

Nuclei Development Tool Guide

Release 2026.10

Nuclei

Nov 11, 2026

CONTENTS

1 Introduction **1**

2 Nuclei Studio IDE **3**

2.1 Introduction to Nuclei Studio IDE . . . 3
2.2 Nuclei Studio Release Notes . . . 4
2.2.1 2026.10 Release Notes . . . 5
2.2.2 2025.10 Release Notes . . . 8
2.2.3 2025.02 Release Notes . . . 10
2.2.4 2024.06 Release Notes . . . 12
2.2.5 2024.02.dev Release Notes . . . 13
2.2.6 2023.10 Release Notes . . . 14
2.2.7 2022.12 Release Notes . . . 28
2.3 Nuclei Studio Download and Installation . . . 28
2.3.1 Nuclei Studio IDE Download . . . 28
2.3.2 Nuclei Studio IDE Installation . . . 28
2.3.3 Launching Nuclei Studio IDE . . . 29
2.4 Nuclei Studio NPK Overview . . . 30
2.4.1 Package Description File (npk.yml) . . . 31
2.4.2 Section Overview . . . 42
2.4.3 UI Components in NPK . . . 53
2.4.4 NPK Syntax . . . 59
2.5 Nuclei Studio NPK Creation and Sharing . . . 63
2.5.1 Develop an NPK Component Package . . . 63
2.5.2 Checking and Troubleshooting NPK Component Packages . . . 74
2.5.3 Share an NPK Component Package . . . 79
2.5.4 Using NPK Component Packages in Nuclei Studio . . . 85
2.6 Nuclei Studio NPK Applications . . . 86
2.6.1 NPK Package Management . . . 86
2.6.2 Create a Project via NPK . . . 95
2.6.3 Import Tools via NPK . . . 99
2.7 Create a Project in Nuclei Studio . . . 101
2.7.1 Automatically Create a Project Using an NPK Template Project . . . 101
2.7.2 Import a Project via an Application Association File . . . 107
2.7.3 Import an Existing Project Directly to Create a New Project . . . 109
2.7.4 Manually Create a Project Without a Template . . . 113
2.7.5 Create a Project Based on an Existing Makefile . . . 130
2.8 Build a Project in Nuclei Studio . . . 134
2.8.1 Build Tools . . . 136
2.8.2 Nuclei SDK Project Settings Tool . . . 142
2.8.3 Build the Hello World Project in Nuclei Studio . . . 146
2.9 Debugging and Running Projects in Nuclei Studio . . . 148
2.9.1 Debug Mode Management . . . 148
2.9.2 Debugging and Running a Project with the HummingBird Debugger and OpenOCD . . . 150
2.9.3 Debugging and Running a Project with J-Link . . . 168

2.9.4 Debugging and Running a Project with DLink . . . 187
2.9.5 Connect to Running Target . . . 193
2.9.6 CPU Information . . . 195
2.9.7 Flash Programming . . . 202
2.9.8 Flash Erase . . . 207
2.10 Nuclei Studio Advanced Features . . . 214
2.10.1 Importing Projects Created by Older Versions . . . 214
2.10.2 Extension Selector . . . 222
2.10.3 LST View . . . 226
2.10.4 Code Coverage and Profiling . . . 229
2.10.5 Trace . . . 250
2.10.6 Flame View . . . 272
2.10.7 Using RVProf . . . 276
2.10.8 Nuclei NICE Wizard . . . 284
2.10.9 Using Nuclei Model Debugger . . . 291
2.10.10 Using Nuclei Model . . . 300
2.10.11 Nuclei Near Cycle Model . . . 300
2.10.12 Using GDB Custom Debugger . . . 306
2.10.13 Using Live Watch . . . 306
2.10.14 Live Peripheral . . . 320
2.10.15 Flash Programming . . . 326
2.10.16 Flash Erase . . . 327
2.10.17 Connect to Running Target . . . 327
2.10.18 Updating evalsoc in Nuclei QEMU . . . 328
2.10.19 Switching from GCC to Clang/ZCC . . . 329
2.10.20 CPU Information . . . 336
2.10.21 MMU View . . . 336
2.10.22 RISC-V Reference . . . 345
2.10.23 GDB Breakpoint Project Filter Option . . . 351
2.10.24 Memory View Address Exclusion . . . 353
2.10.25 Big Endian Compiler Support . . . 354
2.10.26 Register View Filter . . . 356
2.10.27 Disassembly View C Code Linkage . . . 357
2.11 Nuclei Studio Update and Upgrade . . . 358
2.11.1 Online Update . . . 358
2.11.2 Manual Update . . . 360
2.12 Nuclei Studio FAQ . . . 362
2.12.1 Nuclei Studio Starts Slowly . . . 362
2.12.2 Nuclei Studio Builds Programs Slowly . . . 363
2.12.3 Cannot Find the New Nuclei RISC-V C/C++ Project Menu . . . 363
2.12.4 Opening/Closing the Launch Bar . . . 364
2.12.5 Using the Launch Bar . . . 365
2.12.6 Running/Debugging Without the Launch Bar . . . 366
2.12.7 Viewing Registers in the Debug Perspective . . . 366
2.12.8 Terminating Processes in the Debug Perspective . . . 367
2.12.9 Functions of the Buttons in the Nuclei Studio Toolbar . . . 367
2.12.10 Showing Other Windows . . . 368
2.12.11 Restoring the Default Window Layout . . . 370
2.12.12 Comparing with File History . . . 370
2.12.13 Possible Errors When Creating a Project . . . 370
2.12.14 Cached Paths When Adding an Include Path . . . 371
2.12.15 Cannot Find a Settings Page Section . . . 371
2.12.16 Slow Download Speed to the Development Board . . . 372
2.12.17 Multiple Users Using Nuclei Studio on Linux . . . 372
2.12.18 Two Serial Ports Recognized in Device Manager . . . 372
2.12.19 Error “Could Not Determine GDB Version After Sending: riscv-nuclei-elf-gdb version, response:” on Linux . . . 373
2.12.20 Error Reported When Using QEMU on Linux . . . 373

2.12.21 Error About Missing Symbols When Linking the C Library During Project Build 374
2.12.22 Build Error: Fatal Error: rvintrin.h: No Such File or Directory 377
2.12.23 Debug Error: Error: Couldn ' t Find an Available Hardware Trigger 378
2.12.24 Viewing the GDB Trace While Debugging a Project 378
2.12.25 Connecting to a Running Target While Debugging a Project 380
2.12.26 Known Defects in Flash Programming 382
2.12.27 Hardware Check Before Debugging a Project in Nuclei Studio 384
2.12.28 Nuclei Studio Cannot Debug Due to a Low CPU Frequency 387
2.12.29 Setting the Nuclei Studio Theme Colors 388
2.12.30 No Project Creation Menu in the Debug Perspective 389
2.12.31 Build Error: DWARF Error: Mangled Line Number Section (Bad File Number) 391
2.12.32 How to Build a Project That Supports Semihosting 391
2.12.33 How to Debug a Semihosting-Enabled Project with J-Link 393
2.12.34 Debug Session Hangs Due to Address Preloading in the Memory Window 394
2.12.35 How to View GDB Trace 396
2.13 Other Issues Without a Noted Version 398
2.14 Known Issues 398
2.14.1 Known Issues in 2026.10 398

3 Nuclei Studio VSCode **401**
3.1 Introduction to Nuclei Studio VSCode 401
3.1.1 Features 401
3.1.2 The Nuclei Studio VSCode View Container 402
3.1.3 Requirements 403
3.1.4 Extension Settings 403
3.2 Changelog 403
3.2.1 Version 2026.10 403
3.3 Nuclei Studio VSCode Download and Installation 405
3.3.1 Install Nuclei Studio IDE 405
3.3.2 Install the Extension in VS Code 405
3.3.3 The Getting Started Walkthrough 406
3.3.4 Configure the Nuclei Studio IDE Installation Folder 408
3.4 NPK Package Management in Nuclei Studio VSCode 409
3.4.1 Opening the NPK Package Management Page 409
3.4.2 Managing Packages 410
3.5 Create a Project in Nuclei Studio VSCode 410
3.5.1 Workspaces and Projects 410
3.5.2 Create a Project with the New Project Wizard 411
3.5.3 Create a Hello World Project 412
3.5.4 Import a Project 412
3.5.5 Export a Project 413
3.5.6 Manage Projects 413
3.6 Build a Project in Nuclei Studio VSCode 413
3.6.1 Build, Clean, and Rebuild 413
3.6.2 Build Configurations 414
3.6.3 Build or Clean Selected Files 414
3.6.4 Exclude Resources from the Build 414
3.6.5 Toolchains 415
3.7 Debug and Run a Project in Nuclei Studio VSCode 415
3.7.1 Debug Schemes 415
3.7.2 Start Debugging 416
3.7.3 Launch Configuration Editors 416
3.7.4 Run Without Debugging 418
3.7.5 Serial Monitor and Debug Views 418
3.7.6 Limitations 418
3.8 Advanced Usage of Nuclei Studio VSCode 418
3.8.1 Live Watch 418
3.8.2 Nuclei Settings 420

3.8.3 Project Properties and Resource Properties . . . 422
3.8.4 Open Terminal . . . 422
3.8.5 IntelliSense . . . 422
3.8.6 Project Conversion . . . 423
3.9 Nuclei Studio VSCode Update and Upgrade . . . 423
3.9.1 Update the Extension . . . 423
3.9.2 Update the Nuclei Studio IDE and Tools . . . 423
3.9.3 Update NPK Packages . . . 423
3.9.4 Update the RISC-V Extension Data . . . 423
3.10 Nuclei Studio VSCode FAQ . . . 423
3.10.1 The Nuclei Studio VSCode Icon Does Not Appear in the Activity Bar . . . 423
3.10.2 VS Code Reports That the Workspace Is in Restricted Mode . . . 424
3.10.3 A "Configure Nuclei Studio Install Folder" Warning Is Shown in the WELCOME Panel 424
3.10.4 "Please install the required NPK packages first." When Creating a Project . . . 424
3.10.5 "Current workspace differs from Nuclei Studio VSCode workspace." . . . 424
3.10.6 Build Fails Because the Toolchain or `make` Cannot Be Found . . . 424
3.10.7 The Debug Session Does Not Start . . . 424
3.10.8 Live Watch Reports That the Feature Is Not Supported . . . 424
3.10.9 Rebuild Index Reports That the C/C++ Extension Is Not Available . . . 425
3.10.10 "Debug As…" Is Not Available for a Launch File . . . 425
3.10.11 Where Are the NPK Packages Installed . . . 425

4 Nuclei Toolchain **427**
4.1 GNU Toolchain . . . 427
4.1.1 About GNU Toolchain . . . 427
4.1.2 Extensions Support . . . 427
4.1.3 General Options . . . 429
4.1.4 Nuclei Custom Options . . . 431
4.1.5 Libraries . . . 432
4.1.6 Changelog . . . 432
4.1.7 Install and Setup . . . 432
4.1.8 Testsuite . . . 433
4.1.9 Checking GNU Version . . . 433
4.1.10 Known Issues . . . 433
4.1.11 Nuclei Custom BFloat16 Extension . . . 435
4.1.12 Nuclei Custom VPU Extension . . . 445
4.1.13 Nuclei Custom Xxlvw Extension . . . 451
4.2 LLVM Toolchain . . . 463
4.2.1 About LLVM Toolchain . . . 463
4.2.2 Extensions Support . . . 464
4.2.3 General Options . . . 465
4.2.4 Nuclei Custom Options . . . 466
4.2.5 Install and Setup . . . 466
4.2.6 Changelog . . . 466
4.2.7 Checking LLVM Version . . . 467
4.2.8 Known Issues . . . 467
4.3 Changelog . . . 467
4.3.1 Version 2026.10 . . . 467
4.3.2 Version 2025.10 . . . 469
4.3.3 Version 2025.02 . . . 470
4.3.4 Version 2024.06 . . . 470

5 Nuclei C Runtime Library **473**
5.1 About Nuclei C Runtime Library . . . 473
5.2 Nuclei C Runtime Library for Nuclei Toolchain . . . 473
5.2.1 Usage . . . 473
5.2.2 zicfilp landing pad usage . . . 474
5.2.3 Big-endian usage . . . 474

5.2.4 Varieties . . . 475
5.3 Quick start . . . 476
5.3.1 exit() . . . 476
5.3.2 Basic UART I/O functions (-lfileops_uart) . . . 476
5.3.3 Using Semihosting (-lfileops_semi) . . . 476
5.3.4 Using RTT (-lfileops_rtt) . . . 476
5.3.5 Thread-local storage . . . 477
5.3.6 Heap . . . 477
5.4 Runtime support . . . 478
5.4.1 Getting to main() and then exit() . . . 478
5.4.2 Multithreaded protection for the heap . . . 479
5.5 Using customized fileops . . . 479
5.6 C library API . . . 481
5.6.1 <assert.h> . . . 481
5.6.2 <complex.h> . . . 481
5.6.3 <ctype.h> . . . 517
5.6.4 <errno.h> . . . 531
5.6.5 <fcntl.h> . . . 531
5.6.6 <fenv.h> . . . 532
5.6.7 <float.h> . . . 538
5.6.8 <iso646.h> . . . 540
5.6.9 <limits.h> . . . 540
5.6.10 <locale.h> . . . 543
5.6.11 <malloc.h> . . . 545
5.6.12 <math.h> . . . 546
5.6.13 <setjmp.h> . . . 630
5.6.14 <stdalign.h> . . . 631
5.6.15 <stdbool.h> . . . 631
5.6.16 <stddef.h> . . . 632
5.6.17 <stdint.h> . . . 633
5.6.18 <stdio.h> . . . 638
5.6.19 <stdlib.h> . . . 652
5.6.20 <string.h> . . . 682
5.6.21 <time.h> . . . 704
5.6.22 <unistd.h> . . . 711
5.6.23 <wchar.h> . . . 712
5.6.24 <wctype.h> . . . 737
5.6.25 <xlocale.h> . . . 753
5.7 Compiler support API . . . 756
5.7.1 GNU library API . . . 756
5.8 External function interface . . . 790
5.8.1 I/O functions . . . 790
5.8.2 Heap protection functions . . . 792
5.8.3 Error and assertion functions . . . 792
5.8.4 RTC functions . . . 793
5.8.5 Locale functions . . . 794
5.9 Changelog . . . 794
5.9.1 Version 2026.10 . . . 794
5.9.2 Version 2025.02 . . . 795
5.9.3 Version 2024.06 . . . 795
5.9.4 Version 2023.10 . . . 795
5.10 Known Issues . . . 795
5.10.1 Version 2026.10 . . . 795
5.10.2 Version 2025.02 . . . 796
5.10.3 Version 2024.06 . . . 796

6 Nuclei OpenOCD **797**
6.1 OpenOCD Usage Guide . . . 797

6.1.1 Table of Contents . . . 797
6.1.2 Tool Installation and Environment Setup . . . 797
6.1.3 Repositories and Documentation . . . 797
6.1.4 Getting Started . . . 798
6.1.5 Nuclei-Specific Features . . . 799
6.1.6 Configuration File Overview . . . 802
6.1.7 Low-Cost Debugger Solution . . . 806
6.1.8 Frequently Asked Questions . . . 806
6.1.9 Common Problem Troubleshooting and Solutions . . . 807
6.1.10 Changelog . . . 807
6.1.11 Known Issues . . . 809
6.1.12 Other Debuggers . . . 811
6.2 OpenOCD Capture Trace Data . . . 811
6.2.1 Table of Contents . . . 811
6.2.2 Tool Installation and Environment Setup . . . 811
6.2.3 ETrace Version Overview . . . 811
6.2.4 Etrace Command Reference . . . 812
6.2.5 Trace Command Reference . . . 816
6.2.6 Common Problem Troubleshooting and Solutions . . . 818
6.3 TraceDecoder Decoding Trace Data . . . 818
6.3.1 Table of Contents . . . 819
6.3.2 Tool Installation and Environment Setup . . . 819
6.3.3 Features . . . 819
6.3.4 Command Line Usage . . . 819
6.3.5 Usage Examples . . . 824
6.3.6 Common Problem Troubleshooting and Solutions . . . 831
6.3.7 Changelog . . . 832

7 Nuclei QEMU 835
7.1 About Nuclei QEMU . . . 835
7.2 Design and Architecture . . . 835
7.2.1 Nuclei CPU Types Supported on QEMU . . . 836
7.2.2 Address Allocation of EvalSoC on QEMU . . . 836
7.2.3 Nuclei CPU and SoC Features Supported on QEMU . . . 838
7.3 Nuclei QEMU Command-line Options . . . 841
7.3.1 Command-line Syntax and Help . . . 841
7.3.2 EvalSoC Machine Configuration . . . 841
7.3.3 CPU Configuration . . . 846
7.3.4 Memory and CPU Count . . . 848
7.3.5 Boot and Image Loading . . . 848
7.3.6 Console and Character Devices . . . 848
7.3.7 Storage . . . 849
7.3.8 Networking . . . 849
7.3.9 Plugins . . . 849
7.3.10 Execution and Timing . . . 849
7.3.11 Debugging . . . 849
7.4 Use Nuclei QEMU in Nuclei SDK . . . 850
7.5 Use Nuclei QEMU in Nuclei Linux SDK . . . 851
7.6 Nuclei XEC Ethernet User Guide . . . 852
7.6.1 User Mode Networking . . . 852
7.6.2 TAP Mode Networking . . . 854
7.7 Nuclei NICE Plugin . . . 856
7.7.1 How to Load the NICE Plugin . . . 856
7.7.2 How to Extend `user_nice.c` . . . 856
7.7.3 Usage Notes . . . 858
7.8 Changelog . . . 859
7.8.1 Version 2026.10 . . . 859
7.8.2 Version 2025.10 . . . 860

7.8.3 Version 2025.02 . . . 860
7.9 Known Issues . . . 861
7.9.1 Open Known Issues . . . 861
7.9.2 Fixed issues on Nuclei QEMU 2026.10 . . . 861

8 Nuclei Model **863**
8.1 About Nuclei Near Cycle Model . . . 863
8.2 Design and Architecture . . . 864
8.2.1 SystemC components . . . 864
8.2.2 Address Allocation of Evalsoc in xlmodel . . . 864
8.2.3 Nuclei CPU Features Supported in xlmodel . . . 865
8.2.4 Nuclei SDK Cases Supported in xlmodel . . . 866
8.3 Nuclei Model Command-line Options . . . 866
8.3.1 Command-line Syntax and Help . . . 866
8.3.2 CPU Configuration . . . 867
8.3.3 SoC Configuration . . . 869
8.3.4 soc-cfg JSON Configuration . . . 870
8.3.5 Memory Configuration . . . 871
8.3.6 Simulation Speed and Time . . . 872
8.3.7 Trace, Profiling and Logs . . . 872
8.4 DEBUG . . . 873
8.5 Semihosting . . . 874
8.6 Use Nuclei Model in Nuclei SDK . . . 875
8.7 Use Nuclei Model in Nuclei Linux SDK . . . 876
8.8 NICE support . . . 877
8.8.1 NICE build . . . 877
8.8.2 NICE example . . . 879
8.9 SystemC integration . . . 879
8.10 Changelog . . . 880
8.10.1 Version 2026.10 . . . 880
8.10.2 Version 2025.10 . . . 882
8.10.3 Version 2025.02 . . . 883

9 ChangeLog **885**
9.1 2026.10 . . . 885
9.2 2025.10 . . . 886
9.3 2025.02 . . . 887
9.4 2024.06 . . . 887
9.5 2024.02 . . . 888
9.6 2023.10 . . . 888

10 Glossary **889**

11 Appendix **891**

12 Indices and tables **893**

Index **895**

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

---

You can also find our **Nuclei Studio Supply Documents** in https://doc.nucleisys.com/nuclei_studio_supply/, which is used for application notes using Nuclei Studio and Nuclei Tools.

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 Introduction to Nuclei Studio IDE

Note:

- Nuclei Studio now has video tutorials. Related content is continuously updated on the **Nuclei System Technology WeChat video channel**. Search for and follow the **Nuclei video channel** in WeChat to get the latest updates.
- If you encounter problems while using Nuclei Studio or Nuclei Tools, check the content at https://doc.nucleisys.com/nuclei_studio_supply/, or submit an issue to us.

An efficient and easy-to-use Integrated Development Environment (IDE) is essential for any MCU; software developers need an IDE for day-to-day project development and debugging. Many embedded software engineers in mainland China are familiar with ARM ' s commercial IDE, Keil. However, because commercial IDEs (such as Keil) involve licensing and cost issues, major MCU vendors also release their own free IDEs, such as Renesas e2 studio and NXP LPCXpresso. These IDEs are all based on the open-source Eclipse framework, and Eclipse has become the mainstream choice for free, open-source MCU IDEs.

Nuclei Studio IDE is an Eclipse-based integrated development environment developed by Nuclei for its RISC-V processor IP core products.

The Eclipse platform is developed under an open-source model and provides a public license (with free source code) and worldwide distribution rights. Eclipse itself is only a framework platform; apart from the runtime kernel of the Eclipse platform, all of its functionality resides in plugins. Developers can extend the platform ' s functionality through plugins from the Eclipse project or through plugins provided by other developers. One plugin can be plugged into another, enabling a maximum degree of integration.

Because the Eclipse IDE is already widely used in the community, for general Eclipse IDE usage that is not tied to specific hardware or a CPU, you can refer to tutorials about the Eclipse IDE written by others. The following tutorial websites are recommended:

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

The Eclipse IDE platform has the following advantages.

- **Large community**

  Since its launch in 2001, Eclipse has built a large community, which provides designers with many resources, including books, tutorials, and websites, to help them work more efficiently with the Eclipse platform and tools. The Eclipse platform and its related projects and plugins can be downloaded directly from the eclipse.org website.

- **Continuous improvement**

Eclipse ' s open-source platform helps developers continuously take full advantage of large-scale resources. Eclipse is continuously improved through the following projects.

  - Platform project —focuses on Eclipse itself.
  - CDT project —focuses on C/C++ development tooling.
  - PDE project —focuses on the plug-in development environment.

- **Open source code**

  Designers always have access to the source code and can fix tool errors themselves, which helps them save time and stay in control of their development work.

- **Compatibility**

  The Eclipse platform is written in Java and can be used on various development workstations such as Windows and Linux. The open-source tools support multiple languages, multiple platforms, and multiple vendor environments.

- **Extensibility**

  Eclipse adopts an open, extensible architecture. It can work with third-party extensions such as ClearCase, SlickEdit, Rational Rose, and other Unified Modeling Language (UML) suites. In addition, it works with various graphical user interface (GUI) editors and supports a wide range of plugins.

Nuclei Studio IDE takes full advantage of these Eclipse IDE strengths, combines mature community plugins such as Eclipse Embedded CDT and Linux Tools, and adds its own plugins to meet the daily needs of RISC-V embedded development:

- Project creation, management, build, and debug
- Multiple debugging solutions, including OpenOCD, J-Link, Nuclei DLink, and Nuclei QEMU
- Extensible debugging solutions, making it easy to extend support for more debuggers
- Multiple compilers, including gcc, clang, and zcc
- **Nuclei Settings**, a quick tool for common project settings
- On-Chip Trace based on the Nuclei ETrace hardware/software solution
- Enhanced profiling and code coverage solutions based on gprof and gcov
- The Nuclei Package (NPK) software package solution, which makes it easy to extend support for software development kits decoupled from Nuclei Studio, enabling a convenient **software package import -> Project Wizard** workflow. This approach has been widely adopted, for example by Nuclei ' s own **Nuclei SDK**.
- Faster ways to open projects and workspaces
- A more user-friendly Launch Bar

2.2 Nuclei Studio Release Notes

Note:

- To confirm the versions of the RISC-V Toolchain, OpenOCD, QEMU, and Nuclei Model integrated in the current Nuclei Studio, check the content of `toolchain/ReadMe.txt`.

2.2.1 2026.10 Release Notes

The 2026.10 version is developed based on Eclipse Cpp 2024-06 and upgrades Nuclei System Technology ' s tool versions to 2026.10. It adds debugging views and features such as the RISC-V extension selector, Flash Erase, Live Peripheral, CPU Information, MMU View, and RISC-V Reference; optimizes existing features such as Connect to Running Target, Trace, Flame View, and Live Watch; adds GDB breakpoint project filter options and the Memory view address exclusion feature (also applicable to Memory Browser) to the underlying Eclipse CDT; and fixes several known defects.

Upgrade RISC-V Toolchain, OpenOCD, and QEMU Versions

- Nuclei Studio 2026.10 integrates Nuclei RISC-V Toolchain 2026.10. For details, see *Version 2026.10* (page 467).
- Nuclei Studio 2026.10 integrates OpenOCD 2026.10. For details, see *Version 2026.10* (page 807).
- Nuclei Studio 2026.10 integrates Nuclei Qemu 2026.10. For details, see *Version 2026.10* (page 859).
- Nuclei Studio 2026.10 integrates Nuclei Near Cycle Model 2026.10. For details, see *Version 2026.10* (page 880).

Added RISC-V Extension Selector

A graphical RISC-V extension selector has been added. You can select RISC-V ISA extensions in the project wizard or on the Settings page of project properties, and it generates the `-march` string automatically; debug and analysis extensions such as `Live Watch` and `Trace` are also selectable and usable in the extension selector. For details, see *Using the RISC-V Extension Selector* (page 222).

Added Flash Erase Feature

The Flash Erase feature has been added; it can erase Flash up to a specified address. For details, see *Flash Erase* (page 327).

Added Live Peripheral Feature

Live Peripheral is a real-time peripheral monitoring tool developed by Nuclei System Technology. It allows you to view and modify SVD peripheral registers in real time during debugging, and shares the same backend as Live Watch. For details, see *Live Peripheral* (page 320).

Added CPU Information View

The CPU Information view has been added. During debugging, you can view information such as MARCHID/ MHARTID/MIMPID/ISA per hart and decode Nuclei configuration CSRs, and use the CPU Fault Analysis tab to locate trap causes. For details, see *CPU Information* (page 195).

Added MMU View Feature

The RISC-V MMU view has been added. It can read `satp`/`misa` and walk Sv32/Sv39/Sv48/Sv57 page tables while the target is halted, providing virtual address translation, a complete memory map, and a page-table tree. For details, see *MMU View* (page 336).

Added RISC-V Reference Feature

The RISC-V Reference view has been added. You can search instructions/CSRs/peripheral registers, encode and decode between machine code and assembly, and interpret register values by bit field. For details, see *RISC-V Reference* (page 345).

Optimized Flame View Feature

In the 2026.10 version, we further optimized Flame View by supporting the display of multi-level nested interrupt flame graphs and improving flame graph generation efficiency. Flame View is an intuitive and efficient performance analysis tool used to visualize program execution call stacks and time distribution in embedded systems, helping developers quickly identify hot functions, analyze execution paths, and locate performance bottlenecks.

For details, see *Using Flame View* (page 272).

Optimized Trace Feature

Based on earlier practical feedback, this release fixes several known defects of the Trace feature and optimizes its processing performance. When connected to a board, it can automatically read registers to fill in the default values of the Trace/Decode settings, and it adds or improves features such as interrupt queries based on Trace data and Flame View. The main enhancements are as follows:

- **Trace bundle (Trace 2.0)**: the Trace 2.0 transport type is supported, and parameters such as TS Size, Datatrace, Context Size, and Decode Mode are read from the CPU and filled in automatically. The raw data, decode results, and related configuration from one capture can be packaged into a single bundle file for easy saving and sharing, and can be replayed offline (with no board connected) in Nuclei Studio to reproduce the complete analysis process.
- **Headless Mode**: trace files can be decoded without opening the Nuclei Studio workbench, by running the built-in headless decoding application with options compatible with `dqr`, which is suitable for batch decoding, scripts, and CI pipelines.

For details on the Trace feature, see *Using Trace* (page 250); for details on Headless Mode, see *Using Trace in Headless Mode* (page 268).

Optimized Live Watch Feature

Live Watch is a real-time monitoring tool developed by Nuclei System Technology to help developers debug and optimize code more efficiently. The main change for Live Watch in Nuclei Studio 2026.10 is support for obtaining variable values through the GDB channel, along with corresponding optimizations to the refresh and plotting experience.

For details, see *Live Watch* (page 306).

ZCC Upgrade

Nuclei Studio 2026.10 integrates ZCC 5.0.0. For details, see: https://www.terapines.com/products/zcc

Added GDB Breakpoint Project Filter Option

When debugging in a multi-project workspace, the breakpoints of all projects were previously installed to GDB, which could easily interfere with the current debugging session. To address this, a new GDB debug preference `Only install breakpoints that belong to the launched project` has been added, so that only the breakpoints of the currently debugged project are installed to GDB, while the breakpoints of other projects are automatically skipped.

For details, see *GDB Breakpoint Project Filter Option* (page 351).

Added Memory View Address Exclusion

The `Address Exclusion Settings...` item has been added to the Memory view (also applicable to Memory Browser), so you can configure memory address ranges that should not be read, avoiding errors when accessing inaccessible or protected regions.

For details, see *Memory View Address Exclusion* (page 353).

Added Big Endian Compiler Support

Nuclei Studio 2026.10 adds support for the RISC-V Big Endian toolchain (this feature is still in the test phase). After the Big Endian toolchain (bengcc) is placed in `<NucleiStudio>\toolchain\gccbe`, projects can be compiled into big-endian code by setting the `Endian` option and the toolchain information in the project properties.

For details, see *Big Endian Compiler Support* (page 354).

Added Register View Filter

A filter box has been added to the Registers view, so you can quickly locate the registers you want to inspect by entering a keyword during debugging. For details, see *Register View Filter* (page 356).

Added Disassembly View C Code Linkage

The Disassembly view is now linked with the C source editor: clicking a line in the C file makes the Disassembly view jump to the corresponding instructions and highlight them, and navigating to another source file makes the Disassembly view follow. For details, see *Disassembly View C Code Linkage* (page 357).

Fixed Known Bugs

- Fixed garbled Chinese characters under Chinese Windows systems, both in the debug console and build console (for example, messages such as "An existing connection was forcibly closed by the remote host.") and during compilation and at runtime;
- Fixed that GDB command history (history command recall) did not work in Debugger Console;
- Fixed the display of the command prompt and the response to specific commands in Debugger Console;
- Fixed the bug that Connect to Running Target would reset the CPU;
- Fixed that running clean on large projects failed because the generated command line (such as `rm` or compile commands) was too long;

- Fixed the issue where modifying the Nuclei Settings page did not trigger automatic synchronization of the model;
- Fixed the issue where QEMU could not run n100 programs properly on the IDE;
- Fixed the incorrect and incomplete layout in the New Project Wizard;
- Fixed the incorrect display of the Language standard option in the project Settings;
- Fixed other known bugs.

2.2.2 2025.10 Release Notes

Version 2025.10 is developed based on Eclipse Cpp 2024-06. It upgrades the Nuclei tools to version 2025.10, adds features such as code debugging, code performance analysis, and IDE online upgrade, optimizes several existing features, and fixes defects present in version 2025.02.

Upgraded RISC-V Toolchain, OpenOCD, and QEMU Versions

- Nuclei Studio 2025.10 integrates Nuclei RISC-V Toolchain 2025.10; see *Version 2025.10* (page 469) for details. In this release, the Windows version switches to Win64, replacing the previous Win32 version, to resolve the problem that large files could not be compiled.
- Nuclei Studio 2025.10 integrates OpenOCD 2025.10; see *Version 2025.10* (page 808) for details.
- Nuclei Studio 2025.10 integrates Nuclei QEMU 2025.10; see *Version 2025.10* (page 860) for details.
- Nuclei Studio 2025.10 integrates Nuclei Near Cycle Model 2025.10; see *Version 2025.10* (page 882) for details.

Added Support for More New Cores

Added support for the N300E core.

Optimized Frequently Used Menus in Nuclei Studio

Some frequently used menus in Nuclei Studio, such as `New Nuclei RISC-V C/C++ Project`, were not shown in the Debug perspective, resulting in a poor user experience. To make them easier to use, the positions of these menus have been optimized.

See *No project creation menu in the Debug perspective* (page 389) for details.

Added the IDE Online Upgrade Feature

Added three upgrade features: online plugin upgrade, online toolchain upgrade, and online IDE upgrade. You can flexibly upgrade each component independently as needed, without replacing or reinstalling the whole package, greatly improving efficiency and system stability.

See *Nuclei Studio upgrade and update* (page 358) for details.

Optimized the IDE Welcome Page

The Welcome page has been optimized to provide quick entries to frequently used documents and tools, making them easier for users to consult and use.

See *Introduction to the Welcome page* (page 29) for details.

Added the Connect to Running Target Feature

To support connecting directly to a development board, Nuclei Studio added the Connect to Running Target feature. This feature allows users to connect directly to a hardware development board and read board-related information, greatly improving development efficiency.

See *Connect to Running Target feature* (page 193) for details.

Added the Flame View Feature

The new Flame View (flame graph view) is an intuitive and efficient performance analysis tool used to visualize the execution call stack and time distribution of programs in embedded systems. By converting complex trace data into hierarchical stacked graphs, Flame View helps developers quickly identify hot functions, analyze execution paths, and locate performance bottlenecks.

See *Using the Flame View feature* (page 272) for details.

Added Support for Nuclei Model Debugging

Nuclei Model, developed by Nuclei, is a dedicated tool for running the Nuclei Near Cycle Model. In this version update, the functionality of the Nuclei Near Cycle Model has been enhanced, and users can perform Debug/Run operations on projects with the Nuclei Near Cycle Model. A new Nuclei Model Debug Configuration view has been added to better support running and debugging.

For the Flame View content in the Nuclei Model Debugger, see *Using Flame View in Nuclei Model* (page 296).

See *Using the Nuclei Model Debugger feature* (page 291) for details.

Optimized the GDB Custom Debug Feature

GDB Custom is a simplified debug configuration developed by Nuclei to accommodate different debugging tools, allowing users to debug programs in Nuclei Studio with their own debugging tools. This version optimizes it on top of the original functionality for a better user experience.

See *GDB Custom debug feature* (page 306) for details.

Optimized the Trace Feature

Based on feedback from previous versions, known bugs in the Trace feature were fixed, the performance of the Trace tool was optimized so that it processes data faster and more stably, and features such as Trace-data-based interrupt query and Flame View were added.

For an introduction to interrupts in the Trace feature, see *Querying interrupts in Trace* (page 260).

For the Flame View content in the Trace feature, see *Using Flame View in Trace* (page 269).

For details on the Trace feature, see *Using the Trace feature* (page 250).

Optimized the Live Watch Feature

Live Watch is a real-time monitoring tool developed by Nuclei, designed to help developers debug and optimize code more efficiently. In Nuclei Studio 2025.10, the Live Watch feature has been optimized, with a higher sampling frequency and drag-to-add-variable support.

See *Using the Live Watch feature* (page 306) for details.

Optimized and Improved the RVProf Feature

RVProf is a performance analysis tool developed by Nuclei based on the CPU cycle model. Starting with version 2025.10, RVProf can be used on Windows. For details, see *Using the RVProf feature* (page 276). This feature requires support from the corresponding NPK package; to try it, please contact us.

ZCC Upgrade

Nuclei Studio 2025.10 integrates ZCC 4.1.7 and includes software libraries supported by Nuclei. For more information, see https://www.terapines.com/products/zcc.

Fixed Known Bugs

- Fixed an issue where opening a file in Lst View could cause the IDE to freeze in certain scenarios;
- Fixed an issue where opening or deleting `*.elf, *.o` files incorrectly reported that the file was in use;
- Fixed an issue where Coverage data was empty in the Linux version of Nuclei Studio;
- Fixed an issue in Flash Programming where binary files outside the current project could not be selected;
- Fixed an error reported when debugging HummingBird projects with OpenOCD;
- Fixed other known bugs.

2.2.3 2025.02 Release Notes

Version 2025.02 is developed based on Eclipse Cpp 2024-06, with the CDT version updated to Eclipse CDT 2024-06. It upgrades the Nuclei tools to version 2025.02, optimizes several existing features, adds features such as debugging and code performance analysis, and fixes defects present in version 2024.06.

Upgraded the Eclipse Cpp Version

Nuclei Studio 2025.02 is developed based on Eclipse Cpp 2024-06. The base CDT version has been upgraded to 11.6.1.

Upgraded RISC-V Toolchain, OpenOCD, and QEMU Versions

Nuclei Studio 2025.02 integrates Nuclei RISC-V Toolchain 2025.02; see https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2025.02 for details.

Nuclei Studio 2025.02 integrates OpenOCD 2025.02; see https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2025.02 for details.

Nuclei Studio 2025.02 integrates Nuclei QEMU 2025.02; see https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2025.02 for details.

Added Support for More New Cores

Added support for the N200E, N202, N202E, NX1000, NX1000F, NX1000FD, UX1000, and UX1000F CPU cores.

Added the Flash Programming Feature

To support downloading compiled binary files directly to a hardware development board, Nuclei Studio added the Flash Programming feature. This feature allows users to quickly and conveniently download compiled binary files directly to a hardware development board, greatly improving development and debugging efficiency.

See *Flash Programming feature* (page 202) for details.

Added the Nuclei NICE Wizard

Nuclei NICE Wizard is a tool integrated into Nuclei Studio that aims to simplify and accelerate the creation of NICE (custom instruction extensions) and VNICE (vectorized custom instruction extensions) instructions.

See *Nuclei NICE Wizard* (page 284) for details.

Added Nuclei Model Support

Nuclei Model, developed by Nuclei, is a dedicated tool for running the Nuclei Near Cycle Model. To provide a simpler and more efficient user experience, the new Model tool was introduced with functionality simplified from RVProf.

See *Nuclei Model* (page 300) for details.

Upgraded the Nuclei Near Cycle Model Version

Nuclei Near Cycle Model is a simulation testing and performance analysis tool independently developed by Nuclei. It helps developers perform the necessary simulation testing and program performance analysis in the early stages of a project. This version fully supports the Nuclei CPU 200, 300, 600, 900, and 1000 series CPUs, on both Windows and Linux.

See *Nuclei Near Cycle Model* (page 300) for details.

Added the Live Watch Feature

Live Watch is a real-time monitoring tool developed by Nuclei, designed to help developers debug and optimize code more efficiently.

See *Using the Live Watch feature* (page 306) for details.

ZCC Upgrade

Nuclei Studio 2025.02 integrates ZCC 3.2.5 and includes software libraries supported by Nuclei. For more information, see https://www.terapines.com/products/zcc.

2.2.4 2024.06 Release Notes

This version is a relatively major upgrade. Version 2024.06 upgrades CDT to Eclipse CDT 2024-06, upgrades the Nuclei tools to version 2024.06, optimizes several existing features, adds features such as debugging and code performance analysis, and fixes defects present in version 2024.02.

Upgraded the Eclipse CDT Version

In Nuclei Studio 2024.06, the base CDT version has been upgraded to 11.6.0, and this version is developed based on Eclipse CDT 2024-06.

Upgraded RISC-V Toolchain, OpenOCD, and QEMU Versions

Nuclei Studio 2024.06 integrates Nuclei RISC-V Toolchain 2024.06; see https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2024.06 for details.

Nuclei Studio 2024.06 integrates OpenOCD 2024.06; see https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2024.06 for details.

Nuclei Studio 2024.06 integrates Nuclei QEMU 2024.06; see https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2024.06 for details.

Added Support for U600 and UX1000

Support for the U600 and UX1000 cores was added alongside the release of these cores.

Optimized NPK Package Management

The management of NPK package dependencies in Nuclei Package Management was optimized to make it easier to use, and some NPK package installation prompts and logs were improved for a better NPK package management experience. See *NPK package management* (page 86) for details.

Note: This version upgrade changed the configuration of NPK package management. NPK packages installed in version 2024.02 or earlier cannot be recognized by Nuclei Studio 2024.06; download and install the NPK packages again.

Added and Optimized Some Compiler Options

In the Properties and Nuclei Settings pages, a new `-Oz` option was added to Optimization Level; support for group libraries was added to the Libraries page of GNU RISC-V Cross C++ Linker. See *Error about missing symbols when linking the C library during project build* (page 374).

Optimized Debug Mode Switching

Nuclei Studio supports multiple debug modes, such as OpenOCD, J-Link, DLink, and Custom, as well as simulators such as QEMU. Debug mode switching has been optimized to make it easier for users to switch between multiple projects and multiple modes. For details, see *Debug mode management* (page 148).

Optimized and Improved DLink Debugging

Nuclei DLink is a RISC-V debugger developed by Nuclei based on RV Link, with many features added on top of RV Link, making it better suited to Nuclei Studio use cases. For details, see *Debugging and running a project with DLink* (page 187).

Integrated the Terapines ZCC Lite Compiler

Terapines ZCC is a high-performance RISC-V compiler developed by Terapines. Nuclei Studio 2024.06 adds support for Terapines ZCC, which can be used directly in the IDE. See *Building the Hello World project in Nuclei Studio* (page 146) for details.

Added the LST View Tool

LST View is an lst file viewer that makes it easy for users to view files in lst format and provides linkage between *.lst files and source code. For details, see *LST View* (page 226).

Optimized and Improved the Gprof Feature

Gprof is a powerful performance analysis tool that helps developers understand the runtime behavior of C/C++ programs. With Gprof, you can obtain call information, call counts, execution time, and more for each function in a program, which is highly valuable for optimizing programs and improving their runtime efficiency. For details, see *Code Coverage and Profiling features* (page 229).

Optimized and Improved the Gcov Feature

Gcov is a tool for testing C/C++ code coverage. Released together with GCC, it works with GCC to test statement coverage, function coverage, and branch coverage of C/C++ files. For details, see *Code Coverage and Profiling features* (page 229).

Added the Call Graph Feature

Call Graph is a tool for analyzing function call relationships. Used together with Gprof, it helps developers quickly understand a program ' s execution process and call relationships. For details, see *Code Coverage and Profiling features* (page 229).

Added Nuclei Near Cycle Model Support

Nuclei Near Cycle Model is a simulation testing and performance analysis tool independently developed by Nuclei. It helps developers perform the necessary simulation testing and program performance analysis in the early stages of a project. For details, see *Simulation performance analysis with Nuclei Near Cycle Model* (page 300).

2.2.5 2024.02.dev Release Notes

This version is a development version (the content at the download link may change at any time). It fixes defects present in Nuclei Studio 2023.10, optimizes some existing features such as the ETrace feature, and adds some new features such as support for N100 and DLink, to better meet customer evaluation and update needs.

Upgraded the Eclipse CDT Version

In Nuclei Studio 2024.02.dev, the base Eclipse CDT version has been upgraded to Eclipse CDT 2023.12.

Added Support for N100

Support for N100 was added alongside the release of the N100 core.

Added a Batch GCC 13 Project Conversion Tool

In Nuclei Studio 2023.10, after upgrading to GCC 13, converting a large number of projects one by one was inefficient. To make this easier for developers, a batch GCC 13 conversion tool is provided. For details, see *Batch converting projects to support gcc 13* (page 216).

Optimized and Improved the Trace Feature

The Trace feature in Nuclei Studio has been upgraded to support single-core applications, SMP multi-core applications, and AMP multi-core applications in OpenOCD mode; for details, see *Using the Trace feature* (page 250). In DLink mode, only single-core applications are supported. The Trace feature requires support from the corresponding CPU IP; to try it, please contact us.

Optimized and Improved the RVProf Feature

RVProf is a performance analysis tool developed by Nuclei based on the CPU cycle model. For details, see *Using the RVProf feature* (page 276). This feature requires support from the corresponding NPK package; to try it, please contact us.

Added Support for DLink Debug

DLink is a debugging solution independently developed by Nuclei and is supported in this version. This feature requires a DLink debugger; to try it, please contact us.

2.2.6 2023.10 Release Notes

This version is a relatively major upgrade. It integrates the Nuclei 2023.10 versions of the Toolchain, QEMU, and OpenOCD; the Eclipse CDT version has been upgraded; and GCC has also undergone a major iteration to GCC 13. Many new features have been added to NPK to support project creation with GCC or Clang, and many new Configuration field types have been added to allow more flexible project configuration in the Project Wizard.

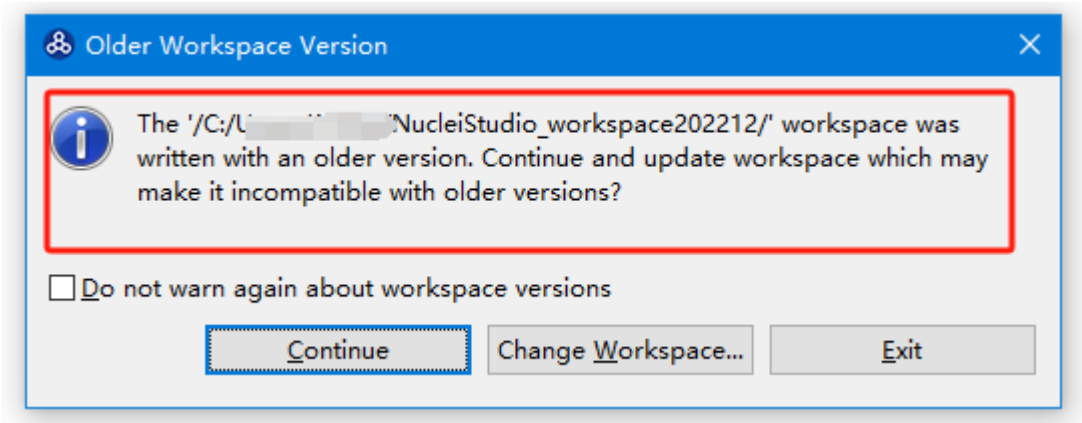
Upgraded the Eclipse CDT Version

In Nuclei Studio 2023.10, the base Eclipse CDT version has been upgraded to Eclipse CDT 2023.06. Eclipse CDT 2023-06 is the Eclipse Foundation ' s second-quarter 2023 simultaneous release, with 64 participating projects, released on June 14, 2023.

Reference: Eclipse IDE for C/C++ Developers[1]

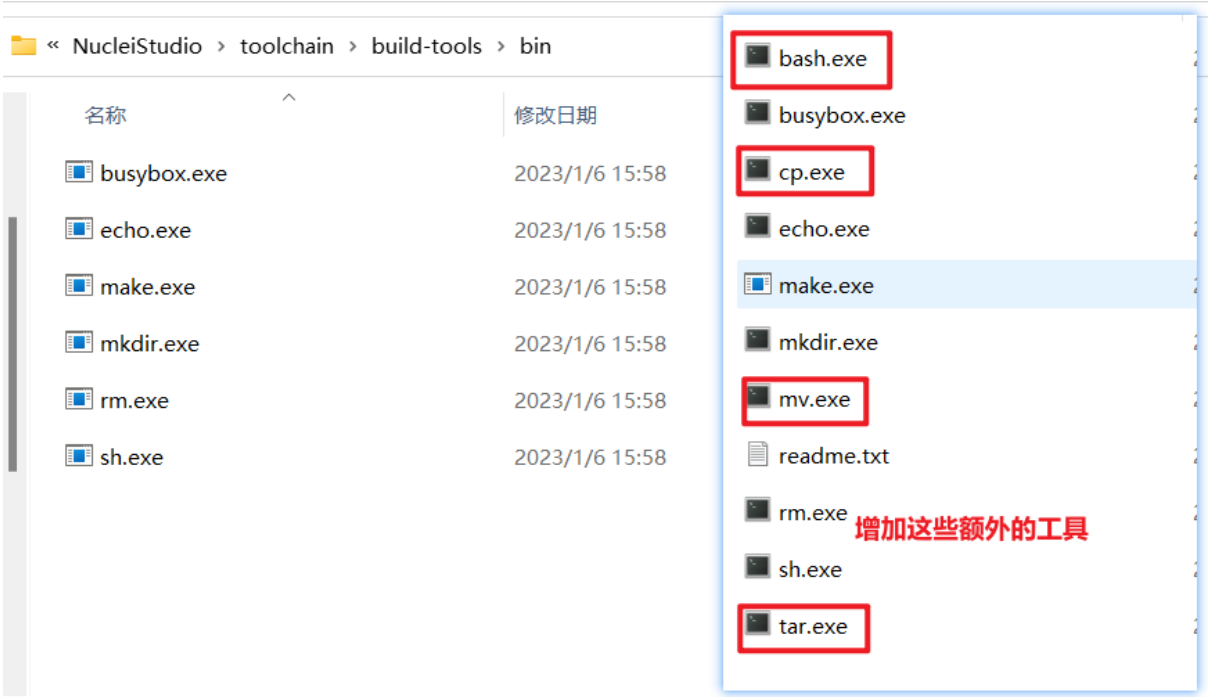
After the upgrade, opening a workspace created by a previous version displays an incompatibility warning, and anomalies may occur during use. Switching to a new workspace directory is recommended.

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



Upgraded the build-tools Version

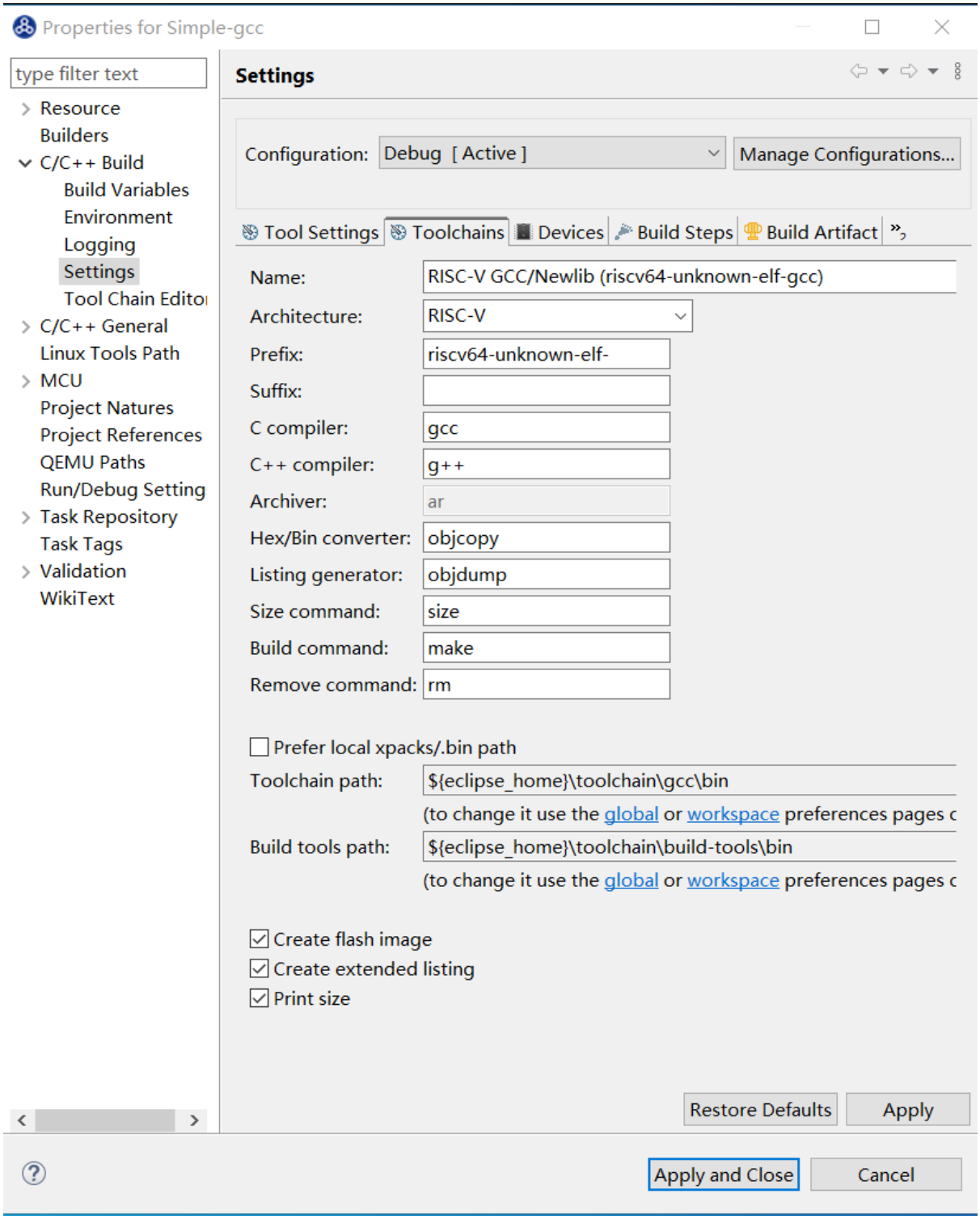
In Nuclei Studio 2023.10, the build-tools in the toolchain have been updated to version 4.4, and the `bash.exe`, `cp.exe`, `mv.exe`, and `tar.exe` tools have been added.

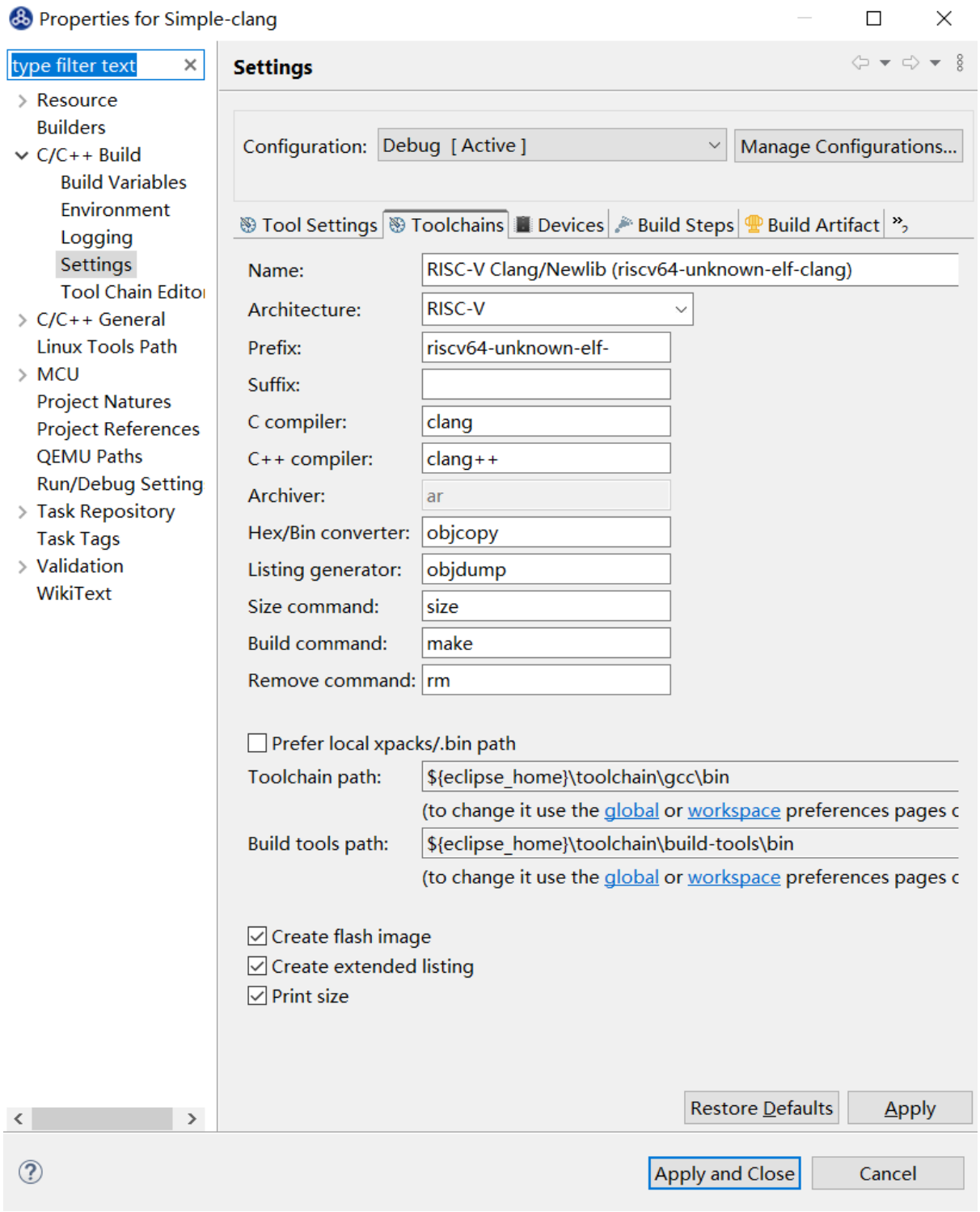


Support for GCC 13 and Clang 17

Nuclei Studio 2023.10 adds support for GCC 13. Compared with the previous gcc10 version, GCC 13 provides more complete support for RISC-V instruction extensions, and the version maintained by Nuclei supports the full RVV Intrinsic API v0.12. Nuclei Studio 2023.10 also adds support for Clang 17 (reference: https://releases.llvm.org/17.0.1/docs/RISCVUsage.html). If you still want to use GCC 10 for project development, the related configuration has been kept, but the toolchain is not integrated into the IDE; download it yourself and place it in the gcc10 directory (see the README.txt inside). A migration path is also provided for upgrading projects created with older versions of Nuclei Studio that use gcc10 to the gcc13 toolchain; for details, see *Importing projects created by older versions of Nuclei Studio* (page 214). For a more detailed description of Nuclei RISC-V Toolchain 2023.10, see https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2023.10


« NucleiStudio_IDE_202310-win64 › NucleiStudio › toolchain
在 too
名称
修改日期
类型
大小
build-tools
2023/10/30 14:47
文件夹
gcc
文件夹
GCC 13和 Clang所在的目录
gcc10
2023/10/30 14:50
文件夹
GCC 10的目录
openocd
2023/10/30 14:50
文件夹
qemu
2023/10/30 14:51
文件夹
ReadMe.txt
2023/10/30 14:25
TXT 文件
1 KB


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

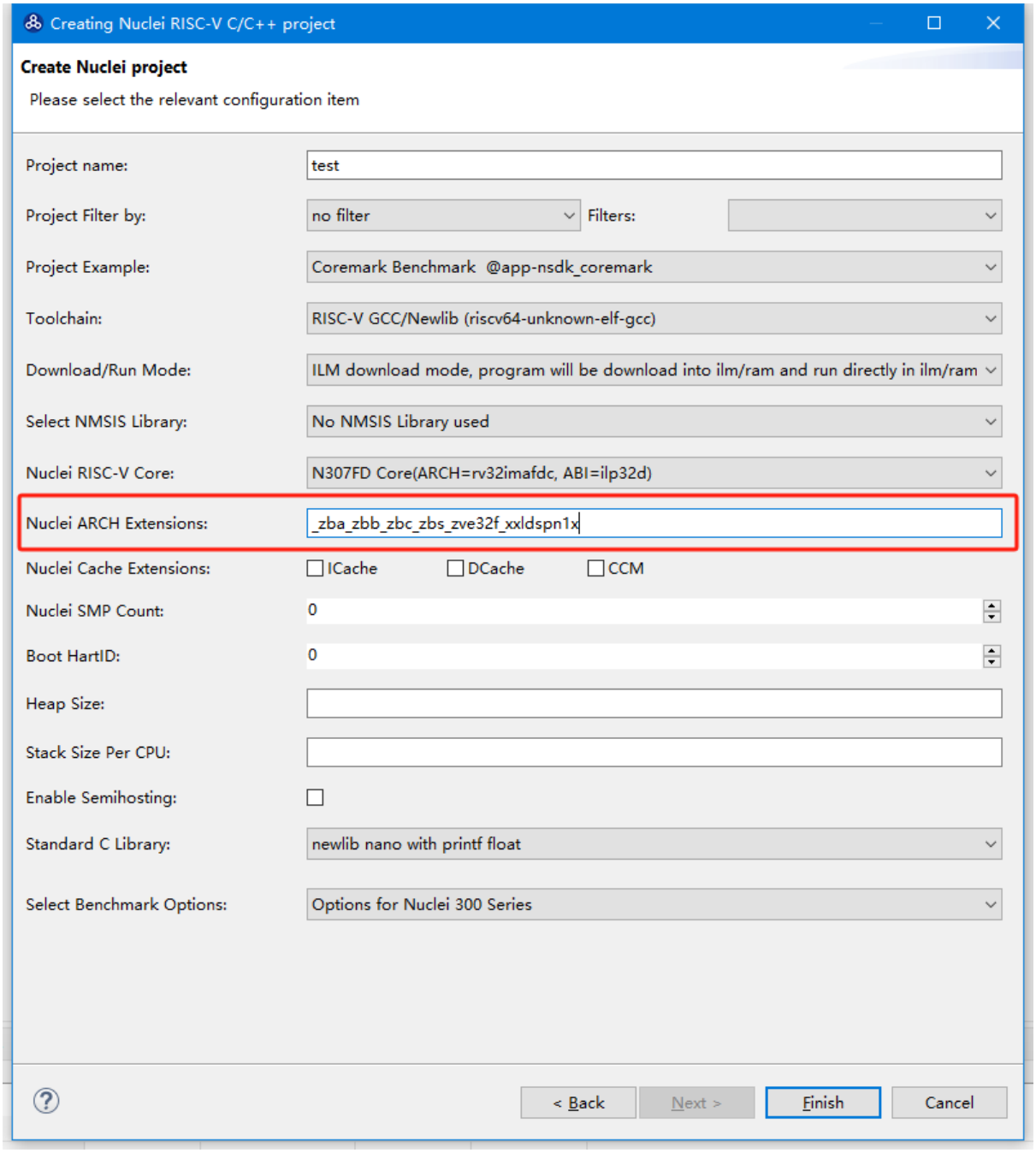

Properties for Simple-clang
type filter text
Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
Tool Chain Editor
C/C++ General
Linux Tools Path
MCU
Project Natures
Project References
QEMU Paths
Run/Debug Setting
Task Repository
Task Tags
Validation
WikiText
Settings
Configuration: 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

Changes in RISC-V Instruction Extension Usage

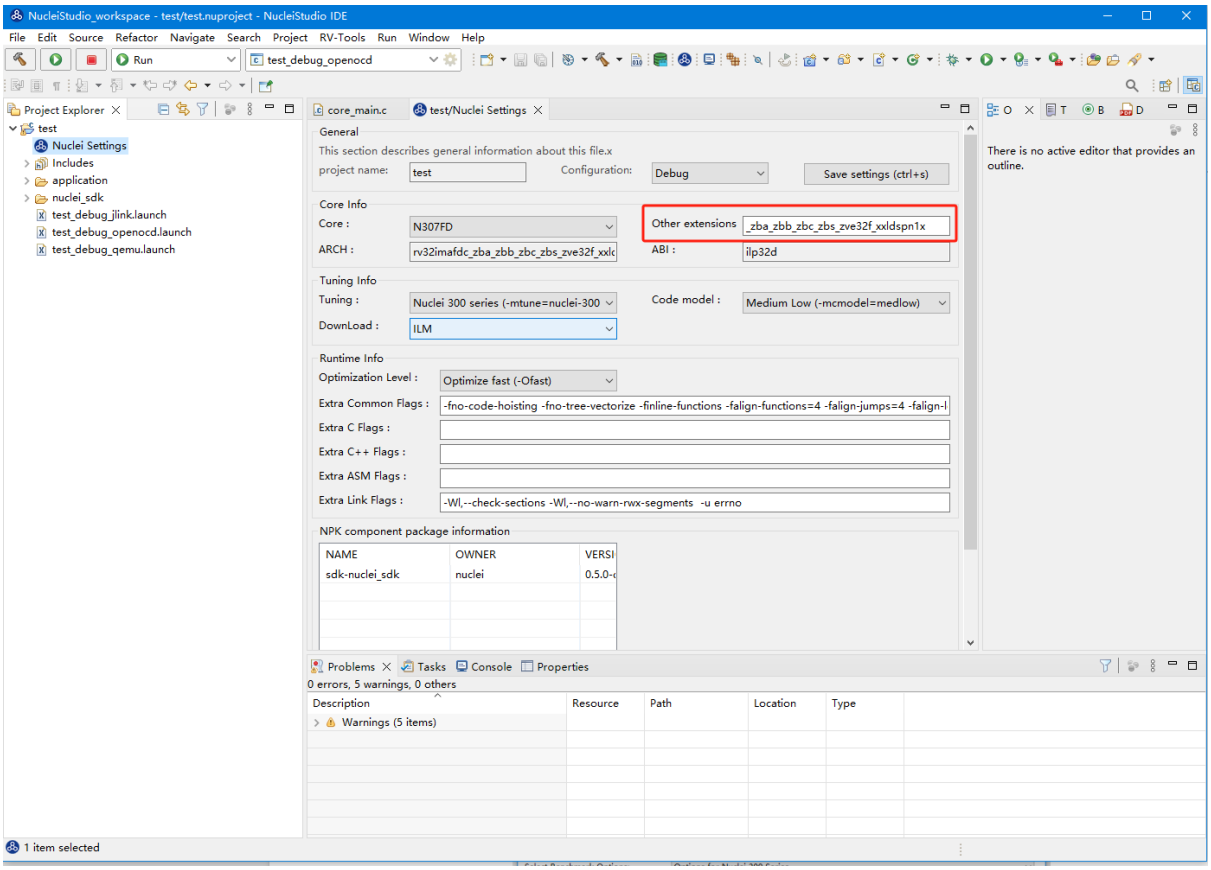
Due to changes in GCC and Clang, extension usage has changed significantly. The mapping between the original bpkv extensions and the new rules is as follows. For a more detailed description, see https://doc.nucleisys.com/nuclei_sdk/develop/buildsystem.html#arch-ext

- `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`

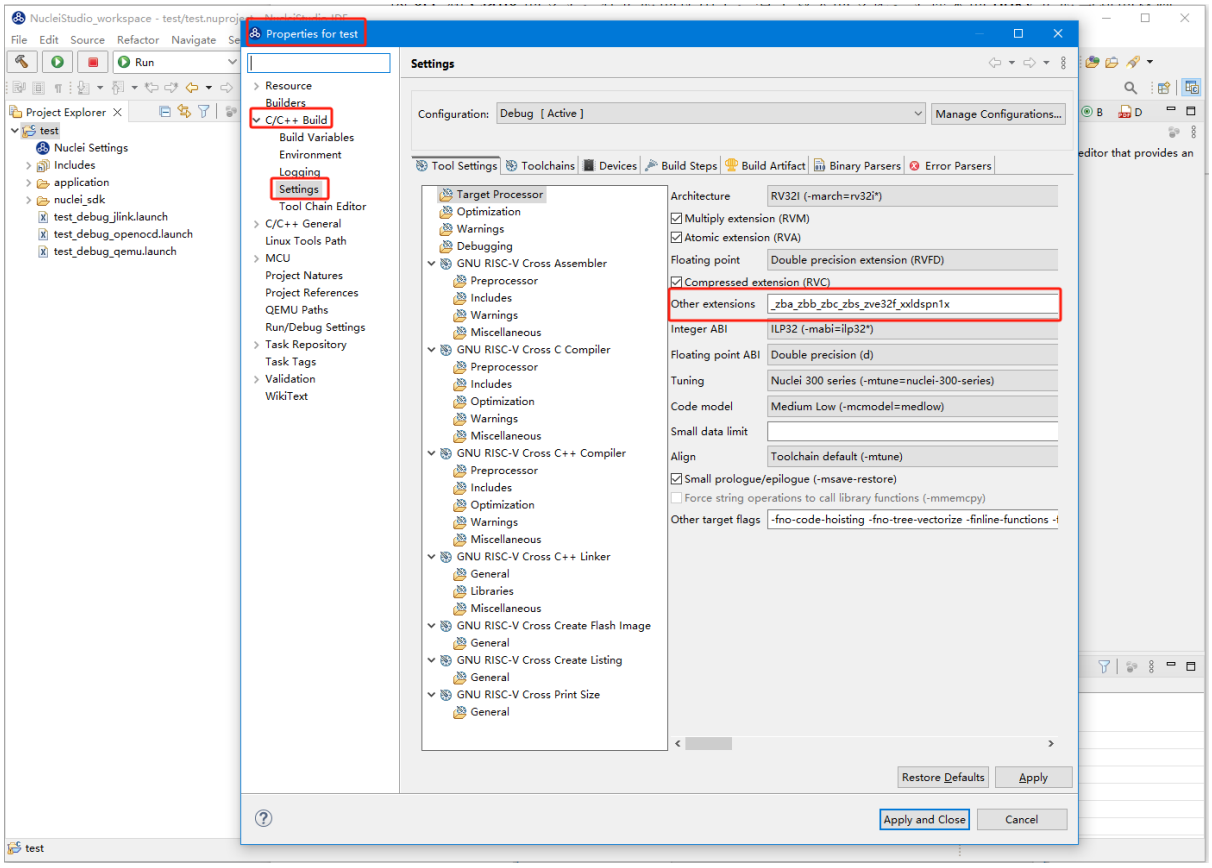
Take N307FD + B + V + Nuclei DSP with the N1 extension as an example: to create an application that uses extensions, enter the correct extension string in Nuclei ARCH Extensions in the project creation wizard. For example, to use the bpv extension, enter `_zba_zbb_zbc_zbs_zve32f_xxldspn1x` according to the rules above.



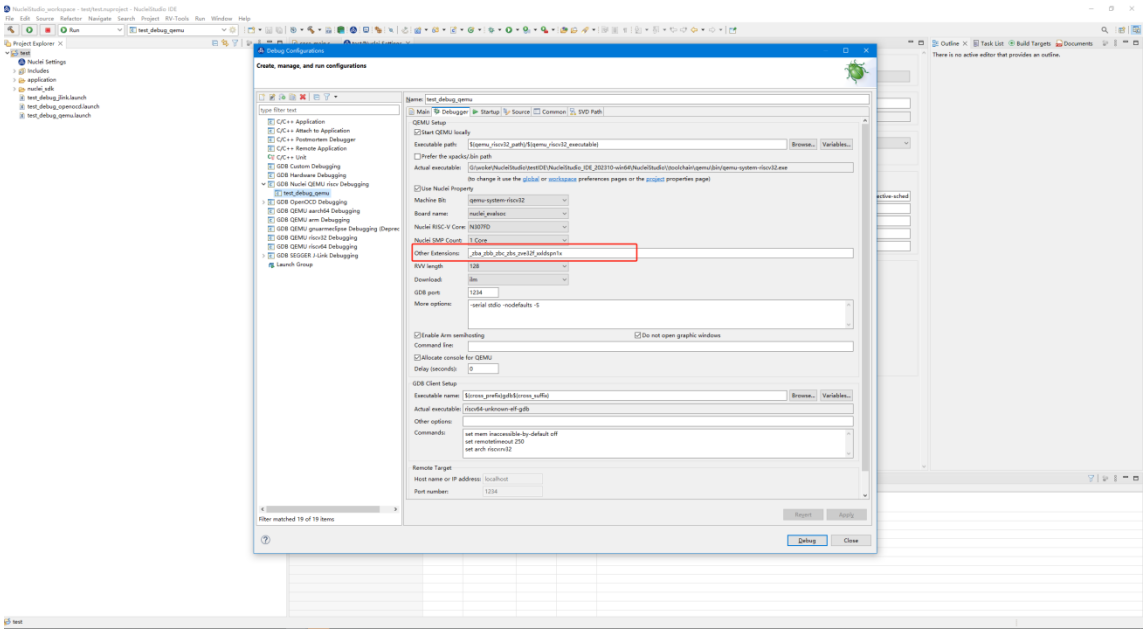
In the generated project, you can view the project ' s **Nuclei Settings**.



Similarly, the project properties also contain configuration items related to RISC-V instruction extensions under `C/C++ Build->Settings->Target Processor`.

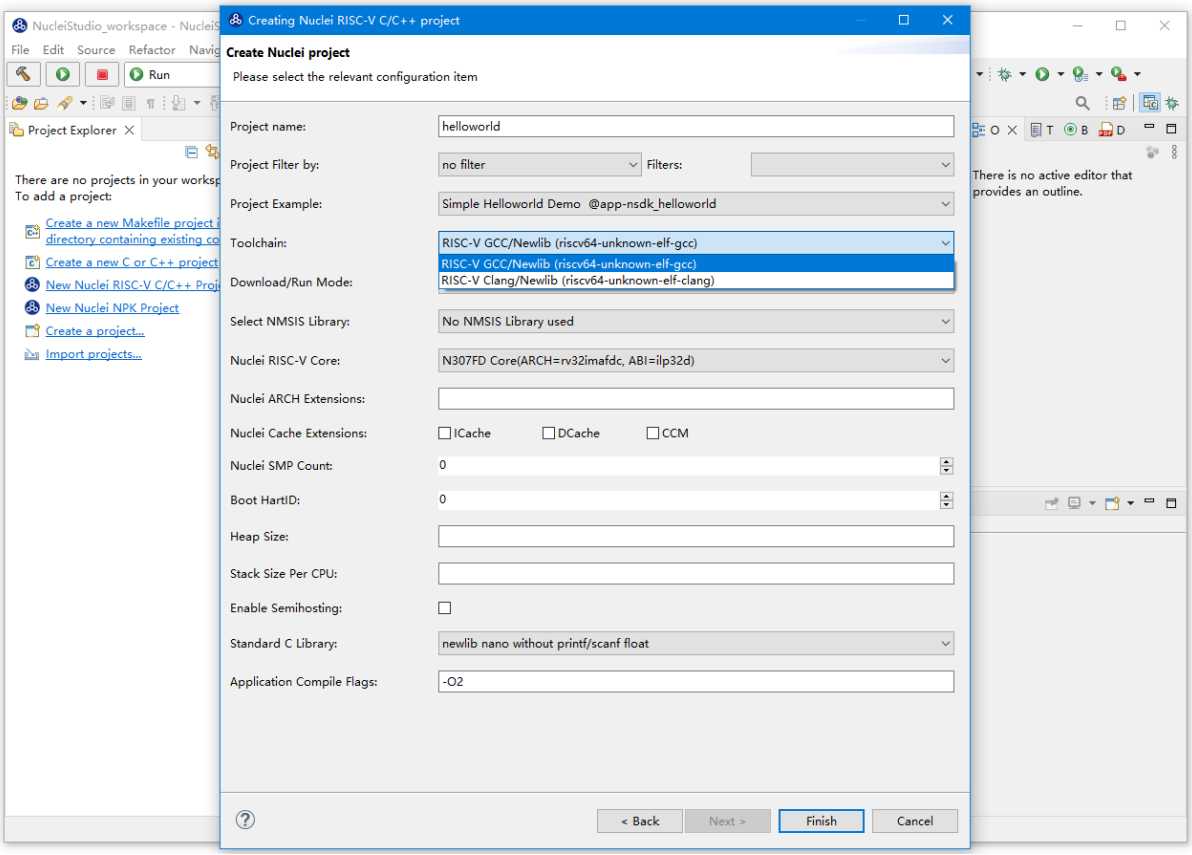


The QEMU configuration also has corresponding RISC-V instruction extension configuration items.

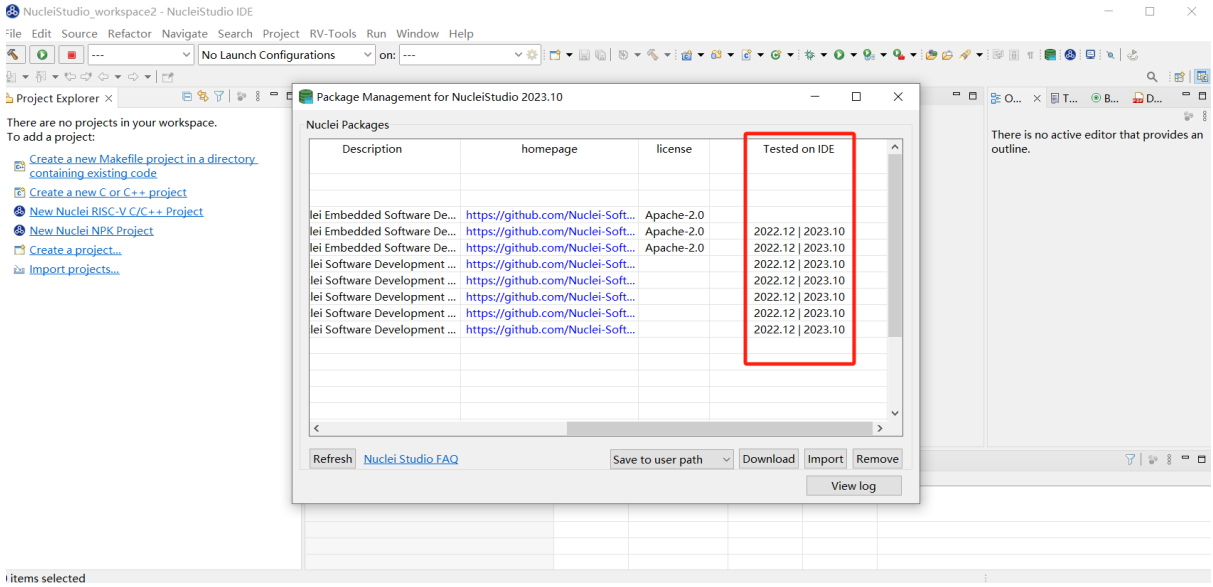


Changes in NPK Package Usage

To support GCC 13 and Clang 17, the Nuclei SDK package has been upgraded to version 0.5.0. When creating a project with the SDK package, users can choose to create a GCC 13 or Clang 17 project as needed. Because the version changes are significant, some features of SDK versions earlier than 0.5.0 may not work properly in Nuclei Studio 2023.10, so a migration tool is provided to help you quickly migrate and upgrade your projects. **Back up your old projects first**. For details, see *Importing projects created by older versions of Nuclei Studio* (page 214).



In addition, Nuclei Studio 2023.10 validates the compatible version of online NPK component packages (uploading a package requires specifying the Nuclei Studio version on which it was tested). The Nuclei Studio version that each component package is compatible with is shown on the Package Management page, and a prompt is displayed if the version does not match during download and installation. However, no prompt is given when importing offline packages; verify yourself whether the package is supported by the Nuclei Studio IDE version you are using, as shown below.



Upgraded OpenOCD

OpenOCD has been upgraded to version 2023.10, with some additional debugging features added, such as viewing CPU information and experimental etrace support. For a more detailed description of the OpenOCD changes, see https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2023.10

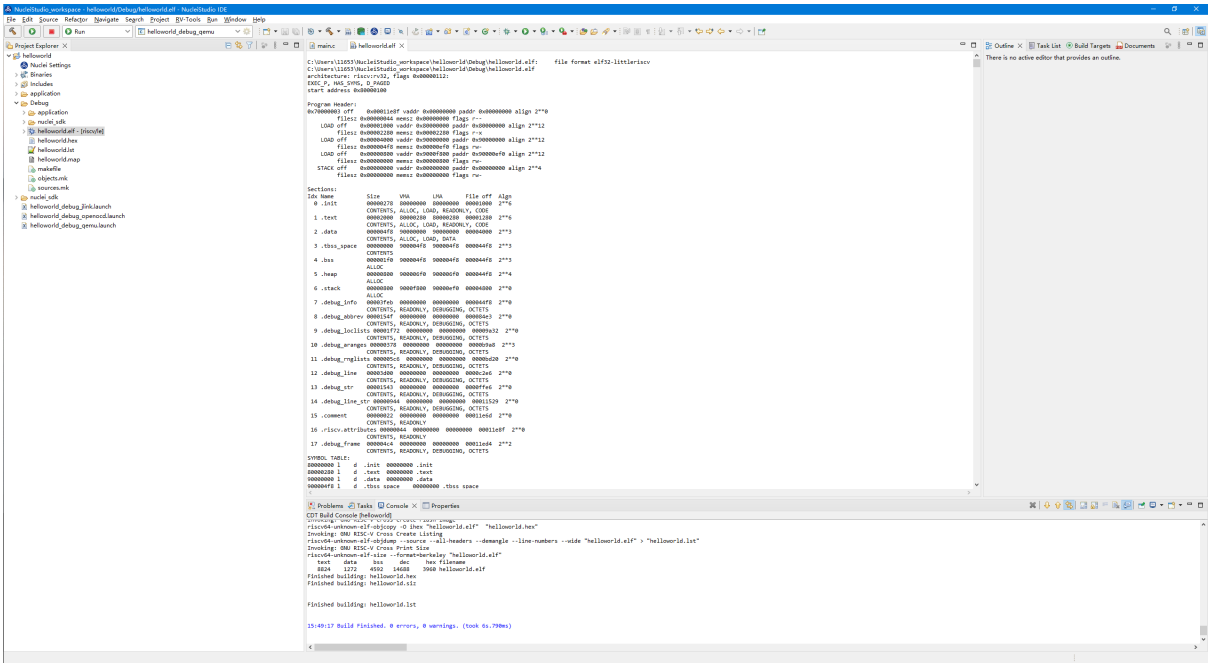
Upgraded QEMU

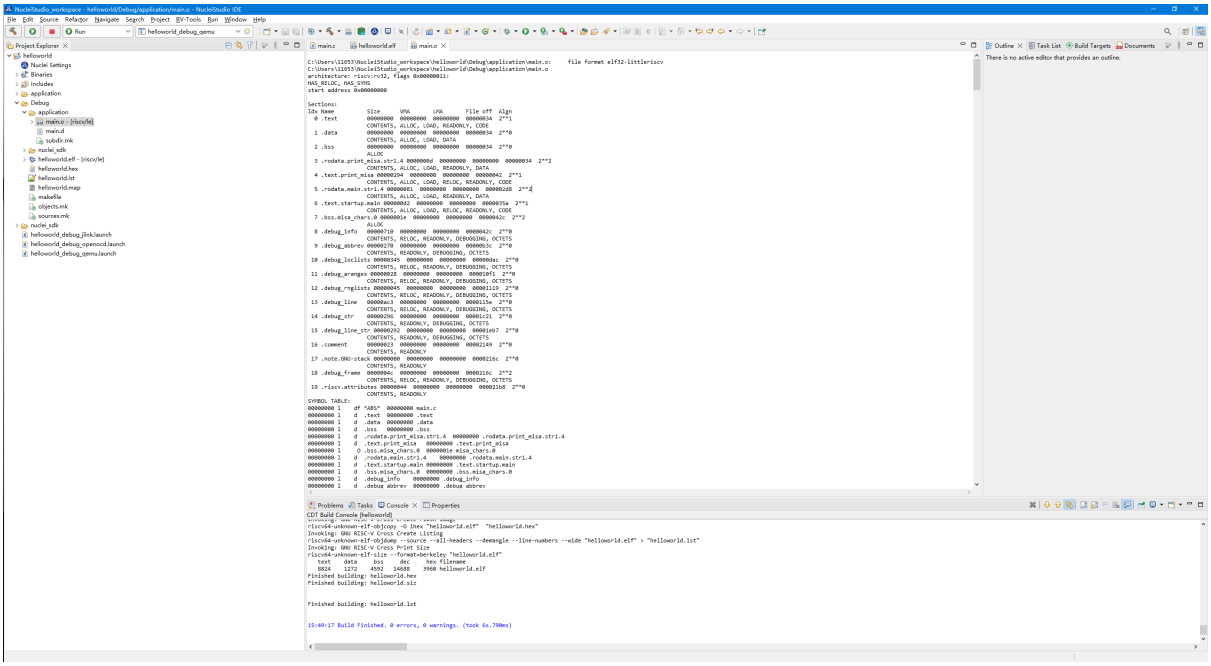
Nuclei Studio 2023.10 integrates Nuclei QEMU 2023.10, which is developed based on QEMU 8.0 (reference: https://wiki.qemu.org/ChangeLog/8.0). This version of QEMU differs significantly from version 2022.10 in terms of usage: it no longer supports the `gd32vf103_rvstar` board and now supports only Nuclei EvalSoC, which can be configured to work seamlessly with Nuclei SDK/Nuclei Linux SDK. The supported machine has also been unified from `nuclei_n`/`nuclei_u` to `nuclei_evalsoc`. For a more detailed description of Nuclei QEMU, see https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2023.10



Added an ELF File Viewer

Nuclei Studio 2023.10 adds an ELF file viewer, making it easy for users to view the compiled `.elf` and `.o` files.





Added Code Coverage and Profiling Features

Nuclei Studio 2023.10 adds support for the Code Coverage and Profiling features. For details, see *Code Coverage and Profiling features* (page 229).

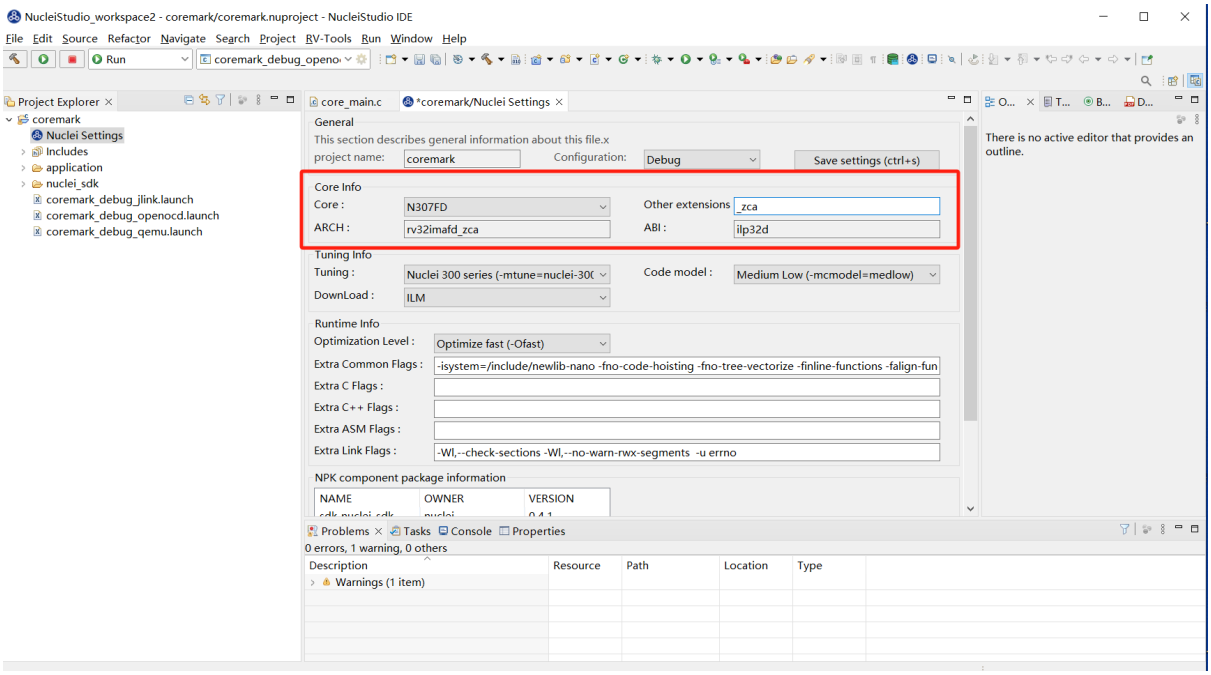
Added the Trace Feature

Nuclei Studio 2023.10 **experimentally** adds the Trace feature. Using this feature requires a CPU with Nuclei Trace IP; to try it, please contact us.

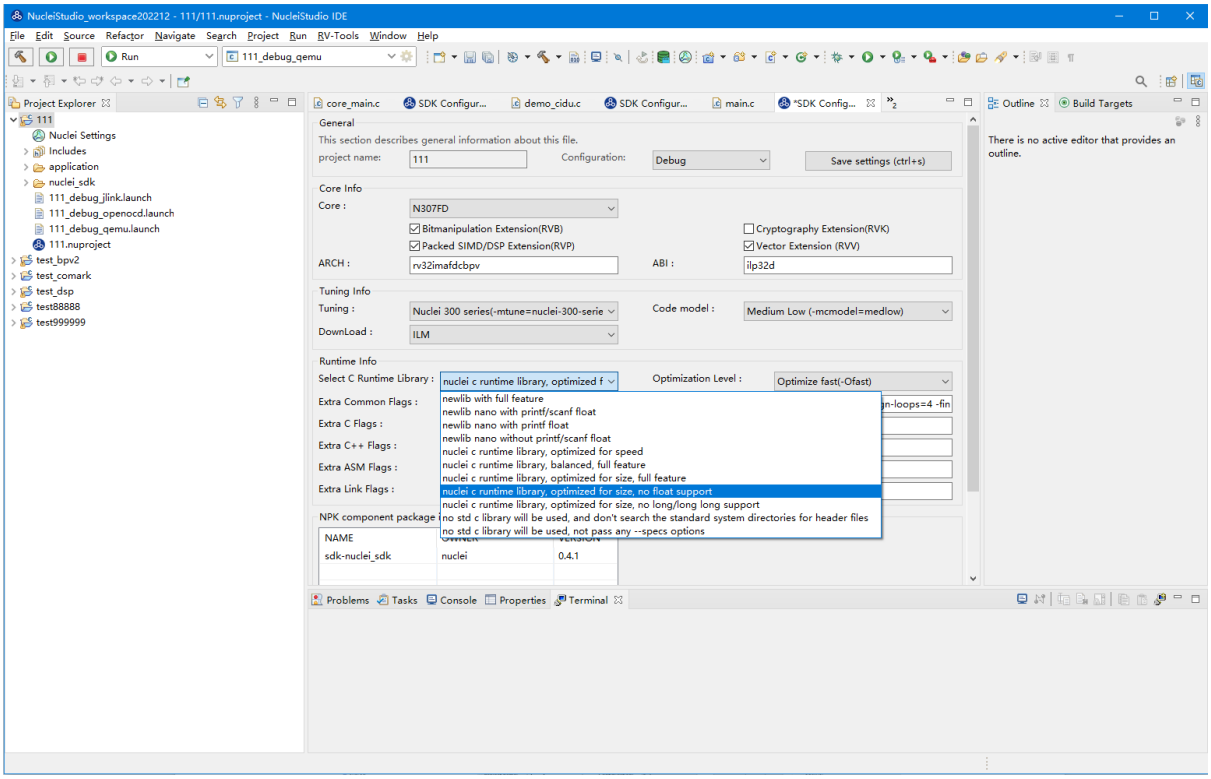
Nuclei Settings Feature Optimization

To accommodate more personalized configuration, some Nuclei Settings functionality was modified.

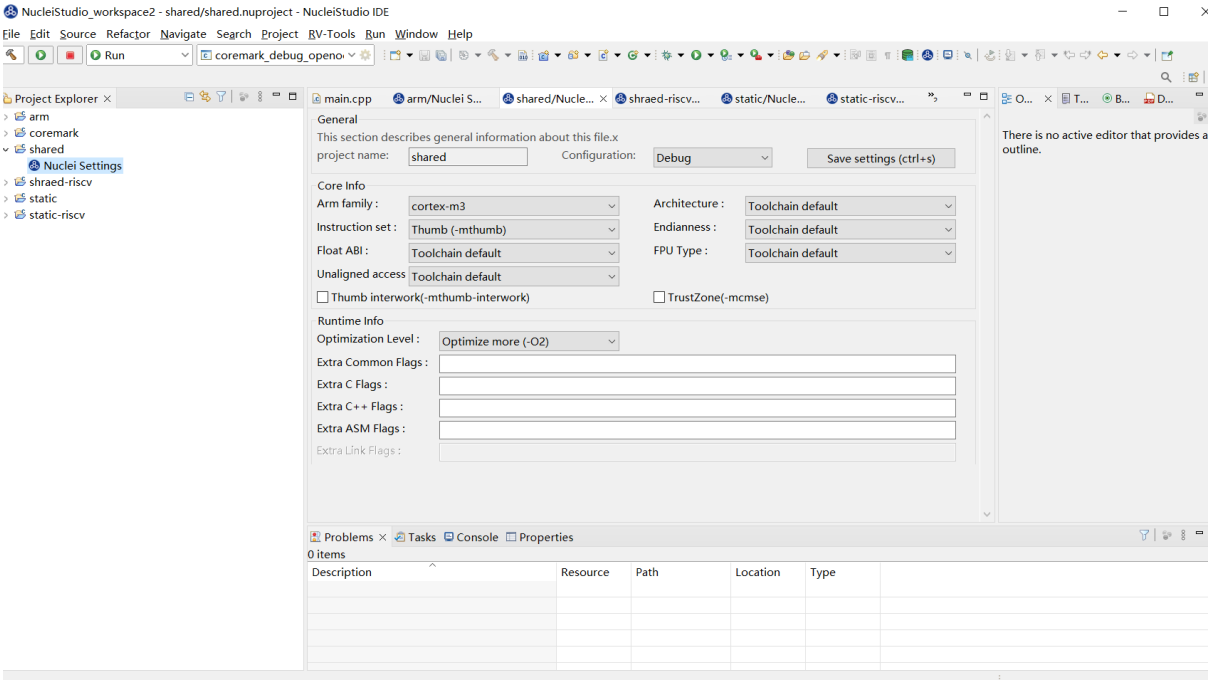
Nuclei Studio 2023.10 removes the original B/P/K/V radio buttons and replaces them with an Other Extensions input box, which users can fill in according to their own needs. For the use of B/P/K/V, see *Changes in RISC-V instruction extension usage* (page 19).

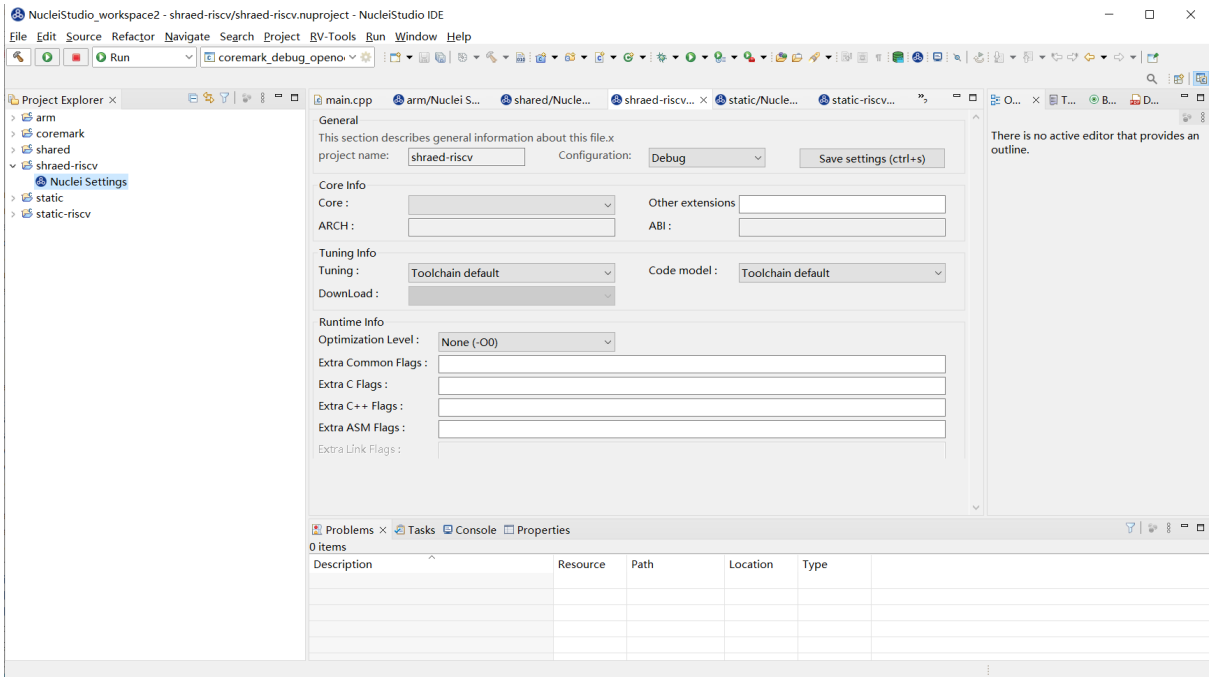


Nuclei Studio 2023.10 removes the original Select C Runtime Library radio buttons. If a project needs this setting, use the `--specs=` option in the project configuration or the Libraries option instead.



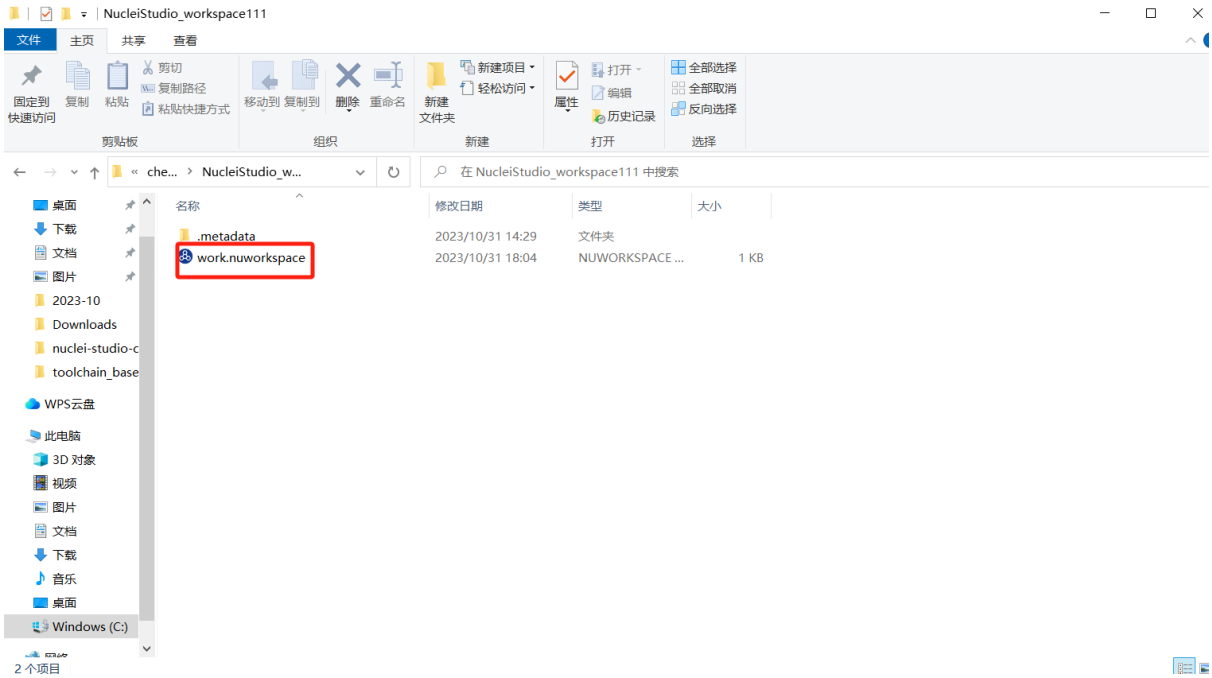
Nuclei Settings is now more versatile: in addition to quickly modifying Nuclei projects, it supports static and shared library projects created with generic RISC-V and Arm toolchains. The figure below shows the shared library case.





Added Quick Open for a Specified Workspace

Similar to double-clicking a `*.nuproject` file in a project to quickly open Nuclei Studio and import that project, Nuclei Studio now creates a `work.nuworkspace` file in workspace directories that have been used. Double-clicking this file opens Nuclei Studio directly. This feature is currently supported only on Windows; to use it, run `install.bat` on Windows after extracting the IDE to set up the file association.



2.2.7 2022.12 Release Notes

Since version 2021.09, Nuclei Studio has completely separated the IDE from the SDK and adopted the brand-new Nuclei Package (NPK) package management approach to manage and use template projects, making it easy for users to import different SDKs and create example projects in the IDE. The SDKs for Nuclei SDK, HBird SDK, and Nuclei ' s SoC IP products can all be packaged as Zip files and imported through Nuclei Package Management.

2.3 Nuclei Studio Download and Installation

2.3.1 Nuclei Studio IDE Download

To help users get started quickly, this document recommends using the pre-packaged Nuclei Studio IDE software archive, which Nuclei has uploaded to the company website at https://www.nucleisys.com/download.php.

Go to the Download Center on the Nuclei website and download the Nuclei Studio archive for Windows or Linux, according to your development environment (note: the content of the Download Center is continuously updated; you can use the latest version or continue using the current version).

Nuclei Studio has been verified and tested on 64-bit Windows 10, Ubuntu 18.04/20.04, and 64-bit Red Hat 7.6; these system versions are recommended.



2.3.2 Nuclei Studio IDE Installation

After downloading the Nuclei Studio IDE archive, extract it; the extracted files are described below.

- Nuclei Studio software package
  - This package contains the Nuclei Studio IDE software. Note: The specific version and file name may change over time.
- HBird_Driver.exe (no longer provided since version 2021.02)
  - **Windows version only.** This file is the USB driver installer for the Nuclei HummingBird debugger.
  - When using this debugger in a Windows environment, you need to install this driver so that the USB device can be recognized by the system.
  - This is because the OpenOCD update in version 2021.02 introduced driver-free functionality.
- SerialDebugging_Tool (no longer provided since version 2021.02)

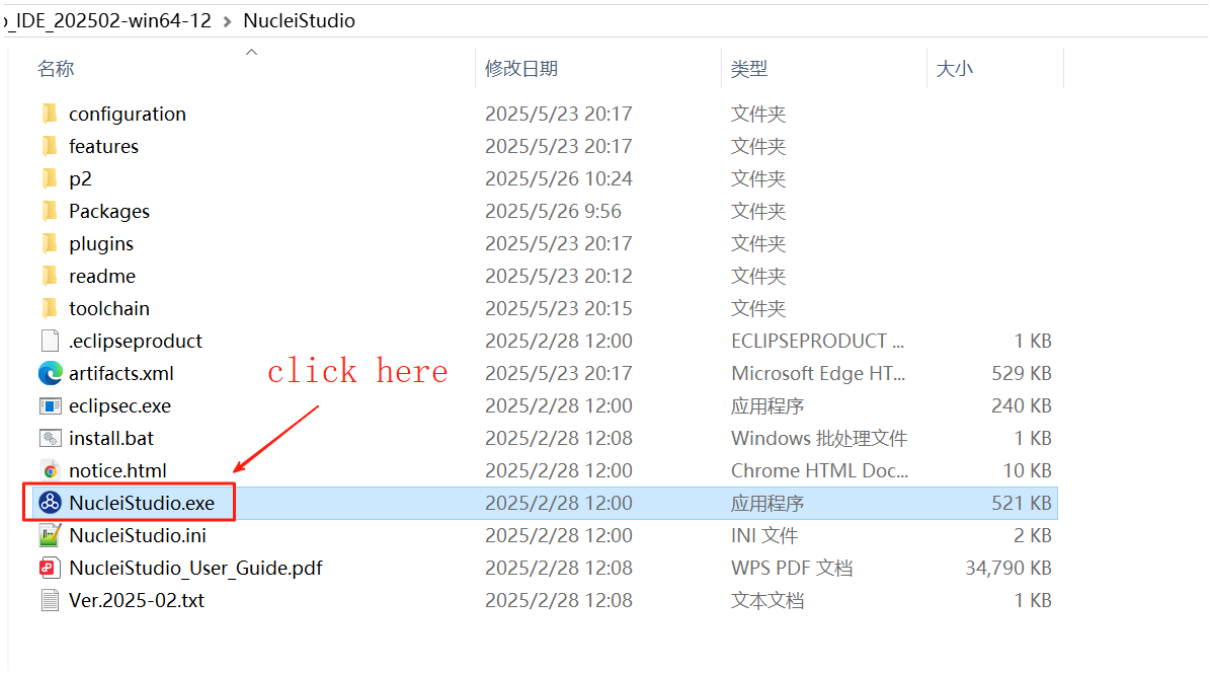
– **Windows version only.** This file is a serial port debugging assistant tool. Use it to print information over the serial port when debugging the software examples later.



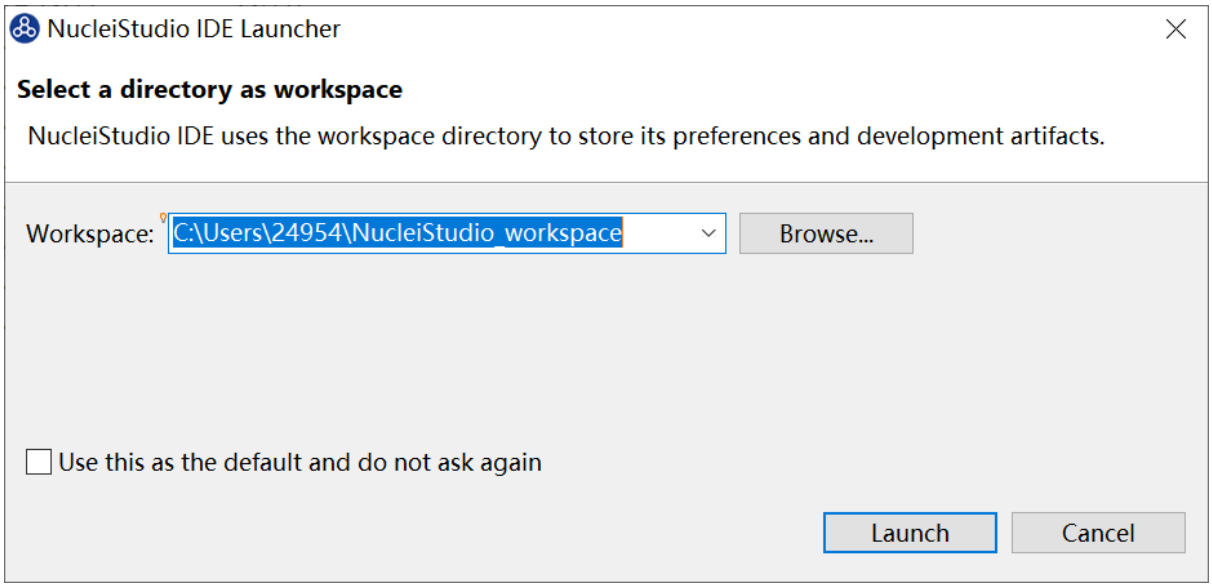
2.3.3 Launching Nuclei Studio IDE

To launch Nuclei Studio (the same steps apply on both Windows and Linux):

Double-click the executable file under the Nuclei Studio folder in the Nuclei Studio IDE package to launch Nuclei Studio.



When you launch Nuclei Studio for the first time, a dialog prompts you to set the Workspace directory path, which is used to store the project files created later.

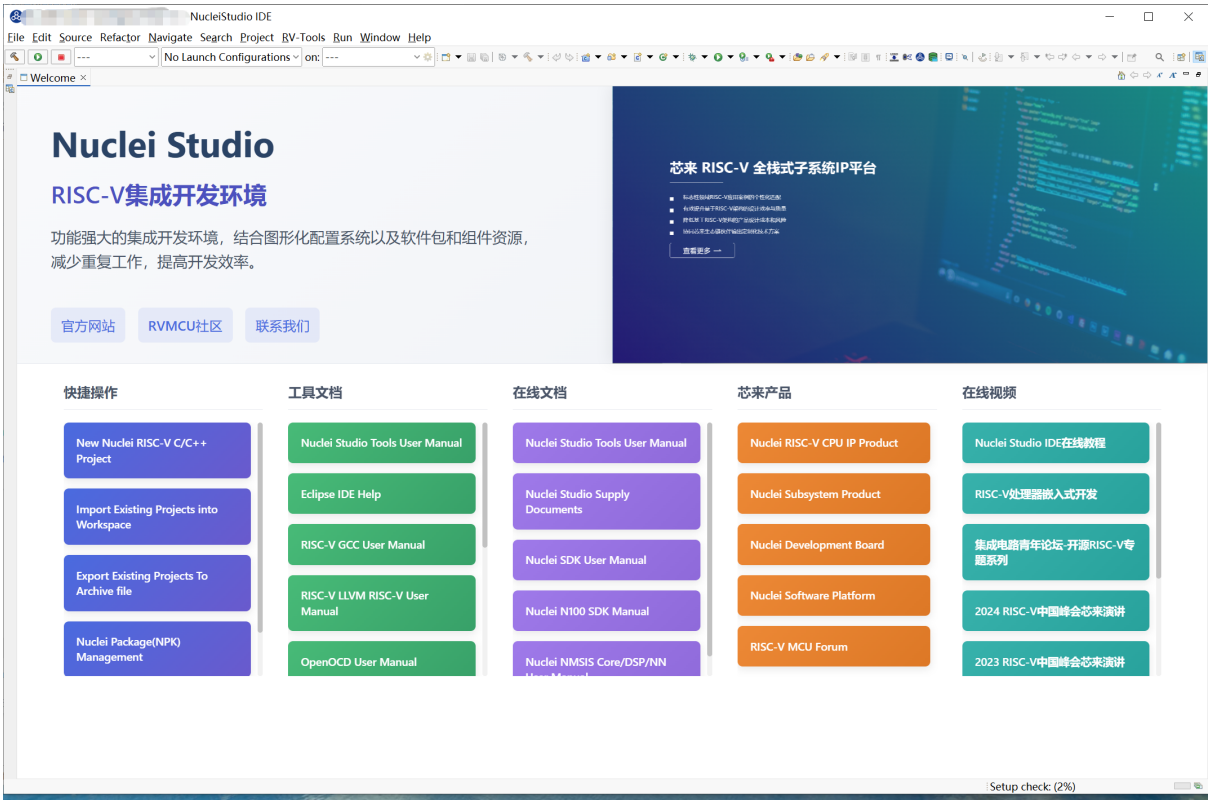


After setting the Workspace directory, click the Launch button to launch Nuclei Studio.

Note: Nuclei Studio 2025.10 has updated the content of the Welcome page.

When Nuclei Studio is launched for the first time, the `Welcome` page is displayed. This page provides quick entries to frequently used functions, including project creation, example project import, and software package (NPK) management, and integrates technical documentation links and tutorial videos provided by Nuclei to help developers get started quickly and master the development process.

To reopen the `Welcome` page later, select `Help → Welcome` from the menu bar.



2.4 Nuclei Studio NPK Overview

Nuclei RISC-V embedded software development introduces the concept of the Nuclei Package (NPK), inspired by npm package management. Packages make it easy for developers to develop, use, and distribute well-designed software components, greatly speeding up software iteration.

A package consists of the following files:

- Package description file (based on YAML): `npk.yml`
- Package-related code and documentation

A package is distributed as a ZIP file. Before a package can be imported and used, the IDE or other tools must verify its integrity and usability.

Note:

- We provide an open-source NPK package checking tool; see https://github.com/Nuclei-Software/npk-checker
- The best reference projects for NPK packages are Nuclei SDK and the other packages available online (search for the npk.yml file): https://www.rvmcu.com/nucleistudio-npk.html

- A package signing mechanism will be provided in the future: packages will be signed before release, and the signature will be verified when a package is imported, so that only packages published with a legitimate identity are imported and untrusted packages are rejected.

2.4.1 Package Description File (npk.yml)

Packages are divided into the following major types:

- **csp:** Core Support Package, e.g. NMSIS
- **ssp:** SoC Support Package, e.g. the SoC support package for the gd32vf103
- **bsp:** Board Support Package, e.g. the board-level support package for the rvstar development board
- **osp:** RTOS Support Package, e.g. various RTOS support packages
- **app:** Application Package, e.g. various high-level applications
- **mwp:** Middleware Package, e.g. third-party middleware such as speech recognition and algorithm libraries
- **sdk:** Software Development Kit, a predefined software development package that generally contains CSP, SSP, BSP, and APP packages, and optionally OSP and MWP packages
- **bdp:** Bundle Package, a group of packages; currently only valid for app

The following special types are used for tool packages and template packages:

- **tool:** Tool Package, e.g. various tool component packages, which may contain other files that need to be referenced
- **tpp:** Template Package, a template type used to create template projects for csp/ssp/bsp/osp/app/mwp/sdk. This type is special: its description file is named **npk_template.yml**

Common Description File Definitions

The following example shows a package description file with detailed annotations:

```
## Package Base Information
name: your_package_name                 # <MUST> Package name ID, no spaces, must␣
↪follow C naming conventions, English name, unique name ID, used for dependency␣
↪management
                                        # Naming rules for name: unique, all␣
↪English, no special characters
                                        # csp type package: csp-xxxx
                                        # ssp type package: ssp-xxxx
                                        # bsp type package: bsp-xxxx
                                        # osp type package: osp-xxxx
                                        # app type package: app-xxxx
                                        # mwp type package: mwp-xxxx
                                        # sdk type package: sdk-xxxx
                                        # tpp type package: tpp-xxxx
                                        # tool type package: tool-xxxx
description: Brief package description  # <MUST> Describe the main features and␣
↪functionality of the package in one sentence, English only, within 20 characters
details: Detailed package description   # <MUST> Clearly describe the main␣
↪features and functionality of the package, English only
version: 1.2.3                          # <OPTIONAL> Optional; if not set,␣
↪defaults to empty. SEMVER 2.0 versioning is recommended, must start with a digit,
↪ e.g. 1.2.3
                                        # <MUST> A version tag is mandatory for␣
↪sdk type packages
type: ssp                               # <MUST> Package type; available types are␣
```

(continues on next page)

(continued from previous page)

```
↪csp, ssp, bsp, osp, app, mwp, sdk, tpp, tool
os: win64                                # <MUST> OS type, e.g. win32, win64, lin64,
↪ lin32; however, the package upload page currently only supports win64 and lin64.
↪ This field only exists in tool type packages
category:                                # <OPTIONAL> Package category, a secondary␣
↪classification based on the package types above, mainly used to classify packages
keywords:                                # <MUST> Package keywords, mainly used for␣
↪indexing and searching
  - soc
  - risc-v
  - nuclei
license: Apache-2.0                      # <OPTIONAL> The license used by the␣
↪package code; available options include GPL, GPLv2, MIT, Apache License v2, BSP,␣
↪etc.
owner: nuclei                            # <MUST>    The owner of the package, used␣
↪for permission matching later
                                         # After integration with the website,␣
↪users must first register a unique owner name before publishing a new package,
                                         # then apply for a package name. A new␣
↪package with that name can only be published if the name is available; the␣
↪number of packages a single user can publish is also limited.
contributors:                            # <OPTIONAL> List of code contributors of␣
↪the package
    - author_name, author_email
homepage:                                # <OPTIONAL> Homepage of the package, if␣
↪any; must be a link

## Package Information

packinfo:                                # <MUST> Only for bsp/ssp type, optional␣
↪for other types. Used to describe SoC-level information
  core_vendor: Nuclei                    # <MUST> only for ssp, the English name of␣
↪the processor core vendor. If the core is based on a Nuclei processor core, fill␣
↪in Nuclei
  vendor: Nuclei                         # <MUST> For ssp/bsp, <OPTIONAL> for others
                                         # For ssp, the English name of the SoC or␣
↪chip vendor; fill in the corresponding vendor name, e.g. GigaDevice
                                         # For bsp, the English name of the board␣
↪provider
                                         # For others, OPTIONAL, the provider of␣
↪this package
  name: demosoc                          # <MUST> The name here serves a different␣
↪purpose than the package name; it is used to display the specific package name
                                         # For ssp, the specific English name of␣
↪the SoC in this package; it can be a series name, e.g. GD32VF103
                                         # For bsp, the name of the board
                                         # For others, the display name of this␣
↪package, Chinese or English; if not set, the root-level name field is used by␣
↪default
  doc:                                   # <OPTIONAL>, if not set, this is empty␣
↪and not displayed
    website:                             # <OPTIONAL>, the website corresponding to␣
↪the package
    sch:                                 # <OPTIONAL> only for bsp, the circuit␣
↪schematic of the board
    datasheet:                           # <OPTIONAL>, package datasheet, local␣
↪path or URL
    usermanual:                          # <OPTIONAL>, Package User Manual
    extra:                               # <OPTIONAL>, additional materials, in␣
↪list form (URI + DESCRIPTION)
      - uri:                             # <OPTIONAL>, file path or web link
```

(continues on next page)

(continued from previous page)

```
      description:                      # <MUST> when uri is defined, description␣
↪for file


## Package Dependency
dependencies:                           # <OPTIONAL> List the dependent packages␣
↪in the form owner/name:version
  - name: csp-nsdk_nmsis                # <MUST> when defined        # 1. For sdk␣
↪type packages, there is an implicit dependency: they depend on exactly one bsp␣
↪type package,
    owner: nuclei # <OPTIONAL> if not defined, it will use the owner definition␣
↪above. Used to depend on a package of a specific owner, in the form owner/
↪name:version; the package is ultimately looked up this way
     version: # <OPTIONAL> when defined empty, use default as version number #   ␣
↪and all npk.yml files under the current path are automatically found and␣
↪included as part of the sdk package
                                        # 2. For packages other than sdk type, if␣
↪they are not placed under the directory of an sdk package but depend on
                                        #    an sdk, the dependency must be␣
↪declared explicitly
                                        # 3. A bsp type package always depends on␣
↪one ssp type package
                                        # 4. Dependencies can also carry a version␣
↪number, and version condition matching is supported. If no version is specified,␣
↪packages without a version are preferred, then the latest package
                                        # See 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
                                        #    e.g. version: master, version: 1.2.3,␣
↪version: >0.1.0
                                        # Dependency handling rules
                                        # 1. sdk and app type packages can depend␣
↪on multiple ssp and bsp type packages, but ultimately only the packages selected␣
↪in the project wizard are used
                                        # 2. app/bsp/ssp/csp/osp/mwp type packages␣
↪can depend on sdk; if a package depends on an sdk type package, it means the␣
↪package belongs to that sdk
                                        # 3. A bsp type package can only depend on␣
↪one ssp/csp type package; an ssp type package can only depend on one csp type␣
↪package
                                        # 4. An sdk type package can depend on␣
↪multiple app/bsp/ssp/csp/osp/mwp types, but this dependency only establishes a␣
↪membership relationship, indicating that the sdk contains these packages
                                        # 5. An sdk type package cannot depend on␣
↪another sdk package, but app/bsp/ssp/csp/osp/mwp packages can depend on multiple␣
↪sdk type packages

## Package Configurations
configuration:                          # <OPTIONAL> Options for package␣
↪configuration, used for Project Wizard creation and internal parameter settings
```

(continues on next page)

(continued from previous page)

```
                                    # The sdk type does not currently support␣
↪configuration parameter definitions
                                    # Configurations defined in configuration␣
↪can override each other
                                    # The override rule is app > mwp  > osp >␣
↪bsp > ssp > csp
  nuclei_core:                      # <OPTIONAL> A configuration option of␣
↪type choice
    default: n307fd                 # <MUST> If this configuration is defined,␣
↪for the choice type, the default value must be one of the choices listed below
    type: choice                    # <MUST> Configuration type; available␣
↪options are choice, list, checkbox, multicheckbox, text
    global: true                    # <OPTIONAL> Can be true or false,␣
↪defaults to true
    description: Nuclei RISC-V Core # <MUST> Description of this configuration␣
↪item, within 20 characters
    choices:                        # <MUST> When the configuration item type␣
↪== choice
      - name: n201                  # <MUST> Each item must contain name and␣
↪description
        arch: rv32iac               # <OPTIONAL> Only used to indicate the␣
↪ARCH information of the RISC-V CORE; not recommended for arbitrary use
        abi: ilp32                  # <OPTIONAL> Only used to indicate the ABI␣
↪information of the RISC-V CORE; not recommended for arbitrary use
        tune:                       # <OPTIONAL> Only used to indicate the␣
↪TUNE information of the RISC-V CORE; not recommended for arbitrary use
        info:                       # <OPTIONAL> Used to access custom key-
↪value data, e.g. ${nuclei_core.info.key1} returns value1
            - name: key1            # <MUST> key in pair with value, string␣
↪type, must not contain any spaces
              value: value1         # <MUST> value in pair with key, string␣
↪type
            - name: key2            # <MUST>
              value: value2         # <MUST>
        description: N201 Core(ARCH=rv32iac, ABI=ilp32)    # <MUST> Describes the␣
↪specific meaning of this item
                                    # Besides name and description, other␣
↪items may be defined with arbitrary names
                                    # For example, arch and abi are defined␣
↪here
      - name: n201e                 # Another item
        arch: rv32eac
        abi: ilp32e
        description: N201E Core(ARCH=rv32eac, ABI=ilp32e)
  extra_flags:                          # <OPTIONAL> A configuration option,
↪ this one is of type text
    value:                              # <MUST>, only accepts English␣
↪strings
    description: Extra compiler flags   # <MUST> Description of this␣
↪configuration item, within 30 characters
  dsp_present:                          # <OPTIONAL> A configuration option,
↪ this one is of type checkbox
    default: 0                          # <MUST>, defaults to 0, can be 0␣
↪or 1
    type: checkbox                      # <MUST>, configuration type,␣
↪checkbox here
    global: true                        # <OPTIONAL> Can be true or false,␣
↪defaults to true
    description: P-Extension(DSP) Present  # <MUST> Description of this␣
↪configuration item, within 30 characters
  libraries:                            # <OPTIONAL> A configuration option,
```

(continues on next page)

(continued from previous page)

```
↪ this one is of type multicheckbox
    default: [dsp, nn]                          # <MUST> Default value, a␣
↪combination from choices
    type: multicheckbox                         # <MUST>, configuration type,␣
↪multicheckbox here
    global: true                                # <OPTIONAL> Can be true or false,␣
↪defaults to true
    description: Libraries Used                 # <MUST> Description of this␣
↪configuration item, within 30 characters
    choices:                                    # <MUST> When the configuration␣
↪item type == multicheckbox
      - name: dsp                               # <MUST> Each item must contain␣
↪name and description
        description: DSP Library                # <MUST> Describes the specific␣
↪meaning of this item
                                                # Besides name and description,␣
↪other items may be defined with arbitrary names
      - name: nn
        description: NN Library
      - name: ai
        description: AI Library

## Source Code Management
codemanage:                                     # <MUST> This is mandatory
  installdir: demosoc                           # <MUST> The directory name where the␣
↪code should be installed, English only, must follow C naming format
                                                # For sdk type packages, installed to
↪<sdk_installdir>; if installdir is not defined, it defaults to SDK; if no sdk␣
↪type package is referenced, sdk_installdir is also set to SDK by default
                                                # For csp type packages, installed to
↪<sdk_installdir>/<csp_installdir>, TBD
                                                # For ssp type packages, installed␣
↪under <sdk_installdir>/SoC/<ssp_installdir>/Common
                                                # For bsp type packages, installed␣
↪under <sdk_installdir>/SoC/<ssp_installdir>/Board/<bsp_installdir>; if it does␣
↪not depend on any ssp type, installed to <sdk_installdir>/BSP/<bsp_installdir>
                                                # For osp type packages, installed␣
↪under <sdk_installdir>/OS/<osp_installdir>
                                                # For app type packages, installed␣
↪under <app_installdir>/; if installdir is not defined, it defaults to application
                                                # For mwp type packages, installed to
↪<sdk_installdir>/Components/<mwp_installdir>
  copyfiles:                                    # <MUST> Files or folders to be copied,
↪ glob pattern matching supported; refers to all directories or files
    - path: ["Source/", "Include/", "demosoc.svd"]        # <MUST> List of paths␣
↪of files or folders to be copied, glob pattern matching supported
    - path: ["DSP_Source", "DSP_Include"]
      condition: $( ${dsp_present} == 1 )                 # <OPTIONAL> The "if"␣
↪here is a fixed identifier; if present, a condition check is performed as follows
                                                          # For example, dsp_
↪present is defined in configuration, and its value is determined by the wizard␣
↪or other selected packages
  incdirs:                                                # <OPTIONAL> List of␣
↪header file directories to be added
    - path: ["Include/"]                                  # <OPTIONAL> Directories␣
↪of header files to be added
  libdirs:                                                # <OPTIONAL> Optional␣
↪directories where lib libraries are located
      - path:
  ldlibs:                                                 # <OPTIONAL> Optional␣
↪libraries to be linked
```

(continues on next page)

(continued from previous page)

```
      - libs:

## Set Configuration for other packages
setconfig:                                                  # <OPTIONAL> Used to set␣
↪options of other packages
                                                            # The following options␣
↪follow the override rule: app > mwp  > osp > bsp > ssp > csp
  - config: nmsislibarch
    value : ${nuclei_core.arch}p                            # <OPTIONAL> Directly␣
↪set options in Configuration
    condition: $( ${dsp_present} == 1 )                     # <OPTIONAL> Determine␣
↪whether to set the nmsislibarch value based on dsp_present here
  - config: nmsislibarch
    value: ${nuclei_core.arch}
    condition: $( ${dsp_present} == 0 )

## Build Configuration
buildconfig:                                    # <OPTIONAL> Configuration of build␣
↪options
                                                # Currently, build options from all␣
↪packages are concatenated or overridden
                                                # The following options are␣
↪overridden: app > mwp  > osp > bsp > ssp > csp
                                                # type is a special field used to␣
↪identify a specific compiler; gcc is currently supported
                                                # cross_prefix, prebuild_steps,␣
↪postbuild_steps, description
                                                # The remaining options are␣
↪concatenated
  - type: gcc                                   # <OPTIONAL> Currently only gcc;␣
↪other interfaces are reserved
    description: Nuclei GNU Toolchain           # <MUST> For ssp
    cross_prefix: riscv-nuclei-elf-             # <OPTIONAL> If not set or left empty,
↪ the toolchain provided by the system is used automatically
    toolchains:                                 # <OPTIONAL> Detailed toolchain␣
↪command settings; each key overrides the corresponding command of the resolved␣
↪toolchain, and an empty value keeps the predefined default; if this field is␣
↪absent, the commands of the toolchain provided by the system are used
      prefix: riscv-nuclei-elf-                 # <OPTIONAL> Prefix of the toolchain␣
↪commands
      suffix:                                   # <OPTIONAL> Suffix of the toolchain␣
↪commands; usually left empty
      ccompiler: gcc                            # <OPTIONAL> C compiler command
      cppcompiler: g++                          # <OPTIONAL> C++ compiler command
      archiver: ar                              # <OPTIONAL> Archiver command
      hexbinconv: objcopy                       # <OPTIONAL> Hex to binary conversion␣
↪command
      listgen: objdump                          # <OPTIONAL> Listing generation␣
↪command
      sizecmd: size                             # <OPTIONAL> Size printing command
      buildcmd: make                            # <OPTIONAL> Build command
      rmcmd: rm                                 # <OPTIONAL> Remove command
      toolchain_path: ${eclipse_home}\toolchain\gcc\bin # <OPTIONAL> Path of the␣
↪toolchain
      buildtool_path: ${eclipse_home}\toolchain\build-tools\bin # <OPTIONAL> Path␣
↪of the build tools
    common_flags:                               # <OPTIONAL> Common build options,␣
↪which will be added to cflags, asmflags, and cxxflags
      - flags: -g -fno-common -ffunction-sections -fdata-sections
      - flags: -march=${nuclei_core.arch} -mabi=${nuclei_core.abi} -mcmodel=medany
    ldflags:                                    # <OPTIONAL> List of linker options;␣
```

(continues on next page)

(continued from previous page)

```
↪empty means no options
      - 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> Definition of the link␣
↪script, must be defined in bsp/ssp
      - script: "Source/GCC/gcc_demosoc_${.download}.ld"
        condition: $(check pattern)         # <OPTIONAL> Perform a conditional␣
↪check
    cflags:                                 # <OPTIONAL> C build options; empty␣
↪means no options
      - flags: -O3
    asmflags:                               # <OPTIONAL> ASM build options; empty␣
↪means no options
      - flags: -O2
    cxxflags:                               # <OPTIONAL> CXX build options; empty␣
↪means no options
      - flags: -O1
    common_defines:                         # <OPTIONAL> Common macro definitions
      - defines: __RISCV_FEATURE_DSP=1
        condition: $(${dsp_present} == 1)
      - defines: DOWNLOAD_MODE_STRING=\"flashxip\"
    cdefines:                               # <OPTIONAL> C macro definitions
      - defines: __RISCV_FEATURE_DSP=1
        condition: $(${dsp_present} == 1)
      - defines: DOWNLOAD_MODE_STRING=\"flashxip\"
    asmdefines:                             # <OPTIONAL> ASM macro definitions
      - defines: __RISCV_FEATURE_DSP=1
        condition: $(${dsp_present} == 1)
      - defines: DOWNLOAD_MODE_STRING=\"flashxip\"
    cxxdefines:                             # <OPTIONAL> CXX macro definitions
      - defines: __RISCV_FEATURE_DSP=1
        condition: $(${dsp_present} == 1)
      - defines: DOWNLOAD_MODE_STRING=\"flashxip\"
    prebuild_steps:                         # <OPTIONAL> Commands executed before␣
↪build
      command:                              # <OPTIONAL> The command line to␣
↪execute
      description:                          # <OPTIONAL> Description of the␣
↪executed command
    postbuild_steps:                        # <OPTIONAL> Commands executed after␣
↪the build completes
      command:                              # <OPTIONAL> The command line to␣
↪execute
      description:                          # <OPTIONAL> Description of the␣
↪executed command

## Debug Configuration
debugconfig:                            # <MUST> For bsp type, optional for app/
↪ssp type
# Currently, debug options from all packages are concatenated or overridden
# type is a special field used to describe a specific debugger; openocd and qemu␣
↪are currently supported
# The following fields are overridden: app > mwp  > osp > bsp > ssp > csp
# description, svd
# The key-value pairs under the configs field are merged: if the corresponding key␣
↪exists, it is overridden (following the same override rule); otherwise it is␣
↪merged
```

(continues on next page)

(continued from previous page)

```
  - type: openocd   # <MUST> The selected tool
    description: Nuclei OpenOCD # <MUST> For bsp type
    svd: gd32vf103.svd  # <OPTIONAL> Optional SVD file
    configs:
      - key: config  # openocd configuration file
        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 Sections
## Only works on tool type
## Each package has a package path, referenced as npk name-version, e.g. ${tool-
↪cmlink-1.0.0};
## other variables are referenced as npk name-version-variable name, e.g. ${tool-
↪cmlink-1.0.0-proxy}
environment:            # Extended variables
  - key: proxy            # Variable name
    value: bin/cmlink_gdbserver.exe        # The actual resolved result is the npk␣
↪file's parent path + value, e.g. 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    # Defaults to false; when system is true, the variable is␣
↪referenced directly by its name, e.g. ${proxy}

## Template File Management
## Only works on tpp type. This type is special; its description file is npk_
↪template.yml, which is an extension of npk.yml
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/${soc}_common.c
    Common/Source/system_demosoc.c.ftl: Common/Source/system_${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
```

(continues on next page)

(continued from previous page)

```
    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
```

Content Conventions

To keep the `npk.yml` file readable and simple, the following conventions apply to how it is written:

- Field order should be consistent with the template; data and DICT entries are saved in the order they are read
- Add comments to fields, especially where explanation is needed
- Fields marked **MUST** are checked against the rules described in the comments above; if a field does not comply, an error is reported and the package is not imported
- Use 2 space characters for indentation
- **OPTIONAL** fields can be left empty or omitted entirely
- Add one blank line between first-level fields; do not use blank lines for second-level and lower fields; do not use blank lines between first-level fields in the first part (base information)
- Do not use quotation marks for strings, except where special syntax requires them
- Keep all `description` fields within 20 characters, English only, for layout and display purposes
- For the conventions on multi-line strings in YAML, see: https://yaml-multiline.info/

Package Import Rules

The package import rules are defined below:

- **If the package being imported depends on other packages (with version matching) that have not been imported, the import is rejected, and the user is prompted that dependent packages are missing and must be imported first.**
  - Once package management goes online in the future, the user can be prompted to download the dependent packages from the network or to import ZIP packages manually
- **If a package being deleted is depended on by other packages, the user is shown which packages depend on it and asked whether to delete it; after deletion, the missing packages are shown in package management**
  - Once package management goes online in the future, downloading missing packages with one click will be supported, or ZIP packages can be imported manually
- If a package of the same version is imported, the user is informed that the package already exists and asked whether to replace it
- If a package of a different version is imported, the user is informed that another version of the package already exists and asked whether to continue with the import
- Imported packages are displayed grouped by their defined type, showing the package version, name, owner, description, homepage, and license

ZIP Package Content Rules

The content rules for a ZIP package are defined below:

- A ZIP package must contain at least one NPK file
- **Package type determination: if a package contains NPK files of multiple types, the package type is determined as follows**
  - sdk > ssp > bsp > osp > mwp > csp > app
  - If multiple NPKs of the determined package type exist, the package is invalid, the import is rejected, and the user is prompted
- A package of a given type must not contain multiple NPK files of that type
- For an `sdk` type package, it must contain at least one `ssp` package, one `bsp` package that depends on that `ssp`, and at least one `app` type package; packages of other types are allowed
- For packages of other types, any other NPKs they contain must explicitly depend on that package

Package Dependency Handling

Package dependency handling defines how packages can be split and organized into sensible dependency relationships so that packages can be maintained independently. The dependency rules for each package type are described in detail below.

Dependencies are declared in the dependency list under the `dependencies` field, which supports depending on a package with a specific name and version from a specific owner. The lookup rule is `owner/name:version`; if owner is not defined, it defaults to the owner defined in the npk file; if `version` is not defined, the package is first looked up within the same package bundle, otherwise the latest package is used.

csp Core Support Package Dependencies

A csp type package is a software package that supports processor cores (CORE). For Nuclei RISC-V cores, the recommended csp is the open-source NMSIS software support package.

A csp type package is a very low-level package, and it must not depend on packages of types such as `ssp/bsp/mwp/rtos/app`. However, it can depend on sdk type packages, which means the package belongs to the environment of the sdk package it depends on.

ssp SoC Support Package Dependencies

An ssp type package is a software package that supports an SoC or chip, such as the SoC support packages for `gd32vf103` and `demosoc`.

An ssp software package can only depend on packages of types `csp/mwp/osp`. Such a dependency means that the code of these packages is imported when the project is created or when code is introduced. **If it depends on an sdk type package, it means the ssp type package belongs to the environment of the sdk type package it depends on.**

In theory, a user can create an ssp software package that does not depend on any `csp/mwp/rtos` package and does not belong to any sdk type package. It can depend on at most one **osp** type package.

bsp Board Support Package Dependencies

A bsp type package is a software support package released for a development board based on a specific SoC or chip, such as the bsp software package for the `gd32vf103-rvstar` development board.

A bsp software package can only depend on packages of types `ssp/csp/mwp/osp`. Such a dependency means that the code of these packages is imported when the project is created or when code is introduced. **If it depends on an sdk type package, it means the bsp type package belongs to the environment of the sdk type package it depends on.**

In theory, a user can create a bsp software package that does not depend on any package. It can depend on at most one **osp** type package.

osp OS Support Package Dependencies

An osp type package is a software support package for a specific RTOS, such as freertos or ucosii.

An osp type package can only depend on packages of types `ssp/csp/mwp`. Such a dependency means that the code of these packages is imported when the project is created or when code is introduced. **If it depends on an sdk type package, it means the osp type package belongs to the environment of the sdk type package it depends on.**

mwp Middleware Support Package Dependencies

An mwp type package is a middleware software package, such as a speech algorithm library or an IoT connectivity library like `mqtt` or `coap`.

An mwp type package can only depend on packages of types `bsp/ssp/csp/mwp/osp`, but depending directly on `bsp/ssp` is not recommended. When creating middleware, keep it generic so that it can be easily integrated into other software.

sdk Software Development Kit Package Dependencies

An sdk type package is a special kind of software package. It contains no code of its own; instead, it is composed entirely of dependencies on other package types. When an sdk type package is imported, all `npk.yml` files under the package directory are scanned for other packages, which are included in the SDK dependency without having to declare the dependency explicitly. This dependency relationship does not directly cause code to be imported when a project is created; an sdk type package is essentially a collection of packages.

An sdk type package may depend on multiple ssp, multiple bsp, multiple csp, multiple app, multiple mwp, and multiple osp packages. For more details, see *Building an SDK Development Package*.

When resolving the dependencies of packages that belong to an sdk type package, packages inside the same sdk package are preferred. For example:

- sdk-nuclei-sdk is an sdk type package that internally contains the packages csp-nsdk_nmsis, bsp-nsdk_nuclei_fpga_eval, ssp-nsdk_demosoc, ssp-nsdk_gd32vf103, osp-nsdk_freertos, and osp-nsdk_ucosii
- External versions of packages such as csp-nsdk_nmsis and osp-nsdk_freertos also exist. During project creation, the internal csp-nsdk_nmsis and osp-nsdk_freertos are preferred as long as version matching is satisfied; the external ones are used only when the version matching requirements are not met
- After the project is created, users can manually upgrade a specific package to another version.

2.4.2 Section Overview

As the example above shows, a standard `npk.yml` file consists of several major blocks, though not all of them are used in practice. A compliant `npk.yml` file works in Nuclei Studio as long as it contains the basic information.

Package Base Information

This block contains the basic information of the NPK; most of the key information must be described clearly here. The key fields are described below.

- **name**

Mandatory. The name ID of the NPK. No spaces; must follow C naming conventions; English name; a unique name ID.

- **version**

Optional. If not set, defaults to empty. SEMVER 2.0 versioning is recommended; must start with a digit, e.g. 1.2.3

- **type**

Mandatory. Available type values are csp, ssp, bsp, osp, app, mwp, sdk, tpp, tool

- **os**

Optional. Indicates which variant of Nuclei Studio the NPK applies to. The Nuclei Studio releases currently distributed come in win64 and lin64 versions. The OS type can be win32, win64, lin64, or lin32, but the package upload page currently only supports win64 and lin64. This field only exists in tool type packages

- **owner**

Mandatory. The owner of the package. This ID is generally an authenticated developer ID, used for permission matching later. If the NPK is only for local testing, any value is acceptable.

Package Information

The **packinfo** block provides descriptive information about the NPK, including documentation and other materials. When the NPK is used in Nuclei Studio, this information is displayed in the New Project wizard.

Package Dependency

dependencies describes the dependency relationships of the NPK. To enable NPK reuse and reduce maintenance costs, an NPK can declare dependencies: an NPK can depend on zero or more NPKs. The dependencies field therefore appears as a group of objects, and each dependency object must specify the **name**, **owner**, and **version** of the NPK.

Package Configurations

The **Configuration** field is a special field, mainly used to provide configurable items that support interactive scenarios during project creation.

The second-level field names under the configuration field can be identical across different packages; identical names carry identical meanings. If the same name is defined more than once, the following override rule applies.

The override rule is: app > mwp > osp > bsp > ssp > csp

The **Configuration** object group contains multiple objects, each with a fixed structure.

- **XXX (variable name)**

The variable name is arbitrary as long as it follows C++ naming conventions. It is referenced elsewhere in the form ${XXX} or ${XXX.XX}.

- **default**

Default value, optional.

- **type**

The type of this variable. To support a richer UI experience, NPK defines many UI component types; see the following chapters for details.

- **global**

Indicates whether this field is displayed on the wizard page during project creation.

- **tips**

Descriptive information about this variable, mainly used for UI tips events.

- **hints**

Descriptive information about this variable, such as value examples, mainly used for UI hints events.

- **description**

The description of this variable in the NPK.

- **UI component information**

Supported types include choice, list, checkbox, multicheckbox, text, etc.; see below for details

Source Code Management

codemanage describes content related to the template project. An NPK often includes many complex features, but only some of its files are needed when a specific project is created, and those files must be configured. **codemanage** describes this information; it contains the following keywords:

- **custom**

Defaults to false. When true, this installdir indicates the directory to install directly into

- **srcroot**

Defaults to `.`, which means the directory where the current `npk.yml` is located; it can be a relative path, e.g. `../`, `../bsp`, etc. Note that once this is set, the root directories of the corresponding copyfiles/incdirs/libdirs paths are all affected and use the newly set path

- **installdir**

The directory name where the code should be installed, English only, must follow C naming format

- For sdk type packages, installed to `<sdk_installdir>`; if installdir is not defined, it defaults to SDK; if no sdk type package is referenced, sdk_installdir is also set to SDK by default
- For csp type packages, installed to `<sdk_installdir>/<csp_installdir>`, TBD
- For ssp type packages, installed under `<sdk_installdir>/SoC/<ssp_installdir>/Common`
- For bsp type packages, installed under `<sdk_installdir>/SoC/<ssp_installdir>/Board/<bsp_installdir>`; if it does not depend on any ssp type, installed to `<sdk_installdir>/BSP/<bsp_installdir>`
- For osp type packages, installed under `<sdk_installdir>/OS/<osp_installdir>`
- For app type packages, installed under `<app_installdir>/`; if installdir is not defined, it defaults to application
- For mwp type packages, installed to `<sdk_installdir>/Components/<mwp_installdir>`

Note: 2023.05.26 Added: `copyfiles/incdirs/libdirs` all support relative parent directories such as `../../`, but when installing or setting paths, they are all placed under `<installdir>`

For example: `path: ["../common/"]` copies the `common` directory from the parent directory and places it under `<installdir>/common`.

If there is already a `common` directory below, create `R1L_common`; for `../../common`, create `R2L_common`.

This scheme is no longer used; same-name directories are created directly, and same-name files are overwritten directly. Use `srcroot: ..` instead for the corresponding `incdirs/libdirs`

When such relative paths are encountered, the final installed path and file name prevail

- **copyfiles**

Files or folders to be copied, covering all directories or files; supports `../`, `*`, `*.*`, and is used together with srcroot.

- **incdirs**

Mandatory. The list of header file directories to be added. Nuclei Studio resolves these paths; the final paths are relative to the project root directory.

- **libdirs**

Optional. The directories where lib libraries are located. Nuclei Studio resolves these paths; the final paths are relative to the project root directory.

- **ldlibs**

Optional libraries to be linked. Nuclei Studio resolves these paths; the final paths are relative to the project root directory.

Set Configuration for Other Packages

setconfig is used to set options of other packages in the NPK, following the override rule app > mwp > osp > bsp > ssp > csp.

setconfig is an object group that can be extended indefinitely; each object has three fields.

- **config**

Variable name, following C++ naming conventions. A general variable XXX is referenced elsewhere in the form ${XXX}

The variable name is not unique; it can take effect conditionally through condition checks, and it also follows the override rule app > mwp > osp > bsp > ssp > csp for automatic overriding.

- **value**

The value of the variable

- **condition**

The condition of the variable. The value takes effect only when the condition is met

Build Configuration

Configures the build tools and build options of the project. Its keywords include the following fixed fields:

- **type**

The type of build tool supported; values are generally gcc, clang, common. Currently only gcc and clang are supported. Because some build option configurations are identical, the common type was added to improve code reusability.

- **description**

A description of this build tool.

- **toolchain_name**

Important field. The name of the build tool.

- **cross_prefix**

Important field. The prefix of the build tool.

- **toolchains**

Detailed settings of the toolchain commands, given as key-value pairs. Each key overrides the corresponding command of the toolchain resolved by `toolchain_name` / `cross_prefix`; if a value is empty, the predefined default command is kept; if this field is absent, the commands of the toolchain provided by the system are used.

Key	Description
`prefix`	Prefix of the toolchain commands, for example `riscv-nuclei-elf-`
`suffix`	Suffix of the toolchain commands
`ccompiler`	C compiler command
`cppcompiler`	C++ compiler command
`archiver`	Archiver command
`hexbinconv`	Hex to binary conversion command
`listgen`	Listing generation command
`sizecmd`	Size printing command
`buildcmd`	Build command
`rmcmd`	Remove command
`toolchain_path`	Path of the toolchain
`buildtool_path`	Path of the build tools

- **unflags**

Takes effect in `common_flags/cflags/asmflags/ldflags/cxxflags` of the buildconfig section; removes previously defined flags.

- **undefines**

Takes effect in `common_defines/cdefines/asmdefines/cxxdefines` of the buildconfig section; removes previously defined defines (takes effect on exact string match).

- **common_flags**

Common build options, which will be added to cflags, asmflags, and cxxflags; empty means no options.

- **ldflags**

List of linker options; empty means no options.

- **linkscript**

Definition of the link script, must be defined in bsp/ssp; empty means no options.

- **cflags**

C build options; empty means no options.

- **asmflags**

  ASM build options; empty means no options.

- **cxxflags**

  CXX build options; empty means no options.

- **common_defines**

  Common macro definitions; empty means no options.

- **cdefines**

  C macro definitions; empty means no options.

- **asmdefines**

  ASM macro definitions; empty means no options.

- **cxxdefines**

  CXX macro definitions; empty means no options.

- **prebuild_steps**

  - **command**

  Commands executed before build; empty means no options.

  - **description**

  Description of the commands executed before build; empty means no options.

- **postbuild_steps**

  - **command**

  Commands executed after build; empty means no options.

  - **description**

  Description of the commands executed after build; empty means no options.

Debug Configuration

Configures the debug type and related parameters of the project. Its keywords include the following fixed fields. This block is optional; if configured, Nuclei Studio generates a launch file from its content when the project is created, so the project can be debugged directly.

- **type**

  Debug type. Currently supports GDB Custom, GDB SEGGER J-Link, GDB OpenOCD, GDB Nuclei QEMU, Nuclei RVProf

- **description**

  Description of the supported debugger (Custom, J-Link, OpenOCD, QEMU, or RVProf)

- **configs**

  The parameters of the corresponding debug type. All parameters appear as key-value pairs. Each debug type requires different parameters; detailed explanations are given below.

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

GDB Custom Debug Arguments

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	Supported types: Commands, General, and DL
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 IP 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 Image 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 binary
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 Debug Arguments

Table 2.2: Arguments of GDB SEGGER J-Link Debug

Name	Reset Value	Description
arch		Nuclei ARCH
executable		OpenOCD Executable Name
tool_extrapreopts		Additional value, prepended to Config options
tool_extraopts		Additional value, replacing Config options
tool_extrapostopts		Additional value, appended to Config options
gdb_extraprecmds		Additional value, prepended to Commands
gdb_extracmds		Additional value, replacing Commands
gdb_extrapostcmds		Additional value, appended to Commands
doStartGdbServer	true	Start the J-Link GDB server locally
doConnectToRunning	false	Connect to running target

continues on next page

Table 2.2 – continued from previous page

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
gdbClientOtherCommands	set mem inaccessible-by-default off	gdb Client Other Commands
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 Server Other
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

continues on next page

Table 2.2 – continued from previous page

svdPath	svd file path

GDB OpenOCD Debug Arguments

Table 2.3: Arguments of GDB OpenOCD Debug

Name	Reset Value	Description
arch		Nuclei ARCH
executable		OpenOCD Executable Name
tool_extrapreopts		Additional value, prepended to Config options
tool_extraopts		Additional value, replacing Config options
tool_extrapostopts		Additional value, appended to Config options
gdb_extraprecmds		Additional value, prepended to Commands
gdb_extracmds		Additional value, replacing Commands
gdb_extrapostcmds		Additional value, appended to Commands
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 IP 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

continues on next page

Table 2.3 – continued from previous page

doContinue	true	Continue
svdPath		svd file path

GDB Nuclei QEMU Arguments

Table 2.4: Arguments of GDB Nuclei QEMU Debug

Name	Reset Value	Description
nuclei_archext		Nuclei archext
extra_board		Nuclei DevelopBoard
extra_cpu		Nuclei CPU Core
riscv_arch		Nuclei ARCH
executable		OpenOCD Executable Name
tool_extrapreopts		Additional value, prepended to Config options
tool_extraopts		Additional value, replacing Config options
tool_extrapostopts		Additional value, appended to Config options
gdb_extraprecmds		Additional value, prepended to Commands
gdb_extracmds		Additional value, replacing Commands
gdb_extrapostcmds		Additional value, appended to Commands
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 IP 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

continues on next page

Table 2.4 – continued from previous page

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

Nuclei Model Arguments

Table 2.5: Arguments of Nuclei Model Debug

Name	Reset Value	Description
executable		Nuclei Model Executable Name
tool_extrapreopts		Tool Extra Pre Options
tool_extraopts		Tool Extra Options
tool_extrapostopts		Tool Extra Post Options
gdb_extraprecmds		gdb Extra Pre Commands
gdb_extracmds		gdb Extra Commands
gdb_extrapostcmds		gdb Extra Post Commands
doStartGdbServer	true	Start the Nuclei Model GDB server locally
gdbServerExecutable	${nucleimodel_path}/${nucleimodel_executable}	Nuclei Model Executable
gdbServerGdbPortNumber	1234	GDB Port Number
gdbServerOther		Model Other Options
gdbServerSMPCount	1	SMP Count
gdbCoreName		Nuclei CPU Core
gdbDownloadName		Download Mode
otherExtensions		Nuclei archext
doGdbServerPreferXpacksBin	false	Prefer Xpacks Bin
doGdbServerAllocateConsole	true	Allocate Console
gdbServerDelaySeconds	0	Server Startup Delay (Seconds)
nucleiModelExecutableTimeOut	30	Model Executable TimeOut
donucleiModelAllowTrace	true	Allow Model Trace
nucleiModelTracePath		Model Trace Path
disableGraphics	true	Disable Graphics
isGdbServerVerbose	false	Verbose mode of the GDB server
useNucleiProperty	true	Use Nuclei Property
DEBUG_NAME	${cross_prefix}gdb${cross_suffix}	GDB Client Command
gdbClientOtherCommands		gdb Client Other Commands
gdbClientOtherOptions		gdb Client Other Options
ipAddress	localhost	Host name or IP address
portNumber	1234	Port number
jtagDevice	RISC-V MCU Nuclei Model	JTAG Device
doFirstReset	false	Initial Reset and Halt
otherInitCommands		Other Init Commands
loadSymbols	true	Load symbols
symbolsFileName		Symbols File Name
symbolsOffset		Symbols Offset
useFileForSymbols	false	Use File for Symbols
useProjBinaryForSymbols	true	Use Project Binary for Symbols
useRemoteTarget	true	Use Remote Target
loadImage	true	Load executable
useFileForImage	false	Use File for Image
imageFileName		Image File Name
imageOffset		Image Offset

continues on next page

Table 2.5 – continued from previous page

useProjBinaryForImage	true	Use Project Binary for Image
doDebugInRam	false	Debug in RAM
otherRunCommands		Other Run Commands
doSecondReset	true	Pre-run/Restart reset
setPcRegister	false	Set program counter
pcRegister		Set program counter at (hex)
setResume	false	Set Resume
setStopAt	true	Set breakpoint
stopAt	main	Set breakpoint at
doContinue	true	Continue
svdPath		SVD file path

Nuclei RVProf Arguments

Table 2.6: Arguments of Nuclei RVProf

Name	Reset Value	Description
cycleModelExecutable	${cyclemodel_path}/${cyclemodel_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 Variables

environment is a configuration used in tool type NPK packages. To share a tool (tool type, such as cycleModel) through NPK, use the environment configuration. Once **environment** is defined, Nuclei Studio automatically generates several global variables, which can be referenced in other NPKs in the form `${xxx-1.0.0-XXX}`.

Note:

- Each package has a package path, referenced as npk name-version, e.g. `${tool-cyclemodel-1.0.0}`
- Other variables are referenced as npk name-version-variable name, e.g. `${tool-cyclemodel-1.0.0-cyclemodel_path}`, `${tool-cyclemodel-1.0.0-cyclemodel_executable}`
- When the system value of a variable is true, an additional variable without a version number is generated (taking the highest version), e.g. `${tool-cyclemodel-cyclemodel_executable}`

```
name: tool-cyclemodel
owner: nuclei
os:
version: 1.0.0
```

(continues on next page)

(continued from previous page)

```
description: Nuclei Tools cyclemodel
details: Nuclei Tools cyclemodel
type: tool
keywords:
  - tool
  - cyclemodel
license: Apache-2.0
homepage:

## Extended variables  tool-cyclemodel-1.0.0 and tool-cyclemodel-1.0.0-proxy
environment:
  - key: cyclemodel_path
    value: bin
    description: cyclemodel path
    system: true
  - key: cyclemodel_executable
    value: bin/n300_best_config_cymodel_latest
    description: cyclemodel executable
    system: true


## This is code from another NPK, demonstrating how to use tool-cyclemodel
debugconfig:
  - type: rvprof
    description: Nuclei RVProf
    configs:
      - key: ncycm_path
        value: ${tool-cyclemodel-1.0.0-cyclemodel_executable}
      - key: rvprof_path
        value: ${tool-rvprof-1.0.0-rvprof_executable}
```

Template File Management

An internally used configuration; not described in detail here.

2.4.3 UI Components in NPK

NPK provides a rich set of UI components. The fields of these components all contain the sub-fields default, description, and global, each with its own meaning.

default indicates the default value, description indicates the meaning of the option, and global indicates whether the option is displayed during project creation (true) or is only used for internal parameter passing (false).

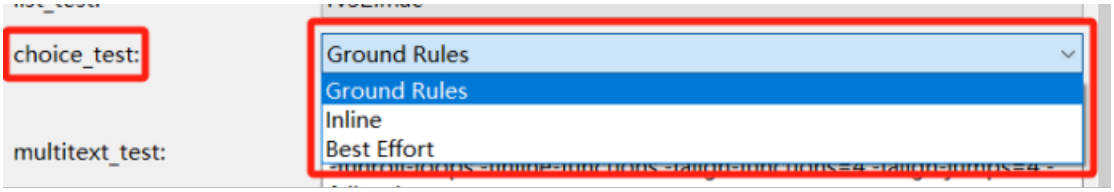
- **Choice single-selection box**

```
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
```

(continues on next page)

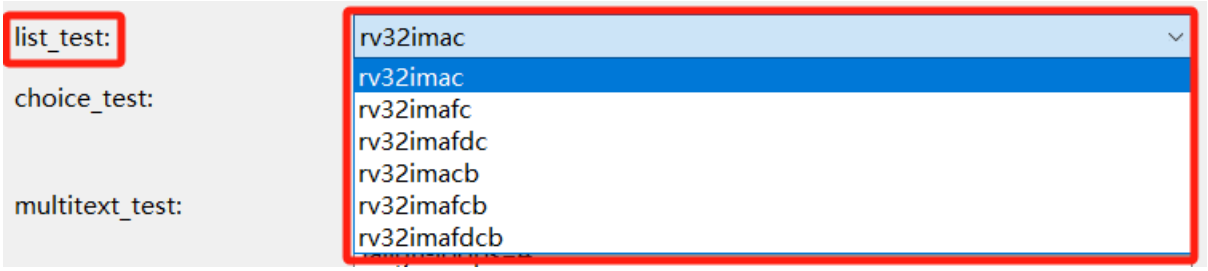
(continued from previous page)

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



- **list single-selection box**

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



- **checkbox single checkbox**

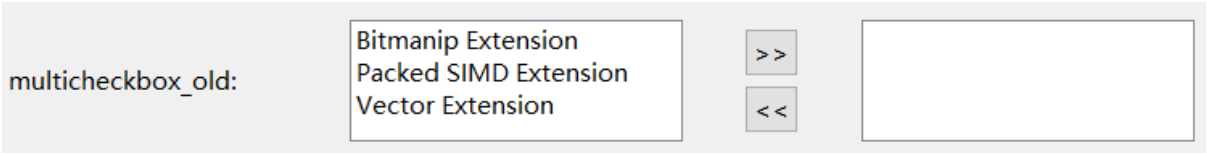
```
checkbox_test:
  default_value: 0
  type: checkbox
  global: true
  description: checkbox_test
```



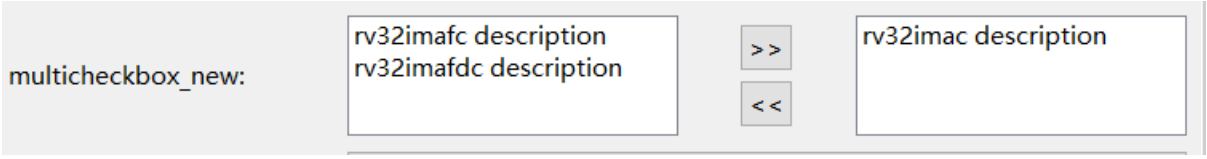
- **multicheckbox shuttle selection box**

  Two syntax forms are shown below

```
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
```

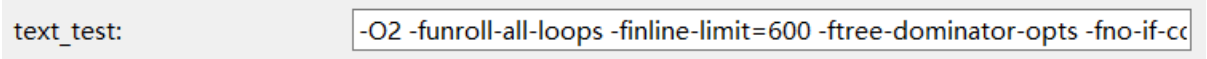



```
multicheckbox_new:
  default_value: rv32imac
  type: multicheckbox
  global: true
  description: multicheckbox_new
  param:
    name: ["rv32imac","rv32imafc","rv32imafdc"]
    description: ["${name} description","${name} description","${name} description
↪"]
```



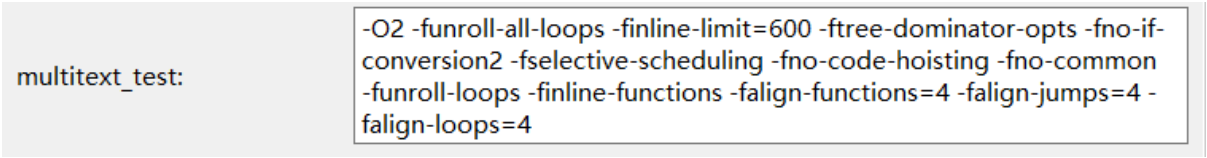
- **text single-line text box**

```
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 multi-line text box**

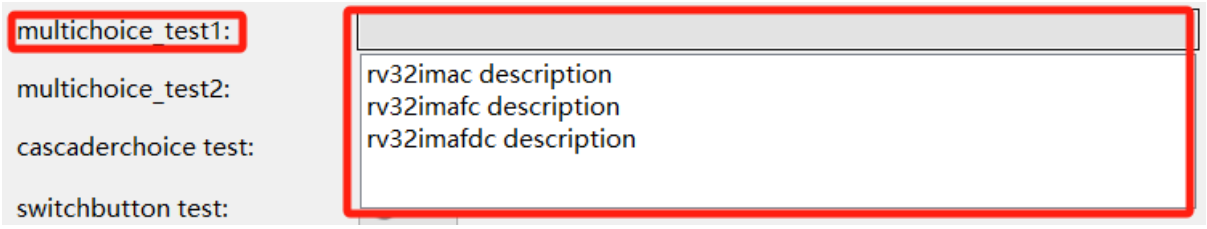
```
multitext_test:
  value: >-
    -O2 -funroll-all-loops -finline-limit=600 -ftree-dominator-opts
    -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting
    -fno-common -funroll-loops -finline-functions -falign-functions=4
    -falign-jumps=4 -falign-loops=4
  type: multitext
  description: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



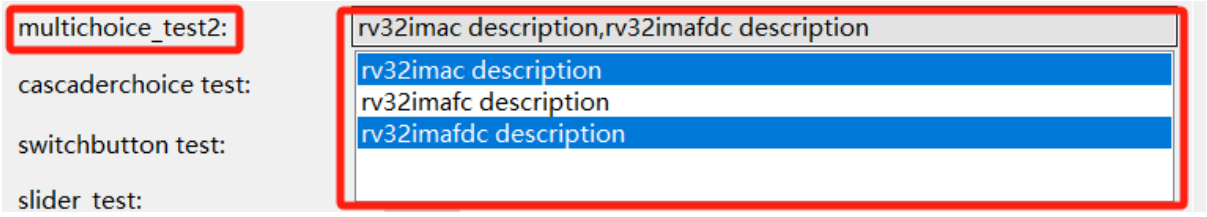
- **multichoice multi-select dropdown**

  Two syntax forms are shown below

```
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
```



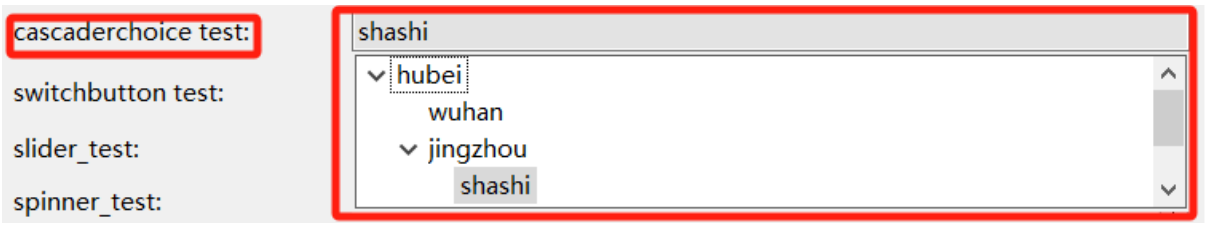
- **cascaderchoice cascading selection box**

```
cascaderchoice_test:
  default_value: >-
    [hubei,jingzhou,shashi]
  type: cascaderchoice
  global: true
  description: cascaderchoice test
  cascader_param:
    - hubei:
```

(continues on next page)

(continued from previous page)

```
        - wuhan
        - jingzhou:
            - shashi
            - jianli
    - hunan:
        - changsha
        - guangdong
```



- **switchbutton switch**

```
switchbutton_test:
  default_value: 0
  type: switchbutton
  global: true
  description: switchbutton test
```



- **slider numeric selection box**

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



- **spinner numeric selection box**

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

- **multispinner multiple numeric selection box**

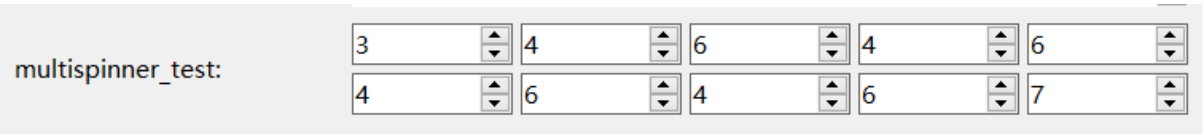
```
multispinner_test:
  default_value: >-
    [3,4,6,4,6,4,6,4,6,7]
  type: multispinner
  global: true
```

(continues on next page)



(continued from previous page)

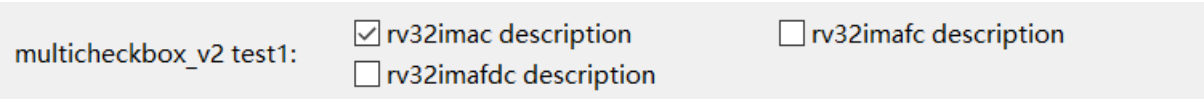
```
  description: multispinner_test
  param:
    range: >-
     [-100,100,1],[-100,100,2],[-100,100,3],[-100,100,3],[-100,100,3],[-100,100,3],
↪[-100,100,3],[-100,100,3],[-100,100,3],[-100,100,4]
```



- **multicheckbox_v2 multiple-checkbox box**

  Two syntax forms are shown below

```
multicheckbox_v2_test1:
default_value: >-
 [rv32imac]
type: multicheckbox_v2
global: true
description: multicheckbox_v2 test1
param:
  name: ["rv32imac","rv32imafc","rv32imafdc"]
  description: ["${name} description","${name} description","${name} description"]
```

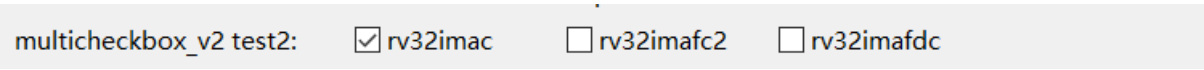


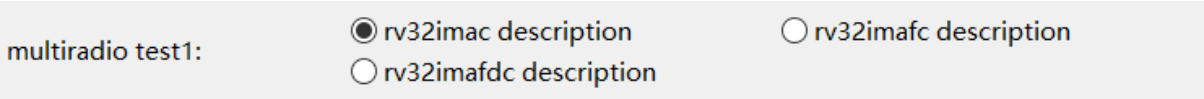

```
multicheckbox_v2_test2:
  default_value: >-
   [rv32imac]
  type: multicheckbox_v2
  global: true
  description: multicheckbox_v2 test2
  choices:
    - name: rv32imac
      description: rv32imac
    - name: rv32imafc
      description: rv32imafc2
    - name: rv32imafdc
      description: rv32imafdc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

- **multiradio radio box**

  Two syntax forms are shown below

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






```yaml
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 Syntax

YAML Language

The NPK description file npk.yml is written in YAML and supports standard YAML syntax. For more information about YAML, see: https://yaml.org/

Variable Definitions

The NPK description language allows users to define a custom variable and use it anywhere in NPKs that have a dependency relationship with the defining NPK.

Note:

- In the example, the NPK defines a variable app_commonflags. In any npk.yml file that has a dependency relationship with this NPK, the value of app_commonflags can be used through `${app_commonflags}`.
- In the example, the NPK defines a list object nuclei_core. In any npk.yml file that has a dependency relationship with this NPK, the arch value of the nuclei_core object can be used through `${nuclei_core.arch}`, the abi value through `${nuclei_core.abi}`, and the tune value through `${nuclei_core.tune}`.

```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
```

(continues on next page)

(continued from previous page)

```
        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:
  - 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}
```

Keywords

NPK defines a number of fields to describe various relationships. Most of the fields are self-explanatory; the following keywords are highlighted here.

- **condition**

condition is a custom keyword used very frequently in npk.yml to handle logical relationships, similar to **if**. It is used as follows.

```
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") )

# In the above description, the value of flags is determined by condition; in␣
↪different scenarios, the value of flags will differ.
# And because flags is an array type, flags in the above example will have␣
↪multiple values; what is ultimately used is the string concatenated from the␣
↪values of flags.
```

- **dependencies**

dependencies is used in npk.yml to describe the dependency relationships of an NPK.

In many cases, an NPK needs to depend on a package with a specific name and version from a specific owner. The lookup rule is `owner/name:version`; if owner is not defined, it defaults to the owner defined in the npk file; if **version** is not defined, the package is first looked up within the same package bundle, otherwise the latest package is used. If a depended-on package cannot be found, the NPK is unusable.

Note: In the example, the NPK depends on three NPKs, as follows:

- For sdk-nuclei_sdk, owner and `version` are not defined, so the package is first looked up within the same package bundle; otherwise the latest package is used
- tool-testmodel explicitly specifies owner and version
- tool-rvprof explicitly specifies owner and version

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

Custom Functions

To meet various needs, NPK (npk.yml) defines a set of commonly used built-in functions.

- **upper**

  Convert a string to uppercase

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

- **lower**

  Convert a string to lowercase

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

- **contains**

  Determine whether a string contains another string

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

- **join**

  Join a string array into a string

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

- **concat**

  Concatenate strings into a new string

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

- **strip**

  Remove whitespace at both ends of a string

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

- **startswith**

  Determine whether a string starts with the specified prefix

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

- **endswith**

  Determine whether a string ends with the specified suffix

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

- **arithop**

  Arithmetic operator. Supports common operators such as +, -, *, /, %, and ? (ternary operation); does not support ++ or

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

- **npack/npack_installdir**

  Checks whether npack contains the specified NPK

  npack_installdir is the path of the contained NPK

```
# 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) )

# This description means: when b.yml depends on a.yml, the value of config can be␣
↪set to /src/openocd.cfg under the directory of a.yml
```

- **list_get**

  Get the element at the specified index of an array

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

- **list_set**

  Modify the element at the specified index of an array

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

- **list_del**

  Delete the element at the specified index of an array

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

- **list_add**

  Insert a value at the specified index of an array

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

- **list_size**

  Get the length of an array

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

- **list_sub**

  Extract a sub-list of the specified length starting from the specified position in a 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**

  Replace a substring inside a string; the third parameter can be empty

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

2.5 Nuclei Studio NPK Creation and Sharing

Nuclei Studio 2022.04 provides an important feature: using the initial templates provided, developers can easily create their own NPK component packages and contribute them through the platform for other developers to use.

2.5.1 Develop an NPK Component Package

Authenticate as a Developer

Before creating an NPK component package, you must first authenticate a developer account to obtain a unique owner ID. This owner ID corresponds to the owner described earlier and makes NPKs easier to manage.

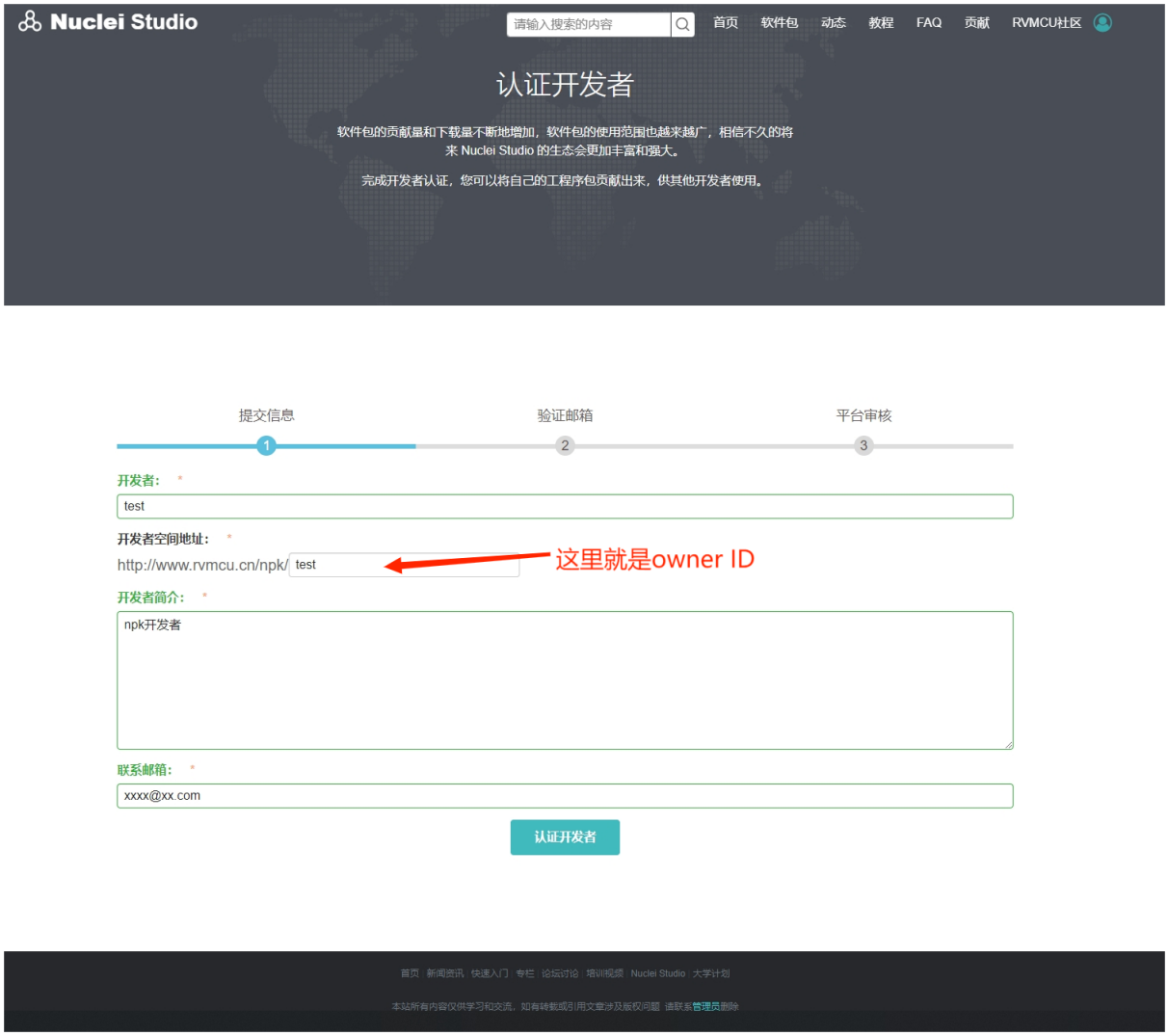
First, log in to Nuclei System Technology ' s RVMCU community website[2]. After logging in, go to the Nuclei Studio -> Contribute page, where you can see the authentication button.

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



Alternatively, visit the authentication page[3] directly and fill in the required information. The suffix of the developer space address corresponding to **Developer** is your owner ID.

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

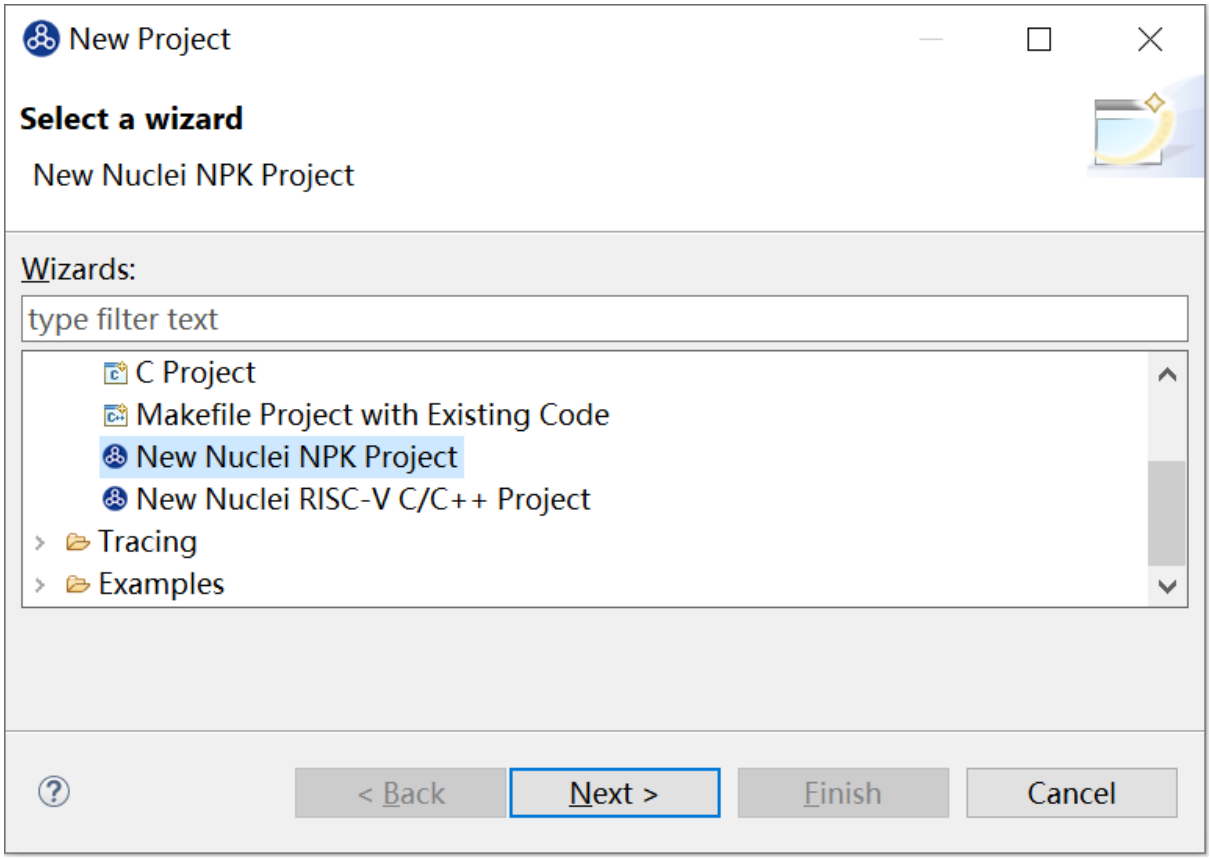


After filling in the required information, submit it. Once the system review is approved, the authentication is complete.

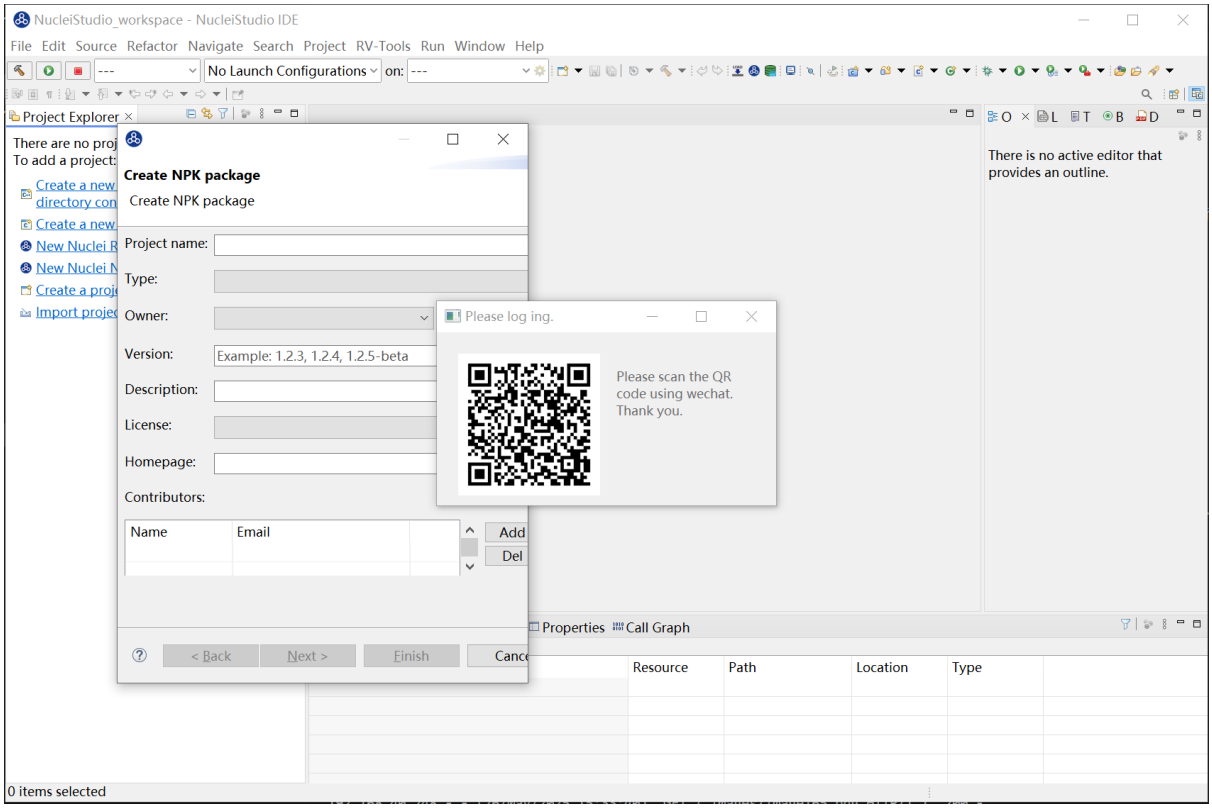


Create an NPK Component Package

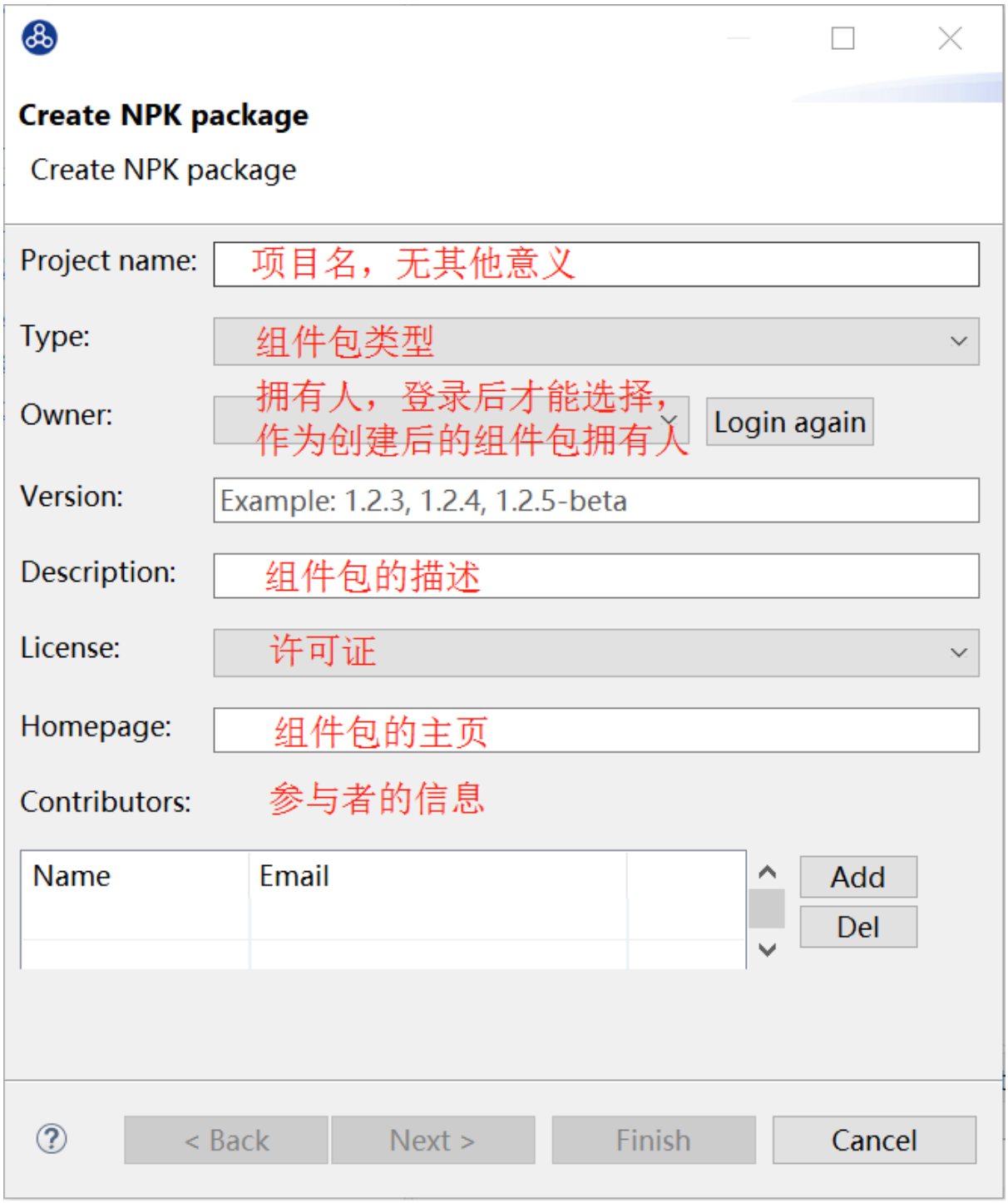
After completing developer authentication, create a new Nuclei Studio component project by selecting `File-> New-> Project-> C/C++-> New Nuclei NPK Project` in the menu bar.



For first-time use, log in with a developer account. The login is valid for 7 days; after that, you must log in again. Login requires scanning a QR code with WeChat, as shown in the figure below. After a successful login, the owner drop-down box lists the bound developer accounts; fill in the remaining information. To switch accounts, click login again.



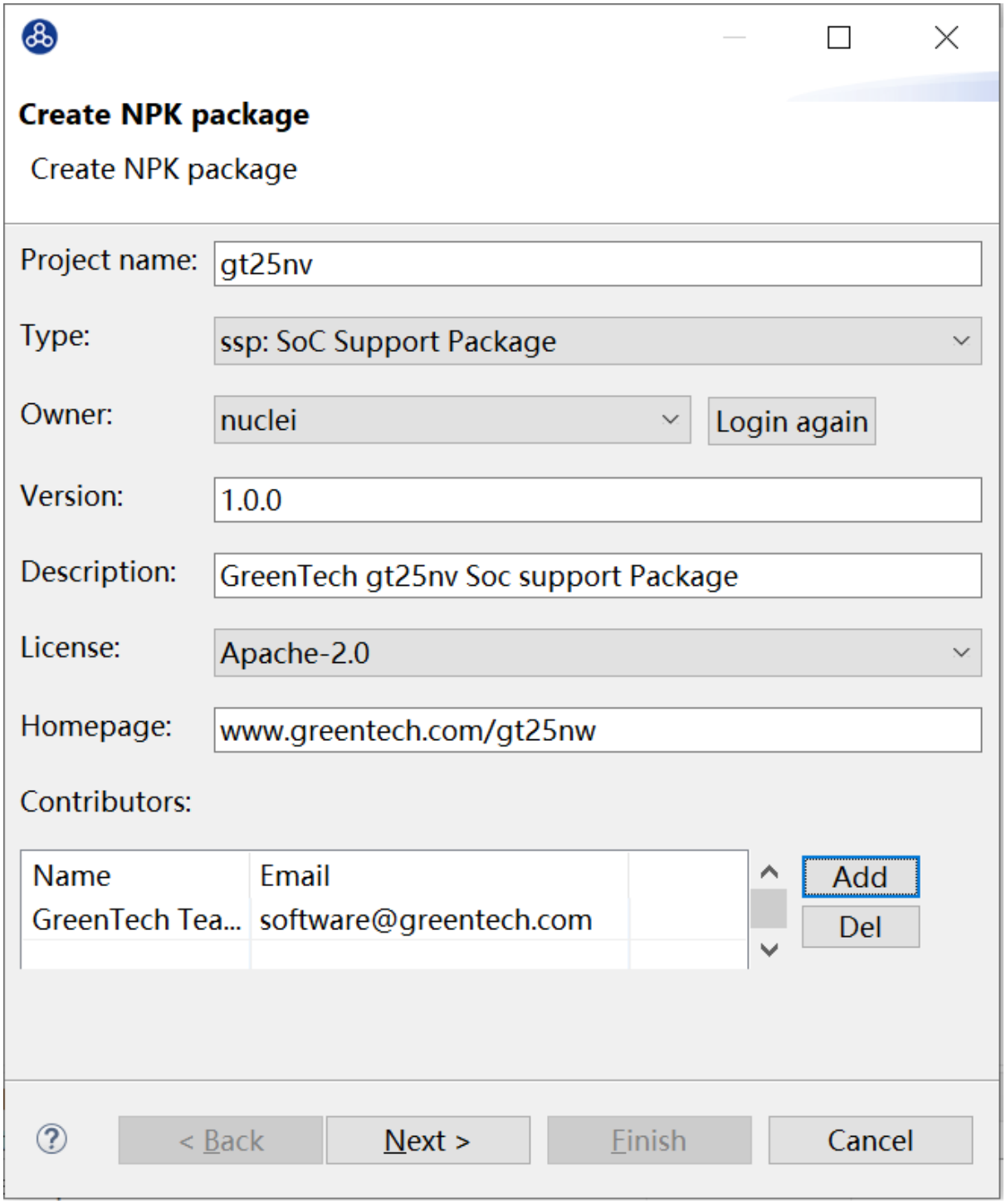
After the developer account is logged in, fill in the project name, project type, and other related information following the project wizard.



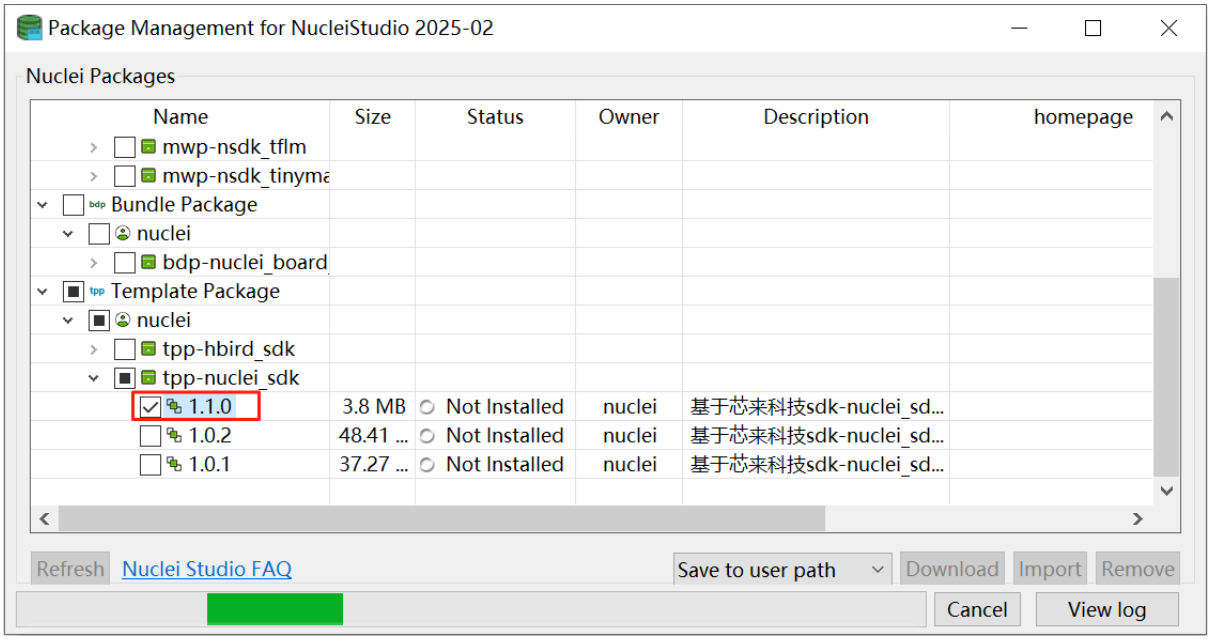
When you select a Type for which no corresponding template exists, a prompt appears. Click OK to open the Nuclei Package Management page and download the corresponding Template Package as needed, or click Import in the lower right corner of the figure below to import it yourself.

The following example uses the ssp type template. Suppose your company name is GreenTech, your SoC name is gt25nv, the adapted board is gt25nv_fpga_eval, it uses the N307FD processor (rv32imafdc) configuration with DSP features configured, and it provides three download modes: ilm, flash, and flashxip.

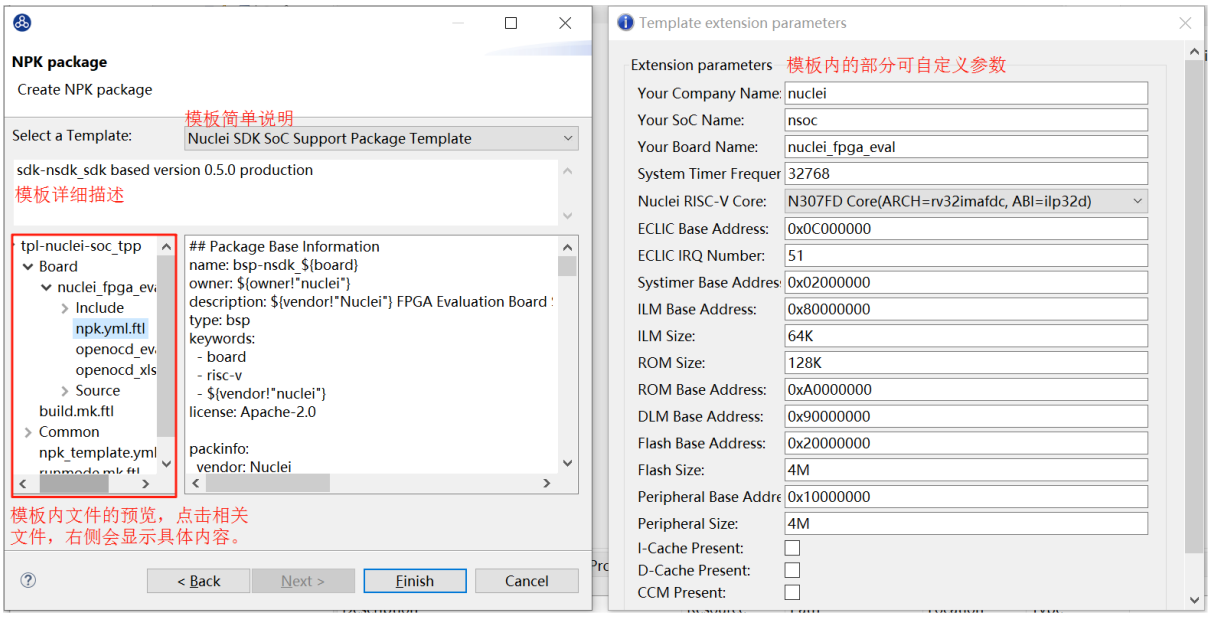
Select ssp for Type: `SoC Support Package`



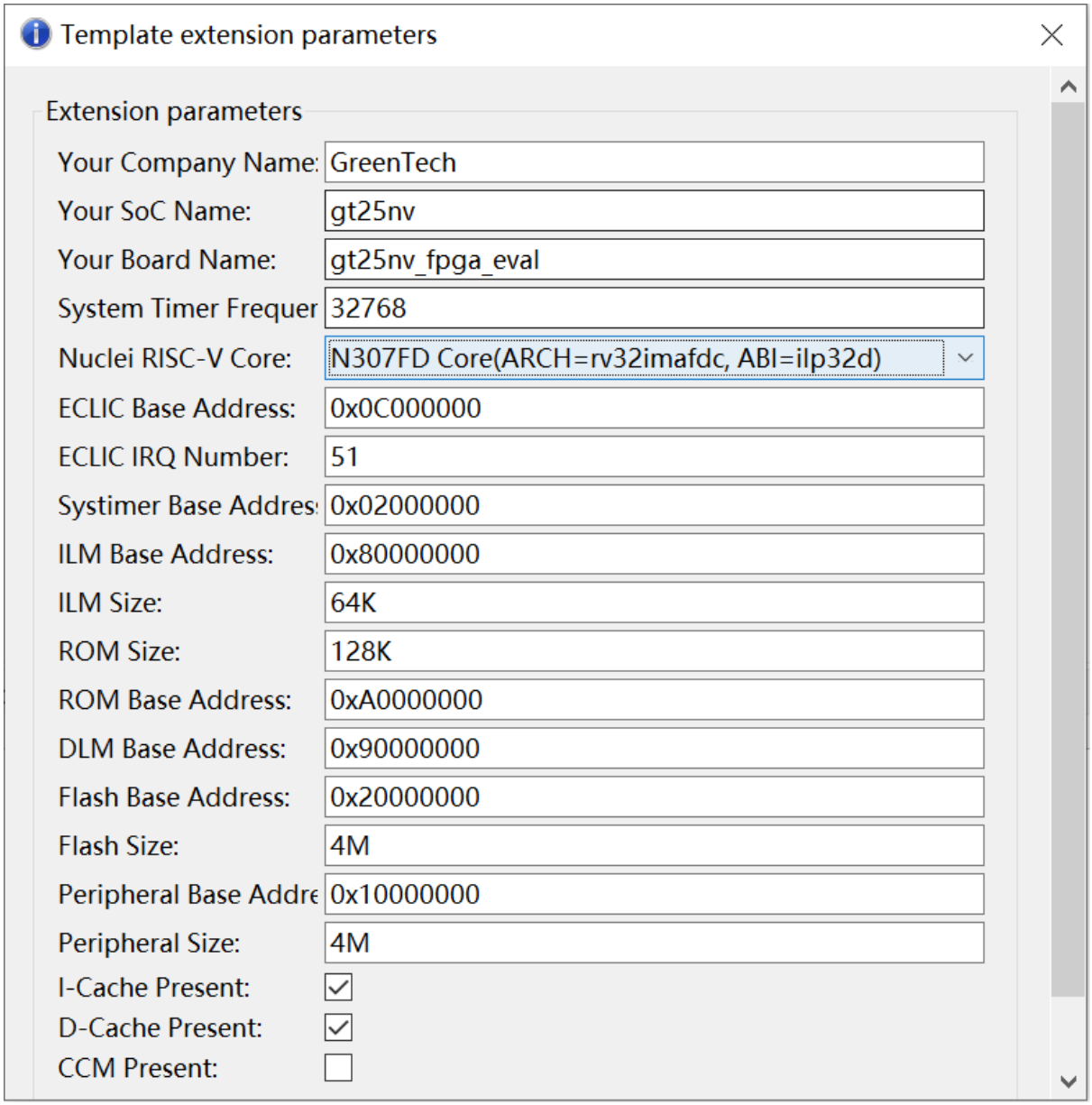
When you select a Type with no related template, you are taken to the Nuclei Package Management page. On this page, select version 1.1.0 of `tpp-nuclei_sdk`, click `Download`, and close the page after the download is complete.



Click Next, and select the `tpp-nuclei_sdk` template you just downloaded in `Select a Template`. The left side shows the template description and a preview of the related files; the right side shows part of the customizable content in the template.

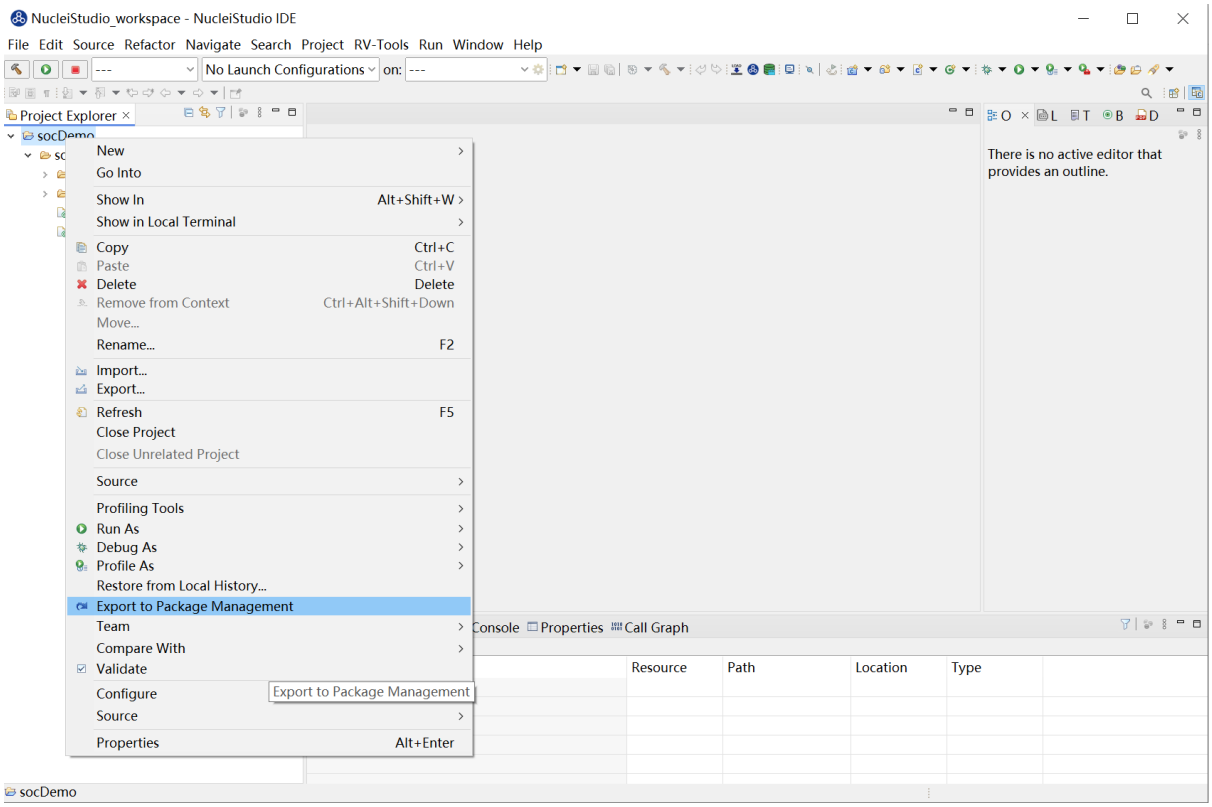


In this example, the company name is `GreenTech`, the SoC name is `gt25nv`, the adapted board is `gt25nv_fpga_eval`, the configuration uses the N307FD processor (`rv32imafdc`) with DSP features configured, and it provides three download modes: ilm, flash, and flashxip. The result after modification is shown in the figure.

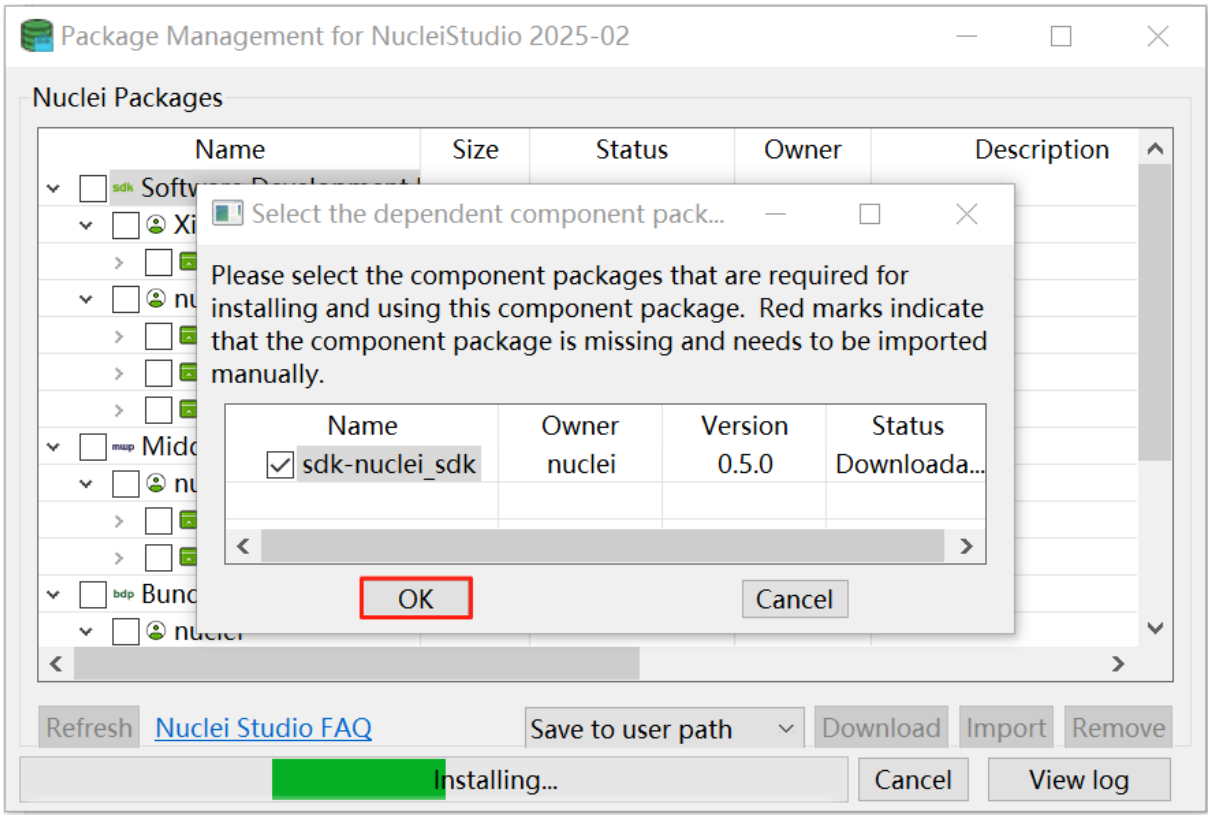


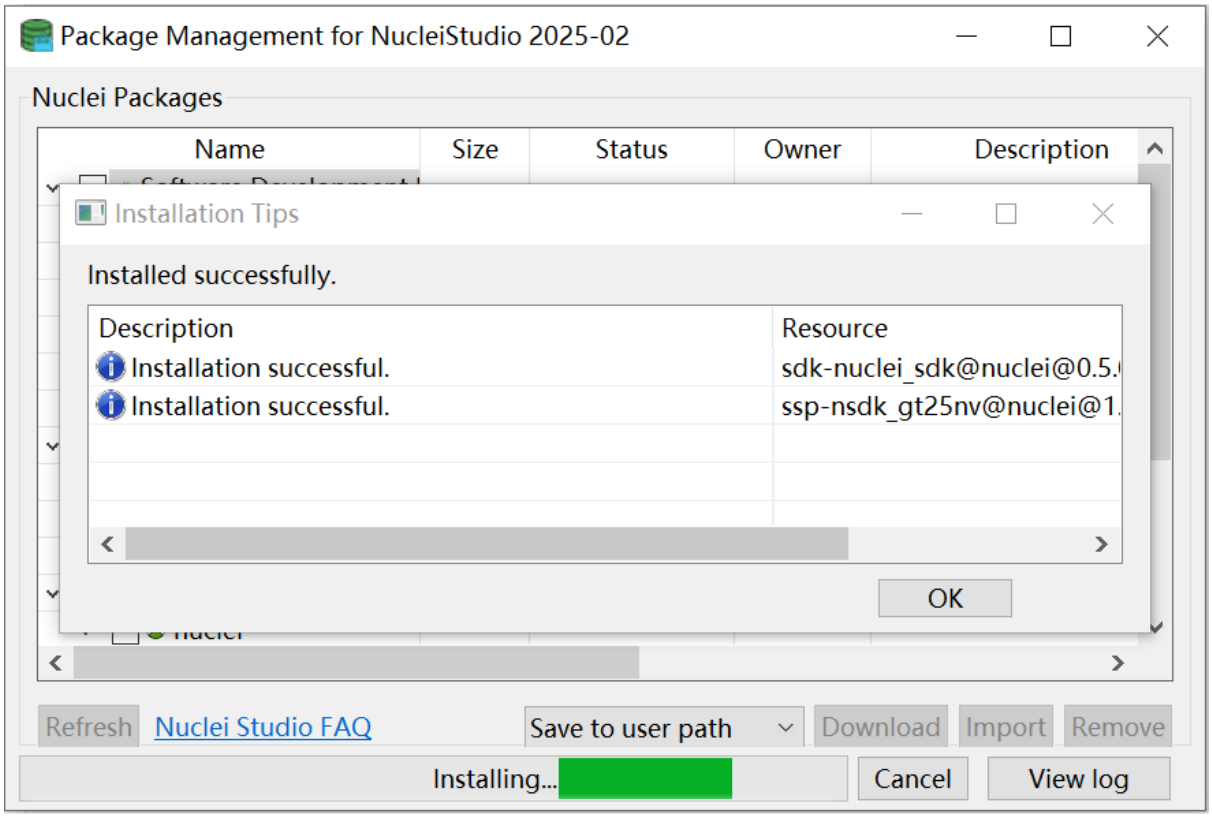
Click Finish to generate the NPK component package. You can then open the directory to view and modify the files and structure as needed, and import the NPK component package after the modification is complete.

Modify the project as needed, then import the NPK component package once the modification is complete.

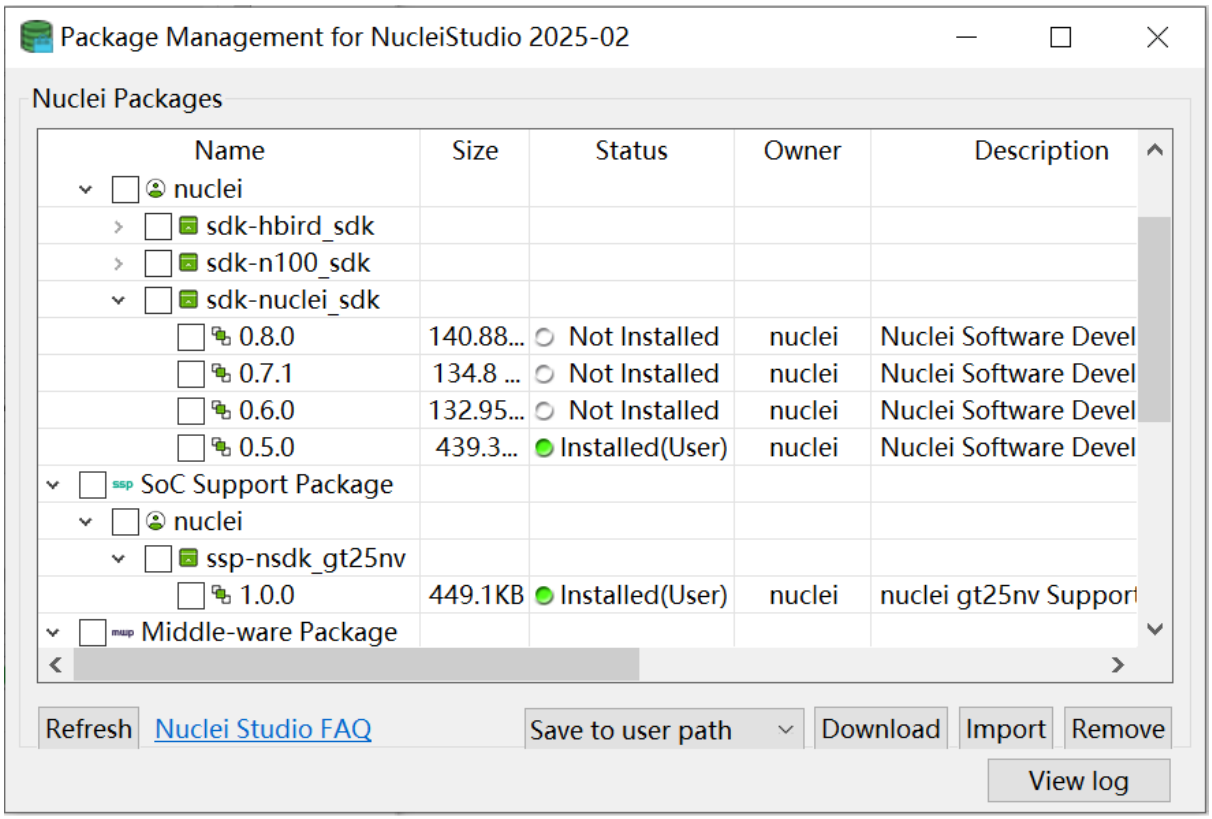


When importing the SoC package created in the example, a prompt indicates that the dependency `sdk-nuclei_sdk` is missing. Click OK to open the `Nuclei Package Management` page, select `sdk-nuclei_sdk`, and click `Download`.





After the download is complete and the NPK component package project has been imported successfully, the result is shown in the figure below.



2.5.2 Checking and Troubleshooting NPK Component Packages

Checking a Newly Created Package Before Import

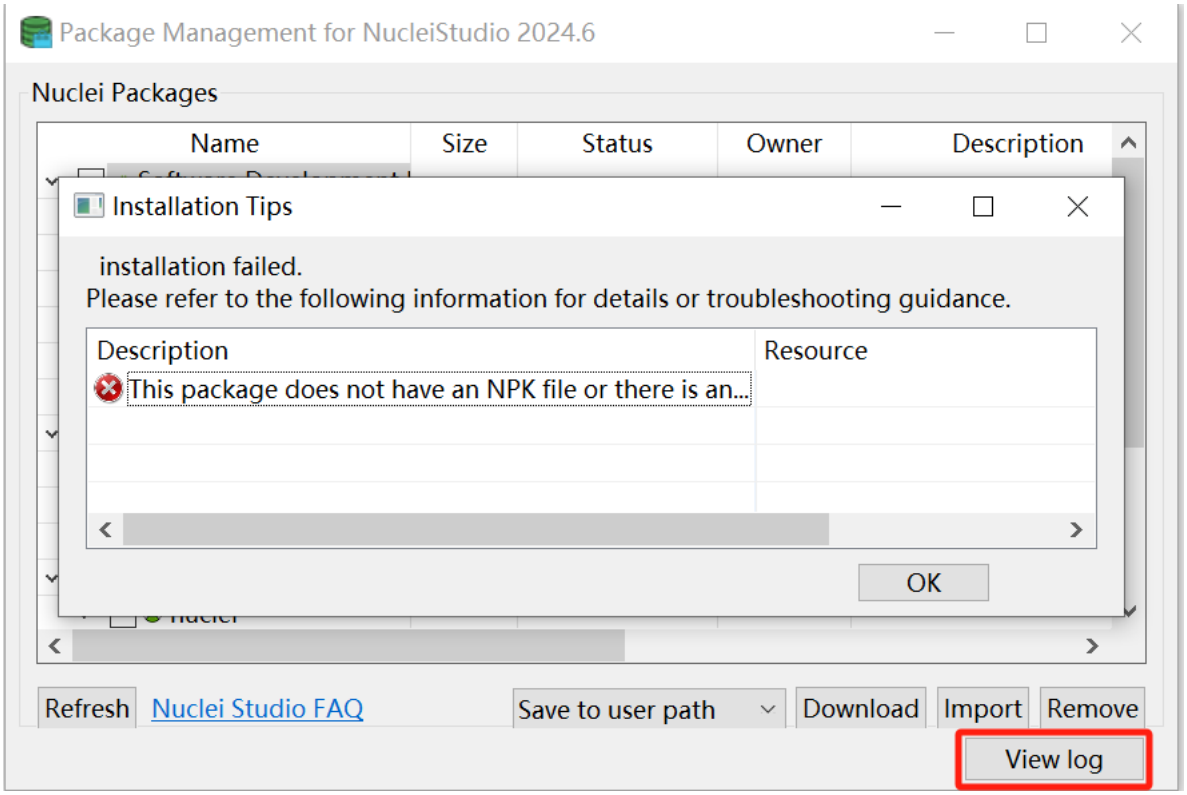
Before importing a completed package, you can use the checking tool to verify the validity of the NPK component package and avoid import failures. Its core functions focus on compliance checks of code style, variable naming, function structure, and dependency relationships to ensure code quality.

For details about this tool, see: Package checking tool[4]

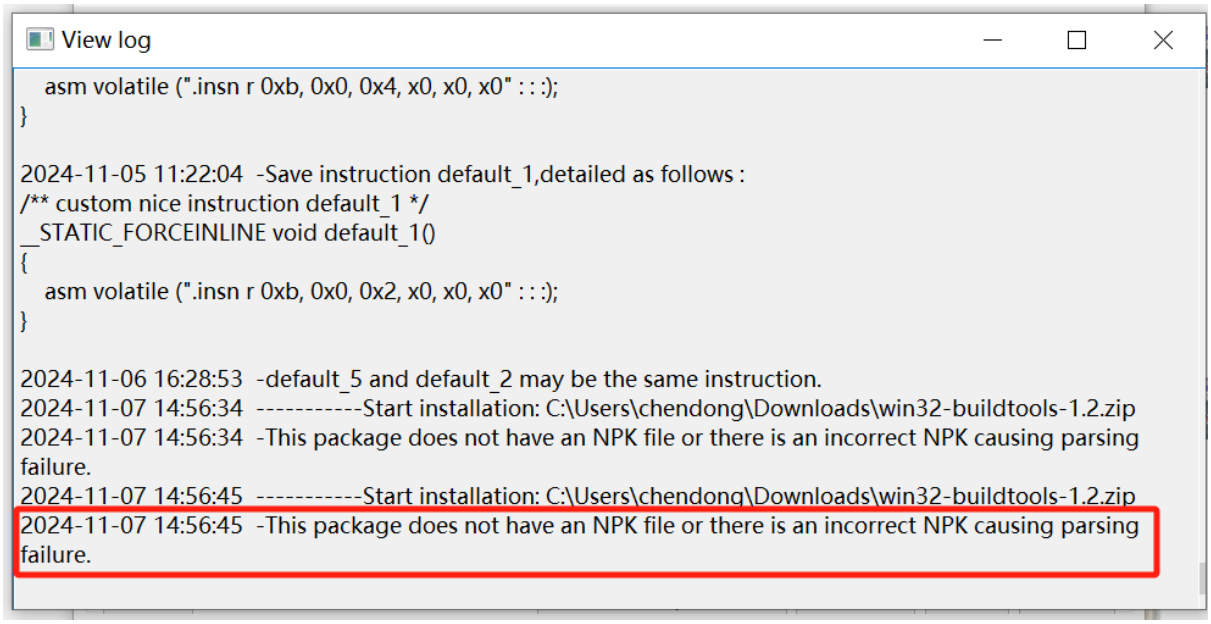
Handling Package Import Failures

When a package fails to import, you can identify the problem from the relevant logs. This section covers issues such as NPK file format errors and unsatisfied dependencies.

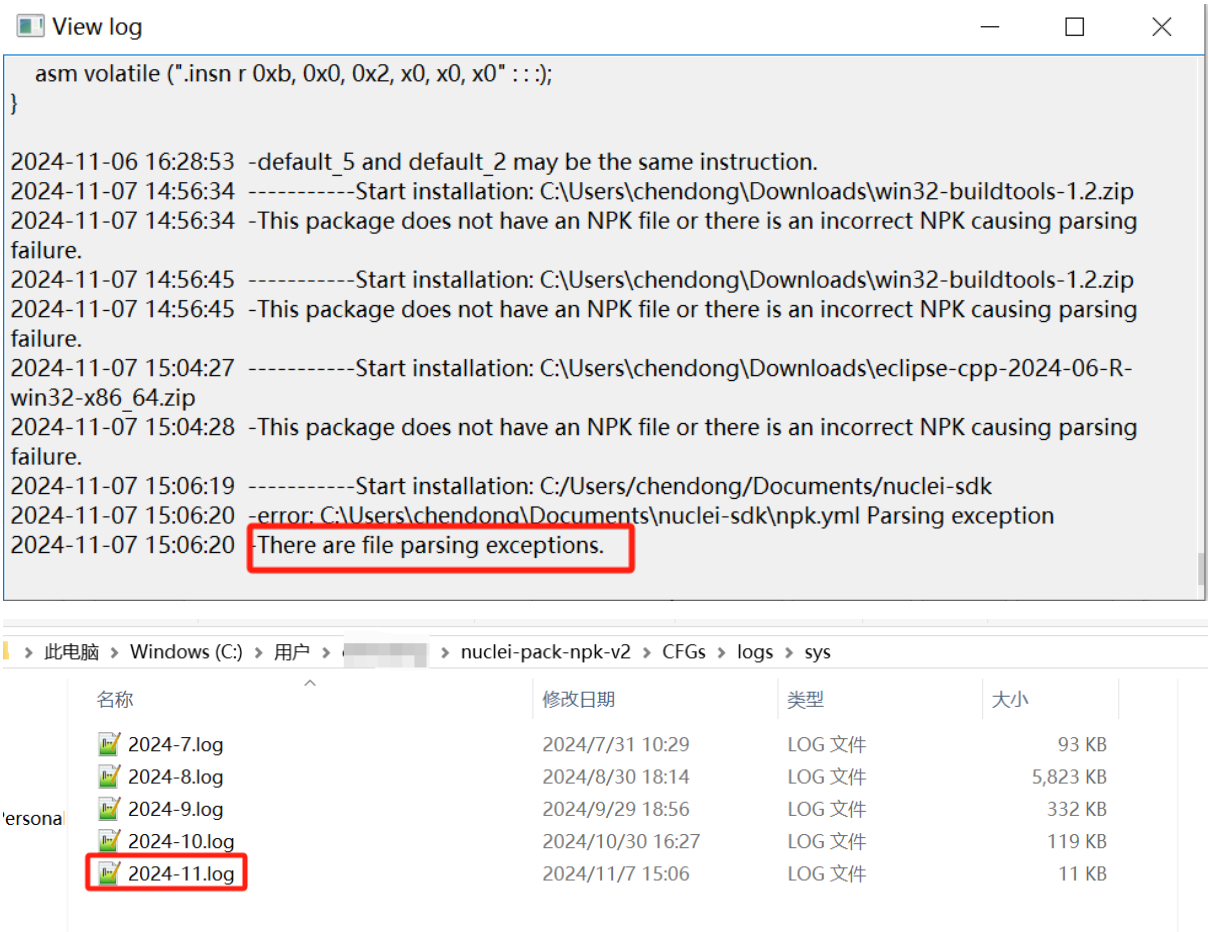
Click View log in the lower right corner of the Package Management page to open a log page that briefly describes the reason for the import failure.



[4] https://github.com/Nuclei-Software/npk-checker



When a message such as "There are file parsing exceptions" appears in the log, check the log file of that day under the `<user directory>\nuclei-pack-npk-v2\CFGs\logs\sys\` path for details about the problems in the related NPK files, and make the corresponding corrections.



```
2024-11.log
    at org.eclipse.equinox.launcher.Main.basicRun(Main.java:605)
    at org.eclipse.equinox.launcher.Main.run(Main.java:1481)

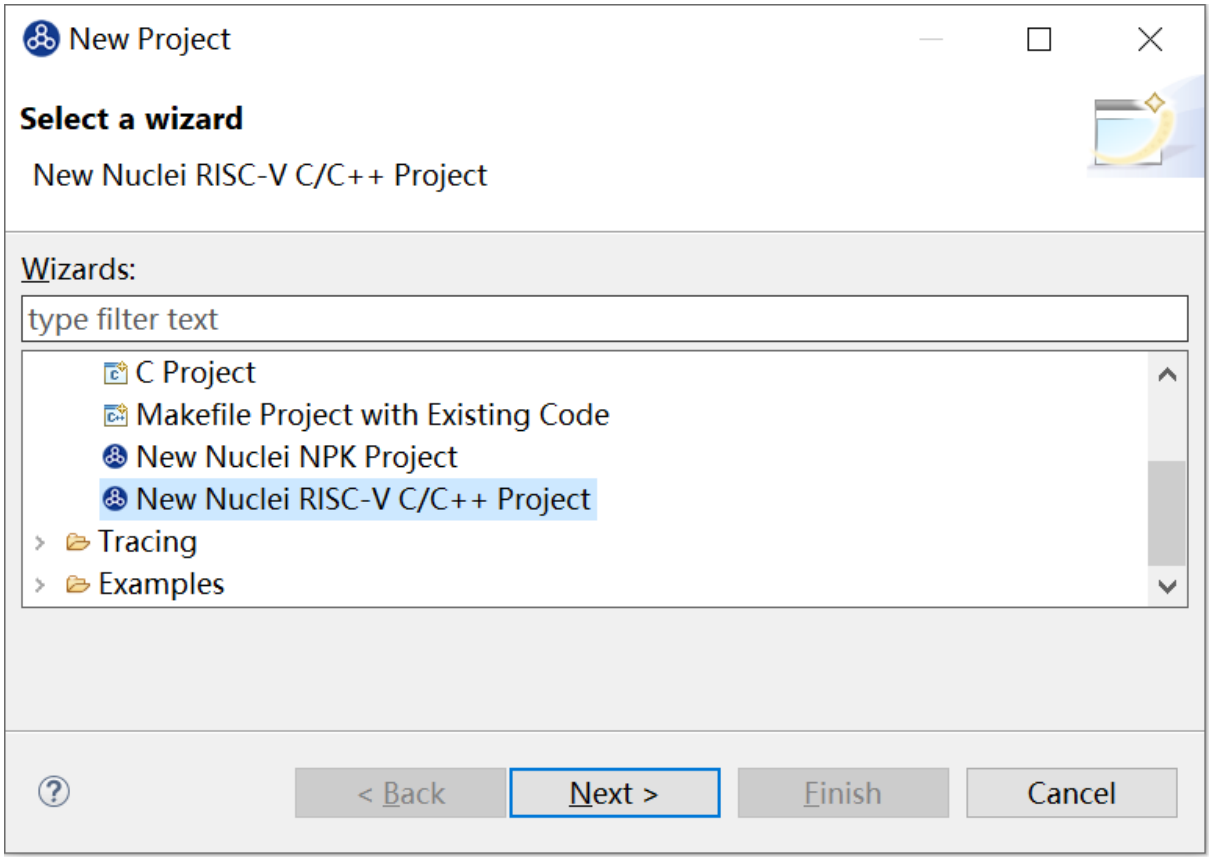
2024-11-07 15:06:20  -mapping values are not allowed here
in 'reader', line 8, column 10:
    keywords:
             ^

    at org.yaml.snakeyaml.scanner.ScannerImpl.fetchValue(ScannerImpl.java:870)
    at org.yaml.snakeyaml.scanner.ScannerImpl.fetchMoreTokens(ScannerImpl.java:358)
    at org.yaml.snakeyaml.scanner.ScannerImpl.checkToken(ScannerImpl.java:227)
    at org.yaml.snakeyaml.parser.ParserImpl$ParseBlockMappingKey.produce(ParserImpl.java:558)
    at org.yaml.snakeyaml.parser.ParserImpl.peekEvent(ParserImpl.java:158)
    at org.yaml.snakeyaml.parser.ParserImpl.checkEvent(ParserImpl.java:148)
    at org.yaml.snakeyaml.composer.Composer.composeMappingNode(Composer.java:217)
    at org.yaml.snakeyaml.composer.Composer.composeNode(Composer.java:144)
    at org.yaml.snakeyaml.composer.Composer.getNode(Composer.java:85)
    at org.yaml.snakeyaml.composer.Composer.getSingleNode(Composer.java:108)
    at org.yaml.snakeyaml.constructor.BaseConstructor.getSingleData(BaseConstructor.java:141)
    at org.yaml.snakeyaml.Yaml.loadFromReader(Yaml.java:525)
    at org.yaml.snakeyaml.Yaml.loadAs(Yaml.java:519)
    at org.riscvstudio.ide.handlers.PackageHandler.getPackageYmlIndexs(PackageHandler.java:93)
    at org.riscvstudio.ide.handlers.PackageHandler.getPackageYmlIndexDtosOfOne(PackageHandler.java:124)
    at org.riscvstudio.ide.pack.ui.PackManage$13.run(PackManage.java:1209)
    at java.base/java.lang.Thread.run(Thread.java:1583)

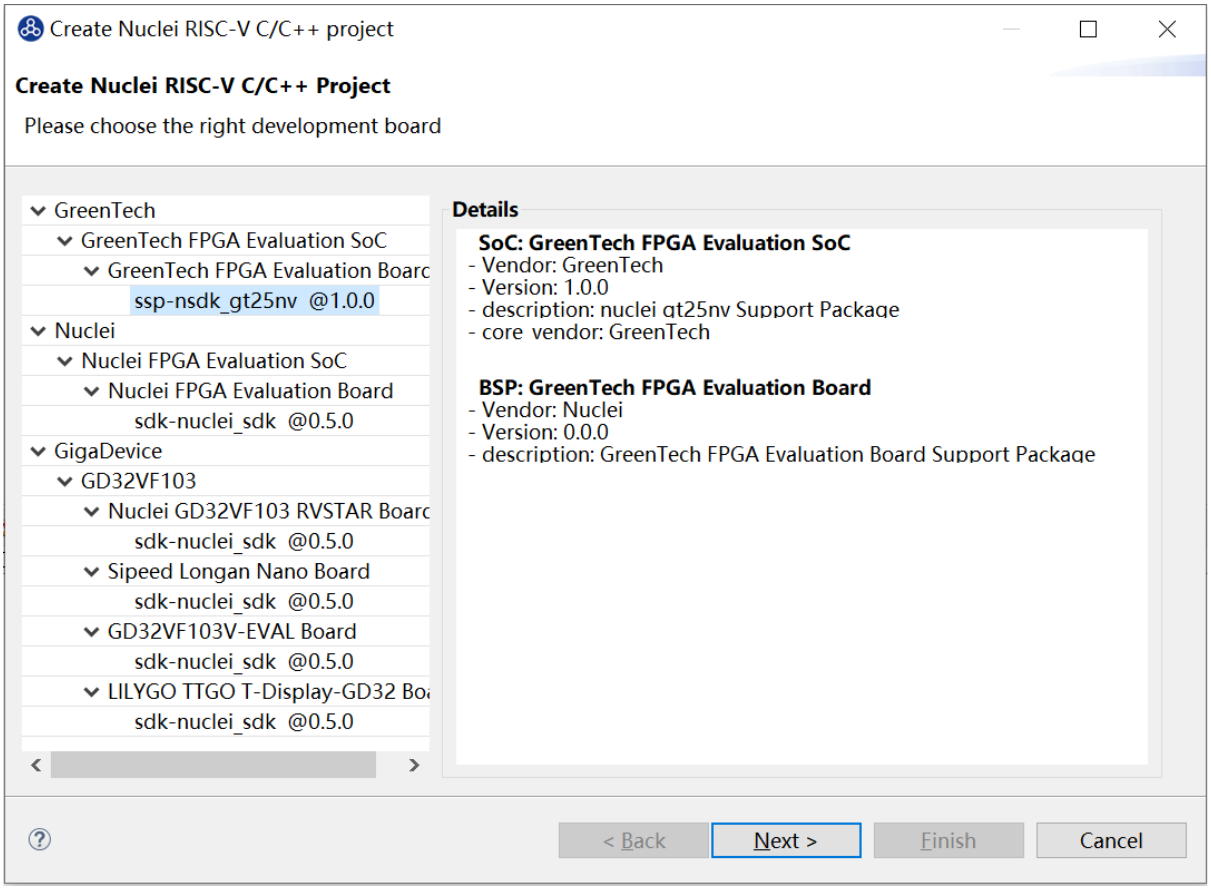
2024-11-07 15:06:20  -java.lang.Exception
    at org.riscvstudio.ide.pack.ui.PackManage$13.run(PackManage.java:1211)
    at java.base/java.lang.Thread.run(Thread.java:1583)
```

Create a Test Project

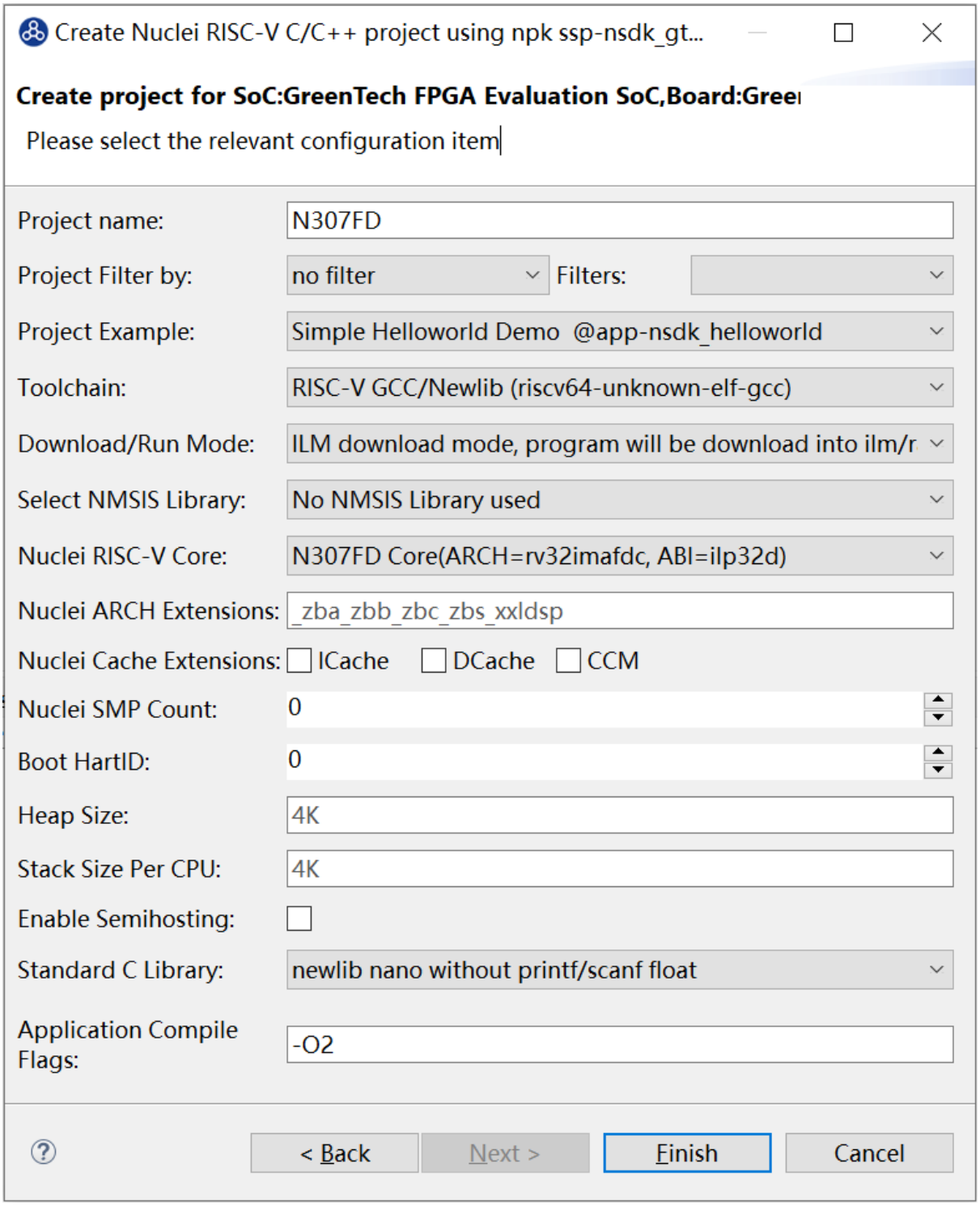
To create a project using the imported NPK component package project, select `File-> New-> Project-> C/C++-> New Nuclei RISC-V C/C++ Project` in the menu bar.



Click Next, and select the SoC and board contained in the package created above.



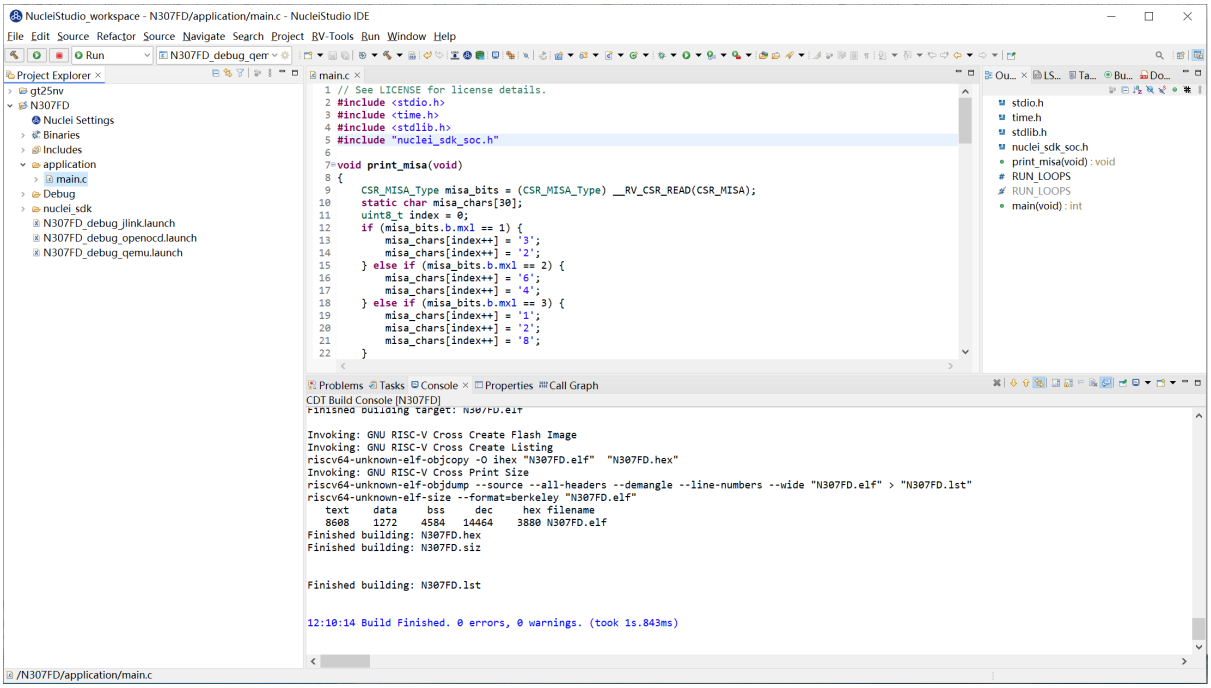
Click Next, fill in the Project Name, select `Simple Helloworld Demo` for Project Example, and click `Finish` to complete the creation of the test project.


Create Nuclei RISC-V C/C++ project using npk ssp-nsdk_gt...
Create project for SoC:GreenTech FPGA Evaluation SoC,Board:Gree
Please select the relevant configuration item
Project name:
N307FD
Project Filter by:
no filter
Filters:
Project Example:
Simple Helloworld Demo @app-nsdk_helloworld
Toolchain:
RISC-V GCC/Newlib (riscv64-unknown-elf-gcc)
Download/Run Mode:
ILM download mode, program will be download into ilm/r
Select NMSIS Library:
No NMSIS Library used
Nuclei RISC-V Core:
N307FD Core(ARCH=rv32imafdc, ABI=ilp32d)
Nuclei ARCH Extensions:
_zba_zbb_zbc_zbs_xxldsp
Nuclei Cache Extensions:
ICache
DCache
CCM
Nuclei SMP Count:
0
Boot HartID:
0
Heap Size:
4K
Stack Size Per CPU:
4K
Enable Semihosting:
Standard C Library:
newlib nano without printf/scanf float
Application Compile Flags:
-O2
< Back
Next >
Finish
Cancel

Build and Debug the Test Project

The project created in the steps above is a test project based on the developer ' s NPK component package. The developer can code, debug, run, and perform other operations on it as with a normal project.

Click to select the N307FD project generated in the previous step and build it; the build succeeds. Subsequent steps such as running are omitted here. At this point, an NPK component package has been successfully created and imported for use.

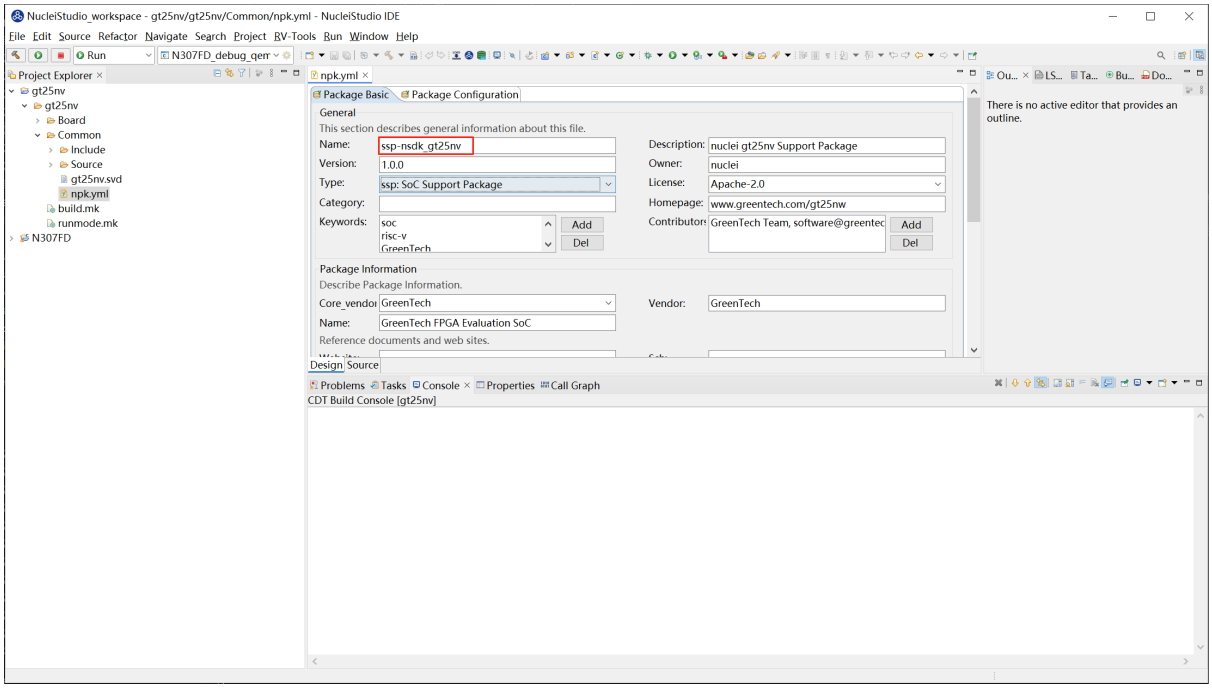


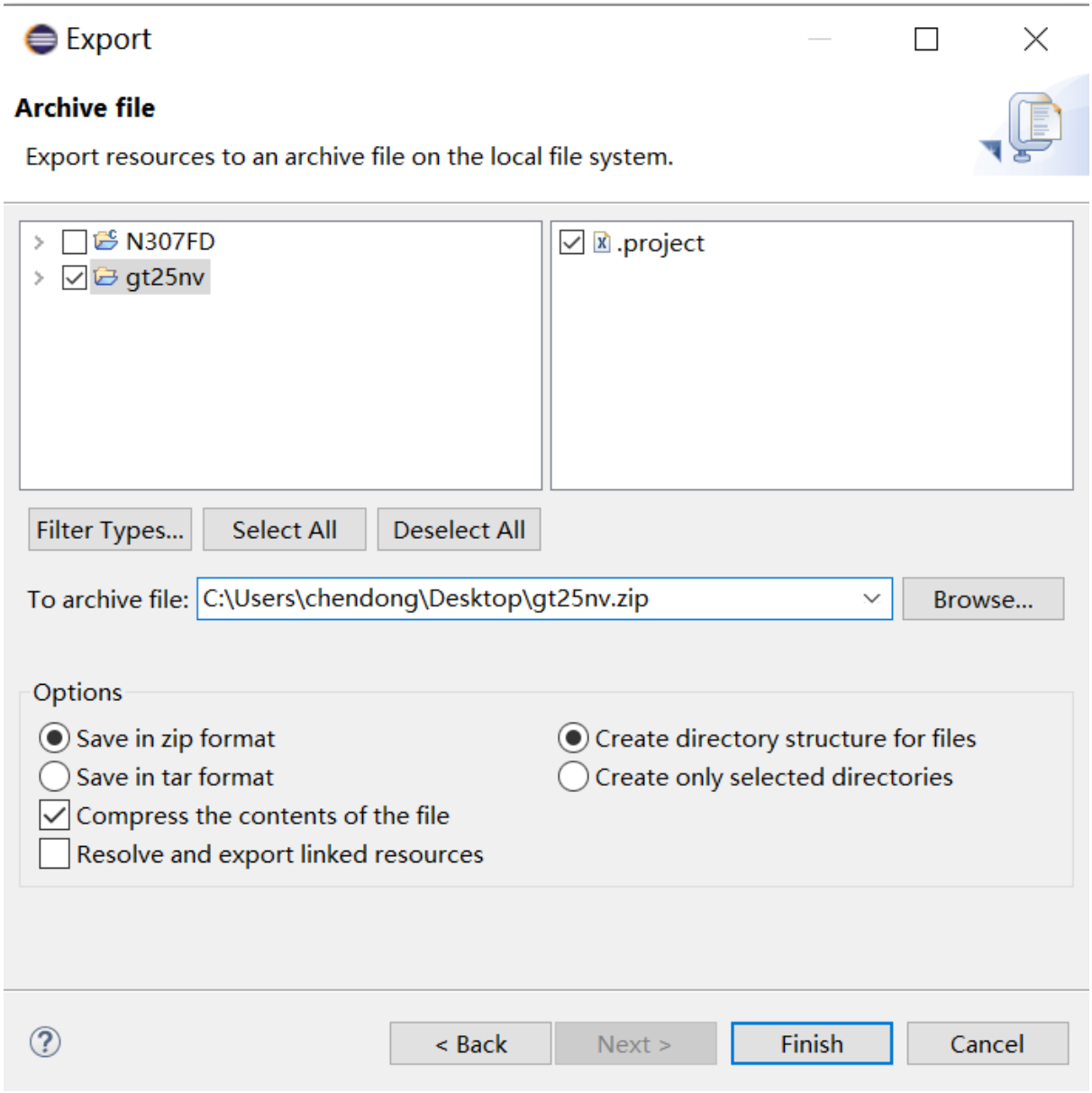
2.5.3 Share an NPK Component Package

Sharing an NPK Component Package

After passing the tests, you can share the NPK component package you created. First, export your NPK component package project as a zip package as follows.

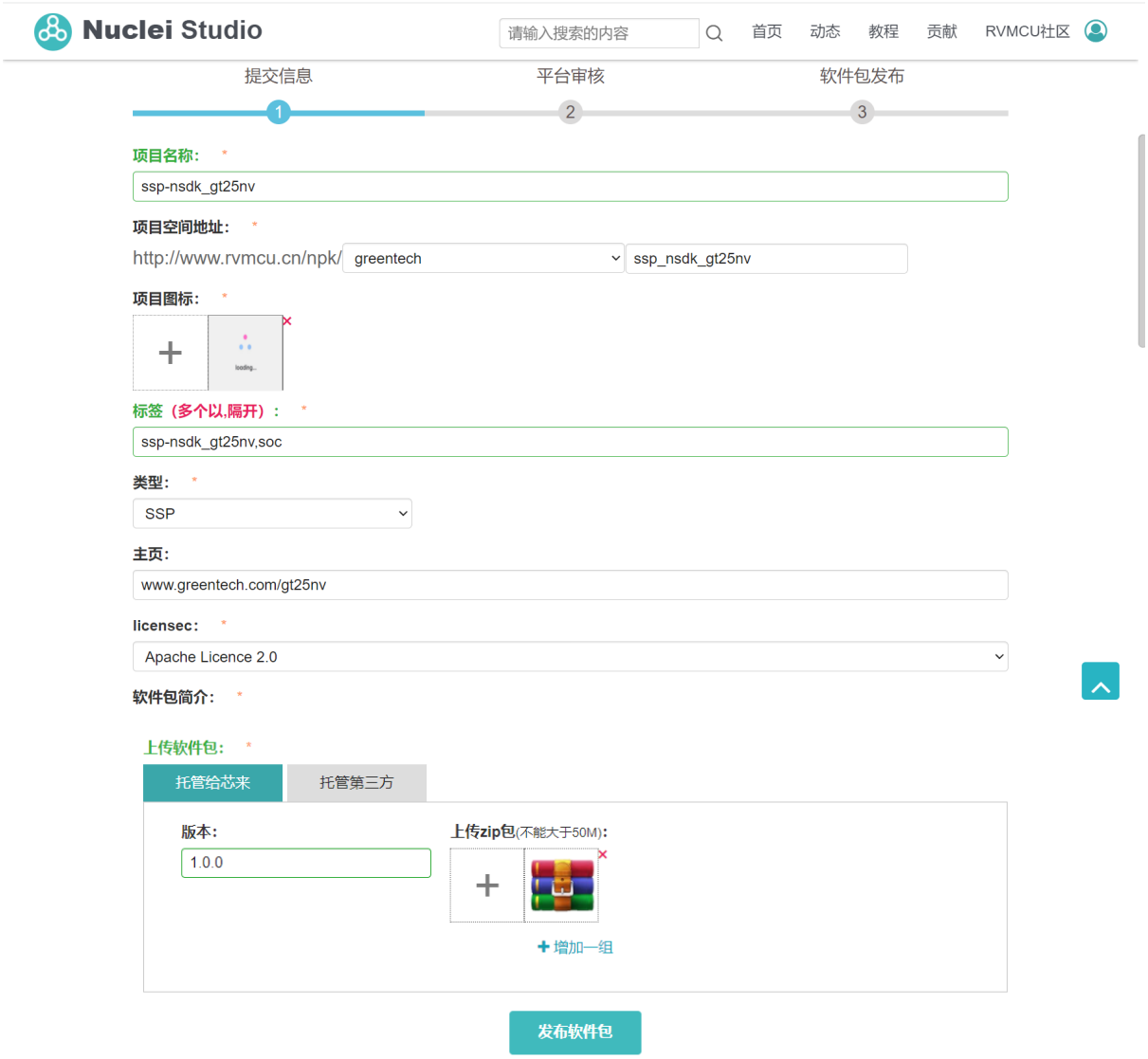
Open the NPK component package project, double-click the outermost `npk.yml` file, and confirm that its Name is `ssp-nsdk_gt25nv`. Then right-click the NPK component package project, click `Export`, select `Archive File`, select the project to export, and specify where to save the exported zip file.


NucleiStudio_workspace - gt25nv/gt25nv/Common/npk.yml - NucleiStudio IDE
File Edit Source Refactor Navigate Search Project RV-Tools Run Window Help
Run
N307FD_debug_qem
Project Explorer
gt25nv
gt25nv
Board
Common
Include
Source
gt25nv.svd
npk.yml
build.mk
runmode.mk
N307FD
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:
nuclei gt25nv Support Package
Owner:
nuclei
License:
Apache-2.0
Homepage:
www.greentech.com/gt25nw
Contributors:
GreenTech Team, software@greentec
Add
Del
Package Information
Describe Package Information.
Core_vendor
GreenTech
Name:
GreenTech FPGA Evaluation SoC
Vendor:
GreenTech
Reference documents and web sites.
Design
Source
There is no active editor that provides an outline.
Problems
Tasks
Console
Properties
Call Graph
CDT Build Console [gt25nv]



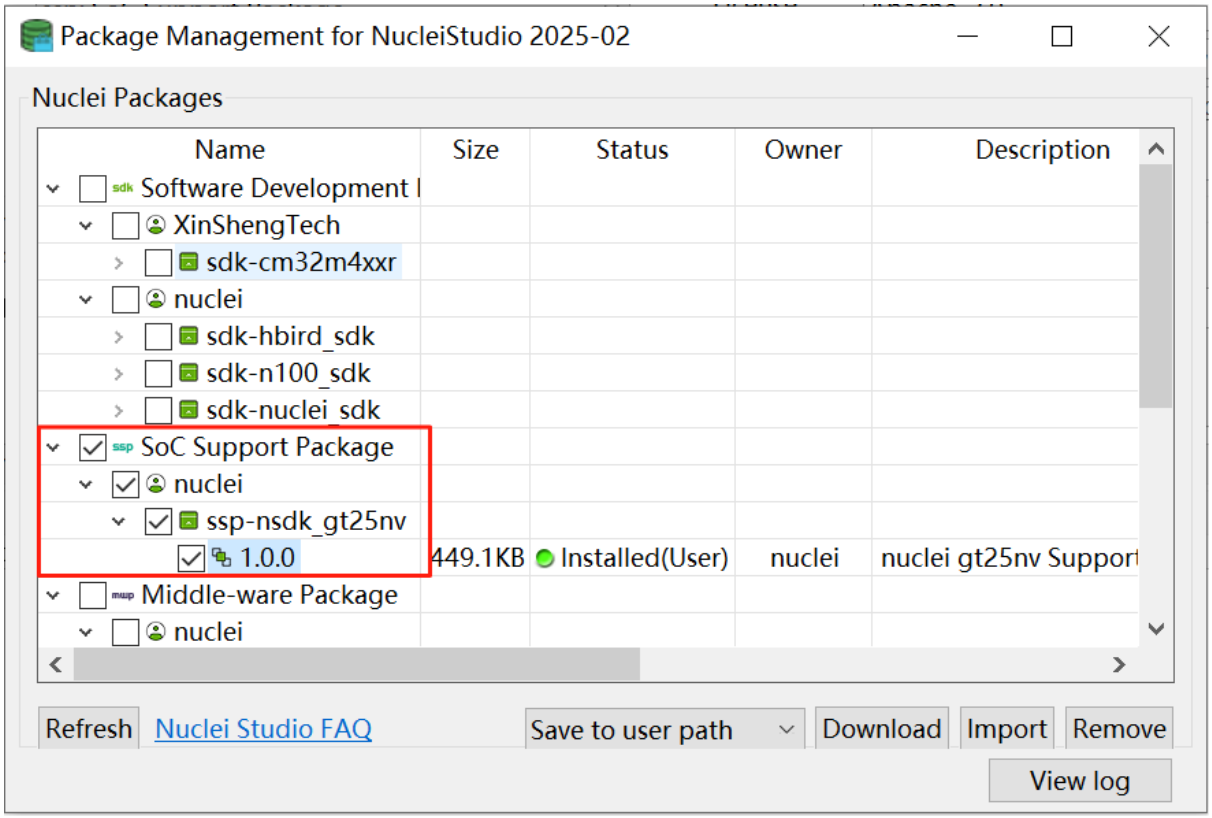
The exported zip package can be shared and contributed through the RVMCU community. Go to the community sharing page[5], fill in the name, type, description, and other information of the NPK component package to be shared, and upload the zip file you just exported. After the information is submitted and approved by the administrator, the NPK component package is successfully contributed. Other developers can then find your NPK component package on the `Nuclei Package Management` page of Nuclei Studio and download it for use. The procedure is shown in the figures below.

[5] 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



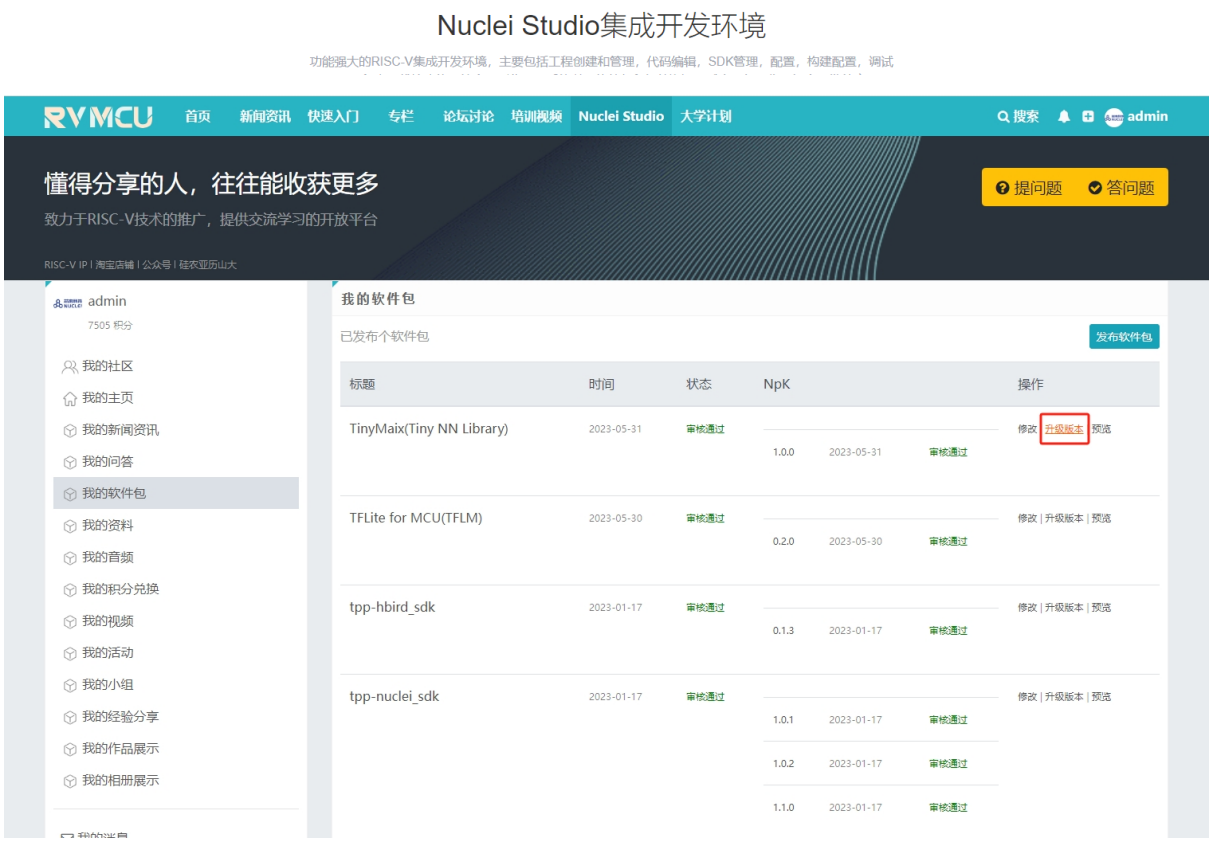
After the shared NPK component package is approved, open the `Nuclei Package Management` page in Nuclei Studio and click `Refresh`. The component package you just shared appears in the list.



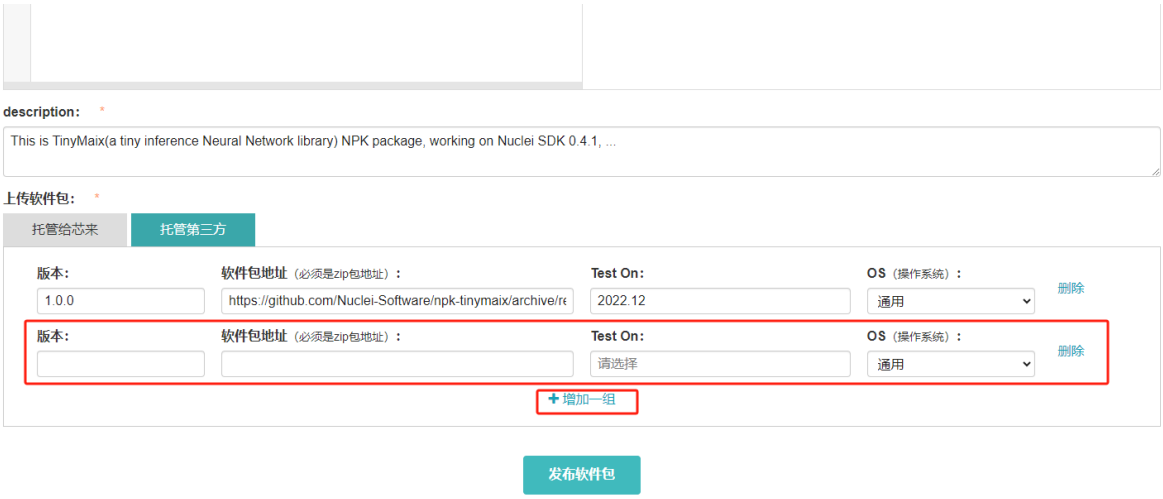
Upgrade an NPK Component Package

After an NPK component package is shared, if a new version needs to be maintained, you can upgrade the existing NPK component package once creation, testing, and packaging are complete. Go to the Nuclei Studio[6] page, find the component package management entry, open the component package management page, click Upgrade Component Package, and upload the NPK component package following the previous steps. After the review is approved, the component package upgrade is complete.





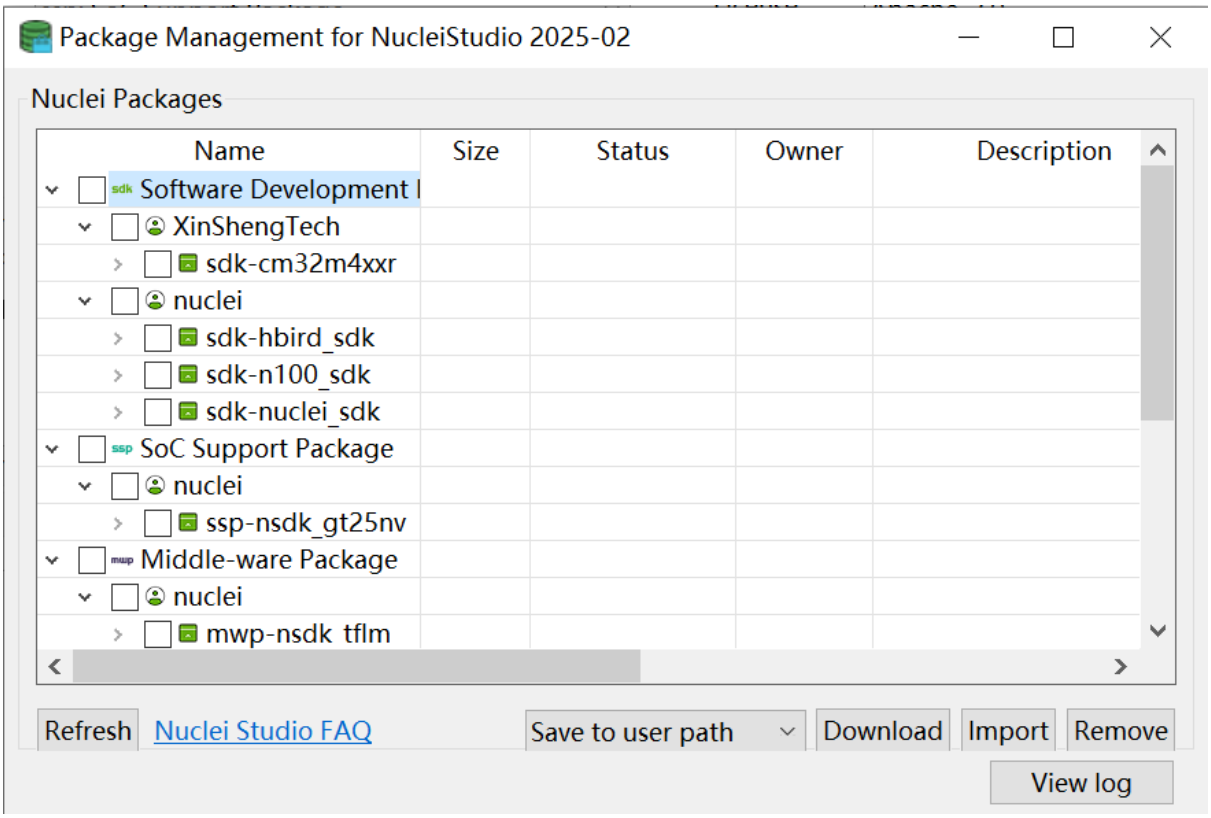
[6] https://www.rvmcu.com/nucleistudio.html



2.5.4 Using NPK Component Packages in Nuclei Studio

NPK component packages enrich the user experience in Nuclei Studio. With NPK component packages, you can define various project creation workflows and easily share mature projects or components with others.

All contributed NPK packages are managed in `NPK Package Management` of Nuclei Studio, where users can download, import, delete, and perform other operations on NPKs.



2.6 Nuclei Studio NPK Applications

Nuclei Studio provides full built-in support for the Nuclei Package (NPK) feature, making it easy for developers to create software development packages and import them into Nuclei Studio through Nuclei Package Management. CPU IP customers can adapt their own chip SDK to support NPK and import it into Nuclei Studio; SoC IP customers can package the provided SDK into a Zip package and import it; developers can also build a BSP package or APP package on top of a specific NPK and distribute it to third parties. For a detailed introduction to the NPK feature and its update notes, see https://github.com/Nuclei-Software/nuclei-sdk/wiki/Nuclei-Studio-NPK-Introduction. Adding NPK support and an npk.yml file to an existing SDK also enables partial customization of the Project Wizard functionality.

To create projects with Nuclei Studio, developers must first install the corresponding SDK NPK Zip package into the IDE. They can then quickly create template projects for different boards, include the required SDK source code based on the selected template, and generate the corresponding compile and link options based on the selected options.

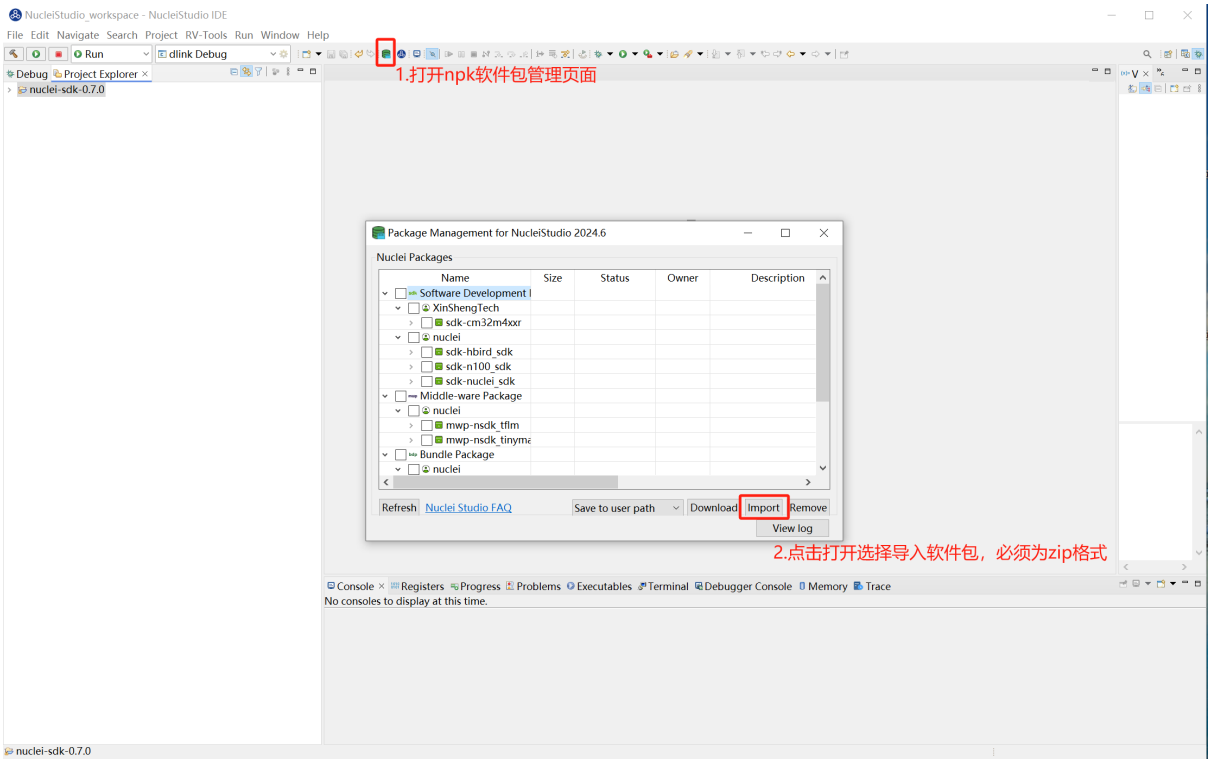
2.6.1 NPK Package Management

Import a Local NPK Package

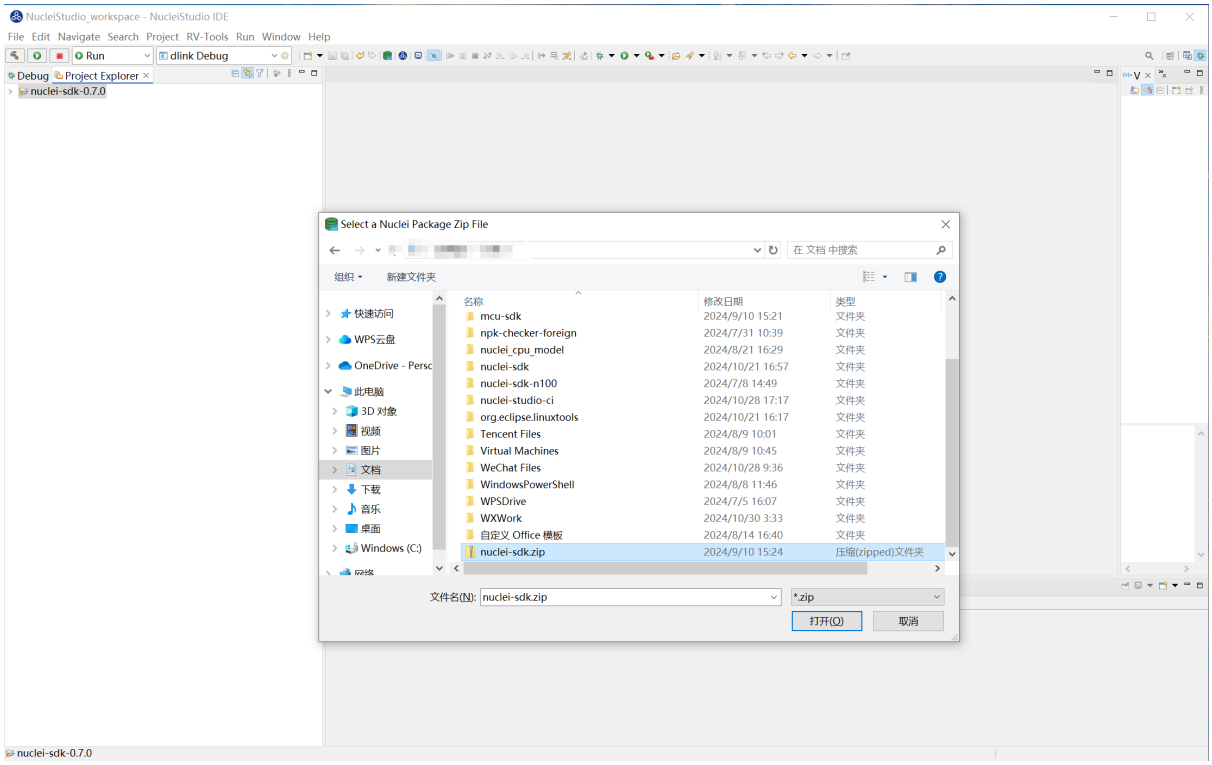
Import a zip package

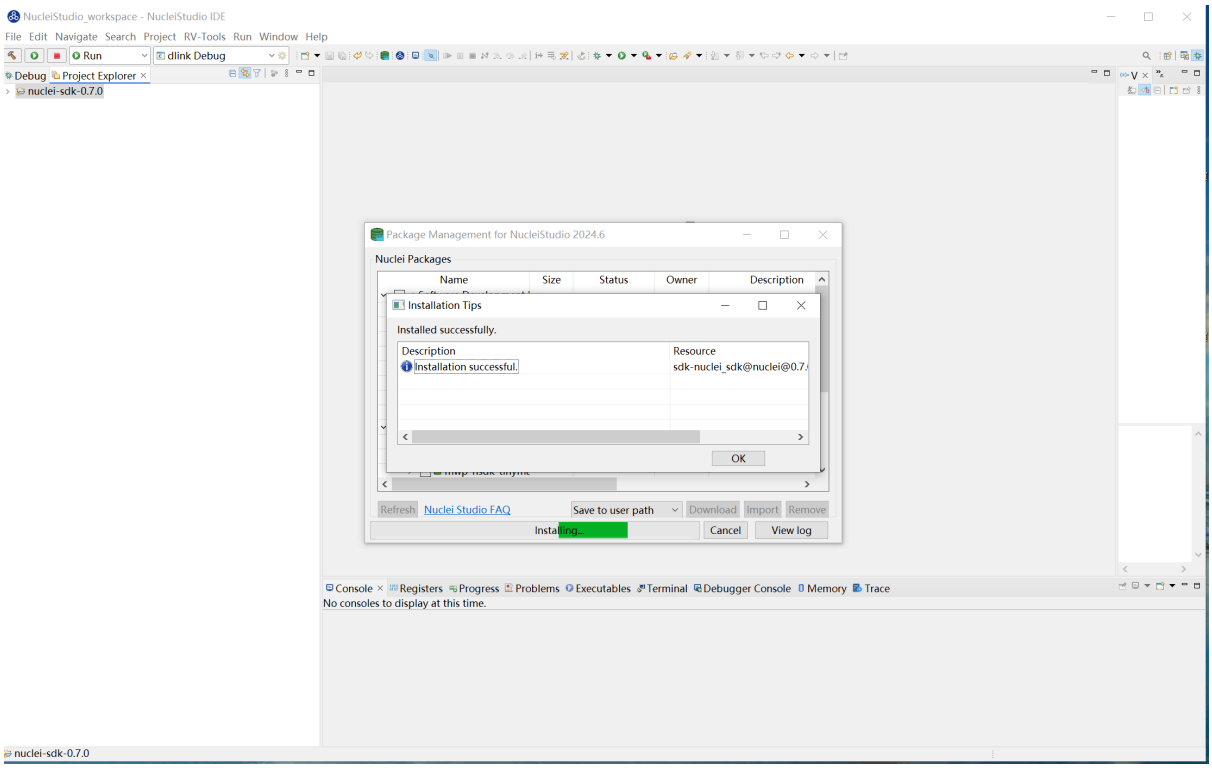
When you have a software package in zip format, import it through Nuclei Package Management as follows:

Click the **Nuclei Package Management** button on the Nuclei Studio IDE toolbar to open the NPK package management page, where you can browse and manage installed packages and import new ones. On this page, click **Import** to select a zip package from the local file system.

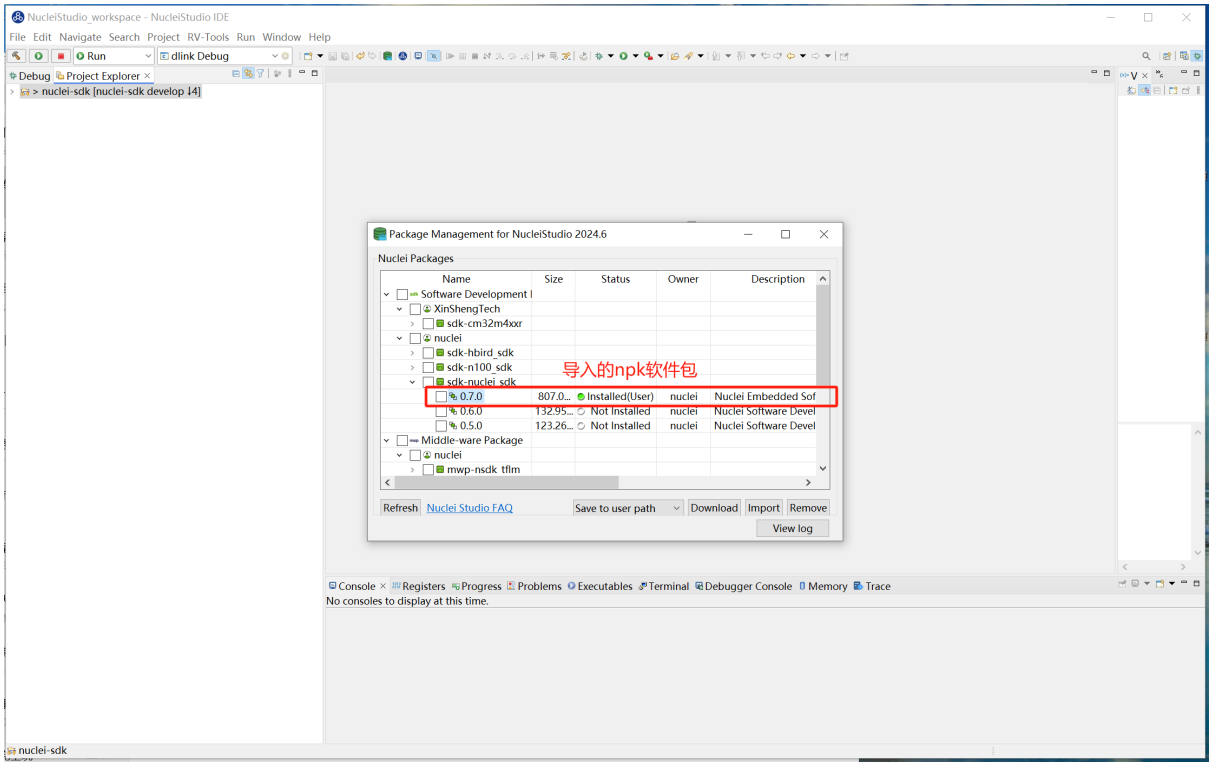


After you click **Import**, the system prompts you to select a package file to import. Browse to the directory containing your zip package and select it. Make sure a valid zip package is selected, then click **Open**. Nuclei Studio starts processing and importing the package.





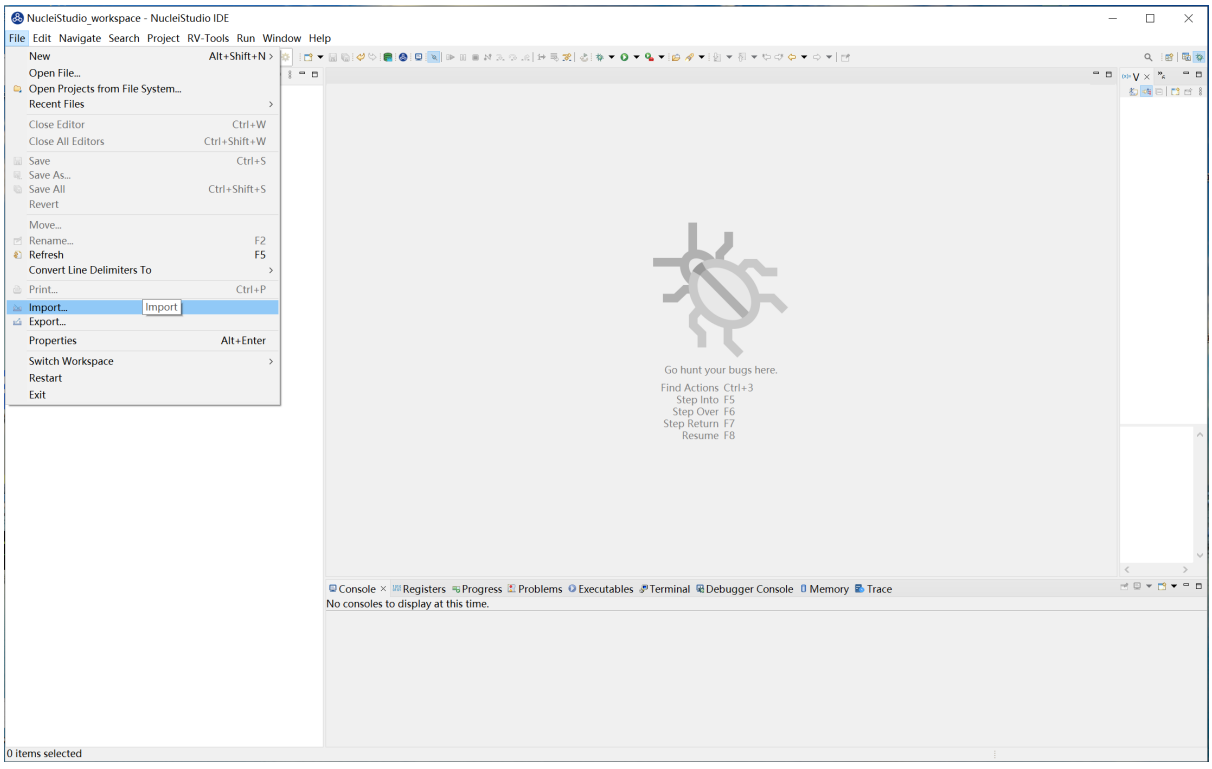
After the import is complete, the newly imported package appears on the Package Management page of Nuclei Studio IDE.



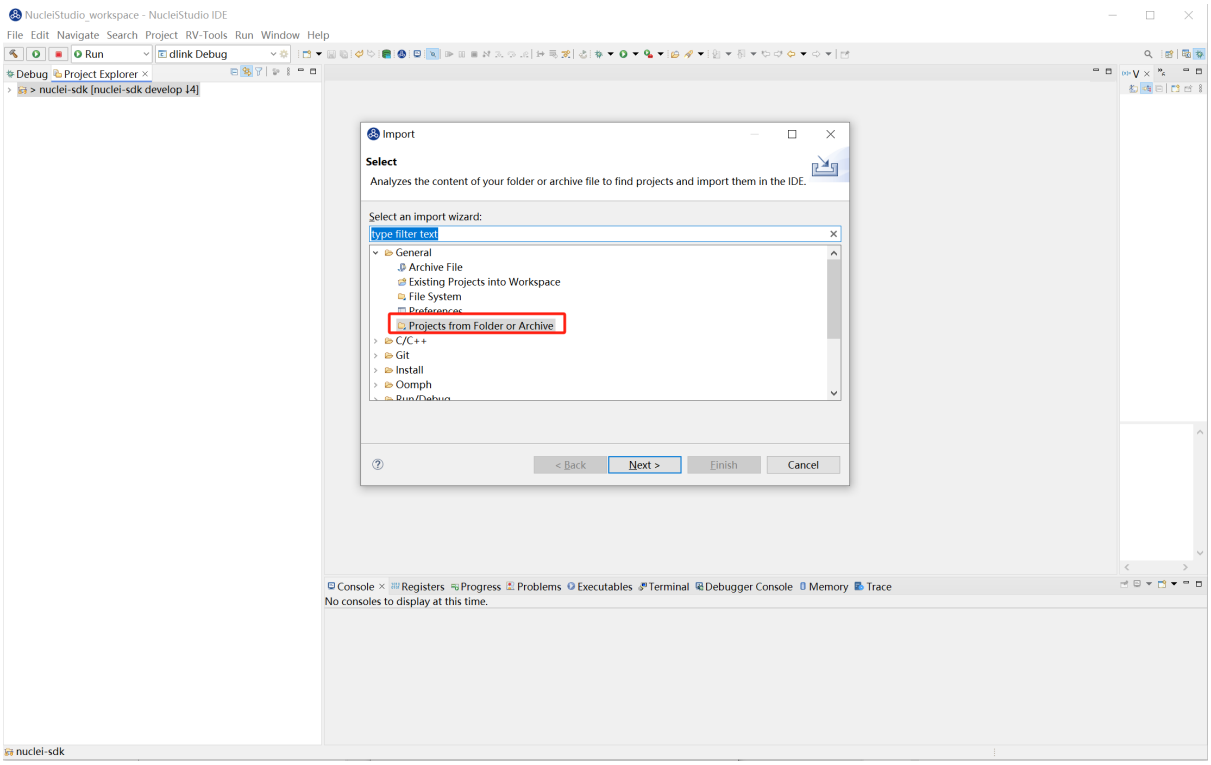
Import an uncompressed NPK package

When you have an uncompressed NPK package project to import into Nuclei Studio, the simplest way is to compress it into a .zip archive and import it using the procedure above. If you want to modify the NPK package project and quickly re-import it into Nuclei Studio for testing after each modification, do the following instead.

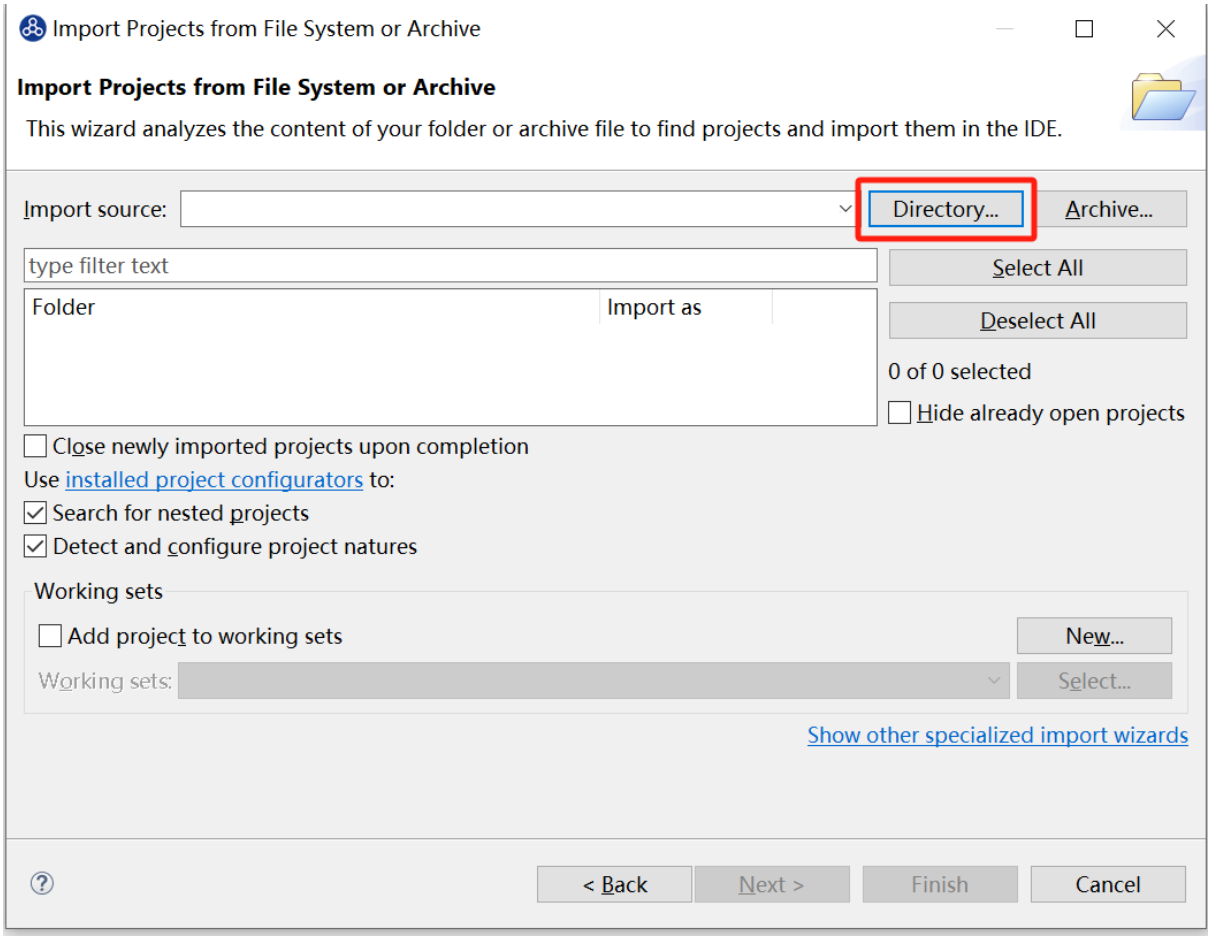
To import an uncompressed package directly, first import it into the Project Explorer view. Click File in the menu bar and select **Import**⋯ from the drop-down menu. This opens the import wizard, which imports projects or files from the local file system or other sources.



Select the **Projects from Folder or Archive** option under General, and click Next.

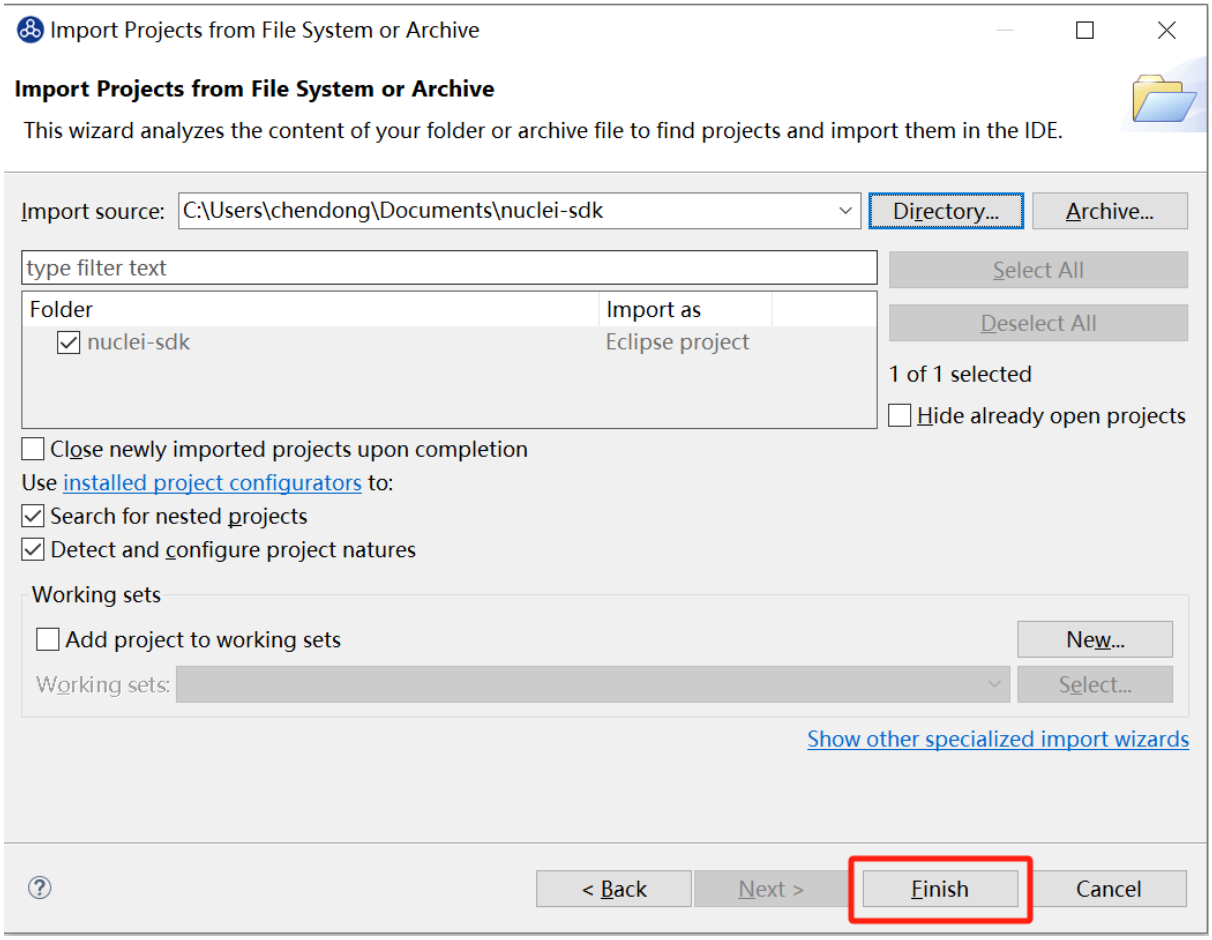


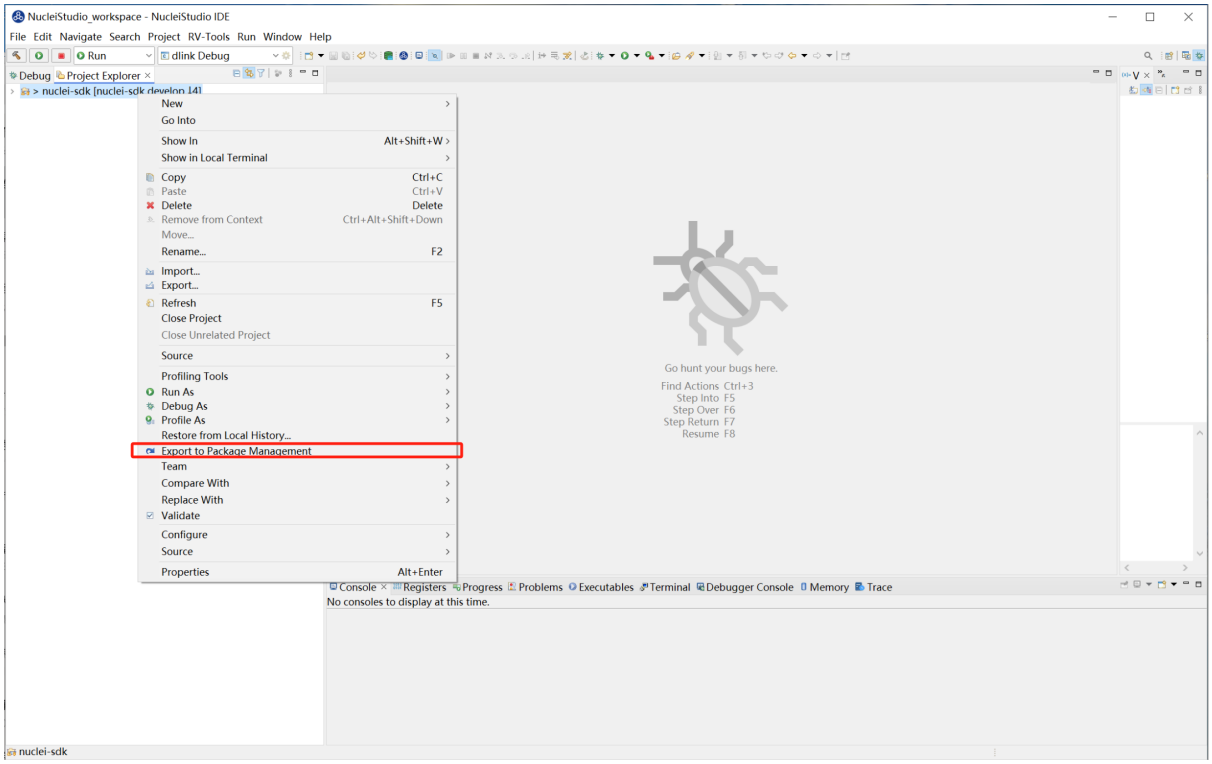
Click **Directory**⋯, browse to the directory containing your package, and select the file or folder to import. Verify that the correct path and files are selected, then click **Finish**.



