Gowin RiscV_AE350_SOC RDS 软件

用户指南

SUG1025-1.0, 2023-07-21

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其它方式授予任何知识产权许可。除高云半导体在其产品的销售条款和条件中声明的责任之外，高云半导体概不承担任何法律或非法律责任。高云半导体对高云半导体产品的销售和／或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等，均不作担保。高云半导体对文档中包含的文字、图片及其它内容的准确性和完整性不承担任何法律或非法律责任，高云半导体保留修改文档中任何内容的权利，恕不另行通知。高云半导体不承诺对这些文档进行适时的更新。

版本信息

日期	版本	描述
2023/07/21	1.0	初始版本。

目录

图目录

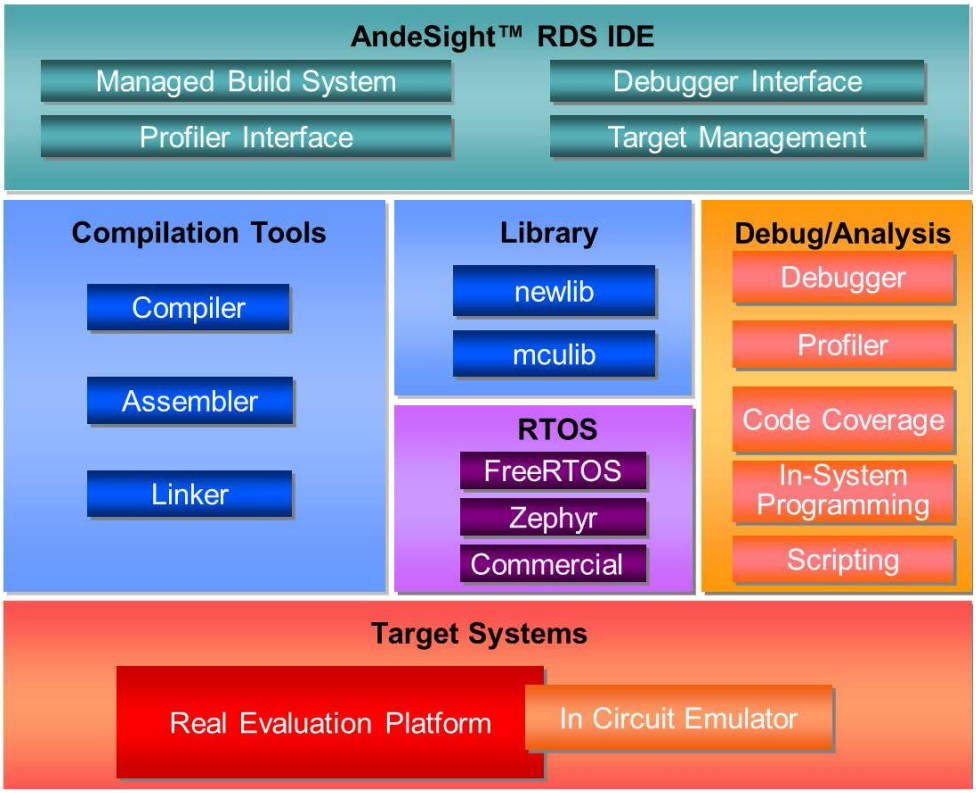
表目录

1 RDS 软件简介

RDS 软件是高云®半导体基于 Andes Technology 的集成开发套件二次开发的软件，用于创建、构建、调试和管理 RiscV_AE350_SOC 应用程序。RDS 软件包括各种定制化组件，如编译器、工具链、ICE（In Circuit Emulator）驱动程序等，可以帮忙用户在 RiscV_AE350_SOC 目标系统上运行、编辑、调试和分析应用程序。

RDS 软件的主要组件和功能，如图 1-1 所示。

图 1-1 RDS 软件结构

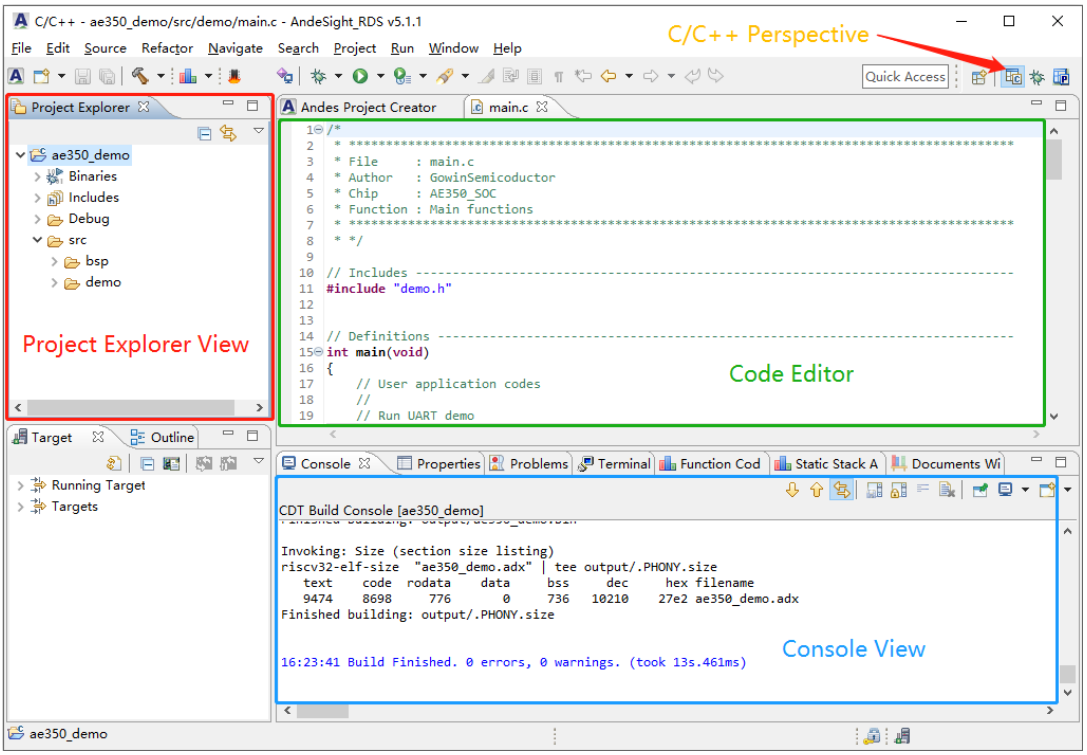


1.1 软件概述

RDS 软件的工作区由各个被称为“视图”的窗格组合而成，各个视图组合在一起形成以功能为导向的“透视图”。

如图 1-2 所示的 C/C++透视图（C/C++ Perspective），包括项目资源管理器视图（Project Explorer View）、代码编辑器（Code Editor）和控制台视图（Console View），便于用户浏览和定位项目文件、编辑代码，以及查看标准输入/输出结果。

图 1-2 C/C++ 透视图



同样 RDS 软件也定义了调试透视图（Debug Perspective）和分析透视图（Profile Perspective），可以通过主菜单“Window > Perspective > Open Perspective”或透视图工具栏“ ”进入不同的透视图，如图 1-3 和图 1-4 所示。

图 1-3 选择 Window > Perspective > Open Perspective

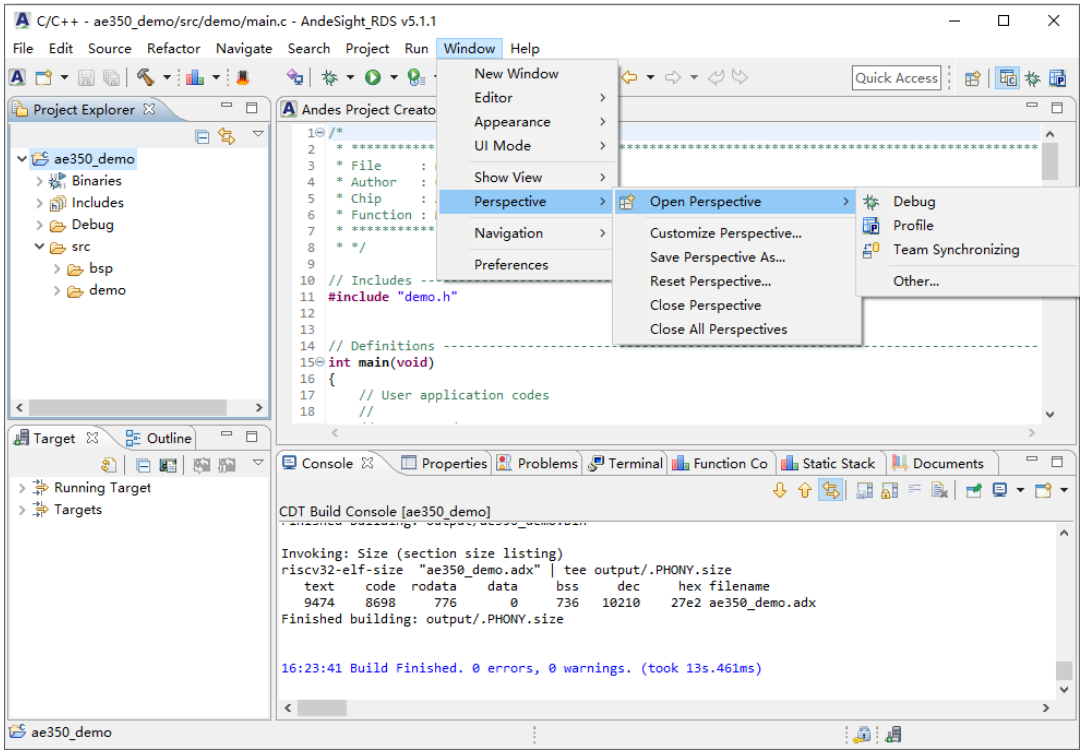
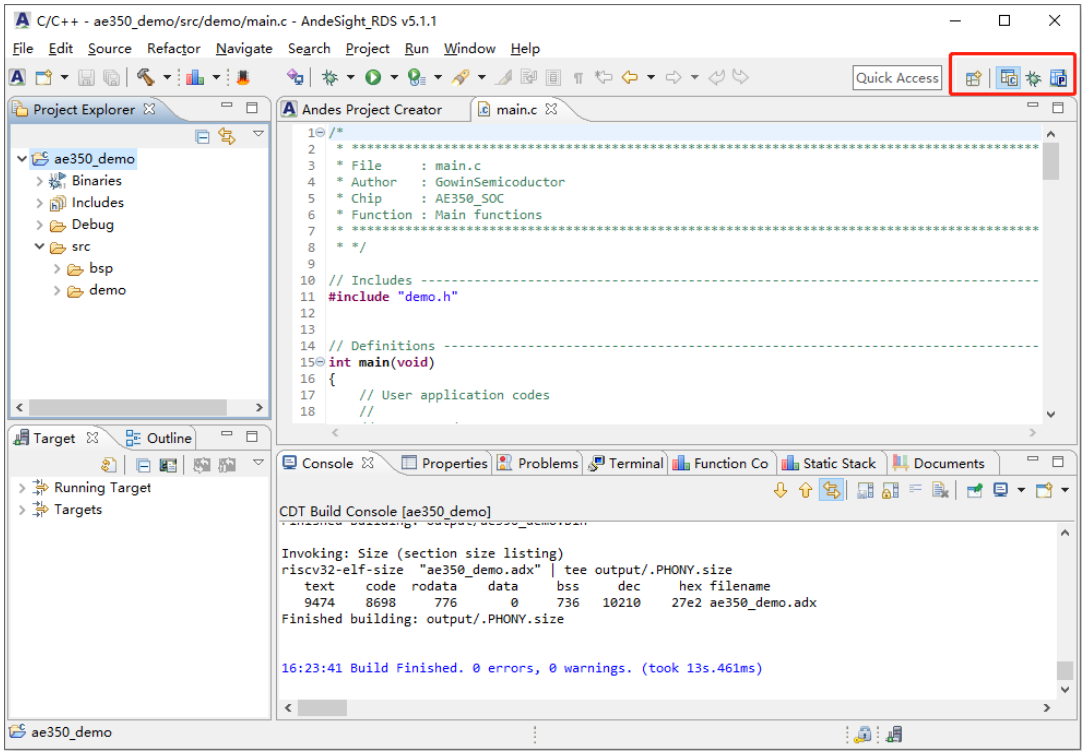
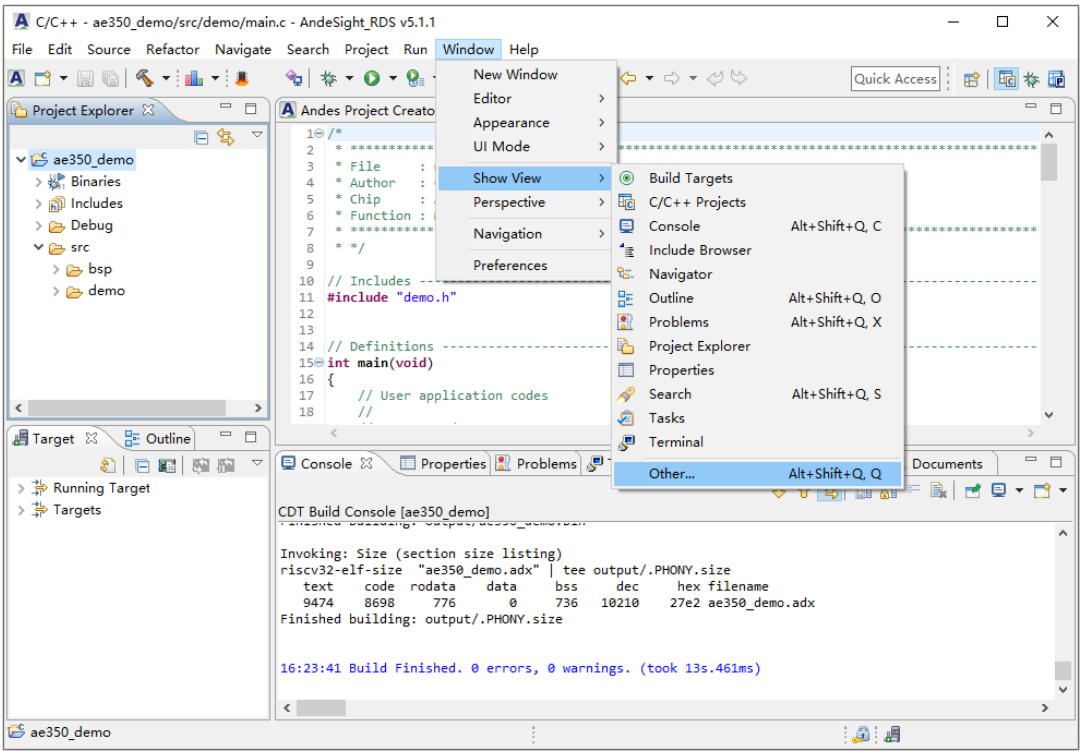


图 1-4 透视图工具栏



如果当前透视图没有包含某些想要的视图，则可以通过主菜单“Window > Show View”打开这些视图，如图 1-5 所示。

图 1-5 选择 Window > Show View



1.2 启动软件

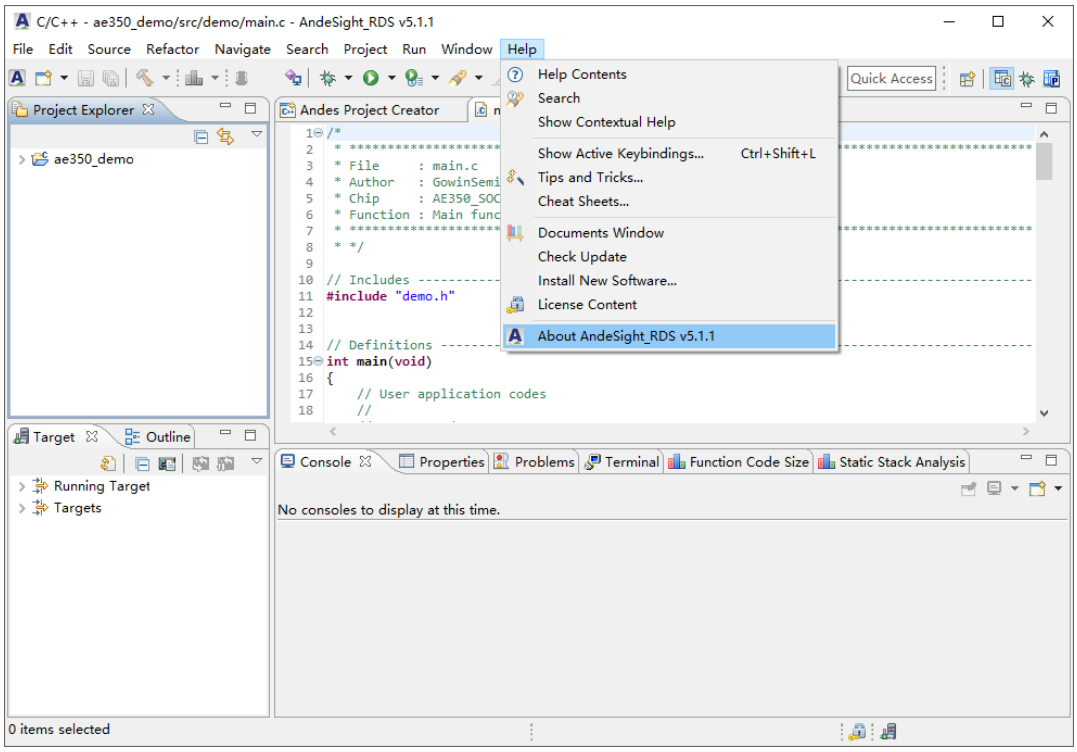
本地 Windows PC 安装 RDS 软件后，可以双击桌面快捷方式“”或安装目录“AndeSight_RDS_v511\ide\AndeSight.exe”，启动 RDS 软件。

1.3 组件版本信息

参照以下步骤查看 RDS 软件已安装的组件：

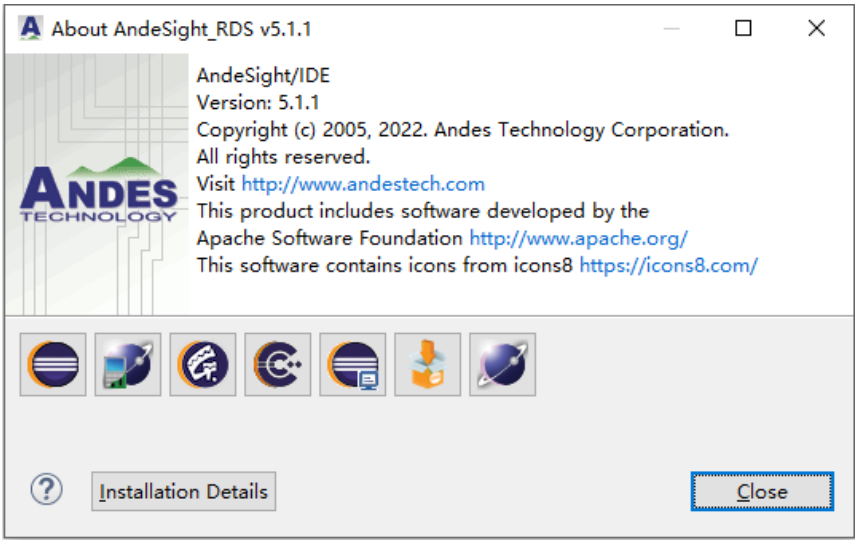
1. 选择 RDS 软件主菜单“Help > About AndeSight_RDS v511”，如图 1-6 所示。

图 1-6 选择 Help > About AndeSight_RDS v511



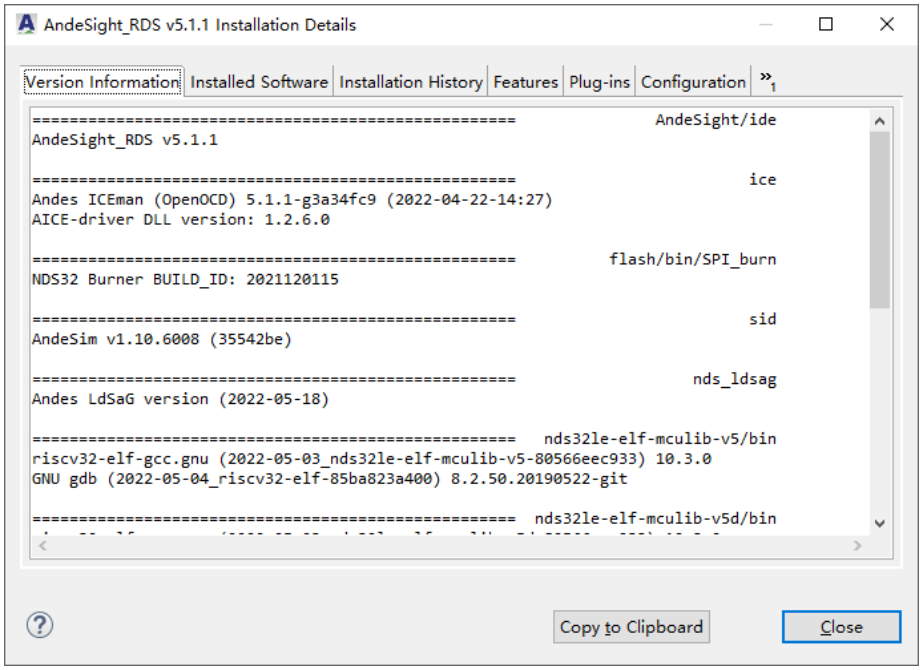
2. 在“About AndeSight_RDS v5.1.1”对话框中，查看当前正在使用的 RDS 软件的版本信息，如图 1-7 所示。

图 1-7 关于 RDS 软件



3. 单击“Installation Details”，在“AndeSight_RDS v5.1.1 Installation Details”对话框中，“Version Information”列表显示 RDS 软件已安装组件的版本信息，如图 1-8 所示。

图 1-8 安装组件的版本信息

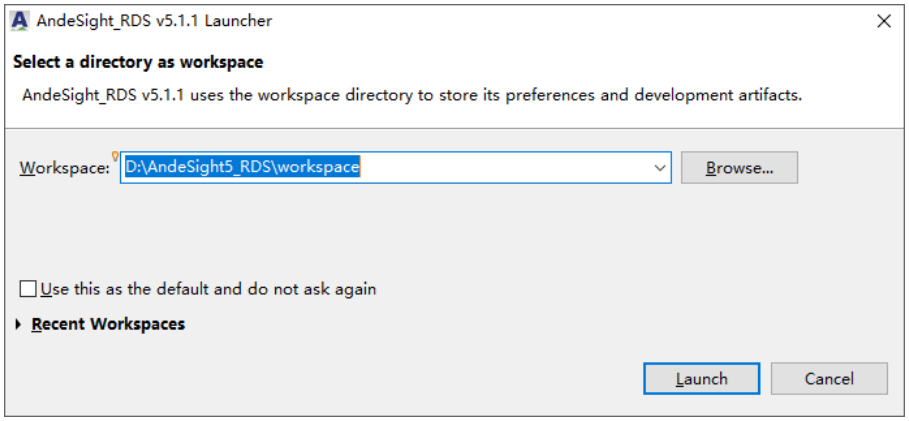


1.4 工作空间

工作空间（Workspace）用于保存和管理 RDS 软件的软件工程，默认路径为“USER_HOME\AndeSight5\workspace”。

首次启动 RDS 软件时，在“AndeSight_RDS v5.1.1 Launcher”对话框中显示 RDS 软件当前工作空间的路径，如图 1-9 所示。如果想要继续在此工作空间保存和管理软件工程，单击“Launch”即可。如果想要更改或创建其他的工作空间，单击“Browse…”，指定其他工作空间的路径。注意工作空间的路径必须由字符（A-Z, a-z, 0-9）、下划线（_）和连字符（-）组成。

图 1-9 RDS Launcher



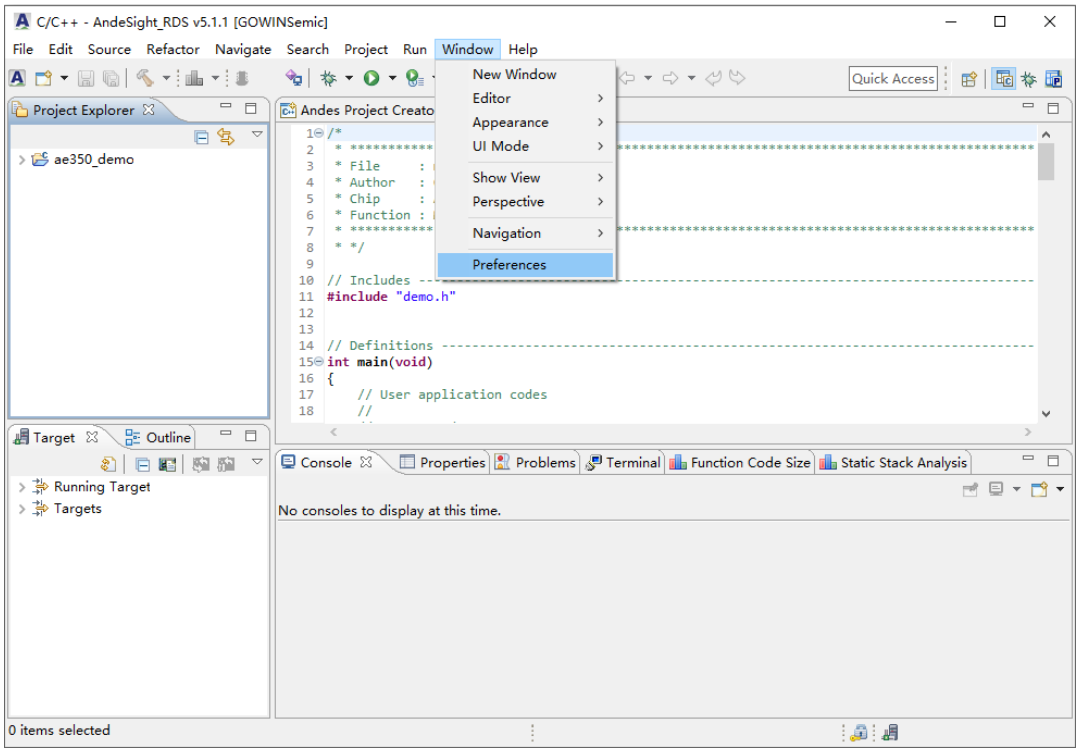
1.5 显示语言选项

RDS 软件包括 3 种语言（英文、日文和简体中文）显示软件的菜单、向导、对话框和其他文本等。

参照以下步骤更换软件的显示语言：

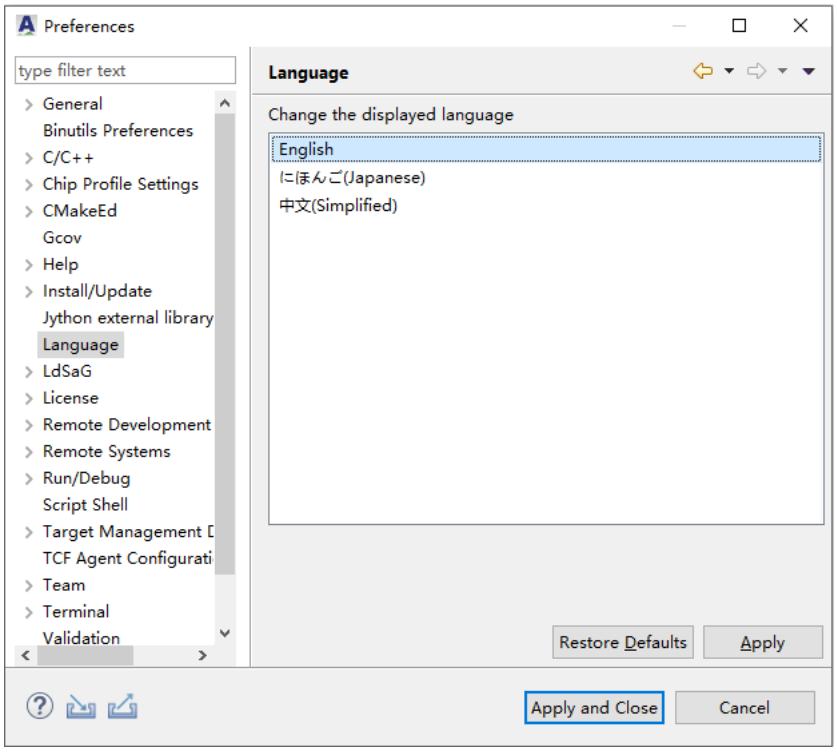
1. 选择 RDS 主菜单“Window > Preferences > Language”，如图 1-10 所示。

图 1-10 选择 Window > Preferences > Language



2. 选择一种显示语言，单击“Apply”。在“Change the displayed language”对话框中，单击“Yes”，重启 RDS 软件，如图 1-11 所示。

图 1-11 更换显示语言



3. RDS 软件重启后，软件以更换过的语言显示。

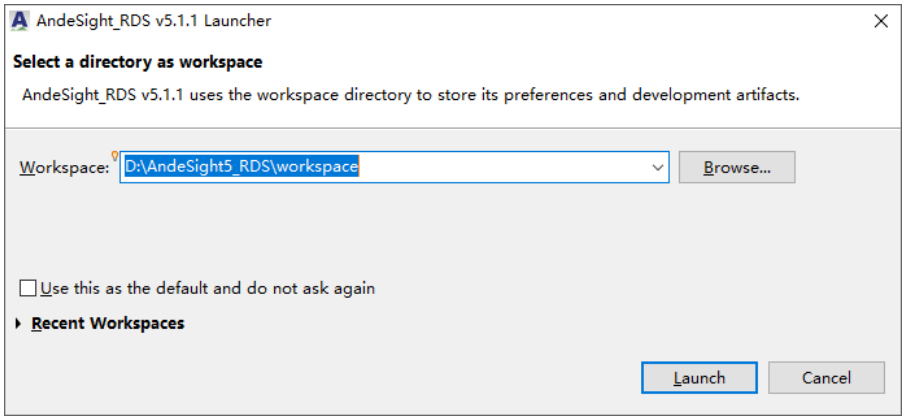
1.6 快速入门

本节提供快速入门教程，旨在便于用户快速掌握 RDS 软件创建/构建/调试应用程序。

参照以下步骤初步了解 RDS 软件的应用流程：

1. 双击桌面快捷方式“”或 RDS 软件安装目录“AndeSight_RDS_v511\ide\ AndeSight.exe”，启动 RDS 软件。
2. “AndeSight_RDS v5.1.1 Launcher”对话框中，提示工作空间的默认路径“USER_HOME\AndeSight5\workspace”，单击“Launch”继续下一步，如图 1-12 所示。

图 1-12 RDS Launcher



3. 在项目创建视图（Andes Project Creator）中，选择连接配置、项目语言和“Chip Profile”，双击选定的“Chip Profile”或单击“Create Project…”，创建软件工程，如图 1-13 所示。

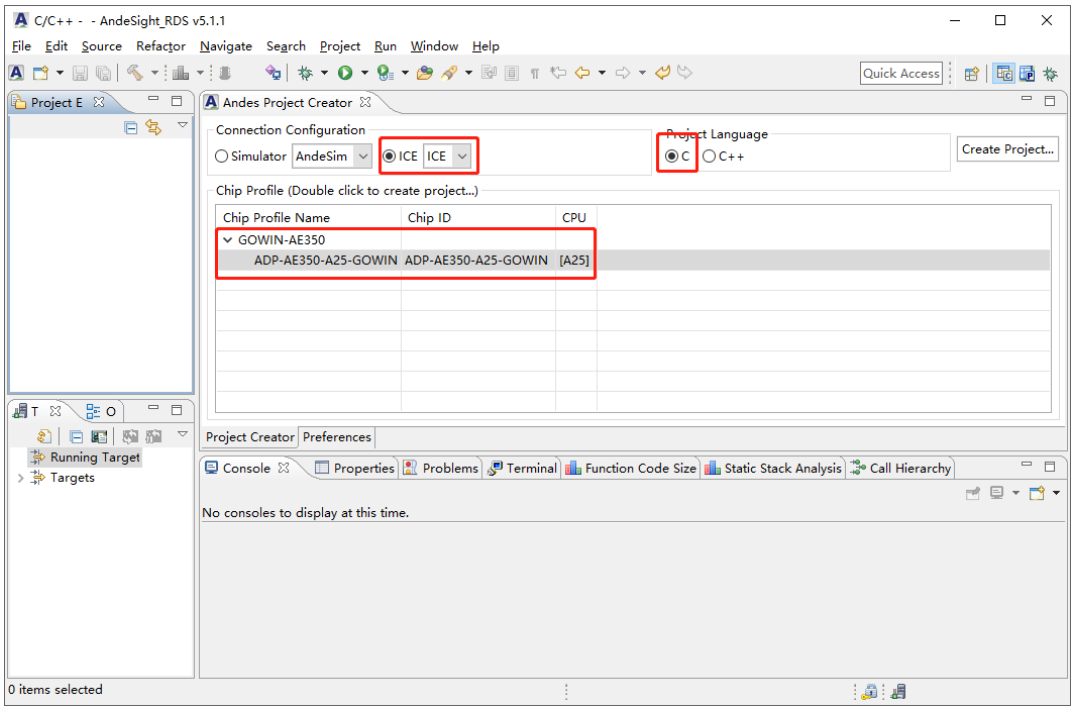
   例如：

   Connection Configuration：ICE

   Project Language：C

   Chip Profile：GOWIN-AE350（ADP-AE350-A25-GOWIN）

图 1-13 项目创建视图



4. “C Project”对话框中，指定 Project name、Location、Project type 和 Toolchains，单击“Next”，如图 1-14 所示。

例如：

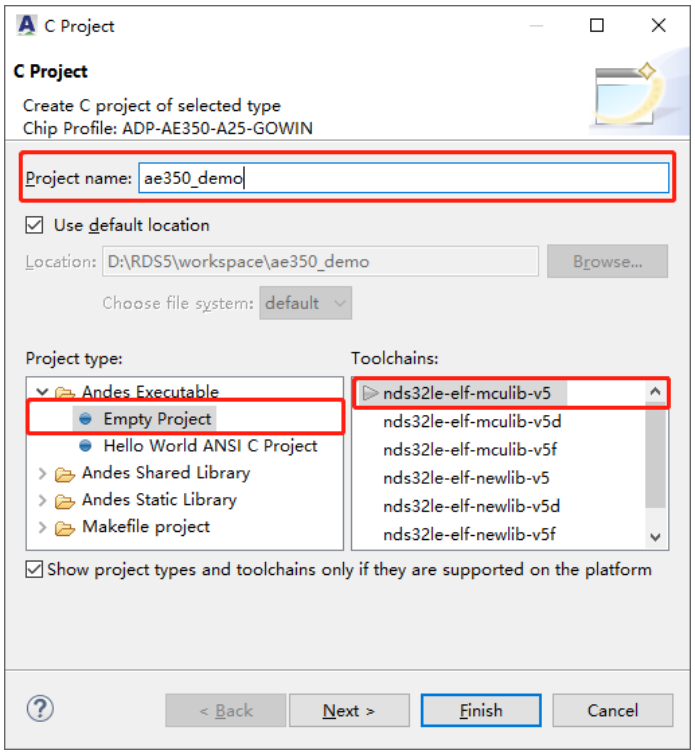
Project name：ae350_demo

Location：D:\RDS\workspace

Project type：Empty Project

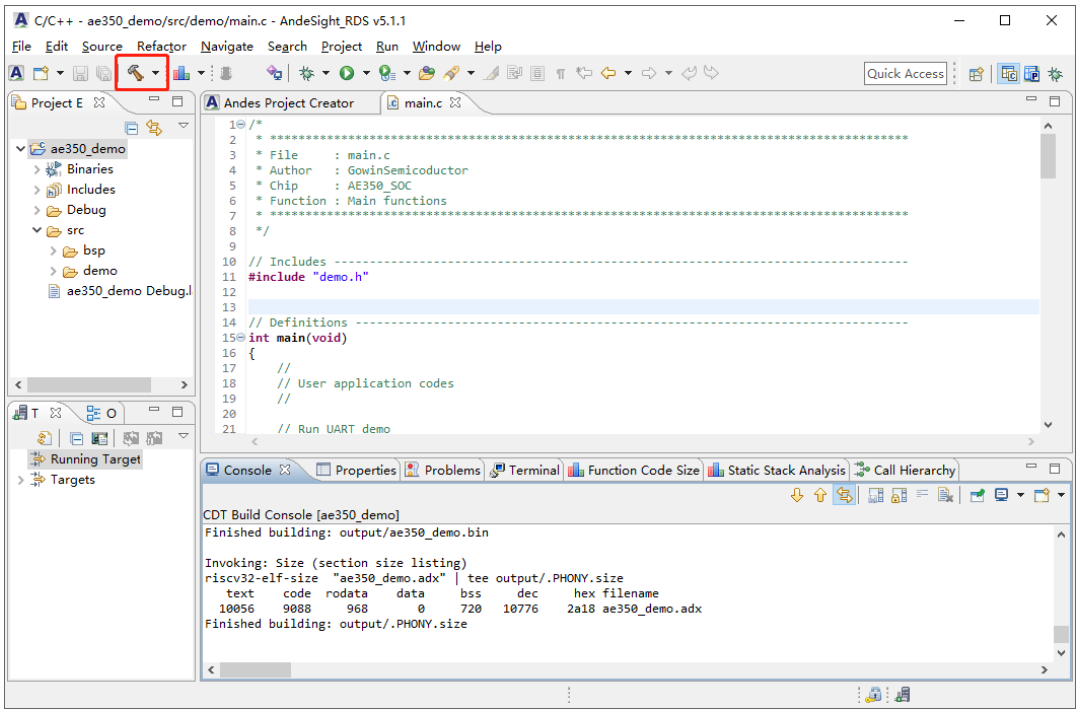
Toolchains：nds32le-elf-mculib-v5

图 1-14 创建新的 C 软件工程



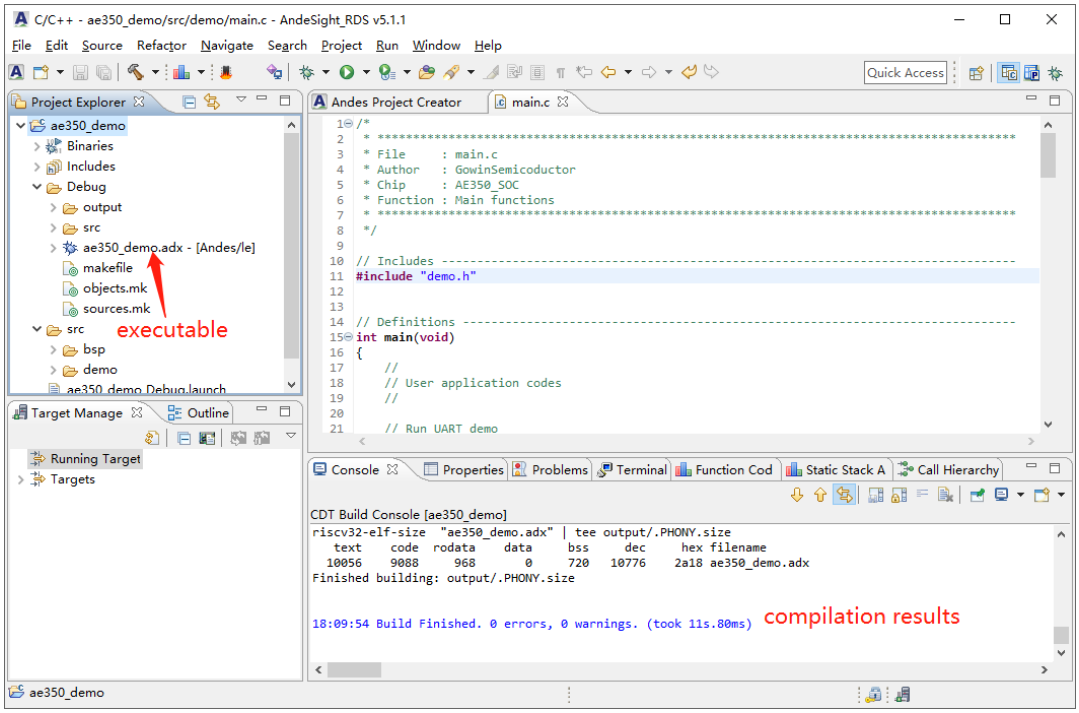
5. 在项目资源管理器视图中，选择已创建的软件工程，单击 RDS 软件工具栏“”（Build），构建软件工程，如图 1-15 所示。

图 1-15 构建软件工程



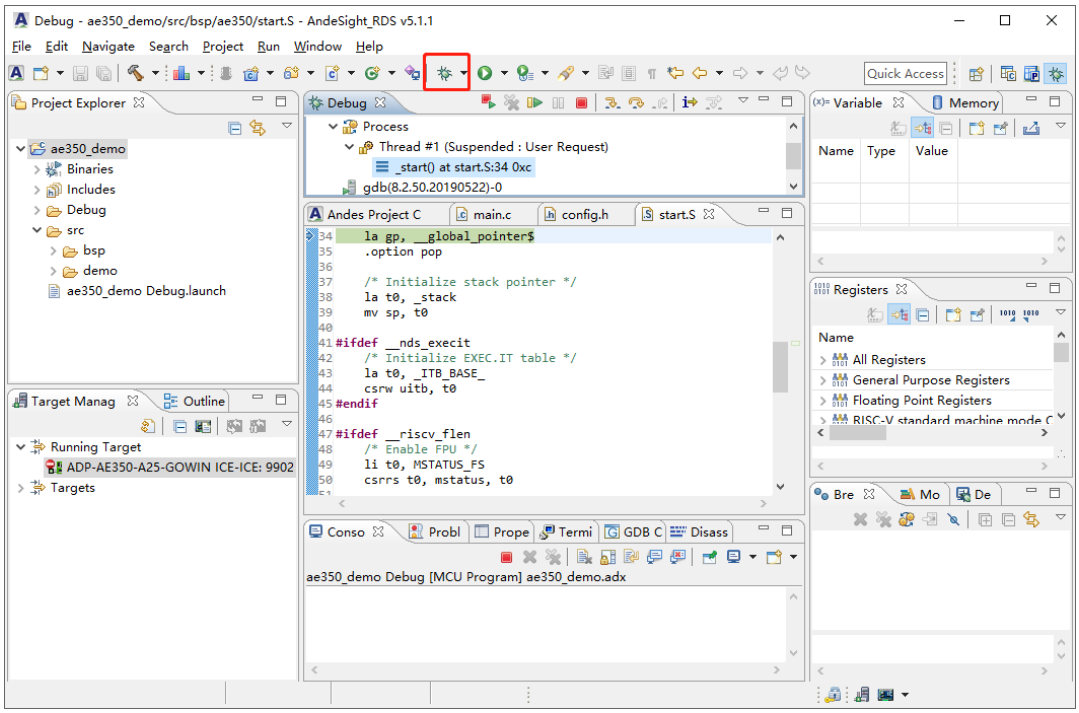
6. 检查控制台视图中的编译结果，验证产生的可执行文件，如图 1-16 所示。

图 1-16 控制台视图的编译结果



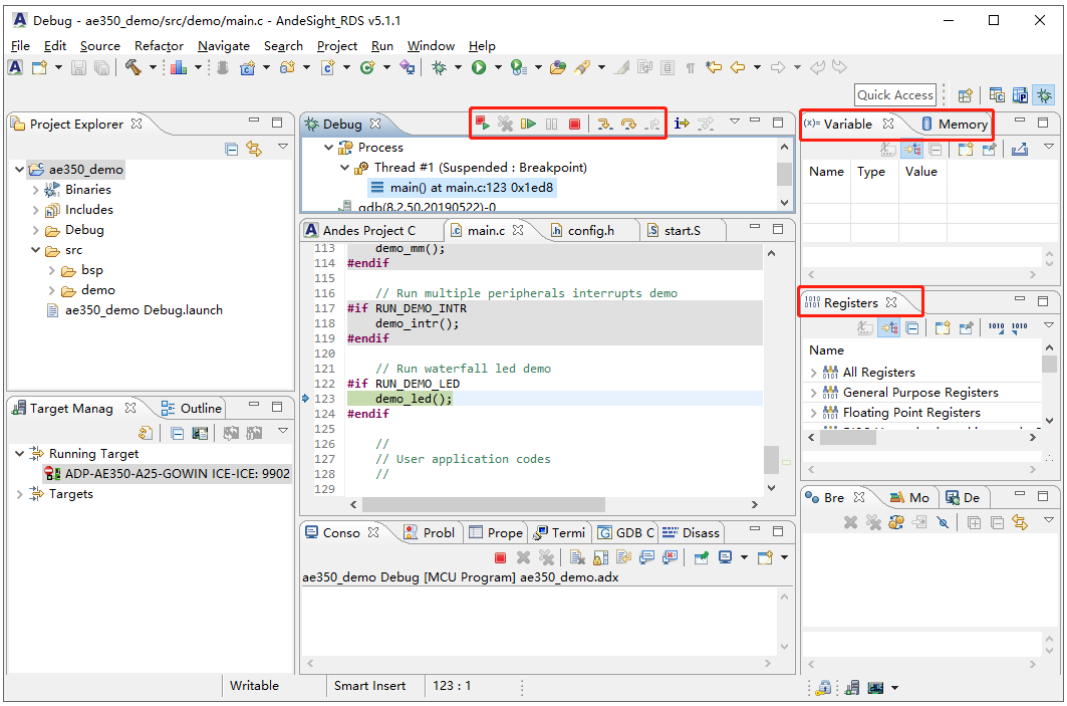
7. 单击 RDS 软件工具栏“ ”（Debug）启动调试，如图 1-17 所示。

图 1-17 调试软件工程



8. 进入调试透视图，单击调试视图（Debug View）工具栏“”、“”、“”等控制调试线程，也可以通过其他调试视图查看调试结果（例如，变量视图、寄存器视图……），如图 1-18 所示。

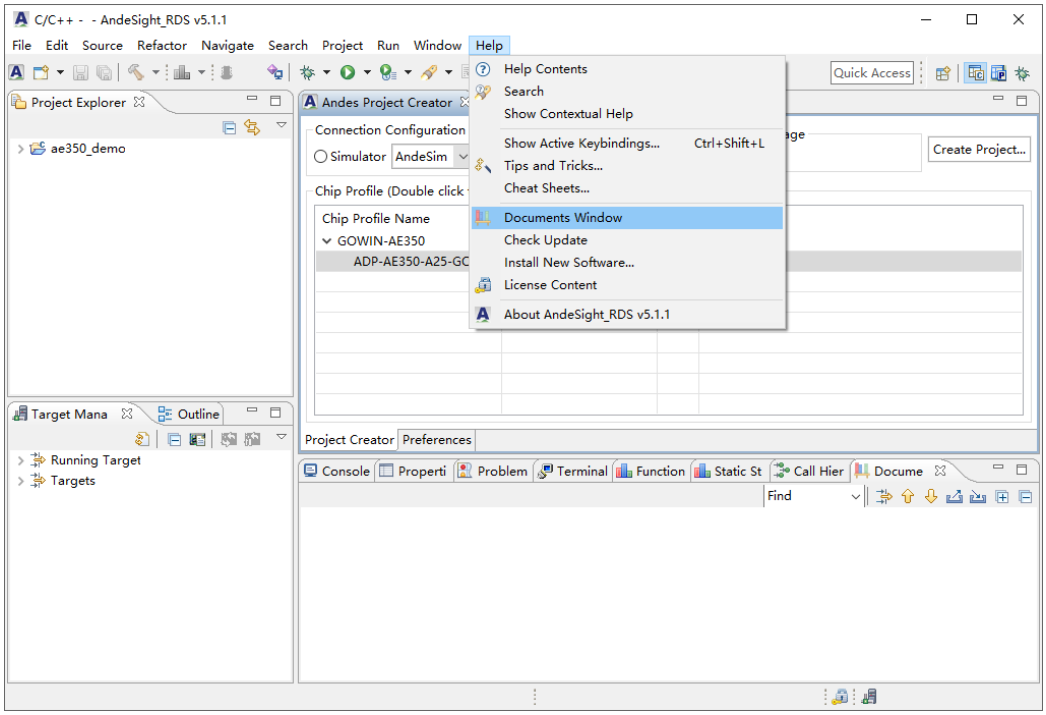
图 1-18 调试视图



1.7 帮助文档

在文档窗口视图（Document Window View）中可以查找 RDS 软件用户手册、工具指南、教程、规范手册和其他已支持的手册，选择 RDS 软件主菜单“Help > Documents Window”，如图 1-19 所示。

图 1-19 选择 Help > Documents Window



2 创建和构建软件工程

RDS 软件可以创建一个新的软件工程，或把已有的软件工程引入工作空间。编译和链接一个软件工程的所有源文件，产生可执行文件或目标文件的过程，称为构建。

2.1 创建软件工程

2.1.1 新建软件工程

1. RDS 软件启动后，找到项目创建视图或单击 RDS 软件工具栏“  ”（Andes Project Creator），选择“Connection Configuration"、“Project Language”和“Chip Profile”，双击选定的“Chip Profile”或单击“Create Project…”，创建软件工程，如图 2-1 所示。

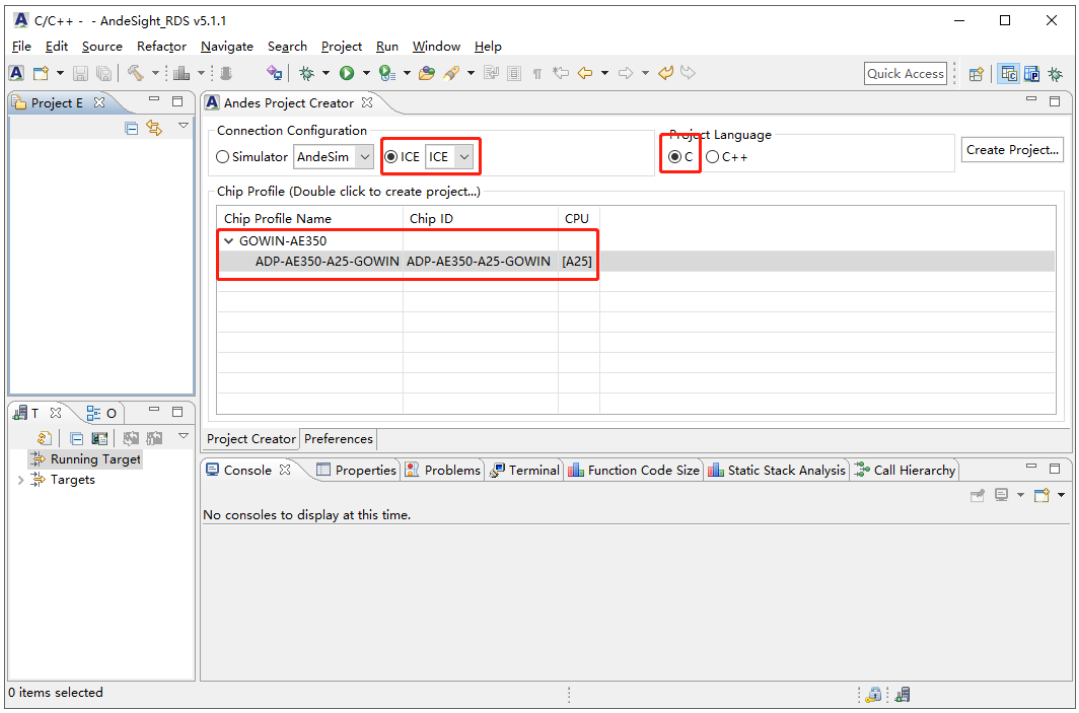
   例如：

   Connection Configuration：ICE

   Project Language：C

   Chip Profile：GOWIN-AE350（ADP-AE350-A25-GOWIN）

图 2-1 项目创建视图



项目创建视图中的选项配置如表 2-1 所示。

表 2-1 项目创建视图选项

选项	配置	描述
Connection Configuration	Simulator：AndeSim	模拟器，不需要连接目标板
	ICE：ICE	连接 ICE 调试器和目标板； ICE 调试器，例如 AICE-MINI+，请联系 Andes Technology 购买。
Project Language	C	C 编程语言
	C++	C++编程语言
Chip Profile	GOWIN-AE350：ADP-AE350-A25-GOWIN	指定 RiscV_AE350_SOC

