

Interim  
Release

# AndeSight™

## STD v5.0 IDE

## User Manual

Document      UM230-10  
Number

Date Issued      2020-12-31

Copyright © 2020 Andes Technology Corporation  
All rights reserved.



# Copyright Notice

Copyright © 2020 Andes Technology Corporation. All rights reserved.

AndesCore™, AndeSight™, AndeShape™, AndESLive™, AndeSoft™, AndeStar™, Andes Custom Extension™, AndesClarity™, AndeSim™, AndeSysC™, Driving Innovations™, Andes-Embedded™, CoDense™, StackSafe™ and QuickNap™ are trademarks owned by Andes Technology Corporation. All other trademarks used herein are the property of their respective owners.



This document contains confidential information pertaining to Andes Technology Corporation. Use of this copyright notice is precautionary and does not imply publication or disclosure. Neither the whole nor part of the information contained herein may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means without the written permission of Andes Technology Corporation.

The product described herein is subject to continuous development and improvement. Thus, all information herein is provided by Andes in good faith but without warranties. This document is intended only to assist the reader in the use of the product. Andes Technology Corporation shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

## Contact Information

Should you have any problems with the information contained herein, you may contact Andes Technology Corporation through

- email – support@andestech.com
- Website – <https://es.andestech.com/eservice/>

Please include the following information in your inquiries:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem

General suggestions for improvements are welcome.

## Revision History

Rev.	Revision Date	Revised Content
<b>1.0</b>	2020/12/31	Initial Release

A faint watermark in the center of the page reads "Interim Release" in a blue, sans-serif font, enclosed within a thin circular border.

## Table of Contents

<b>COPYRIGHT NOTICE.....</b>	I
<b>CONTACT INFORMATION.....</b>	I
<b>REVISION HISTORY.....</b>	II
<b>TABLE OF CONTENTS.....</b>	III
<b>LIST OF TABLES .....</b>	VII
<b>LIST OF FIGURES .....</b>	VIII
<b>1. INTRODUCTION TO ANDESIGHT™ STD IDE.....</b>	1
1.1. ANDESIGHT IDE OVERVIEW.....	3
1.2. ANDES TOOLCHAIN OVERVIEW .....	9
1.3. STARTING ANDESIGHT IDE.....	11
1.4. ANDESIGHT UI MODES .....	12
1.5. VERSION INFORMATION OF ANDESIGHT COMPONENTS .....	18
1.6. WORKSPACE .....	20
1.7. LANGUAGE OPTIONS IN ANDESIGHT.....	21
1.8. GETTING STARTED WITH ANDESIGHT .....	24
1.9. HOTKEYS FOR ANDESIGHT OPERATIONS.....	30
1.10. REFERENCE DOCUMENTATION AND DOCUMENTS WINDOW .....	33
1.11. FUTURE UPDATES OR PATCHES .....	38
<b>2. ANDESIGHT STD IDE .....</b>	44
2.1. CREATING AND BUILDING ANDESIGHT PROJECTS.....	44
2.1.1. <i>Project creation.....</i>	44
2.1.2. <i>Project files.....</i>	58
2.1.3. <i>Target configuration for projects.....</i>	73
2.1.4. <i>Managed build system for Andes projects.....</i>	78
2.1.5. <i>Building a project.....</i>	96
2.1.6. <i>Static analysis.....</i>	99
2.1.7. <i>Creating and building a shared/static library project.....</i>	106
2.1.8. <i>Creating and building a project that uses shared/static libraries.....</i>	109
2.1.9. <i>Project migration.....</i>	113
2.1.10. <i>Project template.....</i>	116
2.1.11. <i>Restoring a project to its default settings.....</i>	120
2.2. TARGET MANAGEMENT .....	122
2.2.1. <i>Chip profiles for defining targets.....</i>	126

2.2.2.	<i>Setting up the target system</i>	157
2.2.3.	<i>Terminal connection to ICE targets or Linux applications</i>	162
2.2.4.	<i>Target management default settings</i>	166
2.2.5.	<i>Target Manager</i>	177
2.2.6.	<i>Resetting a target board or an ICE</i>	182
2.2.7.	<i>Shutting down a target</i>	184
2.2.8.	<i>Diagnosing your target board or ICE</i>	185
2.3.	<b>FLASH PROGRAMMING</b>	187
2.3.1.	<i>Flash programming using a flash burner</i>	188
2.3.2.	<i>Flash programming using a flash burner and a target application</i>	191
2.3.3.	<i>Changing the flash burner or target_burn application in your chip profile</i>	208
2.3.4.	<i>Flash programming UI in AndeSight</i>	210
2.3.5.	<i>Replacing Andes flash burner(s) and AndeSight flash programming UI</i>	216
2.4.	<b>DEBUGGING/RUNNING PROGRAMS</b>	219
2.4.1.	<i>Debug perspective</i>	219
2.4.2.	<i>AndeSight debug views</i>	221
2.4.3.	<i>Engaging a run/debug session</i>	260
2.4.4.	<i>Breakpoints and watchpoints</i>	270
2.4.5.	<i>Application Program debugging</i>	280
2.4.6.	<i>Target Monitor</i>	283
2.4.7.	<i>MCU Program debugging</i>	284
2.4.8.	<i>Linux Application debugging</i>	289
2.4.9.	<i>Attach to Process</i>	306
2.4.10.	<i>Launch Group</i>	311
2.4.11.	<i>Core Group</i>	314
2.4.12.	<i>RTOS awareness debugging</i>	319
2.4.13.	<i>Linux kernel debugging</i>	345
2.5.	<b>GENERAL EXCEPTION HANDLING AND STACK RECORDING/PROTECTION</b>	354
2.5.1.	<i>Stack recording and stack protection</i>	354
2.5.2.	<i>General exception handling</i>	357
2.6.	<b>PROFILING</b>	361
2.6.1.	<i>Profile perspective</i>	362
2.6.2.	<i>Profile views</i>	364
2.6.3.	<i>Engaging a profile session</i>	373
2.7.	<b>CODE COVERAGE</b>	381
2.7.1.	<i>Code coverage views</i>	381
2.7.2.	<i>Engaging a coverage session</i>	385
2.8.	<b>DEVELOPMENT ON A MULTICORE SYSTEM WITHOUT SMP SUPPORT</b>	388

<i>2.8.1. Creating a multi-core chip profile</i> .....	388
<i>2.8.2. Creating a multi-core project</i> .....	392
<i>2.8.3. Configuring build settings and building the project</i> .....	393
<i>2.8.4. Launching a debug session for the multi-core project</i> .....	396
<b>2.9. TRACING RTOS APPLICATIONS.....</b>	<b>404</b>
<i>2.9.1. RTOS trace views</i> .....	405
<i>2.9.2. Tracing an FreeRTOS application</i> .....	417
<b>3. DEMO PROJECTS.....</b>	<b>425</b>
<i>3.1. JPEG DECODER AND MP3 DEMO</i> .....	426
<i>3.1.1. Importing a demo project</i> .....	426
<i>3.1.2. Specifying build configuration and target configuration</i> .....	429
<i>3.1.3. Running the demo project</i> .....	433
<i>3.2. STARTUP DEMO PROGRAMS.....</i>	<i>436</i>
<i>3.2.1. Debugging a startup demo project</i> .....	438
<i>3.2.2. Packages of V5 startup demo programs</i> .....	462
<i>3.2.3. Description and execution result of each V5 startup demo program</i> .....	469
<i>3.3. DHRYSTONE, COREMARK AND WHETSTONE .....</i>	<i>487</i>
<i>3.3.1. Running Dhrystone-V5, CoreMark-V5 or Whetstone-V5 demos</i> .....	488
<i>3.3.2. Replacing Dhrystone with other benchmarks</i> .....	509
<i>3.4. AMP (ASYMMETRIC MULTIPROCESSING) DEMO PROGRAM .....</i>	<i>510</i>
<i>3.4.1. Importing the AMP demo project</i> .....	510
<i>3.4.2. Specifying the target configuration and building the AMP project</i> .....	513
<i>3.4.3. Debugging the AMP project</i> .....	516
<i>3.5. VECTOR DEMOS .....</i>	<i>520</i>
<i>3.5.1. Importing a vector demo project</i> .....	521
<i>3.5.2. Specifying the target configuration and building the project</i> .....	523
<i>3.5.3. Running the vector demo project</i> .....	526
<b>4. PROGRAM DEVELOPMENT USING THE GDB COMMAND LINE CONSOLE .....</b>	<b>527</b>
<i>4.1. INVOKING A CONSOLE FOR DEBUGGING USING ANDES TOOLCHAINS .....</i>	<i>528</i>
<i>4.2. DEBUGGING ON ICE TARGETS .....</i>	<i>529</i>
<i>4.2.1. Connecting the debug host to an ICE target</i> .....	529
<i>4.2.2. Compiling source programs</i> .....	530
<i>4.2.3. Debugging with GDB</i> .....	530
<i>4.2.4. Typical debugging commands</i> .....	531
<i>4.3. DEBUGGING ON SIMULATOR TARGETS .....</i>	<i>532</i>
<i>4.4. HEADLESS BUILD .....</i>	<i>534</i>
<b>5. TIPS AND TRICKS .....</b>	<b>535</b>

5.1.	COLLECTING DIAGNOSTIC INFORMATION VIA A HOTKEY .....	535
5.2.	DIRECTING CONSOLE OUTPUTS TO TEXT FILES .....	537
5.3.	CHANGING DEFAULT COMPILER/LINKER OPTIONS IN CHIP PROFILES.....	539



## List of Tables

TABLE 1. ANDESIGHT PERSPECTIVES AND VIEWS .....	4
TABLE 2. TOOLCHAINS SUGGESTED FOR DEVELOPMENT WITH ANDES V5 CORES .....	9
TABLE 3. DEFAULT HOTKEYS FOR FREQUENTLY-US ED COMMANDS .....	30
TABLE 4. SUPPORTED CODE MODELS FOR V5 RV32 AND RV64 TOOLCHAINS .....	84
TABLE 5. CHIP PROFILE AND ASSOCIATED FILES USED IN DEFINING A TARGET.....	134
TABLE 6. COLUMNS IN SIMULATOR PERFORMANCE METER VIEW.....	366
TABLE 7. COLUMNS IN THE SIMULATOR PROFILING VIEW .....	368
TABLE 8. COLUMNS IN TARGET BOARD PROFILING VIEW .....	371
TABLE 9. SAG FILES AND LINKER SCRIPTS FOR DEMOS OF VARIOUS CONFIGURATIONS.....	451
TABLE 10. VERIFIED PLATFORMS AND PROJECT SETTINGS FOR DHRYSTONE, COREMARK AND WHETSTONE PROGRAMS.....	487

Initial  
Release

## List of Figures

FIGURE 1. ANDESIGHT STD IDE ARCHITECTURE.....	2
FIGURE 2. C/C++ PERSPECTIVE OF ANDESIGHT IDE .....	3
FIGURE 3. DRAG-AND-DROP INTERFACE OF THE SAG EDITOR.....	63
FIGURE 4. SOURCE CODE EDITING INTERFACE OF THE SAG EDITOR.....	64
FIGURE 5. SELECTING PROJECT TYPE IN VERSIONS PRIOR TO ANDESIGHTV1.4 .....	113
FIGURE 6. PREDEFINED TARGETS IN ANDESIGHT STD v5.0.....	123
FIGURE 7. FLOWCHART OF TARGET MANAGEMENT IN ANDESIGHT .....	125
FIGURE 8. CHIP PROFILE PROPERTIES FILE AND THE CORRESPONDING EDITOR .....	135
FIGURE 9. CPU SETTINGS IN A CHIP PROFILE EDITOR.....	137
FIGURE 10. FLASH DRIVER SETTINGS IN A CHIP PROFILE EDITOR .....	137
FIGURE 11. AX25 CPU DIALOG > CACHE .....	143
FIGURE 12. AX25 CPU DIALOG > MEMORY MANAGEMENT UNIT.....	144
FIGURE 13. AX25 CPU DIALOG > LOCAL MEMORY .....	145
FIGURE 14. AX25 CPU DIALOG > PIPELINE.....	146
FIGURE 15. AX25 CPU DIALOG > MISCELLANEOUS.....	147
FIGURE 16. AX25 CPU DIALOG > ISA.....	149
FIGURE 17. AX25 CPU DIALOG > IMPORT/EXPORT .....	150
FIGURE 18. AX25 CPU DIALOG > ADVANCED.....	151
FIGURE 19. CHIP PROFILE GROUPING IN ANDES PROJECT CREATOR .....	154
FIGURE 20. CHIP PROFILE GROUPING IN TARGET MANAGEMENT DEFAULT SETTINGS .....	154
FIGURE 21. PHYSICAL NETWORK BETWEEN ANDESIGHT STD IDE AND TARGET SYSTEMS.....	157
FIGURE 22. EDITING A CONNECTION CONFIGURATION WITH SIMULATOR CONNECTION TYPE .....	173
FIGURE 23. CREATING A NEW CONNECTION CONFIGURATION WITH REMOTE CONNECTION TYPE .....	174
FIGURE 24. IN-SYSTEM PROGRAMMING VIA PAR_BURN OR SPI_BURN.....	188
FIGURE 25. IN-SYSTEM PROGRAMMING BY TARGET_BURN_FRONTEND AND TARGET_BURN .....	191
FIGURE 26. FLASH BURNER SETTINGS IN A TARGETBOARD.PROPERTIES FILE .....	208
FIGURE 27. FLASH BURNER SETTINGS IN CHIP PROFILE EDITOR.....	209
FIGURE 28. FLASH PROGRAMMING WIZARD FOR PROJECTS USING TARGET_BURN ON V5 ICE TARGETS .....	213
FIGURE 29. FLASH PROGRAMMING WIZARD FOR PROJECTS ON SIMULATOR TARGETS.....	215
FIGURE 30. DEBUG PERSPECTIVE.....	219
FIGURE 31. PROFILE PERSPECTIVE OF ANDESIGHT IDE.....	362
FIGURE 32. XIP MODE: BOOT-N-RUN ON FLASH .....	467
FIGURE 33. BURN MODE: BOOT IN FLASH, THEN LOAD TO MEM FOR EXECUTION .....	468

## Typographical Convention Index

Document Element	Font	Font Style	Size	Color
Normal text	Georgia	Normal	12	Black
Command line, source code or file paths	Lucida Console	Normal	11	Indigo
VARIABLES OR PARAMETERS IN COMMAND LINE, SOURCE CODE OR FILE PATHS	LUCIDA CONSOLE	<b>BOLD + ALL-CAPS</b>	11	INDIGO
<a href="#">Hyperlink</a>	Georgia	<u>Underlined</u>	12	Blue

## 1. Introduction to AndeSight™ STD IDE

AndeSight v5.0 is a complete development solution that provides required components and highly-optimized tools for software development. To address the need of different users, it includes two install packages, IDE (Integrated Development Environment) and BSP (Board Support Packages). The IDE package encloses software components in a feature-rich AndeSight IDE that allows programmers to develop with GUI (graphical user interface); the BSP is dedicated for command line users, providing development components and utilities with no GUI. You can determine a desired install package during the AndeSight installation process.

This user manual focuses on the AndeSight IDE package and introduces the uses and features of AndeSight Standard (STD) v5.0 IDE. An integrated development suite based on Eclipse (<http://www.eclipse.org/>), the AndeSight IDE contains toolchains and components like compiler, simulator, flash burner, and ICE (In Circuit Emulator) driver (i.e. ICEman) that makes it possible to run, edit, debug, and profile source programs on target systems including simulator and ICE targets. A simulator target is a virtual evaluation platform connected to the debugging host via an SoC simulator specified by a VEP configuration file, whereas an ICE target is a real evaluation platform (such as AndeShape™) connected to the host through an ICE debugger.

The figure below illustrates the major components and functionalities of the AndeSight STD IDE.

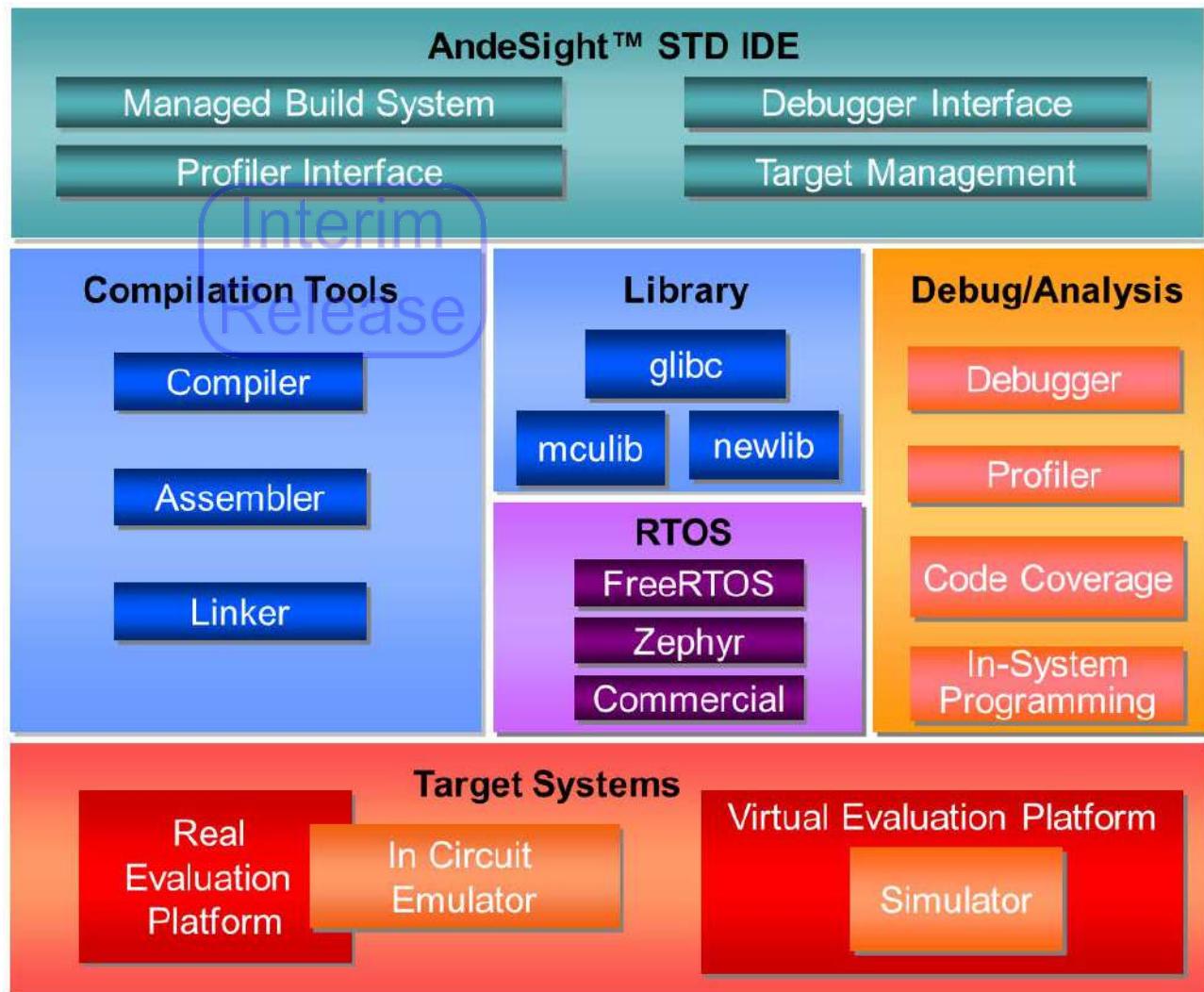


Figure 1. AndeSight STD IDE architecture

**NOTE**

The information in this document, including descriptions and illustrations, is provided by Andes in good faith for developers' reference and subject to continuous development and improvement without notice.

## 1.1. AndeSight IDE overview

The work area of the AndeSight IDE consists of individual panes called “views”, which are grouped together to form functionally-oriented “perspectives”.

For instance, the C/C++ perspective in the figure below contains the **Project Explorer** view, the code editor, and the **Console** view, allowing you to conveniently browse and locate project files, edit code, and read standard input/output results.

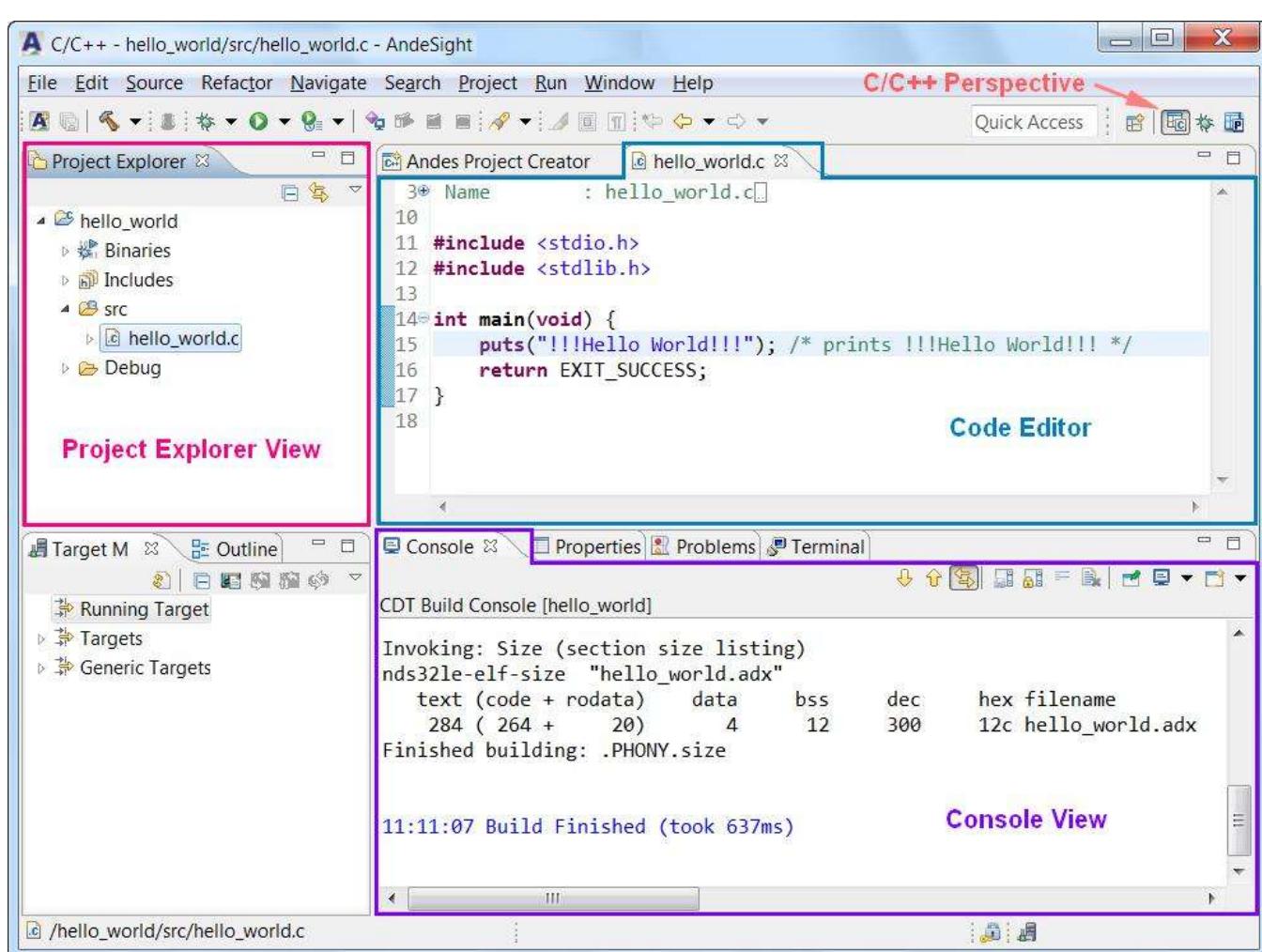


Figure 2. C/C++ perspective of AndeSight IDE

The **Debug** perspective and **Profile** perspective are also predefined in AndeSight. The table below summarizes the views that are most frequently used from these perspectives as well as their functions.

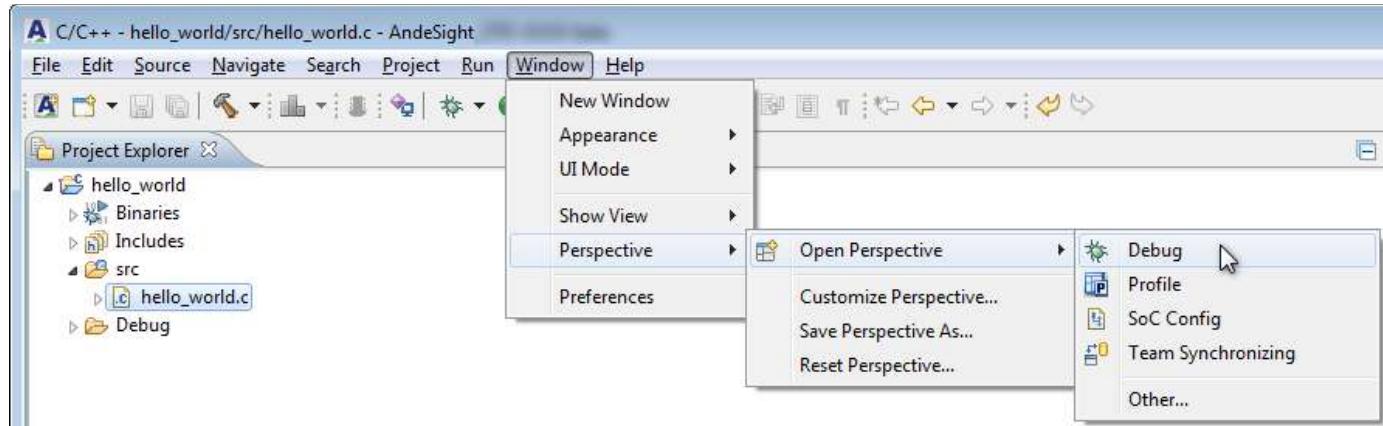
Table 1. AndeSight perspectives and views

<b>View</b>	<b>Description</b>	<b>Perspectives</b>		
		<b>C/C++</b>	<b>Debug</b>	<b>Profile</b>
Breakpoints	Summarizes the breakpoints set		▪	▪
Cache Dump	Displays information for CPU caches after program execution suspends		▪	▪
Console	Displays standard input/output and errors	▪	▪	▪
Debug	Reveals the target system in a tree list and allows you to control the debug session		▪	▪
Disassembly	Displays assembly of functions using source information		▪	▪
Documents Window	Displays reference materials associated with your development and allows you to manage them.	▪	▪	▪
Expression	Displays information related to global variables as well as truncation of the original code aimed at achieving a particular result		▪	
Event	Displays collected events from a trace session of a RTOS application and shows the details of each event occurrence.		▪	▪
Function Code Size	Displays code size information related to each function after a project is built	▪		
Gcov	Displays code coverage data		▪	▪
GDB Command	Serves as a GDB command interface		▪	
Global Variables Live	Displays values of selected global variables at runtime		▪	
Memory	Allows you to inspect and alter the process memory		▪	▪

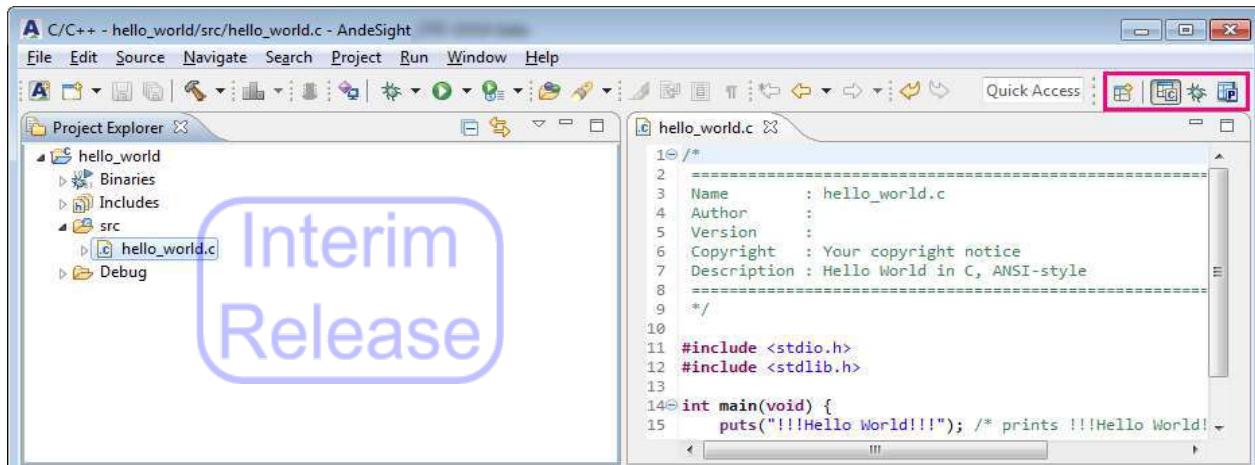
<b>View</b>	<b>Description</b>	<b>Perspectives</b>		
		<b>C/C++</b>	<b>Debug</b>	<b>Profile</b>
Memory Map	Displays the allocation of memory space		▪	
Outline	Displays an outline of the file currently open in the editor in a list of structural elements	▪	▪	▪
Power Monitor	Displays the real-time power usage of a program in a timeline		▪	▪
Problems	Displays all errors encountered while building a project	▪		
Project Explore	Displays projects in the workspace along with their files. You may launch an executable here.	▪	▪	▪
Properties	Displays property names and values of a selected item	▪		
Registers	Lists register information in a selected stack frame		▪	▪
Simulator Performance Meter	Displays profiling data with respect to each step of program execution for simulator targets			▪
Simulator Profiling	Displays statistics related to all executed functions for the program being profiled using simulator targets			▪
SoC Registers	Displays SoC register values for you to read and write		▪	
Static Stack Analysis	Displays sizes of static stacks for built projects	▪		
Statistics	Shows statistical information of contexts in a trace session of a RTOS application.		▪	▪

