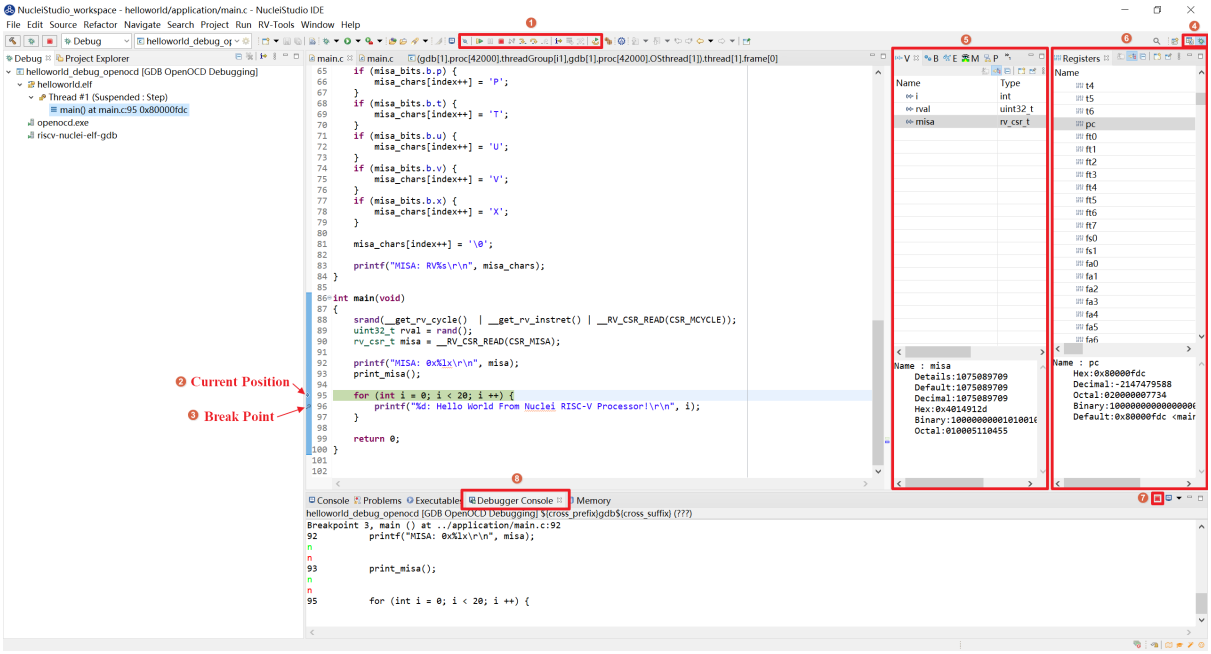
确定设置无误后，在下拉框选中 Debug，之后左侧图标会变为甲虫图标，单击即可进入调试模式并下载程序进入开发板中。



切换至 Debug 模式，如果下载成功，则会启动调试界面。

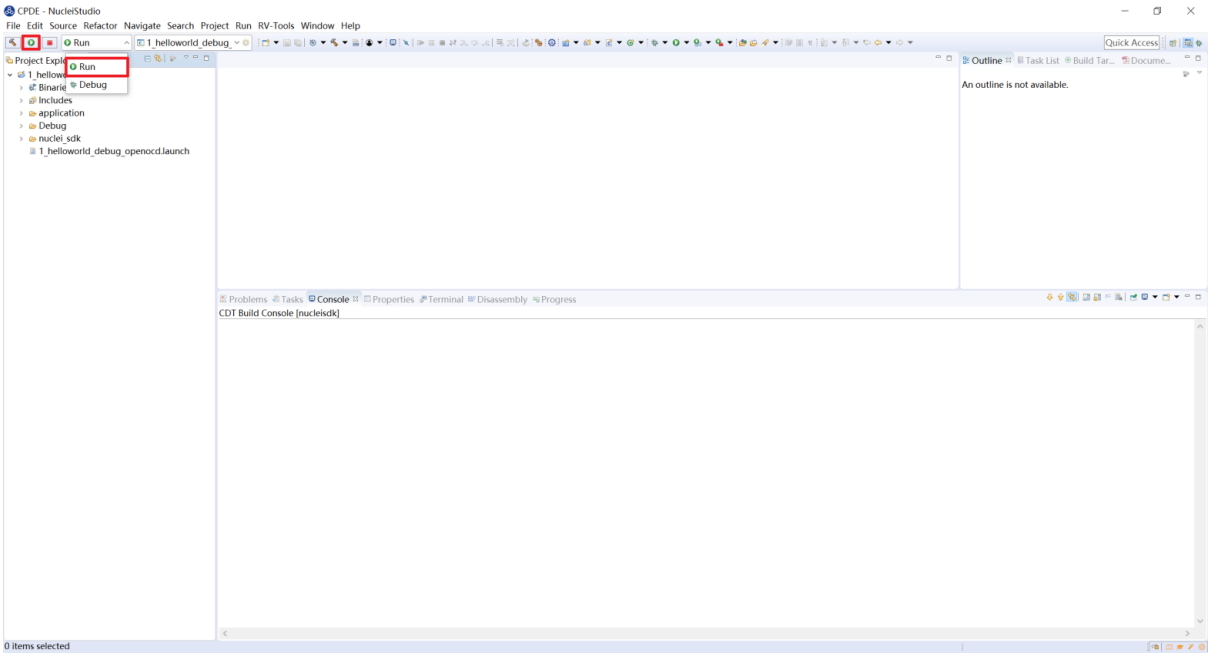
- 如图 1 号标注位置，这里功能包括单步，运行，汇编级调试等。
- 如图 2 号标注位置，这个箭头表示当前程序运行位置。

- 如图 3 号标注位置，在代码的左侧双击即可在该行设置断点，再次双击可以取消断点。
- 如图 4 号标注位置，这里可以切换编辑模式和调试模式。
- 如图 5 号标注位置，这里是函数内变量显示的位置。
- 如图 6 号标注位置，这里是查看寄存器数值的位置。图中显示的是 PC 寄存器当前的数值。
- 如图 7 号标注位置，点击这里红色按钮可以退出调试模式。
- 如图 8 号标注位置的下方，这里可以使用 GDB 控制台指令进行调试。

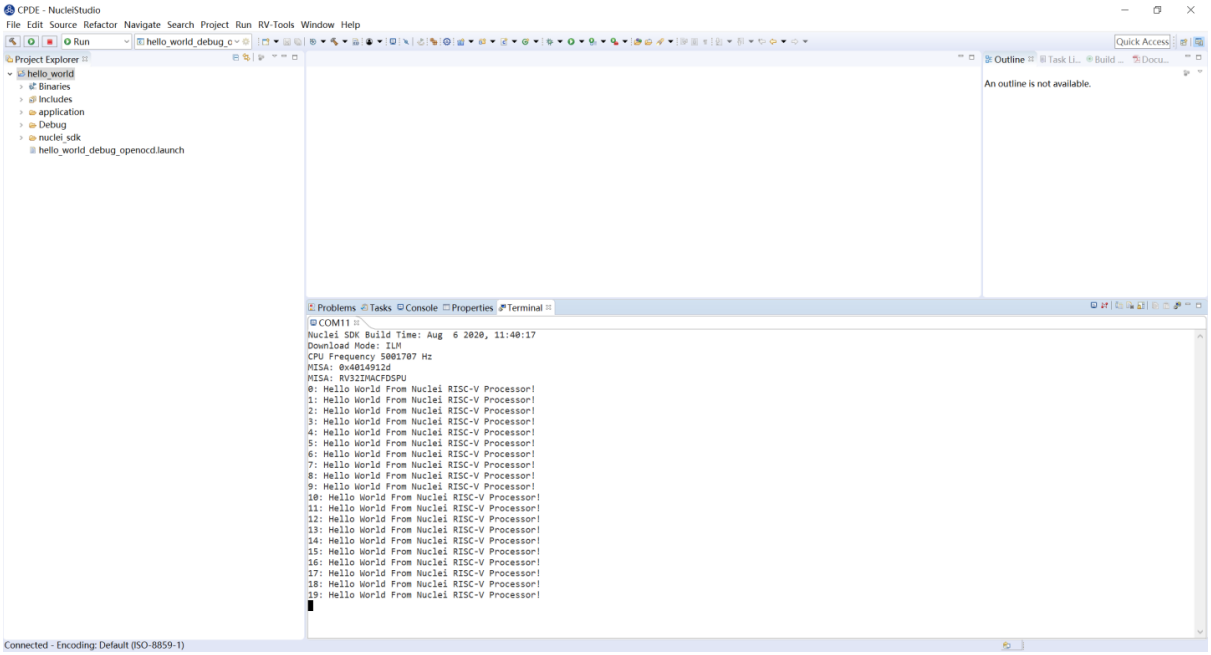


下载运行程序

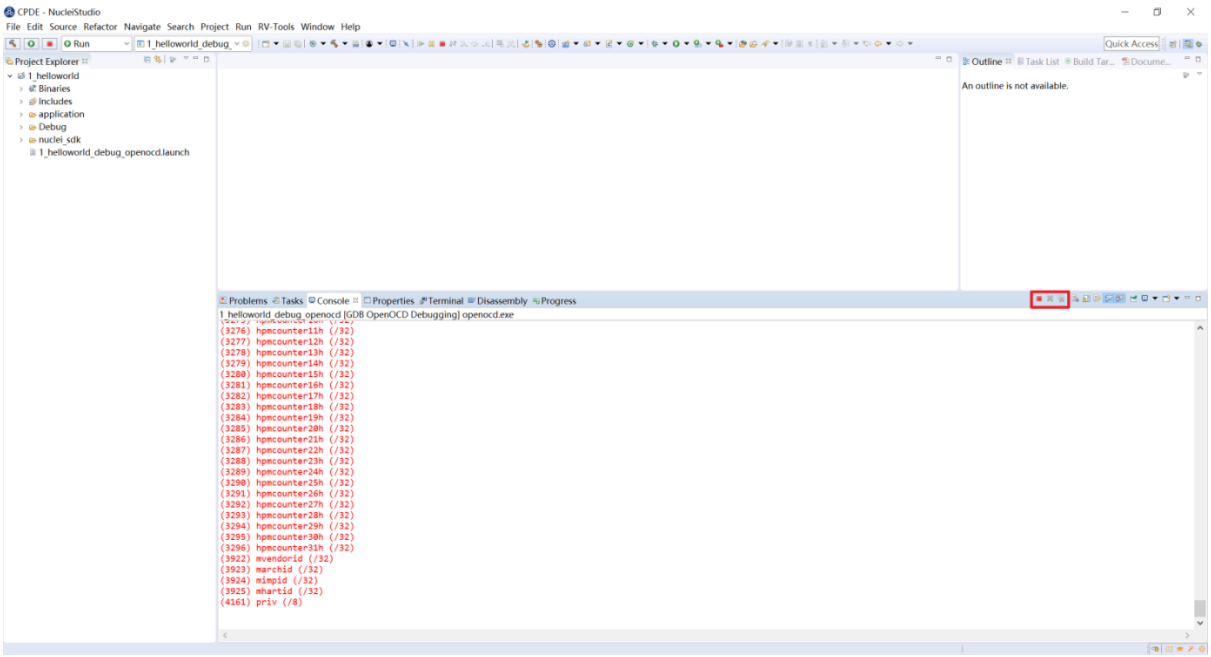
调试程序没有出现问题后，可以将程序下载进开发板。点击下拉框切换至运行模式，此时左侧图标会切换为绿色运行按键，单击即可将程序下载至开发板并运行。由于调试和下载运行使用相同的设置文件，所以不需要再次设置。



程序正常运行后，可以看到串口正确打印出 helloworld 等信息。



如果想要结束程序运行并需要断开连接，在 console 栏目下点击红色按钮断开连接。



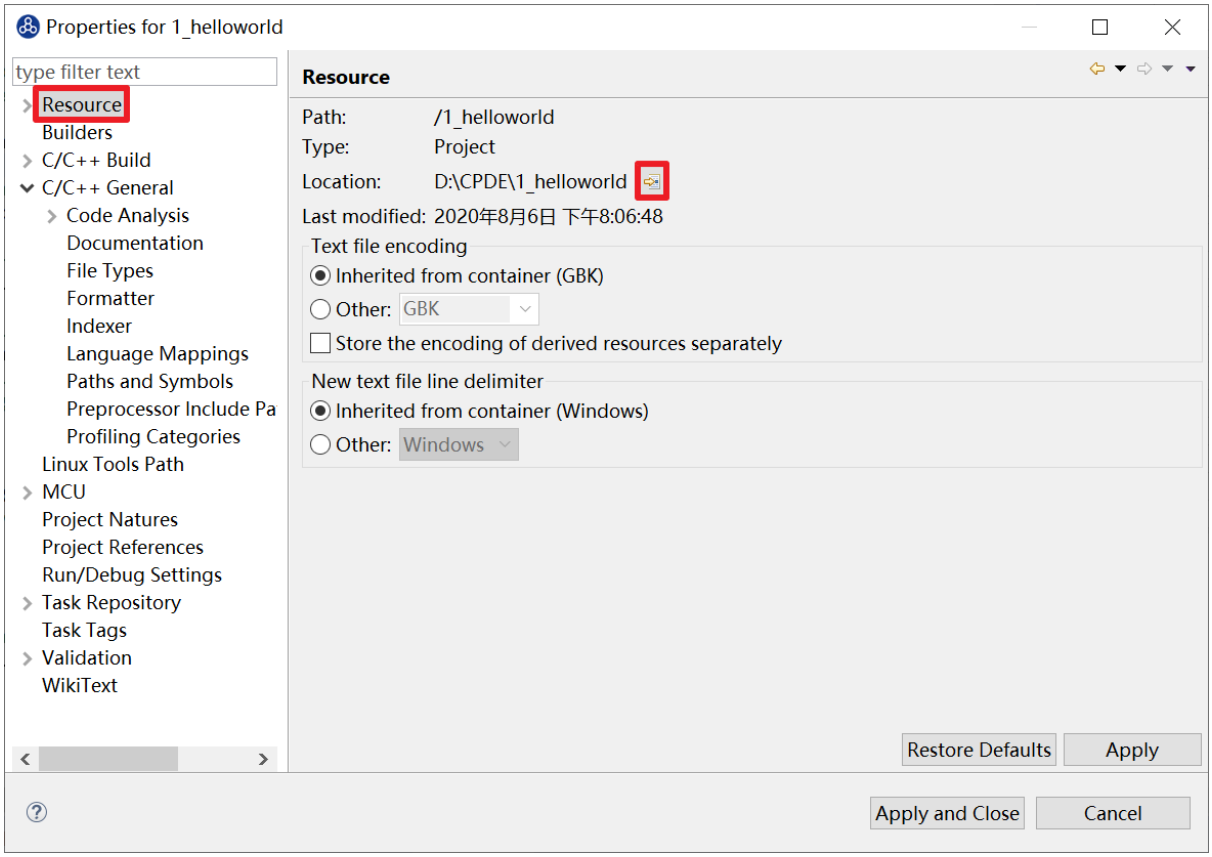
2.9.3 使用 J-Link 调试运行项目

安装 J-Link 驱动并导入 RTT 文件

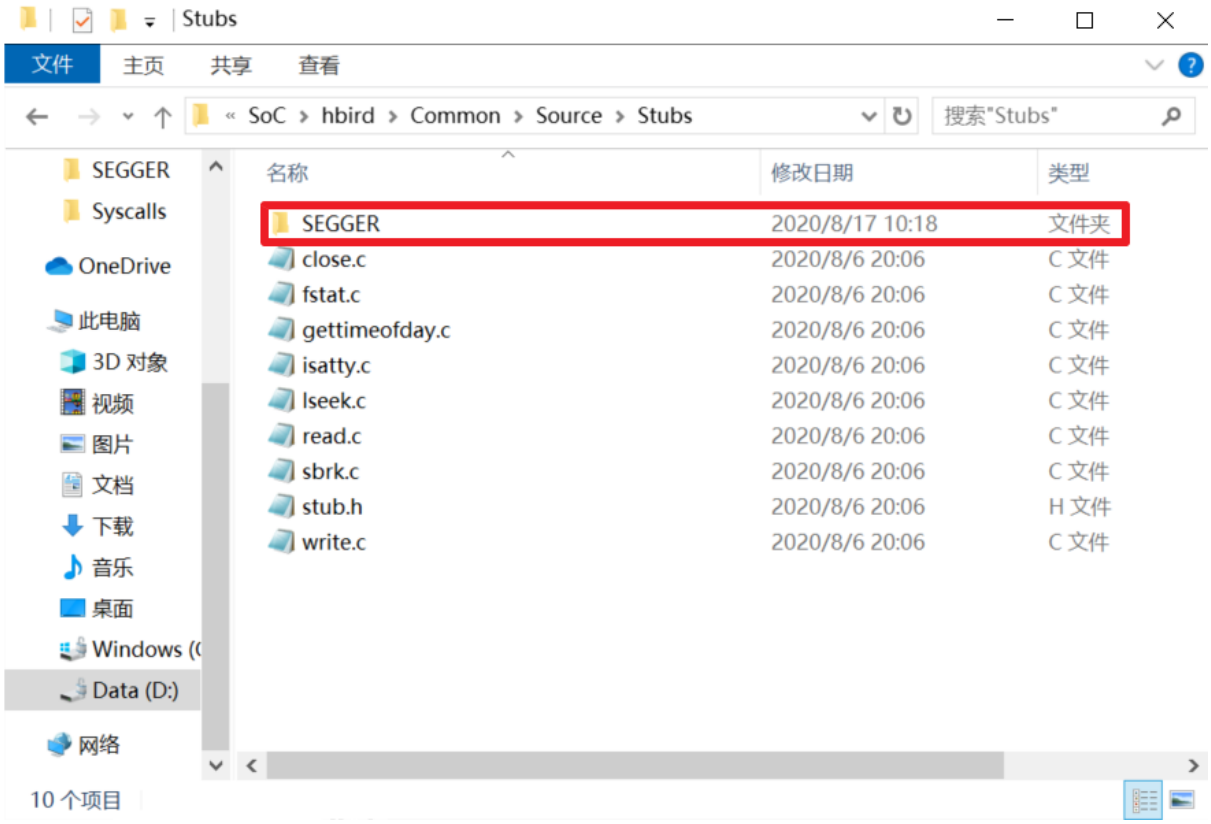
HummingBird Evaluation Board 也支持使用 J-Link 调试。前往 SEGGER 官网 J-Link 页面 (https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack)，根据自己的操作系统下载最新的 J-Link 驱动并安装。注意，J-Link 的版本必须高于 v6.62 版本。

如果使用串口进行打印输出，则可以略过本节后续内容。如果想使用 J-Link 的 RTT 打印输出，请按照以下步骤配置。

打开当前工程的设置页面，在 Resource 选项点击红框标注的图标快速打开工程所在的目录。

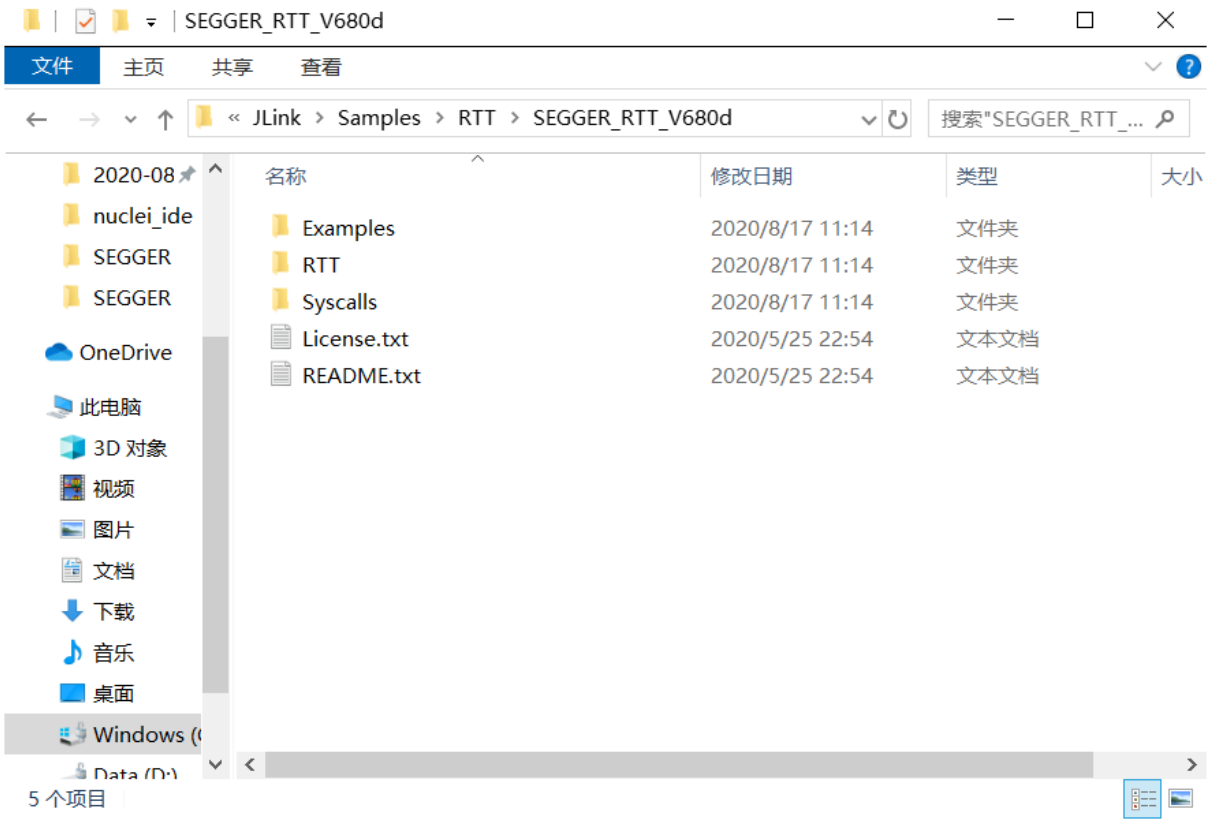


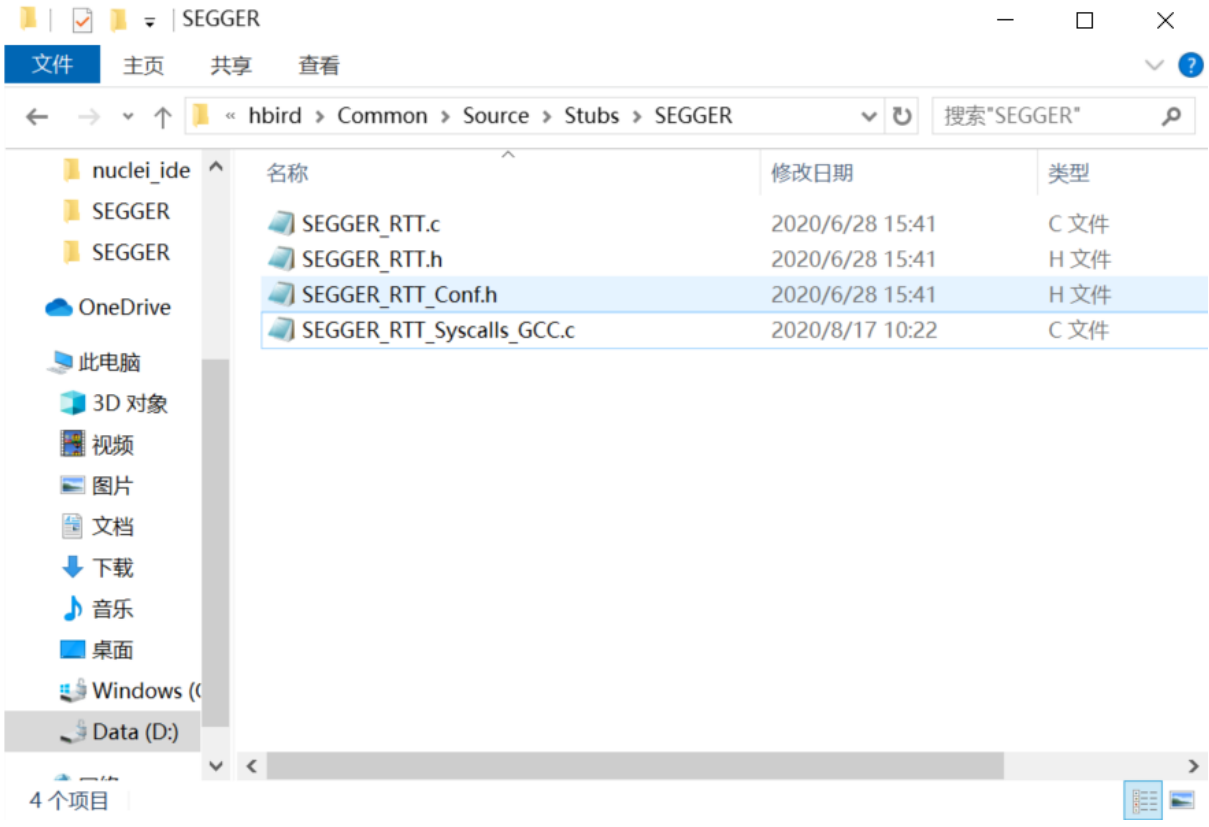
在 nuclei_sdk/SoC/evalsocsoc/Common/Source/Stubs 路径下新建一个 SEGGER 文件夹，此文件夹用来存放 RTT 相关文件。



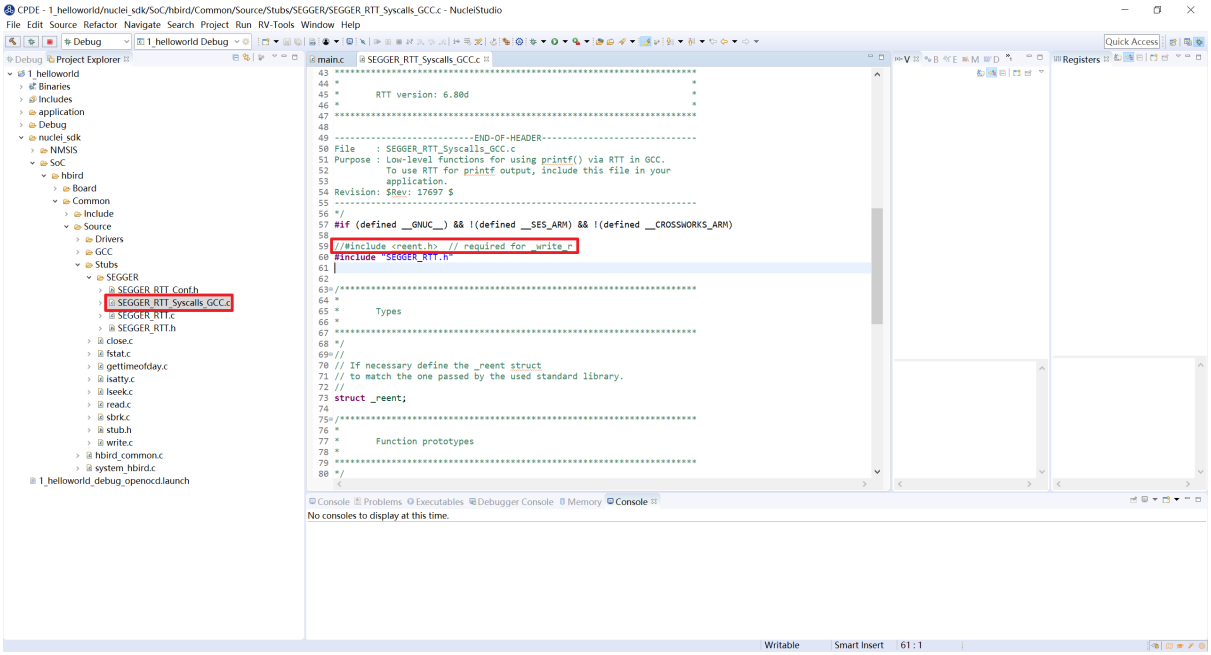
安装完成后打开 J-Link 驱动的根目录，将 Samples -> RTT 路径下的 SEGGER_RTT_V680d.zip 解压缩（具体压缩包名可能因版本不同而变化）。解压缩后文件内容，将 RTT 文件夹下的

SEGGER_RTT.c，SEGGER_RTT.h 和 SEGGER_RTT_Conf.h 三个文件以及 Syscalls 文件夹下的 SEGGER_RTT_Syscalls_GCC.c 这些文件复制到之前新建的 SEGGER 文件夹中。

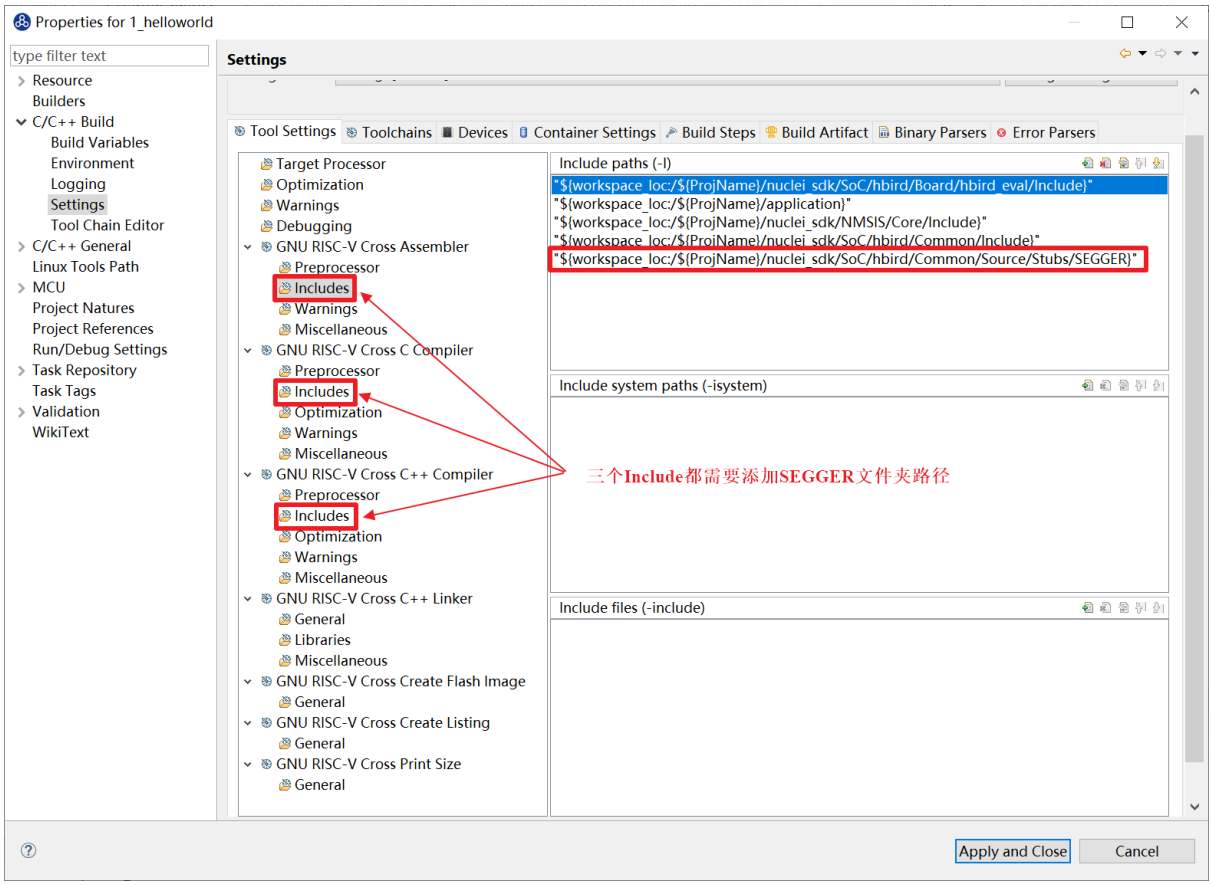




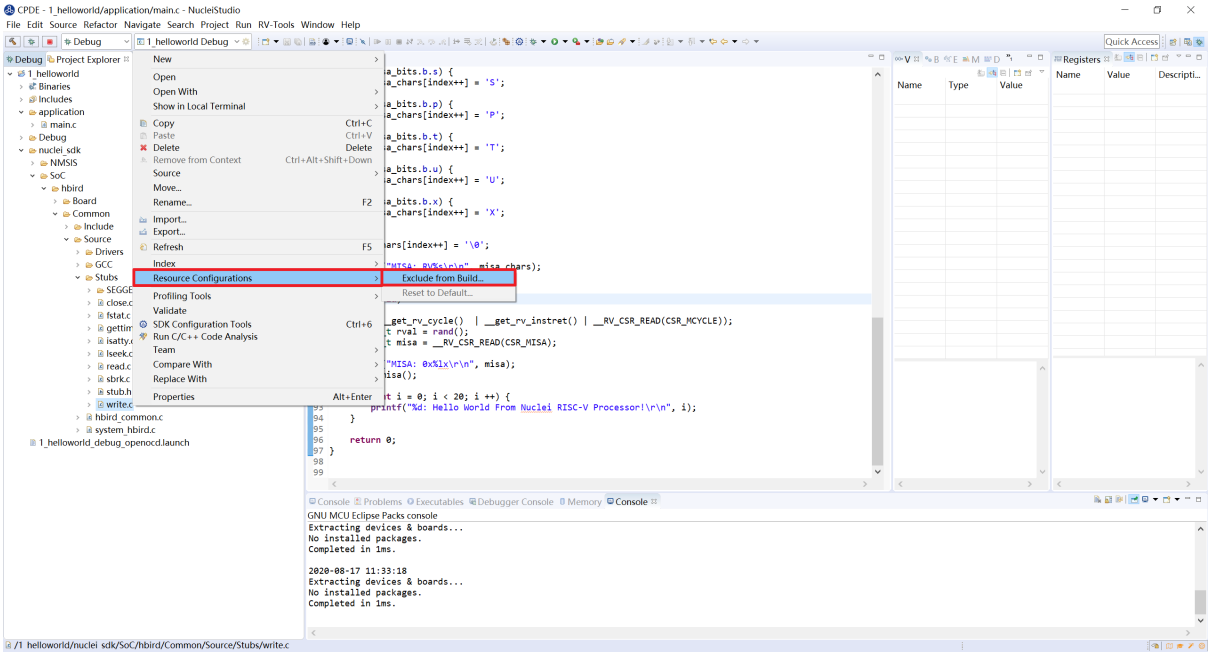
最后在 IDE 中打开 SEGGER_RTT_Syscalls_GCC.c，注释 #include <reent.h> 所在的这一行。



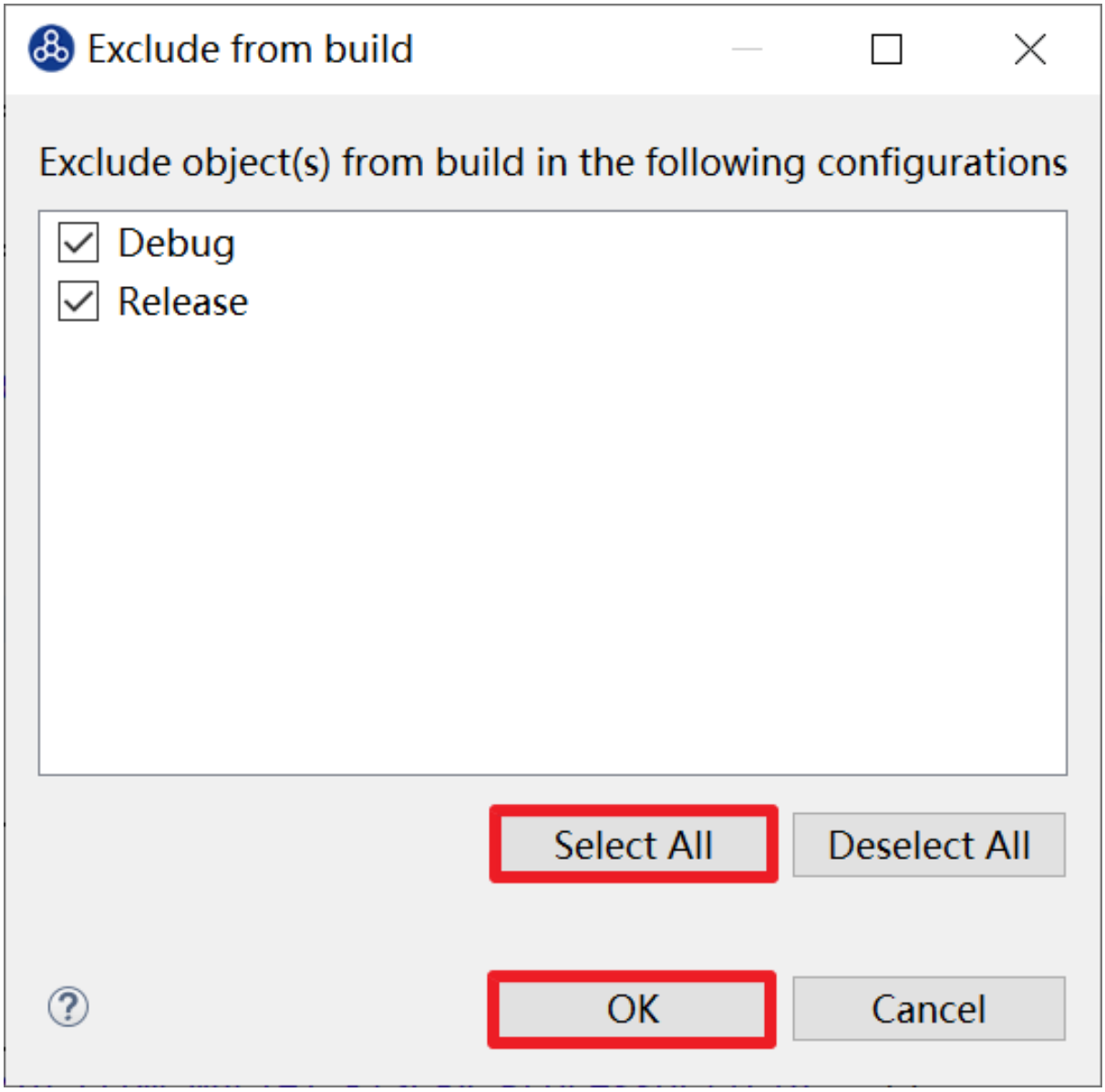
文件添加完成后添加 SEGGER 文件夹路径至 include，打开当前工程的设置页面，添加 SEGGER 文件夹路径至 include 中。



接下来移除原有的 write 函数。在 nuclei_sdk/SoC/evalsoc/Common/Source/Stubs 下的 write.c 文件处右击，选择 Resource Configurations > Exclude from Build。如图 7-30，选择 Select All，点击 OK。



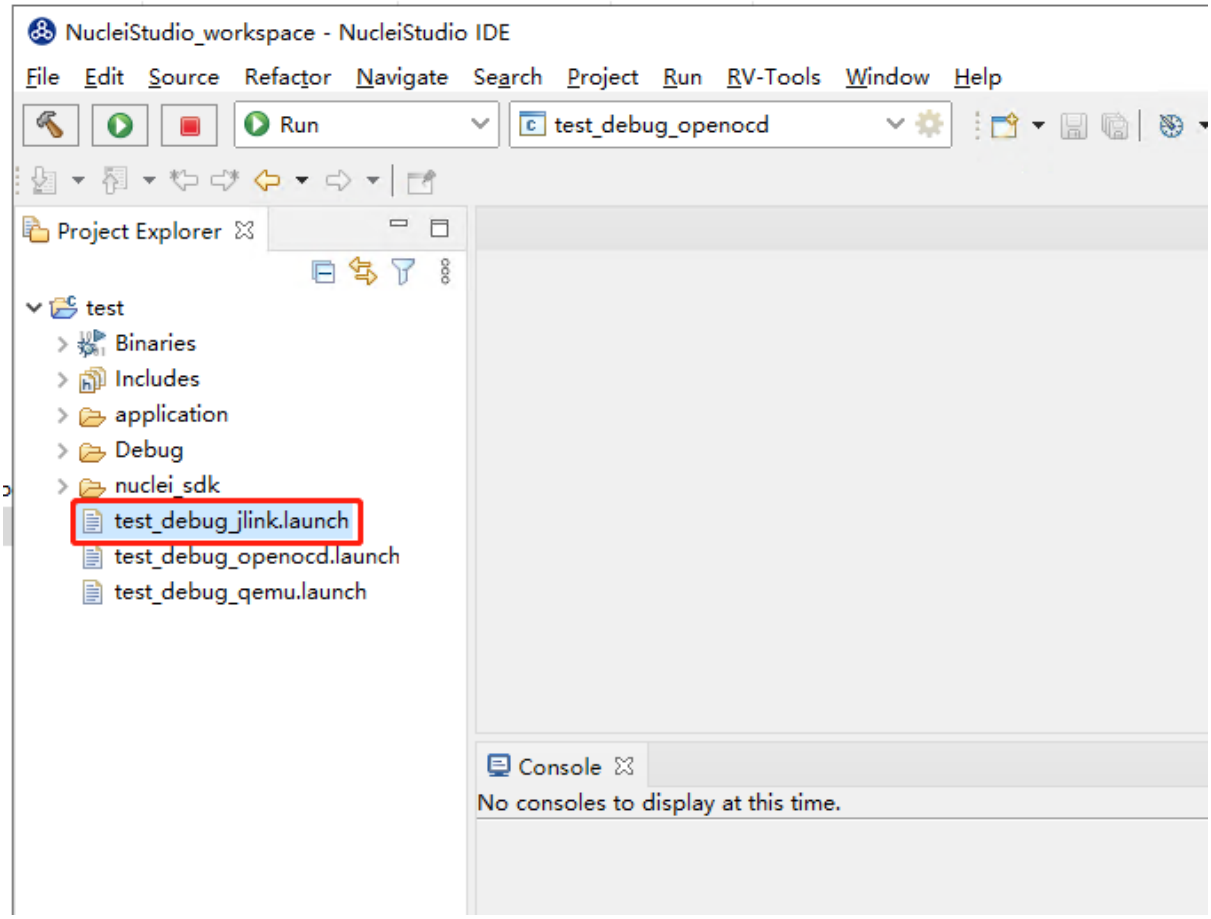
以后如果想切换回使用串口打印，可以使用相同的方式移除 SEGGER 文件夹并把 `write.c` 文件添加回工程。



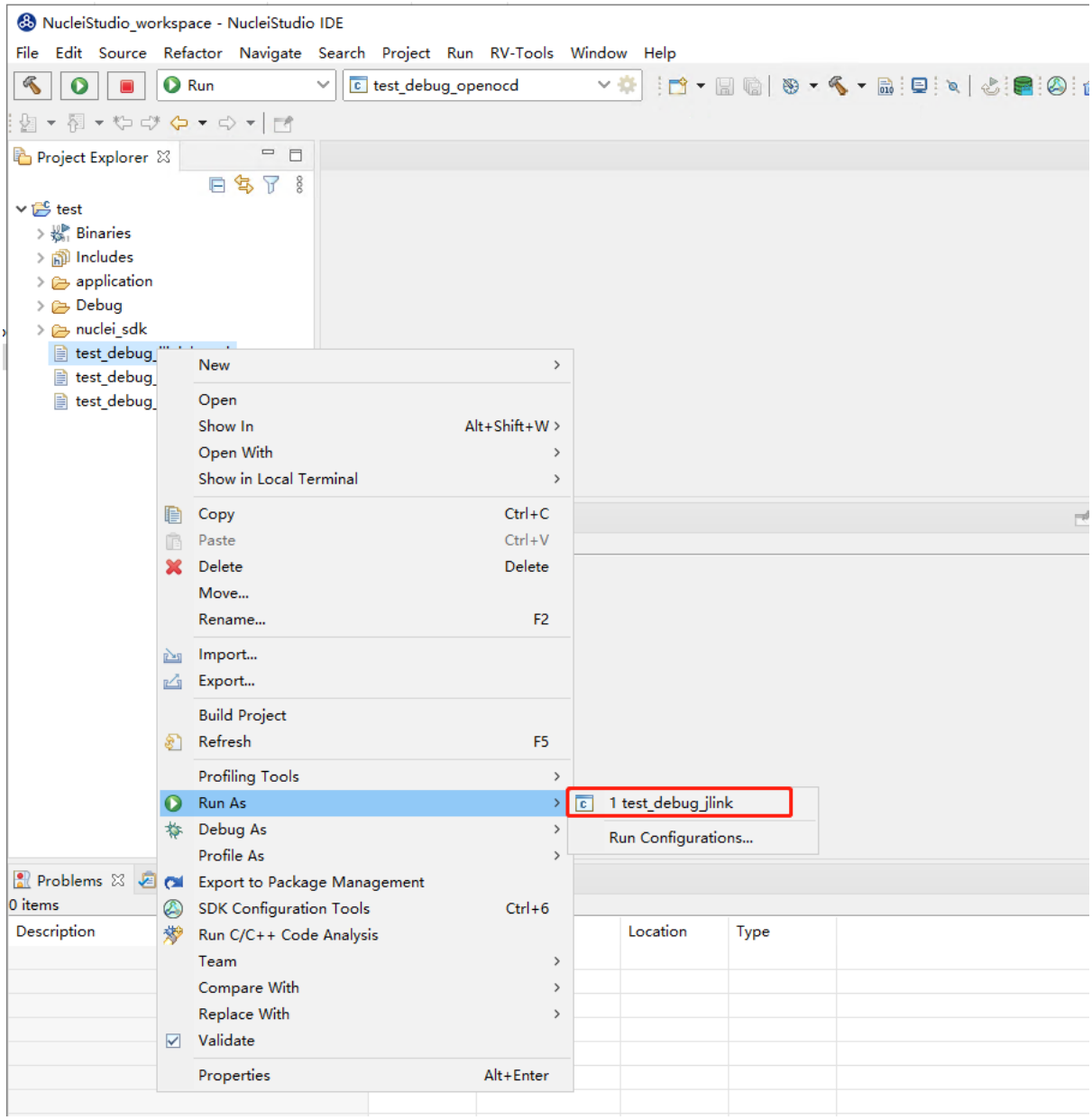
Debug Configuration

使用 Nuclei Studio 生成的 Debug Configuration

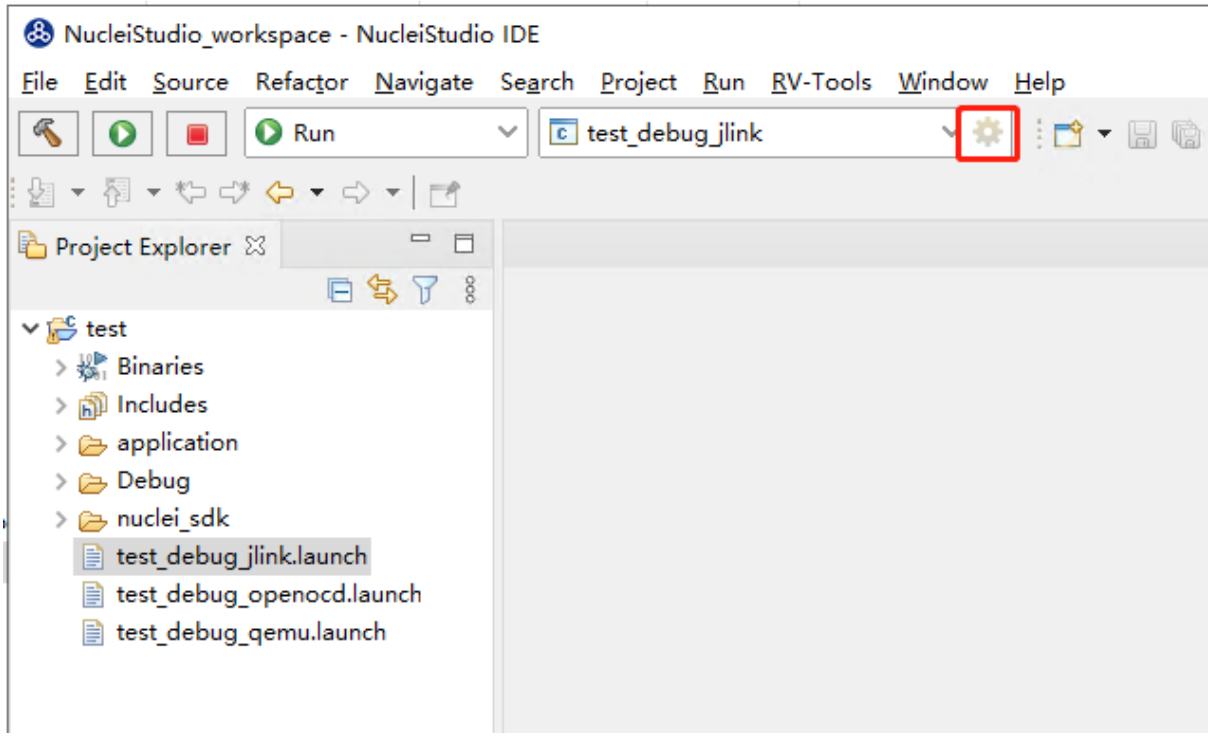
为了方便用户调试，Nuclei Studio 在创建工程时，会根据 NPK 的配置，默认的生成 Debug Configurations 的 Launch 文件。

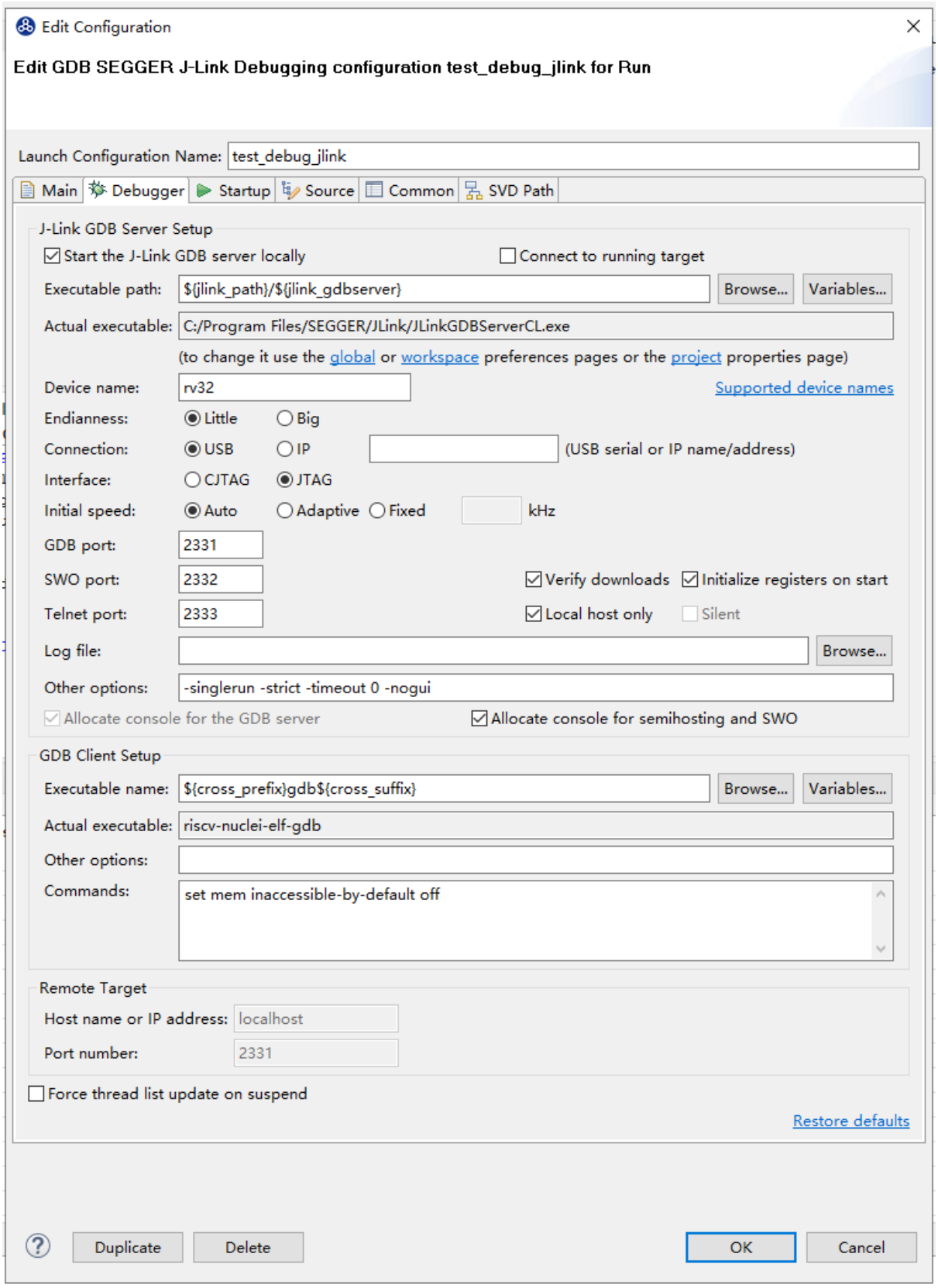


用户可以展开工程，选中对应的 `test_debug_jlink.launch` 文件，在右键菜单中，可以 `Run as/Debug as->test_debug_jlink`，就可以按照对应的 Debug Configurations 操作工程程了，



具体的 Debug Configurations 的内容可以在 Launch Bar 中进行详情查看。

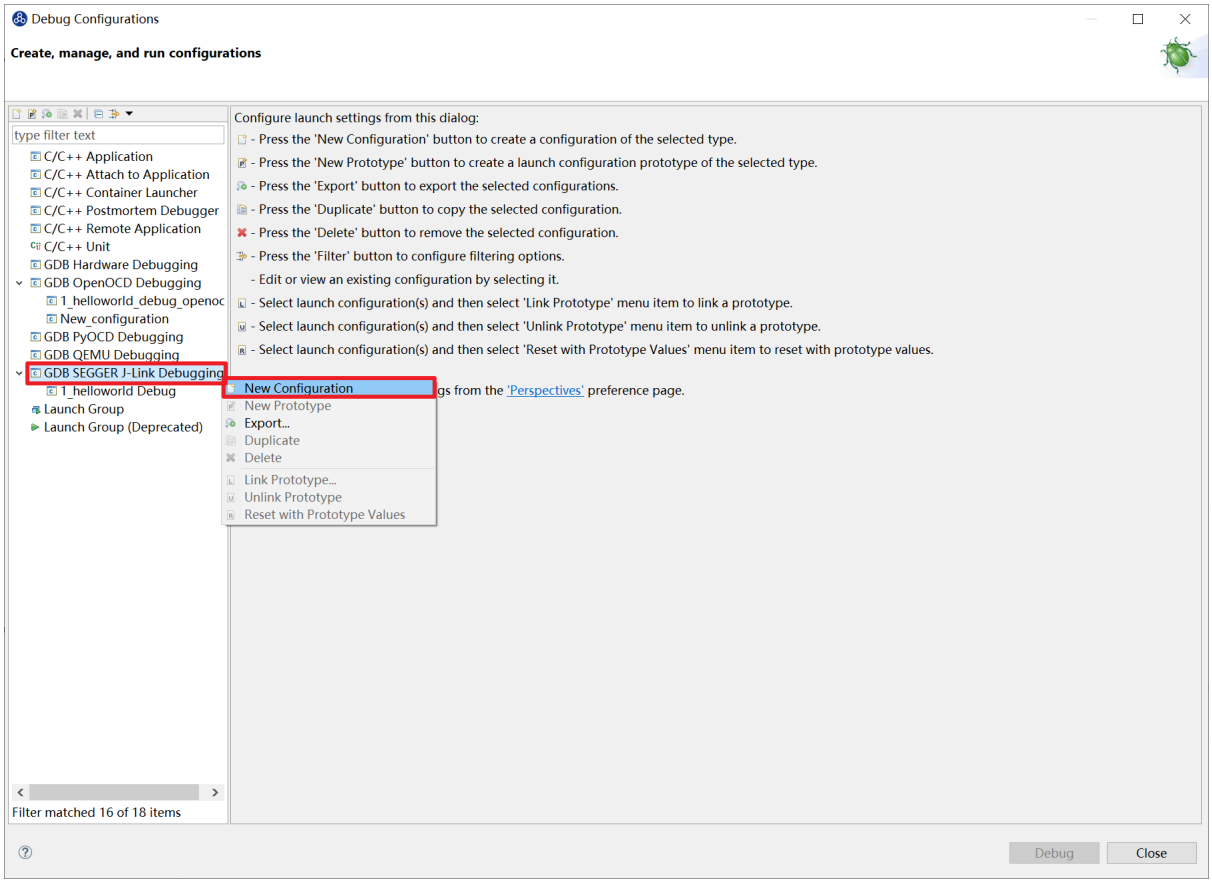

NucleiStudio_workspace - NucleiStudio IDE
File Edit Source Refactor Navigate Search Project Run RV-Tools Window Help
Run
test_debug_jlink
Project Explorer
test
Binaries
Includes
application
Debug
nuclei_sdk
test_debug_jlink.launch
test_debug_openocd.launch
test_debug_qemu.launch


Edit Configuration
Edit GDB SEGGER J-Link Debugging configuration test_debug_jlink for Run
Launch Configuration Name: test_debug_jlink
Main
Debugger
Startup
Source
Common
SVD Path
J-Link GDB Server Setup
Start the J-Link GDB server locally
Connect to running target
Executable path: ${jlink_path}/${jlink_gdbserver}
Browse...
Variables...
Actual executable: C:/Program Files/SEGGER/JLink/JLinkGDBServerCL.exe
(to change it use the global or workspace preferences pages or the project properties page)
Device name: rv32
Supported device names
Endianness: Little Big
Connection: USB IP
(USB serial or IP name/address)
Interface: CJTAG JTAG
Initial speed: Auto Adaptive Fixed
kHz
GDB port: 2331
SWO port: 2332
Verify downloads
Initialize registers on start
Telnet port: 2333
Local host only
Silent
Log file:
Browse...
Other options: -singlerun -strict -timeout 0 -nogui
Allocate console for the GDB server
Allocate console for semihosting and SWO
GDB Client Setup
Executable name: ${cross_prefix}gdb${cross_suffix}
Browse...
Variables...
Actual executable: riscv-nuclei-elf-gdb
Other options:
Commands: set mem inaccessible-by-default off
Remote Target
Host name or IP address: localhost
Port number: 2331
Force thread list update on suspend
Restore defaults
Duplicate
Delete
OK
Cancel

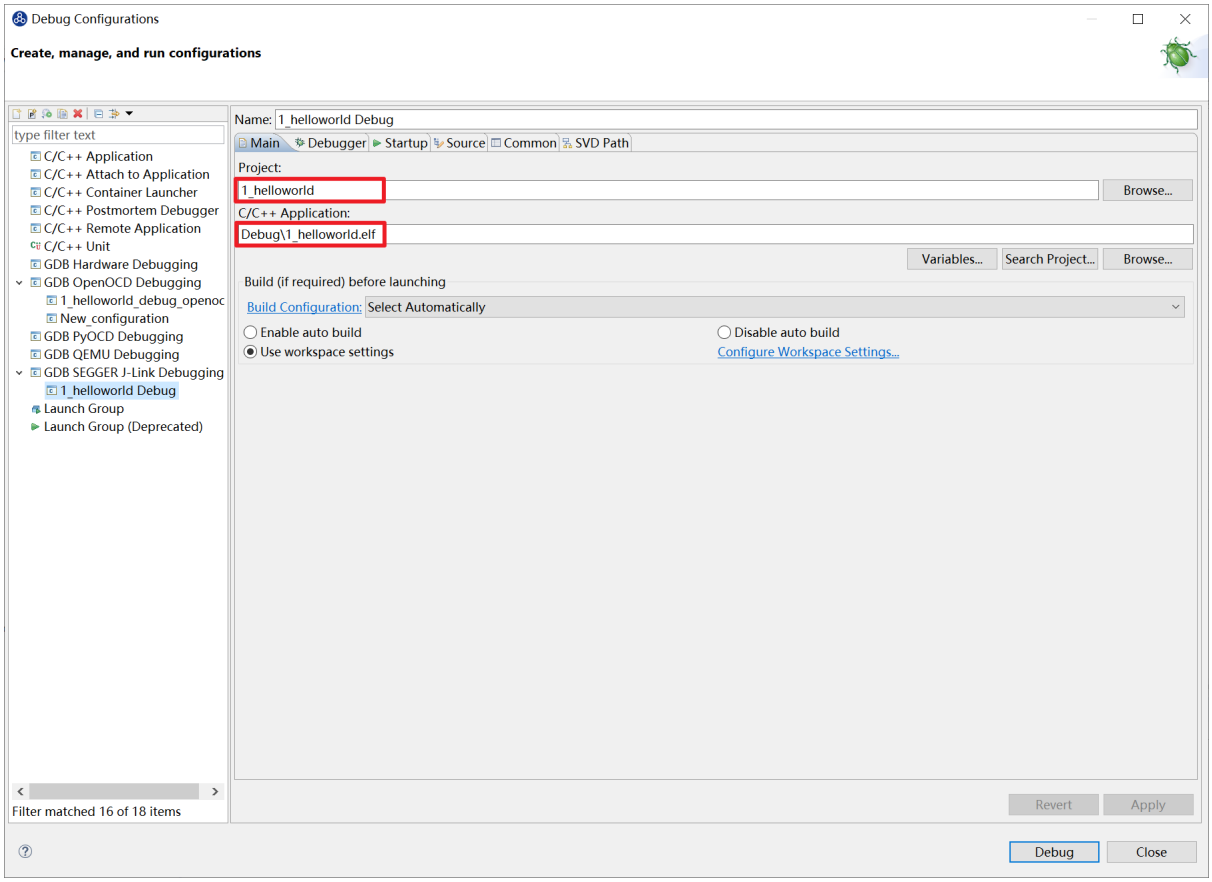
新建并配置 Debug Configuration

新建并配置 J-Link 调试下载的 Debug Configuration 步骤如下：

在菜单栏中选择 Run—>Debug Configurations。在弹出的窗口中，如果没有当前工程的调试设置内容，右键单击 GDB SEGGER J-Link Debugging，选择 New Configuration，将会为本项目新建出一个调试项目 1_helloworld Debug。



确保 Project 是当前需要调试的工程，C/C++ Application 中选择了正确的需要调试的 ELF 文件。



打开 Debugger 栏目，确保 1 号位置 `Start the J-Link GDB server locally` 被选中。

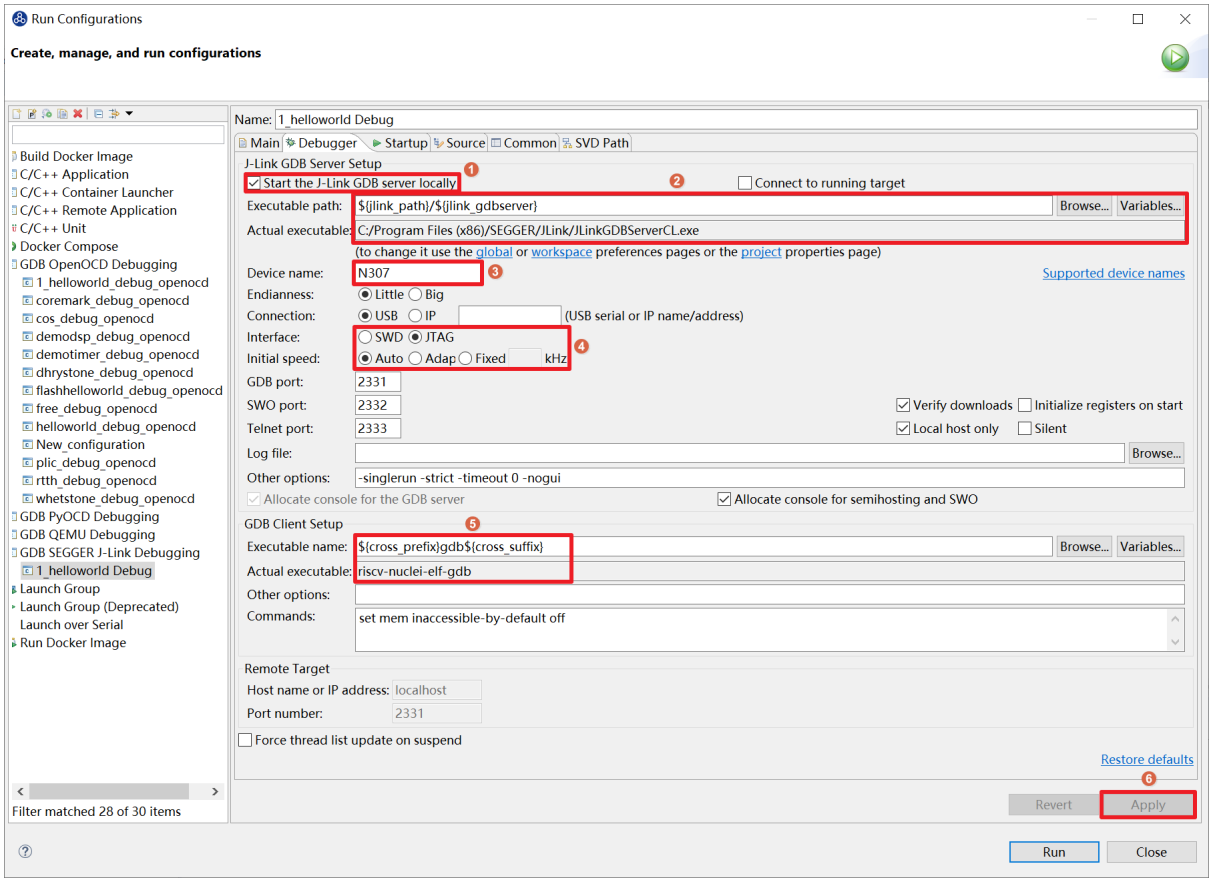
2 号位置正确指向 JLinkGDBServerCL.exe 的路径。

3 号内是当前使用的内核，这里以 N307 为例，输入 N307 即可。如果使用 RV-STAR 开发板，这里输入 GD32VF103VBT6。如果使用其他开发板请参考 J-Link Support Device 网页，链接如下：https://www.segger.com/downloads/supported-devices.php