2. “C Project”对话框中，指定 Project name、Location、Project type 和 Toolchains，单击“Next”，如图 2-2 所示。

   例如：

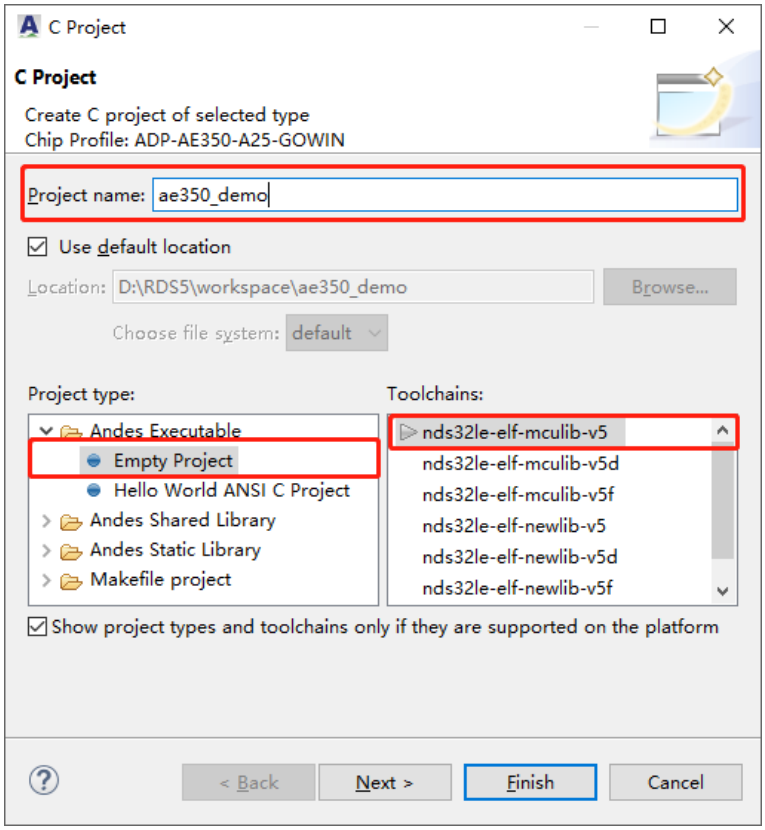
   Project name：ae350_demo

   Location：D:\RDS\workspace

   Project type：Empty Project

   Toolchains：nds32le-elf-mculib-v5

图 2-2 创建新的 C 软件工程



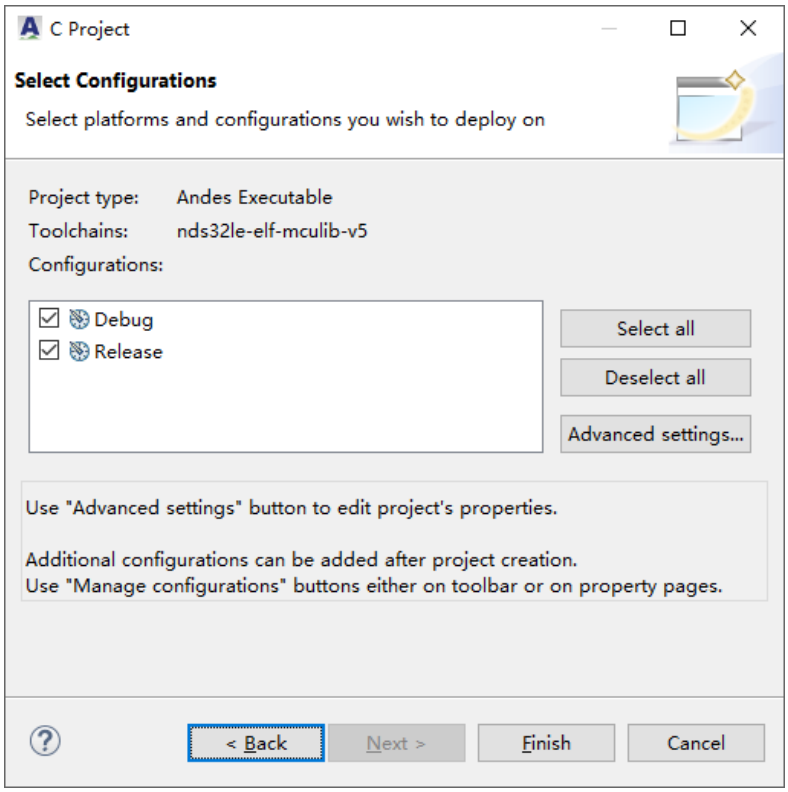
C Project 对话框中的选项配置如表 2-2 所示。

表 2-2 C Project 对话框选项

选项	配置	描述
Project type	Empty Project	创建空的软件工程。
Toolchains	nds32le-elf-mculib-v5 nds32le-elf-mculib-v5d nds32le-elf-mculib-v5f nds32le-elf-newlib-v5 nds32le-elf-newlib-v5d nds32le-elf-newlib-v5f	RDS 软件支持几种编译工具链：MCUlib 和 Newlib： Newlib 是一个用于嵌入式系统的 C 软件库； MCUlib 是一个优化增强的 C 软件库，适用于 MCU 应用程序，代码规模小于 Newlib
Project name	…	软件工程的名称，仅能包含以下字符：A-Z 0-9 ! @ % ^ [ ] . - + , ~ _。
Location	Use default location	软件工程的路径。

3. 指定部署平台和配置，例如“Debug”或“Release”，单击“Finish”，如图 2-3 所示。

图 2-3 指定部署平台和配置

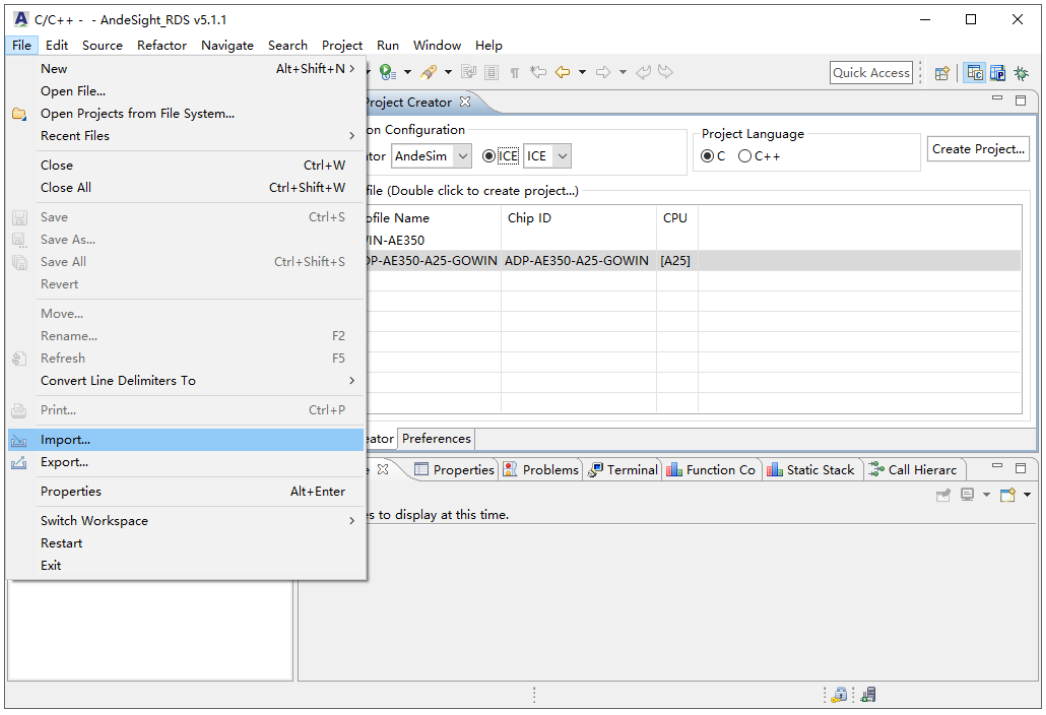


2.1.2 引入已有的软件工程

参照以下步骤引入已有的软件工程到当前工作空间：

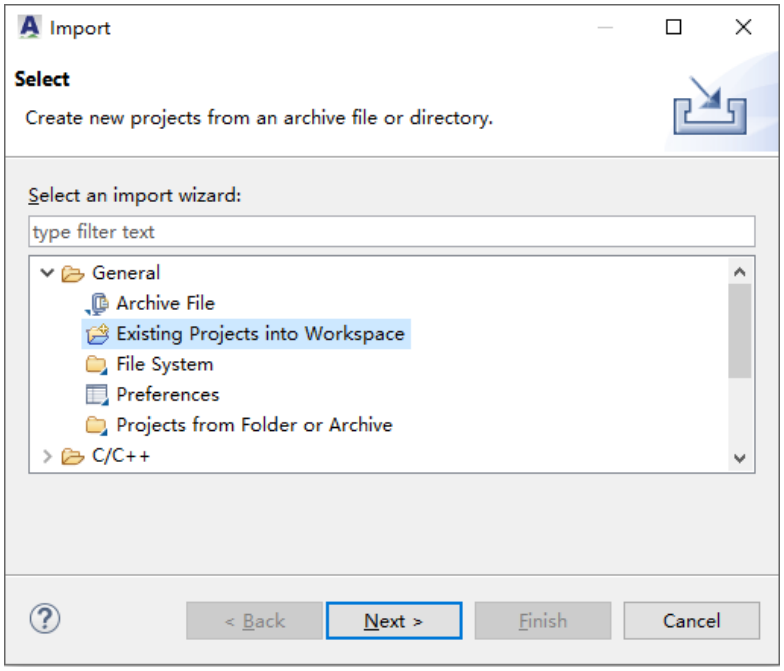
1. 选择 RDS 软件主菜单“File > Import”，如图 2-4 所示。

图 2-4 选择 File > Import



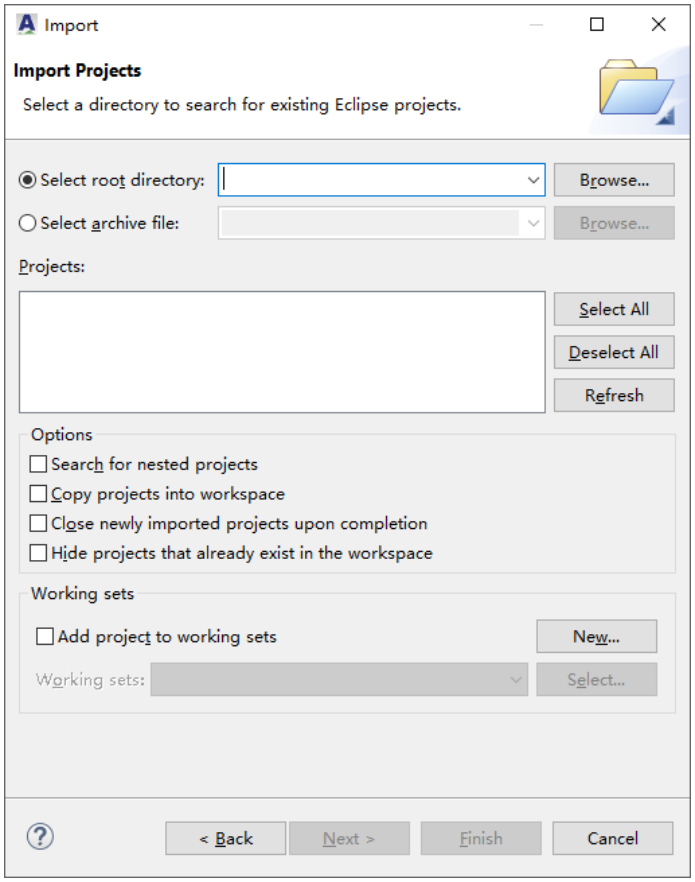
2. 在“Import”对话框中，选择“General > Existing Projects into Workspace”，单击“Next”，如图 2-5 所示。

图 2-5 选择 General > Existing Projects into Workspace



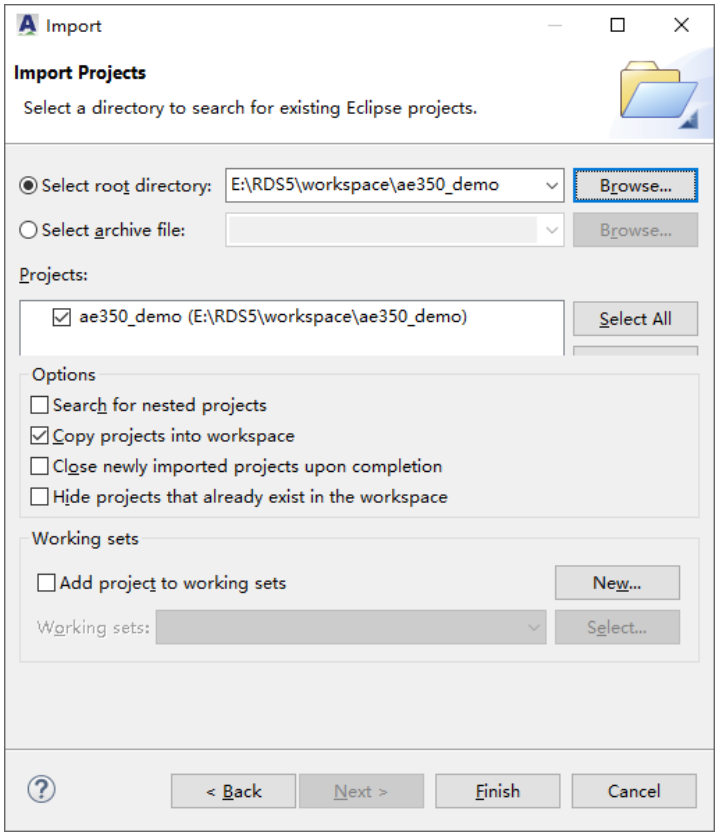
3. 选择“Select root directory”，从文件系统引入一个已有的软件工程，或选择“Select archive file”，引入一个软件工程的压缩文件，单击“Browse…”，如图 2-6 所示。

图 2-6 Import Projects



4. 在“Import Projects”对话框中，选择想要引入的软件工程或压缩文件，单击“Select Folder”或“Open”，引入软件工程。
5. 选定想要引入的软件工程后，选择“Copy projects into workspace”，单击“Finish”，如图 2-7 所示。注意对于引入软件工程的压缩文件，必须选择“Copy projects into workspace”。

图 2-7 选择 Copy projects into workspace



6. 在项目资源管理器视图中显示已引入的软件工程。

为确保能够成功构建和运行引入的软件工程，RDS 软件会自动检查引入的软件工程的芯片配置、工具链和连接配置。如果 RDS 软件不支持引入的软件工程所用的芯片配置或工具链或连接配置，参照软件提示选择一种合适的替代方案。

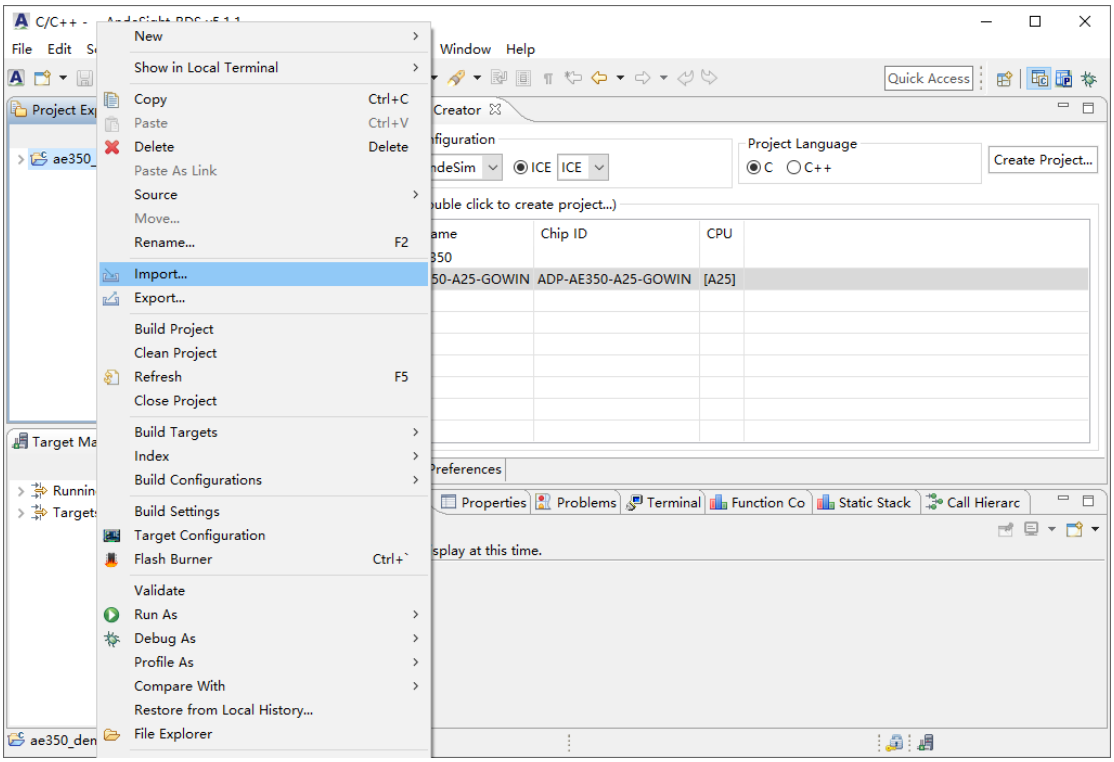
2.2 软件工程文件

在工作空间中创建软件工程后，软件工程可以引入已有的源代码文件，或新建源代码文件。

2.2.1 引入已有的文件

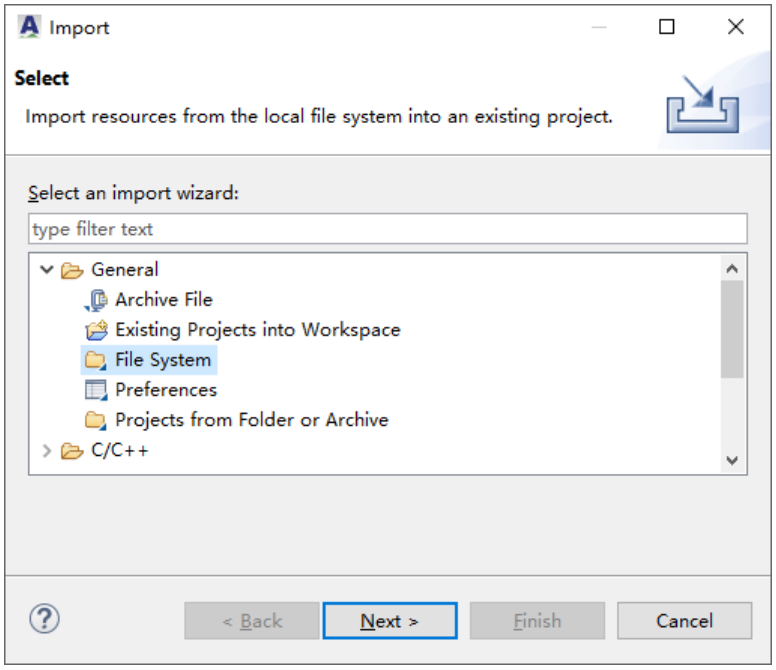
1. 在项目资源管理器视图中，右单击选定的软件工程，下拉菜单中选择“Import…”，如图 2-8 所示。

图 2-8 选择 Import...



2. 在“Import”对话框中，选择“General > File System”，单击“Finish”，如图 2-9 所示。

图 2-9 选择 General > File System

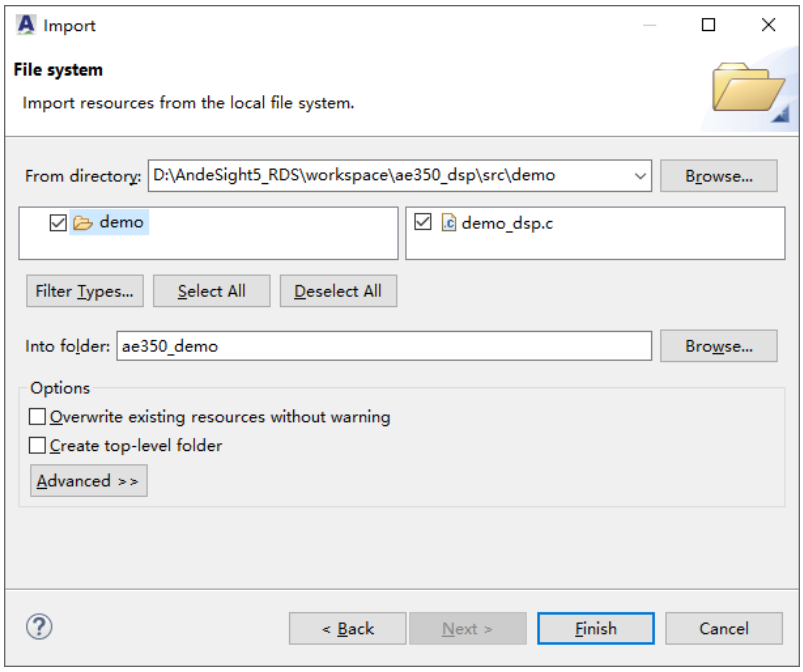


3. 在“File system”对话框中，输入想要引入的文件的路径，或单击

“Browse…”，搜索文件路径。

4. 在“File system”对话框的中间位置，左栏显示想要引入的文件的路径，右栏显示此路径包含的所有文件。在右栏中选择想要引入的文件，“Info folder”指定想要引入到的软件工程，单击“Finish”，如图 2-10 所示。

图 2-10 选择引入的文件

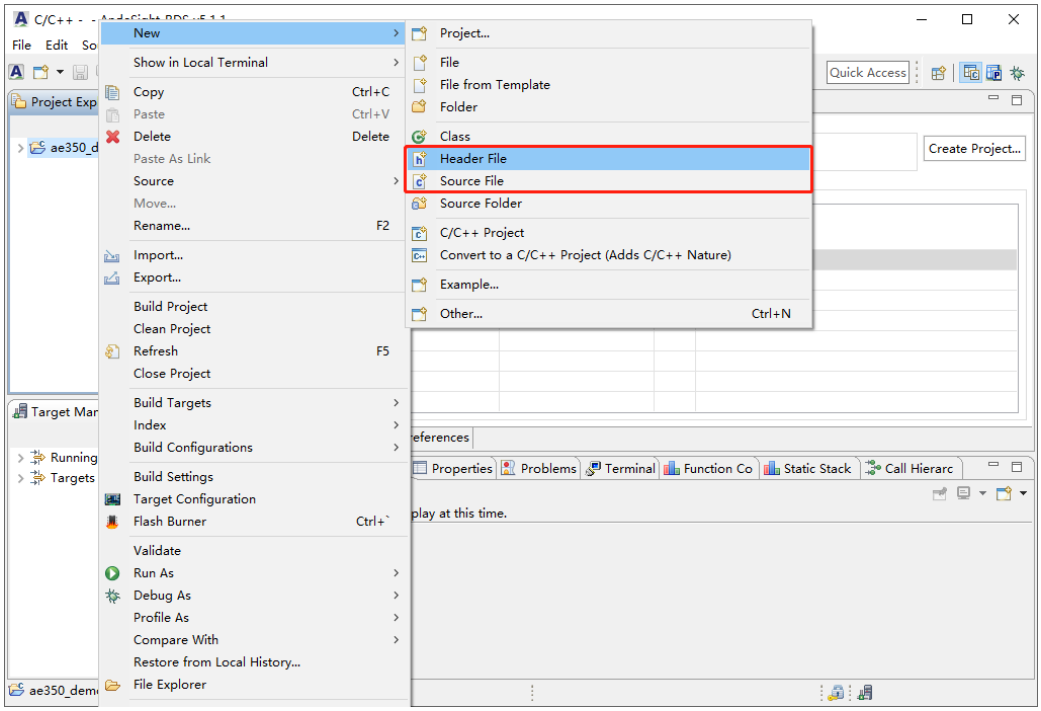


5. 返回项目资源管理器视图，可以查看已引入到软件工程的源代码文件。

2.2.2 新建源/头文件

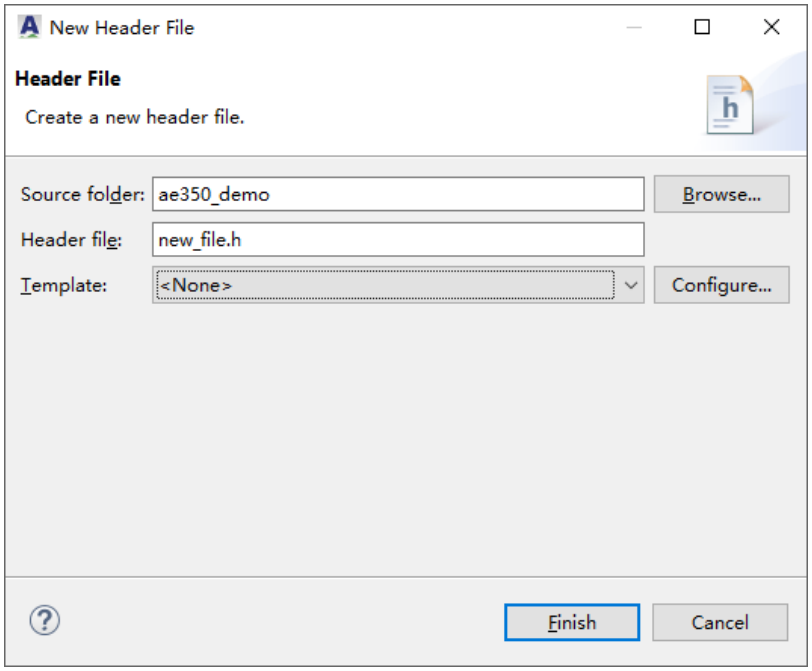
1. 右单击选定的软件工程，下拉菜单中选择“New > Source File”或“Header File”，如图 2-11 所示。

图 2-11 选择 New > Source File 或 Header File



2. 在“New Source File”或“New Header File”对话框中，“Source file”或“Header file”输入新的源/头文件的名称，包括后缀名.c 或.h，“Template”下拉列表选择一种模板，例如“<None>”，单击“Finish”，如图 2-12 所示。

图 2-12 创建新的源/头文件



3. 返回项目资源管理器视图，在选定的软件工程中查看新建的源/头文件，

双击打开此文件，开始编写代码。

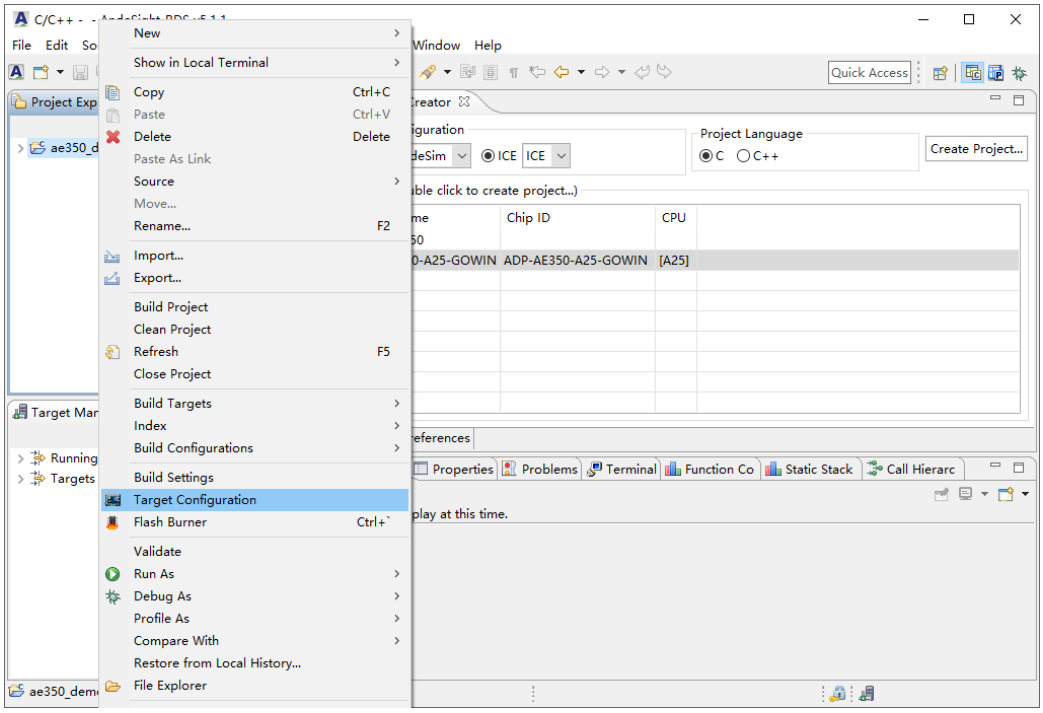
2.3 软件工程目标配置

选定的软件工程的目标配置会覆盖整个工作空间目标管理的默认设置。

参照以下步骤设置软件工程的目标配置：

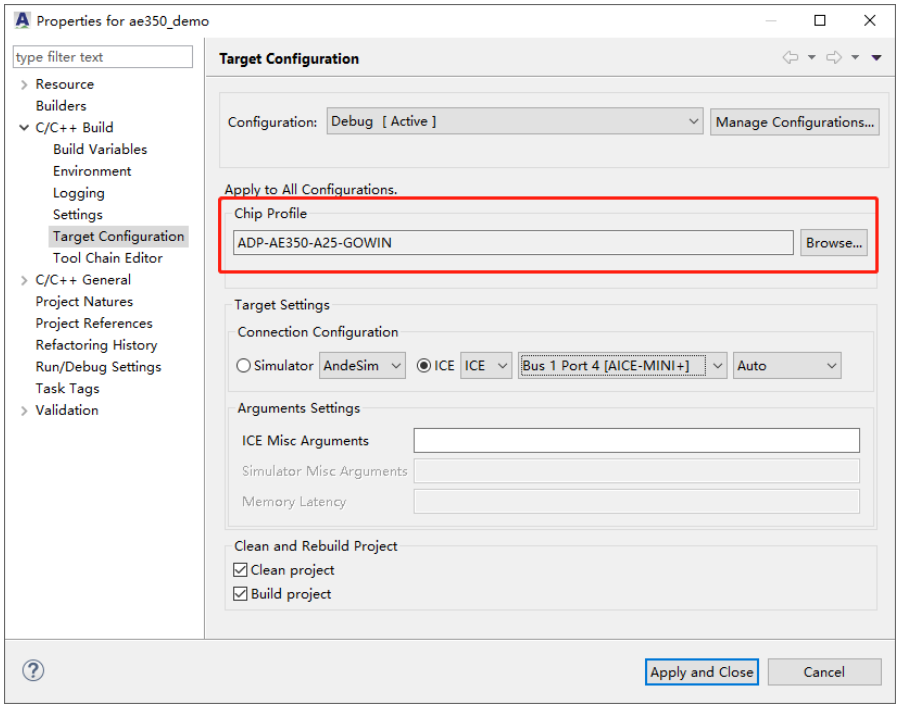
1. 右单击选定的软件工程，下拉菜单中选择“Target Configuration”，如图 2-13 所示。

图 2-13 选择 Target Configuration



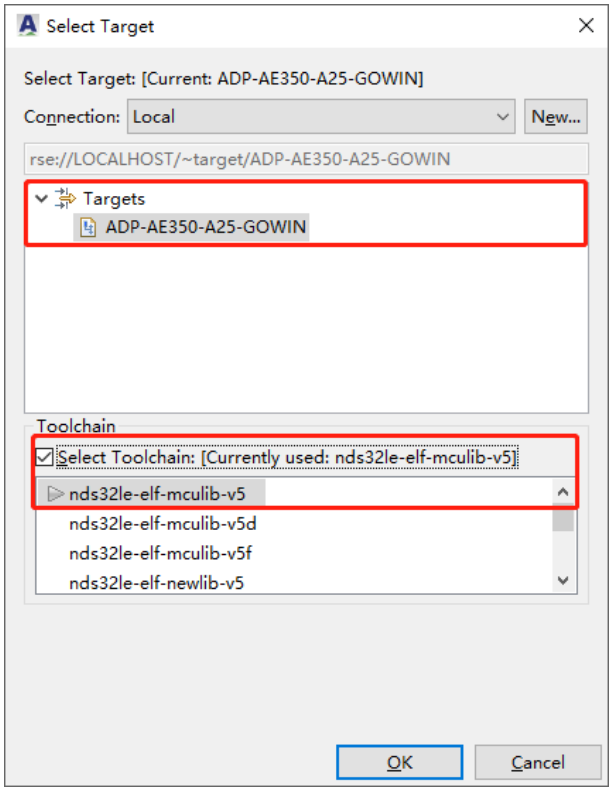
2. 在“Target Configuration”对话框中，选择“Chip Profile”，单击“Browse…”，如图 2-14 所示。

图 2-14 选择 Chip Profile



在“Select Target”对话框中，“Targets”选择目标，例如“ADP-AE350-A25-GOWIN”。单击“Select Toolchains”，下拉列表中选择工具链，例如“nds32le-elf-mculib-v5”，单击“OK”，如图 2-15 所示。

图 2-15 选择 Targets 和 Toolchain



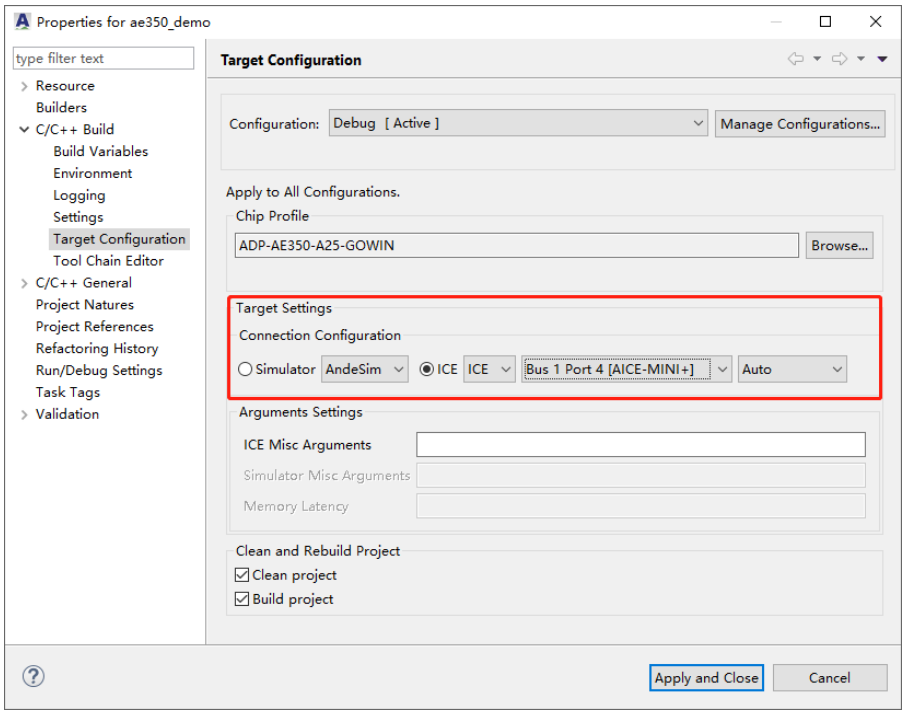
3. 选择“Target Settings”，指定连接配置，如图 2-16 所示。

   例如：

   ICE：

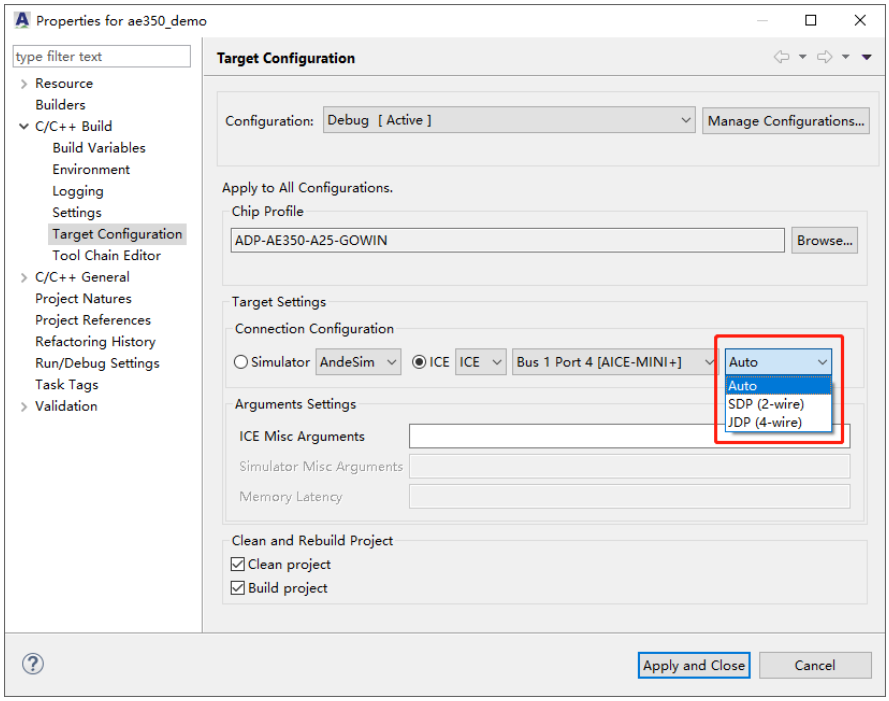
   - ICE
   - Bus 1 Port 4 [AICE-MINI+]
   - Auto

图 2-16 选择 Target Settings



如果选择连接本地目标板，RDS 软件会自动检测和显示插入的 ICE 调试器。对于 AICE-MINI+调试器，须指定目标板的调试接口，例如“SDP (2-wire)”或“Auto”，如图 2-17 所示。

图 2-17 选择 Connection Configuration

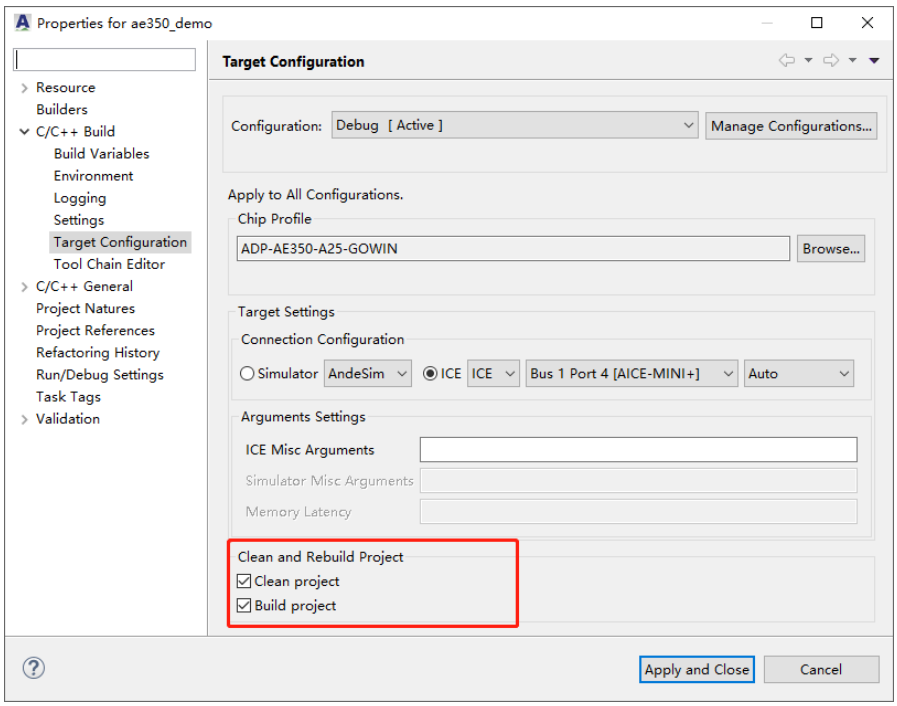


注！

如果有多个可用的 ICE 调试器，须选择一个用于此软件工程，例如“Bus 1 Port 4 [AICE-MINI+]”。

4. 选择“Clean and Rebuild Project”，指定在完成目标配置后是否清理或构建软件工程，如图 2-18 所示。“Clean project”，用于清理软件工程。“Build project”，用于构建软件工程。

图 2-18 选择 Clean and Rebuild Project



5. 单击“Apply and Close”，完成选定的软件工程的目标配置。

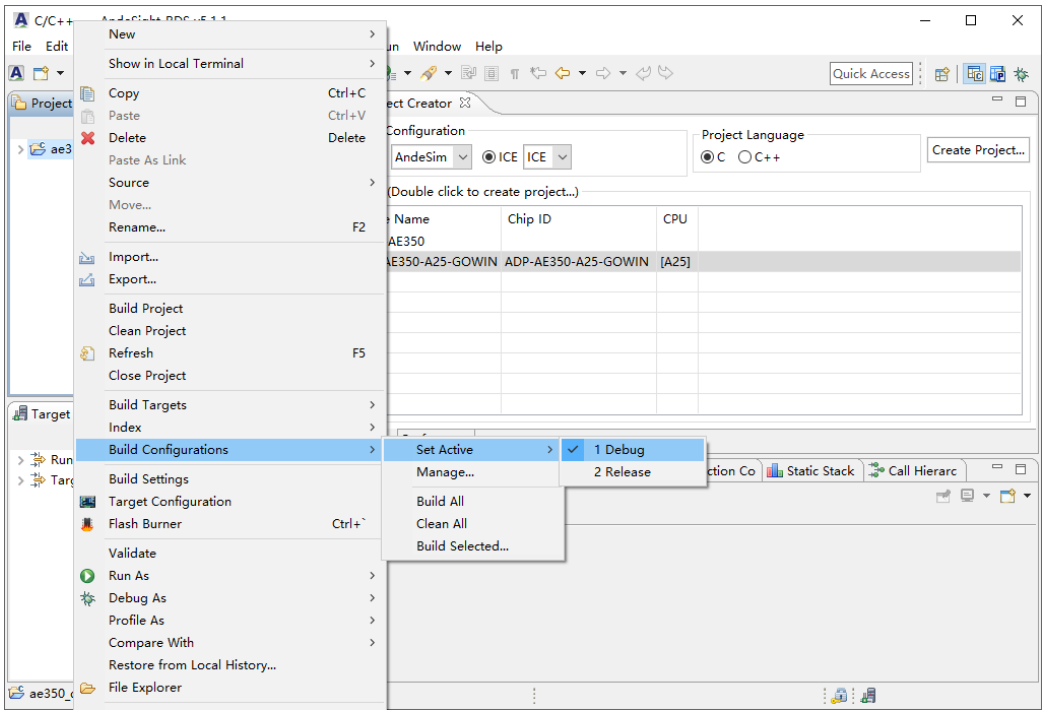
2.4 建立构建系统

使用图形化用户接口为软件工程选择一种构建配置，建立选定的软件工程的构建系统。

2.4.1 指定构建配置

在项目资源管理视图中，右单击选定的软件工程，下拉菜单中选择“Build Configurations > Set Active”，选择一种构建配置，设置其为活动状态，如图 2-19 所示。

图 2-19 选择 Build Configurations > Set Active

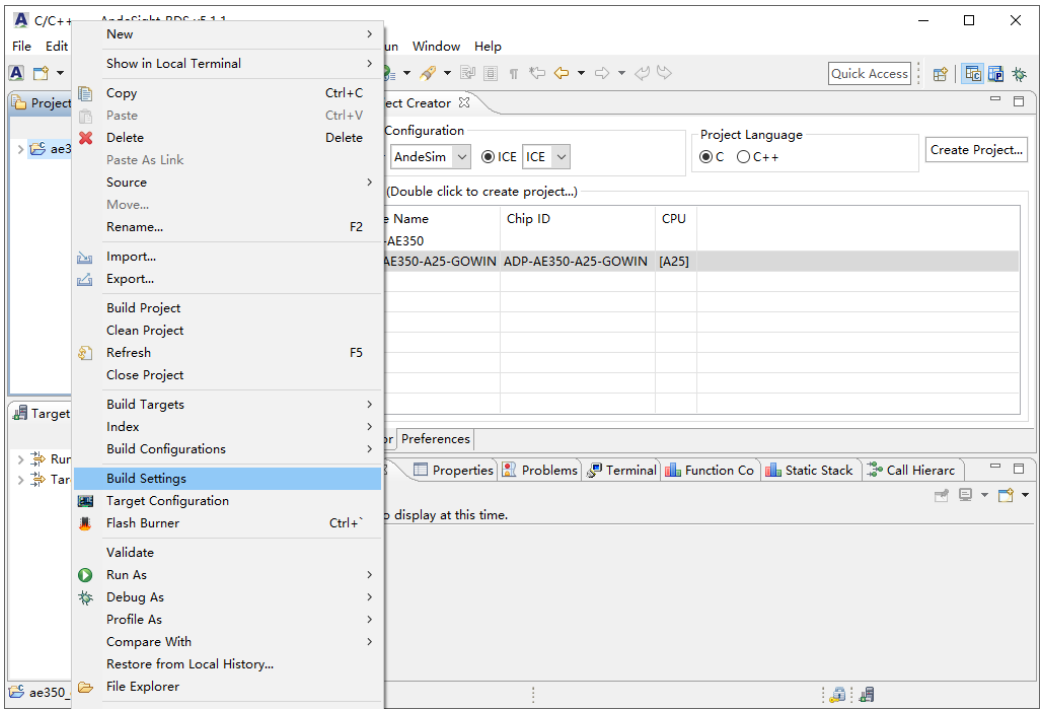


2.4.2 设置构建配置

参照以下步骤设置选定的软件工程的构建配置：

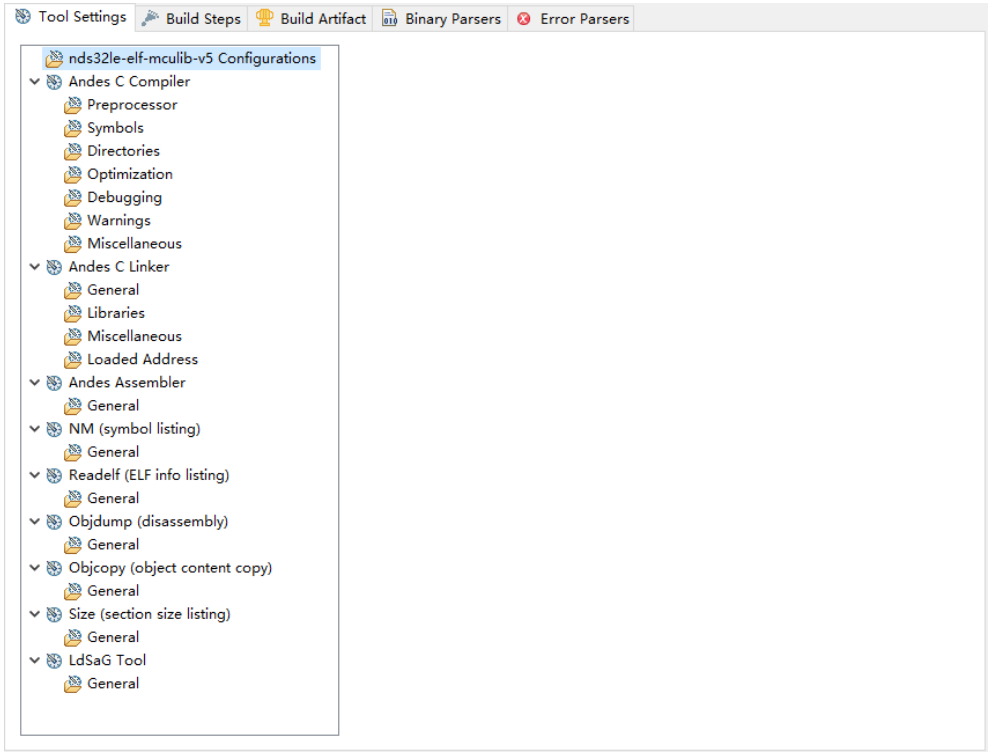
1. 在项目资源管理器视图中，右单击选定的软件工程，下拉菜单中选择“Build Settings”，如图 2-20 所示。

图 2-20 选择 Build Settings



在“Settings”对话框中，“Tool Settings”选项卡列出了软件工程所有可以配置的构建选项，参照应用需求配置各个选项，如图 2-21 所示。

图 2-21 Tool Settings 选项卡



2. 选择“Andes C Compiler > Directories > Include paths (-I)”，指定软件

工程的 C 程序引用的头文件的路径，如图 2-22 所示。

例如：

“${workspace_loc:/${ProjName}/src/bsp/ae350}”；

“${workspace_loc:/${ProjName}/src/bsp/config}”；

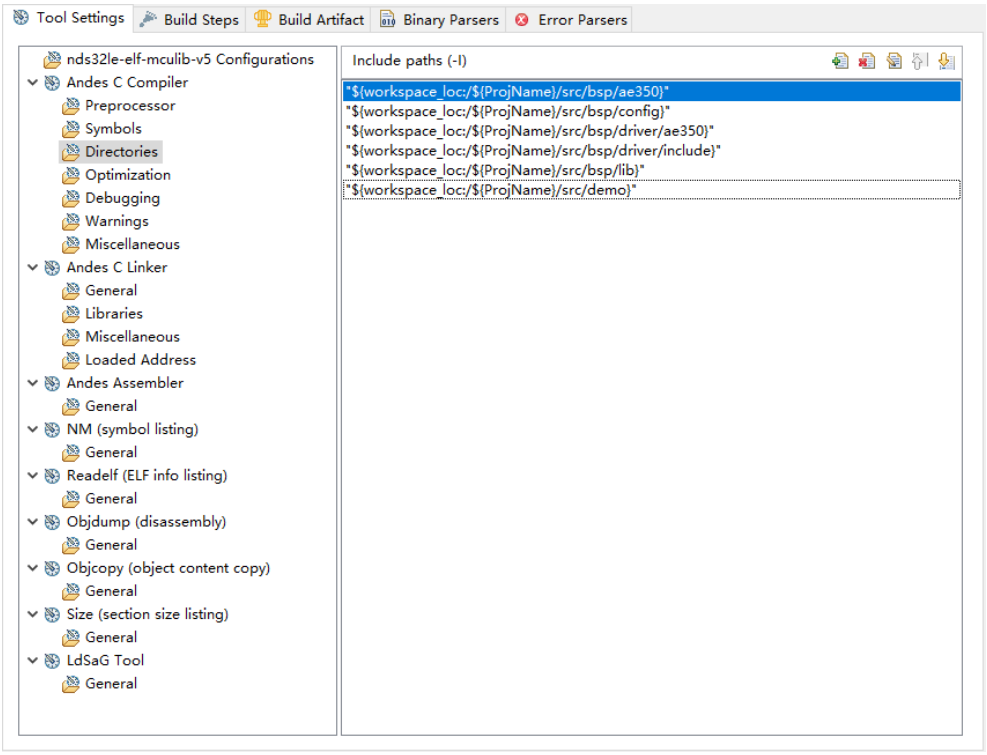
“${workspace_loc:/${ProjName}/src/bsp/driver/ae350}”；

“${workspace_loc:/${ProjName}/src/bsp/driver/include}”；

“${workspace_loc:/${ProjName}/src/bsp/lib}”；

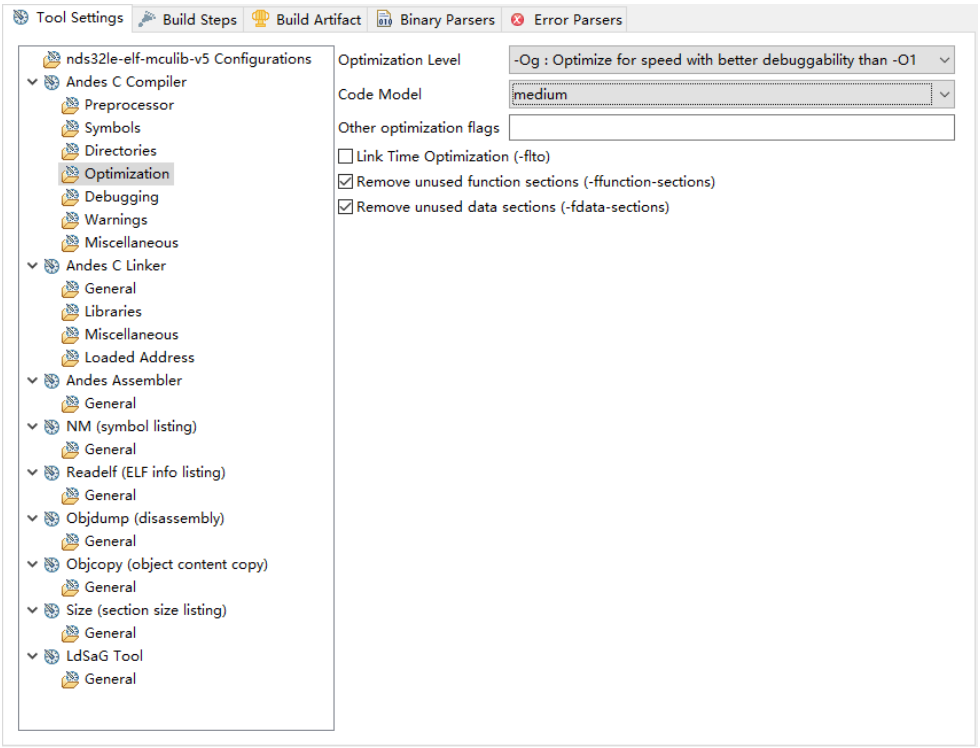
“${workspace_loc:/${ProjName}/src/demo}”。

图 2-22 选择 Andes C Compiler > Directories



3. 选择“Andes C Compiler > Optimization”，指定软件工程的优化等级和代码模型等，如图 2-23 所示。

图 2-23 选择 Andes C Compiler > Optimization



Andes C Compiler > Optimization 选项描述如表 2-3 所示。

表 2-3 Andes C Compiler > Optimization 选项

选项	配置	描述
Optimization Level	-O0: Do not optimize -Og: Optimize for speed with better debuggability than -O1 -O1: Optimize for speed -O2: Optimize more for speed -O3: Optimize most for speed -Os1: Optimize for size -Os2: Optimize more for size -Os: Optimize most for size	参照应用需求，控制软件工程的性能与代码规模的平衡。
Code Model	small medium large	如表 2-4 所示，指定代码模型，表示指令和数据规模
Other optimization flags	…	用户指定的其他优化选项
Link Time Optimization (-flto)	开启/关闭	用于优化应用程序的链接时间
Remove unused function sections (-ffunction-sections)	开启/关闭	用于删除未引用的程序指令

选项	配置	描述
Remove unused data section (-fdata-sections)	开启/关闭	用于删除未引用的程序数据

表 2-4 代码模型

代码模型 / 段	text	data	rodata
small	全 32 位寻址空间		
medium			
large			

RDS 软件的编译器有很多处理优化的编译器选项。表 2-5 列出了一些常用的编译器优化选项，用于控制不同类型的优化。

表 2-5 编译器优化选项

优化类型	选项	描述
Code size	-Os1	This option enables minimum code size optimizations. Performance is still concerned.
	-Os2	This option enables partial code size optimizations with little performance concern.
	-OS3 (-Os)	This option works the same as the -Os option to enable all code size optimizations. Performance may seriously drop with this option.
	-Oz	Supported by LLVM only, this option works the same as -Os option to enable all code size optimizations. Performance may seriously drop with this option.
Code speed	-O3	Sometimes the code size may increase dramatically after this option is applied. This is because -O3 also implies -finline-functions that can expand the content of callee within the caller (See Table 17 for enabled options at -O3). To avoid such function inlining optimization, just use the option -fno-inline-functions.
	-funroll-loops	Use with GCC: Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. The compiler has a set of heuristics to estimate whether to unroll loop or not. Use with LLVM: The compiler has a set of heuristics to estimate whether to unroll loop or not, even if the number of iterations is uncertain when the loop is entered. This option may make programs run

优化类型	选项	描述
		more slowly if it loses locality after unrolling.
	-funroll-all-loops	Use with GCC: Unroll all loops, even if their number of iterations is uncertain when the loop is entered. This option may make programs run more slowly if it loses locality after unrolling. Use with LLVM: This option is accepted but ignored by clang.
	-malways-align	Enforces 4-byte alignment on jump targets, return addresses and function entries. The two options are to prevent extra performance penalty due to misalignment. They are not default applied at -Os (including -Os1, -Os2, -Os3, and -Oz) since they may slightly increase code size.
	-mcpu or -mtune	Optimizes the program for the given processor by specification of the CPU name or series. For example, -mcpu=a25 and -mtune=andes-25-series. Both options will optimize the program according to processor feature such as pipeline latency.
Undefined behavior	-fno-delete-null-pointer-checks	In the C language standard, programs cannot safely dereference null pointers, and no code or data element resides there. However, this assumption is not true in some cases, especially for embedded platform. Thus, if you have to dereference the memory address 0x00000000, please use -fno-delete-null-pointer-checks to tell the compiler not to optimize out null pointer checking.
	-fno-strict-aliasing	In the compiler framework, it may enable strict aliasing optimization on the assumption that the strictest aliasing rules applicable to the language being compiled. If a program contains pointer casting, it may break the strict aliasing rule. Therefore, it would be better not to use pointer casting in your programs. If you must use it, having the option -fno-strict-aliasing is recommended. Otherwise, the execution result may be unexpected.
	-fwrapv	The C language standard considers the overflow of a signed value is undefined behavior. That means a valid program must never generate signed overflow when computing an expression and the compiler is able to perform some optimization under such condition. If you must have invalid code containing signed overflow, please compile it with -fwrapv, which tells the compiler to treat signed overflow as wrapping.