The package is now imported into the **Project Explorer** view. Right-click the folder representing your package

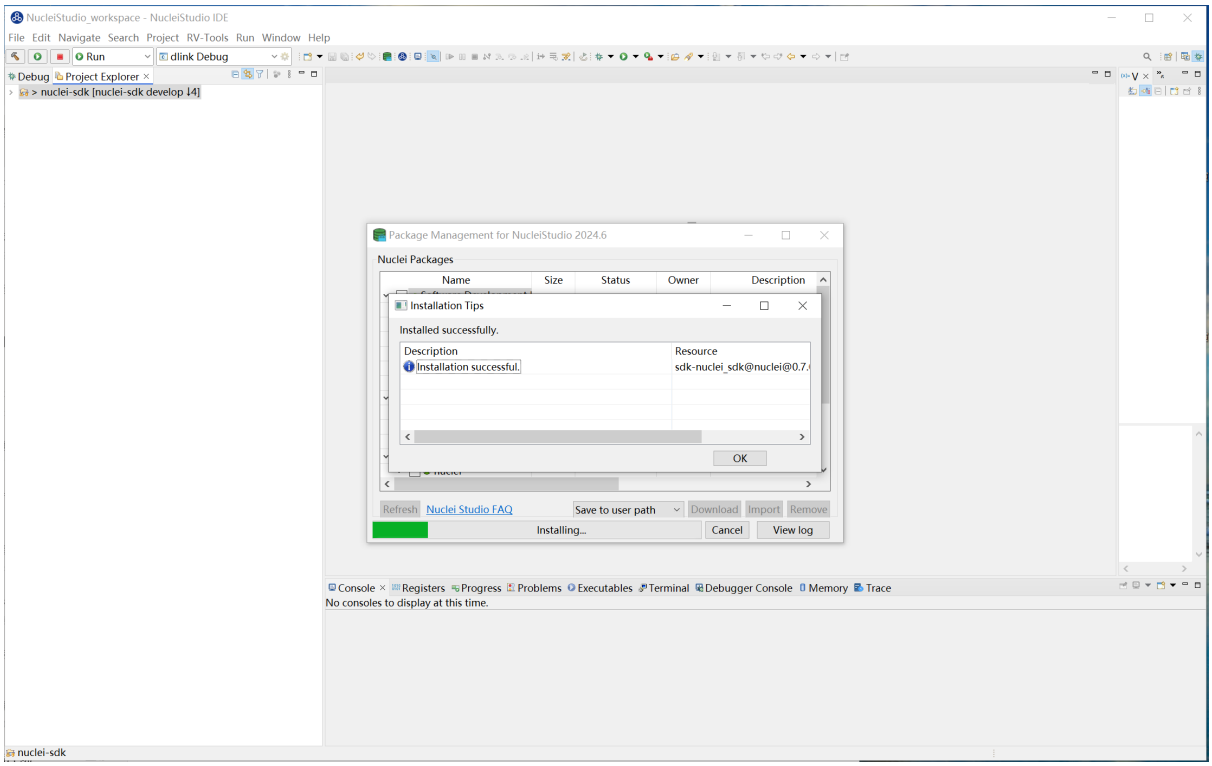
and select **Export to Package Management** from the context menu. Nuclei Studio starts processing and importing the package.





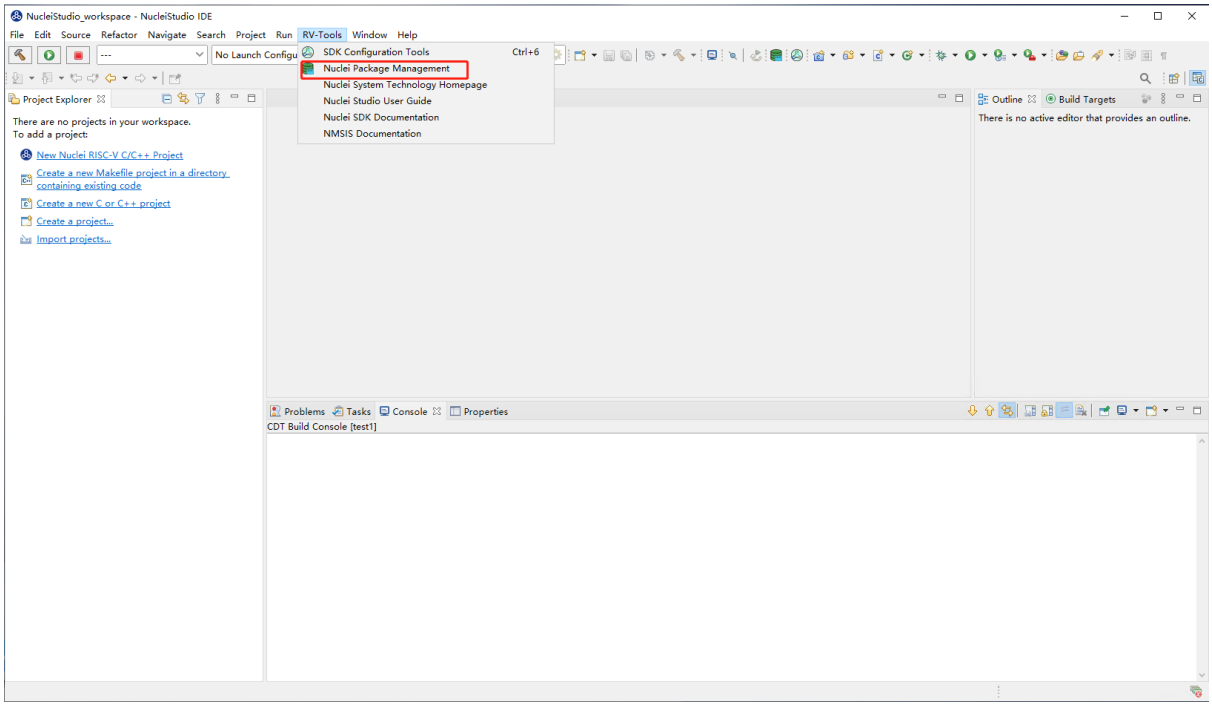
During the import, the Package Management page opens automatically. After the import is complete, the newly

imported package appears on the Package Management page of Nuclei Studio IDE.

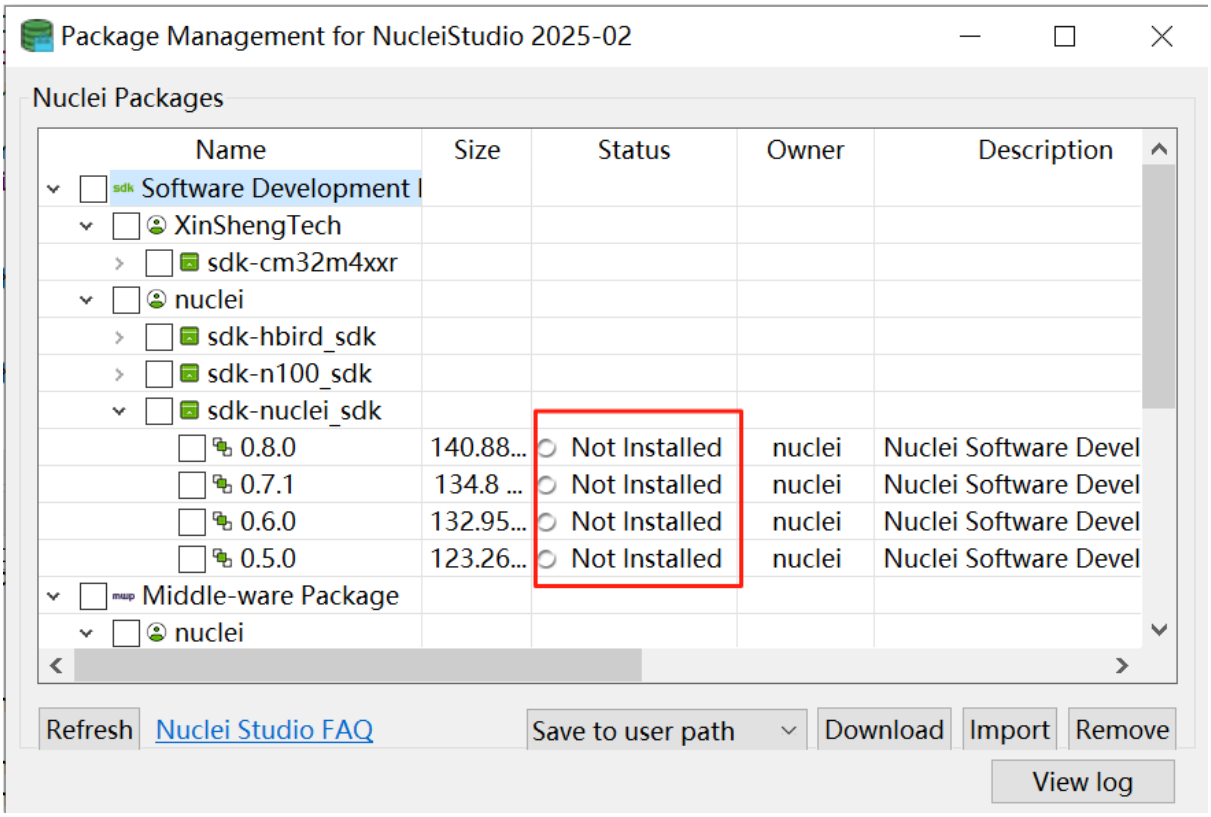


Download a Cloud NPK Package

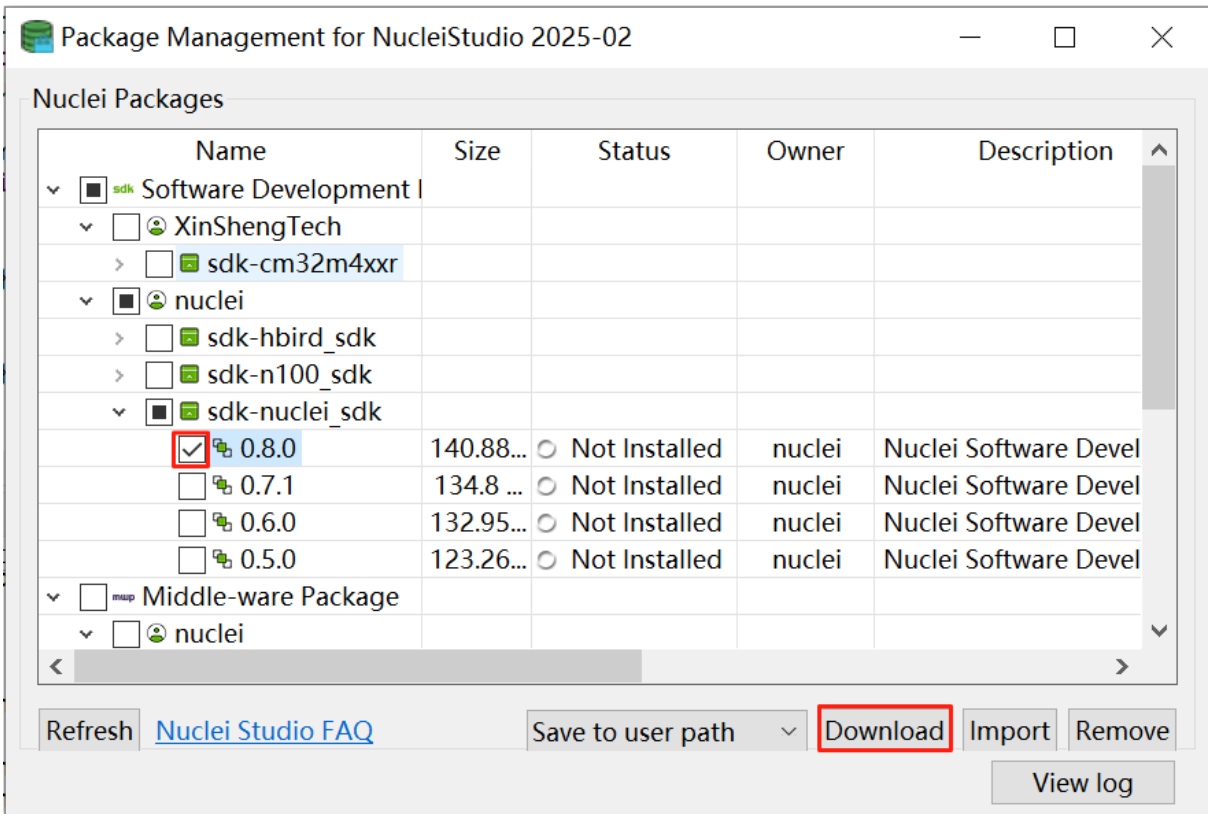
The biggest update in Nuclei Studio is moving NPK to the cloud. You can view all available NPKs directly in Nuclei Studio and install them yourself. Select `RV-Tools-->Nuclei Package Management` in the menu bar to manage NPKs on the **Nuclei Package Management** page.



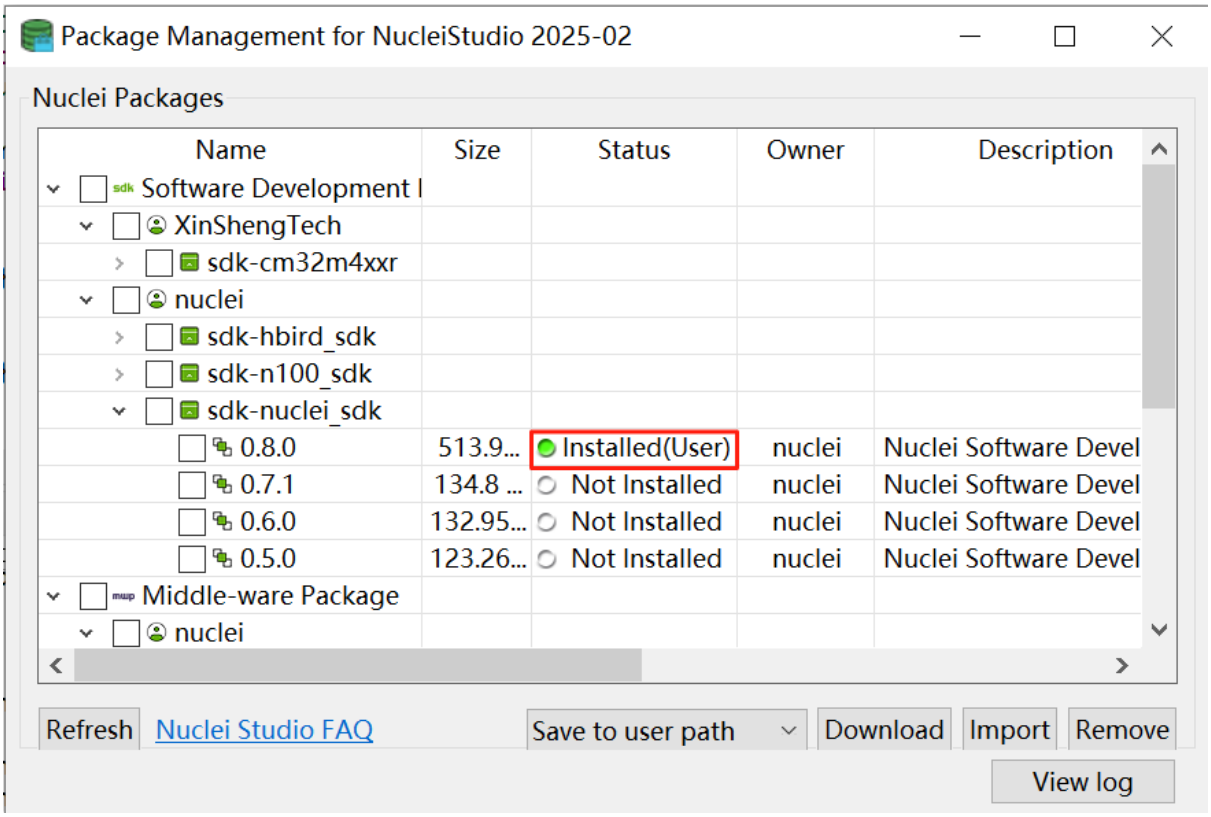
On the Nuclei Package Management page, click Refresh to get the latest NPK information. Before installation, the NPK status is Not Installed.



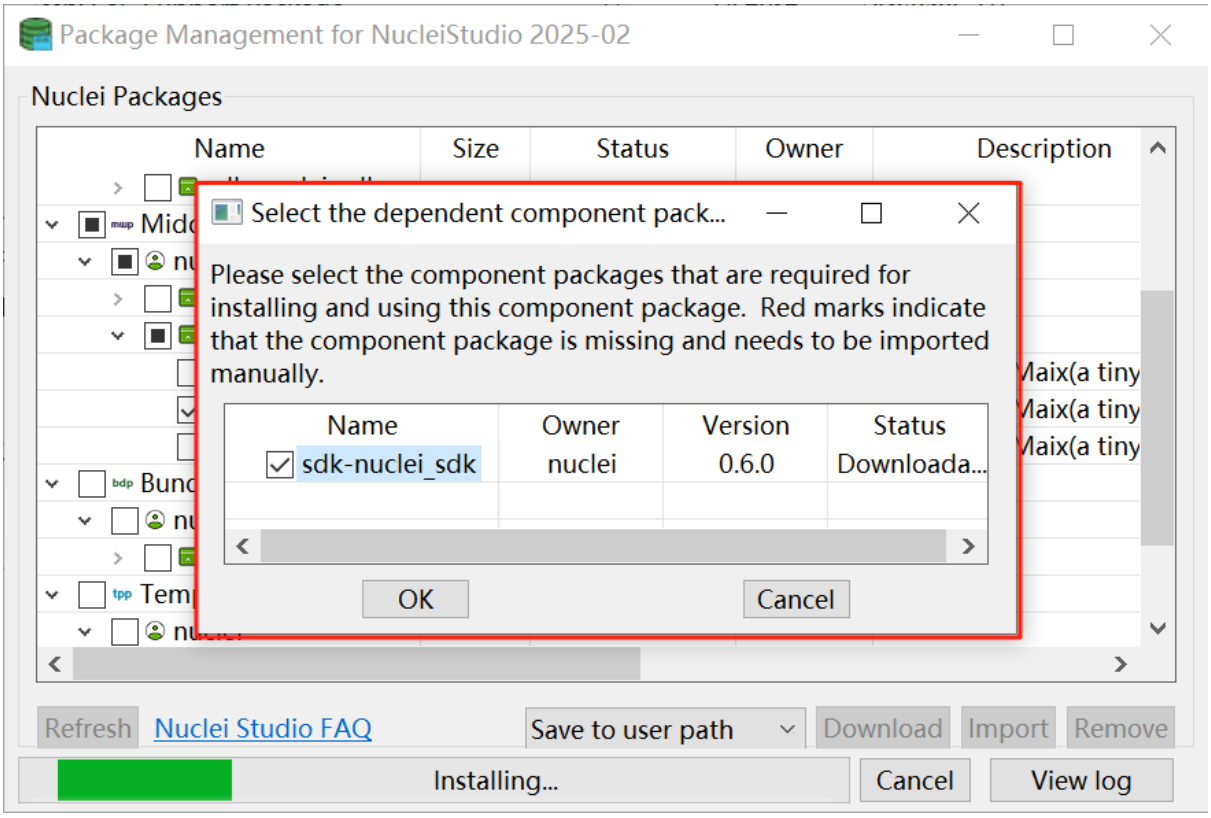
Select the NPK you want to install and click Download to install it. This example installs version 0.8.0 of sdk-nuclei_sdk. Use the latest version of Nuclei Studio for installation; the latest version supports the gcc14 toolchain, while previous versions supported gcc13/gcc10.



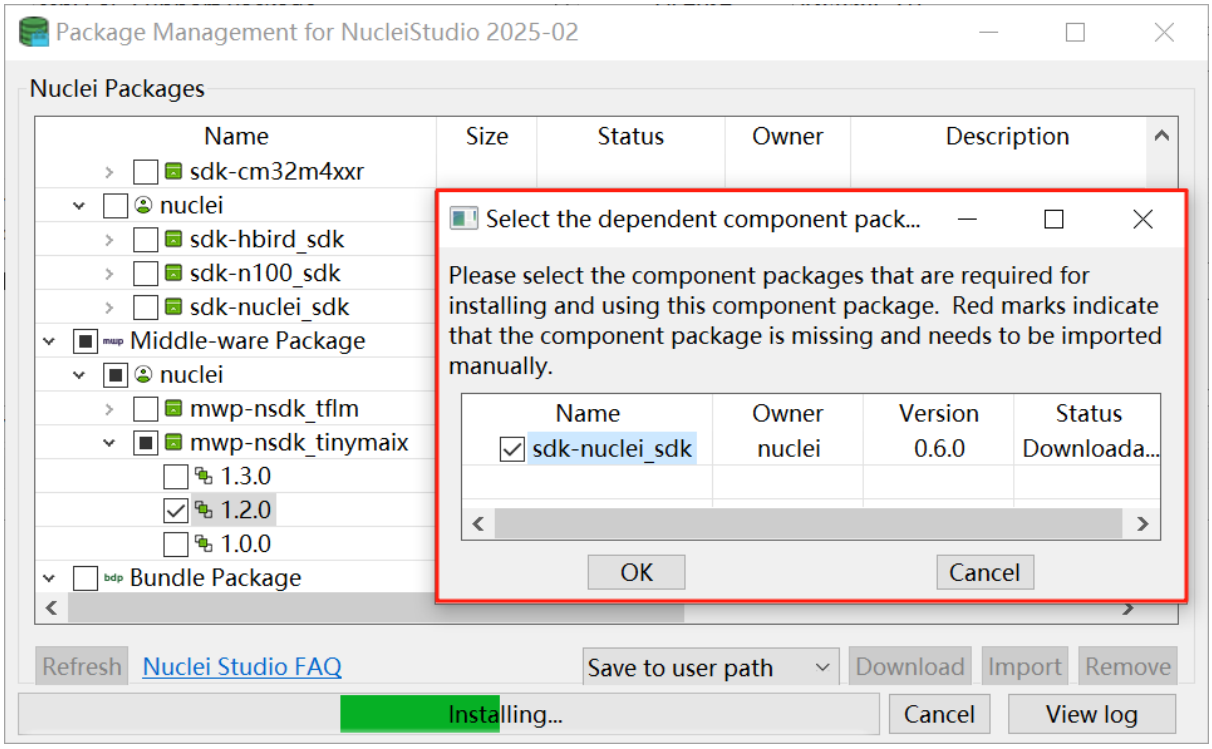
After installation, the NPK status changes to Installed.



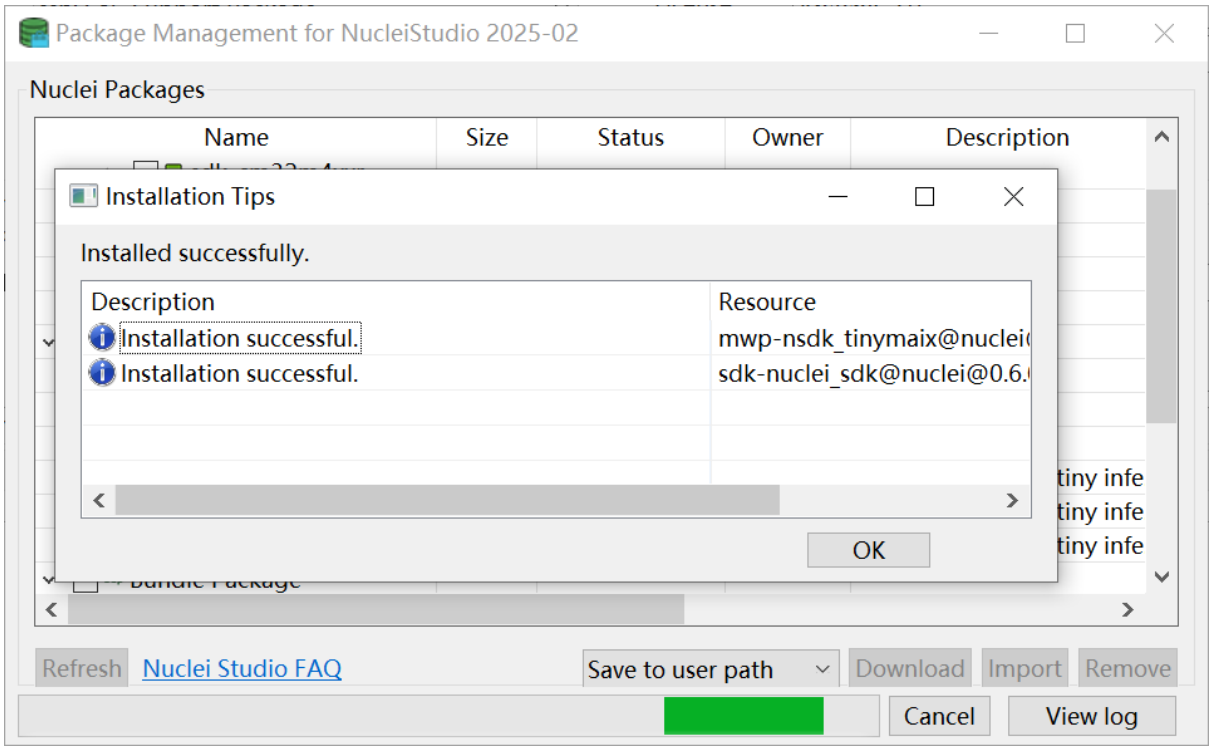
Nuclei Studio 2024.06 and later upgrades dependency management on the Nuclei Package Management page, greatly simplifying NPK package dependency handling.



When you download or install an NPK package, Nuclei Studio determines all of its dependencies based on the dependency relationships in the current NPK package information, combined with locally installed package information and NPK information shared in the cloud, and prompts you to download and install them together.



After the NPK packages are installed, Nuclei Studio reports how many packages were installed in total, the status of each package after installation, and the reason for any errors, making it easy to troubleshoot problems. This largely resolves the problem in previous versions where incorrect dependencies prevented NPK packages from being used.

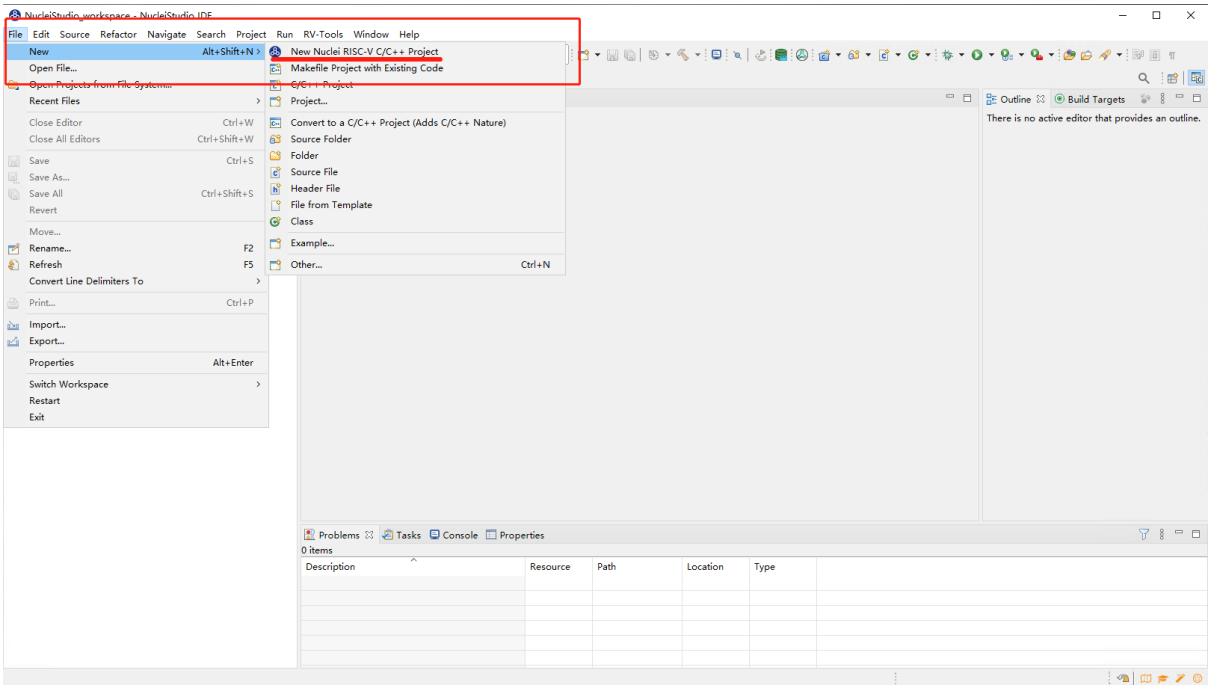


2.6.2 Create a Project via NPK

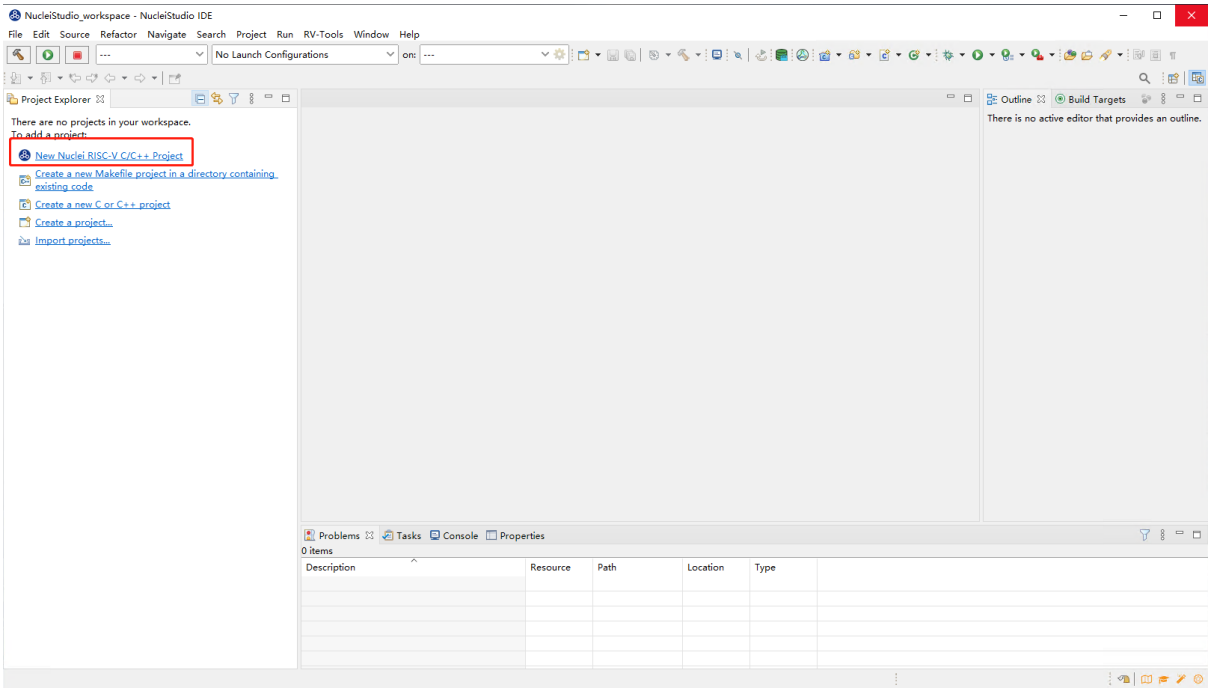
This chapter uses creating and modifying a GD32VF103 project for the RVSTAR board as an example to introduce Nuclei Studio features. Developing with the RVSTAR board requires the nuclei_sdk NPK package; for details, see the following chapters.

Create an NPK Example Project

To create a new project, select `File --> New --> New Nuclei RISC-V C/C++ Project` in the menu bar.



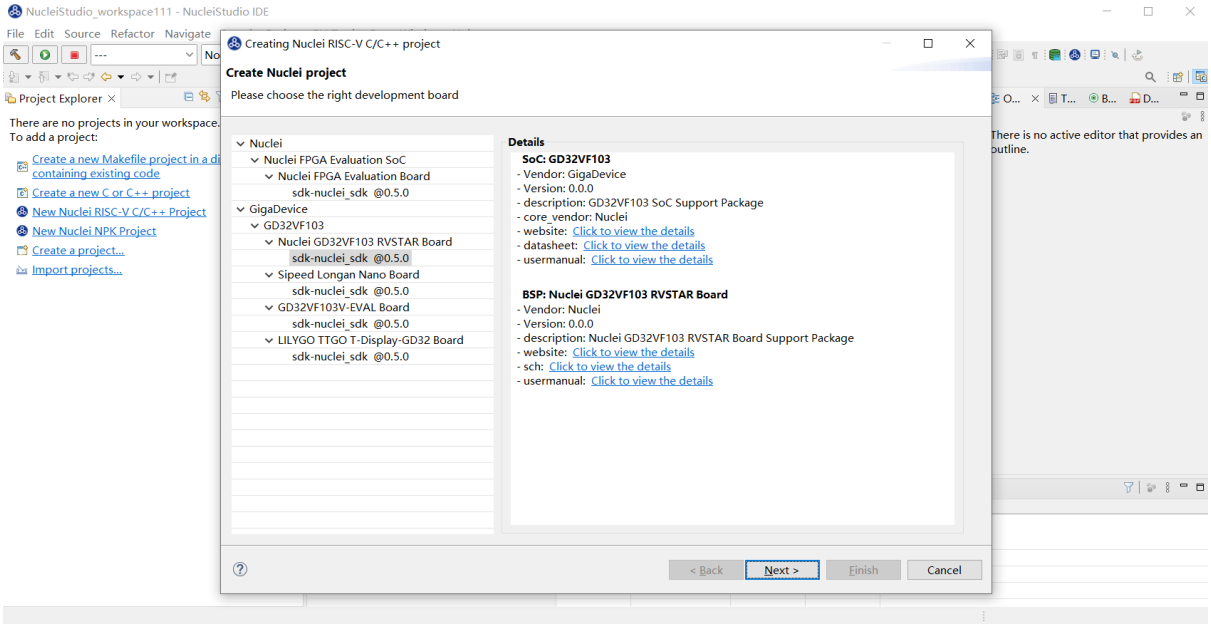
Alternatively, select `New Nuclei RISC-V C/C++ Project` in the Project Explorer view.



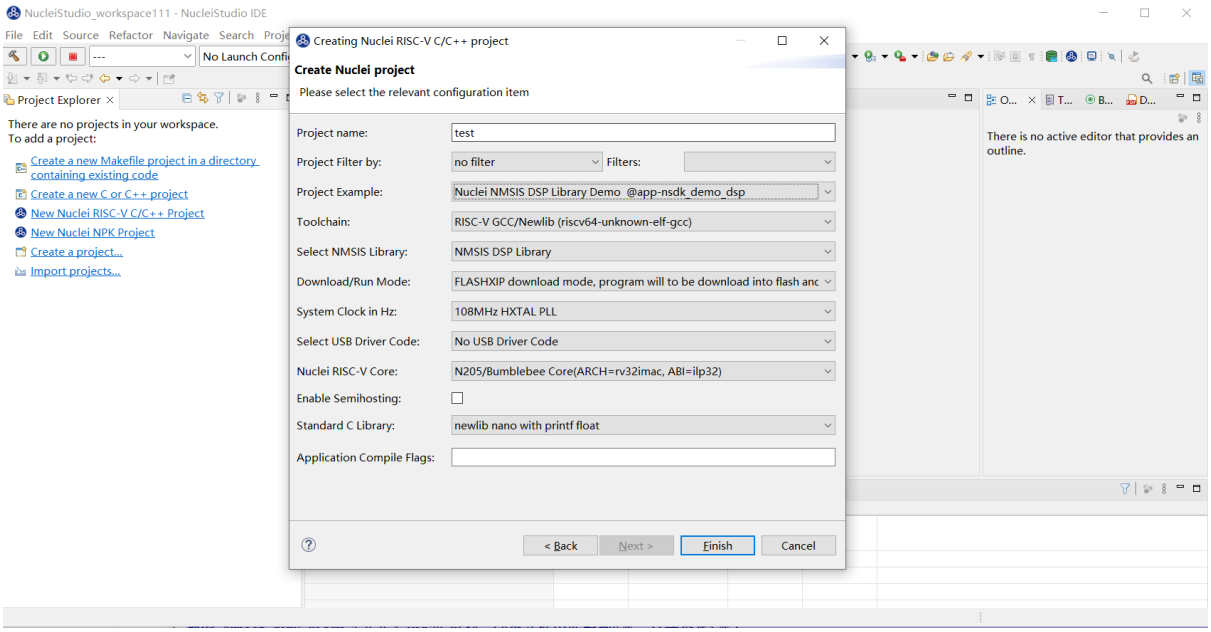
In the pop-up window, you can see different versions of SDKs provided by different vendors. Select an SDK

under a Board to view an introduction to the related SoC and Board. This example uses the RVSTAR board, so select `GigaDevice->GD32VF103->Nuclei GD32VF103 RVSTAR Board->sdk-nuclei_sdk@0.5.0`. Click `Next` to continue.

Note: **Note**: The SDK version number shown here changes as versions iterate and may also depend on a specific version of Nuclei Studio.



The project configuration page opens as shown in the figure. Because the RVSTAR core is fixed at N205 and its corresponding arch and abi are rv32imac and ilp32 respectively, the Core option cannot be modified. Similarly, the RVSTAR board supports only the FLASHXIP download mode, so the DOWNLOAD option cannot be modified either. Click `Finish` to complete project creation. Version 2023.10 adds support for Arm projects.



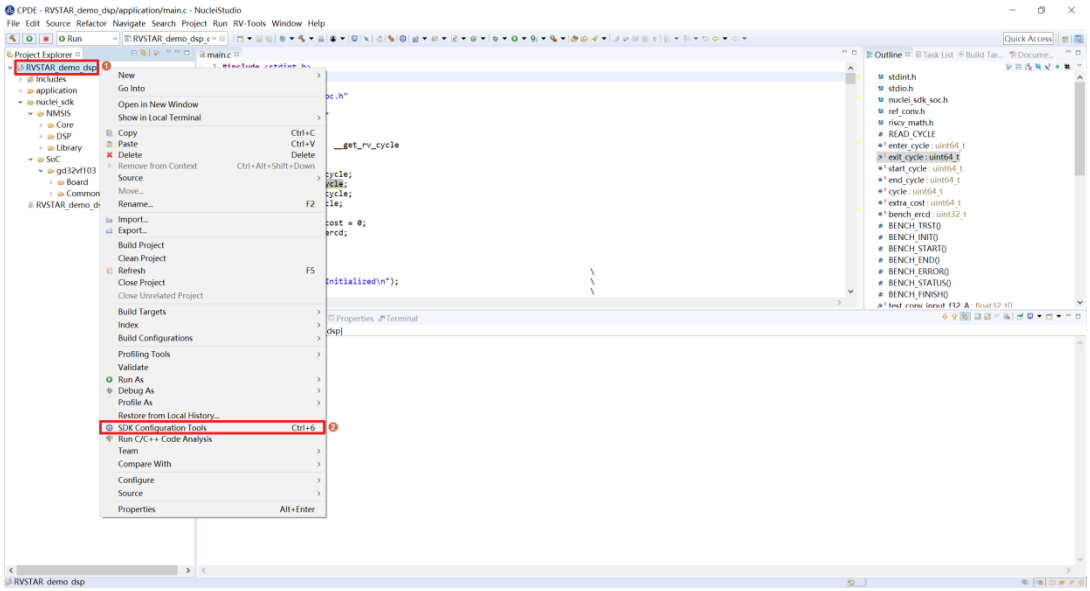
Nuclei Studio adds different SDK source code depending on the selected project template. For example, the FreeRTOS template project adds the corresponding OS content, and the Demo_dsp template project adds NMSIS library files. For detailed information about NMSIS, refer to (https://doc.nucleisys.com/nmsis/index.html). This example uses Demo_dsp: select `Nuclei NMSIS DSP Library Demo` for `Project Example`. Because the DSP project requires the NMSIS library, select `NMSIS DSP Library` for `Libraries`.

Nuclei Studio automatically sets project options based on the options selected during project creation. This example uses floating-point printing, so select `newlib nano with printf float` for `NEWLIB`. Then keep clicking `Next` until `Finish`.

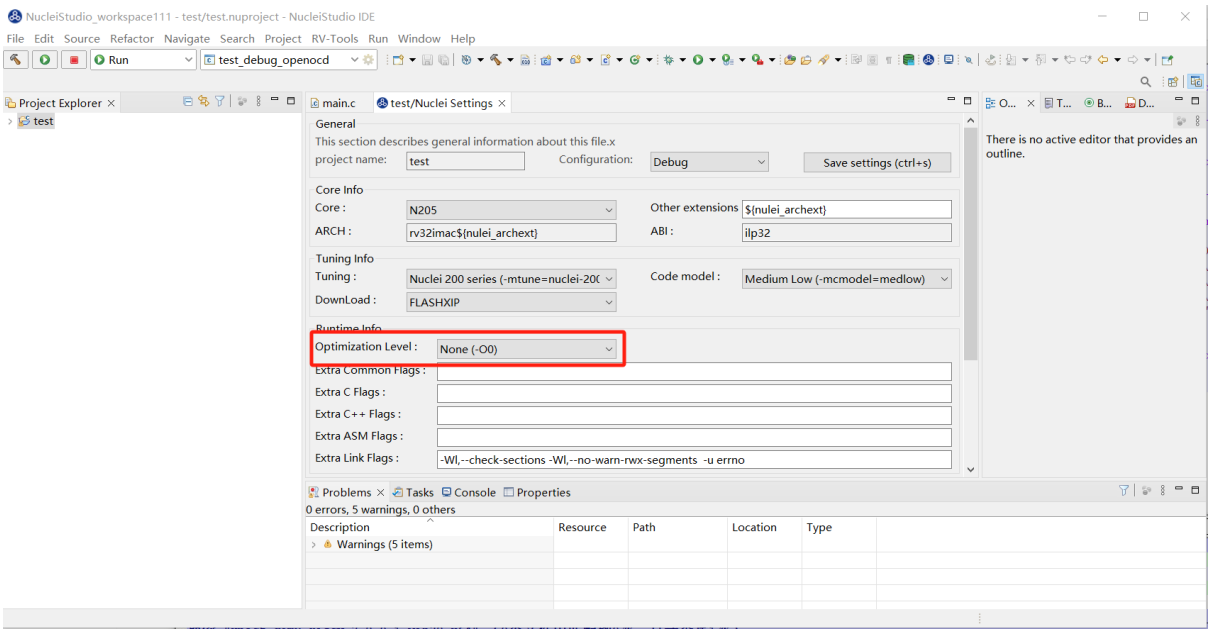
Change Project Configuration with SDK Configuration Tools

In Nuclei Studio, you can quickly modify project settings using the `SDK Configuration Tools` tool. Starting with Nuclei Studio IDE 2022.12, the `SDK Configuration Tools` tool was refactored into the Nuclei Settings menu, which provides a better user experience.

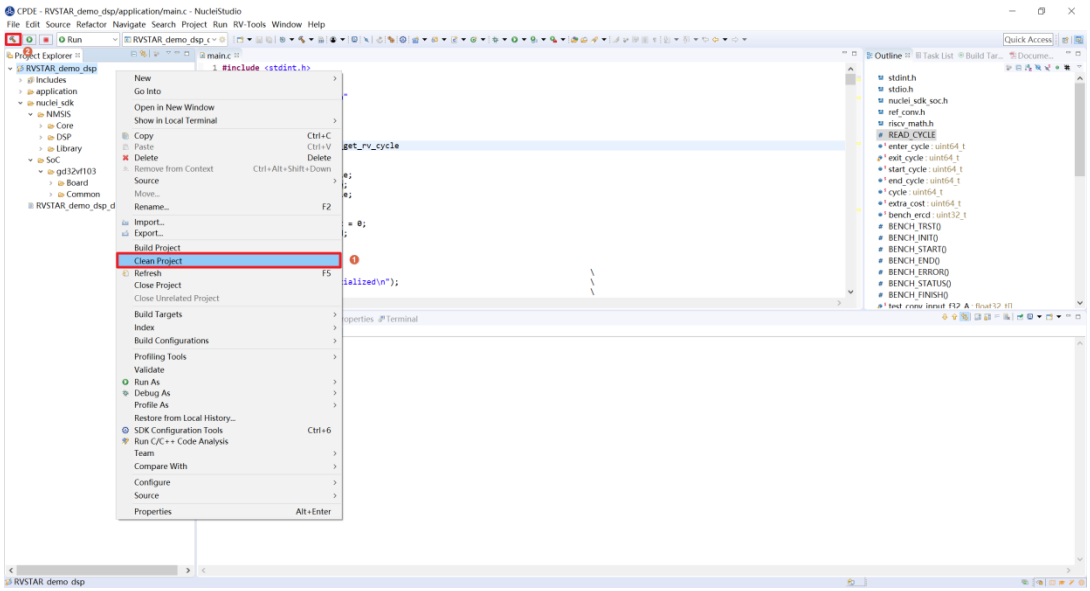
For a newly created project, click the project name, right-click to open the context menu, and select `SDK Configuration Tools` to open the settings tool.



To change the compile optimization level, set `Optimization Level` to `None (-O0)` and click `Save` to apply the change.



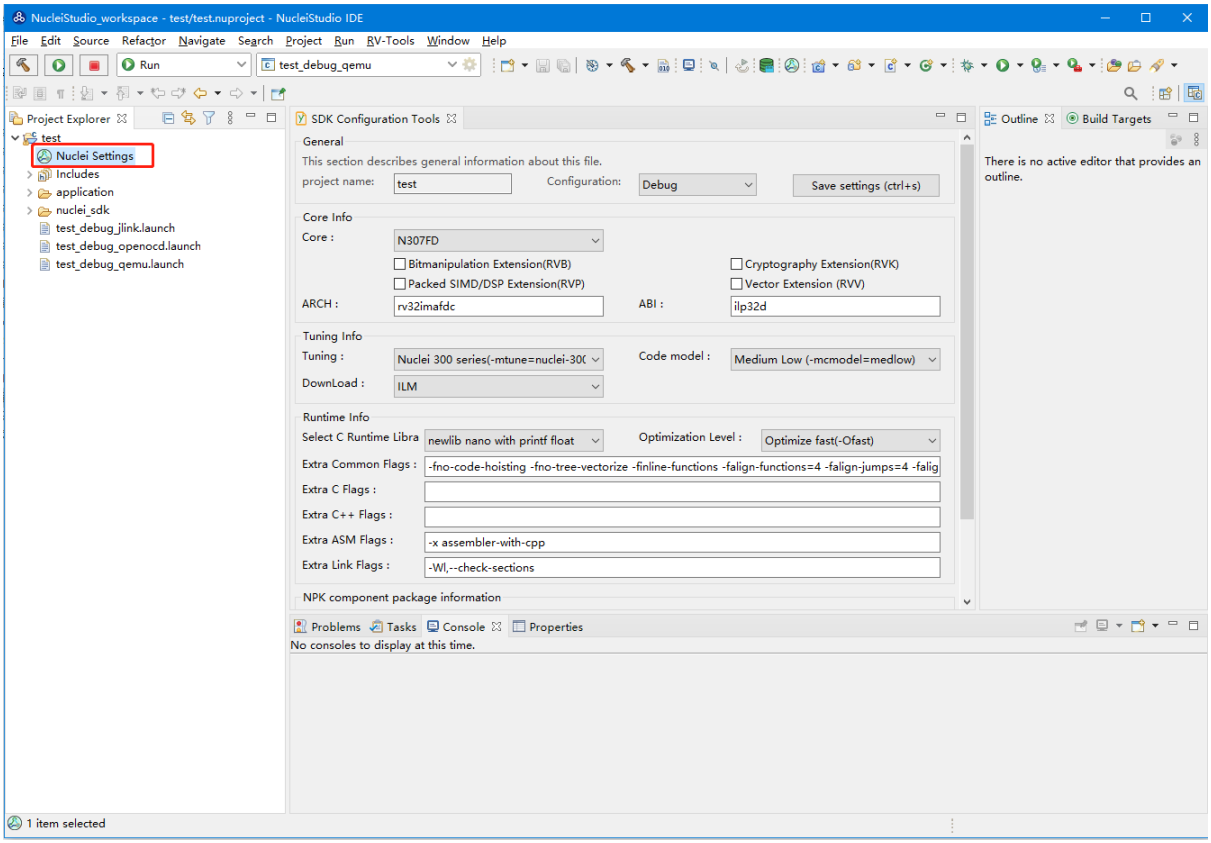
After the modification succeeds, right-click the project, select `clean` in the context menu to clean the project, and then click the hammer icon to build the project.



Note:

- **Note:** After SDK Configuration Tools modifies the compile configuration, the debug configuration (Debug Configurations) is not affected; modify the corresponding debug configuration manually.
- **Note:** In later versions, the `SDK Configuration Tools` feature is no longer maintained and has been replaced by the Nuclei Settings menu feature.

For a better user experience, Nuclei Studio IDE 2022.12 refactored `SDK Configuration Tools`. Newly created projects have an additional Nuclei Settings menu. Double-clicking the Nuclei Settings menu opens the project configuration tool, which is functionally identical to `SDK Configuration Tools`. In version 2023.10 and later, SDK Configuration Tools opens this Nuclei Settings page directly.



2.6.3 Import Tools via NPK

In addition to SDKs, NPK packages can also import various tool packages to extend the capabilities of Nuclei Studio. Nuclei Studio 2022.08 added support for NPK Tools. To improve the extensibility of component packages and make compilation and debugging more convenient, an NPK component package of type tool was introduced. A tool component package can contain gcc, qemu, cmlink-gdb, and other content, and is imported into the IDE as a zip package.

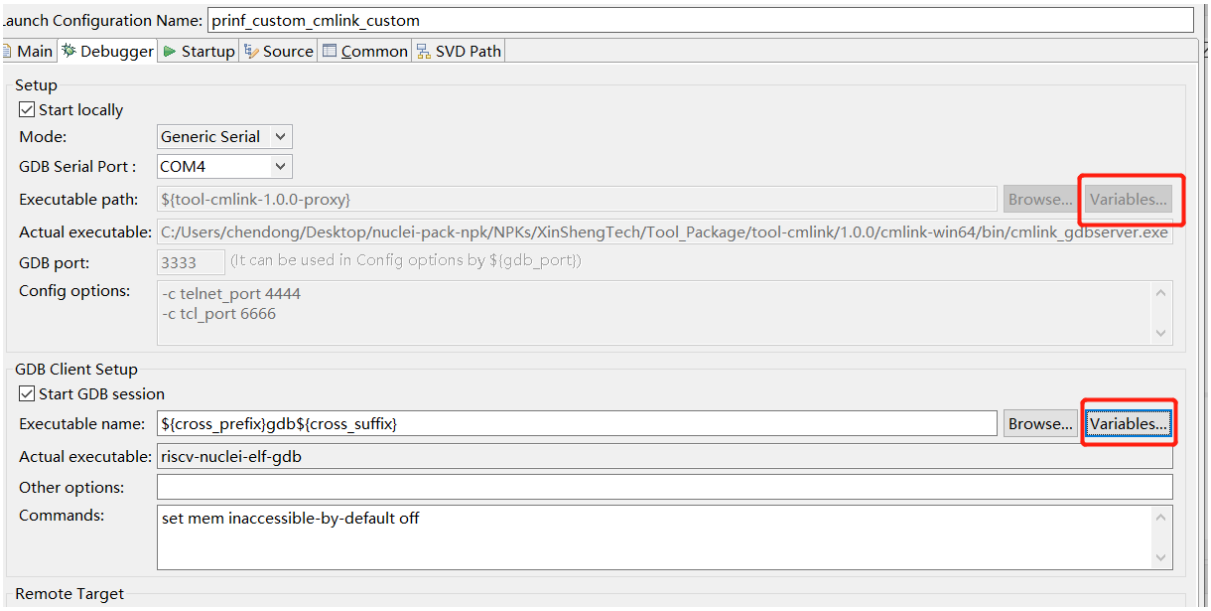
Take the tool-cmlink package as an example. A tool package contains the tool ' s executable files and an npk.yml file. In the npk.yml file, the developer provides a brief description of the tool, such as the tool package ' s developer, version, supported operating systems, and the path of the executable files. The package structure and npk.yml content are shown in the example below. Compress the tool package into a zip file, then refer to *Import a Local NPK Package* (page 86) to import the NPK tools into the IDE, or share them on the www.rvmcu.com[7] website.

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

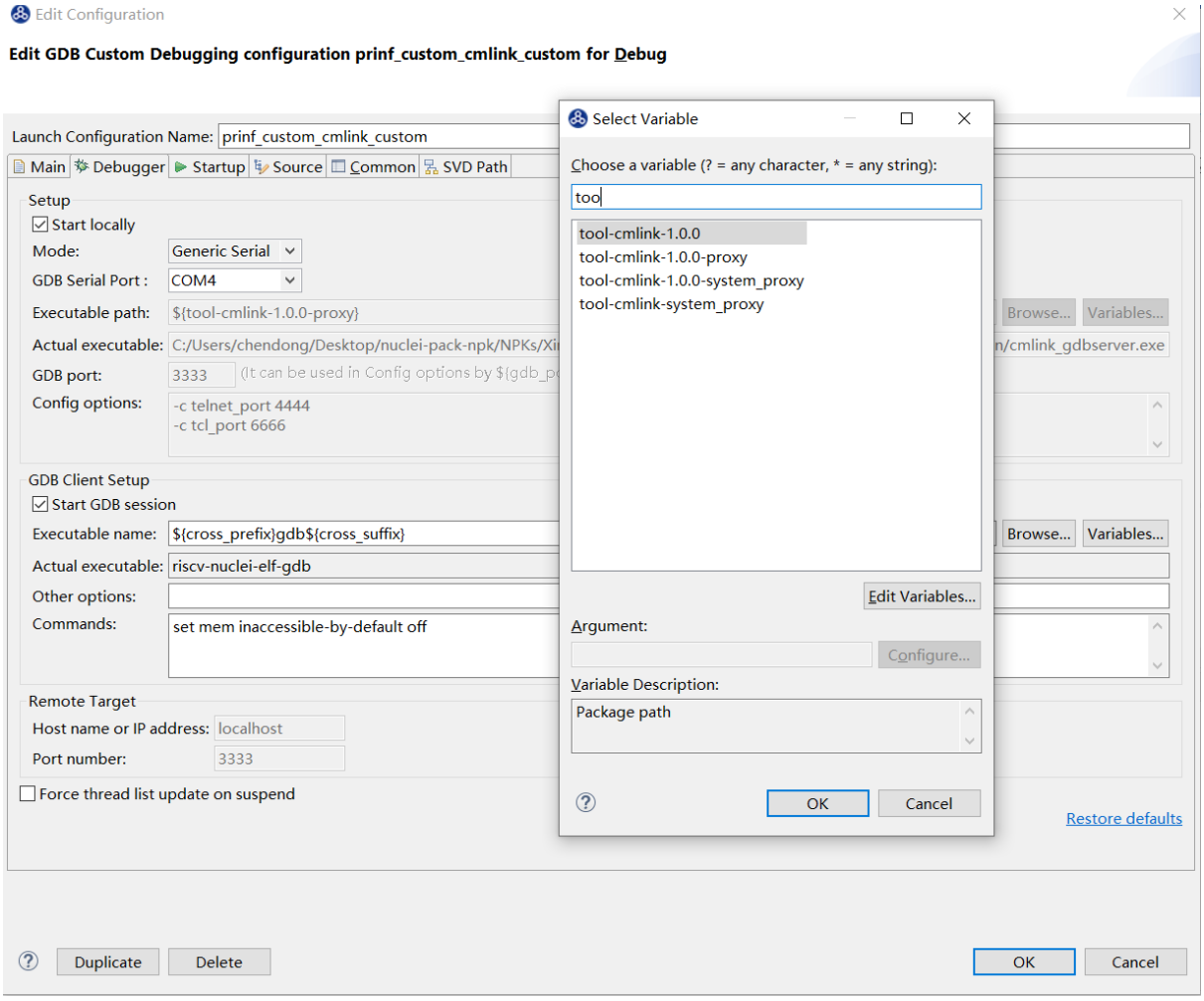


NPK tools can also be managed on the Nuclei Package Management page. After downloading the component package, open any debug page and click Variables to view the parameters corresponding to the NPK tools; selecting the corresponding parameter lets you use the tool.

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



Generally, four extended variables are defined for the component package in an NPK tool. Each package has a package path, referenced as npk name-version number, for example `${tool-cmlink-1.0.0}`. Other variables are referenced as npk name-version number-variable name, for example `${tool-cmlink-1.0.0-proxy}` and `${tool-cmlink-1.0.0-system_proxy}`. When the system value of a variable is true, an additional variable without a version number is added, which resolves to the highest version of that variable, for example `${tool-cmlink-system_proxy}`.



2.7 Create a Project in Nuclei Studio

This chapter uses the Nuclei FPGA Evaluation Board with the N300FD (rv32imafdc) processor core as an example to describe the common ways to create a project in Nuclei Studio.

There are several common ways to create a project in Nuclei Studio IDE:

Automatically create a project using a Nuclei Package (NPK) template project:

This is the simplest and quickest way. The template project feature currently relies on the NPK feature of Nuclei Studio. After importing the corresponding SDK NPK Zip package, you can create template projects in Nuclei Studio. Nuclei System Technology provides NPK Zip packages for Nuclei SDK, HBird SDK, and SoC IP SDK, all of which can be imported into the IDE to create and use projects.

Import an existing project directly to create a new project:

This is the most common way. For example, user A can package the folder of an existing project for sharing, and user B can import the project directly on another computer, then use it as-is or continue developing and modifying it on that basis.

Manually create a project without a template:

This is the most tedious way: in addition to creating the project, you must set various options and paths manually. It is rarely used in practice, but the detailed walkthrough of this method teaches you how to configure the various options and paths.

Create a project based on an existing Makefile:

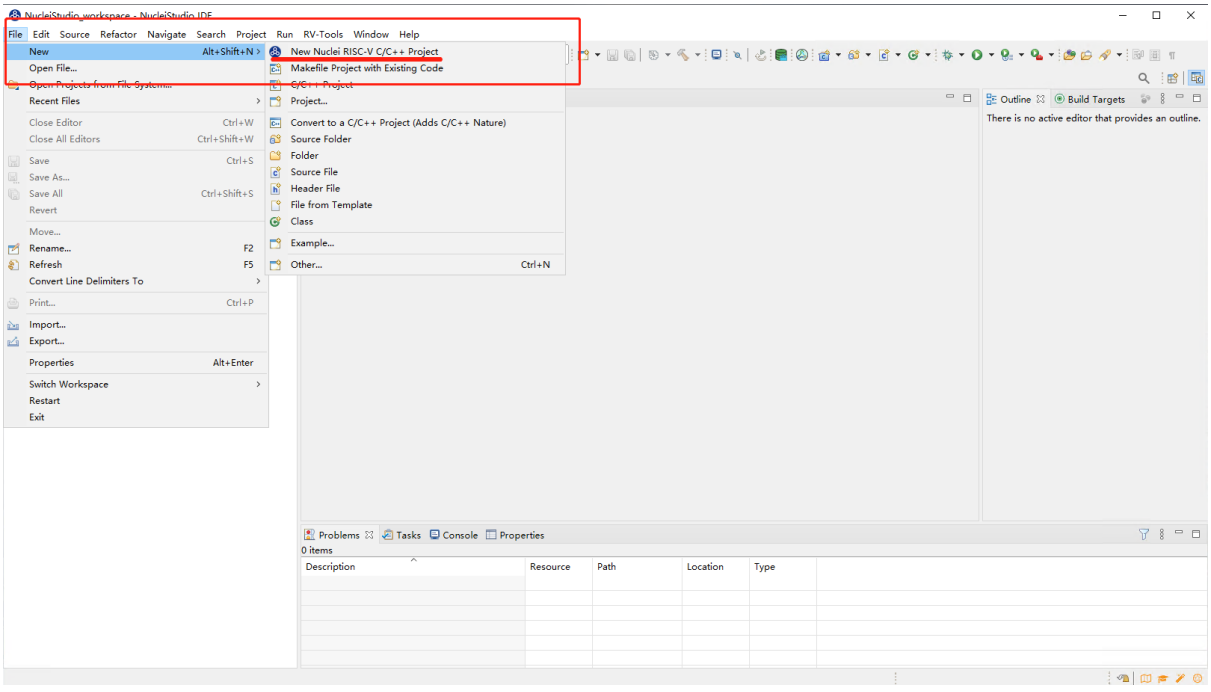
This method is suitable for projects that already use a Makefile or other build tools. It lets you build, clean, and debug the project in the IDE —a good debugging environment —without modifying the build system.

The following sections introduce each of these methods.

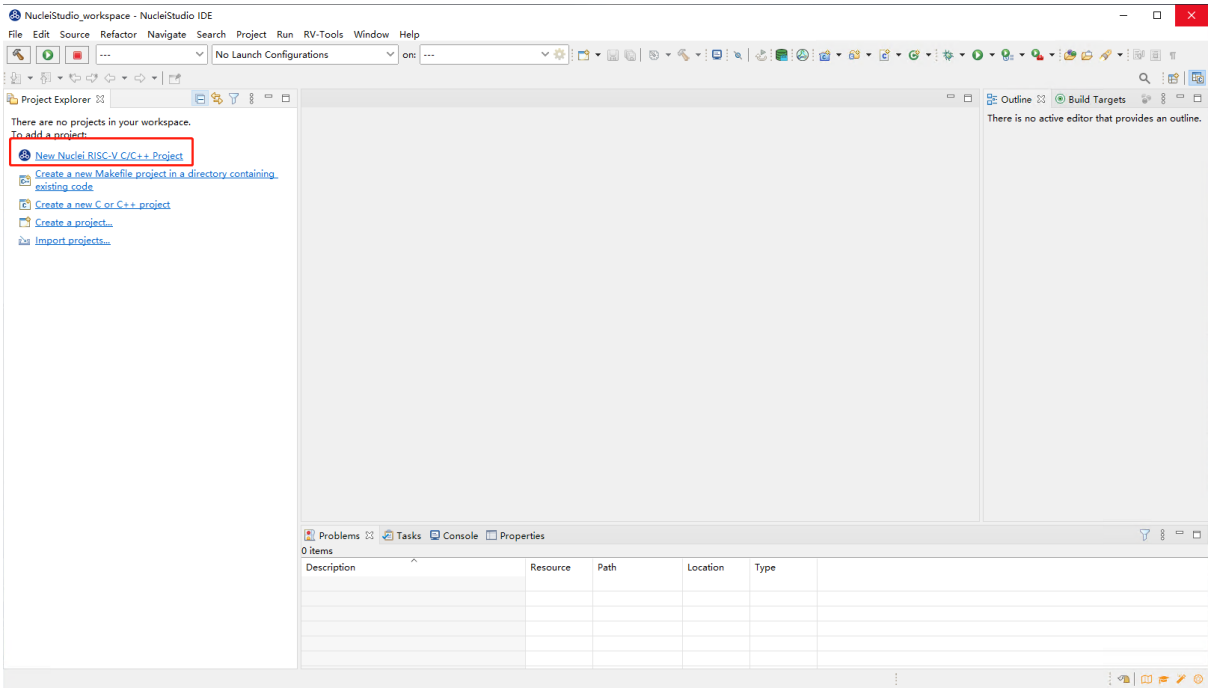
2.7.1 Automatically Create a Project Using an NPK Template Project

This section describes how to use a template to automatically create a simple Hello World project in Nuclei Studio IDE.

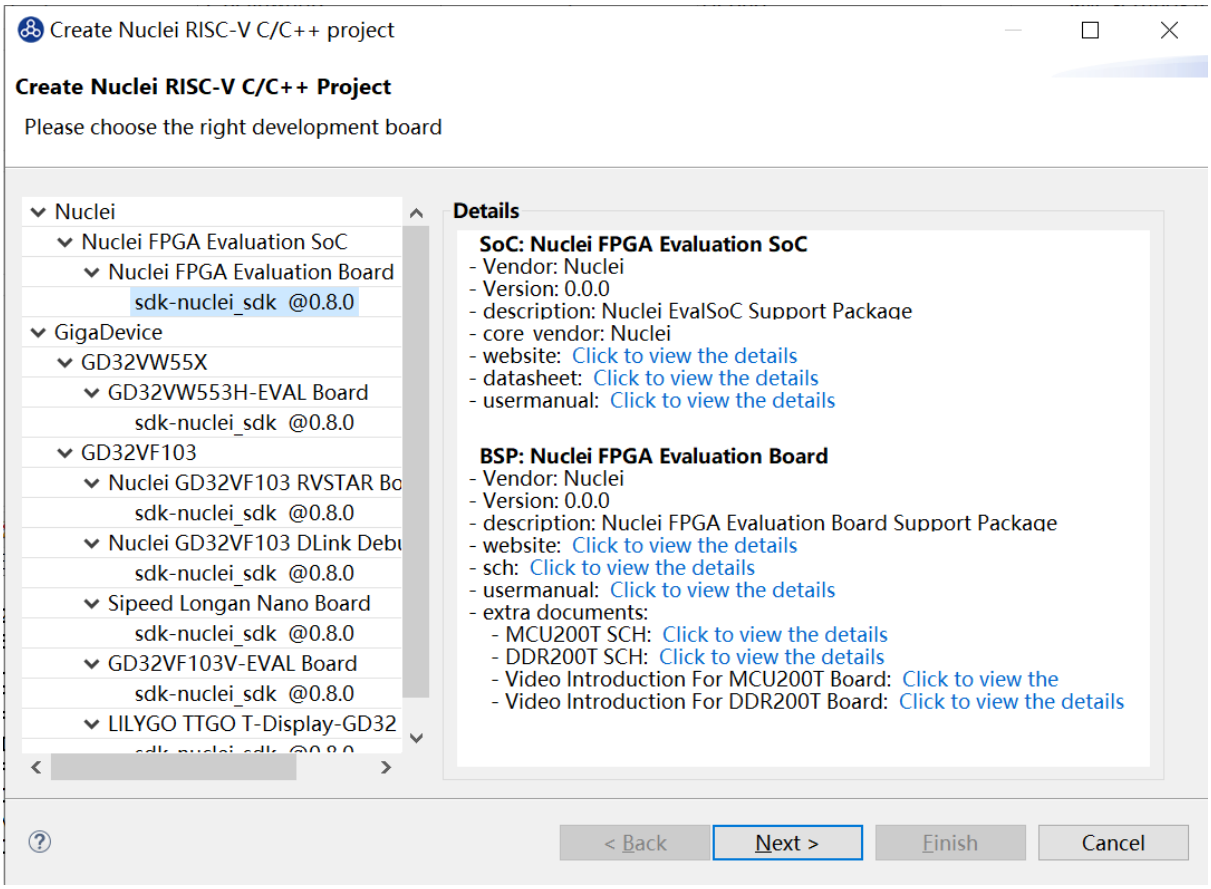
To create a new project, select `File -> New -> New Nuclei RISC-V C/C++ Project` in the menu bar.



You can also select `New Nuclei RISC-V C/C++ Project` in the Project Explorer view.



Select the project type in the pop-up window. Here we use a Nuclei FPGA board with the N300FD core and the nuclei_sdk@0.8.0 SDK for test development. Select the SDK under the corresponding Board, and click `Next` to proceed to the next step.



Set the following parameters in the pop-up window.

- Project name: the project name. Here it is set to `1_helloworld`

- Project Example: select `Simple Helloworld Demo`.
- Toolchains: we use `RISC-V GCC/Newlib`.

Note: This page is automatically parsed and generated from the NPK Configuration field. Different SDKs, different boards, or different examples may produce different option pages.

Our core is N300FD, so select `N300FD` for `Core`.

The HummingBird board supports three download modes, briefly introduced below; here we select the ILM mode.

- **ILM**

In ILM download mode, the program is downloaded directly into the MCU ' s ILM and executed from ILM. ILM consists of SRAM, so its contents are lost when power is off.

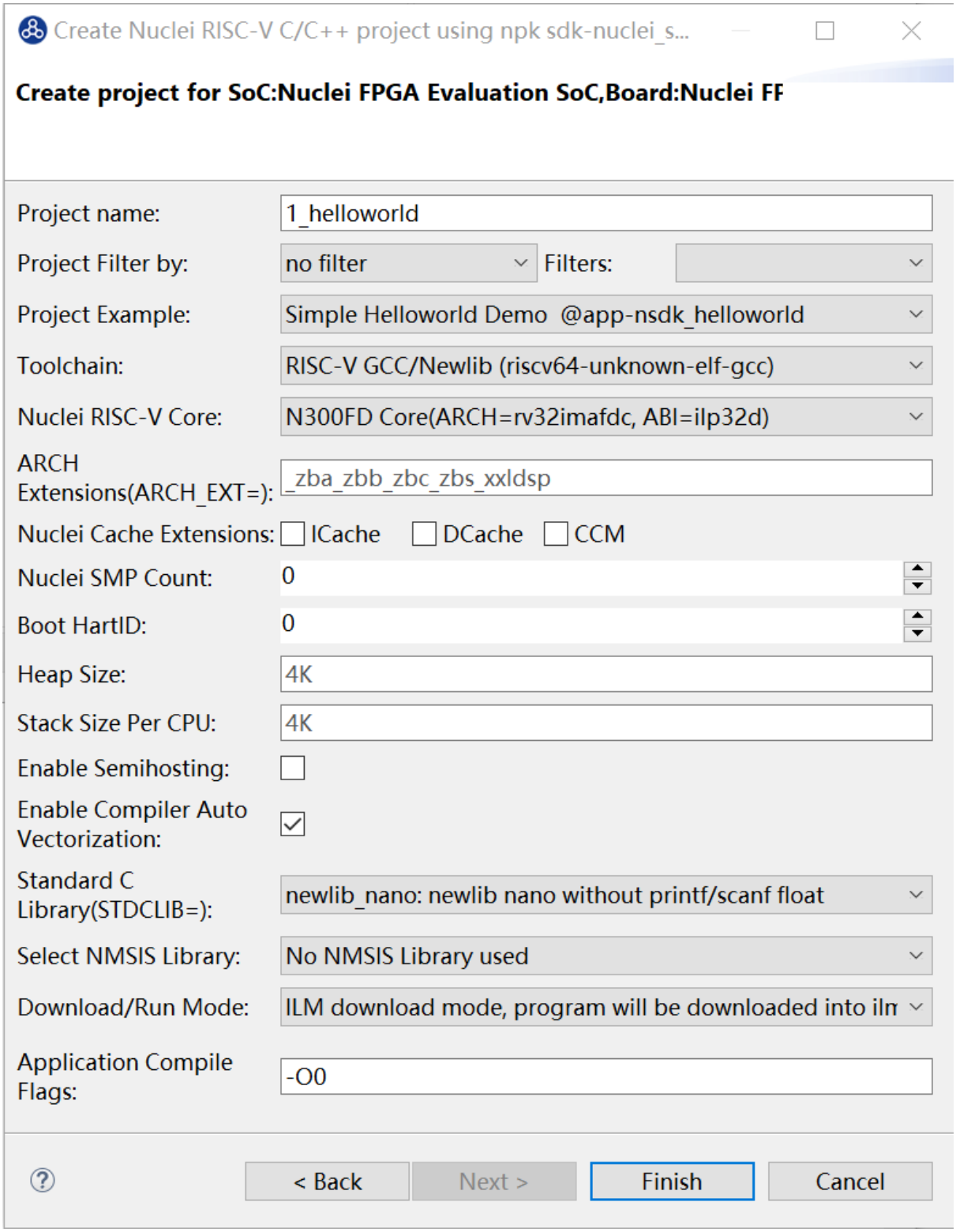
- **FLASH**

In FLASH download mode, the physical address of the program ' s code section is constrained to the Flash region, while the logical address of the code section is constrained to the ILM address region. This means the program is downloaded directly into the MCU ' s Flash, but after power-on a boot program moves the code section into ILM, and execution starts from ILM. The program is flashed into Flash, so it is not lost when power is off.

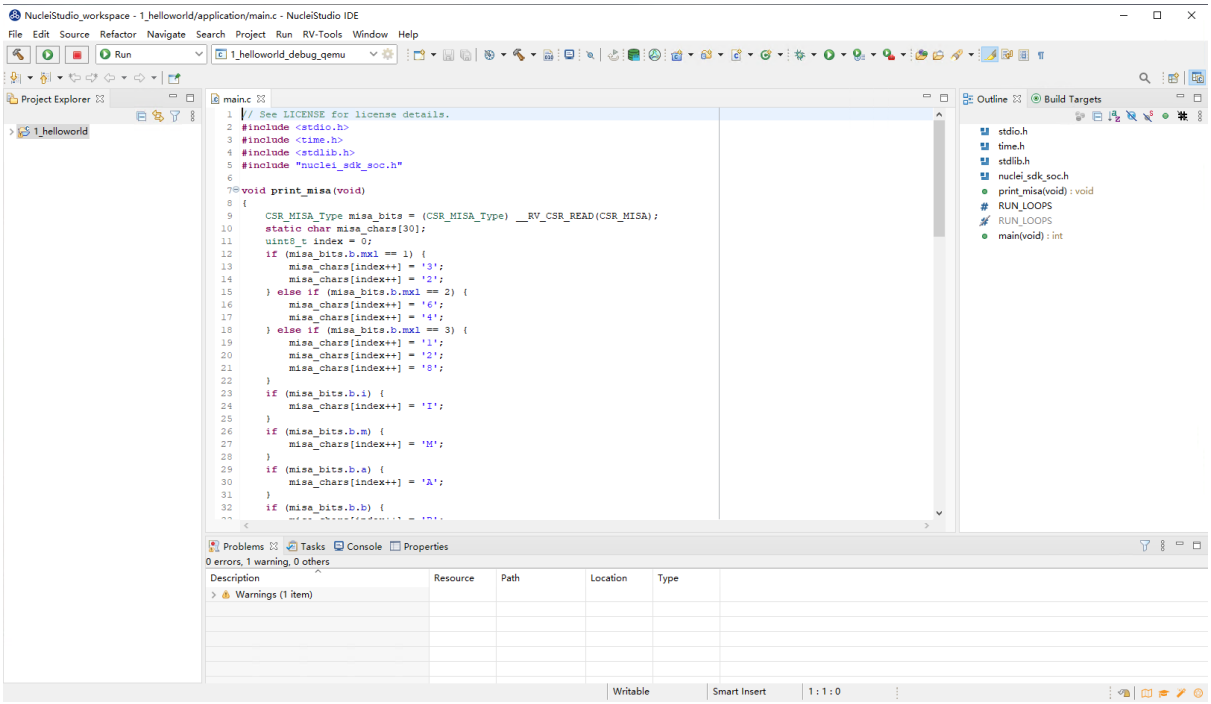
- **FLASHXIP**

In FLASHXIP download mode, the program ' s code section is constrained to the Flash region, which means the program is downloaded directly into the MCU ' s Flash and executed directly from Flash. The program is flashed into Flash, so it is not lost when power is off.

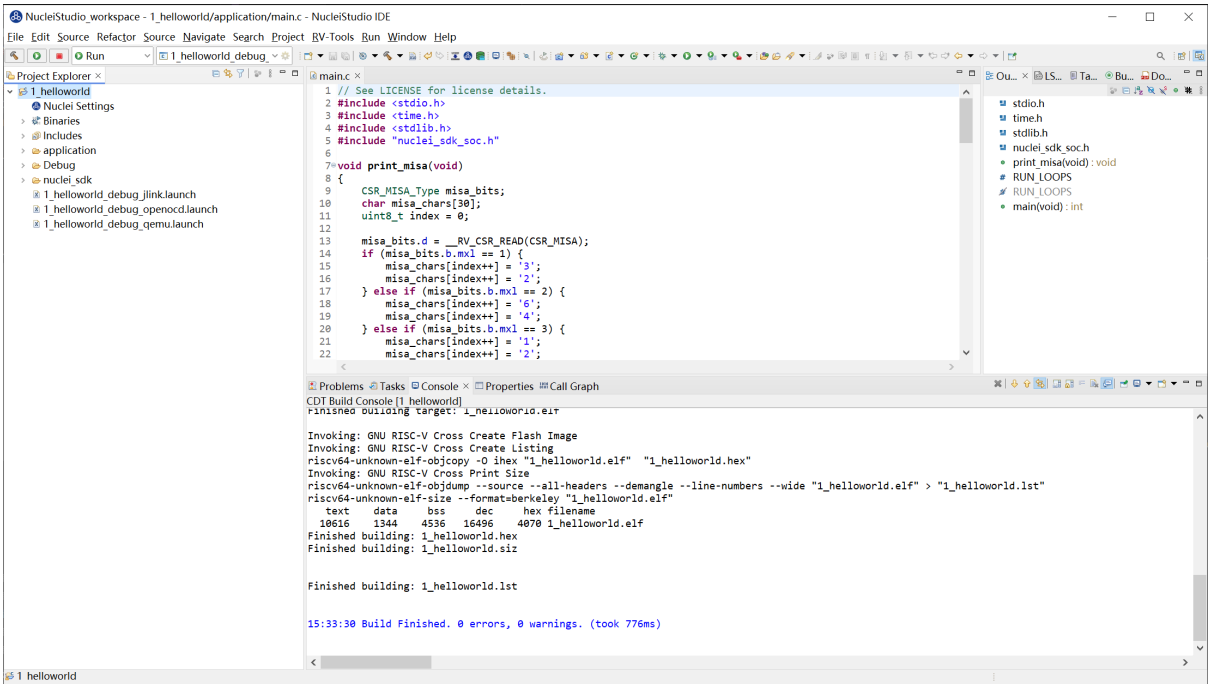
Configure the other options as needed, then click `Finish` to complete project creation.



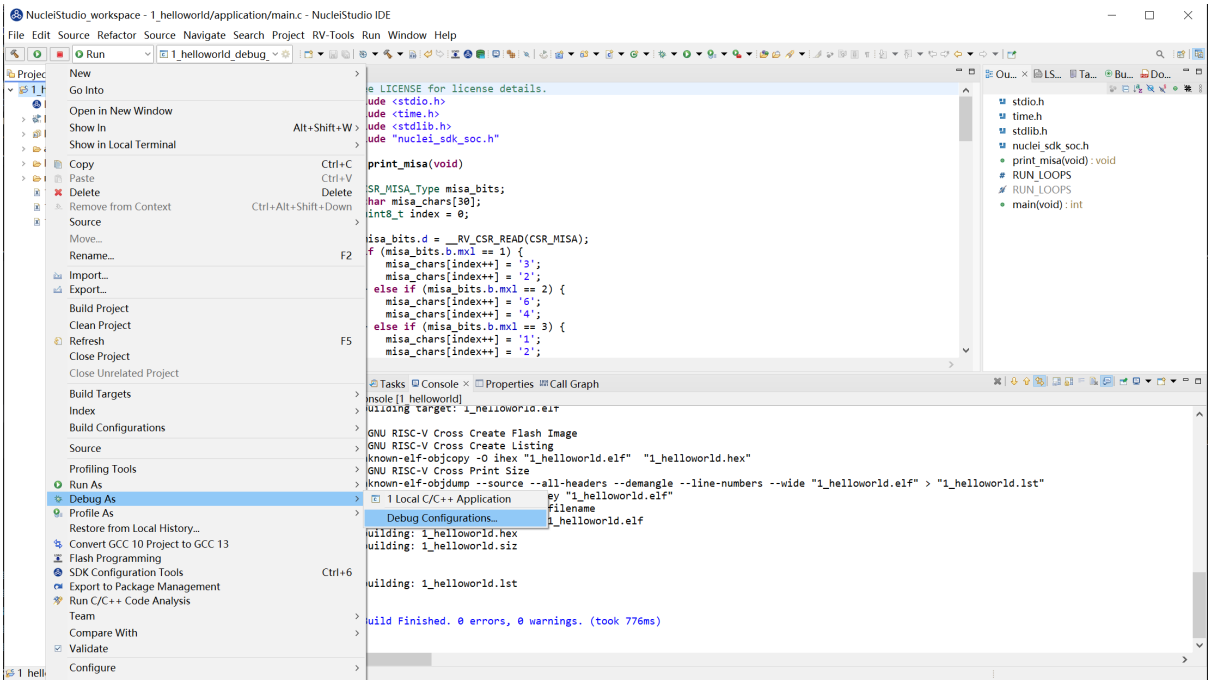
The Hello World project has now been created automatically using the template.



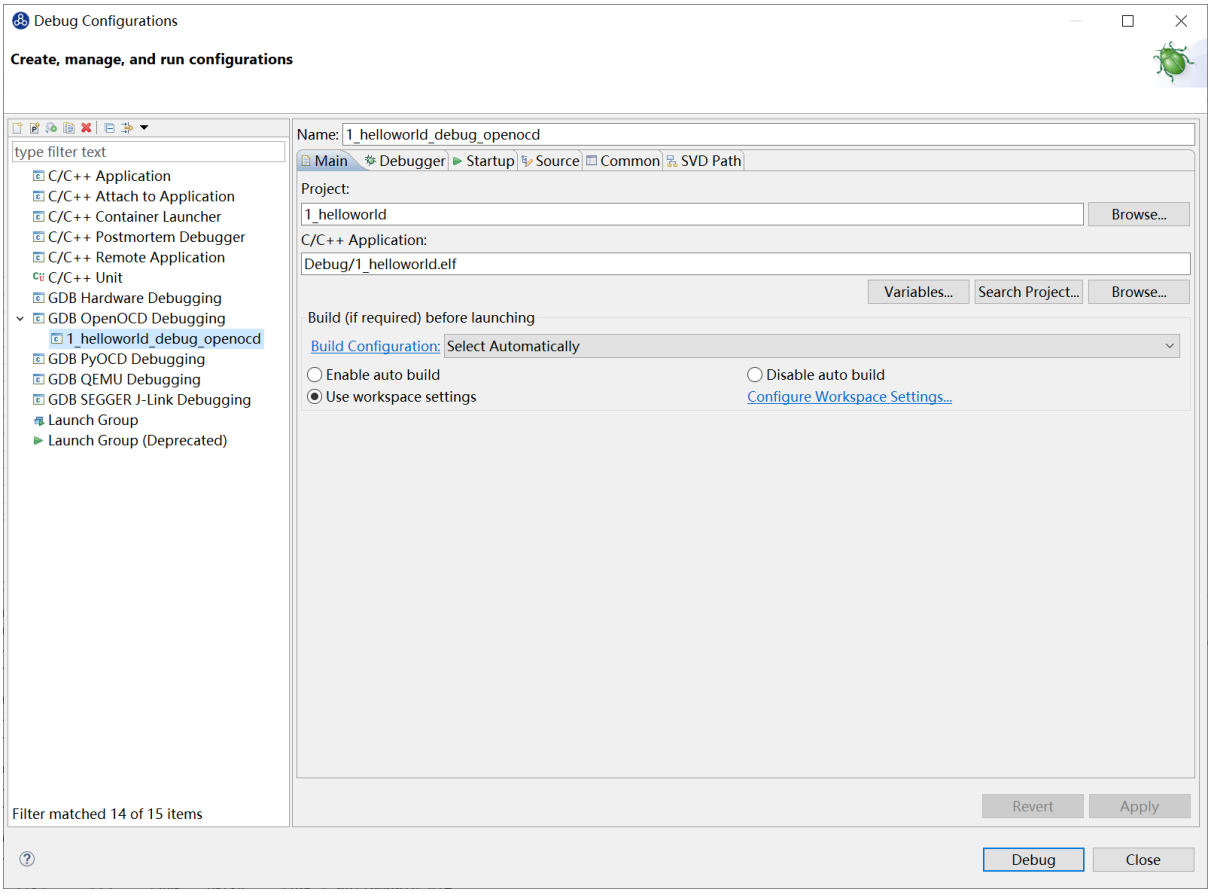
Build the project directly using the menu bar `Project -> Build Project` or the  button.



During the automatic generation of the Hello World project, the corresponding OpenOCD configuration is synchronized. After the project is built, right-click the project `Hello World` in the project list and click `Debug As -> Debug Configurations` to open the debug configuration panel. Debug and Run use the same configuration files, so they can also be opened via `Run As -> Run Configurations`.



The configuration file `Hello World_Debug_OpenOCD` used for debugging is generated automatically. For details on how to use the Nuclei HummingBird debugger together with OpenOCD to download and debug, see *Debugging and Running a Project with the HummingBird Debugger and OpenOCD* (page 150).

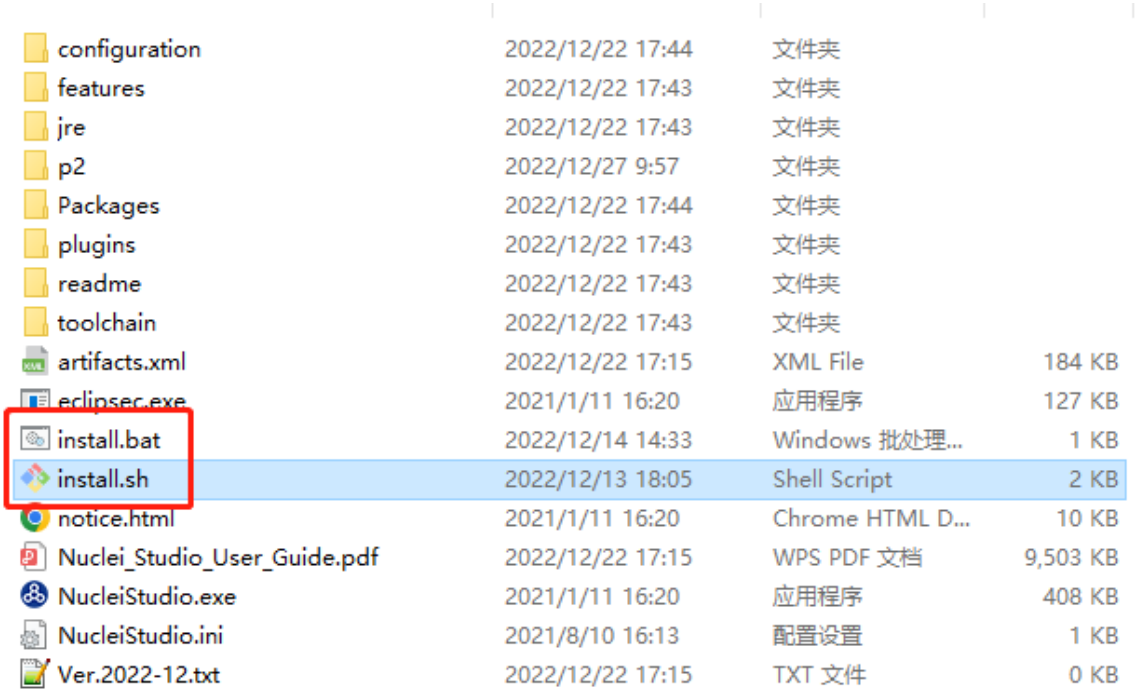


2.7.2 Import a Project via an Application Association File

This section describes how to import a Hello World project into the IDE through the association file of Nuclei Studio IDE. Nuclei Studio IDE 2022.12 added support for application file association. The IDE launch path can be registered in the system registry, after which specific files can be used to launch the IDE, import projects, and so on, which greatly simplifies sharing and quickly importing projects for Nuclei Studio IDE users.

Register Nuclei Studio IDE in the Registry

In Nuclei Studio IDE 2022.12 and later versions, the installation package includes two additional files, `install.bat/install.sh`. On Windows, double-click `install.bat`. Because this needs to write to the registry, user authorization is required; after authorization, the installation succeeds. On Linux, run the `install.sh` file in a shell.



After the `install.sh` file runs, a user authorization screen appears; agree to the authorization.



Import a Project via an Application Association File

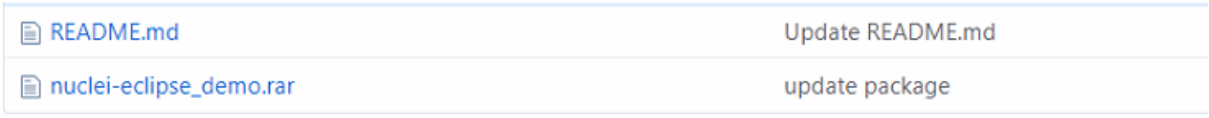
Use Nuclei Studio IDE 2022.12 or later to create a project `test`. The project contains an application association file, `test.nuproject`. If the IDE launch path has been written to the registry, double-clicking the `test.nuproject` file automatically starts Nuclei Studio IDE and imports the test project into the IDE.



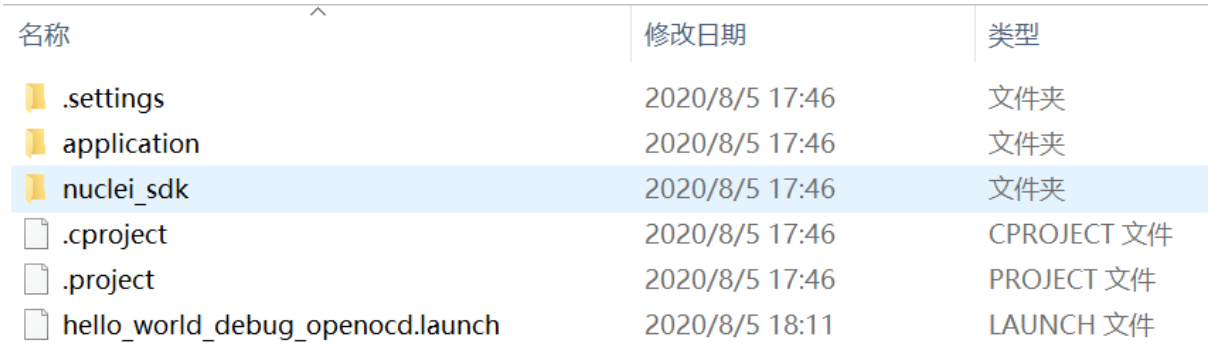
2.7.3 Import an Existing Project Directly to Create a New Project

This section describes how to use the IDE to import an existing project directly and create a new project from it. This document takes the N307 project package as an example. The project package is available at (https://github.com/riscv-mcu/Nuclei-Studio_IDE-Project-Package). For other project packages, contact Nuclei.

In a Windows-based Nuclei Studio IDE development environment, if you use `Manually create a project without a template`, you also need to load the nuclei-sdk folder from this project package. For details, see *Manually Create a Project Without a Template* (page 113).



After downloading and extracting the nuclei-eclipse_demo.rar archive, the contents are as follows:

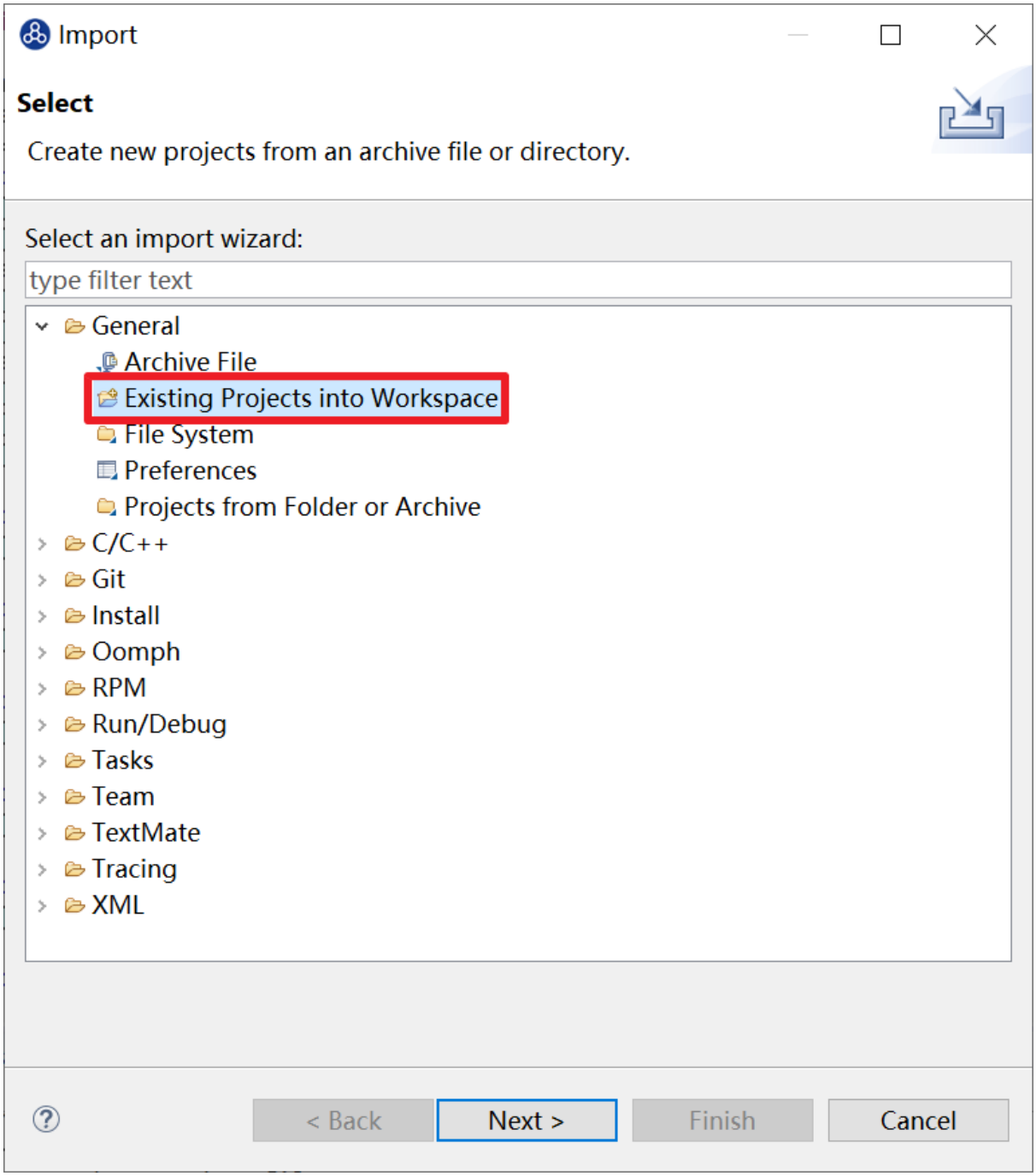


- The project description files `.setting`, `.project`, and `.cproject`
- The project Debug settings files `*.launch`
- The nuclei_sdk folder

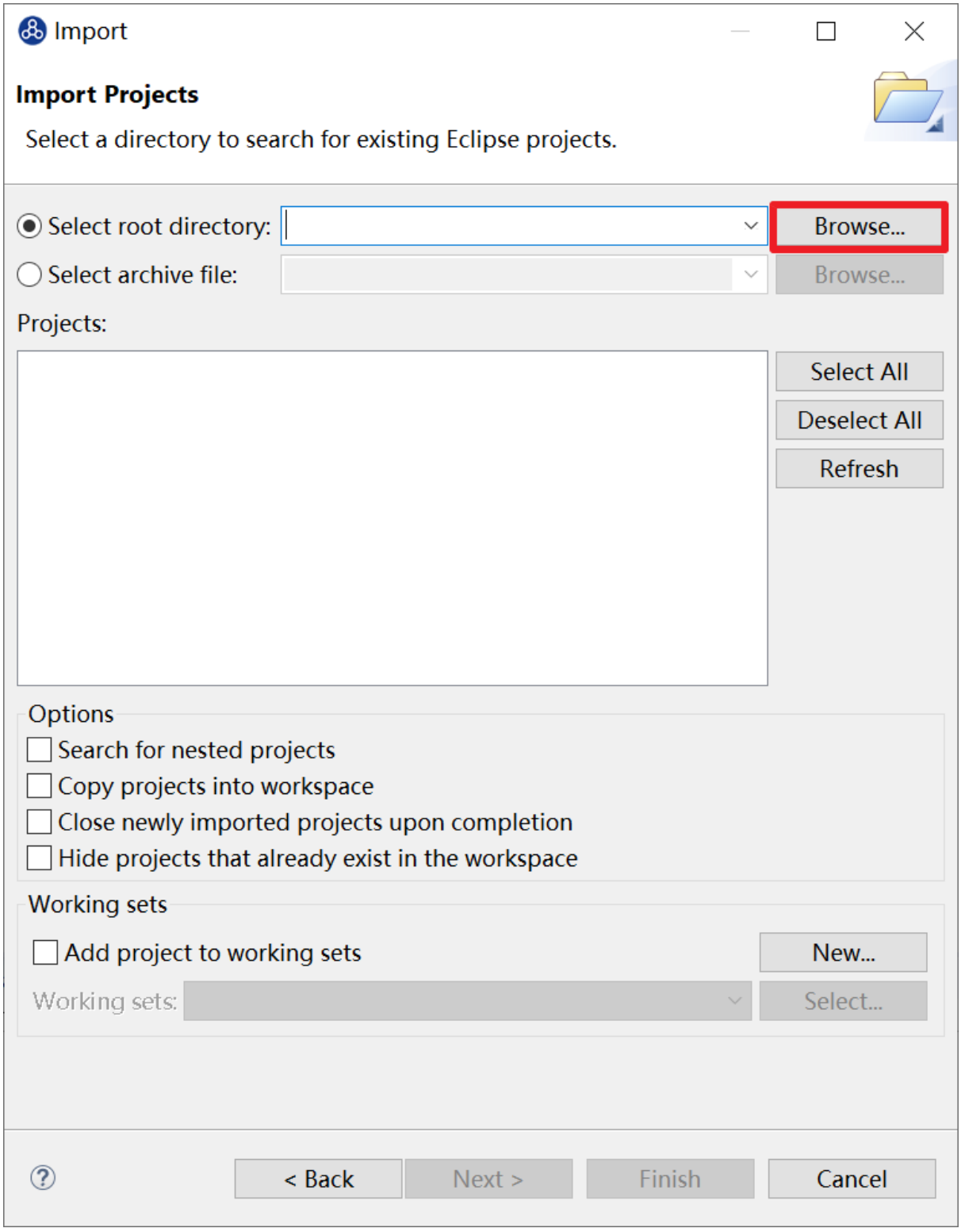
  This folder contains part of the SDK source code.
- The application folder

  This folder contains the main function source code of the hello_world sample program.

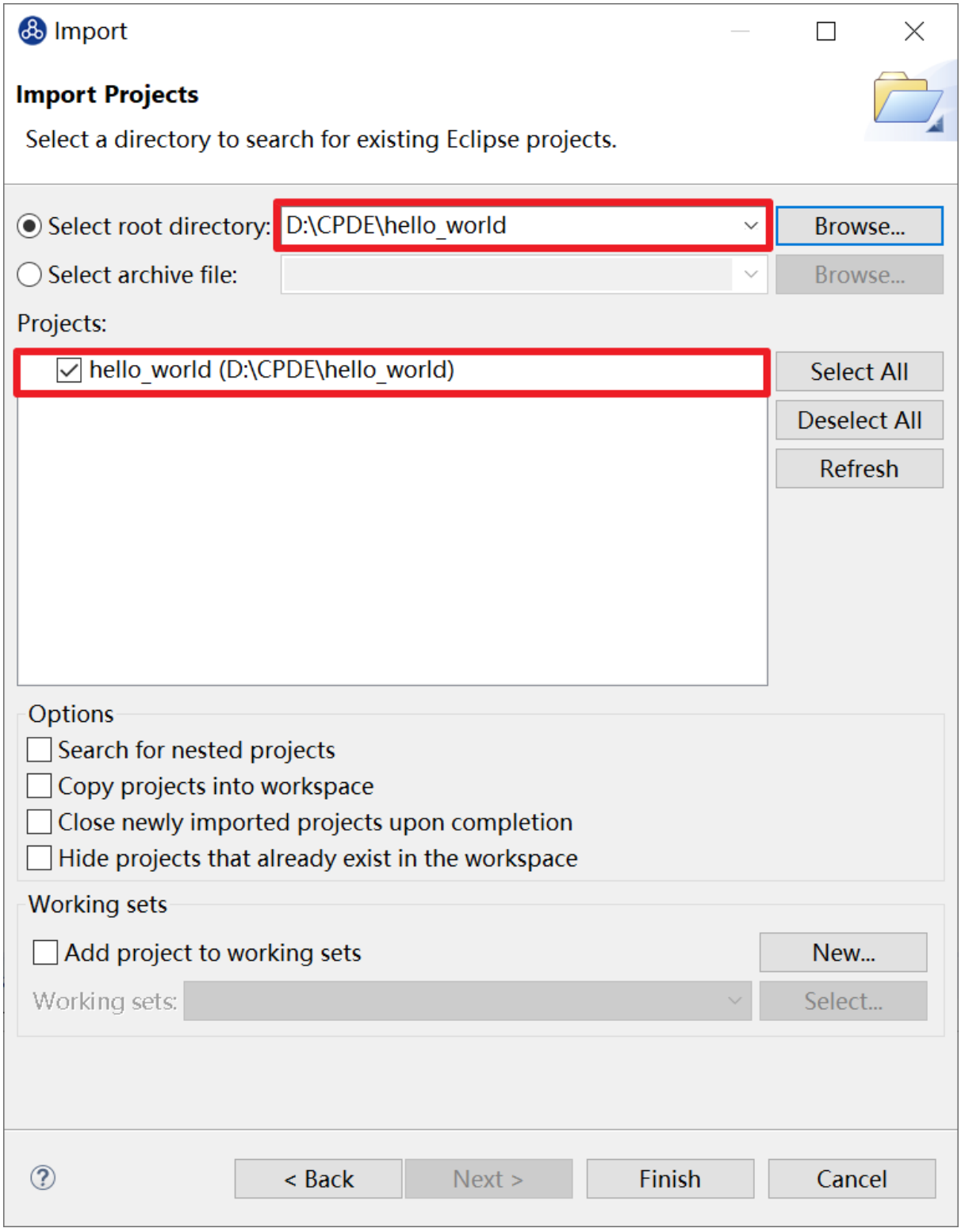
  Next, import the downloaded project package as follows:
- Select `File -> Import` in the menu bar.
- As shown in the figure, select `Existing Projects into Workspace` and click `Next`.



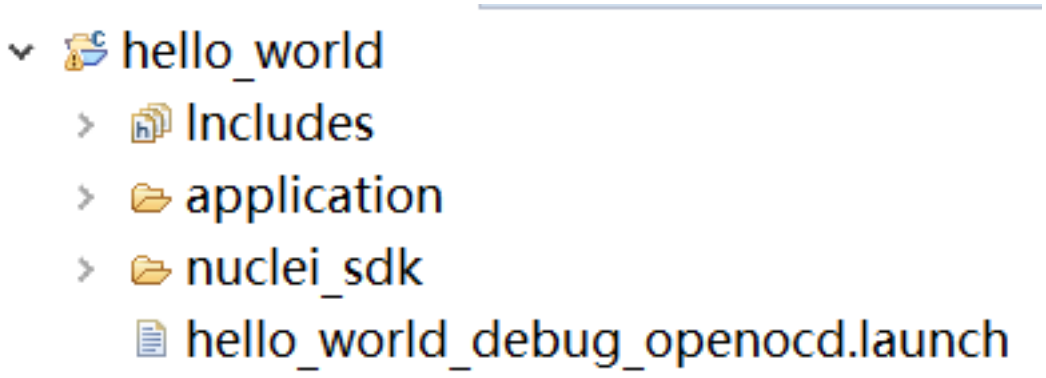
- Click `Browse` and select the path of the project to be imported, as shown in the figure.



- The IDE recognizes the project to be imported. Click `Finish`.



- The directory structure of the imported project appears in the IDE ' s Project Explorer as shown in the figure. The existing project uses the N307 build options by default, and the Nuclei SDK only contains the files used by helloworld. For more Nuclei SDK source code, visit GitHub (https://github.com/riscv-mcu/hbird-sdk).



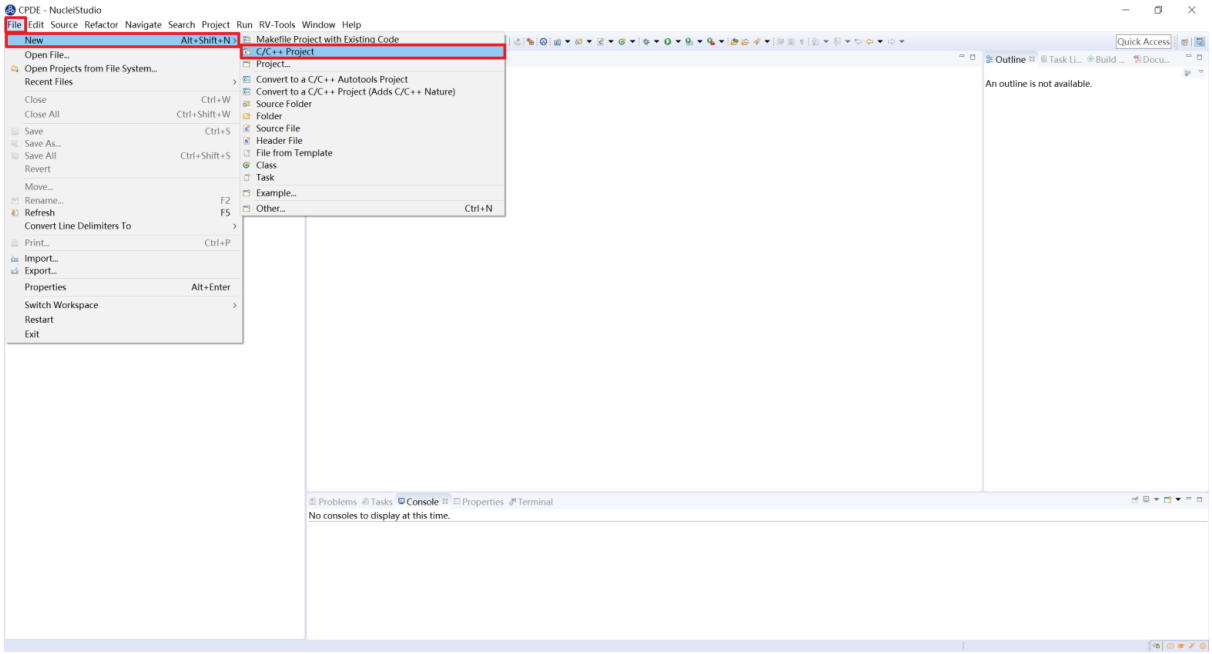
2.7.4 Manually Create a Project Without a Template

This section describes how to manually create a user-defined Hello World project in Nuclei Studio IDE. The board is the Nuclei FPGA Evaluation Board and the core is N307. In addition to creating the project, this method requires manually setting various options and paths.

Note: Not recommended. Use the NPK template method to create a project instead.

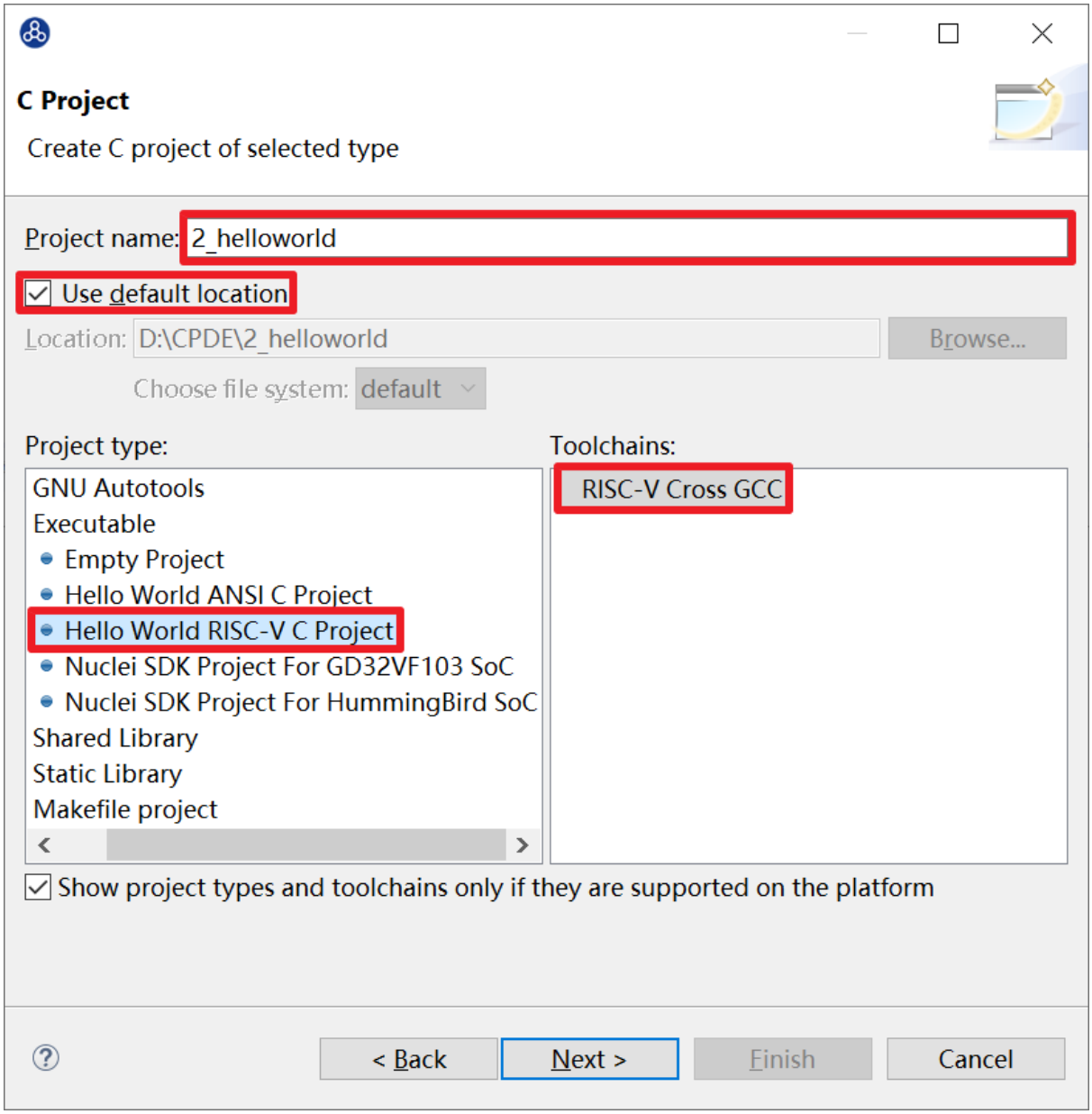
Manually Create a Project

In the Nuclei Studio main menu bar, select `File -> New -> C/C++ Project`.

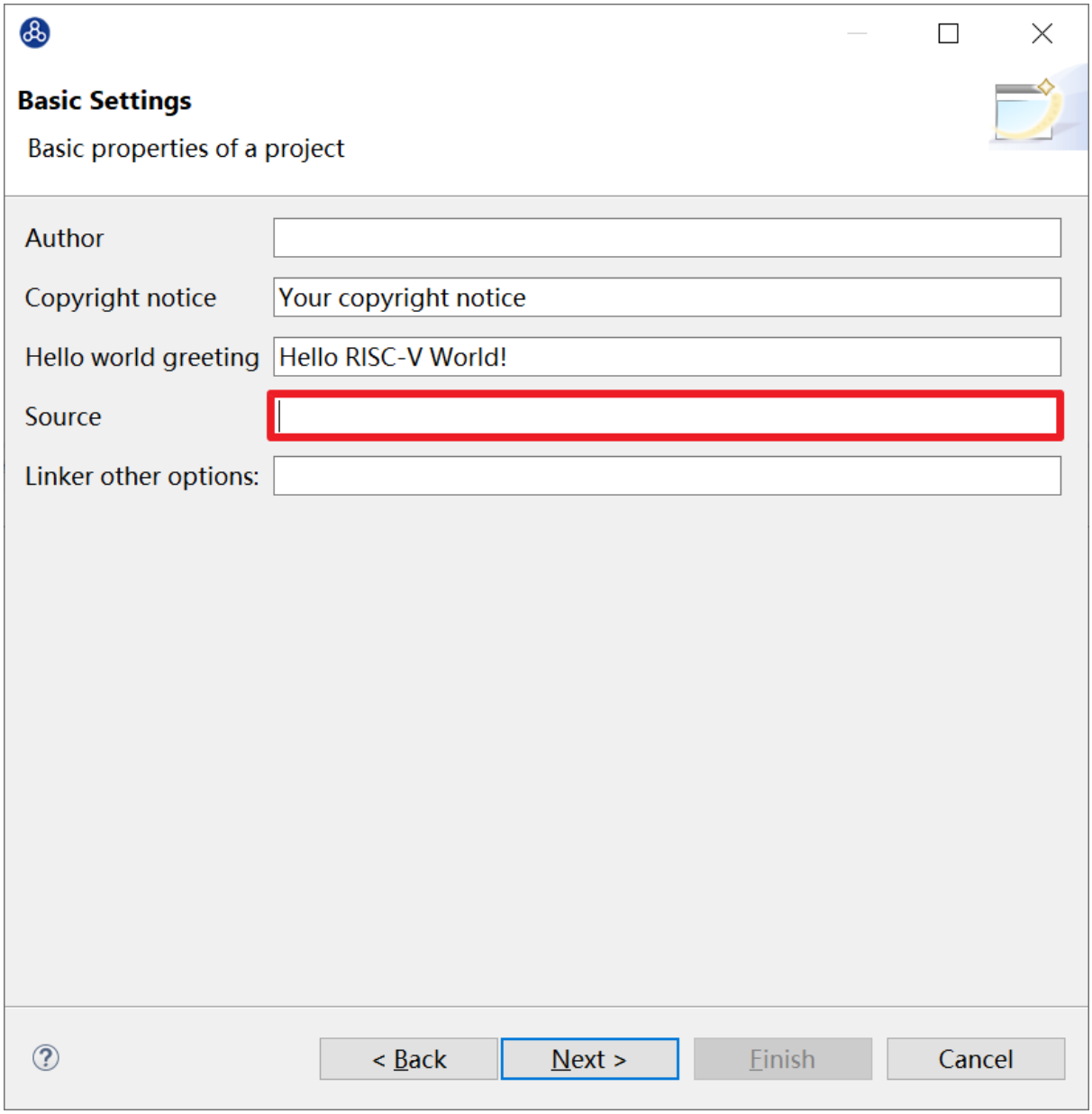


Then set the following parameters in the pop-up window.

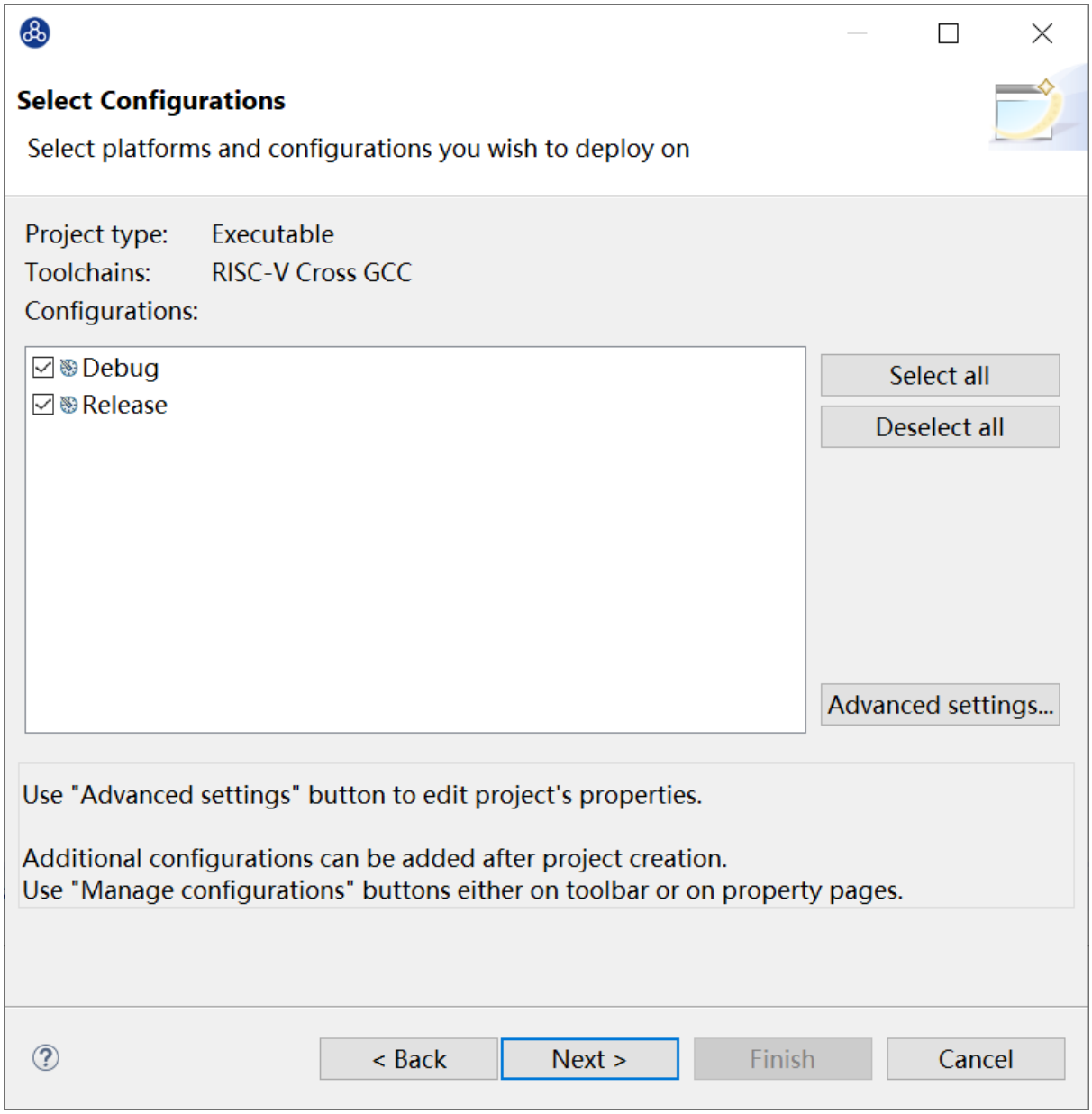
- Project name: the project name.
- Use default location: if this option is checked, the default Workspace folder is used to store this project.
- Project type: select `Hello World RISC-V C Project`.



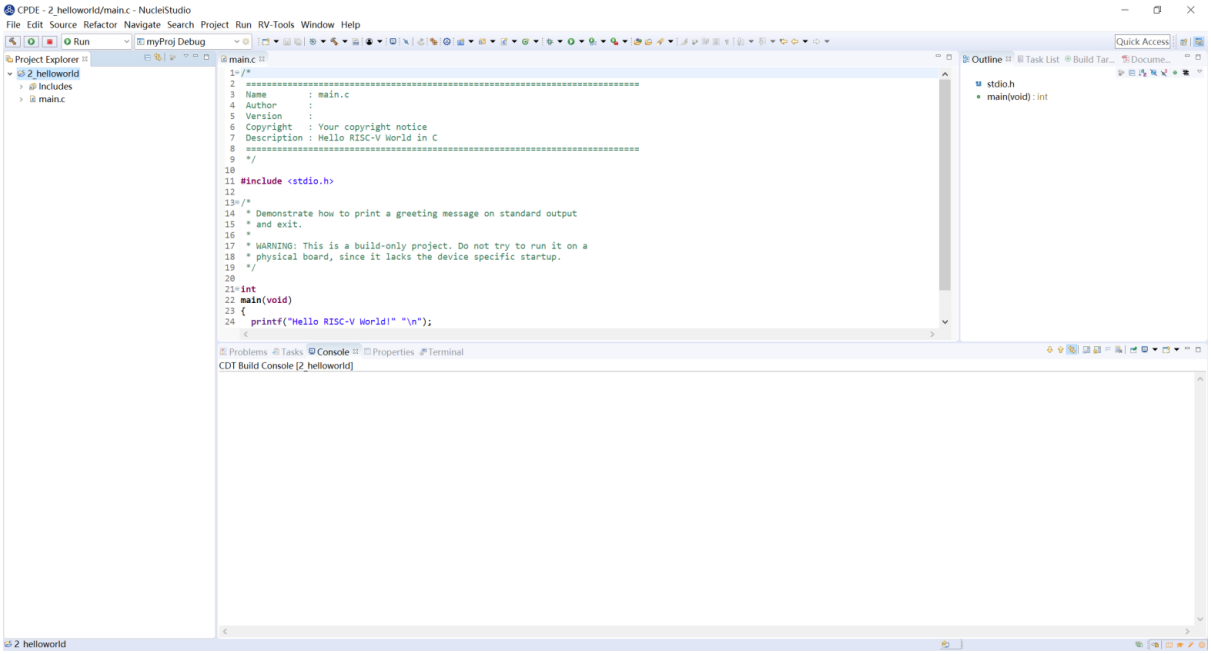
Then click `Next` to proceed to the next step, and set the basic information of the Hello World project in the pop-up window. Leave the `Source` option empty and click `Next` directly to proceed.



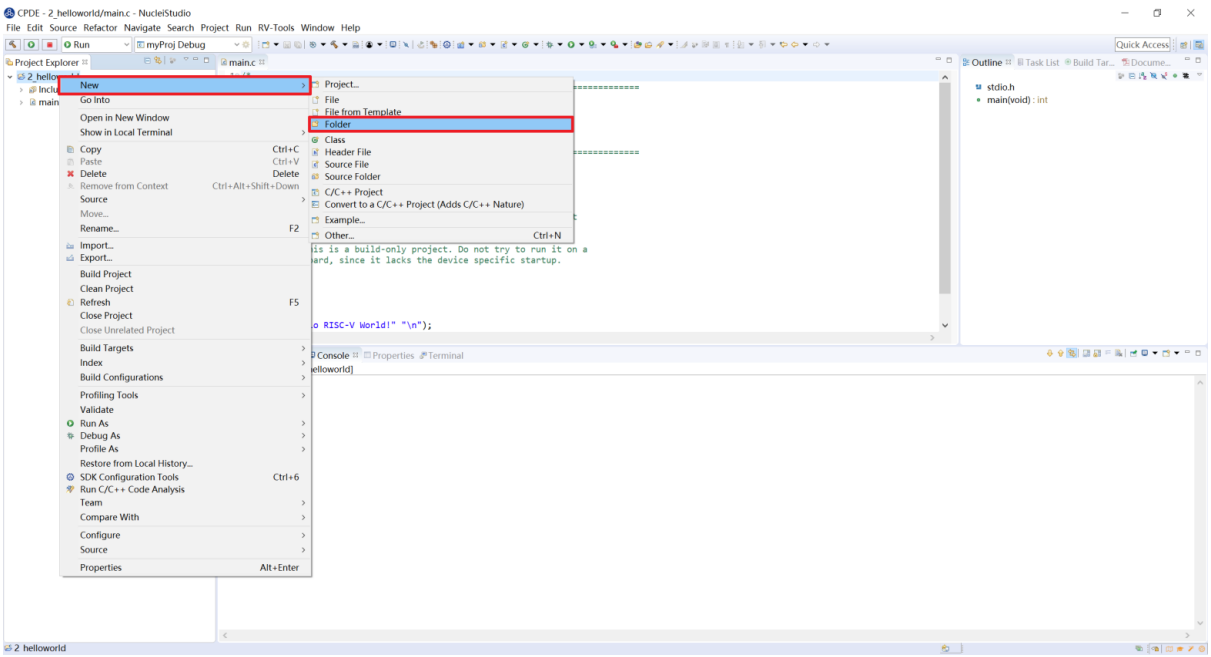
Set the Debug or Release attributes of the project in the pop-up window. Use the default settings in this step without any modification, and click `Next` directly to proceed.



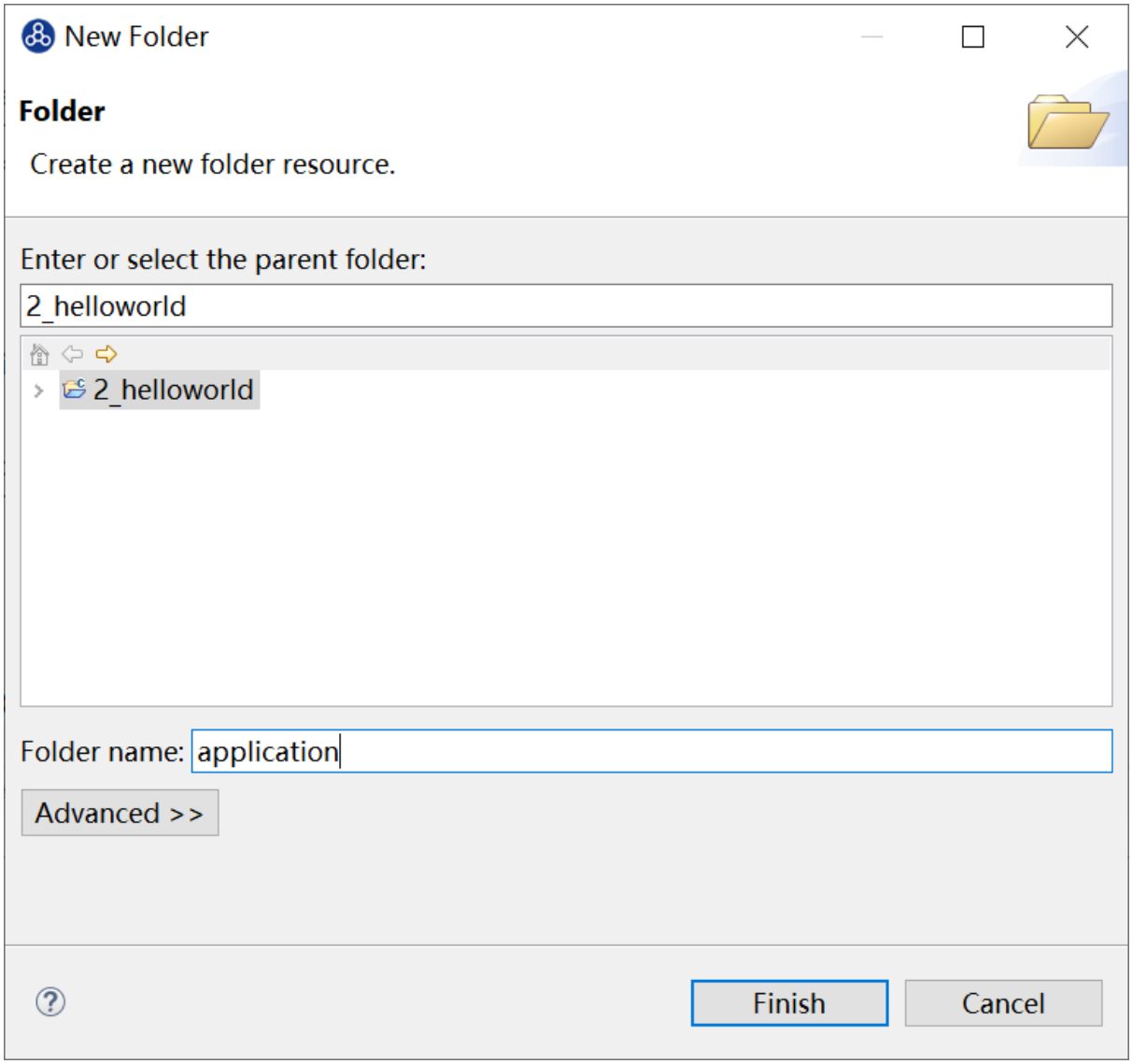
Set the RISC-V toolchain used by the project in the pop-up window. Do not configure anything here; click `Finish` directly. The HelloWorld project is now created.



After creation, the Hello World project is displayed as follows.



Create a new application folder. Right-click the project and select `New -> Folder`, enter `application`, and click `Finish`. Then drag `main.c` into the application folder to organize the files.

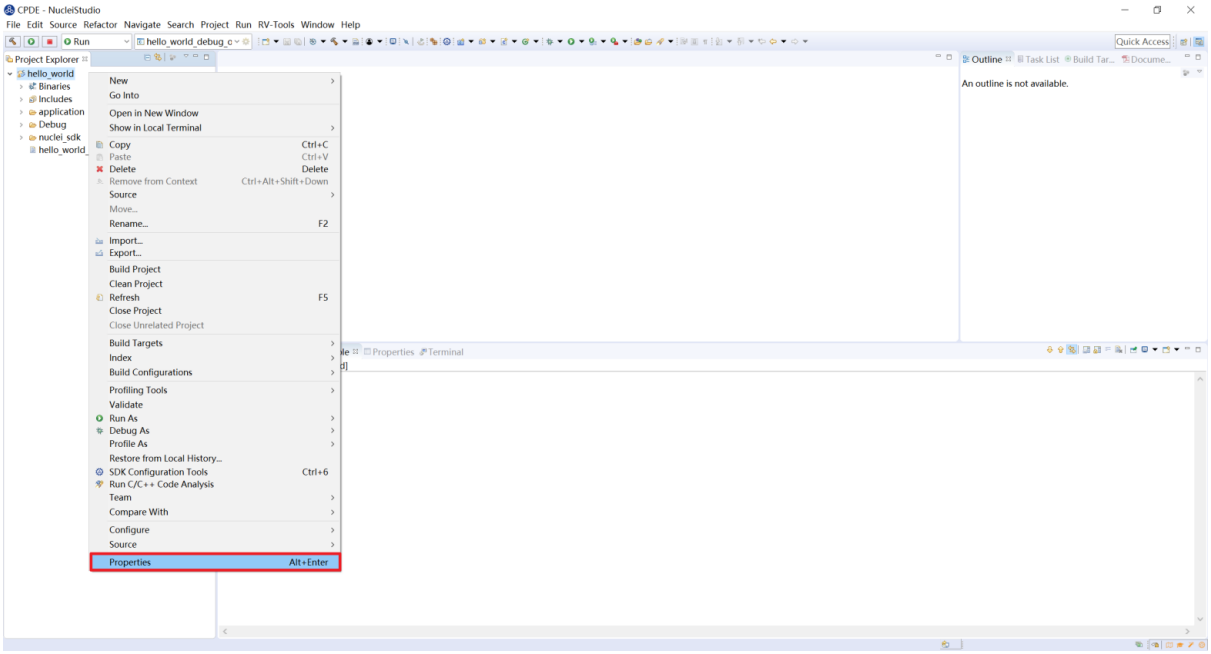


Configure nuclei_sdk for the Project

This section describes how to add nuclei_sdk to the project. The details of the SDK are not covered here; refer to https://doc.nucleisys.com/nuclei_sdk/index.html. If you need to use other SDK source files, obtain the full Nuclei SDK source code (version 0.3.9 is used here as an example) from GitHub: https://github.com/Nuclei-Software/nuclei-sdk/releases. This section only describes the steps to add the files needed by helloworld in nuclei_sdk to the project. If you use a newer version of the SDK, the corresponding directory structure may have changed; adjust accordingly.

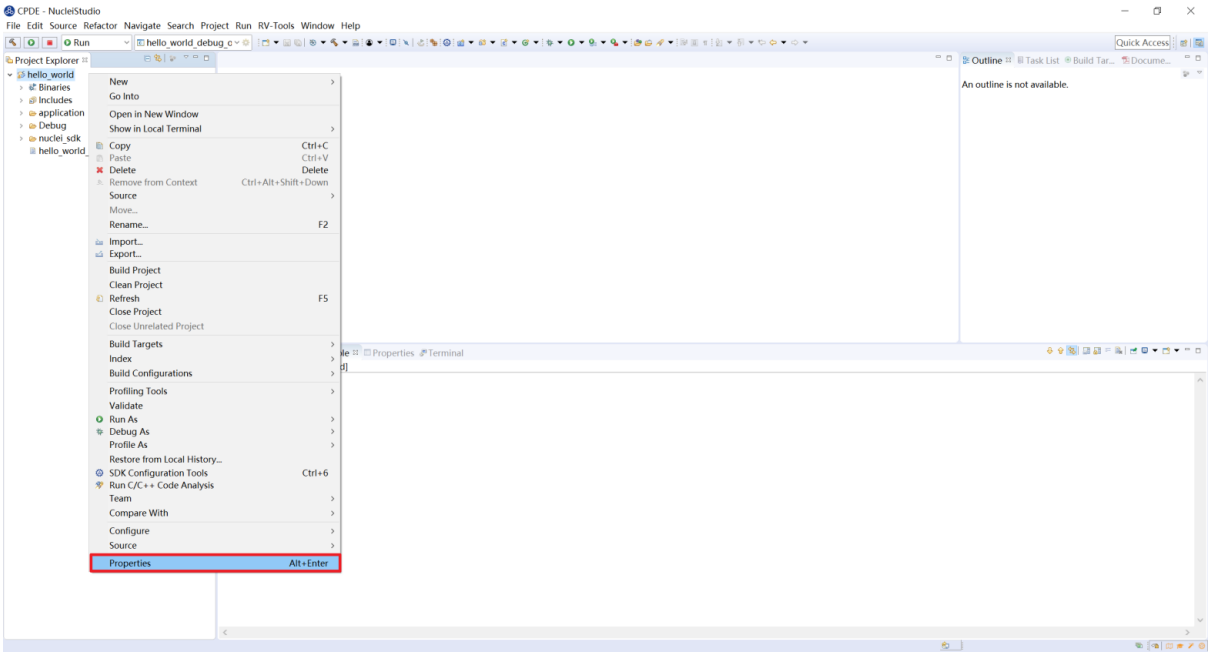
Open the `2_helloworld` project in Nuclei Studio, and add the nuclei_sdk source files as follows.

In the Project Explorer, select the `2_helloworld` project, right-click, and select `Properties` to open the project settings page.

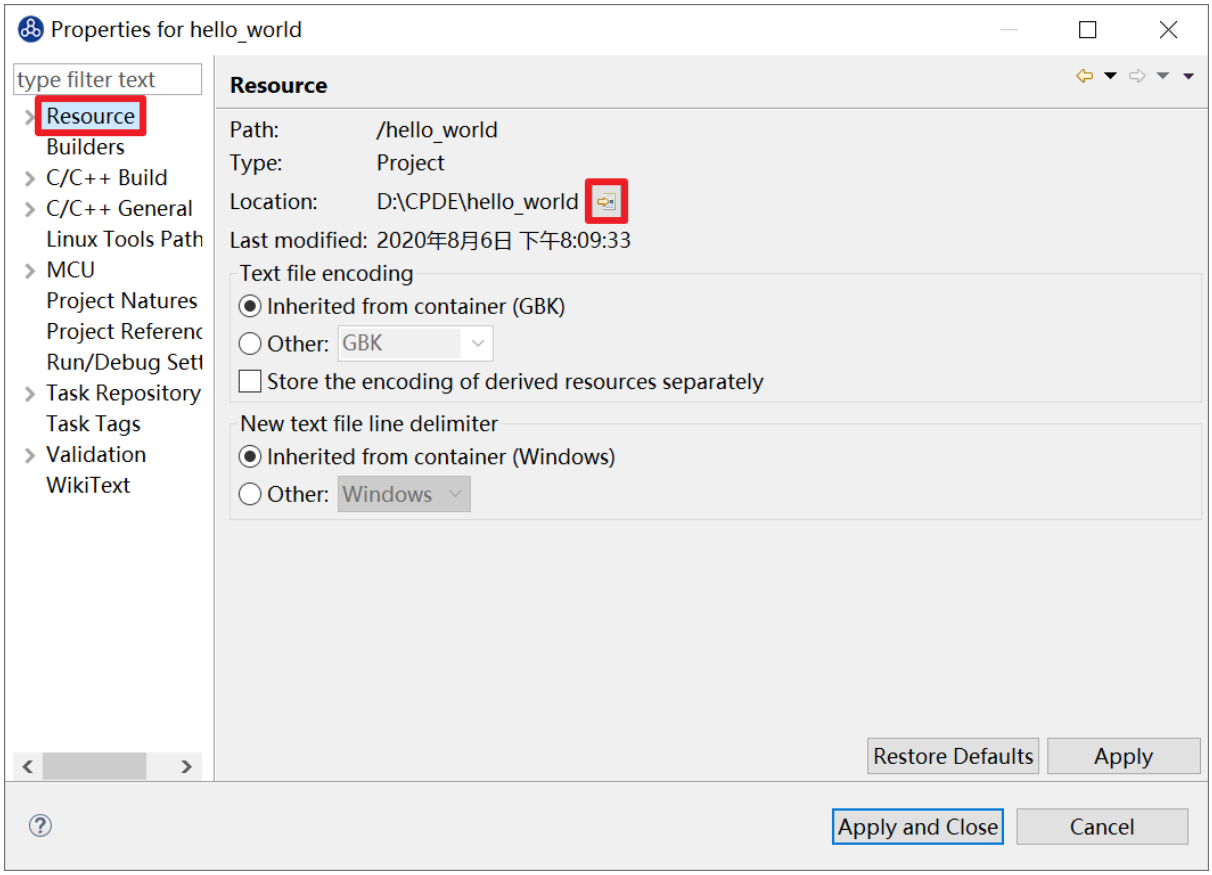




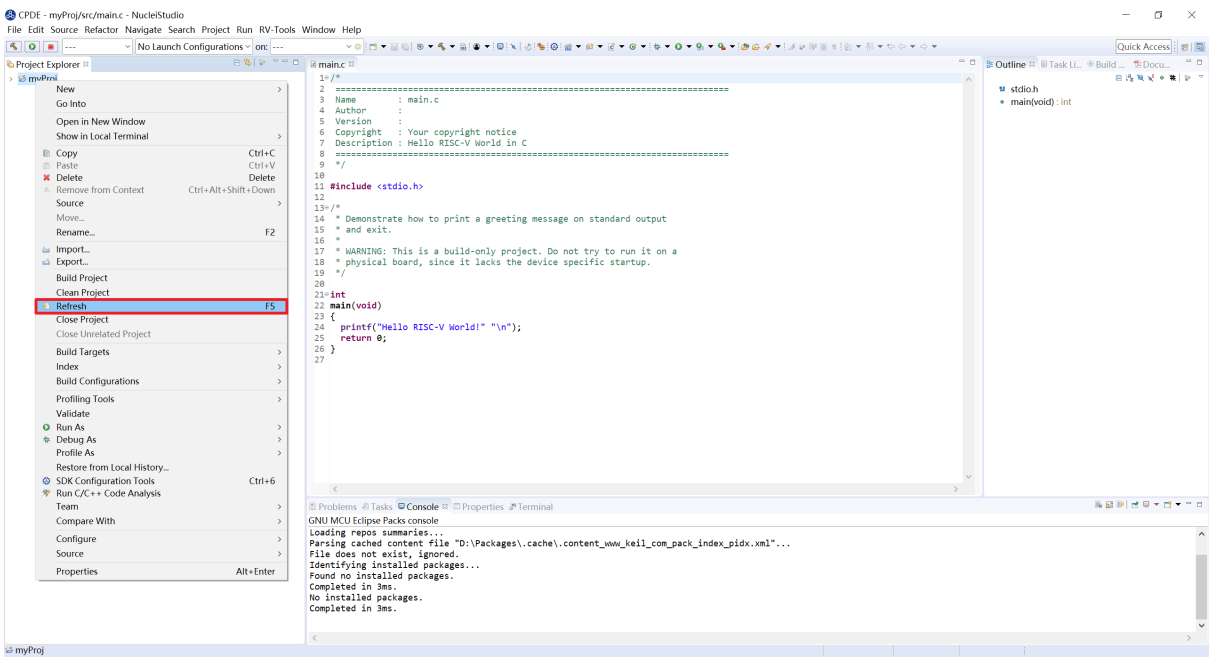
In the pop-up window, click `Resource`, and in the Location column on the right, click the arrow icon on its far right. A file window pops up at the folder location of the `2_helloworld` project.



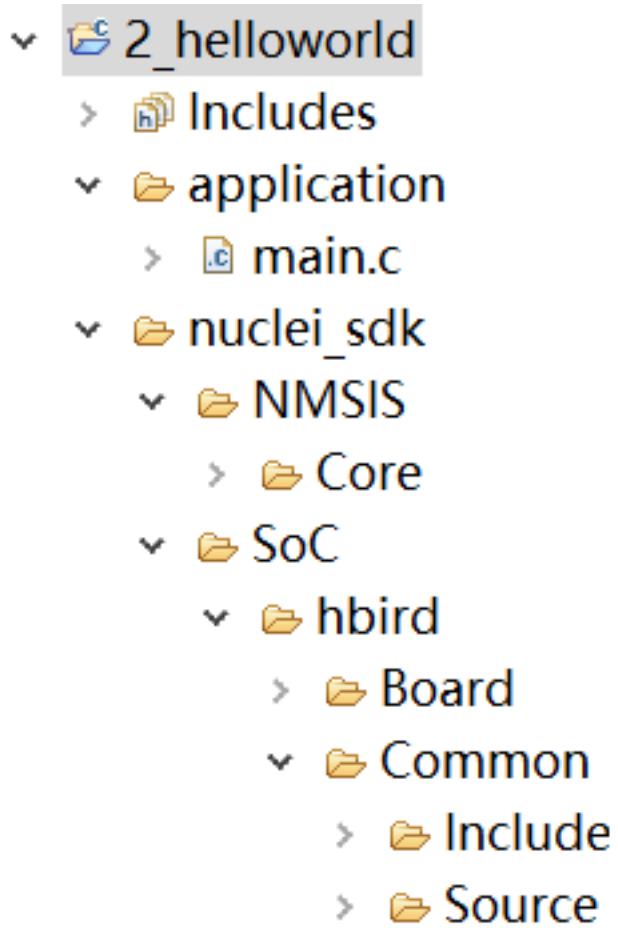
Copy the nuclei_sdk folder from the nuclei-eclipse_demo.rar archive into the directory of the `2_helloworld` project.



Return to Nuclei Studio, select the `2_helloworld` project in the Project Explorer, right-click, and select `Refresh`.



After refreshing, the nuclei_sdk folder appears under the `2_helloworld` project. The import of the nuclei_sdk source files is now complete.



Configure the Compile and Link Options of the Project

To compile the project source code correctly, you need to configure the compile and link options.

Note: The compile and link options set in this section are common options of the GCC toolchain and have the same meaning as the options of the same name used in a Linux environment. They are not described in detail in this section.

Configure the compile and link options as follows:

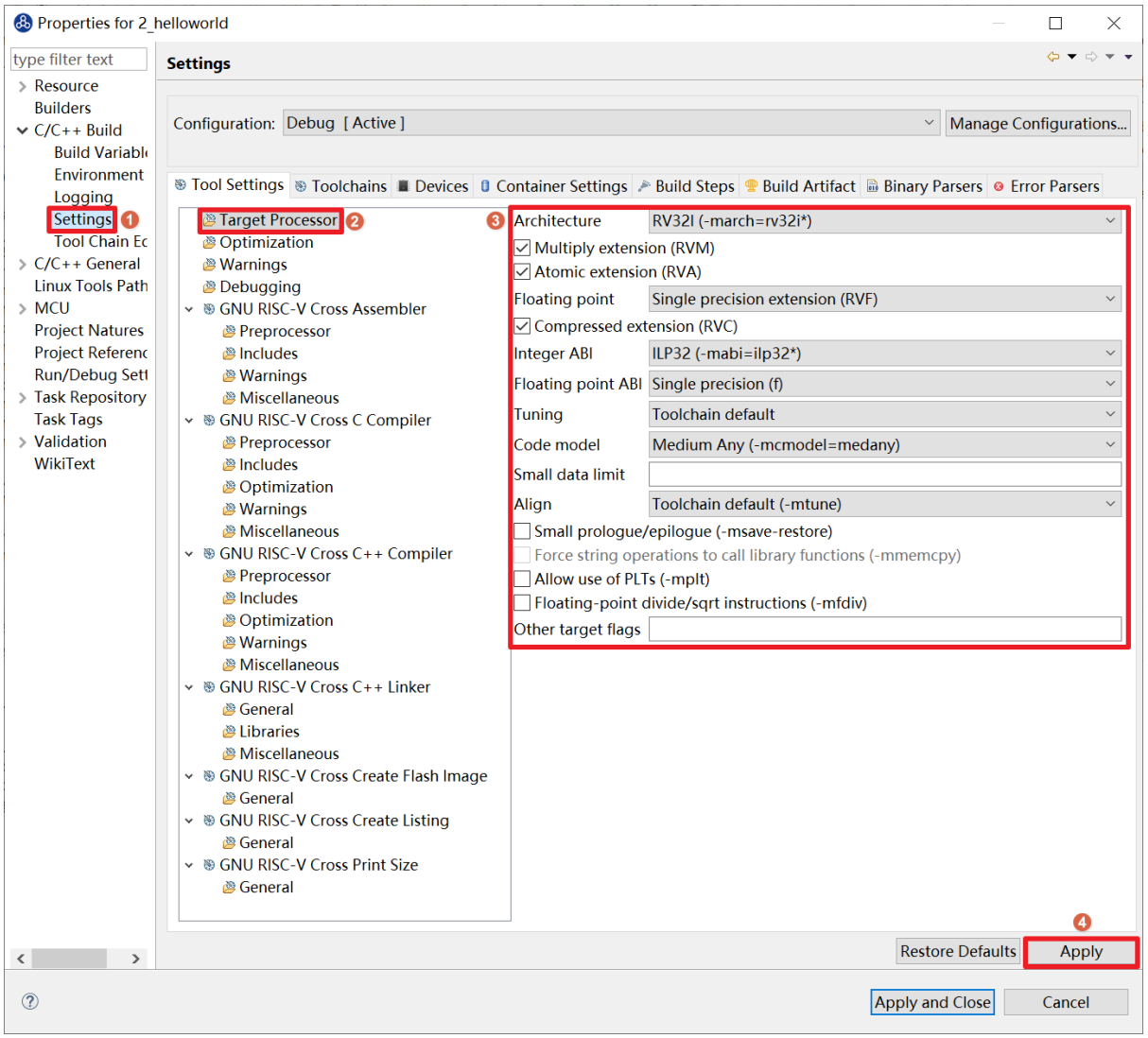
In the Project Explorer, select the hello_world project, right-click, and select `Properties`.

In the pop-up window, expand the C/C++ Build menu, click `Settings`, and configure the options in the Tool Settings column on the right.

Select Target Processor. Our core is N307, so check the configuration options as shown in the figure.

- Architecture: select `RV32I`.
- Multiply extension (RVM): check this option.
- Atomic extension (RVA): check this option.
- Compressed extension (RVC): check this option.
- Integer API: select `ILP32`.
- Floating Point ABI: select `single precision`
- Code model: select `Medium Any`.

- Click the `Apply` button in the lower-right corner.



Select `Optimization` and check the configuration options as shown in the figure.

- Optimization Level: select `Optimization Most (-O2)`.

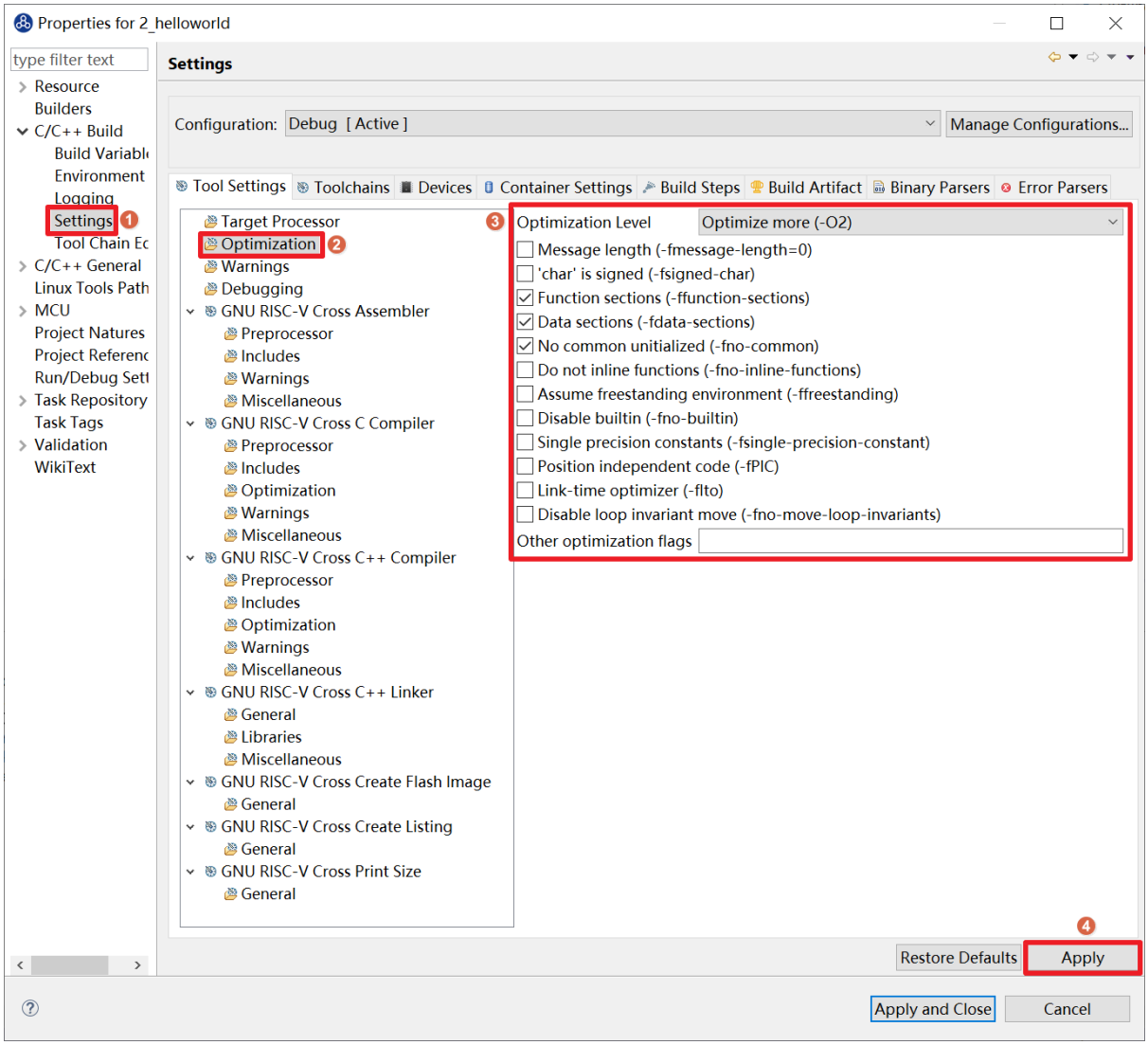
Note: Nuclei Studio 2024.06 adds `-Oz`, which is used to optimize the size of the compiled program.

Check the following options:

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

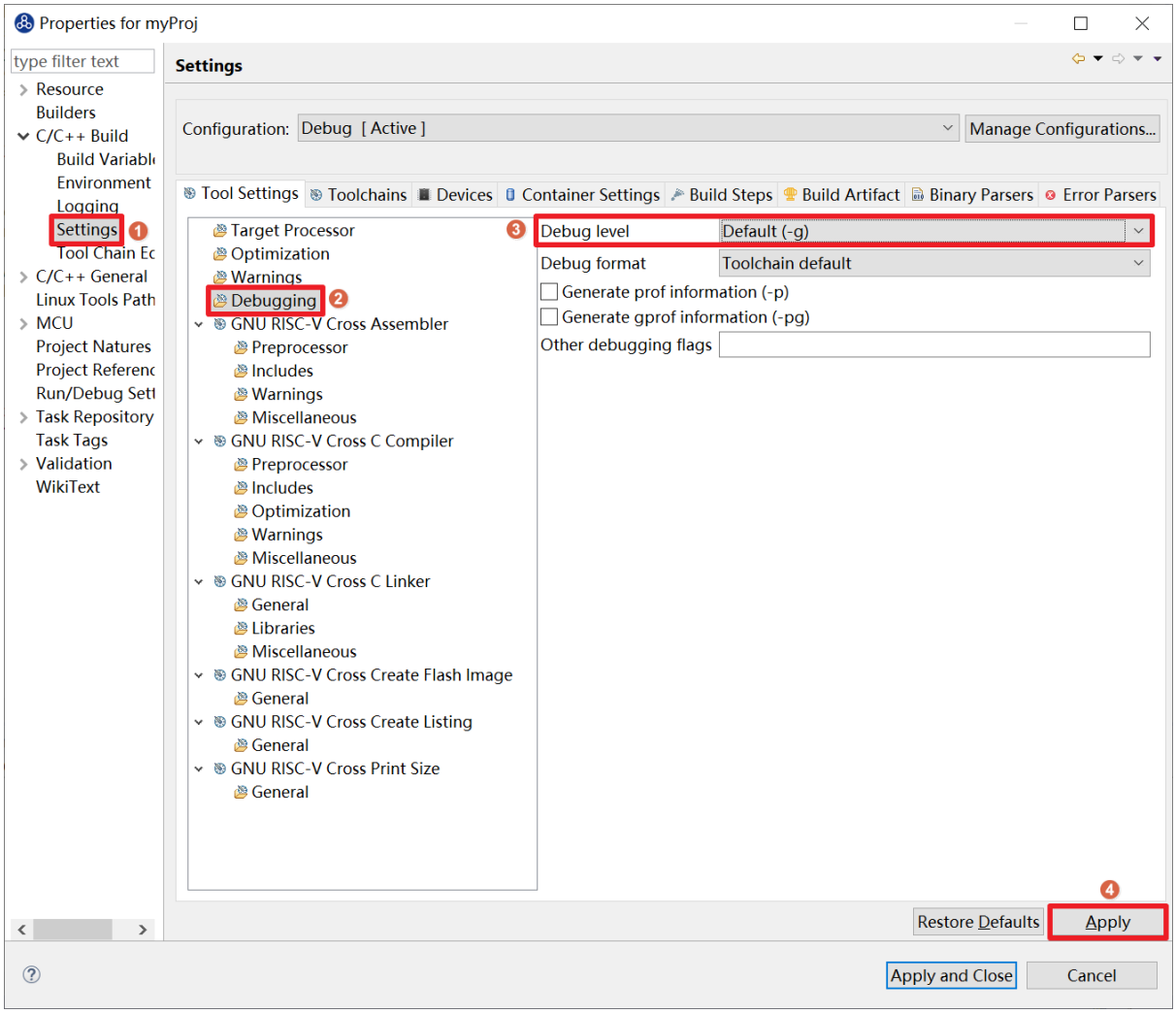
Note: The above options are common GCC compile optimization options. Refer to the GCC manual for their meanings.

Click the `Apply` button in the lower-right corner.



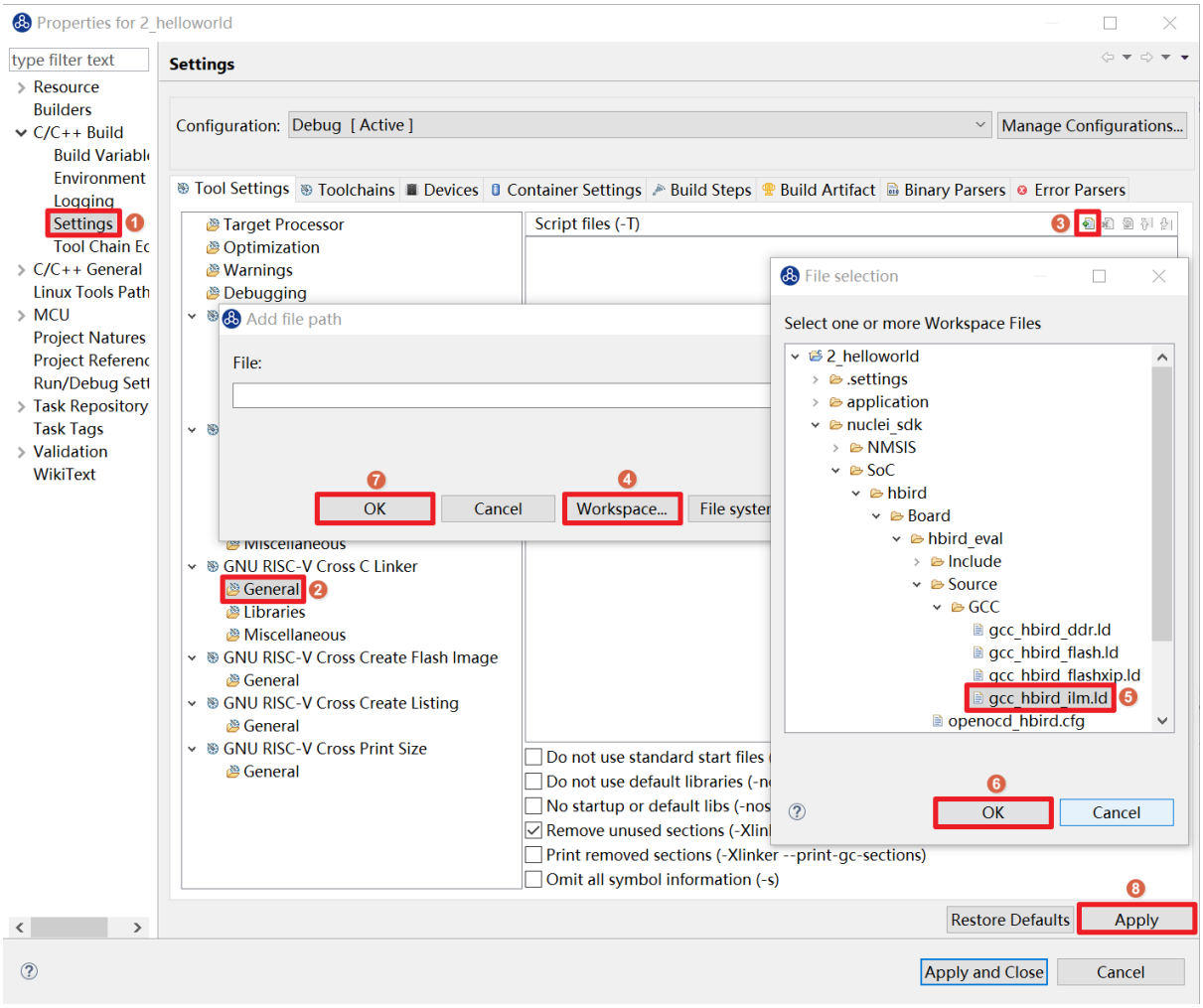
Select Debugging and check the configuration options as shown in the figure:

- Debug Level: select `Default (-g)`.
- Click the `Apply` button in the lower-right corner.



Select General under GNU RISC-V Cross C Linker. Set the linker script required by the linker as follows.

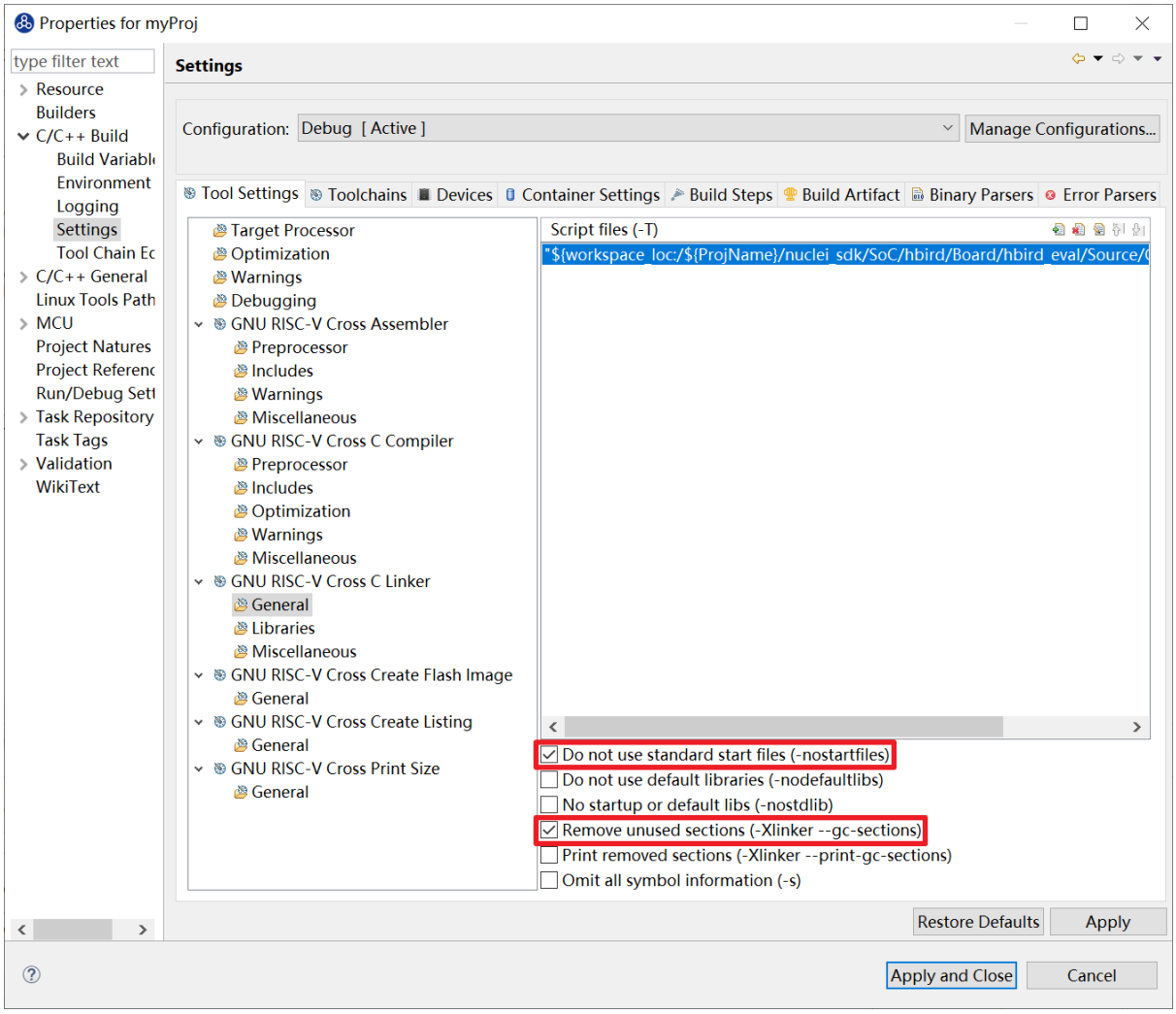
- Click the plus button in the upper-right corner.
- Click the `Workspace` button in the pop-up window.
- Here we use the HummingBird evaluation board, so select the `gcc_hbird_ilm.ld` file corresponding to the ILM download mode. In the pop-up window, select the `gcc_hbird_ilm.ld` file under the `nuclei_sdk/SoC/hbird/Board/hbird_eval/Source/GCC` folder of the Nuclei Studio package. For other download modes, switch the file here. Each file is introduced in detail below; choose according to your actual situation.
  - The `gcc_hbird_ilm.ld` script constrains the program ' s code section to the ILM address region, which means the program is downloaded directly into the MCU ' s ILM and executed from ILM. ILM consists of SRAM, so its contents are lost when power is off.
  - The `gcc_hbird_flash.ld` script constrains the physical address of the program ' s code section to the Flash region and the logical address of the code section to the ILM address region, which means the program is downloaded directly into the MCU ' s Flash, but after power-on a boot program moves the code section into ILM, and execution starts from ILM.
  - The `gcc_hbird_flashxip.ld` script constrains the program ' s code section to the Flash region, which means the program is downloaded directly into the MCU ' s Flash and executed directly from Flash. The program is flashed into Flash, so it is not lost when power is off.
  - Select a suitable linker script according to your needs. This section uses `gcc_hbird_ilm.ld` as an example.
- After the settings are complete, click the `Apply` button in the lower-right corner.



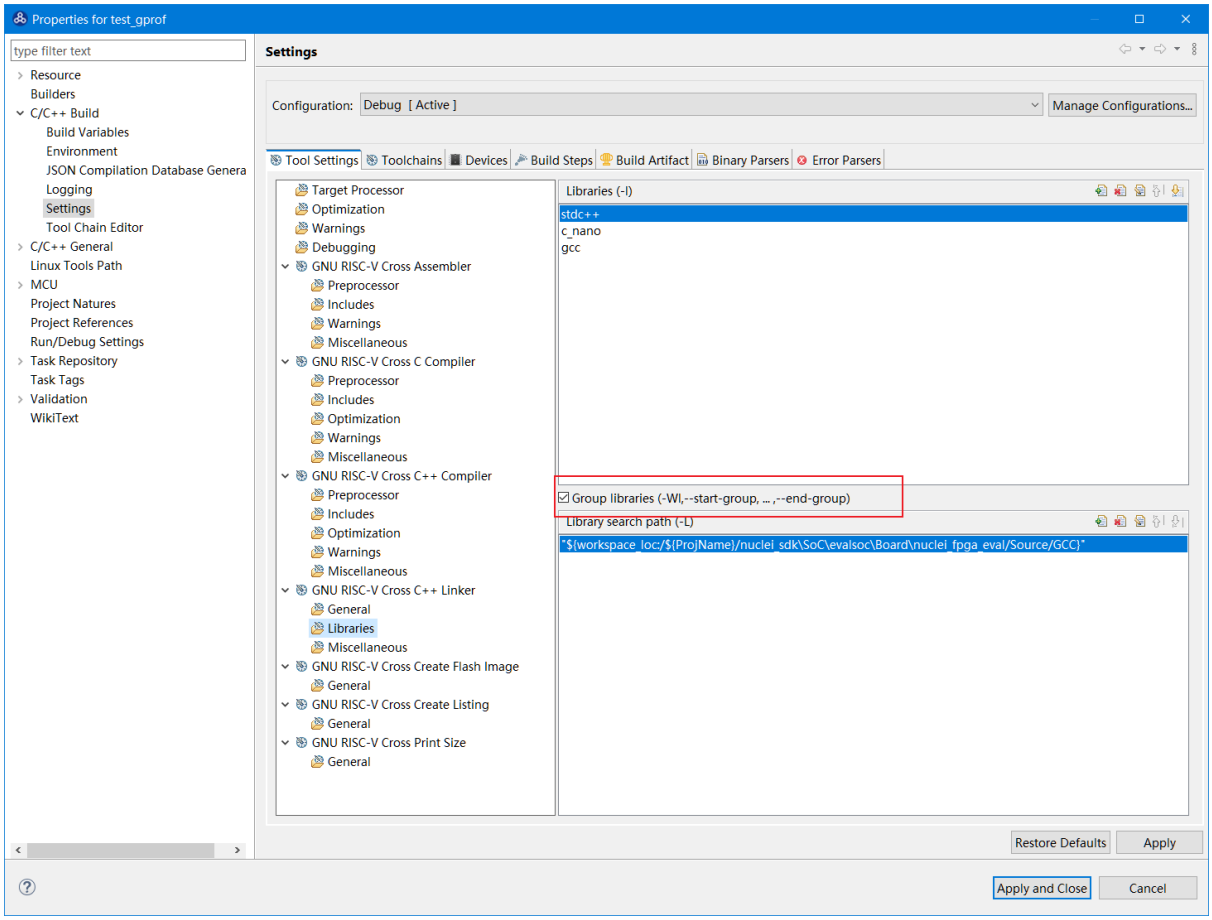
Check the configuration options as shown in the figure below.

- Do not use standard start files (-nostartfiles).
- Remove unused sections (gc-sections).
- Click the `Apply` button in the lower-right corner.

Note: The above options are common GCC link options. Refer to the GCC manual for their meanings.

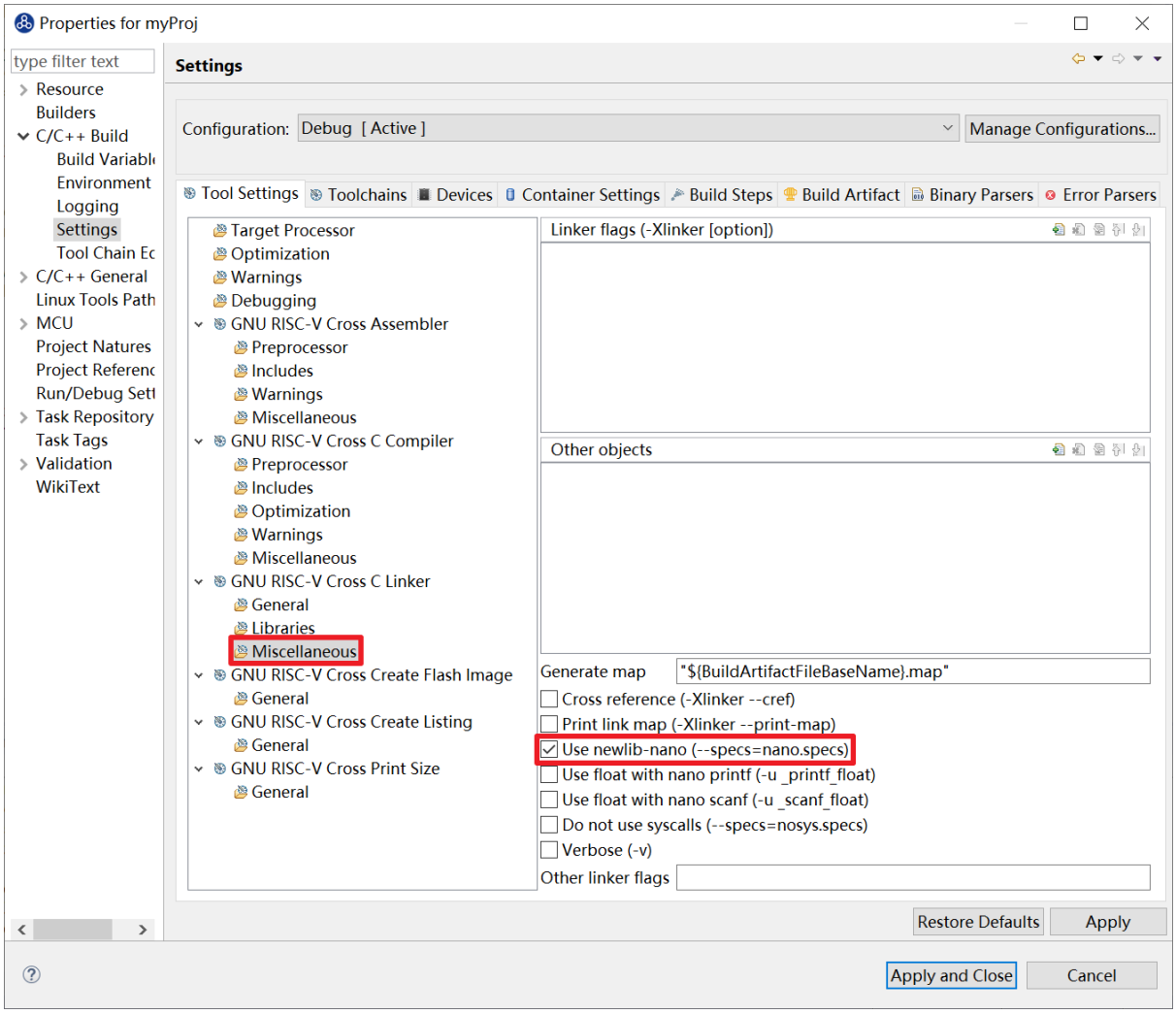


Note: In Nuclei Studio 2024.06, Libraries supports the Group feature. If the Group feature is checked, all Libraries are linked with `-Wl,--start-group ... --end-group`, which can resolve inter-dependency problems within Libraries.



Select Miscellaneous under GNU RISC-V Cross C Linker, and check the configuration options as shown in the figure below.

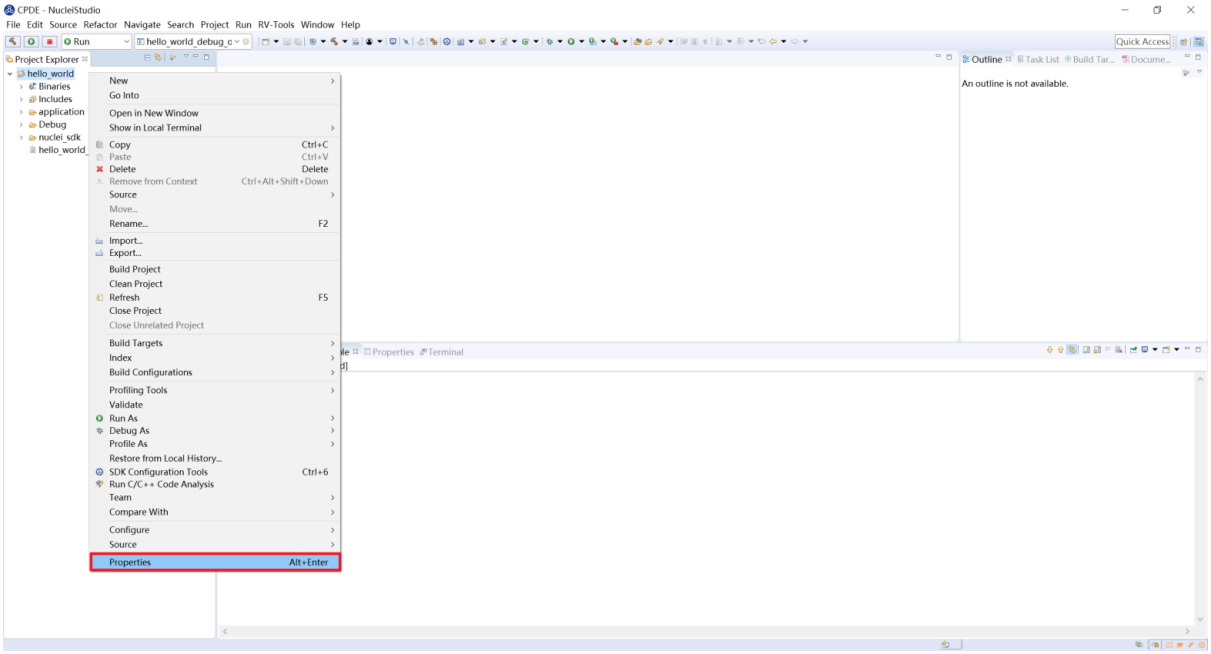
- Check `Use newlib-nano`.
- Because the Hello World program ' s printf does not need to print floating-point numbers, do not check `Use float with nano printf`.
- Click the `Apply` button in the lower-right corner.



Configure the Include Paths and Files of the Project

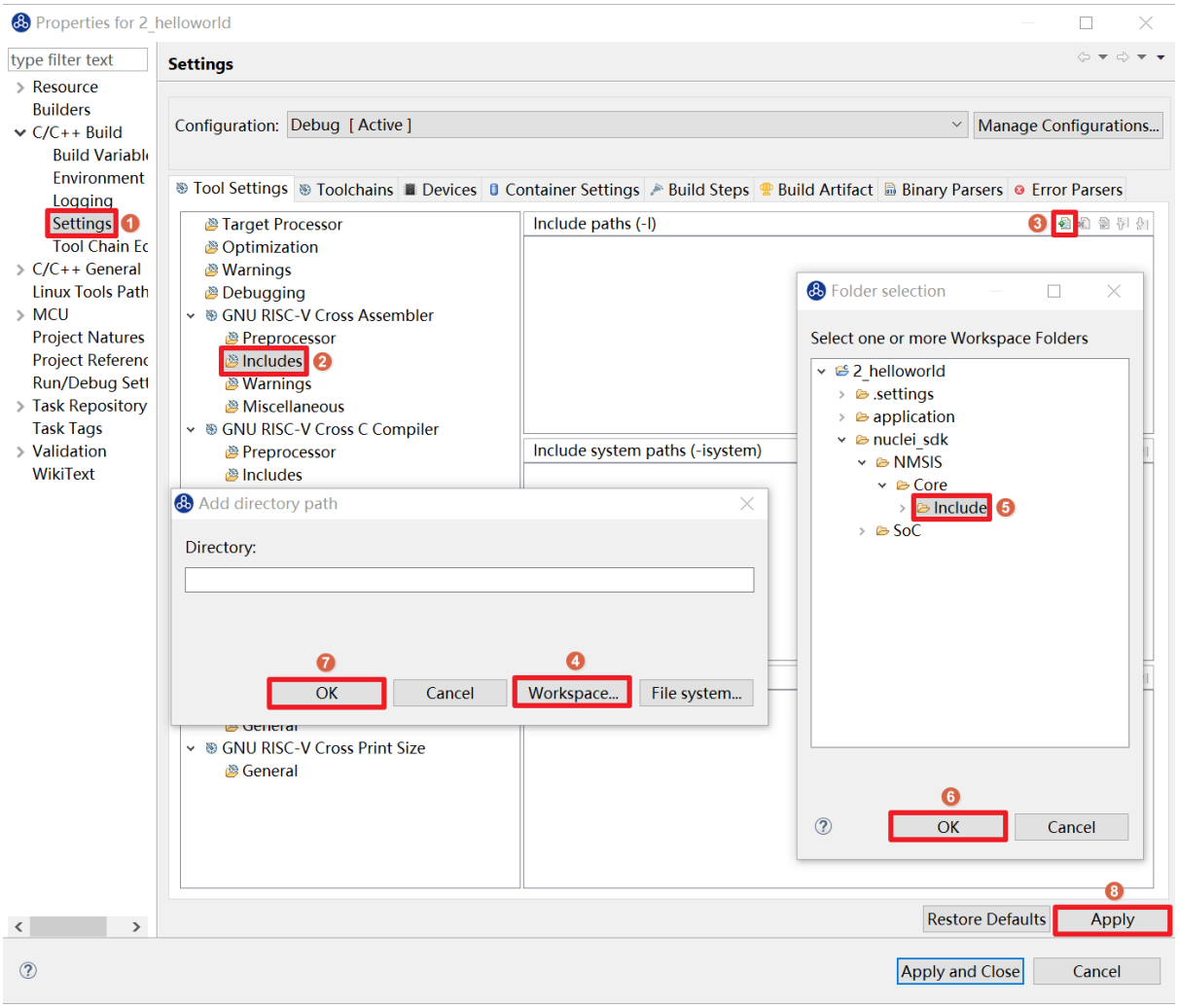
To compile the source files in the nuclei_sdk folder correctly, you need to configure the include paths and include files of the project as follows.

In the Project Explorer, select the hello_world project, right-click, and select `Properties`.



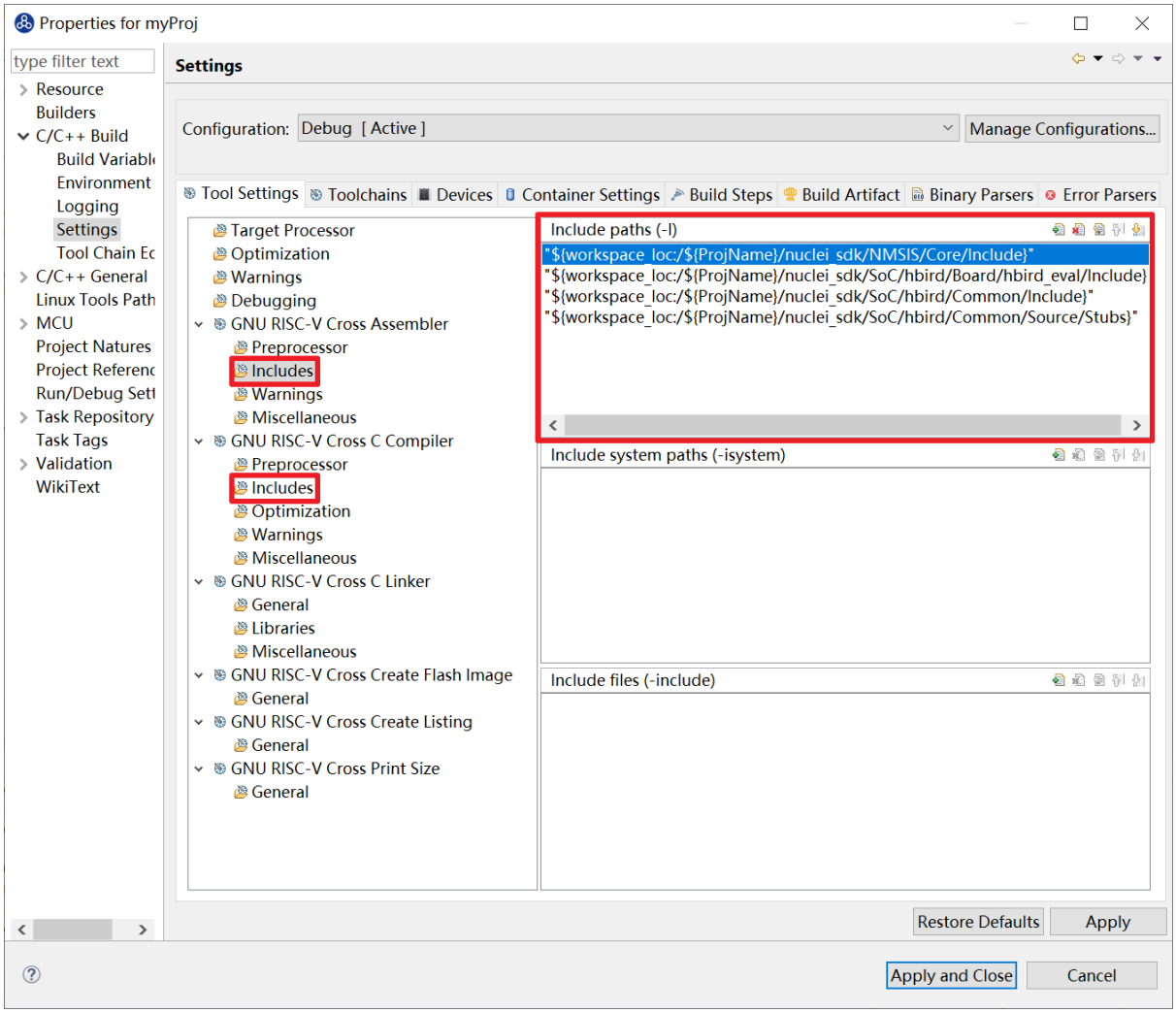
In the pop-up window, expand the C/C++ Build menu, click `Settings`, and configure the options in the Tool Settings column on the right.

Select Includes under GNU RISC-V Cross C Assembler, and configure the include files as shown in the figure.



- Click the plus button in the Include paths column.
- Click `Workspace` in the pop-up window to open the Folder selection window.
- In the Folder selection window, select the NMSIS>Core>Include folder under the nuclei_sdk directory of the project.
- Click `Apply` in the lower-right corner to complete the configuration.

Using the same method, add the `SoC>hbird>Board>hbird_eval>Include`, `SoC>hbird>Common>Include` and `SoC>hbird>Common>Source>Stubs` folders under the nuclei_sdk directory as include paths, and use the same method to set the include paths for the `Includes` column of `GNU RISC-V Cross C Compiler`. The result after the settings are complete is shown in the figure below.

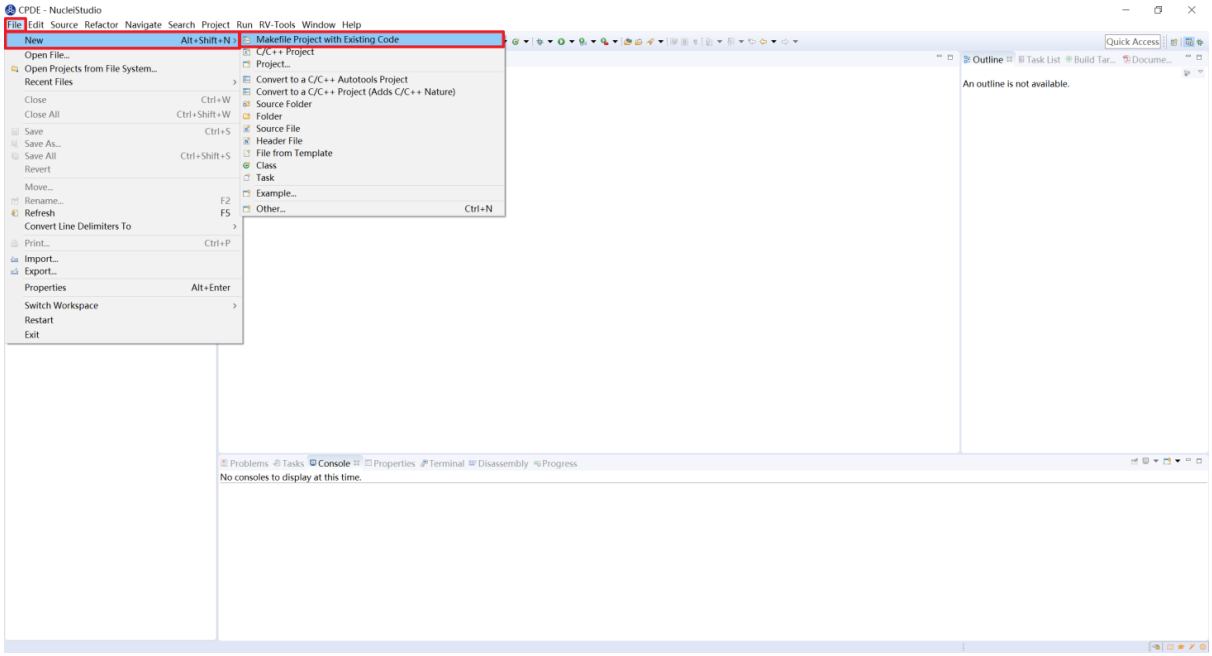


2.7.5 Create a Project Based on an Existing Makefile

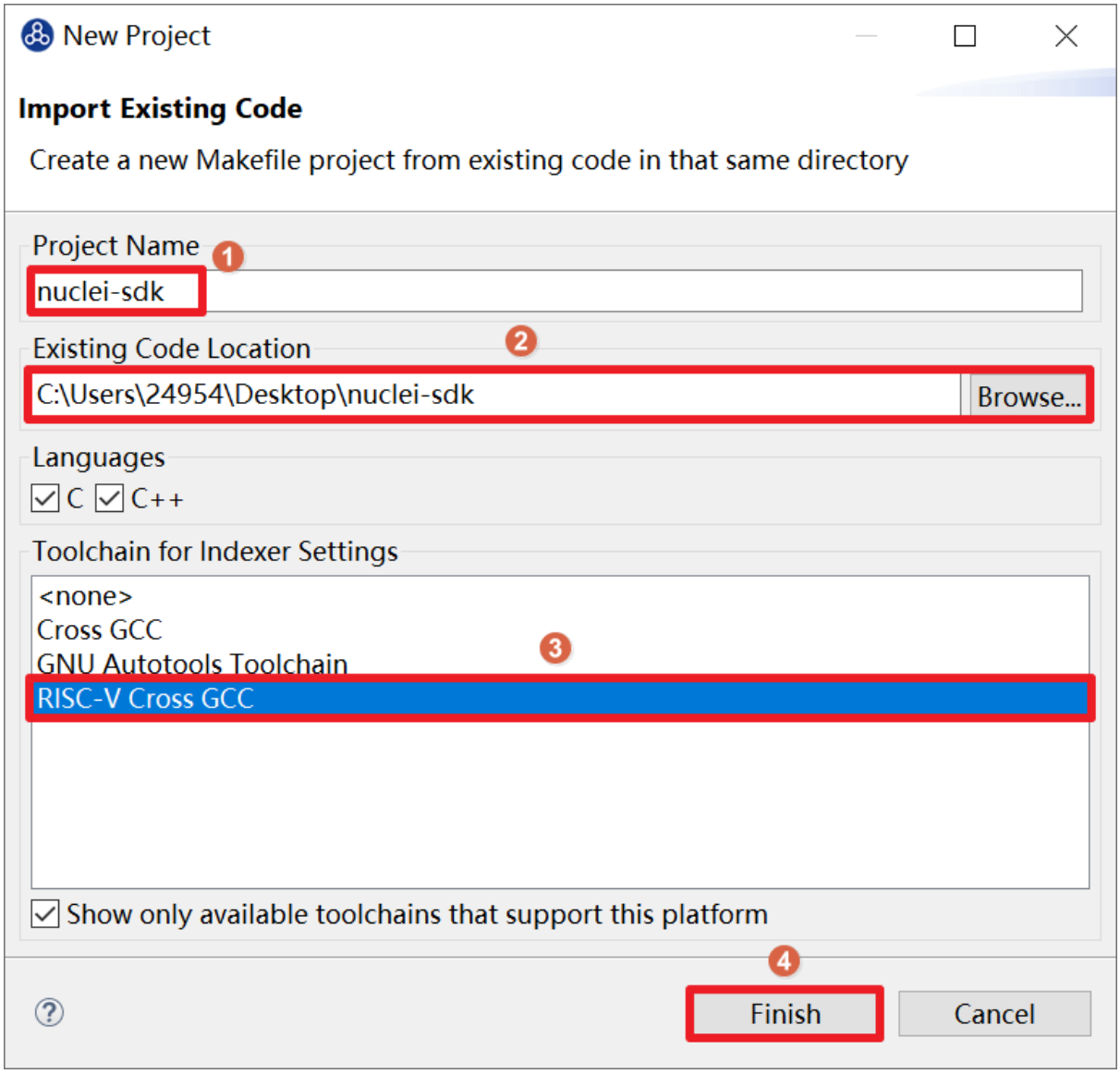
This section describes how to use an existing Makefile to create a Hello World project in Nuclei Studio IDE. The board is the Nuclei FPGA Evaluation Board and the core is N307. Download the Nuclei SDK first; the GitHub link is: https://github.com/Nuclei-Software/nuclei-sdk. In addition to creating the project, this method requires manually setting various options and paths. Here we take helloworld as an example.

Manually Create a New Project

Select `File -> New -> Makefile Project with Existing Code` in the menu bar.

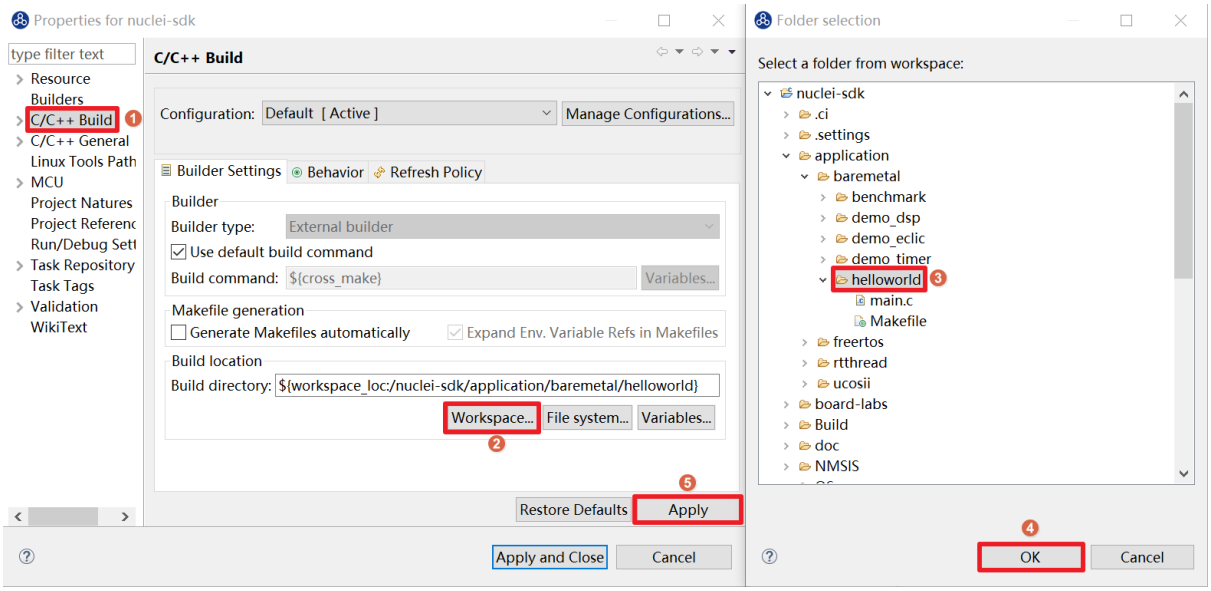


Enter the project name at icon 1; here we name it nuclei-sdk. Enter the actual path of the SDK at icon 2. Select `RISC-V Cross GCC` at icon 3. Click icon 4 to finish creating the project.

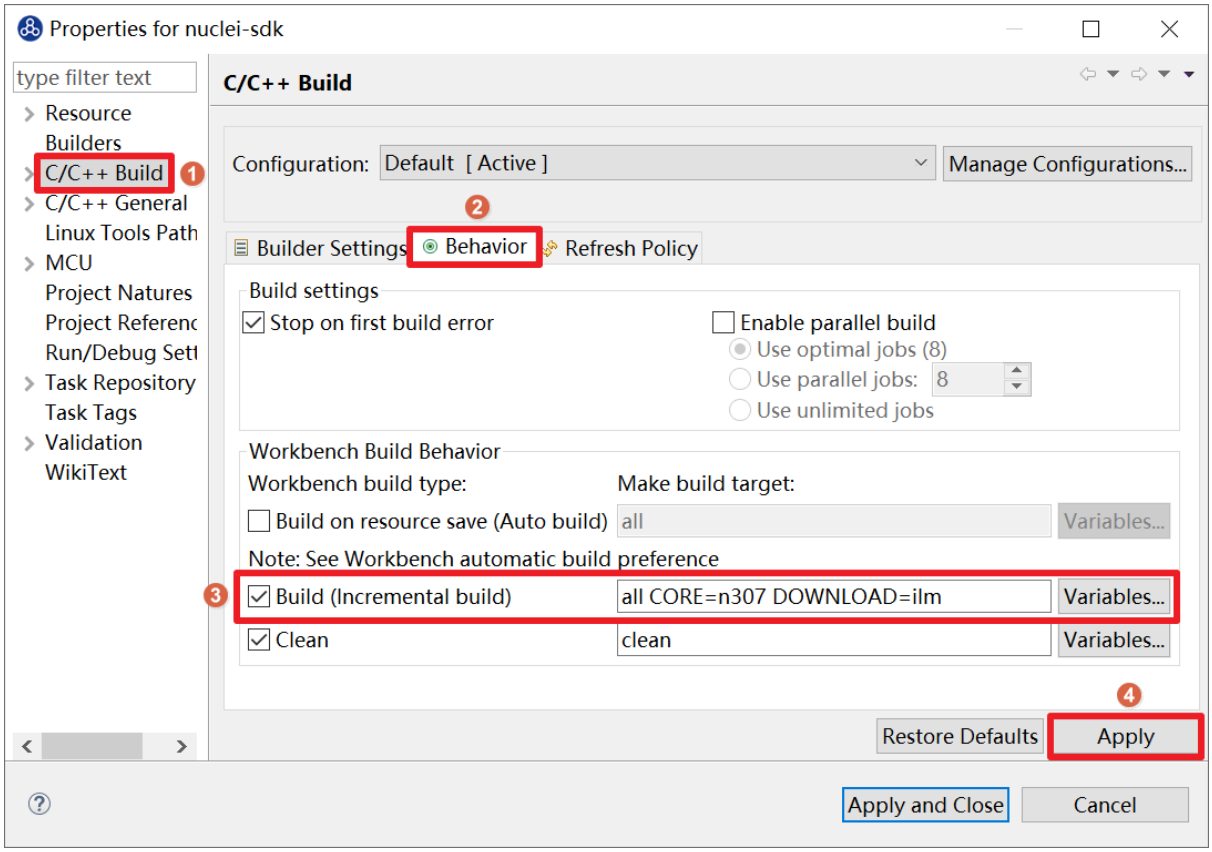


Set the Makefile Path and Build Options

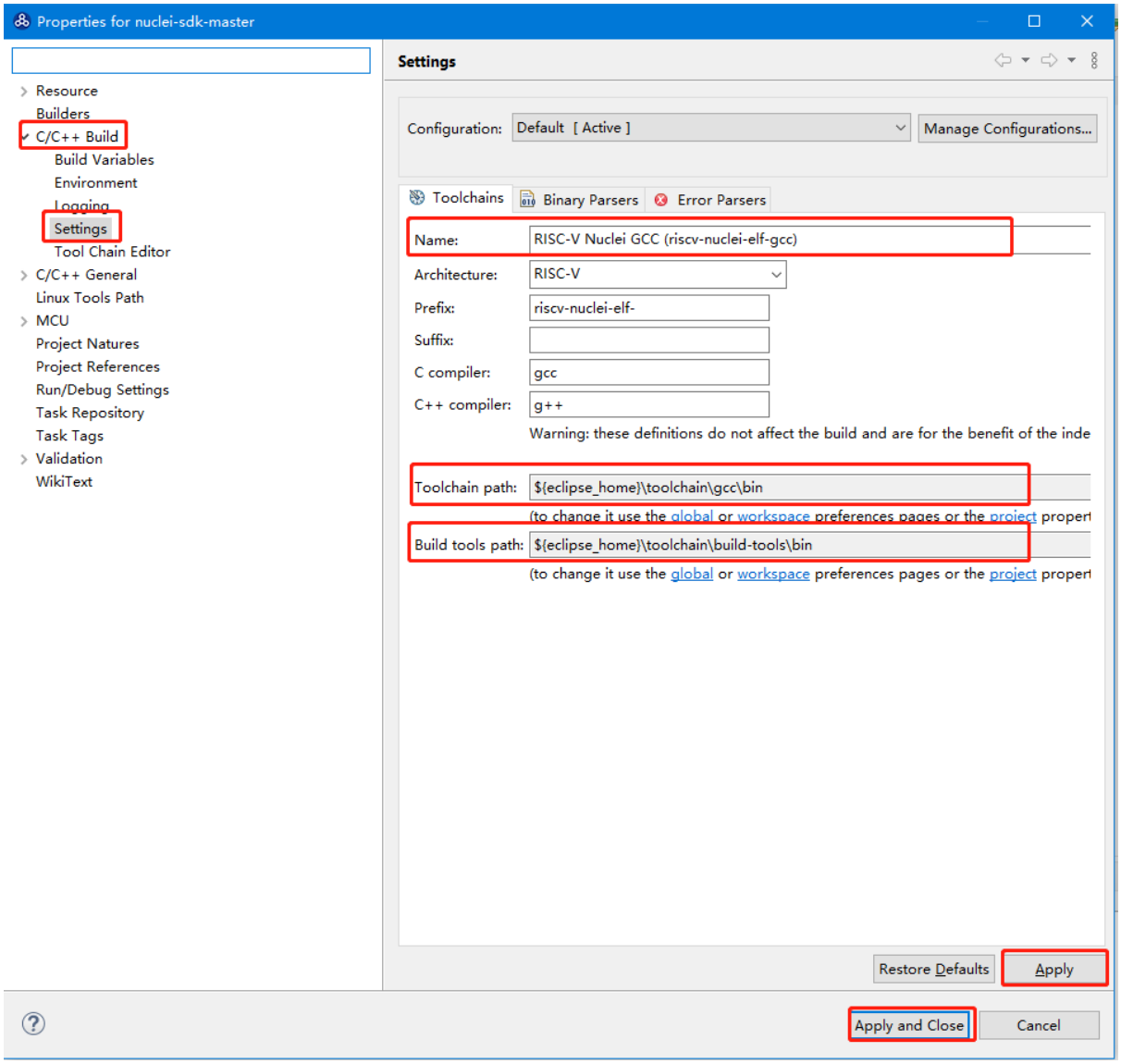
Right-click the newly created project and select `Properties` to open the settings page. Select `C/C++ Build`, and select `Workspace` in `Build Location`. In the pop-up dialog, select `application -> baremetal -> helloworld`, click `OK`, and then click `Apply` to save.



Select the `Behavior` tab in `C/C++ Build`, make sure the `Build (Incremental Build)` option is checked, and enter `all CORE=n307 DOWNLOAD=ilm`. The `CORE` option changes according to the actual core; here n307 is used as an example. The `DOWNLOAD` option can be changed to a different download mode; see Section 5.1 for details. Here, ilm mode is used as an example. Because the example uses the HummingBird Evaluation Board, SoC and Board do not need to be modified. If you use another board —taking RVSTAR as an example — add `SOC=gd32vf103 BOARD=gd32vf103v_rvstar` here, and because RVSTAR only supports FLASHXIP mode, set `DOWNLOAD` to `flashxip` and change `CORE` to `n205`. When finished, click `Apply` to save the changes.

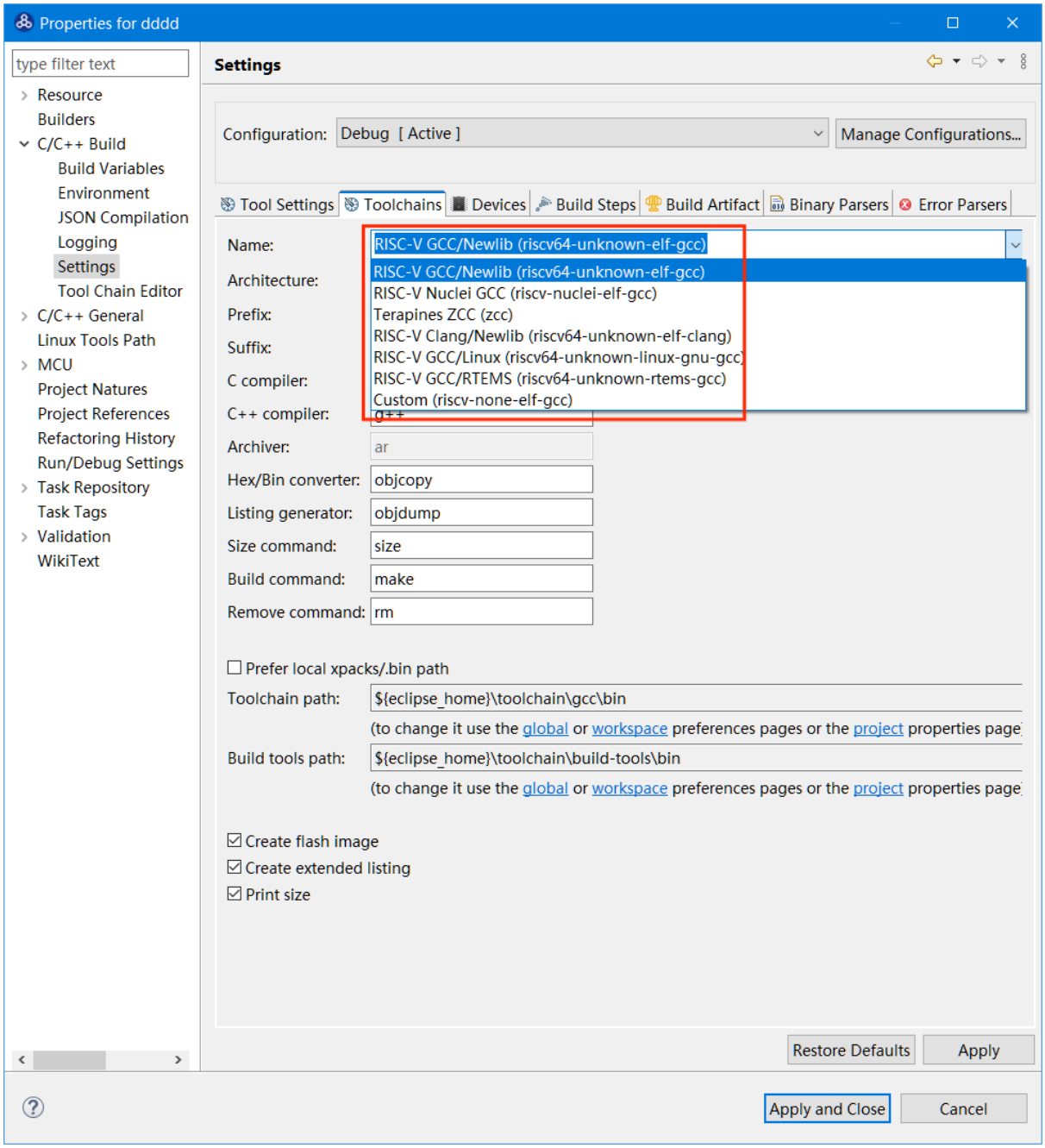


After completing the above steps, open the toolchain configuration page and click `Apply` to save the changes.



2.8 Build a Project in Nuclei Studio

Nuclei Studio supports multiple toolchains, so you can choose a different toolchain when creating a project. Nuclei Studio integrates three compilers—GCC 13, Clang 17, and ZCC Lite—which are ready to use without downloading.



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

Early versions of Nuclei Studio supported the GCC 10 compiler, which has been deprecated since Nuclei Studio 2023.10. If you need GCC 10 support, download Nuclei RISC-V Toolchain 2022.12 (gcc10) yourself and replace the contents of the `<NucleiStudio>/toolchain/gcc10` directory.

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

Nuclei Studio 2023.10 and later supports the GCC 13 compiler. The corresponding package is stored in the `<NucleiStudio>/toolchain/gcc` directory.

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

Nuclei Studio 2023.10 and later supports the Clang 17 compiler. The corresponding package is stored in the `<NucleiStudio>/toolchain/gcc` directory.

- **Terapines ZCC (zcc)**

Nuclei Studio 2024.06 supports the ZCC compiler. The corresponding package is stored in the

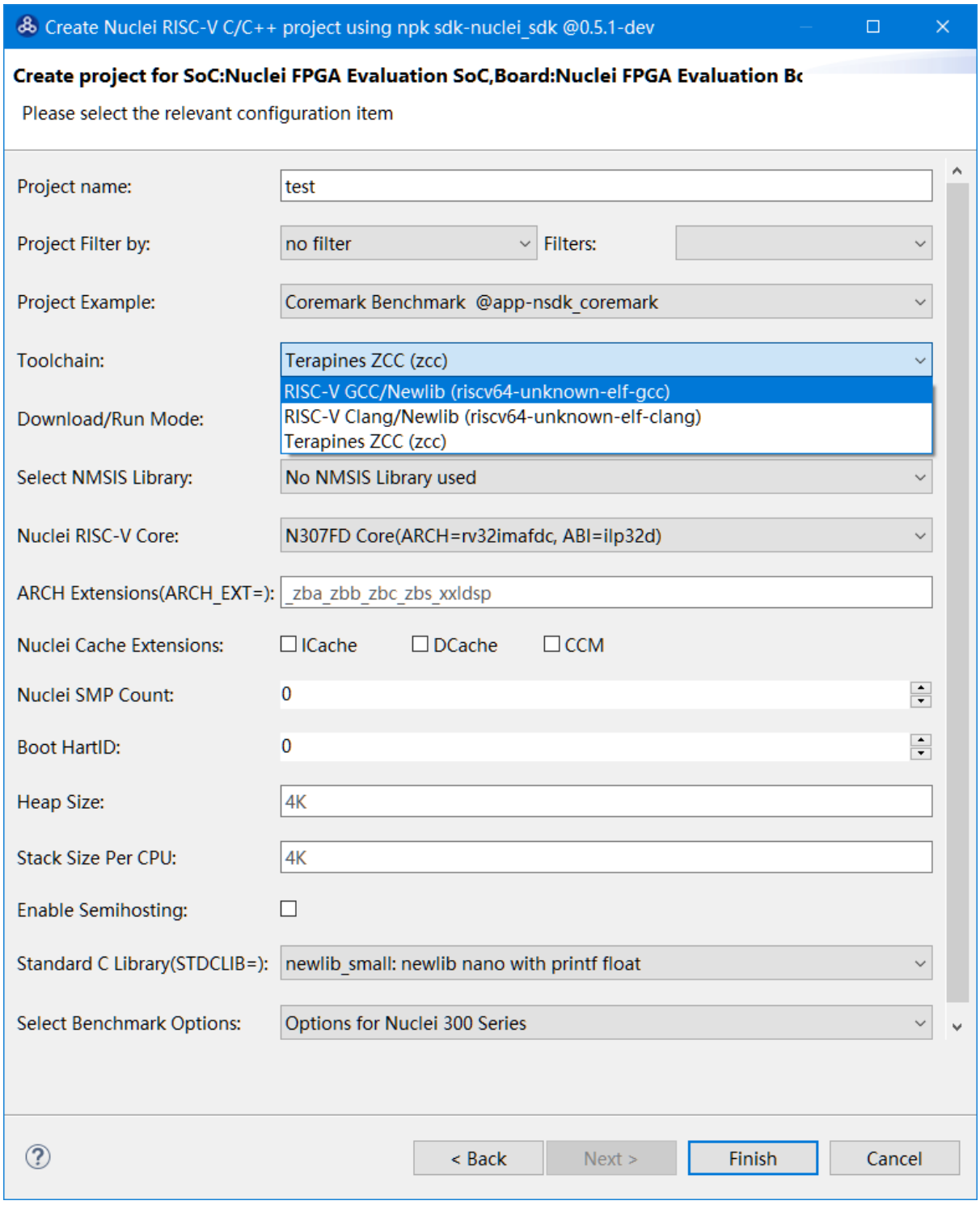
`<NucleiStudio>/toolchain/zcc` directory. Creating a Terapines ZCC toolchain project requires Nuclei SDK **later than 0.6.0** and can currently be tested through the Nuclei SDK command line.

2.8.1 Build Tools

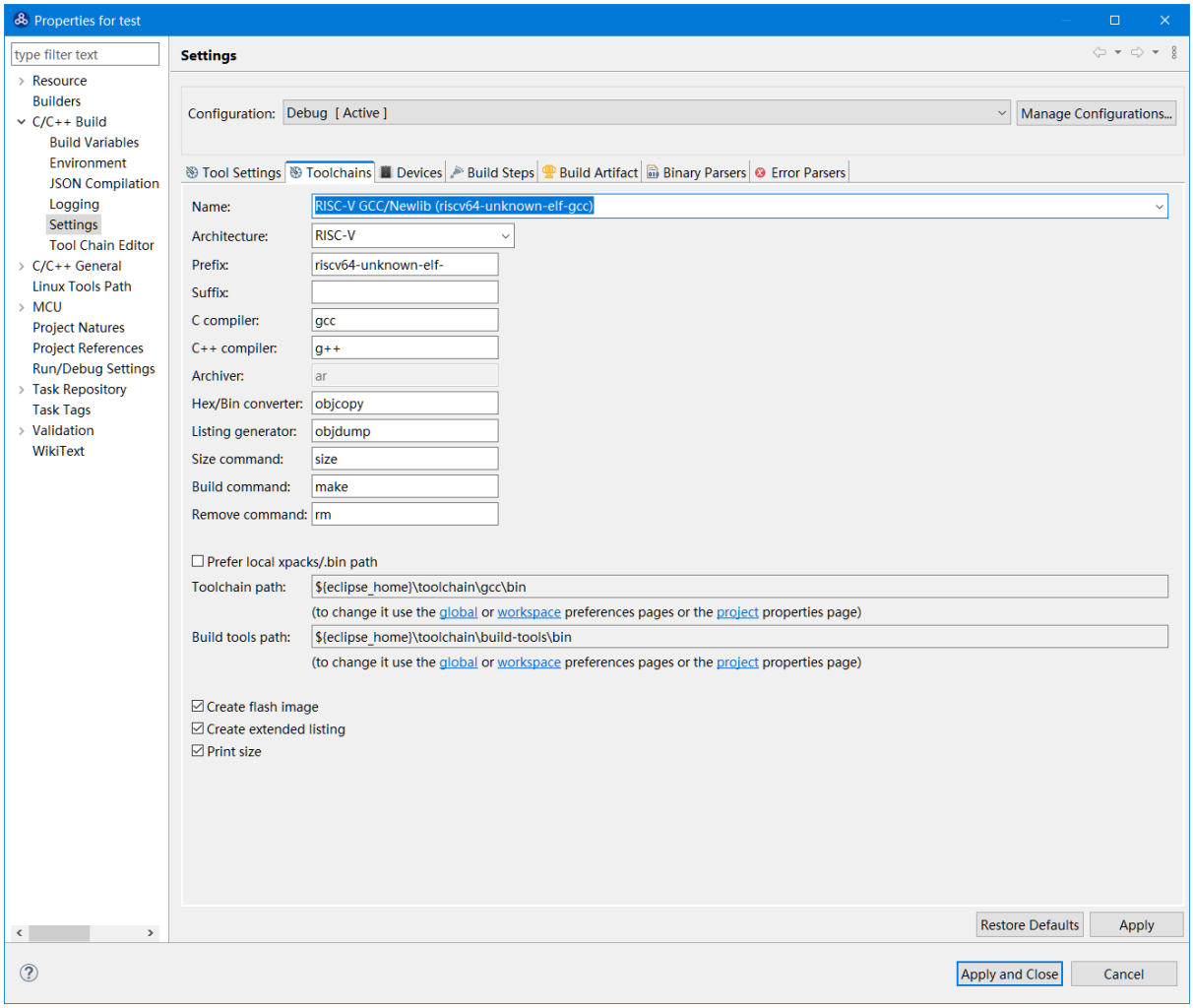
Nuclei GNU Toolchain

The GNU Toolchain is a complete set of software development tools provided by the GNU project, including a compiler, debugger, linker, library files, and other essential tools for developing and building software. Thanks to its open-source, cross-platform, highly customizable, and powerful features, the GNU Toolchain is widely used in the global developer community. Versions of Nuclei Studio before 2023.10 integrated `GCC 10`; Nuclei Studio 2023.10 and later integrates `GCC 13`.

In nuclei_sdk versions later than 0.5.0, you can select `RISC-V GCC/Newlib (riscv64-unknown-elf-gcc)` as the toolchain when creating a project to enable `GCC 13` compilation, and Nuclei Studio configures the corresponding compile options by default. For issues with `GCC 10` and `GCC 13` projects, see *Converting a Project to Support GCC 13 Using the Tool* (page 215).



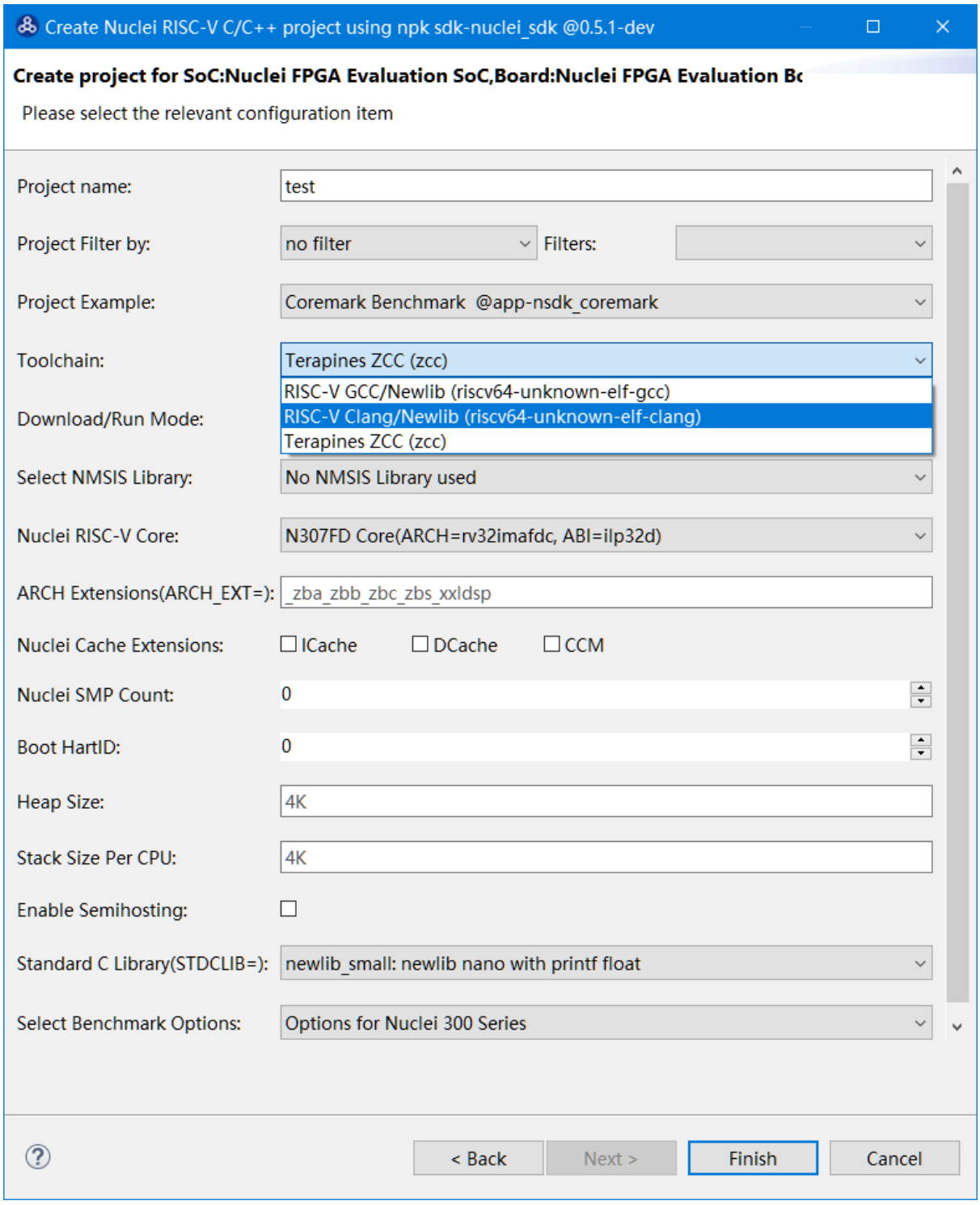
Under `right-click ->Properties->C/C++ Build->Setting->Toolchains` on the project, you can see the tools used by the project ' s toolchain and the path where the toolchain is located.



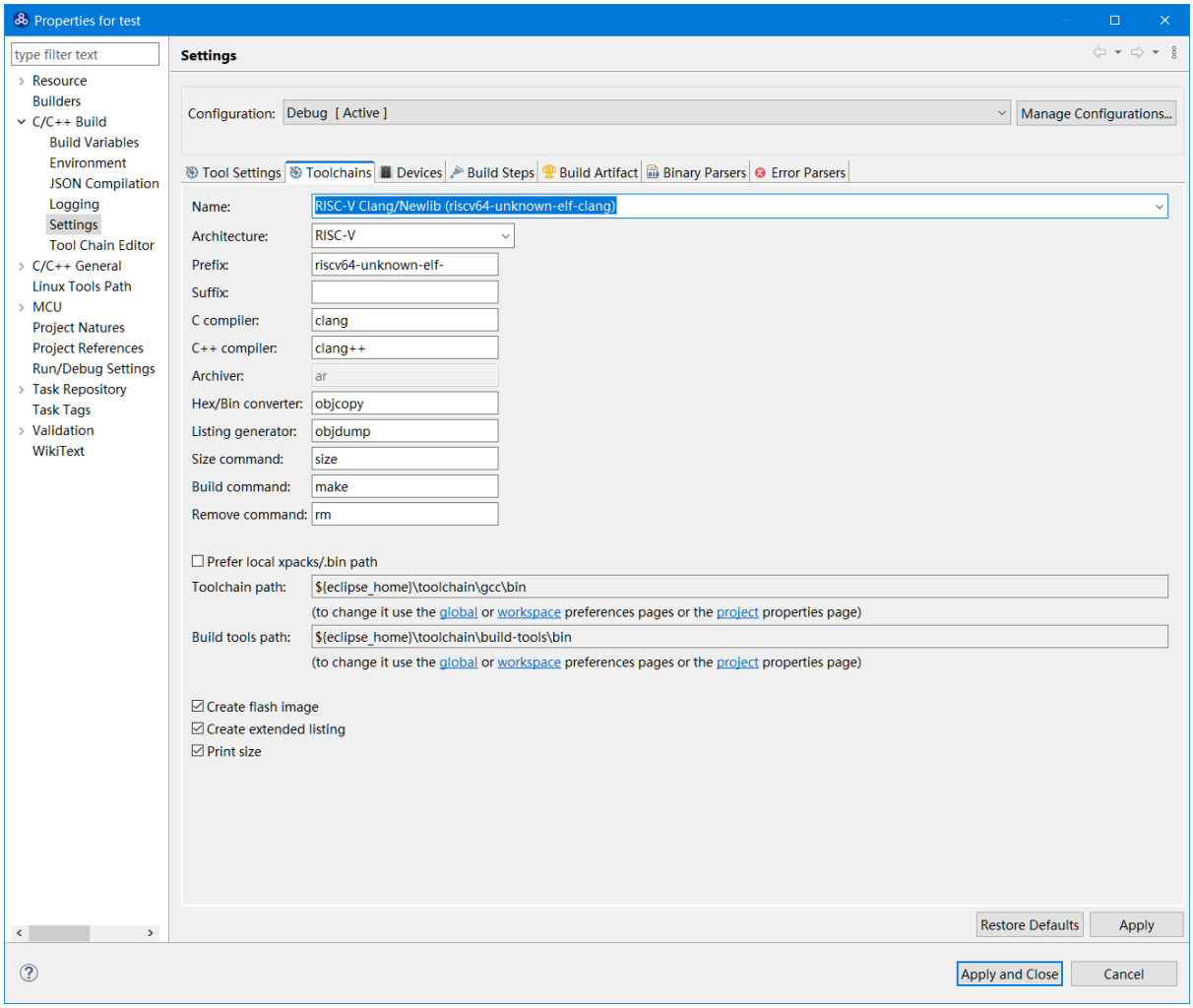
Nuclei LLVM Toolchain

The LLVM Toolchain is a complete toolchain designed for the C family of programming languages, covering the compilation and linking process from source code to executable files. Its core components include the Clang compiler, the LLVM compiler infrastructure, and related tools and libraries. Powerful, flexible, and extensible, the LLVM Toolchain adopts advanced compilation technologies and design concepts to provide developers with efficient compilation and build solutions. Nuclei Studio 2023.10 and later integrate the LLVM Toolchain.

In nuclei_sdk versions later than 0.5.0, you can select `RISC-V Clang/Newlib (riscv64-unknown-elf-clang)` as the toolchain when creating a project to enable `Clang 17` compilation, and Nuclei Studio configures the corresponding compile options by default.



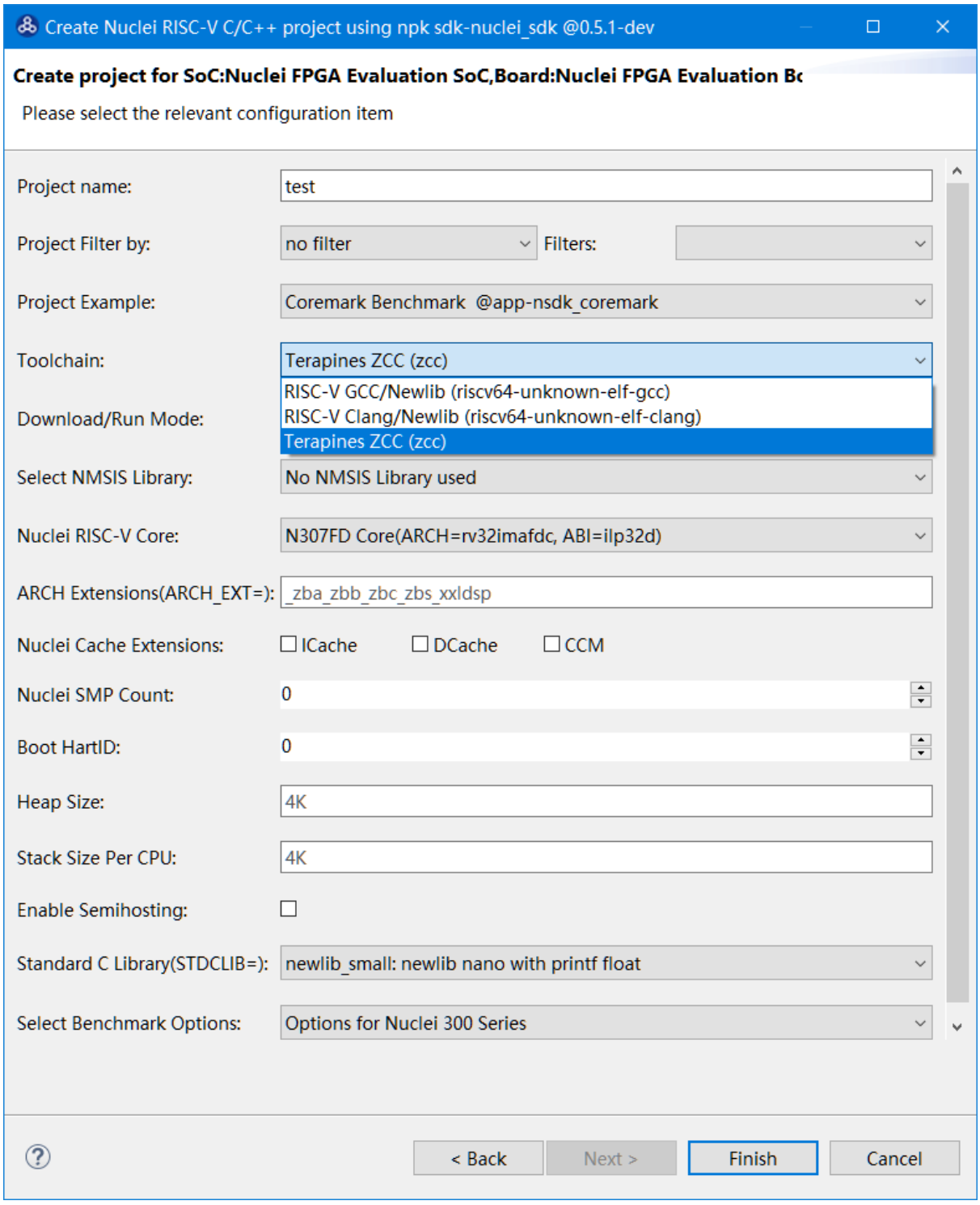
Under `right-click ->Properties->C/C++ Build->Setting->Toolchains` on the project, you can see the tools used by the project ' s toolchain and the path where the toolchain is located.



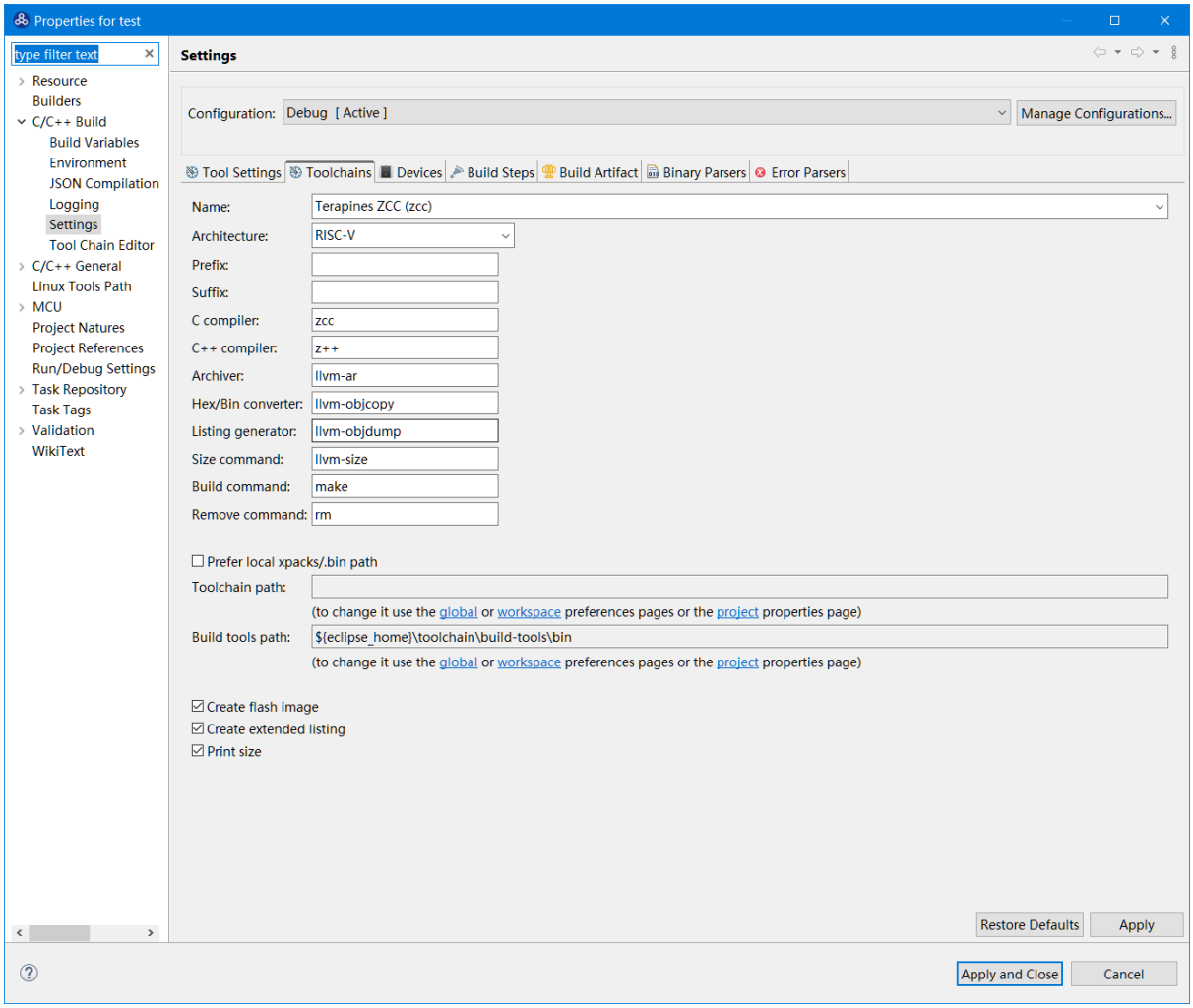
Terapines ZCC

Terapines ZCC is a high-performance RISC-V compiler developed by Terapines Technology. Nuclei Studio 2024.06 supports Terapines ZCC and **integrates the ZCC Lite version of the toolchain**. If you need an update, download Terapines ZCC yourself and replace the contents of the `<NucleiStudio>/toolchain/zcc` directory. You can create a Terapines ZCC project directly in Nuclei Studio and build it with Terapines ZCC.

For Terapines ZCC, see: https://products.terapines.com/downloads



Under `right-click ->Properties->C/C++ Build->Setting->Toolchains` on the project, you can see the tools used by the project ' s toolchain and the path where the toolchain is located. Here you can configure the path of a ZCC compiler you downloaded yourself.

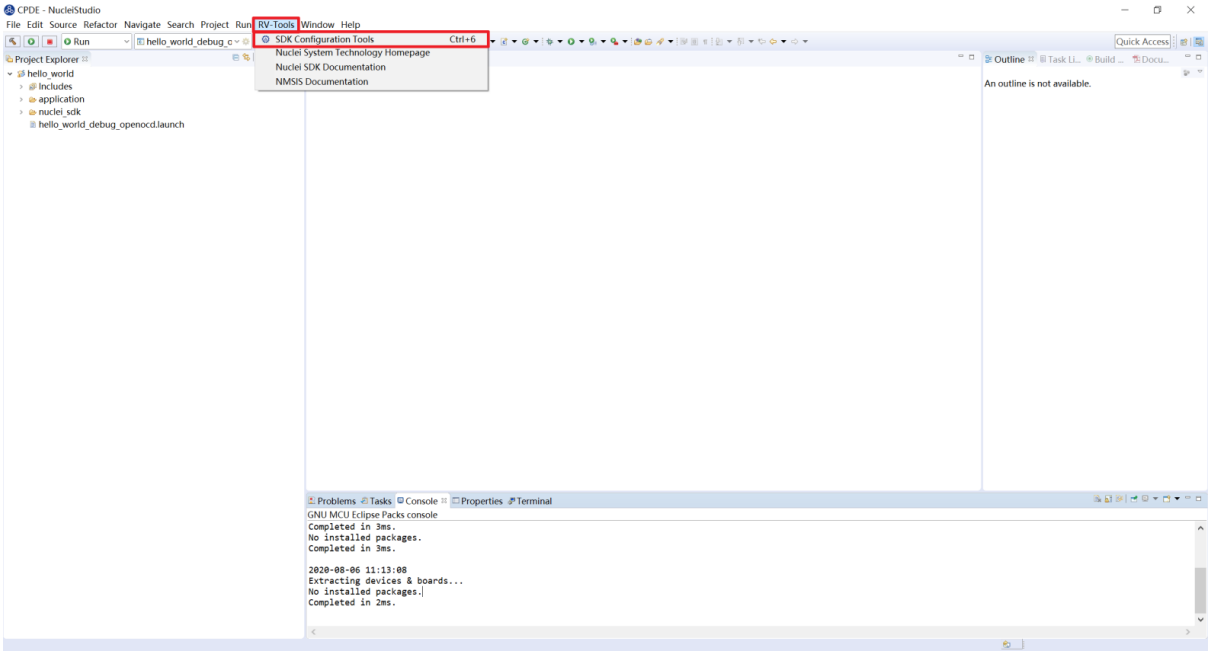


2.8.2 Nuclei SDK Project Settings Tool

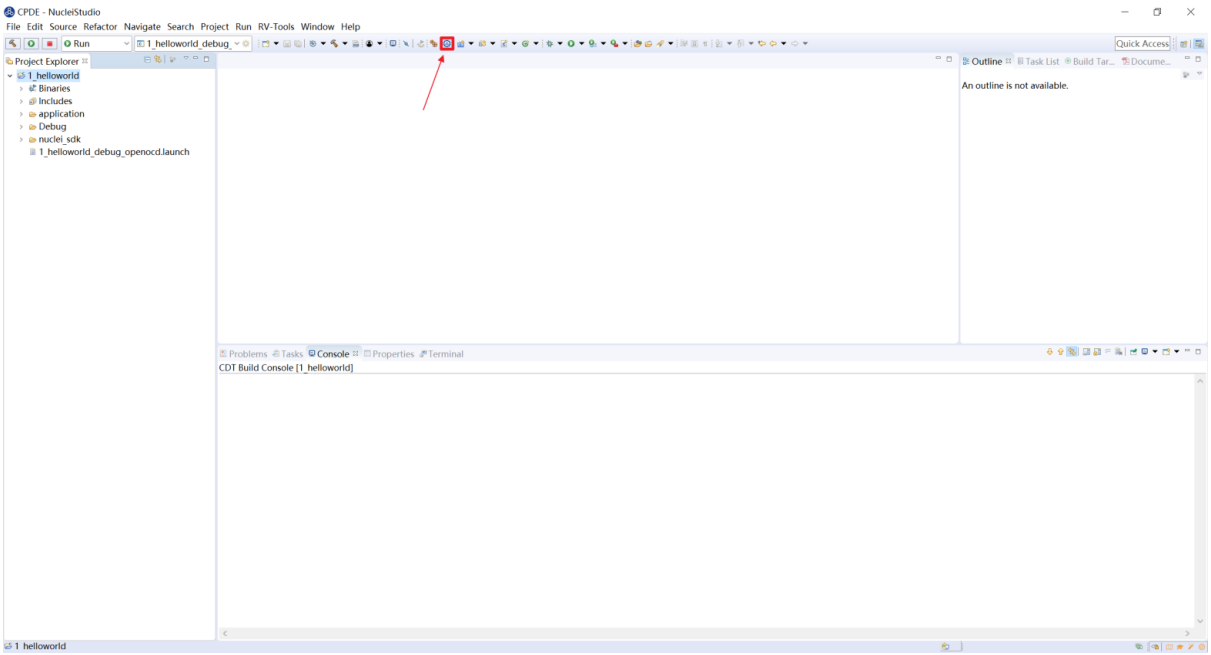
In Nuclei Studio, you can use the Nuclei SDK project settings tool to modify the compile and link options of a project created by the Nuclei SDK Project Wizard with one click.

Note: This tool currently only supports Nuclei SDK, HBird SDK, and Nuclei Subsystem SDK. It does not support projects created manually without a template, Makefile-based projects, or projects created and maintained by yourself.

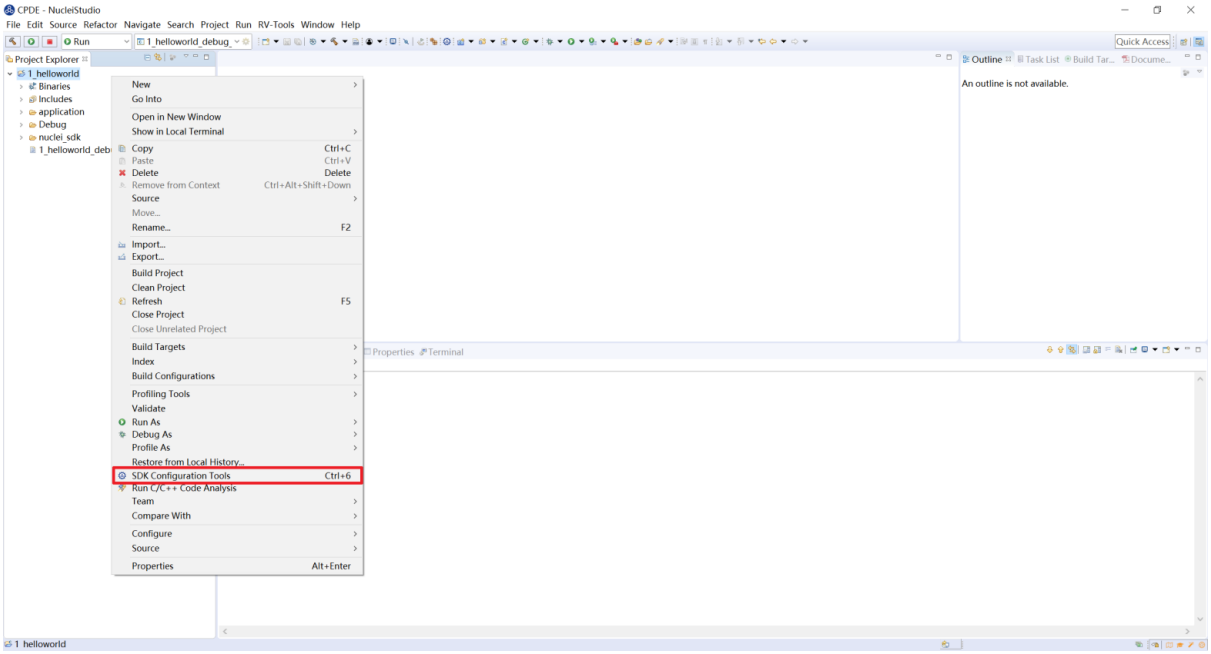
Click to select the project to modify, then open `RV-Tools>SDK Configuration Tools` as shown in Figure 6-1 to open the dialog for modifying compile options. **In versions after 2023.10, the Nuclei Settings page is opened directly**. Alternatively, click the project and then click the project settings tool icon in the toolbar.



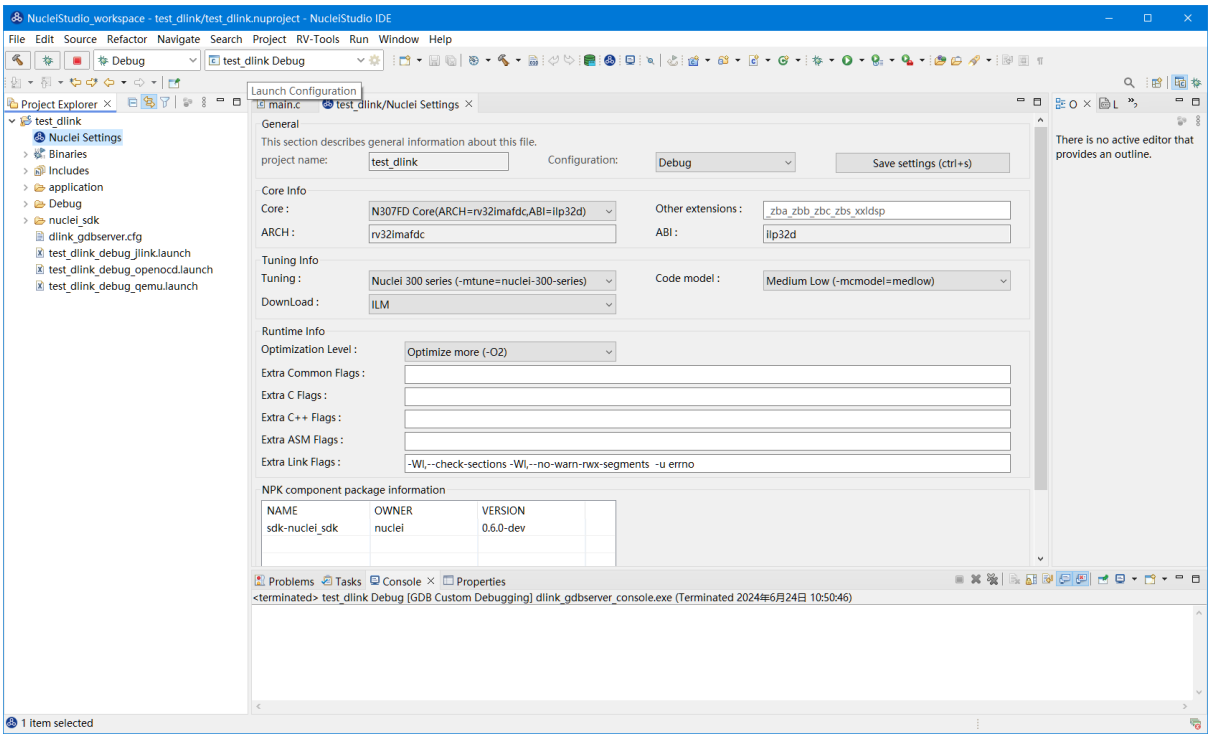
Alternatively, click the project and press `ctrl+6` on the keyboard.



Alternatively, right-click the project and select SDK Configuration Tools from the context menu.



The function of each option in SDK Configuration Tools is described below:



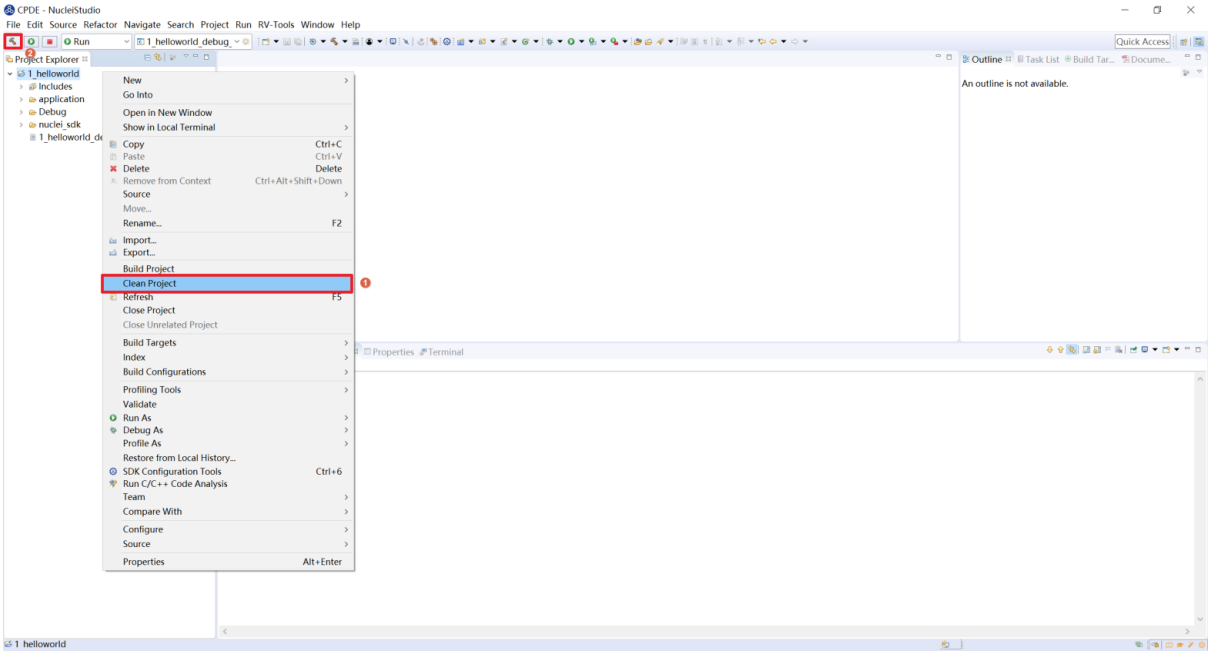
- projectName is the name of the currently selected project.
- Core is the core corresponding to the current project. Because the tool infers the core from the ARCH and ABI options, and different cores may share the same ARCH and ABI options, the display may deviate slightly. As long as ARCH and ABI are correct, this is not a problem. Use this option to quickly switch core options.
- Four checkboxes—`Bitmanipulation Extension(RVB)`, `Cryptography Extension(RVK)`, `Packed SIMD/DSP Extension`, and `Vector Extension(RVV)`—are used to select the corresponding extension instruction sets `(B/K/P/V)`.

Note: This feature was removed in version 2023.10. Use the `Other Extensions` input box to specify

additional extensions.

- ARCH corresponds to the arch option of the current project and is automatically combined according to Core and the checkboxes.
- ABI corresponds to the abi option of the current project.
- Tuning is the gcc option optimized for different processor levels. Selecting Core automatically selects the correct Tuning option; adjusting it yourself is not recommended.
- Code Model is automatically set to `Medium Low` for RV32 processors and `Medium High` for RV64 processors. Selecting Core automatically selects the appropriate Code Model; RV64 processors must use `Medium High`.
- Download corresponds to the download mode of the current project. You can switch between download modes. Currently only the Nuclei FPGA evaluation board supports switching download modes; RVSTAR has only the `FLASHXIP` mode. Switching to flash mode additionally defines the `VECTOR_TABLE_REMAPPED` macro; other modes do not define this macro.
- `Select C Runtime Library` corresponds to the standard C library used. **This feature was removed in version 2023.10**. If the project was created with Newlib, only the newlib version can be switched here; if the project was created with `Nuclei C Runtime Library` (*libncrt* (page 473)), only the libncrt version can be switched here.
- `Optimization Level` corresponds to the compile optimization level.
- `Extra Common Flags` corresponds to additional common compile options. You can add extra common compile options.
- `Extra C Flags` corresponds to additional C compile options. You can add extra C compile options.
- `Extra C++ Flags` corresponds to additional C++ compile options. You can add extra C++ compile options.
- `Extra ASM Flags` corresponds to additional assembly compile options. You can add extra assembly compile options.
- `Extra Link Flags` corresponds to additional link options. If this option already contains default options, add your new options at the beginning or end, separated by a space character.

Modify the options above as needed. This example changes the optimization level to `-Os` to reduce the size of the generated executable file. Click `save` to apply the compile options with one click. After `save`, you must first clean the project: right-click the project, select `Clean Project` from the context menu, and then click the hammer icon to rebuild the project with the modified compile options.



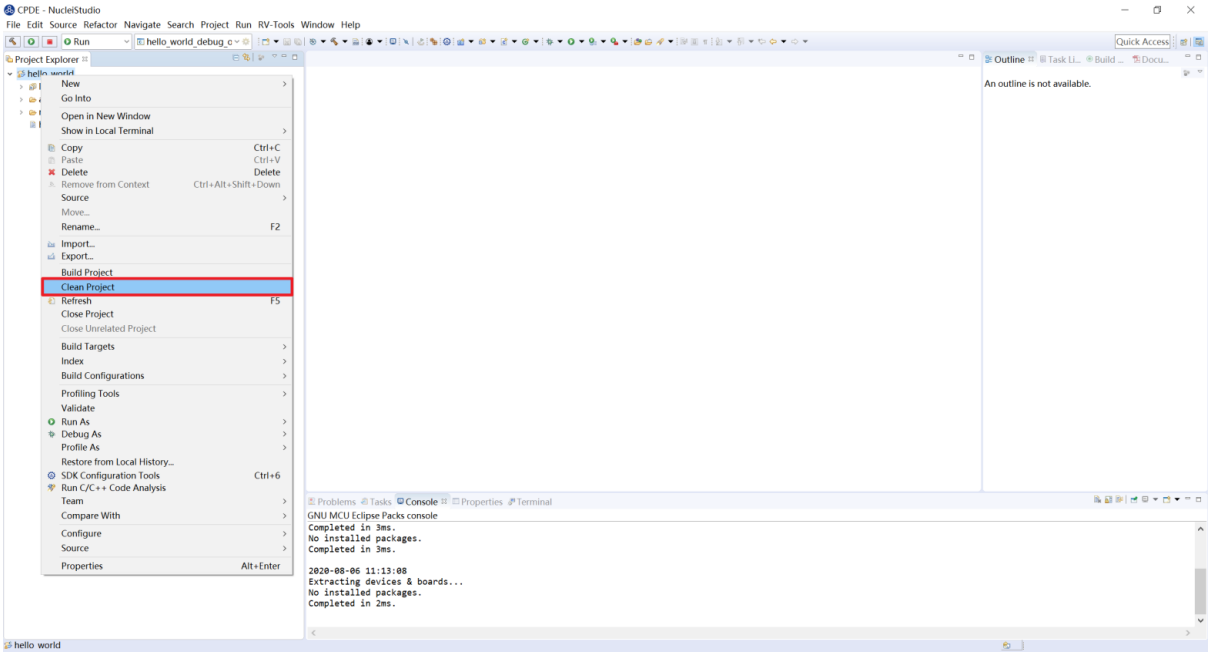
Changes made in SDK Configuration Tools do not affect the Debug Configuration options. Therefore, if you switch the Core, you must also modify the corresponding debug configuration (OpenOCD/QEMU/J-Link) manually.

Note that if you switch the project from 32-bit to 64-bit, you must open the debug settings page and change `set arch riscv:rv32` in `command` to `set arch riscv:rv64`. When switching back from 64-bit to 32-bit, change this parameter back to the corresponding value.

2.8.3 Build the Hello World Project in Nuclei Studio

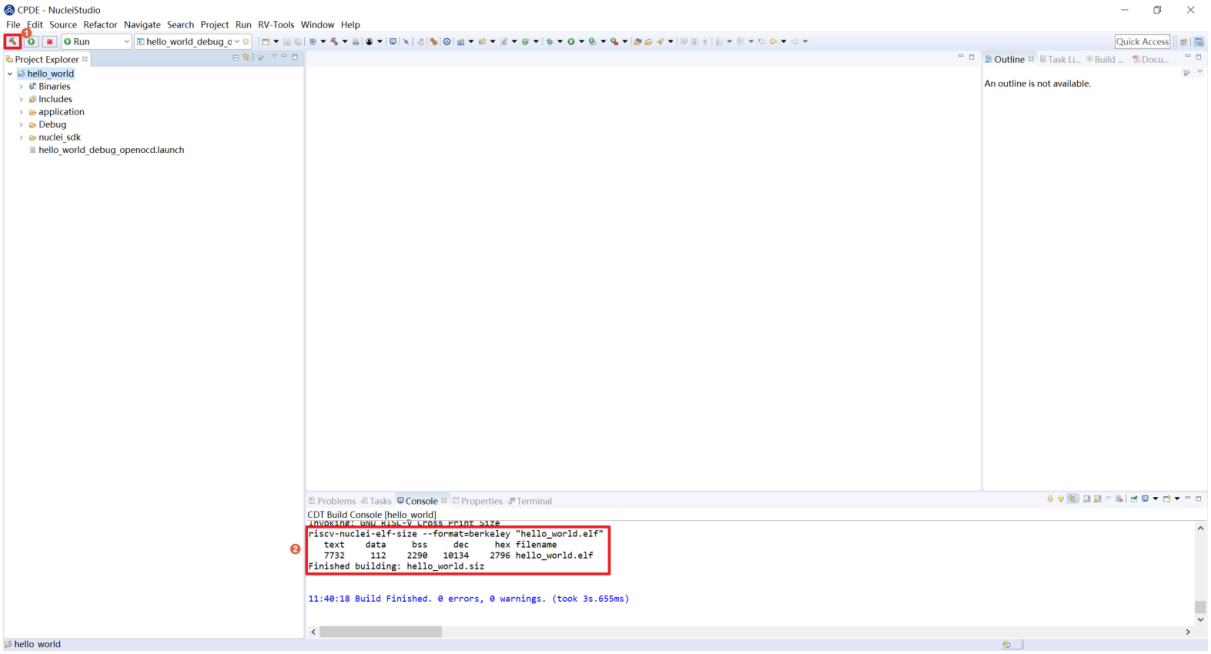
To build the Hello World project in Nuclei Studio, follow these steps:

Before building, clean the project first. In the `Project Explorer` view, select the `hello_world` project, right-click, and select `Clean Project`.



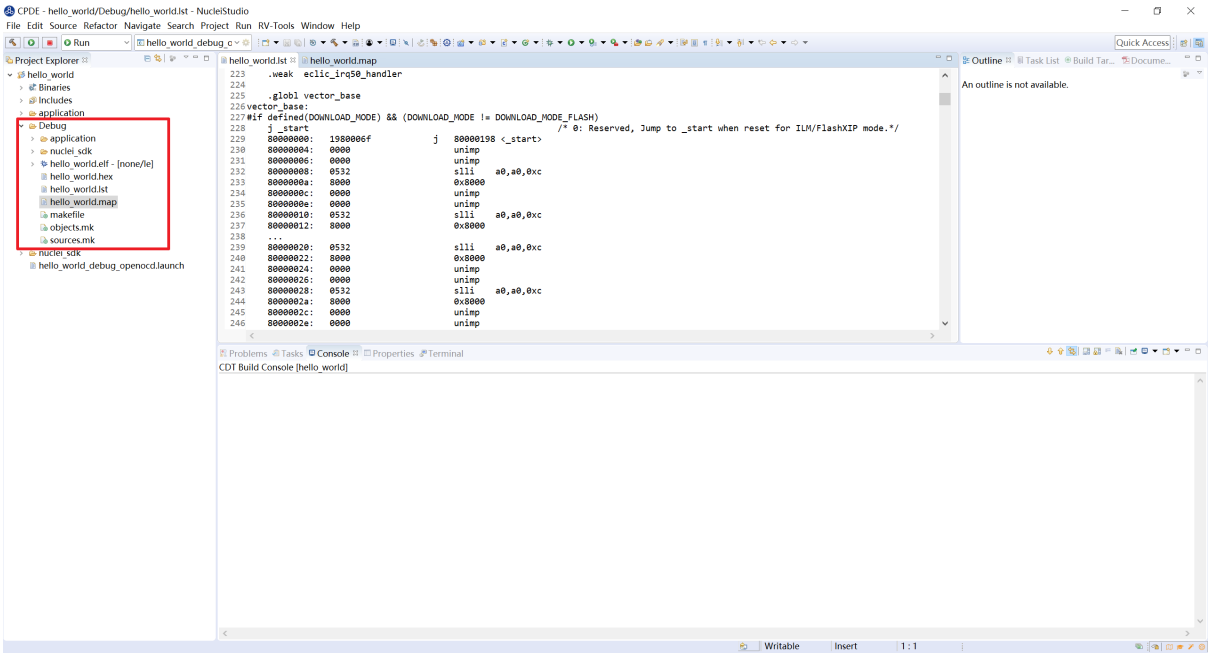
Click the hammer button on the Launch Bar to start building the project. If the build succeeds, you can see the code size of the generated executable file, including the text, data, and bss sections, as well as the total size in decimal

and hexadecimal. For a project created with the Makefile method, select `Build Project` in the context menu to build it.



After a successful build, a Debug folder is added. The function of each file is as follows:

- `hello_world.elf` is the generated executable file.
- `hello_world.hex` is the generated Hex file.
- `hello_world.lst` is the generated list file, where you can see the disassembly and basic code distribution information.
- `hello_world.map` is the generated map file, where you can see the generated code distribution in detail.

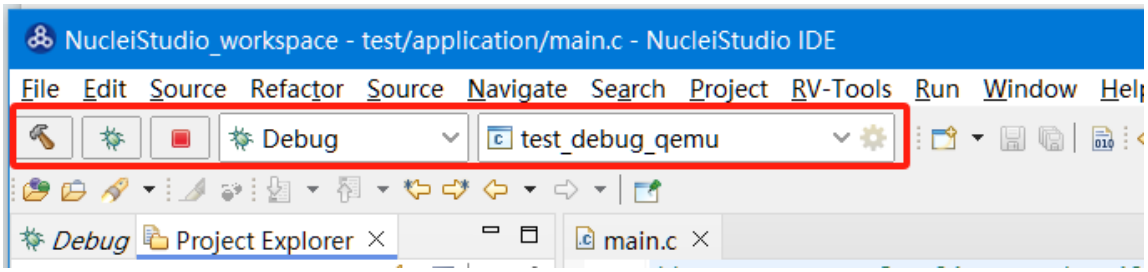


2.9 Debugging and Running Projects in Nuclei Studio

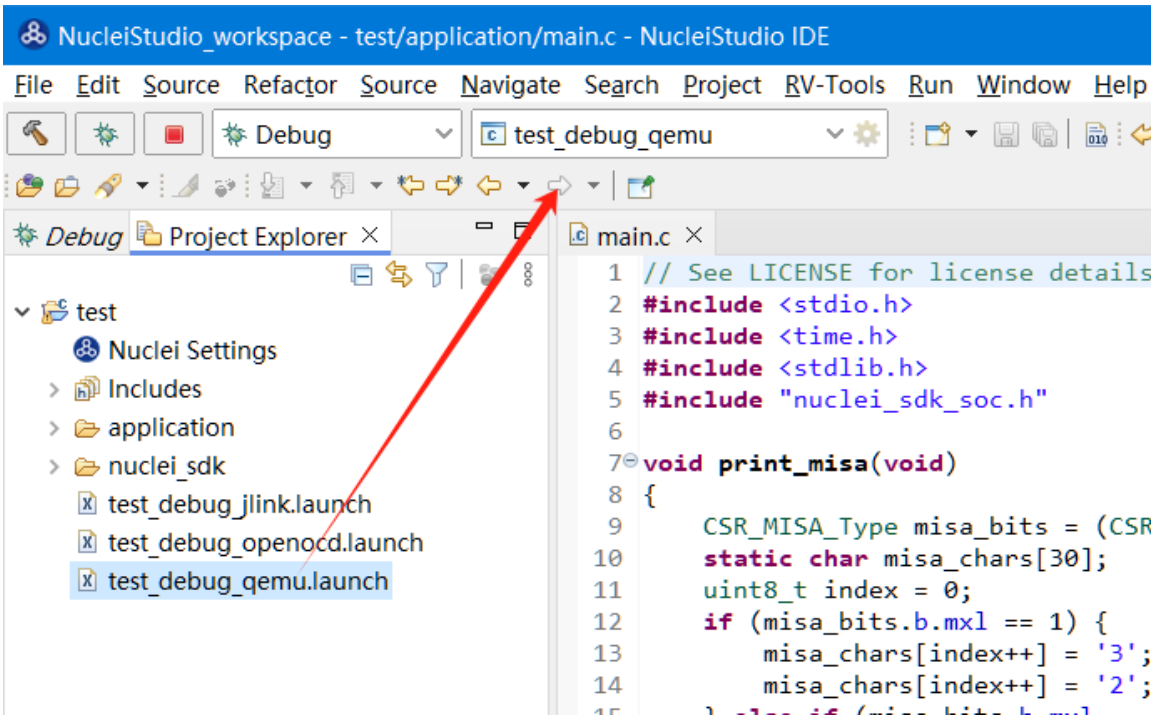
Before debugging a project, first verify that the current hardware setup is correct; see *Hardware check before debugging a project in Nuclei Studio* (page 384).

2.9.1 Debug Mode Management

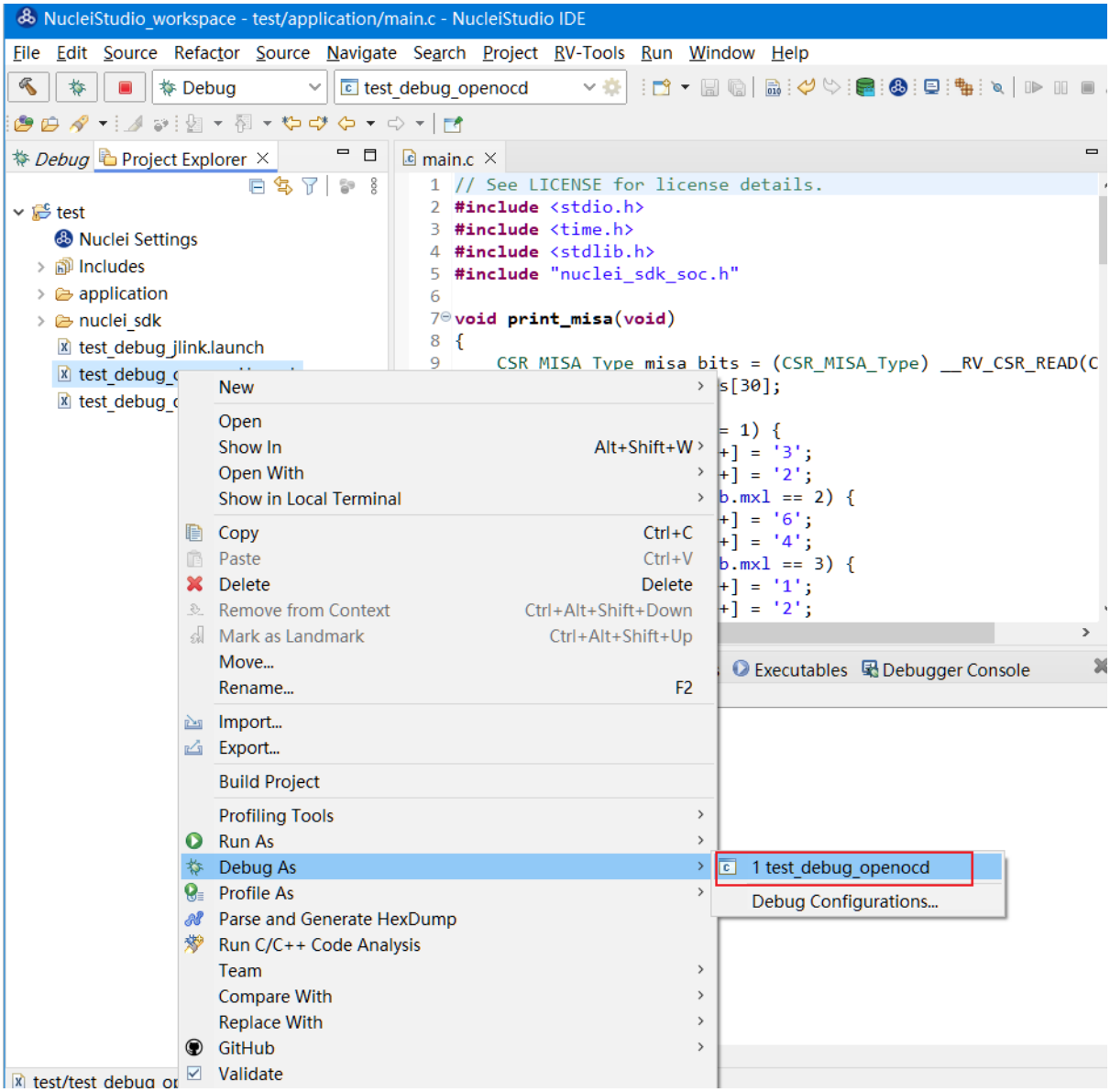
In Nuclei Studio, the Launch Bar manages different debuggers. By default, Nuclei Studio generates the corresponding `*.launch` debug files for OpenOCD, J-Link, and QEMU, and adds any detected `*.launch` file to the Launch Bar for management. You can switch between debug modes in the following three ways.



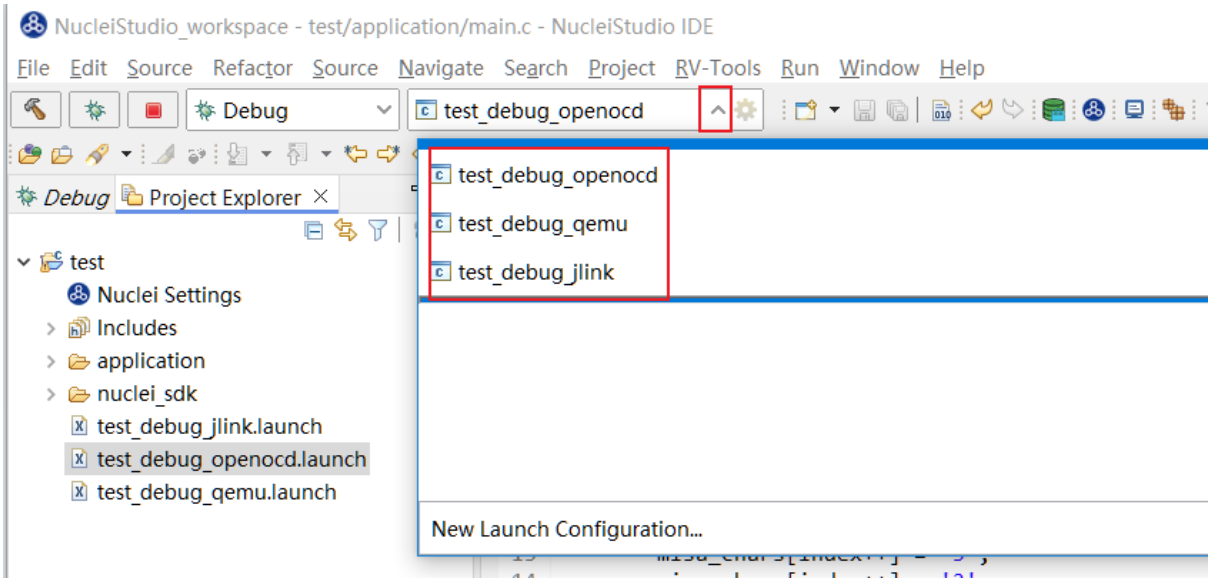
Click a `*.launch` file in the project to switch debug modes. Nuclei Studio sets the clicked file as the selected file in the Launch Bar; you can then perform Run/Debug operations from the Launch Bar.



Expand the project, right-click the `*.launch` file, select `Debug As/Run As` from the context menu, and then click the corresponding mode in the submenu to start the `Run/Debug` operation.



You can also switch between debug modes through the drop-down box in the Launch Bar: click the expand button, select the corresponding debug mode, and perform the `Run/Debug` operation.



2.9.2 Debugging and Running a Project with the HummingBird Debugger and OpenOCD

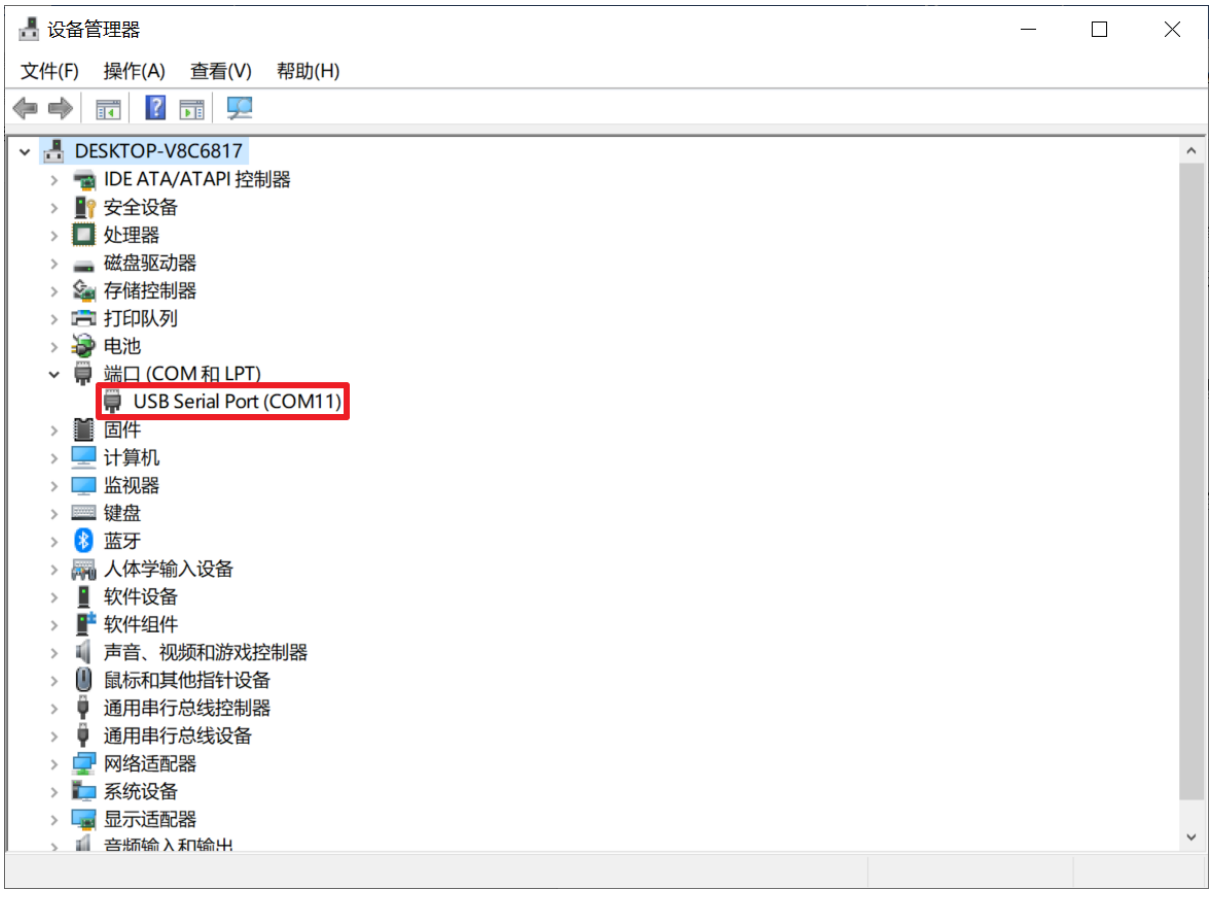
Installing the HummingBird Debugger Driver

Note: Starting with version 2021.02, Nuclei Studio ' s OpenOCD is driver-free on Windows. Windows users on this version or later can skip this section; Linux users must still configure the driver. Users on earlier versions must install the driver as described below.

Installing the Driver on Windows

Once the program builds successfully, it can be downloaded to the FPGA prototype board and run. First, connect the prototype board to the host PC as follows.

Plug one end of the HummingBird debugger into a USB port on the host PC and the other end into the prototype board. Because the HummingBird debugger also `converts the UART output of the prototype board to USB`, once the debugger is recognized by the host PC (that is, once the driver is installed successfully), the host detects it as a COM serial port. You can find the COM port number (for example, COM11) under the `Ports (COM & LPT)` category in the host PC ' s Device Manager. This serial port serves as the printf output interface for the program running on the prototype board.



Note: If Windows on the host PC cannot recognize the HummingBird debugger ' s USB connection, you need to install the driver. Download the driver from https://www.nucleisys.com/developboard.php#debuggerkit and install it.

Note: Before downloading through the `HummingBird debugger`, make sure it is correctly recognized by Windows: the `HBird-Driver.exe` driver must be installed as described in this section, and the COM port number must appear in Device Manager.

Installing the Driver on Linux

To install the driver on Linux:

- 1: Connect the development board to the Linux host and make sure the USB device is recognized. If you are using a virtual machine, make sure the board is connected to the virtual machine.



- 2: Run `lsusb` in the console to view the device information; for example:

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

- 3: In the console, run `sudo vi /etc/udev/rules.d/99-openocd.rules` to open the 99-openocd.rules file, enter the following content, then save and exit.

```
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: Disconnect the debugger and reconnect it.
- 5: Run `ls /dev/ttyUSB*` to view the `ttyUSB` information; for example:

```
/dev/ttyUSB0

/dev/ttyUSB1
```

- 6: Run `ls -l /dev/ttyUSB1` to view the group information; for example:

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

As shown, `ttyUSB1` now belongs to the `plugdev` group. Next, add your user to the `plugdev` group (the group name may differ between environments; adjust it to your actual situation). Run `whoami` to view the current user name, denoted below as `<your_user_name>`.

- 7: Run `sudo usermod -a -G plugdev <your_user_name>` to add yourself to the plugdev group. You must reboot or log out for the change to take effect.
- 8: Run `groups` to confirm that the current user belongs to the `plugdev` group. If `plugdev` appears in the output, the user was added successfully; otherwise, try rebooting.
- 9: Check that all GCC dependencies are present. If any dependency is missing, run `sudo apt install libncursesw5 libtinfo5` to install it.