<b>View</b>	<b>Description</b>	<b>Perspectives</b>		
		<b>C/C++</b>	<b>Debug</b>	<b>Profile</b>
Target Board Profiling	Displays profiling data for the program being profiled using ICE targets			▪
Target Manager	Lists available target systems and the status of each	▪	▪	▪
Tasks	Lists scheduling of processes		▪	▪
Timeline	Depicts the sequence of contexts and CPU transitions in a timeline for a RTOS application.		▪	▪
Variables	Displays information related to local variables. You may edit, enable, or disable the values for each variable.		▪	▪

You may also switch between perspectives throughout the development process as follows:

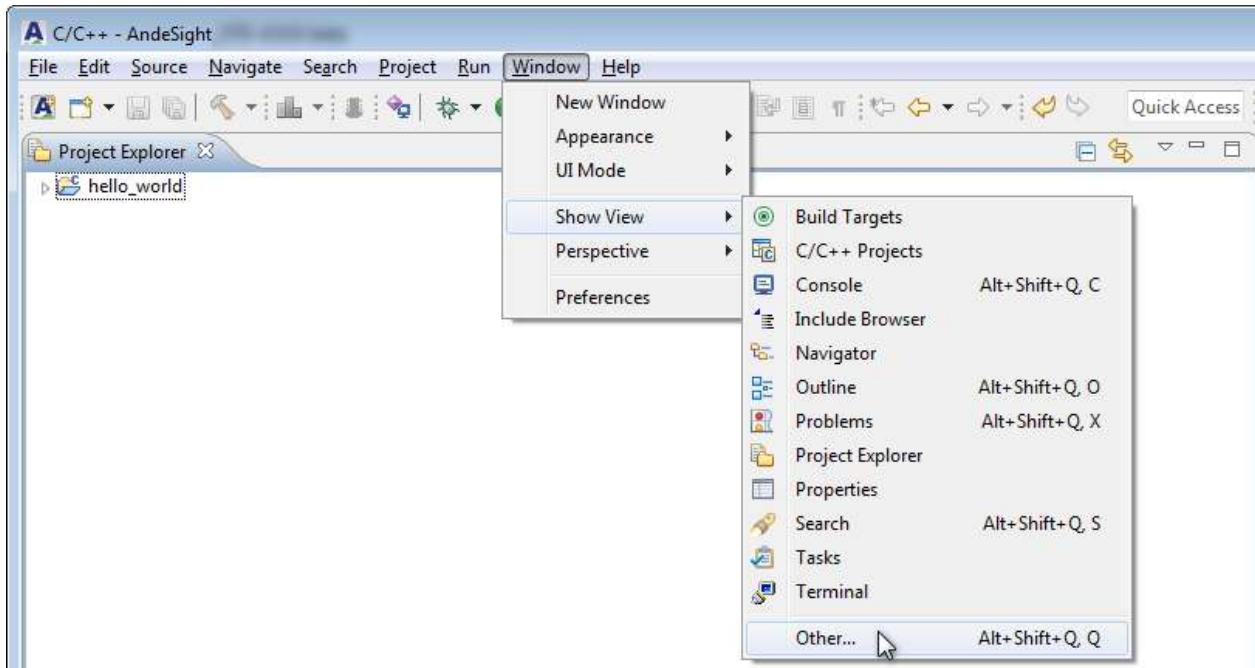


(main menu > Window > Perspective > Open Perspective)



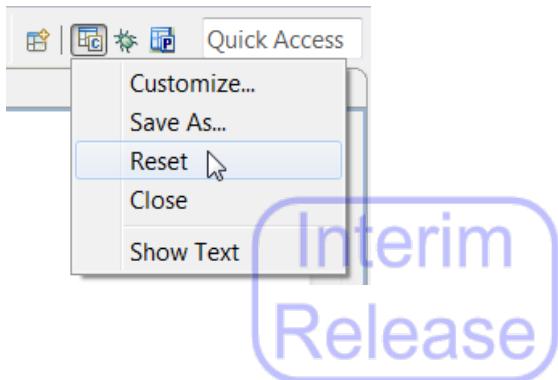
(perspective toolbar)

It is also possible to open views that are not included in the current perspective, as shown below.



(main menu &gt; Window &gt; Show View)

AndeSight supports drag-and-drop functionality to modify the perspective layout. You can drag desired views and drop them into new locations to form different tab groups. You can also rearrange the order of views within a perspective for operational convenience, or tile views side-by-side to enable comparisons. To reset a perspective layout, simply right-click the desired perspective button on the perspective toolbar and select “Reset” in the pull-down menu.



## 1.2. Andes toolchain overview

Andes toolchains are grouped according to the implementation version of the Andes instruction set architecture (ISA). The naming of Andes toolchains indicates the supported CPU version (32-bit or 64-bit), the endianness (“be” = big endian; “le” = little endian), library support (“elf” = newlib/mculib; “linux” = glibc), and the version of the Andes ISA implementation (v5, v5f, v5d, v5e). Andes ISA V5, V5F, V5D and V5E correspond respectively to RISC-V extensions IMAC, IMAFC, IMAFDC and EMAC.

Table 2 summarizes the toolchains in the AndeSight STD suggested for development with Andes V5 cores. Note that in the AndeSight IDE, the term “toolchain” may appear as one word or as separate words, as in **Tool Chain Editor** in the **Target Configuration** dialog.

**Table 2. Toolchains suggested for development with Andes V5 cores**

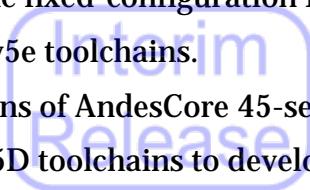
Andes V5 Cores Andes Toolchains	AX45/ NX45	AX27	AX25/ AX25MP	NX27V	NX25F	A45/ N45/ D45	A25/ A25MP /A27	N25F <sup>[1]</sup> /D25F	N22 <sup>[2]</sup>
nds32le-elf-[newlib mculib]-v5						■	■	■	■
nds32le-elf-[newlib mculib]-v5e									■
nds32le-elf-[newlib mculib]-v5d						■ <sup>[3]</sup>	■	■	
nds32le-elf-[newlib mculib]-v5f						■ <sup>[3]</sup>	■	■	
nds32le-linux-glibc-v5d						■	■	■	
nds64le-elf-[newlib mculib]-v5	■	■	■	■	■				
nds64le-elf-[newlib mculib]-v5d	■ <sup>[3]</sup>	■	■	■	■				
nds64le-elf-[newlib mculib]-v5f	■ <sup>[3]</sup>	■	■	■	■				
nds64le-linux-glibc-v5d	■	■	■		■				

### NOTE

1. N25 in the chip profile ADP-AE250-N25 is an Andes core derived from N25F. N25 has the same configuration as N25F except that it comes without FPU support. The suggested

toolchains for development with N25 are nds32le-elf-[newlib|mcilib]-v5.

2. Andes FreeStart program provides two types of N22 RTL, fully-configurable N22 and fixed configuration N22. While the fully-configurable N22 can be developed using v5 or v5e toolchains, the fixed-configuration N22 is compliant to RISC-V EMAC extension and restricted to v5e toolchains.
3. Earlier versions of AndesCore 45-series CPU cores do not have FPU support. If you want to use V5F or V5D toolchains to develop with these cores, make sure to have a FPGA board that supports floating point operations first.

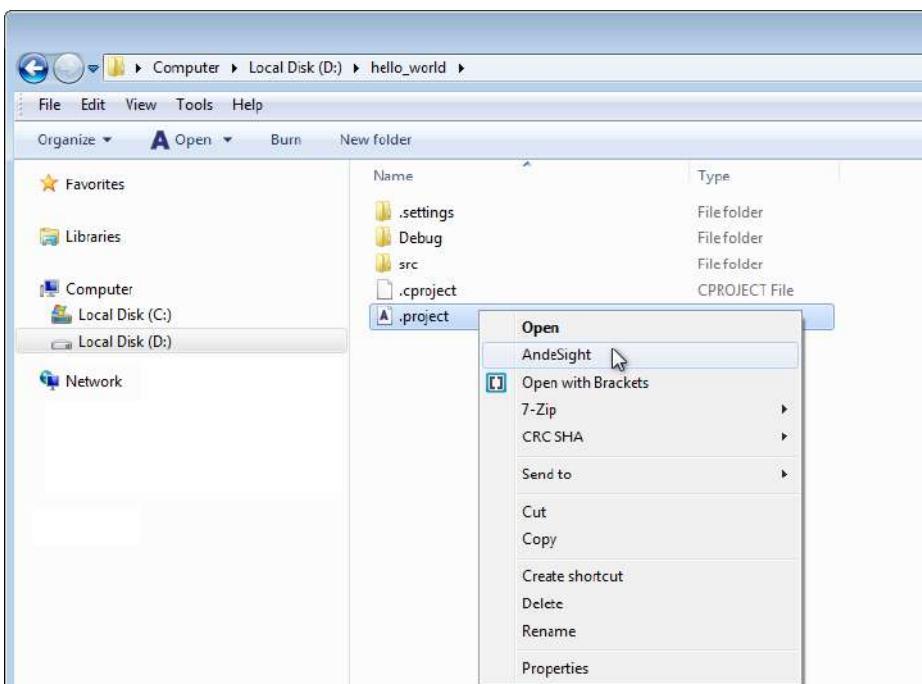


### 1.3. Starting AndeSight IDE

The AndeSight IDE on a local system can be launched by double-clicking the desktop shortcut

 or the execution file `AndeSight.exe` under `ANDESIGHT_ROOT\ide`. If you have administrator access to a multi-user environment, launch the AndeSight IDE by running the script `admin` under `ANDESIGHT_ROOT\ide\advanced` with root privileges. This can ensure that the additions, updates or modifications on AndeSight components can be shared with other users on the same system.

You can also start the AndeSight IDE with an existing AndeSight project. Just double-click a file with the filename extension `.project` under the project or right-click it to select “AndeSight” from the pull-down menu, as shown below.



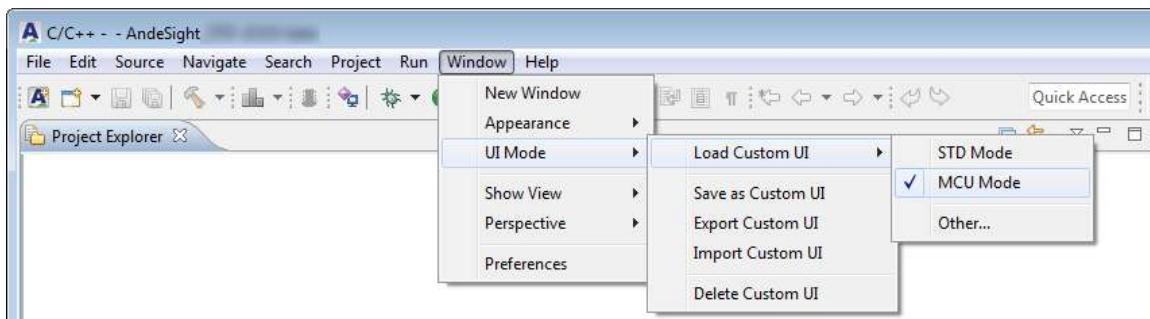
The AndeSight IDE will then be launched with the project opened in the **Project Explorer** view. The project, if not located in your current workspace, will be imported at the point. In the case that this file association feature does not function properly, you can enable it manually by executing `[win7|win10]_project_file_association.bat` or `linux_project_file_association.sh` under `ANDESIGHT_ROOT\ide\advanced\file_association`.

## 1.4. AndeSight UI modes

The AndeSight STD IDE provides two default user interfaces (UI), STD mode and MCU mode, to address different development needs. The MCU mode (the default mode) provides a streamlined UI specialized for developing microcontroller applications whereas the STD mode comprises a full-featured layout to fulfill all development requirements.



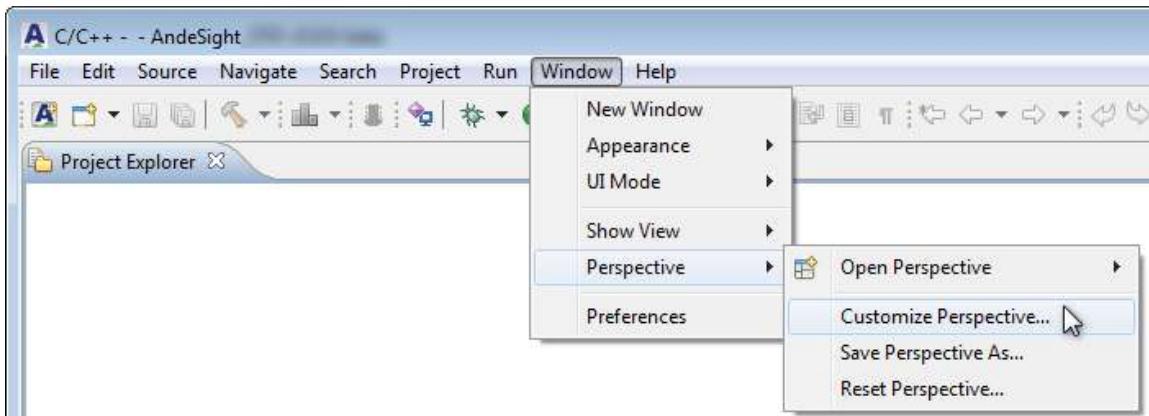
When first launched, the AndeSight IDE is presented in the UI mode you selected during AndeSight installation. To determine in which mode the IDE is displayed or to switch to another mode, click “Window > UI Mode > Load Custom UI” on the AndeSight main menu and make a selection on the menu. In the example below, the AndeSight IDE is presented in MCU mode.

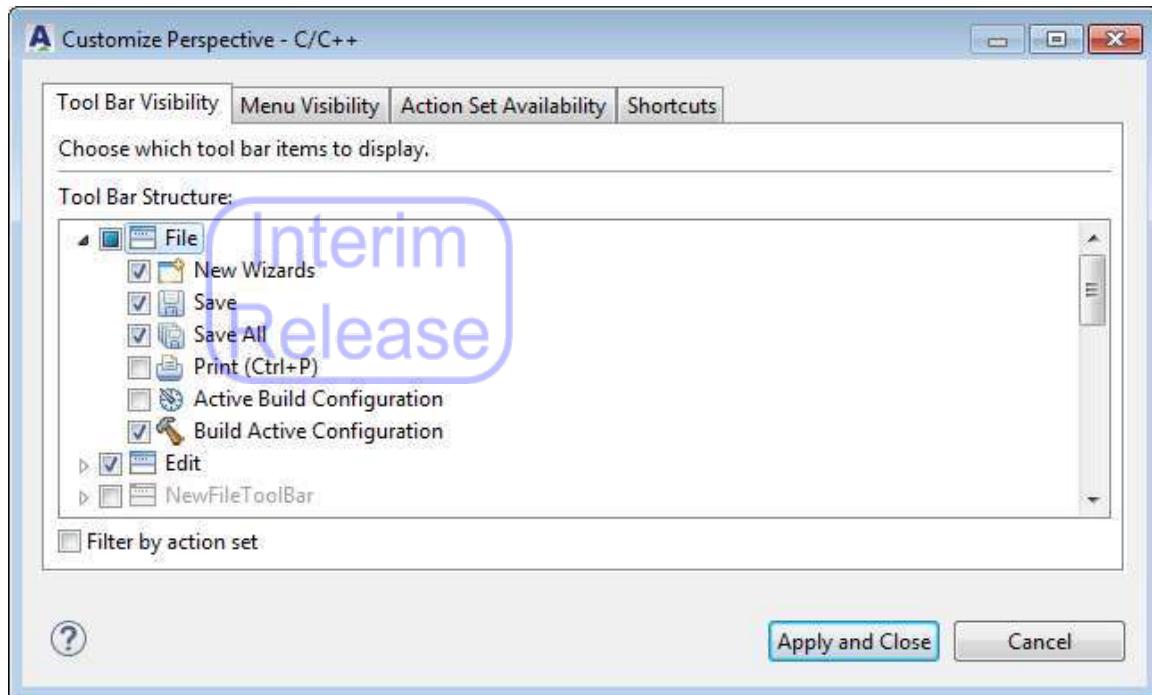


To better meet your development needs, you can further change the appearance of the layout by customizing the following:

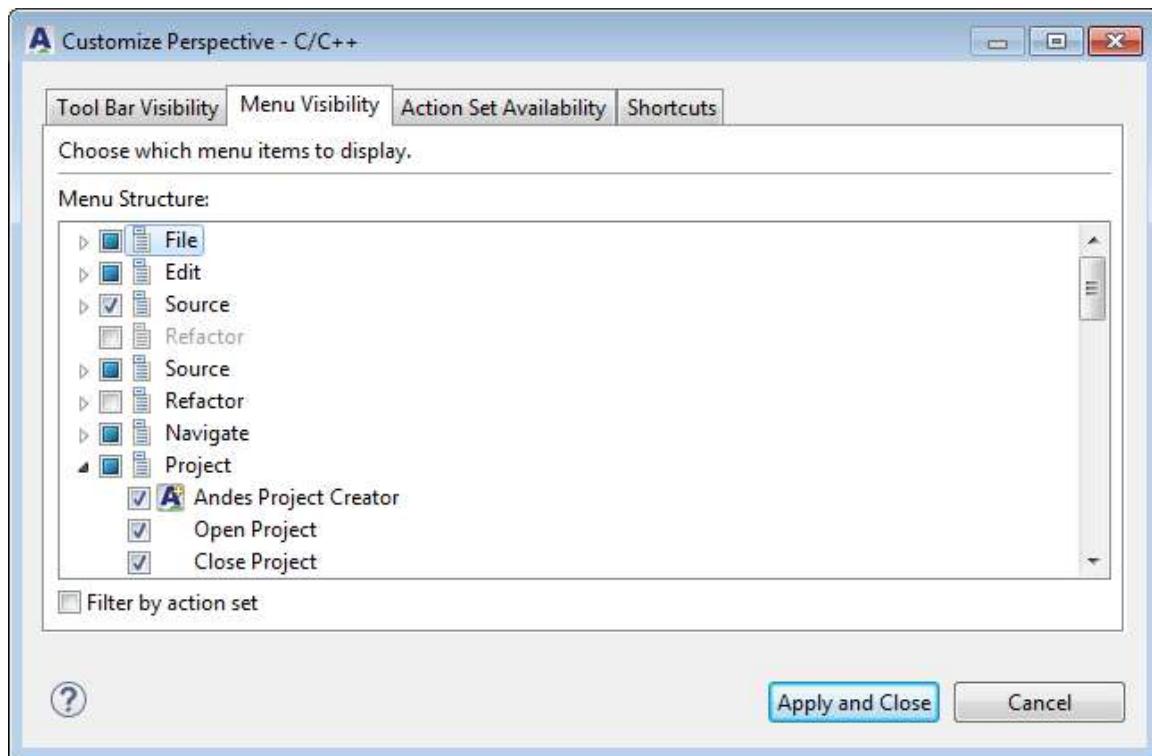
- Toolbar items and menu items in each perspective

Click “Window > Perspective > Customize Perspective...” on the AndeSight main menu to invoke the **Customize Perspective** dialog and make selections under the **Tool Bar Visibility** tab and **Menu Visibility** tab.



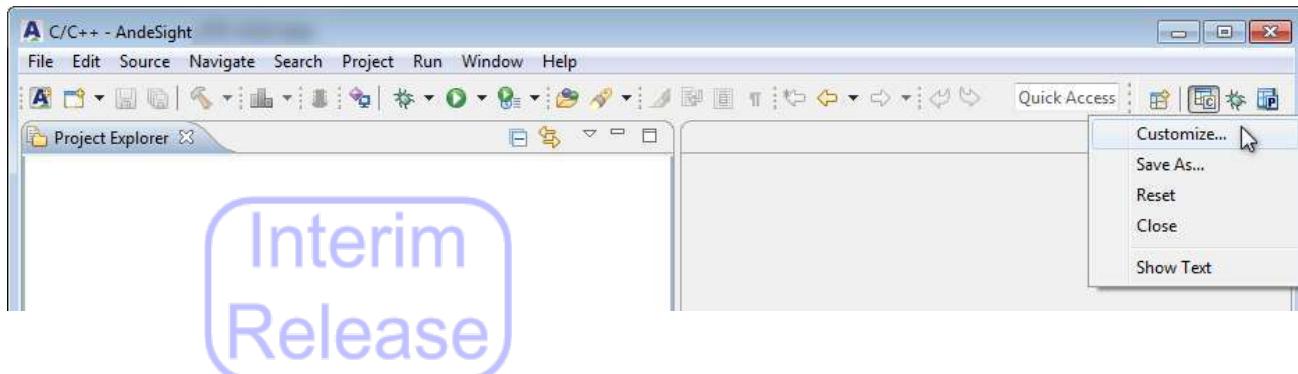


(making selections for toolbar items)



(making selections for main menu items)

The **Customize Perspective** dialog can also be invoked by right-clicking on a perspective button on the AndeSight toolbar and selecting “Customize...” from its pull-down menu.



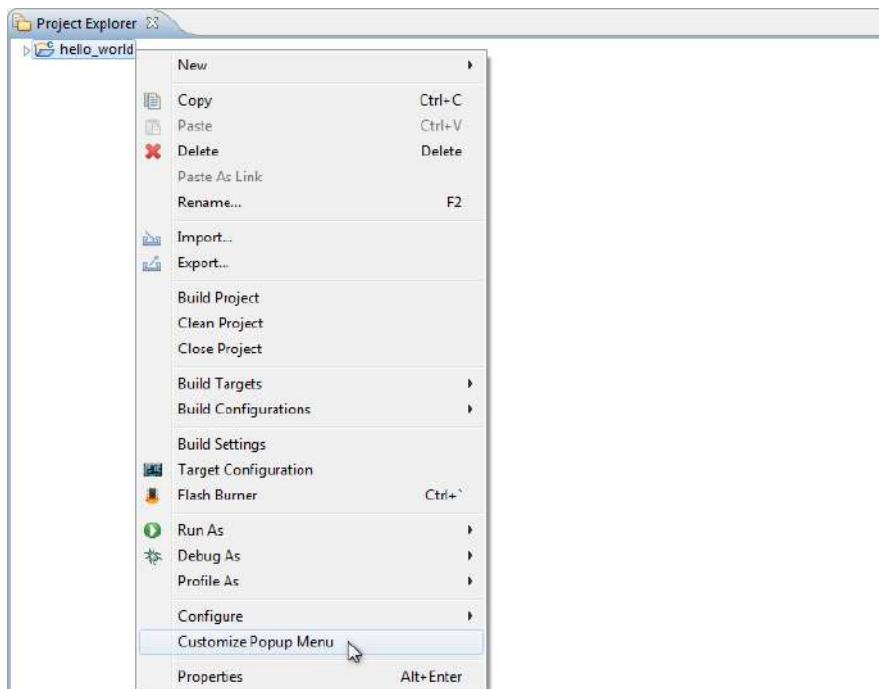
■ Pop-up menu items for **Project Explorer** or resources in the view

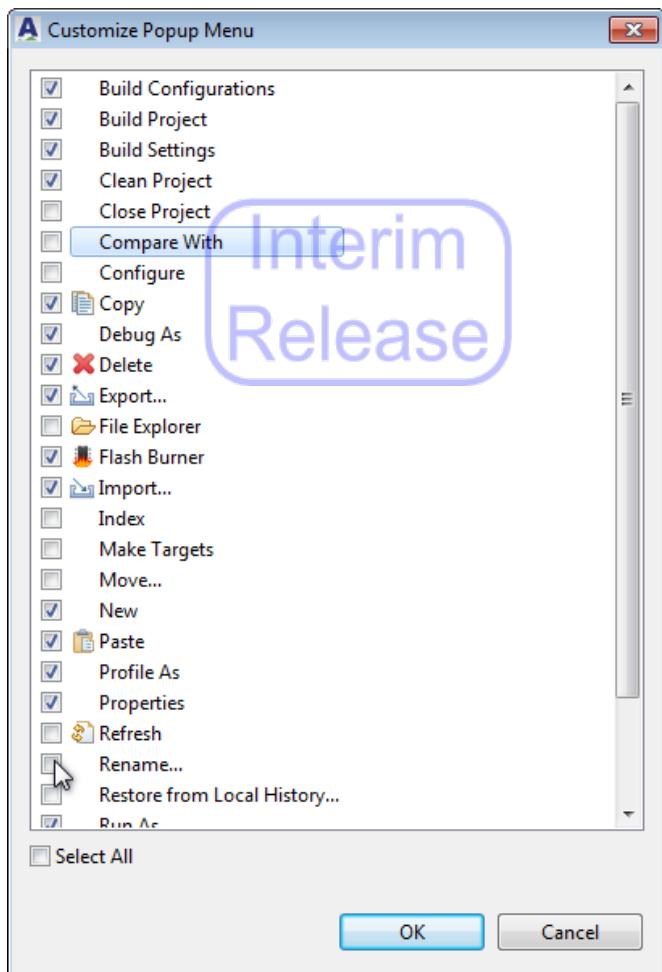
Pop-up menu items for the **Project Explorer** view and for projects, folders or files in the view differ from one another to some extent and can be customized individually.

To customize pop-up menu items for **Project Explorer**, right-click anywhere in the view when it contains no resource and select “Customize Popup Menu” from the pull-down menu to invoke the **Custom Popup Menu** dialog and make selections.

To customize pop-up menu items for projects, folders or files in **Project Explorer**, right-click a desired resource in the view and select “Customize Popup Menu” from the pull-down menu to invoke the **Custom Popup Menu** dialog and make selections.

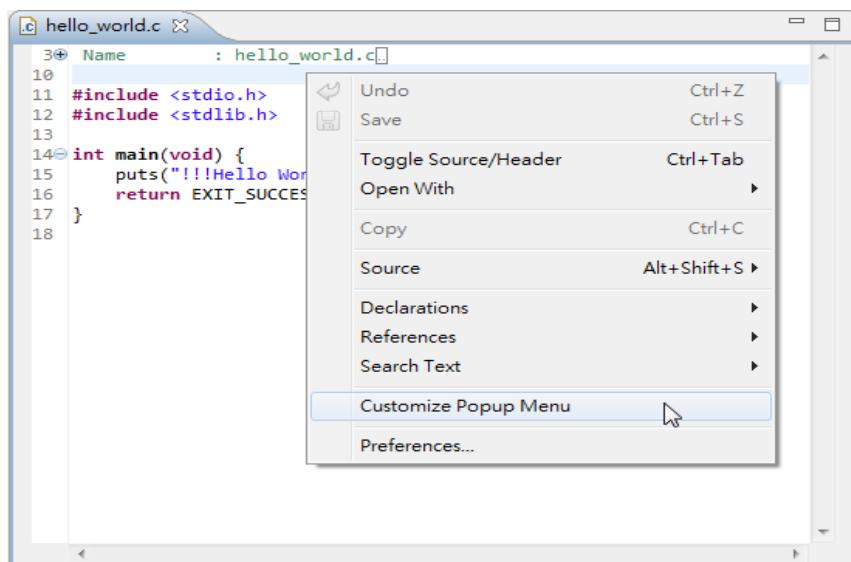
The figures below illustrate the procedure to customize pop-up menu items for projects in **Project Explorer**.