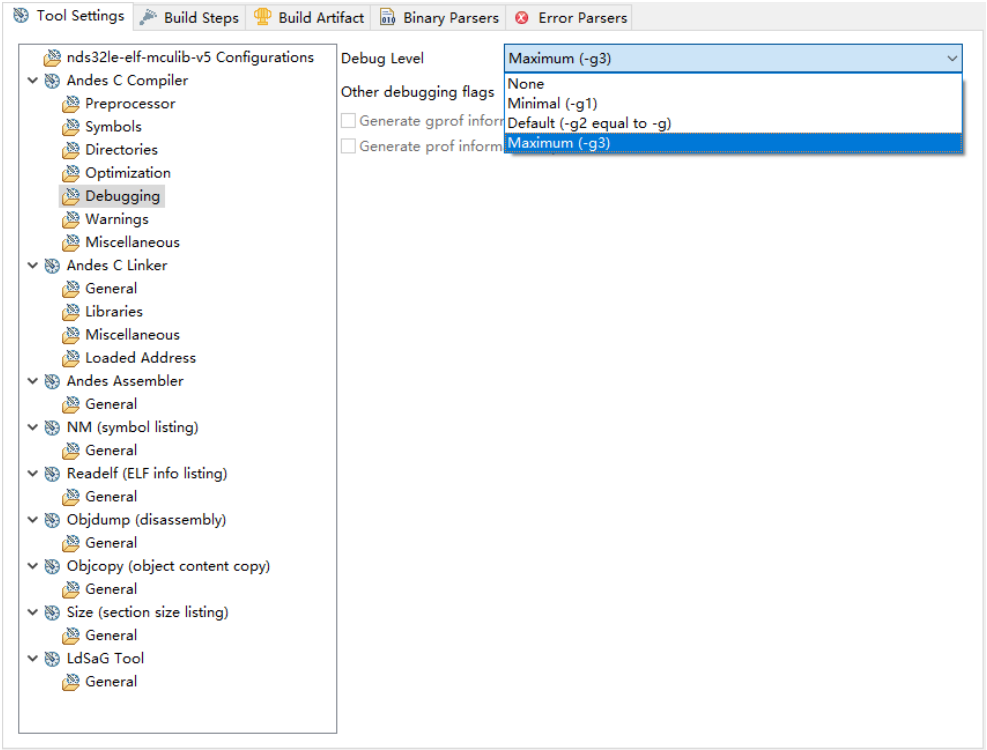
优化类型	选项	描述
	-mfma	The option -fma is used to generate floating-point fused multiply-add (FMA) instructions. It is enabled by default in RDS Compiler to increase the performance, albeit an unsafe floating-point math optimization which may decrease precision. If you need a program result compatible with IEEE 754, use -mno-fma to disable this optimization.

4. 选择“Andes C Compiler > Debugging”，指定软件工程的调试等级，如图 2-24 所示。

   例如：

   Debug Level：Maximum (-g3)

图 2-24 选择 Andes C Compiler > Debugging



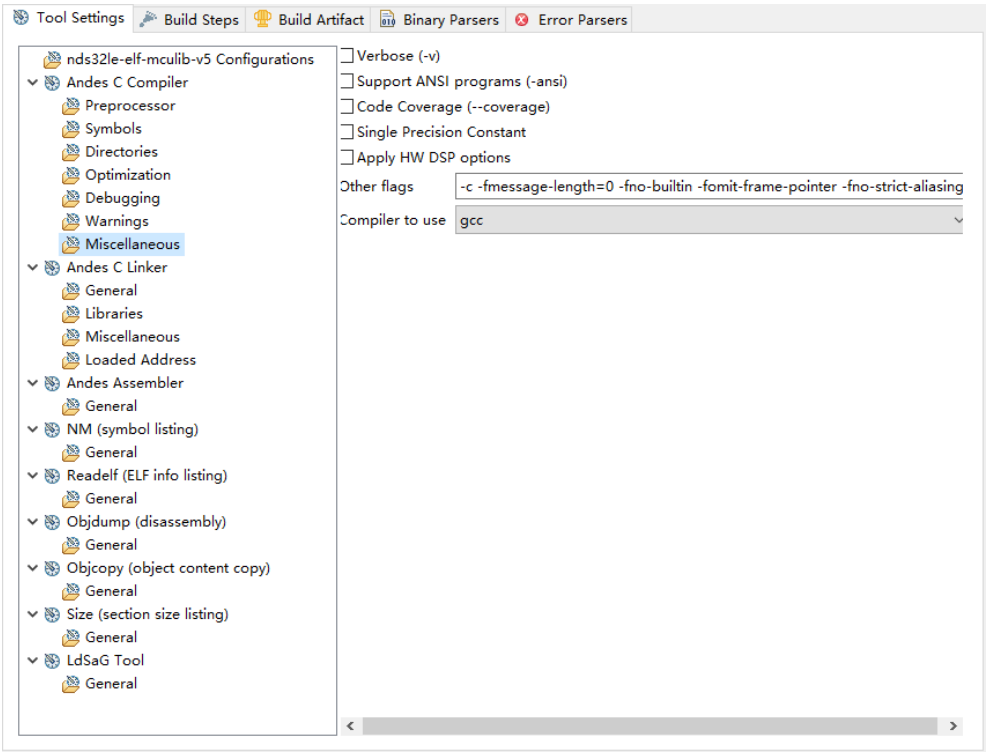
5. 选择“Andes C Compiler > Miscellaneous”，指定软件工程的其他编译器选项，如图 2-25 所示。

   例如：

   Other flags：-c -fmessage-length=0 -fno-builtin -fomit-frame-pointer -fno-strict-aliasing；

   Compiler to use：gcc

图 2-25 选择 Andes C Compiler > Miscellaneous



Andes C Compiler > Miscellaneous 选项描述如表 2-6 所示。

表 2-6 Andes C Compiler > Miscellaneous 选项

选项	配置	描述
Other flags	-c -fmessage-length=0 -fno-builtin -fomit-frame-pointer -fno-strict-aliasing	-fmessage-length=N：格式化错误信息，默认值为 0
Code Coverage (--coverage)	开启/关闭	代码覆盖率分析
Single Precision Constant	开启/关闭	浮点型常量看作单精度常量
Apply HW DSP options	开启/关闭	支持 DSP ISA 扩展选项
Compiler to use	gcc llvm	适用于同时包含 LLVM 和 GCC 编译器的 RDS RISC-V elf 工具链

RDS 软件的编译器由 GNU 和 LLVM 构建而成，继承了 GNU 的 gcc、as 和 ld 选项以及 LLVM 的 clang 和 ld 选项。除了 GNU 和 LLVM 的选项之外，还为一些特殊的特性提供了特定的选项，例如性能和代码规模的平衡。

RDS 软件的 GCC 编译器选项如表 2-7 所示。

表 2-7 GCC 编译器选项

GCC 编译器选项	描述
-pass-exit-codes	Exit with highest error code from a phase.
--help	Display this information.
--target-help	Display target specific command line options.
--version	Display compiler version information.
-dumpspecs	Display all of the built in spec strings.
-dumpversion	Display the version of the compiler.
-dumpmachine	Display the compiler's target processor.
-print-search-dirs	Display the directories in the compiler's search path.
-print-libgcc-file-name	Display the name of the compiler's companion library.
-print-file-name=<lib>	Display the full path to library <lib>.
-print-prog-name=<prog>	Display the full path to compiler component <prog>.
-print-multiarch	Display the target's normalized GNU triplet, used as a component in the library path.
-print-multi-directory	Display the root directory for versions of libgcc.
-print-multi-lib	Display the mapping between command line options and multiple library search directories.
-print-multi-os-directory	Display the relative path to OS libraries.
-print-sysroot	Display the target libraries directory.
-print-sysroot-headers-suffix	Display the sysroot suffix used to find headers.
-Wa,<options>	Pass comma-separated <options> on to the assembler.
-Wp,<options>	Pass comma-separated <options> on to the preprocessor.
-Wl,<options>	Pass comma-separated <options> on to the linker.
-Xassembler <arg>	Pass <arg> on to the assembler.
-Xpreprocessor <arg>	Pass <arg> on to the preprocessor.
-Xlinker <arg>	Pass <arg> on to the linker.
-save-temps	Do not delete intermediate files.
-save-temps=<arg>	Do not delete intermediate files.
-no-canonical-prefixes	Do not canonicalize paths when building relative prefixes to other gcc components.
-pipe	Use pipes rather than intermediate files.
-time	Time the execution of each subprocess.
-specs=<file>	Override built-in specs with the contents of <file>.

GCC 编译器选项	描述
-std=<standard>	Assume that the input sources are for <standard>.
--sysroot=<directory>	Use <directory> as the root directory for headers and libraries.
-B <directory>	Add <directory> to the compiler's search paths.
-v	Display the programs invoked by the compiler.
-E	Preprocess only; do not compile, assemble or link.
-S	Compile only; do not assemble or link.
-c -o <file>	Compile and assemble, but do not link. Place the output into <file>.
-pie	Create a position independent executable.
-shared	Create a shared library.
-mabi=	Specify integer and floating-point calling convention.
-mace	Compile with ACE.
-mace-s2s=	Argument for pass to ACE source-to-source translator.
-malways-align	Always align function entry, jump target and return address.
-march=	Generate code for given RISC-V ISA (e.g. RV64IM). ISA strings must be lower-case.
-matomic	Use atomic extension instructions.
-mbf16	Support BFLOAT16 conversion extension
-mb20282	Avoid conditional jump erratum
-mbranch-cost=N	Set the cost of branches to roughly N instructions.
-mcmodel=	Specify the code model.
-mconfig-mul=	Specify configuration of instruction mul.
-mcpu=	-mcpu=PROCESSOR Optimize the output for PROCESSOR. Same as -mtune=.
-mctor-dtor	Enable constructor/destructor feature.
-mdiv	Use hardware instructions for integer division.
-mexecit	Use special directives to guide linker to perform exec.it optimization.
-mexplicit-relocs	Use %reloc() operators, rather than assembly macros, to load addresses.
-mext-dsp	Use hardware DSP instructions.
-mext-vector	Use vector instructions.
-mext-zbabcs	Use RVB hardware instructions including zba, zbb, zbc, zbs.
-mfdiv	Use hardware floating-point divide and square root instructions.
-mfma	Generating fma instructions.

GCC 编译器选项	描述
-mgp-insn-relax	Use special directives to guide linker doing gp insn relax optimization.
-minnermost-loop	Insert the innermost loop directive.
-mmove-bytes-per-loop=	-mmove-bytes-per-loop=N Set <N> bytes of data movement can be handled per loop iteration.
-mno-16-bit	Do not generate C extension instructions.
-mplt	When generating -fpic code, allow the use of PLTs. Ignored for fno-pic.
-mpreferred-stack-boundary=	Attempt to keep stack aligned to this power of 2.
-mrelax	Take advantage of linker relaxations to reduce the number of instructions required to materialize symbol addresses.
-mrestrict-even-reg-for-regpair	Restrict register pair must start with even register number.
-msave-restore	Use smaller but slower prologue and epilogue code.
-msmall-data-limit=N	Put global and static data smaller than <number> bytes into a special section (on some targets).
-mstrict-align	Do not generate unaligned memory accesses.
-mtune=PROCESSOR	Optimize the output for PROCESSOR.
-mvh	Enable Virtual Hosting support.
-mzfh	Support Zfh half-precision floating-point extension.

RDS 软件的 LLVM 编译器选项如表 2-8 所示。

表 2-8 LLVM 编译器选项

LLVM 编译器选项	描述
-###	Print (but do not run) the commands to run for this compilation
-B <dir>	Add <dir> to search path for binaries and object files used implicitly
-c	Only run preprocess, compile, and assemble steps
-E	Only run the preprocessor
--help-hidden	Display help for hidden options
-help	Display available options
-o <file>	Write output to <file>
-no-canonical-prefixes	Use relative instead of canonical paths
-pipe	Use pipes between commands, when possible
-print-file-name=<file>	Print the full library path of <file>
-print-libgcc-file-name	Print the library path for the currently used compiler runtime library

LLVM 编译器选项	描述
	("libgcc.a" or "libclang_rt.builtins.*.a")
-print-multi-directory	Print the directory name corresponding to the supported multilib
-print-multi-lib	Print the mapping from command line options to multilib search directories
-print-prog-name=<prog>	Print the full program path of <name>
-print-search-dirs	Print the paths used for finding libraries and programs
-save-temps=<value>	Save intermediate compilation results.
-save-temps	Save intermediate compilation results
-std=<value>	Language standard to compile for
--sysroot=<value>	Set the root directory for headers and libraries
-S	Only run preprocess and compilation steps
-time	Time individual commands
--version	Print version informatio
-v	Show commands to run and use verbose output
-Wa,<arg>	Pass the comma separated arguments in <arg> to the assembler
-Wl,<arg>	Pass the comma separated arguments in <arg> to the linker
-Wp,<arg>	Pass the comma separated arguments in <arg> to the preprocessor
-Xassembler <arg>	Pass <arg> to the assembler
-Xlinker <arg>	Pass <arg> on to the linker.
-Xpreprocessor <arg>	Pass <arg> on to the preprocessor.
-x <language>	Treat subsequent input files as having type <language>
-mabi=<value>	Set the RISC-V ABI
-mace-s2s=<value>	Argument for pass to ACE source-to-source translator.
-mace	Compile with ACE.
-malways-align	Always align function entry, jump target and return address.
-march=<value>	Set the RISC-V architecture
-matomic	Use atomic extension instructions.
-mb20282	Avoid conditional jump erratum
-mb22827.1	Insert a NOP after FSHW conditionally
-mb22827	Insert a fclass.x after FDIV.x/FSQRT.x conditionally
-mbf16	Enable bf16 support
-mcmodel=<value>	Specify the code model.
-mctor-dtor	Enable constructor/destructor feature
-mdiv	Use hardware instructions for integer division

LLVM 编译器选项	描述
-mexecit	Use special directives to guide linker doing execit optimization.
-mext-dsp	Enable DSP support
-mext-zbabcs	Use RVB hardware instructions including zba, zbb, zbc, zbs.
-mfdiv	Use hardware floating-point divide and square root instructions.
-mgp-insn-relax	Use special directives to guide linker doing gp insn relax optimization.
-mict-model=<value>	Specify the address generation strategy for ICT call's code model
-minnermost-loop	Insert the innermost loop directive.
-mno-16-bit	Do not generate C extension instructions.
-mrelax	Enable linker relaxation
-msave-restore	Use smaller but slower prologue and epilogue code
-msmall-data-limit=<value>	Put global and static data smaller than <number> bytes into a special section.
-mstrict-align	Force all memory accesses to be aligned (same as mno-unaligned-access
-mvh	Enable link virtual hosting
-mzfh	Enable zfh support

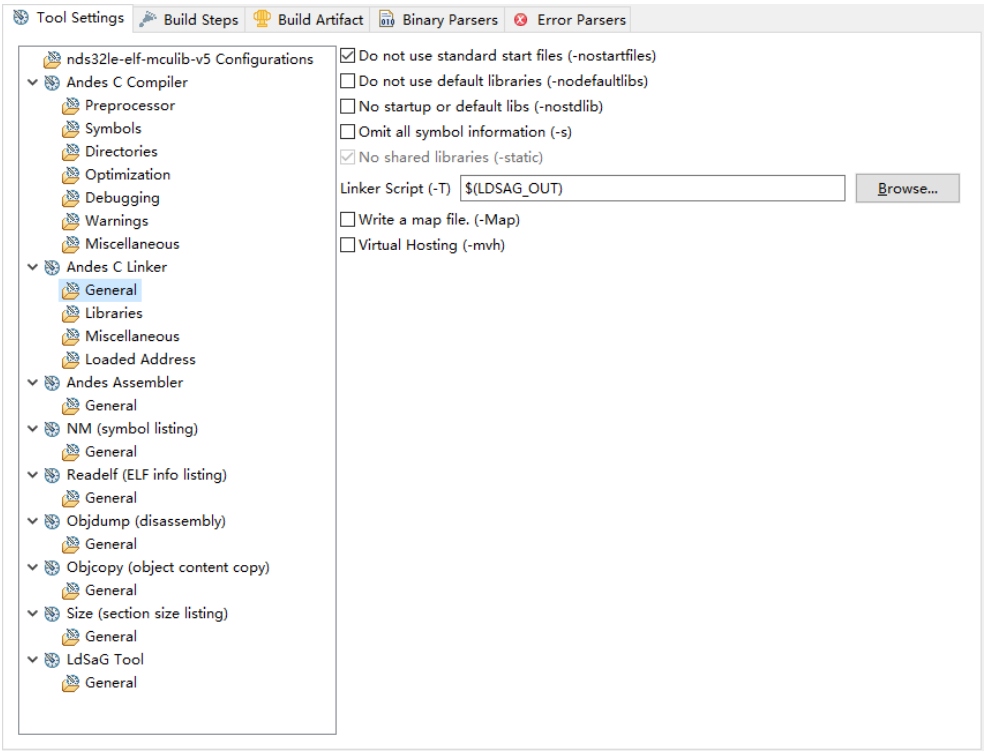
6. 选择“Andes C Linker > General”，指定软件工程的链接选项，如图 2-26 所示。

   例如：

   Do not use standard start files (-nostartfiles)：开启；

   Linker Script (-T)：$(LDSAG_OUT)

图 2-26 选择 Andes C Linker > General



Andes C Linker > General 选项描述如表 2-9 所示。

表 2-9 Andes C Linker > General 选项

选项	配置	描述
Do not use standard start files (-nostartfiles)	开启/关闭	须开启此选项，不使用标准的 start 文件
Linker Script (-T)	$(LDSAG_OUT)	关联于“LdSaG Tool > General”，如果已指定 SaG 文件，软件工程的链接脚本文件由 SaG 文件产生，否则须手动指定
Write a map file. (-Map)	开启/关闭	产生“map.txt”文件
Virtual Hosting (-mvh)	开启/关闭	激活虚拟主机，可以把没有 I/O 设备的目标板的 I/O 请求定向到主机端的 GDB

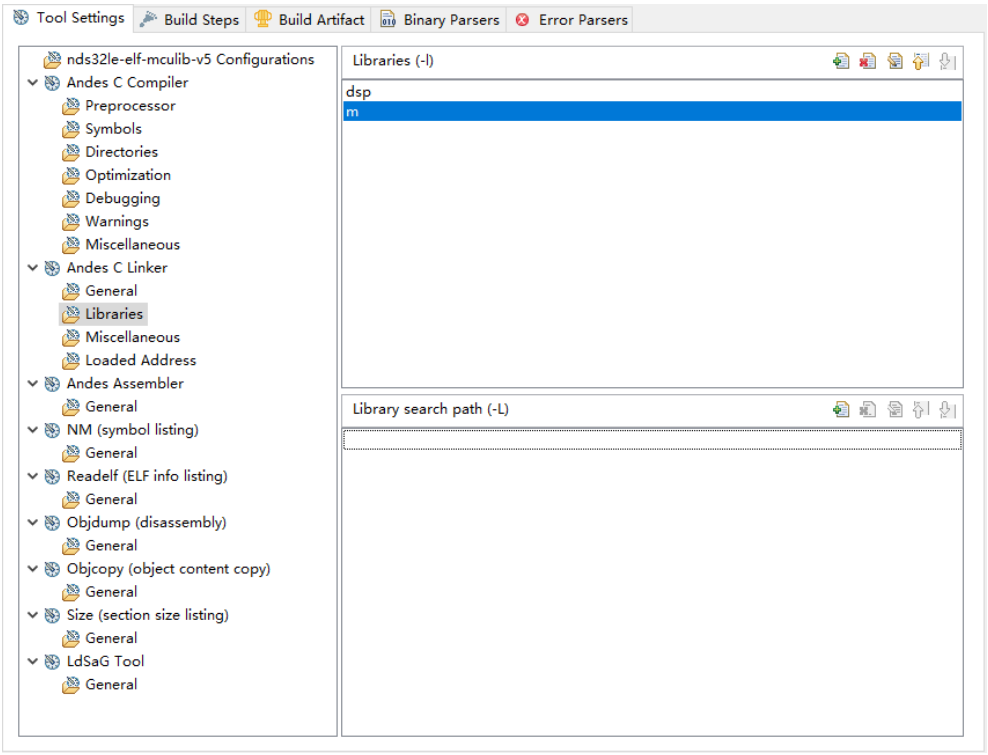
7. 选择“Andes C Linker > Libraries”，指定软件工程链接的外部软件库，如图 2-27 所示。

   例如：

   Libraries (-l)：

   - dsp
   - m

图 2-27 选择 Andes C Linker > Libraries



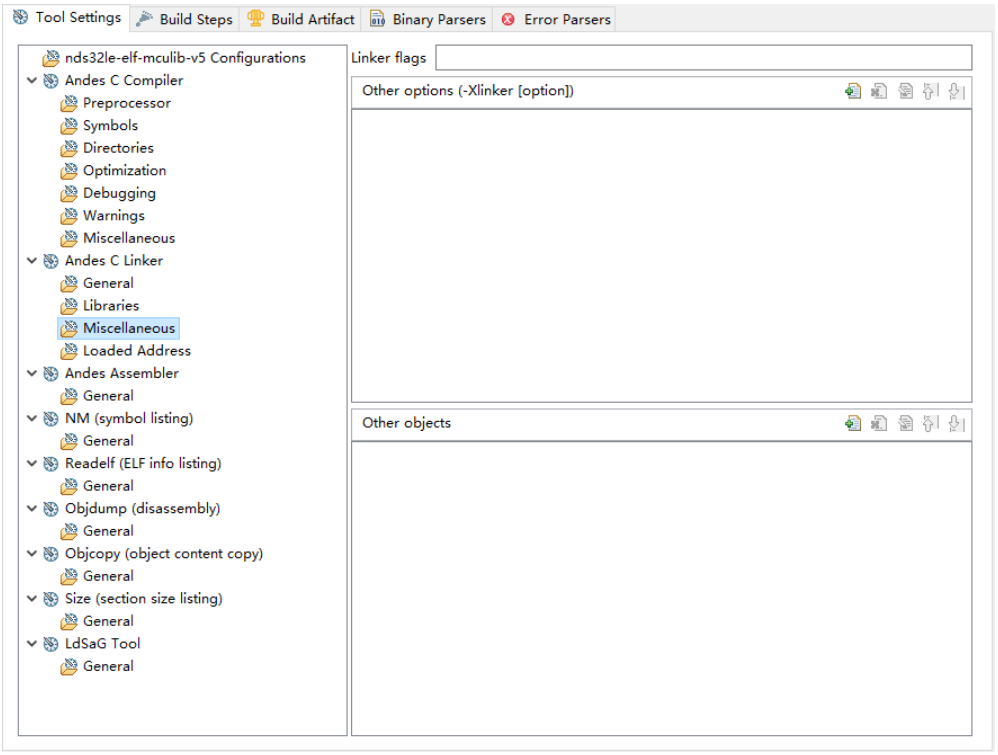
Andes C Linker > Libraries 选项描述如表 2-10 所示。

表 2-10 Andes C Compiler > Libraries 选项

选项	描述
Libraries (-l)	指定软件工程链接的外部软件库，例如 dsp library、math library 等
Library search path (-L)	外部软件库的路径

8. 选择“Andes C Linker > Miscellaneous”，指定软件工程的其他链接器选项，如图 2-28 所示。

图 2-28 选择 Andes C Linker > Miscellaneous



RDS 软件的链接器选项描述如表 2-11 所示。

表 2-11 链接器选项

链接器选项	描述
--allow-multiple-definition	Allow multiple definitions
--as-needed	Only set DT_NEEDED for shared libraries if used
--auxiliary <value>	Set DT_AUXILIARY field to the specified name
--Bdynamic	Link against shared libraries
--Bshareable	Build a shared object
--Bstatic	Do not link against shared libraries
--Bsymbolic-functions	Bind defined function symbols locally
--Bsymbolic	Bind defined symbols locally
--build-id=<value>	Generate build ID note
--build-id	Generate build ID note
--b <value>	Change the input format of the inputs following this option
--call_shared	Link against shared libraries
--color-diagnostics=<value>	Use colors in diagnostics
--color-diagnostics	Use colors in diagnostics
--compress-debug-	Compress DWARF debug sections

链接器选项	描述
sections=<value>	
--compress-debug-sections <value>	Compress DWARF debug sections
--dc	Assign space to common symbols
--define-common	Assign space to common symbols
--defsym=<value>	Define a symbol alias
--defsym <value>	Define a symbol alias
--demangle	Demangle symbol names
--disable-new-dtags	Disable new dynamic tags
--discard-all	Delete all local symbols
--discard-locals	Delete temporary local symbols
--discard-none	Keep all symbols in the symbol table
--dn	Do not link against shared libraries
--dp	Assign space to common symbols
--dynamic-linker=<value> --dynamic-linker <value>	Which dynamic linker to use Which dynamic linker to use
--dynamic-list=<value>	Read a list of dynamic symbols
--dynamic-list <value>	Read a list of dynamic symbols
--dy	Link against shared libraries
-d	Assign space to common symbols
--eh-frame-hdr	Request creation of .eh_frame_hdr section and PT_GNU_EH_FRAME segment header
--emit-relocs	Generate relocations in output
--enable-new-dtags	Enable new dynamic tags
--end-lib	End a grouping of objects that should be treated as if they were together in an archive
--entry=<entry>	Name of entry point symbol
--entry <entry>	Name of entry point symbol
--error-limit=<value>	Maximum number of errors to emit before stopping (0 = no limit)
--error-limit <value>	Maximum number of errors to emit before stopping (0 = no limit)
--error-unresolved-symbols	Report unresolved symbols as errors
--exclude-libs=<value>	Exclude static libraries from automatic export
--exclude-libs <value>	Exclude static libraries from automatic export

链接器选项	描述
--export-dynamic-symbol=<value>	Put a symbol in the dynamic symbol table
--export-dynamic-symbol <value>	Put a symbol in the dynamic symbol table
--export-dynamic	Put symbols in the dynamic symbol table
-E	Put symbols in the dynamic symbol table
-e <value>	Name of entry point symbol
--fatal-warnings	Treat warnings as errors
--filter=<value>	Set DT_FILTER field to the specified name
--filter <value>	Set DT_FILTER field to the specified name
--fini=<symbol>	Specify a finalizer function
--fini <symbol>	Specify a finalizer function
--force-exe-suffix	Force generation of file with .exe suffix
--format=<input-format>	Change the input format of the inputs following this option
--format <input-format>	Change the input format of the inputs following this option
--full-shutdown	Perform a full shutdown instead of calling _exit
-F <value>	Set DT_FILTER field to the specified name
-f <value>	Set DT_AUXILIARY field to the specified name
--gc-sections	Enable garbage collection of unused sections
--gdb-index	Generate .gdb_index section
--hash-style=<value>	Specify hash style (sysv, gnu or both)
--hash-style <value>	Specify hash style (sysv, gnu or both)
--help	Print option help
-h <value>	Set DT_SONAME
--icf-data	Enable ICF to also fold identical read only data
--icf=all	Enable identical code folding
--icf=none	Disable identical code folding
--image-base=<value>	Set the base address
--image-base <value>	Set the base address
--init=<symbol>	Specify an initializer function
--init <symbol>	Specify an initializer function
--just-symbols=<value>	Just link symbols

链接器选项	描述
--just-symbols <value>	Just link symbols
--library-path=<dir>	Add a directory to the library search path
--library-path <dir>	Add a directory to the library search path
--library=<libName>	Root name of library to use
--library <libName>	Root name of library to use
--lto-aa-pipeline=<value>	AA pipeline to run during LTO. Used in conjunction with -lto-newpm-passes
--lto-newpm-passes=<value>	Passes to run during LTO
--lto-O<opt-level>	Optimization level for LTO
--lto-partitions=<value>	Number of LTO codegen partitions
-L <value>	Add a directory to the library search path
-l <value>	Root name of library to use
--Map=<value>	Print a link map to the specified file
--Map <value>	Print a link map to the specified file
--mexecit-limit=<value>	Maximum number of entries in exec.it table
--mexecit-loop-aware	Avoid generating exec.it instruction inside loop
--mexecit	Perform exec.it optimization
--mexport-execit=<value>	Export exec.it table after linking
--mexport-symbols=<value>	Export defined global symbols in form of linker script
--mimport-execit=<value>	Import exec.it table
--mkeep-import-execit<value>	Keep exec.it table
--mno-execit	Don't perform exec.it optimization.
--mupdate-execit<value>	Update exec.it table
-M	Print a link map to the standard output
-m <value>	Set target emulation
--no-as-needed	Always DT_NEEDED for shared libraries
--no-color-diagnostics	Do not use colors in diagnostics
--no-define-common	Do not assign space to common symbols
--no-demangle	Do not demangle symbol names
--no-dynamic-linker	Inhibit output of .interp section
--no-eh-frame-hdr	Do not create .eh_frame_hdr section

链接器选项	描述
--no-gc-sections	Disable garbage collection of unused sections
--no-gdb-index	Do not generate .gdb_index section
--no-gnu-unique	Disable STB_GNU_UNIQUE symbol binding
--no-omagic	Do not set the text data sections to be writable
--no-print-gc-sections	Do not list removed unused sections
--no-relax	Do not perform linker relaxation to reduce code size
--no-rosegment	Do not put read-only non-executable sections in their own segment
--no-threads	Do not run the linker multi-threaded
--no-undefined-version	Report version scripts that refer undefined symbols
--no-undefined	Report unresolved symbols even if the linker is creating a shared library
--no-whole-archive	Restores the default behavior of loading archive members
--noinhibit-exec	Retain the executable output file whenever it is still usable
--non_shared	Do not link against shared libraries
--nopie	Do not create a position independent executable
--nostdlib	Only search directories specified on the command line
-N	Set the text and data sections to be readable and writable
--oformat <format>	Specify the binary format for the output object file
--omagic	Set the text and data sections to be readable and writable
--opt-remarks-filename <value>	YAML output file for optimization remarks
--opt-remarks-with-hotness	Include hotness informations in the optimization remarks file
--orphan-handling=<value>	Control how orphan sections are handled when linker script used
--orphan-handling <value>	Control how orphan sections are handled when linker script used
--output=<value>	Path to file to write output
--output <value>	Path to file to write output
-O <value>	Optimize output file size

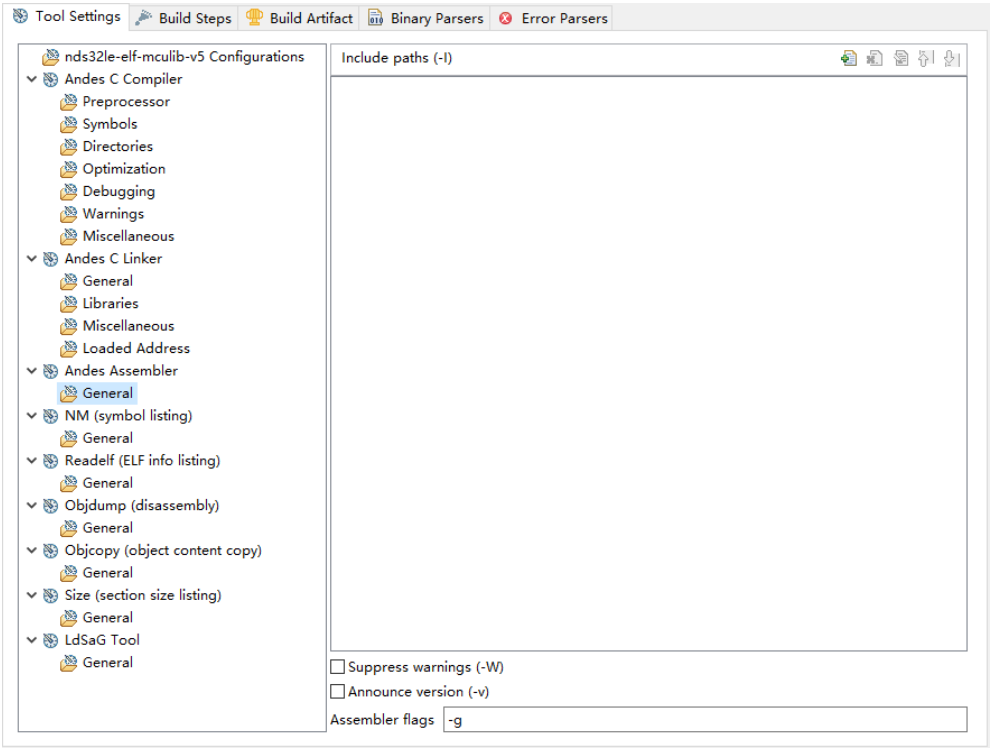
链接器选项	描述
-o <path>	Path to file to write output
--pack-dyn-relocs=<format>	Pack dynamic relocations in the given format (none or android)
--pic-executable	Create a position independent executable
--pie	Create a position independent executable
--plugin-opt=<value>	specifies LTO options for compatibility with GNU linkers
--plugin-opt <value>	specifies LTO options for compatibility with GNU linkers
--print-gc-sections	List removed unused sections
--print-map	Print a link map to the standard output
-q	Generate relocations in output
--relax	Perform linker relaxation to reduce code size
--relocatable	Create relocatable object file
--reproduce=<value>	Dump linker invocation and input files for debugging
--reproduce <value>	Dump linker invocation and input files for debugging
--retain-symbols-file=<file>	Retain only the symbols listed in the file
--retain-symbols-file <file>	Retain only the symbols listed in the file
--rpath=<value>	Add a DT_RUNPATH to the output
--rpath <value>	Add a DT_RUNPATH to the output
--rsp-quoting=<value>	Quoting style for response files. Values supported: windows|posix
--rsp-quoting <value>	Quoting style for response files. Values supported: windows|posix
-R <value>	Alias to --rpath or --just-symbols
-r	Create relocatable object file
--script=<value>	Read linker script
--script <value>	Read linker script
--section-start <address>	Set address of section
--shared	Build a shared object
--soname=<value>	Set DT_SONAME
--soname <value>	Set DT_SONAME
--sort-section=<value>	Specifies sections sorting rule when linkerscript is used
--sort-section <value>	Specifies sections sorting rule when linkerscript is

链接器选项	描述
	used
--start-lib	Start a grouping of objects that should be treated as if they were together in an archive
--static	Do not link against shared libraries
--strip-all	Strip all symbols
--strip-debug	Strip debugging information
--symbol-ordering-file <value>	Layout sections in the order specified by symbol file
--sysroot=<value>	Set the system root
--sysroot <value>	Set the system root
-S	Strip debugging information
-s	Strip all symbols
--Tbss=<value>	Same as --section-start with .bss as the sectionname
--Tbss <value>	Same as --section-start with .bss as the sectionname
--Tdata=<value>	Same as --section-start with .data as the sectionname
--Tdata <value>	Same as --section-start with .data as the sectionname
--thinlto-cache-dir=<value>	Path to ThinLTO cached object file directory
--thinlto-cache-policy <value>	Pruning policy for the ThinLTO cache
--thinlto-jobs=<value>	Number of ThinLTO jobs
--threads	Run the linker multi-threaded
--trace-symbol=<value>	Trace references to symbols
--trace-symbol <value>	Trace references to symbols
--trace	Print the names of the input files
--Ttext-segment=<value>	Same as --section-start with .text as the sectionname
--Ttext-segment <value>	Same as --section-start with .text as the sectionname
--Ttext=<value>	Same as --section-start with .text as the sectionname
--Ttext <value>	Same as --section-start with .text as the sectionname

链接器选项	描述
-T <value>	Read linker script
-t	Print the names of the input files
--undefined=<value>	Force undefined symbol during linking
--undefined <value>	Force undefined symbol during linking
--unresolved-symbols=<value>	Determine how to handle unresolved symbols
--unresolved-symbols <value>	Determine how to handle unresolved symbols
-u <value>	Force undefined symbol during linking
--verbose	Verbose mode
--version-script=<value>	Read a version script
--version-script <value>	Read a version script
--version	Display the version number and exit
-V	Display the version number and exit
-v	Display the version number
--warn-common	Warn about duplicate common symbols
--warn-unresolved-symbols	Report unresolved symbols as warnings
--whole-archive	Force load of all members in a static library
--wrap=<symbol>	Use wrapper functions for symbol
--wrap <symbol>	Use wrapper functions for symbol
-X	Delete temporary local symbols
-x	Delete all local symbols
-y <value>	Trace references to symbols
-z <option>	Linker option extensions

9. 选择“Andes Assembler > General”，指定软件工程的汇编程序引用的头文件的路径和汇编器选项，如图 2-29 所示。

图 2-29 选择 Andes Assembler > General



RDS 软件的汇编器选项描述如表 2-12 所示。

表 2-12 汇编器选项

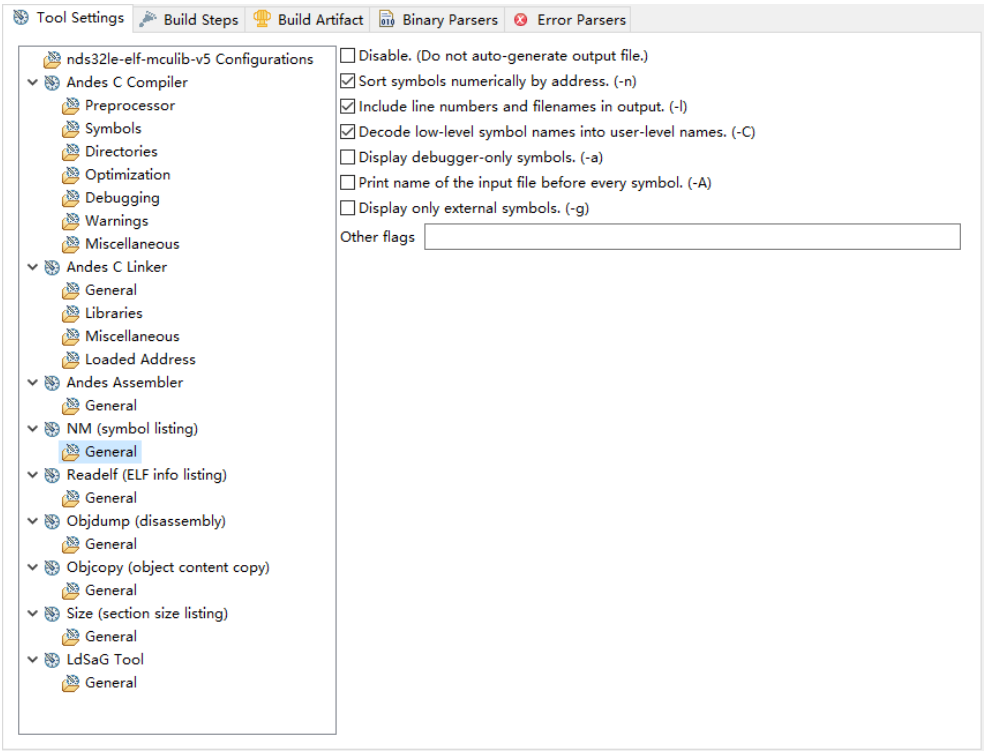
汇编器选项	描述
-a[sub-option...]	turn on listings Sub-options [default hls]: c omit false conditionals d omit debugging directives g include general info h include high-level source l include assembly m include macro expansions n omit forms processing s include symbols =FILE list to FILE (must be last sub-option)
--alternate	initially turn on alternate macro syntax
--compress-debug-sections[={none|zlib|zlib-gnu|zlib-gabi}]	compress DWARF debug sections using zlib
--nocompress-debug-sections	don't compress DWARF debug sections [default]
-D	produce assembler debugging messages

汇编器选项	描述
--debug-prefix-map OLD=NEW	map OLD to NEW in debug information
--defsym SYM=VAL	define symbol SYM to given value
--execstack	require executable stack for this object
--noexecstack	don't require executable stack for this object
--size-check=[error|warning]	ELF .size directive check (default --size-check=error)
--elf-stt-common=[no|yes]	generate ELF common symbols with STT_COMMON type
--sectname-subst	enable section name substitution sequences
-f	skip whitespace and comment preprocessing
-g --gen-debug	generate debugging information
--gdwarf-2	generate DWARF2 debugging information
--gdwarf-sections	generate per-function section names for DWARF line information
--hash-size=<value>	set the hash table size close to <value>
--help	show this message and exi
--target-help	show target specific options
-I DIR	add DIR to search list for .include directives
-J	don't warn about signed overflow
-K	warn when differences altered for long displacements
-L,--keep-locals	keep local symbols (e.g. starting with `L')
-M,--mri	assemble in MRI compatibility mode
--MD FILE	write dependency information in FILE (default none)
-nocpp	ignored
-no-pad-sections	do not pad the end of sections to alignment boundaries
-o OBJFILE	name the object-file output OBJFILE (default a.out)
-R	fold data section into text section
--reduce-memory-overheads	prefer smaller memory use at the cost of longer assembly times
--statistics	print various measured statistics from execution
--strip-local-absolute	strip local absolute symbols
--traditional-format	Use same format as native assembler when possible
--version	print assembler version number and exit
-W --no-warn	suppress warnings
--warn	don't suppress warnings
--fatal-warnings	treat warnings as errors

汇编器选项	描述
-Z	generate object file even after errors
--listing-lhs-width	set the width in words of the output data column of the listing
--listing-lhs-width2	set the width in words of the continuation lines of the output data column; ignored if smaller than the width of the first line
--listing-rhs-width	set the max width in characters of the lines from the source file
--listing-cont-lines	set the maximum number of continuation lines used for the output data column of the listing
-fpic	generate position-independent code
-fPIC	same as -fpic
-fno-pic	don't generate position-independent code (default)
-march=ISA	set the RISC-V architecture
-mabi=ABI	set the RISC-V ABI
-mno-16-bit	don't generate rvc instructions
-matomic	enable atomic extension
-mace	Support user defined instruction extension
-O1	optimize for performance
-Os	optimize for space
-mext-dsp	enable dsp extension

10. 选择“NM (symbol listing) > General”，输出符号列表信息到“symbol.txt”文件，如图 2-30 所示。

图 2-30 选择 NM (symbol listing) > General



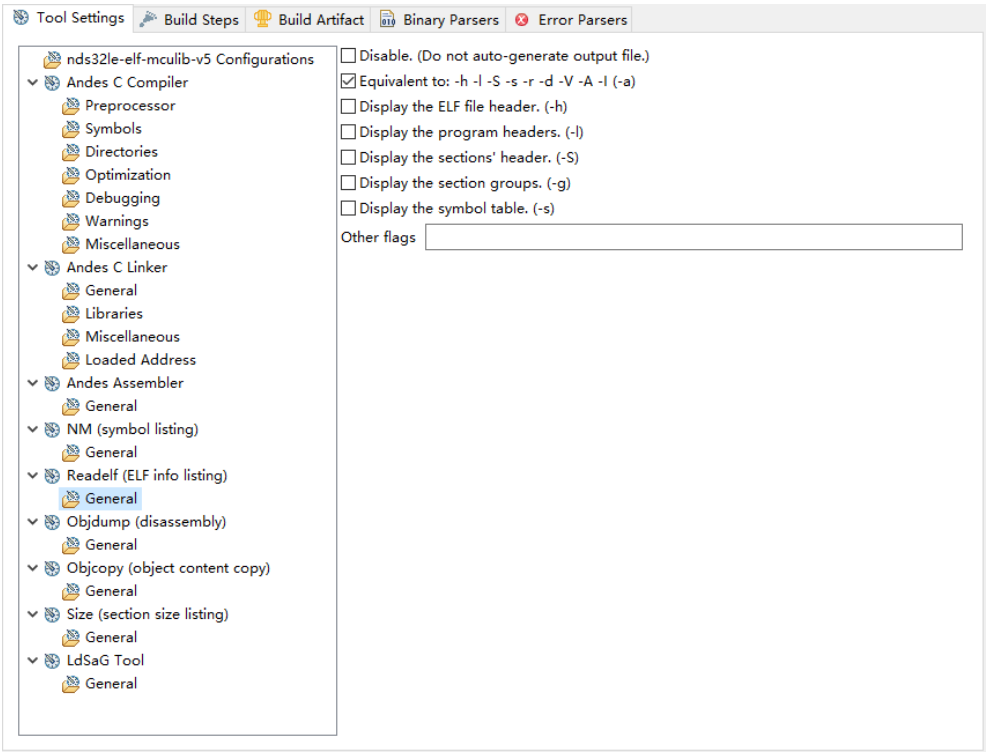
NM (symbol listing) > General 选项描述如表 2-13 所示。

表 2-13 NM (symbol listing) > General 选项

选项	描述
Disable. (Do not auto-generate output file.)	关闭符号列表输出文件
Sort symbols numerically by address. (-n)	按地址排序符号数字
Include line numbers and filenames in output. (-l)	输出信息包含行号和文件名
Decode low-level symbol names into user-level names. (-C)	低级符号解码为用户级名称
Display debugger-only symbols. (-a)	显示调试专用符号
Print name of the input file before every symbol. (-A)	在每个符号之前打印输入文件的名称
Display only external symbols. (-g)	只显示外部符号

11. 选择“Readelf (ELF info listing) > General”，输出 ELF 列表信息到“readelf.txt”文件，如图 2-31 所示。