```
cd NucleiStudio/toolchain/gcc/bin/

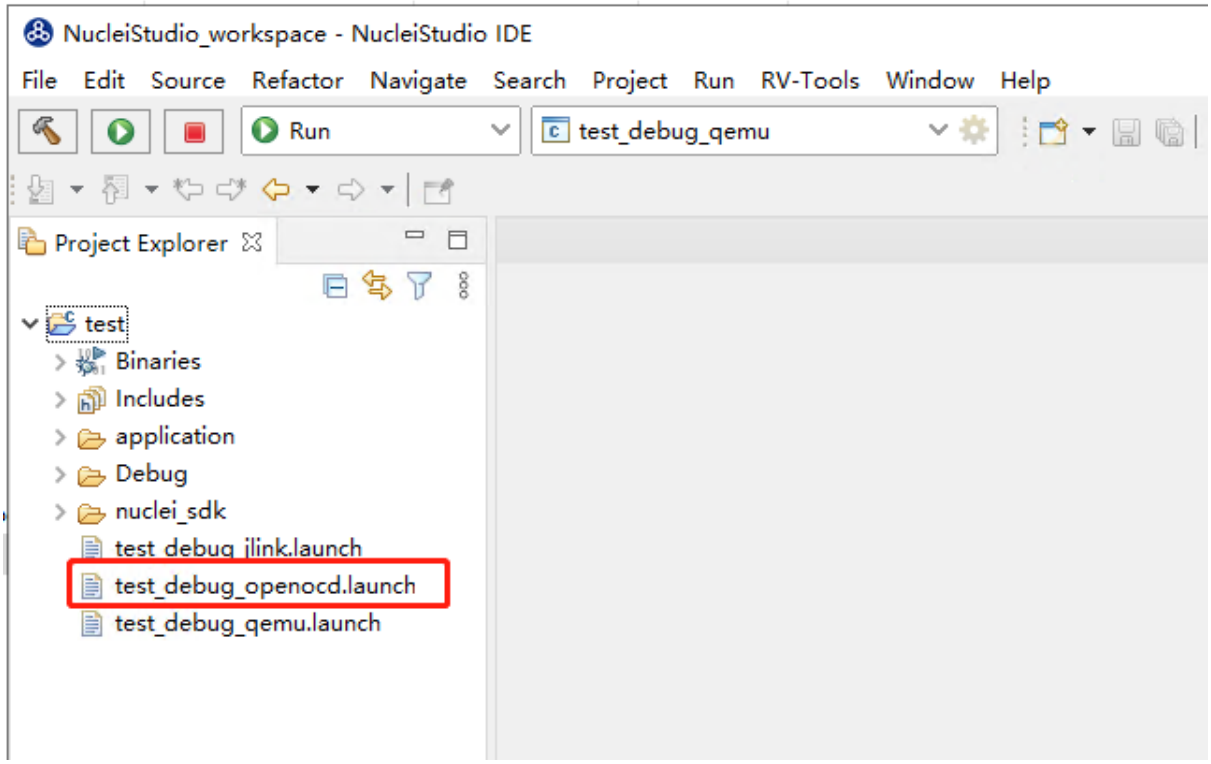
# for gcc10, try the command below
ldd ./riscv-nuclei-elf-gdb
# for gcc13/gcc14 and later versions
ldd ./riscv64-unknown-elf-gdb
```

```
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)
```

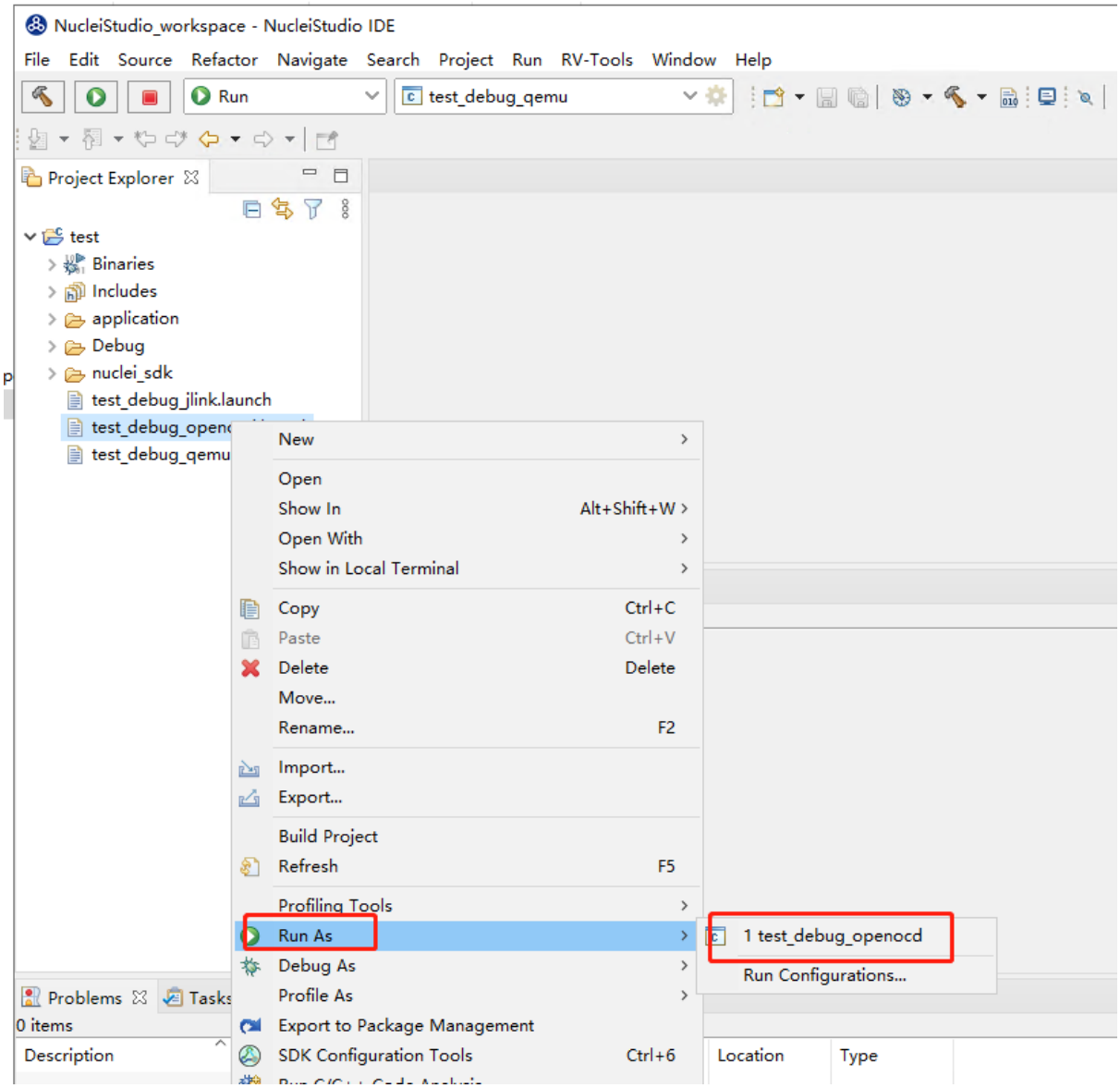
Debug Configuration

Using the Debug Configuration Generated by Nuclei Studio

To simplify debugging, Nuclei Studio generates Debug Configuration launch files by default when a project is created, based on the Nuclei Package (NPK) configuration.

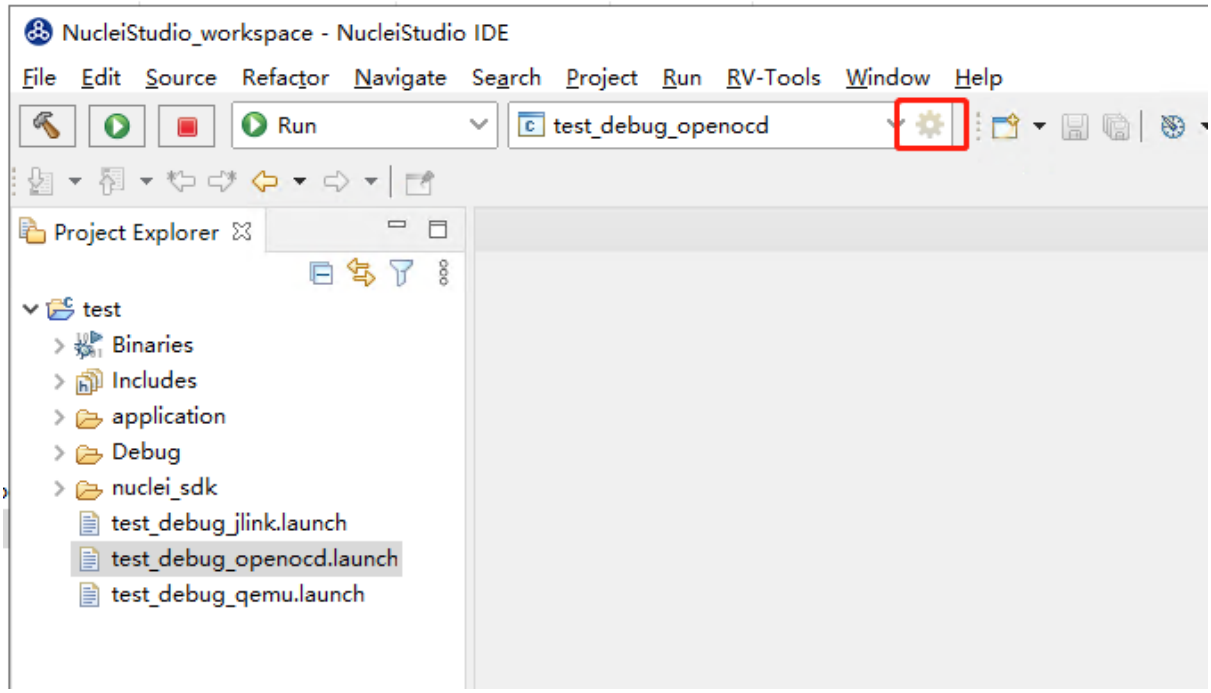


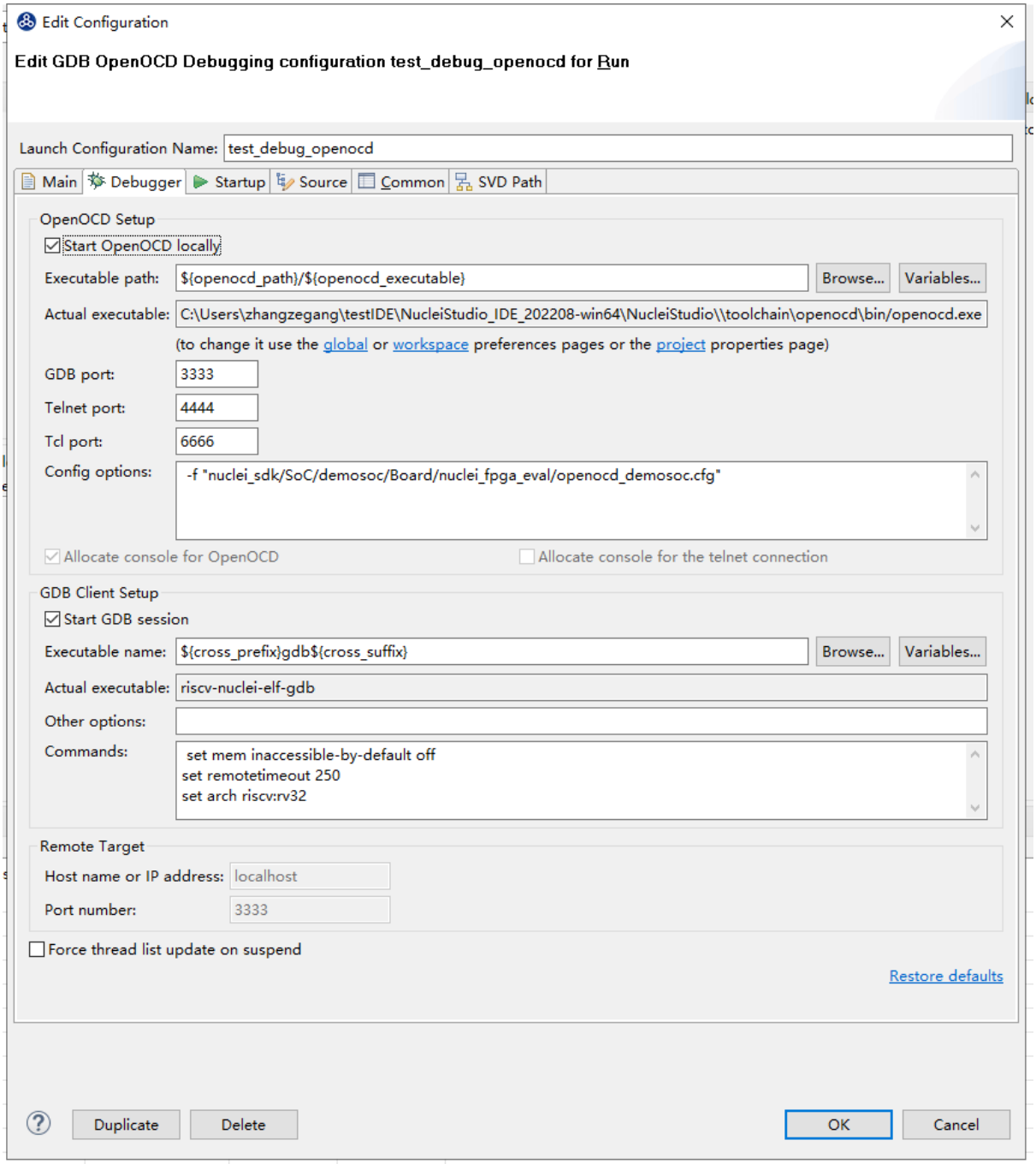
Expand the project, select the corresponding `test_debug_openocd.launch` file, and choose `Run as/Debug as->test_debug_openocd` from the right-click menu to run or debug the project with the corresponding Debug Configuration.



Note: The screenshots may not always be up to date, so figures and text may differ. In that case, follow the text and the behavior of your actual version.

You can view the details of the Debug Configurations in the Launch Bar.

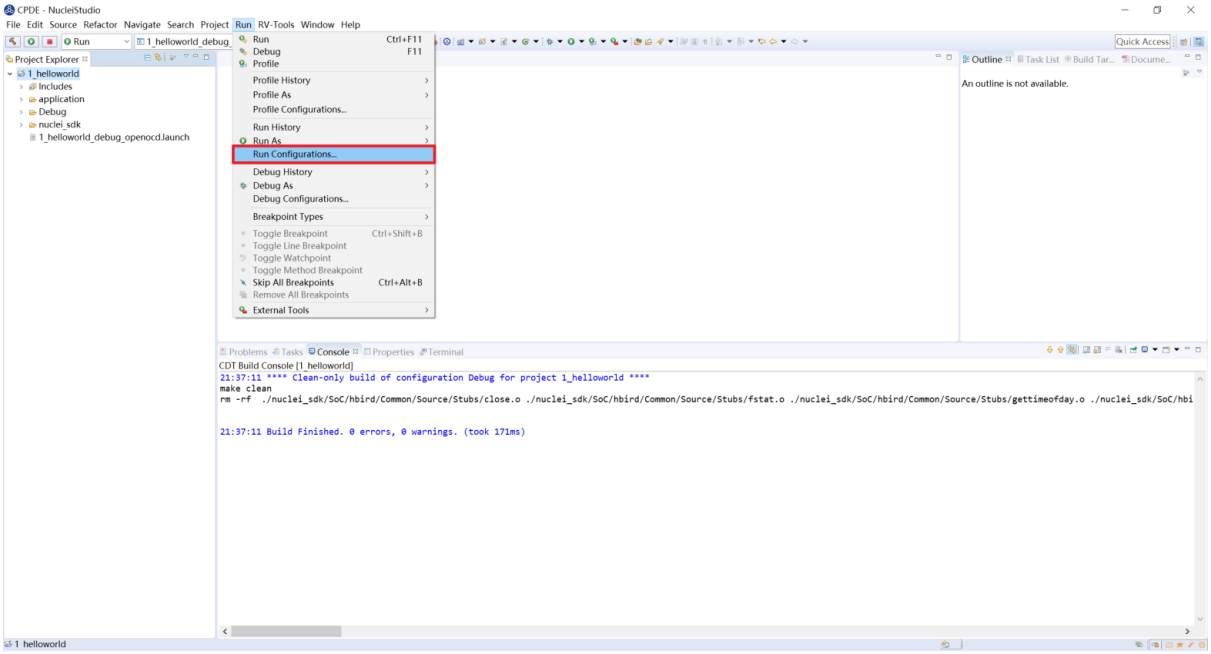

NucleiStudio_workspace - NucleiStudio IDE
File Edit Source Refactor Navigate Search Project Run RV-Tools Window Help
Run
test_debug_openocd
Project Explorer
test
Binaries
Includes
application
Debug
nuclei_sdk
test_debug_jlink.launch
test_debug_openocd.launch
test_debug_qemu.launch



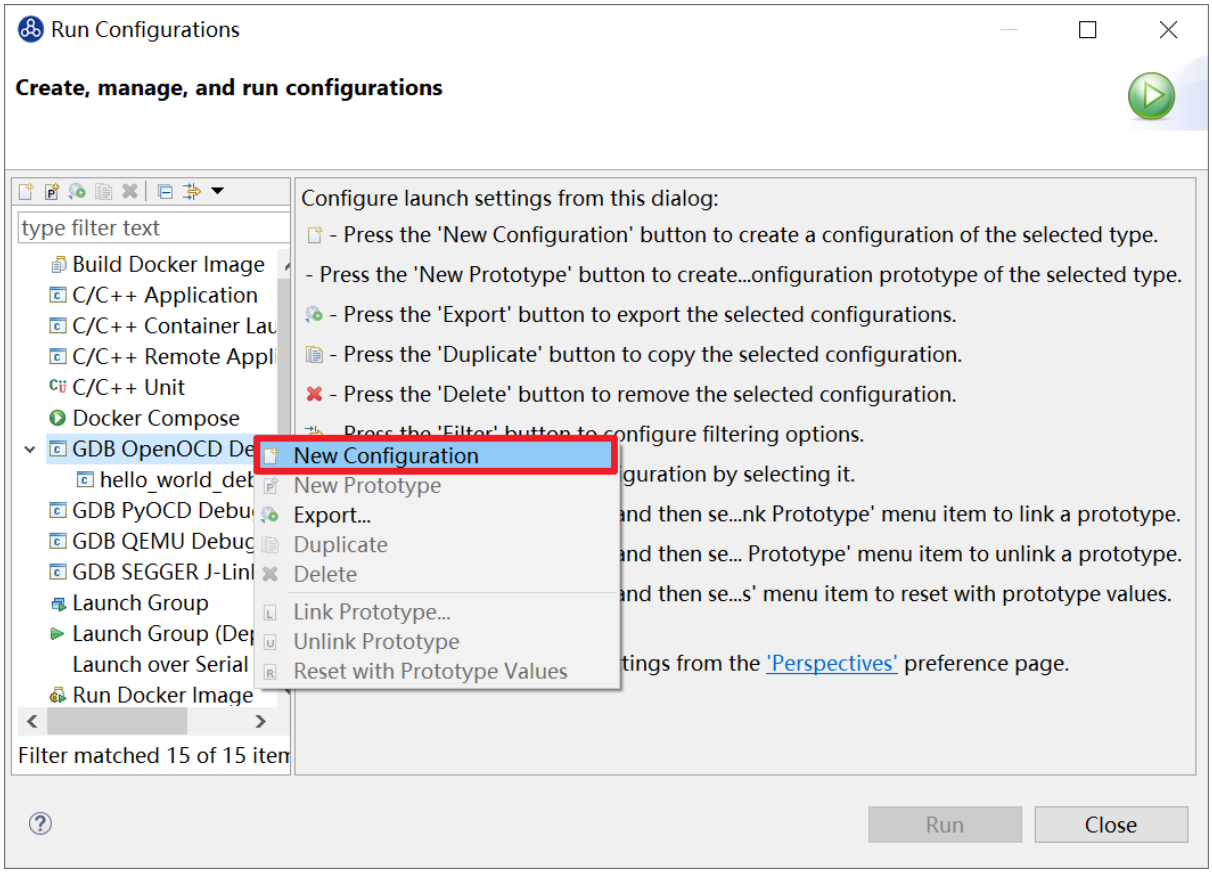
Creating and Configuring a Debug Configuration

To create and configure a Debug Configuration in Nuclei Studio:

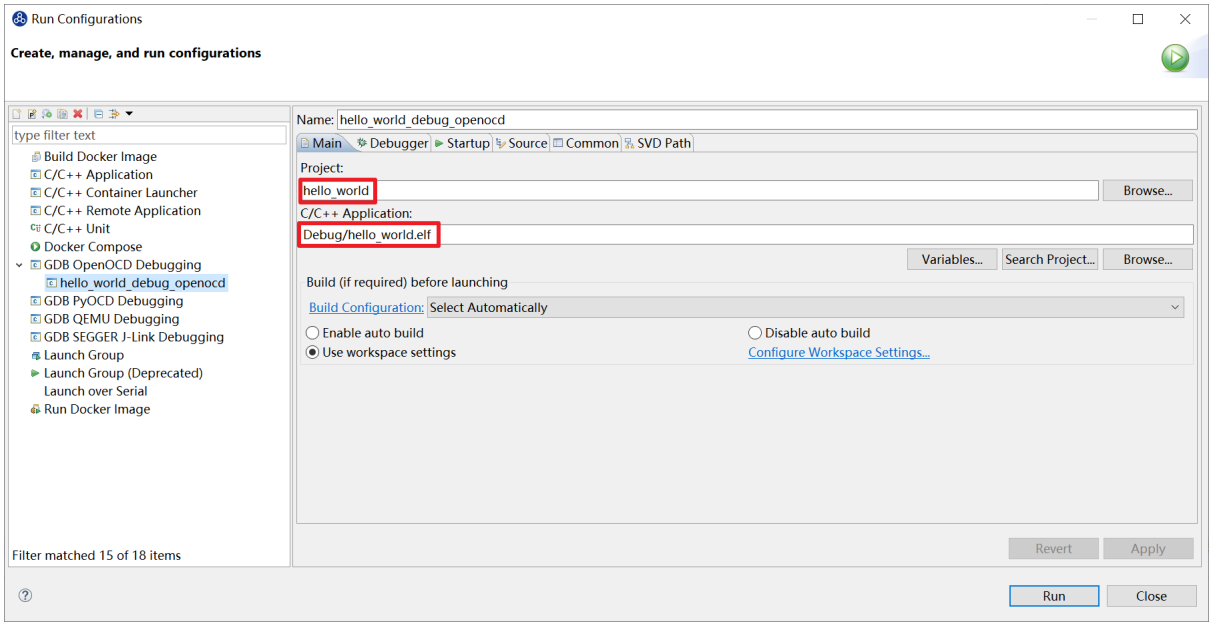
In the Nuclei Studio main menu bar, select `Run—>Debug Configurations`.



In the dialog that opens, if there is no debug configuration for the current project, right-click `GDB OpenOCD Debugging` and select `New` to create a new debug configuration `hello_world_demo Debug` for the project.



Make sure `Project` is the project you want to debug, and that `C/C++ Application` points to the correct ELF file to debug.

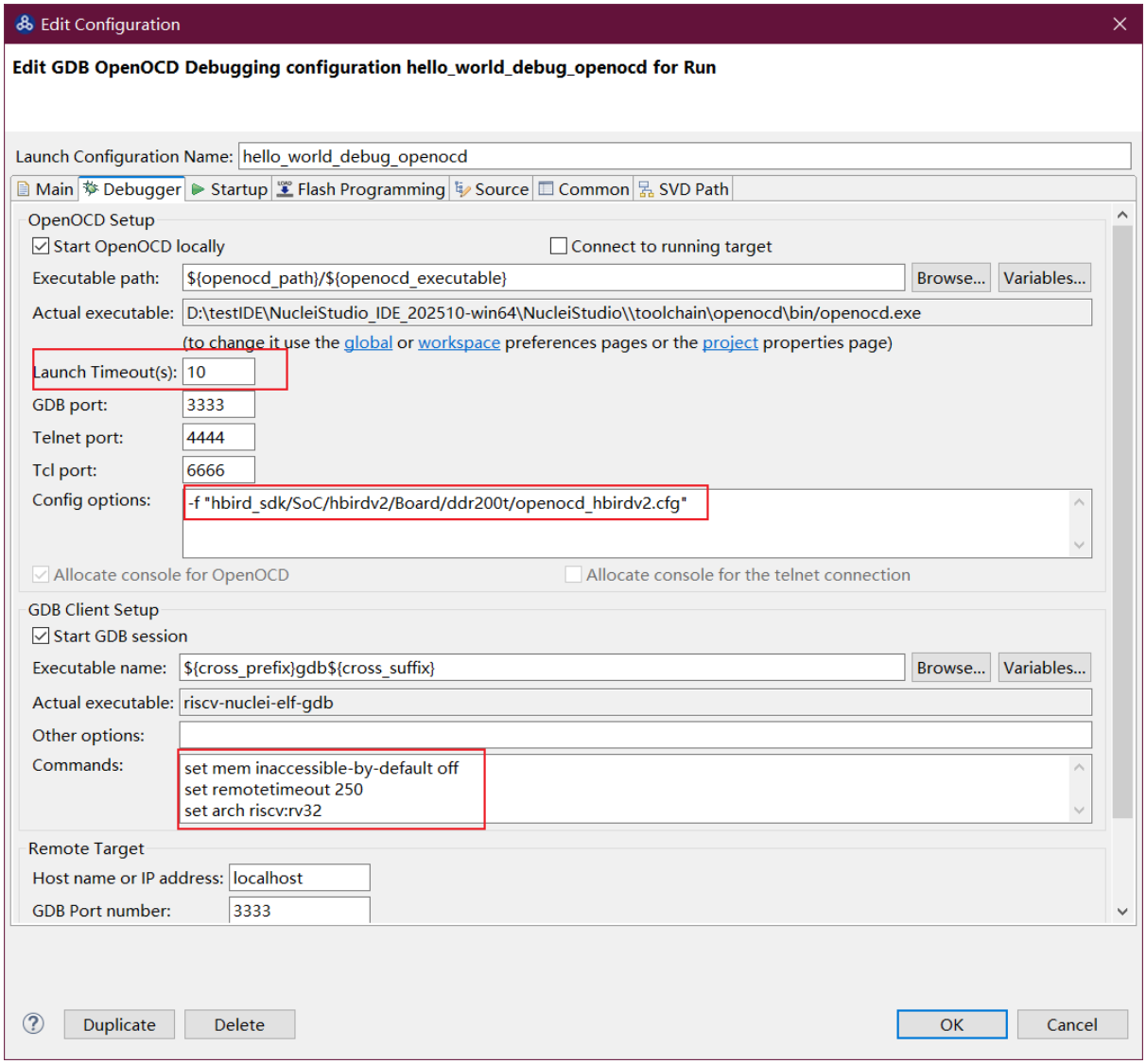


Select the Debugger tab of the debug configuration `hello_world_demo Debug`, and enter `-f "nuclei_sdk/SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg"` in the Config options field so that OpenOCD uses the correct configuration file. The configuration file here (*nuclei_sdk/SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg*) must match the actual OpenOCD configuration file path in your project. For example, if the project is imported using a Makefile, change the content here to `-f "SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg"`.

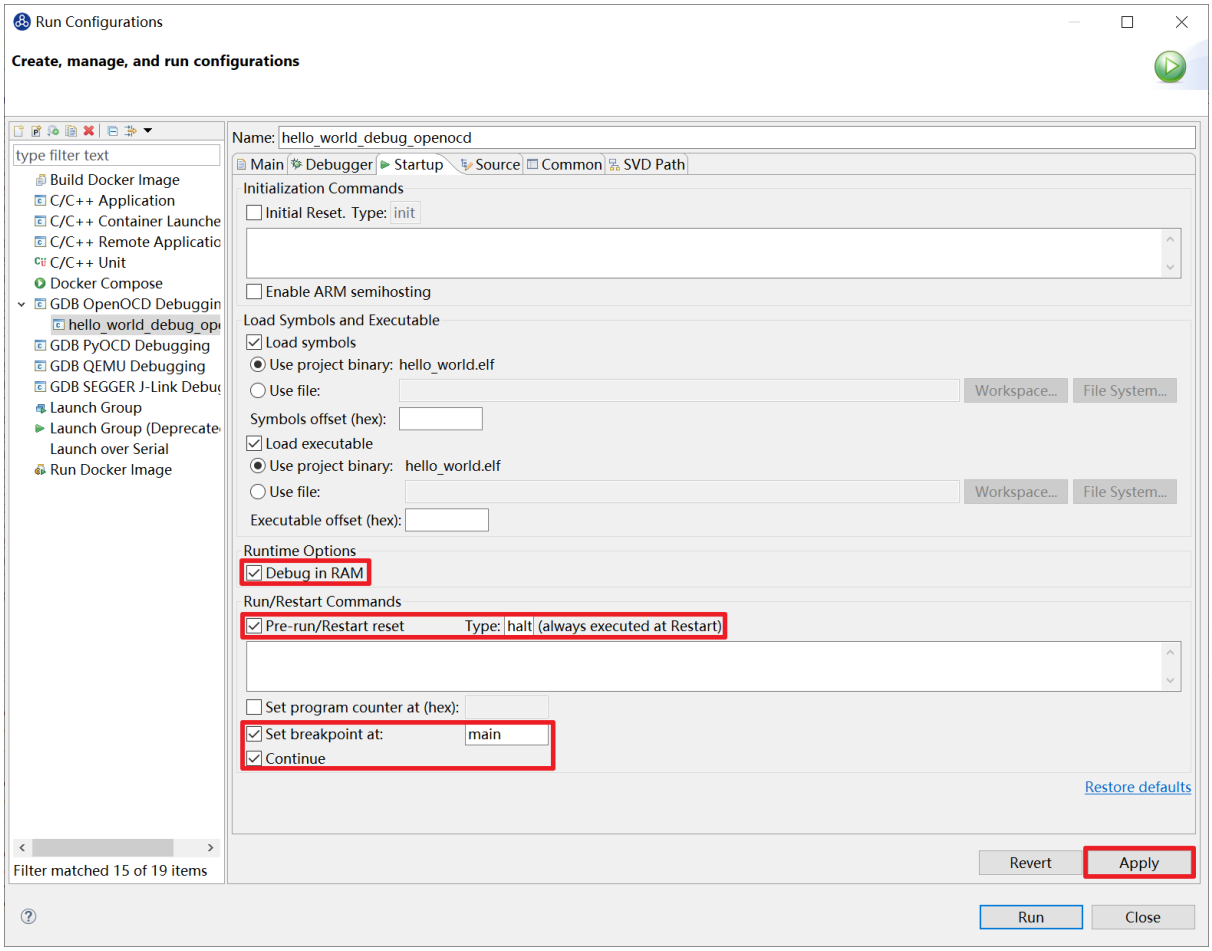
If the current core is a 32-bit RISC-V core, make sure the Commands field includes `set arch riscv:rv32`.

If the current core is 64-bit, use `set arch riscv:rv64` instead.

Note: Launch Timeout is a new configuration option introduced in Nuclei Studio 2025.10. In some scenarios OpenOCD starts very slowly, and if Nuclei Studio does not detect the OpenOCD startup within the default 10 s, the launch times out. Use Launch Timeout to adjust the timeout period.



Select the Startup tab of the debug configuration `hello_world_demo Debug`, and make sure `Debug in RAM`, `Pre-run/Restart reset`, `Set Breakpoint at Main`, and `Continue` are checked.



When the configuration is complete, click `Apply` at the bottom right to save the settings.

Each Startup setting in the Debug Configuration is described in detail below. These settings are ultimately implemented as GDB commands and are executed in the order listed.

The official Eclipse Embedded CDT documentation also provides a basic introduction to the Startup parameters; see https://eclipse-embed-cdt.github.io/debug/openocd/riscv/ for details.

Initial Reset

Specifies the commands GDB runs during initialization. See the usage of GDB init commands for details; this option generally does not need to be checked.

Enable Arm Semihosting

If you need to enable Semihosting during debugging, check this option; you can also add `arm semihosting enable` directly to the OpenOCD configuration file. For more details on project configuration and J-Link debugging, see:

- *How to build a project that supports Semihosting* (page 391)
- *How to debug a Semihosting-enabled project with J-Link* (page 393)

Load symbols

Uses the GDB `file` command so that GDB reads the debug information from the ELF file, allowing GDB to debug correctly. Keep this option checked.

Load executable

Uses the GDB `load` command to download the program to the target: GDB downloads the ELF content to the target and sets the PC of the target CPU to the entry address of the current ELF. If you are debugging a program in RAM (such as LM), keep this option checked. If you are debugging a program in Flash, check whether OpenOCD

supports Flash programming and whether the current debug session needs to reprogram the Flash: if both are true, check this option; otherwise, leave it unchecked. This option is not needed for ROM code.

Debug in Ram

If GDB performs actions such as reset after `load executable`, the CPU boot address may differ from the start address of the ELF file, or the program in RAM may be cleared by the reset. The ELF file must therefore be reloaded after each GDB reset. If you are debugging a program in RAM, this option must be checked.

Pre-run/Reset

Uses the GDB `monitor reset` command, which sends a `reset` command to OpenOCD. OpenOCD drives the `nReset` signal according to the RISC-V Debug Spec, and this signal resets both the `core` and the `peripherals` (this also depends on the specific implementation, but the RISC-V Debug Spec recommends this behavior, and Nuclei ' s example SoCs and FPGA implementations follow this spec). After `reset` is executed, the CPU ' s PC is the `reset_vector` address, and because all peripherals are reset, the RAM content is also cleared. If you are debugging in RAM and this option is checked, the `Debug in RAM` option must also be checked.

The following figure shows the description of `nRESET` in the RISC-V Debug Spec.

Table 6.7: JTAG Connector Pinout

1	VREF DEBUG	Reference voltage for logic high.
2	TMS	JTAG TMS signal, driven by the debug adapter.
4	TCK	JTAG TCK signal, driven by the debug adapter.
6	TDO	JTAG TDO signal, driven by the target.
7	GND or KEY	This pin may be cut on the male and plugged on the female header to ensure the header is always plugged in correctly. It is, however, recommended to use this pin as an additional ground, to allow for fastest TCK speeds. A shrouded connector should be used to prevent the cable from being plugged in incorrectly.
8	TDI	JTAG TDI signal, driven by the debug adapter.
10	nRESET	Active-low reset signal, driven by the debug adapter. Asserting reset should reset any RISC-V cores as well as any other peripherals on the PCB. It should not reset the debug logic. This pin is optional but strongly encouraged. If necessary, this pin could be used as nTRST instead. nRESET should never be connected to the TAP reset, otherwise the debugger might not be able to debug through a reset to discover the cause of a crash or to maintain execution control after the reset.
12	RTCK	Return test clock, driven by the target. A target may relay the TCK signal here once it has processed it, allowing a debugger to adjust its TCK frequency in response.
14	nTRST_PD	Test reset pull-down (optional), driven by the debug adapter. Same function as nTRST, but with pull-down resistor on target.
16	nTRST	Test reset (optional), driven by the debug adapter. Used to reset the JTAG TAP Controller.
18	TRIGIN	Not used, driven low by the debug adapter.
20	TRIGOUT	Not used, driven by the target.

Halt

Uses the GDB `monitor halt` command, which sends a `halt` command through OpenOCD after the `reset` command, so that the CPU halts immediately after reset.

Set Program counter at

Uses the GDB `set $pc` command to set the PC of the target CPU.

Set breakpoint at

Uses the GDB `break` command. The default is the `main` function; for a first debug session or when debugging startup code, change it to something else, such as `_start` or an absolute address.

Continue

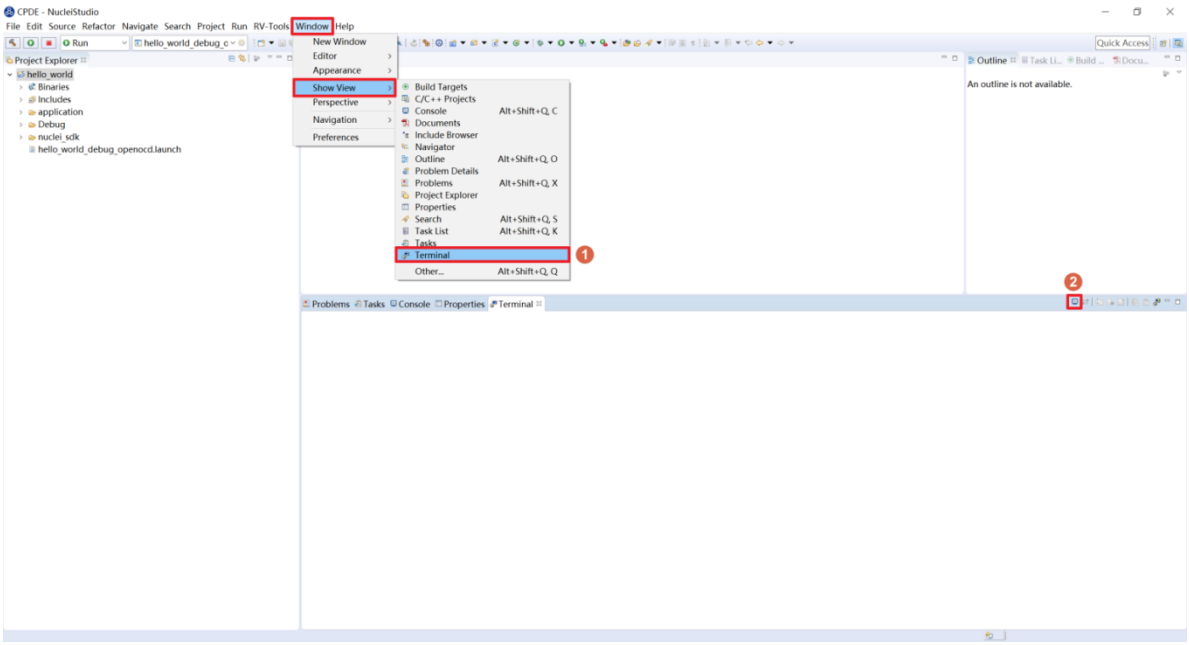
Uses the GDB `continue` command.

Debugging the Program on the Prototype Board

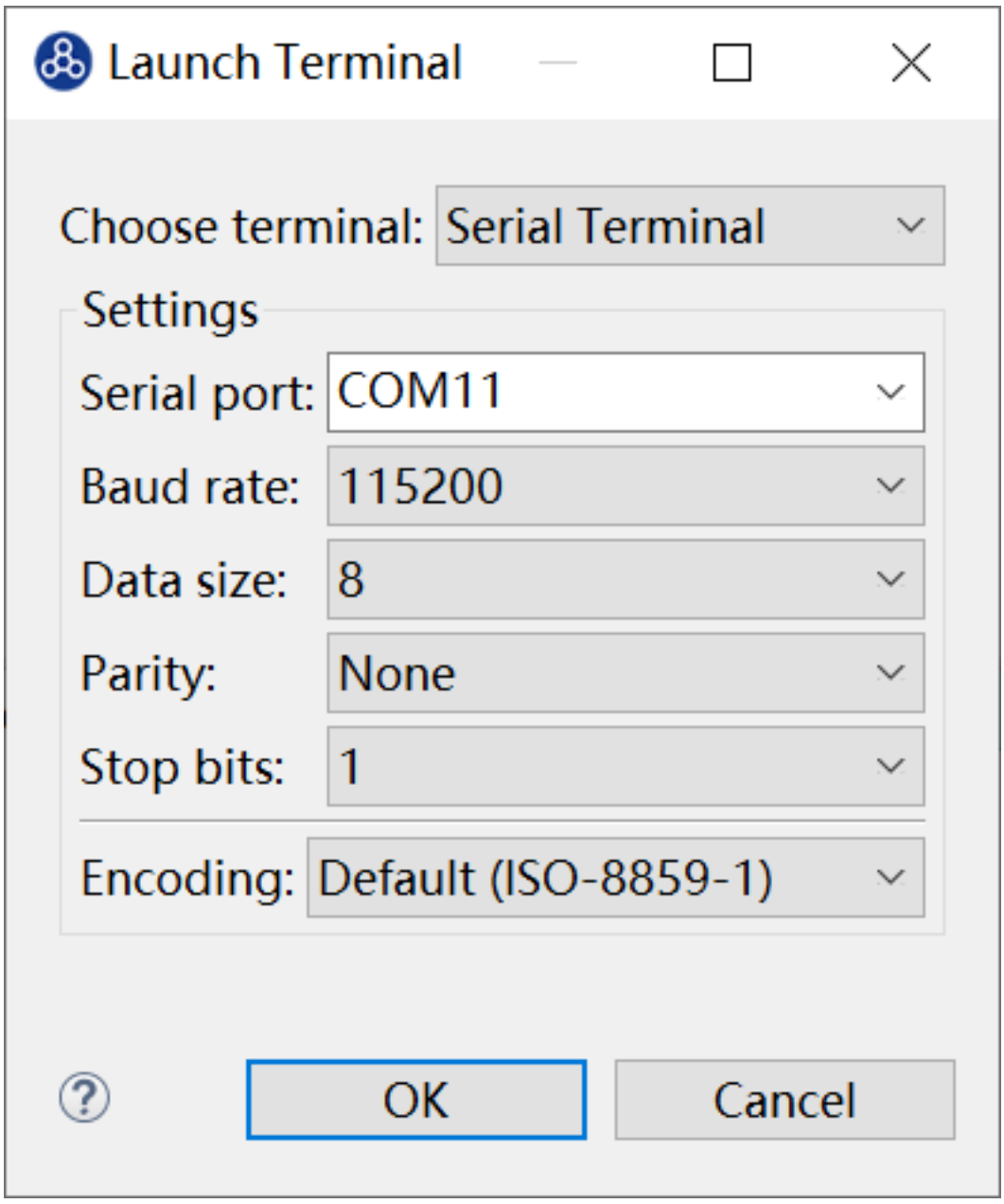
Before debugging on the development board, you need to open the serial port to observe the printf output.

To open the serial port on Windows:

Open the serial terminal built into Nuclei Studio: select `Window>Show View>Terminal`, and, as shown in the figure, click the monitor icon to open the serial port settings.

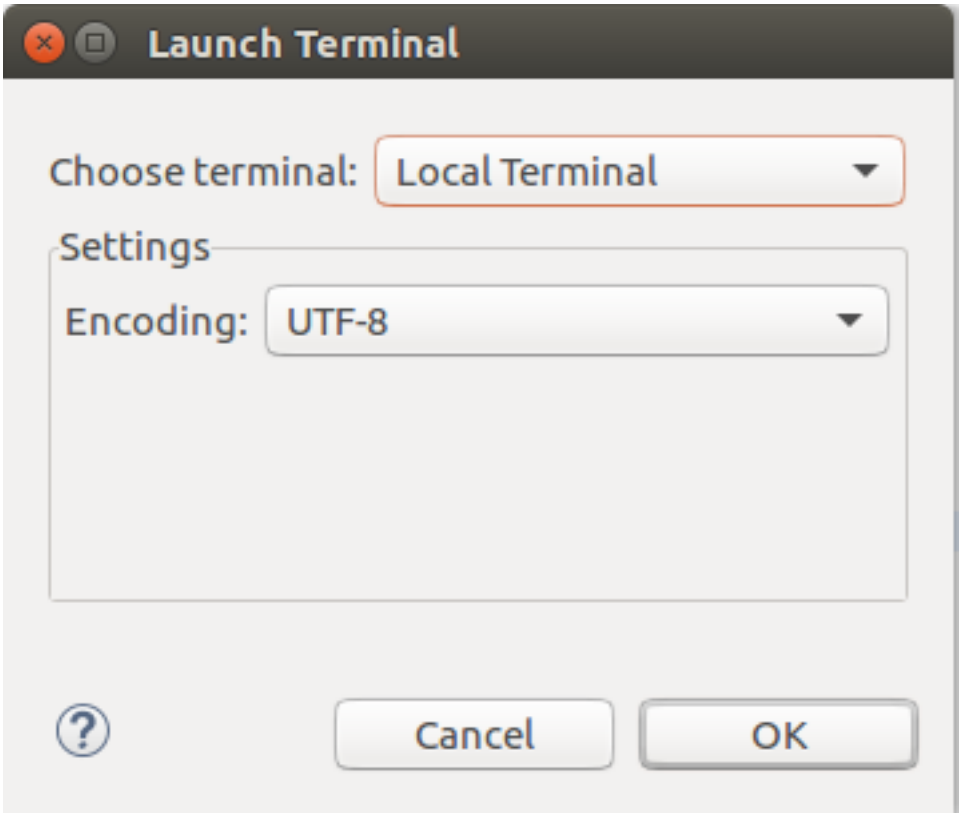


In the window, set Choose terminal to Serial Terminal, enter the `serial port number` (COM11 is used as an example here) and the `baud rate (set to 115200)`, adjust the other parameters as needed, and then click the `OK` button.

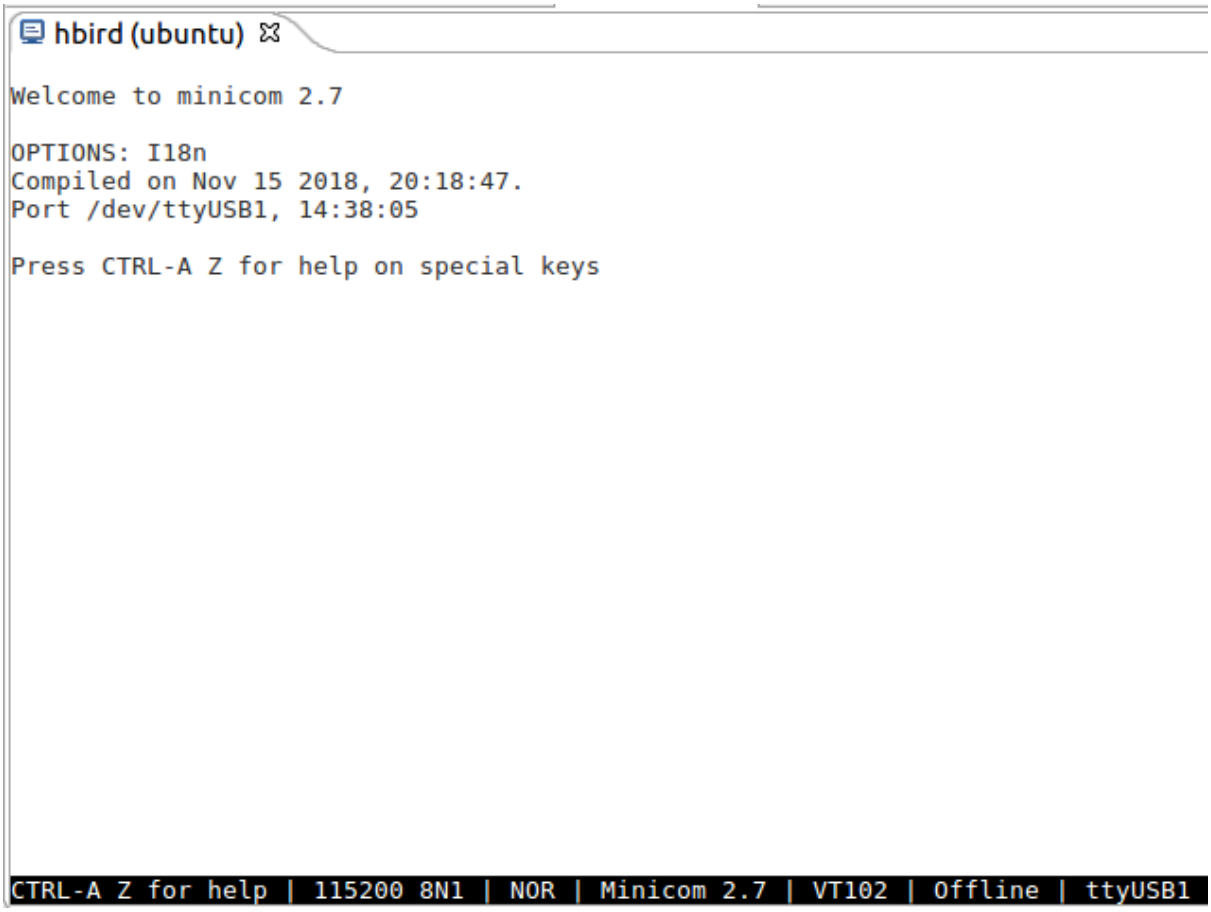


To open the serial port on Linux:

Open the Terminal built into Nuclei Studio: select `Window>Show View>Terminal`, and click the monitor icon to open the terminal settings. Select `Local Terminal` for Choose terminal, and click OK to open the Terminal.



Enter `minicom /dev/ttyUSB1 115200` in the window to open the serial port; you can then view the serial port output in Nuclei Studio.

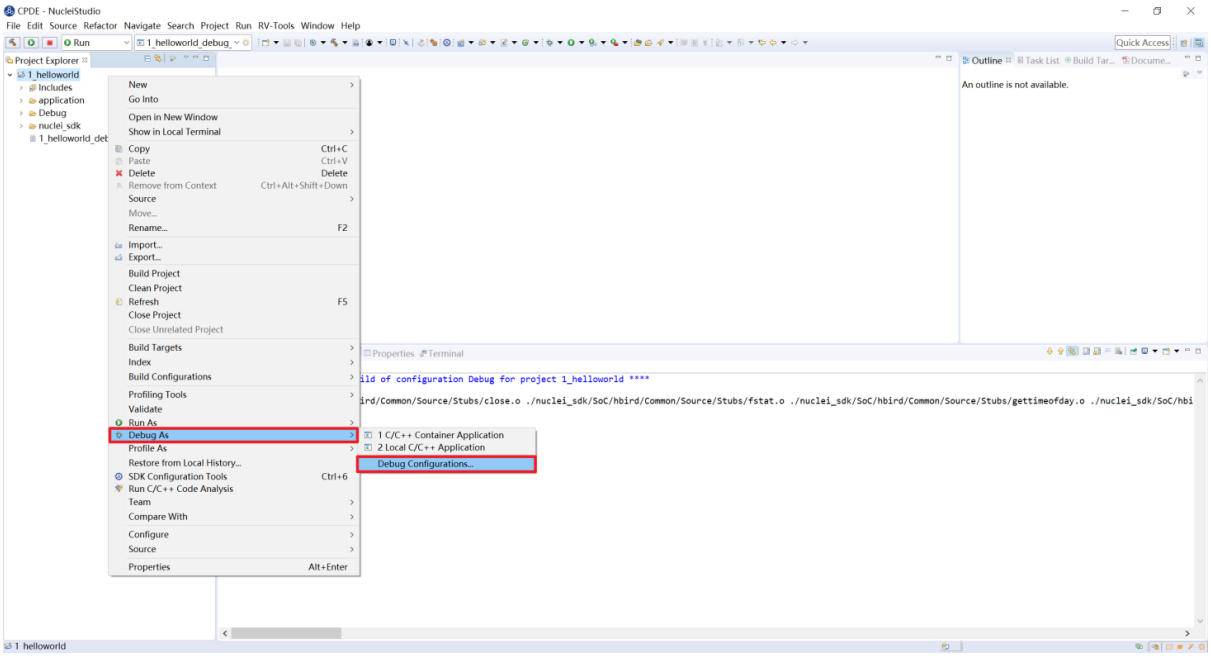


To debug a program running on the prototype board, use the Nuclei Studio IDE. Because the IDE runs on the host PC while the program runs on the prototype board, this kind of debugging is also called `online debugging` or `remote debugging`.

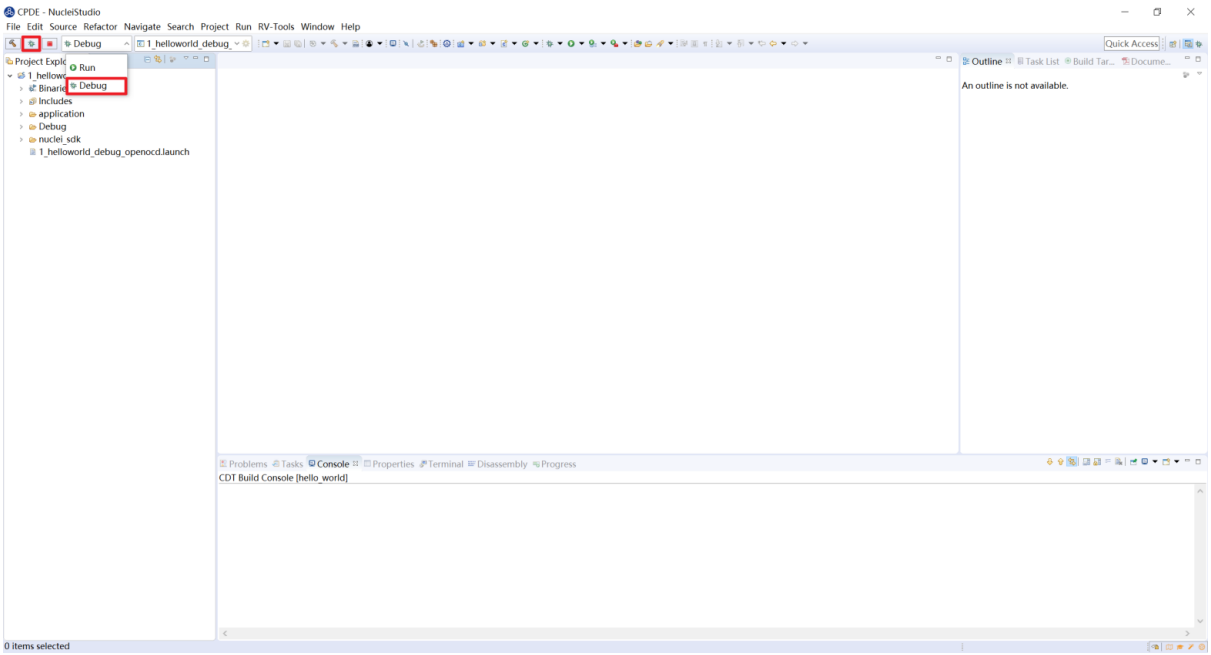
The following steps use 1_helloworld as an example to perform online debugging on the evalsoc prototype board with the Nuclei Studio IDE:

Note: demosoc was removed in Nuclei SDK 5.0; use evalsoc instead.

Make sure the debug settings are correct. Right-click the 1_helloworld project and select `Debug As >Debug Configuration` to open the debug settings page, then select the previously created configuration and check it.

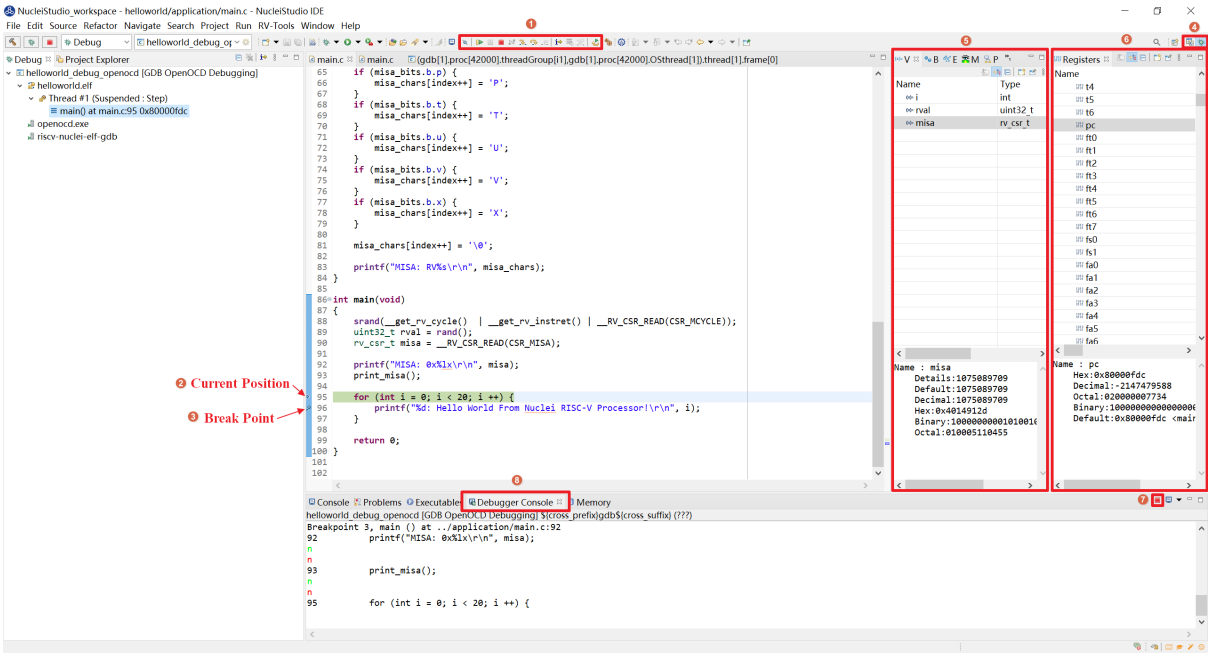


After confirming the settings are correct, select Debug in the drop-down box. The icon on the left changes to a bug icon; click it to enter debug mode and download the program to the development board.



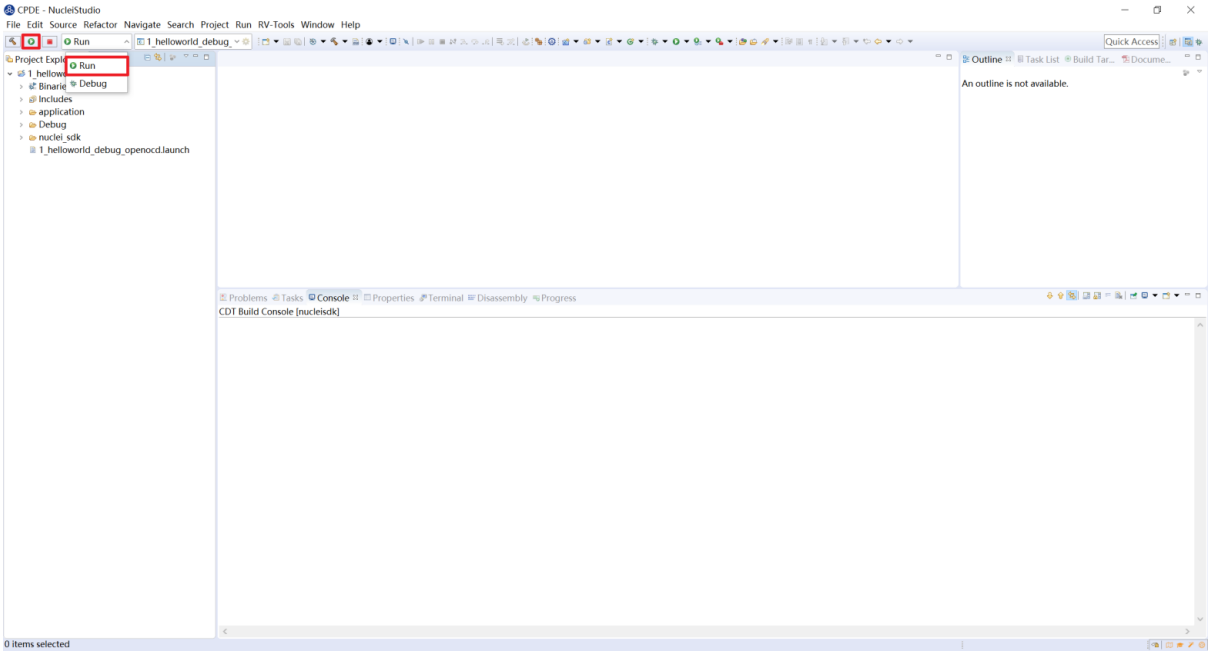
If the download succeeds, the debug perspective opens.

- At marker 1: step, run, and assembly-level debugging functions.
- At marker 2: the arrow indicates the current execution position of the program.
- At marker 3: double-click in the left margin of a code line to set a breakpoint on that line; double-click again to remove the breakpoint.
- At marker 4: switch between edit mode and debug mode here.
- At marker 5: in-function variables are displayed here.
- At marker 6: register values are viewed here. The figure shows the current value of the PC register.
- At marker 7: click the red button here to exit debug mode.
- Below marker 8: GDB console commands can be used here for debugging.

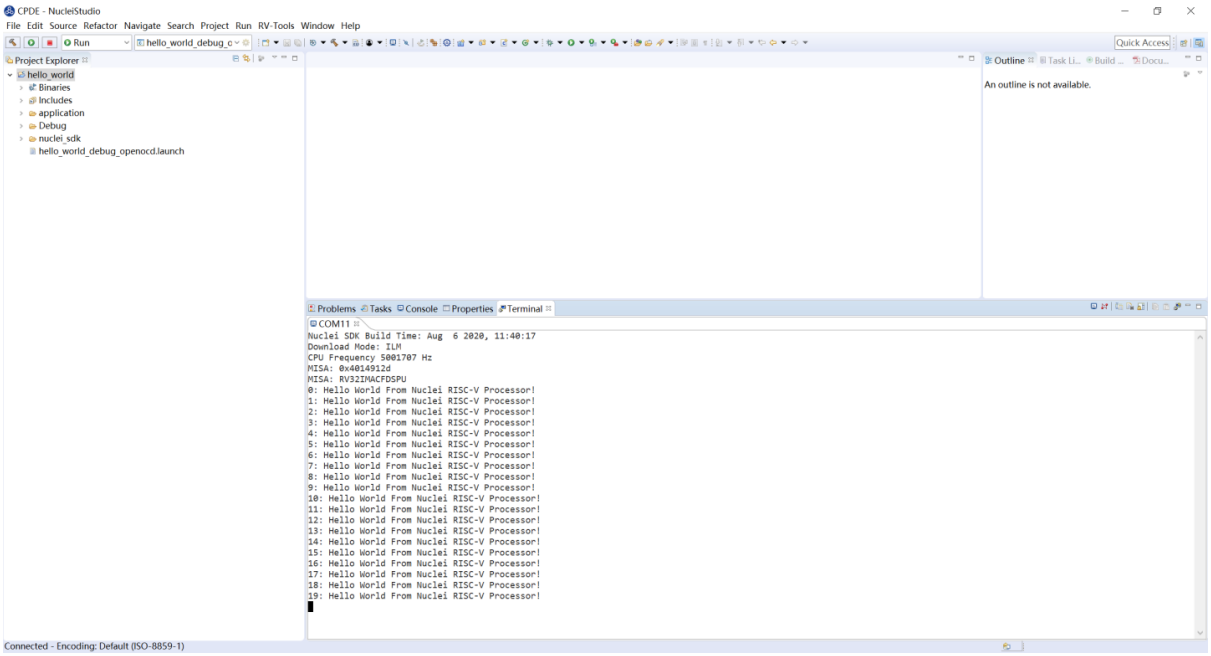


Downloading and Running the Program

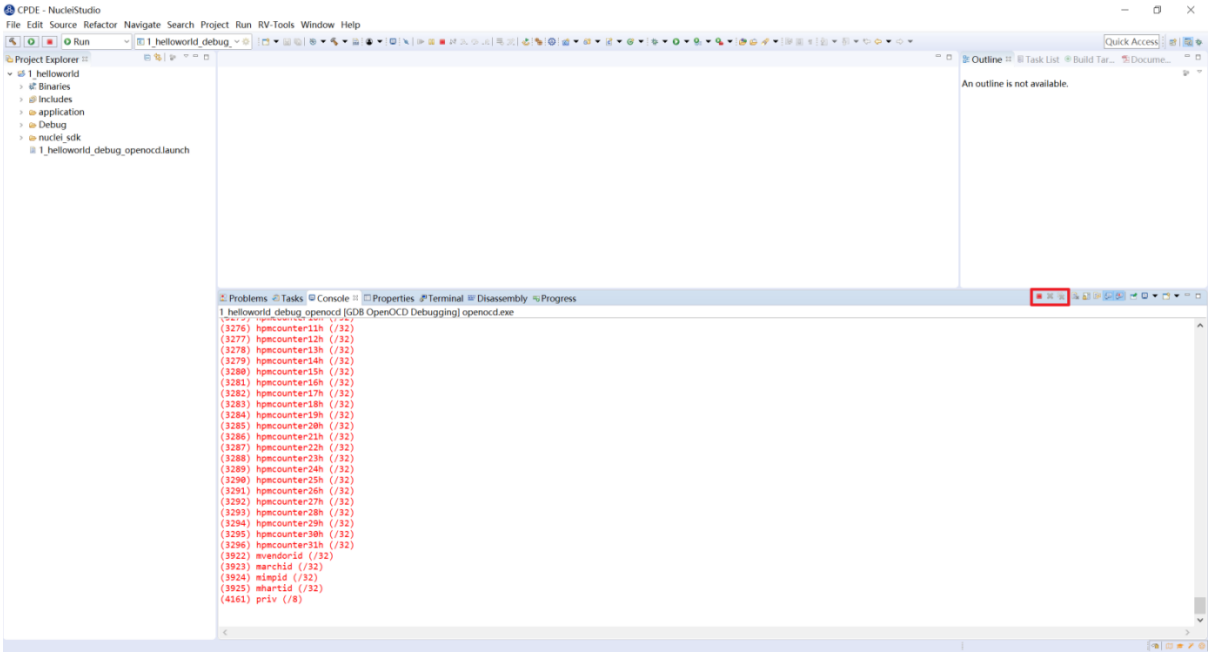
After debugging confirms there are no problems, the program can be downloaded to the development board. Click the drop-down box to switch to run mode; the icon on the left then changes to a green run button. Click it to download the program to the development board and run it. Because debugging and downloading use the same configuration file, no further setup is required.



Once the program runs normally, the serial port correctly prints messages such as helloworld.



To stop the program and disconnect, click the red button in the console view.



2.9.3 Debugging and Running a Project with J-Link

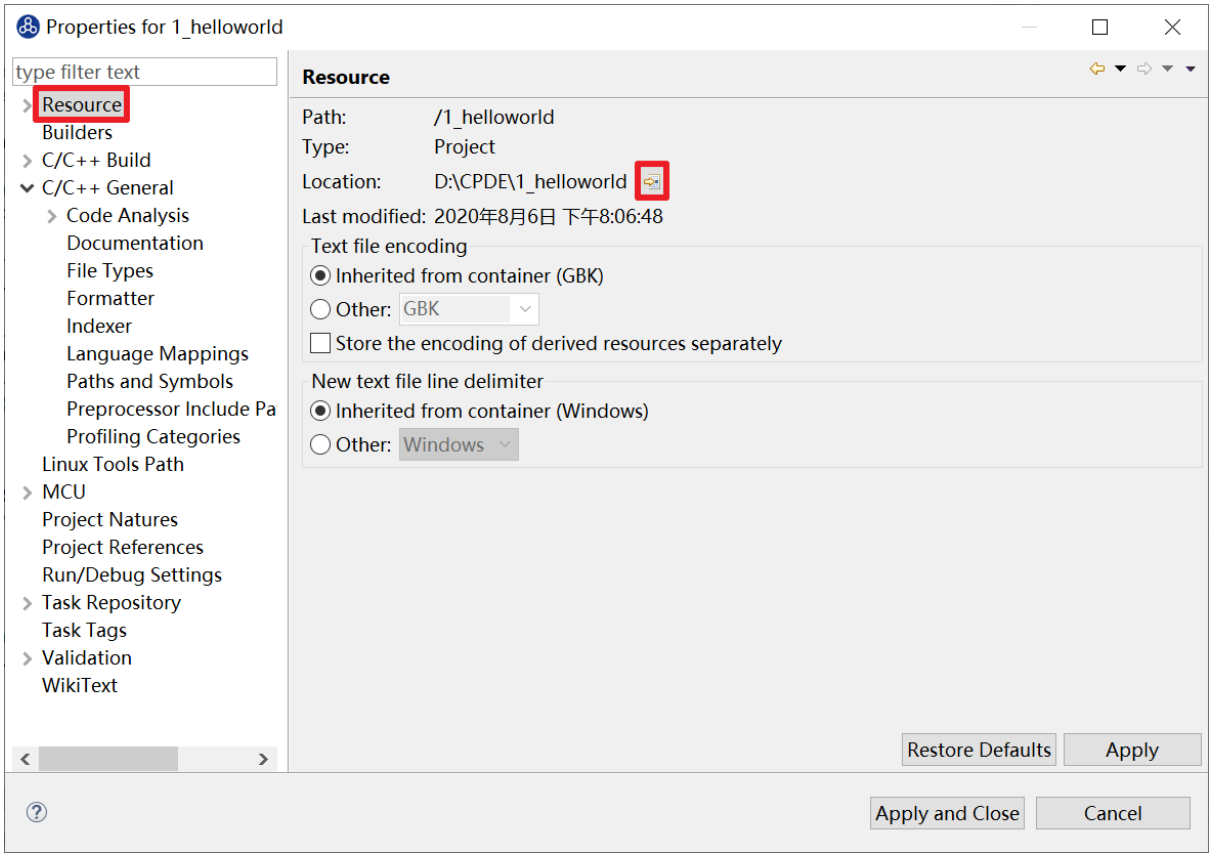
Note: If you are using J-Link GDB Server, read the corresponding J-Link documentation to learn about its usage; see https://kb.segger.com/J-Link_GDB_Server

Installing the J-Link Driver and Importing the RTT Files

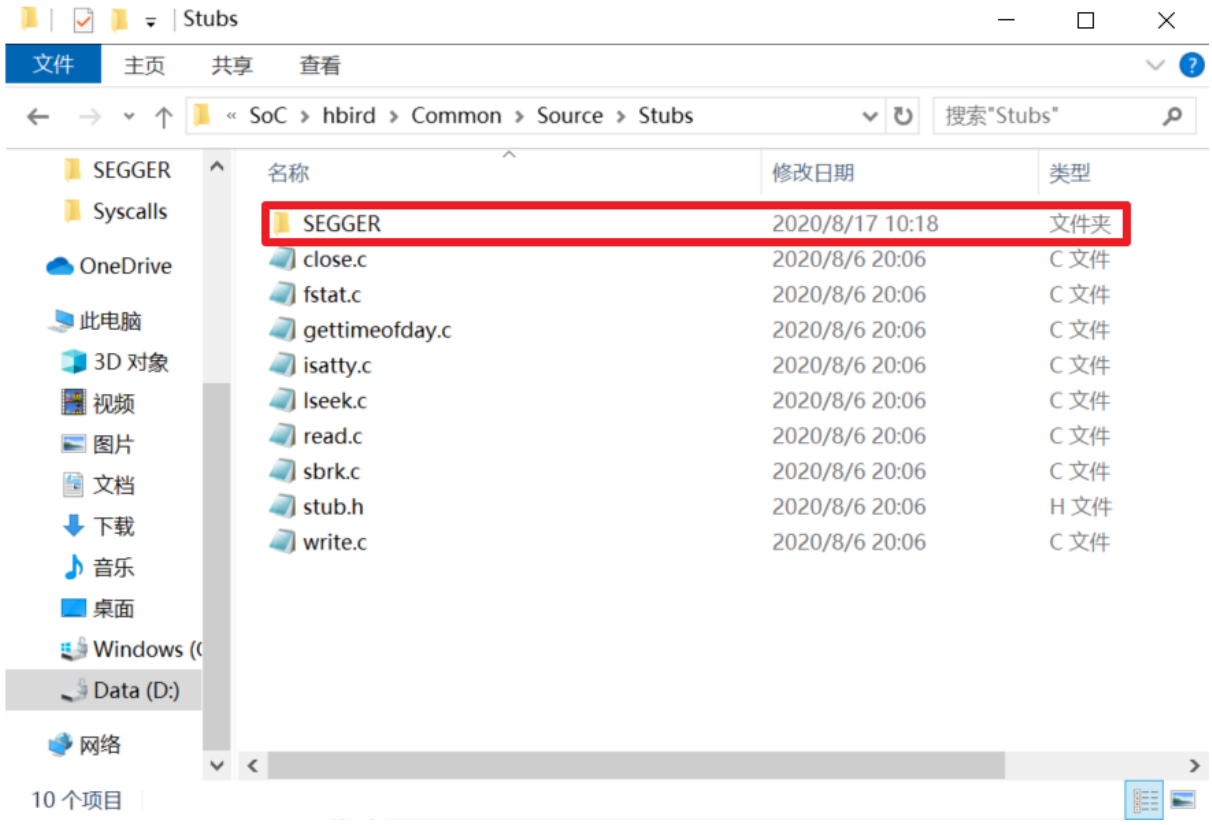
The HummingBird Evaluation Board also supports debugging with J-Link. Go to the J-Link page on the SEGGER website (https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack), download the latest J-Link driver for your operating system, and install it. Note that the J-Link version must be later than v6.62.

If you use the serial port for print output, you can skip the rest of this section. If you want to use J-Link RTT for print output, follow the steps below to configure it.

Open the settings page of the current project, and under the `Resource` option click the icon marked with the red box to quickly open the project directory.

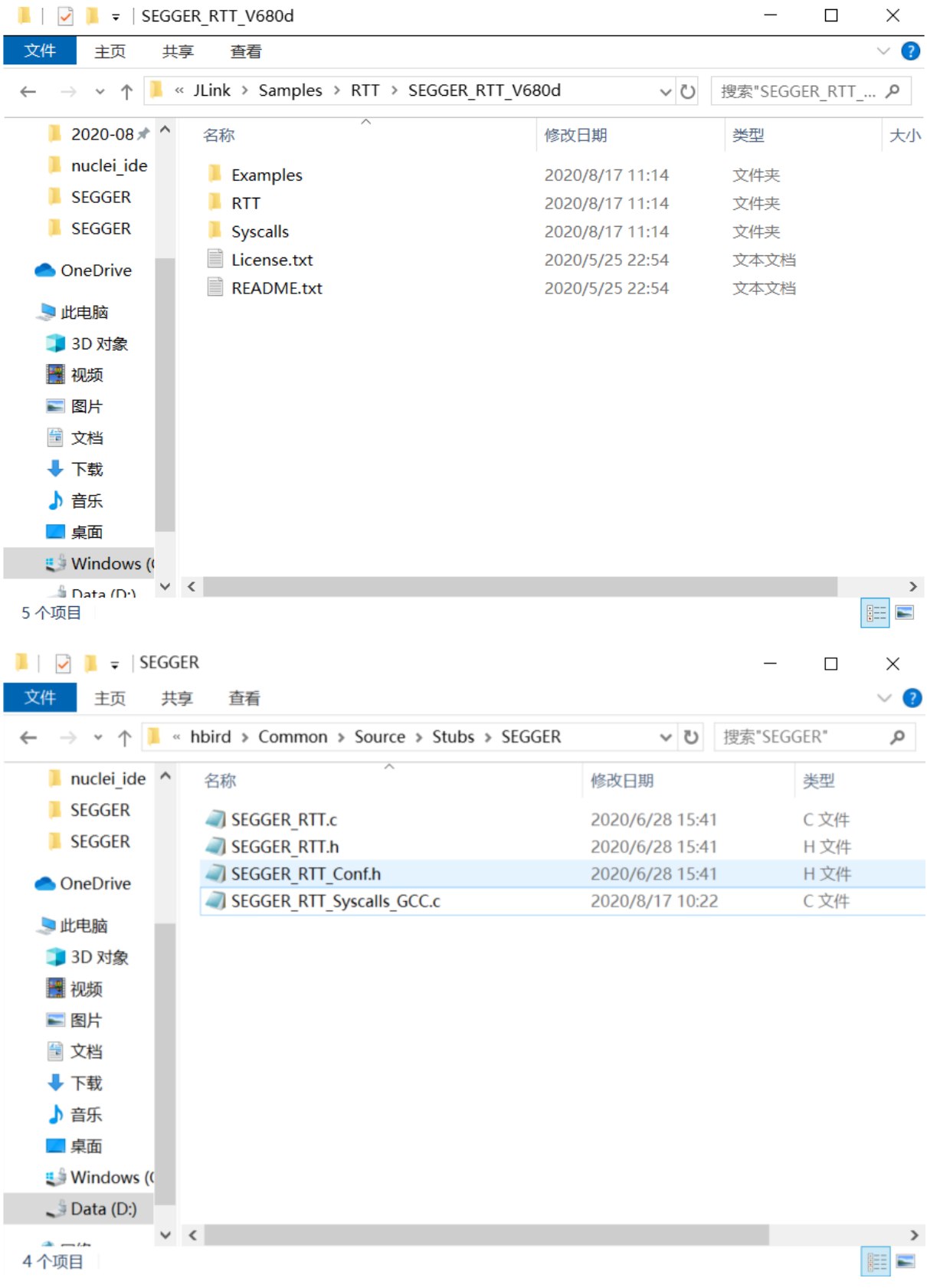


Create a new `SEGGER` folder under `nuclei_sdk/SoC/evalsoc/Common/Source/Stubs` to store the RTT-related files.

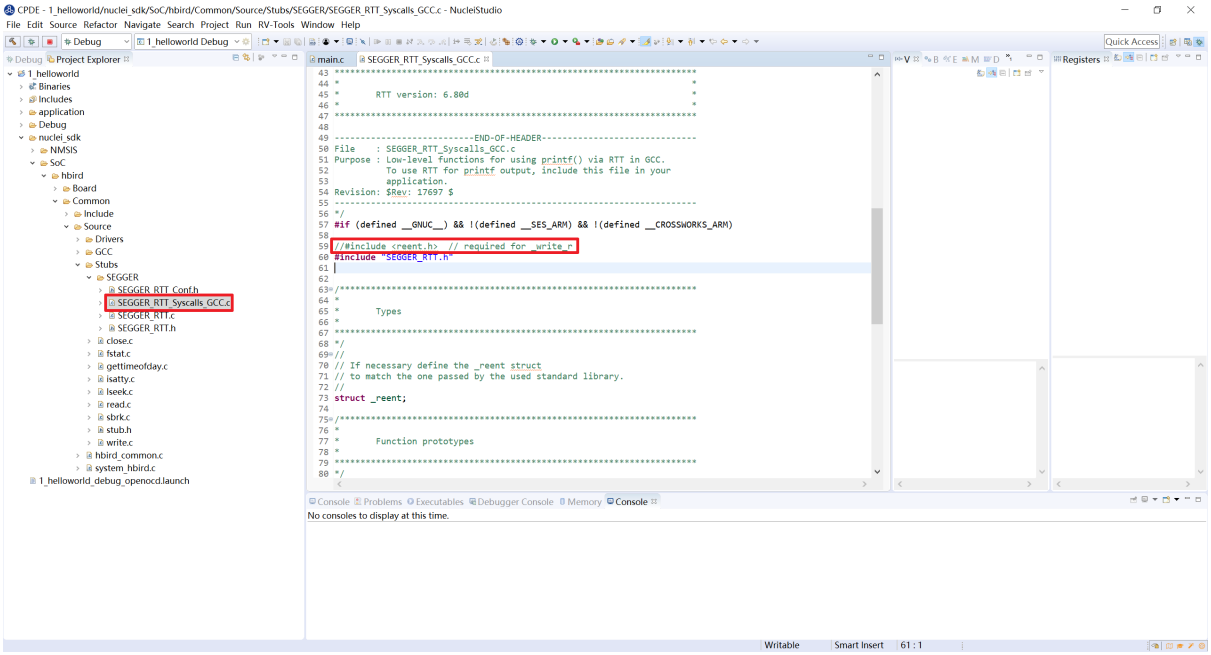


After installation, open the root directory of the J-Link driver and extract `SEGGER_RTT_V680d.zip` under `Samples -> RTT` (the exact archive name may vary by version). From the extracted files, copy

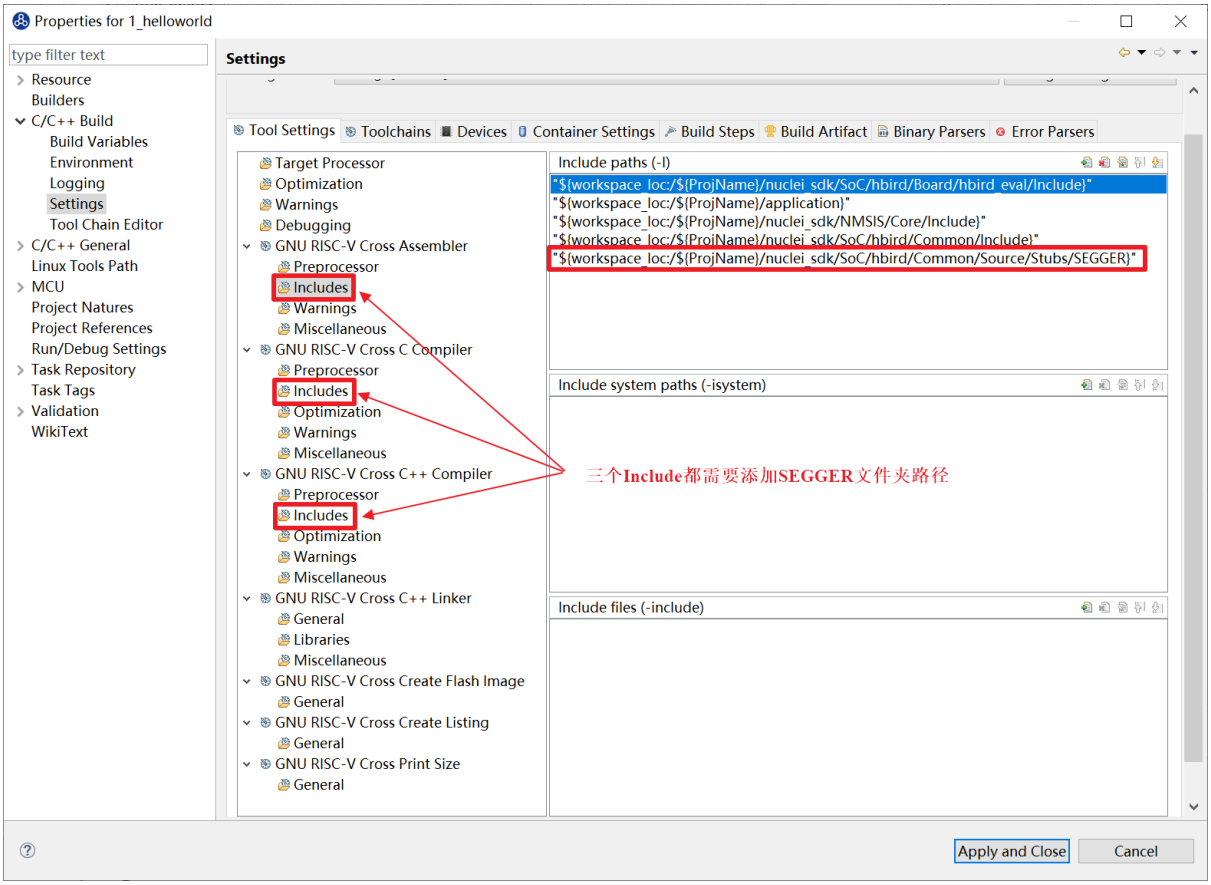
`SEGGER_RTT.c`, `SEGGER_RTT.h`, and `SEGGER_RTT_Conf.h` from the RTT folder, as well as `SEGGER_RTT_Syscalls_GCC.c` from the Syscalls folder, into the newly created SEGGER folder.



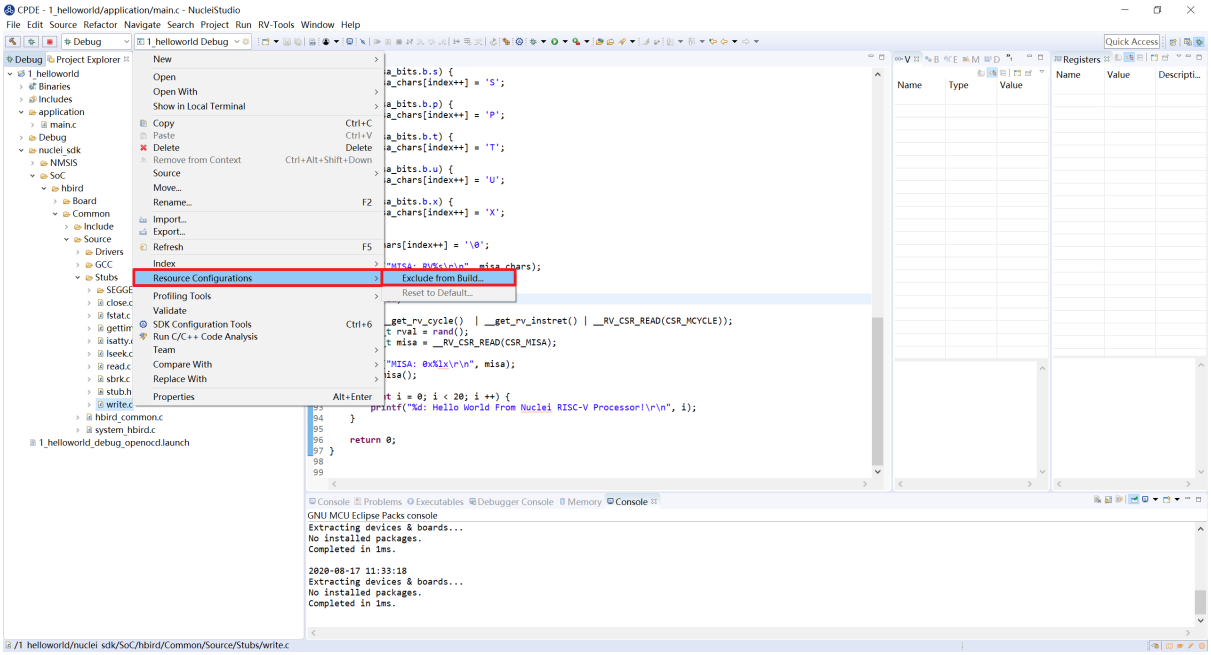
Finally, open `SEGGER_RTT_Syscalls_GCC.c` in the IDE and comment out the line containing `#include <reent.h>`.



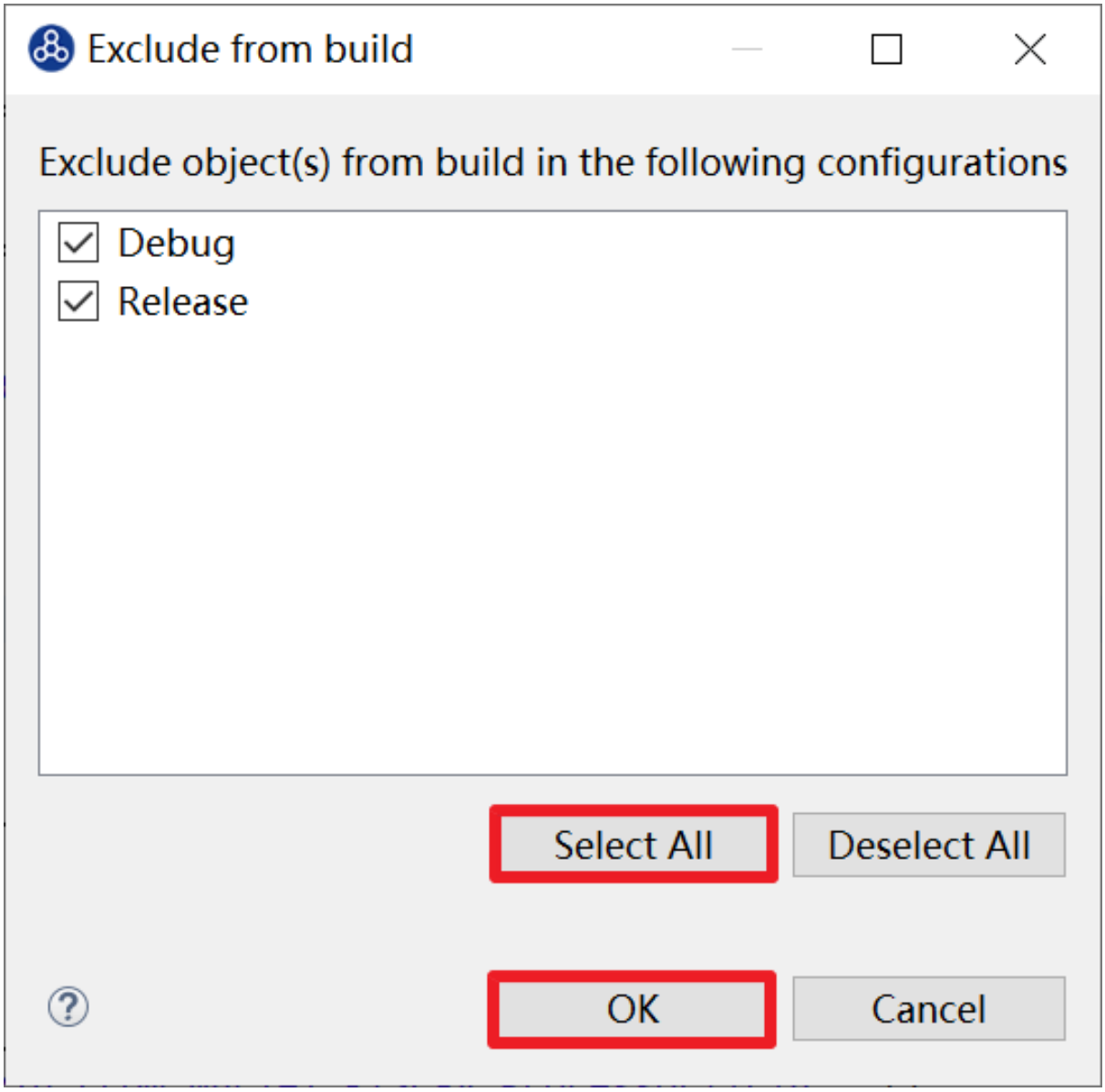
Next, add the SEGGER folder to the include paths: open the settings page of the current project and add the SEGGER folder path to the includes.



Then remove the original write function: right-click the `write.c` file under `nuclei_sdk/SoC/evalsoc/Common/Source/Stubs` and select `Resource Configurations > Exclude from Build`. As shown in the figure, select `Select All` and click `OK`.



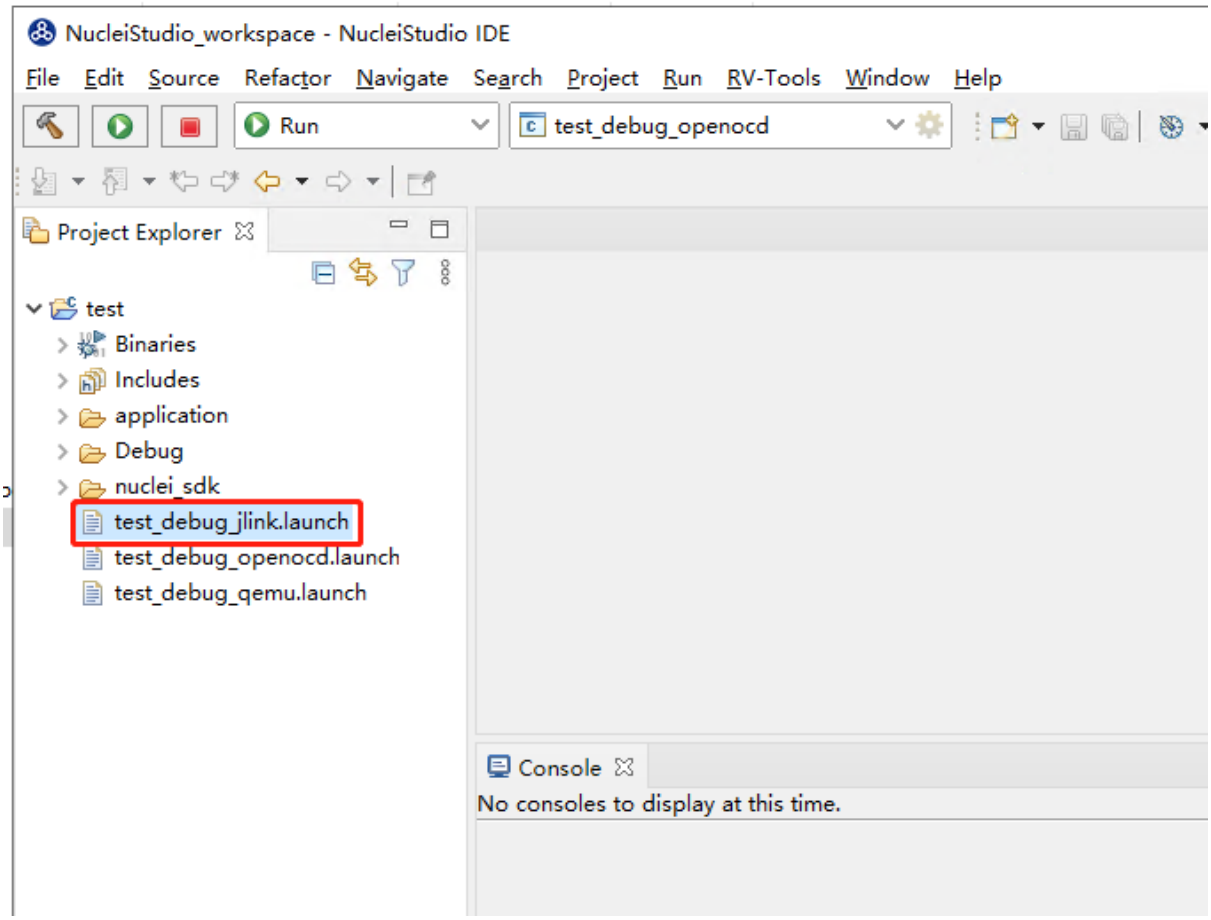
To switch back to serial port printing later, remove the SEGGER folder in the same way and add the `write.c` file back to the build.



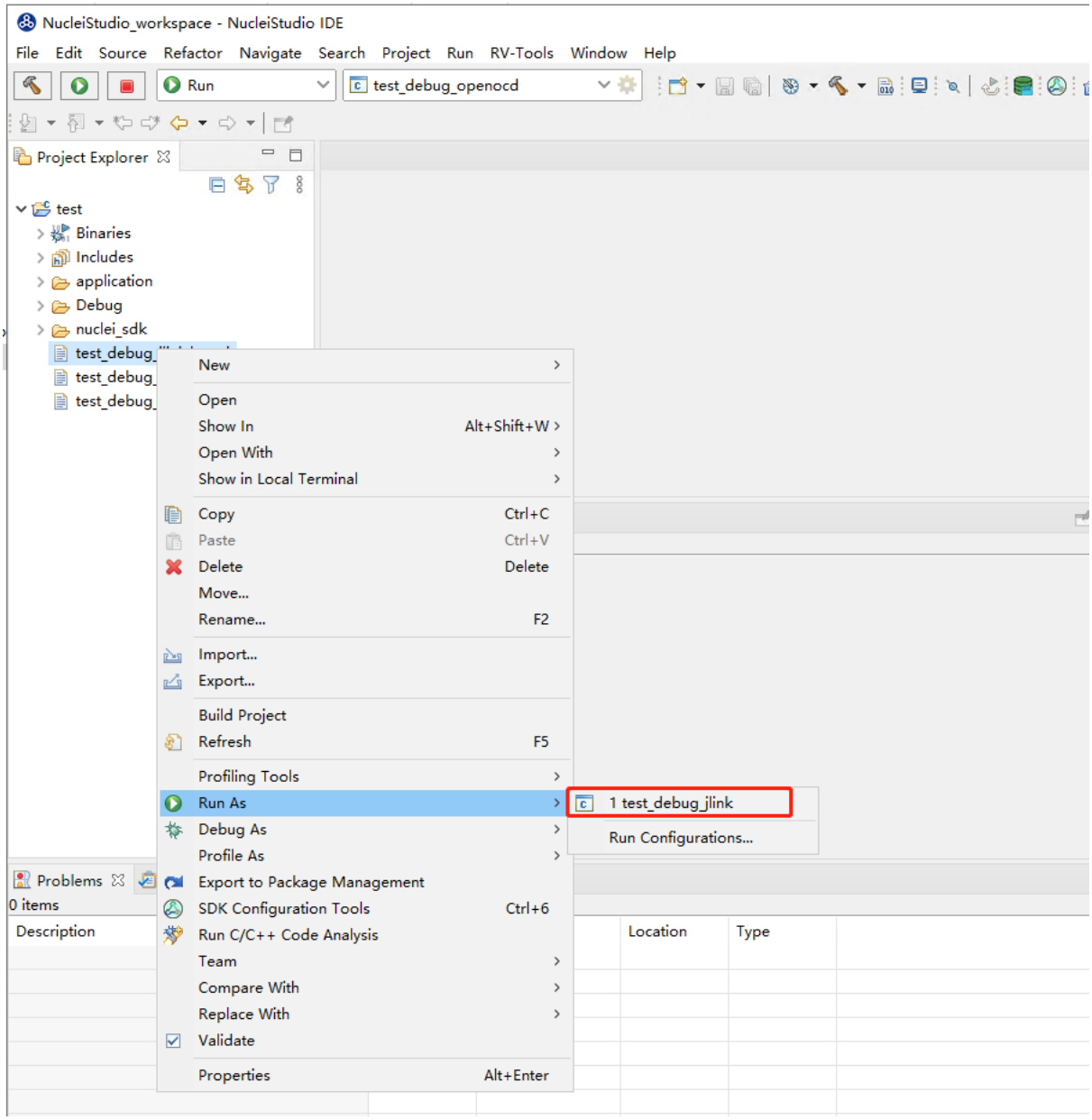
Debug Configuration

Using the Debug Configuration Generated by Nuclei Studio

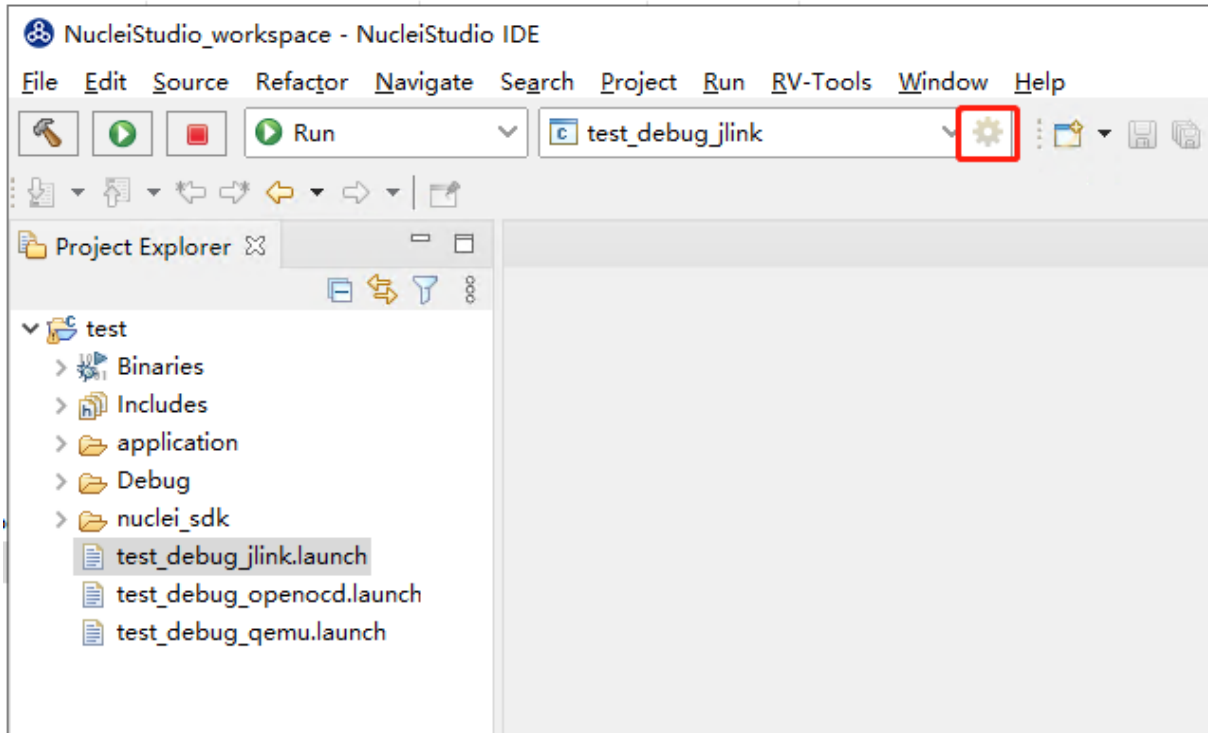
To simplify debugging, Nuclei Studio generates Debug Configuration launch files by default when a project is created, based on the NPK configuration.

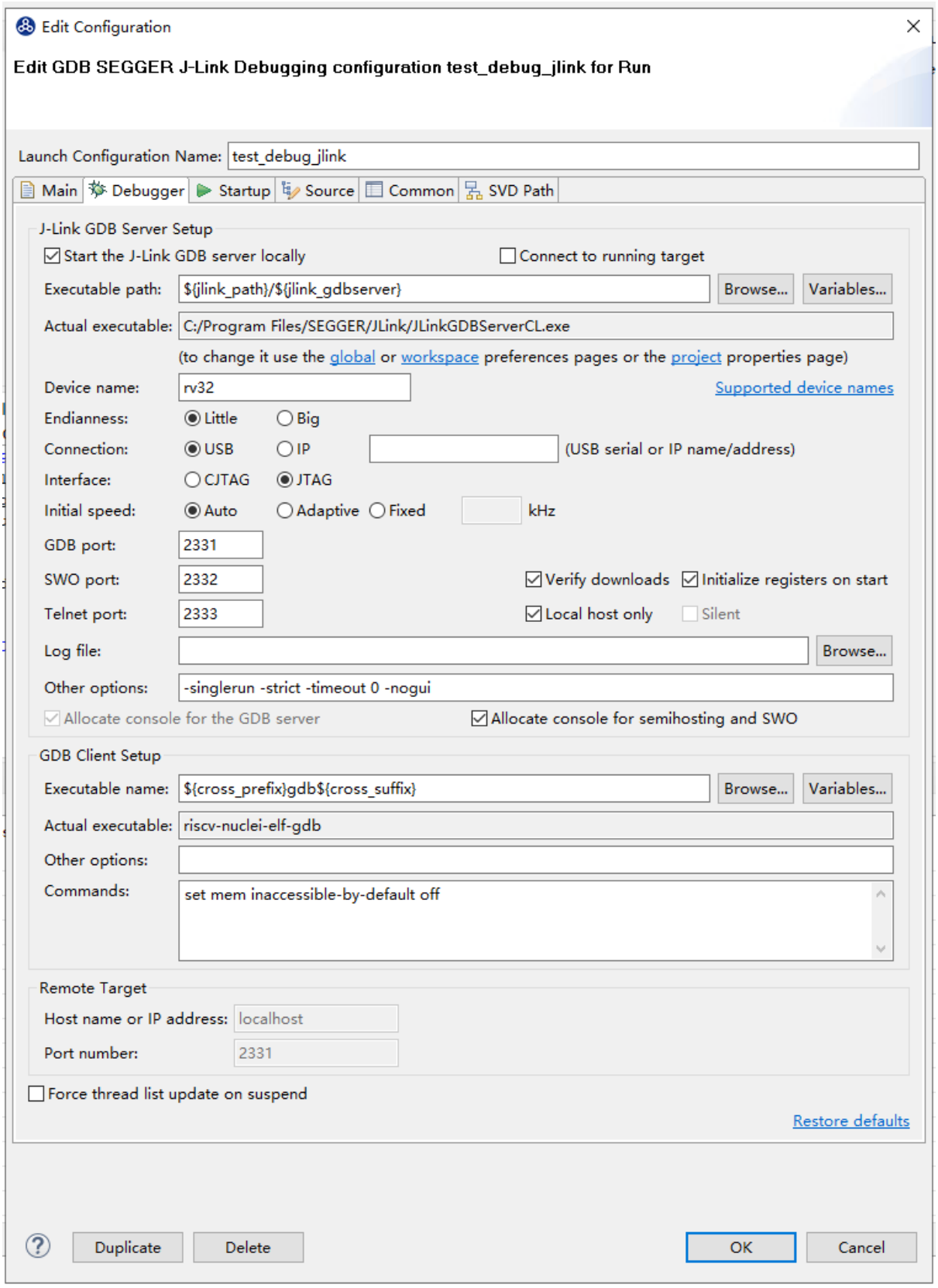


Expand the project, select the corresponding `test_debug_jlink.launch` file, and choose `Run as/Debug as->test_debug_jlink` from the right-click menu to run or debug the project with the corresponding Debug Configuration.



You can view the details of the Debug Configurations in the Launch Bar.

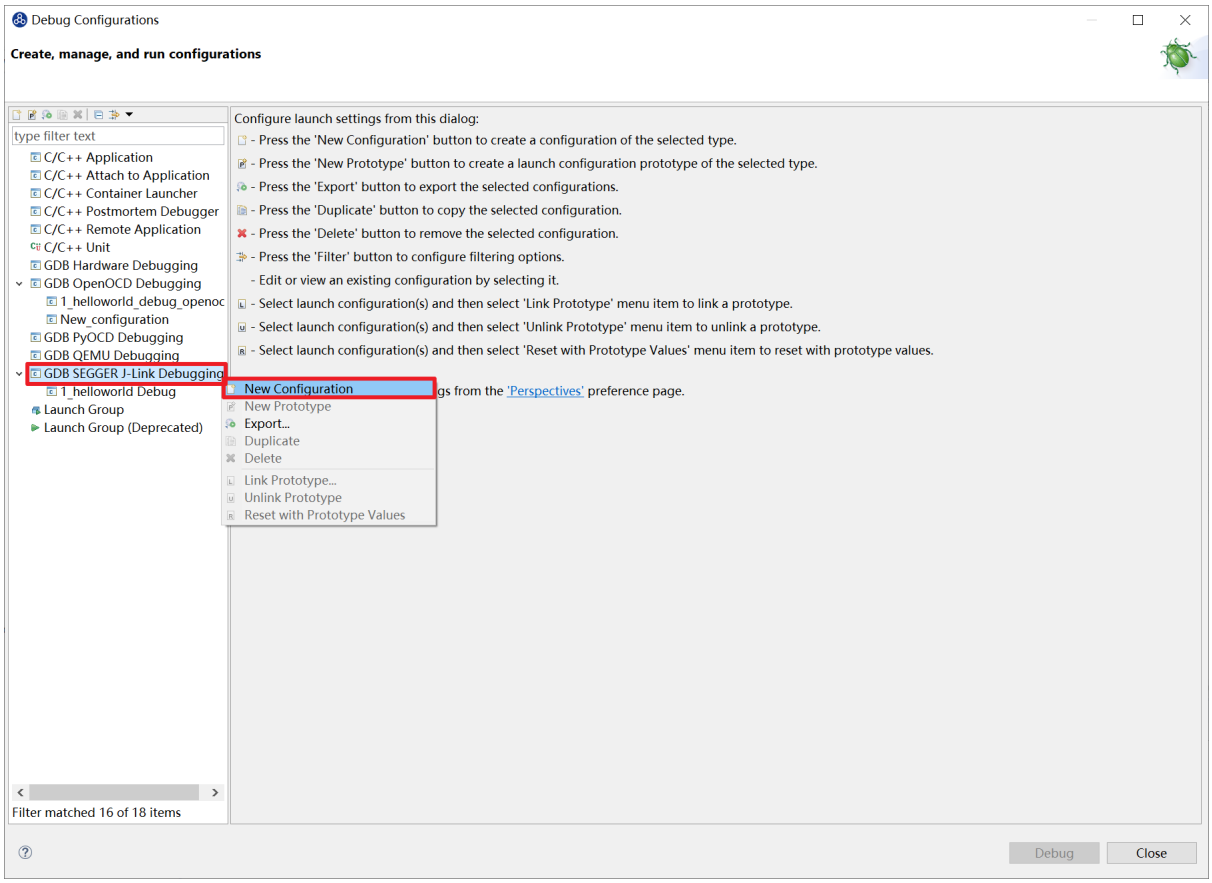

NucleiStudio_workspace - NucleiStudio IDE
File Edit Source Refactor Navigate Search Project Run RV-Tools Window Help
Run
test_debug_jlink
Project Explorer
test
Binaries
Includes
application
Debug
nuclei_sdk
test_debug_jlink.launch
test_debug_openocd.launch
test_debug_qemu.launch


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

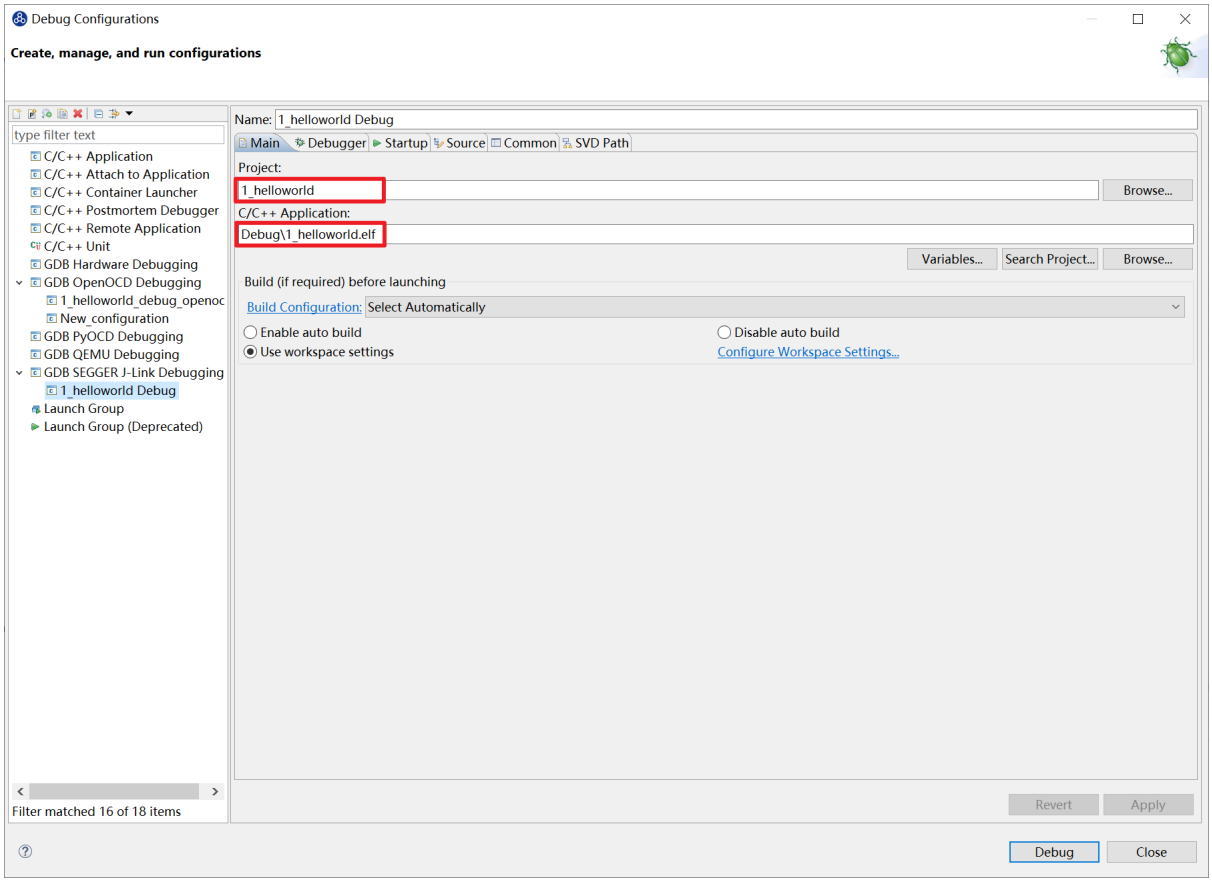
Creating and Configuring a Debug Configuration

To create and configure a Debug Configuration for J-Link debugging and downloading:

Select `Run—>Debug Configurations` in the menu bar. In the dialog that opens, if there is no debug configuration for the current project, right-click `GDB SEGGER J-Link Debugging` and select `New Configuration` to create a new debug configuration `1_helloworld Debug` for the project.



Make sure `Project` is the project you want to debug, and that `C/C++ Application` points to the correct ELF file to debug.



Open the Debugger tab and make sure `Start the J-Link GDB server locally` at position 1 is selected.

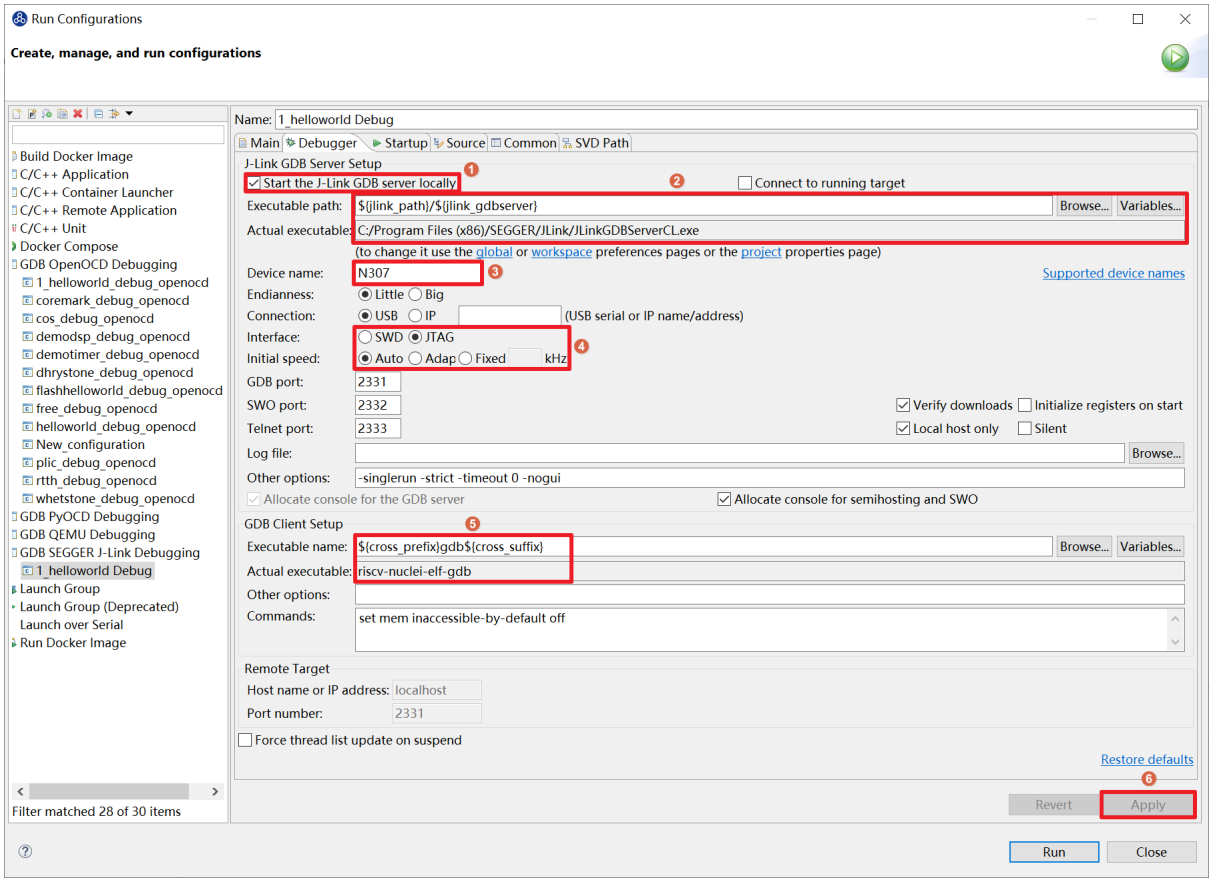
Position 2 must point to the correct path of JLinkGDBServerCL.exe.

Position 3 is the core in use; N307 is used as an example here, so enter N307. If you use the RV-STAR development board, enter GD32VF103VBT6 here. For other development boards, see the J-Link Supported Devices page: https://www.segger.com/downloads/supported-devices.php

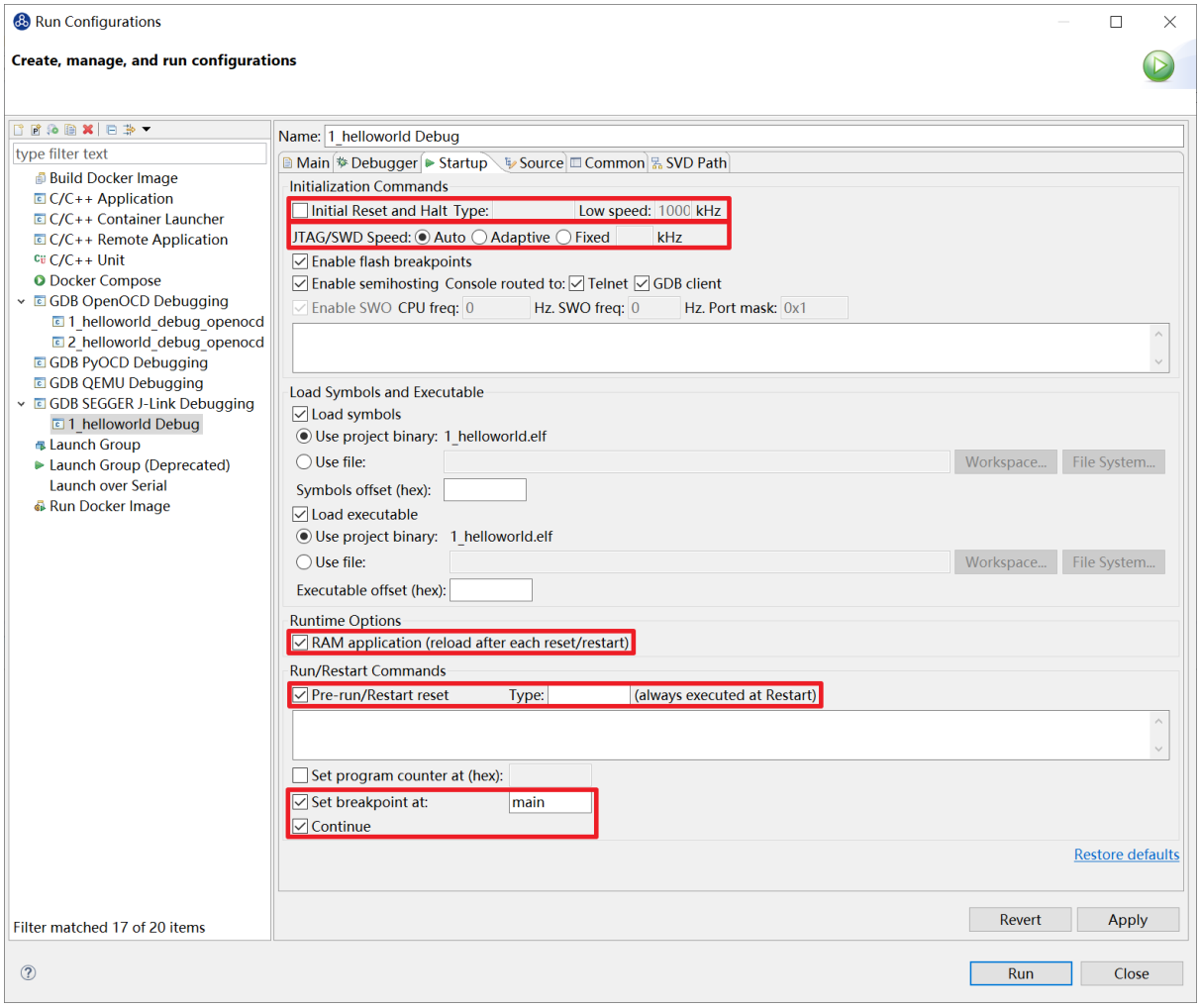
At position 4, select `Interface` as JTAG and `initial speed` as Auto.

At position 5, confirm that it matches the GDB settings in use.

If you modified anything, click `Apply` at position 6 to save.



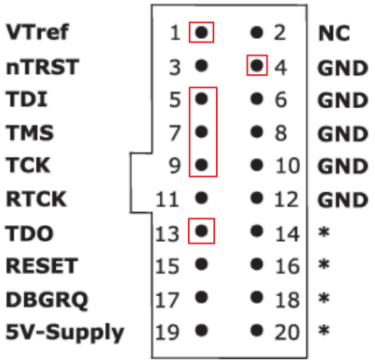
Open the `Startup` tab, make sure JTAG/SWD Speed is Auto, that `set Breakpoint at main`, `Continue`, `Pre-run/Restart reset`, and `RAM application` are checked, and that the `Initial Reset and Halt` option is unchecked.



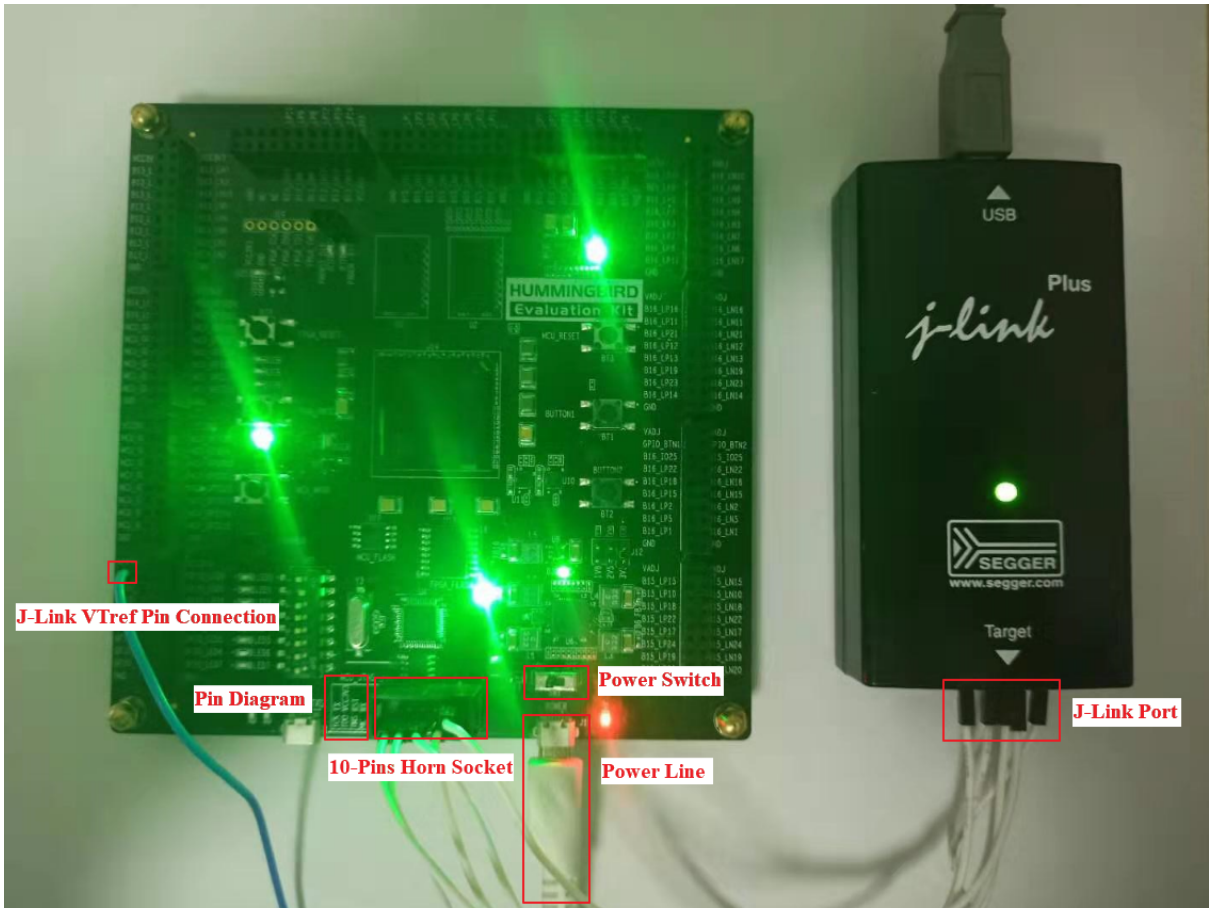
If you changed any of the above settings, click `Apply` at the bottom right to save them; if nothing changed, click `Close`.

Debugging the Program on the Prototype Board

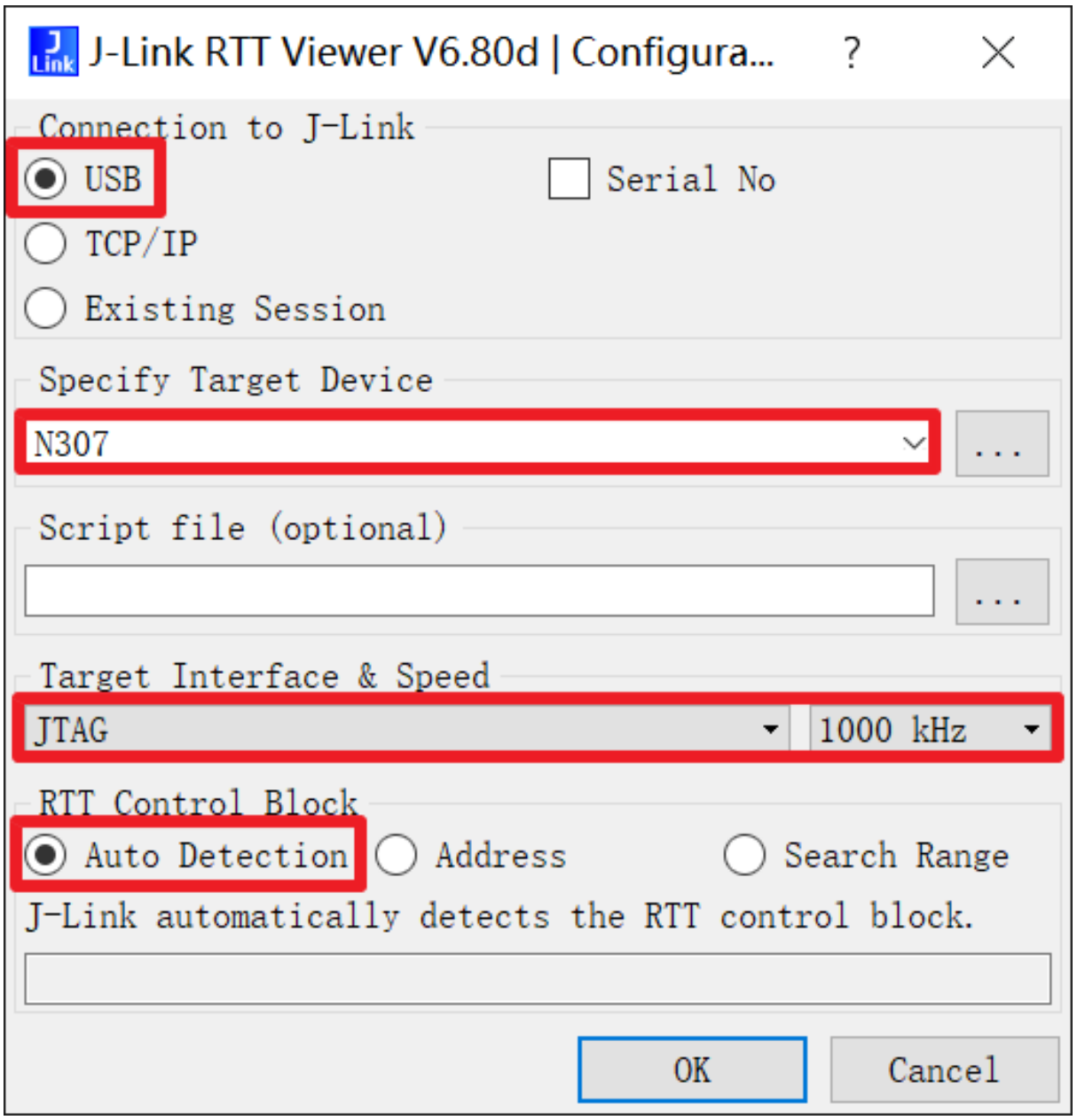
Debugging with J-Link on the HummingBird Evaluation Board requires jumper wires. Connect J-Link to the board at the position marked with the red box.



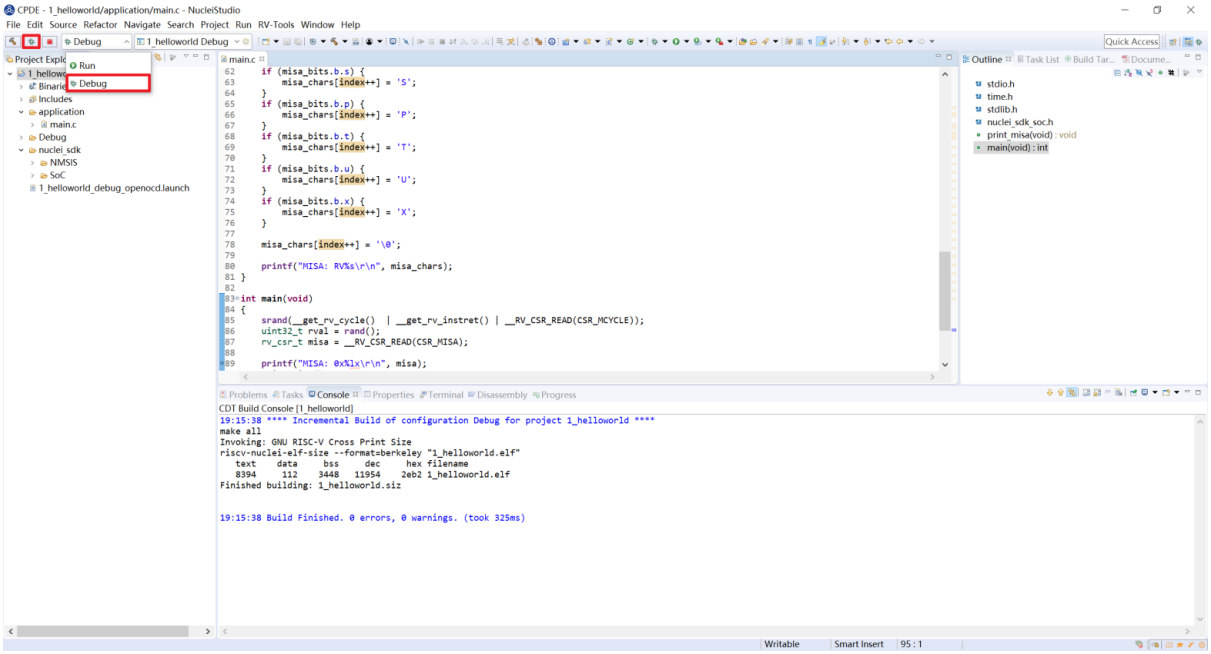
Connect VTref to the `V3.3` connector on the board and the other pins to the JTAG interface. Each pin's silkscreen label is printed right next to it, so connect the pins one-to-one accordingly. The final physical connection is shown below.



Before debugging on the board, if you use serial port printing, connect the UART pins on the JTAG header to your host, and then open the serial port to observe the printf output. If you use RTT printing, open `J-Link RTT Viewer` to view the printf output. Configure it as shown in the figure and select USB as the connection method. Set Specify Target Device according to the core in use; N307 is used as an example here. Set Target Interface & Speed to `1000kHz`, which can be adjusted based on actual use. Select `Auto Detection` for RTT Control Block.

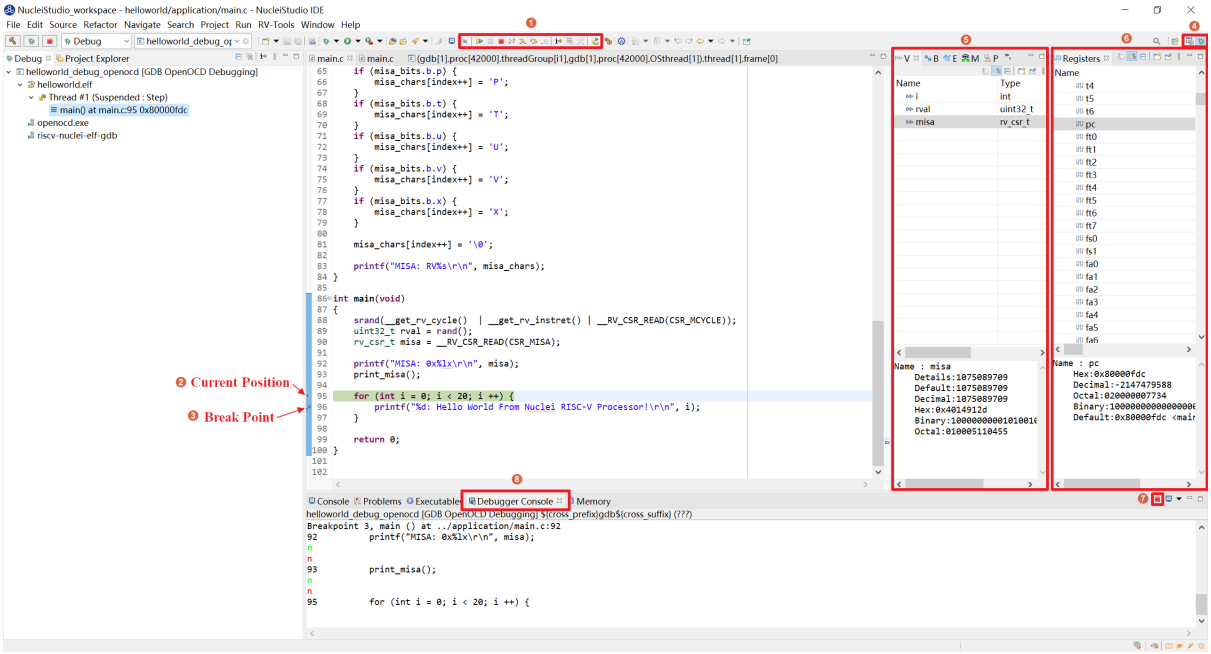


Select Debug in the Launch Bar drop-down box. The icon on the left then changes to a bug icon; click it to enter debug mode and download the program to the development board.



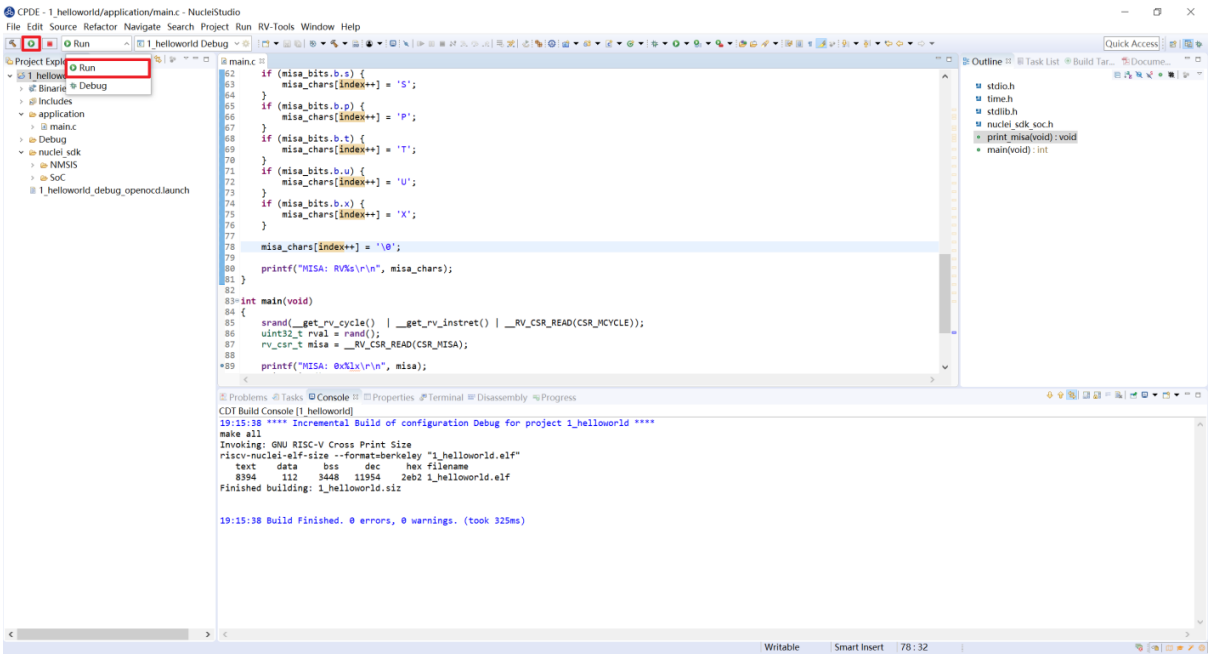
If the program downloads successfully, Nuclei Studio opens the debug perspective.

- At marker 1: step, run, and assembly-level debugging functions.
- At marker 2: the arrow indicates the current execution position of the program.
- At marker 3: double-click in the left margin of a code line to set a breakpoint on that line; double-click again to remove the breakpoint.
- At marker 4: switch between edit mode and debug mode here.
- At marker 5: in-function variables are displayed here.
- At marker 6: register values are viewed here.
- At marker 7: click the red button here to exit debug mode.
- Below marker 8: GDB console commands can be used here for debugging.

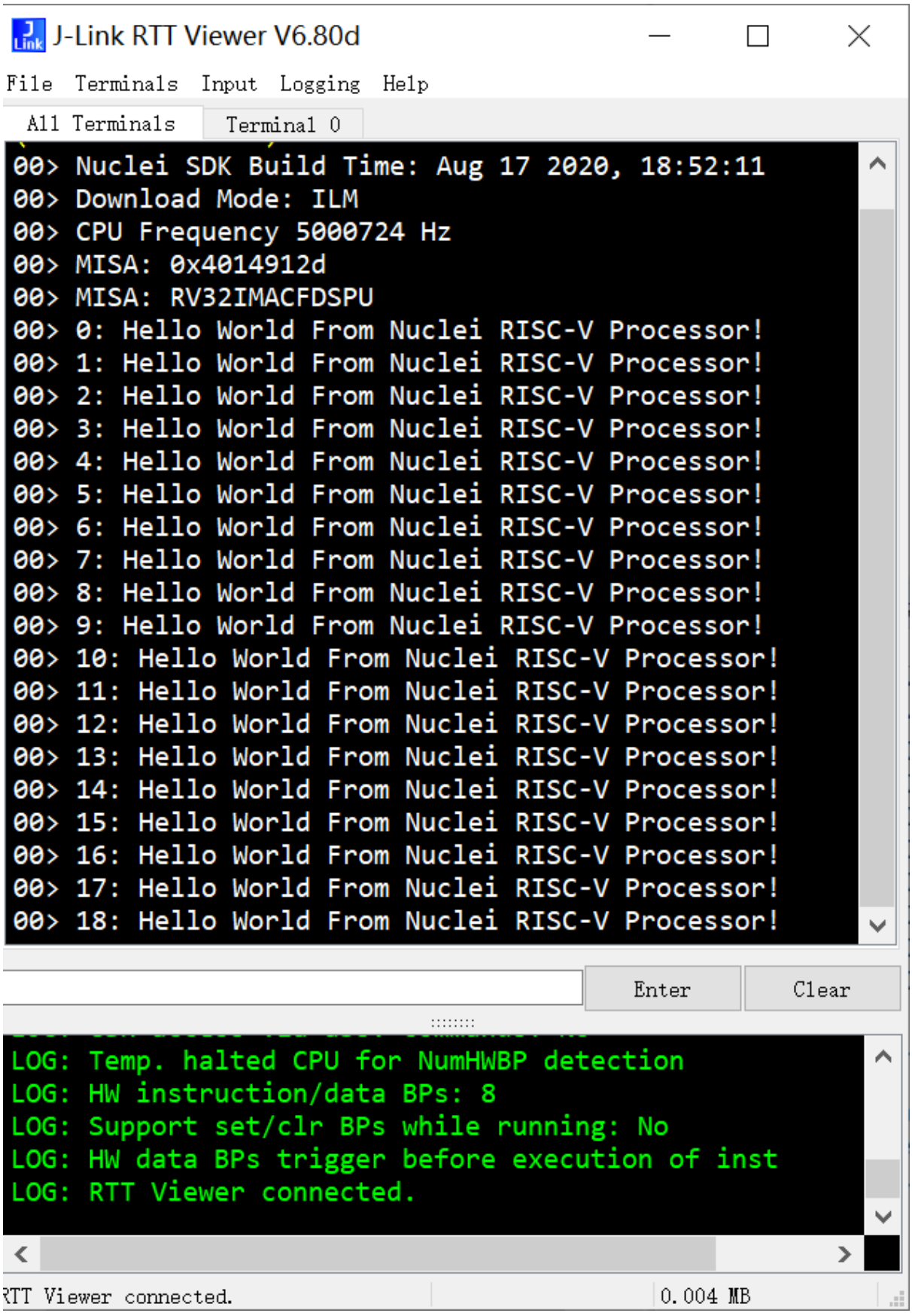


Downloading and Running the Program

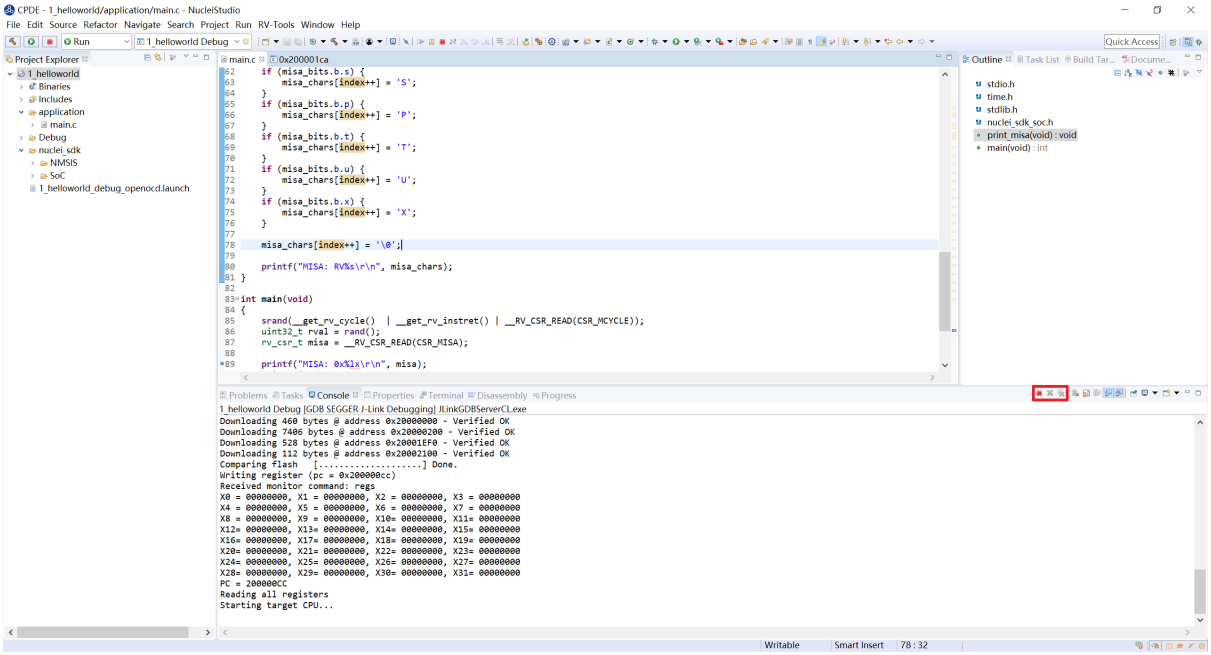
After debugging confirms there are no problems, the program can be downloaded to the development board. Click the drop-down box to switch to run mode; the icon on the left then changes to a green run button. Click it to download the program to the development board and run it.



Because debugging and downloading use the same configuration file, no further setup is required. The RTT Viewer correctly prints messages such as helloworld.



To disconnect, click the red button in the console view.



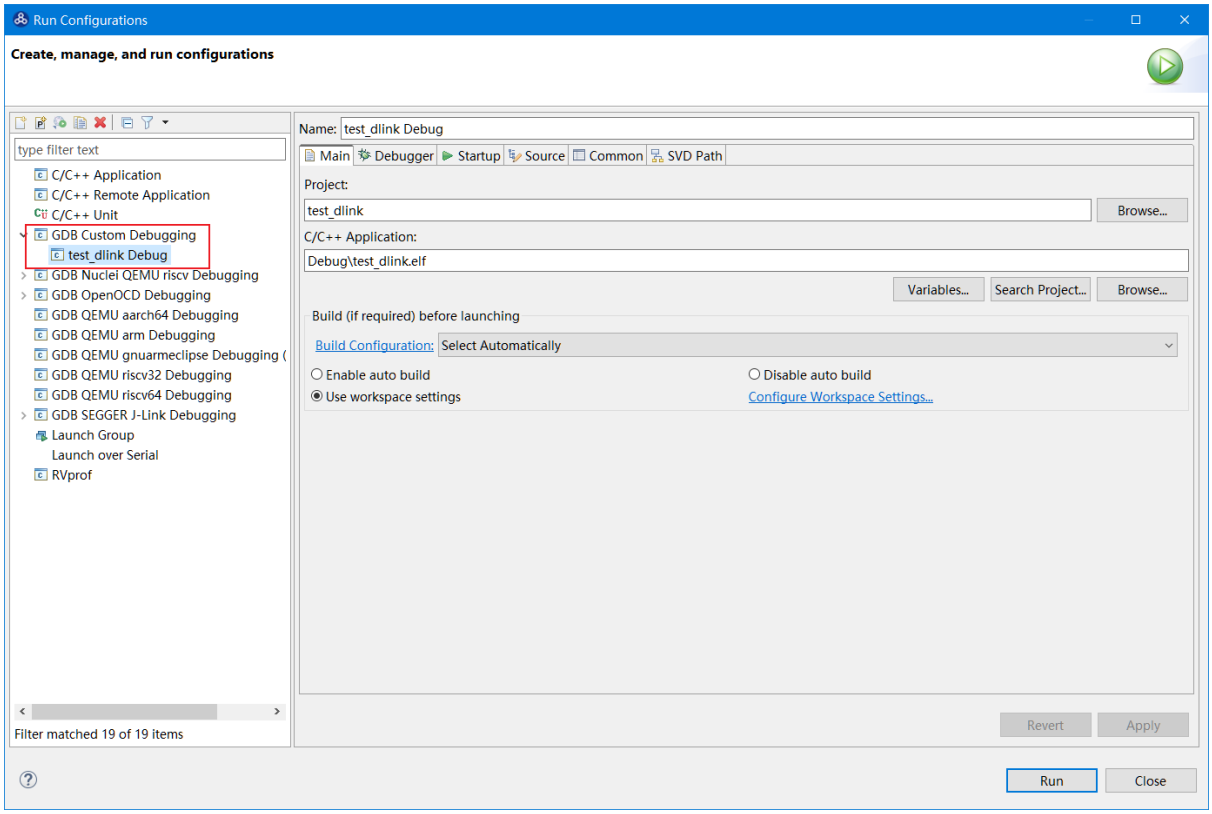
2.9.4 Debugging and Running a Project with DLink

DLink is a RISC-V debugger developed by Nuclei System Technology based on RV Link, with iterative feature upgrades on top of RV Link that make it better suited to Nuclei Studio use cases. DLink currently supports debugging and running single-core RISC-V projects only, and is already in mass production. The actual device is shown below. For the DLink firmware download, see https://github.com/Nuclei-Software/nuclei-dlink/wiki/upload-dlink-firmware.

Note: The **Nuclei WeChat video channel** has a video on **how to debug a program with DLink in Nuclei Studio**. Search for the **Nuclei video channel** in WeChat to view the related content.



To debug with DLink, select `Run -> Run Configurations` in the Nuclei Studio menu to open the Run Configurations page, and configure a DLink Debug Configuration: double-click `GDB Custom Debugging` to create a new configuration, and configure the Main tab as follows:



To install the driver on Windows:

Open the GD website, search for GD32VF1, find GD32 Dfu Drivers in the list, and download and install it.

Address: 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				

To install the driver on Linux:

- 1: Connect the development board to the Linux host and make sure the USB device is recognized.
- 2: Run `lsusb` in the console to view the device information; for example:

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

- 3: In the console, run `sudo vi /etc/udev/rules.d/50-dlink.rules` to open the 50-dlink.rules file, enter the following content, save and exit, and then run `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: Disconnect the debugger and reconnect it.
- 5: Run `ls /dev/ttyACM*` to view the ttyACM information; for example:

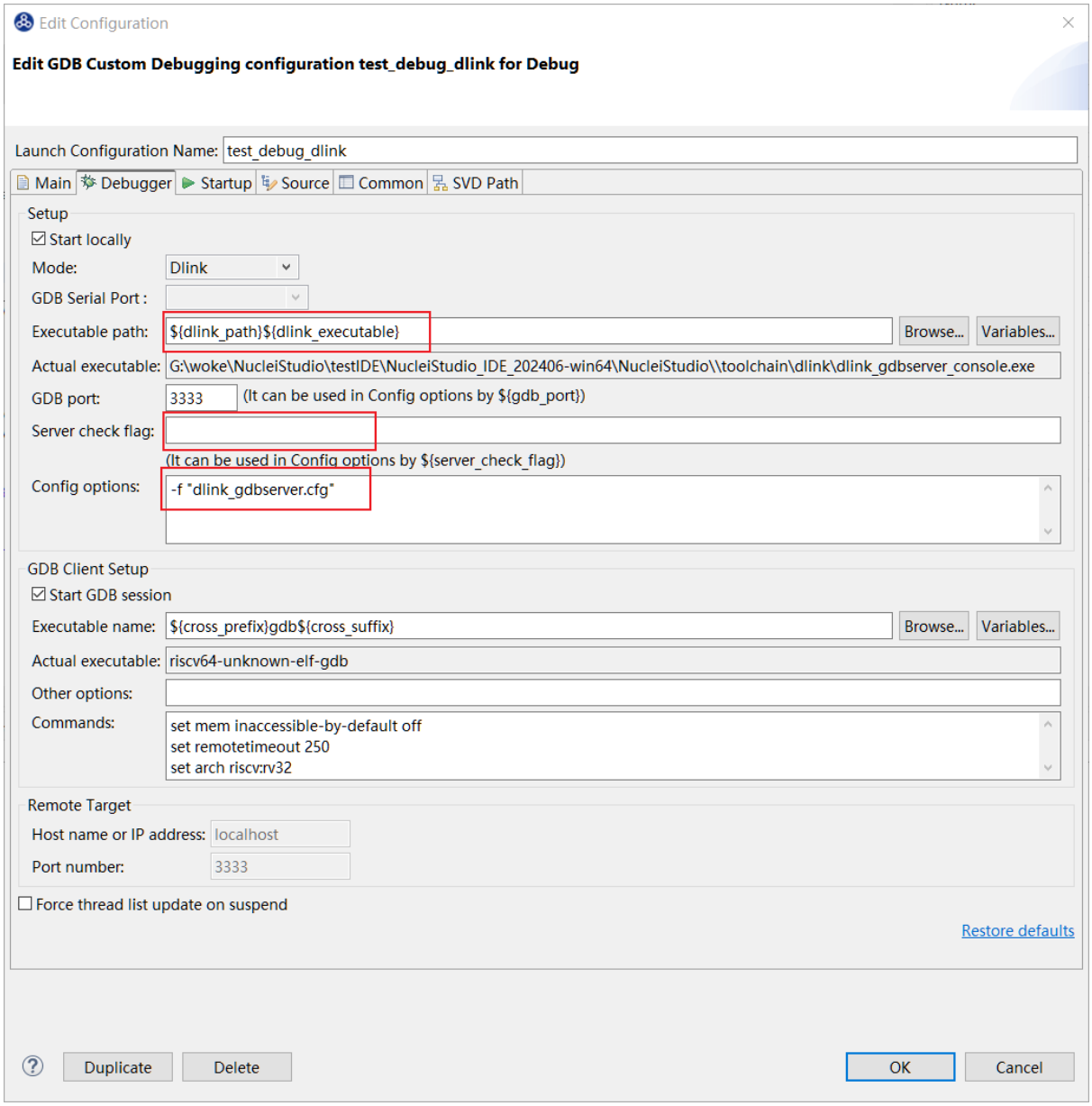
```
/dev/ttyACM0 /dev/ttyACM1
```

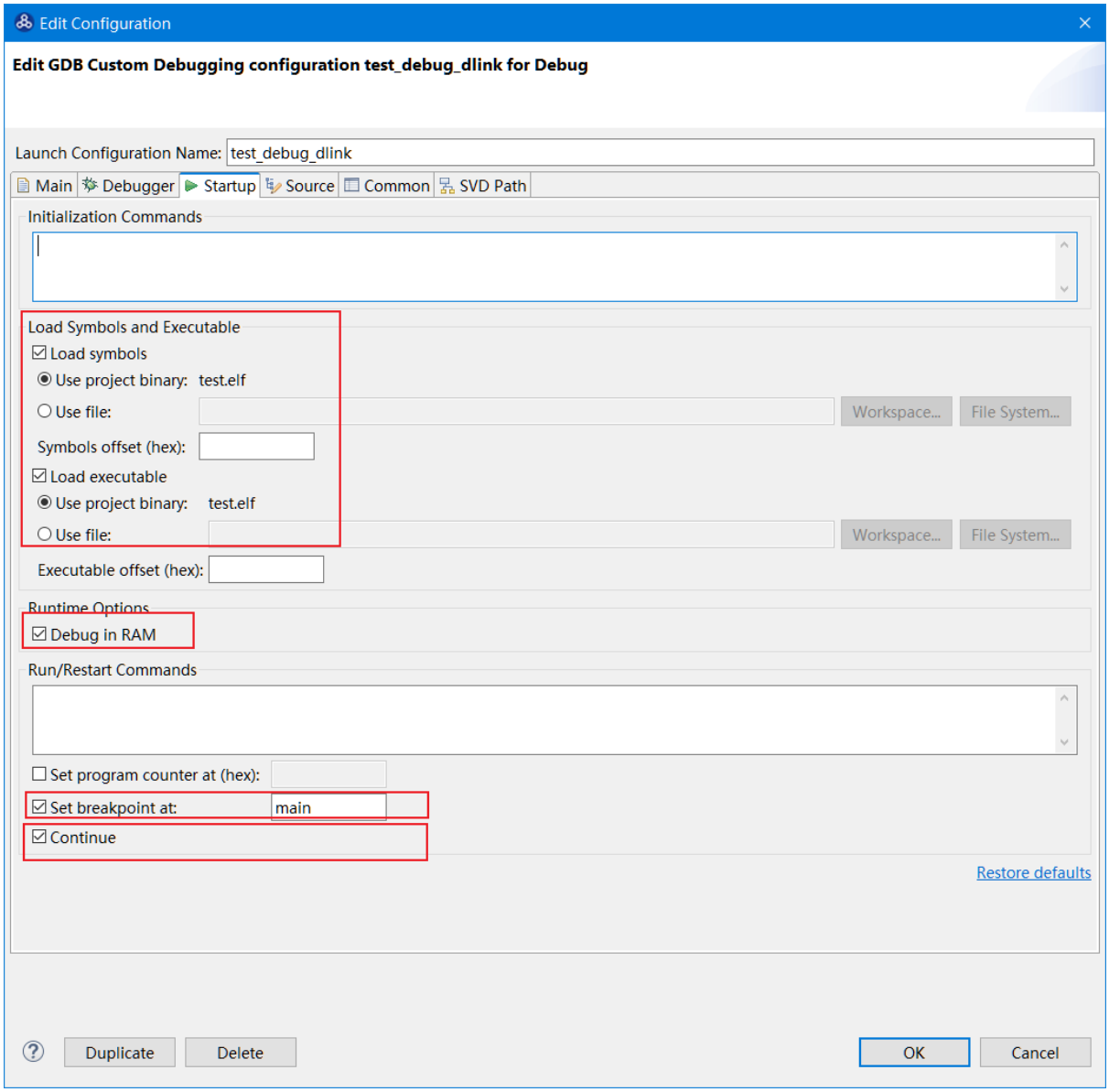
- 6: Run `ls -l /dev/ttyACM0` to view the group information; an example output is shown below. As shown, `ttyACM0` belongs to the `dialout` group. Next, add your user to the `dialout` group (the group name may differ between environments; adjust it to your actual situation). Run `whoami` to view the current user name, denoted below as `<your_user_name>`.

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

- 7: Run `sudo usermod -a -G dialout <your_user_name>` to add yourself to the dialout group. You must reboot or log out for the change to take effect.
- 8: Run `groups` to confirm that the current user belongs to the `dialout` group. If `dialout` appears in the output, the user was added successfully; otherwise, try rebooting.

Then configure the Debugger tab as follows. Custom Debugging supports multiple modes; because DLink is used here, select DLink. Nuclei Studio uses `Server check flag` to confirm whether the server has started normally: when the Custom GDB Server starts normally, it outputs a string, and Nuclei Studio checks this string to confirm that the Custom GDB Server has started; with DLink, this field can be left empty. In Config options, configure the corresponding connection file `dlink_gdbserver.cfg`; a reference configuration file is available in the `<NucleiStudio>/toolchain/dlink` directory.

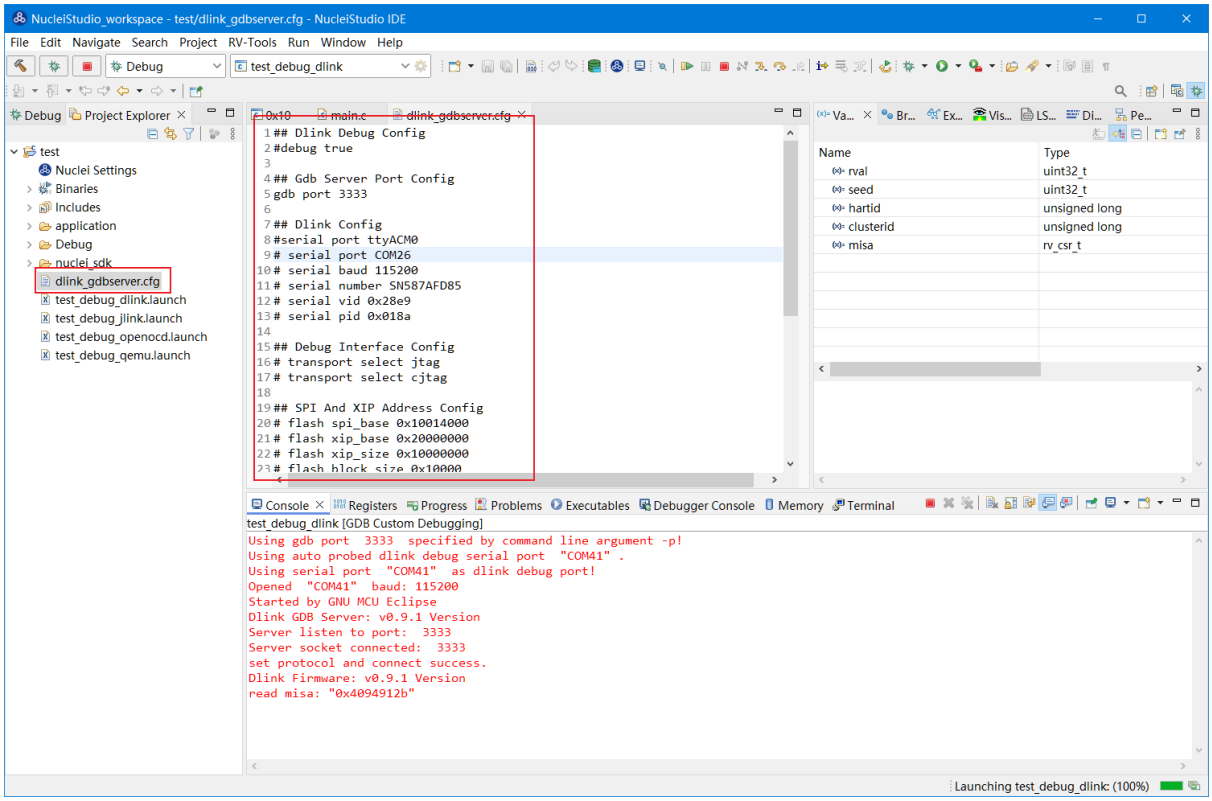




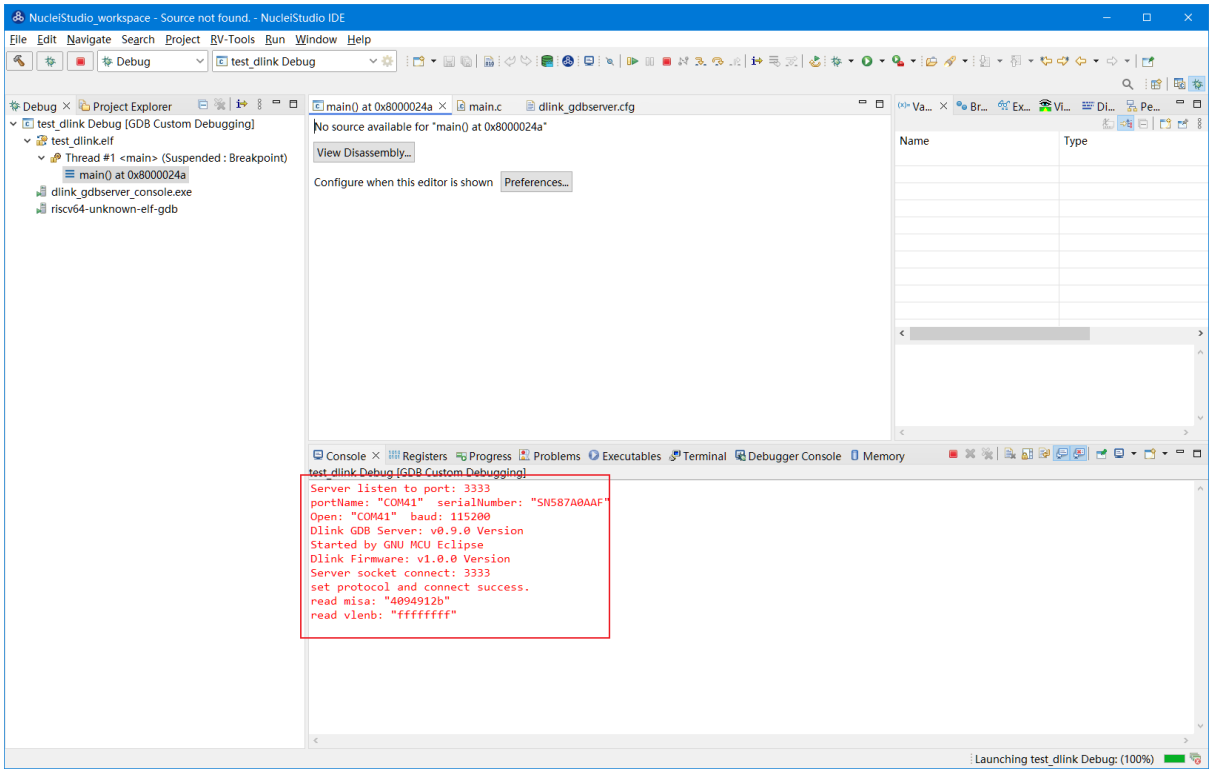
On Windows, once the DLink connects the PC and the development board, a green LED and a blue LED light up, indicating that the DLink is working properly. If they do not, press the NRST button to reset it.



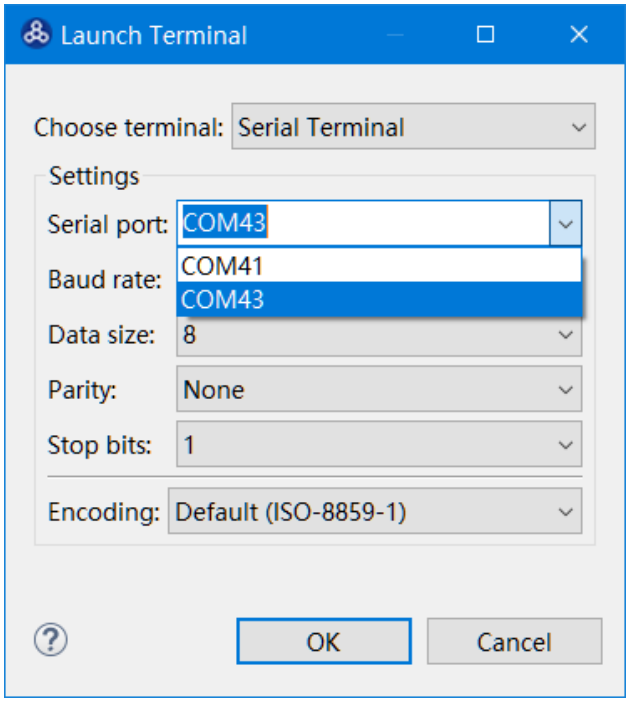
After the DLink is connected, two COM ports appear in the serial port tool. One COM port is used for serial output —generally the COM port with the lower number is used for debugging —and the other is used for serial data exchange. Specify the COM port used for data exchange in `dlink_gdbserver.cfg` by configuring `serial port` and `serial baud`, for example `serial port COM1` and `serial baud 115200` on Windows, or `serial port ttyACM0` and `serial baud 115200` on Linux. If they are not configured, DLink uses the COM port with the lower number and its corresponding baud rate as the defaults. The following is a reference configuration on Windows:



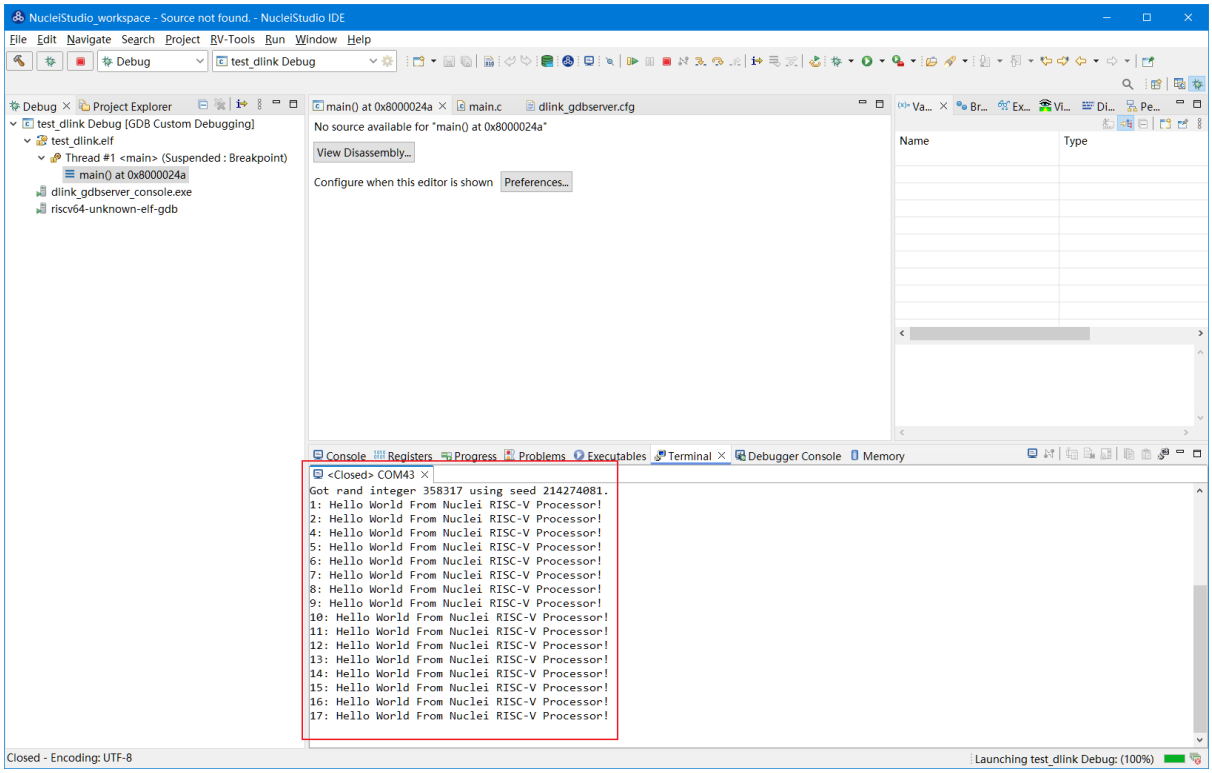
Start debugging. If the configuration is correct, the Console shows the following output, and the DLink lights up green.



Use a serial port tool to connect to the other serial port.



If the serial port shows the correct output as expected, DLink debugging works properly. The remaining steps are largely the same as with OpenOCD.

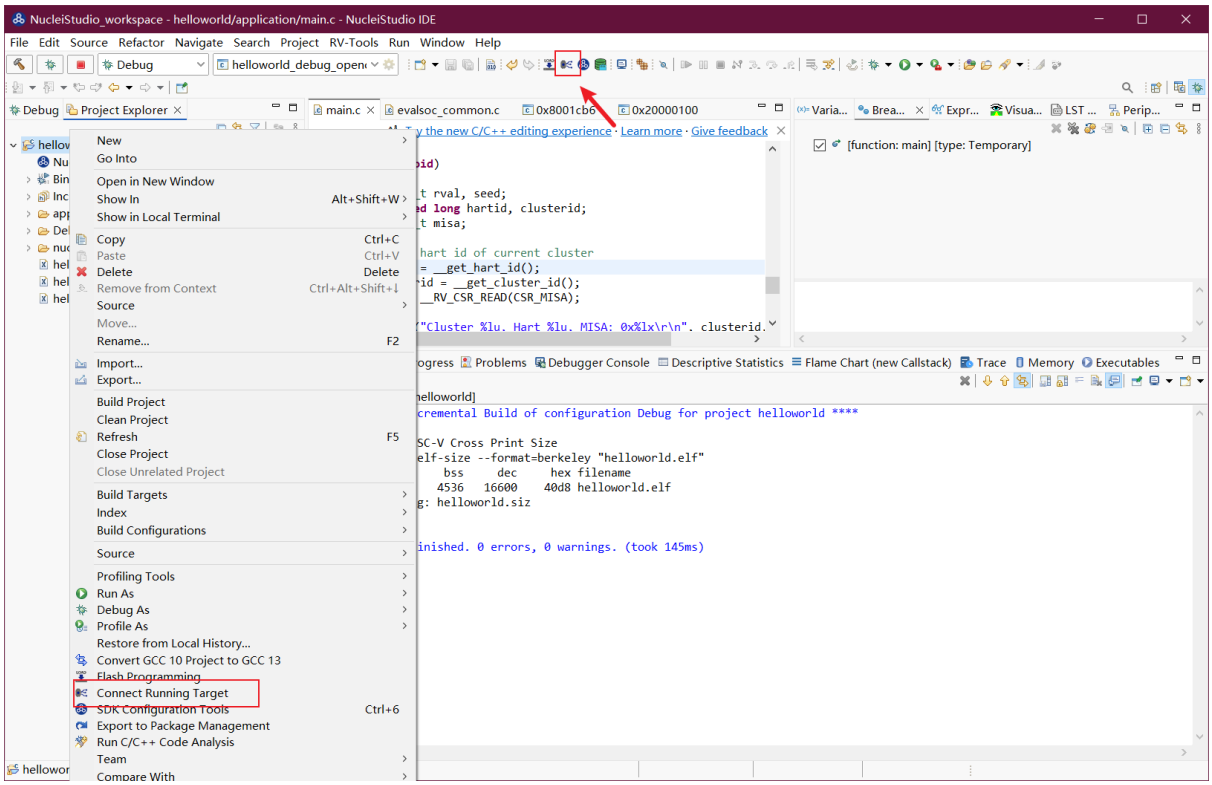


2.9.5 Connect to Running Target

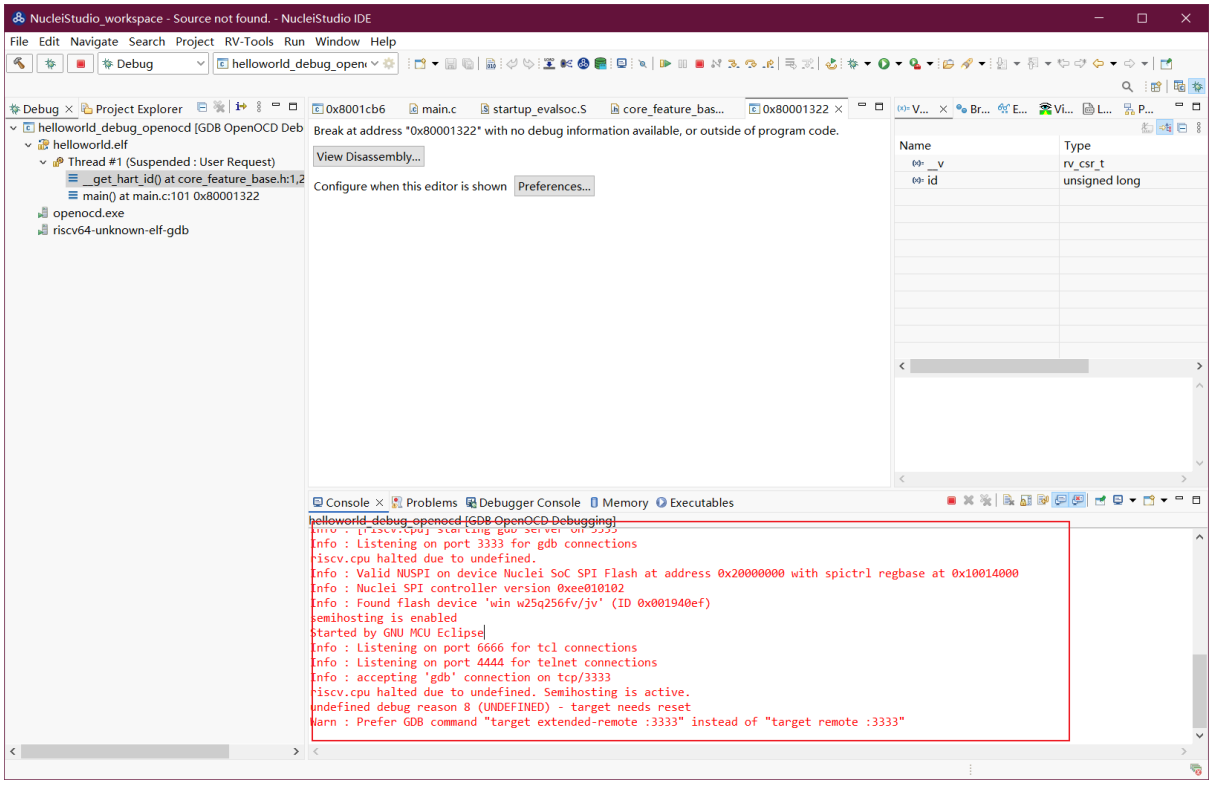
To support connecting directly to a development board, Nuclei Studio provides the `Connect to Running Target` feature. This feature connects directly to a running hardware target board, allowing you to perform operations such as reading board-related information, which greatly improves development efficiency.

In Nuclei Studio, `Connect to Running Target` supports only OpenOCD and J-Link. The following example demonstrates its usage with OpenOCD.

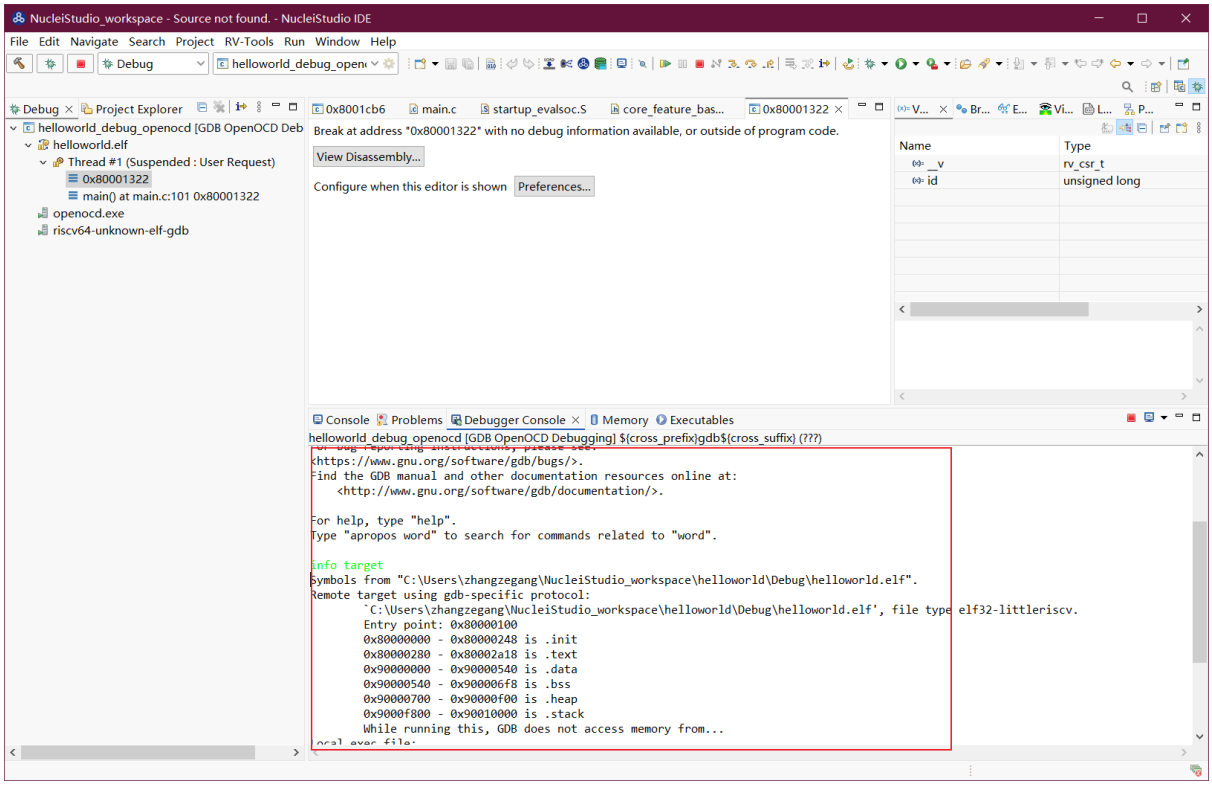
Using the `Connect to Running Target` feature is very simple. Connect the development board and create a demo project. After the project is built, select the project and click `Connect to Running Target` from either the right-click menu or the IDE tool menu.



In the console window, you can see GDB trying to connect to the development board.



Once the IDE is connected to the development board, you can use the IDE ' s `Debugger Console` tool to send GDB commands to view the board ' s current information.



2.9.6 CPU Information

CPU Information View

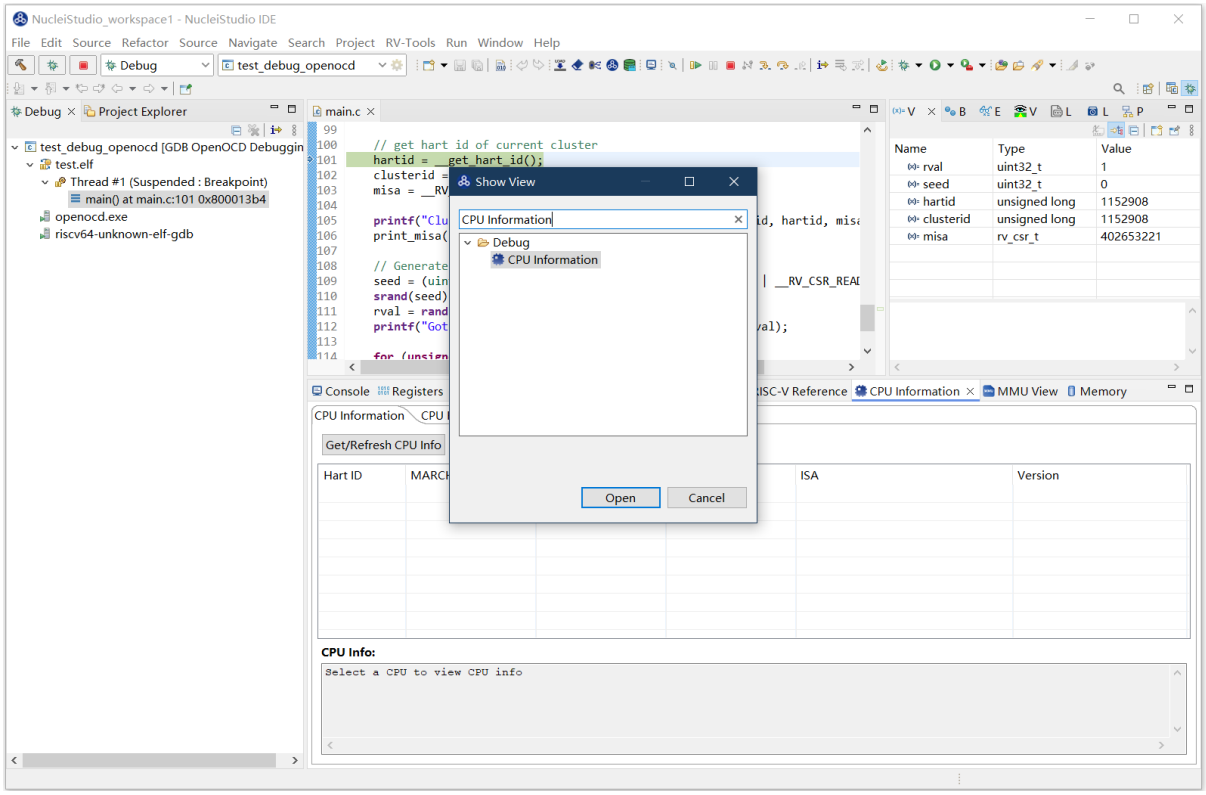
CPU Information targets Nuclei-series RISC-V cores and provides two tabs:

1. CPU Information: shows `MARCHID / MHARTID / MIMPID / ISA / Version` per hart in single-core/multi-core, and decodes the `MISA` extension bits and Nuclei configuration CSRs (`mcfg_info` / `micfg_info` / `mdcfg_info`) bit by bit in the details panel;
2. CPU Fault Analysis: for each detected hart, categorically shows the current mode, trap source (`mcause` / `scause`), trap vector (`mtvec` / `stvec`), M/S-mode CSRs, delegation status, page faults, etc.

The PC / CSR / page-fault virtual address in the view provide clickable links that jump to the source code, *RISC-V Reference* (page 345), and *MMU View* (page 336) respectively, forming a linked debugging chain.

Note: When an older Nuclei Studio does not have this view built in, you can still use the command way to view CPU Information: send `info reg misa` or `monitor nuclei cpuinfo` in the `Debugger Console`; the latter output must be viewed in the `gdb trace` view. See *View CPU Information via GDB Commands* (page 198) for the complete steps.

Open it via the menu `Window -> Show View -> Other...`, search for `CPU Information`, and click `CPU Information` to open the view. The view has two pages: CPU Information and CPU Fault Analysis.



Prerequisite: the target must be connected and halted (values are read-only and no register is written).

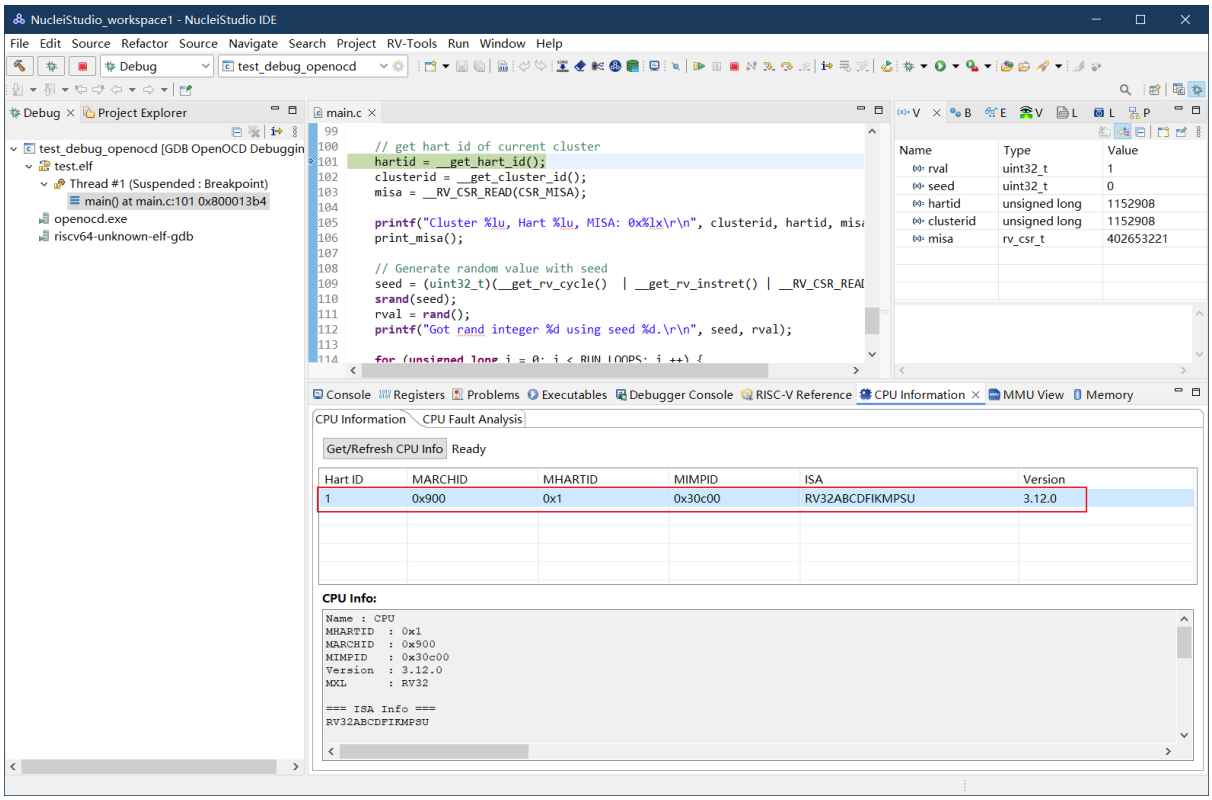
1. Open the view.
2. Start debugging and let the target halt (Suspend or hit a breakpoint).
3. Click `Get/Refresh CPU Info` → the table lists each hart's `MARCHID / MHARTID / MIMPID / ISA / Version`.
4. Select a row → the lower "CPU Info:" panel shows that hart's raw output, MISA decode, and Nuclei configuration CSR decode.

5. When fault analysis is needed, switch to the CPU Fault Analysis tab, click `Analyze Fault`, then click a Hart tab to view.

CPU Information

Example output from UX900 + OpenOCD:

```
Hart ID = 1    MARCHID = 0x900    MHARTID = 0x0    ISA = RV64ABCDFIMSU    Version = 4.
↪2.0
```



Meaning of table columns

Column	Meaning	Remarks
Hart ID	hart index	In multicore, the tab also shows the corresponding GDB thread number
MARCHID	Architecture ID (marchid)	Measured 0x900 on UX900
MHARTID	hart ID (mhartid)	Usually 0x0 for single core
MIMPID	Implementation ID (mimpid)	May be 0x0; shows N/A when unavailable
ISA	Instruction set string	Contains S means supervisor mode is present
Version	CPU version	e.g. 4.2.0

CPU Info detail panel

The panel content is in three segments:

1. The first segment: cleaned raw output, such as `Nuclei CPU Detected: ...`, `Feature: ...`;
2. After `=== RISC-V Info ===`: key-value pairs parsed from `monitor riscv info`;

3. Further on: MISA bit decode + Nuclei configuration CSR expansion —`mcfg_info` (TEE / ECC / ECLIC / PLIC / ILM / DLM / Cache / SMP features), `micfg_info` / `mdcfg_info` (ILM / DLM sizes and Cache geometry parameters).

Determining whether the core supports MMU

You can use this view to judge, but be cautious: check whether the ISA string contains S; more reliably check bit18 of `misa`; but neither is a sufficient condition (in real tests there have been cases where the ISA contains S and misa bit18 is also 1, yet `csrw satp` still triggers an illegal instruction exception). Ultimately the criterion is whether `satp` can be successfully read and written; see *MMU View* (page 336).

Troubleshooting

- Symptom: `MARCHID / MHARTID / MIMPID` are all N/A, but ISA / Version have values. Usually caused by the OpenOCD version or the level of monitor command support. Confirm whether OpenOCD supports the `monitor nuclei cpuinfo` command; the GDB commands and responses sent by the view are echoed to the "Nuclei CPU Info" console, which can be used for comparison.
- Symptom: `MIMPID` is still N/A. Some cores/versions do not provide a valid mimpid value, which is normal and does not affect the use of other fields.

Known limitations

- Values must be taken while the target is halted; no refresh during running.
- Depends on OpenOCD's monitor command support; different versions may return different content, and the view has done fault-tolerant format parsing.
- Values are read-only and no register is modified.

From opening the view to analyzing a fault

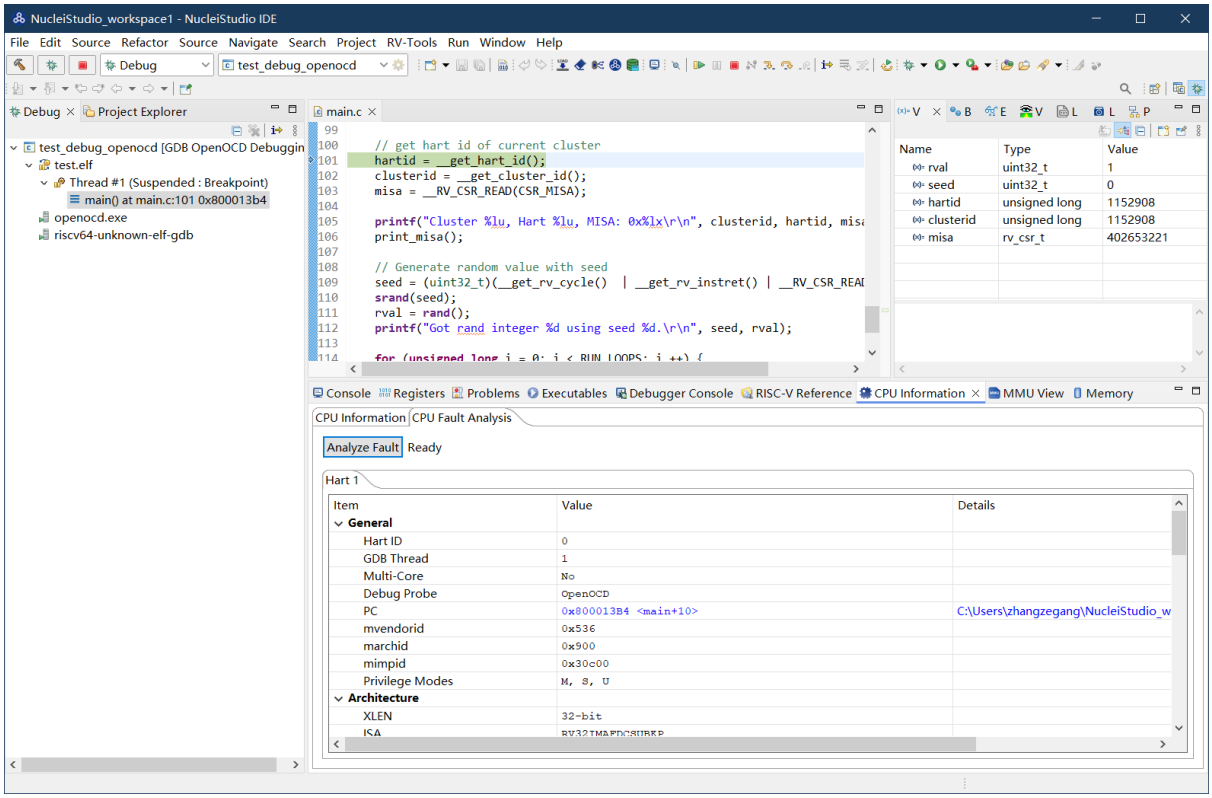
1. The target is connected and halted (CPU Information also only takes values while the target is halted).
2. Open the view: menu `Window -> Show View -> Other... -> CPU Information`.
3. Click `Get/Refresh CPU Info`; the table lists each hart's MARCHID / MHARTID / MIMPID / ISA / Version.
4. Select a row; the lower "CPU Info:" panel shows the raw output, MISA bit-field decode, and Nuclei configuration CSR expansion.
5. Fault analysis: switch to the CPU Fault Analysis tab, click `Analyze Fault`, then click a Hart tab to view the current mode, trap source, trap vector, M/S-mode CSRs, etc. by category tree.
6. Linked debugging: the PC / CSR / page-fault virtual address in the panel are blue clickable links, jumping to the source code, the RISC-V Reference view, and the MMU view respectively.

CPU Fault Analysis

The CPU Fault Analysis tree organizes the analysis results by category nodes: General / Architecture / Current Mode / Trap Analysis / Trap Vector / M-mode Registers / S-mode Registers / Delegation Status / Read Errors.

Key points:

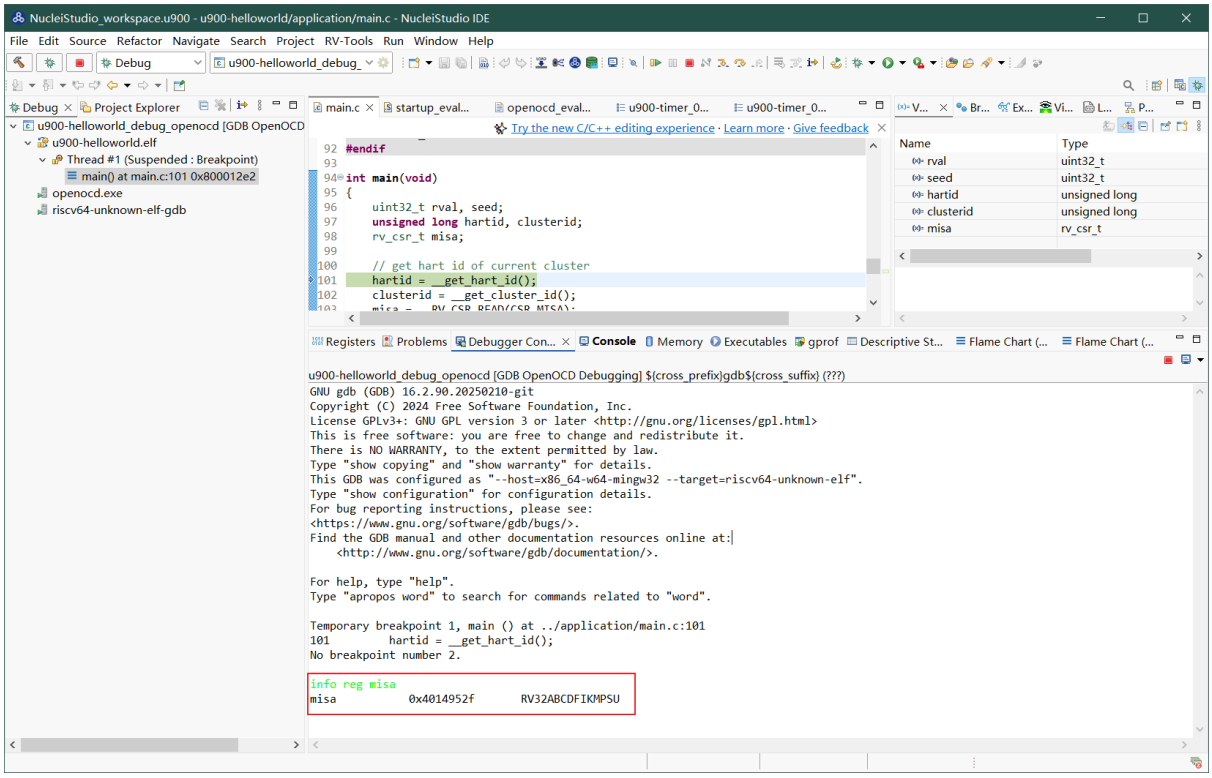
- The trap type is given as a text description rather than just an `mcause` number, e.g. `Load access fault`, `External interrupt (ECLIC ID n)`, `NMI Exception`;
- In CLIC/ECLIC mode, `mpp` / `mpie` / `minhv` / `mpil` are additionally attached;
- The blue links can be double-clicked: PC → jump to the source line; CSR → open the RISC-V Reference view to inspect that CSR; the page-fault Trap Value → open the MMU view to translate that address.



View CPU Information via GDB Commands

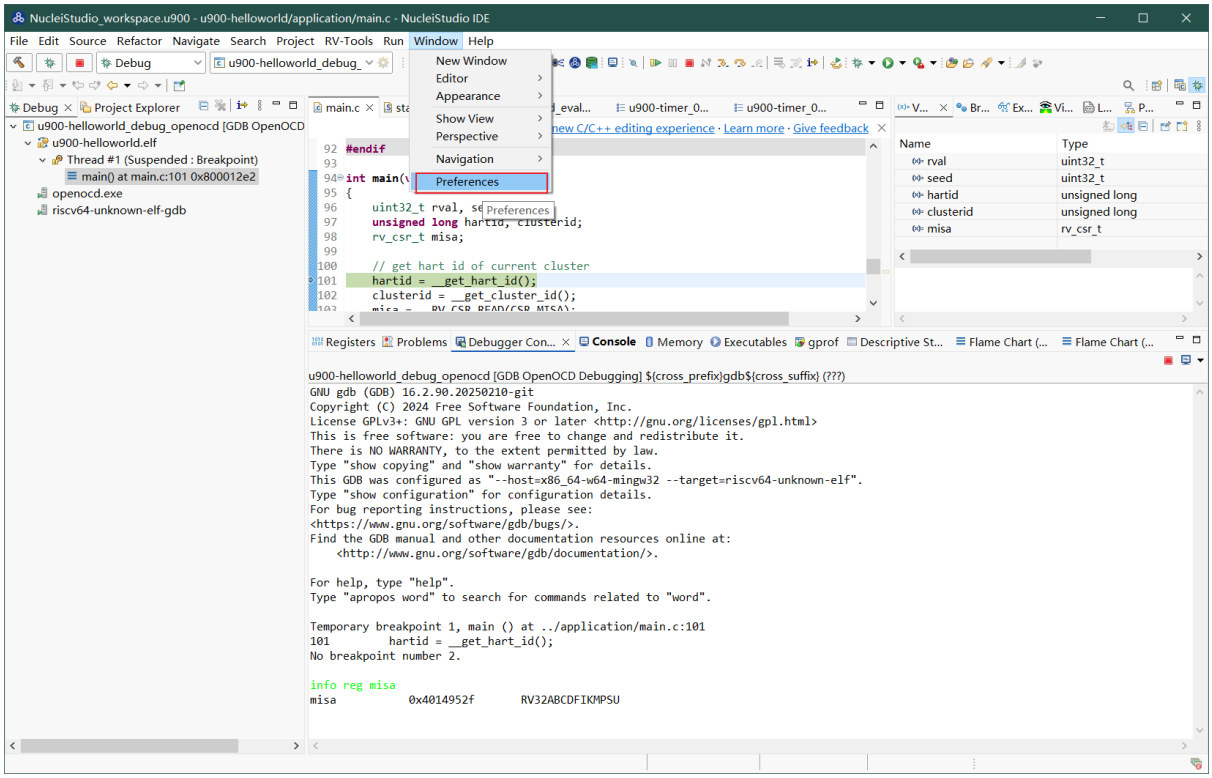
Note: The newer Nuclei Studio has the **CPU Information** feature built in; it is recommended to view and decode each hart ' s information directly through the view. See *CPU Information* (page 195). The command approach recorded in this section applies to older versions that do not have this feature built in.

After connecting to the running target, you can send the command `info reg misa` through the IDE ' s `Debugger Console` tool to view the `misa` information of the current CPU. The `misa` value is displayed directly in the `Debugger Console`.

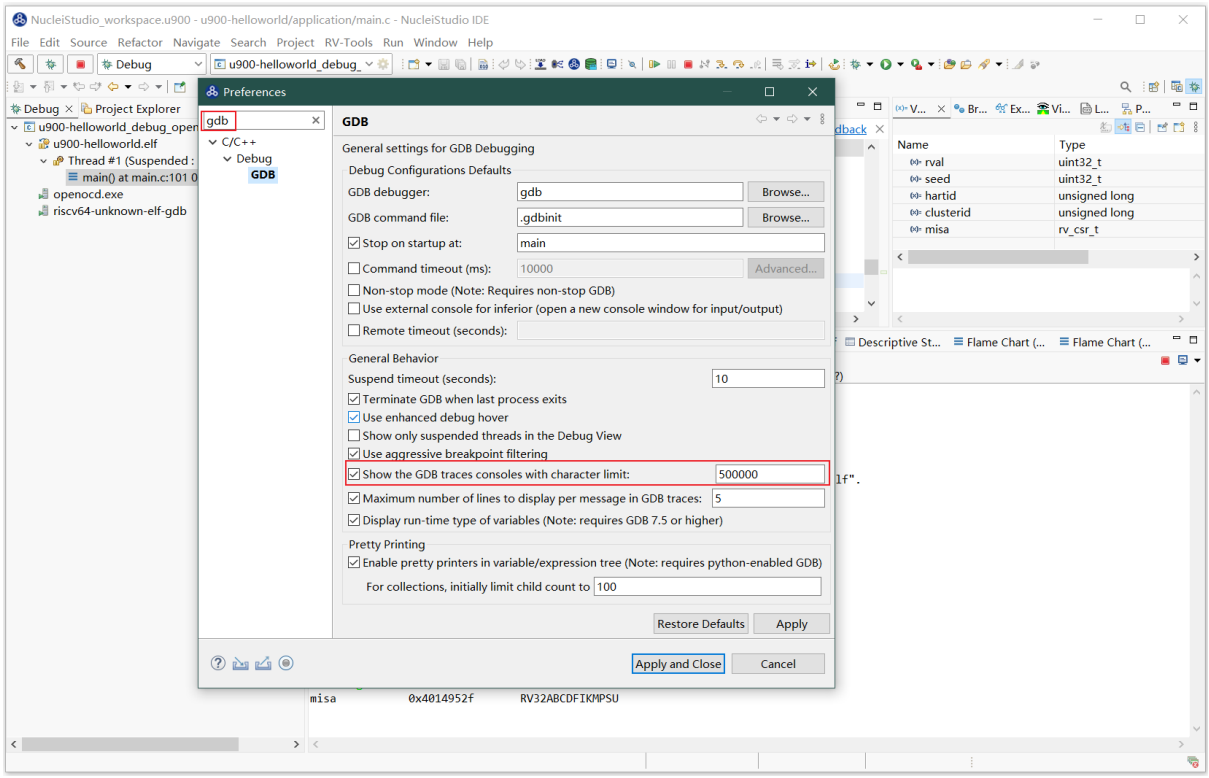


You can also send the command `monitor nuclei cpuinfo` through the IDE's `Debugger Console` tool to view the current Nuclei CPU information. Due to an Eclipse limitation, the Nuclei CPU information may not be displayed in the `Debugger Console`, but it can be viewed in the `gdb trace`.

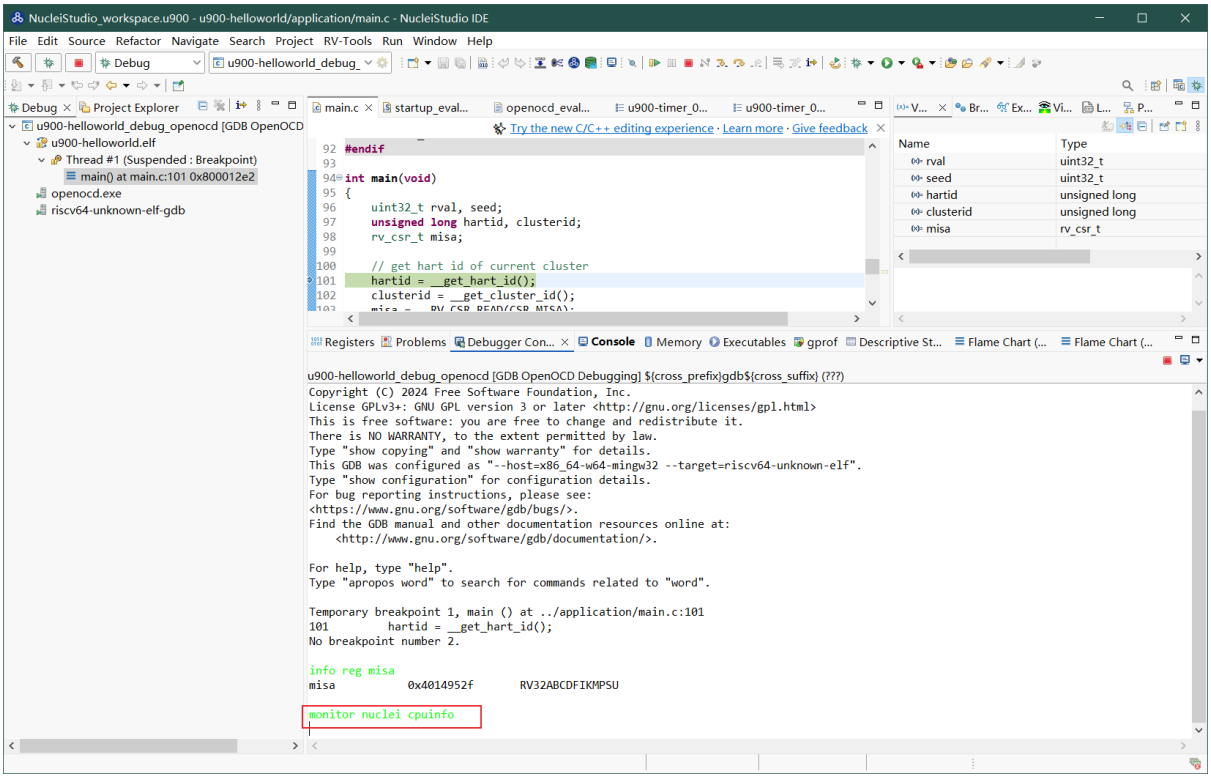
First, enable the `gdb trace` feature. In Nuclei Studio, open the menu `Window -> Preferences`.



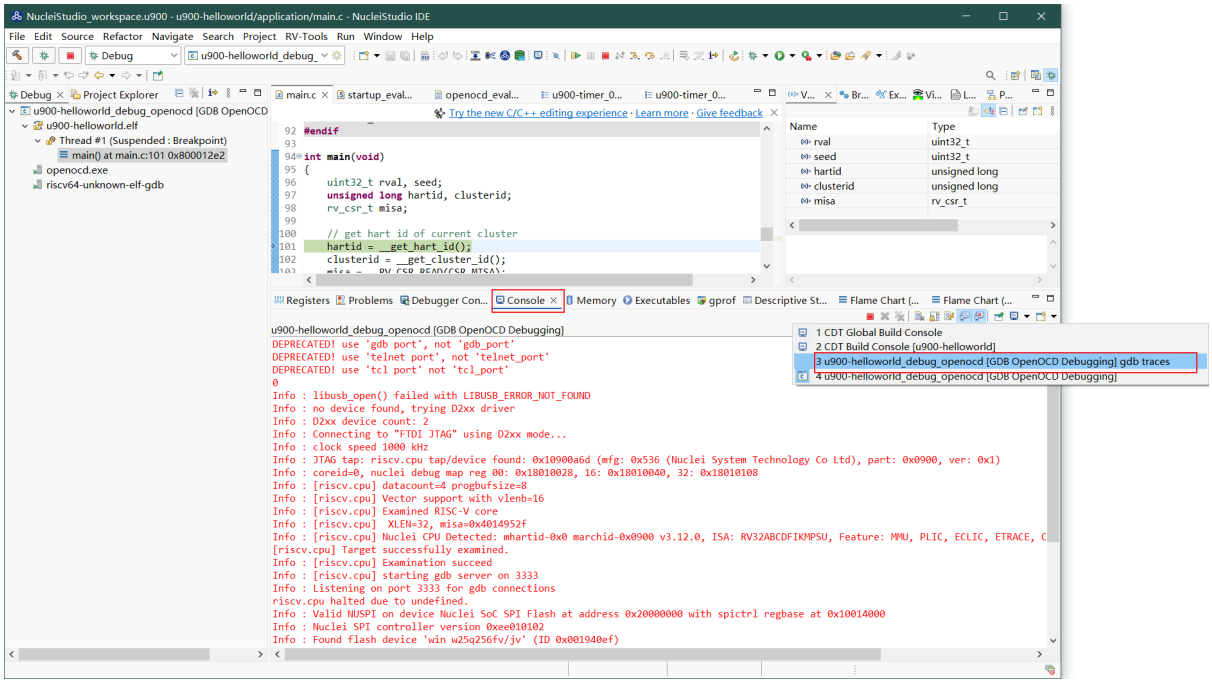
Search for `gdb`, and in the GDB configuration page, find and check `Show the GDB trace console with character limit`.



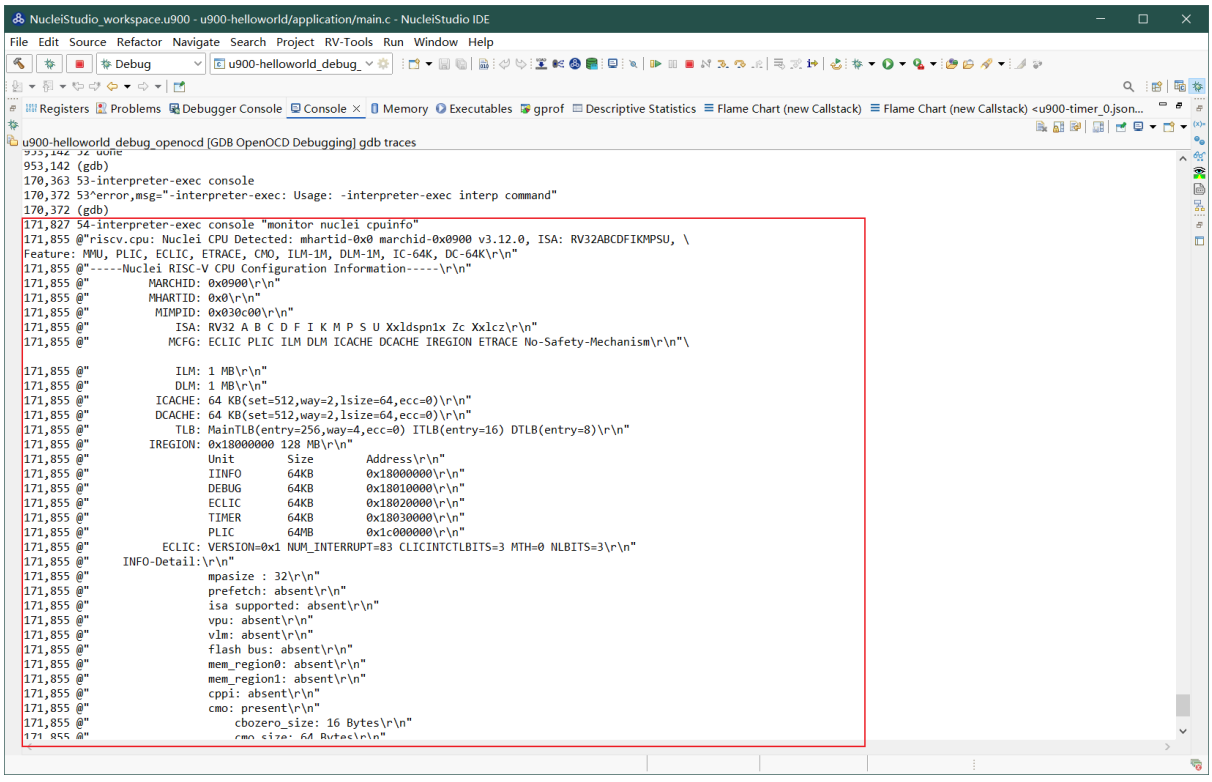
Send the command `monitor nuclei cpuinfo` through the IDE's `Debugger Console` tool.



Then find the `gdb trace` view in the `Console` window and open it.

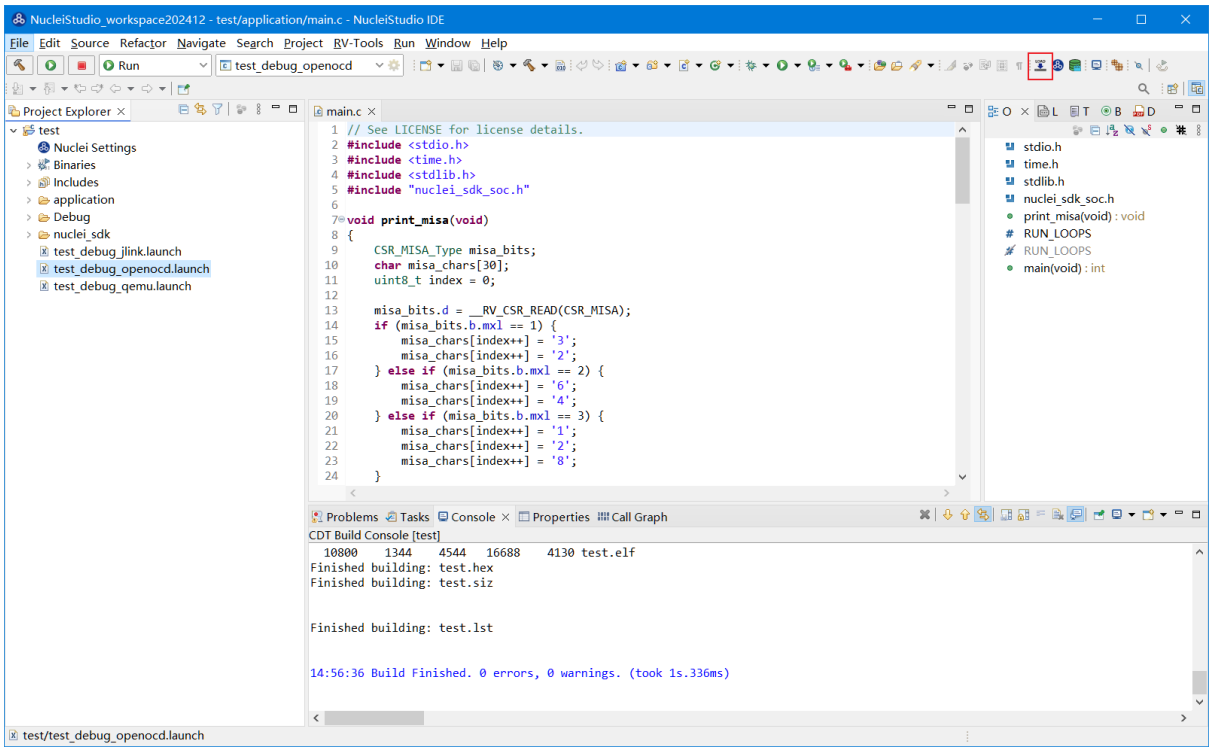


The Nuclei CPU information can be viewed in the `gdb trace` view.

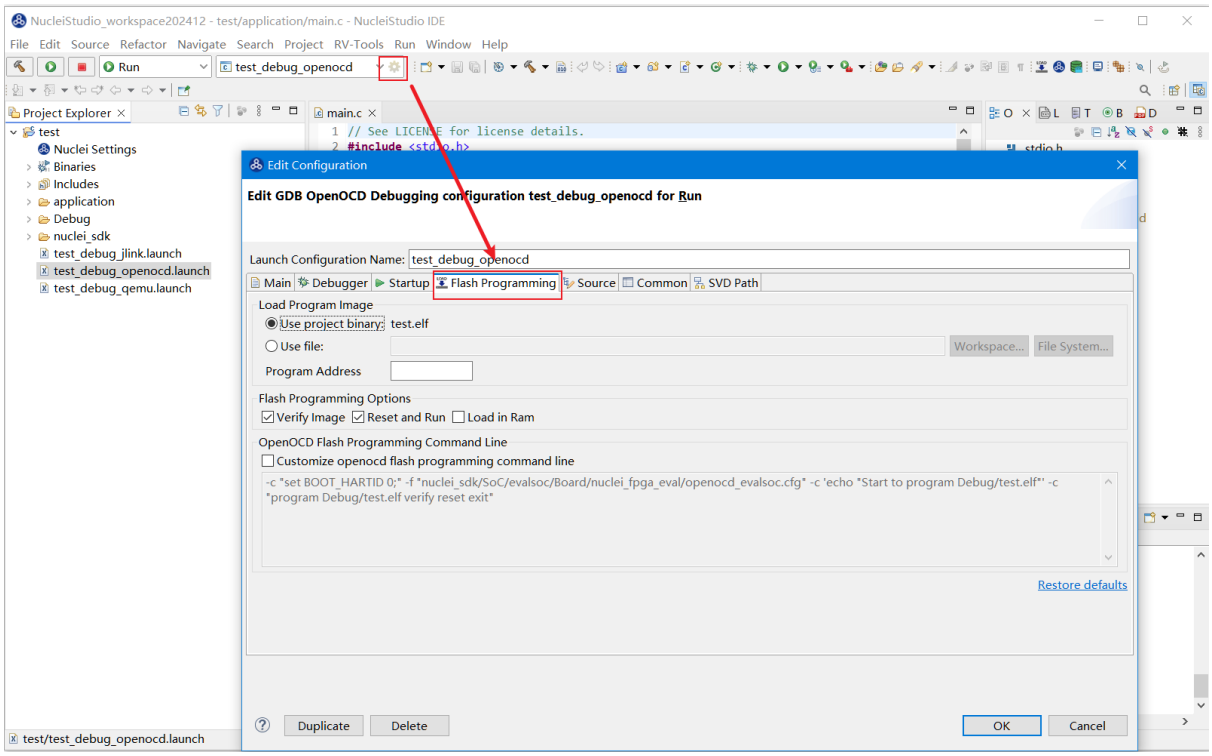


2.9.7 Flash Programming

To support downloading compiled binary files directly to a hardware development board, Nuclei Studio provides the Flash Programming feature. This feature lets you download a compiled binary file to a hardware development board quickly and conveniently, greatly improving development and debugging efficiency; it also simplifies the workflow, so that a single click downloads the binary file. After the project is built, find Flash Programming and click it to download the binary file.

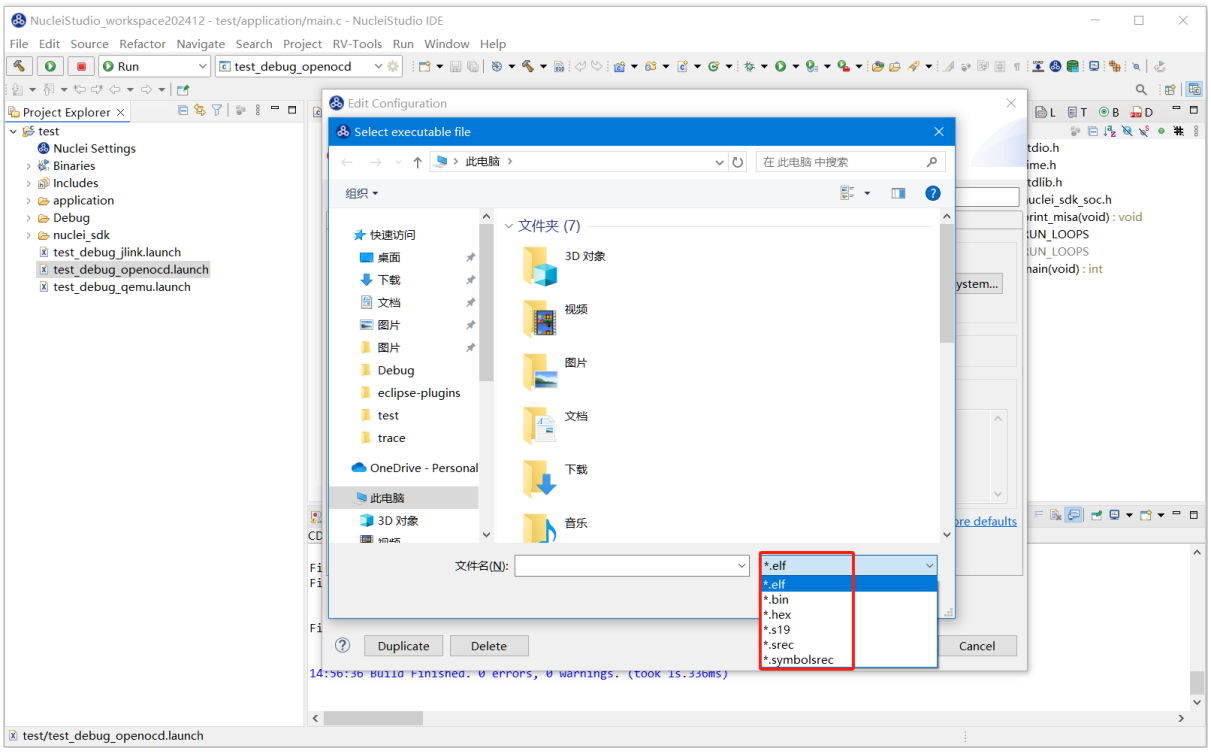


You can also modify its related configuration: click the configuration button in the Launch Bar to open the configuration page, and then select the Flash Programming tab.



Load Program Image

The file to load. ELF format is used by default, and various other formats are supported, such as `*.bin, *.hex, *.s19, *.srec, *.symbolsrec`.



Flash Programming Options

Three Flash Programming options are available:

- Verify Image: After programming completes, verifies that the programmed image matches the file on the currently connected target device, to ensure that programming succeeded.
- Reset and Run: After programming completes, resets the CPU and runs it. Note that if Load in RAM is checked, the CPU only runs without a reset (a reset would erase a program loaded into RAM).
- Load in Ram: When checked, the firmware is downloaded to memory instead of non-volatile storage such as Flash. When selected, the Program Address must be specified.

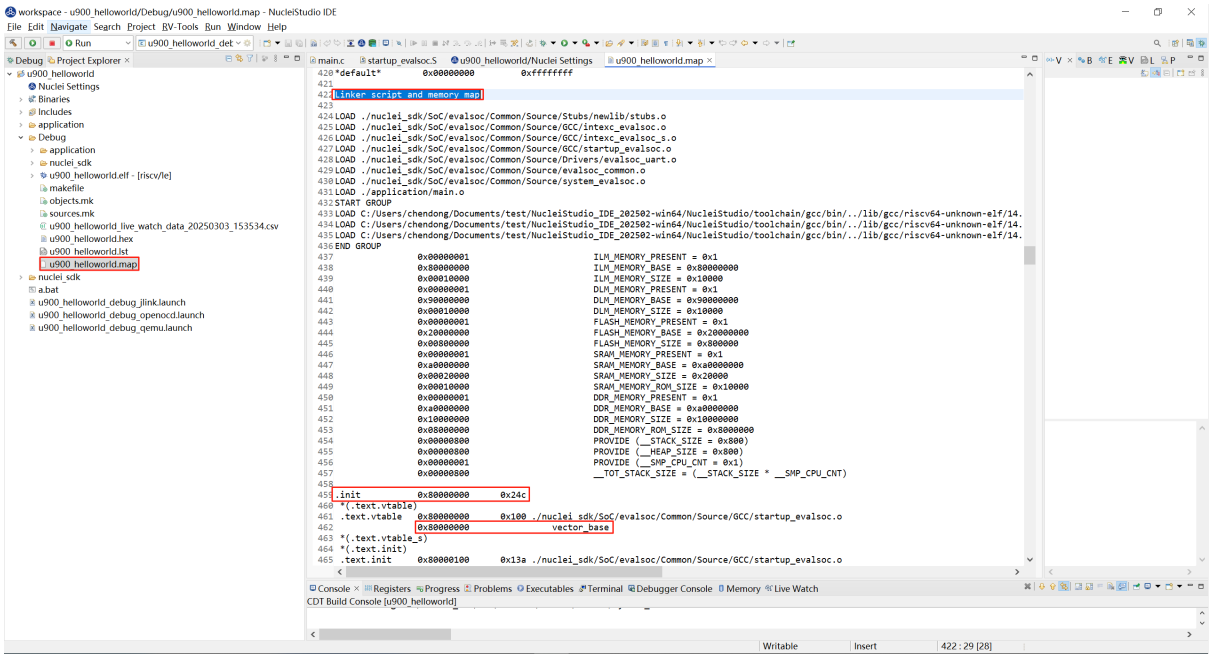
Note: The firmware download method and the OpenOCD commands used differ depending on whether `Load in Ram` is checked. This is described in detail later in this document; read it carefully and choose correctly according to your actual situation.

These parameters correspond to the project ' s Download mode. Nuclei Studio supports `DDR/ILM/SRAM/FLASH/FLASHXIP` by default.

When the project ' s Download mode is `DDR/ILM/SRAM`, `Load in Ram` must be checked, and the `Program Address` must not be empty. The `Program Address` here is the first address where the program is loaded into RAM.

In general, the `Program Address` can be found in the `*.map` file, or by running `riscv64-unknown-elf-readelf -h /path/to/your.elf` and checking the Entry point address. This document only describes how to find the `Program Address` from the `*.map` file.

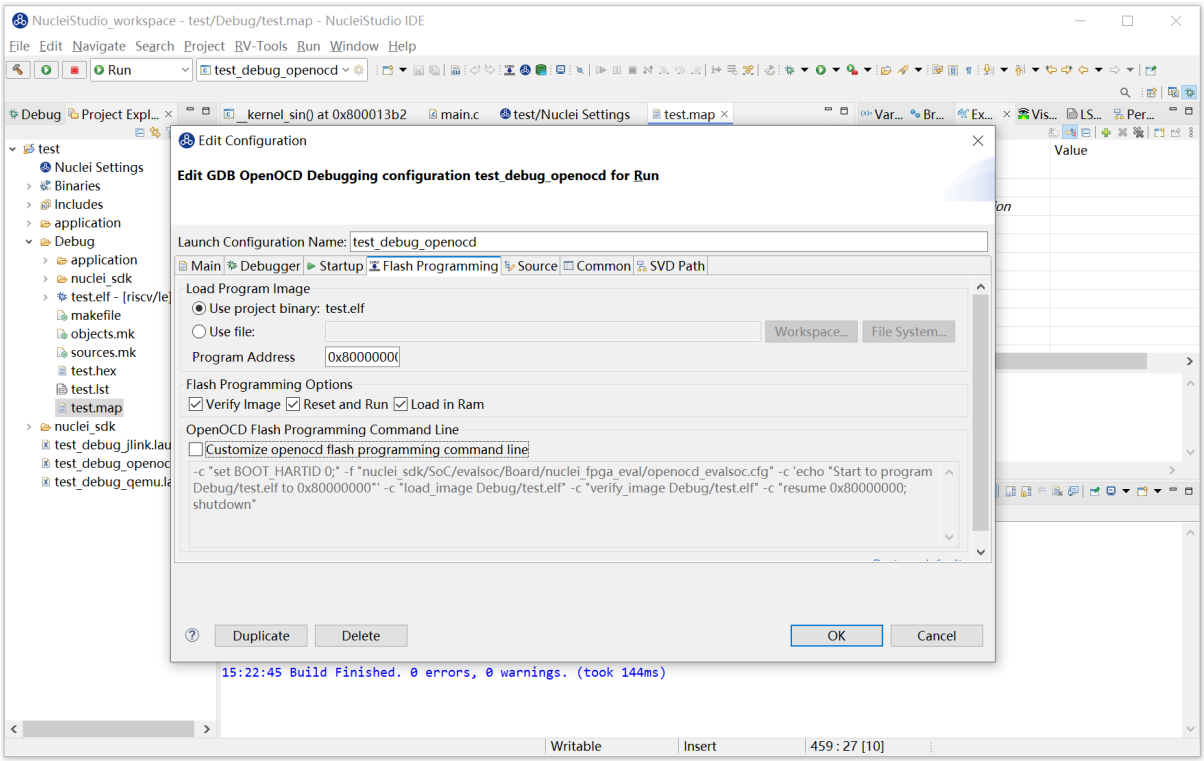
Open the `*.map` file, search for `Linker script and memory map`, and find the address after `.init`; that address is the `Program Address`.



When `Load in Ram` is selected together with `Verify Image`, an extra command `-c "verify_image Debug/test.elf"` appears in the command line; in this case, the image check is implemented through the `verify_image` command.

When `Load in Ram` is selected together with `Reset and Run`, an extra command `-c "resume 0x80000000; shutdown"` appears in the command line; in this case, the system may be forcibly reset after loading through the `resume` command.

```
-c "set BOOT_HARTID 0;"
-f "nuclei_sdk/SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg"
-c 'echo "Start to program Debug/test.elf to 0x80000000"'
-c "load_image Debug/test.elf"
-c "verify_image Debug/test.elf"
-c "resume 0x80000000; shutdown"
```

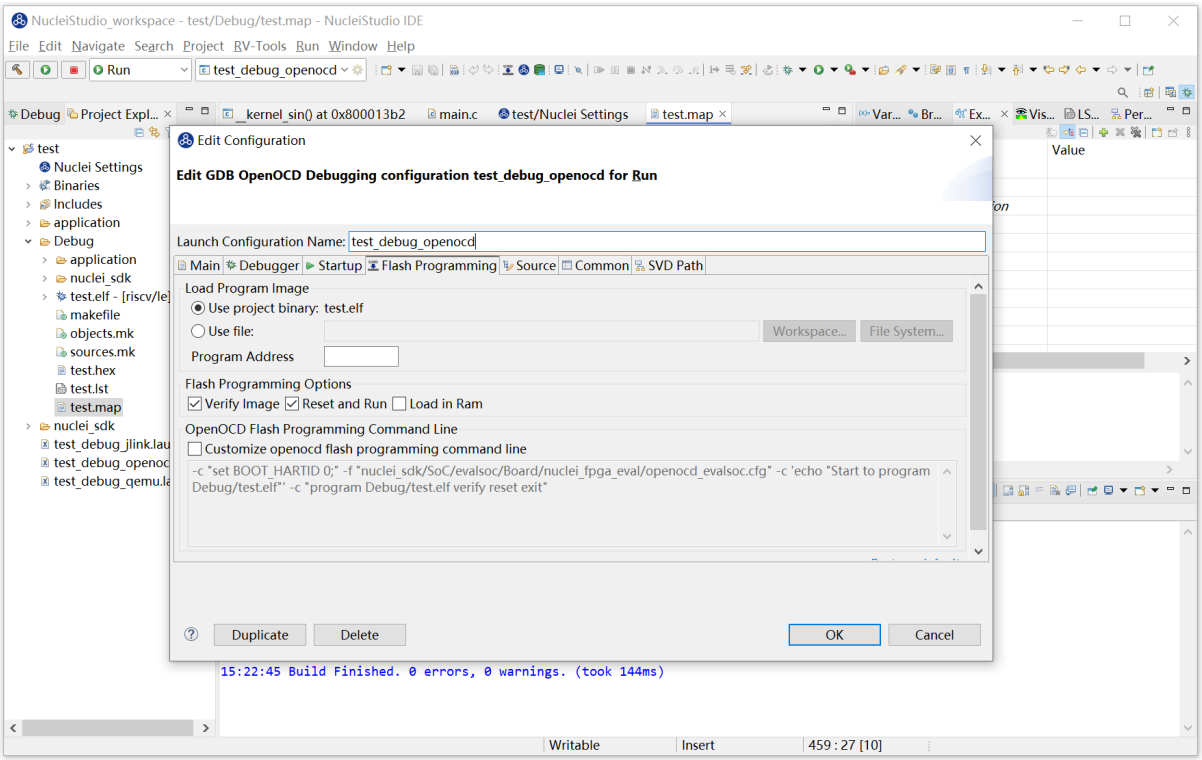


When the project ' s Download mode is `FLASH/FLASHXIP`, `Load in Ram` is not checked, and the `Program Address` must be empty.

When `Verify Image` is selected, an extra command `verify` appears in the command line; in this case, the image check is implemented through the `verify` command.

When `Reset and Run` is selected, an extra command `reset` appears in the command line; in this case, the system may be forcibly reset after loading through the `reset` command.

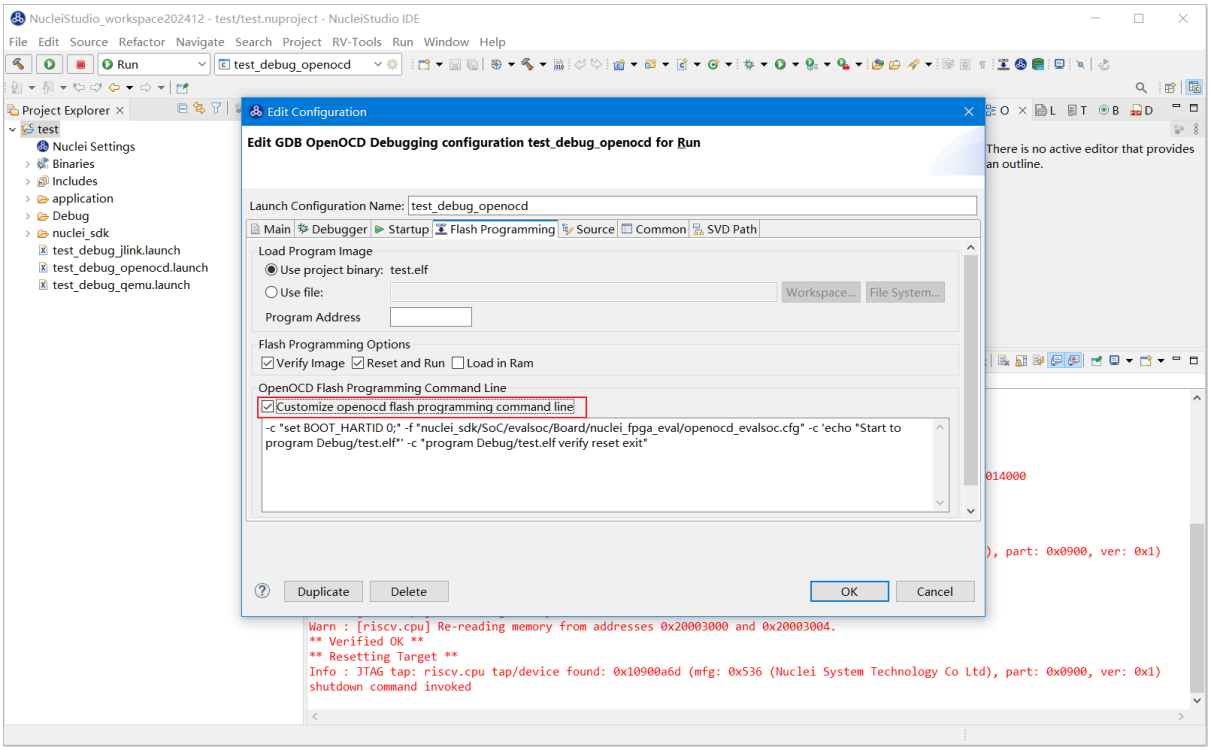
```
-c "set BOOT_HARTID 0;"
-f "nuclei_sdk/SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg"
-c 'echo "Start to program Debug/test.elf"'
-c "program Debug/test.elf verify reset exit"
```



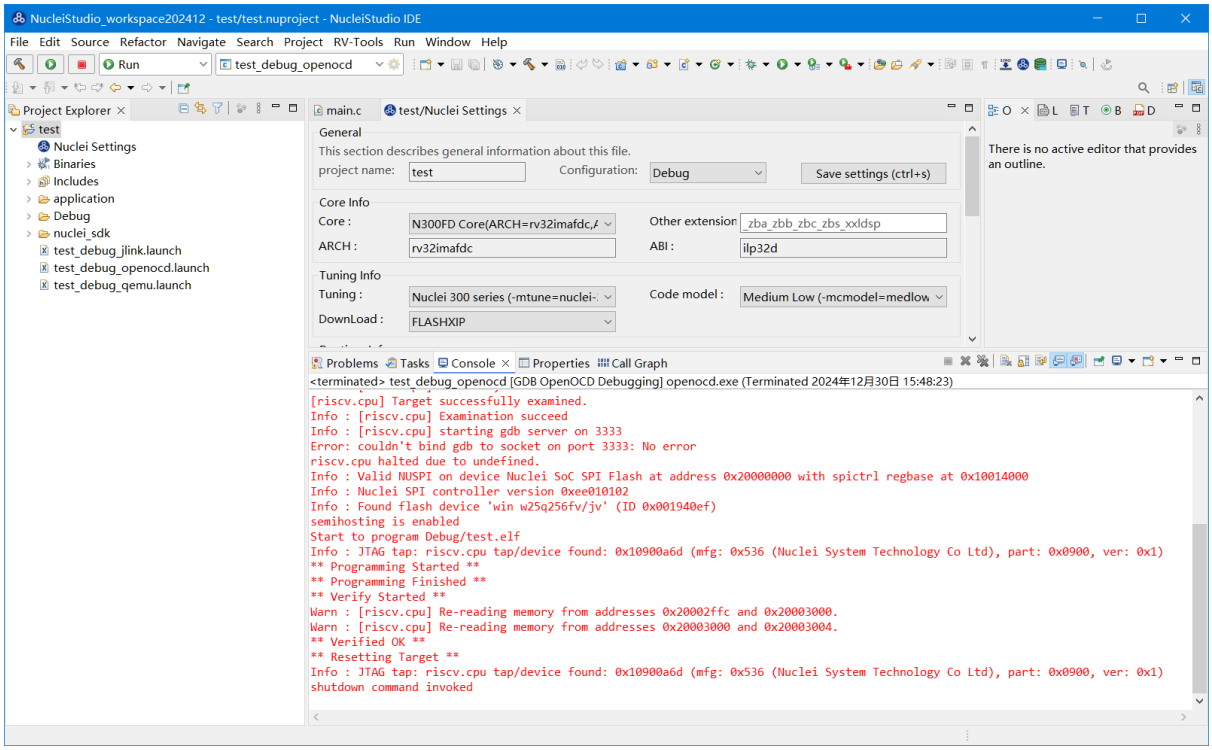
OpenOCD Flash Programming Command line

The Flash Programming parameters are ultimately executed through OpenOCD. By default, the `Customize openocd flash programming command line` option is unchecked. When you check `Customize openocd flash programming command line`, all other related options are disabled, and you can directly enter a custom command in the input box below.

If you are familiar with OpenOCD commands, you can customize the command to meet specific needs. However, if you are not familiar with OpenOCD command-line operations, leave this option unchecked to avoid configuration errors.



After configuring the parameters as needed, click Flash Programming to download the binary code to the hardware. A successful download result is shown below.



2.9.8 Flash Erase

In real development there is often a need to erase Flash independently **without starting a full debug or download session**: for example, to let the chip restart from a blank state to rule out interference from old firmware/residual data, to clear a write-protected region so it can be programmed again, or to erase only a certain sector / address range while keeping the rest. If every erase required first establishing a GDB debug connection and then exiting, it would be cumbersome and could introduce unnecessary side effects, so it is provided as an independent entry point that can be triggered from the toolbar or the right-click menu.

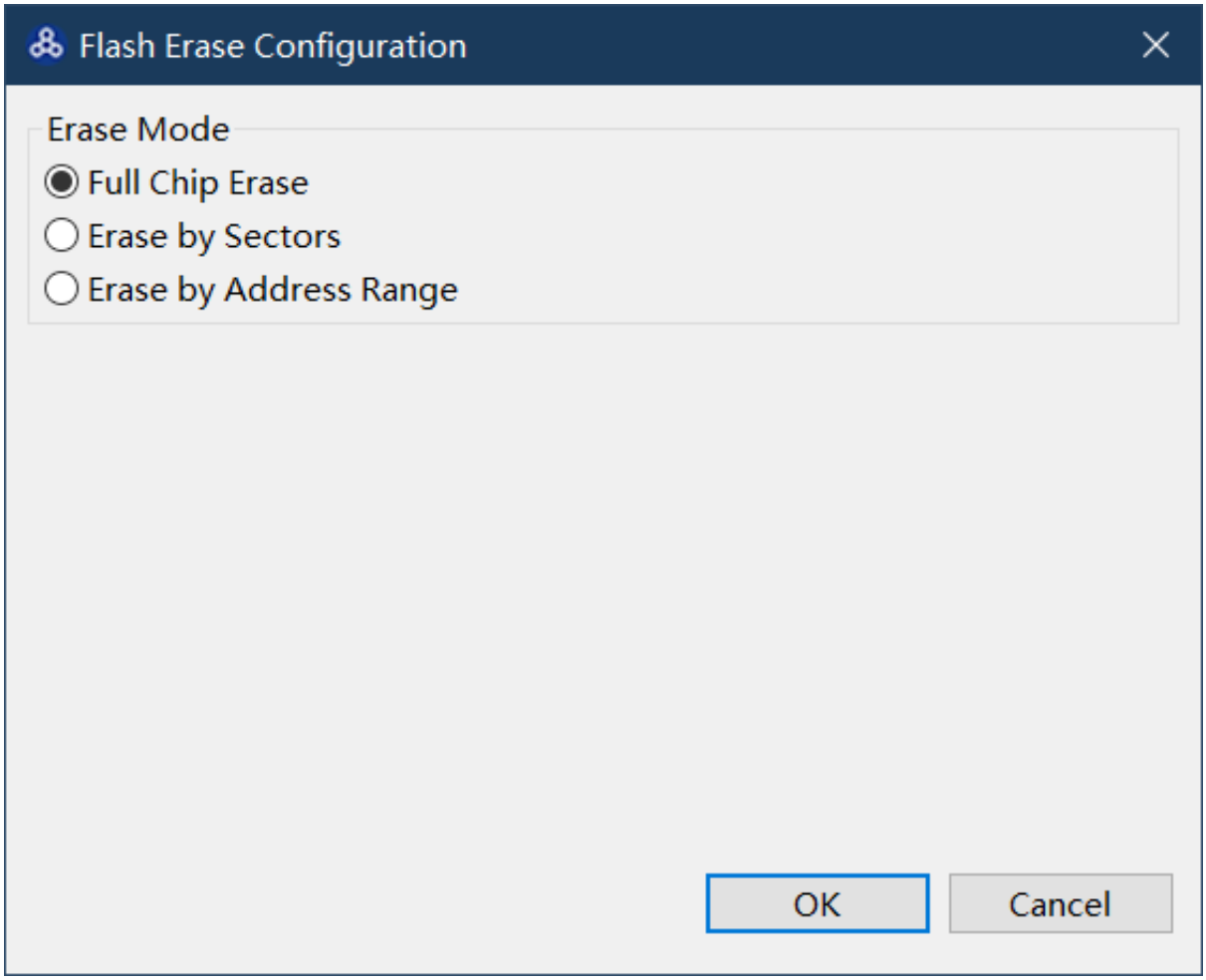
Flash Erase provides an entry point independent of the debug session, driving OpenOCD directly to erase the target chip ' s Flash, and exiting immediately after the erase completes (no GDB start, no program download).

It shares the same set of launch configuration attributes as Erase Before Programming in the debug/download flow, but they are triggered differently: Flash Erase is triggered independently via the toolbar or right-click menu, while Erase Before Programming is executed as a pre-download step when debug/download starts. Three erase modes are supported:

Note: Checking **Erase before programming** on the Flash Programming page is what assembles the erase command into the download flow; the independent Flash Erase is not affected by that switch and executes immediately on click.

- **Full Chip Erase (entire chip erase)**
- **Erase by Sectors (erase by sector range)**
- **Erase by Address Range (erase by address range, optional pad / unlock)**

Full Chip Erase



Parameters

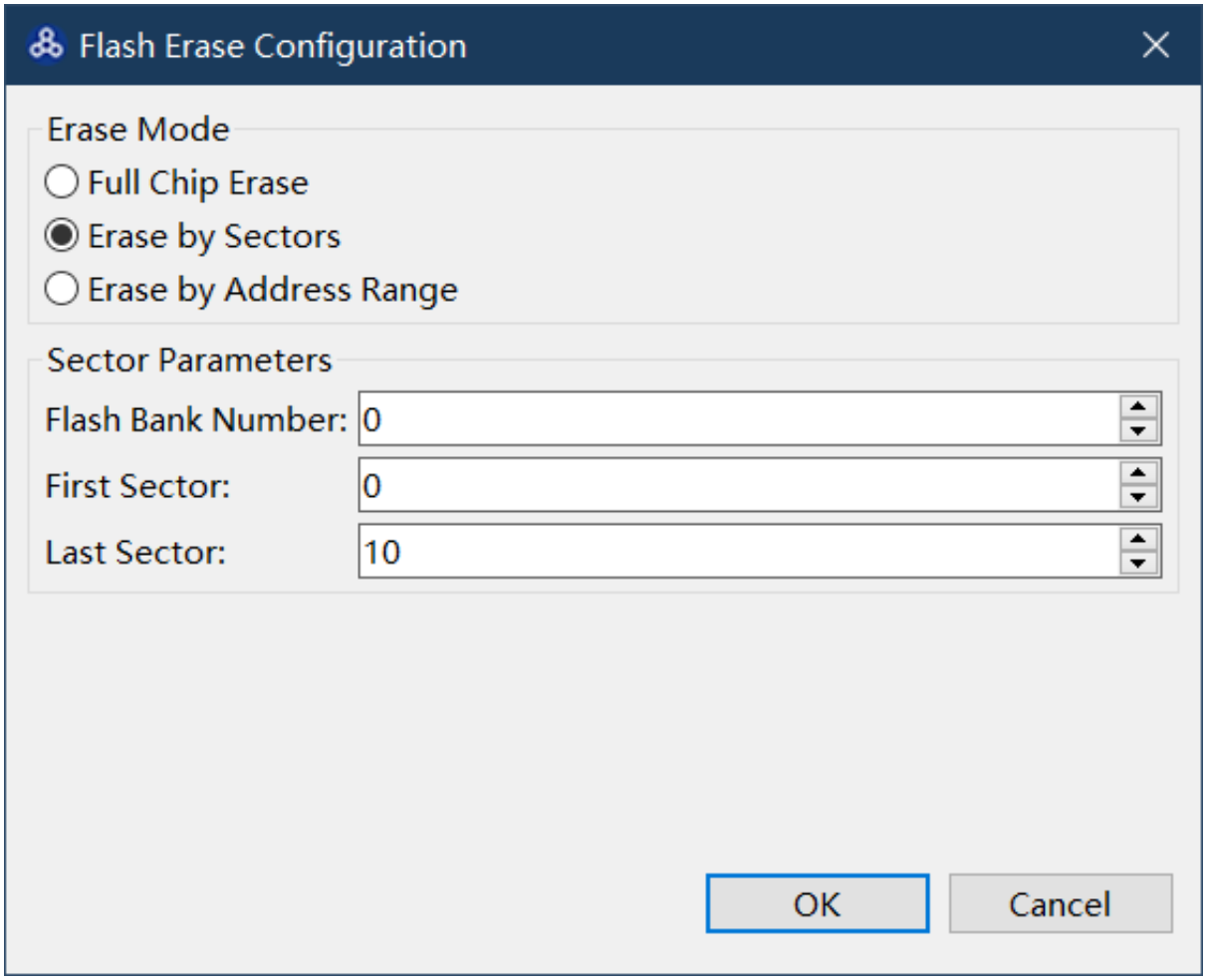
You only need to specify the Flash Bank Number (the Flash bank index, which can be checked with `flash list`; it is usually `0` for Nuclei/RISC-V), and all sectors on that bank will be erased.

Generated OpenOCD command

```
flash erase_sector <bank> 0 last
```

Example: `openocd -c "flash erase_sector 0 0 last" -c exit`

Erase by Sectors



Parameters

- Flash Bank Number (Spinner, range 0 10, default 0)
- First Sector (Spinner, 0 255): starting sector number; `last` is a keyword indicating the last sector of that bank
- Last Sector (Spinner, 0 255): ending sector number

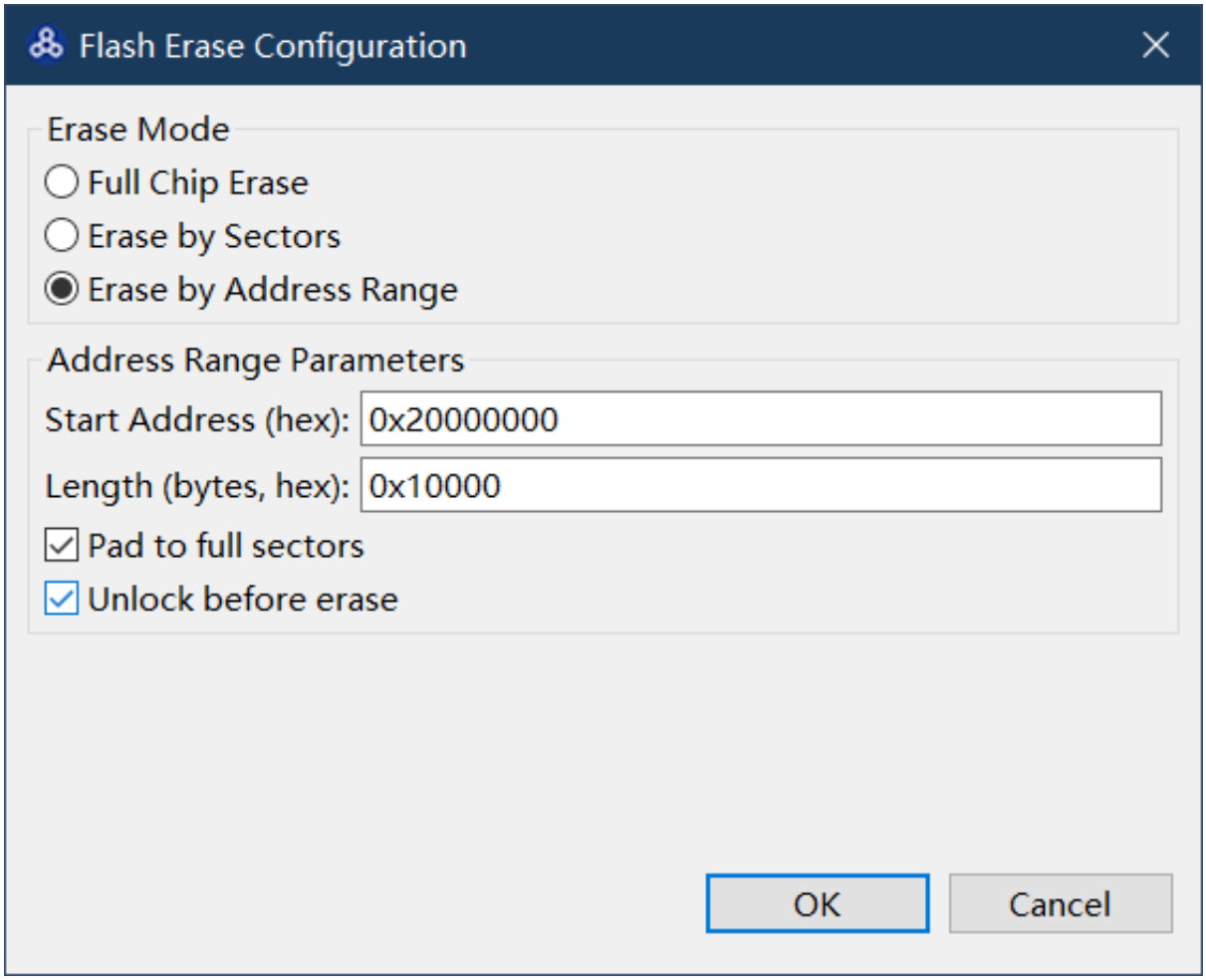
`First Sector ≤ Last Sector` is required; otherwise `Invalid Sector Range` pops up.

Generated OpenOCD command

```
flash erase_sector <bank> <first> <last>
```

Example: `openocd -c "flash erase_sector 0 2 5" -c exit`

Erase by Address Range



Parameters

- Start Address (hex) (default 0x20000000): erase start address; supports `0x`-prefixed hexadecimal, decimal, and also TCL expressions
- Length (bytes, hex) (default 0x10000): erase length in bytes, same format as the address
- Pad to full sectors (checkbox): expand the range to complete sectors (the part less than one sector is erased as well)
- Unlock before erase (checkbox): unlock the Flash before erasing (needed when write protection is present)

The address / length must be a valid numeric value (hexadecimal `0x...` or decimal), otherwise `Invalid Format` pops up.

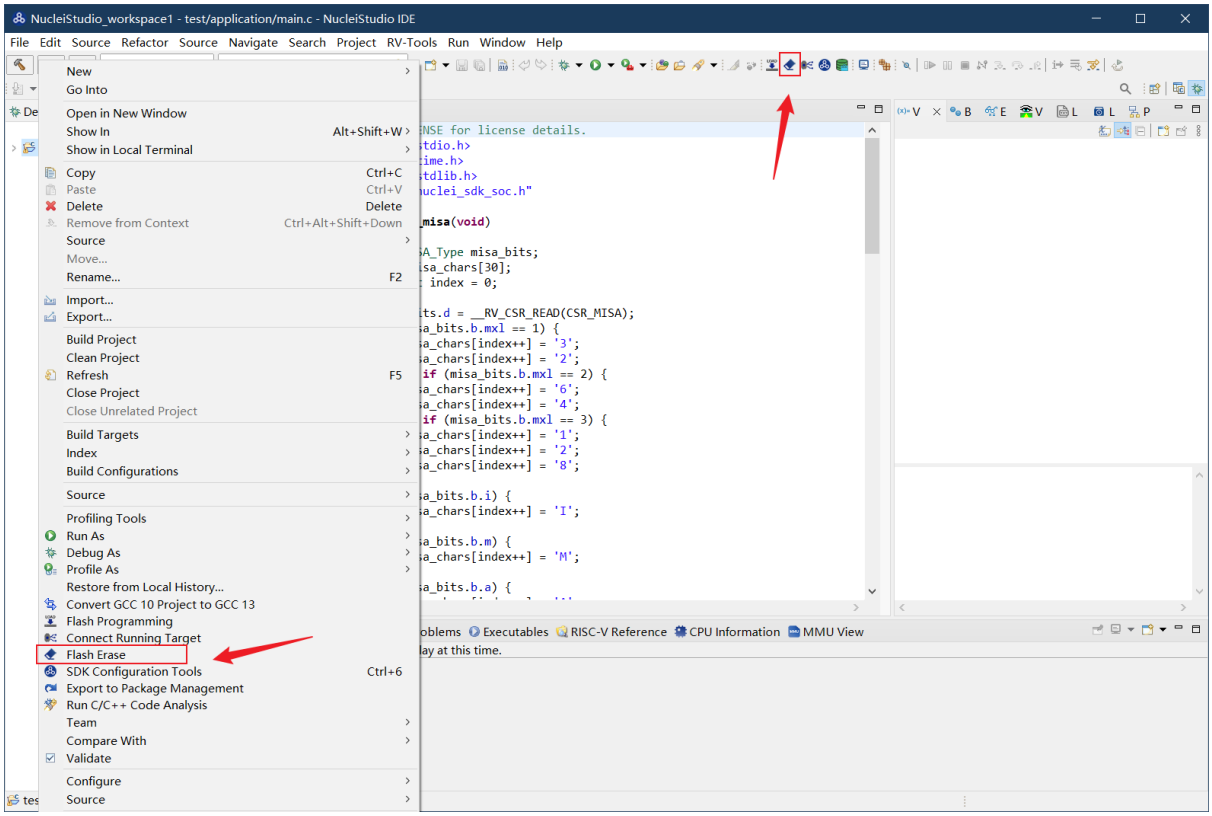
Generated OpenOCD command

```
flash erase_address [pad] [unlock] <addr> <len>
```

Example: `openocd -c "flash erase_address unlock 0x20000000 0x10000" -c exit`

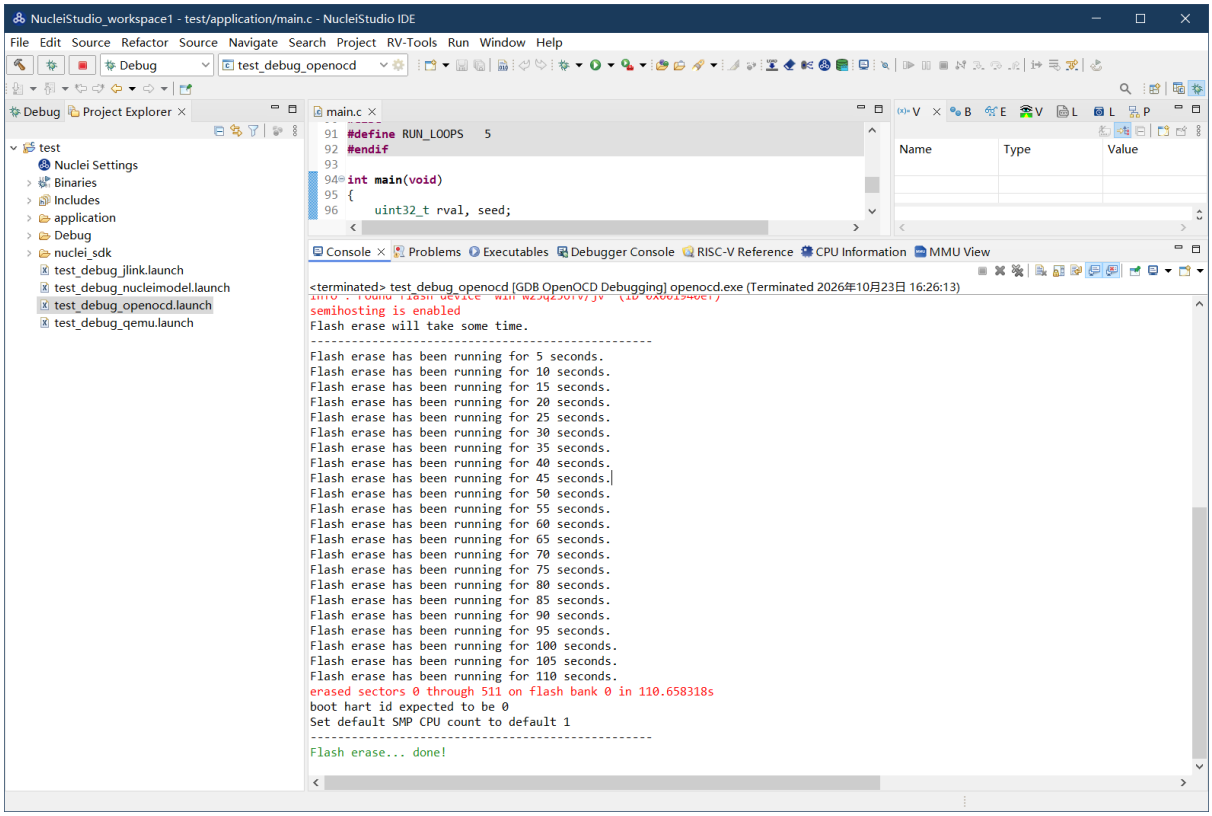
Usage

Use the toolbar button **Flash Erase** or the project ' s **right-click menu → Flash Erase** (shown only for CDT Managed Build projects). After clicking, the **Flash Erase Configuration** dialog pops up. This dialog is the same as the " Flash Erase Configuration " on the Flash Programming page of the debug configuration.



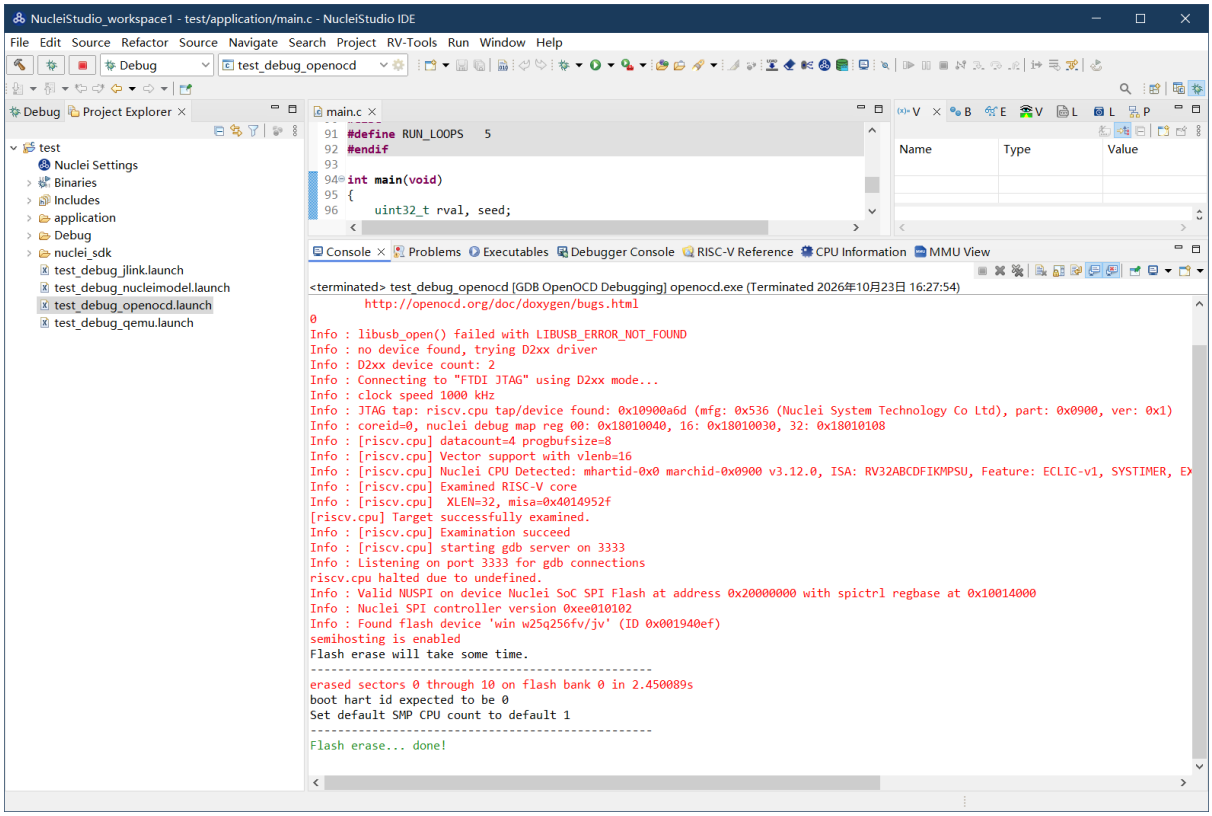
Full chip erase:

1. Select the OpenOCD debug configuration in the Launch Bar and make sure the target is connected;
2. Click the toolbar **Flash Erase** or the project ' s right-click **Flash Erase**;
3. Keep **Full Chip Erase** in the dialog, and fill in the target bank for Bank Number (usually 0);
4. Click **OK**; when `flash erase_sector 0 0 last` appears in the Console, the erase starts, and OpenOCD exits automatically after it completes.



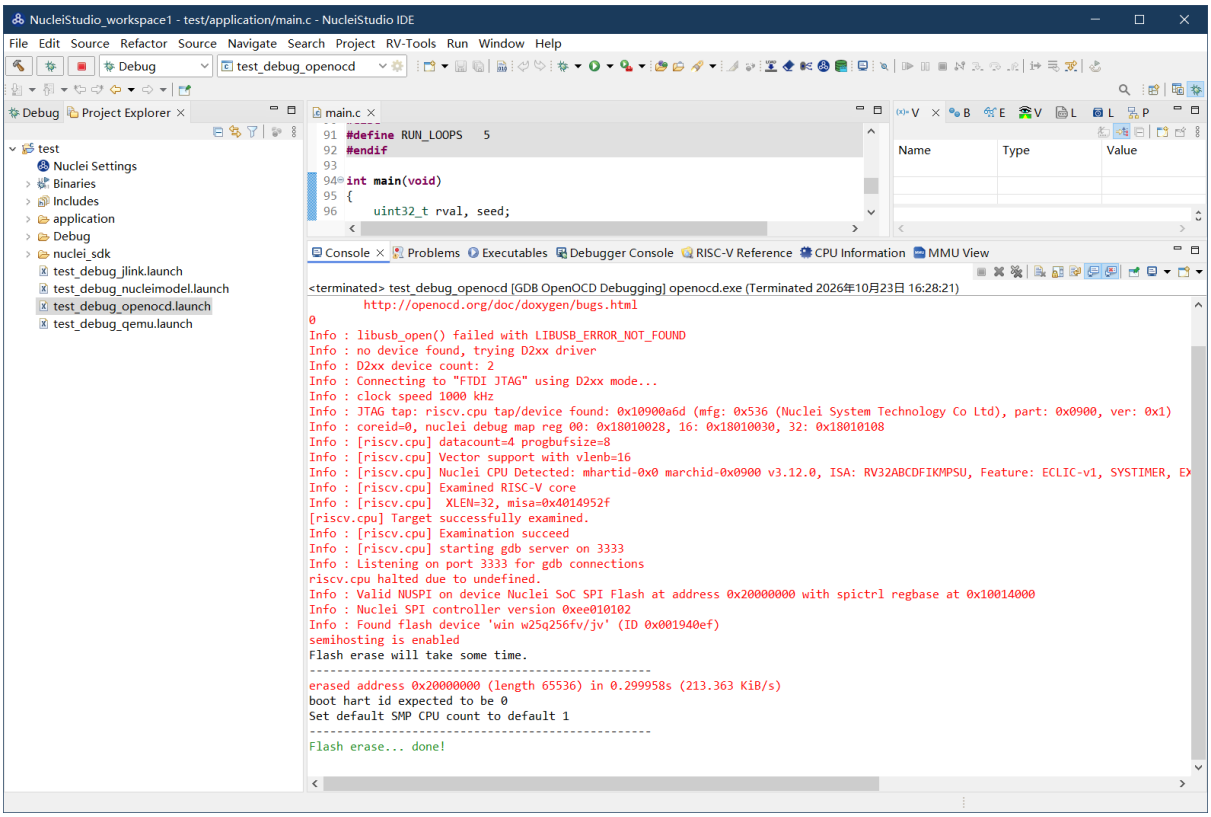
Erase by sector:

1. Open the dialog as above and select **Erase by Sectors**;
2. Fill in Bank / First Sector / Last Sector (ensure First ≤ Last);
3. Click OK to execute.



Erase by address range:

1. Select **Erase by Address Range**;
2. Fill in Start Address and Length (e.g. `0x20000000` / `0x10000`);
3. If the Flash is write-protected, check **Unlock before erase**; if you want to erase the trailing part that is less than one sector as well, check **Pad to full sectors**;
4. Click OK to execute.



Flash Erase starts OpenOCD and executes the above commands through its TCL interpreter; after execution, OpenOCD receives `-c exit` and exits automatically. The three modes correspond respectively to OpenOCD's native full-chip erase, sector erase, and address erase capabilities, and all parameters come from the currently selected launch configuration.

Notes

- This entry point is independent of the debug session; it does not download or start the program, and only exits after completing the Flash erase;
- If there is no active OpenOCD configuration in the Launch Bar, the toolbar/right-click menu prompts you to switch the configuration first;
- Erasing is a destructive operation; before executing, confirm that the address range matches the actual Flash layout of the target;
- Independent erase cannot show variables / registers; the Console shows OpenOCD TCL output;
- The Bank / Sector upper limits are currently hardcoded to 10 / 255; if the number of target banks or sectors exceeds this range, the code needs to be changed;
- Failure detection relies on OpenOCD (**English**) output; if OpenOCD output is localized, detection may fail;
- Erasing is irreversible; confirm that the address / sector range is correct before executing.

2.10 Nuclei Studio Advanced Features

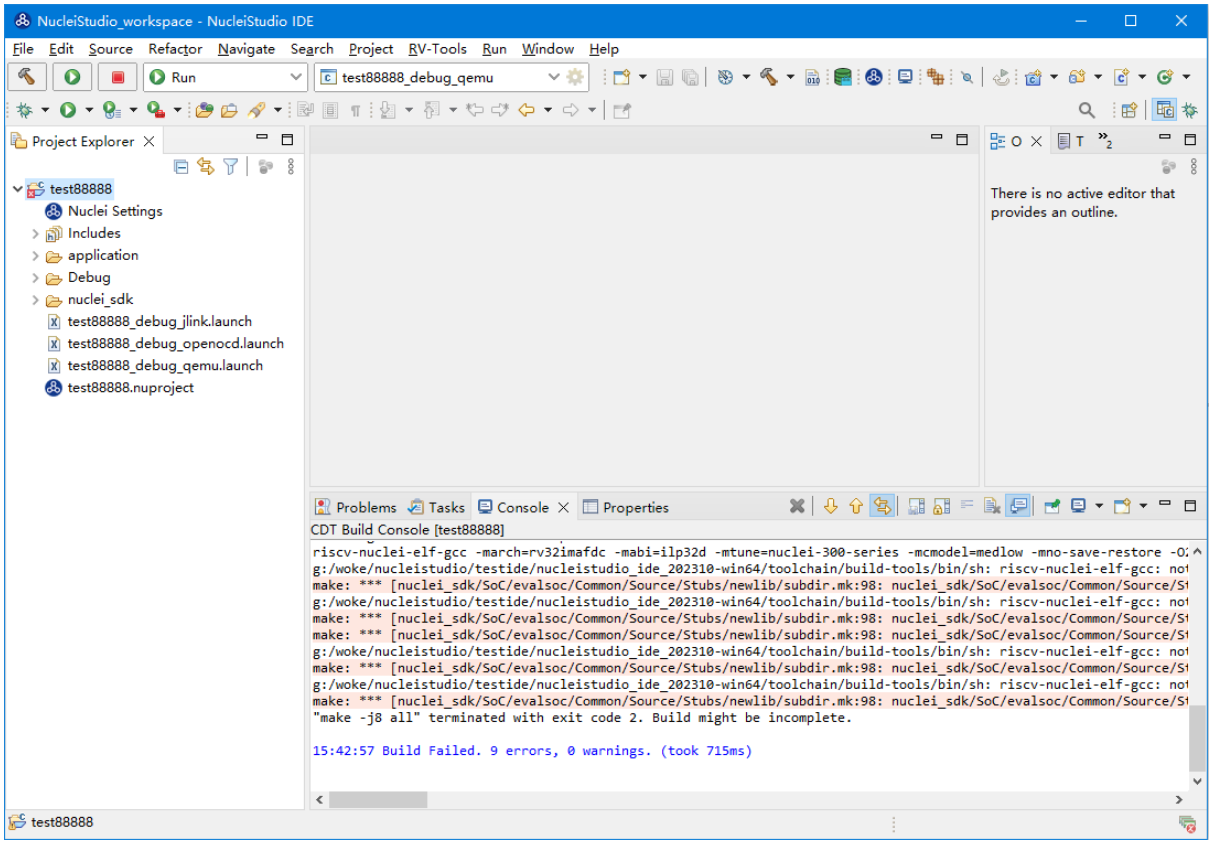
At https://doc.nucleisys.com/nuclei_studio_supply we maintain an extensive and continuously growing collection of usage questions and case studies for Nuclei Studio and Nuclei Tools to help you use them more effectively. Feel free to take a look.

2.10.1 Importing Projects Created by Older Versions

Importing Old Projects Into Nuclei Studio 2023.10

In Nuclei Studio 2023.10, the toolchain, SDK, and other components have changed significantly. To use a project created with an older version of Nuclei Studio, or an older SDK, in the new Nuclei Studio, follow the instructions in this section.

When you import a project created with an older version of Nuclei Studio into Nuclei Studio 2023.10 (for the detailed import procedure, see *Importing Old Projects in Nuclei Studio 2022.12 and Later* (page 218)), or use a project created with an older SDK (that is, any SDK released before the Nuclei Studio 2023.10 release), the project is configured to use GCC 10. If the corresponding toolchain cannot be found, build errors and other problems occur, and the project does not work normally.

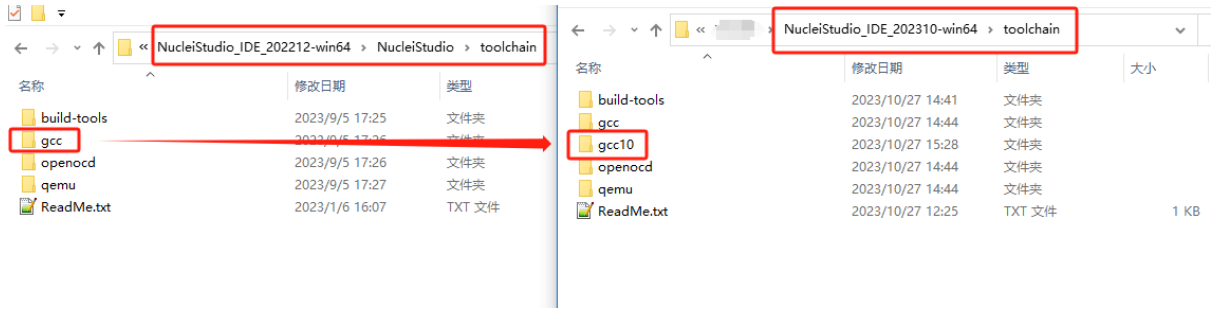


There are two solutions: manually import the GCC 10 toolchain, or convert the project with the conversion tool shipped with Nuclei Studio 2023.10. The second solution is recommended because it converts the project more thoroughly to a format supported by Nuclei Studio 2023.10 and lets you use its latest features.

Manually Importing the GCC 10 Toolchain

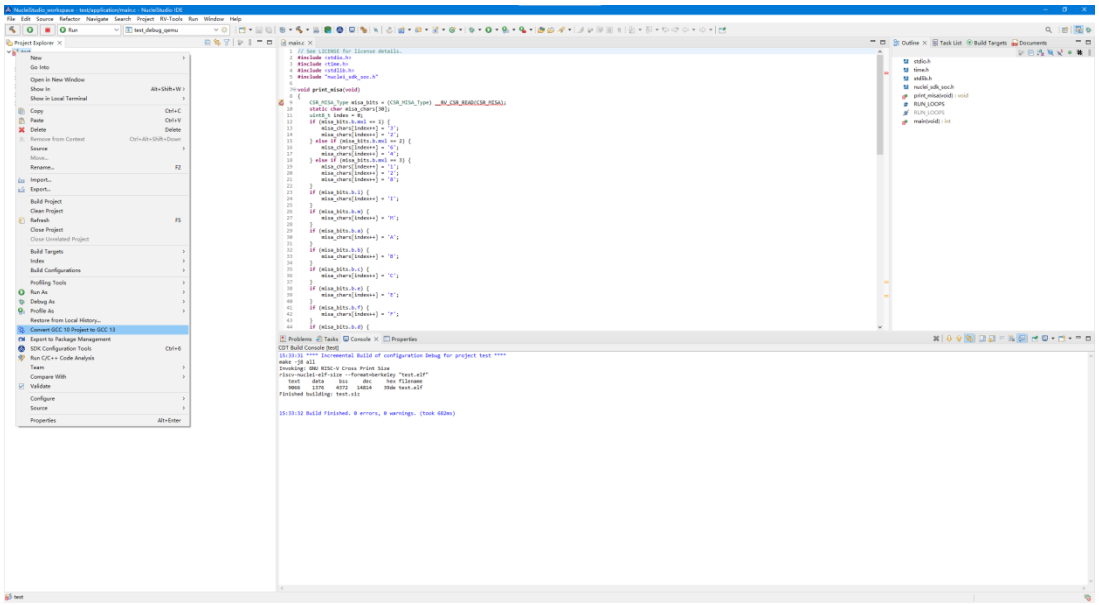
For convenience, Nuclei Studio 2023.10 retains the GCC 10 configuration, but GCC 10 is not packaged with the IDE. To continue developing with GCC 10, import GCC manually.

Locate GCC 10 in the `toolchain\gcc\` directory of the old Nuclei Studio installation and copy its contents into the `toolchain\gcc10\` directory under the Nuclei Studio 2023.10 installation directory, then rebuild the project. The project should now build normally. However, Nuclei QEMU does not work with this method; if your project requires Nuclei QEMU, manually modify the corresponding QEMU debug configuration.

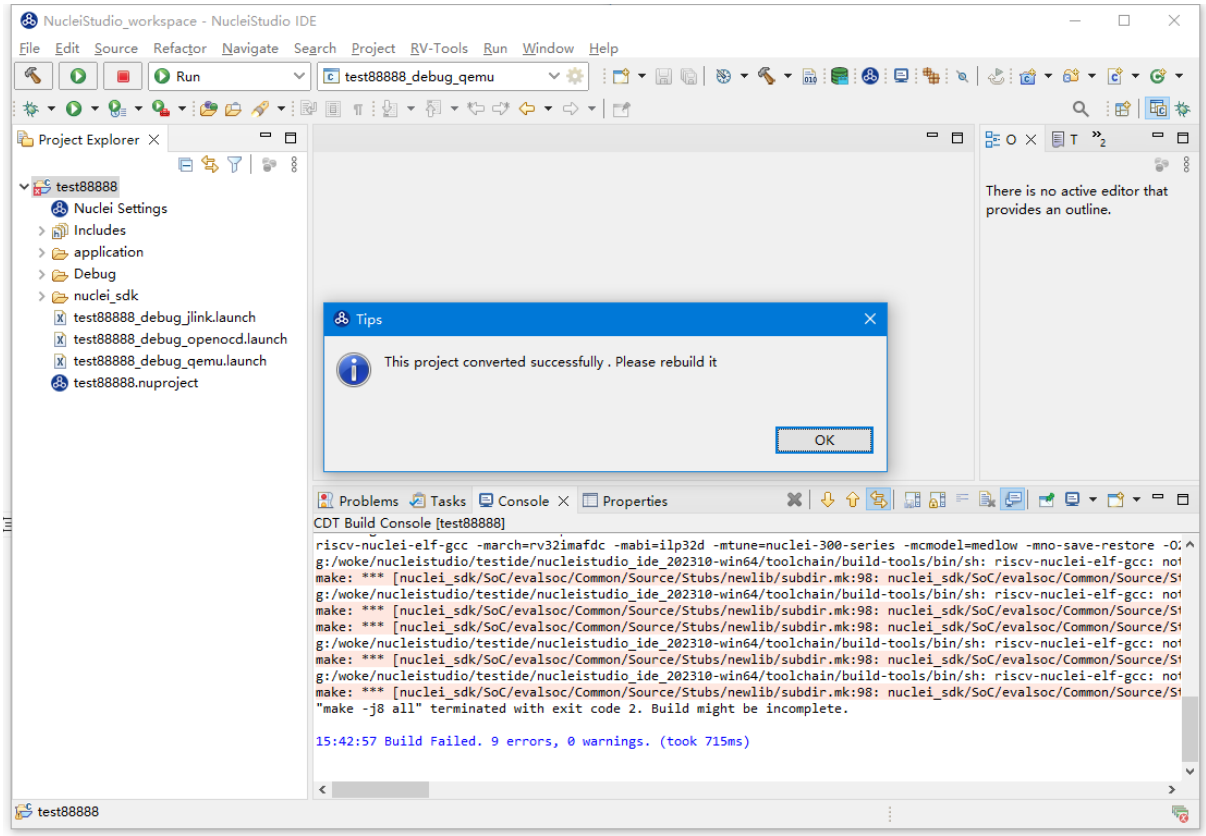


Converting a Project to Support GCC 13 with the Tool

Nuclei Studio 2023.10 provides a quick conversion tool, `Convert GCC 10 Project to GCC 13`, that makes it easier to import old projects. Right-click the project and select `Convert GCC 10 Project to GCC 13` from the context menu.

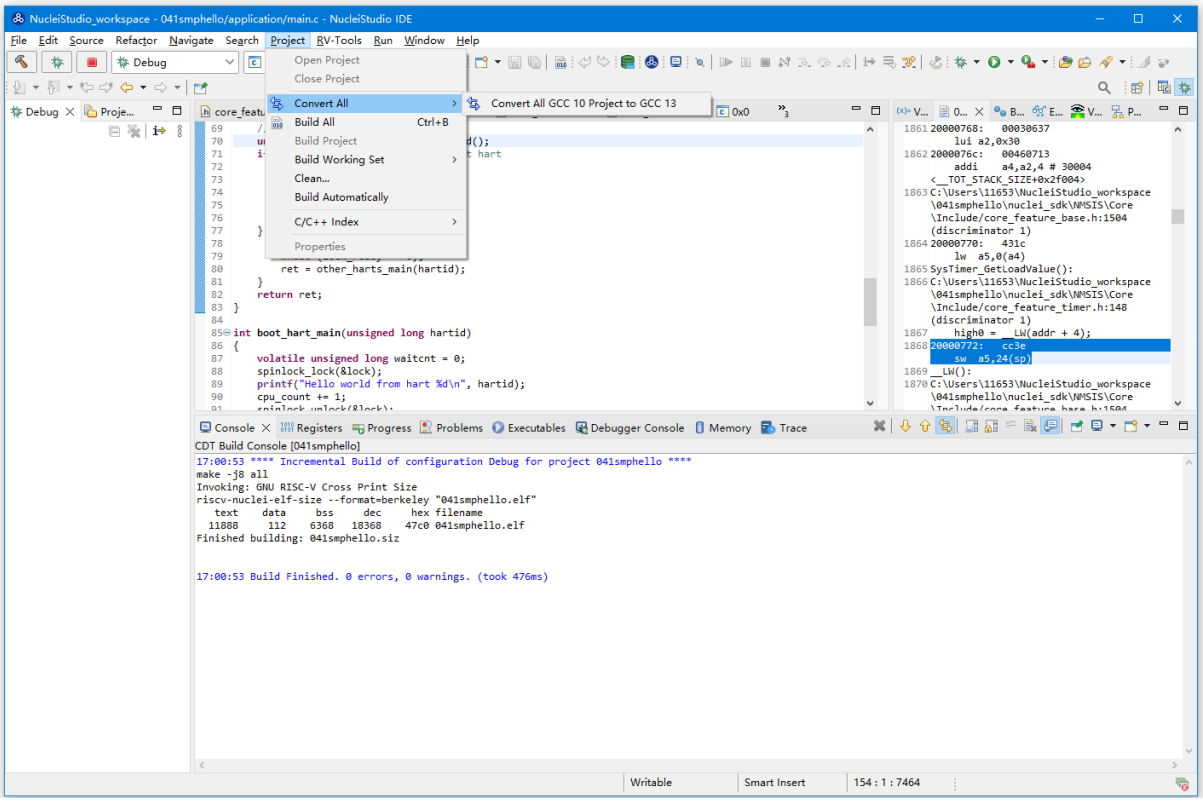


After the project is converted, rebuild it. Nuclei Studio then builds the project with GCC 13, and features such as QEMU work normally.

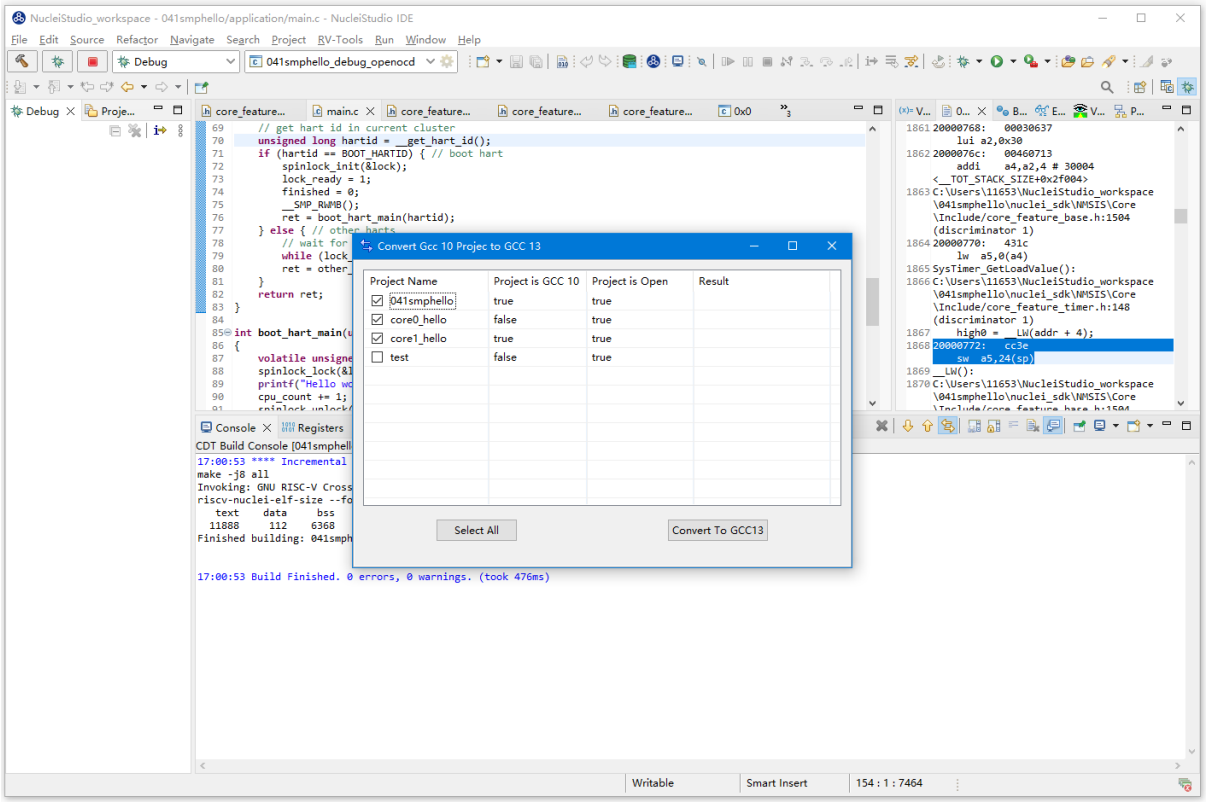


Batch Converting Projects to Support GCC 13

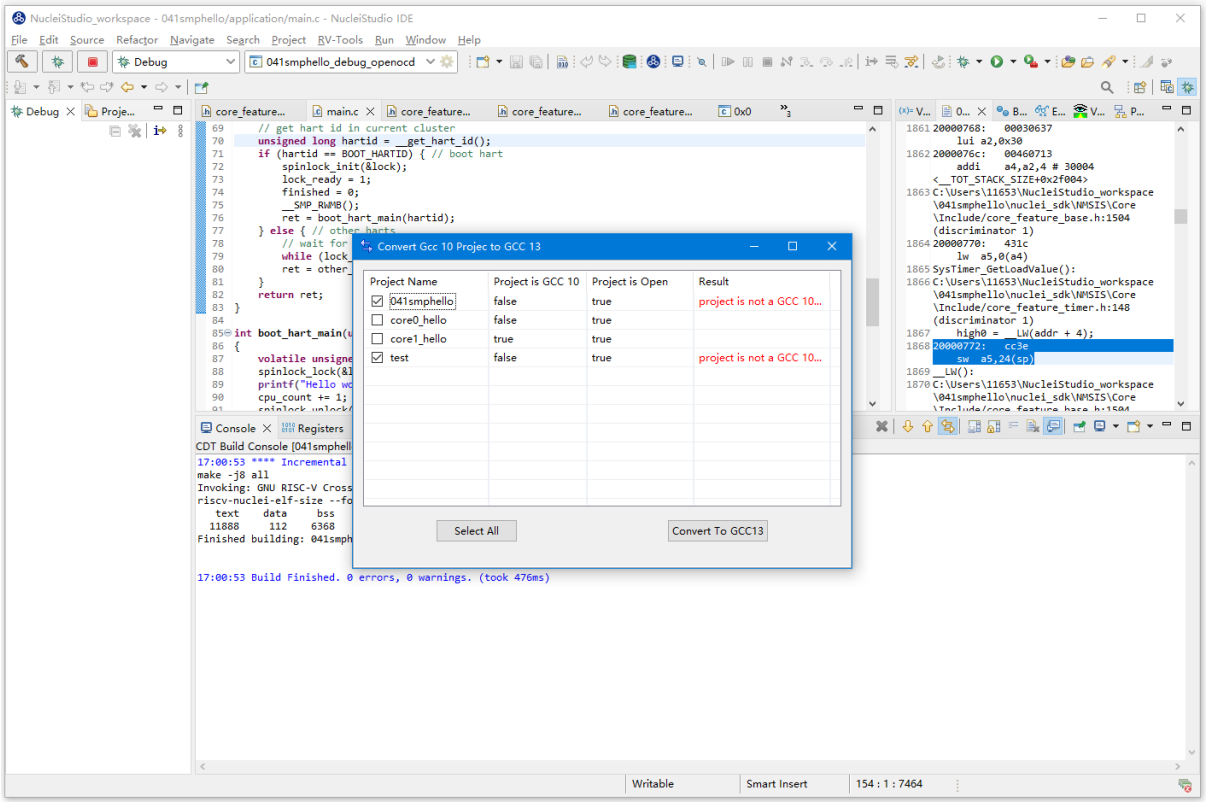
You can also batch convert projects to support GCC 13 in Nuclei Studio 2023.10 and later versions.



Open the conversion tool, select the projects you want to convert, and start the conversion.



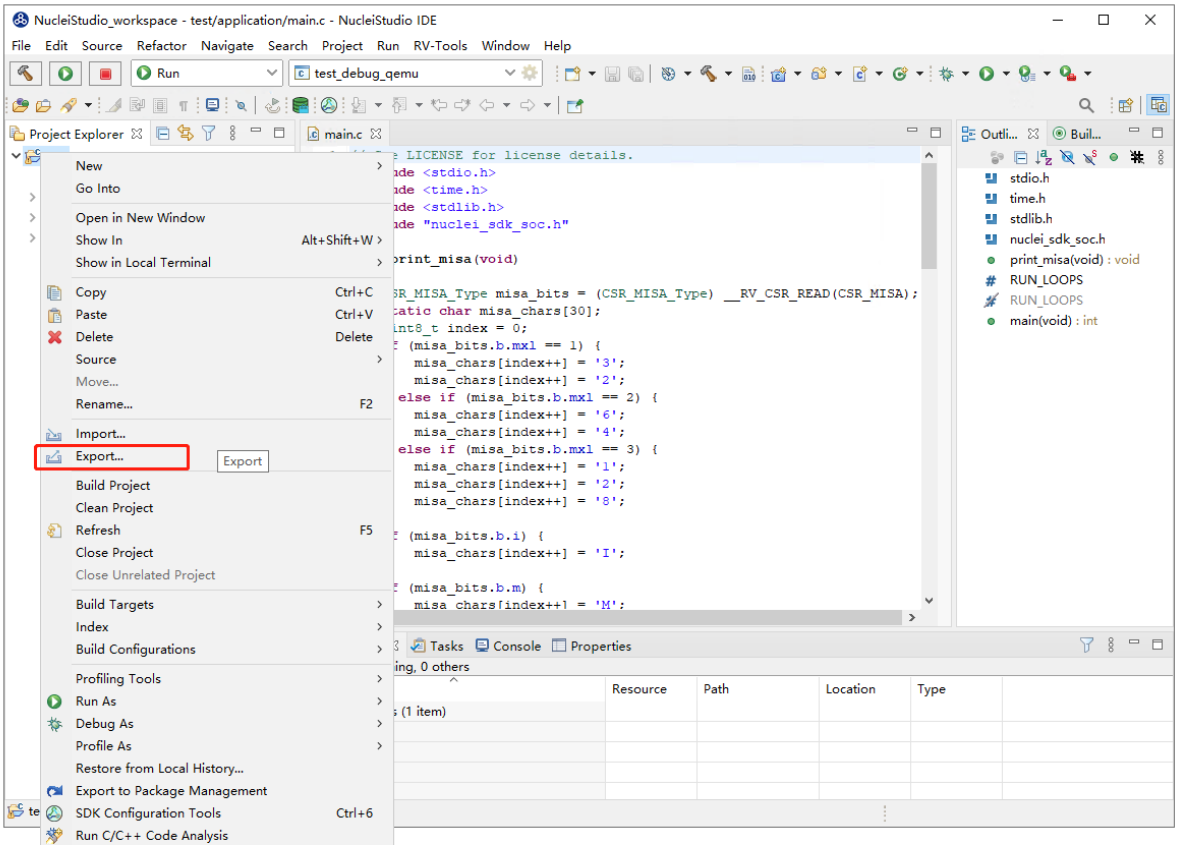
When the conversion is complete, the page displays the conversion results for each project.



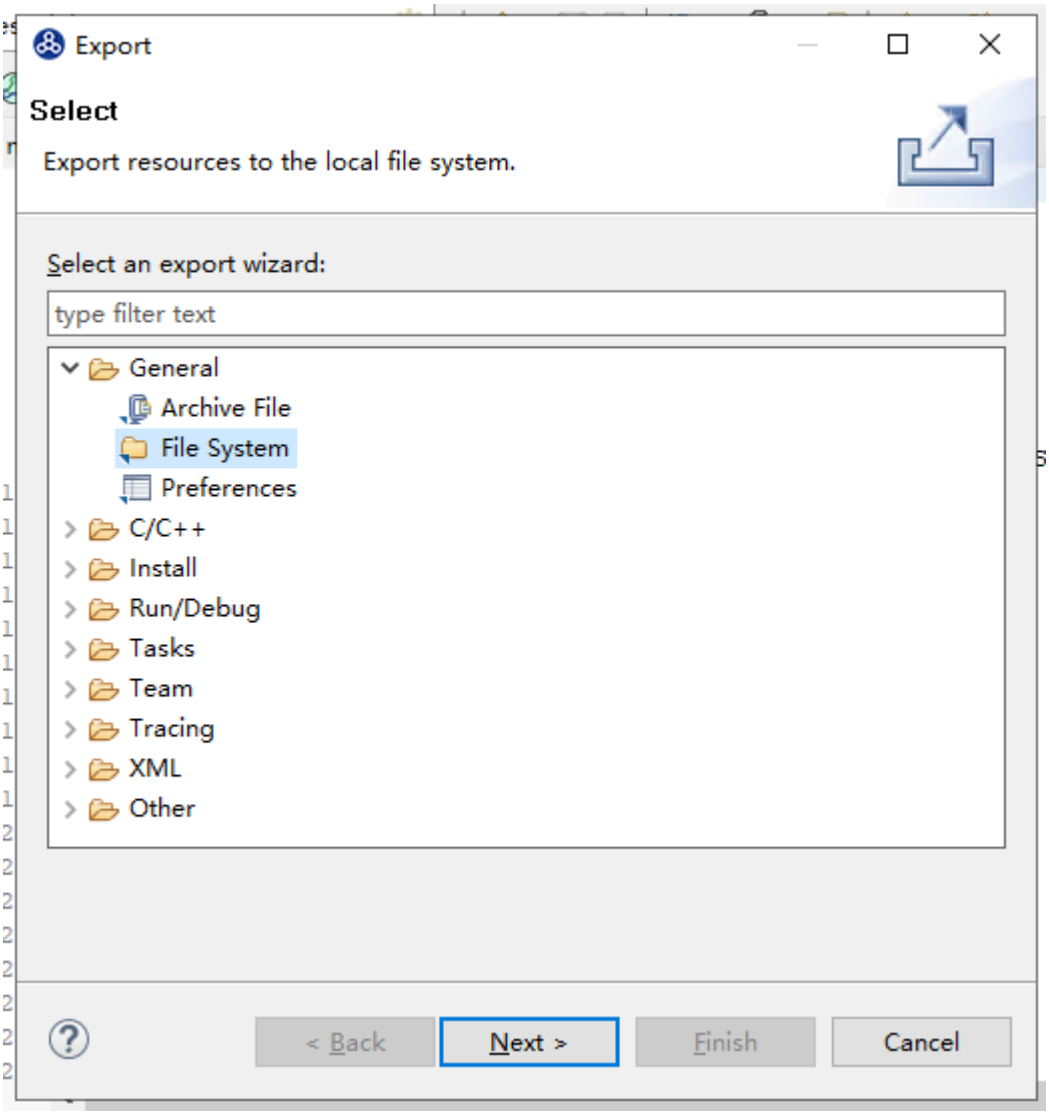
Importing Old Projects in Nuclei Studio 2022.12 and Later

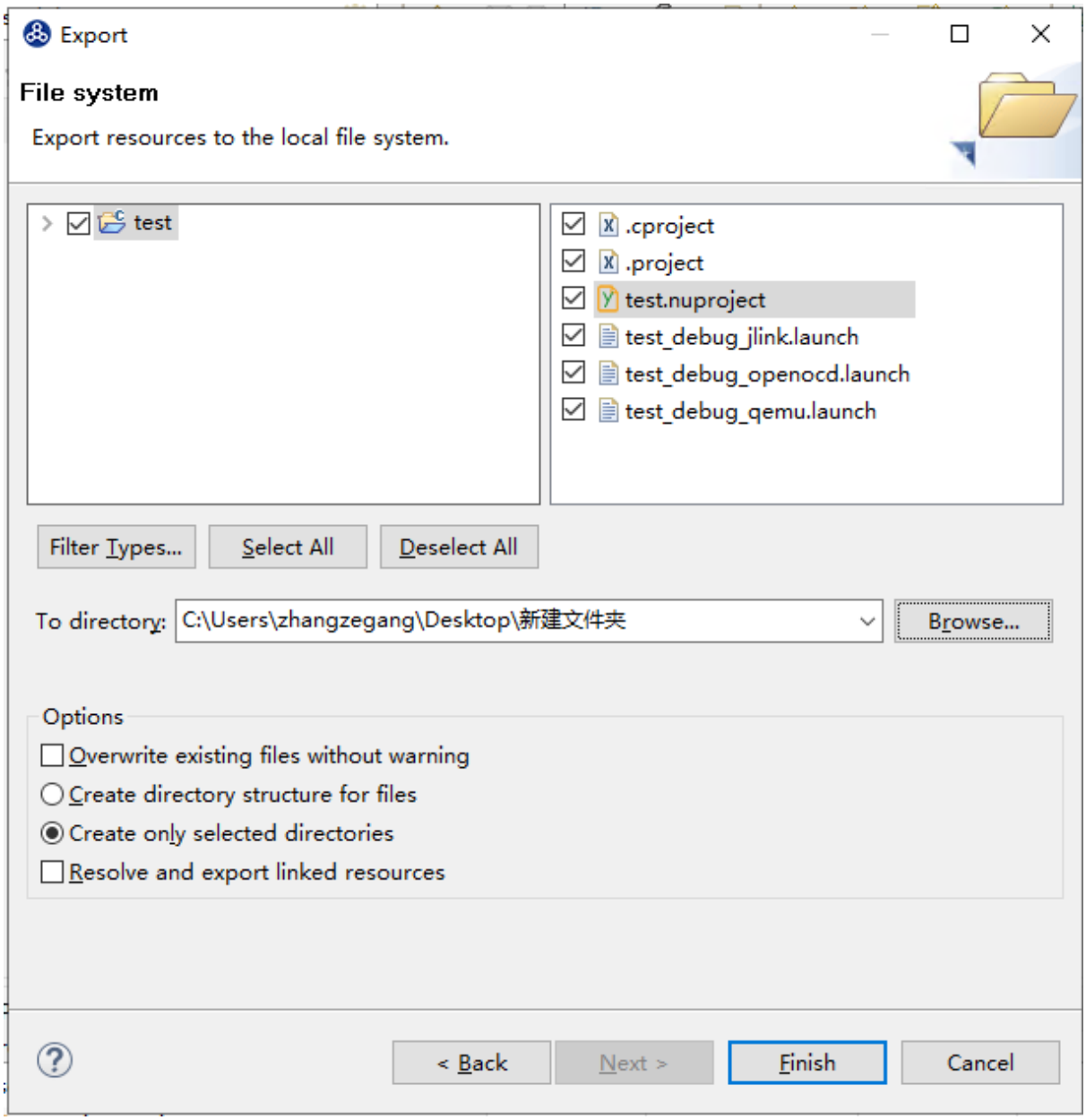
Nuclei Studio 2022.12 optimized project import to make project sharing easier. After a project is exported from the IDE, double-click the `.nuproject` file to quickly import and open it in Nuclei Studio.

First, export the project in Nuclei Studio 2022.12: right-click the project and select Export.

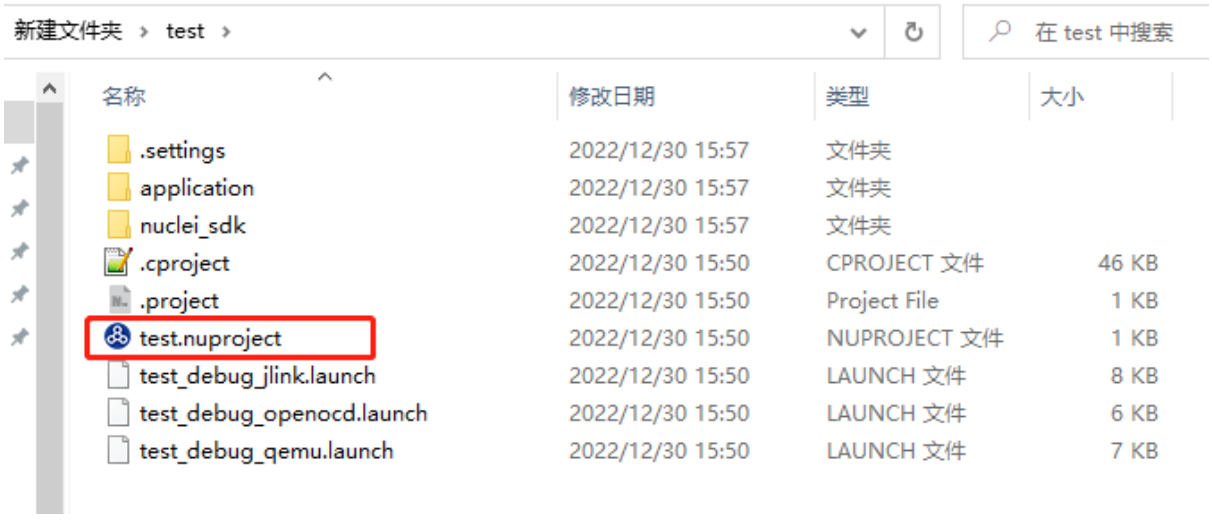


In the Export dialog, select `General->File System` and follow the wizard to export the project to the specified folder.


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



Next, import the project. The exported project contains a test.nuproject file; click this file to import the project into Nuclei Studio. For details, see *Import a Project via an Application Association File* (page 107).

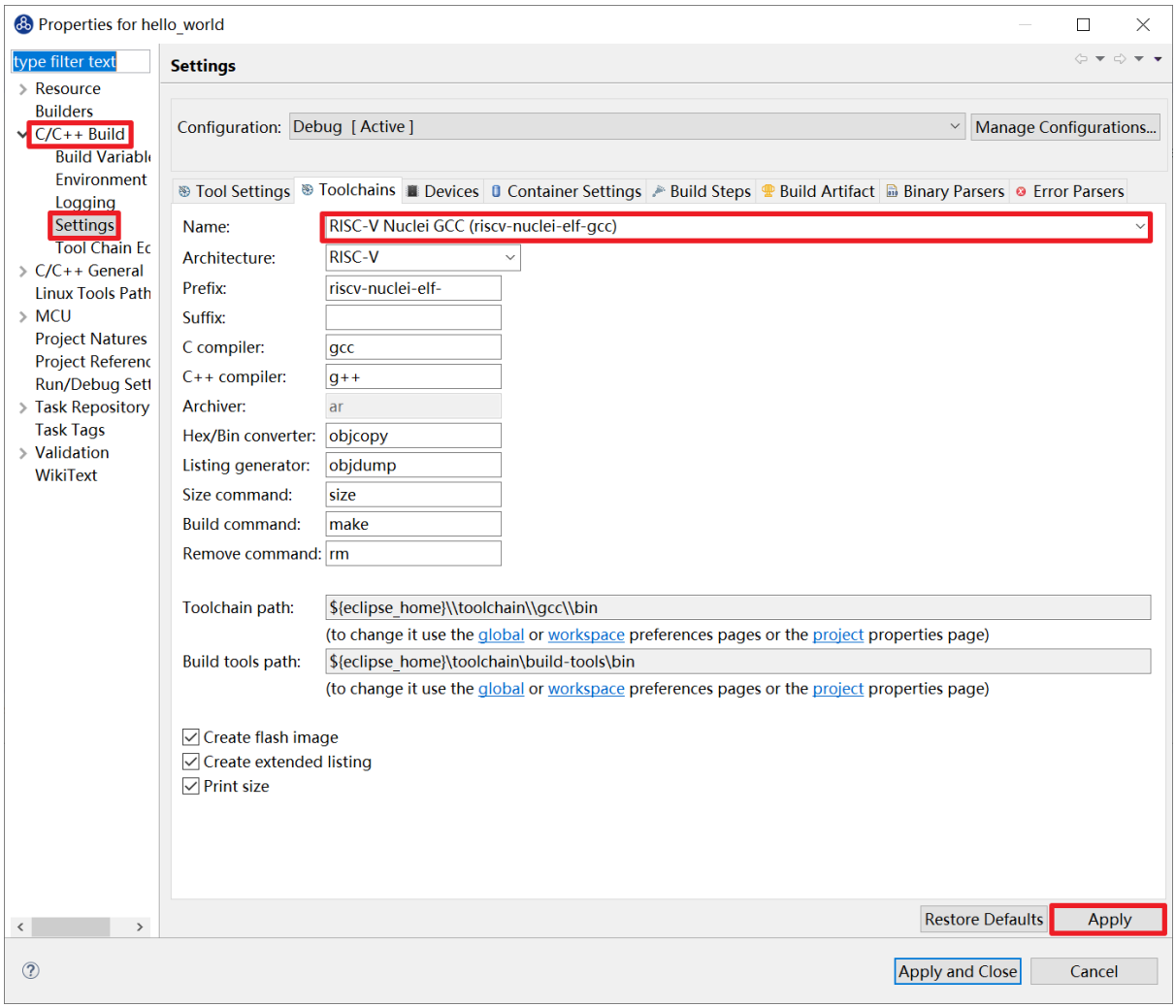


Importing Old Projects in Versions Earlier Than Nuclei Studio 2022.12

In Nuclei Studio 2023.10, the toolchain name was changed from `riscv-nuclei-elf-gcc` to `riscv64-unknown-elf-gcc`, and the GCC version was upgraded from GCC 10 to GCC 13.

Starting with version 2020.08, Nuclei Studio upgraded its official toolchain from `RISC-V Nuclei GCC (riscv-nuclei-elf-gcc)` to `GNU MCU RISC-V GCC (riscv-none-embed-gcc)`. Because the compiler prefix changed, projects created with IDE version 201909 or earlier must have their settings adjusted before they can be used in the new IDE. The following example uses a helloworld project generated with Nuclei Studio 201909 to describe the import procedure and the required settings changes:

- Import the helloworld project generated with version 201909. For the detailed import procedure, see Section 5.2.
- After importing the project, right-click it and select `Properties` to open the settings page. Select `C/C++ Build -> Settings`, open the Toolchains tab, and change the Name drop-down option to `RISC-V Nuclei GCC (riscv-nuclei-elf-gcc)`. Click Apply to save the change.



- Right-click the project again and select `Clean Project`, then `Build Project`.

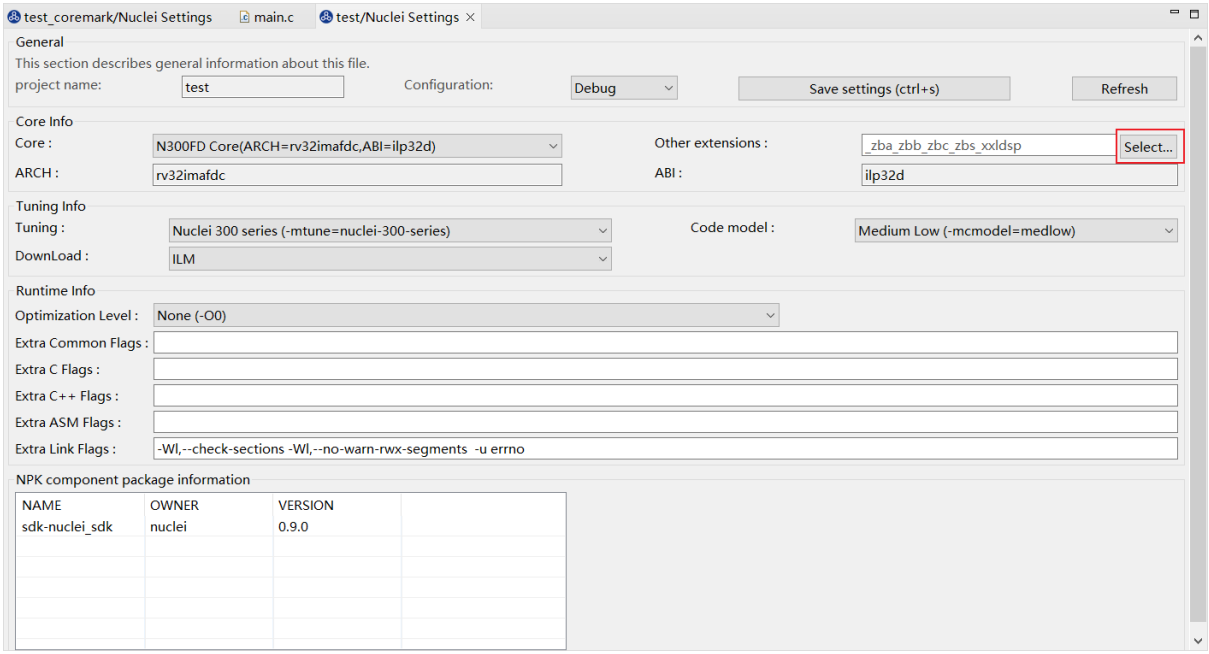
2.10.2 Extension Selector

Overview

Extension Selector is a modal dialog (1200x800) used to visually pick RISC-V ISA extensions, finally producing a `-march` string. It **does not compile**; it is only responsible for producing the march string, and whether the string is legal is ultimately decided by the compiler. * It is a native UI implemented purely with SWT/JFace, not a browser shell.

How to Open

On the **Settings page** of the project wizard or project properties, click the **Select Extensions** button.



UI Composition

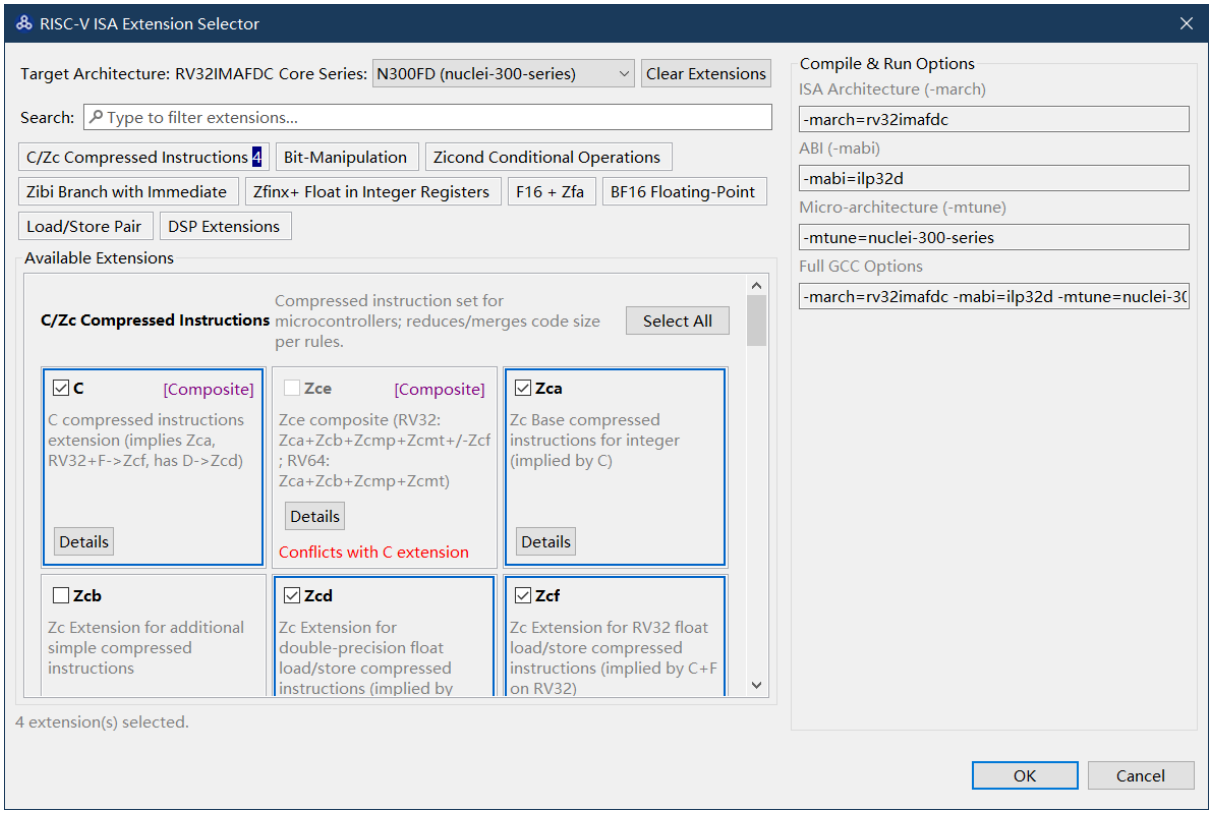
The dialog is divided into left and right parts:

Left panel:

Area	Function
Core selector	Select the target core. This is the only architecture-driven entry and determines RV32/RV64 and which extensions are available
Search box	Search extensions by name or mnemonic
Extension tree / card area	Show all extensions grouped by category; click to check
Action buttons	Clear, select all, OK, Cancel, etc.

Right panel:

Area	Function
ARCH preview	Shows in real time the `-march` / `-mabi` / `-mtune` produced by the current selection
Full GCC Options	The complete GCC command line, which can be copied directly



- **Card colors and states**

Extension cards use colors to distinguish types:

Color	Type
Blue	standard (standard extension)
Orange	custom (custom/vendor extension)
Purple	composite (composite extension, such as B, Zk, Zce)

A card has three states: unselected, selected, and **grayed out (not selectable)**.

Selecting Extensions

1. **Select the core**: select the target core in the core selector on the left. This step determines the available range of all subsequent extensions.
2. **Browse and search**: browse in the extension area, or use the search box to locate by name.
3. **Check extensions**: click a card to check it. Checking triggers automatic propagation (see *Automatic propagation rules for selection* (page 224)).
4. **Check the preview**: observe whether the `-march` / `-mabi` / `-mtune` on the right meet expectations.
5. **Confirm**: click OK, and the result is written back to the project configuration.

Note: When the Core Series is changed manually, the tool first **clears the extensions selected for the previous core**, then re-applies them according to the new core; if a preselected core is brought in from the Settings page when the dialog opens, the original extension combination of that core is retained.

- **Mutually exclusive groups (single choice)**

Some extensions are naturally mutually exclusive; only one in the same group can be kept, and selecting another prompts a replacement. The following three pairs of mutual exclusion are fixed at the code level:

Mutually exclusive group	Options (only one can be selected)
Vector base	`zve32x` `zve32f` `zve64x` `zve64f` `zve64d` `v`
Vector length	`zvl128b` `zvl256b` `zvl512b` `zvl1024b`
DSP	`xxldsp` `xxldspn1x` `xxldspn2x` `xxldspn3x`

There is also a **Clear Extensions** button: it keeps the base architecture letters (such as `rv32i` / `rv64imac`) and clears all other extensions, making it easy to select again from scratch.

- **Automatic propagation rules for selection**

When you check an extension, the tool automatically handles its relationship with other extensions —this is the core value of this dialog. Understanding the following four rules will keep you from being confused by "I didn't check this, how did it get checked by itself":

Rule	Behavior
Dependency auto-check	Selecting an item automatically brings in the extensions it depends on. For example, selecting Zcb / Zcmp / Zcmt automatically brings in Zca, because all three declare `dependencies: ['zca']`
Composite extension expansion	Selecting a composite extension expands it into its sub-extensions. For example, Zce expands into Zca, Zcb, Zcmp, Zcmt; on RV32 with `f` in the base architecture, Zcf is also added dynamically
Mutual exclusion and replacement	Options within a mutually exclusive group exclude each other. When a conflicting item is selected, a **Replace** option is provided, and you decide whether to replace
Reverse dependency cascade on uncheck	When you uncheck an item, other extensions that depend on it are unchecked as well. For example, unchecking the last vector compute extension (V or Zve*) cascades to uncheck vector crypto and the Zvl*b series, otherwise march would become an illegal combination

- **Filtering**

The UI provides triple filtering to quickly locate among the 80+ extensions:

Filter dimension	Description
Architecture	Filter by RV32 / RV64. Some extensions are architecture-specific (for example, Zcf, Zclsd, Zilsd are RV32-only)
Series	Filter by core series: N100 / N200 / N300 / N600 / N900 / N1000
Search	Search by name or mnemonic

- **Why an extension card is not selectable**

A card is grayed out usually for the following reasons; hover the mouse over the card to see the specific reason:

Reason	Example
Not applicable to the current architecture	Zcf, Zclsd, Zilsd, etc. are RV32-only
Architecture prerequisite dependency not met	Zcd requires `d` in the base architecture; Zcf requires `f`
Not part of the current core series	The extension is not supported by the series of the current core
Mutually exclusive with a selected item	Another item in the mutually exclusive group is already selected

- **march generation rules**

The `-march` previewed on the right is not a simple concatenation; it is folded and suppressed according to the specification. Understanding these rules explains why some checked extensions do not appear in the result:

1. **Base letters are ordered per the specification**: e.g. `rv32imaf`.
2. **When zcmp / zcmt are selected, ``c`` is removed from the base architecture**: because the Zcm* series does not coexist with the C extension.
3. **Composite folding**: composite extensions such as B, Zk, Zkn, Zks, Zvkn, Zvks are folded for display.
4. **Implicit C suppression**: when the base architecture still contains `c`, `zca` / `zcf` / `zcd` do not appear again.

About rule 4: on RV32+F, `C` itself equals all the instructions of Zca + ZCF, so writing `zca` or `zcf` again when the base architecture already contains `c` is redundant, not a missing feature.

The actual effect of rule 2: when Zce is checked on N300F (base architecture `rv32imafc`), the result is