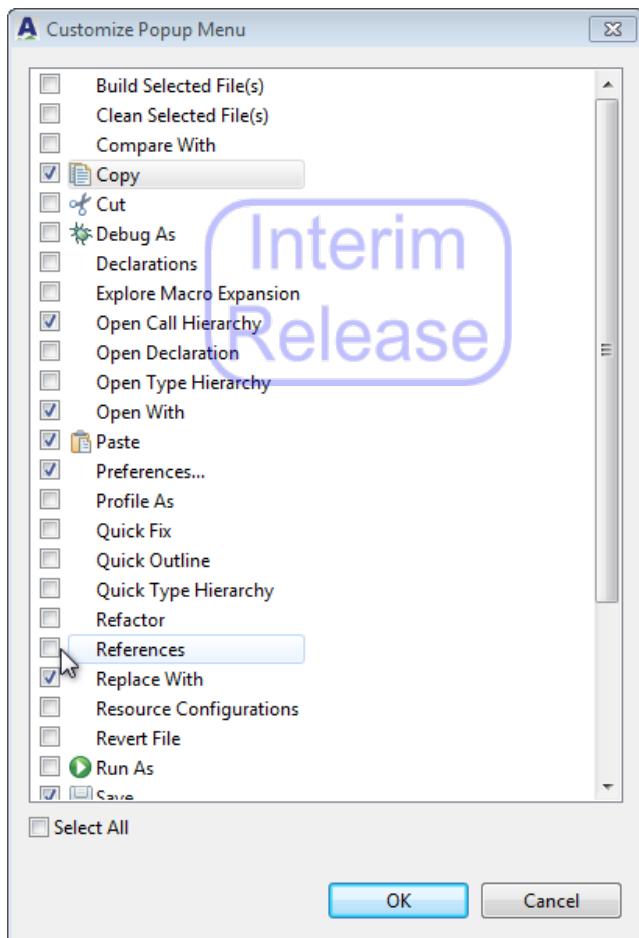




## ■ Pop-up menu items for C/C++ code editors

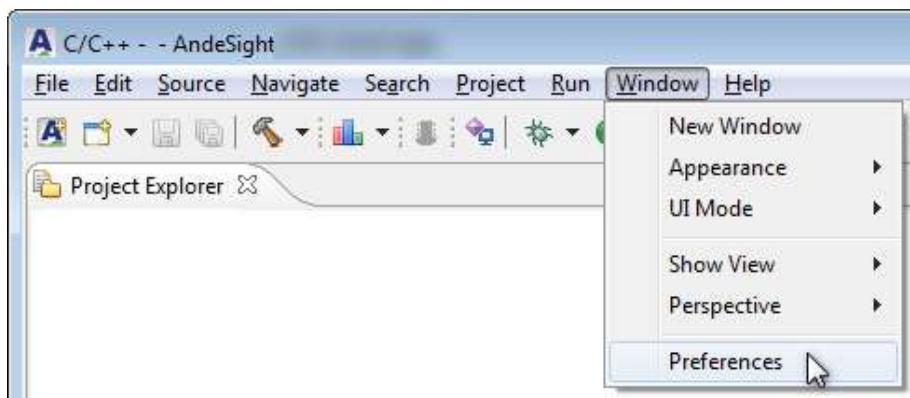
Right-click anywhere in a code editor of C/C++ file and select “Customize Popup Menu” from its pull-down menu to invoke the **Custom Popup Menu** dialog and make selections.

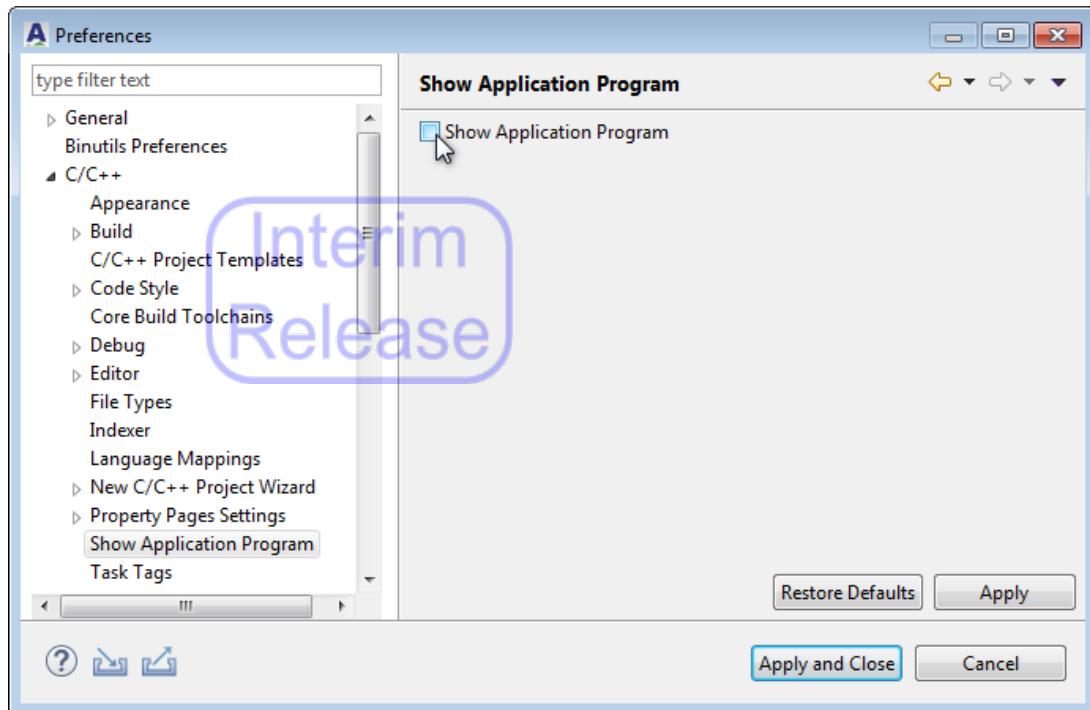




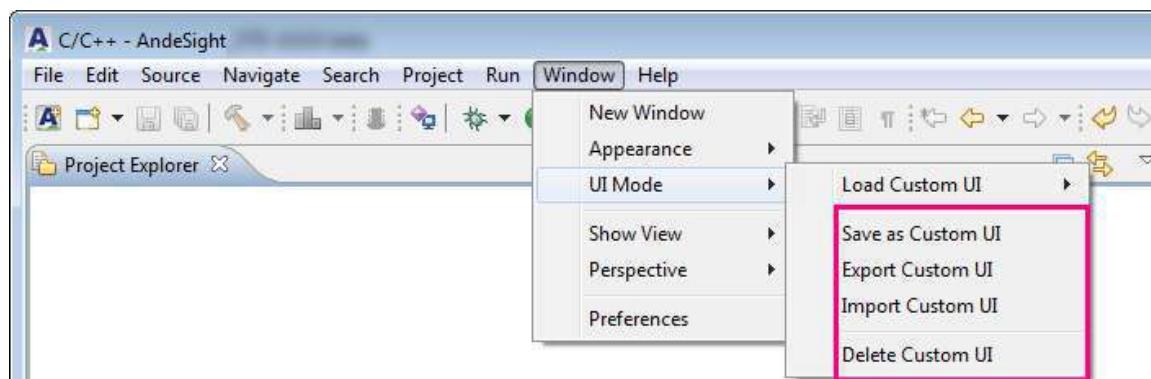
## ■ The **Application Program** debug configuration

Among given debug configurations, the **Application Program** debug configuration is the one you can decide to show or hide. Just click “Window > Preferences” on the AndeSight main menu to invoke the **Preferences** dialog, select “C/C++ > Show Application Program” in the navigation pane and make a selection on the right page.





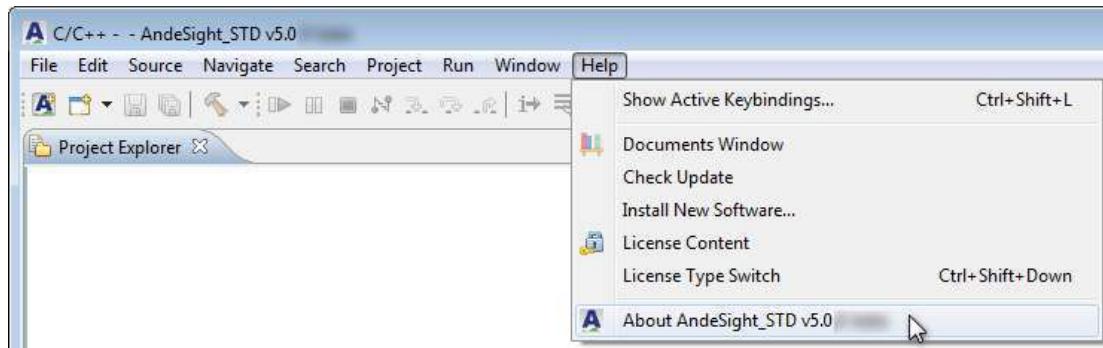
Once you finish the layout configuration, you can save it as a UI mode for future uses or export the mode to share with other AndeSight users. You can also delete or import an existing UI mode. To perform these actions, just click "Window > UI Mode" on the AndeSight main menu and make a selection from the pull-down menu.



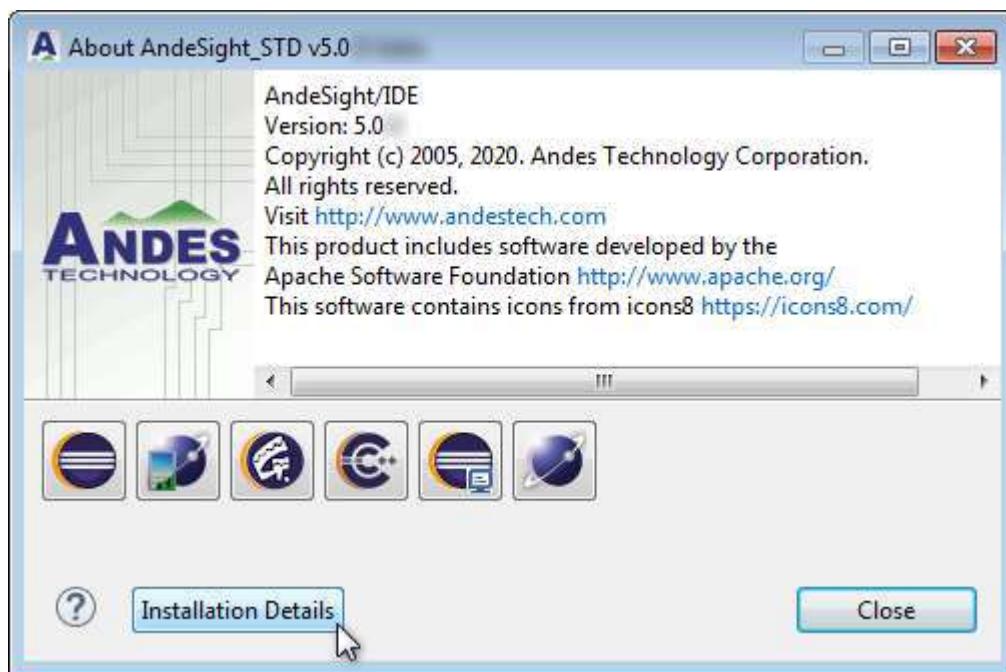
## 1.5. Version information of AndeSight components

You can identify the version of AndeSight you are using as well as the installed software components by following the steps below:

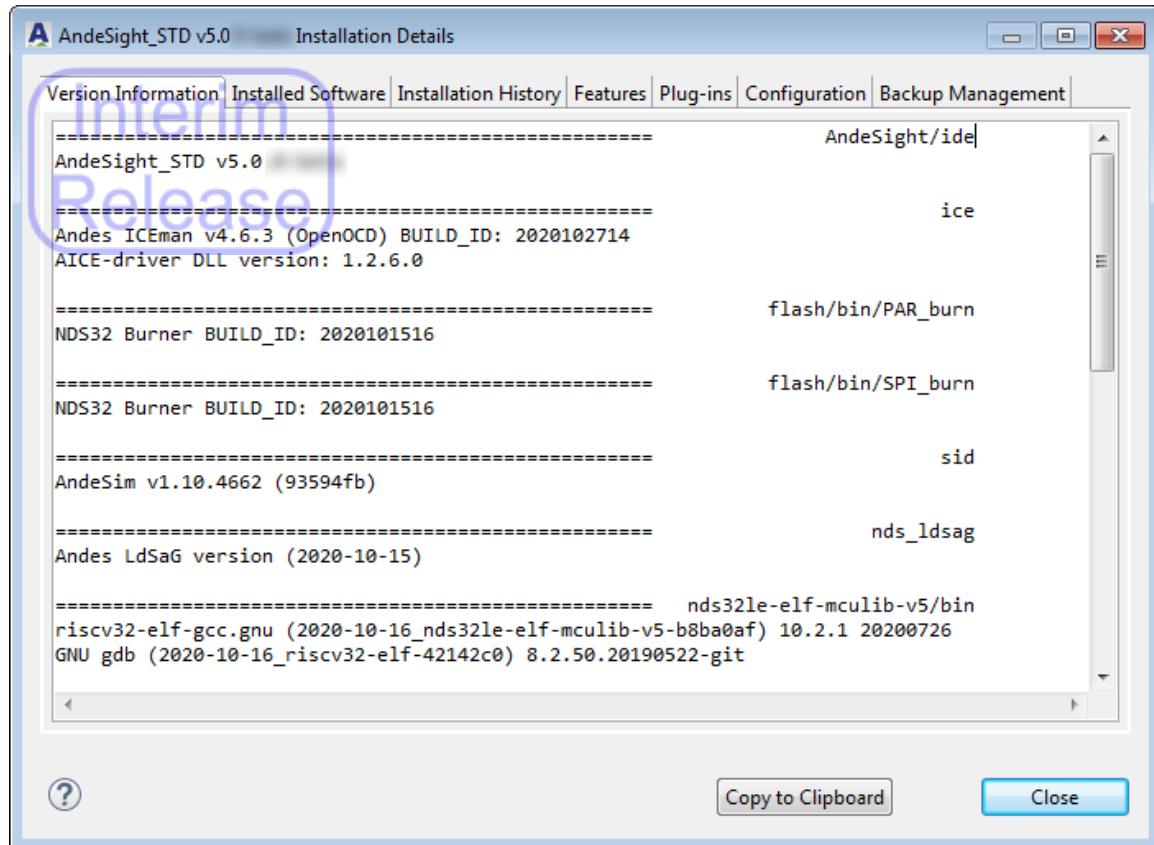
- Step 1** On the AndeSight main menu, select “Help > About AndeSight\_EDITION\_VERSION”.



- Step 2** On the invoked dialog, find the version of the AndeSight IDE currently in use. Click the “Installation Details” button.

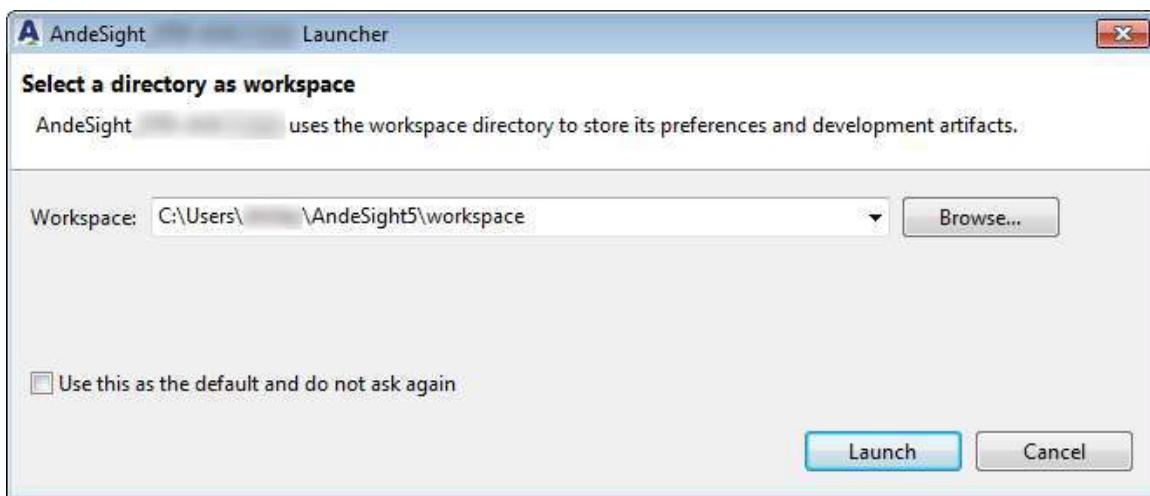


**Step 3** Proceed to the “Version Information” tab to reveal the version information of installed AndeSight components.



## 1.6. Workspace

A workspace is where your AndeSight projects are saved. The default workspace location is **USER\_HOME\AndeSight 5\workspace**. You can find this in the **Workspace Launcher** dialog when AndeSight IDE is launched for the first time. To store your projects in this ready-made workspace, simply click “OK” in the dialog. To create or change a workspace for your projects, click “Browse...” and specify a desired location. Note that the workspace path must only consist of characters (A-Z, a-z, 0-9), underscores (\_), and hyphens (-).

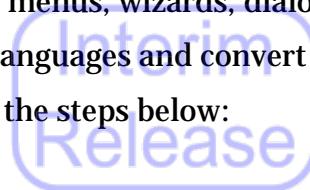


The workspace also stores diagnostic information (see Section 5.1), ICEman logs, and simulator configuration files. Each of these is saved in a different subdirectory, as follows:

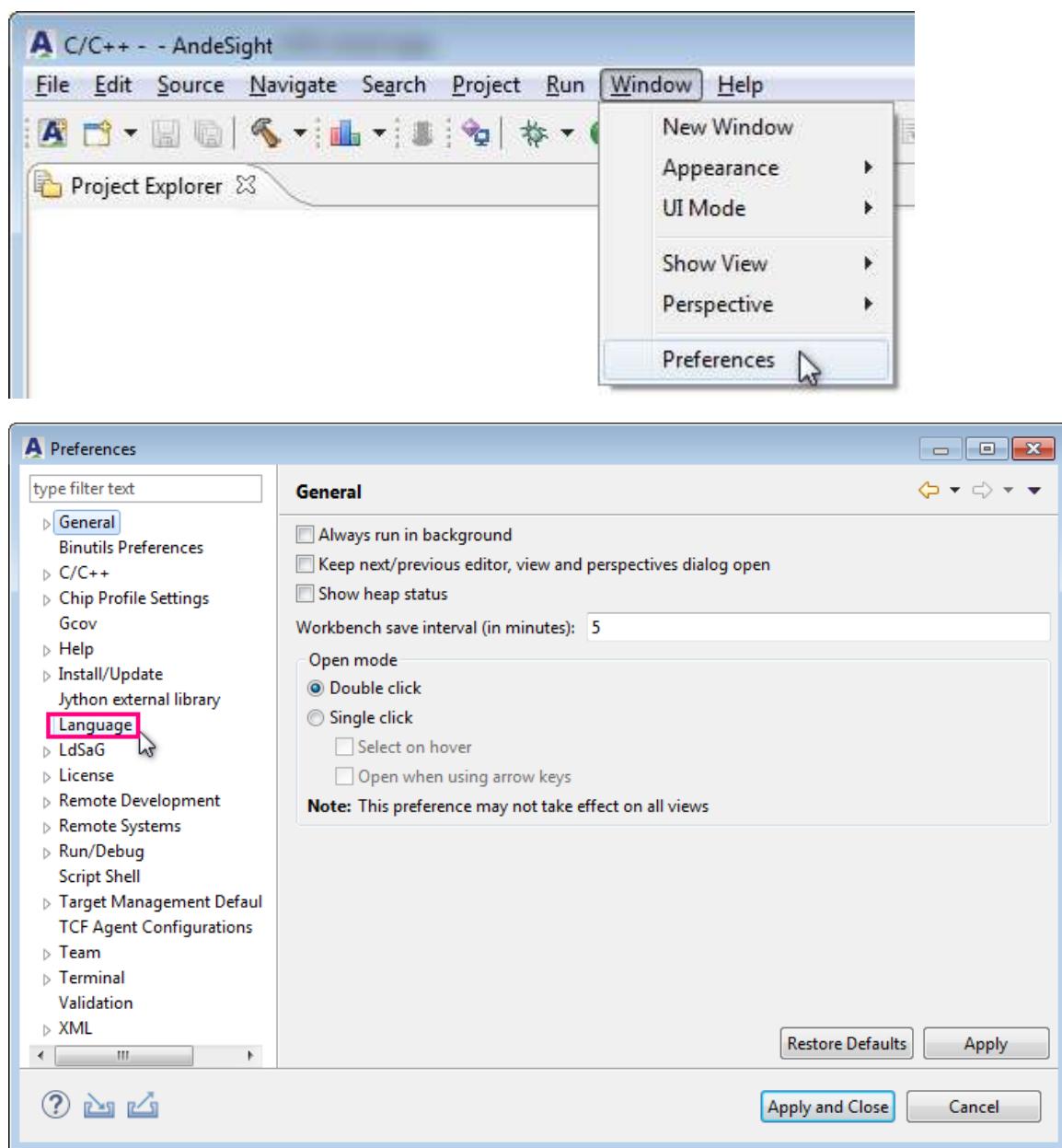
- Diagnostic information: **WORKSPACE\_ROOT\log**
- ICEman logs: **WORKSPACE\_ROOT\ICEman**
- Simulator configuration files (.vep.conf): **WORKSPACE\_ROOT\.metadata\vep.conf**

## 1.7. Language options in AndeSight

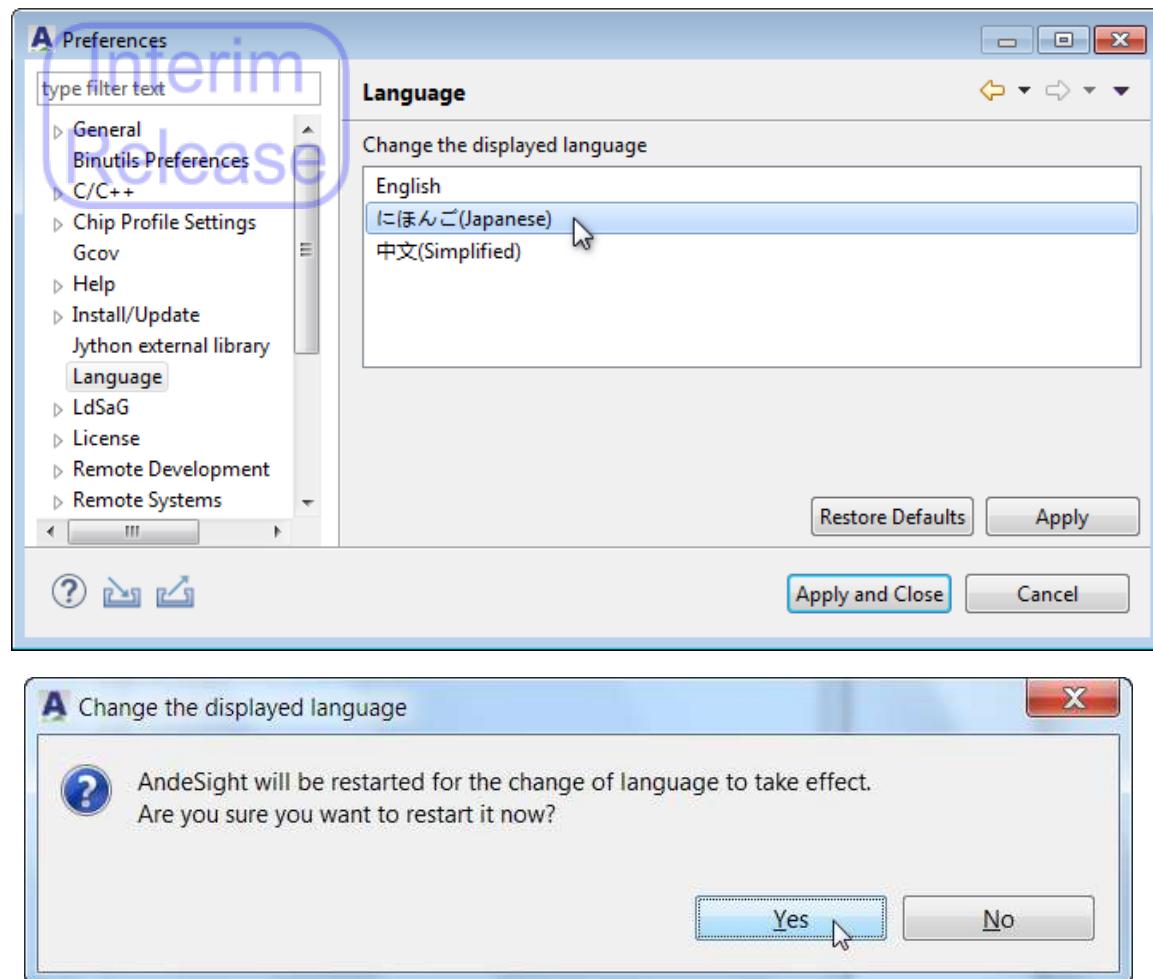
AndeSight has installed three language packs (English, Japanese, and simplified Chinese) for the display of text in menus, wizards, dialog boxes, and other items in the IDE. It is possible to switch between languages and convert any text to the language you select. To change the display language, follow the steps below:



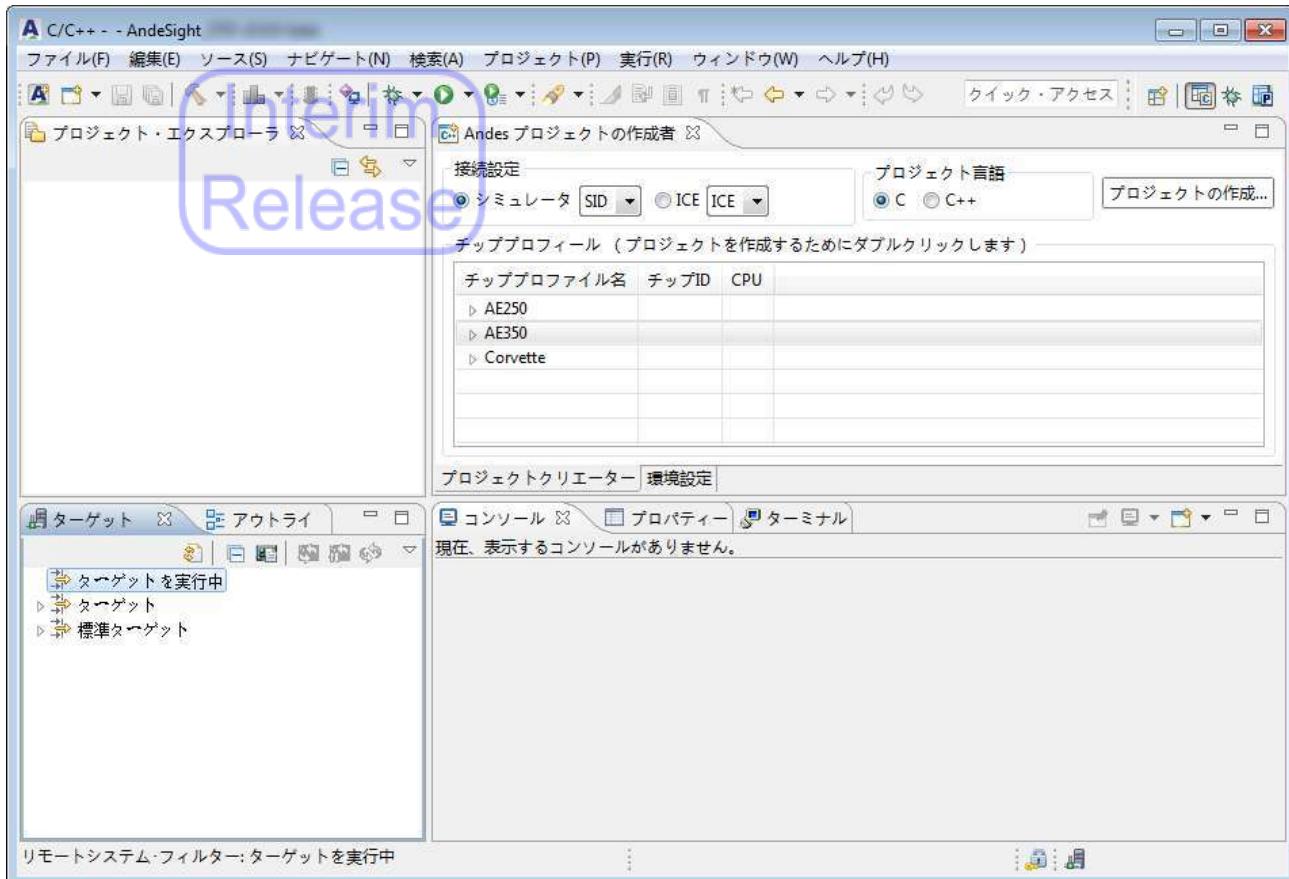
- Step 1** On the AndeSight main menu, select “Window > Preferences” to invoke the **Preferences** dialog and select “Language” in the navigation pane.