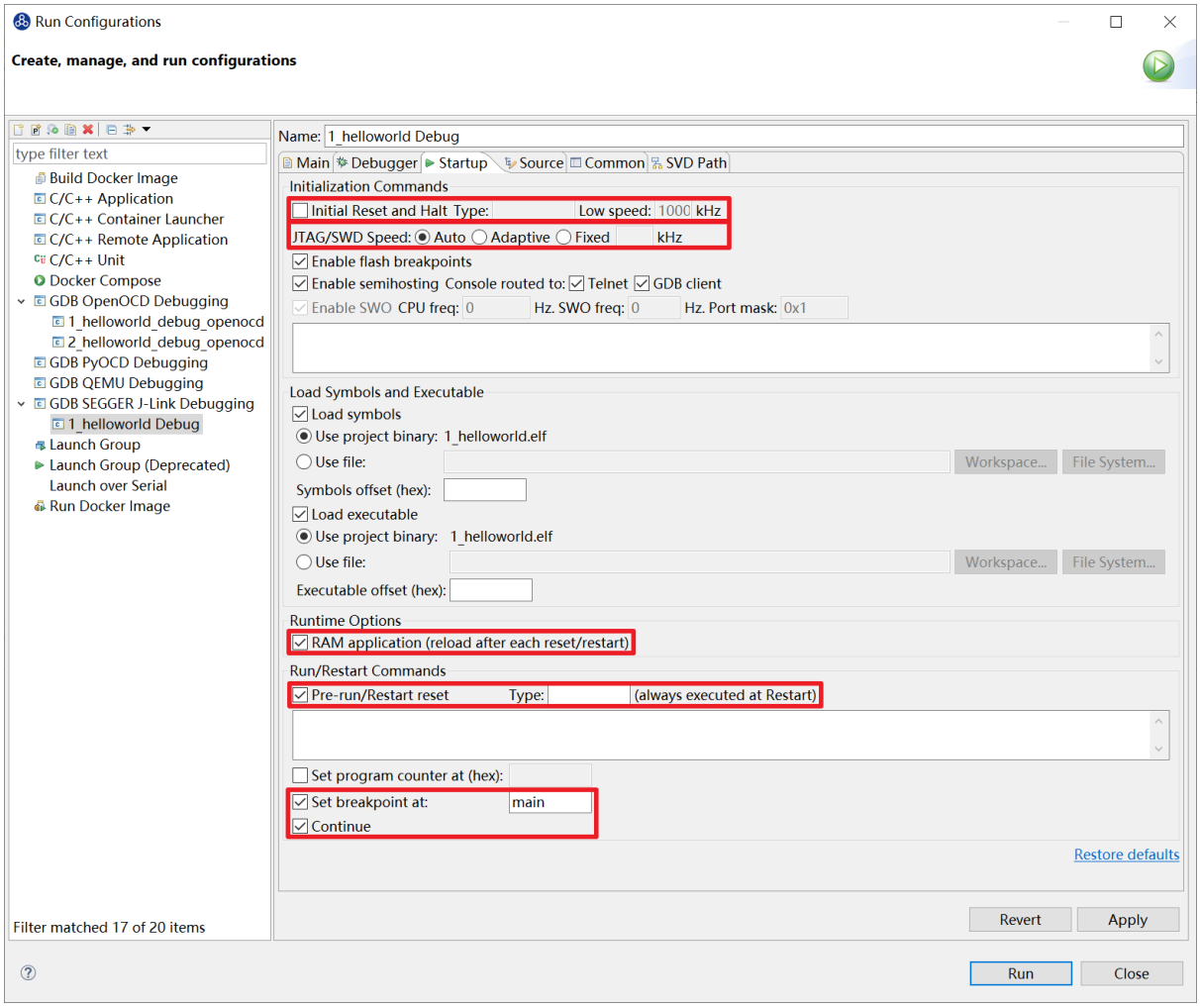
4 号选择 `Interface` 为 JTAG，`initial speed` 为 Auto。

5 号确认与使用的 GDB 设置一致。

如果有修改的内容，点击 6 号位置 `Apply` 保存。



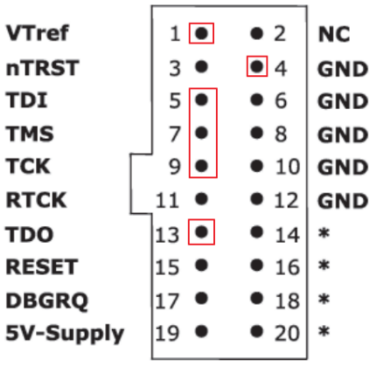
打开 Startup 栏目，确保 JTAG/SWD Speed 为 Auto，set Breakpoint at main，Continue，Pre-run/Restart reset 和 RAM application 选项被勾选，并且取消勾选 Initial Reset and Halt 选项。



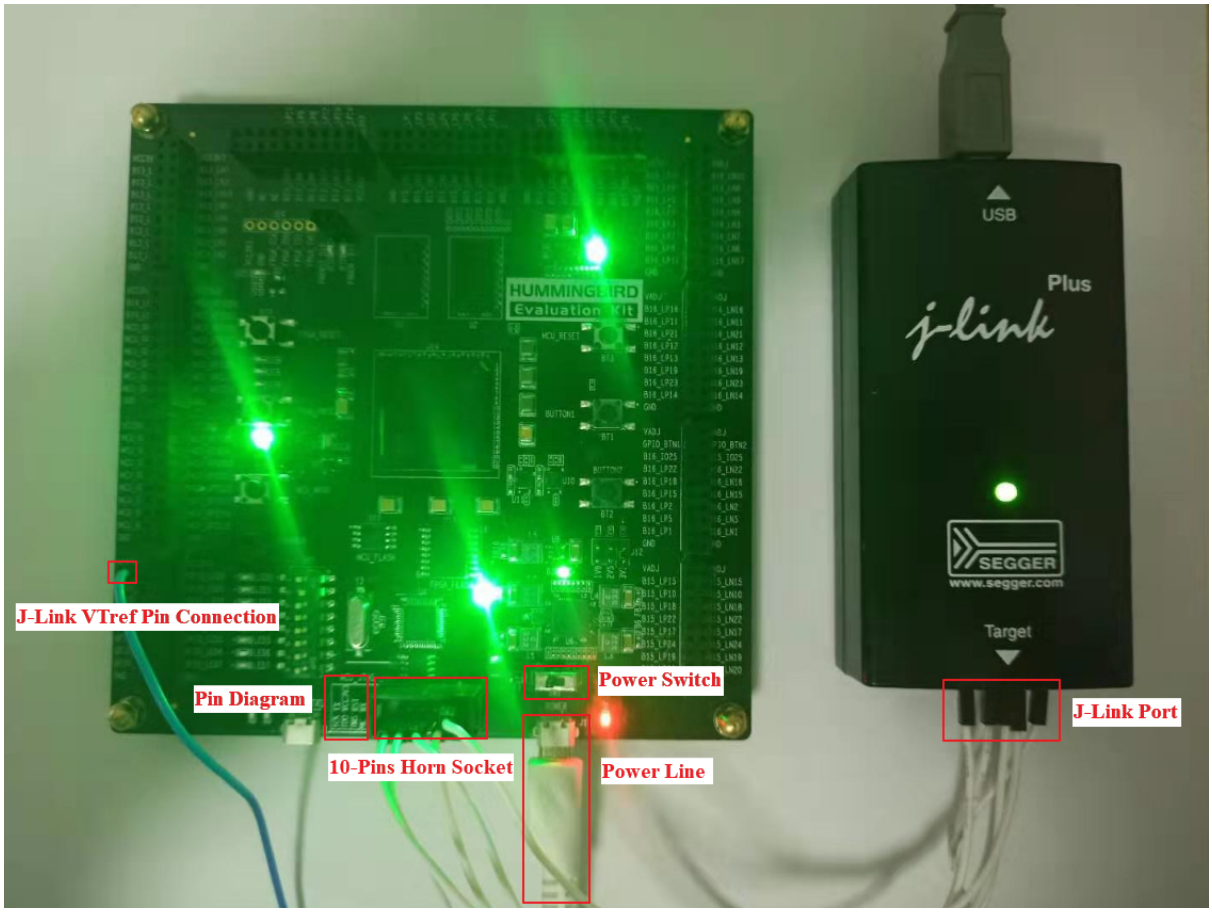
以上设置内容完成后，如果有变动需要点击右下角 Apply 保存设置，如果没有变动点击 close 即可。

在原型开发板上调试程序

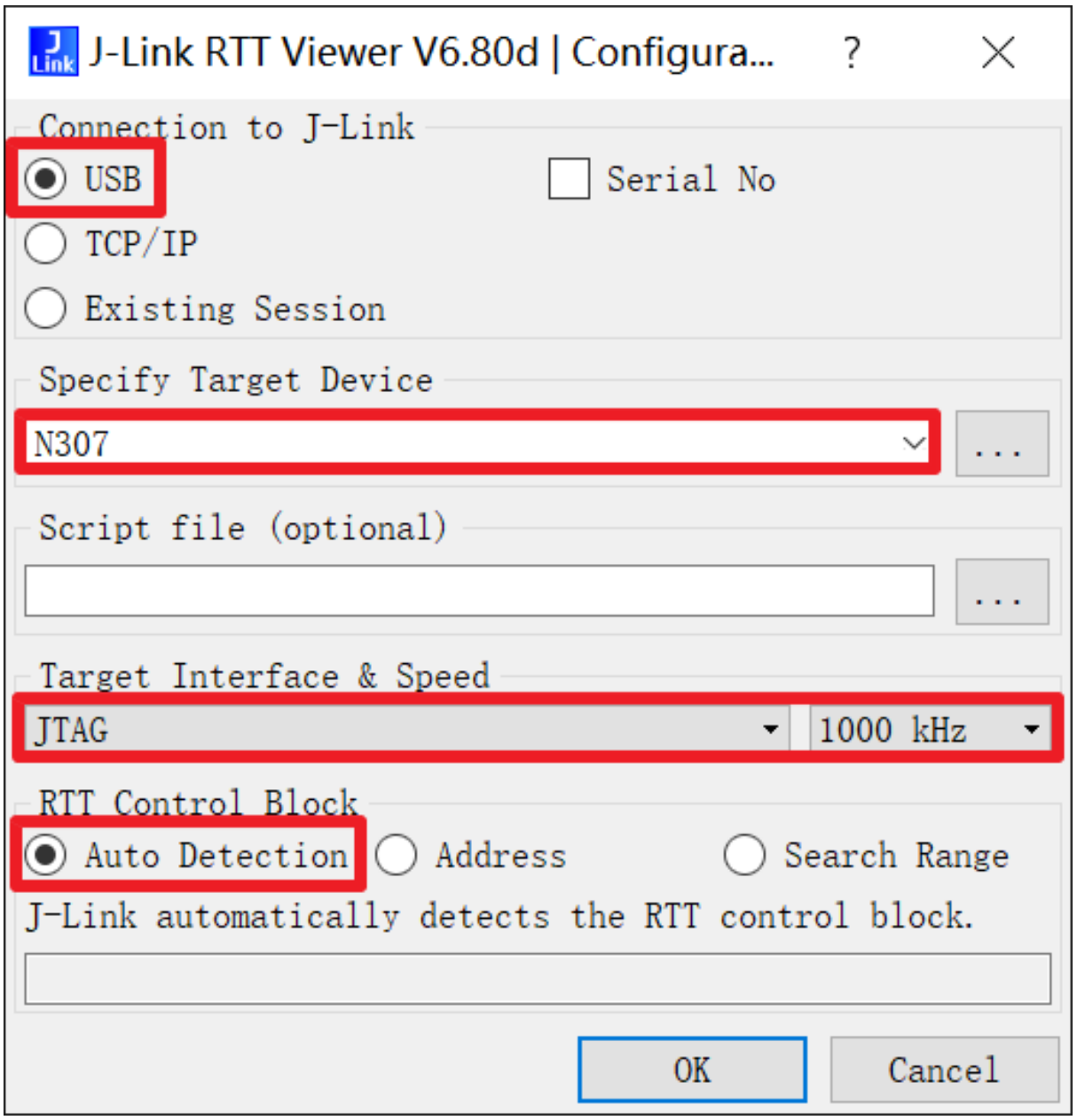
使用 J-Link 在 HummingBird Evaluation Board 调试需要连接跳线。红框标注的部分是 J-Link 需要连接到板子的部分。



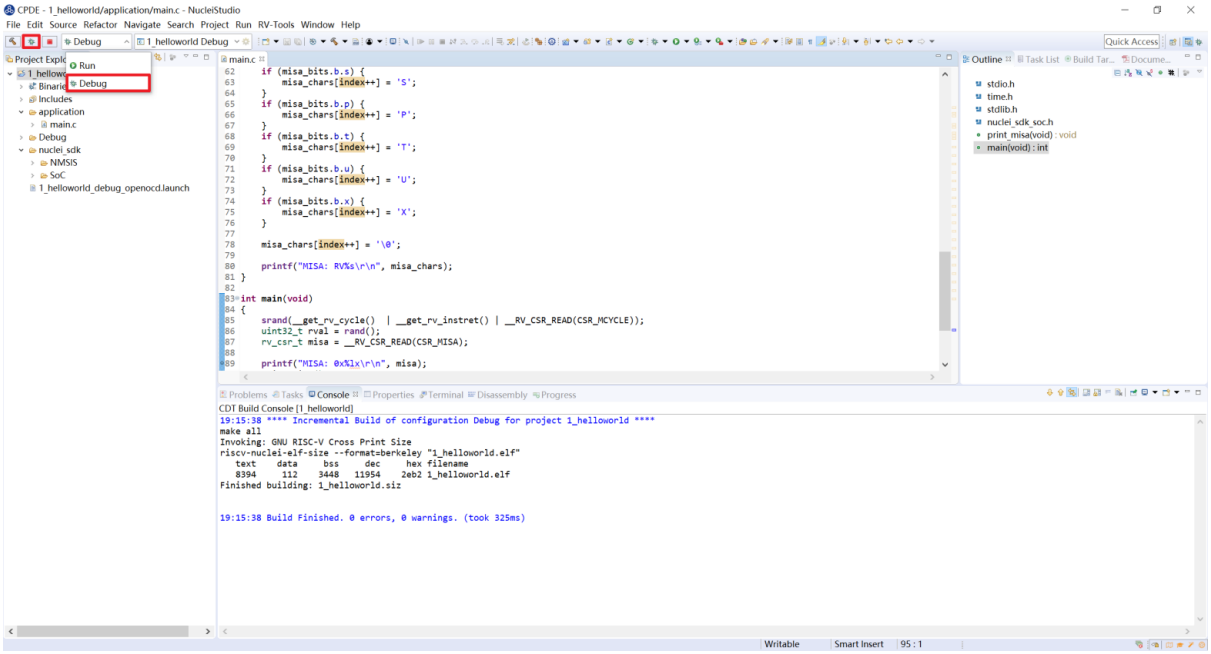
其中 VTref 连接到板子上 V3.3 的接口，其他部分连接到 JTAG 接口，各引脚的丝印就在旁边，一一对应连接即可，最后实物连接如下图。



在开发板上调试之前，如果使用串口打印，需要连接 JTAG 上的串口引脚到自己的主机上，再打开串口以便观察 Printf 函数打印信息。如果使用 RTT 打印，需要打开 `J-Link RTT Viewer` 查看 printf 打印信息。按照图中内容设置，选择 USB 方式连接。Specify Target Device 根据使用的内核来修改，这里以 N307 为例。Target Interface & Speed 设置为 `1000kHz`，可根据实际使用情况来修改。RTT Control Block 选择 `Auto Detection`。

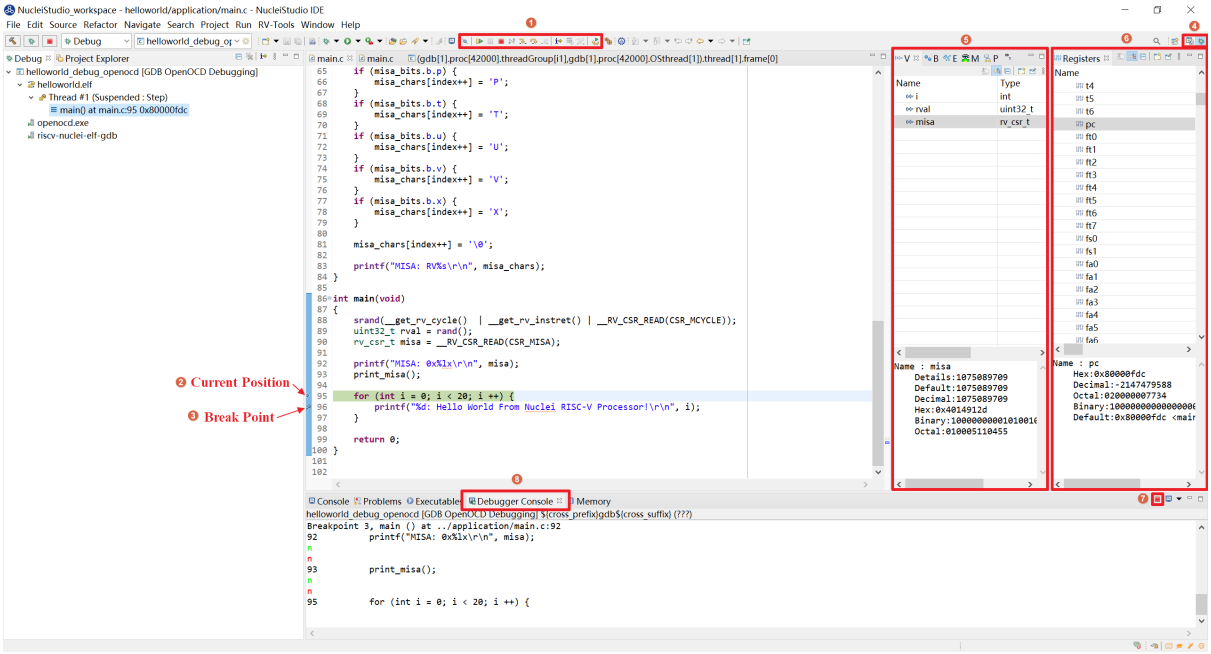


在 Launch Bar 的下拉框选中 Debug，之后左侧图标会变为甲虫图标，单击即可进入调试模式并下载程序进入开发板中。



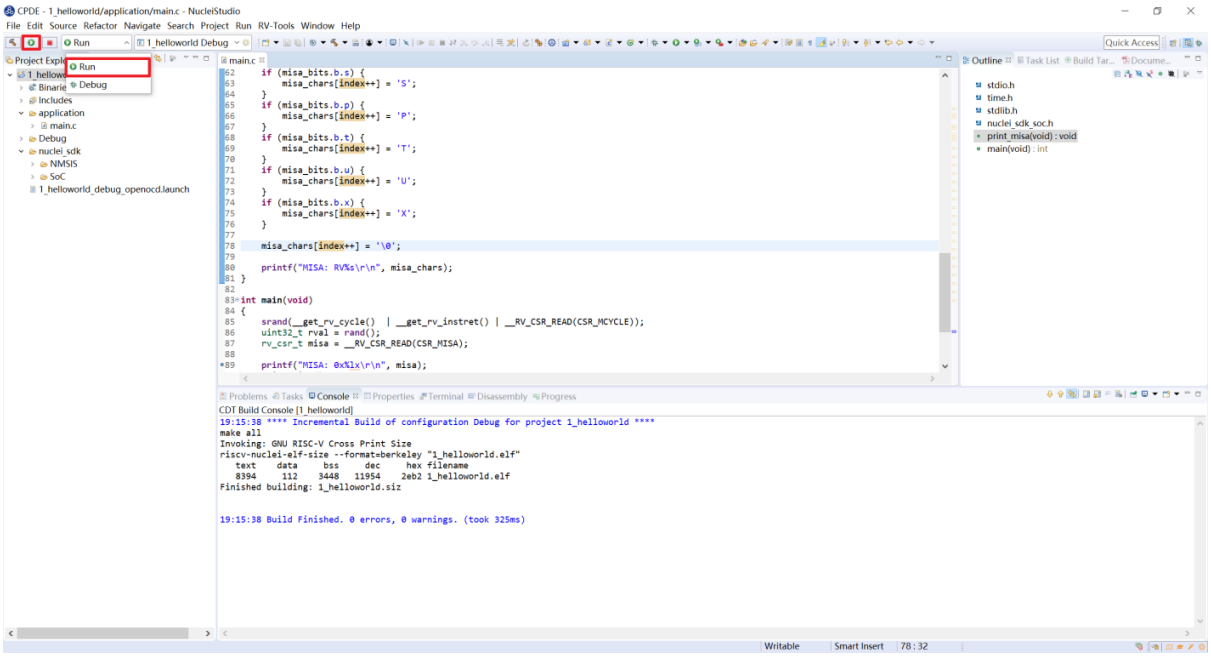
如果程序下载成功，并且 Nuclei Studio 会启动调试界面。

- 如图 1 号标注位置，这里功能包括单步，运行，汇编级调试等。
- 如图 2 号标注位置，这个箭头表示当前程序运行位置。
- 如图 3 号标注位置，在代码的左侧双击即可在该行设置断点，再次双击可以取消断点。
- 如图 4 号标注位置，这里可以切换编辑模式和调试模式。
- 如图 5 号标注位置，这里是函数内变量显示的位置。
- 如图 6 号标注位置，这里是查看寄存器数值的位置。
- 如图 7 号标注位置，点击这里红色按钮可以退出调试模式。
- 如图 8 号标注位置下方，这里可以使用 GDB 控制台指令进行调试。

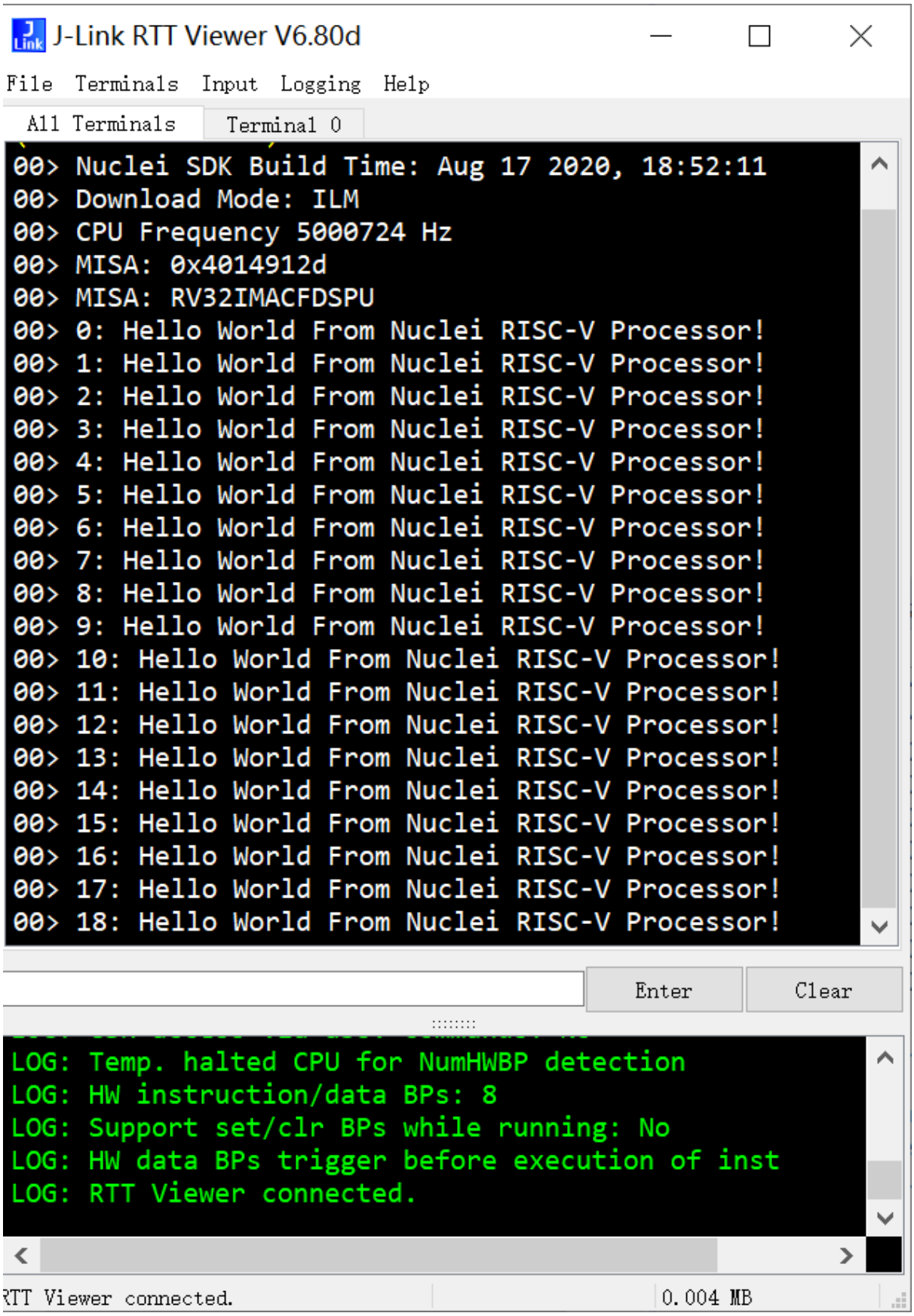


下载运行程序

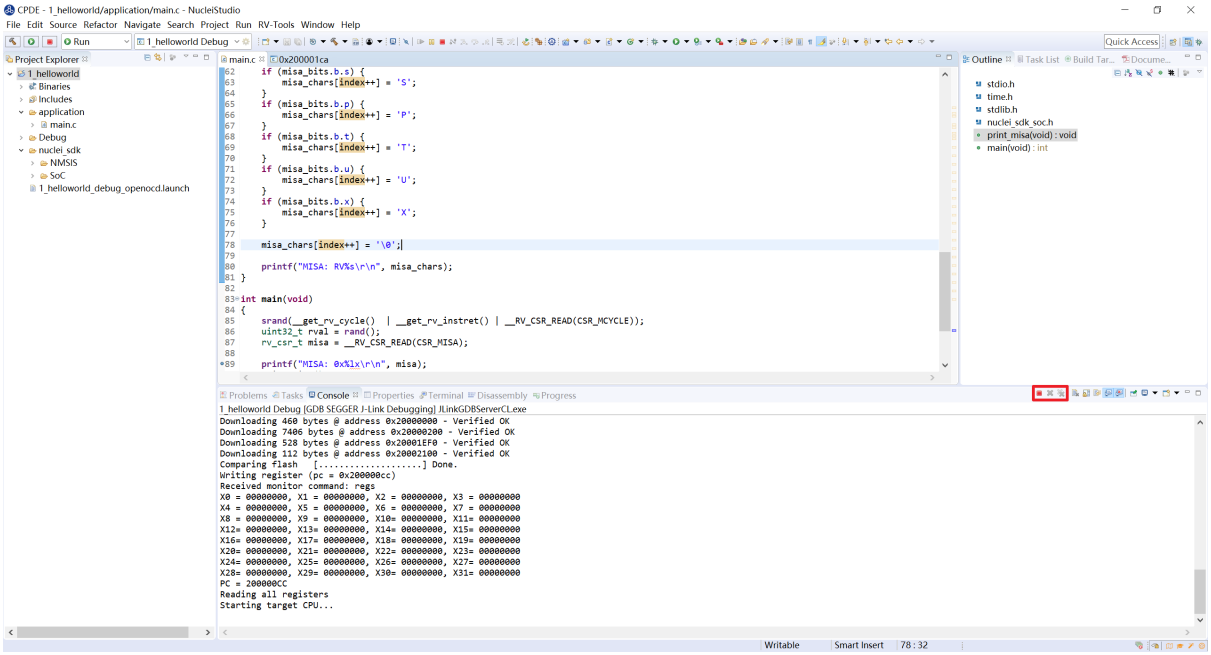
调试程序没有出现问题后，可以将程序下载进开发板，点击下拉框切换至运行模式，此时左侧图标会切换为绿色运行按键，单击即可将程序下载至开发板并运行。



由于调试和下载使用相同的设置文件，所以不需要再次设置。可以看到 RTT Viewer 正确打印出 helloworld 等信息。



如果需要断开连接，在 console 栏目下点击红色按钮即可断开连接。

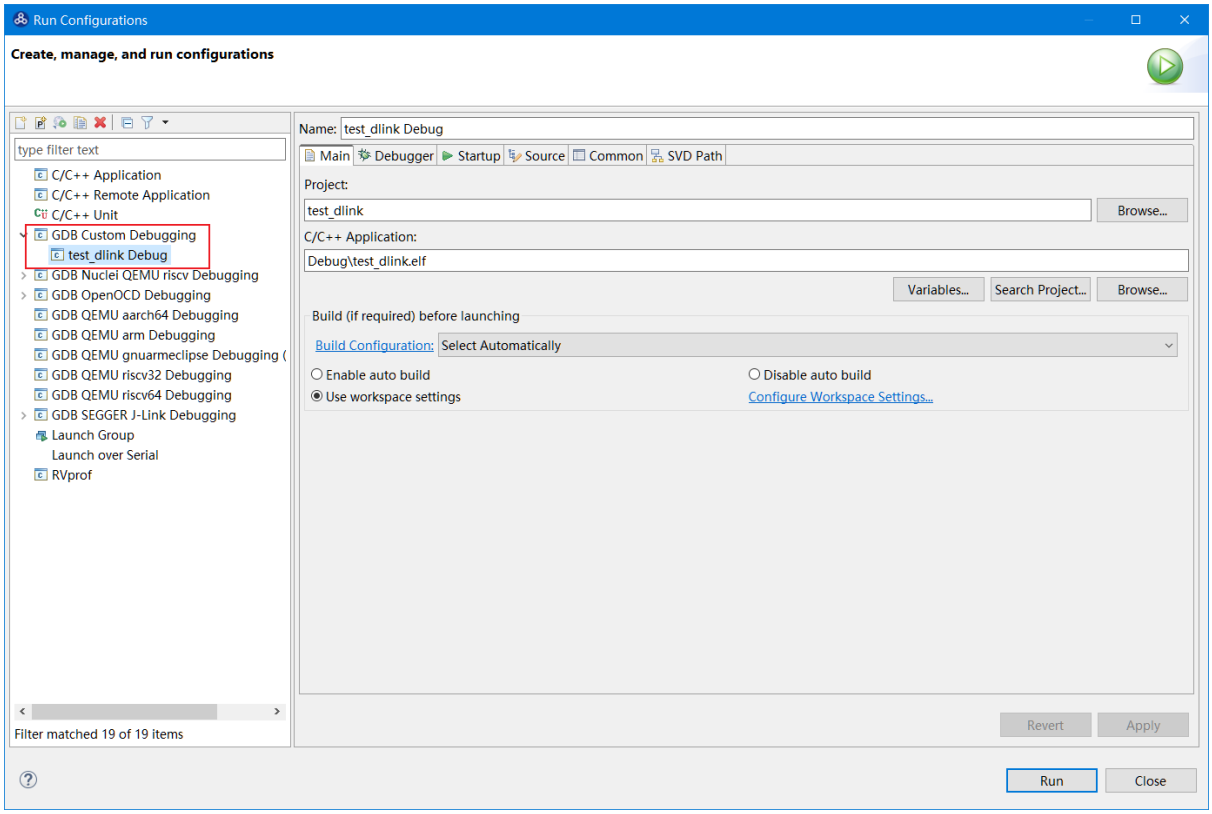


2.9.4 使用 DLink 调试运行项目

DLink 是芯来科技基于 RV Link，并在其基础上做功能迭代升级后，所研发的 RISC-V 调试器，使之更适应于 Nuclei Studio 的应用场景。目前 Dlink 仅针对单核 RISC-V 工程实现调试运行，且已实现量产，具体实物如下，具体关于 Dlink 固件下载参见 https://github.com/Nuclei-Software/nuclei-dlink/wiki/upload-dlink-firmware。

Note: 在芯来科技视频号中有如何在 **Nuclei Studio** 中通过 **Dlink** 调试程的视频，您可以在微信中搜索芯来科技视频号点击查看相关内容。

如需使用 Dlink 进行调试，可以在 NucleiStudio 菜单中 `Run -> Run Configurations` 打开 Run Configurations 的配置页面，并配置一个 Dlink Debug Configuration，双击 GDB Custom Debugging 新建一个配置项，并在 Main 选项卡中配置内容如下：



在 **windows** 环境下安装驱动步骤如下：

打开 GB 网站并搜索 GD32VF1，在列表中打到 GD 32 Dfu Drivers，下载并安装。

地址：https://www.gd32mcu.com/en/download/7?kw=GD32VF1

GD32 Dfu Drivers	3.6.6.6167		none	2020-06-09
Introduction：The drivers for the GD32 Dfu device				

在 **Linux** 环境下安装驱动步骤如下：

- 1：连接开发板到 Linux 中，确保 USB 被 Linux 识别出来。
- 2：在控制台中使用 lsusb 指令查看信息，参考的打印信息如下：

```
Bus 003 Device 057: ID 28e9:018a GDMicroelectronics Dlink Low Cost Scheme
```

- 3：控制台中输入 `sudo vi /etc/udev/rules.d/50-dlink.rules` 指令打开 50-dlink.rules 文件，输入如下内容，保存退出，并执行 `sudo udevadm control --reload`。

```
SUBSYSTEM=="usb", ATTR{idVendor}=="28e9",

ATTR{idProduct}=="018a", MODE="664", GROUP="plugdev"

SUBSYSTEM=="tty", ATTRS{idVendor}=="28e9",

ATTRS{idProduct}=="018a", MODE="664", GROUP="plugdev"
```

- 4：断开调试器再重新连接到 Linux 系统中。
- 5：使用 `ls /dev/ttyACM*` 命令查看 ttyACM 信息，参考输出如下：

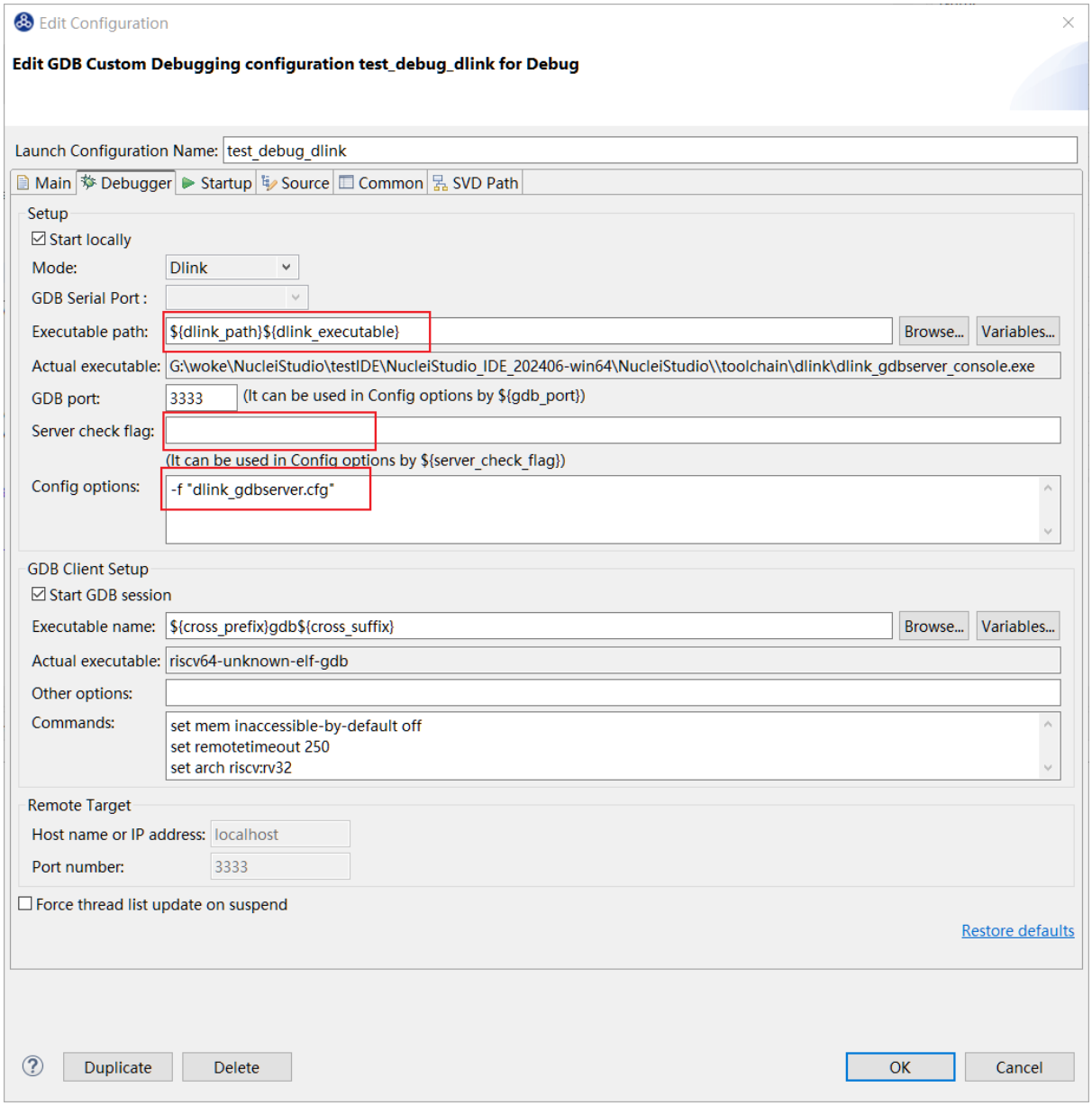
```
/dev/ttyACM0 /dev/ttyACM1
```

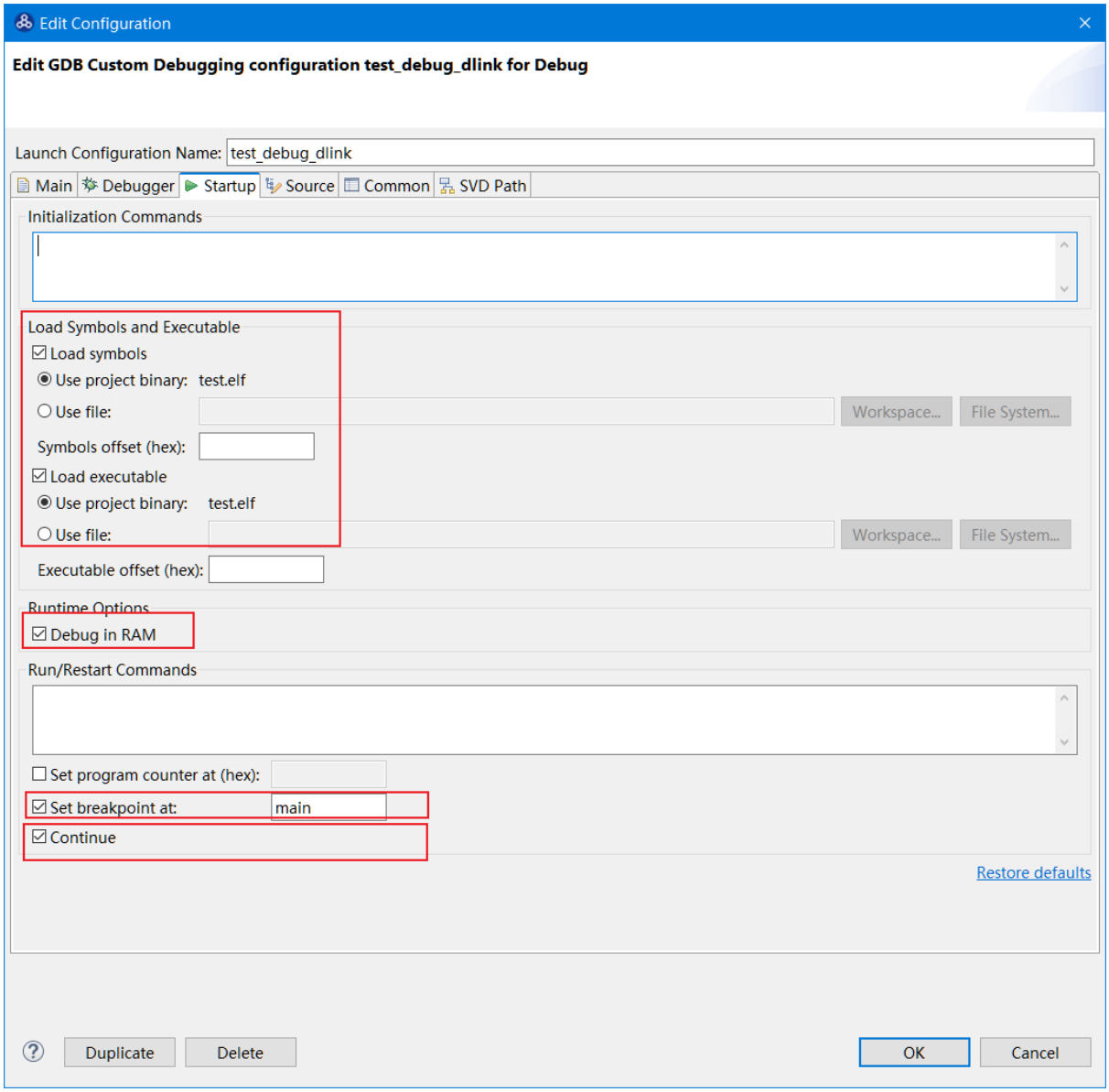
- 6：使用 `ls -l /dev/ttyACM0` 命令查看分组信息，参考输出如下，可以看到 ttyACM0 已经被加入到 dialout 组，接下来我们要将自己添加到 `dialout` 组（不同环境可能名字不同，请根据实际情况修改）。使用 whoami 命令查看当前用户名，我们将其记录为 `< your_user_name >` 。

```
shell crw-rw-r-- 1 root dialout 166, 0 6月 28 15:25 /dev/ttyACM0
```

- 7：使用 `sudo usermod -a -G dialout <your_user_name>` 命令将自己添加进 dialout 组。加入以后一定要重启或者注销操作系统。
- 8：再次确认当前用户名已属于 dialout 组，使用 groups 命令，可以看到打印信息中有 dialout 即成功将当前用户添加至 dialout 组。如果没有可以尝试重启。

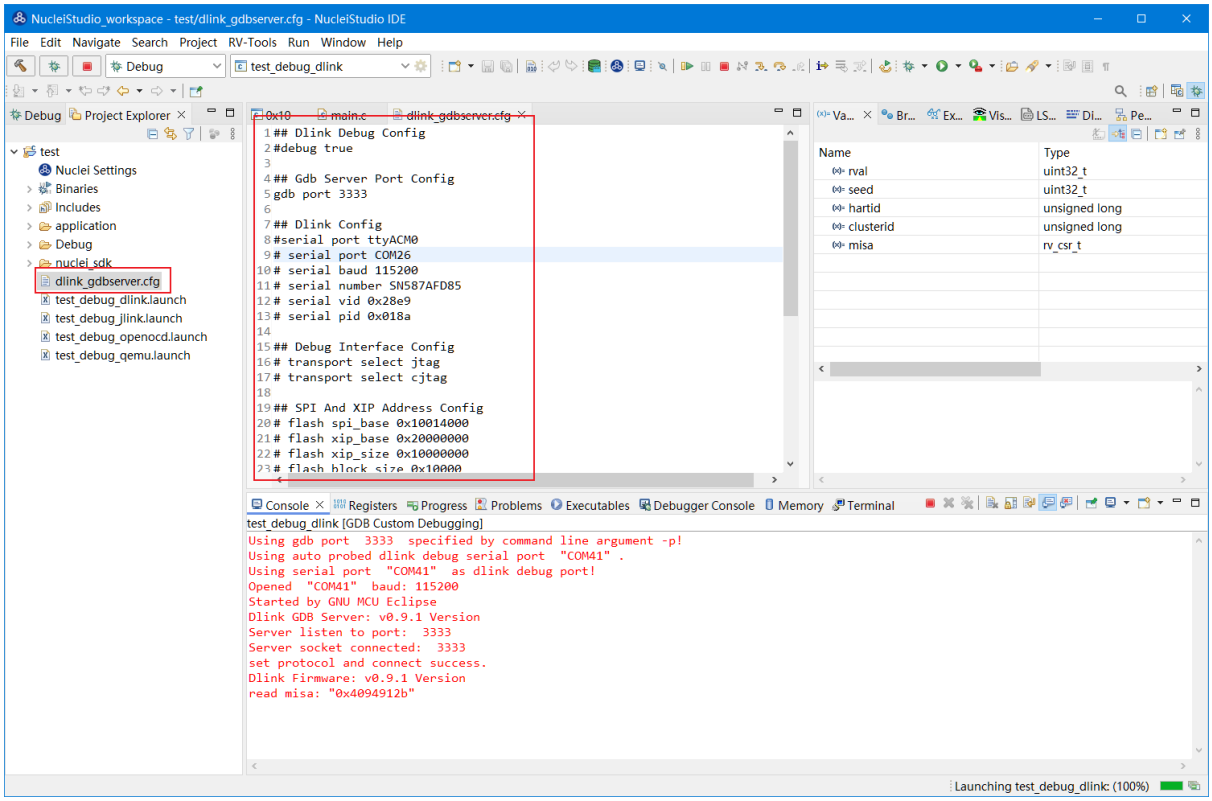
然后在 Debugger 选项卡内配置内容如下，因为在 Custom Debugging 中支持多种 Mode，我们现在需要使用 Dlink，所以选中 Dlink；`Server check flag` 是在 NucleiStudio 中用以确认服务是否正常启动，在 Custom GDB Server 中如果服务正常启动，会输出一段字符串，NucleiStudio 通过判断该字符串以确认 Custom GDB Server 正常启动，在使用 Dlink 时这里可以为空；在 Config options 中需要配置对应的链接文件 `dlink_gdbserver.cfg`，参考配置文件可以在 `<NucleiStudio>/toolchain/dlink` 目录下找到。



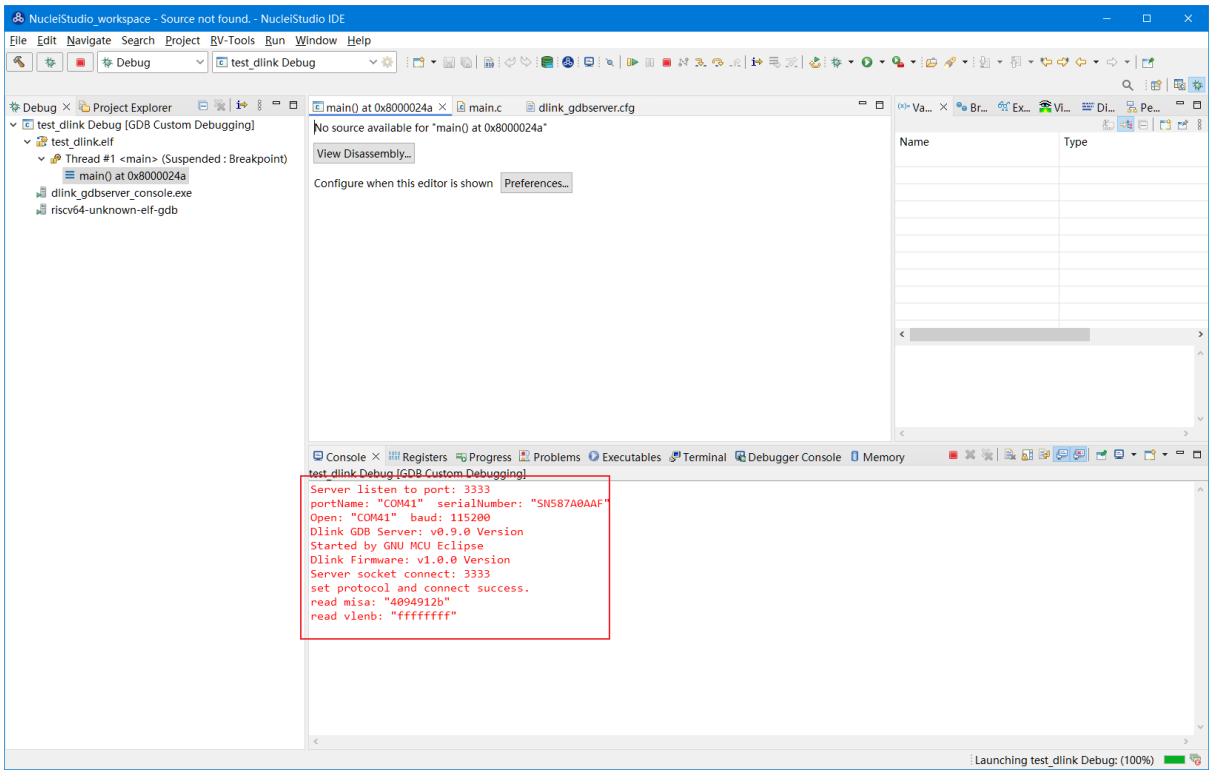


在 windows 下，Dlink 连接上 PC 和开发板后，亮一个绿色灯和一个蓝色灯，说明 Dlink 处理正常工作状态，否则不正常，可以安 NRST 键尝试复位。

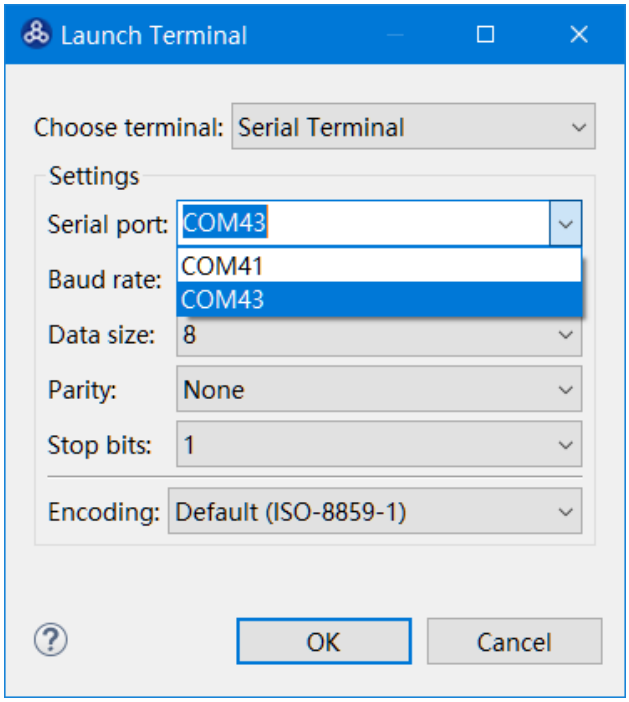
Dlink 连接后，在串口工具下，可以看到两个 COM 口，一个 COM 串用于串口输出，一般情况下数字低的 COM 口是调试的，另一个用于串口数据交换，用户需要在 dlink_gdbserver.cfg 指明用于数据交互的 COM 口，并配置 serial port 和 serial baud，例于在 Windows 下 `serial port COM1`、`serial baud 115200`；在 Linux 下 `serial port ttyACM0`、`serial baud 115200`，如果用户不配置，Dlink 会使用 COM 口中编号较小的那个为 serial port 默认值，并且以其对应的设备号为 serial baud 默认值，以 Windows 下为例，可以参考配置如下：



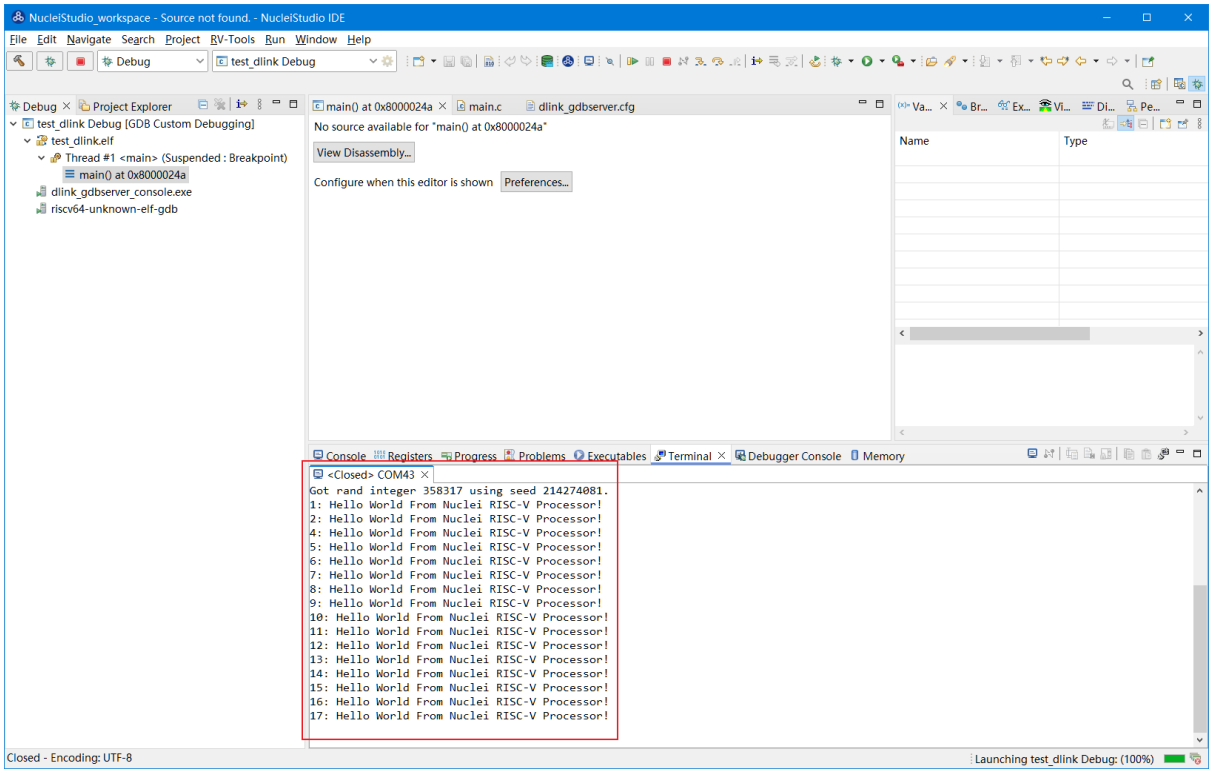
开始 Debug，如果配置正确，则在 Console 中有输出如下，并且 Dlink 亮绿灯。



通过串口工具，联接上另一个串口。



并可以查看到串口中有正确的内容输出，与预期一致，则 Dlink 可以正常调试，其他操作步骤与 OpenOCD 大体一致。



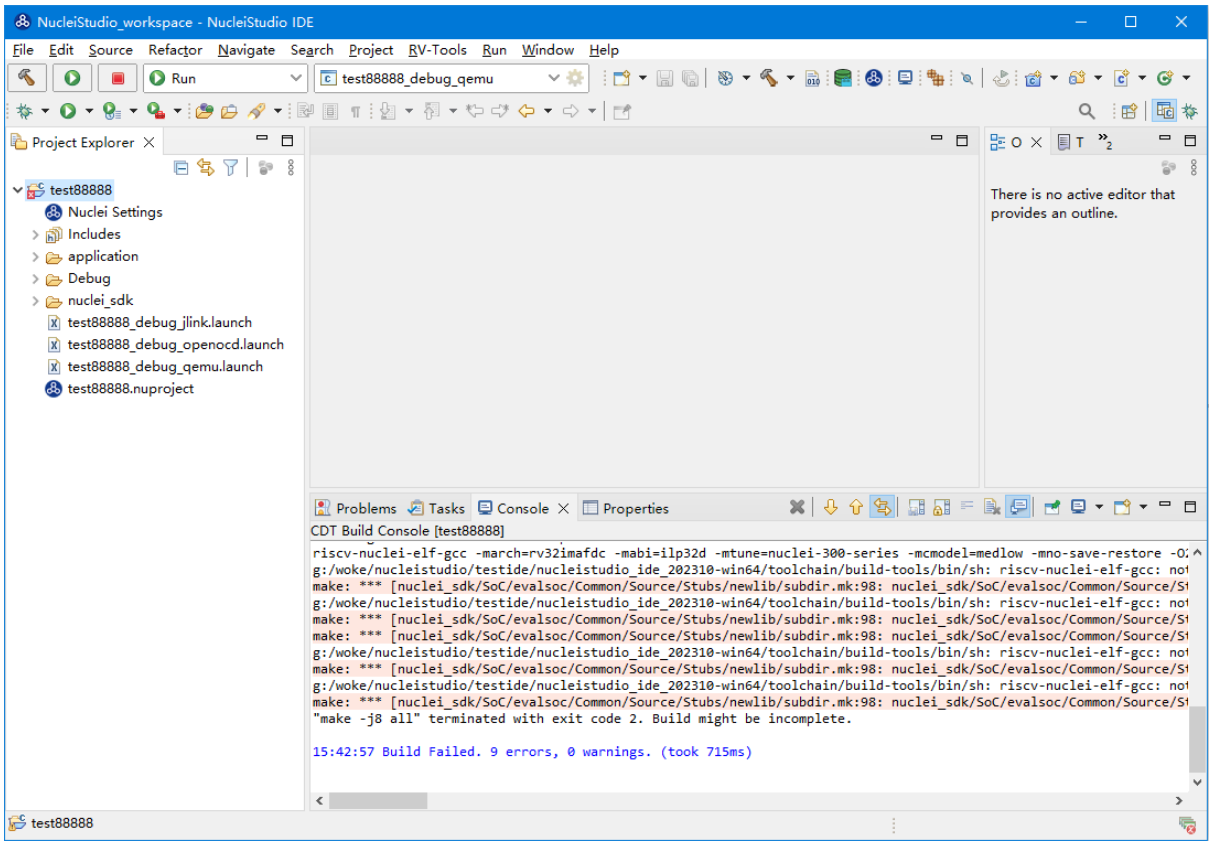
2.10 Nuclei Studio 其它功能

2.10.1 导入旧版本 Nuclei Studio 创建的工程

Nuclei Studio 2023.10 版导导入旧工程

在 Nuclei Studio 2023.10 版本中，因为工具链、sdk 等增均做了较大的修改，如果用户在新的 Nuclei Studio 中想要使用旧版的 Nuclei Studio 创建的工程，或者使用旧的 sdk，需要参考本章节内容进行操作。

将旧的 Nuclei Studio 中的工程导入到 Nuclei Studio 2023.10 中时（具体导入工程的方法，可以阅读*Nuclei Studio 2022.12 之后版本导入旧工程* (page 180)），或者使用旧的 sdk（旧的 sdk 指的是在 Nuclei Studio 2023.10 发布之前所发布的 sdk）所创建的工程，因为工程配置使用使用的是 gcc 10，当找不到对应的工具链，会出现编译报错等问题，导致工程无法正常使用。

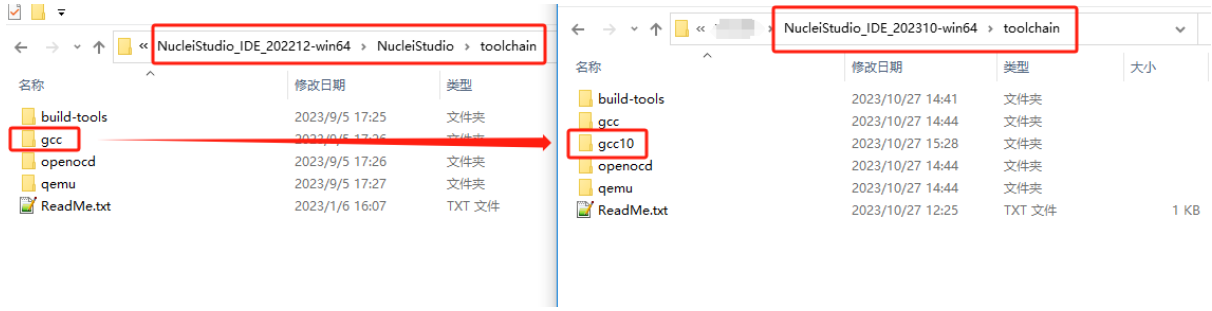


我们提供了两种解决方案，第一种方案是手动导入 gcc 10 工具链，第二种方案是通过 Nuclei Studio 2023.10 所带的转换工具进行转换。在此推荐使用第二种方案，能比较好的将工程转换成 Nuclei Studio 2023.10 所支持的工程，并能使用其最新特性。

手动导入 GCC 10 工具链

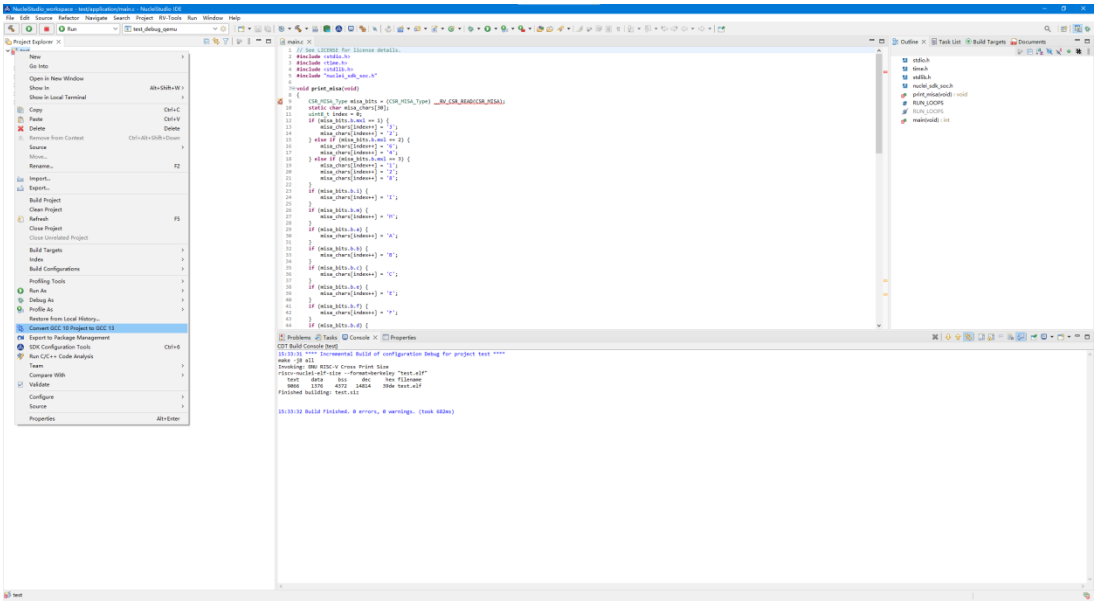
为了方便用户使用，在 Nuclei Studio 2023.10 中保留了 GCC 10 相关的配置，但没有将 GCC 10 打包到 IDE 中，如果用户想要继续在 GCC 10 下进行开发，可以手动将 GCC 导入进来。

首先，在旧版 Nuclei Studio 的安装目录中的 `toolchain\gcc\` 目录找到 GCC 10，并将其中内容复制到 Nuclei Studio 2023.10 的安装目录下的 `toolchain\gcc10\` 内。然后重新编译工程，工程可以正常编译，但这种方法，Nuclei QEMU 是无法正常使用，如果有 Nuclei QEMU 需求的项目，需要收到修改下 QEMU 对应的调试配置。

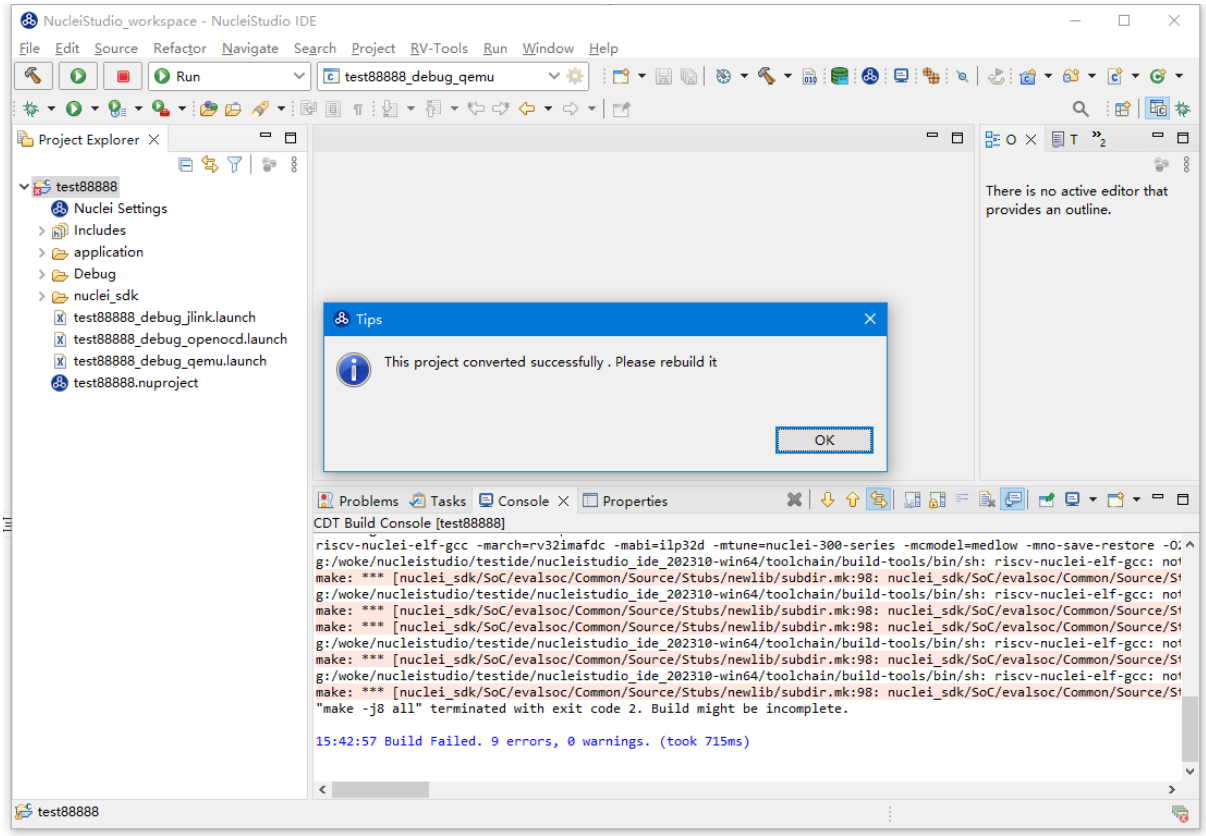


通过工具将工程转换成支持 **GCC 13** 的工程

为了方便用户导入旧的工程，并能正常使用 Nuclei Studio 2023.10 特性，我们提供了快速转换工具 `Convert GCC 10 Project to GCC 13`，选中工程点击鼠标右键，在弹出的菜单中找到 `Convert GCC 10 Project to GCC 13` 并点击。