```
rv32imaf_zca_zcb_zcf_zcmp_zcmt
```

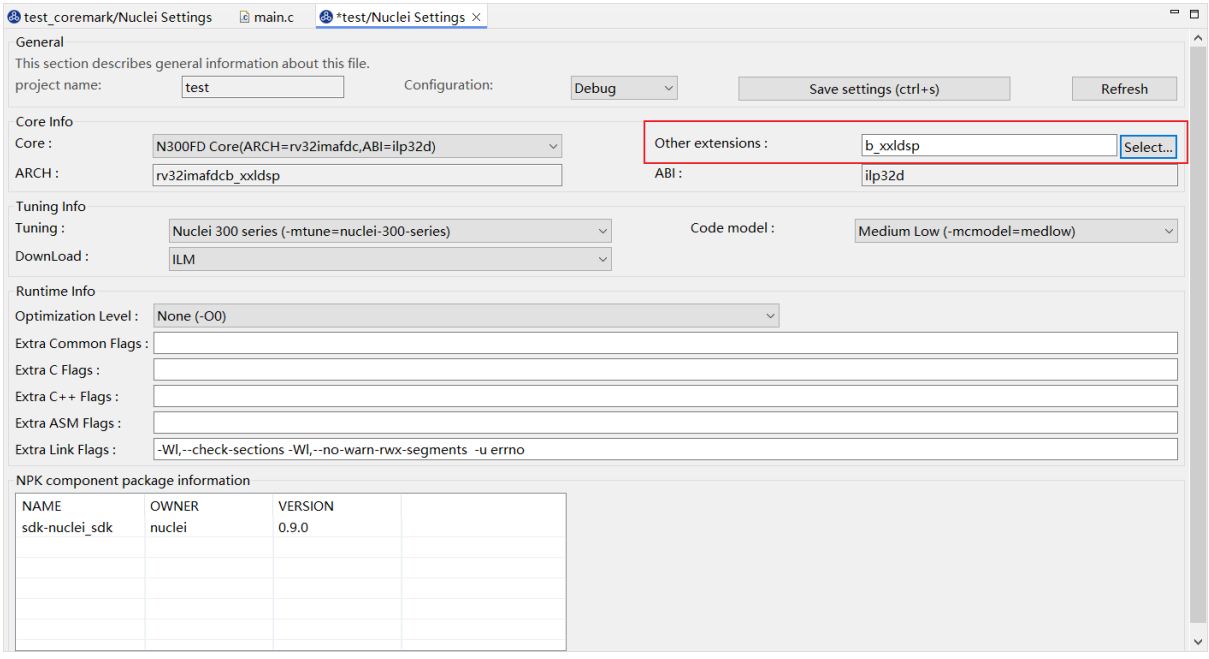
That is, the `c` in the base architecture is removed and expressed explicitly by Zca, and Zcf is also explicitly kept. This is exactly the design intent of Zce —to replace the full C with the Zca subset.

Quick Start

From opening to generating the march string

1. Open the **Settings page** in the project properties: find it in the project → go directly to the **Nuclei Settings** page.
2. Click the **Select Extensions** button.
3. Select the target core in the **core selector** on the left —this step determines RV32/RV64 and which extensions are available.
4. Browse or search in the extension area and click a card to check it; to save effort, you can directly use the **Core Series** drop-down at the top to apply a preset.
5. Observe whether the **-march / -mabi / -mtune** preview on the right meets expectations (checking automatically propagates dependencies and expands composite extensions).
6. Click **OK**; the result is written back to the project configuration, which is equivalent to passing `--config nuclei_archext=...` on the command line.

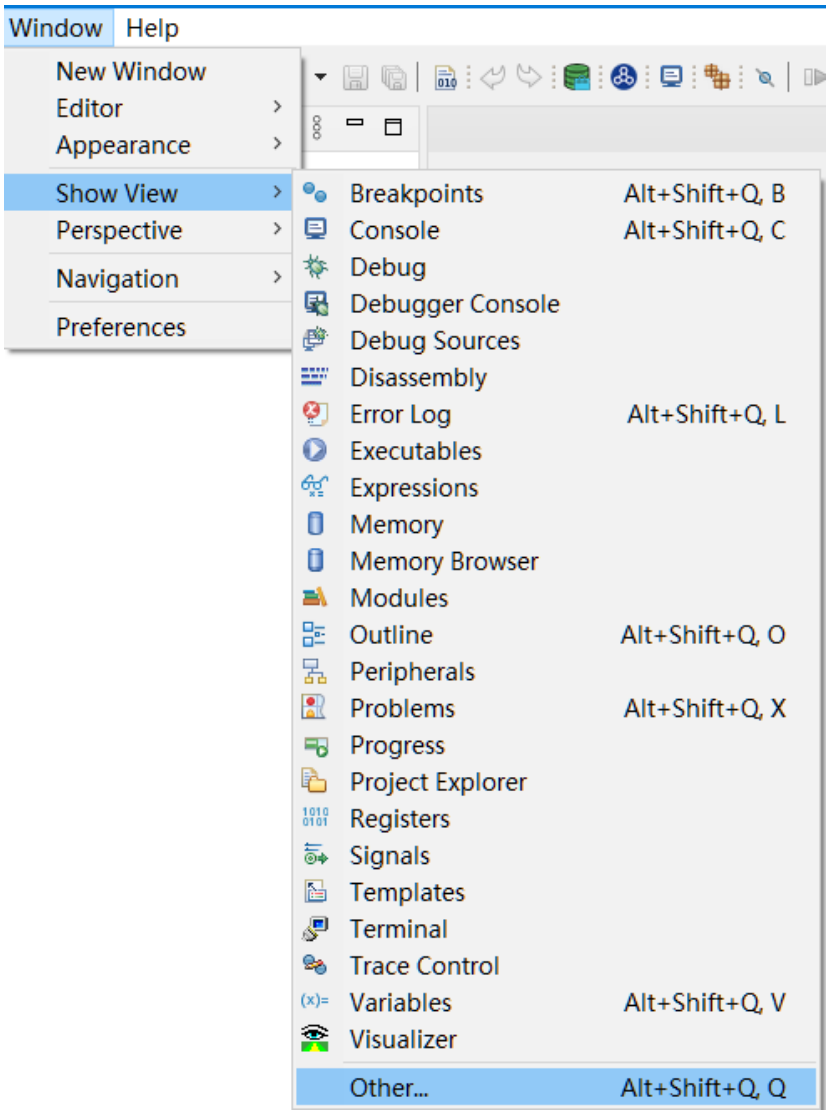
Afterwards, the selected extensions are applied when building the project.



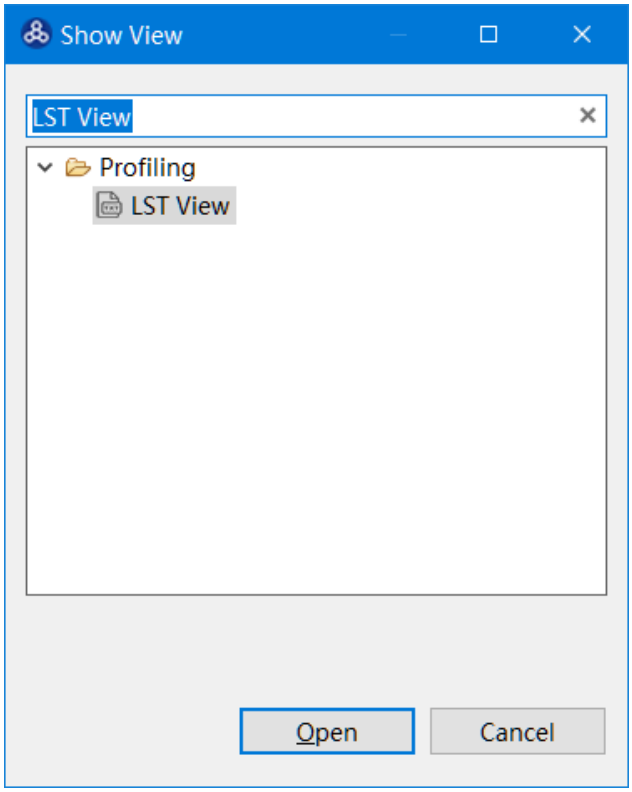
2.10.3 LST View

Nuclei Studio 2024.06 integrates the LST View tool. LST View can be used on its own or invoked from the Trace tool or the GProf tool. It helps you view LST files and link them with the source code.

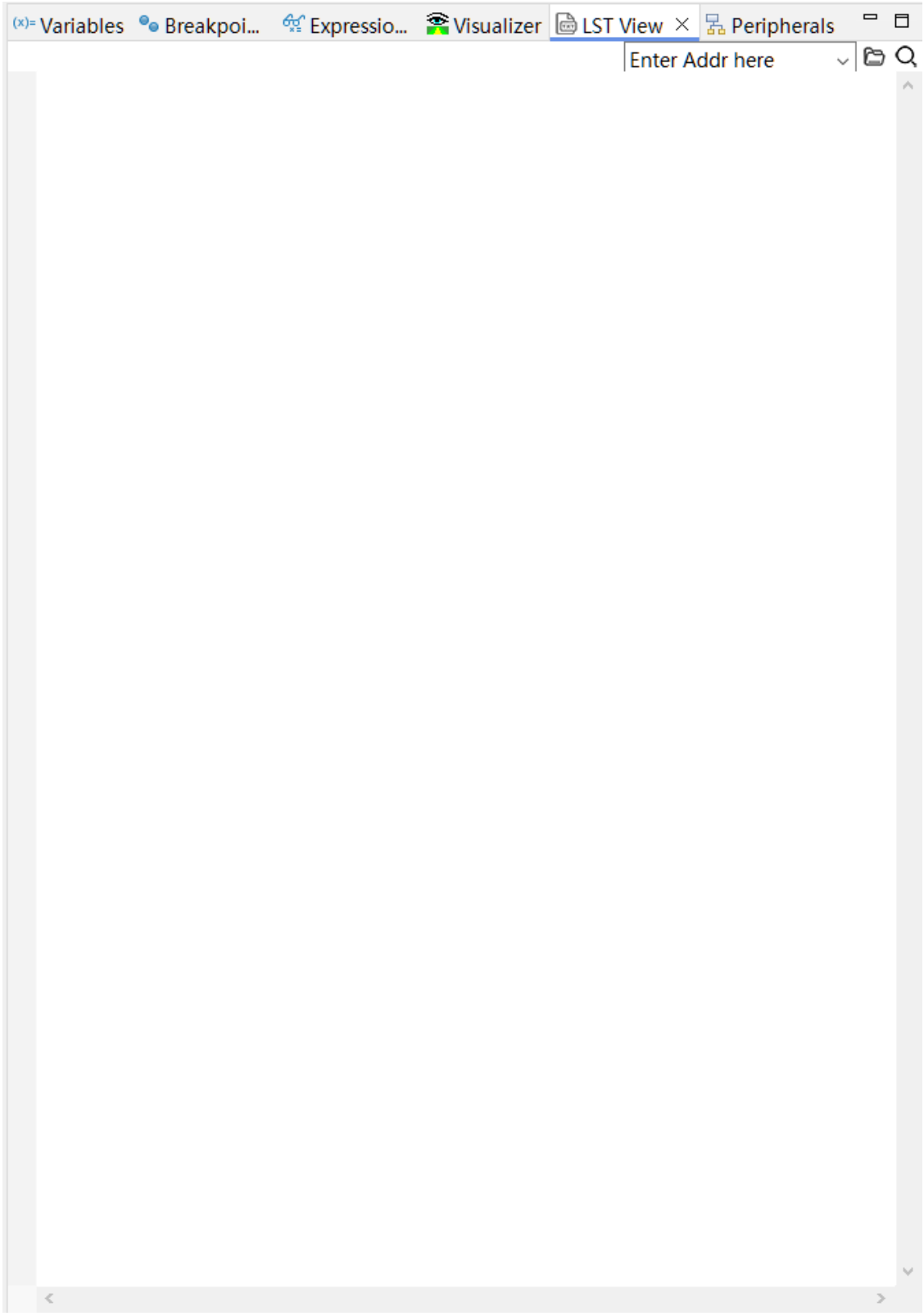
In Nuclei Studio, go to `Window -> Show View -> Other` and search for `LST View` in the Show View dialog.



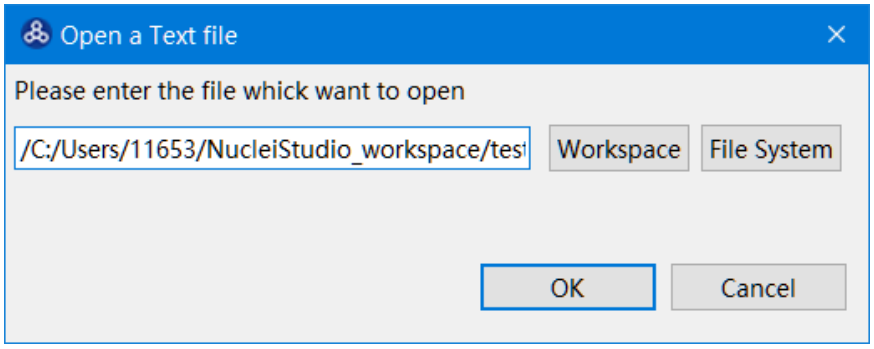
Open the LST View tool.



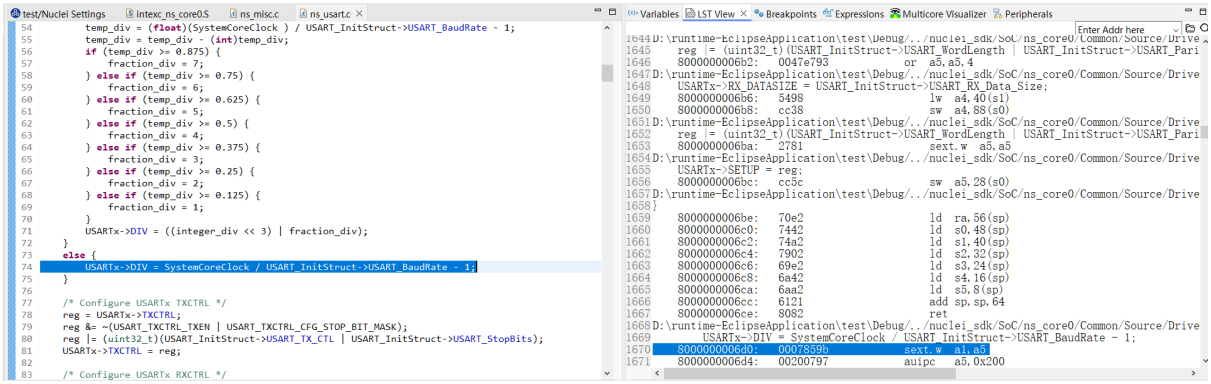
LST View has a toolbar at the top with a search drop-down box for entering an Addr to search for, an Open File menu that opens a file selector for choosing the `.lst` file to open, and a Find menu for searching within LST View.



Use the Open File menu to locate the lst file in the file selector. The file is usually located in the Debug directory after the project is built.



After opening the file, you can search within it. When you select a line of text that contains a valid Addr, LST View uses that Addr to locate the corresponding source file and line number, opens the source file in the editor, and places the cursor on the corresponding line. This links the source file to the lst file so you can view the two together.



2.10.4 Code Coverage and Profiling

Nuclei Studio 2023.10 and later integrates Eclipse Linux Tools[8], whose features have been partially optimized so that Nuclei Studio projects can use the Code Coverage and Profiling related features. In Nuclei Studio 2024.06, these features were further optimized and upgraded to make them easier to use.

For tutorials on using Coverage, Profiling, and Call Graph, see *Using Coverage, Profiling, and Call Graph* (page 234).

For details about Eclipse Linux Tools, see Eclipse Linux Tools[9]

If you encounter any problems during use, check https://github.com/Nuclei-Software/nuclei-studio or submit an issue.

Note: The **Nuclei System Technology WeChat Channel** contains a video on **How to Use Code Coverage and Profiling in Nuclei Studio**. Search for the **Nuclei System Technology WeChat Channel** in WeChat to view it.

---

[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

About Code Coverage

The Code Coverage feature in Nuclei Studio relies on the gcov tool provided by the GCC compiler to view the code coverage of specified source files. It helps developers determine whether their test cases are sufficient and cover all branches and paths of the code under test.

In Nuclei Studio, build the file or folder you want to analyze with the `--coverage` or `-coverage` compile option. When the program runs on a real development board, the semihost feature reads and writes files on the host computer, so the required coverage files (gcda/gcno files) can be collected. Alternatively, use the profiling library provided by Nuclei SDK[10] to print the coverage data to the serial port; the IDE then parses the data and saves it to the host.

Caution:

- The compile option `--coverage` or `-coverage` should only be added to a specific application directory or source file, not to the entire project. Otherwise, the program consumes a large amount of memory at runtime and fails to run.
- When the 2025.02-gcc 14.x or later compilers are used together with the Nuclei SDK Profiling library (<=0.8.0), the coverage feature does not work properly and no coverage information is produced. For the fix, see https://github.com/Nuclei-Software/nuclei-sdk/commit/5aaae0d5a7629013a235657ed3bce3dce18e8e0b

Note: As of the Nuclei Studio 2025.10 release, the library implementation of Nuclei SDK 0.9.0 Profiling/Coverage[11] does not yet support the clang compiler. If you use the clang compiler for testing, an error is reported. This should work normally once Nuclei SDK adds the related support in a future release.

Note: Due to the GCC version upgrade, projects created with the latest SDK may behave abnormally when the Code Coverage feature is enabled. This issue has been fixed in Nuclei Studio 2025.10.

Note: This fix only supports SDK version 0.8.1 and above and is not backward compatible with older projects. For projects created with an older SDK, manually modify the configuration to adapt to the new version. Refer to the changes in the following GitHub commit and adjust your project accordingly: https://github.com/Nuclei-Software/nuclei-sdk/commit/5aaae0d5a7629013a235657ed3bce3dce18e8e0b

- The `.gcno` file is generated when source code is compiled with the `-ftest-coverage` option of the GCC compiler. It contains information for reconstructing the basic block graph and assigning source line numbers to blocks.
- The `.gcda` file is generated when an object file compiled with the `-fprofile-arcs` option of the GCC compiler runs. A separate `.gcda` file is generated for each object file compiled with this option. It contains arc transition counts, value profile counts, and some summary information.

In general, the `--coverage` or `-coverage` option alone instructs the compiler to produce the files described above. Note that `*.gcda` files are produced at runtime, so the runtime environment must support file reading and writing. Here we use semihosting: the files are written to the host through the semihost feature of OpenOCD.

Note: When collecting coverage, compile with `O0` so that the coverage information is as accurate as possible.

For details about the Code Coverage feature, see

- Gcov Intro (Using the GNU Compiler Collection (GCC))[12]

[10] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/Components/profiling
[11] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/Components/profiling
[12] https://gcc.gnu.org/onlinedocs/gcc/Gcov-Intro.html

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

About Profiling

The Profiling feature in Nuclei Studio relies on the gprof tool in the GCC compiler and binutils to view the running time and call counts of functions in a specified file, as well as their call relationships. gprof helps identify program bottlenecks for performance optimization. It collects data while the program runs and generates a report showing the percentage of CPU time each function occupies and the call relationships between functions.

In Nuclei Studio, build the specified source files with the `-pg` compile option. When the program runs on a real development board, the semihost feature reads and writes files on the host computer, so the required coverage files (gcda/gcno files) can be collected. Alternatively, use the profiling library provided by Nuclei SDK[15] to print the coverage data to the serial port; the IDE then parses the data and saves it to the host.

Note: The compile option `-pg` should only be added to a specific application directory or source file, not to the entire project. Otherwise, the program consumes a large amount of memory at runtime and fails to run.

Producing the `gmon.out` file requires compiler support, and the program must actually run on a board whose runtime environment supports file reading and writing before Profiling can produce useful results.

For details about the Profiling feature, see

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

About Call Graph

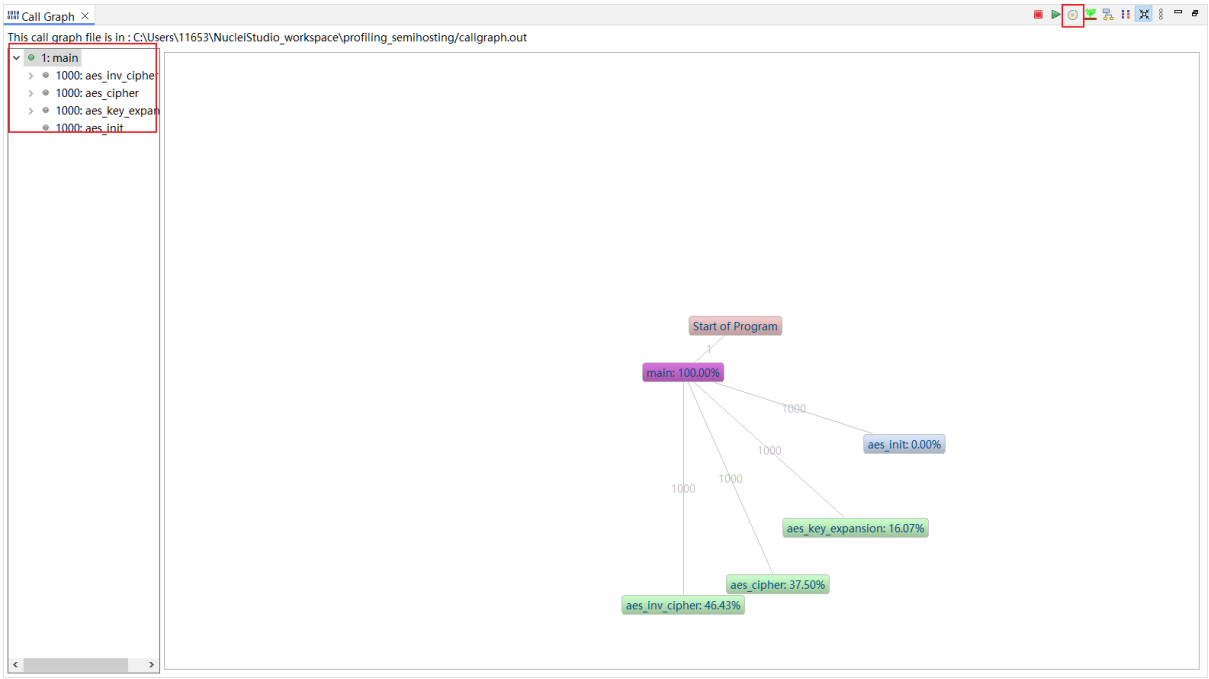
Call Graph is a powerful tool that gives developers an intuitive view of the call relationships between functions or methods in a program. With Call Graph, developers can quickly identify frequently called functions, key entry points, and dependencies between functions. In Nuclei Studio, Call Graph obtains the program ' s call relationships mainly by analyzing Profiling data.

In Nuclei Studio, go to `Window -> Show View -> Other`, search for `Call Graph` in the Show View dialog, and open the `Call Graph` tool. The `Call Graph` tool provides multiple views; the most commonly used are described below.

Radial View

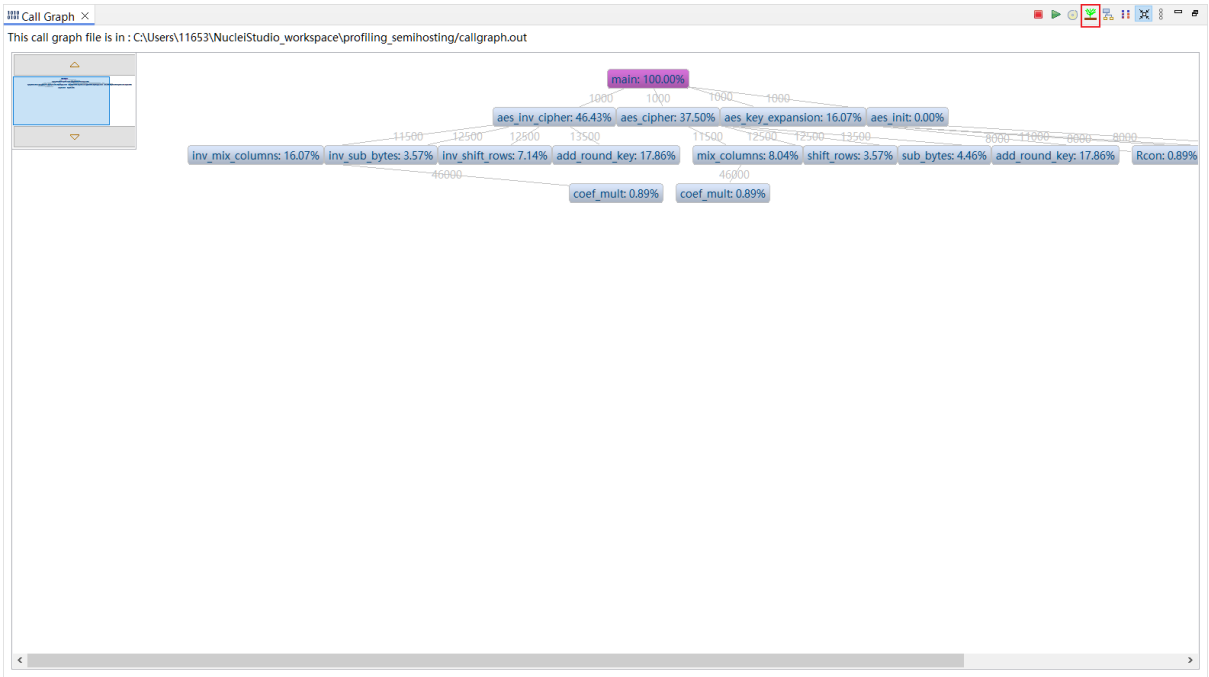
This view shows the call relationships of the program. Double-click a parent node in the menu on the left, and the area on the right displays all call relationships starting from that parent node. You can also use the menu to view the selected call relationship in the other views.

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



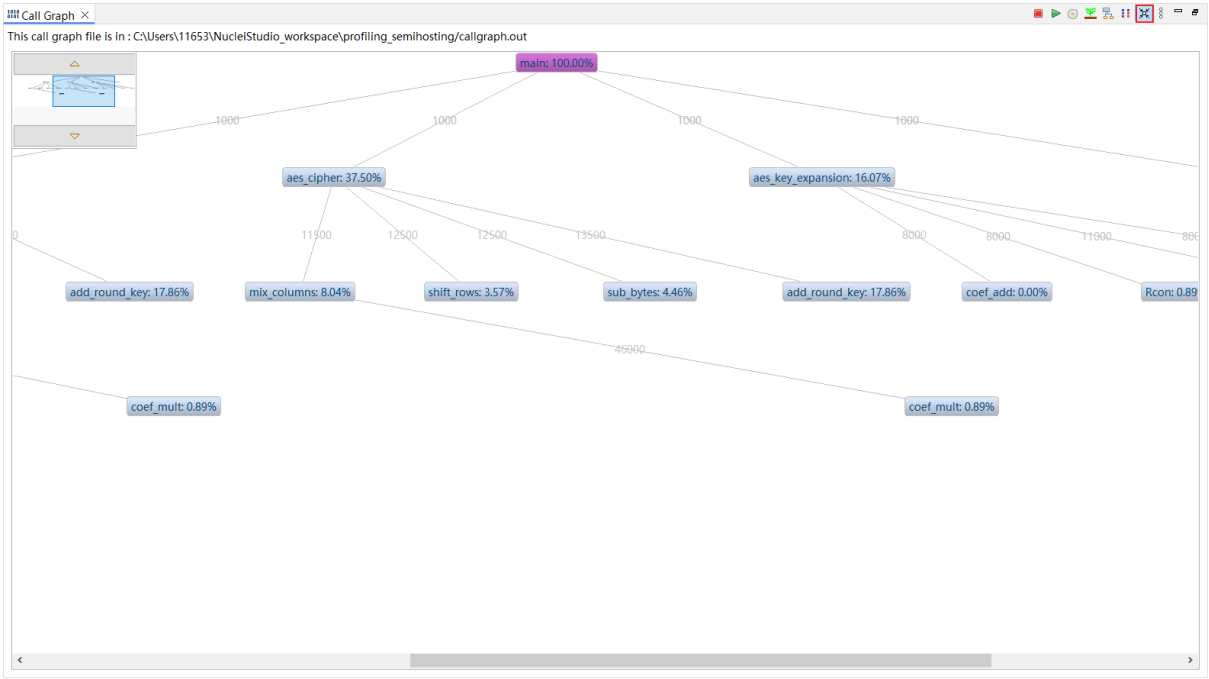
Tree View

Shows the call relationships, time consumption percentages, call counts, and other information for the program selected in Radial View. Select a function to view its parent nodes, child nodes, and other information.



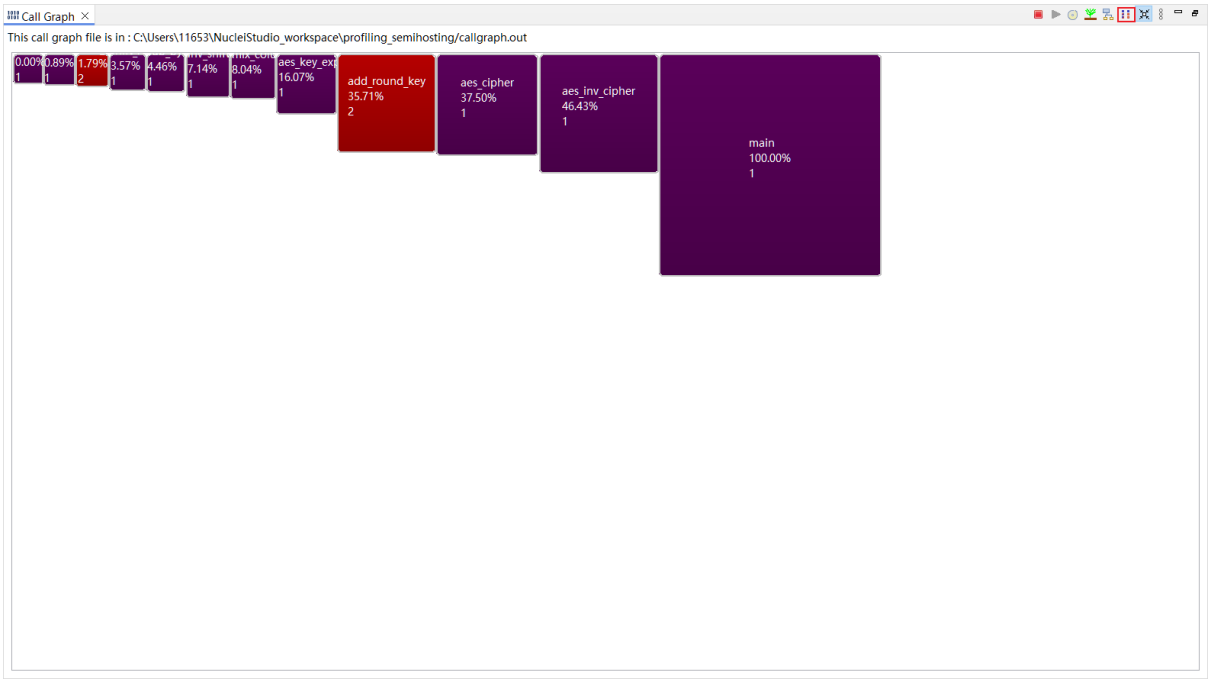
Level View

Similar to Tree View, it shows the call relationships and call counts of the program.



Aggregate View

Shows the program ’ s time consumption visually as a square diagram.

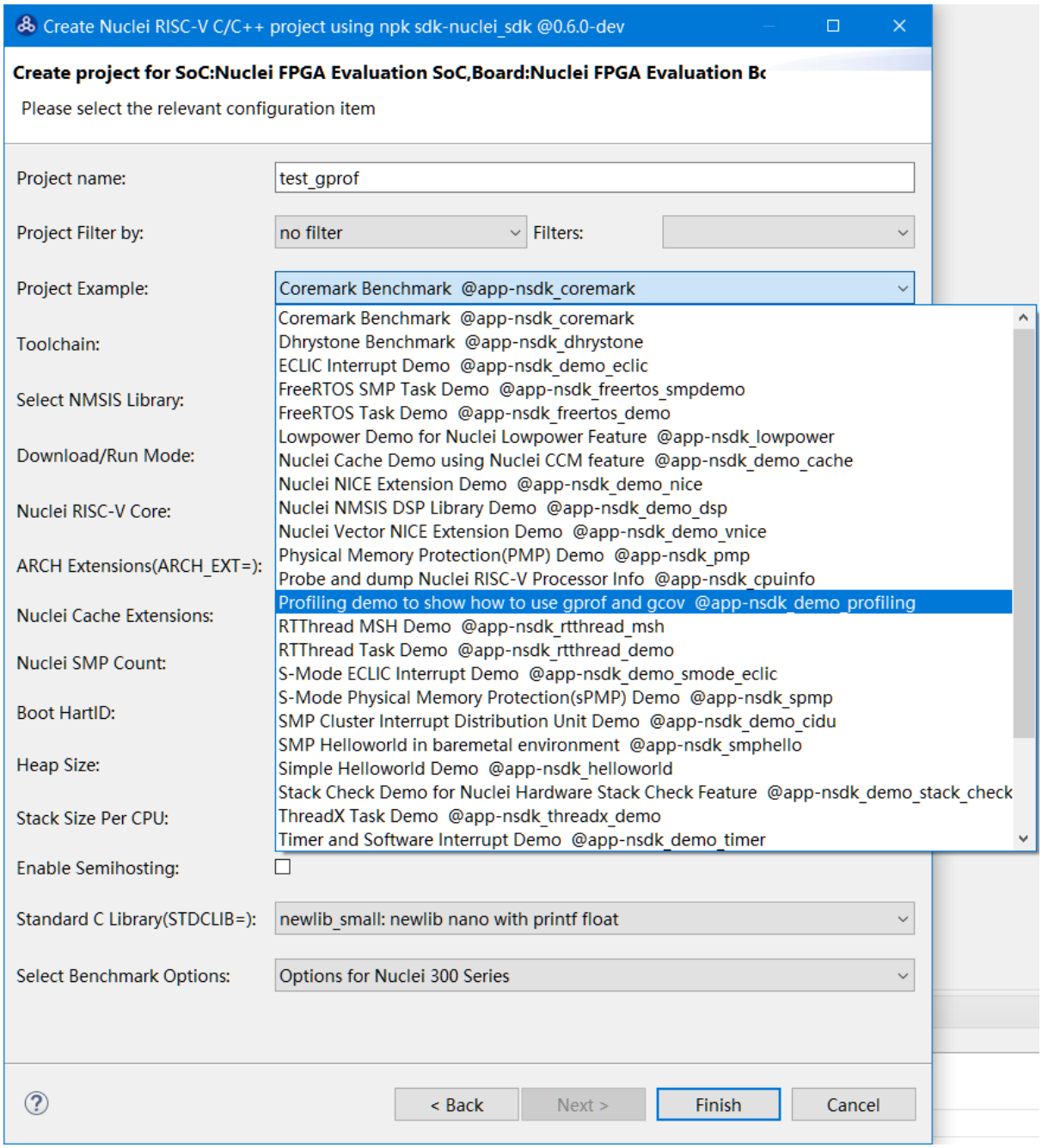


Using Coverage, Profiling, and Call Graph

Using Coverage, Profiling, and Call Graph in Nuclei Studio 2024.06 is straightforward. The following examples use Nuclei Studio 2024.06 and nuclei_sdk 0.6.0 to demonstrate two ways of using the Coverage, Profiling, and Call Graph tools.

Using via Serial Port

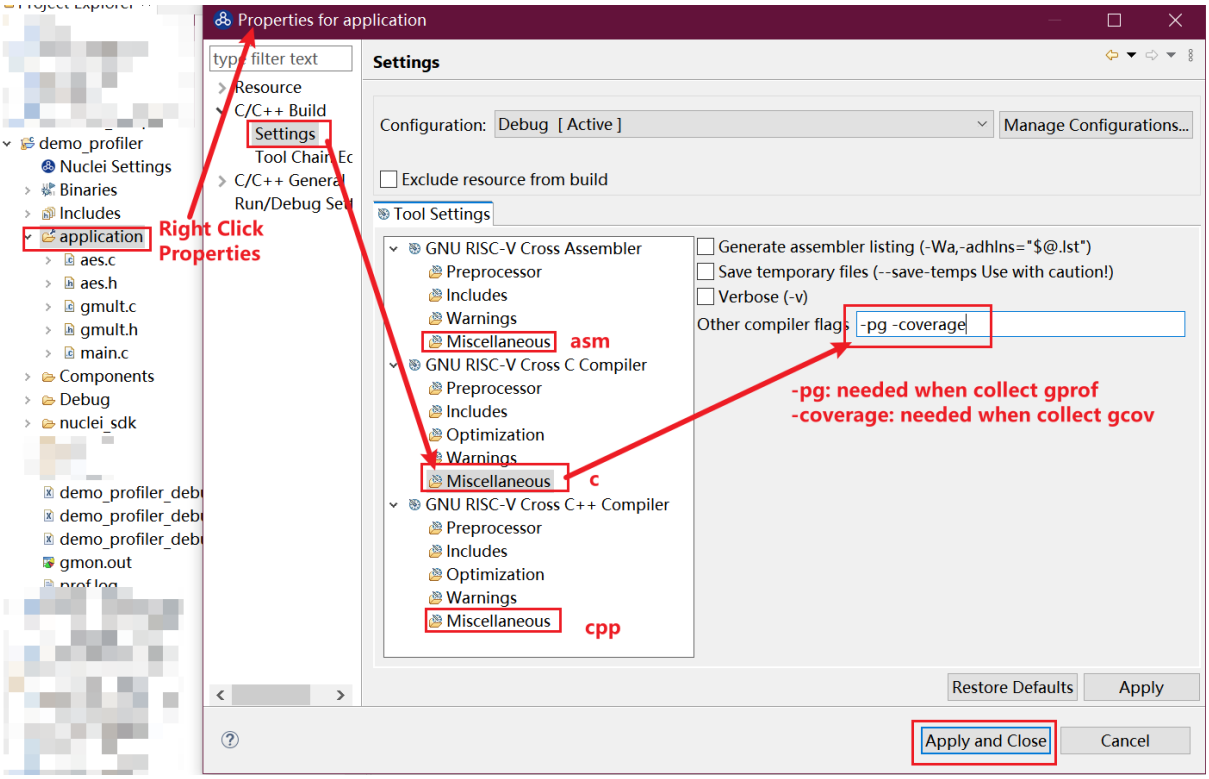
nuclei_sdk 0.6.0 and later versions include a `Profiling demo to show how to use gprof and gcov` test project. After installing nuclei_sdk 0.6.0 in Nuclei Studio, you can create this test project. For details about the `Profiling demo to show how to use gprof and gcov` test project, see demo_profiling[18].



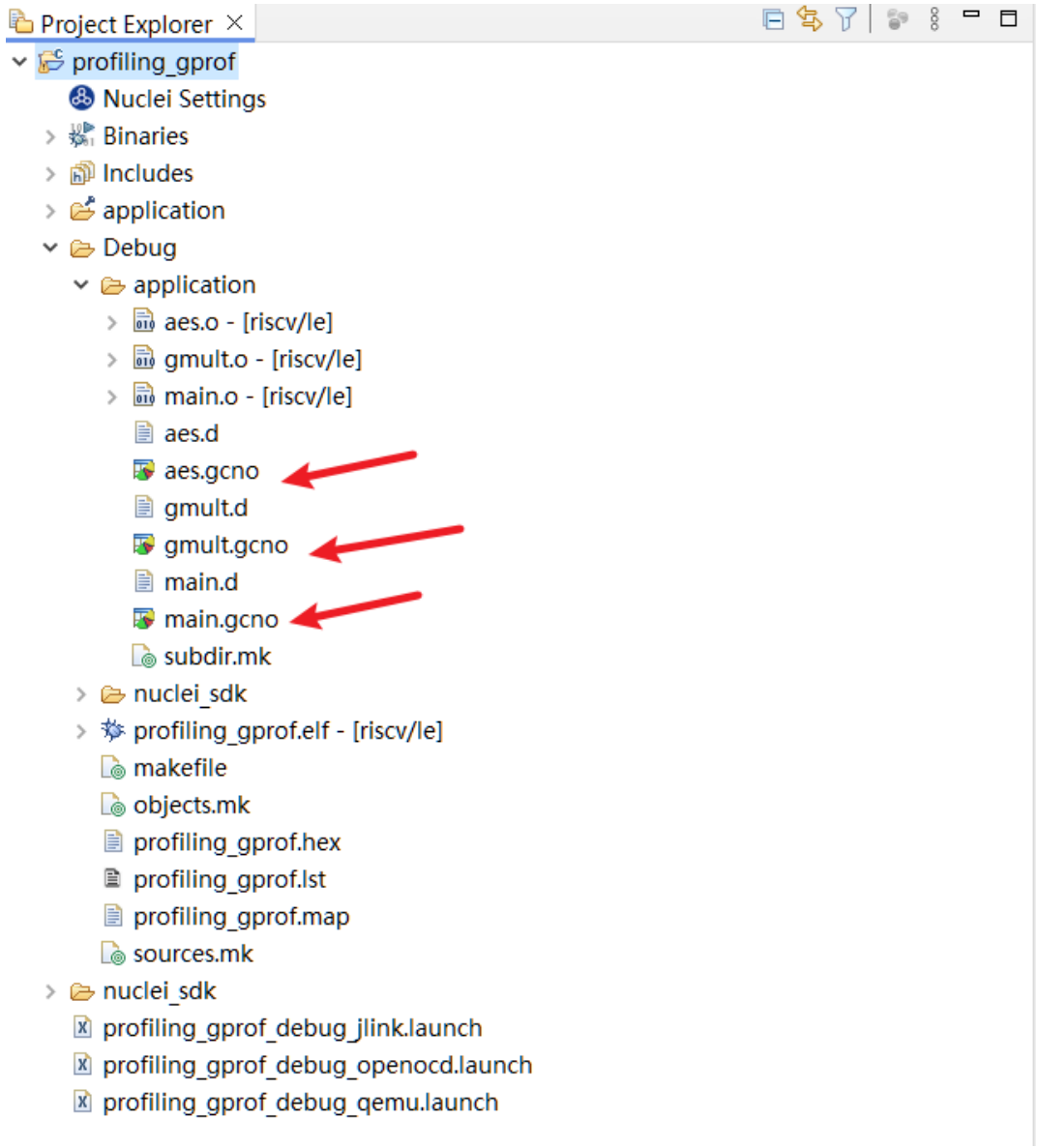
After the project is created, set the `-pg`, `--coverage`, or `-coverage` compile option for the files or folders you want to analyze, then build the project.

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

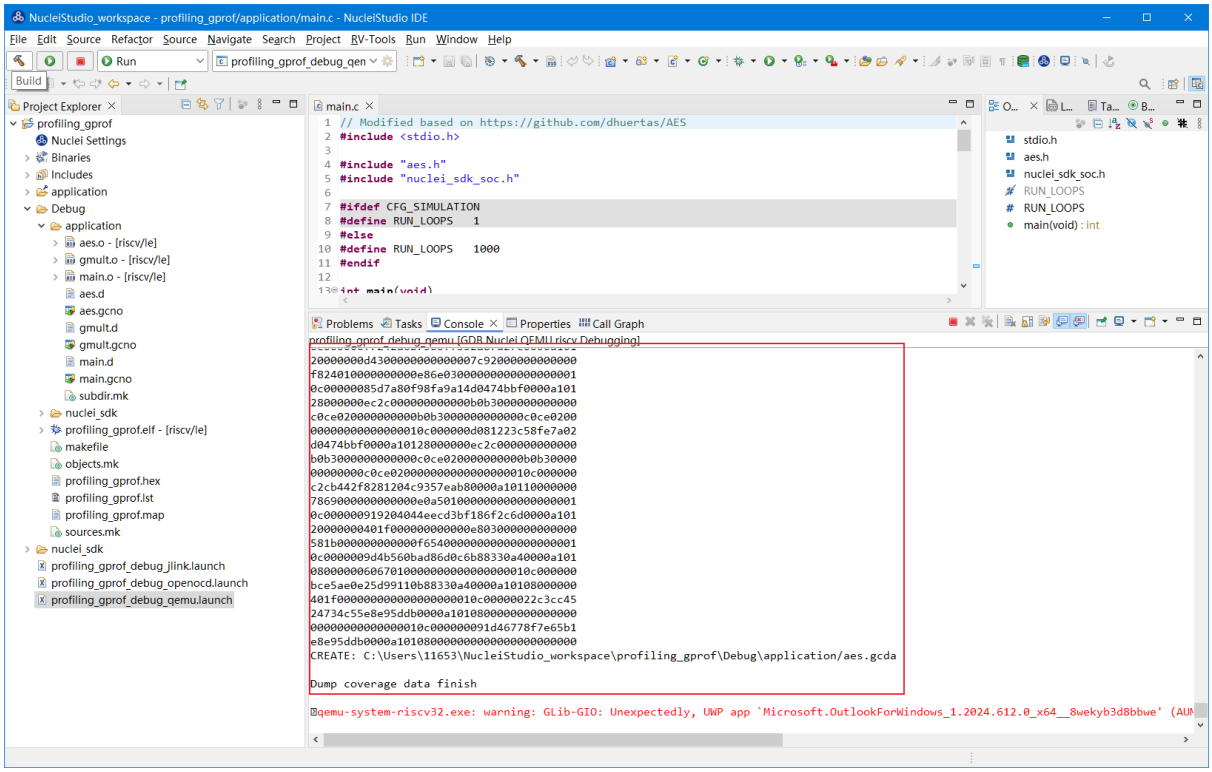
Note: The compile options `-pg`, `--coverage`, or `-coverage` should only be added to a specific application directory or source file, not to the entire project. Otherwise, the program consumes a large amount of memory at runtime and fails to run.



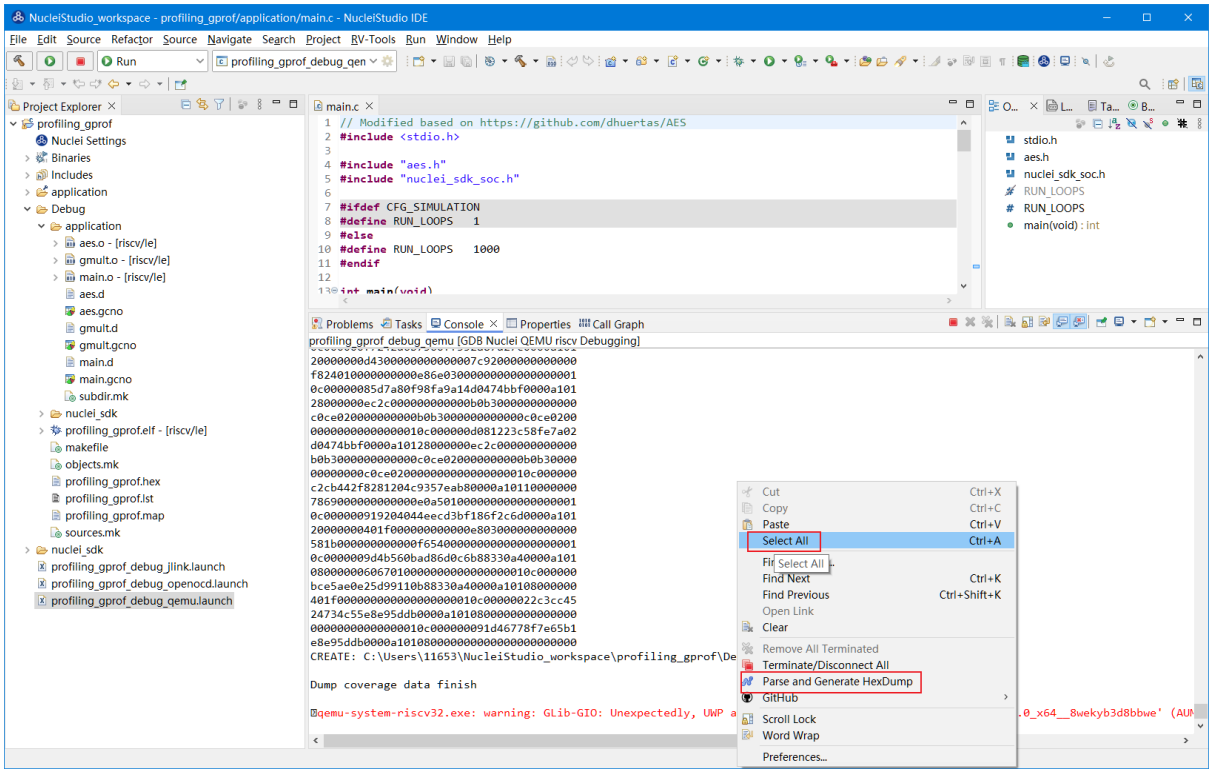
In the Debug directory of the successfully built project, you can see that several `.gcno` files have been generated.



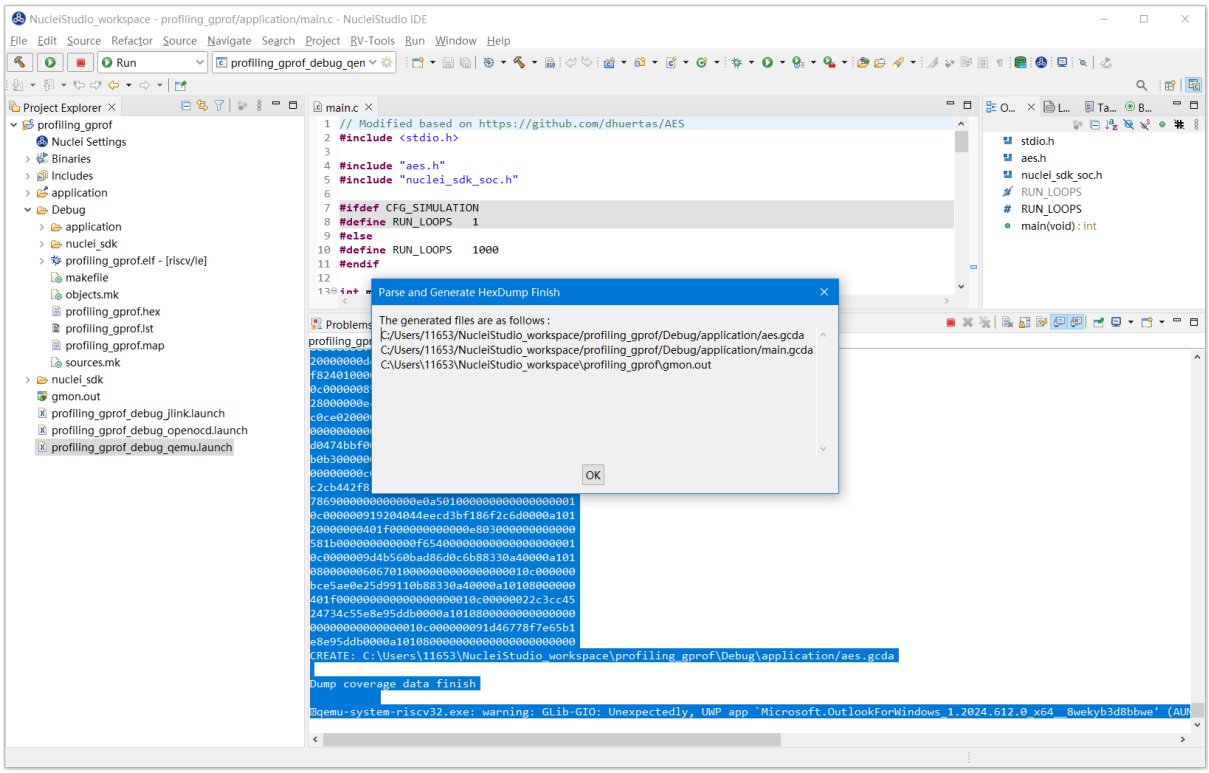
After the project is built, run it under QEMU or debug it on a real development board. This example runs the program under QEMU; the Profiling information is output in the Nuclei Studio Console window. When debugging on a development board, the Profiling information appears in the serial port output.



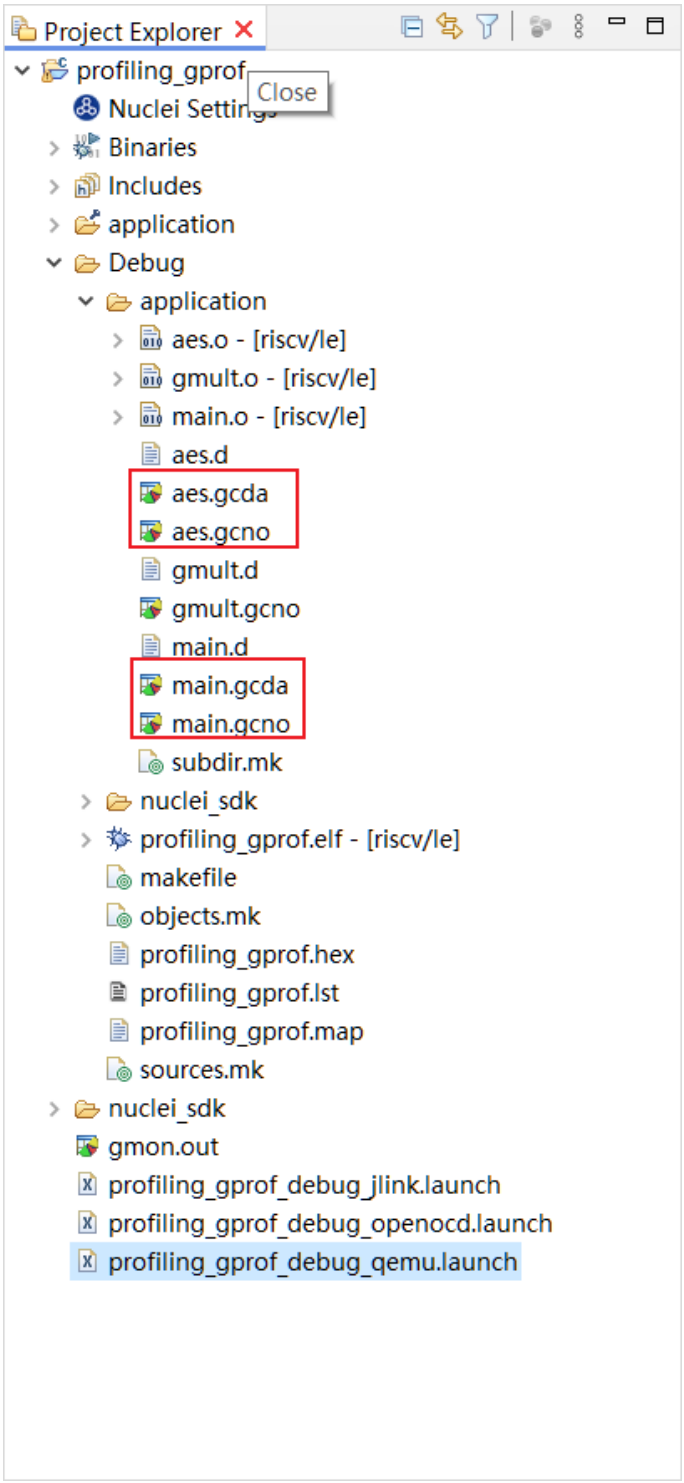
The output Profiling information must be parsed before Nuclei Studio can read it. Right-click in the Console window and select Select All from the context menu to select all output, then right-click again and select `Parse and Generate HexDump`.



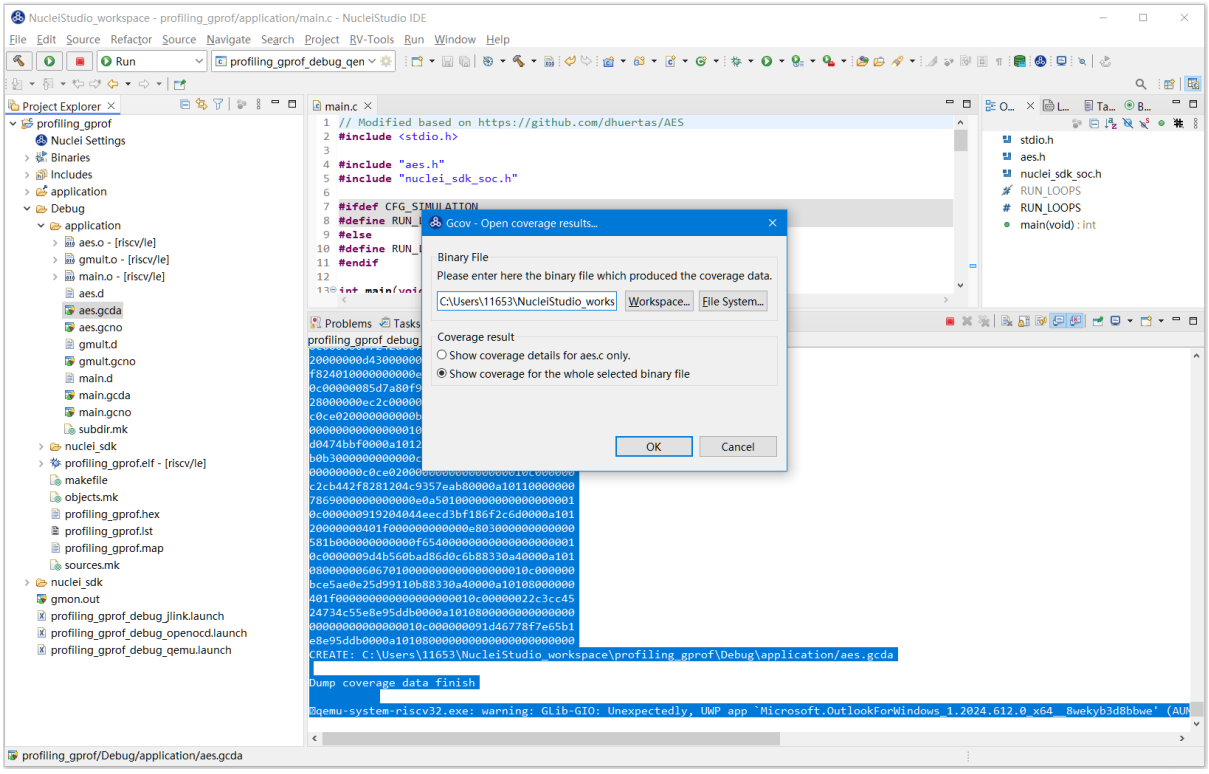
Nuclei Studio then analyzes the output and stores the results in the corresponding files.



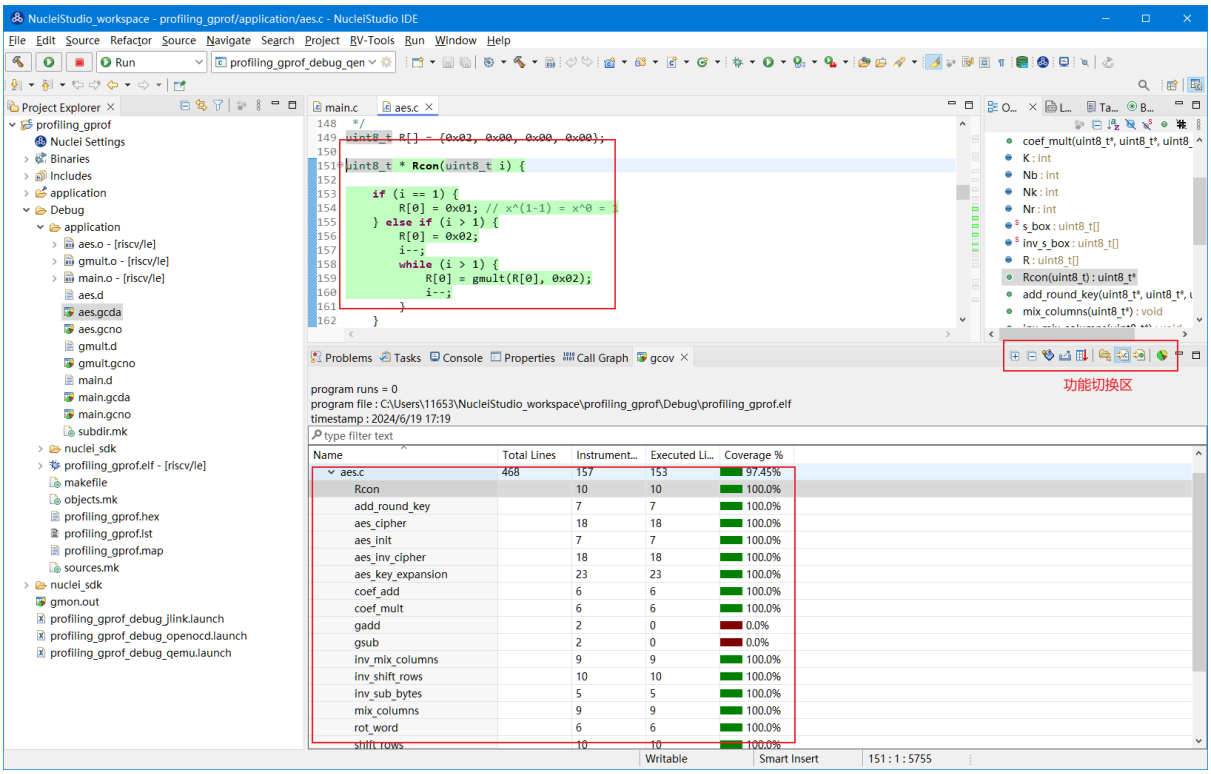
Check the Debug directory of the project again; the corresponding `.gcda` files have been produced.



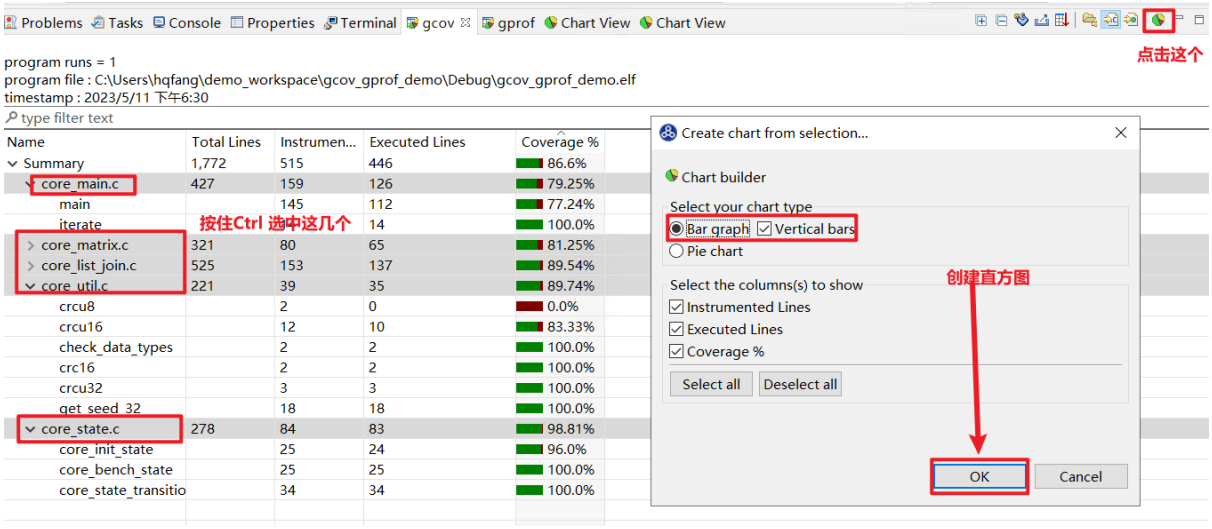
Double-click a `.gcda` file to open the Gcov tool and view the analysis results for the application. The results show whether each file or method was executed during program execution, as well as data such as the code execution coverage ratio.



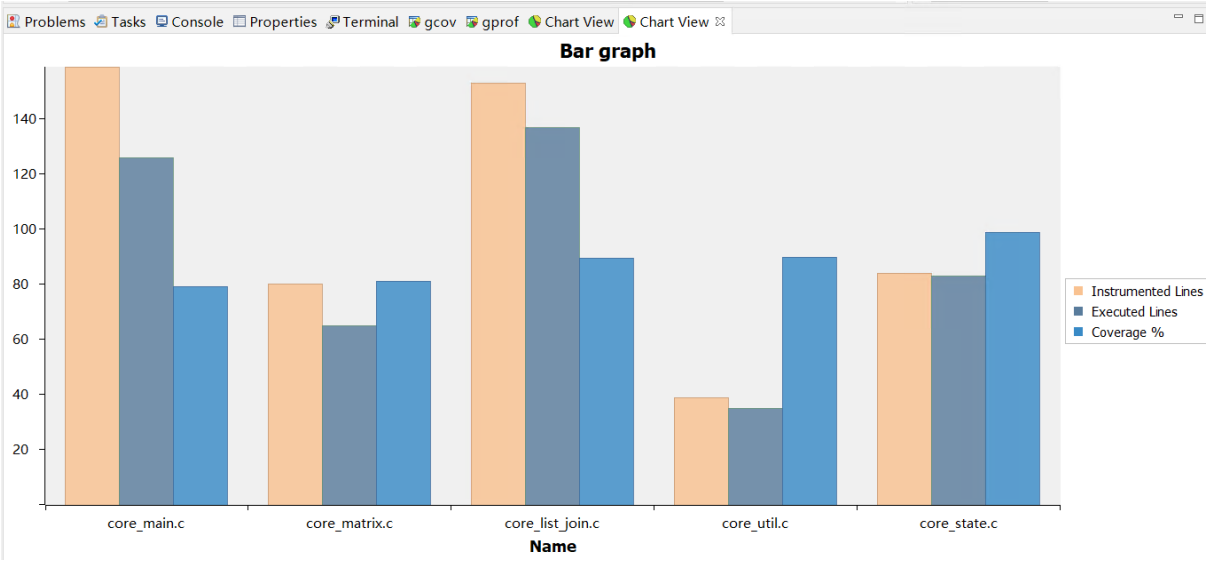
Double-click a row in Gcov, and Nuclei Studio automatically opens the corresponding file and colors its code: green indicates code that was executed during program execution, and red indicates code that was not. Use the Gcov results to optimize the code accordingly.



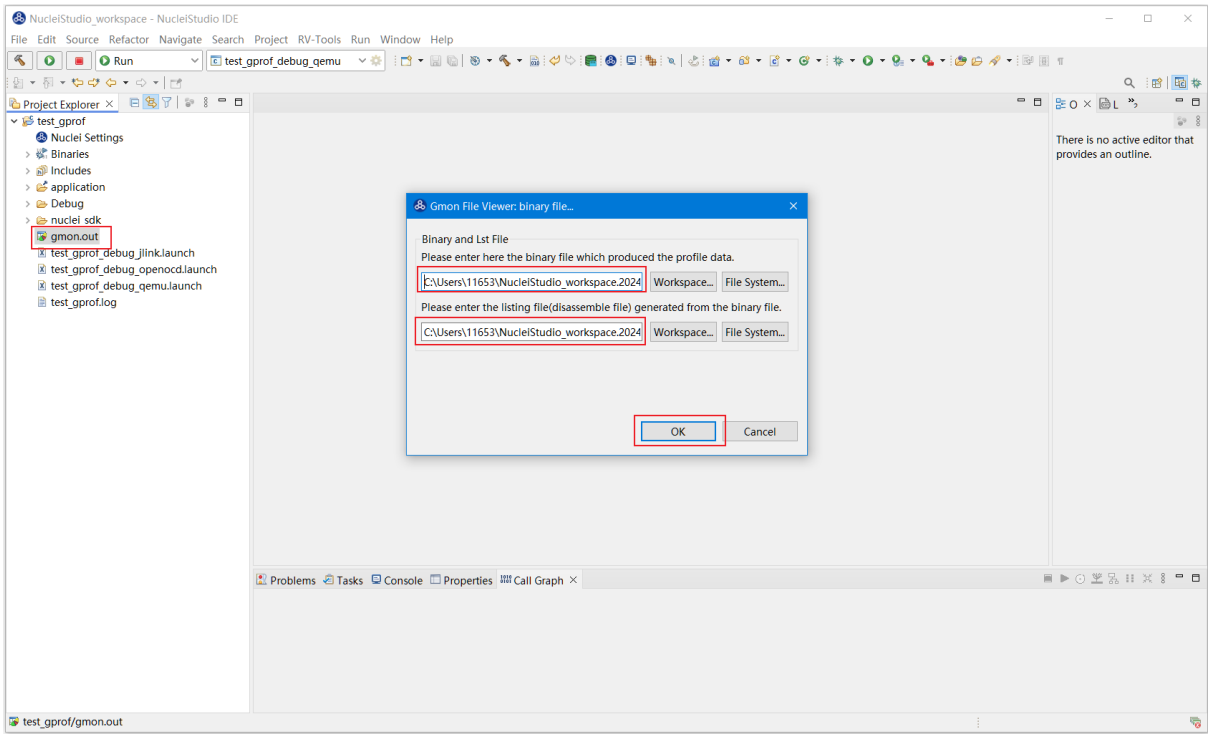
Code coverage also provides a histogram view of the data. Select the data item you want to view, click the histogram menu item, and configure it as needed.



The code coverage histogram is then displayed in Nuclei Studio.

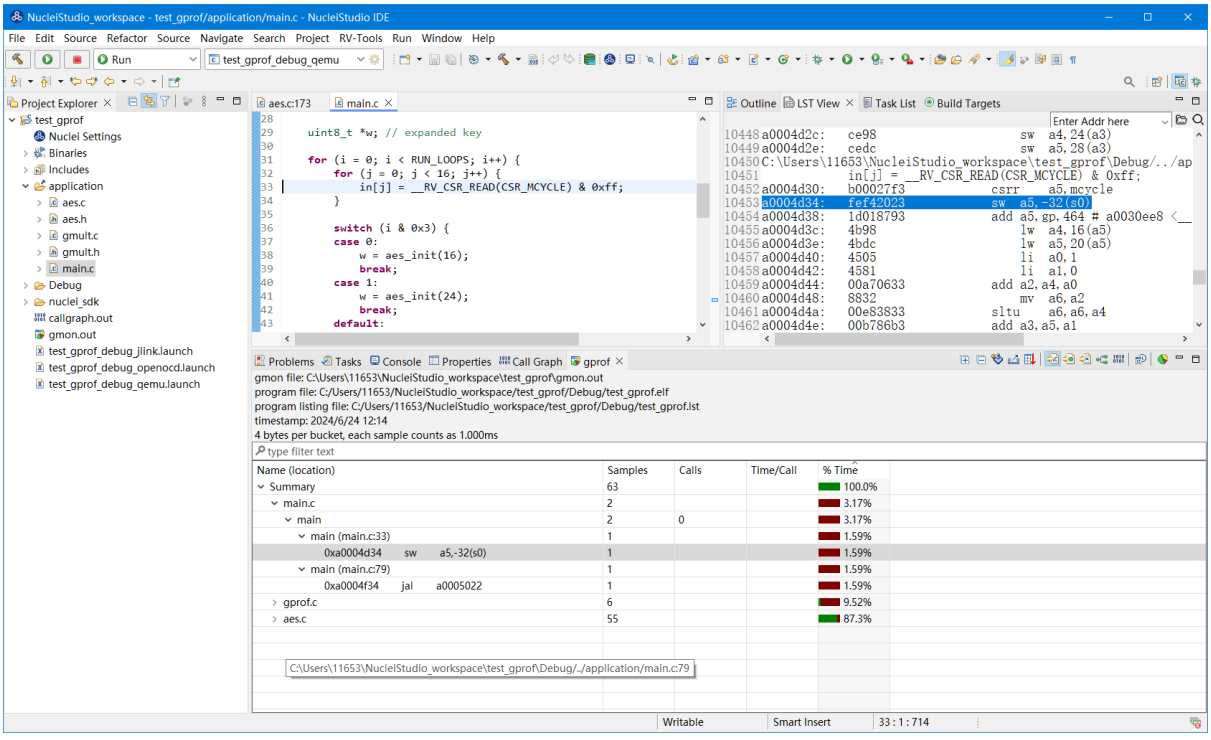


Double-click the `gmon.out` file. A file selection dialog pops up, prompting you to select the elf file and `*.lst` file related to the `gmon.out` file. By default, the `elf file` and `*.lst` file in the corresponding project are filled in automatically based on the current `gmon.out`. Click OK.

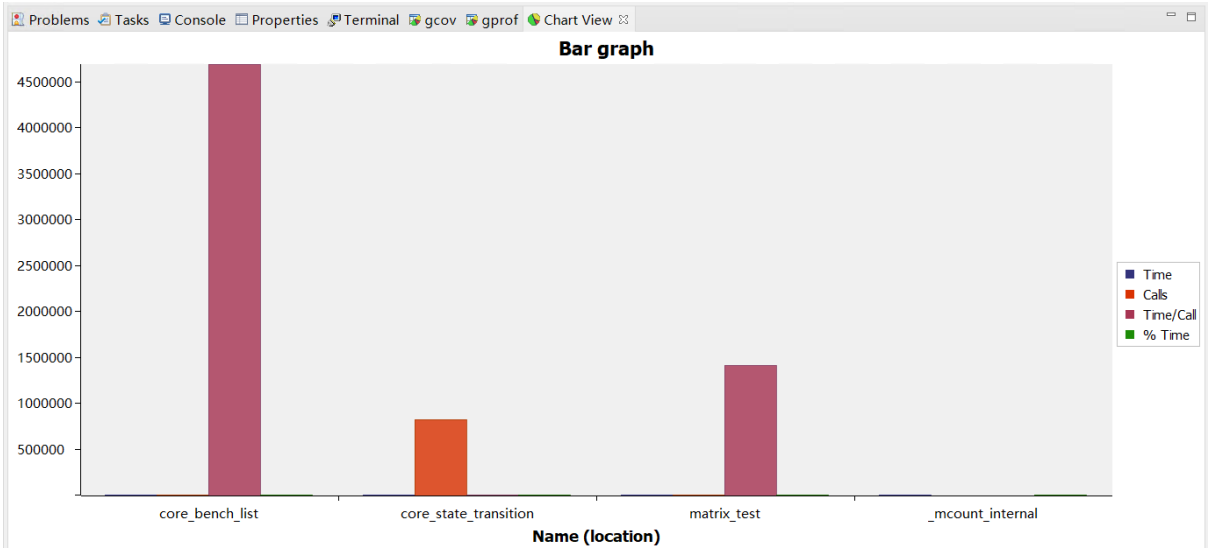


The Gprof tool starts and shows the analysis results for the application, including the call relationships of files and methods.

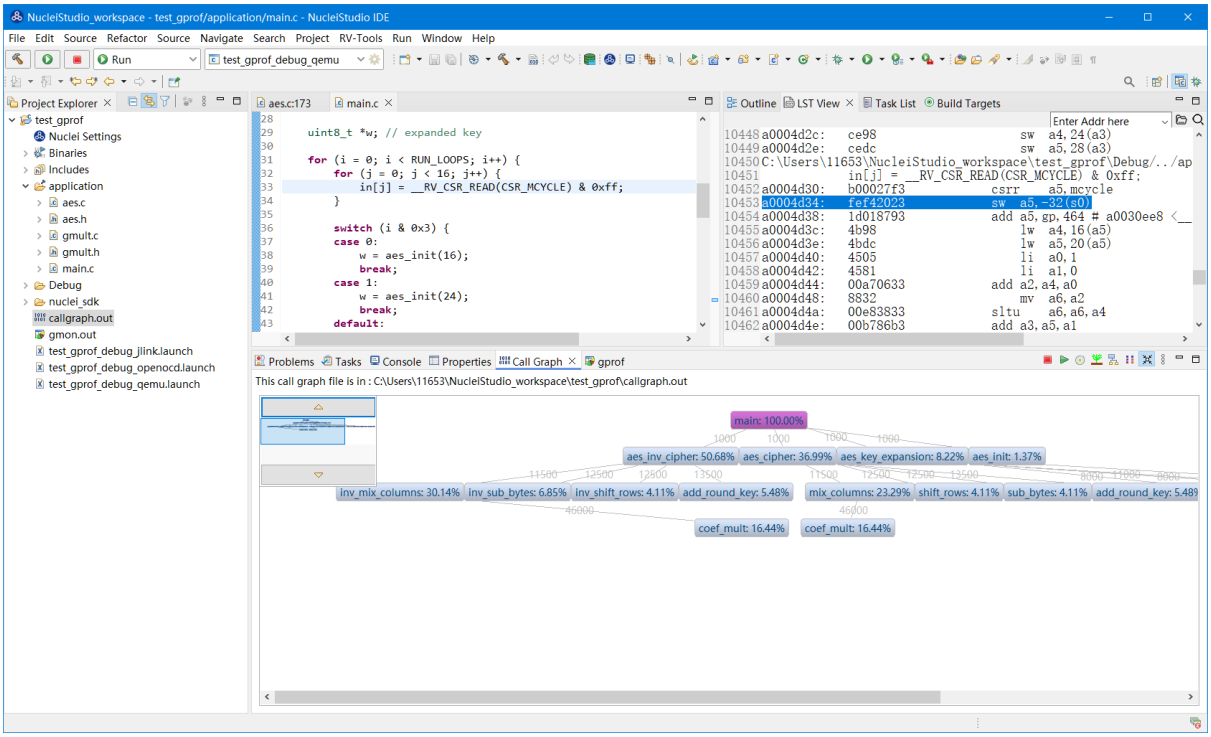
Double-click a row in Gprof, and Nuclei Studio automatically opens the corresponding source file at the corresponding line. It also opens the LST View tool and locates the line by addr, linking Gprof, the source code, and the disassembly code so you can quickly understand the program structure and call relationships.



Likewise, you can view a histogram of the profiling data in Nuclei Studio.



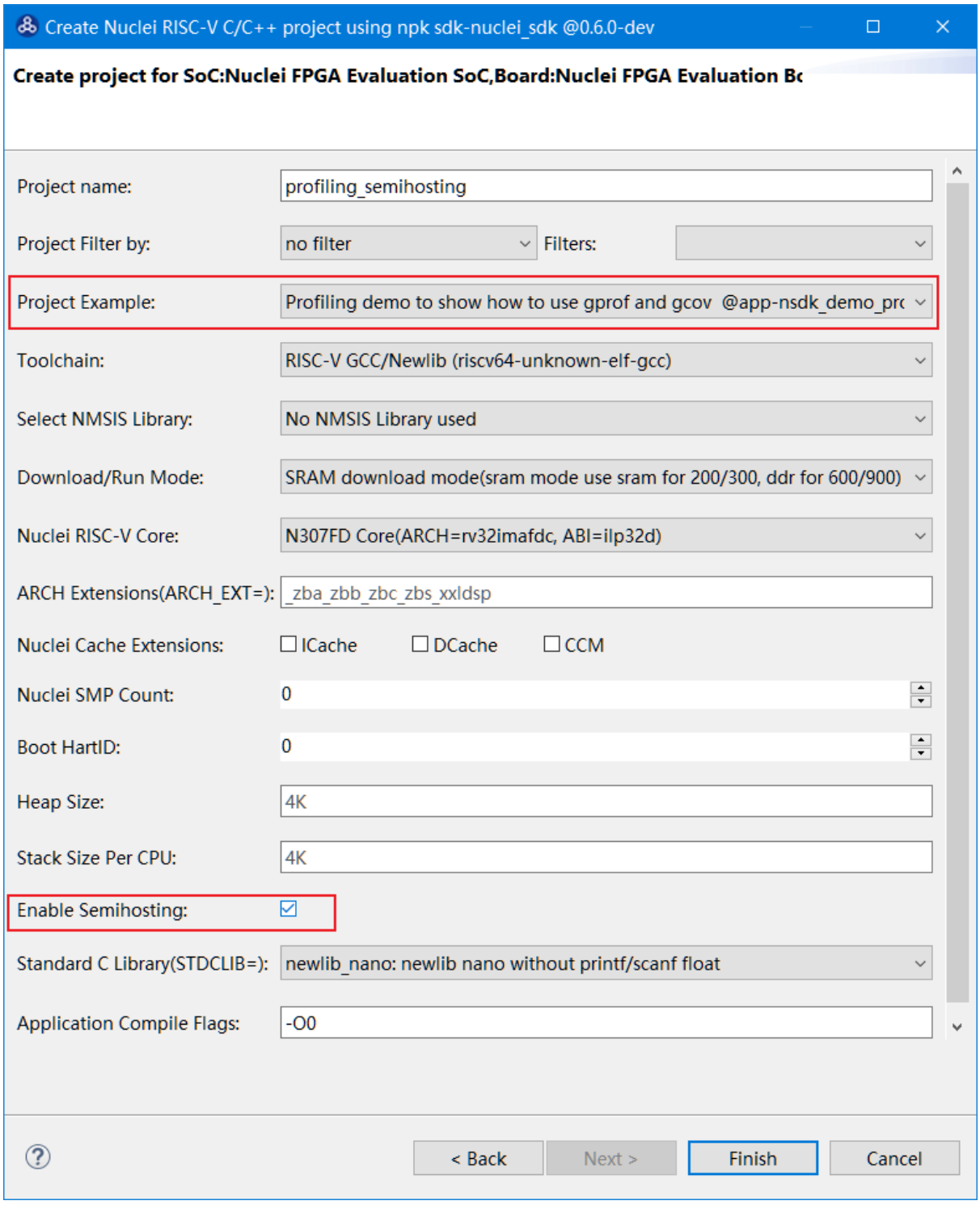
When Gprof is opened, Nuclei Studio parses the program ' s call graph from the gmon.out file and generates a `callgraph.out` file. Double-click the `callgraph.out` file, or click the `Open Call Graph View` button in the Gprof toolbar, to start the Call Graph tool. For details on using Call Graph, see *About Call Graph* (page 231).



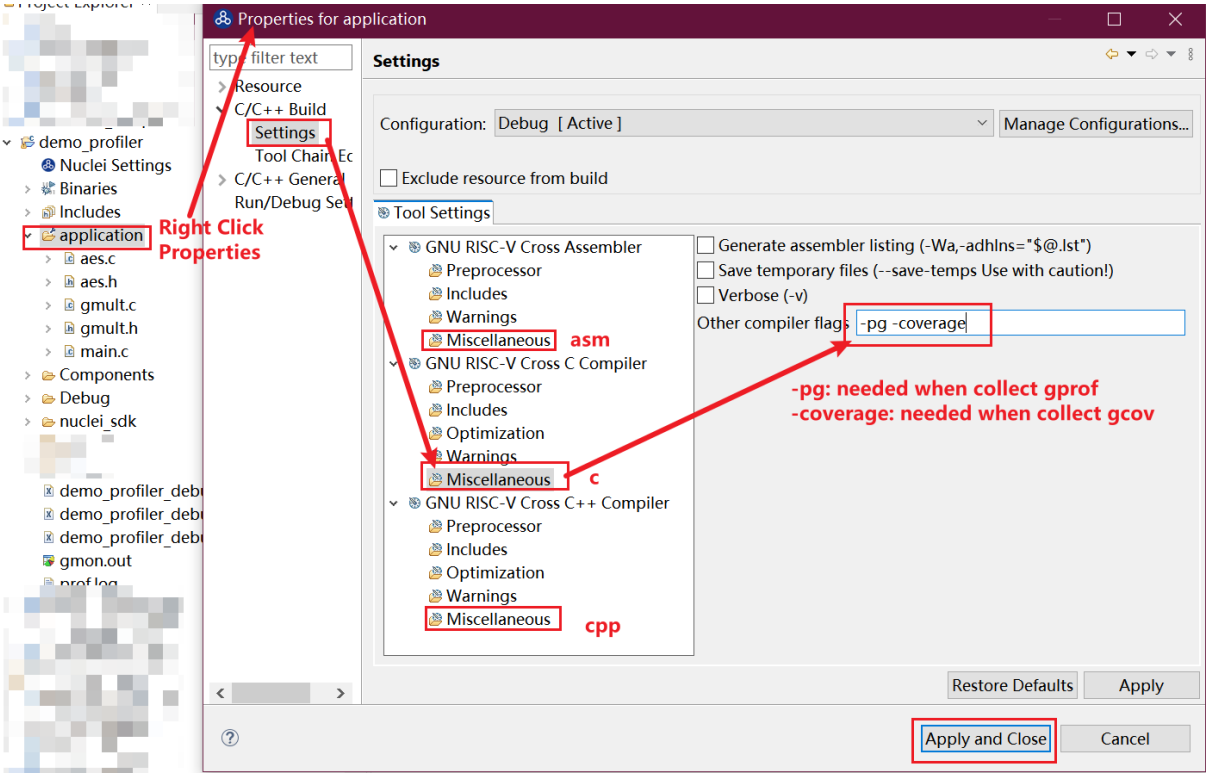
Using via Semihosting

After installing nuclei_sdk 0.6.0 in Nuclei Studio, create a `Profiling demo to show how to use gprof and gcov` test project, this time selecting `Enable Semihosting`. For details about the `Profiling demo to show how to use gprof and gcov` test project, see demo_profiling[19].

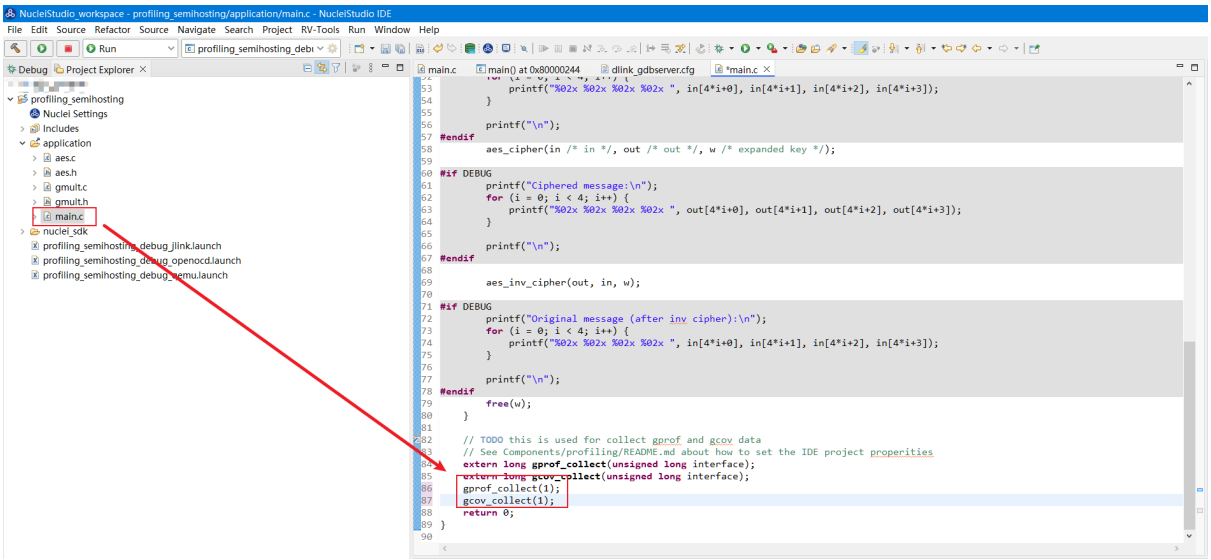
[19] https://doc.nucleisys.com/nuclei_sdk/design/app.html#demo-profiling



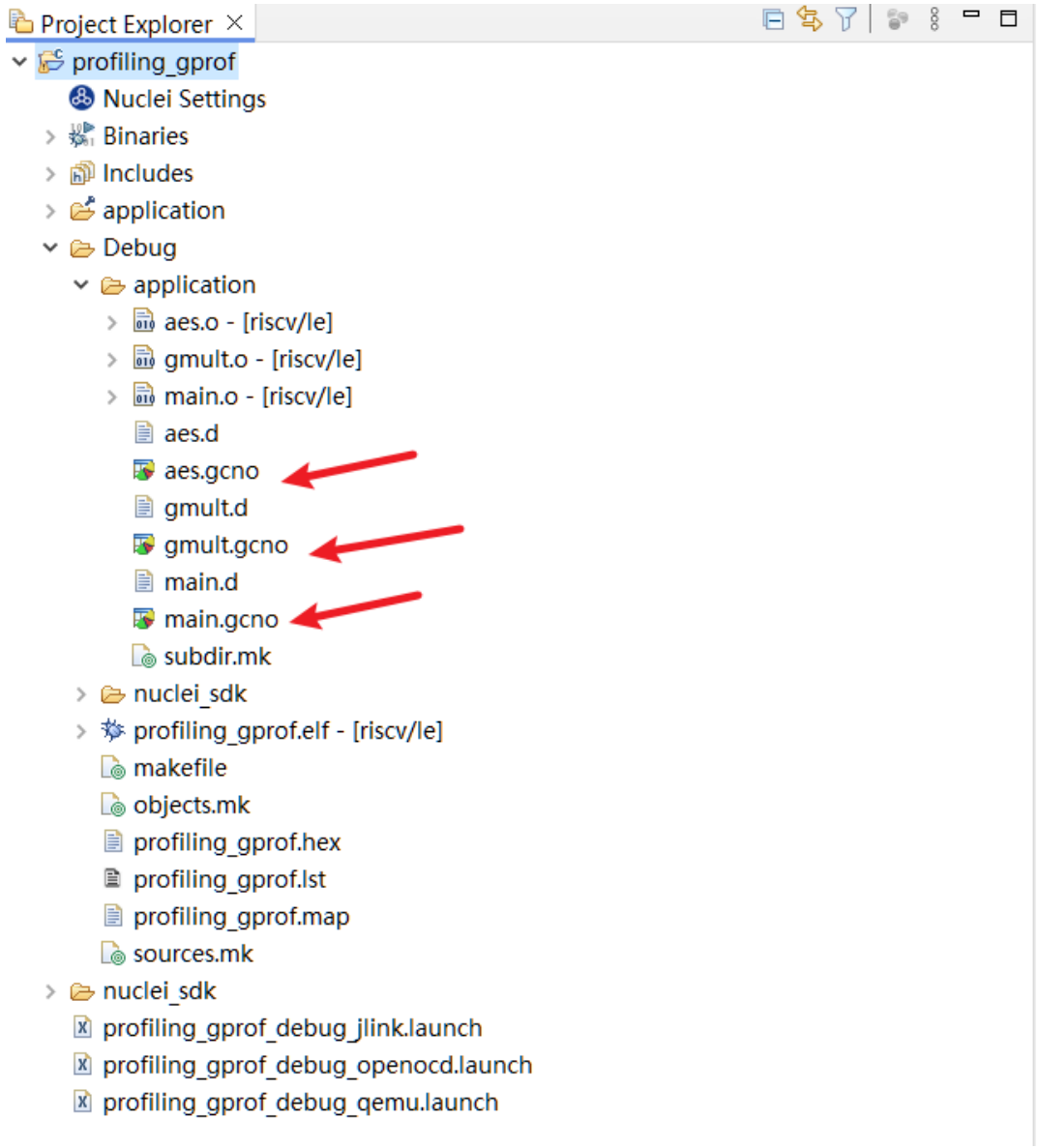
After the project is created, set the `-pg --coverage` compile option for the files or folders you want to analyze, then build the project.



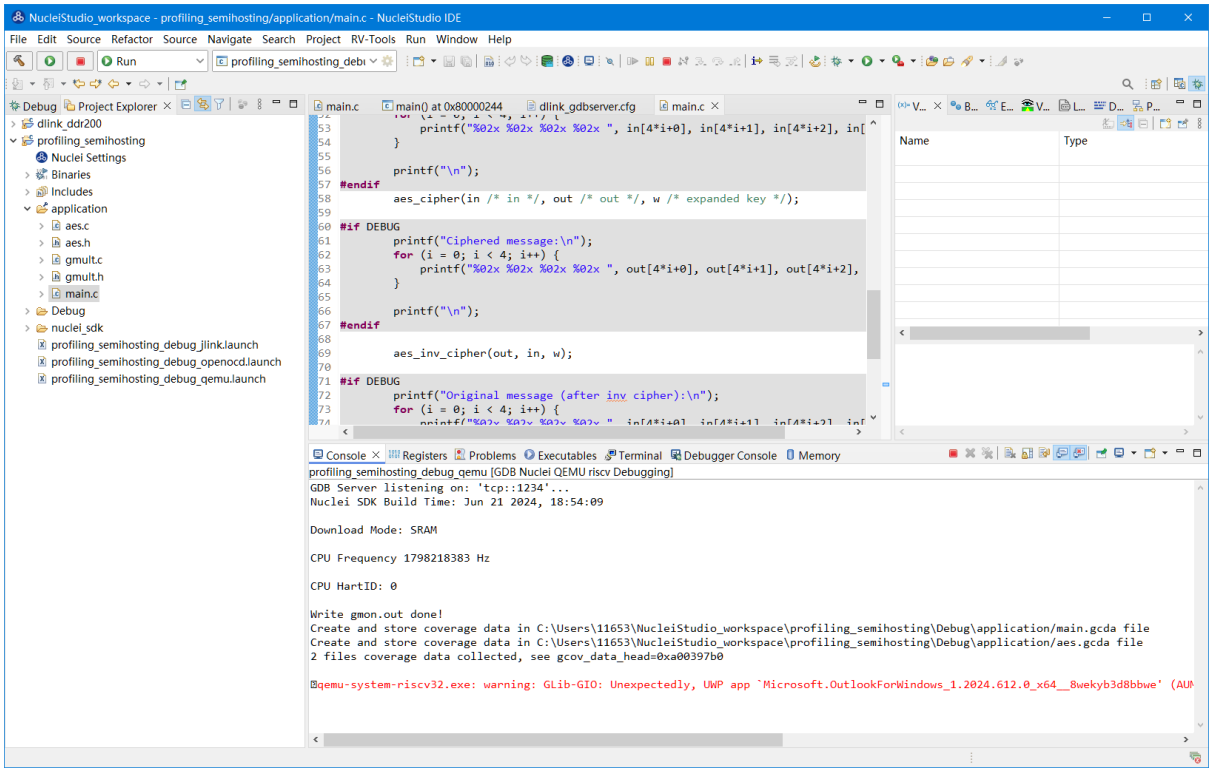
Also change `gprof_collect(2);` to `gprof_collect(1);` and `gcov_collect(2);` to `gcov_collect(1);` in the program (at the end of the main function in the test project). During execution, the results are then output as files through Semihosting.



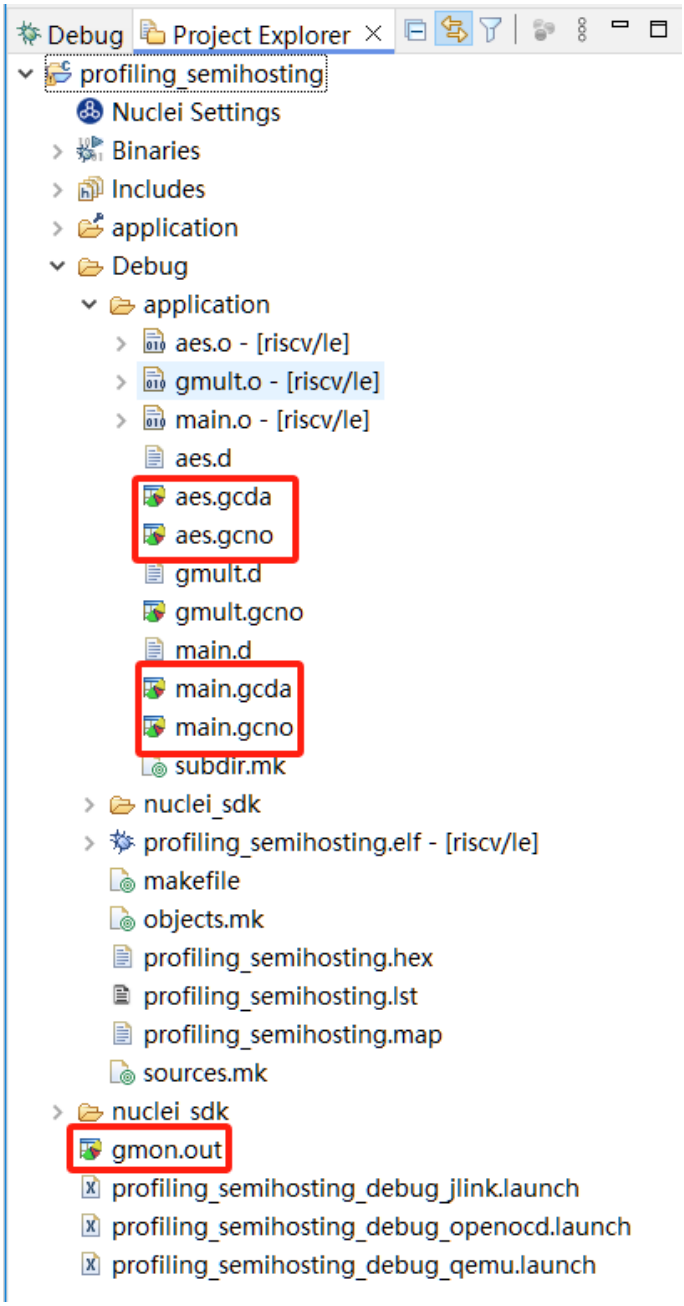
Build the project. In the Debug directory of the successfully built project, you can see that several `.gcno` files have been generated.



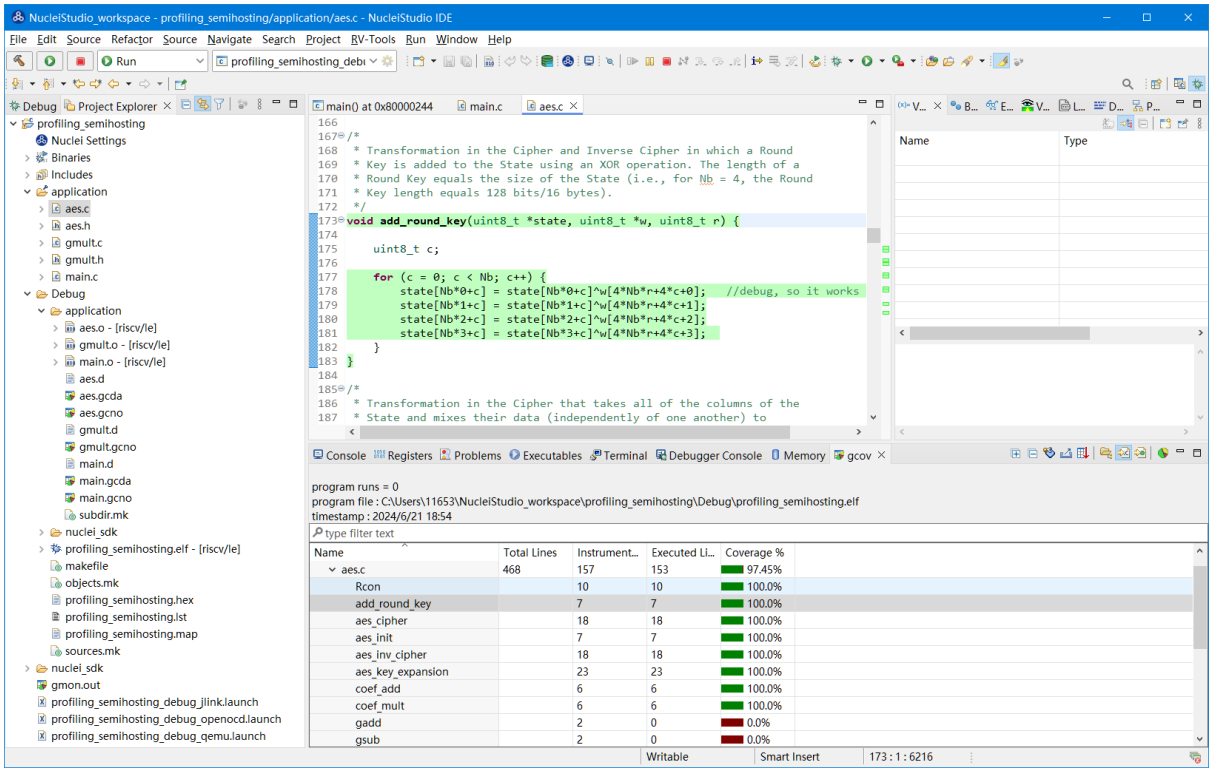
After the project is built, run it under QEMU or debug it on a real development board.



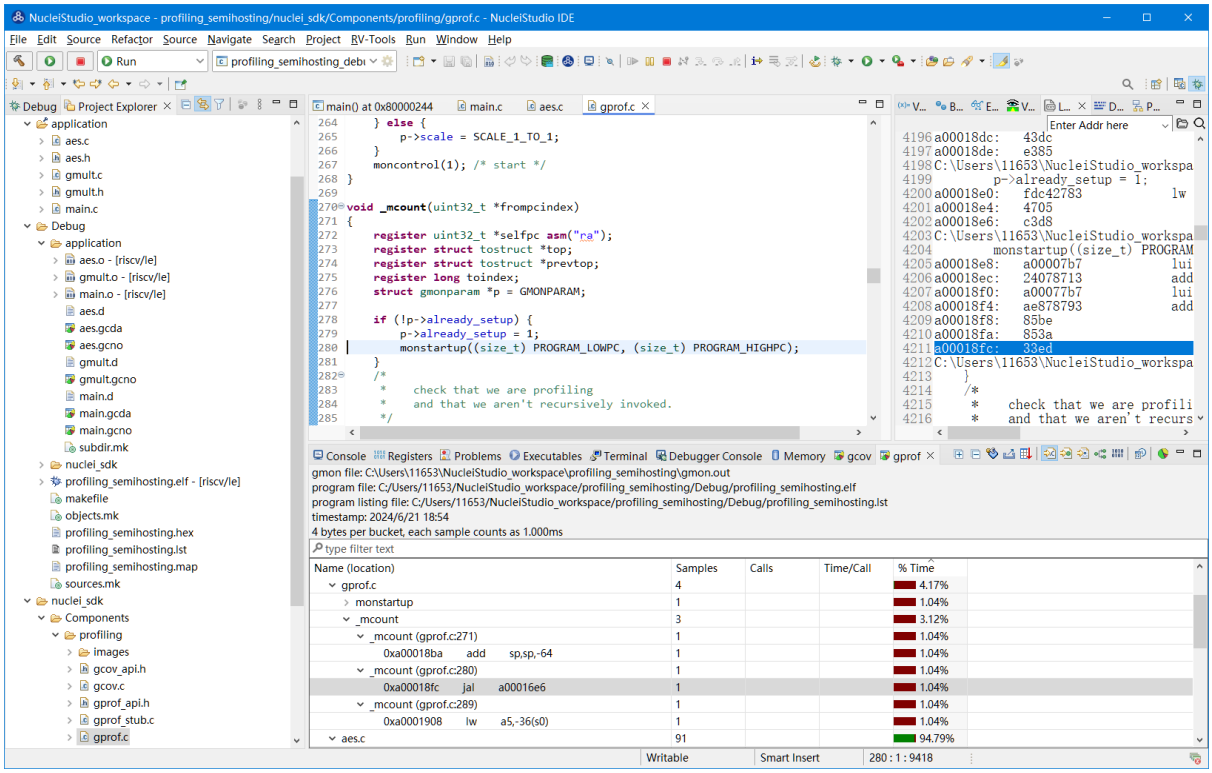
This example runs the program under QEMU. After the program finishes running, refresh the project; several new files appear under the project: `*.gcda` files and `*.out` files. View the results as described above.



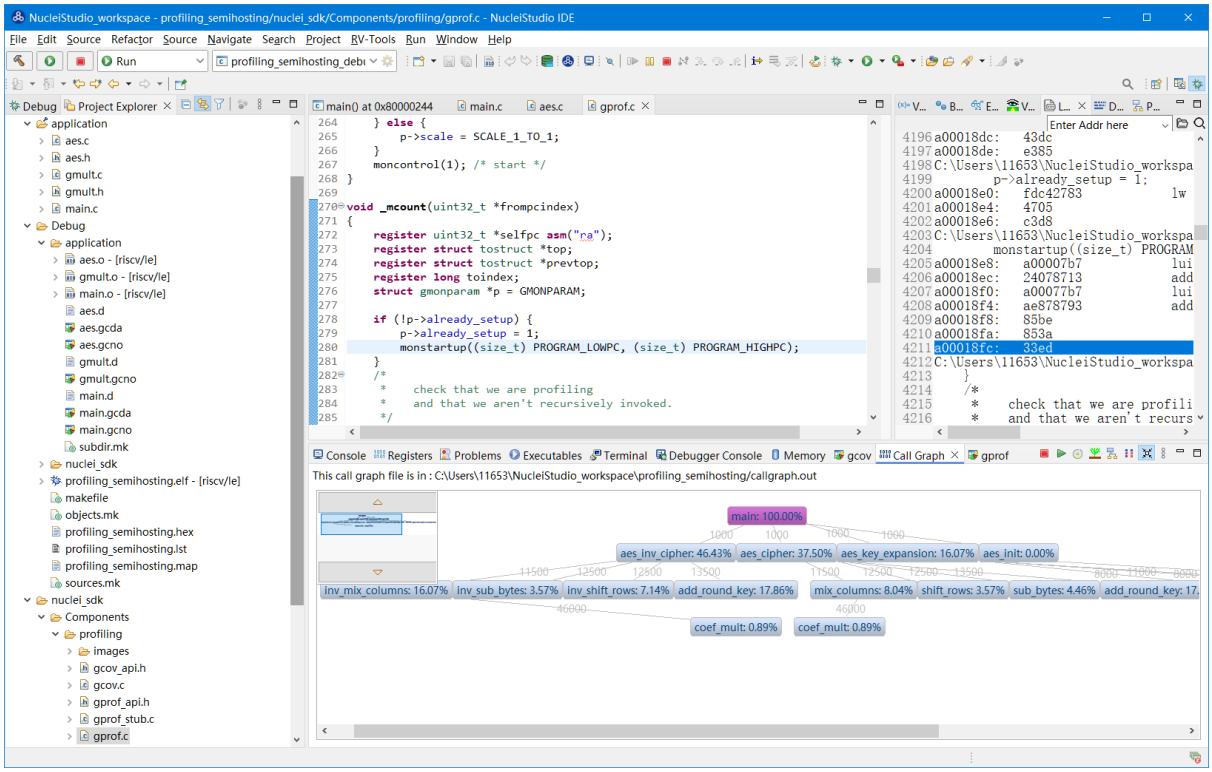
View the Code Coverage information of the application with the gcov tool in Nuclei Studio.



View the Profiling information of the application with the gprof tool in Nuclei Studio.



View the call relationship information with Call Graph in Nuclei Studio.



2.10.5 Trace

Trace is a powerful debugging technology that helps developers track and record key information during program execution, so they can effectively diagnose problems, optimize performance, and improve system stability.

Nuclei Studio integrates a Trace tool. Combined with the corresponding hardware and Nuclei OpenOCD, it lets you view Trace logs while debugging a project and troubleshoot problems together with the source code.

Note: The **Nuclei System Technology WeChat Channel** contains a video on **How to Use the Trace Feature in Nuclei Studio**. Search for the **Nuclei System Technology WeChat Channel** in WeChat to view it.

Note: For the Nuclei ETrace commands of OpenOCD, see the openocd.pdf manual under OpenOCD.

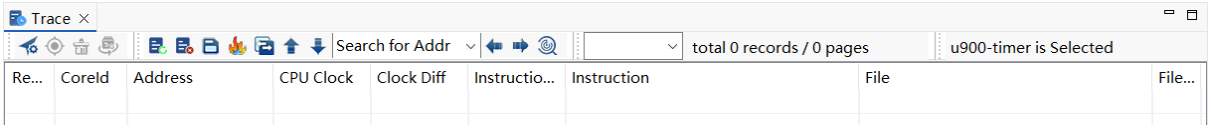
Note: Nuclei Studio **2026.10** adds the **Trace bundle (Trace 2.0)** transport type, and adds the ability to **automatically read parameters from the CPU and fill them in**: when Trace Setting is opened, the plug-in automatically detects the Trace type; after acquisition (clicking Start Trace), the relevant Decode parameters (TS Size / Datatrace / Context Size / Decode Mode, etc.) are automatically read from the registers and filled in, usually without manual modification. See the relevant descriptions at the beginning of Trace Configuration and Trace Decoder Configuration below.

If you have problems during use, you can check the relevant content at https://github.com/Nuclei-Software/nuclei-studio, or submit an issue to us.

Trace UI Overview

Trace View

In Nuclei Studio, open the Show View dialog via `Window->Show View->Other`, find the RV Trace->Trace menu item, and click it to open the Trace view.



The Trace view consists of two parts: the Trace toolbar on top and the Trace record table below. The Trace toolbar provides the following items:

- **Trace setting**

Opens the trace configuration, where you configure Trace ATB2AXI Config Addr, Trace Buffer Base Addr, Trace Buffer Size in Bytes, and Trace Wrap.

- **Start trace/stop trace**

Starts or stops trace recording.

- **Trace clear**

Clears all trace settings on the hardware.

- **Dump trace file**

Dumps the trace file from the hardware.

- **Reload trace file**

Reloads the contents of the trace record table locally.

- **Clear viewer**

Clears the contents of the trace record table, as well as Trace Decode related configuration, such as the relationship between HartID and Thread.

- **Save trace log**

Saves the trace record table as a CSV file.

- **Toggle instruction stepping**

Opens the source code and disassembly code corresponding to the selected record.

- **Decode trace into flame**

Parses flame graph information from the Trace records.

- **step into previous line**

When a record is selected, jumps to the previous record and locates the corresponding source code and disassembly code.

- **step into next line**

When a record is selected, jumps to the next record and locates the corresponding source code and assembly code.

- **Search for Addr**

A search box for finding the trace record row with a given Addr.

- **search backward**

Views the previous result when there are multiple search results.

- **search forward**

Views the next result when there are multiple search results.

- **search exceptions or interruptions**

Views the next result when there are multiple search results.

- **Page**

Pagination controls. When there are many trace records, they are displayed across multiple pages for easier viewing.

The Trace record table displays the records generated after Nuclei Studio decodes the dumped trace file. When you click a record, Nuclei Studio automatically locates the corresponding source code and disassembly code.

- **Record:** record ID
- **CoreId:** Core ID, mainly used to distinguish between different Cores in multi-core scenarios
- **Addr:** instruction address
- **CPU Clock:** clock cycle count
- **Clock Diff:** clock cycle difference
- **Instruction Code:** instruction code in hexadecimal
- **Instruction:** instruction code
- **File:** the file where the source code corresponding to the instruction code is located
- **File Line:** the line number in the file where the source code corresponding to the instruction code is located

Trace Configuration

Click trace setting on the Trace toolbar, and the Set Trace Configuration dialog pops up. The **Trace Mode** radio box at the top of the dialog is used to select the Trace type; the **Settings** area below holds checkboxes common to both modes; the remaining configuration items differ according to the selected mode and are introduced respectively below.

Note: In Nuclei Studio 2026.10 and later, Trace supports two transport types, **Bundle (Trace 2.0)** and the traditional **ATB2AXI (Trace 1.0)**, and adds the ability to **automatically read parameters from the CPU and fill them in**. When the Trace Setting dialog is opened and the debug session is connected to the target (the target is in the halt state), the plug-in reads registers such as `mcfg_info` / `etrace_info` through GDB to automatically detect the Trace type supported by the current CPU: the **Trace Mode** radio box is selected automatically based on the detection result — `mcfg_info` bit 20 = 1 means ATB2AXI is supported; otherwise it is judged based on `etrace_info` bit[8:1]: 0x20 → Bundle, 0x00 → ATB2AXI. If no development board is connected, the software default value is used.

(Here "Bundle" refers to the Trace 2.0 transport type; the usage of packaging one acquisition result into a single bundle file for offline playback is described in the Trace bundle entry of the 2026.10 release notes.)

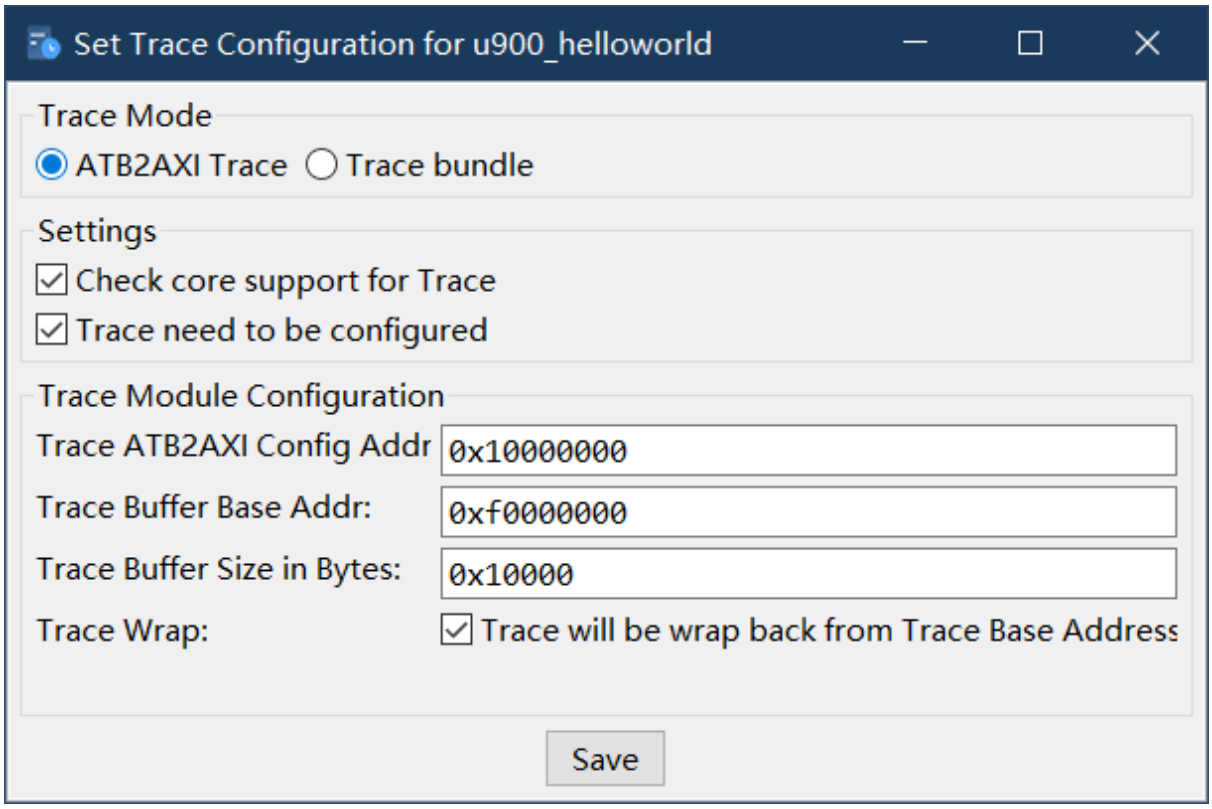
For Bundle mode, the Decode parameters required for acquisition, such as TS Size / Datatrace / Context Size / Decode Mode, are automatically read from the registers and filled in by the plug-in after clicking **Start Trace** to send the configuration command (see Trace Decoder Configuration below for details).

Common settings (Settings, identical for both modes)

- **Check core support for Trace:** if checked, whether the CPU supports the Trace feature is checked before sending Trace commands; if not checked, whether the CPU supports Trace is ignored when sending Trace commands.
- **Trace need to be configured:** check this if the Trace module needs to be configured. If it has already been configured elsewhere, do not check it. For example, in the multicore SMP/AMP case, there is only one Trace module on the SoC; assuming one core has already checked and configured it, the other cores must not check it, or if the configuration was done in C code or somewhere else, do not check it either.

ATB2AXI Trace (Trace 1.0)

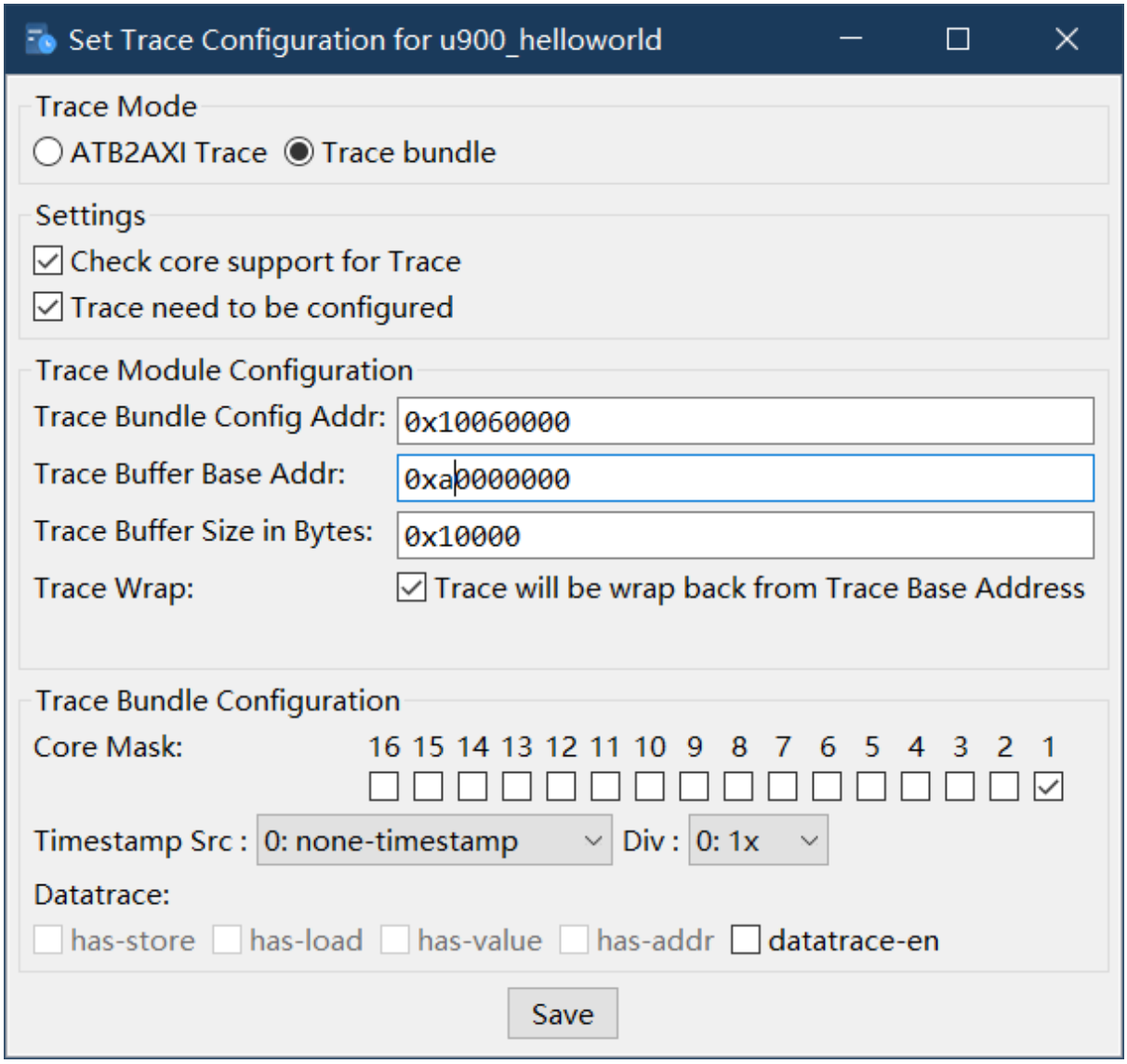
When Trace Mode selects ATB2AXI Trace, you only need to configure the Trace Module Configuration area:



- **Trace ATB2AXI Config Addr:** the base address of the ATB2AXI module controller, e.g. 0x10000000.
- **Trace Buffer Base Addr:** the start address for storing trace records. For example, for a certain SoC: in flashxip mode, ilm (0x1c000000) is used as the cache buffer; in sramxip mode, dlm (0x08010000) is used as the cache buffer.
- **Trace Buffer Size in Bytes:** the size of the Buffer storing trace records, in bytes, e.g. 0x10000.
- **Trace Wrap:** when checked, it shows "Trace will be wrap back from Trace Base Address", meaning automatic overwrite is allowed; when the Buffer is full, records are overwritten again from the beginning.

Trace bundle（Trace 2.0）

When Trace Mode selects Trace bundle, the Trace Module Configuration area uses the Bundle base address instead, and an additional Trace Bundle Configuration area appears:



- **Trace Bundle Config Addr:** the base address of the configuration registers in Bundle mode, e.g. 0x10060000.
- **Trace Buffer Base Addr:** the start address for storing trace records; the meaning is the same as in ATB2AXI mode.
- **Trace Buffer Size in Bytes:** the size of the Buffer storing trace records, in bytes; the meaning is the same as in ATB2AXI mode.
- **Trace Wrap:** allows automatic overwrite; when the Buffer is full, records are overwritten again from the beginning; the meaning is the same as in ATB2AXI mode.
- **Core Mask:** (Trace Bundle Configuration area) core mask; bits 16~1 correspond to the respective cores. Check the cores for which Trace needs to be collected (for single core, check bit 1).
- **Timestamp Src:** timestamp source, e.g. 0: none-timestamp means no timestamp; **Div:** timestamp division, e.g. 0: 1x.
- **Datatrace:** data trace feature configuration, including has-store, has-load, has-value, has-addr, and the datatrace-en enable checkbox.

Trace Decoder Configuration

After clicking Dump trace file, the Set current debug hart configuration dialog pops up, used to configure the trace decoder parameters. The Local trace related files area in the upper part of the dialog is common to both modes, and the Hart decoding configuration area in the lower part differs by mode; they are introduced respectively below.

Note: Starting from Nuclei Studio 2026.10, parameters such as **TS Size**, **Datatrace**, **Context Size**, and **Decode Mode** in Hart decoding configuration support **automatic reading from the CPU and filling in** (that is, the "automatically read from the CPU and fill in parameters" feature):

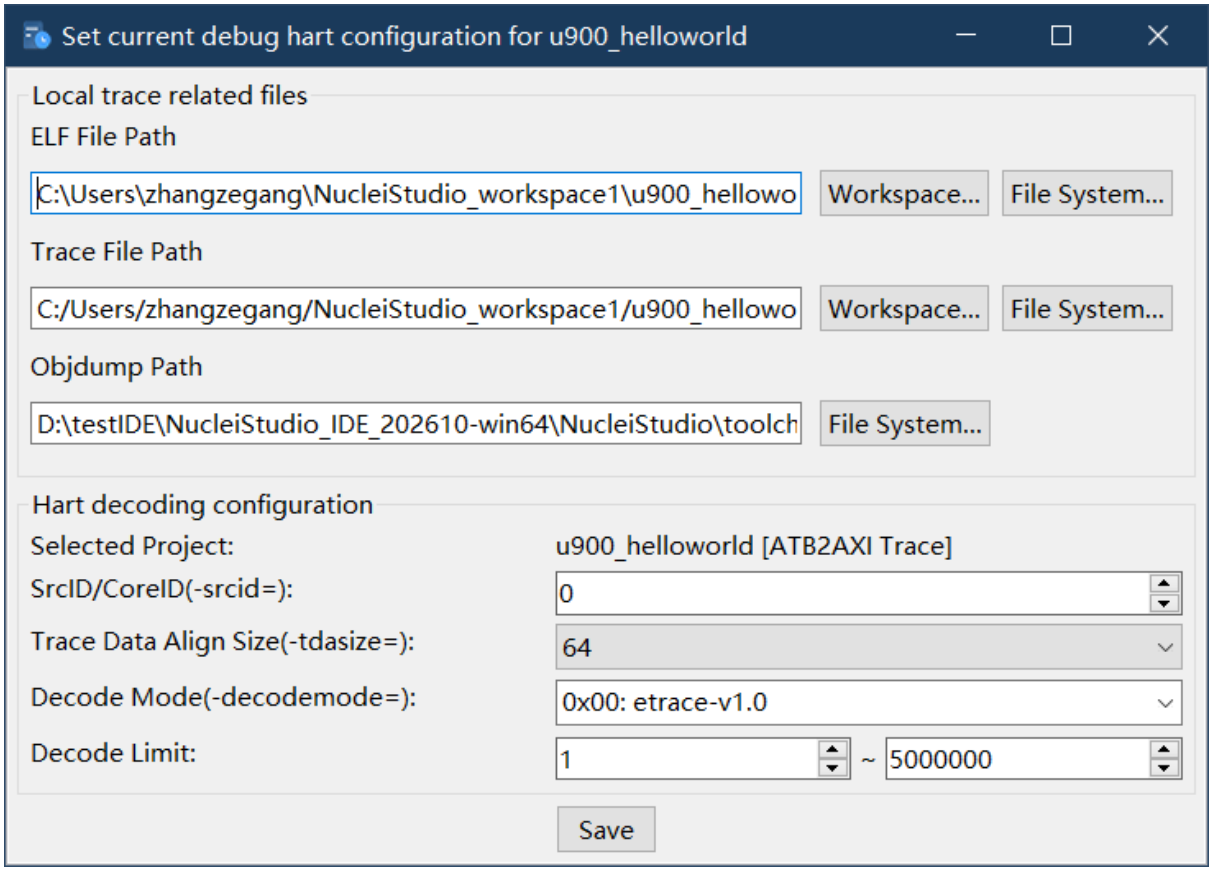
- **ATB2AXI (Trace 1.0)**: they are read from the ATB2AXI registers and filled in during the Trace Setting detection phase; among them Context Size is fixed to 0 and Decode Mode is fixed to `0x00: etrace-v1.0`.
- **Trace bundle (Trace 2.0)**: after clicking **Start Trace** to send the trace config command, the plug-in reads them from the Bundle configuration registers and fills them in; Decode Mode is also selected automatically according to the Bundle version number: v1.0.0 → `0x10`, v1.1.0 / v1.2.0 / v1.3.0 → `0x11`, v1.4.0 and newer → `0x12`.

When no development board is connected, the software default values are used. Manual modification is generally unnecessary; if fine-tuning is needed, you can override it directly in the UI.

Local trace related files (identical for both modes)

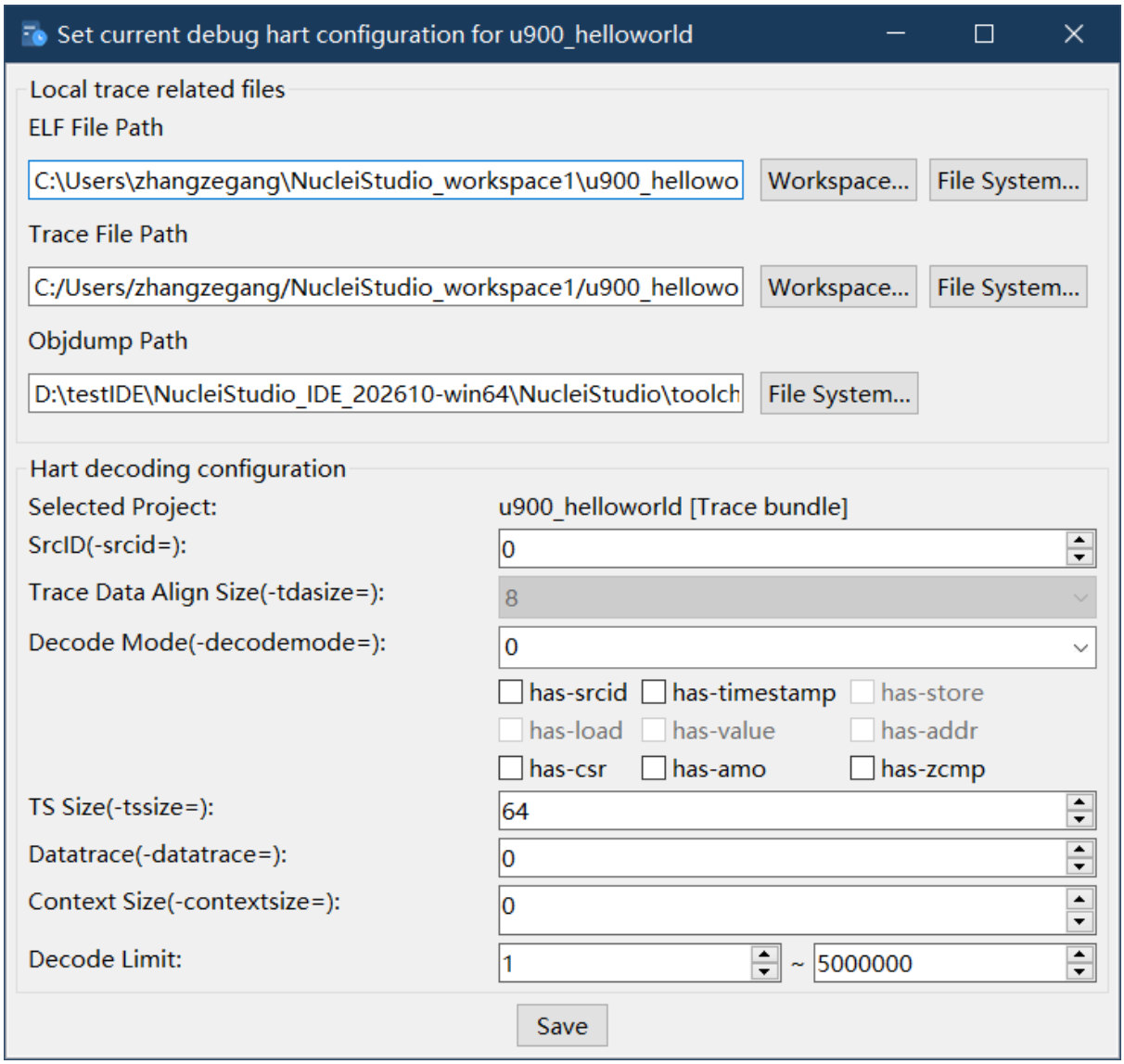
- **ELF File Path:** the path of the elf file executed when the trace is produced.
- **Trace File Path:** the path of the trace file to be parsed.
- **Objdump Path:** the trace decode process needs the objdump tool, so the path of the objdump tool to be used must be specified here.

ATB2AXI Trace（Trace 1.0）



- **Selected Project:** the current project name and its Trace mode, e.g. u900_helloworld [ATB2AXI Trace].
- **SrcID/CoreID(-srcid=):** corresponds to the HartID to which the trace belongs; the default is 0 for a single-core project.
- **Trace Data Align Size(-tdasize=):** trace data alignment size, generally aligned with the trace output bit width of the hardware; the defaults are 8, 32, 64 (64 in the figure).
- **Decode Mode(-decodemode=):** fixed to `0x00： etrace-v1.0` for ATB2AXI (Trace 1.0).
- **Decode Limit:** the limit on the number of trace decode records, ranging from 1 to 5000000, with a default of 5 million; the more records, the higher the requirements on the user ' s PC.

Trace bundle（Trace 2.0）



- **Selected Project:** the current project name and its Trace mode, e.g. u900_helloworld [Trace bundle].
- **SrcID(-srcid=):** corresponds to the HartID to which the trace belongs; the default is 0 for a single-core project.
- **Trace Data Align Size(-tdasize=):** in Bundle mode it is filled in automatically by the plug-in (8 in the figure) and cannot be modified.
- **Decode Mode(-decodemode=):** the default is 0; since 2026.10 it is selected automatically according to the Bundle version number (`TRTRACEBUNDLEVERSION` at Trace Config + 0xE00): v1.0.0 selects `0x10` (tracebundle-v1.0.0), v1.1.0 / v1.2.0 / v1.3.0 select `0x11`, v1.4.0 and newer select `0x12`; it can be overridden manually.

- **Feature checkboxes (has-srcid / has-timestamp / has-store / has-load / has-value / has-addr / has-csr / has-amo / has-zcmp):** used to mark the features contained in the trace data; among them has-store / has-load / has-value / has-addr are determined automatically by the Datatrace configuration in Trace Bundle Configuration during acquisition.
- **TS Size(-tssize=):** timestamp width, from `trTimestampWidth` (bit[7:0]) at Trace Config + 0x4E04; when it is 0, it falls back to bit[29:24] at +0x4040; the default is 64.
- **Datatrace(-datatrace=):** data trace width, from `trIndexWidth` (bit[31:16]) at Trace Config + 0x4E00; the default is 0.
- **Context Size(-contextsize=, the 10th parameter of decode):** context width, from `trContextWidth` (bit[15:8]) at Trace Config + 0x4E04; the default is 0.
- **Decode Limit:** the limit on the number of trace decode records, ranging from 1 to 5000000, with a default of 5 million; the more records, the higher the requirements on the user ' s PC.

Using Trace

Note: Whether you use **ATB2AXI (Trace 1.0)** or **Bundle (Trace 2.0)**, the following acquisition / Dump / Decode flow is the same; the differences only lie in the configuration and decoding parameters:

- In Trace Configuration: select Trace bundle as the Trace Mode, fill in **Trace Bundle Config Addr** (replacing Trace ATB2AXI Config Addr in ATB2AXI mode), and configure Core Mask, Timestamp Src, Datatrace, etc. in the Trace Bundle Configuration area as needed; the remaining configurations such as Buffer and Wrap are the same.
- In Trace Decoder Configuration: the decode parameter UI differs by mode —ATB2AXI mode has SrcID/ CoreID, Trace Data Align Size, and Decode Mode; Bundle mode adds feature checkboxes and parameters such as **TS Size / Datatrace / Context Size**, most of which are filled in automatically by the plug-in (see the per-mode descriptions above for details).

Therefore, the operation steps below, which use a single core as an example, apply to both modes.

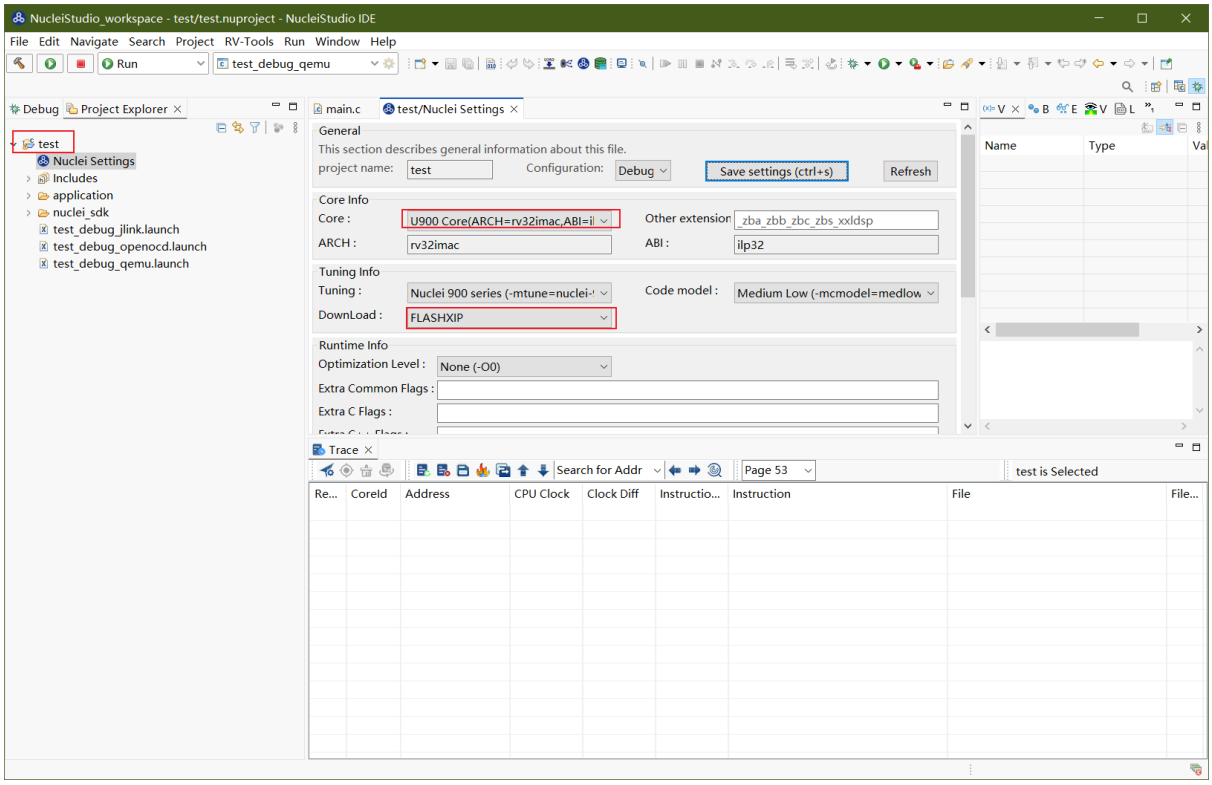
When using the trace feature, you must send the Trace commands to the hardware through Nuclei OpenOCD or Dlink while the project is being debugged. Currently, with OpenOCD, Trace recording can be performed for single-core, multicore SMP, and multicore AMP applications, while Dlink only supports trace recording for single-core applications.

The following example uses OpenOCD to demonstrate the Trace feature.

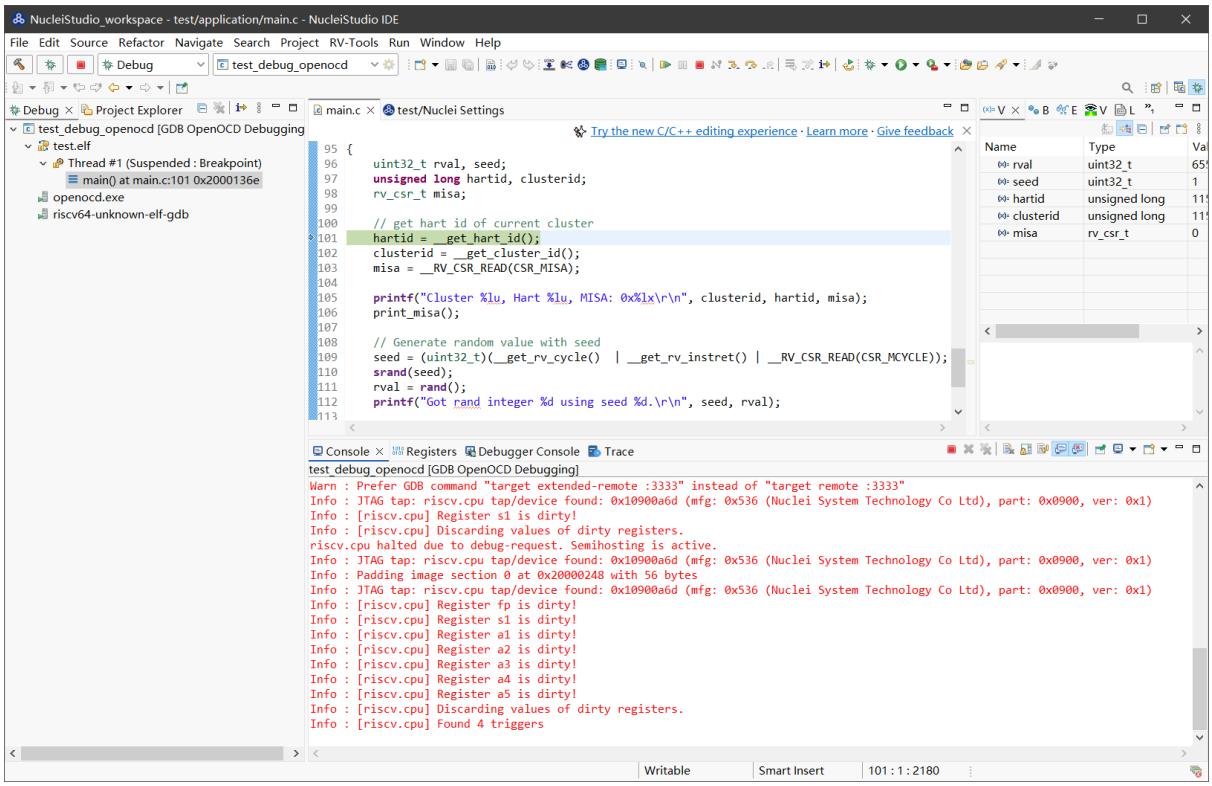
Using Trace in a Single-Core Application

This document does not describe how to obtain the Nuclei-licensed CPU and the related supporting hardware or how to set up the hardware environment. Create the corresponding project and make sure it can run and debug on the hardware. The following example illustrates the process in a test environment we built ourselves.

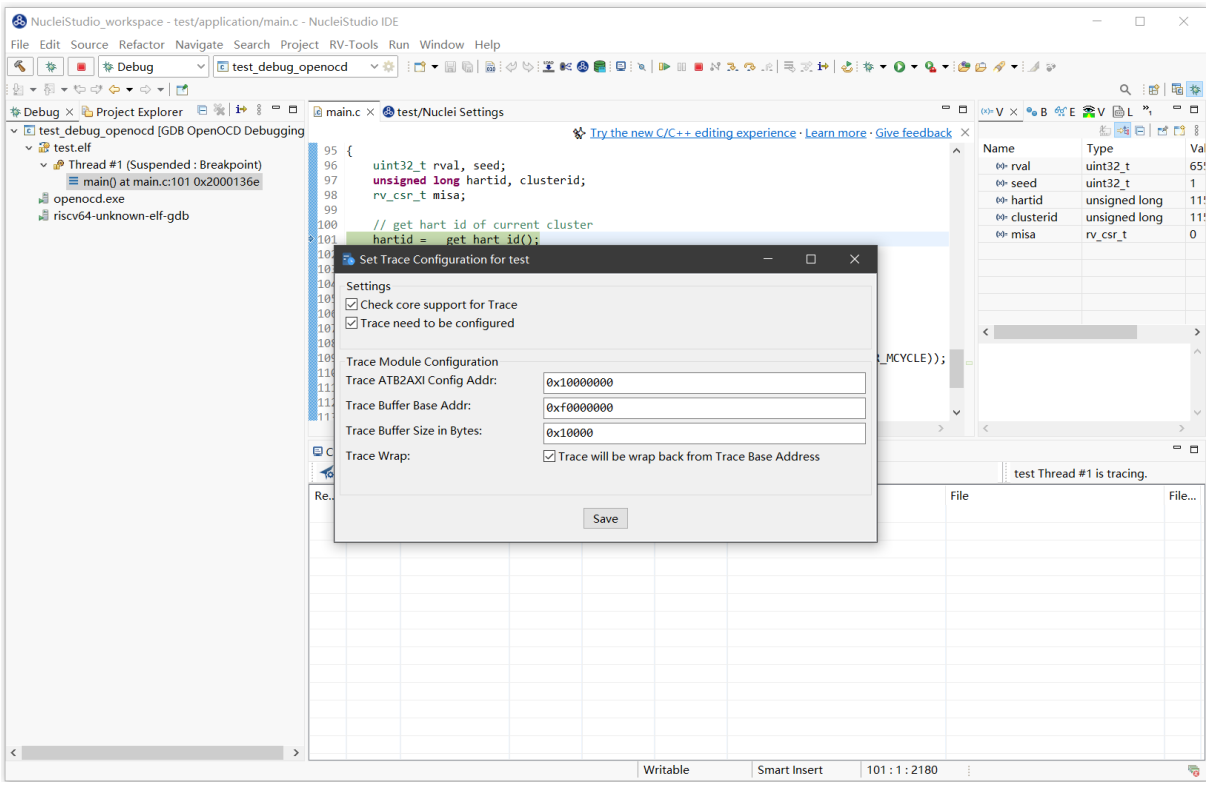
In this example, we create a U900 single-core application named test from the demo timer template project and run it in FLASHXIP mode.



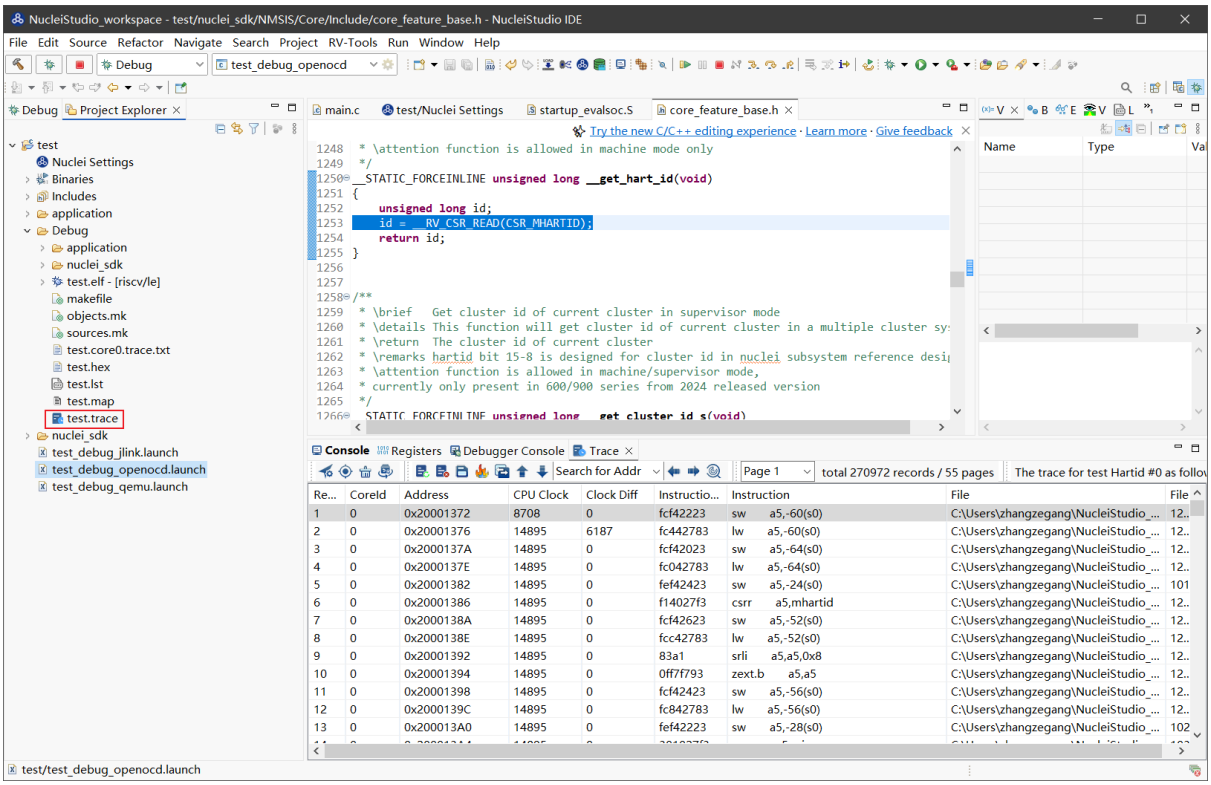
You can record the trace of the entire application run, or only the trace between two Debug breakpoints. After entering Debug mode, open the Trace view.



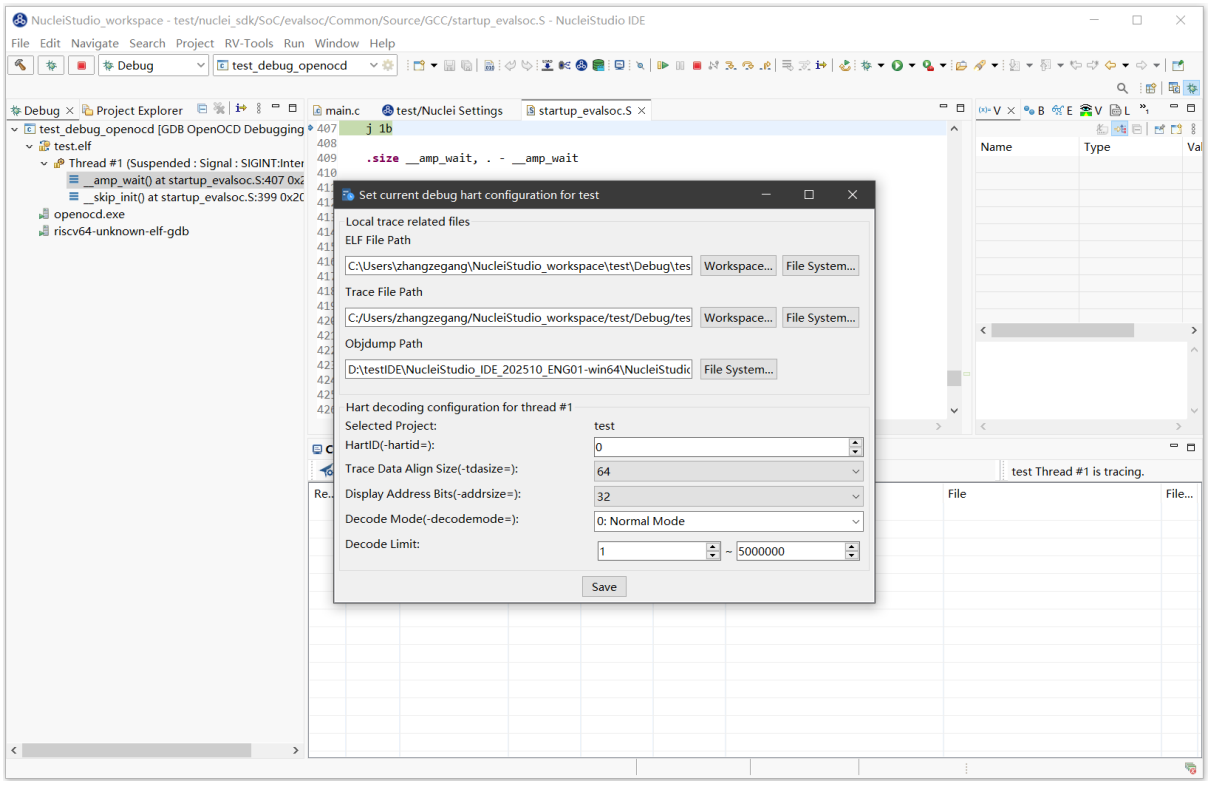
Open the Trace Configuration, configure the trace information, and click Save. If you do not want to save the configuration, close the window.



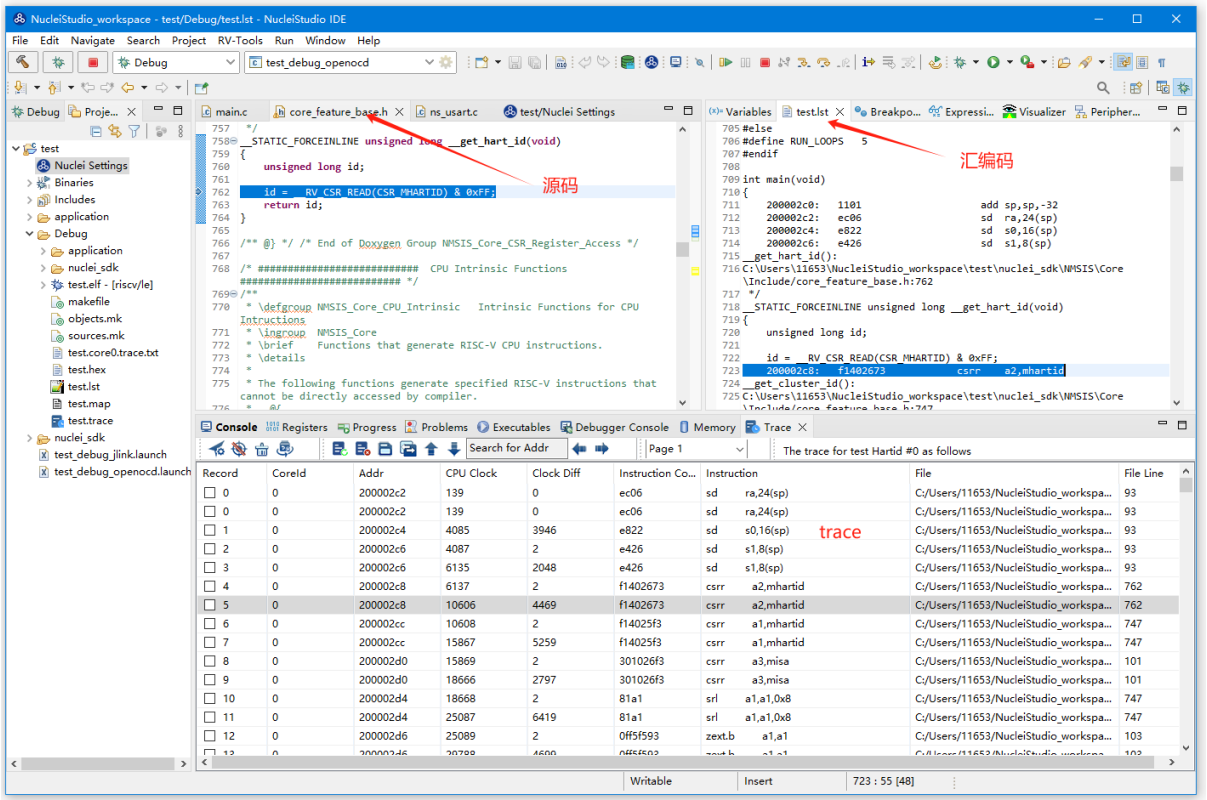
After Trace is configured, set two breakpoints, one as the Trace start point and one as the Trace end point. When the program stops at the start-point breakpoint, click the `start trace` button and continue debugging (such as stepping or running). When the program stops at the end-point breakpoint, click the `stop trace` button to end Trace. This is only one way to use Start/Stop Trace; adapt it to your needs. When the trace ends (in multi-core scenarios, make sure Trace has ended on every CPU), click the `Dump trace file` button to download the trace file from the hardware to the local machine. By default, the downloaded trace file is stored in the debug directory under the project directory as `<projectname>.trace`.



After the trace file is downloaded, Nuclei Studio pops up a `Set current debug hart configuration` dialog.

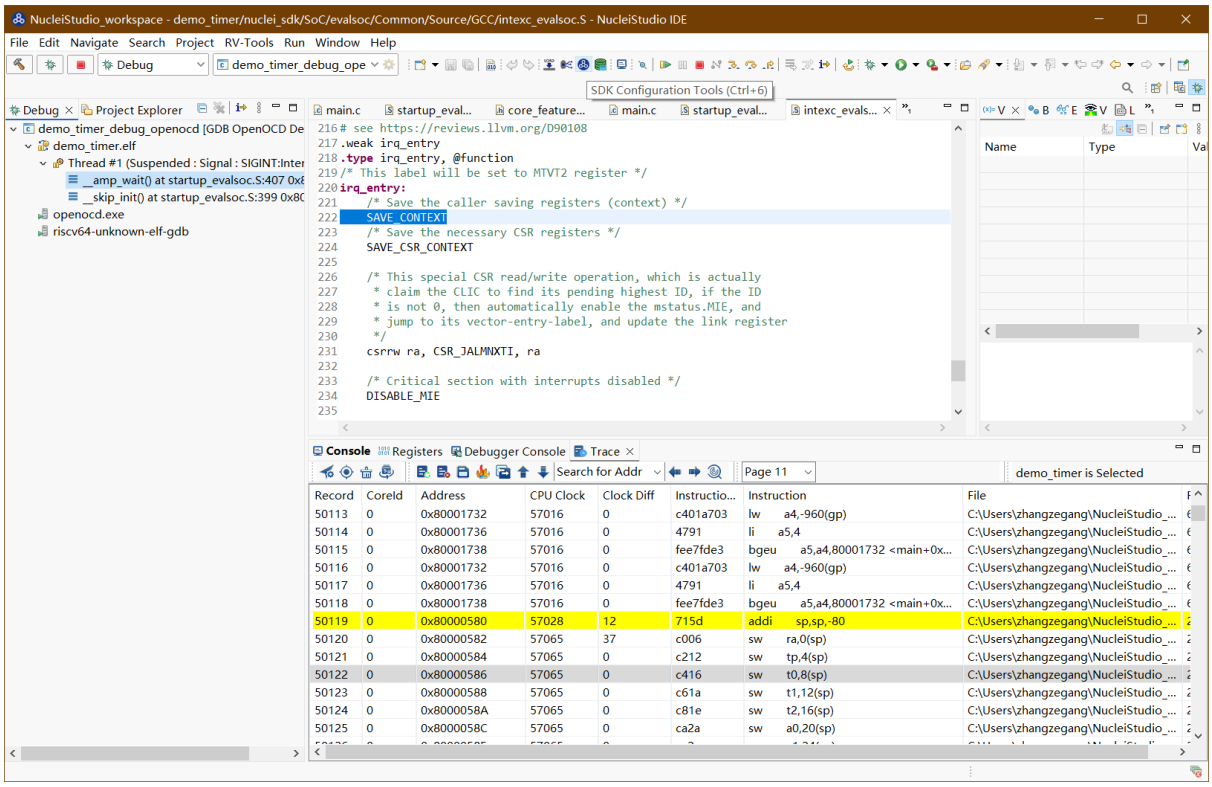


Fill in the correct information in the dialog (the HartID must be the hartid of the corresponding Thread) and confirm. Nuclei Studio then parses the trace file and generates the trace record table. Select any record in the table, and Nuclei Studio automatically finds the source code and disassembly code and locates the corresponding line (because the disassembly code and the source code open in the same view, move the disassembly code to another view yourself).

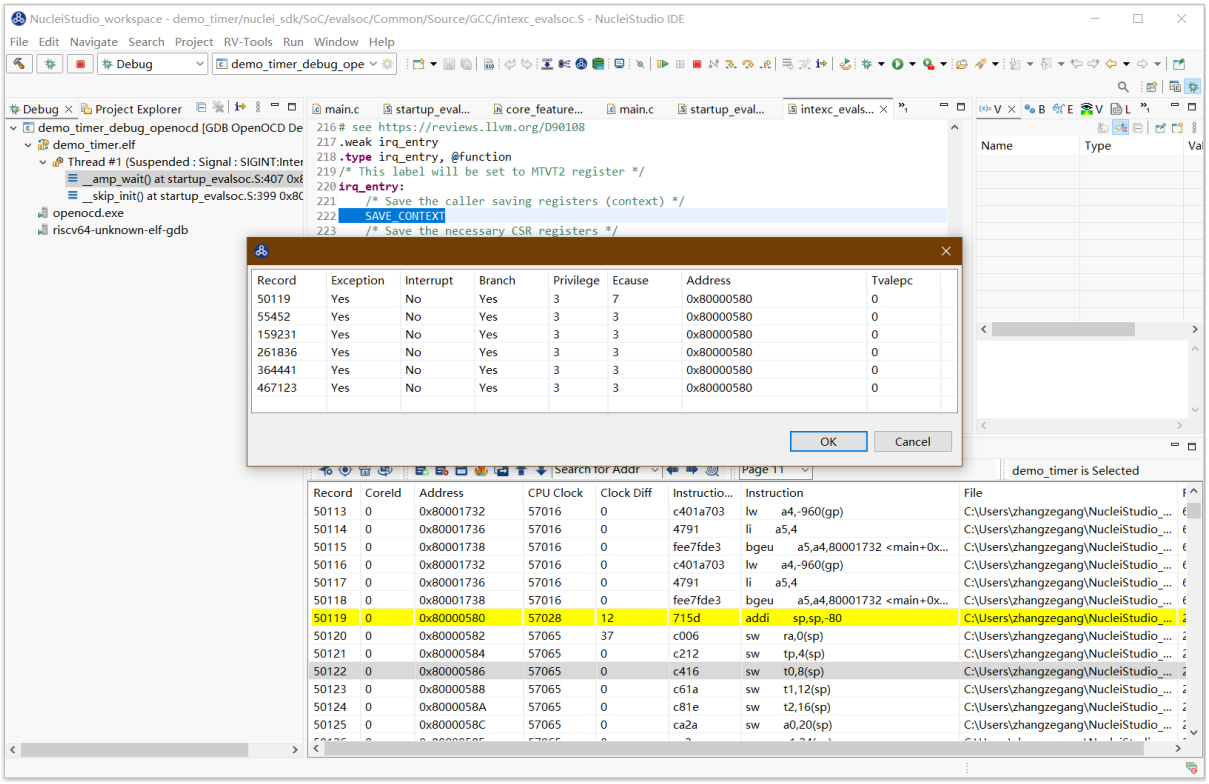


Note: Querying interrupts and exceptions is a new feature in Nuclei Studio 2025.10.

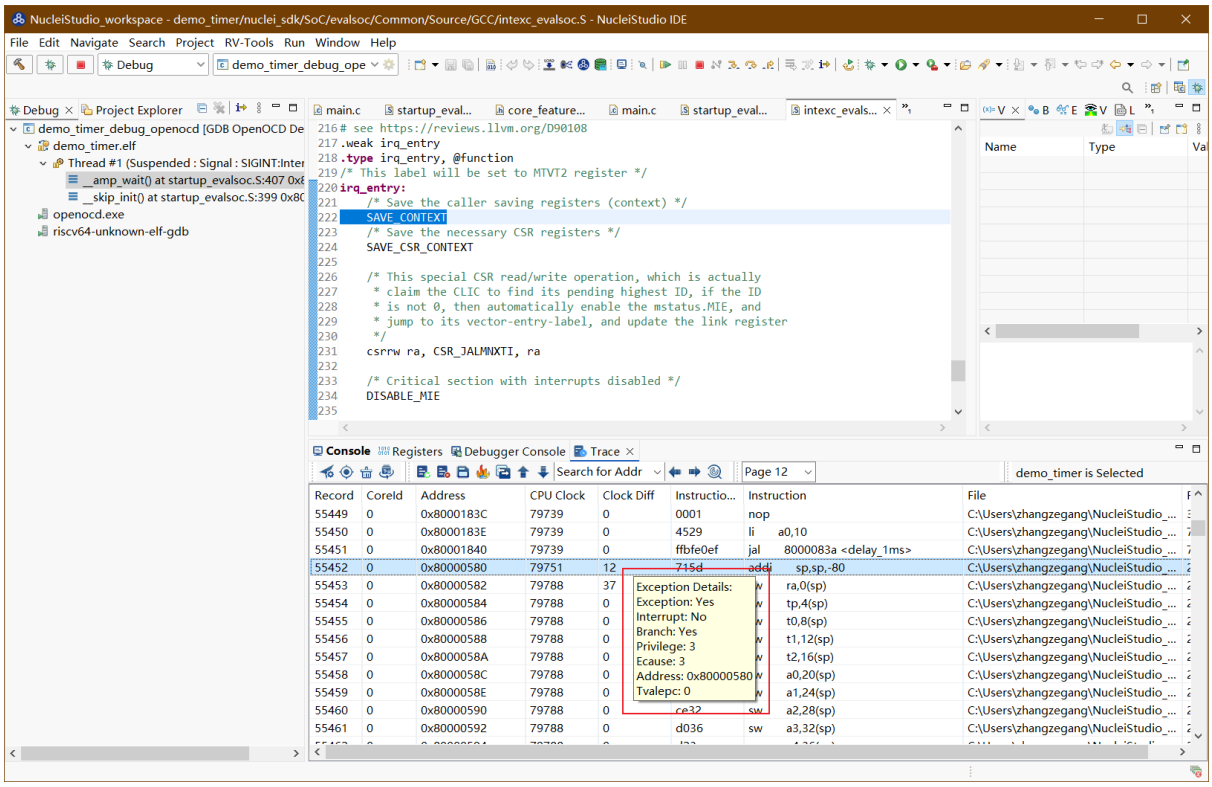
If a record in the Trace list contains an interrupt, the background color of that record turns green; if it contains an exception, the background color turns yellow. This helps you quickly distinguish interrupts from exceptions.



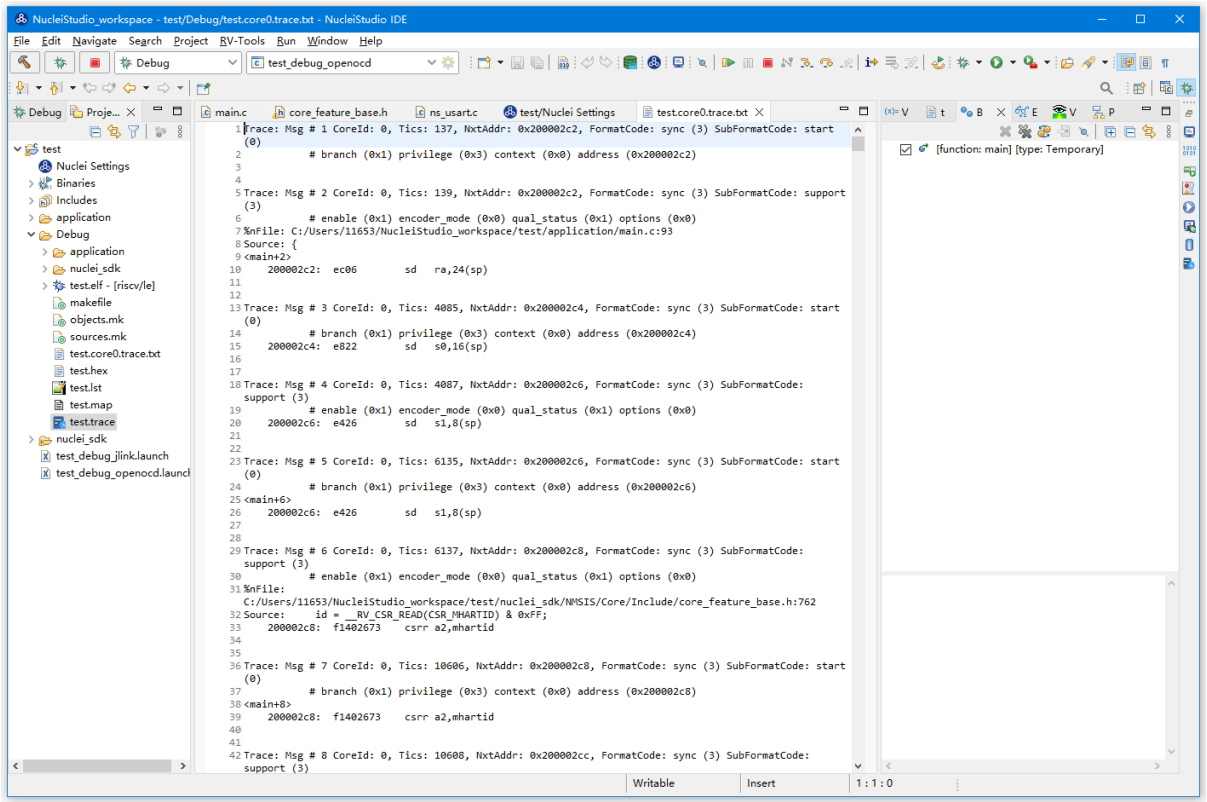
Click search exceptions or interruptions to view all interrupts or exceptions in the current project.



Double-click an interrupt or exception, and the Trace list automatically jumps to that record. Hover the mouse over the record to display its detailed information.

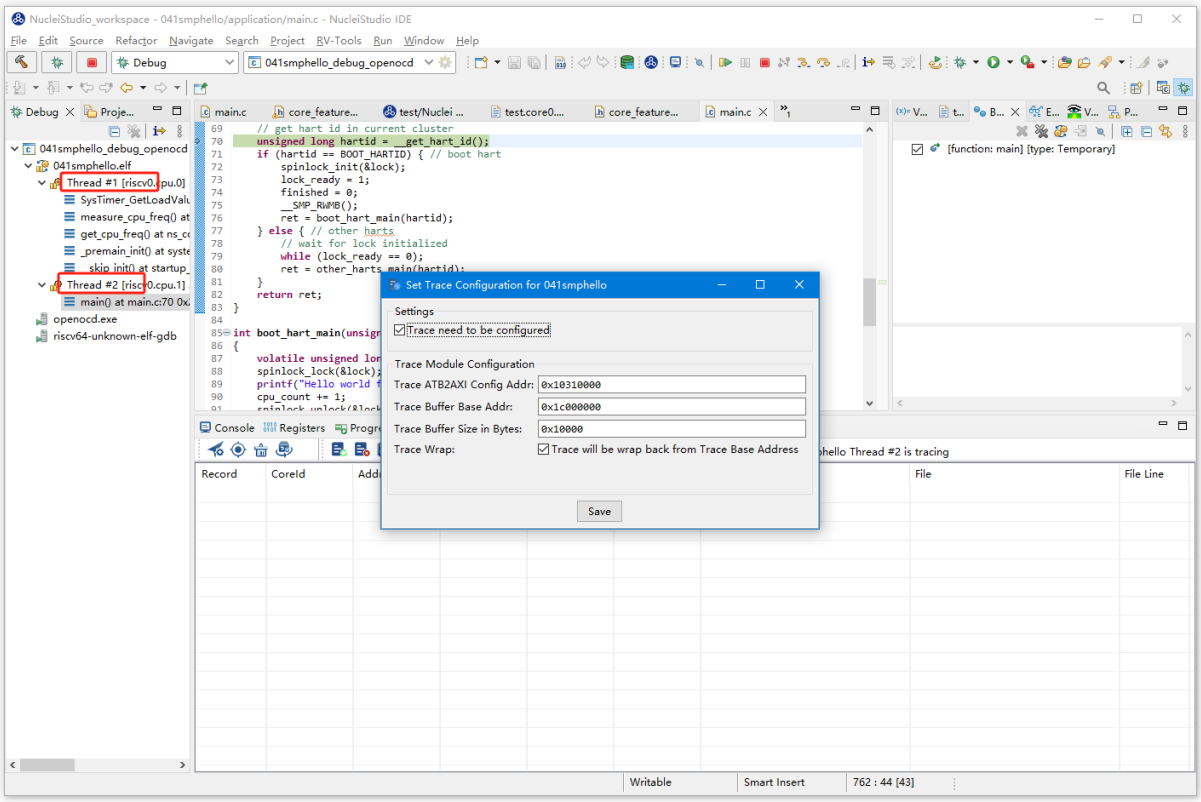


You can also double-click the `<projectname>.trace` file to view the trace file as text.

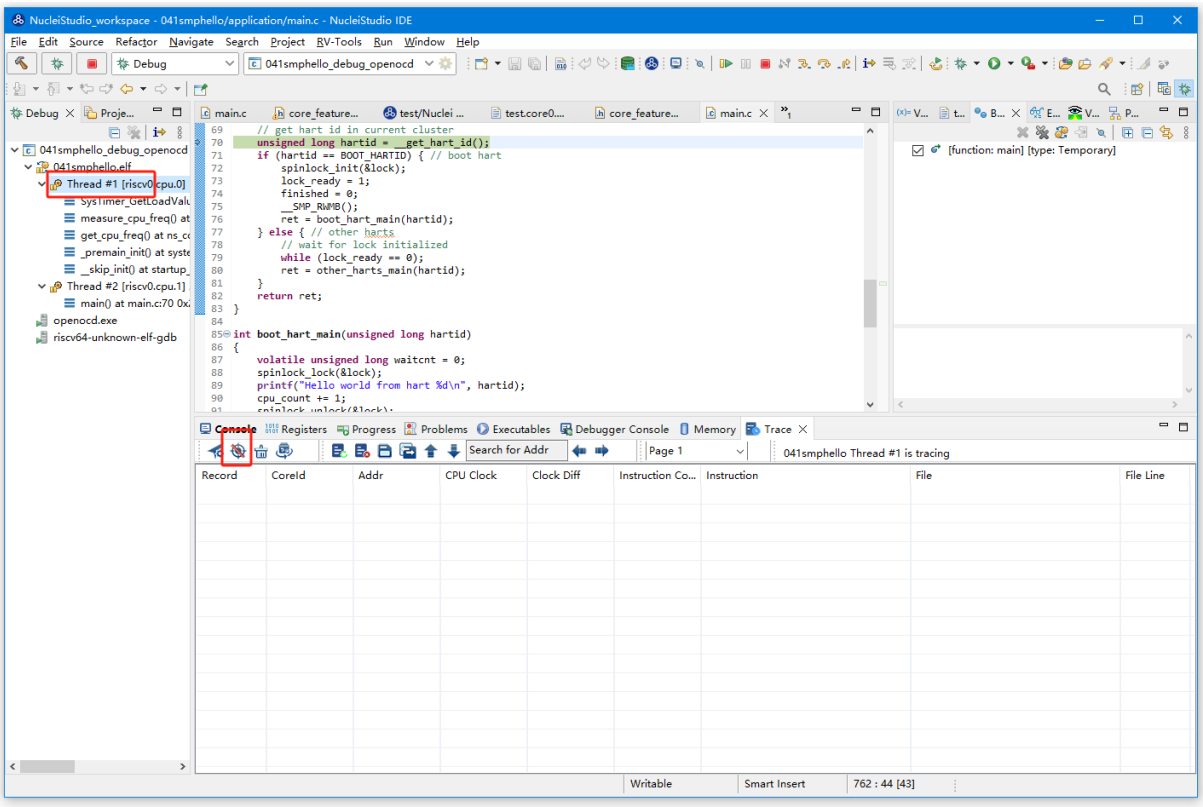


Using Trace in an SMP Multi-Core Application

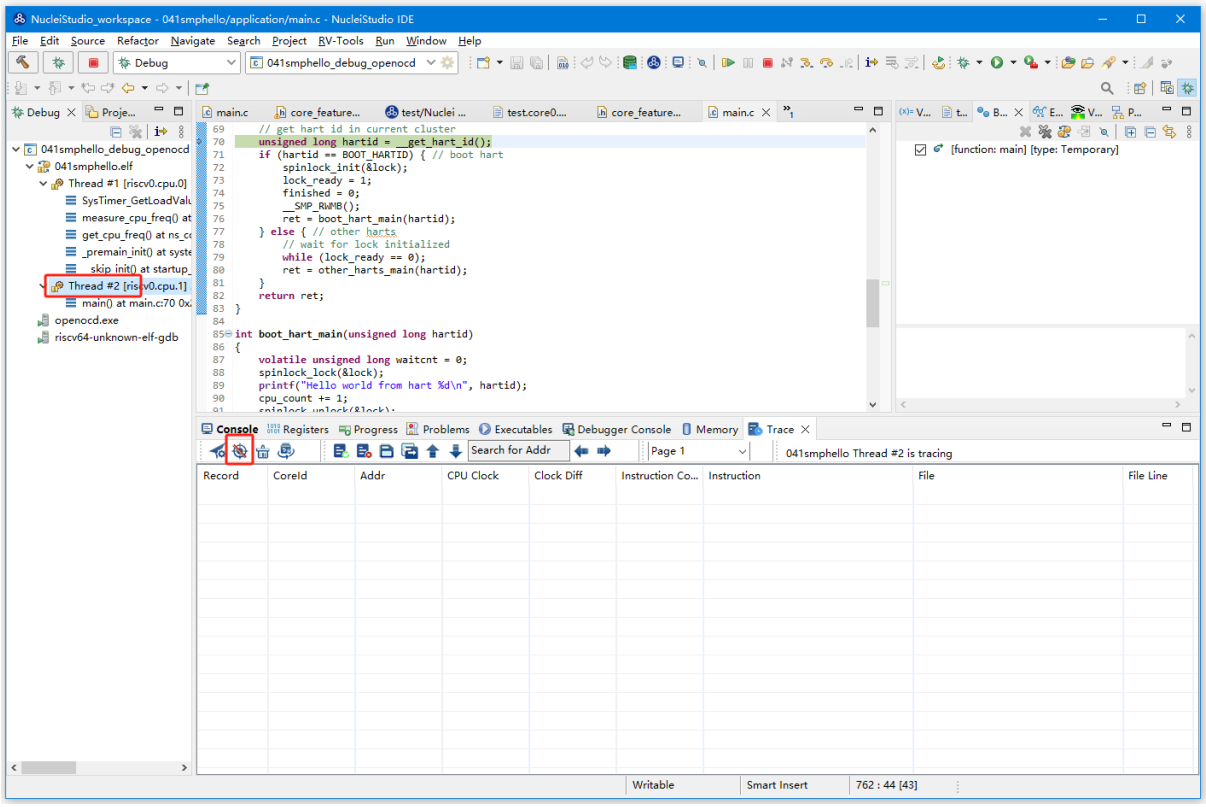
Using trace in an SMP multi-core application is similar to the single-core case. The difference is that in an SMP multi-core Debug session, all threads share one Trace Configuration, and you must select different Threads to `start trace/stop trace` for the individual CPU harts. In the Debug view, click a Thread, then click `trace setting` in the Trace toolbar to set up the Trace Configuration.



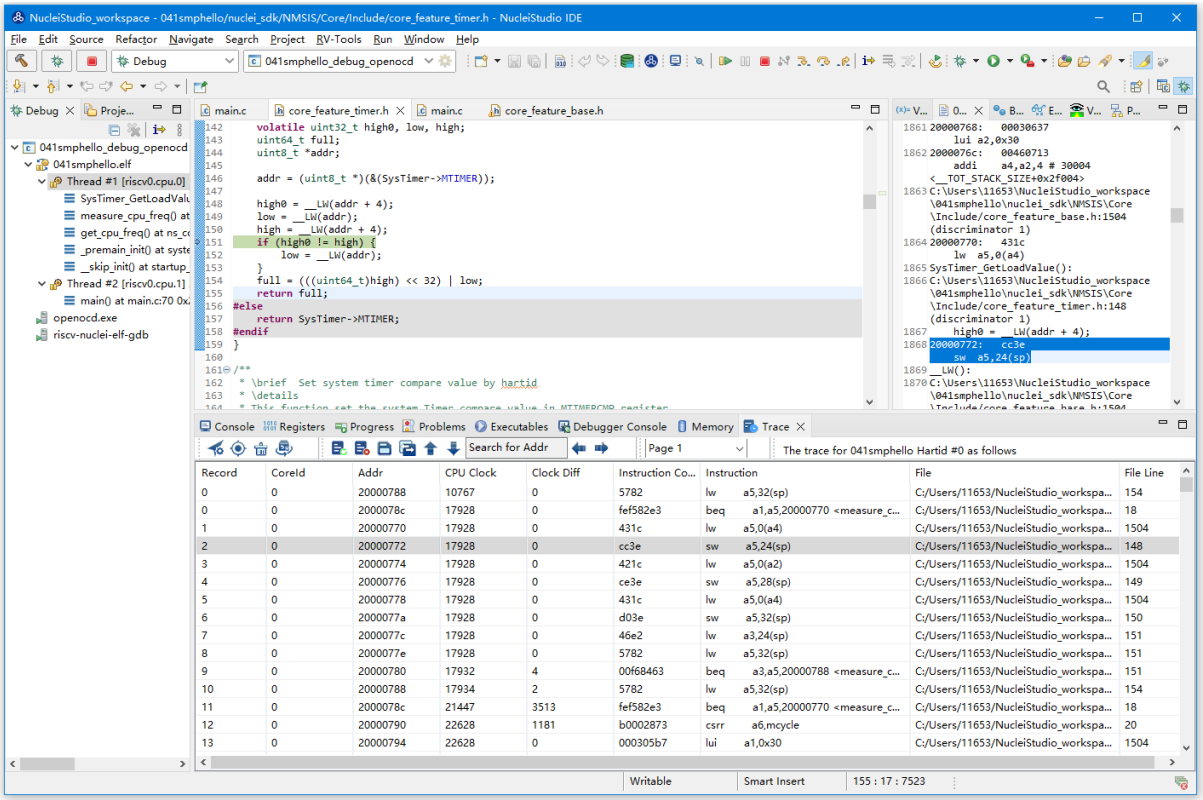
In the Debug view, switch between Cores by clicking different Threads. As shown below, clicking Thread #1 or the function name under Thread #1 selects Core 0 in the SMP multi-core application, and you can enable or disable Trace for Core 0. Once any Core has performed the start trace operation, the Trace Configuration is set in the hardware, and the other cores do not set it again when they perform `start trace`.



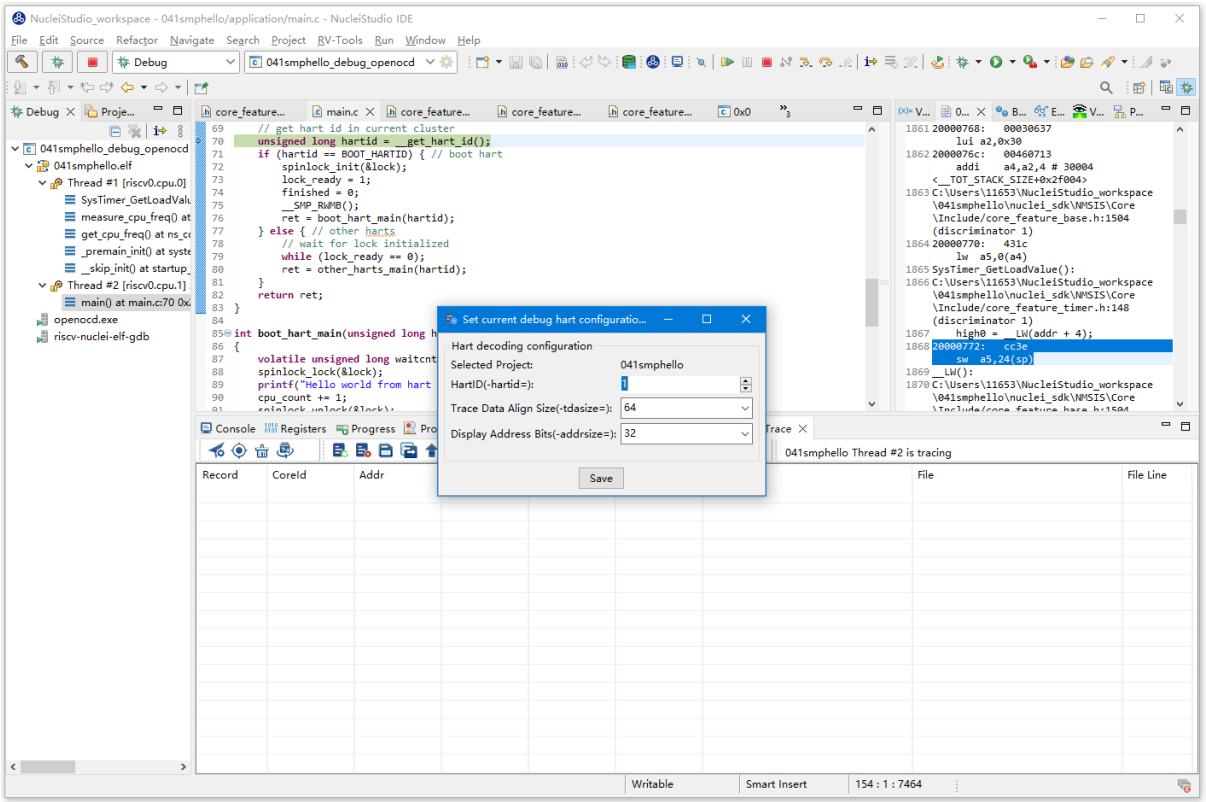
Similarly, click Thread #2 or the function name under Thread #2 in the Debug view to switch to Core 1 and perform the `start trace/stop trace` operations.



In an SMP multi-core application, the `dump trace file` command succeeds only after all Cores have performed `stop trace`. Only one copy of the Trace file needs to be downloaded. When decoding the trace file, set the Hart ID correctly to parse the trace record table for each core. As shown below, when `HartID=0`, you can view the Trace records corresponding to Core 0.

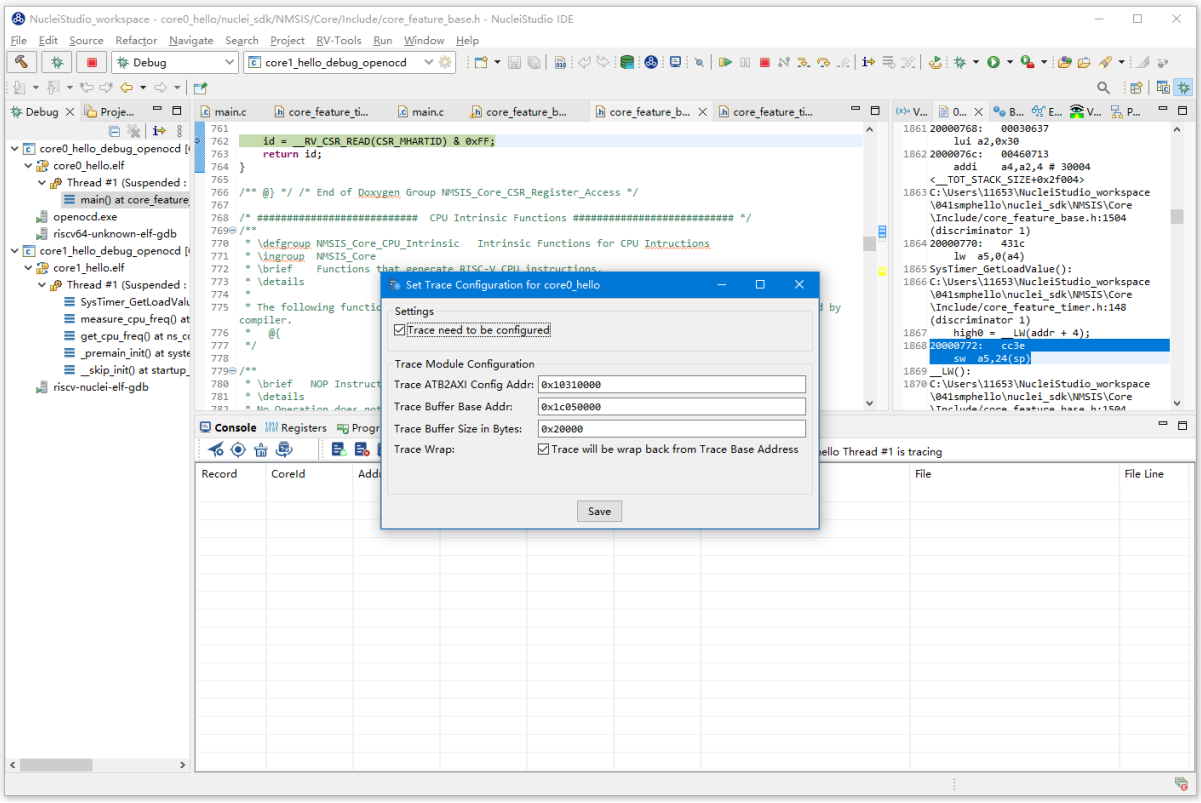


Similarly, when `HartID=1`, you can view the Trace records corresponding to Core 1.

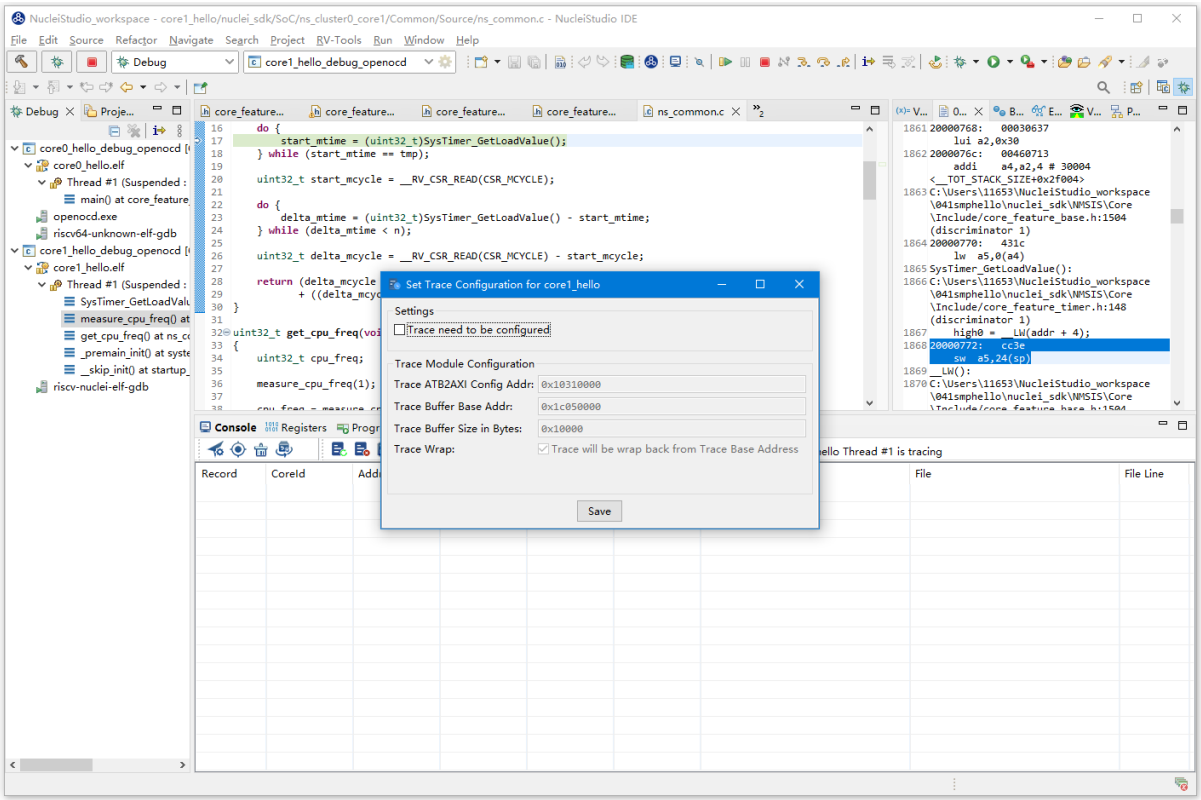


Using Trace in an AMP Multi-Core Application

Using trace in an AMP multi-core application is also similar, and the trace configuration is shared as well: different threads share one trace configuration, but you can `start trace/stop trace` for each core individually through its thread. As shown below, in the Debug view, click `Thread #1` or the function name under `Thread #1` to switch to `Core 0` in the AMP multi-core application, then click `trace setting` in the Trace toolbar to set up the Trace Configuration for Core 0.



In the Debug view, click `Thread #2` or the function name under `Thread #2` to switch to `Core 1` in the AMP multi-core application, then click `trace setting` in the Trace toolbar to set up the Trace Configuration for `Core 1`. Because the trace configuration is shared in an AMP multi-core application, uncheck `Trace need to be configured` here so that the trace feature is used without applying new settings.

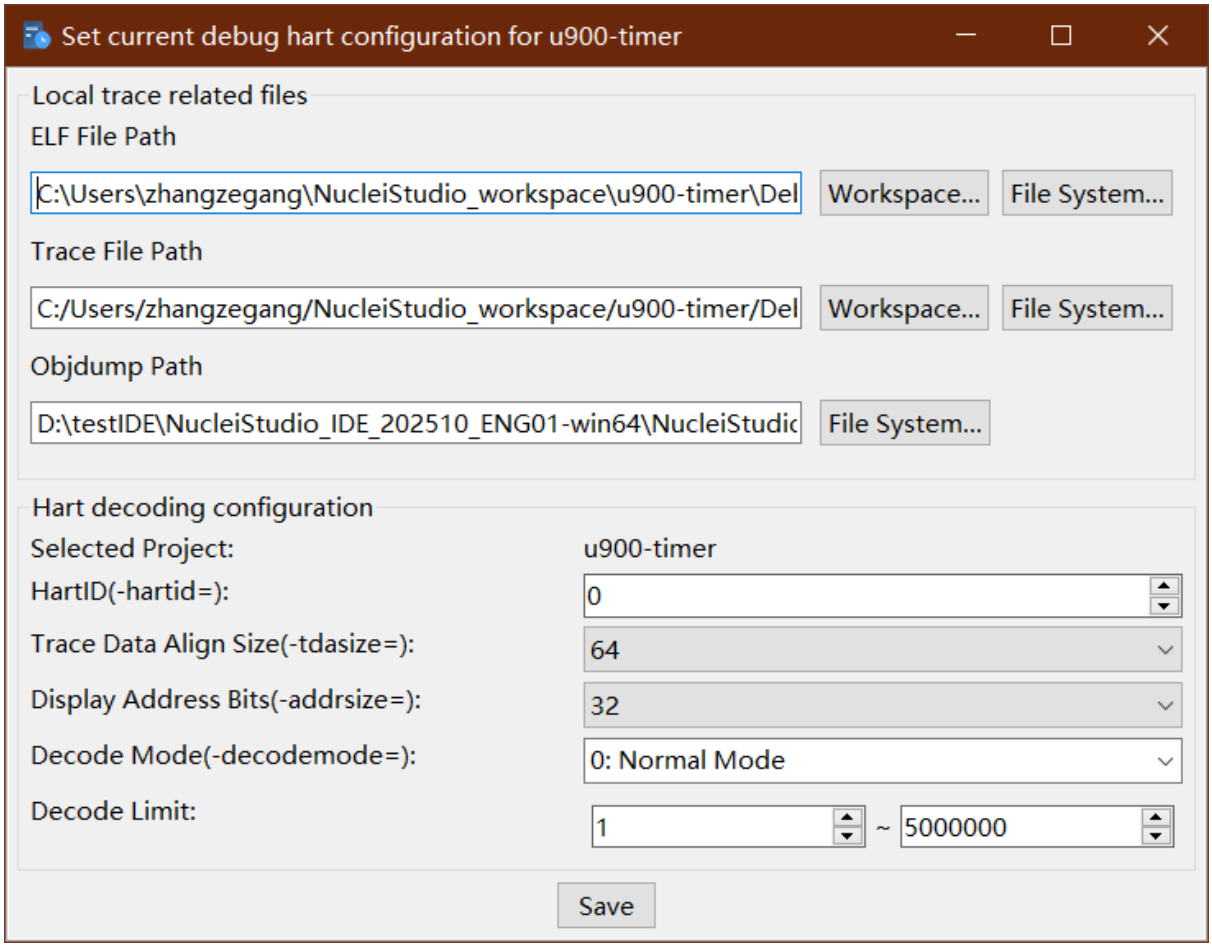


After the Trace Configuration is set, switch between Cores through the Threads in the Debug view to perform the `start trace/stop trace/dump trace file` operations. Note that the Core for which the Trace Configuration is set must `start trace` first, so that the configuration is written to the hardware; only then can the other Cores perform the `start trace/stop trace/dump trace file` operations normally.

In an AMP multi-core application, make sure all Cores have performed `stop trace` before executing `dump trace file`. Otherwise, one Core may dump the trace file while the other Cores are still recording, and the resulting Trace file will not match expectations. Each project application must dump its own copy of the trace file, although the contents of the dumped files are identical. When decoding the trace file, also set the `Core Hart ID` correctly so that the corresponding trace record table can be parsed. The remaining operations are the same as described above.

Viewing Offline Trace

If you have obtained a trace file through the command line or other means, open `Set Current Debug hart Configuration`, configure the parameters as required, and the trace tool in Nuclei Studio parses the trace file.



Using Trace in Headless Mode

Besides decoding trace in the Trace view, Nuclei Studio also provides a headless trace decoding application, `org.riscvstudio.ide.tools.trace.headless`. It uses the same options as `dqr` (see *TraceDecoder Decoding Trace Data* (page 818)), but it does not open the Nuclei Studio workbench, so it can be used in scripts and CI pipelines to decode trace files in batch.

Run the following command in the Nuclei Studio installation directory. On Windows:

```
<NucleiStudio>\eclipsec.exe -noSplash -data <workspace> ^
    -application org.riscvstudio.ide.tools.trace.headless ^
    -t <projectname>.trace -e <projectname>.elf -decodemode=0x11
```

On Linux:

```
<NucleiStudio>/eclipse -noSplash -data <workspace> \
    -application org.riscvstudio.ide.tools.trace.headless \
    -t <projectname>.trace -e <projectname>.elf -decodemode=0x11
```

The commonly used options are as follows. They are the same as the items in the `Set current debug hart configuration` dialog, so fill them in the same way as in the IDE:

Option	Description
-t <tracefile>	Trace file to be decoded, for example <projectname>.trace
-e <elffile>	ELF file executed when the trace is produced, for example <projectname>.elf
-od <objdump>	Objdump tool used for decoding; point it to <NucleiStudio>/toolchain/gcc/bin/riscv64-unknown-elf-objdump
-srcid=<n>	Hart ID to be decoded; the default is 0
-tdasize=<n>	Trace Data Align Size in bits: 8, 32, or 64; the default is 8
-tssize=<n>	TS Size, that is, the timestamp width in bits; the default is 13
-decodemode=	Decode Mode, for example 0x00 for ATB2AXI (Trace 1.0) and 0x11 for Trace bundle; the default is 0
-datatrace=<	Datatrace, that is, the data trace index width; the default is 0
-contextsize	Context Size in bits; the default is 0
-freq=<n>	CPU frequency in Hz; the default is 0
-o <outfile>	Write the decoding result to a file instead of printing it to the console
-analytics	Print the decode statistics, such as the number of decoded instructions

For example:

```
<NucleiStudio>\eclipsec.exe -noSplash -data D:/ws_trace ^
    -application org.riscvstudio.ide.tools.trace.headless ^
    -t D:/work/test/Debug/test.trace ^
    -e D:/work/test/Debug/test.elf ^
    -od <NucleiStudio>\toolchain\gcc\bin\riscv64-unknown-elf-objdump ^
    -tdasize=8 -tssize=13 -srcid=0 -decodemode=0x11 ^
    -o D:/work/test/Debug/trace.txt
```

The decoding result is printed as text, the same as the text exported by the IDE; with -o <outfile>, it is written to the specified file instead.

Note:

- The application always returns the exit code 0, so check the output instead: the decoding succeeds when the output contains Trace: / File: / disassembly lines and contains no Error:.
- If the trace format of the board is unknown, try the decode modes in turn (0, 0x11, 0x10, 0x12, 1, 2, 3) and use the first one that produces records.
- The headless application outputs text only. To obtain a flame graph, use the IDE (see *Using Flame View with Trace* (page 269)).

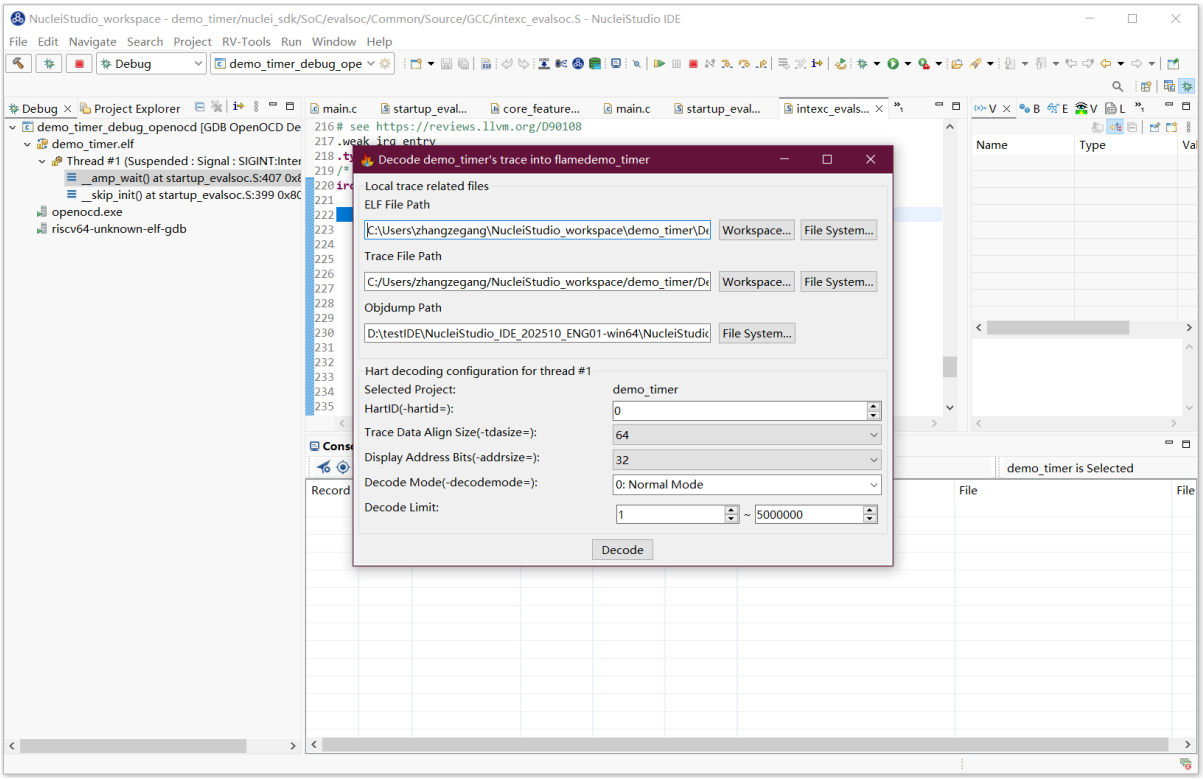
Using Flame View with Trace

Note: In Nuclei Studio 2025.10, we introduced the Flame View feature, which clearly presents the call stack and time consumption during program execution through a visual flame graph, enabling developers to analyze performance hotspots more efficiently and accelerate troubleshooting.

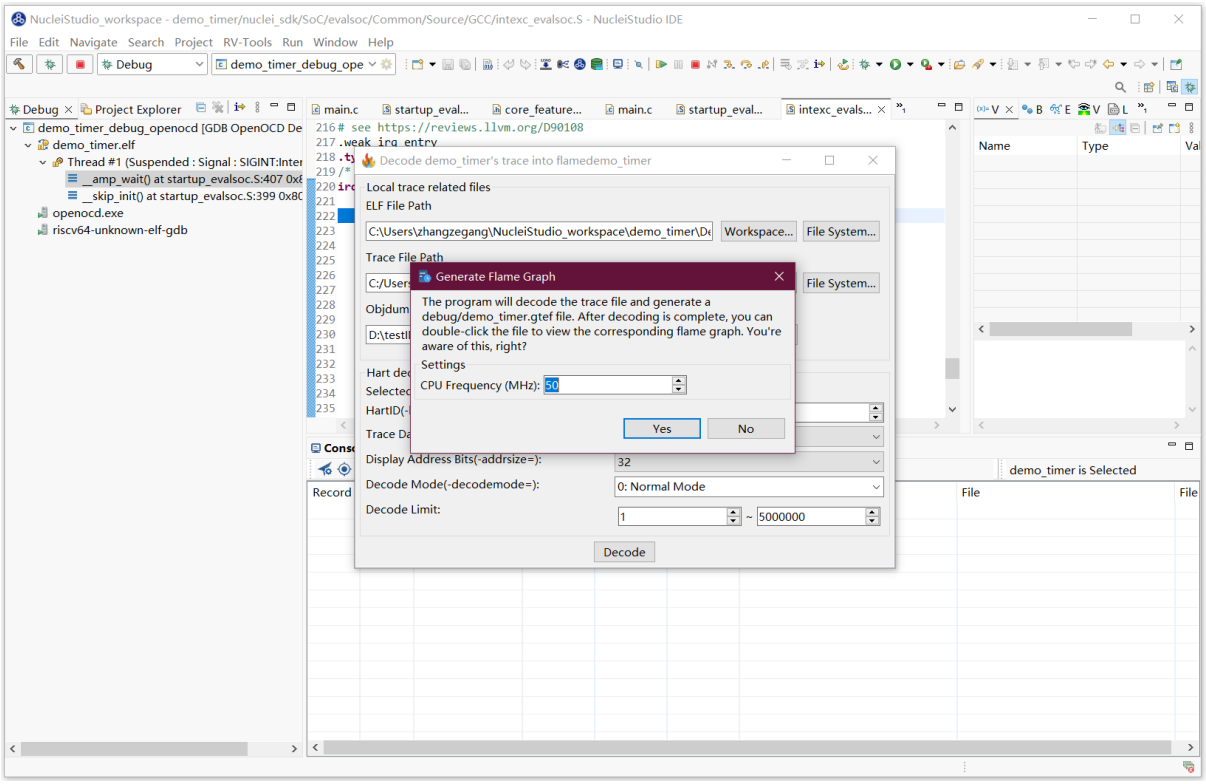
Trace inherently records the program execution process, so it is well suited for parsing into a flame graph. Assuming you have already obtained the <projectname>.trace file, click the Decode trace into flame button.

If the <projectname>.trace file has not been parsed yet, the Trace parsing configuration page pops up first (see the previous sections for the configuration items). Complete the configuration and click Decode to start

parsing.

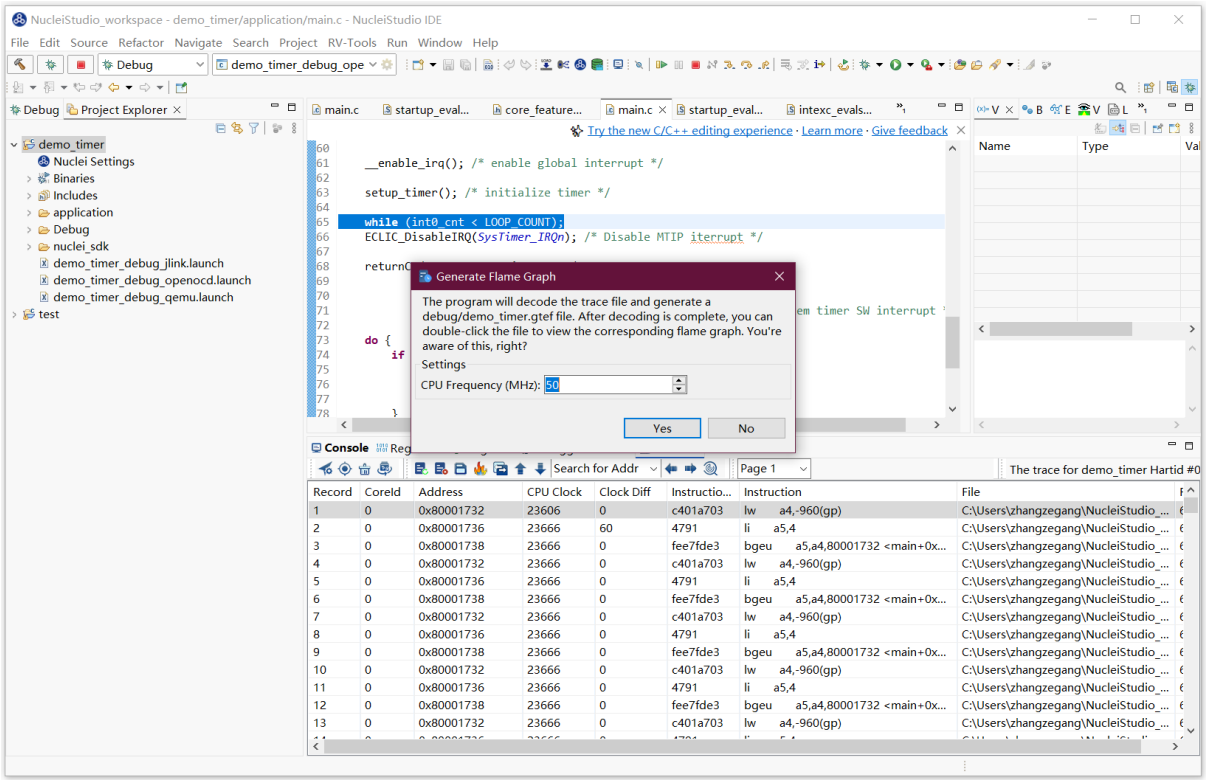


At the same time, the IDE opens the Flame View import wizard, which prompts that the corresponding `<projectname>.trace` file is about to be generated. Once the file is generated, double-click it to import it into Flame View for visual analysis. The wizard asks you to enter the current CPU operating frequency (for example, 50 MHz in this test) so that the timing information can be restored accurately.

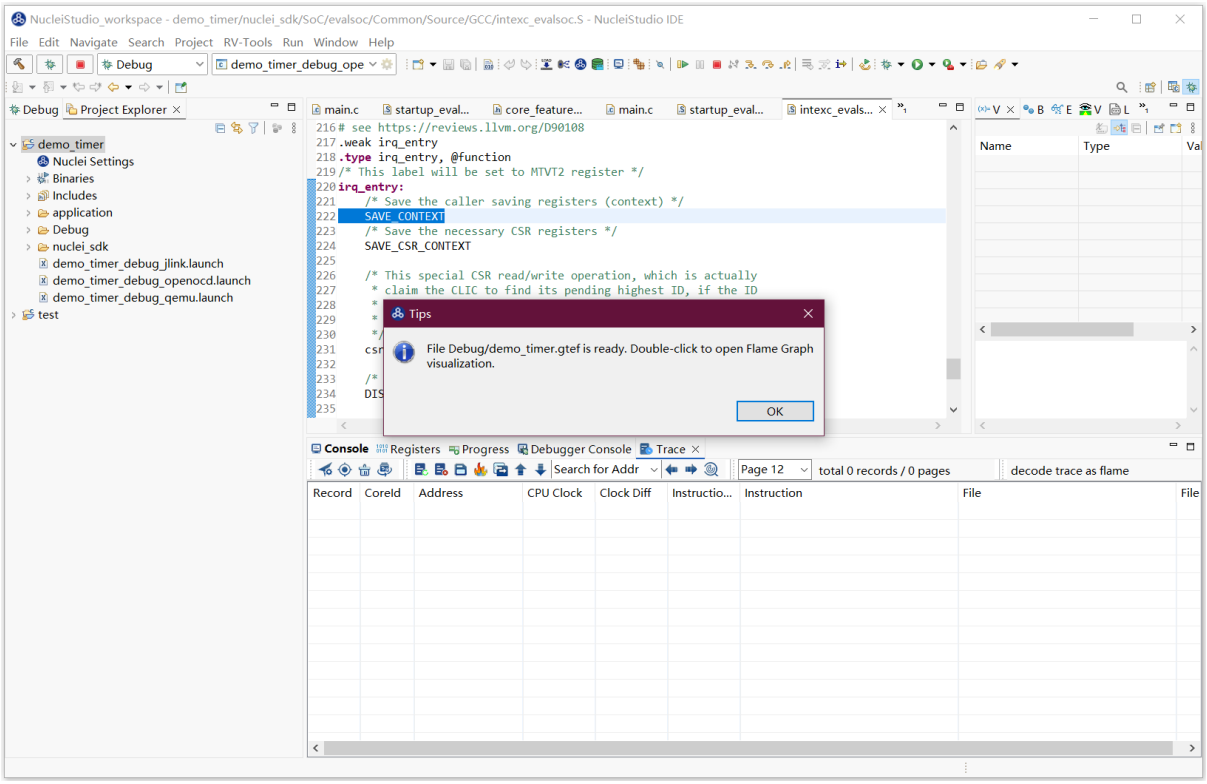


If the `<projectname>.trace` file has already been parsed, the corresponding trace data is extracted directly from the existing Trace list, and the Flame View import wizard pops up immediately, skipping the repeated parsing

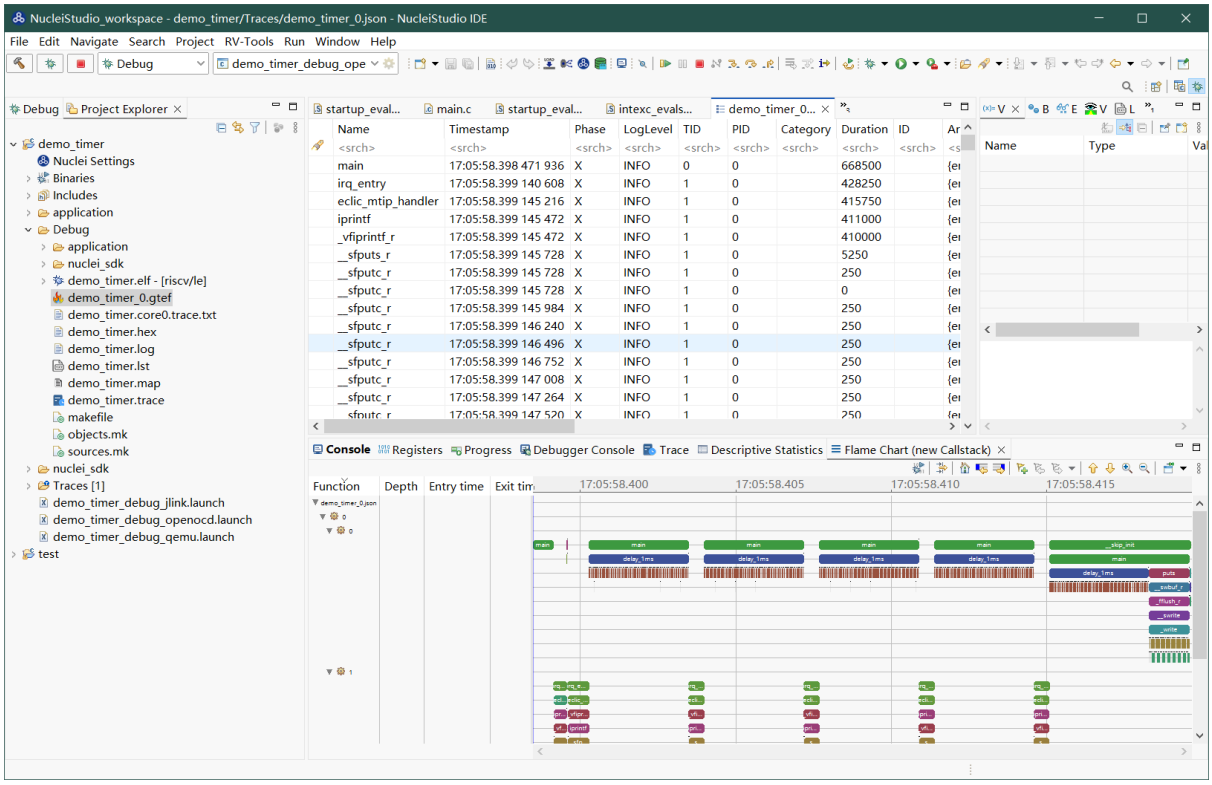
step.



After the Trace list is parsed and the `<projectname>.gtef` file is generated, the IDE automatically prompts that the operation is complete.



A `<projectname>.gtef` file now exists in the Debug directory. Double-click the file to view the flame graph of the project. **Note**: if the `<projectname>.gtef` file is updated, double-click the file again to re-import it into Flame View. For details about the Flame View feature, see *Using Flame View* (page 272).

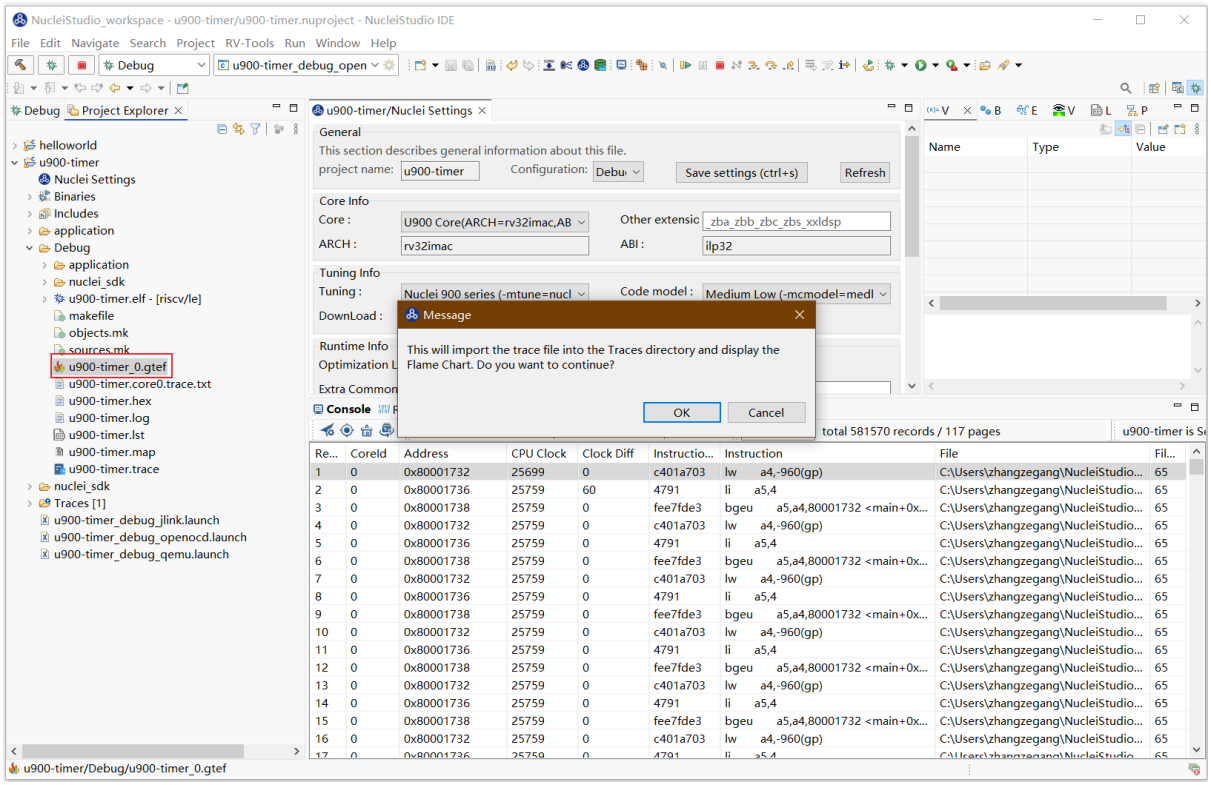


2.10.6 Flame View

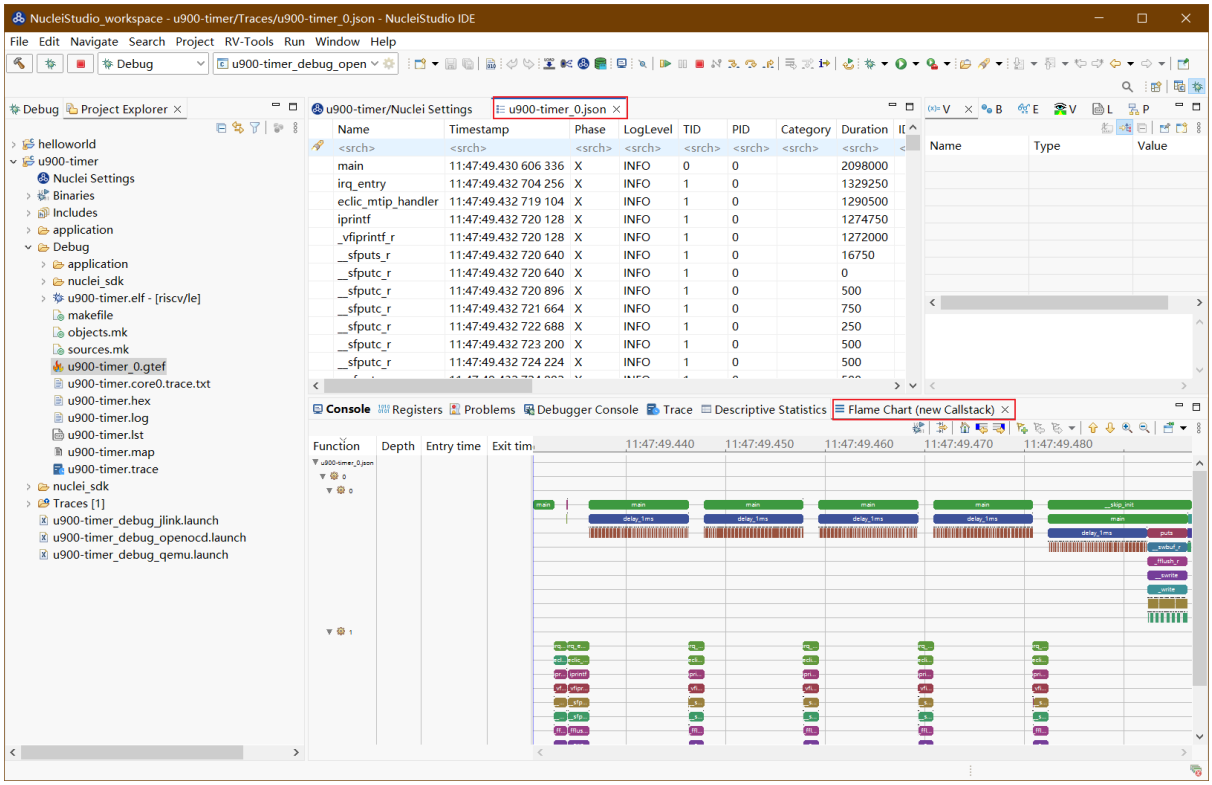
Flame View is an intuitive and efficient performance analysis tool used to visualize the execution call stack and time distribution of programs in embedded systems. By converting complex trace data into a hierarchical stacked graph, Flame View helps developers quickly identify hotspot functions, analyze execution paths, and locate performance bottlenecks.

The `*.gtef` file is a file format defined by us in the IDE specifically for the Flame View tool. Currently, in the IDE, a `*.gtef` file can be parsed from Trace data (see *Using Flame View in Trace* (page 269)), and a `*.gtef` file can also be produced directly by Nuclei Model (see *Using Flame View in Nuclei Model* (page 296)).

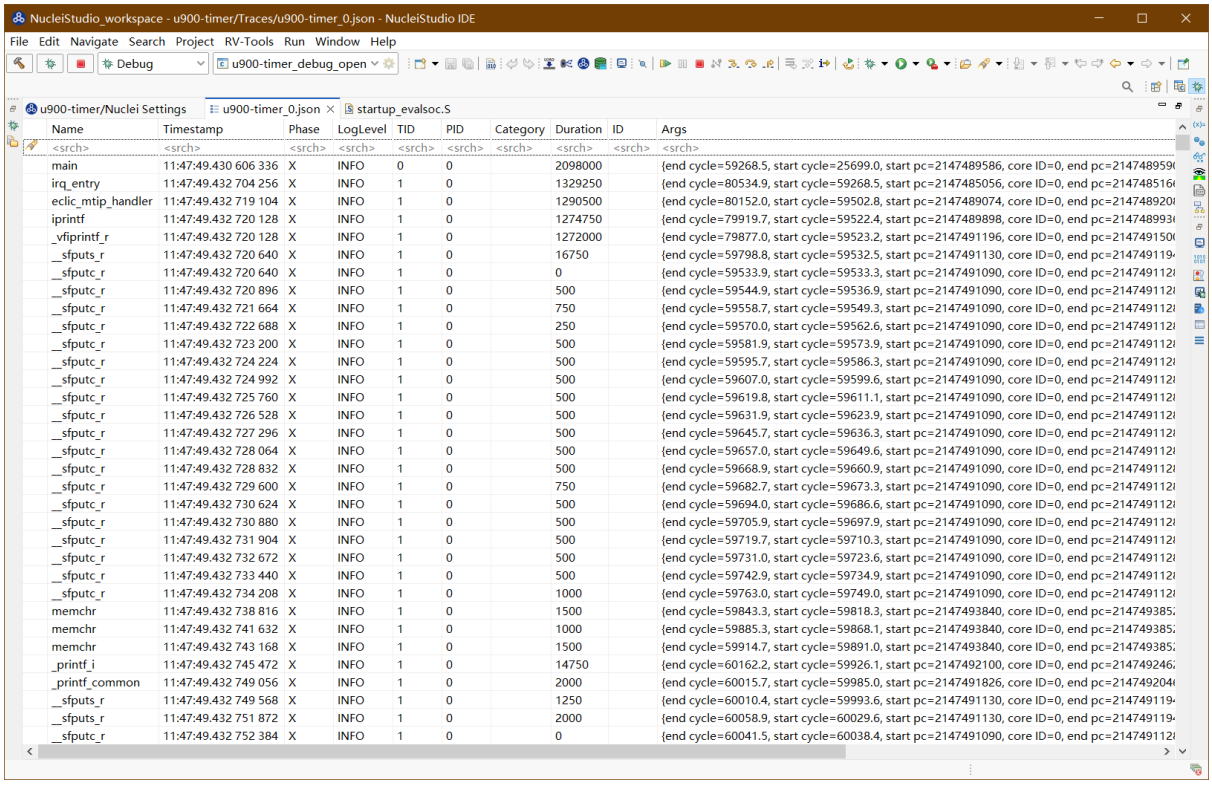
If you already have a `*.gtef` file, you only need to double-click the file to start the Flame View tool. First, a guide prompt pops up, informing the user that the IDE will import the `*.gtef` file into the Traces directory.



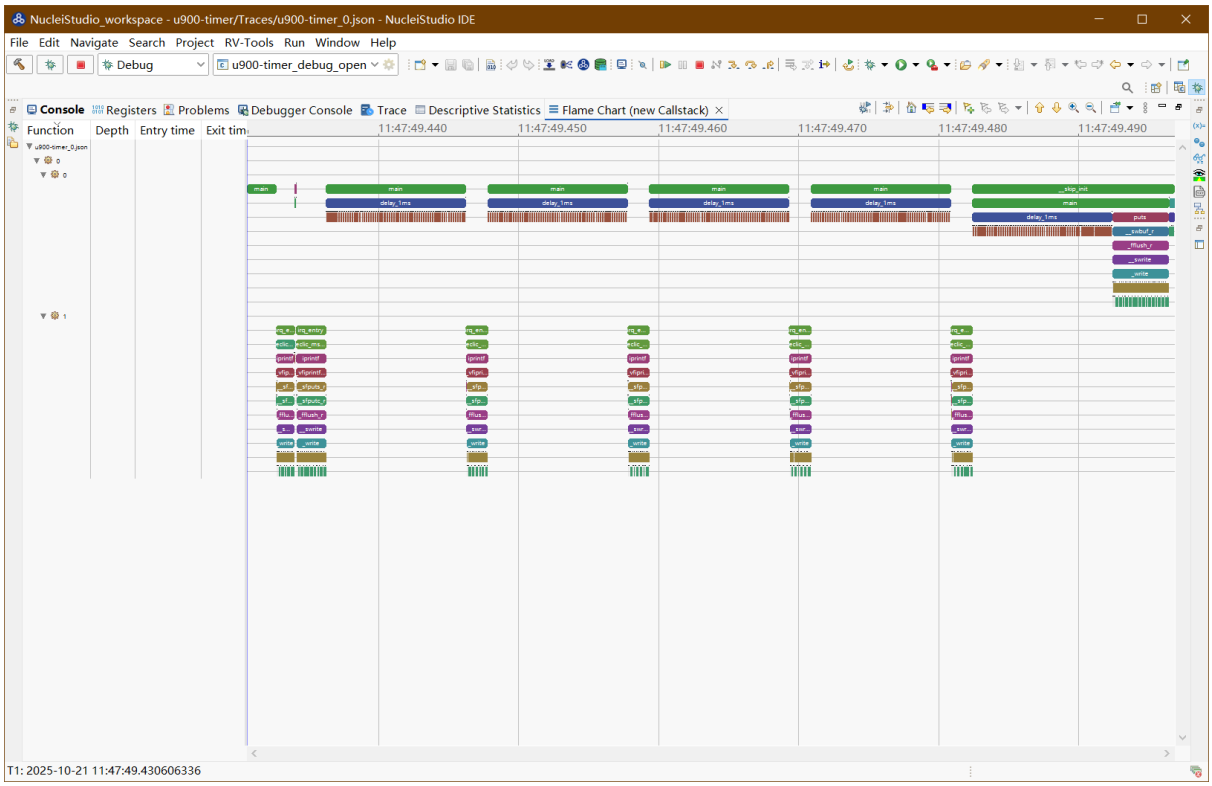
At the same time, the IDE opens the Flame Chart view and the Flame Graph view.



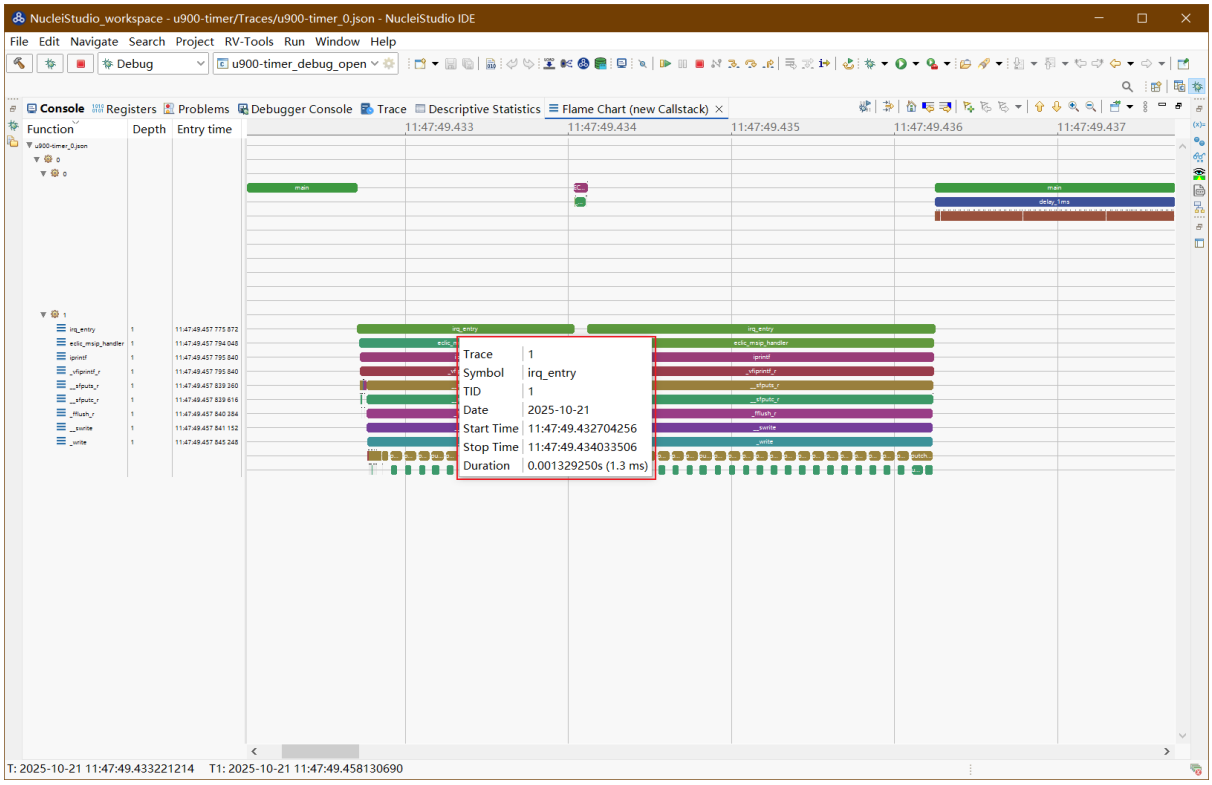
The Flame Chart view shows function-related information in list form, such as start time and execution time consumption.



The Flame Graph view shows data such as the execution time, call hierarchy, and execution time consumption of each function in the program in the form of a flame graph. In the Flame Graph view, users can use `A D W S` to zoom the flame graph in and out to view more detailed information.

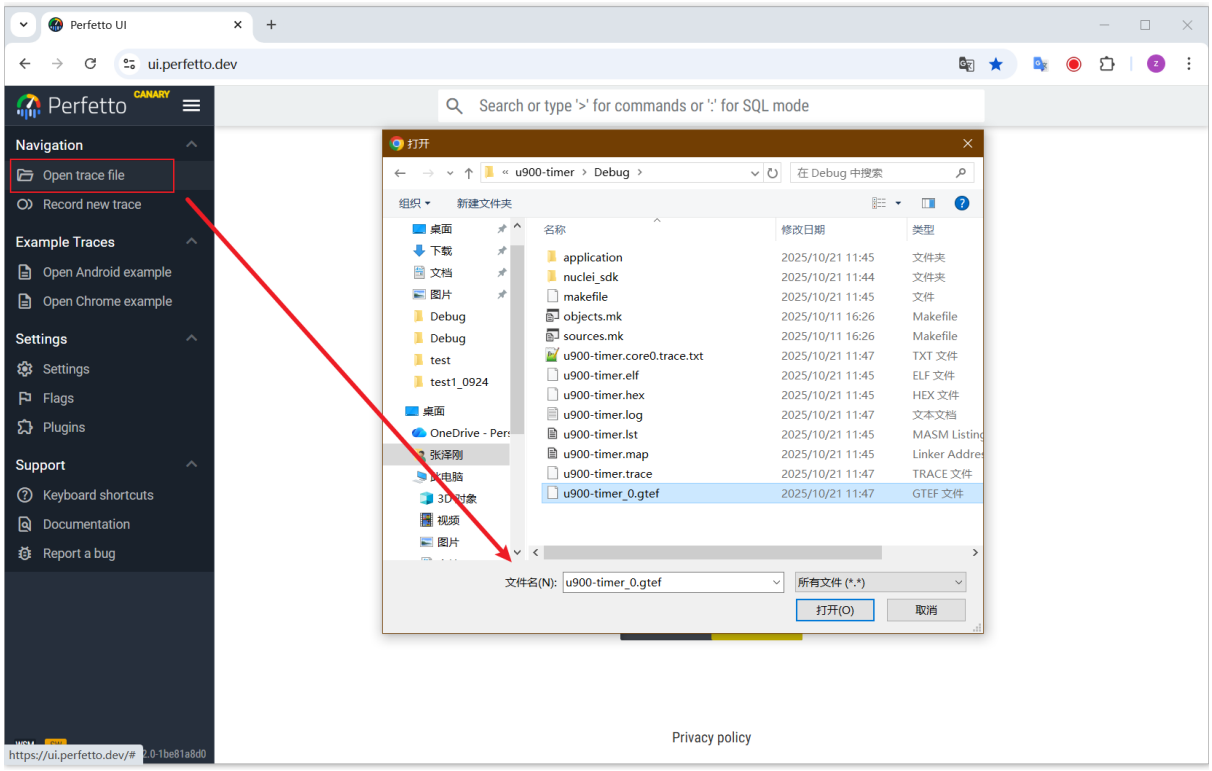


When the mouse clicks a segment in the flame graph, more detailed information about that segment is displayed.

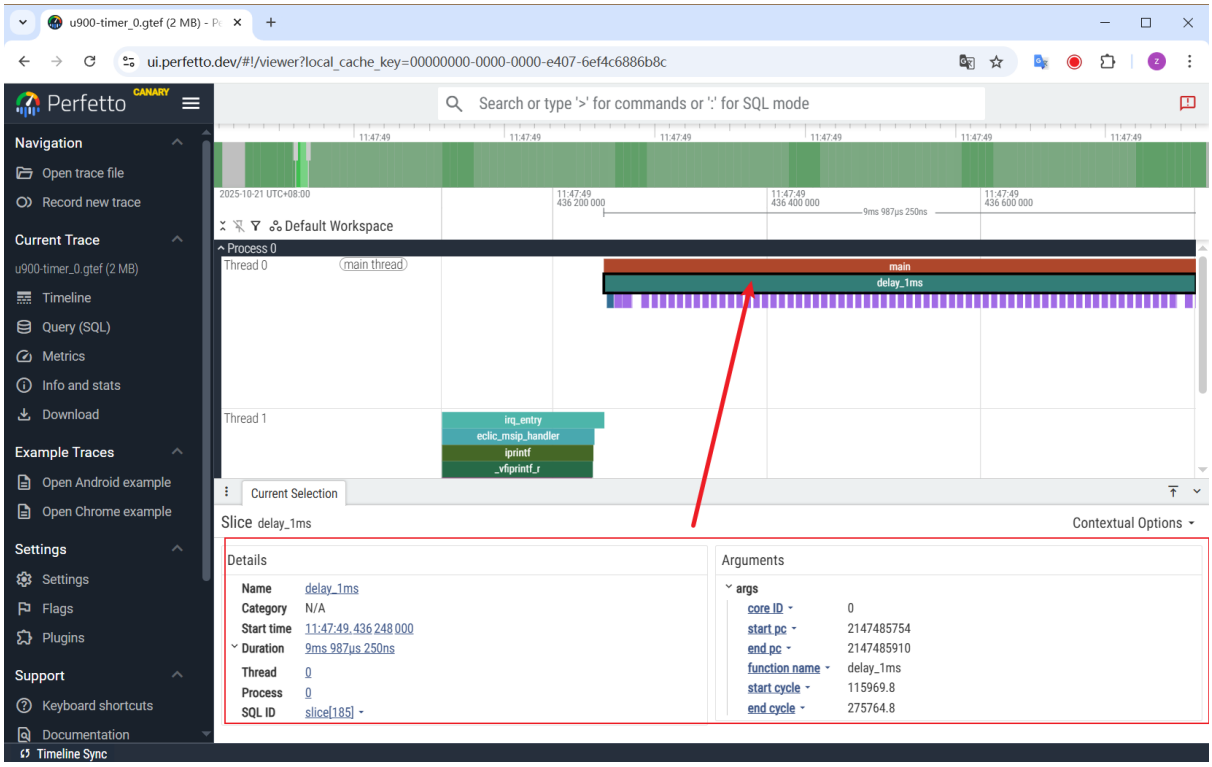


Through the flame graph, users can understand the execution and time consumption of the program, and quickly identify hotspot functions, analyze execution paths, and locate performance bottlenecks.

The `*.gtef` file can also be displayed with Google ' s open-source tool Perfetto Trace Viewer. Open `https://ui.perfetto.dev/`, click `Open trace file`, find the `*.gtef` file generated in the project, and manually load the json file into Perfetto.



In the generated flame graph, users can use `A D W S` to zoom the flame graph in and out to view more detailed information.



2.10.7 Using RVProf

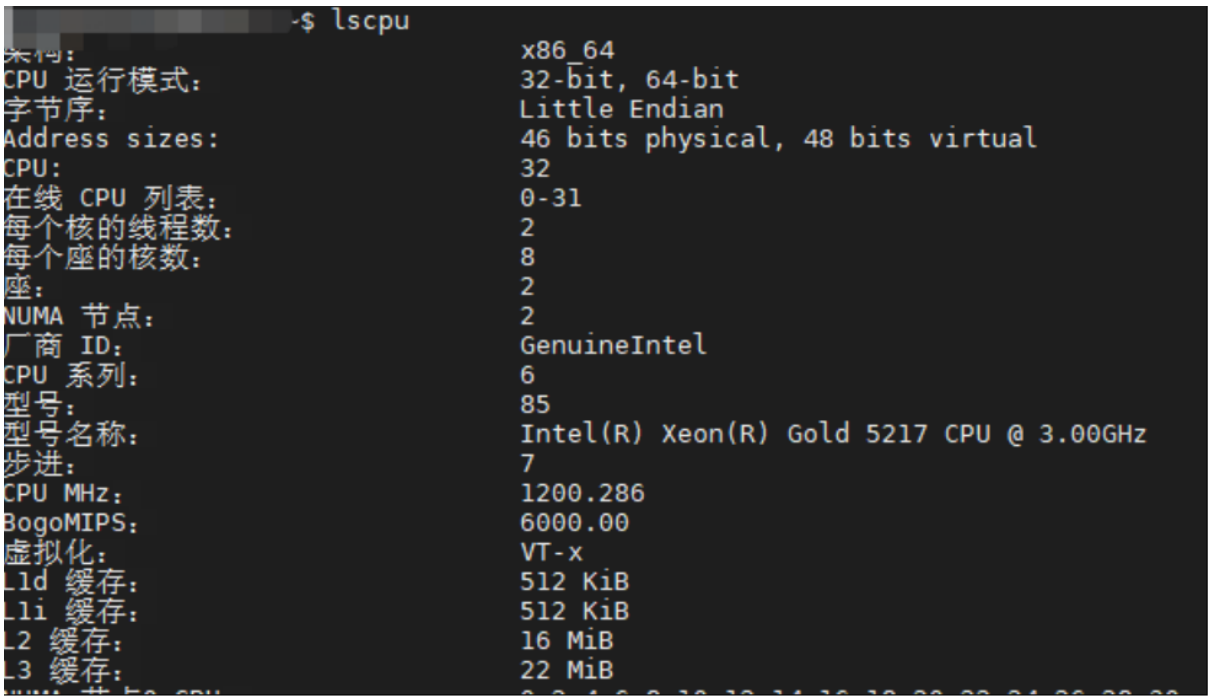
Note: Starting with Nuclei Studio 2025.10, the RVProf feature is supported on Windows, with the same usage as on Linux.

RVProf is a performance analysis tool developed by Nuclei System Technology for the CPU cycle model. Nuclei Studio has supported RVProf since version 2024.02.dev. Using RVProf involves three steps: the Cycle model tool runs the code and produces a `.rvtrace` file; the RVProf tool parses the `.rvtrace` file into a `.json` file; and Google ' s open-source Perfetto Trace Viewer parses and displays the `.json` file. Because the CPU cycle model is currently only available for Linux, this document demonstrates the feature in a Linux environment.

If you encounter any problems during use, check https://github.com/Nuclei-Software/nuclei-studio or submit an issue.

Test Environment

The CPU cycle model places high demands on the hardware at runtime. It runs better on systems with four or more cores, and using it in a virtual machine is generally not recommended. For a better experience, this test was performed on a workstation.



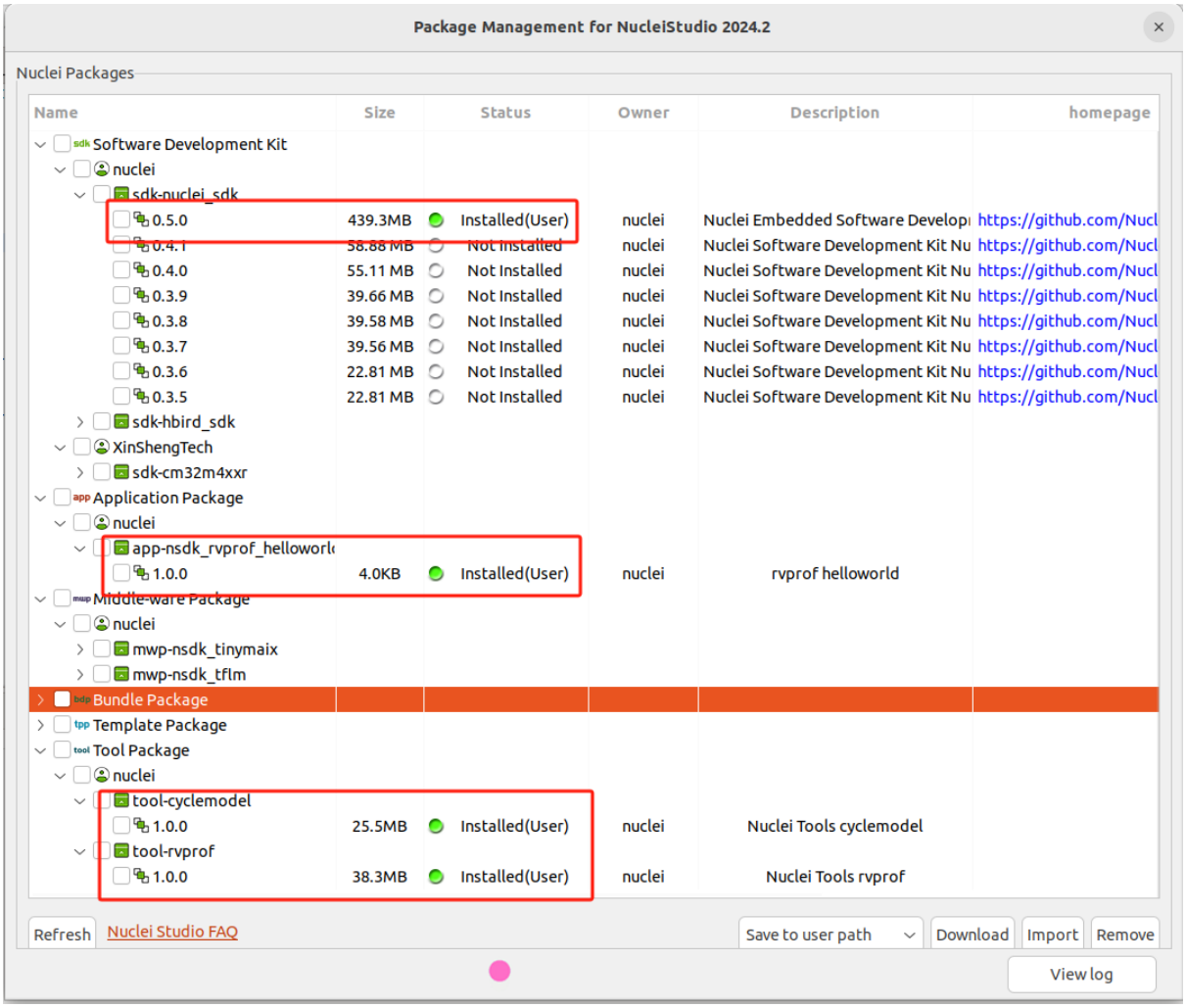
Preparing the Test NPK Packages

Currently, this feature requires test packages in Nuclei Package (NPK) format. Install the related packages into Nuclei Studio; for the installation procedure, see the NPK package sections of the Nuclei Studio manual. Because the RVProf test packages are not public, contact us to obtain them.

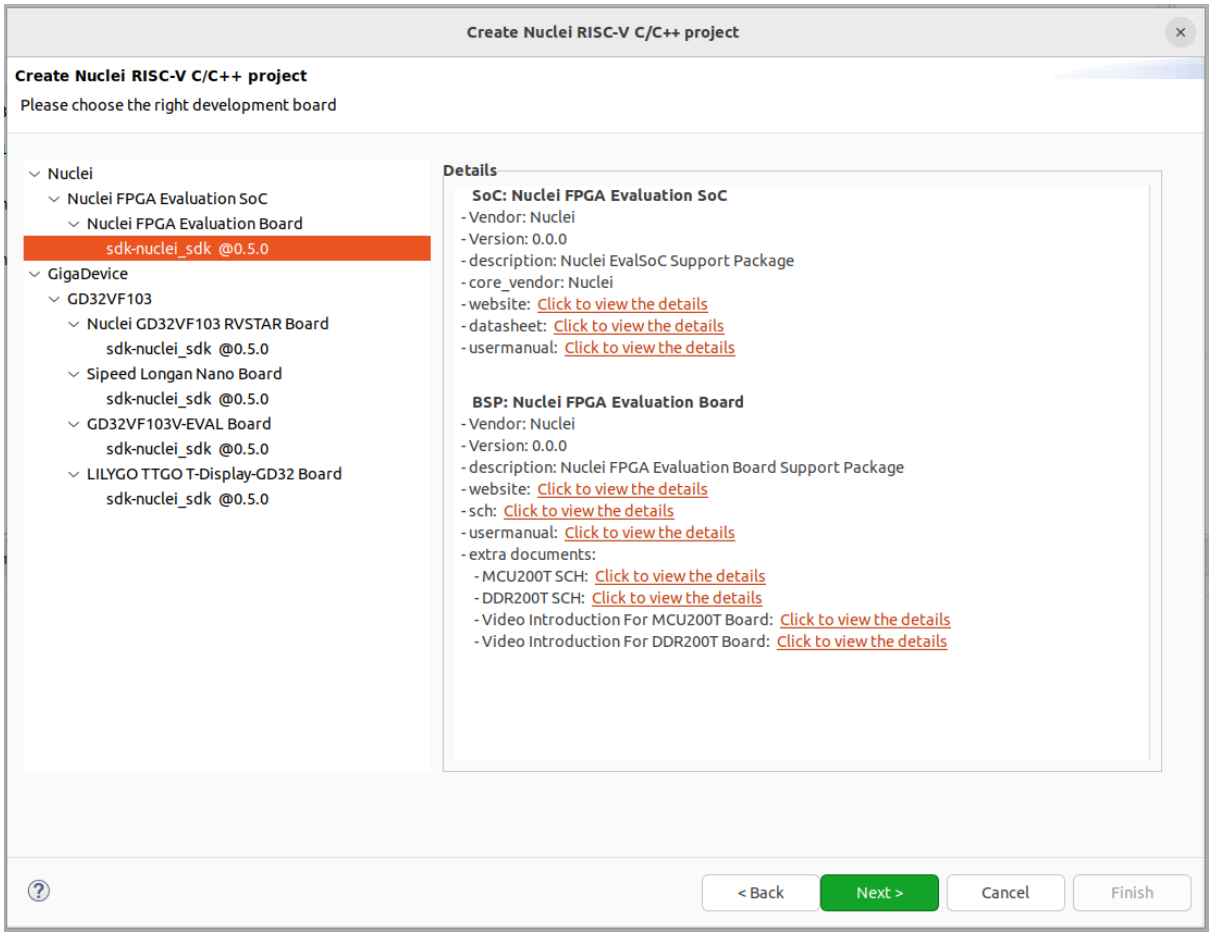
- cymodel.zip: the NPK Tools package for cymodel
- rvprof.zip: the NPK Tools package for RVProf
- Rvprof helloworld.zip: the test demo NPK App package

Creating the RVProf Test Project

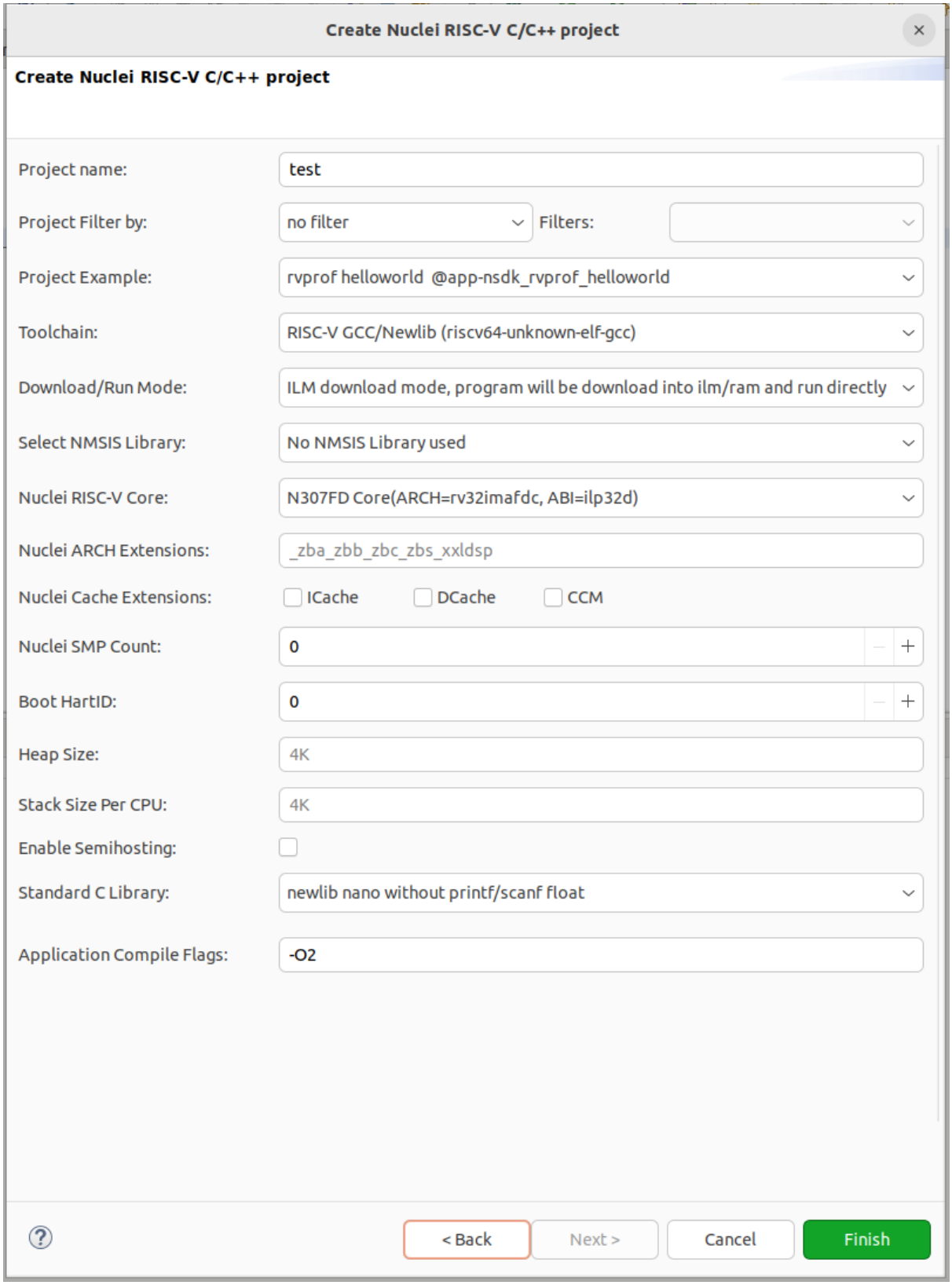
Before creating a project, check in Nuclei Package Management that the NPKs are installed correctly. Because the test demo depends on nuclei_sdk, sdk-nuclei_sdk must also be installed first, as shown below:



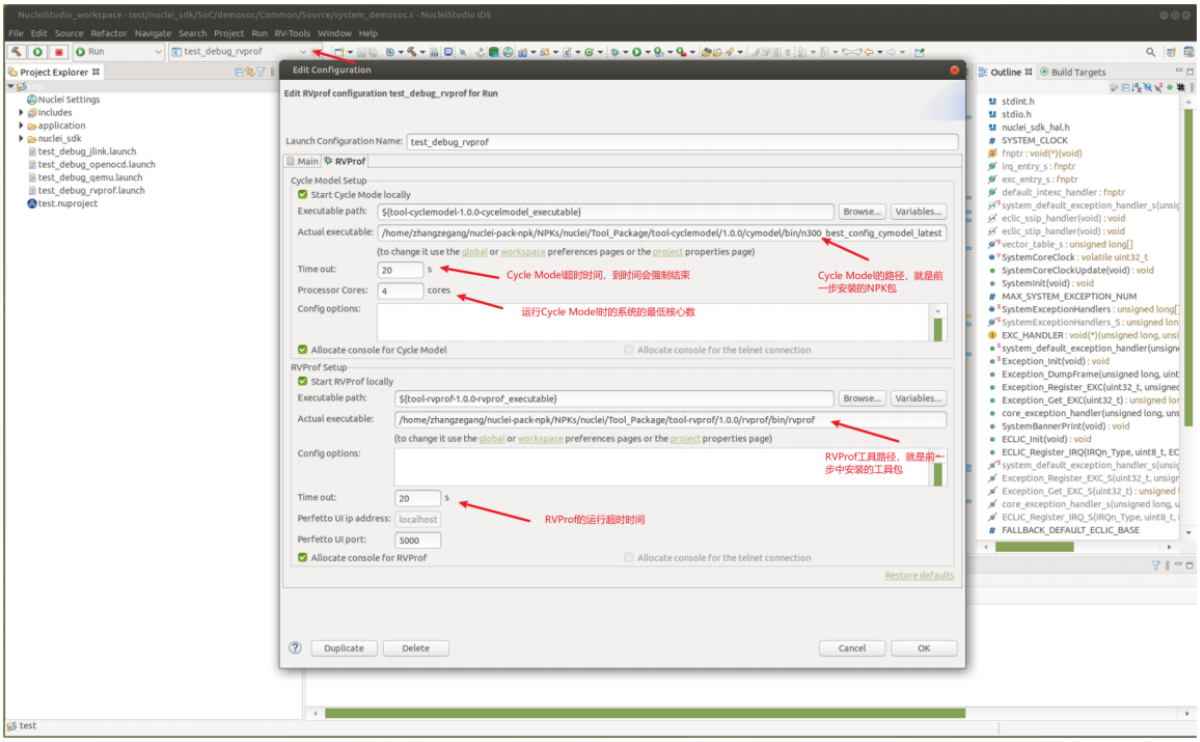
Then create a test project. In the project creation wizard, go through `New Nuclei RISC-V C/C++ Project -> sdk-nuclei_sdk@0.5.0 -> next`. On the project configuration page, enter the project name, select Project Example `rvprof helloworld@app-nsdkrvprof_helloworld`, and set Nuclei RISC-V Core to `N307FD` (the core must match the CPU cycle model).



Under Project Example, select the Rvprof helloworld project from the imported demo NPK App, then click Next to complete the project creation.



The created test project contains an additional `test_debug_rvprof.launch` file with the RVProf configuration, as shown below. The Cycle Model timeout value sets the Cycle Model timeout. Because Cycle Model takes a long time to run, set a short timeout for a relatively simple project so that the Cycle Model run is interrupted promptly when the time is up. The timeout in RVProf serves a similar purpose.

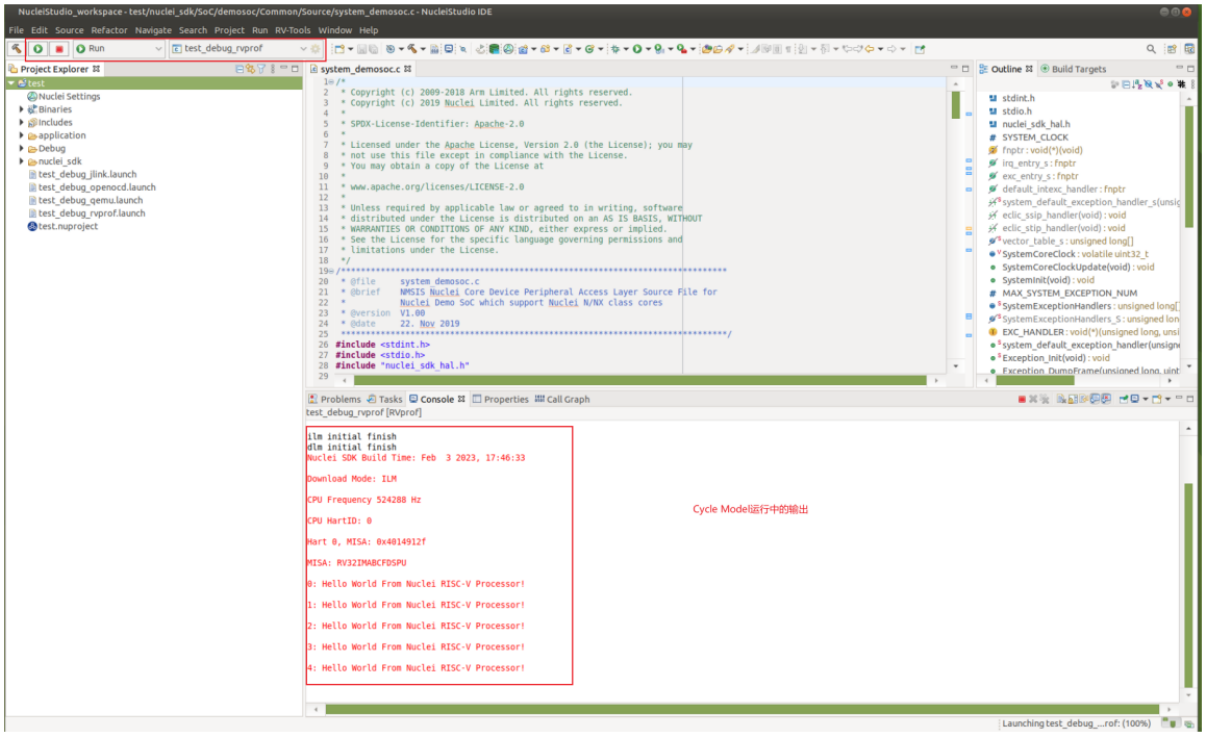


Viewing the RVProf Results

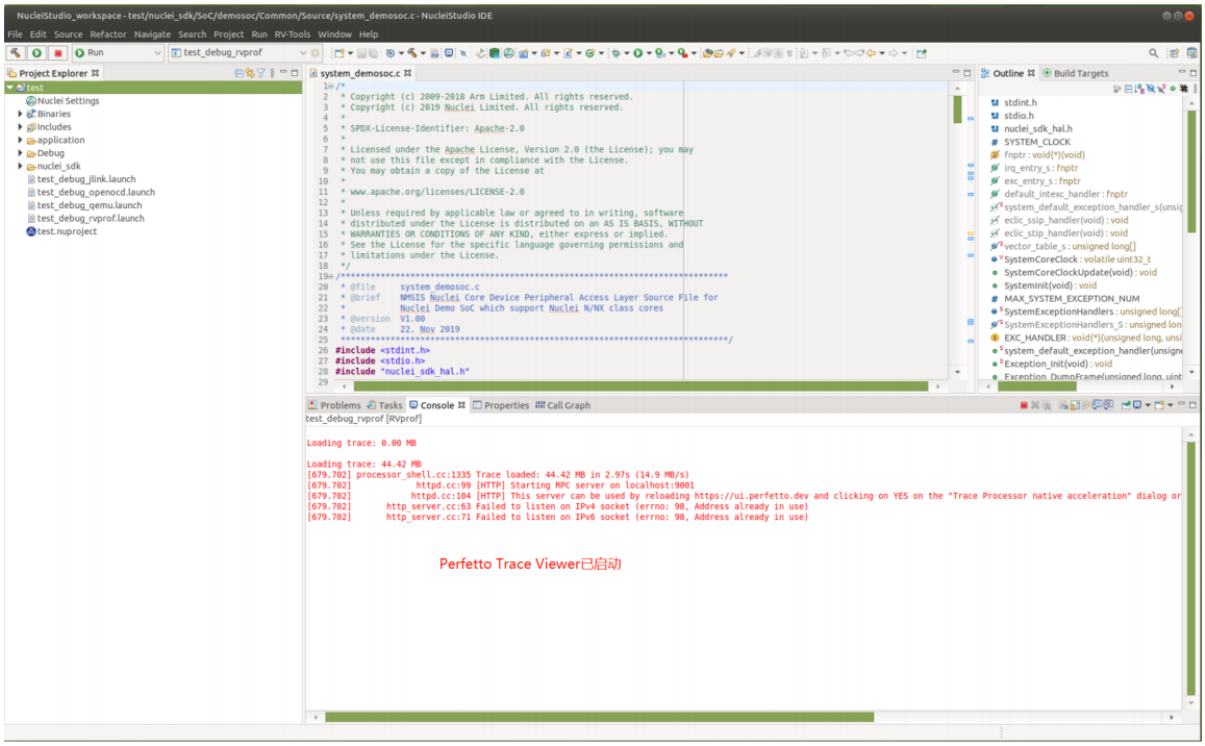
After the project is created, select `test_debug_rvprof.launch` in the Launch Bar and click the run button in the toolbar. Nuclei Studio then completes the following tasks in sequence and displays the final results in Perfetto Trace Viewer.

- Build the project code
- Start Cycle Model and produce the trace file
- Start RVProf to parse the trace file and generate the json file
- Start Perfetto Trace Viewer to display the results

Cycle Model startup and log output

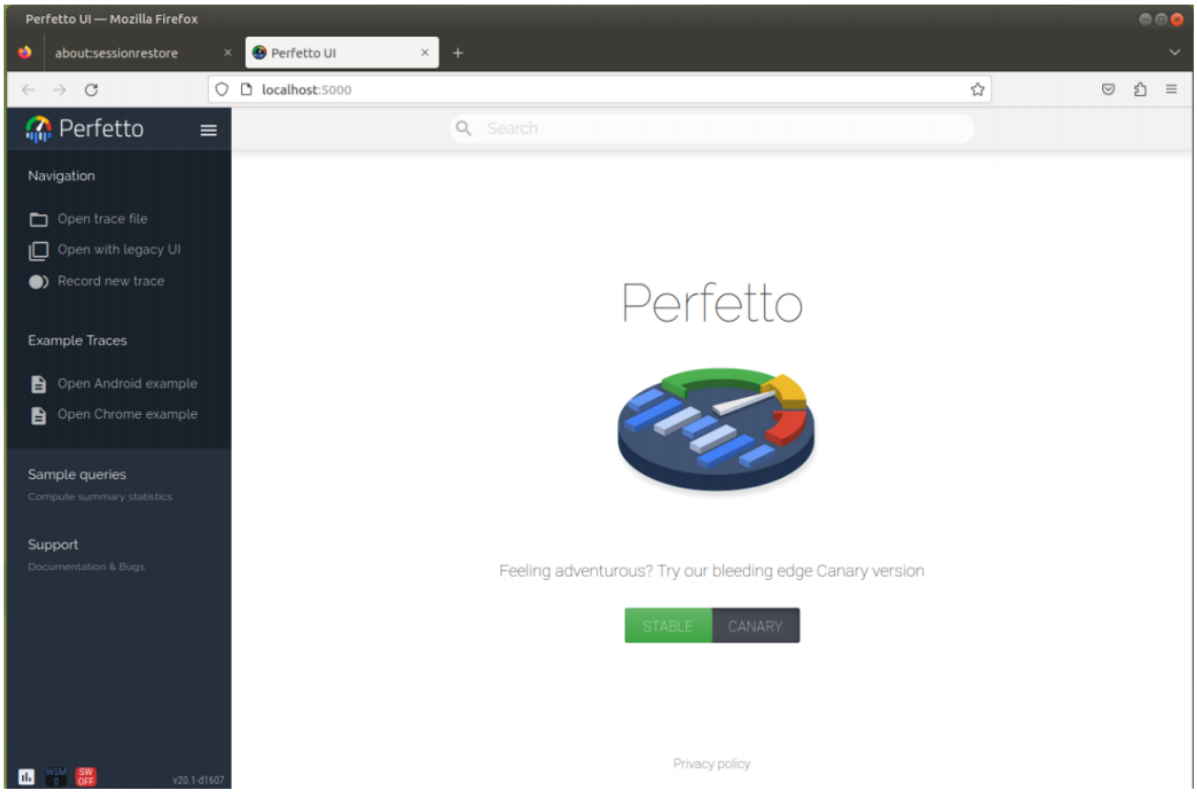


perfetto starting the local service

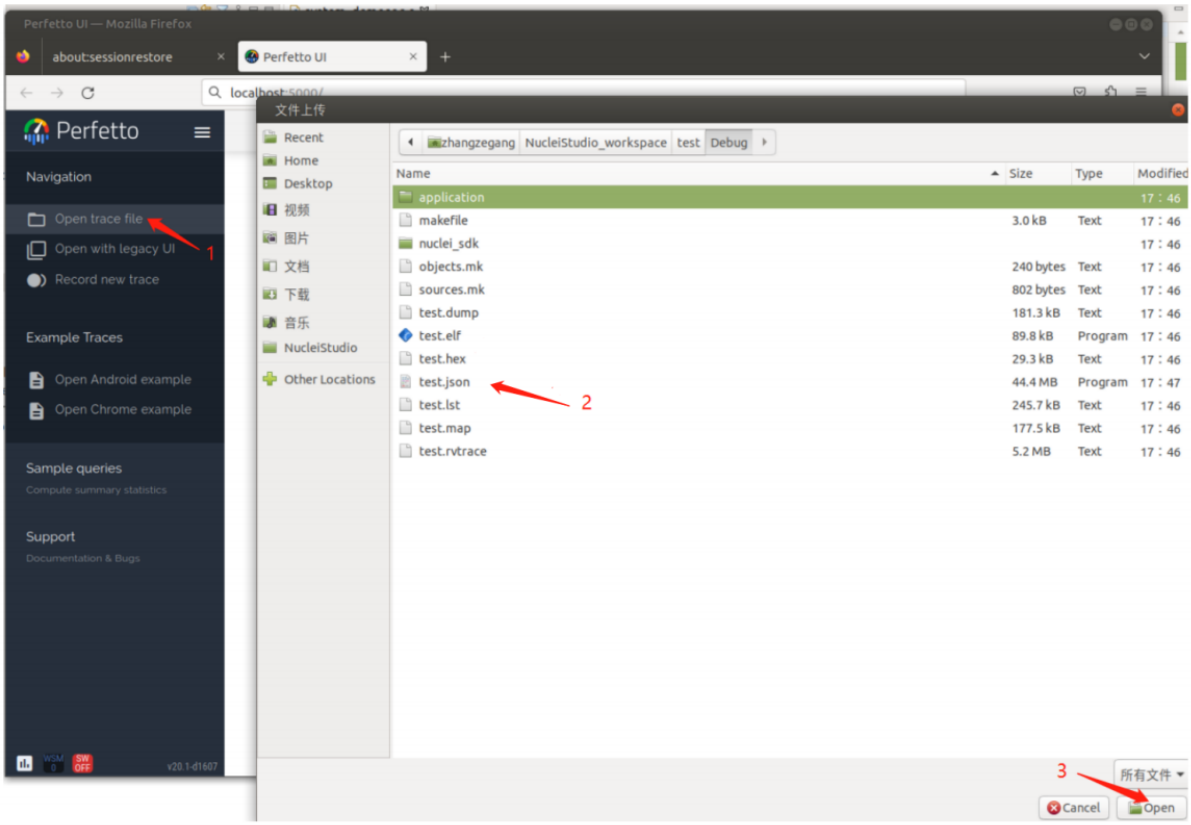


The official address of Perfetto Trace Viewer is https://ui.perfetto.dev/. By default, Nuclei Studio tries to open https://ui.perfetto.dev/ and automatically loads and parses the json file. If the site cannot be opened due to network issues (it is an overseas server), Nuclei Studio starts a local Perfetto Trace Viewer service and automatically opens localhost:5000/. In this case, manually load the `Debug/test.json` file under the project directory. The trace results are then displayed in Perfetto Trace Viewer.

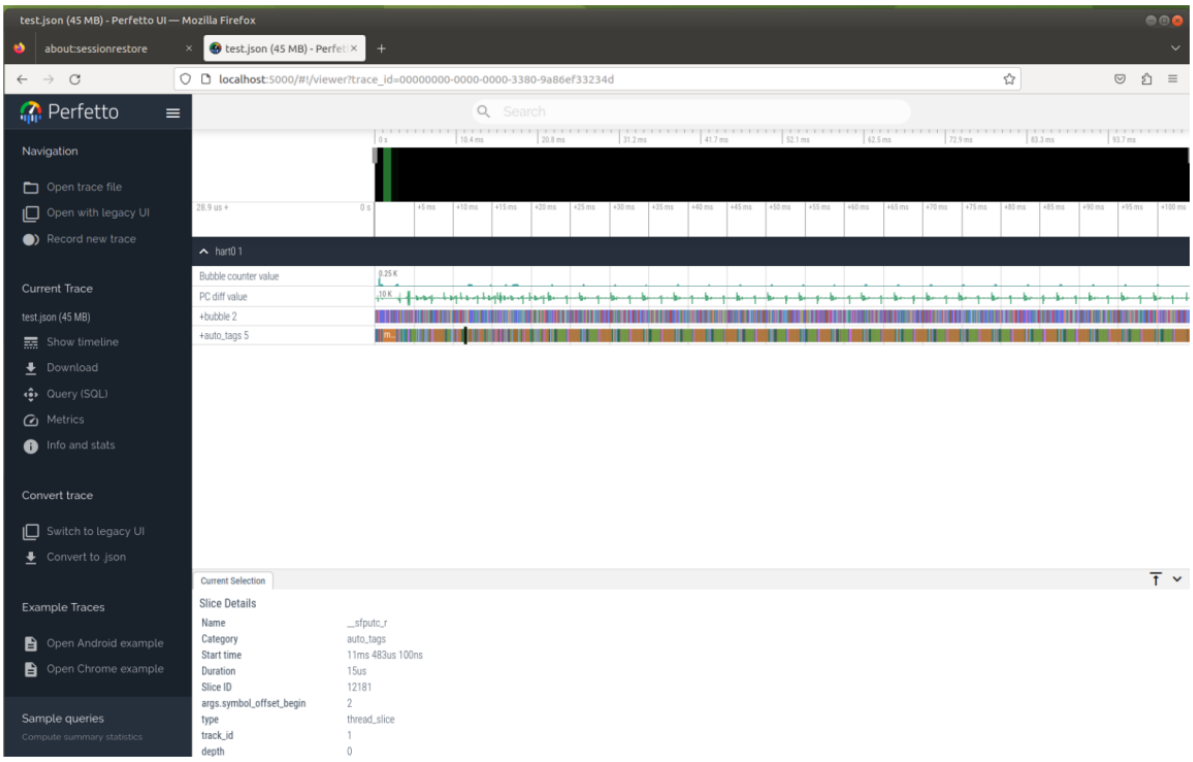
Nuclei Studio starts a web service locally and opens Perfetto Trace Viewer at the same time.



Click Open trace file, locate the json file generated in the project, and load it manually into Perfetto Trace Viewer.



You can now view the RVProf trace results in Perfetto Trace Viewer. Use the `W/A/S/D` keys to navigate and view more detailed information.



2.10.8 Nuclei NICE Wizard

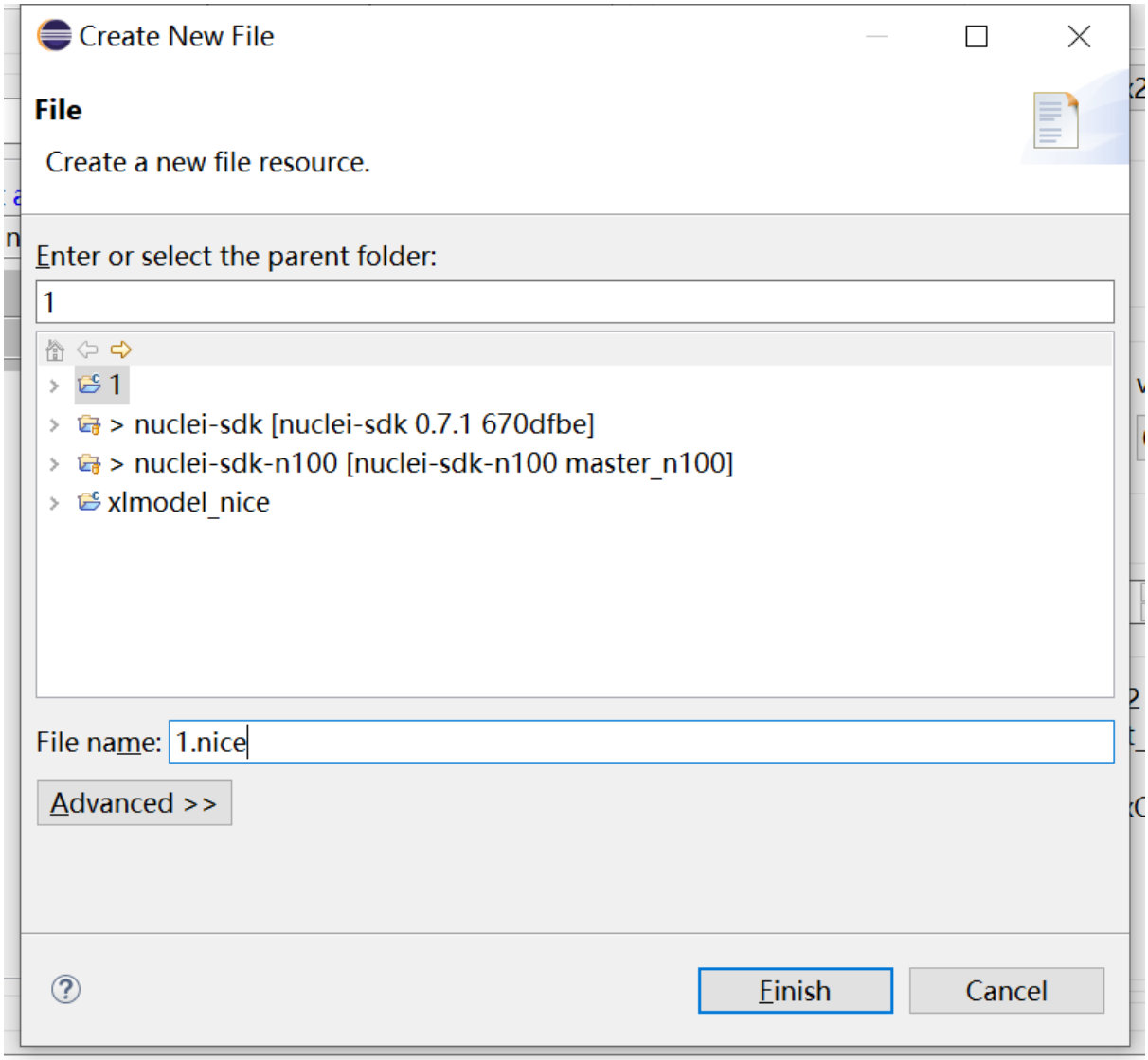
Note: The **Nuclei System Technology WeChat Channel** contains a video on **Nuclei NICE Wizard**. Search for the **Nuclei System Technology WeChat Channel** in WeChat to view it.

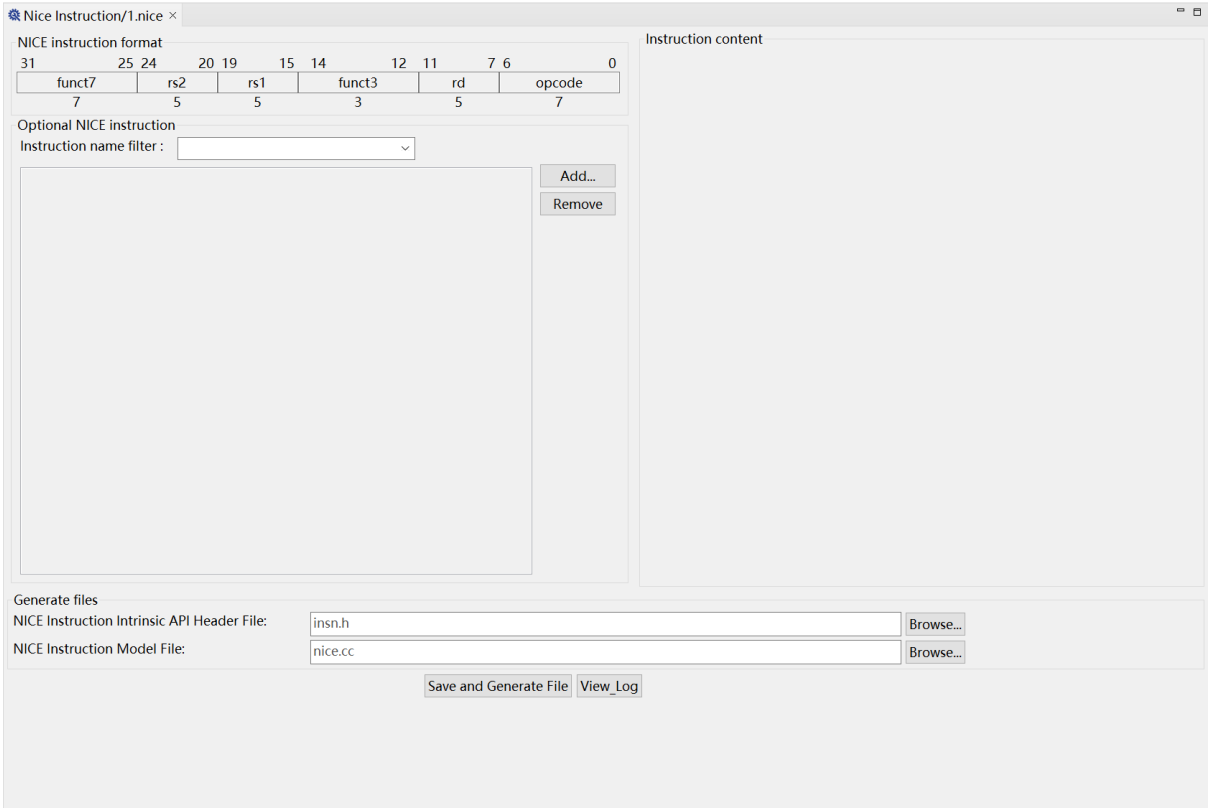
Nuclei NICE Wizard is a tool integrated into Nuclei Studio that simplifies and accelerates the creation of NICE (custom instruction extensions) and VNICE (vectorized custom instruction extensions) instructions. Through a graphical interface, you can quickly configure custom instructions and generate the required code framework, enabling hardware acceleration of specific application algorithms:

- **Simplified development process**: reduces the time from concept to implementation of custom instructions.
- **Improved efficiency**: improves the execution efficiency of applications by generating optimized instruction code.
- **Easy integration**: the generated code can be integrated directly into existing projects.

Creating a .nice File and Opening Nuclei NICE Wizard

Open the target project in Nuclei Studio, create a `*.nice` file (for example aicc.nice) in the project root directory, and double-click it to open Nuclei NICE Wizard.

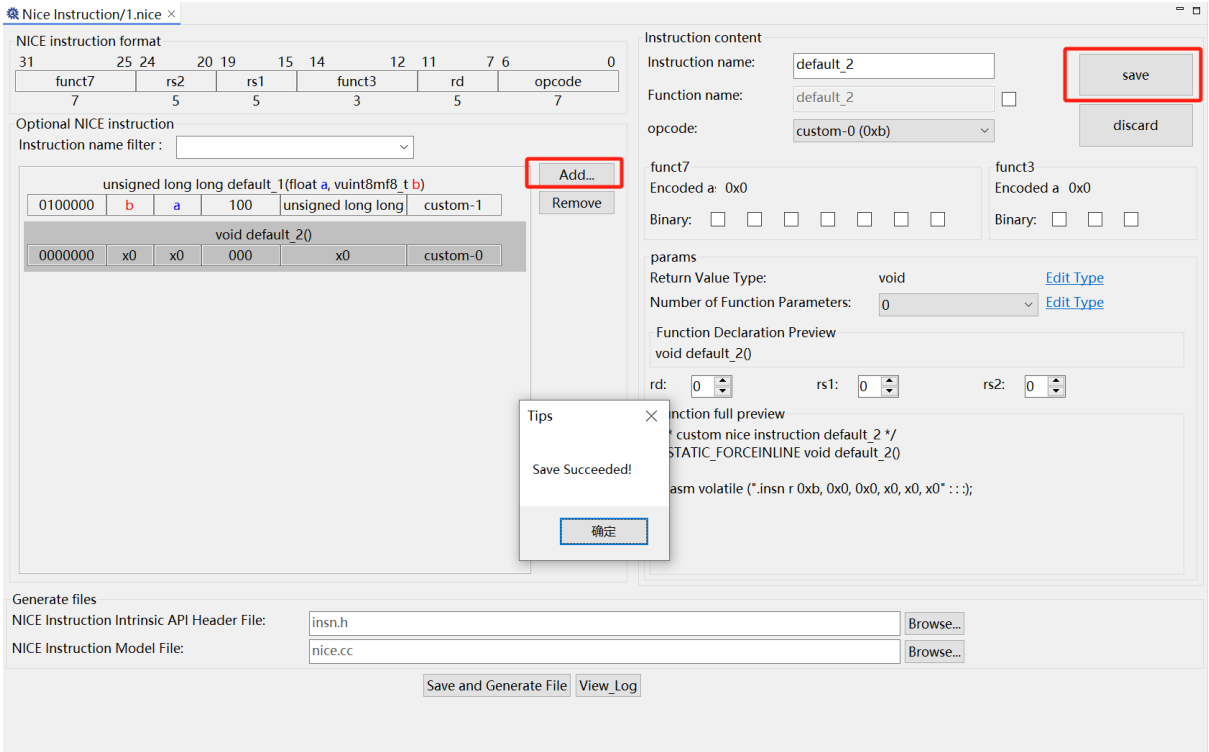




Adding an Instruction

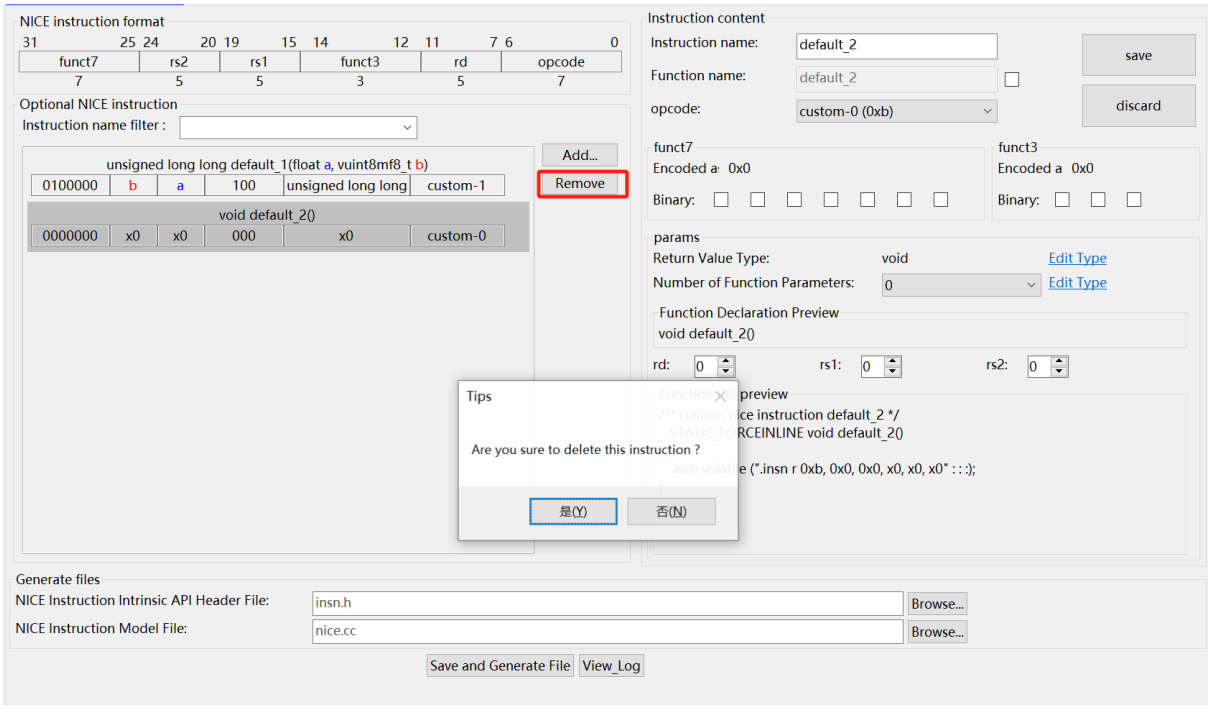
Click `Add...`, modify the instruction content as needed, and click `save` in the upper right corner.

The following example creates two instructions. The instruction selected on the left turns gray, and its content is displayed on the right.



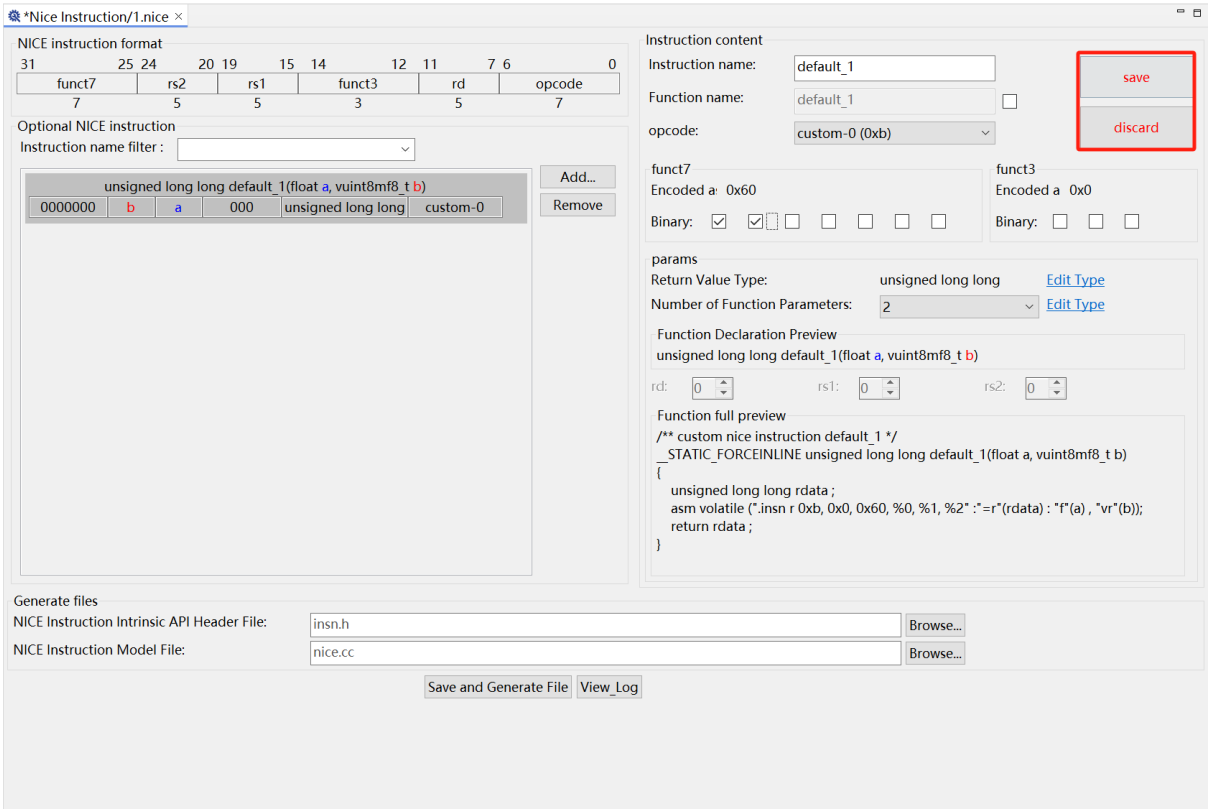
Deleting an Instruction

Select the instruction on the left, click Remove, and confirm the deletion.



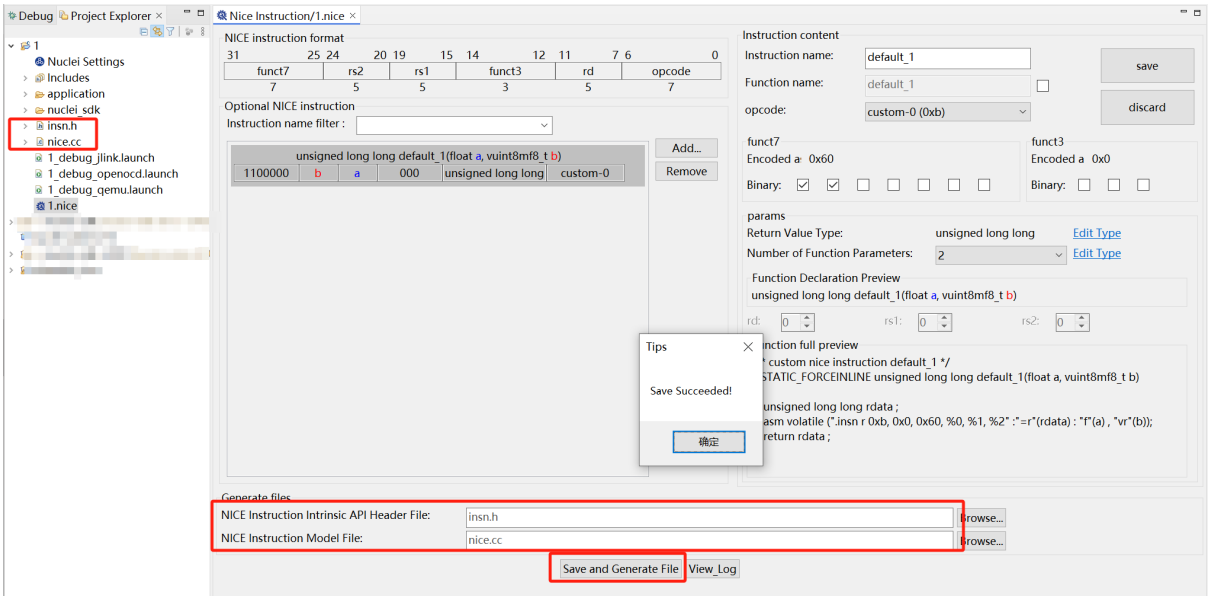
Modifying an Instruction

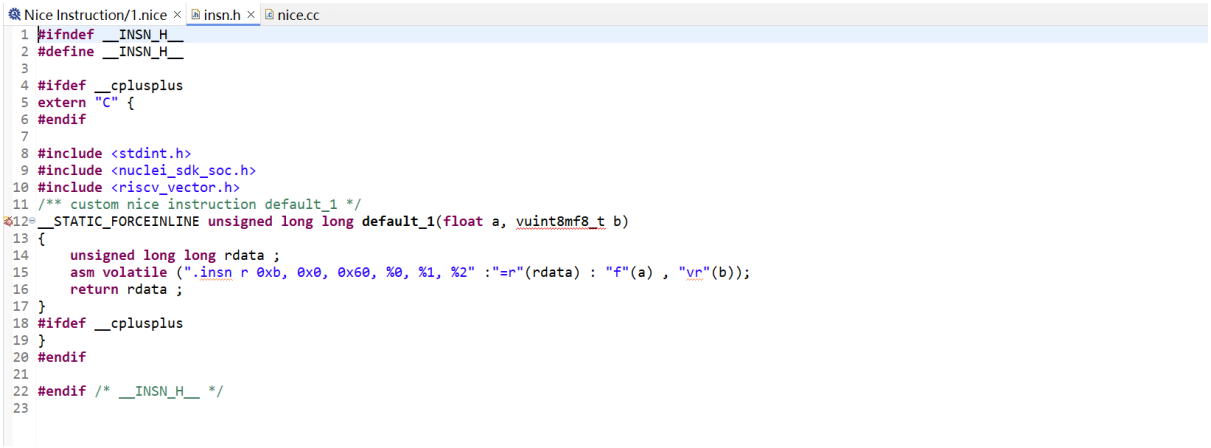
Select the instruction on the left. After you modify the instruction content, the save and discard buttons in the upper right corner turn red; click them to save or discard the changes.



File Generation

Define the save locations of the insn.h file (the inline assembly header) and the nice.cc file (the instruction implementation logic), then click Save and Generate File to generate the files.

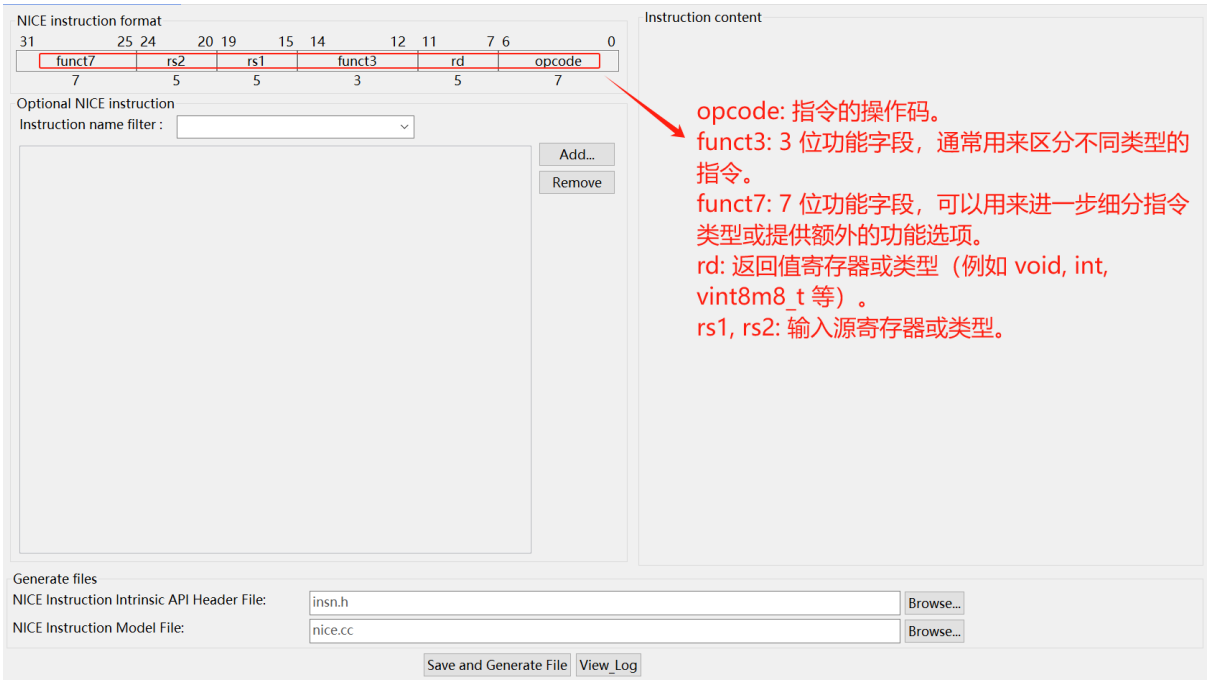