**Step 2** Select a language from among the three installed language packs and click “Apply”. At the “Change the displayed language” prompt, click “Yes” to re-start AndeSight.



**Step 3** After re-launching, AndeSight is displayed in the language of your choice.

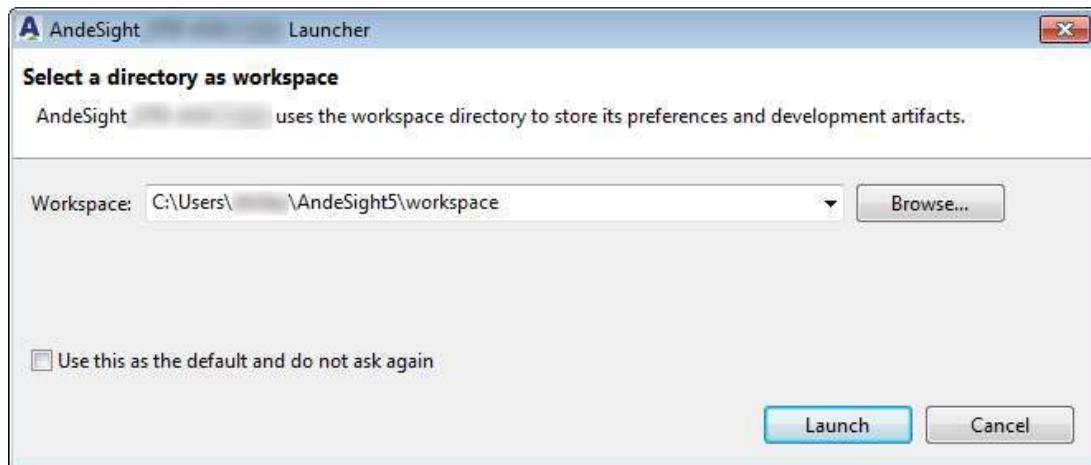


## 1.8. Getting started with AndeSight

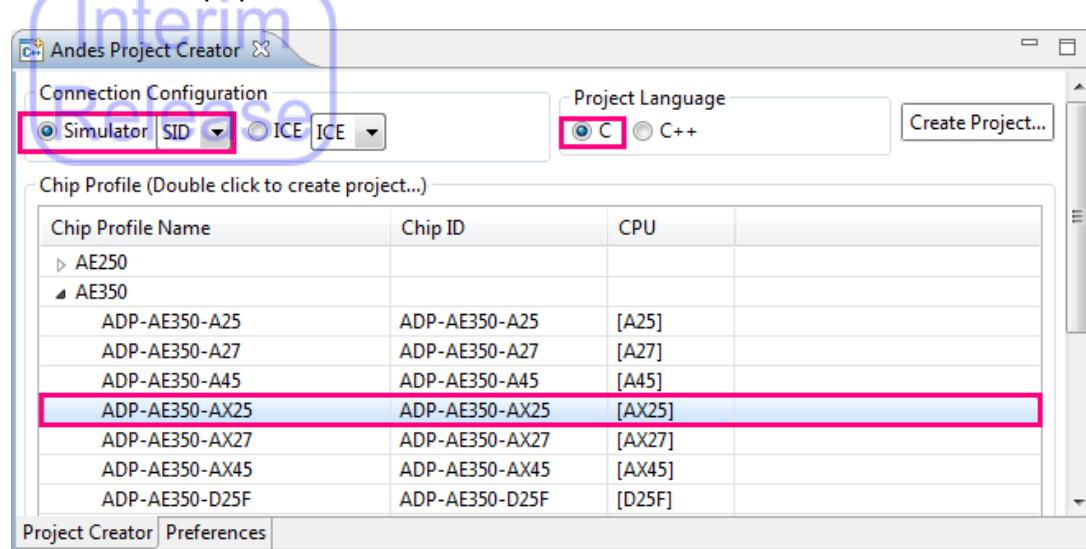
This chapter gives a quick start tutorial about using the AndeSight IDE to create/build/debug an embedded application. Follow the steps below to get an idea of the operation flow in the AndeSight IDE.

**Step 1** Double-click the desktop shortcut  or the file `AndeSight.exe` under `ANDESIGHT_ROOT\ide` to launch the AndeSight IDE.

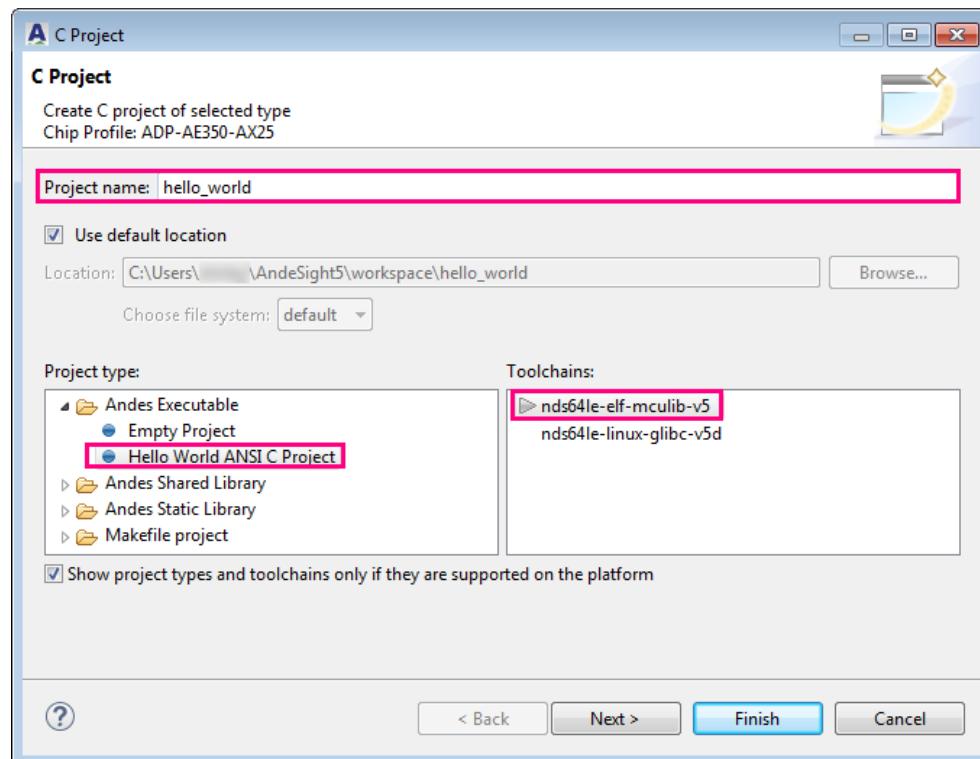
**Step 2** The **Workspace Launcher** dialog pops out, informing you that the default workspace location is at `USER_HOME\AndeSight5\workspace`. Click “OK” to continue.



**Step 3** Find the **Andes Project Creator** view on the launched AndeSight IDE. Select a connection configuration and a project language. Then, expand a group in the Chip Profile section and double-click the desired chip profile.



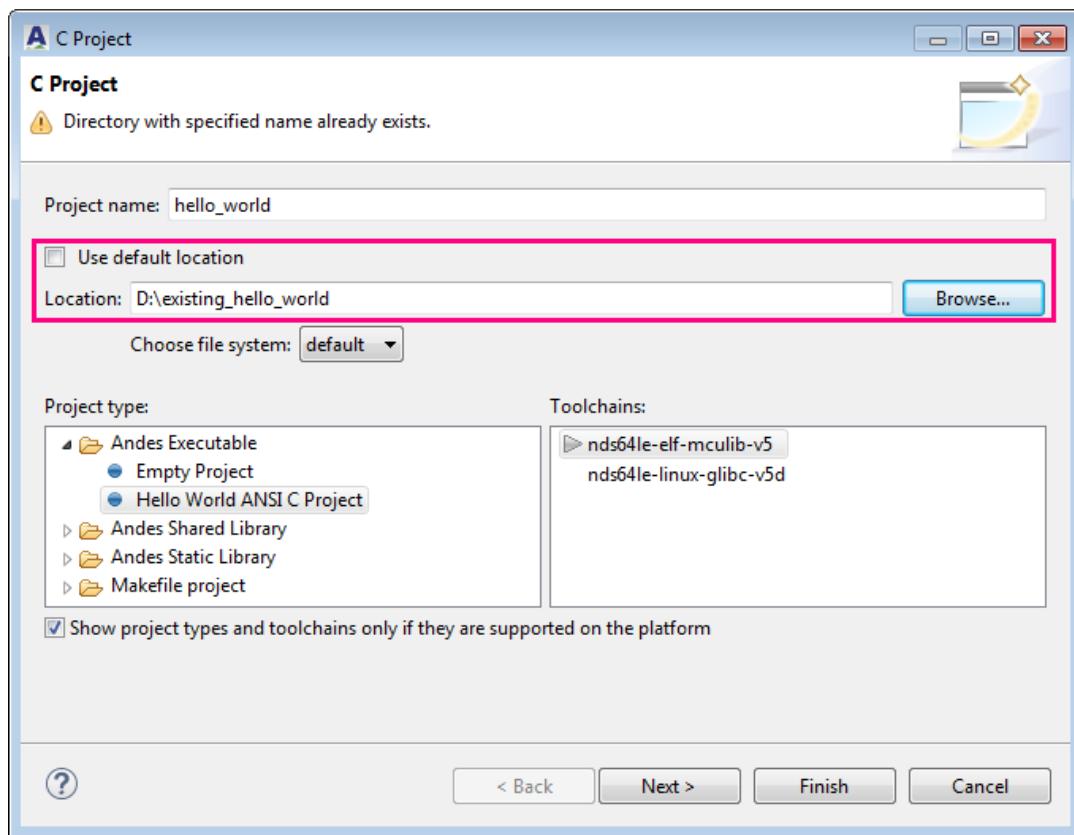
**Step 4** A dialog box opens. Enter a name for your project, select a project type and toolchain(s), and click “Finish.”



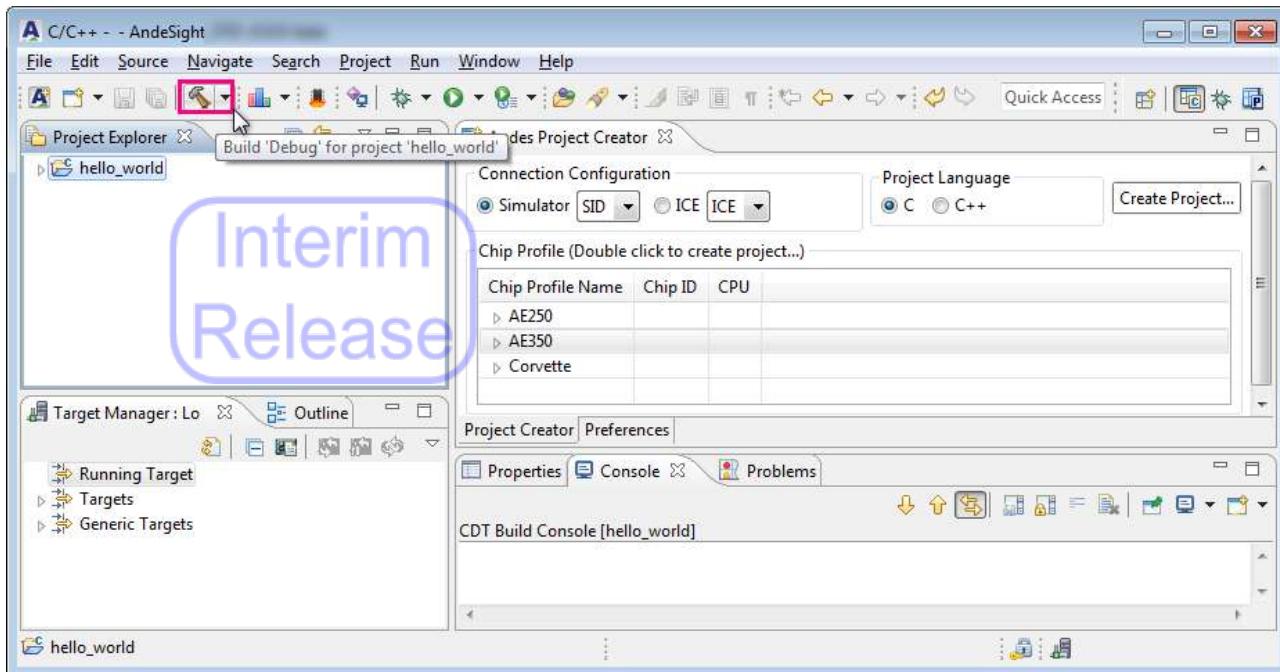
**NOTE**

To use an existing folder for the project, adopt one of the following method and give your new project a name matching the name on the existing folder:

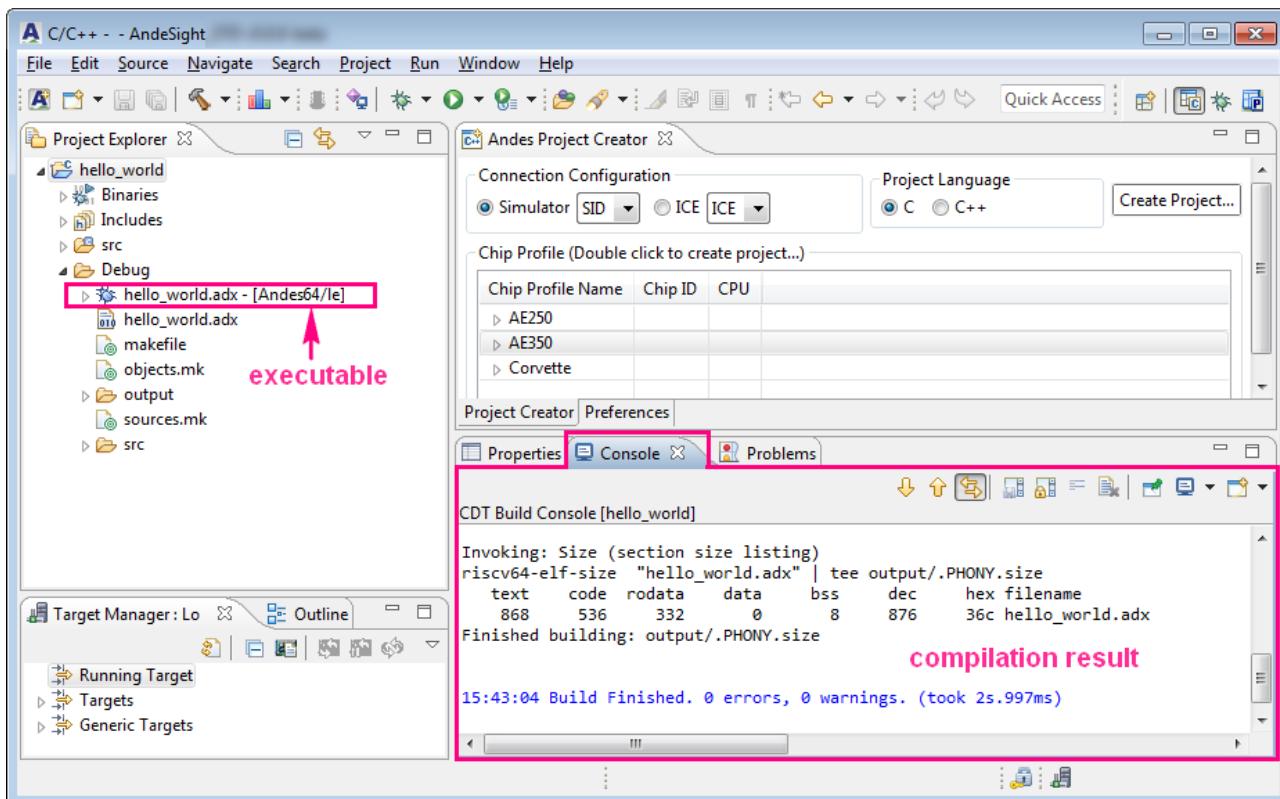
- Import the folder by selecting “File > Import” from the AndeSight main menu.
- Specify the location of the desired folder on this dialog box after deselecting the “Use default location” option.



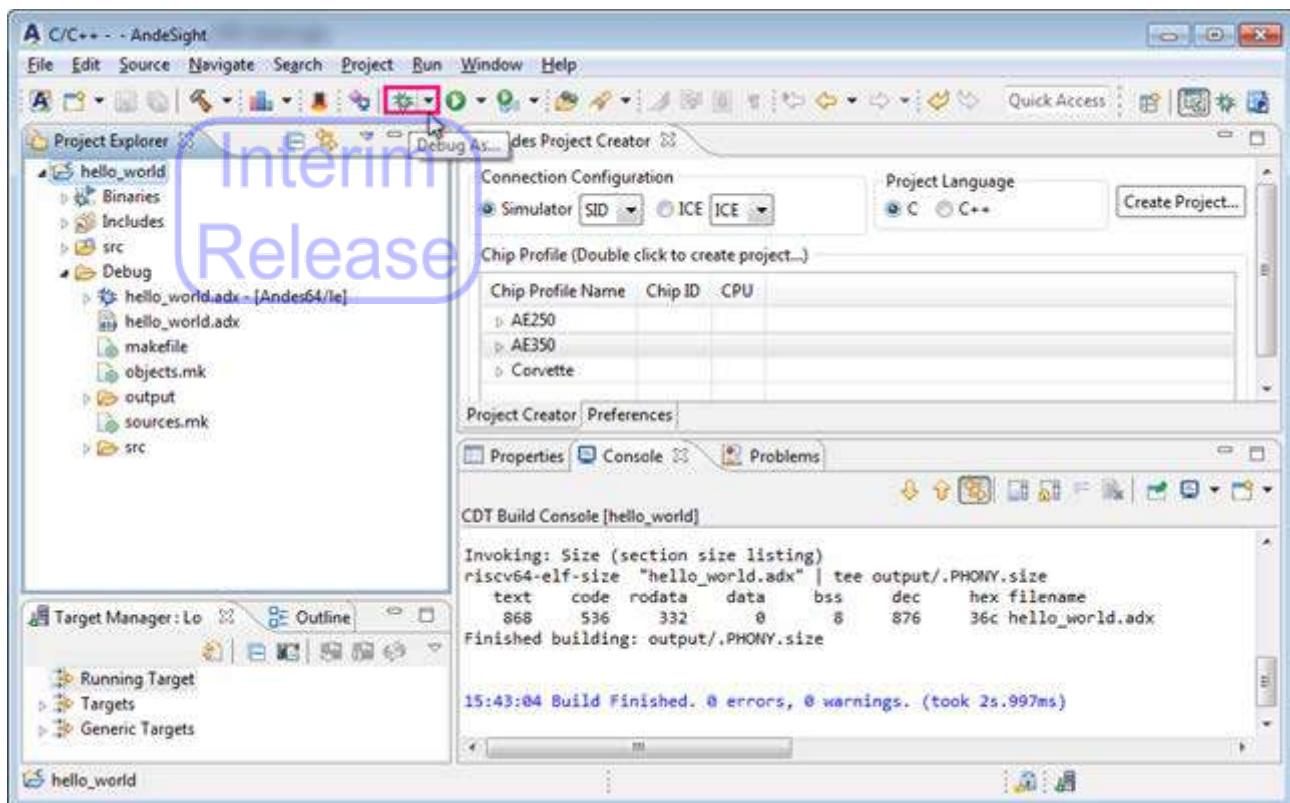
**Step 5** Select the created project in the **Project Explorer** view and click  (Build) on the AndeSight IDE toolbar to build the project.



**Step 6** Check the compilation results in the **Console** view and verify that an executable is generated under the **PROJECT\Debug** folder.



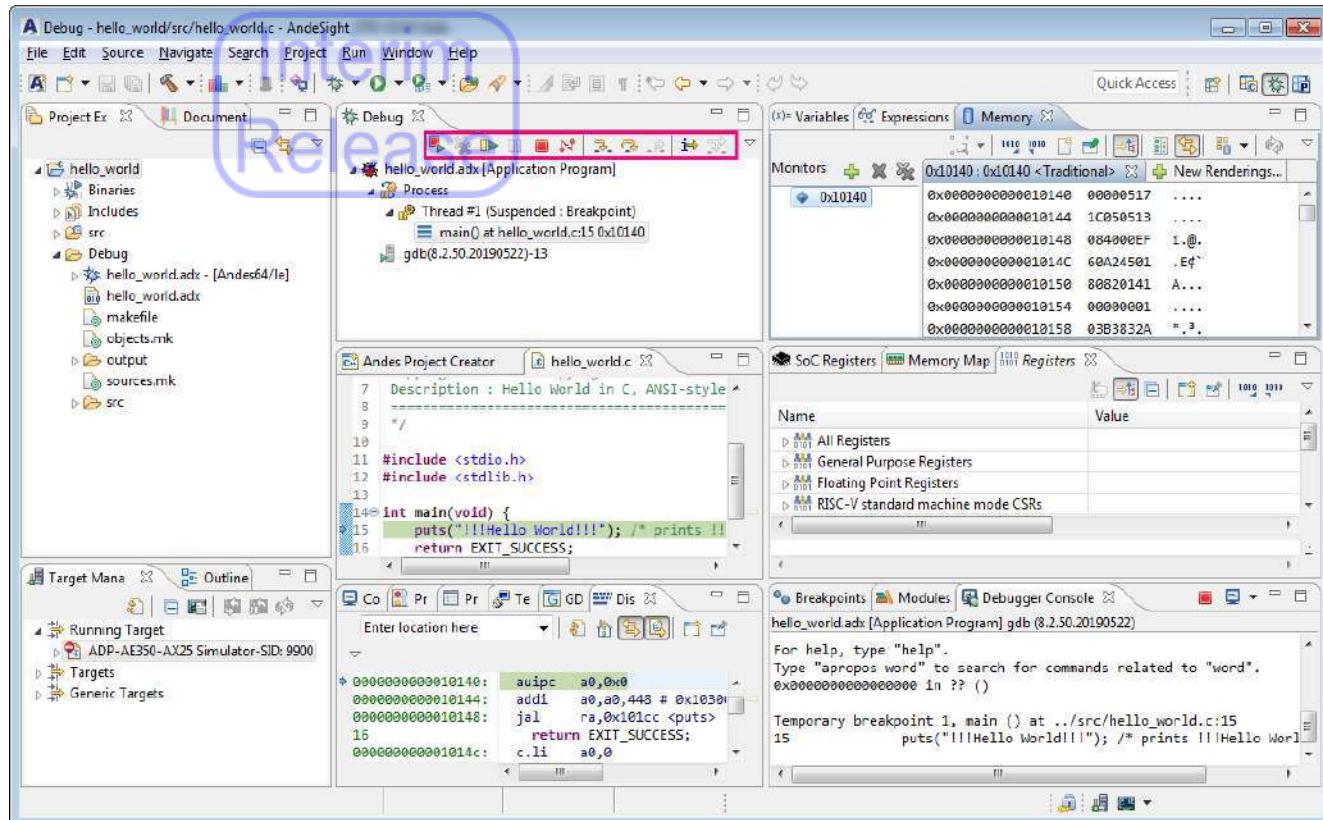
**Step 7** Click  (Debug As...) on the AndeSight IDE toolbar.



**Step 8** Select a debug configuration and click “OK” to begin debugging.

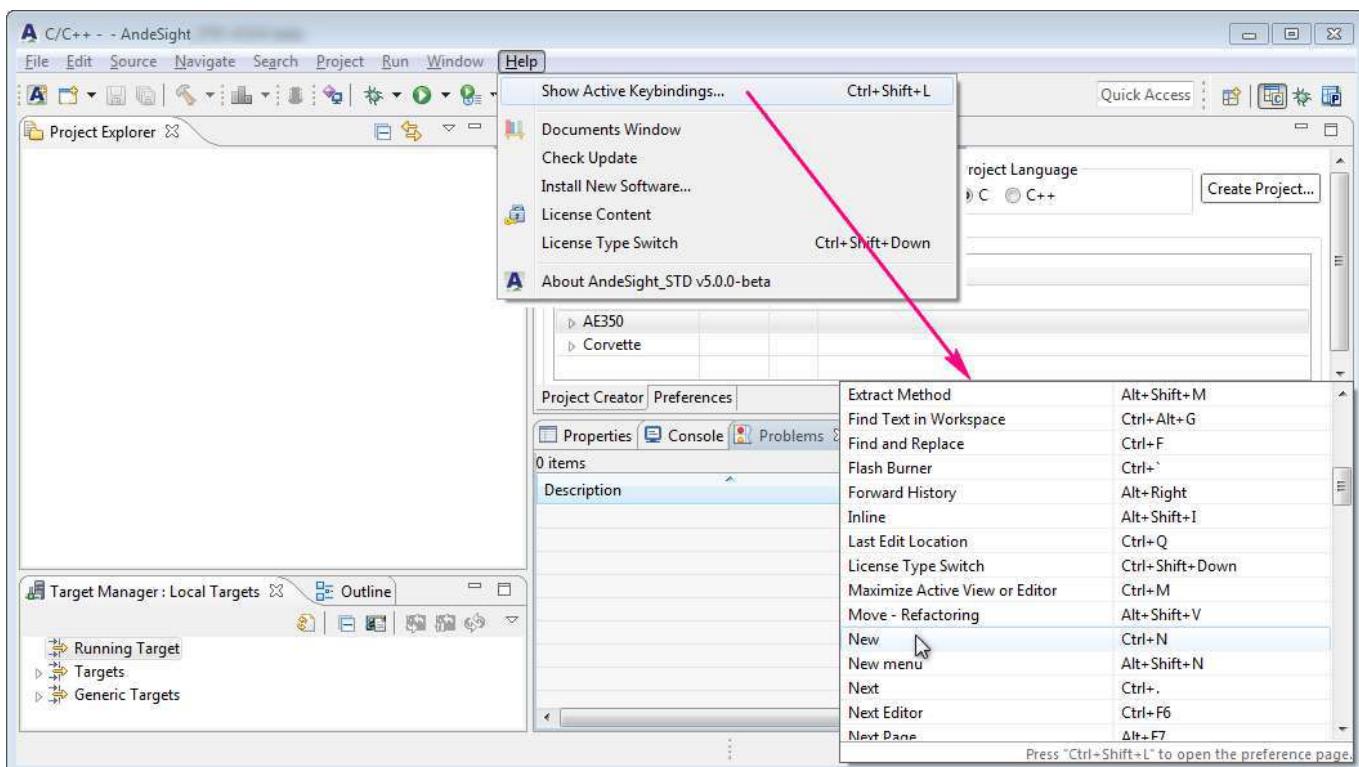


**Step 9** After a debug session is launched, click toolbar buttons on the **Debug** view to observe the results in other views (e.g., the **Variables** view, **Memory** view, **Registers** view ...).



## 1.9. Hotkeys for AndeSight operations

Keyboard shortcuts (hotkeys) can be used to make frequently-used AndeSight operations more convenient. On the AndeSight main menu, go to “Help > Show Active Keybindings...” to invoke a wizard that lists available key bindings for AndeSight operations. The keybinding wizard also allows you to perform an operation by double-clicking the associated command. For example, you can create a project either by pressing “Ctrl+N” or by double-clicking “New” on the keybinding wizard.