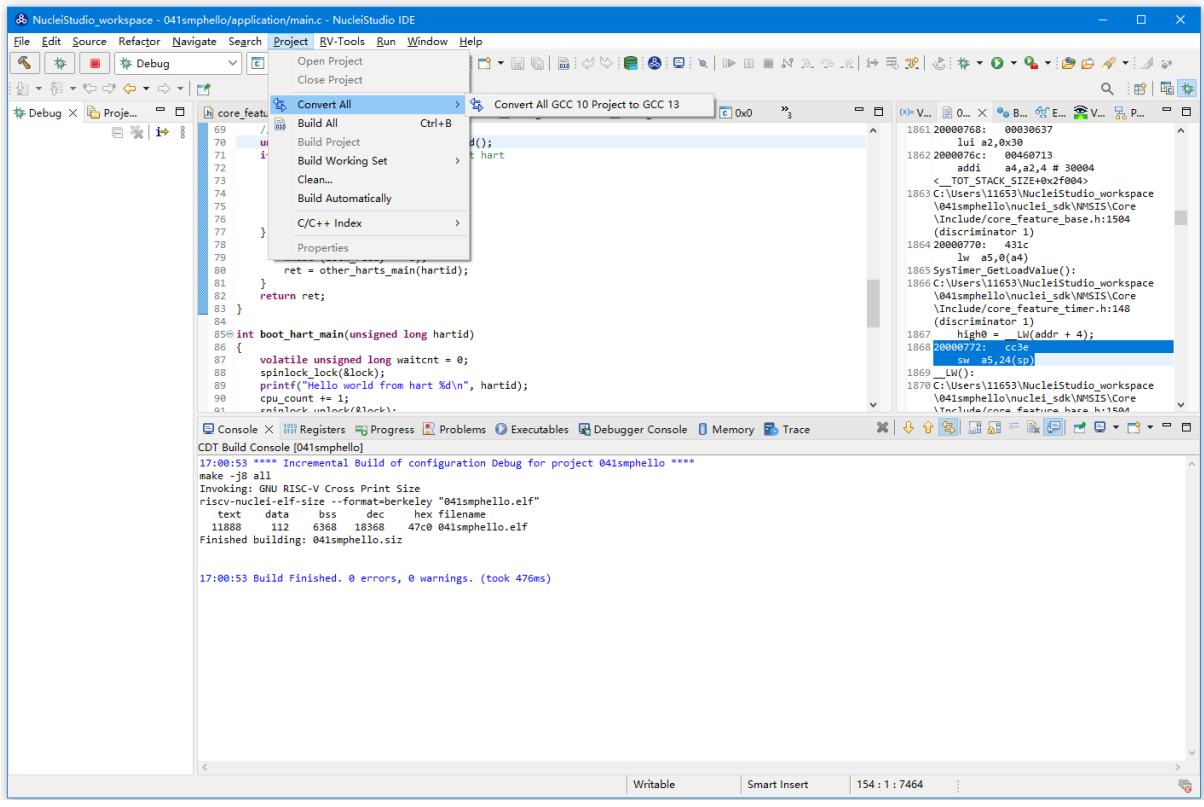


工程转换成功后，重新编译工程，此时 Nuclei Studio 将调用 GCC 13 编译工程，并且 QEMU 等功能也能正常使用。

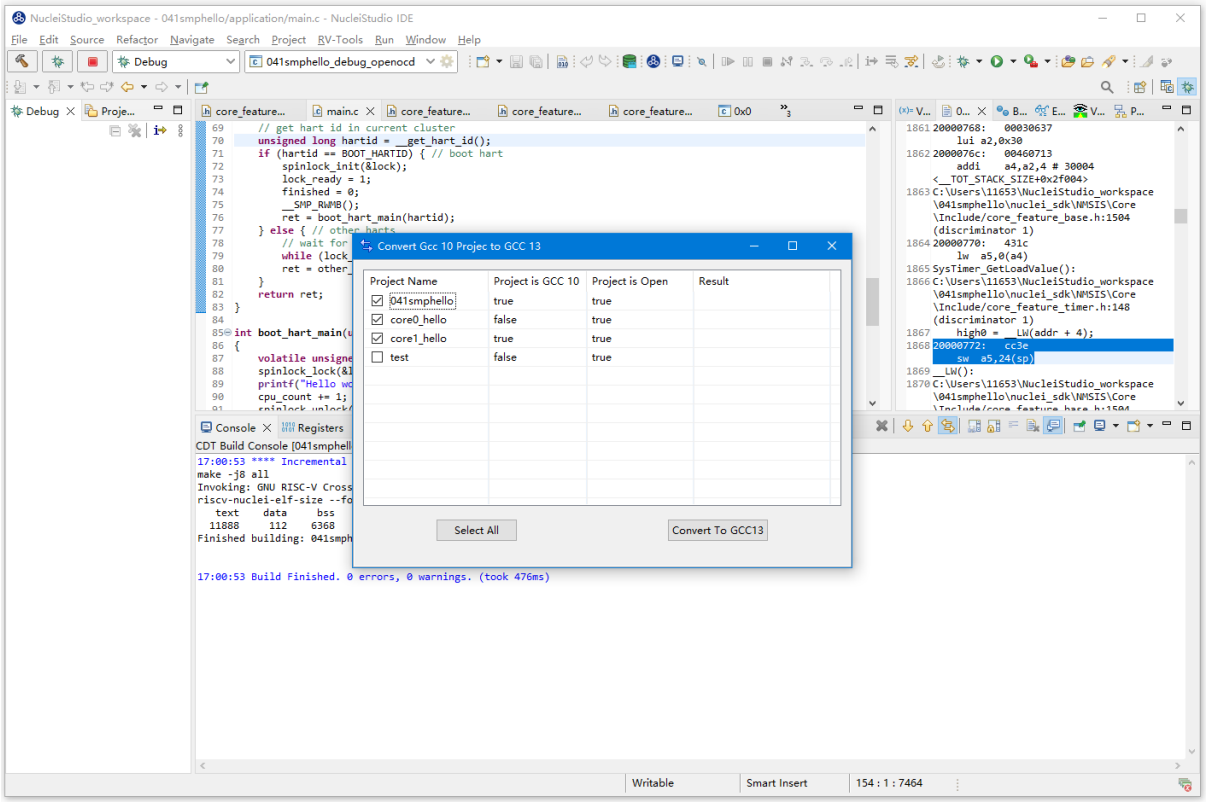


批量将工程转换成支持 GCC 13 的工程

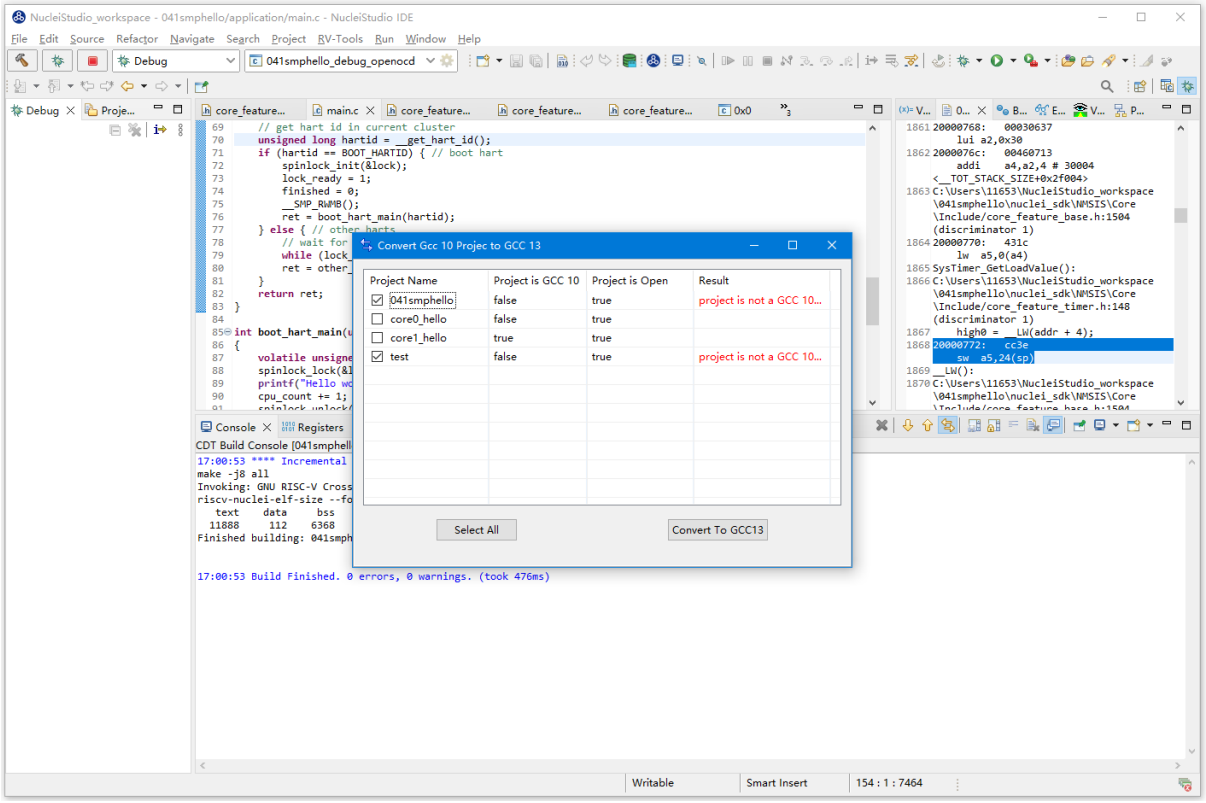
为了方便用户导入旧的工程，并能正常使用 Nuclei Studio 2023.10 以上版本的特性，批量将工程转换成支持 GCC 13 的工程。



打开转换工具，然后选择需要转换的工程，开始转换。



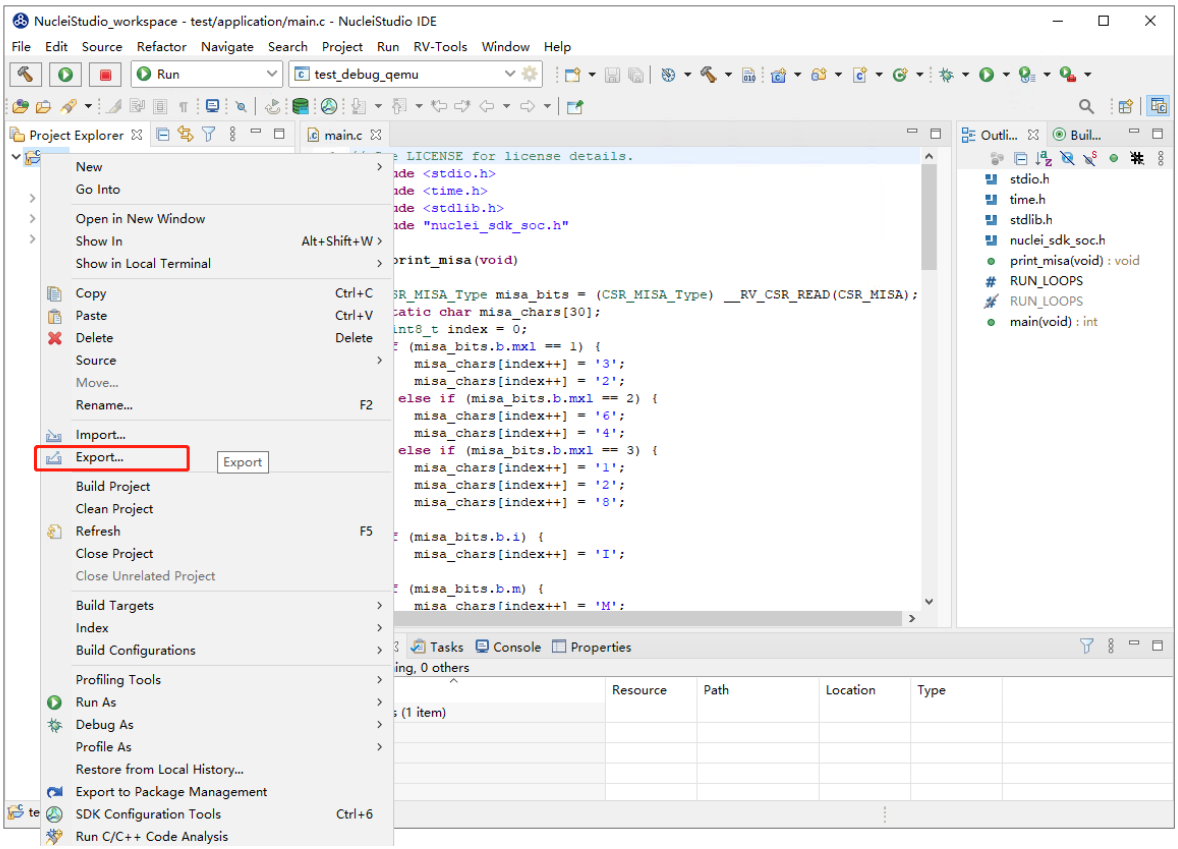
工程转换完成后，页面会显示工程转换成 GCC 13 后的结果。



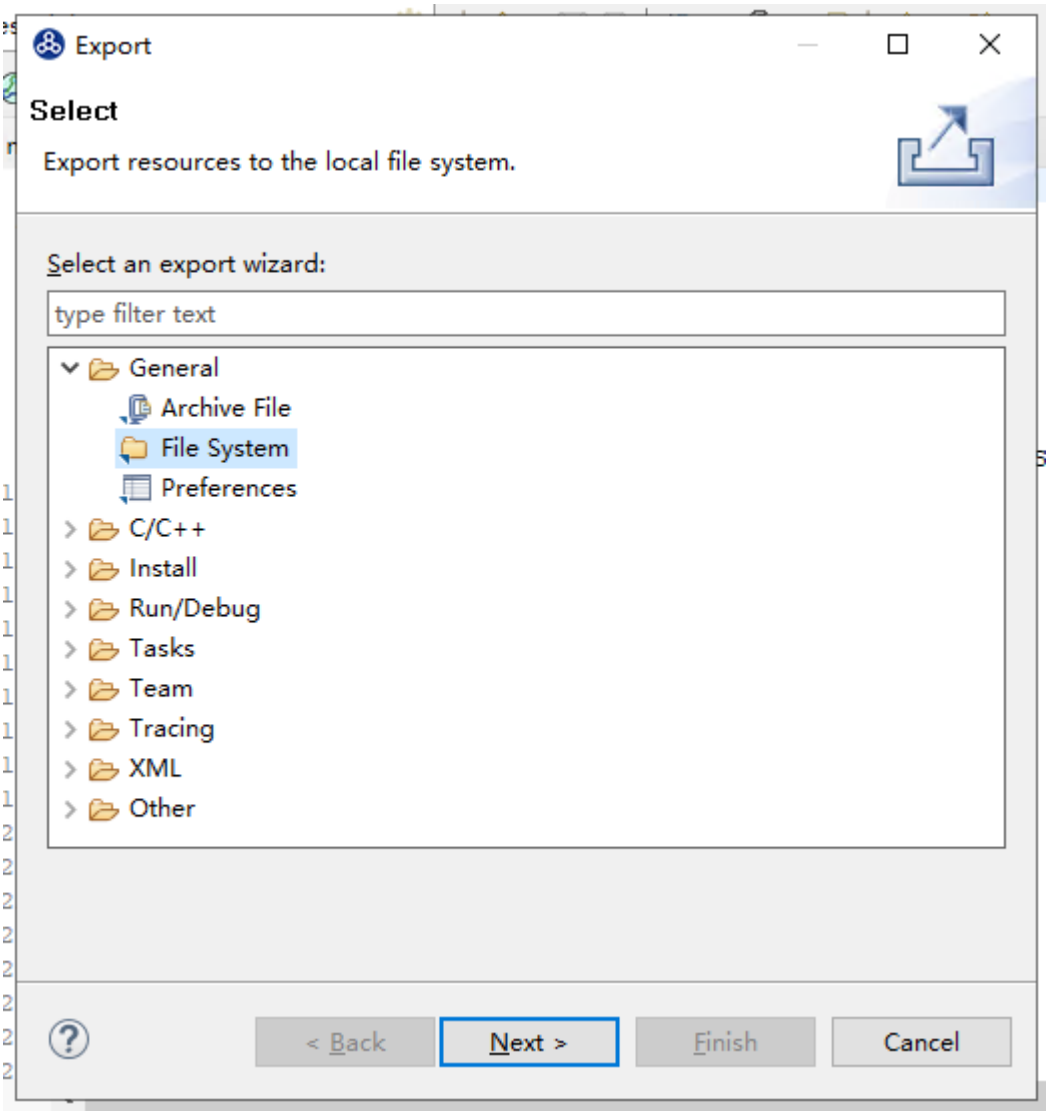
Nuclei Studio 2022.12 之后版本导入旧工程

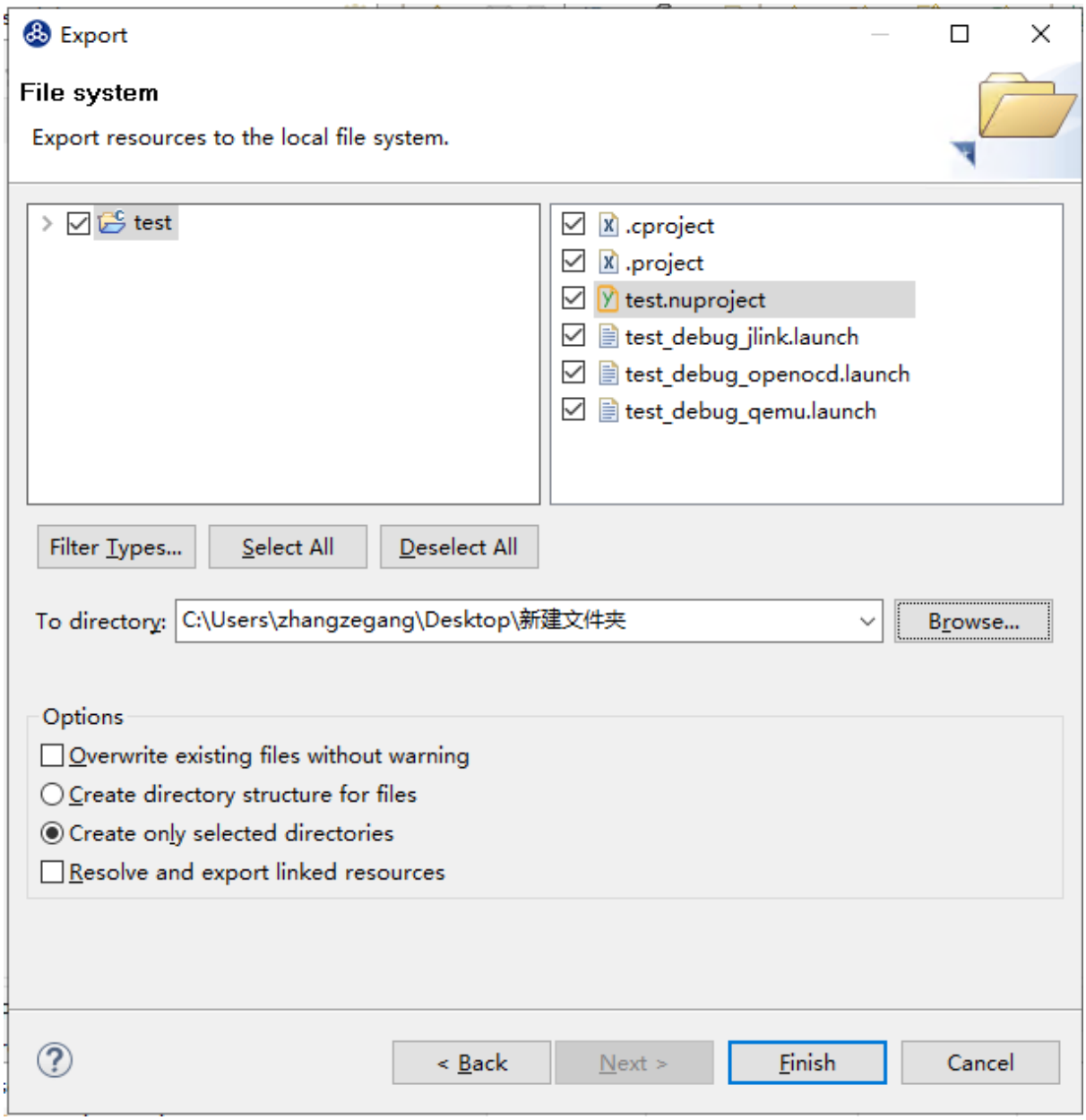
为了方便工程共享，Nuclei Studio 2022.12 版对工程的导入做了优化，在 IDE 中将一个工程导出后，可以双击打开 `.nuproject` 的方式，快速将工程在 Nuclei Studio 中导入并打开。

首先，在 Nuclei Studio 2022.12 版中导出一个工程，在需要导出的工程上右键，选中 Export。

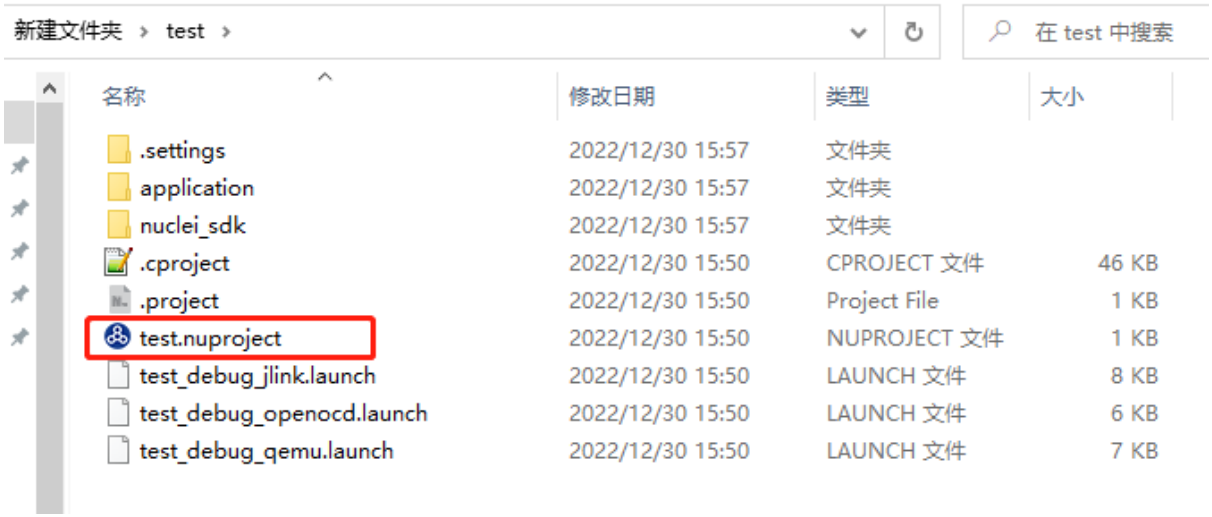


在弹出的 Export 对话框中 `General->File System`，按向导依次操作，可以把工程导出到指定文件夹。


Export
Select
Export resources to the local file system.
Select an export wizard:
type filter text
General
Archive File
File System
Preferences
C/C++
Install
Run/Debug
Tasks
Team
Tracing
XML
Other
< Back
Next >
Finish
Cancel



其次，导入工程。查看导出的工程，可以看到工程中有一个 test.nuproject 文件，点击这个文件，就可以将工程导入到 Nuclei Studio 中去了，具体的可以参考通过应用关联文件导入工程 (page 94)。

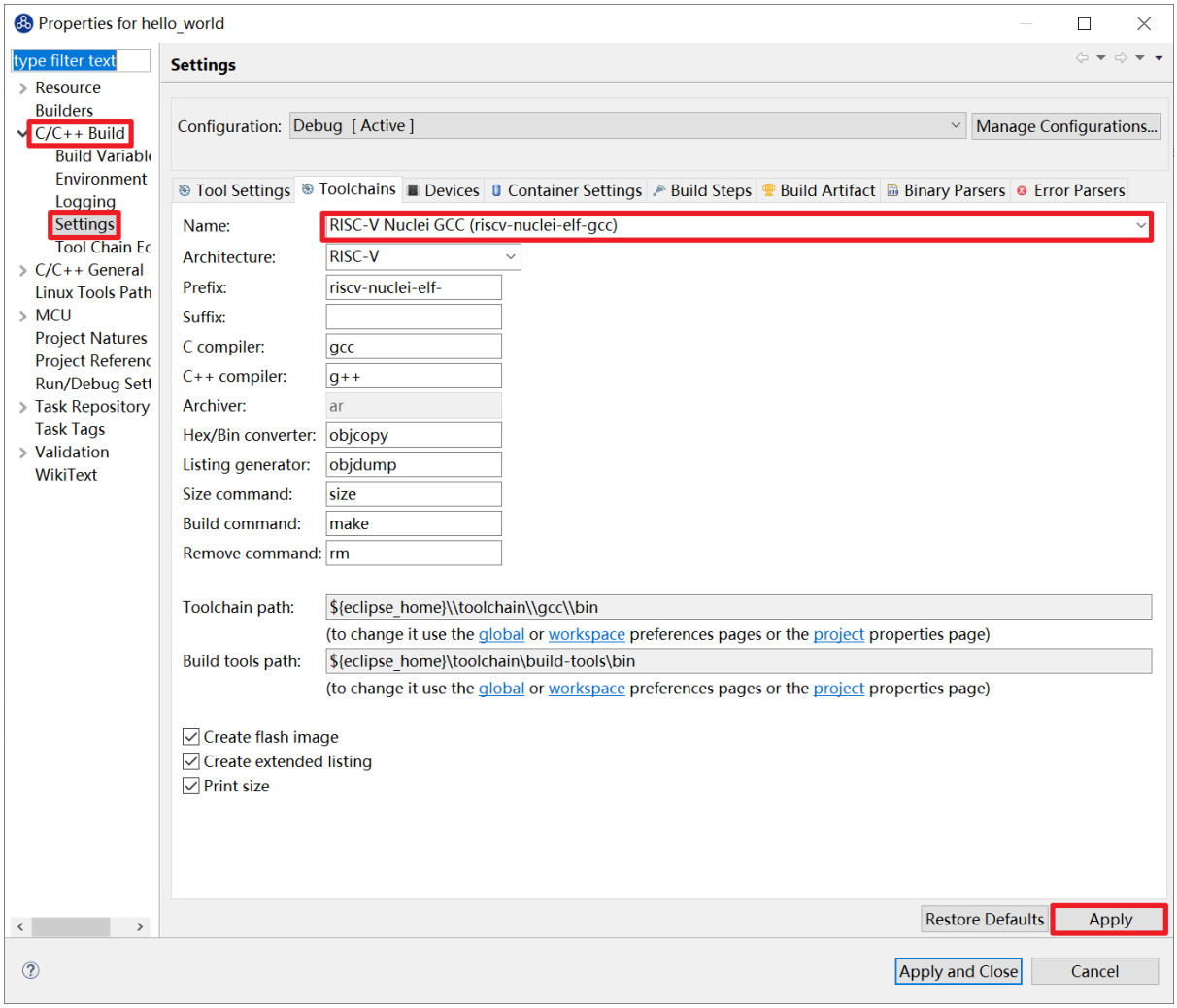


Nuclei Studio 2022.12 之前版本导入旧工程

Nuclei Studio 2023.10 版本工具链名称从 `riscv-nuclei-elf-gcc` 更名为 `riscv64-unknown-elf-gcc`，且 gcc 版本从 gcc10 升级到 gcc13。

Nuclei Studio 从 2020.08 版本开始，官方工具链从 RISC-V Nuclei GCC (`riscv-nuclei-elf-gcc`) 升级到 GNU MCU RISC-V GCC (`riscv-none-embed-gcc`)，因为编译前缀发生变化，所以使用 201909 及其之前版本的 IDE 生成的工程，经过调整设置后才可以在新版本 IDE 中使用。这里以 201909 版本的 Nuclei Studio 生成的 helloworld 工程为例，其导入及修改设置的详细步骤如下：

- 导入 201909 版本生成的 helloworld 工程，详细的导入方式请参考 ?5.2 节，这里不做赘述。
- 导入工程后右击选择 `Properties` 打开设置页面，选择 `C/C++ Build ?? Settings`，打开 Toolchains 栏目然后修改 Name 下拉选项为 `RISC-V Nuclei GCC (riscv-nuclei-elf-gcc)`。修改后点击 Apply 保存修改。

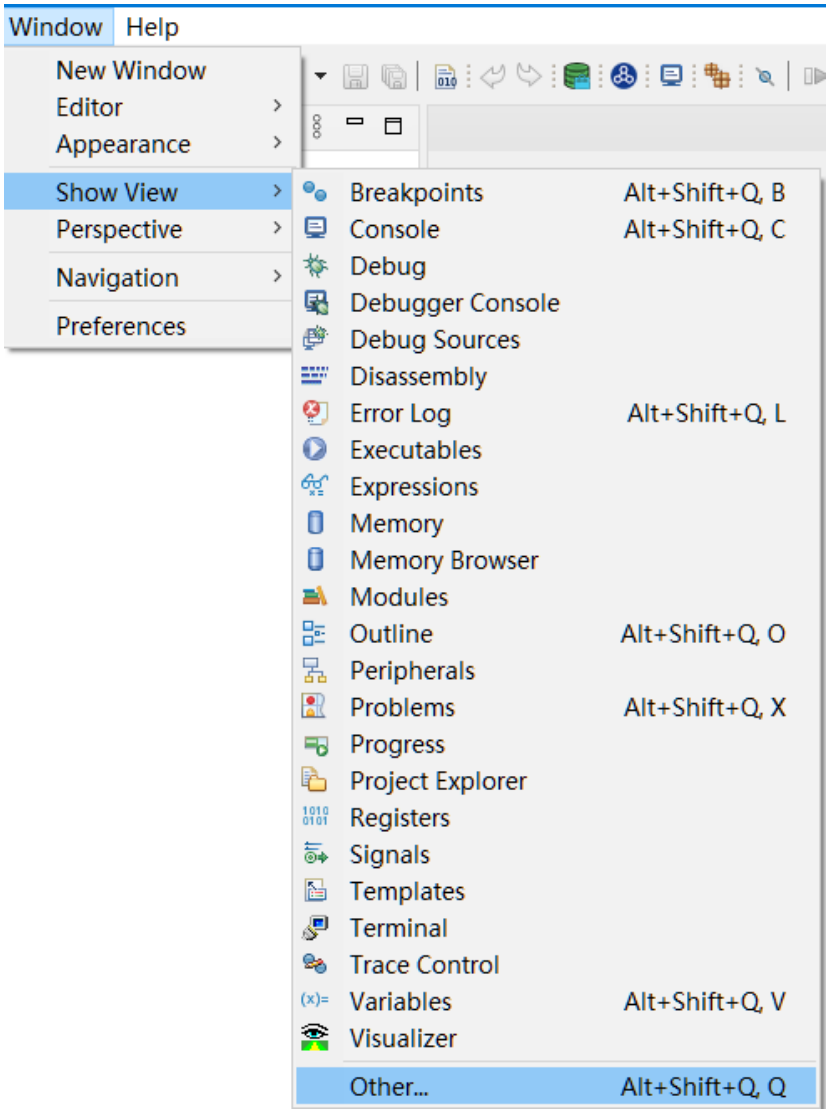


- 修改后右击工程选择 `Clean Project` 再选择 `Build Project` 即可。

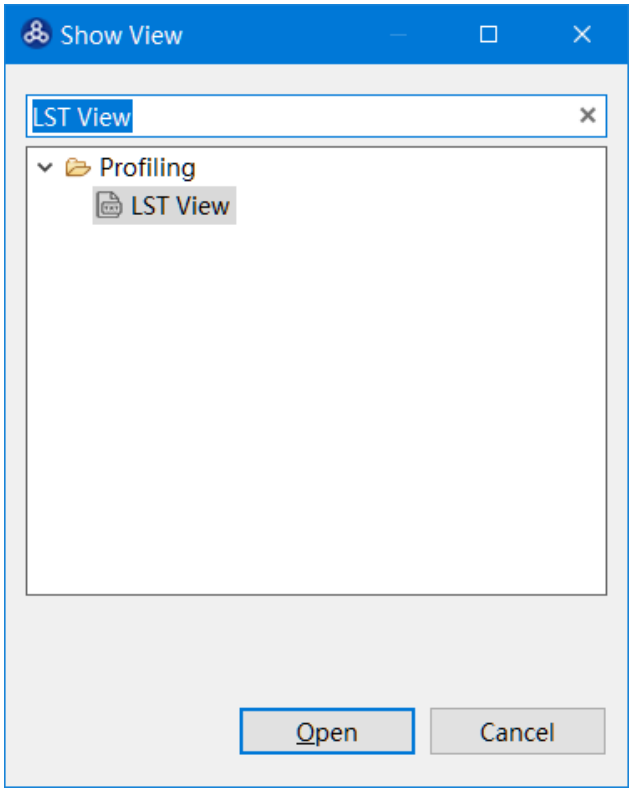
2.10.2 LST View

在 Nuclei Studio 2024.06 版本中，集成了 LST View 工具，LST View 可以单独使用，也可以在 Trace 工具或 GProf 工具中被呼起，主要功能，是帮助用户方便的查看 LST 文件，并实现 LST 文件与源码的联动。

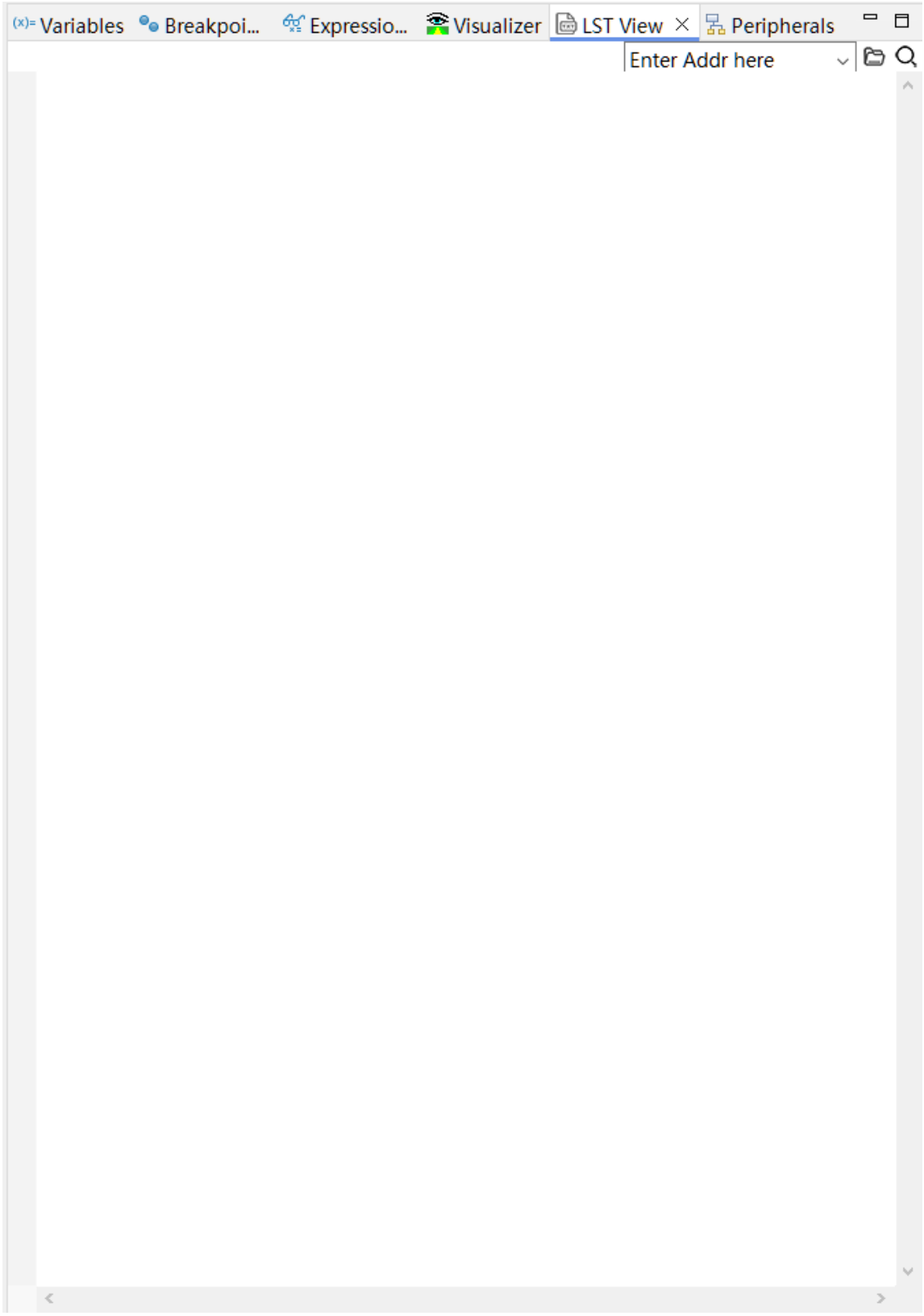
在 Nuclei Studio 中依次 `Window -> Show View -> Other`，在弹出的 Show View 中搜索 `LST View`。



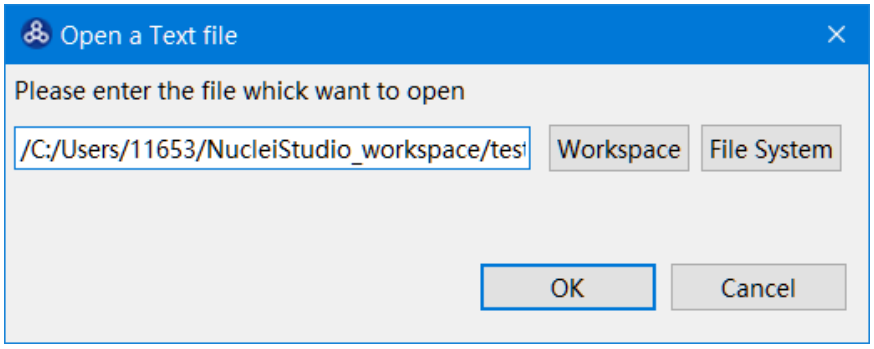
打开 LST View 工具。



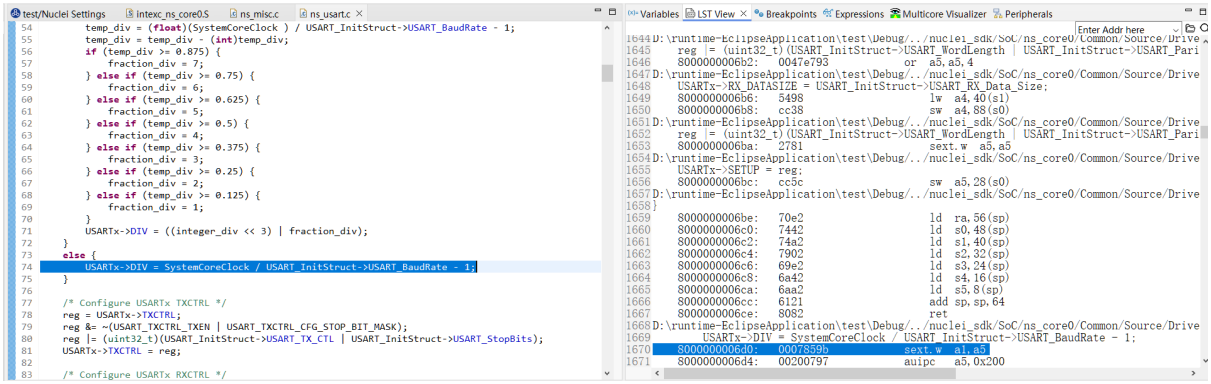
LST View 的顶部有一个工具栏，在工具栏中有搜索下接框，可以输入您想要搜索的 Addr；Open File 菜单，点击会弹出一个文件选择器，可以选择想要打开的 `.lst` 文件；Find 菜单，可以查找任意您想从 LST View 中查找的内容。



Open File 菜单，在弹出的文件选择器中，找到我们想要查看的 lst 文件，一般在工程编译后的 Debug 目录。



打开文件后，可以进行查找操作，同时，当我们选中某行文字，并且文字中包含一个正确的 Addr 时，LST View 会通过这个 Addr 定位到对应的源码所在的文件及行数，并通过程序打开对应的源码文件，并将光标定位到对应的行，通过 lst 文件反定位的源文件，实现两种文件的联动查看。



2.10.3 Code Coverage 和 Profiling 功能

在 Nuclei Studio 2023.10 版以上版本中，集成了Eclipse Linux Tools[7]，并对Eclipse Linux Tools[8]工具进行了部分优化，使其可以支持 Nuclei Studio 工程使用 Code Coverage 和 Profiling 相关功能。在 Nuclei Studio 2024.06 版本中对Eclipse Linux Tools[9]的功能做了进一步的优化和升级，使其更容使用。

关于 Coverage、Profiling 和 Call Graph 的使用教程请查看*Coverage、Profiling 和 Call Graph 使用* (page 191)。

关于 Eclipse Linux Tools 的详细参见 Eclipse Linux Tools[10]

在使用过程，如有问题，可以查看 https://github.com/Nuclei-Software/nuclei-studio 相关内容，也可以向我们提交相关 issue。

Note: 在芯来科技视频号中有如何在 **Nuclei Studio** 中使用 **Code Coverage** 和 **Profiling** 功能的视频，您可以在微信中搜索 芯来科技视频号点击查看相关内容。

[7] https://github.com/eclipse-linuxtools/org.eclipse.linuxtools/blob/master/RELEASE_NOTES.md#eclipse-linux-tools-release-notes
[8] https://github.com/eclipse-linuxtools/org.eclipse.linuxtools/blob/master/RELEASE_NOTES.md#eclipse-linux-tools-release-notes
[9] https://github.com/eclipse-linuxtools/org.eclipse.linuxtools/blob/master/RELEASE_NOTES.md#eclipse-linux-tools-release-notes
[10] https://github.com/eclipse-linuxtools/org.eclipse.linuxtools/blob/master/RELEASE_NOTES.md#eclipse-linux-tools-release-notes

关于 Code Coverage 功能

Nuclei Studio 中的 Code Coverage 功能是借助于 gcc 编译器提供 gcov 工具来查看指定源码文件的代码覆盖率，可以帮助开发人员确定他们的测试用例是否足够充分，是否覆盖了被测代码的所有分支和路径。

在 Nuclei Studio 中，通过给工程中的文件或者文件夹添加 `--coverage` 编译选项编译，在实际开发板上运行时，可以配合 semihost 功能实现文件读写到主机电脑上，就可以收集到需要的 coverage 文件 (gcda/gcno 文件)，或者通过 Nuclei SDK 提供的 profiling 库[11] 来实现将 coverage 数据打印到串口上，然后通过 IDE 来解析并保存到主机上。

Note: 注意：此处只需要将编译选项 `--coverage` 加到特定的应用目录或者源码文件上，而不能加到整个工程，否则在程序运行时将会消耗大量内存，导致运行失败。

- `.gcno` 文件是在使用 GCC 编译器的 `-ftest-coverage` 选项编译源代码时生成的。它包含了重构基本块图和为块分配源代码行号的信息。
- `.gcda` 文件是在使用 GCC 编译器的 `-fprofile-arcs` 选项编译的目标文件运行时生成的。每个使用该选项编译的目标文件都会生成一个单独的 `.gcda` 文件。它包含了弧转移计数、值分布计数以及一些摘要信息。

而一般情况下直接使用 `--coverage` 选项就可以让指示编译器产生上述文件，注意 `*.gcda` 文件是运行时产生的，也就是说需要实际运行的环境支持文件的读写才可以产生这样的文件，这里我们采用的是 semihost 技术，通过 openocd 的 semihost 功能，将文件写到主机上。

Note: 注意：进行 coverage 的时候，建议是使用 `O0` 编译，这样 coverage 的信息才会尽可能的准确。

关于 Code Coverage 的功能详细参见

- Gcov Intro (Using the GNU Compiler Collection (GCC))[12]
- Gcov Data Files (Using the GNU Compiler Collection (GCC))[13]
- Code Coverage for Embedded Target with Eclipse, gcc and gcov | MCU on Eclipse[14]

关于 Profiling 功能

Nuclei Studio 中的 Profiling 功能是借助于 gcc 编译器和 binutils 中的 gprof 工具，来查看指定文件中函数的运行时间和调用次数，以及调用关系。gprof 可以用来确定程序的瓶颈，以便进行性能优化。gprof 通过在程序运行时收集数据来工作，然后生成一个报告，该报告显示每个函数在程序中占用 CPU 时间的百分比以及函数之间的调用关系。

在 Nuclei Studio 中，通过带特定的编译选项 `-pg` 编译指定源码文件，在实际开发板上运行时，可以配合 semihost 功能实现文件读写到主机电脑上，就可以收集到需要的 coverage 文件 (gcda/gcno 文件)，或者通过 Nuclei SDK 提供的 profiling 库[15] 来实现将 coverage 数据打印到串口上，然后通过 IDE 来解析并保存到主机上。

Note: 注意：此处只需要将编译选项 `-pg` 加到特定的应用目录或者源码文件上，而不能加到整个工程，否则在程序运行时将会消耗大量内存，导致运行失败。

产生这个 `gmon.out` 文件需要配合编译器并且实际上板运行，并且运行环境支持文件的读写，才可以进行有效的 Profiling 功能。

关于 Profiling 的功能详细参见

[11] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/Components/profiling
[12] https://gcc.gnu.org/onlinedocs/gcc/Gcov-Intro.html
[13] https://gcc.gnu.org/onlinedocs/gcc/Gcov-Data-Files.html
[14] https://mcuoneclipse.com/2014/12/26/code-coverage-for-embedded-target-with-eclipse-gcc-and-gcov/
[15] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/Components/profiling

- Introduction (GNU gprof)[16]
- Using GNU Profiling (gprof) With ARM Cortex-M - DZone[17]

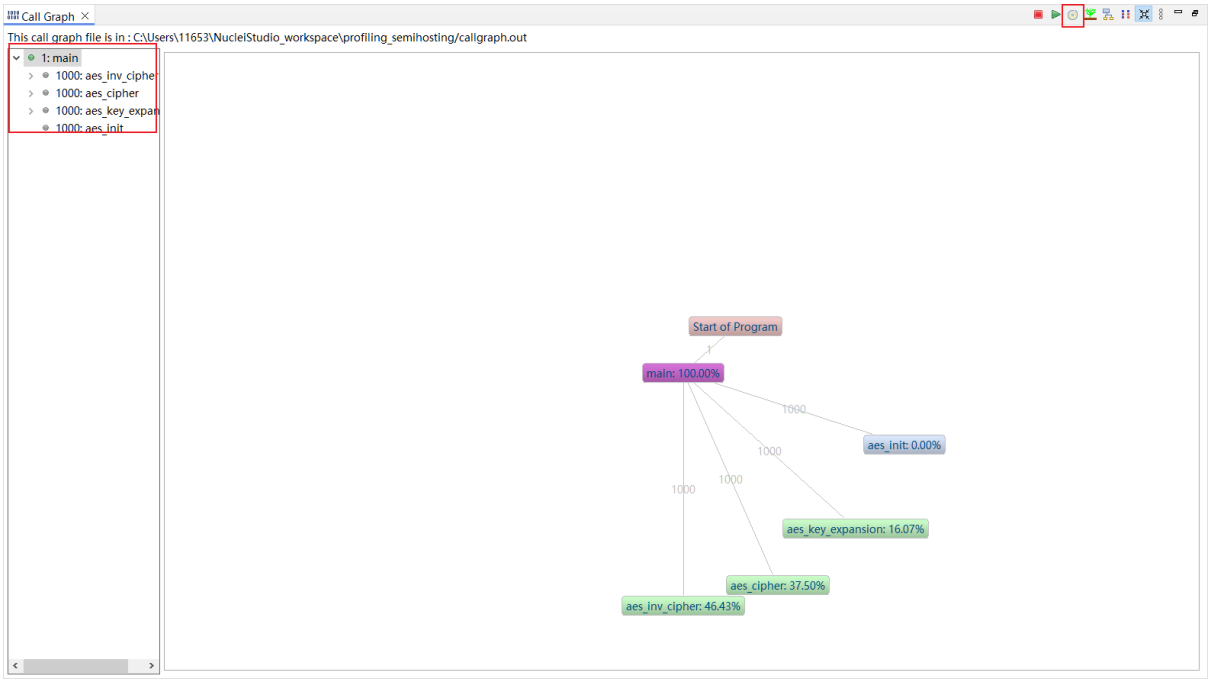
关于 Call Graph 功能

Call Graph（调用图）是一个强大的工具，它允许开发人员直观地理解程序中函数或方法之间的调用关系。通过 Call Graph，开发人员可以迅速识别出哪些函数被频繁调用，哪些函数是关键的入口点，以及函数之间的依赖关系。Nuclei Studio 中 Call Graph 主要是通过分析 Profiling 的数据，来获取到程序的调用关系。

在 NucleiStudio 中依次 `Window -> Show View -> Other`，在弹出的 Show View 中搜索 `Call Graph`，打开 `Call Graph` 工具。`Call Graph` 工具中提供了多处视图，其中常用到的视图有以下几个。

Radial View

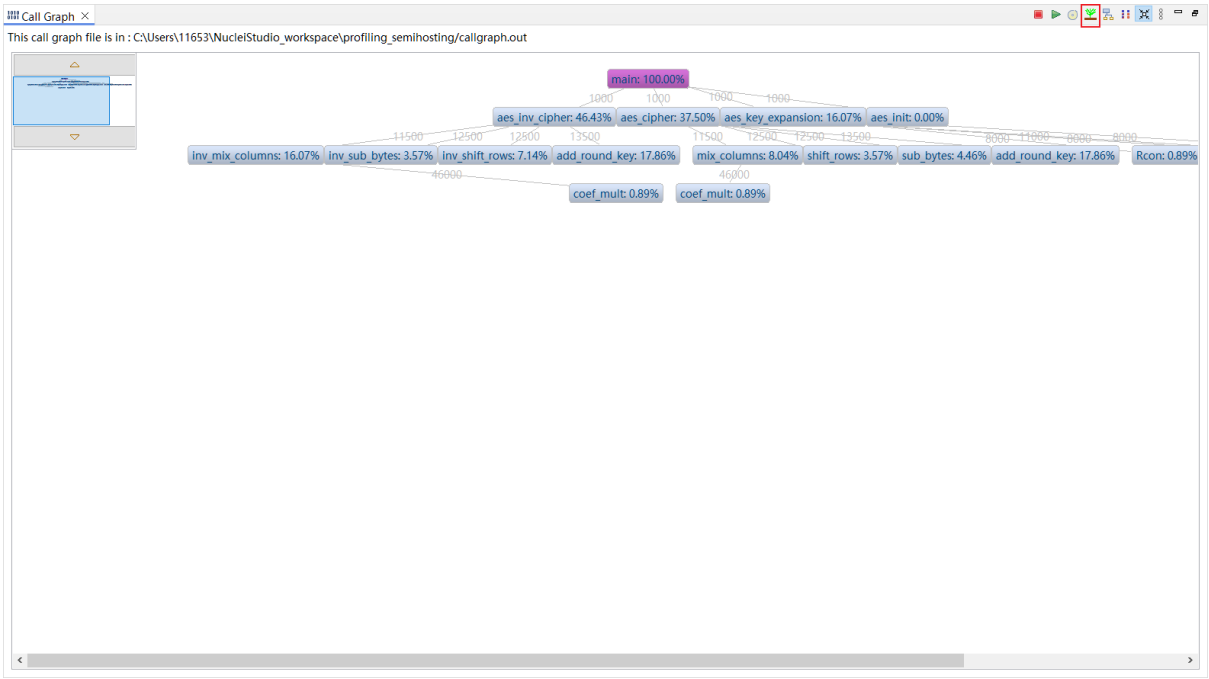
本视图中展示了程序的调用关系，在左侧的菜点中，双击选中某个父节点，在右侧的区域将显示以这个父节点开始的所有的调用关系，也可以通过菜单在其他视图中以不同的方式查看所选中的调用关系。



Tree View

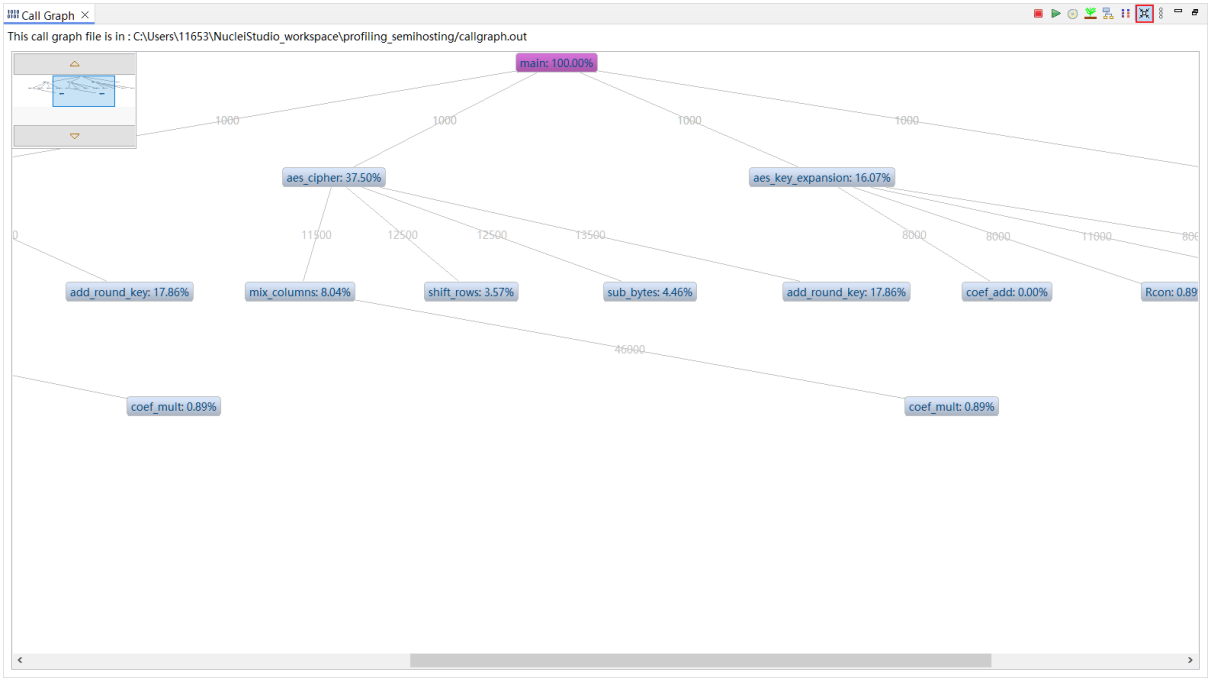
展示了 Radial View 中所选中的程序的调用关系、耗时所占比率、调用次数等信息；选中某一个函数，可以查看到它的父节点以及子节点等信息。

[16] https://sourceware.org/binutils/docs/gprof/Introduction.html
[17] https://dzone.com/articles/using-gnu-profiling-gprof-with-arm-cortex-m



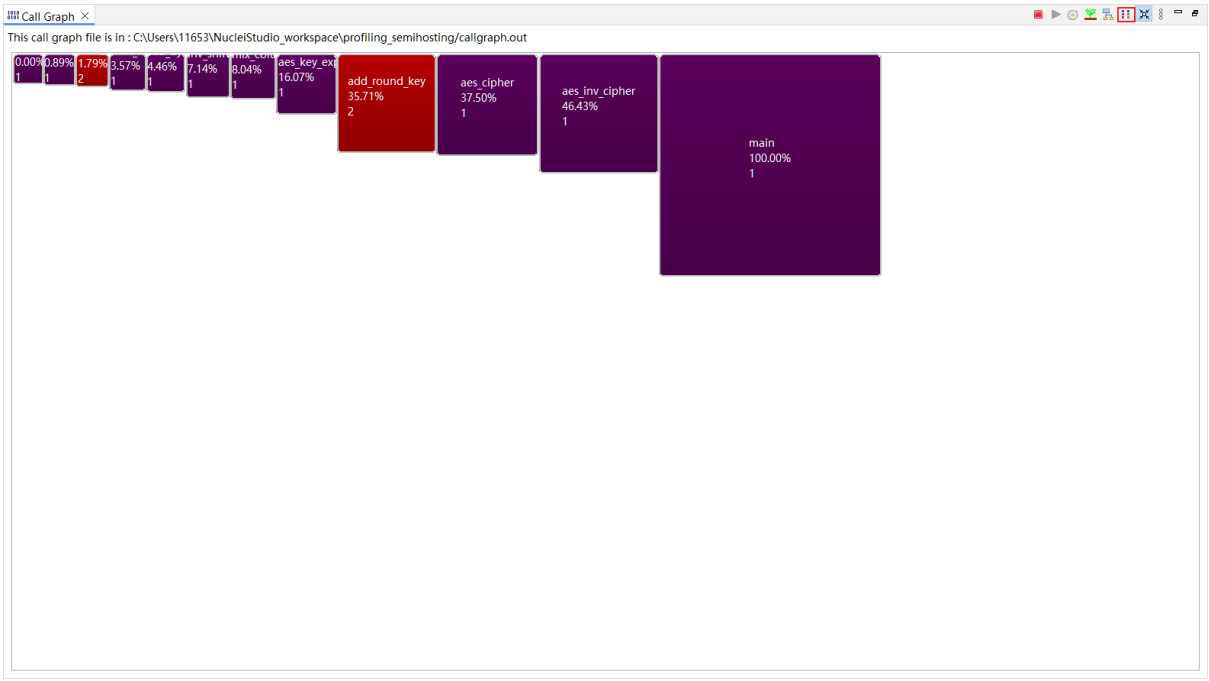
Level View

与 Tree View 有点类似，展示了程序的调用关系以及调用次数。



Aggregate View

以方图的方式，非常直观的展示了程序的耗时关系。



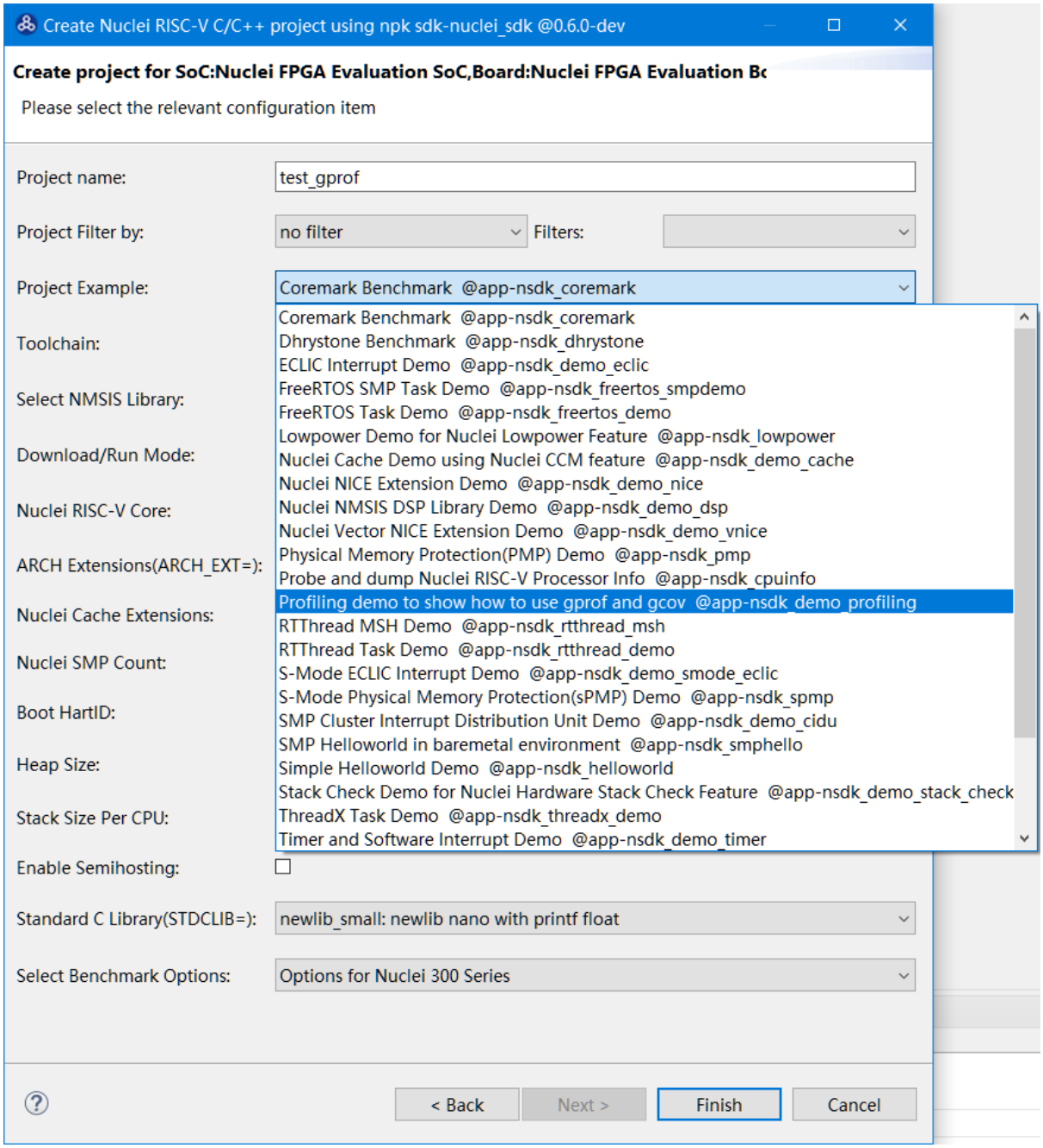
Coverage、Profiling 和 Call Graph 使用

在 NucleiSudio 2024.06 版中使用 Coverage、Profiling 和 Call Graph 方法很简单，下面以 NucleiSudio 2024.06、nuclei_sdk 0.6.0 为例，通过两种方式分别演示如何使用 Coverage、Profiling 和 Call Graph 工具。

通过串口使用

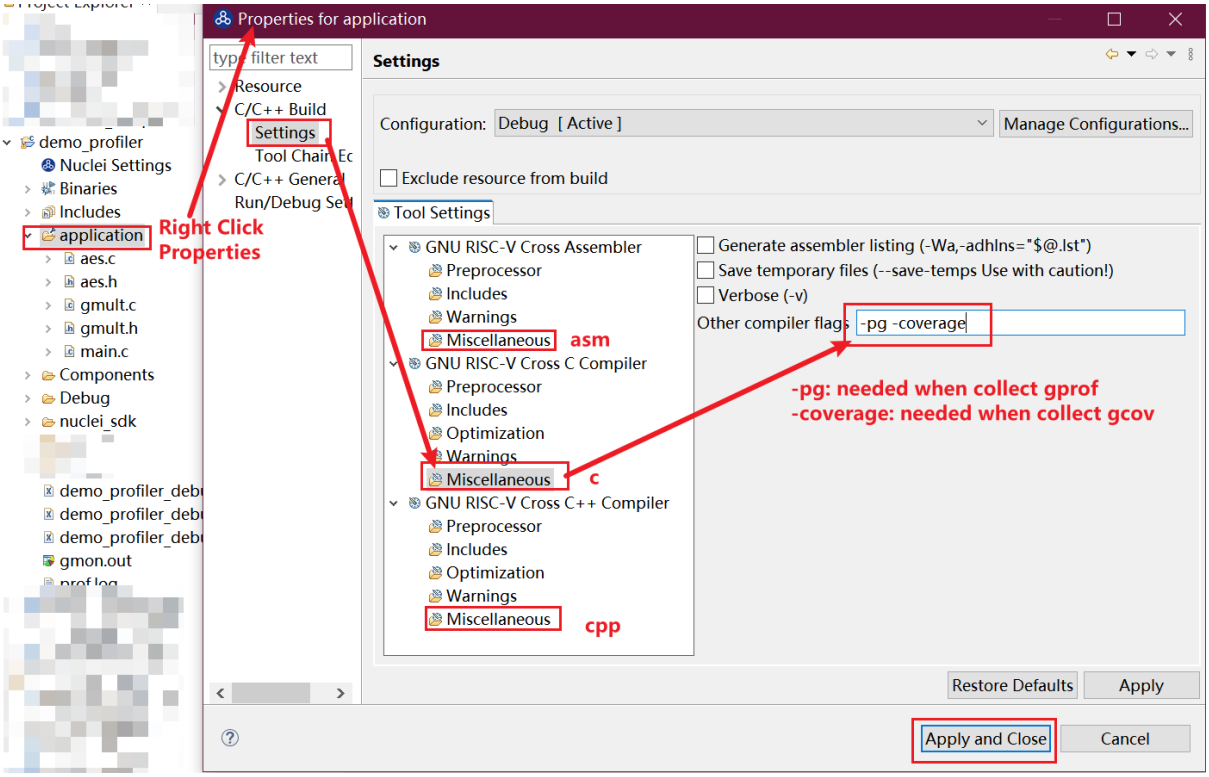
nuclei_sdk 0.6.0 及以上版本的 nulclei_sdk 中，包含一个 `Profiling demo to show how to use gprof and gcov` 测试工程，在 NucleiSudio 安装了 nuclei_sdk 0.6.0 后，可以创建此测试工程。关于 `Profiling demo to show how to use gprof and gcov` 测试工程，可参考 demo_profiling[18] 。