```
Nice Instruction/1.nice × insn.h × nice.cc
 1 #ifndef __INSN_H__
 2 #define __INSN_H__
 3
 4 #ifdef __cplusplus
 5 extern "C" {
 6 #endif
 7
 8 #include <stdint.h>
 9 #include <nuclei_sdk_soc.h>
10 #include <riscv_vector.h>
11 /** custom nice instruction default_1 */
12 __STATIC_FORCEINLINE unsigned long long default_1(float a, vuint8mf8_t b)
13 {
14     unsigned long long rdata ;
15     asm volatile (".insn r 0xb, 0x0, 0x60, %0, %1, %2" :"=r"(rdata) : "f"(a) , "vr"(b));
16     return rdata ;
17 }
18 #ifdef __cplusplus
19 }
20 #endif
21
22 #endif /* __INSN_H__ */
23
```

```
Nice Instruction/1.nice insn.h nice.cc ×
 1 #include "nice.h"
 2
 3 /* The macros NUCLEI_NICE_SCALAR and NUCLEI_NICE_VECTOR respectively represent the scalar and Vector-NICE instructions implemented by Nuclei.
 4 If you need to implement your own NICE instructions, you can comment out these two macros.*/
 5 #define NUCLEI_NICE_SCALAR
 6 #define NUCLEI_NICE_VECTOR
 7
 8
 9 void do_nice(processor_t* p, insn_t insn, reg_t pc) {
10     (void)pc;
11     uint32_t instr = insn.bits();
12     uint32_t opcode = instr & 0x7f;
13     uint32_t funct7 = (instr >> 25) & 0x7f;
14     uint32_t funct3 = (instr >> 12) & 0x7;
15     uint32_t rd = (instr >> 7) & 0x1f;
16     uint32_t rs1 = (instr >> 15) & 0x1f;
17     uint32_t rs2 = (instr >> 20) & 0x1f;
18     if (opcode == 0xb && funct3 == 0x0 && funct7 == 0x60) {
19         /* Implement default_1 here */
20         /* Modify default_1 cycle here, default is 1 */
21         STATE.mcycle->bump(1);
22     }
23 }
24
```

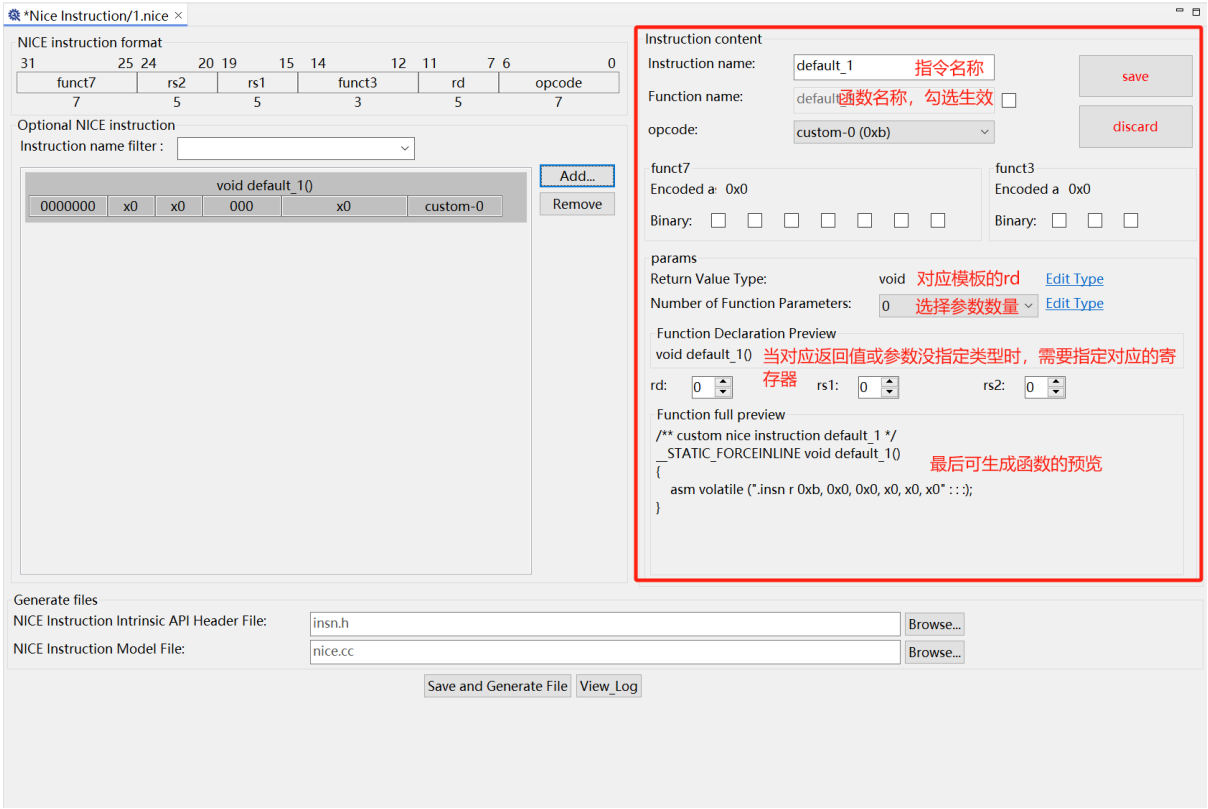
NICE Instruction Template Description



A single instruction template is shown in the figure above:

- opcode: custom-0, custom-1, custom-2, or custom-3
- funct3: 3-bit function field, usually used to distinguish between different types of instructions.
- funct7: 7-bit function field, which can be used to further subdivide instruction types or provide additional function options.
- rd: return value register or type (for example void, int, vint8m8_t, etc.).
- rs1, rs2: input source registers or types.

Instruction Content Editing Description



As shown above, the Instruction content area displays the default content.

- **Instruction name**: the instruction name. The naming rules are as follows:
  - **Letters and digits**: the function name can contain letters `(A-Z, a-z)` and digits `(0-9)`, but cannot start with a digit.
  - **Underscore**: underscores _ can be used in the function name to improve readability, especially in multi-word combinations. For example, `get_user_name` is a valid function name; `< , > , ... , ? , /` are not allowed in a function name.
  - **Special characters**: except for the underscore, other special characters such as `! , @ , # , $ , % , ^ , & , * , ( , ) , { , } , [ , ] , \ , : , ;` are not allowed.
  - **Keywords**: the function name cannot be a C language keyword or reserved word, such as `int , char , float , double , if , else , while , for , return`.
- **Function name**: the function name. When not checked, the generated function name is the instruction name. The naming rules are the same as for `Instruction name`.
- **funct7**: corresponds to the `funct7` of the template, and can be set by checking the corresponding Binary items.
- **funct3**: corresponds to the `funct3` of the template, and can be set by checking the corresponding Binary items.
- **Return Value Type**: corresponds to the rd of the template. Click Edit Type to set it, for example to void.
- **Number of Function Parameters**: the number of parameters. You can set the types of the input parameters rs1, rs2, and rs3 (rs3 serves as both a parameter and a return value).
  - When the number of parameters is 0, Edit Type cannot be set. rs1 and rs2 can be assigned registers below; if rd is of void type, rd can also be assigned a register below.
  - When the number of parameters is 1, Edit Type can set the rs1 type. rs2 can be assigned a register below; if rd is of void type, rd can also be assigned a register below.

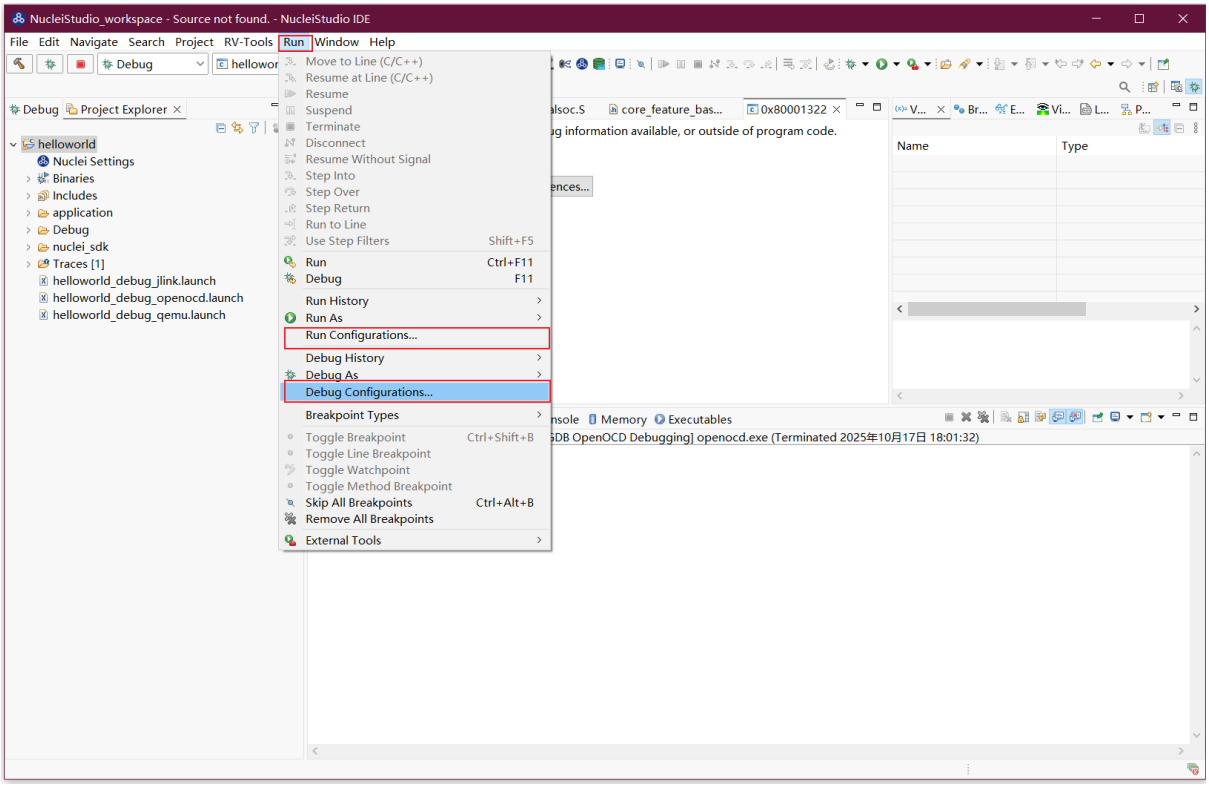
- When the number of parameters is 2, Edit Type can set the rs1 and rs2 types. If rd is of void type, rd can also be assigned a register below.
- When the number of parameters is 3, Edit Type can set the rs1, rs2, and rs3 types.

2.10.9 Using Nuclei Model Debugger

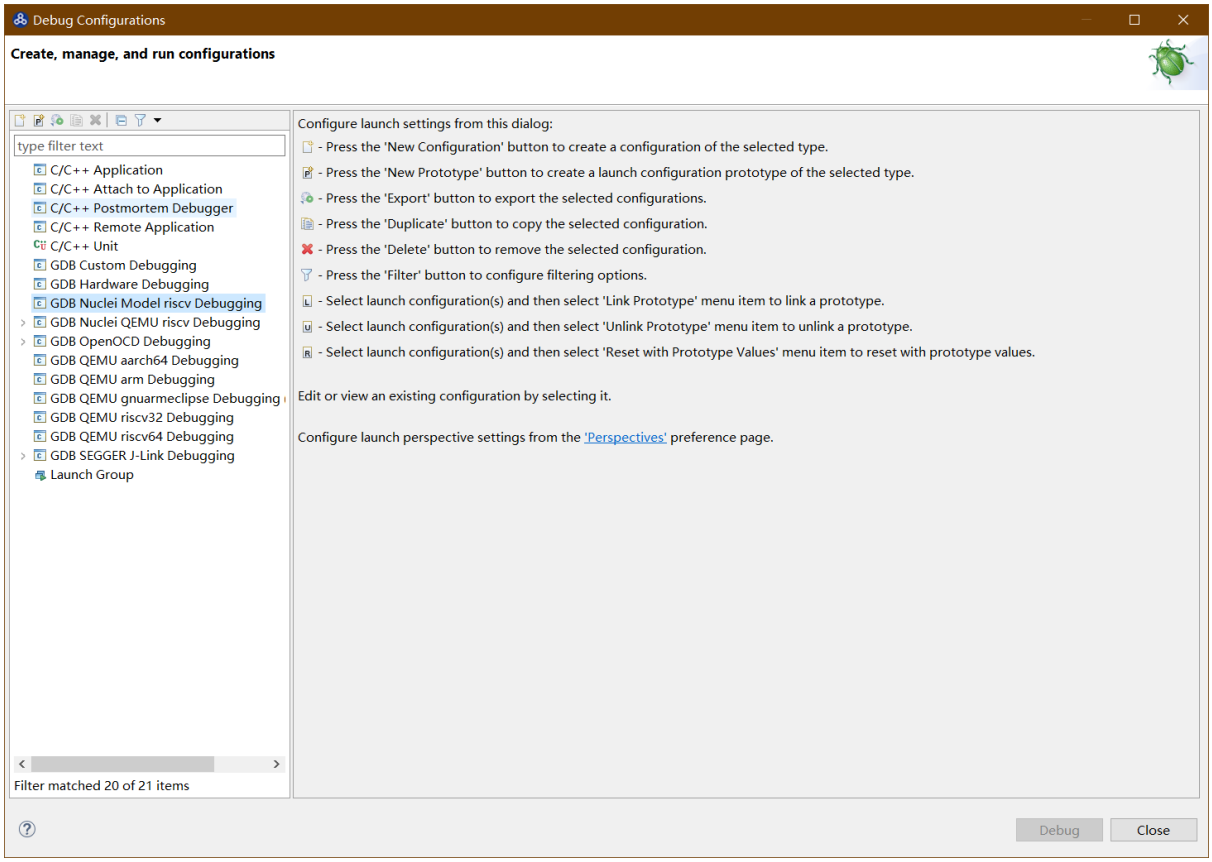
Nuclei Model Debugger is a debugging and running tool developed for the upgraded Nuclei Near Cycle Model.

Starting with Nuclei Studio 2025.10, Nuclei Near Cycle Model provides richer features: it can debug and run programs and accepts custom parameters. To make these features easier to use, the Nuclei Model Debugger was developed based on the QEMU Debugger feature.

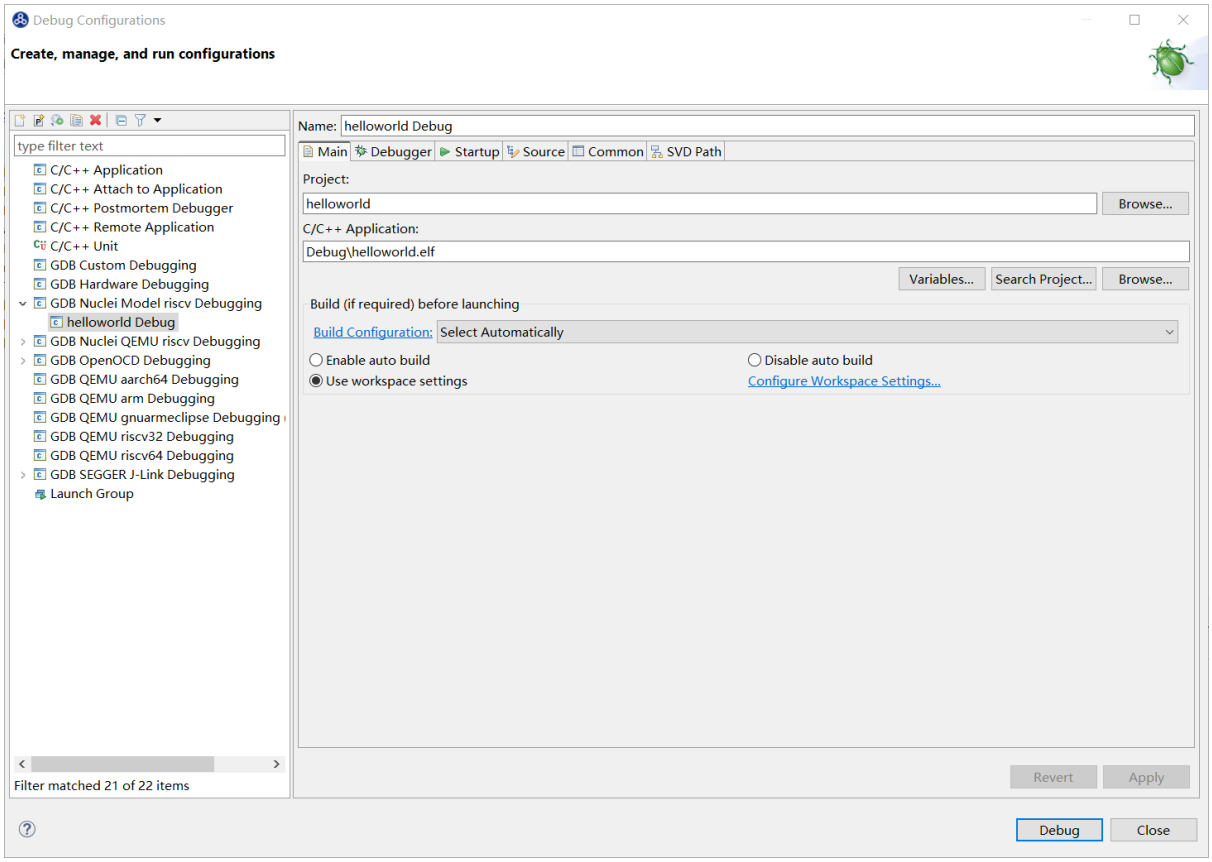
Create a test project and build it. In the Nuclei Studio main menu, click `Run`, then `Run Configurations` or `Debug Configurations`. This example uses Debug: click `Debug Configurations`.



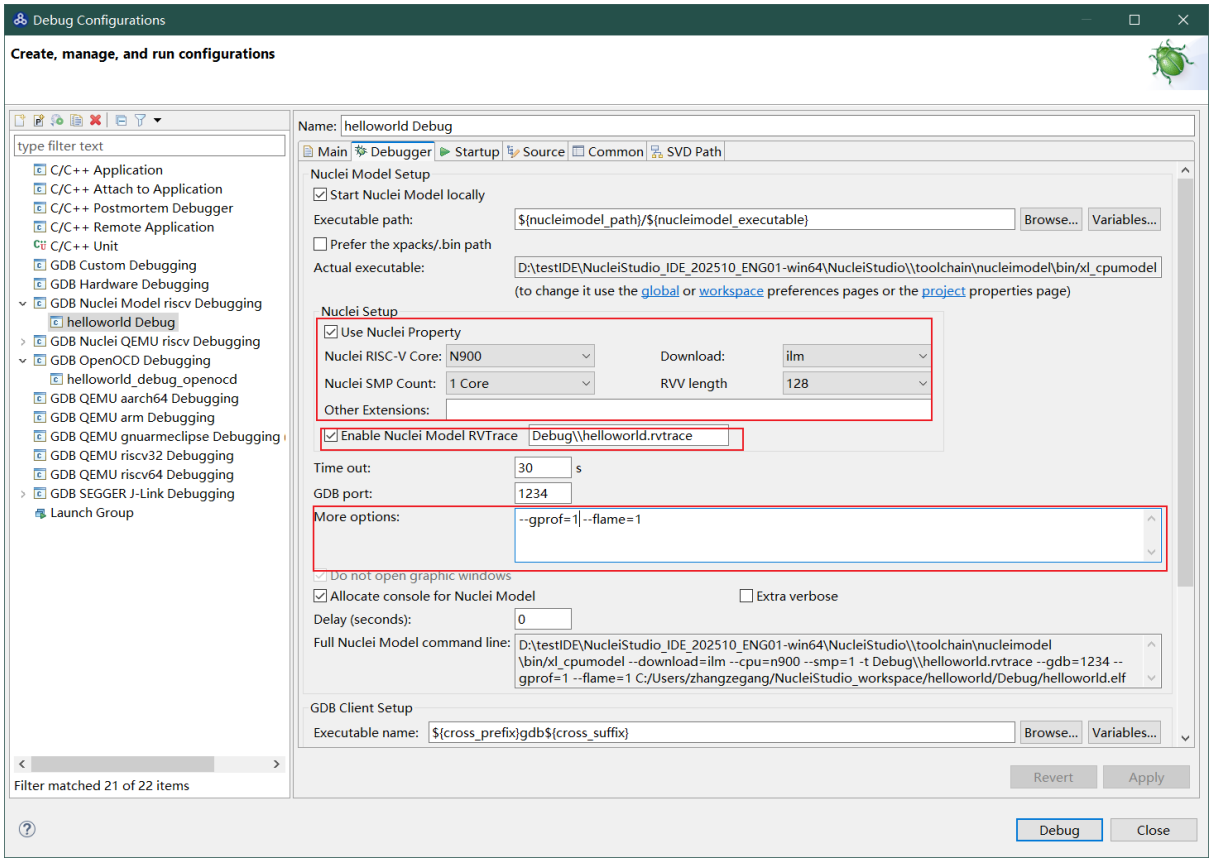
In the `Debug Configurations` dialog, find `GDB Nuclei Model riscv Debugging`.



Double-click `GDB Nuclei Model riscv Debugging`, and Nuclei Studio creates a new Nuclei Model Debugger configuration page where you can configure the command parameters for Model Debug.



The values configured on the `Debugger` page must match the project.



- **Nuclei RISC-V Core** The CPU core, which must match the program
- **Download** Program download mode
- **Nuclei SMP Count CPU** Number of cores
- **RVV length** RISC-V vector extension
- **Other Extensions** Other extensions

If `Enable Nuclei Model RVTrace` is checked and its path is configured, a `*.rvtrace` file is generated when the program runs.

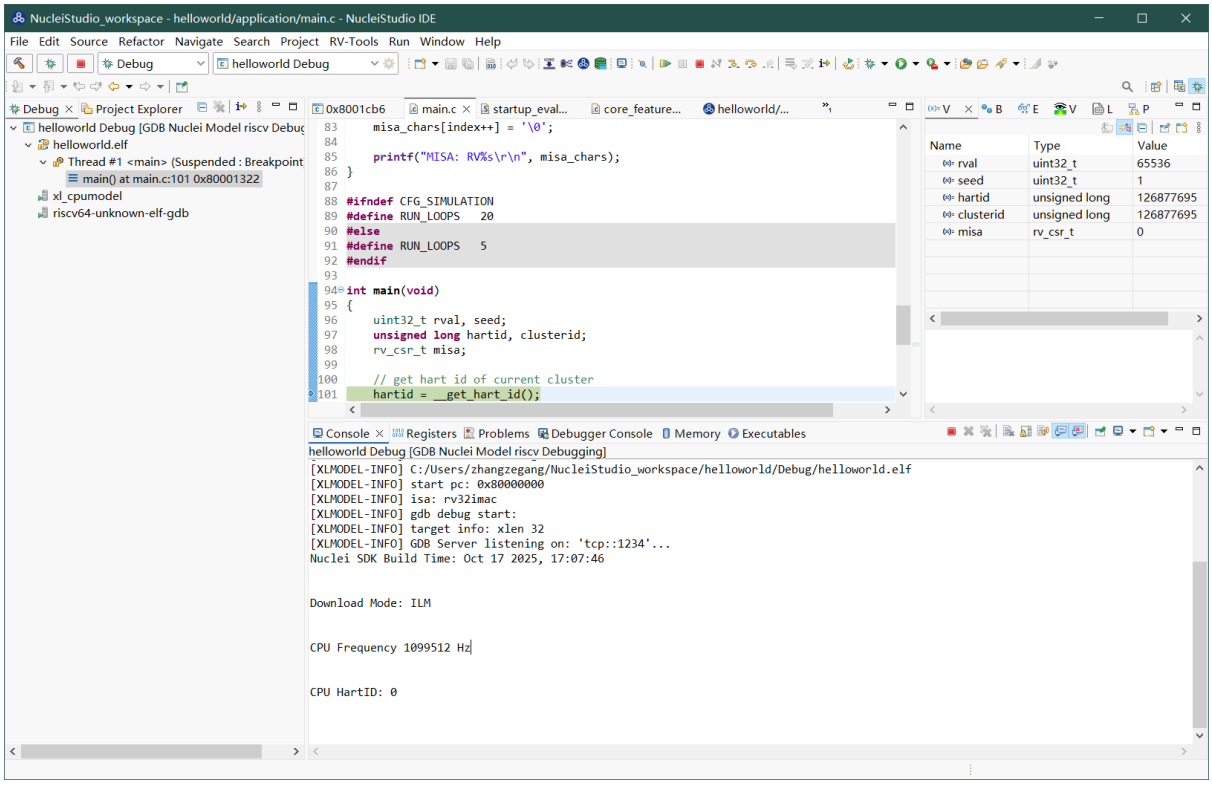
In `More options`, you can add parameters supported by Nuclei Model. In this example, `--gprof=1 --flame=1` is configured in Config options: `--gprof=1` enables the gprof-based Profiling feature, so a gprof file is produced when the program runs, and `--flame=1` enables the Flamegraph feature.

For details about the parameters of Nuclei Near Cycle Model, see *Description of Parameters* (page 866).

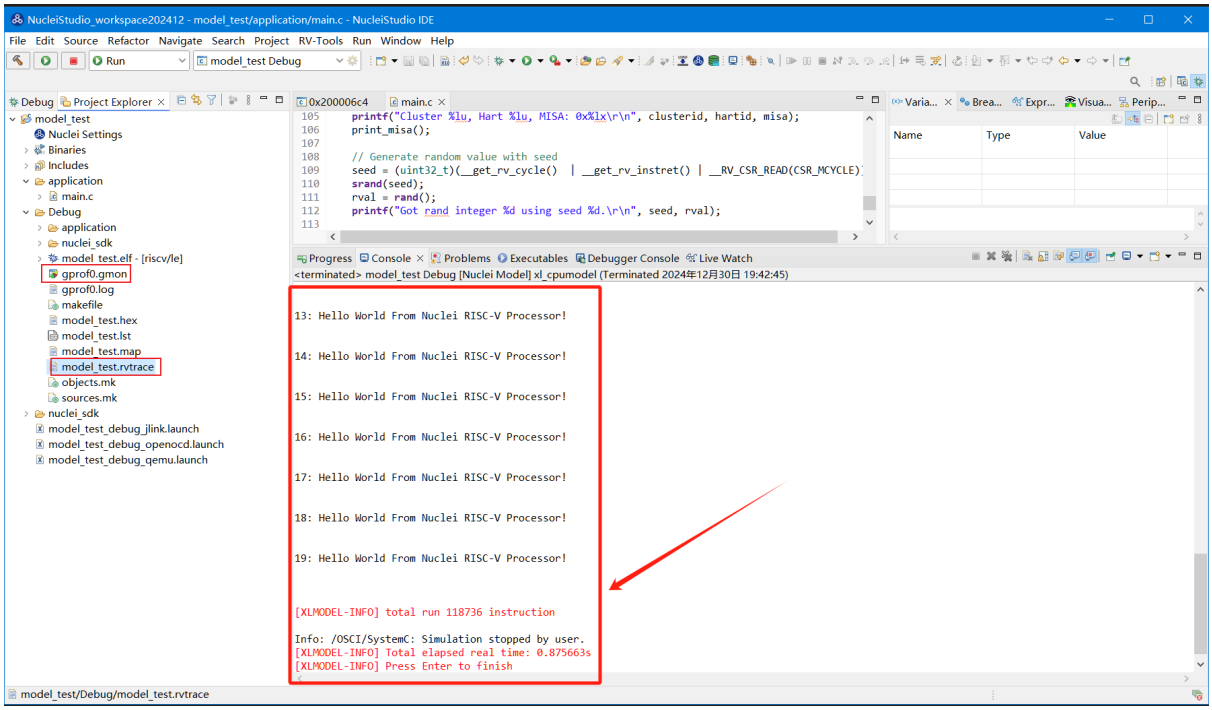
For the gprof Profiling feature, see *Code Coverage and Profiling* (page 229).

For the Flamegraph feature, see *Using Flame View* (page 272).

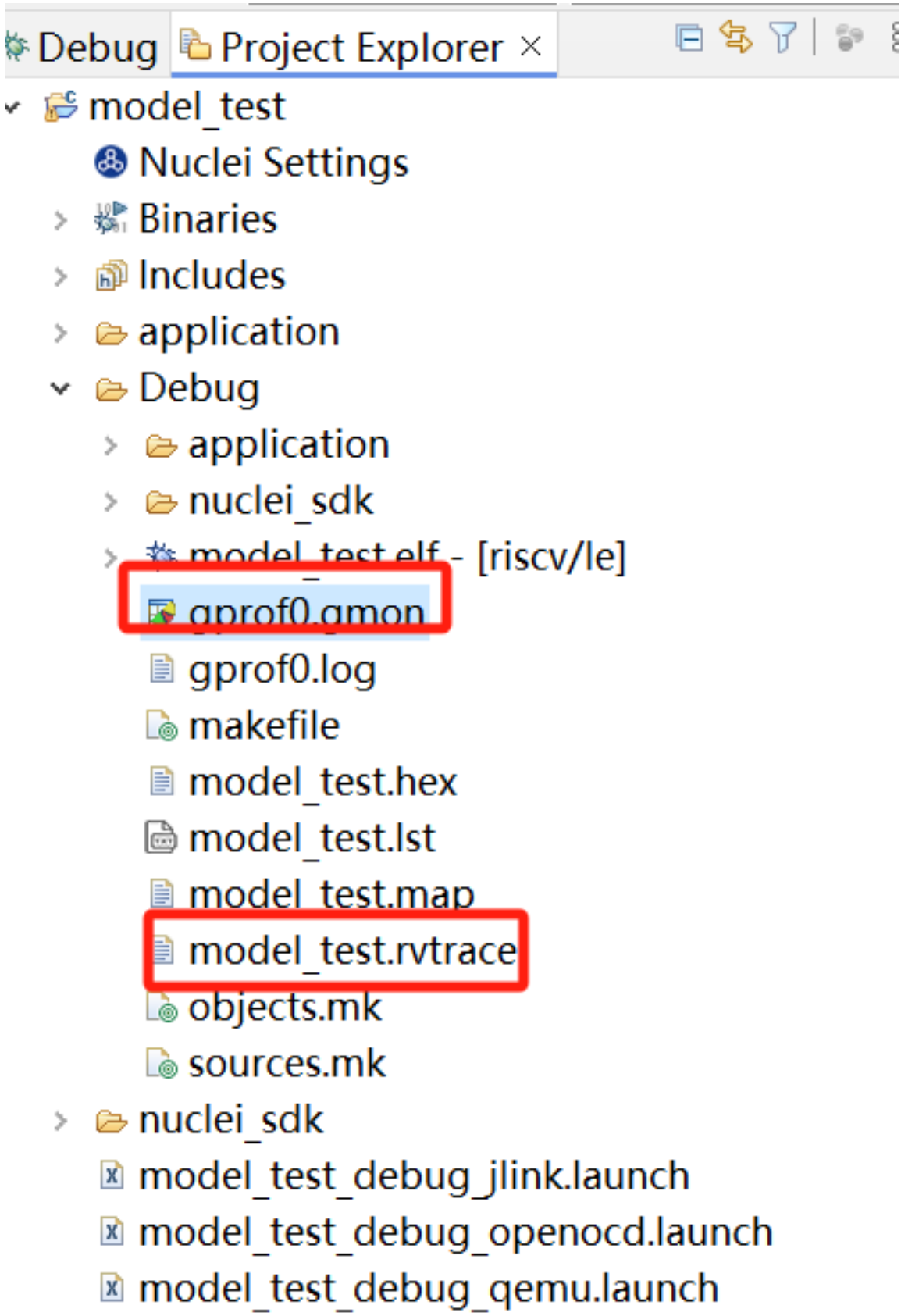
After configuring and saving the parameters, click `Debug` to enter the Nuclei Model Debug mode.



The program runs successfully in Nuclei Near Cycle Model and outputs the corresponding log information.



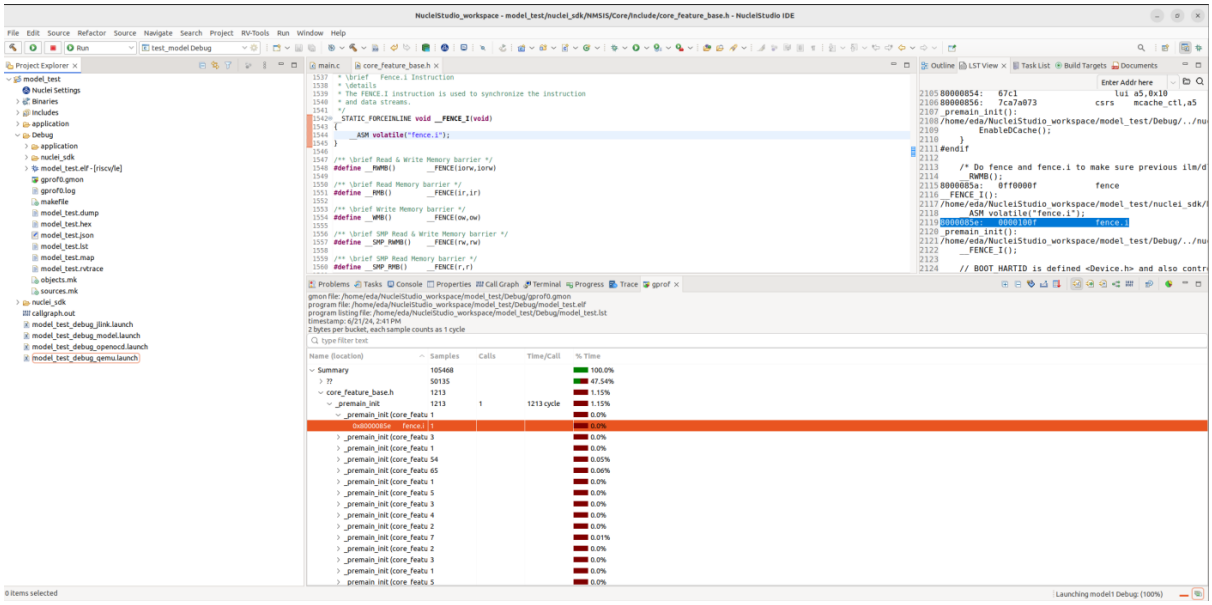
In the Debug directory of the project, you can see that the `.rvtrace` file and the `.gmon` file have been generated.



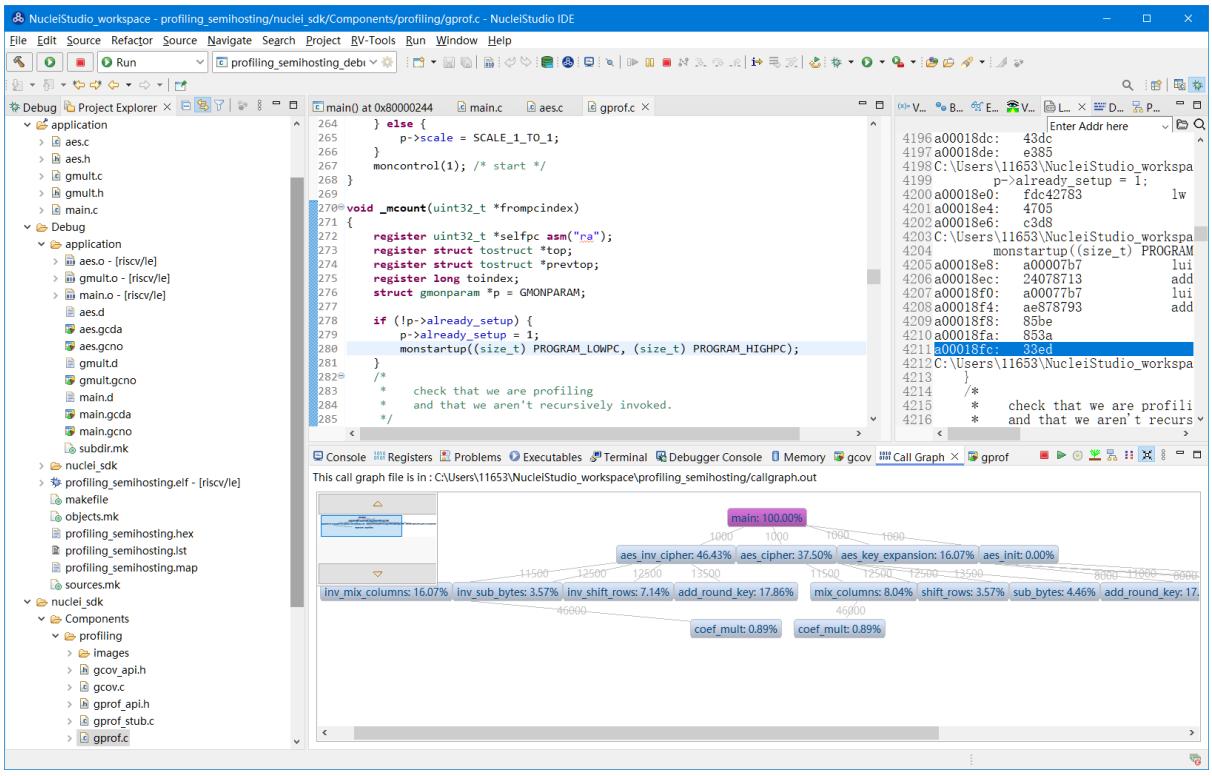
Nuclei Near Cycle Model supports analyzing programs with gprof. Therefore, when `--gprof` is configured, a `.gmon` file is also produced in the Debug directory (the directory configured with `--logdir=XX`) while the program runs. Double-click the `.gmon` file to invoke the gprof tool and analyze the number of cycles consumed by program execution and the call relationships; a corresponding `callgraph.out` file is also produced. Double-click

the `callgraph.out` file to view the program ' s call relationships with Call Graph.

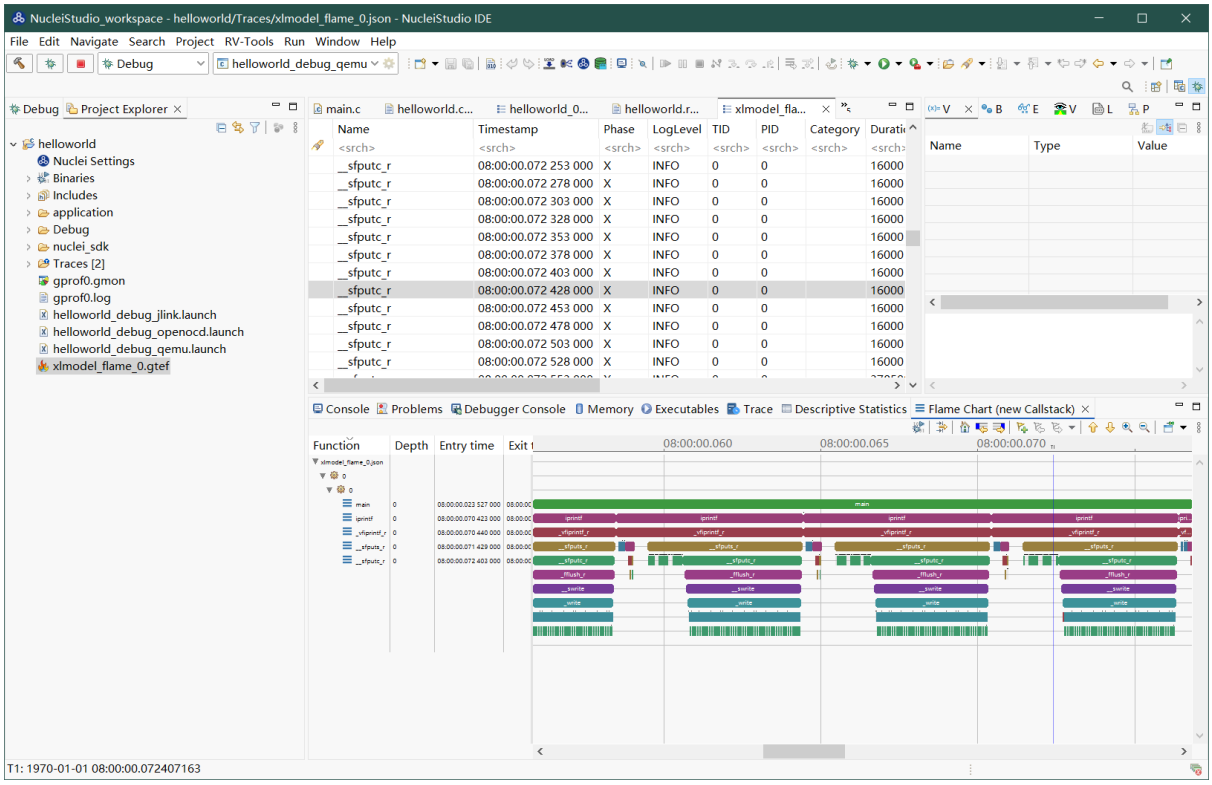
The gprof tool displays the contents of the generated `.gmon` file.



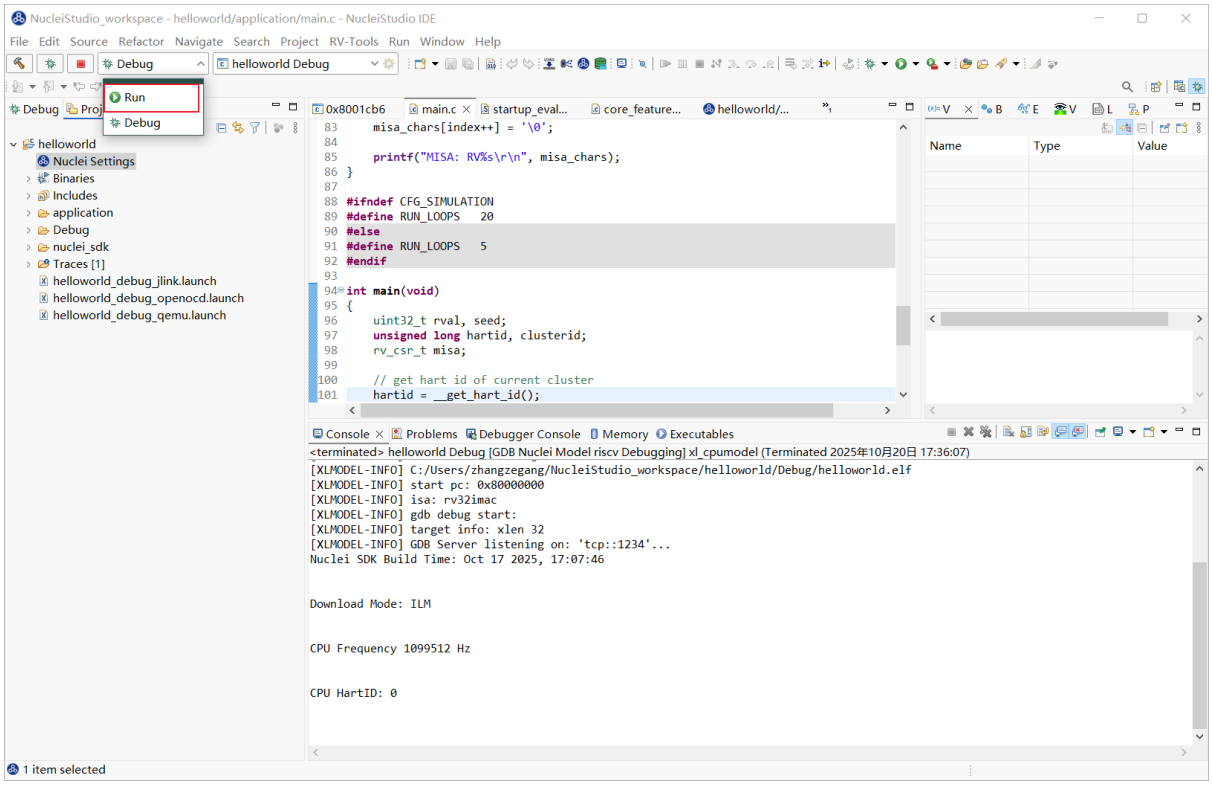
While viewing the `.gmon` file, the gprof tool parses the program ' s call relationships based on its contents and generates a `callgraph.out` file. Double-click `callgraph.out` to view it with the Call Graph tool.



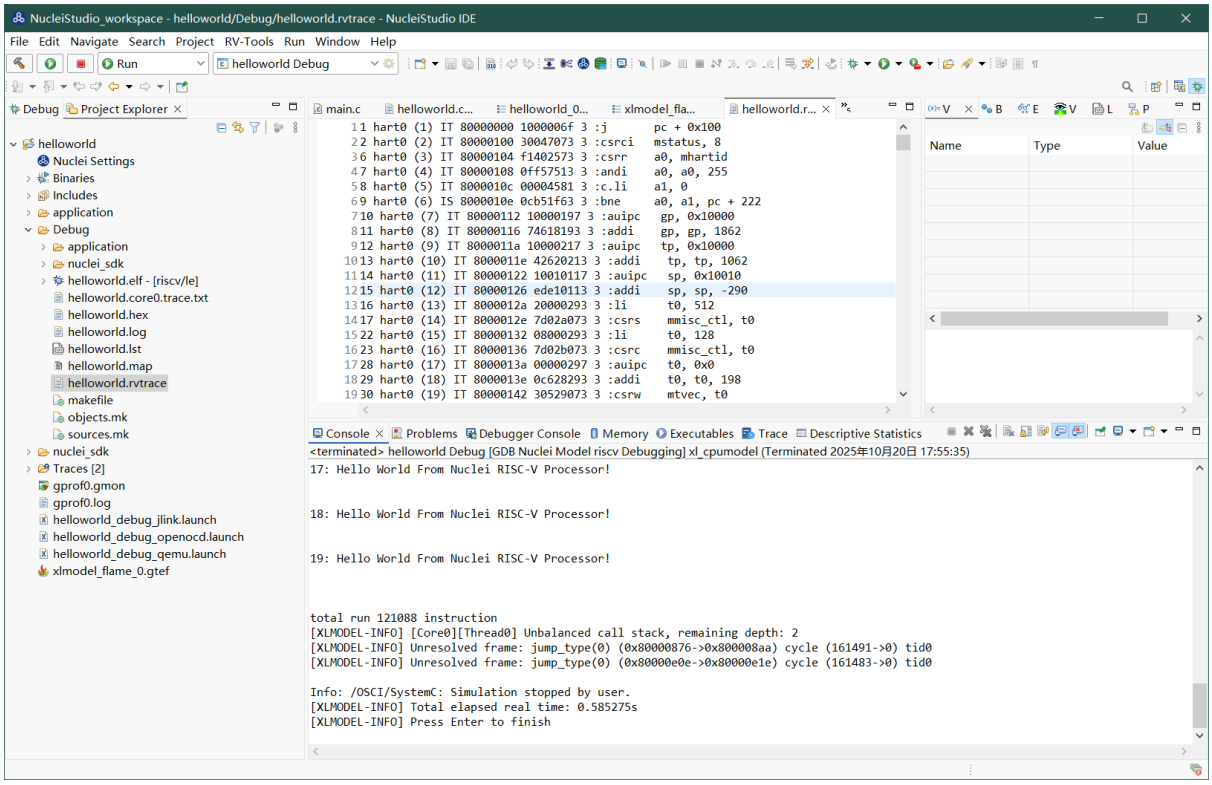
Because `--flame=1` is configured, an `xlmodel_flame_0.gtef` file is produced in the project root directory. Double-click the file to invoke the Flame View tool to parse it and generate a flame graph.



The same configuration supports both Debug and Run. To execute the program directly, switch in the Launch Bar.

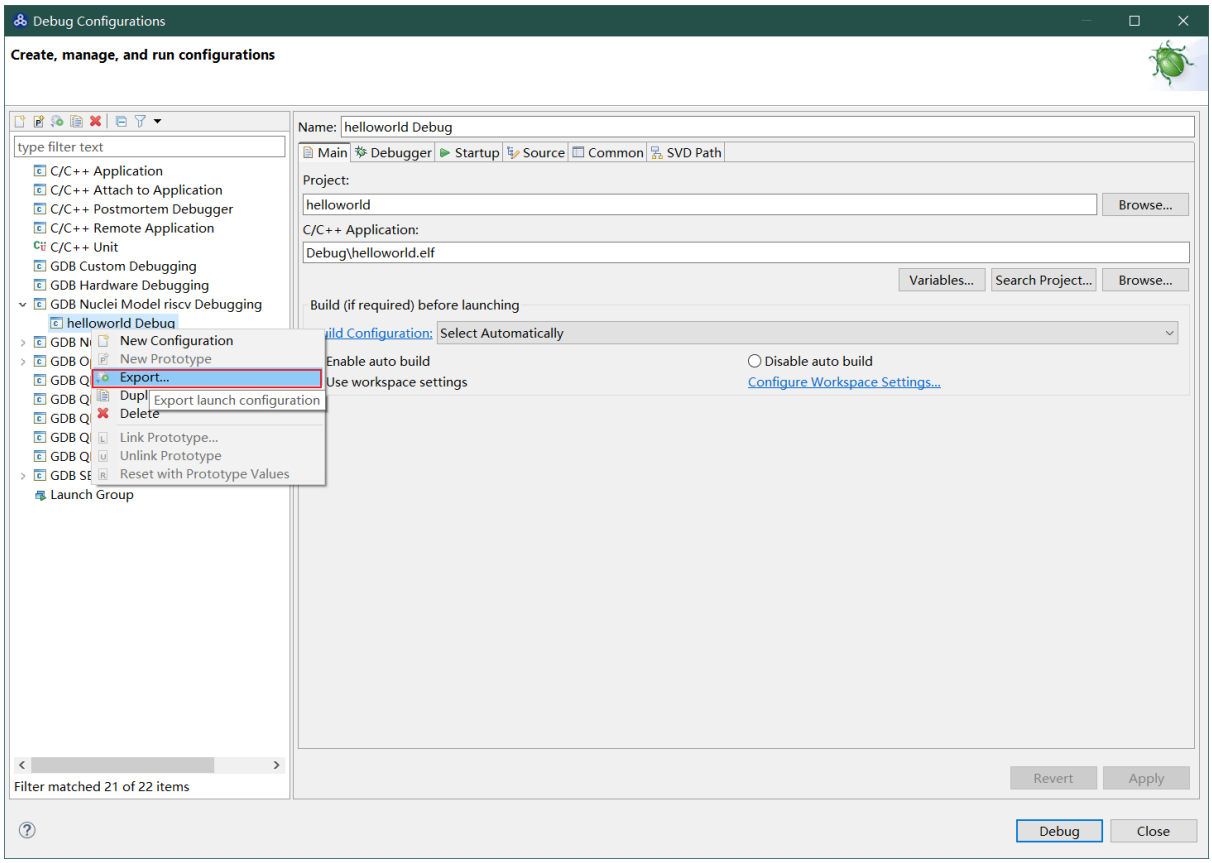


Click Run to enter Nuclei Model Run mode; the program executes directly to completion.

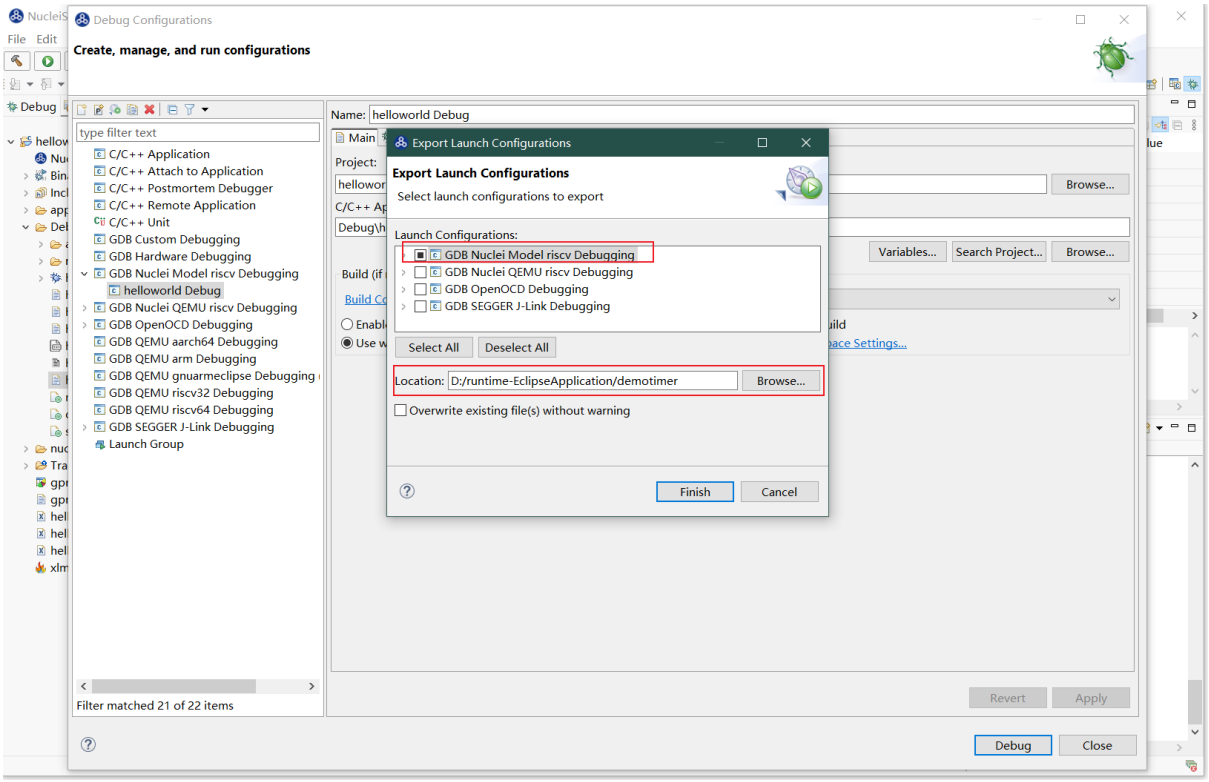


You can also export the configuration file to the project root directory so that it can be shared with other users.

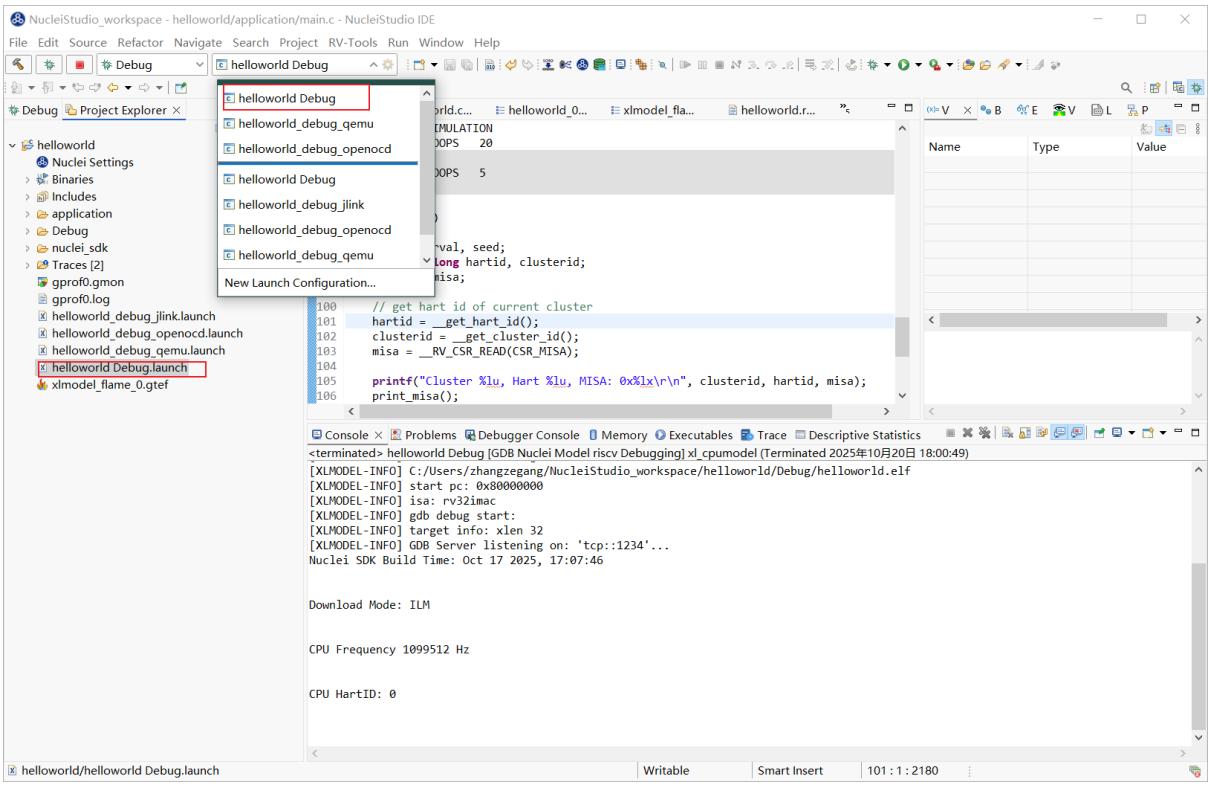
Open the Debug Configurations page, right-click the configuration just created, and select ``Export…`` from the context menu.



In the `Export Launch Configurations` page, select the configuration to export and the export location, and click `Finish` to complete the export.



Refresh the project. A `*.launch` file exists under the project, and the corresponding configuration also appears in the Launch Bar. You can now use this configuration for Debug/Run operations on the program.



2.10.10 Using Nuclei Model

Nuclei has developed a dedicated running tool for Nuclei Near Cycle Model called Model. In Nuclei Studio 2024.06, Nuclei Near Cycle Model was originally run through the RVProf tool. As Nuclei Near Cycle Model continued to evolve, we simplified the RVProf-based functionality and launched the new Model tool to provide a simpler, more efficient user experience.

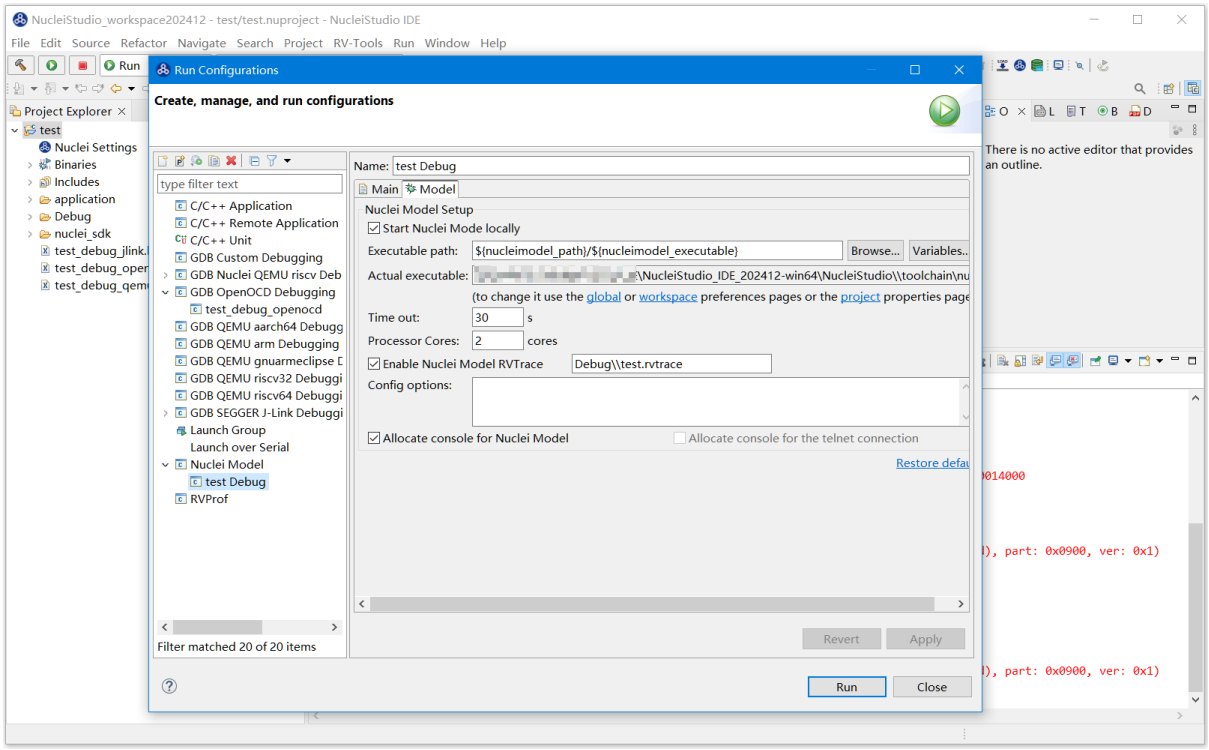
The main features of the new tool include:

Simplified functionality: removes unnecessary complex features so you can focus on the core functionality of Nuclei Near Cycle Model.

Improved efficiency: optimizes the operation flow and reduces configuration time.

Good compatibility: ensures seamless integration with existing workflows while supporting the latest Nuclei Near Cycle Model features.

With these improvements, you can use Nuclei Near Cycle Model for development and debugging more efficiently. Open Run Configuration via the Nuclei Studio menu `Run -> Run Configuration`, find `Nuclei Model`, and double-click it to generate the configuration for the project.



The use of Nuclei Model is described in detail in the Nuclei Near Cycle Model chapter.

2.10.11 Nuclei Near Cycle Model

Nuclei Studio 2024.06 integrates Nuclei Near Cycle Model, a simulation testing and performance analysis tool independently developed by Nuclei. It helps R&D personnel perform simulation testing and program performance analysis in the early stages of a project.

Nuclei Near Cycle Model is only available for Linux in Nuclei Studio 2024.06; Windows support was added in version 2025.02. For a detailed introduction and command line usage, see https://doc.nucleisys.com/nuclei_tools/xlmodel/intro.html. The following sections demonstrate how to use Nuclei Near Cycle Model for simulation and performance analysis in Nuclei Studio.

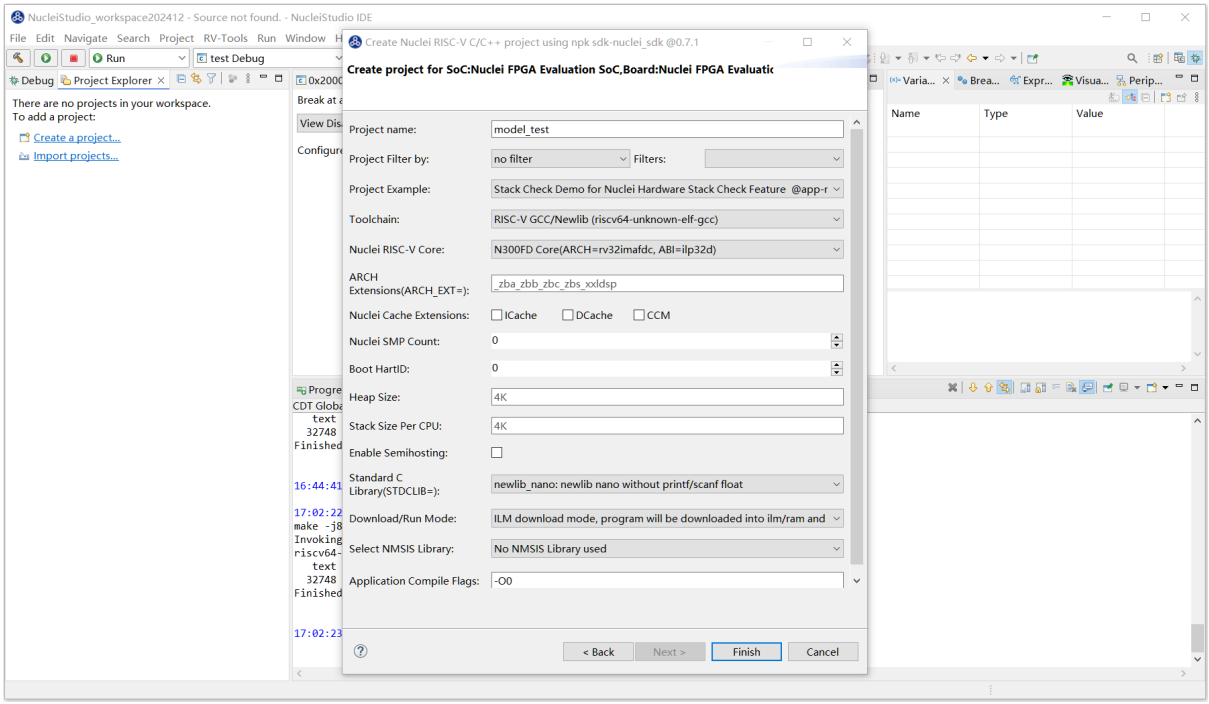
Note: Nuclei Near Cycle Model now supports both Windows and Linux. The tests in this document were all

performed with the Windows version of Nuclei Studio IDE 2025.02.

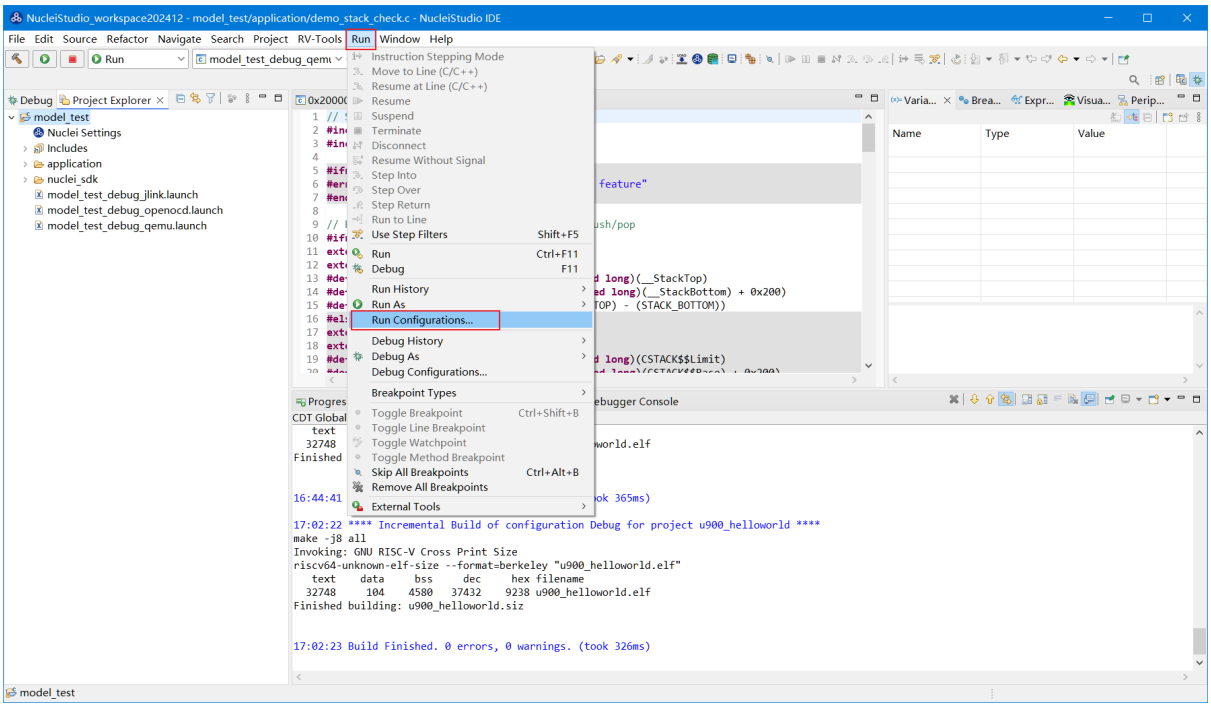
If you encounter any problems during use, check https://github.com/Nuclei-Software/nuclei-studio or submit an issue.

Creating a Test Project

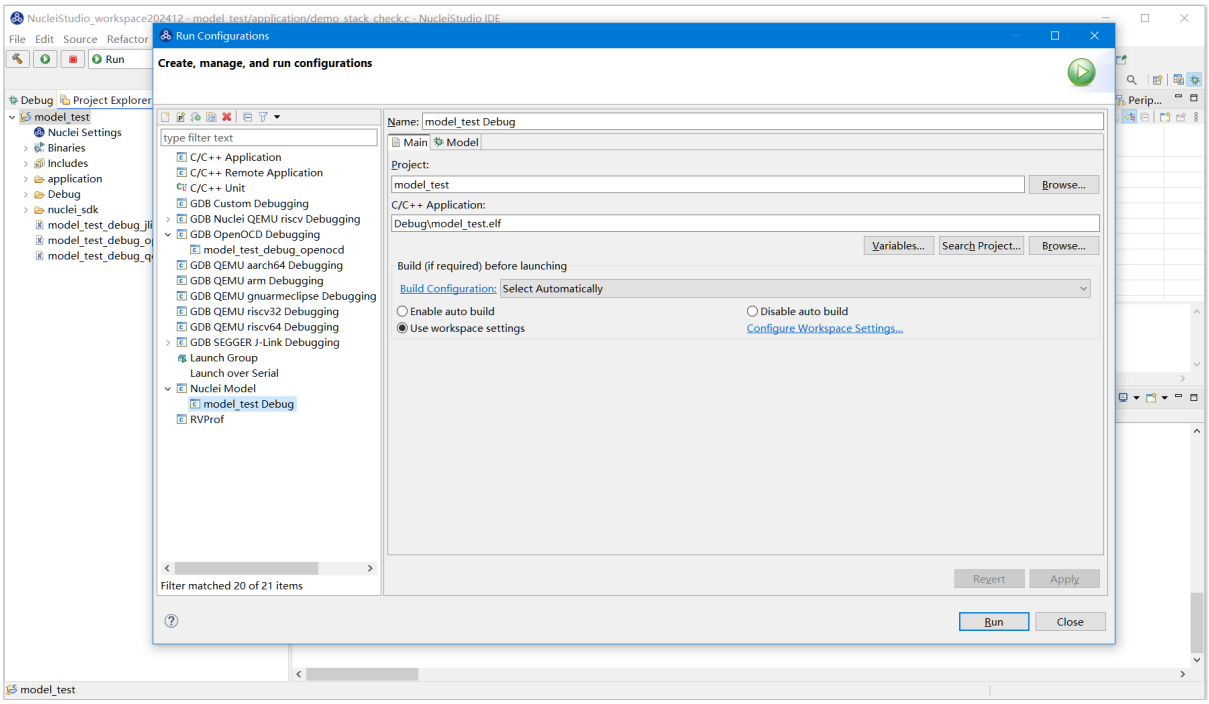
Nuclei Near Cycle Model supports all types of Nuclei cores. Create any demo project and build it.



Nuclei Near Cycle Model runs tests using the Model run configuration in Nuclei Studio. Select the built test project, and then open Run Configurations in Nuclei Studio.



Create a Nuclei Near Cycle Model configuration. The configuration parameters are described below.



In this example, `--trace=1 --gprof=1 --logdir=Debug --cpu=n300fd` is configured in Config options: `--trace=1` enables rvtrace, `--gprof=1` enables the gprof feature, `--logdir=Debug` stores the generated `.rvtrace` and `.gmon` files in the Debug directory under the current project, and `--cpu=n300fd` sets the simulated CPU core to n300fd.

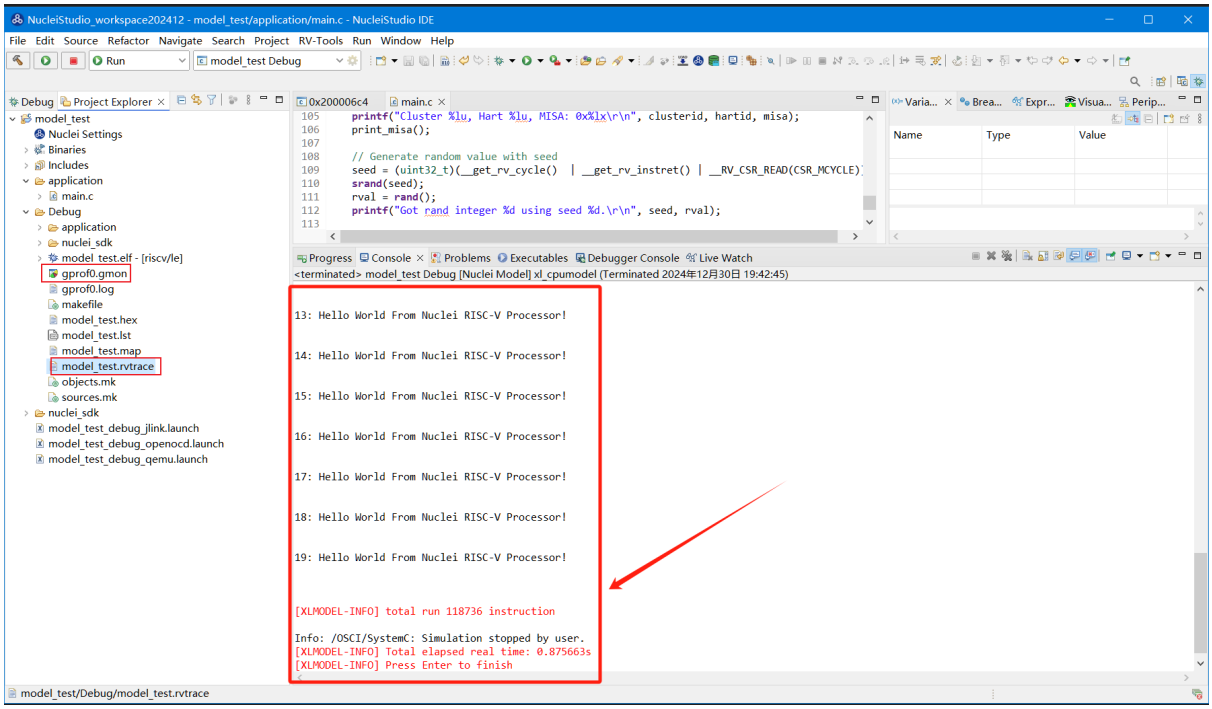
Note: `--cpu=<core type>` must be configured and must be consistent with the Core value configured in Nuclei Setting.

`--ext=<extension type>` must be consistent with the Other extensions value configured in Nuclei Setting.

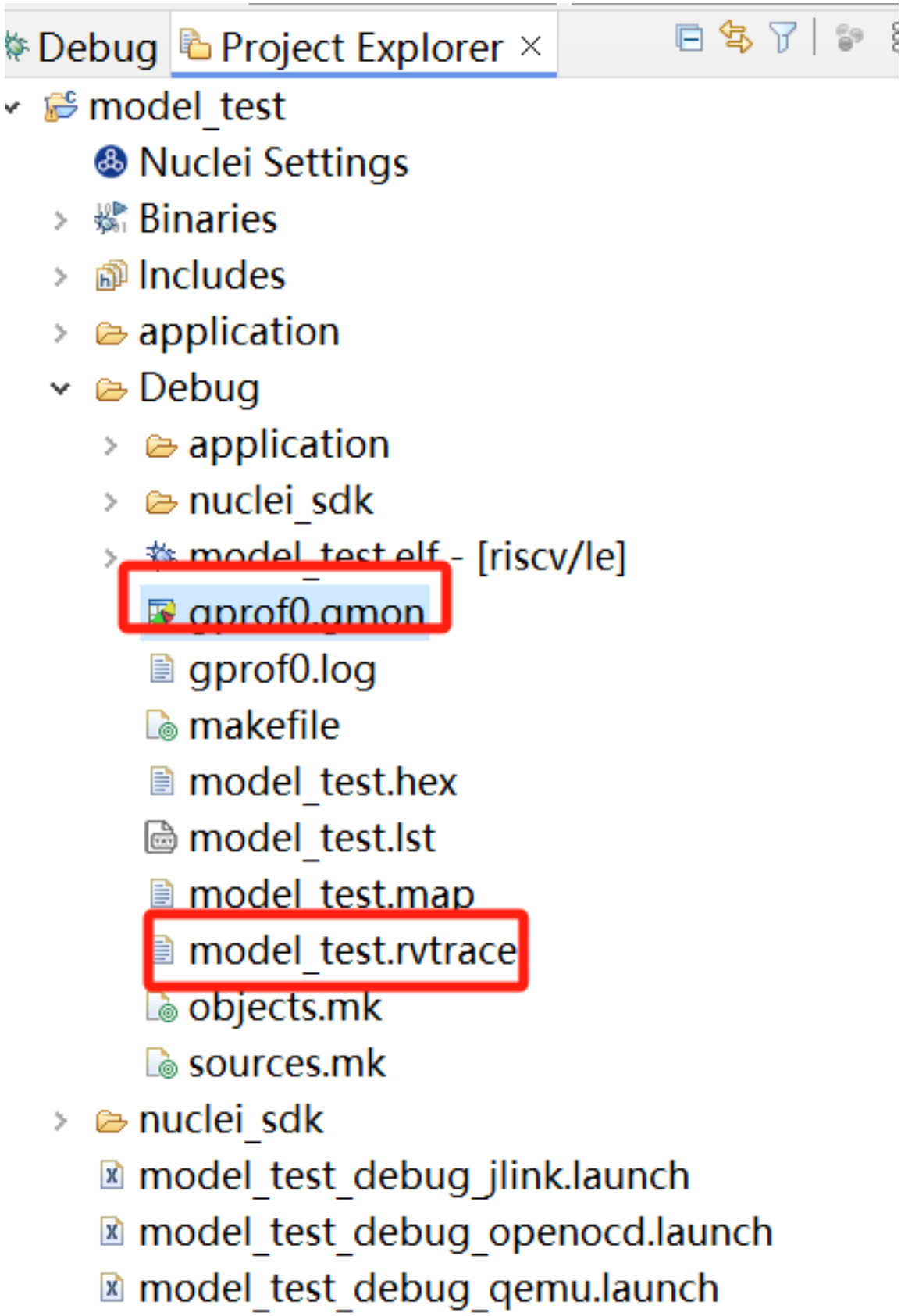
For details about the parameters of Nuclei Near Cycle Model, see *Description of Parameters* (page 866).

Running the Project and Generating Profiling Results

Click the Run button to run the program. The program runs successfully in Nuclei Near Cycle Model and outputs the corresponding log information.



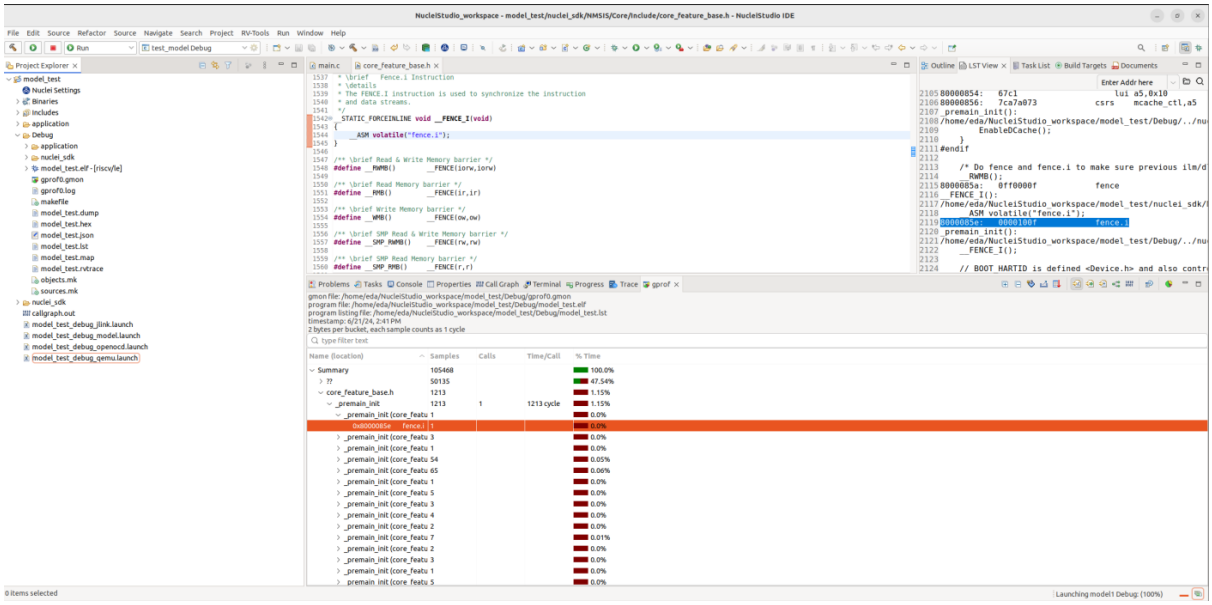
In the Debug directory of the project, you can see that the `.rvtrace` file and the `.gmon` file have been generated.



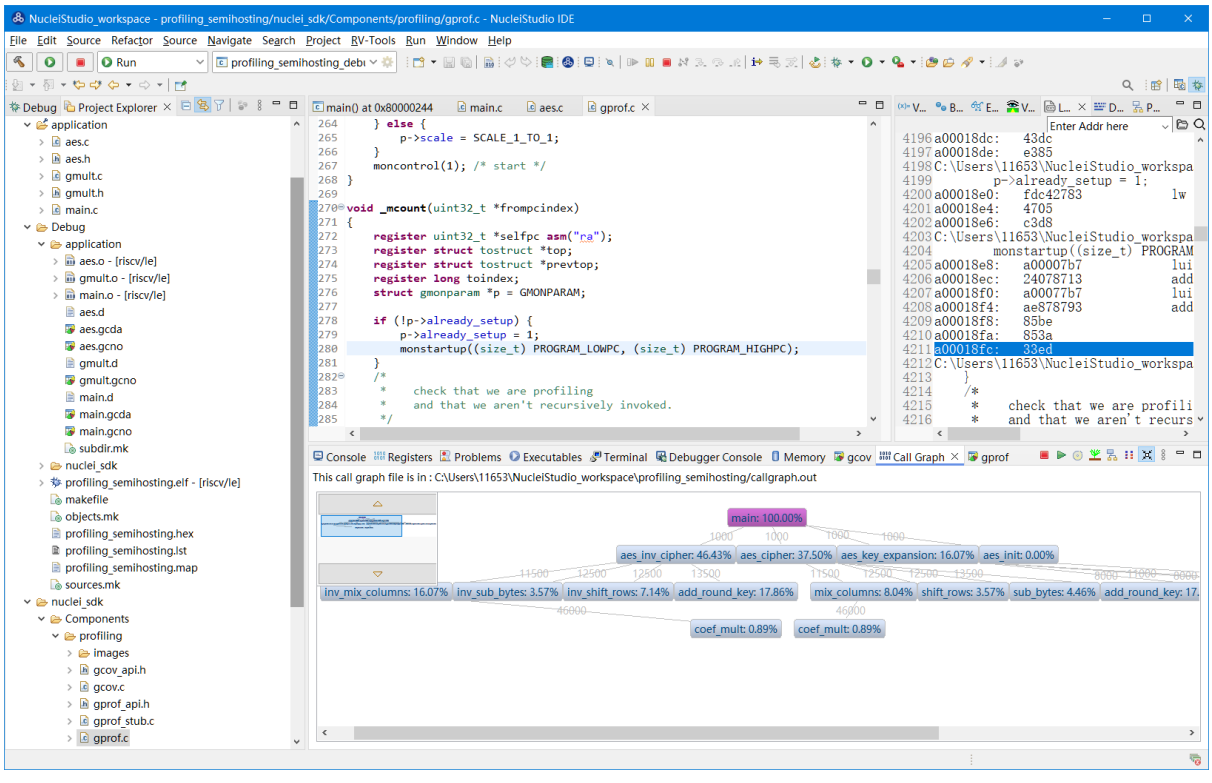
Nuclei Near Cycle Model supports analyzing programs with gprof. Therefore, when `--gprof` is configured, a `.gmon` file is also produced in the Debug directory (the directory configured with `--logdir=XX`) while the program runs. Double-click the `.gmon` file to invoke the gprof tool and analyze the number of cycles consumed by program execution and the call relationships; a corresponding `callgraph.out` file is also produced. Double-click

the `callgraph.out` file to view the program ' s call relationships with Call Graph.

The gprof tool displays the contents of the generated `.gmon` file.



While viewing the `.gmon` file, the gprof tool parses the program ' s call relationships based on its contents and generates a `callgraph.out` file. Double-click `callgraph.out` to view it with the Call Graph tool.



2.10.12 Using GDB Custom Debugger

GDB Custom is a simplified debug configuration developed by Nuclei to accommodate different debugging tools. It lets you debug programs in Nuclei Studio with your own custom debugging tools. GDB Custom supports debugging projects through Commands, Generic Serial, and DLink; choose the method that fits your setup.

The GDB Custom debugging feature is illustrated with DLink, a debugging tool developed by Nuclei. For details, see *Debugging and Running a Project with DLink* (page 187).

2.10.13 Using Live Watch

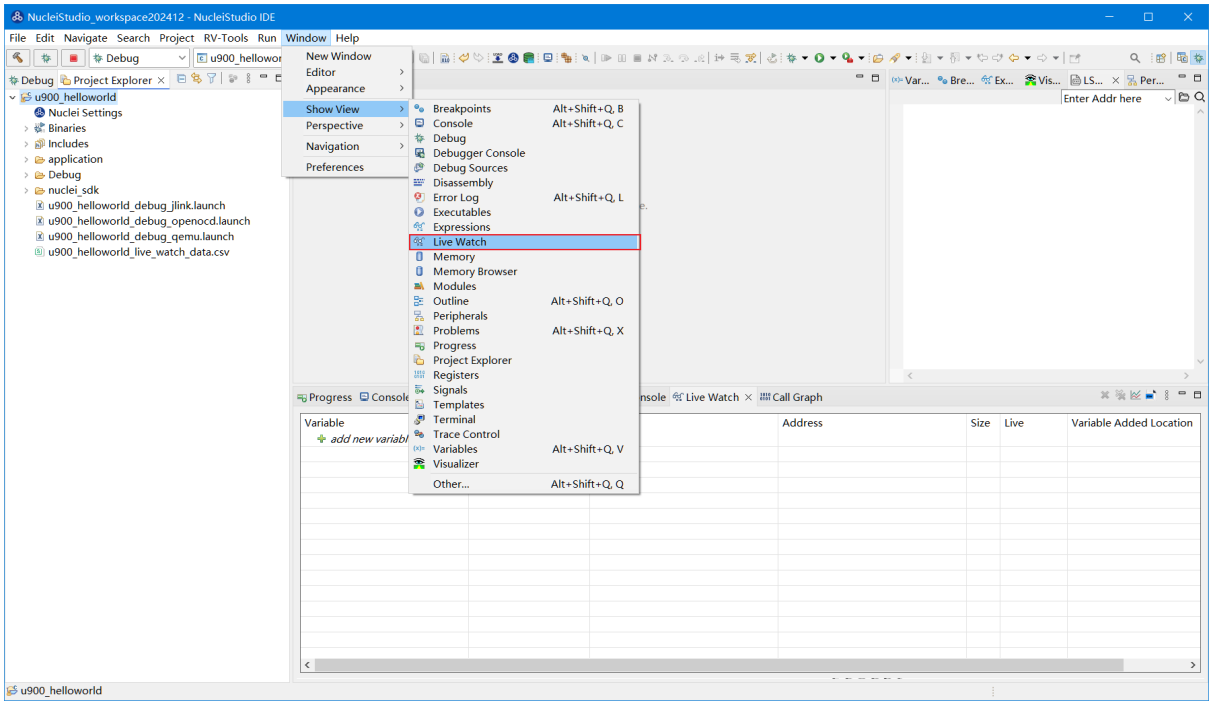
Live Watch is a real-time monitoring tool that helps you debug and optimize code more efficiently. With Live Watch, you can view variable changes instantly while the program is running, without interrupting execution or adding log statements manually. Live Watch was introduced in Nuclei Studio 2025.02. It refreshes variable values automatically so that you always see the latest data, and its intuitive graphical interface makes it easy to manage the variables you want to monitor.

Note: By default, Live Watch reads data through the **Telnet/SBA channel**, which relies on the Nuclei-customized OpenOCD (it must provide the `mbatch` command, supported since 2025.02) and requires the CPU to support RISC-V SBA.

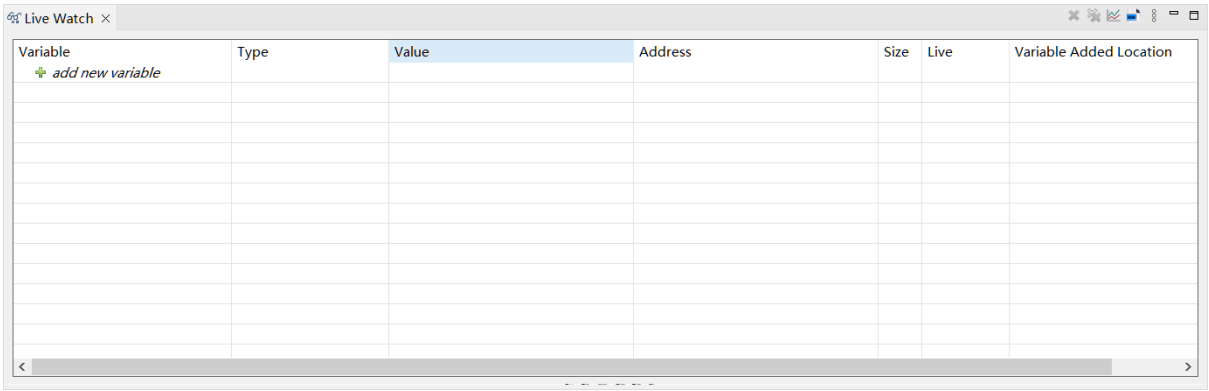
Starting from Nuclei Studio 2026.10, a **GDB channel** is added as another data source: it polls through GDB MI commands (such as `-var-update`), with lower latency and no dependence on SBA, but it **requires GDB non-stop mode to be enabled**. The trade-offs between the two channels and their relationship with non-stop are described below under "Getting Live Watch Data via the GDB Channel".

Live Watch Introduction

Open the Live Watch view via the Nuclei Studio menu `Window -> Show View -> Live Watch`.



The Live Watch view provides several function menus to help you manage and monitor variables:



Remove

- Deletes the selected variable row from the Live Watch view.

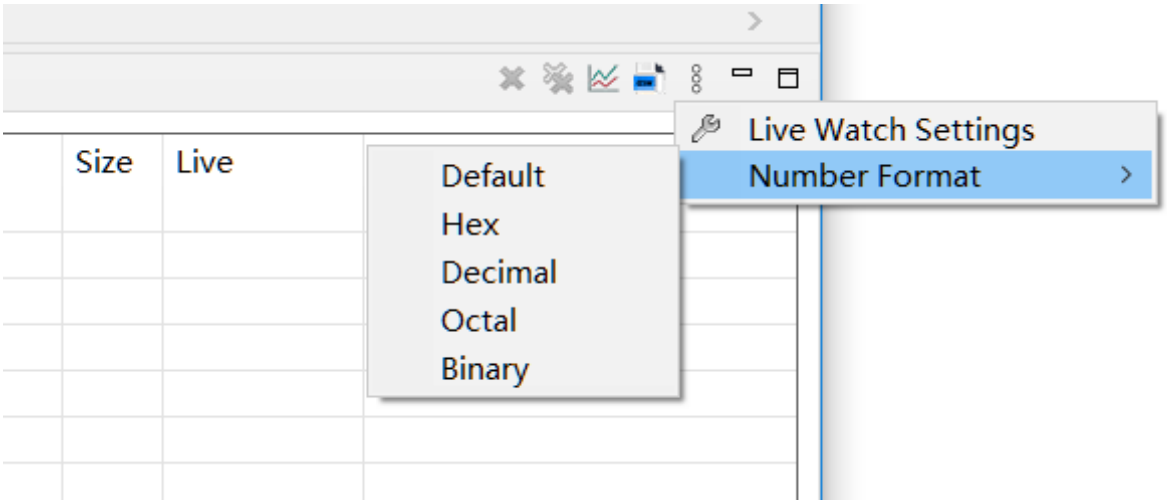
Remove All

- Clears all variables added to the Live Watch view.

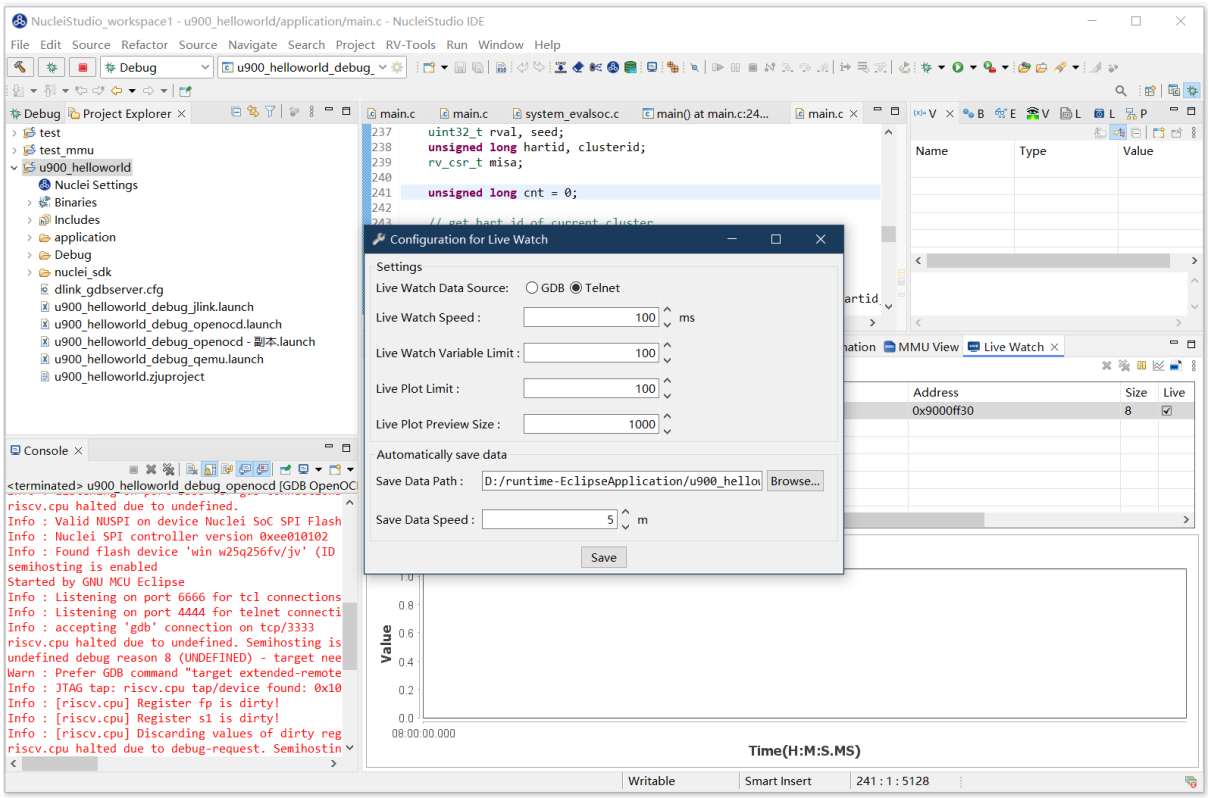
Show Live Plot

- Shows the Live Plot view, which plots the sampled data in real time.

The hidden menu bar contains two settings menus for configuring global properties:



Live Watch Settings



- Common settings in Live Watch:
  - The following settings are available:
    - Live Watch Speed : sets the sampling frequency of Live Watch, up to once every 1 ms. A sampling period of 5 ms or longer is recommended; if the sampling frequency is too high, Live Plot may draw abnormally.
    - Live Watch Variable Limit : limits the number of variables sampled at the same time, up to 10.
    - Live Plot Limit : sets the maximum number of samples that Live Plot draws at the same time, up to 10.
    - Save Data Path : specifies the path where the data sampled by Live Watch is automatically saved, for subsequent analysis.
    - Save Data Speed : sets the frequency at which Live Watch data is automatically saved. The default is once every 10 minutes.
    - Live Watch Data Source : selects GDB or Telnet. GDB is available only for OpenOCD debug configurations with non-stop enabled. When non-stop is not enabled, the GDB option is greyed out and Telnet is selected and saved automatically. To use the GDB channel, open the OpenOCD debug configuration page and check `Enable GDB non-stop mode to support Live Watch`.

Number Format

- Sets the display format of variable values in the Live Watch view.

Getting Live Watch Data via the GDB Channel

In Nuclei Studio 2026.10, support for getting Live Watch data via the **GDB channel** was added, as another data source besides the original Telnet/SBA channel. Its characteristics are lower latency and **no dependence on RISC-V SBA** —which is especially critical for targets that do not support SBA; the cost is that GDB ' s **non-stop mode** must be used.

Trade-offs between the two channels

Channel	Prerequisites	Main commands	Time-out / retry	Depends on SBA
GDB channel	Enable non-stop + newer GDB	`-var-update` / `-data-read-memory`	20 ms	No
Telnet channel (default fallback)	Nuclei-customized OpenOCD (provides `mbatch`) + CPU supports SBA	`mbatch r:<addr>:<size>:<`	5 s, retry 3 times	Yes

Why is non-stop mode required?

What non-stop decides is "whether the GDB channel can be used", not "whether the feature can be used".

In GDB ' s **all-stop mode (default)**, once the target starts running, it refuses to process MI queries and reports `Selected thread is running` —that is, when the program runs at full speed, GDB simply cannot read variables, so real-time polling is impossible. In **non-stop mode**, each thread can run independently, and GDB still accepts MI commands while the target is running, allowing Live Watch to keep polling with `-var-update` without pausing the target.

Therefore:

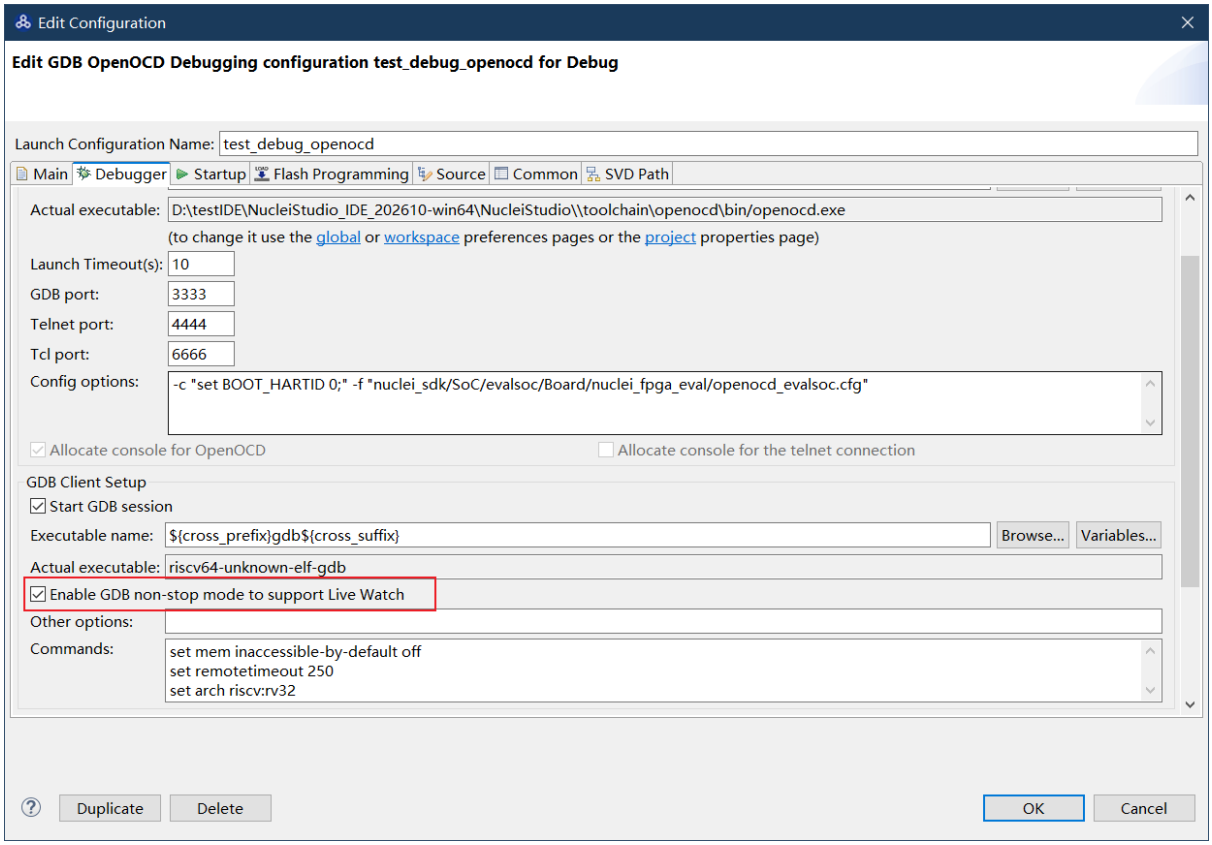
- **Enable non-stop** → uses the GDB channel, with good real-time performance and no SBA dependency;
- **Non-stop not enabled** → the GDB channel is unavailable, and Live Watch automatically falls back to the Telnet/SBA channel. The feature still works, but goes through the slower `mbatch` + SBA path that also requires SBA.

"non-stop must be enabled to use Live Watch" only holds under the premise of "using the GDB channel"; without non-stop, Live Watch simply switches to another channel and keeps working.

How to enable non-stop

Check the option in the **OpenOCD debug configuration page** of Nuclei Studio:

```
Enable GDB non-stop mode to support Live Watch
```

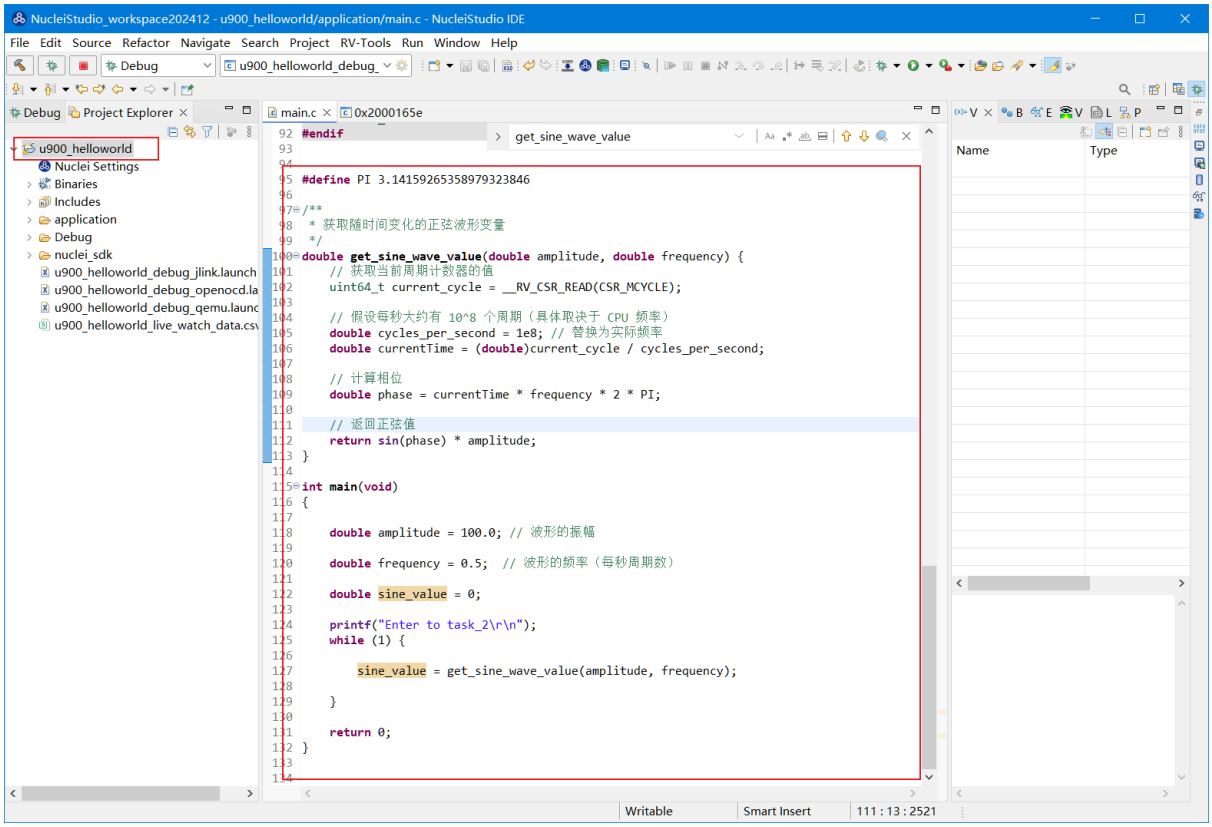


This checkbox writes the launch attribute `GDB_NONSTOP` (`ATTR_DEBUGGER_NON_STOP`); its tooltip text is literally *"You must enable this option to use Live Watch."* After checking it and launching the debug session, Live Watch uses the GDB channel.

Note: If Live Watch still cannot read data after enabling non-stop, first confirm: ① the debug configuration has the option above checked; ② the GDB version meets the non-stop threshold (the launch side validates this, and will report an error at launch if not); ③ the CDT DSF service is available. If necessary, refer to "Summary of Live Watch Issues" to confirm whether the target supports SBA.

Live Watch Usage Demo

Create a test project and implement a sine calculation in it. Open the Live Watch view, find Live Watch Settings, and set the related parameters as needed (or keep the default values).




```c
#include <stdio.h>
#include "nuclei_sdk_soc.h"
#include <math.h>

#define PI 3.14159265358979323846
/**
* Get a sine wave variable that varies with time
*/
double get_sine_wave_value(double amplitude, double frequency) {
    // Get the value of the current cycle counter
    uint64_t current_cycle = __get_rv_cycle();

    // Calculate the current time (in seconds)
    double currentTime = (double)current_cycle / SystemCoreClock;

    // Precompute the frequency-related factor
    double omega = frequency * 2 * PI;

    // Calculate the phase
    double phase = currentTime * omega;

    // Return the sine value
    return sin(phase) * amplitude;
}

int main(void)
{

    double amplitude = 100.0; // amplitude of the waveform

    double frequency = 0.1;  // frequency of the waveform (cycles per second)

    double sine_value = 0;
```

(continues on next page)

(continued from previous page)

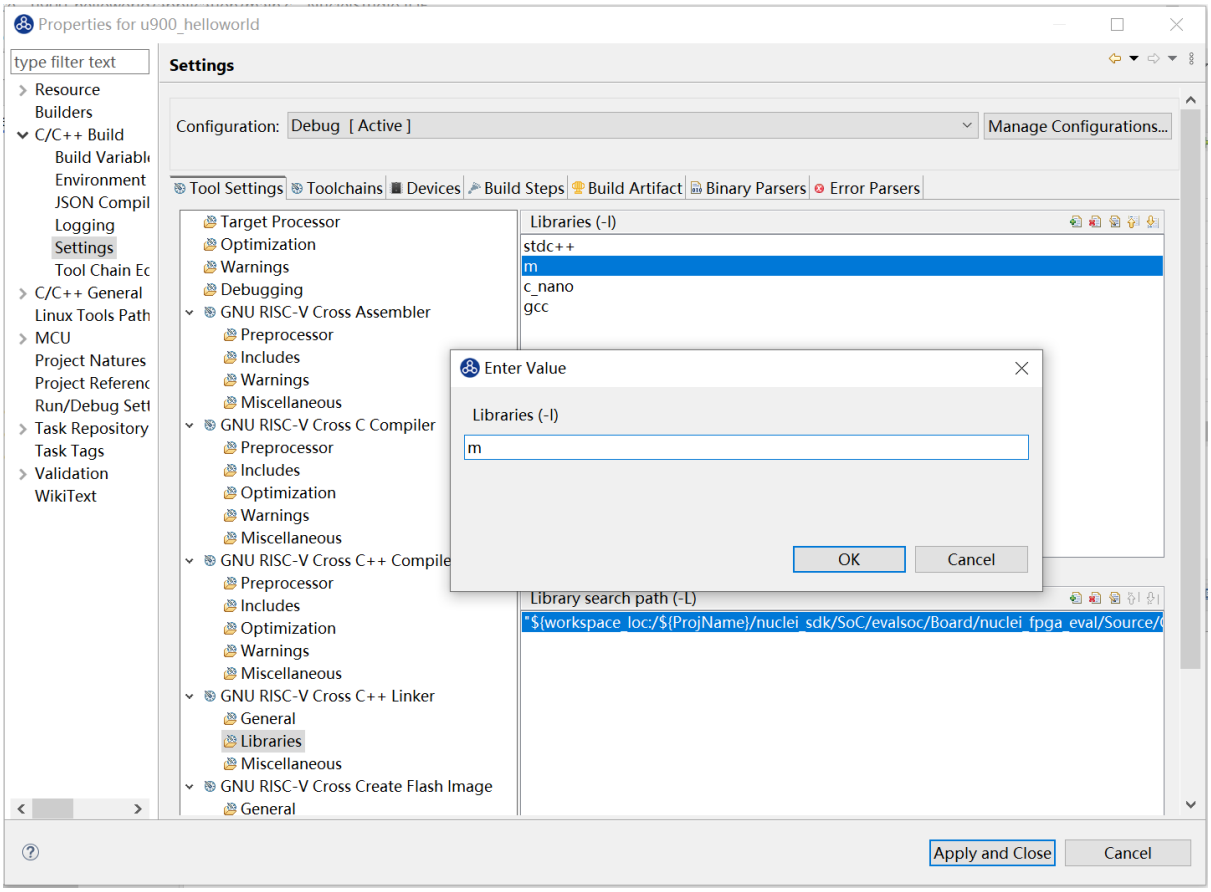
```
    printf("Enter to task_2\r\n");
    while (1) {

        sine_value = get_sine_wave_value(amplitude, frequency);

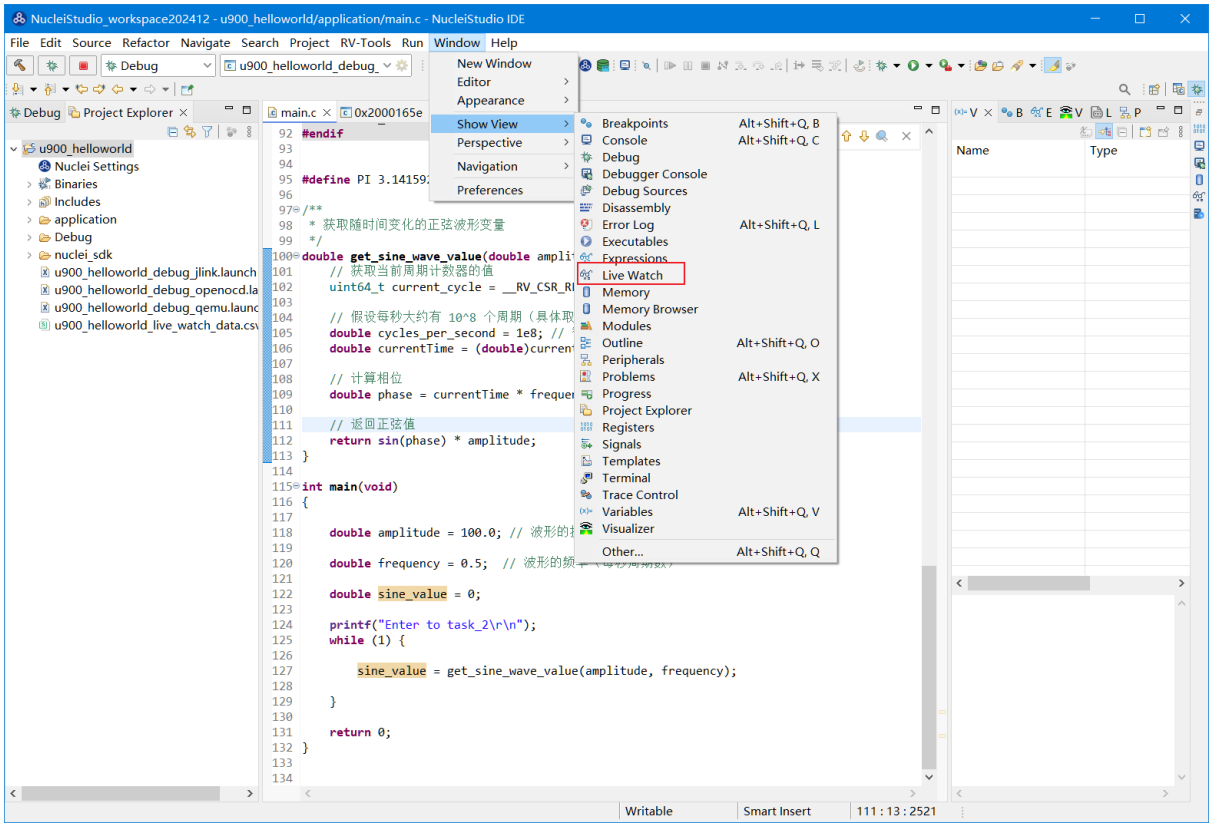
    }

    return 0;
}
```

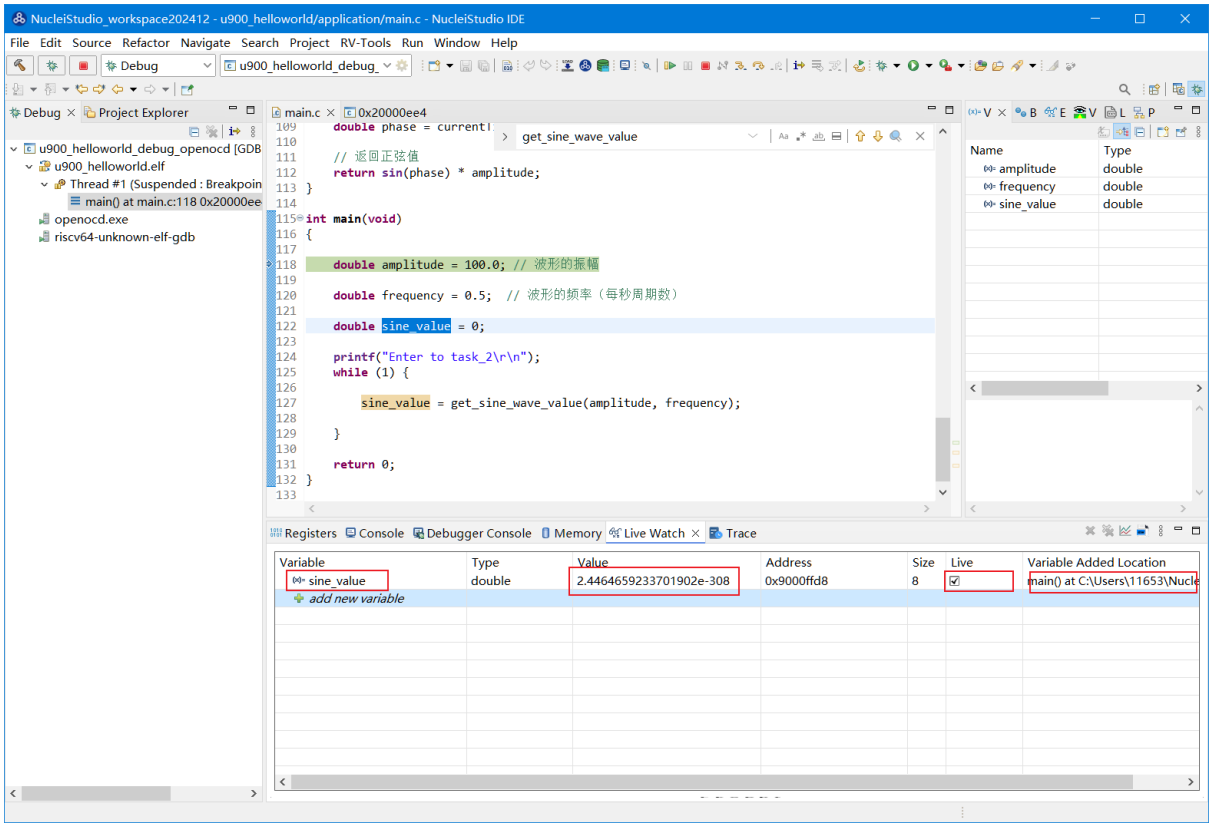
The test demo uses math functions, so you need to add `-lm` to the linker options. In the project properties, go to `Settings -> GNU RISC-V Cross C++ Linker -> Libraries` and add `-lm`.



Open the Live Watch view via `Windows -> Show View -> Live Watch`.

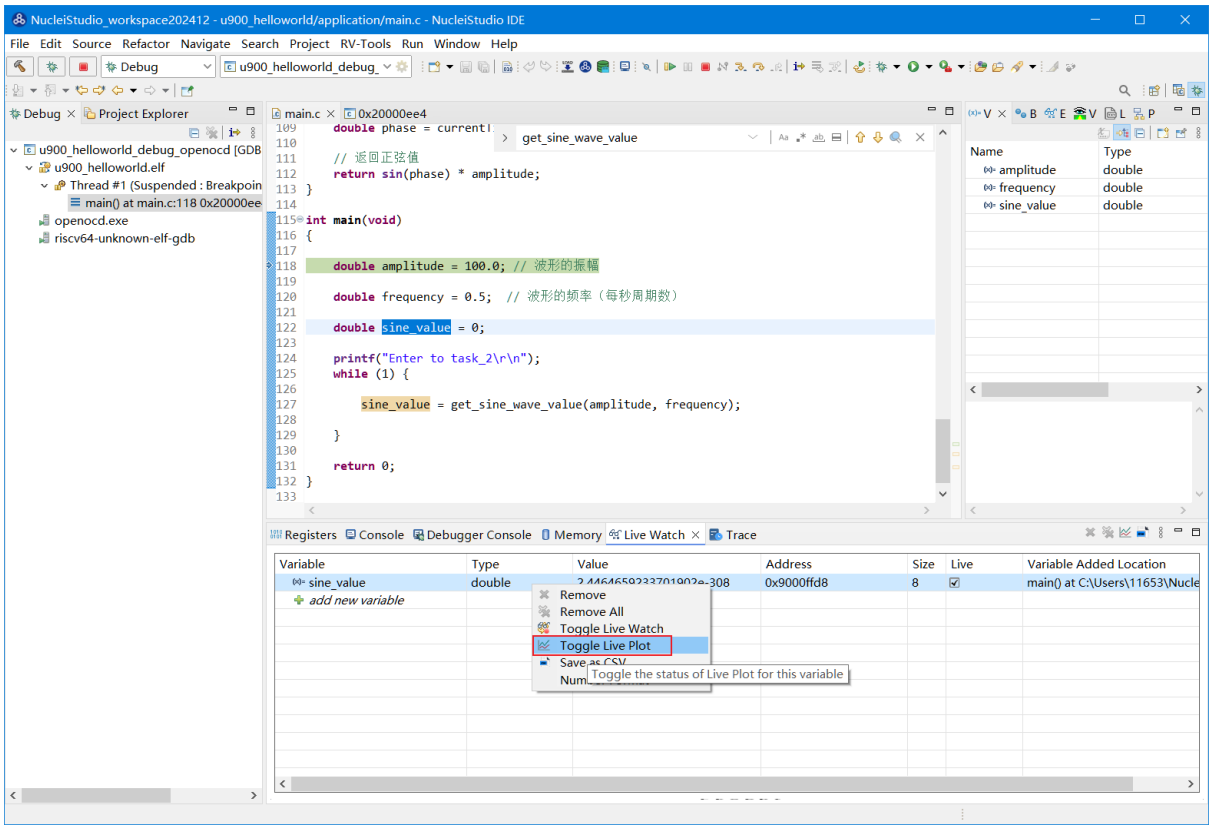


After building the project and running the program in Debug mode, add the variables you want to view in the Live Watch view. There are two ways to add variables: click `add new variable` in the Live Watch window to add them manually, or select a variable in the project and drag it into the Live Watch window with the left mouse button.

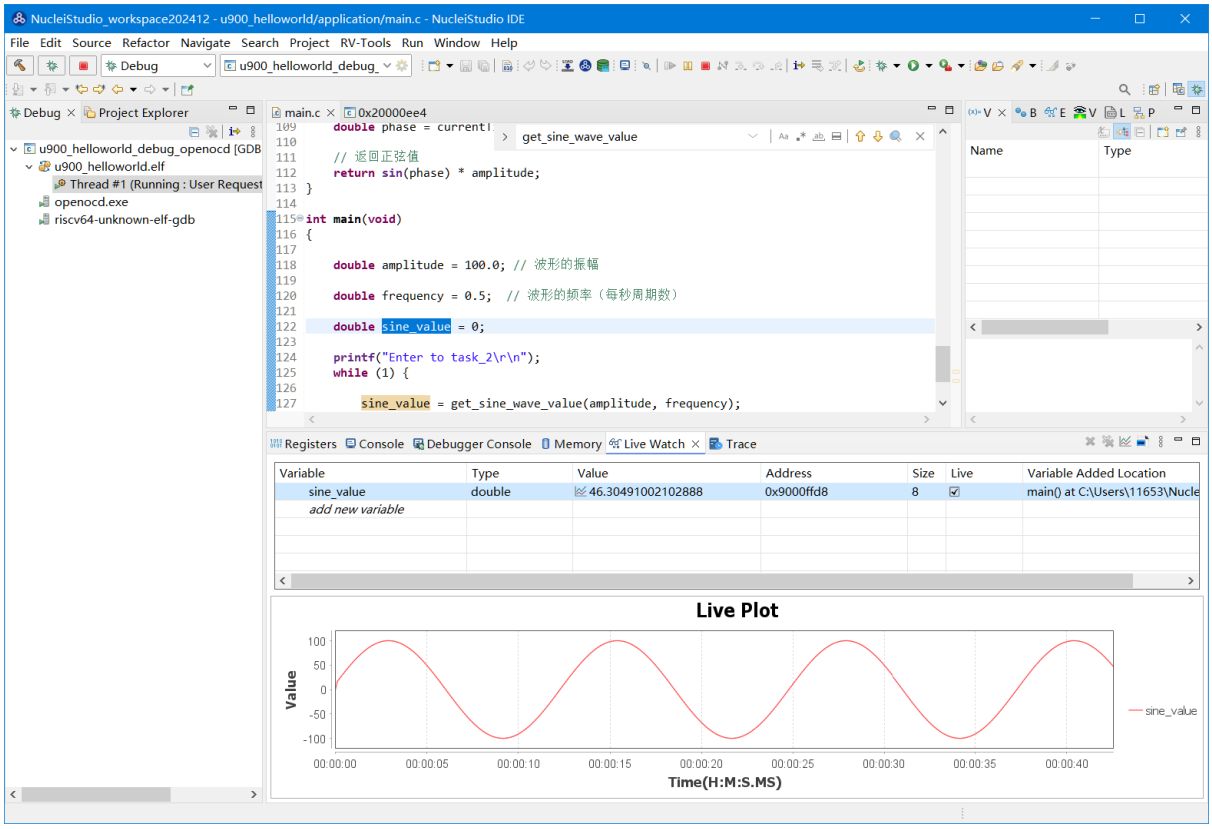


When the program runs at full speed, the variable values change at the configured Live Watch Speed. To view

a variable ' s change curve with Live Plot, select the record, right-click, and select `Toggle Live Plot` from the context menu. The Live Plot tool opens and draws the variable ' s change curve.

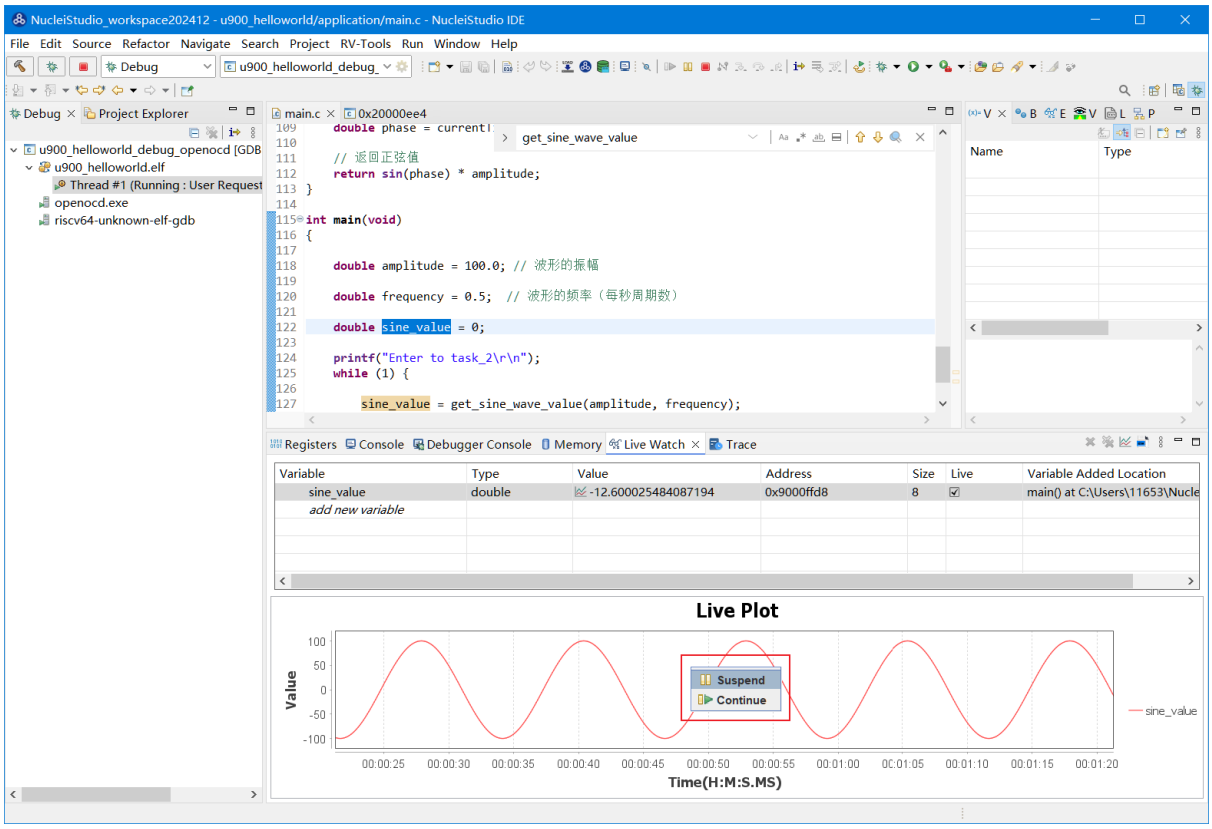


The curve drawn by Live Plot is shown below.

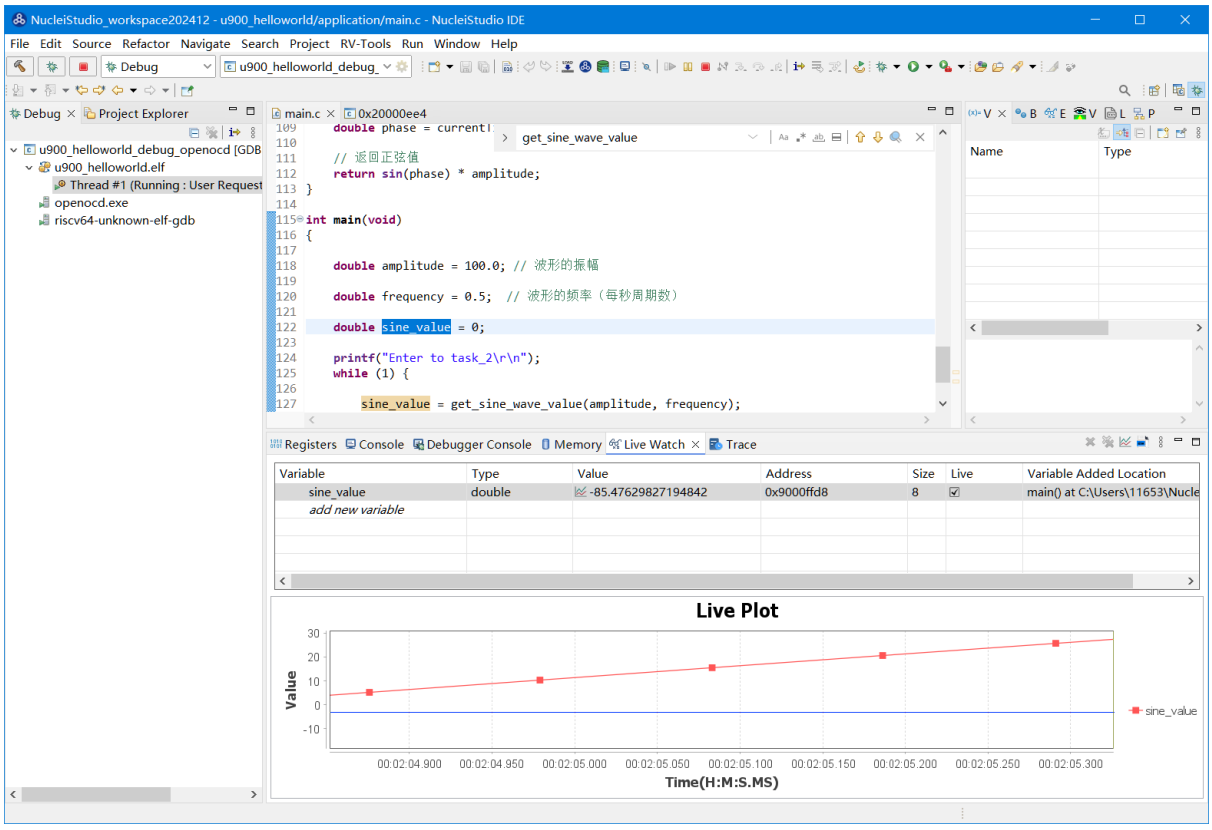


Right-click in Live Plot to open a menu with two items, `Suspend` and `Continue`. Click `Suspend` to pause

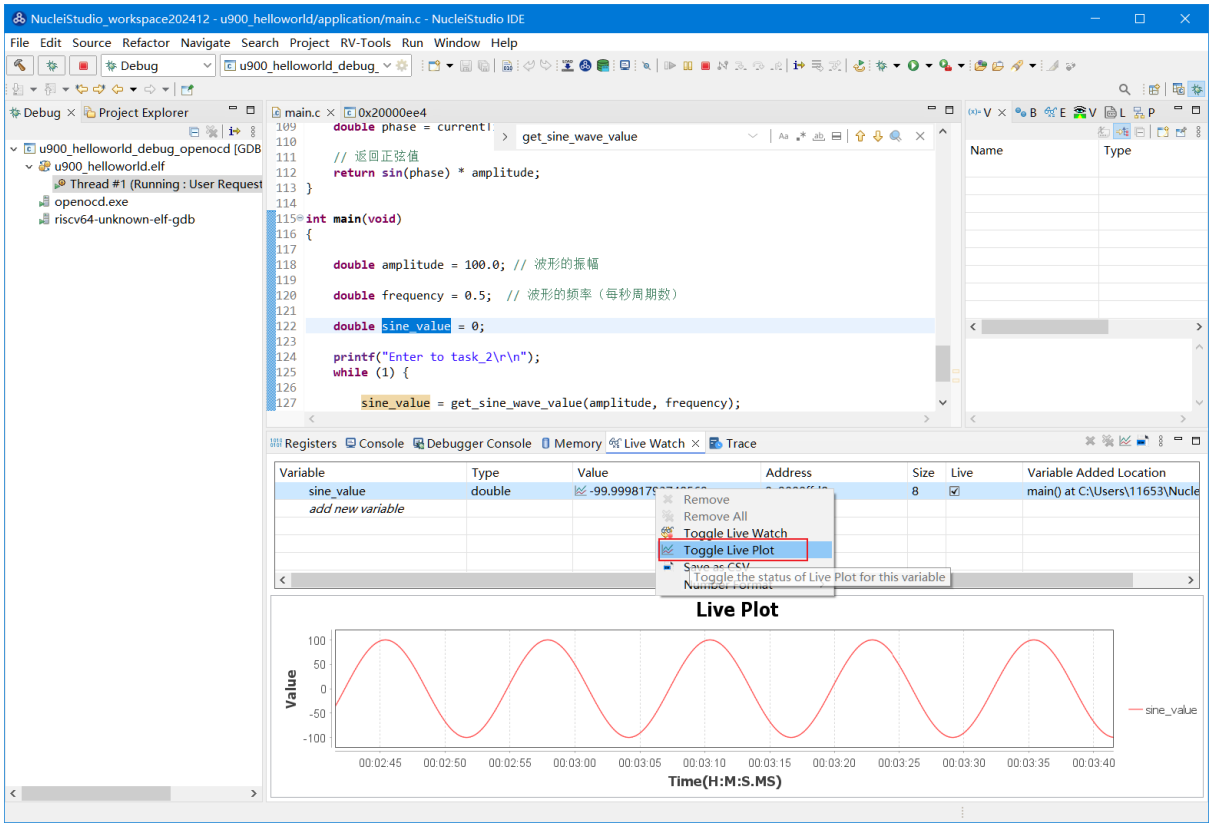
drawing.



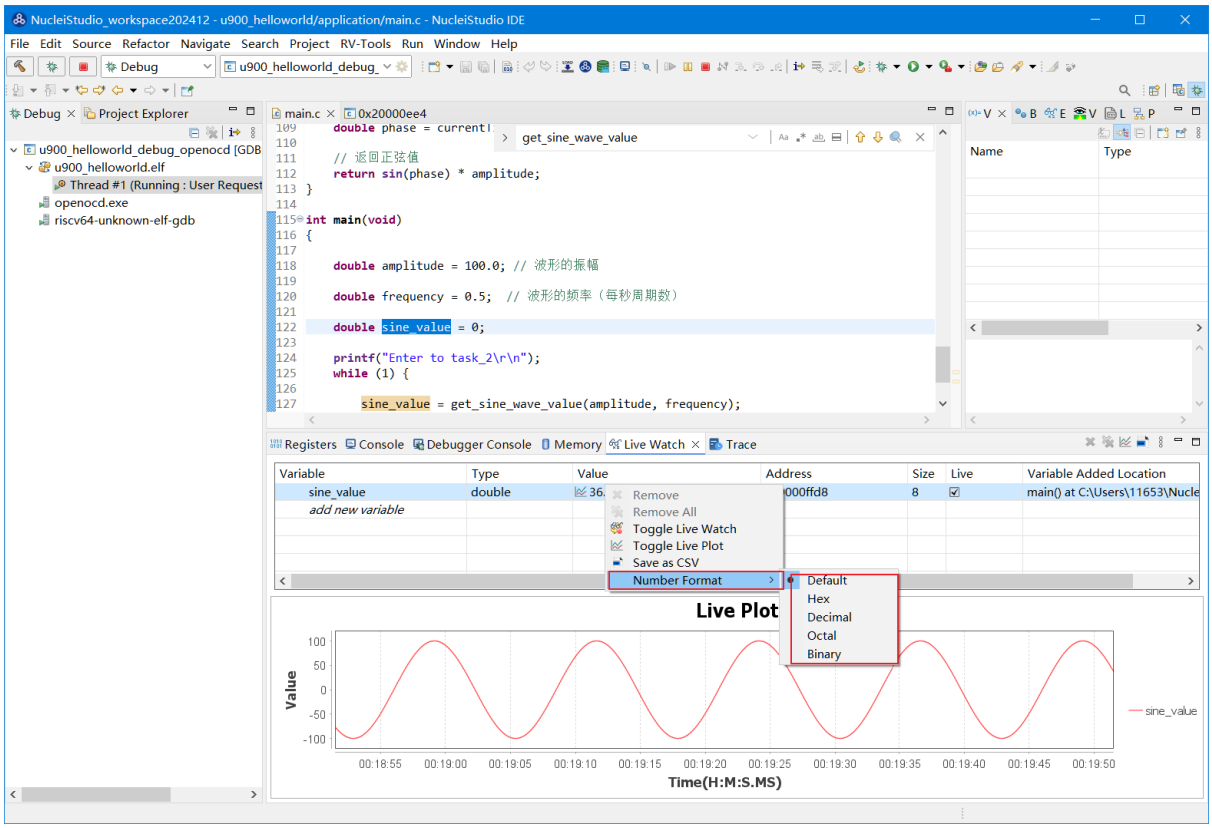
Scroll the mouse wheel to zoom in on the curve and view data details; click `Continue` to resume drawing.



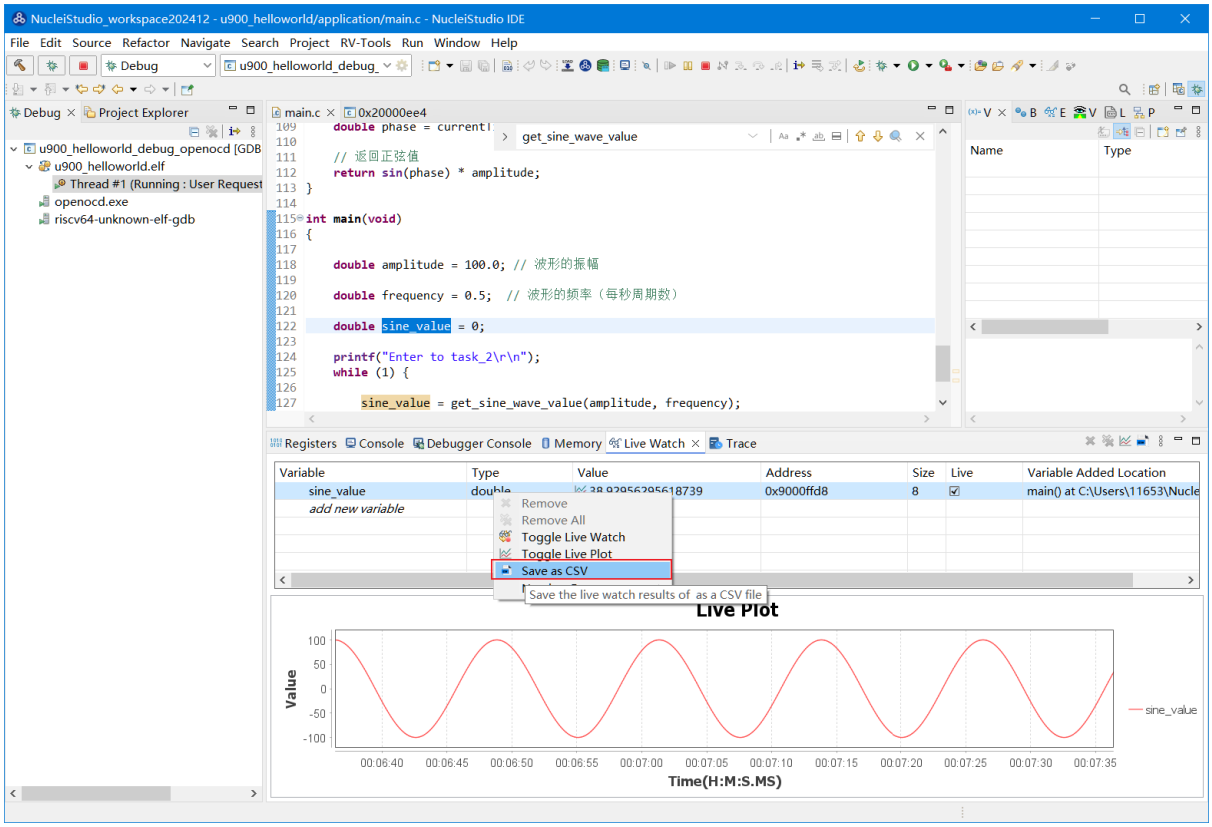
If you no longer want to view the change curve of a variable, click `Toggle Live Plot` again to remove the variable from Live Plot.



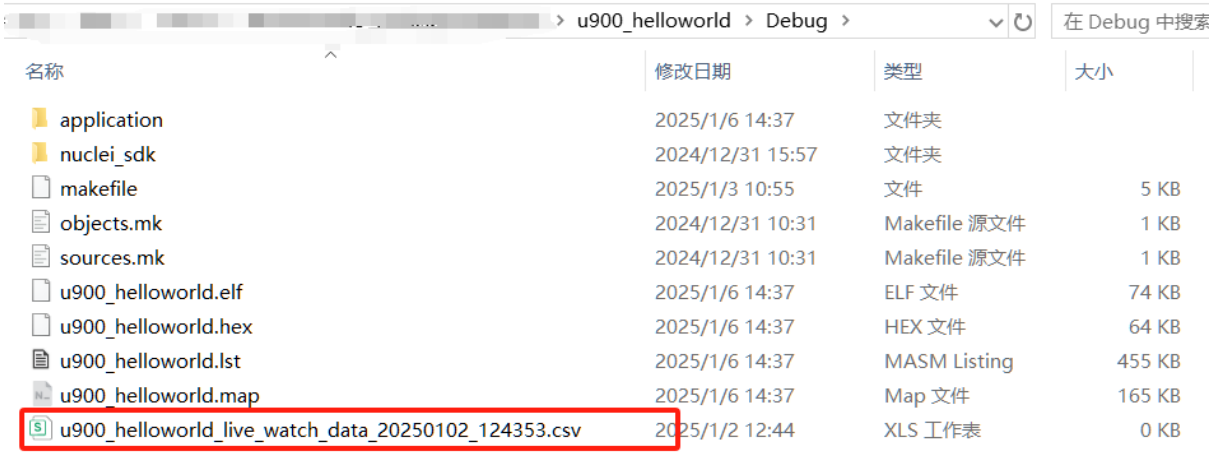
Right-click a variable in the Live Watch view to modify the display format of its data.



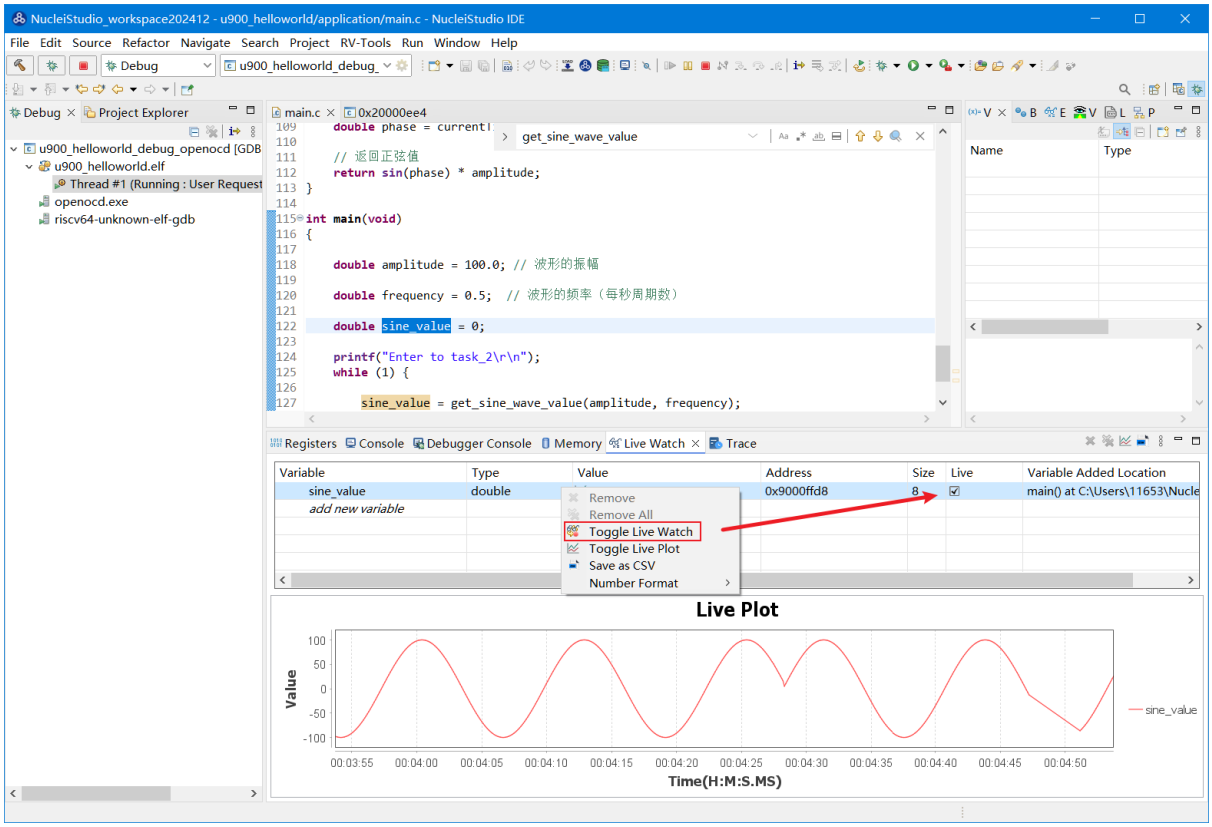
Right-click a variable in the Live Watch view to save its sampled results as a CSV file for later use.



Live Watch also automatically saves the sampled data to `Save Data Path`, where you can find the corresponding CSV files.



To stop watching a variable, select the record, right-click, and select `Toggle Live Watch` from the context menu. Live Watch then stops querying the variable ' s value in real time.

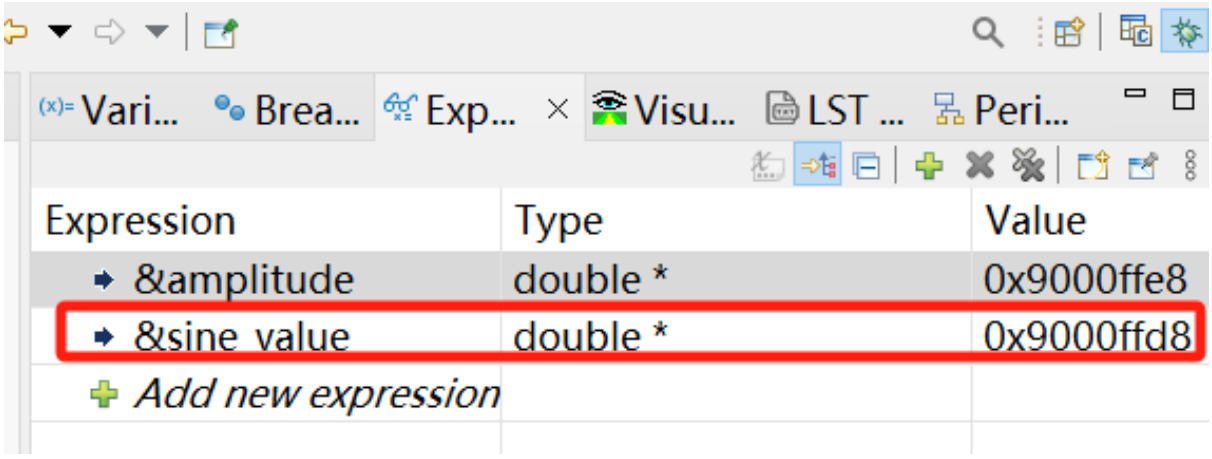


Troubleshooting Live Watch

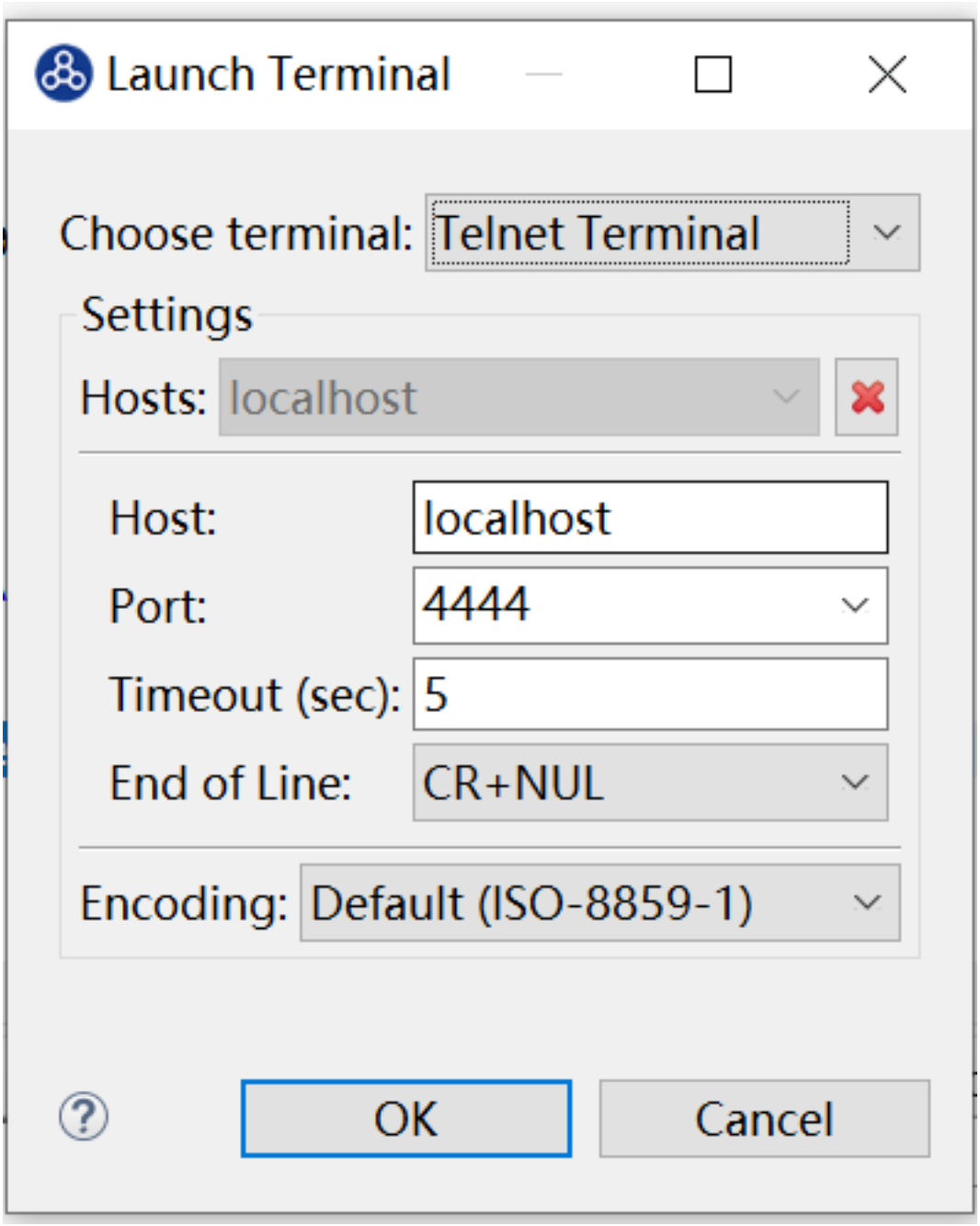
Dependency on RISC-V SBA

As mentioned earlier, Live Watch depends on RISC-V SBA. If Live Watch cannot obtain a variable ' s value, the current CPU may not support RISC-V SBA. Use the following method to confirm whether the CPU supports RISC-V SBA.

While the project is in the Debug state, obtain the address of the variable in Live Watch, as shown below.



Open the telnet tool, configure it, and connect. If the connection fails, check whether the OpenOCD service has started properly and whether the telnet port is open.



While still in the Debug state, use the command `riscv set_mem_access sysbus` in the telnet tool to set the riscv memory access mode to sysbus (SBA), and try to read the value of the variable.

```
> riscv set_mem_access sysbus
> mdw 0x9000ffd8 4
0x9000ffd8: 000ab156 404e8df0 9999999a 3fb99999
```

If the operation above reads the variable ' s value, run the program at full speed and read the value again. If no error is reported and the value can be read, the current CPU supports the SBA feature.

```
> mdw 0x9000ffd8 4
0x9000ffd8: 000ab156 404e8df0 9999999a 3fb99999
> mdw 0x9000ffd8 4
0x9000ffd8: d0b53424 c0510557 9999999a 3fb99999
> mdw 0x8000ffd8 4
0x8000ffd8: 00000000 00000000 00000000 00000000
```

(continues on next page)

(continued from previous page)

```
> mdw 0x9000ffd8 4
0x9000ffd8: b169d356 c0580176 9999999a 3fb99999
> mdw 0x9000ffd8 4
0x9000ffd8: a69861c1 c056547d 9999999a 3fb99999
> mdw 0x9000ffd8 4
0x9000ffd8: 4dbf49f2 404022d3 9999999a 3fb99999
> mdw 0x9000ffd8 4
```

Live Watch Cannot Read Variable Values When the Optimization Level Is Too High

When the project ' s optimization level is high, Live Watch may not be able to obtain the values of some variables, as demonstrated below.

```
int test()
{
   uint32_t live_watch = 0;
   while(1){
      live_watch += 1;
      delay_ms(100U);
   }
}
```

When the project ' s optimization level is set to O3 or higher, Live Watch cannot obtain the value of `live_watch`. You can lower the project ' s optimization level, or declare the monitored variable as a global or static variable so that it is not optimized away by the compiler. Modify the project code as follows.

```
int test()
{
   static uint32_t live_watch = 0;
   while(1){
      live_watch += 1;
      delay_ms(100U);
   }
}
```

2.10.14 Live Peripheral

Live Peripheral displays the chip peripherals, registers, and bit fields in a tree, and can periodically refresh register values during program execution, and also allows directly writing registers. Its biggest difference from the traditional Peripherals view is " real-time " : you can see register values change as the program runs without pausing it. It shares the same backend data service (GDB channel and OpenOCD Telnet channel) with Live Watch; see *Live Watch* (page 306).

Overview

Live Peripheral senses " which peripherals, register addresses, and bit-field meanings " through the SVD file, presents them as a three-level tree of peripheral → register → bit field, and supports real-time refresh and writing during program execution.

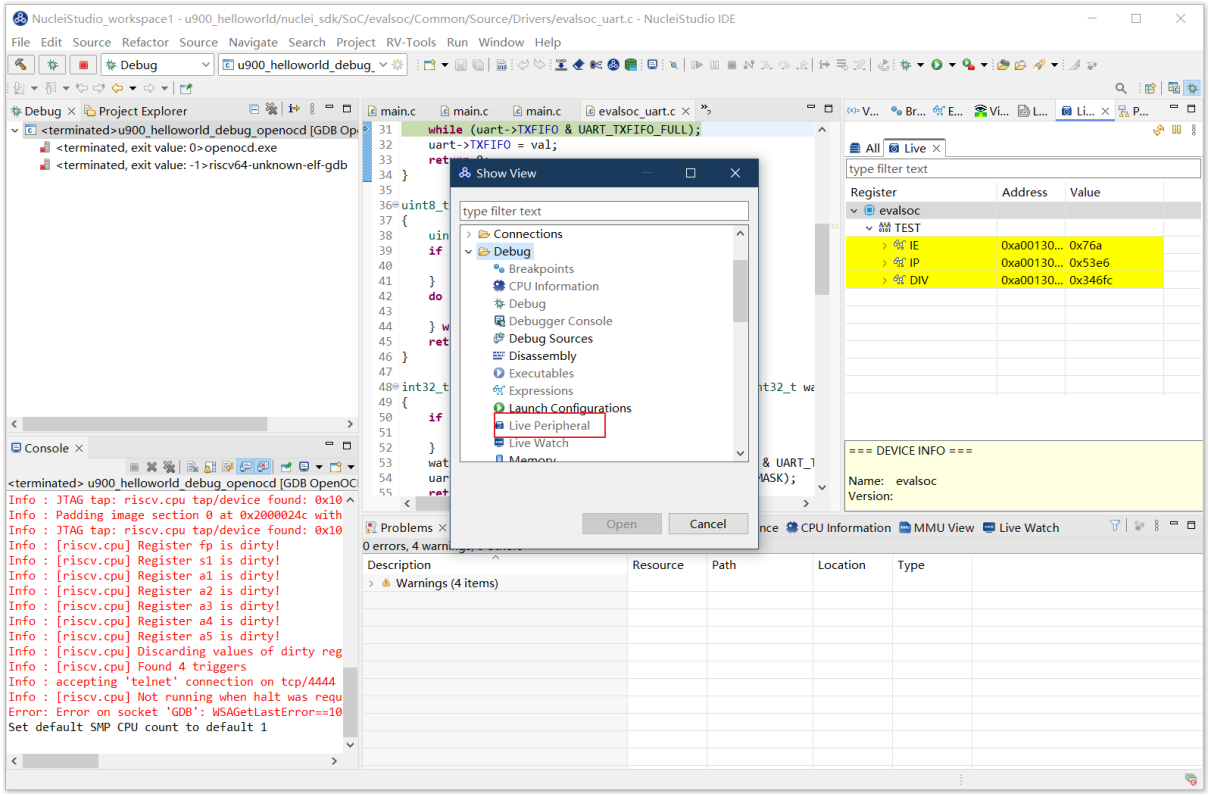
Prerequisites

1. The SVD Path must be configured. Live Peripheral knows the peripheral and register layout only through the SVD file; without it, the view cannot be opened. The SVD is usually provided by the NPK device package, and the configuration location is the SVD Path input box in the debug configuration under `Run -> Debug Configurations...` (in most cases it is automatically written when the NPK creates the project). If the SVD Path is not set or the file is invalid, an error prompt will pop up when opening the view.
2. Real-time sampling requires SBA: the data source can be GDB or Telnet; if an old GDB is detected, it automatically falls back to Telnet.

3. QEMU limitation: under QEMU, only paused-state ordinary register reads are possible; real-time sampling is not supported.

Live Peripheral Introduction

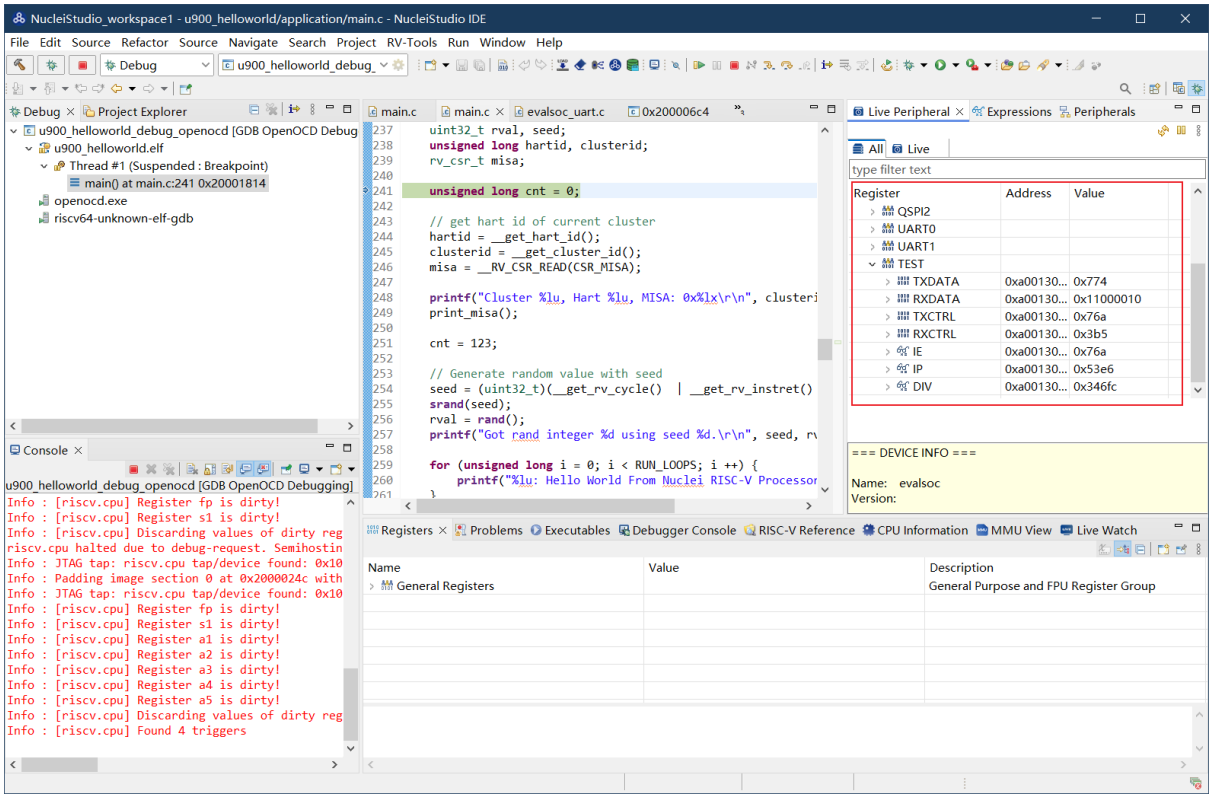
Menu `Window -> Show View -> Debug -> Live Peripheral`. The view is visible by default in the Debug perspective, stacked with the Variables view; in the C/C++ perspective only a shortcut is registered.



The main body is a tree with the hierarchy peripheral → register → bit field. The toolbar provides real-time sampling control (`Start` / `Suspend All Live Watch`).

View registers

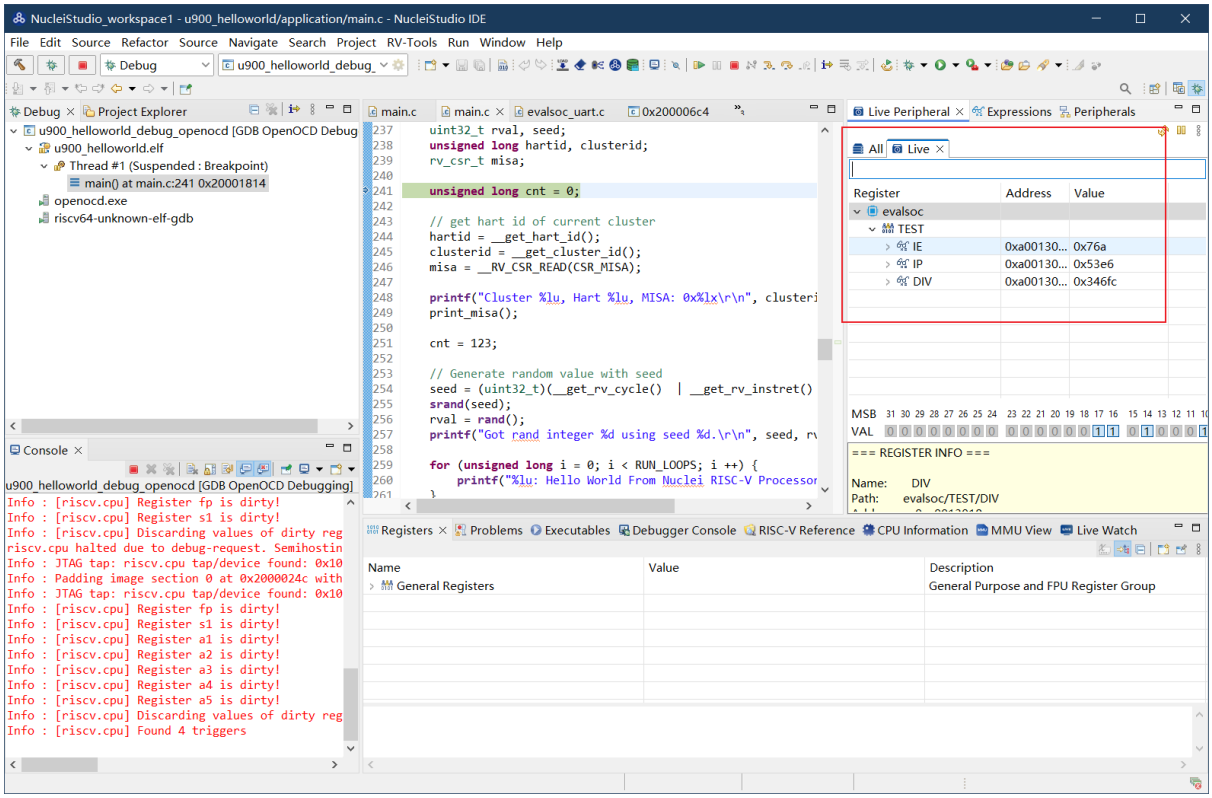
Expand the target peripheral in the tree; after expansion, the current values of each register under that peripheral are read immediately; expanding a register further shows the values and meanings of each bit field.



Real-time sampling

1. Make sure the SVD is configured and the backend supports SBA.
2. Enable Live Watch for the register(s) of interest.
3. Click `Start` on the toolbar to start sampling; register values are refreshed periodically during program execution.
4. Click `Suspend All Live Watch` when you need to pause sampling.

Note: adding new sampling items is not supported in non-stop mode.



Write registers

Select a writable register and write a new value directly. Note: registers marked read-only cannot be written; the Number Format menu can switch the Hex / Binary / Dec / Oct display format, and writes are parsed according to the current format.

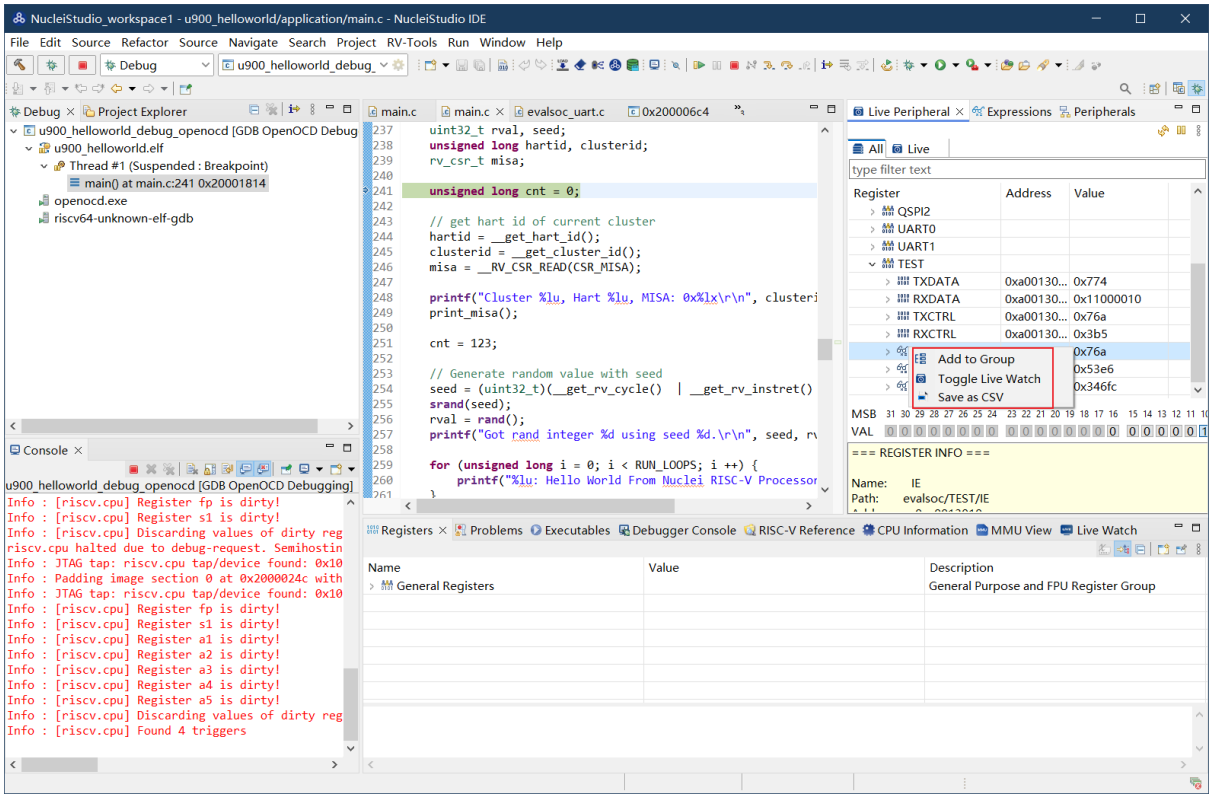
Favorites grouping and export

Add to Group: gather commonly used registers into a dedicated tab to avoid flipping back and forth in the big tree. Select the target register in the register tree (or multi-select multiple registers on a peripheral), right-click (or use the toolbar) and choose `Add to Group`, select an existing group or enter a new group name; after confirmation, the view generates a new tab named after that group.

Save as CSV: export all sampling history of a certain register as a CSV file. Select a register in the register tree (CSV export only takes effect on register nodes), right-click and choose `Save as CSV`; the default file name is `<project name>_<register name>_register.csv`. The CSV contains the following columns:

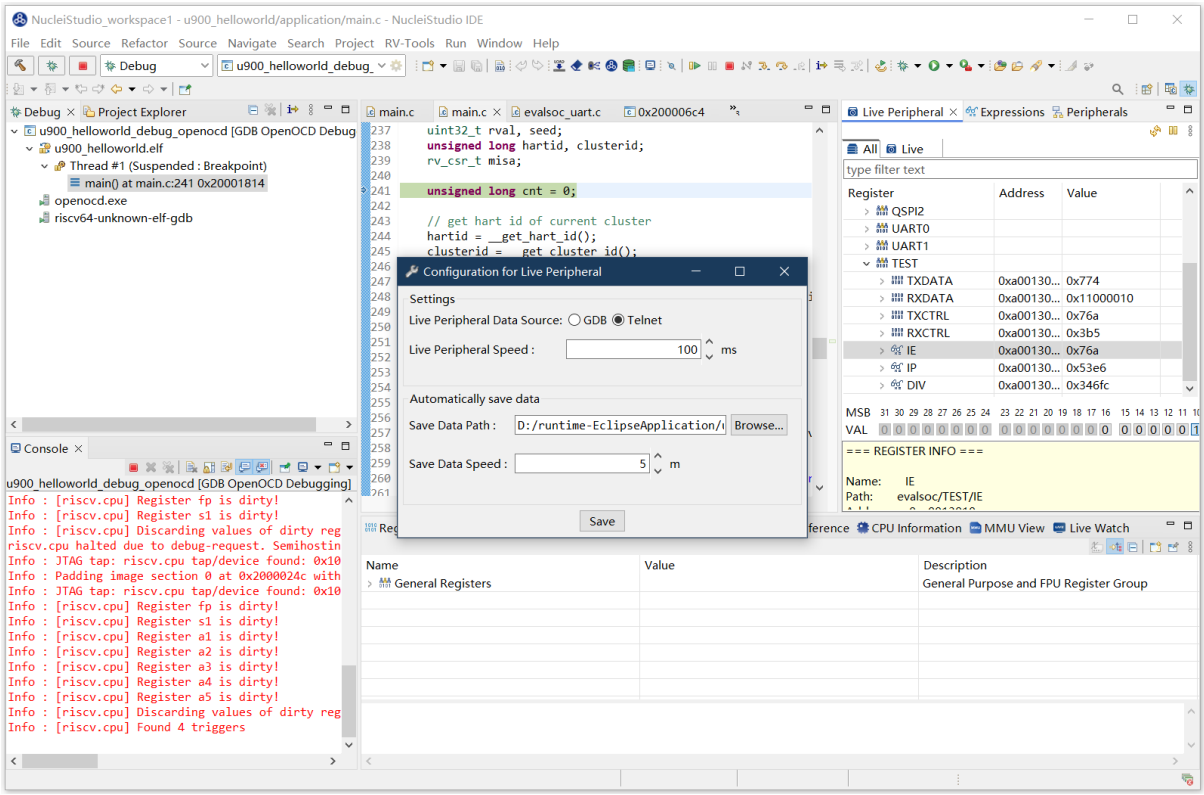
Column	Meaning
requestId	The request sequence number of each sampling
Register Name	Register name
Address	Register address (hexadecimal)
Value	The register value read in that sampling
Hexdata	Raw hexadecimal data
Time (s)	Elapsed time relative to the first sampling (seconds, format second.millisecond)

Tip: the CSV records historical sampling values; you must have enabled Live Watch sampling and accumulated data first, otherwise the export is empty.



Live Peripheral Settings

View menu → `Live Peripheral Settings`:



.. list-table:

```
:header-rows: 1
```

(continues on next page)

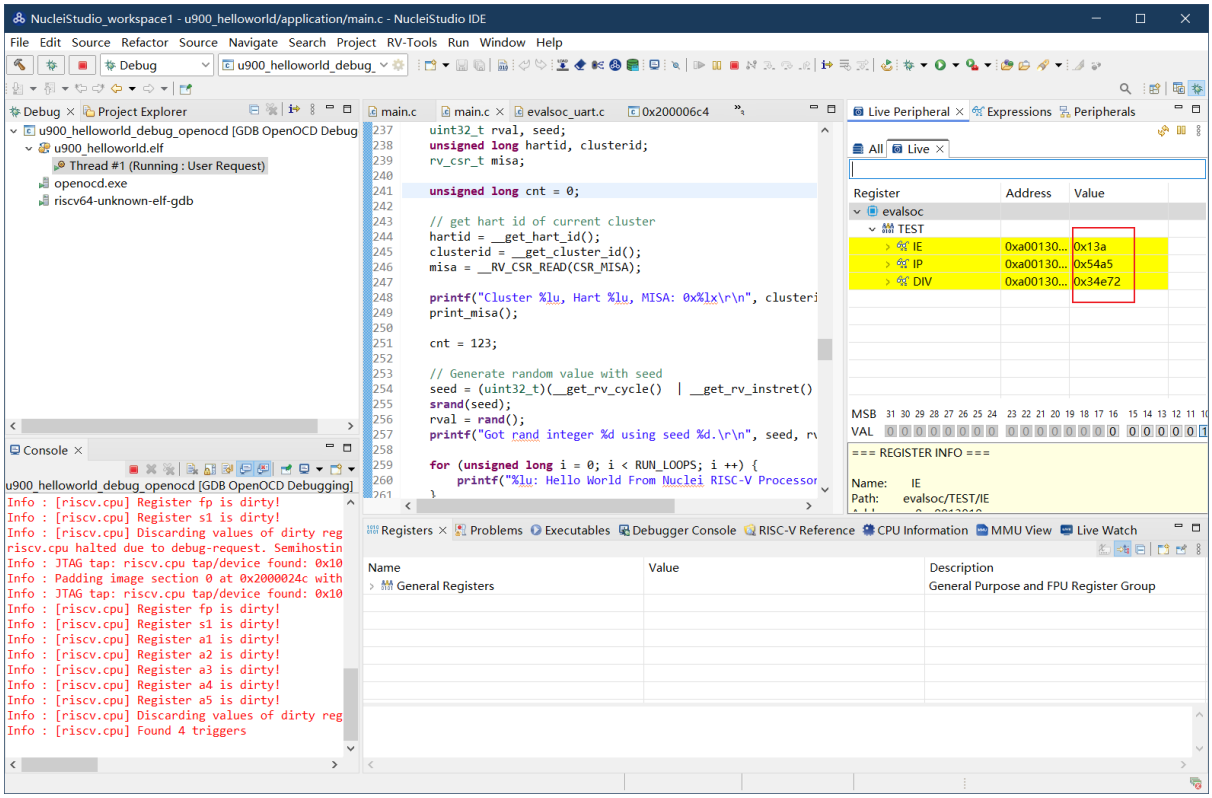
(continued from previous page)

```
* - Setting item
  - Description
* - Data source
  - GDB or Telnet. GDB is available only for OpenOCD debug configurations with non-
↪stop enabled. When non-stop is not enabled, the GDB option is greyed out and␣
↪Telnet is selected and saved automatically. To use the GDB channel, open the␣
↪OpenOCD debug configuration page and check ``Enable GDB non-stop mode to support␣
↪Live Watch``. When an old GDB is detected, it also falls back to Telnet.
* - Speed
  - Sampling speed, range 0 99999, default 10000
* - CSV save path
  - The location where sampling data is written to disk
* - Save frequency
  - The time interval for automatic saving
```

The menu also has `Refresh SVD data`, used to reload after the SVD file is updated.

Using Live Peripheral

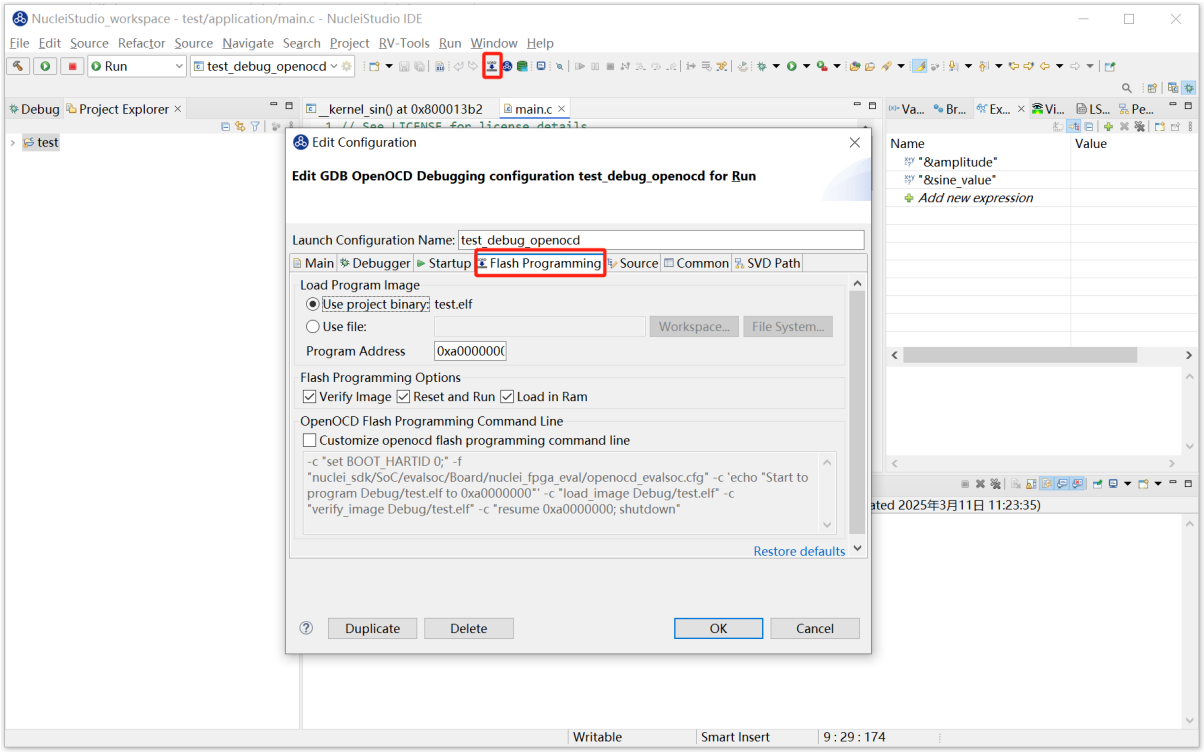
1. Confirm the SVD Path: open `Run -> Debug Configurations...` and check in the debug configuration that the SVD Path is filled in (most NPK projects fill it in automatically). A missing SVD causes an error when opening the view.
2. Start debugging and enter the Debug perspective.
3. Open the view: menu `Window -> Show View -> Debug -> Live Peripheral`.
4. Expand the peripheral tree; expanding a peripheral immediately reads the current values of the registers under it.
5. Enable Live Watch for the register(s) of interest and click `Start` on the toolbar to start real-time sampling; register values are refreshed periodically as the program runs.
6. Click `Suspend All Live Watch` when you need to pause sampling.
7. To watch certain registers for a long time: select them and use `Add to Group` to gather them into a dedicated tab; for trend analysis: right-click a register and use `Save as CSV` to export the sampling history.



2.10.15 Flash Programming

Nuclei Studio provides the Flash Programming feature for downloading compiled binary files directly to hardware development boards quickly and conveniently, greatly improving development and debugging efficiency. The operation flow is simple: after the project is built, click Flash Programming to download the binary file in one step.

For details, see *Flash Programming* (page 202).

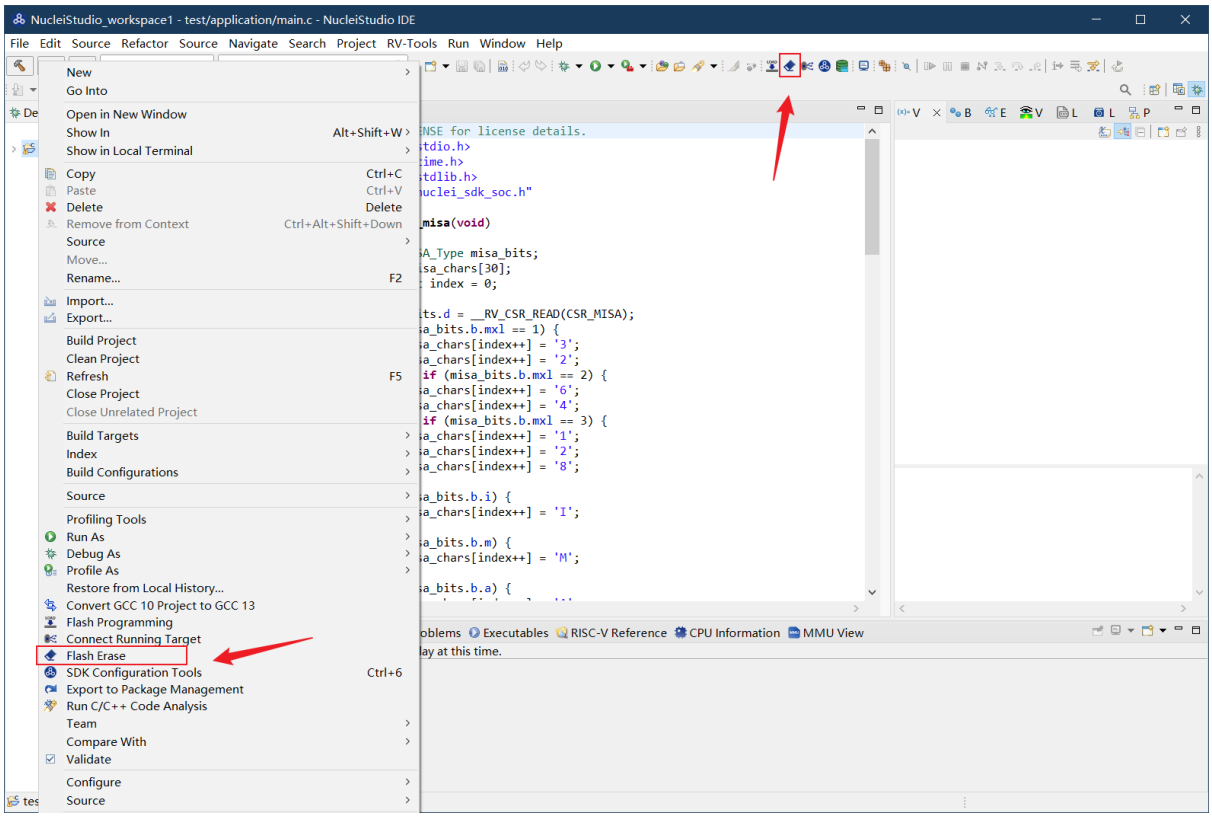


2.10.16 Flash Erase

In real development there is often a need to erase Flash independently **without starting a full debug or download session**: for example, to let the chip restart from a blank state to rule out interference from old firmware/residual data, to clear a write-protected region so it can be programmed again, or to erase only a certain sector / address range while keeping the rest. If every erase required first establishing a GDB debug connection and then exiting, it would be cumbersome and could introduce unnecessary side effects, so it is provided as an independent entry point that can be triggered from the toolbar or the right-click menu.

Flash Erase provides an entry point independent of the debug session, driving OpenOCD directly to erase the target chip ' s Flash, and exiting immediately after the erase completes (no GDB start, no program download).

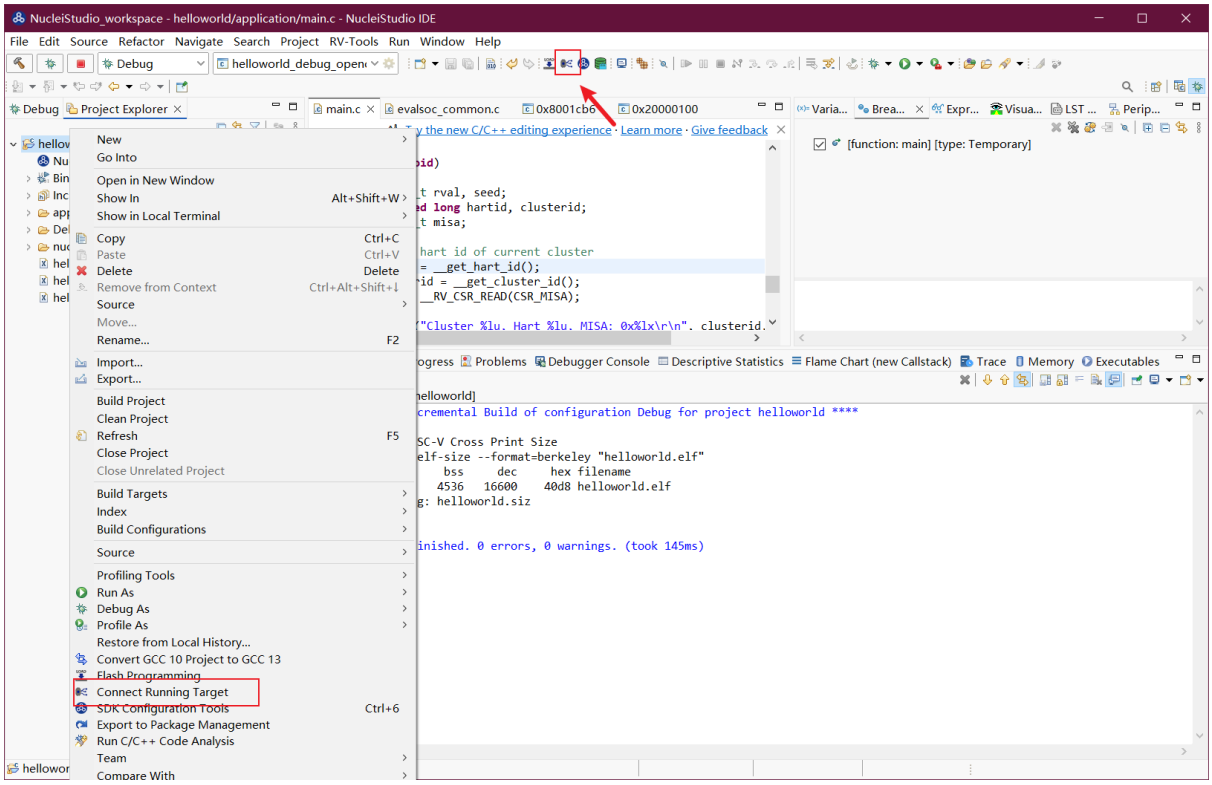
See *Flash Erase* (page 207) for details.



2.10.17 Connect to Running Target

Nuclei Studio provides the Connect to Running Target feature for connecting directly to a running development board. It connects to the running board and reads board information, greatly improving development efficiency.

For details, see *Connect to Running Target* (page 193).

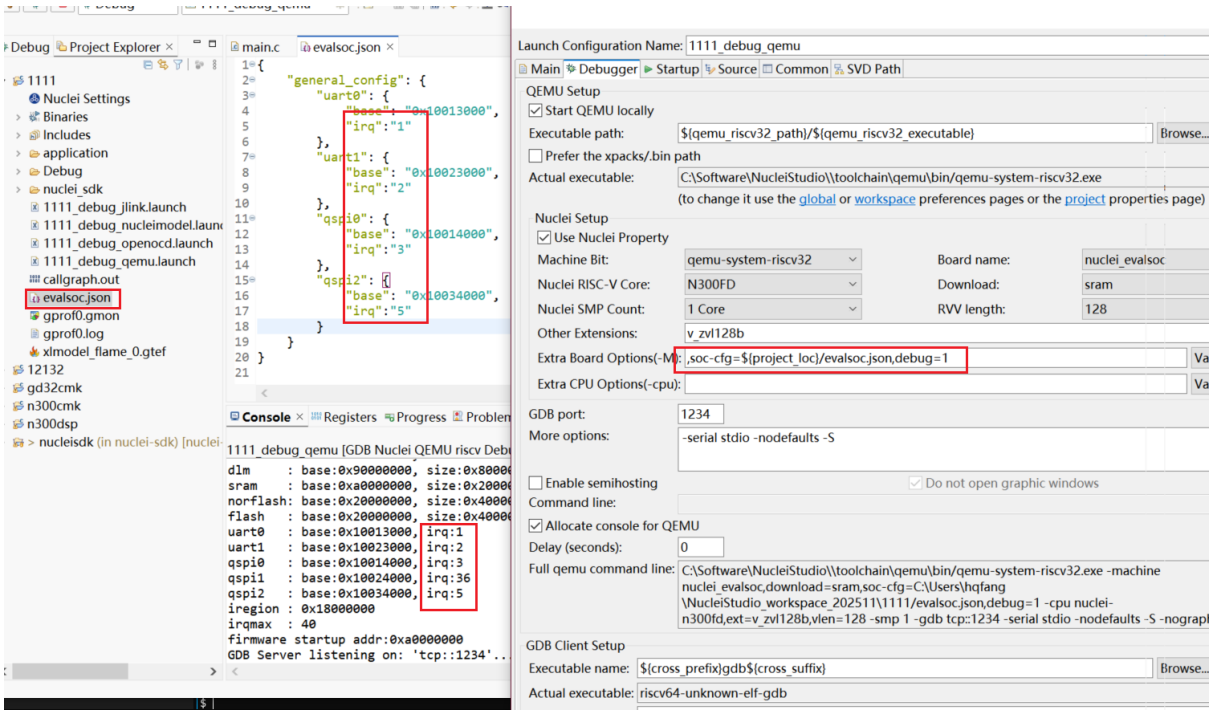


2.10.18 Updating evalsoc in Nuclei QEMU

To let you quickly modify the addresses and sizes of peripherals and the interrupt numbers in the Nuclei QEMU evalsoc —without modifying the Nuclei QEMU source code —and thus adapt to different evalsoc versions, Nuclei QEMU extends the `-M` option with the *soc-cfg* (page 842) option. You write a json configuration file and pass an option such as `-M nuclei_evalsoc,download=ddr,soc-cfg=evalsoc.json,debug=1` on the QEMU command line to modify the evalsoc peripheral configuration.

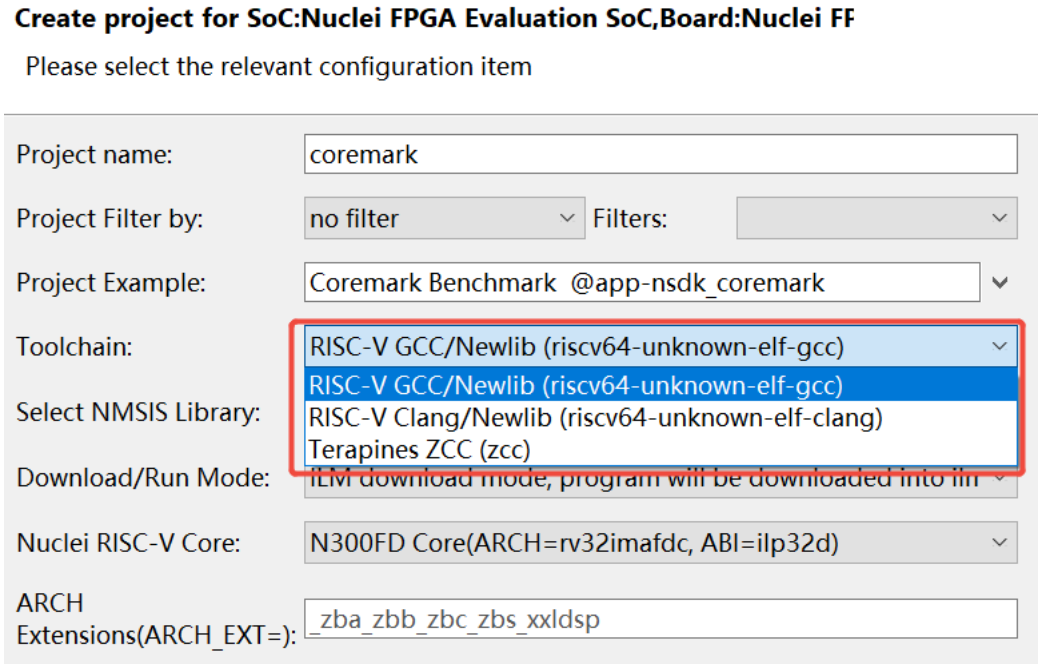
In Nuclei Studio, place the `evalsoc.json` file in the root directory of your project, then modify `Extra Board Options(-M)` on the **Debugger Tab** of the **Nuclei QEMU Debug Configuration** page. Change it to `,soc-cfg=${project_loc}/evalsoc.json,debug=1` and save the debug configuration.

You can then debug Nuclei QEMU features according to the new evalsoc configuration.



2.10.19 Switching from GCC to Clang/ZCC

Nuclei Studio IDE supports three toolchains: `RISC-V GCC/Newlib`, `RISC-V Clang/Newlib`, and `Terapines ZCC`. When creating a project, select the desired toolchain in the `Toolchain` drop-down box.

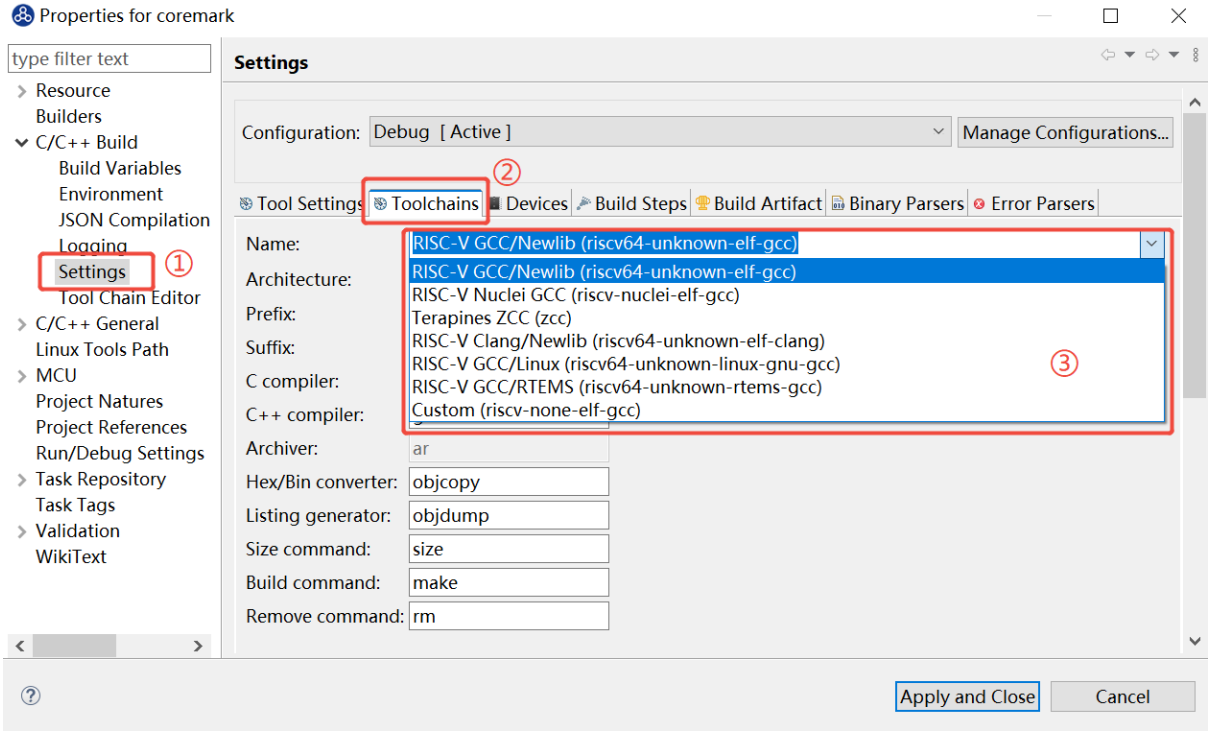


If the current project uses the GCC toolchain and you want to switch to the Clang or ZCC toolchain, you can complete the migration with the following steps:

1. *Switching the Toolchain* (page 330);
2. *Adjusting Compile and Link Options* (page 330);
3. *Adjusting Linked Libraries* (page 331);

Switching the Toolchain

Select the project in `Project Explorer`, right-click and select `Properties` to open the project properties, then go to `C/C++ Build -> Settings -> Toolchains` and select the target toolchain in the `Name` drop-down box.



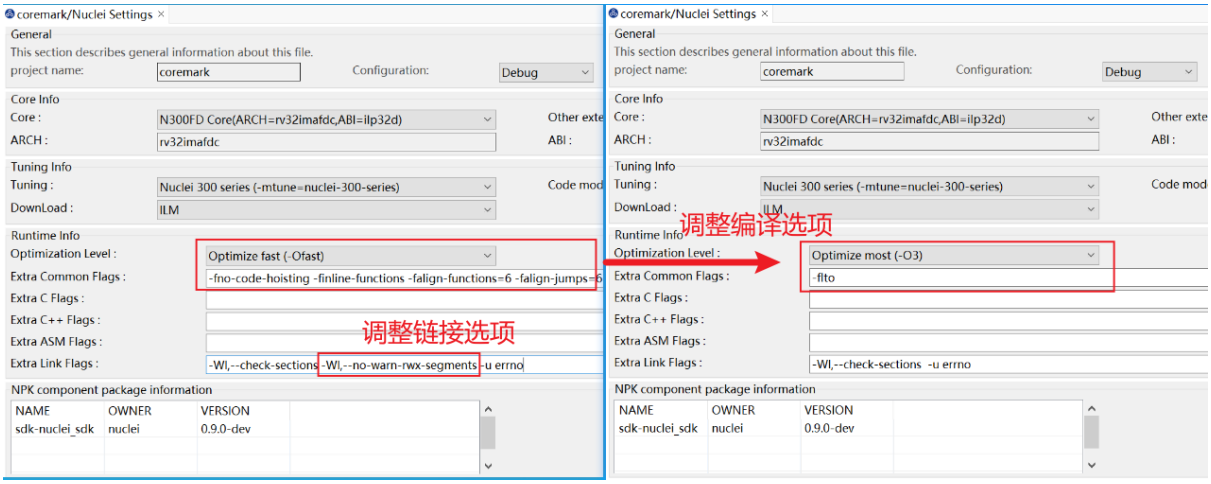
Adjusting Compile and Link Options

RISC-V Clang/Newlib

If you switch to the Clang toolchain, you need to adjust the compile and link options accordingly, because Clang is not fully compatible with all GCC parameters.

Taking the CoreMark project as an example, the parameters used by different CPU series may vary slightly, but the following adjustments are generally recommended:

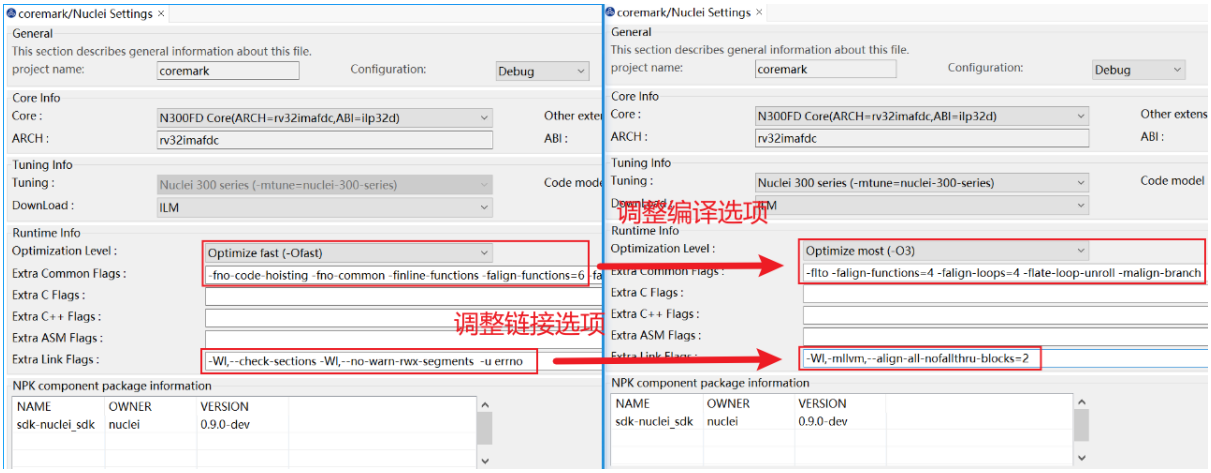
1. Change `Optimization Level` from `-Ofast` to `-O3`;
2. Replace `Extra Common Flags` with `-flto`;
3. Remove `-Wl,--no-warn-rwx-segments` from the link options.



Terapines ZCC

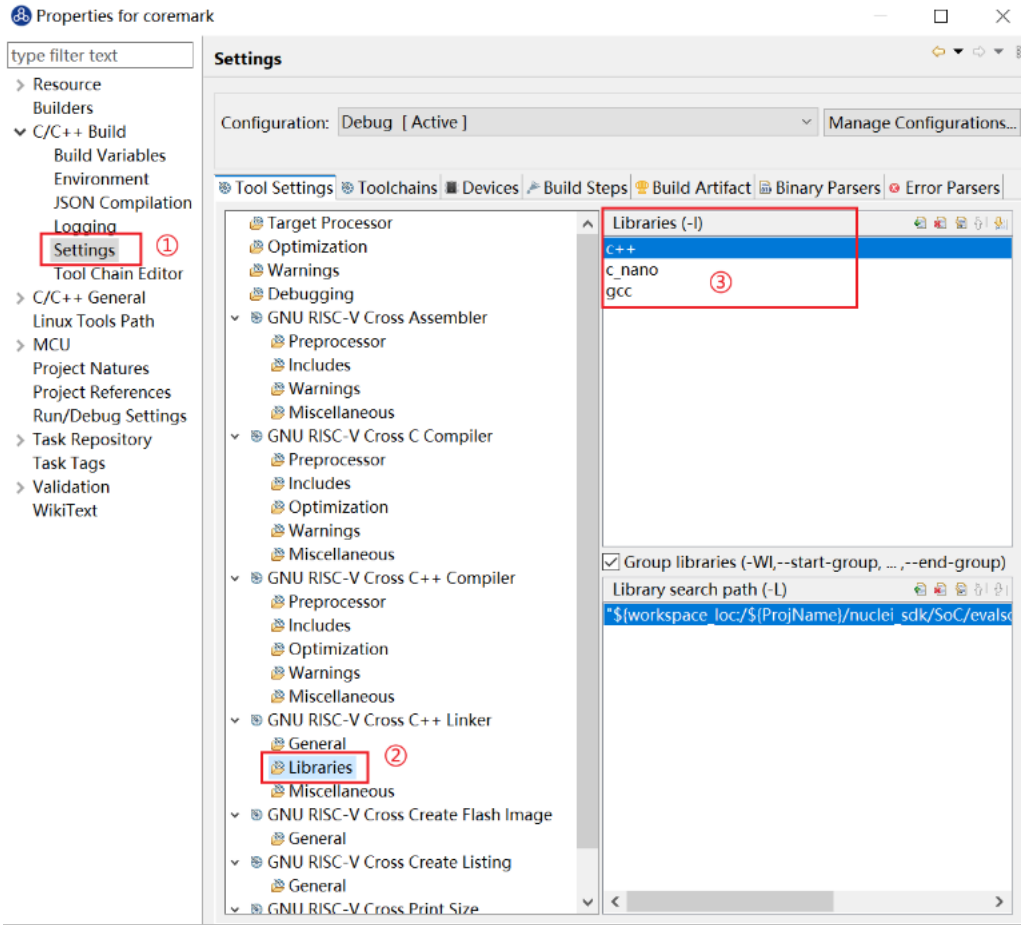
If you switch to the ZCC toolchain, it is recommended to adjust the compile and link options as follows:

1. Change `Optimization Level` from `-Ofast` to `-O3`;
2. Replace `Extra Common Flags` with `-flto -falign-functions=4 -falign-loops=4 -flate-loop-unroll -malign-branch`;
3. Remove `-Wl,--no-warn-rwx-segments` from the link options;
4. Change the link options to `-Wl,-mllvm,--align-all-nofallthru-blocks=2`.



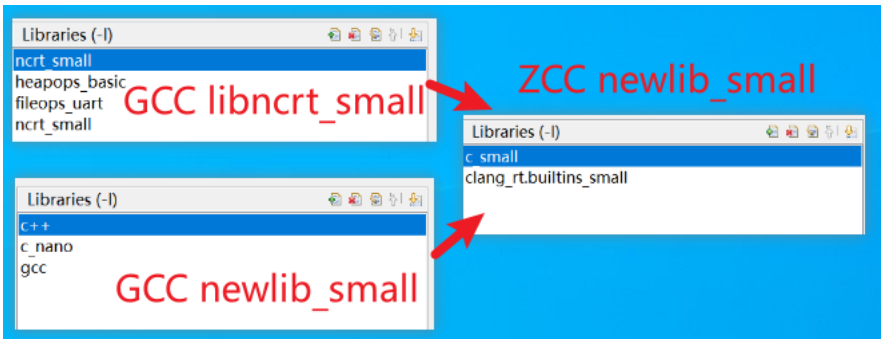
Adjusting Linked Libraries

Different toolchains may depend on different libraries, so you also need to check and adjust the linked libraries after switching. The linked libraries can be modified in the project properties under `C/C++ Build -> Settings -> Tool Settings -> GNU RISC-V Cross C++ Linker -> Libraries`.



For the Clang toolchain, Nuclei Studio still uses the same C standard library, and generally no additional modification of the linked libraries is required. Therefore, when migrating from GCC to Clang, you usually only need to complete the previous two steps, *Switching the Toolchain* (page 330) and *Adjusting Compile and Link Options* (page 330), and then try to rebuild the project.

For the ZCC toolchain, the linked library names must also be adjusted. ZCC only supports the `newlib_small` library, so whether the current GCC project uses `newlib_small` or `libncrt_small`, change the linked libraries to `c_small` and `clang_rt.builtins_small`.



If the current project links `libncrt_small`, you also need to replace `stubs.c` with the newlib version; if the project originally uses the newlib library, this step is not needed.

For example, in an EvalSoC project created based on Nuclei SDK, first exclude the libncrt version of `stubs.c` from the build: select `SoC\evalsoc\Common\Source\Stubs\libncrt`, right-click and select `Resource Configurations -> Exclude from Build...`, then copy a newlib version of `stubs.c` from another project that uses the newlib library into the `Stubs` directory.

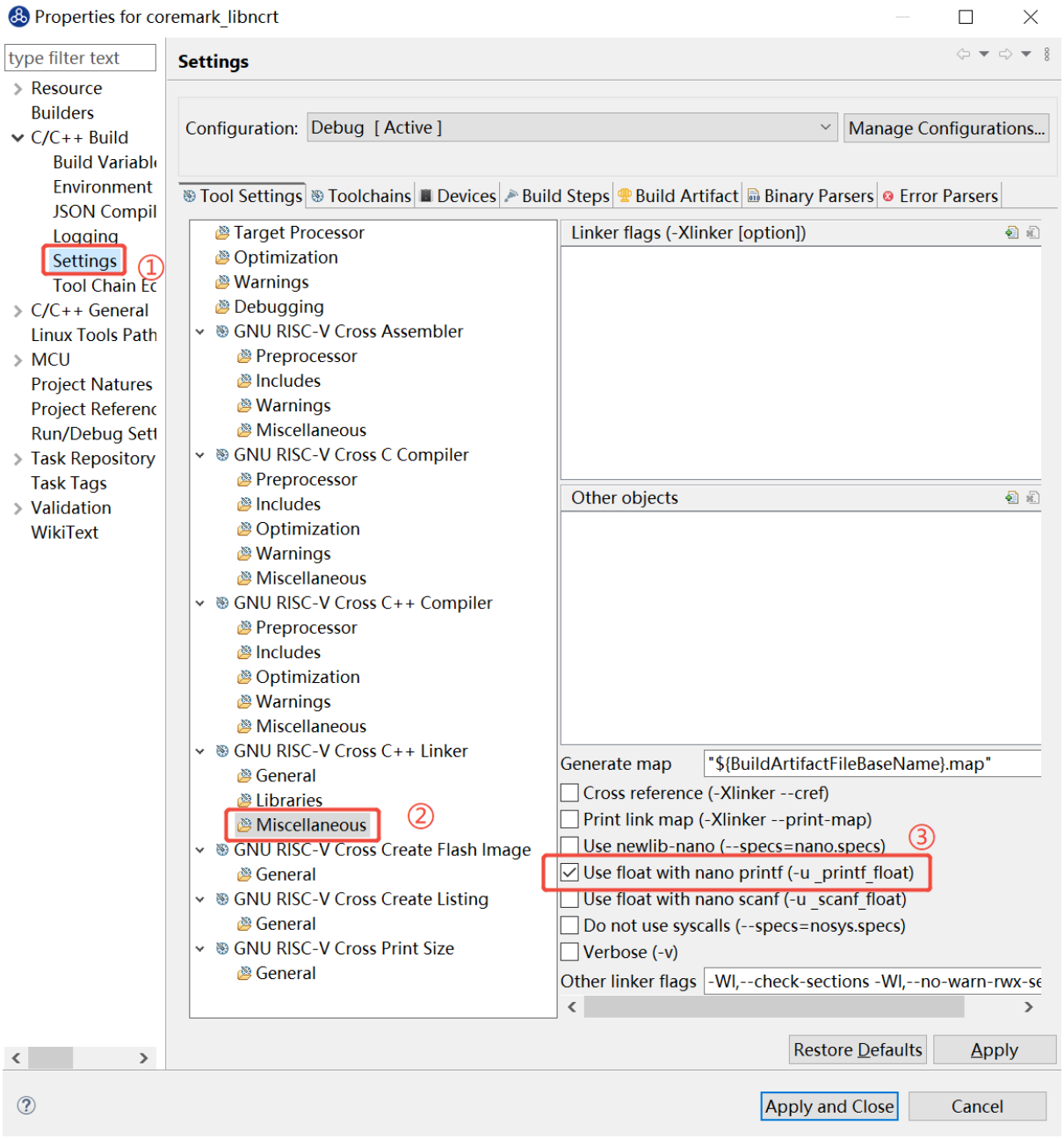


After completing the `stubs.c` adjustment, the migration from GCC to ZCC is complete, and you can try to rebuild the project.

Converting Between newlib_small and libncrt_small

As mentioned earlier, ZCC does not support `libncrt_small`. Therefore, when switching the toolchain from GCC to ZCC, a project that originally uses `libncrt_small` must also be switched to `newlib_small`.

When `newlib_small` is selected in Nuclei Studio, the actual linked library is `c_nano` and floating-point printing is supported. `-u _printf_float` is added to the link options by default. You can check whether this option has been added as shown in the figure below.



The table below summarizes the supported toolchain and C library combinations. When adjusting the toolchain or linked libraries, also check that the `Library List`, `Link Options`, and `stubs File` match.

Toolchain	Linked Library	Supported	Library List	Link Options	stubs File
GCC/Clang	newlib_small	Supported	• c++ • c_nano • gcc	-u _printf_float	newlib/stubs.c
GCC/Clang	libncrt_small	Supported	• ncrt_small • heapops_ba • fileops_uart		libncrt/stubs.c
ZCC	newlib_small	Supported	• c_small • clang_rt.bu		newlib/stubs.c
ZCC	libncrt_small	Not Supported			

Common Issues

Clang/ZCC Incompatibility with GCC

The Clang/ZCC toolchain is not fully compatible with GCC compile options. If errors are reported during compilation, refer to *Adjusting Compile and Link Options* (page 330) for adjustments.

```
riscv64-unknown-elf-clang: error: unknown argument: '-fno-code-hoisting'
riscv64-unknown-elf-clang: error: unknown argument: '-fno-if-conversion'
riscv64-unknown-elf-clang: error: unknown argument: '-fno-if-conversion2'
riscv64-unknown-elf-clang: error: unknown argument: '-fselective-scheduling'
riscv64-unknown-elf-clang: error: unknown argument: '-fno-tree-loop-distribute-
↪patterns'
riscv64-unknown-elf-clang: error: unknown argument: '-mbranch-cost=1'
```

The Clang/ZCC toolchain does not support the `-Wl,--no-warn-rwx-segments` link option. If errors are reported during linking, refer to *Adjusting Compile and Link Options* (page 330) for adjustments.

```
ld.lld: error: unknown argument '--no-warn-rwx-segments'
riscv64-unknown-elf-clang++: error: ld command failed with exit code 1 (use -v to␣
↪see invocation)
make: *** [makefile:92: coremark.elf] Error 1
```

ZCC Link Failures

When switching from GCC to the ZCC toolchain, the following problems may occur if the linked libraries are not adjusted correctly. Refer to *Adjusting Linked Libraries* (page 331) for modifications.

If the original project uses `newlib_small` but the linked libraries are not modified, the following error may occur:

```
ld.lld: error: unable to find library -lstdc++
z++: error: ld command failed with exit code 1 (use -v to see invocation)
make: *** [makefile:92: coremark.elf] Error 1
```

If the original project uses `libncrt_small` but the linked libraries are not modified, the following error may occur:

```
ld.lld: error: unable to find library -lncrt_small
ld.lld: error: unable to find library -lheapops_basic
ld.lld: error: unable to find library -lfileops_uart
ld.lld: error: unable to find library -lncrt_small
z++: error: ld command failed with exit code 1 (use -v to see invocation)
make: *** [makefile:92: coremark.elf] Error 1
```

If the original project uses `libncrt_small` and the linked libraries have been modified, but `stubs.c` has not been replaced accordingly, the following error may occur:

```
ld.lld: error: undefined symbol: _read
>>> referenced by ld-temp.o
>>>               coremark.elf.lto.o:(sys_semihost_getc)