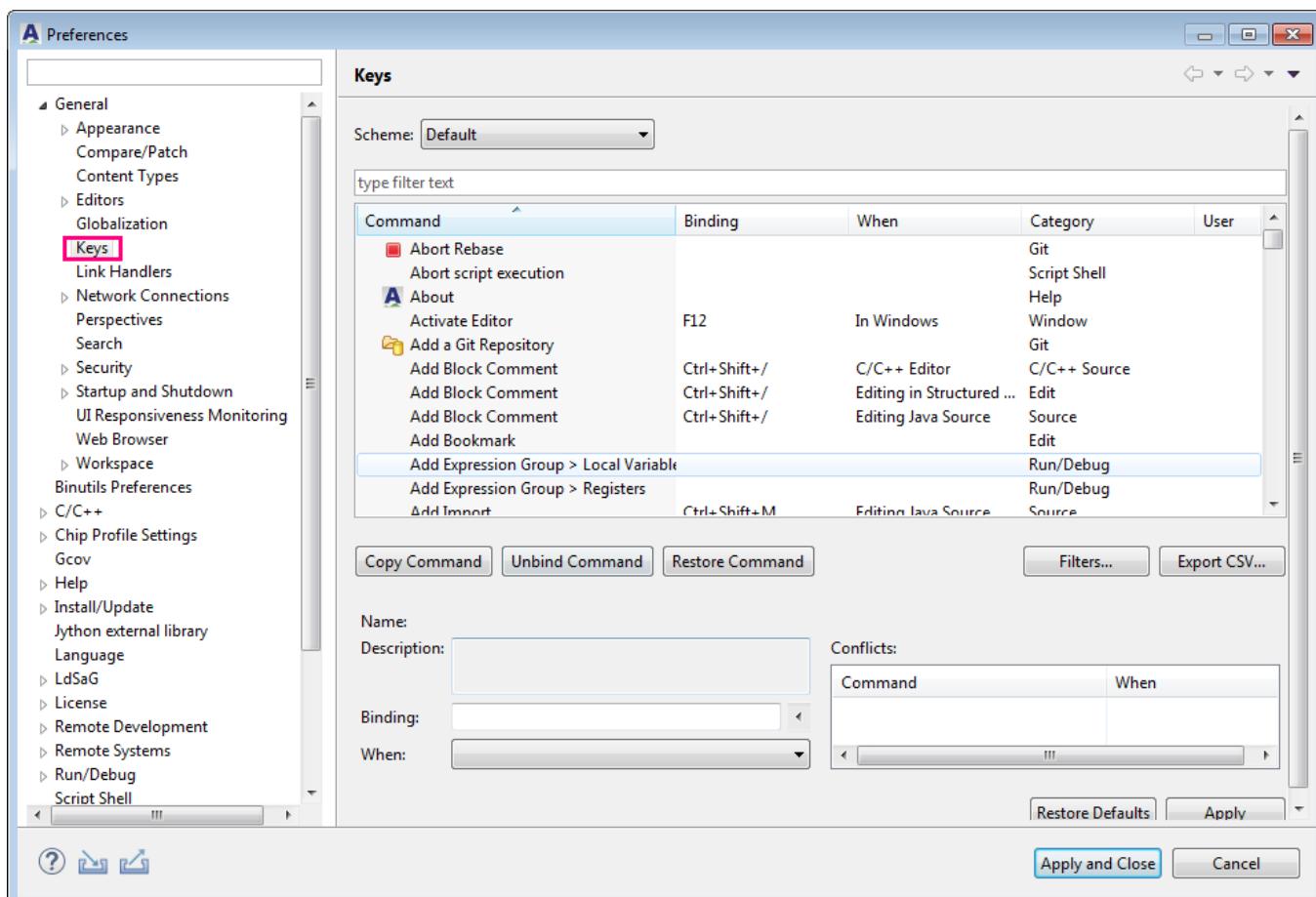
The following lists the default hotkeys for commands frequently used in AndeSight.

Table 3. Default hotkeys for frequently-used commands

Command	Toolbar button	Hotkey
Build Project		Alt+`
Debug		F11
Run		Ctrl+F11
Resume		F8

Command	Toolbar button	Hotkey
Terminate		Ctrl+F2
Terminate and Relaunch		Ctrl+F3
Step Into		F5
Step Over		F6
Step Return		F7
Flash Burner		Ctrl+`

You may assign or change key bindings to better suit your needs. Simply access the **Keys** preference page through “AndeSight main menu > Window > Preferences > General > Keys” or by pressing “Ctrl+Shift+L” on the keybinding wizard. This page provides a complete list of menu commands in AndeSight as well as their default key bindings.



To change or add a hotkey for a particular command, proceed as follows:

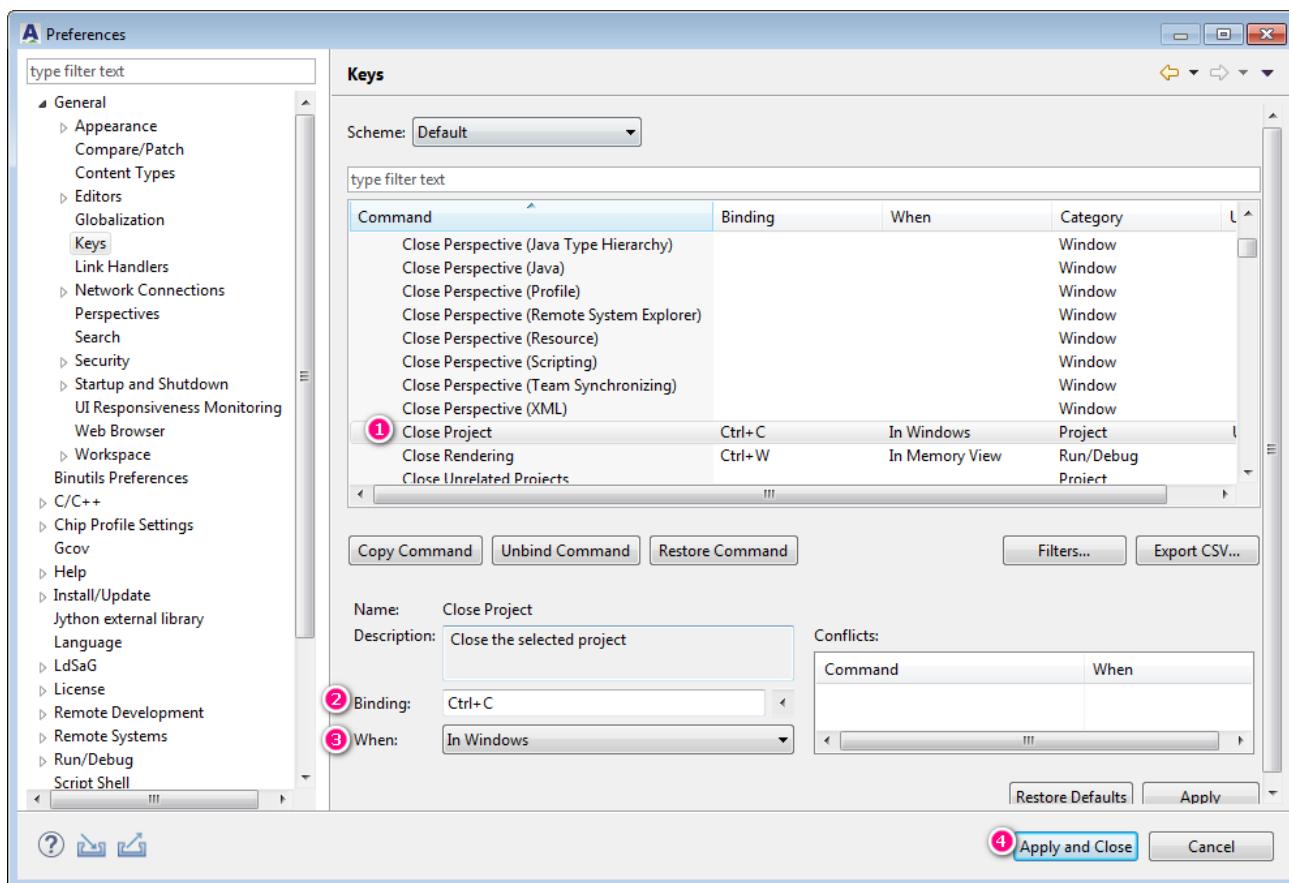
The information contained herein is the exclusive property of Andes Technology Co. and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of Andes Technology Corporation.

**Step 1** From the table on the **Keys** preference page, select the command for which you would like to assign or modify the key bindings.

**Step 2** In the “Binding” field, enter the shortcut keys by pressing the actual keys rather than typing the key names. For example, assign the key binding “Ctrl+C” for the “Close Project” command by holding down the Ctrl key and then pressing the C key.

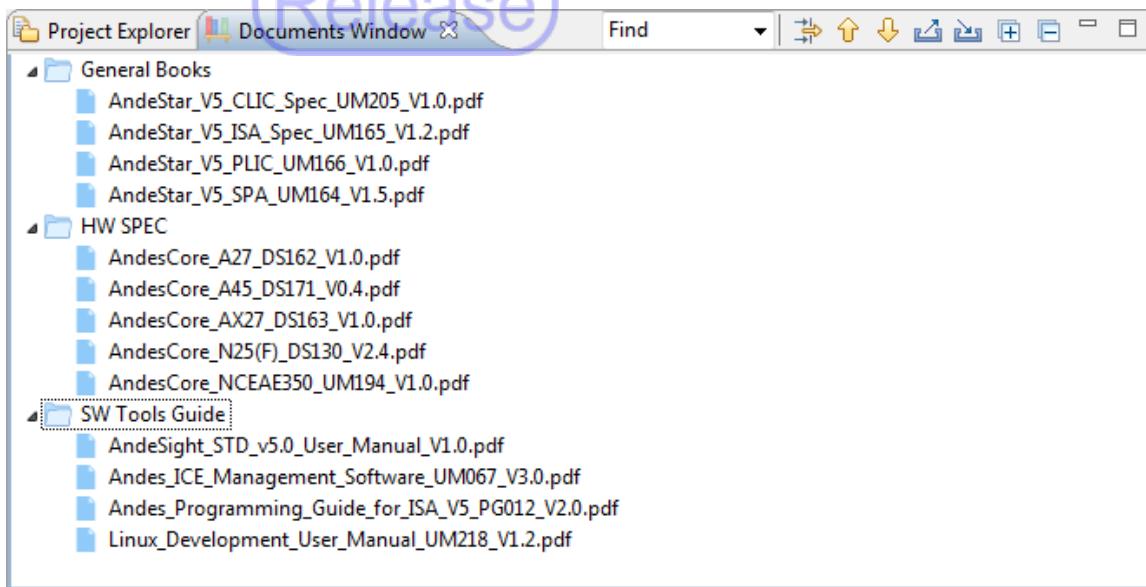
**Step 3** Select the key binding context in the “When” combo box.

**Step 4** Click “Apply and Close” on the **Preferences** dialog.

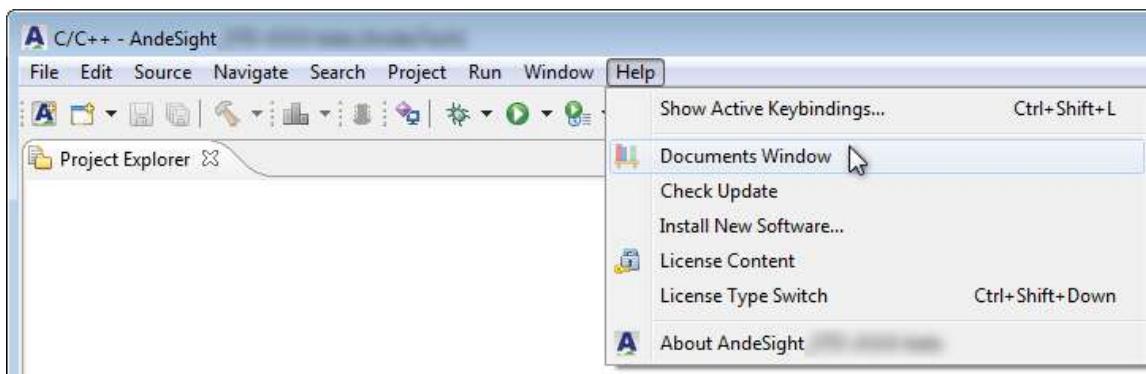


## 1.10. Reference Documentation and Documents Window

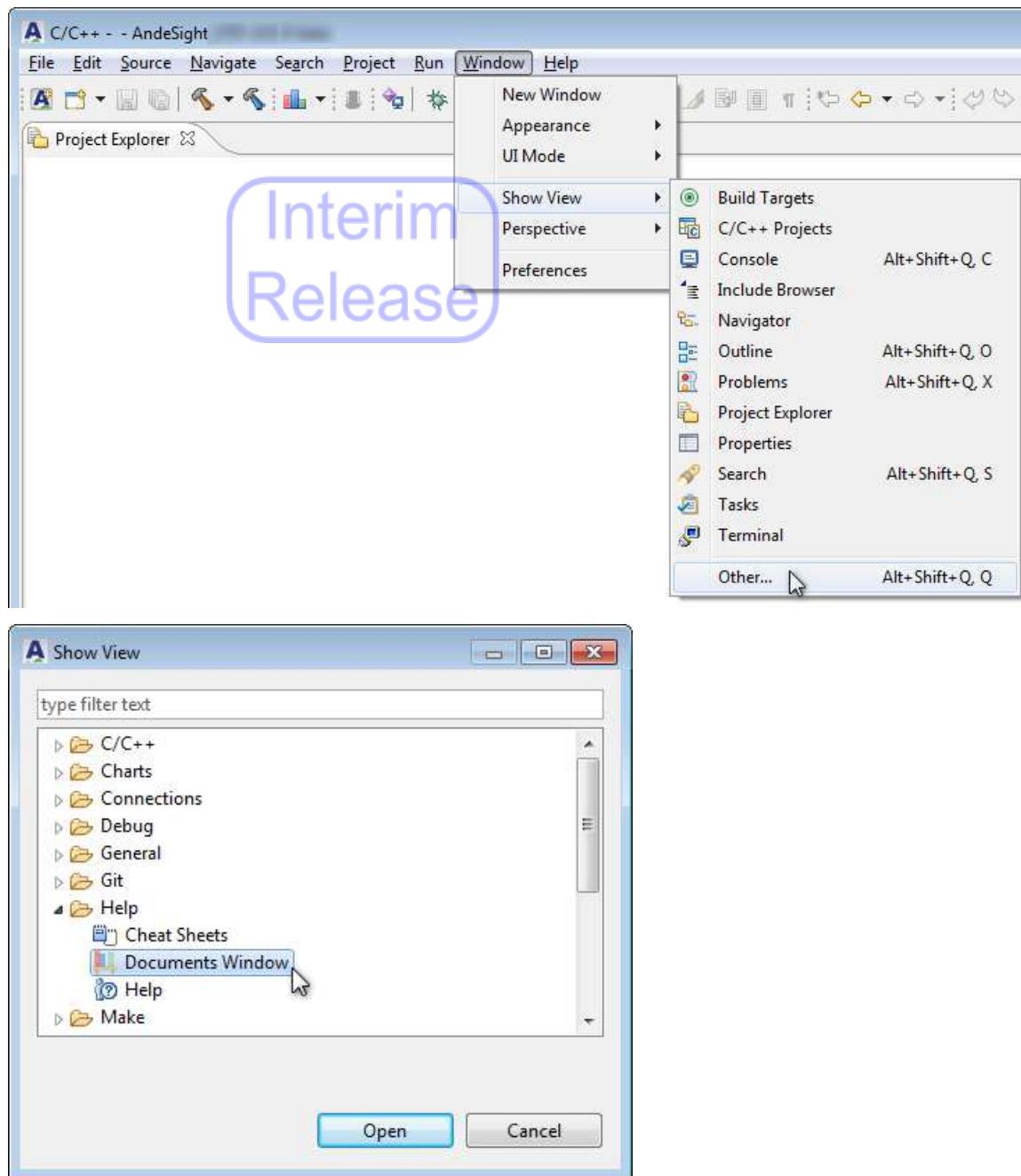
You may find AndeSight manuals, tool guides, tutorials, specification documents or support materials in the **Documents Window** view. This view is a container and editor of links leading to local or online reference resources associated with your development. It enables you to access and manage required reference materials.



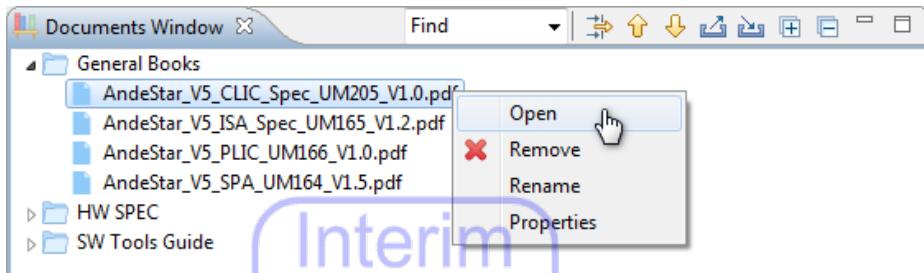
The **Documents Window** view can be invoked through “AndeSight main menu > Window > Help > Documents Window”.



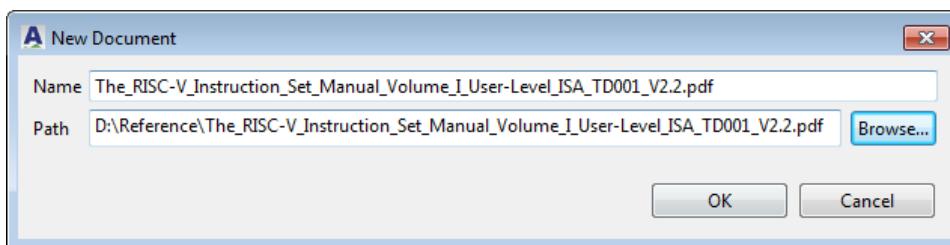
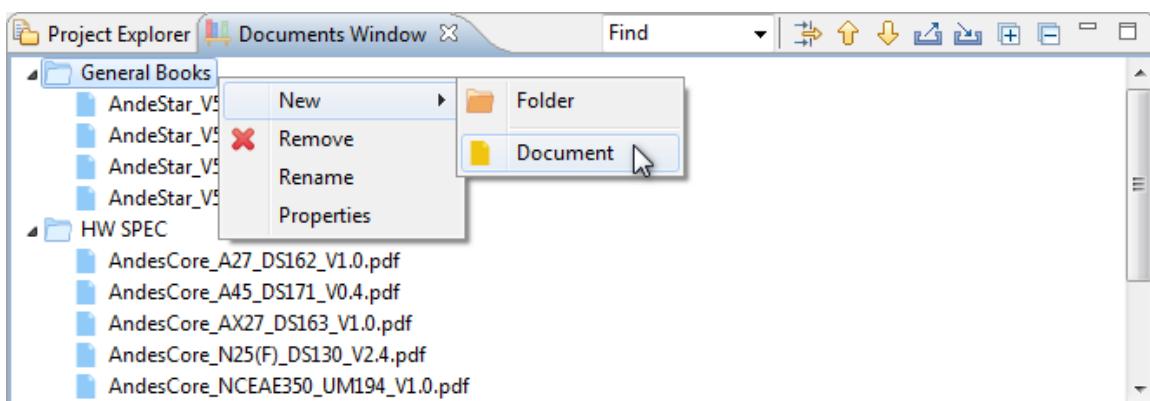
You can also invoke the view by selecting “Window > Show View > Other...” from the AndeSight main menu and then selecting “Help > Documents Window” in the **Show View** dialog.



When first invoked, the view is positioned next to the **Project Explorer** view and displays documents prepared under **ANDESI GHT\_ROOT\doc** by default. You can open a document in the view by double-clicking it or selecting “Open” in its pull-down menu.

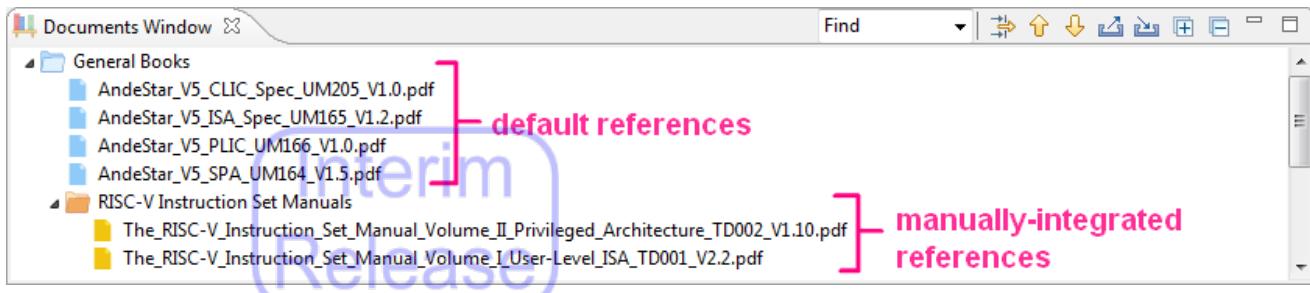


Reference materials in **Documents Window** are organized in a tree structure and categorized in several folders for ease of navigation. You can reorganize the documentation tree for your development needs or preferences by dragging documents to a different folder, deleting unneeded folders or documents, creating a new folder, or integrating external documents into the view. Most of these operations can be carried out via the pull-down menu of a selected folder/document. For example, to incorporate a new reference link into an existing folder in the view, just right-click the folder in **Documents Window** to select “New > Document” in its pull-down menu, and then specify the name and location (i.e., local or network path) of the desired reference material in the pop-up **New Document** dialog.



You can also add a new link to the tree by directly dragging and dropping an external folder/document to the **Documents Window** view. Folder or document links manually integrated into **Documents Window** are denoted by yellow icons. In contrast, links of original

folders/documents in **ANDESI GHT\doc** are denoted by blue icons.



The toolbar of the **Documents Window** view provides more options for you to manage the display of reference documents, as explained below.

### Search

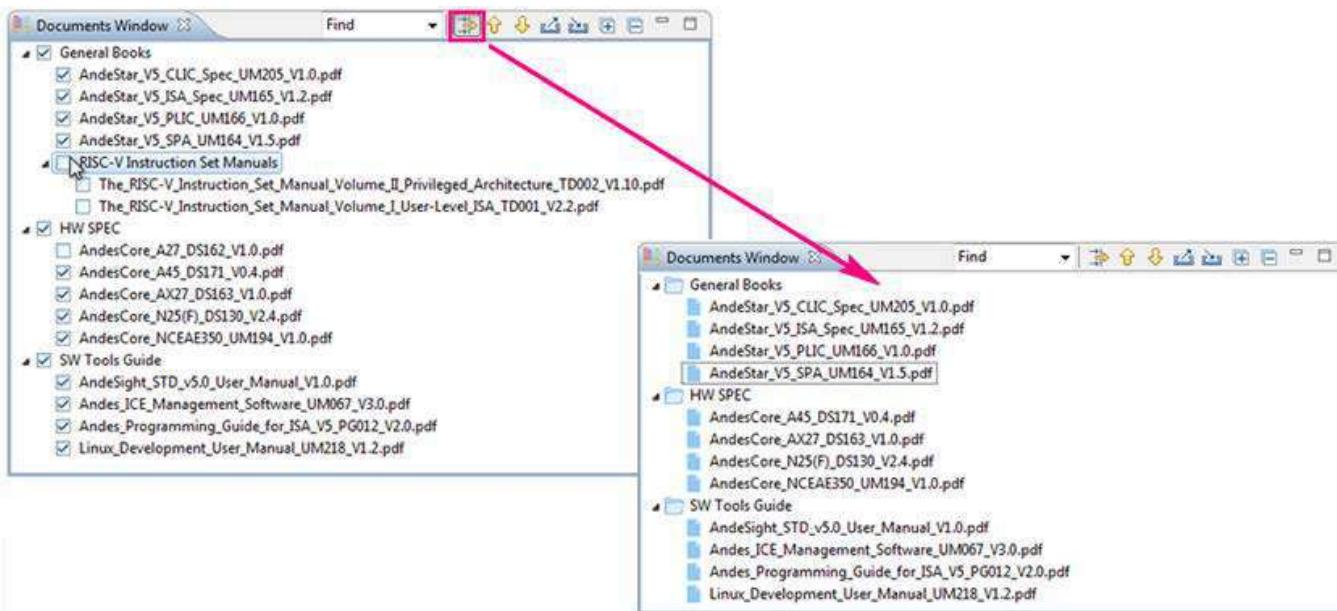


This field allows you to search for specific documents in the documentation tree with supplied keywords.

### Filter



This button helps you to specify what to hide or show in the documentation tree. To prevent certain folders/documents from being displayed in the tree, just select this button to uncheck the items and then unselect this button to apply the display setting.



### Move up



Click this button to move a selected document up within its primary folder.

**Move down**

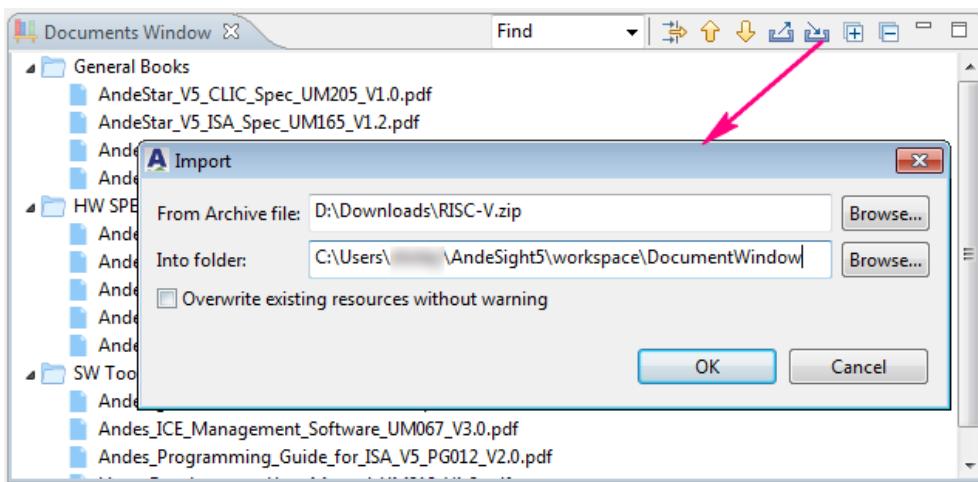
Click this button to move a selected document down within its primary folder.

**Export**

This button allows you to export all documents that were manually integrated into **Documents Window** (i.e., documents denoted by yellow icons) to a compressed file.

**Import**

This button allows you to import a compressed file of reference documents and integrate the documents into the documentation tree. Note that the compressed file must be generated using the toolbar button  (Import) in the **Documents Window** view.

**Expand all**

Click this button to expand all folders in the documentation tree .

**Collapse all**

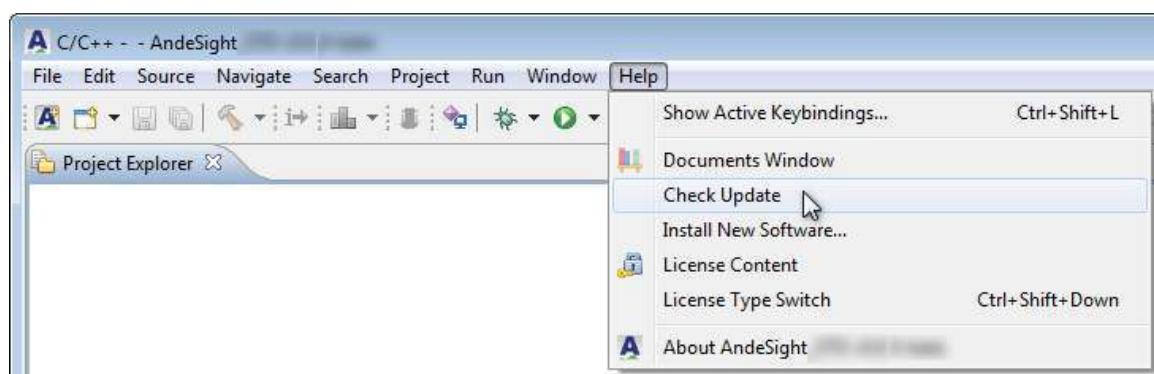
Click this button to collapse all folders in the documentation tree.