[18] https://doc.nucleisys.com/nuclei_sdk/design/app.html#demo-profiling

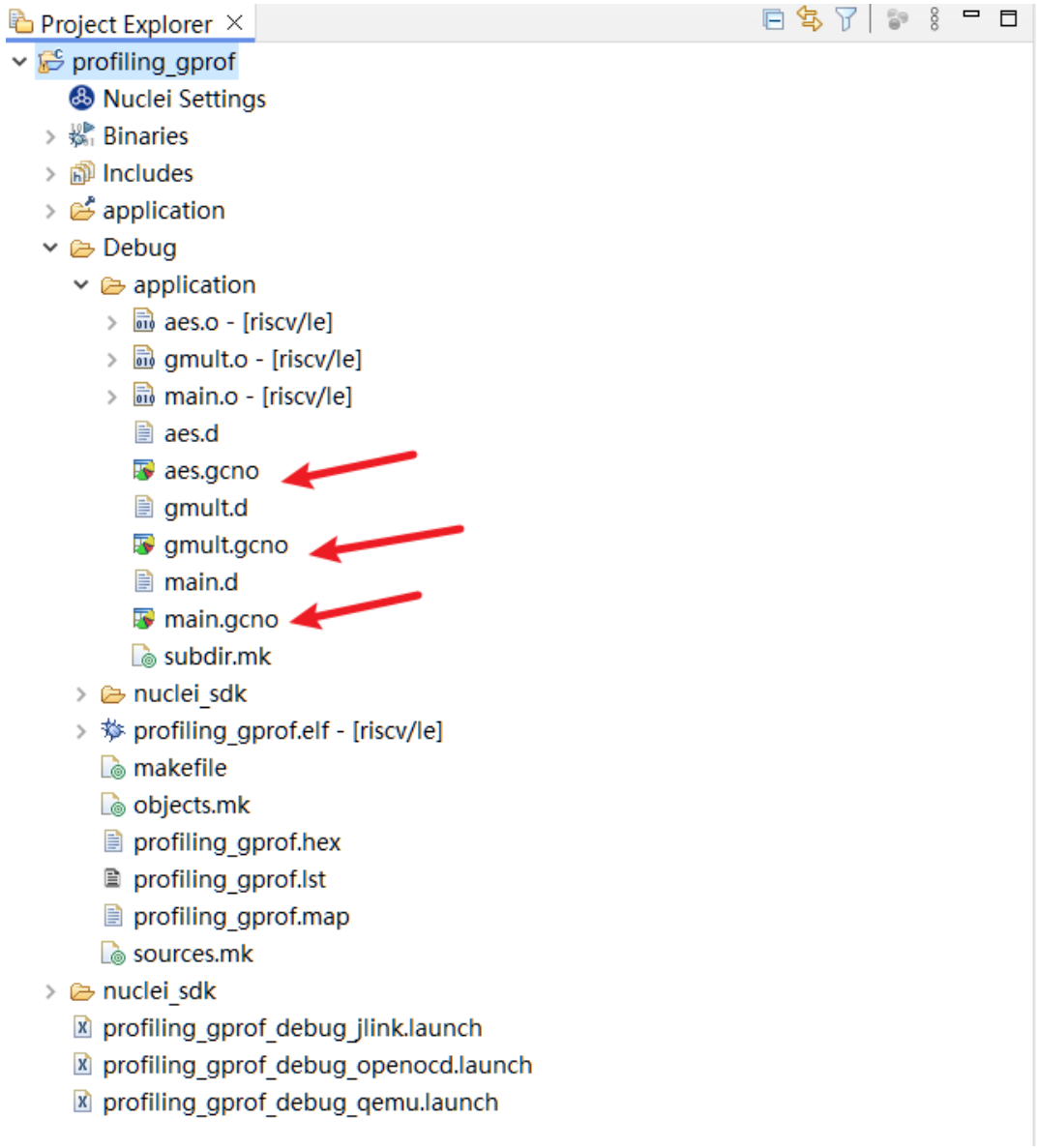


工程创建后，需要对想要进行代码分析的文件或文件夹设置一个 `-pg --coverage` 的编译选项，然后编译工程。

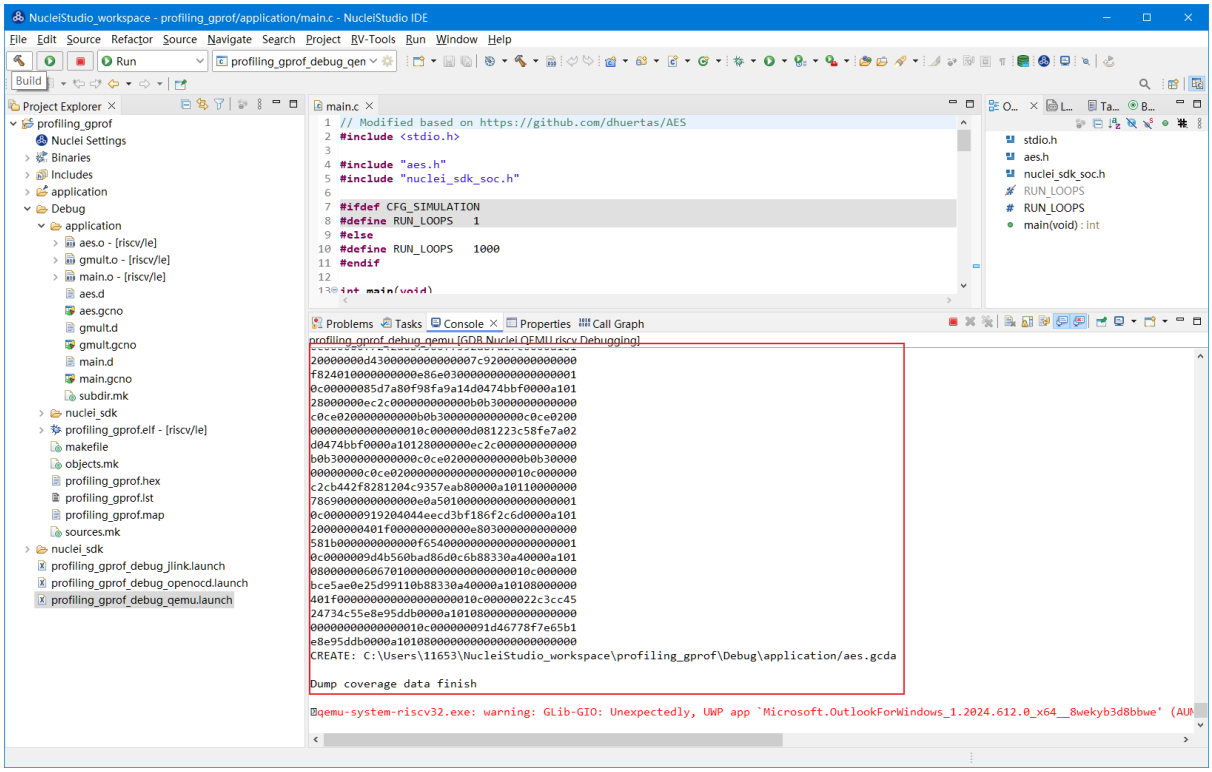
Note: 注意：此处只需要将编译选项 `-pg --coverage` 加到特定的应用目录或者源码文件上，而不能加到整个工程，否则在程序运行时将会消耗大量内存，导致运行失败。



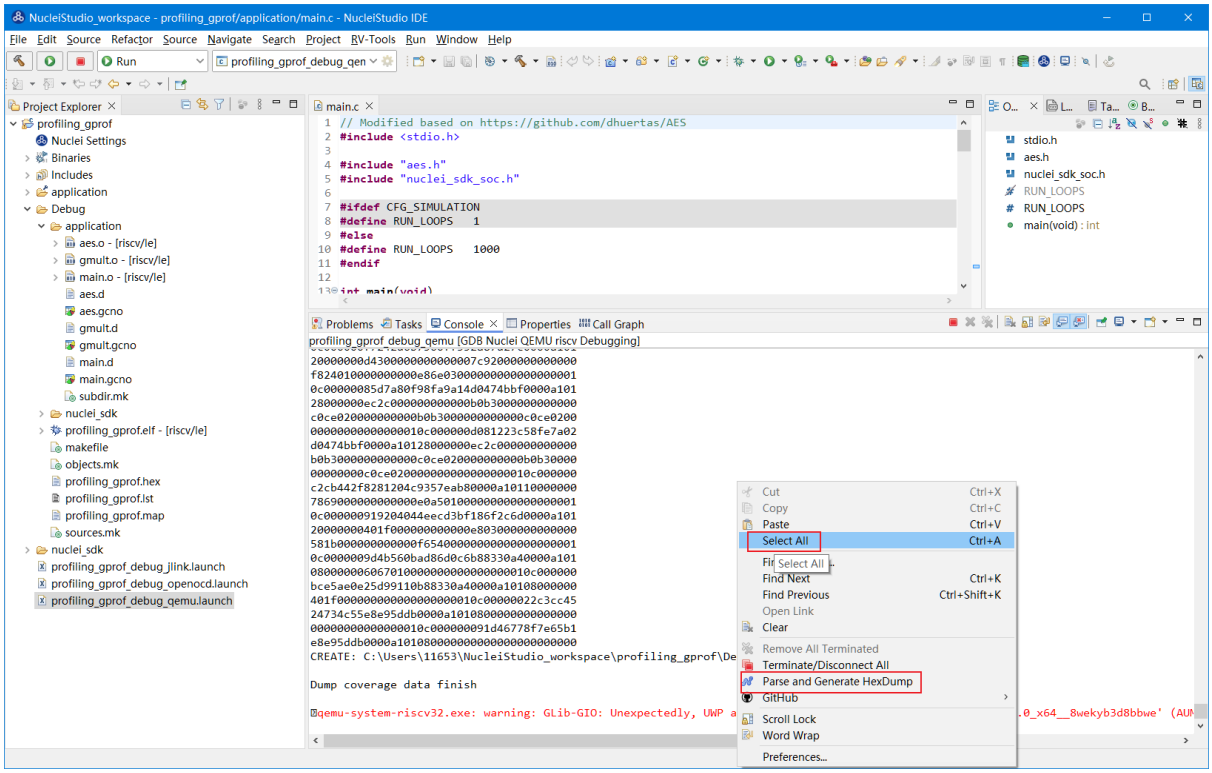
在编译通过的工程的 Debug 目录中，可以看到，已经生成了几个 .gcno 的文件。



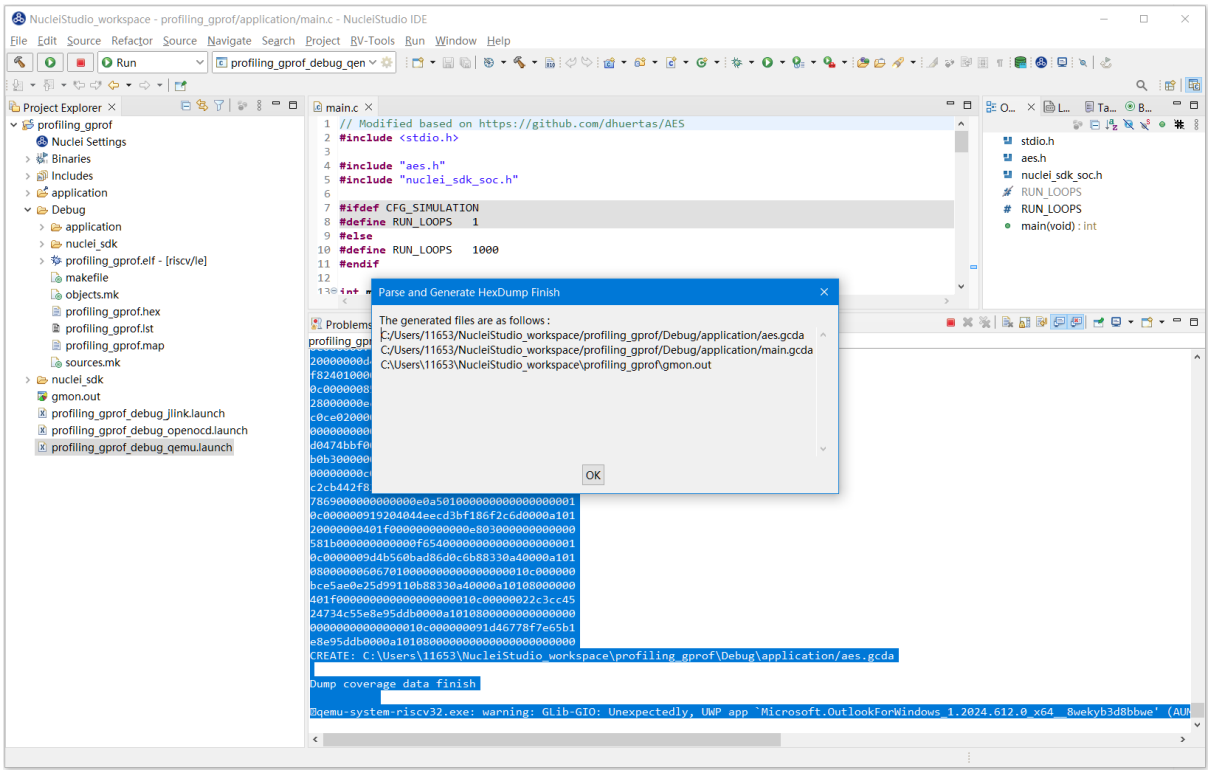
工程编译完后，可以运行或调试工程，我们可以选择在 QEMU 下进行，也可以调试实际的开发板。本例以 QEMU 为例进行运行程序，在 NucleiStudio 的 Console 窗口中可以看到 Profiling 信息输出，如果是在开发板上调试，则是在串口输出中可以找到 Profiling 信息输出。



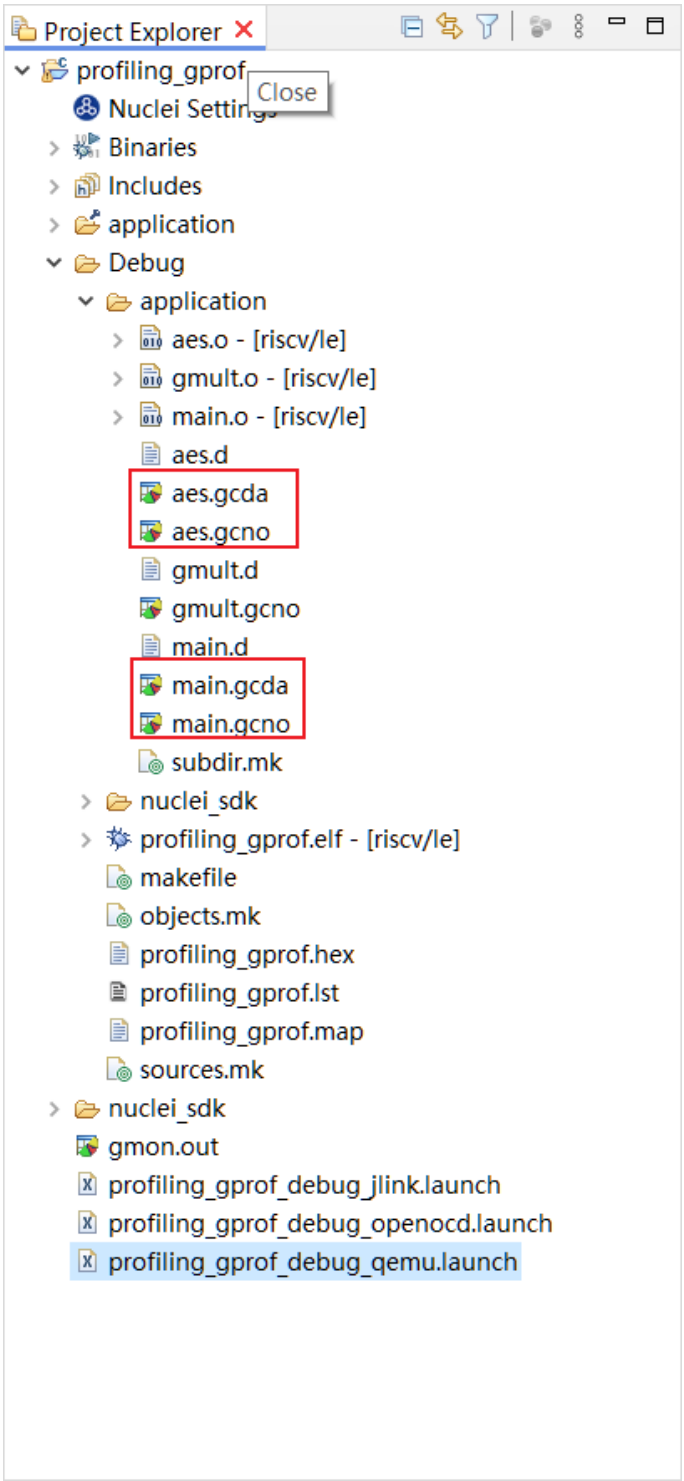
输出的 Profiling 信息需要解析后 NucleiStudio 才可以正确读取，在 Console 框内点击鼠标右键，然后在弹出的菜单中点击 Select All，来选中所有输出，再次击鼠标右键，在弹出菜单中选择 `Parse and Generate HexDump` 菜单。



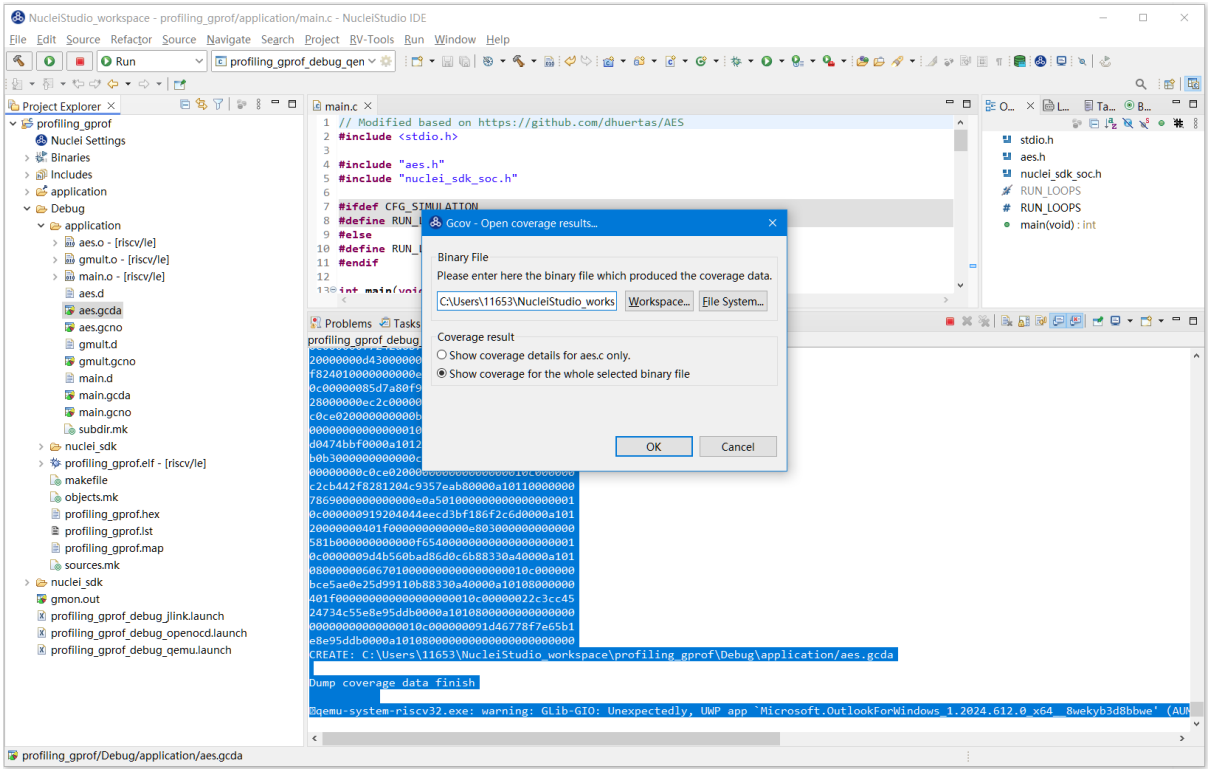
此时 NucleiStudio 会对输出的文件进行分析，并将结果存别分存放在对应的文件中。



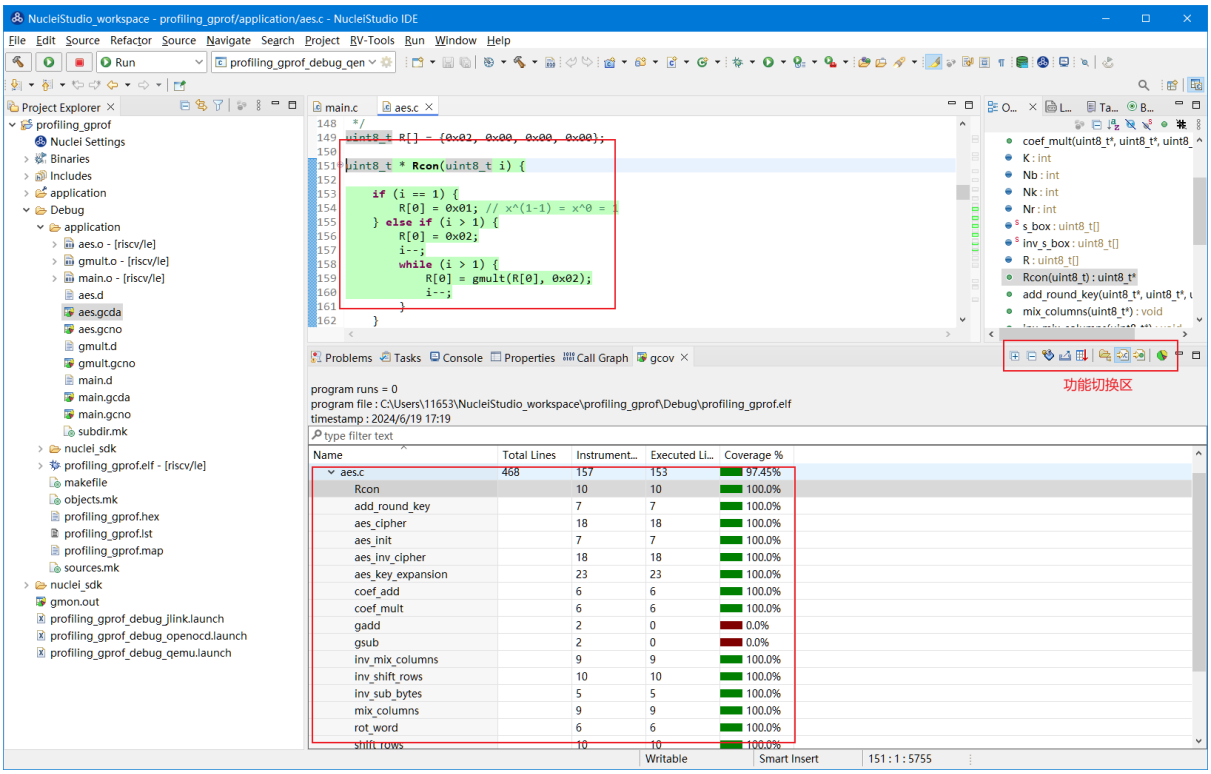
再次查看工程的 Debug 目录，可以看到产生了对应的 .gcda 文件。



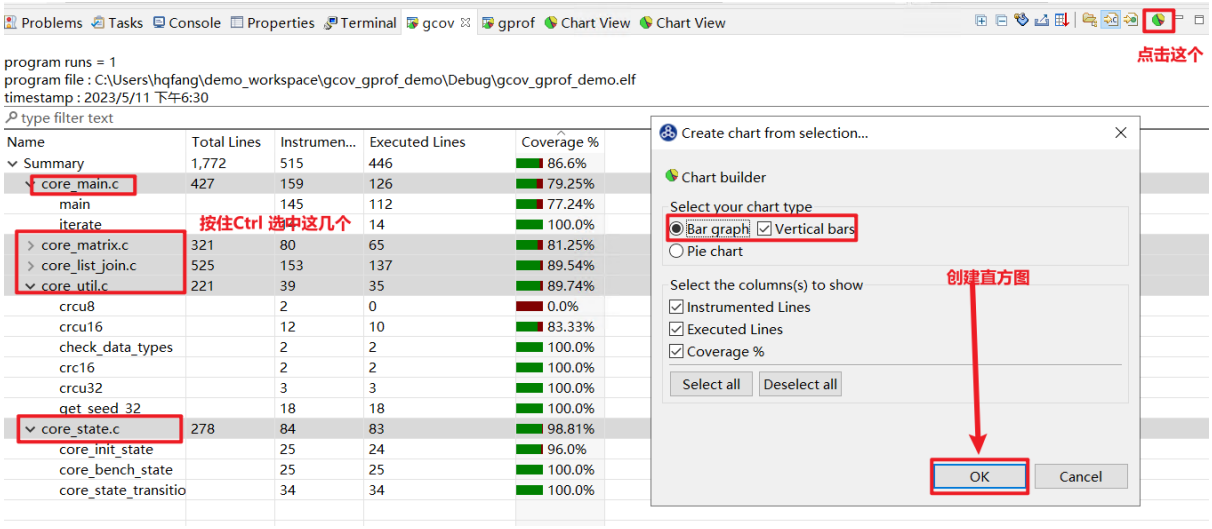
双击 .gcda 文件，打开 Gcov 工具，就可以看到对应用程序的分析结果，在结果中显示了某个文件或某个方法在程序执行过程中是否执行到，以及代码执行复盖比等数据。



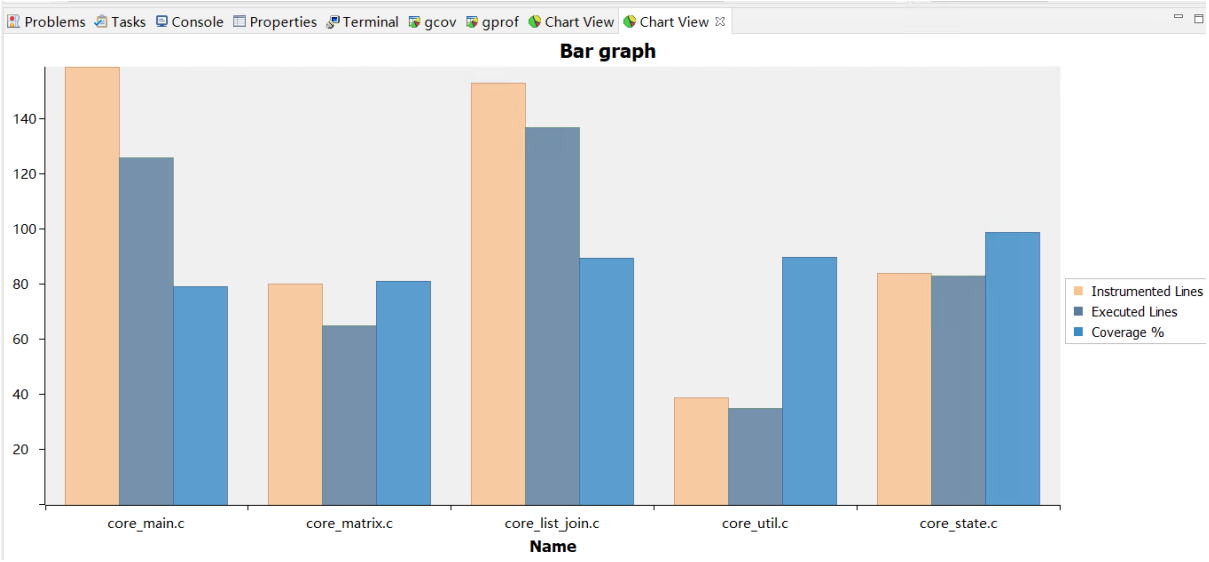
双击 Gcov 中的某一行，NucleiStudio 就会自动打开对应的文，并对文件中的代码着色，绿色表示在程序执行过程中有执行到，红色代表在程序过程中没有被执行到。开发者可以参考 Gcov 的结果，并对代码做出相应的优化。



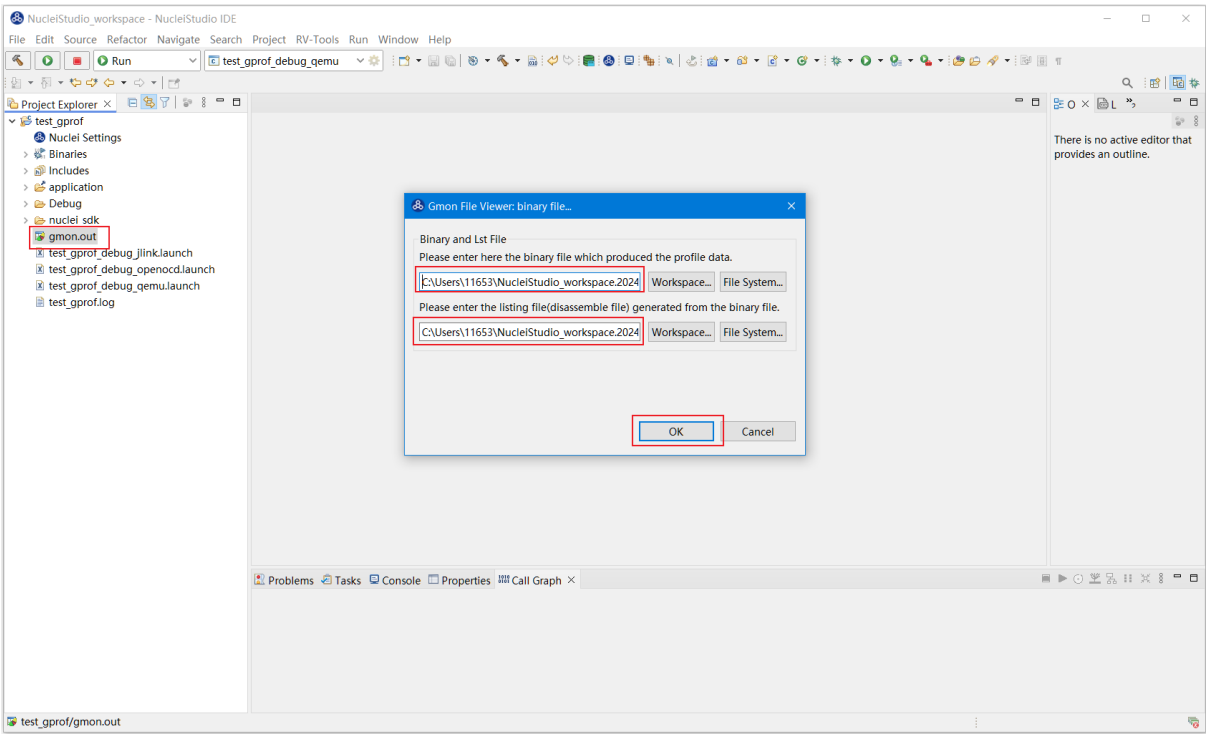
code coverage 也提供了以直方图的方式查看数据，选中想要查看的数据项，点击菜单中的直方图菜单，并按需求配置。



就可以在 Nuclei Studio 中查看 code coverage 直方图信息了。

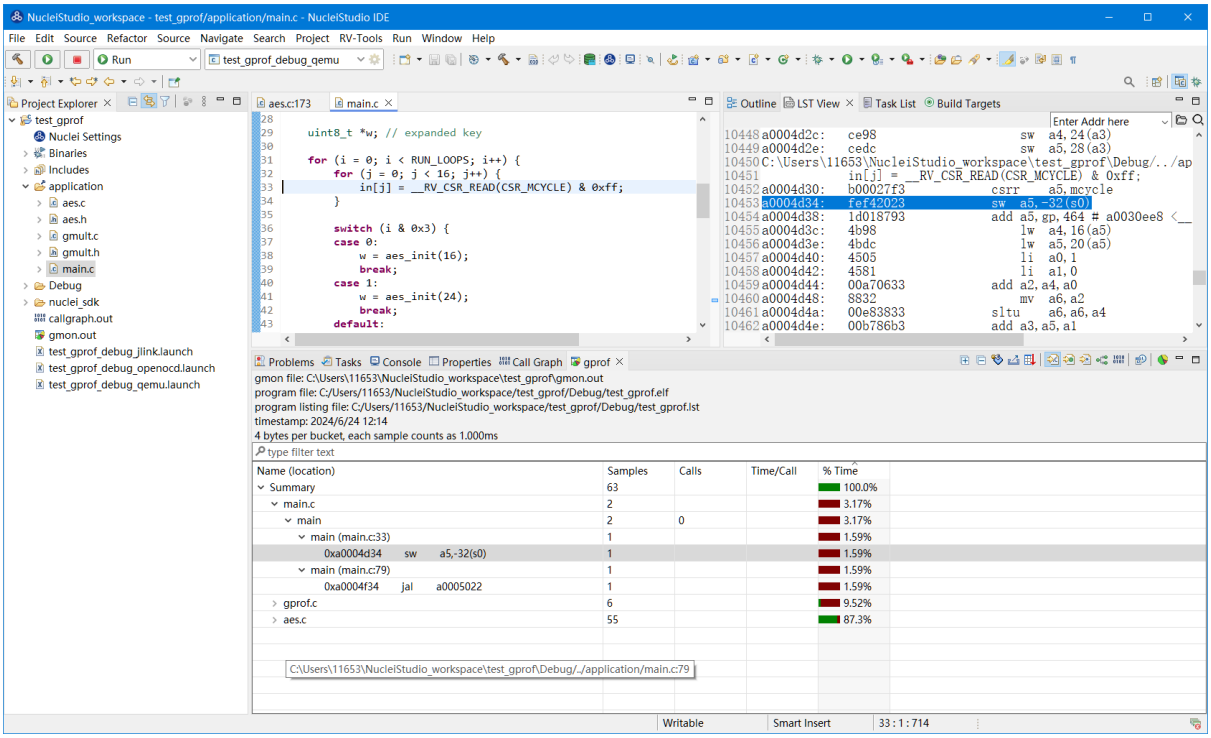


双击 gmon.out 文件，弹出一个文件选择框，提示填写与选中与 gmon.out 文件相关的 elf 文件和 *.lst 文件，默认会根据当 gmon.out ， 自动填入对应的工程内的 elf 文件和 *.lst 文件，点击 OK 按钮。

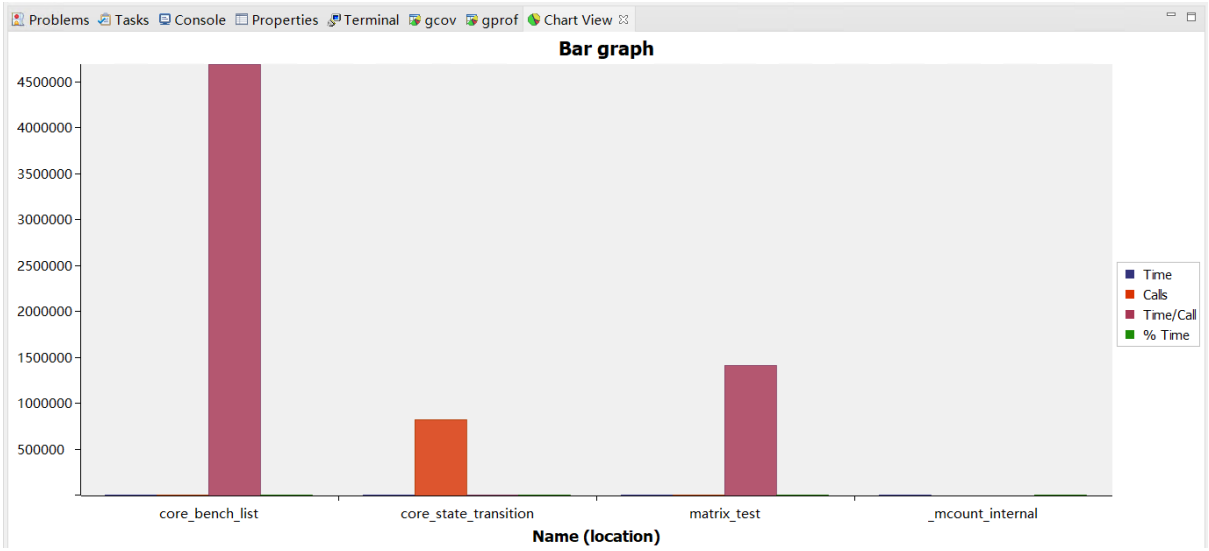


Gprof 工具会启动，就可以看到对应用程序的分析结果，显示了文件、方法的调用关系等。

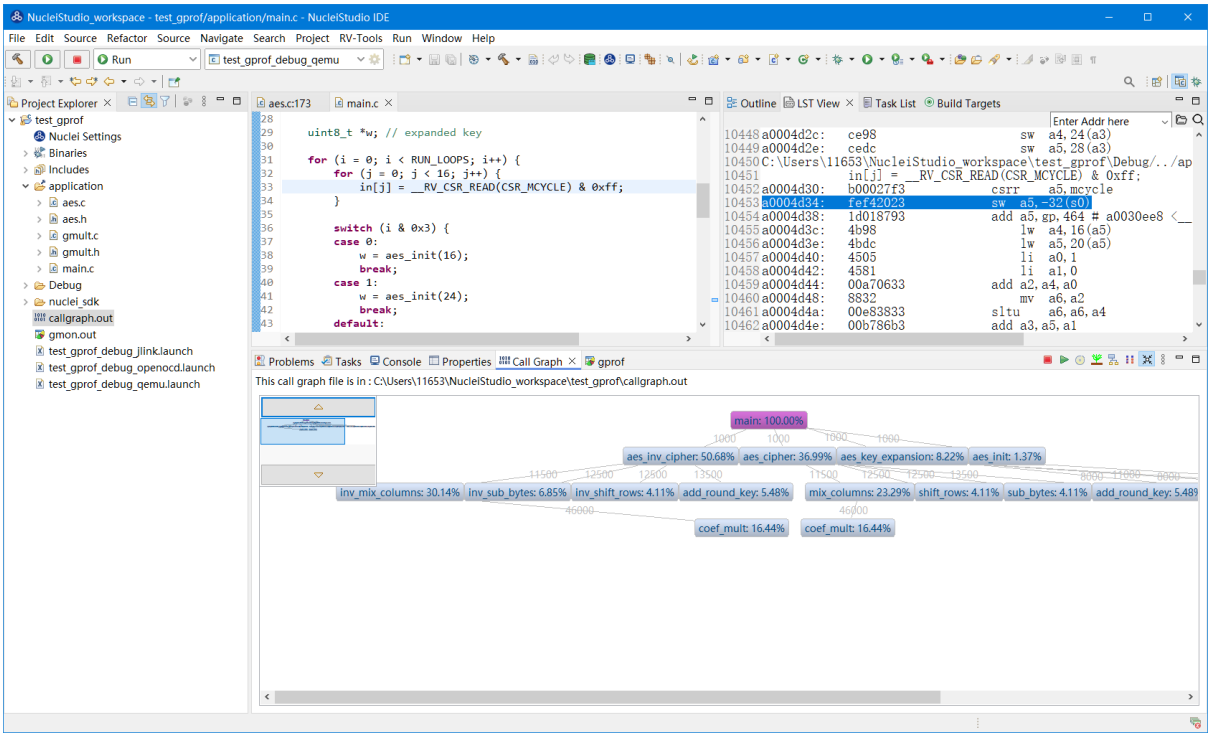
双击 Gprof 中的某一行，NucleiStudio 就会自动打开对应的源文件并定位到对应的行，同时打开 LST View 工具，并根据 addr 定位那那一行，实现 Gprof、源代码、反汇编码的联系，帮用户快速了解程序结构及调用关系。



同样在 Nuclei Studio 中，可以查看 profiling 数据的直方图信息。



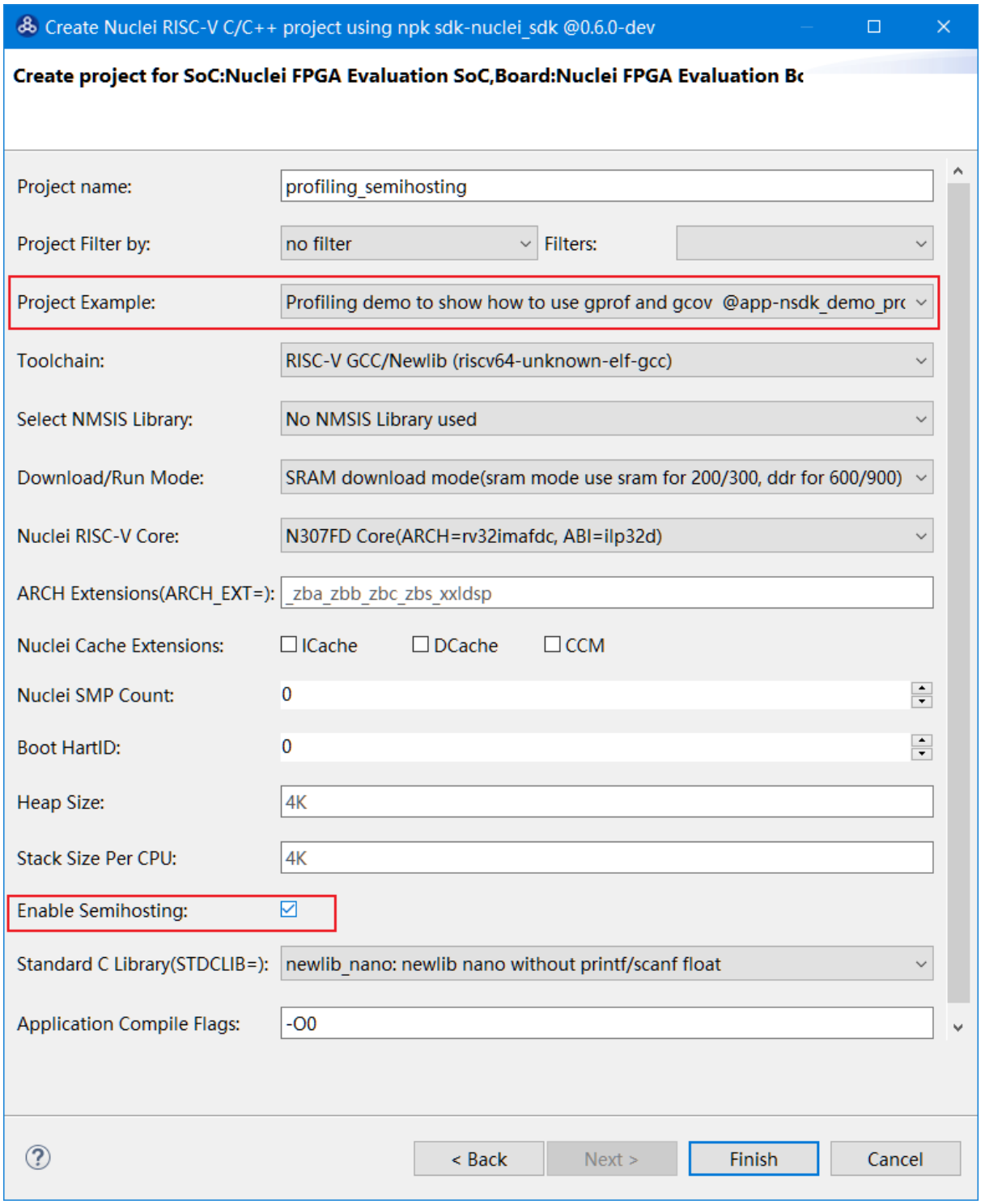
打开 Gprof 的同时，NucleiStudio 会根据 gmon.out 文件解析出程序的 Call Graph 并生成 `callgraph.out` 文件。双击 `callgraph.out` 文件，也可以点击 Gprof 工具的菜单栏中 `Open Call Graph View` 按钮，来启动 Call Graph 工具。关于 Call Graph 的具体使用，可以参考关于 *Call Graph* 功能 (page 189)。



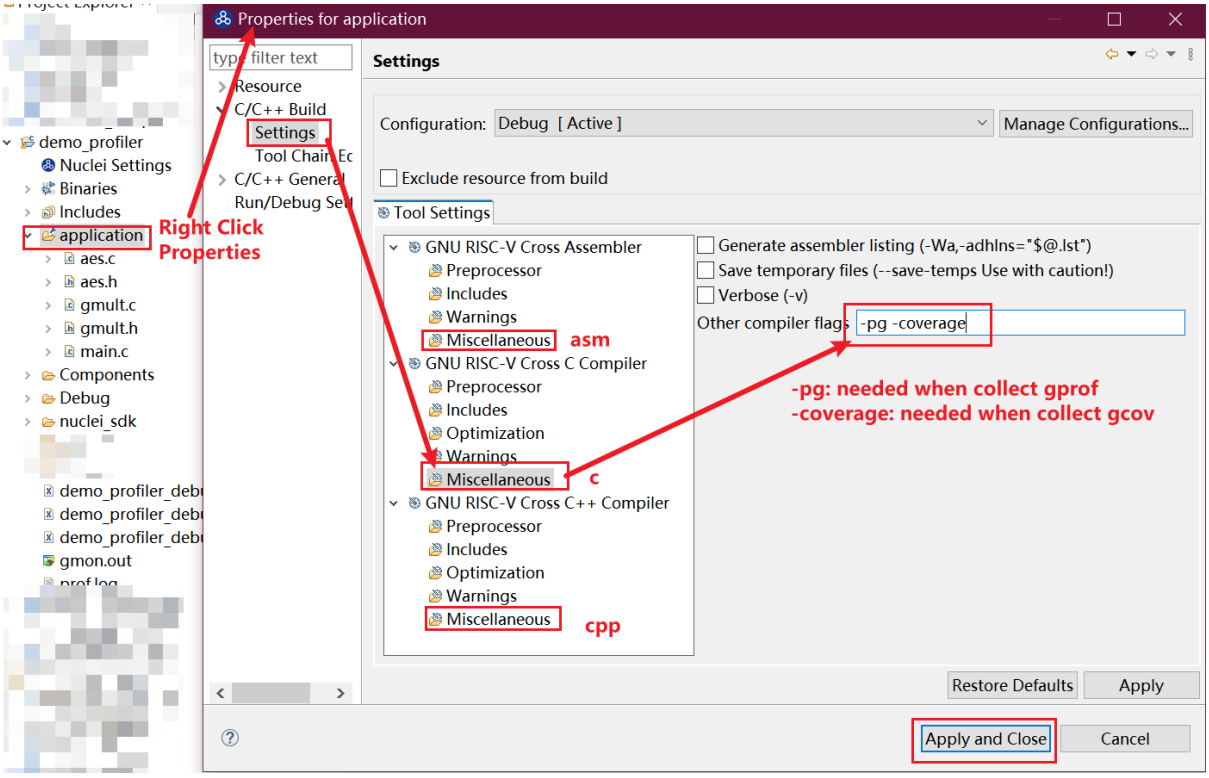
通过 **Semihosting** 使用

NucleiSudio 安装了 nuclei_sdk 0.6.0 后，可以创建一个 `Profiling demo to show how to use gprof and gcov` 的测试工程，此时需要选中 `Enable Semihosting`。关于 `Profiling demo to show how to use gprof and gcov` 测试工程，可参考 demo_profiling[19] 。

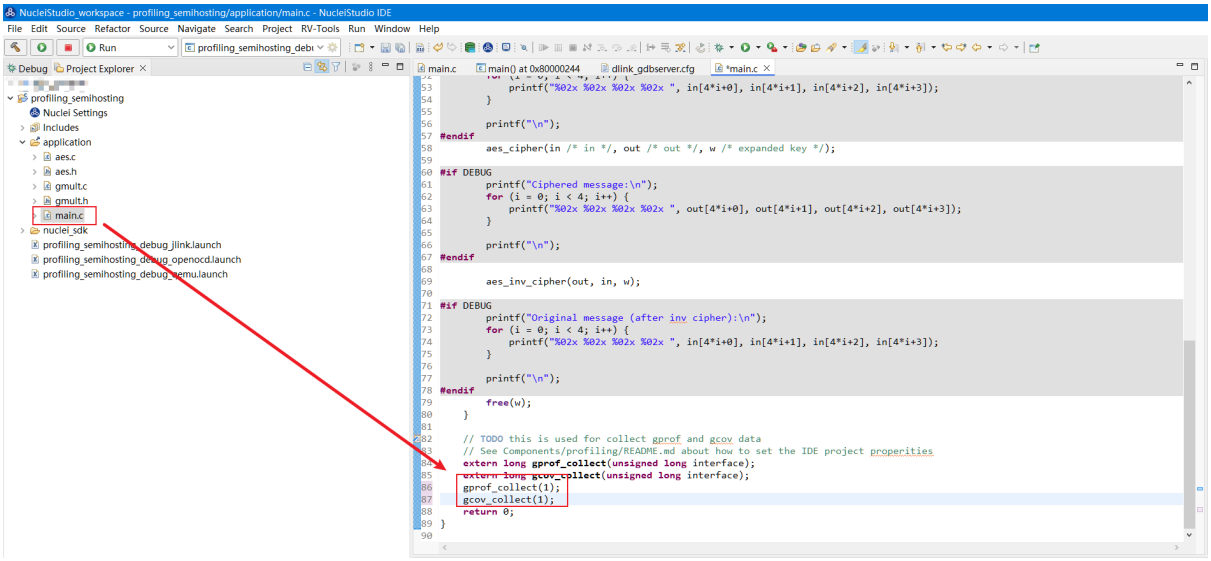
[19] https://doc.nucleisys.com/nuclei_sdk/design/app.html#demo-profiling



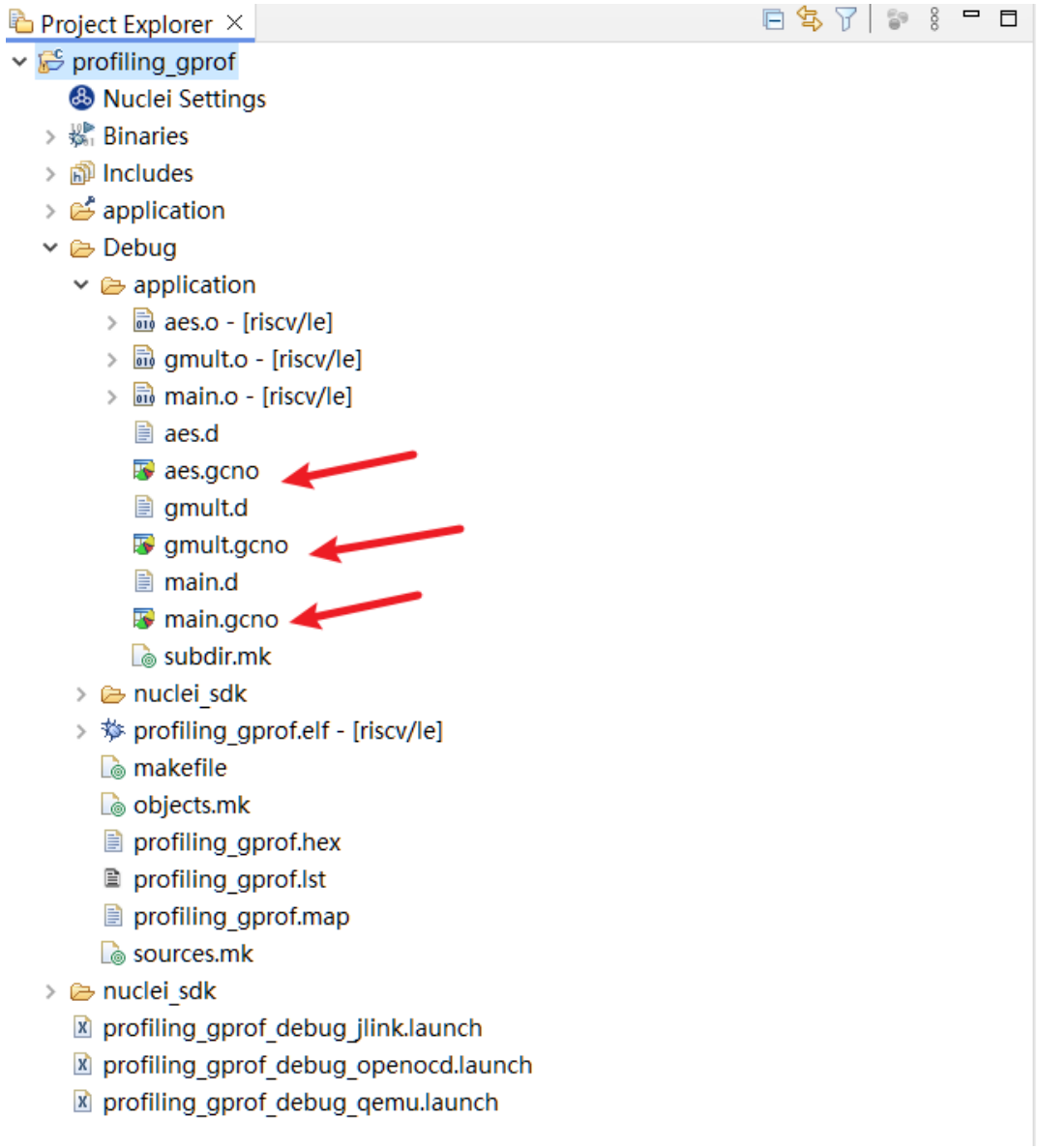
工程创建后，需要对想要进行代码分析的文件或者文件夹设置一个 `-pg --coverage` 的编译选项，然后编译工程。



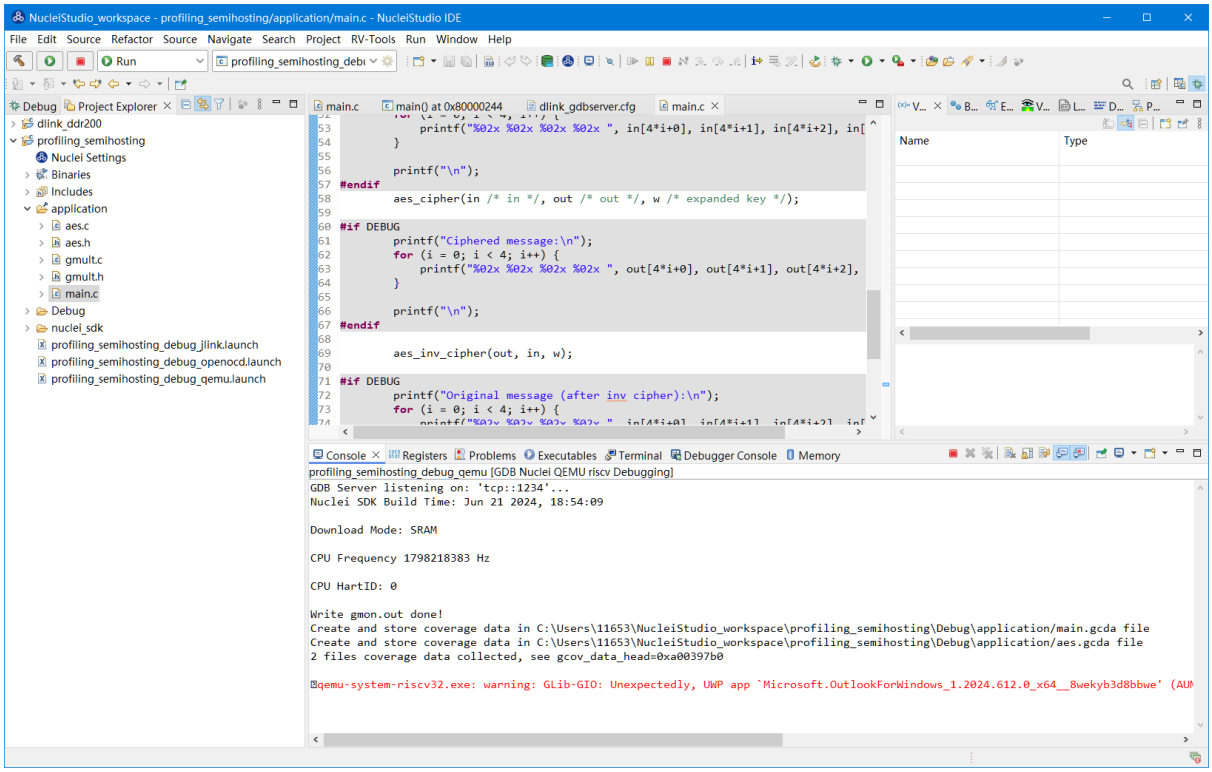
同时，需要修改程序中 `gprof_collect(2);` 为 `gprof_collect(1);` 、`gcov_collect(2);` 为 `gcov_collect(1);` （测试工程中在 main 函数的最后），则在运行过程中，将会通过 Semihosting 将结果输出为文件。



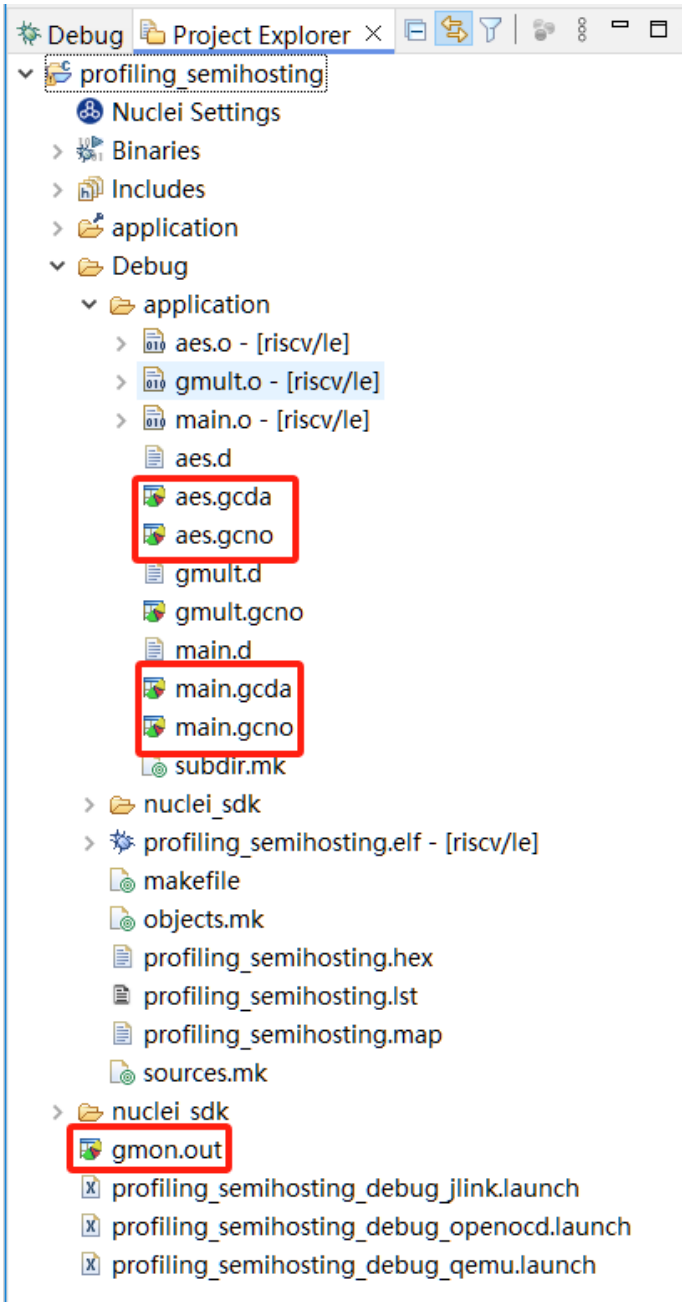
开始编译工程，在编译通过的工程的 Debug 目录中，可以看到，已经生成了几个 `.gcno` 的文件。



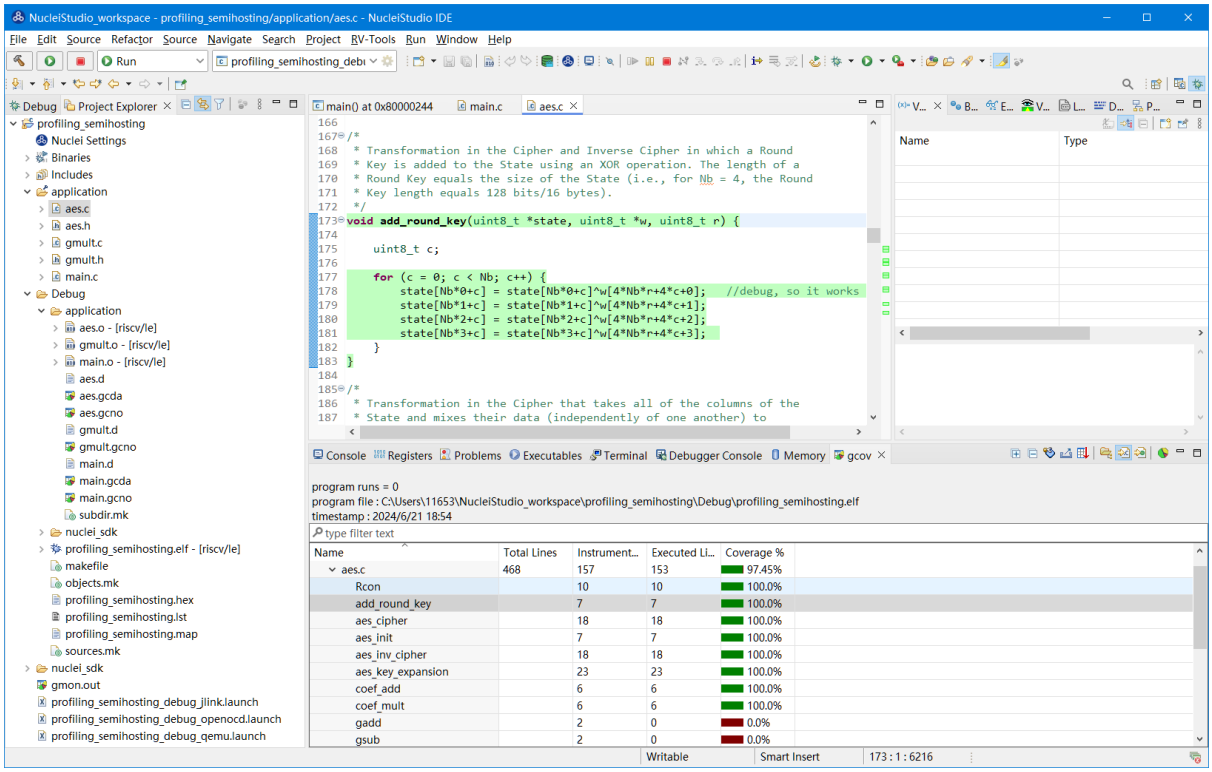
工程编译完成后，可以运行或调试工程，我们可以选择在 QEMU 下进行，也可以调试实际的开发板。



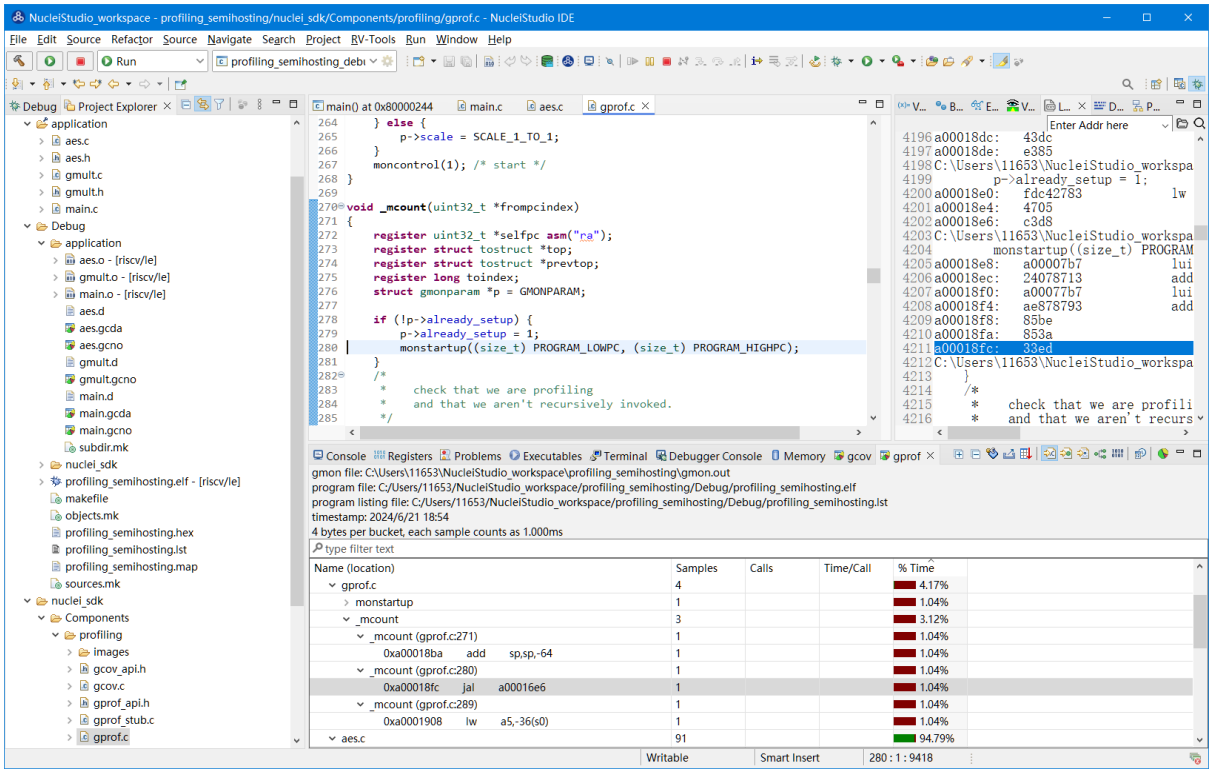
本例以 QEMU 为例进行运行程序，程序运行结束后，刷新工程，可以看到工程下多出了几个文件，`*.gcda` 文件以及 `*.out` 文件。至此，后面查看结果与上面类似。



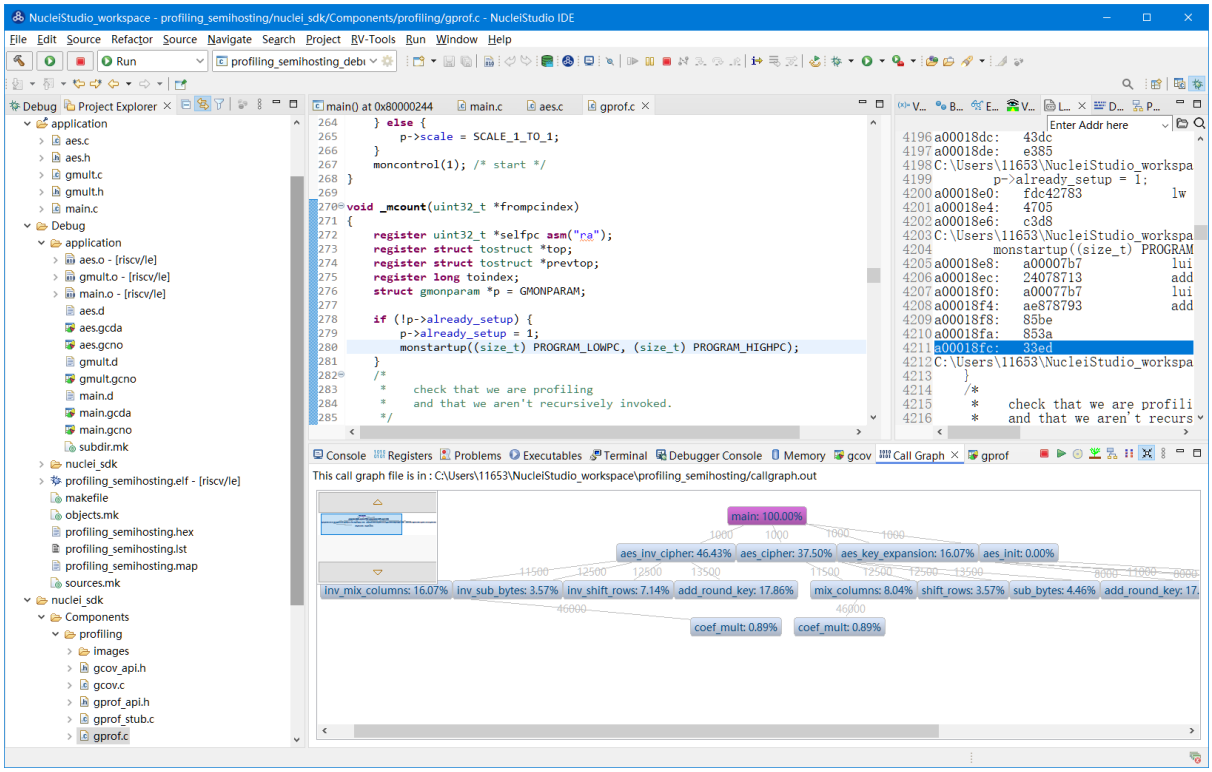
在 Nuclei Studio 中通过 gcov 工具查看应用程序的 Code Coverage 信息。



在 Nuclei Studio 中通过 gprof 工具查看应用程序的 Profiling 信息。



在 Nuclei Studio 中通过 Call Graph 查看调用关系信息。



2.10.4 Trace 功能的使用

Trace 技术是一种强大的调试工具，它能够帮助开发人员跟踪和记录程序执行过程中的关键信息，从而有效地诊断问题、优化性能和提升系统的稳定性。

Nuclei Studio 集成了 Trace 工具，结合相对应的硬件和 Nuclei OpenOCD，用户在对工程进行 Debug 时，也可查看到 Trace 日志，并结合源码时行问题排查。

Note: 在芯来科技视频号中有如何在 **Nuclei Studio** 中使用 **Trace** 功能的视频，您可以在微信中搜索芯来科技视频号点击查看相关内容。

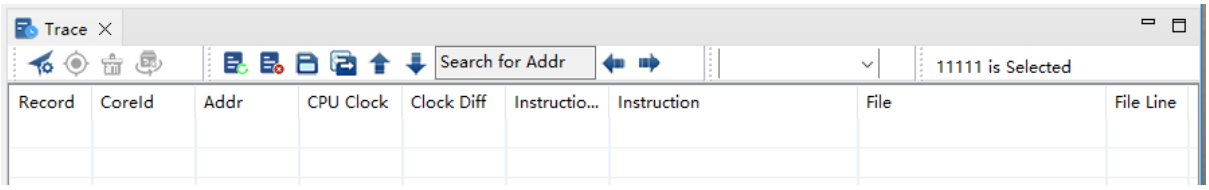
Note: 关于 OpenOCD 的 Nuclei ETrace 的一些命令，请参加 OpenOCD 下的 openocd.pdf 手册。