ld.lld: error: undefined symbol: _write
>>> referenced by ld-temp.o
>>>               coremark.elf.lto.o:(sys_semihost_putc)
z++: error: ld command failed with exit code 1 (use -v to see invocation)
make: *** [makefile:92: coremark.elf] Error 1
```

2.10.20 CPU Information

CPU Information targets Nuclei-series RISC-V cores and provides two tabs, CPU Information and CPU Fault Analysis. It lets you inspect `MARCHID / MHARTID / MIMPID / ISA / Version` per hart, and decodes the `MISA` extension bits and Nuclei configuration CSRs (`mcfg_info` / `micfg_info` / `mdcfg_info`) bit by bit; CPU Fault Analysis can also locate the cause of a trap (current mode, `mcause` / `scause`, trap vector, M/S-mode CSRs, page faults, etc.).

See *CPU Information* (page 195) for details.

2.10.21 MMU View

The RISC-V MMU View is a view under the debug perspective, used to read the `satp` / `misa` CSRs and walk the page table while the target is halted, providing:

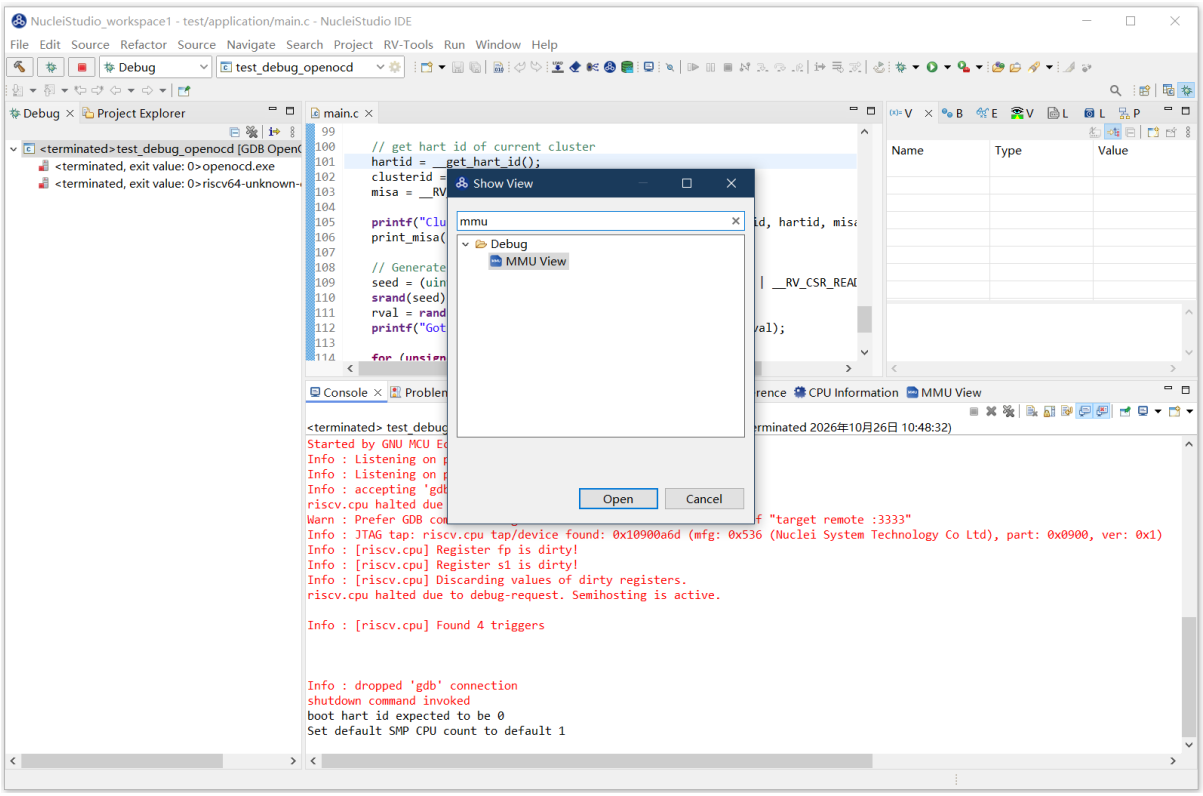
1. Address Translation: virtual address → physical address translation, showing the page-table walk path level by level;
2. Memory Map: folding the whole page table into a series of VA→PA ranges;
3. Page Tables: expanding page-table descriptors as a multi-level tree;
4. MMU Settings dialog: it can follow the hardware `satp`, or inject a custom `satp` for offline analysis.

This view is read-only and writes no registers. The only hardware prerequisite: the core must implement S-mode (supervisor mode); otherwise the `satp` register does not exist at all. There are three ways to tell, which can corroborate each other:

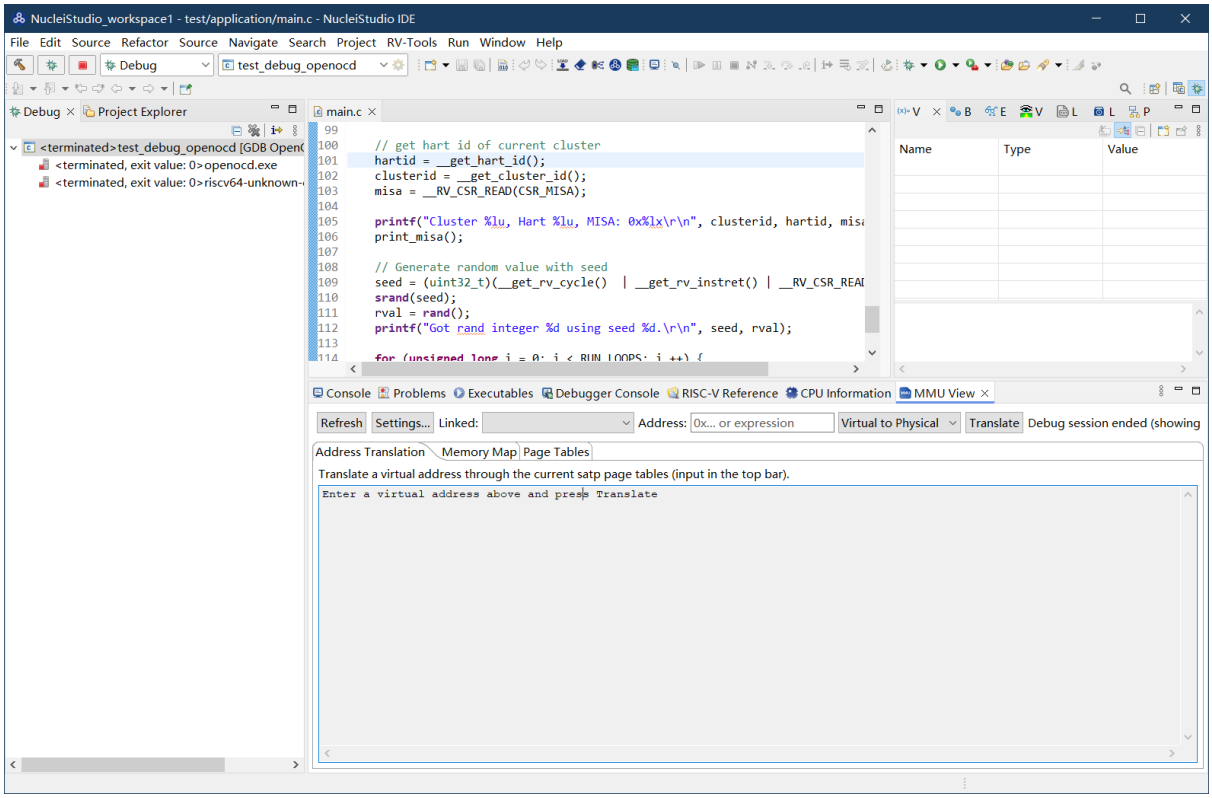
- Whether the ISA string contains S (e.g., `RV64ABCDFIMSU` contains S);
- More reliably: look at bit18 of `misa`;
- The Feature list explicitly shows SMODE.

Important reminder: none of the above three is a sufficient condition. In real tests there have been cases where the ISA contains S and misa bit18 is also 1, but `csrw satp` still triggers an illegal instruction exception (the hardware satp is always 0). Ultimately, the criterion is whether `satp` can be successfully read and written. If bit18 = 0 (for example, Nuclei N300, an MCU core that defaults to M/U + PMP), there is no software way to enable the MMU; you can only switch cores (such as UX600 / UX900) or change the bitstream to enable SMODE.

In the menu `Window -> Show View -> Other...`, search for `MMU` to open the view.



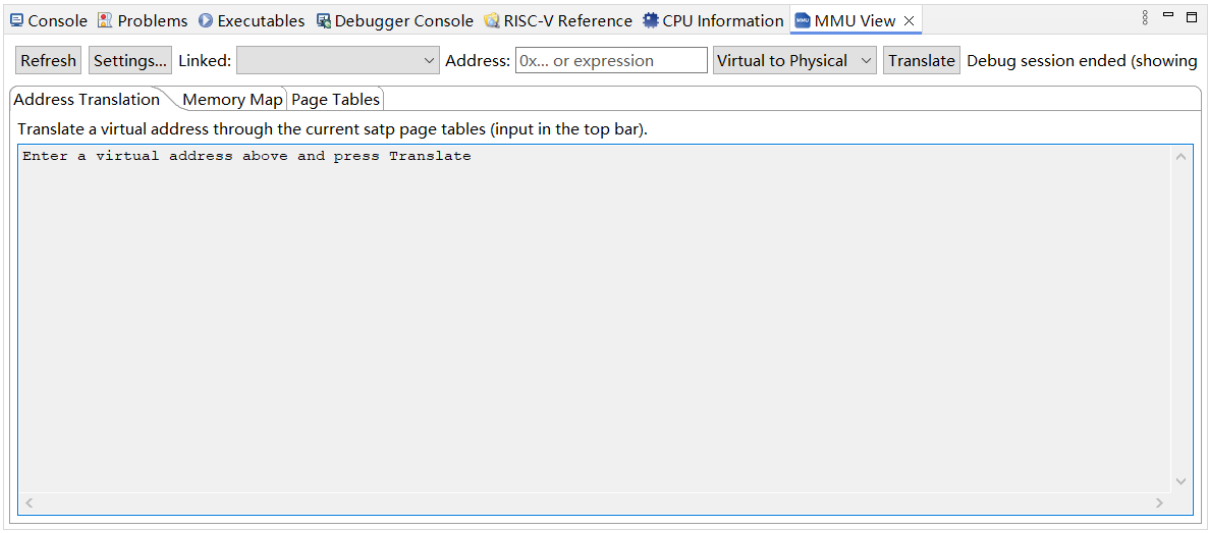
Control	Function
Refresh	Read satp / misa and load the root page table
Settings…	Open the MMU Settings dialog
Linked	hart drop-down selection. On multicore it shows hart N (GDB Thread M); switching hart clears and refreshes again
Address	Address input box, accepts a hexadecimal address or a GDB-evaluable expression
Direction drop-down	Virtual to Physical or Physical to Virtual
Translate	Execute the translation
statusLabel	Shows the current status or error message



The buttons remain clickable in all states; on failure a clear error message is given (for example, if the target is running, it prompts “please suspend first”).

MMU View Introduction

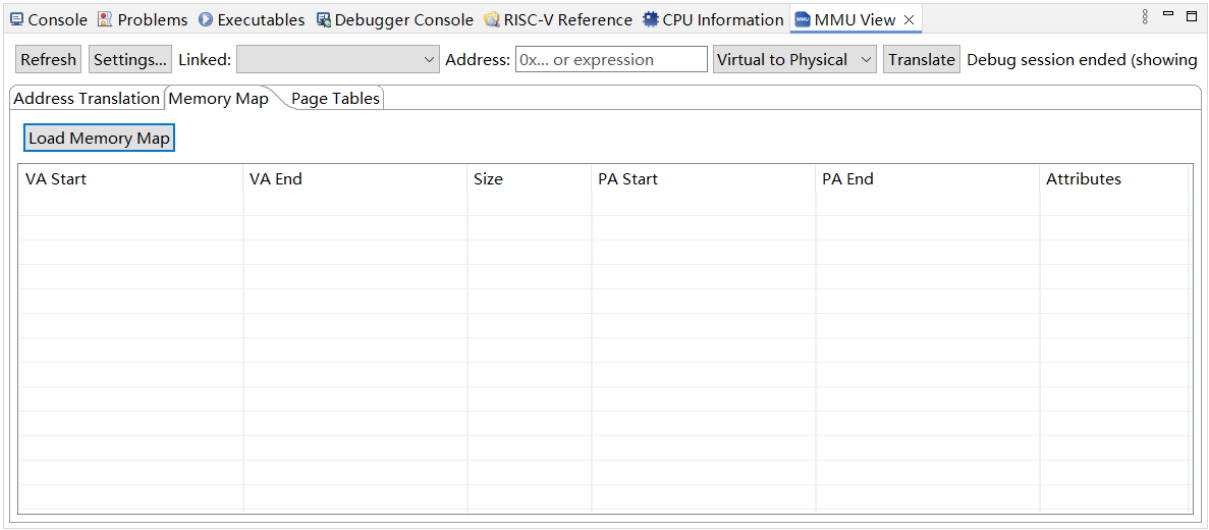
Address Translation



Direction	Behavior
Virtual to Physical	Enter a VA (or GDB expression) → page-table walk runs in the background → the result area shows the level-by-level walk path + translation result + Access Check (whether fetch/read/write are OK or FAULT, with the effect of MXR noted)
Physical to Virtual	Enter a PA → full-table scan runs in the background → lists all VAs mapped to that PA (including VA range, PA range, page size, level, flags), up to 512 results

For both directions, you can right-click in the result area and choose Open PA in Memory View to jump to the physical address. The address input supports expressions; you can directly fill in a symbol name.

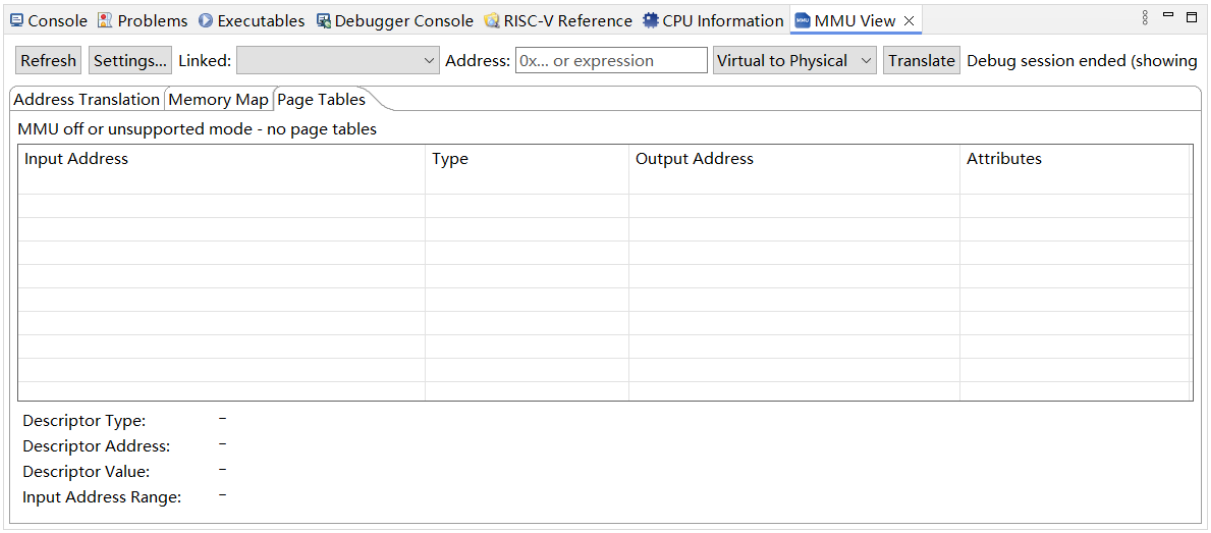
Memory Map



After clicking `Load Memory Map`, the full table is built in the background. Table columns: VA Start / VA End / Size / PA Start / PA End / Attributes. Unmapped ranges are shown as gray `<unmapped>` rows. The status bar shows statistics such as entries / tables / truncated / cancelled / readErrors.

Merge rule: two adjacent items are merged into one row only when they simultaneously satisfy VA continuity, PA continuity, same level, same permission flags, and same memory attributes (PBMT/N/RSW). The unit of the Size column is the largest readable unit (an integer for whole multiples, one decimal place for non-whole multiples).

Page Tables



The page table is expanded as a tree, with columns Input Address / Type / Output Address / Attributes. The next level is read in the background when a node is expanded.

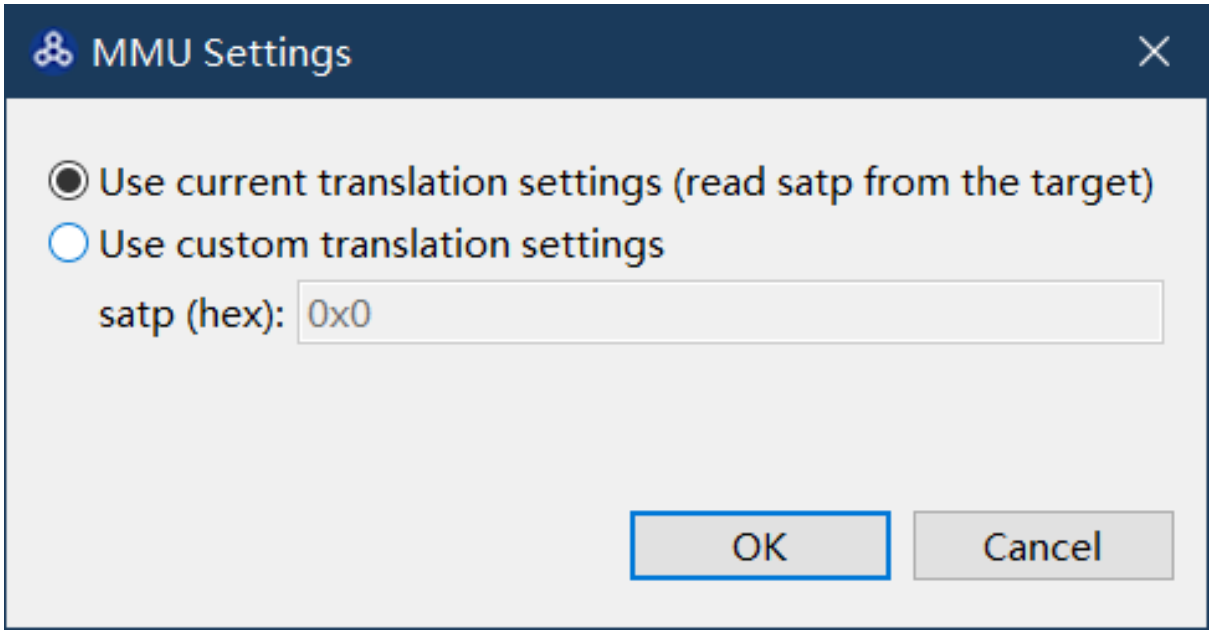
Type is divided into three kinds:

Type	Meaning
Page Table	Page-table pointer, pointing to the next level
N Page	Leaf page-table entry, the actual mapping
Fault (xN)	Fault node aggregated from consecutive invalid PTEs

The Output Address column has different meaning by node type: a leaf (N Page) shows the physical base address of that mapping; a pointer (Page Table) shows `-> 0x...` (the next-level page-table address, prefixed with `->` to avoid being misread as translation output); a root node / Fault shows `-`.

After selecting a node in the tree, the lower panel shows the descriptor details (Descriptor Type / Address / Value / Input Address Range / Next-level Table Address / Output Attributes).

MMU Settings Dialog



Option	Description
Use current translation settings	Read the hardware satp (default)
Use custom translation settings	Inject a custom satp for offline analysis, without modifying the hardware CSR

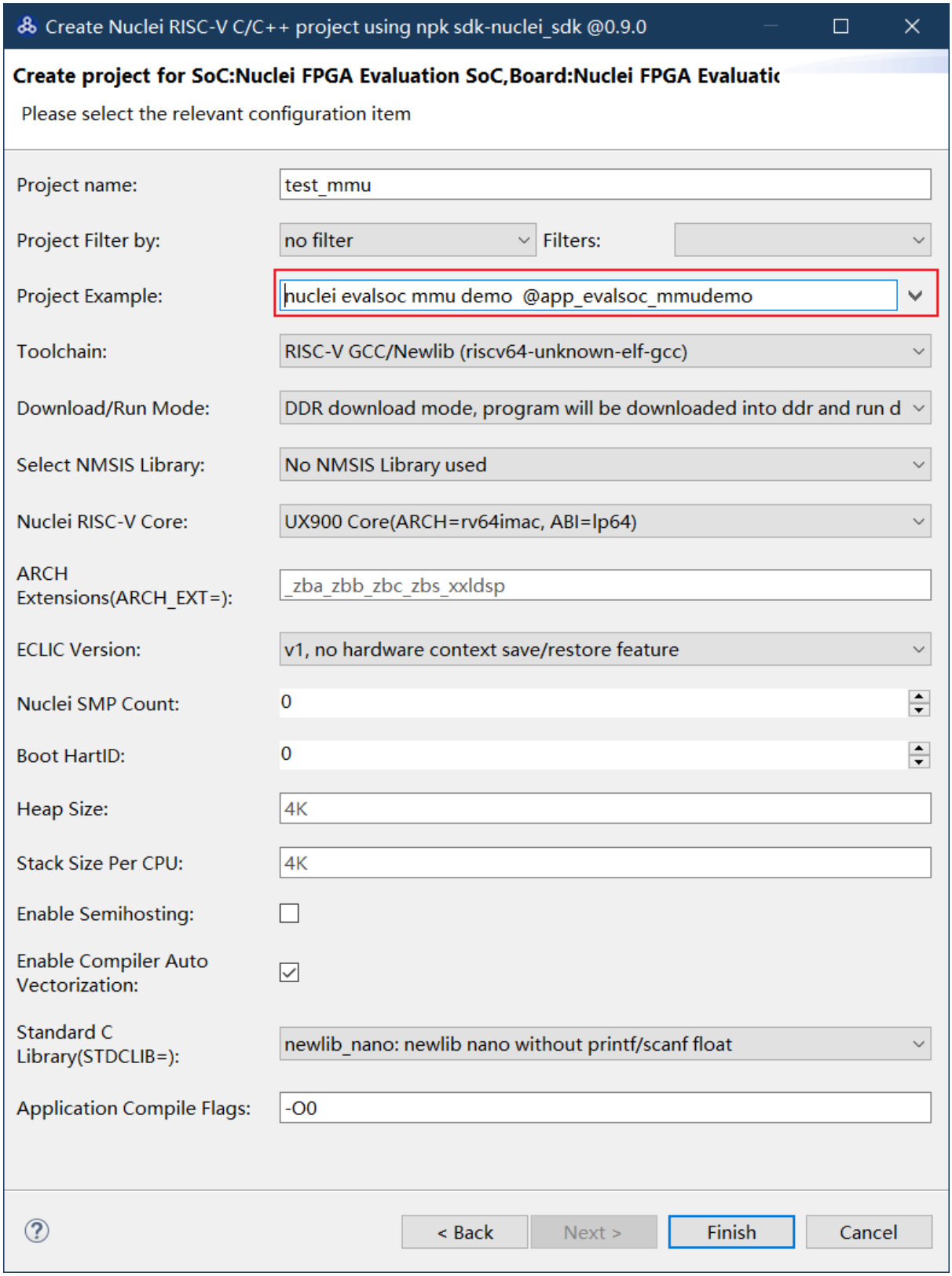
After switching, the view is cleared and refreshed again.

Practical tip: when the target firmware does not enable paging (`satp.MODE = Bare`), this dialog is the main means to make the view work —as long as a valid page table exists in memory, filling in the corresponding `satp` value lets you traverse it completely, without the hardware actually enabling the MMU.

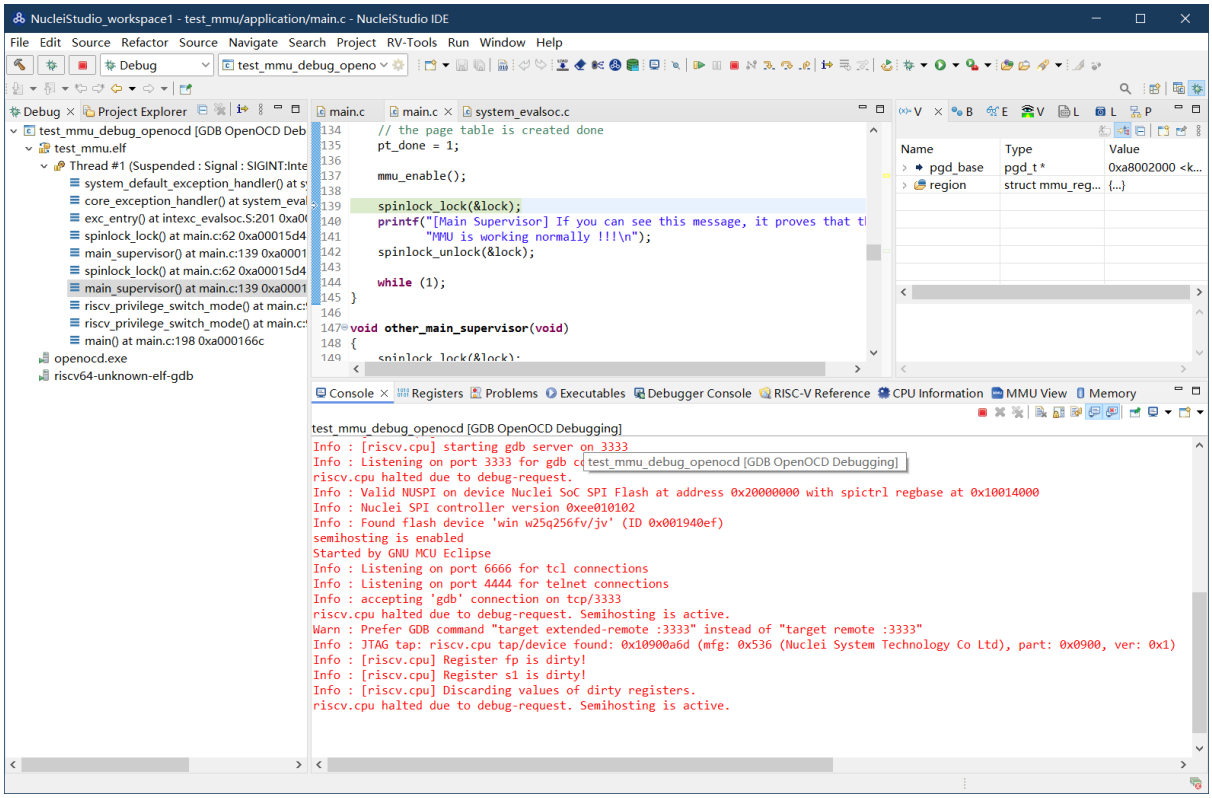
MMU View Usage Example

Before using MMU View, confirm that the core supports S-mode and that satp can be read and written successfully. The runtime verification of MMU View was done on a Nuclei UX900 (RV64) + OpenOCD.

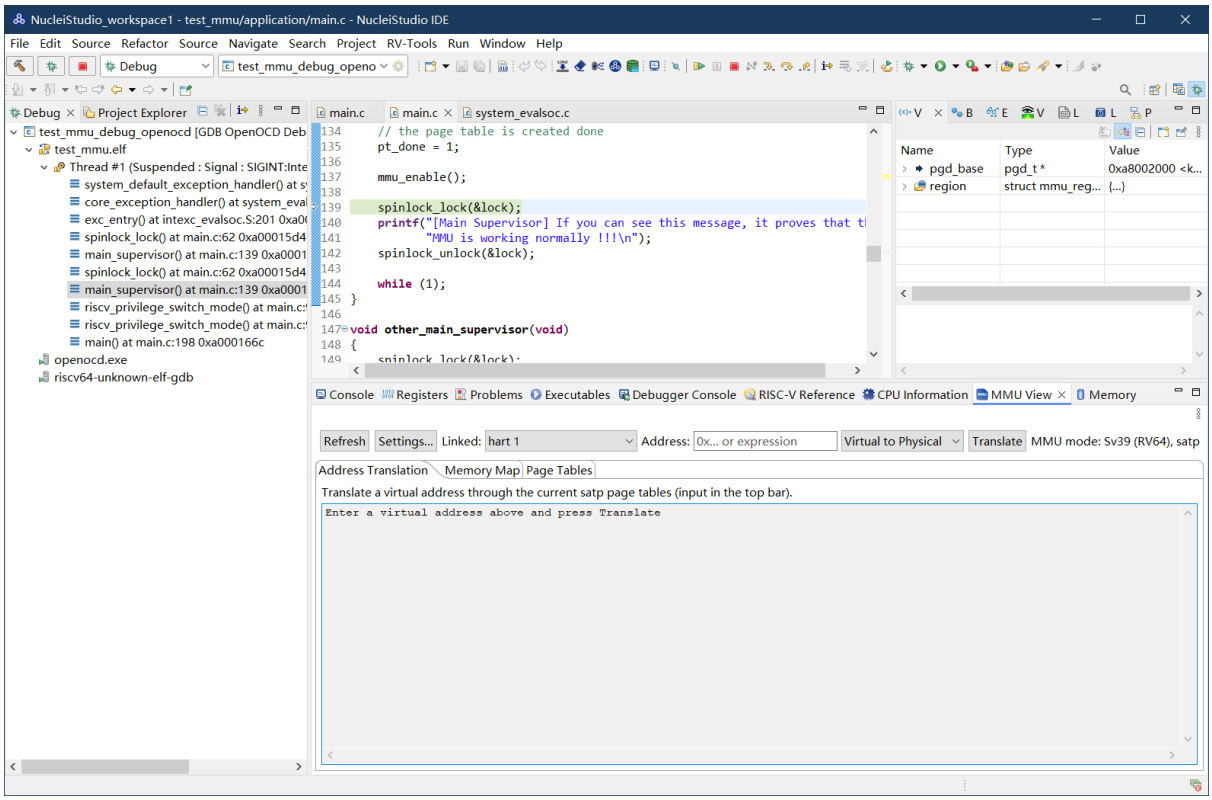
1. Confirm the core supports S-mode (it is recommended to verify read/write with `csrw satp`). MCU cores like N300 that are only M/U + PMP have no MMU and cannot be used.



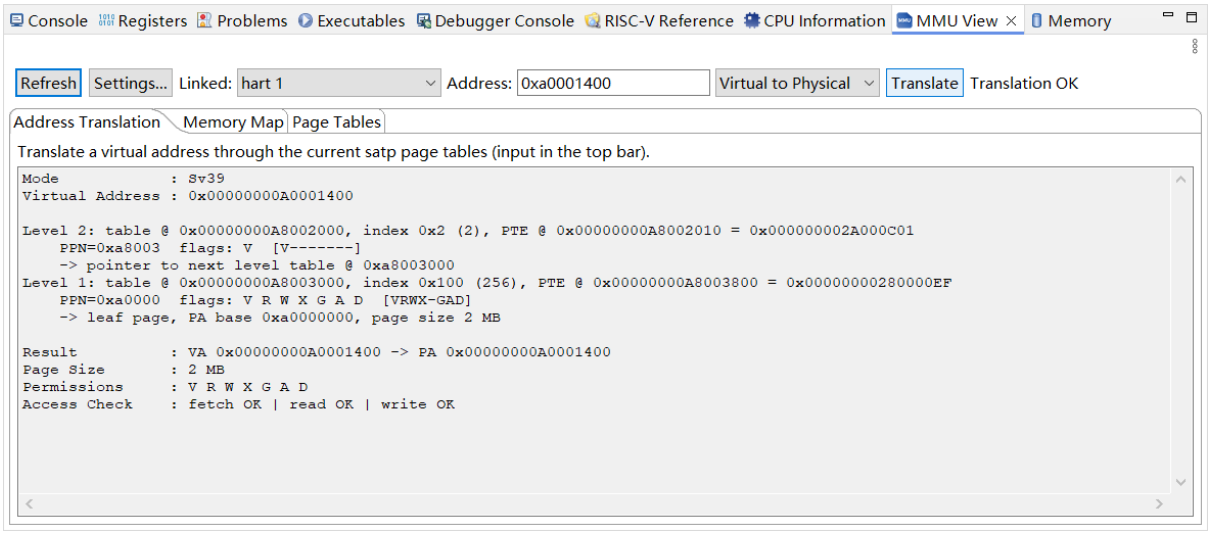
2. Start debugging and let the target halt.



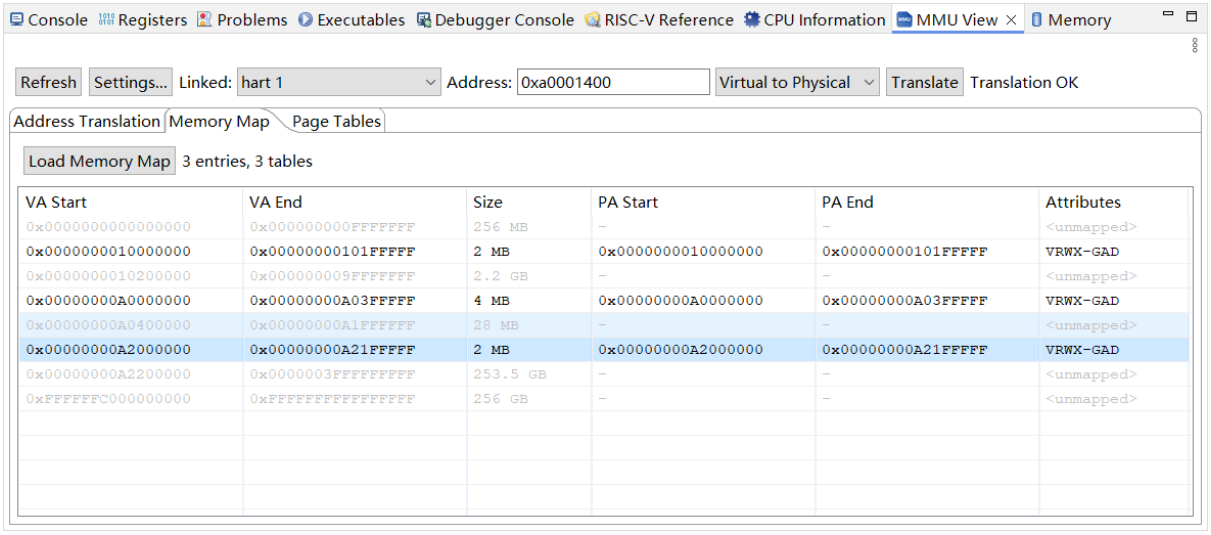
3. Open the view: menu `Window -> Show View -> Other... -> MMU`. The view automatically reads satp / misa and loads the root page table.



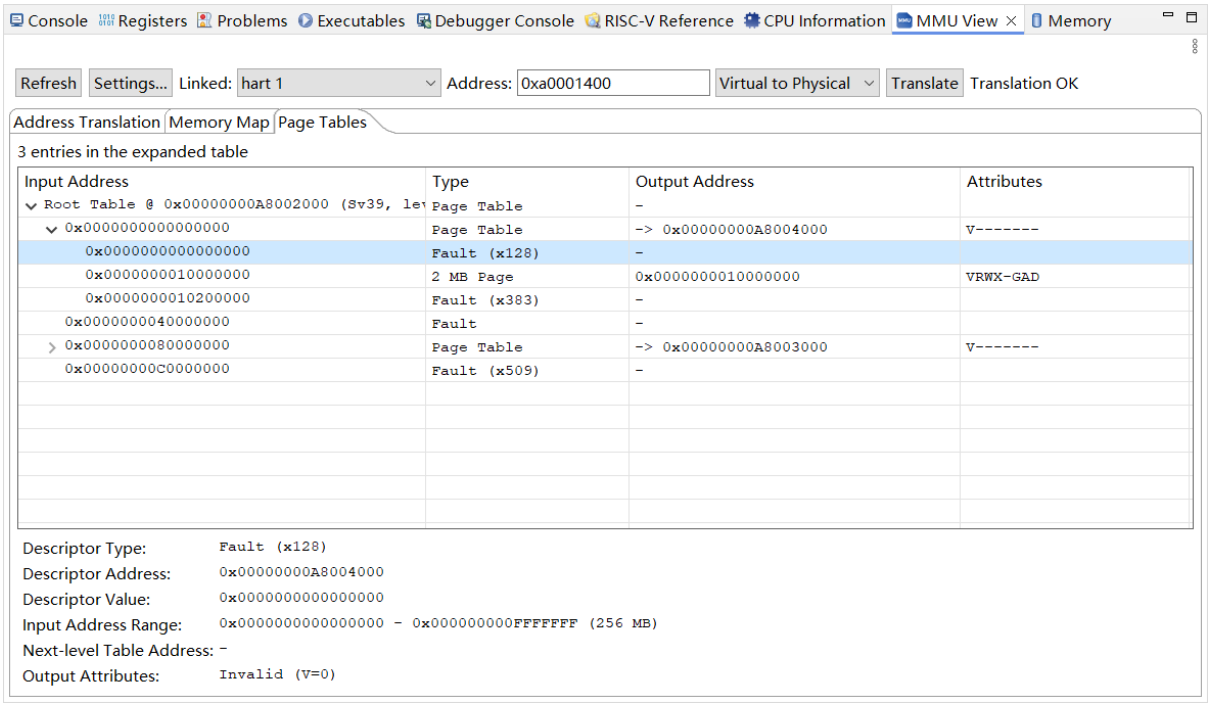
4. Look up a single address: fill the virtual address (supports symbol name / GDB expression) in the top Address input box, choose Virtual to Physical as the direction, and click Translate; the result area shows the level-by-level page-table walk path and Access Check.



5. View the overall mapping: switch to the Memory Map tab and click Load Memory Map.



6. View the page-table structure: switch to the Page Tables tab and expand level by level; select a node to see Descriptor Details; note that the −> prefix on a pointer node indicates the next-level page-table address.



Troubleshooting

Symptom	Cause and handling
Prompt "please suspend first"	The target is running. MMU View only reads while halted; Suspend first or hit a breakpoint
Pop-up "MMU View Unavailable" dialog	Runtime authoritative criterion: the current core misa bit 18 (S-mode) is 0, meaning the core has no MMU (e.g., N300 / N200 series are only M/U + PMP). Switch to a core that implements S-mode (such as N900 / UX900). The "custom satp offline analysis" in MMU Settings can still be used
Debug launch immediately prompts "project core N300 typically has no MMU"	Project-level soft hint (non-blocking): an early warning when the core selected by the project belongs to a known MMU-less series; ultimately the runtime misa S bit is still authoritative
Cannot read satp / view is empty	The core does not implement S-mode, or the hardware satp is always 0
Many Fault nodes in the page-table tree	The corresponding range being unmapped is normal; you can turn off Coalesce Invalid Entries to inspect entry by entry

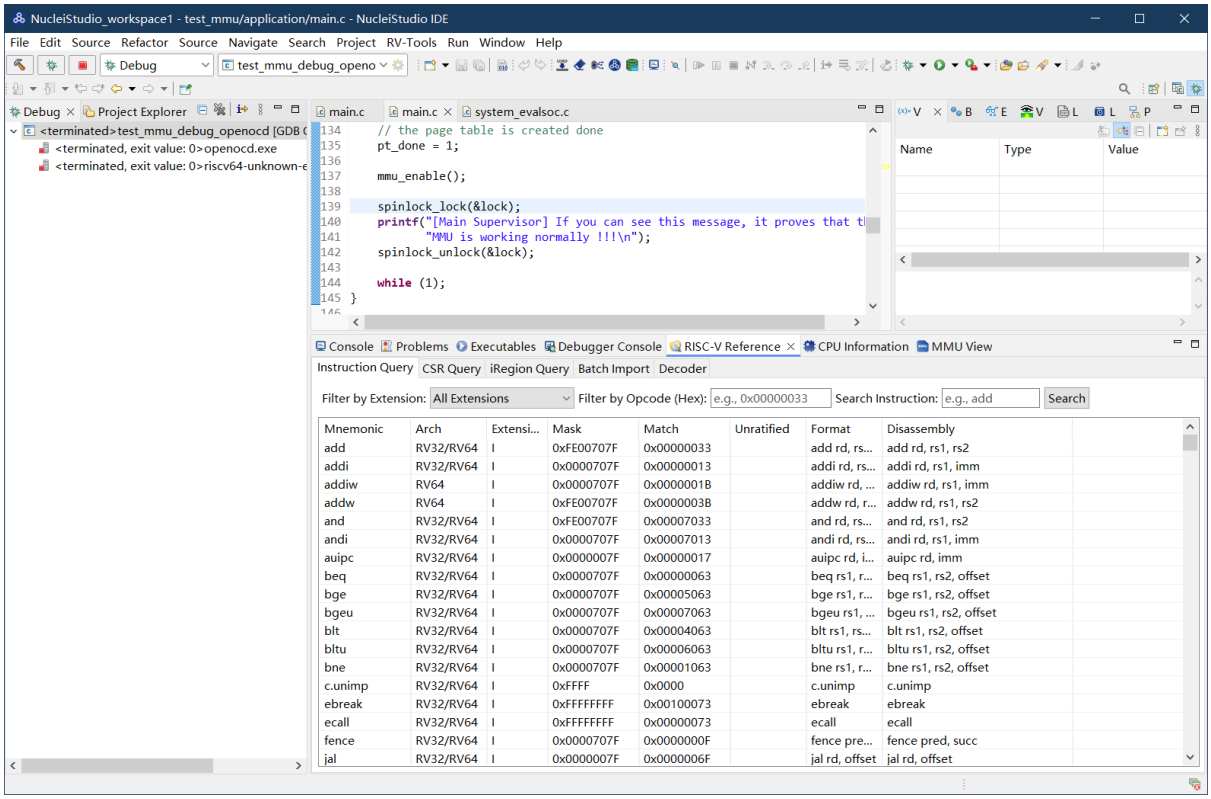
2.10.22 RISC-V Reference

RISC-V Reference is a tool view that combines four capabilities:

1. Instruction / CSR / iRegion reference browser: search by extension, opcode, mnemonic, address, or name;
2. On-demand disassembly / decoding: decode hexadecimal machine code or assembly mnemonic into assembly and show it bit by bit, supporting both single and batch;
3. Register value bit-field interpretation: select a register from the Registers / Variables view to back-fill its current value, and break it down by bit field;
4. Remote data update: incrementally pull instruction/CSR/iRegion data from the server and cache it locally.

From the menu `Window -> Show View -> Other...`, search for `RISC-V Reference`. It is stacked by default in the Debug and C/C++ perspectives (in the same stack as the Console view).

RISC-V Reference Introduction

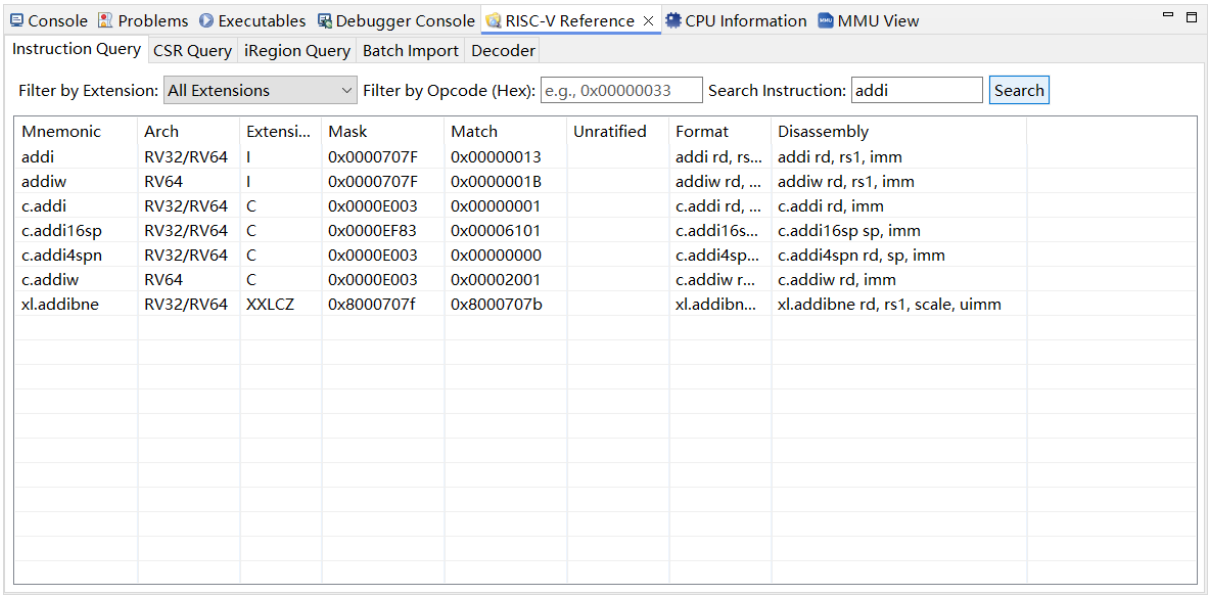


Tab	Purpose	Description
Instruction Query	Query instructions	Search by extension / opcode / mnemonic. Columns include Mnemonic / Arch / Extension / Mask / Match / Unratified / Format / Disassembly
CSR Query	Query CSRs	Search by address or name. Upper and lower lists + bit-field table; you can enter the current value and non-zero fields are highlighted
iRegion Query	Query peripheral registers	Search memory-mapped registers by module / register name and do bit-field decomposition
Batch Import	Batch import	Paste name=value text, parse and match to CSR / iRegion, then jump to or import per checked item
Decoder	Decode machine code	Hexadecimal machine code (or assembly) → decode table, 13 columns in total (see below for the meaning of each column)

Using RISC-V Reference

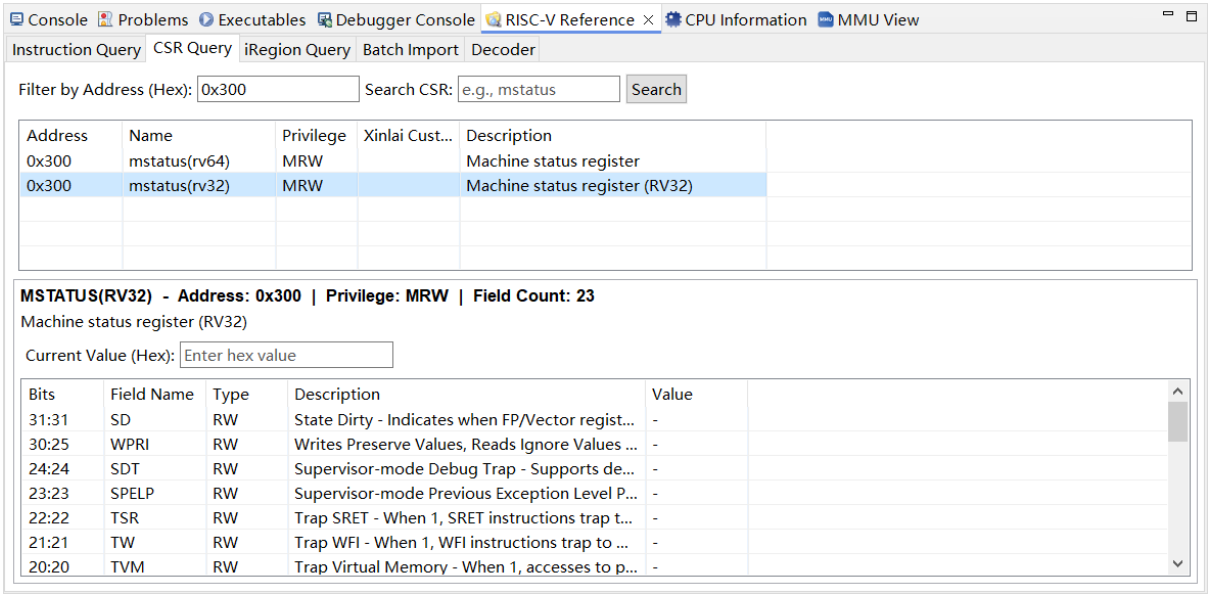
Query Instructions

1. Switch to the Instruction Query tab.
2. Enter a mnemonic (such as `addi`) or an opcode fragment in the search box.
3. In the result table, view the instruction architecture, extension, Mask/Match, format, and disassembly form.



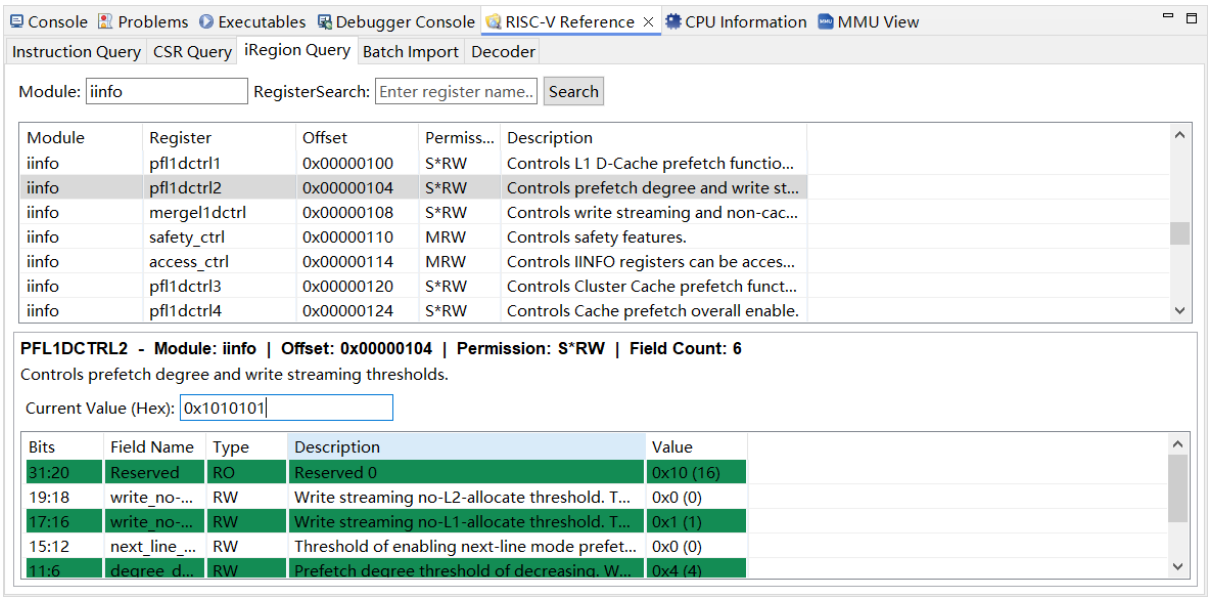
Query CSR and Interpret by Bit Field

1. Switch to the CSR Query tab.
2. Search by address (such as `0x300` for mstatus) or name.
3. Select the target CSR; its bit-field table is shown below.
4. Enter the actually-read value in the "current value" input box; the bit-field table decomposes according to that value, and non-zero fields are highlighted.



Decode Machine Code

1. Switch to the Decoder tab.
2. Enter hexadecimal machine code (you can also enter an assembly mnemonic for a reverse lookup).
3. The view outputs a decode table (13 columns), where you can see the meaning of each bit.
4. Batch is supported: multiple lines of input are decoded one by one.



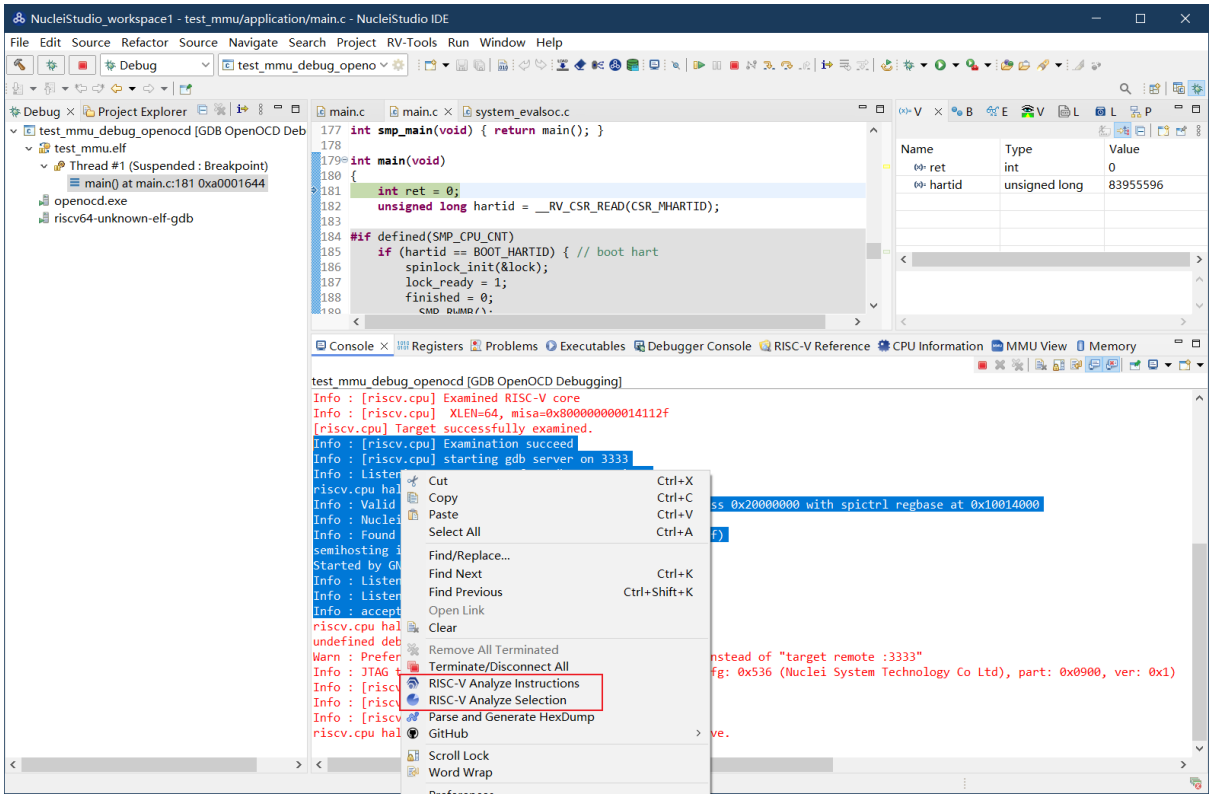
The decode table has 13 columns, with the following meanings:

Column	Meaning
Input	The raw content you entered (hexadecimal machine code or assembly text), echoed as-is
Assembly	Disassembly result, mnemonic + operands, e.g. `add a1, a5, a4`
Hex	Normalized hexadecimal machine code (0x prefix, left-padded with zeros to 32 bits)
Binary	32-bit binary expansion, bit-by-bit against the encoding
Mnemonic	Instruction mnemonic
Arch	Belonging architecture, e.g. RV32I / RV64I
Extension	Belonging extension, e.g. I / M / C / F ⋯
Mask	Encoding mask (works with Match to identify the instruction format)
Match	Encoding match value
Unratified	Whether unratified (draft extensions are marked true)
Format	Instruction format type, e.g. R / I / S / B / U / J
Disassembly	Disassembly form (full form, can be compared with Assembly)
Description	Textual description of the instruction

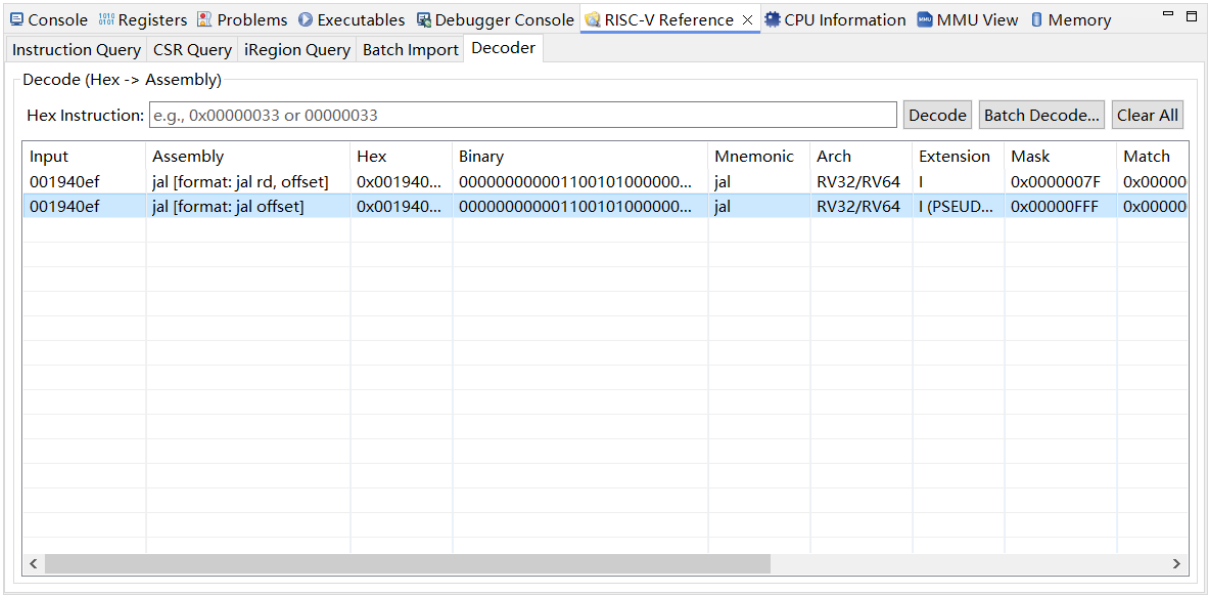
Batch Decode supports pasting a whole block of machine code / assembly, one per line; it also supports formats copied directly from LST / disassembly windows such as `address: machine code`, `address <symbol>: machine code`, `0x80000000: 00000033`.

Analyze from Editor or Console via Right-click

The view contributes right-click menu commands to the following places: Analyze Instructions / Analyze Selection, used after selecting text in the Console, text editor, Terminal, or LST view.

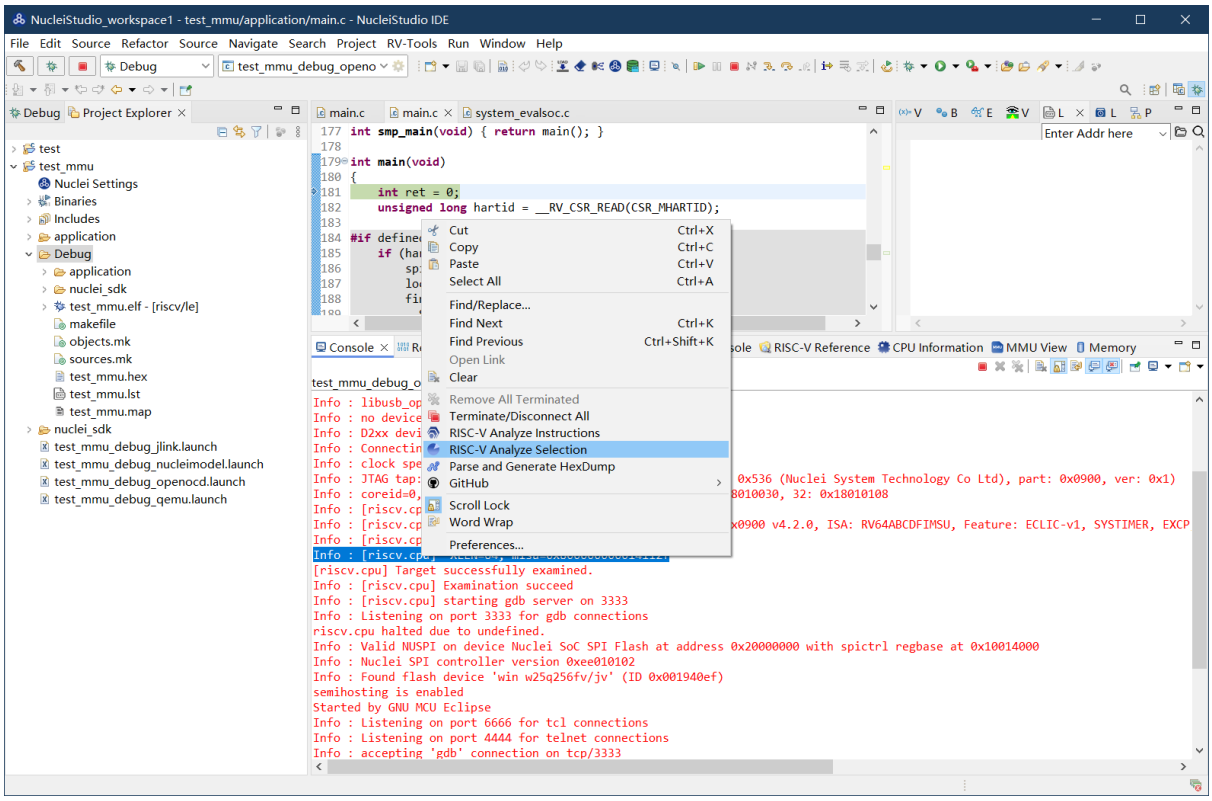


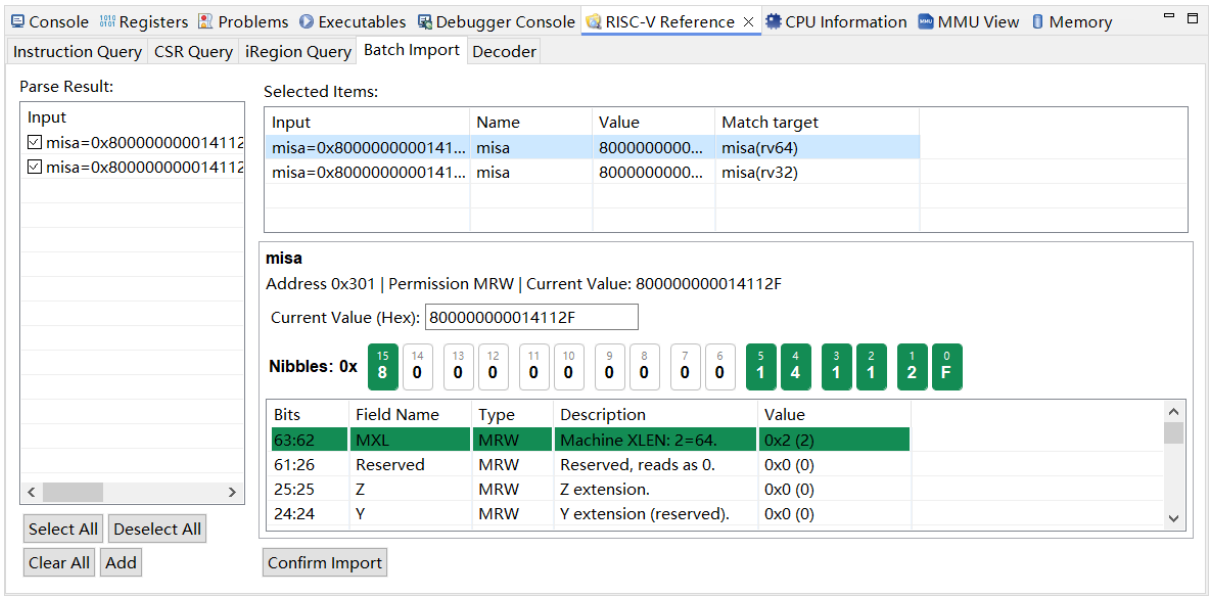
Analyze Instructions sends the result to the Decoder tab; Analyze Selection sends the result to the Batch Import tab. The benefit is: when you see a block of machine code in a log or disassembly window, you can select it directly and right-click to decode, without manually copying and pasting.



Decode Register Values from the Registers View

1. Open the Registers or Variables view during debugging.
2. Select a register (or multi-select several lines in `name=value` form).
3. Right-click and use `Decode Register`.
4. The register ' s current value is back-filled into Batch Import and interpreted by bit field.

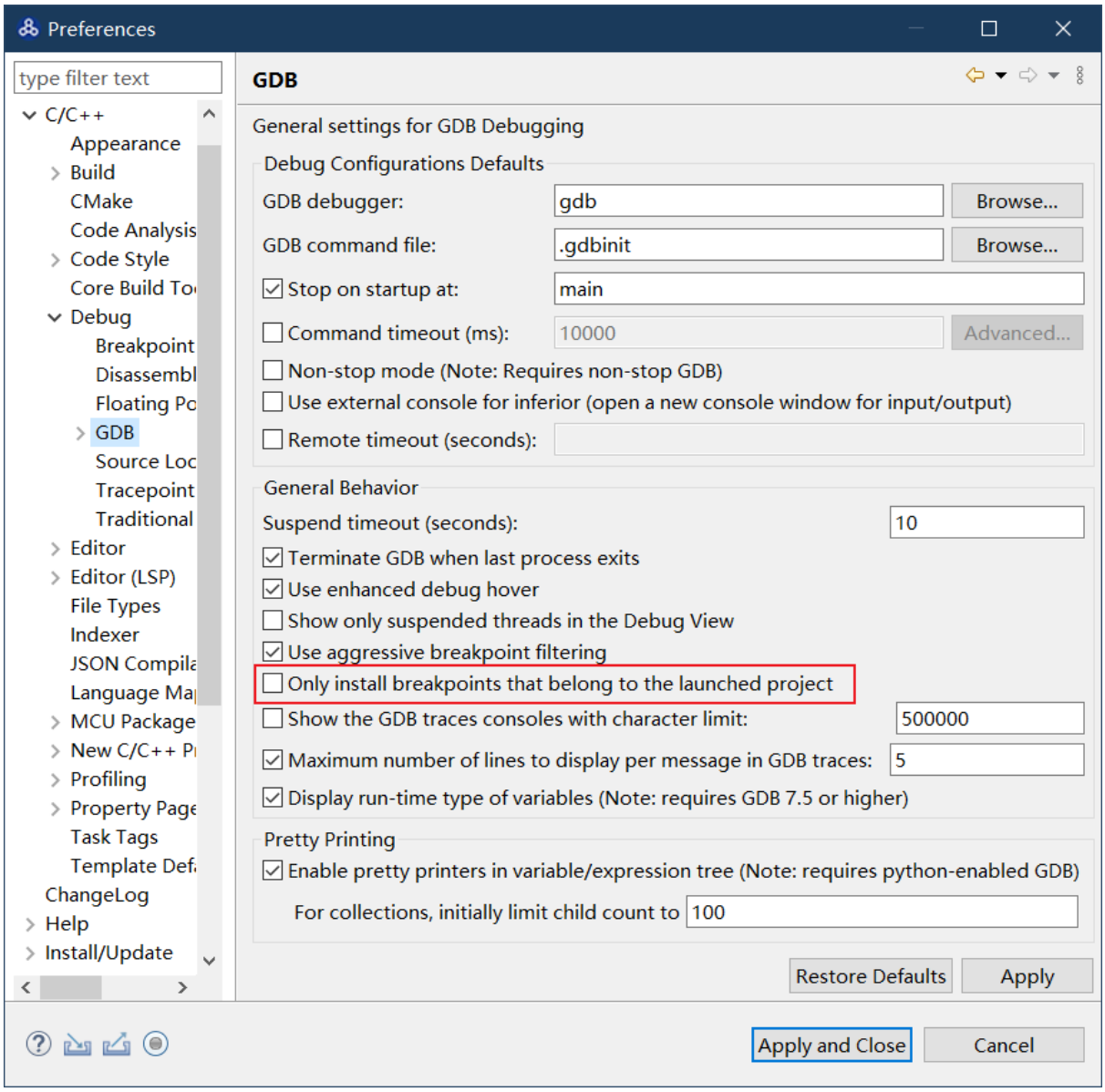




2.10.23 GDB Breakpoint Project Filter Option

In native Eclipse, when a project is debugged, all breakpoints are installed; if a breakpoint or its file does not exist, an error is reported. The GDB breakpoint project filter is a new GDB debug preference based on the underlying Eclipse CDT, which lets you install only the breakpoints of the currently debugged project to GDB in a multi-project workspace.

`Preferences > C/C++ > Debug > GDB`, under the General Behavior group, the option is named `Only install breakpoints that belong to the launched project`.



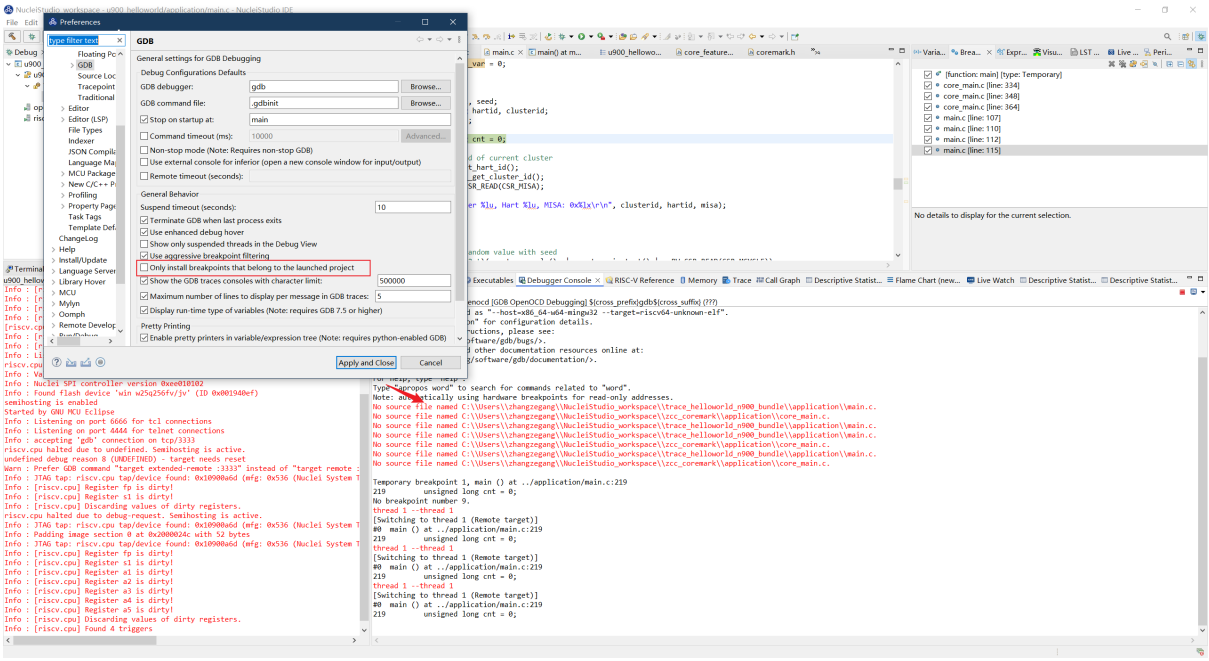
Behavior Notes

- This is a global workspace preference, off by default; when off, the behavior is the same as before (install breakpoints of all projects);
- After checking it, when installing breakpoints the debugger compares the breakpoint owning project with the launched project, and only installs to GDB the breakpoints belonging to the debugged project; breakpoints of other projects are skipped in this debug session;
- Breakpoints without a project set on files outside the workspace are still installed normally, to avoid accidentally dropping legitimate breakpoints.

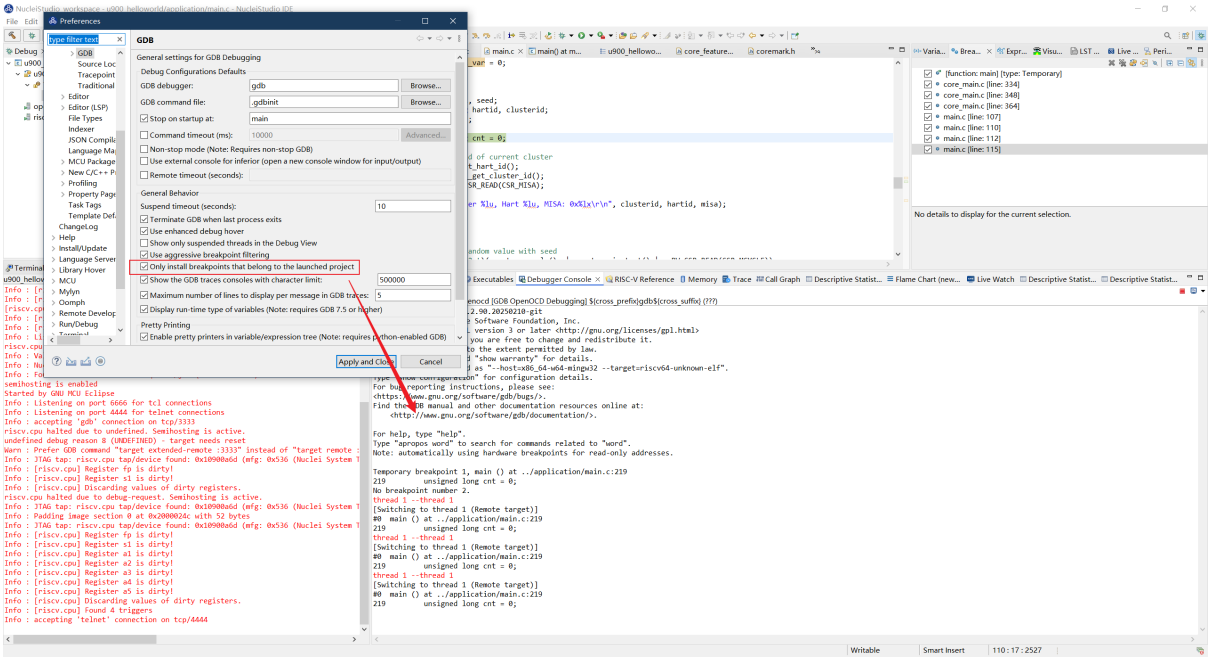
This option changes the breakpoints actually installed to GDB —from the very beginning only the current project

breakpoints are installed.

Before setting it, the IDE installs all breakpoints, so error messages appear.



After setting it, there are no error messages; especially when debugging multicore or multiple projects, it reduces error messages.



2.10.24 Memory View Address Exclusion

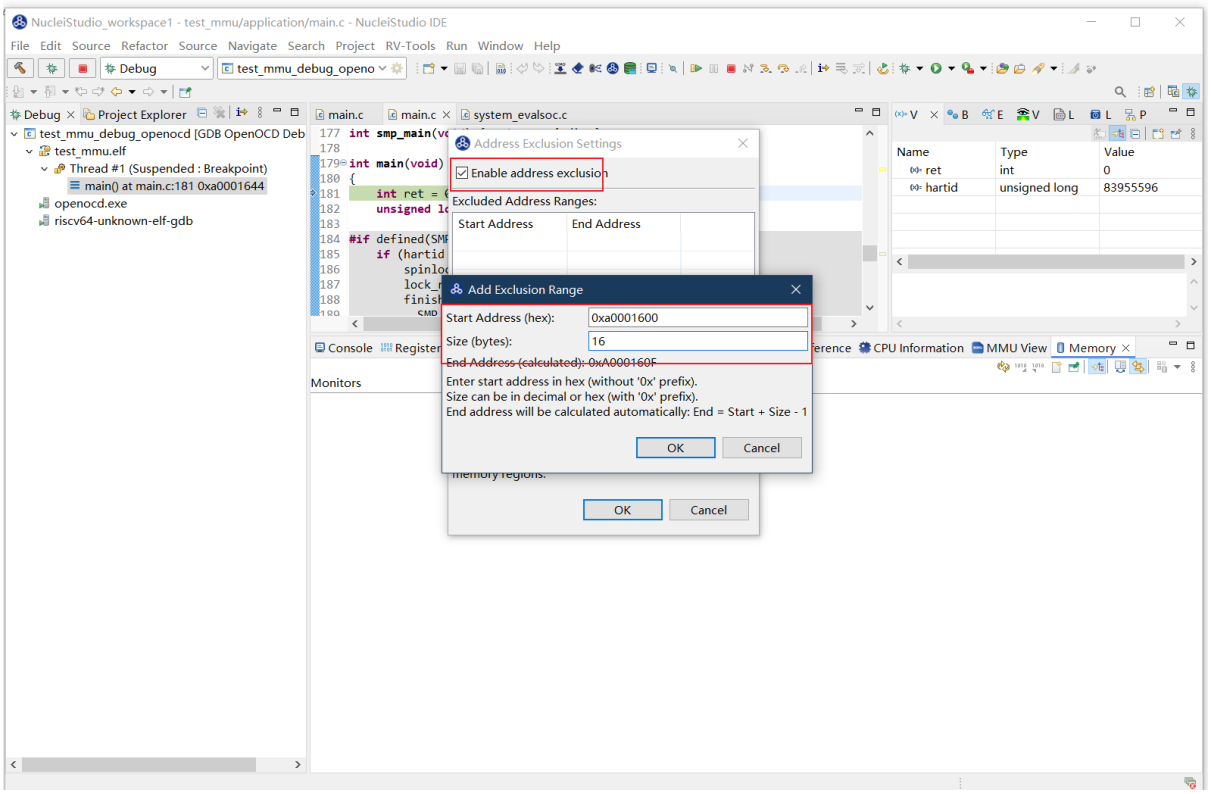
The Memory view adds an address exclusion setting, which lets you configure memory address ranges that the Memory view should not read, avoiding errors when reading inaccessible or protected memory regions; this feature also applies to the Memory Browser.

The Memory view is the standard Eclipse memory view. Nuclei Studio registers a Peripheral render type for it: it shows a memory block per peripheral register, colored by register/bit field, with read-only fields grayed out. Compared with looking at the raw hexadecimal dump directly, the Peripheral render breaks a block of memory into "which register, which bit field, what meaning", making peripheral state reading more intuitive.

The `Address Exclusion Settings...` item is in the Memory view menu (toolbar).

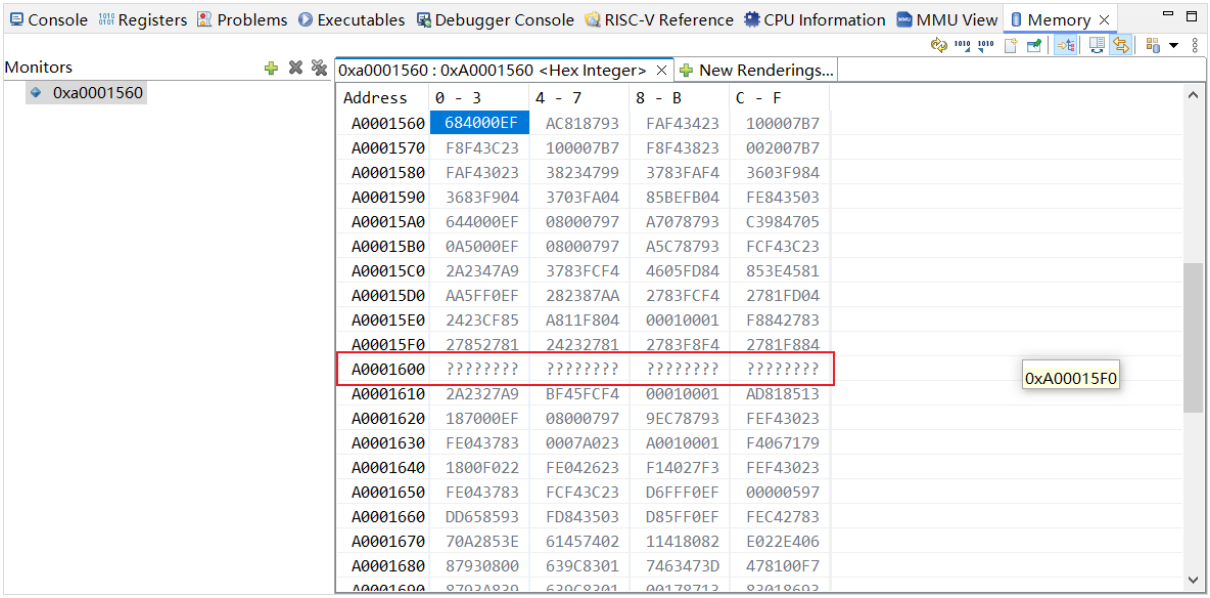


In the pop-up dialog, check `Enable address exclusion` to enable exclusion, and use `Add...` to add an excluded range (`Start Address` / `End Address`, hexadecimal without the `0x` prefix); `Remove` and `Clear All` are supported to manage the configured ranges.



Memory within the excluded address range is not read/installed, and the debugger skips these regions, thereby

avoiding errors or session anomalies caused by accessing illegal addresses. This exclusion setting also applies to the Memory Browser, avoiding errors when reading inaccessible regions across projects or sessions.

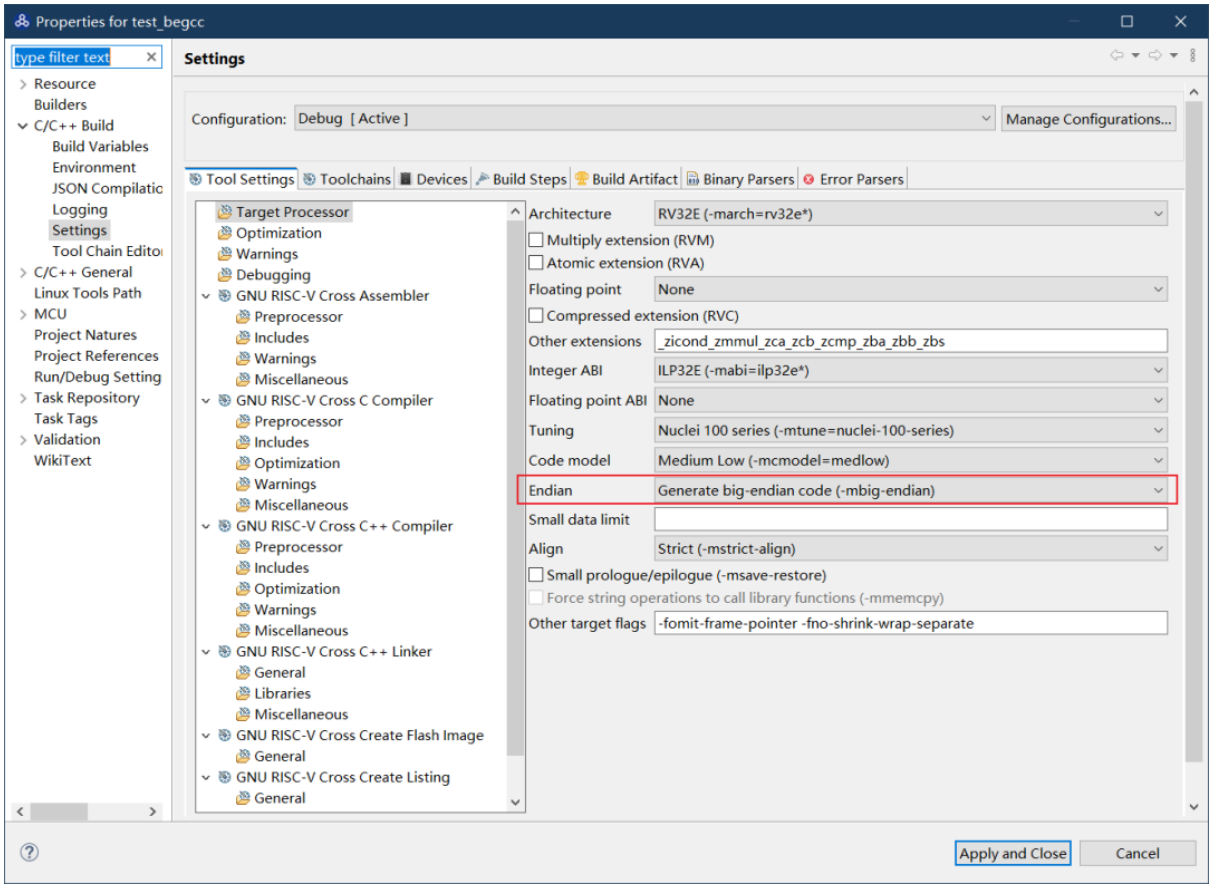


2.10.25 Big Endian Compiler Support

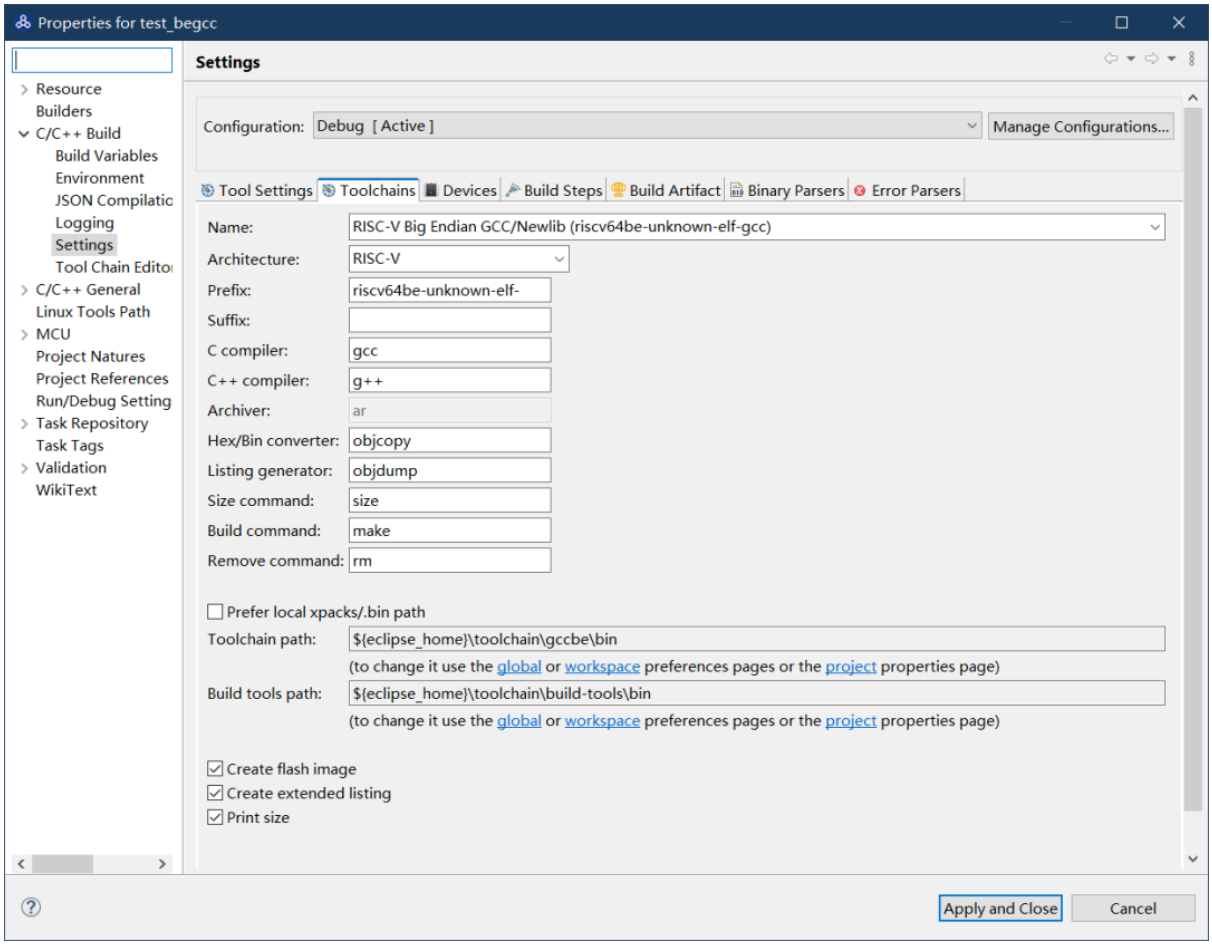
Note: This feature is still in the test phase.

Nuclei Studio 2026.10 adds support for the RISC-V Big Endian toolchain, so that projects can be compiled into big-endian code for big-endian RISC-V targets. To use this feature:

1. Download the Big Endian toolchain (bengcc) and place it in the `<NucleiStudio>\toolchain\gccbe` directory.
2. Modify the project properties: right-click the project and open `Properties -> C/C++ Build -> Settings`, then on the `Tool Settings -> Target Processor` page, set `Endian` to `Generate big-endian code (-mbig-endian)`.



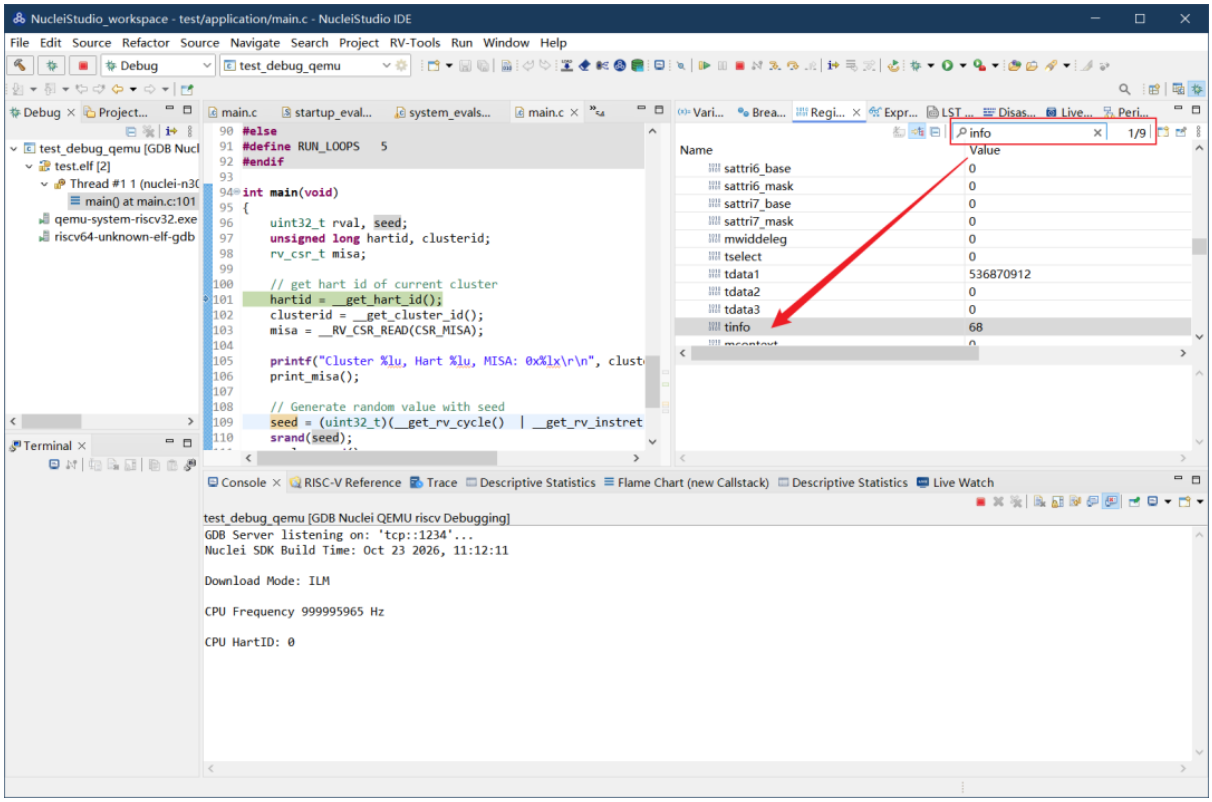
3. Modify the toolchain information of the project: on the `Toolchains` page, set `Name` to `RISC-V Big Endian GCC/Newlib (riscv64be-unknown-elf-gcc)`, set `Prefix` to `riscv64be-unknown-elf-`, and set `Toolchain path` to `${eclipse_home}\toolchain\gccbe\bin`.



Then build the project; the generated code and ELF files are big-endian.

2.10.26 Register View Filter

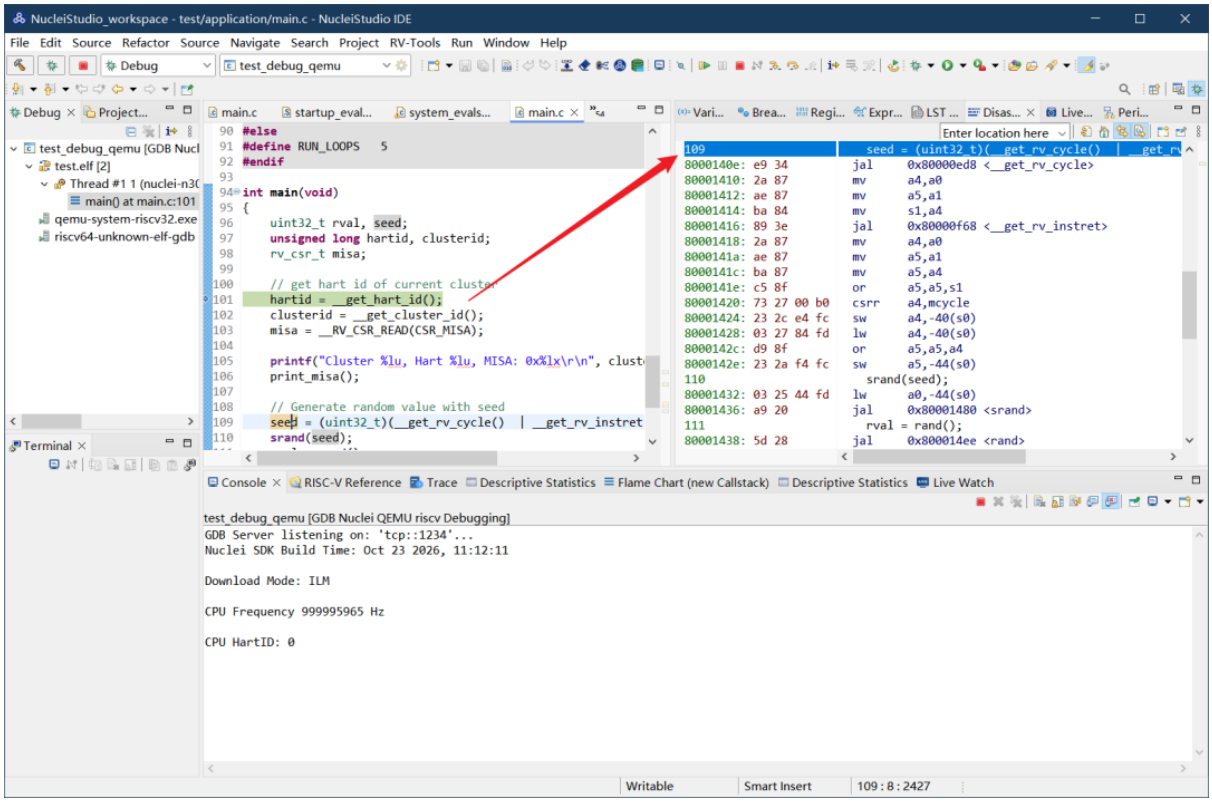
A filter box has been added to the Registers view (and the Variables view). During debugging, enter a keyword in the filter box to narrow the register list down to the matching registers, so you can quickly locate the registers you want to view without scrolling through the full list. A counter is displayed next to the filter box while filtering (for example, `1/9`), indicating how many registers match and which one is currently selected.



2.10.27 Disassembly View C Code Linkage

The Disassembly view is now linked with the C/C++ editor:

- Clicking a line in the C source file makes the Disassembly view jump to the corresponding instructions and highlight them;
- When the editor navigates to another source file (for example, by following a function call), the Disassembly view follows and displays the disassembly of the corresponding content.



2.11 Nuclei Studio Update and Upgrade

2.11.1 Online Update

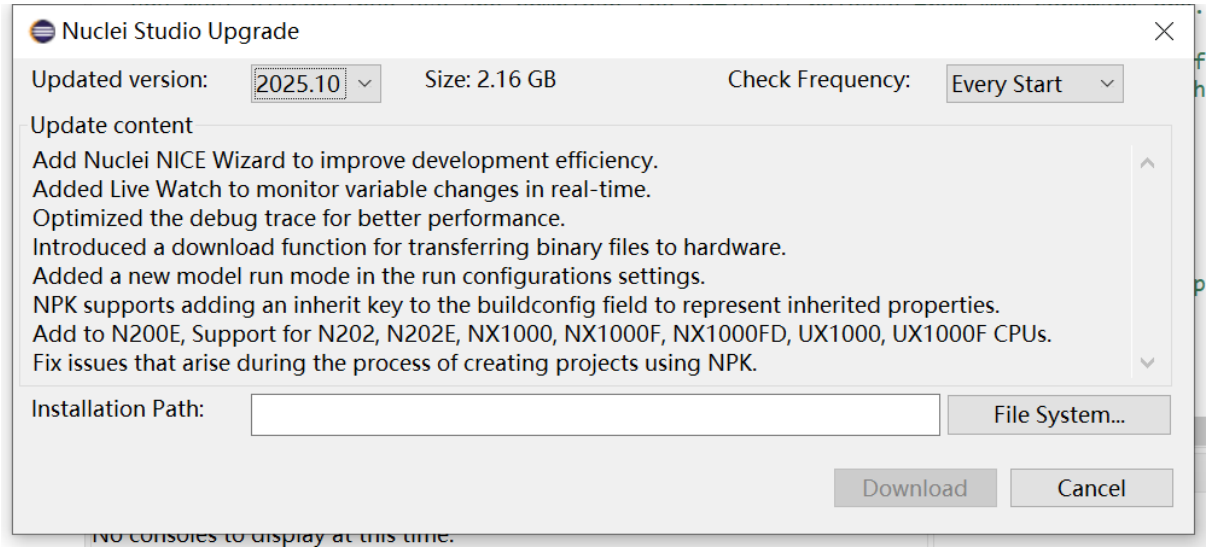
Note: Nuclei Studio 2025.10 and later can use the following methods to update and upgrade the IDE online.

When a new version of Nuclei Studio is released and the IDE detects that a newer version is available for the IDE, plugins, or accompanying toolchains (such as compilers and debuggers) currently in use, it automatically displays a version upgrade prompt at startup.

Installing Nuclei Studio IDE

In versions before Nuclei Studio 2025.10, when a new version of Nuclei Studio was released, users had to manually download the latest installation package and perform the upgrade themselves. Starting with Nuclei Studio 2025.10, a brand-new online upgrade mechanism is available: the built-in upgrade tool downloads the new version online and automatically completes extraction and installation, upgrading the IDE with one click. This significantly simplifies the upgrade process and improves the user experience.

When Nuclei Studio starts, it checks whether a new version is available for download and upgrade. If so, an upgrade prompt dialog pops up in which you can select the version to upgrade to and the installation path of the new version, and set the frequency of upgrade prompts.



- Updated version: the Nuclei Studio versions available for upgrade.
- Check Frequency: the frequency of checking for new versions at startup; the default is Every Start.
- Installation Path: the installation path of the new IDE version.

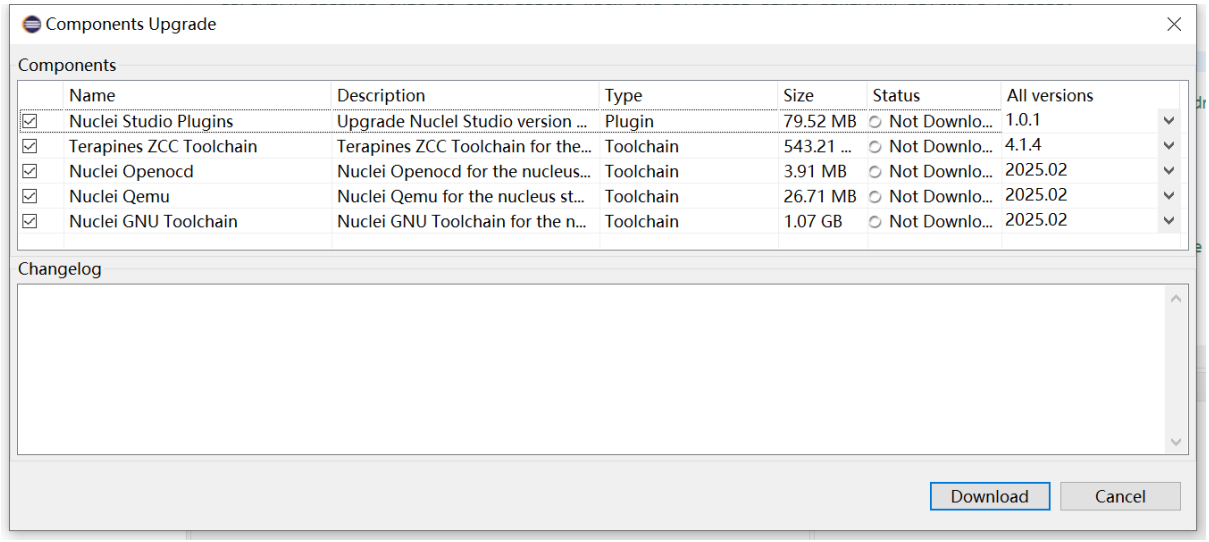
In addition to the automatic pop-up at startup when Nuclei Studio detects a new version, you can also select `Help -> Check for IDE updates` in the menu bar to check for IDE updates.

Installing Toolchains and Plugins

Toolchains and plugins are important components of Nuclei Studio. Toolchains include core development components such as the GCC compiler, the QEMU simulator, and Nuclei Model; plugins extend Nuclei Studio functionality and support the continuous integration of new features and debugging capabilities. Keeping them up to date provides a better user experience.

In versions before Nuclei Studio 2025.10, toolchains and plugins could not be upgraded online either. Starting with Nuclei Studio 2025.10, a brand-new online upgrade mechanism is available: the built-in upgrade tool downloads new versions online and automatically completes the upgrade and installation.

When Nuclei Studio starts, it checks whether new versions of toolchains or plugins are available. If so, an upgrade prompt dialog pops up. Select the toolchain or plugin version to upgrade to, then click Upgrade to update the toolchains and plugins automatically.



- Name: the name of the toolchain or plugin to be upgraded.

- Description: the description of the toolchain or plugin to be upgraded.
- Type: only Plugin and Toolchain, corresponding to the plugin and toolchain types respectively.
- When an item is selected, Changelog displays the changelog of the corresponding toolchain or plugin to be upgraded.

In addition to the automatic pop-up at startup when Nuclei Studio detects new versions of toolchains and plugins, you can also select `Help -> Check for component updates` in the menu bar to check for component updates.

2.11.2 Manual Update

Note: Versions before Nuclei Studio 2025.10 can use the following methods to update and upgrade the IDE manually.

When an online update cannot be completed, you can manually download the latest GNU toolchain and other related tools and upgrade the IDE plugins.

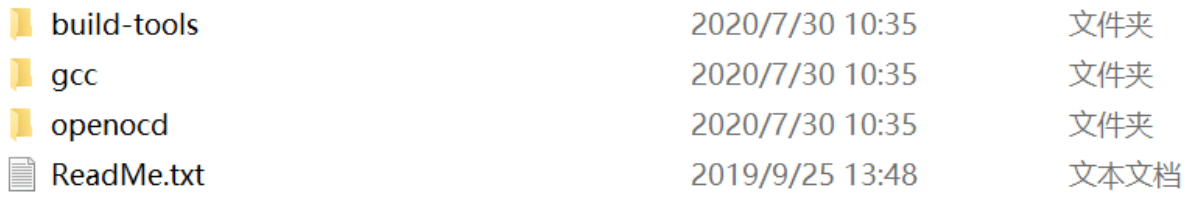
Installing the Toolchain

Nuclei Studio integrates the toolchain inside the IDE. The toolchain is stored in the Nuclei Studio_IDE_XXX folder at the path `Nuclei Studio_IDE_XXX\\NucleiStudio\\toolchain`. The IDE uses the toolchain at this path by default, so do not move the `toolchain` folder. After you create a project with the IDE, no toolchain-related configuration is required.

Because the toolchain is upgraded faster than the IDE, you may need to upgrade the toolchain manually later. To upgrade the toolchain:

- Delete the contents of the GCC or OpenOCD folder to be updated, then download the latest version of GCC or OpenOCD from the Nuclei System Technology official website.
- Copy the contents of GCC or OpenOCD into the corresponding folder to complete the IDE toolchain update. The version numbers of the toolchain bundled with the IDE are recorded in ReadMe.txt.

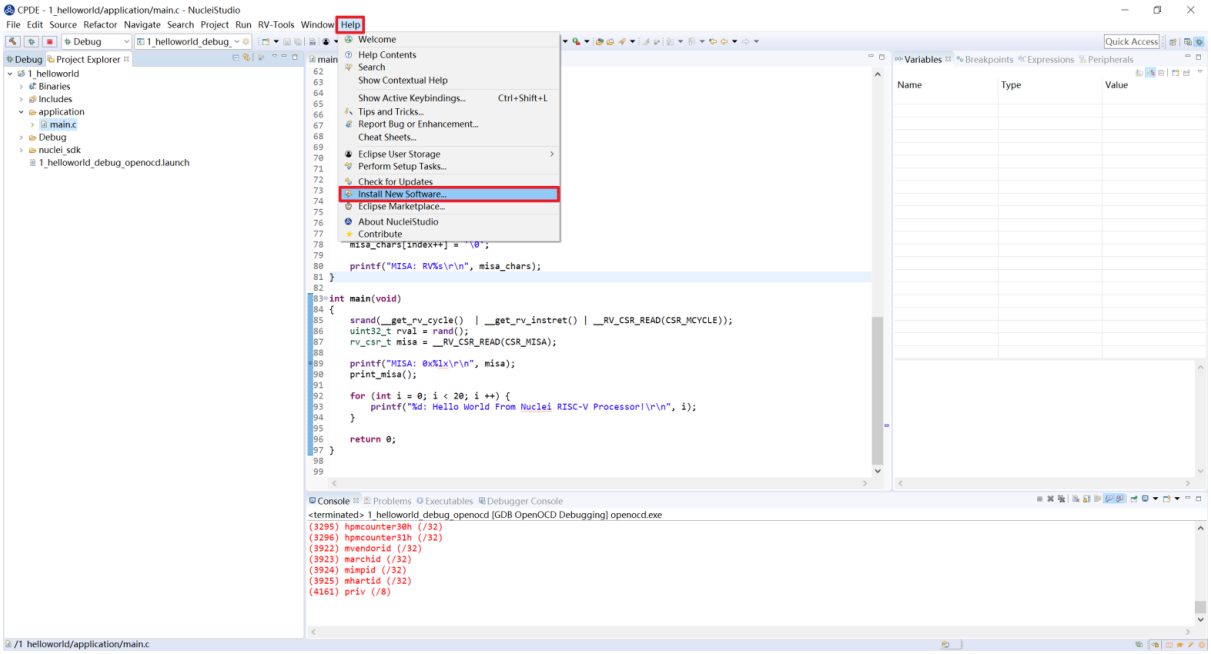
Note: Note: Keep the name of the directory containing gcc unchanged. After replacement, make sure the bin folder is a direct subdirectory of the folder shown in the figure, with no other folders in between.



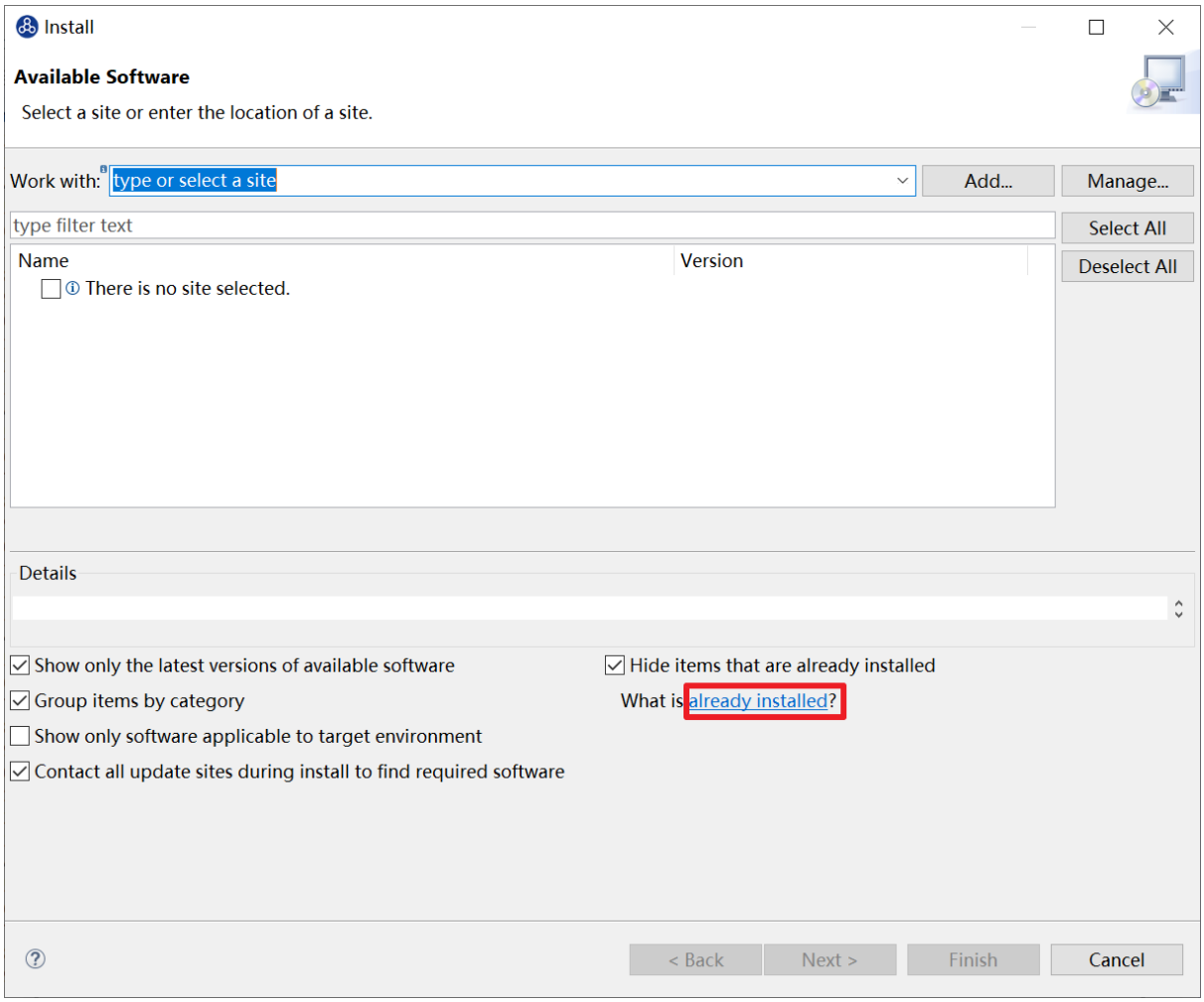
Upgrading IDE Plugins

Nuclei Studio supports online upgrades within the IDE, as follows.

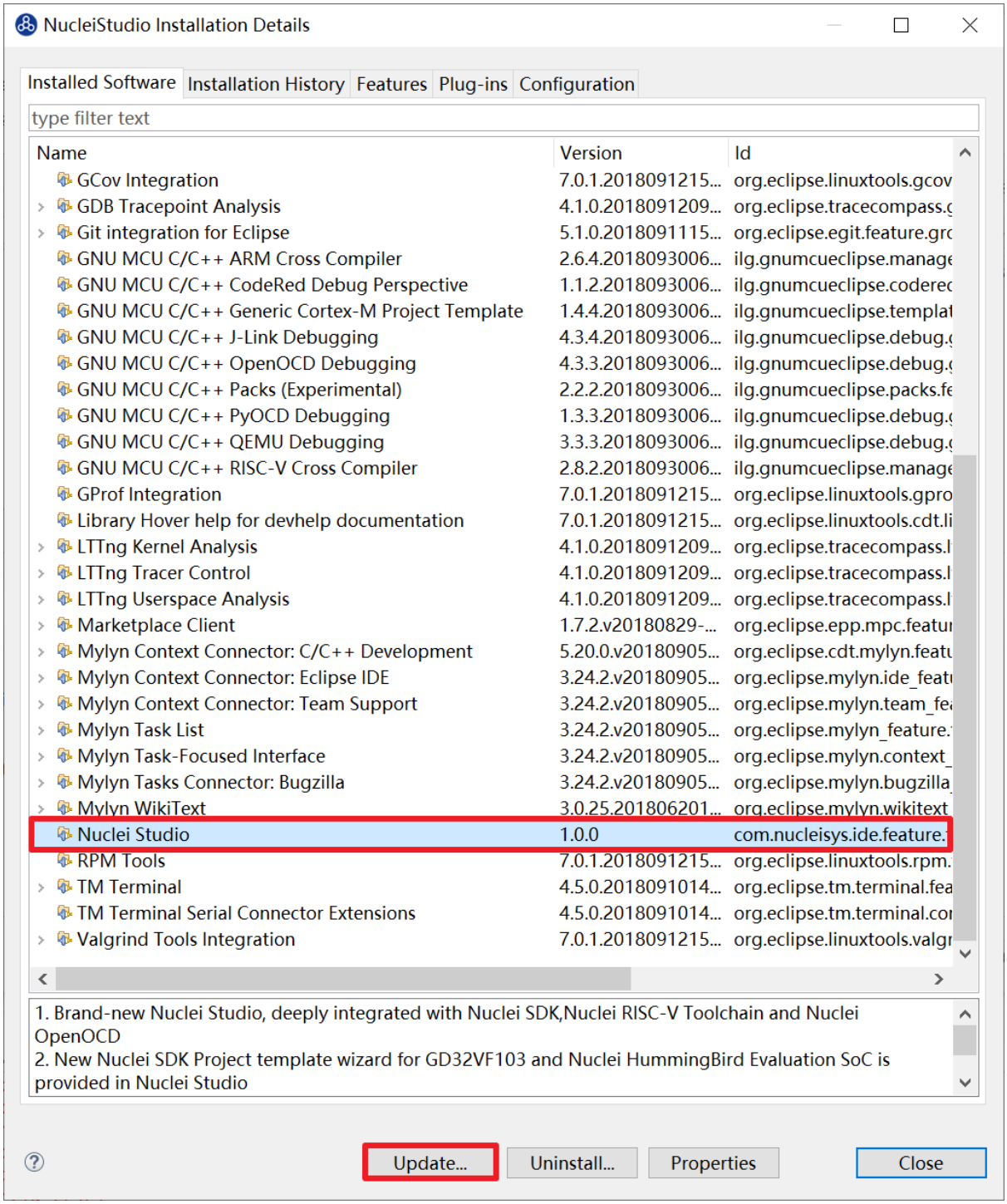
Open the software installation dialog by selecting `Help -> Install New Software` in the menu bar.



Click `already installed` to open the installed software page.



Select `Nuclei Studio` and click `update` to automatically complete the online update.



2.12 Nuclei Studio FAQ

2.12.1 Nuclei Studio Starts Slowly

Nuclei Studio is built on Eclipse. While it inherits Eclipse ' s extensibility, it also suffers from slow startup: on the first launch, the IDE must verify the environment, generate commonly used cache files, and create the workspace. Startup was optimized in Nuclei Studio 2021.09; with adequate hardware, the first launch of Nuclei Studio completes within one minute.

Recommended environment for Nuclei Studio:

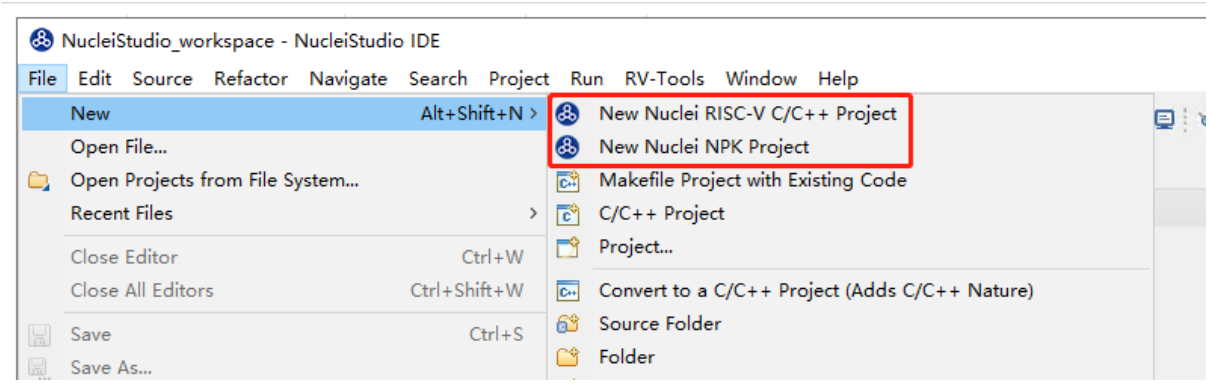
- Windows 10 operating system
- More than 4 GB of memory (2 GB allocated to Nuclei Studio)

2.12.2 Nuclei Studio Builds Programs Slowly

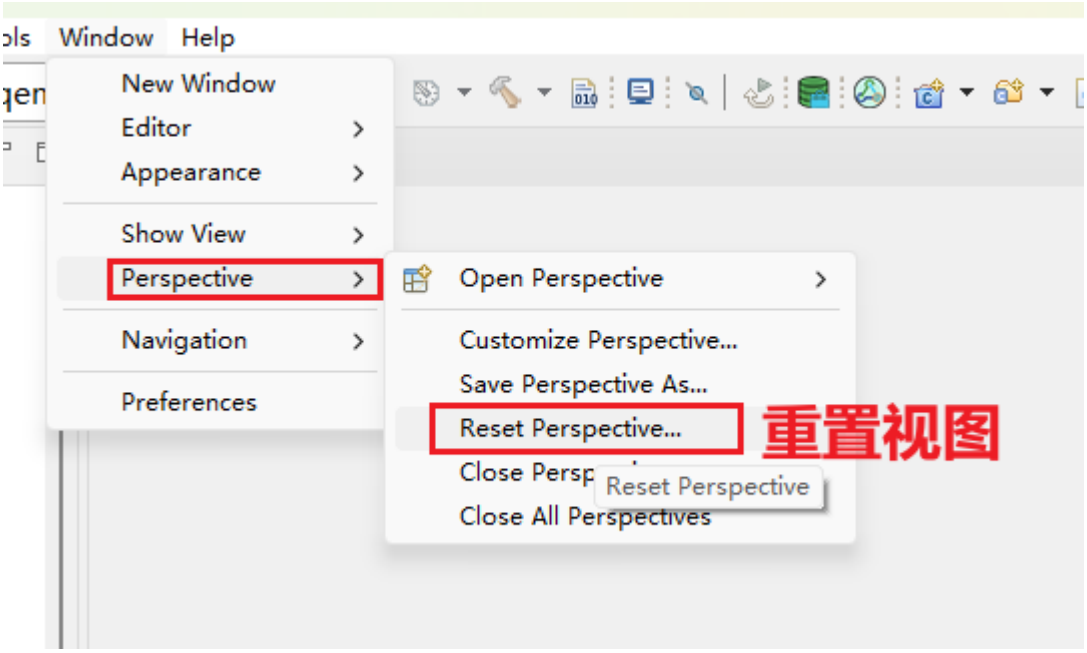
During testing, we found that Nuclei Studio builds projects very slowly when 360 antivirus software is installed; after exiting the 360 antivirus software, builds run at normal speed. To ensure Nuclei Studio works normally, exit antivirus software such as 360 while using it.

2.12.3 Cannot Find the New Nuclei RISC-V C/C++ Project Menu

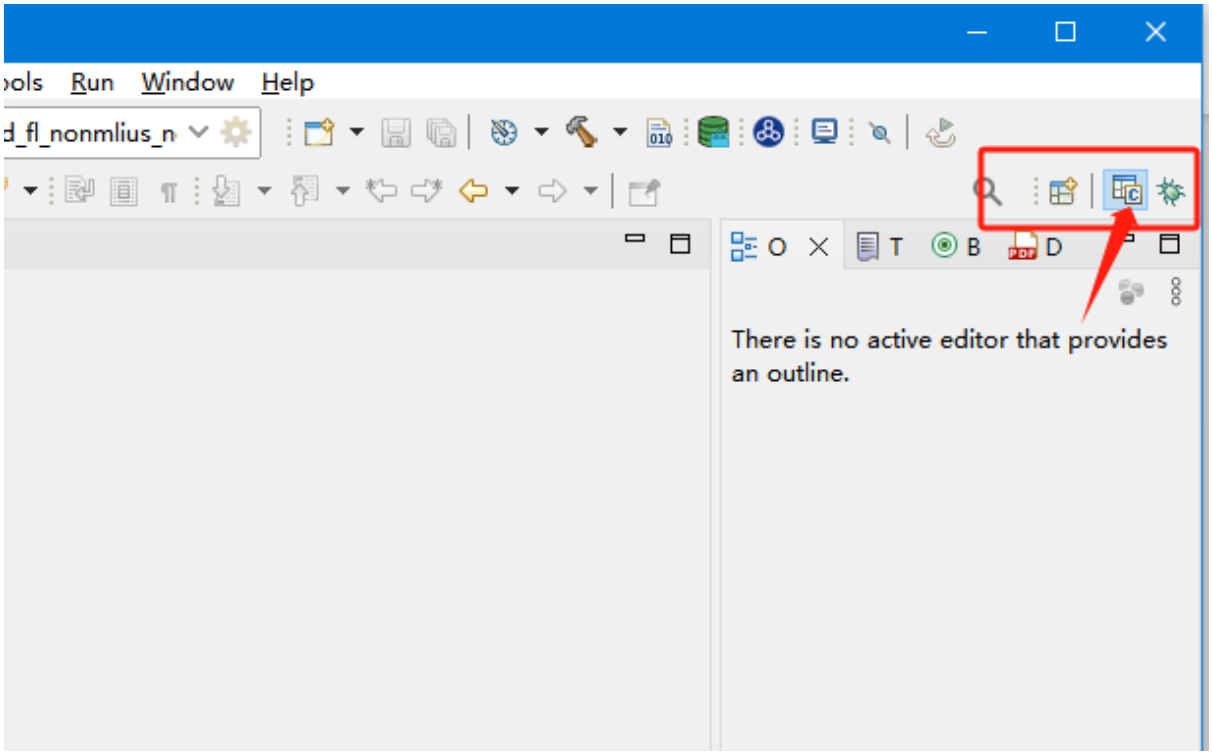
The `New Nuclei RISC-V C/C++ Project` menu is the quick entry point for creating projects, but it may sometimes not appear.



Reset the perspective via the menu `Window -> Perspective -> Reset Perspective`, and `New Nuclei RISC-V C/C++ Project` will reappear.

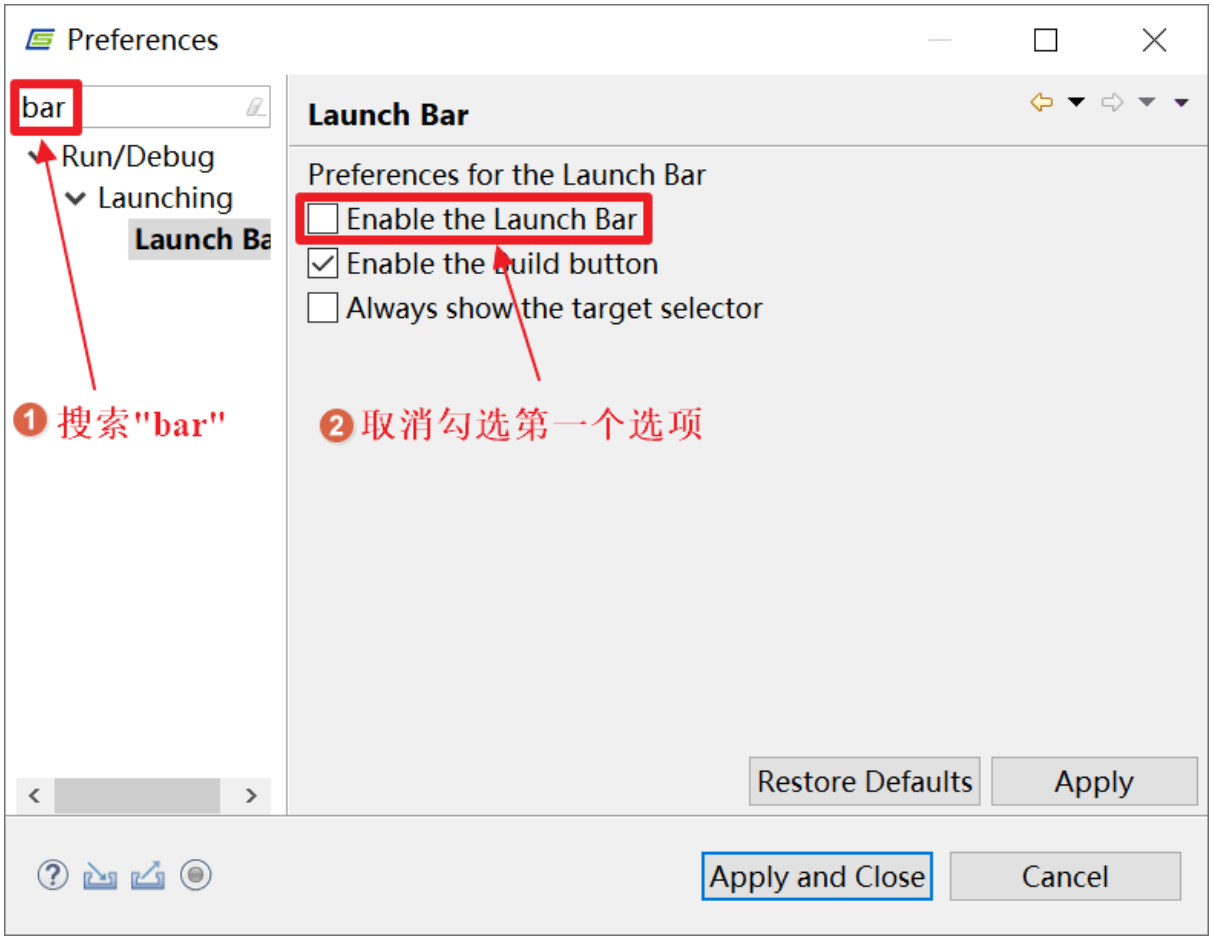


Alternatively, switch the current perspective to the C/C++ perspective.



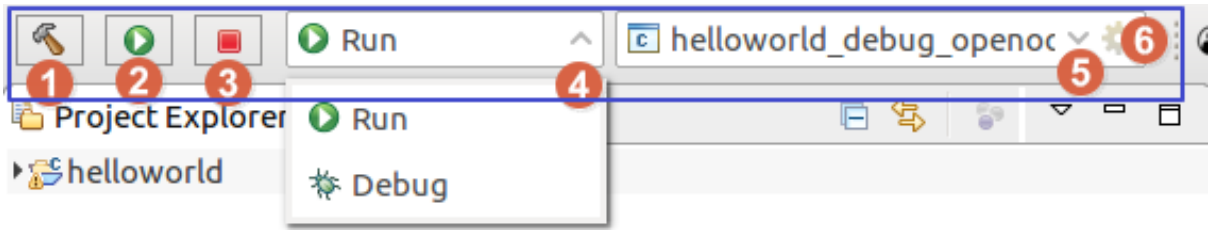
2.12.4 Opening/Closing the Launch Bar

Nuclei Studio 2022.12 and later enables the Launch Bar by default. To close the Launch Bar, open `Window -> Preferences`, search for `bar`, and uncheck the first option. Check the option again to re-enable the Launch Bar.



2.12.5 Using the Launch Bar

The Nuclei Studio Launch Bar (the part marked by the blue box in the figure below) builds, debugs, and downloads the project associated with the debug configuration selected in the drop-down box. Note that the icons are bound to that selected debug configuration. The function of each part of the Launch Bar is described below:



Icon 1: Builds the project associated with the debug configuration selected in drop-down box 5 —not the project selected in Project Explorer.

Icon 2: Runs/debugs the project associated with the selected debug configuration. This icon changes according to the content of drop-down box 4.

Icon 3: Exits download/debug mode.

Drop-down box 4: Switches between download/debug modes; switching changes what icon 2 displays.

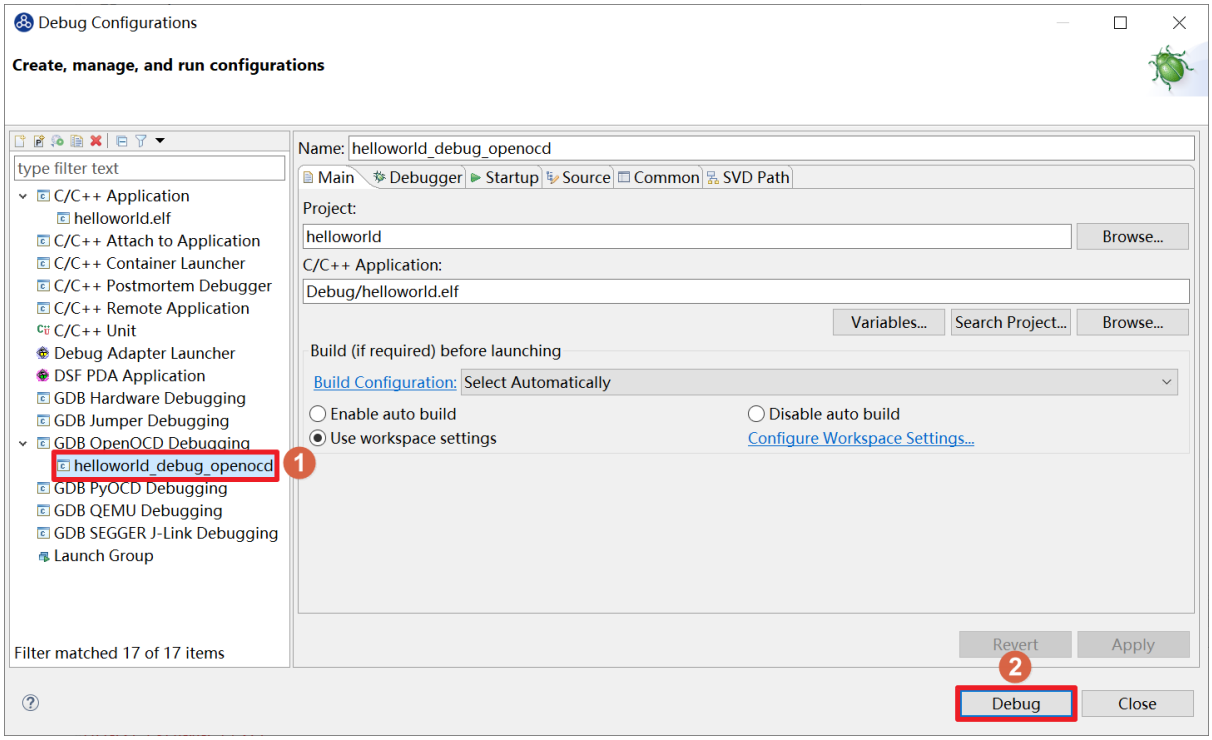
Drop-down box 5: Selects the debug configuration file. This selection determines which projects icons 1, 2, and 3 act on.

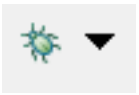
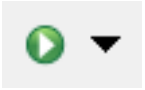
Gear icon 6: Opens the currently selected debug configuration page, where you can modify and save it directly.

2.12.6 Running/Debugging Without the Launch Bar

For a newly created project, if you do not use the Launch Bar, right-click the project and select `Debug As -> Debug Configurations`. In the dialog, select the configuration corresponding to the project —for the helloworld project, select `helloworld_debug_openocd` —and then click `Debug` to start debugging.

Similarly, for the first run, right-click the project and select `Run As -> Run Configurations`. In the dialog, select the configuration corresponding to the project, and then click `Run` to start running.



If the current project has already been debugged or run as described above, use the drop-down option of  in the toolbar to quickly select the corresponding project configuration for debugging, or the drop-down option of  to quickly select the corresponding configuration for running.

2.12.7 Viewing Registers in the Debug Perspective

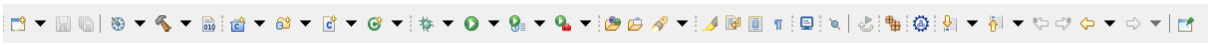
After entering Debug mode, some views may not be visible right away; the Registers view is used as an example here. Select `Window -> Show View -> Registers` in the menu bar to open the Registers view, and use `Ctrl-F` in the view to search for registers. Similarly, to view memory, select `Window -> Show View -> Memory`; to open the debug console, select `Window -> Show View -> Debugger Console`. Other views are not covered here.

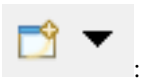
2.12.8 Terminating Processes in the Debug Perspective

To exit debugging in the Debug perspective, click . To prevent errors caused by multiple open processes, right-click the process in the Debug view and select `Terminate / Disconnect All` to close all debug processes.

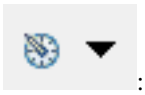
2.12.9 Functions of the Buttons in the Nuclei Studio Toolbar

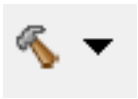
The common Nuclei Studio toolbar icons are shown below. The function of each button is described below.



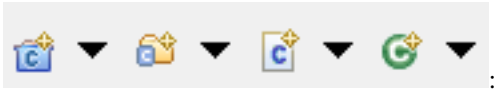
New button : creates various files and projects.

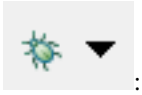
Save the current file , or save all unsaved files .

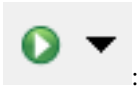
Switch build configuration : only Debug and Release are available by default; you can add more configurations yourself.

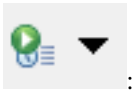
Build project : builds the project selected in Project Explorer using the drop-down options.

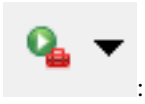
Build all projects : builds all projects in Project Explorer at once.

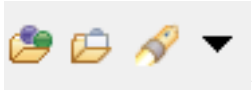
New functionality : creates projects, directories, files, and classes.

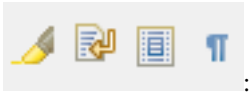
Debug : selects a debug configuration from the drop-down options to debug the project.

Run : selects a configuration from the drop-down options to run the project.

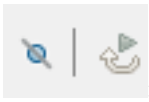
Profile : the Profile feature is not currently supported and is not covered here.

User-defined tools : not covered here; you can explore them on your own.

Search tools : searches tasks, text, and more.

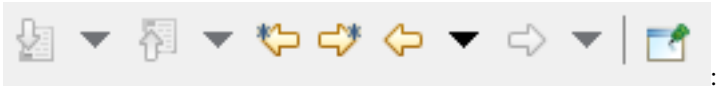
Text reading tools : expands or collapses text, and so on.

Console tools : opens various consoles or serial ports.

Debug buttons  : the first skips all breakpoints; the second performs the Restart function.

View CMSIS  : a built-in Eclipse feature; not currently supported.

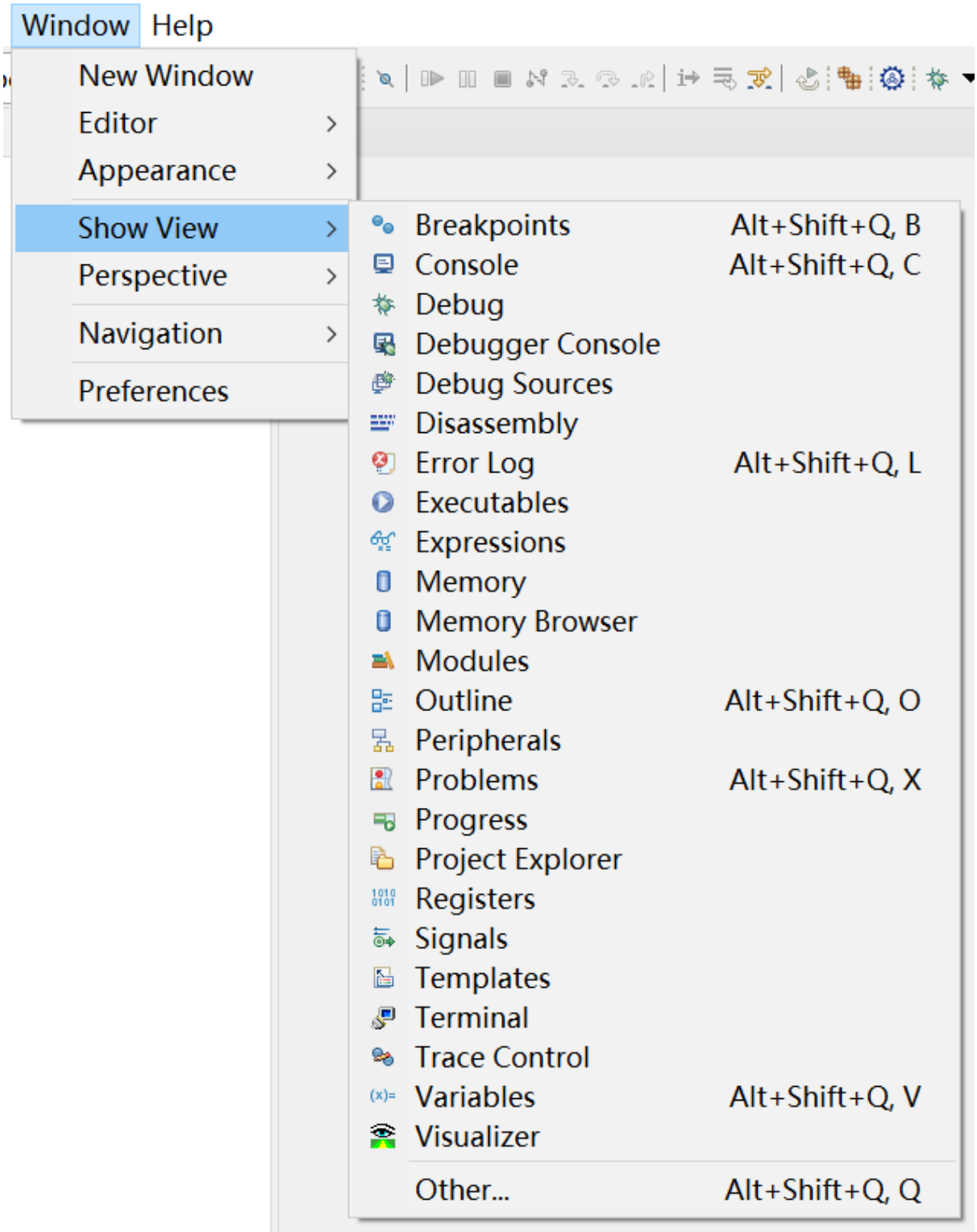
Nuclei SDK project settings tool  : see Section 6.6.1.

Text reading tools  : sets quick navigation; the detailed functions are not covered here.

2.12.10 Showing Other Windows

The  button in the upper-right corner of the IDE switches between different display modes; the most commonly used are the C/C++ perspective and the Debug perspective. When entering Debug mode from another perspective, a dialog prompts you; click the `Switch` option to switch to the Debug perspective. Select `Window -> Perspective -> Open Perspective` in the menu bar to switch perspectives quickly.

The views available under `Window -> Show View` differ between perspectives. In the Debug perspective, you can show views such as Registers, Memory, and the Debugger Console.

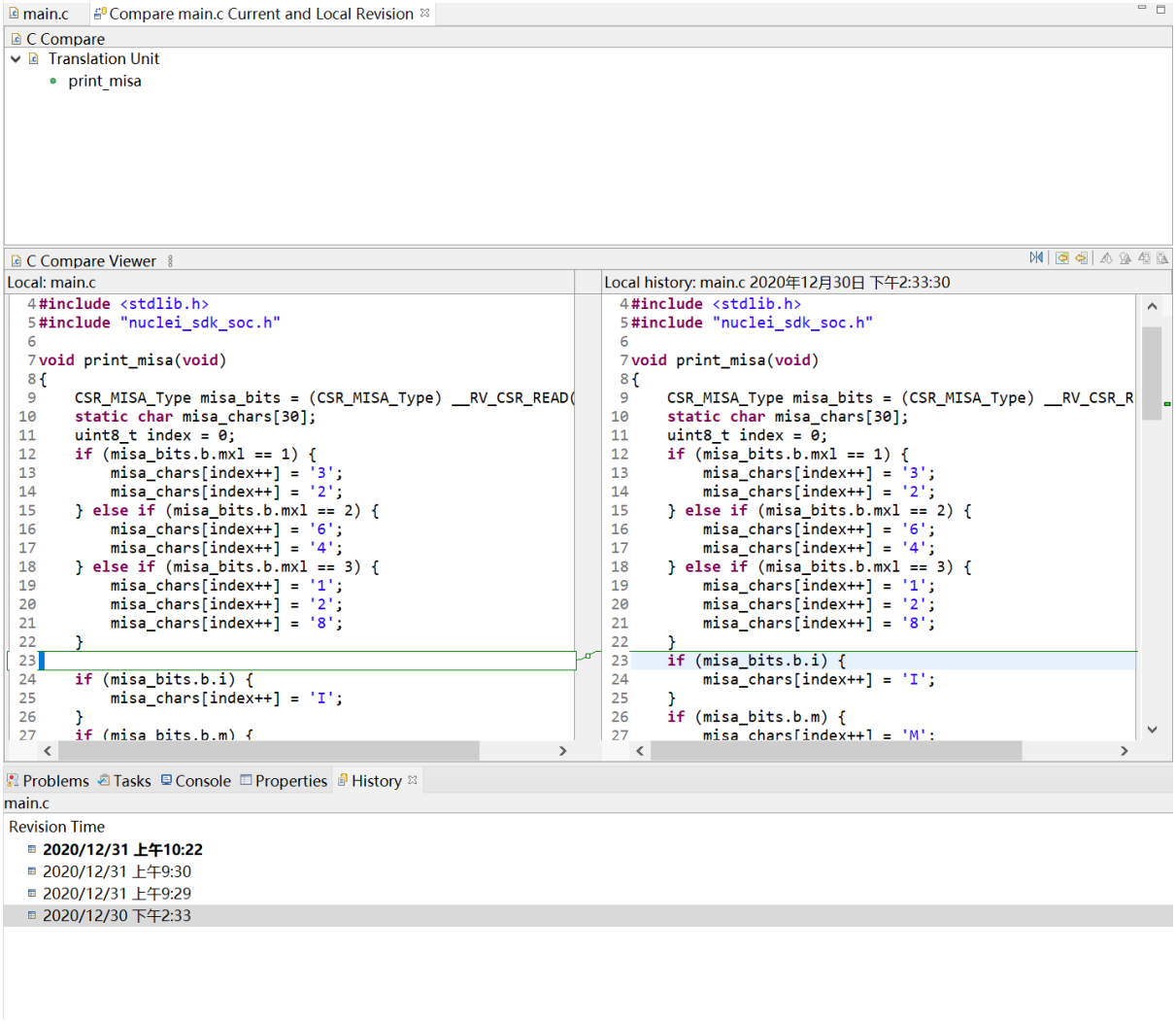

Window
Help
New Window
Editor
Appearance
Show View
Perspective
Navigation
Preferences
Breakpoints Alt+Shift+Q, B
Console Alt+Shift+Q, C
Debug
Debugger Console
Debug Sources
Disassembly
Error Log Alt+Shift+Q, L
Executables
Expressions
Memory
Memory Browser
Modules
Outline Alt+Shift+Q, O
Peripherals
Problems 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 Restoring the Default Window Layout

Select `Window -> Perspective -> Reset Perspective` in the menu bar and choose `Reset Perspective` in the dialog to restore the default window layout.

2.12.12 Comparing with File History

Right-click the file whose history you want to view and select `Compare With -> Local History` to open the history records. In the History view, double-click the file version you want to compare to view the file change history.



2.12.13 Possible Errors When Creating a Project

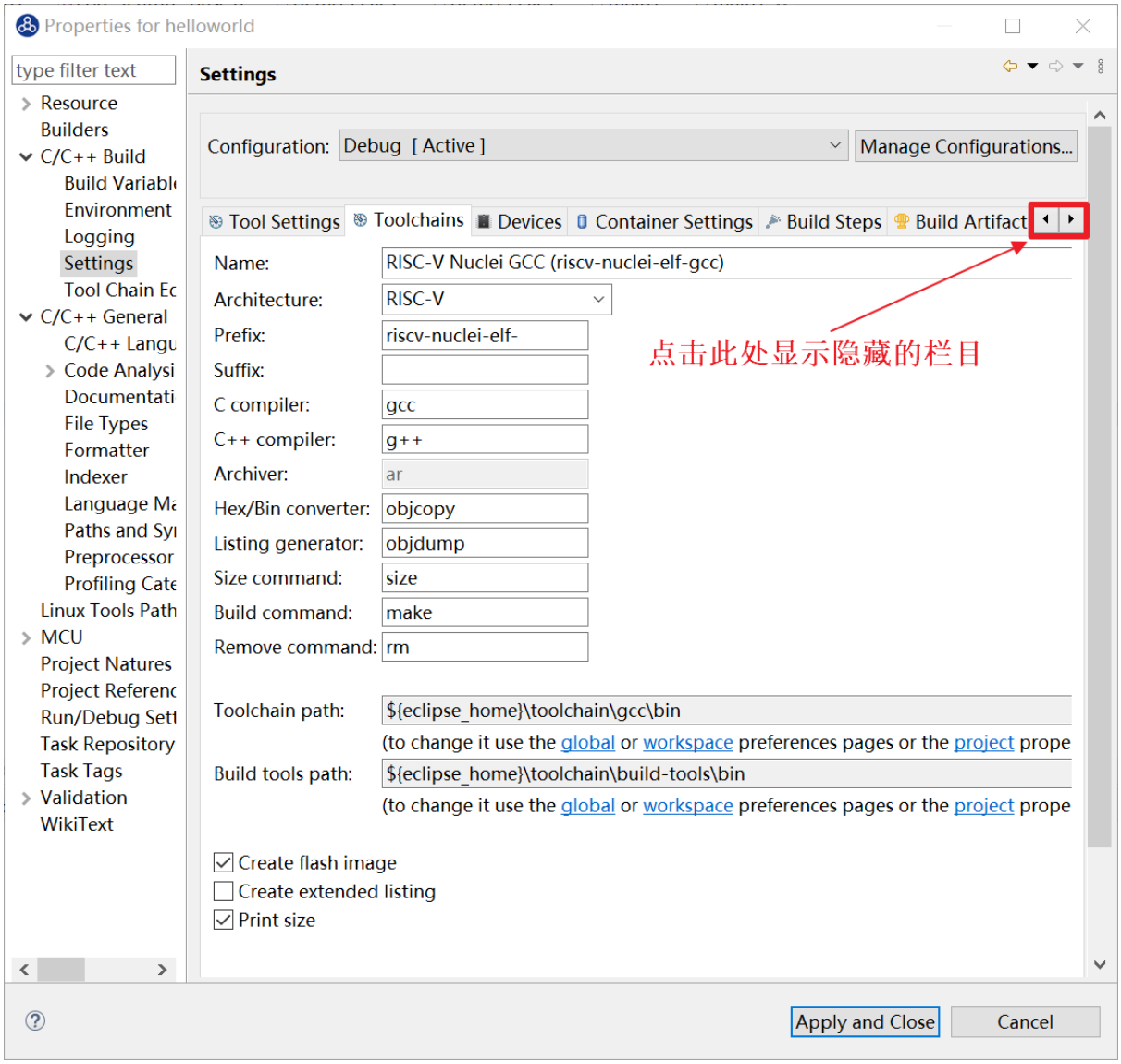
When a new project is created, the IDE indexes the entire project, and the time required varies with host performance. On a low-performance host, transient errors may appear during indexing; these errors disappear once indexing finishes. In this case, rely on the errors reported during the build.

2.12.14 Cached Paths When Adding an Include Path

When adding a path on the include page, cached path content may appear; in this case, click Workspace or File System and select the path to overwrite the cached content.

2.12.15 Cannot Find a Settings Page Section

Depending on the dialog size, some settings sections may be hidden; click the left/right arrow icons in the red box to show the hidden sections.



2.12.16 Slow Download Speed to the Development Board

If downloads to the development board are very slow, or timeout errors occur, switch to a USB 3.0 port. If you are developing in a virtual machine, set the USB interface to 3.0 as well.

2.12.17 Multiple Users Using Nuclei Studio on Linux

If multiple users need to use Nuclei Studio at the same time (not recommended), first open `Window -> Preferences` in the menu bar while running Nuclei Studio. Three settings must be modified in the dialog:

Open `MCU -> Global OpenOCD Path`, enter `openocd` for `Executable`, and enter `${eclipse_home}/toolchain/openocd/bin` for `Folder`.

Open `MCU -> Global QEMU Path`, enter `qemu-system-riscv32` for `Executable`, and enter `${eclipse_home}/tools/qemu` for `Folder`.

Open `MCU -> Global RISC-V Toolchains Paths`, select `RISC-V Nuclei GCC` for `Default toolchain`, and enter `${eclipse_home}/toolchain/gcc/bin` for `Toolchain folder`.

When finished, click `Apply and Close` to save the settings and close the dialog.

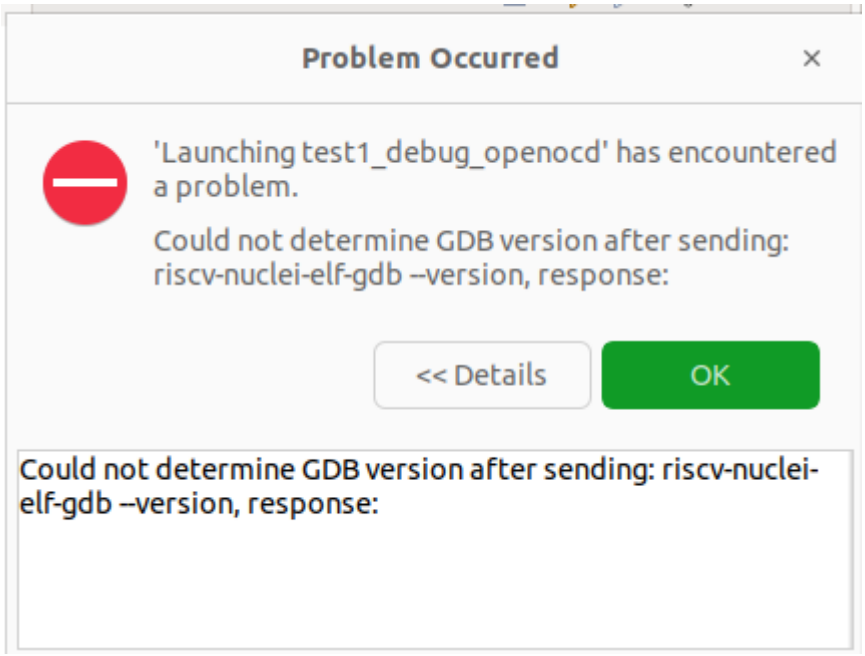
2.12.18 Two Serial Ports Recognized in Device Manager

After connecting the device, two serial ports may appear in Device Manager. The one with the larger COM number is used for serial output; you can connect to it with a tool such as a serial debugging assistant to view the print output. Do not open the serial port with the smaller COM number.



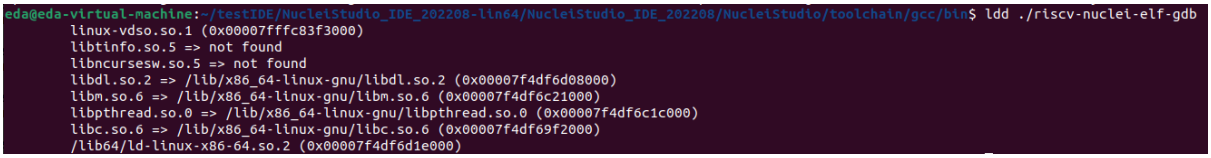
2.12.19 Error "Could Not Determine GDB Version After Sending: riscv-nuclei-elf-gdb version, response:" on Linux

The first time you use Nuclei Studio on Linux, the error `Could not determine GDB version after sending: riscv-nuclei-elf-gdb --version, response:` may be reported.



riscv-nuclei-elf-gdb was renamed to riscv64-unknown-elf-gdb in Nuclei Studio 2023.10 and later.

Use the command `ldd $(which riscv-nuclei-elf-gdb)` to check for missing dependencies.



After installing the required dependencies with the command `sudo apt install libncursesw5 libtinfo5`, the IDE runs normally. For details, see https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/9

2.12.20 Error Reported When Using QEMU on Linux

For example, when using QEMU on Ubuntu 20.04, the following error may be reported:

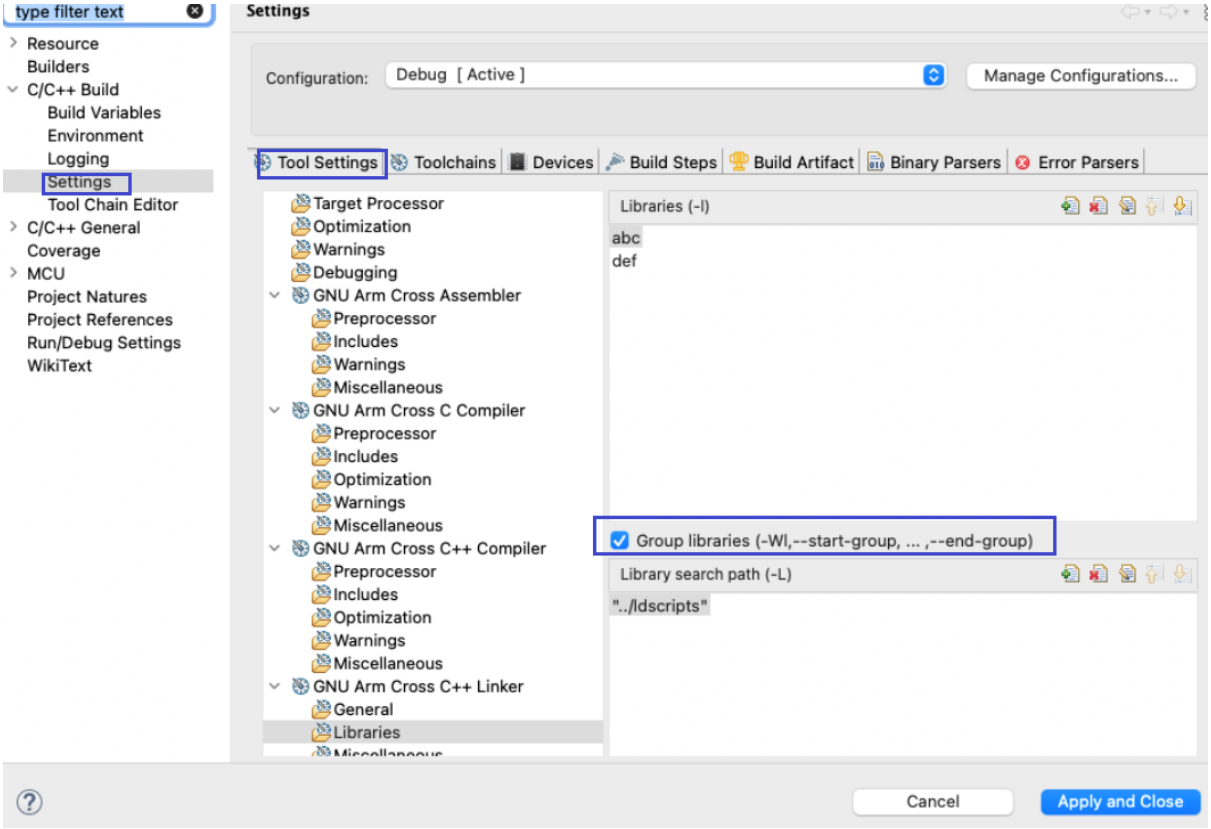
```
error while loading shared libraries: libfdt.so.1: cannot open shared object file:␣
↪No such file or directory
```

This is caused by missing dependencies such as libfdt.so. Install the corresponding dependencies for your version of Linux. For example, on Ubuntu 20.04, use the following command:

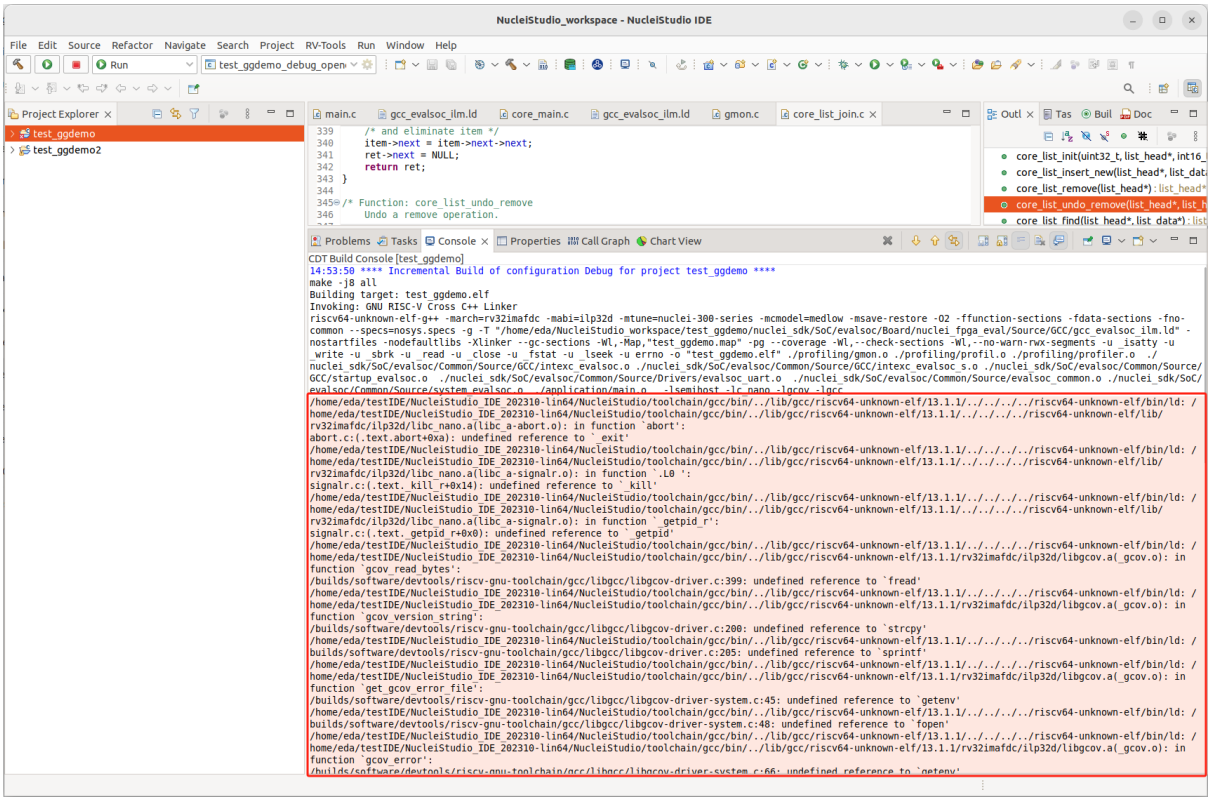
Install libfdt and other dependencies: `sudo apt install libfdt1 libpixman-1-0 libpng16-16 libasound2 libglib2.0-0`

2.12.21 Error About Missing Symbols When Linking the C Library During Project Build

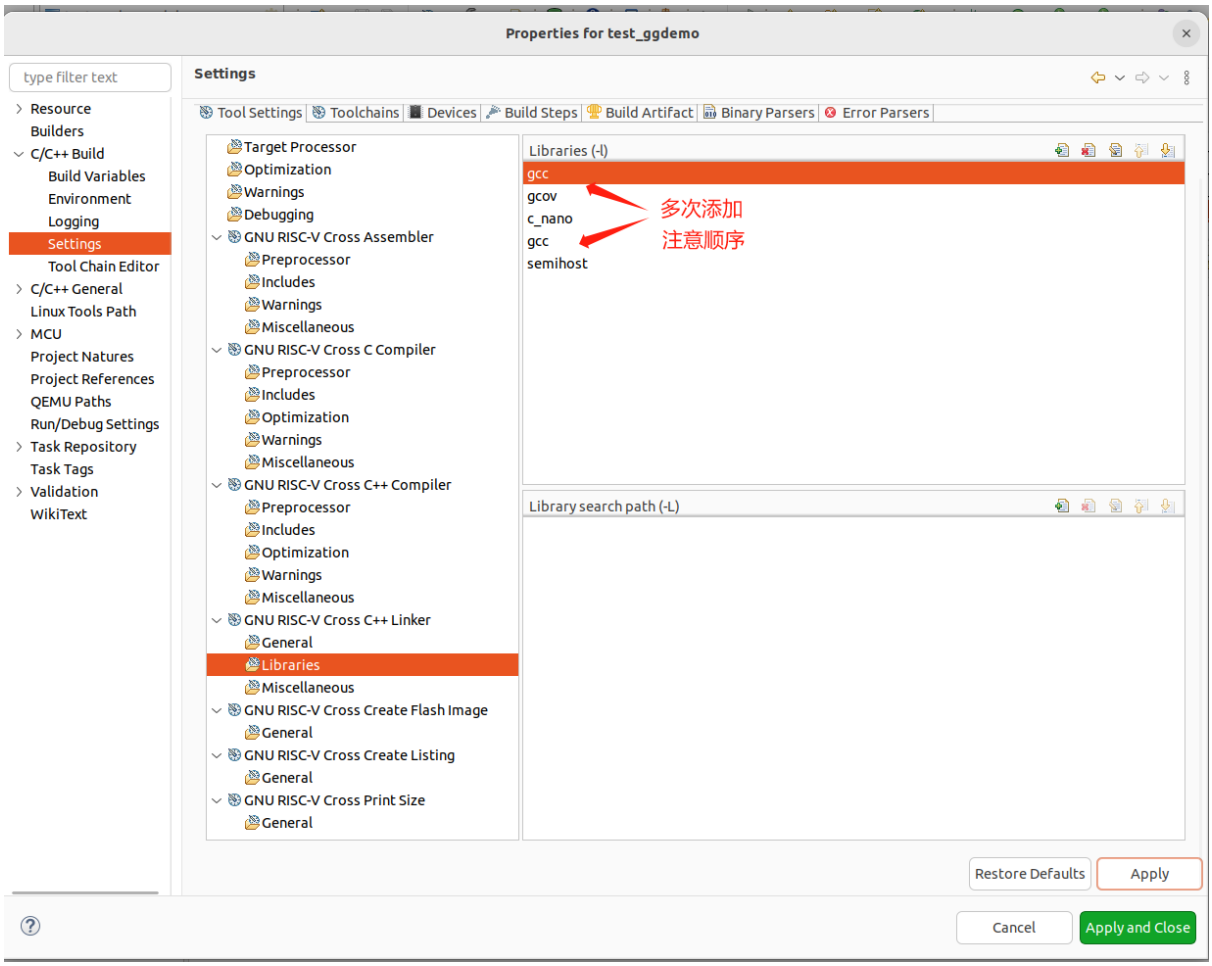
In Nuclei Studio 2024.06, this problem is resolved: simply check Group Libraries on the Linker -> Libraries page.



Because the handling of Libraries does not properly resolve the inter-dependencies between linked libraries, the build may fail when the libraries in use depend on each other. For details, see *https://github.com/eclipse-embed-cdt/eclipse-plugins/issues/592*. There are two ways to solve this problem.



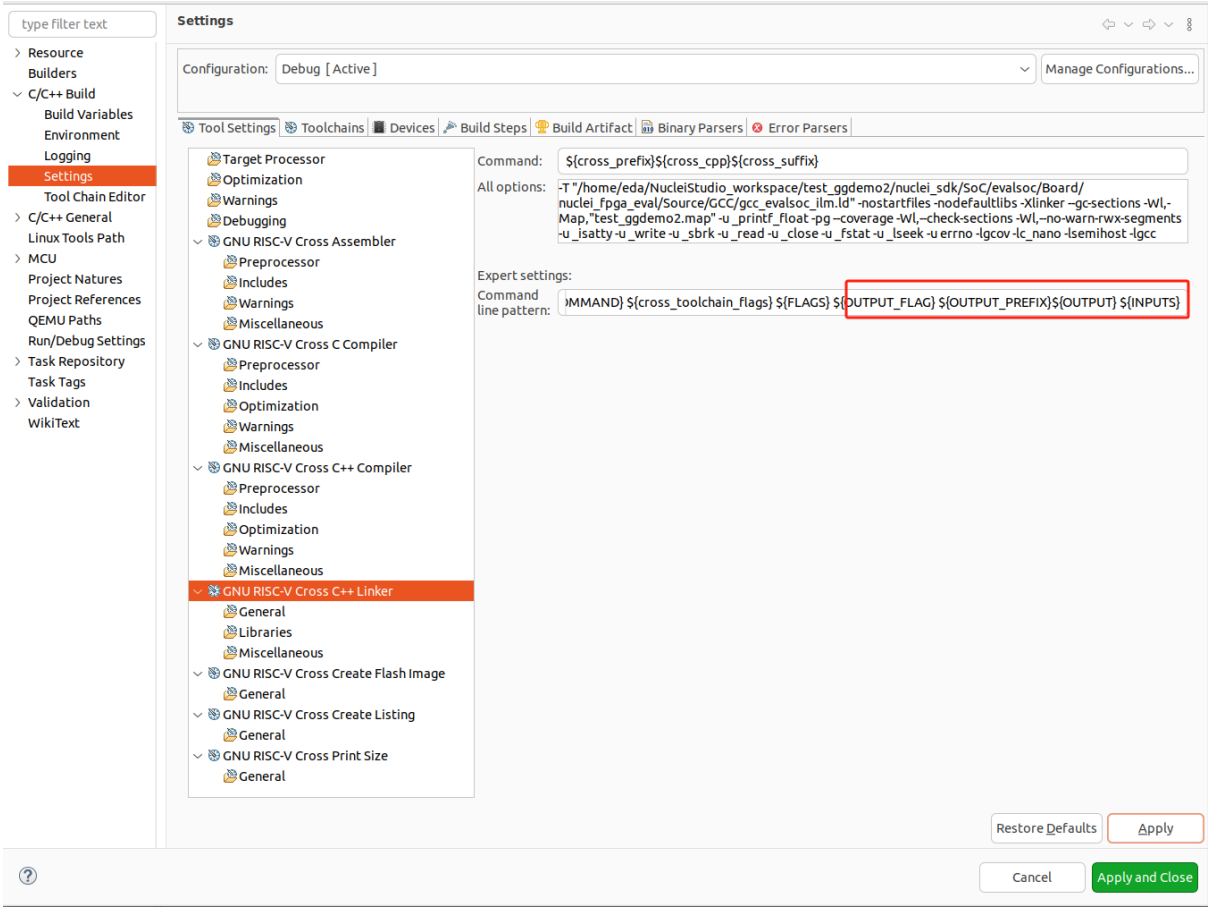
Either adjust the order of the Libraries, or add the same libraries multiple times.

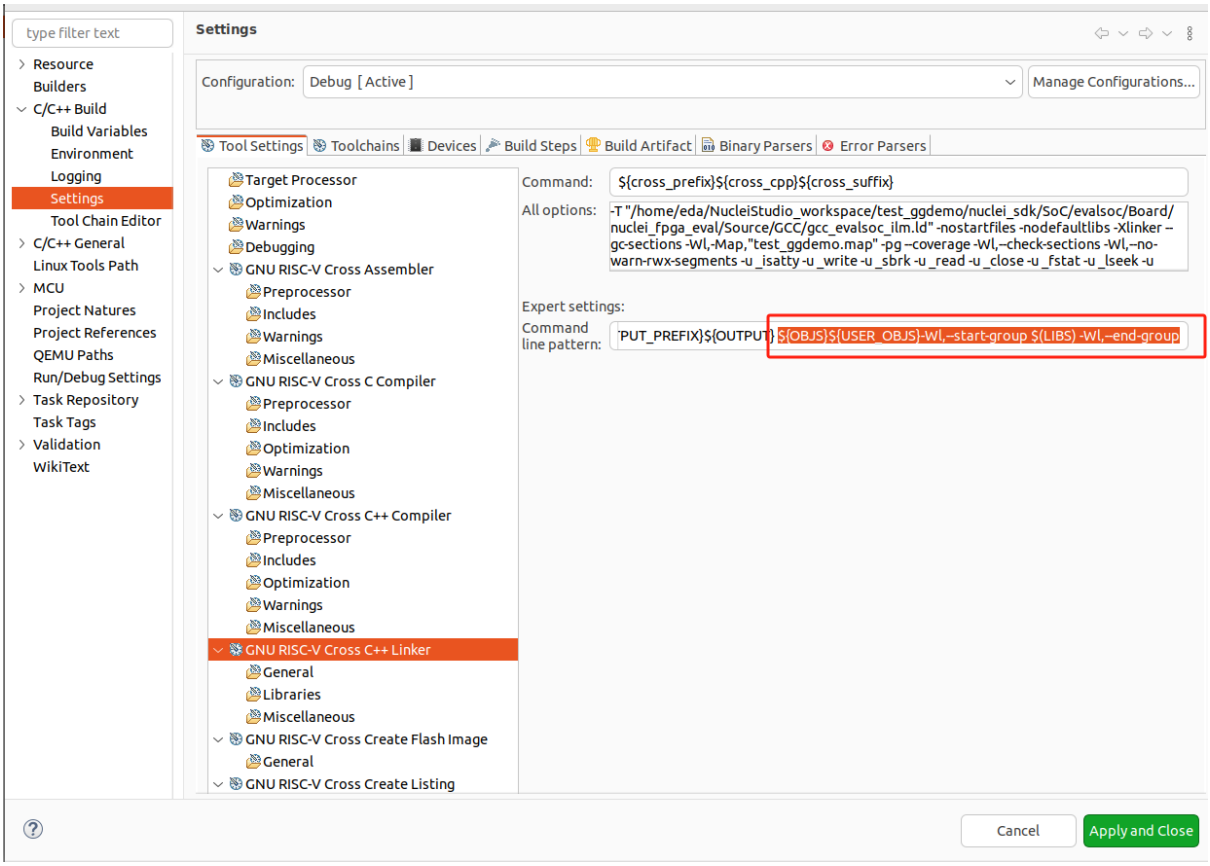


Open `C/C++ Build -> Settings -> Tool Settings -> GNU RISC-V Cross C++ Linker` and modify the Command line pattern as follows:

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

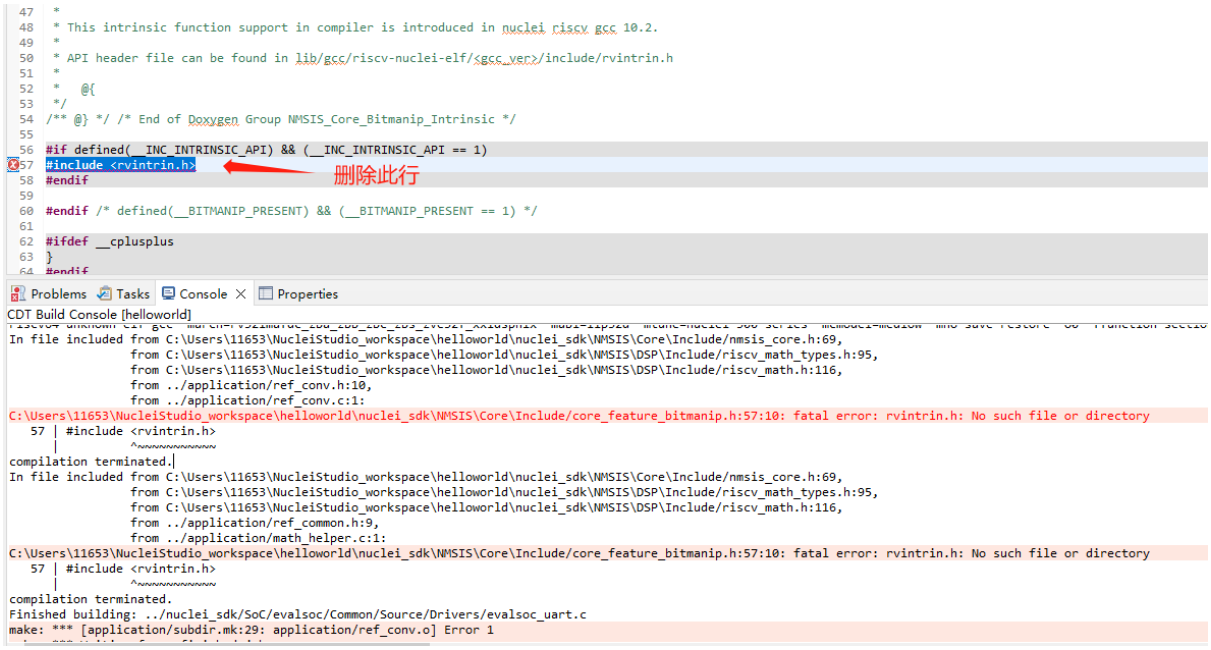
The IDE then processes the contents of Libraries as a group.





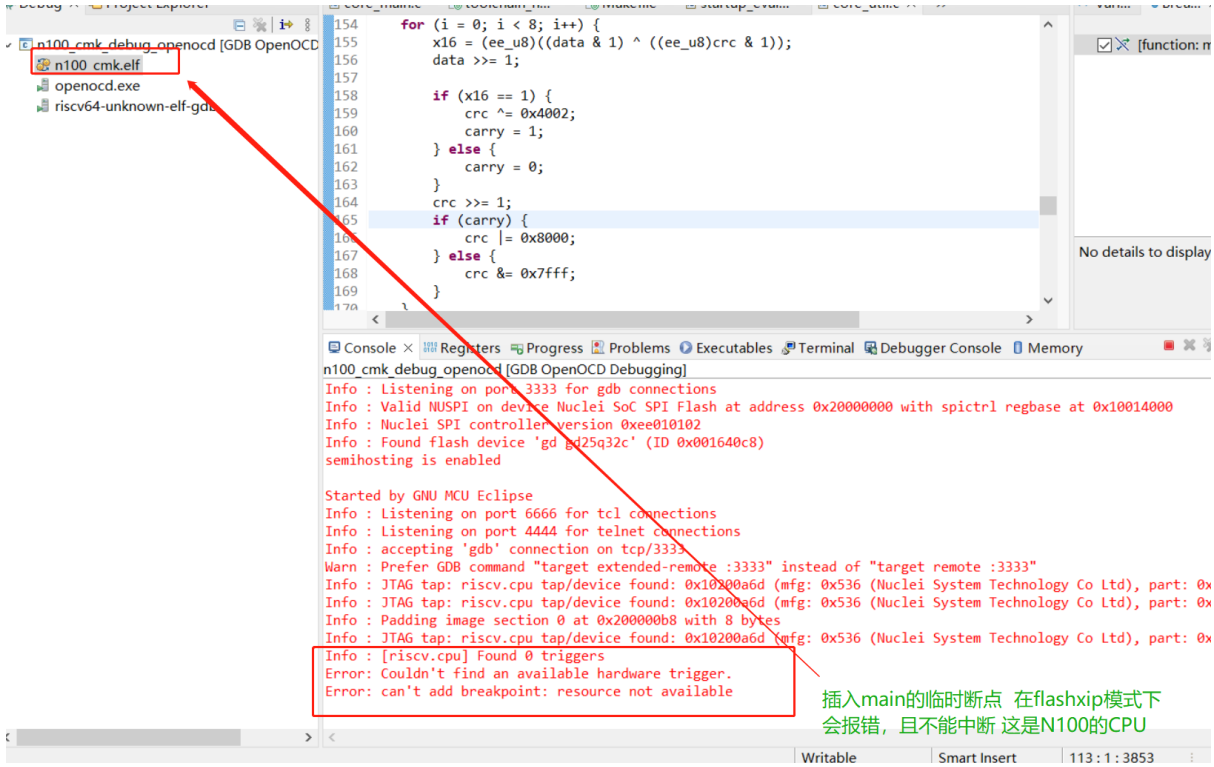
2.12.22 Build Error: Fatal Error: rvintrin.h: No Such File or Directory

In Nuclei Studio 2023.10, if a project created with an old SDK reports `fatal error: rvintrin.h: No such file or directory` when built, it is because the project uses `#include <rvintrin.h>`, which is required in GCC 10 but no longer needed in GCC 13. Simply delete this line and the build will pass.



2.12.23 Debug Error: Error: Couldn't Find an Available Hardware Trigger

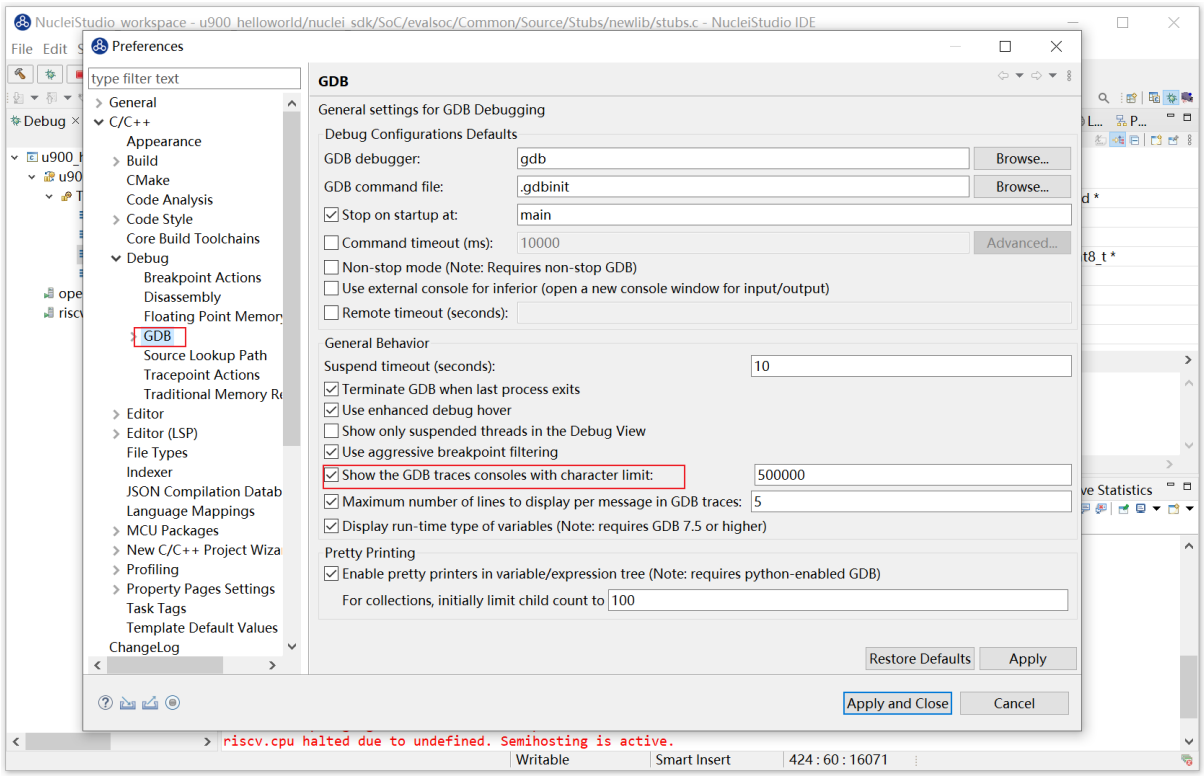
In the Nuclei Studio environment, when a project runs on CPU hardware without hardware breakpoints and the program is downloaded to Flash to run (for Nuclei SDK/Nuclei N100 SDK, this is the flash/flashxip DOWNLOAD mode), the program may fail to stop during debugging with the error message: `Error: Couldn't find an available hardware trigger`. This is because the program runs in Flash, where software breakpoints cannot be written successfully, and the CPU has no hardware breakpoints available.



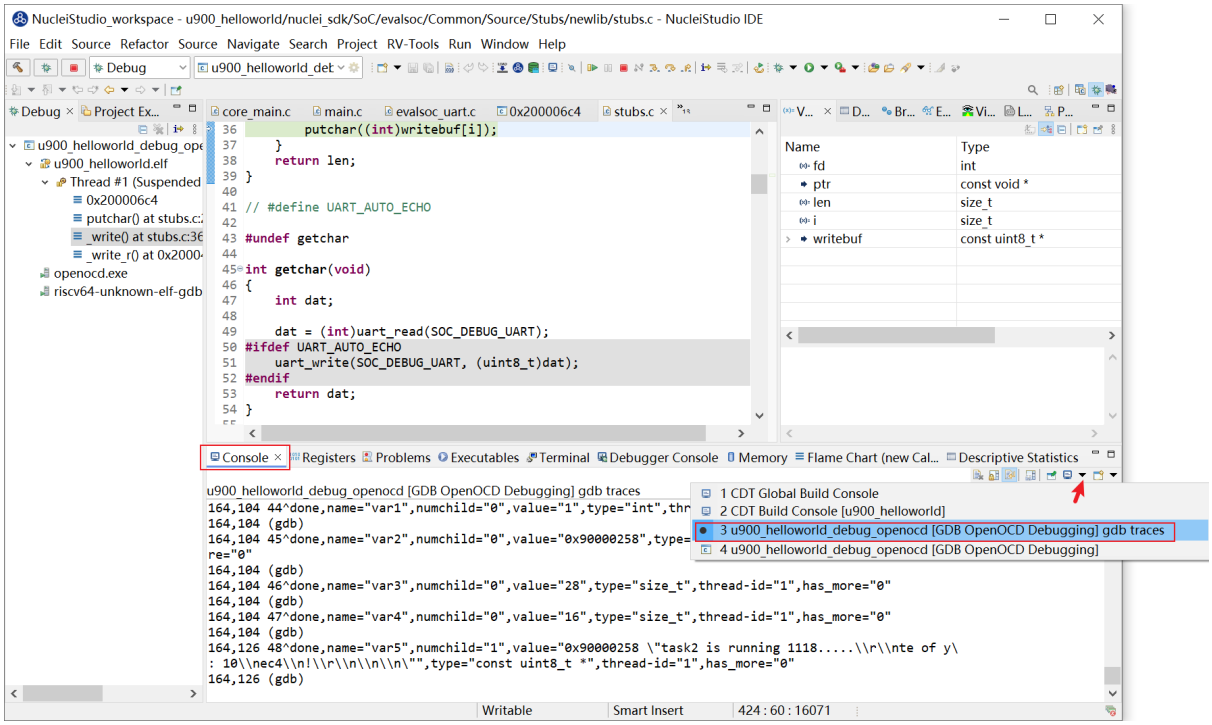
In this case, build the program to RAM so that the IDE can debug it (software breakpoints); to debug it in Flash, use the command line for now.

2.12.24 Viewing the GDB Trace While Debugging a Project

In the Nuclei Studio environment, the GDB Trace feature records in detail the GDB commands used while debugging a program. This feature is disabled by default; enable it if you need to view the GDB Trace. In the Nuclei Studio menu, click `Window -> Preferences`; in the Preferences window that opens, select `C/C++ -> Debug -> GDB`, find the `Show the GDB traces consoles with character limit` option, and check it.



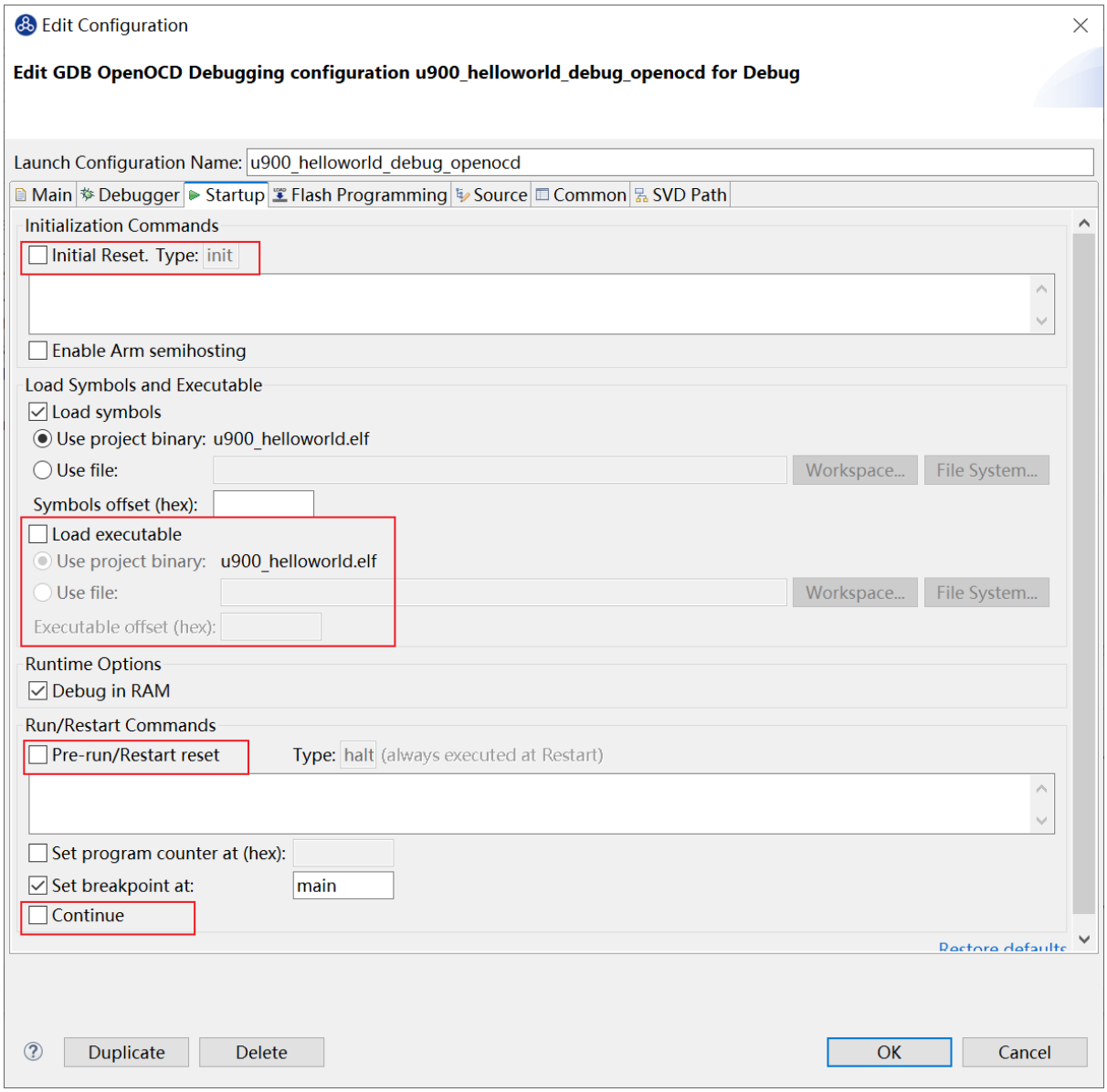
During debugging, click `Display Selected Console` in the `Console` window to switch to the GDB Trace console, where you can view the GDB Trace to help analyze problems encountered during debugging.



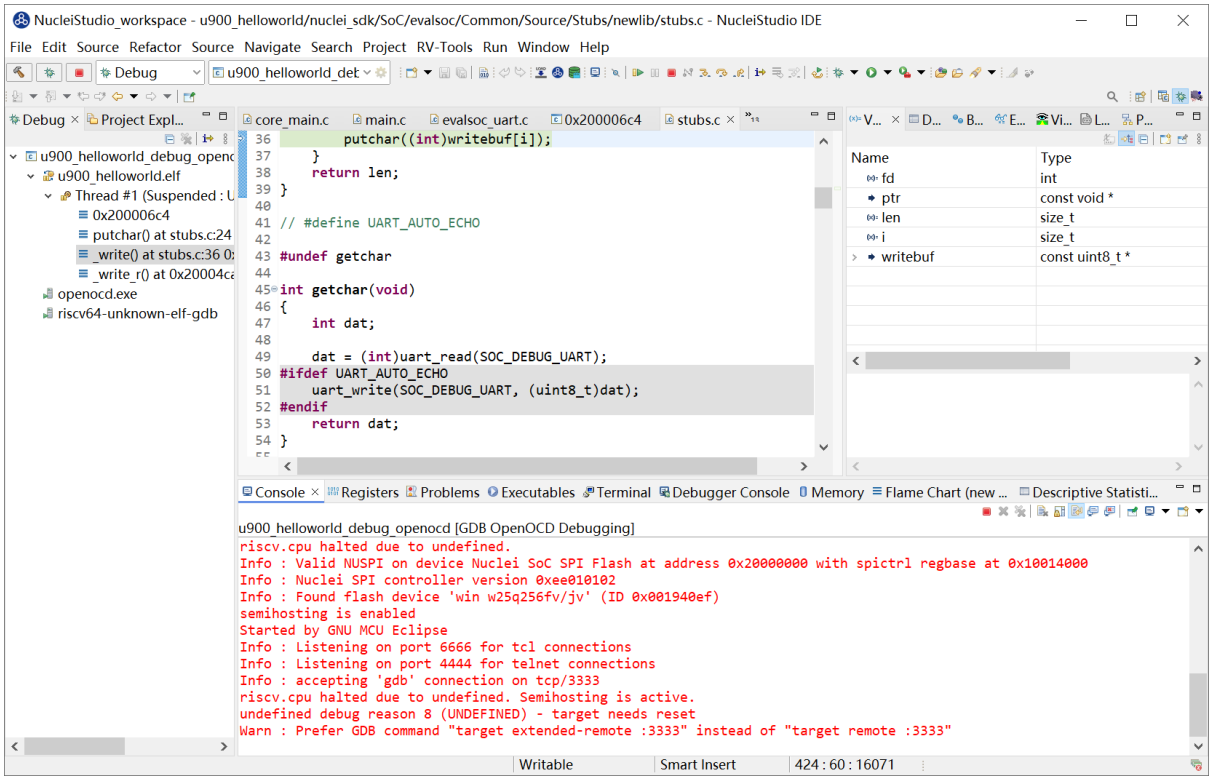
2.12.25 Connecting to a Running Target While Debugging a Project

When using OpenOCD to debug a project in the Nuclei Studio environment, connect to a running target as follows. Open the Debug_openocd configuration file, select the `Startup` tab, and modify the configuration as shown below:

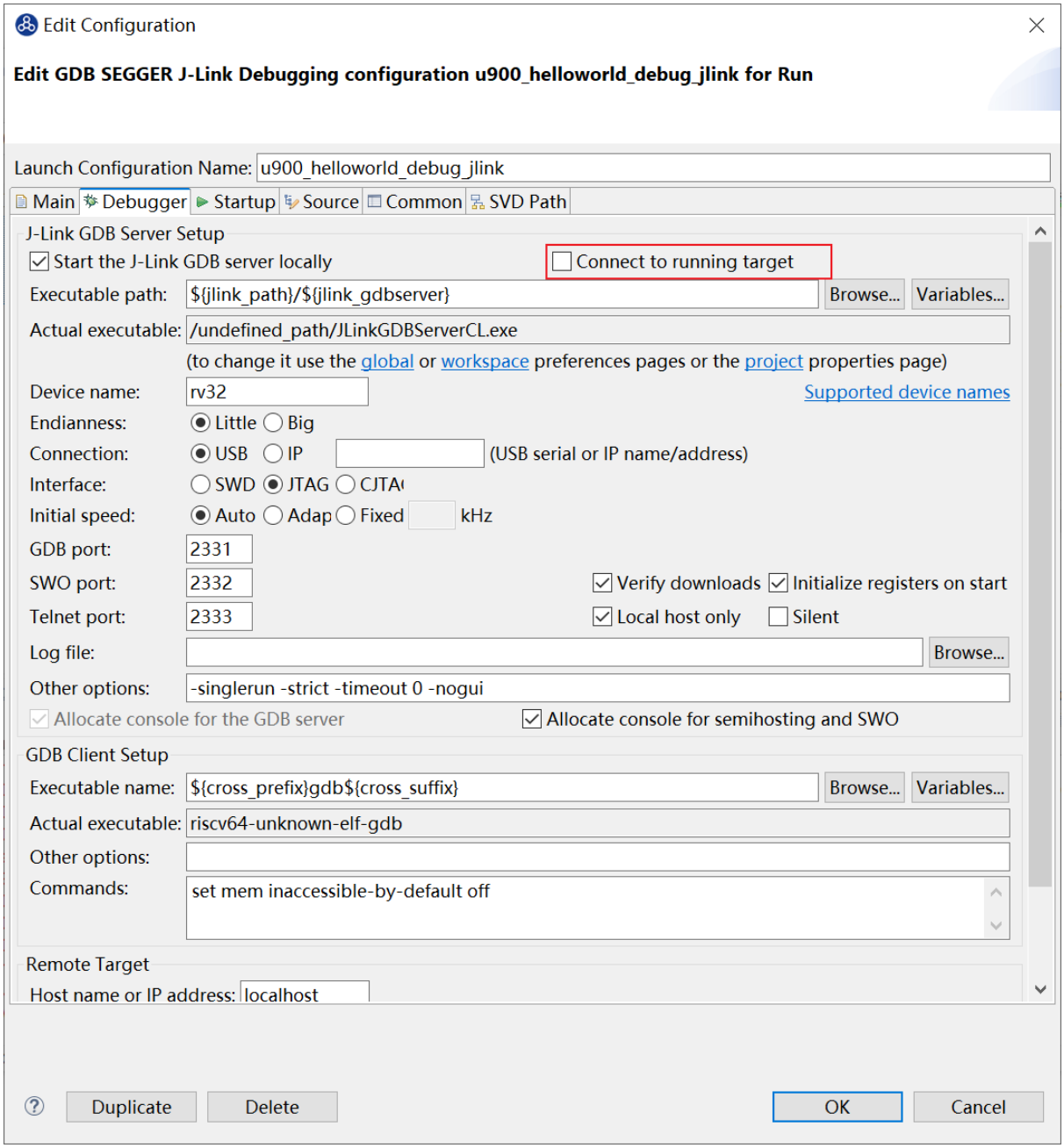
- Uncheck Init reset to avoid sending commands such as monitor reset that would reset the CPU;
- Uncheck Load executable to avoid downloading the ELF again;
- Uncheck Pre-run/restart commands to avoid sending commands such as monitor reset that would reset the CPU;
- Uncheck Continue, so that once OpenOCD is connected, it enters the debug state directly with execution paused.



Click the Debug button; you can see that it is now connected to the running target.

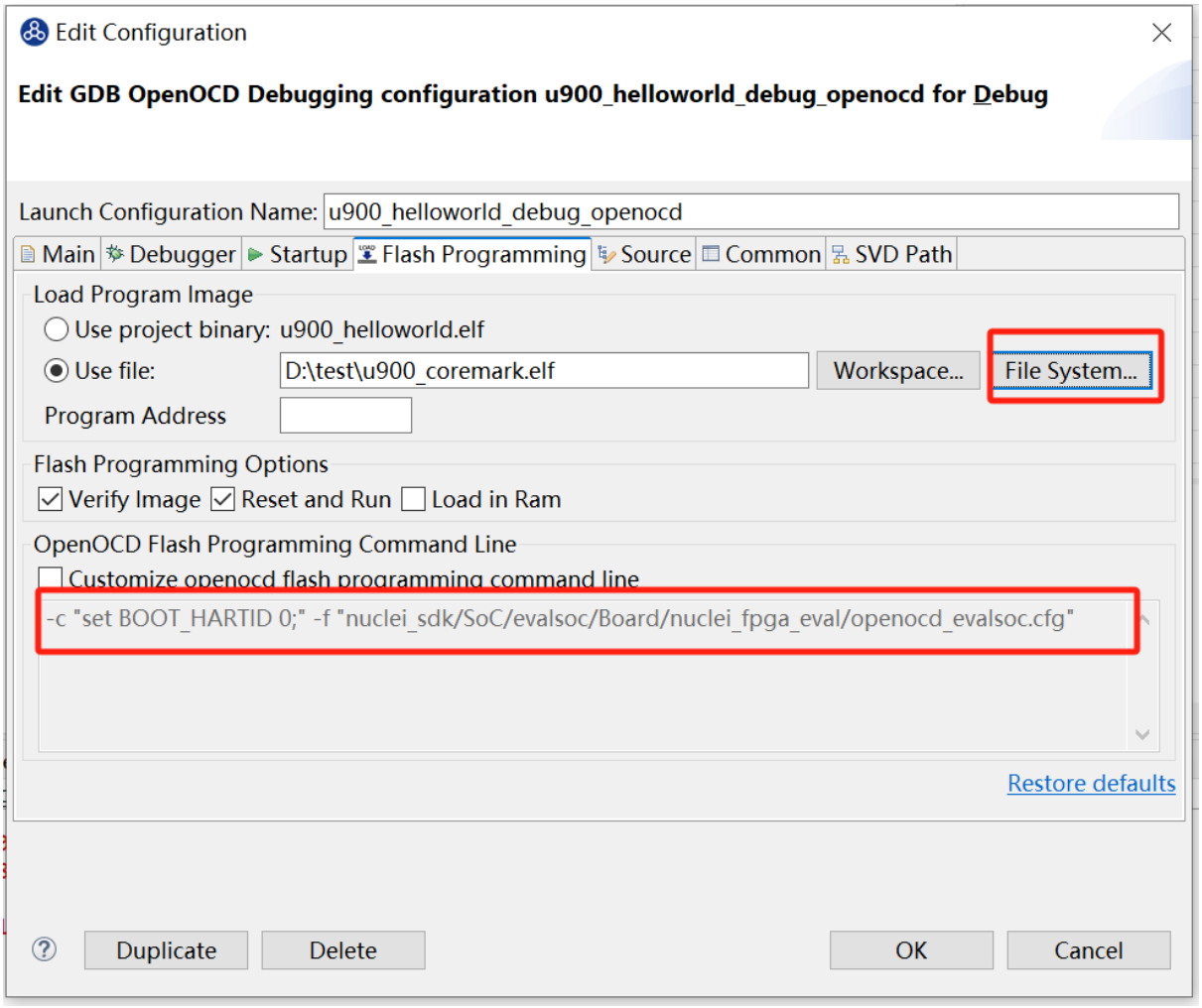


When using J-Link to debug a project, connect to a running target as follows. Open the Debug_jlink configuration file, select the `Debugger` tab, modify the configuration as shown below, and click the Debug button; you can see that it is now connected to the running target.

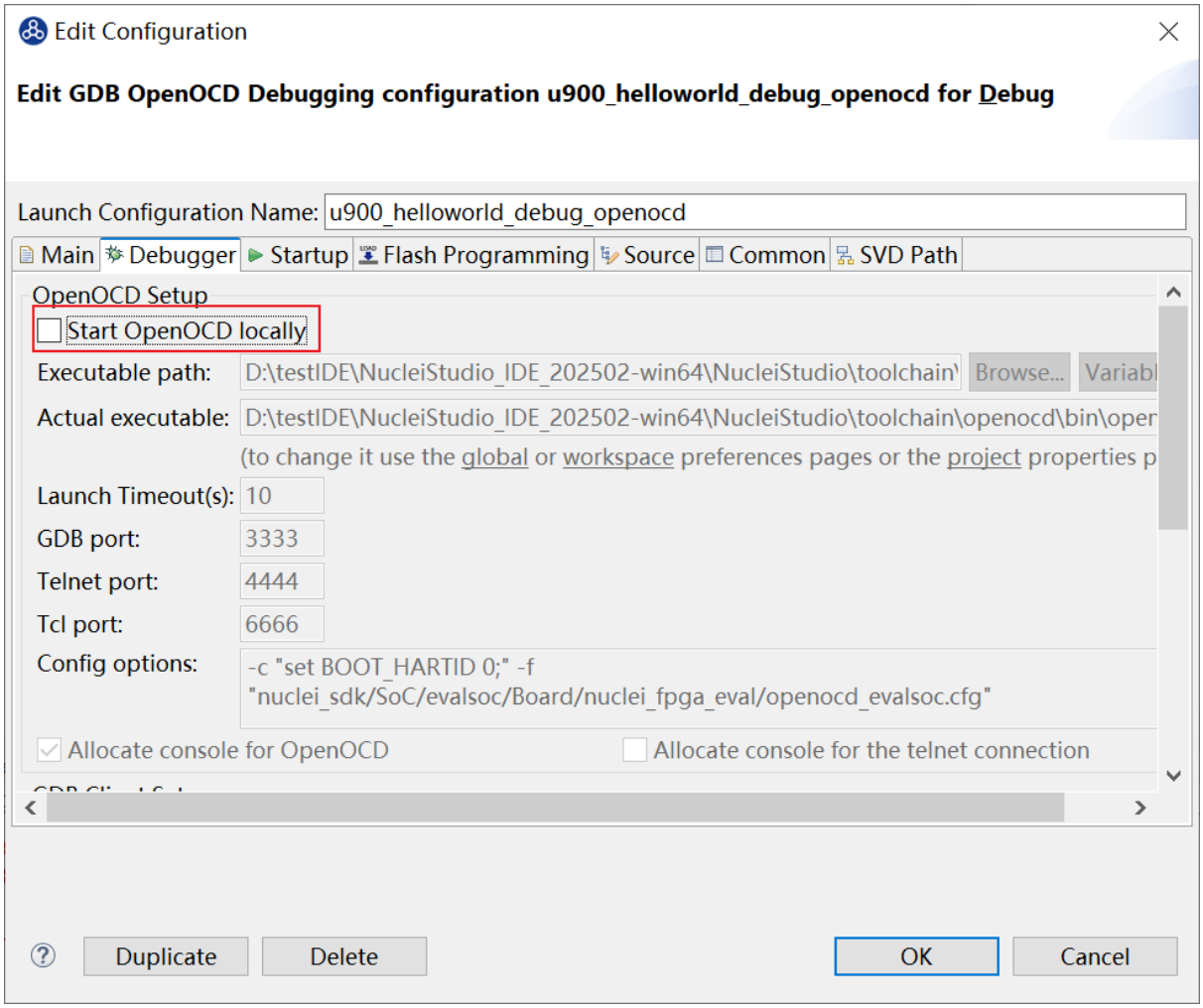


2.12.26 Known Defects in Flash Programming

When using the Flash Programming feature, selecting the file to load through `File System` triggers a code defect that makes the command incomplete. To avoid this problem, place the file to be loaded under the project directory.



Flash Programming depends on OpenOCD. If you use Flash Programming in Nuclei Studio with a configuration that does not launch OpenOCD, Nuclei Studio freezes. If this happens, check whether the configuration is correct.



2.12.27 Hardware Check Before Debugging a Project in Nuclei Studio

Before using the **Nuclei Studio IDE** for software development and debugging, make sure the target hardware (such as an FPGA-based RISC-V system) is correctly configured and running stably. Use the following steps to verify whether the hardware platform is ready for normal software debugging.

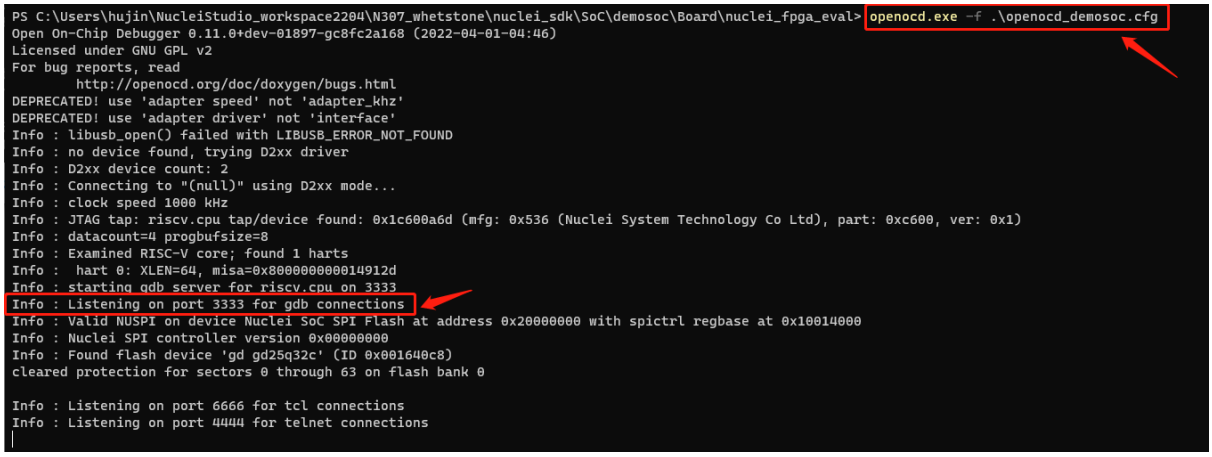
Checking Whether the OpenOCD Configuration File *openocd.cfg* Is Correct

OpenOCD is an important tool for JTAG debugging with the RISC-V core on the FPGA. Starting a debug session requires a correct *.cfg* configuration file. Use the following steps to confirm whether the *openocd.cfg* file is correct.

- **Use the OpenOCD tool under the Nuclei Studio installation directory:**
  - Windows: *NucleiStudio/toolchain/openocd/bin/openocd.exe*
  - Linux: *NucleiStudio/toolchain/openocd/bin/openocd*
- Copy the configuration file (such as *openocd_demosoc.cfg*) to the bin directory of OpenOCD.
- Run the command to start OpenOCD:

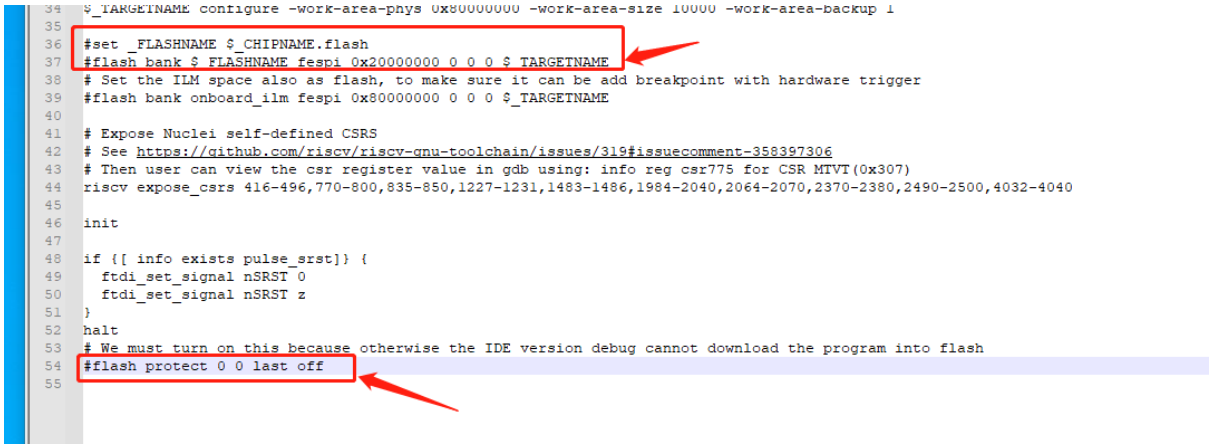
```
openocd -f openocd.cfg
```

If OpenOCD starts successfully and displays connection information, the JTAG interface and CPU on the FPGA board are working properly.



Notes:

- If there is no SPI Flash on the FPGA, or Flash programming is not needed, you can delete the Flash-related configuration items in the *.cfg* file.
- Make sure the JTAG pins, CPU type, frequency, and other parameters in the configuration file match your hardware.



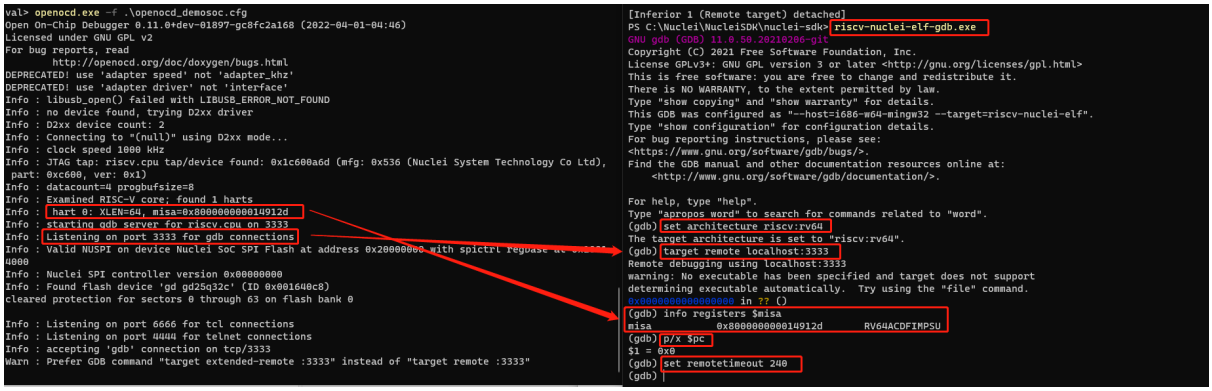
Using the Command Line to Confirm GDB and OpenOCD Can Communicate Properly

To further verify the hardware status, use the GDB command-line tool to establish a connection with OpenOCD.

- Start OpenOCD (see the previous step)
- Open a new terminal and go to the GDB tool path:
  - Windows: *NucleiStudio/toolchain/gcc/bin/riscv-nuclei-elf-gdb.exe*
  - Linux: *NucleiStudio/toolchain/gcc/bin/riscv-nuclei-elf-gdb*
- Start GDB and execute the following commands:

```
(gdb) set arch riscv:rv32     # or rv64, depending on the CPU architecture
(gdb) set remotetimeout 240   # set the remote connection timeout
(gdb) target remote :3333     # connect to the default OpenOCD port
```

If the connection succeeds, the JTAG interface and the RISC-V core on the FPGA are running normally.

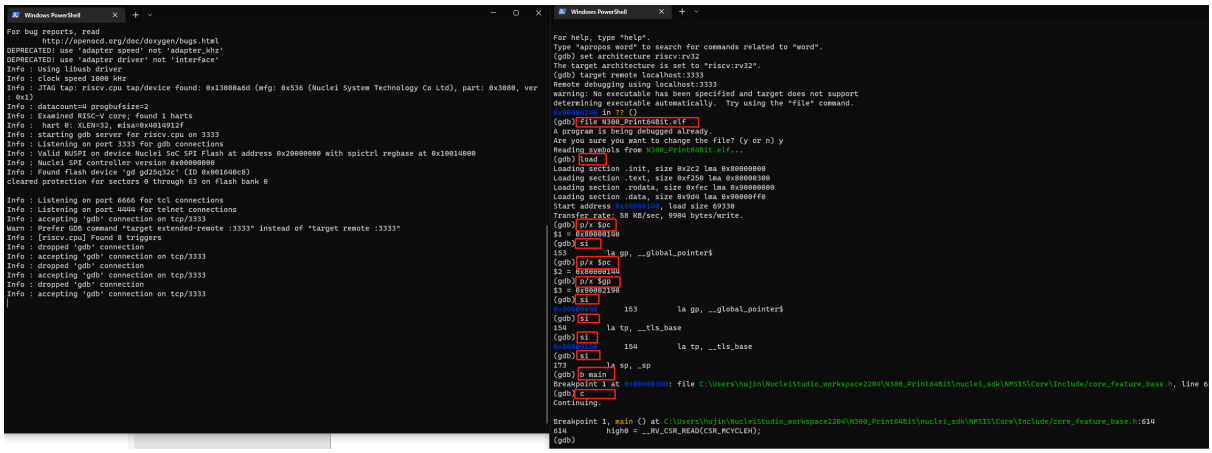


Possible issues:

- The CPU frequency on the FPGA is low → the first connection may time out; it is recommended to set *set remotetimeout 240* in advance. For details, see *Nuclei Studio cannot debug due to a low CPU frequency* (page 387).

Using GDB Commands to Check Low-Level Hardware Functions

Once GDB is successfully connected, you can verify the functions of the CPU, SRAM, and peripheral registers through a series of commands.



Test items and commands:

Read and write general-purpose registers (GPRs)

```
(gdb) p/x $pc          # view the current PC value
(gdb) p/x $a0          # view the a0 register value
```

Read and write CSR registers

```
(gdb) info reg $mstatus     # view the mstatus register
(gdb) info reg $misa        # view the supported instruction set architecture
```

Read and write SRAM memory (verify memory stability)

- Prepare a test bin file and load it into SRAM:

```
(gdb) restore test.bin binary 0x80000000
```

- Dump it from memory and compare:

```
(gdb) dump binary memory dump.bin 0x80000000 0x80001000
```

- Check whether *test.bin* and *dump.bin* are exactly identical.

If the data is inconsistent, the SRAM may have timing or interface problems.

8-bit/16-bit data access test

- Verify whether byte-level access is supported:

```
(gdb) x/1bx 0x80000000      # read a single byte
(gdb) x/1hx 0x80000000      # read a halfword (16-bit)
```

Read and write SoC peripheral registers

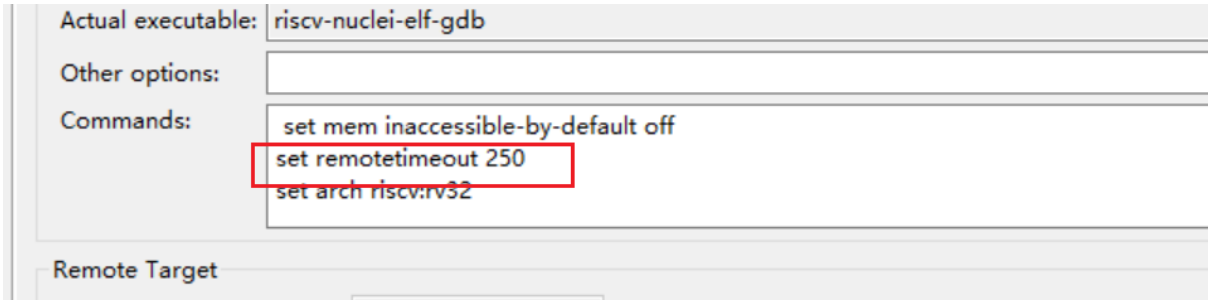
- Perform read/write tests on registers mapped in the peripheral address space:

```
(gdb) p/x *(int *)0x10000000   # read a peripheral register
(gdb) set *(int *)0x10000000 = 0x1  # write a value
```

```
(gdb) restore ../ff-4K.bin binary 0x80000000
Restoring binary file ../ff-4K.bin into memory (0x80000000 to 0x80001000)
(gdb) dump binary memory test.bin 0x80000000 0x80000FFF
(gdb) x/50xw 0x80000000
0x80000000:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000010:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000020:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000030:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000040:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000050:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000060:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000070:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000080:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x80000090:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x800000a0:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x800000b0:     0xffffffff      0xffffffff      0xffffffff      0xffffffff
0x800000c0:     0xffffffff      0xffffffff
(gdb) set *(short *)0x80000000=0x1122
(gdb) x/8b 0x80000000
0x80000000:     0x22    0x11    0xff    0xff    0xff    0xff    0xff    0xff
(gdb) set *(char *)0x80000000=0x55
(gdb) x/8b 0x80000000
0x80000000:     0x55    0x11    0xff    0xff    0xff    0xff    0xff    0xff
(gdb) p/x $a0
$4 = 0x0
```

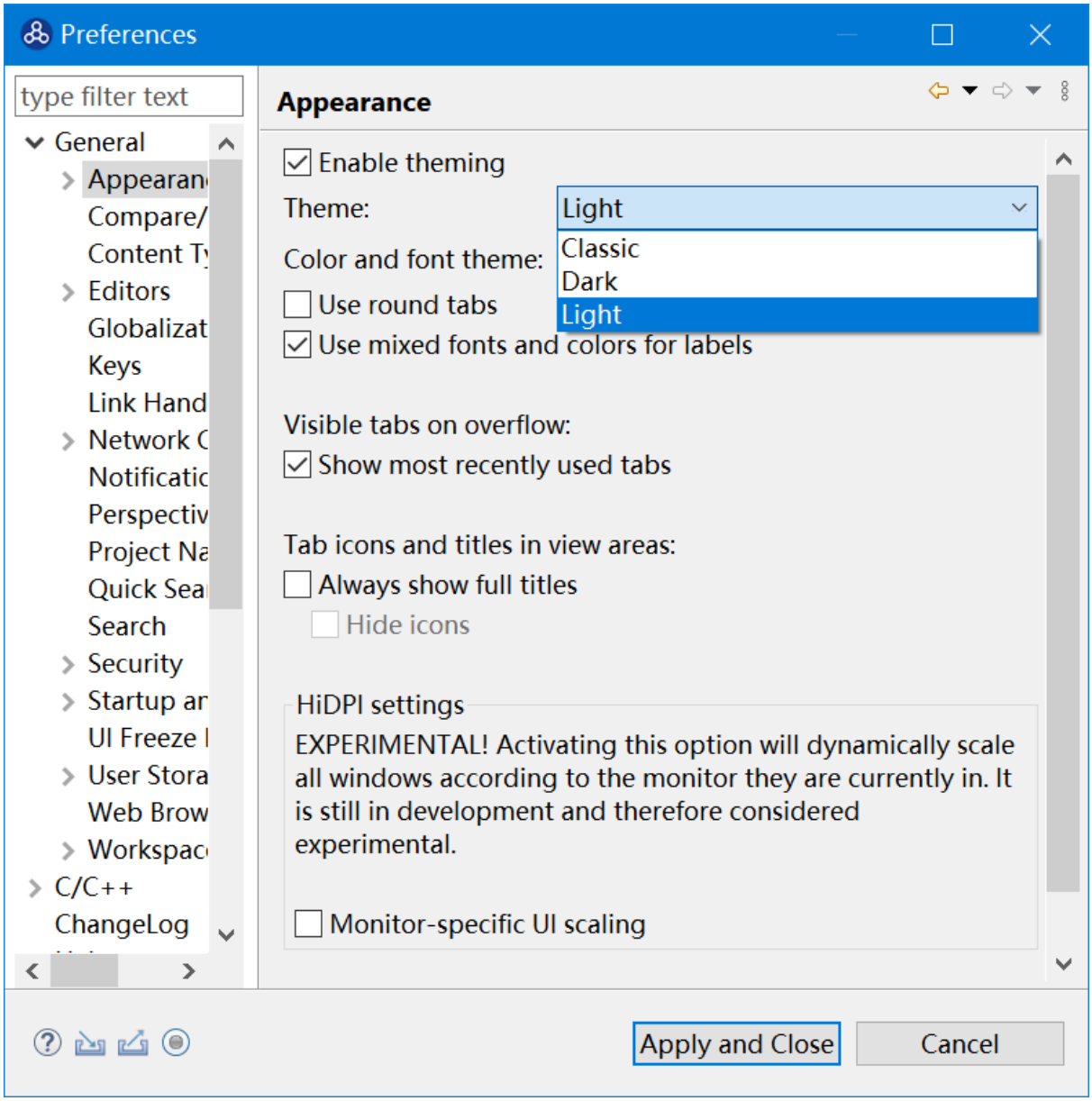
2.12.28 Nuclei Studio Cannot Debug Due to a Low CPU Frequency

If the CPU frequency is low, Nuclei Studio debugging may fail or become unstable. In this case, it is recommended to lower the JTAG frequency in the OpenOCD configuration file; it should generally be less than half of the CPU frequency, and about one quarter of the CPU frequency is recommended. When using Nuclei Studio, also increase the timeout value in the *set remotetimeout 250* command (in seconds) to improve the connection success rate.



2.12.29 Setting the Nuclei Studio Theme Colors

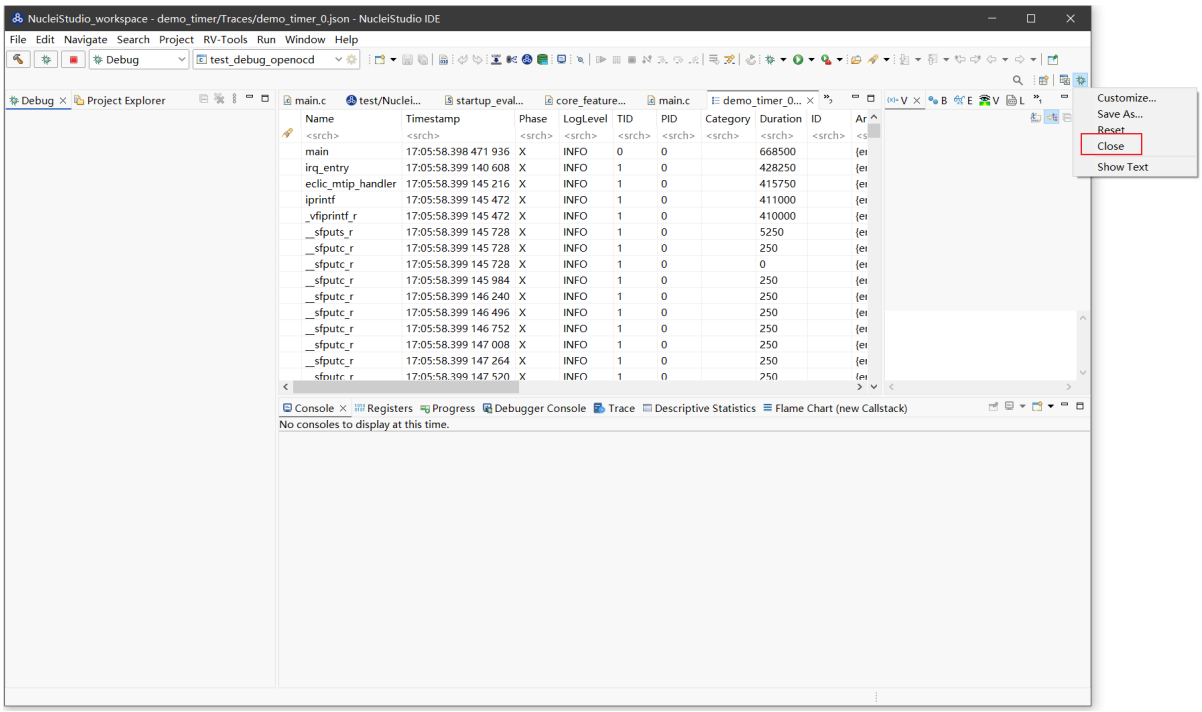
Nuclei Studio is built on Eclipse, and its theme colors have not been modified, so it inherits Eclipse ' s theme color functionality. To change the theme colors of Nuclei Studio, refer to the methods for changing Eclipse theme colors. You can also quickly configure a general theme through the preferences.



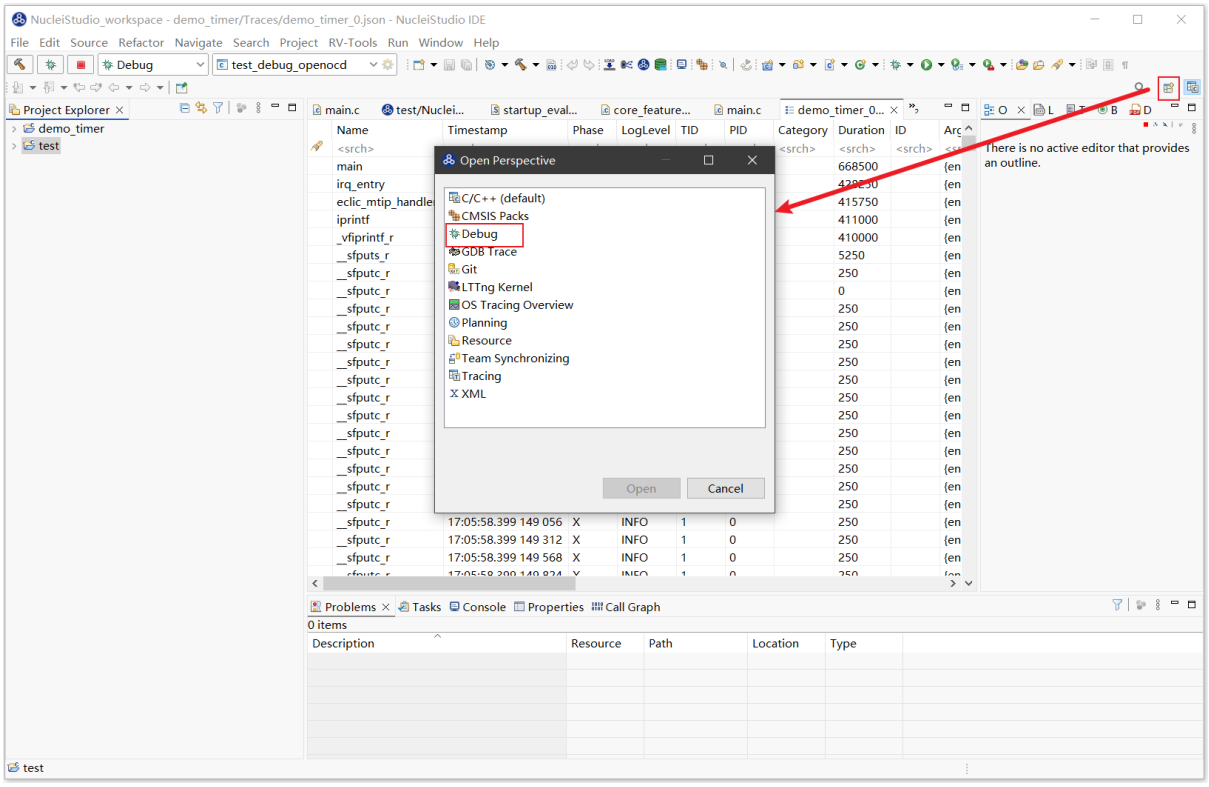
2.12.30 No Project Creation Menu in the Debug Perspective

Nuclei Studio 2025.10 updated the menu display logic so that common menus can be shown in the Debug perspective. If your current version is Nuclei Studio 2025.10 or later and the `New Nuclei RISC-V C/C++ Project` menu is missing under `File -> New` in the Debug perspective, restart the Debug perspective to refresh the menus.

Close the Debug perspective from its upper-right corner.

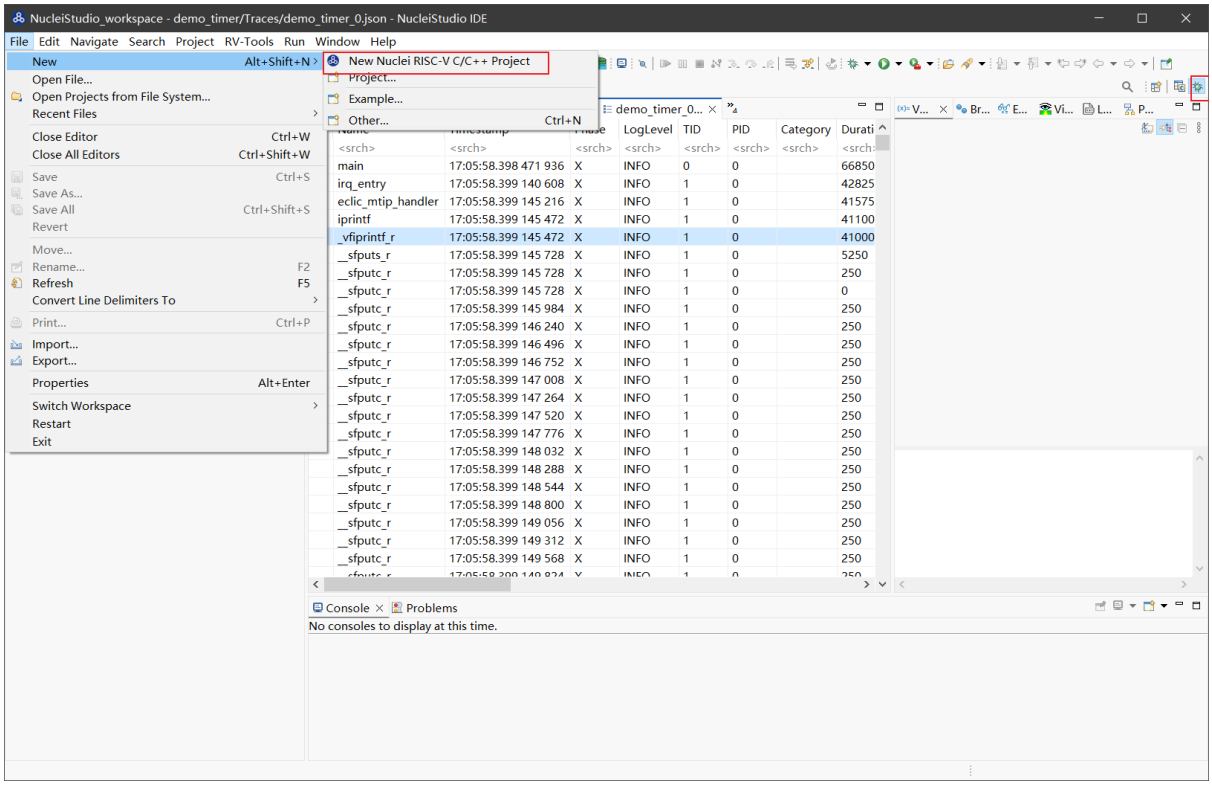


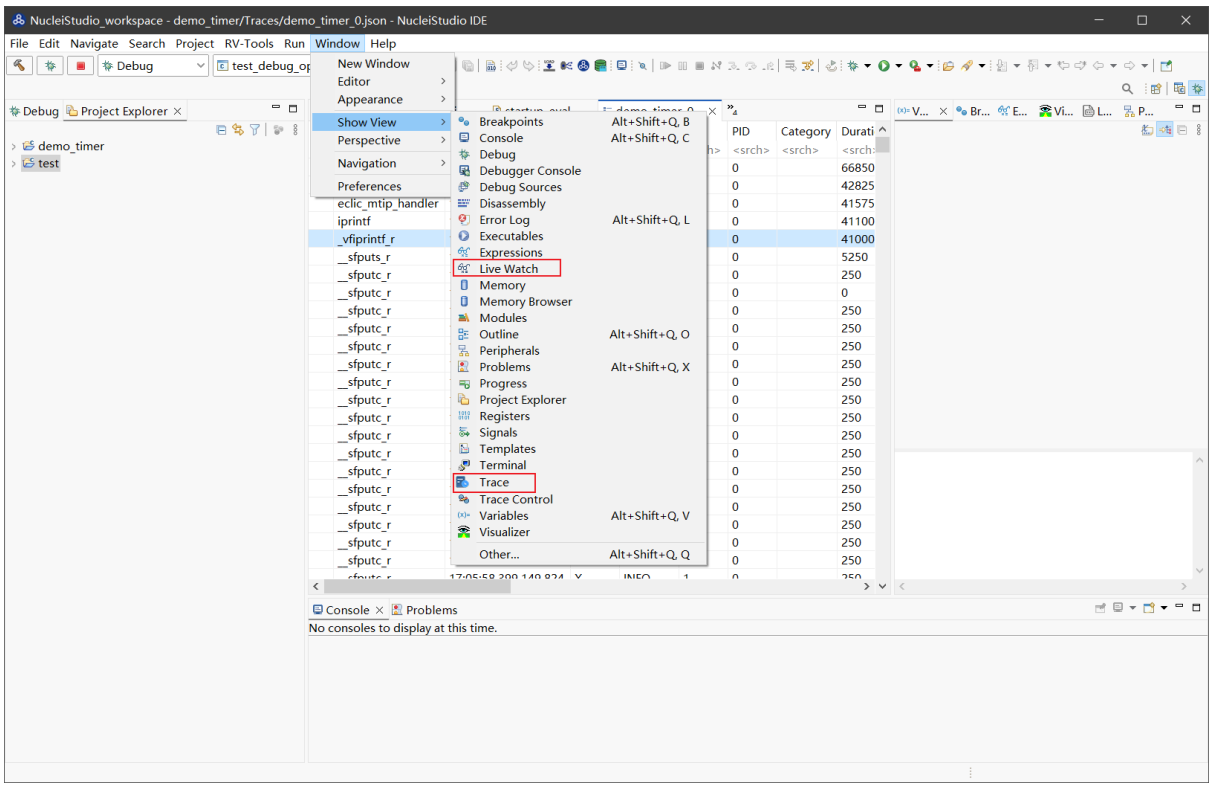
Then reopen the Debug perspective.



The newly added `New Nuclei RISC-V C/C++ Project` menu then appears under `File -> New`.

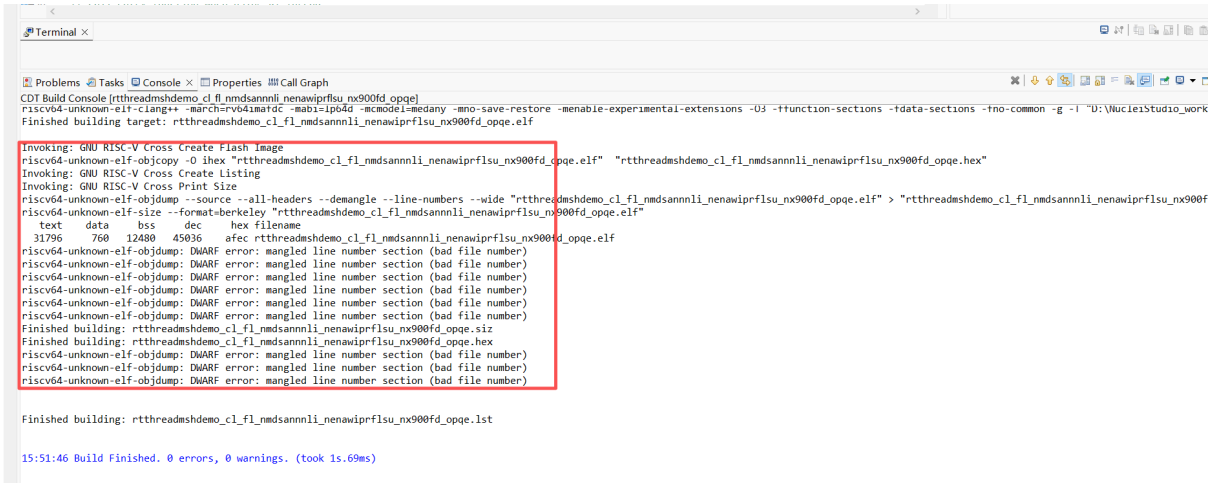
Menus commonly used during debugging, such as Trace and Live Watch, also appear under `Window -> Show View`.





2.12.31 Build Error: DWARF Error: Mangled Line Number Section (Bad File Number)

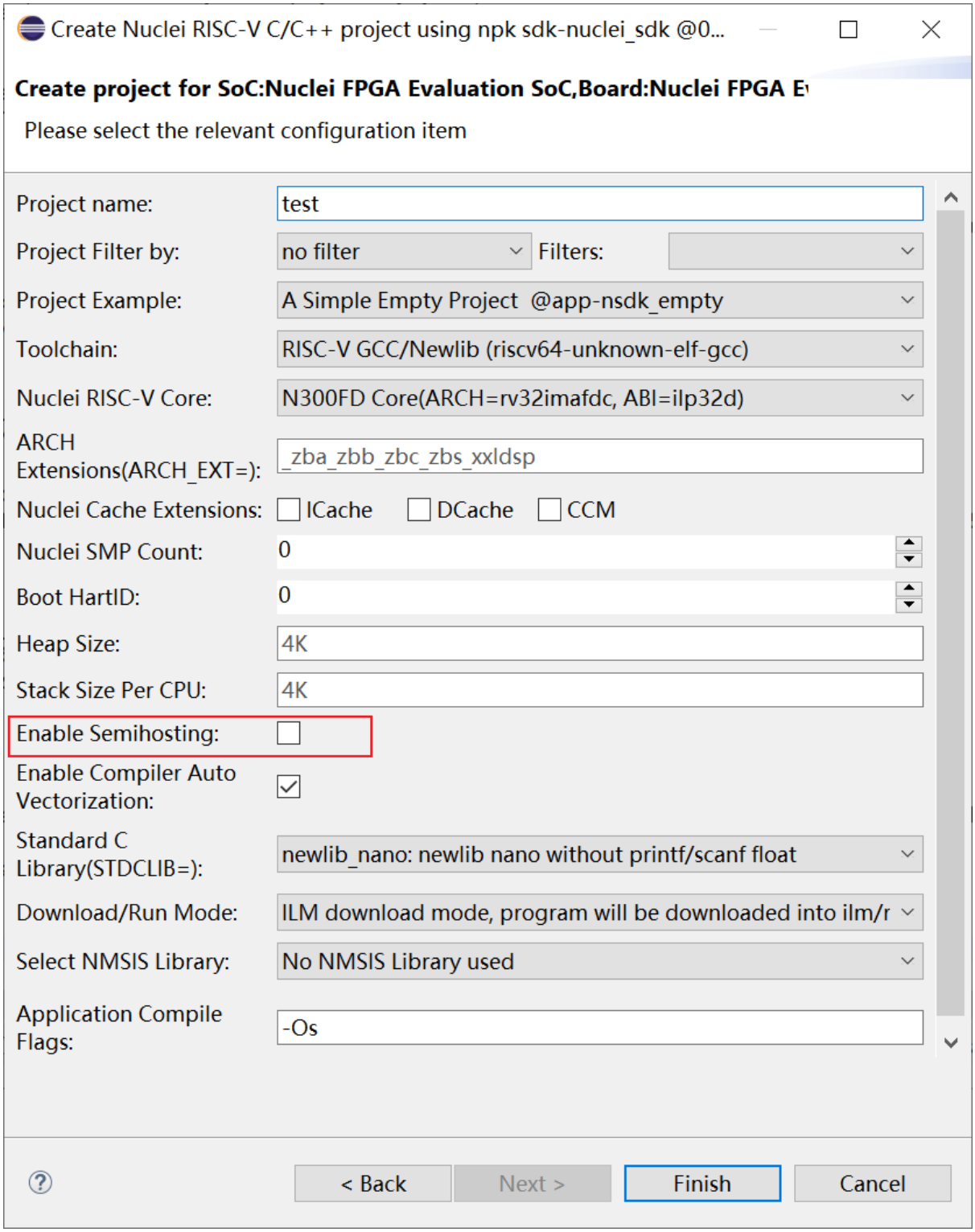
When building a project with Clang in Nuclei Studio, the warning `DWARF error: mangled line number section (bad file number)` may appear. This is usually because Nuclei Studio uses the `objdump` from the GCC toolchain to process the ELF file generated by Clang, and compatibility differences trigger this warning. This problem generally does not affect the normal build and execution of the program.



2.12.32 How to Build a Project That Supports Semihosting

When creating a project with Nuclei SDK, check `Enable Semihosting` to generate a project that supports Semihosting directly.

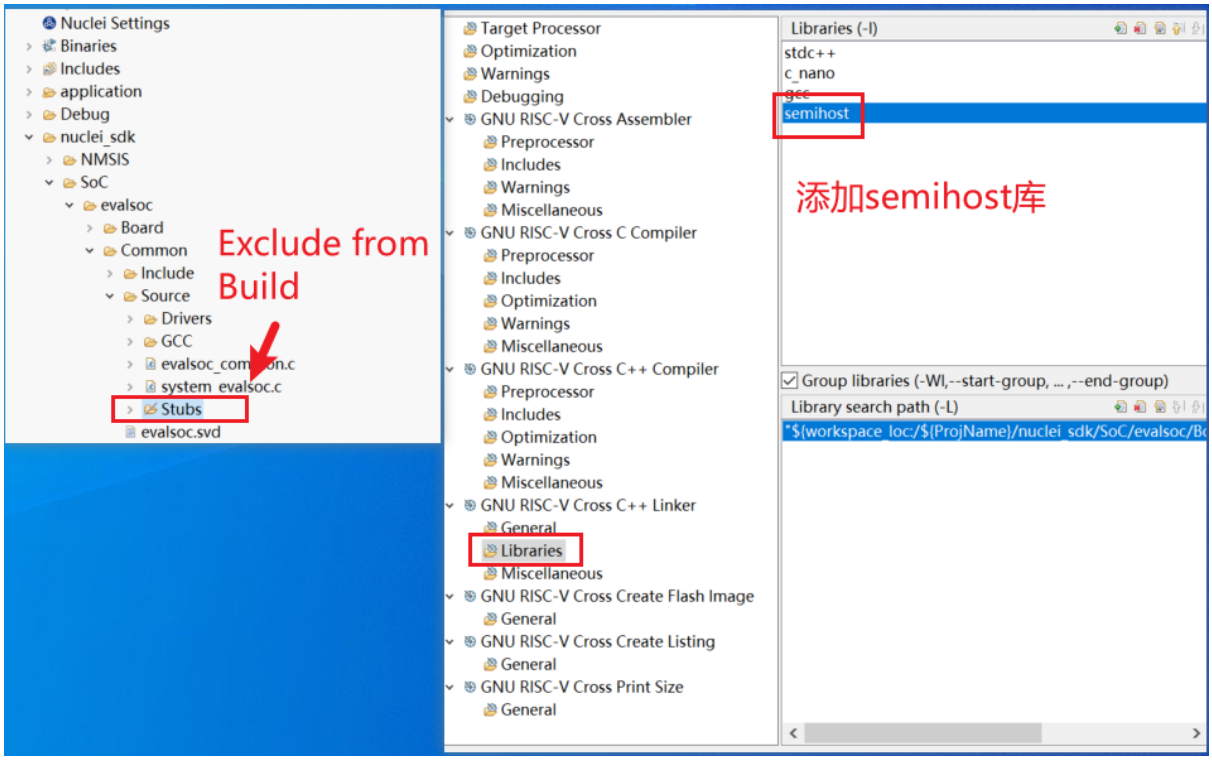
For projects that have already been created, or projects not created with Nuclei SDK, enable Semihosting manually with the following steps.



This kind of project usually implements basic I/O through `Stubs` functions. When enabling Semihosting, you must replace the original `Stubs` implementation with the implementation provided by the semihost library.

First, exclude the original `Stubs` code. Taking a project created with Nuclei SDK as an example, select `SoC\evalsoc\Common\Source\Stubs`, then right-click and choose `Resource Configurations -> Exclude from Build...` to exclude this code from the build. When Semihosting is no longer needed, re-include it in the same way to switch back quickly.

The figure below takes newlib as an example: when linking, the `semihost` library must be used. If you use libncrt, replace `fileops_uart` with `fileops_semi`. If you use the `zcc` toolchain, the `semihost` library must also be added during the link stage.



2.12.33 How to Debug a Semihosting-Enabled Project with J-Link

The IDE debugs with J-Link through the J-Link GDB Server by default, but this method does not support Semihosting. To enable Semihosting while using J-Link as the debugger, refer to the following two solutions:

1. Use J-Link ' s native Semihosting functionality. This solution must be used together with Ozone[20]; for details, see SEGGER ' s Ozone offers enhanced debugging with RISC-V Semihosting[21].
2. Use OpenOCD + J-Link for debugging; see Using J-Link with OpenOCD[22]. This solution only supports the JTAG interface and does not support cJTAG.

For the OpenOCD + J-Link solution, use JLinkConfig or Zadig[23] to switch the J-Link USB driver to WinUSB, then enable Semihosting in the OpenOCD configuration file. The following configuration can be used as a reference:

```
adapter_khz 1000
# The below 2 lines configure openocd to work with jlink
interface jlink
transport select jtag
#reset_config trst_only
set _CHIPNAME riscv
jtag newtap $_CHIPNAME cpu -irlen 5
set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -chain-position $_TARGETNAME
$_TARGETNAME configure -work-area-phys 0x80000000 -work-area-size 10000 -work-area-
↪backup 1
set _FLASHNAME $_CHIPNAME.flash
flash bank $_FLASHNAME fespi 0x20000000 0 0 0 $_TARGETNAME
# Set the ILM space also as flash, to make sure it can be add breakpoint with␣
↪hardware trigger
#flash bank onboard_ilm fespi 0x80000000 0 0 0 $_TARGETNAME
# Expose Nuclei self-defined CSRS
```

(continues on next page)

[20] https://www.segger.com/products/development-tools/ozone-j-link-debugger/
[21] https://www.segger.com/news/pr-241015-ozone-riscv-semihosting/
[22] https://kb.segger.com/OpenOCD#Using_J-Link_with_OpenOCD
[23] https://zadig.akeo.ie/

(continued from previous page)

```
# See https://github.com/riscv/riscv-gnu-toolchain/issues/319#issuecomment-
↪358397306
# Then user can view the csr register value in gdb using: info reg csr775 for CSR␣
↪MTVT(0x307)
# riscv expose_csrs 416-496,770-800,835-850,1227-1231,1483-1486,1984-2040,2064-
↪2070,2370-2380,2490-2500,4032-4040
init
if {[ info exists pulse_srst]} {
  ftdi_set_signal nSRST 0
  ftdi_set_signal nSRST z
}
halt
# We must turn on this because otherwise the IDE version debug cannot download the␣
↪program into flash
flash protect 0 0 last off
arm semihosting enable
```

2.12.34 Debug Session Hangs Due to Address Preloading in the Memory Window

Problem description

When using the Memory window in the IDE to view memory, the interface freezes and the debug session becomes unresponsive. A typical scenario: when viewing the address *0xf0030000*, the IDE automatically preloads data from adjacent addresses and touches an unmapped region (such as *0xf002xxxx*), causing the target to return an access error.

Root cause

The **Automatic rendering mode** of the Memory window preloads data outside the visible area to optimize the scrolling experience. When the preloaded addresses fall into:

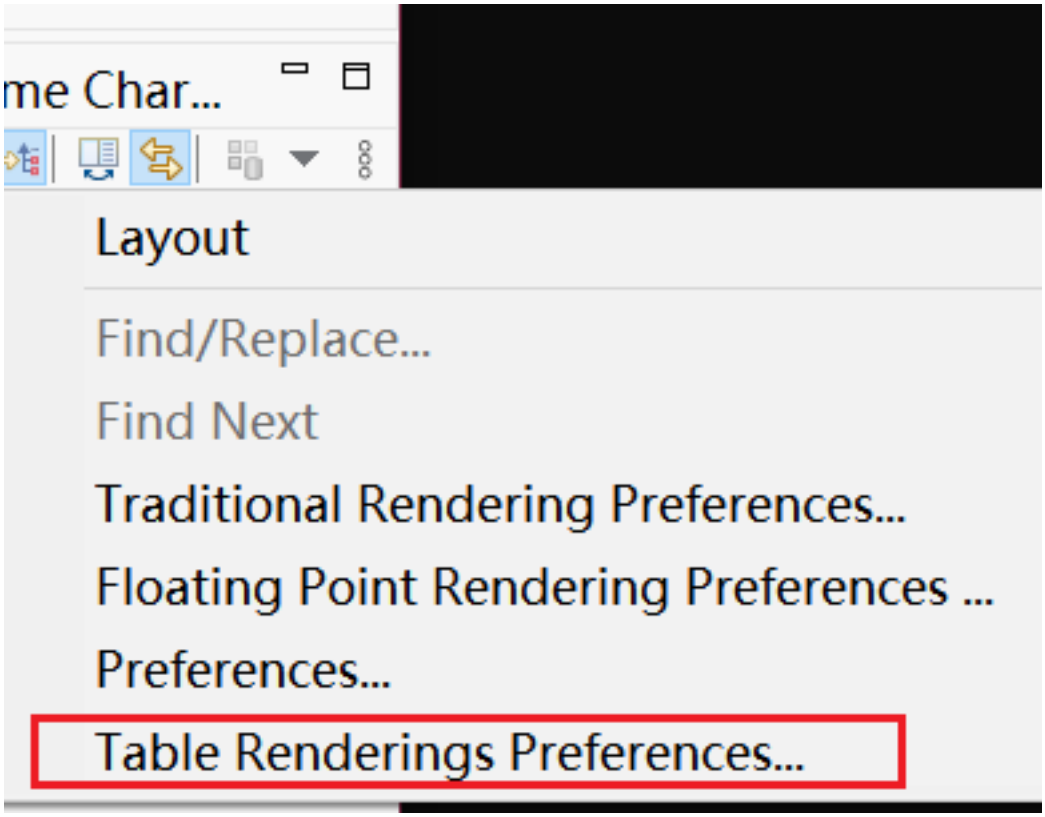
- unmapped memory regions
- reserved regions protected by the MPU
- peripheral address holes

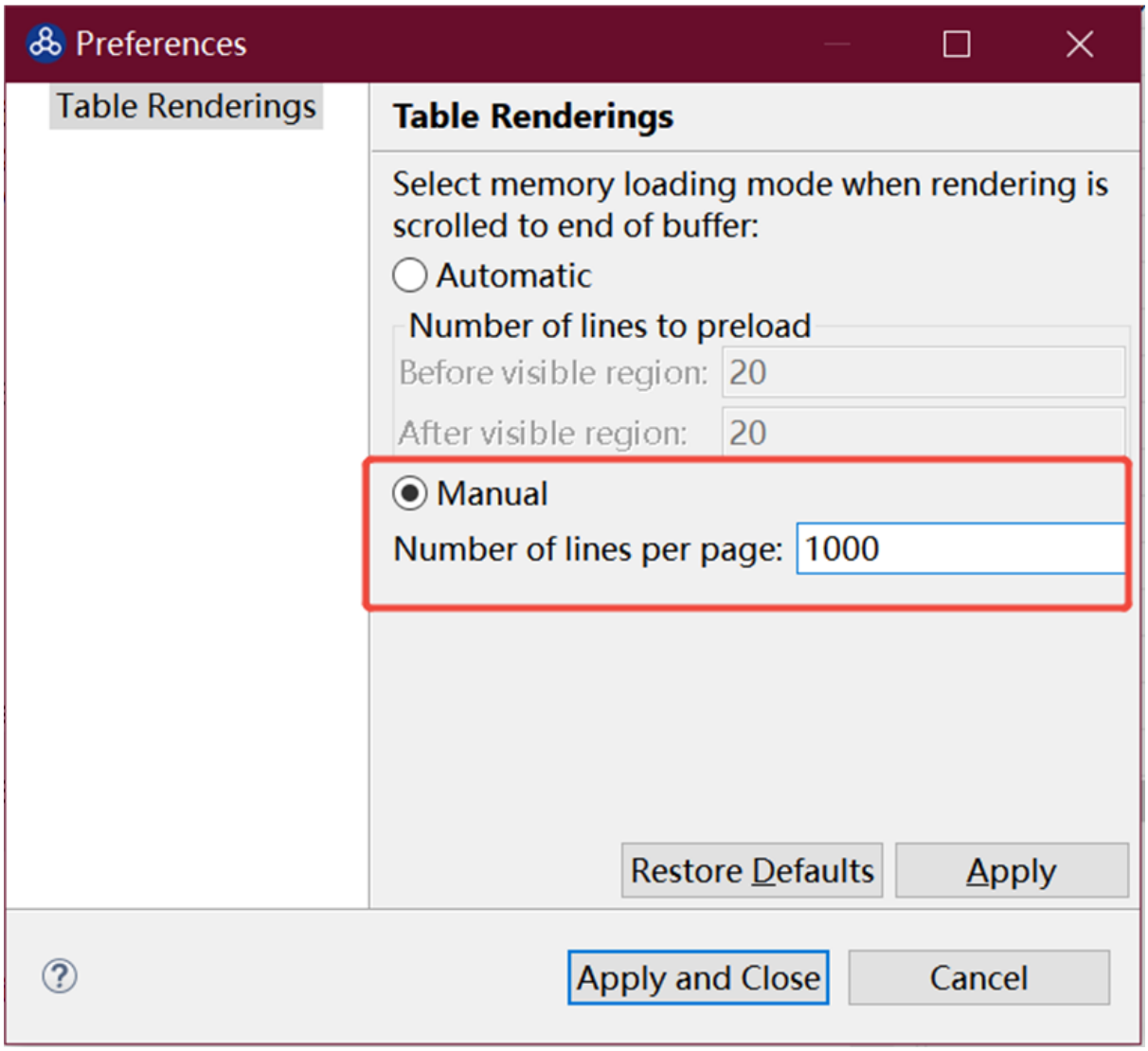
the target hardware returns a Bus Fault or access error, and GDB enters a waiting/error state, which appears as the IDE freezing.

Solution

Recommended: switch to Manual mode

1. Right-click in the Memory window → **Table Renderings Preferences**···
2. Select **Manual** for Memory loading mode
3. Click OK

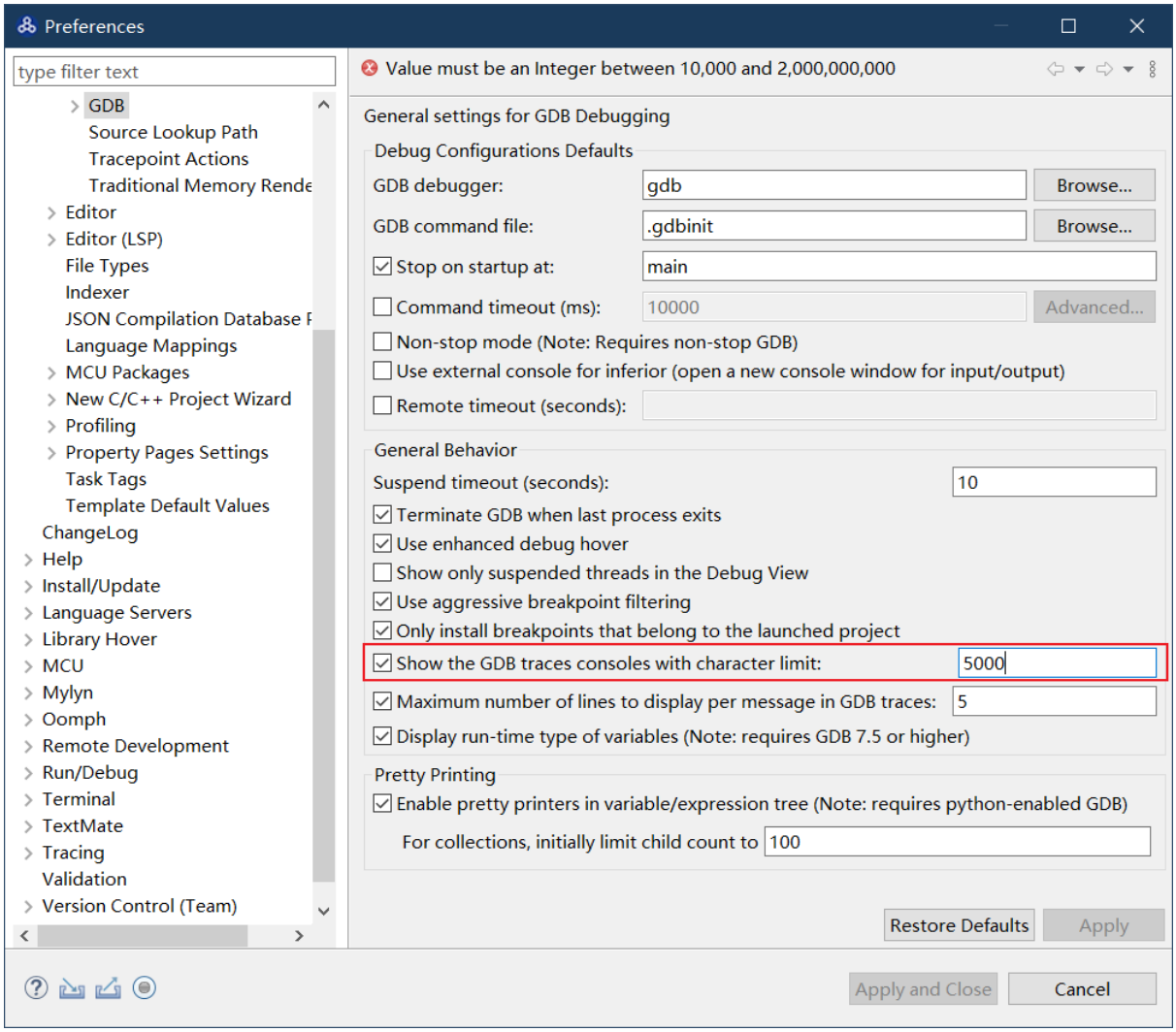

me Char...
Layout
Find/Replace...
Find Next
Traditional Rendering Preferences...
Floating Point Rendering Preferences ...
Preferences...
Table Renderings Preferences...



2.12.35 How to View GDB Trace

When debugging, you may need to check the raw commands exchanged between the IDE and GDB to locate debugger problems. Enable the GDB traces console as follows:

1. Open `Window -> Preferences -> C/C++ -> Debug -> GDB`, and check `Show the GDB traces consoles with character limit`. The character limit (5000 by default) and the maximum number of lines to display per message can be adjusted as needed, then click `Apply and Close`.



2. Run or debug the project. A `<projectname> ... gdb traces` console is added to the Console view, which shows the GDB commands and responses exchanged during the debug session.



2.13 Other Issues Without a Noted Version

If there are any omissions in this document, refer to https://www.rvmcu.com/nucleistudio-faq.html. Common problems encountered in later versions are listed there.

2.14 Known Issues

This page records known defects and issues that are being tracked, grouped by release version.

2.14.1 Known Issues in 2026.10

Generate gprof Tool Has No Response

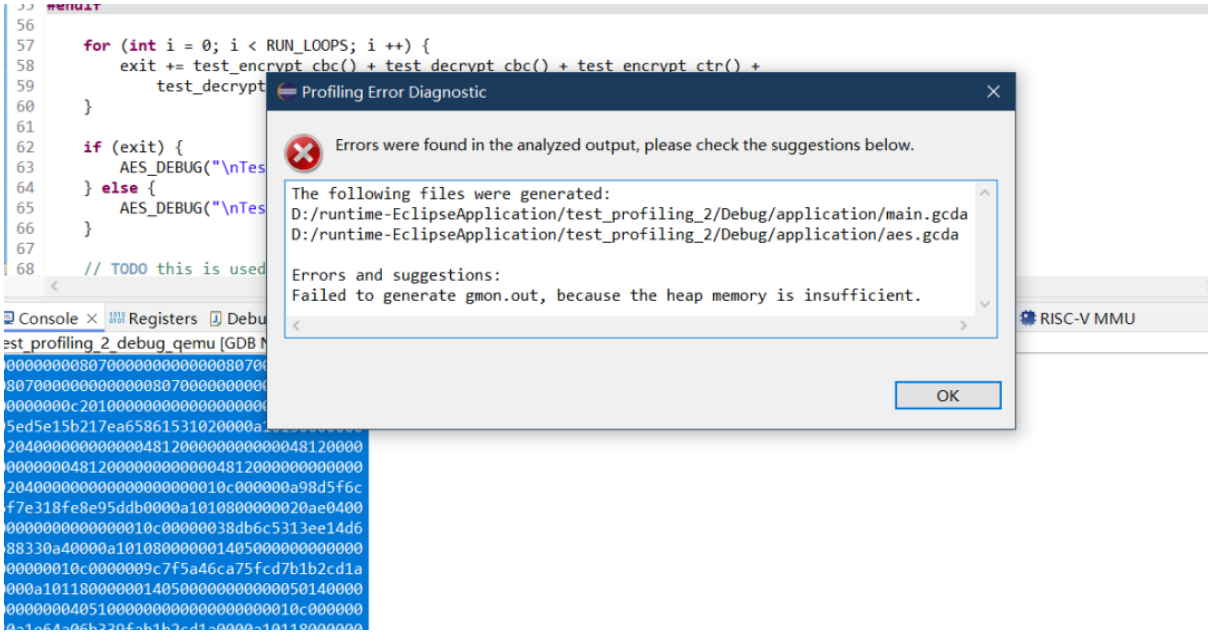
When the **Generate gprof** tool appears to produce no response, the usual cause is insufficient heap memory on the target: `-pg` (gprof) and `-coverage` (gcov) both allocate memory at runtime, so enabling them together may exhaust the heap. In this case the target only prints errors such as `monstartup: out of memory` or `gprof_collect: unable to malloc enough memory to store gprof data`, and `gmon.out` is never generated.

Therefore, test gprof and gcov separately: build and run with `-pg` alone for gprof, and with `-coverage` alone for gcov, instead of enabling both at the same time.

Nuclei Studio 2026.10 also improves the error feedback of profiling. Errors found in the analyzed output are captured and shown in a **Profiling Error Diagnostic** dialog together with suggestions. The dialog is titled `Profiling Error Diagnostic`, and its body begins with `Errors were found in the analyzed output, please check the suggestions below.` (or `...no files have been generated.` when no output file was produced).

The diagnostic recognizes the following errors (each is a concise, de-duplicated message):

- **Heap memory insufficient (gmon.out)** —`Failed to generate gmon.out, because the heap memory is insufficient.` Matched from `monstartup: out of memory` or `gprof_collect: unable to malloc enough memory to store gprof data`.
- **Heap memory insufficient (.gcda file)** —`Failed to generate <file>, because the heap memory is insufficient.` Matched from `Can't allocate gcda buffer for <file>`.
- **Call-arc table overflow** —`gmon.out may be incomplete, because the call-arc table overflowed (tos overflow); some call relations were lost.` Matched from `mcount: tos overflow` or `_mcleanup: tos overflow`. This message is shown only when no heap error is present; when a heap error occurs, the overflow is treated as a cascade of the heap error and is suppressed.
- **File open failure** —`Failed to generate <file>, because the file could not be opened on the target (check semihosting / file system).` Matched from `Unable to open <file>`. If the same file already reports a heap error, this message is suppressed to avoid contradictory prompts for that file.

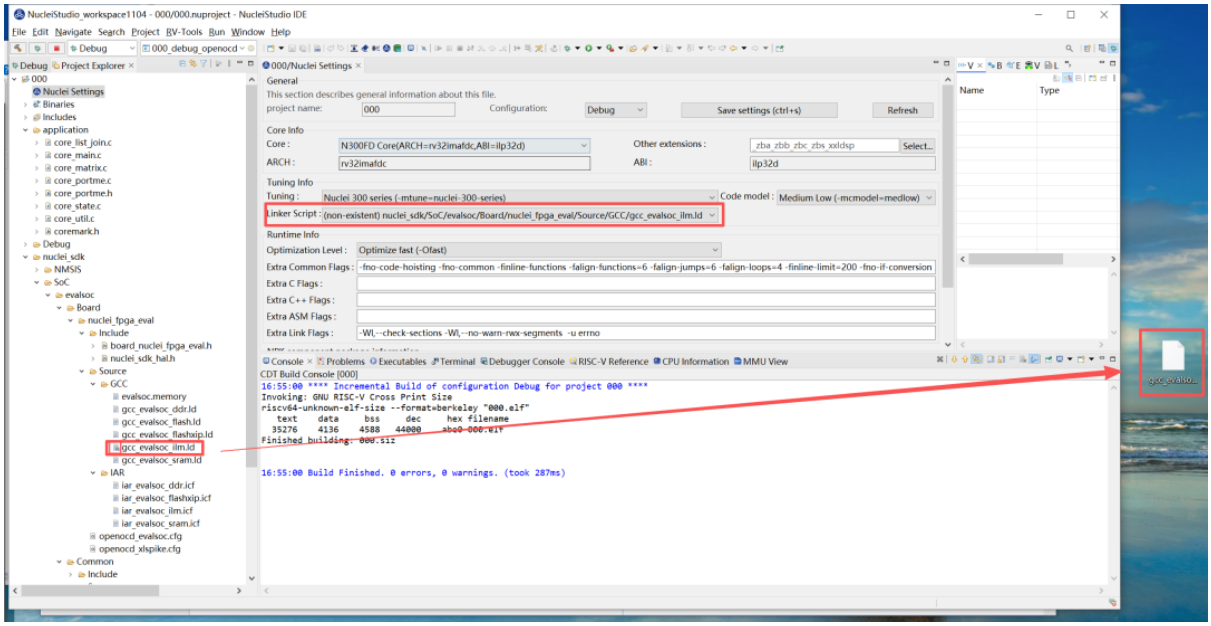


Linker Script Selected via File System Displayed as Non-existent

In Nuclei Studio 2026.10, an issue was fixed where a linker script (`.ld`) selected through **File System** was incorrectly displayed as `non-existent` in Nuclei Settings.

The root cause is that Nuclei Settings resolves the selected `.ld` file as a regular file. If the selected file is a linked file, or the path contains linked folders, the file cannot be located and is therefore reported as `non-existent`. The same applies to virtual files and folders.

As shown in the figure below, the icon in Project Explorer indicates that `gcc_evalsoc_ilm.ld` is a linked file, yet the Linker Script field still reports it as non-existent.



CHAPTER

THREE

NUCLEI STUDIO VSCODE

3.1 Introduction to Nuclei Studio VSCode

Nuclei Studio VSCode is a Visual Studio Code extension developed by Nuclei for developing, building, and debugging embedded applications on Nuclei RISC-V processors. It works on top of the *Nuclei Studio IDE* (page 3): the build tools, RISC-V toolchains, OpenOCD, and Nuclei Model shipped with the IDE are reused directly, and the paths of these tools are derived automatically from the IDE installation folder once the `nucleiStudio.installFolder` setting is configured.

Projects created by Nuclei Studio VSCode are Eclipse CDT compatible (`.project` / `.cproject`), so they can be opened in Nuclei Studio IDE, and projects created in Nuclei Studio IDE can be imported into Nuclei Studio VSCode.

Note: Nuclei Studio VSCode uses its own version number; the current version is **0.3.0**. The tools it drives come from the Nuclei Studio release (such as **2026.10**), whose version covers the whole package, including the Nuclei Studio IDE, the RISC-V toolchain, OpenOCD, QEMU, and Nuclei Model.

3.1.1 Features

Project Management

- Create projects from NPK template projects or Hello World C/C++ templates
- Import and export projects
- Workspace and project management: switch workspace, set the active project, rename, delete, and refresh projects
- Project content view with full file operations, per-file build settings, and build configurations (Debug/Release)

Build & Debug

- Build, clean, and rebuild projects with one click, using the toolchain and build tools shipped with Nuclei Studio IDE
- Debug on hardware or simulators via OpenOCD, J-Link, QEMU, or Nuclei Model, with graphical editors for `.launch` files
- Live Watch view for real-time variable monitoring, Live Plot, and CSV logging during debug

SDK & Package Management

- Manage SDKs and install NPK component packages (SDK/SSP/BSP/APP and more) with the NPK Package Management page
- Open a terminal with the paths of all the Nuclei tools set up automatically

Code Intelligence

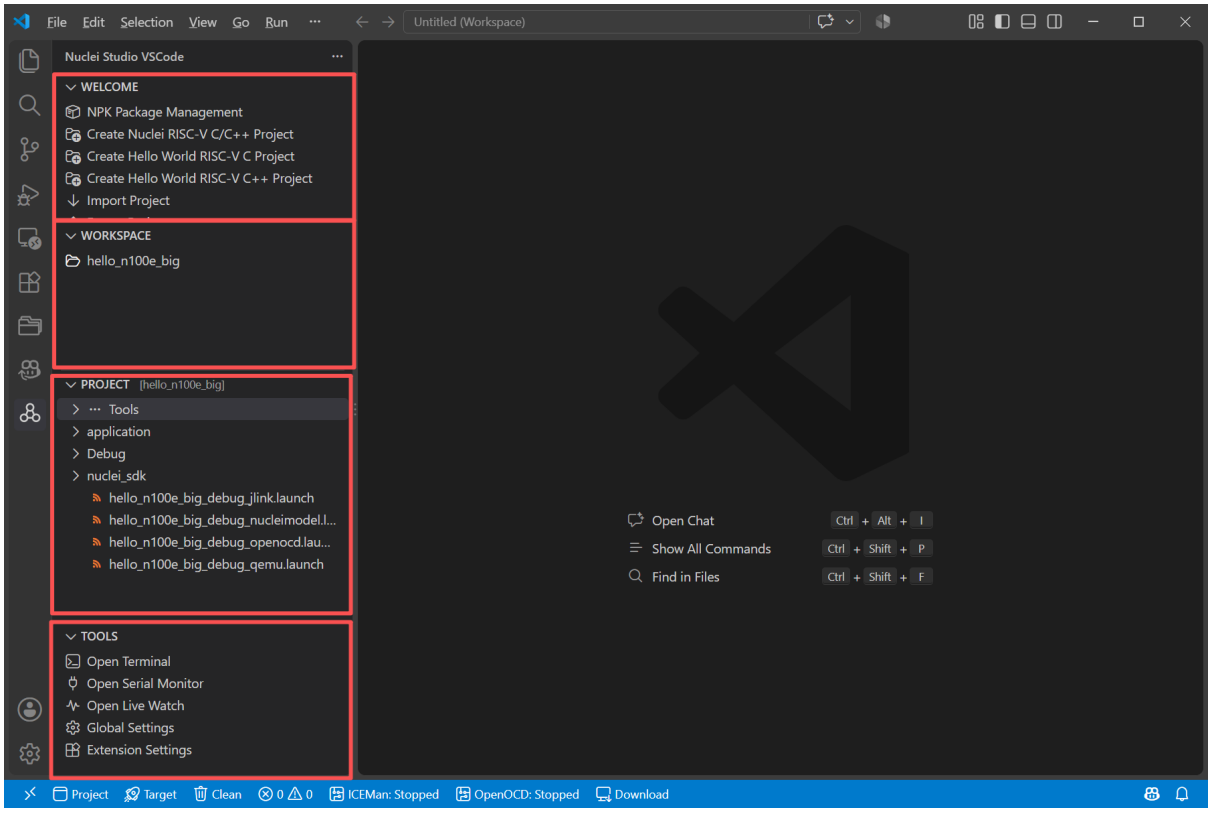
- IntelliSense support based on the project build model, through the Microsoft C/C++ extension

3.1.2 The Nuclei Studio VSCode View Container

The extension provides a **Nuclei Studio VSCode** icon in the VS Code activity bar. Clicking it opens a view container with 4 panels:

- **WELCOME**: the entry for new users and a collection of frequently used operations, including `Configure Nuclei Studio Install Folder` (only shown when the install folder is not configured), `NPK Package Management`, `Create Nuclei RISC-V C/C++ Project`, `Create Hello World RISC-V C / C++ Project`, `Import Project` / `Export Project`, and `Switch Workspace`.
- **WORKSPACE**: lists all projects in the current Nuclei Studio workspace. Click a project to make it the active project; build and debug always target the active project. Title bar buttons provide new, import, export, refresh, and delete project actions, and a Rebuild Index action for IntelliSense.
- **PROJECT**: shows the file content of the active project as a tree, with full file operations (including `Ctrl+C` / `Ctrl+X` / `Ctrl+V`, `Delete`, `F2`, `Ctrl+Z` / `Ctrl+Y` shortcuts) and a `Tools` toolbar providing build, clean, rebuild, run, debug, Nuclei Settings, and project properties actions.
- **TOOLS**: entries to commonly used tools: `Open Terminal`, `Open Serial Monitor`, `Open Live Watch`, `Global Settings`, and `Extension Settings`.

In addition, a separate **Live Watch** view container in the activity bar hosts the Live Watch view used during debug sessions; see *Live Watch* (page 418).



3.1.3 Requirements

- Visual Studio Code[24].
- Nuclei Studio IDE[25] installed; Nuclei Studio VSCode works on top of it. See *Nuclei Studio VSCode Download and Installation* (page 405).
- The following companion extensions, which are installed automatically from the VS Code Marketplace as dependencies of this extension:
  - Serial Monitor (`ms-vscode.vscode-serial-monitor`)
  - CDT GDB Debug Adapter Extension (`eclipse-cdt.cdt-gdb-vscode`)
  - Memory Inspector (`eclipse-cdt.memory-inspector`)
  - Peripheral Inspector (`eclipse-cdt.peripheral-inspector`)
- Optional: the Microsoft C/C++ extension (`ms-vscode.cpptools`) for IntelliSense support; see *IntelliSense* (page 422).

3.1.4 Extension Settings

The extension contributes the following settings (under the **MCU** category in the VS Code Settings page). Most of them are derived automatically from the Nuclei Studio IDE installation folder; see *Configure the Nuclei Studio IDE Installation Folder* (page 408).

- `nucleiStudio.installFolder`: Nuclei Studio IDE installation folder
- `nucleiStudio.buildToolsFolder`: build tools folder
- `nucleiStudio.riscvToolchain.*.path`: paths of the various RISC-V toolchains (GCC Newlib, Big Endian GCC/Newlib, Nuclei GCC, Terapines ZCC, Clang Newlib, GCC Linux, GCC RTEMS, and custom toolchains)
- `nucleiStudio.openocd_path` / `nucleiStudio.openocd_executable`: OpenOCD location
- `nucleiStudio.jlink_path` / `nucleiStudio.jlink_gdbserver`: SEGGER J-Link location
- `nucleiStudio.nucleimodel_path` / `nucleiStudio.nucleimodel_executable`: Nuclei Model location
- `nucleiStudio.npk.globalPath` / `nucleiStudio.npk.userPath`: NPK package installation paths; see *NPK Package Management in Nuclei Studio VSCode* (page 409)
- `nucleiStudio.liveWatch.*`: Live Watch sampling interval, variable/plot limits, and CSV auto-save options; see *Live Watch Settings* (page 420)

3.2 Changelog

3.2.1 Version 2026.10

Note:

- The release version of Nuclei Studio VSCode follows the Nuclei Tools release, so this release is **2026.10**.
- The Nuclei Studio VSCode **extension** has its own version number, which is **0.3.0** in the 2026.10 release.

[24] https://code.visualstudio.com/
[25] https://www.nucleisys.com/download.php

This is the first official release of Nuclei Studio VSCode, a Visual Studio Code extension for developing, building, and debugging embedded applications on Nuclei RISC-V processors. The extension itself is versioned as **0.3.0**, and is released together with **Nuclei Studio 2026.10**, the version of the whole Nuclei Studio package, which integrates the Nuclei Studio IDE, Nuclei RISC-V Toolchain 2026.10, OpenOCD 2026.10, QEMU 2026.10, and Nuclei Near Cycle Model 2026.10.

Nuclei Studio VSCode works on top of the Nuclei Studio IDE and reuses its toolchains, OpenOCD, Nuclei Model, and NPK packages, and the projects it creates are Eclipse CDT compatible and interchangeable with Nuclei Studio IDE.

The main features of this release are as follows.

Project Management

- New project wizard based on NPK packages: select the SoC/board from the installed SDK packages, choose an application example and the toolchain, and create a ready-to-use project
- Hello World RISC-V C/C++ project templates, usable without installing NPK packages
- Import and export projects; projects created by Nuclei Studio IDE can be imported directly
- Workspace management: switch workspaces, set the active project, rename, delete, and refresh projects
- Project content view with full file operations, keyboard shortcuts, Locate Active File, and Rebuild Index

Project Build

- Build, clean, and rebuild projects with the toolchains and build tools shipped with Nuclei Studio IDE, in a CDT compatible managed-build way
- Support for the RISC-V Big Endian GCC/Newlib toolchain, plus an Endian option (`-mbig-endian` / `-mlittle-endian`) in the Target Processor settings
- Debug/Release build configurations, with Set Active, Manage, Build All, Clean All, and Build Selected actions
- Build or clean selected source files, and exclude files or folders from the build per configuration
- Build output in the Nuclei Build channel with an error/warning summary

Project Debug and Run

- Debug schemes based on OpenOCD, J-Link, QEMU, and Nuclei Model, connected through the CDT GDB Debug Adapter
- Graphical editors for OpenOCD, J-Link, QEMU, and Nuclei Model `.launch` files, with Main, Debugger, Startup, and SVD Path tabs
- Multiple ways to start a session: Debug Project (`F5`), Debug Project (Select Server), and Debug As / Run As on `.launch` files; a debug configuration is generated automatically when no `launch.json` exists
- Integration with the Memory Inspector, Peripheral Inspector, and Serial Monitor companion extensions
- Live Watch for real-time variable monitoring without halting the target, with Live Plot curves and CSV export (requires Nuclei OpenOCD with the SBA feature)

NPK Package Management

- Browse, download, import, and remove NPK component packages, with dependency resolution and IDE compatibility checks
- User/Global installation paths, derived automatically from the Nuclei Studio IDE installation folder

Project Configuration

- Nuclei Settings page for quick project configuration: core, RISC-V ISA extensions, tuning, code model, download mode, optimization level, and extra flags
- RISC-V ISA Extension Selector with dependency and conflict handling and `-march`/`-mabi`/`-mtune` preview
- Project Properties and Resource Properties pages for advanced project settings

Ease of Use

- Get Started walkthrough guiding through setup, package installation, project creation, build, and debug
- Tool paths derived automatically from the Nuclei Studio IDE installation folder
- IntelliSense support through the Microsoft C/C++ extension
- Nuclei Studio Terminal with the paths of all Nuclei tools set up automatically

3.3 Nuclei Studio VSCode Download and Installation

The overall setup flow is: install Nuclei Studio IDE -> install the extension in VS Code -> configure the IDE installation folder -> install NPK packages -> create, build, and debug a project.

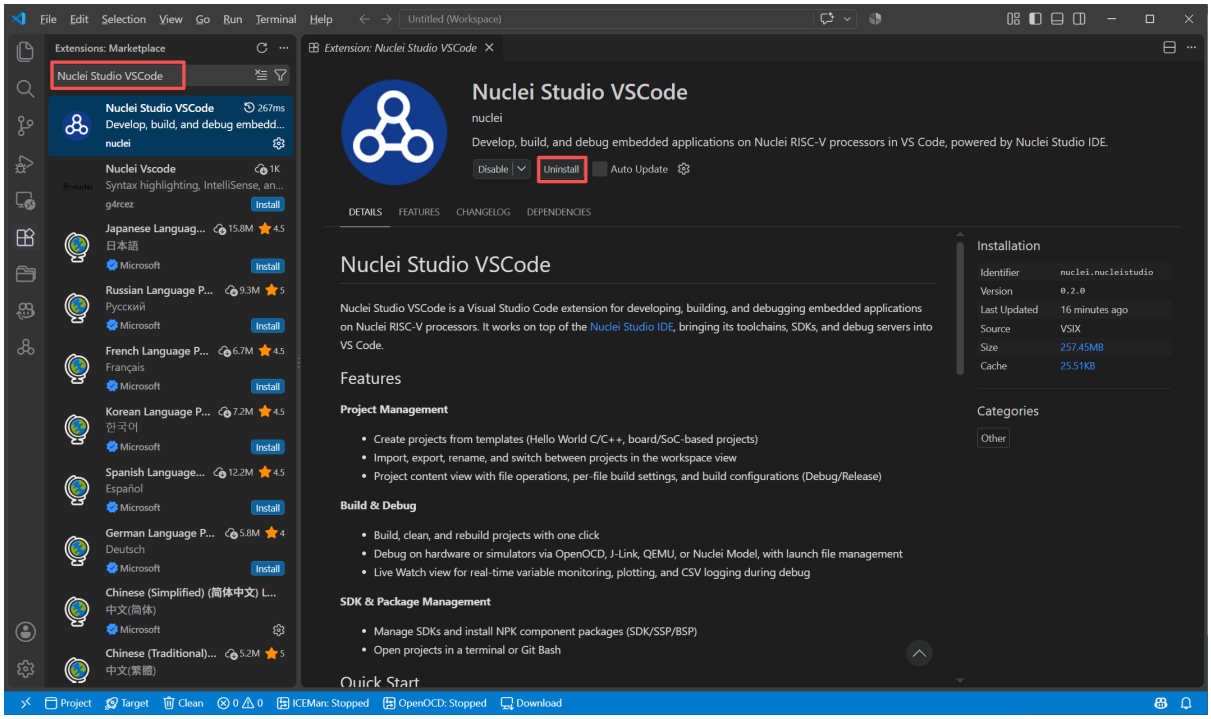
3.3.1 Install Nuclei Studio IDE

Nuclei Studio VSCode works on top of Nuclei Studio IDE: the build tools, OpenOCD, RISC-V toolchains, and Nuclei Model are all used from the IDE installation folder. Download and install Nuclei Studio IDE from the Nuclei website[26] first. For the download and installation of Nuclei Studio IDE, see *Nuclei Studio IDE* (page 3).

3.3.2 Install the Extension in VS Code

Open VS Code, open the Extensions view (`Ctrl+Shift+X`), search for `Nuclei Studio VSCode`, and click `Install`.

[26] https://www.nucleisys.com/download.php



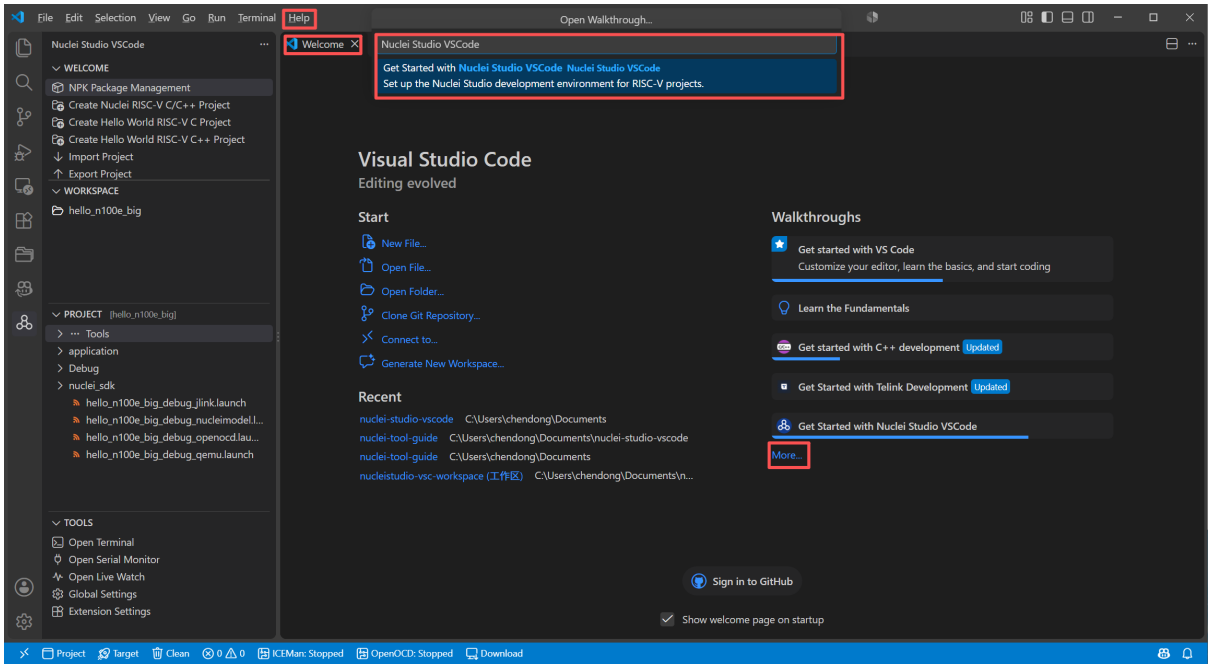
Note: If you have a local `nucleistudio.vsix` package of the extension (for example, a test version), you can also install it manually: select `Install from VSIX...` in the menu of the Extensions view and choose the package file.

Note: The extension depends on the following companion extensions, which are installed automatically from the VS Code Marketplace: Serial Monitor (`ms-vscode.vscode-serial-monitor`), CDT GDB Debug Adapter Extension (`eclipse-cdt.cdt-gdb-vscode`), Memory Inspector (`eclipse-cdt.memory-inspector`), and Peripheral Inspector (`eclipse-cdt.peripheral-inspector`). If the Nuclei Studio VSCode icon does not appear in the activity bar after installation, check whether any of these extensions was installed before but is disabled; enable them and reload the window (`Ctrl+Shift+P` -> `Developer: Reload Window`).

Note: If VS Code reports that the current workspace is in Restricted Mode, click the `Manage` button in the notification and choose `Trust` for the workspace; otherwise the extension features will not work properly.

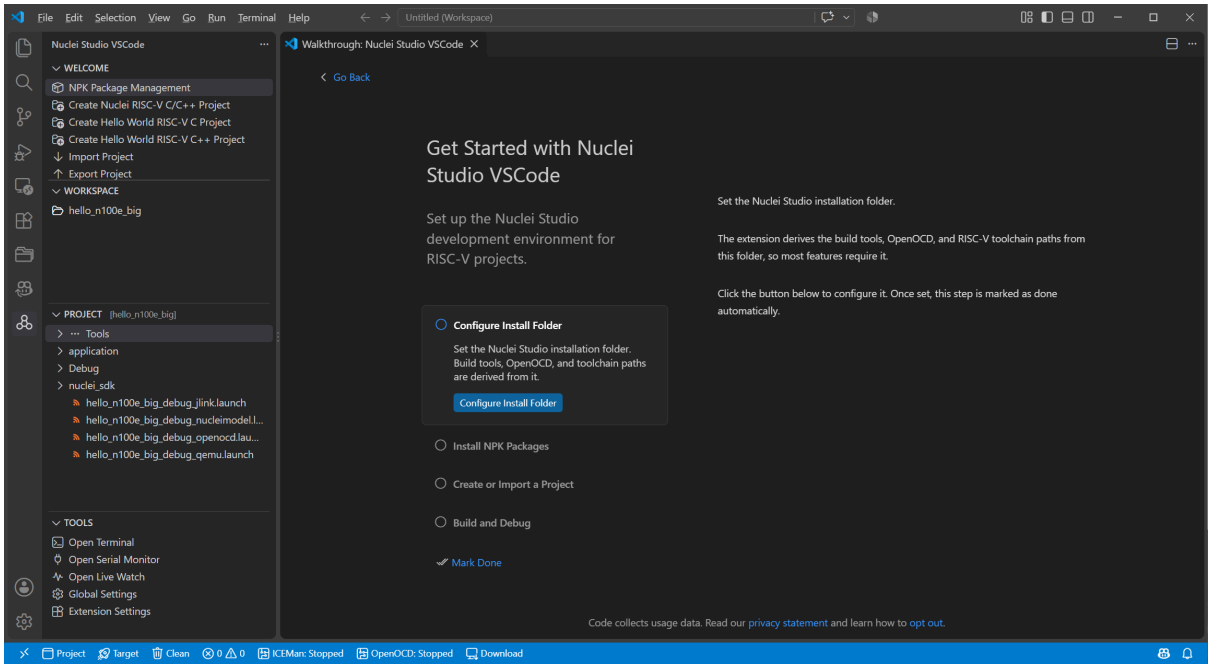
3.3.3 The Getting Started Walkthrough

After the extension is installed, click the Nuclei Studio VSCode icon in the activity bar. On the Welcome page, click `More...` under Walkthroughs, type `nuclei` in the input box at the top, and select **Get Started with Nuclei Studio VSCode** to open the getting started guide. The Welcome page can also be opened via `Help -> Welcome` in the toolbar; if no Walkthroughs content is shown, press `Ctrl+Shift+P` and run `Welcome: Open Walkthrough...`.



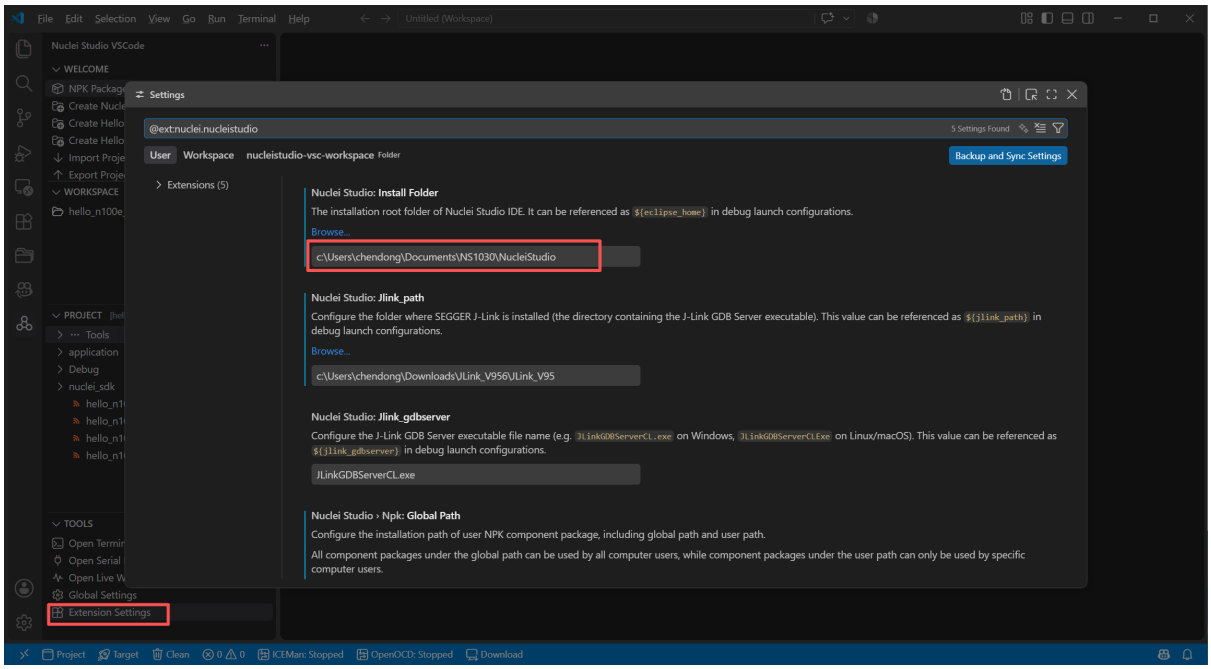
The walkthrough contains 4 steps, which correspond to the chapters of this documentation as follows:

Walkthrough step	Related chapter
Configure Install Folder	*Configure the Nuclei Studio IDE Installation Folder* (page 408)
Install NPK Packages	*NPK Package Management in Nuclei Studio VSCode* (page 409)
Create or Import a Project	*Create a Project in Nuclei Studio VSCode* (page 410)
Build and Debug	*Build a Project in Nuclei Studio VSCode* (page 413) and *Debug and Run a Project in Nuclei Studio VSCode* (page 415)



3.3.4 Configure the Nuclei Studio IDE Installation Folder

Most features of the extension depend on the Nuclei Studio IDE installation folder. When it is not configured, a warning item `Configure Nuclei Studio Install Folder` is shown at the top of the WELCOME panel. Click it and select the folder where Nuclei Studio IDE is installed, or set `nucleiStudio.installFolder` in the VS Code Settings page.



Once the installation folder is configured, the following settings are derived and written into the user `settings.json` automatically:

- Build tools: `<NucleiStudio>/toolchain/build-tools/bin` (Windows only)
- OpenOCD: `<NucleiStudio>/toolchain/openocd/bin`
- Nuclei Model: `<NucleiStudio>/toolchain/nucleimodel/bin`
- RISC-V GCC/Newlib and Clang toolchains: `<NucleiStudio>/toolchain/gcc/bin`
- RISC-V Big Endian GCC/Newlib toolchain: `<NucleiStudio>/toolchain/gccbe/bin`
- RISC-V Nuclei GCC toolchain: `<NucleiStudio>/toolchain/gcc10/bin`
- Terapines ZCC toolchain: `<NucleiStudio>/toolchain/zcc/bin`
- NPK package paths: the global path defaults to `<NucleiStudio>/Packages`, and the user path defaults to the user home directory

If any derived path does not exist on disk, a warning message lists the paths that could not be resolved; check that the selected folder is a complete Nuclei Studio IDE installation.

The installation folder can be referenced as `${eclipse_home}` in debug launch configurations, which keeps `.launch` files interchangeable with Nuclei Studio IDE.

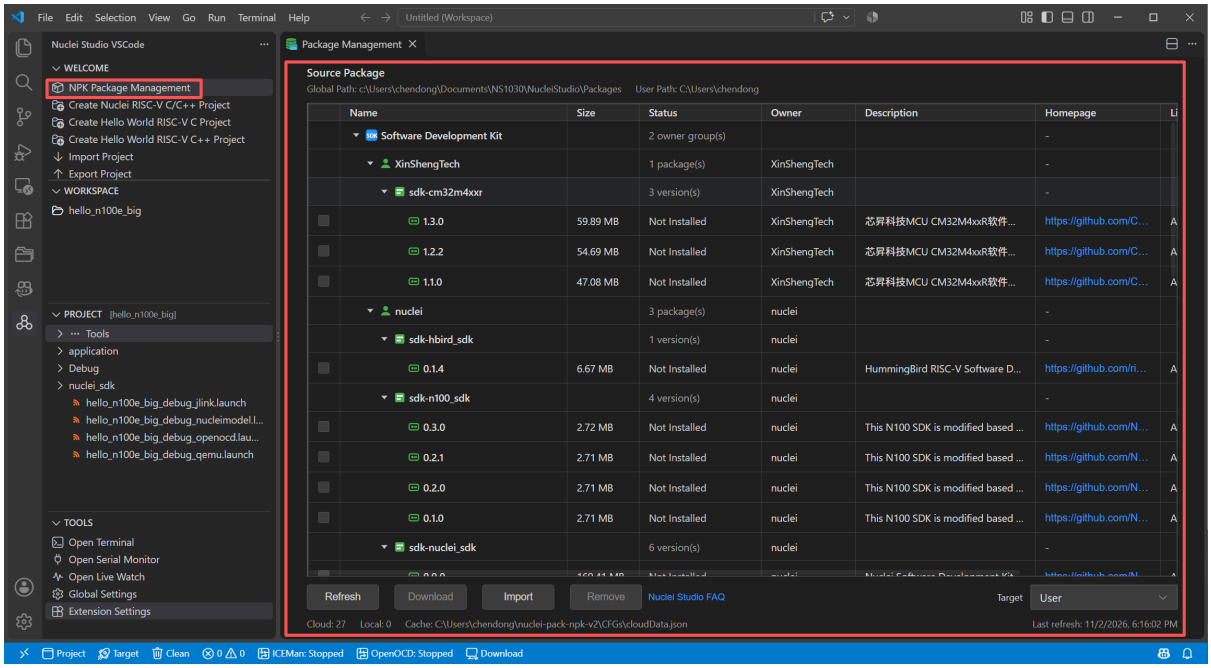
3.4 NPK Package Management in Nuclei Studio VSCode

Nuclei Studio VSCode uses the same Nuclei Package (NPK) mechanism as Nuclei Studio IDE: the SDK / SSP / BSP / APP components a project depends on are distributed as NPK packages, and packages installed by either tool are shared. For the NPK concept and package structure, see *Nuclei Studio NPK Overview* (page 30); for package development, see *Nuclei Studio NPK Creation and Sharing* (page 63). This chapter only describes how to manage NPK packages in Nuclei Studio VSCode.

Note: Install the required NPK packages before creating a project; otherwise no board or SDK will be selectable in the project wizard, and the wizard shows "Please install the required NPK packages first." with an `Open NPK Package Management` button.

3.4.1 Opening the NPK Package Management Page

Click `NPK Package Management` in the WELCOME panel to open the Package Management page.



The page shows the NPK packages available from the cloud and installed locally, organized as a tree of **Type -> Owner -> Name -> Version**, with columns for Name, Size, Status, Owner, Description, Homepage, License, and IDE Version. The supported package types include SDK (Software Development Kit), SSP (SoC Support Package), CSP (Core Support Package), BSP (Board Support Package), APP (Application Package), OSP (RTOS Support Package), MWP (Middle-ware Package), BDP (Bundle Package), TPP (Template Package), and Tool Package.

The status of a package version can be `Not Installed`, `Installed (User)`, `Installed (Global)`, or `Installed (Directory)`:

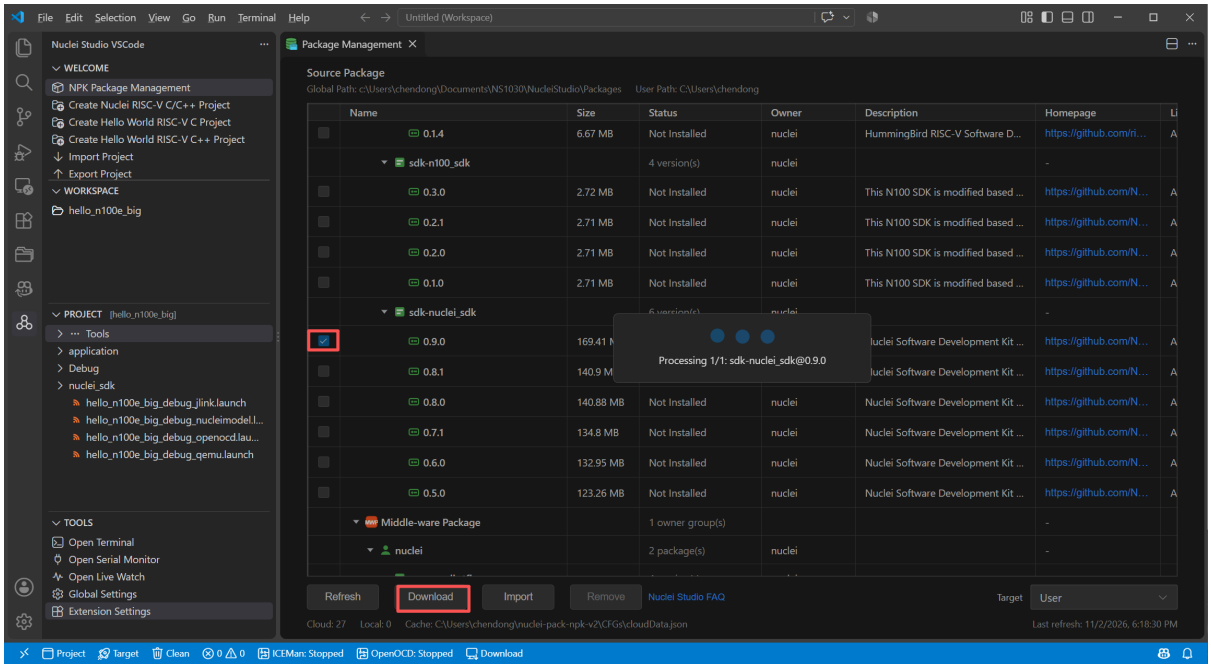
- Packages installed to the **user path** can only be used by the current computer user; the user path defaults to the user home directory and can be changed with `nucleiStudio.npk.userPath`.
- Packages installed to the **global path** can be used by all computer users; the global path defaults to the `Packages` folder of the Nuclei Studio IDE installation and can be changed with `nucleiStudio.npk.globalPath`.

The installation paths are derived automatically when the Nuclei Studio IDE installation folder is configured; see *Configure the Nuclei Studio IDE Installation Folder* (page 408).

3.4.2 Managing Packages

The buttons at the bottom of the page provide the following actions:

- **Refresh**: refresh the package list from the cloud. The list is also refreshed automatically the first time the page is opened when no local package data exists.
- **Download**: check one not-installed package version and click `Download` to download and install it. Dependent packages are resolved and downloaded together, and a warning is shown if the package is not marked as compatible with the current IDE version. Use the `Target` selector next to the buttons to choose whether the package is installed to the `User` or `Global` path.
- **Import**: import a local NPK package from a ZIP file or a folder. A folder imported this way is referenced in place (`Installed (Directory)`) instead of being copied.
- **Remove**: remove the selected installed packages. Packages referenced from a directory are only unregistered; their files are not deleted.



Creating projects from the installed NPK packages is described in *Create a Project in Nuclei Studio VSCode* (page 410).

3.5 Create a Project in Nuclei Studio VSCode

This chapter describes how to create and manage projects in Nuclei Studio VSCode.

3.5.1 Workspaces and Projects

In Nuclei Studio VSCode, a **workspace** is a plain container directory that holds project folders. The first time the extension runs, `~/Documents/nucleistudio-vsc-workspace` is used as the default workspace. Click `Switch Workspace` in the WELCOME panel to switch between workspaces: the QuickPick shows the current workspace, the recently used workspaces, `Browse...` for choosing an existing directory, and `New Workspace...` for creating a new one. Switching the workspace also re-points the VS Code workspace folder to the Nuclei Studio workspace directory.

Note: If the folder opened in VS Code differs from the Nuclei Studio workspace, the WORKSPACE view shows a `Switch to Nuclei Studio VSCode workspace` entry; click it to switch.

Each workspace has exactly one **active project** at a time. Click a project in the WORKSPACE view (or right-click it and choose `Set Active`) to make it the active project; its name is shown in the title of the PROJECT view. Build and debug always target the active project.

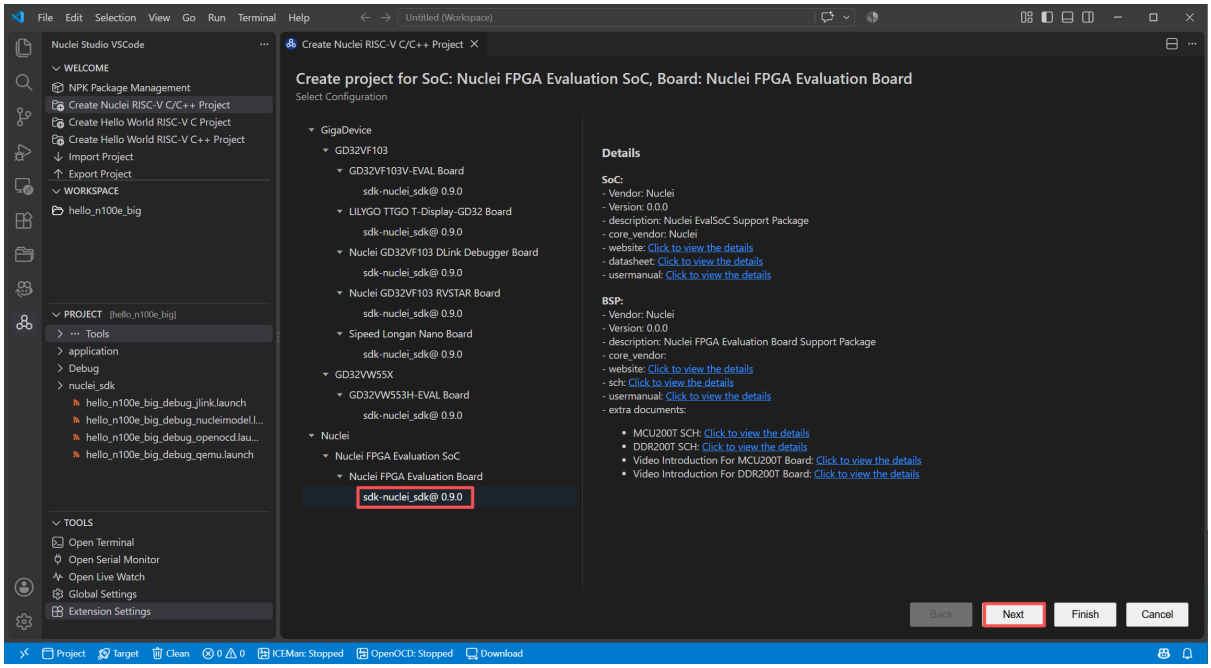
A project created by Nuclei Studio VSCode is Eclipse CDT compatible: the project folder contains `.project` and `.cproject` files, a `<project>.nuproject` file recording the NPK packages it depends on, and `.launch` debug configuration files, so the project can also be opened in Nuclei Studio IDE.

3.5.2 Create a Project with the New Project Wizard

This is the recommended way to create a project. It requires the corresponding NPK packages to be installed first; see *NPK Package Management in Nuclei Studio VSCode* (page 409).

Click `Create Nuclei RISC-V C/C++ Project` in the WELCOME panel to open the wizard.

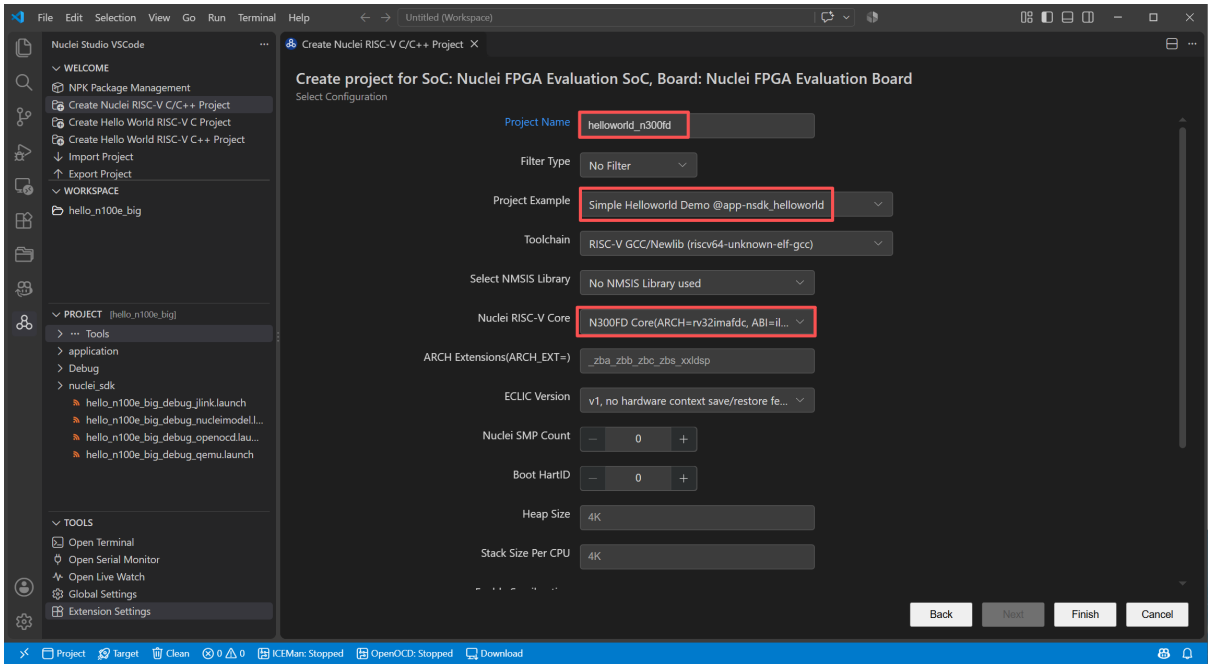
On the first page, choose the development board: the left pane shows the installed packages organized as **Vendor -> SoC (SSP) -> Board (BSP) -> package name and version**, and the right pane shows the details of the selected SoC or board, such as the vendor, version, description, and links to the datasheet and user manual.



On the second page, configure the project:

- **Project Name**: the name of the project; it is validated against the projects already in the workspace.
- **Filter Type** and **Project Example**: filter and select an application example from the installed NPK packages that are compatible with the selected board, such as the Coremark Benchmark.
- **Version**: shown when multiple versions of the example exist.
- **Toolchain**: select the toolchain used to build the project, such as RISC-V GCC/Newlib or RISC-V Clang/Newlib.
- The remaining configuration items are provided by the selected NPK package and vary by SDK.

Click `Finish` to create the project. The project is created in the current workspace and set as the active project, and its main source file is opened automatically.



3.5.3 Create a Hello World Project

Click `Create Hello World RISC-V C Project` or `Create Hello World RISC-V C++ Project` in the WELCOME panel to create a minimal example project without installing NPK packages. The wizard has 3 pages:

1. **Basic properties**: project name, author, copyright notice, the greeting printed by the program, and the source folder name.
2. **Configurations**: the build configurations to create (`Debug` and `Release`); more configurations can be added later.
3. **Toolchain**: the toolchain used to build the project; the path is filled in automatically from the extension settings.

The generated project contains a `main.c` (or `main.cpp`) that prints the greeting.

Note: The generated Hello World project is a build-only example: it is linked without a board support package, so do not run it on a physical board directly.

3.5.4 Import a Project

Click `Import Project` in the WELCOME panel (or the import button in the WORKSPACE view title bar) to import an existing project into the current workspace. Two methods are provided:

- **Import Project Folder**: import a project folder in place. The folder (or one of its subfolders) must contain the `.project` and `.cproject` files; projects created by Nuclei Studio IDE can be imported this way.
- **Import ZIP Package**: extract a packaged project into the current workspace; a confirmation is shown before existing files are overwritten.

The imported project is set as the active project.

3.5.5 Export a Project

Click `Export Project` in the WELCOME panel (or the export button in the WORKSPACE view) to copy the files of the selected projects to a target directory. In the Export dialog, check the projects and files to export, choose the target directory, and set the options as needed (overwrite existing files without warning, create directory structure for files, create only selected directories, resolve and export linked resources).

3.5.6 Manage Projects

The WORKSPACE view and its context menus provide the common project management actions:

- **Set Active**: make the selected project the active project.
- **Rename**: rename a project. Only the project name is changed; the project folder on disk keeps its original name.
- **Delete**: delete a project. You can choose `Remove from List Only` to keep the project files on disk, or `Delete Permanently (Including Files)`.
- **Refresh Projects**: refresh the project list; projects whose folders no longer exist are removed from the list.
- **Convert GCC 10 Project to GCC 13**: convert a project created with the GCC 10 toolchain to the GCC 13 toolchain; see *Advanced Usage of Nuclei Studio VSCode* (page 418).

3.6 Build a Project in Nuclei Studio VSCode

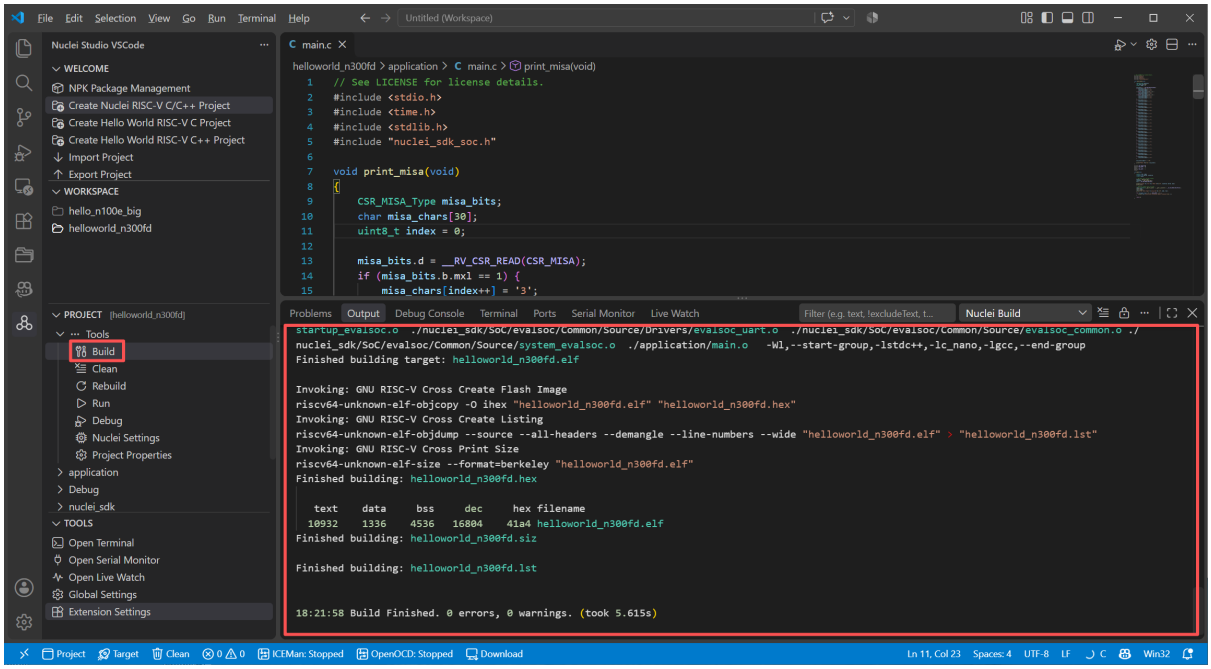
Nuclei Studio VSCode builds projects with the toolchain and build tools shipped with Nuclei Studio IDE. The build system is compatible with Eclipse CDT managed build: makefiles are generated from the project's `.cproject` file into the build configuration directory (such as `Debug/`), and `make` from the IDE's build tools is invoked to build the project. Pre-build and post-build steps configured in the project are supported, and parallel build is honored according to the project settings.

Build output is shown in the **Nuclei Build** channel of the VS Code Output panel, ending with a summary such as `Build Finished. 0 errors, 0 warnings.`.

3.6.1 Build, Clean, and Rebuild

Make sure the intended project is the active project (see *Create a Project in Nuclei Studio VSCode* (page 410)), then use one of the following entries:

- The `Tools` toolbar at the top of the PROJECT view: `Build`, `Clean`, `Rebuild`, and `Debug`.
- The context menu of a project in the WORKSPACE or PROJECT view.



3.6.2 Build Configurations

Each project has `Debug` and `Release` build configurations. The active configuration is stored per project. Use `Build Configurations` in the project context menu to manage them:

- `Set Active`: switch the active build configuration between `Debug` and `Release`.
- `Manage...`: manage the build configurations.
- `Build All` / `Clean All` / `Build Selected...`: batch build or clean operations over the build configurations.

3.6.3 Build or Clean Selected Files

In the PROJECT view, right-click one or more source files and choose `Build Selected File(s)` or `Clean Selected File(s)` to compile only the selected files, or delete their generated object files. This is useful for a quick syntax check of a single file.

3.6.4 Exclude Resources from the Build

In the PROJECT view, right-click a file or folder and choose `Resource Configurations` -> `Exclude from Build...` to exclude it from the selected build configurations. Excluded items are shown greyed out with a slashed-circle icon. Choose `Resource Configurations` -> `Reset to Default...` to remove the exclusions and restore the defaults.

3.6.5 Toolchains

The toolchain used by a project is selected when the project is created, and can be changed later in the Nuclei Settings page or the Project Properties page (see *Advanced Usage of Nuclei Studio VSCode* (page 418)). The toolchain paths are derived automatically from the Nuclei Studio IDE installation folder (see *Configure the Nuclei Studio IDE Installation Folder* (page 408)):

- **RISC-V GCC/Newlib (riscv64-unknown-elf-gcc)** and **RISC-V Clang/Newlib (riscv64-unknown-elf-clang)**: `<NucleiStudio>/toolchain/gcc/bin`
- **RISC-V Big Endian GCC/Newlib (riscv64be-unknown-elf-gcc)**: `<NucleiStudio>/toolchain/gccbe/bin`
- **RISC-V Nuclei GCC (riscv-nuclei-elf-gcc)**: `<NucleiStudio>/toolchain/gcc10/bin`
- **Terapines ZCC**: `<NucleiStudio>/toolchain/zcc/bin`

If a build fails because the toolchain, build tools, or `make` cannot be found, check the `nucleiStudio.installFolder` setting and the derived paths; see *Nuclei Studio VSCode FAQ* (page 423).

3.7 Debug and Run a Project in Nuclei Studio VSCode

Nuclei Studio VSCode debugs projects through the CDT GDB Debug Adapter (the `gdbtarget` debugger type provided by the companion extension `eclipse-cdt.cdt-gdb-vscode`), connecting to the target through the GDB server of each debug scheme.

The debug configurations of a project are stored as Eclipse `.launch` files in the project folder (created by the project wizard from the NPK package), which are the source of truth for debugging. The `launch.json` file is regenerated automatically from them when a debug or run session starts. The project is always built before a session starts; if the build fails, the session is not started.

When a debug session starts, the Memory Inspector and the Peripheral Inspector are opened automatically, and the SVD file of the project is used for the peripheral register view.

3.7.1 Debug Schemes

The following debug schemes are supported:

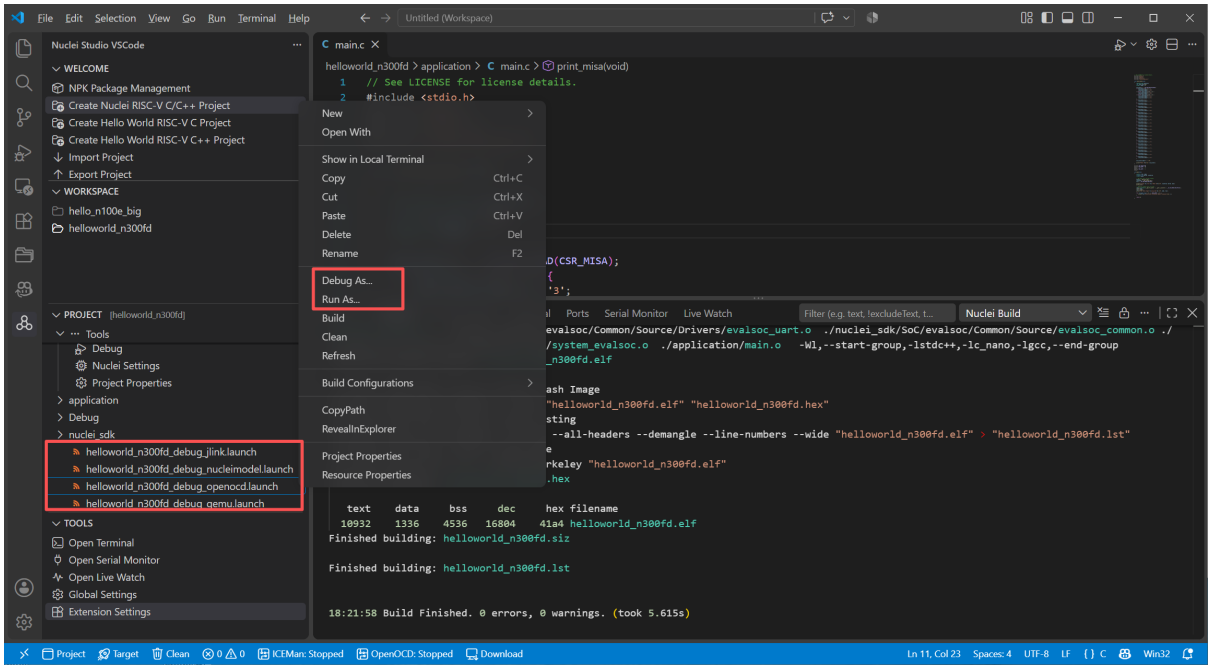
Debug scheme	Default GDB port	Default server executable	Notes
OpenOCD	`localhost:3333`	`openocd`	Default telnet port 4444, Tcl port 6666; required by *Live Watch* (page 418)
J-Link	`localhost:2331`	`JLinkGDBServerCL`	Default SWO port 2332, telnet port 2333
QEMU	`localhost:1234`	`qemu-system-riscv32` / `qemu-system-riscv64`	Runs the program on the Nuclei EvalSoC simulation machine
Nuclei Model	`localhost:1234`	`xl_cpumodel`	Runs the program on the Nuclei Near Cycle Model; see *Nuclei Model* (page 863)

The locations of the servers are configured by the `nucleiStudio.openocd_*`, `nucleiStudio.jlink_*`, and `nucleiStudio.nucleimodel_*` settings, which are derived automatically from the Nuclei Studio IDE installation folder; see *Configure the Nuclei Studio IDE Installation Folder* (page 408).

3.7.2 Start Debugging

Make sure the intended project is the active project, then start debugging in one of the following ways:

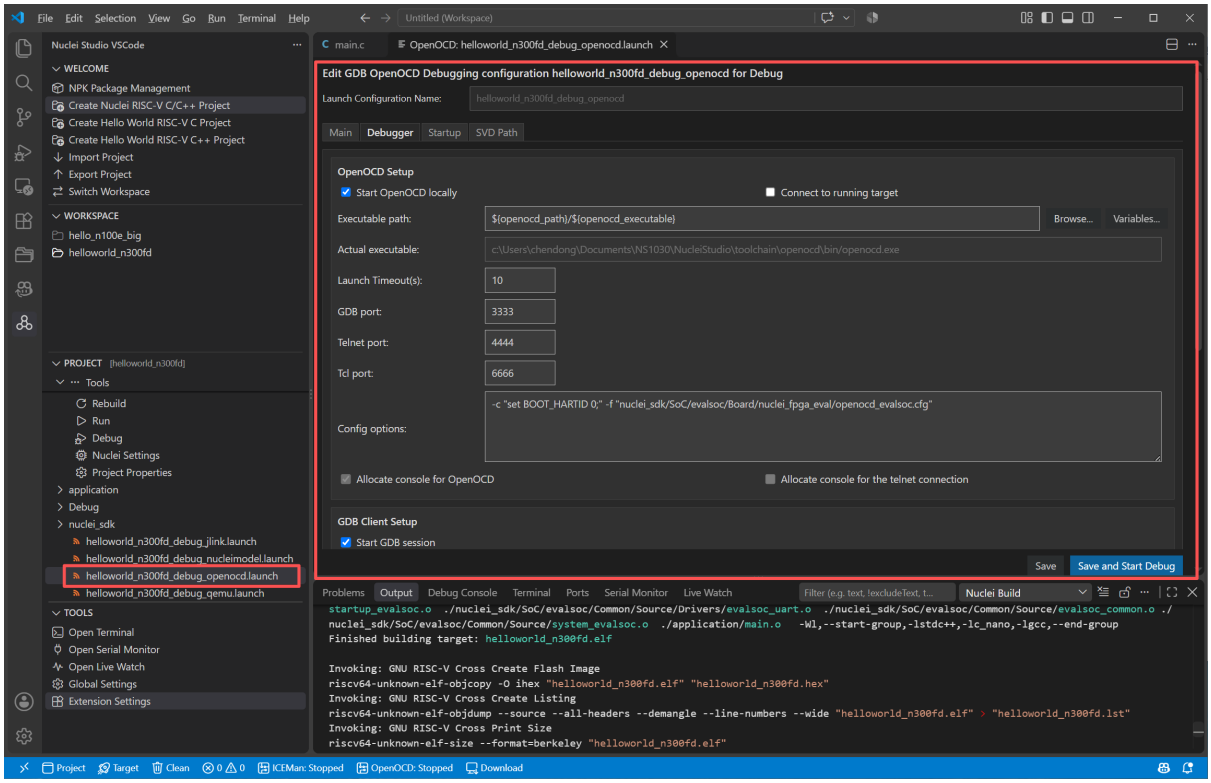
- **Debug Project** (command `Nuclei Studio: Debug Project`, or press `F5`): build the active project and start debugging with the debug scheme detected from its `.launch` files (OpenOCD by default). If there is no `launch.json`, a debug configuration is generated automatically from the project's `.launch` files.
- **Debug Project (Select Server)**: select a debug server (OpenOCD, J-Link, QEMU, and so on) manually; `.launch` files that already exist in the project are marked with a check.
- **Debug / Run** in the project context menu: open the OpenOCD launch configuration editor of the project in debug or run mode.
- **Debug As⋯ / Run As⋯** in the context menu of a `.launch` file in the PROJECT view: open the corresponding graphical launch configuration editor. Only OpenOCD, J-Link, QEMU, and Nuclei Model `.launch` files are supported.



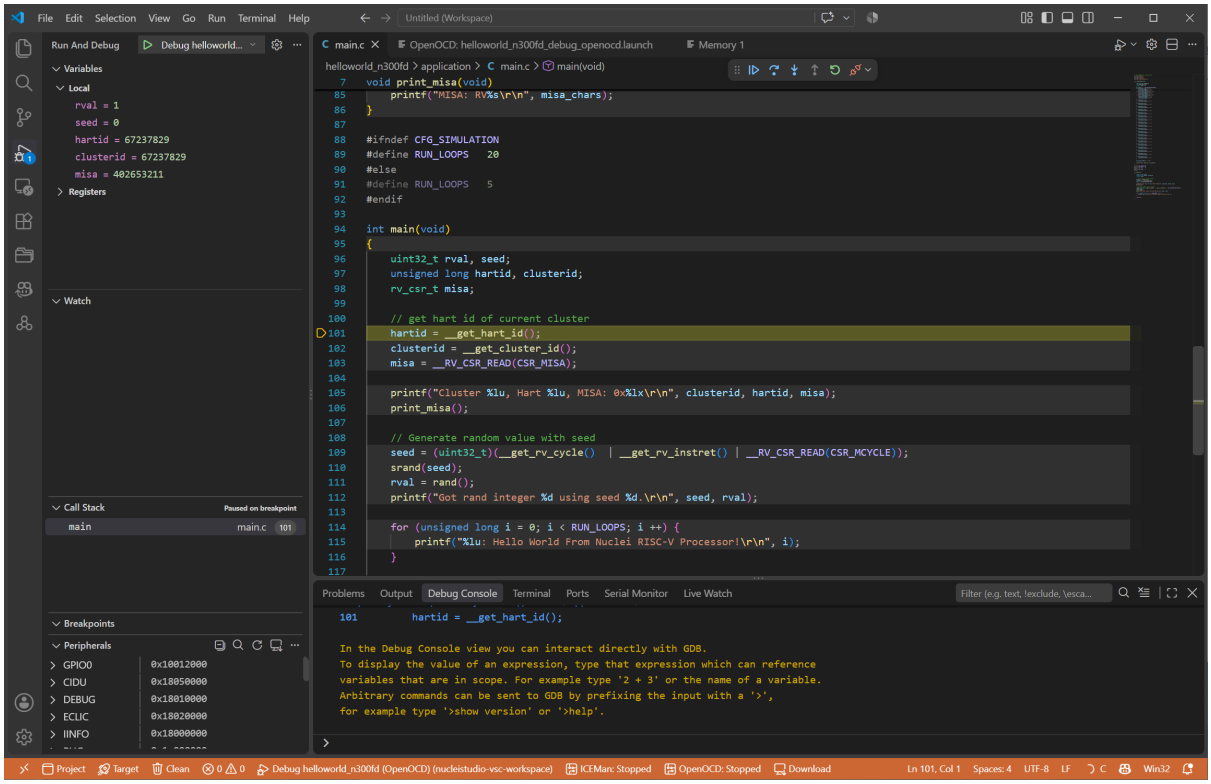
3.7.3 Launch Configuration Editors

Graphical editors are provided for the OpenOCD, J-Link, QEMU, and Nuclei Model debug schemes. Opening a `.launch` file with `Debug As...` / `Run As...`, or using the `Debug` / `Run` actions of the project, opens the corresponding editor. The editors share the same layout, with **Main**, **Debugger**, **Startup**, and **SVD Path** tabs, and **Save**, **Save and Start Debug** (or **Save and Start Run**) buttons at the bottom. Changes are saved back into the `.launch` file.

- **Main**: the project and the application (ELF) to debug, and the server startup options, such as starting the server locally or connecting to a running target, the executable path, ports, and additional server options. For QEMU and Nuclei Model, the simulation options are configured here, such as the board/machine name, the RISC-V core, the download mode (`ddr`/`flash`/`flashxip`/`ilm`/`sram`), SMP count, vector length, and extra extensions.
- **Debugger**: GDB client setup (executable, options, commands) and remote target settings (host and port).
- **Startup**: reset and load behavior, such as initial reset, load symbols and executable, debug in RAM, set breakpoint at `main`, and continue.
- **SVD Path**: the SVD file used by the Peripheral Inspector.



Take OpenOCD as an example: connect the target board, open the project's `<project>_debug_openocd.launch` file, check the configuration, and click `Save and Start Debug` to start the debug session.



3.7.4 Run Without Debugging

The same schemes can also run the program without stopping at breakpoints: use `Run` in the project context menu or `Run As...` on a `.launch` file, which starts the session in run mode.

3.7.5 Serial Monitor and Debug Views

- **Serial Monitor**: click `Open Serial Monitor` in the TOOLS panel to open the Serial Monitor view provided by the companion extension `ms-vscode.vscode-serial-monitor` for UART output of the board.
- **Memory Inspector** and **Peripheral Inspector**: opened automatically when a debug session starts, provided by the companion extensions `eclipse-cdt.memory-inspector` and `eclipse-cdt.peripheral-inspector`.

3.7.6 Limitations

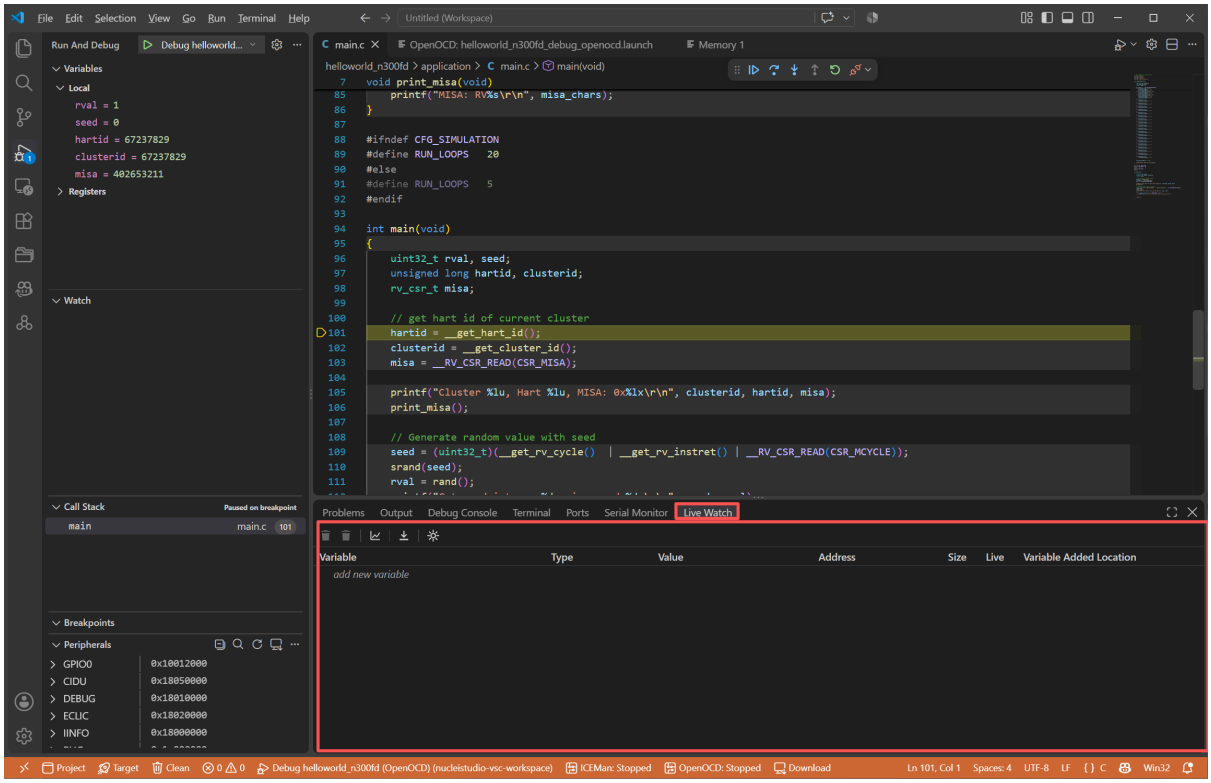
- QEMU ARM/AArch64 and Custom `.launch` files can be used with `F5` / Debug Project, but no graphical editor is provided for them.
- DLink debugging is only available when the project ' s NPK package provides a DLink `.launch` file; there is no dedicated DLink editor.

3.8 Advanced Usage of Nuclei Studio VSCode

3.8.1 Live Watch

Live Watch is a real-time monitoring tool: while the target is running at full speed (without halting), it reads variable values from the target memory periodically and displays them in the Live Watch view, with Live Plot real-time curves and CSV data export. For the Live Watch feature in Nuclei Studio IDE, see *Using Live Watch* (page 306).

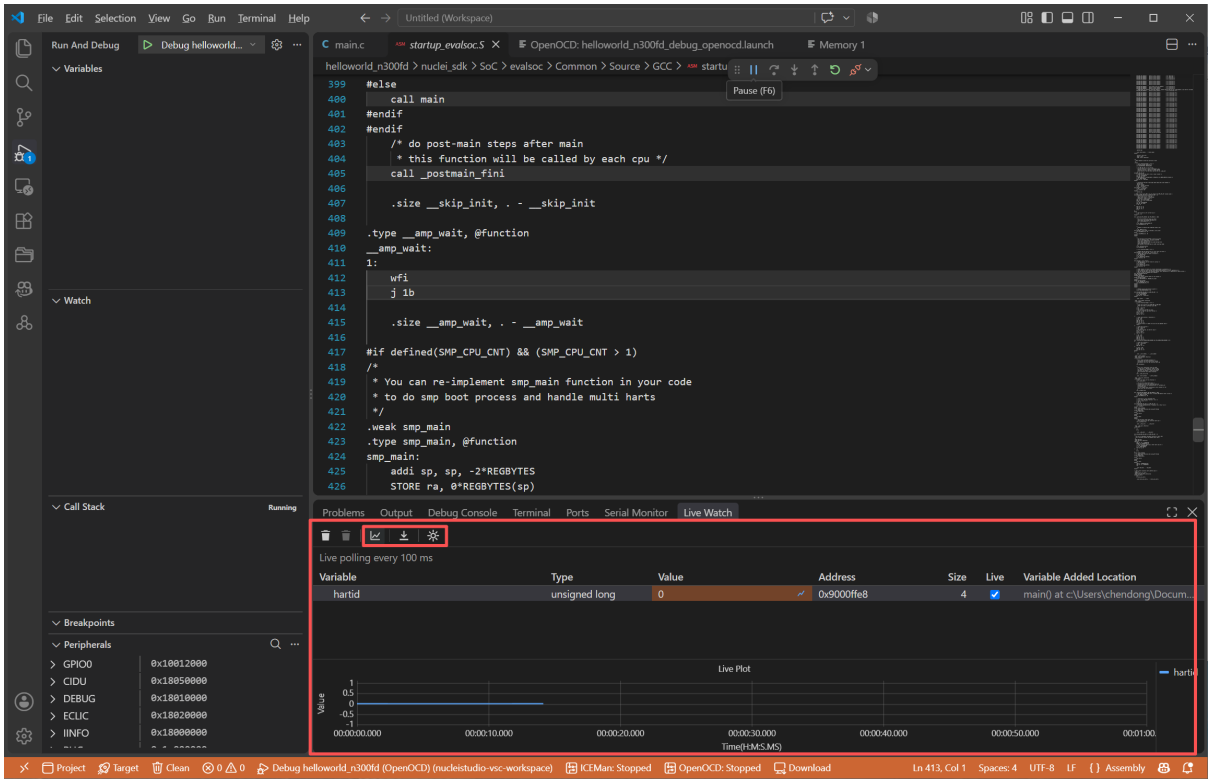
Live Watch requires a debug session based on **Nuclei OpenOCD with the SBA (System Bus Access) feature**, and the CPU must support SBA as well. While the target is running, values are polled through the OpenOCD telnet port (default 4444); while the target is halted, values are read through the GDB channel.



Using Live Watch

Open the Live Watch view from the Live Watch container in the activity bar, or click `Open Live Watch` in the TOOLS panel.

- **Add variables**: variables can only be added or edited while the target is halted. Each variable shows its type, value, address, size, and the location where it was added.
- **Modify values**: values can be written to the target while it is halted.
- **Number format**: switch the display format of a variable (Default, Hex, Decimal, Octal, Binary) from its context menu.
- **Live Plot**: toggle `Live Plot` on a variable to plot its value over time in the built-in chart. Right-click the chart to suspend or continue plotting.
- **Save as CSV**: save the recorded values of a variable (or all variables) to a CSV file from the context menu.



Live Watch Settings

Click the settings button in the Live Watch view to open the Live Watch Settings dialog, or edit the corresponding settings directly:

- `nucleiStudio.liveWatch.speedMs`: sampling interval in milliseconds; the default is 100 and the minimum is 10.
- `nucleiStudio.liveWatch.variableLimit`: maximum number of variables watched live at the same time (0-10); the default is 10.
- `nucleiStudio.liveWatch.plotLimit`: maximum number of variables plotted in Live Plot at the same time (0-10); the default is 10.
- `nucleiStudio.liveWatch.saveDataPath`: directory for automatically saved Live Watch CSV data; empty means the project ' s `Debug` folder.
- `nucleiStudio.liveWatch.saveDataSpeedMin`: interval in minutes for automatically saving Live Watch CSV data; the default is 10.

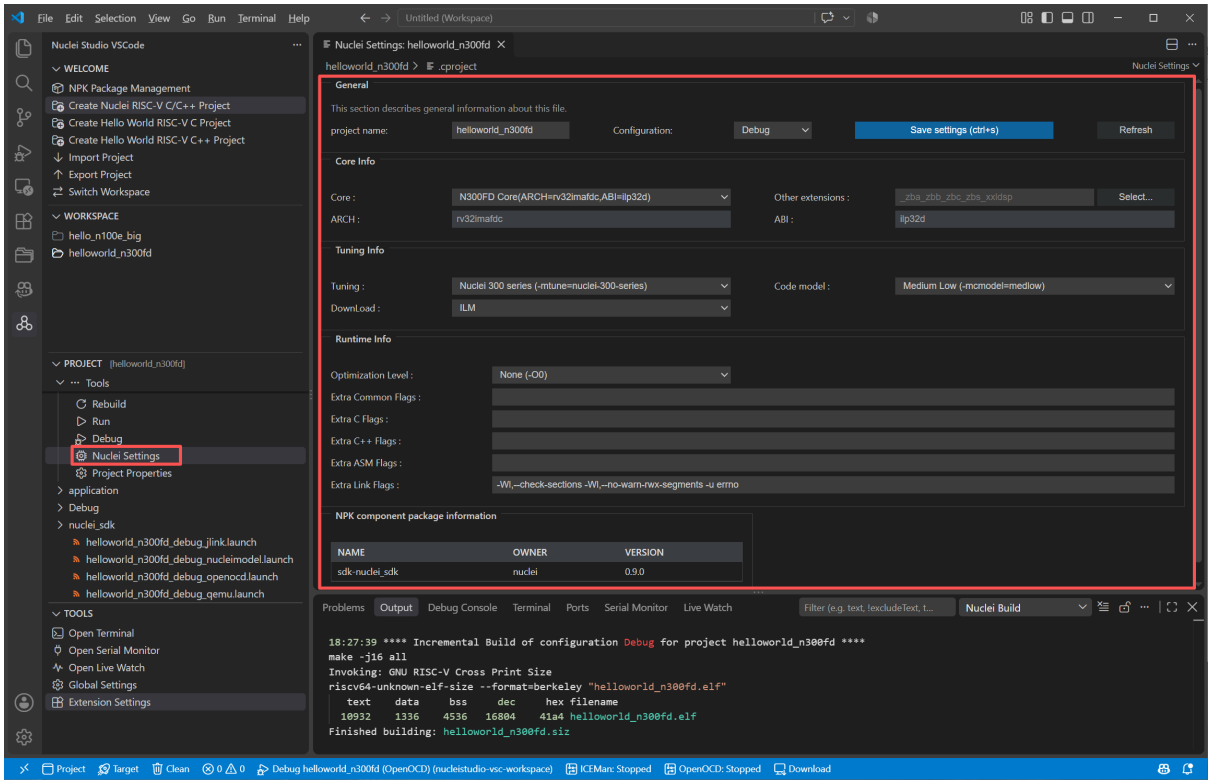
3.8.2 Nuclei Settings

Nuclei Settings is the quick configuration page of a project. Right-click a project and choose `Nuclei Settings` (or open the project ' s `.cproject` file with the Nuclei Settings editor) to open it. It provides the following sections:

- **General**: the project name, and the build configuration (`Debug` / `Release`) the page applies to.
- **Core Info**: the Nuclei processor core of the project; the ARCH and ABI are derived automatically. Use `Other extensions` or the `Select...` button to configure RISC-V ISA extensions; see *RISC-V ISA Extension Selector* (page 421).
- **Tuning Info**: the tuning (`-mtune`) option, the code model, and the download mode / linker script of the project.
- **Runtime Info**: the optimization level (`-O0` ⋯`-Oz`) and extra flags for the C, C++, ASM, and link steps.

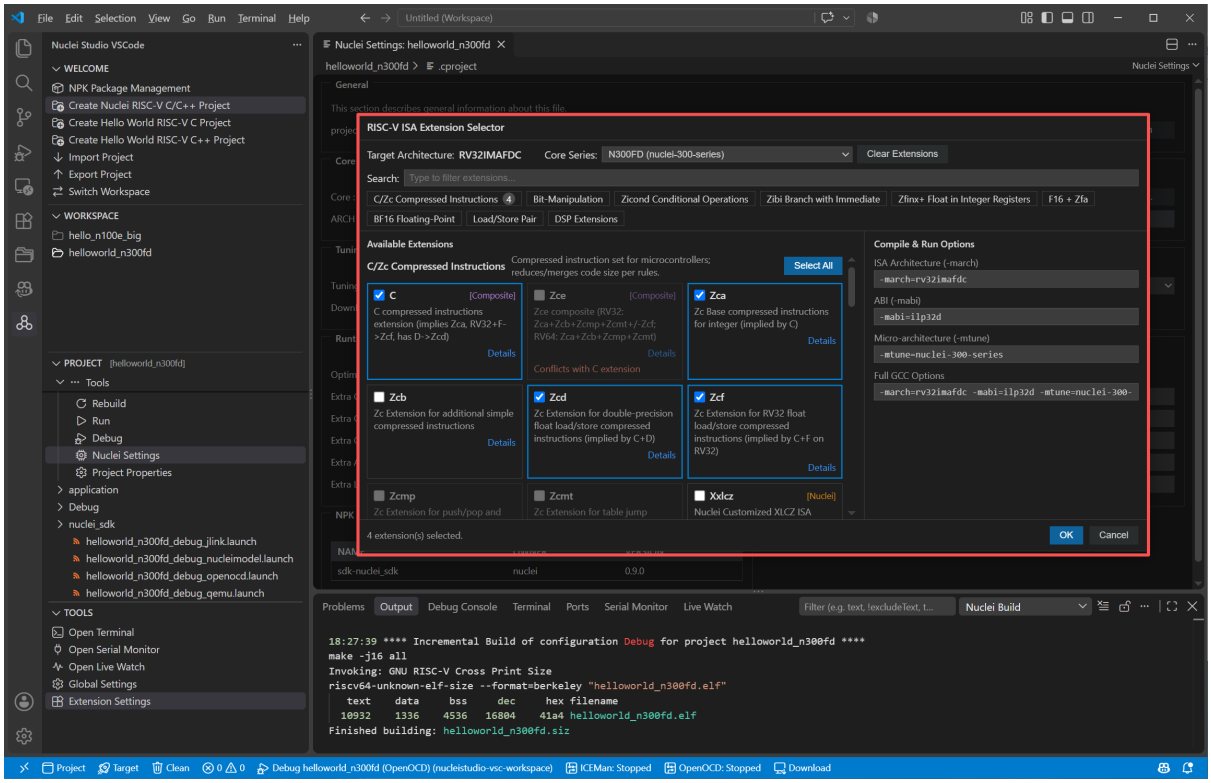
- **NPK component package information**: the NPK packages the project depends on.

Click `Save settings` (or press `Ctrl+S`) to apply the changes; the project's `.cproject`, `.nuproject`, and `.launch` files are updated accordingly.



RISC-V ISA Extension Selector

Click `Select...` next to `Other extensions` in the Nuclei Settings page to open the **RISC-V ISA Extension Selector**. Select the target architecture (RV32/RV64) and core series, then check the extensions in the Available Extensions list; dependency and conflict rules between extensions are handled automatically. The resulting `-march`, `-mabi`, `-mtune`, and the full GCC options are previewed on the right. Confirm with `OK` to fill the extension string back into the Nuclei Settings page.



3.8.3 Project Properties and Resource Properties

- **Project Properties**: right-click a project and choose `Project Properties` to edit resource information, linked resources, build environment, builder settings and behavior, and the tool settings of each toolchain tool (target processor, optimization, warnings, debugging, and more). The Target Processor page also provides the **Endian** option (`-mbig-endian` / `-mlittle-endian`), typically used with the RISC-V Big Endian GCC/Newlib toolchain.
- **Resource Properties**: right-click a file or folder and choose `Resource Properties` to view and edit the settings of that resource, such as excluding it from the build or customizing the build of a single file.

3.8.4 Open Terminal

Click `Open Terminal` in the TOOLS panel to open a **Nuclei Studio Terminal**, in which the paths of the build tools, OpenOCD, all configured toolchains, J-Link, Nuclei Model, and QEMU are prepended to `PATH`, so the Nuclei tools can be used directly from the command line.

In the PROJECT view, right-click a folder and choose `Show in Local Terminal` -> `Terminal` or `Git Bash` to open a terminal at that location.

3.8.5 IntelliSense

If the Microsoft C/C++ extension (`ms-vscode.cpptools`) is installed, Nuclei Studio VSCode provides IntelliSense for the projects: the include paths, defines, and compiler options are computed from the same build model used for building, and kept up to date when the project changes.

- Symbol browsing is scoped to the active project.
- Click the Rebuild Index button in the WORKSPACE view title bar to rebuild the IntelliSense index.

3.8.6 Project Conversion

Right-click a project and choose `Convert GCC 10 Project to GCC 13` to convert a project created with the legacy GCC 10 toolchain to the GCC 13 toolchain: the toolchain prefix and options in `.cproject` are rewritten, and the related `.launch` files are adjusted. Rebuild the project after the conversion.

3.9 Nuclei Studio VSCode Update and Upgrade

3.9.1 Update the Extension

If the extension is installed from the VS Code Marketplace, it is updated automatically by VS Code; you can also check for updates manually in the Extensions view. If the extension is installed from a local `nucleistudio.vsix` package, install the newer package with `Install from VSIX...` in the same way as the initial installation (see *Nuclei Studio VSCode Download and Installation* (page 405)). The extension settings and workspaces are preserved across updates.

3.9.2 Update the Nuclei Studio IDE and Tools

The build tools, toolchains, OpenOCD, and Nuclei Model used by the extension come from the Nuclei Studio IDE installation; their versions follow the Nuclei Studio release version (such as 2026.10), not the extension version. To upgrade these tools, upgrade the Nuclei Studio IDE itself; see *Nuclei Studio Update and Upgrade* (page 358).

If the IDE is installed to a new folder, update the `nucleiStudio.installFolder` setting to the new installation folder; the paths of the build tools, toolchains, OpenOCD, Nuclei Model, and NPK packages are then derived again automatically (see *Configure the Nuclei Studio IDE Installation Folder* (page 408)). A warning lists any derived path that does not exist.

3.9.3 Update NPK Packages

Open the NPK Package Management page and click `Refresh` to get the latest package list from the cloud, then download the new versions of the packages you need; see *NPK Package Management in Nuclei Studio VSCode* (page 409). Old versions can be removed with `Remove`.

3.9.4 Update the RISC-V Extension Data

The extension data used by the RISC-V ISA Extension Selector (see *RISC-V ISA Extension Selector* (page 421)) is updated automatically: the extension checks for new extension data in the background at most once every 7 days, and updates the local cache silently. No manual operation is required; the bundled data is used when offline.

3.10 Nuclei Studio VSCode FAQ

3.10.1 The Nuclei Studio VSCode Icon Does Not Appear in the Activity Bar

The extension depends on the following companion extensions: Serial Monitor (`ms-vscode.vscode-serial-monitor`), CDT GDB Debug Adapter Extension (`eclipse-cdt.cdt-gdb-vscode`), Memory Inspector (`eclipse-cdt.memory-inspector`), and Peripheral Inspector (`eclipse-cdt.peripheral-inspector`). If any of them was installed before but is disabled, this extension cannot work properly. Enable them one by one in the Extensions view, then reload the window (`Ctrl+Shift+P` -> `Developer: Reload Window`).

3.10.2 VS Code Reports That the Workspace Is in Restricted Mode

Click the `Manage` button in the notification and choose `Trust` for the workspace; otherwise the extension features will not work properly.

3.10.3 A "Configure Nuclei Studio Install Folder" Warning Is Shown in the WELCOME Panel

The Nuclei Studio IDE installation folder is not configured. Click the warning item and select the folder where Nuclei Studio IDE is installed, or set `nucleiStudio.installFolder` in the VS Code Settings page. See *Configure the Nuclei Studio IDE Installation Folder* (page 408).

3.10.4 "Please install the required NPK packages first." When Creating a Project

The project wizard requires at least the SSP and BSP packages of your target. Click `Open NPK Package Management` in the dialog to install the required NPK packages first; see *NPK Package Management in Nuclei Studio VSCode* (page 409).

3.10.5 "Current workspace differs from Nuclei Studio VSCode workspace."

The folder opened in VS Code is not the Nuclei Studio workspace. Click the `Switch to Nuclei Studio VSCode workspace` entry shown in the WORKSPACE view to switch.

3.10.6 Build Fails Because the Toolchain or `make` Cannot Be Found

The paths of the toolchain and build tools are derived from the Nuclei Studio IDE installation folder. Check that `nucleiStudio.installFolder` points to a complete Nuclei Studio IDE installation; when it is changed, a warning lists the derived paths that do not exist. See *Configure the Nuclei Studio IDE Installation Folder* (page 408).

3.10.7 The Debug Session Does Not Start

- The project is always built before debugging; if the build fails, the debug session is not started. Fix the build errors first; see *Build a Project in Nuclei Studio VSCode* (page 413).
- If an error such as "GDB executable not found" or "GDB server executable not found" is reported, check the derived toolchain and debug server paths; see *Configure the Nuclei Studio IDE Installation Folder* (page 408).
- If "No active Nuclei project found" is reported, select a project in the WORKSPACE view first.

3.10.8 Live Watch Reports That the Feature Is Not Supported

Live Watch requires a debug session based on Nuclei OpenOCD with the SBA feature, and the CPU must support SBA; it does not work with J-Link, QEMU, or Nuclei Model sessions. If "Unable to connect openocd telnet server" is reported, check the telnet port in the OpenOCD launch configuration. See *Live Watch* (page 418).

3.10.9 Rebuild Index Reports That the C/C++ Extension Is Not Available

IntelliSense and Rebuild Index are implemented through the Microsoft C/ C++ extension (`ms-vscode.cpptools`). Install it from the VS Code Marketplace. See *IntelliSense* (page 422).

3.10.10 "Debug As⋯" Is Not Available for a Launch File

The graphical launch configuration editors only support OpenOCD, J-Link, QEMU, and Nuclei Model `.launch` files. Other `.launch` files (such as QEMU ARM/AArch64 or Custom) can still be used with `F5` / Debug Project. See *Debug and Run a Project in Nuclei Studio VSCode* (page 415).

3.10.11 Where Are the NPK Packages Installed

Packages are installed under the user path (default: the user home directory) or the global path (default: the `Packages` folder of the Nuclei Studio IDE installation), which can be changed with `nucleiStudio.npk.userPath` and `nucleiStudio.npk.globalPath`. See *NPK Package Management in Nuclei Studio VSCode* (page 409).

CHAPTER

FOUR

NUCLEI TOOLCHAIN

4.1 GNU Toolchain

4.1.1 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.

The latest release **2026.10** branch for gcc14 and llvm19 is `nuclei/2026.10`, in which the tools included versions are: gcc 14.2.1, binutils 2.44, gdb 16.2, newlib 4.5.0, llvm 19.1.7, glibc 2.41, and also have merged some important patches from their upstream, as well as additional support for Nuclei custom extensions and pipelines, etc.

Note: The toolchain is implemented based on the upstream version. Many features can be referenced in the GCC online documentation: https://gcc.gnu.org/onlinedocs/. Please check the corresponding GNU release version in the online documentation for accurate details. Additionally, local documentation is available under `gcc/share/doc` in the toolchain directory.

4.1.2 Extensions Support

Standard Extensions

- Basic Extensions

  i, m, a, f, d, c, h, q(Assembly only), zaamo, zalrsc, zicsr, zifencei, zicond, zibi, 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.

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

Note: The Zcmt table jump auto-generation conditions (the linker relaxes the jumps into `jt`/`jalt` table jump instructions):

- 2025.10: enabled by default. When `zcmt` is present in `-march=`, the Zcmt table jump instructions are generated automatically.
- 2026.10: both conditions must be satisfied: `zcmt` present in `-march=`, and the linker option `--relax-zcmt` (pass it as `-Wl,--relax-zcmt` when linking with GCC). The default `--no-relax-zcmt` disables the auto-generation.

Attention: Projects that relied on the automatic Zcmt table jump generation in 2025.10 will get larger code after upgrading to 2026.10 unless `-Wl,--relax-zcmt` is added to the link options.

Attention: The Zcmt table jump support has not been merged into the binutils mainline branch yet, it is an experimental feature in the Nuclei toolchain. Please be aware of the risks when using it in production.

- Zvb Extensions

  zvbb, zvbc

- Zvk Extensions

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

- BFloat 16 Extensions
  - Standard: Zfbfmin, Zvfbfmin, Zvfbfwma
  - Nuclei customized: *Xxlfbf, Xxlvfbf* (page 435)

- Zilsd Extensions

  Zilsd, Zclsd

- Smrnmi & Double Trap Extensions (added since 2026.10)

  smrnmi, smdbltrp, ssdbltrp.

Nuclei Custom Extensions

- Packed SIMD Extension 0.5.4 + Nuclei Customized DSP Extensions

  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.

- Nuclei Custom Vector MAC Extension

xxlvqmacc (page 445)

- Nuclei Custom Xxlvw Vector Wireless DSP Extension

  xxlvw (page 451)

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

4.1.3 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 specified processor by either microarchitecture or a specific CPU name. The allowable values for this option include: `nuclei-100-series`, `nuclei-200-series`, `nuclei-300-series`, `nuclei-600-series`, `nuclei-900-series`, `nuclei-1000-series`, `nuclei-1000-3w-series`, and `nuclei-1000-4w-series`. Note that `nuclei-1000-series` and `nuclei-1000-4w-series` are considered equivalent. All these options are valid for the -mcpu= flag.

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.

-fno-strict-aliasing

It is recommended to add the optimization option `-fno-strict-aliasing` to the project. In some circumstances, `-fstrict-aliasing` (enabled by default at `-O2` and above) allows the compiler to assume that pointers to different types do not alias, which may break code that accesses the same memory through pointers of different types.

-ftree-loop-vectorize

If you need to disable the RISC-V RVV automatic vectorization, you can use the options `-fno-tree-loop-vectorize` and `-fno-tree-slp-vectorize`. Since 2026.10, you can also use the option `-mno-autovec-rvv`. For GCC 13, you can use `--param=riscv-autovec-preference=none`.

-fno-builtin

The `-fno-builtin` option instructs the compiler to avoid replacing standard library function calls with optimized built-in versions. If your program requires implementing its own system functions, such as memcpy, memset, etc., you need to use this option.

-fstack-usage

Generate a stack usage file (`<auxname>.su`) for each translation unit at compile time, which can be used to analyze the static stack usage of each function. Each entry records the function name, the size of its stack frame in bytes, and one or more qualifiers: `static` (a fixed number of bytes allocated at entry and released at exit), `dynamic` (the function also adjusts the stack in its body, such as alloca or variable length arrays) and `bounded` (the dynamic adjustments are bounded at compile time, and the number is an upper bound on the total stack usage). The file is generated next to the object file.

-fcallgraph-info

Generate a callgraph file (`<auxname>.ci` in VCG format) for each translation unit at compile time, which records the static call relationships between functions. With the `su` mark (`-fcallgraph-info=su`), each function in the callgraph is annotated with its stack usage information, so that the call chains and the worst-case stack usage can be analyzed statically. For example:

```
riscv64-unknown-elf-gcc -c -O2 -fcallgraph-info=su -fstack-usage main.c -
↪o main.o
```

generates `main.o`, `main.su` and `main.ci` in the output directory.

Note: When compiling with `-flto`, no callgraph information is output along with the object file.

Third-party tools can be used to analyze the generated files, for example:

- callgraph[27]: aggregates the `.ci` callgraph files generated by GCC into a single DOT file, which can be rendered by Graphviz to visualize the call graph of the whole program.
- checkStackUsage[28]: combines the `.su` stack usage files with the disassembly of the ELF binary to compute the accumulated worst-case stack usage of each call chain. Note that calls made through function pointers cannot be seen by the static analysis.

Optimization Options

-O0

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

[27] https://github.com/Miliox/callgraph
[28] https://github.com/ttsiodras/checkStackUsage

-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-14.2.0/gcc/RISC-V-Options.html

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

4.1.4 Nuclei Custom Options

In addition to the standard RISC-V options above, the Nuclei GNU toolchain provides the following Nuclei custom options:

-mautovec-rvv/-mno-autovec-rvv

Controls the RISC-V RVV automatic vectorization. Enabled by default, use `-mno-autovec-rvv` to disable it. This option is added in 2026.10.

Since 2026.10, RVV automatic vectorization is always disabled in interrupt handlers (functions marked with the `interrupt` attribute), regardless of this option. In interrupt handlers, vector registers are only saved when they are directly used by the handler itself; if the handler or its callees use the vector unit, the full vector state must be saved and restored by the user.

-mautovec-dsp/-mno-autovec-dsp

Controls the generation of automatic vectorization of Nuclei DSP instructions. Since 2026.10, Nuclei DSP instruction auto-vectorization is disabled by default, use `-mautovec-dsp` to enable it.

-mautogen-xxlcz/-mno-autogen-xxlcz

Controls the auto-generation of the Nuclei `xxlcz` extension instructions (including the `xl.addibne` instruction). Disabled by default, use `-mautogen-xxlcz` to enable it. This option is added in 2026.10.

-maddibne/-mno-addibne

This option was added in 2025.10 to control the `xl.addibne` instruction auto-generation, and is removed in 2026.10. The auto-generation is now controlled by the `-mautogen-xxlcz/-mno-autogen-xxlcz` option.

4.1.5 Libraries

Note:

- `glibc` is used in Linux GNU Glibc toolchain used to compile linux kernel, opensbi, uboot, and linux applications.
- `newlib` is used in Baremetal or RTOS toolchain, used to compile baremetal or rtos source code, which contains `newlib`, `newlib-nano` and *libncrt* (page 473)

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 (page 473)

`libncrt` is short for **Nuclei C Runtime Library**, which currently supports Nuclei RV32 processor, and is released by Nuclei to reduce c library code size, and improve math library speed, for details, please refer to *About Nuclei C Runtime Library* (page 473)

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`. For more details about libncrt usage, please refer to *Nuclei C Runtime Library for Nuclei Toolchain* (page 473).

4.1.6 Changelog

Please check *Changelog* (page 467) here.

4.1.7 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/2025/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/2025/README.md. To get other technical support, please send issues directly to the upstream repository https://github.com/riscv-collab/riscv-gnu-toolchain.

4.1.8 Testsuite

Since 2026.10, a GCC testsuite test framework has been added to the Nuclei toolchain, and the built-in testcases of GCC and Binutils are already covered by it:

- The built-in GCC and Binutils testsuites are run against the multilib combinations (target/library/extension/ABI) of the toolchain in the CI.
- The known failing testcases are recorded in the allowlist files under `test/allowlist/<toolname>/` of the toolchain repository, named by the rule `[<lib>.][rv(32|64|128).][<ext>.][<abi>.]log` and filtered by `scripts/testsuite-filter`, so the regressions can be detected easily.
- Nuclei custom extension testcases, such as `xxlfbf`, `xxlvfbf`, `xxlvw`, `xxlvqmacc`, `xxlcz`, `xxldsp` and the ECLIC V2 CSRs, are also added to the testsuite.

4.1.9 Checking GNU Version

To check the basic version information of GCC, you can use the command `riscv64-unknown-elf-gcc -v`.

If you need to report an issue to the developers, please provide:

1. The `gcc/build.txt` file (located in the toolchain directory) for toolchain build details.
2. The `gcc/gitrepo.txt` file to confirm commit IDs of each tool, which helps determine more detailed build information.

4.1.10 Known Issues

Note: This chapter only lists the bugs that are still unfixed at present (or known usage limitations). For the bugs that have already been fixed, and for more bug issues, please refer to https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/18.

`sscanf` Format Specifiers Issue in Newlib-nano

Note: This issue affects the 2025.02/2025.10/2026.10 toolchains, and is not fixed yet.

sscanf fails with hhu, llu, etc., due to missing C99 support (%hhX, %llX, %jX, %zX, %tX).

References:

- https://docs.zephyrproject.org/latest/develop/languages/c/newlib.html
- https://github.com/32bitmicro/newlib-nano-1.0/blob/master/newlib/README.nano
- https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/60

GDB RISC-V Memory Access at Address 0 When Symbols Are Missing

Note: This issue affects the 2025.02/2025.10/2026.10 toolchains, and is not fixed yet.

When debugging a RISC-V target via OpenOCD+GDB, running a stepping command (si) before loading an ELF with symbols causes GDB to incorrectly read memory near address 0x0.

References:

- https://github.com/riscv-mcu/riscv-openocd/issues/16

Access to Address 0 Is Compiled Into `ebreak`

Note: This issue affects the 2025.02/2025.10/2026.10 toolchains, and is not fixed yet.

GCC enables the `-fdelete-null-pointer-checks` optimization by default on most targets at all optimization levels. It assumes that a program cannot safely dereference the null pointer and that no code or data element resides at address zero. Under this assumption, access to address 0 is treated as undefined behavior and compiled into a trap instruction (`ebreak`), and null-pointer checks after a dereference are removed by the data flow analysis.

However, this assumption does not always hold on Nuclei CPUs: the `IREGION` can be configured so that address 0 is a valid address, for example when the `IINFO` region is mapped to address 0. In this case, accesses to address 0 are legal and the generated `ebreak` is wrong.

If the CPU is configured with `IREGION` mapped to address 0, you need to add the `-fno-delete-null-pointer-checks` option globally to your project to disable this optimization.

References:

- https://gcc.gnu.org/onlinedocs/gcc-14.4.0/gcc/Optimize-Options.html#index-fdelete-null-pointer-checks
- https://github.com/riscv-collab/riscv-gnu-toolchain/issues/694
- https://github.com/riscv-collab/riscv-gnu-toolchain/issues?q=-fno-delete-null-pointer-checks
- https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/61

`naked` Attribute With Variadic Functions Is Not Supported

Note: This issue affects the 2025.02/2025.10/2026.10 toolchains, and will not be fixed (upstream WONTFIX).

On RISC-V, do not use the `naked` function attribute together with variadic functions (varargs). When a `naked` function has a variadic parameter list, GCC still emits code from `riscv_setup_incoming_varargs` to spill the unnamed argument registers to the stack. Since the `naked` attribute suppresses the function prologue, these stores write into the caller ' s stack frame and corrupt memory.

Upstream has confirmed that naked functions and varargs are fundamentally incompatible with each other and should not be used together, and this issue is closed as WONTFIX and will not be fixed (GCC bug 125798[29]).

References:

- https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/62

[29] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=125798

4.1.11 Nuclei Custom BFloat16 Extension

Introduction

Matrix computations in machine learning and AI applications often require high throughput but relatively low precision. The traditional 32-bit floating-point format (FP32) imposes clear bottlenecks on both storage and computation speed, resulting in low efficiency and wasted resources. BF16 (Brain Floating-Point 16), an emerging 16-bit data type, offers a better balance between computational performance and precision, making it well suited to these applications.

To date, however, upstream RISC-V support for BF16 covers only a subset of conversion instructions and vector multiply-accumulate instructions, which falls far short of the complete requirements for applying BF16 to artificial intelligence, machine learning, and similar scenarios.

To close this gap, the Nuclei GCC toolchain, developed in close coordination with the Nuclei hardware design, maintains compatibility with the upstream BF16 instructions while providing full support for BF16 arithmetic and conversion operations, unlocking the potential of BF16 for a wide range of high-performance computing scenarios.

Register Configuration

Bit 1 (`fp16mode`) of the hardware custom register `mmisc_ctl1` controls the data format stored in the 16-bit FPU registers:

- **fp16mode = 0**: fp16 mode
- **fp16mode = 1**: bf16 mode

Therefore, before using the Nuclei BF16 arithmetic operations, the `fp16mode` bit in `mmisc_ctl1` must be set to 1.

Note: The Nuclei BF16 arithmetic operations reuse the fp16 instructions; this switch determines whether they execute in fp16 or bf16 mode.

The register configuration code is shown below:

```
#include "nuclei_sdk_soc.h" // This header file must be included

__RV_CSR_SET(CSR_MFP16MODE, 0x1); // Enable Nuclei BF16

__RV_CSR_CLEAR(CSR_MFP16MODE, 0x1); // Disable Nuclei BF16
```

BF16 Scalar Support

Extension

Nuclei BF16 scalar arithmetic requires the `xxlfbf` extension to be enabled.

Example: `-march=rv64imafdc_xxlfbf`

Supported Instructions

- `fadd.h/fsub.h/fmul.h/fdiv.h`
- `fmadd.h/fmsub.h/fnmadd.h/fnmsub.h`
- `fsqrt.h`
- `fcvt.bf16.s/fcvt.s.bf16` (compatible with the instructions of the Zfbfmin extension)
- `fcvt.h.d/fcvt.d.h` (added in the 2025.10 release)
- `fcvt.h.w[u]/fcvt.w[u].h` (int32 <—> bf16)
- `fcvt.h.l[u]/fcvt.l[u].h` (int64 <—> bf16)
- `fmin.h/fmax.h`
- `flt.h/fle.h/feq.h`
- `fclass.h`

Examples

Listing 4.1: Basic arithmetic on scalar BF16 values

```
#include <stdio.h>
#include "nuclei_sdk_soc.h"

__bf16 a = 1.11;
__bf16 b = 11.8;
const __bf16 c = 10.3;

void test_add(){
    __bf16 sum = a + b;
    printf("%f\n",sum);
}

void test_sub(){
    __bf16 sum = a-c;
    printf("%f\n",sum);
}

void test_mul(){
    __bf16 sum = a * 5;
    printf("%f\n",sum);
}

void test_div(){
    __bf16 sum = a/(__bf16)3.5;
    printf("%f\n",sum);
}

int main(void)
{
    __RV_CSR_SET(CSR_MFP16MODE, 0x1);
    test_add();
    test_sub();
    test_mul();
    test_div();

    return 0;
}
```

BF16 Vector Support (RVV Intrinsics)

Extension

The Nuclei BF16 RVV intrinsics require the `V` extension and the `xxlvfbf` extension to be enabled.

Example: `-march=rv64imafdcv_xxlvfbf`

Nuclei BF16 RVV Intrinsic Naming Scheme

RVV intrinsic naming convention: **Chapter 6** of `v-intrinsic-spec.pdf` in rvv-intrinsic-doc[30].

Our naming convention follows the above specification, with the additional prefix `_xl` to distinguish our custom intrinsics. For example, the intrinsic for the float16 `vfadd` operation is named:

```
vfloat16mf4_t __riscv_vfadd_vv_f16mf4(vfloat16mf4_t vs2, vfloat16mf4_t
vs1, size_t vl);
```

`vfloat16mf4_t` data type of vfloat16

`__riscv_` intrinsic name prefix

`vfadd` name of the operation performed by the intrinsic (vector addition in this case)

`vv` operand types of the operation (`v` denotes a vector type; see the RVV intrinsic naming convention above for details)

`f16` abbreviation of the float16 data type

`mf4` LMUL value

Nuclei custom bfloat16 intrinsics follow the same RVV intrinsic naming convention, with an additional prefix, as shown below:

```
vbfloat16mf4_t __riscv_xl_vfadd_vv_bf16mf4(vbfloat16mf4_t vs2,
vbfloat16mf4_t vs1, size_t vl);
```

`vbfloat16mf4_t` data type of vbfloat16

`__riscv_xl_` prefix for Nuclei custom intrinsics

`vfadd` name of the operation performed by the intrinsic

`vv` operand types of the operation

`bf16` abbreviation of the bfloat16 data type

`mf4` LMUL value

Note: For the vector data types of bfloat16, refer to vector-bfloat16-spec.adoc[31].

---

[30] https://github.com/riscv-non-isa/rvv-intrinsic-doc/releases/tag/v1.0-ratified
[31] https://github.com/riscv-non-isa/rvv-intrinsic-doc/blob/9328aba3fca494717de08502ff32819a7c168daa/doc/vector-bfloat16-spec.adoc

RVV Instructions Supported by Nuclei BF16

Nuclei Custom Instructions

- **Vector Single-Width Floating-Point Add/Subtract Instructions**
  - `vfadd.vv/vfadd.vf`
  - `vfsub.vv/vfsub.vf`
  - `vfrsub.vf`
- **Vector Widening Floating-Point Add/Subtract Instructions**
  - `vfwadd.vv/vfwadd.vf`
  - `vfwsub.vv/vfwsub.vf`
  - `vfwadd.wv/vfwadd.wf`
  - `vfwsub.wv/vfwsub.wf`
- **Vector Single-Width Floating-Point Multiply/Divide Instructions**
  - `vfmul.vv/vfmul.vf`
  - `vfdiv.vv/vfdiv.vf`
  - `vfrdiv.vf`
- **Vector Widening Floating-Point Multiply**
  - `vfwmul.vv/vfwmul.vf`
- **Vector Single-Width Floating-Point Fused Multiply-Add Instructions**
  - `vfmacc.vv/vfmacc.vf`
  - `vfnmacc.vv/vfnmacc.vf`
  - `vfmsac.vv/vfmsac.vf`
  - `vfnmsac.vv/vfnmsac.vf`
  - `vfmadd.vv/vfmadd.vf`
  - `vfnmadd.vv/vfnmadd.vf`
  - `vfmsub.vv/vfmsub.vf`
  - `vfnmsub.vv/vfnmsub.vf`
- **Vector Widening Floating-Point Fused Multiply-Add Instructions**
  - `vfwmacc.vv/vfwmacc.vf`
  - `vfwnmacc.vv/vfwnmacc.vf`
  - `vfwmsac.vv/vfwmsac.vf`
  - `vfwnmsac.vv/vfwnmsac.vf`
- **Vector Floating-Point Square-Root Instruction**
  - `vfsqrt.v`
- **Vector Floating-Point Reciprocal Square-Root Estimate Instruction**
  - `vfrsqrt7.v`
- **Vector Floating-Point Reciprocal Estimate Instruction**
  - `vfrec7.v`
- **Vector Floating-Point MIN/MAX Instructions**

  - vfmin.vv/vfmin.vf
  - vfmax.vv/vfmax.vf
- **Vector Floating-Point Sign-Injection Instructions**
  - vfsgnj.vv/vfsgnj.vf
  - vfsgnjn.vv/vfsgnjn.vf
  - vfsgnjx.vv/vfsgnjx.vf
- **Vector Floating-Point Compare Instructions**
  - vmfeq.vv/vmfeq.vf
  - vmfne.vv/vmfne.vf
  - vmflt.vv/vmflt.vf
  - vmfle.vv/vmfle.vf
  - vmfgt.vf
  - vmfge.vf
- **Vector Floating-Point Classify Instruction**
  - vfclass.v
- **Vector Floating-Point Merge Instruction**
  - vfmerge.vfm
- **Vector Floating-Point Move Instruction**
  - vmv.v
  - vfmv.v
- **Single-Width Floating-Point/Integer Type-Convert Instructions**
  - vfcvt.xu.f.v
  - vfcvt.x.f.v
  - vfcvt.rtz.xu.f.v
  - vfcvt.rtz.x.f.v
  - vfcvt.f.xu.v
  - vfcvt.f.x.v
- **Widening Floating-Point/Integer Type-Convert Instructions**
  - vfwcvt.xu.f.v
  - vfwcvt.x.f.v
  - vfwcvt.rtz.xu.f.v
  - vfwcvt.rtz.x.f.v
  - vfwcvt.f.xu.v
  - vfwcvt.f.x.v
  - vfwcvt.f.f.v
- **Narrowing Floating-Point/Integer Type-Convert Instructions**
  - vfncvt.xu.f.w
  - vfncvt.x.f.w
  - vfncvt.rtz.xu.f.w

  - vfncvt.rtz.x.f.w
  - vfncvt.f.xu.w
  - vfncvt.f.x.w
  - vfncvt.f.f.w
  - vfncvt.rod.f.f.w
- **Vector Single-Width Floating-Point Reduction Instructions**
  - vfredosum.vs
  - vfredusum.vs
  - vfredmax.vs
  - vfredmin.vs
- **Vector Widening Floating-Point Reduction Instructions**
  - vfwredosum.vs
  - vfwredusum.vs
- **Vector Slideup Instructions**
  - vslideup.vx
- **Vector Slidedown Instructions**
  - vslidedown.vx
- **Vector Floating-Point Slide1up Instruction**
  - vfslide1up.vf
- **Vector Floating-Point Slide1down Instruction**
  - vfslide1down.vf
- **Vector Register Gather Instructions**
  - vrgather.vv
  - vrgatherei16.vv
- **Vector Compress Instruction**
  - vcompress.vm

Instructions Compatible with Upstream

- **Vector Unit-Stride Load Intrinsics**
  - vle16.v
- **Vector Unit-Stride Store Intrinsics**
  - vse16.v
- **Vector Strided Load Intrinsics**
  - vlse16.v
- **Vector Strided Store Intrinsics**
  - vsse16.v
- **Vector Indexed Load Intrinsics**
  - vloxei[8/16/32/64].v
  - vluxei[8/16/32/64].v

- **Vector Indexed Store Intrinsics**
  - `vsoxei[8/16/32/64].v`
  - `vsuxei[8/16/32/64].v`
- **Unit-stride Fault-Only-First Loads Intrinsics**
  - `vle16ff.v`
- **Vector Unit-Stride Segment Load Intrinsics**
  - `vlseg[2-8]e16.v`
  - `vlseg[2-8]e16ff.v`
- **Vector Unit-Stride Segment Store Intrinsics**
  - `vsseg[2-8]e16.v`
- **Vector Strided Segment Load Intrinsics**
  - `vlsseg[2-8]e16.v`
- **Vector Strided Segment Store Intrinsics**
  - `vssseg[2-8]e16.v`
- **Vector Indexed Segment Load Intrinsics**
  - `vloxseg[2-8]ei8.v`
  - `vloxseg[2-8]ei16.v`
  - `vloxseg[2-8]ei32.v`
  - `vloxseg[2-8]ei64.v`
  - `vluxseg[2-8]ei8.v`
  - `vluxseg[2-8]ei16.v`
  - `vluxseg[2-8]ei32.v`
  - `vluxseg[2-8]ei64.v`
- **Vector Indexed Segment Store Intrinsics**
  - `vsoxseg[2-8]ei8.v`
  - `vsoxseg[2-8]ei16.v`
  - `vsoxseg[2-8]ei32.v`
  - `vsoxseg[2-8]ei64.v`
  - `vsuxseg[2-8]ei8.v`
  - `vsuxseg[2-8]ei16.v`
  - `vsuxseg[2-8]ei32.v`
  - `vsuxseg[2-8]ei64.v`

Note: 2025.02 release:

The custom Xxlvfbf extension is incompatible with the Zvfbfmin and Zvfbfwma extensions defined in riscv-bfloat16 release v1.0[32] (`riscv-bfloat16.pdf`). Their functionality is implemented by the `vfncvt.f.f.w`, `vfwcvt.f.f.v`, and `vfwmacc.vv/vfwmacc.vf` instructions; when using these operations, refer to the intrinsics of the corresponding instructions.

If the Zvfbfmin/Zvfbfwma intrinsics are required, they must be used separately, i.e., without the Xxlvfbf extension.

[32] https://github.com/riscv/riscv-bfloat16/releases

When both are present, the Zvfbfmin/Zvfbfwma intrinsics are mapped to `vfncvt.f.f.w`, `vfwcvt.f.f.v`, and `vfwmacc.vv/vfwmacc.vf`, which may produce incorrect results.

2025.10 release:

The custom Xxlvfbf extension is now compatible with the Zvfbfmin and Zvfbfwma extensions.

The intrinsics of the Zvfbfmin and Zvfbfwma extensions can be used correctly under Xxlvfbf and are properly mapped to the instructions defined by those extensions: `vfncvtbf16.f.f.w`, `vfwcvtbf16.f.f.v`, and `vfwmaccbf16.vv/vfwmaccbf16.vf`. They can be used together with our custom intrinsics without producing errors.

RVV Intrinsics Supported by Nuclei BF16

Nuclei Custom Intrinsics

Refer to **Appendix A** of `v-intrinsic-spec.pdf` in rvv-intrinsic-doc[33].

The RVV intrinsics corresponding to the Nuclei custom instructions differ from the float16 RVV intrinsics in the above document only in their names; see *Nuclei BF16 RVV Intrinsic Naming Scheme* (page 437) for details.

Note: Currently only Appendix A is fully supported; the intrinsics in Appendices B, C, and D have not yet been fully tested.

Example:

- Vector Single-Width Floating-Point Add/Subtract Instructions
  - Float16

```
vfloat16mf4_t    __riscv_vfadd_vv_f16mf4(vfloat16mf4_t    vs2,
vfloat16mf4_t vs1, size_t vl);

vfloat16mf4_t     __riscv_vfadd_vv_f16mf4_m(vbool64_t     vm,
vfloat16mf4_t vs2, vfloat16mf4_t vs1, size_t vl);
```

  - Nuclei Bfloat16

```
vbfloat16mf4_t __riscv_xl_vfadd_vv_bf16mf4(vbfloat16mf4_t vs2,
vbfloat16mf4_t vs1, size_t vl);

vbfloat16mf4_t   __riscv_xl_vfadd_vv_bf16mf4_m(vbool64_t   vm,
vbfloat16mf4_t vs2, vbfloat16mf4_t vs1, size_t vl);
```

- Vector Widening Floating-Point Add/Subtract Instructions
  - Float16

```
vfloat32mf2_t    __riscv_vfwadd_vf_f32mf2(vfloat16mf4_t    vs2,
_Float16 rs1, size_t vl);
```

  - Nuclei Bfloat16

```
vfloat32mf2_t __riscv_xl_vfwadd_vf_f32mf2(vbfloat16mf4_t vs2,
__bf16 rs1, size_t vl);
```

[33] https://github.com/riscv-non-isa/rvv-intrinsic-doc/releases/tag/v1.0-ratified

Intrinsics Compatible with Upstream

The naming of these intrinsics also differs from that of the float16 RVV intrinsics in **Appendix A** of `v-intrinsic-spec.pdf` in rvv-intrinsic-doc[34] only in the names themselves. To obtain the corresponding names, simply replace the float16 data types with the bfloat16 data types.

Example:

- Vector Unit-Stride Load Intrinsics
  - Float16

```
vfloat16mf4_t __riscv_vle16_v_f16mf4(const _Float16 *rs1,
size_t vl);
```

  - Bfloat16

```
vbfloat16mf4_t __riscv_vle16_v_bf16mf4(const __bf16 *rs1,
size_t vl);
```

For the detailed intrinsic APIs, refer to bfloat16 intrinsic_funcs[35].

Note:

1. The intrinsics of some upstream-supported instructions are not listed in bfloat16 intrinsic_funcs[36], but they remain available for use, as follows:

- **Vector Indexed Load Intrinsics**
  - `vloxei[8/32/64].v`
  - `vluxei[8/32/64].v`
- **Vector Indexed Store Intrinsics**
  - `vsoxei[8/32/64].v`
  - `vsuxei[8/32/64].v`
- **Vector Indexed Segment Load Intrinsics**
  - `vloxseg[2-8]ei8.v`
  - `vloxseg[2-8]ei32.v`
  - `vloxseg[2-8]ei64.v`
  - `vluxseg[2-8]ei8.v`
  - `vluxseg[2-8]ei32.v`
  - `vluxseg[2-8]ei64.v`
- **Vector Indexed Segment Store Intrinsics**
  - `vsoxseg[2-8]ei8.v`
  - `vsoxseg[2-8]ei32.v`
  - `vsoxseg[2-8]ei64.v`
  - `vsuxseg[2-8]ei8.v`
  - `vsuxseg[2-8]ei32.v`
  - `vsuxseg[2-8]ei64.v`

[34] https://github.com/riscv-non-isa/rvv-intrinsic-doc/releases/tag/v1.0-ratified
[35] https://github.com/riscv-non-isa/rvv-intrinsic-doc/tree/9328aba3fca494717de08502ff32819a7c168daa/auto-generated/bfloat16/intrinsic_funcs
[36] https://github.com/riscv-non-isa/rvv-intrinsic-doc/tree/9328aba3fca494717de08502ff32819a7c168daa/auto-generated/bfloat16/intrinsic_funcs

For the corresponding intrinsic names, refer to *Intrinsics Compatible with Upstream* (page 443).

2. In bfloat16 intrinsic_funcs[37], the following items in `03_bfloat16_arithmetic_intrinsics.adoc`:

- Vector BFloat16 Move Intrinsics
- Vector BFloat16 Merge Intrinsics

have been redefined by Nuclei BFloat16; when using them, refer to *Nuclei Custom Intrinsics* (page 442).

Examples

Attention: After completing the register configuration, the following header file must also be included to use the Nuclei BF16 RVV intrinsics:

```
#include <riscv_vector.h>
```

Listing 4.2: An implementation of the add function using Nuclei BFloat16 RVV intrinsics.