图 2-31 选择 Readelf (ELF info listing) > General



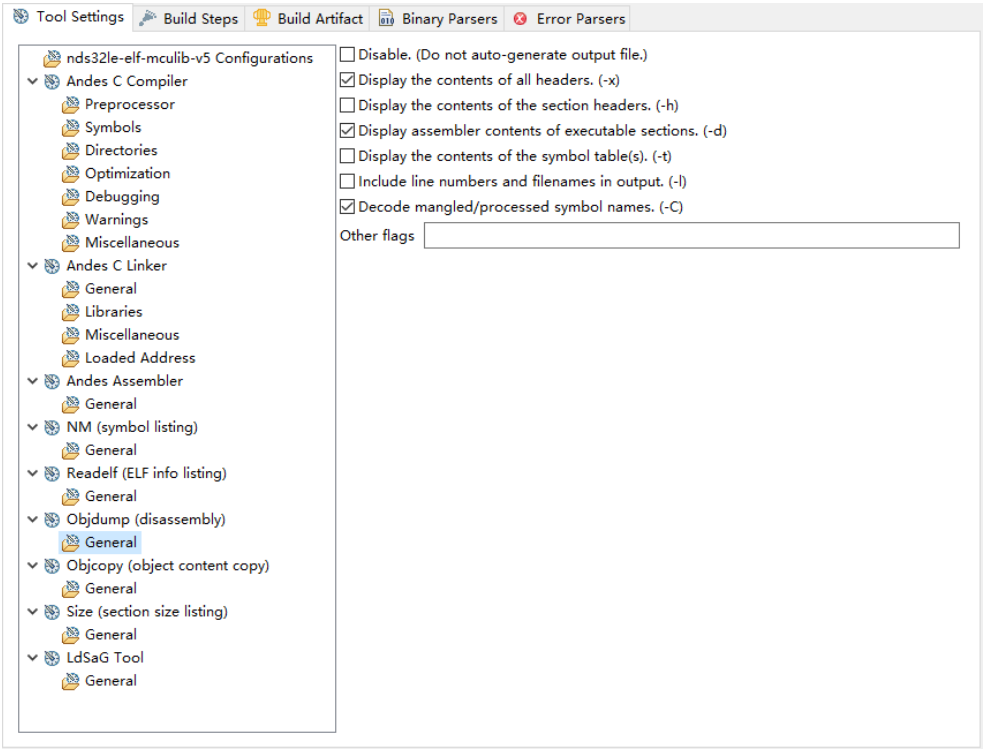
Readelf (ELF info listing) > General 选项描述如表 2-14 所示。

表 2-14 Readelf (ELF info listing) > General 选项

配置	描述
Disable. (Do not auto-generate output file.)	关闭 ELF 列表输出文件。
Display the ELF file header. (-h)	显示 ELF 文件头信息。
Display the program headers. (-l)	显示程序头信息。
Display the sections' header. (-S)	显示段头信息。
Display the section groups. (-g)	显示段组。
Display the symbol table. (-s)	显示符号表。

12. 选择“Objdump (disassembly) > General”，输出反汇编信息到“objdump.txt”文件，如图 2-32 所示。

图 2-32 选择 Objdump (disassembly) > General



Objdump (disassembly) > General 选项描述如表 2-15 所示。

表 2-15 Objdump (disassembly) > General 选项

配置	描述
Disable. (Do not auto-generate output file.)	关闭反汇编输出文件
Display the contents of all headers. (-x)	显示所有头信息的内容
Display the contents of the section headers. (-h)	显示段头信息的内容
Display assembler contents of executable sections. (-d)	显示可执行段的汇编程序的内容
Display the contents of the symbol table(s). (-t)	显示符号表的内容
Include line numbers and filenames in output. (-l)	输出信息包含行号和文件名
Decode mangled/processed symbol names. (-C)	解码重整/处理过的符号名称

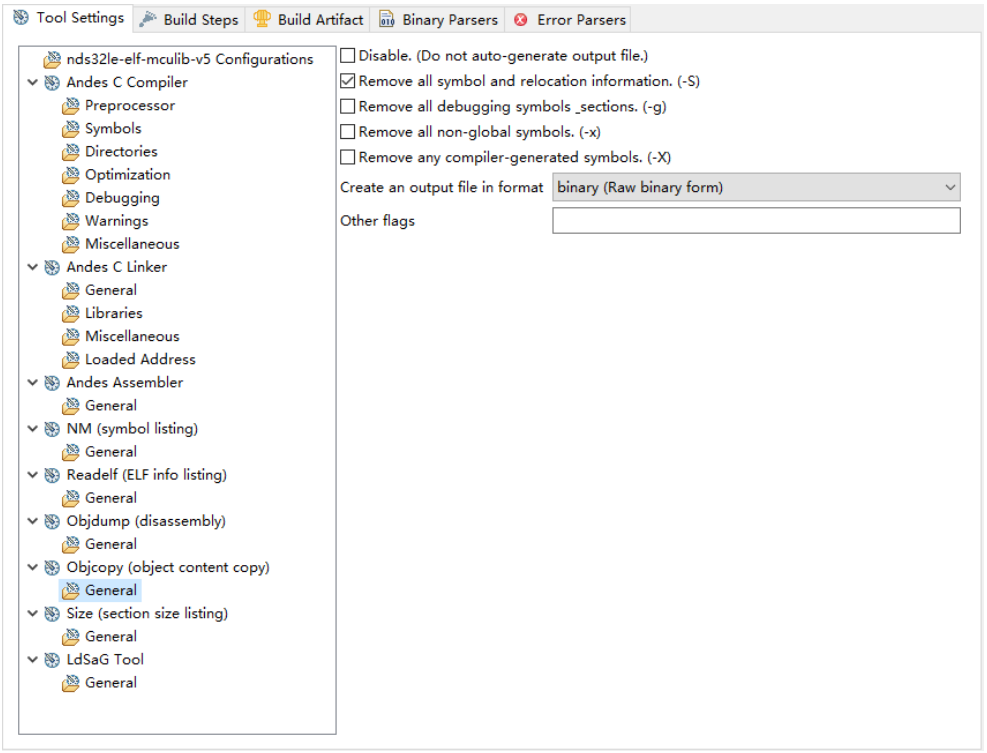
13. 选择“Objcopy (object content copy) > General”，复制目标文件的内容到另一个文件，一般用于产生 Binary 文件，如图 2-33 所示。

    例如：

    Disable. (Do not auto-generate output file.)：关闭；

    Create an output file in format：binary (Raw binary form)

图 2-33 选择 Objcopy (object content copy) > General



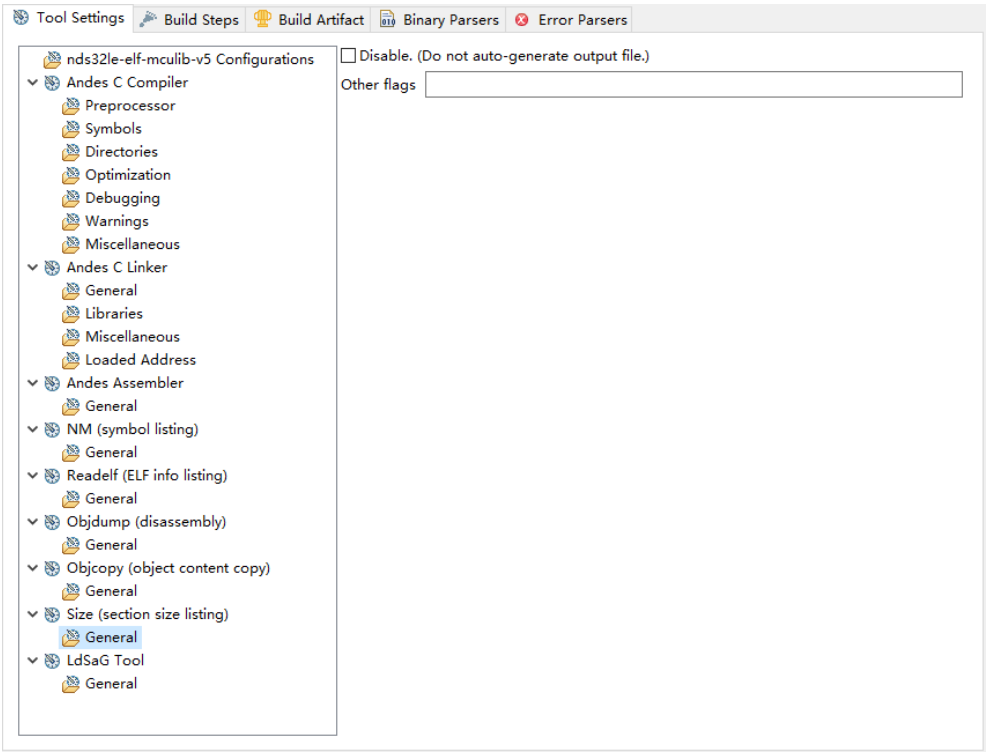
Objcopy (object content copy) > General 选项描述如表 2-16 所示。

表 2-16 Objcopy (object content copy) > General 选项

选项	描述
Disable. (Do not auto-generate output file.)	关闭 Object 输出文件
Remove all symbol and relocation information. (-S)	删除所有符号和重定位信息
Remove all debugging symbols _sections. (-g)	删除所有调试符号段
Remove all non-global symbols. (-x)	删除所有非全局符号
Remove any compiler-generated symbols. (-X)	删除任何编译器产生的符号
Create an output file in format	指定目标文件的格式，支持以下 3 种行业标准格式： binary (Raw binary form) ihex (Intel Hex) srec (Motorola S-record)

14. 选择“Size (section size listing) > General”，输出段大小列表信息到“.PHONY.size”文件，如图 2-34 所示。

图 2-34 选择 Size (section size listing) > General



Size (section size listing) > General 选项描述如表 2-17 所示。

表 2-17 Size (section size listing) > General 选项

选项	描述
Disable. (Do not auto-generate output file.)	关闭 size 输出文件。

15. 选择“LdSaG Tool > General”，指定 SaG 文件，用于产生软件工程的链接脚本文件，如图 2-35 所示。

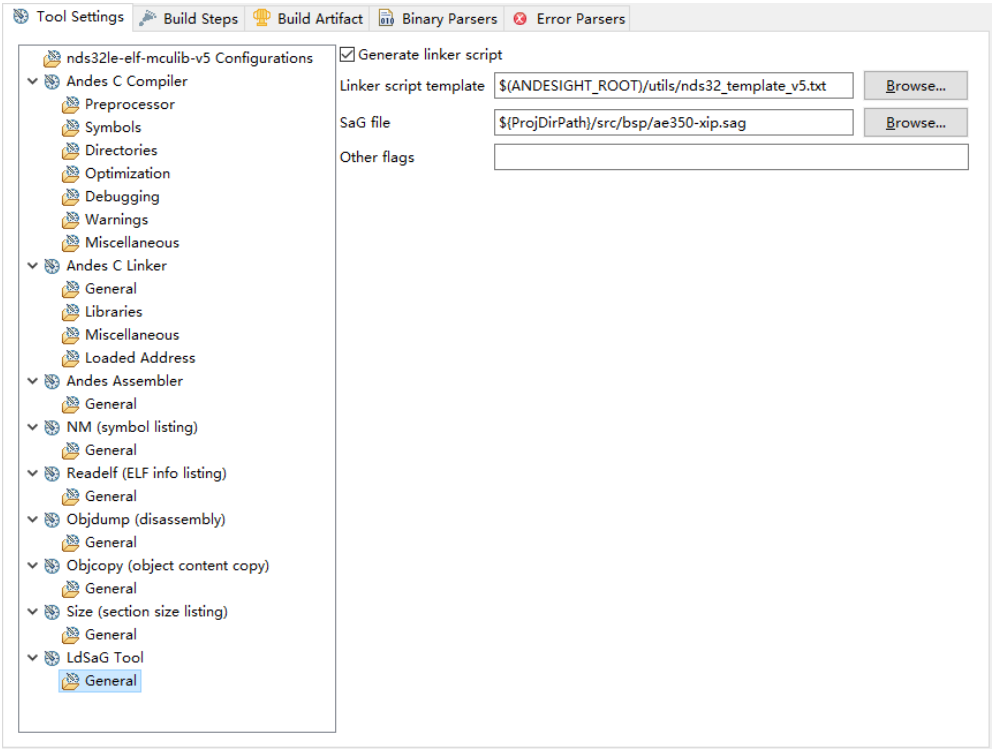
    例如：

    Generate linker script：开启；

    Linker script template：$(ANDESIGHT_ROOT)/utils/nds32_template_v5.txt；

    SaG file：${ProjDirPath}/src/bsp/sag/ae350-ddr.sag

图 2-35 选择 LdSaG Tool > General



LdSaG Tool > General 选项描述如表 2-18 所示。

表 2-18 LdSaG Tool > General 选项

选项	描述
Generate linker script	“PROJECT\BUILD_CONFIG\output”路径产生一个链接脚本文件，“Andes C Linker > General > Linker Script (-T):”选项自动设置为“$(LDSAG_OUT)”
Linker script template	指定用于产生链接脚本文件的模板，RiscV_AE350_SOC 默认为“nds32_template_v5.txt”
SaG file	指定一个 SaG 文件，描述可执行映像的内存映射

16. 完成软件工程的构建配置后，单击“Apply and Close”。

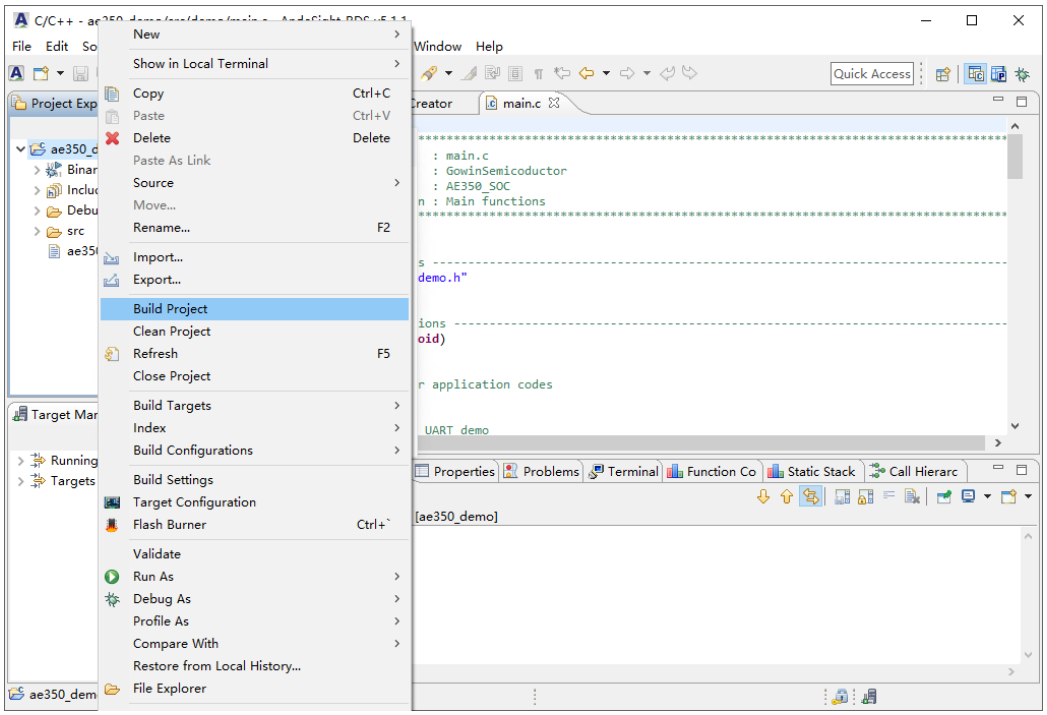
2.5 构建软件工程

构建软件工程指转换源代码为指定目标系统的可执行文件。

参照以下步骤构建软件工程：

1. 参照 2.4 建立构建系统建立选定的软件工程的构建系统。
2. 单击 RDS 软件工具栏“”（Build），或右单击选定的软件工程，下拉菜单中选择“Build Project”，构建软件工程，如图 2-36 所示。

图 2-36 构建软件工程



3. 在控制台视图中验证软件工程的构建结果。
4. “PROJECT\BUILD_CONFIG\output”路径，找到可执行文件和产生的其他文件。

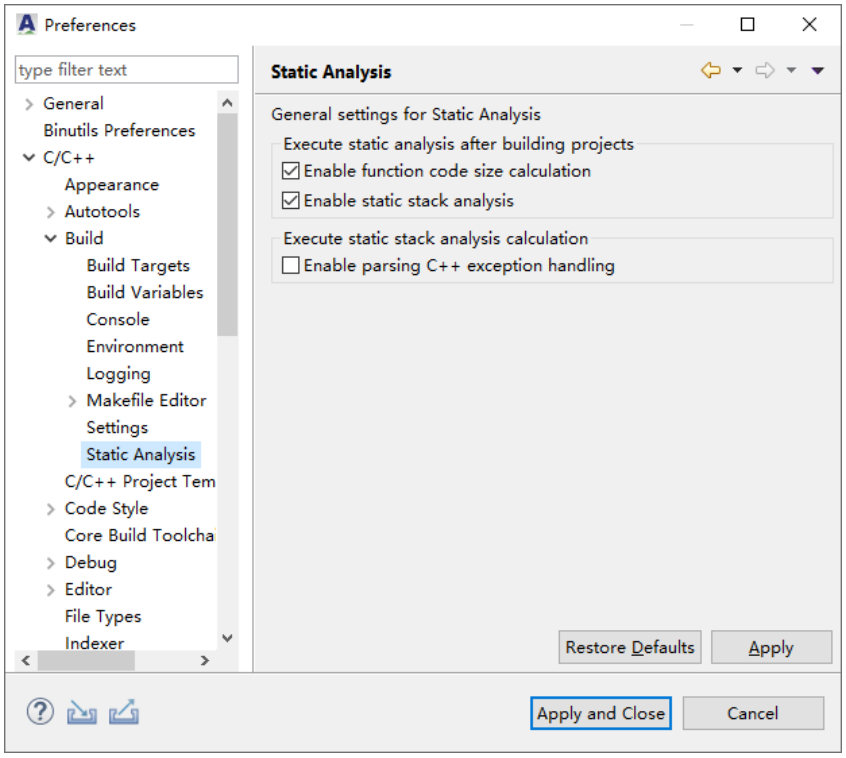
2.6 静态分析

RDS 软件以选定的软件工程的可执行文件为基础，支持以下两种类型的静态程序分析：函数代码规模分析和静态堆栈分析。函数代码规模分析提供单个函数的代码规模和用户定义的所有函数的代码规模。静态堆栈分析提供每个函数和整个程序所用的堆栈大小。

构建选定的软件工程后，RDS 软件就会自动执行静态分析。所以在构建软件工程之前，参照以下步骤设置相关属性，开启静态分析：

1. 选择 RDS 软件主菜单“Window > Preferences”，打开“Preferences”对话框。
2. “Preferences”对话框中，选择“C/C++ > Build > Static Analysis”。
3. “Static Analysis”对话框中，选择“Enable function code size calculation”和/或“Enable static stack analysis”，单击“Apply and Close”，如图 2-37 所示。

图 2-37 静态分析设置

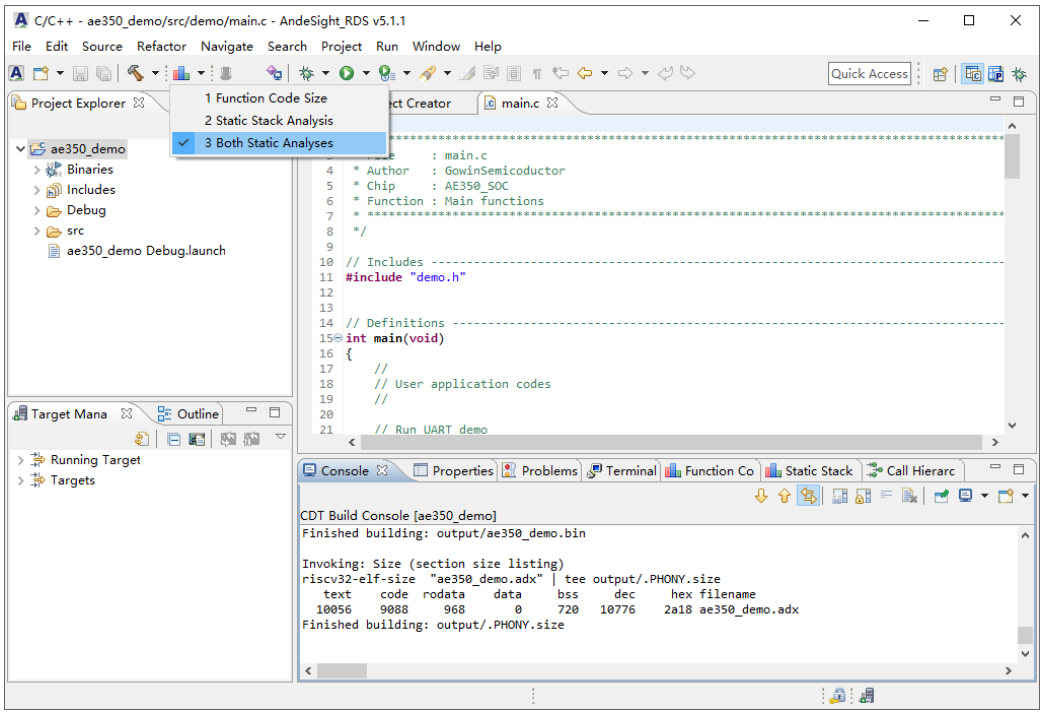


注！

C++程序的静态分析默认不包含异常处理相关的代码，如需分析所有代码，请选择“Enable parsing C++ exception handling”。

如果静态分析默认是关闭的，则可以在构建软件工程后，单击 RDS 软件工具栏“ ”下拉列表，选择“1 Function Code Size”、“2 Static Stack Analysis”或“3 Both Static Analyses”，如图 2-38 所示。

图 2-38 开启静态分析

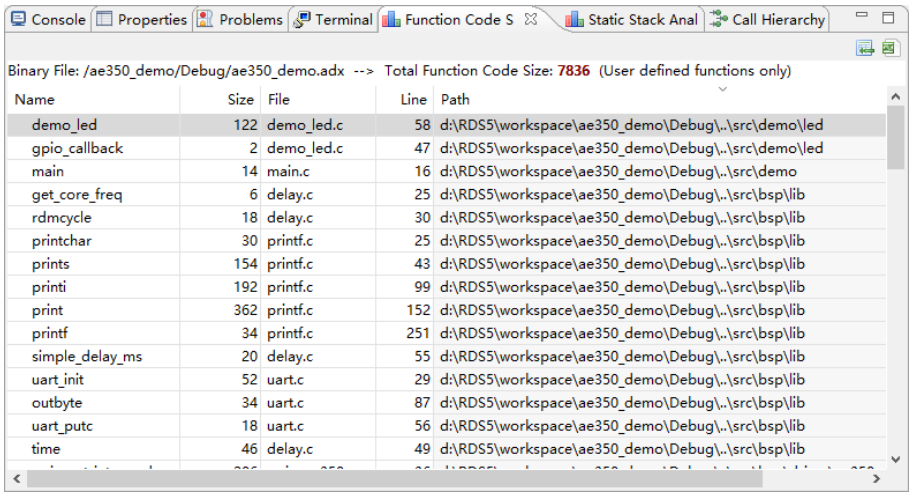


开启静态分析时，会弹出对话框提示分析过程正在进行中。静态分析完成后，函数代码规模视图（Function Code Size View）和/或静态堆栈分析视图（Static Stack Analysis View）显示静态分析的结果。

2.6.1 函数代码规模视图

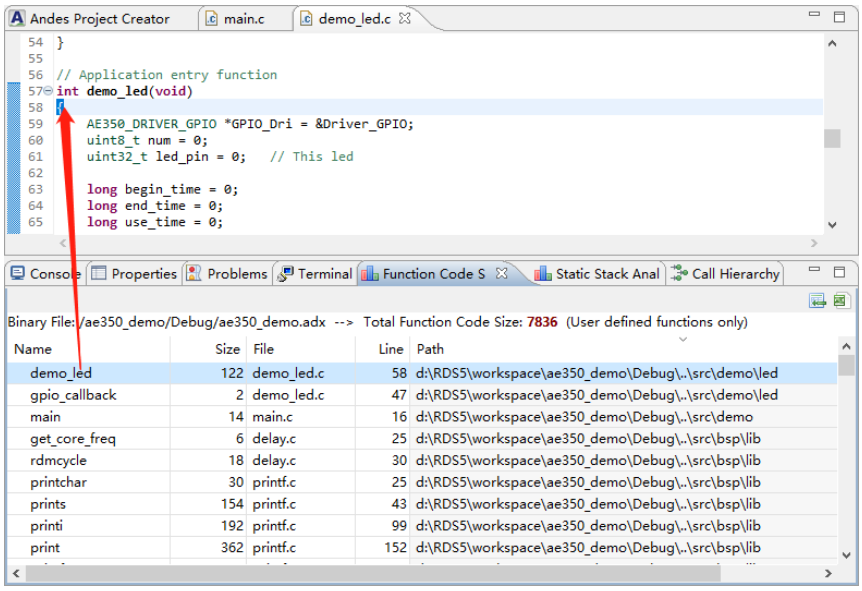
函数代码规模视图，用于查看构建的软件工程的函数规模等有关信息。此视图显示所有用户定义的函数的规模和单个函数的规模，如图 2-39 所示。

图 2-39 函数代码规模视图



此视图还显示每个函数的代码细节，通过单击列标题可以按名称或大小对函数排序，也可以双击一个函数直接定位到其代码的文件位置，如图图 2-40 所示。

图 2-40 定位函数代码位置



注！

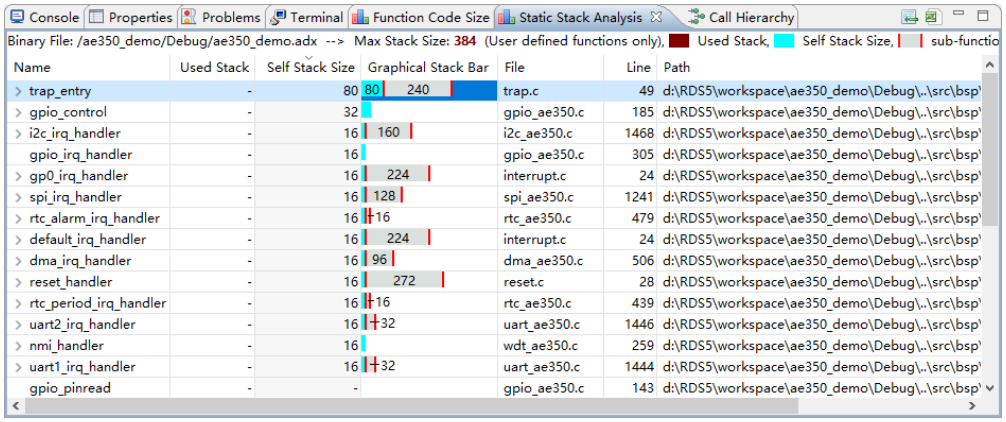
对于构建过程中被优化的函数，此视图不提供其代码规模的信息。此外，视图中“Total

Function Code Size”只包含“.text”的段，如果想要整个程序完整的函数规模，请合计“.text 段”和“.nds32_init”段。

2.6.2 静态堆栈分析视图

静态堆栈分析视图用于查看构建的软件工程的静态堆栈大小，如图 2-41 所示。产生分析结果后，此视图顶部列出整个程序所需堆栈的最大值，函数调用关系图列出源代码的详细信息。注意中断服务程序和间接调用的函数的堆栈大小，不计入所需堆栈的最大值中。

图 2-41 静态堆栈分析视图

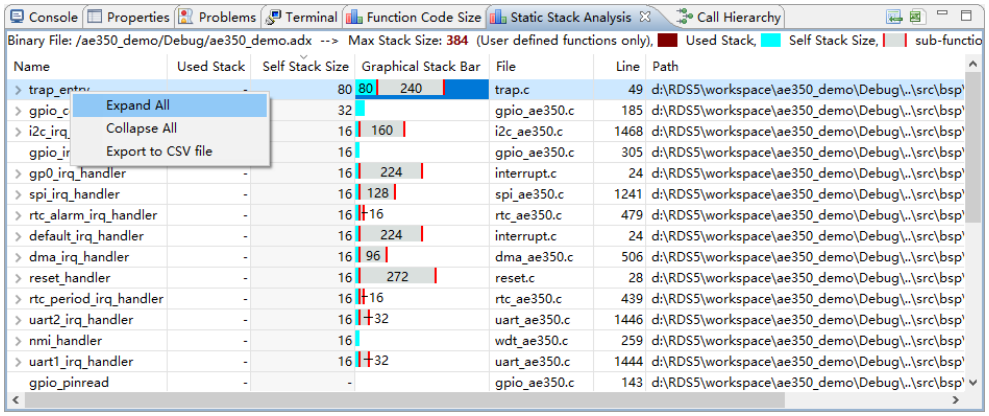


对于每个函数，此视图提供以下堆栈信息：

- Self Stack Size：当前函数使用的堆栈的大小
- Used Stack：当前函数被调用前，已使用的堆栈的大小
- Graphical Stack Bar：图形化显示当前函数的堆栈的累计使用情况，此栏列出“Self Stack Size”、“Used Stack”、“sub-function max required stack”和“Max Stack Size”。分别以不同颜色表示 3 种堆栈大小：“ ”表示“Self Stack Size”，“ ”表示“Used Stack”，“ ”表示“sub-function max required stack”

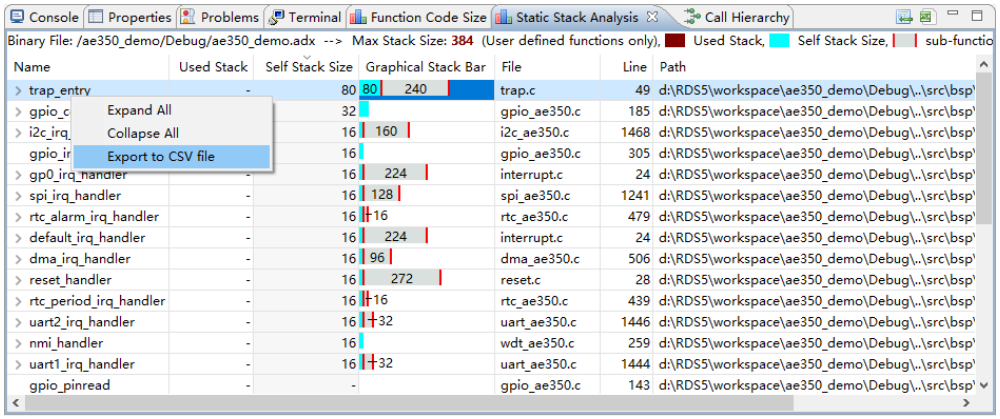
第一次调用时，此视图显示根函数的堆栈信息，包括入口函数、中断服务程序和间接调用的函数。右单击一个根函数，下拉菜单中选择“Expand all”，展开显示所有子函数及其堆栈信息，如图 2-42 所示。此堆栈信息的调用关系图，用于识别出使用堆栈最多的函数执行路径。

图 2-42 展开所有子函数



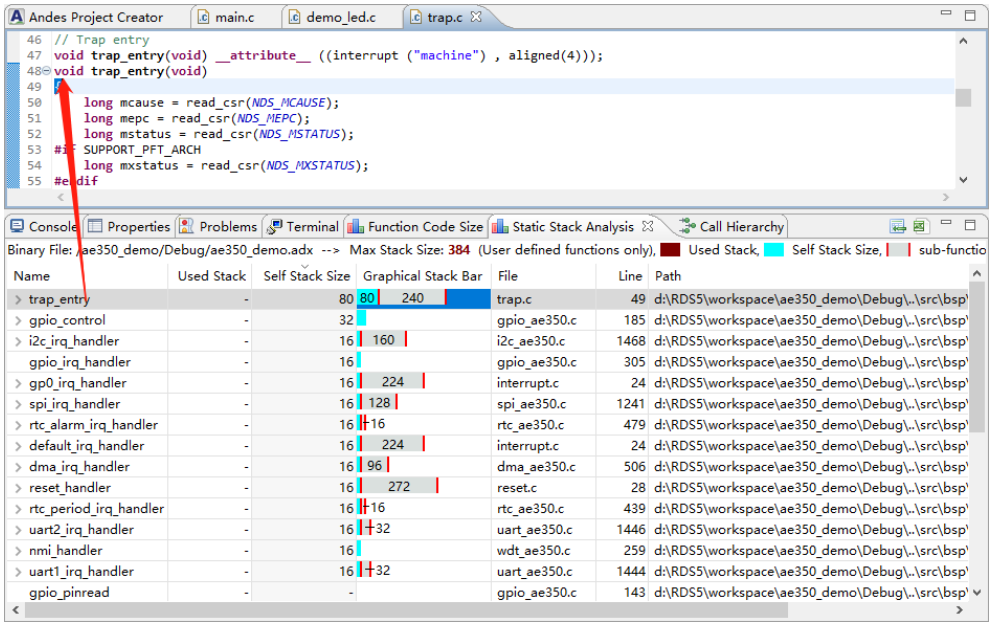
右单击一个根函数，下拉菜单中选择“Export to CSV file”，引出堆栈信息及其子函数堆栈分析到一个.csv 文件，如图 2-43 所示。

图 2-43 引出 CSV 文件



在静态堆栈分析视图中可以通过单击列标题，按名称、大小、“Self Stack Size”或“Used Stack”对函数排序，也可以双击一个函数直接定位到代码文件位置，如图 2-44 所示。

图 2-44 定位函数代码位置



3 目标管理

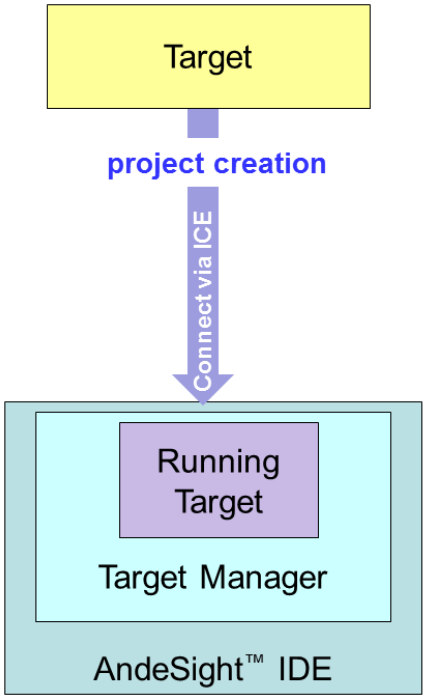
RDS 软件通过目标管理器视图（Target Manager View）管理目标（Target）、目标板（Target Board）和目标系统（Target System），具体参照 3.2 目标管理器。

以下步骤概括了 RDS 软件目标管理的典型方法：

1. 建立准备运行应用程序的目标板
2. 创建软件工程（具体参见 2.1 创建软件工程），用于目标板
3. 构建软件工程（具体参见 2.5 构建软件工程），产生可执行文件
4. 启动目标板，运行/调试/分析应用程序，由目标管理器视图管理

RDS 软件的典型目标管理流程，如图 3-1 所示。

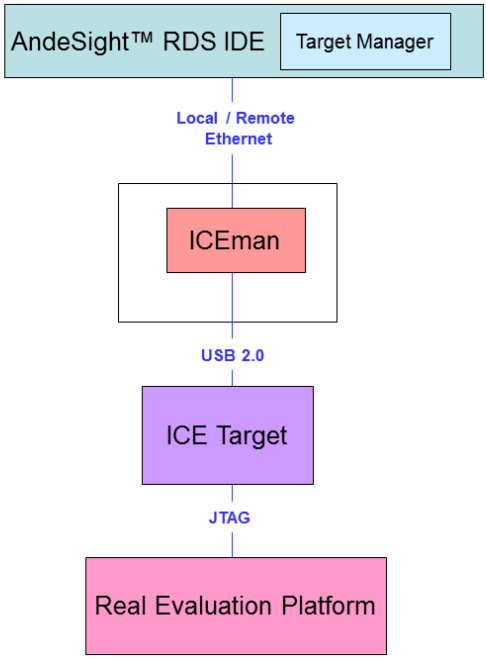
图 3-1 目标管理流程



3.1 建立目标系统

目标系统是一个通过 ICE 调试器连接到本地目标板的真实评估平台，包括目标、目标板和主机 RDS 软件。建立目标系统前，参照图 3-2 建立 RDS 软件与目标板的物理连接。

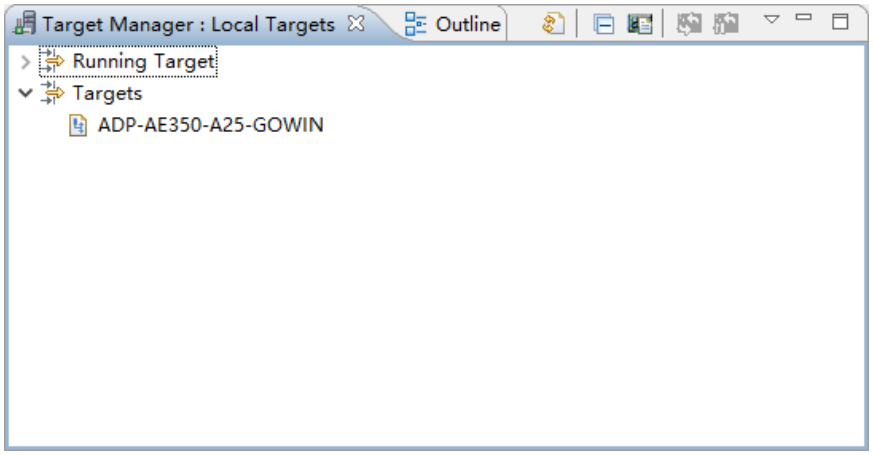
图 3-2 RDS 软件与目标板物理连接



3.2 目标管理器

RDS 软件使用目标管理器（Target Manager）来管理 RDS 软件与目标板的连接，如图 3-3 所示。目标管理器视图中，“Targets”显示 RDS 软件所支持的目标。

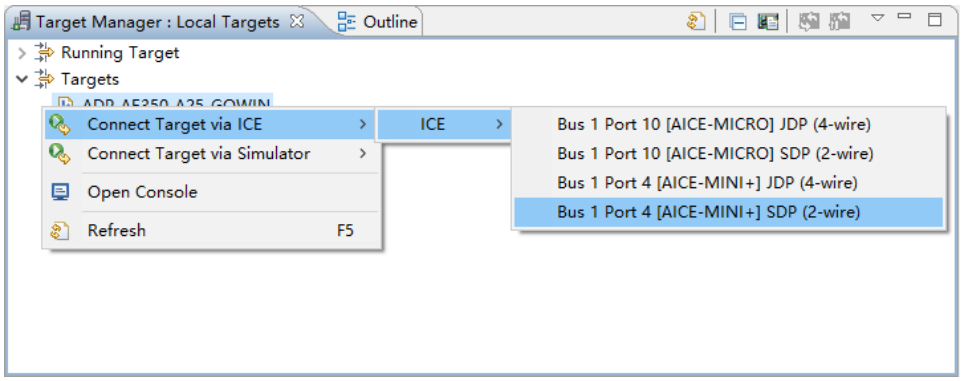
图 3-3 目标管理器



目标管理器在后台自动建立和管理主机 RDS 软件与目标板之间的通

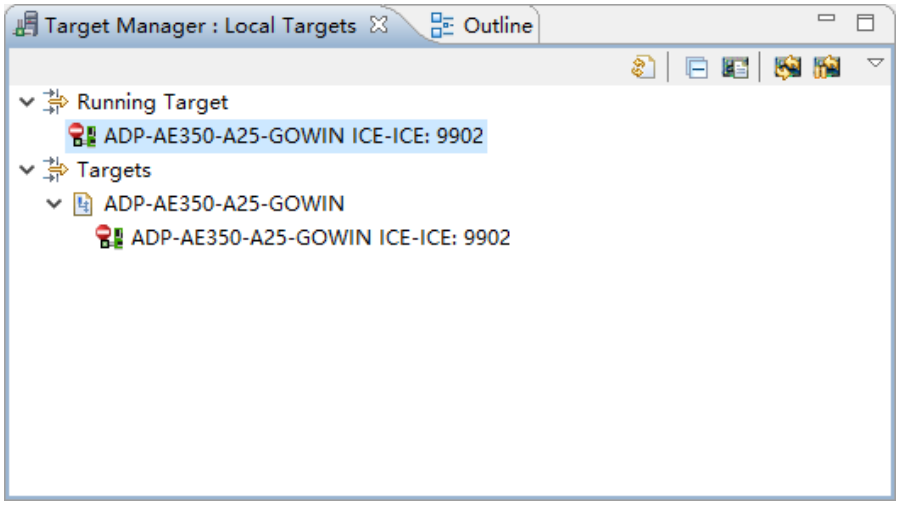
信，也可以右单击“Targets”中选定的目标，下拉菜单中选择“Connect Target via ICE > ICE”，下拉菜单中指定连接配置，例如“Bus 1 Port 4 [AICE-MINI+] SDP (2-wire)”，手动启动目标系统。为启动目标系统，须进一步指定插入的 ICE 调试器和调试接口等信息，如图 3-4 所示。

图 3-4 建立目标系统



启动目标系统后，“Targets”显示连接类型、连接配置和用于 GDB 连接的端口号，如图 3-5 所示，“Running Target”也显示相同的信息。

图 3-5 启动目标系统



目标管理器视图还会显示每个目标系统的连接状态，包括：

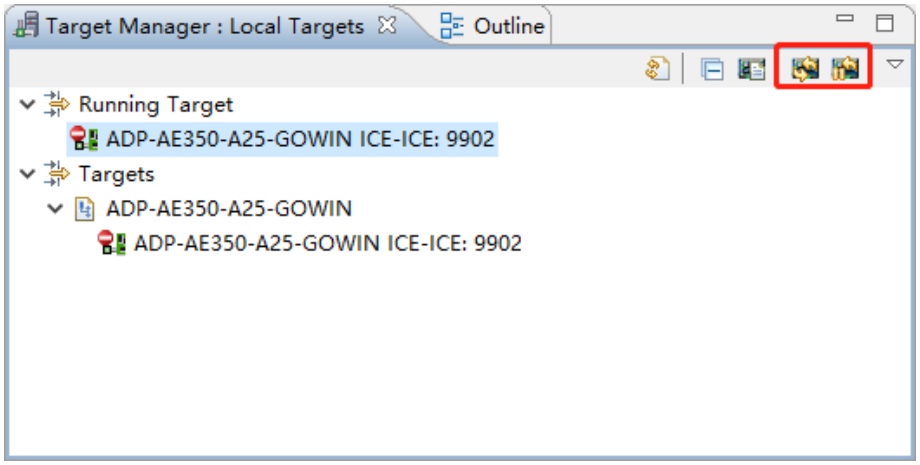
- 可用状态“ ”：目标系统处于可用状态，可以连接和开发。
- 已启动状态“ ”：目标板已连接到主机 RDS 软件，但还没有加载程序。
- 忙状态“ ”：目标系统已启动，程序已加载，已开始程序开发。

3.3 复位目标板

RDS 软件提供一个用户接口，用于在发生异常时复位目标板。

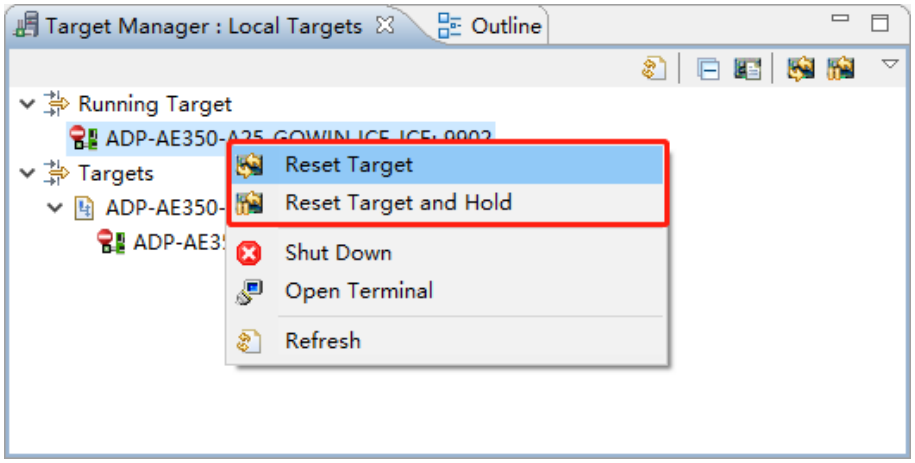
复位正在运行的目标板，在目标管理器视图中，单击工具栏“ ”复位目标板或“ ”复位目标板并保持目标在 PC=0 的位置，如图 3-6 所示。如果在程序调试阶段复位，则调试线程将会被关闭。

图 3-6 复位目标板



右单击选定的目标，下拉菜单中选择“Reset Target”或“Reset Target and Hold”，也可以复位目标板，如图 3-7 所示。

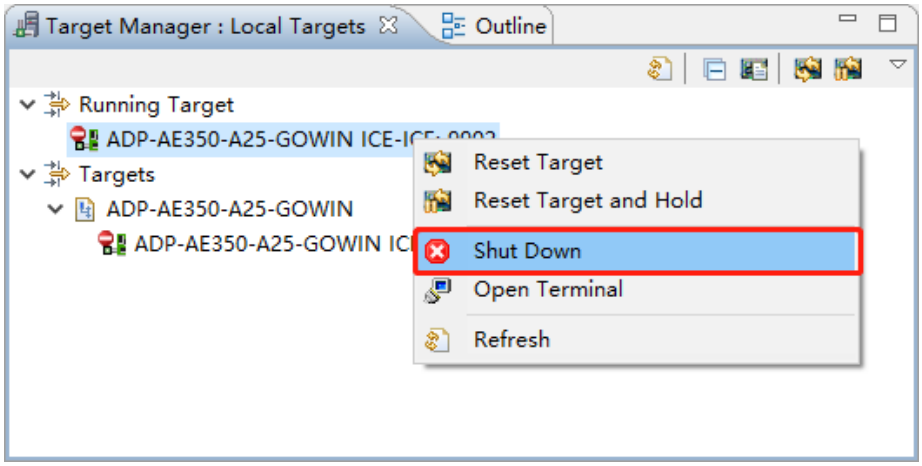
图 3-7 复位目标板



3.4 关闭目标

目标管理器视图中，右单击选定的目标，下拉菜单中选择“Shut Down”，关闭目标，如图 3-8 所示。

图 3-8 关闭目标

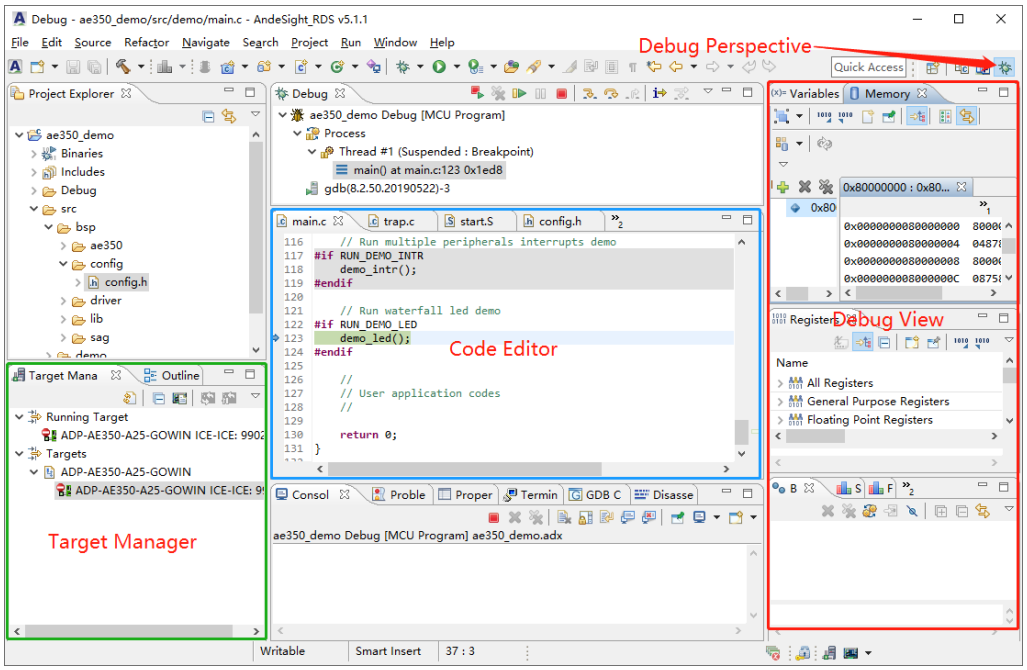

Target Manager : Local Targets
Outline
Running Target
ADP-AE350-A25-GOWIN ICE-ICE-0002
Targets
ADP-AE350-A25-GOWIN
ADP-AE350-A25-GOWIN IC
Reset Target
Reset Target and Hold
Shut Down
Open Terminal
Refresh

4 调试软件工程

4.1 调试透视图

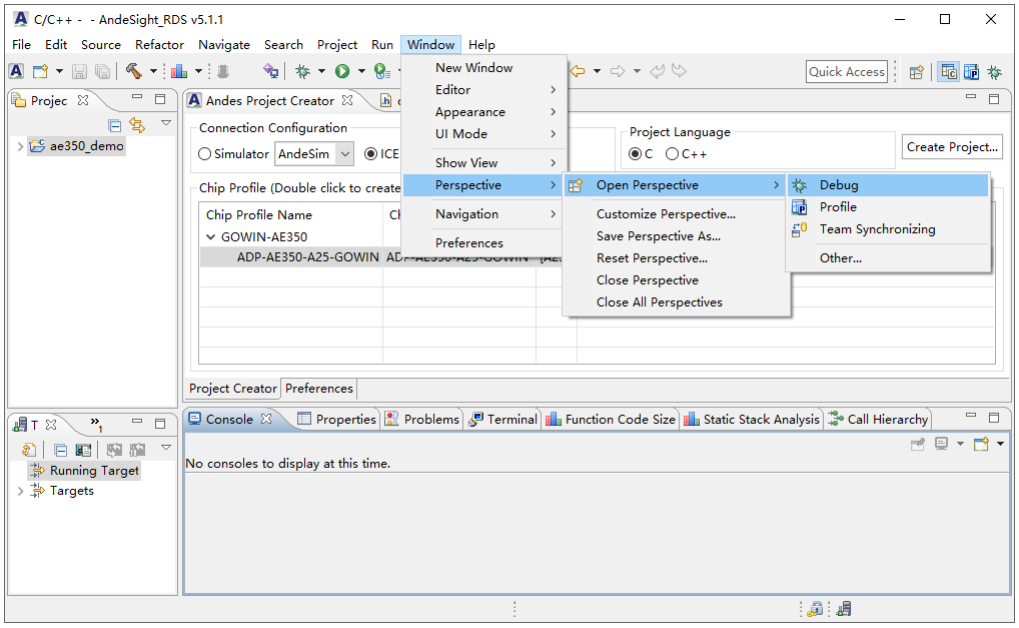
调试透视图包括代码编辑器、目标管理器视图和多个调试视图，用于控制应用程序的执行过程，诊断每一步的运行是否正确，如图 4-1 所示。

图 4-1 调试透视图



调试会话（Debug Session）启动后，RDS 软件自动进入调试透视图，或单击 RDS 软件透视图工具栏“”（Debug），手动进入调试透视图，或也可以选择 RDS 软件主菜单“Window > Perspective > Open Perspective > Debug”，如图 4-2 所示。

图 4-2 选择 Window > Perspective > Open Perspective > Debug



4.2 调试视图

调试会话启动后，RDS 软件会自动调用相应的调试视图。为调用指定的调试视图，选择 RDS 软件主菜单“Window > Show View> Other …”，如图 4-3 所示。“Show View”对话框中，单击“Debug”下拉箭头，在展开的列表中选择想要显示的调试视图，如图 4-4 所示。

图 4-3 选择 Window > Show View > Other...