在使用过程，如有问题，可以查看 https://github.com/Nuclei-Software/nuclei-studio 相关内容，也可以向我们提交相关 issue。

Trace 界面介绍

Trace View

在 Nuclei Studio 中，通过菜单 `Window->Show View->Other` 打开 View 管理器，在里面找到 RV Trace->Trace 菜单，打击打开 Trace 菜单。



Trace 的视图分两部分，上面为 Trace 工具栏，下面是 Trace 记录表格。Trace 工具栏的介绍和功能分别如下：

- **Trace setting**

trace 的配置信息，在这里配置 Trace ATB2AXI Config Addr、Trace Buffer Base Addr、Trace Buffer Size in Bytes、Trace Wrap

- **Start trace/stop trace**

设置开始/停止 trace 操作。

- **Trace clear**

清空硬件上的所有的 trace 设置。

- **Dump trace file**

从硬件上 Dump trace 文件。

- **Reload trace file**

本地重新加载 trace 记录表内容。

- **Clear viewer**

清空 trace 记录表内容，以及 Trace Decode 相关的配置，如 HartID 和 Thread 的关系等。

- **Save trace log**

将 trace 记录表保存为 csv 表格。

- **Toggle instruction stepping**

当选种某条记录时，可以打开并定位到该条记录所对应的源码和反汇编码。

- **step into previous line**

当选种某条记录时，跳转到该条记录的上一条记录，并定位到所对应的源码和反汇编码。

- **step into next line**

当选种某条记录时，跳转到该条记录的下一条记录，并定位到所对应的源码和汇编码。

- **Search for Addr**

搜索框，可以通过 Addr 搜索到对应的那一行 trace 记录。

- **search backward**

搜索结果的记录是多条时，可以查看上一条搜索结果。

- **search forward**

搜索结果的记录是多条时，可以查看下一条搜索结果。

- **Page**

多页的翻页，trace 如果条数很多时，为了方便查看，会采用多页显示。

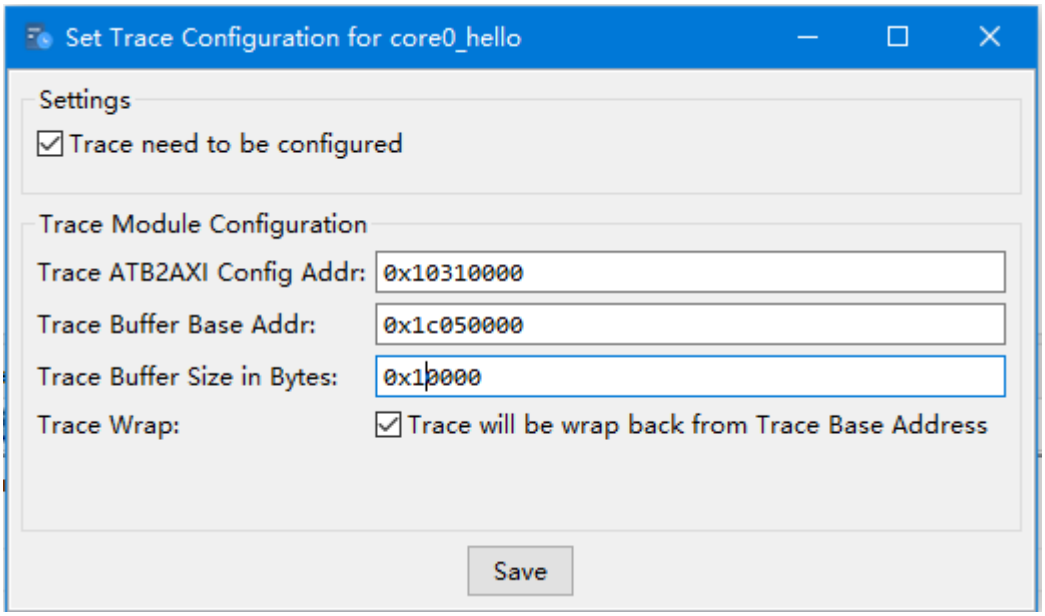
Trace 记录表格，是 Nuclei Studio 将 dump 到的 trace 文件进行解密之后，生成的记录进行展示，并且当用户点击某条记录时，会自动定位到对应的源代码和反汇编代码的行数。

- **Record**：记录 id
- **CoreId**： Coreid，主要是在多核时可以用于区分不同的 Core
- **Addr**：指令地址
- **CPU Clock**：时钟 Cycle 计数
- **Clock Diff**：时钟 Cycle 差
- **Instruction Code**：十六进制表示的指令码
- **Instruction**：指令码

- **File**：指令码对应的源码所在的文件
- **File Line**：指令码对应的源码所在的文件的行数

Trace Configuration

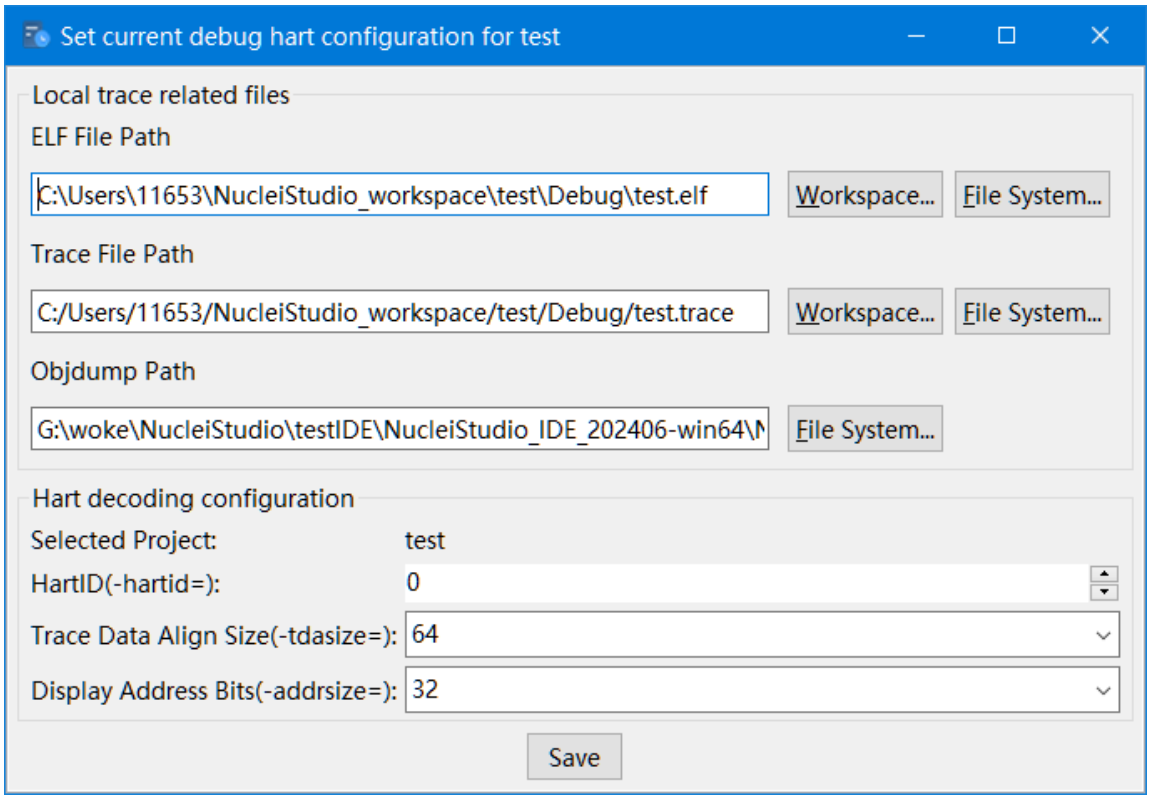
用户可以在这里配置 Trace 的 Trace ATB2AXI Config Addr、Trace Buffer Base Addr、Trace Buffer Size in Bytes、Trace Wrap。具体的信息，根据不同的硬件而不同。



- **Trace need to be configured:** 如果需要配置 Trace 模块就勾选，如果其他地方已经配置过了，就千万不要勾选了，例如多核 SMP/AMP 的情况下，SoC 上只有一个 Trace 模块，假设其中一个核心已经勾选配置了，其他的核心就不能勾选了，或者是配置是在 C 代码中或者其他地方做了，也千万不要勾选。
- **Trace ATB2AXI Config Addr**： ATB2AXI 模块控制器的基地址。
- **Trace Buffer Base Addr**：存放 trace 记录的开始地址，例如：针对某个 SoC, 举例如下在 flashxip 模式，使用 ilm（0x1c000000）作为缓存 buffer；在 sramxip 模式，使用 dlm（0x08010000）作为缓存 buffer。
- **Trace Buffer Size in Bytes**：存放 trace 记录的 Buffer 大小，单位为字节。
- **Trace Wrap**：是否允许自动复盖，允许则在 Buffer 满时，将再次从头开始覆盖记录。

Trace Decoder Configuration

Set Current Debug hart Configuration 弹框中，用户可以自定义 trace decoder 的参数，具体如下。



- **ELF File Path**： trace 生产时执行的 elf 文件的地址。
- **Trace File Path**： 需要解析的 trace 文件的地址。
- **Objdump Path**： trace decode 过程中，需要用到 objdump 工具，所以这里需要指定所使用到的 objdump 工具的地址。
- **HartID**： trace decode 时需要指定当前需要查看的 trace 对应的 HartID，单核工程默认 HartID=0。
- **Trace Data Align Size**：跟踪数据对齐大小，一般与硬件的 trace 输出位宽对齐，默认有 8、32、64。
- **Display Address Bits**： trace decode 后显示地址的位数，一般是 32、64、128 位。

Trace 的使用

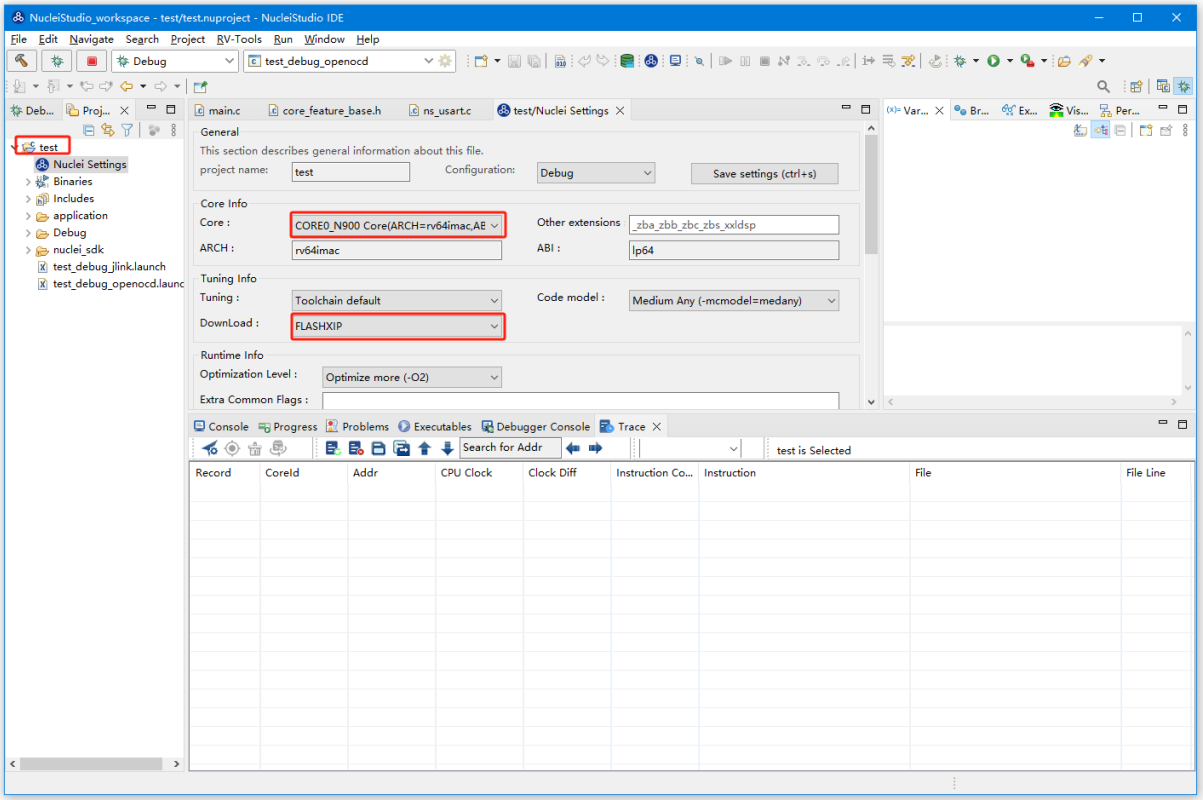
在使用 trace 功能时，必须在工程 Debug 时，通过 Nuclei OpenOCD 或者 Dlink 将 Trace 命令下发到硬件，目前通过 OpenOCD，可以实现在单核、多核 SMP 和多核 AMP 应用下进行 Trace 记录，而 Dlink 仅支持在单核应用下的 trace 记录。

下面我们以 OpenOCD 为例，演示如何使用 Trace 功能。

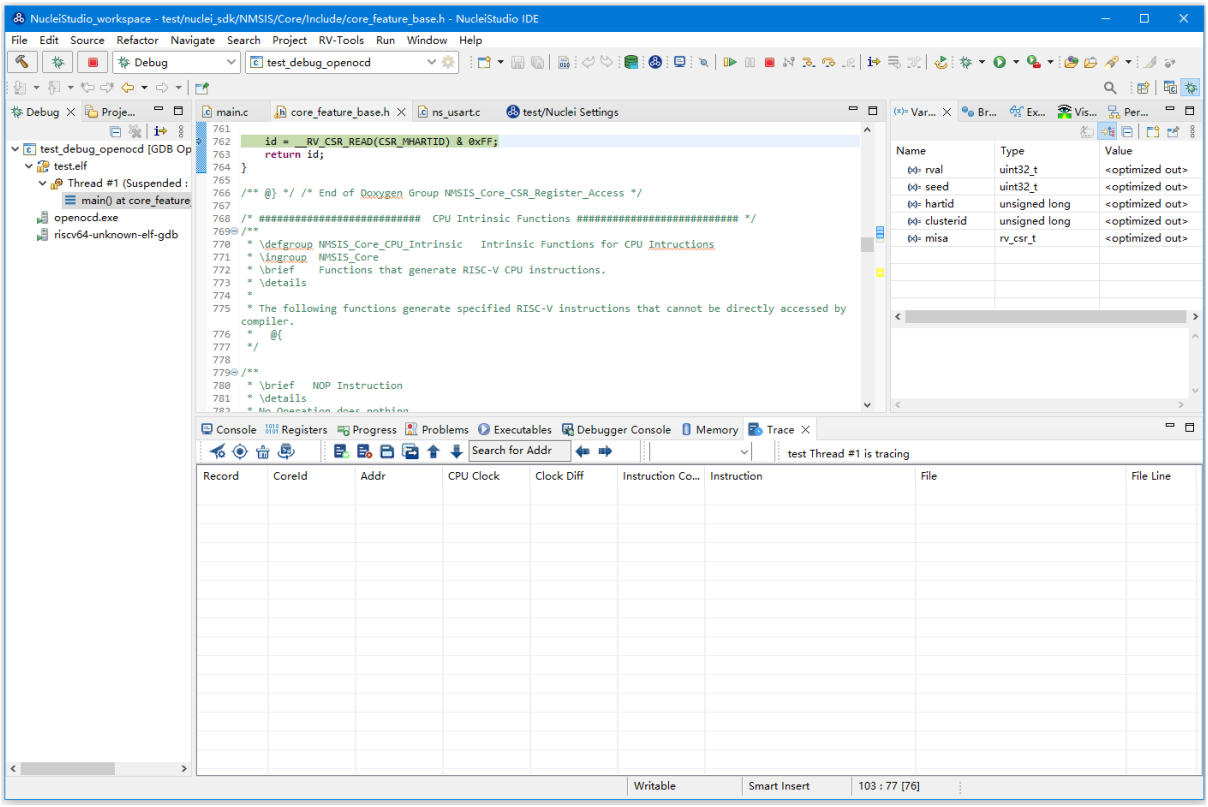
在单核应用中使用 Trace

如果您已获取到芯来授权的 CPU 和相关配套硬件并准备好硬件环境，这里不详细说明。然后创建好对应工程并确保它能在硬件上运行和调试。以下示例是在我们自己构建的一个测试环境上的流程举例说明。

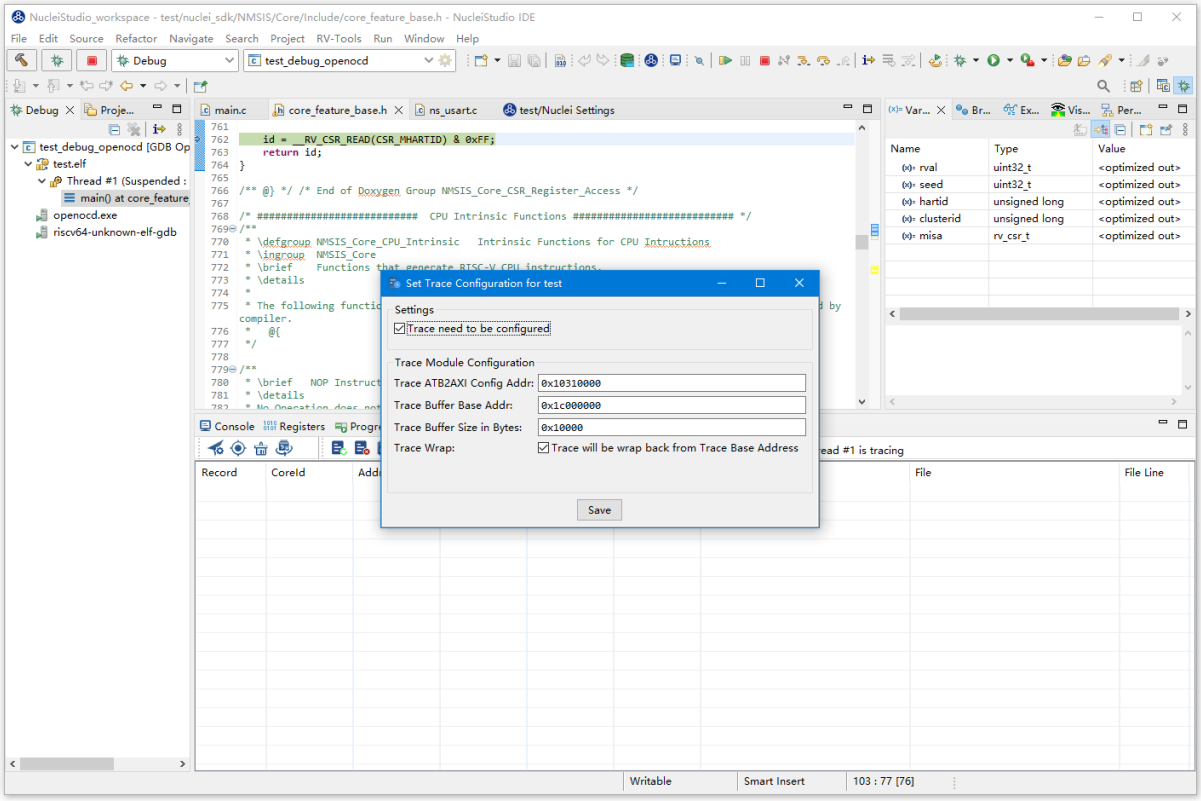
我们在这里创建了一个 N900 的单核应用 helloworld，并让它跑在 FLASHXIP 模式下。



我们可以记录整个应用运行完的 trace，也可以记录某一段 Debug 断点之间的 trace。进入 Debug 模式后，打开 Trace 视图。

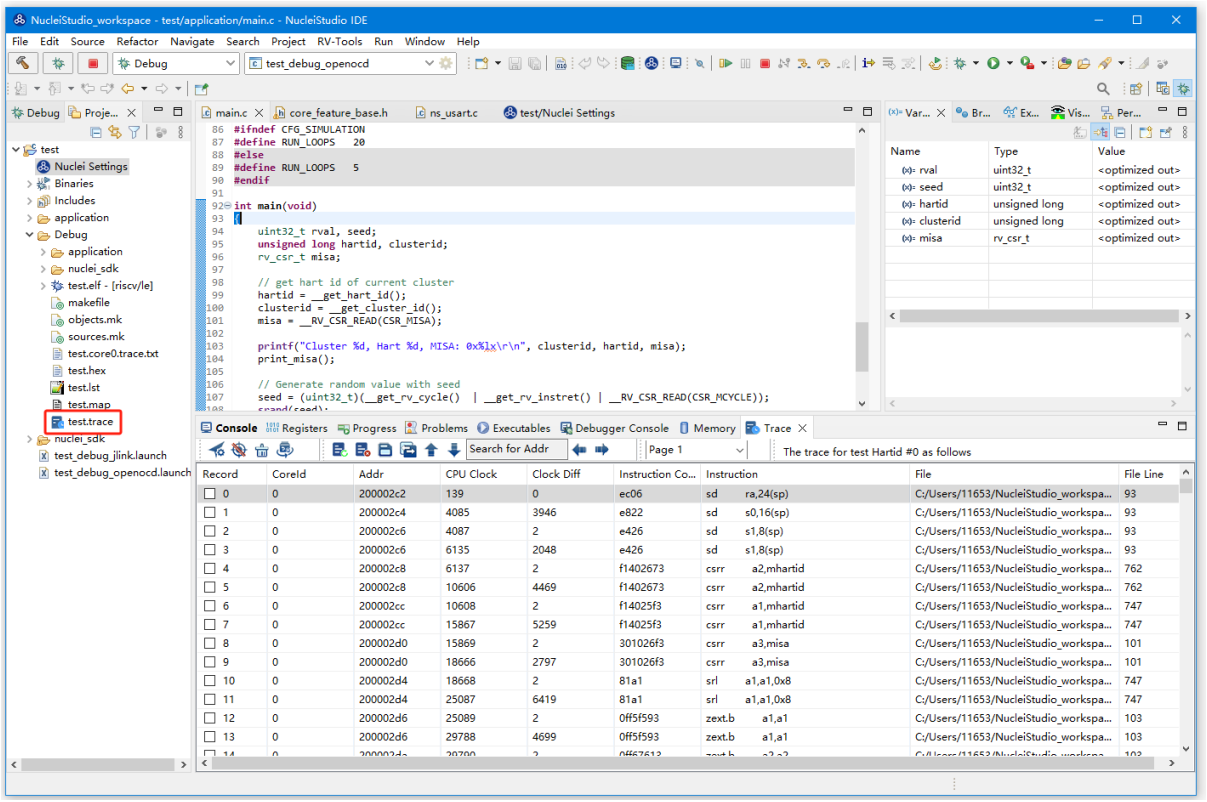


设置 Trace Configuration，设置 trace 配置信息并保存 (Save)，如果不想保存，就关闭窗口。

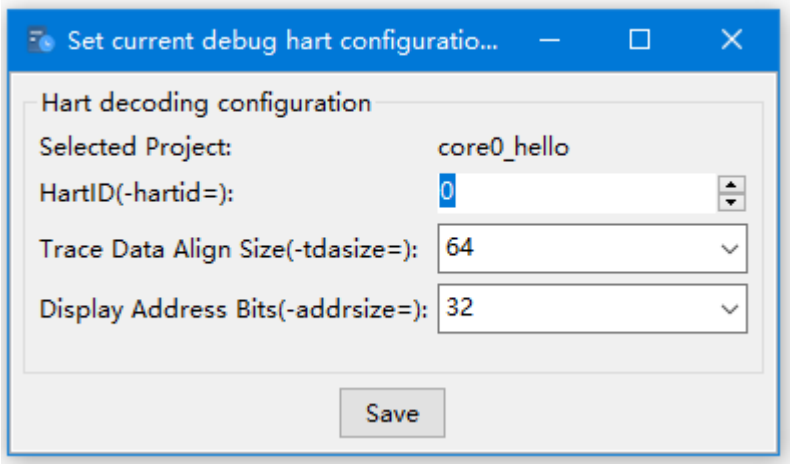


Trace 配置完毕后，可以设置两个断点，一个断点用于 Trace 开始点，一个断点用于 Trace 结束点，在开始点断点停下后就可以点击 `start trace` 按钮，就可以继续 debug 操作 (如单步或者运行等) 了，在结束点断电停下后，就可以点击 `stop trace` 按钮来结束 Trace。上面只是 Start/Stop Trace 的一种使用示例，也可以更灵活一些，请根据自己需要进行使用。当 trace 结束时（多核情况下请确保每个 CPU 的 Trace 都结束了），就可以点 `Dump trace file` 按钮，将 trace 文件从硬件上下载到本地，默认下载的

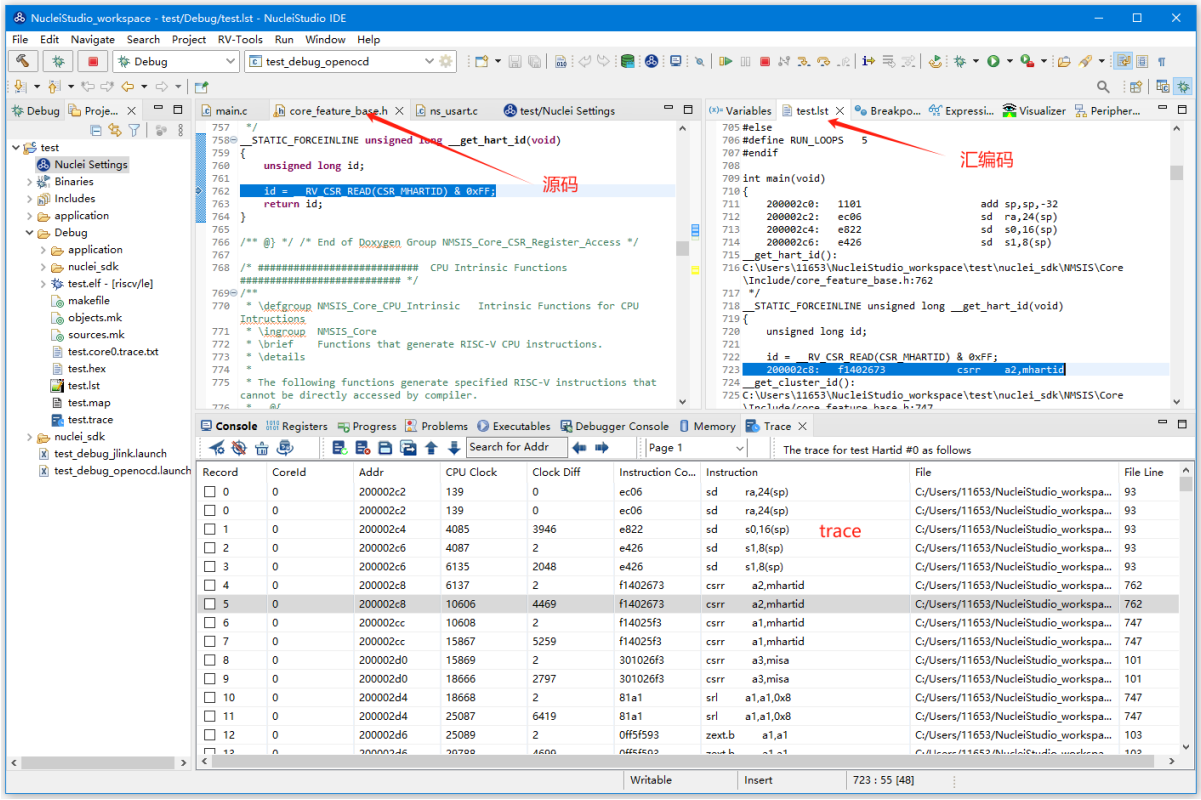
trace 文件存在工程目下的 debug 目录下，有一个 工程名.trace 的文件。



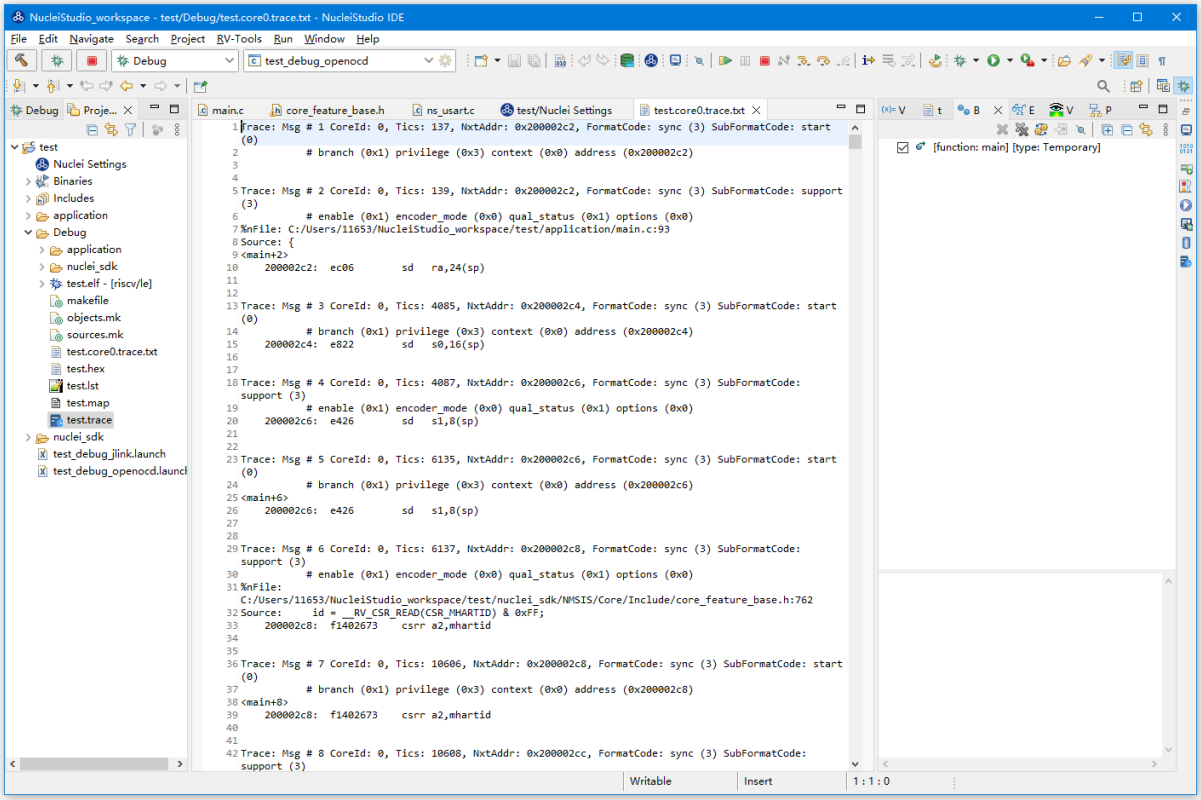
Trace 文件下载完后，Nuclei Studio 会弹出一个 Set current debug hart configuration 框。



在框中填写确正信息（这里的 HartID 指的是对应的 Thread 的 hartid，请不要填错了）并确认，Nuclei Studio 对 trace 文件开始解析，并生成 trace 记录表格。在 trace 记录表格，选中任意一条记录，Nuclei Studio 会自动找到源码和反汇编码，并定位那对应的那一行（因反汇编码与源码在同一个视图中打开，需要用户自己把反汇编码移到另一个视图中）。

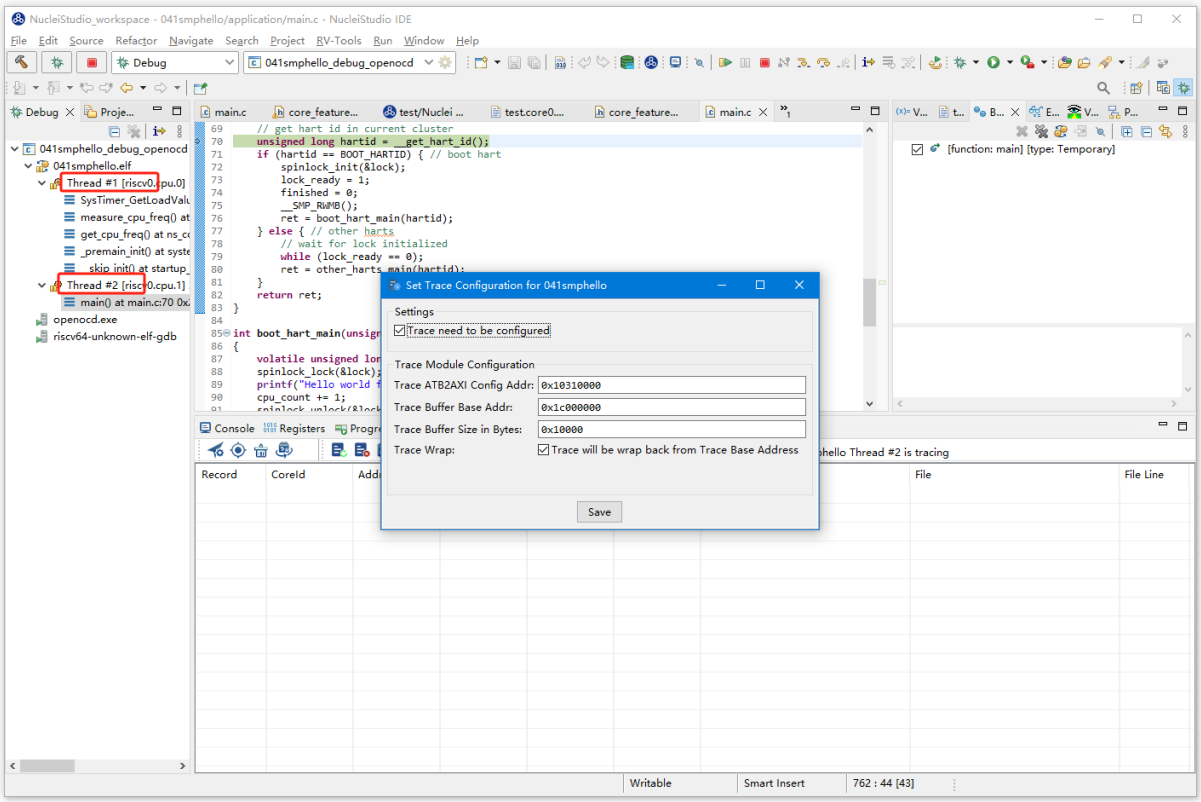


也可以双击 工程名.trace 文件，以文本的方式查看 trace 文件。

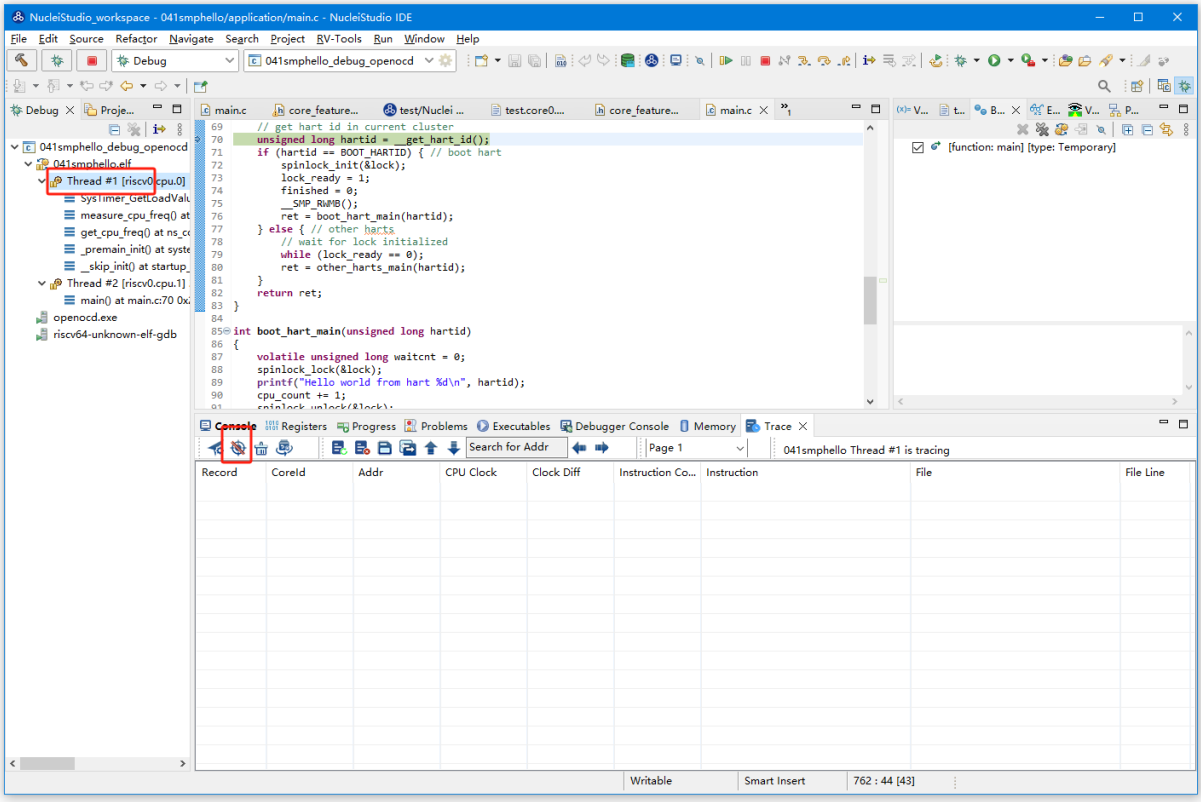


在 SMP 多核应用中使用 Trace

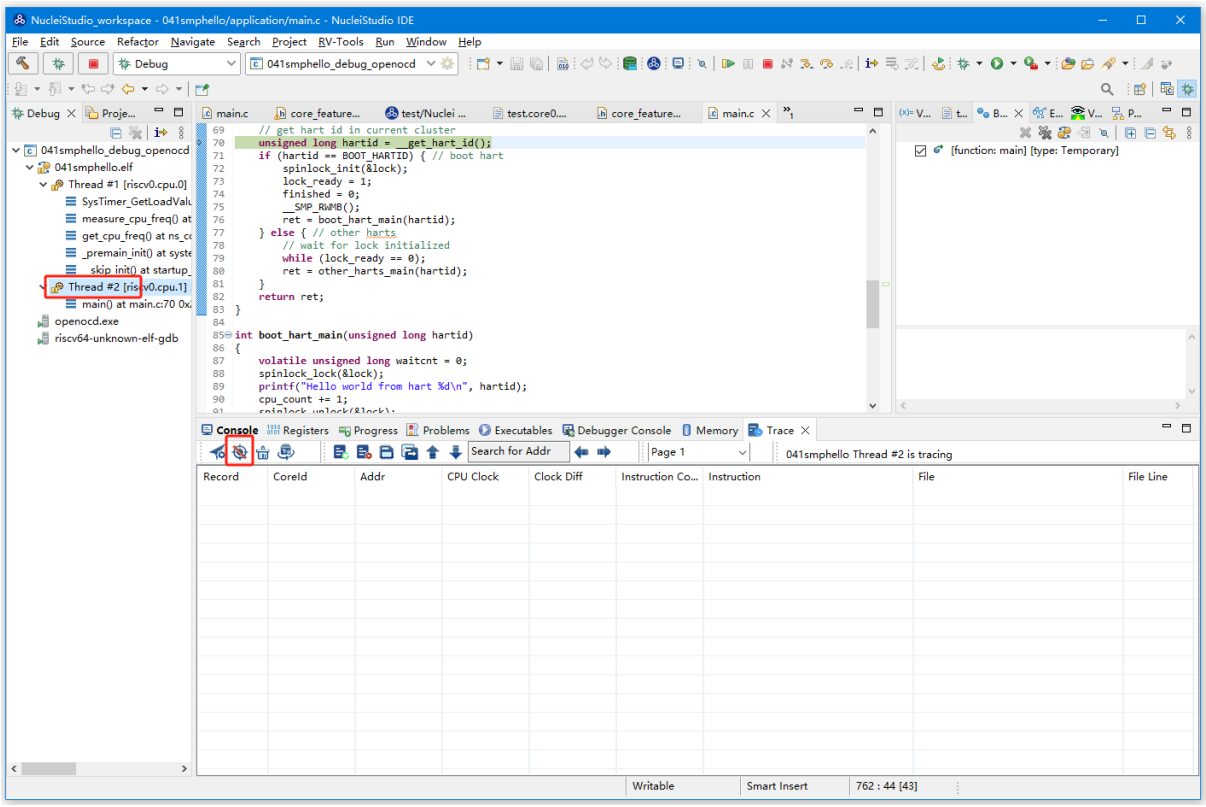
在 SMP 多核应用中使用 trace 与单核大体相似，差别在于 SMP 多核在 Debug 时，不同的 thread 共用一个 Trace Configuration，且需要通过选择不同的 Thread 来对不同的 CPU Hart 核心单独 `start trace/stop trace` 。在 Debug 视图中，点击任意一个 Thread, 然后点击 Trace 工具栏中的 `trace setting` 来设置 Trace Configuration。



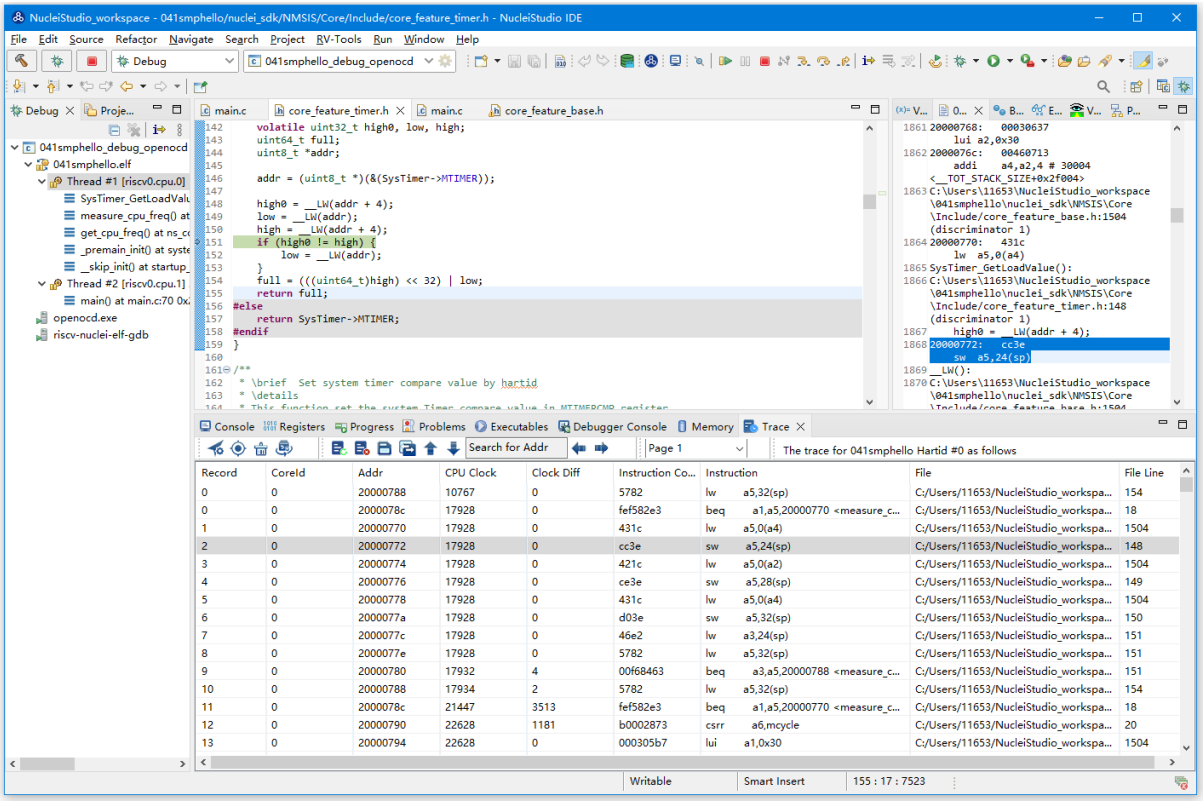
在 Debug 视图中，可以通过点击不同的 Thread, 来切换不同的 Core, 如下图点击 Thread #1 或者 Thread #1 下对应的函数名来选中对应的是 SMP 多核应用中的 Core 0, 可以对 Core 0 开启或者关闭 Trace，在 SMP 多核应用中，只要有一个 Core 在完成 start trace 操作时,Trace Configuration 中的信息就会在硬件中设置好，其他的 core 在 `start trace` 操作时，就不会重复设置 trace Configuration。



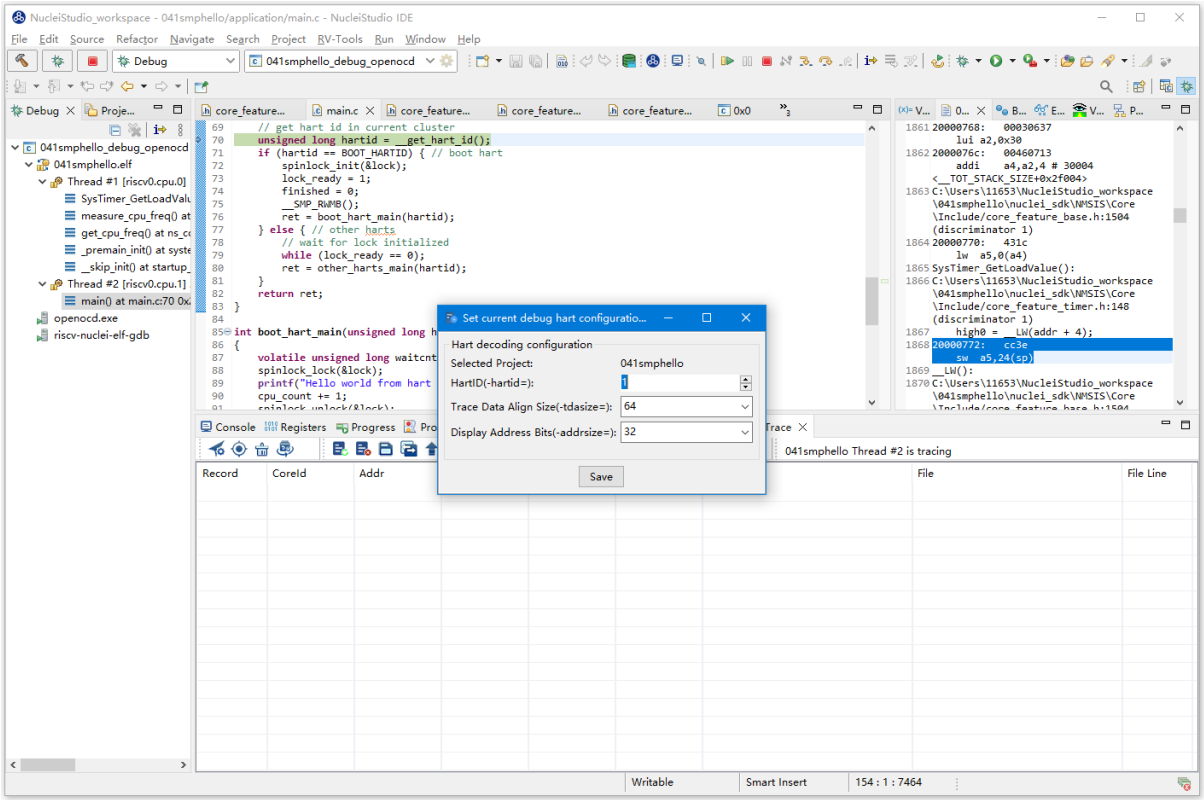
同理，在 Debug 视图中点击 Thread #2 或者 Thread #2 下对应的函数名，来切换到 Core 1 上进行 `start trace/stop trace` 的操作。



在 `dump trace file` 操作时，在 SMP 多核应用中，只有当所有的 Core 都 `stop trace`，才可以执行 `dump trace file` 的指令并成功下载 Trace 文件。Trace 文件的下载，在 SMP 多核应用中，只需要下载一份，在对 trace 文件进行 decode 时，注意设置 Hart ID，就可以解析出不同的 trace 记录表，如下图，当 `HardID=0` 时，就可以查看到 Core 0 对应的 Trace 记录。

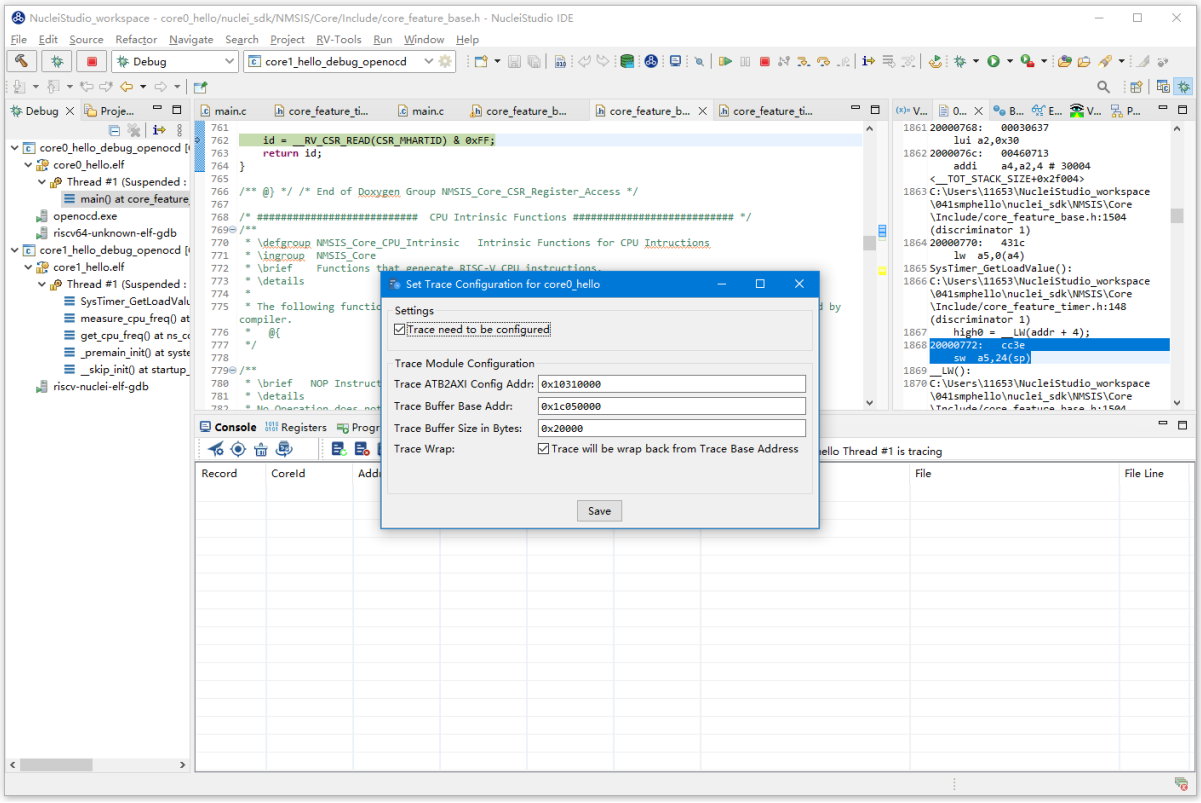


同理当 HardID=1 时，就可以查看到 Core 1 对应的 Trace 记录。

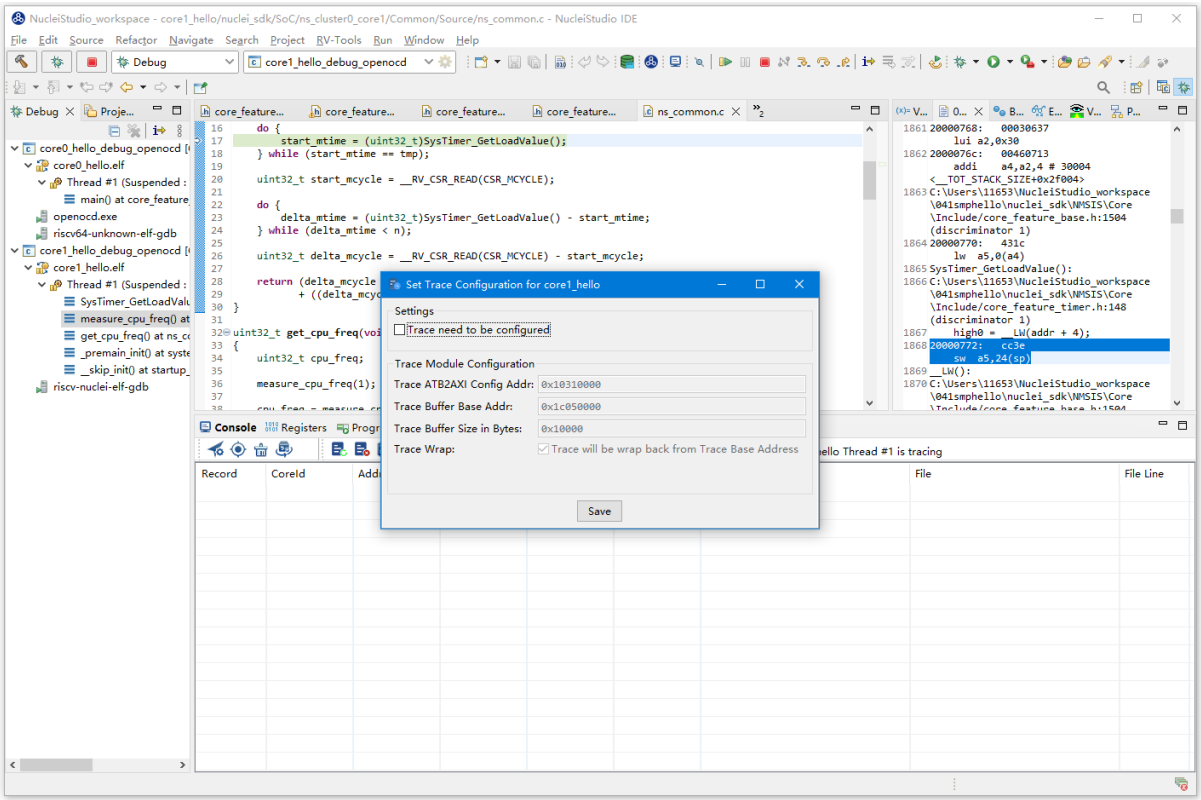


在 AMP 多核应用中使用 Trace

在 AMP 多核应用中使用 trace 也类似，trace 配置也是共享。不同的 thread 共用一个 trace configuration，但可以通过不同的 thread，对不同的核单独 `start trace/stop trace`。如下图，在 Debug 视图，点击 `Thread #1` 或者 `Thread #1` 下的函数名，切换到 AMP 多核应用中 `Core 0`，然后点击 Trace 工具栏中的 `trace setting` 来设置 Core 0 对应的 Trace Configuration。



在 Debug 视图，点击 `Thread #2` 或者 `Thread #2` 下的函数名，切换到 AMP 多核应用中 `Core 1`，然后点击 Trace 工具栏中的 `trace setting` 来设置 `Core 1` 对应的 Trace Configuration，因为在 AMP 多核应用中 trace 配置是共用，所以此处设置需要将 `Trace need to be configured` 的勾去掉，表示可以使用 trace 功能，但不需要有任何设置。

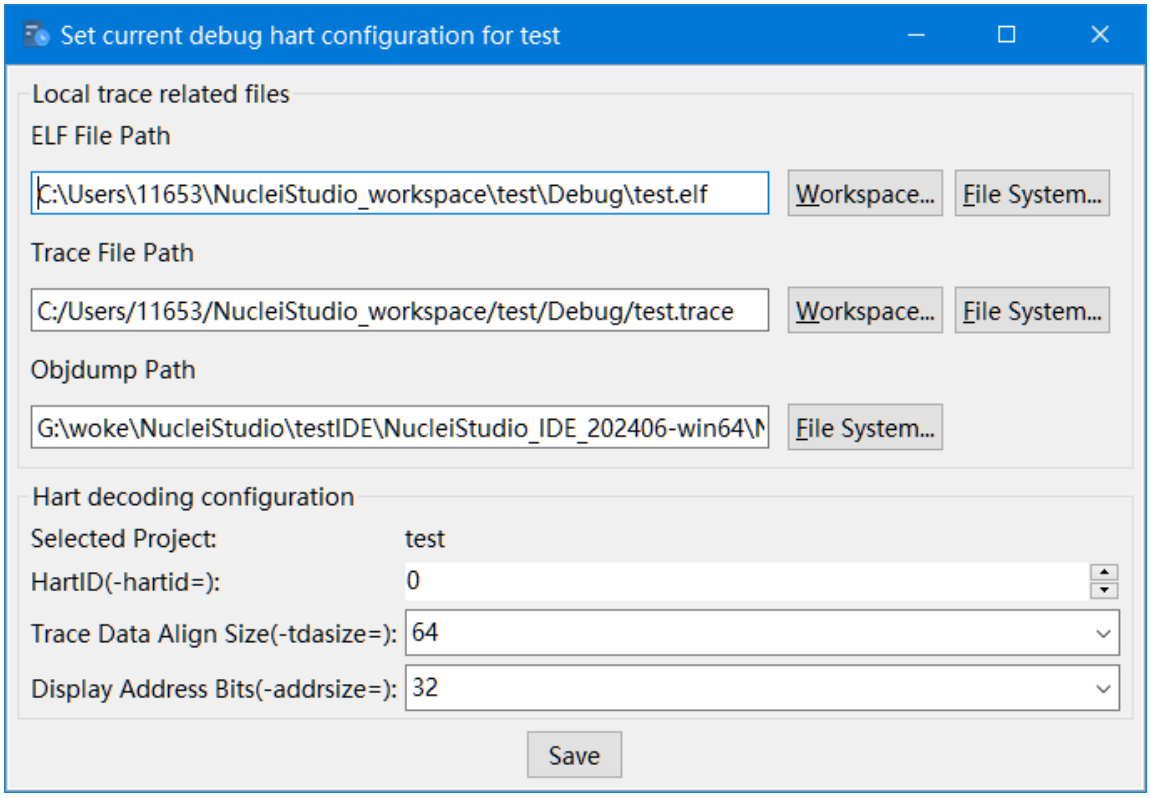


Trace Configuration 设置完成后，同样的通过 Debug 视图的 Thread 来切换不同的 Core，进行 `start trace/stop trace/dump trace file` 操作, 注意，设置了 Trace Configuration 的 Core 需要优先于其它 Core 开始 `start trace`，并将 Trace Configuration 的信息设置好，其他的 Core 才可以正常的 `start trace/stop trace/dump trace file` 操作。

在 `dump trace file` 操作时，在 AMP 多核应用中，请确定所有的 Core 都 `stop trace`，才执行 `dump trace file` 的指令，否则可能在某一下 Core 在 `dump trace file`，其他的 Core 还在记录 trace，最后得到的 Trace 文件并与预期不符。Trace 文件下载，在 AMP 多核应用中，需要每一个工程应用单独 dump 一份 trace 文件，其实 dump 到的 trace 文件内容是一样的，在对 trace 文件进行 decoder 时，同样需要注意设置 `Core Hart ID`，就可以解析出对应的 trace 记录表。其他操作与上文内容中所述类似。

查看脱机 Trace

在某些场景下，用户可能通过命令行或其他方式，得到了一个 trace 文件，这时只需打开 `Set Current Debug hart Configuration`，并按要求配置好参数，即可通过 NucleiStudio 的 trace 工具解析这个 trace 文件了。



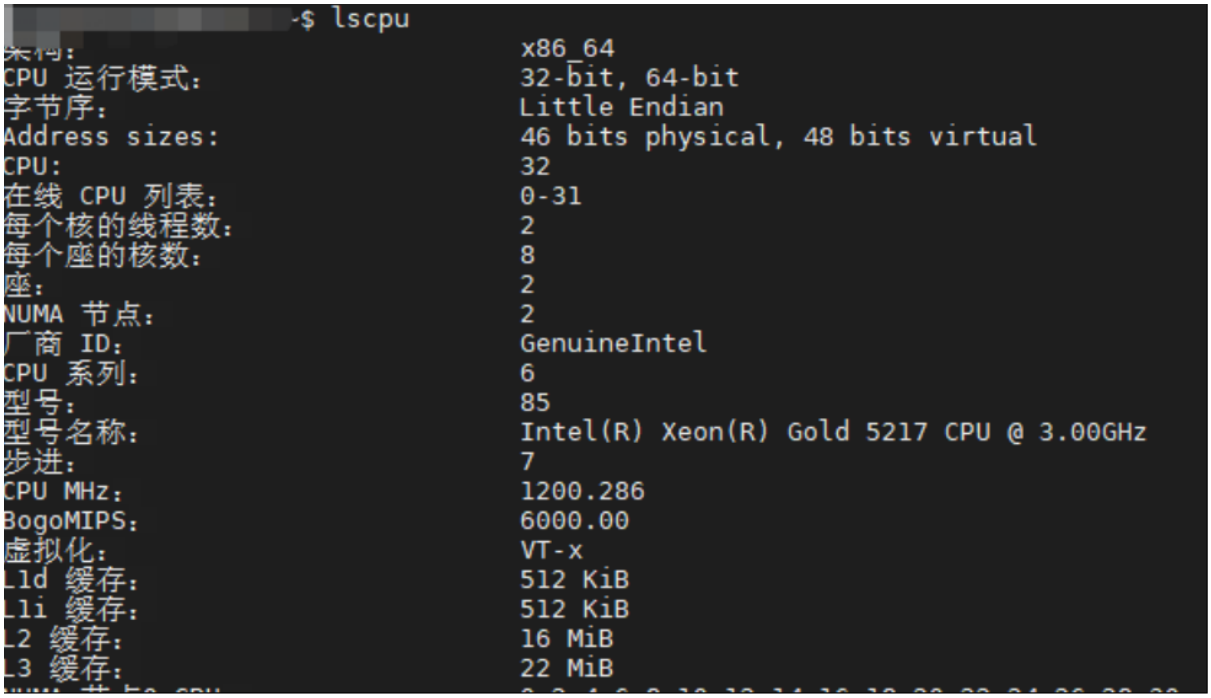
2.10.5 RVProf 功能的使用

RVProf 是芯来科技针对 cpu cycle model 开发的性能分析工具，Nuclei Studio 在 2024.02.dev 版本中，完成对 RVProf 的支持。在实际使用中，RVProf 功能分三步完成，首先通过 Cycle model 工具，运行代码，产生 `.rvtrace` 文件，然后 RVProf 工具，将 `.rvtrace` 解析成对应的 `.json` 文件，最后通过 google 的开源工具 Perfetto Trace Viewer 对 `.json` 文件进行解析并展示。因为 cpu cycle model 当前仅提供了 linux 版本，所以本文档均是在 linux 环境下演示此功能。

在使用过程，如有问题，可以查看 https://github.com/Nuclei-Software/nuclei-studio 相关内容，也可以向我们提交相关 issue。

测试环境

cpu cycle model 在运行过程中，对硬件环境的性能要求较高，在实际使用，四核及以上的系统中运行效果较好，一般不建议在虚拟机环境下使用。为了较好的体验效果，本测试在工作站上进行。



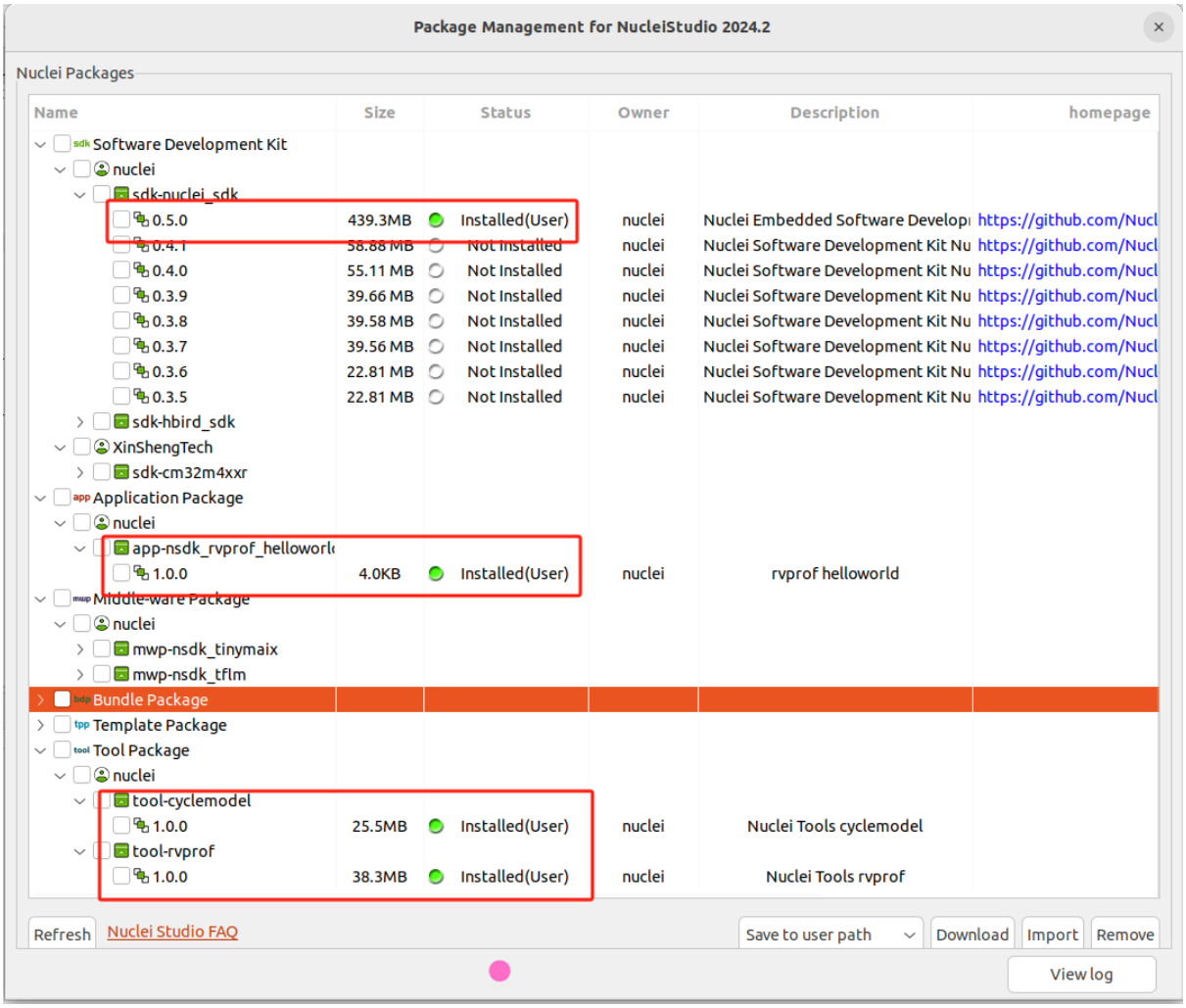
准备测试 NPK 软件或者工具包

目前此功能仅提供测试用的 NPK 包，将相关的包安装到 Nuclei Studio 中，关于安装 NPK 包，可以查看 Nuclei Studio 手册中相关章节，因为 RVProf 测试包没有公开，请联系我们索取。