```
#include <stdio.h>
#include "nuclei_sdk_soc.h"
#include <riscv_vector.h>

void csr_set_bf16_mode(void)
{
    __RV_CSR_SET(CSR_MFP16MODE, 0x1);
} // Enable Nuclei BF16


void csr_clr_bf16_mode(void)
{
    __RV_CSR_CLEAR(CSR_MFP16MODE, 0x1);
} // Disable Nuclei BF16

typedef __bf16 bfloat16_t;

void Add_bfloat16_rvv(bfloat16_t *pSrcA, bfloat16_t *pSrcB, bfloat16_t *pDst,␣
↪uint32_t blockSize)
{
    size_t blkCnt = blockSize; /* Loop counter */
    size_t vl;
    vbfloat16m1_t vx, vy;
    for (; (vl = __riscv_vsetvl_e16m1(blkCnt)) > 0; blkCnt -= vl) {
        vx = __riscv_vle16_v_bf16m1(pSrcA, vl);
        vy = __riscv_vle16_v_bf16m1(pSrcB, vl);
        pSrcA += vl;
        pSrcB += vl;
        __riscv_vse16_v_bf16m1(pDst, __riscv_xl_vfadd_vv_bf16m1(vx, vy, vl), vl);
        pDst += vl;
    }
}


int main()
{
    csr_set_bf16_mode();
```

(continues on next page)

[37] https://github.com/riscv-non-isa/rvv-intrinsic-doc/tree/9328aba3fca494717de08502ff32819a7c168daa/auto-generated/bfloat16/intrinsic_funcs

(continued from previous page)

```
    bfloat16_t SrcA[8] = {5.312, 5.312, 5.312, 5.312, 5.312, 5.312, 5.312, 5.312};
    bfloat16_t SrcB[8] = {5.312, 5.312, 5.312, 5.312, 5.312, 5.312, 5.312, 5.312};
    bfloat16_t Dst[8];

    Add_bfloat16_rvv(SrcA, SrcB, Dst, 8);

    printf("Dst[3] = %f\n", (float)Dst[3]);

    csr_clr_bf16_mode();

    return 0;
}
```

4.1.12 Nuclei Custom VPU Extension

Introduction

To accelerate matrix computations, the Nuclei GCC toolchain provides the custom VPU extension `Xxlvqmacc`, which introduces integer matrix multiply-accumulate instructions that widen 8-bit integer input values to 32 bits.

Extension

Using this extension requires the `V` extension and the `xxlvqmacc` extension to be enabled.

Example: `-march=rv64imafdcv_xxlvqmacc`

> **Attention:** To use the intrinsics of this extension, the following header file must be included:
>
> `#include <riscv_vector.h>`

Supported Instructions

- `xl.vqmacc.vv`
- `xl.vqmaccu.vv`
- `xl.vqmaccus.vv`
- `xl.vqmaccsu.vv`

Note: These instructions perform signed or unsigned integer matrix multiply-accumulate operations and widen 8-bit integer inputs to 32 bits.

Their functionality is illustrated by the following examples:

Dest (C) += Input (A) * Input (B)

int32 += int8 * int8 //xl.vqmacc.vv vd, vs2, vs1

int32 += uint8 * uint8 //xl.vqmaccu.vv vd, vs2, vs1

int32 += uint8 * int8 //xl.vqmaccus.vv vd, vs2, vs1

int32 += int8 * uint8 //xl.vqmaccsu.vv vd, vs2, vs1

where the (u)int8 matrices A and B and the int32 matrix C are all 4x4.

These instructions are designed to process 8-bit integer matrix operations efficiently while preventing overflow of intermediate results through widening to 32 bits.

Intrinsic Naming Convention

RVV intrinsic naming convention: **Chapter 6** of `v-intrinsic-spec.pdf` in https://github.com/riscv-non-isa/rvv-intrinsic-doc/releases/tag/v1.0-ratified.

Our naming convention follows the above specification with some modifications to distinguish our custom intrinsics. For example:

`vint32m1_t __riscv_xl_vqmacc_4x4x4_i32m1(vint32m1_t vd, vint8m1_t vs1, vint8mf4_t vs2, size_t vl);`

`vint32m1_t` data type of int32_t

`__riscv_xl` prefix for Nuclei custom intrinsics

`vqmacc` name of the operation performed by the intrinsic

`4x4x4` dimensions of the input/output matrices: input matrix A is `4x4`, input matrix B is `4x4`, and output matrix C is `4x4`

`i32` abbreviation of the int32 data type

`m1` LMUL value

Nuclei Custom Intrinsics

- `xl.vqmacc.vv`

```
vint32m1_t __riscv_xl_vqmacc_4x4x4_i32m1(vint32m1_t vd, vint8m1_t vs1, vint8mf4_t␣
↪vs2, size_t vl);
vint32m2_t __riscv_xl_vqmacc_4x4x4_i32m2(vint32m2_t vd, vint8m1_t vs1, vint8mf2_t␣
↪vs2, size_t vl);
vint32m4_t __riscv_xl_vqmacc_4x4x4_i32m4(vint32m4_t vd, vint8m1_t vs1, vint8m1_t␣
↪vs2, size_t vl);
vint32m8_t __riscv_xl_vqmacc_4x4x4_i32m8(vint32m8_t vd, vint8m1_t vs1, vint8m2_t␣
↪vs2, size_t vl);
vint32m1_t __riscv_xl_vqmacc_4x4x4(vint32m1_t vd, vint8m1_t vs1, vint8mf4_t vs2,␣
↪size_t vl);
vint32m2_t __riscv_xl_vqmacc_4x4x4(vint32m2_t vd, vint8m1_t vs1, vint8mf2_t vs2,␣
↪size_t vl);
vint32m4_t __riscv_xl_vqmacc_4x4x4(vint32m4_t vd, vint8m1_t vs1, vint8m1_t vs2,␣
↪size_t vl);
vint32m8_t __riscv_xl_vqmacc_4x4x4(vint32m8_t vd, vint8m1_t vs1, vint8m2_t vs2,␣
↪size_t vl);
vint32m1_t __riscv_xl_vqmacc_4x4x4_i32m1_tu(vint32m1_t vd, vint8m1_t vs1, vint8mf4_
↪t vs2, size_t vl);
vint32m2_t __riscv_xl_vqmacc_4x4x4_i32m2_tu(vint32m2_t vd, vint8m1_t vs1, vint8mf2_
↪t vs2, size_t vl);
vint32m4_t __riscv_xl_vqmacc_4x4x4_i32m4_tu(vint32m4_t vd, vint8m1_t vs1, vint8m1_
↪t vs2, size_t vl);
vint32m8_t __riscv_xl_vqmacc_4x4x4_i32m8_tu(vint32m8_t vd, vint8m1_t vs1, vint8m2_
↪t vs2, size_t vl);
vint32m1_t __riscv_xl_vqmacc_4x4x4_tu(vint32m1_t vd, vint8m1_t vs1, vint8mf4_t vs2,
↪ size_t vl);
vint32m2_t __riscv_xl_vqmacc_4x4x4_tu(vint32m2_t vd, vint8m1_t vs1, vint8mf2_t vs2,
↪ size_t vl);
vint32m4_t __riscv_xl_vqmacc_4x4x4_tu(vint32m4_t vd, vint8m1_t vs1, vint8m1_t vs2,␣
↪size_t vl);
```

(continues on next page)

(continued from previous page)

```
vint32m8_t __riscv_xl_vqmacc_4x4x4_tu(vint32m8_t vd, vint8m1_t vs1, vint8m2_t vs2,␣
↪size_t vl);
```

- xl.vqmaccu.vv

```
vint32m1_t __riscv_xl_vqmaccu_4x4x4_i32m1(vint32m1_t vd, vuint8m1_t vs1, vuint8mf4_
↪t vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccu_4x4x4_i32m2(vint32m2_t vd, vuint8m1_t vs1, vuint8mf2_
↪t vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccu_4x4x4_i32m4(vint32m4_t vd, vuint8m1_t vs1, vuint8m1_
↪t vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccu_4x4x4_i32m8(vint32m8_t vd, vuint8m1_t vs1, vuint8m2_
↪t vs2, size_t vl);
vint32m1_t __riscv_xl_vqmaccu_4x4x4(vint32m1_t vd, vuint8m1_t vs1, vuint8mf4_t vs2,
↪ size_t vl);
vint32m2_t __riscv_xl_vqmaccu_4x4x4(vint32m2_t vd, vuint8m1_t vs1, vuint8mf2_t vs2,
↪ size_t vl);
vint32m4_t __riscv_xl_vqmaccu_4x4x4(vint32m4_t vd, vuint8m1_t vs1, vuint8m1_t vs2,␣
↪size_t vl);
vint32m8_t __riscv_xl_vqmaccu_4x4x4(vint32m8_t vd, vuint8m1_t vs1, vuint8m2_t vs2,␣
↪size_t vl);
vint32m1_t __riscv_xl_vqmaccu_4x4x4_i32m1_tu(vint32m1_t vd, vuint8m1_t vs1,␣
↪vuint8mf4_t vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccu_4x4x4_i32m2_tu(vint32m2_t vd, vuint8m1_t vs1,␣
↪vuint8mf2_t vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccu_4x4x4_i32m4_tu(vint32m4_t vd, vuint8m1_t vs1,␣
↪vuint8m1_t vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccu_4x4x4_i32m8_tu(vint32m8_t vd, vuint8m1_t vs1,␣
↪vuint8m2_t vs2, size_t vl);
vint32m1_t __riscv_xl_vqmaccu_4x4x4_tu(vint32m1_t vd, vuint8m1_t vs1, vuint8mf4_t␣
↪vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccu_4x4x4_tu(vint32m2_t vd, vuint8m1_t vs1, vuint8mf2_t␣
↪vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccu_4x4x4_tu(vint32m4_t vd, vuint8m1_t vs1, vuint8m1_t␣
↪vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccu_4x4x4_tu(vint32m8_t vd, vuint8m1_t vs1, vuint8m2_t␣
↪vs2, size_t vl);
```

- xl.vqmaccus.vv

```
vint32m1_t __riscv_xl_vqmaccus_4x4x4_i32m1(vint32m1_t vd, vuint8m1_t vs1, vint8mf4_
↪t vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccus_4x4x4_i32m2(vint32m2_t vd, vuint8m1_t vs1, vint8mf2_
↪t vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccus_4x4x4_i32m4(vint32m4_t vd, vuint8m1_t vs1, vint8m1_
↪t vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccus_4x4x4_i32m8(vint32m8_t vd, vuint8m1_t vs1, vint8m2_
↪t vs2, size_t vl);
vint32m1_t __riscv_xl_vqmaccus_4x4x4(vint32m1_t vd, vuint8m1_t vs1, vint8mf4_t vs2,
↪ size_t vl);
vint32m2_t __riscv_xl_vqmaccus_4x4x4(vint32m2_t vd, vuint8m1_t vs1, vint8mf2_t vs2,
↪ size_t vl);
vint32m4_t __riscv_xl_vqmaccus_4x4x4(vint32m4_t vd, vuint8m1_t vs1, vint8m1_t vs2,␣
↪size_t vl);
vint32m8_t __riscv_xl_vqmaccus_4x4x4(vint32m8_t vd, vuint8m1_t vs1, vint8m2_t vs2,␣
↪size_t vl);
vint32m1_t __riscv_xl_vqmaccus_4x4x4_i32m1_tu(vint32m1_t vd, vuint8m1_t vs1,␣
↪vint8mf4_t vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccus_4x4x4_i32m2_tu(vint32m2_t vd, vuint8m1_t vs1,␣
↪vint8mf2_t vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccus_4x4x4_i32m4_tu(vint32m4_t vd, vuint8m1_t vs1,␣
```

(continues on next page)

(continued from previous page)

```
↪vint8m1_t vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccus_4x4x4_i32m8_tu(vint32m8_t vd, vuint8m1_t vs1,␣
↪vint8m2_t vs2, size_t vl);
vint32m1_t __riscv_xl_vqmaccus_4x4x4_tu(vint32m1_t vd, vuint8m1_t vs1, vint8mf4_t␣
↪vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccus_4x4x4_tu(vint32m2_t vd, vuint8m1_t vs1, vint8mf2_t␣
↪vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccus_4x4x4_tu(vint32m4_t vd, vuint8m1_t vs1, vint8m1_t␣
↪vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccus_4x4x4_tu(vint32m8_t vd, vuint8m1_t vs1, vint8m2_t␣
↪vs2, size_t vl);
```

- `xl.vqmaccsu.vv`

```
vint32m1_t __riscv_xl_vqmaccsu_4x4x4_i32m1(vint32m1_t vd, vint8m1_t vs1, vuint8mf4_
↪t vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccsu_4x4x4_i32m2(vint32m2_t vd, vint8m1_t vs1, vuint8mf2_
↪t vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccsu_4x4x4_i32m4(vint32m4_t vd, vint8m1_t vs1, vuint8m1_
↪t vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccsu_4x4x4_i32m8(vint32m8_t vd, vint8m1_t vs1, vuint8m2_
↪t vs2, size_t vl);
vint32m1_t __riscv_xl_vqmaccsu_4x4x4(vint32m1_t vd, vint8m1_t vs1, vuint8mf4_t vs2,
↪ size_t vl);
vint32m2_t __riscv_xl_vqmaccsu_4x4x4(vint32m2_t vd, vint8m1_t vs1, vuint8mf2_t vs2,
↪ size_t vl);
vint32m4_t __riscv_xl_vqmaccsu_4x4x4(vint32m4_t vd, vint8m1_t vs1, vuint8m1_t vs2,␣
↪size_t vl);
vint32m8_t __riscv_xl_vqmaccsu_4x4x4(vint32m8_t vd, vint8m1_t vs1, vuint8m2_t vs2,␣
↪size_t vl);
vint32m1_t __riscv_xl_vqmaccsu_4x4x4_i32m1_tu(vint32m1_t vd, vint8m1_t vs1,␣
↪vuint8mf4_t vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccsu_4x4x4_i32m2_tu(vint32m2_t vd, vint8m1_t vs1,␣
↪vuint8mf2_t vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccsu_4x4x4_i32m4_tu(vint32m4_t vd, vint8m1_t vs1,␣
↪vuint8m1_t vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccsu_4x4x4_i32m8_tu(vint32m8_t vd, vint8m1_t vs1,␣
↪vuint8m2_t vs2, size_t vl);
vint32m1_t __riscv_xl_vqmaccsu_4x4x4_tu(vint32m1_t vd, vint8m1_t vs1, vuint8mf4_t␣
↪vs2, size_t vl);
vint32m2_t __riscv_xl_vqmaccsu_4x4x4_tu(vint32m2_t vd, vint8m1_t vs1, vuint8mf2_t␣
↪vs2, size_t vl);
vint32m4_t __riscv_xl_vqmaccsu_4x4x4_tu(vint32m4_t vd, vint8m1_t vs1, vuint8m1_t␣
↪vs2, size_t vl);
vint32m8_t __riscv_xl_vqmaccsu_4x4x4_tu(vint32m8_t vd, vint8m1_t vs1, vuint8m2_t␣
↪vs2, size_t vl);
```

Examples

Listing 4.3: Example of using the intrinsic functions corresponding to `xl.vqmacc.vv`

```
#include <stdio.h>
#include <riscv_vector.h>


#define MATRIX_SIZE (4 * 4)
#define ARRAY_CNT   (4)
#define DATA_CNT    (ARRAY_CNT * MATRIX_SIZE)
```

(continues on next page)

(continued from previous page)

```
/* Case does matrix multiply-add like below:
*   C[j] += A * B[j], for j in [0, vl/16)
*/

void normal_case(int8_t *addr_in1, int8_t *addr_in2, int32_t *addr_out, int32_t␣
↪data_cnt)
{
    int8_t *pin1 = addr_in1;
    int8_t *pin2 = addr_in2;
    int32_t *pout = addr_out;
    int sum;
    int array_cnt = 0;

    while (data_cnt)
    {
        for (int32_t ii = 0; ii < 4; ii++)
        {
            for (int32_t jj = 0; jj < 4; jj++)
            {
                sum = 0;
                for (int32_t kk = 0; kk < 4; kk++)
                {
                    sum += pin1[ii * 4 + kk] * pin2[kk * 4 + jj];
                }
                pout[ii * 4 + jj] += sum;
            }
        }
        pin2 += 16;
        pout += 16;
        data_cnt -= 16;
    }
}

void vpu_case(int8_t *addr_in1, int8_t *addr_in2, int32_t *addr_out, int32_t data_
↪cnt)
{
    int8_t *pin1 = addr_in1;
    int8_t *pin2 = addr_in2;
    int32_t *pout = addr_out;
    size_t vl;

    vint8m1_t vin1;
    vint8m2_t vin2;
    vint32m8_t vout;
    for (; (vl = __riscv_vsetvl_e8m1(data_cnt)) > 0; data_cnt -= vl) {
        vin1 = __riscv_vle8_v_i8m1(pin1, vl);
        vin2 = __riscv_vle8_v_i8m2(pin2, vl);
        vout = __riscv_vle32_v_i32m8(pout, vl);
        vout = __riscv_xl_vqmacc_4x4x4_i32m8(vout, vin1, vin2, vl);
        __riscv_vse32_v_i32m8(pout, vout, vl);
        pin2 += vl;
        pout += vl;
    }
}

int compare_result(int32_t* normal_out, int32_t* vpu_out, int32_t data_cnt)
{
    int i, ret = 0;

    for (i = 0; i < data_cnt; i++) {
```

(continues on next page)

(continued from previous page)

```
        if (normal_out[i] != vpu_out[i]) {
            printf("num %d not match: %d vs %d\r\n", i, normal_out[i], vpu_out[i]);
            ret = -1;
        }
    }
    return ret;
}


int main(void)
{
    int ret = 0;

    int8_t Input_A[MATRIX_SIZE] = {0};
    int8_t Input_B[DATA_CNT] = {0};
    int32_t Dest_C1[DATA_CNT] = {0};
    int32_t Dest_C2[DATA_CNT] = {0};


    printf("1. Arrays Init\r\n");
    // Input_A
    for (int i = 0; i < MATRIX_SIZE; i++) {
        Input_A[i] = i;
    }

    // Input_B, Dest_C1, Dest_C2
    for (int i = 0; i < ARRAY_CNT; i++) {
        for (int j = 0; j < MATRIX_SIZE; j++) {
            Input_B[i * 16 + j] = j * 2;
        }
    }

    printf("TEST xl.vqmacc\r\n");

    printf("2. Do normal matrix multiply-add\r\n");
    normal_case(Input_A, Input_B, Dest_C1, DATA_CNT);

    printf("3. Do vpu matrix multiply-add\r\n");
    vpu_case(Input_A, Input_B, Dest_C2, DATA_CNT);

    printf("4. Compare results: ");
    if (compare_result(Dest_C1, Dest_C2, DATA_CNT) == 0) {
        printf("PASS\r\n");
    } else {
        printf("FAIL\r\n");
        ret = 1;
    }


    return ret;
}
```

4.1.13 Nuclei Custom Xxlvw Extension

Introduction

The RVV 1.0 vector extension is widely used in AI and image processing. However, its basic vector instructions cannot fully meet the demands of wideband wireless signal processing, such as complex-number processing in 4G/5G communications and high-precision fixed-point dynamic scaling. Vector instructions targeting wireless communications therefore need to be added on top of RVV to cover the extensive vector signal-processing requirements of 4G, 5G, and Wi-Fi scenarios.

This instruction set is a complementary extension to RISC-V RVV 1.0, targeting wideband wireless communication signal processing. It is identified by `w` (Wireless) and builds upon the base `V` extension; in the Nuclei toolchain it is provided as the custom extension `xxlvw`.

The instructions added for wireless signal processing mainly include:

- Complex-number format conversion instructions
- Fixed-point dynamic scaling operation instructions
- Complex dynamic scaling operation instructions
- Dynamic scaling reduction operation instructions
- Inter-element operation instructions
- Fast non-linear operation instructions
- Format conversion instructions

In this context, "operations" act on the individual elements of a vector; the trip count follows the LMUL and VL rules of the RVV specification; the operand width is controlled by SEW; whether an element participates in the operation is controlled by vm; and the default values of inactive elements are determined by the mask-agnostic and tail-agnostic mechanisms.

This document describes the support for this extension in the Nuclei toolchain, including how to enable it, the supported instructions, and the usage of the corresponding intrinsics. For the detailed definition of the extension, refer to riscv-vector4wireless-extension[38].

Extension

Using this extension requires the `V` extension and the `xxlvw` extension to be enabled. Note that this extension supports RV32 only.

Example: `-march=rv32imafcv_xxlvw -mabi=ilp32f`

> **Attention:** To use the intrinsics of this extension, the following header file must be included:
>
> ```
> #include <riscv_vector.h>
> ```

[38] https://github.com/gaoshanlee193/riscv-vector4wireless-extension#introduction

Supported Instructions

Complex number format conversion

- vcpack.vv vd, vs2, vs1, vm
- vcunpackr.v vd, vs2, vm
- vcunpacki.v vd, vs2, vm

Fixed-point dynamic scaling operations

- vdsmul.vv/vs vd, vs2, vs1, vm
- vdsmacini.v vs2, vm
- vdsmacini.s rs1, vm
- vdsmacini.i uimm, vm
- vdsmac.vv/vs vs2, vs1, vm
- vdsmaco.vv/vs vd, vs2, vs1, vm
- vlsb.v vd, vs2, vm

Complex dynamic scaling operations

- vconj.v vd, vs2, vm
- vdscmul.vv/vs vd, vs2, vs1, vm
- vdscmulj.vv/vs vd, vs2, vs1, vm
- vdscredsum.v vd, vs2, vm
- vdscmac.vv/vs vs2, vs1, vm
- vdscmacj.vv/vs vs2, vs1, vm
- vdscmaco.vv/vs vd, vs2, vs1, vm
- vdscmacjo.vv/vs vd, vs2, vs1, vm
- vdscmacor.vv/vs vd, vs2, vs1, vm
- vdscmacoi.vv/vs vd, vs2, vs1, vm
- vdscmacjor.vv/vs vd, vs2, vs1, vm
- vdscmacjoi.vv/vs vd, vs2, vs1, vm
- vdscmulr.vv/vs vd, vs2, vs1, vm
- vdscmuli.vv/vs vd, vs2, vs1, vm
- vdscmuljr.vv/vs vd, vs2, vs1, vm
- vdscmulji.vv/vs vd, vs2, vs1, vm

Dynamic scaling reduced operations

- vdsredsum.v vd, vs2, vm
- vdsredsumn.vs vd, vs2, rs1
- vdsredsumn.vi vd, vs2, uimm
- vredmaxi.vv vd, vs2, vs1, vm
- vredmini.vv vd, vs2, vs1, vm

Inter-element operation instructions

- vperm.vi vd, vs2, uimm

- `vfsl.vv vd, vs2, vs1`
- `vfsr.vv vd, vs2, vs1`

Fast non-linear operations

- `vlnlp0.v vnlpr0, vs2`
- `vlnlp1.v vnlpr1, vs2`
- `vnle.vv vd, vs2, vs1, vm`
- `vnle.vs vd, vs2, vs1, vm`
- `vnlm.vv vd, vs2, vs1, vm`
- `vnlm.vs vd, vs2, vs1, vm`

Format conversion instructions

- `vfcvt.b2h.v vd, vs2`
- `vfcvt.b2w.v vd, vs2`
- `vfcvt.h2w.v vd, vs2`
- `vfcvt.p2c.v vd, vs2`
- `vfcvt.h2b.v vd, vs2`
- `vfcvt.w2b.v vd, vs2`
- `vfcvt.w2h.v vd, vs2`
- `vfcvt.c2p.v vd, vs2`

Note: The format conversion instructions convert between different data widths; they are annotated below as `sew = input width → output width`. The actual data types of each intrinsic can also be seen from its function signature.

For detailed information about these instructions, refer to https://github.com/gaoshanlee193/riscv-vector4wireless-extension/blob/Zvw/src/RISCV-Vector-Extension-for-Wireless.adoc

---

Intrinsic Naming Convention

RVV intrinsic naming convention: **Chapter 6** of `v-intrinsic-spec.pdf` in https://github.com/riscv-non-isa/rvv-intrinsic-doc/releases/tag/v1.0-ratified.

Our naming convention follows the above specification and adds the `_xl` prefix.

Nuclei Custom Intrinsics

Note: Example intrinsics are given below for each instruction; for the complete set of intrinsics, derive the names according to the naming convention above.

In the following, `sew` denotes the width of the data types supported by the instruction, and `full_preds` denotes the set of suffix forms supported by the intrinsic. `full` means that all suffix forms are supported —none (no suffix), _m, _tu, _tum, _tumu, and _mu (six forms in total); `none_m_preds[none,_m]` means that only the none and _m forms are supported; `none_tu_preds` means that only the none and _tu forms are supported; `none_preds` means that only the none (unsuffixed) form is supported.

- `vcpack.vv` sew = 32 full_preds

```
vint32m1_t __riscv_xl_vcpack_vv_i32m1(vint32m1_t vs2, vint32m1_t vs1, size_t vl);
vint32m1_t __riscv_xl_vcpack_vv_i32m1_m(vbool32_t vm, vint32m1_t vs2, vint32m1_t␣
↪vs1, size_t vl);
vint32m1_t __riscv_xl_vcpack_vv_i32m1_tu(vint32m1_t vd, vint32m1_t vs2, vint32m1_t␣
↪vs1, size_t vl);
vint32m1_t __riscv_xl_vcpack_vv_i32m1_tum(vbool32_t vm,vint32m1_t vd, vint32m1_t␣
↪vs2, vint32m1_t vs1, size_t vl);
vint32m1_t __riscv_xl_vcpack_vv_i32m1_tumu(vbool32_t vm,vint32m1_t vd, vint32m1_t␣
↪vs2, vint32m1_t vs1, size_t vl);
vint32m1_t __riscv_xl_vcpack_vv_i32m1_mu(vbool32_t vm,vint32m1_t vd, vint32m1_t␣
↪vs2, vint32m1_t vs1, size_t vl);
```

Note: For the intrinsics of the following instructions, only the full set of none (unsuffixed) intrinsics is listed; construct the other suffix forms according to the rules above.

- `vcunpackr.v` sew = 32 full_preds

```
vint32mf2_t __riscv_xl_vcunpackr_v_i32mf2(vint32mf2_t vs2, size_t vl);
vint32m1_t __riscv_xl_vcunpackr_v_i32m1(vint32m1_t vs2, size_t vl);
vint32m2_t __riscv_xl_vcunpackr_v_i32m2(vint32m2_t vs2, size_t vl);
vint32m4_t __riscv_xl_vcunpackr_v_i32m4(vint32m4_t vs2, size_t vl);
vint32m8_t __riscv_xl_vcunpackr_v_i32m8(vint32m8_t vs2, size_t vl);
```

- `vcunpacki.v` sew = 32 full_preds

```
vint32mf2_t __riscv_xl_vcunpacki_v_i32mf2(vint32mf2_t vs2, size_t vl);
vint32m1_t  __riscv_xl_vcunpacki_v_i32m1(vint32m1_t vs2, size_t vl);
vint32m2_t  __riscv_xl_vcunpacki_v_i32m2(vint32m2_t vs2, size_t vl);
vint32m4_t  __riscv_xl_vcunpacki_v_i32m4(vint32m4_t vs2, size_t vl);
vint32m8_t  __riscv_xl_vcunpacki_v_i32m8(vint32m8_t vs2, size_t vl);
```

- `vdsmul.vv` sew = 8/16/32 full_preds

```
vint8mf8_t __riscv_xl_vdsmul_vv_i8mf8(vint8mf8_t vs2, vint8mf8_t vs1, size_t vl);
vint8mf4_t __riscv_xl_vdsmul_vv_i8mf4(vint8mf4_t vs2, vint8mf4_t vs1, size_t vl);
vint8mf2_t __riscv_xl_vdsmul_vv_i8mf2(vint8mf2_t vs2, vint8mf2_t vs1, size_t vl);
vint8m1_t __riscv_xl_vdsmul_vv_i8m1(vint8m1_t vs2, vint8m1_t vs1, size_t vl);
vint8m2_t __riscv_xl_vdsmul_vv_i8m2(vint8m2_t vs2, vint8m2_t vs1, size_t vl);
vint8m4_t __riscv_xl_vdsmul_vv_i8m4(vint8m4_t vs2, vint8m4_t vs1, size_t vl);
vint8m8_t __riscv_xl_vdsmul_vv_i8m8(vint8m8_t vs2, vint8m8_t vs1, size_t vl);
vint16mf4_t __riscv_xl_vdsmul_vv_i16mf4(vint16mf4_t vs2, vint16mf4_t vs1, size_t␣
↪vl);
vint16mf2_t __riscv_xl_vdsmul_vv_i16mf2(vint16mf2_t vs2, vint16mf2_t vs1, size_t␣
↪vl);
vint16m1_t __riscv_xl_vdsmul_vv_i16m1(vint16m1_t vs2, vint16m1_t vs1, size_t vl);
vint16m2_t __riscv_xl_vdsmul_vv_i16m2(vint16m2_t vs2, vint16m2_t vs1, size_t vl);
vint16m4_t __riscv_xl_vdsmul_vv_i16m4(vint16m4_t vs2, vint16m4_t vs1, size_t vl);
vint16m8_t __riscv_xl_vdsmul_vv_i16m8(vint16m8_t vs2, vint16m8_t vs1, size_t vl);
vint32mf2_t __riscv_xl_vdsmul_vv_i32mf2(vint32mf2_t vs2, vint32mf2_t vs1, size_t␣
↪vl);
vint32m1_t __riscv_xl_vdsmul_vv_i32m1(vint32m1_t vs2, vint32m1_t vs1, size_t vl);
vint32m2_t __riscv_xl_vdsmul_vv_i32m2(vint32m2_t vs2, vint32m2_t vs1, size_t vl);
vint32m4_t __riscv_xl_vdsmul_vv_i32m4(vint32m4_t vs2, vint32m4_t vs1, size_t vl);
vint32m8_t __riscv_xl_vdsmul_vv_i32m8(vint32m8_t vs2, vint32m8_t vs1, size_t vl);
```

- `vdsmul.vs` sew = 8/16/32 full_preds

```
vint8mf8_t __riscv_xl_vdsmul_vs_i8mf8_i8mf8(vint8mf8_t vs2, vint8mf8_t vs1, size_t␣
↪vl);
vint8mf4_t __riscv_xl_vdsmul_vs_i8mf4_i8mf4(vint8mf4_t vs2, vint8mf4_t vs1, size_t␣
```

(continues on next page)

(continued from previous page)

```
↪vl);
vint8mf2_t __riscv_xl_vdsmul_vs_i8mf2_i8mf2(vint8mf2_t vs2, vint8mf2_t vs1, size_t␣
↪vl);
vint8m1_t __riscv_xl_vdsmul_vs_i8m1_i8m1(vint8m1_t vs2, vint8m1_t vs1, size_t vl);
vint8m2_t __riscv_xl_vdsmul_vs_i8m2_i8m2(vint8m2_t vs2, vint8m2_t vs1, size_t vl);
vint8m4_t __riscv_xl_vdsmul_vs_i8m4_i8m4(vint8m4_t vs2, vint8m4_t vs1, size_t vl);
vint8m8_t __riscv_xl_vdsmul_vs_i8m8_i8m8(vint8m8_t vs2, vint8m8_t vs1, size_t vl);
vint16mf4_t __riscv_xl_vdsmul_vs_i16mf4_i16mf4(vint16mf4_t vs2, vint16mf4_t vs1,␣
↪size_t vl);
vint16mf2_t __riscv_xl_vdsmul_vs_i16mf2_i16mf2(vint16mf2_t vs2, vint16mf2_t vs1,␣
↪size_t vl);
vint16m1_t __riscv_xl_vdsmul_vs_i16m1_i16m1(vint16m1_t vs2, vint16m1_t vs1, size_t␣
↪vl);
vint16m2_t __riscv_xl_vdsmul_vs_i16m2_i16m2(vint16m2_t vs2, vint16m2_t vs1, size_t␣
↪vl);
vint16m4_t __riscv_xl_vdsmul_vs_i16m4_i16m4(vint16m4_t vs2, vint16m4_t vs1, size_t␣
↪vl);
vint16m8_t __riscv_xl_vdsmul_vs_i16m8_i16m8(vint16m8_t vs2, vint16m8_t vs1, size_t␣
↪vl);
vint32mf2_t __riscv_xl_vdsmul_vs_i32mf2_i32mf2(vint32mf2_t vs2, vint32mf2_t vs1,␣
↪size_t vl);
vint32m1_t __riscv_xl_vdsmul_vs_i32m1_i32m1(vint32m1_t vs2, vint32m1_t vs1, size_t␣
↪vl);
vint32m2_t __riscv_xl_vdsmul_vs_i32m2_i32m2(vint32m2_t vs2, vint32m2_t vs1, size_t␣
↪vl);
vint32m4_t __riscv_xl_vdsmul_vs_i32m4_i32m4(vint32m4_t vs2, vint32m4_t vs1, size_t␣
↪vl);
vint32m8_t __riscv_xl_vdsmul_vs_i32m8_i32m8(vint32m8_t vs2, vint32m8_t vs1, size_t␣
↪vl);
```

- `vdsmacini.v` sew = 8/16/32 none_m_preds[none,_m]

```
vint8mf8_t __riscv_xl_vdsmacini_v_i8mf8(vint8mf8_t vs2, size_t vl);
vint8mf4_t __riscv_xl_vdsmacini_v_i8mf4(vint8mf4_t vs2, size_t vl);
vint8mf2_t __riscv_xl_vdsmacini_v_i8mf2(vint8mf2_t vs2, size_t vl);
vint8m1_t __riscv_xl_vdsmacini_v_i8m1(vint8m1_t vs2, size_t vl);
vint8m2_t __riscv_xl_vdsmacini_v_i8m2(vint8m2_t vs2, size_t vl);
vint8m4_t __riscv_xl_vdsmacini_v_i8m4(vint8m4_t vs2, size_t vl);
vint8m8_t __riscv_xl_vdsmacini_v_i8m8(vint8m8_t vs2, size_t vl);
vint16mf4_t __riscv_xl_vdsmacini_v_i16mf4(vint16mf4_t vs2, size_t vl);
vint16mf2_t __riscv_xl_vdsmacini_v_i16mf2(vint16mf2_t vs2, size_t vl);
vint16m1_t __riscv_xl_vdsmacini_v_i16m1(vint16m1_t vs2, size_t vl);
vint16m2_t __riscv_xl_vdsmacini_v_i16m2(vint16m2_t vs2, size_t vl);
vint16m4_t __riscv_xl_vdsmacini_v_i16m4(vint16m4_t vs2, size_t vl);
vint16m8_t __riscv_xl_vdsmacini_v_i16m8(vint16m8_t vs2, size_t vl);
vint32mf2_t __riscv_xl_vdsmacini_v_i32mf2(vint32mf2_t vs2, size_t vl);
vint32m1_t  __riscv_xl_vdsmacini_v_i32m1(vint32m1_t vs2, size_t vl);
vint32m2_t  __riscv_xl_vdsmacini_v_i32m2(vint32m2_t vs2, size_t vl);
vint32m4_t  __riscv_xl_vdsmacini_v_i32m4(vint32m4_t vs2, size_t vl);
vint32m8_t  __riscv_xl_vdsmacini_v_i32m8(vint32m8_t vs2, size_t vl);
```

- `vdsmacini.s` sew = 8/16/32 none_m_preds[none,_m]

```
vint8mf8_t __riscv_xl_vdsmacini_x_i8mf8(int8_t rs1, size_t vl);
vint8mf4_t __riscv_xl_vdsmacini_x_i8mf4(int8_t rs1, size_t vl);
vint8mf2_t __riscv_xl_vdsmacini_x_i8mf2(int8_t rs1, size_t vl);
vint8m1_t __riscv_xl_vdsmacini_x_i8m1(int8_t rs1, size_t vl);
vint8m2_t __riscv_xl_vdsmacini_x_i8m2(int8_t rs1, size_t vl);
vint8m4_t __riscv_xl_vdsmacini_x_i8m4(int8_t rs1, size_t vl);
vint8m8_t __riscv_xl_vdsmacini_x_i8m8(int8_t rs1, size_t vl);
vint16mf4_t __riscv_xl_vdsmacini_x_i16mf4(int16_t rs1, size_t vl);
```

(continues on next page)

(continued from previous page)

```
vint16mf2_t __riscv_xl_vdsmacini_x_i16mf2(int16_t rs1, size_t vl);
vint16m1_t __riscv_xl_vdsmacini_x_i16m1(int16_t rs1, size_t vl);
vint16m2_t __riscv_xl_vdsmacini_x_i16m2(int16_t rs1, size_t vl);
vint16m4_t __riscv_xl_vdsmacini_x_i16m4(int16_t rs1, size_t vl);
vint16m8_t __riscv_xl_vdsmacini_x_i16m8(int16_t rs1, size_t vl);
vint32mf2_t __riscv_xl_vdsmacini_x_i32mf2(int32_t rs1, size_t vl);
vint32m1_t __riscv_xl_vdsmacini_x_i32m1(int32_t rs1, size_t vl);
vint32m2_t __riscv_xl_vdsmacini_x_i32m2(int32_t rs1, size_t vl);
vint32m4_t __riscv_xl_vdsmacini_x_i32m4(int32_t rs1, size_t vl);
vint32m8_t __riscv_xl_vdsmacini_x_i32m8(int32_t rs1, size_t vl);
```

Note: 2025.10 release:

Although the `vdsmacini.v` and `vdsmacini.s` instructions have no vd operand, the corresponding intrinsics return a value that must not be ignored.

At present, the intrinsics of all instructions without a vd operand return a value.

If the return value is not used by subsequent code, it must be declared `volatile` to prevent the optimizer from removing it at -O2 and higher optimization levels.

2026.10 release:

For instructions without a vd operand, the intrinsic return value may now be ignored (and used when needed); declaring it `volatile` is no longer required. See *Examples* (page 462) for the usage.

- `vdsmac.vv/vs` sew = 8/16/32 none_m_preds[none,_m]

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdsmac`.

- `vdsmaco.vv/vs` sew = 8/16/32 full_preds

Tip: As above, simply replace `vdsmul` with `vdsmaco`.

- `vlsb.v` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmacini.v`; simply replace `vdsmacini` with `vlsb`.

- `vconj.v` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmacini.v`; simply replace `vdsmacini` with `vconj`.

- `vdscmul.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmul`.

- `vdscmulj.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmulj`.

- `vdscredsum.v` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmacini.v`; simply replace `vdsmacini` with `vdscredsum`.

- `vdscmac.vv/vs` sew = 8/16/32 none_m_preds[none,_m]

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmac`.

- `vdscmacj.vv/vs` sew = 8/16/32 none_m_preds[none,_m]

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmacj`.

- `vdscmaco.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmaco`.

- `vdscmacjo.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmacjo`.

- `vdscmacor.vv` sew = 32 full_preds

```
vint32mf2_t __riscv_xl_vdscmacor_vv_i32mf2(vint32mf2_t vs2, vint32mf2_t vs1, size_
↪t vl);
vint32m1_t __riscv_xl_vdscmacor_vv_i32m1(vint32m1_t vs2, vint32m1_t vs1, size_t␣
↪vl);
vint32m2_t __riscv_xl_vdscmacor_vv_i32m2(vint32m2_t vs2, vint32m2_t vs1, size_t␣
↪vl);
vint32m4_t __riscv_xl_vdscmacor_vv_i32m4(vint32m4_t vs2, vint32m4_t vs1, size_t␣
↪vl);
vint32m8_t __riscv_xl_vdscmacor_vv_i32m8(vint32m8_t vs2, vint32m8_t vs1, size_t␣
↪vl);
```

- `vdscmacor.vs` sew = 32 full_preds

```
vint32mf2_t __riscv_xl_vdscmacor_vs_i32mf2_i32mf2(vint32mf2_t vs2, vint32mf2_t vs1,
↪ size_t vl);
vint32m1_t __riscv_xl_vdscmacor_vs_i32m1_i32m1(vint32m1_t vs2, vint32m1_t vs1,␣
↪size_t vl);
vint32m2_t __riscv_xl_vdscmacor_vs_i32m2_i32m2(vint32m2_t vs2, vint32m2_t vs1,␣
↪size_t vl);
vint32m4_t __riscv_xl_vdscmacor_vs_i32m4_i32m4(vint32m4_t vs2, vint32m4_t vs1,␣
↪size_t vl);
vint32m8_t __riscv_xl_vdscmacor_vs_i32m8_i32m8(vint32m8_t vs2, vint32m8_t vs1,␣
↪size_t vl);
```

- `vdscmacoi.vv/vs` sew = 32 full_preds

Tip: See the intrinsic names listed under `vdscmacor.vv/vs`; simply replace `vdscmacor` with `vdscmacoi`.

- `vdscmacjor.vv/vs` sew = 32 full_preds

Tip: See the intrinsic names listed under `vdscmacor.vv/vs`; simply replace `vdscmacor` with `vdscmacjor`.

- `vdscmacjoi.vv/vs` sew = 32 full_preds

Tip: See the intrinsic names listed under `vdscmacor.vv/vs`; simply replace `vdscmacor` with `vdscmacjoi`.

- `vdscmulr.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmulr`.

- `vdscmuli.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmuli`.

- `vdscmuljr.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmuljr`.

- `vdscmulji.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vdscmulji`.

- `vdsredsum.v` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmacini.v`; simply replace `vdsmacini` with `vdsredsum`.

- `vdsredsumn.vs vd, vs2, rs1` sew = 8/16/32 none_tu_preds

```
vint8mf8_t __riscv_xl_vdsredsumn_vx_i8mf8(vint8mf8_t vs2, int8_t rs1, size_t vl);
vint8mf4_t __riscv_xl_vdsredsumn_vx_i8mf4(vint8mf4_t vs2, int8_t rs1, size_t vl);
vint8mf2_t __riscv_xl_vdsredsumn_vx_i8mf2(vint8mf2_t vs2, int8_t rs1, size_t vl);
vint8m1_t __riscv_xl_vdsredsumn_vx_i8m1(vint8m1_t vs2, int8_t rs1, size_t vl);
vint8m2_t __riscv_xl_vdsredsumn_vx_i8m2(vint8m2_t vs2, int8_t rs1, size_t vl);
vint8m4_t __riscv_xl_vdsredsumn_vx_i8m4(vint8m4_t vs2, int8_t rs1, size_t vl);
vint8m8_t __riscv_xl_vdsredsumn_vx_i8m8(vint8m8_t vs2, int8_t rs1, size_t vl);
vint16mf4_t __riscv_xl_vdsredsumn_vx_i16mf4(vint16mf4_t vs2, int16_t rs1, size_t␣
↪vl);
vint16mf2_t __riscv_xl_vdsredsumn_vx_i16mf2(vint16mf2_t vs2, int16_t rs1, size_t␣
↪vl);
vint16m1_t __riscv_xl_vdsredsumn_vx_i16m1(vint16m1_t vs2, int16_t rs1, size_t vl);
vint16m2_t __riscv_xl_vdsredsumn_vx_i16m2(vint16m2_t vs2, int16_t rs1, size_t vl);
vint16m4_t __riscv_xl_vdsredsumn_vx_i16m4(vint16m4_t vs2, int16_t rs1, size_t vl);
vint16m8_t __riscv_xl_vdsredsumn_vx_i16m8(vint16m8_t vs2, int16_t rs1, size_t vl);
vint32mf2_t __riscv_xl_vdsredsumn_vx_i32mf2(vint32mf2_t vs2, int32_t rs1, size_t␣
↪vl);
vint32m1_t __riscv_xl_vdsredsumn_vx_i32m1(vint32m1_t vs2, int32_t rs1, size_t vl);
vint32m2_t __riscv_xl_vdsredsumn_vx_i32m2(vint32m2_t vs2, int32_t rs1, size_t vl);
vint32m4_t __riscv_xl_vdsredsumn_vx_i32m4(vint32m4_t vs2, int32_t rs1, size_t vl);
vint32m8_t __riscv_xl_vdsredsumn_vx_i32m8(vint32m8_t vs2, int32_t rs1, size_t vl);
```

- `vdsredsumn.vi vd, vs2, uimm` sew = 8/16/32 none_tu_preds

Tip: See the intrinsic names listed under `vdsredsumn.vs`; simply replace `_vx` with `_vi`. For the `vdsredsumn.vs/vi` instructions, the rs1 and uimm values must be integers in the range [1, 2, 3, 4].

- `vredmaxi.vv` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv`; simply replace `vdsmul` with `vredmaxi`.

- `vredmini.vv` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv`; simply replace `vdsmul` with `vredmini`.

- `vperm.vi` sew = 8/16/32 none_tu_preds

```
vint8mf8_t __riscv_xl_vperm_vi_i8mf8(vint8mf8_t vs2, int8_t rs1, size_t vl);
vint8mf4_t __riscv_xl_vperm_vi_i8mf4(vint8mf4_t vs2, int8_t rs1, size_t vl);
vint8mf2_t __riscv_xl_vperm_vi_i8mf2(vint8mf2_t vs2, int8_t rs1, size_t vl);
vint8m1_t __riscv_xl_vperm_vi_i8m1(vint8m1_t vs2, int8_t rs1, size_t vl);
vint8m2_t __riscv_xl_vperm_vi_i8m2(vint8m2_t vs2, int8_t rs1, size_t vl);
vint8m4_t __riscv_xl_vperm_vi_i8m4(vint8m4_t vs2, int8_t rs1, size_t vl);
vint8m8_t __riscv_xl_vperm_vi_i8m8(vint8m8_t vs2, int8_t rs1, size_t vl);
vint16mf4_t __riscv_xl_vperm_vi_i16mf4(vint16mf4_t vs2, int16_t rs1, size_t vl);
vint16mf2_t __riscv_xl_vperm_vi_i16mf2(vint16mf2_t vs2, int16_t rs1, size_t vl);
vint16m1_t __riscv_xl_vperm_vi_i16m1(vint16m1_t vs2, int16_t rs1, size_t vl);
vint16m2_t __riscv_xl_vperm_vi_i16m2(vint16m2_t vs2, int16_t rs1, size_t vl);
vint16m4_t __riscv_xl_vperm_vi_i16m4(vint16m4_t vs2, int16_t rs1, size_t vl);
vint16m8_t __riscv_xl_vperm_vi_i16m8(vint16m8_t vs2, int16_t rs1, size_t vl);
vint32mf2_t __riscv_xl_vperm_vi_i32mf2(vint32mf2_t vs2, int32_t rs1, size_t vl);
vint32m1_t __riscv_xl_vperm_vi_i32m1(vint32m1_t vs2, int32_t rs1, size_t vl);
vint32m2_t __riscv_xl_vperm_vi_i32m2(vint32m2_t vs2, int32_t rs1, size_t vl);
vint32m4_t __riscv_xl_vperm_vi_i32m4(vint32m4_t vs2, int32_t rs1, size_t vl);
vint32m8_t __riscv_xl_vperm_vi_i32m8(vint32m8_t vs2, int32_t rs1, size_t vl);
```

- `vfsl.vv` sew = 8/16/32 none_tu_preds

Tip: See the intrinsic names listed under `vdsmul.vv`; simply replace `vdsmul` with `vfsl`.

- `vfsr.vv` sew = 8/16/32 none_tu_preds

Tip: See the intrinsic names listed under `vdsmul.vv`; simply replace `vdsmul` with `vfsr`.

- `vlnlp0.v/vlnlp1.v` sew = 8/16/32 none_preds

  The intrinsics of these instructions must satisfy the following relation:

  LMUL = 8 when VLEN = 128

  LMUL = 4 when VLEN = 256

  LMUL = 2 when VLEN = 512

  LMUL = 1 when VLEN = 1024

  The compiler sets the VLEN value through `-march=*_zvl${vlen}b`, where `vlen` may be {128, 256, 512, 1024, ⋯}; if it is not specified, the default is 128.

- Intrinsics available with `_zvl1024b` (VLEN = 1024):

```
vint8m1_t __riscv_xl_vlnlp0_v_i8m1(vint8m1_t vs, size_t vl);

vint16m1_t __riscv_xl_vlnlp0_v_i16m1(vint16m1_t vs, size_t vl);

vint32m1_t __riscv_xl_vlnlp0_v_i32m1(vint32m1_t vs, size_t vl);

vint8m1_t __riscv_xl_vlnlp1_v_i8m1(vint8m1_t vs, size_t vl);
```

(continues on next page)

(continued from previous page)

```
vint16m1_t __riscv_xl_vlnlp1_v_i16m1(vint16m1_t vs, size_t vl);

vint32m1_t __riscv_xl_vlnlp1_v_i32m1(vint32m1_t vs, size_t vl);
```

- Intrinsics available with `_zvl512b` (VLEN = 512):

```
vint8m2_t __riscv_xl_vlnlp0_v_i8m2(vint8m2_t vs, size_t vl);

vint16m2_t __riscv_xl_vlnlp0_v_i16m2(vint16m2_t vs, size_t vl);

vint32m2_t __riscv_xl_vlnlp0_v_i32m2(vint32m2_t vs, size_t vl);

vint8m2_t __riscv_xl_vlnlp1_v_i8m2(vint8m2_t vs, size_t vl);

vint16m2_t __riscv_xl_vlnlp1_v_i16m2(vint16m2_t vs, size_t vl);

vint32m2_t __riscv_xl_vlnlp1_v_i32m2(vint32m2_t vs, size_t vl);
```

- Intrinsics available with `_zvl256b` (VLEN = 256):

```
vint8m4_t __riscv_xl_vlnlp0_v_i8m4(vint8m4_t vs, size_t vl);

vint16m4_t __riscv_xl_vlnlp0_v_i16m4(vint16m4_t vs, size_t vl);

vint32m4_t __riscv_xl_vlnlp0_v_i32m4(vint32m4_t vs, size_t vl);

vint8m4_t __riscv_xl_vlnlp1_v_i8m4(vint8m4_t vs, size_t vl);

vint16m4_t __riscv_xl_vlnlp1_v_i16m4(vint16m4_t vs, size_t vl);

vint32m4_t __riscv_xl_vlnlp1_v_i32m4(vint32m4_t vs, size_t vl);
```

- Intrinsics available with `_zvl128b` (VLEN = 128):

```
vint8m8_t __riscv_xl_vlnlp0_v_i8m8(vint8m8_t vs, size_t vl);

vint16m8_t __riscv_xl_vlnlp0_v_i16m8(vint16m8_t vs, size_t vl);

vint32m8_t __riscv_xl_vlnlp0_v_i32m8(vint32m8_t vs, size_t vl);

vint8m8_t __riscv_xl_vlnlp1_v_i8m8(vint8m8_t vs, size_t vl);

vint16m8_t __riscv_xl_vlnlp1_v_i16m8(vint16m8_t vs, size_t vl);

vint32m8_t __riscv_xl_vlnlp1_v_i32m8(vint32m8_t vs, size_t vl);
```

Tip: When using the above intrinsics, you may encounter an `unrecognizable insn` error similar to the following:

```
vlnlp_m1.c: In function 'test_vlnlp0_v_i8m1':
vlnlp_m1.c:8:1: error: unrecognizable insn:
    8 | }
    | ^
(insn 7 4 11 2 (set (reg:RVVM1QI 134 [ <retval> ])
        (if_then_else:RVVM1QI (unspec:RVVMF8BI [
                    (const_vector:RVVMF8BI repeat [
                            (const_int 1 [0x1])
                        ])
                    (reg/v:SI 136 [ vl ])
```

(continues on next page)

(continued from previous page)

```
                    (const_int 2 [0x2]) repeated x2
                    (const_int 0 [0])
                    (reg:SI 66 vl)
                    (reg:SI 67 vtype)
                ] UNSPEC_VPREDICATE)
            (unspec:RVVM1QI [
                    (reg/v:RVVM1QI 135 [ vs ])
                ] UNSPEC_VLNLP0)
            (unspec:RVVM1QI [
                    (reg:SI 0 zero)
                ] UNSPEC_VUNDEF))) "vlnlp_m1.c":7:10 -1
    (nil))
during RTL pass: vregs
vlnlp_m1.c:8:1: internal compiler error: in extract_insn, at recog.cc:2812
0x7f8636076082 __libc_start_main
        ../csu/libc-start.c:308
Please submit a full bug report, with preprocessed source (by using -freport-bug).
Please include the complete backtrace with any bug report.
See <https://gcc.gnu.org/bugs/> for instructions.
```

This is usually caused by a mismatch between VLEN and the LMUL used by the intrinsic.

- `vnle.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vnle`.

- `vnlm.vv/vs` sew = 8/16/32 full_preds

Tip: See the intrinsic names listed under `vdsmul.vv/vs`; simply replace `vdsmul` with `vnlm`.

- `vfcvt.b2h.v` sew = 8→16 none_tu_preds

```
vint16mf4_t __riscv_xl_vfcvt_b2h_v_i16mf4(vint8mf8_t vs2, size_t vl);
vint16mf2_t __riscv_xl_vfcvt_b2h_v_i16mf2(vint8mf4_t vs2, size_t vl);
vint16m1_t __riscv_xl_vfcvt_b2h_v_i16m1(vint8mf2_t vs2, size_t vl);
vint16m2_t __riscv_xl_vfcvt_b2h_v_i16m2(vint8m1_t vs2, size_t vl);
vint16m4_t __riscv_xl_vfcvt_b2h_v_i16m4(vint8m2_t vs2, size_t vl);
vint16m8_t __riscv_xl_vfcvt_b2h_v_i16m8(vint8m4_t vs2, size_t vl);
```

- `vfcvt.b2w.v` sew = 8→32 none_tu_preds

```
vint32mf2_t __riscv_xl_vfcvt_b2w_v_i32mf2(vint8mf8_t vs2, size_t vl);
vint32m1_t __riscv_xl_vfcvt_b2w_v_i32m1(vint8mf4_t vs2, size_t vl);
vint32m2_t __riscv_xl_vfcvt_b2w_v_i32m2(vint8mf2_t vs2, size_t vl);
vint32m4_t __riscv_xl_vfcvt_b2w_v_i32m4(vint8m1_t vs2, size_t vl);
vint32m8_t __riscv_xl_vfcvt_b2w_v_i32m8(vint8m2_t vs2, size_t vl);
```

- `vfcvt.h2w.v` sew = 16→32 none_tu_preds

```
vint32mf2_t __riscv_xl_vfcvt_h2w_v_i32mf2(vint16mf4_t vs2, size_t vl);
vint32m1_t __riscv_xl_vfcvt_h2w_v_i32m1(vint16mf2_t vs2, size_t vl);
vint32m2_t __riscv_xl_vfcvt_h2w_v_i32m2(vint16m1_t vs2, size_t vl);
vint32m4_t __riscv_xl_vfcvt_h2w_v_i32m4(vint16m2_t vs2, size_t vl);
vint32m8_t __riscv_xl_vfcvt_h2w_v_i32m8(vint16m4_t vs2, size_t vl);
```

- `vfcvt.p2c.v` sew = 16→32 none_tu_preds

```
vint32mf2_t __riscv_xl_vfcvt_p2c_v_i32mf2(vint16mf4_t vs2, size_t vl);
vint32m1_t __riscv_xl_vfcvt_p2c_v_i32m1(vint16mf2_t vs2, size_t vl);
vint32m2_t __riscv_xl_vfcvt_p2c_v_i32m2(vint16m1_t vs2, size_t vl);
vint32m4_t __riscv_xl_vfcvt_p2c_v_i32m4(vint16m2_t vs2, size_t vl);
vint32m8_t __riscv_xl_vfcvt_p2c_v_i32m8(vint16m4_t vs2, size_t vl);
```

- `vfcvt.h2b.v` sew = 16→8 none_tu_preds

```
vint8mf8_t __riscv_xl_vfcvt_h2b_v_i8mf8(vint16mf4_t vs2, size_t vl);
vint8mf4_t __riscv_xl_vfcvt_h2b_v_i8mf4(vint16mf2_t vs2, size_t vl);
vint8mf2_t __riscv_xl_vfcvt_h2b_v_i8mf2(vint16m1_t vs2, size_t vl);
vint8m1_t __riscv_xl_vfcvt_h2b_v_i8m1(vint16m2_t vs2, size_t vl);
vint8m2_t __riscv_xl_vfcvt_h2b_v_i8m2(vint16m4_t vs2, size_t vl);
vint8m4_t __riscv_xl_vfcvt_h2b_v_i8m4(vint16m8_t vs2, size_t vl);
```

- `vfcvt.w2b.v` sew = 32→8 none_tu_preds

```
vint8mf8_t __riscv_xl_vfcvt_w2b_v_i8mf8(vint32mf2_t vs2, size_t vl);
vint8mf4_t __riscv_xl_vfcvt_w2b_v_i8mf4(vint32m1_t vs2, size_t vl);
vint8mf2_t __riscv_xl_vfcvt_w2b_v_i8mf2(vint32m2_t vs2, size_t vl);
vint8m1_t __riscv_xl_vfcvt_w2b_v_i8m1(vint32m4_t vs2, size_t vl);
vint8m2_t __riscv_xl_vfcvt_w2b_v_i8m2(vint32m8_t vs2, size_t vl);
```

- `vfcvt.w2h.v` sew = 32→16 none_tu_preds

```
vint16mf4_t __riscv_xl_vfcvt_w2h_v_i16mf4(vint32mf2_t vs2, size_t vl);
vint16mf2_t __riscv_xl_vfcvt_w2h_v_i16mf2(vint32m1_t vs2, size_t vl);
vint16m1_t __riscv_xl_vfcvt_w2h_v_i16m1(vint32m2_t vs2, size_t vl);
vint16m2_t __riscv_xl_vfcvt_w2h_v_i16m2(vint32m4_t vs2, size_t vl);
vint16m4_t __riscv_xl_vfcvt_w2h_v_i16m4(vint32m8_t vs2, size_t vl);
```

- `vfcvt.c2p.v` sew = 32→16 none_tu_preds

```
vint16mf4_t __riscv_xl_vfcvt_c2p_v_i16mf4(vint32mf2_t vs2, size_t vl);
vint16mf2_t __riscv_xl_vfcvt_c2p_v_i16mf2(vint32m1_t vs2, size_t vl);
vint16m1_t __riscv_xl_vfcvt_c2p_v_i16m1(vint32m2_t vs2, size_t vl);
vint16m2_t __riscv_xl_vfcvt_c2p_v_i16m2(vint32m4_t vs2, size_t vl);
vint16m4_t __riscv_xl_vfcvt_c2p_v_i16m4(vint32m8_t vs2, size_t vl);
```

Examples

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <riscv_vector.h>

typedef union {
    struct {
        int16_t re;
        int16_t im;
    } sc16;
    int32_t i32;
} cplx_sc16;

cplx_sc16 src1[4];
cplx_sc16 src2[4];
cplx_sc16 dst_cmul[4];
cplx_sc16 dst_mac[4];
cplx_sc16 dst_cmac[4];
```

(continues on next page)

(continued from previous page)

```
void cplx_mul(const cplx_sc16 *src1, const cplx_sc16 *src2, cplx_sc16 *dst, int␣
↪len) {
    for (int i = 0; i < len; ++i) {
        int64_t re = (int64_t)src1[i].sc16.re * src2[i].sc16.re - (int64_t)src1[i].
↪sc16.im * src2[i].sc16.im;
        int64_t im = (int64_t)src1[i].sc16.re * src2[i].sc16.im + (int64_t)src1[i].
↪sc16.im * src2[i].sc16.re;
        dst[i].sc16.re = re >> 15;
        dst[i].sc16.im = im >> 15;
    }
}

int main()
{
    unsigned long vcsr = (0x0 << 8) | (0x1 << 3);
    asm volatile ("csrw 0xf,%0" : : "r"(vcsr));
    asm volatile("csrr %0,0xf" : "=r"(vcsr));
    printf("vcsr = %x\r\n", vcsr);

    memset(&src1[0], 0, sizeof(cplx_sc16) * 4);
    memset(&src2[0], 0, sizeof(cplx_sc16) * 4);
    memset(&dst_cmul[0], 0, sizeof(cplx_sc16) * 4);
    memset(&dst_mac[0], 0, sizeof(cplx_sc16) * 4);
    memset(&dst_cmac[0], 0, sizeof(cplx_sc16) * 4);

    for (int i = 0; i < 4; ++i) {
        src1[i].sc16.re = rand() % 32 - 16;
        src1[i].sc16.im = rand() % 32 - 16;
        src2[i].sc16.re = rand() % 32 - 16;
        src2[i].sc16.im = rand() % 32 - 16;
    }

    size_t vl = 4;

    vint32m1_t vs1 = __riscv_vle32_v_i32m1(&src1[0].i32, vl);
    vint32m1_t vs2 = __riscv_vle32_v_i32m1(&src2[0].i32, vl);
    vint32m1_t vd = __riscv_xl_vdscmul_vv_i32m1(vs2, vs1, vl);
    __riscv_vse32_v_i32m1(&dst_cmul[0].i32, vd, vl);

    // vint32m1_t tmp = __riscv_xl_vdsmacini_x_i32m1(1, vl); // Usage when the␣
↪return value is needed
    __riscv_xl_vdsmacini_x_i32m1(1, vl);

    vd = __riscv_xl_vdscmaco_vv_i32m1(vs2, vs1, vl);
    __riscv_vse32_v_i32m1(&dst_cmac[0].i32, vd, vl);
    return 0;
}
```

4.2 LLVM Toolchain

4.2.1 About LLVM Toolchain

The current llvm toolchain is developed based on the upstream V19.1.7, for extensions we supported, check below:

You can find our llvm source code in https://github.com/riscv-mcu/llvm-project/tree/nuclei/19.x

Note: The LLVM toolchain is implemented based on the upstream version. Many features can be referenced in the official LLVM documentation: https://llvm.org/docs/. Please refer to the corresponding LLVM release version

in the documentation for accurate details. Local documentation is also available under `gcc/share/doc/llvm/docs/html` within the toolchain directory.

4.2.2 Extensions Support

Ratified Extensions

- Basic Extensions

  i, m, a, f, d, c, h(Assembly only), zaamo, zalrsc, 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.

- Zilsd Extensions

  Zilsd, Zclsd

- Smrnmi & Double Trap Extensions(Assembly only, added since 2026.10)

  smrnmi, smdbltrp, ssdbltrp.

- Xxlcz Extensions(Assembly only)

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

- Packed SIMD Extension 0.5.4(assembly only before 2026.10, intrinsic support added since 2026.10)

  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

Note: Fixed in the 2026.10 toolchain: the `r` inline assembly constraint now allocates a register pair for 64-bit operands on rv32, and the new `R` constraint can be used to explicitly allocate a register pair.

Before 2026.10, the xxldsp assembly instructions implemented in LLVM only support single registers, while instructions using register pairs under rv32 behave abnormally. See this issue[39].

- Nuclei custom VPU Extensions

  Xxlvqmacc (page 445)

- Nuclei custom BFloat16 Extensions

  Xxlfbf (page 435), *Xxlvfbf* (page 435)

Experimental Extensions

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

ssnpm, smnpm, smmpm, sspm, supm, ssqosid, zacas, zalasr, zicfilp, zicfiss.

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-`.

4.2.3 General Options

-march=<isa-string>

Generate code for the given RISC-V ISA string. ISA strings must be lower-case, for example `rv64imafdc`, `rv32gcv`, or with extension suffixes such as `rv64imafdcv_xxlcz`. When `-march=` is not specified, the default ISA string of the target triple is used.

-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.

-fno-vectorize

Use `-fno-vectorize` and `-fno-slp-vectorize` to disable LLVM19 automatic vectorization for RVV; by default, automatic vectorization is enabled.

Optimization Option

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

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

[39] https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/34

-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/19.1.0/docs/RISCVUsage.html

4.2.4 Nuclei Custom Options

In addition to the standard RISC-V options above, clang in the Nuclei LLVM toolchain provides the following Nuclei custom options:

Note: The actual functionality of these options is not implemented yet, currently clang only accepts them without reporting errors.

-mautovec-rvv/-mno-autovec-rvv

Controls the RISC-V RVV automatic vectorization. Enabled by default, use `-mno-autovec-rvv` to disable it. This option is added in 2026.10.

-mautovec-dsp/-mno-autovec-dsp

Controls the generation of automatic vectorization of Nuclei DSP instructions. Disabled by default, use `-mautovec-dsp` to enable it. This option is added in 2026.10.

-mautogen-xxlcz/-mno-autogen-xxlcz

Controls the auto-generation of the Nuclei `xxlcz` extension instructions. Disabled by default, use `-mautogen-xxlcz` to enable it. This option is added in 2026.10.

4.2.5 Install and Setup

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

4.2.6 Changelog

Please check *Changelog* (page 467) here.

4.2.7 Checking LLVM Version

To check the basic version information of LLVM, you can use the command `riscv64-unknown-elf-clang -v`.

If you need to report an issue to the developers, please provide:

1. The `gcc/build.txt` file (located in the toolchain directory) for toolchain build details.
2. The `gcc/gitrepo.txt` file to confirm commit IDs of each tool, which helps determine more detailed build information.

4.2.8 Known Issues

Note: This chapter only lists the bugs that are still unfixed at present (or known usage limitations). For the bugs that have already been fixed, and for more bug issues, please refer to https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/18.

Compared with the GCC toolchain, the LLVM toolchain generally has more limitations, and its support for Nuclei custom extensions is less complete (many of them are assembly-only or intrinsic-only). The missing support will be added in later releases.

4.3 Changelog

Note: To check the known issues encountered after the toolchain release, please refer to the content at the following link:

- GNU Known Issues: *Known Issues* (page 433)
- LLVM Known Issues: *Known Issues* (page 467)
- libncrt Known Issues: *Known Issues* (page 795)

4.3.1 Version 2026.10

Note:

- This version is modified based on version 2025.10; all tool components kept the same versions
- Merged important upstream bugfixes, such as the VXRM state restore fix, the constant synthesis codegen fixes, and the Newlib `sqrt/sqrtf` FPU fix
- Added support for new extensions, such as `Zibi` (v0.1), `RNMI` (`Smrnmi` v1.0), `Double Trap` (`Smdbltrp/Ssdbltrp` v1.0), and enhanced the Nuclei custom `Xxlcz` extension support and the `Xxldsp` auto-generation capability

---

- Toolchain components are unchanged from 2025.10: GCC 14.2.1, Binutils 2.44, GDB 16.2, Newlib 4.5.0, LLVM 19.1.7, and GLIBC 2.41.
- Feature: Added `RNMI & Double Trap` extension support (`Smrnmi` v1.0, `Smdbltrp/Ssdbltrp` v1.0) in GCC/LLVM/Binutils/GDB.
- Feature: Added `Zibi` extension support (v0.1) in GCC/Binutils/GDB.
- Feature: Added new option `-mautovec-rvv/-mno-autovec-rvv` to enable (default) or disable RVV auto-vectorization.

- Feature: The Nuclei DSP auto-vectorization (`-mautovec-dsp`) and the Nuclei `Xxlcz` auto-generation (`-mautogen-xxlcz`) are disabled by default now, and Binutils adds the same `-mautogen-xxlcz/-mno-autogen-xxlcz` assembler option. The `-maddibne/-mno-addibne` option is removed, the `xl.addibne` auto-generation is now controlled by `-mautogen-xxlcz`.
- Feature: Added GDB Python support, which can be enabled with `--with-python` when building the toolchain.
- Feature: GDB now supports reading memory variables while the target is running, which allows Nuclei Studio 2026.10 and later versions to view variables without interrupting the target.
- Feature: Updated the Nuclei custom ECLIC V2 CSRs in Binutils/GDB/LLVM.
- Feature: GCC added the new constraint `R` for register even-odd pairs.
- Feature: Interrupt handlers are RVV-free now; no RVV instructions will be generated in them.
- Feature: Improved the implementation of the Nuclei `Xxlvqmacc` extension, and added the `vsetvli` instructions required by the `Xxlvw` extension intrinsic functions.
- Feature: Enhanced the Nuclei `Xxldsp` and `Xxlcz` extension capabilities.
- Feature: Updated `libncrt` with RISC-V big-endian and `Zicfilp` landing pad support, see *libncrt changelog Version 2026.10* (page 794) for details.
- Fixed: The link failure when RVV intrinsics are used in LTO builds (PR target/110812[40]).
- Fixed: VXRM state not restored after calls (PR target/123626[41]).
- Fixed: `get_biv_step_1` ICE for RISC-V (PR target/117506[42]).
- Fixed: The missing left shift in the RISC-V constant synthesis (PR target/117690[43]), which was a regression introduced in the 2025.10 toolchain.
- Fixed: Newlib `sqrt/sqrtf` used FPU when the `F` or `D` extension is present.
- Fixed: The `xl.addibne` scale field error in Binutils/GDB.
- Fixed: The error when compiling pure assembly files with `Xxlcz` in `-march=`, which existed since the 2025.02 toolchain.
- Fixed: The null pointer issue in `riscv_elf_after_check_relocs` in Binutils.
- Fixed: The abort when displaying data and partial instructions in Binutils.
- Fixed: The `objdump -d -l/-S` out of memory issue in Binutils.
- Fixed: The GDB disassemble core dump issue when the section name is NULL.
- Fixed: Zcmt-related issues in Binutils/ GDB, such as adjusting the Zcmt relax pass, fixing the conditions for automatic generation of Zcmt, fixing the Zcmt GDB table jump index, and adding the `--relax-zcmt`/`--no-relax-zcmt` (default) linker options to control the Zcmt table jump relaxation.
- Fixed: The GCC `nuclei-900-series` pipeline model was 1-issue (single-issue) before, and is now corrected to 2-issue (dual-issue).
- LLVM: Added inline assembly support for GPR Pairs (`R`).
- LLVM: Added Nuclei `Xxldsp` intrinsic support and `Xxlvfbf` vector bf16 intrinsic support.
- LLVM: Reworked the Nuclei `-mtune` support, which fixes issue 37[44] and supports the Nuclei 100/200/300/600/900/1000 series.
- LLVM: Fixed the llvm-objdump error and the Nuclei `Xxldsp` disassembler issues.

[40] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=110812
[41] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=123626
[42] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=117506
[43] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=117690
[44] https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/37

- LLVM: Fixed the issue that `objdump -d` wrongly disassembles 32-bit instructions as 16-bit, by adding the RISC-V mapping symbol (`$x`/`$d`) support to llvm-objdump.
- LLVM: Fixed the wrong `_mcount` behavior of the gprof instrumentation (`-pg`): the `selfpc` (return address `ra`) passed to `_mcount` was incorrect, which made the callgraph unable to be generated correctly (upstream issue 121103[45]).

4.3.2 Version 2025.10

Note:

- This version is modified based on version 2025.02; only Newlib and GLIBC received upstream version bumps
- Windows toolchain now changed from Win32 to Win64
- A lot of known issues are fixed in this release

- Updated toolchain components to these versions: GCC 14.2.1, Binutils 2.44, GDB 16.2, Newlib 4.5.0, LLVM 19.1.7, and GLIBC 2.41.
- Feature: Windows Newlib GCC now supports Win32_x64 (64-bit), replacing Win32 (32-bit).
- Feature: Added auto-generation support for the `Zcmt` extension.
- Feature: Initial support for `ECLIC V2` instructions and CSRs added to Binutils and GDB.
- Feature: Added multilib support for `Zfinx`, `Zdinx`, and related extensions.
- Feature: Binutils and GDB now add support for `Virtual Supervisor-level` CSRs.
- Feature: Add missing `fcvt.d.h`/`fcvt.h.d` instructions for `Xxlfbf` extension.
- Feature: Added `Xxlvw` extension support.
- Feature: Add a new `-maddibne`/`-mno-addibne` option to control whether Xxlcz's `xl.addibne` instruction is auto-generated.
- Performance optimization: Aligned Newlib memory/ string routines (`memcpy`/`memset`/`setjmp`/`strcmp`/`strcpy`/`strlen`) to 4-byte boundaries.
- Fixed: Compilation issues with semihosting functions `_times` and `_gettimeofday` in Newlib.
- Fixed: Incorrect inline assembly translation for the `clrov` instruction in `Xxldsp` (Binutils).
- Fixed: Conflicts between the `Xxlvfbf` extension and `Zvfbfmin`/`Zvfbfwma` extensions.
- Fixed: Codegen regression in branch condition optimization, fixed by removing a redundant AND -1 instruction.
- Fixed: Unnecessary `slli`/`srai` instruction generation when using int16 type.
- LLVM: Initial implementation of pipeline support for Nuclei processor `100`/`200`/`300`/`600`/`900`/`1000` series.
- LLVM: Added scalar BFloat16 support for Nuclei processors.
- LLVM: Implemented Nuclei-specific VPU matrix instructions.
- LLVM: Added missing `Virtual Supervisor-level` CSRs (`hedelegh`/`medelegh`) support.
- LLVM: Added Nuclei custom macro `__riscv_dsp`, which can be used instead of `__riscv_xxldsp` in the old version.
- LLVM: Added missing instructions `dkabs32`, `dsma32.u`, `clrov`, `rdov`, `kmada32`, and `smbb32`.
- LLVM: Fixed the issue where the `Xxldsp` extension could not be disassembled correctly.

[45] https://github.com/llvm/llvm-project/issues/121103

4.3.3 Version 2025.02

- Updated toolchain components to these versions: GCC 14.2.1, Binutils 2.44, GDB 16.2, Newlib 4.4.0, LLVM 19.1.7, and GLIBC 2.40.
- Support for the `Zilsd` and `Zclsd` extensions.
- Some Nuclei custom CSR naming has been re-revised and corrected.
- Implemented custom VPU intrinsics for Nuclei RISC-V CPU.
- Nuclei introduced the `bf16` type and support for the `Xxlvfbf` and `Xxlfbf` extensions.
- Added support for 3-issue (`nuclei-1000-3w-series`) and 4-issue (`nuclei-1000-4w-series`) in the Nuclei 1000 series CPUs.
- GCC 14 introduces additional function attribute checks compared to GCC 13. For more details, you can refer to https://gcc.gnu.org/gcc-14/porting_to.html.
- Add the option to automatically generate control for `Xxldsp` with `-mautovec-dsp/-mno-autovec-dsp` for GCC, which is enabled by default.
- The `riscv_vector.h` must be included when using intrinsic type(s) and API(s). The scope of this attribute should not exceed the function body. Meanwhile, to make RVV types and API(s) available for this attribute, including `riscv_vector.h` will not report an error for now if `v` is not present in `-march=`.
- LLVM: Add Zilsd & Zclsd v1.0 assembly support.
- LLVM: Add Nuclei `Xxldsp`/`Xxldspn1x`/`Xxldspn2x`/`Xxldspn3x` extensions assembly support.
- LLVM: Add Nuclei `Xxlcz` extension assembly support.
- LLVM: Add Nuclei `Xxlvqmacc` extension intrinsic support.
- LLVM: Update Nuclei custom CSRs.
- LLVM: Update the multilib list.

4.3.4 Version 2024.06

- Updated toolchain components to these versions: GCC 13.1, Binutils 2.40, GDB 13.2, Newlib 4.3.0, LLVM 17.0.2, and GLIBC 2.38.
- Instead of using single-letter `bkp` to enable these extensions as we did on GCC 10, 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 GCC 13 upgraded from GCC 10.
- Implemented a 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 extension. The official version is `20191213`, but it is set to `2.2` when configuring the Nuclei toolchain. The difference between them is that in the `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=`. For details, see 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 GCC 10, but is `riscv64-unknown-elf-` on GCC 13.
- On GCC 10, RISC-V intrinsic API heads contain `riscv_vector.h`, `riscv_vector_itr.h`, `rvintrin.h`, `rvp_intrinsic.h`, but now only `riscv_vector.h`, `rvp_intrinsic.h`, `riscv_nuclei_xlcz.h` are provided in GCC 13, 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 GCC 13 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 GCC 13.

CHAPTER

FIVE

NUCLEI C RUNTIME LIBRARY

5.1 About Nuclei C Runtime Library

`Nuclei C Runtime Library` is written in standard ANSI C and RISC-V assembly language and can run on RISC-V CPU. Here`s a list summarising the main features of `Nuclei C Runtime Library`:

libncrt is short of `Nuclei C Runtime Library`.

- Clean ISO/ANSI C source code.
- **Fast assembly language floating point support**.
- Conforms to standard runtime ABIs for the RISC-V architectures.
- `Nuclei C Runtime Library` supports RV32 and optimizations for the **B extension and P extension**.
- GCC 10.2 *riscv-nuclei-elf-gcc* to GCC 14 **riscv64-unknown-elf-gcc** for GNU toolchain.
- We also bring `Nuclei C Runtime Library` support for **LLVM toolchain**.
- `Nuclei C Runtime Library` now separated into three parts, *libncrt*, *'heapops'* and *'fileops'* library.
- Previously, if you want to select a variant of `Nuclei C Runtime Library`, you can pass `--specs=libncrt_xxx.specs` command when using gcc, but now it is not enough, you need to link extra *fileops* and *heapops* static libraries during the linking phase.
- For the *'fileops'* static library, you must select one of the three options: *uart*, *semi* or *rtt*, and for the *'heapops'* static library, you must select one of the three options: *basic*, *realtime* or *minimal*.
- All assembly language routines are valid **zicfilp landing pad** targets when the library is built with CFI landing pad enabled, so indirect calls guarded by the **zicfilp** extension can safely land in runtime library code (see *zicfilp landing pad usage* (page 474)).
- `Nuclei C Runtime Library` supports the **RISC-V big-endian**: using the big-endian C library requires the tools prefixed with `riscv64be-unknown-elf-` together with the `-mbig-endian` compile option (see *Big-endian usage* (page 474)).

5.2 Nuclei C Runtime Library for Nuclei Toolchain

5.2.1 Usage

You can use prebuilt libncrt library for both **Nuclei GNU** and **LLVM** toolchain to replace **libgcc(-lgcc)**, **newlib c(-lc/-lc_nano)** and **math(-lm)** library.

If you want to select a small variant of libncrt library, with basic *'heapops'* and uart *'fileops'*, you can use it like below:

For **Nuclei GNU** Toolchain(*riscv64-unknown-elf-gcc*) case:

```
riscv64-unknown-elf-gcc --specs=libncrt_small.specs -Wl,--start-group -lncrt_small␣
↪-lheapops_basic -lfileops_uart -Wl,--end-group
```

For **Nuclei LLVM** Toolchain(*riscv64-unknown-elf-clang*) case:

You can't use *specs=libncrt_small.specs* option, since it is not supported, gcc can also use following command just like clang.

```
riscv64-unknown-elf-clang -nodefaultlibs -isystem=/include/libncrt -Wl,--start-
↪group -lncrt_small -lheapops_basic -lfileops_uart -Wl,--end-group
```

For details about *libncrt*, '*heapops*' and '*fileops*' variant, please read following documentation.

When using this method, please don't link with **-lm/lc_nano/-lgcc/-lc** library to avoid potential linker error and code size increasement. Please also note that `libncrt` requires a few functions to be provided by the application itself, such as `exit` and the basic I/O functions used by the selected '*fileops*' library, see *Quick start* (page 476) for the minimal set of functions to implement.

5.2.2 zicfilp landing pad usage

`libncrt` supports the **zicfilp** control-flow integrity extension: when the runtime library is built with CFI landing pads enabled, every assembly language routine starts with a valid `lpad` instruction, so indirect calls guarded by zicfilp can safely land in runtime library code.

Using it requires a **zicfilp-enabled toolchain**: the standard released toolchain neither supports `-fcf-protection=branch` for RISC-V targets nor ships the CFI variants of the `libncrt`, '*heapops*' and '*fileops*' libraries. On a zicfilp-enabled toolchain, simply add `_zicfilp` to `-march` and compile and link with `-fcf-protection=branch`: the CFI variants of the runtime libraries (the `cfi` multilib directory) are then selected automatically, and there is no separate library variant to select. An example is as follows:

```
riscv64-unknown-elf-gcc -march=rv32ima_zicfilp_zicond_zaamo_zalrsc_zca_zcb_zcmp_
↪zba_zbb_zbs -mabi=ilp32 -fcf-protection=branch --specs=libncrt_small.specs -Wl,--
↪start-group -lncrt_small -lheapops_basic -lfileops_uart -Wl,--end-group
```

5.2.3 Big-endian usage

`libncrt` also supports RISC-V big-endian targets. The big-endian toolchain is **not publicly released**: to use the big-endian C library, you need to use the tools prefixed with `riscv64be-unknown-elf-` together with the `-mbig-endian` compile option. There is no separate library variant to select for big-endian: the byte order used by the runtime library is automatically configured from the byte-order macros predefined by the compiler, so the libraries are used in exactly the same way as on little-endian targets, taking the small variant with basic '*heapops*' and uart '*fileops*' as an example:

```
riscv64be-unknown-elf-gcc -mbig-endian --specs=libncrt_small.specs -Wl,--start-
↪group -lncrt_small -lheapops_basic -lfileops_uart -Wl,--end-group
```

5.2.4 Varieties

Libncrt Library choice

	GCC Specs	Library
Fast	libncrt_fast.specs	libncrt_fast.a
Balanced	libncrt_balanced.specs	libncrt_balanced.a
Small	libncrt_small.specs	libncrt_small.a
Nano	libncrt_nano.specs	libncrt_nano.a
Pico	libncrt_pico.specs	libncrt_pico.a

If you are not using gcc, and want to use libncrt small variant, you can pass **-isystem=/include/libncrt** during compile phase, and **-nodefaultlibs -lncrt_small** during link phase.

Optimization

	Optimization
Fast	Favor speed at the expense of size
Balanced	Balanced
Small	Favor size at the expense of speed
Nano	Favor size at the expense of speed
Pico	Favor size at the expense of speed

Formatted I/O

	int	long	long long	float	double	wide character	input character class	Width and precision specification	stdout formatting buffer (byte)
Fast	yes	yes	yes	yes	yes	yes	yes	yes	64
Balanced	yes	yes	yes	yes	yes	yes	yes	yes	64
Small	yes	yes	yes	yes	yes	yes	yes	yes	64
Nano	yes	yes	yes	no	no	no	no	yes	64
Pico	yes	no	no	no	no	no	no	no	32

Algorithm

	Scaled-integer Algorithm
Fast	Algorithms use C-language floating-point arithmetic.
Balanced	Algorithms use C-language floating-point arithmetic.
Small	Algorithms use C-language floating-point arithmetic.
Nano	Algorithms use C-language floating-point arithmetic.
Pico	IEEE single-precision functions use scaled integer arithmetic if there is a scaled-integer implementation of the function.

Extra fileops and heapops libraries used together with libncrt

	Library Type	Library description
-lfileops_uart (page 476)	fileops	Using a UART for I/O, need to implement at least metal_tty_putc and metal_tty_getc
-lfileops_semi (page 476)	fileops	Using semihosting for I/O
-lfileops_rtt (page 476)	fileops	Using SEGGER RTT for I/O
-lheapops_basic (page 477)	heapops	Using a basic heap implementation
-lheapops_realtime (page 477)	heapops	Using a real-time heap implementation
-lheapops_minimal (page 477)	heapops	Using a minimal heap implementation

5.3 Quick start

There are four fundamental parts you need to implement for a minimal `Nuclei C Runtime Library` evaluation. And you can find the reference implementation in the latest **Nuclei SDK** release which support **riscv64-unknown-elf-gcc (>=gcc13)**. The documentation for advanced features is in *External function interface* (page 790).

5.3.1 exit()

```
void exit(int fd);
```

5.3.2 Basic UART I/O functions (-lfileops_uart)

To simplify this tutorial, here you only need to implement 2 basic I/O functions:

```
int metal_tty_putc(int c); // UART output function
int metal_tty_getc(void);  // UART input function
```

5.3.3 Using Semihosting (-lfileops_semi)

If you want to use the semihosting function, simply link the *'fileops_semi'* static library; there is no need to implement any additional stub functions.

5.3.4 Using RTT (-lfileops_rtt)

To use SEGGER's RTT, you need to use J-LINK(**>=v7.92f**).

You can download the J-LINK installation package from the J-LINK page on the SEGGER official website, then unzip it. Next, use the SEGGER J-LINK GDB Server included in it to ensure that your J-LINK is connected.

Then, you need to compile and link the program with the *'fileops_rtt'* static library, use the *riscv64-unknown-elf-gdb* to debug the program, and please note that you need to use *monitor reset* and *monitor halt* to reset the J-Link before loading the program.

Finally, open the J-LINK RTT Viewer to observe bi-directional communication with the target program. Please note that if nothing is displayed on the terminal, you may need to disconnect and then reconnect the terminal.

5.3.5 Thread-local storage

In the linker script, setup *tdata*, *tbss*, *__tls_base* and *__tls_end*:

```
.tdata           : ALIGN(8)
{
    PROVIDE( __tls_base = . );
    *(.tdata .tdata.* .gnu.linkonce.td.*)
} >RAM AT>DATA_LMA

.tbss (NOLOAD)   : ALIGN(8)
{
    *(.tbss .tbss.* .gnu.linkonce.tb.*)
    *(.tcommon)
    PROVIDE( __tls_end = . );
} >RAM AT>RAM
```

In *startup.S*, load *__tls_base* to tp register:

```
/* Initialize GP, TP and Stack Pointer SP */
.option push
.option norelax
la gp, __global_pointer$
la tp, __tls_base
.option pop
la sp, _sp
```

5.3.6 Heap

Heap implementation classification

libncrt provides three heap implementations which you may choose from:

- Basic Heap (**-lheapops_basic**)

  Basic heap implement a low-overhead best-fit heap where allocation and deallocation have very little internal fragmentation.

- Real-Time Heap (**-lheapops_realtime**)

  Real-Time heap implement a real-time heap where allocation and deallocation have O(1) performance.

- Minimal Heap (**-lheapops_minimal**)

  Minimal heap implement an allocate-only heap where deallocation and reallocation are not implemented.

SEGGER heap API support

Specific *'heapops'* static libraries can be chosen based on the API support. Here are the **SEGGER heap API** support conditions for the three *'heapops'* static libraries:

	heapops_realtime	heapops_basic	heapops_minimal
__SEGGER_RTL_alloc	yes	yes	yes
__SEGGER_RTL_aligned_alloc	yes	no	no
__SEGGER_RTL_realloc	yes	yes	no
__SEGGER_RTL_free	yes	yes	no

Before calling heap-related APIs such as malloc, you need to implement the initialization of the heap as shown in nuclei-sdk init_libncrt_heap()[46] . You can call the function during startup such as in the *startup.S* file.

```
extern void __SEGGER_RTL_init_heap(void *ptr, size_t size);
extern char __heap_start[];
extern char __heap_end[];

void init_libncrt_heap(void)
{
    size_t heapsz = (size_t)__heap_end - (size_t)__heap_start;
    __SEGGER_RTL_init_heap((void *)__heap_start, heapsz);
}
```

In the linker script, setup *__heap_start* and *__heap_end*. You need to align the heap to a 16-byte boundary and reserve *__HEAP_SIZE* bytes for it.

```
.heap (NOLOAD)    : ALIGN(16)
{
    . = ALIGN(16);
    PROVIDE( __heap_start = . );
    . += __HEAP_SIZE;
    . = ALIGN(16);
    PROVIDE( __heap_limit = . );
} >RAM AT>RAM

.stack ORIGIN(RAM) + LENGTH(RAM) - __TOT_STACK_SIZE (NOLOAD) :
{
    . = ALIGN(16);
    PROVIDE( _heap_end = . );
    PROVIDE( __heap_end = . );
    PROVIDE( __StackLimit = . );
    PROVIDE( __StackBottom = . );
    . += __TOT_STACK_SIZE;
    . = ALIGN(16);
    PROVIDE( __StackTop = . );
    PROVIDE( _sp = . );
} >RAM AT>RAM
```

5.4 Runtime support

This section describes how to set up the execution environment for the C library.

5.4.1 Getting to main() and then exit()

Before entering *main()* the execution environment must be set up such that the C standard library will function correctly.

Note: This section does not describe the compiler or linker support for placing code and data into memory, how to configure any RAM, or how to zero memory required for zero-initialized data. For this, please refer to your toolset compiler and linker documentation.

Nor does this section document how to call constructors and destructors in the correct order. Again, refer to your toolset manuals.

---

[46] https://github.com/Nuclei-Software/nuclei-sdk/blob/master/SoC/evalsoc/Common/Source/Stubs/libncrt/stubs.c

At-exit function support

After returning from *main()* or by calling *exit()*, any registered At-exit functions must be called to close down. To do this, call *__SEGGER_RTL_execute_at_exit_fns()* from the runtime startup immediately after the call to *main()*.

5.4.2 Multithreaded protection for the heap

Heap functions (allocation, reallocation, deallocation) can be protected from reentrancy in a multithreaded environment by implementing **lock and unlock** functions. By default, these functions do nothing and memory allocation functions are not protected.

See *__SEGGER_RTL_X_heap_lock()* (page 792) and *__SEGGER_RTL_X_heap_unlock()* (page 792).

5.5 Using customized fileops

If you prefer not to use our provided *fileops_uart* (page 476), *fileops_semi* (page 476) or *fileops_rtt* (page 476) static libraries and would like to implement your own customized *'fileops'*, do not link the *'fileops'* static library. Instead, you should implement all of the following *'fileops'* functions.

```
// Open file. Return NULL if file not opened, !=NULL if opened
__SEGGER_RTL_FILE *__SEGGER_RTL_X_file_open(const char *filename, const char␣
↪*mode) {

}

// Test for file-error condition. Return <0 if stream is closed, ==0 if stream is␣
↪not in error, >0 if stream is in error
int __SEGGER_RTL_X_file_error(__SEGGER_RTL_FILE *stream) {

}

// Test for end-of-file condition. Return <0 if stream is closed, ==0 if stream is␣
↪not at end of file, >0 if stream is at end of file
int __SEGGER_RTL_X_file_end(__SEGGER_RTL_FILE *stream) {

}

// Get file status. Return <0 if stream is not a valid file, >=0 if stream is a␣
↪valid file
int __SEGGER_RTL_X_file_stat(__SEGGER_RTL_FILE *stream) {

}

// Get stream buffer size. Return 1 for unbuffered I/O, nonzero number of␣
↪characters to use for
// buffered I/O
int __SEGGER_RTL_X_file_bufsize(__SEGGER_RTL_FILE *stream) {

}

// Flush unwritten data to file. Return <0 if failure, ==0 if success
int __SEGGER_RTL_X_file_flush(__SEGGER_RTL_FILE *stream) {

}

// Get file position. Return <0 if position not retrieved successfully, ==0 if␣
↪retrieved successfully
int __SEGGER_RTL_X_file_getpos(__SEGGER_RTL_FILE *stream, fpos_t *pos) {
```

(continues on next page)

(continued from previous page)

```
}

// Set file position. Return !=0 if position is not set, ==0 if position is set
int __SEGGER_RTL_X_file_seek(__SEGGER_RTL_FILE *stream, long offset, int whence) {

}

// Clear file-error status.
void __SEGGER_RTL_X_file_clrerr(__SEGGER_RTL_FILE *stream)- {

}

// Close file. Return <0 if stream if already closed, >=0 if stream is closed
int __SEGGER_RTL_X_file_close(__SEGGER_RTL_FILE *stream) {

}

// Read data from file. Return <0 if failure, >=0 if success
int __SEGGER_RTL_X_file_read(__SEGGER_RTL_FILE *stream, char *s, unsigned len) {

}

// Write data to file. Return <0 if failure, >=0 if success
int __SEGGER_RTL_X_file_write(__SEGGER_RTL_FILE *stream, const char *s, unsigned␣
↪len) {

}

// Rename file. Return !=0 if rename failure, ==0 if rename success
int __SEGGER_RTL_X_file_rename(const char *old, const char *new) {

}

// Remove file, Return !=0 if remove failed, ==0 if remove success
int __SEGGER_RTL_X_file_remove(const char *filename) {

}

// Generate name for temporary file. Return !=NULL if pointer to temporary name␣
↪generated, ==NULL if cannot generate a unique temporary name
char *__SEGGER_RTL_X_file_tmpnam(char *s, unsigned max) {

}

// Generate temporary file. Return != NULL if pointer to temporary file, ==NULL if␣
↪cannot generate a unique temporary file
__SEGGER_RTL_FILE *__SEGGER_RTL_X_file_tmpfile(void) {

}

// Push character back to stream. Return <0 if failure, >=0 if success
int __SEGGER_RTL_X_file_unget(__SEGGER_RTL_FILE *stream, int c) {

}
```

You can refer to the example of UART using the complete *'fileops'* API, which includes the functions mentioned above, in the file **libncrt_fileops_reference.c** located in the same directory as this document.

Of course, you still need to additionally implement these two basic I/O functions is essential for a complete UART *'fileops'* implementation:

```
int metal_tty_putc(int c); // UART output function

int metal_tty_getc(void);  // UART input function
```

5.6 C library API

5.6.1 <assert.h>

Assertion functions

Function	Description
assert	Place assertion.

assert

Description

Place assertion.

Definition

```
#define assert(e)    ...
```

Additional information

If NDEBUG is defined as a macro name at the point in the source file where <assert.h> is included, the assert() macro is defined as:

```
#define assert(ignore) ((void)0)    ...
```

If NDEBUG is not defined as a macro name at the point in the source file where <assert.h> is included, the assert() macro expands to a void expression that calls __SEGGER_RTL_X_assert().

When such an assert is executed and e is false, assert() calls the function __SEGGER_RTL_X_assert() with information about the particular call that failed: the text of the argument, the name of the source file, and the source line number. These are the stringized expression and the values of the preprocessing macros __FILE__ and __LINE__.

Notes

The assert() macro is redefined according to the current state of NDEBUG each time that <assert.h> is included.

5.6.2 <complex.h>

`Nuclei C Runtime Library` provides complex math library functions, including all of those required by ISO C99. These functions are implemented to balance performance with correctness. Because producing the correctly rounded result may be prohibitively expensive, these functions are designed to efficiently produce a close approximation to the correctly rounded result. In most cases, the result produced is within +/-1 ulp of the correctly rounded result, though there may be cases where there is greater inaccuracy.

Manipulation functions

Function	Description
cabs() (page 482)	Compute magnitude, double complex.
cabsf() (page 482)	Compute magnitude, float complex.
cabsl() (page 483)	Compute magnitude, long double complex.
carg() (page 483)	Compute phase, double complex.
cargf() (page 483)	Compute phase, float complex.
cargl() (page 484)	Compute phase, long double complex.
cimag() (page 484)	Imaginary part, double complex.
cimagf() (page 484)	Imaginary part, float complex.
cimagl() (page 485)	Imaginary part, long double complex.
creal() (page 485)	Real part, double complex.
crealf() (page 485)	Real part, float complex.
creall() (page 486)	Real part, long double complex.
cproj() (page 486)	Project, double complex.
cprojf() (page 487)	Project, float complex.
cprojl() (page 487)	Project, long double complex.
conj() (page 488)	Conjugate, double complex.
conjf() (page 488)	Conjugate, float complex.
conjl() (page 488)	Conjugate, long double complex.

cabs()

Description

Compute magnitude, double complex.

Prototype

```
double cabs(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

The magnitude of x, |x|.

cabsf()

Description

Compute magnitude, float complex.

Prototype

```
float cabsf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

The magnitude of x, |x|.

cabsl()

Description

Compute magnitude, long double complex.

Prototype

```
long double cabsl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

The magnitude of x, |x|.

carg()

Description

Compute phase, double complex.

Prototype

```
double carg(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute phase of.

Return value

The phase of x.

cargf()

Description

Compute phase, float complex.

Prototype

```
float cargf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute phase of.

Return value

The phase of x.

cargl()

Description

Compute phase, long double complex.

Prototype

```
long double cargl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute phase of.

Return value

The phase of x.

cimag()

Description

Imaginary part, double complex.

Prototype

```
double cimag(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The imaginary part of the complex value.

cimagf()

Description

Imaginary part, float complex.

Prototype

```
float cimagf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The imaginary part of the complex value.

cimagl()

Description

Imaginary part, long double complex.

Prototype

```
long double cimagl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The imaginary part of the complex value.

creal()

Description

Real part, double complex.

Prototype

```
double creal(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The real part of the complex value.

crealf()

Description

Real part, float complex.

Prototype

```
float crealf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The real part of the complex value.

creall()

Description

Real part, long double complex.

Prototype

```
long double creall(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The real part of the complex value.

cproj()

Description

Project, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cproj(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to project.

Return value

The projection of x to the Reimann sphere.

Additional information

x projects to x, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If x has an infinite part, then cproj(x) is be equivalent to:

- INFINITY + I * copysign(0.0, cimag(x))

cprojf()

Description

Project, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cprojf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to project.

Return value

The projection of x to the Reimann sphere.

Additional information

x projects to x, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If x has an infinite part, then cproj(x) is be equivalent to:

- INFINITY + I * copysign(0.0, cimag(x))

cprojl()

Description

Project, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cprojl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to project.

Return value

The projection of x to the Reimann sphere.

Additional information

x projects to x, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If x has an infinite part, then cproj(x) is be equivalent to:

- INFINITY + I * copysignl(0.0, cimagl(x))

conj()

Description

Conjugate, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX conj(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to conjugate.

Return value

The complex conjugate of x.

conjf()

Description

Conjugate, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX conjf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to conjugate.

Return value

The complex conjugate of x.

conjl()

Description

Conjugate, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX conjl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to conjugate.

Return value

The complex conjugate of x.

Trigonometric functions

Function	Description
csin() (page 489)	Compute sine, double complex.
csinf() (page 489)	Compute sine, float complex.
csinl() (page 490)	Compute sine, long double complex.
ccos() (page 490)	Compute cosine, double complex.
ccosf() (page 490)	Compute cosine, float complex.
ccosl() (page 491)	Compute cosine, long double complex.
ctan() (page 491)	Compute tangent, double complex.
ctanf() (page 491)	Compute tangent, float complex.
ctanl() (page 492)	Compute tangent, long double complex.
casin() (page 492)	Compute inverse sine, double complex.
casinf() (page 492)	Compute inverse sine, float complex.
casinl() (page 493)	Compute inverse sine, long double complex.
cacos() (page 493)	Compute inverse cosine, double complex.
cacosf() (page 494)	Compute inverse cosine, float complex.
cacosl() (page 494)	Compute inverse cosine, long double complex.
catan() (page 494)	Compute inverse tangent, double complex.
catanf() (page 495)	Compute inverse tangent, float complex.
catanl() (page 495)	Compute inverse tangent, long double complex.

csin()

Description

Compute sine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX csin(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute sine of.

Return value

The sine of x.

csinf()

Description

Compute sine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX csinf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute sine of.

Return value

The sine of x.

csinl()

Description

Compute sine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX csinl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute sine of.

Return value

The sine of x.

ccos()

Description

Compute cosine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX ccos(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute cosine of.

Return value

The cosine of x.

ccosf()

Description

Compute cosine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ccosf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute cosine of.

Return value

The cosine of x.

ccosl()

Description

Compute cosine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ccosl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute cosine of.

Return value

The cosine of x.

ctan()

Description

Compute tangent, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX ctan(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute tangent of.

Return value

The tangent of x.

ctanf()

Description

Compute tangent, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ctanf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute tangent of.

Return value

The tangent of x.

ctanl()

Description

Compute tangent, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ctanl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute tangent of.

Return value

The tangent of x.

casin()

Description

Compute inverse sine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX casin(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Inverse sine of x.

Notes

casin(z) = -i casinh(i.z)

casinf()

Description

Compute inverse sine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX casinf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Inverse sine of x.

Notes

casin(z) = -i casinh(i.z)

casinl()

Description

Compute inverse sine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX casinl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Inverse sine of x.

Notes

casinl(z) = -i casinhl(i.z)

cacos()

Description

Compute inverse cosine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cacos(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

The inverse cosine of x.

cacosf()

Description

Compute inverse cosine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cacosf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

The inverse cosine of x.

cacosl()

Description

Compute inverse cosine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cacosl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

The inverse cosine of x.

catan()

Description

Compute inverse tangent, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX catan(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Inverse tangent of x.

Notes

catan(z) = -i catanh(i.z)

catanf()

Description

Compute inverse tangent, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX catanf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Inverse tangent of x.

Notes

catan(z) = -i catanh(i.z)

catanl()

Description

Compute inverse tangent, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX catanl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Inverse tangent of x.

Notes

catanl(z) = -i catanhl(i.z)

Hyperbolic functions

Function	Description
csinh() (page 496)	Compute hyperbolic sine, double complex.
csinhf() (page 497)	Compute hyperbolic sine, float complex.
csinhl() (page 498)	Compute hyperbolic sine, long double complex.
ccosh() (page 498)	Compute hyperbolic cosine, double complex.
ccoshf() (page 499)	Compute hyperbolic cosine, float complex.
ccoshl() (page 500)	Compute hyperbolic cosine, long double complex.
ctanh() (page 500)	Compute hyperbolic tangent, double complex.
ctanhf() (page 501)	Compute hyperbolic tangent, float complex.
ctanhl() (page 502)	Compute hyperbolic tangent, long double complex.
casinh() (page 502)	Compute inverse hyperbolic sine, double complex.
casinhf() (page 503)	Compute inverse hyperbolic sine, float complex.
casinhl() (page 504)	Compute inverse hyperbolic sine, long double complex.
cacosh() (page 504)	Compute inverse hyperbolic cosine, double complex.
cacoshf() (page 505)	Compute inverse hyperbolic cosine, float complex.
cacoshl() (page 506)	Compute inverse hyperbolic cosine, long double complex.
catanh() (page 506)	Compute inverse hyperbolic tangent, double complex.
catanhf() (page 507)	Compute inverse hyperbolic tangent, float complex.
catanhl() (page 508)	Compute inverse hyperbolic tangent, long double complex.

csinh()

Description

Compute hyperbolic sine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX csinh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

The hyperbolic sine of x according to the following table:

Argument	csinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	±0 + NaNi, sign of real part unspecified
+0 + NaNi	±0 + NaNi, sign of real part unspecified
a + ∞i	NaN + NaNi, for positive finite a
a + NaNi	NaN + NaNi, for finite nonzero a
+∞ + 0i	+∞ + 0i
+∞ + bi	+∞×cos(b) + +∞×sin(b).i for positive finite b
+∞ + ∞i	±∞ + NaNi, sign of real part unspecified
+∞ + NaNi	±∞ + NaNi, sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- csinh(conj(z)) = conj(csinh(z)).

For arguments with a negative real component, use the equality:

- csinh(-z) = -csinh(z).

csinhf()

Description

Compute hyperbolic sine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX csinhf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

The hyperbolic sine of x according to the following table:

Argument	csinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	±0 + NaNi, sign of real part unspecified
+0 + NaNi	±0 + NaNi, sign of real part unspecified
a + ∞i	NaN + NaNi, for positive finite a
a + NaNi	NaN + NaNi, for finite nonzero a
+∞ + 0i	+∞ + 0i
+∞ + bi	+∞×cos(b) + +∞×sin(b).i for positive finite b
+∞ + ∞i	±∞ + NaNi, sign of real part unspecified
+∞ + NaNi	±∞ + NaNi, sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- csinh(conj(z)) = conj(csinh(z)).

For arguments with a negative real component, use the equality:

- csinh(-z) = -csinh(z).

csinhl()

Description

Compute hyperbolic sine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX csinhl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

The hyperbolic sine of x according to the following table:

Argument	csinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	±0 + NaNi, sign of real part unspecified
+0 + NaNi	±0 + NaNi, sign of real part unspecified
a + ∞i	NaN + NaNi, for positive finite a
a + NaNi	NaN + NaNi, for finite nonzero a
+∞ + 0i	+∞ + 0i
+∞ + bi	+∞×cos(b) + +∞×sin(b).i for positive finite b
+∞ + ∞i	±∞ + NaNi, sign of real part unspecified
+∞ + NaNi	±∞ + NaNi, sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- csinh(conj(z)) = conj(csinh(z)).

For arguments with a negative real component, use the equality:

- csinh(-z) = -csinh(z).

ccosh()

Description

Compute hyperbolic cosine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX ccosh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

The hyperbolic cosine of x according to the following table:

Argument	ccosh(Argument)
+0 + 0i	+1 + 0i
+0 + ∞i	NaN + ±0i, sign of imaginary part unspecified
+0 + NaNi	NaN + ±0i, sign of imaginary part unspecified
a + ∞i	NaN + NaNi, for finite nonzero a
a + NaNi	NaN + NaNi, for finite nonzero a
+∞ + 0i	+∞ + 0i
+∞ + bi	+∞×cos(b) + Inf×sin(b).i for finite nonzero b
+∞ + ∞i	+∞ + NaNi
+∞ + NaNi	+∞ + NaNi
NaN + 0i	NaN + ±0i, sign of imaginary part unspecified
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- ccosh(conj(z)) = conj(ccosh(z)).

ccoshf()

Description

Compute hyperbolic cosine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ccoshf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

The hyperbolic cosine of x according to the following table:

Argument	ccosh(Argument)
+0 + 0i	+1 + 0i
+0 + ∞i	NaN + ±0i, sign of imaginary part unspecified
+0 + NaNi	NaN + ±0i, sign of imaginary part unspecified
a + ∞i	NaN + NaNi, for finite nonzero a
a + NaNi	NaN + NaNi, for finite nonzero a
+∞ + 0i	+∞ + 0i
+∞ + bi	+∞×cos(b) + Inf×sin(b).i for finite nonzero b
+∞ + ∞i	+∞ + NaNi
+∞ + NaNi	+∞ + NaNi
NaN + 0i	NaN + ±0i, sign of imaginary part unspecified
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- ccosh(conj(z)) = conj(ccosh(z)).

ccoshl()

Description

Compute hyperbolic cosine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ccoshl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

The hyperbolic cosine of x according to the following table:

Argument	ccosh(Argument)
+0 + 0i	+1 + 0i
+0 + ∞i	NaN + ±0i, sign of imaginary part unspecified
+0 + NaNi	NaN + ±0i, sign of imaginary part unspecified
a + ∞i	NaN + NaNi, for finite nonzero a
a + NaNi	NaN + NaNi, for finite nonzero a
+∞ + 0i	+∞ + 0i
+∞ + bi	+∞×cos(b) + Inf×sin(b).i for finite nonzero b
+∞ + ∞i	+∞ + NaNi
+∞ + NaNi	+∞ + NaNi
NaN + 0i	NaN + ±0i, sign of imaginary part unspecified
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- ccosh(conj(z)) = conj(ccosh(z)).

ctanh()

Description

Compute hyperbolic tangent, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX ctanh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

The hyperbolic tangent of x according to the following table:

Argument	ctanh(Argument)
+0 + 0i	+0 + 0i
a + ∞i	NaN + NaNi, for finite a
a + NaNi	NaN + NaNi, for finite a
+∞ + bi	+1 + sin(2b)×0i for positive-signed finite b
+∞ + ∞i	+1 + ±0i, sign of imaginary part unspecified
+∞ + NaNi	+1 + ±0i, sign of imaginary part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- ctanh(conj(z)) = conj(ctanh(z)).

For arguments with a negative real component, use the equality:

- ctanh(-z) = -ctanh(z).

ctanhf()

Description

Compute hyperbolic tangent, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ctanhf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

The hyperbolic tangent of x according to the following table:

Argument	ctanh(Argument)
+0 + 0i	+0 + 0i
a + ∞i	NaN + NaNi, for finite a
a + NaNi	NaN + NaNi, for finite a
+∞ + bi	+1 + sin(2b)×0i for positive-signed finite b
+∞ + ∞i	+1 + ±0i, sign of imaginary part unspecified
+∞ + NaNi	+1 + ±0i, sign of imaginary part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- ctanhf(conj(z)) = conj(ctanhf(z)).

For arguments with a negative real component, use the equality:

- ctanhf(-z) = -ctanhf(z).

ctanhl()

Description

Compute hyperbolic tangent, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ctanhl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

The hyperbolic tangent of x according to the following table:

Argument	ctanh(Argument)
+0 + 0i	+0 + 0i
a + ∞i	NaN + NaNi, for finite a
a + NaNi	NaN + NaNi, for finite a
+∞ + bi	+1 + sin(2b)×0i for positive-signed finite b
+∞ + ∞i	+1 + ±0i, sign of imaginary part unspecified
+∞ + NaNi	+1 + ±0i, sign of imaginary part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- ctanh(conj(z)) = conj(ctanh(z)).

For arguments with a negative real component, use the equality:

- ctanh(-z) = -ctanh(z).

casinh()

Description

Compute inverse hyperbolic sine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX casinh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sineof.

Return value

The inverse hyperbolic sine of x according to the following table:

Argument	casinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	+∞ + ½πi
a + NaNi	NaN + NaNi
+∞ + bi	+∞ + 0i, for positive-signed b
+∞ + ∞i	+Pi + 0i
+∞ + NaNi	+∞ + NaNi
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for finite nonzero b
NaN + ∞i	±∞ + NaNi, sign of real part unspecified
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- casinh(conj(z)) = conj(casinh(z)).

For arguments with a negative real component, use the equality:

- casinh(-z) = -casinh(z).

casinhf()

Description

Compute inverse hyperbolic sine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX casinhf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sineof.

Return value

The inverse hyperbolic sine of x according to the following table:

Argument	casinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	+∞ + ½πi
a + NaNi	NaN + NaNi
+∞ + bi	+∞ + 0i, for positive-signed b
+∞ + ∞i	+Pi + 0i
+∞ + NaNi	+∞ + NaNi
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for finite nonzero b
NaN + ∞i	±∞ + NaNi, sign of real part unspecified
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- casinh(conj(z)) = conj(casinh(z)).

For arguments with a negative real component, use the equality:

- casinh(-z) = -casinh(z).

casinhl()

Description

Compute inverse hyperbolic sine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX casinhl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sineof.

Return value

The inverse hyperbolic sine of x according to the following table:

Argument	casinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	+∞ + ½πi
a + NaNi	NaN + NaNi
+∞ + bi	+∞ + 0i, for positive-signed b
+∞ + ∞i	+Pi + 0i
+∞ + NaNi	+∞ + NaNi
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for finite nonzero b
NaN + ∞i	±∞ + NaNi, sign of real part unspecified
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- casinh(conj(z)) = conj(casinh(z)).

For arguments with a negative real component, use the equality:

- casinh(-z) = -casinh(z).

cacosh()

Description

Compute inverse hyperbolic cosine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cacosh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

The inverse hyperbolic cosine of x according to the following table:

Argument	cacosh(Argument)
±0 + 0i	+0 + 0i
a + ∞i	+∞ + ½πi, for finite a
a + NaNi	NaN + NaNi, for finite a
-∞ + bi	+∞ + πi, for positive-signed finite b
+∞ + bi	+∞ + 0i, for positive-signed finite b
-∞ + ∞i	±∞ + ¾πi
+∞ + ∞i	±∞ + ¼πi
±∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + NaNi
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- cacosh(conj(z)) = conj(cacosh(z)).

cacoshf()

Description

Compute inverse hyperbolic cosine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cacoshf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

The inverse hyperbolic cosine of x according to the following table:

Argument	cacosh(Argument)
±0 + 0i	+0 + 0i
a + ∞i	+∞ + ½πi, for finite a
a + NaNi	NaN + NaNi, for finite a
-∞ + bi	+∞ + πi, for positive-signed finite b
+∞ + bi	+∞ + 0i, for positive-signed finite b
-∞ + ∞i	±∞ + ¾πi
+∞ + ∞i	±∞ + ¼πi
±∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + NaNi
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- cacosh(conj(z)) = conj(cacosh(z)).

cacoshl()

Description

Compute inverse hyperbolic cosine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cacoshl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

The inverse hyperbolic cosine of x according to the following table:

Argument	cacosh(Argument)
±0 + 0i	+0 + 0i
a + ∞i	+∞ + ½πi, for finite a
a + NaNi	NaN + NaNi, for finite a
-∞ + bi	+∞ + πi, for positive-signed finite b
+∞ + bi	+∞ + 0i, for positive-signed finite b
-∞ + ∞i	±∞ + ¾πi
+∞ + ∞i	±∞ + ¼πi
±∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + NaNi
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- cacosh(conj(z)) = conj(cacosh(z)).

catanh()

Description

Compute inverse hyperbolic tangent, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX catanh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

The inverse hyperbolic tangent of x according to the following table:

Argument	catanh(Argument)
+0 + 0i	+0 + 0i
+0 + NaNi	+0 + NaNi
+1 + 0i	+∞ + 0i
a + ∞i	+0 + ½πi for positive-signed a
a + NaNi	NaN + NaNi, for nonzero finite a
+∞ + bi	+0 + ½πi for positive-signed b
+∞ + ∞i	+0 + ½πi
+∞ + NaNi	+0 + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- catanh(conj(z)) = conj(catanh(z)).

For arguments with a negative real component, use the equality:

- catanh(-z) = -catanh(z).

catanhf()

Description

Compute inverse hyperbolic tangent, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX catanhf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

The inverse hyperbolic tangent of x according to the following table:

Argument	catanh(Argument)
+0 + 0i	+0 + 0i
+0 + NaNi	+0 + NaNi
+1 + 0i	+∞ + 0i
a + ∞i	+0 + ½πi for positive-signed a
a + NaNi	NaN + NaNi, for nonzero finite a
+∞ + bi	+0 + ½πi for positive-signed b
+∞ + ∞i	+0 + ½πi
+∞ + NaNi	+0 + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- catanh(conj(z)) = conj(catanh(z)).

For arguments with a negative real component, use the equality:

- catanh(-z) = -catanh(z).

catanhl()

Description

Compute inverse hyperbolic tangent, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX catanhl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

The inverse hyperbolic tangent of x according to the following table:

Argument	catanh(Argument)
+0 + 0i	+0 + 0i
+0 + NaNi	+0 + NaNi
+1 + 0i	+∞ + 0i
a + ∞i	+0 + ½πi for positive-signed a
a + NaNi	NaN + NaNi, for nonzero finite a
+∞ + bi	+0 + ½πi for positive-signed b
+∞ + ∞i	+0 + ½πi
+∞ + NaNi	+0 + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- catanh(conj(z)) = conj(catanh(z)).

For arguments with a negative real component, use the equality:

- catanh(-z) = -catanh(z).

Power and absolute value

Function	Description
cabs() (page 482)	Compute magnitude, double complex.
cabsf() (page 482)	Compute magnitude, float complex.
cabsl() (page 483)	Compute magnitude, long double complex.
cpow() (page 510)	Power, double complex.
cpowf() (page 510)	Power, float complex.
cpowl() (page 510)	Power, long double complex.
csqrt() (page 511)	Square root, double complex.
csqrtf() (page 511)	Square root, float complex.
csqrtl() (page 512)	Square root, long double complex.

cabs()

Description

Compute magnitude, double complex.

Prototype

```
double cabs(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

The magnitude of x, |x|.

cabsf()

Description

Compute magnitude, float complex.

Prototype

```
float cabsf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

The magnitude of x, |x|.

cabsl()

Description

Compute magnitude, long double complex.

Prototype

```
long double cabsl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

The magnitude of x, |x|.

cpow()

Description

Power, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cpow(__SEGGER_RTL_FLOAT64_C_COMPLEX x,
                                    __SEGGER_RTL_FLOAT64_C_COMPLEX y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power of y.

cpowf()

Description

Power, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cpowf(__SEGGER_RTL_FLOAT32_C_COMPLEX x,
                                     __SEGGER_RTL_FLOAT32_C_COMPLEX y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power of y.

cpowl()

Description

Power, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cpowl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x,
                                     __SEGGER_RTL_LDOUBLE_C_COMPLEX y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power of y.

csqrt()

Description

Square root, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX csqrt(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute squate root of.

Return value

The square root of x according to the following table:

Argument	csqrt(Argument)
±0 + 0i	+0 + 0i
a + ∞i	+∞ + ∞i, for all a
a + NaNi	+NaN + NaNi, for finite a
-∞ + bi	+0 + ∞i for finite positive-signed b
+∞ + bi	+∞ + 0i, for finite positive-signed b
+∞ + ∞i	+∞ + ¼πi
-∞ + NaNi	+NaN + +/∞i, sign of imaginary part unspecified
+∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + ∞i
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- csqrt(conj(z)) = conj(csqrt(z)).

csqrtf()

Description

Square root, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX csqrtf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute squate root of.

Return value

The square root of x according to the following table:

Argument	csqrt(Argument)
±0 + 0i	+0 + 0i
a + ∞i	+∞ + ∞i, for all a
a + NaNi	+NaN + NaNi, for finite a
-∞ + bi	+0 + ∞i for finite positive-signed b
+∞ + bi	+∞ + 0i, for finite positive-signed b
+∞ + ∞i	+∞ + ¼πi
-∞ + NaNi	+NaN + +/∞i, sign of imaginary part unspecified
+∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + ∞i
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- csqrt(conj(z)) = conj(csqrt(z)).

csqrtl()

Description

Square root, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX csqrtl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute squate root of.

Return value

The square root of x according to the following table:

Argument	csqrt(Argument)
±0 + 0i	+0 + 0i
a + ∞i	+∞ + ∞i, for all a
a + NaNi	+NaN + NaNi, for finite a
-∞ + bi	+0 + ∞i for finite positive-signed b
+∞ + bi	+∞ + 0i, for finite positive-signed b
+∞ + ∞i	+∞ + ¼πi
-∞ + NaNi	+NaN + +/∞i, sign of imaginary part unspecified
+∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + ∞i
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- csqrt(conj(z)) = conj(csqrt(z)).

Exponential and logarithm functions

Function	Description
clog() (page 513)	Compute natural logarithm, double complex.
clogf() (page 514)	Compute natural logarithm, float complex.
clogl() (page 515)	Compute natural logarithm, long double complex.
cexp() (page 515)	Compute base-e exponential, double complex.
cexpf() (page 516)	Compute base-e exponential, float complex.
cexpl() (page 517)	Compute base-e exponential, long double complex.

clog()

Description

Compute natural logarithm, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX clog(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

The natural logarithm of x according to the following table:

Argument	clog(Argument)
-0 + 0i	-∞ + πi
+0 + 0i	-∞ + 0i
a + ∞i	+∞ + ½πi, for finite a
a + NaNi	NaN + NaNi, for finite a
-∞ + bi	+∞ + πi, for finite positive b
+∞ + bi	+∞ + 0i, for finite positive b
-∞ + ∞i	+∞ + ¾πi
+∞ + ∞i	+∞ + ¼πi
±∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + NaNi
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- clog(conj(z)) = conj(clog(z)).

clogf()

Description

Compute natural logarithm, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX clogf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

The natural logarithm of x according to the following table:

Argument	clog(Argument)
-0 + 0i	-∞ + πi
+0 + 0i	-∞ + 0i
a + ∞i	+∞ + ½πi, for finite a
a + NaNi	NaN + NaNi, for finite a
-∞ + bi	+∞ + πi, for finite positive b
+∞ + bi	+∞ + 0i, for finite positive b
-∞ + ∞i	+∞ + ¾πi
+∞ + ∞i	+∞ + ¼πi
±∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + NaNi
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- clog(conj(z)) = conj(clog(z)).

clogl()

Description

Compute natural logarithm, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX clogl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

The natural logarithm of x according to the following table:

Argument	clog(Argument)
-0 + 0i	-∞ + πi
+0 + 0i	-∞ + 0i
a + ∞i	+∞ + ½πi, for finite a
a + NaNi	NaN + NaNi, for finite a
-∞ + bi	+∞ + πi, for finite positive b
+∞ + bi	+∞ + 0i, for finite positive b
-∞ + ∞i	+∞ + ¾πi
+∞ + ∞i	+∞ + ¼πi
±∞ + NaNi	+∞ + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + ∞i	+∞ + NaNi
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- clog(conj(z)) = conj(clog(z)).

cexp()

Description

Compute base-e exponential, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cexp(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute exponential of.

Return value

The base-e exponential of x=a+bi according to the following table:

Argument	cexp(Argument)
-/-0 + 0i	+1 + 0i
a + ∞i	NaN + NaNi, for finite a
a + NaNi	NaN + NaNi, for finite a
+∞ + 0i	+∞ + 0i, for finite positive b
-∞ + bi	+0 cis(b) for finite b
+∞ + bi	+∞ cis(b) for finite nonzero b
-∞ + ∞i	±∞ + ±0i, signs unspecified
+∞ + ∞i	±∞ + i.NaN, sign of real part unspecified
-∞ + NaNi	±0 + ±0i, signs unspecified
+∞ + NaNi	±∞ + NaNi, sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality

- cexp(conj(x)) = conj(cexp(x)).

cexpf()

Description

Compute base-e exponential, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cexpf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute exponential of.

Return value

The base-e exponential of x=a+bi according to the following table:

Argument	cexp(Argument)
-/-0 + 0i	+1 + 0i
a + ∞i	NaN + NaNi, for finite a
a + NaNi	NaN + NaNi, for finite a
+∞ + 0i	+∞ + 0i, for finite positive b
-∞ + bi	+0 cis(b) for finite b
+∞ + bi	+∞ cis(b) for finite nonzero b
-∞ + ∞i	±∞ + ±0i, signs unspecified
+∞ + ∞i	±∞ + i.NaN, sign of real part unspecified
-∞ + NaNi	±0 + ±0i, signs unspecified
+∞ + NaNi	±∞ + NaNi, sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality

- cexp(conj(x)) = conj(cexp(x)).

cexpl()

Description

Compute base-e exponential, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cexpl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute exponential of.

Return value

The base-e exponential of x=a+bi according to the following table:

Argument	cexp(Argument)
-/-0 + 0i	+1 + 0i
a + ∞i	NaN + NaNi, for finite a
a + NaNi	NaN + NaNi, for finite a
+∞ + 0i	+∞ + 0i, for finite positive b
-∞ + bi	+0 cis(b) for finite b
+∞ + bi	+∞ cis(b) for finite nonzero b
-∞ + ∞i	±∞ + ±0i, signs unspecified
+∞ + ∞i	±∞ + i.NaN, sign of real part unspecified
-∞ + NaNi	±0 + ±0i, signs unspecified
+∞ + NaNi	±∞ + NaNi, sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality

- cexp(conj(x)) = conj(cexp(x)).

5.6.3 <ctype.h>

Classification functions

Function	Description
iscntrl() (page 518)	Is character a control?
iscntrl_l() (page **??**)	Is character a control, per locale? (POSIX.1).
isblank() (page 519)	Is character a blank?
isblank_l() (page **??**)	Is character a blank, per locale? (POSIX.1).
isspace() (page 520)	Is character a whitespace character?
isspace_l() (page **??**)	Is character a whitespace character, per locale? (POSIX.1).
ispunct() (page 521)	Is character a punctuation mark?
ispunct_l() (page **??**)	Is character a punctuation mark, per locale? (POSIX.1).
isdigit() (page 522)	Is character a decimal digit?
isdigit_l() (page **??**)	Is character a decimal digit, per locale? (POSIX.
isxdigit() (page 522)	Is character a hexadecimal digit?
isxdigit_l() (page **??**)	Is character a hexadecimal digit, per locale? (POSIX.1).
isalpha() (page 523)	Is character alphabetic?
isalpha_l() (page **??**)	Is character alphabetic, per locale? (POSIX.1).
isalnum() (page 524)	Is character alphanumeric?
isalnum_l() (page **??**)	Is character alphanumeric, per locale? (POSIX.1).
isupper() (page 525)	Is character an uppercase letter?
isupper_l() (page **??**)	Is character an uppercase letter, per locale? (POSIX.1).
islower() (page 526)	Is character a lowercase letter?
islower_l() (page **??**)	Is character a lowercase letter, per locale? (POSIX.1).
isprint() (page 527)	Is character printable?
isprint_l() (page **??**)	Is character printable, per locale? (POSIX.1).
isgraph() (page 527)	Is character any printing character?
isgraph_l() (page **??**)	Is character any printing character, per locale? (POSIX.1).

iscntrl()

Description

Is character a control?

Prototype

```
int iscntrl(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a control character in the current locale.

iscntrl_l()

Description

Is character a control, per locale? (POSIX.1).

Prototype

```
int iscntrl_l(int c,
              locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a control character in the locale loc.

Notes

Conforms to POSIX.1-2017.

isblank()

Description

Is character a blank?

Prototype

```
int isblank(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is either a space character or tab character in the current locale.

isblank_l()

Description

Is character a blank, per locale? (POSIX.1).

Prototype

```
int isblank_l(int c,
              locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is either a space character or the tab character in locale loc.

Notes

Conforms to POSIX.1-2017.

isspace()

Description

Is character a whitespace character?

Prototype

```
int isspace(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a standard white-space character in the current locale. The standard white-space characters are space, form feed, new-line, carriage return, horizontal tab, and vertical tab.

isspace_l()

Description

Is character a whitespace character, per locale? (POSIX.1).

Prototype

```
int isspace_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a standard white-space character in the locale loc.

Notes

Conforms to POSIX.1-2017.

ispunct()

Description

Is character a punctuation mark?

Prototype

```
int ispunct(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) for every printing character for which neither *isspace()* (page 520) nor *isalnum()* (page 524) is true in the current locale.

ispunct_l()

Description

Is character a punctuation mark, per locale? (POSIX.1).

Prototype

```
int ispunct_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) for every printing character for which neither *isspace()* (page 520) nor *isalnum()* (page 524) is true in the locale loc.

Notes

Conforms to POSIX.1-2017.

isdigit()

Description

Is character a decimal digit?

Prototype

```
int isdigit(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a digit in the current locale.

isdigit_l()

Description

Is character a decimal digit, per locale? (POSIX.1)

Prototype

```
int isdigit_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a digit in the locale loc.

Notes

Conforms to POSIX.1-2017.

isxdigit()

Description

Is character a hexadecimal digit?

Prototype

```
int isxdigit(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a hexadecimal digit in the current locale.

isxdigit_l()

Description

Is character a hexadecimal digit, per locale? (POSIX.1).

Prototype

```
int isxdigit_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a hexadecimal digit in the current locale.

Notes

Conforms to POSIX.1-2017.

isalpha()

Description

Is character alphabetic?

Prototype

```
int isalpha(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns true if the character c is alphabetic in the current locale. That is, any character for which *isupper()* (page 525) or *islower()* (page 526) returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of *iscntrl()* (page 518), *isdigit()* (page 522), *ispunct()* (page 521), or *isspace()* (page 520) is true.

In the C locale, *isalpha()* (page 523) returns nonzero (true) if and only if *isupper()* (page 525) or *islower()* (page 526) return true for value of the argument c.

isalpha_l()

Description

Is character alphabetic, per locale? (POSIX.1).

Prototype

```
int isalpha_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns true if the character c is alphabetic in the locale loc. That is, any character for which *isupper()* (page 525) or *islower()* (page 526) returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of *iscntrl_l()* (page **??**), *isdigit_l()* (page **??**), *ispunct_l()* (page **??**), or *isspace_l()* (page **??**) is true in the locale loc.

In the C locale, *isalpha_l()* (page **??**) returns nonzero (true) if and only if *isupper_l()* (page **??**) or *islower_l()* (page **??**) return true for value of the argument c.

Notes

Conforms to POSIX.1-2017.

isalnum()

Description

Is character alphanumeric?

Prototype

```
int isalnum(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is an alphabetic or numeric character in the current locale.

isalnum_l()

Description

Is character alphanumeric, per locale? (POSIX.1).

Prototype

```
int isalnum_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is an alphabetic or numeric character in the locale loc.

Notes

Conforms to POSIX.1-2017.

isupper()

Description

Is character an uppercase letter?

Prototype

```
int isupper(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is an uppercase letter in the current locale.

isupper_l()

Description

Is character an uppercase letter, per locale? (POSIX.1).

Prototype

```
int isupper_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is an uppercase letter in the locale loc.

Notes

Conforms to POSIX.1-2017.

islower()

Description

Is character a lowercase letter?

Prototype

```
int islower(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a lowercase letter in the current locale.

islower_l()

Description

Is character a lowercase letter, per locale? (POSIX.1).

Prototype

```
int islower_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a lowercase letter in the locale loc.

Notes

Conforms to POSIX.1-2017.

isprint()

Description

Is character printable?

Prototype

```
int isprint(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character including space in the current locale.

isprint_l()

Description

Is character printable, per locale? (POSIX.1).

Prototype

```
int isprint_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character including space in the locale loc.

Notes

Conforms to POSIX.1-2017.

isgraph()

Description

Is character any printing character?

Prototype

```
int isgraph(int c);
```

Parameters

Parameter	Description
c	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character except space in the current locale.

isgraph_l()

Description

Is character any printing character, per locale? (POSIX.1).

Prototype

```
int isgraph_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character except space in the locale loc.

Notes

Conforms to POSIX.1-2017.

Conversion functions

Function	Description
toupper() (page 528)	Convert lowercase character to uppercase.
toupper_l() (page **??**)	Convert lowercase character to uppercase, per locale (POSIX.1).
tolower() (page 530)	Convert uppercase character to lowercase.
tolower_l() (page **??**)	Convert uppercase character to lowercase, per locale (POSIX.1).

toupper()

Description

Convert lowercase character to uppercase.

Prototype

```
int toupper(int c);
```

Parameters

Parameter	Description
c	Character to convert.

Return value

Converted character.

Additional information

Converts a lowercase letter to a corresponding uppercase letter.

If the argument c is a character for which *islower()* (page 526) is true and there are one or more corresponding characters, as specified by the current locale, for which *isupper()* (page 525) is true, *toupper()* (page 528) returns one of the corresponding characters (always the same one for any given locale); otherwise, the argument is returned unchanged.

Notes

Even though *islower()* (page 526) can return true for some characters, *toupper()* (page 528) may return that lowercase character unchanged as there are no corresponding uppercase characters in the locale.

toupper_l()

Description

Convert lowercase character to uppercase, per locale (POSIX.1).

Prototype

```
int toupper_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to convert.
loc	Locale used to convert c.

Return value

Converted character.

Additional information

Converts a lowercase letter to a corresponding uppercase letter in locale loc. If the argument c is a character for which *islower_l()* (page **??**) is true in locale loc, *tolower_l()* (page **??**) returns the corresponding uppercase letter; otherwise, the argument is returned unchanged.

Notes

Conforms to POSIX.1-2017.

tolower()

Description

Convert uppercase character to lowercase.

Prototype

```
int tolower(int c);
```

Parameters

Parameter	Description
c	Character to convert.

Return value

Converted character.

Additional information

Converts an uppercase letter to a corresponding lowercase letter.

If the argument c is a character for which *isupper()* (page 525) is true and there are one or more corresponding characters, as specified by the current locale, for which *islower()* (page 526) is true, the *tolower()* (page 530) function returns one of the corresponding characters (always the same one for any given locale); otherwise, the argument is returned unchanged.

Notes

Even though *isupper()* (page 525) can return true for some characters, *tolower()* (page 530) may return that uppercase character unchanged as there are no corresponding lowercase characters in the locale.

tolower_l()

Description

Convert uppercase character to lowercase, per locale (POSIX.1).

Prototype

```
int tolower_l(int c, locale_t loc);
```

Parameters

Parameter	Description
c	Character to convert.
loc	Locale used to convert c.

Return value

Converted character.

Additional information

Converts an uppercase letter to a corresponding lowercase letter in locale loc. If the argument is a character for which *isupper_l()* (page **??**) is true in locale loc, *tolower_l()* (page **??**) returns the corresponding lowercase letter; otherwise, the argument is returned unchanged.

Notes

Conforms to POSIX.1-2017.

5.6.4 <errno.h>

Errors

Error names

Description

Symbolic error names for raised errors.

Definition

```
#define EDOM      0x01
#define EILSEQ    0x02
#define ERANGE    0x03
#define EHEAP     0x04
#define ENOMEM    0x05
#define EINVAL    0x06
```

Symbols

Definition	Description
EDOM	Domain error
EILSEQ	Illegal multibyte sequence in conversion
ERANGE	Range error
EHEAP	Heap is corrupt
ENOMEM	Out of memory
EINVAL	Invalid parameter

errno

Description

Macro returning the current error.

Definition

```
#define errno    (*__SEGGER_RTL_X_errno_addr())
```

Additional information

The value in errno is significant only when the return value of the call indicated an error. A function that succeeds is allowed to change errno. The value of errno is never set to zero by a library function.

5.6.5 <fcntl.h>

The `<fcntl.h>` header is provided as an empty compatibility header. The runtime library implements no file control operations on an embedded target, so this header declares nothing: it only allows portable code which includes `<fcntl.h>` to build without errors.

5.6.6 <fenv.h>

Floating-point exceptions

Function	Description
feclearexcept() (page 532)	Clear floating-point exceptions.
feraiseexcept() (page 532)	Raise floating-point exceptions.
fegetexceptflag() (page 533)	Get floating-point exceptions.
fesetexceptflag() (page 533)	Set floating-point exceptions.
fetestexcept() (page 534)	Test floating-point exceptions.

feclearexcept()

Description

Clear floating-point exceptions.

Prototype

```
int feclearexcept(int excepts);
```

Parameters

Parameter	Description
excepts	Bitmask of floating-point exceptions to clear.

Return value

= 0	Floating-point exceptions successfully cleared.
≠ 0	Floating-point exceptions not cleared or not supported.

Additional information

This function attempts to clear the floating-point exceptions indicated by excepts.

Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

feraiseexcept()

Description

Raise floating-point exceptions.

Prototype

```
int feraiseexcept(int excepts);
```

Parameters

Parameter	Description
excepts	Bitmask of floating-point exceptions to raise.

Return value

= 0	All floating-point exceptions successfully raised.
≠ 0	Floating-point exceptions not successuly raised or not supported.

Additional information

This function attempts to raise the floating-point exceptions indicated by excepts.

Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

fegetexceptflag()

Description

Get floating-point exceptions.

Prototype

```
int fegetexceptflag(fexcept_t * flagp,
                    int         excepts);
```

Parameters

Parameter	Description
flagp	Pointer to object that receives the floating-point exception state.
excepts	Bitmask of floating-point exceptions to store.

Return value

= 0	Floating-point exceptions correctly stored.
≠ 0	Floating-point exceptions not correctly stored.

Additional information

This function attempts to save the floating-point exceptions indicated by excepts to the object pointed to by flagp.

See also

fesetexceptflag() (page 533).

fesetexceptflag()

Description

Set floating-point exceptions.

Prototype

```
int fesetexceptflag(const fexcept_t * flagp,
                    int         excepts);
```

Parameters

Parameter	Description
flagp	Pointer to object containing a previously-stored floating-point exception state.
excepts	Bitmask of floating-point exceptions to restore.

Return value

= 0	Floating-point exceptions correctly restored.
≠ 0	Floating-point exceptions not correctly restored.

Additional information

This function attempts to restore the floating-point exceptions indicated by excepts from the object pointed to by flagp. The exceptions to restore as indicated by excepts must have at least been specified when storing the exceptions using *fegetexceptflag()* (page 533).

See also

fegetexceptflag() (page 533).

fetestexcept()

Description

Test floating-point exceptions.

Prototype

```
int fetestexcept(int excepts);
```

Parameters

Parameter	Description
excepts	Bitmask of floating-point exceptions to test.

Return value

The bitmask of all floating-point exceptions that are currently set and are specified in excepts.

Additional information

This function determines which of the floating-point exceptions indicated by excepts are currently set.

Floating-point rounding mode

Function	Description
fegetround() (page 535)	Get floating-point rounding mode.
fesetround() (page 535)	Set floating-point rounding mode.

fegetround()

Description

Get floating-point rounding mode.

Prototype

```
int fegetround(void);
```

Return value

≥ 0	Current floating-point rounding mode.
< 0	Floating-point rounding mode cannot be determined.

Additional information

This function attempts to read the current floating-point rounding mode.

See also

fesetround() (page 535).

fesetround()

Description

Set floating-point rounding mode.

Prototype

```
int fesetround(int round);
```

Parameters

Parameter	Description
round	Rounding mode to set.

Return value

= 0	Current floating-point rounding mode is set to round.
≠ 0	Requested floating-point rounding mode cannot be set.

Additional information

This function attempts to set the current floating-point rounding mode to round.

See also

fegetround() (page 535).

Floating-point environment

Function	Description
fegetenv() (page 536)	Get floating-point environment.
fesetenv() (page 536)	Set floating-point environment.
feupdateenv() (page 537)	Restore floating-point environment and reraise exceptions.
feholdexcept() (page 537)	Save floating-point environment and set non-stop mode.

fegetenv()

Description

Get floating-point environment.

Prototype

```
int fegetenv(fenv_t * envp);
```

Parameters

Parameter	Description
envp	Pointer to object that receives the floating-point environment.

Return value

= 0	Current floating-point environment successfully stored.
≠ 0	Floating-point environment cannot be stored.

Additional information

This function attempts to store the current floating-point environment to the object pointed to by envp.

Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

See also

fesetenv() (page 536).

fesetenv()

Description

Set floating-point environment.

Prototype

```
int fesetenv(const fenv_t * envp);
```

Parameters

Parameter	Description
envp	Pointer to object containing previously-stored floating-point environment.

Return value

= 0	Current floating-point environment successfully restored.
≠ 0	Floating-point environment cannot be restored.

Additional information

This function attempts to restore the floating-point environment from the object pointed to by envp.

Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

See also

fegetenv() (page 536).

feupdateenv()

Description

Restore floating-point environment and reraise exceptions.

Prototype

```
int feupdateenv(const fenv_t * envp);
```

Parameters

Parameter	Description
envp	Pointer to object containing previously-stored floating-point environment.

Return value

= 0	Environment restored and exceptions raised successfully.
≠ 0	Failed to restore environment and raise exceptions.

Additional information

This function attempts to save the currently raised floating-point exceptions, restore the floating-point environment from the object pointed to by envp, and raise the saved exceptions.

Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

feholdexcept()

Description

Save floating-point environment and set non-stop mode.

Prototype

```
int feholdexcept(fenv_t * envp);
```

Parameters

Parameter	Description
envp	Pointer to object that receives the floating-point environment.

Return value

= 0	Environment stored and non-stop mode set successfully.
≠ 0	Failed to store environment or set non-stop mode.

Additional information

This function function saves the current floating-point environment to the object pointed to by envp, clears the floating-point status flags, and then installs a non-stop mode for all floating-point exceptions

5.6.7 <float.h>

Floating-point constants

Common parameters

Description

Applies to single-precision and double-precision formats.

Definition

```
#define FLT_ROUNDS          1
#define FLT_EVAL_METHOD     0
#define FLT_RADIX           2
#define DECIMAL_DIG         17
```

Symbols

Definition	Description
FLT_ROUNDS	Rounding mode of floating-point addition is round to nearest.
FLT_EVAL_METHO	All operations and constants are evaluated to the range and precision of the type.
FLT_RADIX	Radix of the exponent representation.
DECIMAL_DIG	Number of decimal digits that can be rounded to a floating-point number without change to the value.

Float parameters

Description

IEEE 32-bit single-precision floating format parameters.

Definition

```
#define FLT_MANT_DIG        24
#define FLT_EPSILON         1.19209290E-07f
#define FLT_DIG             6
#define FLT_MIN_EXP         -125
#define FLT_MIN             1.17549435E-38f
#define FLT_MIN_10_EXP      -37
```

(continues on next page)

(continued from previous page)

```
#define FLT_MAX_EXP       +128
#define FLT_MAX           3.40282347E+38f
#define FLT_MAX_10_EXP    +38
```

Symbols

Definition	Description
FLT_MANT_DIG	Number of base FLT_RADIX digits in the mantissa part of a float.
FLT_EPSILON	Minimum positive number such that 1.0f + FLT_EPSILON ≠ 1.0f.
FLT_DIG	Number of decimal digits of precision of a float.
FLT_MIN_EXP	Minimum value of base FLT_RADIX in the exponent part of a float.
FLT_MIN	Minimum value of a float.
FLT_MIN_10_EXP	Minimum value in base 10 of the exponent part of a float.
FLT_MAX_EXP	Maximum value of base FLT_RADIX in the exponent part of a float.
FLT_MAX	Maximum value of a float.
FLT_MAX_10_EXP	Maximum value in base 10 of the exponent part of a float.

Double parameters

Description

IEEE 64-bit double-precision floating format parameters.

Definition

```
#define DBL_MANT_DIG      53
#define DBL_EPSILON       2.2204460492503131E-16
#define DBL_DIG           15
#define DBL_MIN_EXP       -1021
#define DBL_MIN           2.2250738585072014E-308
#define DBL_MIN_10_EXP    -307
#define DBL_MAX_EXP       +1024
#define DBL_MAX           1.7976931348623157E+308
#define DBL_MAX_10_EXP    +308
```

Symbols

Definition	Description
DBL_MANT_DIG	Number of base DBL_RADIX digits in the mantissa part of a double.
DBL_EPSILON	Minimum positive number such that 1.0 + DBL_EPSILON ≠ 1.0.
DBL_DIG	Number of decimal digits of precision of a double.
DBL_MIN_EXP	Minimum value of base DBL_RADIX in the exponent part of a double.
DBL_MIN	Minimum value of a double.
DBL_MIN_10_EXP	Minimum value in base 10 of the exponent part of a double.
DBL_MAX_EXP	Maximum value of base DBL_RADIX in the exponent part of a double.
DBL_MAX	Maximum value of a double.
DBL_MAX_10_EXP	Maximum value in base 10 of the exponent part of a double.

5.6.8 <iso646.h>

The header <iso646.h> defines macros that expand to the corresponding tokens to ease writing C programs with keyboards that do not have keys for frequently-used operators.

Macros

Replacement macros

Description

Standard replacement macros.

Definition

```
#define and       &&
#define and_eq    &=
#define bitand    &
#define bitor     |
#define compl     ~
#define not       !
#define not_eq    !=
#define or        ||
#define or_eq     |=
#define xor       ^
#define xor_eq    ^=
```

5.6.9 <limits.h>

Minima and maxima

Character minima and maxima

Description

Minimum and maximum values for character types.

Definition

```
#define CHAR_BIT     8
#define CHAR_MIN     0
#define CHAR_MAX     255
#define SCHAR_MAX    127
#define SCHAR_MIN    (-128)
#define UCHAR_MAX    255
```

Symbols

Definition	Description
CHAR_BIT	Number of bits for smallest object that is not a bit-field (byte).
CHAR_MIN	Minimum value of a plain character.
CHAR_MAX	Maximum value of a plain character.
SCHAR_MAX	Maximum value of a signed character.
SCHAR_MIN	Minimum value of a signed character.
UCHAR_MAX	Maximum value of an unsigned character.

Short integer minima and maxima

Description

Minimum and maximum values for short integer types.

Definition

```
#define SHRT_MIN     (-32767 - 1)
#define SHRT_MAX     32767
#define USHRT_MAX    65535
```

Symbols

Definition	Description
SHRT_MIN	Minimum value of a short integer.
SHRT_MAX	Maximum value of a short integer.
USHRT_MAX	Maximum value of an unsigned short integer.

Integer minima and maxima

Description

Minimum and maximum values for integer types.

Definition

```
#define INT_MIN     (-2147483647 - 1)
#define INT_MAX     2147483647
#define UINT_MAX    4294967295u
```

Symbols

Definition	Description
INT_MIN	Minimum value of an integer.
INT_MAX	Maximum value of an integer.
UINT_MAX	Maximum value of an unsigned integer.

Long integer minima and maxima (32-bit)

Description

Minimum and maximum values for long integer types.

Definition

```
#define LONG_MIN     (-2147483647L - 1)
#define LONG_MAX     2147483647L
#define ULONG_MAX    4294967295uL
```

Symbols

Definition	Description
LONG_MIN	Maximum value of a long integer.
LONG_MAX	Minimum value of a long integer.
ULONG_MAX	Maximum value of an unsigned long integer.

Long integer minima and maxima (64-bit)

Description

Minimum and maximum values for long integer types.

Definition

```
#define LONG_MIN     (-9223372036854775807L - 1)
#define LONG_MAX     9223372036854775807L
#define ULONG_MAX    18446744073709551615uL
```

Symbols

Definition	Description
LONG_MIN	Minimum value of a long integer.
LONG_MAX	Maximum value of a long integer.
ULONG_MAX	Maximum value of an unsigned long integer.

Long long integer minima and maxima

Description

Minimum and maximum values for long integer types.

Definition

```
#define LLONG_MIN     (-9223372036854775807LL - 1)
#define LLONG_MAX     9223372036854775807LL
#define ULLONG_MAX    18446744073709551615uLL
```

Symbols

Definition	Description
LLONG_MIN	Minimum value of a long long integer.
LLONG_MAX	Maximum value of a long long integer.
ULLONG_MAX	Maximum value of an unsigned long long integer.

Multibyte characters

Description

Maximum number of bytes in a multi-byte character.

Definition

```
#define MB_LEN_MAX    4
```

Symbols

Definition	Description
MB_LEN_MAX	Maximum

Additional information

The maximum number of bytes in a multi-byte character for any supported locale. Unicode (ISO 10646) characters between 0x000000 and 0x10FFFF inclusive are supported which convert to a maximum of four bytes in the UTF-8 encoding.

5.6.10 <locale.h>

Data types

__SEGGER_RTL_lconv

Type definition

```
typedef struct {
  char * decimal_point;
  char * thousands_sep;
  char * grouping;
  char * int_curr_symbol;
  char * currency_symbol;
  char * mon_decimal_point;
  char * mon_thousands_sep;
  char * mon_grouping;
  char * positive_sign;
  char * negative_sign;
  char   int_frac_digits;
  char   frac_digits;
  char   p_cs_precedes;
  char   p_sep_by_space;
  char   n_cs_precedes;
  char   n_sep_by_space;
  char   p_sign_posn;
  char   n_sign_posn;
  char   int_p_cs_precedes;
  char   int_n_cs_precedes;
  char   int_p_sep_by_space;
  char   int_n_sep_by_space;
  char   int_p_sign_posn;
  char   int_n_sign_posn;
} __SEGGER_RTL_lconv;
```

Structure members

Member	Description
decimal_point	Decimal point separator.
thousands_sep	Separators used to delimit groups of digits to the left of the decimal point for non-monetary quantities.
grouping	Specifies the amount of digits that form each of the groups to be separated by thousands_sep separator for non-monetary quantities.
int_curr_symb	International currency symbol.
currency_symbol	Local currency symbol.
mon_decimal_	Decimal-point separator used for monetary quantities.
mon_thousand	Separators used to delimit groups of digits to the left of the decimal point for monetary quantities.
mon_grouping	Specifies the amount of digits that form each of the groups to be separated by mon_thousands_sep separator for monetary quantities.
positive_sign	Sign to be used for nonnegative (positive or zero) monetary quantities.
negative_sign	Sign to be used for negative monetary quantities.
int_frac_digits	Amount of fractional digits to the right of the decimal point for monetary quantities in the international format.
frac_digits	Amount of fractional digits to the right of the decimal point for monetary quantities in the local format.
p_cs_precedes	Whether the currency symbol should precede nonnegative (positive or zero) monetary quantities.
p_sep_by_spac	Whether a space should appear between the currency symbol and nonnegative (positive or zero) monetary quantities.
n_cs_precedes	Whether the currency symbol should precede negative monetary quantities.
n_sep_by_spac	Whether a space should appear between the currency symbol and negative monetary quantities.
p_sign_posn	Position of the sign for nonnegative (positive or zero) monetary quantities.
n_sign_posn	Position of the sign for negative monetary quantities.
int_p_cs_prece	Whether int_curr_symbol precedes or succeeds the value for a nonnegative internationally formatted monetary quantity.
int_n_cs_prece	Whether int_curr_symbol precedes or succeeds the value for a negative internationally formatted monetary quantity.
int_p_sep_by_	Value indicating the separation of the int_curr_symbol, the sign string, and the value for a nonnegative internationally formatted monetary quantity.
int_n_sep_by_	Value indicating the separation of the int_curr_symbol, the sign string, and the value for a negative internationally formatted monetary quantity.
int_p_sign_pos	Value indicating the positioning of the positive_sign for a nonnegative internationally formatted monetary quantity.
int_n_sign_pos	Value indicating the positioning of the positive_sign for a negative internationally formatted monetary quantity.

Locale management

Function	Description
setlocale() (page 545)	Set locale.
localeconv() (page 545)	Get current locale data.

setlocale()

Description

Set locale.

Prototype

```
char *setlocale(      int    category,
                const char * loc);
```

Parameters

Parameter	Description
category	Category of locale to set, see below.
loc	Pointer to name of locale to set or, if NULL, the current locale.

Return value

Returns the name of the current locale.

Additional information

The category parameter can have the following values:

Value	Description
LC_ALL	Entire locale.
LC_COLLATE	Affects strcoll() and strxfrm().
LC_CTYPE	Affects character handling.
LC_MONETARY	Affects monetary formatting information.
LC_NUMERIC	Affects decimal-point character in I/O and string formatting operations.
LC_TIME	Affects *strftime()* (page 709).

localeconv()

Description

Get current locale data.

Prototype

```
localeconv(void);
```

Return value

Returns a pointer to a structure of type lconv with the corresponding values for the current locale filled in.

5.6.11 <malloc.h>

The `<malloc.h>` header is provided for POSIX compatibility. Including `<malloc.h>` is equivalent to including `<stdlib.h>`: the heap allocation functions *malloc()* (page 657), *calloc()* (page 657), *realloc()* (page 658) and *free()* (page 658) are declared by `<stdlib.h>`.

5.6.12 <math.h>

Exponential and logarithm functions

Function	Description
sqrt() (page 547)	Compute square root, double.
sqrtf() (page 547)	Compute square root, float.
sqrtl() (page 548)	Compute square root, long double.
cbrt() (page 548)	Compute cube root, double.
cbrtf() (page 549)	Compute cube root, float.
cbrtl() (page 549)	Compute cube root, long double.
rsqrt() (page 550)	Compute reciprocal square root, double.
rsqrtf() (page 550)	Compute reciprocal square root, float.
rsqrtl() (page 551)	Compute reciprocal square root, long double.
exp() (page 551)	Compute base-e exponential, double.
expf() (page 552)	Compute base-e exponential, float.
expl() (page 552)	Compute base-e exponential, long double.
expm1() (page 553)	Compute base-e exponential, modified, double.
expm1f() (page 553)	Compute base-e exponential, modified, float.
expm1l() (page 553)	Compute base-e exponential, modified, long double.
exp2() (page 554)	Compute base-2 exponential, double.
exp2f() (page 554)	Compute base-2 exponential, float.
exp2l() (page 555)	Compute base-2 exponential, long double.
exp10() (page 555)	Compute base-10 exponential, double.
exp10f() (page 556)	Compute base-10 exponential, float.
exp10l() (page 556)	Compute base-10 exponential, long double.
frexp() (page 557)	Split to significand and exponent, double.
frexpf() (page 557)	Split to significand and exponent, float.
frexpl() (page 558)	Split to significand and exponent, long double.
hypot() (page 558)	Compute magnitude of complex, double.
hypotf() (page 559)	Compute magnitude of complex, float.
hypotl() (page 559)	Compute magnitude of complex, long double.
log() (page 560)	Compute natural logarithm, double.
logf() (page 560)	Compute natural logarithm, float.
logl() (page 561)	Compute natural logarithm, long double.
log2() (page 561)	Compute base-2 logarithm, double.
log2f() (page 562)	Compute base-2 logarithm, float.
log2l() (page 562)	Compute base-2 logarithm, long double.
log10() (page 563)	Compute common logarithm, double.
log10f() (page 563)	Compute common logarithm, float.
log10l() (page 564)	Compute common logarithm, long double.
logb() (page 564)	Radix-indpendent exponent, double.
logbf() (page 565)	Radix-indpendent exponent, float.
logbl() (page 565)	Radix-indpendent exponent, long double.
ilogb() (page 566)	Radix-independent exponent, double.
ilogbf() (page 566)	Radix-independent exponent, float.
ilogbl() (page 567)	Radix-independent exponent, long double.
log1p() (page 567)	Compute natural logarithm plus one, double.
log1pf() (page 568)	Compute natural logarithm plus one, float.
log1pl() (page 568)	Compute natural logarithm plus one, long double.
ldexp() (page 569)	Scale by power of two, double.
ldexpf() (page 569)	Scale by power of two, float.
ldexpl() (page 570)	Scale by power of two, long double.
pow() (page 570)	Raise to power, double.
powf() (page 571)	Raise to power, float.

continues on next page

Table 5.1 – continued from previous page

Function	Description
powl() (page 571)	Raise to power, long double.
scalbn() (page 571)	Scale, double.
scalbnf() (page 572)	Scale, float.
scalbnl() (page 573)	Scale, long double.
scalbln() (page 573)	Scale, double.
scalblnf() (page 574)	Scale, float.
scalblnl() (page 574)	Scale, long double.

sqrt()

Description

Compute square root, double.

Prototype

```
double sqrt(double x);
```

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- If x < 0, return NaN.
- Else, return square root of x.

Additional information

sqrt() (page 547) computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero, *sqrt()* (page 547) deviates and always uses IEC 60559 semantics.

sqrtf()

Description

Compute square root, float.

Prototype

```
float sqrtf(float x);
```

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- If x < 0, return NaN.
- Else, return square root of x.

Additional information

sqrt() (page 547) computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero, *sqrt()* (page 547) deviates and always uses IEC 60559 semantics.

sqrtl()

Description

Compute square root, long double.

Prototype

```
long double sqrtl(long double x);
```

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- If x < 0, return NaN.
- Else, return square root of x.

Additional information

sqrtl() (page 548) computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero, *sqrtl()* (page 548) deviates and always uses IEC 60559 semantics.

cbrt()

Description

Compute cube root, double.

Prototype

```
double cbrt(double x);
```

Parameters

Parameter	Description
x	Value to compute cube root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return cube root of x.

cbrtf()

Description

Compute cube root, float.

Prototype

```
float cbrtf(float x);
```

Parameters

Parameter	Description
x	Value to compute cube root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return cube root of x.

cbrtl()

Description

Compute cube root, long double.

Prototype

```
long double cbrtl(long double x);
```

Parameters

Parameter	Description
x	Value to compute cube root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return cube root of x.

rsqrt()

Description

Compute reciprocal square root, double.

Prototype

```
double rsqrt(double x);
```

Parameters

Parameter	Description
x	Value to compute reciprocal square root of.

Return value

- If x is +/-zero, return +/-infinity.
- If x is positively infinite, return 0.
- If x is NaN, return x.
- If x < 0, return NaN.
- Else, return reciprocal square root of x.

rsqrtf()

Description

Compute reciprocal square root, float.

Prototype

```
float rsqrtf(float x);
```

Parameters

Parameter	Description
x	Value to compute reciprocal square root of.

Return value

- If x is +/-zero, return +/-infinity.
- If x is positively infinite, return 0.
- If x is NaN, return x.
- If x < 0, return NaN.
- Else, return reciprocal square root of x.

rsqrtl()

Description

Compute reciprocal square root, long double.

Prototype

```
long double rsqrtl(long double x);
```

Parameters

Parameter	Description
x	Value to compute reciprocal square root of.

Return value

- If x is +/-zero, return +/-infinity.
- If x is positively infinite, return 0.
- If x is NaN, return x.
- If x < 0, return NaN.
- Else, return reciprocal square root of x.

exp()

Description

Compute base-e exponential, double.

Prototype

```
double exp(double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

expf()

Description

Compute base-e exponential, float.

Prototype

```
float expf(float x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

expl()

Description

Compute base-e exponential, long double.

Prototype

```
long double expl(long double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

expm1()

Description

Compute base-e exponential, modified, double.

Prototype

```
double expm1(double x);
```

Parameters

Parameter	Description
x	Value to compute exponential of.

Return value

- If x is NaN, return x.
- Else, return base-e exponential of x minus 1 (e**x - 1).

expm1f()

Description

Compute base-e exponential, modified, float.

Prototype

```
float expm1f(float x);
```

Parameters

Parameter	Description
x	Value to compute exponential of.

Return value

- If x is NaN, return x.
- Else, return base-e exponential of x minus 1 (e**x - 1).

expm1l()

Description

Compute base-e exponential, modified, long double.

Prototype

```
long double expm1l(long double x);
```

Parameters

Parameter	Description
x	Value to compute exponential of.

Return value

- If x is NaN, return x.
- Else, return base-e exponential of x minus 1 (e**x - 1).

exp2()

Description

Compute base-2 exponential, double.

Prototype

```
double exp2(double x);
```

Parameters

Parameter	Description
x	Value to compute base-2 exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

exp2f()

Description

Compute base-2 exponential, float.

Prototype

```
float exp2f(float x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

exp2l()

Description

Compute base-2 exponential, long double.

Prototype

```
long double exp2l(long double x);
```

Parameters

Parameter	Description
x	Value to compute base-2 exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

exp10()

Description

Compute base-10 exponential, double.

Prototype

```
double exp10(double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

exp10f()

Description

Compute base-10 exponential, float.

Prototype

```
float exp10f(float x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

exp10l()

Description

Compute base-10 exponential, long double.

Prototype

```
long double exp10l(long double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

frexp()

Description

Split to significand and exponent, double.

Prototype

```
double frexp(double   x,
             int    * exp);
```

Parameters

Parameter	Description
x	Floating value to operate on.
exp	Pointer to integer receiving the power-of-two exponent of x.

Return value

- If x is zero, infinite or NaN, return x and store zero into the integer pointed to by exp.
- Else, return the value f, such that f has a magnitude in the interval [0.5, 1) and x equals f * pow(2, *exp)

Additional information

Breaks a floating-point number into a normalized fraction and an integral power of two.

frexpf()

Description

Split to significand and exponent, float.

Prototype

```
float frexpf(float   x,
             int   * exp);
```

Parameters

Parameter	Description
x	Floating value to operate on.
exp	Pointer to integer receiving the power-of-two exponent of x.

Return value

- If x is zero, infinite or NaN, return x and store zero into the integer pointed to by exp.
- Else, return the value f, such that f has a magnitude in the interval [0.5, 1) and x equals f * pow(2, *exp)

Additional information

Breaks a floating-point number into a normalized fraction and an integral power of two.

frexpl()

Description

Split to significand and exponent, long double.

Prototype

```
long double frexpl(long double   x,
                   int          * exp);
```

Parameters

Parameter	Description
x	Floating value to operate on.
exp	Pointer to integer receiving the power-of-two exponent of x.

Return value

- If x is zero, infinite or NaN, return x and store zero into the integer pointed to by exp.
- Else, return the value f, such that f has a magnitude in the interval [0.5, 1) and x equals f * pow(2, *exp)

Additional information

Breaks a floating-point number into a normalized fraction and an integral power of two.

hypot()

Description

Compute magnitude of complex, double.

Prototype

```
double hypot(double x,
             double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x or y are infinite, return infinity.
- If x or y is NaN, return NaN.
- Else, return sqrt(x*x + y*y).

Additional information

Computes the square root of the sum of the squares of x and y without undue overflow or underflow. If x and y are the lengths of the sides of a right-angled triangle, then this computes the length of the hypotenuse.

hypotf()

Description

Compute magnitude of complex, float.

Prototype

```
float hypotf(float x,
             float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x or y are infinite, return infinity.
- If x or y is NaN, return NaN.
- Else, return sqrt(x*x + y*y).

Additional information

Computes the square root of the sum of the squares of x and y without undue overflow or underflow. If x and y are the lengths of the sides of a right-angled triangle, then this computes the length of the hypotenuse.

hypotl()

Description

Compute magnitude of complex, long double.

Prototype

```
long double hypotl(long double x,
                   long double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x or y are infinite, return infinity.
- If x or y is NaN, return NaN.
- Else, return sqrtl(x*x + y*y).

Additional information

Computes the square root of the sum of the squares of x and y without undue overflow or underflow. If x and y are the lengths of the sides of a right-angled triangle, then this computes the length of the hypotenuse.

log()

Description

Compute natural logarithm, double.

Prototype

```
double log(double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return -∞.
- If x is +∞, return +∞.
- ELse, return base-e logarithm of x.

logf()

Description

Compute natural logarithm, float.

Prototype

```
float logf(float x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-e logarithm of x.

logl()

Description

Compute natural logarithm, long double.

Prototype

```
long double logl(long double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return -∞.
- If x is +∞, return +∞.
- ELse, return base-e logarithm of x.

log2()

Description

Compute base-2 logarithm, double.

Prototype

```
double log2(double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-10 logarithm of x.

log2f()

Description

Compute base-2 logarithm, float.

Prototype

```
float log2f(float x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-10 logarithm of x.

log2l()

Description

Compute base-2 logarithm, long double.

Prototype

```
long double log2l(long double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-10 logarithm of x.

log10()

Description

Compute common logarithm, double.

Prototype

```
double log10(double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-10 logarithm of x.

log10f()

Description

Compute common logarithm, float.

Prototype

```
float log10f(float x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-10 logarithm of x.

log10l()

Description

Compute common logarithm, long double.

Prototype

```
long double log10l(long double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-10 logarithm of x.

logb()

Description

Radix-indpendent exponent, double.

Prototype

```
double logb(double x);
```

Parameters

Parameter	Description
x	Floating value to operate on.

Return value

- If x is zero, return -∞.
- If x is infinite, return +∞.
- If x is NaN, return NaN.
- Else, return integer part of $\log_{\text{FLTRADIX}}(x)$.

Additional information

Calculates the exponent of x, which is the integral part of the FLTRADIX-logarithm of x.

logbf()

Description

Radix-indpendent exponent, float.

Prototype

```
float logbf(float x);
```

Parameters

Parameter	Description
x	Floating value to operate on.

Return value

- If x is zero, return -∞.
- If x is infinite, return +∞.
- If x is NaN, return NaN.
- Else, return integer part of $\log_{\text{FLTRADIX}}(x)$.

Additional information

Calculates the exponent of x, which is the integral part of the FLTRADIX-logarithm of x.

logbl()

Description

Radix-indpendent exponent, long double.

Prototype

```
long double logbl(long double x);
```

Parameters

Parameter	Description
x	Floating value to operate on.

Return value

- If x is zero, return -∞.
- If x is infinite, return +∞.
- If x is NaN, return NaN.
- Else, return integer part of $\log_{\text{FLTRADIX}}(x)$.

Additional information

Calculates the exponent of x, which is the integral part of the FLTRADIX-logarithm of x.

ilogb()

Description

Radix-independent exponent, double.

Prototype

```
int ilogb(double x);
```

Parameters

Parameter	Description
x	Floating value to operate on.

Return value

- If x is zero, return FP_ILOGB0.
- If x is NaN, return FP_ILOGBNAN.
- If x is infinite, return MAX_INT.
- Else, return integer part of $\log_{\text{FLTRADIX}}(x)$.

ilogbf()

Description

Radix-independent exponent, float.

Prototype

```
int ilogbf(float x);
```

Parameters

Parameter	Description
x	Floating value to operate on.

Return value

- If x is zero, return FP_ILOGB0.
- If x is NaN, return FP_ILOGBNAN.
- If x is infinite, return MAX_INT.
- Else, return integer part of $\log_{\text{FLTRADIX}}(x)$.

ilogbl()

Description

Radix-independent exponent, long double.

Prototype

```
int ilogbl(long double x);
```

Parameters

Parameter	Description
x	Floating value to operate on.

Return value

- If x is zero, return FP_ILOGB0.
- If x is NaN, return FP_ILOGBNAN.
- If x is infinite, return MAX_INT.
- Else, return integer part of $\log_{\text{FLTRADIX}}(x)$.

log1p()

Description

Compute natural logarithm plus one, double.

Prototype

```
double log1p(double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-e logarithm of x+1.

log1pf()

Description

Compute natural logarithm plus one, float.

Prototype

```
float log1pf(float x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-e logarithm of x+1.

log1pl()

Description

Compute natural logarithm plus one, long double.

Prototype

```
long double log1pl(long double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-e logarithm of x+1.

ldexp()

Description

Scale by power of two, double.

Prototype

```
double ldexp(double x,
             int    n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of two to scale by.

Return value

- If x is ±0, return x;
- If x is ±∞, return x.
- If x is NaN, return x.
- Else, return x * 2 ^ n.

Additional information

Multiplies a floating-point number by an integral power of two.

See also

scalbn() (page 571)

ldexpf()

Description

Scale by power of two, float.

Prototype

```
float ldexpf(float x,
             int   n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of two to scale by.

Return value

- If x is zero, return x;
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * 2^n.

Additional information

Multiplies a floating-point number by an integral power of two.

See also

scalbnf() (page 572)

ldexpl()

Description

Scale by power of two, long double.

Prototype

```
long double ldexpl(long double x,
                   int         n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of two to scale by.

Return value

- If x is ±0, return x;
- If x is ±∞, return x.
- If x is NaN, return x.
- Else, return x * 2 ^ n.

Additional information

Multiplies a floating-point number by an integral power of two.

See also

scalbn() (page 571)

pow()

Description

Raise to power, double.

Prototype

```
double pow(double x,
           double y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

powf()

Description

Raise to power, float.

Prototype

```
float powf(float x,
           float y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

powl()

Description

Raise to power, long double.

Prototype

```
long double powl(long double x,
                 long double y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

scalbn()

Description

Scale, double.

Prototype

```
double scalbn(double x,
              int    n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of DBL_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * DBL_RADIX ^ n.

Additional information

Multiplies a floating-point number by an integral power of DBL_RADIX.

As floating-point arithmetic conforms to IEC 60559, DBL_RADIX is 2 and *scalbn()* (page 571) is (in this implementation) identical to *ldexp()* (page 569).

See also

ldexp() (page 569)

scalbnf()

Description

Scale, float.

Prototype

```
float scalbnf(float x,
              int   n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of FLT_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * FLT_RADIX ^ n.

Additional information

Multiplies a floating-point number by an integral power of FLT_RADIX.

As floating-point arithmetic conforms to IEC 60559, FLT_RADIX is 2 and *scalbnf()* (page 572) is (in this implementation) identical to *ldexpf()* (page 569).

See also

ldexpf() (page 569)

scalbnl()

Description

Scale, long double.

Prototype

```
long double scalbnl(long double x,
                    int         n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of LDBL_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * LDBL_RADIX ^ n.

Additional information

Multiplies a floating-point number by an integral power of LDBL_RADIX.

As floating-point arithmetic conforms to IEC 60559, LDBL_RADIX is 2 and *scalbnl()* (page 573) is (in this implementation) identical to *ldexpl()* (page 570).

See also

ldexpl() (page 570)

scalbln()

Description

Scale, double.

Prototype

```
double scalbln(double x,
               long   n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of DBL_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * DBL_RADIX ^ n.

Additional information

Multiplies a floating-point number by an integral power of DBL_RADIX.

As floating-point arithmetic conforms to IEC 60559, DBL_RADIX is 2 and *scalbln()* (page 573) is (in this implementation) identical to *ldexp()* (page 569).

See also

ldexp() (page 569)

scalblnf()

Description

Scale, float.

Prototype

```
float scalblnf(float x,
               long  n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of FLT_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * FLT_RADIX ^ n.

Additional information

Multiplies a floating-point number by an integral power of FLT_RADIX.

As floating-point arithmetic conforms to IEC 60559, FLT_RADIX is 2 and *scalbnf()* (page 572) is (in this implementation) identical to *ldexpf()* (page 569).

scalblnl()

Description

Scale, long double.

Prototype

```
long double scalblnl(long double x,
                     long        n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of LDBL_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * LDBL_RADIX ^ n.

Additional information

Multiplies a floating-point number by an integral power of LDBL_RADIX.

As floating-point arithmetic conforms to IEC 60559, LDBL_RADIX is 2 and *scalblnl()* (page 574) is (in this implementation) identical to *ldexpl()* (page 570).

See also

ldexpl() (page 570)

Trigonometric functions

Function	Description
sin() (page 575)	Calculate sine, double.
sinf() (page 576)	Calculate sine, float.
sinl() (page 576)	Calculate sine, long double.
cos() (page 577)	Calculate cosine, double.
cosf() (page 577)	Calculate cosine, float.
cosl() (page 577)	Calculate cosine, long double.
tan() (page 578)	Compute tangent, double.
tanf() (page 578)	Compute tangent, float.
tanl() (page 579)	Compute tangent, long double.
sinh() (page 579)	Compute hyperbolic sine, double.
sinhf() (page 580)	Compute hyperbolic sine, float.
sinhl() (page 580)	Compute hyperbolic sine, long double.
cosh() (page 580)	Compute hyperbolic cosine, double.
coshf() (page 581)	Compute hyperbolic cosine, float.
coshl() (page 581)	Compute hyperbolic cosine, long double.
tanh() (page 582)	Compute hyperbolic tangent, double.
tanhf() (page 582)	Compute hyperbolic tangent, float.
tanhl() (page 582)	Compute hyperbolic tangent, long double.

sin()

Description

Calculate sine, double.

Prototype

```
double sin(double x);
```

Parameters

Parameter	Description
x	Angle to compute sine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.

- Else, return circular sine of x.

sinf()

Description

Calculate sine, float.

Prototype

```
float sinf(float x);
```

Parameters

Parameter	Description
x	Angle to compute sine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular sine of x.

sinl()

Description

Calculate sine, long double.

Prototype

```
long double sinl(long double x);
```

Parameters

Parameter	Description
x	Angle to compute sine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular sine of x.

cos()

Description

Calculate cosine, double.

Prototype

```
double cos(double x);
```

Parameters

Parameter	Description
x	Angle to compute cosine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular cosine of x.

cosf()

Description

Calculate cosine, float.

Prototype

```
float cosf(float x);
```

Parameters

Parameter	Description
x	Angle to compute cosine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular cosine of x.

cosl()

Description

Calculate cosine, long double.

Prototype

```
long double cosl(long double x);
```

Parameters

Parameter	Description
x	Angle to compute cosine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular cosine of x.

tan()

Description

Compute tangent, double.

Prototype

```
double tan(double x);
```

Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

Return value

- If x is zero, return x.
- If x is infinite, return NaN.
- If x is NaN, return x.
- Else, return tangent of x.

tanf()

Description

Compute tangent, float.

Prototype

```
float tanf(float x);
```

Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

Return value

- If x is zero, return x.
- If x is infinite, return NaN.

- If x is NaN, return x.
- Else, return tangent of x.

tanl()

Description

Compute tangent, long double.

Prototype

```
long double tanl(long double x);
```

Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

Return value

- If x is zero, return x.
- If x is infinite, return NaN.
- If x is NaN, return x.
- Else, return tangent of x.

sinh()

Description

Compute hyperbolic sine, double.

Prototype

```
double sinh(double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

- If x is NaN, return x.
- If x is infinite, return x.
- Else, return hyperbolic sine of x.

sinhf()

Description

Compute hyperbolic sine, float.

Prototype

```
float sinhf(float x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

- If x is NaN, return x.
- If x is infinite, return x.
- Else, return hyperbolic sine of x.

sinhl()

Description

Compute hyperbolic sine, long double.

Prototype

```
long double sinhl(long double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

- If x is NaN, return x.
- If x is infinite, return x.
- Else, return hyperbolic sine of x.

cosh()

Description

Compute hyperbolic cosine, double.

Prototype

```
double cosh(double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

- If x is NaN, return x.
- If x is infinite, return +∞.
- Else, return hyperbolic cosine of x.

coshf()

Description

Compute hyperbolic cosine, float.

Prototype

```
float coshf(float x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

- If x is NaN, return x.
- If x is infinite, return +∞.
- Else, return hyperbolic cosine of x.

coshl()

Description

Compute hyperbolic cosine, long double.

Prototype

```
long double coshl(long double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

- If x is NaN, return x.
- If x is infinite, return +∞.
- Else, return hyperbolic cosine of x.

tanh()

Description

Compute hyperbolic tangent, double.

Prototype

```
double tanh(double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

- If x is NaN, return x.
- Else, return hyperbolic tangent of x.

tanhf()

Description

Compute hyperbolic tangent, float.

Prototype

```
float tanhf(float x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

- If x is NaN, return x.
- Else, return hyperbolic tangent of x.

tanhl()

Description

Compute hyperbolic tangent, long double.

Prototype

```
long double tanhl(long double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

- If x is NaN, return x.
- Else, return hyperbolic tangent of x.

Inverse trigonometric functions

Function	Description
asin() (page 583)	Compute inverse sine, double.
asinf() (page 584)	Compute inverse sine, float.
asinl() (page 584)	Compute inverse sine, long double.
acos() (page 585)	Compute inverse cosine, double.
acosf() (page 585)	Compute inverse cosine, float.
acosl() (page 586)	Compute inverse cosine, long double.
atan() (page 586)	Compute inverse tangent, double.
atanf() (page 587)	Compute inverse tangent, float.
atanl() (page 587)	Compute inverse tangent, long double.
atan2() (page 588)	Compute inverse tangent, with quadrant, double.
atan2f() (page 588)	Compute inverse tangent, with quadrant, float.
atan2l() (page 589)	Compute inverse tangent, with quadrant, long double.
asinh() (page 589)	Compute inverse hyperbolic sine, double.
asinhf() (page 590)	Compute inverse hyperbolic sine, float.
asinhl() (page 590)	Compute inverse hyperbolic sine, long double.
acosh() (page 591)	Compute inverse hyperbolic cosine, double.
acoshf() (page 591)	Compute inverse hyperbolic cosine, float.
acoshl() (page 591)	Compute inverse hyperbolic cosine, long double.
atanh() (page 592)	Compute inverse hyperbolic tangent, double.
atanhf() (page 592)	Compute inverse hyperbolic tangent, float.
atanhl() (page 593)	Compute inverse hyperbolic tangent, long double.

asin()

Description

Compute inverse sine, double.

Prototype

```
double asin(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse sine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular sine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

asinf()

Description

Compute inverse sine, float.

Prototype

```
float asinf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse sine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular sine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

asinl()

Description

Compute inverse sine, long double.

Prototype

```
long double asinl(long double x);
```

Parameters

Parameter	Description
x	Value to compute inverse sine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular sine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

acos()

Description

Compute inverse cosine, double.

Prototype

```
double acos(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular cosine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular cosine of x. The principal value lies in the interval [0, Pi] radians.

acosf()

Description

Compute inverse cosine, float.

Prototype

```
float acosf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular cosine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular cosine of x. The principal value lies in the interval [0, Pi] radians.

acosl()

Description

Compute inverse cosine, long double.

Prototype

```
long double acosl(long double x);
```

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular cosine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular cosine of x. The principal value lies in the interval [0, Pi] radians.

atan()

Description

Compute inverse tangent, double.

Prototype

```
double atan(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse tangent of.

Return value

- If x is NaN, return x.
- Else, return inverse tangent of x.

Additional information

Calculates the principal value, in radians, of the inverse tangent of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

atanf()

Description

Compute inverse tangent, float.

Prototype

```
float atanf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse tangent of.

Return value

- If x is NaN, return x.
- Else, return inverse tangent of x.

Additional information

Calculates the principal value, in radians, of the inverse tangent of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

atanl()

Description

Compute inverse tangent, long double.

Prototype

```
long double atanl(long double x);
```

Parameters

Parameter	Description
x	Value to compute inverse tangent of.

Return value

- If x is NaN, return x.
- Else, return inverse tangent of x.

Additional information

Calculates the principal value, in radians, of the inverse tangent of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

atan2()

Description

Compute inverse tangent, with quadrant, double.

Prototype

```
double atan2(double y,
             double x);
```

Parameters

Parameter	Description
y	Rise value of angle.
x	Run value of angle.

Return value

Inverse tangent of y/x.

Additional information

This calculates the value, in radians, of the inverse tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval [-Pi, +Pi] radians.

atan2f()

Description

Compute inverse tangent, with quadrant, float.

Prototype

```
float atan2f(float y,
             float x);
```

Parameters

Parameter	Description
y	Rise value of angle.
x	Run value of angle.

Return value

Inverse tangent of y/x.

Additional information

This calculates the value, in radians, of the inverse tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval [-Pi, +Pi] radians.

atan2l()

Description

Compute inverse tangent, with quadrant, long double.

Prototype

```
long double atan2l(long double y,
                   long double x);
```

Parameters

Parameter	Description
y	Rise value of angle.
x	Run value of angle.

Return value

Inverse tangent of y/x.

Additional information

This calculates the value, in radians, of the inverse tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval [-Pi, +Pi] radians.

asinh()

Description

Compute inverse hyperbolic sine, double.

Prototype

```
double asinh(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return inverse hyperbolic sine of x.

asinhf()

Description

Compute inverse hyperbolic sine, float.

Prototype

```
float asinhf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return inverse hyperbolic sine of x.

Additional information

Calculates the inverse hyperbolic sine of x.

asinhl()

Description

Compute inverse hyperbolic sine, long double.

Prototype

```
long double asinhl(long double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return inverse hyperbolic sine of x.

acosh()

Description

Compute inverse hyperbolic cosine, double.

Prototype

```
double acosh(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

- If x < 1, return NaN.
- If x is NaN, return x.
- Else, return non-negative inverse hyperbolic cosine of x.

acoshf()

Description

Compute inverse hyperbolic cosine, float.

Prototype

```
float acoshf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

- If x < 1, return NaN.
- If x is NaN, return x.
- Else, return non-negative inverse hyperbolic cosine of x.

acoshl()

Description

Compute inverse hyperbolic cosine, long double.

Prototype

```
long double acoshl(long double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

- If x < 1, return NaN.
- If x is NaN, return x.
- Else, return non-negative inverse hyperbolic cosine of x.

atanh()

Description

Compute inverse hyperbolic tangent, double.

Prototype

```
double atanh(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- If x = +/-1, return +/-infinity.
- Else, return non-negative inverse hyperbolic tangent of x.

atanhf()

Description

Compute inverse hyperbolic tangent, float.

Prototype

```
float atanhf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.

- If x = +/-1, return +/-infinity.
- Else, return non-negative inverse hyperbolic tangent of x.

atanhl()

Description

Compute inverse hyperbolic tangent, long double.

Prototype

```
long double atanhl(long double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- If x = +/-1, return +/-infinity.
- Else, return non-negative inverse hyperbolic tangent of x.

Special functions

Function	Description
erf() (page 593)	Error function, double.
erff() (page 594)	Error function, float.
erfl() (page 594)	Error function, long double.
erfc() (page 595)	Complementary error function, double.
erfcf() (page 595)	Complementary error function, float.
erfcl() (page 595)	Complementary error function, long double.
lgamma() (page 596)	Log-Gamma function, double.
lgammaf() (page 596)	Log-Gamma function, float.
lgammal() (page 596)	Log-Gamma function, long double.
tgamma() (page 597)	Gamma function, double.
tgammaf() (page 597)	Gamma function, float.
tgammal() (page 597)	Gamma function, long double.

erf()

Description

Error function, double.

Prototype

```
double erf(double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

erf(x).

erff()

Description

Error function, float.

Prototype

```
float erff(float x);
```

Parameters

Parameter	Description
x	Argument.

Return value

erf(x).

erfl()

Description

Error function, long double.

Prototype

```
long double erfl(long double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

erf(x).

erfc()

Description

Complementary error function, double.

Prototype

```
double erfc(double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

erfc(x).

erfcf()

Description

Complementary error function, float.

Prototype

```
float erfcf(float x);
```

Parameters

Parameter	Description
x	Argument.

Return value

erfc(x).

erfcl()

Description

Complementary error function, long double.

Prototype

```
long double erfcl(long double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

erfc(x).

lgamma()

Description

Log-Gamma function, double.

Prototype

```
double lgamma(double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

log(gamma(x)).

lgammaf()

Description

Log-Gamma function, float.

Prototype

```
float lgammaf(float x);
```

Parameters

Parameter	Description
x	Argument.

Return value

log(gamma(x)).

lgammal()

Description

Log-Gamma function, long double.

Prototype

```
long double lgammal(long double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

log(gamma(x)).

tgamma()

Description

Gamma function, double.

Prototype

```
double tgamma(double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

gamma(x).

tgammaf()

Description

Gamma function, float.

Prototype

```
float tgammaf(float x);
```

Parameters

Parameter	Description
x	Argument.

Return value

gamma(x).

tgammal()

Description

Gamma function, long double.

Prototype

```
long double tgammal(long double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

gamma(x).

Rounding and remainder functions

Function	Description
ceil() (page 599)	Compute smallest integer not less than, double.
ceilf() (page 599)	Compute smallest integer not less than, float.
ceill() (page 600)	Compute smallest integer not less than, long double.
floor() (page 600)	Compute largest integer not greater than, double.
floorf() (page 601)	Compute largest integer not greater than, float.
floorl() (page 601)	Compute largest integer not greater than, long double.
trunc() (page 602)	Truncate to integer, double.
truncf() (page 602)	Truncate to integer, float.
truncl() (page 602)	Truncate to integer, long double.
rint() (page 603)	Round to nearest integer, double.
rintf() (page 603)	Round to nearest integer, float.
rintl() (page 604)	Round to nearest integer, long double.
lrint() (page 604)	Round to nearest integer, double.
lrintf() (page 604)	Round to nearest integer, float.
lrintl() (page 605)	Round to nearest integer, long double.
llrint() (page 605)	Round to nearest integer, double.
llrintf() (page 606)	Round to nearest integer, float.
llrintl() (page 606)	Round to nearest integer, long double.
round() (page 606)	Round to nearest integer, double.
roundf() (page 607)	Round to nearest integer, float.
roundl() (page 607)	Round to nearest integer, long double.
lround() (page 608)	Round to nearest integer, double.
lroundf() (page 608)	Round to nearest integer, float.
lroundl() (page 608)	Round to nearest integer, long double.
llround() (page 609)	Round to nearest integer, double.
llroundf() (page 609)	Round to nearest integer, float.
llroundl() (page 610)	Round to nearest integer, long double.
nearbyint() (page 610)	Round to nearest integer, double.
nearbyintf() (page 610)	Round to nearest integer, float.
nearbyintl() (page 611)	Round to nearest integer, long double.
fmod() (page 611)	Compute remainder after division, double.
fmodf() (page 612)	Compute remainder after division, float.
fmodl() (page 612)	Compute remainder after division, long double.
modf() (page 613)	Separate integer and fractional parts, double.
modff() (page 614)	Separate integer and fractional parts, float.
modfl() (page 614)	Separate integer and fractional parts, long double.
remainder() (page 615)	Compute remainder after division, double.
remainderf() (page 615)	Compute remainder after division, float.
remainderl() (page 616)	Compute remainder after division, long double.
remquo() (page 616)	Compute remainder after division, double.
remquof() (page 617)	Compute remainder after division, float.
remquol() (page 618)	Compute remainder after division, long double.

ceil()

Description

Compute smallest integer not less than, double.

Prototype

```
double ceil(double x);
```

Parameters

Parameter	Description
x	Value to compute ceiling of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the smallest integer value not greater than x.

ceilf()

Description

Compute smallest integer not less than, float.

Prototype

```
float ceilf(float x);
```

Parameters

Parameter	Description
x	Value to compute ceiling of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the smallest integer value not greater than x.

ceill()

Description

Compute smallest integer not less than, long double.

Prototype

```
long double ceill(long double x);
```

Parameters

Parameter	Description
x	Value to compute ceiling of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the smallest integer value not greater than x.

floor()

Description

Compute largest integer not greater than, double.

Prototype

```
double floor(double x);
```

Parameters

Parameter	Description
x	Value to floor.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the largest integer value not greater than x.

floorf()

Description

Compute largest integer not greater than, float.

Prototype

```
float floorf(float x);
```

Parameters

Parameter	Description
x	Value to floor.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the largest integer value not greater than x.

floorl()

Description

Compute largest integer not greater than, long double.

Prototype

```
long double floorl(long double x);
```

Parameters

Parameter	Description
x	Value to floor.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the largest integer value not greater than x.

trunc()

Description

Truncate to integer, double.

Prototype

```
double trunc(double x);
```

Parameters

Parameter	Description
x	Value to truncate.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x with fractional part removed.

truncf()

Description

Truncate to integer, float.

Prototype

```
float truncf(float x);
```

Parameters

Parameter	Description
x	Value to truncate.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x with fractional part removed.

truncl()

Description

Truncate to integer, long double.

Prototype

```
long double truncl(long double x);
```

Parameters

Parameter	Description
x	Value to truncate.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x with fractional part removed.

rint()

Description

Round to nearest integer, double.

Prototype

```
double rint(double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

rintf()

Description

Round to nearest integer, float.

Prototype

```
float rintf(float x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

rintl()

Description

Round to nearest integer, long double.

Prototype

```
long double rintl(long double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

lrint()

Description

Round to nearest integer, double.

Prototype

```
long lrint(double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

lrintf()

Description

Round to nearest integer, float.

Prototype

```
long lrintf(float x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

lrintl()

Description

Round to nearest integer, long double.

Prototype

```
long lrintl(long double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

llrint()

Description

Round to nearest integer, double.

Prototype

```
long long llrint(double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

llrintf()

Description

Round to nearest integer, float.

Prototype

```
long long llrintf(float x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

llrintl()

Description

Round to nearest integer, long double.

Prototype

```
long long llrintl(long double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

round()

Description

Round to nearest integer, double.

Prototype

```
double round(double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x, ties away from zero.

roundf()

Description

Round to nearest integer, float.

Prototype

```
float roundf(float x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x, ties away from zero.

roundl()

Description

Round to nearest integer, long double.

Prototype

```
long double roundl(long double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x, ties away from zero.

lround()

Description

Round to nearest integer, double.

Prototype

```
long lround(double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

lroundf()

Description

Round to nearest integer, float.

Prototype

```
long lroundf(float x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

lroundl()

Description

Round to nearest integer, long double.

Prototype

```
long lroundl(long double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

llround()

Description

Round to nearest integer, double.

Prototype

```
long long llround(double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

llroundf()

Description

Round to nearest integer, float.

Prototype

```
long long llroundf(float x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

llroundl()

Description

Round to nearest integer, long double.

Prototype

```
long long llroundl(long double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

nearbyint()

Description

Round to nearest integer, double.

Prototype

```
double nearbyint(double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

nearbyintf()

Description

Round to nearest integer, float.

Prototype

```
float nearbyintf(float x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

nearbyintl()

Description

Round to nearest integer, long double.

Prototype

```
long double nearbyintl(long double x);
```

Parameters

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

fmod()

Description

Compute remainder after division, double.

Prototype

```
double fmod(double x,
            double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.

- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

fmodf()

Description

Compute remainder after division, float.

Prototype

```
float fmodf(float x,
            float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

fmodl()

Description

Compute remainder after division, long double.

Prototype

```
long double fmodl(long double x,
                  long double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

modf()

Description

Separate integer and fractional parts, double.

Prototype

```
double modf(double   x,
            double * iptr);
```

Parameters

Parameter	Description
x	Value to separate.
iptr	Pointer to object that receives the integral part of x.

Return value

The signed fractional part of x.

Additional information

Breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and *modf()* (page 613) returns the signed fractional part of x.

modff()

Description

Separate integer and fractional parts, float.

Prototype

```
float modff(float   x,
            float * iptr);
```

Parameters

Parameter	Description
x	Value to separate.
iptr	Pointer to object that receives the integral part of x.

Return value

The signed fractional part of x.

Additional information

Breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and *modff()* (page 614) returns the signed fractional part of x.

modfl()

Description

Separate integer and fractional parts, long double.

Prototype

```
long double modfl(long double   x,
                  long double * iptr);
```

Parameters

Parameter	Description
x	Value to separate.
iptr	Pointer to object that receives the integral part of x.

Return value

The signed fractional part of x.

Additional information

Breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and *modf()* (page 613) returns the signed fractional part of x.

remainder()

Description

Compute remainder after division, double.

Prototype

```
double remainder(double x,
                 double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

remainderf()

Description

Compute remainder after division, float.

Prototype

```
float remainderf(float x,
                 float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.

- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

remainderl()

Description

Compute remainder after division, long double.

Prototype

```
long double remainderl(long double x,
                       long double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

remquo()

Description

Compute remainder after division, double.

Prototype

```
double remquo(double   x,
              double   y,
              int    * quo);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.
quo	Pointer to object that receives the integer part of x divided by y.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

remquof()

Description

Compute remainder after division, float.

Prototype

```
float remquof(float   x,
              float   y,
              int   * quo);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.
quo	Pointer to object that receives the integer part of x divided by y.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

remquol()

Description

Compute remainder after division, long double.

Prototype

```
long double remquol(long double   x,
                    long double   y,
                    int         * quo);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.
quo	Pointer to object that receives the integer part of x divided by y.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

Absolute value functions

Function	Description
fabs() (page 619)	Compute absolute value, double.
fabsf() (page 619)	Compute absolute value, float.
fabsl() (page 619)	Compute absolute value, long double.

fabs()

Description

Compute absolute value, double.

Prototype

```
double fabs(double x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

- If x is NaN, return x.
- Else, absolute value of x.

fabsf()

Description

Compute absolute value, float.

Prototype

```
float fabsf(float x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

- If x is NaN, return x.
- Else, absolute value of x.

fabsl()

Description

Compute absolute value, long double.

Prototype

```
long double fabsl(long double x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

- If x is NaN, return x.
- Else, absolute value of x.

Fused multiply functions

Function	Description
fma() (page 620)	Compute fused multiply-add, double.
fmaf() (page 620)	Compute fused multiply-add, float.
fmal() (page 621)	Compute fused multiply-add, long double.

fma()

Description

Compute fused multiply-add, double.

Prototype

```
double fma(double x,
           double y,
           double z);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.
z	Summand.

Return value

Return (x * y) + z.

fmaf()

Description

Compute fused multiply-add, float.

Prototype

```
float fmaf(float x,
           float y,
           float z);
```

Parameters

Parameter	Description
x	Multiplier.
y	Multiplicand.
z	Summand.

Return value

Return (x * y) + z.

fmal()

Description

Compute fused multiply-add, long double.

Prototype

```
long double fmal(long double x,
                 long double y,
                 long double z);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.
z	Summand.

Return value

Return (x * y) + z.

Maximum, minimum, and positive difference functions

Function	Description
fmin() (page 621)	Compute minimum, double.
fminf() (page 622)	Compute minimum, float.
fminl() (page 622)	Compute minimum, long double.
fmax() (page 623)	Compute maximum, double.
fmaxf() (page 623)	Compute maximum, float.
fmaxl() (page 624)	Compute maximum, long double.
fdim() (page 624)	Positive difference, double.
fdimf() (page 624)	Positive difference, float.
fdiml() (page 625)	Positive difference, long double.

fmin()

Description

Compute minimum, double.

Prototype

```
double fmin(double x,
            double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return minimum of x and y.

fminf()

Description

Compute minimum, float.

Prototype

```
float fminf(float x,
            float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return minimum of x and y.

fminl()

Description

Compute minimum, long double.

Prototype

```
long double fminl(long double x,
                  long double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return minimum of x and y.

fmax()

Description

Compute maximum, double.

Prototype

```
double fmax(double x,
            double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return maximum of x and y.

fmaxf()

Description

Compute maximum, float.

Prototype

```
float fmaxf(float x,
            float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return maximum of x and y.

fmaxl()

Description

Compute maximum, long double.

Prototype

```
long double fmaxl(long double x,
                  long double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return maximum of x and y.

fdim()

Description

Positive difference, double.

Prototype

```
double fdim(double x,
            double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x > y, x-y.
- Else, +0.

fdimf()

Description

Positive difference, float.

Prototype

```
float fdimf(float x,
            float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x > y, x-y.
- Else, +0.

fdiml()

Description

Positive difference, long double.

Prototype

```
long double fdiml(long double x,
                  long double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x > y, x-y.
- Else, +0.

Miscellaneous functions

Function	Description
nextafter() (page 626)	Next machine-floating value, double.
nextafterf() (page 626)	Next machine-floating value, float.
nextafterl() (page 626)	Next machine-floating value, long double.
nexttoward() (page 627)	Next machine-floating value, double.
nexttowardf() (page 627)	Next machine-floating value, float.
nexttowardl() (page 628)	Next machine-floating value, long double.
nan() (page 628)	Parse NaN, double.
nanf() (page 628)	Parse NaN, float.
nanl() (page 629)	Parse NaN, long double.
copysign() (page 629)	Copy sign, double.
copysignf() (page 629)	Copy sign, float.
copysignl() (page 630)	Copy sign, long double.

nextafter()

Description

Next machine-floating value, double.

Prototype

```
double nextafter(double x,
                 double y);
```

Parameters

Parameter	Description
x	Value to step from.
y	Director to step in.

Return value

Next machine-floating value after x in direction of y.

nextafterf()

Description

Next machine-floating value, float.

Prototype

```
float nextafterf(float x,
                 float y);
```

Parameters

Parameter	Description
x	Value to step from.
y	Director to step in.

Return value

Next machine-floating value after x in direction of y.

nextafterl()

Description

Next machine-floating value, long double.

Prototype

```
long double nextafterl(long double x,
                       long double y);
```

Parameters

Parameter	Description
x	Value to step from.
y	Director to step in.

Return value

Next machine-floating value after x in direction of y.

nexttoward()

Description

Next machine-floating value, double.

Prototype

```
double nexttoward(double      x,
                  long double y);
```

Parameters

Parameter	Description
x	Value to step from.
y	Direction to step in.

Return value

Next machine-floating value after x in direction of y.

nexttowardf()

Description

Next machine-floating value, float.

Prototype

```
float nexttowardf(float       x,
                  long double y);
```

Parameters

Parameter	Description
x	Value to step from.
y	Direction to step in.

Return value

Next machine-floating value after x in direction of y.

nexttowardl()

Description

Next machine-floating value, long double.

Prototype

```
long double nexttowardl(long double x,
                        long double y);
```

Parameters

Parameter	Description
x	Value to step from.
y	Direction to step in.

Return value

Next machine-floating value after x in direction of y.

nan()

Description

Parse NaN, double.

Prototype

```
double nan(const char * tag);
```

Parameters

Parameter	Description
tag	NaN tag.

Return value

Quiet NaN formed from tag.

nanf()

Description

Parse NaN, float.

Prototype

```
float nanf(const char * tag);
```

Parameters

Parameter	Description
tag	NaN tag.

Return value

Quiet NaN formed from tag.

nanl()

Description

Parse NaN, long double.

Prototype

```
long double nanl(const char * tag);
```

Parameters

Parameter	Description
tag	NaN tag.

Return value

Quiet NaN formed from tag.

copysign()

Description

Copy sign, double.

Prototype

```
double copysign(double x,
                double y);
```

Parameters

Parameter	Description
x	Floating value to inject sign into.
y	Floating value carrying the sign to inject.

Return value

x with the sign of y.

copysignf()

Description

Copy sign, float.

Prototype

```
float copysignf(float x,
                float y);
```

Parameters

Parameter	Description
x	Floating value to inject sign into.
y	Floating value carrying the sign to inject.

Return value

x with the sign of y.

copysignl()

Description

Copy sign, long double.

Prototype

```
long double copysignl(long double x,
                      long double y);
```

Parameters

Parameter	Description
x	Floating value to inject sign into.
y	Floating value carrying the sign to inject.

Return value

x with the sign of y.

5.6.13 <setjmp.h>

Non-local flow control

setjmp()

Description

Save calling environment for non-local jump.

Prototype

```
int setjmp(jmp_buf buf);
```

Parameters

Parameter	Description
buf	Buffer to save context into.

Return value

On return from a direct invocation, returns the value zero. On return from a call to the *longjmp()* (page 631) function, returns a nonzero value determined by the call to *longjmp()* (page 631).

Additional information

Saves its calling environment in env for later use by the *longjmp()* (page 631) function.

The environment saved by a call to setjmp () consists of information sufficient for a call to the *longjmp()* (page 631) function to return execution to the correct block and invocation of that block, were it called recursively.

longjmp()

Description

Restores the saved environment.

Prototype

```
void longjmp(jmp_buf buf,
             int     val);
```

Parameters

Parameter	Description
buf	Buffer to restore context from.
val	Value to return to *setjmp()* (page 630) call.

Additional information

Restores the environment saved by *setjmp()* (page 630) in the corresponding env argument. If there has been no such invocation, or if the function containing the invocation of *setjmp()* (page 630) has terminated execution in the interim, the behavior of *longjmp()* (page 631) is undefined.

After *longjmp()* (page 631) is completed, program execution continues as if the corresponding invocation of *setjmp()* (page 630) had just returned the value specified by val.

Objects of automatic storage allocation that are local to the function containing the invocation of the corresponding *setjmp()* (page 630) that do not have volatile-qualified type and have been changed between the *setjmp()* (page 630) invocation and *longjmp()* (page 631) call are indeterminate.

Notes

longjmp() (page 631) cannot cause *setjmp()* (page 630) to return the value 0; if val is 0, *setjmp()* (page 630) returns the value 1.

5.6.14 <stdalign.h>

Macros

Macro	Description
alignas	Expands to the _Alignas keyword.
alignof	Expands to the _Alignof keyword.
__alignas_is_defined	Always defined as 1.
__alignof_is_defined	Always defined as 1.

5.6.15 <stdbool.h>

Macros

bool

Description

Macros expanding to support the Boolean type.

Definition

```
#define bool     _Bool
#define true     1
#define false    0
```

Symbols

Definition	Description
bool	Underlying boolean type
true	Boolean true value
false	Boolean false value

5.6.16 <stddef.h>

Macros

NULL

Description

Null-pointer constant.

Definition

```
#define NULL    0
```

Symbols

Definition	Description
NULL	Null pointer

offsetof

Description

Calculate offset of member from start of structure.

Definition

```
#define offsetof(s,m)    ((size_t)&(((s *)0)->m))
```

Symbols

Definition	Description
offsetof(s,m)	Offset of m within s

Types

size_t

Description

Unsigned integral type returned by the sizeof operator.

Type definition

```
typedef __SEGGER_RTL_SIZE_T size_t;
```

ptrdiff_t

Description

Signed integral type of the result of subtracting two pointers.

Type definition

```
typedef __SEGGER_RTL_PTRDIFF_T ptrdiff_t;
```

wchar_t

Description

Integral type that can hold one wide character.

Type definition

```
typedef __SEGGER_RTL_WCHAR_T wchar_t;
```

5.6.17 <stdint.h>

Minima and maxima

Signed integer minima and maxima

Description

Minimum and maximum values for signed integer types.

Definition

```
#define INT8_MIN     (-128)
#define INT8_MAX     127
#define INT16_MIN    (-32767-1)
#define INT16_MAX    32767
#define INT32_MIN    (-2147483647L-1)
#define INT32_MAX    2147483647L
#define INT64_MIN    (-9223372036854775807LL-1)
#define INT64_MAX    9223372036854775807LL
```

Symbols

Definition	Description
INT8_MIN	Minimum value of *int8_t* (page **??**)
INT8_MAX	Maximum value of *int8_t* (page **??**)
INT16_MIN	Minimum value of *int16_t* (page **??**)
INT16_MAX	Maximum value of *int16_t* (page **??**)
INT32_MIN	Minimum value of *int32_t* (page **??**)
INT32_MAX	Maximum value of *int32_t* (page **??**)
INT64_MIN	Minimum value of *int64_t* (page **??**)
INT64_MAX	Maximum value of *int64_t* (page **??**)

Unsigned integer minima and maxima

Description

Minimum and maximum values for unsigned integer types.

Definition

```
#define UINT8_MAX     255
#define UINT16_MAX    65535
#define UINT32_MAX    4294967295UL
#define UINT64_MAX    18446744073709551615ULL
```

Symbols

Definition	Description
UINT8_MAX	Maximum value of *uint8_t* (page **??**)
UINT16_MAX	Maximum value of *uint16_t* (page **??**)
UINT32_MAX	Maximum value of *uint32_t* (page **??**)
UINT64_MAX	Maximum value of *uint64_t* (page **??**)

Maximal integer minima and maxima

Description

Minimum and maximum values for signed and unsigned maximal-integer types.

Definition

```
#define INTMAX_MIN     INT64_MIN
#define INTMAX_MAX     INT64_MAX
#define UINTMAX_MAX    UINT64_MAX
```

Symbols

Definition	Description
INTMAX_MIN	Minimum value of *intmax_t* (page **??**)
INTMAX_MAX	Maximum value of *intmax_t* (page **??**)
UINTMAX_MAX	Maximum value of *uintmax_t* (page **??**)

Least integer minima and maxima

Description

Minimum and maximum values for signed and unsigned least-integer types.

Definition

```
#define INT_LEAST8_MIN      INT8_MIN
#define INT_LEAST8_MAX      INT8_MAX
#define INT_LEAST16_MIN     INT16_MIN
#define INT_LEAST16_MAX     INT16_MAX
#define INT_LEAST32_MIN     INT32_MIN
#define INT_LEAST32_MAX     INT32_MAX
#define INT_LEAST64_MIN     INT64_MIN
#define INT_LEAST64_MAX     INT64_MAX
#define UINT_LEAST8_MAX     UINT8_MAX
#define UINT_LEAST16_MAX    UINT16_MAX
#define UINT_LEAST32_MAX    UINT32_MAX
#define UINT_LEAST64_MAX    UINT64_MAX
```

Symbols

Definition	Description
INT_LEAST8_MIN	Minimum value of *int_least8_t* (page **??**)
INT_LEAST8_MAX	Maximum value of *int_least8_t* (page **??**)
INT_LEAST16_MIN	Minimum value of *int_least16_t* (page **??**)
INT_LEAST16_MAX	Maximum value of *int_least16_t* (page **??**)
INT_LEAST32_MIN	Minimum value of *int_least32_t* (page **??**)
INT_LEAST32_MAX	Maximum value of *int_least32_t* (page **??**)
INT_LEAST64_MIN	Minimum value of *int_least64_t* (page **??**)
INT_LEAST64_MAX	Maximum value of *int_least64_t* (page **??**)
UINT_LEAST8_MAX	Maximum value of *uint_least8_t* (page **??**)
UINT_LEAST16_MAX	Maximum value of *uint_least16_t* (page **??**)
UINT_LEAST32_MAX	Maximum value of *uint_least32_t* (page **??**)
UINT_LEAST64_MAX	Maximum value of *uint_least64_t* (page **??**)

Fast integer minima and maxima

Description

Minimum and maximum values for signed and unsigned fast-integer types.

Definition

```
#define INT_FAST8_MIN      INT8_MIN
#define INT_FAST8_MAX      INT8_MAX
#define INT_FAST16_MIN     INT32_MIN
#define INT_FAST16_MAX     INT32_MAX
#define INT_FAST32_MIN     INT32_MIN
#define INT_FAST32_MAX     INT32_MAX
#define INT_FAST64_MIN     INT64_MIN
#define INT_FAST64_MAX     INT64_MAX
#define UINT_FAST8_MAX     UINT8_MAX
#define UINT_FAST16_MAX    UINT32_MAX
#define UINT_FAST32_MAX    UINT32_MAX
#define UINT_FAST64_MAX    UINT64_MAX
```

Symbols

Definition	Description
INT_FAST8_MIN	Minimum value of *int_fast8_t* (page **??**)
INT_FAST8_MAX	Maximum value of *int_fast8_t* (page **??**)
INT_FAST16_MIN	Minimum value of *int_fast16_t* (page **??**)
INT_FAST16_MAX	Maximum value of *int_fast16_t* (page **??**)
INT_FAST32_MIN	Minimum value of *int_fast32_t* (page **??**)
INT_FAST32_MAX	Maximum value of *int_fast32_t* (page **??**)
INT_FAST64_MIN	Minimum value of *int_fast64_t* (page **??**)
INT_FAST64_MAX	Maximum value of *int_fast64_t* (page **??**)
UINT_FAST8_MAX	Maximum value of *uint_fast8_t* (page **??**)
UINT_FAST16_MAX	Maximum value of *uint_fast16_t* (page **??**)
UINT_FAST32_MAX	Maximum value of *uint_fast32_t* (page **??**)
UINT_FAST64_MAX	Maximum value of *uint_fast64_t* (page **??**)

Pointer types minima and maxima

Description

Minimum and maximum values for pointer-related types.

Definition

```
#define PTRDIFF_MIN    INT64_MIN
#define PTRDIFF_MAX    INT64_MAX
#define SIZE_MAX       INT64_MAX
#define INTPTR_MIN     INT64_MIN
#define INTPTR_MAX     INT64_MAX
#define UINTPTR_MAX    UINT64_MAX
```

Symbols

Definition	Description
PTRDIFF_MIN	Minimum value of *ptrdiff_t* (page **??**)
PTRDIFF_MAX	Maximum value of *ptrdiff_t* (page **??**)
SIZE_MAX	Maximum value of *size_t* (page **??**)
INTPTR_MIN	Minimum value of *intptr_t* (page **??**)
INTPTR_MAX	Maximum value of *intptr_t* (page **??**)
UINTPTR_MAX	Maximum value of *uintptr_t* (page **??**)
PTRDIFF_MIN	Minimum value of *ptrdiff_t* (page **??**)
PTRDIFF_MAX	Maximum value of *ptrdiff_t* (page **??**)
SIZE_MAX	Maximum value of *size_t* (page **??**)
INTPTR_MIN	Minimum value of *intptr_t* (page **??**)
INTPTR_MAX	Maximum value of *intptr_t* (page **??**)
UINTPTR_MAX	Maximum value of *uintptr_t* (page **??**)

Wide integer minima and maxima

Description

Minimum and maximum values for the wint_t type.

Definition

```
#define WINT_MIN    (-2147483647L-1)
#define WINT_MAX    2147483647L
```

Symbols

Definition	Description
WINT_MIN	Minimum value of wint_t
WINT_MAX	Maximum value of wint_t

Constant construction macros

Signed integer construction macros

Description

Macros that create constants of type intx_t.

Definition

```
#define INT8_C(x)     (x)
#define INT16_C(x)    (x)
#define INT32_C(x)    (x)
#define INT64_C(x)    (x##LL)
```

Symbols

Definition	Description
INT8_C(x)	Create constant of type *int8_t* (page **??**)
INT16_C(x)	Create constant of type *int16_t* (page **??**)
INT32_C(x)	Create constant of type *int32_t* (page **??**)
INT64_C(x)	Create constant of type *int64_t* (page **??**)

Unsigned integer construction macros

Description

Macros that create constants of type uintx_t.

Definition

```
#define UINT8_C(x)     (x##u)
#define UINT16_C(x)    (x##u)
#define UINT32_C(x)    (x##u)
#define UINT64_C(x)    (x##uLL)
```

Symbols

Definition	Description
UINT8_C(x)	Create constant of type *uint8_t* (page **??**)
UINT16_C(x)	Create constant of type *uint16_t* (page **??**)
UINT32_C(x)	Create constant of type *uint32_t* (page **??**)
UINT64_C(x)	Create constant of type *uint64_t* (page **??**)

Maximal integer construction macros

Description

Macros that create constants of type *intmax_t* (page **??**) and uintmax_t.

Definition

```
#define INTMAX_C(x)      (x##LL)
#define UINTMAX_C(x)     (x##uLL)
```

Symbols

Definition	Description
INTMAX_C(x)	Create constant of type *intmax_t* (page **??**)
UINTMAX_C(x)	Create constant of type *uintmax_t* (page **??**)

5.6.18 <stdio.h>

Formatted output control strings

The functions in this section that accept a formatted output control string do so according to the specification that follows.

Composition

The format is composed of zero or more directives: ordinary characters (not %, which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments, converting them, if applicable, according to the corresponding conversion specifier, and then writing the result to the output stream.

Each conversion specification is introduced by the character %. After the % the following appear in sequence:

- Zero or more flags (in any order) that modify the meaning of the conversion specification.
- An optional minimum field width. If the converted value has fewer characters than the field width, it is padded with spaces (by default) on the left (or right, if the left adjustment flag has been given) to the field width. The field width takes the form of an asterisk * or a decimal integer.
- An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversions, the number of digits to appear after the decimal-point character for e, E, f, and F conversions, the maximum number of significant digits for the g and G conversions, or the maximum number of bytes to be written for s conversions. The precision takes the form of a period . followed either by an asterisk * or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional length modifier that specifies the size of the argument.
- A conversion specifier character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both, may be indicated by an asterisk. In this case, an int argument supplies the field width or precision. The arguments specifying field width, or precision, or both, must appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

Flag characters

The flag characters and their meanings are:

Flag	Description
•	The result of the conversion is left-justified within the field. The default, if this flag is not specified, is that the result of the conversion is left-justified within the field.
•	The result of a signed conversion always begins with a plus or minus sign. The default, if this flag is not specified, is that it begins with a sign only when a negative value is converted.
space	If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space is prefixed to the result. If the space and + flags both appear, the space flag is ignored.
#	The result is converted to an alternative form. For o conversion, it increases the precision, if and only if necessary, to force the first digit of the result to be a zero (if the value and precision are both zero, a single 0 is printed). For x or X conversion, a nonzero result has 0x or 0X prefixed to it. For e, E, f, F, g, and G conversions, the result of converting a floating-point number always contains a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if a digit follows it.) For g and F conversions, trailing zeros are not removed from the result. As an extension, when used in p conversion, the results has # prefixed to it. For other conversions, the behavior is undefined.
0	For d, i, o, u, x, X, e, E, f, F, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the 0 and - flags both appear, the 0 flag is ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag is ignored. For other conversions, the behavior is undefined.

Length modifiers

The length modifiers and their meanings are:

Flag	Description
hh	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a signed char or unsigned char argument (the argument will have been promoted according to the integer promotions, but its value will be converted to signed char or unsigned char before printing); or that a following n conversion specifier applies to a pointer to a signed char argument.
h	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a short int or unsigned short int argument (the argument will have been promoted according to the integer promotions, but its value is converted to short int or unsigned short int before printing); or that a following n conversion specifier applies to a pointer to a short int argument.
l	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long int or unsigned long int argument; that a following n conversion specifier applies to a pointer to a long int argument; or has no effect on a following e, E, f, F, g, or G conversion specifier.
ll	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long long int or unsigned long long int argument; that a following n conversion specifier applies to a pointer to a long long int argument.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

Conversion specifiers

The conversion specifiers and their meanings are:

Flag	Description
d, i	The argument is converted to signed decimal in the style [-]dddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading spaces. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
o, u, x, X	The unsigned argument is converted to unsigned octal for o, unsigned decimal for u, or unsigned hexadecimal notation for x or X in the style dddd the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading spaces. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
f, F	A double argument representing a floating-point number is converted to decimal notation in the style [-]ddd.ddd, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits. A double argument representing an infinity is converted to inf. A double argument representing a NaN is converted to nan. The F conversion specifier produces INF or NAN instead of inf or nan, respectively.
e, E	A double argument representing a floating-point number is converted in the style [-]d.ddde±dd, where there is one digit (which is nonzero if the argument is nonzero) before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The E conversion specifier produces a number with E instead of e introducing the exponent. The exponent always contains at least two digits, and only as many more digits as necessary to represent the exponent. If the value is zero, the exponent is zero. A double argument representing an infinity is converted to inf. A double argument representing a NaN is converted to nan. The E conversion specifier produces INF or NAN instead of inf or nan, respectively.
g, G	A double argument representing a floating-point number is converted in style f or e (or in style F or e in the case of a G conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted; style e (or E) is used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result unless the # flag is specified; a decimal-point character appears only if it is followed by a digit. A double argument representing an infinity is converted to inf. A double argument representing a NaN is converted to nan. The G conversion specifier produces INF or NAN instead of inf or nan, respectively.
c	The argument is converted to an unsigned char, and the resulting character is written.
s	The argument is be a pointer to the initial element of an array of character type. Characters from the array are written up to (but not including) the terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array must contain a null character.
p	The argument is a pointer to void. The value of the pointer is converted in the same format as the x conversion specifier with a fixed precision of 2*sizeof(void *).
n	The argument is a pointer to a signed integer into which is written the number of characters written to the output stream so far by the call to the formatting function. No argument is converted, but one is consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
%	A % character is written. No argument is converted.

Note that the C99 width modifier l used in conjunction with the c and s conversion specifiers is not supported and nor are the conversion specifiers a and A.

Formatted input control strings

The format is composed of zero or more directives: one or more white-space characters, an ordinary character (neither % nor a white-space character), or a conversion specification.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- An optional assignment-suppressing character *.
- An optional nonzero decimal integer that specifies the maximum field width (in characters).
- An optional length modifier that specifies the size of the receiving object.
- A conversion specifier character that specifies the type of conversion to be applied.

The formatted input function executes each directive of the format in turn. If a directive fails, the function returns. Failures are described as input failures (because of the occurrence of an encoding error or the unavailability of input characters), or matching failures (because of inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary character is executed by reading the next characters of the stream. If any of those characters differ from the ones composing the directive, the directive fails and the differing and subsequent characters remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a character from being read, the directive fails.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

- Input white-space characters (as specified by the *isspace()* (page 520) function) are skipped, unless the specification includes a [, c, or n specifier.
- An input item is read from the stream, unless the specification includes an n specifier. An input item is defined as the longest sequence of input characters which does not exceed any specified field width and which is, or is a prefix of, a matching input sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails; this condition is a matching failure unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.
- Except in the case of a % specifier, the input item (or, in the case of a %n directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a *, the result of the conversion is placed in the object pointed to by the first argument following the format argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the object, the behavior is undefined.

Length modifiers

The length modifiers and their meanings are:

Flag	Description
hh	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to signed char or pointer to unsigned char.
h	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short int or unsigned short int.
l	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long int or unsigned long int; that a following e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to double.
ll	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long long int or unsigned long long int.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined. Note that the C99 length modifiers j, z, and t are not supported.

Conversion specifiers

Flag	Description
d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtol()* (page 667) function with the value 10 for the base argument. The corresponding argument must be a pointer to signed integer.
i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the *strtol()* (page 667) function with the value zero for the base argument. The corresponding argument must be a pointer to signed integer.
o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the *strtol()* (page 667) function with the value 18 for the base argument. The corresponding argument must be a pointer to signed integer.
u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtoul()* (page 669) function with the value 10 for the base argument. The corresponding argument must be a pointer to unsigned integer.
x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the *strtoul()* (page 669) function with the value 16 for the base argument. The corresponding argument must be a pointer to unsigned integer.
e, f, g	Matches an optionally signed floating-point number whose format is the same as expected for the subject sequence of the *strtod()* (page 671) function. The corresponding argument shall be a pointer to floating.
c	Matches a sequence of characters of exactly the number specified by the field width (one if no field width is present in the directive). The corresponding argument must be a pointer to the initial element of a character array large enough to accept the sequence. No null character is added.
s	Matches a sequence of non-white-space characters The corresponding argument must be a pointer to the initial element of a character array large enough to accept the sequence and a terminating null character, which will be added automatically.
[	Matches a nonempty sequence of characters from a set of expected characters (the scanset). The corresponding argument must be a pointer to the initial element of a character array large enough to accept the sequence and a terminating null character, which will be added automatically. The conversion specifier includes all subsequent characters in the format string, up to and including the matching right bracket]. The characters between the brackets (the scanlist) compose the scanset, unless the character after the left bracket is a circumflex ^, in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with [] or[^], the right bracket character is in the scanlist and the next following right bracket character is the matching right bracket that ends the specification; otherwise the first following right bracket character is the one that ends the specification. If a - character is in the scanlist and is not the first, nor the second where the first character is a ^, nor the last character, it is treated as a member of the scanset.
p	Reads a sequence output by the corresponding %p formatted output conversion. The corresponding argument must be a pointer to a pointer to void.
n	No input is consumed. The corresponding argument shall be a pointer to signed integer into which is to be written the number of characters read from the input stream so far by this call to the formatted input function. Execution of a %n directive does not increment the assignment count returned at the completion of execution of the fscanf function. No argument is converted, but one is consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.
%	Matches a single % character; no conversion or assignment occurs.

Note that the C99 width modifier l used in conjunction with the c, s, and [conversion specifiers is not supported and nor are the conversion specifiers a and A.

Character and string I/O functions

Function	Description
getchar() (page 644)	Read character from standard input.
gets() (page 644)	Read string from standard input.
putc() (page 645)	Write character to file.
putchar() (page 645)	Write character to standard output.
puts() (page 645)	Write string to standard output.

getchar()

Description

Read character from standard input.

Prototype

```
int getchar(void);
```

Return value

If the stream is at end-of-file or a read error occurs, returns EOF, otherwise a nonnegative value.

Additional information

Reads a single character from the standard input stream.

gets()

Description

Read string from standard input.

Prototype

```
char *gets(char * s);
```

Parameters

Parameter	Description
s	Pointer to object that receives the string.

Return value

Returns s if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, the array contents are indeterminate and a null null pointer is return.

Additional information

This function reads characters from standard input into the array pointed to by s until end-of-file is encountered or a new-line character is read. Any new-line character is discarded, and a null character is written immediately after the last character read into the array.

putc()

Description

Write character to file.

Prototype

```
int putc(int     c,
         FILE * stream);
```

Parameters

Parameter	Description
c	Character to write.
stream	Pointer to stream to write to.

Return value

If no error, the character written. If a write error occurs, returns EOF.

Additional information

Writes the character c to stream.

putchar()

Description

Write character to standard output.

Prototype

```
int putchar(int c);
```

Parameters

Parameter	Description
c	Character to write.

Return value

If no error, the character written. If a write error occurs, returns EOF.

Additional information

Writes the character c to the standard output stream.

puts()

Description

Write string to standard output.

Prototype

```
int puts(const char * s);
```

Parameters

Parameter	Description
s	Pointer to zero-terminated string.

Return value

Returns EOF if a write error occurs; otherwise it returns a nonnegative value.

Additional information

Writes the string pointed to by s to the standard output stream using *putchar()* (page 645) and appends a new-line character to the output. The terminating null character is not written.

Formatted input functions

Function	Description
scanf() (page 646)	Formatted read from standard input.
sscanf() (page 647)	Formatted read from string.
vscanf() (page 647)	Formatted read from standard input, variadic.
vsscanf() (page 648)	Formatted read from string, variadic.

scanf()

Description

Formatted read from standard input.

Prototype

```
int scanf(const char * format,
                      ...);
```

Parameters

Parameter	Description
format	Pointer to zero-terminated format control string.

Return value

Returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Additional information

Reads input from the standard input stream under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

sscanf()

Description

Formatted read from string.

Prototype

```
int sscanf(const char * s,
           const char * format,
                        ...);
```

Parameters

Parameter	Description
s	Pointer to string to read from.
format	Pointer to zero-terminated format control string.

Return value

Returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Additional information

Reads input from the string s under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

vscanf()

Description

Formatted read from standard input, variadic.

Prototype

```
int vscanf(const char    * format,
           va_list         arg);
```

Parameters

Parameter	Description
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

Return value

Returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Additional information

Reads input from the standard input stream under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers

to the objects to receive the converted input. Before calling *vscanf()* (page 647), arg must be initialized by the va_start() macro (and possibly subsequent va_arg() calls). *vscanf()* (page 647) does not invoke the va_end() macro.

If there are insufficient arguments for the format, the behavior is undefined.

vsscanf()

Description

Formatted read from string, variadic.

Prototype

```
int vsscanf(const char    * s,
            const char    * format,
                  va_list   arg);
```

Parameters

Parameter	Description
s	Pointer to string to read from.
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

Return value

Returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Additional information

Reads input from the standard input stream under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input. Before calling *vsscanf()* (page 648), arg must be initialized by the va_start() macro (and possibly subsequent va_arg() calls). *vsscanf()* (page 648) does not invoke the va_end() macro.

If there are insufficient arguments for the format, the behavior is undefined.

Formatted output functions

Function	Description
printf() (page 649)	Formatted write to standard output.
sprintf() (page 649)	Formatted write to string.
snprintf() (page 650)	Formatted write to string, limit length.
vprintf() (page 650)	Formatted write to standard output, variadic.
vsprintf() (page 651)	Formatted write to string, variadic.
vsnprintf() (page 651)	Formatted write to string, limit length, variadic.

printf()

Description

Formatted write to standard output.

Prototype

```
int printf(const char * format,
                        ...);
```

Parameters

Parameter	Description
format	Pointer to zero-terminated format control string.

Return value

Returns the number of characters written, or a negative value if an output or encoding error occurred.

Additional information

Writes to the standard output stream under control of the string pointed to by format that specifies how subsequent arguments are converted for output.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

sprintf()

Description

Formatted write to string.

Prototype

```
int sprintf(      char * s,
            const char * format,
                         ...);
```

Parameters

Parameter	Description
s	Pointer to array that receives the formatted output.
format	Pointer to zero-terminated format control string.

Return value

Returns number of characters written to s (not counting the terminating null), or a negative value if an output or encoding error occurred.

Additional information

Writes to the string pointed to by s under control of the string pointed to by format that specifies how subsequent arguments are converted for output. A null character is written at the end of the characters written; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

snprintf()

Description

Formatted write to string, limit length.

Prototype

```
int snprintf(       char    * s,
                    size_t    n,
             const char    * format,
                               ...);
```

Parameters

Parameter	Description
s	Pointer to array that receives the formatted output.
n	Maximum number of characters to write to the array pointed to by s.
format	Pointer to zero-terminated format control string.

Return value

Returns the number of characters that would have been written had n been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than n.

Additional information

Writes to the string pointed to by s under control of the string pointed to by format that specifies how subsequent arguments are converted for output.

If n is zero, nothing is written, and s can be a null pointer. Otherwise, output characters beyond count n-1 are discarded rather than being written to the array, and a null character is written at the end of the characters actually written into the array. A null character is written at the end of the conversion; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

vprintf()

Description

Formatted write to standard output, variadic.

Prototype

```
int vprintf(const char    * format,
                  va_list   arg);
```

Parameters

Parameter	Description
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

Return value

Returns the number of characters written, or a negative value if an output or encoding error occurred.

Additional information

Writes to the standard output stream using under control of the string pointed to by format that specifies how subsequent arguments are converted for output. Before calling *vprintf()* (page 650), arg must be initialized by the va_start macro (and possibly subsequent va_arg calls). *vprintf()* (page 650) does not invoke the va_end macro.

vsprintf()

Description

Formatted write to string, variadic.

Prototype

```
int vsprintf(      char     * s,
             const char     * format,
                   va_list    arg);
```

Parameters

Parameter	Description
s	Pointer to array that receives the formatted output.
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

Return value

Returns number of characters written to s (not counting the terminating null), or a negative value if an output or encoding error occurred.

Additional information

Writes to the string pointed to by s under control of the string pointed to by format that specifies how subsequent arguments are converted for output. A null character is written at the end of the characters written; it is not counted as part of the returned value.

Before calling *vsprintf()* (page 651), arg must be initialized by the va_start macro (and possibly subsequent va_arg calls). *vsprintf()* (page 651) does not invoke the va_end macro.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

Notes

This is equivalent to *sprintf()* (page 649) with the variable argument list replaced by arg.

vsnprintf()

Description

Formatted write to string, limit length, variadic.

Prototype

```
int vsnprintf(      char     * s,
                    size_t     n,
              const char     * format,
                    va_list    arg);
```

Parameters

Parameter	Description
s	Pointer to array that receives the formatted output.
n	Maximum number of characters to write to the array pointed to by s.
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

Return value

Returns the number of characters that would have been written had n been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than n.

Additional information

Writes to the string pointed to by s under control of the string pointed to by format that specifies how subsequent arguments are converted for output. Before calling *vsnprintf()* (page 651), arg must be initialized by the va_start macro (and possibly subsequent va_arg() calls). *vsnprintf()* (page 651) does not invoke the va_end macro.

If n is zero, nothing is written, and s can be a null pointer. Otherwise, output characters beyond count n-1 are discarded rather than being written to the array, and a null character is written at the end of the characters actually written into the array. A null character is written at the end of the conversion; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

Notes

This is equivalent to *snprintf()* (page 650) with the variable argument list replaced by arg.

5.6.19 <stdlib.h>

Process control functions

Function	Description
atexit() (page 652)	Set function to be called on exit.
abort() (page 653)	Abort execution.

atexit()

Description

Set function to be called on exit.

Prototype

```
int atexit(__SEGGER_RTL_exit_func fn);
```

Parameters

Parameter	Description
fn	Function to register.

Return value

= 0	Success registering function.
≠ 0	Did not register function.

Additional information

Registers function fn to be called when the application has exited. The functions registered with *atexit()* (page 652) are executed in reverse order of their registration.

abort()

Description

Abort execution.

Prototype

```
void abort(void);
```

Additional information

Calls exit() with the exit status -1.

Integer arithmetic functions

Function	Description
abs() (page 653)	Calculate absolute value, int.
labs() (page 654)	Calculate absolute value, long.
llabs() (page 654)	Calculate absolute value, long long.
div() (page 654)	Divide returning quotient and remainder, int.
ldiv() (page 655)	Divide returning quotient and remainder, long.
lldiv() (page 655)	Divide returning quotient and remainder, long long.

abs()

Description

Calculate absolute value, int.

Prototype

```
int abs(int Value);
```

Parameters

Parameter	Description
Value	Integer value.

Return value

The absolute value of the integer argument Value.

labs()

Description

Calculate absolute value, long.

Prototype

```
long int labs(long int Value);
```

Parameters

Parameter	Description
Value	Long integer value.

Return value

The absolute value of the long integer argument Value.

llabs()

Description

Calculate absolute value, long long.

Prototype

```
long long int llabs(long long int Value);
```

Parameters

Parameter	Description
Value	Long long integer value.

Return value

The absolute value of the long long integer argument Value.

div()

Description

Divide returning quotient and remainder, int.

Prototype

```
div_t div(int Numer,
          int Denom);
```

Parameters

Parameter	Description
Numer	Numerator.
Denom	Demoninator.

Return value

Returns a structure of type div_t comprising both the quotient and the remainder. The structures contain the members quot (the quotient) and rem (the remainder), each of which has the same type as the arguments Numer and Denom. If either part of the result cannot be represented, the behavior is undefined.

Additional information

This computes Numer divided by Denom and Numer modulo Denom in a single operation.

See also

div_t

ldiv()

Description

Divide returning quotient and remainder, long.

Prototype

```
ldiv_t ldiv(long Numer,
            long Denom);
```

Parameters

Parameter	Description
Numer	Numerator.
Denom	Demoninator.

Return value

Returns a structure of type ldiv_t comprising both the quotient and the remainder. The structures contain the members quot (the quotient) and rem (the remainder), each of which has the same type as the arguments Numer and Denom. If either part of the result cannot be represented, the behavior is undefined.

Additional information

This computes Numer divided by Denom and Numer modulo Denom in a single operation.

See also

ldiv_t

lldiv()

Description

Divide returning quotient and remainder, long long.

Prototype

```
lldiv_t lldiv(long long Numer,
              long long Denom);
```

Parameters

Parameter	Description
Numer	Numerator.
Denom	Demoninator.

Return value

Returns a structure of type lldiv_t comprising both the quotient and the remainder. The structures contain the members quot (the quotient) and rem (the remainder), each of which has the same type as the arguments Numer and Denom. If either part of the result cannot be represented, the behavior is undefined.

Additional information

This computes Numer divided by Denom and Numer modulo Denom in a single operation.

See also

lldiv_t

Pseudo-random sequence generation functions

Function	Description
rand() (page 656)	Return next random number in sequence.
srand() (page 656)	Set seed of random number sequence.

rand()

Description

Return next random number in sequence.

Prototype

```
int rand(void);
```

Return value

Returns the computed pseudo-random integer.

Additional information

This computes a sequence of pseudo-random integers in the range 0 to RAND_MAX.

See also

srand() (page 656)

srand()

Description

Set seed of random number sequence.

Prototype

```
void srand(unsigned s);
```

Parameters

Parameter	Description
s	New seed value for pseudo-random sequence.

Additional information

This uses the argument Seed as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to *rand()* (page 656). If *srand()* (page 656) is called with the same seed value, the same sequence of pseudo-random numbers is generated.

If *rand()* (page 656) is called before any calls to *srand()* (page 656) have been made, a sequence is generated as if *srand()* (page 656) is first called with a seed value of 1.

See also

rand() (page 656)

Memory allocation functions

Function	Description
malloc() (page 657)	Allocate space for single object.
calloc() (page 657)	Allocate space for multiple objects and zero them.
realloc() (page 658)	Resize or allocate memory space.
free() (page 658)	Free allocated memory for reuse.

malloc()

Description

Allocate space for single object.

Prototype

```
void *malloc(size_t sz);
```

Parameters

Parameter	Description
sz	Number of characters to allocate for the object.

Return value

Returns a null pointer if the space for the object cannot be allocated from free memory; if space for the object can be allocated, *malloc()* (page 657) returns a pointer to the start of the allocated space.

Additional information

Allocates space for an object whose size is specified by sz and whose value is indeterminate.

calloc()

Description

Allocate space for multiple objects and zero them.

Prototype

```
void *calloc(size_t nobj,
             size_t sz);
```

Parameters

Parameter	Description
nobj	Number of objects to allocate.
sz	Number of characters to allocate per object.

Return value

Returns a null pointer if the space for the object cannot be allocated from free memory; if space for the object can be allocated, *malloc()* (page 657) returns a pointer to the start of the allocated space.

Additional information

Allocates space for an array of nobj objects, each of whose size is sz. The space is initialized to all zero bits.

realloc()

Description

Resize or allocate memory space.

Prototype

```
void *realloc(void   * ptr,
              size_t   sz);
```

Parameters

Parameter	Description
ptr	Pointer to resize, or NULL to allocate.
sz	New size of object.

Return value

Returns a pointer to the new object (which may have the same value as a pointer to the old object), or a null pointer if the new object could not be allocated.

Additional information

Deallocates the old object pointed to by ptr and returns a pointer to a new object that has the size specified by sz. The contents of the new object is identical to that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values.

If ptr is a null pointer, *realloc()* (page 658) behaves like *malloc()* (page 657) for the specified size. If memory for the new object cannot be allocated, the old object is not deallocated and its value is unchanged.

If ptr does not match a pointer earlier returned by *calloc()* (page 657), *malloc()* (page 657), or *realloc()* (page 658), or if the space has been deallocated by a call to *free()* (page 658) or *realloc()* (page 658), the behavior is undefined.

free()

Description

Free allocated memory for reuse.

Prototype

```
void free(void * ptr);
```

Parameters

Parameter	Description
ptr	Pointer to object to free.

Additional information

Causes the space pointed to by ptr to be deallocated, that is, made available for further allocation. If ptr is a null pointer, no action occurs.

If ptr does not match a pointer earlier returned by *calloc()* (page 657), *malloc()* (page 657), or *realloc()* (page 658), or if the space has been deallocated by a call to *free()* (page 658) or *realloc()* (page 658), the behavior is undefined.

Search and sort functions

Function	Description
qsort() (page 659)	Sort array.
bsearch() (page 659)	Search sorted array.

qsort()

Description

Sort array.

Prototype

```
void qsort(void   * base,
           size_t   nmemb,
           size_t   sz,
           int      ( *compare)(const void * elem1 , const void * elem2 ));
```

Parameters

Parameter	Description
base	Pointer to the start of the array.
nmemb	Number of array elements.
sz	Number of characters per array element.
compare	Pointer to element comparison function.

Additional information

Sorts the array pointed to by base using the compare function. The array should have nmemb elements of sz bytes. The compare function should return a negative value if the first parameter is less than the second parameter, zero if the parameters are equal, and a positive value if the first parameter is greater than the second parameter.

bsearch()

Description

Search sorted array.

Prototype

```
void *bsearch
            (const void   * key,
             const void   * base,
                   size_t   nmemb,
                   size_t   sz,
                   int      ( *compare)(const void * elem1 , const void * elem2 ));
```

Parameters

Parameter	Description
key	Pointer to object to search for.
base	Pointer to the start of the array.
nmemb	Number of array elements.
sz	Number of characters per array element.
compare	Pointer to element comparison function.

Return value

= NULL	Key not found.
≠ NULL	Pointer to found object.

Additional information

Searches the array pointed to by base for the specified key and returns a pointer to the first entry that matches, or null if no match. The array should have nmemb elements of sz bytes and be sorted by the same algorithm as the compare function.

The compare function should return a negative value if the first parameter is less than second parameter, zero if the parameters are equal, and a positive value if the first parameter is greater than the second parameter.

Number to string conversions

Function	Description
itoa() (page 660)	Convert to string, int.
ltoa() (page 661)	Convert to string, long.
lltoa() (page 662)	Convert to string, long long.
utoa() (page 662)	Convert to string, unsigned.
ultoa() (page 663)	Convert to string, unsigned long.
ulltoa() (page 664)	Convert to string, unsigned long long.

itoa()

Description

Convert to string, int.

Prototype

```
char *itoa(int    val,
           char * buf,
           int    radix);
```

Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

Return value

Returns buf.

Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

ltoa() (page 661), *lltoa()* (page 662), *utoa()* (page 662), *ultoa()* (page 663), *ulltoa()* (page 664)

ltoa()

Description

Convert to string, long.

Prototype

```
char *ltoa(long   val,
           char * buf,
           int    radix);
```

Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

Return value

Returns buf.

Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

itoa() (page 660), *lltoa()* (page 662), *utoa()* (page 662), *ultoa()* (page 663), *ulltoa()* (page 664)

lltoa()

Description

Convert to string, long long.

Prototype

```
char *lltoa(long long   val,
            char      * buf,
            int         radix);
```

Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

Return value

Returns buf.

Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

itoa() (page 660), *ltoa()* (page 661), *utoa()* (page 662), *ultoa()* (page 663), *ulltoa()* (page 664)

utoa()

Description

Convert to string, unsigned.

Prototype

```
char *utoa(unsigned   val,
           char     * buf,
           int        radix);
```

Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

Return value

Returns buf.

Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

itoa() (page 660), *ltoa()* (page 661), *lltoa()* (page 662), *ultoa()* (page 663), *ulltoa()* (page 664)

ultoa()

Description

Convert to string, unsigned long.

Prototype

```
char *ultoa(unsigned long   val,
            char          * buf,
            int             radix);
```

Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

Return value

Returns buf.

Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

itoa() (page 660), *ltoa()* (page 661), *lltoa()* (page 662), *ulltoa()* (page 664), *utoa()* (page 662)

ulltoa()

Description

Convert to string, unsigned long long.

Prototype

```
char *ulltoa(unsigned long long   val,
             char                 * buf,
             int                    radix);
```

Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

Return value

Returns buf.

Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

itoa() (page 660), *ltoa()* (page 661), *lltoa()* (page 662), *ultoa()* (page 663), *utoa()* (page 662)

String to number conversions

Function	Description
atoi() (page 665)	Convert to number, int.
atol() (page 665)	Convert to number, long.
atoll() (page 666)	Convert to number, long long.
atof() (page 666)	Convert to number, double.
strtol() (page 667)	Convert to number, long.
strtoll() (page 668)	Convert to number, long long.
strtoul() (page 669)	Convert to number, unsigned long.
strtoull() (page 670)	Convert to number, unsigned long long.
strtof() (page 671)	Convert to number, float.
strtod() (page 671)	Convert to number, double.
strtold() (page 672)	Convert to number, long double.

atoi()

Description

Convert to number, int.

Prototype

```
int atoi(const char * nptr);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.

Return value

Returns the converted value, if any. If the value of the result cannot be represented, the behavior is undefined.

Additional information

Converts the initial portion of the string pointed to by nptr to an int representation.

atoi() (page 665) does not affect the value of errno on an error.

Notes

Except for the behavior on error, *atoi()* (page 665) is equivalent to (int)strtol(nptr, NULL, 10).

See also

strtol() (page 667)

atol()

Description

Convert to number, long.

Prototype

```
long int atol(const char * nptr);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.

Return value

Returns the converted value, if any. If the value of the result cannot be represented, the behavior is undefined.

Additional information

Converts the initial portion of the string pointed to by nptr to a long representation.

atol() (page 665) does not affect the value of errno on an error.

Notes

Except for the behavior on error, *atol()* (page 665) is equivalent to strtol(nptr, NULL, 10).

See also

strtol() (page 667)

atoll()

Description

Convert to number, long long.

Prototype

```
long long int atoll(const char * nptr);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.

Return value

Returns the converted value, if any. If the value of the result cannot be represented, the behavior is undefined.

Additional information

Converts the initial portion of the string pointed to by nptr to a long-long representation.

atoll() (page 666) does not affect the value of errno on an error.

Notes

Except for the behavior on error, *atoll()* (page 666) is equivalent to strtoll(nptr, NULL, 10).

See also

strtoll() (page 668)

atof()

Description

Convert to number, double.

Prototype

```
double atof(const char * nptr);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.

Return value

Returns the converted value, if any. If the value of the result cannot be represented, the behavior is undefined.

Additional information

Converts the initial portion of the string pointed to by nptr to an double representation.

atof() (page 666) does not affect the value of errno on an error.

Notes

Except for the behavior on error, *atof()* (page 666) is equivalent to (int)strtod(nptr, NULL).

See also

strtod() (page 671)

strtol()

Description

Convert to number, long.

Prototype

```
long strtol(const char  * nptr,
            char ** endptr,
            int     base);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.
base	Radix to use for conversion, 2 to 36.

Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, LONG_MIN or LONG_MAX is returned according to the sign of the value, if any, and the value of the macro ERANGE is stored in errno.

Additional information

Converts the initial portion of the string pointed to by nptr to a long representation.

First, *strtol()* (page 667) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by *isspace()* (page 520), a subject sequence resembling an integer represented in some radix determined by the value of base, and a final string of one or more unrecognized characters, including the terminating null character of the input string. *strtol()* (page 667) then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of base is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of base is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by base. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted.

If the value of base is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits, following the optional sign.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

strtoll()

Description

Convert to number, long long.

Prototype

```
long long strtoll(const char  * nptr,
                  char ** endptr,
                  int     base);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.
base	Radix to use for conversion, 2 to 36.

Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, LLONG_MIN or LLONG_MAX is returned according to the sign of the value, if any, and the value of the macro ERANGE is stored in errno.

Additional information

Converts the initial portion of the string pointed to by nptr to a long representation.

First, *strtoll()* (page 668) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by *isspace()* (page 520), a subject sequence resembling an integer represented in some radix determined by the value of base, and a final string of one or more unrecognized characters, including the terminating null character of the input string. *strtoll()* (page 668) then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of base is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of base is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by base. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted.

If the value of base is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits, following the optional sign.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

strtoul()

Description

Convert to number, unsigned long.

Prototype

```
unsigned long strtoul(const char  * nptr,
                      char ** endptr,
                      int     base);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.
base	Radix to use for conversion, 2 to 36.

Return value

strtoul() (page 669) returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, ULONG_MAX is and the value of the macro ERANGE is stored in errno.

Additional information

Converts the initial portion of the string pointed to by nptr to a long int representation.

First, *strtoul()* (page 669) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by *isspace()* (page 520), a subject sequence resembling an integer represented in some radix determined by the value of base, and a final string of one or more unrecognized characters, including the terminating null character of the input string. *strtoul()* (page 669) then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of base is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of base is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by base. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted.

If the value of base is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits, following the optional sign.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

strtoull()

Description

Convert to number, unsigned long long.

Prototype

```
unsigned long long strtoull(const char  * nptr,
                            char ** endptr,
                            int     base);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.
base	Radix to use for conversion, 2 to 36.

Return value

strtoull() (page 670) returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, ULLONG_MAX is and the value of the macro ERANGE is stored in errno.

Additional information

Converts the initial portion of the string pointed to by nptr to a long int representation.

First, *strtoull()* (page 670) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by *isspace()* (page 520), a subject sequence resembling an integer represented in some radix determined by the value of base, and a final string of one or more unrecognized characters, including the terminating null character of the input string. *strtoull()* (page 670) then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of base is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of base is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by base. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted.

If the value of base is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits, following the optional sign.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

strtof()

Description

Convert to number, float.

Prototype

```
float strtof(const char  * nptr,
             char ** endptr);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.

Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, HUGE_VALF is returned according to the sign of the value, if any, and the value of the macro ERANGE is stored in errno.

Additional information

Converts the initial portion of the string pointed to by nptr to float representation.

First, *strtof()* (page 671) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by *isspace()* (page 520), a subject sequence resembling a floating-point constant, and a final string of one or more unrecognized characters, including the terminating null character of the input string. *strtof()* (page 671) then attempts to convert the subject sequence to a floating-point number, and return the result.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

The expected form of the subject sequence is an optional plus or minus sign followed by a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

See also

strtod() (page 671)

strtod()

Description

Convert to number, double.

Prototype

```
double strtod(const char  * nptr,
              char ** endptr);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.

Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, HUGE_VAL is returned according to the sign of the value, if any, and the value of the macro ERANGE is stored in errno.

Additional information

Converts the initial portion of the string pointed to by nptr to double representation.

First, *strtod()* (page 671) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by *isspace()* (page 520), a subject sequence resembling a floating-point constant, and a final string of one or more unrecognized characters, including the terminating null character of the input string. *strtod()* (page 671) then attempts to convert the subject sequence to a floating-point number, and return the result.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

The expected form of the subject sequence is an optional plus or minus sign followed by a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

See also

strtof() (page 671)

strtold()

Description

Convert to number, long double.

Prototype

```
long double strtold(const char  * nptr,
                    char ** endptr);
```

Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.

Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, HUGE_VAL is returned according to the sign of the value, if any, and the value of the macro ERANGE is stored in errno.

Additional information

Converts the initial portion of the string pointed to by nptr to long double representation.

First, *strtold()* (page 672) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by *isspace()* (page 520), a subject sequence resembling a floating-point constant, and a final string of one or more unrecognized characters, including the terminating null character of the input string. *strtod()* (page 671) then attempts to convert the subject sequence to a floating-point number, and return the result.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

The expected form of the subject sequence is an optional plus or minus sign followed by a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

See also

strtod() (page 671)

Multi-byte/wide character functions

Function	Description
btowc() (page 673)	Convert single-byte character to wide character.
btowc_l() (page **??**)	Convert single-byte character to wide character, per locale, (POSIX.1).
mblen() (page 674)	Count number of bytes in multi-byte character.
mblen_l() (page **??**)	Count number of bytes in multi-byte character, per locale (POSIX.1).
mbtowc() (page 676)	Convert multi-byte character to wide character.
mbtowc_l() (page **??**)	Convert multi-byte character to wide character, per locale (POSIX.1).
mbstowcs() (page 677)	Convert multi-byte string to wide string.
mbstowcs_l() (page **??**)	Convert multi-byte string to wide string, per locale (POSIX.1).
mbsrtowcs() (page 678)	Convert multi-byte string to wide character string, restartable.
mbsrtowcs_l() (page **??**)	Convert multi-byte string to wide character string, restartable, per locale (POSIX.1).
wctomb() (page 680)	Convert wide character to multi-byte character.
wctomb_l() (page **??**)	Convert wide character to multi-byte character, per locale (POSIX.1).
wcstombs() (page 681)	Convert wide string to multi-byte string.
wcstombs_l() (page **??**)	Convert wide string to multi-byte string.

btowc()

Description

Convert single-byte character to wide character.

Prototype

```
wint_t btowc(int c);
```

Parameters

Parameter	Description
c	Character to convert.

Return value

Returns WEOF if c has the value EOF or if c, converted to an unsigned char and in the current locale, does not constitute a valid single-byte character in the initial shift state.

Additional information

Determines whether c constitutes a valid single-byte character in the current locale. If c is a valid single-byte character, *btowc()* (page 673) returns the wide character representation of that character.

btowc_l()

Description

Convert single-byte character to wide character, per locale, (POSIX.1).

Prototype

```
wint_t btowc_l(int c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Character to convert.
loc	Locale used for conversion.

Return value

Returns WEOF if c has the value EOF or if c, converted to an unsigned char and in the locale loc, does not constitute a valid single-byte character in the initial shift state.

Additional information

Determines whether c constitutes a valid single-byte character in the locale loc. If c is a valid single-byte character, *btowc_l()* (page **??**) returns the wide character representation of that character.

Notes

Conforms to POSIX.1-2008.

mblen()

Description

Count number of bytes in multi-byte character.

Prototype

```
int mblen(const char   * s,
          size_t   n);
```

Parameters

Parameter	Description
s	Pointer to multi-byte character.
n	Maximum number of bytes to examine.

Return value

If s is a null pointer, returns a nonzero or zero value, if multi-byte character encodings, respectively, do or do not have state-dependent encodings.

If s is not a null pointer, either returns 0 (if s points to the null character), or returns the number of bytes that are contained in the multi-byte character (if the next n or fewer bytes form a valid multi-byte character), or returns -1 (if they do not form a valid multi-byte character).

Additional information

Determines the number of bytes contained in the multi-byte character pointed to by s in the current locale.

Except that the conversion state of the *mbtowc()* (page 676) function is not affected, it is equivalent to

mbtowc(NULL, s, n);

See also

mblen_l() (page **??**), *mbtowc()* (page 676)

mblen_l()

Description

Count number of bytes in multi-byte character, per locale (POSIX.1).

Prototype

```
int mblen_l(const char   * s,
            size_t       n,
                         locale_t loc);
```

Parameters

Parameter	Description
s	Pointer to multi-byte character.
n	Maximum number of bytes to examine.
loc	Locale to use for conversion.

Return value

If s is a null pointer, returns a nonzero or zero value, if multi-byte character encodings, respectively, do or do not have state-dependent encodings in locale loc.

If s is not a null pointer, either returns 0 (if s points to the null character), or returns the number of bytes that are contained in the multi-byte character (if the next n or fewer bytes form a valid multi-byte character), or returns -1 (if they do not form a valid multi-byte character).

Additional information

Determines the number of bytes contained in the multi-byte character pointed to by s in the locale loc.

Except that the conversion state of the *mbtowc()* (page 676) function is not affected, it is equivalent to

mbtowc_l(NULL, s, n, loc);

Notes

Conforms to POSIX.1-2008.

See also

mblen() (page 674), *mbtowc()* (page 676)

mbtowc()

Description

Convert multi-byte character to wide character.

Prototype

```
int mbtowc(      wchar_t * pwc,
           const char    * s,
                 size_t    n);
```

Parameters

Parameter	Description
pwc	Pointer to object that receives the wide character.
s	Pointer to multi-byte character string.
n	Maximum number of bytes that will be examined.

Return value

If s is a null pointer, *mbtowc()* (page 676) returns a nonzero value if multi-byte character encodings are state-dependent in the current locale, and zero otherwise.

If s is not null and the object that s points to is a wide character null, *mbtowc()* (page 676) returns 0.

If s is not null and the object that s points to forms a valid multi-byte character, *mbtowc()* (page 676) returns the length in bytes of the multi-byte character.

If the object that *mbtowc()* (page 676) points to does not form a valid multi-byte character within the first n characters, it returns -1.

Additional information

Converts a single multi-byte character to a wide character in the current locale. The wide character, if the multi-byte character string is converted correctly, is stored into the object pointed to by pwc.

See also

mbtowc_l() (page **??**)

mbtowc_l()

Description

Convert multi-byte character to wide character, per locale (POSIX.1).

Prototype

```
int mbtowc_l(      wchar_t * pwc,
             const char    * s,
                   size_t    n,
                             locale_t loc);
```

Parameters

Parameter	Description
pwc	Pointer to object that receives the wide character.
s	Pointer to multi-byte character string.
n	Maximum number of bytes that will be examined.
loc	Locale used to convert the multi-byte character.

Return value

If s is a null pointer, *mbtowc_l()* (page **??**) returns a nonzero value if multi-byte character encodings are state-dependent in locale loc, and zero otherwise.

If s is not null and the object that s points to is a wide null character, *mbtowc_l()* (page **??**) returns 0.

If s is not null and the object that s points to forms a valid multi-byte character, *mbtowc_l()* (page **??**) returns the length in bytes of the multi-byte character.

If the object that *mbtowc_l()* (page **??**) points to does not form a valid multi-byte character within the first n characters, it returns -1.

Additional information

Converts a single multi-byte character to a wide character in the locale loc. The wide character, if the multi-byte character string is converted correctly, is stored into the object pointed to by pwc.

Notes

Conforms to POSIX.1-2008.

See also

mbtowc() (page 676)

mbstowcs()

Description

Convert multi-byte string to wide string.

Prototype

```
size_t mbstowcs(      wchar_t * pwcs,
                const char    * s,
                      size_t    n);
```

Parameters

Parameter	Description
pwcs	Pointer to array that receives the wide character string.
s	Pointer to array that contains the multi-byte string.
n	Maximum number of wide characters to write into pwcs.

Return value

Returns -1 if an invalid multi-byte character is encountered, otherwise returns the number of array elements modified (if any), not including a terminating null wide character.

Additional information

Converts a sequence of multi-byte characters, in the current locale, that begins in the initial shift state from the array pointed to by s into a sequence of corresponding wide characters and stores not more than n wide characters into the array pointed to by pwcs.

No multi-byte characters that follow a null character (which is converted into a null wide character) will be examined or converted. Each multi-byte character is converted as if by a call to the *mbtowc()* (page 676) function, except that the conversion state of the *mbtowc()* (page 676) function is not affected.

No more than n elements will be modified in the array pointed to by pwcs. If copying takes place between objects that overlap, the behavior is undefined.

mbstowcs_l()

Description

Convert multi-byte string to wide string, per locale (POSIX.1).

Prototype

```
size_t mbstowcs_l(      wchar_t * pwcs,
                  const char    * s,
                        size_t    n,
                                  locale_t loc);
```

Parameters

Parameter	Description
pwcs	Pointer to array that receives the wide character string.
s	Pointer to array that contains the multi-byte string.
n	Maximum number of wide characters to write into pwcs.
loc	Locale to use for conversion.

Return value

Returns -1 if an invalid multi-byte character is encountered, otherwise returns the number of array elements modified (if any), not including a terminating null wide character.

Additional information

Converts a sequence of multi-byte characters, in the locale loc, that begins in the initial shift state from the array pointed to by s into a sequence of corresponding wide characters and stores not more than n wide characters into the array pointed to by pwcs.

No multi-byte characters that follow a null character (which is converted into a null wide character) will be examined or converted. Each multi-byte character is converted as if by a call to the *mbtowc()* (page 676) function, except that the conversion state of the *mbtowc()* (page 676) function is not affected.

No more than n elements will be modified in the array pointed to by pwcs. If copying takes place between objects that overlap, the behavior is undefined.

Notes

Conforms to POSIX.1-2017.

mbsrtowcs()

Description

Convert multi-byte string to wide character string, restartable.

Prototype

```
size_t mbsrtowcs(      wchar_t     * dst,
                 const char       ** src,
                       size_t        len,
                       mbstate_t   * ps);
```

Parameters

Parameter	Description
dst	Pointer to object that receives the converted wide characters.
src	Pointer to pointer to multi-byte character string.
len	Maximum number of wide characters that will be written to dst.
ps	Pointer to multi-byte conversion state.

Return value

The number of wide characters written to dst (not including the eventual terminating null character).

Additional information

Converts a sequence of multi-byte characters, in the current locale, that begins in the conversion state described by the object pointed to by ps, from the array indirectly pointed to by src into a sequence of corresponding wide characters.

If dst is not a null pointer, the converted characters are stored into the array pointed to by dst. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multi-byte character, or (if dst is not a null pointer) when len wide characters have been stored into the array pointed to by dst. Each conversion takes place as if by a call to the *mbrtowc()* (page 732) function.

If dst is not a null pointer, the pointer object pointed to by src is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multi-byte character converted (if any). If conversion stopped due to reaching a terminating null character and if dst is not a null pointer, the resulting state described is the initial conversion state.

See also

mbsrtowcs_l() (page **??**), *mbrtowc()* (page 732)

mbsrtowcs_l()

Description

Convert multi-byte string to wide character string, restartable, per locale (POSIX.1).

Prototype

```
size_t mbsrtowcs_l(      wchar_t     * dst,
                   const char       ** src,
                         size_t        len,
                         mbstate_t   * ps,
                                       locale_t loc);
```

Parameters

Parameter	Description
dst	Pointer to object that receives the converted wide characters.
src	Pointer to pointer to multi-byte character string.
len	Maximum number of wide characters that will be written to dst.
ps	Pointer to multi-byte conversion state.
loc	Locale used for conversion.

Return value

The number of wide characters written to dst (not including the eventual terminating null character).

Additional information

Converts a sequence of multi-byte characters, in the locale loc, that begins in the conversion state described by the object pointed to by ps, from the array indirectly pointed to by src into a sequence of corresponding wide characters.

If dst is not a null pointer, the converted characters are stored into the array pointed to by dst. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multi-byte character, or (if dst is not a null pointer) when len wide characters have been stored into the array pointed to by dst. Each conversion takes place as if by a call to the *mbrtowc()* (page 732) function.

If dst is not a null pointer, the pointer object pointed to by src is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multi-byte character converted (if any). If conversion stopped due to reaching a terminating null character and if dst is not a null pointer, the resulting state described is the initial conversion state.

Notes

Conforms to POSIX.1-2008.

See also

mbsrtowcs() (page 678), *mbrtowc()* (page 732)

wctomb()

Description

Convert wide character to multi-byte character.

Prototype

```
int wctomb(char * s,
           wchar_t wc);
```

Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte character.
wc	Wide character to convert.

Return value

Returns the number of bytes stored in the array object. When wc is not a valid wide character, an encoding error occurs: *wctomb()* (page 680) stores the value of the macro EILSEQ in errno and returns (*size_t* (page **??**))(-1); the conversion state is unspecified.

Additional information

If s is a null pointer, *wctomb()* (page 680) is equivalent to the call wcrtomb(buf, 0, ps) where buf is an internal buffer.

If s is not a null pointer, *wctomb()* (page 680) determines the number of bytes needed to represent the multi-byte character that corresponds to the wide character given by wc in the current locale, and stores the multi-byte character representation in the array whose first element is pointed to by s. At most MB_CUR_MAX bytes are stored. If wc is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

wctomb_l()

Description

Convert wide character to multi-byte character, per locale (POSIX.1).

Prototype

```
int wctomb_l(char * s,
             wchar_t wc,
             locale_t loc);
```

Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte character.
wc	Wide character to convert.
loc	Locale used for conversion.

Return value

Returns the number of bytes stored in the array object. When wc is not a valid wide character, an encoding error occurs: *wctomb_l()* (page **??**) stores the value of the macro EILSEQ in errno and returns (*size_t* (page **??**))(-1); the conversion state is unspecified.

Additional information

If s is a null pointer, *wctomb_l()* (page **??**) is equivalent to the call wcrtomb_l(buf, 0, ps, loc) where buf is an internal buffer.

If s is not a null pointer, *wctomb_l()* (page **??**) determines the number of bytes needed to represent the multi-byte character that corresponds to the wide character given by wc in the locale loc, and stores the multi-byte character representation in the array whose first element is pointed to by s. At most MB_CUR_MAX bytes are stored. If wc is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

Notes

Conforms to POSIX.1-2008.

wcstombs()

Description

Convert wide string to multi-byte string.

Prototype

```
size_t wcstombs(      char    * s,
                const wchar_t * pwcs,
                      size_t    n);
```

Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte string.
pwcs	Pointer to wide character string to convert.
n	Maximum number of bytes to write into s.

Return value

If a wide character is encountered that does not correspond to a valid multibyte character in the current locale, returns (*size_t* (page **??**))(-1). Otherwise, returns the number of bytes written, not including a terminating null character (if any).

Additional information

Converts a sequence of wide characters in the current locale from the array pointed to by pwcs into a sequence of corresponding multi-byte characters that begins in the initial shift state, and stores these multi-byte characters into the array pointed to by s, stopping if a multi-byte character would exceed the limit of n total bytes or if a null character is stored. Each wide character is converted as if by a call to *wctomb()* (page 680), except that the conversion state of *wctomb()* (page 680) is not affected.

wcstombs_l()

Description

Convert wide string to multi-byte string.

Prototype

```
size_t wcstombs_l(      char    * s,
                  const wchar_t * pwcs,
                        size_t    n,
                                  locale_t loc);
```

Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte string.
pwcs	Pointer to wide character string to convert.
n	Maximum number of bytes to write into s.
loc	Locale used for conversion.

Return value

If a wide character is encountered that does not correspond to a valid multibyte character in the locale loc, returns (*size_t* (page **??**))(-1). Otherwise, returns the number of bytes written, not including a terminating null character (if any).

Additional information

Converts a sequence of wide characters in the locale loc from the array pointed to by pwcs into a sequence of corresponding multi-byte characters that begins in the initial shift state, and stores these multi-byte characters into the array pointed to by s, stopping if a multi-byte character would exceed the limit of n total bytes or if a null character is stored. Each wide character is converted as if by a call to *wctomb()* (page 680), except that the conversion state of *wctomb()* (page 680) is not affected.

5.6.20 <string.h>

The header file <string.h> defines functions that operate on arrays that are interpreted as null-terminated strings.

Various methods are used for determining the lengths of the arrays, but in all cases a char * or void * argument points to the initial (lowest addressed) character of the array. If an array is accessed beyond the end of an object, the behavior is undefined.

Where an argument declared as *size_t* (page **??**) n specifies the length of an array for a function, n can have the value zero on a call to that function. Unless explicitly stated otherwise in the description of a particular function, pointer arguments must have valid values on a call with a zero size. On such a call, a function that locates a character finds no occurrence, a function that compares two character sequences returns zero, and a function that copies characters copies zero characters.

Copying functions

Function	Description
memset() (page 683)	Set memory to character.
memcpy() (page 684)	Copy memory.
memccpy() (page 684)	Copy memory, specify terminator (POSIX.1).
mempcpy() (page 685)	Copy memory (GNU).
memmove() (page 685)	Copy memory, tolerate overlaps.
strcpy() (page 686)	Copy string.
strncpy() (page 686)	Copy string, limit length.
strlcpy() (page 687)	Copy string, limit length, always zero terminate (BSD).
stpcpy() (page 687)	Copy string, return end.
stpncpy() (page 688)	Copy string, limit length, return end.
strcat() (page 688)	Concatenate strings.
strncat() (page 689)	Concatenate strings, limit length.
strlcat() (page 689)	Concatenate strings, limit length, always zero terminate (BSD).
strdup() (page 690)	Duplicate string (POSIX.1).
strndup() (page 690)	Duplicate string, limit length (POSIX.1).

memset()

Description

Set memory to character.

Prototype

```
void *memset(void   * s,
             int      c,
             size_t   n);
```

Parameters

Parameter	Description
s	Pointer to destination object.
c	Character to copy.
n	Length of destination object in characters.

Return value

Returns s.

Additional information

Copies the value of c (converted to an unsigned char) into each of the first n characters of the object pointed to by s.

memcpy()

Description

Copy memory.

Prototype

```
void *memcpy(       void    * s1,
             const void    * s2,
                   size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of characters to copy.

Return value

Returns a pointer to the destination object.

Additional information

Copies n characters from the object pointed to by s2 into the object pointed to by s1. The behavior of *memcpy()* (page 684) is undefined if copying takes place between objects that overlap.

memccpy()

Description

Copy memory, specify terminator (POSIX.1).

Prototype

```
void *memccpy(       void    * s1,
              const void    * s2,
                    int       c,
                    size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
c	Character that terminates copy.
n	Maximum number of characters to copy.

Return value

Returns a pointer to the character immediately following c in s1, or NULL if c was not found in the first n characters of s2.

Additional information

Copies at most n characters from the object pointed to by s2 into the object pointed to by s1. The copying stops as soon as n characters are copied or the character c is copied into the destination object pointed to by s1.

The behavior of *memccpy()* (page 684) is undefined if copying takes place between objects that overlap.

Notes

Conforms to POSIX.1-2008.

mempcpy()

Description

Copy memory (GNU).

Prototype

```
void *mempcpy(      void    * s1,
              const void    * s2,
                    size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of characters to copy.

Return value

Returns a pointer to the character immediately following the final character written into s1.

Additional information

Copies n characters from the object pointed to by s2 into the object pointed to by s1. The behavior of *mempcpy()* (page 685) is undefined if copying takes place between objects that overlap.

Notes

This is an extension found in GNU libc.

memmove()

Description

Copy memory, tolerate overlaps.