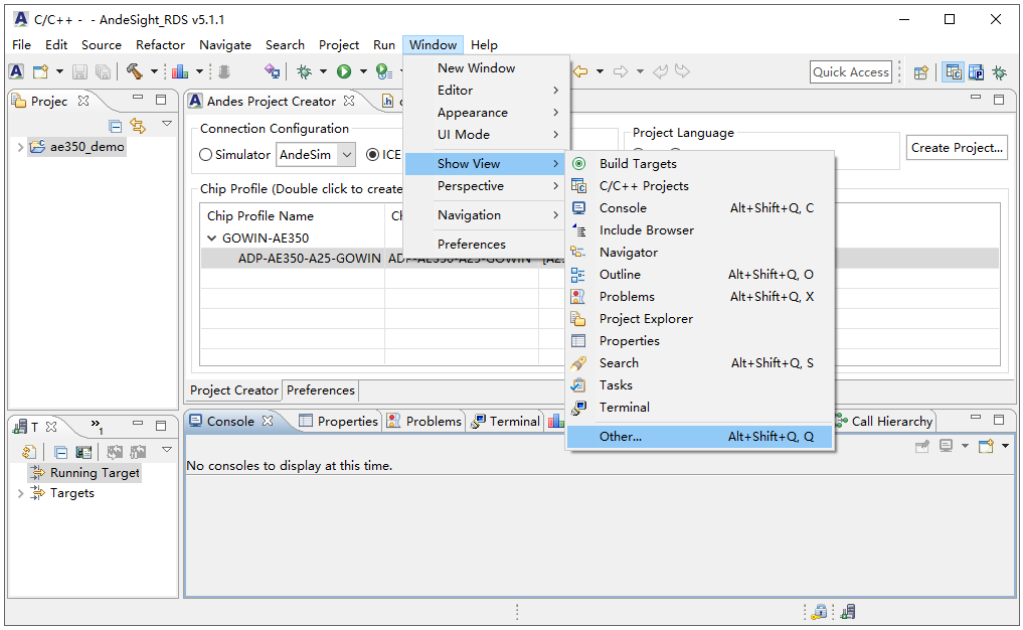
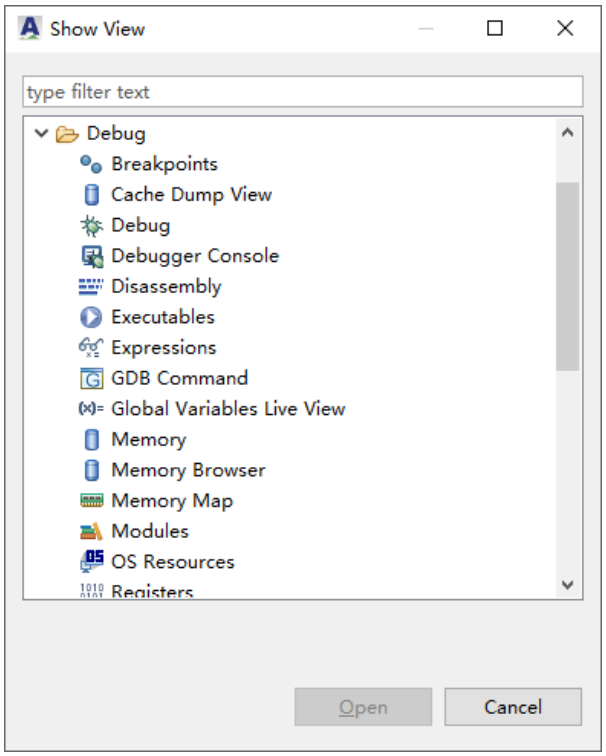


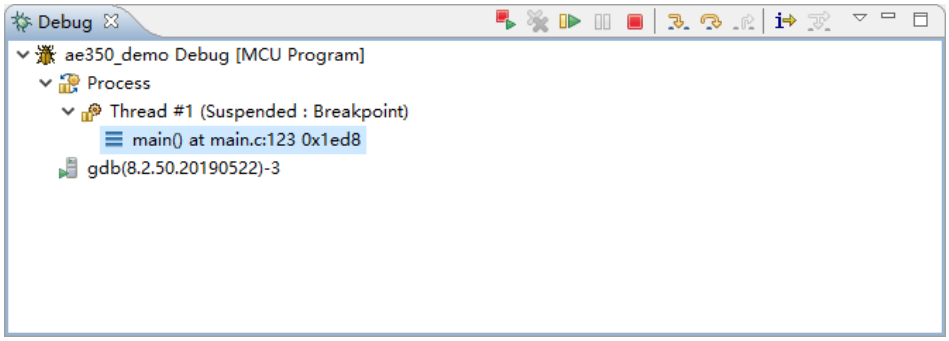
图 4-4 选择调试视图



4.2.1 调试视图

调试视图用于控制应用程序的执行过程，显示目标系统在程序暂停或执行时的堆栈情况，如图 4-5 所示。

图 4-5 调试视图



调试视图工具栏的功能包括：

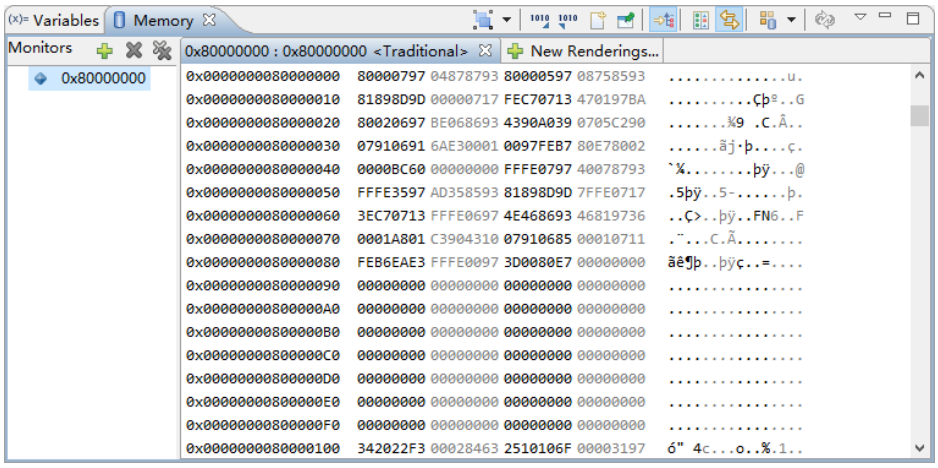
- “ ”（Terminate and Relaunch）：停止本次调试线程，重启新的调试线程。
- “ ”（Resume）：继续执行暂停的调试线程。
- “ ”（Suspend）：临时暂停调试线程。

- “ ”（Terminate）：停止调试线程。
- “ ”（Step Into）：单步执行调试线程的当前语句，进入子函数调用。
- “ ”（Step Over）：单步执行调试线程的当前语句，跳过子函数调用。
- “ ”（Step Return）：单步执行调试线程的当前语句，直到返回其上一级调用者。
- “ ”（Instruction Stepping Mode）：进入反汇编视图（Disassembly View），单步执行汇编指令。

4.2.2 内存视图

内存视图（Memory View）以内存监控列表的形式检查和修改程序的内存，如图 4-6 所示。每个内存监控都关联一段指定基地址或表达式的内存，每个内存监控的数据都可以以多种预定义的格式显示。

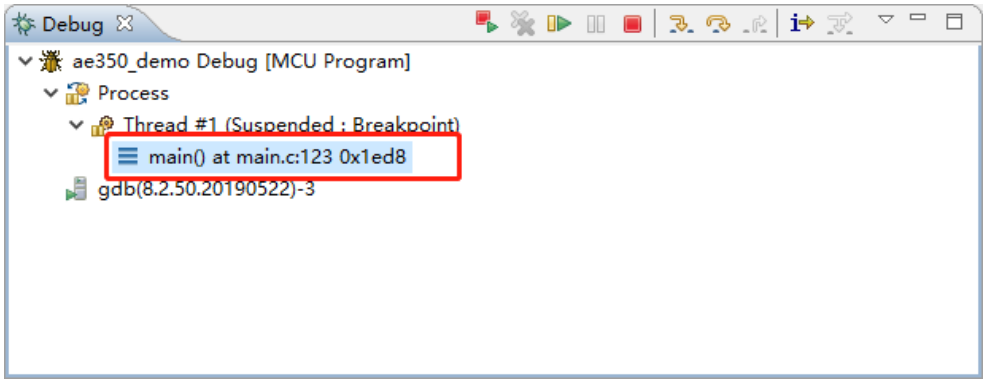
图 4-6 内存视图



建立内存监控

1. 程序执行暂停后，在调试视图中选择一个想要监控的线程或堆栈结构，如图 4-7 所示。

图 4-7 选择一个线程或堆栈



2. 内存视图中，单击监控面板的“ ”（Add Memory Monitor），如图 4-8 所示。“Monitor Memory”对话框中，输入一个想要查看的内存地址或表达式，单击“OK”，如图 4-9 所示。

图 4-8 Add Memory Monitor

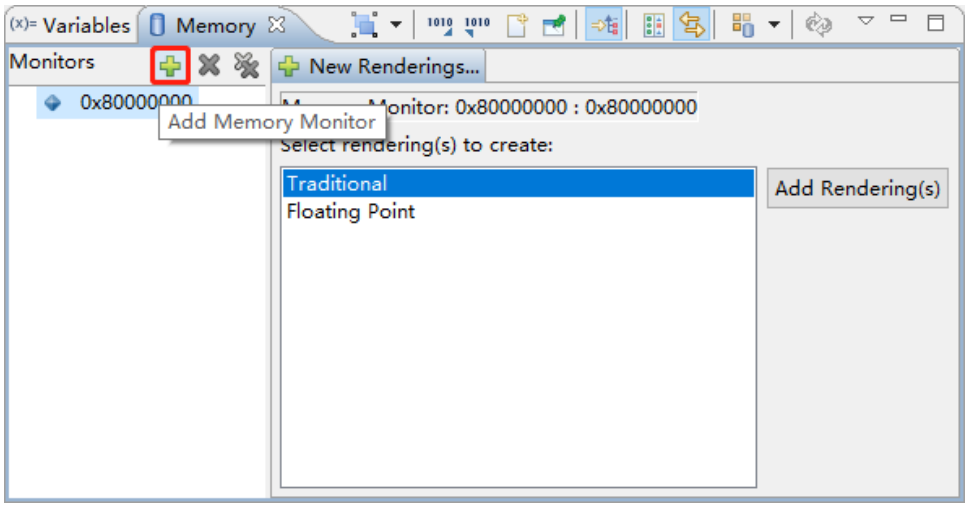
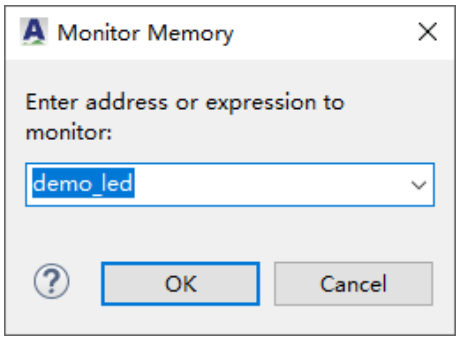
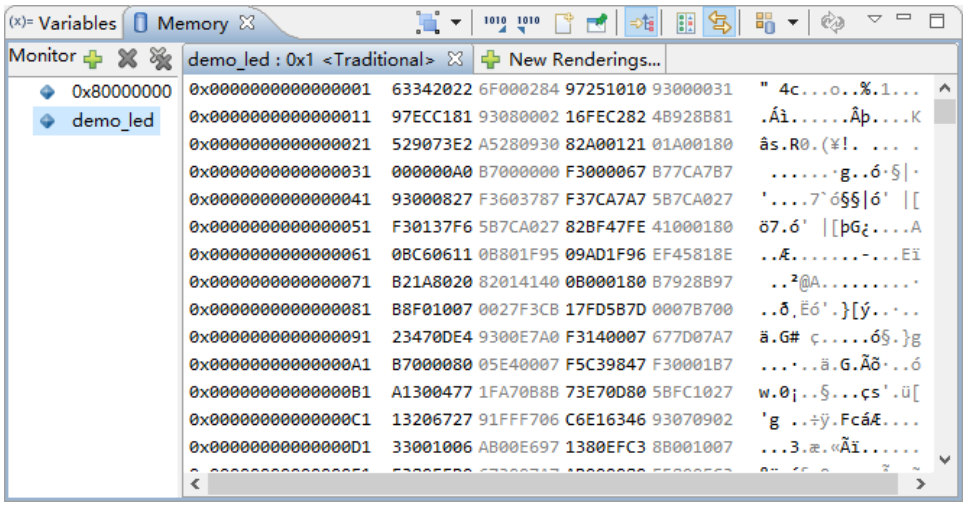


图 4-9 添加内存监控



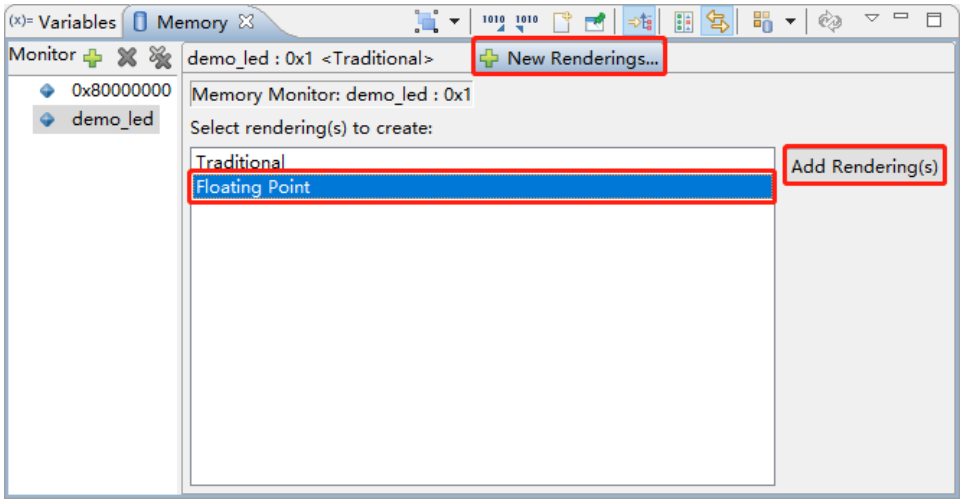
3. 内存视图中，默认显示一个传统的内存渲染，如图 4-10 所示。

图 4-10 传统的内存渲染



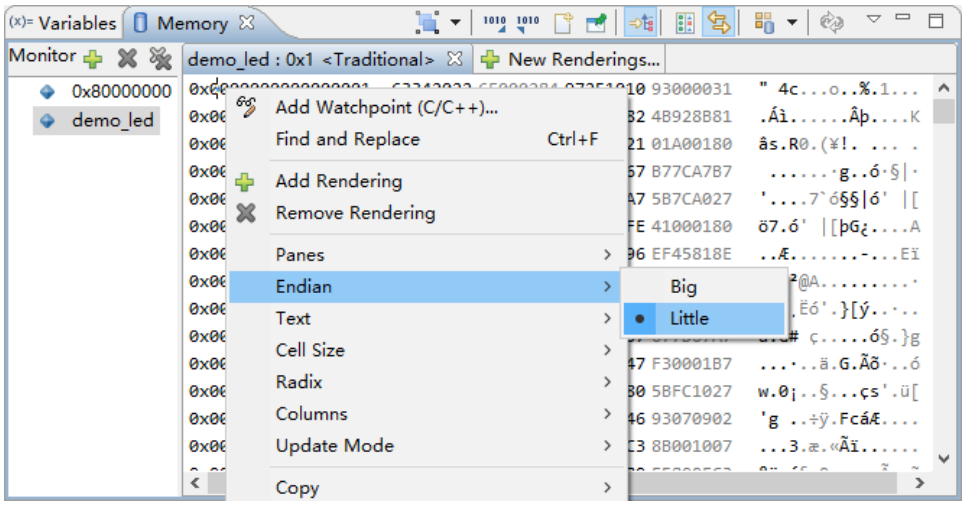
单击内存视图右边面板的“New Renderings…”，选择“Floating Point”，单击“Add Rendering(s)”，创建一个浮点型内存渲染，如图 4-11 所示。

图 4-11 创建浮点型内存渲染



4. 右单击内存渲染，下拉菜单中选择“Endian”，可以更改数据显示的字节顺序或可寻址大小，如图 4-12 所示。

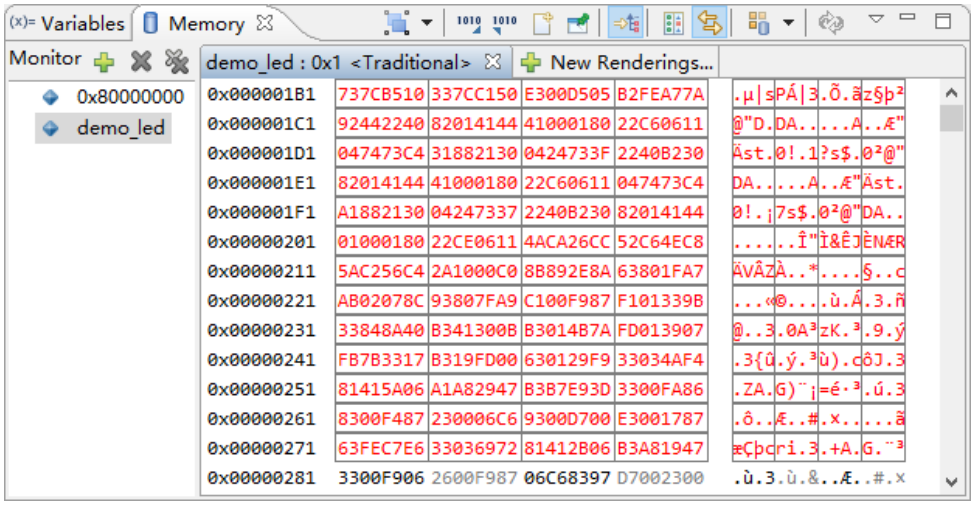
图 4-12 字节顺序设置



可以修改每个单元格中的内存值，但是如果输入了不合适的值，程序就会死机。

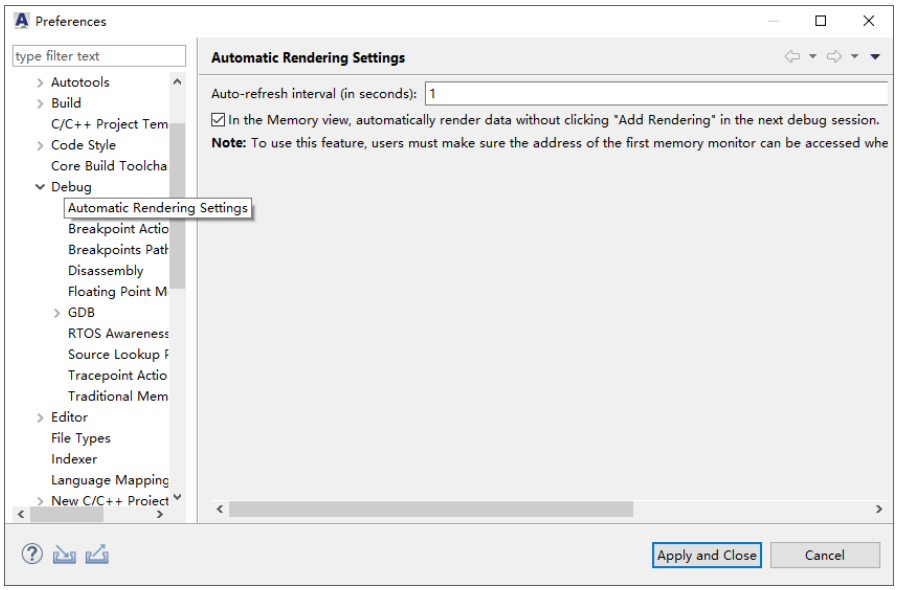
5. 程序执行过程中，存储内容的偏差会以红色标记，如图 4-13 所示。

图 4-13 存储内容



6. 如果要在重启调试会话后自动渲染相同内存监控的数据，则须在首选项设置中指定，选择 RDS 软件主菜单“Window > Preferences > C/C++ > Debug > Automatic Rendering Settings”，选择“In the Memory view, automatically render data without clicking “Add Rendering” in the next debug session.”，如图 4-14 所示。注意只有在程序停止后可以访问第一个内存监控的地址时，此特征才能起作用，否则会发生内存访问错误。

图 4-14 Automatic Rendering Settings

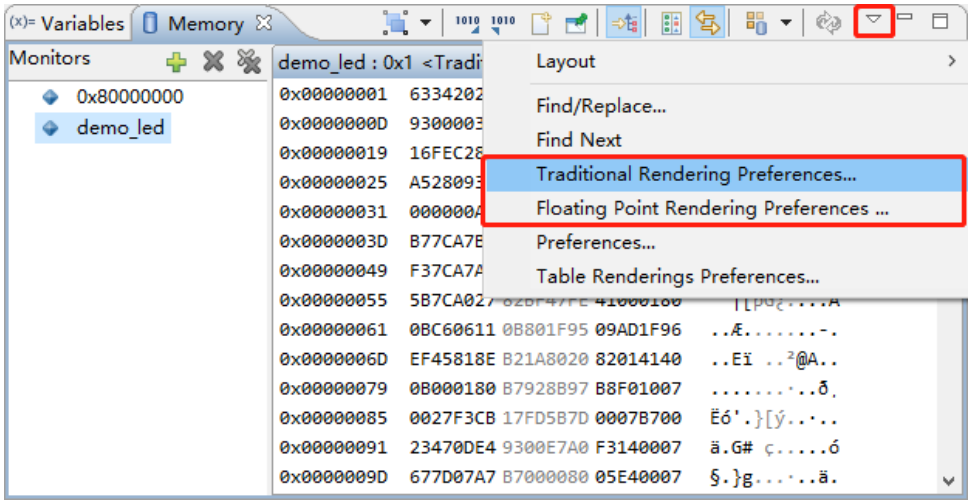


设置显示效果

内存渲染默认以黑色文本显示，参照以下步骤以不同颜色区分渲染的数据和地址：

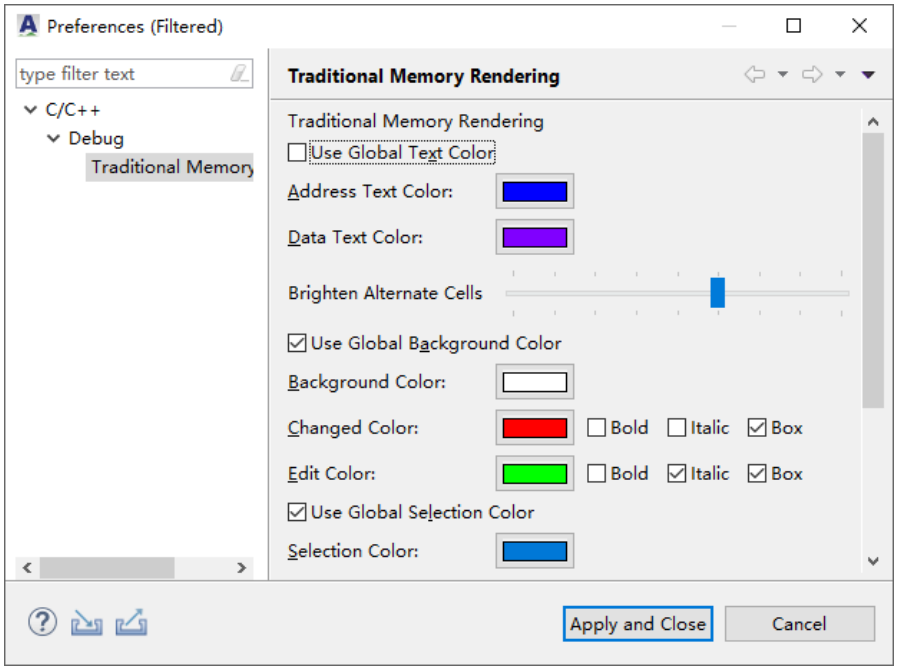
1. 单击内存视图工具栏“ ”（View Menu），下拉菜单中选择“Traditional Rendering Preferences…”或“Floating Point Rendering Preferences…”，如图 4-15 所示。

图 4-15 设置显示效果



2. “Preferences > C/C++ > Debug”对话框中，首先取消选中“Use Global Text Color”，然后为地址和数据各自定义文本颜色，如图 4-16 所示。

图 4-16 传统内存渲染设置



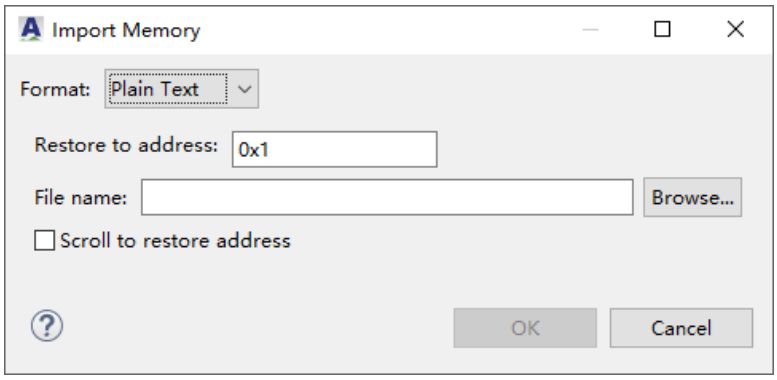
3. 单击“Apply and Close”，保存设置。

内存视图工具栏

内存视图工具栏的功能包括：

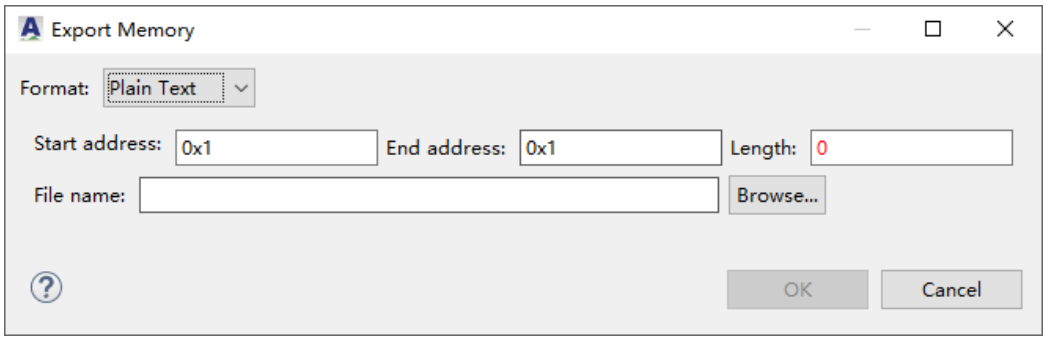
- “ ”（switch source）：指定内存内容的来源，CPU 或 BUS。CPU 模式，提供系统或高速缓存的内存内容。BUS 模式，仅提供系统的内存内容。
- “ ”（Import）：引入一组指定的内存值，如图 4-17 所示。

图 4-17 引入内存值



- “ ”（Export）：内存视图显示的内存值引出到一个文件，如图 4-18 所示。

图 4-18 引出内存值



RDS 软件支持 3 种引入/引出内存数据的格式：S-record、纯文本和原始二进制。纯文本和原始二进制格式，一般更常用。纯文本数据，包含 16 进制值，一个字节保存 2 个 16 进制字符。原始二进制数据，包含二进制值。

- “ ”（Auto Refresh）：用于定时刷新内存内容。对于具有自动刷新调试模块的 RiscV_AE350_SOC，可以启用自动刷新功能。

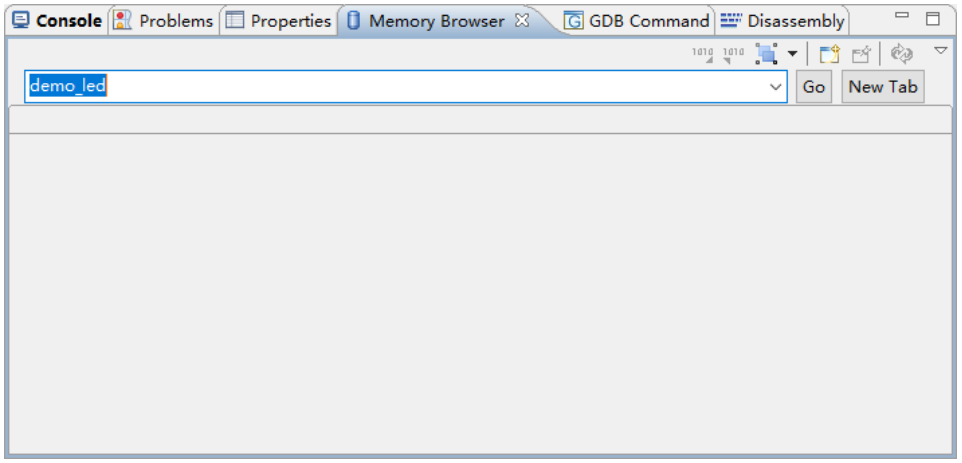
4.2.3 内存浏览器视图

内存浏览器视图（Memory Brower View）是内存视图的一种备选替代方式，用于检查和修改进程内存。

建立内存监控

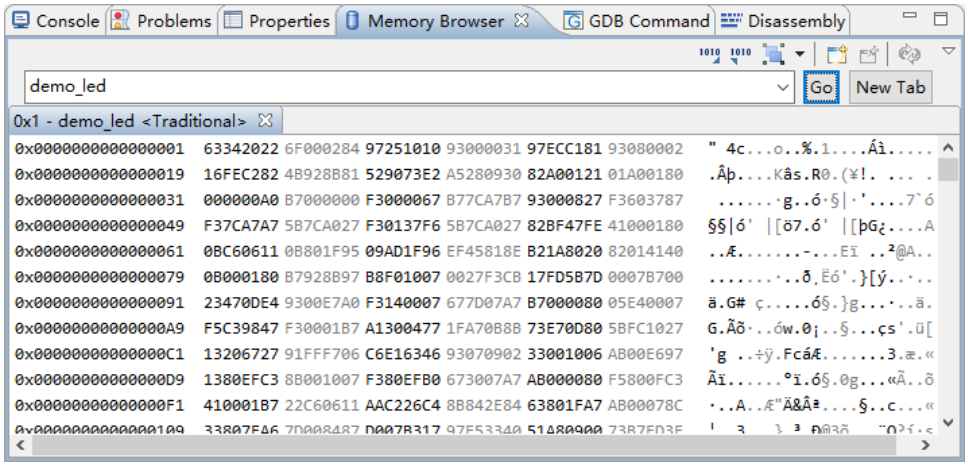
1. 程序执行暂停后，内存浏览器视图的文本框中输入想要查看的内存地址或表达式，单击“Go”，如图 4-19 所示。

图 4-19 内存浏览器视图



2. 内存浏览器视图中，默认显示一个传统的内存渲染，如图 4-20 所示。

图 4-20 传统的内存渲染



单击内存浏览器视图工具栏“ ”（View Menu），下拉菜单中选择“Default Rendering > Floating Point”，然后单击“New Tab”，如图 4-21 和图 4-22 所示。

图 4-21 浮点型内存渲染

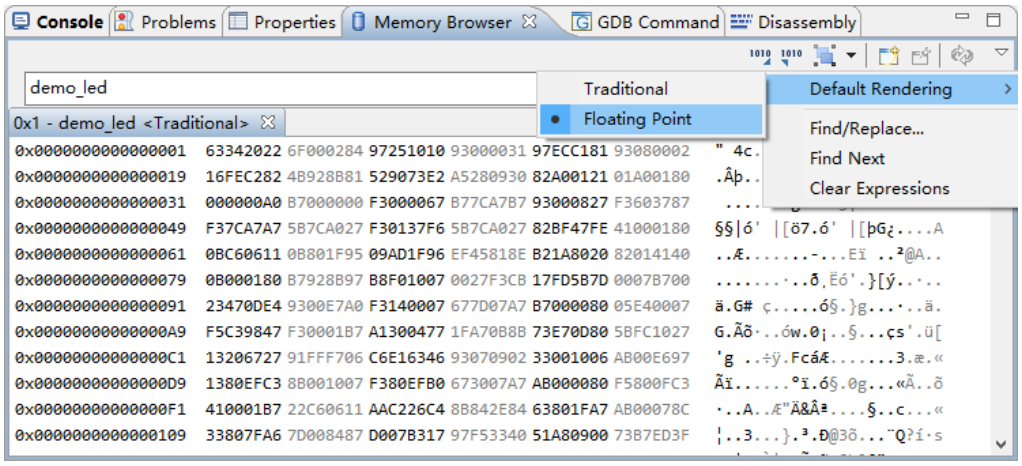
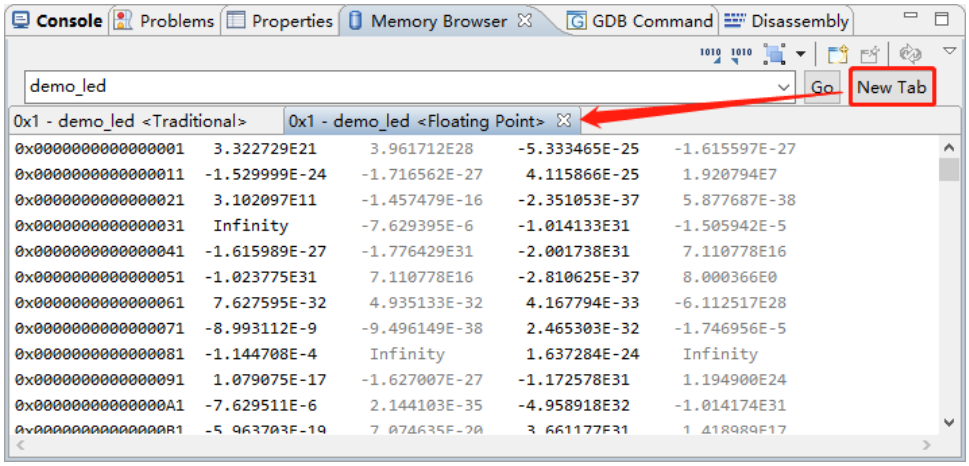


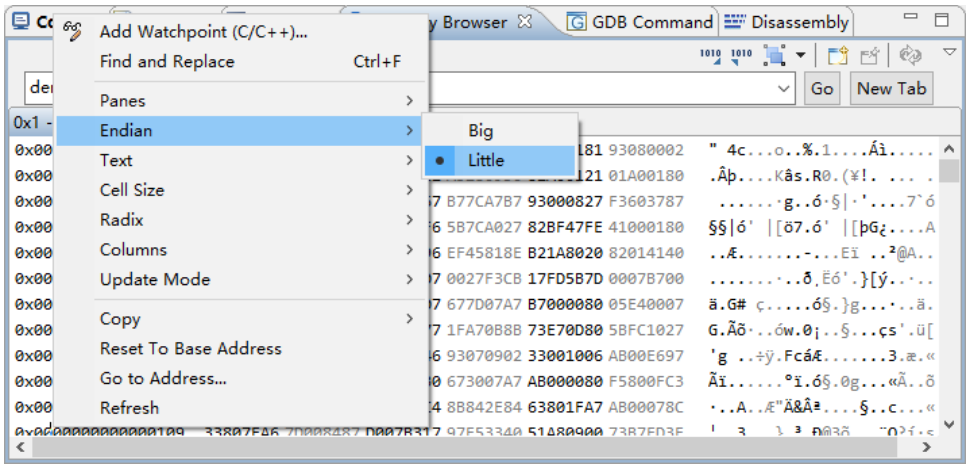
图 4-22 New Tab



3. 右单击内存渲染，下拉菜单中选择“Endian”，可以更改数据显示的字

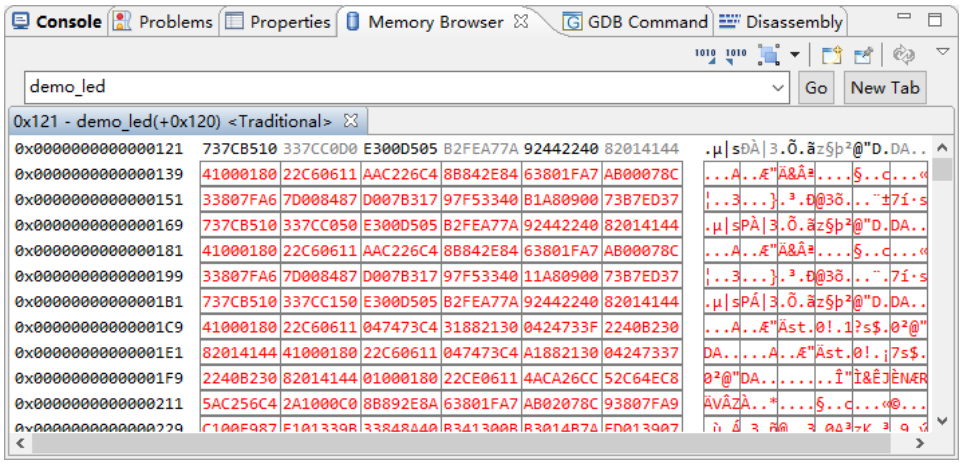
节顺序或可寻址大小，如图 4-23 所示。

图 4-23 字节顺序设置



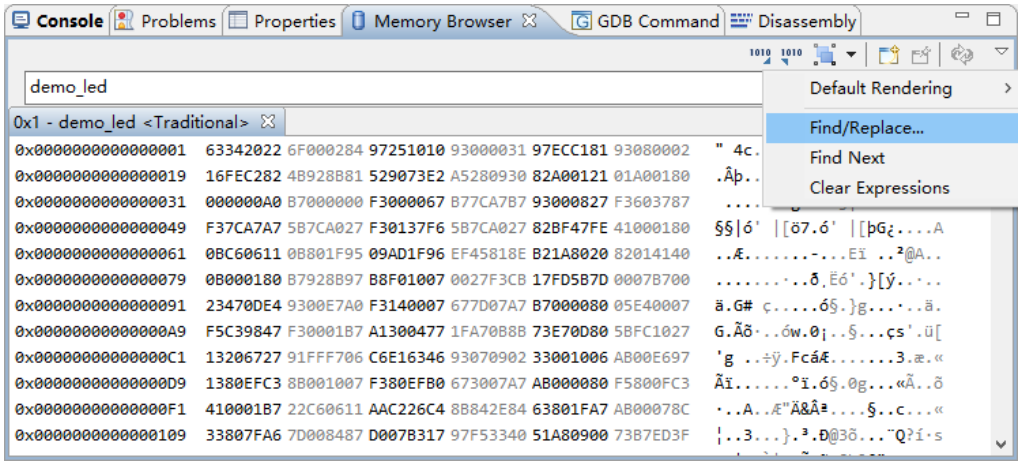
4. 程序执行过程中，存储内容的偏差会以红色标记，如图 4-24 所示。

图 4-24 存储内容



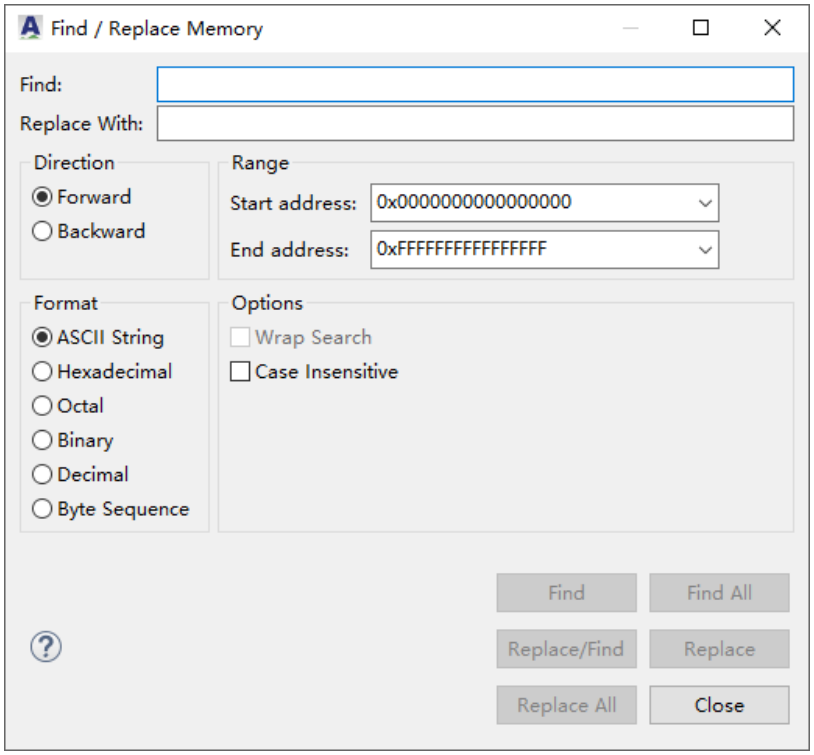
5. 单击内存浏览器视图工具栏“ ”（View Menu），下拉菜单中选择“Find/Replace…”，可以查找和替换选定的内存监控的指定数据，如图 4-25 所示。

图 4-25 Find/Replace...



“Find / Replace Memory”对话框中，输入要查找和替换的数据，单击“Find”或“Replace”，如图 4-26 所示。

图 4-26 查找/替换内存



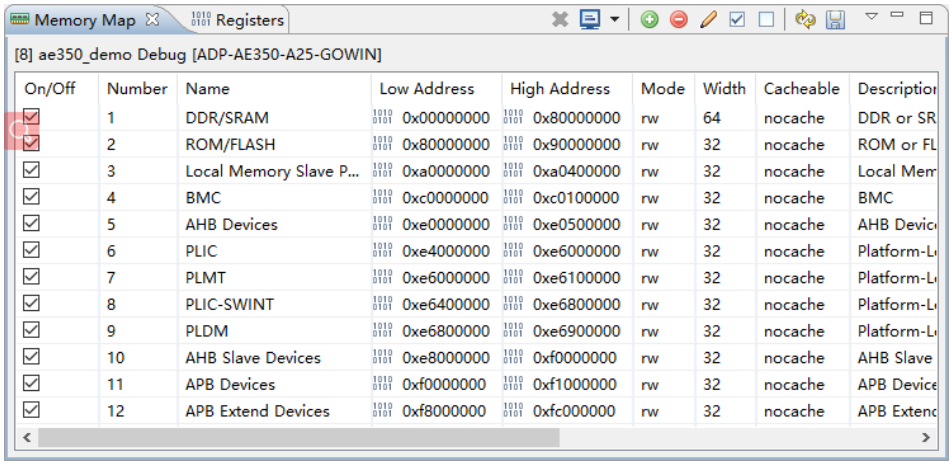
6. 对于具有调试模块的目标系统，支持自动刷新功能，单击内存浏览器视图工具栏“ ”（Auto Refresh），定期刷新内存内容。

4.2.4 内存映射视图

内存映射视图（Memory Map View）是一个接口，用于检查或添加/修改/删除内存区域、内存属性（只读、只写、读/写）和内存映射设备的描

述，如图 4-27 所示。

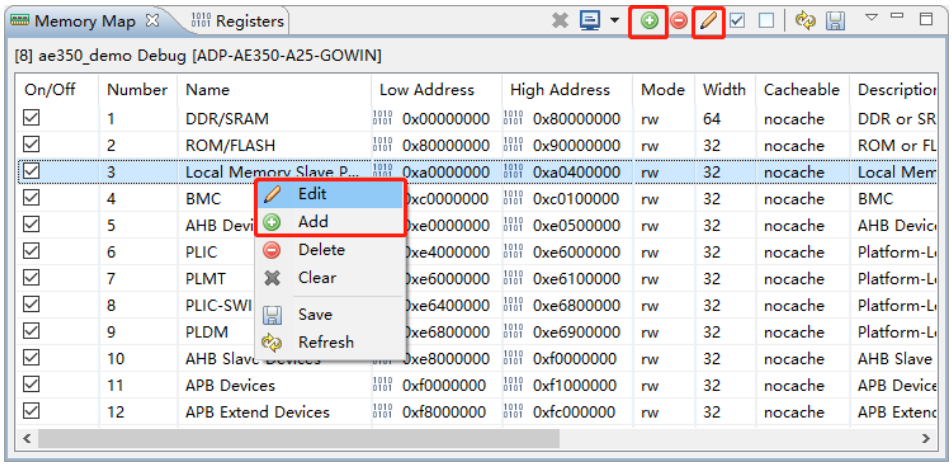
图 4-27 内存映射视图



参照以下步骤添加/修改/删除用户自定义的内存区域：

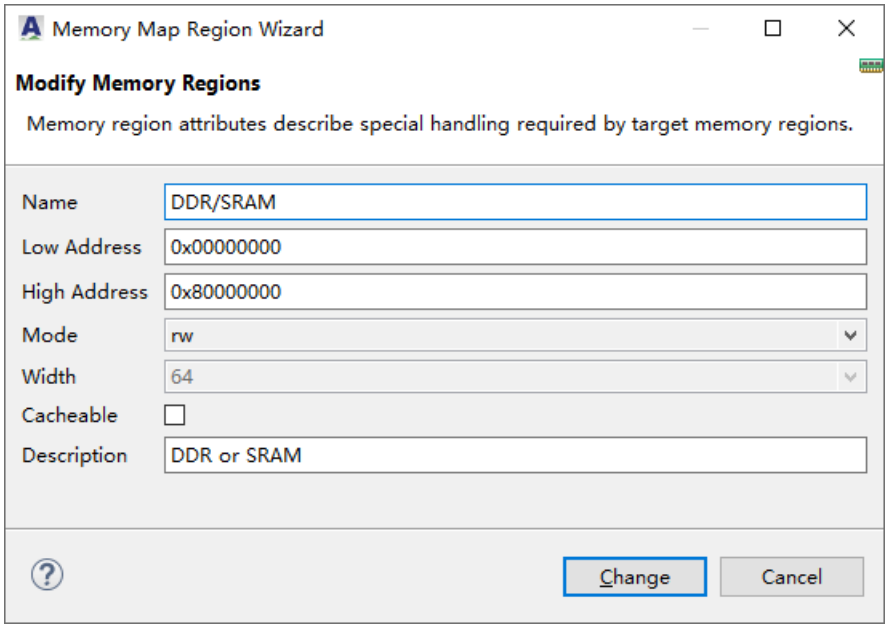
1. 添加内存区域，单击内存映射视图工具栏“ ”（Add Memory Regions），或右单击内存映射视图的任意位置，下拉菜单中选择“Add”。修改内存区域，内存映射视图中选定想要修改的行，单击内存映射视图工具栏“ ”（Modify Memory Regions），或右单击内存映射视图的任意位置，下拉菜单中选择“Edit”。添加/修改内存区域，如图 4-28 所示。

图 4-28 添加/修改内存区域



2. 双击一个内存映射区域，“Memory Map Region Wizard”对话框中，配置相关的选项，单击“Add”或“Change”，如图 4-29 所示。

图 4-29 Memory Map Region Wizard



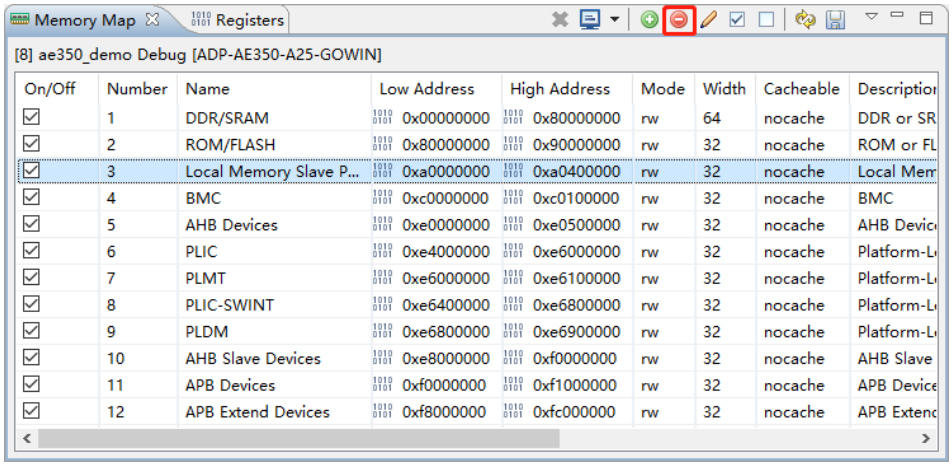
内存映射区域的选项，如表 4-1 所示。

表 4-1 内存映射区域选项

选项	描述
Name	内存区域的名称
Low Address	指定内存区域的起始地址
High Address	指定内存区域的结束地址
Description	内存区域的详细描述
Mode	指定 GDB 访问内存区域的模式，包括： ro：只读 wo：只写 rw：读/写（默认模式）
Cacheable	指定 GDB 是否缓存目标内存

3. 创建/修改的内存区域会立即显示在内存映射视图中，通过选中/取消选中“On/Off”列的复选框，可以开启/关闭此内存区域。
4. 删除内存区域，选中一行内存区域，单击内存映射视图工具栏“ ”（Delete Memory Regions），如图 4-30 所示。

图 4-30 删除内存区域

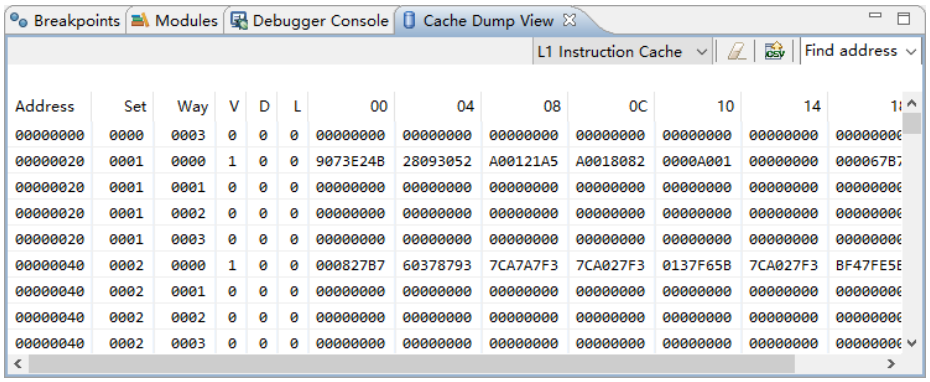


On/Off	Number	Name	Low Address	High Address	Mode	Width	Cacheable	Description
☑	1	DDR/SRAM	0x00000000	0x80000000	rw	64	nocache	DDR or SR
☑	2	ROM/FLASH	0x80000000	0x90000000	rw	32	nocache	ROM or FL
☑	3	Local Memory Slave P...	0xa0000000	0xa0400000	rw	32	nocache	Local Mem
☑	4	BMC	0xc0000000	0xc0100000	rw	32	nocache	BMC
☑	5	AHB Devices	0xe0000000	0xe0500000	rw	32	nocache	AHB Devic
☑	6	PLIC	0xe4000000	0xe6000000	rw	32	nocache	Platform-L
☑	7	PLMT	0xe6000000	0xe6100000	rw	32	nocache	Platform-L
☑	8	PLIC-SWINT	0xe6400000	0xe6800000	rw	32	nocache	Platform-L
☑	9	PLDM	0xe6800000	0xe6900000	rw	32	nocache	Platform-L
☑	10	AHB Slave Devices	0xe8000000	0xf0000000	rw	32	nocache	AHB Slave
☑	11	APB Devices	0xf0000000	0xf1000000	rw	32	nocache	APB Device
☑	12	APB Extend Devices	0xf8000000	0xfc000000	rw	32	nocache	APB Extenc

4.2.5 缓存转储视图

缓存转储视图（Cache Dump View），用于在程序执行暂停后，监控目标系统的 CPU 缓存。对于每一个缓存行，此视图显示其当前信息，包括“Address”、“Set”、“Way”、“V”（Valid state）、“D”（Dirty state）、“L”（Lock state）和数据，如图 4-31 所示。