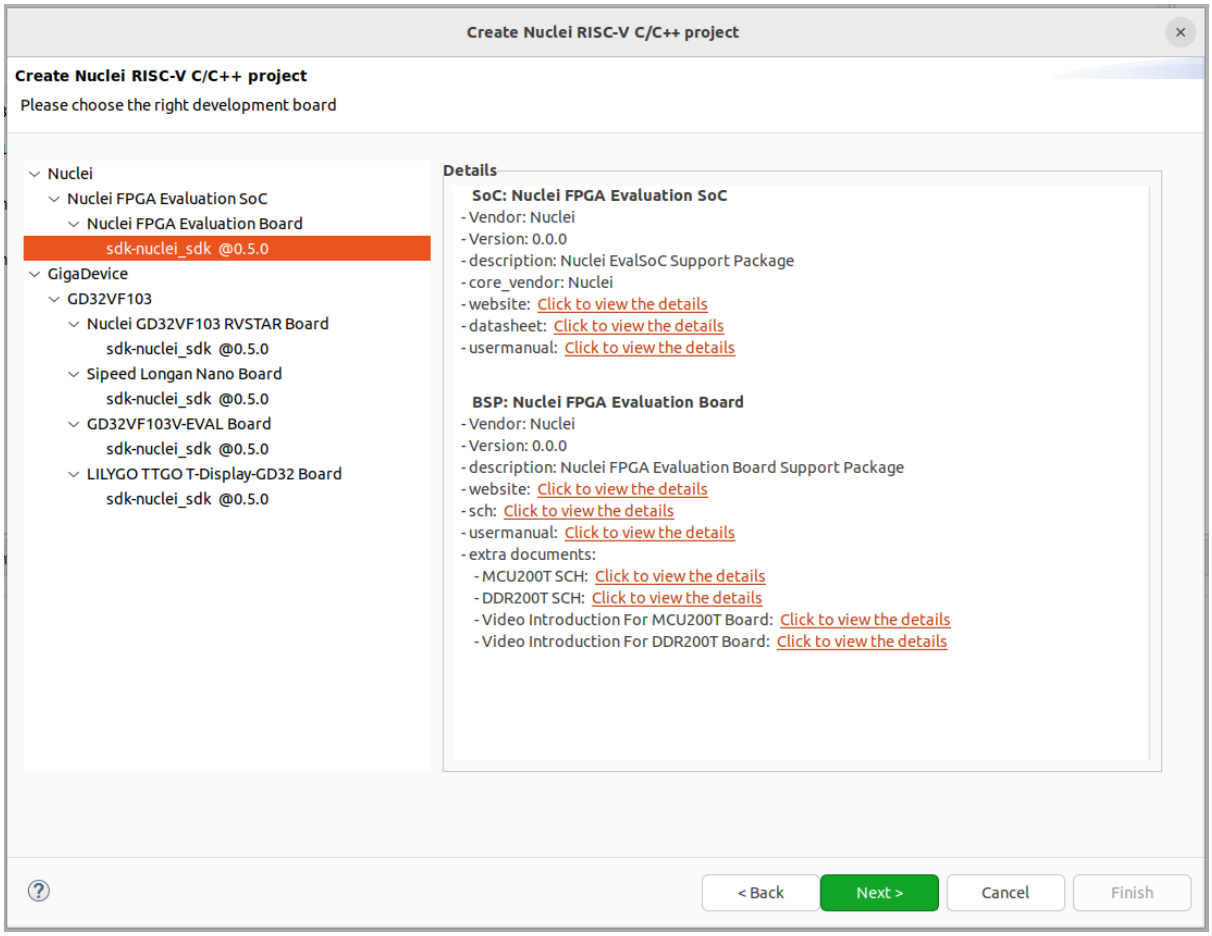
- cymodel.zip cymodel 的 NPK Tools 包
- rvprof.zip RVProf 的 NPK Tools 包
- Rvprof helloworld.zip 测试 demo NPK App 包

创建 rvprof 测试工程

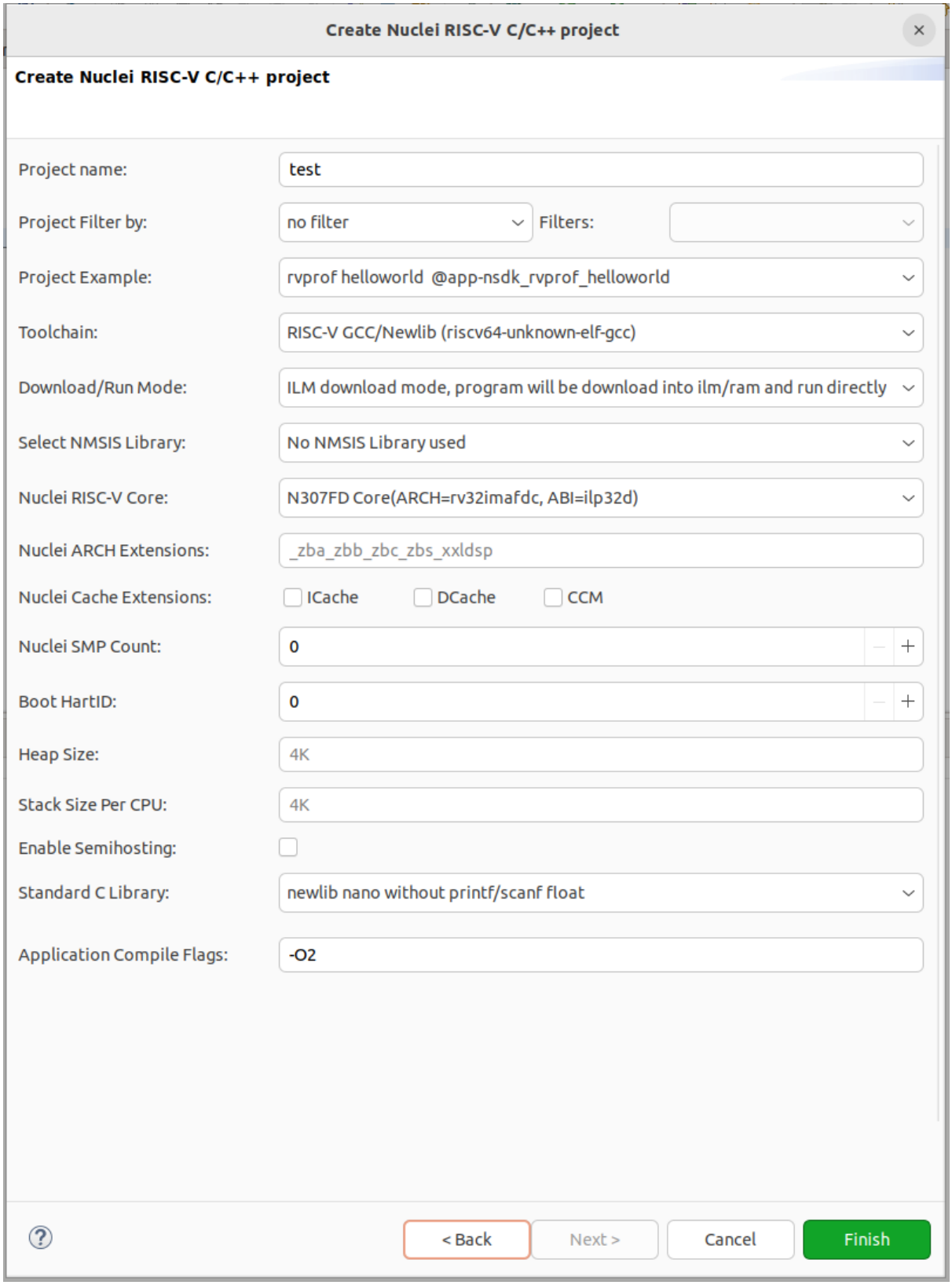
创建工程前，先查看 Nuclei Package Management 中 NPK 是否安装正确，因为测式 demo 是依赖于 nuclei_sdk，所以也要先安装 sdk-nuclei_sdk，具体如下：



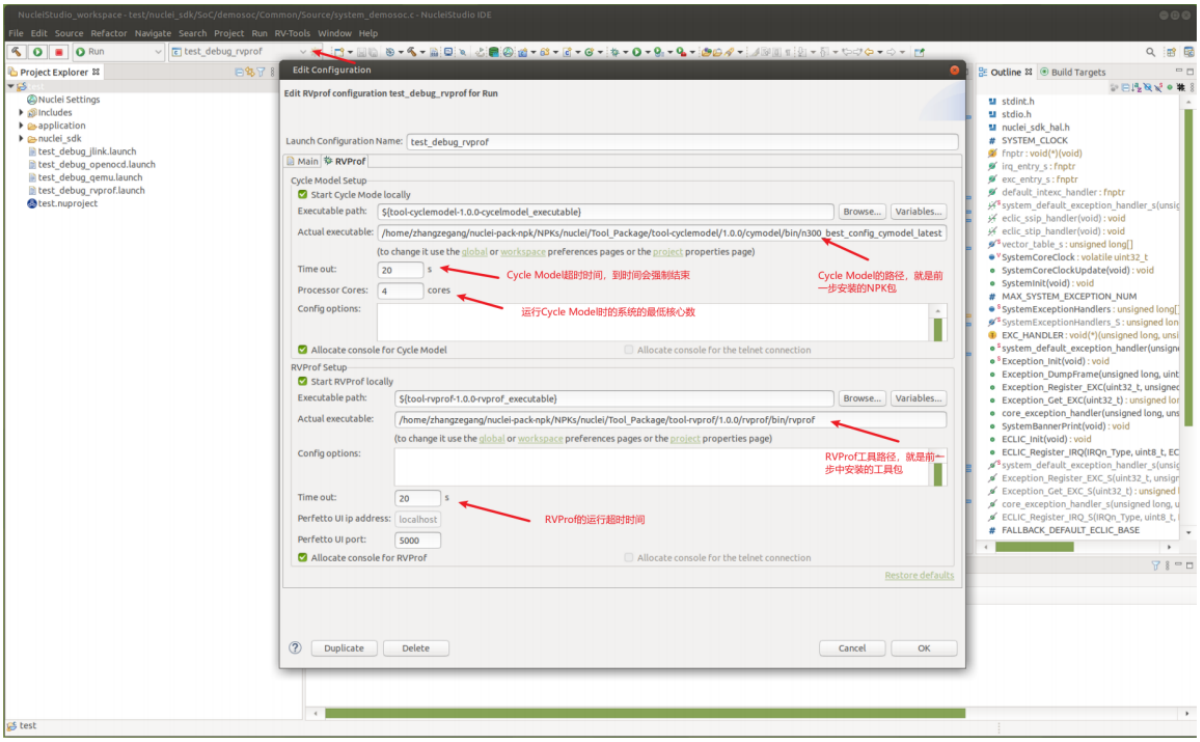
然后创建一个 test 测试工程, 在创建工程的向导中，依次 `New Nuclei RISC-V C/C++ Project -> sdk-nuclei_sdk@0.5.0 -> next` , 在工程配置页面，依次填写工程名、选择 Project Example : `rvprof helloworld@app-nsdkrvprof_helloworld` ,Nuclei RISC-V Core: `N307FD` （这里的 code 要跟 cpu cycle model 对应）。



在 Project Example 可以看到我们导入的 demo NPK App 中的 Rvprof helloworld 工程，选择此工程，然后下一步，完工程的创建。



在创建的 test 工程中，可以看到多了一个 `test_debug_rvprof.launch` 文件，rvprof 相关的配置在此文件中，可以查看内容如下。其中 Cycle Model 的 time out 时间，用来设置 Cycle Model 超时时间，因为 Cycle Model 运行时比较耗时，如果工程比较简单，可以设置一个较短的起时时间，到时间后，可以及时中断 Cycle Model 的运行；RVProf 中的超时时间的功能也是类似。

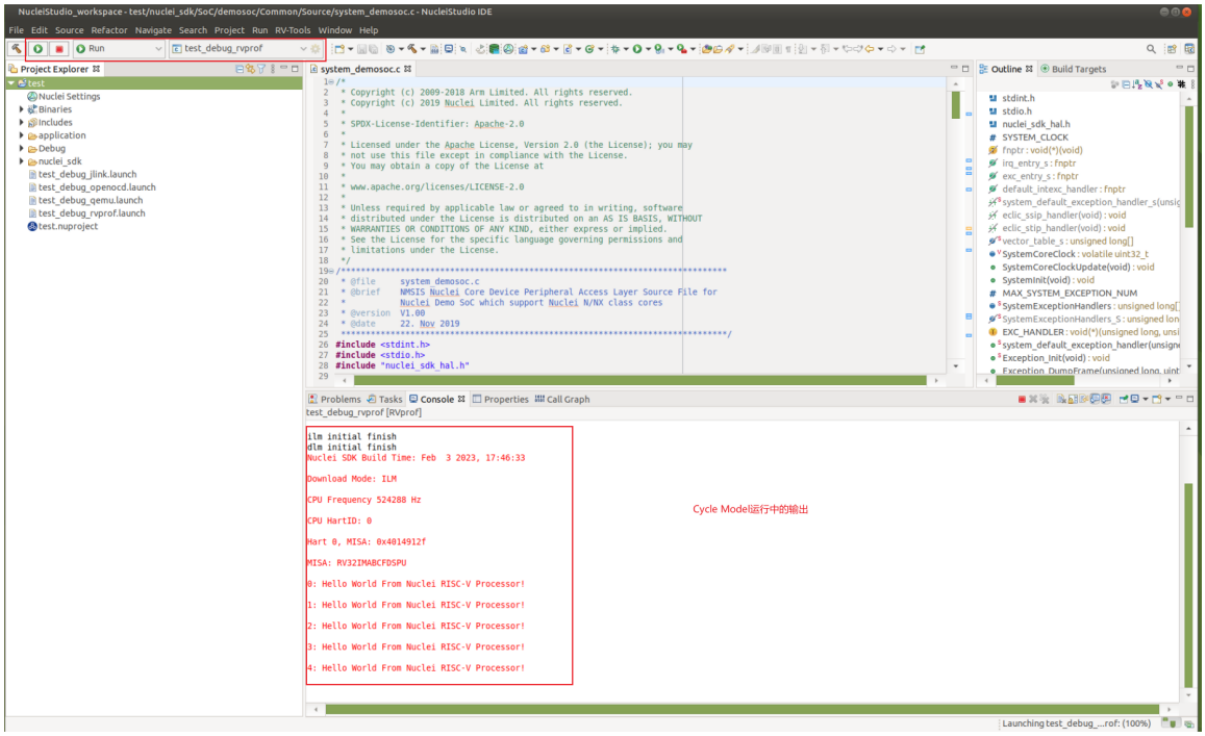


查看 rvprof 的结果

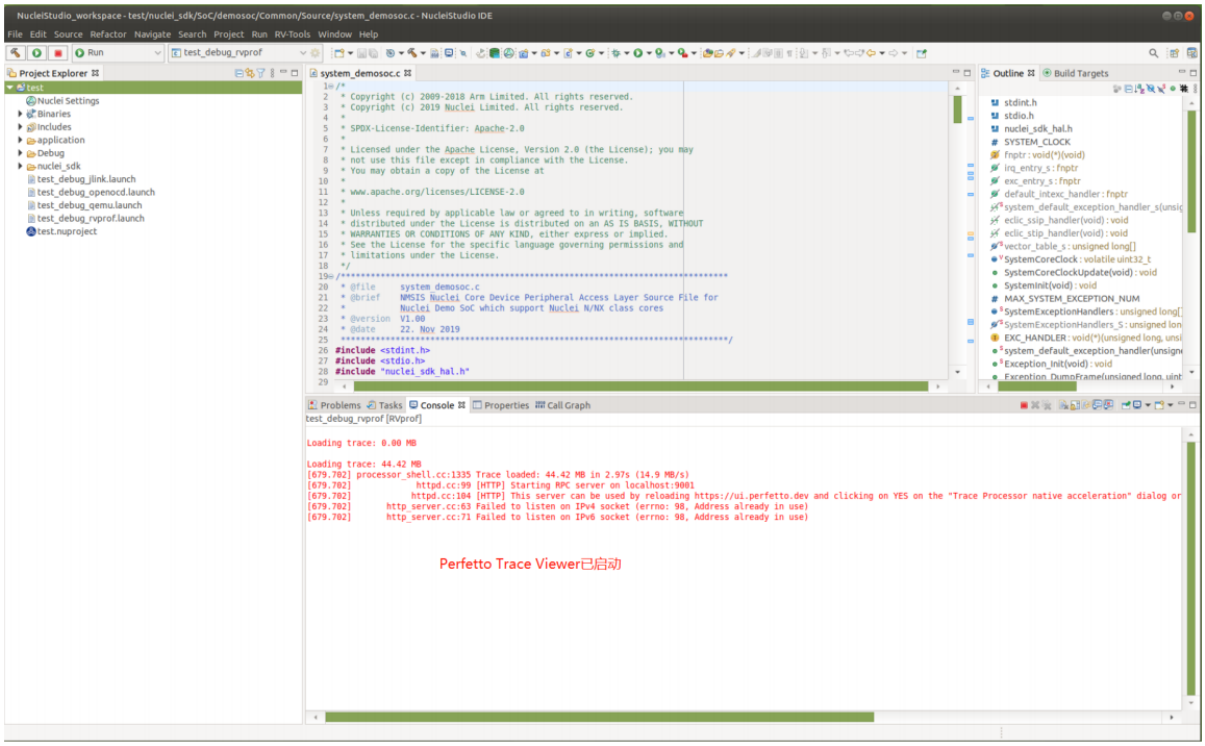
创建完工程后，在 Nuclei Studio 的 launch bar 上，选中 `test_debug_rvprof.launch`，并点击工具栏中的运行按钮，Nuclei Studio 依次完成以下任务，并将最终的结果在在 Perfetto Trace Viewer 中展示。

- 编译工程代码
- 启动 Cycle Model 并产生 trace 文件
- 启动 RVProf 解析 trace 文件生成 json 文件
- 启动 Perfetto Trace Viewer 展示结果

Cycle Model 启动及 log 输出

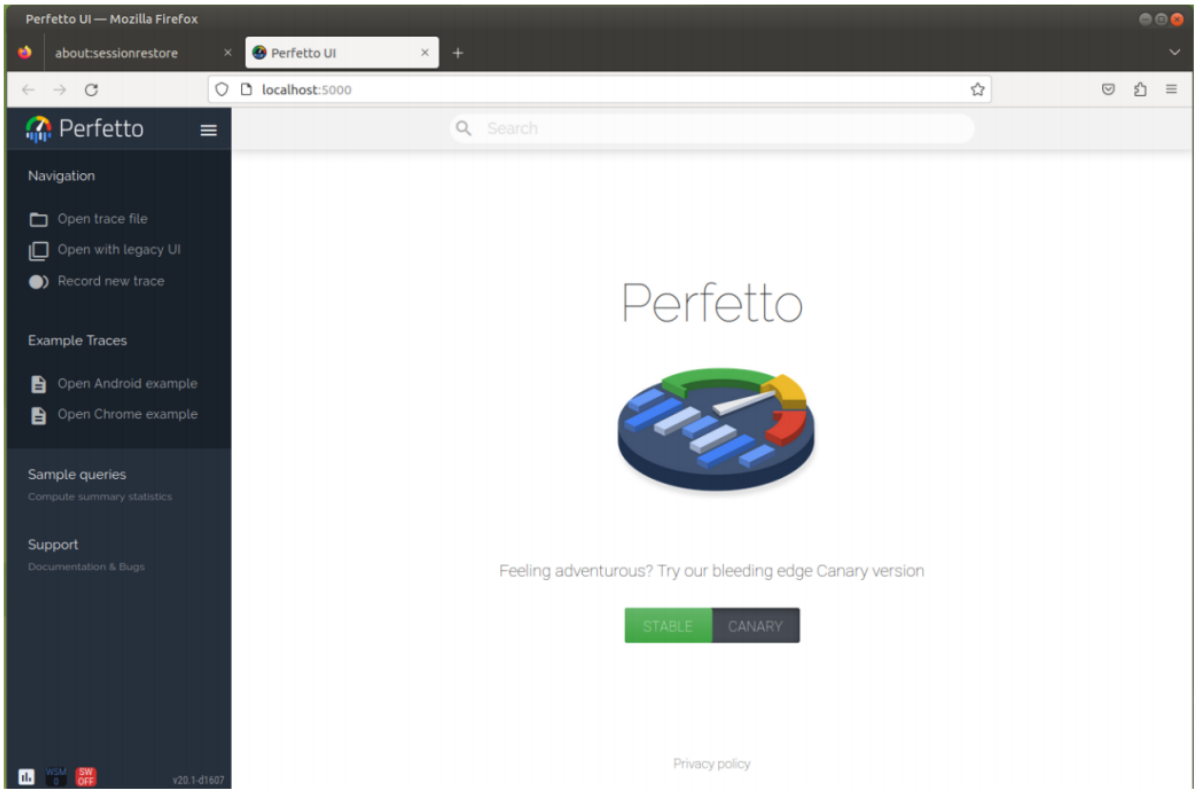


perfetto 启动本地服务

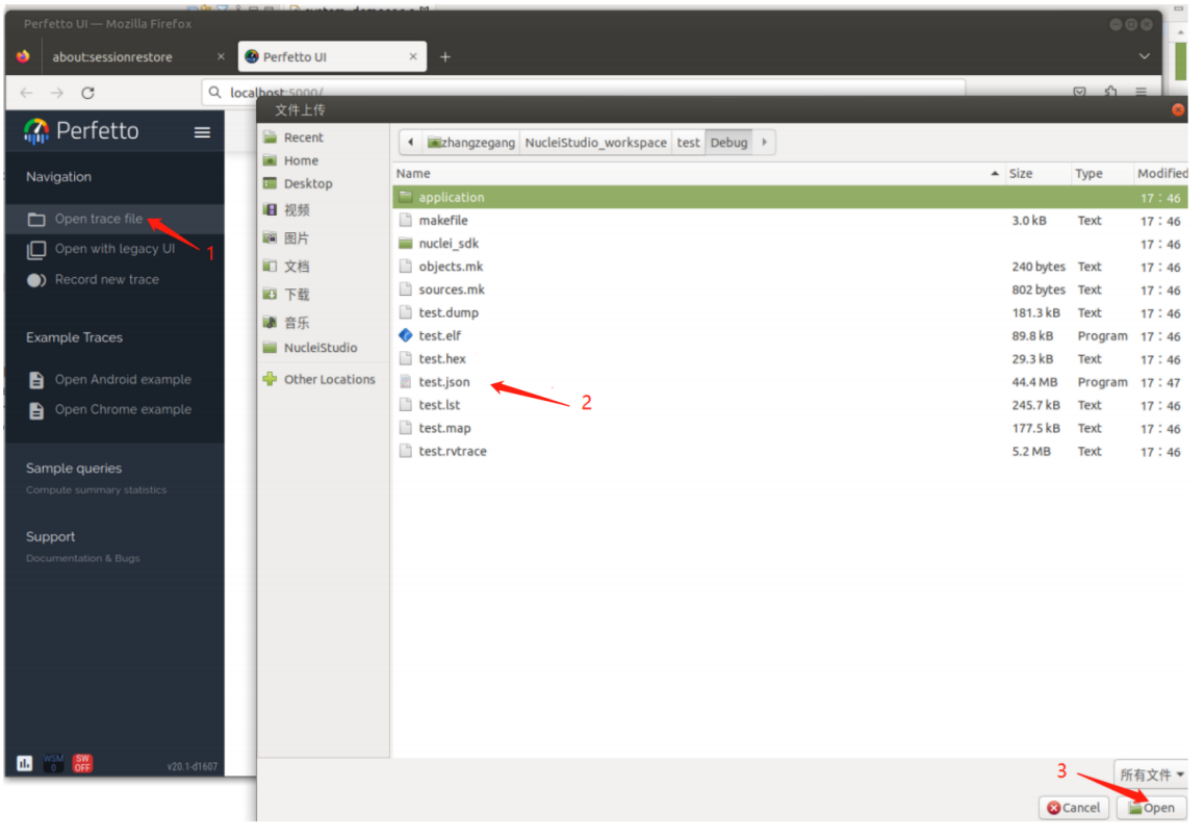


Perfetto Trace Viewer 的官方地址是 https://ui.perfetto.dev/ 。Nuclei Studio 默认会尝试打开 https://ui.perfetto.dev/ ，同时自动载入 json 文件并解析。如果因为网络原因（国外服务器）打开失败，Nuclei Studio 会在本地启一个 Perfetto Trace Viewer 本地服务，并自动打开本地 localhost:5000/，此时需要用户手动载入工程目录下的 `Debug/test.json` 文件。在 Perfetto Trace Viewer 中可以看到 trace 的展示结果。

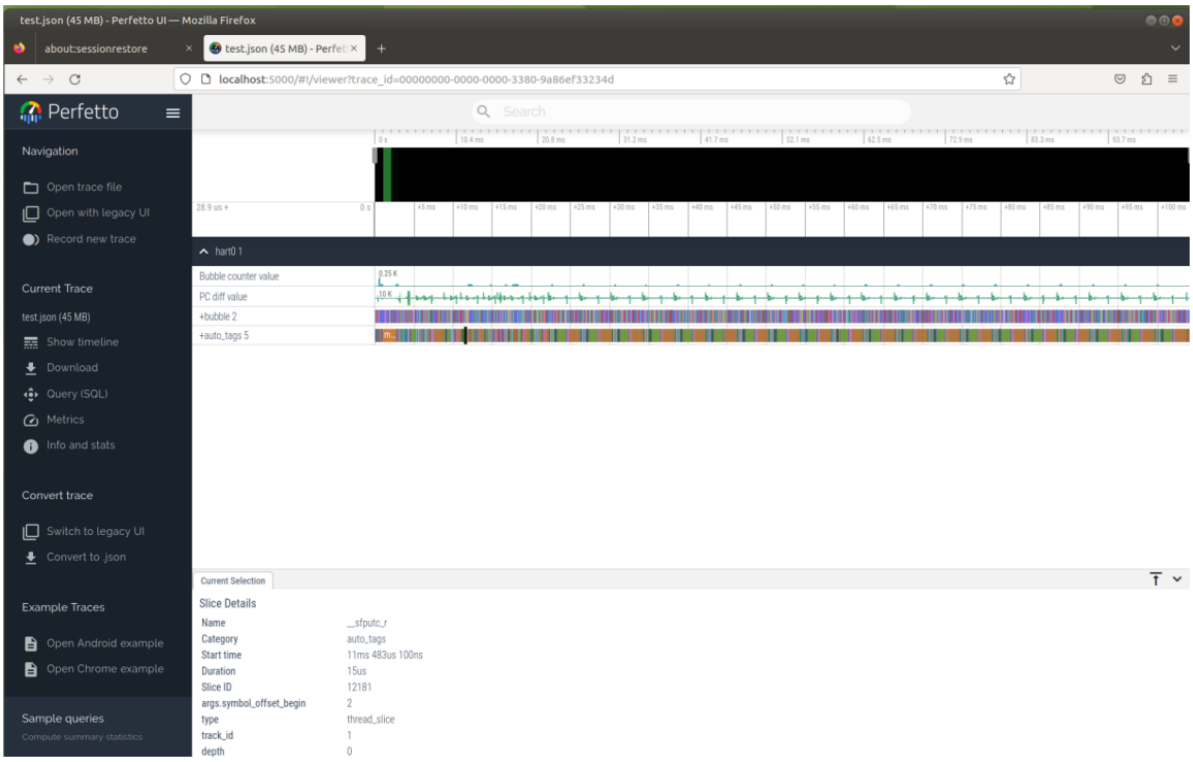
Nuclei Studio 会在本地启一个 web 服务，同时打开 Perfetto Trace Viewer。



点击 Open trace file，找到工程中生成的 json 文件，手动将 json 文件 load 到 Perfetto Trace Viewer 中。



些时，在 Perfetto Trace Viewer 就可以查看到 rvprof trace 结果展示了，用户可以通过键盘的 W/A/S/D 按键查看更详细的信息。



2.10.6 使用 Nuclei Near Cycle Model 仿真性能分析

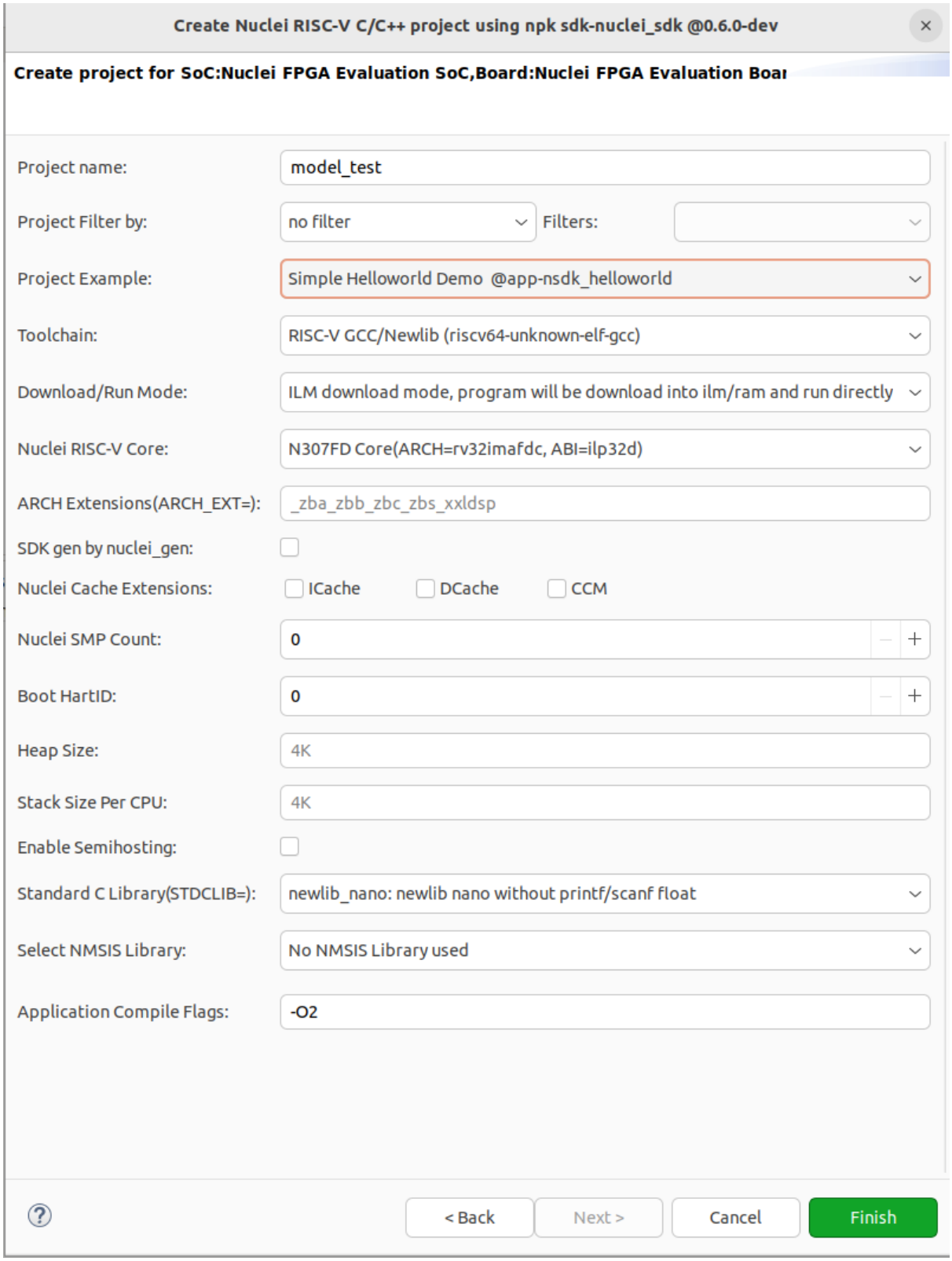
在 Nuclei Studio 2024.06 版中，集成了 Nuclei Near Cycle Model，它是由芯来科技自主研发的仿真测试和性能分析工具，可以帮助研发人员在项目初期进行一些必要的仿真测试和程序性能分析。

Nuclei Near Cycle Model 当前只有 Linux 版本，其具体介绍和命令行上使用参见（https://doc.nucleisys.com/nuclei_tools/xlmodel/intro.html），下面将在 Nuclei Studio 上演示如何使用 Nuclei Near Cycle Model 进行仿真和性能分析。

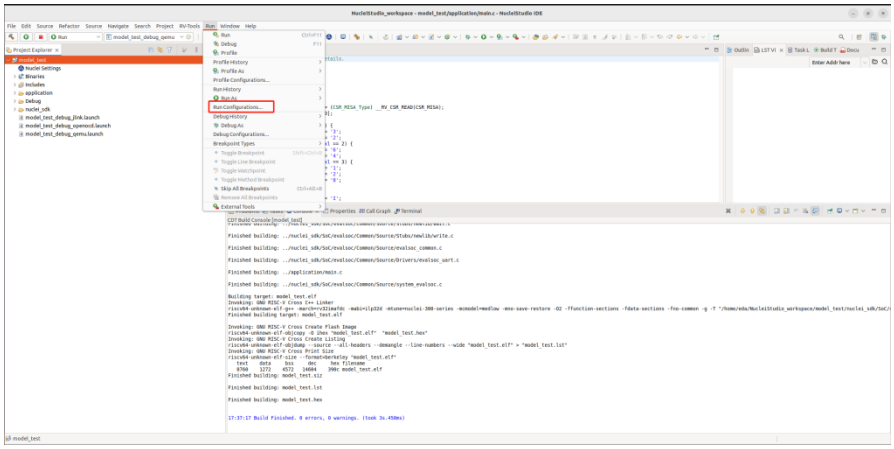
在使用过程，如有问题，可以查看 https://github.com/Nuclei-Software/nuclei-studio 相关内容，也可以向我们提交相关 issue。

创建测试工程

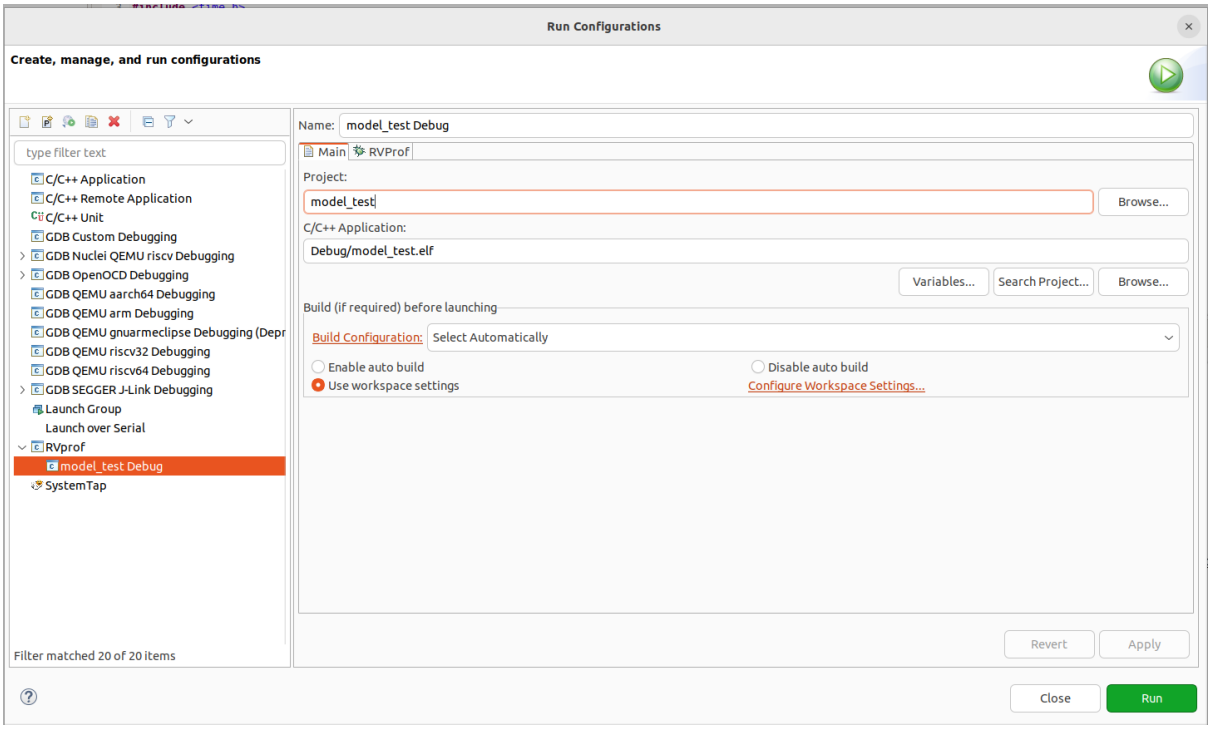
Nuclei Near Cycle Model 对芯来全类型的 Core 都有支持，可以创建任意一个 demo 工程并编译。创建任意一个 demo 工程并编译。



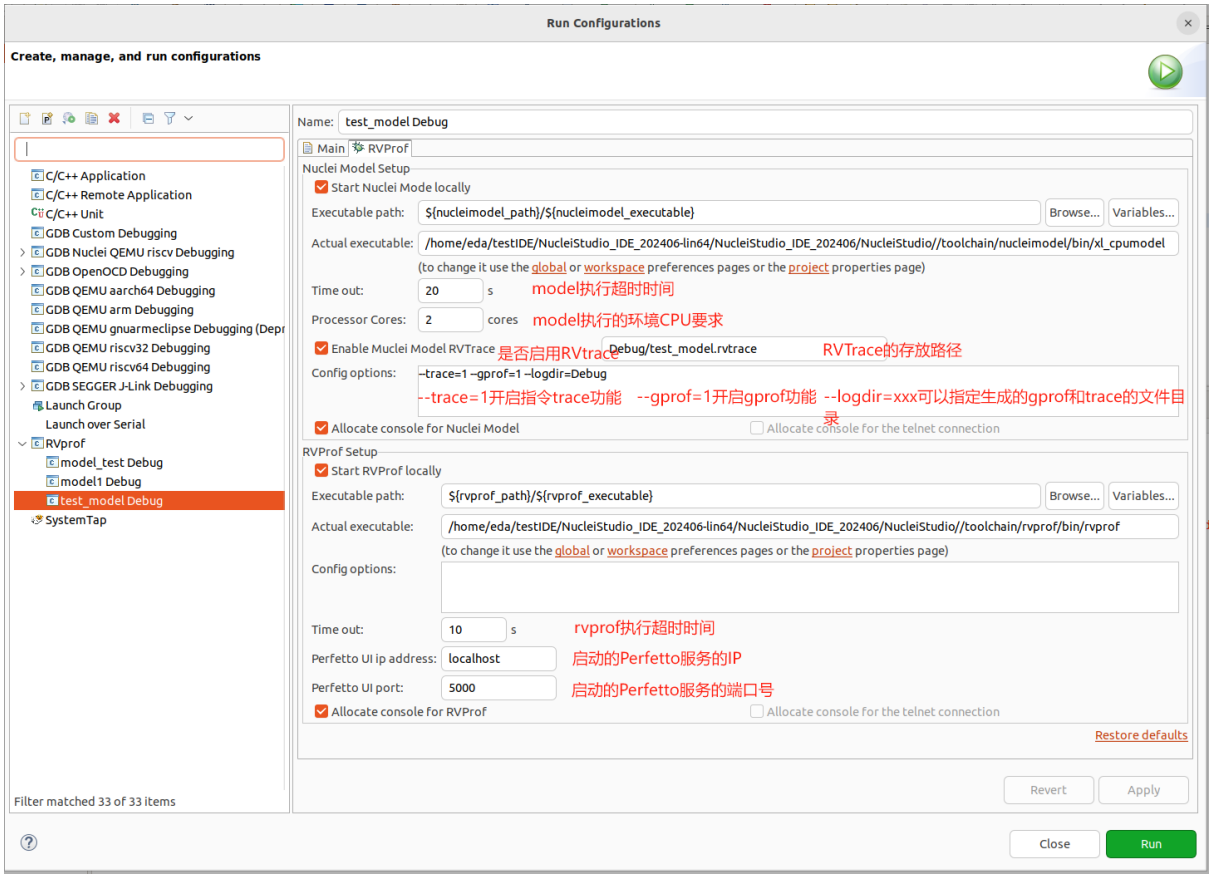
Nuclei Near Cycle Model 采用 Nuclei Studio 中的 RVProf 运行配置来进行运行测试，选中编译好的测试工程，然后打开 NucleiStudio 的 Run Configurations。



并创建一个 RVProf 的配置，具体的配置及参数说明如下。



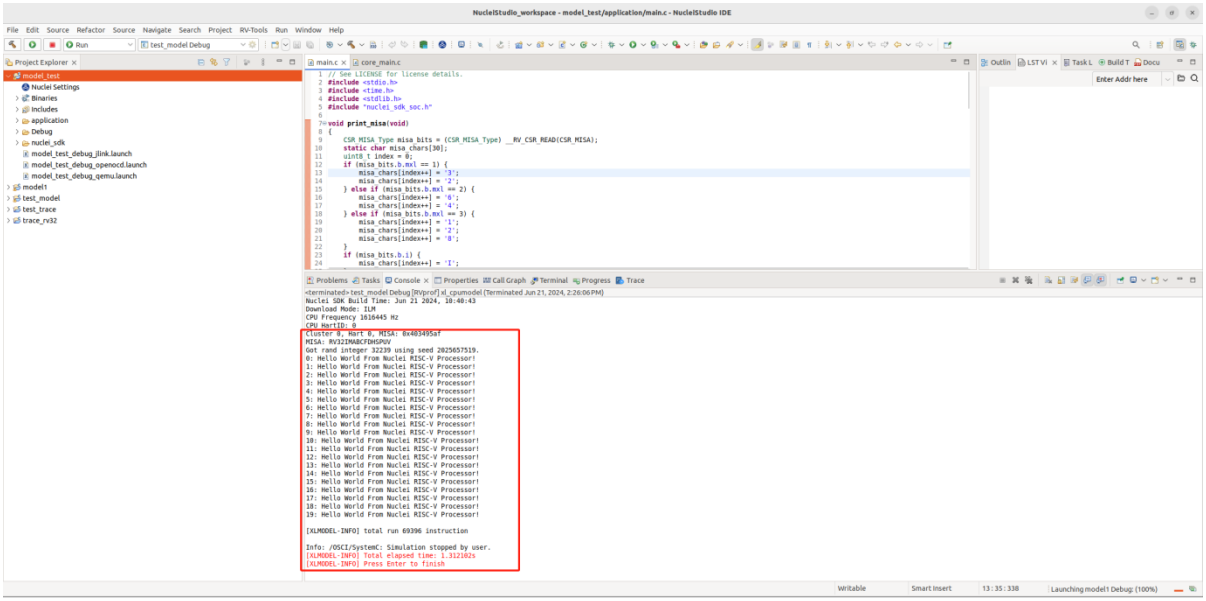
其中在 Config options 中需要配置 `--trace=1 --gprof=1 --logdir=Debug`，`--trace=1` 表示开启 rvtrace，`--gprof=1` 表示开启 gprof 功能，`--logdir=Debug` 则表示最终生成的 `.rvtrace` 文件、`.gmon` 文件存存放的路径为当前工程下的 Debug 目录。



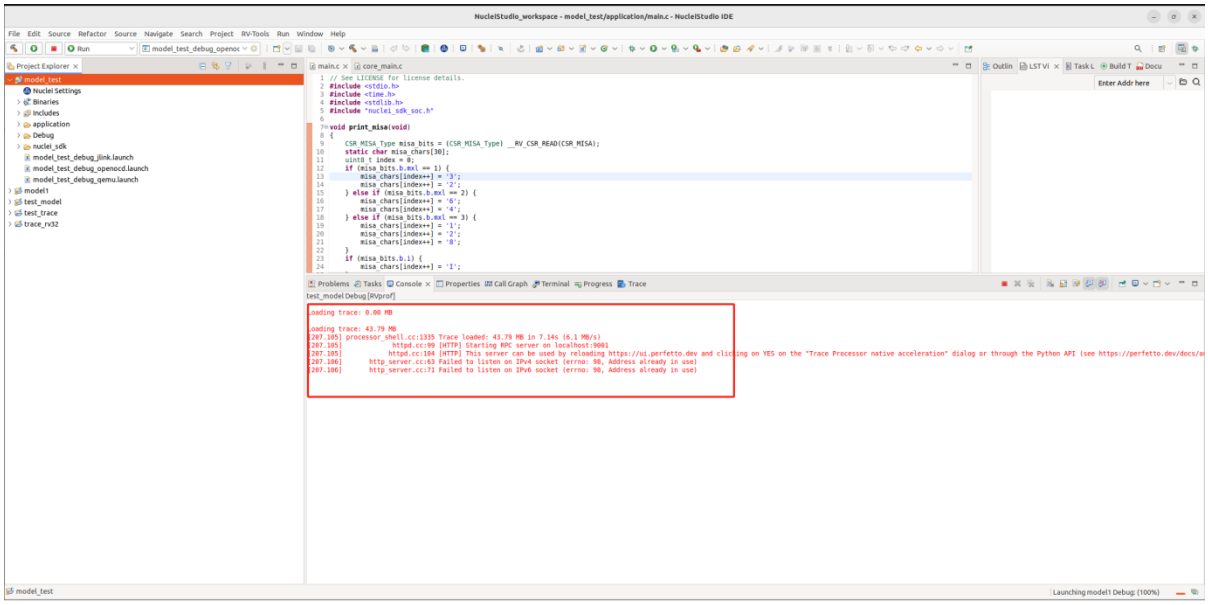
运行工程并生成性能分析结果

点击运行（Run）工程，NucleiStudio 会依次调用 Nuclei Near Cycle Model 来仿真程序执行，并产生 .rvtrace 文件，再调用 rvprof 来解析 .rvtrace 文件并生成 .json 文件，最后启用一个 perfetto 服务来用来查看 rvprof 解析 .rvtrace 文件所产生的 .json 文件。

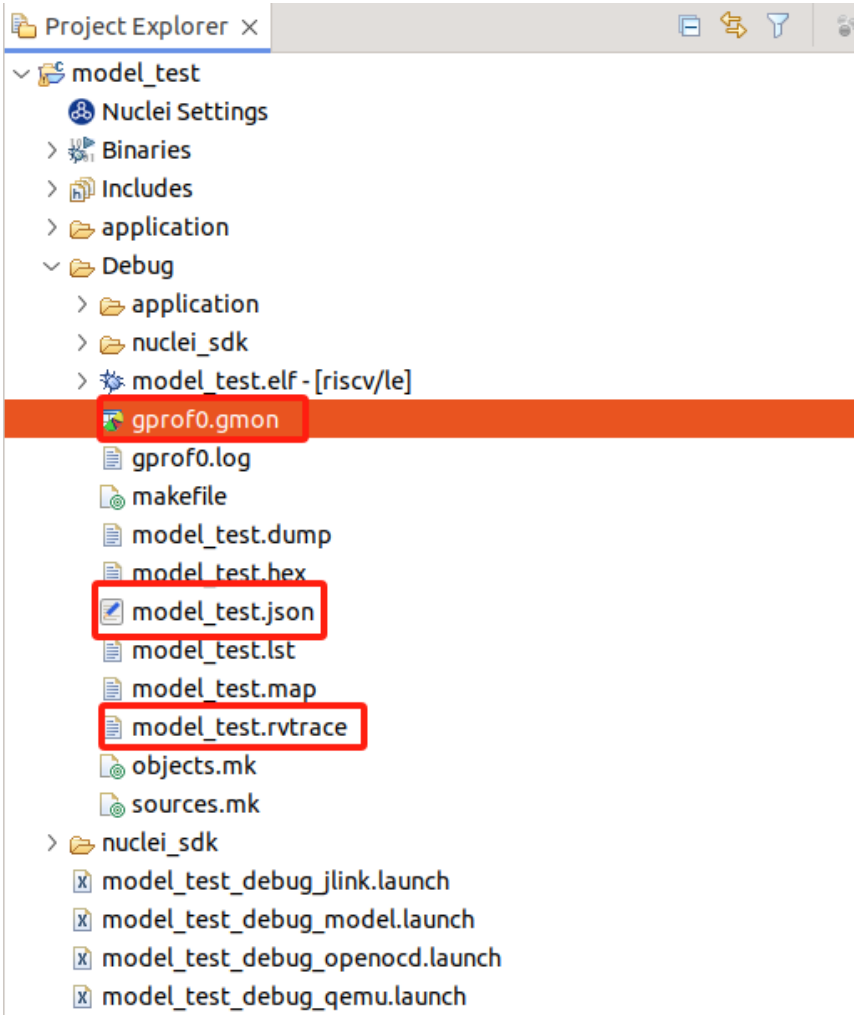
点击 Run 按钮，开始运行程序。



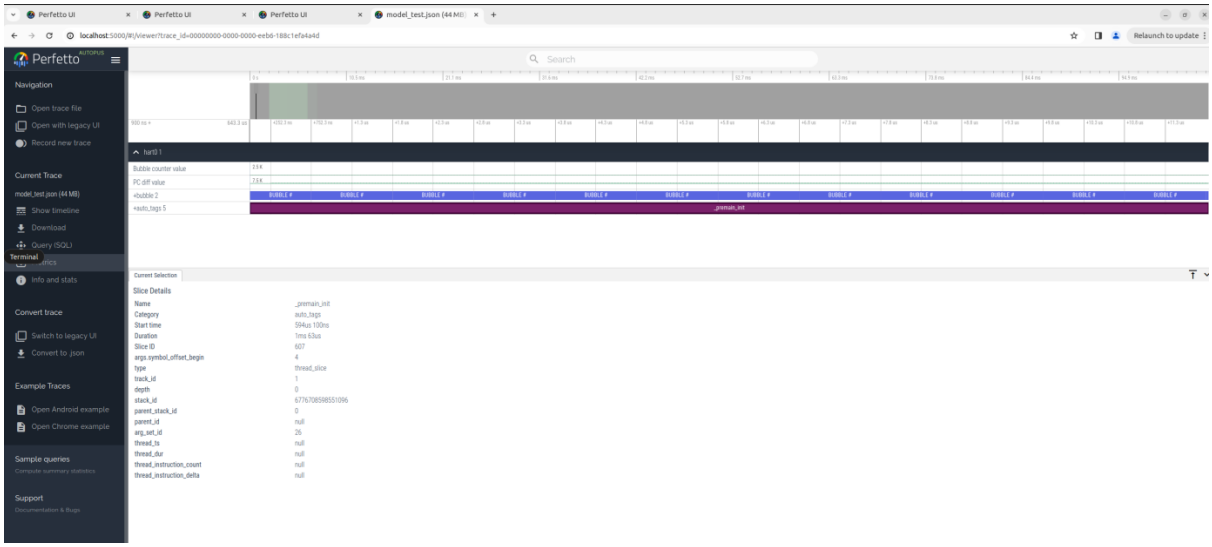
程序在 Nuclei Near Cycle Model 中成功执行，输出了对应的 Log 信息。



在工程的 Debug 目录中可以查看到已经生成 .rvtrace 文件、.json 文件、.gmon 文件。

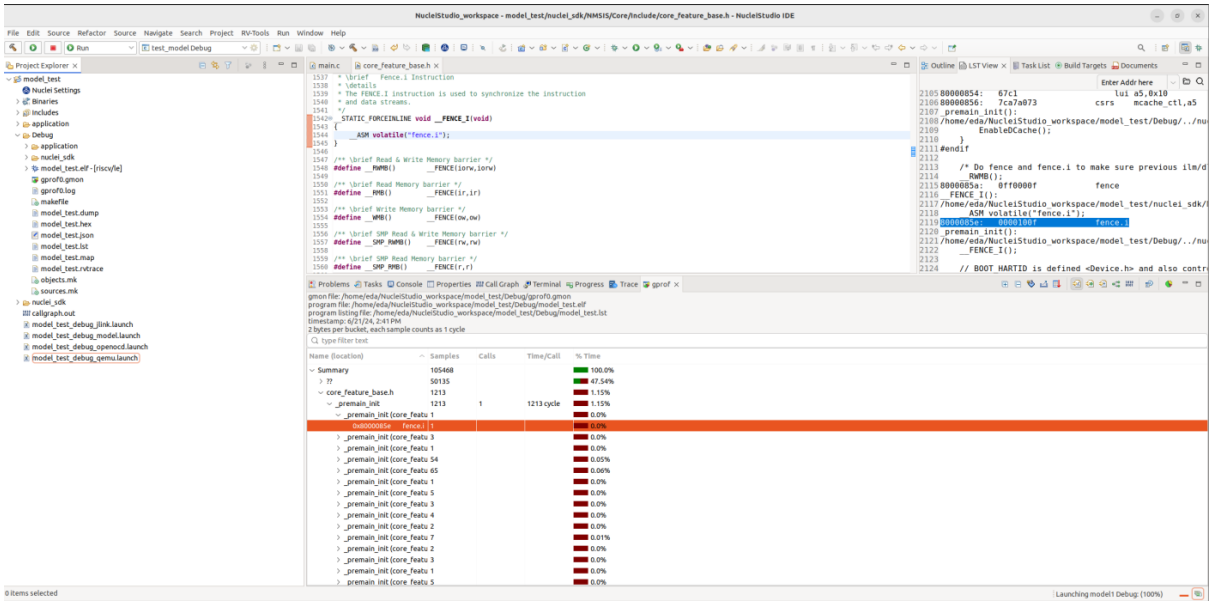


Nuclei Studio 会在本地启一个 web 服务，同时打开 Perfetto Trace Viewer。通过 Perfetto Trace Viewer，可以查看 .json 文件。

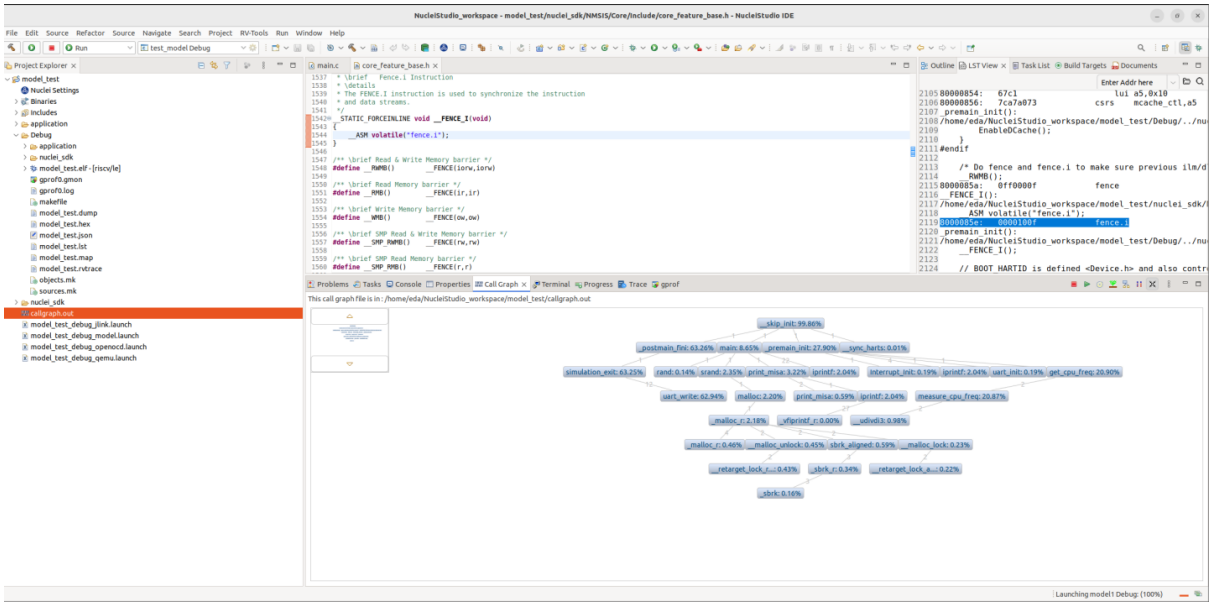


Nuclei Near Cycle Model 中支持通过 gprof 来分析程序，所以当我们配置了 `--gprof`，在程序运行时，也会在 Debug 目录（`--logdir=XX` 所配置的目录）下同步产生一个 `.gmon` 文件，双击 `.gmon` 文件，将调用 gprof 工具来分析程序执行所消耗的 cycle 数及调用关系；同时也会产生对应的 `callgraph.out` 文件，双击 `callgraph.out` 文件，调用 Call Graph 查看程序的调用关系。

调用 gprof 工具，可以查看生成的 `.gmon` 文件中的内容。



gprof 工具在查看 `.gmon` 文件的同时，会根据其内容，解析出程序的调用关系，并生成 `callgraph.out` 文件，双击 `callgraph.out` 调用 Call Graph 工具查看。



2.11 Nuclei Studio 升级

一般情况下，如果 Nuclei Studio 没有大的版本变动，Nuclei Studio 升级工作可以由用户下载最新的 GNU 工具链以及其他相关工具和升级 IDE Plugins 的方式来完成。

2.11.1 GCC/OpenOCD 等工具链的安装

Nuclei Studio 已经把工具链集成在 IDE 内部，工具链存放在 Nuclei Studio_IDE_XXX 文件夹内，路径为：`Nuclei Studio_IDE_XXX\\NucleiStudio\\toolchain`。IDE 默认使用此路径的工具链，所以请不要移动 `toolchain` 此文件夹，使用 IDE 创建工程后也不需要进行工具链的相关配置。

由于工具链升级的速度会比 IDE 快，所以后期需要用户手动升级工具链，工具链升级方法如下：

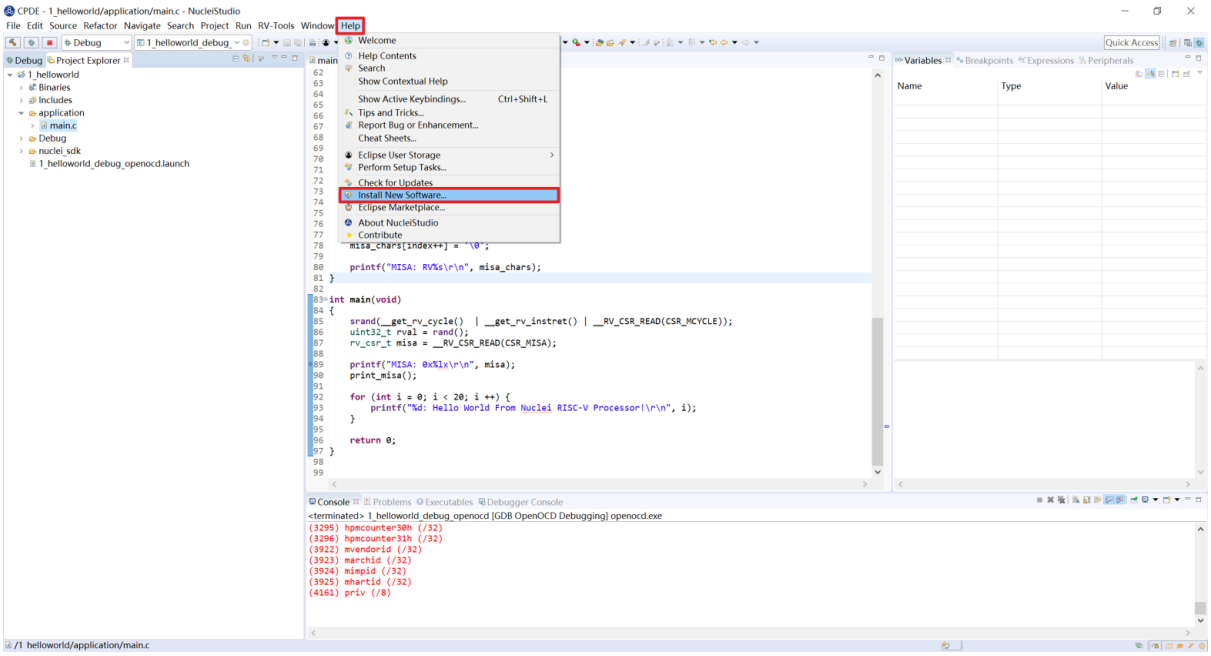
- 首先删除需要更新的 GCC 或者 OpenOCD 文件内的内容，然后在芯来科技官网下载最新版本的 GCC 或者 OpenOCD。
- 将 GCC 或者 OpenOCD 的内容复制到对应的文件夹下，即可完成 IDE 工具链的更新，IDE 自带的工具链的版本号记录在 ReadMe.txt 中。

Note: 注意：要保证 gcc 所在的这一级目录文件夹名字不变，替换后保证 bin 文件夹所在层级是图中文件夹的子目录，中间不要有其他文件夹。

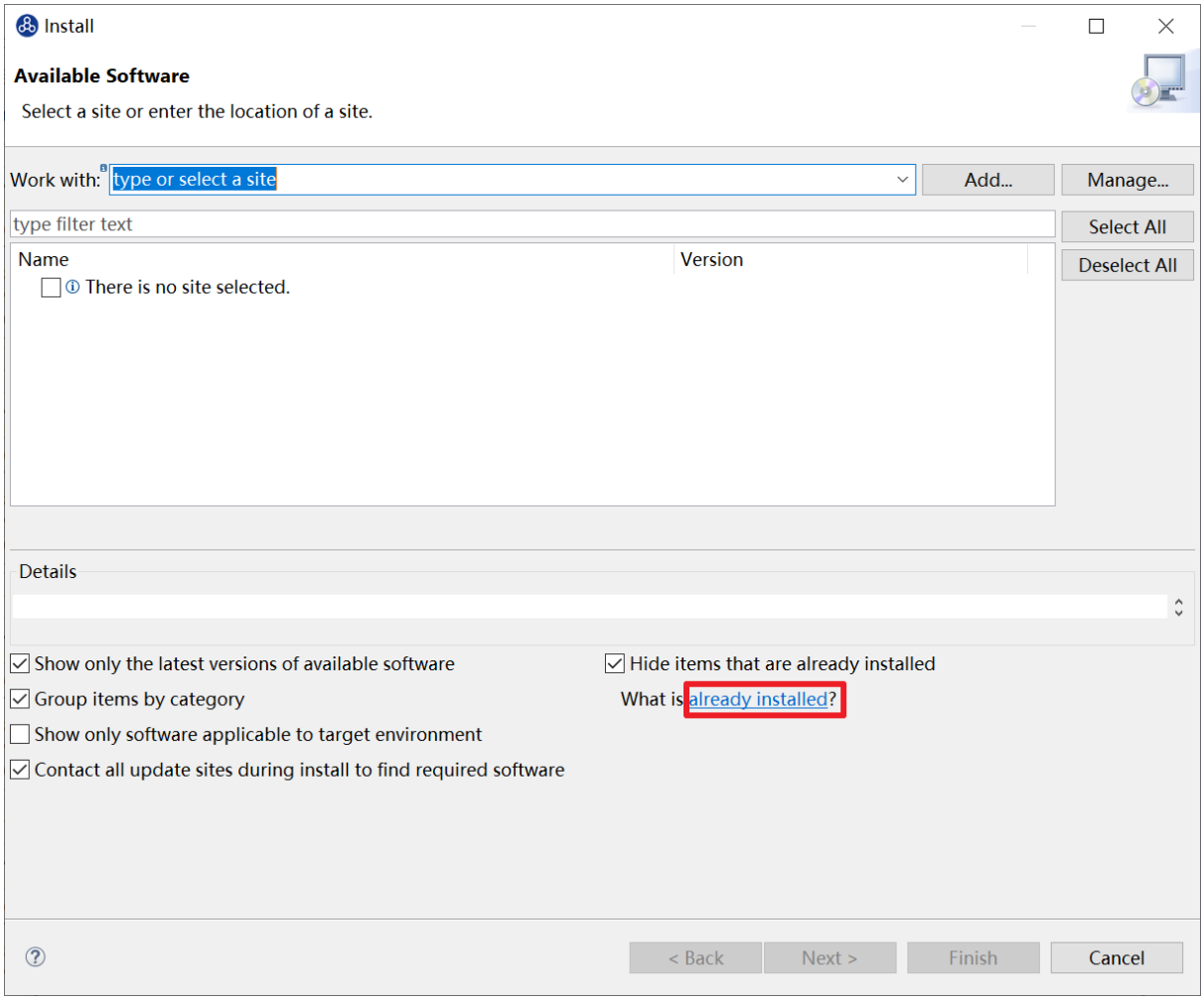
2.11.2 IDE Plugins 升级

Nuclei Studio 支持 IDE 内在线升级，步骤如下。

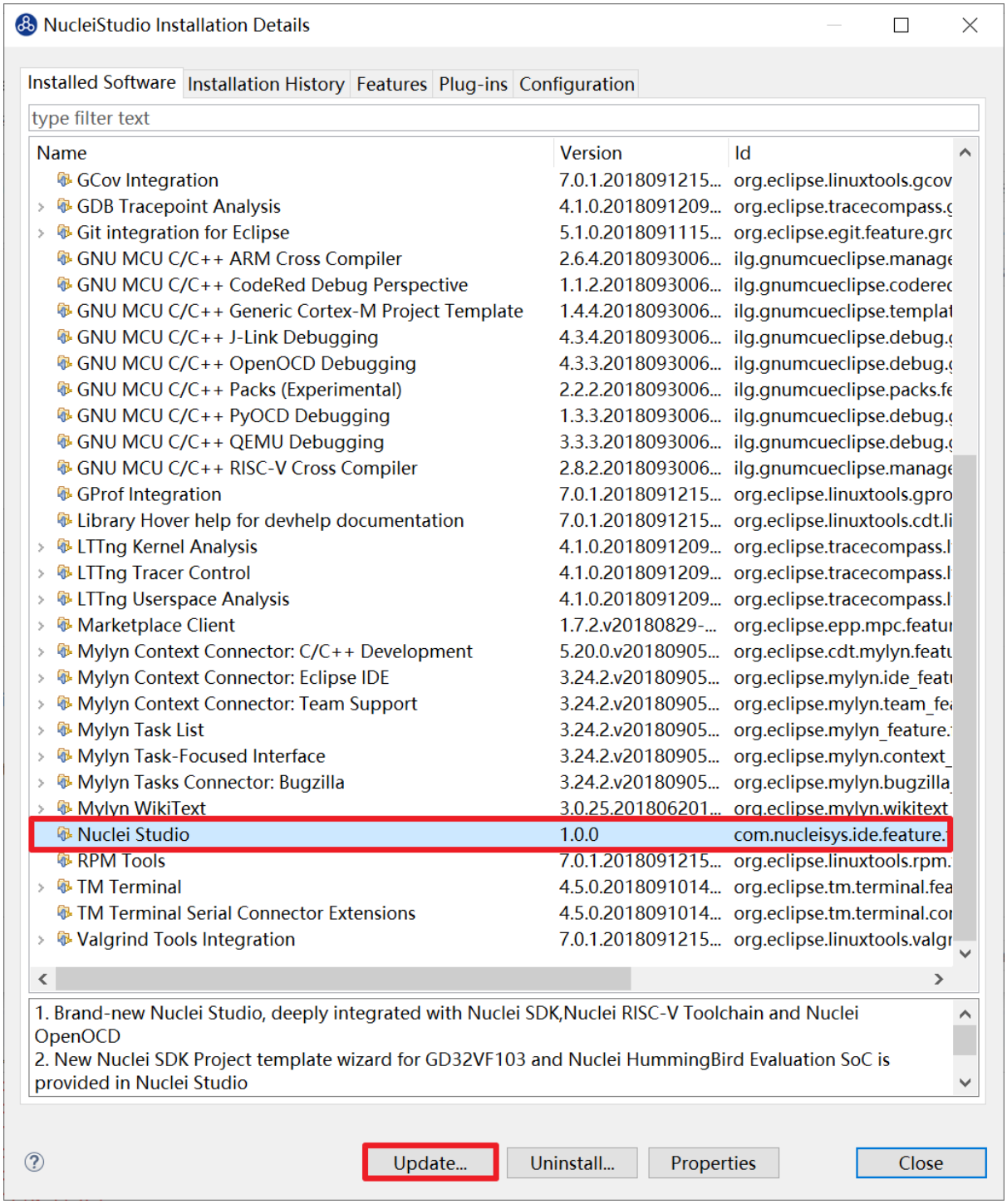
打开安装软件弹窗，在菜单栏选择 Help → Install New Software。