Prototype

```
void *memmove(      void    * s1,
              const void    * s2,
                    size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of characters to copy.

Return value

Returns the value of s1.

Additional information

Copies n characters from the object pointed to by s2 into the object pointed to by s1 ensuring that if s1 and s2 overlap, the copy works correctly. Copying takes place as if the n characters from the object pointed to by s2 are first copied into a temporary array of n characters that does not overlap the objects pointed to by s1 and s2, and then the n characters from the temporary array are copied into the object pointed to by s1.

strcpy()

Description

Copy string.

Prototype

```
char *strcpy(       char * s1,
             const char * s2);
```

Parameters

Parameter	Description
s1	String to copy to.
s2	String to copy.

Return value

Returns the value of s1.

Additional information

Copies the string pointed to by s2 (including the terminating null character) into the array pointed to by s1. The behavior of *strcpy()* (page 686) is undefined if copying takes place between objects that overlap.

strncpy()

Description

Copy string, limit length.

Prototype

```
char *strncpy(      char   * s1,
              const char   * s2,
                    size_t   n);
```

Parameters

Parameter	Description
s1	String to copy to.
s2	String to copy.
n	Maximum number of characters to copy.

Return value

Returns the value of s1.

Additional information

Copies not more than n characters from the array pointed to by s2 to the array pointed to by s1. Characters that follow a null character in s2 are not copied. The behavior of *strncpy()* (page 686) is undefined if copying takes place

between objects that overlap. If the array pointed to by s2 is a string that is shorter than n characters, null characters are appended to the copy in the array pointed to by s1, until n characters in all have been written.

Notes

No null character is implicitly appended to the end of s1, so s1 will only be terminated by a null character if the length of the string pointed to by s2 is less than n.

strlcpy()

Description

Copy string, limit length, always zero terminate (BSD).

Prototype

```
size_t strlcpy(      char   * s1,
               const char   * s2,
                     size_t   n);
```

Parameters

Parameter	Description
s1	Pointer to string to copy to.
s2	Pointer to string to copy.
n	Maximum number of characters, including terminating null, in s1.

Return value

Returns the number of characters it tried to copy, which is the length of the string s2 or n, whichever is smaller.

Additional information

Copies up to n-1 characters from the string pointed to by s2 into the array pointed to by s1 and always terminates the result with a null character.

The behavior of *strlcpy()* (page 687) is undefined if copying takes place between objects that overlap.

Notes

Commonly found in BSD libraries and contrasts with *strncpy()* (page 686) in that the resulting string is always terminated with a null character.

stpcpy()

Description

Copy string, return end.

Prototype

```
char *stpcpy(      char * s1,
             const char * s2);
```

Parameters

Parameter	Description
s1	String to copy to.
s2	String to copy.

Return value

A pointer to the end of the string s1, i.e. the terminating null byte of the string s1, after s2 is copied to it.

Additional information

Copies the string pointed to by s2 (including the terminating null character) into the array pointed to by s1. The behavior of *stpcpy()* (page 687) is undefined if copying takes place between objects that overlap.

stpncpy()

Description

Copy string, limit length, return end.

Prototype

```
char *stpncpy(      char   * s1,
              const char   * s2,
                    size_t   n);
```

Parameters

Parameter	Description
s1	String to copy to.
s2	String to copy.
n	Maximum number of characters to copy.

Return value

stpncpy() (page 688) returns a pointer to the terminating null byte in s1 after it is copied to, or, if s1 is not null-terminated, s1+n.

Additional information

Copies not more than n characters from the array pointed to by s2 to the array pointed to by s1. Characters that follow a null character in s2 are not copied. The behavior of *strncpy()* (page 686) is undefined if copying takes place between objects that overlap. If the array pointed to by s2 is a string that is shorter than n characters, null characters are appended to the copy in the array pointed to by s1, until n characters in all have been written.

Notes

No null character is implicitly appended to the end of s1, so s1 will only be terminated by a null character if the length of the string pointed to by s2 is less than n.

strcat()

Description

Concatenate strings.

Prototype

```
char *strcat(      char * s1,
             const char * s2);
```

Parameters

Parameter	Description
s1	Zero-terminated string to append to.
s2	Zero-terminated string to append.

Return value

Returns the value of s1.

Additional information

Appends a copy of the string pointed to by s2 (including the terminating null character) to the end of the string pointed to by s1. The initial character of s2 overwrites the null character at the end of s1. The behavior of *strcat()* (page 688) is undefined if copying takes place between objects that overlap.

strncat()

Description

Concatenate strings, limit length.

Prototype

```
char *strncat(      char   * s1,
              const char   * s2,
                    size_t   n);
```

Parameters

Parameter	Description
s1	String to append to.
s2	String to append.
n	Maximum number of characters in s1.

Return value

Returns the value of s1.

Additional information

Appends not more than n characters from the array pointed to by s2 to the end of the string pointed to by s1. A null character in s1 and characters that follow it are not appended. The initial character of s2 overwrites the null character at the end of s1. A terminating null character is always appended to the result.

The behavior of *strncat()* (page 689) is undefined if copying takes place between objects that overlap.

strlcat()

Description

Concatenate strings, limit length, always zero terminate (BSD).

Prototype

```
size_t strlcat(      char   * s1,
               const char   * s2,
                     size_t   n);
```

Parameters

Parameter	Description
s1	Pointer to string to append to.
s2	Pointer to string to append.
n	Maximum number of characters, including terminating null, in s1.

Return value

Returns the number of characters it tried to copy, which is the sum of the lengths of the strings s1 and s2 or n, whichever is smaller.

Additional information

Appends no more than n-strlen(s1}-1 characters pointed to by s2 into the array pointed to by s1 and always terminates the result with a null character if n is greater than zero. Both the strings s1 and s2 must be terminated with a null character on entry to *strlcat()* (page 689) and a character position for the terminating null should be included in n.

The behavior of *strlcat()* (page 689) is undefined if copying takes place between objects that overlap.

Notes

Commonly found in BSD libraries.

strdup()

Description

Duplicate string (POSIX.1).

Prototype

```
char *strdup(const char * s1);
```

Parameters

Parameter	Description
s1	Pointer to string to duplicate.

Return value

Returns a pointer to the new string or a null pointer if the new string cannot be created. The returned pointer can be passed to *free()* (page 658).

Additional information

Duplicates the string pointed to by s1 by using *malloc()* (page 657) to allocate memory for a copy of s and then copyies s, including the terminating null, to that memory

Notes

Conforms to POSIX.1-2008 and SC22 TR 24731-2.

strndup()

Description

Duplicate string, limit length (POSIX.1).

Prototype

```
char *strndup(const char   * s,
              size_t   n);
```

Parameters

Parameter	Description
s	Pointer to string to duplicate.
n	Maximum number of characters to duplicate.

Return value

Returns a pointer to the new string or a null pointer if the new string cannot be created. The returned pointer can be passed to *free()* (page 658).

Additional information

Duplicates at most n characters from the the string pointed to by s by using *malloc()* (page 657) to allocate memory for a copy of s.

If the length of string pointed to by s is greater than n characters, only n characters will be duplicated. If n is greater than the length of the string pointed to by s, all characters in the string are copied into the allocated array including the terminating null character.

Notes

Conforms to POSIX.1-2008 and SC22 TR 24731-2.

Comparison functions

Function	Description
memcmp() (page 691)	Compare memory.
strcmp() (page 692)	Compare strings.
strncmp() (page 692)	Compare strings, limit length.
strcasecmp() (page 693)	Compare strings, ignore case (POSIX.1).
strncasecmp() (page 693)	Compare strings, ignore case, limit length (POSIX.1).

memcmp()

Description

Compare memory.

Prototype

```
int memcmp(const void   * s1,
           const void   * s2,
                 size_t   n);
```

Parameters

Parameter	Description
s1	Pointer to object #1.
s2	Pointer to object #2.
n	Number of characters to compare.

Return value

< 0	s1 is less than s2.
= 0	s1 is equal to s2.
> 0	s1 is greater than to s2.

Additional information

Compares the first n characters of the object pointed to by s1 to the first n characters of the object pointed to by s2. *memcmp()* (page 691) returns an integer greater than, equal to, or less than zero as the object pointed to by s1 is greater than, equal to, or less than the object pointed to by s2.

strcmp()

Description

Compare strings.

Prototype

```
int strcmp(const char * s1,
           const char * s2);
```

Parameters

Parameter	Description
s1	Pointer to string #1.
s2	Pointer to string #2.

Return value

Returns an integer greater than, equal to, or less than zero, if the null-terminated array pointed to by s1 is greater than, equal to, or less than the null-terminated array pointed to by s2.

strncmp()

Description

Compare strings, limit length.

Prototype

```
int strncmp(const char   * s1,
            const char   * s2,
                  size_t   n);
```

Parameters

Parameter	Description
s1	Pointer to string #1.
s2	Pointer to string #2.
n	Maximum number of characters to compare.

Return value

Returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2.

Additional information

Compares not more than n characters from the array pointed to by s1 to the array pointed to by s2. Characters that follow a null character are not compared.

strcasecmp()

Description

Compare strings, ignore case (POSIX.1).

Prototype

```
int strcasecmp(const char * s1,
               const char * s2);
```

Parameters

Parameter	Description
s1	Pointer to string #1.
s2	Pointer to string #2.

Return value

< 0	s1 is less than s2.
= 0	s1 is equal to s2.
> 0	s1 is greater than to s2.

Additional information

Compares the string pointed to by s1 to the string pointed to by s2 ignoring differences in case.

strcasecmp() (page 693) returns an integer greater than, equal to, or less than zero if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2.

Notes

Conforms to POSIX.1-2008.

strncasecmp()

Description

Compare strings, ignore case, limit length (POSIX.1).

Prototype

```
int strncasecmp(const char   * s1,
                const char   * s2,
                      size_t   n);
```

Parameters

Parameter	Description
s1	Pointer to string #1.
s2	Pointer to string #2.
n	Maximum number of characters to compare.

Return value

< 0	s1 is less than s2.
= 0	s1 is equal to s2.
> 0	s1 is greater than to s2.

Additional information

Compares not more than n characters from the array pointed to by s1 to the array pointed to by s2 ignoring differences in case. Characters that follow a null character are not compared.

strncasecmp() (page 693) returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2.

Notes

Conforms to POSIX.1-2008.

Search functions

Function	Description
memchr() (page 694)	Find character in memory, forward.
memrchr() (page 695)	Find character in memory, reverse (BSD).
memmem() (page 695)	Find memory in memory, forward (BSD).
strchr() (page 696)	Find character within string, forward.
strnchr() (page 696)	Find character within string, forward, limit length.
strrchr() (page 697)	Find character within string, reverse.
strlen() (page 697)	Calculate length of string.
strnlen() (page 698)	Calculate length of string, limit length (POSIX.1).
strstr() (page 698)	Find string within string, forward.
strnstr() (page 699)	Find string within string, forward, limit length (BSD).
strcasestr() (page 699)	Find string within string, forward, ignore case (BSD).
strncasestr() (page 700)	Find string within string, forward, ignore case, limit length (BSD).
strpbrk() (page 700)	Find first occurrence of characters within string.
strspn() (page 701)	Compute size of string prefixed by a set of characters.
strcspn() (page 701)	Compute size of string not prefixed by a set of characters.
strtok() (page 702)	Break string into tokens.
strtok_r() (page **??**)	Break string into tokens, reentrant (POSIX.1).
strsep() (page 703)	Break string into tokens (BSD).

memchr()

Description

Find character in memory, forward.

Prototype

```
void *memchr(const void   * s,
             int      c,
             size_t   n);
```

Parameters

Parameter	Description
s	Pointer to object to search.
c	Character to search for.
n	Number of characters in object to search.

Return value

= NULL	c does not occur in the object.
≠ NULL	Pointer to the located character.

Additional information

Locates the first occurrence of c (converted to an unsigned char) in the initial n characters (each interpreted as unsigned char) of the object pointed to by s. Unlike *strchr()* (page 696), *memchr()* (page 694) does not terminate a search when a null character is found in the object pointed to by s.

memrchr()

Description

Find character in memory, reverse (BSD).

Prototype

```
void *memrchr(const void   * s,
              int      c,
              size_t   n);
```

Parameters

Parameter	Description
s	Pointer to object to search.
c	Character to search for.
n	Number of characters in object to search.

Return value

Returns a pointer to the located character, or a null pointer if c does not occur in the string.

Additional information

Locates the last occurrence of c (converted to a char) in the string pointed to by s.

Notes

Commonly found in Linux and BSD C libraries.

memmem()

Description

Find memory in memory, forward (BSD).

Prototype

```
void *memmem(const void   * s1,
             size_t   n1,
       const void   * s2,
             size_t   n2);
```

Parameters

Parameter	Description
s1	Pointer to object to search.
n1	Number of characters to search in s1.
s2	Pointer to object to search for.
n2	Number of characters to search from s2.

Return value

= NULL	(s2, n2) does not occur in (s1, n1).
≠ NULL	Pointer to the first occurrence of (s2, n2) in (s1, n1).

Additional information

Locates the first occurrence of the octet string s2 of length n2 in the octet string s1 of length n1.

Notes

Commonly found in Linux and BSD C libraries.

strchr()

Description

Find character within string, forward.

Prototype

```
char *strchr(const char * s,
             int    c);
```

Parameters

Parameter	Description
s	String to search.
c	Character to search for.

Return value

Returns a pointer to the located character, or a null pointer if c does not occur in the string.

Additional information

Locates the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string.

strnchr()

Description

Find character within string, forward, limit length.

Prototype

```
char *strnchr(const char   * s,
              size_t   n,
              int      c);
```

Parameters

Parameter	Description
s	String to search.
n	Number of characters to search.
c	Character to search for.

Return value

Returns a pointer to the located character, or a null pointer if c does not occur in the string.

Additional information

Searches not more than n characters to locate the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string.

strrchr()

Description

Find character within string, reverse.

Prototype

```
char *strrchr(const char * s,
              int    c);
```

Parameters

Parameter	Description
s	String to search.
c	Character to search for.

Return value

Returns a pointer to the located character, or a null pointer if c does not occur in the string.

Additional information

Locates the last occurrence of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string.

strlen()

Description

Calculate length of string.

Prototype

```
size_t strlen(const char * s);
```

Parameters

Parameter	Description
s	Pointer to zero-terminated string.

Return value

Returns the length of the string pointed to by s, that is the number of characters that precede the terminating null character.

strnlen()

Description

Calculate length of string, limit length (POSIX.1).

Prototype

```
size_t strnlen(const char   * s,
               size_t   n);
```

Parameters

Parameter	Description
s	Pointer to string.
n	Maximum number of characters to examine.

Return value

Returns the length of the string pointed to by s, up to a maximum of n characters. *strnlen()* (page 698) only examines the first n characters of the string s.

Notes

Conforms to POSIX.1-2008.

strstr()

Description

Find string within string, forward.

Prototype

```
char *strstr(const char * s1,
             const char * s2);
```

Parameters

Parameter	Description
s1	String to search.
s2	String to search for.

Return value

Returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length, *strstr()* (page 698) returns s1.

Additional information

Locates the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2.

strnstr()

Description

Find string within string, forward, limit length (BSD).

Prototype

```
char *strnstr(const char   * s1,
              const char   * s2,
                    size_t   n);
```

Parameters

Parameter	Description
s1	String to search.
s2	String to search for.
n	Maximum number of characters to search for.

Return value

Returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length, *strnstr()* (page 699) returns s1.

Additional information

Searches at most n characters to locate the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2.

Notes

Commonly found in Linux and BSD C libraries.

strcasestr()

Description

Find string within string, forward, ignore case (BSD).

Prototype

```
char *strcasestr(const char * s1,
                 const char * s2);
```

Parameters

Parameter	Description
s1	String to search for.
s2	String to search.

Return value

Returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length, returns s1.

Additional information

Locates the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2 without regard to character case.

Notes

This extension is commonly found in Linux and BSD C libraries.

strncasestr()

Description

Find string within string, forward, ignore case, limit length (BSD).

Prototype

```
char *strncasestr(const char   * s1,
                  const char   * s2,
                        size_t   n);
```

Parameters

Parameter	Description
s1	String to search for.
s2	String to search.
n	Maximum number of characters to compare in s2.

Return value

Returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length, returns s1.

Additional information

Searches at most n characters to locate the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2 without regard to character case.

Notes

This extension is commonly found in Linux and BSD C libraries.

strpbrk()

Description

Find first occurrence of characters within string.

Prototype

```
char *strpbrk(const char * s1,
              const char * s2);
```

Parameters

Parameter	Description
s1	Pointer to string to search.
s2	Pointer to string to search for.

Return value

Returns a pointer to the first character, or a null pointer if no character from s2 occurs in s1.

Additional information

Locates the first occurrence in the string pointed to by s1 of any character from the string pointed to by s2.

strspn()

Description

Compute size of string prefixed by a set of characters.

Prototype

```
size_t strspn(const char * s1,
              const char * s2);
```

Parameters

Parameter	Description
s1	Pointer to zero-terminated string to search.
s2	Pointer to zero-terminated acceptable-set string.

Return value

Returns the length of the string pointed to by s1 which consists entirely of characters from the string pointed to by s2

Additional information

Computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of characters from the string pointed to by s2.

strcspn()

Description

Compute size of string not prefixed by a set of characters.

Prototype

```
size_t strcspn(const char * s1,
               const char * s2);
```

Parameters

Parameter	Description
s1	Pointer to string to search.
s2	Pointer to string containing characters to skip.

Return value

Returns the length of the segment of string s1 prefixed by characters from s2.

Additional information

Computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of characters not from the string pointed to by s2.

strtok()

Description

Break string into tokens.

Prototype

```
char *strtok(      char * s1,
             const char * s2);
```

Parameters

Parameter	Description
s1	Pointer to zero-terminated string to parse.
s2	Pointer to zero-terminated set of separators.

Return value

NULL if no further tokens else a pointer to the next token.

Additional information

A sequence of calls to *strtok()* (page 702) breaks the string pointed to by s1 into a sequence of tokens, each of which is delimited by a character from the string pointed to by s2. The first call in the sequence has a non-null first argument; subsequent calls in the sequence have a null first argument. The separator string pointed to by s2 may be different from call to call.

The first call in the sequence searches the string pointed to by s1 for the first character that is not contained in the current separator string pointed to by s2. If no such character is found, then there are no tokens in the string pointed to by s1 and *strtok()* (page 702) returns a null pointer. If such a character is found, it is the start of the first token.

strtok() (page 702) then searches from there for a character that is contained in the current separator string. If no such character is found, the current token extends to the end of the string pointed to by s1, and subsequent searches for a token will return a null pointer. If such a character is found, it is overwritten by a null character, which terminates the current token. *strtok()* (page 702) saves a pointer to the following character, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

Notes

strtok() (page 702) maintains static state and is therefore not reentrant and not thread safe. See *strtok_r()* (page **??**) for a thread-safe and reentrant variant.

See also

strsep() (page 703), *strtok_r()* (page **??**).

strtok_r()

Description

Break string into tokens, reentrant (POSIX.1).

Prototype

```
char *strtok_r(      char  * s1,
               const char  * s2,
                     char ** lasts);
```

Parameters

Parameter	Description
s1	Pointer to zero-terminated string to parse.
s2	Pointer to zero-terminated set of separators.
lasts	Pointer to pointer to char that maintains parse state.

Return value

NULL if no further tokens else a pointer to the next token.

Additional information

strtok_r() (page **??**) is a reentrant version of the function *strtok()* (page 702) where the state is maintained in the object of type char * pointed to by s3.

Notes

Conforms to POSIX.1-2008 and is commonly found in Linux and BSD C libraries.

See also

strtok() (page 702)

strsep()

Description

Break string into tokens (BSD).

Prototype

```
char *strsep(      char ** stringp,
             const char  * delim);
```

Parameters

Parameter	Description
stringp	Pointer to pointer to zero-terminated string.
delim	Pointer to delimiter set string.

Return value

See below.

Additional information

Locates, in the string referenced by *stringp, the first occurrence of any character in the string delim (or the terminating null character) and replaces it with a null character. The location of the next character after the delimiter character (or NULL, if the end of the string was reached) is stored in *stringp. The original value of *stringp is returned.

An empty field (that is, a character in the string delim occurs as the first character of *stringp) can be detected by comparing the location referenced by the returned pointer to the null wide character.

If *stringp is initially null, *strsep()* (page 703) returns null.

Notes

Commonly found in Linux and BSD C libraries.

Miscellaneous functions

Function	Description
strerror() (page 704)	Decode error code.

strerror()

Description

Decode error code.

Prototype

```
char *strerror(int num);
```

Parameters

Parameter	Description
num	Error number.

Return value

Returns a pointer to the message string. The program must not modify the returned message string. The message may be overwritten by a subsequent call to *strerror()* (page 704).

Additional information

Maps the number in num to a message string. Typically, the values for num come from errno, but *strerror()* (page 704) can map any value of type int to a message.

5.6.21 <time.h>

Operations

Function	Description
mktime() (page 704)	Convert a struct tm to time_t.
difftime() (page 705)	Calculate difference between two times.

mktime()

Description

Convert a struct tm to time_t.

Prototype

```
time_t mktime(tm * tp);
```

Parameters

Parameter	Description
tp	Pointer to time object.

Return value

Number of seconds since UTC 1 January 1970 of the validated object.

Additional information

Validates (and updates) the object pointed to by tp to ensure that the tm_sec, tm_min, tm_hour, and tm_mon fields are within the supported integer ranges and the tm_mday, tm_mon and tm_year fields are consistent. The validated object is converted to the number of seconds since UTC 1 January 1970 and returned.

difftime()

Description

Calculate difference between two times.

Prototype

```
double difftime(time_t time2,
                time_t time1);
```

Parameters

Parameter	Description
time2	End time.
time1	Start time.

Return value

returns time2-time1 as a double precision number.

Conversion functions

Function	Description
ctime() (page 705)	Convert time_t to a string.
ctime_r() (page **??**)	Convert time_t to a string, reentrant.
asctime() (page 706)	Convert time_t to a string.
asctime_r() (page **??**)	Convert time_t to a string, reentrant.
gmtime() (page 707)	Convert time_t to struct tm.
gmtime_r() (page **??**)	Convert time_t to struct tm, reentrant.
localtime() (page 708)	Convert time to local time.
localtime_r() (page **??**)	Convert time to local time, reentrant.
strftime() (page 709)	Convert time to a string.
strftime_l() (page **??**)	Convert time to a string.

ctime()

Description

Convert time_t to a string.

Prototype

```
char *ctime(const time_t * tp);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.

Return value

Pointer to zero-terminated converted string.

Additional information

Converts the time pointed to by tp to a null-terminated string.

Notes

The returned string is held in a static buffer: this function is not thread safe.

ctime_r()

Description

Convert time_t to a string, reentrant.

Prototype

```
char *ctime_r(const time_t * tp,
              char   * buf);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.
buf	Pointer to array of characters that receives the zero-terminated string; the array must be at least 26 characters in length.

Return value

Returns the value of buf.

Additional information

Converts the time pointed to by tp to a null-terminated string.

Notes

The returned string is held in a static buffer: this function is not thread safe.

asctime()

Description

Convert time_t to a string.

Prototype

```
char *asctime(const tm * tp);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.

Return value

Pointer to zero-terminated converted string.

Additional information

Converts the time pointed to by tp to a null-terminated string of the Sun Sep 16 01:03:52 1973. The returned string is held in a static buffer.

Notes

The returned string is held in a static buffer: this function is not thread safe.

asctime_r()

Description

Convert time_t to a string, reentrant.

Prototype

```
char *asctime_r(const tm   * tp,
                char * buf);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.
buf	Pointer to array of characters that receives the zero-terminated string; the array must be at least 26 characters in length.

Return value

Returns the value of buf.

Additional information

Converts the time pointed to by tp to a null-terminated string of the Sun Sep 16 01:03:52 1973. The converted string is written into the array pointed to by buf.

gmtime()

Description

Convert time_t to struct tm.

Prototype

```
gmtime(const time_t * tp);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.

Return value

Pointer to converted time.

Additional information

Converts the time pointed to by tp to a struct tm.

Notes

The returned pointer points to a static buffer: this function is not thread safe.

gmtime_r()

Description

Convert time_t to struct tm, reentrant.

Prototype

```
gmtime_r(const time_t * tp,
                        tm *tm);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.
tm	Pointer to object that receives the converted time.

Return value

Returns tm.

Additional information

Converts the time pointed to by tp to a struct tm.

localtime()

Description

Convert time to local time.

Prototype

```
localtime(const time_t * tp);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.

Return value

Pointer to a statically-allocated object holding the local time.

Additional information

Converts the time pointed to by tp to local time format.

Notes

The returned pointer points to a static object: this function is not thread safe.

localtime_r()

Description

Convert time to local time, reentrant.

Prototype

```
localtime_r(const time_t * tp,
                           tm *tm);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.
tm	Pointer to object that receives the converted local time.

Return value

Returns tm.

Additional information

Converts the time pointed to by tp to local time format and writes it to the object pointed to by tm.

strftime()

Description

Convert time to a string.

Prototype

```
size_t strftime(      char    * s,
                      size_t    smax,
                const char    * fmt,
                const tm      * tp);
```

Parameters

Parameter	Description
s	Pointer to object that receives the converted string.
smax	Maximum number of characters written to the array pointed to by s.
fmt	Pointer to zero-terminated format control string.
tp	Pointer to time to convert.

Return value

Returns the name of the current locale.

Additional information

Formats the time pointed to by tp to a null-terminated string of maximum size smax-1 into the pointed to by *s based on the fmt format string. The format string consists of conversion specifications and ordinary characters. Conversion specifications start with a "%" character followed by an optional "#" character.

The following conversion specifications are supported:

Specification	Description
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for locale
%#c	Date and time formatted as "%A, %B %#d, %Y, %H:%M:%S" (Microsoft extension)
%C	Century number
%d	Day of month as a decimal number [01,31]
%#d	Day of month without leading zero [1,31]
%D	Date in the form %m/%d/%y (POSIX.1-2008 extension)
%e	Day of month [1,31], single digit preceded by space
%F	Date in the format %Y-%m-%d
%h	Abbreviated month name as %b
%H	Hour in 24-hour format [00,23]
%#H	Hour in 24-hour format without leading zeros [0,23]
%I	Hour in 12-hour format [01,12]
%#I	Hour in 12-hour format without leading zeros [1,12]
%j	Day of year as a decimal number [001,366]
%#j	Day of year as a decimal number without leading zeros [1,366]
%k	Hour in 24-hour clock format [0,23] (POSIX.1-2008 extension)
%l	Hour in 12-hour clock format [0,12] (POSIX.1-2008 extension)
%m	Month as a decimal number [01,12]
%#m	Month as a decimal number without leading zeros [1,12]
%M	Minute as a decimal number [00,59]
%#M	Minute as a decimal number without leading zeros [0,59]
%n	Insert newline character (POSIX.1-2008 extension)
%p	Locale ' s a.m or p.m indicator for 12-hour clock
%r	Time as %I:%M:%s %p (POSIX.1-2008 extension)
%R	Time as %H:%M (POSIX.1-2008 extension)
%S	Second as a decimal number [00,59]
%t	Insert tab character (POSIX.1-2008 extension)
%T	Time as %H:%M:%S
%#S	Second as a decimal number without leading zeros [0,59]
%U	Week of year as a decimal number [00,53], Sunday is first day of the week
%#U	Week of year as a decimal number without leading zeros [0,53], Sunday is first day of the week
%w	Weekday as a decimal number [0,6], Sunday is 0
%W	Week number as a decimal number [00,53], Monday is first day of the week
%#W	Week number as a decimal number without leading zeros [0,53], Monday is first day of the week
%x	Locale ' s date representation
%#x	Locale ' s long date representation
%X	Locale ' s time representation
%y	Year without century, as a decimal number [00,99]
%#y	Year without century, as a decimal number without leading zeros [0,99]
%Y	Year with century, as decimal number
%z,%Z	Timezone name or abbreviation
%%	%

strftime_l()

Description

Convert time to a string.

Prototype

```
size_t strftime_l(      char    * s,
                        size_t    smax,
                  const char    * fmt,
                  const tm      * tp,
                                  locale_t loc);
```

Parameters

Parameter	Description
s	Pointer to object that receives the converted string.
smax	Maximum number of characters written to the array pointed to by s.
fmt	Pointer to zero-terminated format control string.
tp	Pointer to time to convert.
loc	Locale to use for conversion.

Return value

Returns the name of the current locale.

Additional information

Formats the time pointed to by tp to a null-terminated string of maximum size smax-1 into the pointed to by *s based on the fmt format string and using the locale loc.

The format string consists of conversion specifications and ordinary characters. Conversion specifications start with a "%" character followed by an optional "#" character.

See *strftime()* (page 709) for a description of the format conversion specifications.

5.6.22 <unistd.h>

The `<unistd.h>` header is provided for POSIX compatibility. It includes `<stdlib.h>` and additionally declares the following function.

Miscellaneous functions

Function	Description
isatty	Test for a terminal.

isatty

Description

Test for a terminal.

Prototype

```
int isatty(int fd);
```

Parameters

Parameter	Description
fd	File descriptor to be tested.

Return value

nonzero	fd refers to a terminal.
zero	fd does not refer to a terminal.

Additional information

The runtime library only declares `isatty()`, it does not implement it. If the application uses `isatty()`, an implementation must be provided by the application itself.

5.6.23 <wchar.h>

Copying functions

Function	Description
wmemset() (page 713)	Set memory to wide character.
wmemcpy() (page 713)	Copy memory.
wmemccpy() (page 714)	Copy memory, specify terminator (POSIX.1).
wmempcpy() (page 714)	Copy memory (GNU).
wmemmove() (page 715)	Copy memory, tolerate overlaps.
wcscpy() (page 715)	Copy string.
wcsncpy() (page 716)	Copy string, limit length.
wcslcpy() (page 716)	Copy string, limit length, always zero terminate (BSD).
wcscat() (page 717)	Concatenate strings.
wcsncat() (page 717)	Concatenate strings, limit length.
wcslcat() (page 718)	Concatenate strings, limit length, always zero terminate (BSD).
wcsdup() (page 719)	Duplicate string (POSIX.1).
wcsndup() (page 719)	Duplicate string, limit length (GNU).

wmemset()

Description

Set memory to wide character.

Prototype

```
wchar_t *wmemset(wchar_t * s,
                 wchar_t   c,
                 size_t    n);
```

Parameters

Parameter	Description
s	Pointer to destination object.
c	Wide character to copy.
n	Length of destination object in wide characters.

Return value

Returns s.

Additional information

Copies the value of c into each of the first n wide characters of the object pointed to by s.

wmemcpy()

Description

Copy memory.

Prototype

```
wchar_t *wmemcpy(      wchar_t * s1,
                 const wchar_t * s2,
                       size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of wide characters to copy.

Return value

Returns the value of s1.

Additional information

Copies n wide characters from the object pointed to by s2 into the object pointed to by s1. The behavior of *wmemcpy()* (page 713) is undefined if copying takes place between objects that overlap.

wmemccpy()

Description

Copy memory, specify terminator (POSIX.1).

Prototype

```
wchar_t *wmemccpy(      wchar_t * s1,
                  const wchar_t * s2,
                        wchar_t   c,
                        size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
c	Character that terminates copy.
n	Maximum number of characters to copy.

Return value

Returns a pointer to the wide character immediately following c in s1, or NULL if c was not found in the first n wide characters of s2.

Additional information

Copies at most n wide characters from the object pointed to by s2 into the object pointed to by s1. The copying stops as soon as n wide characters are copied or the wide character c is copied into the destination object pointed to by s1.

The behavior of *wmemccpy()* (page 714) is undefined if copying takes place between objects that overlap.

Notes

Conforms to POSIX.1-2008.

wmempcpy()

Description

Copy memory (GNU).

Prototype

```
wchar_t *wmempcpy(      wchar_t * s1,
                  const wchar_t * s2,
                        size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of wide characters to copy.

Return value

Returns a pointer to the wide character immediately following the final wide character written into s1.

Additional information

Copies n wide characters from the object pointed to by s2 into the object pointed to by s1. The behavior of *wmempcpy()* (page 714) is undefined if copying takes place between objects that overlap.

Notes

This is an extension found in GNU libc.

wmemmove()

Description

Copy memory, tolerate overlaps.

Prototype

```
wchar_t *wmemmove(      wchar_t * s1,
                  const wchar_t * s2,
                        size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of wide characters to copy.

Return value

Returns the value of s1.

Additional information

Copies n wide characters from the object pointed to by s2 into the object pointed to by s1 ensuring that if s1 and s2 overlap, the copy works correctly. Copying takes place as if the n wide characters from the object pointed to by s2 are first copied into a temporary array of n wide characters that does not overlap the objects pointed to by s1 and s2, and then the n wide characters from the temporary array are copied into the object pointed to by s1.

wcscpy()

Description

Copy string.

Prototype

```
wchar_t *wcscpy(      wchar_t * s1,
                const wchar_t * s2);
```

Parameters

Parameter	Description
s1	Pointer to wide string to copy to.
s2	Pointer to wide string to copy.

Return value

Returns the value of s1.

Additional information

Copies the wide string pointed to by s2 (including the terminating null wide character) into the array pointed to by s1. The behavior of *wcscpy()* (page 715) is undefined if copying takes place between objects that overlap.

wcsncpy()

Description

Copy string, limit length.

Prototype

```
wchar_t *wcsncpy(      wchar_t * s1,
                 const wchar_t * s2,
                       size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to wide string to copy to.
s2	Pointer to wide string to copy.
n	Maximum number of wide characters to copy.

Return value

Returns the value of s1.

Additional information

Copies not more than n wide characters from the array pointed to by s2 to the array pointed to by s1. Wide characters that follow a null wide character in s2 are not copied. The behavior of *wcsncpy()* (page 716) is undefined if copying takes place between objects that overlap. If the array pointed to by s2 is a wide string that is shorter than n wide characters, null wide characters are appended to the copy in the array pointed to by s1, until n characters in all have been written.

Notes

No wide null character is implicitly appended to the end of s1, so s1 will only be terminated by a wide null character if the length of the wide string pointed to by s2 is less than n.

wcslcpy()

Description

Copy string, limit length, always zero terminate (BSD).

Prototype

```
size_t wcslcpy(      wchar_t * s1,
               const wchar_t * s2,
                     size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to wide string to copy to.
s2	Pointer to wide string to copy.
n	Maximum number of wide characters, including terminating null, in s1.

Return value

Returns the number of wide characters it tried to copy, which is the length of the wide string s2 or n, whichever is smaller.

Additional information

Copies up to n-1 wide characters from the wide string pointed to by s2 into the array pointed to by s1 and always terminates the result with a null character.

The behavior of *strlcpy()* (page 687) is undefined if copying takes place between objects that overlap.

Notes

Commonly found in BSD libraries and contrasts with *wcsncpy()* (page 716) in that the resulting string is always terminated with a null wide character.

wcscat()

Description

Concatenate strings.

Prototype

```
wchar_t *wcscat(      wchar_t * s1,
                const wchar_t * s2);
```

Parameters

Parameter	Description
s1	Zero-terminated wide string to append to.
s2	Zero-terminated wide string to append.

Return value

Returns the value of s1.

Additional information

Appends a copy of the wide string pointed to by s2 (including the terminating null wide character) to the end of the wide string pointed to by s1. The initial character of s2 overwrites the null wide character at the end of s1. The behavior of *wcscat()* (page 717) is undefined if copying takes place between objects that overlap.

wcsncat()

Description

Concatenate strings, limit length.

Prototype

```
wchar_t *wcsncat(      wchar_t * s1,
                 const wchar_t * s2,
                       size_t    n);
```

Parameters

Parameter	Description
s1	Wide string to append to.
s2	Wide string to append.
n	Maximum number of wide characters in s1.

Return value

Returns the value of s1.

Additional information

Appends not more than n wide characters from the array pointed to by s2 to the end of the wide string pointed to by s1. A null wide character in s1 and wide characters that follow it are not appended. The initial wide character of s2 overwrites the null wide character at the end of s1. A terminating wide null character is always appended to the result. The behavior of *wcsncat()* (page 717) is undefined if copying takes place between objects that overlap.

wcslcat()

Description

Concatenate strings, limit length, always zero terminate (BSD).

Prototype

```
size_t wcslcat(      char    * s1,
               const char    * s2,
                     size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to wide string to append to.
s2	Pointer to wide string to append.
n	Maximum number of characters, including terminating wide null, in s1.

Return value

Returns the number of wide characters it tried to copy, which is the sum of the lengths of the wide strings s1 and s2 or n, whichever is smaller.

Additional information

Appends no more than n-strlen(s1}-1 wide characters pointed to by s2 into the array pointed to by s1 and always terminates the result with a wide null character if n is greater than zero. Both the wide strings s1 and s2 must be terminated with a wide null character on entry to *wcslcat()* (page 718) and a character position for the terminating wide null should be included in n.

The behavior of *wcslcat()* (page 718) is undefined if copying takes place between objects that overlap.

Notes

Commonly found in BSD libraries.

wcsdup()

Description

Duplicate string (POSIX.1).

Prototype

```
wchar_t *wcsdup(const wchar_t * s);
```

Parameters

Parameter	Description
s	Pointer to wide string to duplicate.

Return value

Returns a pointer to the new wide string or a null pointer if the new wide string cannot be created. The returned pointer can be passed to *free()* (page 658).

Additional information

Duplicates the wide string pointed to by s by using *malloc()* (page 657) to allocate memory for a copy of s and then copies s, including the terminating null, to that memory

Notes

Conforms to POSIX.1-2008 and SC22 TR 24731-2.

wcsndup()

Description

Duplicate string, limit length (GNU).

Prototype

```
wchar_t *wcsndup(const wchar_t * s,
                 size_t     n);
```

Parameters

Parameter	Description
s	Pointer to wide string to duplicate.
n	Maximum number of wide characters to duplicate.

Return value

Returns a pointer to the new wide string or a null pointer if the new wide string cannot be created. The returned pointer can be passed to *free()* (page 658).

Additional information

Duplicates at most n wide characters from the the string pointed to by s by using *malloc()* (page 657) to allocate memory for a copy of s.

If the length of string pointed to by s is greater than n wide characters, only n wide characters will be duplicated. If n is greater than the length of the wide string pointed to by s, all characters in the string are copied into the allocated array including the terminating null character.

Notes

This is a GNU extension.

Comparison functions

Function	Description
wmemcmp() (page 720)	Compare memory.
wcsncmp() (page 720)	Compare strings, limit length.
wcscasecmp() (page 721)	Compare strings, ignore case (POSIX.1).
wcsncasecmp() (page 722)	Compare strings, ignore case, limit length (POSIX.1).

wmemcmp()

Description

Compare memory.

Prototype

```
int wmemcmp(const wchar_t * s1,
            const wchar_t * s2,
                  size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to object #1.
s2	Pointer to object #2.
n	Number of wide characters to compare.

Return value

< 0	s1 is less than s2.
= 0	s1 is equal to s2.
> 0	s1 is greater than to s2.

Additional information

Compares the first n wide characters of the object pointed to by s1 to the first n wide characters of the object pointed to by s2. *wmemcmp()* (page 720) returns an integer greater than, equal to, or less than zero as the object pointed to by s1 is greater than, equal to, or less than the object pointed to by s2.

wcsncmp()

Description

Compare strings, limit length.

Prototype

```
int wcsncmp(const wchar_t * s1,
            const wchar_t * s2,
                  size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to wide string #1.
s2	Pointer to wide string #2.
n	Maximum number of wide characters to compare.

Return value

Returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2.

Additional information

Compares not more than n wide characters from the array pointed to by s1 to the array pointed to by s2. Wide characters that follow a null wide character are not compared.

wcscasecmp()

Description

Compare strings, ignore case (POSIX.1).

Prototype

```
int wcscasecmp(const char * s1,
               const char * s2);
```

Parameters

Parameter	Description
s1	Pointer to wide string #1.
s2	Pointer to wide string #2.

Return value

< 0	s1 is less than s2.
= 0	s1 is equal to s2.
> 0	s1 is greater than to s2.

Additional information

Compares the wide string pointed to by s1 to the wide string pointed to by s2 ignoring differences in case.

wcscasecmp() (page 721) returns an integer greater than, equal to, or less than zero if the wide string pointed to by s1 is greater than, equal to, or less than the wide string pointed to by s2.

Notes

Conforms to POSIX.1-2017.

wcsncasecmp()

Description

Compare strings, ignore case, limit length (POSIX.1).

Prototype

```
int wcsncasecmp(const char   * s1,
                const char   * s2,
                      size_t   n);
```

Parameters

Parameter	Description
s1	Pointer to wide string #1.
s2	Pointer to wide string #2.
n	Maximum number of wide characters to compare.

Return value

< 0	s1 is less than s2.
= 0	s1 is equal to s2.
> 0	s1 is greater than to s2.

Additional information

Compares not more than n wide characters from the array pointed to by s1 to the array pointed to by s2 ignoring differences in case. Characters that follow a wide null character are not compared.

strncasecmp() (page 693) returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2.

Notes

Conforms to POSIX.1-2017.

Search functions

Function	Description
wmemchr() (page 723)	Find character in memory, forward.
wcschr() (page 723)	Find character within string, forward.
wcsnchr() (page 724)	Find character within string, forward, limit length.
wcsrchr() (page 724)	Find character within string, reverse.
wcslen() (page 725)	Calculate length of string.
wcsnlen() (page 725)	Calculate length of string, limit length (POSIX.1).
wcsstr() (page 725)	Find string within string, forward.
wcsnstr() (page 726)	Find string within string, forward, limit length (BSD).
wcspbrk() (page 727)	Find first occurrence of characters within string.
wcsspn() (page 727)	Compute size of string prefixed by a set of characters.
wcscspn() (page 728)	Compute size of string not prefixed by a set of characters.
wcstok() (page 728)	Break string into tokens.
wcssep() (page 729)	Break string into tokens (BSD).

wmemchr()

Description

Find character in memory, forward.

Prototype

```
wchar_t *wmemchr(const wchar_t * s,
                 wchar_t   c,
                 size_t    n);
```

Parameters

Parameter	Description
s	Pointer to object to search.
c	Wide character to search for.
n	Number of wide characters in object to search.

Return value

= NULL	c does not occur in the object.
≠ NULL	Pointer to the located wide character.

Additional information

Locates the first occurrence of c in the initial n wide characters of the object pointed to by s. Unlike *wcschr()* (page 723), *wmemchr()* (page 723) does not terminate a search when a null wide character is found in the object pointed to by s.

wcschr()

Description

Find character within string, forward.

Prototype

```
wchar_t *wcschr(const wchar_t * s,
                wchar_t   c);
```

Parameters

Parameter	Description
s	Wide string to search.
c	Wide character to search for.

Return value

Returns a pointer to the located wide character, or a null pointer if c does not occur in the wide string.

Additional information

Locates the first occurrence of c in the wide string pointed to by s. The terminating wide null character is considered to be part of the string.

wcsnchr()

Description

Find character within string, forward, limit length.

Prototype

```
wchar_t *wcsnchr(const wchar_t * s,
                 size_t    n,
                 wchar_t   c);
```

Parameters

Parameter	Description
s	Pointer to wide string to search.
n	Number of wide characters to search.
c	Wide character to search for.

Return value

Returns a pointer to the located wide character, or a null pointer if c does not occur in the string.

Additional information

Searches not more than n wide characters to locate the first occurrence of c in the wide string pointed to by s. The terminating wide null character is considered to be part of the wide string.

wcsrchr()

Description

Find character within string, reverse.

Prototype

```
wchar_t *wcsrchr(const wchar_t * s,
                 wchar_t   c);
```

Parameters

Parameter	Description
s	Pointer to wide string to search.
c	Wide character to search for.

Return value

Returns a pointer to the located wide character, or a null pointer if c does not occur in the string.

Additional information

Locates the last occurrence of c in the wide string pointed to by s. The terminating wide null character is considered to be part of the string.

wcslen()

Description

Calculate length of string.

Prototype

```
size_t wcslen(const wchar_t * s);
```

Parameters

Parameter	Description
s	Pointer to zero-terminated wide string.

Return value

Returns the length of the wide string pointed to by s, that is the number of wide characters that precede the terminating wide null character.

wcsnlen()

Description

Calculate length of string, limit length (POSIX.1).

Prototype

```
size_t wcsnlen(const wchar_t * s,
               size_t    n);
```

Parameters

Parameter	Description
s	Pointer to wide string.
n	Maximum number of wide characters to examine.

Return value

Returns the length of the wide string pointed to by s, up to a maximum of n wide characters. *wcsnlen()* (page 725) only examines the first n wide characters of the string s.

Notes

Conforms to POSIX.1-2008.

wcsstr()

Description

Find string within string, forward.

Prototype

```
wchar_t *wcsstr(const wchar_t * s1,
                const wchar_t * s2);
```

Parameters

Parameter	Description
s1	Pointer to wide string to search.
s2	Pointer to wide string to search for.

Return value

Returns a pointer to the located wide string, or a null pointer if the wide string is not found. If s2 points to a wide string with zero length, *wcsstr()* (page 725) returns s1.

Additional information

Locates the first occurrence in the wide string pointed to by s1 of the sequence of wide characters (excluding the terminating null wide character) in the wide string pointed to by s2.

wcsnstr()

Description

Find string within string, forward, limit length (BSD).

Prototype

```
wchar_t *wcsnstr(const wchar_t * s1,
                 const wchar_t * s2,
                       size_t    n);
```

Parameters

Parameter	Description
s1	Pointer to wide string to search.
s2	Pointer to wide string to search for.
n	Maximum number of characters to search for.

Return value

Returns a pointer to the located wide string, or a null pointer if the wide string is not found. If s2 points to a wide string with zero length, *wcsnstr()* (page 726) returns s1.

Additional information

Searches at most n wide characters to locate the first occurrence in the wide string pointed to by s1 of the sequence of wide characters (excluding the terminating wide null character) in the string pointed to by s2.

Notes

Commonly found in Linux and BSD C libraries.

wcspbrk()

Description

Find first occurrence of characters within string.

Prototype

```
wchar_t *wcspbrk(const wchar_t * s1,
                 const wchar_t * s2);
```

Parameters

Parameter	Description
s1	Pointer to wide string to search.
s2	Pointer to wide string to search for.

Return value

Returns a pointer to the first wide character, or a null pointer if no wide character from s2 occurs in s1.

Additional information

Locates the first occurrence in the wide string pointed to by s1 of any wide character from the string pointed to by s2.

wcsspn()

Description

Compute size of string prefixed by a set of characters.

Prototype

```
size_t wcsspn(const wchar_t * s1,
              const wchar_t * s2);
```

Parameters

Parameter	Description
s1	Pointer to zero-terminated wide string to search.
s2	Pointer to zero-terminated acceptable-set wide string.

Return value

Returns the length of the wide string pointed to by s1 which consists entirely of wide characters from the wide string pointed to by s2

Additional information

Computes the length of the maximum initial segment of the wide string pointed to by s1 which consists entirely of wide characters from the string pointed to by s2.

wcscspn()

Description

Compute size of string not prefixed by a set of characters.

Prototype

```
size_t wcscspn(const wchar_t * s1,
               const wchar_t * s2);
```

Parameters

Parameter	Description
s1	Pointer to wide string to search.
s2	Pointer to wide string containing characters to skip.

Return value

Returns the length of the segment of wide string s1 prefixed by wide characters from s2.

Additional information

Computes the length of the maximum initial segment of the wide string pointed to by s1 which consists entirely of wide characters not from the wide string pointed to by s2.

wcstok()

Description

Break string into tokens.

Prototype

```
wchar_t *wcstok(      wchar_t  * s1,
                const wchar_t  * s2,
                      wchar_t ** ptr);
```

Parameters

Parameter	Description
s1	Pointer to zero-terminated wide string to parse.
s2	Pointer to zero-terminated set of separators.
ptr	Pointer to object that maintains parse state.

Return value

NULL if no further tokens else a pointer to the next token.

Additional information

A sequence of calls to *wcstok()* (page 728) breaks the wide string pointed to by s1 into a sequence of tokens, each of which is delimited by a wide character from the wide string pointed to by s2. The first call in the sequence has a non-null first argument; subsequent calls in the sequence have a null first argument. The separator wide string pointed to by s2 may be different from call to call.

The first call in the sequence searches the wide string pointed to by s1 for the wide first character that is not contained in the current separator wide string pointed to by s2. If no such wide character is found, then there are no tokens in the string pointed to by s1 and *wcstok()* (page 728) returns a null pointer. If such a wide character is found, it is the start of the first token.

wcstok() (page 728) then searches from there for a wide character that is contained in the current separator wide string. If no such wide character is found, the current token extends to the end of the wide string pointed to by s1, and subsequent searches for a token will return a null pointer. If such a wide character is found, it is overwritten by a null wide character, which terminates the current token. *wcstok()* (page 728) saves a pointer to the following wide character, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

See also

wcssep() (page 729).

wcssep()

Description

Break string into tokens (BSD).

Prototype

```
wchar_t *wcssep(      wchar_t ** stringp,
                const wchar_t  * delim);
```

Parameters

Parameter	Description
stringp	Pointer to pointer to zero-terminated wide string.
delim	Pointer to delimiter set wide string.

Return value

See below.

Additional information

Locates, in the wide string referenced by *stringp, the first occurrence of any wide character in the wide string delim (or the terminating null character) and replaces it with a null wide character. The location of the next wide character after the delimiter wide character (or NULL, if the end of the wide string was reached) is stored in *stringp. The original value of *stringp is returned.

An empty field (that is, a wide character in the string delim occurs as the first character of *stringp) can be detected by comparing the location referenced by the returned pointer to the null wide character.

If *stringp is initially null, *wcssep()* (page 729) returns null.

Notes

Commonly found in Linux and BSD C libraries.

Multi-byte/wide string conversion functions

Function	Description
mbsinit() (page 730)	Query initial conversion state.
mbrlen() (page 730)	Count number of bytes in multi-byte character, restartable.
mbrlen_l() (page **??**)	Count number of bytes in multi-byte character, restartable, per locale (POSIX.1).
mbrtowc() (page 732)	Convert multi-byte character to wide character, restartable.
mbrtowc_l() (page **??**)	Convert multi-byte character to wide character, restartable, per locale (POSIX.1).
wctob() (page 733)	Convert wide character to single-byte character.
wctob_l() (page **??**)	Convert wide character to single-byte character, per locale (POSIX.1).
wcrtomb() (page 734)	Convert wide character to multi-byte character, restartable.
wcrtomb_l() (page **??**)	Convert wide character to multi-byte character, restartable, per locale (POSIX.1).
wcsrtombs() (page 735)	Convert wide string to multi-byte string, restartable.
wcsrtombs_l() (page **??**)	Convert wide string to multi-byte string, restartable (POSIX.1).

mbsinit()

Description

Query initial conversion state.

Prototype

```
int mbsinit(const mbstate_t * ps);
```

Parameters

Parameter	Description
ps	Pointer to conversion state.

Return value

Returns nonzero (true) if ps is a null pointer or if the pointed-to object describes an initial conversion state; otherwise, returns zero.

mbrlen()

Description

Count number of bytes in multi-byte character, restartable.

Prototype

```
size_t mbrlen(const char      * s,
              size_t          n,
              mbstate_t * ps);
```

Parameters

Parameter	Description
s	Pointer to multi-byte character.
n	Maximum number of bytes to examine.
ps	Pointer to multi-byte conversion state.

Return value

Number of bytes in multi-byte character.

Additional information

Determines the number of bytes contained in the multi-byte character pointed to by s in the current locale.

Except that except that the expression designated by ps is evaluated only once, this function is equivalent to the call:

```
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

where internal is the mbstate_t object for the *mbrlen()* (page 730) function.

See also

mbrlen_l() (page **??**), *mbrtowc()* (page 732)

mbrlen_l()

Description

Count number of bytes in multi-byte character, restartable, per locale (POSIX.1).

Prototype

```
size_t mbrlen_l(const char       * s,
                size_t           n,
                mbstate_t * ps,
                locale_t loc);
```

Parameters

Parameter	Description
s	Pointer to multi-byte character.
n	Maximum number of bytes to examine.
ps	Pointer to multi-byte conversion state.
loc	Locale used for conversion.

Return value

Number of bytes in multi-byte character.

Additional information

Determines the number of bytes contained in the multi-byte character pointed to by s in the locale loc.

Except that except that the expression designated by ps is evaluated only once, this function is equivalent to the call:

```
mbrtowc_l(NULL, s, n, ps != NULL ? ps : &internal, loc);
```

where internal is the mbstate_t object for the *mbrlen()* (page 730) function,

Notes

Conforms to POSIX.1-2008.

See also

mbrlen_l() (page **??**), *mbrtowc()* (page 732)

mbrtowc()

Description

Convert multi-byte character to wide character, restartable.

Prototype

```
size_t mbrtowc(      wchar_t    * pwc,
               const char       * s,
                     size_t       n,
                     mbstate_t  * ps);
```

Parameters

Parameter	Description
pwc	Pointer to object that receives the wide character.
s	Pointer to multi-byte character string.
n	Maximum number of bytes that will be examined.
ps	Pointer to multi-byte conversion state.

Return value

If s is a null pointer, *mbrtowc()* (page 732) is equivalent to mbrtowc(NULL, “” , 1, ps), ignoring pwc and n.

If s is not null and the object that s points to is a wide null character, *mbrtowc()* (page 732) returns 0.

If s is not null and the object that s points to forms a valid multi-byte character in the current locale with a most n bytes, *mbrtowc()* (page 732) returns the length in bytes of the multi-byte character and stores that wide character to the object pointed to by pwc (if pwc is not null).

If the object that s points to forms an incomplete, but possibly valid, multi-byte character, *mbrtowc()* (page 732) returns -2.

If the object that s points to does not form a partial multi-byte character, *mbrtowc()* (page 732) returns -1.

Additional information

Converts a single multi-byte character to a wide character in the current locale.

See also

mbtowc() (page 676), *mbrtowc_l()* (page **??**)

mbrtowc_l()

Description

Convert multi-byte character to wide character, restartable, per locale (POSIX.1).

Prototype

```
size_t mbrtowc_l(      wchar_t    * pwc,
                 const char       * s,
                       size_t       n,
                       mbstate_t  * ps,
                                    locale_t loc);
```

Parameters

Parameter	Description
pwc	Pointer to object that receives the wide character.
s	Pointer to multi-byte character string.
n	Maximum number of bytes that will be examined.
ps	Pointer to multi-byte conversion state.
loc	Locale used for conversion.

Return value

If s is a null pointer, *mbrtowc()* (page 732) is equivalent to mbrtowc(NULL, "" , 1, ps), ignoring pwc and n.

If s is not null and the object that s points to is a wide null character, *mbrtowc()* (page 732) returns 0.

If s is not null and the object that s points to forms a valid multi-byte character in the locale loc with a most n bytes, *mbrtowc()* (page 732) returns the length in bytes of the multi-byte character and stores that wide character to the object pointed to by pwc (if pwc is not null).

If the object that s points to forms an incomplete, but possibly valid, multi-byte character, *mbrtowc()* (page 732) returns -2.

If the object that s points to does not form a partial multi-byte character, *mbrtowc()* (page 732) returns -1.

Additional information

Converts a single multi-byte character to a wide character in the locale loc.

Notes

Conforms to POSIX.1-2008.

See also

mbtowc() (page 676), *mbrtowc_l()* (page **??**)

wctob()

Description

Convert wide character to single-byte character.

Prototype

```
int wctob(wint_t c);
```

Parameters

Parameter	Description
c	Character to convert.

Return value

Returns EOF if c does not correspond to a multi-byte character with length one in the initial shift state in the current locale. Otherwise, it returns the single-byte representation of that character as an unsigned char converted to an int.

Additional information

Determines whether c corresponds to a member of the extended character set whose multi-byte character representation is a single byte in the current locale when in the initial shift state.

wctob_l()

Description

Convert wide character to single-byte character, per locale (POSIX.1).

Prototype

```
int wctob_l(wint_t c,
            locale_t loc);
```

Parameters

Parameter	Description
c	Character to convert.
loc	Locale used for conversion.

Return value

Returns EOF if c does not correspond to a multi-byte character with length one in the initial shift state in the locale loc. Otherwise, it returns the single-byte representation of that character as an unsigned char converted to an int.

Additional information

Determines whether c corresponds to a member of the extended character set whose multi-byte character representation is a single byte in the locale loc when in the initial shift state.

wcrtomb()

Description

Convert wide character to multi-byte character, restartable.

Prototype

```
size_t wcrtomb(char      * s,
               wchar_t wc,
           mbstate_t * ps);
```

Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte character.
wc	Wide character to convert.
ps	Pointer to multi-byte conversion state.

Return value

Returns the number of bytes stored in the array object. When wc is not a valid wide character, an encoding error occurs: *wcrtomb()* (page 734) stores the value of the macro EILSEQ in errno and returns (*size_t* (page **??**))(-1); the conversion state is unspecified.

Additional information

If s is a null pointer, *wcrtomb()* (page 734) is equivalent to the call wcrtomb(buf, 0, ps) where buf is an internal buffer.

If s is not a null pointer, *wcrtomb()* (page 734) determines the number of bytes needed to represent the multi-byte character that corresponds to the wide character given by wc in the locale loc, and stores the multi-byte character

representation in the array whose first element is pointed to by s. At most MB_CUR_MAX bytes are stored. If wc is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

wcrtomb_l()

Description

Convert wide character to multi-byte character, restartable, per locale (POSIX.1).

Prototype

```
size_t wcrtomb_l(char      * s,
                   wchar_t wc,
                 mbstate_t * ps,
                   locale_t loc);
```

Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte character.
wc	Wide character to convert.
ps	Pointer to multi-byte conversion state.
loc	Locale used for conversion.

Return value

Returns the number of bytes stored in the array object. When wc is not a valid wide character, an encoding error occurs: *wcrtomb_l()* (page **??**) stores the value of the macro EILSEQ in errno and returns (*size_t* (page **??**))(-1); the conversion state is unspecified.

Additional information

If s is a null pointer, *wcrtomb()* (page 734) is equivalent to the call wcrtomb(buf, 0, ps) where buf is an internal buffer.

If s is not a null pointer, *wcrtomb()* (page 734) determines the number of bytes needed to represent the multi-byte character that corresponds to the wide character given by wc in the current locale, and stores the multi-byte character representation in the array whose first element is pointed to by s. At most MB_CUR_MAX bytes are stored. If wc is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

wcsrtombs()

Description

Convert wide string to multi-byte string, restartable.

Prototype

```
size_t wcsrtombs(      char       * dst,
                 const wchar_t    ** src,
                       size_t       len,
                       mbstate_t  * ps);
```

Parameters

Parameter	Description
dst	Pointer to array that receives the multi-byte string.
src	Indirect pointer to wide character string being converted.
len	Maximum number of bytes to write into the array pointed to by dst.
ps	Pointer to multi-byte conversion state.

Return value

If conversion stops because a wide character is reached that does not correspond to a valid multi-byte character, an encoding error occurs: *wcsrtombs()* (page 735) stores the value of the macro EILSEQ in errno and returns (*size_t* (page **??**))(-1); the conversion state is unspecified. Otherwise, it returns the number of bytes in the resulting multi-byte character sequence, not including the terminating null character (if any).

Additional information

Converts a sequence of wide characters in the current locale from the array indirectly pointed to by src into a sequence of corresponding multi-byte characters that begins in the conversion state described by the object pointed to by ps. If dst is not a null pointer, the converted characters are then stored into the array pointed to by dst. Conversion continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases: when a wide character is reached that does not correspond to a valid multi-byte character, or (if dst is not a null pointer) when the next multi-byte character would exceed the limit of len total bytes to be stored into the array pointed to by dst. Each conversion takes place as if by a call to *wcrtomb()* (page 734).

If dst is not a null pointer, the pointer object pointed to by src is assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described is the initial conversion state.

wcsrtombs_l()

Description

Convert wide string to multi-byte string, restartable (POSIX.1).

Prototype

```
size_t wcsrtombs_l(      char        * dst,
                   const wchar_t    ** src,
                         size_t        len,
                         mbstate_t   * ps,
                                       locale_t loc);
```

Parameters

Parameter	Description
dst	Pointer to array that receives the multi-byte string.
src	Indirect pointer to wide character string being converted.
len	Maximum number of bytes to write into the array pointed to by dst.
ps	Pointer to multi-byte conversion state.
loc	Locale used for conversion.

Return value

If conversion stops because a wide character is reached that does not correspond to a valid multi-byte character, an encoding error occurs: *wcsrtombs()* (page 735) stores the value of the macro EILSEQ in errno and returns (*size_t*

(page **??**))(-1); the conversion state is unspecified. Otherwise, it returns the number of bytes in the resulting multi-byte character sequence, not including the terminating null character (if any).

Additional information

Converts a sequence of wide characters in the locale loc from the array indirectly pointed to by src into a sequence of corresponding multi-byte characters that begins in the conversion state described by the object pointed to by ps. If dst is not a null pointer, the converted characters are then stored into the array pointed to by dst. Conversion continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases: when a wide character is reached that does not correspond to a valid multi-byte character, or (if dst is not a null pointer) when the next multi-byte character would exceed the limit of len total bytes to be stored into the array pointed to by dst. Each conversion takes place as if by a call to *wcrtomb_l()* (page **??**).

If dst is not a null pointer, the pointer object pointed to by src is assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described is the initial conversion state.

Notes

Conforms to POSIX.1-2008.

5.6.24 <wctype.h>

Classification functions

Function	Description
iswcntrl() (page 738)	Is character a control?
iswcntrl_l() (page **??**)	Is character a control, per locale? (POSIX.1).
iswblank() (page 738)	Is character a blank?
iswblank_l() (page **??**)	Is character a blank, per locale? (POSIX.1).
iswspace() (page 739)	Is character a whitespace character?
iswspace_l() (page **??**)	Is character a whitespace character, per locale? (POSIX.1).
iswpunct() (page 740)	Is character a punctuation mark?
iswpunct_l() (page **??**)	Is character a punctuation mark, per locale? (POSIX.1).
iswdigit() (page 741)	Is character a decimal digit?
iswdigit_l() (page **??**)	Is character a decimal digit, per locale? (POSIX.
iswxdigit() (page 742)	Is character a hexadecimal digit?
iswxdigit_l() (page **??**)	Is character a hexadecimal digit, per locale? (POSIX.1).
iswalpha() (page 742)	Is character alphabetic?
iswalpha_l() (page **??**)	Is character alphabetic, per locale? (POSIX.1).
iswalnum() (page 743)	Is character alphanumeric?
iswalnum_l() (page **??**)	Is character alphanumeric, per locale? (POSIX.1).
iswupper() (page 744)	Is character an uppercase letter?
iswupper_l() (page **??**)	Is character an uppercase letter, per locale? (POSIX.1).
iswlower() (page 745)	Is character a lowercase letter?
iswlower_l() (page **??**)	Is character a lowercase letter, per locale? (POSIX.1).
iswprint() (page 746)	Is character printable?
iswprint_l() (page **??**)	Is character printable, per locale? (POSIX.1).
iswgraph() (page 746)	Is character any printing character?
iswgraph_l() (page **??**)	Is character any printing character, per locale? (POSIX.1).
iswctype() (page 747)	Construct character mapping.
iswctype_l() (page **??**)	Construct character mapping, per locale (POSIX.1).
wctype() (page 748)	Construct character class.

iswcntrl()

Description

Is character a control?

Prototype

```
int iswcntrl(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a control character in the current locale.

iswcntrl_l()

Description

Is character a control, per locale? (POSIX.1).

Prototype

```
int iswcntrl_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a control character in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswblank()

Description

Is character a blank?

Prototype

```
int iswblank(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is either a space character or tab character in the current locale.

iswblank_l()

Description

Is character a blank, per locale? (POSIX.1).

Prototype

```
int iswblank_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is either a space character or the tab character in locale loc.

Notes

Conforms to POSIX.1-2017.

iswspace()

Description

Is character a whitespace character?

Prototype

```
int iswspace(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a standard white-space character in the current locale. The standard white-space characters are space, form feed, new-line, carriage return, horizontal tab, and vertical tab.

iswspace_l()

Description

Is character a whitespace character, per locale? (POSIX.1).

Prototype

```
int iswspace_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a standard white-space character in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswpunct()

Description

Is character a punctuation mark?

Prototype

```
int iswpunct(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) for every printing character for which neither *isspace()* (page 520) nor *isalnum()* (page 524) is true in the current locale.

iswpunct_l()

Description

Is character a punctuation mark, per locale? (POSIX.1).

Prototype

```
int iswpunct_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) for every printing character for which neither *isspace()* (page 520) nor *isalnum()* (page 524) is true in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswdigit()

Description

Is character a decimal digit?

Prototype

```
int iswdigit(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a digit in the current locale.

iswdigit_l()

Description

Is character a decimal digit, per locale? (POSIX.1)

Prototype

```
int iswdigit_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a digit in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswxdigit()

Description

Is character a hexadecimal digit?

Prototype

```
int iswxdigit(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a hexadecimal digit in the current locale.

iswxdigit_l()

Description

Is character a hexadecimal digit, per locale? (POSIX.1).

Prototype

```
int iswxdigit_l(wint_t c,
                locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a hexadecimal digit in the current locale.

Notes

Conforms to POSIX.1-2017.

iswalpha()

Description

Is character alphabetic?

Prototype

```
int iswalpha(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns true if the character c is alphabetic in the current locale. That is, any character for which *iswupper()* (page 744) or *iswlower()* (page 745) returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of *iswcntrl()* (page 738), *iswdigit()* (page 741), *iswpunct()* (page 740), or *isspace()* (page 520) is true.

In the C locale, *isalpha()* (page 523) returns nonzero (true) if and only if *isupper()* (page 525) or *islower()* (page 526) return true for value of the argument c.

iswalpha_l()

Description

Is character alphabetic, per locale? (POSIX.1).

Prototype

```
int iswalpha_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns true if the wide character c is alphabetic in the locale loc. That is, any character for which *iswupper()* (page 744) or *iswlower()* (page 745) returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of *iswcntrl_l()* (page **??**), *iswdigit_l()* (page **??**), *iswpunct_l()* (page **??**), or *iswspace_l()* (page **??**) is true in the locale loc.

In the C locale, *iswalpha_l()* (page **??**) returns nonzero (true) if and only if *iswupper_l()* (page **??**) or *iswlower_l()* (page **??**) return true for value of the argument c.

Notes

Conforms to POSIX.1-2017.

iswalnum()

Description

Is character alphanumeric?

Prototype

```
int iswalnum(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is an alphabetic or numeric character in the current locale.

iswalnum_l()

Description

Is character alphanumeric, per locale? (POSIX.1).

Prototype

```
int iswalnum_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is an alphabetic or numeric character in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswupper()

Description

Is character an uppercase letter?

Prototype

```
int iswupper(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is an uppercase letter in the current locale.

iswupper_l()

Description

Is character an uppercase letter, per locale? (POSIX.1).

Prototype

```
int iswupper_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is an uppercase letter in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswlower()

Description

Is character a lowercase letter?

Prototype

```
int iswlower(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a lowercase letter in the current locale.

iswlower_l()

Description

Is character a lowercase letter, per locale? (POSIX.1).

Prototype

```
int iswlower_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a lowercase letter in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswprint()

Description

Is character printable?

Prototype

```
int iswprint(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character including space in the current locale.

iswprint_l()

Description

Is character printable, per locale? (POSIX.1).

Prototype

```
int iswprint_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character including space in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswgraph()

Description

Is character any printing character?

Prototype

```
int iswgraph(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character except space in the current locale.

iswgraph_l()

Description

Is character any printing character, per locale? (POSIX.1).

Prototype

```
int iswgraph_l(wint_t c,
               locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character except space in the locale loc.

Notes

Conforms to POSIX.1-2017.

iswctype()

Description

Construct character mapping.

Prototype

```
int iswctype(wint_t   c,
             wctype_t t);
```

Parameters

Parameter	Description
c	Wide character to test.
t	Property to test, typically delivered by calling *wctype()* (page 748).

Return value

Returns nonzero (true) if and only if the wide character c has the property t in the current locale.

Additional information

Determines whether the wide character c has the property described by t in the current locale.

iswctype_l()

Description

Construct character mapping, per locale (POSIX.1).

Prototype

```
int iswctype_l(wint_t   c,
               wctype_t t,
                        locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
t	Property to test, typically delivered by calling wctype_l().
loc	Locale used for mapping.

Return value

Returns nonzero (true) if and only if the wide character c has the property t in the locale loc.

Additional information

Determines whether the wide character c has the property described by t in the locale loc.

Notes

Conforms to POSIX.1-2017.

wctype()

Description

Construct character class.

Prototype

```
wctype_t wctype(char const * name);
```

Parameters

Parameter	Description
name	Name of mapping.

Return value

Character class; zero if class unrecognized.

Additional information

Constructs a value of type wctype_t that describes a class of wide characters identified by the string argument property.

If property identifies a valid class of wide characters in the current locale, returns a nonzero value that is valid as the second argument to *iswctype()* (page 747); otherwise, it returns zero.

Notes

The only mappings supported are:

- “alnum”
- “alpha”,
- “blank”
- “cntrl”
- “digit”
- “graph”
- “lower”
- “print”
- “punct”
- “space”
- “upper”
- “xdigit”

Conversion functions

Function	Description
towupper() (page 749)	Convert uppercase character to lowercase.
towupper_l() (page **??**)	Convert uppercase character to lowercase, per locale (POSIX.1).
towlower() (page 750)	Convert uppercase character to lowercase.
towlower_l() (page **??**)	Convert uppercase character to lowercase, per locale (POSIX.1).
towctrans() (page 751)	Translate character.
towctrans_l() (page **??**)	Translate character, per locale (POSIX.1).
wctrans() (page 752)	Construct character mapping.
wctrans_l() (page **??**)	Construct character mapping, per locale (POSIX.1).

towupper()

Description

Convert uppercase character to lowercase.

Prototype

```
wint_t towupper(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to convert.

Return value

Converted wide character.

Additional information

Converts a lowercase letter to a corresponding uppercase letter.

If the argument c is a wide character for which *iswlower()* (page 745) is true and there are one or more corresponding wide characters, in the current current locale, for which *iswupper()* (page 744) is true, *towupper()* (page 749) returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, c is returned unchanged.

towupper_l()

Description

Convert uppercase character to lowercase, per locale (POSIX.1).

Prototype

```
wint_t towupper_l(wint_t c,
                  locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to convert.
loc	Locale used to convert c.

Return value

Converted wide character.

Additional information

Converts a lowercase letter to a corresponding uppercase letter.

If the argument c is a wide character for which *iswlower_l()* (page **??**) is true and there are one or more corresponding wide characters, in the current locale loc, for which *iswupper_l()* (page **??**) is true, *towupper_l()* (page **??**) returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, c is returned unchanged.

Notes

Conforms to POSIX.1-2017.

towlower()

Description

Convert uppercase character to lowercase.

Prototype

```
wint_t towlower(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to convert.

Return value

Converted wide character.

Additional information

Converts an uppercase letter to a corresponding lowercase letter.

If the argument c is a wide character for which *iswupper()* (page 744) is true and there are one or more corresponding wide characters, in the current locale, for which *iswlower()* (page 745) is true, *towlower()* (page 750) returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, c is returned unchanged.

towlower_l()

Description

Convert uppercase character to lowercase, per locale (POSIX.1).

Prototype

```
wint_t towlower_l(wint_t c,
                  locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to convert.
loc	Locale used to convert c.

Return value

Converted wide character.

Additional information

Converts an uppercase letter to a corresponding lowercase letter.

If the argument c is a wide character for which *iswupper_l()* (page **??**) is true and there are one or more corresponding wide characters, in the locale loc, for which *iswlower_l()* (page **??**) is true, *towlower_l()* (page **??**) returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, c is returned unchanged.

Notes

Conforms to POSIX.1-2017.

towctrans()

Description

Translate character.

Prototype

```
wint_t towctrans(wint_t    c,
                 wctrans_t t);
```

Parameters

Parameter	Description
c	Wide character to convert.
t	Mapping to use for conversion.

Return value

Converted wide character.

Additional information

Maps the wide character c using the mapping described by t in the current locale.

towctrans_l()

Description

Translate character, per locale (POSIX.1).

Prototype

```
wint_t towctrans_l(wint_t    c,
                   wctrans_t t,
                             locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to convert.
t	Mapping to use for conversion.
loc	Locale used for conversion.

Return value

Converted wide character.

Additional information

Maps the wide character c using the mapping described by t in the locale loc.

Notes

Conforms to POSIX.1-2017.

wctrans()

Description

Construct character mapping.

Prototype

```
wctrans_t wctrans(const char * name);
```

Parameters

Parameter	Description
name	Name of mapping.

Return value

Transformation mapping; zero if mapping unrecognized.

Additional information

Constructs a value of type wctrans_t that describes a mapping between wide characters identified by the string argument property.

If property identifies a valid mapping of wide characters in the current locale, *wctrans_l()* (page **??**) returns a nonzero value that is valid as the second argument to *towctrans()* (page 751); otherwise, it returns zero.

The only mappings supported are "tolower" and "toupper".

wctrans_l()

Description

Construct character mapping, per locale (POSIX.1).

Prototype

```
wctrans_t wctrans_l(const char * name,
                    locale_t loc);
```

Parameters

Parameter	Description
name	Name of mapping.
loc	Locale used for mapping.

Return value

Transformation mapping; zero if mapping unrecognized.

Additional information

Constructs a value of type wctrans_t that describes a mapping between wide characters identified by the string argument property.

If property identifies a valid mapping of wide characters in the locale loc, *wctrans_l()* (page **??**) returns a nonzero value that is valid as the second argument to *towctrans()* (page 751); otherwise, it returns zero.

The only mappings supported are "tolower" and "toupper".

Notes

Conforms to POSIX.1-2017.

5.6.25 <xlocale.h>

Locale management

Function	Description
newlocale() (page 754)	Duplicate locale data (POSIX.1).
duplocale() (page 755)	Duplicate locale data (POSIX.1).
freelocale() (page 755)	Free locale (POSIX.1).
localeconv_l() (page **??**)	Get locale data (POSIX.1).

newlocale()

Description

Duplicate locale data (POSIX.1).

Prototype

```
locale_t newlocale(      int         category_mask,
                   const char      * loc,
                         locale_t    base);
```

Parameters

Parameter	Description
category_mask	Locale categories to be set or modified.
loc	Locale name.
base	Base to modify or NULL to create a new locale.

Return value

Pointer to modified locale.

Additional information

Creates a new locale object or modifies an existing one. If the base argument is NULL, a new locale object is created.

category_mask specifies the locale categories to be set or modified. Values for category_mask are constructed by a bitwise-inclusive OR of the symbolic constants LC_CTYPE_MASK, LC_NUMERIC_MASK, LC_TIME_MASK, LC_COLLATE_MASK, LC_MONETARY_MASK, and LC_MESSAGES_MASK.

For each category with the corresponding bit set in category_mask, the data from the locale named by loc is used. In the case of modifying an existing locale object, the data from the locale named by loc replaces the existing data within the locale object. If a completely new locale object is created, the data for all sections not requested by category_mask are taken from the default locale.

The locales "C" and "POSIX" are equivalent and defined for all settings of category_mask:

If loc is NULL, then the "C" locale is used. If loc is an empty string, *newlocale()* (page 754) will use the default locale.

If base is NULL, the current locale is used. If base is LC_GLOBAL_LOCALE, the global locale is used.

If mask is LC_ALL_MASK, base is ignored.

Notes

Conforms to POSIX.1-2008.

POSIX.1-2008 does not specify whether the locale object pointed to by base is modified or whether it is freed and a new locale object created.

The category mask LC_MESSAGES_MASK is not implemented as POSIX messages are not implemented.

duplocale()

Description

Duplicate locale data (POSIX.1).

Prototype

```
locale_t duplocale(locale_t base);
```

Parameters

Parameter	Description
base	Locale to duplicate.

Return value

If there is insufficient memory to duplicate loc, returns a NULL and sets errno to ENOMEM as required by POSIX.1-2008. Otherwise, returns a new, duplicated locale.

Additional information

Duplicates the locale object referenced by loc. Duplicated locales must be freed with *freelocale()* (page 755).

Notes

Conforms to POSIX.1-2008.

freelocale()

Description

Free locale (POSIX.1).

Prototype

```
int freelocale(locale_t loc);
```

Parameters

Parameter	Description
loc	Locale to free.

Return value

Zero on success, -1 on error.

Additional information

Frees the storage associated with loc.

Notes

Conforms to POSIX.1-2008.

localeconv_l()

Description

Get locale data (POSIX.1).

Prototype

```
localeconv_l(locale_t loc);
```

Parameters

Parameter	Description
loc	Locale to inquire.

Return value

Returns a pointer to a structure of type lconv with the corresponding values for the locale loc filled in.

Notes

Conforms to POSIX.1-2008.

5.7 Compiler support API

5.7.1 GNU library API

Integer arithmetic

Function	Description
__mulsi3() (page 757)	Multiply, signed 32-bit integer.
__muldi3() (page 757)	Multiply, signed 64-bit integer.
__divsi3() (page 757)	Divide, signed 32-bit integer.
__divdi3() (page 758)	Divide, signed 64-bit integer.
__udivsi3() (page 758)	Divide, unsigned 32-bit integer.
__udivdi3() (page 759)	Divide, unsigned 64-bit integer.
__modsi3() (page 759)	Remainder after divide, signed 32-bit integer.
__moddi3() (page 759)	Remainder after divide, signed 64-bit integer.
__umodsi3() (page 760)	Remainder after divide, unsigned 32-bit integer.
__umoddi3() (page 760)	Remainder after divide, unsigned 64-bit integer.
__udivmodsi4() (page 761)	Divide with remainder, unsigned 32-bit integer.
__udivmoddi4() (page 761)	Divide with remainder, unsigned 64-bit integer.
__clzsi2() (page 761)	Count leading zeros, 32-bit integer.
__clzdi2() (page 762)	Count leading zeros, 64-bit integer.
__ctzsi2() (page 762)	Count trailing zeros, 32-bit integer.
__ctzdi2() (page 763)	Count trailing zeros, 64-bit integer.
__ffssi2() (page 763)	Find first set, 32-bit integer.
__ffsdi2() (page 763)	Find first set, 64-bit integer.
__bswapsi2() (page 764)	Byte swap, 32-bit integer.
__bswapdi2() (page 764)	Byte swap, 64-bit integer.
__popcountsi2() (page 764)	Population count, 32-bit integer.
__popcountdi2() (page 765)	Population count, 64-bit integer.
__paritysi2() (page 765)	Parity, 32-bit integer.
__paritydi2() (page 765)	Parity, 64-bit integer.

__mulsi3()

Description

Multiply, signed 32-bit integer.

Prototype

```
__SEGGER_RTL_U32 __mulsi3(__SEGGER_RTL_U32 a,
                          __SEGGER_RTL_U32 b);
```

Parameters

Parameter	Description
a	Multiplier.
b	Multiplicand.

Return value

Product.

__muldi3()

Description

Multiply, signed 64-bit integer.

Prototype

```
__SEGGER_RTL_U64 __muldi3(__SEGGER_RTL_U64 a,
                          __SEGGER_RTL_U64 b);
```

Parameters

Parameter	Description
a	Multiplier.
b	Multiplicand.

Return value

Product.

__divsi3()

Description

Divide, signed 32-bit integer.

Prototype

```
int32_t __divsi3(int32_t num,
                 int32_t den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Quotient.

__divdi3()

Description

Divide, signed 64-bit integer.

Prototype

```
int64_t __divdi3(int64_t num,
                 int64_t den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Quotient.

__udivsi3()

Description

Divide, unsigned 32-bit integer.

Prototype

```
__SEGGER_RTL_U32 __udivsi3(__SEGGER_RTL_U32 num,
                           __SEGGER_RTL_U32 den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Quotient.

__udivdi3()

Description

Divide, unsigned 64-bit integer.

Prototype

```
__SEGGER_RTL_U64 __udivdi3(__SEGGER_RTL_U64 num,
                           __SEGGER_RTL_U64 den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Quotient.

__modsi3()

Description

Remainder after divide, signed 32-bit integer.

Prototype

```
int32_t __modsi3(int32_t num,
                 int32_t den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Remainder.

__moddi3()

Description

Remainder after divide, signed 64-bit integer.

Prototype

```
int64_t __moddi3(int64_t num,
                 int64_t den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Remainder.

__umodsi3()

Description

Remainder after divide, unsigned 32-bit integer.

Prototype

```
__SEGGER_RTL_U32 __umodsi3(__SEGGER_RTL_U32 num,
                           __SEGGER_RTL_U32 den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Remainder.

__umoddi3()

Description

Remainder after divide, unsigned 64-bit integer.

Prototype

```
__SEGGER_RTL_U64 __umoddi3(__SEGGER_RTL_U64 num,
                           __SEGGER_RTL_U64 den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Remainder.

__udivmodsi4()

Description

Divide with remainder, unsigned 32-bit integer.

Prototype

```
__SEGGER_RTL_U32 __udivmodsi4(__SEGGER_RTL_U32 num,
                              __SEGGER_RTL_U32 den,
                              __SEGGER_RTL_U32 *rem);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.
rem	Pointer to object that receives the remainder.

Return value

Quotient.

__udivmoddi4()

Description

Divide with remainder, unsigned 64-bit integer.

Prototype

```
__SEGGER_RTL_U64 __udivmoddi4(__SEGGER_RTL_U64 num,
                              __SEGGER_RTL_U64 den,
                              __SEGGER_RTL_U64 *rem);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.
rem	Pointer to object that receives the remainder.

Return value

Quotient.

__clzsi2()

Description

Count leading zeros, 32-bit integer.

Prototype

```
int __clzsi2(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Argument; x must not be zero.

Return value

Number of leading zeros in x.

__clzdi2()

Description

Count leading zeros, 64-bit integer.

Prototype

```
int __clzdi2(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Argument; x must not be zero.

Return value

Number of leading zeros in x.

__ctzsi2()

Description

Count trailing zeros, 32-bit integer.

Prototype

```
int __ctzsi2(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Argument; x must not be zero.

Return value

Number of trailing zeros in x.

__ctzdi2()

Description

Count trailing zeros, 64-bit integer.

Prototype

```
int __ctzdi2(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Argument; x must not be zero.

Return value

Number of trailing zeros in x.

__ffssi2()

Description

Find first set, 32-bit integer.

Prototype

```
int __ffssi2(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Argument; Return value zero if x is zero.

Return value

The index of the least significant 1-bit in x.

__ffsdi2()

Description

Find first set, 64-bit integer.

Prototype

```
int __ffsdi2(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Argument; Return value zero if x is zero.

Return value

The index of the least significant 1-bit in x.

__bswapsi2()

Description

Byte swap, 32-bit integer.

Prototype

```
int __bswapsi2(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The result of byte swap in x.

__bswapdi2()

Description

Byte swap, 64-bit integer.

Prototype

```
int __bswapdi2(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The result of byte swap in x.

__popcountsi2()

Description

Population count, 32-bit integer.

Prototype

```
int __popcountsi2(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Count of number of one bits in x.

__popcountdi2()

Description

Population count, 64-bit integer.

Prototype

```
int __popcountdi2(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Count of number of one bits in x.

__paritysi2()

Description

Parity, 32-bit integer.

Prototype

```
int __paritysi2(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Argument.

Return value

1	number of one bits in x is odd.
0	number of one bits in x is even.

__paritydi2()

Description

Parity, 64-bit integer.

Prototype

```
int __paritydi2(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Argument.

Return value

1	number of one bits in x is odd.
0	number of one bits in x is even.

Floating arithmetic

Function	Description
__addsf3() (page 766)	Add, float.
__adddf3() (page 766)	Add, double.
__addtf3() (page 767)	Add, long double.
__subsf3() (page 767)	Subtract, float.
__subdf3() (page 768)	Subtract, double.
__subtf3() (page 768)	Subtract, long double.
__mulsf3() (page 768)	Multiply, float.
__muldf3() (page 769)	Multiply, double.
__multf3() (page 769)	Multiply, long double.
__divsf3() (page 770)	Divide, float.
__divdf3() (page 770)	Divide, double.
__divtf3() (page 770)	Divide, long double.

__addsf3()

Description

Add, float.

Prototype

```
float __addsf3(float x,
               float y);
```

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

__adddf3()

Description

Add, double.

Prototype

```
double __adddf3(double x,
                double y);
```

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

__addtf3()

Description

Add, long double.

Prototype

```
long double __addtf3(long double x,
                     long double y);
```

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

__subsf3()

Description

Subtract, float.

Prototype

```
float __subsf3(float x,
               float y);
```

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

__subdf3()

Description

Subtract, double.

Prototype

```
double __subdf3(double x,
                double y);
```

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

__subtf3()

Description

Subtract, long double.

Prototype

```
long double __subtf3(long double x,
                     long double y);
```

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

__mulsf3()

Description

Multiply, float.

Prototype

```
float __mulsf3(float x,
               float y);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

__muldf3()

Description

Multiply, double.

Prototype

```
double __muldf3(double x,
                double y);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

__multf3()

Description

Multiply, long double.

Prototype

```
long double __multf3(long double x,
                     long double y);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

__divsf3()

Description

Divide, float.

Prototype

```
float __divsf3(float x,
               float y);
```

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

__divdf3()

Description

Divide, double.

Prototype

```
double __divdf3(double x,
                double y);
```

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

__divtf3()

Description

Divide, long double.

Prototype

```
long double __divtf3(long double x,
                     long double y);
```

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

Floating conversions

Function	Description
__fixsfsi() (page 771)	Convert float to int.
__fixdfsi() (page 772)	Convert double to int.
__fixtfsi() (page 772)	Convert long double to int.
__fixsfdi() (page 773)	Convert float to long long.
__fixdfdi() (page 773)	Convert double to long long.
__fixtfdi() (page 773)	Convert long double to long long.
__fixunssfsi() (page 774)	Convert float to unsigned.
__fixunsdfsi() (page 774)	Convert double to unsigned.
__fixunstfsi() (page 775)	Convert long double to int.
__fixunssfdi() (page 775)	Convert float to unsigned long long.
__fixunsdfdi() (page 775)	Convert double to unsigned long long.
__fixunstfdi() (page 776)	Convert long double to unsigned long long.
__floatsisf() (page 776)	Convert int to float.
__floatsidf() (page 776)	Convert int to double.
__floatsitf() (page 777)	Convert int to long double.
__floatdisf() (page 777)	Convert long long to float.
__floatdidf() (page 777)	Convert long long to double.
__floatditf() (page 778)	Convert long long to long double.
__floatunsisf() (page 778)	Convert unsigned to float.
__floatunsidf() (page 778)	Convert unsigned to double.
__floatunsitf() (page 779)	Convert unsigned to long double.
__floatundisf() (page 779)	Convert unsigned long long to float.
__floatundidf() (page 779)	Convert unsigned long long to double.
__floatunditf() (page 780)	Convert unsigned long long to long double.
__extendsfdf2() (page 780)	Extend float to double.
__extendsftf2() (page 780)	Extend float to long double.
__extenddftf2() (page 781)	Extend double to long double.
__truncdfsf2() (page 781)	Truncate double to float.
__trunctfsf2() (page 782)	Truncate long double to float.
__trunctfdf2() (page 781)	Truncate long double to double.

__fixsfsi()

Description

Convert float to int.

Prototype

```
__SEGGER_RTL_I32 __fixsfsi(float x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

__fixdfsi()

Description

Convert double to int.

Prototype

```
__SEGGER_RTL_I32 __fixdfsi(double x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

__fixtfsi()

Description

Convert long double to int.

Prototype

```
__SEGGER_RTL_I32 __fixtfsi(long double x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

__fixsfdi()

Description

Convert float to long long.

Prototype

```
__SEGGER_RTL_I64 __fixsfdi(float f);
```

Parameters

Parameter	Description
f	Floating value to convert.

Return value

Integerized value.

Notes

The RV32 compiler converts a float to a 64-bit integer by calling runtime support to handle it.

__fixdfdi()

Description

Convert double to long long.

Prototype

```
__SEGGER_RTL_I64 __fixdfdi(double x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

Notes

RV32 always calls runtime for double to int64 conversion.

__fixtfdi()

Description

Convert long double to long long.

Prototype

```
__SEGGER_RTL_I64 __fixtfdi(long double x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

__fixunssfsi()

Description

Convert float to unsigned.

Prototype

```
__SEGGER_RTL_U32 __fixunssfsi(float x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

__fixunsdfsi()

Description

Convert double to unsigned.

Prototype

```
__SEGGER_RTL_U32 __fixunsdfsi(double x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

__fixunstfsi()

Description

Convert long double to int.

Prototype

```
int __fixunstfsi(long double x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

__fixunssfdi()

Description

Convert float to unsigned long long.

Prototype

```
__SEGGER_RTL_U64 __fixunssfdi(float f);
```

Parameters

Parameter	Description
f	Float value to convert.

Return value

Integerized value.

__fixunsdfdi()

Description

Convert double to unsigned long long.

Prototype

```
__SEGGER_RTL_U64 __fixunsdfdi(double x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

__fixunstfdi()

Description

Convert long double to unsigned long long.

Prototype

```
__SEGGER_RTL_U64 __fixunstfdi(long double x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

__floatsisf()

Description

Convert int to float.

Prototype

```
float __floatsisf(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

__floatsidf()

Description

Convert int to double.

Prototype

```
double __floatsidf(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

__floatsitf()

Description

Convert int to long double.

Prototype

```
long double __floatsitf(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

__floatdisf()

Description

Convert long long to float.

Prototype

```
float __floatdisf(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

__floatdidf()

Description

Convert long long to double.

Prototype

```
double __floatdidf(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

__floatditf()

Description

Convert long long to long double.

Prototype

```
long double __floatditf(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

__floatunsisf()

Description

Convert unsigned to float.

Prototype

```
float __floatunsisf(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

__floatunsidf()

Description

Convert unsigned to double.

Prototype

```
double __floatunsidf(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Unsigned value to convert.

Return value

Double value.

__floatunsitf()

Description

Convert unsigned to long double.

Prototype

```
long double __floatunsitf(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Unsigned value to convert.

Return value

Long double value.

__floatundisf()

Description

Convert unsigned long long to float.

Prototype

```
float __floatundisf(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Unsigned long long value to convert.

Return value

Float value.

__floatundidf()

Description

Convert unsigned long long to double.

Prototype

```
double __floatundidf(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Unsigned long long value to convert.

Return value

Double value.

__floatunditf()

Description

Convert unsigned long long to long double.

Prototype

```
long double __floatunditf(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Unsigned long long value to convert.

Return value

Long double value.

__extendsfdf2()

Description

Extend float to double.

Prototype

```
double __extendsfdf2(float x);
```

Parameters

Parameter	Description
x	Float value to extend.

Return value

Double value.

__extendsftf2()

Description

Extend float to long double.

Prototype

```
long double __extendsftf2(float x);
```

Parameters

Parameter	Description
x	Float value to extend.

Return value

Double value.

__extenddftf2()

Description

Extend double to long double.

Prototype

```
long double __extenddftf2(double x);
```

Parameters

Parameter	Description
x	Double value to extend.

Return value

Long double value.

__truncdfsf2()

Description

Truncate double to float.

Prototype

```
float __truncdfsf2(double x);
```

Parameters

Parameter	Description
x	Double value to truncate.

Return value

Float value.

__trunctfdf2()

Description

Truncate long double to double.

Prototype

```
double __trunctfdf2(long double x);
```

Parameters

Parameter	Description
x	Long double value to truncate.

Return value

Double value.

__trunctfsf2()

Description

Truncate long double to float.

Prototype

```
float __trunctfsf2(long double x);
```

Parameters

Parameter	Description
x	Long double value to truncate.

Return value

Float value.

Floating comparisons

Function	Description
__eqsf2() (page 782)	Equal, float.
__eqdf2() (page 783)	Equal, double.
__eqtf2() (page 783)	Equal, long double.
__nesf2() (page 784)	Not equal, float.
__nedf2() (page 784)	Not equal, double.
__netf2() (page 784)	Not equal, long double.
__ltsf2() (page 785)	Less than, float.
__ltdf2() (page 785)	Less than, double.
__lttf2() (page 786)	Less than, long double.
__lesf2() (page 786)	Less than or equal, float.
__ledf2() (page 786)	Less than or equal, double.
__letf2() (page 787)	Less than or equal, long double.
__gtsf2() (page 787)	Greater than, float.
__gtdf2() (page 788)	Greater than, double.
__gttf2() (page 788)	Greater than, long double.
__gesf2() (page 788)	Greater than or equal, float.
__gedf2() (page 789)	Greater than or equal, double.
__getf2() (page 789)	Greater than or equal, long double.

__eqsf2()

Description

Equal, float.

Prototype

```
int __eqsf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

__eqdf2()

Description

Equal, double.

Prototype

```
int __eqdf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

__eqtf2()

Description

Equal, long double.

Prototype

```
int __eqtf2(long double x,
            long double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

__nesf2()

Description

Not equal, float.

Prototype

```
int __nesf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

__nedf2()

Description

Not equal, double.

Prototype

```
int __nedf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

__netf2()

Description

Not equal, long double.

Prototype

```
int __netf2(long double x,
            long double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

__ltsf2()

Description

Less than, float.

Prototype

```
int __ltsf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return < 0 if both operands are non-NaN and a < b (GNU three-way boolean).

__ltdf2()

Description

Less than, double.

Prototype

```
int __ltdf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return < 0 if both operands are non-NaN and a < b (GNU three-way boolean).

__lttf2()

Description

Less than, long double.

Prototype

```
int __lttf2(long double x,
            long double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return < 0 if both operands are non-NaN and a < b (GNU three-way boolean).

__lesf2()

Description

Less than or equal, float.

Prototype

```
int __lesf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≤ 0 if both operands are non-NaN and a < b (GNU three-way boolean).

__ledf2()

Description

Less than or equal, double.

Prototype

```
int __ledf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≤ 0 if both operands are non-NaN and a < b (GNU three-way boolean).

__letf2()

Description

Less than or equal, long double.

Prototype

```
int __letf2(long double x,
            long double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≤ 0 if both operands are non-NaN and a < b (GNU three-way boolean).

__gtsf2()

Description

Greater than, float.

Prototype

```
int __gtsf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return > 0 if both operands are non-NaN and a > b (GNU three-way boolean).

__gtdf2()

Description

Greater than, double.

Prototype

```
int __gtdf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return > 0 if both operands are non-NaN and a > b (GNU three-way boolean).

__gttf2()

Description

Greater than, long double.

Prototype

```
int __gttf2(long double x,
            long double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return > 0 if both operands are non-NaN and a > b (GNU three-way boolean).

__gesf2()

Description

Greater than or equal, float.

Prototype

```
int __gesf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≥ 0 if both operands are non-NaN and a ≥ b (GNU three-way boolean).

__gedf2()

Description

Greater than or equal, double.

Prototype

```
int __gedf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≥ 0 if both operands are non-NaN and a ≥ b (GNU three-way boolean).

__getf2()

Description

Greater than or equal, long double.

Prototype

```
int __getf2(long double x,
            long double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≥ 0 if both operands are non-NaN and a ≥ b (GNU three-way boolean).

5.8 External function interface

This section summarises the functions to be provided by the implementor when integrating `Nuclei C Runtime Library` into an application or library.

5.8.1 I/O functions

Function	Description
__SEGGER_RTL_X_file_read() (page 790)	Read data from file.
__SEGGER_RTL_X_file_write() (page 790)	Write data to file.
__SEGGER_RTL_X_file_unget() (page 791)	Push character back to file.

__SEGGER_RTL_X_file_read()

Description

Read data from file.

Prototype

```
int __SEGGER_RTL_X_file_read(           __SEGGER_RTL_FILE *stream,
                             char     * s,
                             unsigned   len);
```

Parameters

Parameter	Description
stream	Pointer to file to read from.
s	Pointer to object that receives the input.
len	Number of characters to read from file.

Return value

≥ 0	Success.
< 0	Failure.

Additional information

This reads len octets from the file stream into the object pointed to by s.

__SEGGER_RTL_X_file_write()

Description

Write data to file.

Prototype

```
int __SEGGER_RTL_X_file_write(                __SEGGER_RTL_FILE *stream,
                              const char     * s,
                              unsigned   len);
```

Parameters

Parameter	Description
stream	Pointer to stream to write to.
s	Pointer to object to write to stream.
len	Number of characters to write to the stream.

Return value

≥ 0	Success.
< 0	Failure.

Additional information

This writes len octets to the file stream from the object pointed to by s.

__SEGGER_RTL_X_file_unget()

Description

Push character back to file.

Prototype

```
int __SEGGER_RTL_X_file_unget(    __SEGGER_RTL_FILE *stream,
                              int c);
```

Parameters

Parameter	Description
stream	File to push character to.
c	Character to push back to file.

Return value

= EOF	Failed to push character back.
≠ EOF	The character pushed back to the file.

Additional information

This function pushes the character c back to the file so that it can be read again. If c is EOF, the function fails and EOF is returned. One character of pushback is guaranteed; if more than one character is pushed back without an intervening read, the pushback may fail.

5.8.2 Heap protection functions

Function	Description
__SEGGER_RTL_X_heap_lock() (page 792)	Lock heap.
__SEGGER_RTL_X_heap_unlock() (page 792)	Unlock heap.

__SEGGER_RTL_X_heap_lock()

Description

Lock heap.

Prototype

```
void __SEGGER_RTL_X_heap_lock(void);
```

Additional information

This function is called to lock access to the heap before allocation or deallocation is processed. This is only required for multitasking systems where heap operations may possibly be called called from different threads.

__SEGGER_RTL_X_heap_unlock()

Description

Unlock heap.

Prototype

```
void __SEGGER_RTL_X_heap_unlock(void);
```

Additional information

This function is called to unlock access to the heap after allocation or deallocation has completed. This is only required for multitasking systems where heap operations may possibly be called called from different threads.

5.8.3 Error and assertion functions

Function	Description
__SEGGER_RTL_X_assert() (page 792)	User-defined behavior for the assert macro.
__SEGGER_RTL_X_errno_addr() (page 793)	Return pointer to object holding errno.

__SEGGER_RTL_X_assert()

Description

User-defined behavior for the assert macro.

Prototype

```
void __SEGGER_RTL_X_assert(const char * expr,
                           const char * filename,
                           int    line);
```

Parameters

Parameter	Description
expr	Stringized expression that caused failure.
filename	Filename of the source file where the failure was signaled.
line	Line number of the failed assertion.

Additional information

The default implementation of __SEGGER_RTL_X_assert() prints the filename, line, and error message to standard output and then calls *abort()* (page **??**).

__SEGGER_RTL_X_assert() is defined as a weak function and can be replaced by user code.

__SEGGER_RTL_X_errno_addr()

Description

Return pointer to object holding errno.

Prototype

```
int *__SEGGER_RTL_X_errno_addr(void);
```

Return value

Pointer to errno object.

Additional information

The default implementation of this function is to return the address of a variable declared with the __SEGGER_RTL_THREAD storage class. Thus, for multithreaded environments that implement thread-local variables through __SEGGER_RTL_THREAD, each thread receives its own thread-local errno.

It is beyond the scope of this manual to describe how thread-local variables are implemented by the compiler and any associated real-time operating system.

When __SEGGER_RTL_THREAD is defined as an empty macro, this function returns the address of a singleton errno object.

5.8.4 RTC functions

Function	Description
__SEGGER_RTL_X_set_time_of_day() (page 793)	Set RTC time.
__SEGGER_RTL_X_get_time_of_day() (page 794)	Get RTC time.

__SEGGER_RTL_X_set_time_of_day()

Description

Set RTC time.

Prototype

```
int __SEGGER_RTL_X_set_time_of_day(const struct timeval *__tp);
```

Parameters

tp - Pointer to timeval.

Return value

return 0 for success, or -1 for failure.

__SEGGER_RTL_X_get_time_of_day()

Description

Get RTC time.

Prototype

```
int __SEGGER_RTL_X_get_time_of_day(struct timeval *__tp);
```

Parameters

tp - Pointer to timeval.

Return value

return 0 for success, or -1 for failure.

5.8.5 Locale functions

Function	Description
__SEGGER_RTL_X_find_locale() (page 794)	Find locale.

__SEGGER_RTL_X_find_locale()

Description

Find locale.

Prototype

```
const __SEGGER_RTL_locale_t *__SEGGER_RTL_X_find_locale(const char *locale);
```

Parameters

locale - Pointer to zero-terminated locale name.

Return value

Returns a pointer to a locale or NULL if none can be found.

5.9 Changelog

5.9.1 Version 2026.10

New Features:

- Added **RISC-V big-endian** support, the runtime library automatically matches the byte order of the target and no separate library variant needs to be selected. Using the big-endian C library requires the `riscv64be-unknown-elf-` prefixed tools together with `-mbig-endian`, see *Big-endian usage* (page 474).

- Added **zicfilp landing pad** support, all runtime library routines are valid landing pad targets when built with CFI landing pad enabled, so indirect calls guarded by the `zicfilp` extension can safely land in runtime library code, see *zicfilp landing pad usage* (page 474).
- Added the POSIX-compatible headers `malloc.h`, `unistd.h`, `fcntl.h` and `stdalign.h` to prevent include errors when building existing code with `libncrt`.

5.9.2 Version 2025.02

This release contains bug fixes only, see *Version 2025.02* (page 796).

5.9.3 Version 2024.06

New Features:

- Improved the performance of `memset`, `memcpy`, `strlen`, `strcmp`, `strcpy`, `strcat` and the integer to floating-point conversions `__floatsisf`, `__floatunsisf`, `__floatdisf`, `__floatundisf`, `__floatsidf`, `__floatunsidf`, `__floatdidf` and `__floatundidf` under `libncrt_small` with minimal increase in code size.
- Improved the performance of the assembly routines such as `memcpy` and `__divsf3` by optimizing the instruction scheduling to avoid load-to-use stalls.

5.9.4 Version 2023.10

New Features:

- `libncrt` is released as three parts, `libncrt`, '*heapops*' (`basic`/`realtime`/`minimal`) and '*fileops*' (`uart`/`semi`/`rtt`).
- Added the bit operation compiler support APIs `__ctzsi2`, `__ctzdi2`, `__ffssi2`, `__ffsdi2`, `__bswapsi2` and `__bswapdi2`.
- Added optimizations for the `Zcmp` extension (`cm.popret` in the assembly routine epilogues)
- Added the Nuclei `xxlcz` extension (`xl.lb`/`xl.lbu`/`xl.sb`/`xl.lw`/`xl.ffnz`/`xl.nzmsk`/`xl.bezm`/`xl.bnei` etc. in the string and floating-point assembly routines).
- `__SEGGER_RTL_init_heap` is no longer a weak function.

5.10 Known Issues

This section lists the `libncrt` known issues. Issues already fixed in a release are listed under that release as fixed issues.

5.10.1 Version 2026.10

Open Known Issues:

No open known issues in this release.

Fixed Issues:

- Fixed `strcmp` returning a wrong result in some cases when comparing longer strings, refer to issue #66[47] for details.

[47] https://github.com/riscv-mcu/riscv-gnu-toolchain/issues/66

5.10.2 Version 2025.02

Open Known Issues:

No open known issues in this release.

Fixed Issues:

- Fixed the 64-bit integer multiplication `__muldi3` returning a wrong result when built with GCC 14 `-Os`.

5.10.3 Version 2024.06

Open Known Issues:

No open known issues in this release.

Fixed Issues:

- Fixed the double-precision floating-point division `__divdf3` failing on cores which only have the `Zmmul` extension, since it used the `divu` instruction which is not available without the `M` extension.
- Fixed a linker error when linking the UART *'fileops'* library without `-gc-sections` due to the unimplemented `__SEGGER_RTL_X_file_eof`.

CHAPTER

SIX

NUCLEI OPENOCD

6.1 OpenOCD Usage Guide

This document provides comprehensive guidance on OpenOCD installation, configuration, and usage for Nuclei RISC-V CPUs.

6.1.1 Table of Contents

- *Tool Installation and Environment Setup* (page 797)
- *Repositories and Documentation* (page 797)
- *Getting Started* (page 798)
- *Nuclei-Specific Features* (page 799)
- *Configuration File Overview* (page 802)
- *Low-Cost Debugger Solution* (page 806)
- *Frequently Asked Questions* (page 806)
- *Changelog* (page 807)

6.1.2 Tool Installation and Environment Setup

Download and install the latest version of the OpenOCD tool from the Nuclei official website: https://nucleisys.com/download.php

After downloading and extracting to the installation directory, add the following directory to the PATH environment variable:

- `Nuclei\OpenOCD\xxx\bin`

6.1.3 Repositories and Documentation

OpenOCD Repositories

GitHub (RISC-V OpenOCD) Repository[48].

Gitee (RISC-V OpenOCD) Repository[49].

OpenFlashLoader

[48] https://github.com/riscv-mcu/riscv-openocd
[49] https://gitee.com/riscv-mcu/riscv-openocd

Note: The OpenOCD Flashloader enables support for various customized flash programming implementations without requiring modifications to OpenOCD itself. For detailed usage instructions, please refer to the documentation in the source code repository.

GitHub (OpenFlashLoader) Repository[50].

Gitee (OpenFlashLoader) Repository[51].

Note: Nuclei OpenOCD is modified based on OpenOCD with some customized features, so in this guide, we don ' t provide any detailed usage about openocd, please directly refer to the openocd.pdf in our released Nuclei OpenOCD package.

OpenOCD User Guide Location: `openocd/doc/pdf/openocd.pdf`

6.1.4 Getting Started

Checking OpenOCD Version

To determine the installed OpenOCD version:

1. Open a command prompt (Windows) or terminal (Linux)
2. Execute the command: `openocd -v` to see version output

```
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: Version Information parsed for above screenshot not for current release:

- Git Commit ID: 787e48e66
- Compile Date: 2022-12-29 08:49

Running OpenOCD

[50] https://github.com/riscv-mcu/openflashloader
[51] https://gitee.com/riscv-mcu/openflashloader

Common Command Line Parameters

Parameter	Description
-v	Display version information
-d	Set debug level to 3
-d0	Error messages only
-d1	Error and warning messages
-d2	Error, warning, and info (default)
-d3	Error, warning, info, and debug messages
-d4	All messages including low-level debug
-f file	Use specified configuration file
-s dir	Directory to search for config files
-l logfile	Redirect log output to specified file

6.1.5 Nuclei-Specific Features

CPU Information Display

The `nuclei cpuinfo` command provides detailed information about the CPU ' s capabilities, including:

- Supported instruction sets
- Available functional components
- Component configurations

This command simplifies the process of querying CPU features by automatically reading and interpreting the relevant CSR (Control and Status Register) values, eliminating the need for manual register inspection and calculation.

This command implementation is updated with latest nuclei sdk 0.9.0 cpuinfo[52] to provide same analysis using same source code and more detailed information are analyzed from 2025.10 release.

When you use `riscv64-unknown-elf-gdb` to connect to Nuclei OpenOCD GDB debug server, you will be able to use command `monitor nuclei cpuinfo` to dump Nuclei Detailed CPU Information.

NUSPI (Nuclei SPI) Driver

The NUSPI driver provides support for Nuclei ' s SPI controller, which is utilized in Nuclei RISC-V FPGA evaluation boards and other compatible hardware.

Usage:

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

Custom Driver with OpenFlashLoader

The custom driver provides compatibility with various SPI controllers and flash memory types. When using this driver, it must be combined with the OpenFlashLoader to achieve optimal functionality.

Usage:

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

flashloader_path could be one of the following:

[52] https://doc.nucleisys.com/nuclei_sdk/design/app.html#cpuinfo

Note: This requires **2025.10** version to provide correct path search handling.

- Absolute path: eg. `C:\\flashloader\\loader.bin` or `/home/loader/loader.bin`
- Relative path: eg. `loader.bin`, it will search under where openocd executed, and `scripts` folder of where openocd installed such as `toolchain/openocd/scripts`

Nuclei-Specific CSRs

The Nuclei version of OpenOCD supports several custom CSRs (Control and Status Registers). For a complete list and detailed information, refer to:

GitHub (src/target/riscv/encoding.h)[53].

Gitee (src/target/riscv/encoding.h)[54].

Embedded Trace (ETrace) Support

Note: The ETrace feature is currently in the experimental stage and should not be used in production environments.

Some Nuclei CPUs include embedded trace support, enabling detailed examination of instruction execution. The trace functionality is managed through an Embedded Trace (ETrace) module integrated into the CPU ' s scan chains.

Current Implementation:

- RISC-V ETrace Instruction Trace (available)
- Data Trace (not yet implemented)

ETrace Commands:

1. **Configuration:**
   - `nuclei etrace config etrace-addr buffer-addr buffer-size wrap`: Initializes ETrace and configures operational parameters
2. **Control:**
   - `nuclei etrace enable`: Activates ETrace functionality
   - `nuclei etrace disable`: Deactivates ETrace functionality
   - `nuclei etrace start`: Begins trace data collection
   - `nuclei etrace stop`: Stops trace data collection
3. **Data Management:**
   - `nuclei etrace dump filename`: Exports captured trace data to a file
   - `nuclei etrace clear`: Resets trace buffer pointers
   - `nuclei etrace info`: Displays current ETrace status

Note: For detailed command documentation, please refer to `openocd.pdf`. The ETrace feature is also available in Nuclei Studio IDE.

[53] https://github.com/riscv-mcu/riscv-openocd/blob/nuclei/2024/src/target/riscv/encoding.h#L3109
[54] https://gitee.com/riscv-mcu/riscv-openocd/blob/nuclei/2024/src/target/riscv/encoding.h#L3109

Debug Map Feature

Note: The debug map for each hardware thread (hart) is automatically read and displayed during OpenOCD initialization. Alternatively, you can access the debug map at runtime using the `examine_cpu_core` command.

For detailed documentation about the Nuclei debug map feature, please contact your application engineer.

Commands:

- `nuclei expose_cpu_core`
  - Configures the list of indices for `nuclei_examine_cpu_core`
  - Must be executed before the `init` command
- `nuclei examine_cpu_core`
  - Returns a 64-bit value combining `dm-custom1` and `dm-custom2` registers
  - Value calculation: `(dm-custom2 << 32) + dm-custom1`

Cross-Trigger Interface (CTI)

The Cross-Trigger Interface (CTI) provides an advanced debugging mechanism that enables developers to:

- Trigger specific debugging actions
- Synchronize multiple debugging-related events

Available Commands:

- `nuclei cti halt_group on/off target_name0 target_name1 ...`
  - Controls halt group triggers
- `nuclei cti resume_group on/off target_name0 target_name1 ...`
  - Controls resume group triggers

Debug with Hypervisor

Need **2025.10** version to work with, only work with `riscv virt2phys_mode sw`, see riscv-openocd issue #1278[55]

Reset and Halt Command

The `init resethalt` command addresses situations where:

- The CPU becomes unresponsive due to software issues, the debugger and development board cannot establish a normal connection, and debugging is still not possible after power cycling or software reset.

The `init resethalt` command works by sending a reset and halt request to the CPU via the JTAG interface. After reset, the CPU automatically enters the halt state after executing only one instruction, preventing it from running problematic software code and enabling normal debugging.

[55] https://github.com/riscv-collab/riscv-openocd/issues/1278#issuecomment-3166660685

FTDI nSCAN1 Mode Command

The `ftdi nscan1_mode` command controls Nuclei ' s Compact JTAG (cJTAG) mode functionality.

Usage:

```
ftdi nscan1_mode on|off
```

- **on**: Means select 2-wire cJTAG mode
- **off**: Means select 4-wire JTAG mode

Note: This command follows the same syntax and behavior as the standard `ftdi oscan1_mode` command.

6.1.6 Configuration File Overview

The OpenOCD configuration file defines how to establish a connection with the development board through the debug interface. Nuclei provides a sample configuration file that can be adapted to specific hardware requirements.

Example Configuration:

- Using Nuclei HBird Debugger (FTDI-based), see Reference implementation[56].

Debugger Speed Configuration

To adjust the debugger communication speed:

```
adapter_khz 1000
adapter speed 1000
```

Both commands set the debugger speed to 1000 kHz.

Debugger Interface Configuration

The following configuration selects and initializes the FTDI 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
```

Configuration Details:

- FTDI chip VID/PID must match the connected hardware
- JTAG transport protocol selected
- Signal layout configured for HBird Debugger compatibility

[56] https://github.com/Nuclei-Software/nuclei-sdk/blob/master/SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg

Debugger Mode Configuration

OpenOCD supports two debugging modes:

- **JTAG Mode**: Enabled with `ftdi nscan1_mode off`
- **Compact JTAG (cJTAG) Mode**: Enabled with `ftdi nscan1_mode on`

JTAG Link Configuration

The JTAG link configuration varies depending on the system architecture:

Single Core System

This is a n900 single core system example.

```
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 (Symmetric Multiprocessing) System

This is a n900-smp4 SMP system example.

```
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 create $_TARGETNAME0.3 riscv -chain-position $_TARGETNAME0 -coreid 3
target smp $_TARGETNAME0.0 $_TARGETNAME0.1 $_TARGETNAME0.2 $_TARGETNAME0.3
```

AMP (Asymmetric Multiprocessing) System

This is a n300 + n900-smp2 AMP system example.

```
set _CHIPNAME0 riscv0
jtag newtap $_CHIPNAME0 cpu -irlen 5 -expected-id 0x10300a6d

set _CHIPNAME1 riscv1
jtag newtap $_CHIPNAME1 cpu -irlen 5 -expected-id 0x10900a6d

set _TARGETNAME0 $_CHIPNAME0.cpu
target create $_TARGETNAME0 riscv -chain-position $_TARGETNAME0 -coreid 0

set _TARGETNAME1 $_CHIPNAME1.cpu
target create $_TARGETNAME1.0 riscv -chain-position $_TARGETNAME1 -coreid 0 -rtos␣
↪hwthread
target create $_TARGETNAME1.1 riscv -chain-position $_TARGETNAME1 -coreid 1
target smp $_TARGETNAME1.0 $_TARGETNAME1.1
```

Note: The `-rtos hwthread` option enables OpenOCD's pseudo RTOS functionality, which:

- Presents CPU cores ("hardware threads") as threads to GDB
- Allows inspection of SMP system state through GDB commands
- Enables core-specific debugging operations:

- `info threads` lists active CPU cores
- `thread` switches between CPU core views
- `step` and `stepi` operate on individual cores

Work Area Configuration

The work area is a dedicated memory region that accelerates various operations, including:

- Memory read/write operations
- Execution of small program fragments
- Flash memory operations

Configuration Example:

```
$_TARGETNAME0 configure -work-area-phys 0x08000000 -work-area-size 0x10000 -work-
↪area-backup 1
```

Note: Work Area Requirements:

- Must be a readable, writable, and executable memory region
- Base address (0x08000000) and size (0x10000) should be adjusted according to system requirements

NOR Flash Configuration

The NOR flash configuration specifies the memory mapping and controller settings:

```
set _FLASHNAME0 $_CHIPNAME0.flash
flash bank $_FLASHNAME0 nuspi 0x20000000 0 0 0 $_TARGETNAME0.0 0x10180000
```

Note: Configuration Parameters:

- `nuspi`: Flash driver type (adjust as needed)
- `0x20000000`: QSPI XIP address (adjust as needed)
- `0x10180000`: QSPI controller base address (adjust as needed)

Debugger Connection Specification

When multiple debuggers are present in the debugging environment, you can specify which debugger to connect to using:

```
ftdi_serial FT4YR31I
```

Replace "FT4YR31I" with the actual serial number of your debugger.

Debugging Service Ports Configuration

OpenOCD provides three debugging service ports:

1. **GDB Port** (default: 3333)
2. **Telnet Port** (default: 4444)
3. **TCL Port** (default: 6666)

Configuration Example:

```
gdb_port 3333
telnet_port 4444
tcl_port 6666
```

Note:

- Port numbers can be customized if the default ports are unavailable
- To disable a service, set its port to `disabled`
- Ensure port numbers don ' t conflict with other services

Semihosting Support

OpenOCD supports ARM semihosting, which allows target programs to use host system resources. To enable:

```
arm semihosting enable
```

Note: Semihosting provides access to:

- File I/O operations
- Console input/output
- System clock information
- Other host system services

For more detailed information about how to use openocd, please check the `openocd.pdf` distributed in openocd release.

Target Defer Examine

In some multicore systems, the slave-target is not debuggable and needs to be unlocked by the host-target before it can be used, in which case the `-defer-examine` parameter is needed.

```
# Configuring the "-defer-examine" parameter
$SLAVE_TARGETNAME configure -defer-examine

# Override events, the following events will trigger reset, slave-target needs to␣
↪be initialized manually after reset
$_TARGETNAME1_0 configure -event gdb-flash-erase-start {
  init
}
$_TARGETNAME1_0 configure -event gdb-flash-write-end {
  init
}
```

(continues on next page)

(continued from previous page)

```
# Initialization, since the slave-target specifies the "-defer-examine" parameter,␣
↪only the other targets will be initialized.
init

# The host-target unlocks the slave-target with the command configuration registers
$HOST_TARGETNAME dmi_write/dm_write
$HOST_TARGETNAME dmi_read/dm_read

# Manually initialize slave-target
$SLAVE_TARGETNAME arp_examine
```

Segmentation Fault Caused by Missing Configuration in Hummingbird-Debugger-Kit

When debugging with Hummingbird-Debugger-Kit and OpenOCD, you may encounter a "segmentation fault" issue. This is most likely caused by the following missing content in the OpenOCD configuration file:

```
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
```

6.1.7 Low-Cost Debugger Solution

Nuclei provides an affordable debugging solution called DLink[57] for Nuclei RISC-V CPUs:

- Supports both JTAG and cJTAG protocols
- Fully compatible with Nuclei Studio
- Open-source implementation available

6.1.8 Frequently Asked Questions

For additional troubleshooting and common issues, please refer to:

- GitHub FAQ[58]
- Gitee FAQ[59]
- Known Issues and common questions[60]

[57] https://github.com/Nuclei-Software/nuclei-dlink
[58] https://github.com/riscv-mcu/riscv-openocd/wiki
[59] https://gitee.com/riscv-mcu/riscv-openocd/wikis
[60] https://github.com/riscv-mcu/riscv-openocd/issues/11

6.1.9 Common Problem Troubleshooting and Solutions

Reading DTM Registers (idcode/dtmcs) using OpenOCD

OpenOCD can only read DTM registers using a combination of irscan+drscan commands, and cannot directly read the idcode register.

- You can add the following commands directly in the configuration file (taking reading idcode register as an example)

```
irscan $_TARGETNAME 0x01
set idcode [drscan $_TARGETNAME 32 0x00]
puts "IDCODE = 0x$idcode"
```

- Reading directly in GDB will not return the correct value because OpenOCD is continuously polling the board status in the background (polling changes the IR register state). After you execute irscan to set IR=0x01, before the next drscan is executed, OpenOCD ' s polling may have already changed IR to another value, resulting in an incorrect read. The actual execution result is as follows:

```
(gdb) monitor irscan riscv.cpu x1
(gdb) monitor drscan riscv.cpu 32 0
00000000
```

6.1.10 Changelog

Version 2026.10

This release is based on **2025.10** version with some new features and bug fixes introduced.

- Add `nuclei trace` command group for trace-bundle instruction-trace and data-trace support
- Update `nuclei etrace` command group for ATB2AXI instruction-trace support
- Update `nuclei cpuinfo` command implementation using nuclei-sdk same source code to provide same behavior
- Sync nuclei custom CSR support for eclic_v2
- Modify N100 `vlenb`/`mtopi`/`mtopei` display
- Add flash support for HG25Q16B, GD25LE16E, GD25LQ128E, MX25R8035F, XTD25W02A, XTD25W04A, GT25Q16A, GT25Q32EB, GT25Q64EB
- Add dummy flash driver to support automatic hardware breakpoint placement in specified address ranges
- Update custom flash driver with `wrsize` support, flash-id based device matching, and fix memory issues in custom.c and nuspi.c
- Fix FTD2xx vendor IDs and product IDs to resolve matching issues when multiple FTDI devices are present
- Modify GD32VF103 IDCODE to correctly display chip information on the target
- Update TraceDecoder to v1.0.0, see *TraceDecoder Changelog* (page 818)

Version 2025.10

This release is based on **2025.02** version with some new features and bug fixes introduced.

- Update `nuclei cpuinfo` command implementation using nuclei sdk 0.9.0 cpuinfo same source code to provide same behavior
  - When you connect to your FPGA evaluation board or Chip with Nuclei CPU inside it using Nuclei OpenOCD, you will be able to see message like this: `Info: [riscv.cpu] Nuclei CPU Detected: mhartid-0x0 marchid-0x0300 v5.2.0, ISA: RV32ABCIM, Feature: ECLIC, ILM-128K, DLM-128K, IC-32K`, sometimes it may not show, it is expected.
- Correct custom flashloader binary file location search rules, where openocd executed, openocd `scripts` folder, and absolute path
- Sync nuclei custom CSRs support
- Optimize flash probe process, when no flash probed, it will exit with error instead of hang on it
- Support debug for hypervisor mode
- Fix debug `Assertion failed!` on HBird opensource cpu
- Fix nuclei riscv cross trigger command implementation `nuclei cti xxx`
- Clear `haltreq` signal when openocd connect failed
- Fix segmentation fault issue when debug level set to 3 or 4
- When debug on FPGA CPU MHz is very slow such as **5MHz**, openocd may work not correctly, it will be fixed by make CPU run faster such as **10MHz**

Version 2025.02

In this release, Nuclei OpenOCD bump openocd code base from **0.11** to **0.12**, with this base changes, a lot of new features supported by upstream will be taken into Nuclei OpenOCD.

Known Issues:

- There is a probability that writing flash under smp multi-core architecture will fail, which can be solved by clarifying the number of SMP cores or reducing the JTAG frequency.

New Features:

- Live watch feature implementation, see *Using Live Watch* (page 306)
- Nuclei CTI command group support
- Enhanced ETrace command group functionality, see *Trace* (page 250)
- Optimized CPU information command

Improvements:

- Continuous integration and documentation enhancements
- Code organization: consolidated Nuclei commands into `nuclei_riscv.c`
- Register access optimization: replaced `vslide1down_vx` with direct `vx` register access for lite vpu case

CSR Updates:

New Custom CSRs:

Address Range	CSR Name
0x1a4~0x1af	smpuaddr4~15
0x1c0~0x1ef	smpuaddr16~63

CSR Renaming:

Old Name	New Name
spmpcfg0~3	smpucfg0~3
spmpaddr0~15	smpuaddr0~15
mfp16mode	mmisc_ctl1
mecc_ctrl	mecc_ctl
mstack_ctrl	mstack_ctl

Base Version:

- Changes based on riscv-collab/riscv-openocd@f82c5a7[61].

6.1.11 Known Issues

Nuclei OpenOCD Known Issues. For more complex issues, we create separate issues on GitHub. Please refer to GitHub Issues[62].

Custom Flash Driver Memory Out-of-Bounds Bug

Note: This issue affects the 2025.10 OpenOCD, and is fixed in the 2026.10 OpenOCD.

When using the custom flash driver on Linux systems, a memory out-of-bounds issue may occur. This is caused by the custom flash driver not accounting for the null terminator (' 0 ') during malloc. The error log is as follows:

```
malloc():invalid next size(unsorted)
```

Multiple vid_pid Matching Failure Bug

Note: This issue affects the 2025.10 OpenOCD, and is fixed in the 2026.10 OpenOCD.

When multiple vid_pid pairs are configured, only the first vid_pid pair is used, while the others are ignored. This causes subsequent vid_pid devices to fail to find the debug device during connection. The error log is as follows:

```
Error: unrecognized USB device descriptor
Error: unable to open ftdi device with description '*', serial '*' at bus␣
↪location '*'
```

First Connection basic_cpuinfo Read Failure Bug

Note: This issue affects the 2025.10 OpenOCD, and is fixed in the 2026.10 OpenOCD.

An error occurs on the first connection with OpenOCD, but reconnecting with the same debug environment works fine. This is caused by an issue with the basic_cpuinfo call location leading to CSR read failure. The error log is as follows:

[61] https://github.com/riscv-collab/riscv-openocd/commit/f82c5a7
[62] https://github.com/riscv-mcu/riscv-openocd/issues/11

```
Error: [riscv.cpu] Unexpected read of fp via program buffer.
Warn : [riscv.cpu] Couldn't read vlenb; vector register access won't work.
Error: [riscv.cpu] Unexpected read of fp via program buffer.
Error: [riscv.cpu] Unexpected read of fp via program buffer.
Error: [riscv.cpu] Examination failed
```

TraceDecoder Decoding Error on Chinese-Locale Linux Systems

Note: This issue affects the v0.9.0 TraceDecoder shipped with 2025.10, and is fixed in the v1.0.0 TraceDecoder shipped with 2026.10.

When using tracedecoder on Linux systems with Chinese locale settings, ELF file parsing errors may occur. This is caused by tracedecoder not accounting for the Chinese locale during decoding. The error log is as follows:

```
Error: parseElfName():expected
Error: parseobjDump():expected file name and type
Error: ElfReader::ElfReader(): objdump() failed
Error: Trace::buildElfProcess(): Error creating ElfReader object
Error: Trace::configure(): buildElfProcess()failed
Error: new Trace(trace.bin,rom.elf)failed
```

TraceDecoder Runtime Library Mismatch on Linux Systems

Note: This issue affects the v0.9.0 TraceDecoder shipped with 2025.10, and is fixed in the v1.0.0 TraceDecoder shipped with 2026.10.

When running tracedecoder on Linux systems, a runtime library mismatch error may occur. This is caused by incompatibility between the runtime library versions on the test system and those used when compiling tracedecoder. The error log is as follows:

```
bin/dqr: /lib64/libc.so.6: version 'GLIBc 2.14' not found (required by bin/dqr)
bin/dqr: /usr/lib64/libstdc++,so.6: version 'GLIBcxx 3.4.21'not found (required by␣
↪bin/dqr)
```

TraceDecoder May Lose a Few Instructions Before the Sync Packet

Note: This issue affects the v0.9.0 TraceDecoder shipped with 2025.10, and is fixed in the v1.0.0 TraceDecoder shipped with 2026.10.

TraceDecoder may lose a few instructions before the sync packet. Previously, upon receiving a sync packet, the address within the packet was directly used as the next instruction address to continue parsing. We have now adjusted the decoding logic so that upon receiving a sync packet, instruction inference is still attempted first, rather than directly using the address within the packet as the next instruction address to continue parsing.

6.1.12 Other Debuggers

For information on debugging Nuclei RISCV CPU with JLink and Lauterbach, please refer to the links below:

Jlink[63].

Lauterbach[64].

6.2 OpenOCD Capture Trace Data

This document guides users on how to use the OpenOCD tool to capture Trace data via command line.

6.2.1 Table of Contents

- *Tool Installation and Environment Setup* (page 811)
- *ETrace Version Overview* (page 811)
- *Etrace Command Reference* (page 812)
- *Trace Command Reference* (page 816)

6.2.2 Tool Installation and Environment Setup

Download and install the latest version of the OpenOCD tool from the Nuclei official website: https://nucleisys.com/download.php

After downloading and extracting to the installation directory, add the following directory to the PATH environment variable:

- `Nuclei\OpenOCD\xxx\bin`

6.2.3 ETrace Version Overview

Nuclei processors support different versions of ETrace functionality. The command group you should use depends on the ETrace version implemented in your Core:

- **ETrace-v1.x**: Uses the ATB2AXI component as the SoC system bridge. This version supports instruction Trace capture only. If your Core implements etrace-v1.x, see *Etrace Command Reference* (page 812).
- **ETrace-v2.x**: Uses the TraceBundle component as the SoC system bridge. This version supports both instruction Trace and data Trace capture. If your Core implements etrace-v2.x, see *Trace Command Reference* (page 816).

To determine which ETrace version your Core supports, check the processor documentation or the Trace module configuration in your SoC.

[63] https://doc.nucleisys.com/nuclei_tools/ide/projectrun.html#debugging-and-running-a-project-with-j-link
[64] https://github.com/Nuclei-Software/lauterbach-startup

6.2.4 Etrace Command Reference

This command group is for Cores with etrace-v1.x version Trace functionality, where the SoC system component is ATB2AXI. It is used for instruction Trace capture on such systems.

Note: The commands in this section are OpenOCD commands executed in the OpenOCD Tcl shell. If you are accessing these commands from GDB, please add the `monitor` prefix, for example: `monitor nuclei etrace config <atb2axi-addr> <buffer-addr> <buffer-size> <wrap>`

etrace config

Configure the ETrace system.

Syntax:

```
etrace config <atb2axi-addr> <buffer-addr> <buffer-size> <wrap>
```

Parameters:

Parameter	Description	Range/Values
atb2axi-addr	Base address of the ATB2AXI configuration registers	Physical address
buffer-addr	Start address of the trace buffer	Physical address
buffer-size	Size of the trace buffer	In bytes, e.g., 0x10000
wrap	Buffer mode	0 = compact mode, 1 = wrap mode

Buffer Modes:

- **Wrap mode (wrap=1)**: When the buffer is full, new trace data wraps around to overwrite old data
- **Compact mode (wrap=0)**: Trace data is compacted to save space, and stops when the buffer is full

Notes:

- ETrace must be stopped before reconfiguration
- If ETrace is running, an error will be reported

etrace enable

Enable ETrace for the current target Core.

Syntax:

```
etrace enable
```

Notes:

- Uses hardware trigger to set trace-on action at the current PC address
- An error will be reported if no hardware trigger is available

etrace disable

Disable ETrace for the current target Core.

Syntax:

```
etrace disable
```

Notes:

- Uses hardware trigger to set trace-off action at the current PC address
- An error will be reported if no hardware trigger is available

etrace start

Configure ATB2AXI to start capturing Trace data.

Syntax:

```
etrace start
```

etrace stop

Configure ATB2AXI to stop capturing Trace data.

Syntax:

```
etrace stop
```

etrace dump

Read the ATB2AXI register status and save the captured Trace data to the specified file.

Syntax:

```
etrace dump <filename>
```

Notes:

- Filename must be a valid path to a writable file
- If the file already exists, it will be overwritten
- If the file is not writable, an error will be reported

etrace clear

Stop capturing Trace data and clear the ATB2AXI register status.

Syntax:

```
etrace clear
```

etrace info

Display configuration information and show the current data capture status.

Syntax:

```
etrace info
```

Preparation

Connect to the development board using OpenOCD + GDB to enter debug mode, load the ELF file to be debugged into the development board, and run the program to the position where Trace data capture needs to start. The following commands are all executed in GDB.

```
PS D:\Git\nuclei-sdk> make CORE=n900 run_openocd
make -C application/baremetal/helloworld run_openocd
make[1]: Entering directory 'D:/Git/nuclei-sdk/application/baremetal/helloworld'
"Start openocd server"
openocd -c "set BOOT_HARTID 0;; gdb_port 3333"  -f ../../../SoC/evalsoc/Board/
↪nuclei_fpga_eval/openocd_evalsoc.cfg
Open On-Chip Debugger 0.12.0+dev-03629-g3a2163834-dirty (2026-05-21-00:06)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
DEPRECATED! use 'gdb port', not 'gdb_port'
boot hart id expected to be 0
Set default SMP CPU count to default 1
Info : libusb_open() failed with LIBUSB_ERROR_NOT_FOUND
Info : no device found, trying D2xx driver
Info : D2xx device count: 2
Info : Connecting to "FTDI JTAG" using D2xx mode...
Info : clock speed 1000 kHz
Info : JTAG tap: riscv.cpu tap/device found: 0x10600a6d (mfg: 0x536 (Nuclei System␣
↪Technology Co Ltd), part: 0x0600, ver: 0x1)
Info : coreid=0, nuclei debug map reg 00: 0x200001ec, 16: 0x0, 32: 0x20000900
Info : [riscv.cpu] datacount=4 progbufsize=8
Info : [riscv.cpu] Nuclei CPU Detected: mhartid-0x0 marchid-0x0600 v4.6.0, ISA:␣
↪RV32ABCDFIMPU, Feature: ECLIC, PPI, NICE, CMO, ILM-256K, DLM-256K, IC-32K, DC-32K
Info : [riscv.cpu] Examined RISC-V core
Info : [riscv.cpu]  XLEN=32, misa=0x4010912f
[riscv.cpu] Target successfully examined.
Info : [riscv.cpu] Examination succeed
Info : [riscv.cpu] starting gdb server on 3333
Info : Listening on port 3333 for gdb connections
riscv.cpu halted due to debug-request.
Info : Valid NUSPI on device Nuclei SoC SPI Flash at address 0x20000000 with␣
↪spictrl regbase at 0x10014000
Info : Nuclei SPI controller version 0xee010102
Info : Found flash device 'win w25q256fv/jv' (ID 0x001940ef)
semihosting is enabled
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

```
PS D:\Git\nuclei-sdk\application\baremetal\demo_eclic> make CORE=n900 run_gdb
"Run gdb to connect openocd server and debug"
riscv64-unknown-elf-gdb demo_eclic.elf  -ex "set remotetimeout 240" -ex "target␣
↪extended-remote localhost:3333"
GNU gdb (GDB) 16.2.90.20250210-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
```

(continues on next page)

(continued from previous page)

```
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-w64-mingw32 --target=riscv64-unknown-elf
↪".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from demo_eclic.elf...
Remote debugging using localhost:3333
0x200001f0 in ?? ()
(gdb) load
Loading section .init, size 0x248 lma 0x80000000
Loading section .text, size 0x2db8 lma 0x80000280
Loading section .data, size 0x628 lma 0x90000000
Start address 0x80000100, start size 13864
Transfer rate: 76 KB/sec, 4621 bytes/write.
(gdb) b main
Breakpoint 1 at 0x80001aae: file demo_eclic.c, line 93.
(gdb) c
Continuing.

Breakpoint 1, main (argc=0, argv=0x0) at demo_eclic.c:93
93          mcfg_info.d = __RV_CSR_READ(CSR_MCFG_INFO);
(gdb) b demo_eclic.c:142
Breakpoint 2 at 0x80001b72: file demo_eclic.c, line 142.
(gdb)
```

Usage Example

```
# Configure ETrace with 64KB buffer in wrap mode
monitor nuclei etrace config 0x10020000 0x90000000 0x10000 1

# Enable ETrace
monitor nuclei etrace enable

# Start tracing
monitor nuclei etrace start

# Run your application
continue

# Press Ctrl+C or stop at breakpoint

# Disable ETrace
monitor nuclei etrace disable

# Stop tracing
monitor nuclei etrace stop

# View trace info (optional)
monitor nuclei etrace info

# Dump trace data
monitor nuclei etrace dump /tmp/trace.bin
```

(continues on next page)

(continued from previous page)

```
# Clear trace status (optional)
monitor nuclei etrace clear
```

Common Problem Troubleshooting and Solutions

- When wrap=0, ATB2AXI ' s compact mode is enabled, and tdasize is fixed at 8 bits. TraceDecoder needs to specify tdasize as 8 bits during decoding.
- In heterogeneous multi-core systems (AMP) where Trace data is captured simultaneously, you need to use `monitor nuclei etrace enable/disable` in respective GDB terminals to enable and disable ETrace separately. Other commands can be executed in any GDB terminal corresponding to any Core.
- In homogeneous multi-core systems (SMP) where Trace data is captured simultaneously, you need to use the `thread` command in the GDB terminal to switch to the specified Core, and then use `monitor nuclei etrace enable/disable` to enable and disable ETrace separately. Other commands can be executed in any thread.

6.2.5 Trace Command Reference

This command group is for Cores with etrace-v2.x version Trace functionality, where the SoC system component is TraceBundle. It is used for instruction Trace or data Trace capture on such systems.

Note: The commands in this section are OpenOCD commands executed in the OpenOCD Tcl shell. If you are accessing these commands from GDB, please add the `monitor` prefix, for example: `monitor nuclei trace config <tracebundle-addr> <buffer-addr> <buffer-size> <wrap> <core-mask> <timestamp> <datatrace>`

---

trace config

Configure the trace system before starting trace collection.

Syntax:

`trace config <tracebundle-addr> <buffer-addr> <buffer-size> <wrap> <core-mask> <timestamp> <datatrace>`

Parameters:

Parameter	Description	Range/Values
tracebundle-addr	Base address of the trace bundle registers	Physical address
buffer-addr	Start address of the trace buffer in RAM	Physical address
buffer-size	Size of the trace buffer in bytes	Power of 2, e.g., 0x10000 (64KB)
wrap	Wrap mode enable	0 = stop on full, 1 = wrap around
core-mask	Bitmask of cores to trace	Bit i = core i (e.g., 0x3 = cores 0 and 1)
timestamp	Timestamp configuration	See Timestamp Configuration section below
datatrace	Data trace configuration	See Data Trace Configuration section below

Timestamp Configuration:

The `timestamp` parameter is an 8-bit value:

- Bits [7:4]: `timestamp-div` - Divider for timestamp counter
- Bits [3:0]: `timestamp-src` - Timestamp source

  - 0: No timestamp
  - 1: CPU timestamp, support divider (from each core)
  - 2: Funnel timestamp, support divider (shared)
  - 3: External timestamp

Data Trace Configuration:

The `datatrace` parameter is a 5-bit value:

- Bit [4]: `has-store` - Enable store tracing, when datatrace-en is 1
- Bit [3]: `has-load` - Enable load tracing, when datatrace-en is 1
- Bit [2]: `has-value` - Include data value in trace, when datatrace-en is 1
- Bit [1]: `has-addr` - Include data address in trace, when datatrace-en is 1
- Bit [0]: `datatrace-en` - Enable data tracing

Notes:

- The trace system must be configured before starting trace collection
- Buffer size should be aligned to power of 2
- When wrap mode is disabled (wrap=0), tracing stops when buffer is full
- When multiple cores are enabled (core-mask has multiple bits), srcid is automatically configured

trace start

Start trace collection.

Syntax:

```
trace start
```

trace stop

Stop trace collection.

Syntax:

```
trace stop
```

trace dump

Dump trace data to a file.

Syntax:

```
trace dump <filename>
```

trace info

Show trace configuration and status.

Syntax:

```
trace info
```

Preparation

Connect to the development board using OpenOCD + GDB to enter debug mode, load the ELF file to be debugged into the development board, and run the program to the position where Trace data capture needs to start. The following commands are all executed in GDB.

Usage Example

```
# Configure TraceBundle with 64KB buffer in wrap mode, trace core0, enable CPU␣
↪timestamp (no divider), disable data trace
monitor nuclei trace config 0x10060000 0x90000000 0x10000 1 1 1 0

# Start tracing
monitor nuclei trace start

# Run your application
continue

# Press Ctrl+C or stop at breakpoint

# Stop tracing
monitor nuclei trace stop

# View trace info (optional)
monitor nuclei trace info

# Dump trace data
monitor nuclei trace dump /tmp/trace.bin
```

6.2.6 Common Problem Troubleshooting and Solutions

- Trace capture is triggered by configuring TraceBundle registers, so hardware trigger resources are no longer occupied
- In multi-core systems where Trace is captured simultaneously, you need to specify the core-mask parameter in trace config. For example, core-mask=0x3 means capturing Trace data from Core 0 and Core 1. Whether it is Core 0 or Core 1 is distinguished by the TraceBundle module interface signals.

6.3 TraceDecoder Decoding Trace Data

TraceDecoder is a tool for decoding Nuclei RISC-V architecture trace data. It supports reconstructing instruction streams and data streams from Etrace/TraceBundle format trace data. The tool provides a standalone executable `dqr` and dynamic libraries (`dqr.dll` or `libdqr.so`).

6.3.1 Table of Contents

- *Tool Installation and Environment Setup* (page 819)
- *Features* (page 819)
- *Command Line Usage* (page 819)
- *Usage Examples* (page 824)
- *Common Problem Troubleshooting and Solutions* (page 831)

6.3.2 Tool Installation and Environment Setup

Download and install the latest version of the TraceDecoder tool from the Nuclei official website: https://nucleisys.com/download.php

After downloading and extracting to the installation directory, add the following directory to the PATH environment variable:

- `Nuclei\TraceDecoder\xxx\bin`

6.3.3 Features

- Supports 32-bit and 64-bit trace data decoding
- Provides both command-line tool and dynamic library usage modes
- Cross-platform support (Windows, Linux, macOS)
- Can reconstruct program execution flow for debugging and performance analysis

6.3.4 Command Line Usage

Basic Syntax

```
Usage: dqr -t <tracefile> -e <elffile> [-od <objdump-path>] [options]

-t <tracefile>:   Specify the name of the Nuclei trace message file. Must contain␣
↪the file extension (such as .rtd).
-e <elffile>:     Specify the name of the executable elf file. Must contain the␣
↪file extension (such as .elf).
-od <objdump>:    Specify the path and name of the objdump executable to use. By␣
↪default uses riscv64-unknown-elf-objdump.
-nofile:          Do not display source file information (enabled by default).
-nodasm:          Do not display disassembled code in output (enabled by default).
-nofunc:          Do not display function information with source information␣
↪(enabled by default).
-notrace:         Do not display trace information in output (enabled by default).
-nobranches:      Do not display conditional branches with taken or not taken␣
↪information (annotated by default).
-cutpath=<path>:  Strip off the specified path when displaying source file name/
↪path.
-path=unix:       Show all file paths using unix-type '/' path separators␣
↪(default).
-path=windows:    Show all file paths using windows-type '\\' path separators.
-path=raw:        Show all file paths in the format stored in the elf file.
-tdasize=<n>:     Specify size in bits of trace data align size. Must be [8, 32,␣
↪64], default is 8.
-tssize=<n>:      Specify size in bits of timestamp counter; used for timestamp␣
```

(continues on next page)

(continued from previous page)

```
↪wrap, default is 13.
-freq=<n>:         Specify the frequency in Hz for the timestamp tics clock,␣
↪default is 0.
-srcid=<n>:        Set srcid for trace decoder, default is 0.
-decodemode=<n>:   Set trace decode mode [7:0] and feature [31:8], default is 0.
                    mode[7:0]=0x00: etrace-v1.0 (default)
                    mode[7:0]=0x01: etrace-v1.1
                    mode[7:0]=0x02: etrace-v1.2 (2bit srcid)
                    mode[7:0]=0x03: etrace-v1.3 (2bit srcid + thaddr)
                    mode[7:0]=0x10: tracebundle-v1.0.0
                    mode[7:0]=0x11: tracebundle-v1.1.0/v1.2.0/v1.3.0
                    mode[7:0]=0x12: tracebundle-v1.4.0
                    feature[08]: has-srcid
                    feature[09]: has-timestamp
                    feature[10]: has-load
                    feature[11]: has-store
                    feature[12]: has-csr
                    feature[13]: has-amo
                    feature[14]: has-zcmp
                    feature[15]: has-value
                    feature[16]: has-addr
-datatrace=<n>:    Specify whether hardware supports data-trace. n is index_width,␣
↪default is 0.
-contextsize=<n>: Specify size in bits of context, default is 0.
-analytics:        Compute and display trace analytics.
-debug:            Display some debug information for the trace to aid in debugging␣
↪the trace decoder.
-v:                Display the version number of the dqr and exit.
-h:                Display this usage information.
```

Common Parameter Descriptions

-tdasize (Trace Data Align Size in Bits)

This parameter should be set according to the actual hardware configuration.

- **TraceBundle module**: tdasize is typically **8**
- **Etrace module**: tdasize can be **8, 32, or 64**, please confirm with your hardware team

Example: If a trace packet contains only 22 bits of data:

- When tdasize=8, the final output will be 24 bits (aligned to 8 bits)
- When tdasize=32, the final output will be 32 bits (aligned to 32 bits)
- When tdasize=64, the final output will be 64 bits (aligned to 64 bits)

-tssize (Timestamp Size in Bits)

This parameter specifies the size of the timestamp counter in bits, used for timestamp wrap detection.

- **etrace-v1.0** and **etrace-v1.1**: tssize is fixed at **13 bits** (default value), can be omitted
- **etrace-v1.2** and **etrace-v1.3**: tssize is fixed at **16 bits**
- **TraceBundle module**: The tssize you can obtain it by reading the *[7:0]* bit field of the TRPATAMETER1 (0x4e04) register in the tracebundle module.

-srcid

This parameter was previously named `-srcid=n`, and both serve the same purpose.

`-srcid=n` is used to specify which srcid ' s trace information to parse. In multi-core trace scenarios, the trace data contains information from multiple cores. Each core is assigned a unique srcid when capturing trace data. Therefore, you need to specify `-srcid=n` during decoding to indicate which core ' s trace data to decode.

-decodemode

This is a **critical** decoding parameter that consists of two fields: **mode field** and **feature field**.

Mode Field [7:0]: Specifies the trace hardware version

- Used to identify the format version of the trace data
- Must match the hardware version that generated the trace

Mode Values:

Note: For all etrace versions, you need to check the hardware code and packet format to determine the specific version.

- `0x00` (etrace-v1.0): Nuclei ' s initial trace version
  - Supports instruction trace only (follows RISC-V trace spec v1.0)
  - 5-bit srcid, 13-bit timestamp

Trace Packet Format (etrace-v1.0):

Field	Size	Description
Timestamp Switch	1 bit	Fixed to 1
halt-id(l2)	2 bits	Lower 2 bits of halt ID
payload-length	5 bits	Length of payload data
halt-id(h3)	3 bits	Upper 3 bits of halt ID
timestamp	13 bits	Timestamp value
payload	0-31 bytes	Variable length payload

- `0x01` (etrace-v1.1): Special version for specific customers (rarely used)
- `0x02` (etrace-v1.2): Same as v1.0 but with different srcid/timestamp size
  - 2-bit srcid, 16-bit timestamp

Trace Packet Format (etrace-v1.2):

Field	Size	Description
Timestamp Switch	1 bit	Fixed to 1
halt-id	2 bits	2 bits of halt ID
payload-length	5 bits	Length of payload data
timestamp	16 bits	Timestamp value
payload	0-31 bytes	Variable length payload

- `0x03` (etrace-v1.3): Same as v1.2 with additional thaddr field
  - 2-bit srcid, 16-bit timestamp
  - Format 3 subformat 1 (support packet) includes thaddr field
- `0x10` (tracebundle-v1.0.0): Supports both instruction and data trace (follows RISC-V trace spec v2.0)

- `0x11` (tracebundle-v1.1.0): Updated tracebundle version
- `0x12` (tracebundle-v1.4.0): Updated tracebundle version with 5bit ecause

Trace Packet Format (tracebundle-v1.x.x):

Field	Size	Description
header	1 byte	Protocol header
src_id	1 byte (optional)	Source ID
Timestamp	n bytes (optional)	Timestamp (controlled by header bit[7])
type	1 bit (optional)	Packet type (0=Instruction Trace, 1=Data Trace), only when data trace supported
Trace Package	Variable length	Trace data payload

Header Byte Format:

Bit Field	Description
bit[4:0]	Trace package byte size (5 bits)
bit[6:5]	Reserved to 0 (2 bits)
bit[7]	Timestamp exist flag (1=exists, 0=not exist)

Note: For tracebundle, you can read the TRTRACEBUNDLEVERSION (0xe00) register to determine the version:

- Register value = 0: tracebundle-v1.0.0
- Register value = 0x10100: tracebundle-v1.1.0
- Register value = 0x10200: tracebundle-v1.2.0
- Register value = 0x10300: tracebundle-v1.3.0
- Register value = 0x10400: tracebundle-v1.4.0

Feature Field [31:8]: Specifies which features were enabled when the trace hardware captured the data

- Indicates what types of trace information are included in the data
- Must match the hardware configuration used during trace capture

Note: Only `has-zcmp` is valid for both etrace and tracebundle. Other features are only valid for tracebundle.

Feature Bits:

- `feature[08]` (has-srcid): Trace packet contains srcid field
  - Typically only present in multi-core trace scenarios (when using Nuclei ' s OpenOCD to capture trace)
- `feature[09]` (has-timestamp): Trace packet contains timestamp field
  - Should match the configuration used during trace capture
- `feature[10]` (has-load): Enable data trace for load operations (tracebundle only)
- `feature[11]` (has-store): Enable data trace for store operations (tracebundle only)
- `feature[12]` (has-csr): Enable data trace for CSR operations (tracebundle only)
- `feature[13]` (has-amo): Enable data trace for AMO operations (tracebundle only)
- `feature[14]` (has-zcmp): Current decoding includes ZCMP instructions

  - **Valid for both etrace and tracebundle**
  - Required because ZCMP instruction encoding overlaps with floating-point instruction encoding
- `feature[15]` (has-value): Include data values in data trace (tracebundle only)
- `feature[16]` (has-addr): Include data addresses in data trace (tracebundle only)

Important: Both mode and feature fields must be set correctly according to your hardware configuration for successful decoding.

-datatrace

This parameter is **only valid for tracebundle**. It indicates whether the trace hardware supports data trace.

If data trace is supported, you need to pass the `index_width` (split transaction ID width) via `-datatrace=n`.

- `-datatrace=0`: Data trace is disabled (default)
- `-datatrace=n` (n > 0): Data trace is enabled, where n specifies the index_width
- The index_width you can obtain it by reading the *[31:16]* bit field of the TRPATAMETER0 (0x4e00) register in the tracebundle module.

-contextsize

This parameter is **valid for tracebundle-v1.1.0 and later versions**.

- Default value is 0
- The contextsize is ASID + VMID. You can obtain it by reading the *[15:8]* bit field of the TRPATAMETER1 (0x4e04) register in the tracebundle module.

-rvtrace

This parameter switches the decoder output to the Nuclei **rvtrace** format, which prints one line per committed instruction in a compact, machine-parseable form. It is intended for users who want a simpler view of the committed instruction stream (without the default file/trace/dasm annotations).

When `-rvtrace` is specified, the per-instruction output line has the following format:

```
<commit_time> hart<hart_id> (<ins_id>) IS <addr> <inst> <priv> [:<inst_text>]
```

Field	Description
`commit_time`	Monotonically increasing commit counter, starting from 1
`hart<hart_id>`	Hart (core) identifier that produced the instruction. Taken from the trace message `srcId` when available, otherwise from `-srcid`
`(<ins_id>)`	Per-hart instruction index, starting from 1
`IS`	Fixed marker indicating an instruction commit
`addr`	Instruction memory address (hex, zero-padded to the trace ' s address print width)
`inst`	Raw instruction encoding (hex, 8 digits)
`priv`	Current privilege mode, propagated from the most recent trace message that carried privilege info
`:<inst_text>`	Optional disassembled instruction text, omitted when not available

Notes:

- `commit_time` and `ins_id` both start at 1 and increment by 1 per committed instruction.

- The privilege field is updated whenever a trace message carries privilege information, even for messages that do not themselves produce an instruction commit line. Therefore the printed `priv` reflects the most recent privilege known at that commit.
- When `-rvtrace` is enabled, the default file/source/trace/dasm output sections are suppressed, and only the per-instruction rvtrace lines are emitted (the trailing `End of Trace File` marker is also suppressed).

Example output:

```
1 hart0 (1) IS a0000004 00004181 3 :c.li        gp,0
2 hart0 (2) IS a0000006 06400213 3 :addi        tp,zero,100
3 hart0 (3) IS a000000a 00004295 3 :c.li        t0,5
4 hart0 (4) IS a000000c a0002337 3 :lui t1,0xa0002
```

6.3.5 Usage Examples

Example 1: ATB2AXI (etrace-v1.x) Instruction Trace Decode

This example demonstrates decoding instruction trace data captured from an ATB2AXI-based system (etrace-v1.x) with 64-bit trace data alignment.

Decode Command:

```
# Windows
.\install\bin\dqr.exe -t tests\atb2axi_tdasize64_demo_eclic.bin -e tests\atb2axi_
↪tdasize64_demo_eclic.elf -tdasize=64

# Linux
./install/bin/dqr -t tests/atb2axi_tdasize64_demo_eclic.bin -e tests/atb2axi_
↪tdasize64_demo_eclic.elf -tdasize=64
```

Parameters Explained:

- `-tdasize=64`: Specify 64-bit trace data alignment size (required for this ATB2AXI configuration)

Output Sample:

```
Trace: Msg # Num: 1, SrcId: 0, Tics: 634
           # Raw: 85 02 7a 73 8e 1d 00 80
           # NxtAddr: 0x80001d8e, FormatCode: sync(3) SubFormatCode: start(0)
           # branch(0x1) privilege(0x3) context(0x0) address(0x80001d8e)

File: C:/software/nucleiIDE/workspace/demo_eclic/application/demo_eclic.c:97
<_text+1b0e>
    80001d8e:         4785        c.li      a5,1

Trace: Msg # Num: 2, SrcId: 0, Tics: 887
           # Raw: 82 03 77 81 06 00 00 00
           # NxtAddr: 0x80001d90, FormatCode: diff-delta(1)
           # branches(0x1f) branch_map(0xd) address(0x0) notify(0x0) updiscon(0x0)␣
↪irreport(0x0) irdepth(0x0)
    80001d90:         fef40723    sb        a5,-18(s0)

File: C:/software/nucleiIDE/workspace/demo_eclic/application/demo_eclic.c:101
    80001d94:         91bff0ef    jal       ra,800016ae <setup_timer>

File: C:/software/nucleiIDE/workspace/demo_eclic/application/demo_eclic.c:31
<_text+142e>
    800016ae:         1141        c.addi    sp,-16
    800016b0:         c606        c.swsp    ra,12(sp)
    800016b2:         c422        c.swsp    s0,8(sp)
    800016b4:         0800        c.addi4spn        s0,sp,16
```

Example 2: TraceBundle Instruction Trace Decode

This example demonstrates decoding instruction-only trace data captured from a TraceBundle-based system with CPU timestamp (16-bit).

Decode Command:

```
# Windows
.\install\bin\dqr.exe -t tests\tracebundle_inst_cpu16_helloworld.bin -e tests\
↪tracebundle_inst_cpu16_helloworld.elf -datatrace=5 -decodemode=0x310 -tssize=16

# Linux
./install/bin/dqr -t tests/tracebundle_inst_cpu16_helloworld.bin -e tests/
↪tracebundle_inst_cpu16_helloworld.elf -datatrace=5 -decodemode=0x310 -tssize=16
```

Parameters Explained:

- `-datatrace=5`: Specify index_width for data trace (required even for instruction-only traces in tracebundle)
- `-decodemode=0x310`: tracebundle-v1.0.0 (0x10) with has-timestamp (0x100) and has-srcid (0x200) features
- `-tssize=16`: 16-bit timestamp counter size

Output Sample:

```
Trace: Msg # Num: 1, SrcId: 0, Tics: 0
           # Raw: 86 00 00 00 e6 08 02 00 00 ff
           # NxtAddr: 0x80000104, FormatCode: sync(3) SubFormatCode: start(0)
           # branch(0x1) privilege(0x3) context(0x0) address(0x80000104)

File: D:/Git/nuclei-sdk/SoC/evalsoc/Common/Source/GCC/startup_evalsoc.S:169
<_start+4>
    80000104:         f1402573    csrrs     a0,mhartid,zero

Trace: Msg # Num: 2, SrcId: 0, Tics: 0
           # Raw: 82 00 00 00 02 09
           # NxtAddr: 0x80000108, FormatCode: diff-delta(1)
           # branches(0x1f) branch_map(0x9) address(0x0) notify(0x0) updiscon(0x0)␣
↪irreport(0x0) irdepth(0x0)

File: D:/Git/nuclei-sdk/SoC/evalsoc/Common/Source/GCC/startup_evalsoc.S:170
    80000108:         0ff57513    andi      a0,a0,255
```

Example 3: TraceBundle Instruction + Data Trace Decode

This example demonstrates decoding both instruction and data trace captured from a TraceBundle-based system. Data trace includes load/store operations with addresses and values.

Decode Command:

```
# Windows
.\install\bin\dqr.exe -t tests\tracebundle_inst_data_cpu16_dhrystone.bin -e tests\
↪tracebundle_inst_data_cpu16_dhrystone.elf -datatrace=4 -decodemode=0x3f10 -
↪tssize=16

# Linux
./install/bin/dqr -t tests/tracebundle_inst_data_cpu16_dhrystone.bin -e tests/
↪tracebundle_inst_data_cpu16_dhrystone.elf -datatrace=4 -decodemode=0x3f10 -
↪tssize=16
```

Parameters Explained:

- `-datatrace=4`: Enable data trace with index_width=4 (split transaction ID width)

- `-decodemode=0x3f10`: tracebundle-v1.0.0 (0x10) with has-load(0x100), has-store(0x200), has-value(0x4000), has-addr(0x8000), has-timestamp(0x2000), and has-srcid(0x400) features
- `-tssize=16`: 16-bit timestamp counter size

Output Sample:

```
Trace: Msg # Num: 3, SrcId: 0, Tics: 28822
           # Raw: 84 00 96 70 1e 80 21 20
           # NxtAddr: 0x800031a0, FormatCode: sync(3) SubFormatCode: support(3)
           # ienable(0x0) encoder_mode(0x0) qual_status(0x0) ioptions(0xc0)␣
↪denable(0x1) dloss(0x0) doptions(0x40)

Trace: Msg # Num: 4, SrcId: 0, Tics: 28824
           # Raw: 86 00 98 70 e6 40 63 00 00 ff
           # NxtAddr: 0x800031a0, FormatCode: sync(3) SubFormatCode: start(0)
           # branch(0x1) privilege(0x3) context(0x0) address(0x800031a0)
<_text+2f60>
    800031a0:         85aa        c.mv      a1,a0

Trace: Msg # Num: 5, SrcId: 0, Tics: 28853
           # Raw: 83 00 b5 70 1a 32 05
           # NxtAddr: 0x800031a2, FormatCode: diff-delta(1)
           # branches(0x3) branch_map(0x2) address(0xa6) notify(0x0) updiscon(0x0)␣
↪irreport(0x0) irdepth(0x0)
    800031a2:         80c1a503    lw        a0,-2036(gp) # 900001d4 <_impure_ptr>

Trace: Msg # Num: 6, SrcId: 0, Tics: 94360
           # Raw: 86 00 98 70 21 a9 03 00 20 ff
           # Data Trace Format: split load address,aligned(0)
           # size(0x2) index(0x4) diff(0x0) daddress(0x900001d4)
    800031a6:         a8b1        c.j       80003202 <_malloc_r>
<_text+2fc2>
    80003202:         1101        c.addi    sp,-32
    80003204:         ca26        c.swsp    s1,20(sp)

Trace: Msg # Num: 7, SrcId: 0, Tics: 94363
           # Raw: 86 00 9b 70 29 09 00 00 00 f9
           # Data Trace Format: split load data(4)
           # size(0x2) index(0x4) resp(0x2) data(0x90000000)

Trace: Msg # Num: 8, SrcId: 0, Tics: 94365
           # Raw: 87 00 9d 70 a5 4a c0 04 80 fd 00
           # Data Trace Format: store,aligned address(2)
           # size(0x2) diff(0x2) data_len(0x2) data(0x13012) daddress(0x9000ff54)
    80003206:         00358493    addi      s1,a1,3
    8000320a:         98f1        c.andi    s1,-4
    8000320c:         c84a        c.swsp    s2,16(sp)

Trace: Msg # Num: 9, SrcId: 0, Tics: 94367
           # Raw: 83 00 9f 70 a5 04 fc
           # Data Trace Format: store,aligned address(2)
           # size(0x2) diff(0x2) data_len(0x0) data(0x1) daddress(0x9000ff50)
    8000320e:         ce06        c.swsp    ra,28(sp)
```

Example 4: TraceBundle with ZCMP Extension Decode

This example demonstrates decoding trace data that includes ZCMP (RISC-V Code Size Reduction) instructions. ZCMP instructions require special handling because their encoding overlaps with floating-point instruction encoding.

Decode Command:

```
# Windows
.\install\bin\dqr.exe -t tests\rv32im_zc.bin -e tests\rv32im_zc.elf -datatrace=4 -
↪decodemode=0x7f10 -tssize=16

# Linux
./install/bin/dqr -t tests/rv32im_zc.bin -e tests/rv32im_zc.elf -datatrace=4 -
↪decodemode=0x7f10 -tssize=16
```

Parameters Explained:

- `-datatrace=4`: Enable data trace with index_width=4
- `-decodemode=0x7f10`: tracebundle-v1.0.0 (0x10) with has-load(0x100), has-store(0x200), has-csr(0x1000), has-amo(0x2000), has-zcmp(0x4000), has-value(0x8000), has-addr(0x10000), has-timestamp(0x400), and has-srcid(0x200) features
- `-tssize=16`: 16-bit timestamp counter size
- **Note**: The `has-zcmp` feature (0x4000) is critical for correct decoding when ZCMP instructions are present in the code

Output Sample:

```
Trace: Msg # Num: 1, SrcId: 0, Tics: 29356
           # Raw: 84 00 ac 72 3e 80 21 20
           # NxtAddr: 0x00000000, FormatCode: sync(3) SubFormatCode: support(3)
           # ienable(0x1) encoder_mode(0x0) qual_status(0x0) ioptions(0xc0)␣
↪denable(0x1) dloss(0x0) doptions(0x40)

Trace: Msg # Num: 2, SrcId: 0, Tics: 29367
           # Raw: 86 00 b7 72 e6 08 00 00 40 ff
           # NxtAddr: 0xa0000004, FormatCode: sync(3) SubFormatCode: start(0)
           # branch(0x1) privilege(0x3) context(0x0) address(0xa0000004)
<_start+4>
    a0000004:         4181        c.li      gp,0

Trace: Msg # Num: 3, SrcId: 0, Tics: 30846
           # Raw: 82 00 7e 78 14 04
           # NxtAddr: 0xa0000006, FormatCode: addr-only(2)
           # address(0x82) notify(0x0) updiscon(0x0) irreport(0x0) irdepth(0x0)
    a0000006:         06400213    addi      tp,zero,100
    a000000a:         4295        c.li      t0,5
    a000000c:         a0002337    lui       t1,0xa0002
    a0000010:         6385        c.lui     t2,0x1
```

Output Sample - ZCMP Operations:

```
Trace: Msg # Num: 11, SrcId: 0, Tics: 95139
           # Raw: 82 00 a3 73 61 04
           # Data Trace Format: split load address,aligned(0)
           # size(0x2) index(0x1) diff(0x1) daddress(0xa0002000)
    a0000056:         0001        c.addi    zero,0
    a0000058:         4085        c.li      ra,1
    a000005a:         b846        cm.push   {ra},-32

Trace: Msg # Num: 13, srcId: 0, Tics: 95200
           # Raw: 84 00 e0 73 a5 04 fc ef
```

(continues on next page)

(continued from previous page)

```
          # Data Trace Format: store,aligned address(2)
          # size(0x2) diff(0x2) data_len(0x0) data(0x1) daddress(0xa0000ffc)
    a000005c:         0001        c.addi    zero,0
    a000005e:         ba46        cm.pop    {ra},32

Trace: Msg # Num: 14, SrcId: 0, Tics: 95282
          # Raw: 82 00 32 74 e1 04
          # Data Trace Format: split load address,aligned(0)
          # size(0x2) index(0x3) diff(0x1) daddress(0xa0000ffc)
    a0000060:         b856        cm.push   {ra,s0},-32

Trace: Msg # Num: 17, SrcId: 0, Tics: 95330
          # Raw: 83 00 62 74 a5 04 fc
          # Data Trace Format: store,aligned address(2)
          # size(0x2) diff(0x2) data_len(0x0) data(0x1) daddress(0xa0000ff8)
    a0000062:         0001        c.addi    zero,0
    a0000064:         ba56        cm.pop    {ra,s0},32

Trace: Msg # Num: 19, SrcId: 0, Tics: 95418
          # Raw: 82 00 ba 74 61 fc
          # Data Trace Format: split load address,aligned(0)
          # size(0x2) index(0x1) diff(0x1) daddress(0xa0000ff8)
    a0000066:         b866        cm.push   {ra,s0-s1},-32

Trace: Msg # Num: 24, SrcId: 0, Tics: 95489
          # Raw: 83 00 01 75 a5 04 fc
          # Data Trace Format: store,aligned address(2)
          # size(0x2) diff(0x2) data_len(0x0) data(0x1) daddress(0xa0000ff4)
    a0000068:         0001        c.addi    zero,0
    a000006a:         ba66        cm.pop    {ra,s0-s1},32
```

Key Observations:

ZCMP Instructions in the Output:

- `cm.push {ra},-32` (0xb846): Push return address register onto stack, allocate 32 bytes
- `cm.pop {ra},32` (0xba46): Pop return address register from stack, deallocate 32 bytes
- `cm.push {ra,s0},-32` (0xb856): Push ra and s0 registers, allocate 32 bytes
- `cm.pop {ra,s0},32` (0xba56): Pop ra and s0 registers, deallocate 32 bytes
- `cm.push {ra,s0-s1},-32` (0xb866): Push ra, s0, and s1 registers, allocate 32 bytes
- `cm.pop {ra,s0-s1},32` (0xba66): Pop ra, s0, and s1 registers, deallocate 32 bytes

ZCMP Trace Characteristics:

- ZCMP instructions are displayed with their mnemonics (cm.push, cm.pop) in the disassembly
- The decoder correctly identifies these as code size reduction instructions rather than floating-point instructions
- Stack operations (push/pop) are tracked with corresponding data trace entries showing memory writes/reads
- The `has-zcmp` feature ensures proper disassembly of these instructions

Key Points about ZCMP:

- ZCMP (Code Size Reduction) instructions share encoding space with floating-point instructions
- Without `has-zcmp` feature enabled, the decoder may misinterpret ZCMP instructions as floating-point instructions
- The `has-zcmp` feature bit must be set in the decodemode when the target code contains ZCMP instructions

Example 5: TraceBundle with CSR and AMO Operations Decode

This example demonstrates decoding trace data that includes CSR (Control and Status Register) operations and AMO (Atomic Memory Operation) instructions.

Decode Command:

```
# Windows
.\install\bin\dqr.exe -t tests\rv32imc_csr_amo.bin -e tests\rv32imc_csr_amo.elf -
↪datatrace=4 -decodemode=0x3f10 -tssize=16

# Linux
./install/bin/dqr -t tests/rv32imc_csr_amo.bin -e tests/rv32imc_csr_amo.elf -
↪datatrace=4 -decodemode=0x3f10 -tssize=16
```

Parameters Explained:

- `-datatrace=4`: Enable data trace with index_width=4
- `-decodemode=0x3f10`: tracebundle-v1.0.0 (0x10) with has-load(0x100), has-store(0x200), has-csr(0x1000), has-amo(0x2000), has-value(0x4000), has-addr(0x8000), has-timestamp(0x2000), and has-srcid(0x400) features
- `-tssize=16`: 16-bit timestamp counter size

Output Sample - CSR Operations:

```
Trace: Msg # Num: 4, SrcId: 0, Tics: 69454
           # Raw: 8b 00 4e 0f 1b 1f 44 40 00 31 03 00 00 00 04
           # Data Trace Format: csr(5) subtype(1)
           # diff(0x0) data_len(0x3) data(0x40101107) addr_msbs(0xc) op_len(0x3)␣
↪operand(0x0) addr_lsbs(0x1) csr_addr(0x301)
    a0000032:         b0002573    csrrs     a0,mcycle,zero

Trace: Msg # Num: 5, SrcId: 0, Tics: 69469
           # Raw: 87 00 5d 0f 1b e7 87 12 29 b1 03
           # Data Trace Format: csr(5) subtype(1)
           # diff(0x0) data_len(0x3) data(0x4a44a1f9) addr_msbs(0x2c) op_len(0x3)␣
↪operand(0x0) addr_lsbs(0x0) csr_addr(0xb00)
    a0000036:         0001        c.addi    zero,0
    a0000038:         34031073    csrrw     zero,mscratch,t1
```

Output Sample - AMO Operations:

```
Trace: Msg # Num: 16, SrcId: 0, Tics: 69686
           # Raw: 8a 00 36 10 0d 1d 00 04 00 74 00 10 00 50
           # Data Trace Format: unified atomic or split atomic address(6)␣
↪subtype(0)
           # size(0x2) diff(0x2) op_len(0x3) operand(0xa0002000) data_len(0x3)␣
↪data(0xa0002000) daddress(0xa0002000)
           # amo_data(0xa0002000)
    a000006c:         0001        c.addi    zero,0
    a000006e:         000323af    amoadd.w  t2,zero,(t1)

Trace: Msg # Num: 17, SrcId: 0, Tics: 69718
           # Raw: 8a 00 56 10 1d 1d 00 00 00 60 00 10 00 50
           # Data Trace Format: unified atomic or split atomic address(6)␣
↪subtype(1)
           # size(0x2) diff(0x2) op_len(0x3) operand(0x0) data_len(0x3)␣
↪data(0xa0002000) daddress(0xa0002000)
           # amo_data(0xa0002000)
    a0000072:         0001        c.addi    zero,0
    a0000074:         606323af    amoand.w  t2,t1,(t1)
```

Key Observations:

CSR Trace Format:

- **Data Trace Format**: `csr(5)` indicates a CSR operation trace packet
- **subtype**: Different subtypes for different CSR operations (read, write, set, clear)
- **csr_addr**: The address of the CSR being accessed (e.g., 0x301 for misa, 0xb00 for mcycle)
- **data**: The value read from or written to the CSR
- **operand**: The source register value for CSR write operations

AMO Trace Format:

- **Data Trace Format**: `unified atomic or split atomic address(6)` indicates an AMO operation
- **subtype**: Different subtypes for different AMO operations:
  - subtype(0): AMOADD
  - subtype(1): AMOAND
  - subtype(2): AMOXOR
  - subtype(4): AMAX
  - subtype(5): AMAXU/AMOMINU
  - subtype(6): AMOMIN
- **daddress**: The memory address for the atomic operation
- **data**: The data value involved in the atomic operation
- **operand**: The operand value for the atomic operation
- **amo_data**: The result of the atomic operation

Supported AMO Instructions:

- `amoswap.w`: Atomic swap
- `amoadd.w`: Atomic add
- `amoand.w`: Atomic AND
- `amoxor.w`: Atomic XOR
- `amomax.w`/`amomaxu.w`: Atomic maximum (signed/unsigned)
- `amomin.w`/`amominu.w`: Atomic minimum (signed/unsigned)
- `lr.w`/`sc.w`: Load-reserved / Store-conditional

Trace Packet Output Format Description

```
Trace: Msg # Num: 1, SrcId: 0, Tics: 634
```

This line contains three fields: Num, SrcId, and Tics. Num is a unique identifier assigned by the decoder software; SrcId is the source ID parsed from the trace packet; Tics/Time is the complete timestamp calculated from the timestamp information within the trace packet.

```
# Raw: 85 02 7a 73 8e 1d 00 80
```

This line shows the raw data of the trace packet, with each byte represented in hexadecimal.

```
# NxtAddr: 0x80001d8e, FormatCode: sync(3) SubFormatCode: start(0)
```

NxtAddr is the next instruction address predicted by the decoder, calculated based on the trace packet information and the current instruction. FormatCode indicates the specific format of the trace packet.

```
# branch(0x1) privilege(0x3) context(0x0) address(0x80001d8e)
```

These are the individual fields decoded from the raw trace packet data.

- To learn about the packet formats and features supported by Nuclei Trace hardware, please refer to the <Nuclei_Trace_Bundle_Specification.pdf> document.
- For more detailed information about trace formats, please refer to the <RISC-V Etrace v1.0> and <RISC-V Etrace v2.0> specifications. Please visit https://github.com/riscv-non-isa/riscv-trace-spec/releases/tag/v2.0-20250616 for more details about etrace.

6.3.6 Common Problem Troubleshooting and Solutions

- **Dependency Tool**: TraceDecoder requires the RISC-V version of the `objdump` tool to parse the ELF file ' s symbol table and perform disassembly
- **File Path**: If files specified with `-t` and `-e` are not in the current directory, full paths need to be provided
- **Decode Mode**: Ensure the selected decode mode matches the actual format of the trace data, otherwise decoding may fail
- **Memory Usage**: When processing large trace files, ensure the system has sufficient memory resources

Default (etrace-v1.0) Decoding srcid Mismatch Issue

After adding the -debug parameter, the error scenario is as follows: nm.srcId shows multiple abnormal values. In this case, try adding `-decodemode=0x2` or `-decodemode=0x3` for decoding

```
Trace: Msg # Num: 1, SrcId: 0, tics: 82901

Raw: 85 03 d5 01 32 20 01 f2
NxtAddr: 0x00000000, FormatCode: diff-delta(1)
branches(0x1f) branch_map(0x64024064) address(0xffffffff) notify(0x1)␣
↪updisc0n(0x1) irreport(0x1) irdepth(0x1)
state:0
readNewTraceMessage:true
readDataTrace:0
nm.srcId 0x20, srcid 0x0, nm.srcId != srcid, ignored
state:0
readNewTraceMessage:true
readDataTrace:0
nm.srcId 0x8, srcid 0x0, nm.srcId != srcid, ignored
state:0
readNewTraceMessage:true
readDataTrace:0
nm.srcId 0x16, srcid 0x0, nm.srcId != srcid, ignored
state:0
readNewTraceMessage:true
readDataTrace:0
nm.srcId 0x28, srcid 0x0, nm.srcId != srcid, ignored
```

Decoding Exception Scenarios for tracebundle Versions Higher than v1.1.0

When the tracebundle version is higher than v1.1.0, regardless of whether the software has enabled timestamp or not, the tssize parameter must be passed during decoding. If the CPU is configured with MMU, the contextsize parameter must also be passed, otherwise the following decoding phenomena may occur

```
trace_decoder\bin>dqr -t core0_trace_02.bin -e yunduu_fpga.elf -decodemode=0x111
Trace: Msg # Num: 1, SrcId: 0
           # Raw: 0d 00 33 00 00 00 00 00 00 00 00 00 42 73 01
           # NxtAddr: 0x1000000000000000, FormatCode: sync(3) SubFormatCode:␣
↪start(0)
           # branch(0x1) privilege(0x1) time(0x0) context(0x0) address(0x0)
Error: ElfReader::getInstructionByAddress(): Address 0x1000000000000000 not found␣
↪in any ELF code section. The ELF file may not contain the code for this trace␣
↪address.
Error: status:2 getInstructionByAddress failed - addr:0x1000000000000000
Error (5) terminated trace decode
```

6.3.7 Changelog

Version v1.0.0

This version is developed based on v0.9.0, mainly optimizing code structure, removing redundant code and command line parameters, adding new feature support, and fixing some bugs.

- New features added:
  - Added support for multiple versions of trace hardware (divided into atb2axi and tracebundle categories)
  - Added support for data-trace decoding
  - Added CI automated testing process and new test sets
  - Added TraceDecoder decoding documentation
- Bug fixes:
  - Fixed the issue of discontinuous Msg Num during multi-core decoding
  - Fixed the issue where adjacent trace packets with the same timestamp were incorrectly identified as timestamp overflow
  - Fixed the libc version incompatibility issue on Linux by switching to static linking
  - Fix objdump locale issue on Linux
- Command line parameter changes:
  - Removed command line parameters: `[-src]` `[-nosrc]` `[-file]` `[-func]` `[-dasm]` `[-trace]` `[-branches]` `[--strip=path]` `[-addrsize=n]` `[-addrsize=n+]` `[-32]` `[-64]` `[-32+]` `[-addrsep]` `[-noaddrsep]` `[-analytics=n]` `[-noanalytics]` `[-callreturen]` `[-nocallreturn]` `[-msglevel=n]` `[-r addr]` `[-nodebug]`
  - Merged `[-pathunix]` `[-pathwindows]` `[-pathraw]` into `[-path=unix|windows|raw]`
  - Renamed `[-hartid=n]` to `[-srcid=n]`
  - Added `[-datatrace=n]`
  - Extended `[-decodemode=n]`

Command parameter table:

```
Usage: dqr -t tracefile -e elffile [-od objdump]
           [-nofile] [-nofunc] [-nodasm] [-notrace] [-nobranches]
           [-cutpath=path] [-path=unix|windows|raw]
           [-tdasize=n] [-tssize=n] [-freq=n] [-srcid=n] [-decodemode=n] [-
↪datatrace=n]
           [-analytics] [-debug] [-v] [-h]
```

Version v0.9.0

This is the first release version. Since TraceDecoder does not have a separate release plan, it was released together with Nuclei Studio 2025.10.

Command parameter table:

```
Usage: dqr -t tracefile -e elffile [-od objdump] -basename name
           [-src] [-nosrc] [-file] [-nofile] [-func] [-nofunc] [-dasm] [-nodasm]
           [-trace] [-notrace] [-pathunix] [-pathwindows] [-pathraw] [--strip=path]
           [-addrsize=n] [-addrsize=n+] [-32] [-64] [-32+] [-addrsep] [-noaddrsep]␣
↪[-analytics | -analytics=n]
           [-noanalytics] [-freq nn] [-tssize=n] [-callreturn] [-nocallreturn] [-
↪branches] [-nobranches] [-msglevel=n]
           [-cutpath=<base path>] [-r addr] [-debug] [-nodebug] [-tdasize=n] [-
↪hartid=n] [-decodemode=n] [-v] [-h]
```

CHAPTER

SEVEN

NUCLEI QEMU

7.1 About Nuclei QEMU

Nuclei QEMU is developed based on QEMU version 9.0 and supports the machine features of Nuclei EvalSoC. In addition to the extensions supported by upstream QEMU, it supports the following unratified and custom extensions implemented by Nuclei.

- **Xxldsp**: P 0.5.4 draft extension and Nuclei custom N1/N2/N3 extensions
- **Xxlcz**: Nuclei custom code size reduction extension
- **Nice & Vnice**: Nuclei custom Nice & Vnice extensions
- **Xxlvqmacc**: Nuclei custom vpu extension
- **Zilsd & Zclsd**: riscv-zilsd Release 1.0
- **Xxlfbf & Xxlvfbf**: Nuclei custom scalar & vector BF16 extensions

Note: Before using Nuclei QEMU, you can check the current version on which Nuclei QEMU is based and the commit id through the `--version` option of Nuclei QEMU, for example:

```
$ qemu-system-riscv32 --version
QEMU emulator version 9.0.4 (v9.0.4-302-gd2e9c7c)
Copyright (c) 2003-2024 Fabrice Bellard and the QEMU Project developers
```

The above shows that Nuclei QEMU is currently developed based on the upstream version 9.0.4, and commit id is d2e9c7c[65].

The source code is available in the open-source Nuclei QEMU GitHub repository[66]. For more information about QEMU, see the official QEMU documentation[67].

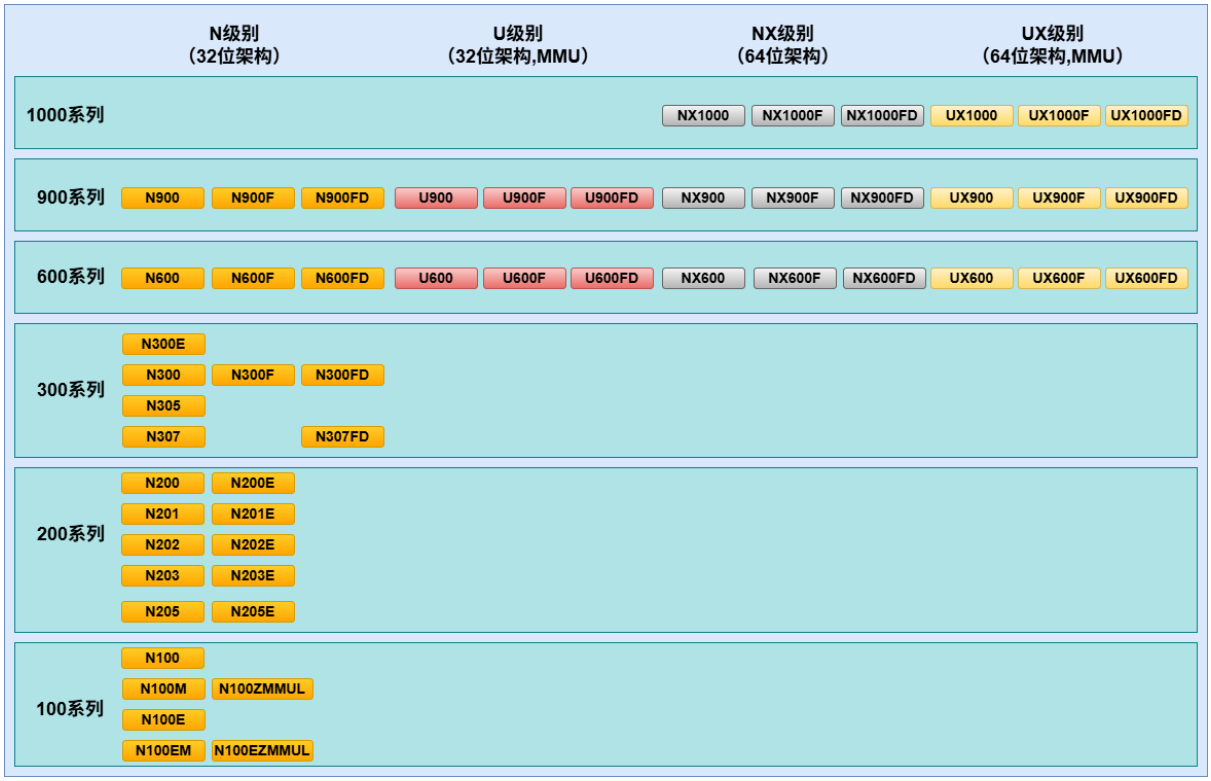
7.2 Design and Architecture

Previously, Nuclei QEMU supports two machine types: `nuclei_evalsoc` and `nuclei_demosoc`, but now, `nuclei_demosoc` has been abandoned on Nuclei QEMU 2025.02 and later versions.

[65] https://github.com/riscv-mcu/qemu/tree/d2e9c7c
[66] https://github.com/riscv-mcu/qemu/tree/nuclei/9.0
[67] https://www.qemu.org/documentation/

7.2.1 Nuclei CPU Types Supported on QEMU

The following image shows the relevant information of the Nuclei CPU types supported by Nuclei QEMU.



7.2.2 Address Allocation of EvalSoC on QEMU

The following table shows the default EvalSoC address map. Fixed devices can be relocated or disabled with `soc-cfg`. IREGION devices use fixed offsets from `iregion.base`, whose default value is `0x18000000`.

Table 7.1: Default EvalSoC address map

Type	Component	Base address	Size	Description
Memory	XIP/NOR flash	`0x20000000`	`0x04000`	Boot flash and XIP memory.
Memory	QSPI0 XIP window	`0x24000000`	`0x04000`	Direct XIP window provided by QSPI0.
Memory	DDR	`0x80000000`	`0x04000`	General-purpose system memory.
Memory	ILM	`0x80000000`	`0x00800`	Instruction local memory.
Memory	DLM	`0x90000000`	`0x00800`	Data local memory.
Memory	SRAM	`0xA0000000`	`0x20000`	On-chip SRAM address window.
Fixed peripheral	TEST	`0x00100000`	`0x00010`	EvalSoC test finisher and reset device.
Fixed peripheral	XEC0	`0x10002000`	`0x00001`	Nuclei Ethernet controller.
Fixed peripheral	MISC	`0x10012000`	`0x00001`	Counters, NMI/event generation and miscellaneous controls.
Fixed peripheral	UART0	`0x10013000`	`0x00001`	Legacy UART or USART serial controller.
Fixed peripheral	QSPI0	`0x10014000`	`0x00001`	SPI/QSPI controller with an XIP window.
Fixed peripheral	uDMA	`0x10015000`	`0x00001`	Nuclei uDMA controller.
Fixed peripheral	QSPI1	`0x10024000`	`0x00001`	Reserved configuration entry; the current board model realizes QSPI0 and QSPI2.
Fixed peripheral	QSPI2	`0x10034000`	`0x00001`	SPI/QSPI controller.
Fixed peripheral	APLIC_M	`0x10040000`	`0x00004`	Machine-level advanced interrupt controller.
Fixed peripheral	APLIC_S	`0x10044000`	`0x00004`	Supervisor-level advanced interrupt controller.
Fixed peripheral	IMSIC_M	`0x4F000000`	`0x00001` per hart (default)	Machine-level per-hart interrupt files; size is configurable.
Fixed peripheral	IMSIC_S	`0x4F200000`	Variable	Supervisor-level per-hart interrupt files; each slot is `0x1000` and the slot count rounds `aia-guests + 1` up to a power of two.
IREGION	IINFO	`iregion` + `0x00000`	`0x00010`	CPU capability and implementation information registers.
IREGION	DEBUG	`iregion` + `0x10000`	`0x00001`	Debug address space.
IREGION	ECLIC	`iregion` + `0x20000`	`0x00010`	Nuclei ECLIC unit.
IREGION	TIMER	`iregion` + `0x30000`	`0x00010`	Nuclei SysTimer/CLINT-compatible window.
IREGION	SMP/CC	`iregion` + `0x40000`	`0x00001`	SMP and cluster-cache control unit.
IREGION	CIDU	`iregion` + `0x50000`	`0x00010`	Core interrupt distribution unit.
IREGION	PLIC	`iregion` + `0x4000000`	`0x04000`	Platform-level interrupt controller.
Information address	PPI	`0xB0000000`	N/A	Used to view CPU info, not implemented.
Information address	FIO	`0xC0000000`	N/A	Used to view CPU info, not implemented.

Note: The MROM reset trampoline at `0x1000` is no longer instantiated. EvalSoC now resets each hart directly to the selected `download=` start address.

7.2.3 Nuclei CPU and SoC Features Supported on QEMU

The following table summarizes the current implementation status.

Table 7.2: Nuclei feature status

Feature	Status
TIMER	Supported; see *TIMER Support* (page 838).
PLIC	Supported; see *PLIC Support* (page 839).
AIA	Supported; see *AIA Support* (page 839).
ECLIC v1/v2	Supported; see *ECLIC Support* (page 839).
CIDU	Supported; see *CIDU Support* (page 839).
PMP	Supported; see *PMP Support* (page 839).
TEE	Supported; see *TEE Support* (page 839).
ECC	ECC-related CSRs are implemented; see *ECC Support* (page 839).
CCM	CCM-related CSRs are implemented; see *CCM Support* (page 839).
Nuclei stack check	Supported; see *STACK-CHECK Support* (page 840).
WFI / WFE	Supported; see *WFI/WFE Support* (page 840).
SPMP	Not supported.
SMP / CLUSTER CACHE	Supported; see *SMP & CLUSTER CACHE Support* (page 840).
UART / USART	Supported; see *UART Support* (page 840).
MISC	Supported; see *MISC Support* (page 840).
QSPI	Supported; see *QSPI Support* (page 840).
uDMA	Supported; see *uDMA Support* (page 840).
XEC Ethernet	Supported; see *XEC Ethernet Support* (page 840).
IREGION	Supported; see *IREGION Support* (page 841).
TEST FINISHER	Supported; see *TEST Support* (page 841).

TIMER Support

The Nuclei SysTimer implements a shared time base, per-hart timer comparison and delivery, timer pause/control behavior, and multi-core ECLIC operation. EvalSoC also provides a CLINT-compatible timer model for conventional RISC-V software and supports the `Sstc` supervisor timer extension.

PLIC Support

The PLIC model supports platform-level interrupt prioritization and routing for machine-mode and supervisor-mode interrupt contexts.

AIA Support

EvalSoC supports the RISC-V Advanced Interrupt Architecture in APLIC-only and APLIC-with-IMSIC forms, covering wired interrupt routing and message-signaled interrupt delivery. AIA operation is supported with both TCG and KVM.

ECLIC Support

ECLIC supports single-core and multi-core operation in both M-mode and S-mode. The ECLIC v1 model provides the basic interrupt configuration, arbitration, nesting and trigger behavior. ECLIC v2 additionally supports automatic context save/restore and shadow general-purpose register groups, together with the associated nested-trap state handling.

CIDU Support

The CIDU distributes external interrupts among per-hart ECLIC instances. It also implements inter-core interrupts, multiple interrupt-distribution modes, hardware semaphores, and supervisor-mode access control for multicore software.

PMP Support

QEMU provides the standard RISC-V PMP address-matching and memory-access permission behavior used by Nuclei CPUs.

TEE Support

QEMU ' s Nuclei TEE-related support includes the sMPU v0.9.0 memory-protection model and S-mode ECLIC operation. These capabilities provide supervisor/user memory isolation, protection-fault reporting, and supervisor-level interrupt handling.

ECC Support

QEMU implements the Nuclei ECC-related CSR interfaces used by software. Hardware ECC generation, error detection and correction, and error injection are not functionally modeled.

CCM Support

QEMU implements the Nuclei CCM-related CSR interfaces used by software. The corresponding cache-control commands and cache hardware behavior are not functionally modeled.

WFI/WFE Support

Standard RISC-V `WFI` behavior is supported. Nuclei `WFE` support currently covers its architectural CSR interface, but does not model the complete hardware event and low-power behavior.

STACK-CHECK Support

The Nuclei stack-check model provides stack-bound tracking and detects stack overflow and underflow conditions, reporting them through the corresponding stack exception flow.

SMP & CLUSTER CACHE Support

The EvalSoC SMP/cluster block models the multicore control interface, the Snoop Control Unit (SCU) control surface, and cluster-local memory (CLM) instantiation. It does not model cache contents, coherence traffic, cache timing, or the functional behavior of the hardware cache hierarchy.

UART Support

EvalSoC supports the legacy Nuclei UART model and the newer USART v3.1.0 model. UART0 operates in the legacy UART mode by default. Compared with the legacy version, USART v3.1.0 adds transmit and receive FIFOs, programmable FIFO thresholds, receive-idle detection, baud timing, and expanded interrupt status handling. The current EvalSoC model instantiates UART0 only; UART1 is no longer present.

MISC Support

The MISC block provides programmable interrupt counters, NMI and event generation, latency/debug controls, SDIO control, and version reporting. Event and SDIO-control outputs are represented by the device, but are not connected to other devices on the current EvalSoC board.

QSPI Support

QSPI0 and QSPI2 support SPI/QSPI transfers and interrupt generation and operate in the legacy SPI v1.1.0 mode by default. QSPI0 additionally provides direct execute-in-place (XIP) access. Compared with SPI v1.1.0, SPI v1.2.8 adds expanded transfer control, status/error reporting and interrupt behavior, including QSPI0 uDMA transfers. QSPI1 is not instantiated by the current EvalSoC model.

uDMA Support

The uDMA v2.5.0 model supports memory-to-memory and peripheral transfers, single, continuous and repeat transfer modes, linked-list descriptors, and channel completion and error reporting. It is integrated with QSPI0 for transmit and receive data movement when QSPI0 operates in SPI v1.2.8 mode.

XEC Ethernet Support

XEC0 implements the Nuclei Ethernet MAC, PHY/MDIO access, descriptor-based transmit and receive DMA, packet filtering, status reporting, and interrupts. It integrates with standard QEMU network backends.

IREGION Support

The IINFO block primarily maps the selected CPU ' s enabled ISA extensions into the corresponding IREGION capability information. Other IINFO registers are provided as register-interface examples. Because QEMU does not model the associated hardware structures, most of these registers expose only predefined values or simple storage and do not provide actual hardware-derived mappings or functional behavior.

TEST Support

The EvalSoC TEST finisher provides guest-to-host pass/fail reporting and a full-system reset request. Writing `0x3333` or `0x5555` to the TEST address (`0x100000` by default) exits QEMU with a failure or pass result, respectively. Writing `0x7777` triggers a full-system reset, reinitializes all devices, and starts the program again.

7.3 Nuclei QEMU Command-line Options

This section covers EvalSoC-specific settings and commonly used QEMU options. For the full list of standard options, see the upstream QEMU invocation reference[68].

7.3.1 Command-line Syntax and Help

Use `qemu-system-riscv32` for RV32 programs and `qemu-system-riscv64` for RV64 programs, together with a CPU model matching the target architecture.

```
qemu-system-riscv32 -M nuclei_evalsoc[,property=value,...] \
  -cpu <model>[,property=value,...] [options]
```

Machine properties such as `download` and `soc-cfg` follow `-M`. CPU properties such as `ext` and `vlen` follow `-cpu`. Other options, such as `-smp` and `-kernel`, are separate command-line arguments.

Use the following commands to discover the options and models available in the installed build:

```
qemu-system-riscv32 --help
qemu-system-riscv32 -machine help
qemu-system-riscv32 -M nuclei_evalsoc,help
qemu-system-riscv32 -cpu help
qemu-system-riscv32 -d help
```

7.3.2 EvalSoC Machine Configuration

`-M nuclei_evalsoc[,property=value,...]`: Selects the EvalSoC machine and sets its properties.

`download=flashxip/flash/ilm/sram/ddr`: Selects the download mode; the default is `flashxip`. See *Boot and Image Loading* (page 848) for its relationship with image loading and startup addresses.

`aia=none/aplic/aplic-imsic`: Selects the external interrupt controller: PLIC, APLIC or APLIC with IMSIC, respectively. The default is `none`.

`aia-guests=0..4`: Sets the number of guest interrupt contexts for AIA-IMSIC. The S-level IMSIC window allocates one `0x1000` page for the supervisor context and each guest context, rounding the per-hart allocation up to a power-of-two number of pages. Its size therefore varies with `aia-guests`.

`cidu=on/off`: Enables or disables CIDU; it is enabled by default. `iregion.cidu=0` in `soc-cfg` also disables it, even if `cidu=on` is supplied.

`soc-cfg=<file.json>`: Overrides the built-in board configuration. The JSON format and supported fields are described below.

[68] https://www.qemu.org/docs/master/system/invocation.html

`debug=1`: Prints the configured memory layout, peripheral addresses and interrupt information at startup. Use it to inspect the effective board configuration.

soc-cfg JSON Configuration

Pass the configuration file as a machine property, for example `-M nuclei_evalsoc,download=ilm,soc-cfg=soc.json`. Without `soc-cfg`, QEMU uses the built-in default layout.

Note: The current implementation recognizes two top-level JSON objects: `general_config` and `download`. Partial overrides and arbitrary key order are supported; omitted or empty fields keep their defaults. Write numeric leaf values as strings. Decimal and `0x`-prefixed hexadecimal values are accepted, together with `M`/`G` suffixes for sizes.

The following example overrides memory regions, selected peripherals and download-mode startup addresses:

```
{
    "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"
        },
        "iregion": {
            "base": "0x18000000",
            "eclic": "1",
            "cidu": "1",
            "plic": "1",
            "clicintctlbits": "4"
        },
        "uart0": {
            "base": "0x20013000",
            "size": "0x1000",
            "irq": "1",
            "version": "0x00030100"
        },
        "qspi0": {
            "base": "0x20014000",
            "irq": "3",
            "version": "0x00010208",
            "xip_base": "0x34000000",
            "xip_size": "64M"
        },
        "qspi2": {
            "base": "0x20034000",
            "irq": "5",
            "enable": "0"
```

(continues on next page)

(continued from previous page)

```
        },
        "udma": {
            "base": "0x10015000",
            "size": "0x1000",
            "enable": "1"
        },
        "misc": {
            "base": "0x10012000",
            "size": "0x1000",
            "counter0_irq": "6",
            "counter1_irq": "7"
        },
        "xec0": {
            "base": "0x20002000",
            "irq": "8",
            "enable": "0"
        },
        "timer_freq": "32768",
        "irqmax": "1024"
    },
    "download": {
        "ilm": {
            "startaddr": "0x90000000"
        },
        "flashxip": {
            "startaddr": "0x30000000"
        },
        "flash": {
            "startaddr": "0x30000000"
        },
        "sram": {
            "startaddr": "0xB0000000"
        },
        "ddr": {
            "startaddr": "0x70000000"
        }
    }
}
```

The `soc-cfg` JSON file contains the `general_config` and `download` top-level objects. The table below lists each supported field, the objects that accept it, and its purpose. Object names without a prefix belong to `general_config`; download-mode objects are shown with their full `download.*` paths.

Table 7.3: `soc-cfg` fields and applicable objects

Field	Applies to objects	Description
`base`	`ddr`, `ilm`, `dlm`, `sram`, `norflash`, `test`, `uart0`, `qspi0`, `qspi1`, `qspi2`, `udma`, `misc`, `xec0`, `iregion`, `aplic_m`, `aplic_s`, `imsic_m`	Sets the base address of the corresponding memory or MMIO region.
`size`	`ddr`, `ilm`, `dlm`, `sram`, `norflash`, `test`, `uart0`, `qspi0`, `qspi1`, `qspi2`, `udma`, `misc`, `xec0`, `aplic_m`, `aplic_s`, `imsic_m`, `imsic_s`	Sets the region size or per-hart stride where applicable. Some sizes are recomputed by QEMU.
`irq`	`uart0`, `qspi0`, `qspi1`, `qspi2`, `udma`, `xec0`	Sets the peripheral's PLIC/APLIC source ID; routed to ECLIC/CIDU according to the board mapping below.
`counter0_i` `counter1_i`	`misc`	Set the MISC Counter0 and Counter1 PLIC/APLIC source IDs, respectively.
`enable`	`test`, `uart0`, `qspi0`, `qspi1`, `qspi2`, `udma`, `misc`, `xec0`, `aplic_m`, `aplic_s`, `imsic_m`, `imsic_s`	Enables or disables the corresponding device or address block.
`version`	`uart0`, `qspi0`, `qspi1`, `qspi2`	Selects a modeled peripheral revision where supported.
`xip_base`, `xip_size`, `xip_enable`	`qspi0`	Configures the QSPI0 XIP window.
`debug`	`iregion`	Accepted by the parser; it does not independently remove the fixed DEBUG subwindow.
`eclic`, `smpcc`, `cidu`, `plic`	`iregion`	Enable or disable the corresponding IREGION block. `cidu` is also subject to the machine-level `cidu=` option.
`clicintctl` `bits`	`iregion`	Sets ECLIC `CLICINTCTLBITS`.
`timer_freq`	`general_config`	Sets the SysTimer clock frequency.
`cpu_freq`	`general_config`	Sets the CPU/MISC countdown clock frequency.
`irqmax`	`general_config`	Sets the external interrupt-ID limit shared by ECLIC, PLIC and APLIC.
`startaddr`	`download.ilm`, `download.flashxip`, `download.flash`, `download.sram`, `download.ddr`	Sets the firmware entry address for the corresponding download mode.

`irqmax`

Defines the number of interrupt IDs visible to ECLIC, PLIC and APLIC. The default is `1024` and the accepted range is `40` through `1024`. The minimum covers the built-in layout, whose highest PLIC/APLIC source ID is 39. The visible range is therefore `0` through `irqmax - 1`. PLIC/APLIC source 0 is reserved, so their usable source IDs are `1` through `irqmax - 1`.

Note: UART0, uDMA, QSPI0/1/2 and the two MISC counters use the EvalSoC dual-line routing convention. XEC0 is a single-line source.

- Without an explicit IRQ override, each dual-line source is routed to both its small and large IDs, separated by 32.
- If a dual-line source has an explicit `soc-cfg` IRQ value, that source is routed only to the configured ID.
- Once any enabled dual-line source has an explicit IRQ value, every other dual-line source without an override is routed only to its large ID.

The following table shows the built-in allocation when no IRQ values are overridden:

IRQ_ID	ECLIC Interrupt ID	PLIC Interrupt ID	APLIC Interrupt ID	Source
0 & 32	19 & 51	1 & 33	1 & 33	UART0
1 & 33	20 & 52	2 & 34	2 & 34	uDMA
2 & 34	21 & 53	3 & 35	3 & 35	QSPI0
3 & 35	22 & 54	4 & 36	4 & 36	QSPI1
4 & 36	23 & 55	5 & 37	5 & 37	QSPI2
5 & 37	24 & 56	6 & 38	6 & 38	MISC Counter0
6 & 38	25 & 57	7 & 39	7 & 39	MISC Counter1
7	26	8	8	XEC0 Ethernet
n	n + 19	n + 1	n + 1	

`download.<mode>.startaddr`: Defines a startup-address override for each download mode. The JSON file can contain overrides for multiple modes; the machine-level `download=` option selects which one is used for the current run. These fields do not change the corresponding memory-region mappings.

For example, the following partial configuration maps ILM at `0x90000000` and SRAM at `0xB0000000`, while assigning a separate startup address to each mode:

```
{
    "general_config": {
        "ilm": {
            "base": "0x90000000",
            "size": "0x100000"
        },
        "sram": {
            "base": "0xB0000000",
            "size": "0x10000000"
        }
    },
    "download": {
        "ilm": {
            "startaddr": "0x90001000"
        },
        "sram": {
            "startaddr": "0xB0002000"
        }
    }
}
```

With this file, the selected mode determines which override is active:

```
# Uses 0x90001000; the ILM mapping still begins at 0x90000000.
qemu-system-riscv32 -M nuclei_evalsoc,download=ilm,soc-cfg=soc.json ...

# Uses 0xB0002000; the SRAM mapping still begins at 0xB0000000.
qemu-system-riscv32 -M nuclei_evalsoc,download=sram,soc-cfg=soc.json ...
```

For ILM, SRAM and flash XIP, omitting the selected mode ' s `startaddr` uses the corresponding region ' s base. `flash` inherits the flash XIP startup address unless overridden. For `ddr`, an explicitly configured DDR base is also the default startup address; otherwise, the historical startup address `0xA0000000` is retained.

7.3.3 CPU Configuration

`-cpu <model>[,property=value,...]`: Selects the CPU model and its properties. Use `-cpu help` to list available models, or refer to *the CPU model diagram* (page 836).

`ext=<extension_list>`: Enables additional ISA extensions, separated by _. For example, `ext=_v_xxldsp` enables the RISC-V vector and Nuclei DSP extensions. See *Supported ISA Extensions* (page 846) for the available extensions.

`vlen=<bits>`: Sets vector-register width when vector extensions are enabled. The default is 128 bits; values must be powers of two from 128 through 1024.

`elen=<bits>`: Sets the maximum vector element width; values must be powers of two from 8 through 64. The default is 32 when only Zve32* extensions are enabled, and 64 otherwise. An explicit value overrides this default.

`s=true/false`: Enables or disables supervisor privilege mode. Supervisor support includes S-mode ECLIC and Sstc when the corresponding features are enabled.

`big-endian=auto/on/off`: `auto` (the default) detects endianness from the boot ELF on EvalSoC; `on` forces big-endian and `off` forces little-endian, overriding ELF detection.

Supported ISA Extensions

The following extensions can be selected through `ext=`. Choose extensions supported by the selected CPU configuration and the guest program.

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
zibi	RISC-V Zibi Extension
zimop	RISC-V Zimop Extension
zcmop	RISC-V Zcmop Extension
zicfilp	RISC-V Zicfilp Extension
zicfiss	RISC-V Zicfiss Extension
zicsr	RV32/RV64 Zicsr Standard Extension
zifencei	RV32/RV64 Zifencei Standard Extension
zihintpause	Zihintpause Extension
zilsd	Zilsd extension (RV32 ONLY)
zclsd	Zclsd 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

continues on next page

Table 7.4 – continued from previous page

Extension	Functionality
zbkb	RISC-V Bitmanipulation Extension
zbkc	RISC-V Bitmanipulation Extension
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
smrnmi	Smrnmi Extension
smdbltrp	Machine-mode double-trap handling
ssdbltrp	Supervisor-mode double-trap handling
smcntrpmf	Smcntrpmf 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
xxlvqmacc	Nuclei custom vpu extension
xxlfbf	Nuclei scalar BF16 extension
xxlvfbf	Nuclei vector BF16 extension

7.3.4 Memory and CPU Count

`-m <size>`: Sets the DDR size, for example `-m 2G`. Without an override, QEMU uses the DDR size from `soc-cfg`, or the built-in default of 64 MiB.

Note: EvalSoC retains a compatibility rule: omitting `-m` or passing `-m 128M` keeps the DDR size from `soc-cfg` or the board default. To request exactly 128 MiB, set `general_config.ddr.size` to `"128M"` in the JSON file and omit `-m`. Large memory configurations also require sufficient host memory.

`-smp <count>`: Sets the number of harts. EvalSoC supports 1 through 64 harts and defaults to 1. The guest program must support the selected hart count.

7.3.5 Boot and Image Loading

`-kernel <file>` or `-bios <file>`: Selects the image to load. The choice is independent of the interrupt architecture; PLIC or ECLIC configuration no longer determines which option must be used.

The `download` machine property selects the startup region, and `download.<mode>.startaddr` in `soc-cfg` overrides its startup address. Match these settings and the memory layout to the guest image. See *soc-cfg JSON Configuration* (page 842) for configuration examples.

The following options apply to direct kernel boot. When using a bootloader such as the Nuclei Linux SDK freeloader, follow that bootloader ' s image and device-tree conventions instead:

`-append "<command line>"`: Sets the kernel command line in the generated device tree.

`-initrd <file>`: Loads an initial RAM disk when QEMU loads the kernel through `-kernel`.

`-dtb <file>`: Supplies a device tree in place of the one generated by EvalSoC. Its CPU, memory and peripheral descriptions must match the configured machine.

7.3.6 Console and Character Devices

`-nographic`: Disables graphical output and redirects the default serial console and QEMU monitor to the terminal. With the default multiplexed console, `Ctrl-a h` displays help and `Ctrl-a c` switches between the serial console and monitor.

`-display none`: Disables graphical output without changing serial or monitor routing.

`-serial stdio`: Connects UART0 to the host terminal. Use `-serial mon:stdio` to share that terminal with the QEMU monitor, or `-serial none` to disconnect the serial backend.

`-monitor none`: Disables the QEMU monitor. Use `-monitor stdio` for a dedicated monitor terminal when serial output uses another backend.

`-chardev <backend>,id=<id>,...`: Defines a character backend that can be connected with `-serial chardev:<id>`. For example, replace the terminal options in a command with the following to record UART0 output in a file:

```
-display none -monitor none \
-chardev file,id=uart0,path=uart.log -serial chardev:uart0
```

`-nodefaults`: Suppresses automatically created default devices and backends. Fixed EvalSoC peripherals remain present; explicitly select the serial, monitor and other backends needed by the guest.

7.3.7 Storage

`-drive file=<image>,if=sd,format=raw`: Attaches a raw SD-card image to the card connected to QSPI2. This is the filesystem disk used by the Linux SDK example.

Keep the corresponding QSPI controller enabled in `soc-cfg`. Disk and flash images are writable by default; add `readonly=on` to a drive for read-only access, or use `-snapshot` for temporary writes that are discarded on exit.

7.3.8 Networking

`-nic user,model=nuclei.xec`: Configures QEMU ' s user-mode network backend and connects it to the Nuclei XEC device. QEMU also supports other network backends; see *the XEC Ethernet usage guide* (page 852) for usage examples and TAP setup.

7.3.9 Plugins

`-plugin <file>[,arg=value,...]`: Loads a QEMU plugin. For the Nuclei NICE/VNICE plugin, pass the plugin file and user instruction library with `-plugin <plugin-file>,lib=<user-library>`. See *the NICE/VNICE plugin guide* (page 856) for build instructions and a complete example.

7.3.10 Execution and Timing

`-accel tcg`: Selects software emulation. TCG supports the bare-metal examples in this guide. For SMP execution, `-accel tcg,thread=multi` requests multi-threaded TCG; use it without `-icount`.

`-icount shift=0`: Derives virtual time from executed instructions, with one instruction corresponding to one nanosecond at `shift=0`. This controls virtual timing rather than modelling hardware cycle counts. `icount` is incompatible with multi-threaded TCG, so it is not recommended when running SMP programs: omit `-icount` to allow the vCPUs to run on separate TCG threads. Use it with SMP only when instruction-based virtual time or deterministic execution is specifically required. See TCG instruction counting[69] for additional settings.

7.3.11 Debugging

`-S`: Pauses the CPUs at startup until a debugger or monitor resumes execution.

`-s`: Starts the GDB server on TCP port 1234. It is shorthand for `-gdb tcp::1234`; combine it with `-S` to debug from the first instruction. In GDB, connect with `target remote localhost:1234` and start execution with `continue`.

`-gdb <endpoint>`: Selects another GDB endpoint, for example `-gdb tcp::3333`. See the upstream GDB usage guide[70] for debugging details.

`-d <items>`: Enables diagnostic logging. Run `-d help` to list supported items; for example, `-d guest_errors,unimp` reports guest errors and unimplemented accesses.

`-D <file>`: Writes diagnostic logs to the specified file, for example `-d guest_errors,unimp -D qemu.log`. These logs are separate from UART output.

Note: Before using GDB or diagnostic logs for in-depth debugging, first check that the command-line options and guest image match: verify the QEMU/CPU architecture, ELF entry and load addresses, selected `download` mode, and memory settings. If `soc-cfg` is used, add `debug=1` to the `-M nuclei_evalsoc` properties to inspect the effective memory regions and peripheral layout, then compare them with the guest linker script, device tree and driver configuration. Rerun QEMU with `debug=1` after changing `soc-cfg` to confirm the resulting configuration.

---

[69] https://www.qemu.org/docs/master/devel/tcg-icount.html
[70] https://www.qemu.org/docs/master/system/gdb.html

7.4 Use Nuclei QEMU in Nuclei SDK

Setup Tools and Environment

1. Download the nuclei-sdk[71], checkout to `master` branch.
2. Download RISC-V GNU Toolchain form Nuclei Download Center[72].
3. Download Nuclei Qemu form Nuclei Download Center[73].
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 xxldsp.

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.

Support for Nuclei SDK Cases on QEMU

`Y` - Successfully run and consistent with hardware

`N` - Successfully run but inconsistent with hardware

`F` - Failed

Cases	SMP=1	SMP>1	Description (additional compilation parameters and running status)
benchmark/coremark	Y		
benchmark/dhrystone	Y		
benchmark/ dhrystone_v2.2	Y		
benchmark/whetstone	Y		
benchmark/ whetstone_v1.2	Y		
cpuinfo/	Y		
demo_backtrace/	Y		
demo_cache/	F		QEMU does not support cache emulation.
demo_cidu/		Y	SMP=4, XLCFG_CIDU=1, ICOUNT_OPT="" . ECLIC v2: XLCFG_ECL
demo_clint_timer/	Y		
demo_dsp/	Y		
demo_ecc/	N		ECC error injection and correction behavior is not modeled.
demo_eclic/	Y		
demo_eclic_stress/	Y		XLCFG_SMODE=0/1, XLCFG_SSTC=0/1; ECLIC v2 uses XLCFG_ECLI
demo_eclic_umode/	Y		XLCFG_SMPU=1; ECLIC v2 uses XLCFG_ECLIC=2 and ECLIC_HWCTX

[71] https://github.com/Nuclei-Software/nuclei-sdk
[72] https://nucleisys.com/download.php
[73] https://nucleisys.com/download.php

Table 7.5 – continued from previous page

Cases	SMP=1	SMP>1	Description (additional compilation parameters and running status)
demo_nice/	Y		
demo_plic/	Y	Y	XLCFG_PLIC=1.
demo_pma/	Y		XLCFG_CCM=1, XLCFG_HPM=1.
demo_pmp/	Y		XLCFG_SMODE=0 for M-mode-only; ECLIC v2 uses XLCFG_ECLIC=2 a
demo_profiling/	Y		DOWNLOAD=sram.
demo_smode_clint/	Y		
demo_smode_eclic/	Y		XLCFG_SMODE=1; ECLIC v2 uses XLCFG_ECLIC=2 and ECLIC_HWC
demo_smode_plic/	Y		XLCFG_PLIC=1.
demo_smpcc/	N		Control registers are modeled; cluster-cache behavior is not modeled.
demo_smpu/	Y		XLCFG_SMPU=1, CHK_MODE=0 through 5.
demo_sstc/	Y		
demo_stack_check/	Y		
demo_timer/	Y		
demo_vnice/	Y		
empty/	Y		
exception_mmode/	Y		
exception_smode/	Y		XLCFG_SMODE=1.
exception_umode/	Y		XLCFG_SMODE=1, XLCFG_SMPU=1.
helloworld/	Y		
lowpower/	Y		
smphello/		Y	SMP=4, ICOUNT_OPT="" .
freertos/demo/	Y		
freertos/smpdemo/		Y	SMP=4, ICOUNT_OPT="" .
rtthread/demo/	Y		
rtthread/demo_smode/	Y		XLCFG_SMODE=1, XLCFG_SSTC=1.
rtthread/msh/	Y		
threadx/demo/	Y		
threadx/smpdemo/		Y	SMP=4, ICOUNT_OPT="" .
ucosii/demo/	Y		

And Nuclei QEMU and Nuclei SDK are deeply integrated in Nuclei Studio, you can also use it in Nuclei Studio, see *Nuclei Studio IDE* (page 3).

7.5 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
```

Use `soc.json`, `freeloader_qemu.elf` and `disk.img` from the same Linux SDK configuration. The JSON supplies the board layout, the freeloader handles the SDK boot flow, and the disk image provides the SD-card contents through QSPI2. Match the CPU features, hart count and DDR size to the SDK build.

For individual options, see *soc-cfg configuration* (page 842), *CPU Configuration* (page 846), *Memory and CPU Count* (page 848) and *Storage* (page 849).

7.6 Nuclei XEC Ethernet User Guide

Official QEMU networking documentation: https://wiki.qemu.org/Documentation/Networking

QEMU supports multiple network backends. To meet the requirements of the Nuclei Ethernet specification (Nuclei_xec_gen2_doc_v1.0.1) and its intended use cases, the current Nuclei XEC implementation supports both User and TAP mode networking.

User mode is recommended for most use cases. It works on both Linux and Windows, requires no administrator privileges or host network setup, and provides outbound network access through a built-in NAT service. TAP mode is intended for cases that require the guest to communicate directly with the host, other virtual machines, or external hosts, or that require protocols and traffic patterns not supported by User mode. TAP mode provides more complete networking and better performance, but requires administrator privileges and additional host configuration.

Note: Before starting QEMU, check that the Linux device tree matches the QEMU XEC configuration. The guest image must include the Nuclei XEC driver, the XEC node must be enabled, and the PHY address must be `2` for the current QEMU XEC model. If a custom `soc-cfg` file is used, its `xec0` base address, size, interrupt number, and enable state must also match the Linux device tree.

7.6.1 User Mode Networking

QEMU's User mode networking uses the Slirp library to provide NAT forwarding in user space. With the `-nic user` option, QEMU provides a complete user mode network stack without requiring root privileges. The default virtual network configuration is as follows:

```
guest (10.0.2.15)  <------>  Firewall/DHCP server <-----> Internet
                     |          (10.0.2.2)
                     |
                     ---->  DNS server (10.0.2.3)
                     |
                     ---->  SMB server (10.0.2.4)
```

QEMU's User mode networking has the following main limitations:

- Forwarding runs in user space without kernel acceleration, resulting in lower throughput and making this mode unsuitable for high-volume data transfers.
- ICMP support is limited, so `ping` may not work with hosts outside the virtual network.
- External hosts cannot directly access the guest; network connections must be initiated by the guest.
- Multicast and direct networking between multiple QEMU instances are not supported.

Both the Windows and Linux versions of Nuclei QEMU include the Slirp library required for User mode networking. To access the network, specify `-nic user,model=nuclei.xec` to select the User network backend and the Nuclei XEC model. For example: `qemu-system-riscv64.exe -M nuclei_evalsoc,download=flashxip -smp 8 -m 2G -cpu nuclei-ux900fd,ext=_zicbom_svpbmt_sscof -bios freeloader.elf -nographic -drive file=/path/to/your/img,if=sd,format=raw -nic user,model=nuclei.xec`.

With a matching guest device tree and XEC-enabled Linux image, the startup log should contain messages similar to the following. The exact timestamps and MAC address may vary:

```
[    1.361511] nuclei-xec 10002000.xec eth0: using RGMII interface
[    1.370178] Generic PHY 10002000.xec-ffffffff:02: attached PHY driver
[    1.371734] nuclei-xec 10002000.xec eth0: XEC mac at 0x10002000 irq 4
[    2.850067] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpc: broadcasting discover
udhcpc: broadcasting select for 10.0.2.15, server 10.0.2.2
```

(continues on next page)

(continued from previous page)

```
udhcpc: lease of 10.0.2.15 obtained from 10.0.2.2, lease time 86400
adding dns 10.0.2.3
```

The assigned address, default route, link state, and neighbor resolution can then be checked from the guest. The following commands also confirm that the emulated PHY is attached at address `2` and that packets are exchanged without interface errors:

```
# ip -4 addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0
# ip route
default via 10.0.2.2 dev eth0
10.0.2.0/24 dev eth0 scope link src 10.0.2.15
# cat /etc/resolv.conf
nameserver 10.0.2.3
# ethtool eth0 | grep -E 'PHYAD|Link detected'
    PHYAD: 2
    Link detected: yes
# ping -c 1 10.0.2.2 > /dev/null
# ip neigh show dev eth0
10.0.2.2 lladdr 52:55:0a:00:02:02 ref 1 used 0/0/0 probes 4 REACHABLE
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr E6:C4:F2:16:EC:B1
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
```

For normal network use, `wget` can retrieve a web page or download a file from an accessible server. These commands depend on the host ' s DNS and external network connectivity when a domain name or external server is used:

```
# wget -T 10 -O - http://<server>/
# wget -T 10 -O /tmp/file.bin http://<server>/file.bin
```

Use `netstat` to inspect the routing table, active TCP and UDP sockets, or listening TCP ports:

```
# netstat -rn
# netstat -tn
# netstat -un
# netstat -ltn
```

To verify TCP and an application-layer protocol without depending on Internet access, start a temporary HTTP server on the host:

```
$ python3 -m http.server 18080 --bind 127.0.0.1
```

Add a `guestfwd` rule to the User backend. It maps the guest-visible address `10.0.2.100:80` to the temporary server on `127.0.0.1:18080`:

```
-nic user,model=nuclei.xec,guestfwd=tcp:10.0.2.100:80-tcp:127.0.0.1:18080
```

The guest can then retrieve content with `wget`. A successful transfer confirms that TCP and HTTP traffic pass through XEC and the User backend. The response body and transfer size depend on the directory served by the host and are abbreviated below:

```
# wget -T 10 -O - http://10.0.2.100/
Connecting to 10.0.2.100 (10.0.2.100:80)
writing to stdout
...
```

(continues on next page)

(continued from previous page)

```
-                    100% |*******************************|   876  0:00:00 ETA
written to stdout
```

7.6.2 TAP Mode Networking

QEMU ' s TAP mode networking is a high-performance, full-featured networking mode. It connects the guest ' s network interface directly to the host ' s physical or virtual network, enabling bidirectional communication between the guest and the host, external networks, and other virtual machines. Unlike User mode networking, TAP mode uses kernel-space forwarding, supports nearly all network protocols, and provides performance close to that of a physical network interface. However, it requires root privileges and manual network configuration. The configuration steps for Linux and Windows are provided below.

Linux

Host Forwarding Rules

Configure forwarding rules on the host so that packets from the guest are not blocked or filtered.

```
# Configure IPv4 forwarding rules
$ cat /proc/sys/net/ipv4/ip_forward # Check whether the value is 1 (usually the␣
↪default), which indicates that IPv4 forwarding is enabled; otherwise, enable it␣
↪below
$ sudo sysctl -w net.ipv4.ip_forward=1
# If the host uses NAT networking, also configure NAT forwarding rules
$ sudo iptables -t nat -A POSTROUTING -o ens33 -j MASQUERADE # Replace ens33 with␣
↪the host's network interface name here and below
$ sudo iptables -t filter -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j␣
↪ACCEPT
$ sudo iptables -t filter -A FORWARD -i tap0 -o ens33 -j ACCEPT
```

Creating a TAP Device

To use QEMU ' s TAP mode networking, create a TAP device on the host to connect the QEMU guest to the host network.

```
$ sudo ip tuntap add mode tap dev tap0 # Create a TAP device; if tap0 is already␣
↪in use, create tap1, tap2, etc.
$ sudo ip link set tap0 up # Enable the TAP device
$ sudo ifconfig tap0 192.168.1.1 netmask 255.255.255.0 # Configure the TAP device
↪'s IP address and netmask
```

Example QEMU Command

```
$ qemu-system-riscv64 -M nuclei_evalsoc,download=flashxip -smp 8 -m 2G -cpu nuclei-
↪ux900fd,ext=_zicbom_svpbmt_sscof -bios /path/to/freeloader.elf -nographic -drive␣
↪file=/path/to/disk.img,if=sd,format=raw -nic tap,ifname=tap0,script=no,
↪downscript=no
```

Parameter Descriptions

- `-bios`: Specifies the freeloader image, which must be built with the XEC driver enabled.
- `-nic tap,ifname=tap0,script=no,downscript=no`: XEC is already attached directly to the system bus of the Nuclei QEMU evalsoc machine, so no additional option is required to specify the Ethernet device. Use `-nic` to select the TAP network backend and `ifname` to specify the host TAP device. The `script=no,downscript=no` settings disable QEMU ' s automatic TAP configuration, leaving the network parameters entirely under user control. If script-based configuration is required, paths to the corresponding scripts can be specified instead.

Guest Configuration

- Configure the XEC virtual network interface.

```
$ ifconfig eth0 192.168.1.100 netmask 255.255.255.0 # Configure the XEC virtual␣
↪network interface's IP address and netmask
$ route add default gw 192.168.1.1 # Use the TAP device as the default gateway
```

- Edit `/etc/resolv.conf` to configure DNS.

Windows

The following steps were tested on Windows 10.

Creating a TAP Device

Creating a TAP device on Windows requires an additional driver package, such as `tap-windows-9.21.2`, which must be downloaded and installed separately.

After installation, a virtual network device appears under `Control Panel\Network and Internet\Network Connections`. Rename the device to the name that will be used on the QEMU command line, such as `tap0`.

Host Forwarding Rules

Run the following commands, as well as those in the remaining Windows steps, in an elevated Command Prompt or PowerShell session.

```
# Enable IPv4 forwarding
$ Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
↪" -Name IPEnableRouter -Value 1 -Type DWord
$ netsh interface ipv4 set interface "tap0" forwarding=enabled
$ netsh interface ipv4 set interface "Ethernet0" forwarding=enabled  # Replace␣
↪Ethernet0 with the actual physical network interface name
```

Creating a NAT Rule

Configure NAT to forward traffic from the subnet shared by the TAP device and the guest to external networks through the host ' s physical network interface. Using the same TAP and guest IP addresses as in the Linux example above, run:

```
$ New-NetNat -Name "QEMU_NAT" -InternalIPInterfaceAddressPrefix "192.168.1.0/24" #␣
↪QEMU_NAT is the NAT network name and can be customized
```

Configuring Firewall Rules (Optional)

Depending on the local firewall configuration, the following rules may be required to allow traffic between subnets:

```
# Create an inbound rule to allow NAT-related traffic
$ New-NetFirewallRule -DisplayName "Allow NAT Inbound" -Direction Inbound -
↪Protocol Any -LocalPort 1024-65535 -Action Allow
# Create an outbound rule to allow all traffic
$ New-NetFirewallRule -DisplayName "Allow All Outbound" -Direction Outbound -
↪Action Allow
```

Running QEMU

Run QEMU in the same way as in the Linux environment.

7.7 Nuclei NICE Plugin

Nuclei QEMU provides a NICE plugin mechanism for custom RISC-V instructions. With this mechanism, users can keep the standard Nuclei QEMU binary unchanged and implement their own NICE or VNICE instruction behavior in an external shared library.

This mechanism is mainly used to emulate user-defined instructions in the custom opcode space. At runtime, QEMU decodes the instruction fields, reads the source operands, and dispatches the instruction to the handler function defined by the user library. If no handler matches the current instruction, QEMU raises an illegal-instruction exception.

7.7.1 How to Load the NICE Plugin

The plugin is loaded through qemu ' s generic `-plugin` option. Taking Linux as an example:

```
$ qemu-system-riscv32 -M nuclei_evalsoc,download=ilm \
    -cpu nuclei-n300fd,ext=_v,vlen=128,elen=64 \
    -plugin /path/to/qemu/lib/qemu/libnice.so,lib=/path/to/libuser_nice.so,
↪verbose=on \
    -nographic -serial stdio -kernel demo_nice.elf
```

In this command:

- `/path/to/qemu/lib/qemu/libnice.so` is the NICE plugin provided by Nuclei QEMU itself for Linux version, it is built and installed together with qemu. For Windows version, the `libnice.dll` file is used and is also stored in the `/bin/lib/qemu/` directory of the Windows QEMU package.
- `lib=` is a required parameter. Users should implement their custom NICE & VNICE instructions in `user_nice.c` first, then build that file into a shared library, and finally pass the generated library path through `lib=`. For example, on Linux it can be built as follows:

  ```
  gcc -O2 -shared -fPIC -I/path/to/qemu/include user_nice.c -o libuser_nice.so
  ```

  On Windows, you need to build it into a `.dll` library, for example:

  ```
  gcc -O2 -shared -fPIC -I/path/to/qemu/bin/include user_nice.c -o libuser_nice.
  ↪dll
  ```

- `verbose=on` is an optional parameter. When enabled, the plugin will print the loaded instruction entries and related matching information at startup, which is convenient for debugging.

7.7.2 How to Extend `user_nice.c`

UUsers need to provide a `user_nice.c` file to describe and implement their custom instructions. They only need to fill in `nice_def_t nice_defs[]` according to the specification in `/path/to/qemu/include/nice_api.h`. Several key macros are described below:

- `CI_ANY`: used to indicate that the corresponding register field is not used as a sub-opcode and instead carries a normal register value; if the corresponding register field is used as a sub-opcode, the sub-opcode value must be specified explicitly.
- `CI_CALL_INT`: integer mode, handler prototype is `uint64_t fn(uint64_t...)`.
- `CI_CALL_F32`: single-precision floating-point mode, handler prototype is `float fn(float...)`.
- `CI_CALL_F64`: double-precision floating-point mode, handler prototype is `double fn(double...)`.
- `CI_CALL_VEC`: vector mode, handler prototype is `void fn(qemu_plugin_nice_info_t *info)`. In this mode, the handler can directly access vector registers through `rd_vreg`, `rs1_vreg`, and `rs2_vreg`.
- `NICE_VSEW_E*`: used to set the element width of VNICE instructions.

The following is an example file:

```
/*
 * user_nice.c — Example user-side instruction handler library.
 *
 * This file demonstrates how to define custom RISC-V instruction handlers
 * for use with the QEMU NICE plugin.
 */

#include "nice_api.h"

/* --- custom-0 integer handlers ------------- */

static uint64_t insn_add(uint64_t a, uint64_t b)
{
    return a + b;
}

static uint64_t insn_mac(uint64_t acc, uint64_t a, uint64_t b)
{
    return acc + a * b;
}

static uint64_t insn_popcount(uint64_t a)
{
    return (uint64_t)__builtin_popcountll(a);
}

/* --- custom-0 FPU handlers: ---------------- */

static float insn_fadd(float a, float b)
{
    return a + b;
}

/* --- custom-2 RVV vector handlers (CI_CALL_VEC) ---------------------------
 *
 * Pure-C loop implementation; runs on the QEMU host (x86-64, aarch64, ⋯).
 * Each handler receives qemu_plugin_nice_info_t* with zero-copy pointers
 * directly into QEMU's vreg[] array:
 *
 *   info->rd_vreg   — vd  destination (writable, info->vlenb bytes)
 *   info->rs1_vreg  — vs1 source (valid when xs1=1)
 *   info->rs2_vreg  — vs2 source (valid when xs2=1)
 *   info->vl        — current vector length
 *   info->vsew      — element width: 0=e8 1=e16 2=e32 3=e64
 *   info->vstart    — first active element
 * ------------------------------------------------------------------------- */

/* vadd_i32: vd[i] = vs1[i] + vs2[i]  (e32, funct7=0x01, funct3=7) */
static void insn_vadd_i32(qemu_plugin_nice_info_t *info)
{
    uint32_t vl = (uint32_t)info->vl;
    int32_t       *vd  = (int32_t *)info->rd_vreg;
    const int32_t *vs1 = (const int32_t *)info->rs1_vreg;
    const int32_t *vs2 = (const int32_t *)info->rs2_vreg;
    for (uint32_t i = (uint32_t)info->vstart; i < vl; i++) {
        vd[i] = vs1[i] + vs2[i];
    }
}

/* --- Instruction definition table ---------------------------------------- */
nice_def_t nice_defs[] = {
```

(continues on next page)

(continued from previous page)

```
    /* opcode,  funct7, rs1_enc, rs2_enc, funct3, fn,           call_type,   vsew_
↪enc */

    /* custom-0 integer */
    { 0x0b,     0x00,   CI_ANY,  CI_ANY,  7,      insn_add                      ␣
↪   },
    { 0x0b,     0x40,   CI_ANY,  CI_ANY,  7,      insn_mac                      ␣
↪   },

    /* custom-1 bitcount (rs2 field = sub-opcode; xs2=0 → funct3=6) */
    { 0x2b,     0x00,   CI_ANY,  1,       6,      insn_popcount                 ␣
↪   },

    /* custom-0 FPU (funct7[5]=1 → is_fpu=1; lower bits = FPU sub-opcode) */
    { 0x0b,     0x20,   CI_ANY,  CI_ANY,  7,      insn_fadd,    CI_CALL_F32     ␣
↪   },

    /* custom-2 (0x5b) RVV vector — CI_CALL_VEC
     *
     * vadd: same encoding for e8 and e32; vsew_enc routes to the right handler
     *   funct7=0x01 (vm=1=unmasked), funct3=7 (xd=xs1=xs2=1)              */
    { 0x5b,     0x01,   CI_ANY,  CI_ANY,  7,      insn_vadd_i32,CI_CALL_VEC, NICE_
↪VSEW_E32 },

    { CI_TABLE_END, 0, 0, 0, 0, NULL }
};
```

On Windows, users should follow the same workflow: keep implementing custom instructions in `user_nice.c`, build it into a Windows shared library, and pass that generated library through `lib=`.

7.7.3 Usage Notes

The following points should also be noted when using the NICE plugin:

- The current dispatch path is used for instructions in `custom-0`, `custom-1`, `custom-2` and `custom-3` opcode spaces. If QEMU enables the `Xxldsp` extension at the same time, please **DO NOT** use the `custom-3` opcode space. If QEMU enables the `Xxlcz` extension at the same time, please **DO NOT** use the `custom-2` or `custom-3` opcode space.
- Instructions already handled by QEMU built-in decoders will not enter the NICE plugin dispatch path. For example, built-in Nuclei NICE or VNICE instructions that have already been implemented in QEMU will continue to use the built-in implementation.
- When using `CI_CALL_F32` or `CI_CALL_F64`, the corresponding floating-point extension needs to be enabled on the cpu.
- When using `CI_CALL_VEC`, the cpu needs to enable the RISC-V V extension, otherwise the vector register pointers in the callback context will not be valid.
- If the current instruction does not match any entry in `nice_defs[]`, QEMU will trigger an illegal instruction exception.

7.8 Changelog

7.8.1 Version 2026.10

This release is based on **2025.10** version with some new features and bug fixes introduced.

Interrupt and Timer Support

- Improved ECLIC multi-hart routing and M/S-mode interrupt support.
- Added ECLIC V2 automatic context save/restore, shadow GPRs and nested-trap handling.
- Fixed ECLIC interrupt tail-chaining and `jalmnxti` return behavior.
- Fixed ECLIC state reset so interrupts continue to work after a restart.
- Reworked the SysTimer around shared MTIME state.
- Improved Sstc integration on EvalSoC.
- Improved multicore timer delivery, timer control and RV32 compare access.
- Improved CIDU distribution, inter-core interrupts, semaphores and supervisor access control.
- Improved PLIC interrupt routing and integration.
- Improved AIA configuration, IMSIC address alignment and KVM AIA integration.

CPU Features and Extensions

- Added Nuclei sMPU protection.
- Added Nuclei stack-bound checking.
- Added RNMI and double-trap extension support.
- Added NICE plugin support for Nuclei NICE and VNICE instructions.
- Added Zicfilp, Zicfiss, Zibi, Zimop and Zcmop extension support.
- Fixed incorrect results for several Nuclei Xxldsp instructions.
- Fixed Xxlcz GP-relative stores, operand-boundary behavior and RV64 support.
- Fixed BF16/VBF16 extension dependencies, conversions and reciprocal-estimate behavior.
- Fixed Nuclei custom VNICE instructions.
- Adjusted RVV tail-agnostic handling to match Nuclei hardware behavior.
- Improved fractional-LMUL instruction behavior and dependency validation.
- Fixed crashes when writing some Nuclei custom CSRs.
- Fixed scratch-stack switching for synchronous exceptions nested inside ECLIC interrupts.
- Updated the default RISC-V Privileged ISA specification version to 1.13.

Peripheral and SoC Module Support

- Added Nuclei USART support.
- Improved Nuclei QSPI support with the SPI v1.2.8 revision and QSPI0 uDMA transfers.
- Added Nuclei uDMA support.
- Added Nuclei XEC Ethernet controller support.
- Added Nuclei MISC support. The MISC device is instantiated at `0x10012000`, the address used by the GPIO device in 2025.10; GPIO is no longer modeled in this release.
- Added IREGION capability register support.
- Improved SD-card SPI CRC handling.

- Removed UART1, only UART0 is instantiated now.

EvalSoC Configuration and Boot

- Expanded `soc-cfg` support for device instantiation, MMIO sizes and partial overrides.
- Added `irqmax` validation.
- Improved memory-map overlap checks.
- Added big-endian guest support and ELF-based automatic endianness detection.
- Added ELF ISA compatibility validation.
- Removed the MROM reset trampoline.

Platform Compatibility and Fixes

- Fixed GLib version-related warnings in the Windows QEMU build.

7.8.2 Version 2025.10

- Iregion now can be accessed via 1-8bytes per ops
- Adjust the configuration of ELEN
- Support pausing Nuclei systimer
- Support smode ECLIC
- Add support for Nuclei xxlfbf & xxlvfbf extensions
- Add optional AIA APLIC & IMSIC support on nuclei_evalsoc
- Cherry-pick Smcntrpmf extension support
- Add Nuclei N300e core support
- Update to reflect the overflow flag of xxldsp instructions through ‘ ucode ’
- Implement some Nuclei custom csrs
- Fixed some issues related to csr access
- Adapt the SSTC extension on nuclei_evalsoc
- Add zhinx/zhinxmin/zdinx imply rules
- Fixed the support for single-letter input of the ‘ ext= ’ option
- Increase the default size of the flash to 64MB
- Fixed overflow caused by setting the ‘ mtimecmp ’ too large in clint timer
- Trigger system reset rather than systimer reset when writing ‘ msftrst ’
- Fix a series of issues during interrupt handling in eclic

7.8.3 Version 2025.02

- Synchronize to the upstream QEMU 9.0
- Remove the Nuclei Demosoc support
- Add new cpu support for the Nuclei 200 and 1000 series
- Add Nuclei custom Xxlvqmacc extension support
- Add basic support for Nuclei gpio, qspi, test finisher, etc
- Zilsd/Zclsd extensions sync to upstream releases 1.0

- Add support for rv32/64 linux-user
- The maximum number of cpus supported by the Nuclei evalsoc smp has been adjusted from 16 to 64
- Fixed failure to respond to external interrupt in ECLIC
- Fixed Nuclei systimer support in CLINT mode

7.9 Known Issues

For a summary of reported Nuclei QEMU issues, see https://github.com/riscv-mcu/qemu/issues/1.

7.9.1 Open Known Issues

No open known issues are recorded in the user guide for this release. Please refer to https://github.com/riscv-mcu/qemu/issues/1 for the latest reported issues.

7.9.2 Fixed issues on Nuclei QEMU 2026.10

- The PLIC/ECLIC interrupt controller cannot be automatically switched, see https://github.com/riscv-mcu/qemu/issues/8
- Can not run SMP version RTOS such FreeRTOS, RT-Thread, Zephyr and ThreadX, see https://github.com/riscv-mcu/qemu/issues/9
- Sstc is not fully compatible with the Nuclei SysTimer, see https://github.com/riscv-mcu/qemu/issues/10
- Calculation errors in the `dkmda` and `dkmxda` instructions in Nuclei Xxldsp, see https://github.com/riscv-mcu/qemu/issues/11
- Incorrect results for several Nuclei Xxldsp instructions, see https://github.com/riscv-mcu/qemu/issues/12
- Incorrect Xxlcz instruction behavior and incomplete RV64 support, see https://github.com/riscv-mcu/qemu/issues/13
- BF16/VBF16 workloads may trap or produce incorrect results, see https://github.com/riscv-mcu/qemu/issues/14
- Writing some Nuclei custom CSRs may crash QEMU, see https://github.com/riscv-mcu/qemu/issues/15
- Incorrect scratch stack switching during nested traps, see https://github.com/riscv-mcu/qemu/issues/16
- ECLIC interrupts may stop working after a GDB restart, see https://github.com/riscv-mcu/qemu/issues/17
- Per-hart CLINT MSIP readback may prevent secondary harts from booting in ECLIC mode, see https://github.com/riscv-mcu/qemu/issues/18

CHAPTER

EIGHT

NUCLEI MODEL

8.1 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 2026.10**.
- The *xlmodel* is supported on both **Linux and Windows** system.
- The *xlmodel* supports **GDB debugging** capabilities (GDB RSP interface) with Nuclei Studio, including **SMP multi-thread debugging**, breakpoints and watchpoints.
- 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, offering a detailed view of the functions call graph and the cycle count distribution across functions in the test code.
- The *xlmodel* supports **flame graph** generation for visualizing call stack performance and multi-core/multi-threaded application execution.
- The *xlmodel* supports **semihosting** (guest `printf`/file-I/O/exit via the RISC-V semihosting convention).
- The *xlmodel* can run **big-endian** RISC-V ELF files directly; target endianness is detected from the ELF header.
- The *xlmodel* can boot a **RISC-V Linux kernel** with the Nuclei Linux SDK, in both uniprocessor and SMP mode. See *Use Nuclei Model in Nuclei Linux SDK* (page 876).
- The *xlmodel* supports SoC configuration through a `soc-cfg` JSON file, and provides a `--fastmode` option for maximum functional simulation speed.
- The *xlmodel* can easily extend **user-defined** instructions, namely **Nuclei NICE instructions**.
- Xlspike supports RV32IMAFDCBPV and RV64IMAFDCBPV ISA.

Since this model is also integrated in Nuclei Studio, you can use it to do

- Profiling and Code Coverage
  - Demonstrating the acceleration of Nuclei Model NICE/ VNICE instructions through Profiling: https://doc.nucleisys.com/nuclei_studio_supply/18-demonstrate_NICE_VNICE_acceleration_of_the_Nuclei_Model_through_profiling/
  - Nuclei Model combined with Nice Wizard for rapid verification of NICE/ VNICE instruction acceleration: https://doc.nucleisys.com/nuclei_studio_supply/19-rapid_verification_of_NICE_VNICE_acceleration_with_Nuclei_Model_and_NICE_Wizard/
  - *Using Coverage, Profiling, and Call Graph* (page 234)
- NICE Instruction Design and Modeling - *Nuclei NICE Wizard* (page 284)

- Debugging in Nuclei Studio - *Using Nuclei Model Debugger* (page 291)
- Flame Graph - *Nuclei Studio use xlmodel flame graph* (page 296)

Note: Before using *xlmodel*, you should check the current version using the `--version` parameter to verify whether it is the latest release. The version output also contains the git commit hash of the build.

The version output of the current latest version is

```
    SystemC 2.3.4-Accellera --- May 16 2024 16:02:03
    Copyright (c) 1996-2022 by all Contributors,
    ALL RIGHTS RESERVED
[XLMODEL-INFO] xl_cpumodel Commit: ded3124
[XLMODEL-INFO] xl_cpumodel Version 2026.10 - 1.3.0, Build Time Nov 10 2026,␣
↪17:12:02
```

8.2 Design and Architecture

8.2.1 SystemC components

Brief description of the *xlmodel* SystemC components:

- Evalsoc: SystemC modeling of EvalSoC, including the binding relationships of all components and load image functionality
- Cluster: Modeling of single-core or multi-core CPU, using xlspike internally to execute instructions
- Bus: Simple bus manager
- Memory: Simple memory, including default sram/ddr iregion and flashxip iregion
- Timer: Nuclei SysTimer unit (ECLIC mode and CLINT-compatible window)
- Cidu: Core Interrupt Distribution Unit for multi-core interrupt routing, inter-core interrupts and semaphores
- EvalUart: Nuclei EvalSoC UART
- EvalQspi: Nuclei EvalSoC QSPI

8.2.2 Address Allocation of Evalsoc in xlmodel

The following table shows the default EvalSoC address map in *xlmodel*. Memory regions and peripherals can be relocated or resized with command-line options (such as `--ddrbase` and `--ddrsz`) or through `soc-cfg`. IREGION devices use fixed offsets from `iregion.base`, whose default value is `0x18000000`.

Table 8.1: Default EvalSoC address map

Type	Component	Base address	Size	Description
Memory	XIP/NOR flash	`0x20000000`	`0x10000000`	Boot flash and XIP memory.
Memory	DDR	`0x80000000`	`0x04000000`	General-purpose system memory; enlarge it when booting Linux.
Memory	ILM	`0x80000000`	`0x00800000`	Instruction local memory.
Memory	DLM	`0x90000000`	`0x00800000`	Data local memory.
Memory	SRAM	`0xA0000000`	`0x20000000`	On-chip SRAM address window.
Fixed peripheral	UART0	`0x10013000`	`0x00001000`	EvalSoC serial controller.
Fixed peripheral	QSPI0	`0x10014000`	`0x00001000`	SPI/QSPI controller.
Fixed peripheral	QSPI1	`0x10024000`	`0x00001000`	SPI/QSPI controller.
Fixed peripheral	QSPI2	`0x10034000`	`0x00001000`	SPI/QSPI controller.
IREGION	ECLIC	`iregion + 0x20000`	`0x00010000`	Nuclei ECLIC unit.
IREGION	TIMER	`iregion + 0x30000`	`0x00010000`	Nuclei SysTimer/ CLINT-compatible window.
IREGION	SMP/CC	`iregion + 0x40000`	`0x00001000`	SMP and cluster-cache control unit.
IREGION	CIDU	`iregion + 0x50000`	`0x00010000`	Core interrupt distribution unit.
IREGION	PLIC	`iregion + 0x4000000`	`0x04000000`	Platform-level interrupt controller.

For details, please see *Nuclei Model Command-line Options* (page 866).

8.2.3 Nuclei CPU Features Supported in xlmodel

The following table summarizes the current implementation status.

Table 8.2: Nuclei feature status

Feature	Status
NMI	Supported.
TIMER	Supported; see *TIMER Support* (page 866).
PLIC	Supported; see *PLIC Support* (page 866).
ECLIC v1/v2	Supported; see *ECLIC Support* (page 866).
CIDU	Supported; see *CIDU Support* (page 866).
PMP	Supported.
TEE	Supported.
WFI	Supported.
WFE	Not supported.
ECC	ECC-related CSRs are implemented.
CCM	CCM-related CSRs are implemented.
SPMP	Not supported.
SMPU	Supported.
SMP / CLUSTER CACHE	Register-level access is supported; cache behavior is not modeled.
Big-endian	Supported; driven by the ELF header.
Semihosting	Supported; see *Semihosting* (page 874).

TIMER Support

The Nuclei SysTimer implements a shared time base, per-hart timer comparison and delivery, and multi-core ECLIC operation. EvalSoC also provides a CLINT-compatible timer model for conventional RISC-V software, and the `Sstc` supervisor timer extension is supported.

PLIC Support

The PLIC model supports platform-level interrupt prioritization and routing for machine-mode and supervisor-mode interrupt contexts, with up to 1024 interrupt sources. Multi-core PLIC mode is not supported.

ECLIC Support

ECLIC supports single-core and multi-core operation in both M-mode and S-mode. The ECLIC v1 model provides the basic interrupt configuration, arbitration, nesting and trigger behavior. ECLIC v2 additionally supports automatic context save/restore and shadow general-purpose register groups, together with the associated nested-trap state handling. The interrupt number count and `CLICINTCTLBITS` can be configured with `--eclic`.

CIDU Support

The CIDU distributes external interrupts among per-hart ECLIC instances. It also implements inter-core interrupts, multiple interrupt-distribution modes, and hardware semaphores for multicore software.

8.2.4 Nuclei SDK Cases Supported in xlmodel

See *Use Nuclei Model in Nuclei SDK* (page 875) for the case status table.

8.3 Nuclei Model Command-line Options

This section covers EvalSoC-specific settings and commonly used *xlmodel* options. Run `xl_cpumodel --help` for the full list.

8.3.1 Command-line Syntax and Help

Use `bin/xl_cpumodel` on Linux and `bin/xl_cpumodel.exe` on Windows, together with the ELF file of the test program:

```
xl_cpumodel [options] <elf-file>
```

The following parameter examples are based on a Linux system. If you are using a Windows system, simply replace **`xl_cpumodel`** with **`xl_cpumodel.exe`**.

Use the following commands to discover the options and cores available in the installed package:

```
xl_cpumodel --help
xl_cpumodel --version
xl_cpumodel --cpu=?
```

The default test code ELF files are provided in the `tests` directory of the package, for example:

```
./xl_cpumodel --cpu=nx900 --download=ilm ../tests/demotimer/demotimer_nx900.elf
```

8.3.2 CPU Configuration

`--cpu=<str>`: Selects the CPU model, defaults to `n300fd`. Use `--cpu=?` to list all supported Nuclei cores (rv32/rv64), or refer to the Supported Nuclei Processor Cores section[74] of the Nuclei SDK documentation.

`--ext=<str>`: Enables additional ISA extensions, separated by `_`, for example `--ext=_zba_zbb_zbc_zbs_xxldspn1x`. See *Supported ISA Extensions* (page 868) for the available extensions. The `Sstc` extension is enabled by default and does not need to be passed; an extension can also be disabled or re-enabled by appending `name=true/false` items, where later items take precedence, for example `--ext=_zba_zbb,sstc=false`. Example:

```
./xl_cpumodel --cpu=n300fd --download=ilm --ext=_zba_zbb_zbc_zbs_xxldspn1x ../
↪tests/demodsp/demodsp_n300fd.elf
```

`--bpu=<str>`: Selects the **BPU strategy** based on Nuclei core types, which affects the cycle count of branch and jump instructions. `--bpu=n300` is applicable to Nuclei cores up to and including N300, and `--bpu=n900` is applicable to Nuclei cores from N300 onwards:

```
./xl_cpumodel --cpu=n300fd --ext=_zba_zbb_zbc_zbs_xxldspn1x --download=ilm --
↪bpu=n300 ../tests/demodsp/demodsp_n300fd.elf

./xl_cpumodel --cpu=nx900 --download=ilm --bpu=n900 ../tests/demotimer/demotimer_
↪nx900.elf
```

`--priv=<str>`: Selects the implemented privilege modes `m`, `mu` or `msu`, defaulting to `msu`. Use `--priv=m` for M-mode only CPUs.

`--eclic=<str>`: Configures the ECLIC with `irqnums:<n>,intctlbits:<n>,shdnum:<n>`, defaulting to `irqnums:64,intctlbits:3,shdnum:9`. `irqnums` sets the total number of interrupt lines, `intctlbits` sets `CLICINTCTLBITS`, and `shdnum` sets the number of ECLIC v2 shadow register groups. Use it together with `--ext=_xlclicv2` to enable ECLIC v2:

```
./xl_cpumodel --cpu=n900 --download=ilm --ext=_xlclicv2 --eclic=irqnums:64,
↪intctlbits:3,shdnum:9 --soc-cfg=../tests/demoeclicstress_eclicv2_n900/cpu.json ..
↪/tests/demoeclicstress_eclicv2_n900/demoeclicstress.elf
```

`--varch=vlen:n,elen:n`: Sets the RISC-V Vector micro-architecture configuration when the V extension is enabled, defaulting to `vlen:128,elen:64`. **VLEN and ELEN must comply with the RISC-V Vector specifications.** Example:

```
./xl_cpumodel --cpu=ux900fd --ext=v --download=ilm --varch=vlen:128,elen:64 ../
↪tests/rvv_conv_f32/rvv_conv_f32.elf
./xl_cpumodel --cpu=ux900fd --ext=v --download=ilm --varch=vlen:512,elen:64 ../
↪tests/rvv_conv_f32/rvv_conv_f32.elf
```

VLEN values of 2048 and 4096 are also supported, which is useful for VNICE instruction modeling with very wide vector registers:

```
./xl_cpumodel --cpu=ux900fd --ext=v --download=ilm --varch=vlen:2048,elen:64 ../
↪tests/rvv_conv_f32/rvv_conv_f32.elf
```

`--smp=<n>`: Sets the SMP system core number, with a maximum of 16. If this parameter is not set, 1 CPU is used. Example:

```
./xl_cpumodel --cpu=nx900 --download=sram --smp=4 ../tests/smphello_4core/smphello_
↪nx900.elf
```

[74] https://doc.nucleisys.com/nuclei_sdk/develop/buildsystem.html#core

Supported ISA Extensions

The following extensions can be selected through `--ext=`. Choose extensions supported by the selected CPU configuration and the guest program.

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)
_zimop	RISC-V Zimop Extension
_zcmop	RISC-V Zcmop Extension
_zclsd	Zclsd extension (RV32 ONLY); use `_zcmlsd` in xlmodel
_zawrs	Zawrs extension
_zacas	Zacas extension (AMO CAS)
_zfh	Zfh Extension
_zfa	Zfa Extension
_zfhmin	Zfhmin Extension
_zfbfmin	Scalar BF16 conversions (Zfbfmin)
_zfinx	Zfinx Extension
_zdinx	Zdinx Extension
_zibi	RISC-V Zibi Extension
_zicfilp	RISC-V Zicfilp Extension (landing pad)
_zca	RISC-V ZC* Extension
_zcb	RISC-V ZC* Extension
_zcf	RISC-V ZC* Extension
_zcd	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
_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

continues on next page

Table 8.3 – continued from previous page

Extension	Functionality
_zvfh	RISC-V V Extension
_zvfhmin	RISC-V V Extension
_zvfbfmin	Vector BF16 conversions, V extension required
_zvfbfwma	Vector BF16 multiply-accumulate, V extension required
_zvkg	RISC-V Vector Crypto Extension
_zvkned	RISC-V Vector Crypto Extension
_zvksed	RISC-V Vector Crypto Extension
_zvksh	RISC-V Vector Crypto Extension
zhinx	Zhinx Extension
zhinxmin	Zhinxmin Extension
smaia	Smaia Extension
_smrnmi	Smrnmi Extension
ssaia	Ssaia Extension
_sscofpmf	Sscofpmf Extension
_sstc	Sstc Extension (enabled by default)
_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
_xxlvqmacc	Nuclei custom VPU extension; use `_xlvqmacc` in xlmodel
_xxlfbf	Nuclei scalar BF16 extension; not enabled via `--ext` in xlmodel: set the `fp16mode` bit of the `MMISC_CTL1` CSR
_xxlvfbf	Nuclei vector BF16 extension; not enabled via `--ext` in xlmodel: set the `fp16mode` bit of the `MMISC_CTL1` CSR
_xxlclicv2	Nuclei ECLIC v2 extension (use `_xlclicv2` in xlmodel)
_agnostic	Nuclei vector tail/mask-agnostic policy: fill ones (V required)

8.3.3 SoC Configuration

`-M/--machine=<str>`: Machine type config, defaults to `nuclei_evalsoc`.

`--download=<str>`: Selects the download mode; the default is `flashxip` and it can be set to `ilm`, `ddr`, `sram` or `flash`. The startup address of the selected mode is the EvalSoC memory base by default, and can be overridden per mode through `soc-cfg` or globally with `--vectorbase`. Example:

```
./xl_cpumodel --cpu=nx900 --download=sram --smp=4 ../tests/democidu_en_firclaim/
↪democidu_en_firclaim_nx900.elf
```

`--soc-cfg=<file.json>`: Overrides the built-in board configuration with a JSON file. See *soc-cfg JSON Configuration* (page 870).

`--mem=<str>`: Defines additional user memory regions when the test code uses external memory sections, for example `--mem="0x30000000:0x10000000,0x40000000:0x10000000"`. Example:

```
./xl_cpumodel --cpu=nx900 --download=ilm --mem="0x30000000:0x10000000,
↪0x40000000:0x10000000" ../tests/demoeclic_swirq_high/demoeclic_swirq_high_nx900.
↪elf
```

`--vectorbase=<addr>`: Sets the reset vector address explicitly. It takes precedence over the startup address selected by `--download`, including the `download.<mode>.startaddr` item of the `--soc-cfg` file and the default memory base. When it is set, the consistency check between the ELF entry point and the selected download mode is skipped.

8.3.4 soc-cfg JSON Configuration

The `--soc-cfg=<file.json>` option loads a JSON file that overrides the built-in EvalSoC board configuration, aligned with the `soc-cfg` format used by Nuclei QEMU. Without `--soc-cfg`, *xlmodel* uses the built-in default layout. For any configurable item, the effective value is resolved in the order **CLI parameter > ``soc-cfg`` JSON > built-in default**.

The following example overrides memory regions, selected peripherals and download-mode startup addresses:

```
{
    "general_config": {
        "ddr": {
            "base": "0x80000000",
            "size": "64M"
        },
        "ilm": {
            "base": "0x80000000",
            "size": "0x800000"
        },
        "dlm": {
            "base": "0x90000000",
            "size": "0x800000"
        },
        "sram": {
            "base": "0xA0000000",
            "size": "512M"
        },
        "norflash": {
            "base": "0x20000000",
            "size": "256M"
        },
        "iregion": {
            "base": "0x18000000",
            "eclic": "1",
            "smpcc": "1",
            "cidu": "1",
            "plic": "1"
        },
        "uart0": {
            "base": "0x10013000",
            "size": "0x1000",
            "irq": "32",
            "enable": "1"
        },
        "qspi0": {
            "base": "0x10014000",
            "size": "0x1000",
            "irq": "34",
            "enable": "1"
        },
        "timer_freq": "32768",
        "cpu_freq": "16000000"
    },
    "download": {
        "ilm": {
            "startaddr": "0x80000000"
        },
        "flashxip": {
            "startaddr": "0x20000000"
        },
        "flash": {
            "startaddr": "0x20000000"
```

(continues on next page)

(continued from previous page)

```
        },
        "sram": {
            "startaddr": "0xA0000000"
        },
        "ddr": {
            "startaddr": "0x80000000"
        }
    }
}
```

Table 8.4: `soc-cfg` fields supported by xlmodel

Field	Applies to objects	Description
`base`, `size`	`ilm`, `dlm`, `sram`, `ddr`, `norflash`, `uart0`, `qspi0`, `iregion`	Sets the base address and size of the corresponding memory or MMIO region.
`irq`	`uart0`, `qspi0`	Sets the peripheral interrupt ID routed to ECLIC/CIDU.
`enable`	`uart0`, `qspi0`	Enables or disables the corresponding device.
`eclic`, `smpcc`, `cidu`, `plic`	`iregion`	Enable or disable the corresponding IREGION block.
`timer_freq`	`general_config`	Sets the SysTimer clock frequency; overridden by `--timefreq`.
`cpu_freq`	`general_config`	Sets the CPU frequency; overridden by `--cpufreq`.
`startaddr`	`download.ilm`, `download.ddr`, `download.sram`, `download.flashxip`, `download.flash`	Sets the firmware startup address for the corresponding download mode.

Notes on the JSON format:

- Numeric leaf values can be written as decimal strings, `0x`-prefixed hexadecimal strings, or with a `K`/`M`/`G` size suffix; plain JSON numbers and `true`/`false` are also accepted.
- Partial overrides are supported: omitted objects and fields keep their defaults.
- `download.<mode>.startaddr` only changes the startup address of that download mode; it does not change the memory-region mapping.

An example of running with a `soc-cfg` file:

```
./xl_cpumodel --cpu=nx900 --download=ilm --soc-cfg=soc.json ../tests/demotimer/
↪demotimer_nx900.elf
```

8.3.5 Memory Configuration

The base address and size of each memory region can be adjusted with the following options; they override the same items in `soc-cfg`:

- `--ilmbase` / `--ilmsz`: ILM base address and size
- `--dlmbase` / `--dlmsz`: DLM base address and size
- `--srambase` / `--sramsz`: SRAM base address and size
- `--ddrbase` / `--ddrsz`: DDR base address and size
- `--flashxipbase` / `--flashxipsz`: XIP (NOR flash) base address and size

- `--iregionbase`: IREGION base address

For example, enlarging DDR to 2 GB when booting Linux:

```
./xl_cpumodel --cpu=ux900fd --download=ddr --ddrbase=0x80000000 --ddrsz=0x80000000
↪<linux-image.elf>
```

8.3.6 Simulation Speed and Time

`--cpufreq=<n>`: Simulated CPU frequency in Hz (default 655360). It takes precedence over the `cpu_freq` item of the `--soc-cfg` file. The frequency does not change the per-instruction cycle estimates: it only defines the cycle-to-second conversion and the mtime tick interval (one tick every `cpu_freq / timer_freq` cycles). Example:

```
./xl_cpumodel --cpu=nx900 --download=ilm --cpufreq=16000000 ../tests/demotimer/
↪demotimer_nx900.elf
```

Note: Set `--cpufreq` to the real board frequency when comparing simulated timings against hardware, and keep the default for functional runs. A higher value does not improve cycle accuracy: timer-driven workloads (RTOS ticks, busy-wait delays) retire instructions proportionally to `cpu_freq / timer_freq` while waiting, so raising it slows the simulation proportionally —at 50 MHz demotimer retires ~76x more instructions than at the default and takes minutes instead of seconds.

`--timefreq=<n>`: Sets the timer frequency in Hz, defaulting to 32768 Hz.

`--timeout=<n>`: Sets the expected real execution time in seconds for a simulation run; otherwise, it is unlimited. **This parameter is ignored during GDB debugging.** When the timeout is reached, the enabled gprof/flame graph files are generated. Example:

```
./xl_cpumodel --cpu=nx900 --download=sram --smp=2 --gprof=1 --flamegraph=1 --
↪timeout=3 ../tests/freertos_smpdemo/freertos_smpdemo_nx900.elf
```

Pressing `Ctrl+C` also stops the simulation gracefully, and similarly generates gprof/flame graph files for the period from startup to shutdown.

`--funcmode=1`: Every retired instruction is counted as a fixed 1 cycle and the per-instruction near-cycle calculation is skipped. Trace, gprof and flame graph are still available in this mode.

`--fastmode=1`: Switches to the fast functional mode to maximize simulation speed. The functional behavior (instruction execution, peripherals, interrupts) is unchanged, so use it when the exact cycle count is not required, such as booting Linux or running very long functional tests. Cycle-related features such as `--trace`, `--gprof` and `--flamegraph` are not available in this mode:

```
./xl_cpumodel ../tests/demodsp/demodsp_n300fd.elf --cpu=n300fd --ext=_zba_zbb_zbc_
↪zbs_xxldspn1x --download=ilm --fastmode=1
```

8.3.7 Trace, Profiling and Logs

`--trace=1`: Generates the instruction trace file `<elf-name>.rvtrace` in the current execution path. It contains the instruction count, cycle count, hart, pc, opcode and disassembly of each instruction. Example:

```
./xl_cpumodel --cpu=nx900 --download=ilm --trace=1 ../tests/demoeclic_swirq_low/
↪demoeclic_swirq_low_nx900.elf
```

`-t <path>`: Specifies the trace file path directly and enables tracing at the same time.

`--gprof=1`: Enables the built-in **gprof** functionality. The `gprof<n>.gmon` and `gprof<n>.log` files are generated when the simulation completes, where `n` is the hart ID. Import them into Nuclei Studio for detailed analysis. Example:

```
./xl_cpumodel --cpu=nx900fd --download=ilm --gprof=1 ../tests/whet/whet_nx900fd.elf
```

`--flamegraph=1`: Enables the built-in **flame graph** functionality. The `xlmodel_flame_<n>.gtef` files are generated when the simulation completes, where `n` is the hart ID. For each hart, the flame graph shows the main thread, RTOS task threads, and M-mode/S-mode interrupt threads. Import them into Nuclei Studio, see *Nuclei Studio use xlmodel flame graph* (page 296) for detailed usage. Example:

```
./xl_cpumodel --cpu=n300fd --download=ilm --flamegraph=1 ../tests/demo_clint_timer/
↪demo_clint_timer_n300fd.elf
```

`--logdir=<str>`: The directory to store generated files (instruction trace, gprof files, flame graphs). Example:

```
./xl_cpumodel --cpu=nx900 --download=ilm --trace=1 --gprof=1 --flamegraph=1 --
↪logdir=../log ../tests/demoeclic_swirq_low/demoeclic_swirq_low_nx900.elf
```

`--log=<str>`: Sets the log level, from least to most detailed: `error`, `info`, `debug`, `tlm`. The default is `info`, and `--log=debug --trace=1` can adds register updates, exceptions and CSRs to the trace file. Example:

```
./xl_cpumodel --cpu=nx900 --download=ilm --trace=1 --log=debug ../tests/demotimer/
↪demotimer_nx900.elf
```

8.4 DEBUG

You can debug applications running with *xlmodel* by passing the `--gdbport=<port_num>` parameter. This enables the GDB server and allows external GDB (e.g., `riscv64-unknown-elf-gdb`) to connect via the specified portnum for software debugging.

To debug via the command line, follow these steps on a Linux system: (Windows is similar)

1. Start *xlmodel* with GDB debugging enabled:

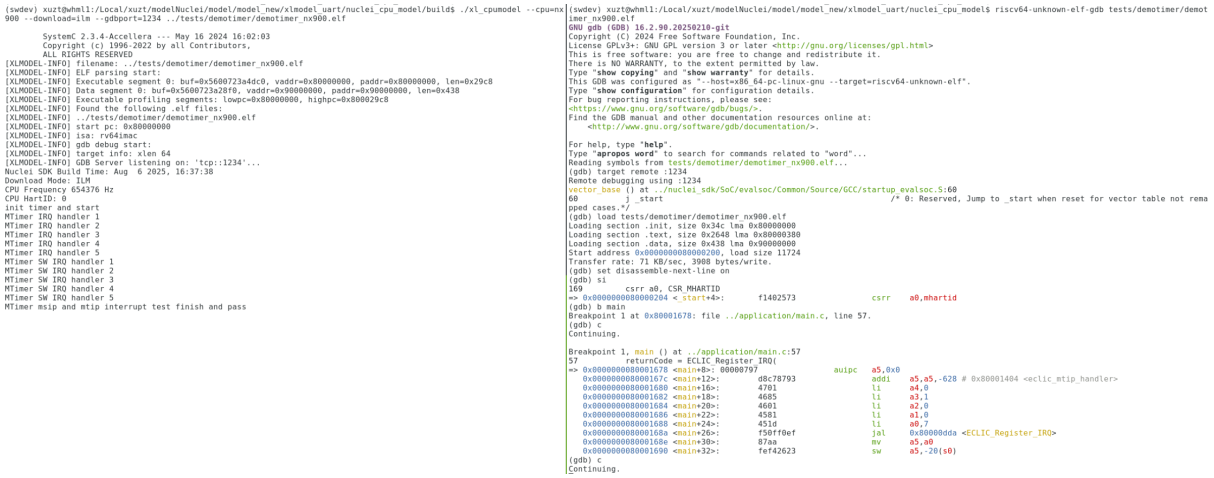
   ```
   ./xl_cpumodel --cpu=nx900 --download=ilm --gdbport=1234 ../tests/demotimer/
   ↪demotimer_nx900.elf
   ```

2. Open another command-line terminal and run GDB:

   ```
   riscv64-unknown-elf-gdb tests/demotimer/demotimer_nx900.elf
   ```

3. Connect to the GDB port, load the ELF file, and perform standard GDB debugging:

   ```
   target remote :1234
   load tests/demotimer/demotimer_nx900.elf
   set disassemble-next-line on
   info register all
   si
   b xxx
   c
   ```



The GDB stub supports:

- **SMP debugging**: `info threads` lists every hart of the cluster; `thread <n>` switches the target hart and `si` single-steps only the selected hart, while `continue` runs the whole cluster.
- **Breakpoints and watchpoints**: software breakpoints (`b`), hardware breakpoints (`hbreak`), and watchpoints (`watch`/`rwatch`/`awatch`) on physical or virtual addresses. Stop replies report the stop reason, e.g. `swbreak`, `hwbreak`, `watch`, `rwatch` or `awatch`.
- **Memory access after MMU is enabled**: addresses are translated from VA to PA automatically. Use `monitor pa on` / `monitor pa off` to force the GDB stub to access physical addresses directly.

For an easier experience using GDB debugging with *xlmodel*, we recommend using Nuclei Studio. When GDB is enabled, passing `--gprof=1` and `--flamegraph=1` allows you to capture profiling result and generated flame graph visualizations while the program is suspended in GDB.

For more information on using *xlmodel* debugging in Nuclei Studio, please refer to the *Using Nuclei Model Debugger* (page 291).

Note: The `--timeout=<n>` parameter has no effect when software debugging is enabled.

8.5 Semihosting

xlmodel supports the RISC-V semihosting convention: when `--semihosting=1` is passed, an `EBREAK` instruction surrounded by the semihosting magic words traps into the host and executes the requested semihosting call, with the operation number in register `a0` and the parameter block pointer in `a1`. Semihosting calls are executed by the host directly and bypass the UART.

Supported operations include basic printing (`SYS_WRITE0`/`SYS_WRITEC`), file I/O (`SYS_OPEN`, `SYS_CLOSE`, `SYS_READ`, `SYS_WRITE`, `SYS_FLEN`, `SYS_SEEK`), and program exit (`SYS_EXIT`).

Build the guest program with semihosting enabled (for example `--specs=semihost.specs` or the SDK semihosting support), then run:

```
./xl_cpumodel --cpu=nx900 --download=ilm --semihosting=1 <semihosting-enabled.elf>
```

8.6 Use Nuclei Model in Nuclei SDK

Setup Tools and Environment

1. Download the nuclei-sdk[75], checkout to `master` branch.
2. Download RISC-V GNU Toolchain from Nuclei Download Center[76].
3. Download the *xlmodel* software package from Nuclei Download Center[77] (or obtain it in Nuclei Studio), extract it and add `xlmodel/bin` to the `PATH` environment variable.

Example

If you want to use *xlmodel* on Nuclei SDK. The example here uses the CPU of the n300fd, but other CPU types can also be used for testing. The example is demo_dsp.

First, you need to configure the toolchain, nuclei-sdk, and xlmodel environments according to the documentation https://doc.nucleisys.com/nuclei_sdk/quickstart.html

```
# Enter the example folder of demo_dsp
cd nuclei-sdk/application/baremetal/demo_dsp/
# Clear the compilation cache
make clean
# Compile the program for the n300fd, set the download mode to ILM, and enable the␣
↪xxldsp extension
make CORE=n300fd SOC=evalsoc DOWNLOAD=ilm ARCH_EXT=_xxldsp dasm
# Automatically generate xlmodel running commands and execute the program
make CORE=n300fd SOC=evalsoc DOWNLOAD=ilm ARCH_EXT=_xxldsp run_xlmodel
```

Where **ARCH_EXT** can be used to pass the extension name. The `run_xlmodel` target calls `xl_cpumodel` with the matching `--cpu`, `--download` and `--ext` options (and `--smp` when `CPU_CNT` is set), so no extra *xlmodel* options are needed and make sure `xl_cpumodel` is in the `PATH`. Under normal circumstances, you should see the test cases finish successfully.

Support for Nuclei SDK Cases on xlmodel

`Y` - Successfully run and consistent with hardware

`N` - Successfully run but inconsistent with hardware

`F` - Failed

Cases	SMP=1	SMP>1	Description (additional compilation parameters and running status)
benchmark/coremark	Y		
benchmark/dhrystone	Y		
benchmark/ dhrystone_v2.2	Y		
benchmark/whetstone	Y		
benchmark/ whetstone_v1.2	Y		
cpuinfo/	N		
demo_backtrace/	Y		
demo_cache/	F		Cache emulation is not supported.
demo_cidu/		Y	SMP=4, XLCFG_CIDU=1.
demo_clint_timer/	Y		
demo_dsp/	Y		
demo_ecc/	N		ECC error injection and correction behavior is not modeled.
demo_eclic/	Y		
demo_eclic_stress/	Y		XLCFG_ECLIC=2, ECLIC_HWCTX=0/1; run with an increased `--cpufr`
demo_eclic_umode/	Y		

continue

[75] https://github.com/Nuclei-Software/nuclei-sdk
[76] https://nucleisys.com/download.php
[77] https://nucleisys.com/download.php

Table 8.5 – continued from previous page

Cases	SMP=1	SMP>1	Description (additional compilation parameters and running status)
demo_nice/	Y		
demo_plic/	Y		XLCFG_PLIC=1.
demo_pma/	N		XLCFG_CCM=1, XLCFG_HPM=1.
demo_pmp/	Y		
demo_profiling/	N		xlmodel already implements its own profiling.
demo_smode_clint/	Y		
demo_smode_eclic/	Y		XLCFG_TEE=1.
demo_smode_plic/	Y		XLCFG_PLIC=1.
demo_smpcc/	N		Control registers are modeled; cluster-cache behavior is not modeled.
demo_smpu/	Y		XLCFG_SMPU=1, CHK_MODE=0 through 5.
demo_sstc/	Y		XLCFG_TEE=1, XLCFG_SSTC=1.
demo_stack_check/	N		Only CSRs are implemented.
demo_timer/	Y		
demo_vnice/	Y		
empty/	Y		
exception_mmode/	Y		
exception_smode/	Y		XLCFG_TEE=1.
exception_umode/	Y		XLCFG_SMODE=1, XLCFG_SMPU=1.
helloworld/	Y		
lowpower/	Y		
smphello/		Y	SMP=4.
freertos/demo/	Y		
freertos/smpdemo/		Y	SMP=4.
rtthread/demo/	Y		
rtthread/demo_smode/	Y		XLCFG_TEE=1, XLCFG_SSTC=1.
rtthread/msh/	Y		
threadx/demo/	Y		
threadx/smpdemo/		Y	SMP=4.
ucosii/demo/	Y		

And *xlmodel* and Nuclei SDK are deeply integrated in Nuclei Studio, you can run Nuclei SDK test code using *xlmodel* in Nuclei Studio, please refer to the *Nuclei Studio use xlmodel* (page 300).

8.7 Use Nuclei Model in Nuclei Linux SDK

xlmodel can boot and test a RISC-V Linux Kernel using the emulated Nuclei EvalSoC, in both uniprocessor and SMP mode, for RV64 and RV32. For more information about the Nuclei Linux SDK, see https://github.com/Nuclei-Software/nuclei-linux-sdk .

Prebuilt boot images built from the Linux SDK are provided in the `tests` directory of the *xlmodel* package:

Table 8.6: Prebuilt Linux boot images

File	Arch	Description
`tests/boot_linux_rv64/fw_payload_xlspike_ux900fd_flash_job_elf`	RV64	Linux 6.6 flash payload for ux900fd from the Linux SDK CI.
`tests/boot_linux_rv32/fw_payload_xlspike_u900_flash_job_5c2elf`	RV32	Linux 6.6 flash payload for u900 from the Linux SDK CI.

An example of booting Linux on a single core:

```
./xl_cpumodel --cpu=ux900fd --download=ddr \
    --ext=_svpbmt_zicbom_sstc_sscofpmf_zba_zbb_zbc_zbs_zicond \
    --fastmode=1 --ddrbase=0x80000000 --ddrsz=0x80000000 \
    --timeout=3600 ../tests/boot_linux_rv64/fw_payload_xlspike_ux900fd_flash_job_
↪5c231cec.elf
```

An example of booting Linux on two cores (SMP): add `--smp=2` and disable ILM with `--ilmsz=0`, because the SMP image uses a DDR-only memory map and the default ILM overlaps the enlarged DDR region:

```
./xl_cpumodel --cpu=ux900fd --download=ddr \
    --ext=_svpbmt_zicbom_sstc_sscofpmf_zba_zbb_zbc_zbs_zicond \
    --fastmode=1 --smp=2 --ilmsz=0 --ddrbase=0x80000000 --ddrsz=0x80000000 \
    --timeout=3600 ../tests/boot_linux_rv64/fw_payload_xlspike_ux900fd_flash_job_
↪5c231cec.elf
```

The RV32 image boots in the same way with `--cpu=u900` and the `tests/boot_linux_rv32` image.

Note: Booting Linux requires a large DDR (2 GB in the examples above, via `--ddrsz`), and `--fastmode=1` is recommended. Tracing and profiling options are not available in fast mode.

8.8 NICE support

8.8.1 NICE build

If you need to validate your custom **NICE** instructions, you need to contact Nuclei Support to obtain software package of model (*xlmodel_nice*).

The directory structure of *xlmodel_nice* is as follows, you need to implement the **NICE** instructions in *nice/nice.cc*.

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	simple test codes
CMakeLists.txt	CMake file required for compilation

After implementing the **NICE** instruction, you need to recompile *xlmodel_nice*.

nice build for Linux

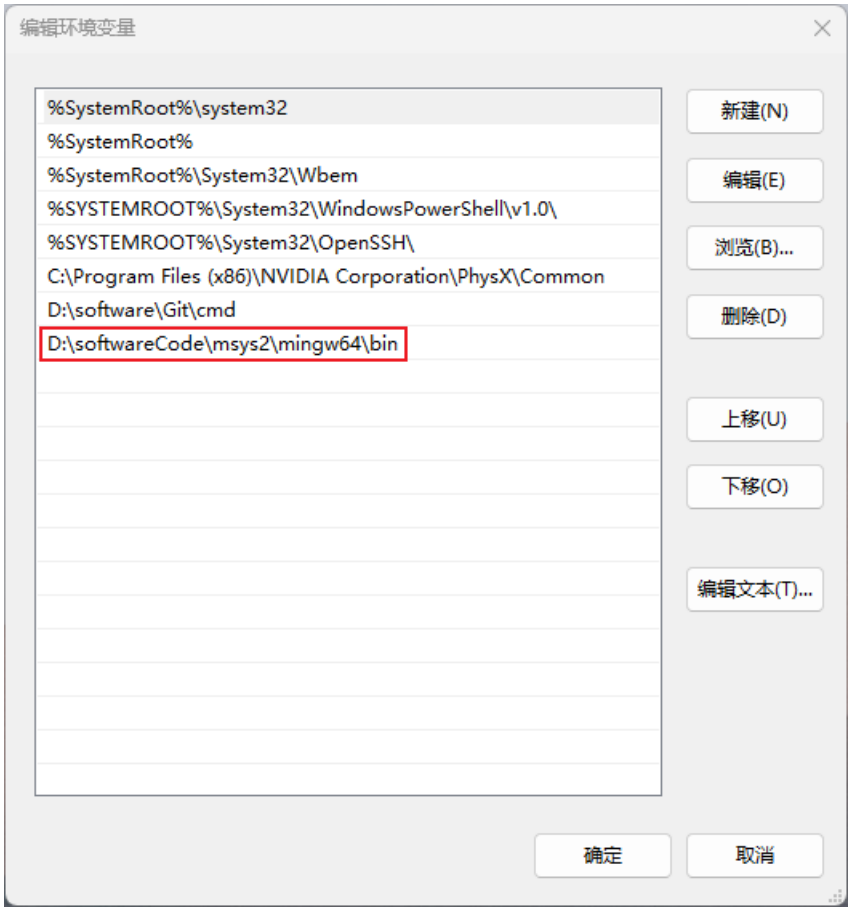
```
# Install essential compilation tools
sudo apt install build-essential cmake
# Check the tools have been installed successfully
gcc -v && g++ -v && cmake --version
# Change to the root directory of the xlmodel_nice package
cd <xlmodel_nice root directory>
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make -j$(nproc)
```

nice build for Windows

To compile *xlmodel_nice* on Windows, you need to download a Windows-compatible GCC tool, such as **MinGW64**. You can download **MSYS2** to easily obtain the MinGW64 toolchain, which simplifies the installation and management of MinGW64 on Windows.

Below are the steps to use MSYS2 ' s MinGW64 to compile *xlmodel_nice* on Windows:

1. Install the latest version of MSYS2 from https://www.msys2.org/, and then add the MinGW64 toolchain path to the **environment variables**:



2. Install MinGW64 toolchain, CMake, and other basic compilation tools in the **MSYS2 terminal**:

```
# Sync the package databases first, rerun until it prints "there is nothing to␣
↪do"
pacman -Syu
pacman -S base-devel mingw-w64-x86_64-gcc mingw-w64-x86_64-cmake
```

3. Compile *xlmodel_nice* in the **MinGW64 terminal**:

```
# Check the tools have been installed successfully
gcc -v && g++ -v && cmake --version
# Change to the root directory of the xlmodel_nice package
cd <xlmodel_nice root directory>
mkdir build && cd build
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release ..
make -j$(nproc)
```

Note: If you need to import and compile *xlmodel_nice* in Nuclei Studio (*Nuclei Studio use xlmodel* (page 300)), you also need to configure the **MSYS2** environment as described above first.

8.8.2 NICE example

If you are unfamiliar with how to implement the NICE instruction, refer to the implementation in *xlmodel_nice/nice/nice.cc* for custom **Nuclei NICE/VNICE** instructions.

The test example for Nuclei NICE instruction features are located in *tests/demonice*:

```
./xl_cpumodel --cpu=nx900 --download=ilm ../tests/demonice/demonice_nx900.elf
```

The test example for Nuclei VNICE instruction features are located in *tests/demovnice*:

```
./xl_cpumodel --cpu=nx900fd --ext=v --download=ilm ../tests/demovnice/demovnice_
↪nx900fd.elf
```

8.9 SystemC integration

If you need to integrate *xlmodel* into a third-party SoC simulation platform, you need to contact Nuclei AE for the software integration package of the model (*xlmodel_integrate*).

xlmodel_integrate mainly includes the source code of *SystemC components* (page 864). You can extract the parts you need and use them on third-party platforms with relatively little adaptation.

The integration package is delivered per target platform. The standard package is built against **SystemC 2.3.4** and is available for several Linux distributions (linux64, Debian 10, CentOS 7) and Windows (win32); for Ubuntu hosts, additional variants built against **SystemC 3.0.0 and SystemC 3.0.2** are provided, so the package can be linked into SystemC 3.x based platforms as well.

Table 8.7: Main directory structure of `xlmodel_integrate`

directory	description
src	All SystemC components, Cluster and Timer are mandatory to integrate, while others can be integrated as needed.
inc	All header files related to xlmodel, which are integrated as needed.
systemc	SystemC 2.3.4 header files and the platform-specific static library (`library/<platform>/libsystemc.a`).
cycle	The near cycle lib file, must be integrated.
profiling	The xlmodel profiling lib file, which is integrated as needed.
xlspike	xlspike header files and the platform-specific static library (`library/<platform>/libxl_spike.a`), must be fully integrated.
tests	simple test codes.
nice	header and source files for the NICE interface, must be fully integrated.
CMake-Lists.txt	CMake file required for compilation.

Here are the integration steps instructions:

1. Find the Cluster class in *inc/cluster.h*:
   - The Cluster class contains two types of variables:
     - **Internal variables**: Initialized to default values in the constructor ' s member initialization list.
     - **User-defined variables**: Initialization in the constructor ' s member list depends on your **firmware, CPU configurations and extra simulation feature needs**.
   - Usage recommendations:
     - You can inherit the Cluster class in your SystemC platform class to use base methods of your platform.
2. Find the Cluster class in *src/cluster.cpp*:

- Usage recommendations:
  - If you do not need to replace the SystemC sockets and TLM methods in the Cluster class, you can directly use the constructor and methods in the Cluster class without any modification.
  - Adjust the user-defined variables in the constructor ' s member initialization list.
  - If your platform implements TLM transmission from CPU to Bus, you can substitute the read/write implementations into `cpu_read_tlm()` and `cpu_write_tlm()`.

3. Use External Interrupts:
   - To connect User-defined external interrupts to the CPU:
     - Pass the number of external interrupt sources to the Cluster constructor. It decides the size of the Cluster ' s `irq_sockets`, and each source is bound to one socket.
     - The total number of interrupt lines is configured with `irq_num`. An external interrupt ID must be smaller than `irq_num` minus the built-in interrupt offset (19 for ECLIC, 1 for PLIC).
     - Bind your device ' s interrupt socket to one of the Cluster ' s `irq_sockets`.

8.10 Changelog

Note: To check the known issues of the current release, see *Known Issues* (page 882).

8.10.1 Version 2026.10

This release is based on **2025.10** version with some new features and bug fixes introduced.

New Features:

- Support booting a RISC-V Linux kernel with the Nuclei Linux SDK (RV64/RV32, uniprocessor and SMP)
- SMP GDB debugging: per-hart thread switching, single-step on the selected hart, watchpoints (watch/rwatch/awatch) on virtual addresses, stop-reason reporting
- Support ECLIC v2 hardware context and shadow register groups (`--ext=_xlclicv2`); add `--eclic=irqnums:<n>,intctlbits:<n>,shdnum:<n>`
- Support for vector tail/mask-agnostic policies via `--ext=v,agnostic=true` (fill ones) and `--ext=v,noagnostic=true` (keep old values).
- ECLIC interrupt sources scale to 4096(M-only) or 1024(M/ S), and Privilege can be configured by `--priv=m/mu/msu` options; PLIC interrupt sources scale to 1024
- Support big-endian RISC-V ELF files; target endianness is detected from the ELF header, including big-endian GDB debugging
- SMP simulation now supports up to 16 cores (previously limited to 8)
- Support RISC-V semihosting (`--semihosting=1`) for guest printf/file-I/O/exit via EBREAK
- Support `soc-cfg` JSON configuration files (`--soc-cfg`), aligned with the Nuclei QEMU format, with CLI > soc-cfg > default priority
- Add `--fastmode=1` for maximum functional simulation speed
- Support the `Sstc` supervisor timer extension by default
- Faster flamegraph `.gtef` generation with improved symbol resolution

- Add new extensions: Zibi, Zicfilp, Zacas, Zaamo/Zalrsc, vector crypto (Zvkg/Zvkned/Zvksed/Zvksh), BF16 support: standard extensions (`Zfbfmin`/`Zvfbfmin`/`Zvfbfwma`) and Nuclei `Xxlfbf`/`Xxlvfbf` (enabled via the `MMISC_CTL1 fp16mode` bit); Nuclei `Xxlvqmacc` VPU extension (use `_xlvqmacc`); DSP extensions differentiated into Xxldspn1x/N2x/N3x; VLEN 2048/4096 support
- UART: RX polling mode: guest can poll RXDATA without enabling the UART interrupt
- Add `--cpu=?` core listing, `--timefreq`, `--srambase`/`--sramsz` and `--vectorbase` options; `--ext` supports per-extension `name=true/false` overrides; `--version` output now includes the git commit hash
- Support pure-BSS ELF loading
- The integrate package now supports multiple Linux distributions (linux64, Debian 10, CentOS 7) and provides SystemC 3.0.0/3.0.2 variants on Ubuntu, in addition to the SystemC 2.3.4 based standard package

Bug Fixes:

- Fixed memory issues reported by AddressSanitizer, including incomplete spike-side teardown on model exit and several uninitialized variables
- Fixed ECLIC pending-bit handling that caused continuous interrupt re-triggering, ECLIC re-arbitration after configuration register writes, edge-trigger handling and S-mode software interrupt issues
- Fixed PLIC interrupts not being triggered when enabling an already-pending source or setting a pending bit of an already-enabled source
- Registered the SSCRATCHCSW/SSCRATCHCSWL CSRs, fixing illegal-instruction errors in S-mode RT-Thread demos
- Fixed lr/sc reservation invalidation across harts in SMP
- Fixed the mtime advance model in Nearcycle/func mode (exact cycle-proportional Bresenham ratio instead of a floored ratio)
- Fixed PMP fetch/load/store checks, instruction access fault reporting and page-fault exception injection
- Fixed VNICE masked instructions not following the vector tail/mask-agnostic policy, and user-defined VNICE instructions being wrongly masked by `v0` when bit 25 of the encoding is 0
- Fixed GDB connection issues: oversized packets that were not split and reassembled could terminate the GDB session, and bytes left over from an abnormally disconnected client corrupted the next client ' s handshake
- Fixed a spurious GDB server error (*recv error: No error* on Windows) reported when the GDB client is terminated without detaching, e.g. when a debug session is stopped from Nuclei Studio
- Fixed flame graph issues: RTOS tasks migrating across SMP cores were shown with duplicated thread identities, M-mode/S-mode interrupt handlers were attributed to the wrong interrupt thread lane, and frames of nested interrupt handlers were handled incorrectly across cross-mode preemptions, leaving orphaned frames
- Fixed Ctrl+C not stopping the model properly in GDB mode; it now works in all modes and dumps the collected gprof and flame graph results on exit
- Fixed UART issues: the interrupt output now uses transition-detected levels and the pending status is updated while the TX FIFO drains; interactive stdin input was restored and is fed to the RX FIFO regardless of the guest interrupt-enable state

Known Issues

- Does not support CACHE emulation
- Two ThreadX SMP test cases livelock on nx900fd: `threadx_event_flag_isr_set_clear_test` and `threadx_event_flag_isr_wait_abort_test`. The timer tick lands in a window where the test ISR exit conditions never hold, so hart 0 spins inside `_tx_thread_smp_protect` without releasing the lock and the other harts starve. Both cases pass on n900fd, and the issue is sensitive to timing-model changes

8.10.2 Version 2025.10

This release is based on **2025.02** version with some new features and bug fixes introduced.

- Support GDB debugging
- Support flame graph
- Add the CIDU module
- Add Nuclei systimer support in CLINT mode
- Support SMP ECLIC, can run smpdemo of FreeRTOS
- Add new cpu support for the Nuclei n300e and 1000 series
- Add zfinx/zhinx/zhinxmin/zdinx/zimop/zcmop/smaia/ssaia extensions
- Add hypervisor aia
- The base address and size of memory resources can be configured
- Support for setting the CPU frequency
- Output is separated into stdout and stderr

Known Issues

Note: The status of each issue in the 2026.10 release is annotated below.

- Does not support CACHE emulation.

  Note: Still open in 2026.10.

- Does not support booting a Linux Kernel.

  Note: Fixed in 2026.10, which supports booting a RISC-V Linux kernel in both uniprocessor and SMP mode.

- Some SMP cases for RT-Thread and Zephyr are not yet running correctly.

  Note: 2026.10 fixes the illegal-instruction error in S-mode RT-Thread demos, but the SMP test cases have not been re-verified for this release.

- When debugging with Nuclei Studio, if the ELF file lacks DWARF debug information, you need to add the `--mem=0x0:0x10000000` option as a workaround to prevent GDB from reporting memory read errors.

8.10.3 Version 2025.02

Known Issues

- Does not support smp ECLIC, CIDU and CACHE emulation

New Features:

- Nuclei 200/300/900/1000 are supported, and support cpu core name are aligned with Nuclei SDK 0.7.1 release.
- Added support for Windows system, now both windows and linux system are supported.
- Added important parameters such as `--cpu`, `--ext`, and `--mem`, and changed the `--time` parameter to `--timeout`
- Added rough cycle calculation for vector instructions

Improvements:

- Improved the CPU, Timer, and UART models
- Improved the *xlmodel_nice* software package
- Resolved the issue where some instructions could not be profiled
- Adapted to run *demo_vnice*, *demo_plic*, and multiple RTOS cases in nuclei-sdk

CHAPTER

NINE

CHANGELOG

Nuclei Tools official releases can be found in https://nucleisys.com/download.php#tools, you can download it from there.

You can also find our **Nuclei Studio Supply Documents** in https://doc.nucleisys.com/nuclei_studio_supply/, which is used for application notes using Nuclei Studio and Nuclei Tools.

9.1 2026.10

This is **2026.10** release of Nuclei Tools.

Note:

- This release is mostly based on 2025.10, the toolchain components such as GCC, Binutils, GDB, Newlib, LLVM and GLIBC keep the same versions.
- The **Nuclei Studio VSCode** extension is released for the first time in this release, its version is **0.3.0**.
- RISC-V **big-endian** target support is added in Nuclei QEMU and Nuclei Model, while the big-endian toolchain is not released yet.

Attention: Upgrade notes. The following changes may silently change the behavior of existing projects, please check your build options after upgrading:

- The `Zcmt` table jump auto-generation is now disabled by default. Pass `-Wl,--relax-zcmt` to the linker to enable it again, see *the Zcmt note* (page 427).
- The Nuclei DSP auto-vectorization (`-mautovec-dsp`) and the Nuclei `xxlcz` auto-generation (`-mautogen-xxlcz`) are disabled by default now, pass the corresponding option to enable them again.
- The `-maddibne/-mno-addibne` option is removed, use `-mautogen-xxlcz/-mno-autogen-xxlcz` instead. Build scripts still passing `-maddibne` will fail to compile.
- Nuclei QEMU no longer instantiates UART1 and no longer provides the MROM reset trampoline at `0x1000`; each hart now resets directly to the selected `download=` start address.

For this release pdf doc, please check https://doc.nucleisys.com/nuclei_tools/nuclei_tool_user_guide.pdf

Note: To check the known issues encountered after this release, please refer to the content at the following link:

- All Nuclei Tools Known Issues: Nuclei Tools Known Issues[78]
- GNU Toolchain Known Issues: *Known Issues* (page 433)
- LLVM Toolchain Known Issues: *Known Issues* (page 467)

[78] https://github.com/Nuclei-Software/nuclei-tool-guide/issues/18

- libncrt Known Issues: *Known Issues* (page 795)
- QEMU Known Issues: *Known Issues* (page 861)
- OpenOCD Known Issues: *Known Issues* (page 809)
- Nuclei Studio IDE Known Issues: *Known Issues* (page 398)
- Nuclei Model Known Issues: *Known Issues* (page 882)

Find release note below:

- Nuclei Studio 2026.10[79] : Add new features such as RISC-V Extension Selector, Flash Erase, Live Peripheral, CPU Information, MMU View and RISC-V Reference, optimize Connect to Running Target, Trace, Flame View and Live Watch, and add IDE support for the big-endian toolchain which is supplied separately, see *2026.10 Release Notes* (page 5).
- Nuclei RISC-V Toolchain 2026.10[80] : Based on 2025.10, add RNMI/Double Trap and Zibi extension support, enhance the Nuclei `xxldsp`/`xxlcz`/`xxlvqmacc` extensions, and fix a lot of known issues, see *Version 2026.10* (page 467).
- Nuclei QEMU 2026.10[81] : Based on 2025.10, add Nuclei USART/QSPI/uDMA/XEC support, improve ECLIC V2, timer and AIA support, and add big-endian guest support, see *Version 2026.10* (page 859).
- Nuclei OpenOCD 2026.10[82] : Based on 2025.10, add the `nuclei trace` command group, enhance `nuclei etrace` and `nuclei cpuinfo`, and add more flash drivers, see *Version 2026.10* (page 807).
- **Nuclei Model 2026.10** : Based on 2025.10, and support booting a RISC-V Linux kernel, SMP GDB debugging, big-endian ELF, semihosting and `soc-cfg` configuration files, see *Version 2026.10* (page 880).
- **Nuclei Studio VSCode 2026.10** : First official release of the Nuclei Studio VSCode extension (extension version **0.3.0**), bringing project management, build, debug, Live Watch, and NPK package management to Visual Studio Code, see *Nuclei Studio VSCode Changelog* (page 403).

9.2 2025.10

This is **2025.10** release of Nuclei Tools.

Note:

- This release is mostly based on 2025.02. but introduced some new features and fixed some known issues reported.
- Windows RISC-V Toolchain changed from win32 to win64 now, please take care!

For this release pdf doc, please check https://doc.nucleisys.com/nuclei_tools/nuclei_tool_user_guide.pdf

Find release note below:

- Nuclei Studio 2025.10[83] : Add a lot new features such as New Welcome, Connect to Running Target, Flame View, Nuclei Model Debug integration and IDE online update, see *2025.10 Release Notes* (page 8).
- Nuclei RISC-V Toolchain 2025.10[84] : Based on 2025.02, only bump newlib and glibc version, and do lot of fixup and some new features, see *Version 2025.10* (page 469).
- Nuclei QEMU 2025.10[85] : Based on 2025.02, and fixup some known issues and introduce some new features, see *Version 2025.10* (page 860).

[79] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2026.10
[80] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2026.10
[81] https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2026.10
[82] https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2026.10
[83] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2025.10
[84] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2025.10
[85] https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2025.10

- Nuclei OpenOCD 2025.10[86] : Based on 2025.02. and fixup some known issues and enhance existing features such as etrace, cpuinfo, cross trigger and etc, see *Version 2025.10* (page 808).
- **Nuclei Model 2025.10** : Introduced a lot of new features in this release, gdb debug and flame graph are also supported, see *Version 2025.10* (page 882).

Click https://github.com/Nuclei-Software/nuclei-tool-guide/issues/18 for known issues.

9.3 2025.02

This is 2025.02 release of Nuclei Tools.

For this release pdf doc, please check https://doc.nucleisys.com/nuclei_tools/nuclei_tool_user_guide_2025.02.pdf

Find release note below:

- Nuclei Studio 2025.02[87] : Add a lot new features such as Live watch, Modeling integration, flash programming, and Nice Wizard etc.
- Nuclei RISC-V Toolchain 2025.02[88] : Bump to GCC 14 and LLVM 19.
- Nuclei QEMU 2025.02[89] : Bump to Qemu 9.0.
- Nuclei OpenOCD 2025.02[90] : Bump to OpenOCD 0.12.0

Click https://github.com/Nuclei-Software/nuclei-tool-guide/issues/18 for this release known issues.

9.4 2024.06

This is 2024.06 release of Nuclei Tools.

This release introduced *About Nuclei Near Cycle Model* (page 863) first release which can be used in Ubuntu 20.04 environment.

For this release pdf doc, please check https://doc.nucleisys.com/nuclei_tools/nuclei_tool_user_guide_2024.06.pdf

Find release note below:

- Nuclei Studio 2024.06[91]
- Nuclei RISC-V Toolchain 2024.06[92]
- Nuclei QEMU 2024.06[93]
- Nuclei OpenOCD 2024.06[94]

Click https://github.com/Nuclei-Software/nuclei-tool-guide/issues/4 for this release known issues.

[86] https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2025.10
[87] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2025.02
[88] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2025.02
[89] https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2025.02
[90] https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2025.02
[91] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2024.06
[92] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2024.06
[93] https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2024.06
[94] https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2024.06

9.5 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

9.6 2023.10

This is 2023.10 release of Nuclei Tools.

Find release note below:

- Nuclei Studio 2023.10[95]
- Nuclei RISC-V Toolchain 2023.10[96]
- Nuclei QEMU 2023.10[97]
- Nuclei OpenOCD 2023.10[98]

Click https://github.com/Nuclei-Software/nuclei-tool-guide/issues/1 for this release known issues.

[95] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2023.10
[96] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2023.10
[97] https://github.com/riscv-mcu/qemu/releases/tag/nuclei-2023.10
[98] https://github.com/riscv-mcu/riscv-openocd/releases/tag/nuclei-2023.10

CHAPTER

TEN

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

ELEVEN

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

TWELVE

INDICES AND TABLES

- genindex
- search

INDEX

A

API, **889**

I

ISR, **889**

X

XIP, **889**