图 4-31 缓存转储视图



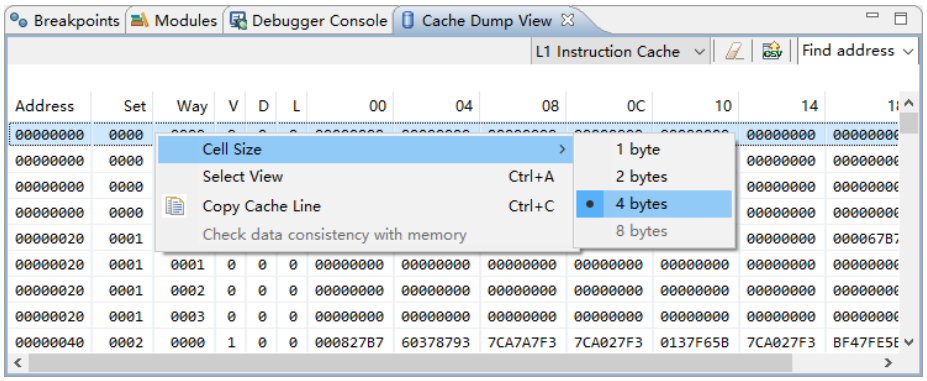
Address	Set	Way	V	D	L	00	04	08	0C	10	14	1
00000000	0000	0003	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	0000000
00000020	0001	0000	1	0	0	9073E24B	28093052	A00121A5	A0018082	0000A001	00000000	000067B7
00000020	0001	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	0000000
00000020	0001	0002	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	0000000
00000020	0001	0003	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	0000000
00000040	0002	0000	1	0	0	000827B7	60378793	7CA7A7F3	7CA027F3	0137F65B	7CA027F3	BF47FE5E
00000040	0002	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	0000000
00000040	0002	0002	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	0000000
00000040	0002	0003	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	0000000

每当程序执行或缓存操作后，如果与前一个缓存内容有差异，则会用不同颜色标记单元格的变化。

设置单元格大小

通过更改单元格大小（1/2/4/8 字节），可以调整缓存内容的显示方式，右单击一行缓存信息，下拉菜单中选择“Cell Size”，如图 4-32 所示。

图 4-32 单元格大小设置



检查数据与内存一致性

通过缓存转储视图可以检查缓存与内存之间的数据一致性，右单击一个有效的缓存行，下拉菜单中选择“Check data consistency with memory”，如图 4-33 所示。此视图的顶部立即显示缓存行的数据是否与对应的内存中的数据一致，如图 4-34 所示。

图 4-33 检查数据与内存一致性

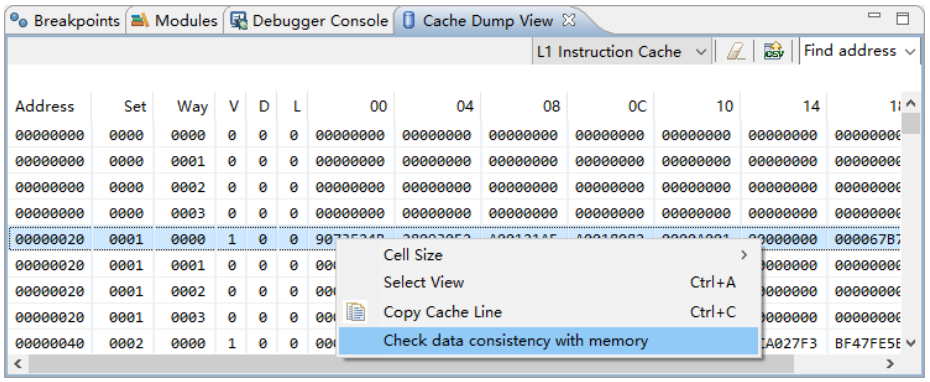
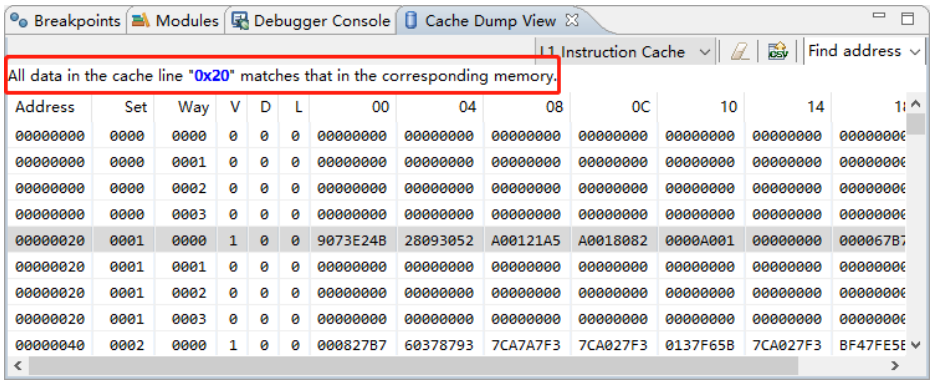


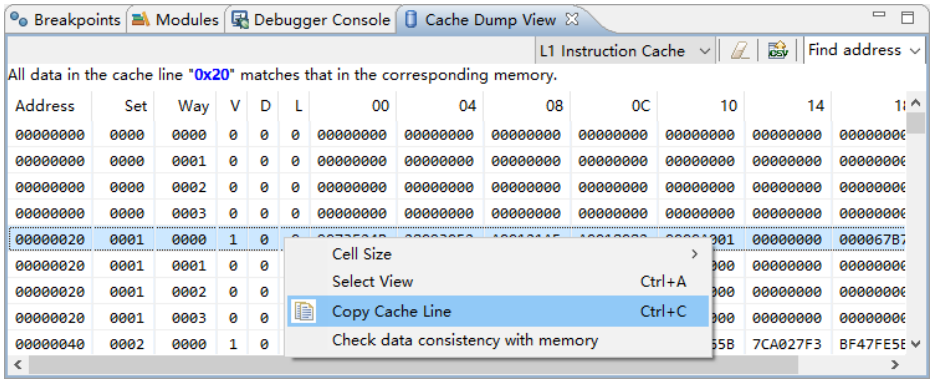
图 4-34 数据与内存一致



复制缓存行

如果需要保存此视图中显示的信息，则右单击一个缓存行，下拉菜单中选择“Copy Cache Line”，粘贴到一个文件中，如图 4-35 所示。

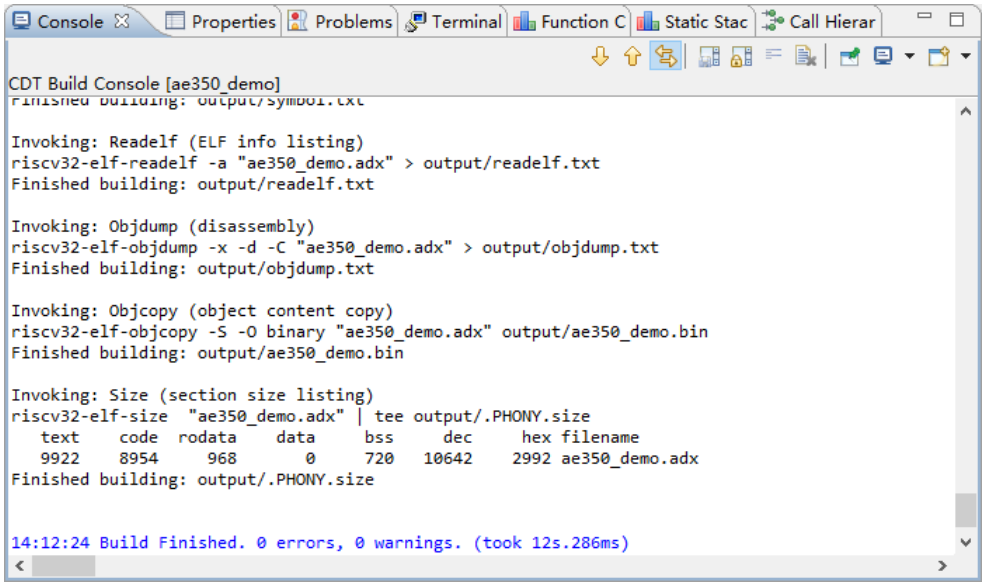
图 4-35 复制缓存行



4.2.6 控制台视图

控制台视图显示进程输出，以及为命令输入提供命令行接口。例如，一个 CDT 构建控制台，显示软件工程的构建结果，如图 4-36 所示。

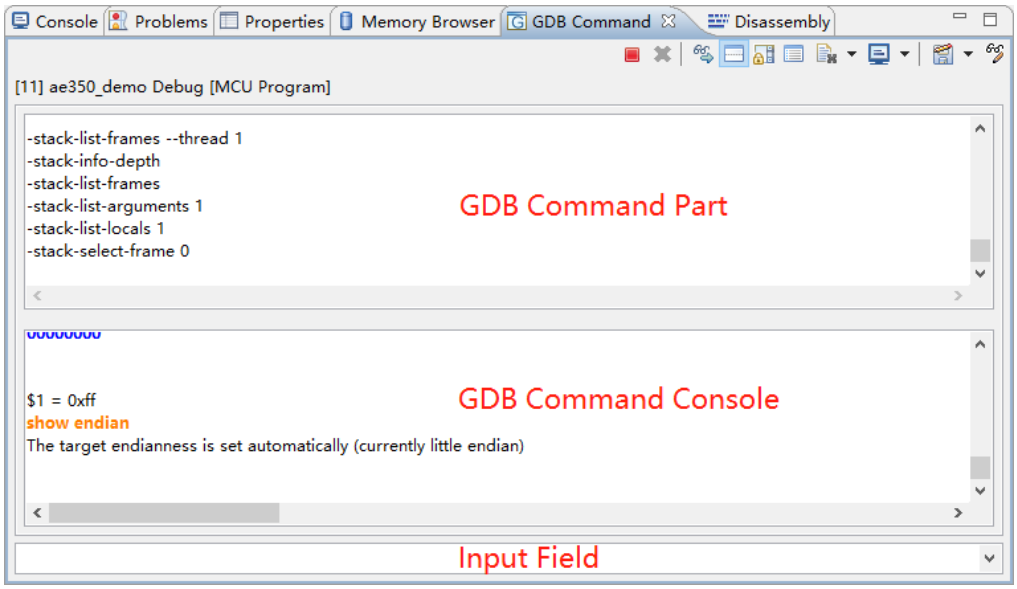
图 4-36 控制台视图



4.2.7 GDB 命令视图

GDB 命令视图（GDB Command View），用于 GDB 命令的接口，如图 4-37 所示。

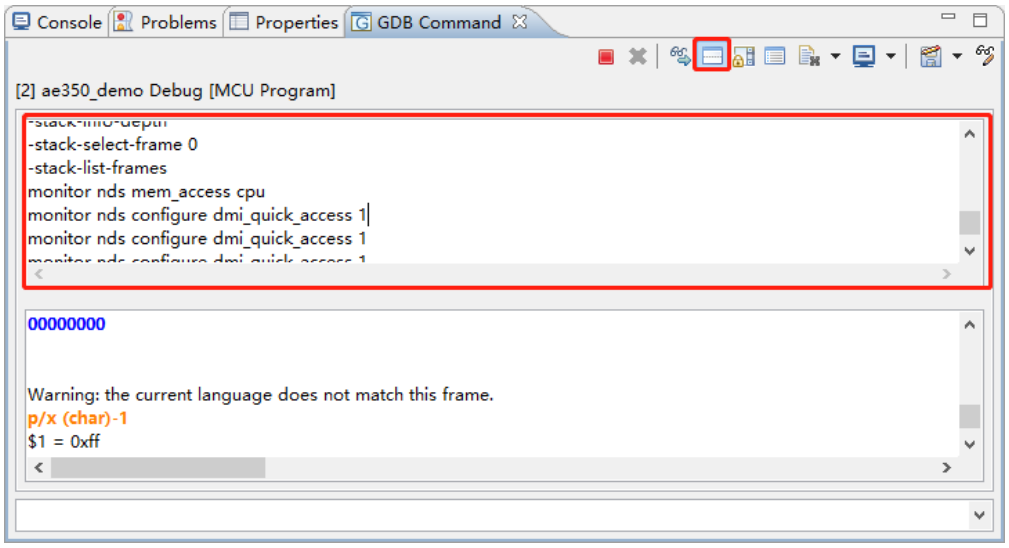
图 4-37 GDB 命令视图



GDB 命令视图包括 3 部分：GDB 命令部分、GDB 命令控制台和 GDB 命令输入。

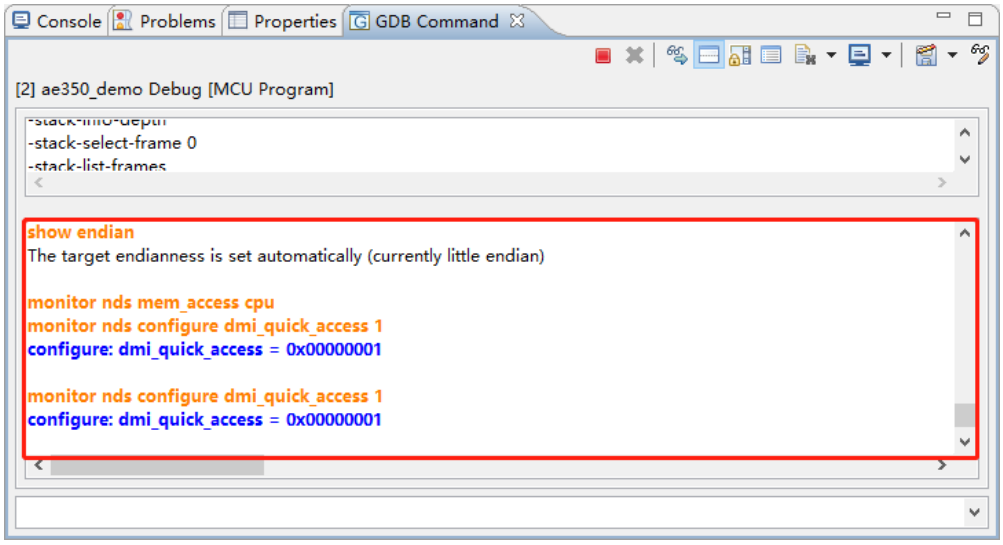
此视图的顶部部分是一个只读的日志视图，显示调试会话中可以发送的所有 GDB 命令，此部分默认隐藏，可以单击 GDB 命令视图工具栏 “ ”（Show Command Part）来调用，如图 4-38 所示。

图 4-38 GDB 命令部分



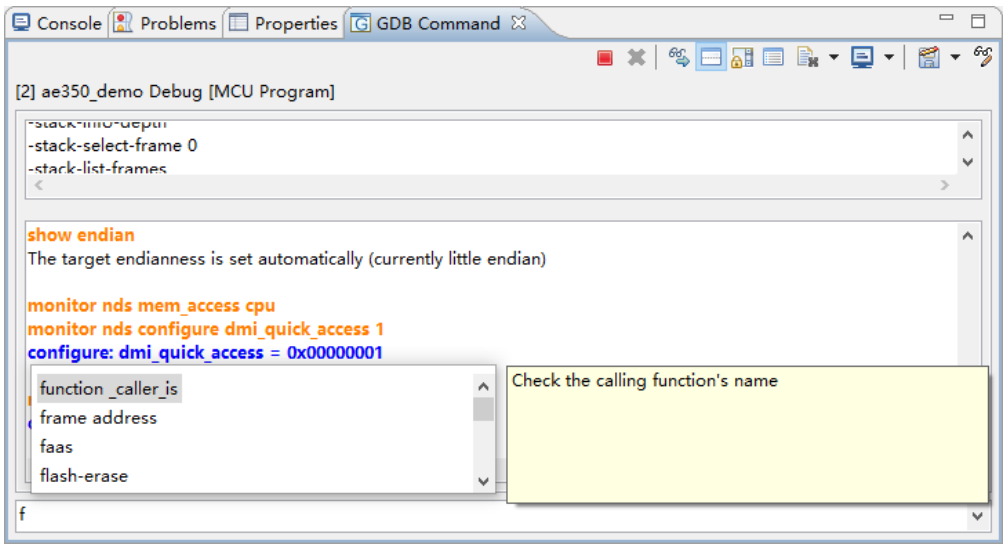
此视图的中间部分是 GDB 命令控制台，显示 GDB 命令和结果。控制台以不同的颜色显示不同的类型信息：橙色表示 GDB 命令，蓝色表示程序输出，黑色表示 GDB 命令结果，如图 4-39 所示。

图 4-39 GDB 命令控制台



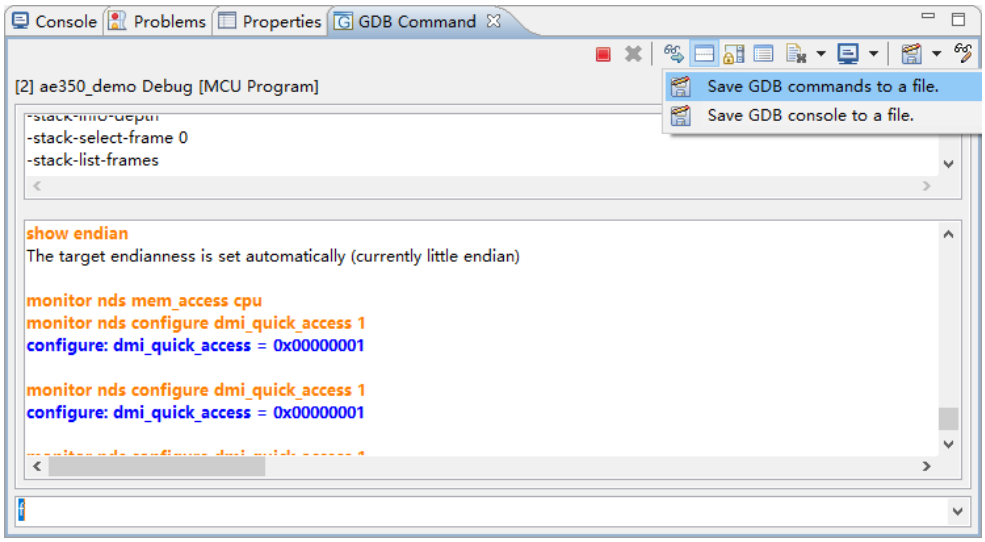
此视图的底部部分是命令输入接口，用来发送 GDB 命令。此部分提供对输入命令的建议和描述，最多可以保存 20 条 GDB 命令记录，如图 4-40 所示。

图 4-40 GDB 命令输入



GDB 命令部分或 GDB 命令控制台的信息可以引出到一个文件，单击 GDB 命令视图工具栏“ ”（Save GDB Context）下拉箭头，选择“Save GDB commands to a file.”或“Save GDB console to a file.”，如图 4-41 所示。

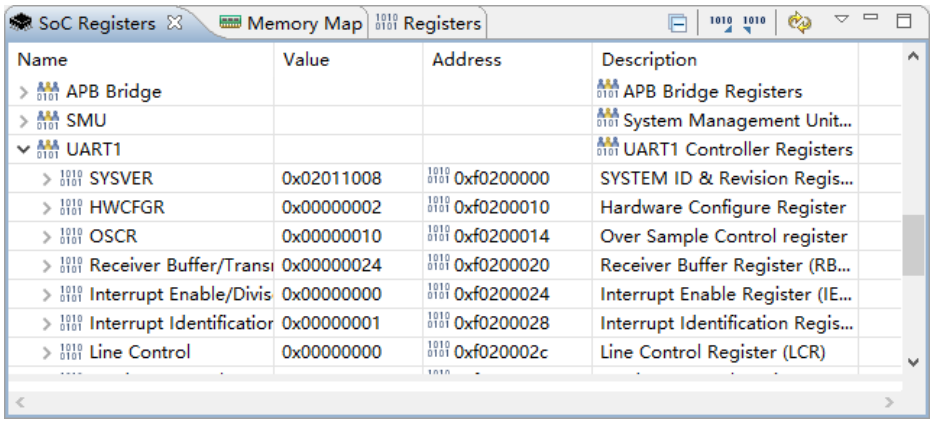
图 4-41 引出 GDB 命令



4.2.8 SoC 寄存器视图

SoC 寄存器视图（SoC Registers View）用于读取或写入位域级别的 SoC 寄存器的值。此视图作为调试透视图的一部分，在调试会话中自动触发，如图 4-42 所示。

图 4-42 SoC 寄存器视图



引入/引出寄存器的值

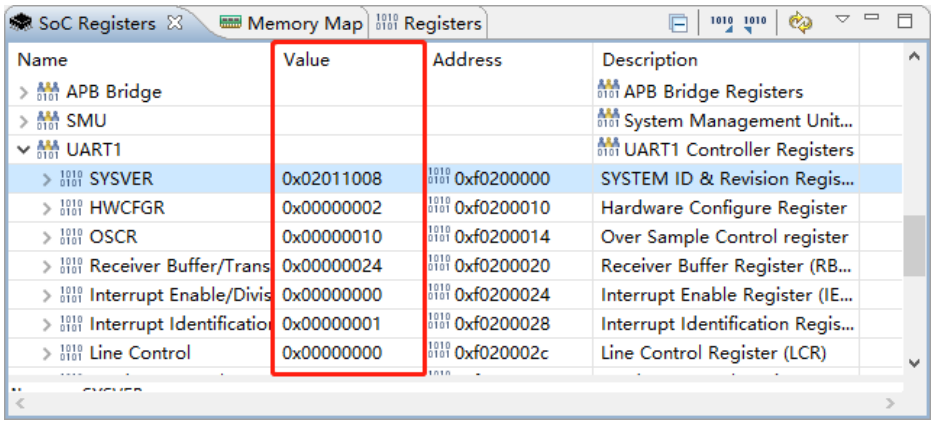
可以在程序运行时引出指定的 SoC 寄存器的值，选择想要引出的 SoC 组件或寄存器（可以通过按住“Ctrl”进行多次选择），单击 SoC 寄存器视图工具栏“ ”（Export）。

单击 SoC 寄存器视图工具栏“ ”（Import），可以引入 SoC 寄存器数据。

修改寄存器的值

可以直接修改单元格中的寄存器或位域的值，如图 4-43 所示。

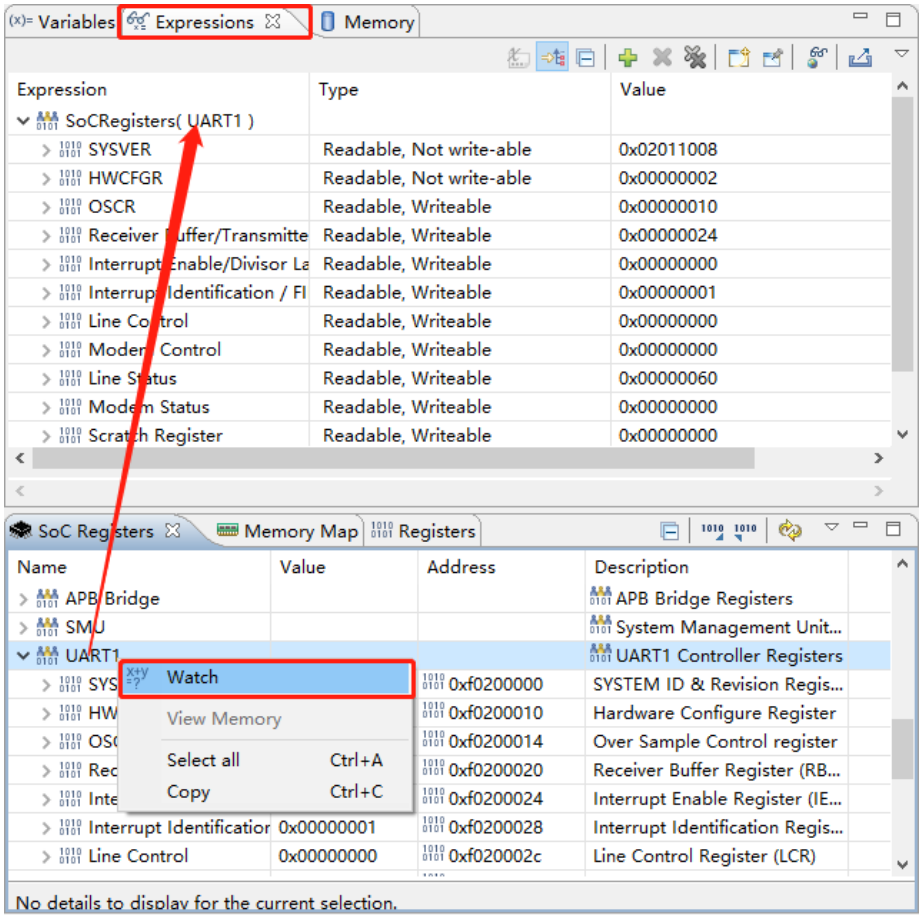
图 4-43 修改寄存器的值



查看 SoC 寄存器

可以查看单个 SoC 寄存器，也可以查看 SoC 寄存器组。SoC 寄存器视图中，右单击想要查看的寄存器组或单个寄存器，下拉菜单中选择“Watch”，表达式视图（Expressions View）显示选定的 SoC 寄存器的信息，如图 4-44 所示。

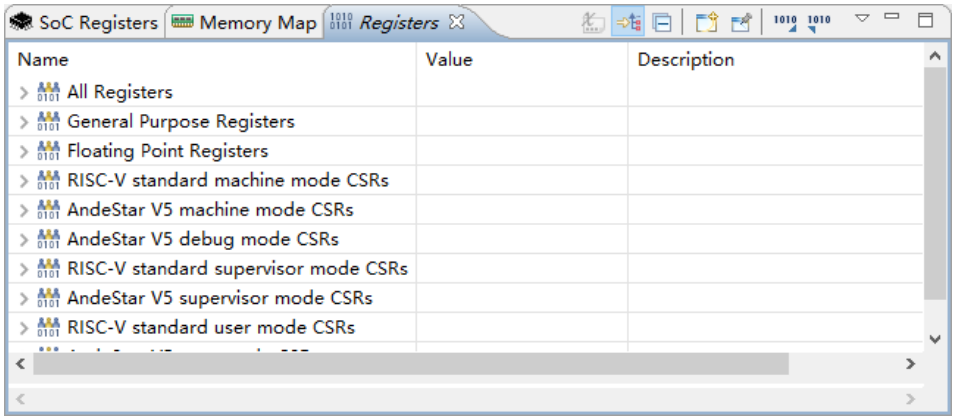
图 4-44 查看 SoC 寄存器



4.2.9 寄存器视图

寄存器视图（Registers View）显示寄存器位域的级别的 CPU 信息，可以直接在单元格中修改寄存器或位域的值，如图 4-45 所示。单击寄存器视图工具栏“”（Export）或“”（Import），可以引出或引入寄存器的值。

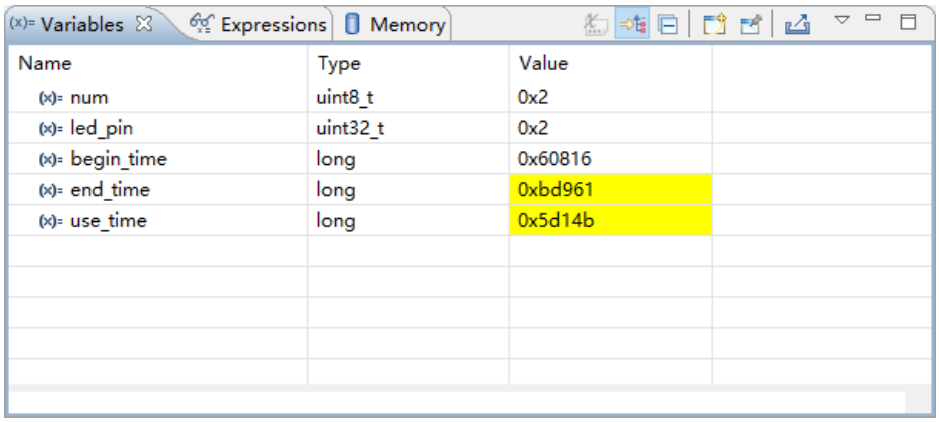
图 4-45 寄存器视图



4.2.10 变量视图

变量视图（Variables View），列出了与在调试视图中选定的堆栈结构相关联的变量的信息。程序执行暂停时，高亮突出显示变量值的变化，如图 4-46 所示。

图 4-46 变量视图

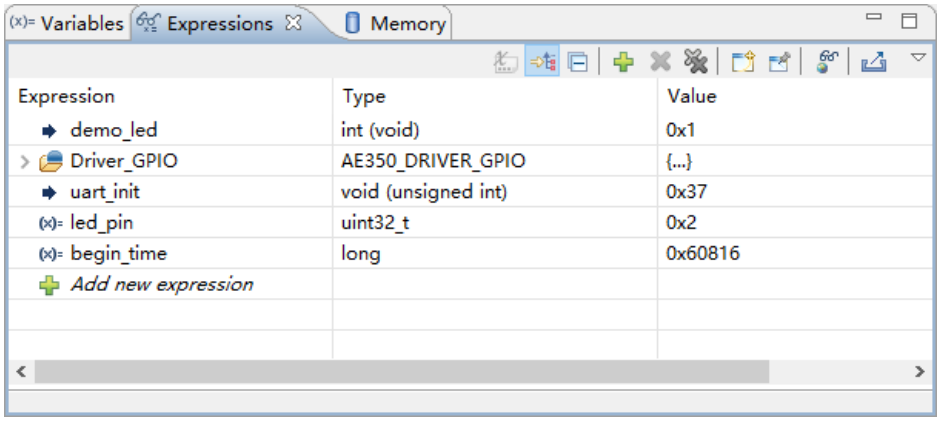


4.2.11 表达式视图

变量视图不会显示全局变量，而是由表达式视图或全局变量实时视图（Global Variables Live View）来显示，具体参考 *4.2.12 全局变量实时视图*。如果被监视的全局变量的值发生更新后，此视图中会以黄色高亮突出显示。对于表达式视图，只有在程序执行暂停后，才会显示全局变量更新的值。对于全局变量实时视图，只要程序运行在 RISC-V 内核的目标系统上，就会实时显示全局变量的值。

例如，一个应用程序在调试会话中的表达式视图，如图 4-47 所示。

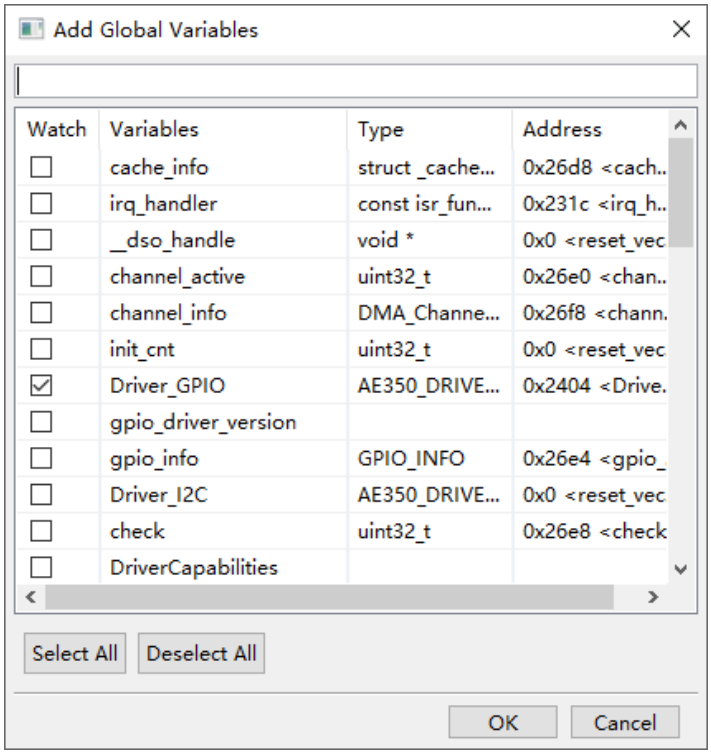
图 4-47 表达式视图



表达式视图的工具栏包括：

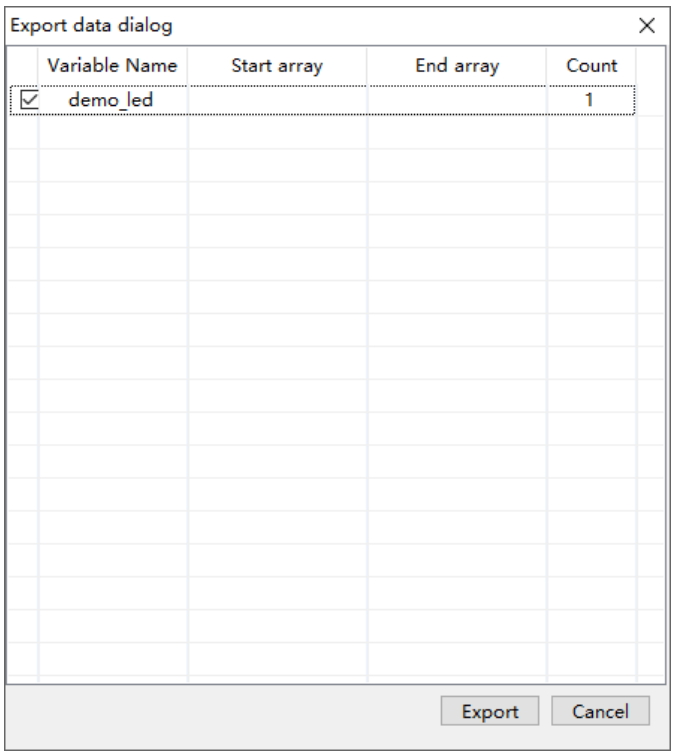
- “ ”（add global variables）：程序执行暂停后，单击“ ”，“Add Global Variables”对话框中，选择想要监视的全局变量，表达式视图显示选定的全局变量及其值，如图 4-48 所示。

图 4-48 Add Global Variables



- “ ”（Export）：可以引出想要的全局变量，选择一个全局变量，单击“ ”，如图 4-49 所示。

图 4-49 引出全局变量



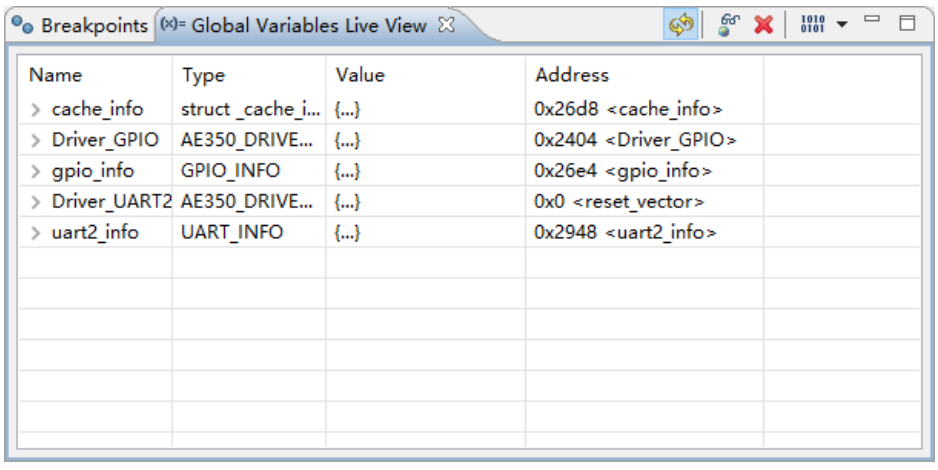
“Export data dialog”对话框中，指定用于引出的变量的开始和结束数组，然后单击“Export”。

4.2.12 全局变量实时视图

全局变量实时视图，只要程序运行在 RISC-V 内核的目标系统上，此视图就可以监视全局变量，实时检查全局变量的值，如图 4-50 所示。程序执行暂停后，可以在扩展列表中检查所有的变量成员。此视图中，程序执行期间更新的全局变量或结构成员的值，以黄色高亮突出显示。

如果使用模拟器目标系统，此视图仅在程序执行暂停后才会显示全局变量更新的值。

图 4-50 全局变量实时视图

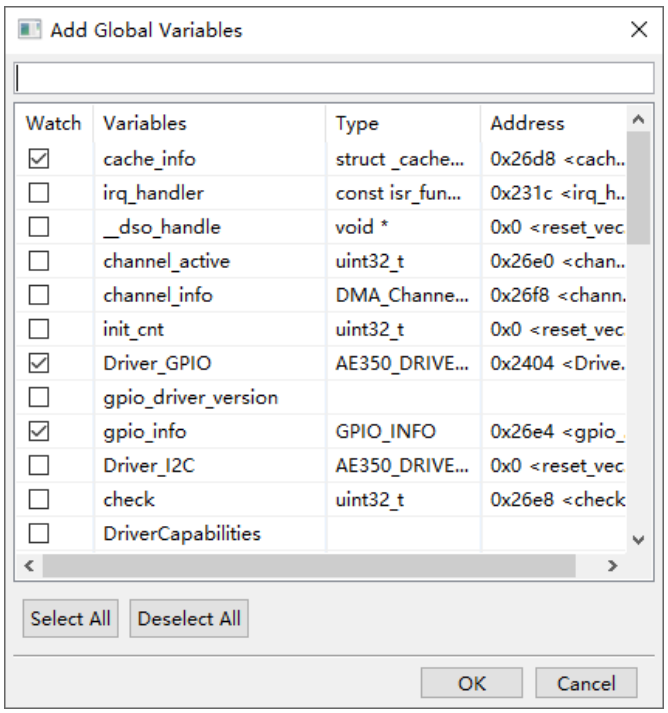


全局变量实时视图的工具栏包括：

- “  ”（Auto Refresh）：实时刷新全局变量。

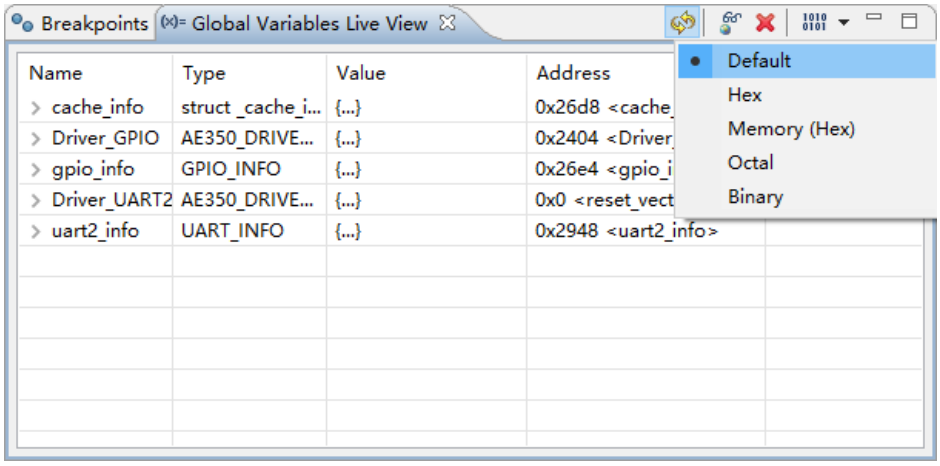
- “  ”（Add global variables）：程序执行暂停后，单击“  ”，“Add Global Variables”对话框中，选择想要监视的全局变量，全局变量实时视图显示选定的全局变量及其值，如图 4-51 所示。

图 4-51 Add Global Variables



- “”（Delete the selected variables）：删除被监视的全局变量。
- “  ”（Select the format for values）：单击下拉箭头，下拉菜单中为监视的全局变量选择一种显示格式，包括“Default”、“Hex”、“Memory (Hex)”、“Octal”和“Binary”，如图 4-52 所示。注意“Memory (hex)”指保存在内存中的 16 进制值，字节顺序与目标系统的字节顺序保持一致。

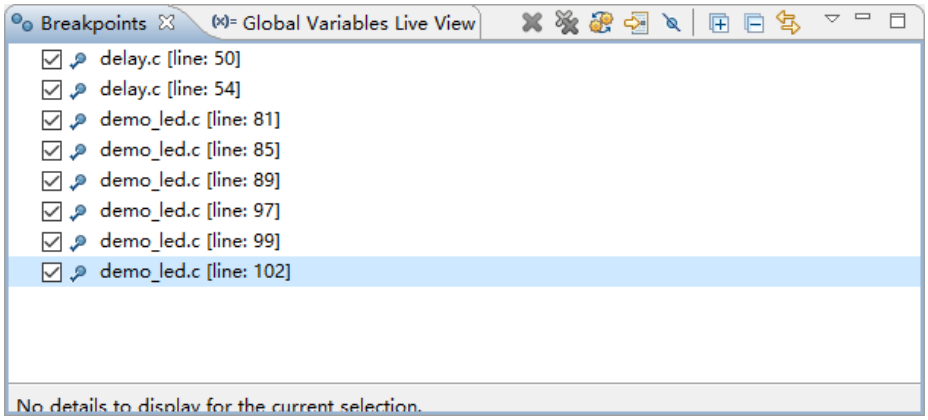
图 4-52 选择显示格式



4.2.13 断点视图

断点视图（Breakpoints View），列出了在软件工程中设置的所有断点，可以增加、取消、开启或关闭断点，如图 4-53 所示。

图 4-53 断点视图



4.2.14 反汇编视图

反汇编视图，显示了高级语言及其相关的汇编代码，用于在调试时查找指令问题，如图 4-54 所示。

图 4-54 反汇编视图

4.3 调试会话

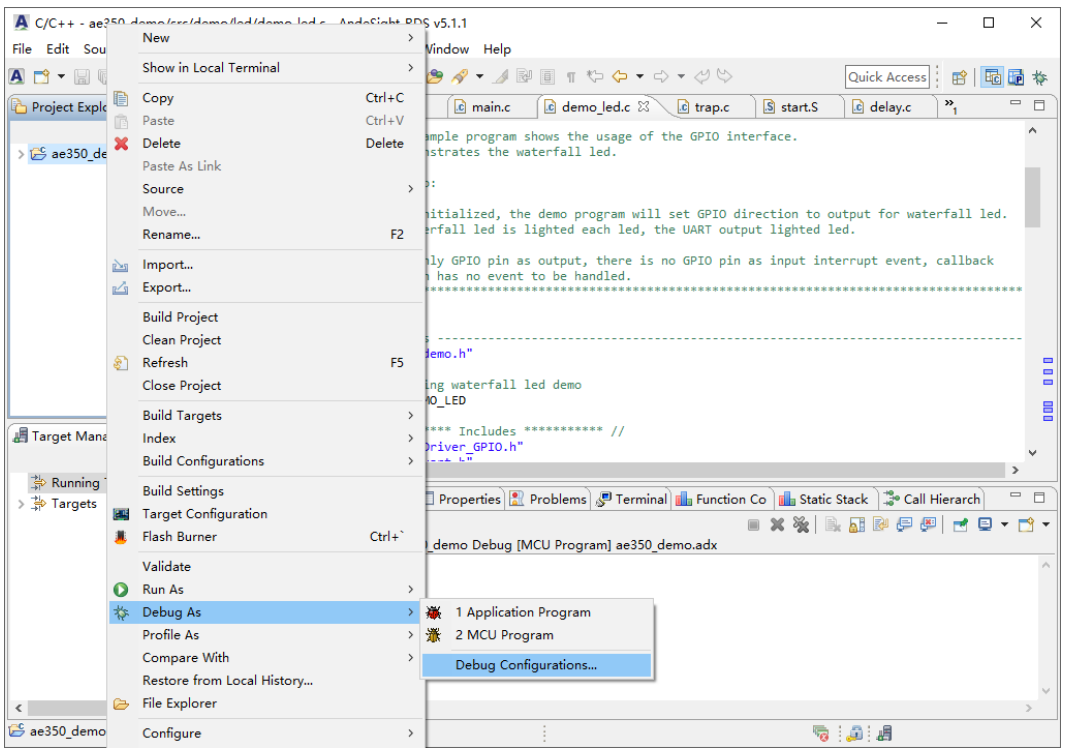
调试会话是一种使用了一组指定的程序可执行文件、调试器行为、相关调试工具和其他相关信息等组态设定的调试进程。软件工程创建调试会话配置后，单击 RDS 软件工具栏“ ”（Debug）即可开始调试会话。

4.3.1 创建调试会话配置

参照以下步骤创建调试会话配置：

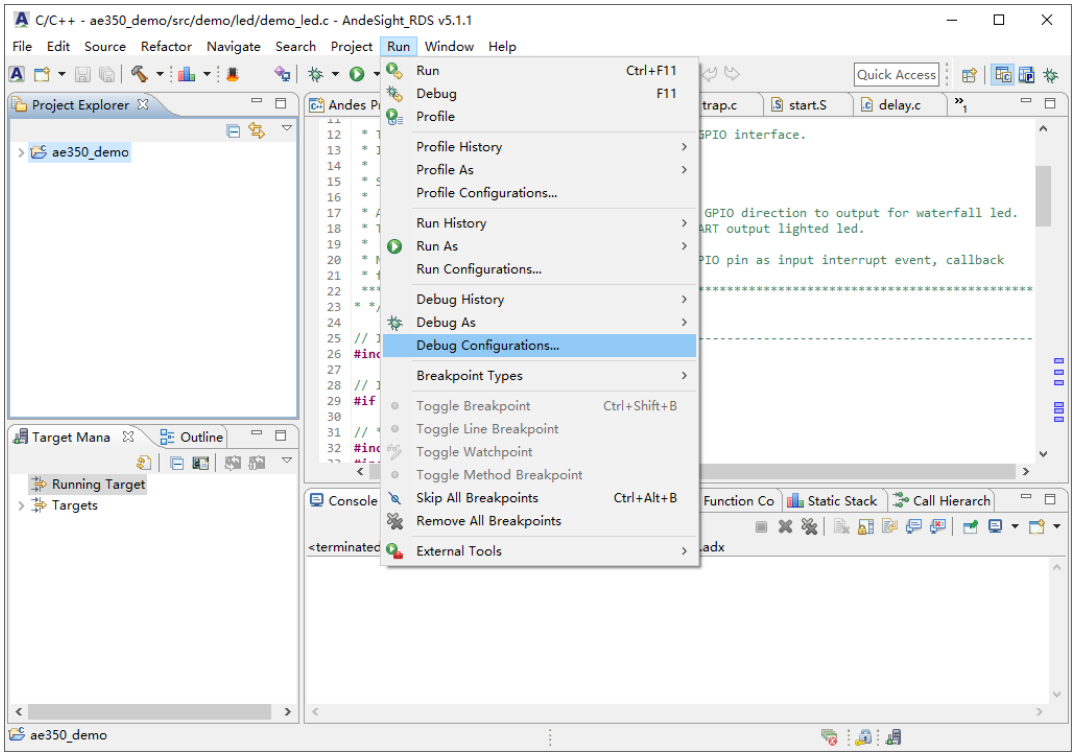
1. 项目资源管理器视图中，右单击选定的软件工程，下拉菜单中选择“Debug As > Debug Configurations…”，如图 4-55 所示。

图 4-55 选择 Debug As > Debug Configurations...