点击 already installed 打开已安装界面。



选择 Nuclei Studio，点击 update 即可自动完成在线更新。



2.12 常见问题

2.12.1 Nuclei Studio 启动慢

Nuclei Studio 是基于 eclipse 进行开发，继承了 eclipse 的可扩展性的特点的是同时，也存在着启动较慢的问题，第一次启动时，软件需要验证环境；生成常用的缓存文件；创建 Workspace. 在 2021.09 版的 Nuclei Studio 中，我们针对启动做了优化，在电脑性能足够的前提下，第一次启动 Nuclei Studio 能在一分钟内完成。

Nuclei Studio 最佳运行环境：

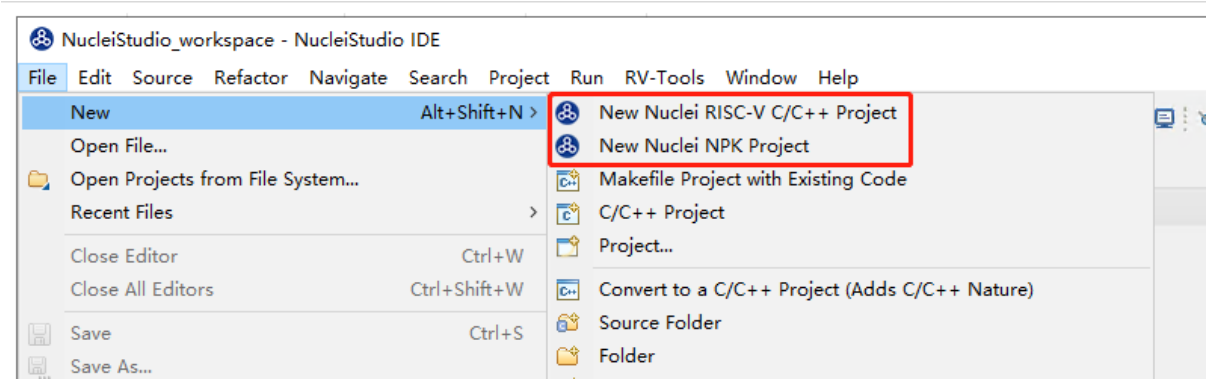
- Windows 10 操作系统
- 4G 以上内存（2G 可以被 Nuclei Studio 分配使用）

2.12.2 Nuclei Studio 编译程序很慢

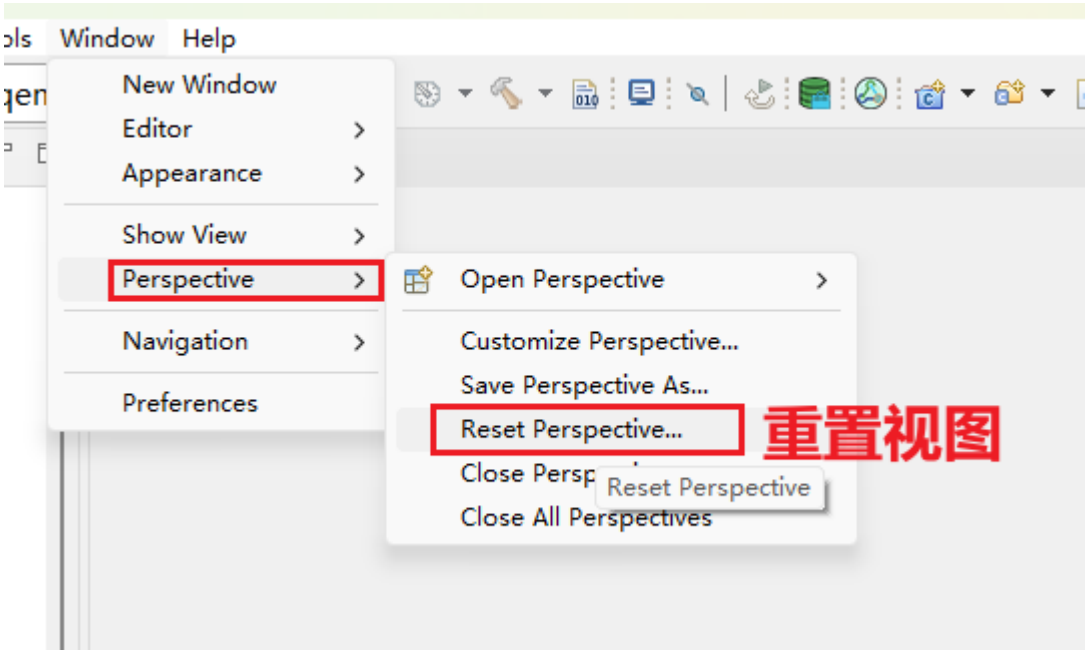
在测试使用过程中，我们发现 Nuclei Studio 在编译项目时会出现很慢的现像，经过排查发现，当电脑安装 360 杀毒软件时，编译指令执行很慢，退出 360 杀毒软件时，编译指令可以按正常速度执行。为了保障 Nuclei Studio 的正常使用，建议在使用时退出 360 等杀毒软件。

2.12.3 找不到 New Nuclei Risc-V C/C++ Project 菜单

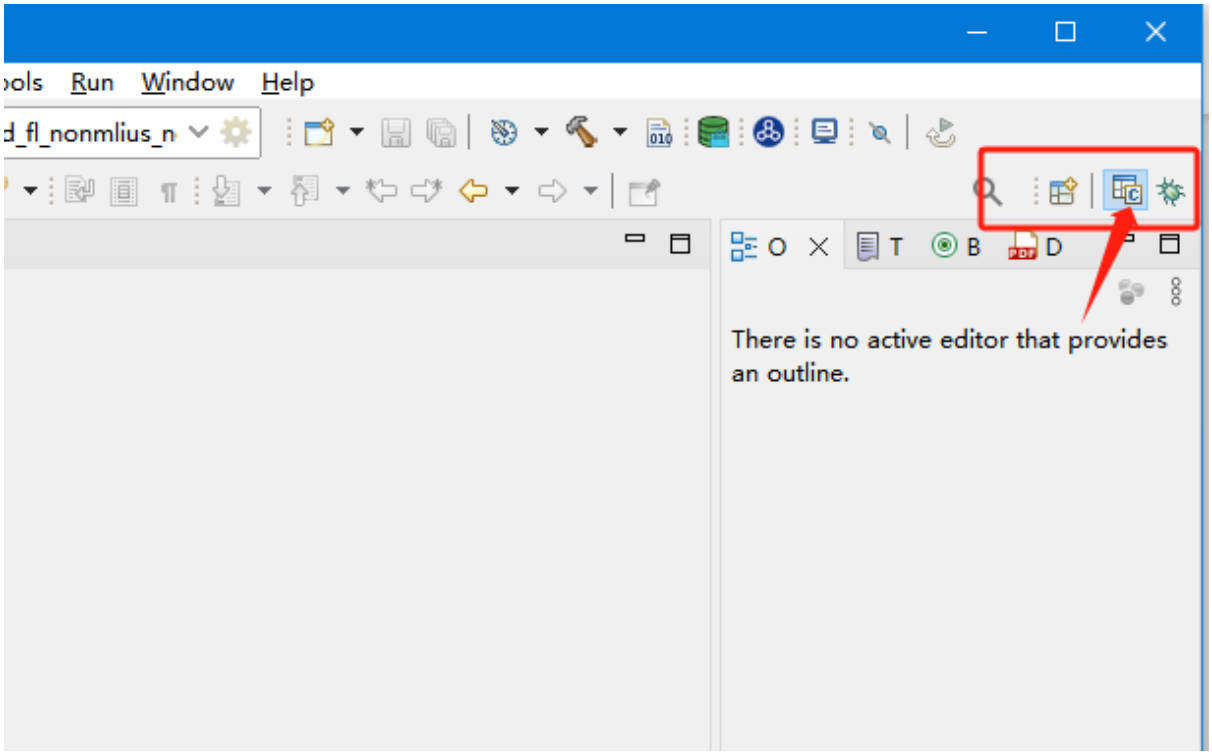
New Nuclei Risc-V C/C++ Project 是创建工程的快捷入口，有时候如果找到不到 New Nuclei Risc-V C/C++ Project 菜单。



可以在菜单 Window->Perspective->Reset Perspective 里进行视图重置，New Nuclei Risc-V C/C++ Project 就会重新出现。

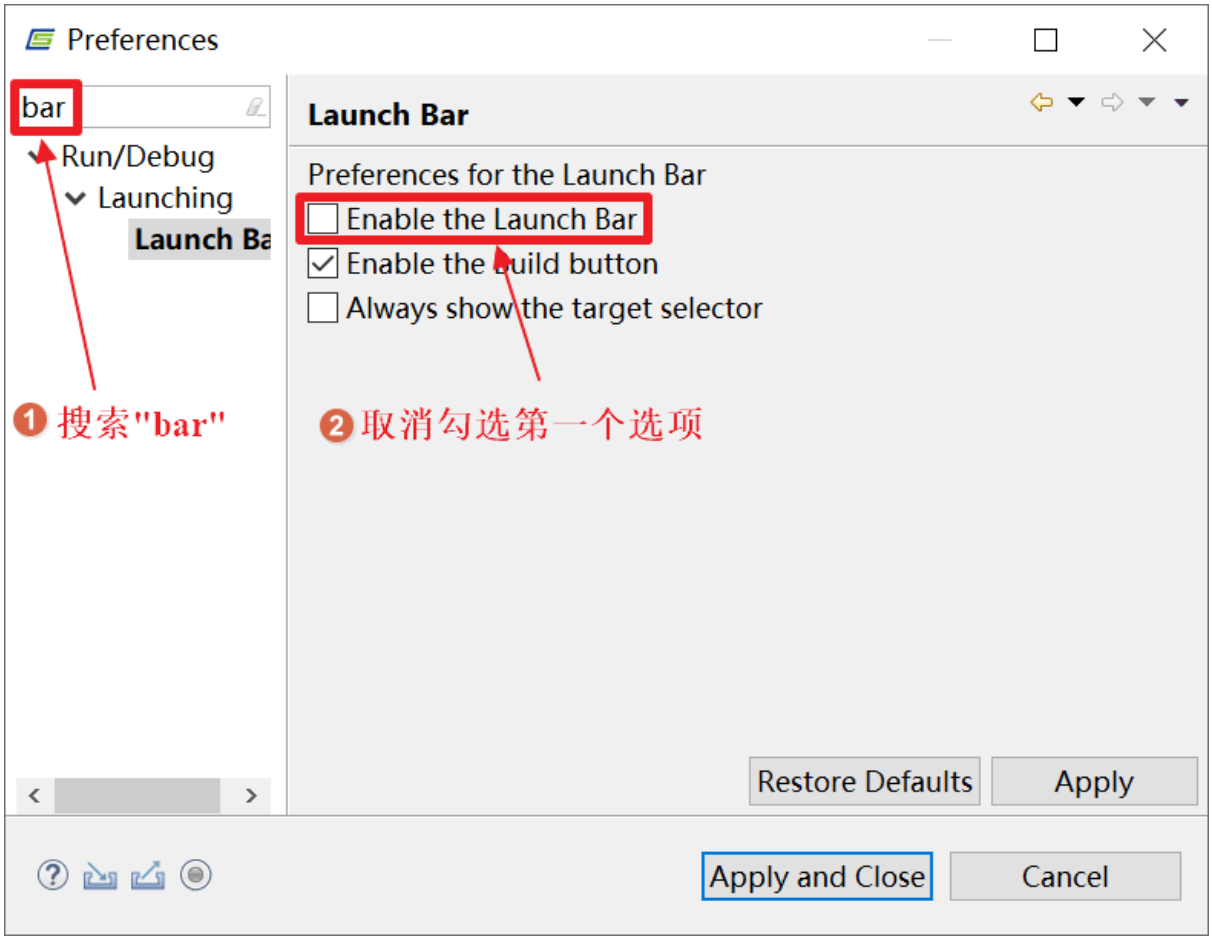


或者将当前视图切换到 C/C++ 视图。



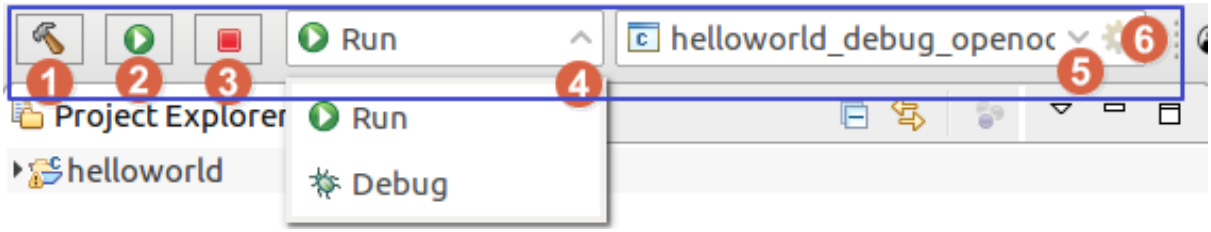
2.12.4 打开/关闭 Launch Bar

Nuclei Studio2022.12 及以后的版本中，默认是开户启了 Launch Bar 功能。打开菜单栏 `Window -> Preferences`，搜索 `bar`，取消勾选第一个选项即可关闭 Launch Bar。同理，想要使用 Launch Bar 功能，请勾选此选项。



2.12.5 使用 Launch Bar

Nuclei Studio 的 Launch Bar（下图中蓝色框内标注部分）是对后边下拉框中选中的调试配置对应的工程进行编译、调试和下载等操作，如需使用此部分请注意，图标是与下拉框中调试配置内容相绑定。下面详细介绍 Launch Bar 各部分功能：



图标 1：编译选中的调试配置对应的工程。此处选中的工程为 5 号下拉框选中的调试配置对应工程，并非 Project Explorer 中选中的工程。

图标 2：运行/调试选中的调试配置对应的工程。此处图标会根据下拉框 4 的内容变化。

图标 3：退出下载/调试模式。

下拉框 4：切换选择下载/调试模式。切换后会改变图标 2 的显示内容。

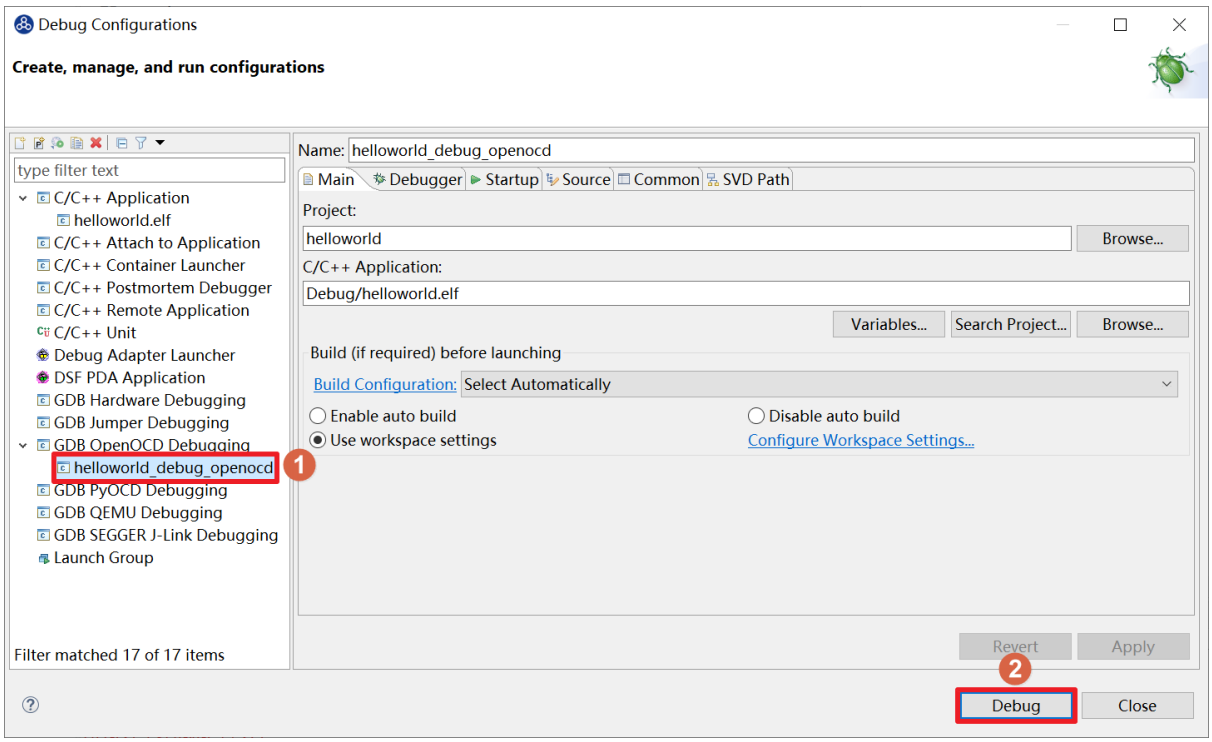
下拉框 5：选中调试配置文件。此处的内容会影响图标 1、2、3 对应的工程。

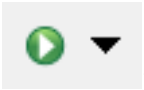
齿轮图标 6：打开当前选中的调试配置页面，可直接进行修改保存。

2.12.6 不使用 Launch Bar 进行运行/调试

对于初次新建的工程，如果不使用 Launch Bar 功能，可以右键工程，选择 Debug As -> Debug Configurations 。在弹窗中选择工程对应的设置选项，这里以 helloworld 工程为例，选择 helloworld_debug_openocd ，之后选择 Debug 开始调试。

同理，初次运行需要右击工程，选择 Run As -> Run Configurations 。在弹窗中选择工程对应的设置选项，之后选择 Run 开始运行。



若当前工程已按以上方式进行过调试或运行操作，可在上方按键中选择  的下拉选项，快速选择对应的工程设置进行调试。对于运行，可以在  的下拉选项，快速选择对应的设置。

2.12.7 Debug 页面查看寄存器

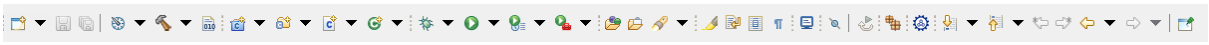
进入 Debug 模式后，可能有些页面并不能第一时间找到，这里以查看寄存器栏目为例。在菜单栏选择 Window -> Show View -> Registers 打开寄存器栏目，在 Register 页面可以通过 Ctrl F 来查找寄存器。同样，如果想查看内存，可以选择 Window -> Show View -> Memory ，想打开调试控制台，可以选择 Window -> Show View -> Debugger Console ，其他页面此处不做过多介绍，请用户自行体验。

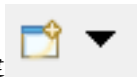
2.12.8 Debug 页面结束进程

在 Debug 页面要退出调试，可点击  退出调试。为防止打开多个进程导致报错，在 Debug 页面，右击 Debug 栏目中的进程，选择 `Terminate / Disconnect All` 关闭所有 Debug 进程。

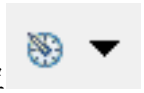
2.12.9 Nuclei Studio 工具栏中各按键功能

在 Nuclei Studio 中，常见的工具栏图标如下，本节将依次介绍各按键的功能。



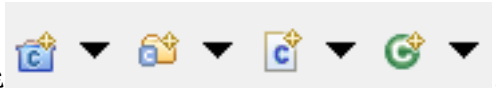
新建按键  ，包括新建各种文件和工程。

保存当前文件  和保存所有未保存文件  。

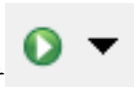
切换编译设置  ，默认只有 Debug 和 Release，用户可自行添加，这里不做介绍。

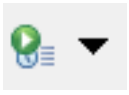
编译工程  ，可以在下拉选项中编译 Project Explorer 中选中的工程。

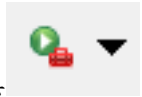
编译所有工程  ，一次性编译所有 Project Explorer 中的工程。

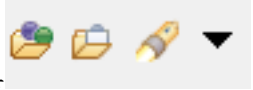
新建功能  ，包含工程创建，目录创建，文件创建和类创建。

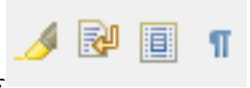
调试  ，可根据下拉选项选择不同调试设置来调试不同的工程。

运行  ，可根据下拉选项选择不同的设置来选择不同的工程。

Profile  ，使用 Profile 功能，暂不支持，此处不做介绍。

用户自定义工具  ，此处不做介绍，用户可自己深入研究。

搜索工具  ，可搜索任务，文本等。

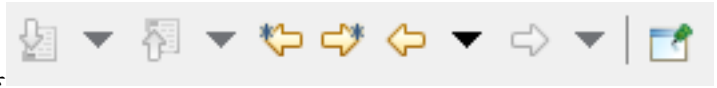
文本阅读工具  ，可以展开或折叠文本等功能。

控制台工具  ，可选择打开各种控制台或串口等。

Debug 用  ，前者跳过所有断点，后者实现 Restart 功能。

查看 CMSIS  ，eclipse 自带功能，暂不支持。

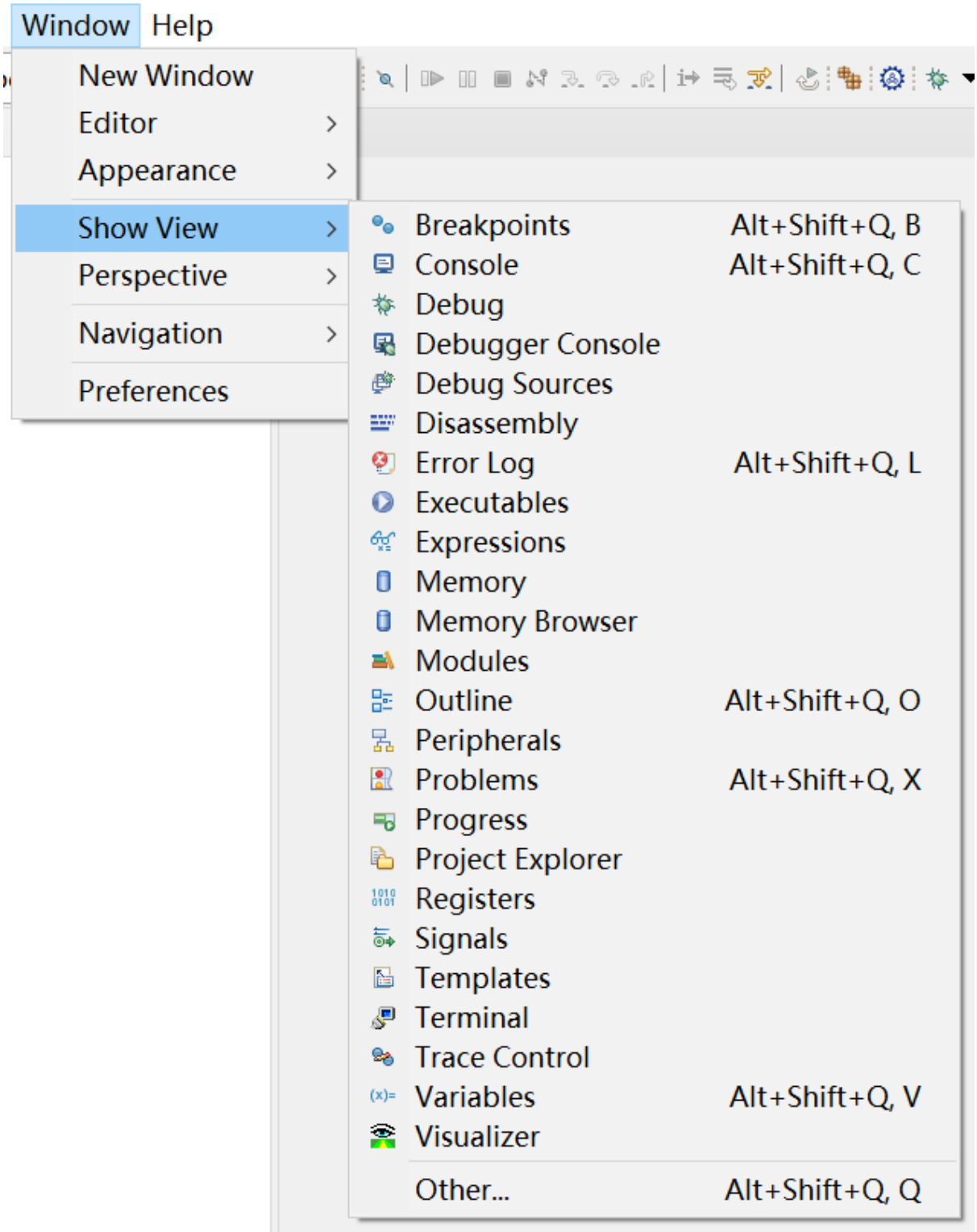
Nuclei SDK 工程设置工具  ，参见 6.6.1。

文本阅读工具  ，可设置快速跳转，具体功能此处不作过多介绍。

2.12.10 显示其他窗口

在 IDE 的右上角  按键可以切换不同的显示模式，常用的是 C/C++ 页面和 Debug 页面。其中，在进入 Debug 模式时，若不在 Debug 页面，会有弹窗提示，此时点击 `Switch` 选项可以切换至 Debug 页面。在菜单栏中选择 `Window -> Perspective -> Open Perspective` 可以快速切换显示窗口。

在不同的窗口下，`Window -> Show View` 显示的内容也不尽相同。在 Debug 窗口中可以选择显示如寄存器（Registers），内存（Memory）和调试控制台（Debugger Console）。

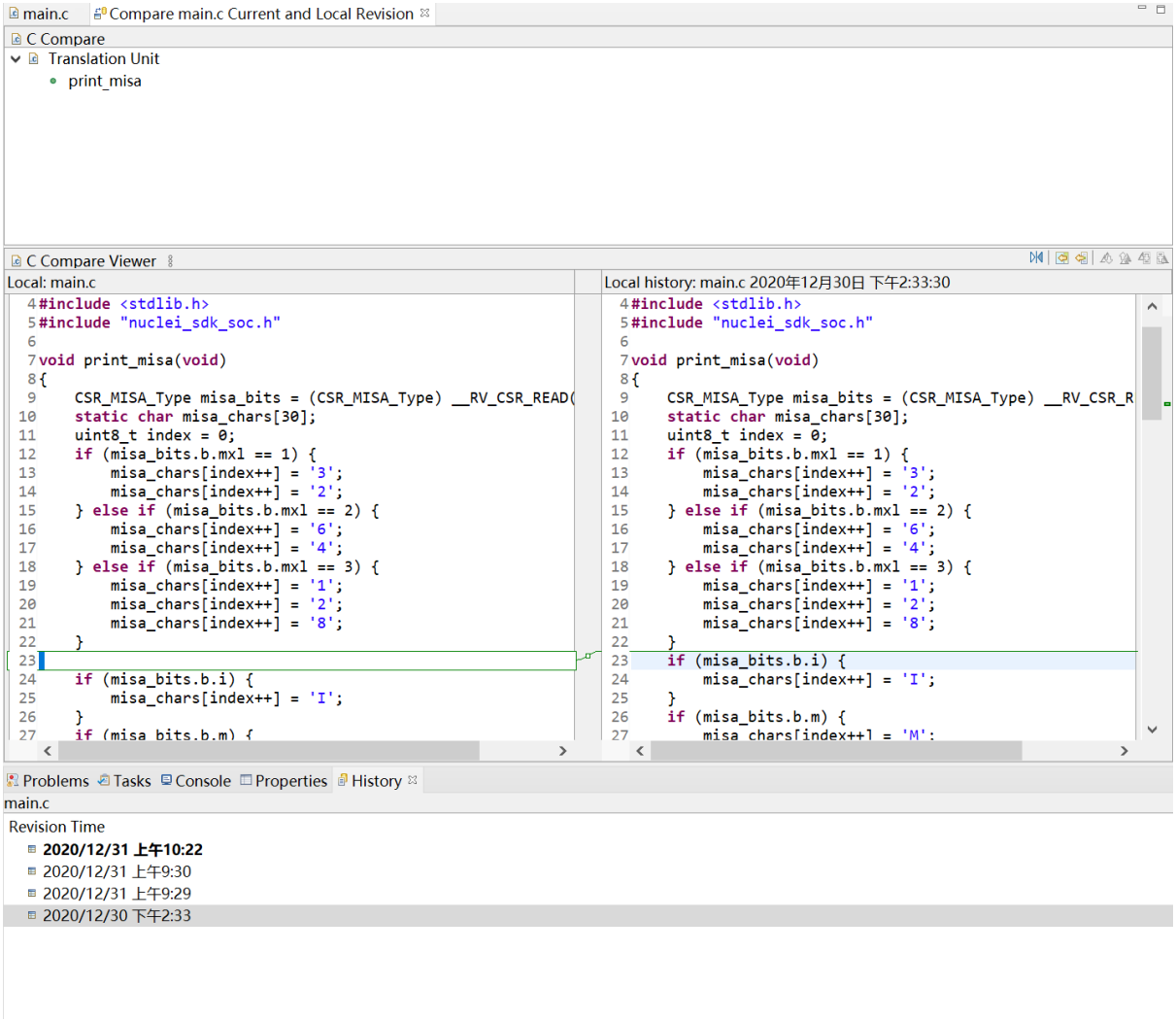

Window
Help
New Window
Editor
Appearance
Show View
Perspective
Navigation
Preferences
Breakpoints Alt+Shift+Q, B
Console Alt+Shift+Q, C
Debug
Debugger Console
Debug Sources
Disassembly
Error Log Alt+Shift+Q, L
Executables
Expressions
Memory
Memory Browser
Modules
Outline Alt+Shift+Q, O
Peripherals
Problems Alt+Shift+Q, X
Progress
Project Explorer
Registers
Signals
Templates
Terminal
Trace Control
Variables Alt+Shift+Q, V
Visualizer
Other... Alt+Shift+Q, Q

2.12.11 恢复默认窗口布局

在菜单栏中选择 Window -> Perspective -> Reset Perspective，在弹窗中选择 Reset Perspective 可以恢复窗口默认布局。

2.12.12 对比历史文件

右击要查看历史的文件，选择 Compare With -> Local History 打开历史记录。在打开的 History 栏目双击选择要对比的文件历史版本，可以查看文件变动历史。



2.12.13 新建工程时可能出现报错

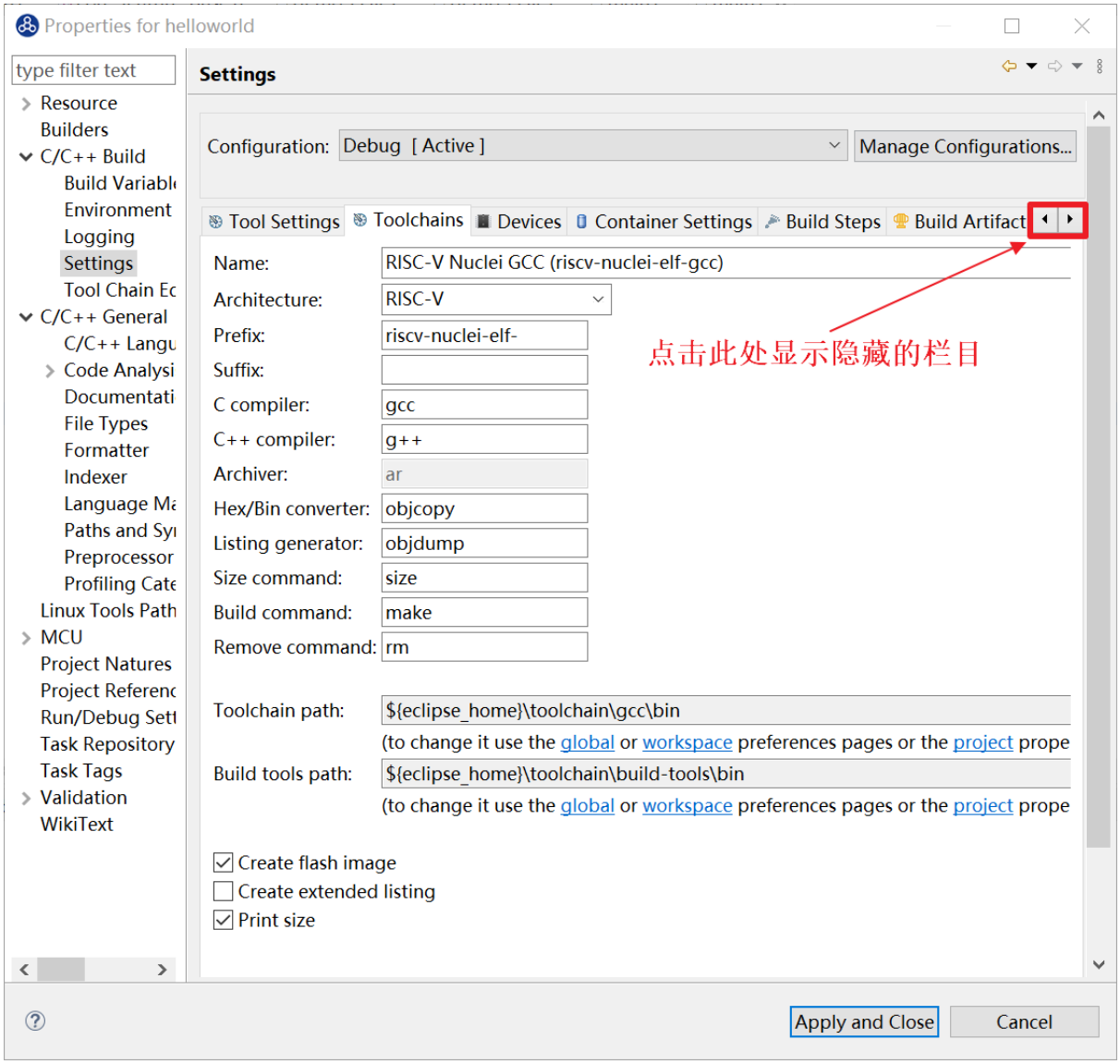
新建工程时 IDE 需要索引整个工程，根据主机性能其所用时间不同。如果主机性能较差，可能会看到索引时产生 error，索引结束后 error 会消失。如出现以上现象，请以编译时是否报错为准。

2.12.14 新增 Include 路径出现缓存

在 include 页面新增路径时，可能会出现缓存的路径内容，此时点击 Workspace 或 File System 选中后可覆盖其内容。

2.12.15 设置页面栏目找不到

有时可能因为弹窗大小，部分设置栏目被隐藏起来，可点击红框中左右方向图标显示隐藏栏目。



2.12.16 开发板下载速度很慢

如果遇到开发板下载速度很慢，甚至出现超时的报错，请切换至使用 USB3.0 接口，如果使用虚拟机开发，也请同时将 USB 接口设置为 3.0。

2.12.17 Linux 环境下多用户使用 Nuclei Studio

如果需要多用户同时使用 Nuclei Studio（不推荐），用户在运行 Nuclei Studio 时首先要打开菜单栏的 `Window->Preferences`。在弹窗中需要修改三个地方：

打开 `MCU->Global OpenOCD Path`，`Executable` 输入 `openocd`，`Folder` 输入 `${eclipse_home}/toolchain/openocd/bin`。

打开 `MCU->Global QEMU Path`，`Executable` 输入 `qemu-system-riscv32`，`Folder` 输入 `${eclipse_home}/tools/qemu`。

打开 `MCU->Global RISC-V Toolchains Paths`，`Default toolchain` 选中 `RISC-V Nuclei GCC`，`Toolchain folder` 输入 `${eclipse_home}/toolchain/gcc/bin`。

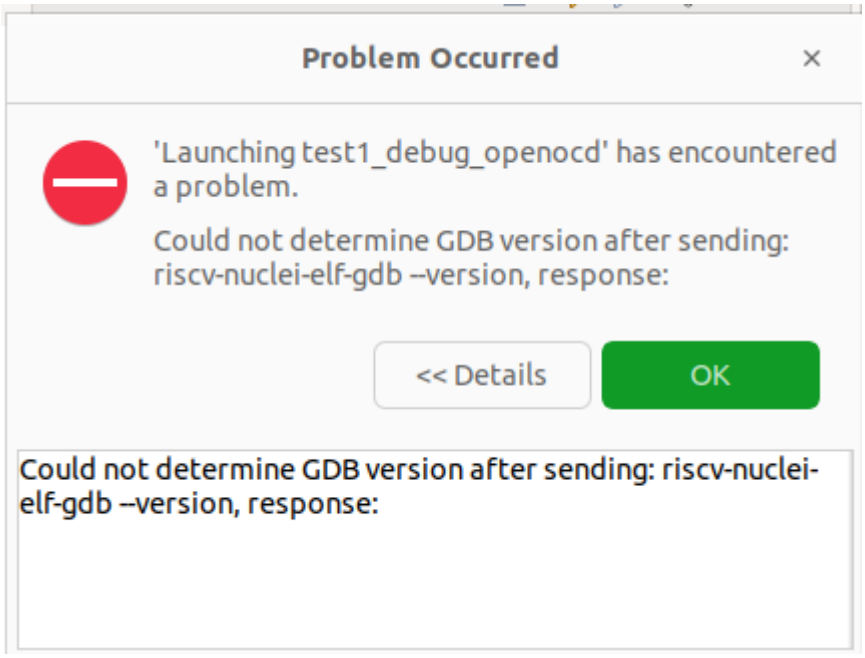
修改后点击 `Apply and Close` 保存并关闭设置弹窗。

2.12.18 设备管理器中识别出两个串口

可能连接设备后在任务管理器中识别出两个串口，其中 COM 编号较大的是串口打印使用，可以使用串口调试助手等工具连接查看打印信息，但是请不要占用 COM 编号较小的串口。

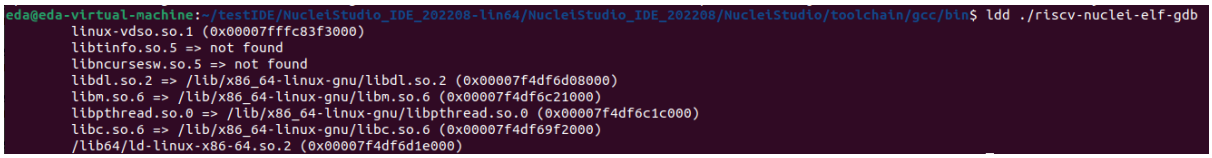
2.12.19 Linux 下使用时报 Could not determine GDB version after sending:riscv-nuclei-elf-gdb version,response: 的错误

第一次在 linux 下使用 Nuclei Studio 时，会报错 `Could not determine GDB version after sending:riscv-nuclei-elf-gdb version,response:`.



riscv-nulcei-elf-gdb 在 Nuclei studio 2023.10 及之后的版本变更为 riscv64-unknown-elf-gdb。

可以使用命令 `ldd $(which riscv-nulcei-elf-gdb)` 查看，依赖缺失



使用命令 `sudo apt install libncursesw5 libtinfo5` 安装相关依赖后，ide 运行正常，具体可以参考 https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/9

2.12.20 在 linux 下使用 QEMU 时报错

例如在 Ubuntu 20.04 上使用 QEMU 时，可能会报错：

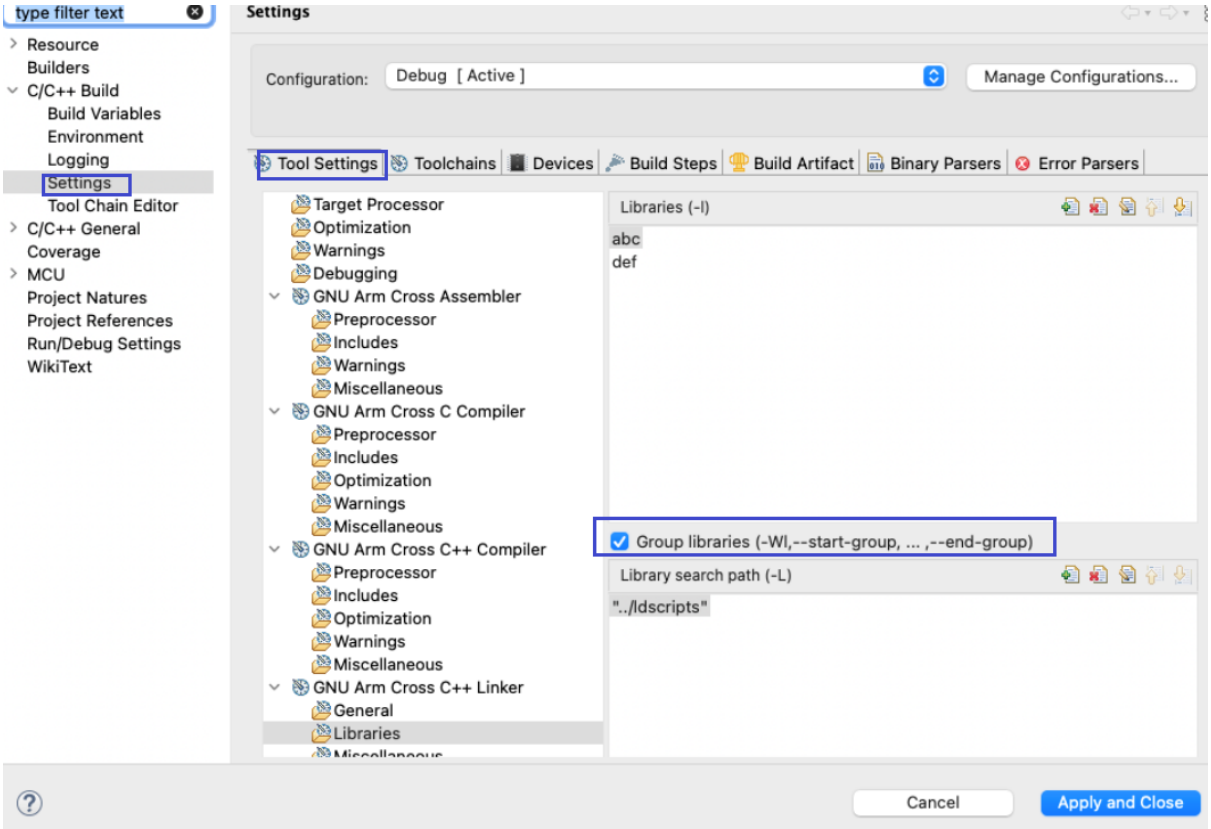
```
error while loading shared libraries: libfdt.so.
↪1: cannot open shared object file: No such file or directory,
```

这是因为缺少 libfdt.so 等依赖所致，只需要在对应版本的 linux 上安装对应的依赖。例如针对 Ubuntu 20.04 可以使用如下命令:

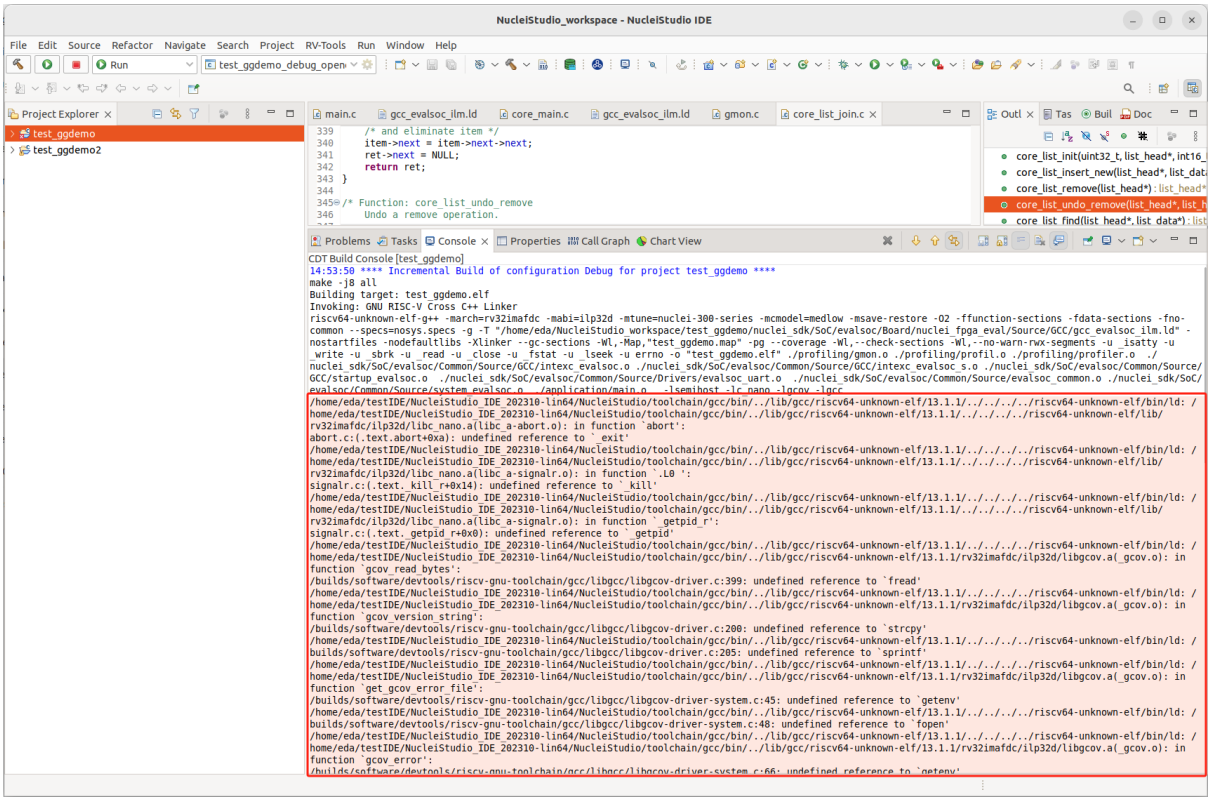
安装 libfdt 等依赖：`sudo apt install libfdt1 libpixman-1-0 libpng16-16 libasound2 libglib2.0-0`

2.12.21 工程编译链接 C 库找不到符号报错

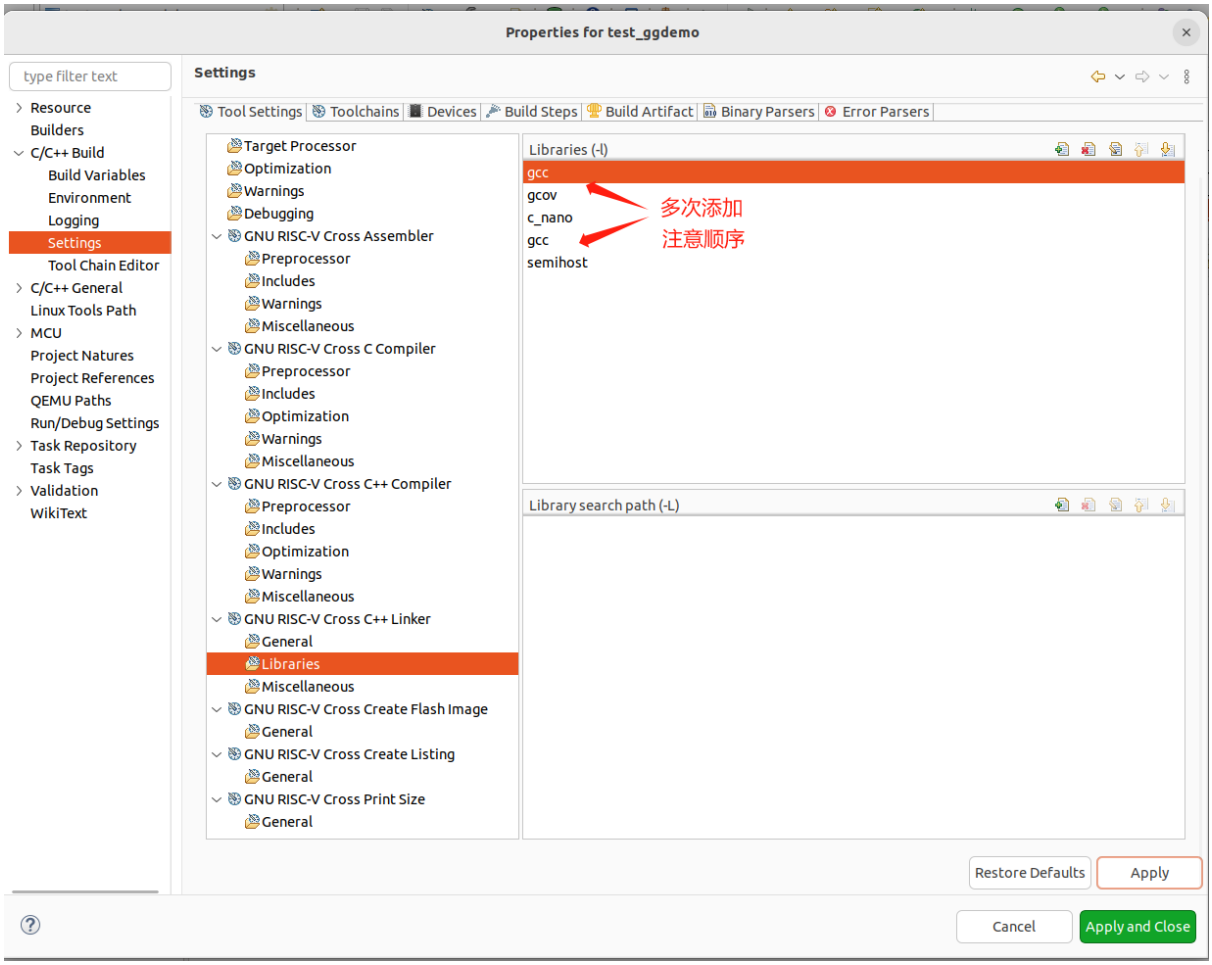
这个问题在 Nuclei Studio 2024.06 可以得到解决，只需要在 Linker -> Libraries 页面勾选 Group Libraries 即可。



因为在对 Libraries 的处理中，没有能很好的处理链接之间的内部依赖关系，当使用的链接之前互相有依赖时，可能会导至编译不通过，详细参见 *https://github.com/eclipse-embed-cdt/eclipse-plugins/issues/592*。解决这个问题，可以有两种办法。



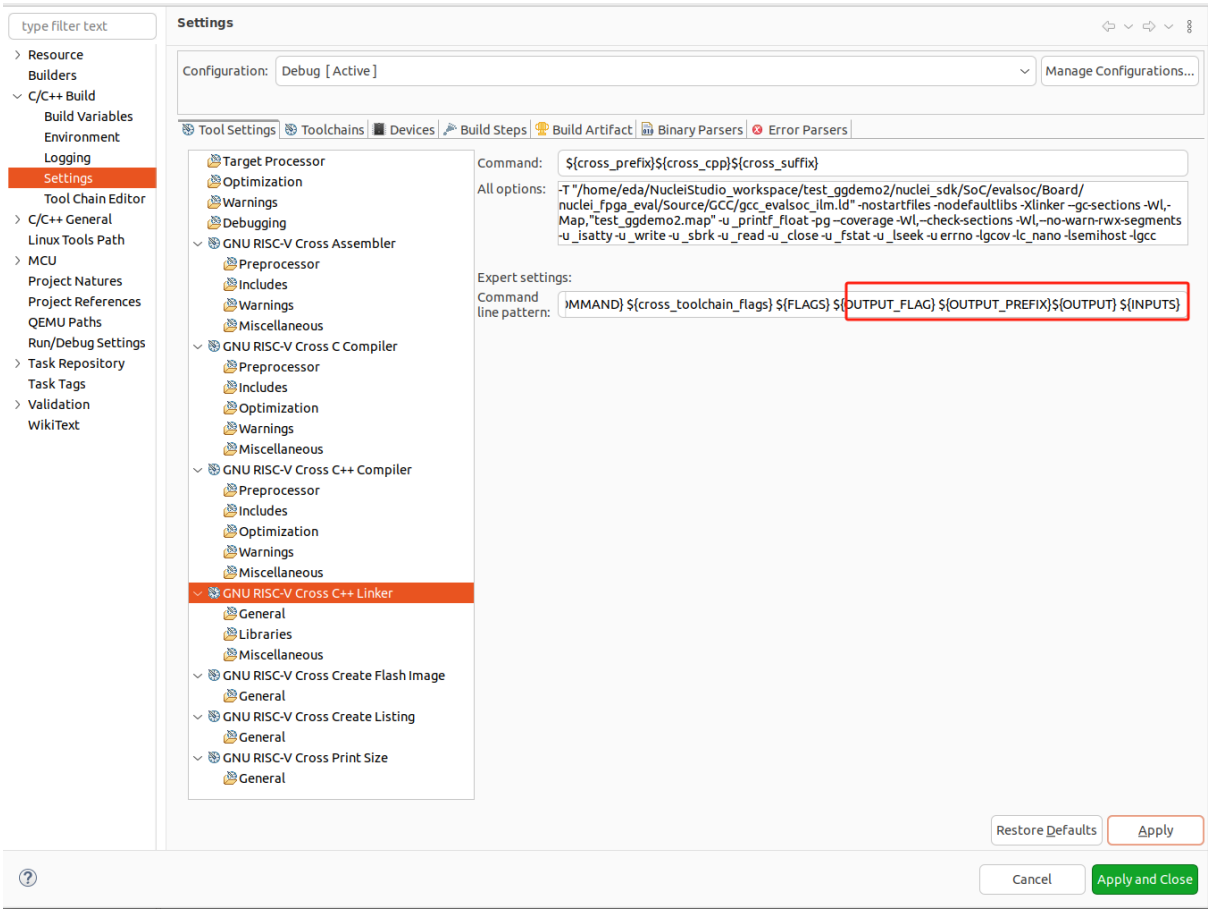
通过调整 Libraries 的顺序或者添加多个链接来实现

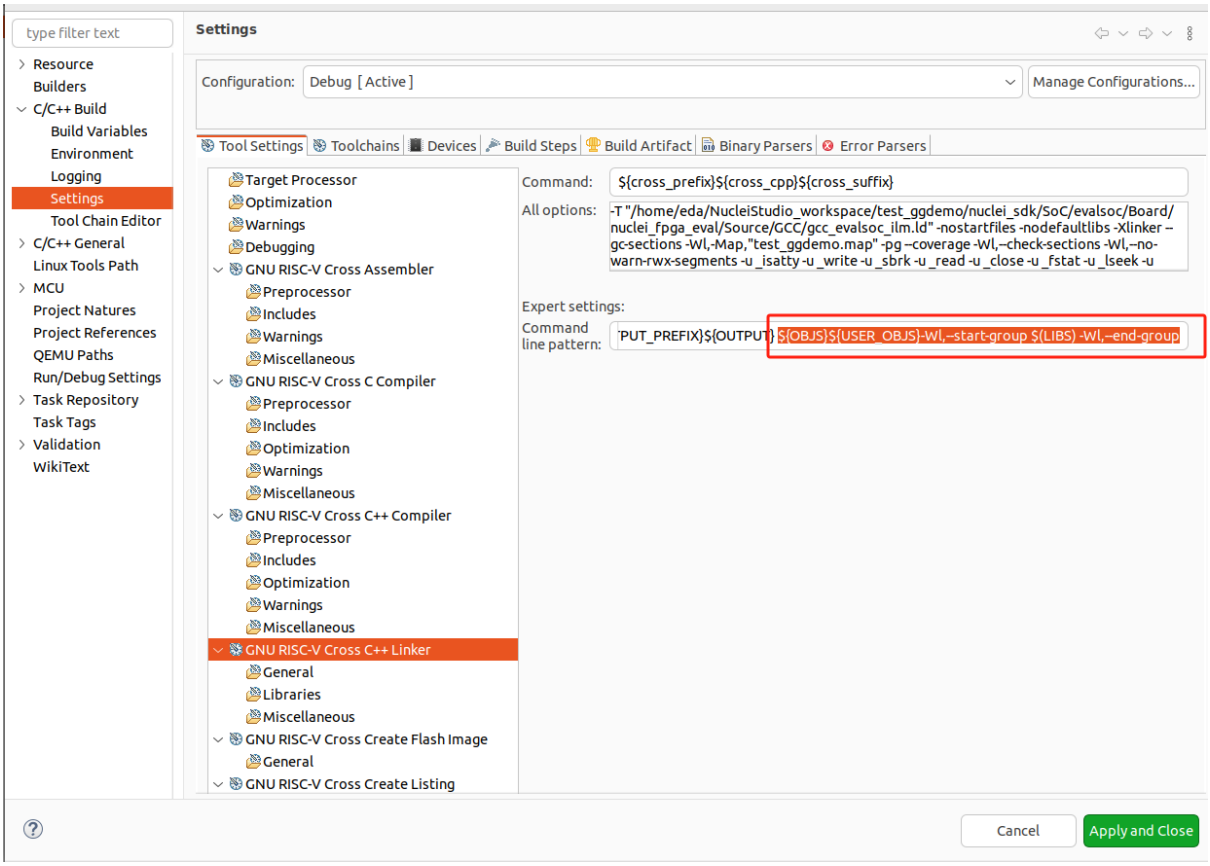


打开 C/C++ Build -> Settings -> Tool Settings ->GUN RISC-V Cross C++ Linker ，并修改 Command line paatern 内容，将其修改为

```
${COMMAND}
${cross_toolchain_flags} ${FLAGS} ${OUTPUT_FLAG}
${OUTPUT_PREFIX}${OUTPUT} ${OBJS}${USER_OBJS} -Wl,--start-group
$(LIBS)
-Wl,--end-group，
```

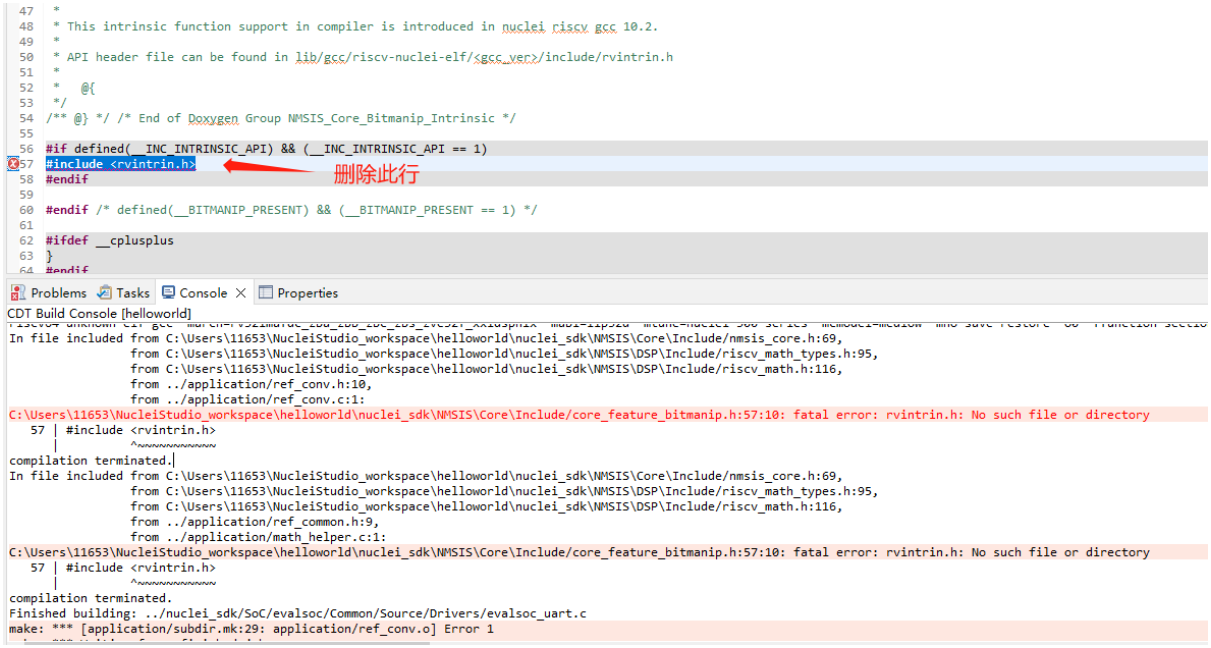
IDE 就会将 Libraries 内的内容以 group 的方式处理。





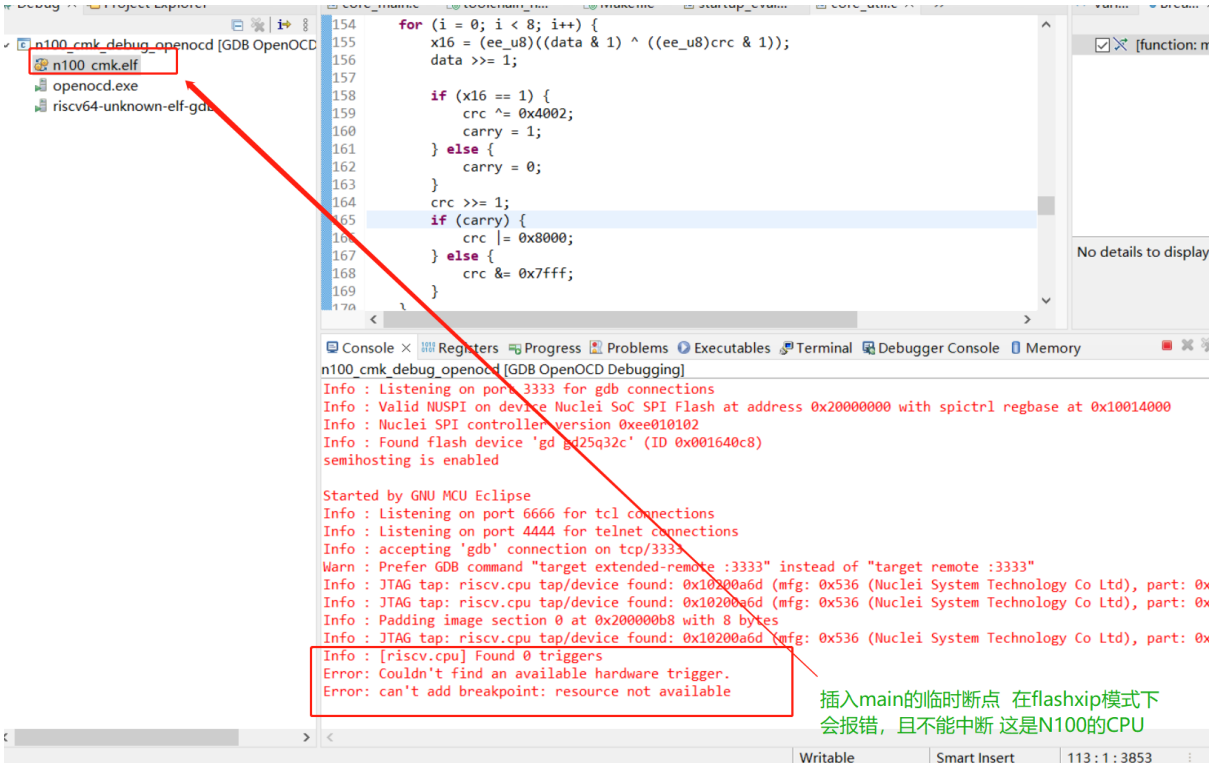
2.12.22 编译工程报错 fatal error: rvintrin.h: No such file or directory

在 Nuclei Studio 2023.10 中，如果使用旧的 sdk 所创建的工程，如果编译时报错 `fatal error: rvintrin.h: No such file or directory`，是因为在 GCC 10 时，工程中 `#include <rvintrin.h>`，而在 GCC 13 中，不需要再 `#include <rvintrin.h>`，只需要删除此行，即可编译通过。



2.12.23 Debug 时报错 Error: Couldn't find an available hardware trigger.

在 Nuclei Studio 环境中，当工程在没有硬件断点的 CPU 硬件上运行时，且选择程序下载到 Flash 中运行以 Nuclei SDK/Nuclei N100 SDK 为例就是 (flash/flashxip DOWNLOAD 模式)，可能会遇到程序 Debug 无法停住的问题，并收到错误提示：`Error: Couldn't find an available hardware trigger`。这是因为程序运行在 Flash 上，软件断点无法被成功写入，而 CPU 上又没有硬件断点可以被使用，从而导致报错。



这种情况下需要将程序编译到 RAM 上才可以支持 IDE 上进行调试（软件断点），如果需要调试则暂时只能通过命令行的方式进行调试。

2.13 其他未注明或者遇到的版本问题

如本文档中有疏漏的地方，请关注 https://www.rvmcu.com/NucleiStudio-faq.html[20] 这里将列出不同版本后续遇到的常见问题。

[20] https://www.rvmcu.com/nucleistudio-faq.html

CHAPTER

THREE

NUCLEI TOOLCHAIN

3.1 GNU Toolchain

3.1.1 About GNU Toolchain

The official toolchain repository is located at https://github.com/riscv-collab/riscv-gnu-toolchain.git. Nuclei maintained toolchain repo is located at https://github.com/riscv-mcu/riscv-gnu-toolchain, and the branch is `nuclei/2024-gcc13`, in which the tools included versions are: gcc13, binutils2.40, gdb13.2, and also have merged some important patches from their upstream, as well as additional support for Nuclei custom extensions and pipelines, etc.

3.1.2 Extensions Support

Standard Extensions

- Basic Extensions

  i, m, a, f, d, c, h, q(Assembly only), zicsr, zifencei, zicond, zawrs, zfh, zfhmin, zmmul, svinval, svnapot.

- Z*Inx Extensions

  zfinx, zdinx, zhinx, zhinxmin.

- CMO Extensions

  zicboz, zicbom, zicbop.

- Bitmanip Extensions

  zba, zbb, zbc, zbs

- Crypto Extensions

  zbkb, zbkc, zbkx, zknd, zkne, zknh, zkr, zks, zksed, zksh, zkt.

- Vector Extensions

  zve32x, zve32f, zve64x, zve64f, zve64d, zvfh, zvfhmin, v.

- Zc Extensions

  zca, zcb, zce, zcf, zcd, zcmp, zcmt(Assembly only).