## 1.11. Future updates or patches

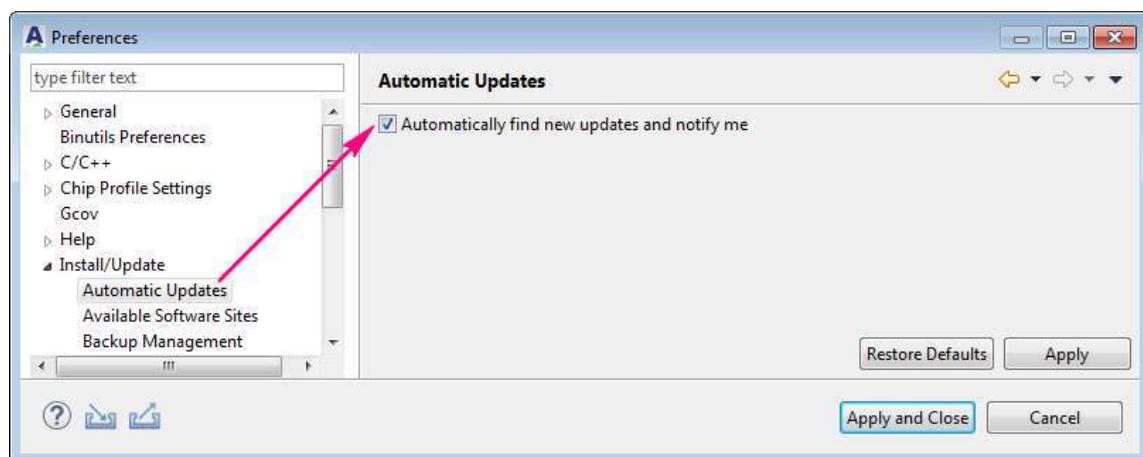
AndeSight provides a friendly interface for you to apply updates or patches. Before you commence the update or patch process, make sure that there is no running target and that no file or folder under the AndeSight root directory is in use. Then, follow below to check for updates and install:



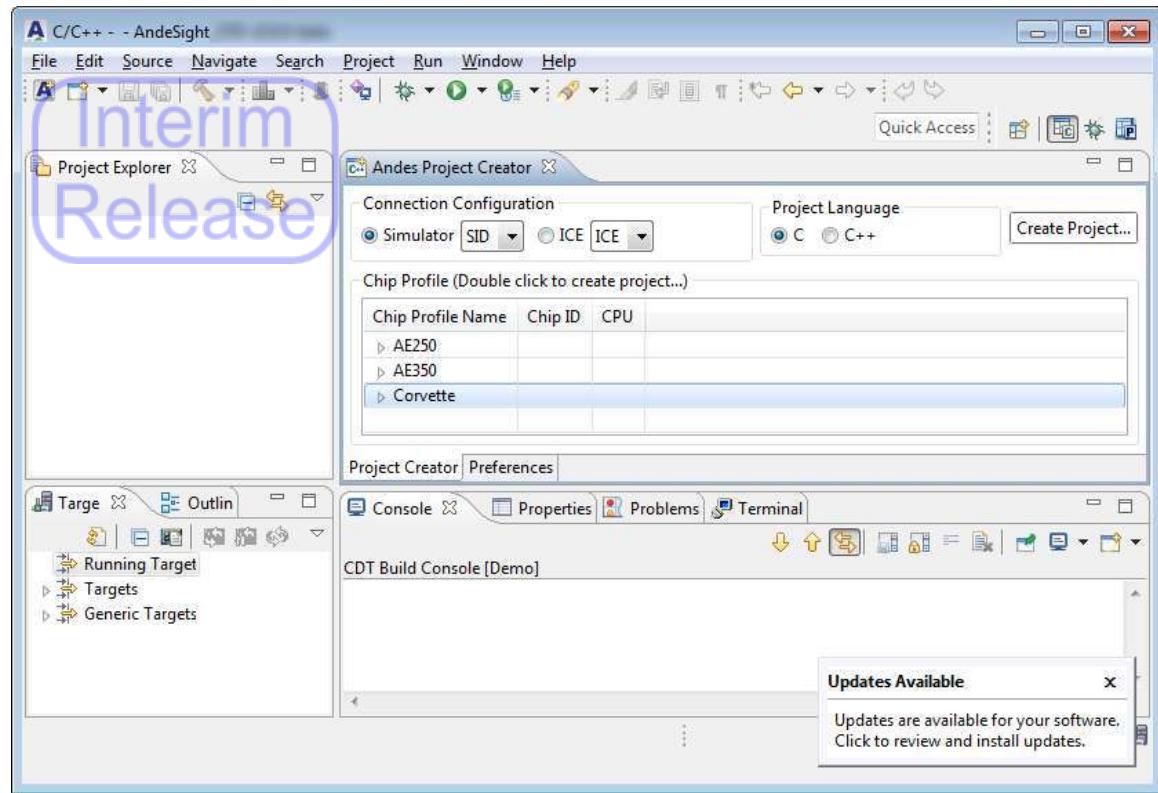
- Step 1** Click “Help > Check Update” on the AndeSight main menu to see if there is any available update or patch.



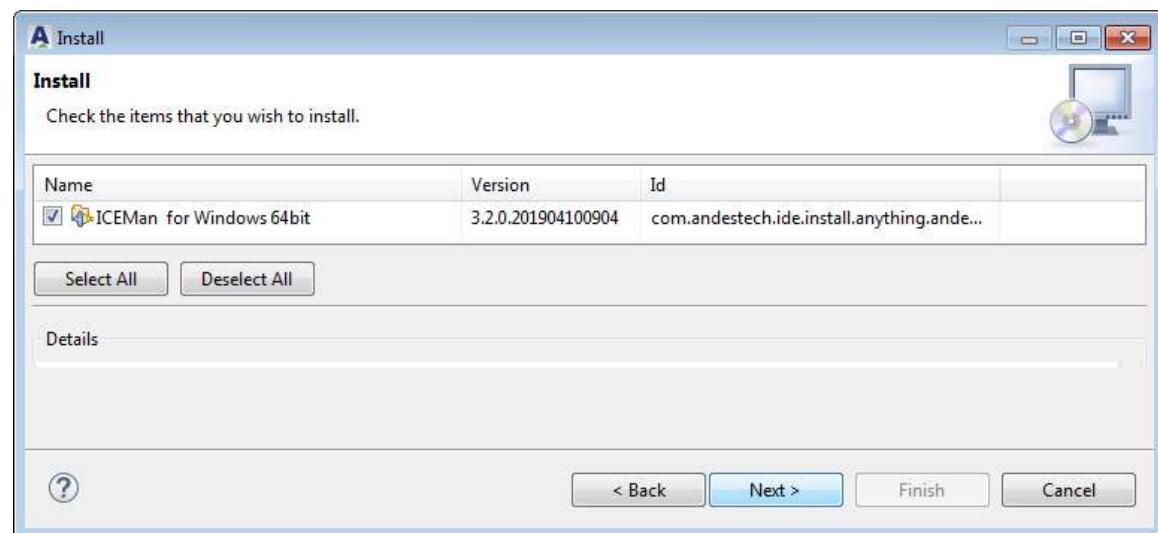
You can also configure AndeSight to check for updates automatically every time when the IDE is launched. Just click “Window > Preferences” on the AndeSight main menu to invoke the **Preferences** dialog, select “Install/Update > Automatic Updates” in the navigation pane and check the option “Automatically find new updates and notify me” on the **Automatic Updates** page.



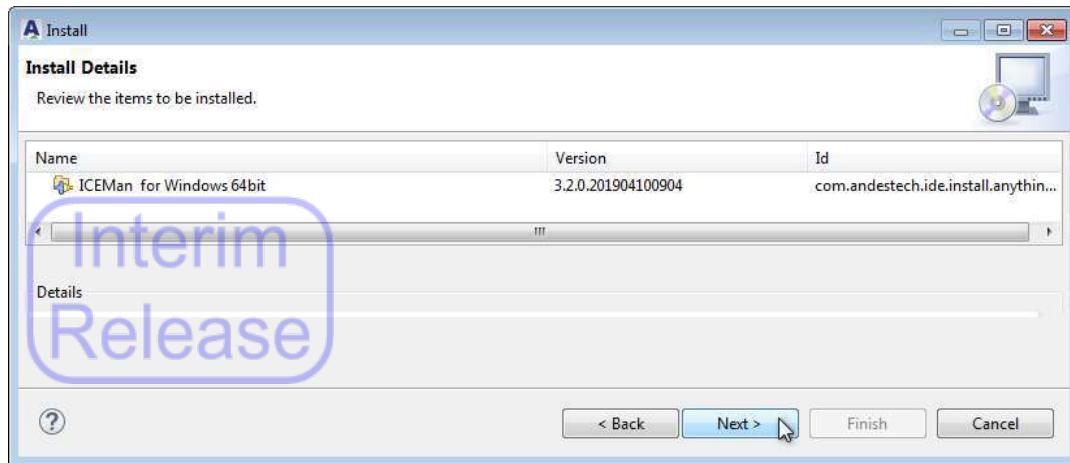
**Step 2** If updates are found, a notification wizard will appear in the lower-right corner of the workbench. Click on the wizard.



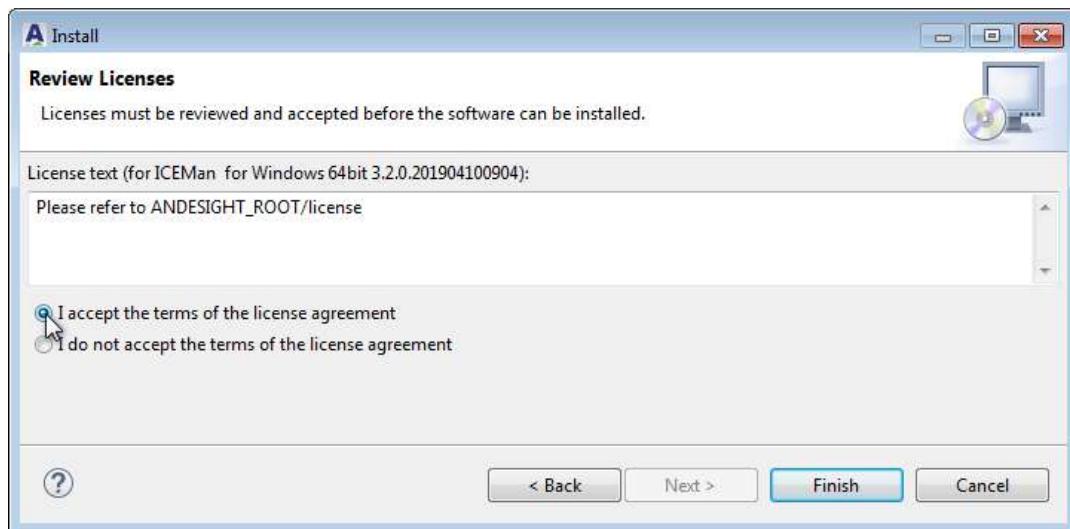
**Step 3** View the available update package(s), select the desired one(s) to be installed, and click “Next.”



**Step 4** Review the installation summary and click “Next.”



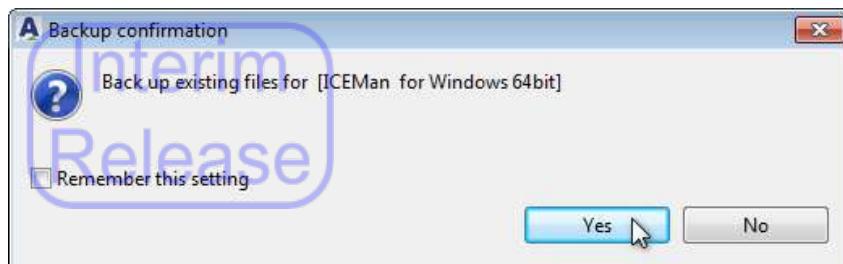
**Step 5** On the **Install** dialog, select the option “I accept the terms of the license agreement” and click “Finish” to start the installation.



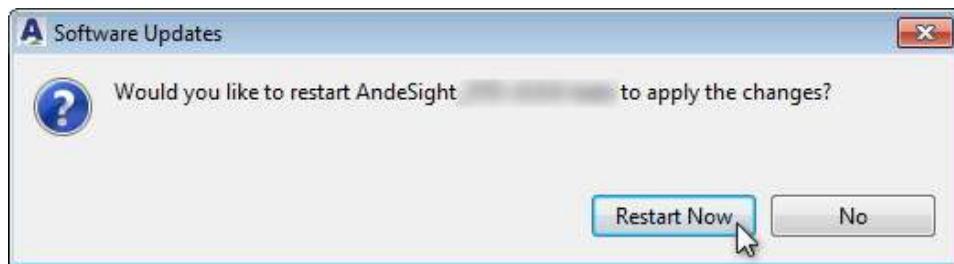
**Step 6** The **Installing Software** dialog pops up to show the installation progress.



During the installation, you will be prompted to back up existing files for the component(s) to be updated. Select “Yes” to do the backup or “No” to overwrite the existing files.

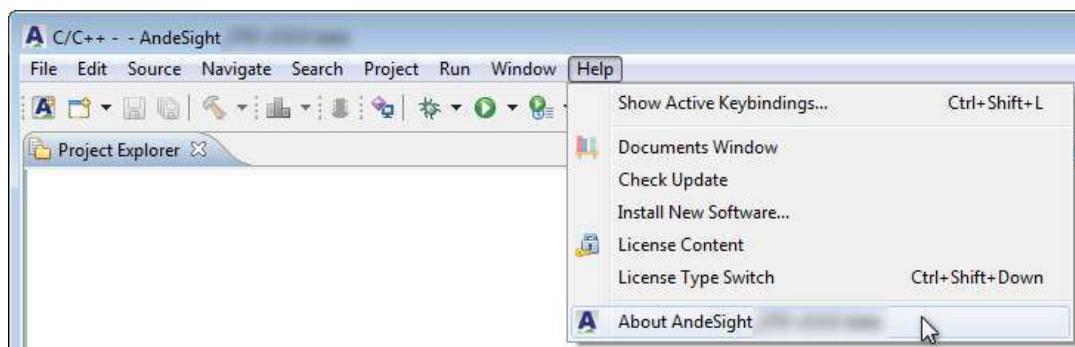


**Step 7** Upon the completion of the installation, you will be prompted to restart AndeSight. Click “Restart Now” on the **Software Updates** dialog.

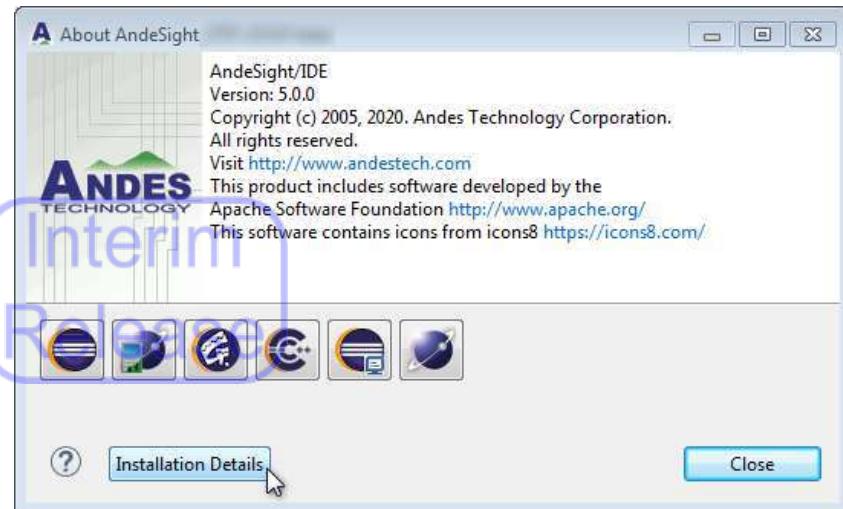


**Step 8** On the restarted AndeSight, follow below to confirm that selected update packages are installed.

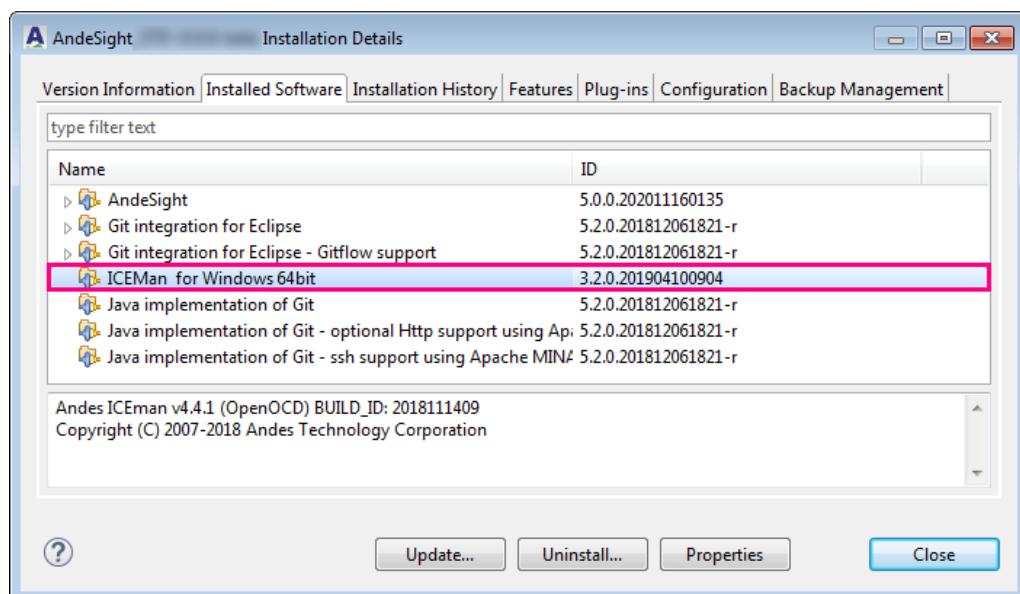
1. From the AndeSight main menu, select “Help > AndeSight\_EDITION\_VERSION”.



2. On the invoked dialog, click the button “Installation Details.”

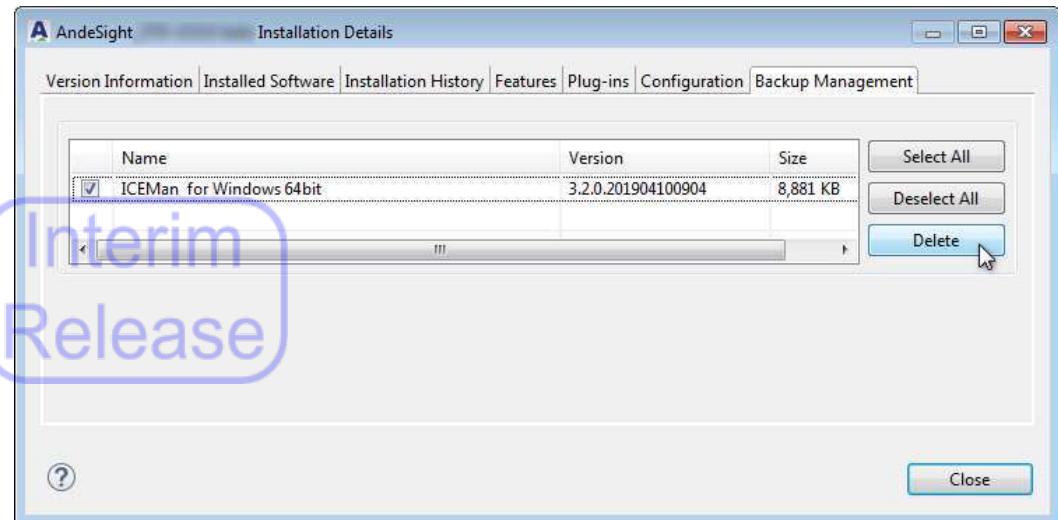


3. Click the **Installed Software** tab and confirm that the installed software include the package(s) you just installed.



If necessary, you may also revert the updated packages from the backup files by selecting them in this tab and clicking “Uninstall...”.

4. Click the **Backup Management** tab to view packages you've backed up for the installation. If any of them is no longer needed, just select it from the list and click “Delete” to clear the backup files.



## 2. AndeSight STD IDE

### 2.1. Creating and building AndeSight projects

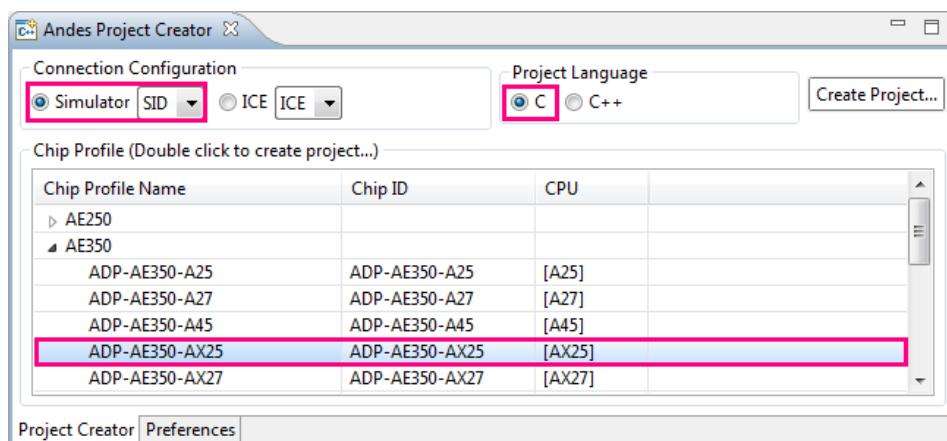
You can create a project or import an existing project to the AndeSight workbench. A project is said to be “built” when its source files are compiled and linked to make executables or object files.

#### 2.1.1. Project creation

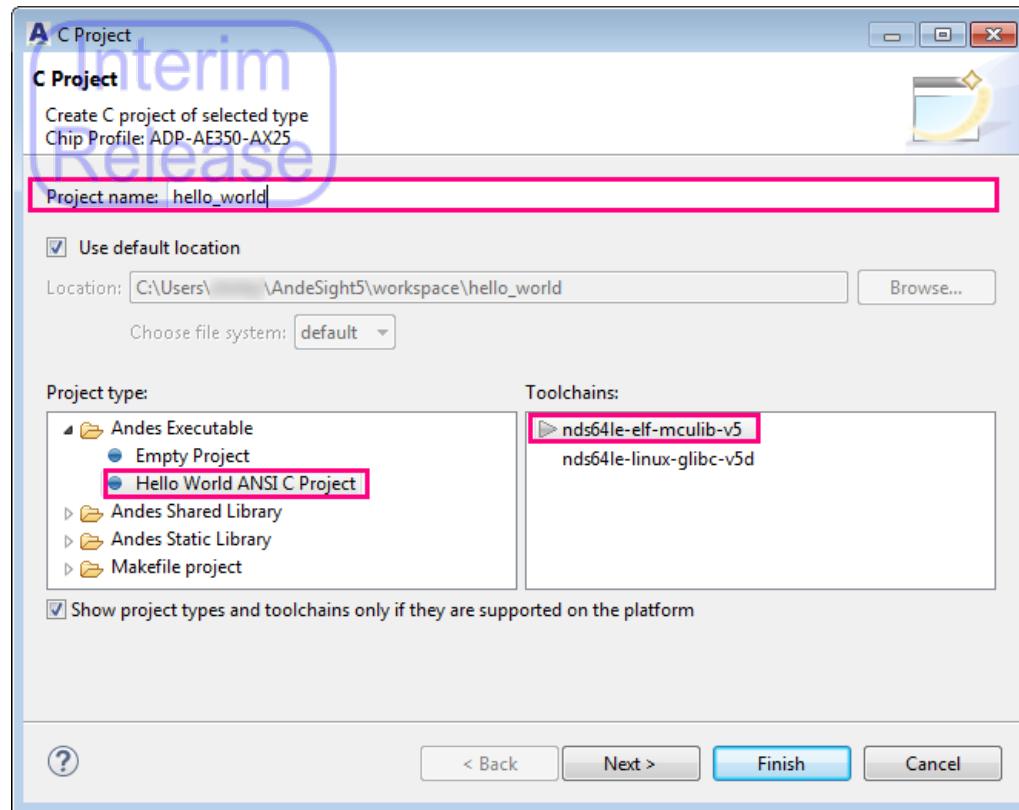
AndeSight introduces distinct project creation flows for targets with detailed specifications (Pre-defined Targets) and targets defined only by the Andes Instruction Set Architecture, endian type, library, and executable format (Generic Targets). For Pre-defined Targets, project creation begins from the **Andes Project Creator** view. For Generic Targets, project creation follows the conventional approach used for C/C++ projects. For step-by-step instructions, please refer to Sections 2.1.1.1 (Pre-defined Targets) or 2.1.1.2 (Generic Targets). Additional information pertaining to these targets may also be found in Section 2.2.

##### 2.1.1.1 Creating a new project: Pre-defined target

**Step 1** After launching the AndeSight IDE, find the **Andes Project Creator** view or click the **Andes Project Creator** button  on the AndeSight toolbar. Select a connection configuration and language for your project. Then, expand a group tag in the Chip Profile section and double-click the desired chip profile.

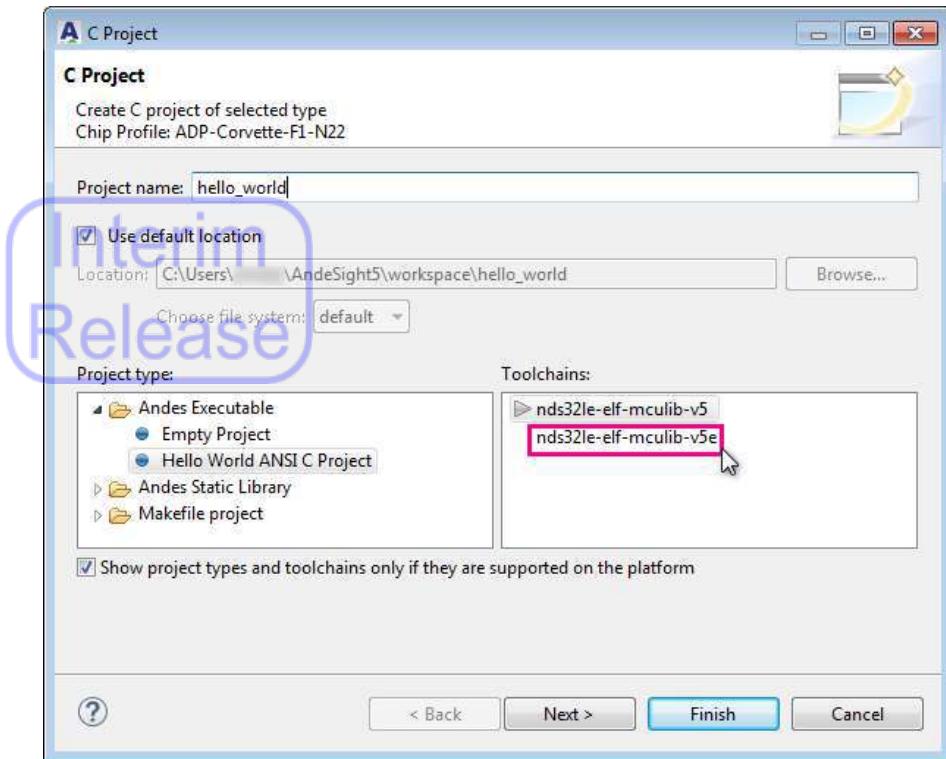


**Step 2** At the invoked prompt, enter a name for your project, select the type of project, and specify one or multiple toolchains for the project. Click “Next.”

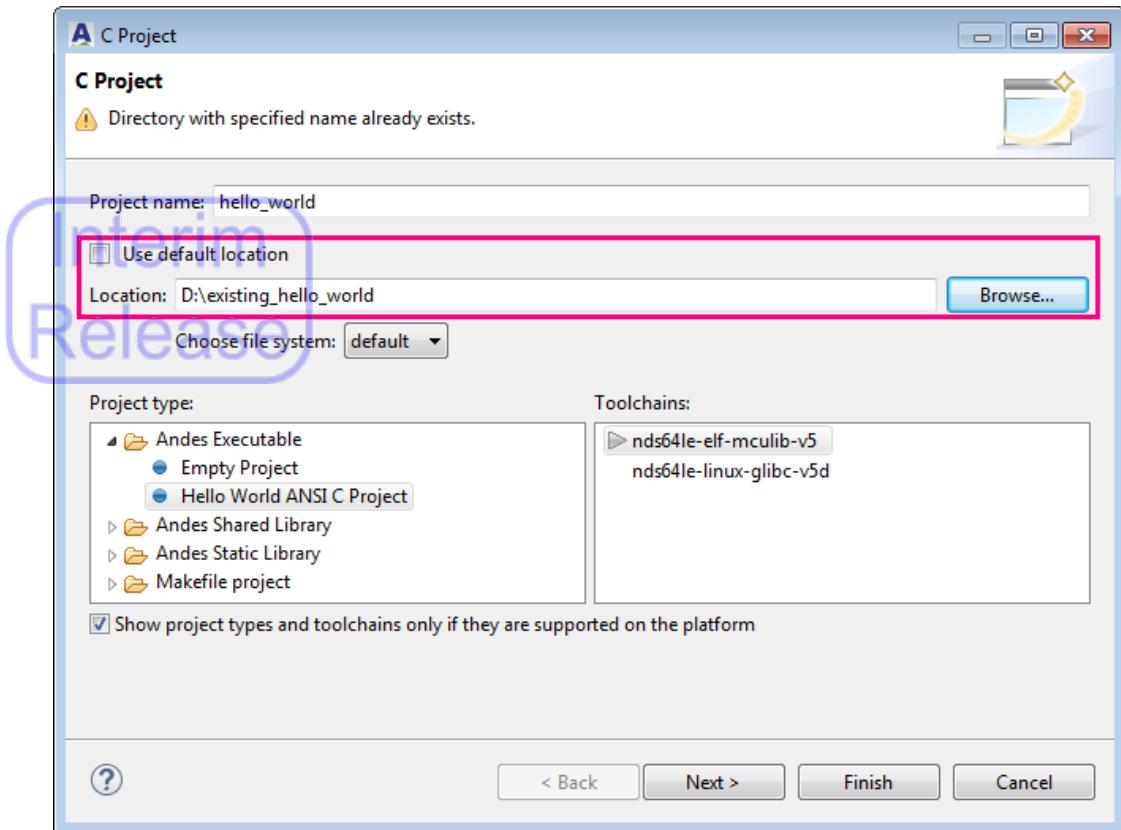


#### NOTE

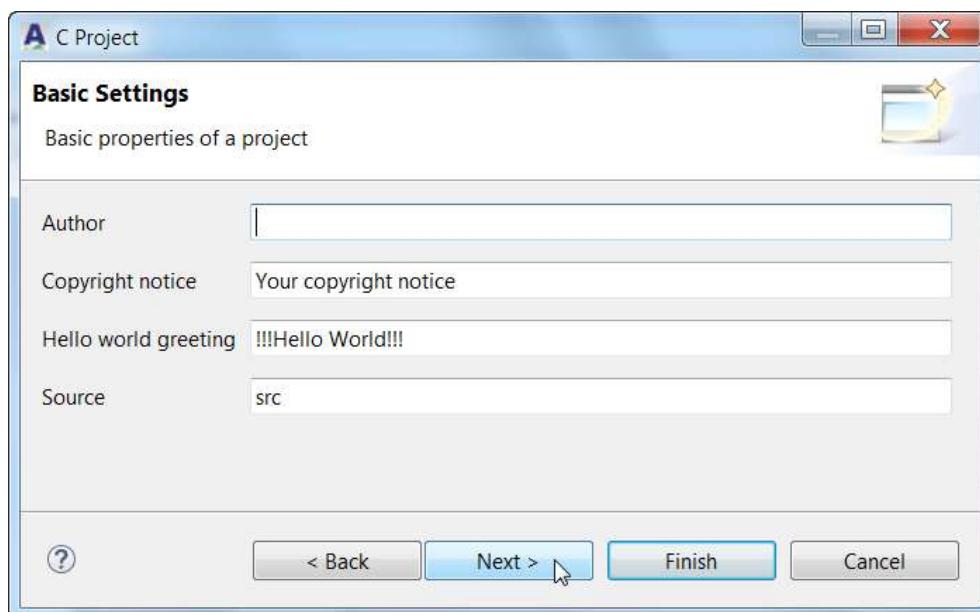
1. Some characters cannot be used for project names. Please ensure your project name contains characters only from the following set:  
A-Z a-z 0-9 ! @ % ^ [ ] . - + , ~ \_.
2. To use a Corvette-F1 target pre-integrated with fixed-configuration N22 RTL in Andes FreeStart program, make sure you change the toolchain for the chip profile “ADP-Corvette-F1-N22” to nds32le-elf-[newlib|mculib]-v5e.