或者选择 RDS 主菜单“Run > Debug Configurations…”，如图 4-56 所示。

图 4-56 选择 Run > Debug Configurations…

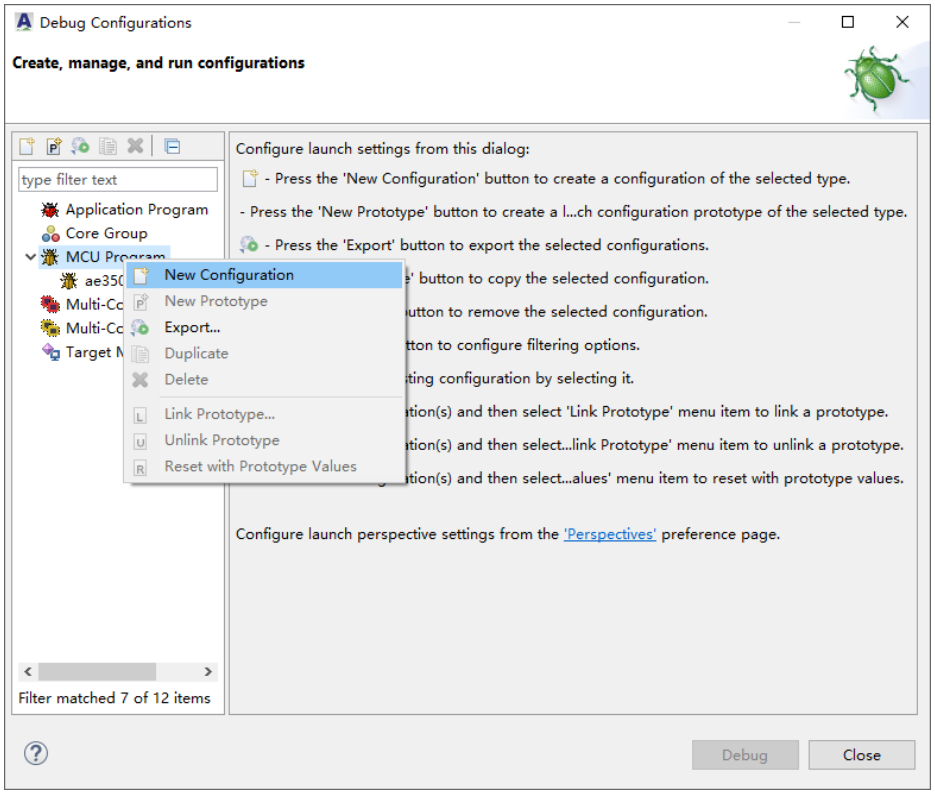


2. 在“Debug Configurations”对话框中，参照表 4-2 所示的调试配置类型，右单击选定的配置类型，例如“MCU Program”，下拉菜单中选择“New Configuration”，创建一个调试会话配置，如图 4-57 所示。

表 4-2 调试配置类型

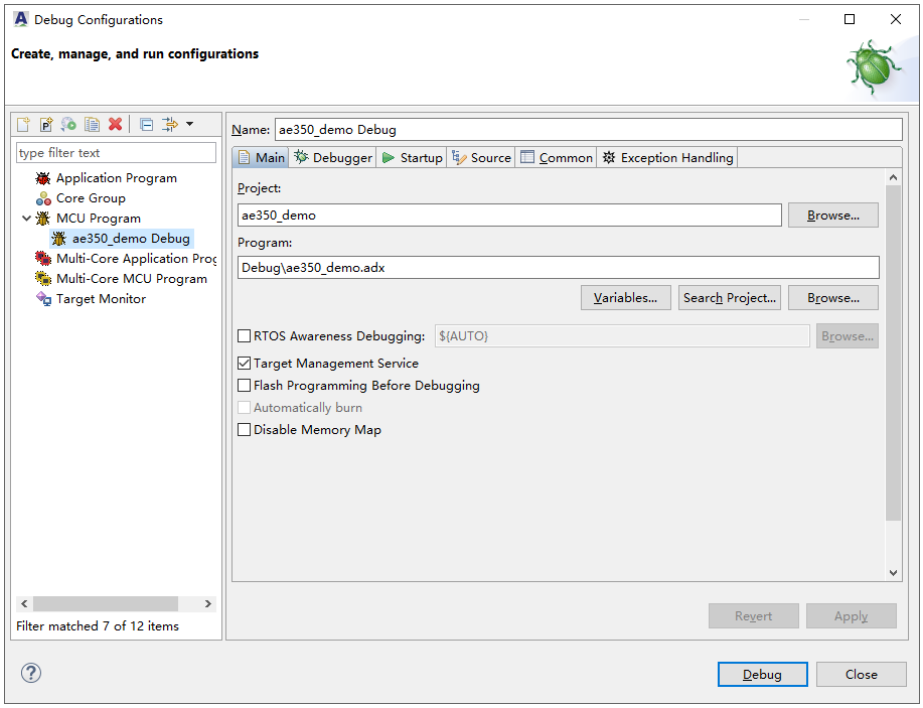
类型	描述
Application Program	用于提供系统服务的目标系统
MCU Program	用于不提供系统服务的目标系统
Core Group	用于同时发起多个调试配置，同时向内核组合发送调试命令
Multi-Core Application Program	用于提供系统服务的多核目标系统
Multi-Core MCU Program	用于不提供系统服务的多核目标系统
Target Monitor	用于检查目标的默认状态

图 4-57 创建新的调试会话配置



3. 完成每个选项卡中的配置设置，单击“Apply”和“Debug”，开始调试会话，如图 4-58 所示。

图 4-58 调试会话配置



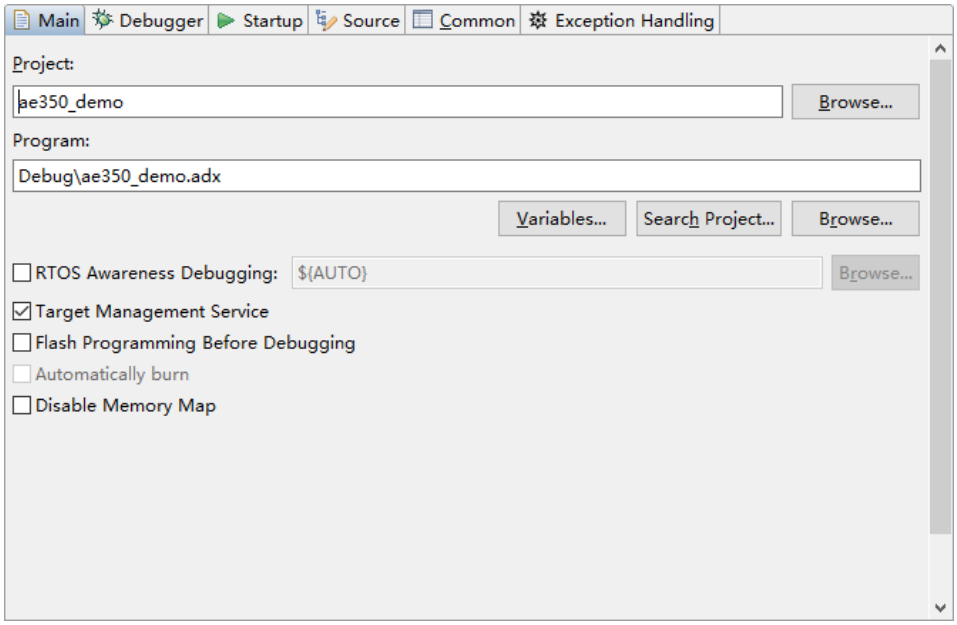
4.3.2 设置调试会话配置

以下各节详细描述如何设置“MCU Program”调试配置。

Main 选项卡

“Main”选项卡，如图 4-59 所示。

图 4-59 Main 选项卡



Main 选项描述如表 4-3 所示。

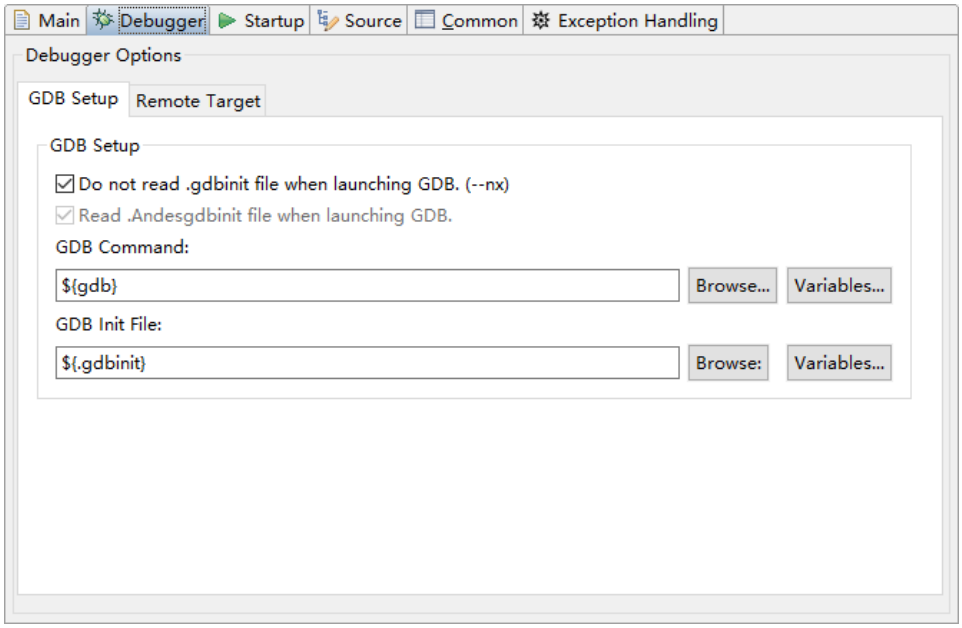
表 4-3 Main 选项

选项	描述
Project	要调试的软件工程
Program	软件工程的可执行文件
RTOS Awareness Debugging	创建用于显示 RTOS 信息的表格，此功能可根据源码中定义的符号自动检测 RTOS 内核版本，也可以单击“Browse…”手动指定一个定义了 GDB 命令的 Python 脚本
Target Management Service	目标管理服务
Disable Memory Map	加载程序可执行文件，无需映射到预定义的内存区域

Debugger 选项卡

“Debugger”选项卡，如图 4-60 所示。

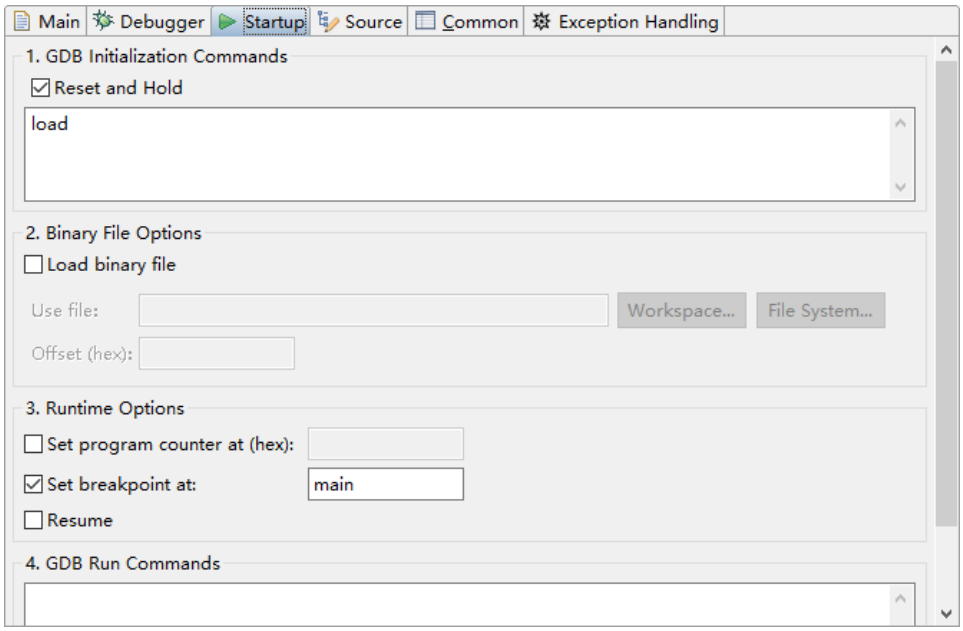
图 4-60 Debugger 选项卡



Startup 选项卡

“Startup”选项卡，如图 4-61 所示。

表 4-4 Startup 选项卡



“Startup”选项描述如表 4-4 所示。

表 4-5 Startup 选项

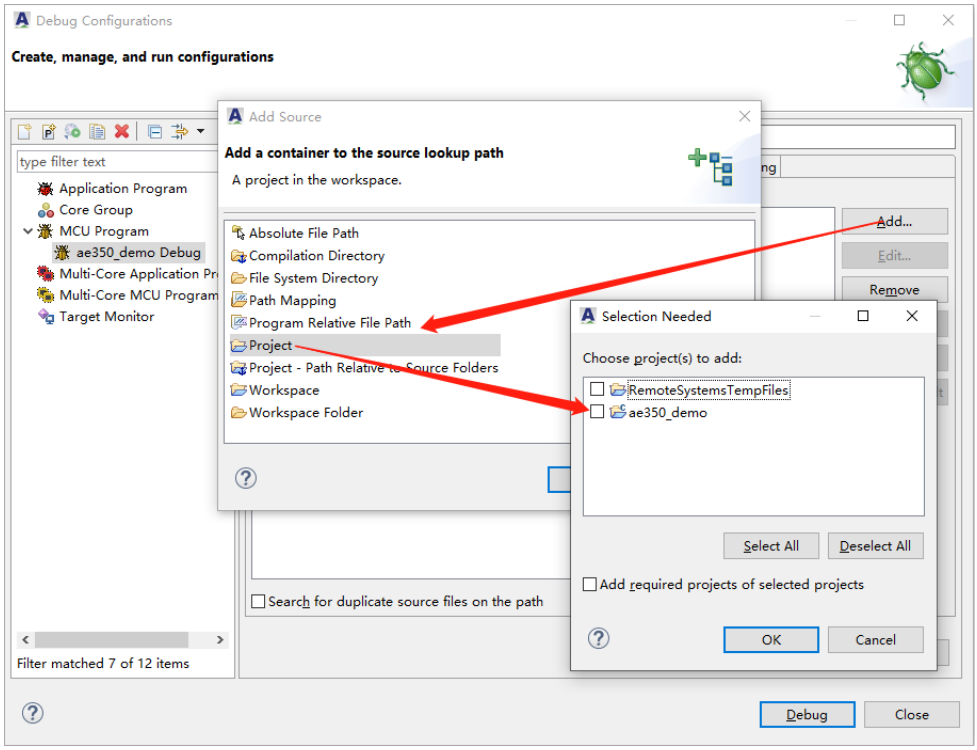
选项	描述
Reset and Hold	复位后立即保持在“_start”； load：GDB 加载程序

选项	描述
Set breakpoint at	设置断点位置，例如“main”
GDB Run Commands	GDB 运行命令

Source 选项卡

“Source”选项卡，如图 4-62 所示。

图 4-61“Source”选项卡

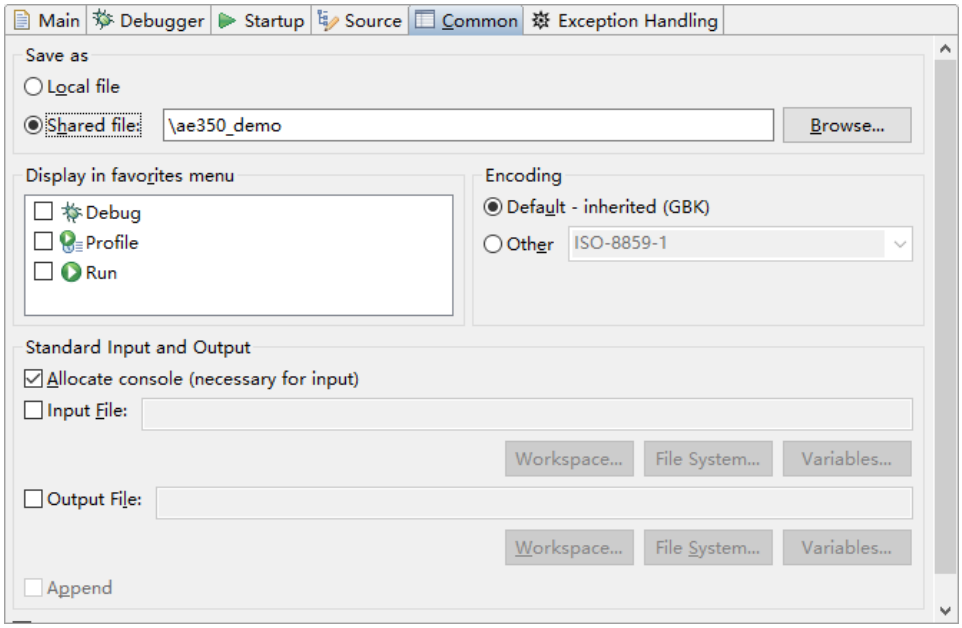


此选项卡，可以指定用于调试 C 或 C++应用程序的源文件，例如，可以指定一个包含共享库的软件工程，并加入查找路径。

Common 选项卡

“Command”选项卡，如图 4-62 所示。

图 4-62 Common 选项卡



“Common”选项描述如表 4-5 所示。

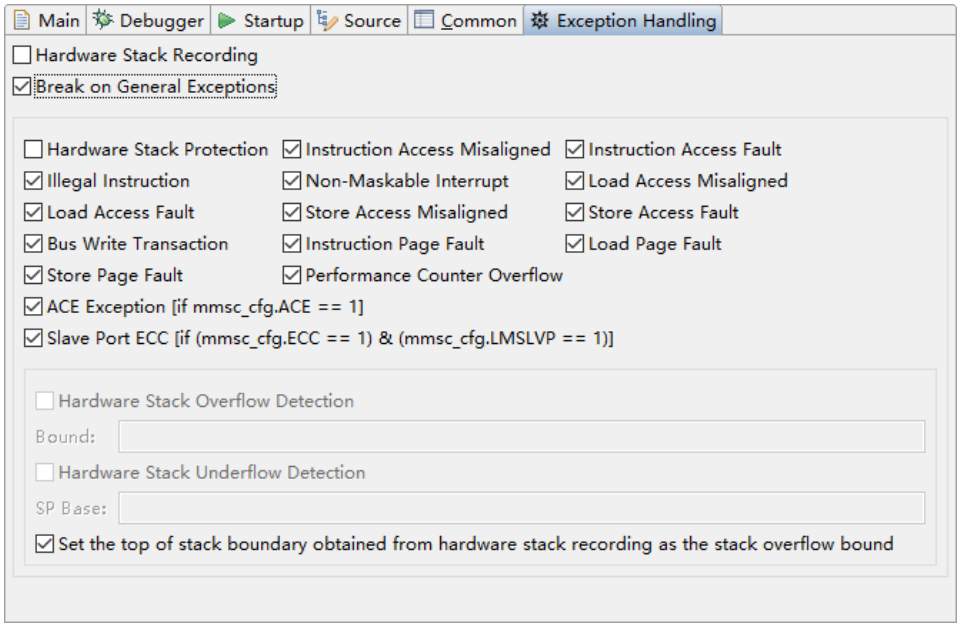
表 4-6 Common 选项

选项	描述
Save as	启动配置文件保存为文件（*.launch）
Standard Input and Output	Allocate console (necessary for input)：设置控制台视图作为输入或输出提示符，具体参考 *4.2.6 控制台视图*; Input File：从项目资源（单击“Workspace…”）或文件（单击“File system…”）分配标准输入； Output File：从项目资源（单击“Workspace…”）或文件（单击“File system…”）分配标准输出

Exception Handling 选项卡

“Exception Handling”选项卡，如图 4-63 所示。

图 4-63 Exception Handling 选项卡



此选项卡用于开启硬件堆栈记录，指定程序执行期间必须捕获的异常类型，详细描述请参照第 5 章 通用异常处理和堆栈记录/保护。注意不能同时选择“Hardware Stack Recording”和“Hardware Stack Protection”。

4.4 断点和观察点

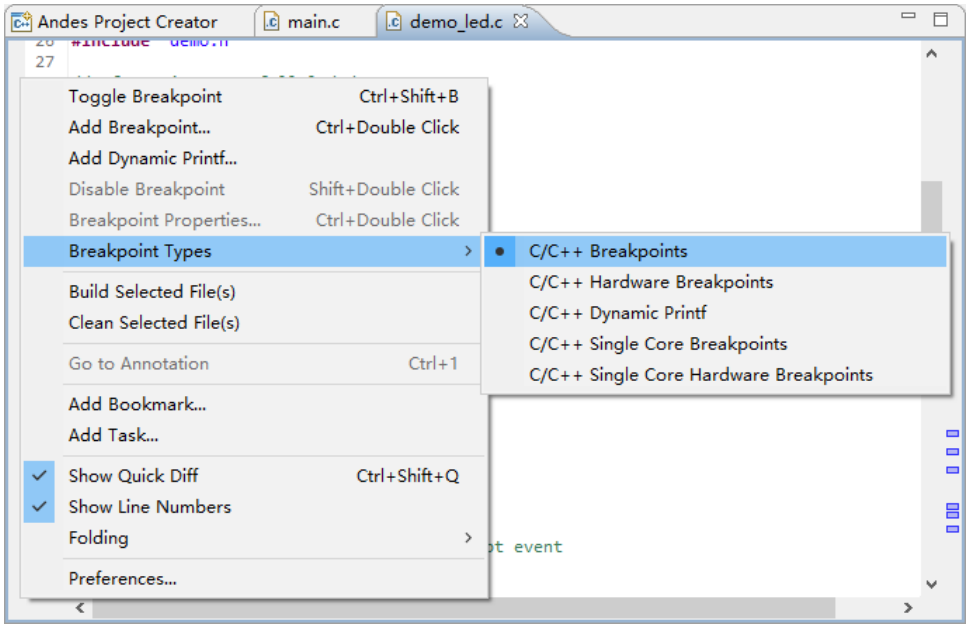
RDS 软件可以设置的断点和观察点个数，取决于目标所支持的硬件断点个数，请参考目标的数据手册以确定所支持的硬件断点或观察点的个数。

4.4.1 设置断点

参照以下步骤设置断点：

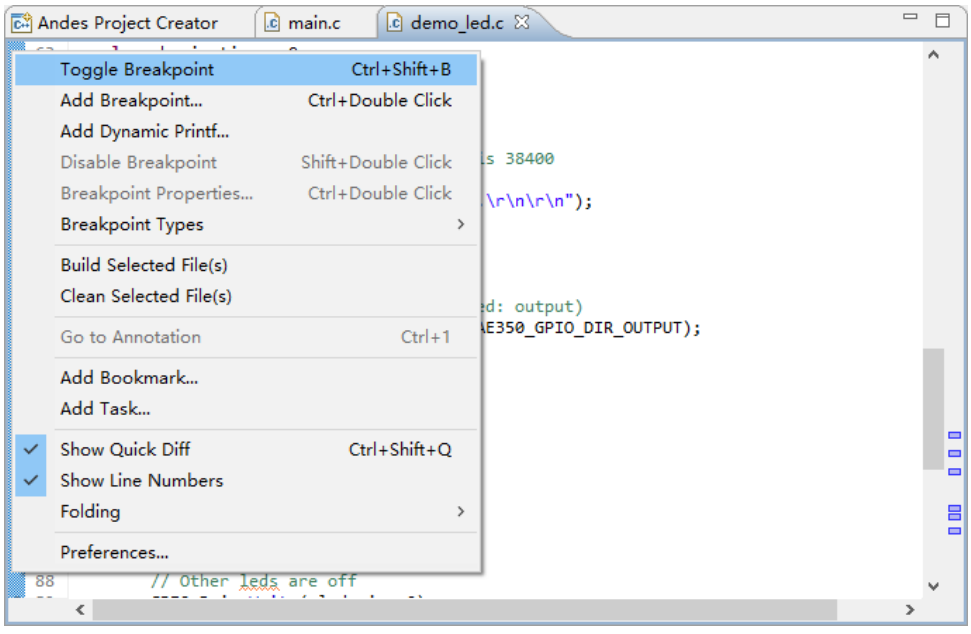
1. 代码编辑器打开一个源文件。
2. 右单击代码编辑器左侧边缘的标尺，下拉菜单中选择“Breakpoint Types > C/C++ Hardware Breakpoints”或“C/C++ Breakpoints”，指定断点类型，如图 4-64 所示。

图 4-64 选择 Breakpoint Types



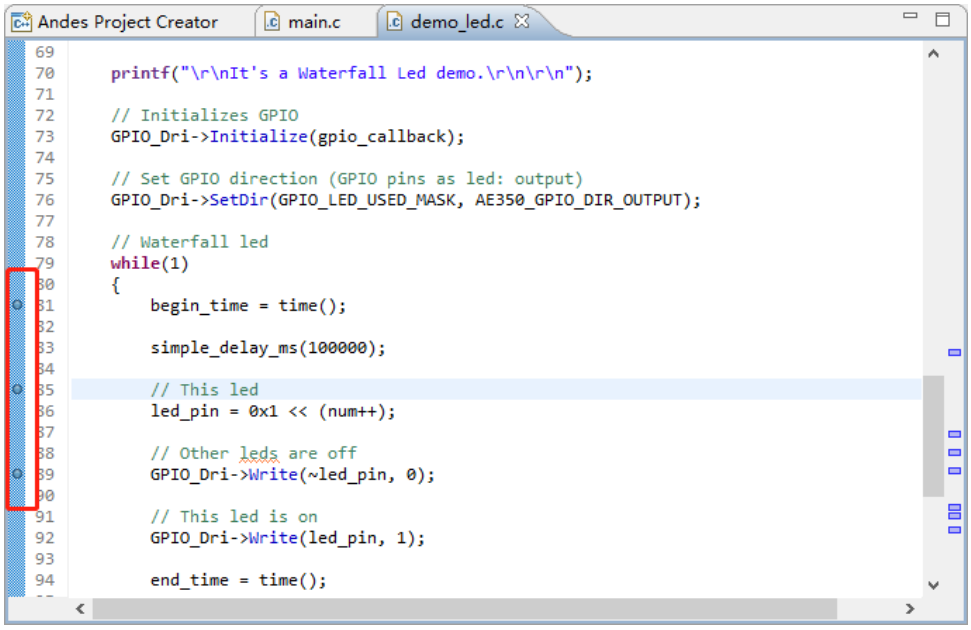
3. 在需要设置断点的行的左侧，双击标尺，或右单击标尺，下拉菜单中选择“Toggle Breakpoint”，如图 4-65 所示。

图 4-65 选择“Toggle Breakpoint”



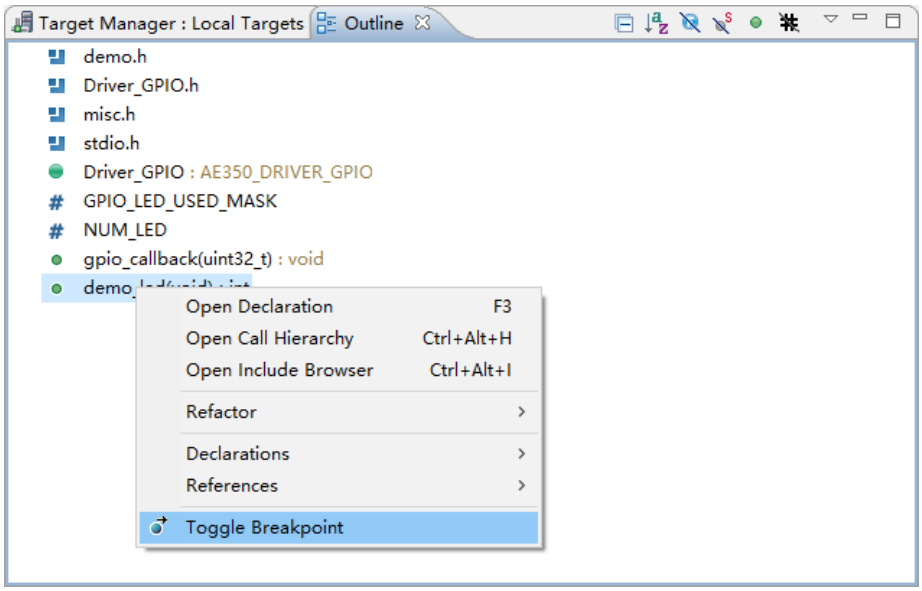
标尺上显示的断点标记，表示想要程序执行暂停的位置，如图 4-66 所示。

图 4-66 断点



也可以在大纲视图（Outline View）中设置函数断点，右单击一个函数，下拉菜单中选择“Toggle Breakpoint”，如图 4-67 所示。

图 4-67 大纲视图选择 Toggle Breakpoint



4. 指定断点的触发条件，在断点视图中或代码编辑器标尺上，右单击一个断点，下拉菜单中选择“Breakpoint Properties…”，如图 4-68 和图 4-69 所示。

图 4-68 断点视图选择 Breakpoint Properties...

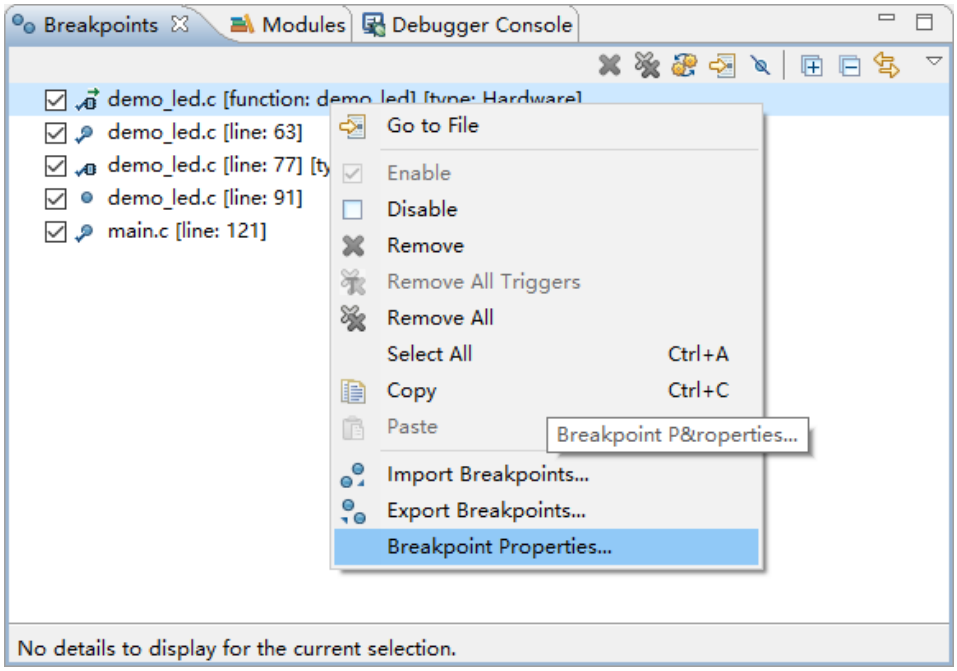
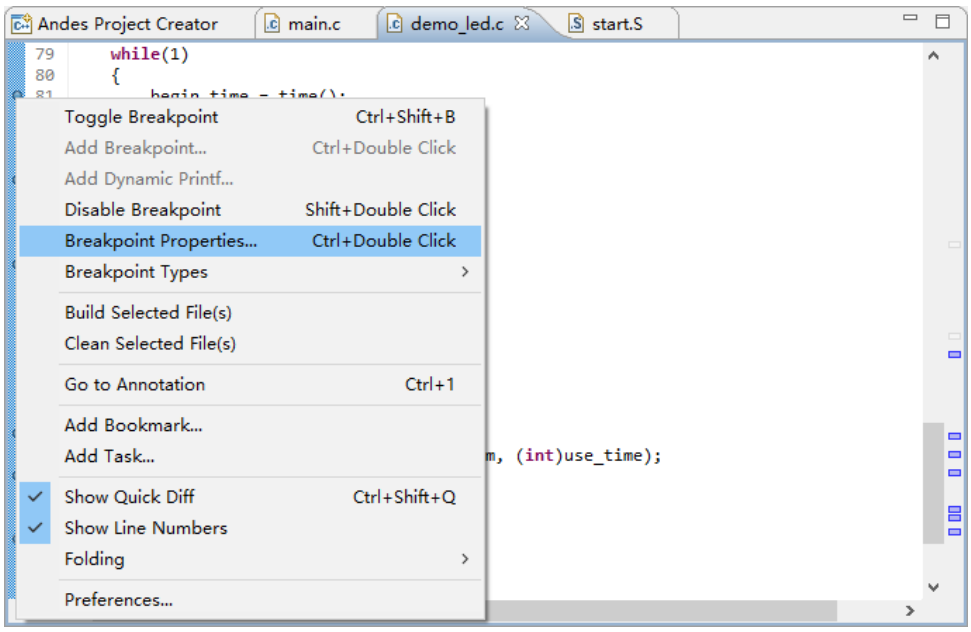
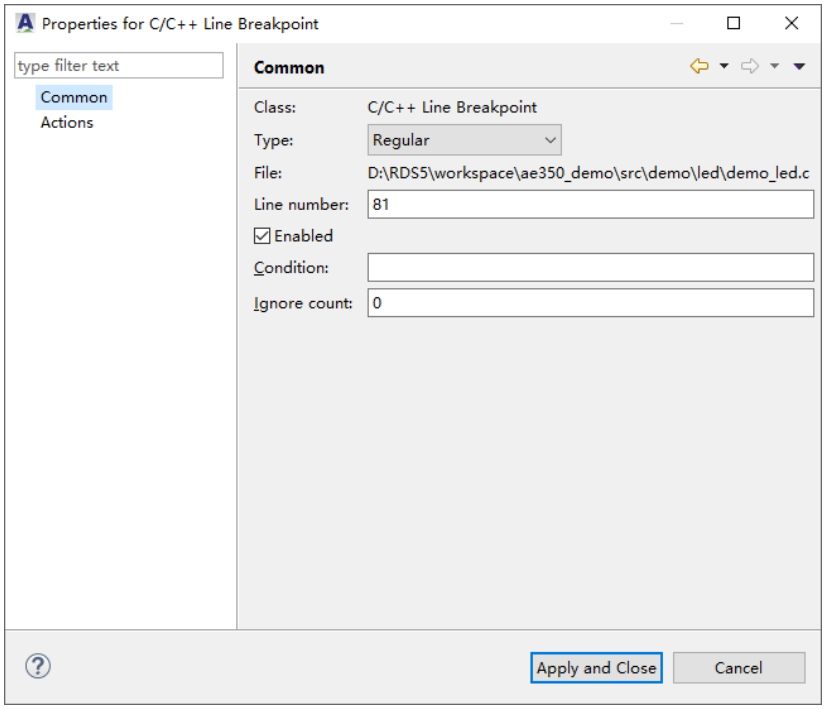


图 4-69 代码编辑器选择 Breakpoint Properties...



“Properties for C/C++ Line Breakpoint”对话框中，选择“Common”，“Condition:”输入条件表达式，单击“Apply and Close”，如图 4-70 所示。

图 4-70 Properties for C/C++ Line Breakpoint



5. 如需清除断点，双击代码编辑器的断点标记即可。

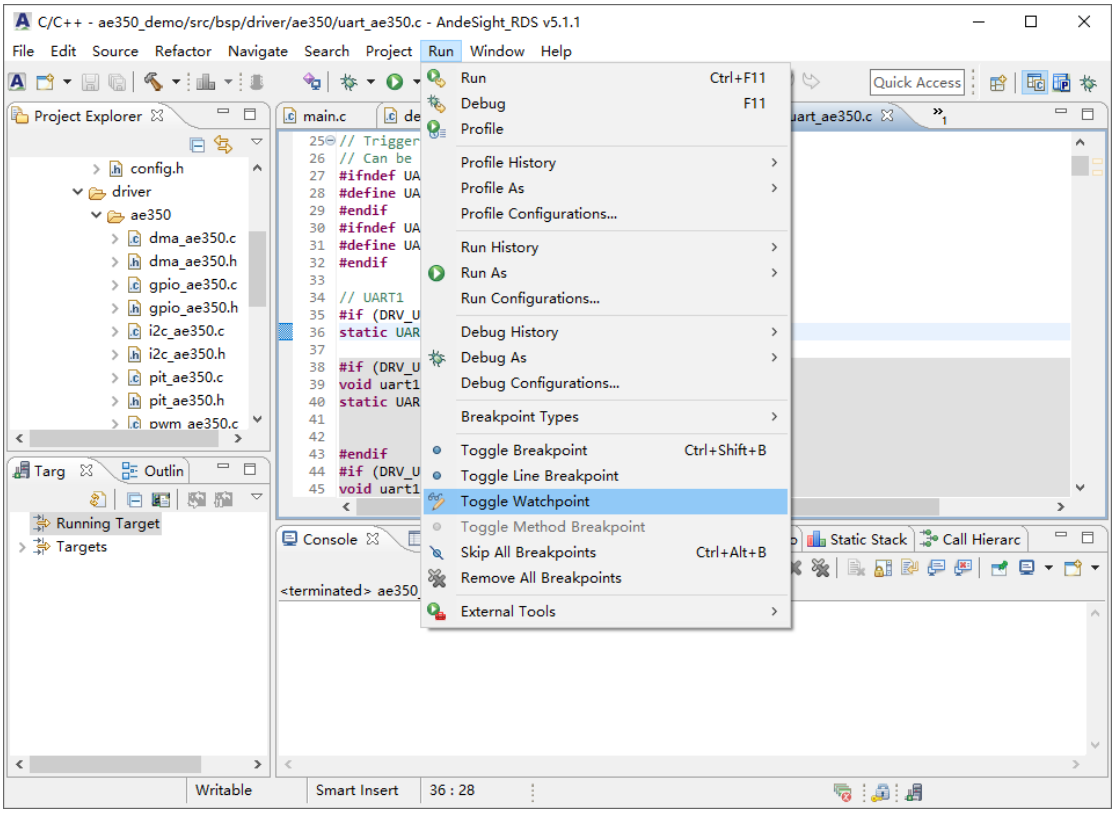
4.4.2 设置观察点

不同于断点，观察点只在变量或表达式的值发生变化时，才停止程序执行。

请参照以下步骤设置观察点：

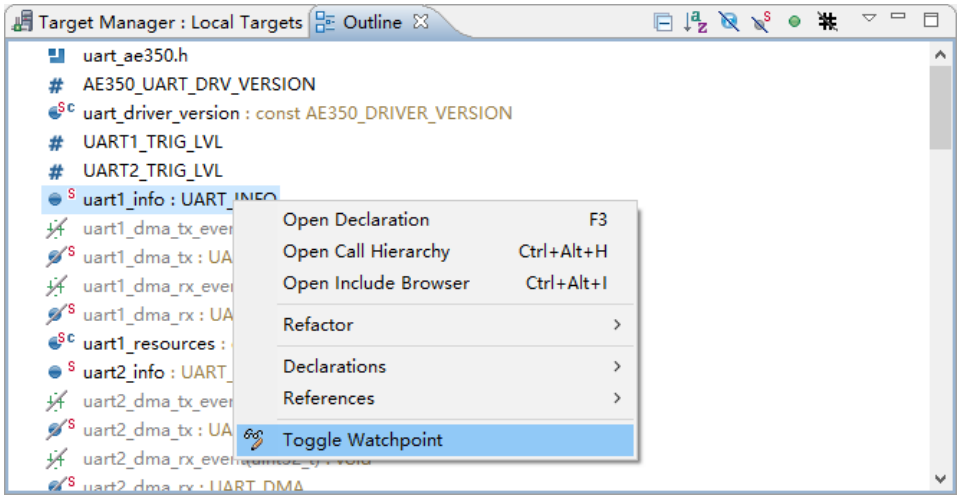
1. 可参照以下几种方法设置观察点：
   - 代码编辑器：在代码编辑器中，选择想要观察的全局变量，选择 RDS 软件主菜单“Run > Toggle Watchpoint”，如图 4-71 所示。

图 4-71 选择 Run > Toggle Watchpoint



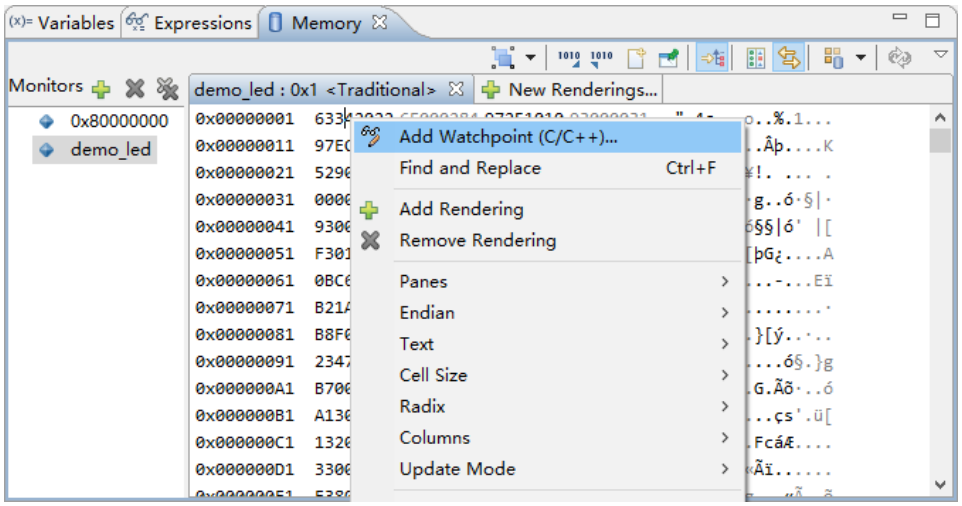
- 大纲视图：在大纲视图中，选择想要观察的全局变量，右单击全局变量，下拉菜单中选择“Toggle Watchpoint”，如图 4-72 所示。

图 4-72 大纲视图选择 Toggle Watchpoint



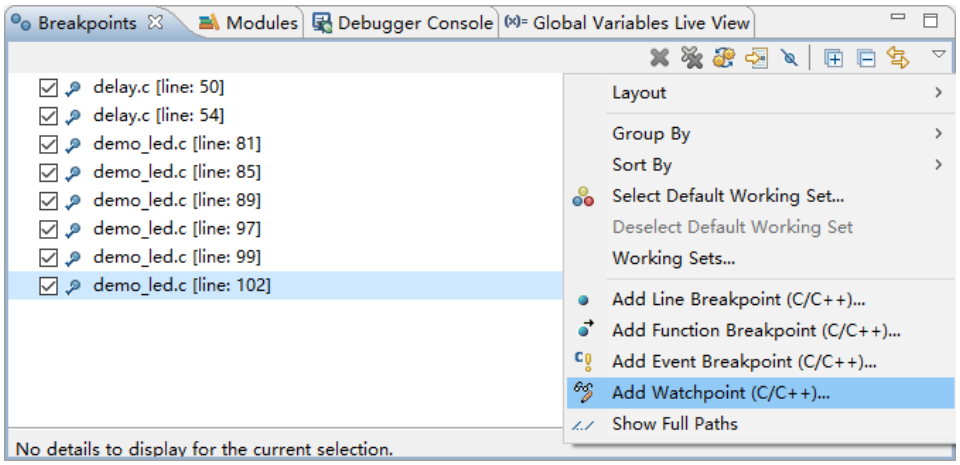
- 内存视图：在内存视图中，右单击内存渲染的一个想要的可寻址单元或单元范围，下拉菜单中选择“Add Watchpoint (C/C++)...”，如图 4-73 所示。

图 4-73 内存视图选择 Add Watchpoint (C/C++)...



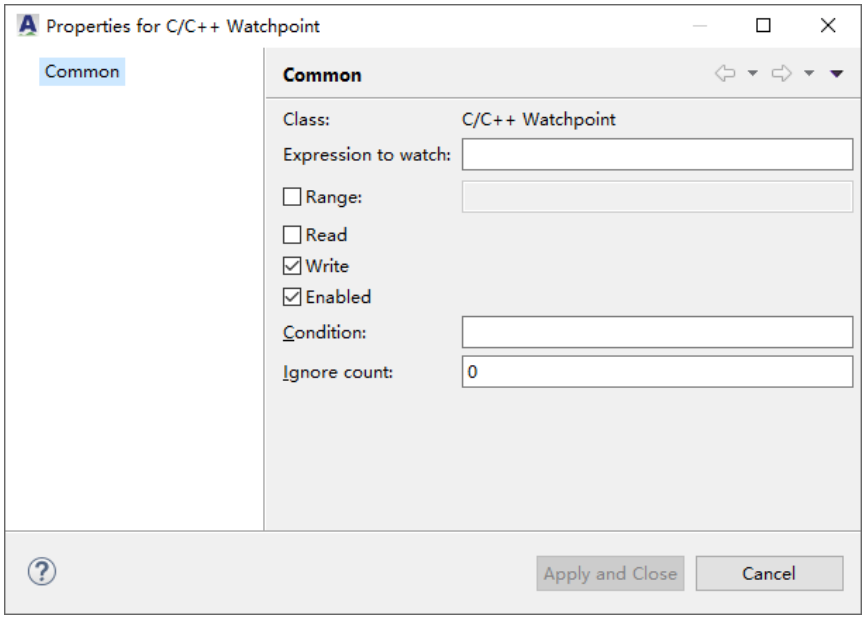
- 断点视图：在断点视图中，单击断点视图工具栏“ ”（View Menu），下拉菜单中选择“Add Watchpoint (C/C++)…”，如图 4-74 所示。

图 4-74 断点视图选择 Add Watchpoint (C/C++)...



2. “Properties for C/C++ Watchpoint”对话框中，指定观察点的各种设置，单击“Apply and Close”，如图 4-75 所示。

图 4-75 Properties for C/C++ Watchpoint



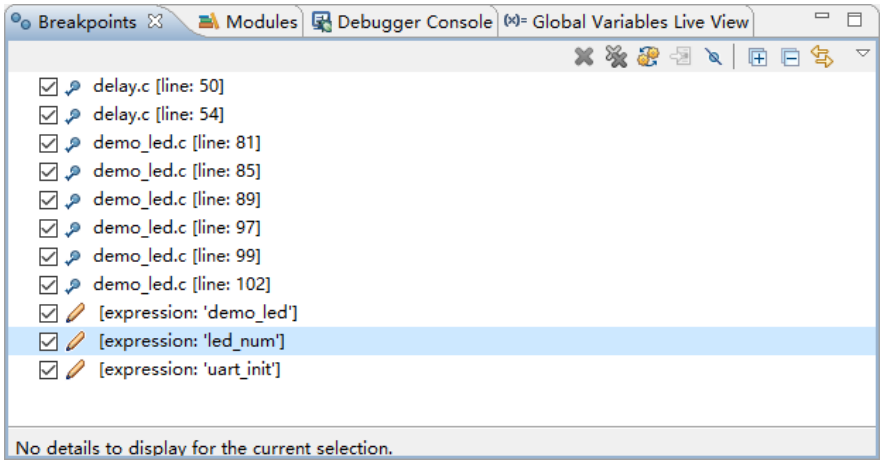
观察点的选项描述如表 4-7 所示。

表 4-7 观察点选项

选项	描述
Expression to watch	如果要在“Range”选项中指定要监视的区域，请在这里输入一个常量地址或变量地址（&i）
Range	指定由观察点监视的可寻址单元
Read	在读取监视的表达式时停止程序执行
Write	在写入监视的表达式时停止程序执行
Enabled	开启观察点
Conditions	指定一个在观察点被击中时计算的表达式
Ignore count	指定表达式的值的变化次数

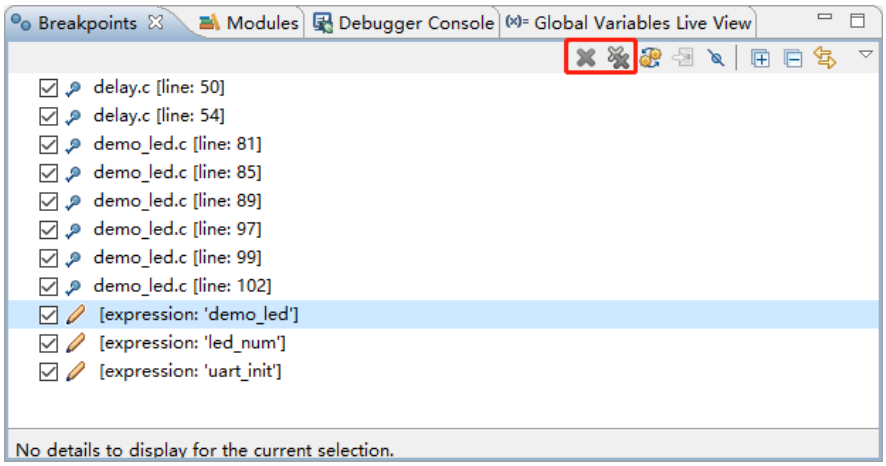
3. 断点视图列出了所有设置的观察点，观察点标记“ ”表示写观察点，“ ”表示读观察点，如图 4-76 所示。

图 4-76 断点视图中的观察点



4. 清除观察点，在断点视图中选中要删除的观察点，单击断点视图工具栏“”（Remove Selected Breakpoints），如图 4-77 所示。

图 4-77 清除断点视图中的观察点



5 通用异常处理和堆栈记录/保护

5.1 堆栈记录/保护

对于支持硬件堆栈保护的 CPU 内核，RDS 软件允许硬件栈顶记录或硬件堆栈保护，但是这两种功能不能同时使用。硬件栈顶记录功能，记录程序运行时的栈顶边界，帮助用户查找应用程序所需的堆栈内存大小。硬件堆栈保护功能，报告堆栈溢出或下溢问题。

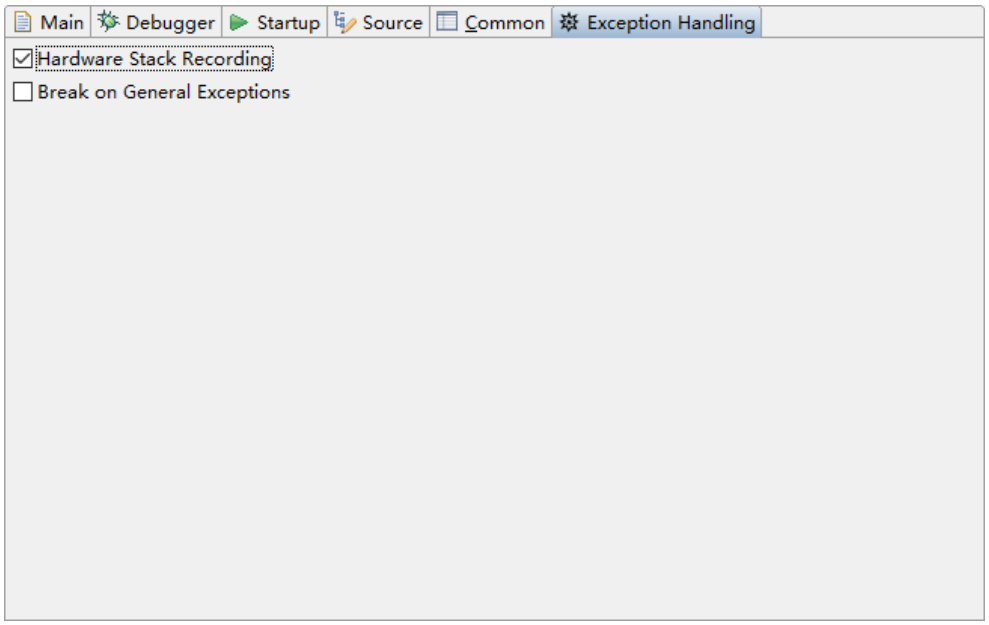
须满足以下条件：

- 不能使用程序代码修改用于硬件堆栈保护的系统寄存器的值
- 可用的硬件断点数量有限，必须预留一个给堆栈保护处理程序使用

请参照以下步骤开启硬件堆栈记录功能：

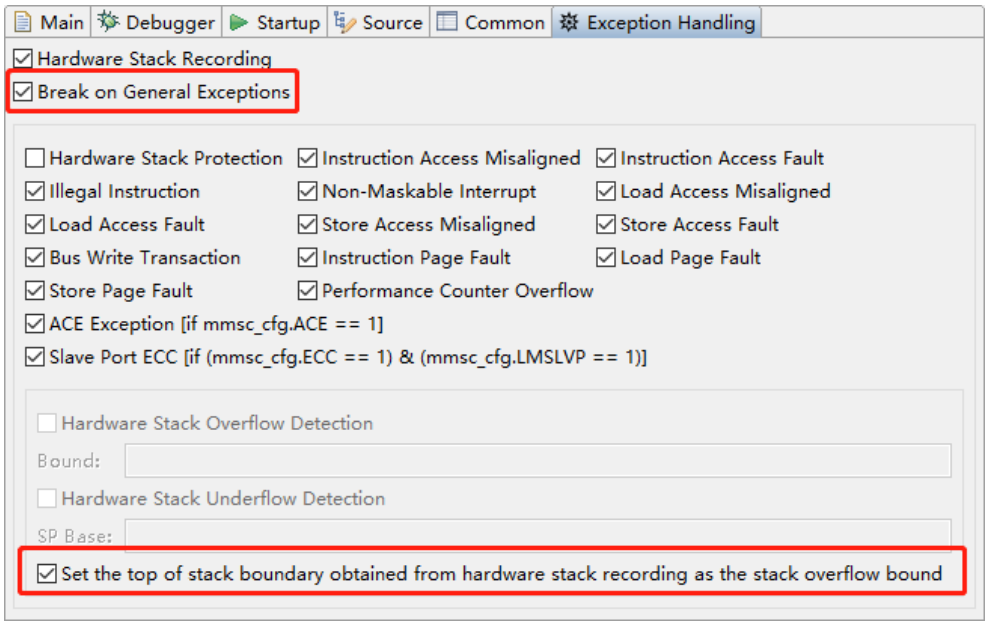
1. 参照 4.3.1 创建调试会话配置步骤 1 和步骤 2，创建一个调试会话配置，“Debug Configurations”对话框中，单击“Exception Handling”选项卡，选择“Hardware Stack Recording”，如图 5-1 所示。

图 5-1 选择 Hardware Stack Recording



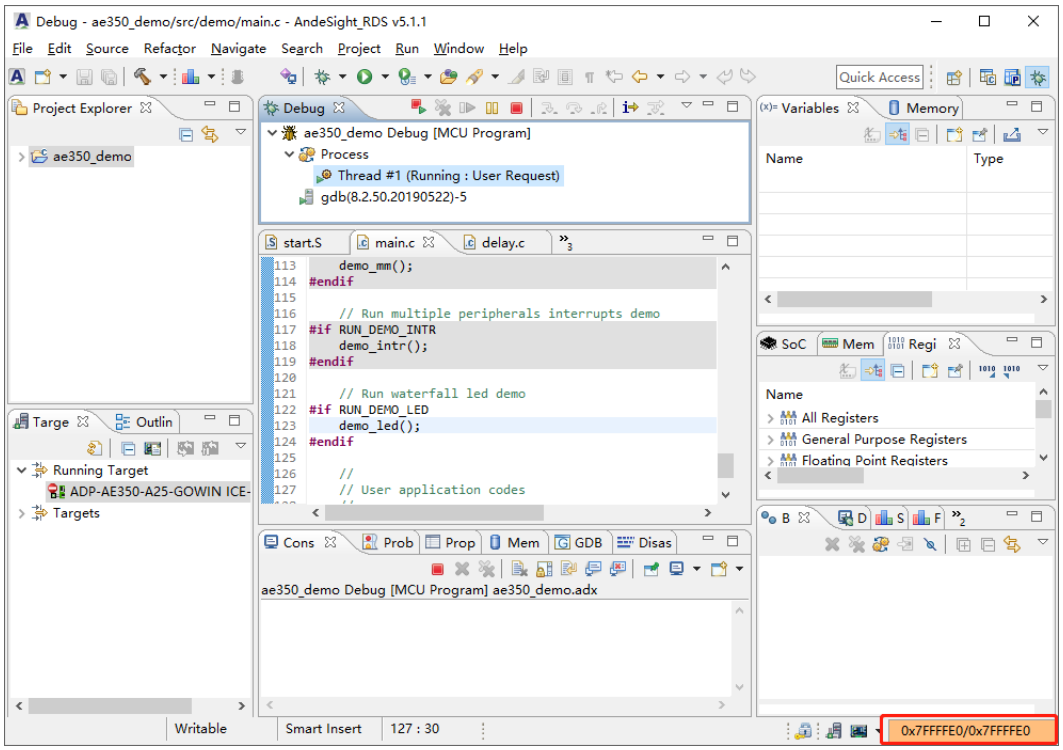
如果想要把调试会话中记录的最大栈顶边界设置为栈溢出检测的边界，须选择“Break on General Exceptions > Set the top of stack boundary obtained from hardware stack recording as the stack overflow bound”，如图 5-2 所示。