  - zce = zca + zcb + zcmp + zcmt
  - f + zce = zca + zcb + zcf + zcmp + zcmt
  - f + d + zce = zca + zcb + zcf + zcd + zcmp + zcmt

- Zvb Extensions

  zvbb, zvbc

- Zvk Extensions

  zvkg, zvkned, zvknha, zvknhb, zvksed, zvksh, zvkn, zvknc, zvkng, zvks, zvksc, zvksg, zvkt.

- BFloat 16 Extensions

  Zfbfmin, Zvfbfmin, Zvfbfwma

- Zilsd Extensions

  Zilsd, Zcmlsd

Nuclei Custom Extensions

- Packed SIMD Extension 0.5.4

  xxldsp, xxldspn1x, xxldspn2x, xxldspn3x.

  - xxldsp: P-spec-v0.54 + Nuclei Custom EXPD* instructions
  - xxldspn1x: xxldsp + Nuclei Custom N1
  - xxldspn2x: xxldsp + Nuclei Custom N1 & N2
  - xxldspn3x: xxldsp + Nuclei Custom N1 & N2 & N3

- Xxlcz Extensions

  xxlczpstinc, xxlczbmrk, xxlczbitop, xxlczslet, xxlczabs, xxlczmac, xxlczbri, xxlczbitrev, xxlczgp.

Note: Extensions starting with **x** are generally reserved for manufacturers to customize, and should be placed after extensions starting with **z** when used.

3.1.3 General Options

-march=ISA-string

Generate code for given RISC-V ISA. ISA strings must be lower-case. Examples include `rv64i`, `rv32g`, `rv32e`, and `rv32imaf`. When `-march=` is not specified, use the setting from `-mcpu`. If both `-march` and `-mcpu=` are not specified, the default for this argument is system dependent, users who want a specific architecture extensions should specify one explicitly.

-mabi=ABI-string

Specify integer and floating-point calling convention. ABI-string contains two parts: the size of integer types and the registers used for floating-point types. For example `-march=rv64ifd -mabi=lp64d` means that **long** and **pointers** are 64-bit (implicitly defining **int** to be 32-bit), and that floating-point values up to 64 bits wide are passed in F registers. Contrast this with `-march=rv64ifd -mabi=lp64f`, which still allows the compiler to generate code that uses the F and D extensions but only allows floating-point values up to 32 bits long to be passed in registers; or `-march=rv64ifd -mabi=lp64`, in which no floating-point arguments will be passed in registers.

The default for this argument is system dependent, users who want a specific calling convention should specify one explicitly. The valid calling conventions are: `ilp32`, `ilp32f`, `ilp32d`, `lp64`, `lp64f`, and `lp64d`. Some calling conventions are impossible to implement on some ISAs: for example, `-march=rv32if -mabi=ilp32d` is invalid because the ABI requires 64-bit values be passed in F registers, but F registers are only 32 bits wide. There is also the `ilp32e` ABI that can only be used with the `rv32e` architecture. This ABI is not well specified at present, and is subject to change.

-mcmodel=medlow

Generate code for the medium-low code model. The program and its statically defined symbols must lie within a single 2 GiB address range and must lie between absolute addresses -2 GiB and +2 GiB. Programs can be statically or dynamically linked. This is the default code model.

-mcmodel=medany

Generate code for the medium-any code model. The program and its statically defined symbols must be within any single 2 GiB address range. Programs can be statically or dynamically linked.

The code generated by the medium-any code model is position-independent, but is not guaranteed to function correctly when linked into position-independent executables or libraries.

-mtune=processor-string

Optimize the output for the given processor, specified by microarchitecture or particular CPU name. Permissible values for this option are: `nuclei-100-series`, `nuclei-200-series`, `nuclei-300-series`, `nuclei-600-series`, `nuclei-900-series`, `nuclei-1000-series`, and all valid options for `-mcpu=`.

When `-mtune=` is not specified, use the setting from `-mcpu`, the default is `rocket` if both are not specified.

The `size` choice is not intended for use by end-users. This is used when -Os is specified. It overrides the instruction cost info provided by `-mtune=`, but does not override the pipeline info. This helps reduce code size while still giving good performance.

Optimization Options

-O0
Reduce compilation time and make debugging produce the expected results. This is the default.

-O/-O1
With -O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

-O2
Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to -O, this option increases both compilation time and the performance of the generated code.

-O3
This option turns on all options in -O2, as well as several other optimizations to improve the performance of the object code.

-Os
This optimization option is often used to tell the compiler to reduce the size of the object code as much as possible while maintaining performance. It will remove some optimization strategies that increase the object code size from all options enabled by -O2.

-Ofast
Disregard strict standards compliance. -Ofast enables all -O3 optimizations. It also enables optimizations that are not valid for all standard-compliant programs.

For more information about RISC-V Options used in GCC, please check https://gcc.gnu.org/onlinedocs/gcc-13.1.0/gcc/RISC-V-Options.html

For RISC-V ELF psABI Document, please check https://github.com/riscv-non-isa/riscv-elf-psabi-doc

3.1.4 Libraries

Note:

- `glibc` is used in Linux GNU Glibc toolchain used to compile linux kernel, opensbi, uboot, and linux applications.
- `newlibc` is used in Baremetal or RTOS toolchain, used to compile baremetal or rtos source code, which contains `newlib`, `newlib-nano` and `libncrt`

glibc

glibc stands for GNU C Library which is the standard system C library for all GNU systems. It provides the system API for all programs written in C and C-compatible languages such as C++ and Objective C; the runtime facilities of other programming languages use the C library to access the underlying operating system. This library is only supported on Nuclei linux toolchain, not on Nuclei bare-metal toolchain.

newlib

newlib is written as a glibc replacement for embedded systems. It can be used with no OS (" bare metal ") or with a lightweight RTOS. Newlib is the default library for embedded GCC distributions.

newlib-nano

Newlib-nano is a derivative of the newlib C library for embedded systems. It is smaller and faster than newlib by code and data size reduction through optimization and removal of non-MCU features.

libncrt

`libncrt` is short of Nuclei C Runtime Library, which currently support Nuclei RV32 processor, which is released by Nuclei to reduce c library code size, and improve math library speed, for details, please refer to the user guide located in `gcc\share\pdf\Nuclei C Runtime Library Doc.pdf`

3.1.5 Significant Changes Brought by GCC13 Compared to GCC10

- Instead of using single-letter `bkp` to enable these extensions as we did on gcc10, we split them all into corresponding sub-extensions, for example, `_zba_zkr_zve32f`, please check https://doc.nucleisys.com/nuclei_sdk/develop/buildsystem.html#arch-ext to learn about how to adapt Nuclei SDK to support gcc13 upgraded from gcc10.
- Implement new style of architecture extension test macros: each architecture extension has a corresponding feature test macro, which can be used to test its existence and version information. In addition, we add several custom macros, `__riscv_dsp`, `__riscv_bitmanip`.
- Add new option `-misa-spec=*` to control ISA spec version. This controls the default version of each extensions. The official version is `20191213`, but it is set to `2.2` when configuring nuclei toolchain. The difference between them is that in `20191213` version, `Zicsr` and `Zifencei` are separated from the `i` extension into two independent extensions, and using `-misa-spec=2.2` can avoid incompatible errors when the `Zicsr` and `Zifencei` are not passed to `-march=`. See for details at https://github.com/riscv-collab/riscv-gnu-toolchain/issues/1315
- Support for vector intrinsics as specified in version 0.12 of the RISC-V vector intrinsic specification.
- The toolchain component prefix is `riscv-nuclei-elf-` on gcc10, but is `riscv64-unknown-elf-` on gcc13.
- On gcc10, RISCV intrinsic api heads contain `riscv_vector.h`, `riscv_vector_itr.h`, `rv-intrin.h`, `rvp_intrinsic.h`, but now only `riscv_vector.h`, `rvp_intrinsic.h`, `riscv_nuclei_xlcz.h` are provided in gcc13, if you want to find `b` or `k` intrinsic API, please check https://github.com/riscv/riscv-crypto/blob/main/benchmarks/share/rvintrin.h and https://github.com/riscv/riscv-crypto/blob/main/benchmarks/share/riscv-crypto-intrinsics.h , and for RVV intrinsic API, we support 0.12 in gcc13 now, see https://github.com/riscv-non-isa/rvv-intrinsic-doc/releases/tag/v1.0-rc0
- The version of the libncrt was changed from v2.0.0 to v3.0.0, and libncrt is now split into three parts, ' libncrt ' , ' heapops ' and ' fileops ' , click https://doc.nucleisys.com/nuclei_sdk/develop/buildsystem.html#stdclib to learn about how the newlib/libncrt are used in Nuclei SDK with gcc13.

3.1.6 Install and Setup

Build Toolchain

For more information about how to build a toolchain, see https://github.com/riscv-mcu/riscv-gnu-toolchain/tree/nuclei/2024-gcc13/scripts/toolchain. (Only for Nuclei internal use, no technical support is provided)

Development

The process of user compilation and development can see from https://github.com/riscv-mcu/riscv-gnu-toolchain/blob/nuclei/2024-gcc13/README.md. To get other technical support, please send issues directly to the upstream repository https://github.com/riscv-collab/riscv-gnu-toolchain.

Examples

1. If you choose a core of Nuclei N300FD, then the parameter you pass to ‘ march ’ should be `rv32*fd*`, and ‘ mabi ’ should choose `ilp32d`.
2. If you want to bring the full `B/K/P` extension, then you also need to bring all the subsets of them in the ‘ march ’ . For example, for the `B` extension, the parameter you pass to ‘ march ’ is `_zba_zbb_zbc_zbs`.
3. When using a library, we can tell the linker which library we need to link by using the ‘ -l ’ , for example, `-lc` for newlib-full, `-lc_nano` for newlib-nano. For libncrt, you should pass `--specs=libncrt_xxx.specs` when using gcc. In addition, you need to link extra ‘ fileops ’ and ‘ heapops ’ static libraries during the linking phase by using the ‘ -l ’ , and for the ‘ fileops ’ , you must select one of the three options: ‘ uart ’ , ‘ semi ’ or ‘ rtt ’ , and for the ‘ heapops ’ , you must select one of the three options: ‘ basic ’ , ‘ realtime ’ or ‘ minimal ’ .

3.2 LLVM Toolchain

3.2.1 About LLVM Toolchain

The current llvm toolchain is developed based on the upstream V17.0, and only Nuclei custom CSR is supported.

3.2.2 Extensions Support

Ratified Extensions

- Basic Extensions

  i, m, a, f, d, c, h(Assembly only), zicsr, zifencei, zihintpause(Assembly only), zicond, zawrs(Assembly only), zfh, zfhmin, zmmul, svinval(Assembly only), svnapot(Assembly only), svpbmt.

- Z*inx Extensions

  zfinx, zdinx, zhinx, zhinxmin.

- CMO Extensions(Assembly only)

  zicboz, zicbom, zicbop.

- Bitmanip Extensions

  zba, zbb, zbc, zbs

- Crypto Extensions

  zbkb, zbkc, zbkx, zknd, zkne, zknh, zkr, zks, zksed, zksh, zkt.

- Vector Extensions

  zve32x, zve32f, zve64x, zve64f, zve64d, zvfh, zvfhmin, v.

- Zc Extensions

  zca, zcb, zce, zcf, zcd, zcmp(Assembly only), zcmt(Assembly only).

- Zvb Extensions

  zvbb, zvbc

- Zvk Extensions

  zvkg, zvkned, zvknha, zvknhb, zvksed, zvksh, zvkn, zvknc, zvkng, zvks, zvksc, zvksg, zvkt.

Experimental Extensions

LLVM supports (to various degrees) a number of experimental extensions. All experimental extensions have `experimental-` as a prefix. Listed below:

> smaia, ssaia, zacas, zfa, zfbfmin, zvfbfmin, zvfbfwma, zicond, zihintntl, ztso, zvbb, zvbc, zvkg, zvkn, zvknc, zvkned, zvkng, zvknha, zvknhb, zvks, zvksc, zvksed, zvksg, zvksh, zvkt.

Note: To use an experimental extension from *clang*, you must add `-menable-experimental-extensions` to the command line, and specify the exact version of the experimental extension you are using. To use an experimental extension with LLVM ' s internal developer tools (e.g. *llc*, *llvm-objdump*, *llvm-mc*), you must prefix the extension name with `experimental-`.

3.2.3 General Options

-march=<cpu>

> Specify that Clang should generate code for a specific processor family member and later. For example, if you specify -march=i486, the compiler is allowed to generate instructions that are valid on i486 and later processors, but which may not exist on earlier ones.

-mcmodel=<arg>

> -mcmodel=medany (equivalent to -mcmodel=medium), -mcmodel=medlow (equivalent to -mcmodel=small)

-mcpu=help, -mtune=help

> Print out a list of supported processors for the given target (specified through `--target=<architecture>` or `-arch <architecture>`). If no target is specified, the system default target will be used.

Optimization Option

-O0
> Means "no optimization": this level compiles the fastest and generates the most debuggable code.

-O/-O1
> Somewhere between -O0 and -O2.

-O2
> Moderate level of optimization which enables most optimizations.

-O3
> Like -O2, except that it enables optimizations that take longer to perform or that may generate larger code (in an attempt to make the program run faster).

-Ofast
: Enables all the optimizations from -O3 along with other aggressive optimizations that may violate strict compliance with language standards.

-Os
: Like -O2 with extra optimizations to reduce code size.

For more about RISC-V options used by LLVM toolchain, please check https://releases.llvm.org/17.0.1/docs/RISCVUsage.html

3.2.4 Install and Setup

More information on building and running LLVM, see https://llvm.org/docs/GettingStarted.html#getting-the-source-code-and-building-llvm

CHAPTER

FOUR

NUCLEI OPENOCD

4.1 About OpenOCD

4.1.1 Repository and Doc

openocd

- github riscv-openocd: https://github.com/riscv-mcu/riscv-openocd/tree/nuclei/2024.06
- gitee riscv-openocd: https://gitee.com/riscv-mcu/riscv-openocd/tree/nuclei/2024.06

openflashloader

Note: OpenOCD Flashloader is developed to support various different customized flash programming without modify openocd and rebuilt it, please check the source code repo ' s doc.

- github openflashloader: https://github.com/riscv-mcu/openflashloader
- gitee openflashloader: https://gitee.com/riscv-mcu/openflashloader

openocd doc path: `openocd/doc/pdf/openocd.pdf`

4.2 How to Use

4.2.1 How to determine the version of OpenOCD

- start a cmd(Windows)/terminal(Linux)
- run the *openocd -v* command

```
nuclei@nuclei-v:~$ openocd -v
Open On-Chip Debugger 0.11.0+dev-02424-g787e48e66 (2022-12-29-08:49)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
nuclei@nuclei-v:~$
```

Note: git commit id: 787e48e66

compile date: (2022-12-29-08:49)

4.2.2 Start OpenOCD

Frequently used command line parameters

parameter	description
-v	display version info
-d	set debug level to 3
-d0	error messages only
-d1	error warning
-d2	error warning info (default)
-d3	error warning info debug
-d4	error warning info debug low-level-debug
-f file	use configuration file
-s dir	dir to search for config files
-l logfile	redirect log output to logfile

4.3 Nuclei Customized Features

Display CPU information

Usually we don ' t know what instruction sets a CPU supports, what functional components it contains, and the various configurations of the components. The only way to get the desired information is to read the corresponding CSR registers and then do the math. It would be a pain in the ass to do this for every function point that needs to be known.

This command is designed to solve this problem by automatically reading the CSRs and doing the cpu feature probing, and then formatting the output.

```
nuclei cpuinfo
```

Nuspi(nuclei spi) driver

Nuclei ' s SPI controller, used in Nuclei RISC-V fpga evaluation board and other boards.

```
flash bank name nuspi base size chip_width bus_width target spi_base
[simulation]
```

Custom driver and open-flashloader

Custom exists for compatibility with any SPI controller and any Flash. It also needs to be used in conjunction with openflashloader to achieve the desired results.

```
flash bank name custom base size chip_width bus_width target spi_base
flashloader_path [simulation] [sectorsize=]
```

Nuclei Customized CSRs

Nuclei released openocd supports a number of nuclei customized CSRs, please check here https://github.com/riscv-mcu/riscv-openocd/blob/nuclei/2024.06/src/target/riscv/encoding.h#L3109-L3223

Nuclei embedded trace

Note: Still in experiment stage, not for production usage.

Some Nuclei cpus are equipped with trace support, which permits examination of the instruction activity. Trace activity is controlled through an Embedded Trace(Etrace) Module on the core ' s scan chains. The following commands are for etrace.

Currently, we only implemented RISC-V ETrace Instruction Trace in CPU, Data Trace not yet ready.

`nuclei etrace config etrace-addr buffer-addr buffer-size wrap`

This command is used to initialize Etrace and configure related parameters.

`nuclei etrace enable`

This command triggers the Etrace enable signal by setting the Core internal trigger.

`nuclei etrace disable`

This command triggers the Etrace disable signal by setting the Core internal trigger.

`nuclei etrace start`

This command is used to enable Etrace data collection.

`nuclei etrace stop`

This command is used to disable Etrace data collection.

`nuclei etrace dump filename`

This command is used to dump the data captured by Etrace.

`nuclei etrace clear`

This command is used to clear the read and write pointers for Etrace.

`nuclei etrace info`

This command displays the current Etrace status.

You can also use ETrace feature in Nuclei Studio IDE, please check its documentation for more details.

Nuclei Debug Map Feature

The debug map for each hart is automatically read and printed during OpenOCD startup, or you can read the debug map at runtime with the `examine_cpu_core` command.

About the detailed nuclei debug map feature, please contact with our AE for more documentation.

`nuclei expose_cpu_core`

Configure a list of index for *nuclei_examine_cpu_core* to expose in this must be executed before `init`.

`nuclei examine_cpu_core`

Return the 64-bit value read from `dm-custom1` and `dm-custom2` value = `dm-custom2 << 32 + dm-custom1`.

Init resethalt command

In practice, usually encountered due to software problems caused by the CPU stuck, then the debugger will not be connected to the development board, only to the development board power off. If your code is running in flash, powering down the board will not solve the problem. resethalt is designed to solve this problem.

```
init resethalt
```

FTDI nscan1_mode command

Enable or disable Nuclei CJTAG mode. Usage is the same as `ftdi oscan1_mode`.

```
ftdi nscan1_mode on|off
```

4.4 About the configuration file

The openocd configuration file is used to configure how to connect to the development board ' s window through the Debug interface. nuclei provides an example of the openocd configuration file, which can be modified based on the example.

Here we take example using Nuclei HBird Debugger(FTDI based) as to explain this openocd configuration file.

Here is an working example for openocd configuration file https://github.com/Nuclei-Software/nuclei-sdk/blob/master/SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg

Modify debugger rate

`adapter_khz 1000` or `adapter speed 1000`

Select debugger interface

```
adapter driver ftdi
ftdi vid_pid 0x0403 0x6010

transport select jtag

ftdi layout_init 0x0008 0x001b
ftdi layout_signal nSRST -oe 0x0020 -data 0x0020
ftdi layout_signal TCK -data 0x0001
ftdi layout_signal TDI -data 0x0002
ftdi layout_signal TDO -input 0x0004
ftdi layout_signal TMS -data 0x0008
ftdi layout_signal JTAG_SEL -data 0x0100 -oe 0x0100
```

The above code are used to select fdti debugger, the ftdi chip pid/vid must match selected id, transport is selected as JTAG, and ftdi layout is setup to match HBird Debugger hardware settings.

Modify debugger mode

There are two debugging modes JTAG and cJTAG.

- JTAG <-> `ftdi nscan1_mode off`
- cJTAG <-> `ftdi nscan1_mode on`

Describe the JTAG link

- single core

```
set _CHIPNAME0 riscv0
jtag newtap $_CHIPNAME0 cpu -irlen 5 -expected-id 0x10900a6d

set _TARGETNAME0 $_CHIPNAME0.cpu
target create $_TARGETNAME0 riscv -chain-position $_TARGETNAME0 -coreid 0
```

- smp system

```
set _CHIPNAME0 riscv0
jtag newtap $_CHIPNAME0 cpu -irlen 5 -expected-id 0x10900a6d

set _TARGETNAME0 $_CHIPNAME0.cpu
target create $_TARGETNAME0.0 riscv -chain-position $_TARGETNAME0 -coreid 0 -rtos␣
↪hwthread
target create $_TARGETNAME0.1 riscv -chain-position $_TARGETNAME0 -coreid 1
target create $_TARGETNAME0.2 riscv -chain-position $_TARGETNAME0 -coreid 2
target smp $_TARGETNAME0.0 $_TARGETNAME0.1 $_TARGETNAME0.2
```

- amp system

```
set _CHIPNAME0 riscv0
jtag newtap $_CHIPNAME0 cpu -irlen 5 -expected-id 0x10900a6d

set _CHIPNAME1 riscv1
jtag newtap $_CHIPNAME1 cpu -irlen 5 -expected-id 0x10300a6d

set _TARGETNAME0 $_CHIPNAME0.cpu
target create $_TARGETNAME0 riscv -chain-position $_TARGETNAME0 -coreid 0

set _TARGETNAME1 $_CHIPNAME1.cpu
target create $_TARGETNAME1.0 riscv -chain-position $_TARGETNAME0 -coreid 0 -rtos␣
↪hwthread
target create $_TARGETNAME1.1 riscv -chain-position $_TARGETNAME0 -coreid 1
target smp $_TARGETNAME1.0 $_TARGETNAME1.1
```

Note:

- `-rtos hwthread`

OpenOCD includes a pseudo RTOS called hwthread that presents CPU cores (“ hardware threads ”) in an SMP system as threads to GDB. With this extension, GDB can be used to inspect the state of an SMP system in a natural way. After halting the system, using the GDB command info threads will list the context of each active CPU core in the system. GDB ’ s thread command can be used to switch the view to a different CPU core. The step and stepi commands can be used to step a specific core while other cores are free-running or remain halted, depending on the scheduler-locking mode configured in GDB.

Describe the workarea

workarea is mainly used to speed up certain operations, such as reading and writing large chunks of memory, running small program fragments, reading and writing flash, and so on.

```
$_TARGETNAME0 configure -work-area-phys 0x08000000 -work-area-size 0x10000 -work-
↪area-backup 1
```

Note: The workarea should be a readable, writable, and executable area of memory.

`0x08000000` workarea base address, modified according to the actual situation.

`0x10000` workarea size of byte, modified according to the actual situation.

Describe the nor flash

```
set _FLASHNAME0 $_CHIPNAME0.flash
flash bank $_FLASHNAME0 nuspi 0x20000000 0 0 0 $_TARGETNAME0.0 0x10180000
```

Note: `nuspi` openocd flash drivers type, modified according to the actual situation.

`0x20000000` qspi-xip address, modified according to the actual situation.

`0x10180000` qspi controller base address, modified according to the actual situation.

Connect to the specified debugger

When there is more than one debugger in a debugging environment, we need to connect to specify the debugger, in this case you can use the following command to specify.

```
ftdi_serial FT4YR31I
```

How to set up gdb/telnet/tcl ports

openocd provides three kinds of debugging service ports are gdb/telnet/tcl, choose the appropriate service according to the situation, and set the port number of the corresponding service by the following command.

```
gdb_port 3333
telnet_port 4444
tcl_port 6666
```

Note: The above shows the default port number, you are free to change the port number if it is free. Of course we can also disable the port numbers we don ' t need, it ' s easy just change the port number to *disable*.

semihosting

OpenOCD also supports the ARM semihosting feature, use the following command to enable it.

```
arm semihosting enable
```

For more detailed information about how to use openocd, please check the `openocd.pdf` distributed in openocd release.

4.5 Frequently asked questions

There are a few more FAQs please see:

- Github: https://github.com/riscv-mcu/riscv-openocd/wiki
- Gitee: https://gitee.com/riscv-mcu/riscv-openocd/wikis

4.6 Low-cost debugger solution

We also provided a low cost mcu solution to debug RISC-V CPU, which support JTAG and cJTAG, please check the following repo to learn more about it, and it is also supported in Nuclei Studio.

Nuclei Dlink: https://github.com/Nuclei-Software/nuclei-dlink

CHAPTER

FIVE

NUCLEI QEMU

5.1 About Nuclei QEMU

Nuclei QEMU is developed based on QEMU version 8.0, supporting the machine features of Nuclei Demosoc and Evalsoc. In terms of extensions, it supports RVV 1.0, Nuclei DSP extension, ZC extension (RISC-V Code Size Reduction), Nuclei Xlcz extension, as well as features like the B extension, K extension, and Nuclei Nice instructions.

If you want to access the code of Nuclei QEMU, you can visit our opensource Nuclei QEMU Github Repository[21].

5.2 Design and Architecture

Nuclei QEMU has several types of parameters that can be configured. You can enter `qemu-system-riscv32 --help` to view the parameters that can be configured in Nuclei QEMU.

Nuclei QEMU supports two main programs: `qemu-system-riscv32` and `qemu-system-riscv64`. `qemu-system-riscv32` is used to support 32-bit programs, while `qemu-system-riscv64` supports 64-bit programs.

Nuclei QEMU supports two machine types: `nuclei_evalsoc` and `nuclei_demosoc`. Among them, `nuclei_demosoc` will be deprecated in future versions, and currently only `nuclei_evalsoc` is supported.

The following image shows the relevant information of the Nuclei CPU types supported by Nuclei QEMU.

5.3 Description of Parameters

Nuclei QEMU adds some custom features and functionalities based on the original capabilities of qemu. If you want to learn more about the usage of qemu, you can refer to the documentation at https://www.qemu.org/docs/master/.

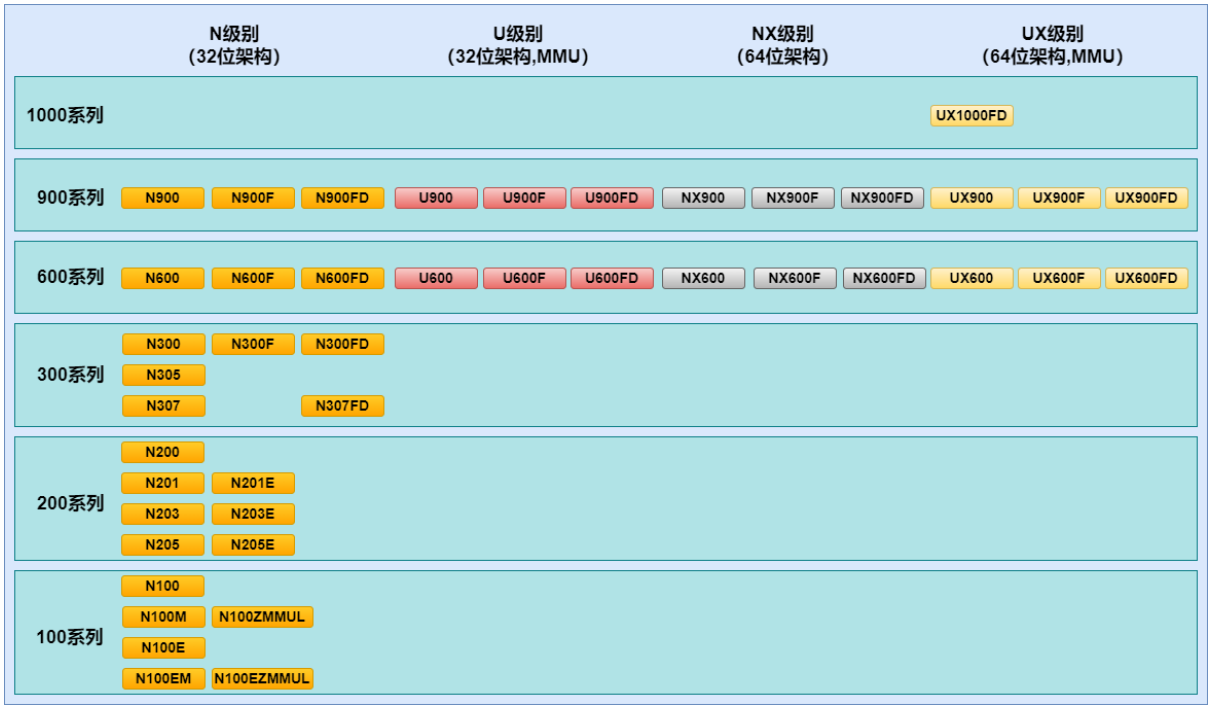
This is an example of a fully functional parameter for Nuclei QEMU: `qemu-system-riscv32 -M nuclei_evalsoc,download=ddr,soc-cfg=evalsoc.json,debug=1 -cpu nuclei-n300fd,ext=_v_xxldsp,vlen=128,elen=64,s=true -m 512M -smp 1 -icount shift=0 -nodefaults -nographic -serial stdio -kernel dhrystone.elf`.

Let ' s describe the meaning of this complete command:

- `-M nuclei_evalsoc,download=ddr,soc-cfg=evalsoc.json,debug=1`:

  `-M` represents `machine`, which means selecting the type of machine. Currently, Nuclei QEMU has added `nuclei_demosoc` (which will be deprecated in future versions) and `nuclei_evalsoc` to the existing options. This option must exist.

[21] https://github.com/riscv-mcu/qemu/tree/nuclei/8.0



`download=` is used to choose the download mode, and currently, it supports four download modes: `sram, flashxip, flash, ilm, and ddr`.If this parameter is not present, the default value is `flashxip`.

`soc-cfg=` is an optional option to pass dynamic modifications to the initial configuration of the machine with a json file. If this parameter is not set, the default value of qemu will be used .Here is an example:

```
{
    "general_config": {
        "ddr": {
                "base":"0x70000000",
                "size":"2G"
        },
        "ilm": {
                "base":"0x90000000",
                "size":"0x100000"
        },
        "dlm": {
                "base":"0xA0000000",
                "size":"0x100000"
        },
        "sram": {
                "base":"0xB0000000",
                "size":"0x10000000"
        },
        "norflash": {
                "base":"0x30000000",
                "size":"32M"
        },
        "uart0": {
                "base":"0x20013000",
                "irq":"34"
        },
        "uart1": {
                "base":"0x20023000",
                "irq":"35"
        },
        "qspi0": {
```

(continues on next page)

(continued from previous page)

```
                "base":"0x20014000",
                "irq":"36"
        },
        "qspi2": {
                "base":"0x20034000",
                "irq":"37"
        },
        "iregion": {
                "base":"0x1000000"
        },
        "cpu_freq":"50000000",
        "timer_freq":"32768",
        "irqmax":"100"
    },
    "download": {
        "ilm": {
                "startaddr":"0x90000000"
        },
        "flashxip": {
                "startaddr":"0x30000000"
        },
        "flash": {
                "startaddr":"0x30000000"
        },
        "sram": {
                "startaddr":"0xB0000000"
        },
        "ddr": {
                "startaddr":"0x70000000"
        }
    }
}
```

general_config : mainly used to configure the board resource or chip base address

base: module base address, only support hex format

size: module size, support hex, dec, size string format

irq: peripheral interrupt id, dec format

download: firmware startup address

The irq peripheral interrupt id is equal to hardware interrupt wire connect number plus one, users should follow this rule when configuring irq.

IRQ_HW_ID	PLIC Interrupt ID	Source
32	33	uart0
34	35	qspi0
35	36	qspi1
36	37	qspi2

In the above script, if there is no **download startaddr** information, the program entry will be the start address of the address range relative to the download mode. For example, when `download=ilm`, if the following configuration is not in the script,

```
"download": {
        "ilm": {
                "startaddr":"0x90000000"
        }
```

then the ilm base in **general_config** will be used as the program start address by default.

```
"general_config": {
     "ilm": {
              "base":"0x90000000",
              "size":"0x100000"
     }
```

Other configurations follow this rule as well.

Note: In the **general_config** JSON configuration script, the **base** attribute must coexist with either **size** or **irq**, and the format requires **base** to be written first, followed by either **size** or **irq**.

`debug=1` list the start address of the current device ' s peripherals and memory distribution information or irq info for debugging purposes. It is generally not recommended to enable this feature under normal circumstances.

- `-cpu nuclei-n300fd,ext=_v_xxldsp,vlen=128,elen=64,s=true`:

  Using the `-cpu` option, `nuclei-n300fd` represents the selectable CPU type for Nuclei, and the complete list of types can be referred to in the diagrams within the `Design and Architecture` section. This operation is necessary.

  `ext=` This parameter is optional, used to pass different riscv extension, The way to enable different extensions is to add them inside it, for example, `xxldsp` represents enable the nuclei DSP extension, `v` represents enable RISC-V V-Extension, When enabling multiple extensions, they are connected through _. Currently, Nuclei QEMU supports the following common RISC-V instruction set extension types:

Extension	Functionality
v	RISC-V V-Extension
h	RISC-V H-Extension
zicbom	RISC-V Zicbom Extension
zicboz	RISC-V Zicboz Extension
zicond	RISC-V Zicond Extension
zicsr	RV32/RV64 Zicsr Standard Extension
zifencei	RV32/RV64 Zifencei Standard Extension
zihintpause	ZiHintPause extension
zilsd	Zilsd extension (RV32 ONLY)
zcmlsd	Zcmlsd extension (RV32 ONLY)
zawrs	Zawrs extension
zfh	Zfh Extension
zfa	Zfa Extension
zfhmin	Zfhmin Extension
zfinx	Zfinx Extension
zdinx	Zdinx Extension
zca	RISC-V ZC* Extension
zcb	RISC-V ZC* Extension
zcf	RISC-V ZC* Extension
zcd	RISC-V ZC* Extension
zce	RISC-V ZC* Extension
zcmp	RISC-V ZC* Extension
zcmt	RISC-V ZC* Extension
zba	RISC-V Bitmanipulation Extension
zbb	RISC-V Bitmanipulation Extension
zbc	RISC-V Bitmanipulation Extension
zbkb	RISC-V Bitmanipulation Extension
zbkc	RISC-V Bitmanipulation Extension

continues on next page

Table 5.1 – continued from previous page

Extension	Functionality
zbkx	RISC-V Bitmanipulation Extension
zbs	RISC-V Bitmanipulation Extension
zk	RISC-V Scalar Crypto Extension
zkn	RISC-V Scalar Crypto Extension
zknd	RISC-V Scalar Crypto Extension
zkne	RISC-V Scalar Crypto Extension
zknh	RISC-V Scalar Crypto Extension
zkr	RISC-V Scalar Crypto Extension
zks	RISC-V Scalar Crypto Extension
zksed	RISC-V Scalar Crypto Extension
zksh	RISC-V Scalar Crypto Extension
zkt	RISC-V Scalar Crypto Extension
zve32x	RISC-V V-Extension
zve32f	RISC-V V-Extension
zve64x	RISC-V V-Extension
zve64f	RISC-V V-Extension
zve64d	RISC-V V-Extension
zvfh	RISC-V V-Extension
zvfhmin	RISC-V V-Extension
zhinx	Zhinx Extension
zhinxmin	Zhinxmin Extension
smaia	Smaia Extension
ssaia	Ssaia Extension
sscofpmf	Sscofpmf Extension
sstc	Sstc Extension
svadu	Svadu Extension
svinval	Svinval Extension
svnapot	Svnapot Extension
svpbmt	Svpbmt Extension
xxldsp	Nuclei DSP Extension based on P-ext 0.5.4 + default 8 EXPD instructions
xxldspn1x	Xxldsp + Nuclei N1 extension
xxldspn2x	Xxldspn1x + Nuclei N2 extension
xxldspn3x	Xxldspn2x + Nuclei N3 extension
xxlcz	Nuclei code size reduction extension

vlen=128,elen=64: The VLEN and ELEN are only effective when the V extension instructions of RISC-V are enabled. The default value of VLEN is 128, and it must be a multiple of 2 when set, with a value range of [128, 1024]. The default value of ELEN is 64, and ELEN must also be a multiple of 2, with a value range of [8, 64].

s=true: This parameter is optional, If you wish for RISC-V to support the S (supervisor) privilege mode, you can add s=true to the parameters to meet this requirement. Nuclei QEMU currently only supports interrupt handling in M-privilege mode.

- `-m 512M`: To set the DDR size in QEMU, if the DDR size is not passed with `-m`, then the JSON config will be used to determine the size, and lastly, if neither is specified, it will initialize with 32MB.

Note: The following is the current default qemu memory size configuration, **xip: 32MB**, **ddr:64MB**, **ilm: 8MB**, **dlm: 8MB**, **sram: 512MB**. You can change the size of the DDR by using **-m size**. When **-m 128M** or no `-m` is passed, the default DDR size configured in the JSON or the size initialized by the program will be used. If the DDR size is configured too large and the computer does not have enough memory to allocate, an error such as `qemu-system-riscv32: cannot set up guest memory 'riscv.evalsoc.ram.sram'` may occur.

- `-smp 1`: Nuclei Qemu currently supports up to 16 CPUs. If this parameter is not set, the uses 1 CPU.

- `-icount shift=0`: This parameter is optional, Qemu TCG Instruction Counting. By enabling this option, you can enable qemu ' s instruction count. For more detailed information, refer to https://www.qemu.org/docs/master/devel/tcg-icount.html
- `-nodefaults`: QEMU is used to disable all default devices and configurations, and some custom parameters and commands can be passed.
- `-nographic`: Disable qemu ' s graphical interface and redirect standard output to the console.
- `-serial stdio`: Direct standard output to the console.
- `-kernel or -bios`: Choose the boot mode for the firmware. By default, programs on nuclei-sdk load using the `-kernel` mode, while on Linux, they load using the `-bios` mode. In the design of Nuclei Qemu, `-kernel` enables the use of **ECLIC**. For bare metal or RTOS, `-kernel` is used to transfer ELF file, while `-bios` is used to enable **PLIC+CLINT** timers, which are more suitable for Linux applications.

5.4 Use Nuclei QEMU in Nuclei SDK

Setup Tools and Environment

1. Download the nuclei-sdk[22], checkout to `master` branch.
2. Download RISC-V GNU Toolchain form Nuclei Download Center[23].
3. Download Nuclei Qemu form Nuclei Download Center[24].
4. Set up the system environment variables to ensure that the directories containing `riscv64-unknown-elf-gcc` and `qemu-system-riscv32` are included in the global system variable environment.

Example

If you want to use QEMU on Nuclei-SDK.The example here uses the CPU of the nx900fd, but other CPU types can also be used for testing. The example is xxdsp.

First, you need to configure the toolchain, nuclei-sdk, and qemu environments according to the documentation, https://doc.nucleisys.com/nuclei_sdk/quickstart.html

```
# Enter the example folder of xxldsp
cd nuclei-sdk/application/baremetal/demo_dsp/
# Clear the compilation cache
make clean
# Compile the program for the nx900fd, set the download mode to ILM, and enable␣
↪the xxldsp extension
make CORE=nx900fd SOC=evalsoc DOWNLOAD=ilm ARCH_EXT=_xxldsp dasm
# Automatically generate qemu running commands and execute the program
make CORE=nx900fd SOC=evalsoc DOWNLOAD=ilm ARCH_EXT=_xxldsp run_qemu
```

Where **ARCH_EXT** can be used to pass the extension name. Under normal circumstances, you should see the final output `NMSIS_TEST_PASS`, which indicates that all test cases have passed successfully.

And Nuclei QEMU and Nuclei SDK are deeply integrated in Nuclei Studio, you can also use it in Nuclei Studio, see https://nucleisys.com/upload/files/doc/nucleistudio/Nuclei_Studio_User_Guide.202406.pdf

[22] https://github.com/Nuclei-Software/nuclei-sdk
[23] https://nucleisys.com/download.php
[24] https://nucleisys.com/download.php

5.5 Use Nuclei QEMU in Nuclei Linux SDK

Nuclei QEMU can also used to boot and test RISC-V Linux Kernel using emulated Nuclei EvalSoC, please check documentation here https://github.com/Nuclei-Software/nuclei-linux-sdk#booting-linux-on-nuclei-qemu .

An example of a typical Nuclei QEMU running Nuclei Linux SDK is as follows:

```
qemu-system-riscv64 -M nuclei_evalsoc,download=flashxip,soc-cfg=soc.json -cpu␣
↪nuclei-ux900fd,ext= -smp 8 -m 2G -bios freeloader_qemu.elf -nographic -drive␣
↪file=disk.img,if=sd,format=raw
```

This command sets up QEMU to emulate a Nuclei processor and environment specifically for the Nuclei Linux SDK. Here ' s a breakdown of the parameters:

- `qemu-system-riscv64`: This is the QEMU emulator for the RISC-V 64-bit architecture.
- `-M nuclei_evalsoc`: Specifies the machine type for nuclei_evalsoc, nuclei_demosoc will be deprecated in future versions.
- `download=flashxip`: The download mode of firmware, which is an optional parameter. If not set, the default download mode is flashxip.
- `soc-cfg=evalsoc.json`: optional, additional configuration scripts can customize the interrupt information and memory address information of peripherals. For details, see Description of Parameters.
- `-cpu nuclei-ux900fd`: Selects the Nuclei UX900FD CPU model for emulation.
- `-ext=`: You can pass the extensions supported by riscv, and connect multiple extensions with _, eg. `_zba_zbb_zbc_zbs_zicond`.
- `-smp 8`: Enables Symmetric Multi-Processing (SMP) with 8 CPU cores.
- `-m 2G`: Allocates 2GB of RAM to the virtual machine.
- `-bios freeloader_qemu.elf`: Specifies the BIOS or bootloader to use, in this case a freeloader named freeloader_qemu.elf specifically for QEMU.
- `-nographic`: Disables graphical output, making QEMU run in a text-only mode.
- `-drive file=disk.img,if=sd,format=raw`: Attaches a virtual disk image named `disk.img` to the virtual machine, using the SD card interface (if=sd) and a raw file format (format=raw). This disk image likely contains the Nuclei Linux SDK filesystem.

CHAPTER

SIX

NUCLEI MODEL

6.1 About Nuclei Near Cycle Model

The `Nuclei Near Cycle Model` (referred to as *xlmodel*) is a co-simulation using **SystemC TLM-2 combined with xlspike**. Xlspike uses spike as the RISC-V ISA simulator and adds support for Nuclei ' s N/NX/UX RISC-V processors. SystemC establishes the TLM 2.0 interaction relationships among the components under **Nuclei EvalSoC**.

- The *xlmodel* can be obtained in **Nuclei Studio 2024.06**.
- The *xlmodel* is only supported on Linux system.
- The *xlmodel* is a **near cycle model** that can test the performance of firmware with different ISA configurations.
- The *xlmodel* has SystemC built-in and uses version 2.3.4 by default.
- The *xlmodel* has built-in **gprof** functionality, allowing for a more intuitive display of the function call stack, the cycle count proportion of each function within the test code.
- The *xlmodel* can easily extend **user-defined** instructions, namely **Nuclei NICE instructions**.
- Xlspike supports RV32IMAFDCBPV and RV64IMAFDCBPV ISA.

6.2 SystemC components

Brief description of the *xlmodel* SystemC components:

- Top: Top-level entity that builds & starts the SystemC simulation
- Cluster: Cluster contains multiple cores and can be used to start xlspike for co-simulation
- Bus: Simple bus manager
- Memory: Memory comprises both instruction memory and data memory, as well as the loading of ELF sections
- Timer: Nuclei timer unit
- EvalUart: Nuclei EvalSoC uart
- EvalQspi: Nuclei EvalSoC qspi

6.3 How to run

6.3.1 model help

Frequently used command line parameters

parameter	description
`--version`	display version info
`--time=<n>`	set simulation time(us); otherwise, it is unlimited
`--bpu=<str>`	core bpu type config, defaults to `--bpu=n300` and can be set to `n900`
`--smp=<n>`	SMP system core number configuration, with a maximum of 16
`--trace=<n>`	whether generate the trace file, `--trace=1` means generating
`--gprof=<n>`	whether to use profiling, `--gprof=1` means using profiling
`--varch=<str>`	RISCV Vector uArch config, defaults to `--varch=vlen:128,elen:64`
`--log=<str>`	logging system level, defaults to `--log=info` and can be set to `error`, `tlm`, `debug`
`--logdir=<str>`	the directory to save trace and gprof files

You need to pass the different parameters above based on the results you want to obtain and the specific test code. In most cases, you only need to pass the path of the ELF file.

6.3.2 normal test case

When you want to use the model, you can select the *bin/xl_cpumodel* as the executable file.

The default test codes ELF files are provided in the *tests* directory, this is an example of demo timer test code below:

```
./xl_cpumodel ../tests/demotimer/demotimer_nx900.elf
```

```
xuzt@whml1:/Local/xuzt/clib/model_ide/nuclei_cpu_model/build$ ./xl_cpumodel ../tests/demotimer/demotimer_nx900.elf

        SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
        Copyright (c) 1996-2022 by all Contributors,
        ALL RIGHTS RESERVED
[XLMODEL-INFO] filename[0]: ../tests/demotimer/demotimer_nx900.elf
[XLMODEL-INFO] Found the following .elf files:
[XLMODEL-INFO] ../tests/demotimer/demotimer_nx900.elf
[XLMODEL-INFO] Created Cluster0
[XLMODEL-INFO] start pc: 0x80000200
[XLMODEL-INFO] rv64 file
[XLMODEL-INFO] argv[0]: -t
[XLMODEL-INFO] argv[1]: -p1
[XLMODEL-INFO] argv[2]: +permissive-off
[XLMODEL-INFO] argv[3]: ../tests/demotimer/demotimer_nx900.elf
Nuclei SDK Build Time: Mar 27 2024, 10:30:49
Download Mode: ILM
CPU Frequency 692715 Hz
CPU HartID: 0
init timer and start
MTimer IRQ handler 1
MTimer IRQ handler 2
MTimer IRQ handler 3
MTimer IRQ handler 4
MTimer IRQ handler 5
MTimer SW IRQ handler 1
MTimer SW IRQ handler 2
MTimer SW IRQ handler 3
MTimer SW IRQ handler 4
MTimer SW IRQ handler 5
MTimer msip and mtip interrupt test finish and pass

[XLMODEL-INFO] total run 431608 instruction

Info: /OSCI/SystemC: Simulation stopped by user.
[XLMODEL-INFO] Total elapsed time: 2.591830s
[XLMODEL-INFO] Press Enter to finish
```

The *xlmodel* can select different **BPU strategies** based on Nuclei core types, which will affect the cycle count of branch and jump instructions.

You can pass the parameter *bpu=n300*, which is applicable to Nuclei cores up to and including N300 (<= N300). You can also pass the parameter *bpu=n900*, which is applicable to Nuclei cores from N300 onwards (> N300):

```
./xl_cpumodel ../tests/demotimer/demotimer_nx900.elf --bpu=n300
```

```
xuzt@whml1:/Local/xuzt/clib/model_ide/nuclei_cpu_model/build$ ./xl_cpumodel ../tests/demotimer/demotimer_nx900.elf --bpu=n300

        SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
        Copyright (c) 1996-2022 by all Contributors,
        ALL RIGHTS RESERVED
[XLMODEL-INFO] filename[0]: ../tests/demotimer/demotimer_nx900.elf
[XLMODEL-INFO] Found the following .elf files:
[XLMODEL-INFO] ../tests/demotimer/demotimer_nx900.elf
[XLMODEL-INFO] Created Cluster0
[XLMODEL-INFO] start pc: 0x80000200
[XLMODEL-INFO] rv64 file
[XLMODEL-INFO] argv[0]: -t
[XLMODEL-INFO] argv[1]: -p1
[XLMODEL-INFO] argv[2]: +permissive-off
[XLMODEL-INFO] argv[3]: ../tests/demotimer/demotimer_nx900.elf
Nuclei SDK Build Time: Mar 27 2024, 10:30:49
Download Mode: ILM
CPU Frequency 692715 Hz
CPU HartID: 0
init timer and start
MTimer IRQ handler 1
MTimer IRQ handler 2
MTimer IRQ handler 3
MTimer IRQ handler 4
MTimer IRQ handler 5
MTimer SW IRQ handler 1
MTimer SW IRQ handler 2
MTimer SW IRQ handler 3
MTimer SW IRQ handler 4
MTimer SW IRQ handler 5
MTimer msip and mtip interrupt test finish and pass

[XLMODEL-INFO] total run 431608 instruction

Info: /OSCI/SystemC: Simulation stopped by user.
[XLMODEL-INFO] Total elapsed time: 2.609912s
[XLMODEL-INFO] Press Enter to finish
```

When testing an SMP system case, you need to pass the parameter *smp=n*, specifying the number of cores, to the command line:

```
./xl_cpumodel ../tests/smphello_nx900/smphello_nx900.elf --smp=4
```

```
xuzt@whml1:/Local/xuzt/clib/model_ide/nuclei_cpu_model/build$ ./xl_cpumodel ../tests/smphello_nx900/smphello_nx900.elf --smp=4

        SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
        Copyright (c) 1996-2022 by all Contributors,
        ALL RIGHTS RESERVED
[XLMODEL-INFO] cluster0 smp num: 4
[XLMODEL-INFO] filename[0]: ../tests/smphello_nx900/smphello_nx900.elf
[XLMODEL-INFO] Found the following .elf files:
[XLMODEL-INFO] ../tests/smphello_nx900/smphello_nx900.elf
[XLMODEL-INFO] Created Cluster0
[XLMODEL-INFO] start pc: 0x80000200
[XLMODEL-INFO] rv64 file
[XLMODEL-INFO] argv[0]: -t
[XLMODEL-INFO] argv[1]: -p4
[XLMODEL-INFO] argv[2]: +permissive-off
[XLMODEL-INFO] argv[3]: ../tests/smphello_nx900/smphello_nx900.elf
Nuclei SDK Build Time: Apr  2 2024, 10:11:24
Download Mode: ILM
CPU Frequency 156988 Hz
CPU HartID: 0
Current RUNMODE=icdcccm, ilm:1, dlm 1, icache 1, dcache 1, ccm 1
CSR: MILM_CTL 0x0, MDLM_CTL 0x0, MCACHE_CTL 0x10001
Hello world from hart 0
Hello world from hart 1
Hello world from hart 2
Hello world from hart 3
All harts boot successfully!


[XLMODEL-INFO] total run 28381 instruction

Info: /OSCI/SystemC: Simulation stopped by user.
[XLMODEL-INFO] Total elapsed time: 0.766246s
[XLMODEL-INFO] Press Enter to finish
```

If you need to test Vector extension case and specify vlen or elen, you need to pass the parameter *varch=vlen:n,elen:n*. Note that vlen and elen must comply with the RISC-V Vector specifications:

```
./xl_cpumodel ../tests/rvv_conv_f32/rvv_conv_f32.elf --varch=vlen:1024,elen:64
```

```
xuzt@whml1:/Local/xuzt/clib/model_ide/nuclei_cpu_model/build$ ./xl_cpumodel ../tests/rvv_conv_f32/rvv_conv_f32.elf --varch=vlen:1024,elen:64

        SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
        Copyright (c) 1996-2022 by all Contributors,
        ALL RIGHTS RESERVED
[XLMODEL-INFO] filename[0]: ../tests/rvv_conv_f32/rvv_conv_f32.elf
[XLMODEL-INFO] Found the following .elf files:
[XLMODEL-INFO] ../tests/rvv_conv_f32/rvv_conv_f32.elf
[XLMODEL-INFO] Created Cluster0
[XLMODEL-INFO] start pc: 0x80000200
[XLMODEL-INFO] rv64 file
[XLMODEL-INFO] argv[0]: -t
[XLMODEL-INFO] argv[1]: -p1
[XLMODEL-INFO] argv[2]: --varch=vlen:1024,elen:64
[XLMODEL-INFO] argv[3]: +permissive-off
[XLMODEL-INFO] argv[4]: ../tests/rvv_conv_f32/rvv_conv_f32.elf
Nuclei SDK Build Time: Jun  4 2024, 10:46:53
Download Mode: ILM
CPU Frequency 691404 Hz
CPU HartID: 0
Benchmark initialized
vlen = 1024 bits
The example is : C(1151) = conv(A(1024), B(128))
CSV, conv_golden, 5499267
CSV, conv_rvv, 7344
CSV, conv_rvv_noseg, 4636
pass

[XLMODEL-INFO] total run 4137041 instruction

Info: /OSCI/SystemC: Simulation stopped by user.
[XLMODEL-INFO] Total elapsed time: 24.829407s
[XLMODEL-INFO] Press Enter to finish
```

6.3.3 test case with trace

When you want to see the complete trace of the firmware elf, you need to pass the parameter *trace=1* in the command line:

```
./xl_cpumodel ../tests/rvv_conv_f32/rvv_conv_f32.elf --varch=vlen:1024,elen:64 --
↪trace=1
```

You can obtain information such as the pc, opcode, disassembly, and others for each instruction in generated *<elf-name>.rvtrace* file:

```
rv64 file        pc    opcode   disassemble
1 hart0 (1) IT 80000200 30047073 3 :csrci   mstatus, 8
1 hart0 R CSR 0x300:CSR_MSTATUS:0xa00000000
2 hart0 (2) IT 80000204 f1402573 3 :csrr    a0, mhartid
2 hart0 R x10:0x0
3 hart0 (3) IT 80000208 0ff57513 3 :andi    a0, a0, 255
3 hart0 R x10:0x0
4 hart0 (4) IT 8000020c 00004581 3 :c.li    a1, 0
4 hart0 R x11:0x0
5 hart0 (5) IS 8000020e 12b51463 3 :bne     a0, a1, pc + 296
5 hart0 B 0x80000336
6 hart0 (6) IT 80000212 10001197 3 :auipc   gp, 0x10001
6 hart0 R x3:0x90001212
7 hart0 (7) IT 80000216 83e18193 3 :addi    gp, gp, -1986
7 hart0 R x3:0x90000a50
8 hart0 (8) IT 8000021a 10001217 3 :auipc   tp, 0x10001
8 hart0 R x4:0x9000121a
9 hart0 (9) IT 8000021e 93e20213 3 :addi    tp, tp, -1730
9 hart0 R x4:0x90000b58
10 hart0 (10) IT 80000222 10080117 3 :auipc   sp, 0x10080
10 hart0 R x2:0x90080222
```

6.3.4 test case with log

The *xlmodel* has multiple log levels, listed from least to most detailed as *error*, *info*, *tlm*, *debug*. By default, it provides log information at the *info* level, which can be changed by passing *log=xxx*.

When *log=tlm* is selected, logs related to all TLM memory reads and writes can be printed:

```
        SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
        Copyright (c) 1996-2022 by all Contributors,
        ALL RIGHTS RESERVED
[XLMODEL-INFO] filename[0]: ../tests/helloworld/helloworld.elf
[XLMODEL-INFO] Found the following .elf files:
[XLMODEL-INFO] ../tests/helloworld/helloworld.elf
[XLMODEL-INFO] Created Cluster0
[XLMODEL-INFO] start pc: 0x80000100
[XLMODEL-INFO] rv32 file
[XLMODEL-INFO] argv[0]: -t
[XLMODEL-INFO] argv[1]: -p1
[XLMODEL-INFO] argv[2]: +permissive-off
[XLMODEL-INFO] argv[3]: ../tests/helloworld/helloworld.elf
[XLMODEL-TLM] TLM Write: Address=0x900004f8 Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x900004fc Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x90000500 Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x90000504 Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x90000508 Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x9000050c Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x90000510 Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x90000514 Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x90000518 Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x9000051c Data=0x0 Size=4
[XLMODEL-TLM] TLM Write: Address=0x90000520 Data=0x0 Size=4
```

When *log=debug* is selected, TLM information and more detailed instruction trace information, including register updates, exceptions, and CSRs, will be printed:

```
        SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
        Copyright (c) 1996-2022 by all Contributors,
        ALL RIGHTS RESERVED
[XLMODEL-INFO] filename[0]: ../tests/helloworld/helloworld.elf
[XLMODEL-INFO] Found the following .elf files:
[XLMODEL-INFO] ../tests/helloworld/helloworld.elf
[XLMODEL-INFO] Created Cluster0
[XLMODEL-INFO] start pc: 0x80000100
[XLMODEL-INFO] rv32 file
[XLMODEL-INFO] argv[0]: -t
[XLMODEL-INFO] argv[1]: -p1
[XLMODEL-INFO] argv[2]: +permissive-off
[XLMODEL-INFO] argv[3]: ../tests/helloworld/helloworld.elf
[XLMODEL-DEBUG] 1 hart0 (1) IT 80000100 30047073 3 :csrci   mstatus, 8
[XLMODEL-DEBUG] 1 hart0 R CSR ID 0x300 value 0x0
[XLMODEL-DEBUG] CSR_MSTATUS
[XLMODEL-DEBUG] 2 hart0 (2) IT 80000104 f1402573 3 :csrr    a0, mhartid
[XLMODEL-DEBUG] 2 hart0 R x10: 0x0
[XLMODEL-DEBUG] 3 hart0 (3) IT 80000108 0ff57513 3 :andi    a0, a0, 255
[XLMODEL-DEBUG] 3 hart0 R x10: 0x0
[XLMODEL-DEBUG] 4 hart0 (4) IT 8000010c 00004581 3 :c.li    a1, 0
[XLMODEL-DEBUG] 4 hart0 R x11: 0x0
[XLMODEL-DEBUG] 5 hart0 (5) IS 8000010e 10b51a63 3 :bne     a0, a1, pc + 276
[XLMODEL-DEBUG] 5 hart0 B 0xffffffff80000222
```

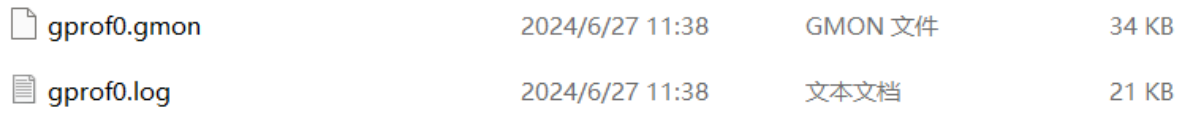
6.3.5 test case with gprof

If you want to use the model ' s built-in **gprof** functionality, you need to pass *gprof=1* to the command line:

```
./xl_cpumodel ../tests/whet/whet_nx900.elf --gprof=1
```

```
xuzt@whml1:/Local/xuzt/clib/model_ide/nuclei_cpu_model_old/nuclei_cpu_model/xlmodel/bin$ ./xl_cpumodel ../tests/whet/whet_nx900.elf --gprof=1

        SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
        Copyright (c) 1996-2022 by all Contributors,
        ALL RIGHTS RESERVED
[XLMODEL-INFO] filename[0]: ../tests/whet/whet_nx900.elf
[XLMODEL-INFO] Found the following .elf files:
[XLMODEL-INFO] ../tests/whet/whet_nx900.elf
[XLMODEL-INFO] Created Cluster0
[XLMODEL-INFO] start pc: 0x80000200
[XLMODEL-INFO] rv64 file
[XLMODEL-INFO] argv[0]: -t
[XLMODEL-INFO] argv[1]: -p1
[XLMODEL-INFO] argv[2]: +permissive-off
[XLMODEL-INFO] argv[3]: ../tests/whet/whet_nx900.elf
Nuclei SDK Build Time: Mar 21 2024, 19:02:22
Download Mode: ILM
CPU Frequency 830013 Hz
CPU HartID: 0

########################################
Double Precision C Whetstone Benchmark Opt 3 32 Bit
Calibrate
       6.36 Seconds          1   Passes (x 100)

Use 1  passes (x 100)


          Double Precision C/C++ Whetstone Benchmark
Loop content                  Result              MFLOPS      MOPS   Seconds

N1 floating point -1.12441415430188418         0.667              0.029
N2 floating point -1.12239951147853723         0.751              0.179
N3 if then else    1.00000000000000000                  10.277    0.010
N4 fixed point    12.00000000000000000                   8.251    0.038
N5 sin,cos etc.    0.49907428465337039                   0.026    3.142
N6 floating point  0.99999988495142078         0.794              0.679
N7 assignments     3.00000000000000000                   3.094    0.060
N8 exp,sqrt etc.   0.75095530233203855                   0.017    2.221

MWIPS                                             1.573              6.358


MWIPS/MHz                                         1.895              6.358


CSV, Benchmark, MWIPS/MHz
CSV, Whetstone, 1.895
IPC = Instret/Cycle = 4629917/6813377 = 0.679

[XLMODEL-INFO] total run 4652262 instruction

Info: /OSCI/SystemC: Simulation stopped by user.
[XLMODEL-INFO] Total elapsed time: 69.523014s
[XLMODEL-INFO] Press Enter to finish
```

You can obtain the *gprof<n>.gmon* and *gprof<n>.log* files when the simulation is complete, where n represents the hart ID:



To use them further, you need to import them into the IDE, then you can refer to the model usage guide in the **Nuclei Studio** for detailed instructions on using **gprof**.

6.4 NICE support

The current **NICE** example in *tests/demonice* demonstrates how to use Nuclei **NICE** feature:

```
./xl_cpumodel ../tests/demonice/demonice_nx900.elf
```

```
xuzt@whml1:/Local/xuzt/clib/model_ide/nuclei_cpu_model_old/nuclei_cpu_model/build$ ./xl_cpumodel ../tests/demonice/demonice_nx900.elf

        SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
        Copyright (c) 1996-2022 by all Contributors,
        ALL RIGHTS RESERVED
[XLMODEL-INFO] filename[0]: ../tests/demonice/demonice_nx900.elf
[XLMODEL-INFO] Found the following .elf files:
[XLMODEL-INFO] ../tests/demonice/demonice_nx900.elf
[XLMODEL-INFO] Created Cluster0
[XLMODEL-INFO] start pc: 0x80000200
[XLMODEL-INFO] rv64 file
[XLMODEL-INFO] argv[0]: -t
[XLMODEL-INFO] argv[1]: -p1
[XLMODEL-INFO] argv[2]: +permissive-off
[XLMODEL-INFO] argv[3]: ../tests/demonice/demonice_nx900.elf
Nuclei SDK Build Time: Jun 21 2024, 18:26:14
Download Mode: ILM
CPU Frequency 693370 Hz
CPU HartID: 0

Nuclei Nice Acceleration Demonstration
Warning: This demo required CPU to implement Nuclei provided NICE Demo instructions.
         Otherwise this example will trap to cpu core exception!

1. Print input matrix array
the element of array is :
        10      30      90
        20      40      80
        30      90      120

2. Do reference matrix column sum and row sum
2. Do nice matrix column sum and row sum
3. Compare reference and nice result
  1) Reference result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  2) Nice result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  3) Compare reference vs nice: PASS
4. Performance summary
         normal:
              instret: 459, cycle: 567
         nice  :
              instret: 121, cycle: 145

[XLMODEL-INFO] total run 101741 instruction

Info: /OSCI/SystemC: Simulation stopped by user.
[XLMODEL-INFO] Total elapsed time: 0.617883s
[XLMODEL-INFO] Press Enter to finish
```

If you need to validate your custom **NICE** instructions, you need to contact Nuclei Support to obtain software package of model.

The directory structure of the package is as follows:

nice directory	description
nice	header and source files for the NICE interface
systemc	SystemC 2.3.4 header files and static libraries
xl_model	xlmodel header files and library files
xl_spike	xlspike header files and library files
tests	test codes
CMakeLists.txt	CMake file required for compilation

CHAPTER

SEVEN

CHANGELOG

Nuclei Tools official releases can be found in https://nucleisys.com/download.php#tools, you can download it from there.

7.1 2024.06

This is 2024.06 release of Nuclei Tools.

This release introduced *About Nuclei Near Cycle Model* (page 279) first release which can be used in Ubuntu 20.04 environment.

Find release note below:

- Nuclei Studio 2024.06[25]
- Nuclei RISC-V Toolchain 2024.06[26]
- Nuclei QEMU 2024.06[27]
- Nuclei OpenOCD 2024.06[28]

Click https://github.com/Nuclei-Software/nuclei-tool-guide/issues/4 for this release known issues.

7.2 2024.02

This release is a bugfix release for 2023.10, which fix many issues reported by 2023.10.

See https://github.com/Nuclei-Software/nuclei-tool-guide/issues/1

7.3 2023.10

This is 2023.10 release of Nuclei Tools.

Find release note below:

- Nuclei Studio 2023.10[29]
- Nuclei RISC-V Toolchain 2023.10[30]
- Nuclei QEMU 2023.10[31]

[25] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2024.06
[26] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2024.06
[27] https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2024.06
[28] https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2024.06
[29] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2023.10
[30] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2023.10
[31] https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2023.10

- Nuclei OpenOCD 2023.10[32]

Click https://github.com/Nuclei-Software/nuclei-tool-guide/issues/1 for this release known issues.

[32] https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2023.10

CHAPTER

EIGHT

GLOSSARY

API

(Application Program Interface) A defined set of routines and protocols for building application software.

ISR

(Interrupt Service Routine) Also known as an interrupt handler, an ISR is a callback function whose execution is triggered by a hardware interrupt (or software interrupt instructions) and is used to handle high-priority conditions that require interrupting the current code executing on the processor.

XIP

(eXecute In Place) a method of executing programs directly from long term storage rather than copying it into RAM, saving writable memory for dynamic data and not the static program code.

CHAPTER

NINE

APPENDIX

- **Nuclei Tools and Documents**: https://nucleisys.com/download.php
- **Nuclei Software Opensource Organization**: https://github.com/Nuclei-Software
- **RISC-V MCU Opensource Organization**: https://github.com/riscv-mcu
- **Nuclei Toolchain Repo**: https://github.com/riscv-mcu/riscv-gnu-toolchain
- **Nuclei OpenOCD Repo**: https://github.com/riscv-mcu/riscv-openocd
- **Nuclei QEMU Repo**: https://github.com/riscv-mcu/qemu
- **Nuclei SDK**: https://github.com/Nuclei-Software/nuclei-sdk
- **NMSIS**: https://github.com/Nuclei-Software/NMSIS
- **Nuclei RISC-V IP Products**: https://www.nucleisys.com/product.php
- **RISC-V MCU Community Website**: https://www.riscv-mcu.com/
- **Nuclei RISC-V CPU Spec**: https://doc.nucleisys.com/nuclei_spec
- **RISC-V ELF psABI Document**: https://github.com/riscv-non-isa/riscv-elf-psabi-doc#navigation
- **RISC-V ISA Specfications(Ratified)**: https://riscv.org/technical/specifications
- **RISC-V Architecture Profiles**: https://github.com/riscv/riscv-profiles
- **RISC-V Bitmanip(B) Extension Spec**: https://github.com/riscv/riscv-bitmanip
- **RISC-V Packed SIMD(P) Extension Spec**: https://github.com/riscv/riscv-p-spec
- **RISC-V Cryptography(K) Extension Spec**: https://github.com/riscv/riscv-crypto
- **RISC-V Vector(V) Extension Spec**: https://github.com/riscv/riscv-v-spec
- **RISC-V Vector Intrinsic API Spec**: https://github.com/riscv-non-isa/rvv-intrinsic-doc
- **RISC-V ISA Extension Spec Status**: https://wiki.riscv.org/display/HOME/Specification+Status
- **Nuclei Bumblebee Core Document**: https://github.com/nucleisys/Bumblebee_Core_Doc

CHAPTER

TEN

INDICES AND TABLES

- genindex
- search

INDEX