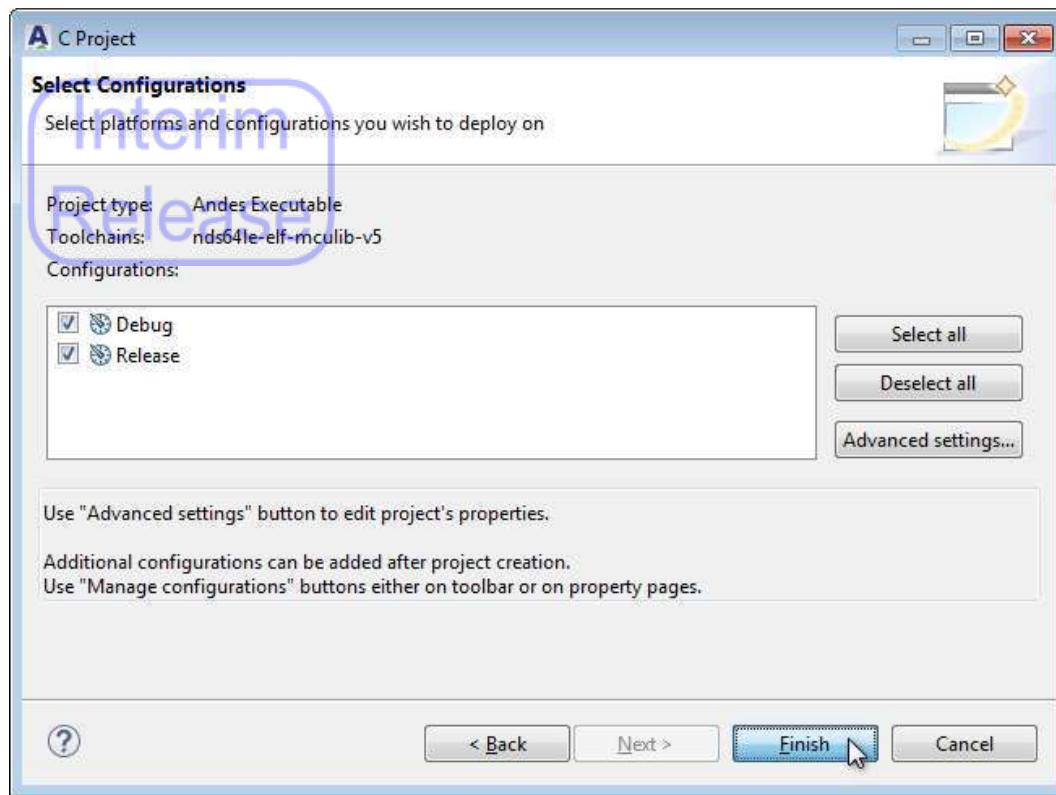
3. To use an existing folder for the project, adopt one of the following method and give your new project a name matching the name on the existing folder:
  - Import the folder by selecting “File > Import” from the AndeSight main menu.
  - Specify the location of the desired folder on this dialog box after deselecting the “Use default location” option.



**Step 3** On the **Basic Settings** prompt, you have the option to configure the basic properties of the project by filling in the fields. Click “Next” then.

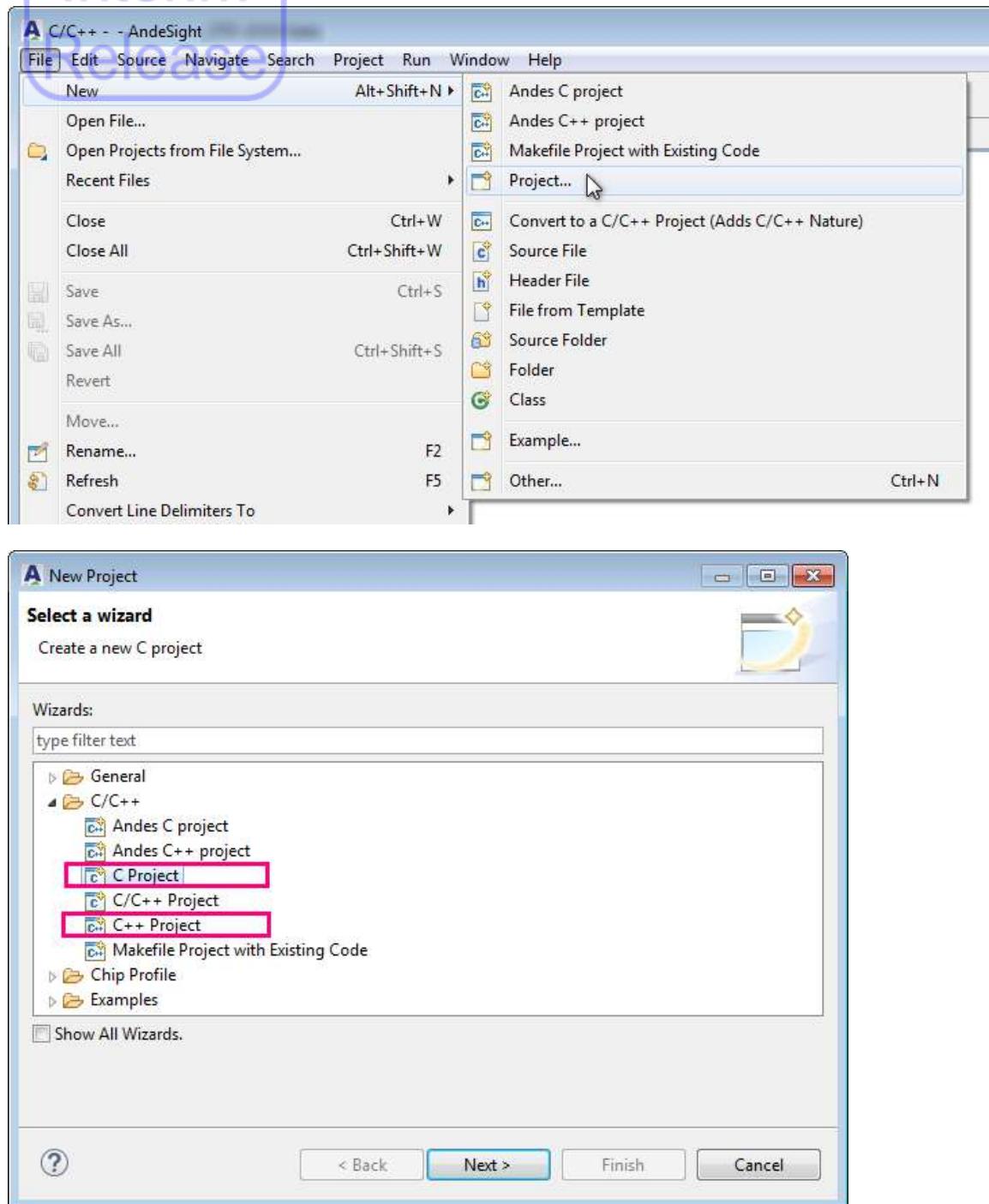


**Step 4** Select the configuration(s) for the project and click “Finish” to create the project.

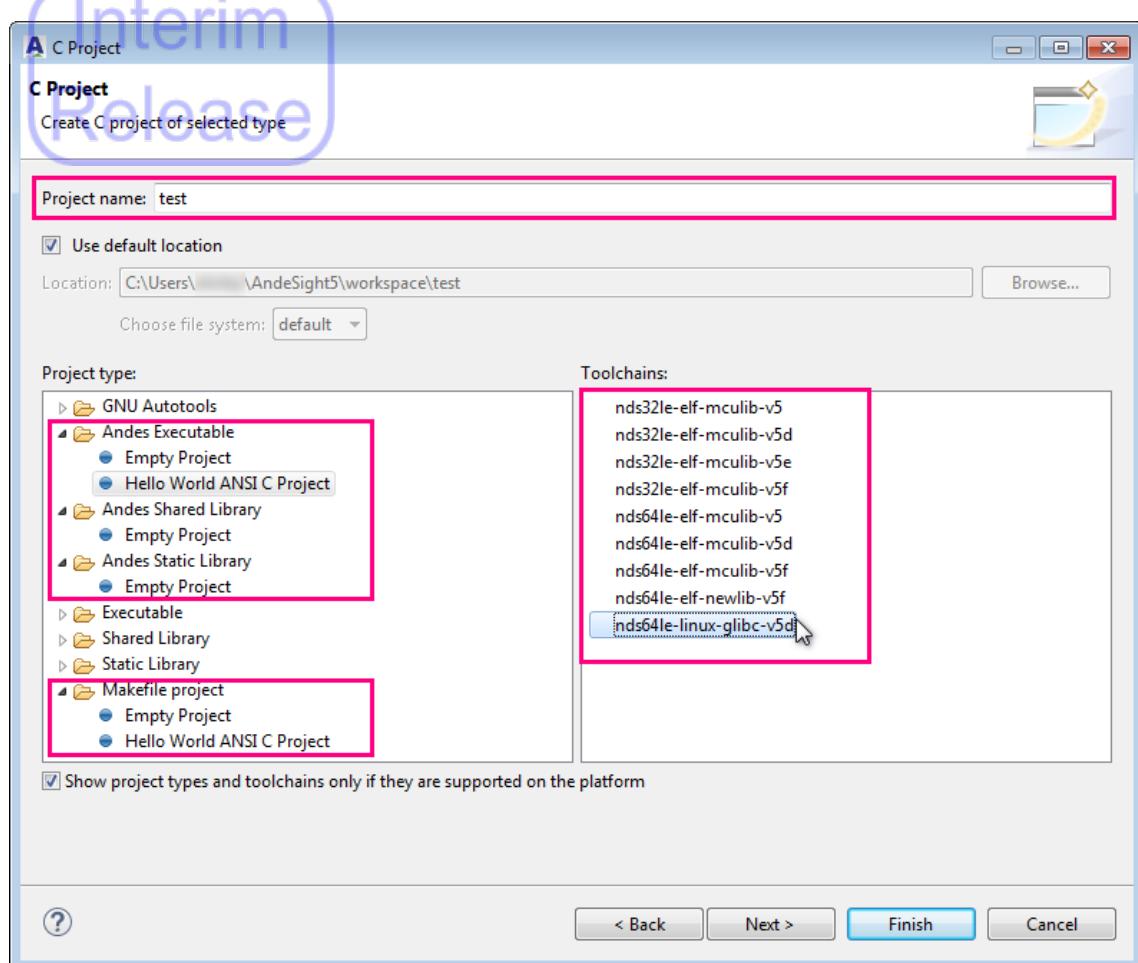


### 2.1.1.2 Creating a new project: Generic target

**Step 1** On the AndeSight main menu, select “File > Project...”. At the **New Project** prompt, double-click “C/C++” to expand the folder and then select between “C Project” and “C++ Project.”



**Step 2** At the invoked prompt, enter a project name and select a project type under “Andes Executable”, “Andes Shared Library”, “Andes Static Library” or “Makefile project”. Then, specify one or multiple toolchains for the project and click on “Next”.

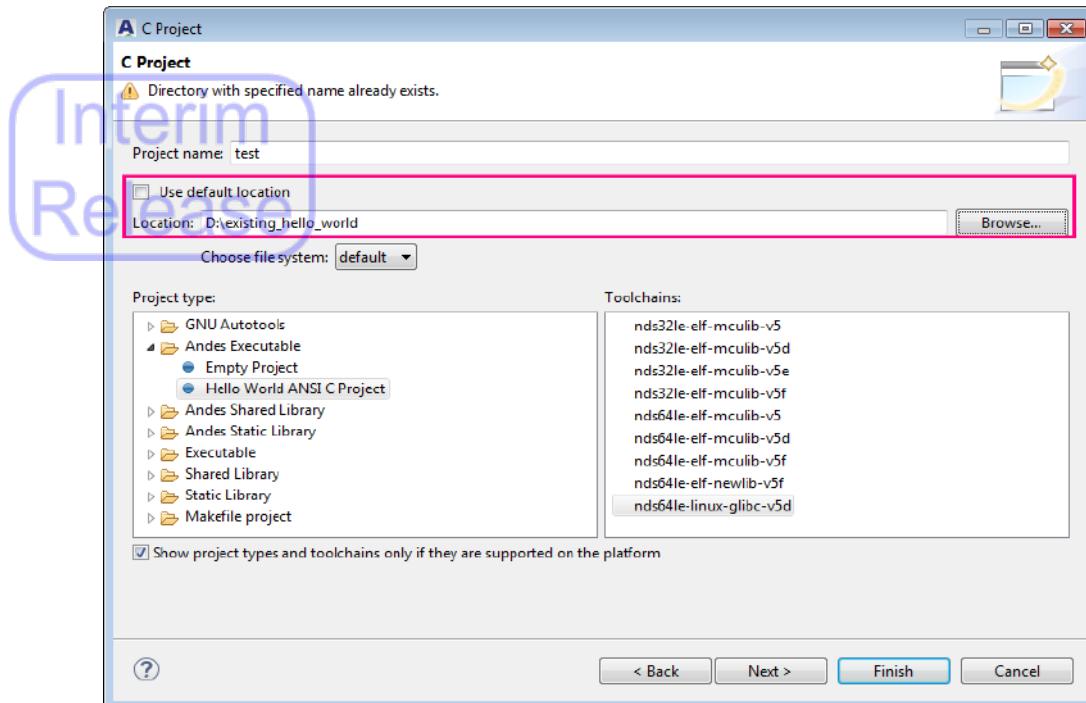



---

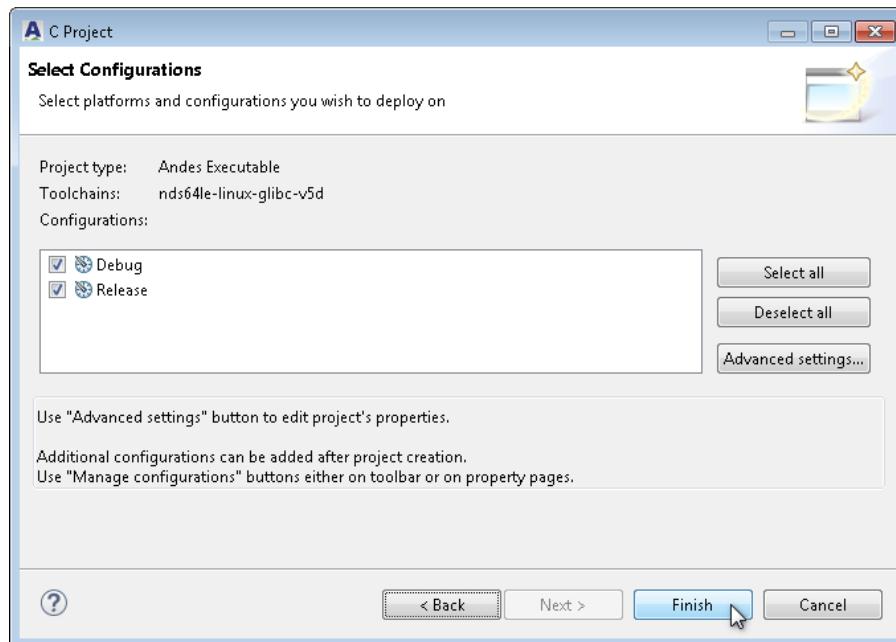
#### NOTE

1. Some characters are not allowed for project names. Please ensure your project names contain characters only from the following set:  
A-Z a-z 0-9 ! @ % ^ [ ] . - + , ~ \_.
2. To use an existing folder for the project, adopt one of the following method and give your new project a name matching the name on the existing folder:
  - Import the folder by selecting “File > Import” from the AndeSight main menu.

- Specify the location of the desired folder on this dialog box after deselecting the “Use default location” option.



**Step 3** Select the configuration(s) for the project and click “Finish” to create the project.

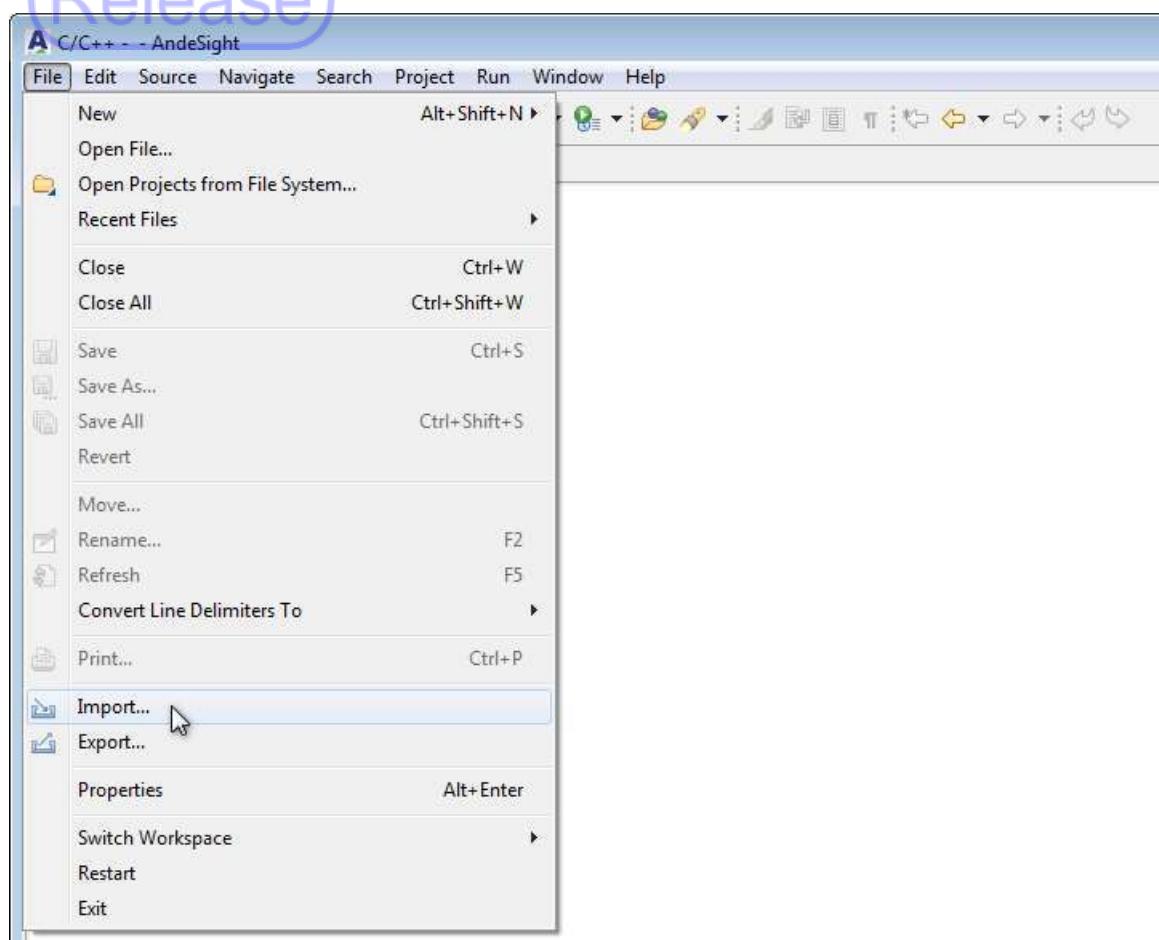


### 2.1.1.3 Importing an existing project

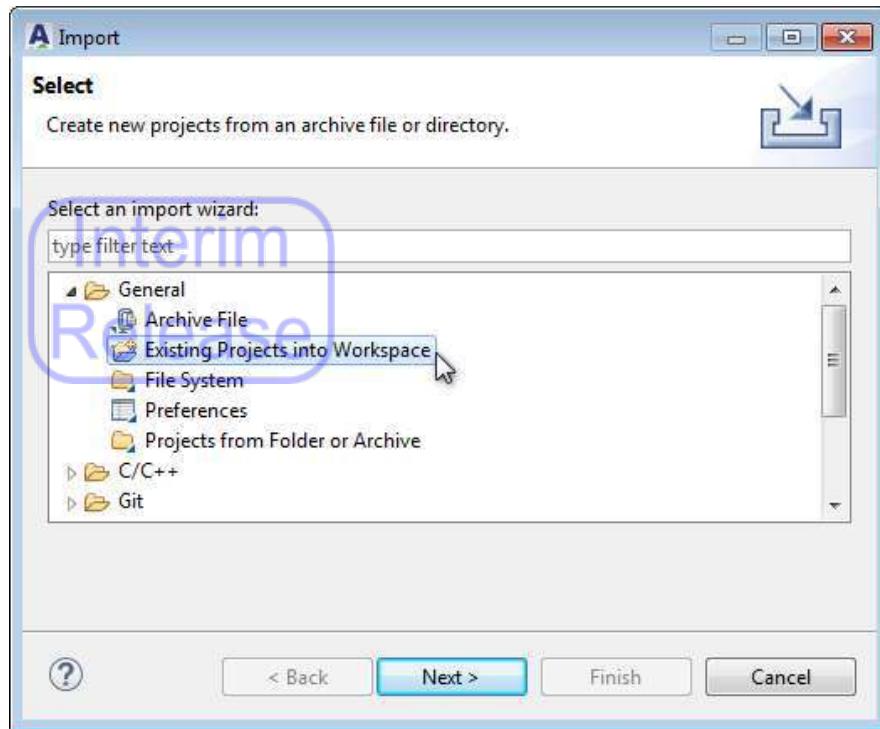
You can import an existing AndeSight project to the AndeSight IDE and added it to the current workspace following the standard import procedure or using drag-and-drop.

#### Standard import procedure

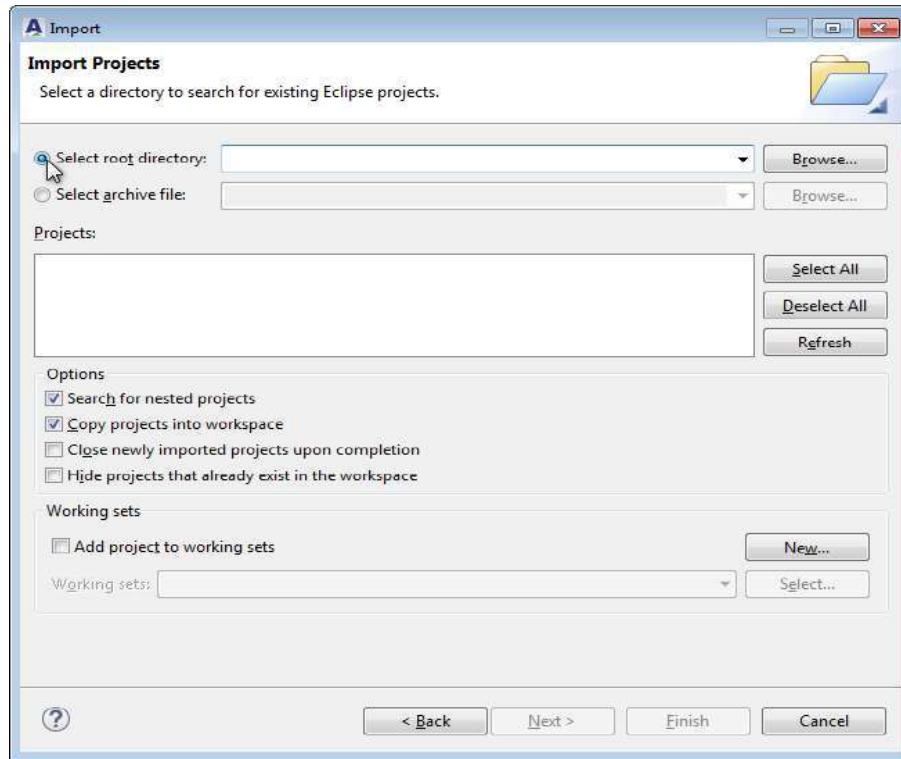
**Step 1** On the AndeSight main menu, select “File > Import...”.



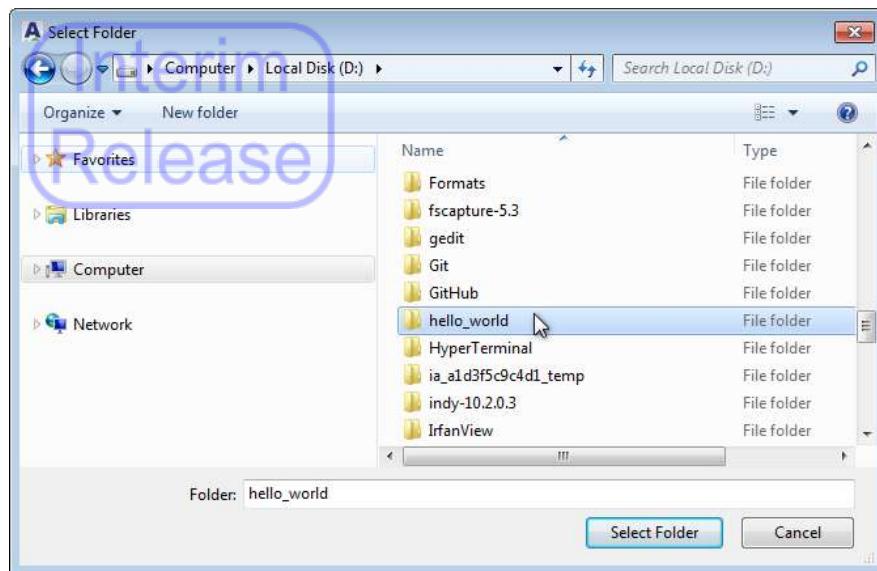
**Step 2** At the **Import** prompt, select “General > Existing Projects into Workspace” and click “Next.”



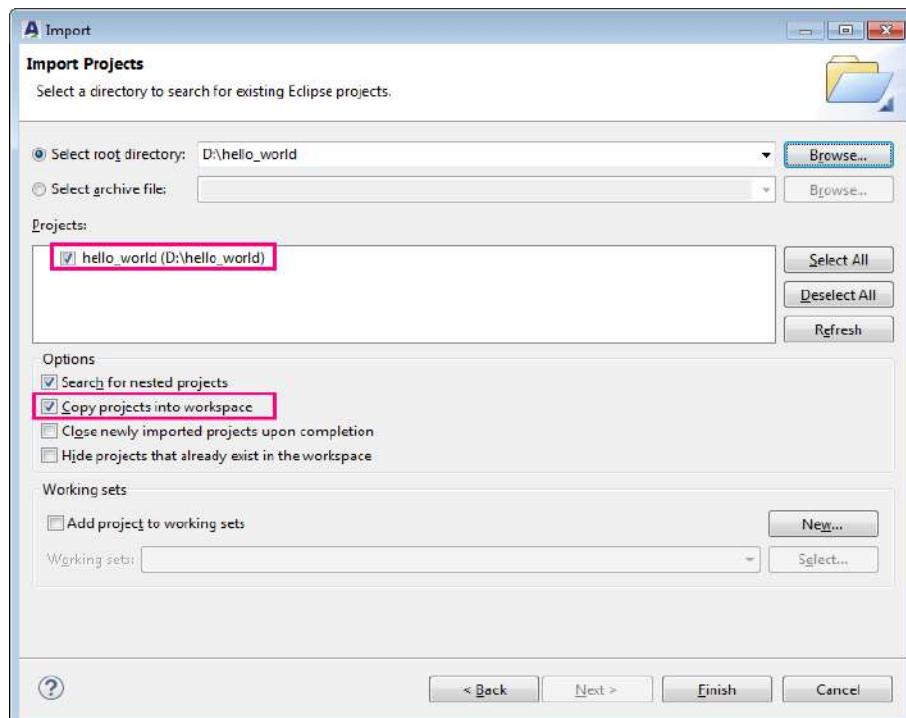
**Step 3** Select the “Select root directory” radio button to import a project from your file system or “Select archive file” to import a compressed file. Click the “Browse...” button.