图 5-2 设置栈顶边界



2. “Debug Configurations”对话框中，单击“Apply”和“Debug”，开始调试会话。
3. 开始调试会话后，RDS 软件自动进入调试透视图，单击调试视图工具栏“”（Resume），继续执行程序。RDS 软件右下角状态栏显示程序运行时的栈顶边界“Top”的值，如图 5-3 所示。

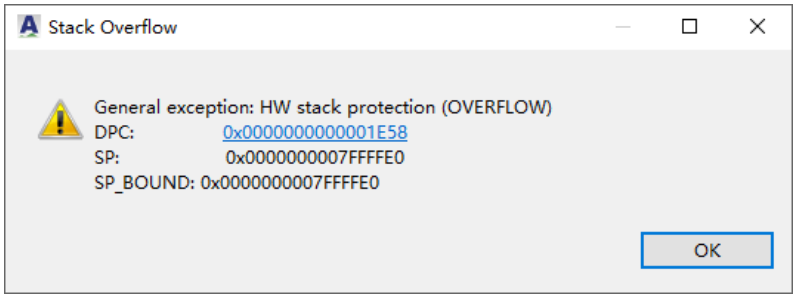
图 5-3 显示栈顶边界



5.2 通用异常处理

RDS 软件允许在选定的通用异常类型发生时停止程序执行，“Stack Overflow”对话框提示异常信息，如图 5-4 所示。

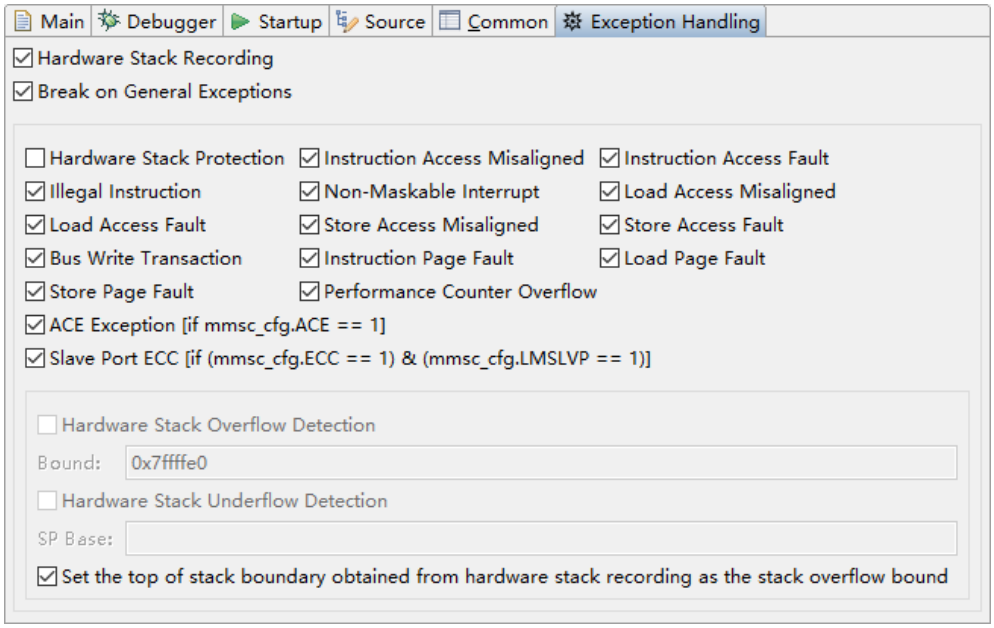
图 5-4 Stack Overflow



参照以下步骤了解如何使用 RDS 软件捕获调试会话中的通常异常类型。如果中断向量在 ROM 或 Flash 中，那么程序执行期间指定的异常类型会占用一个硬件断点，这种情况下，首先须确保目标系统有足够可用的硬件断点。

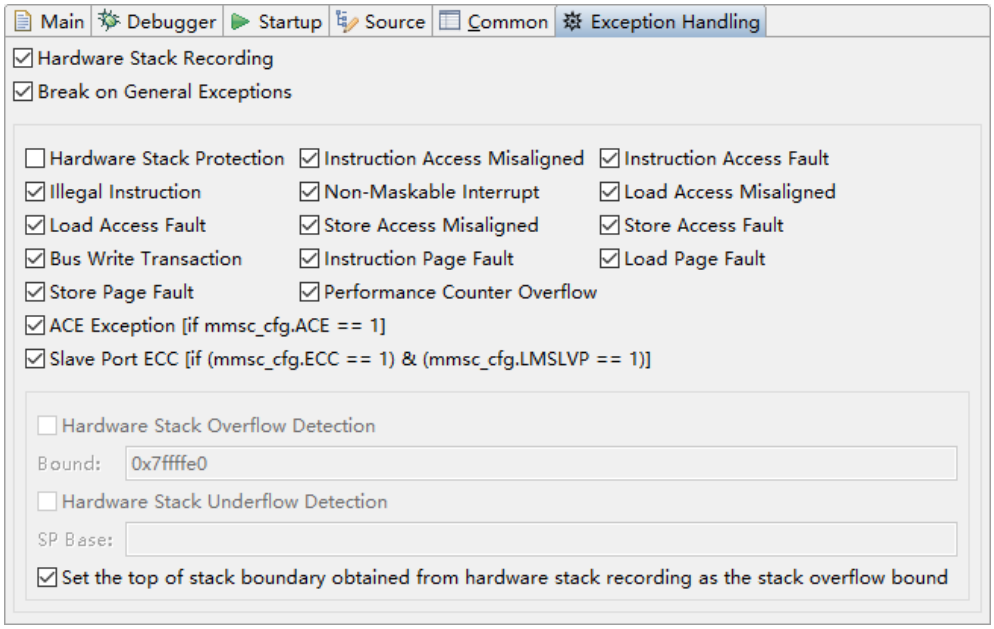
1. 参照 4.3.1 创建调试会话配置步骤 1 和步骤 2，创建一个调试会话配置，“Debug Configurations”对话框中，单击“Exception Handling”选项卡，选择“Break on General Exceptions”，如图 5-5 所示。

图 5-5 选择 Break on General Exceptions



2. 指定在程序执行期间要捕获的一个或多个异常类型，如图 5-6 所示。

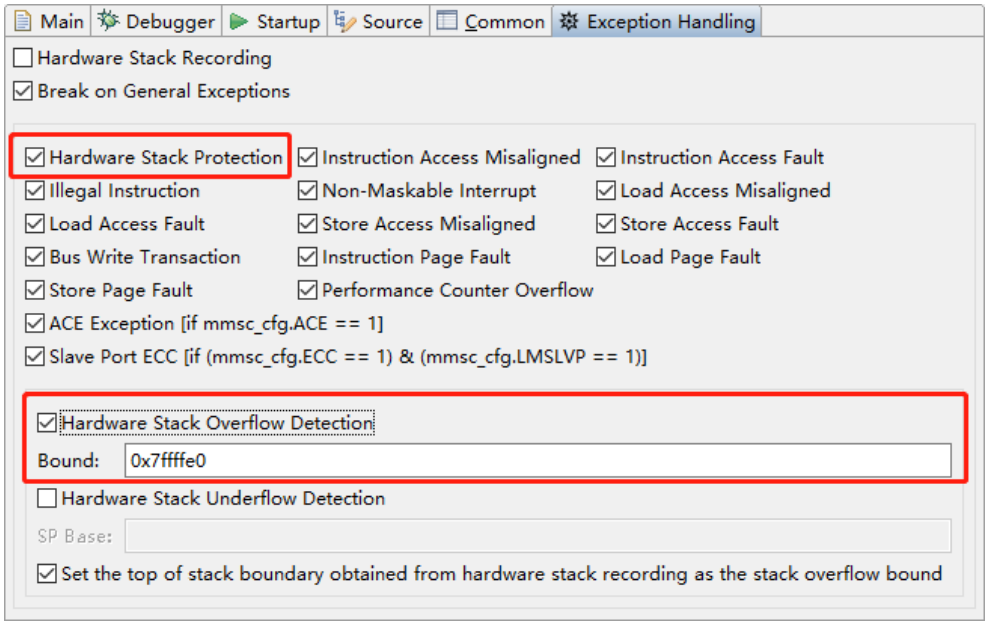
图 5-6 指定异常类型



对于“Hardware Stack Protection”，须指定是否执行“Hardware Stack Overflow Detection”和/或“Hardware Stack Underflow Detection”，如图 5-7 所示。对于“Hardware Stack Overflow Detection”，“Bound:”输入堆栈边界寄存器的值，用于与堆栈指针（SP）寄存器的值作比较。对于“Hardware Stack Underflow Detection”，“SP Base:”输入堆栈基地址寄存器的值，用于与 SP 寄存器的值作比较。如果之前已选过“Hardware Stack Recording”和“Set the top of stack boundary

obtained from hardware stack recording as the stack overflow bound”，则“Hard Stack Overflow Detection”的“Bound:”会自动被设置为已记录的堆栈边界最大值。

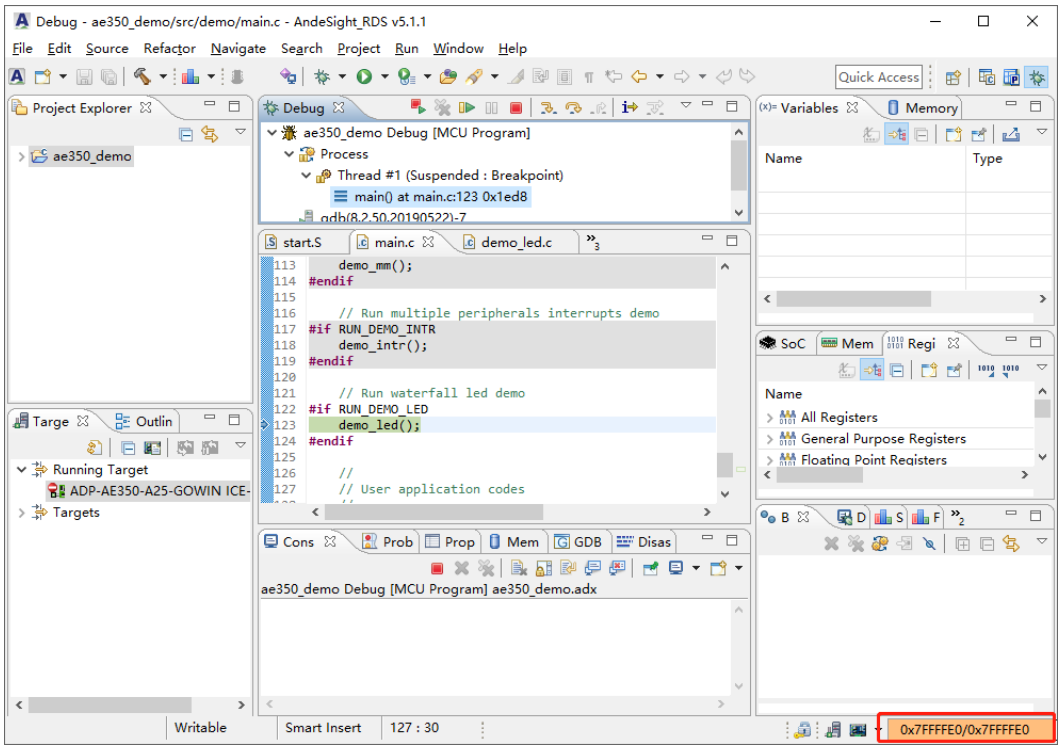
图 5-7 选择 Hardware Stack Protection



3. “Debug Configurations”对话框中，单击“Apply”和“Debug”，开始调试会话。
4. 开始调试会话后，RDS 软件自动进入调试透视图，在源码中设置断点，单击调试视图工具栏“”（Resume），继续执行程序。

如果已指定“hardware Stack Protection”，则 RDS 软件右下角状态栏显示程序运行时的堆栈保护状态，如图 5-8 所示。

图 5-8 显示堆栈保护状态



5. 每当发生指定的异常时，都会弹出一个对话框提示相关的信息，如果要在源码中定位发生异常的位置，只需单击对话框中的 IPC 值。

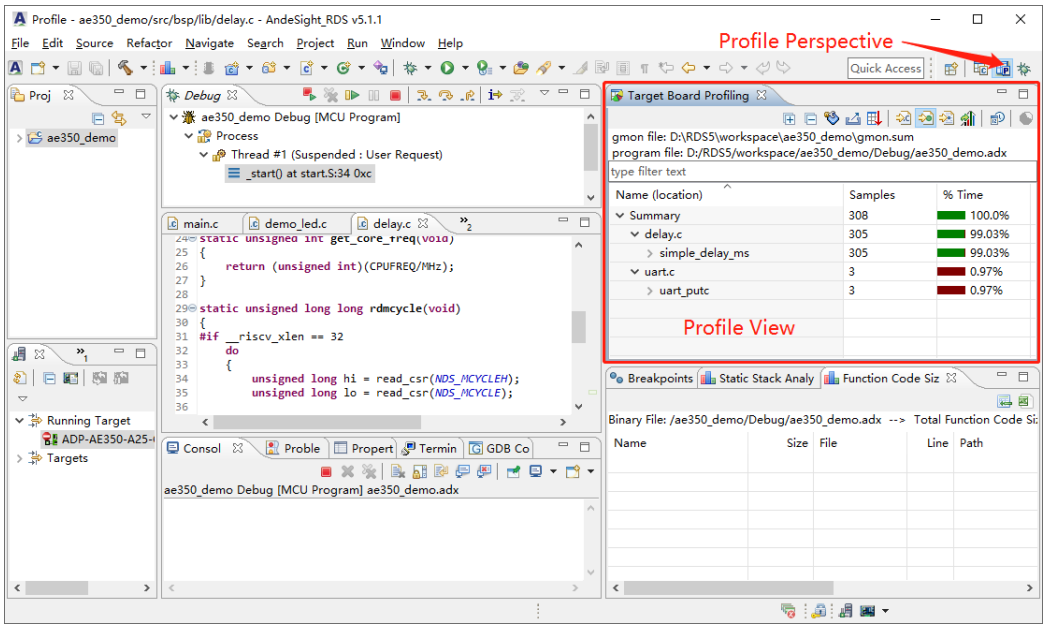
6 分析软件工程

RDS 软件可以在目标板上实时分析 GCC 编译器构建的软件工程，当程序由于断点暂停执行或单步执行暂停后，会立即收集、统计和分析相关的信息。

6.1 分析透视图

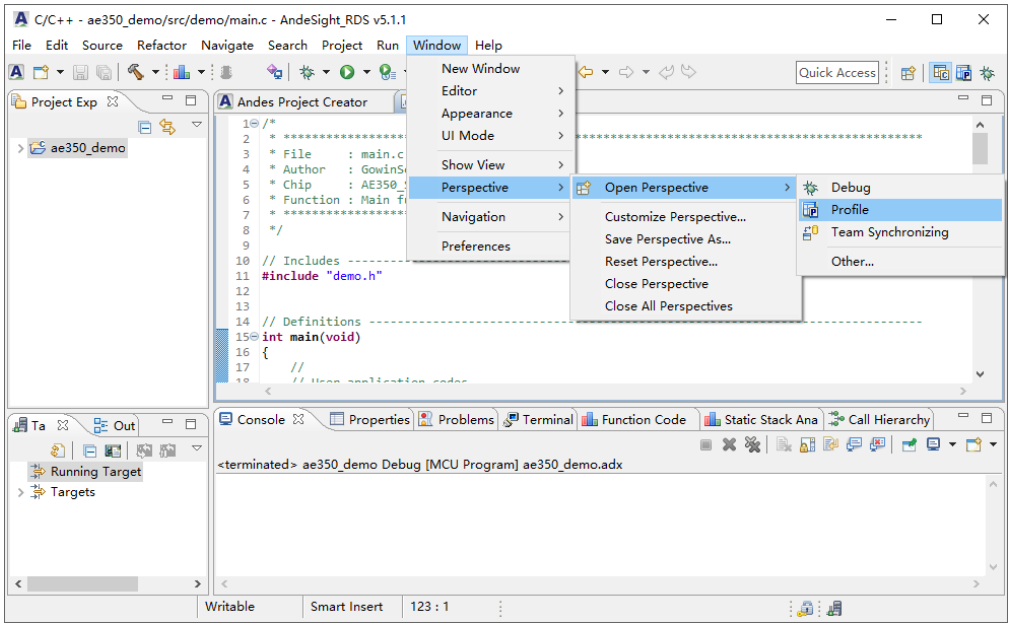
分析透视图（Profile Perspective），包含分析视图（Profile View），用于显示分析数据，如图 6-1 所示。

图 6-1 分析透视图



开始分析会话（Profile Session）后，RDS 软件自动进入分析透视图，或者，选择 RDS 软件主菜单“Window > Perspective > Open Perspective > Profile”，手动进入分析透视图，如图 6-2 所示。

图 6-2 选择 Window > Perspective > Open Perspective > Profile

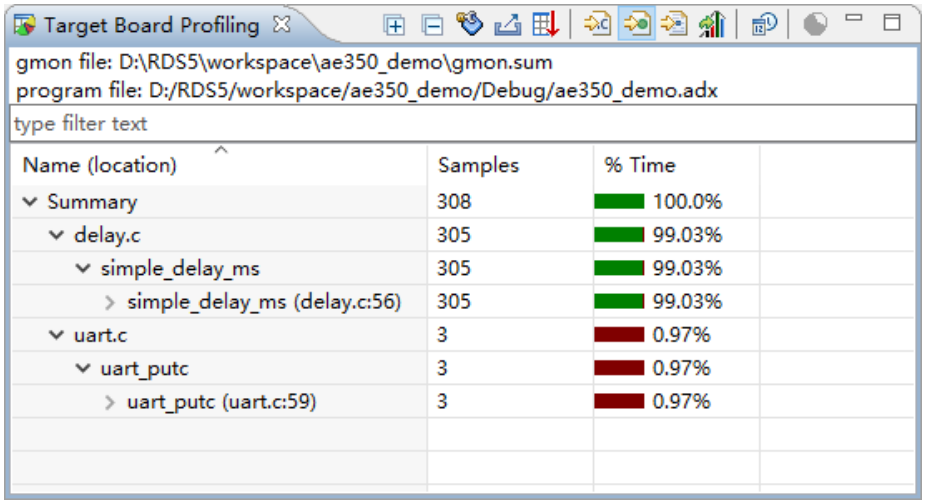


或者也可以单击 RDS 软件透视图工具栏“ ”（Profile），进入分析透视图。

6.2 目标板分析视图

目标板分析视图（Target Board Profiling View）用于目标系统的分析视图，显示程序哪些部分占用了最多的执行时间，提供每个函数的调用图信息，可以使用分析数据来创建图表，如图 6-3 所示。此视图提供了一个实时接口，用来跟踪和调查程序执行的瓶颈。

图 6-3 目标板分析视图



作为分析透视图的一部分，开始分析会话后，自动启动目标板分析视图。也可以选择 RDS 软件主菜单“Windows > Show View > Other…”，如图 6-4 所示。“Show View”对话框中，选择“Profile > Target Board

Profiling”，手动启动目标板分析视图，如图 6-5 所示。

图 6-4 选择 Windows > Show View > Other...

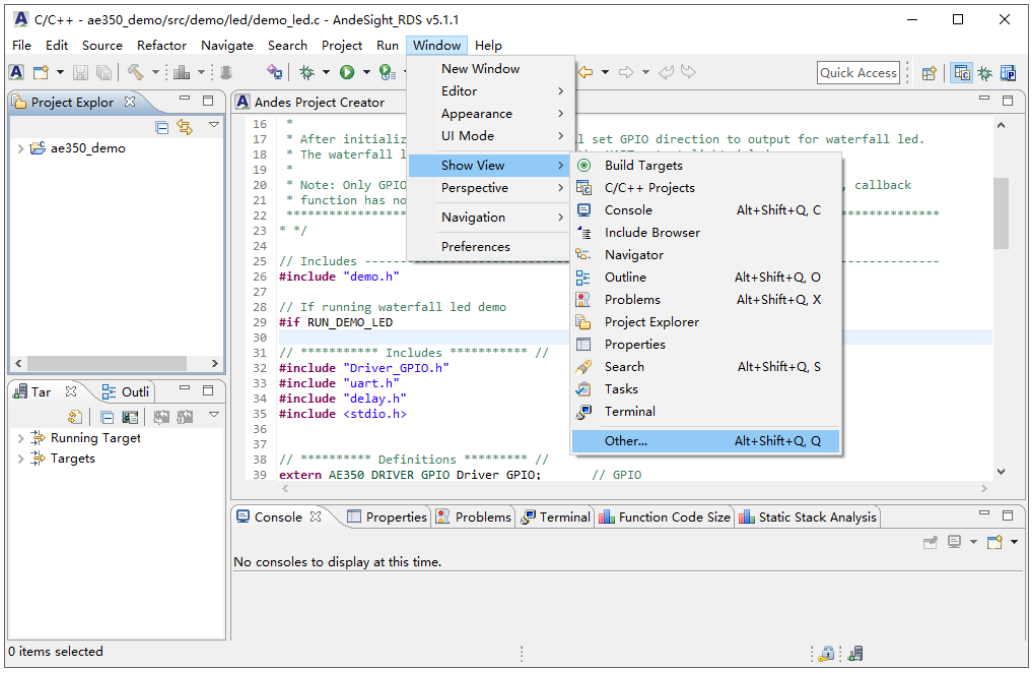
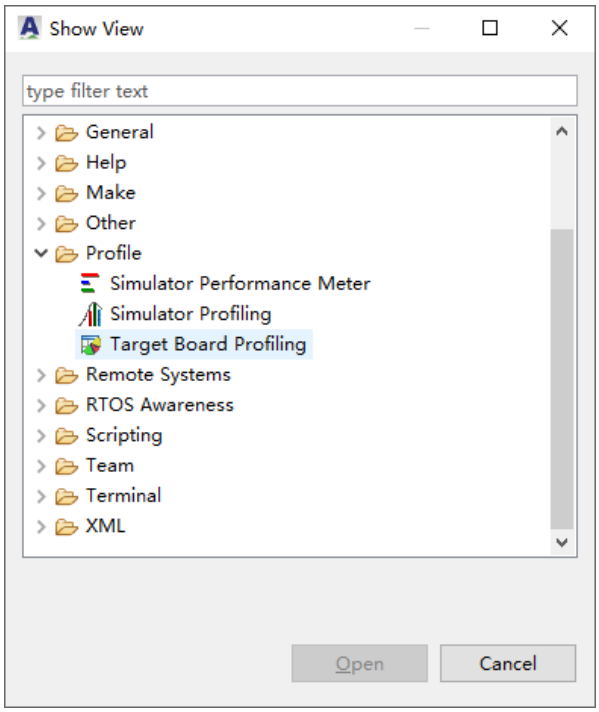


图 6-5 选择 Profile > Target Board Profiling



目标板分析视图的显示列如表 6-1 所示。

表 6-1 目标板分析视图显示列

显示列	描述
Name (location)	被分析的源文件及其指令的列表
Samples	指令的采样计数，指令调用每 10 毫秒采样一次
%Time	特定指令的执行时间占程序总执行时间的百分比

目标板分析视图的工具栏，包括：

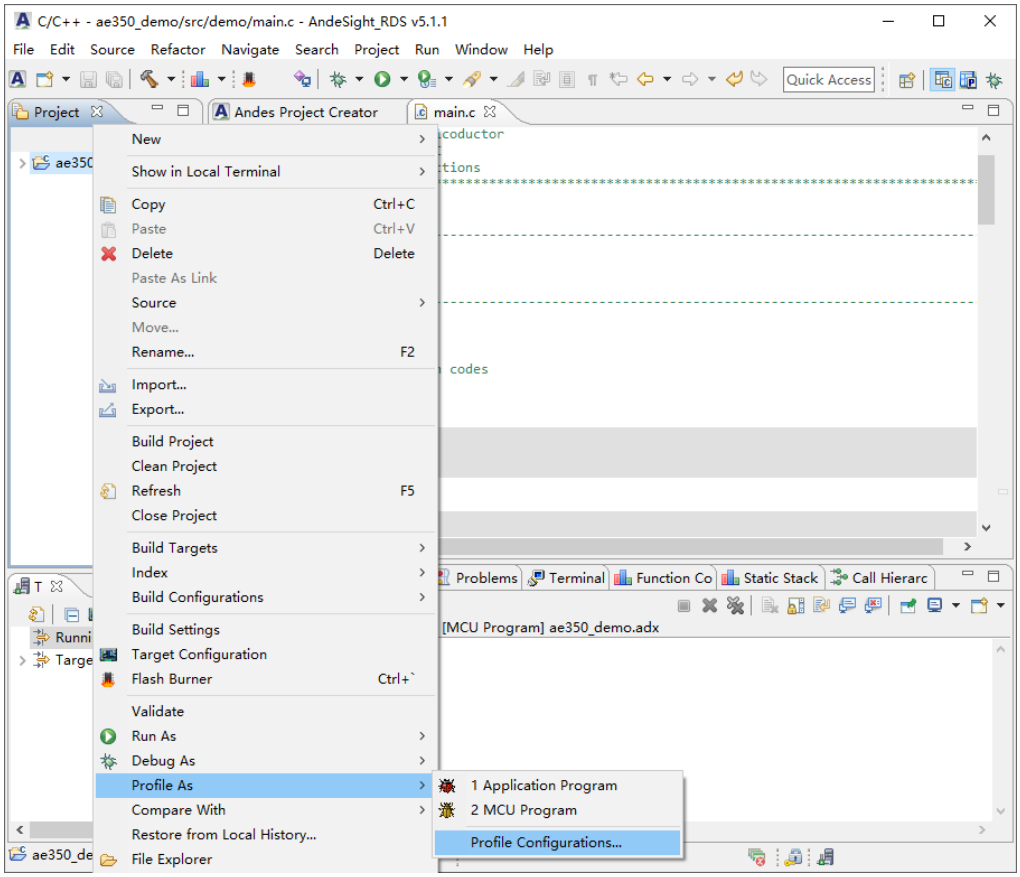
- “”（Show/Hide columns）：“Show/Hide columns”对话框中，指定要在目标板分析视图中显示的列。
- “”（Export to CSV）：引出分析数据到.csv 文件。
- “”（Sorting）：“Sorting”对话框中，指定目标板分析视图中分析数据的排列次序。
- “”（Sort samples per file）：按文件对分析数据排序。
- “”（Sort samples per function）：按功能对分析数据排序。
- “”（Sort samples per line）：按行对分析数据排序。
- “”（Reset）：分析数据设置为 0。
- “”（Switch samples/time）：数据从样本信息切换到时间信息，或反之。
- “”（Create chart…）：使用指定的分析数据，创建条形图、垂直条形图或饼图。

6.3 分析会话

参照以下步骤创建分析会话：

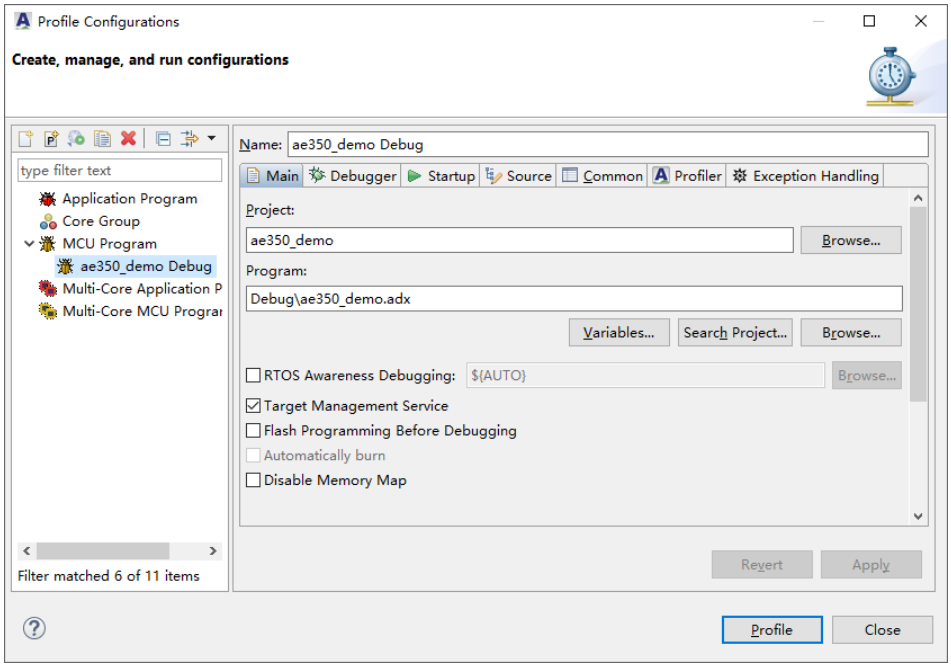
1. 参照第 2 章_创建和构建软件工程创建和构建一个软件工程。
2. 项目资源管理器视图中，右单击选中的软件工程，下拉菜单中选择“Profile As > Profile Configurations…”，如图 6-6 所示。

图 6-6 选择 Profile As > Profile Configurations...



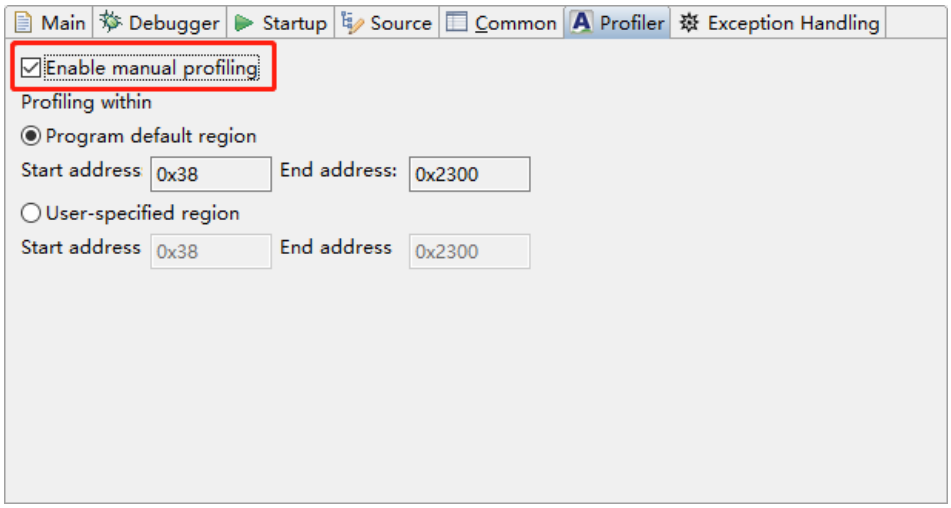
“Profile Configurations”对话框中，双击配置类型“MCU Program”，创建一个分析配置，如图 6-7 所示。

图 6-7 创建分析配置



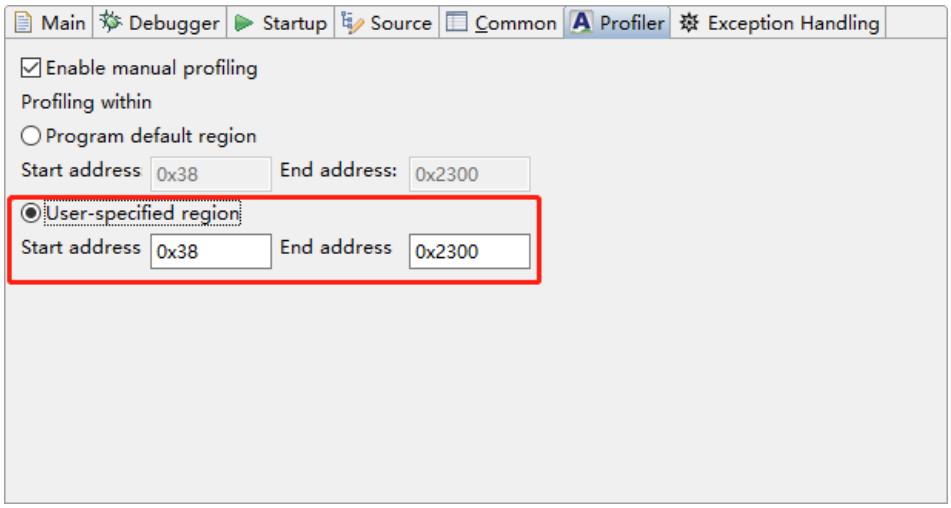
“Profiler”选项卡中，如果要手动开启分析，则选择“Enable manual profiling”，否则关闭，RDS 软件自动开启分析，单击“Profile”，如图 6-8 所示。

图 6-8 选择 Enable manual profiling



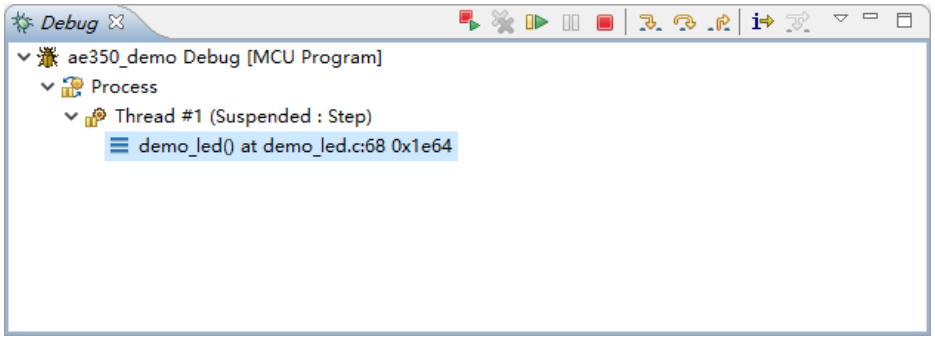
“.text”段是默认被分析的段，如果要更改被分析的区域，选择“User-specified region”，指定要分析区域的起始地址和结束地址，如图 6-9 所示。

图 6-9 选择 User-specified region



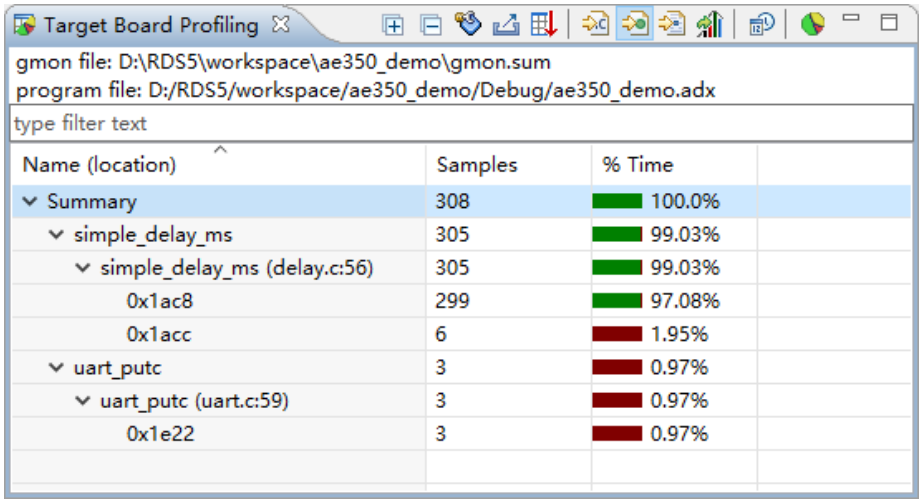
3. 开始分析会话，RDS 软件自动进入分析透视图，单击调试视图工具栏“ ”（Resume）、“ ”（Step Info）或“ ”（Step Over），继续执行程序，如图 6-10 所示。

图 6-10 调试视图



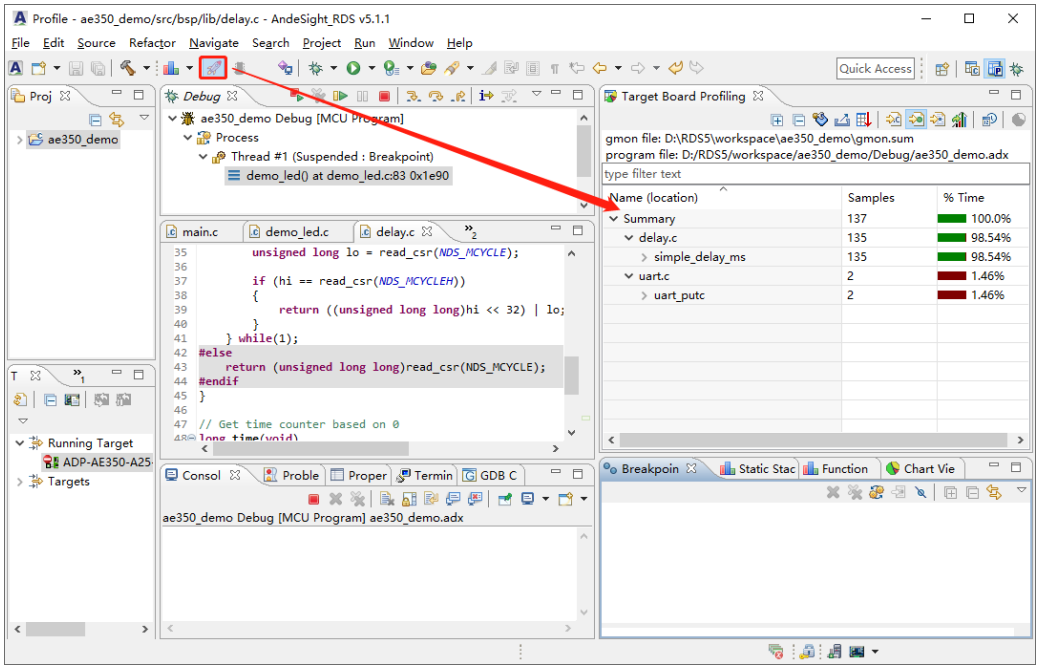
对于没有选定“Enable manual profiling”的分析会话，程序执行暂停或停止后，目标分析视图立即自动产生和显示分析数据，如图 6-11 所示。

图 6-11 目标板分析视图中的分析数据



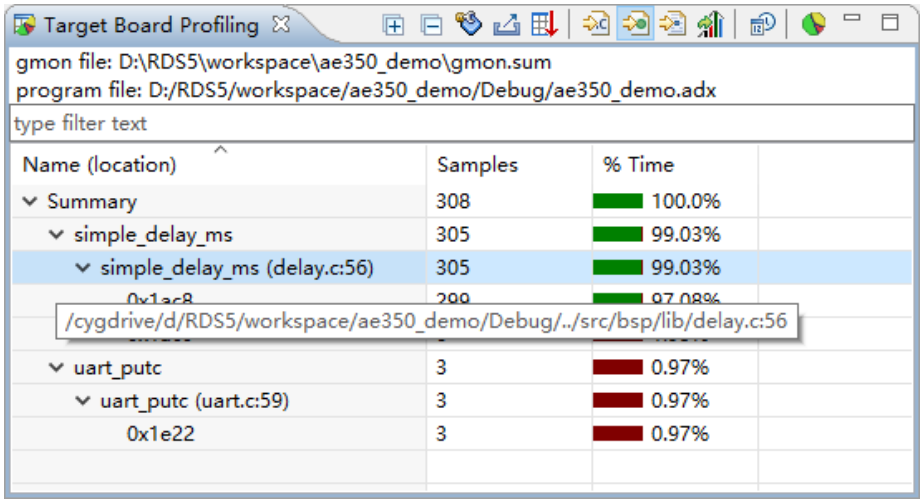
对于选定“Enable manual profiling”的分析会话，参照 4.4.1 设置断点设置断点，单击调试视图工具栏“ ”继续执行程序，程序到达断点处暂停，此时单击 RDS 软件工具栏“ ”（Start Profiling），目标板分析视图立即显示分析数据，如果再次单击“ ”，则停止分析，如图 6-12 所示。

图 6-12 手动启动分析视图



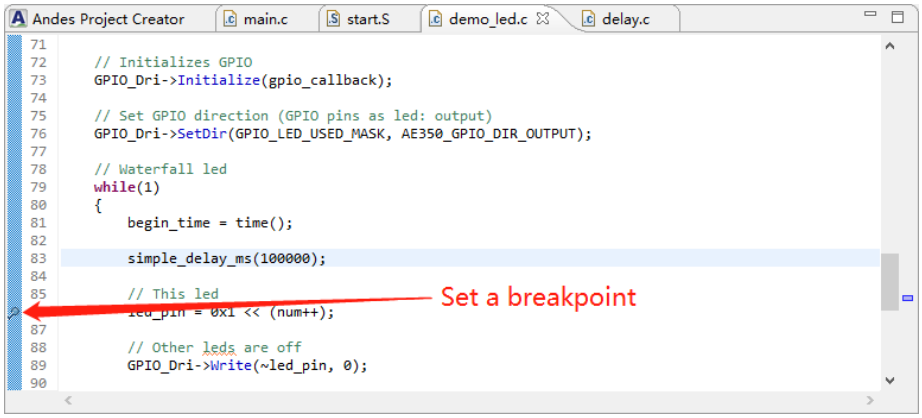
4. 根据分析数据，可以找到程序执行的瓶颈，双击想要查看的函数，直接定位到源码位置，如图 6-13 所示。

图 6-13 定位到源码位置



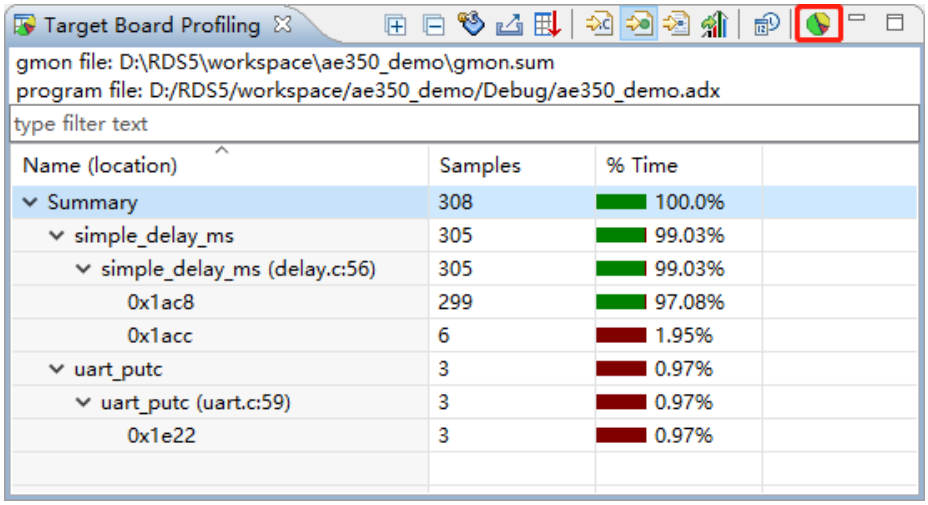
5. 源码中在感兴趣的函数语句处设置断点，如图 6-14 所示。

图 6-14 设置断点



6. 单击 RDS 软件工具栏“ ”（Profile），开始分析会话，程序执行到“main”函数处暂停。
7. 单击调试视图工具栏“ ”（Resume），继续执行程序，到达第一个设置的断点处暂停。
8. 单击调试视图工具栏“ ”（Step Info）或“ ”（Step Over），单步执行感兴趣的函数语句。
9. 目标板分析视图中，再次检查分析数据，参照以下步骤产生图表查看分析数据：
   - 目标板分析视图中，选择想要查看的文件或函数，单击目标分析视图工具栏“ ”（Create chart…），如图 6-15 所示。

图 6-15 选择 Create chart…



   - “Create chart from selection…”对话框中，配置“Select your char type”和“Select the column(s) to show”，单击“OK”，产生图表，如图 6-16 和图 6-17 所示。

图 6-16 Create chart from selection...

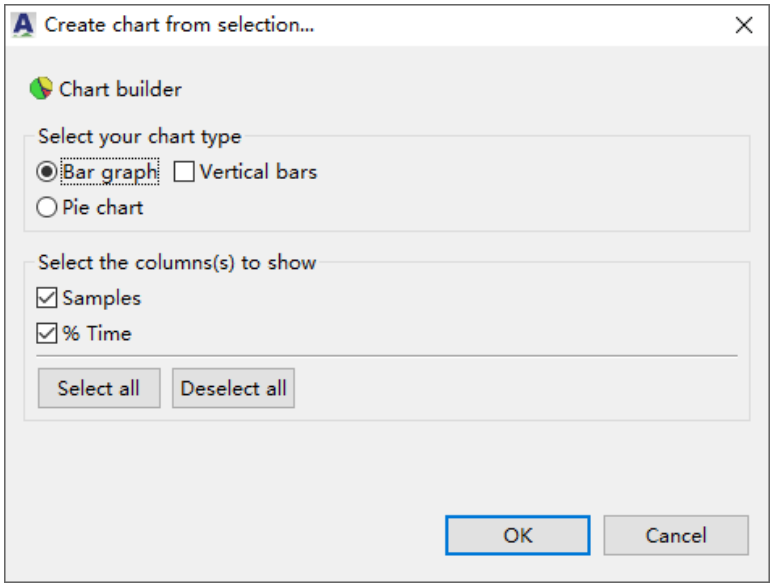
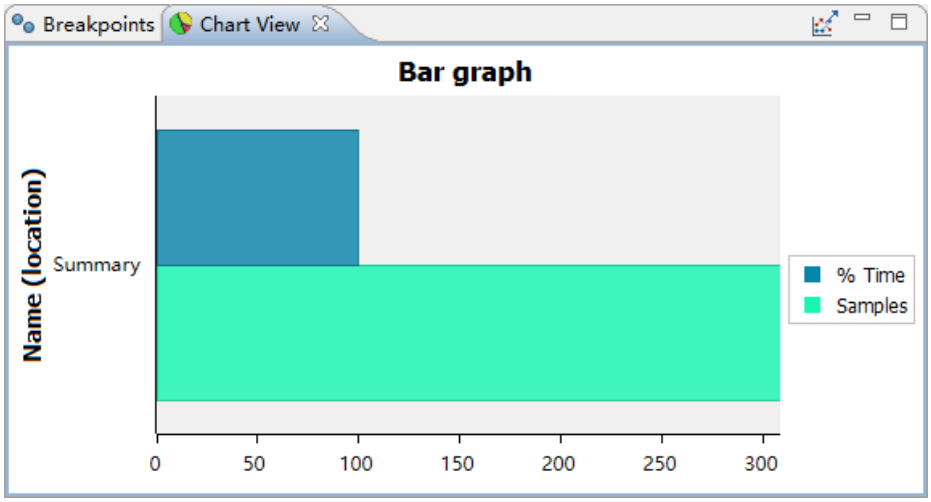


图 6-17 图表视图



10. 如果程序中有汇编代码，单击调试视图工具栏“”（Instruction Stepping Mode），可以切换到反汇编视图，单步执行每条汇编指令，如图 6-18 和图 6-19 所示。

图 6-18 选择 Instruction Stepping Mode

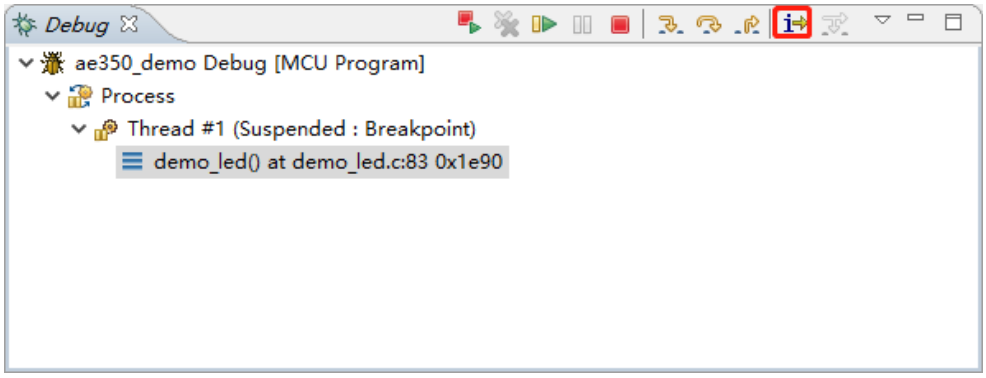


图 6-19 反汇编视图


Console
Problems
Properties
Terminal
GDB Command
Disassembly
Enter location here
00001ac8: addi a0,a0,-1
00001acc: c.bnez a0,0x1ac8 <simple_delay_ms+12>
57 }
00001ace: c.jr ra
00001ad0: c.nop
00001ad2: c.nop
28 if (str)
printchar:
00001ad4: c.beqz a0,0x1ae4 <printchar+16>
30 **str = c;
00001ad6: c.lw a5,0(a0)
00001ad8: sb a1,0(a5)
31 ++(*str);
00001adc: c.lw a5,0(a0)
00001ade: c.addi a5,1
00001ae0: c.sw a5,0(a0)

7 ICEman 软件

ICEman 软件位于“AndeSight_RDS_v511\ice\andes-iceman.bat”，或桌面快捷方式“ICEman_AndeSight_RDS_v5.1.1”，选项描述如表 7-1 所示。

表 7-1 ICEman 选项

选项	描述
-a, --reset-aice	Reset AICE as ICEman startup, for AICE only
-b, --bport	Socket port number for burner connection (default: 2354)
-c, --clock	Specify JTAG clock setting Usage: -c num num should be the following: 0: 30 MHz 1: 15 MHz 2: 7.5 MHz 3: 3.75 MHz 4: 1.875 MHz 5: 909.091 KHz 6: 461.538 KHz 7: 232.558 KHz 10: 10 MHz 11: 6 MHz 12: 3 MHz 13: 1.5 MHz 14: 750 KHz 15: 375 KHz AICE-MINI+ only supports 11 ~ 15
-C, --check-times	Second to check DTM

选项	描述
	(default: 3 seconds) Example: 1. -C 100 to check 100 millisecond 2. -C 100s or -C 100S to check 100 seconds
-D, --larger-logfile	The maximum size of the log file is 1MBx2. The size is increased to 512MBx2 with this option.
-f, --log-output	output log file path
-h, --help	The usage is for ICEman
-H, --reset-hold	Reset-and-hold while ICEman startup
-I, --interface	Specify an interface config file in ice/interface
-l, --custom-srst	Use custom script to do SRST
-L, --custom-trst	Use custom script to do TRST
-N, --custom-restart	Use custom script to do RESET-HOLD
-o, --reset-time	Reset time of reset-and-hold (milliseconds) (default: 1000 milliseconds)
-p, --port	Socket port number for gdb connection
-s, --source	Show commit ID of this version
-S, --stop-seq	Specify the SOC device operation sequence while CPU stop
-R, --resume-seq	Specify the SOC device operation sequence before CPU resume Usage: --stop-seq A1:D1[:M1],A2:D2[:M2] --resume-seq A3:D3[:M3],A2:rst A*:address, D*:data, [:M*]:optional mask, rst:restore A*,D*,[M*] should be hex as following example Example: --stop-seq 0x500000:0x80,0x600000:0x20 --resume-seq 0x500000:0x80,0x600000:rst
-t, --tport	Socket port number for Telnet connection
-T, --boot-time	Boot time of target board (milliseconds) (default: 5000 milliseconds)
-v, --version	Version of ICEman
-x, --diagnosis	Diagnose connectivity issue Usage: --diagnosis[=address]

选项	描述
-X, --uncnd-reset-hold	Unconditional Reset-and-hold while ICEman startup (This implies -H)
-z, --ace-conf	Specify ACE file on each core Usage: --ace-conf <core#id>=<ace_conf>[,<core#id>=<ace_conf>]* Example: --ace-conf core0=core0.aceconf,core1=core1.aceconf
-Z, --target	Specify target type
--target-cfg	Specify the CPU configuration file for a complex multicore
--halt-on-reset	Enable/Disable halt-on-reset functionality


GOWIN高云
智慧逻辑 定制未来