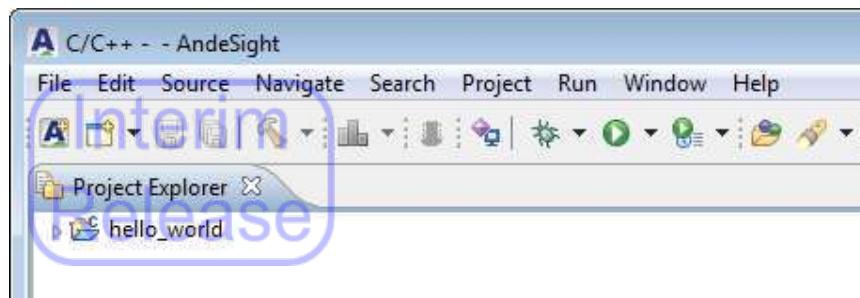
**Step 4** In the invoked dialog, select the directory or the compressed file of the desired project first and click the button “Select Folder” or “Open” to import.



**Step 5** On the **Import Projects** dialog, select the project of interest, check the “Copy projects into workspace” option and click “Finish”. Note that for a project in a compressed file, the “Copy projects into workspace” option is required and thus already selected.



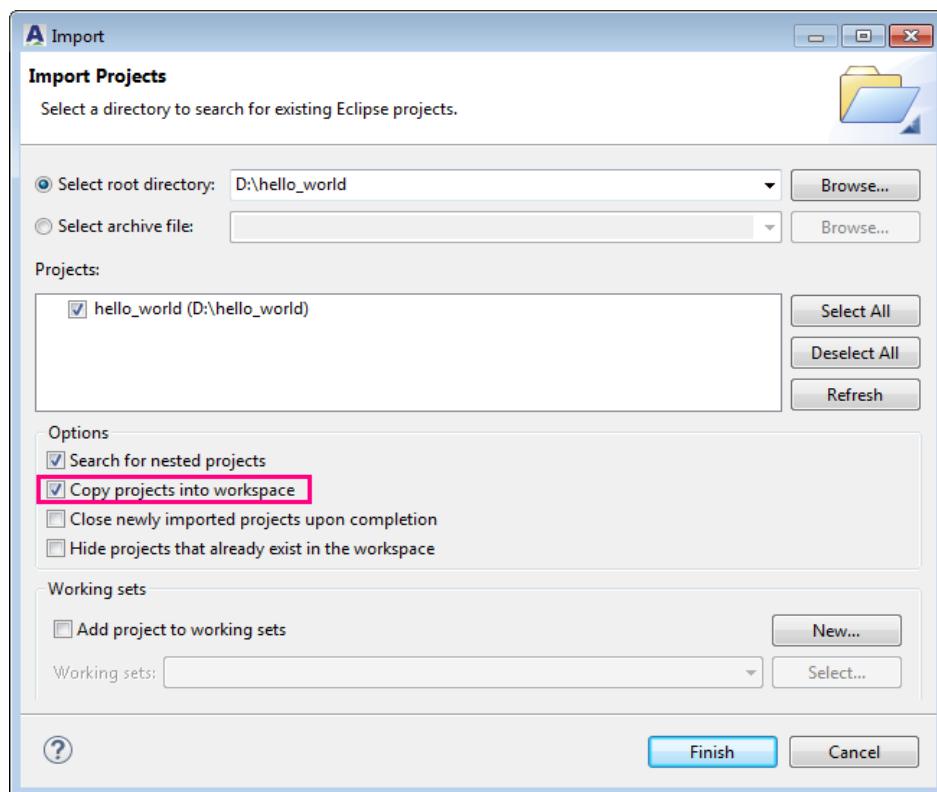
**Step 6** The AndeSight IDE shows the imported project in the **Project Explorer** view.



### Drag-and-drop

**Step 1** Drag and drop the project directory or the compressed file of the project (e.g. \*.tgz) to the **Project Explorer** view of the IDE.

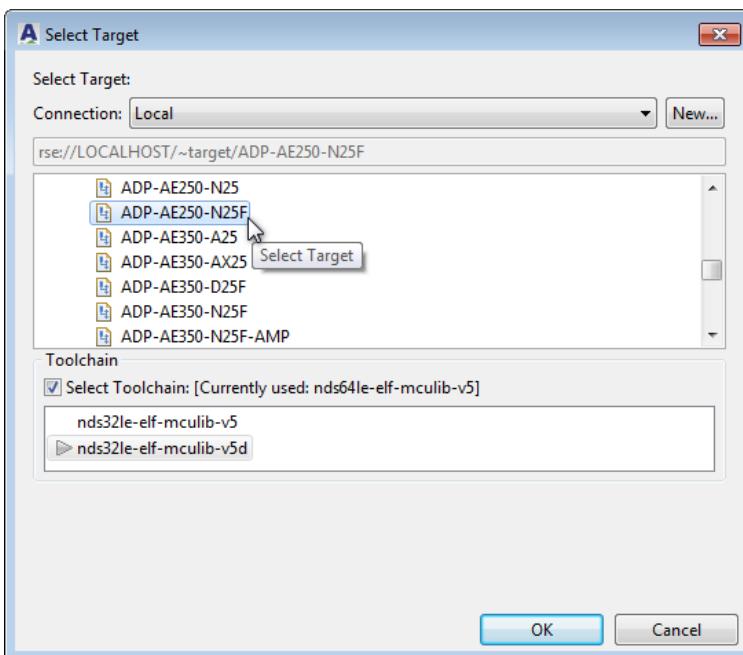
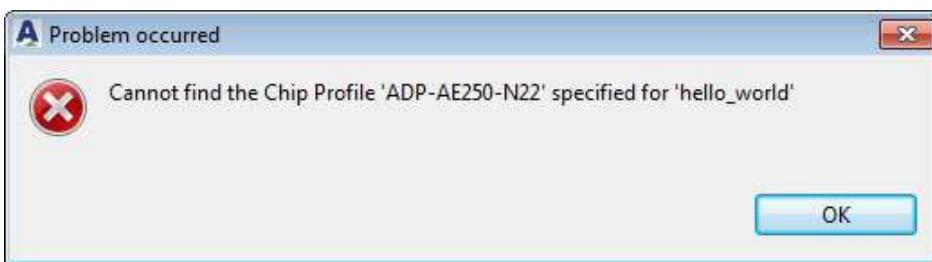
**Step 2** In the popped out **Import wizard**, select the “Copy projects into workspace” option and click “Finish”. Note that for a project in a compressed file, the “Copy projects into workspace” option is required and thus already selected.



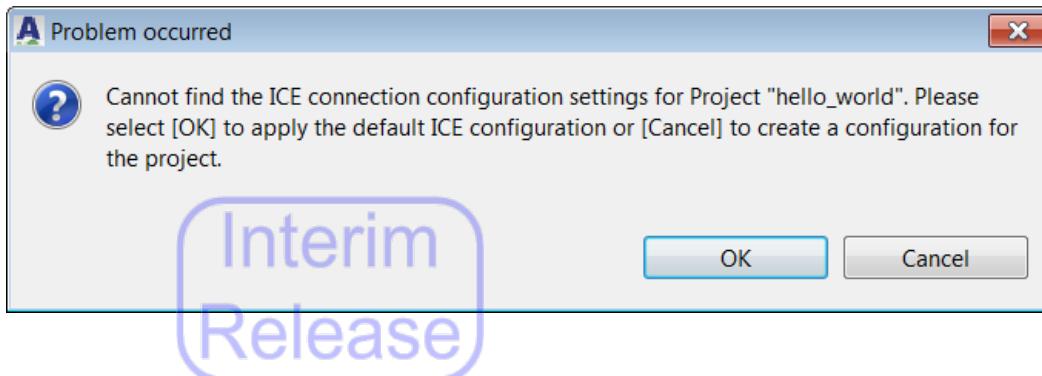
**Step 3** The AndeSight IDE shows the imported project in the **Project Explorer** view.

### Automatic checking for imported projects

To ensure that the imported project can be built and run successfully, AndeSight automatically checks the availability of its chip profile, toolchain and connection configuration in the current IDE. If the chip profile or toolchain specified for the imported project does not exist in the IDE, you will be notified and prompted to select an alternative for the project from the **Select Target** dialog.



If the imported project was created originally with AndeSight v3.2 or later version, its connection configuration will also be checked. If the connection configuration specified for the imported project is not available in the current IDE, you will be prompted to assign the default connection to the project or create a connection configuration for the project.

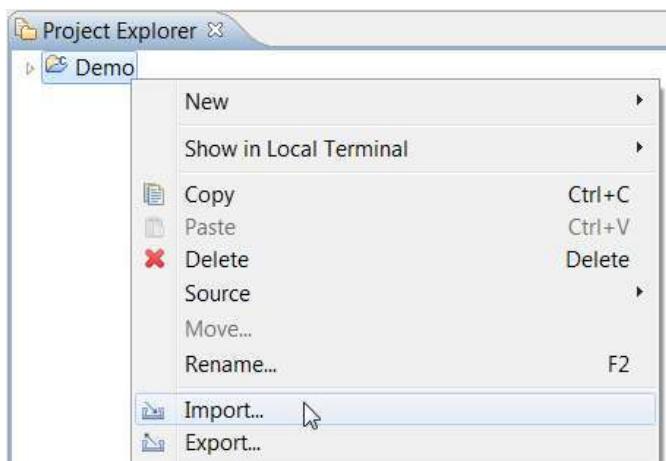


## 2.1.2. Project files

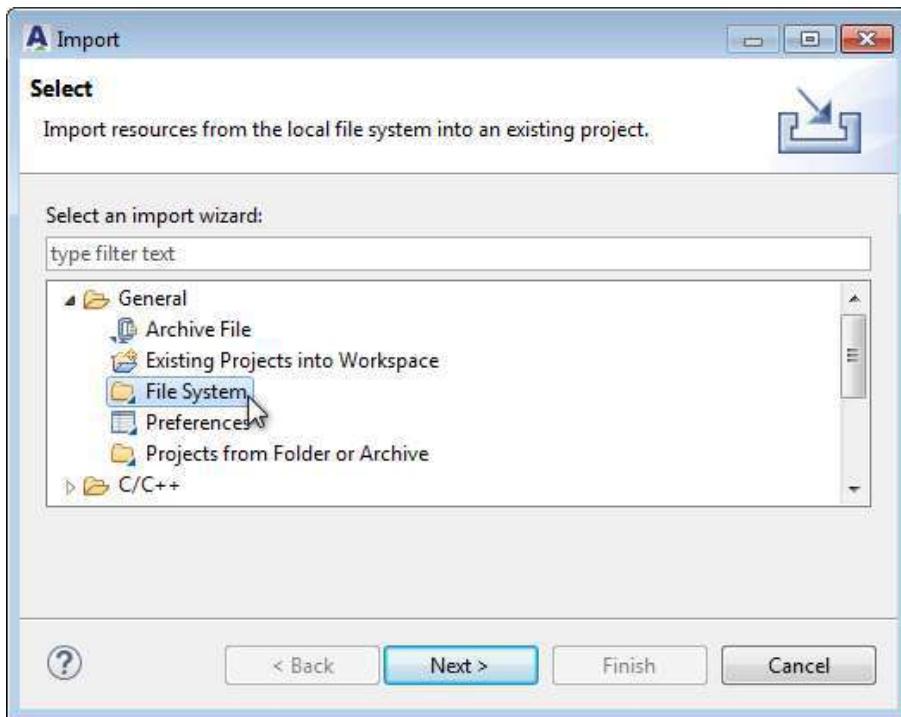
Once a project folder has been created in the workspace, you may import existing files into the project or create a source file for the project.

### 2.1.2.1 Importing files into a project folder

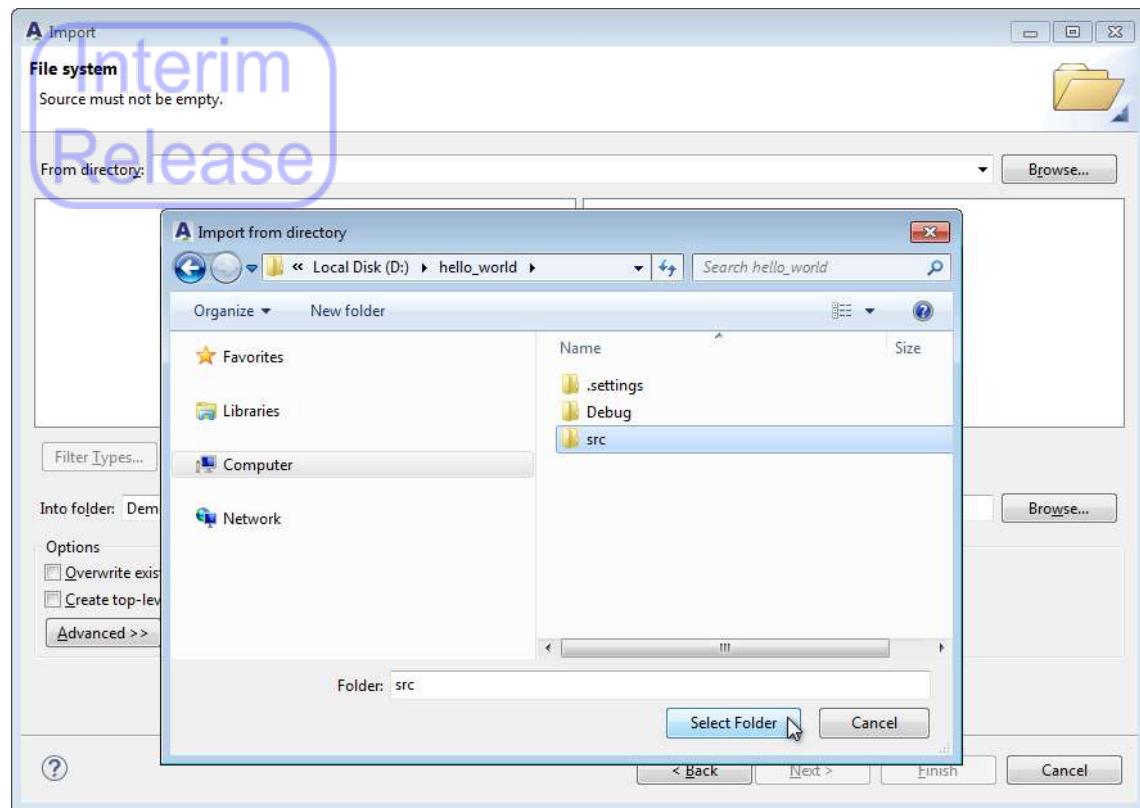
**Step 1** In the **Project Explorer** view, right-click the current project and select “Import...” from the pull down menu.



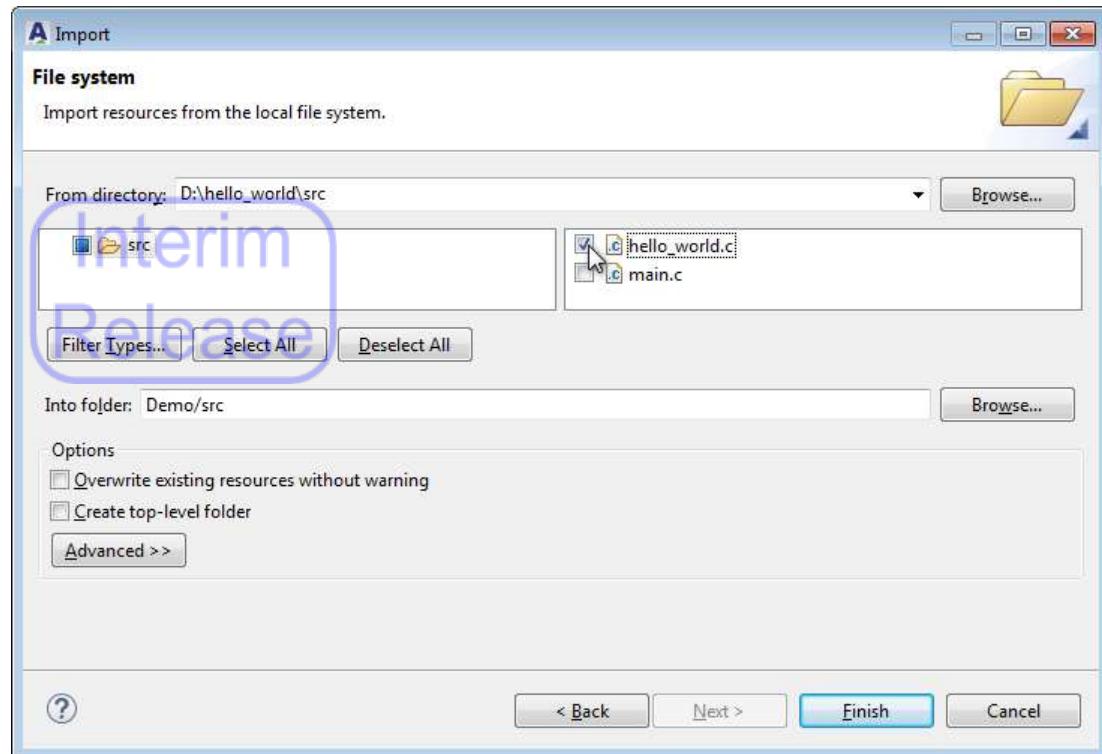
**Step 2** In the **Import** dialog, select “General > File System” and click “Next.”



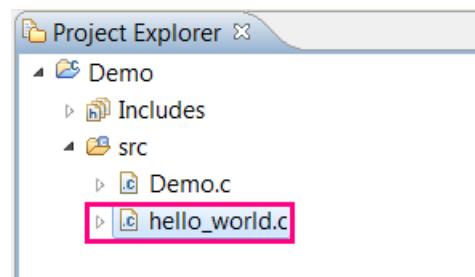
**Step 3** In the **Import** dialog, enter the path to the folder in which the desired file is located, or click “Browse...” to search for the folder using the invoked dialog.



**Step 4** In the middle section of the **Import** dialog, the left column shows the folder selected in the previous step, whereas the right column shows the files contained in that folder. Select the file(s) you wish to import and designate a project folder for them in the “Into folder” field. Click “Finish.”

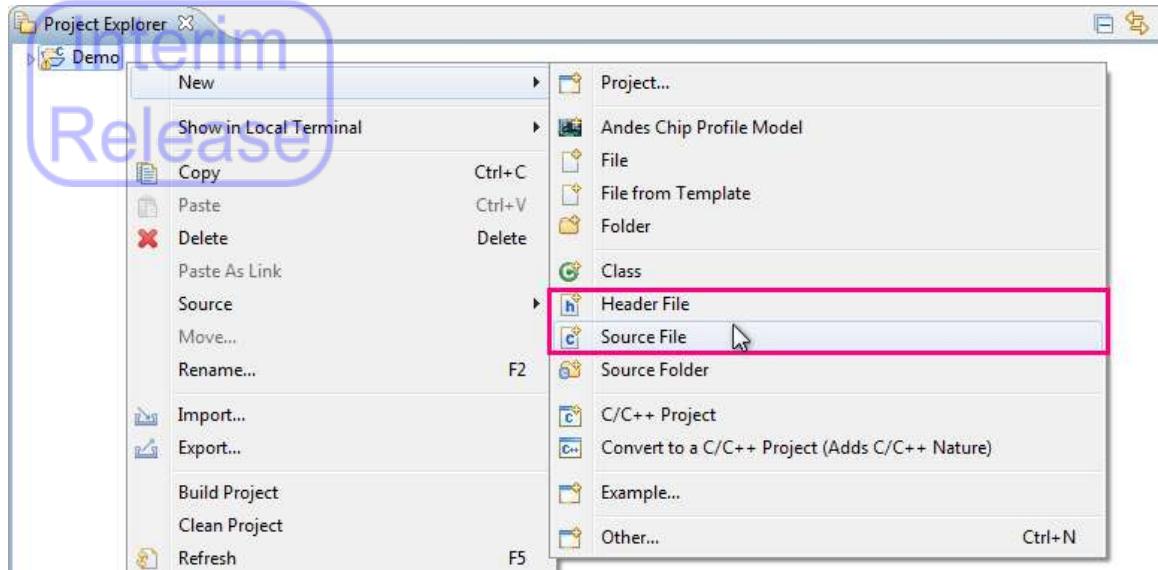


**Step 5** Back in the **Project Explorer** view, the desired file(s) can be found in the current project folder.

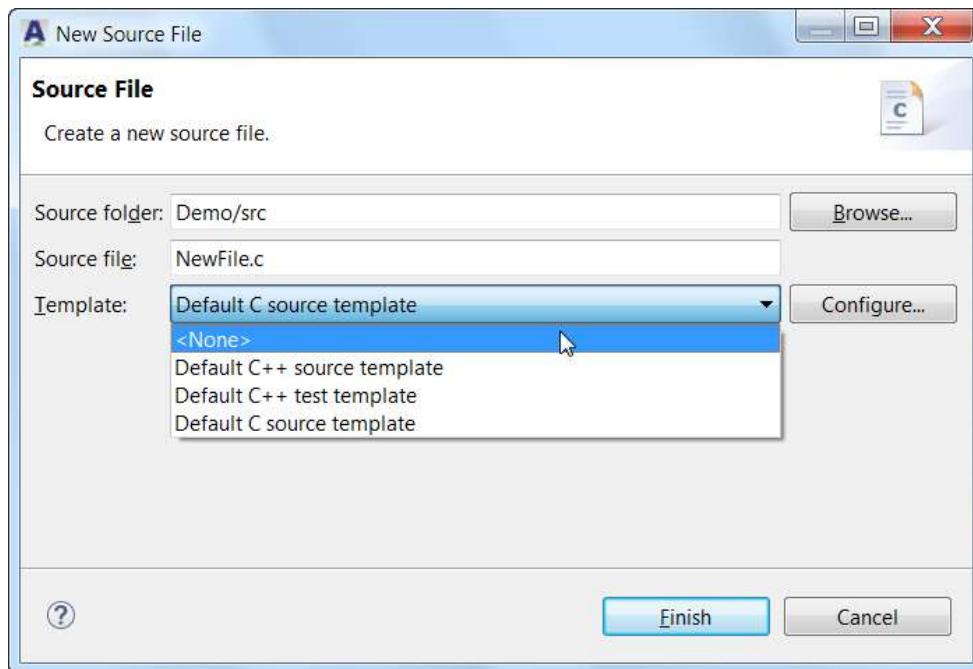


### 2.1.2.2 Creating a source/header file

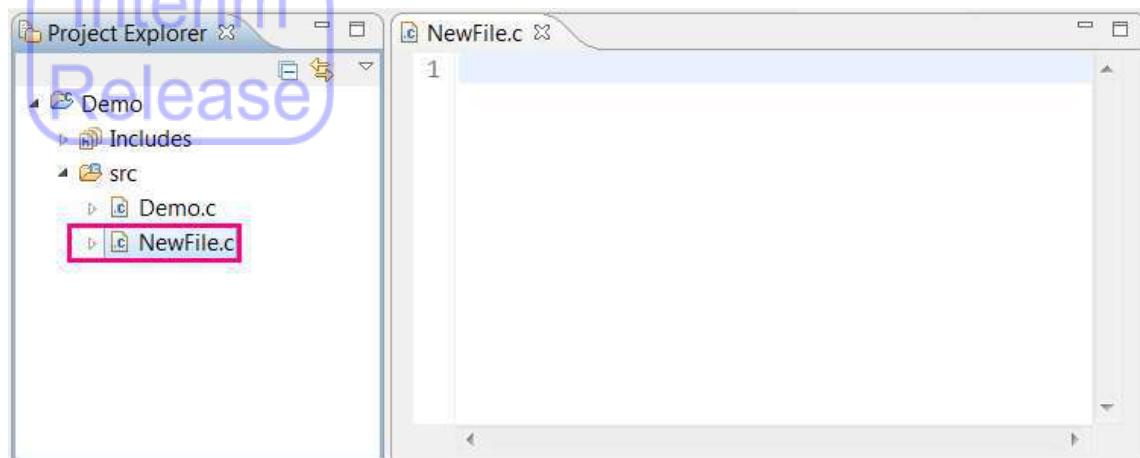
**Step 1** Right-click the current project folder, select “New” and choose “Source File” or “Header File.”



**Step 2** Under the **New Source/Header File** dialog, enter the name of the new source/header file including the extension and select a template from the drop-down list. Click “Finish.”



**Step 3** Back in the **Project Explorer** view, the new source/header file can be found in the current project folder. Clicking on this file automatically opens an editor bearing a tag with the name of the new source file, allowing you to begin coding.



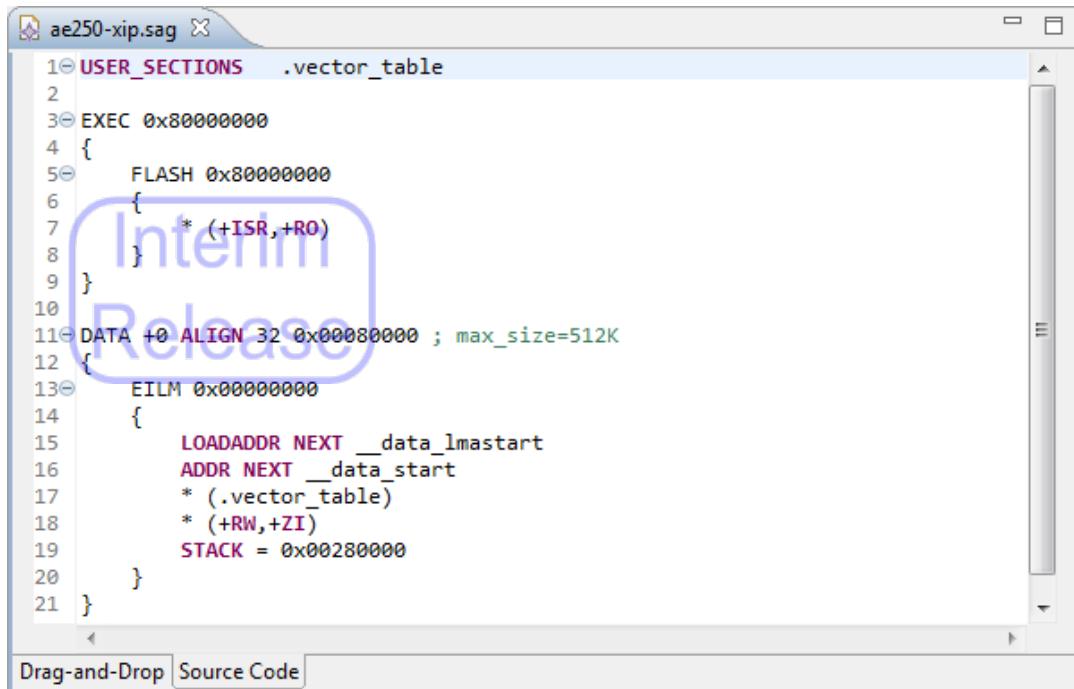
### 2.1.2.3 Creating a SaG file and generating linker script

A SaG (Scattering-and-Gathering) file is written in a concise scripting language defined by Andes and specifying the memory layout of executable images. A SaG file can be converted into GNU ld linker script using the Andes LdSaG utility. For details concerning SaG syntax and the generation of linker script, please refer to the *Andes Programming Guide*.

The SaG editor in AndeSight simplifies the creation of SaG files and the generation of linker scripts. The SaG editor features a drag-and-drop interface to construct SaG files. It also features a text-editing interface for direct SaG input. Both editing interfaces support syntax highlighting. After the SaG file is saved, the results are updated for both interfaces.



Figure 3. Drag-and-drop interface of the SaG editor



```

1 USER_SECTIONS .vector_table
2
3 EXEC 0x80000000
4 {
5   FLASH 0x80000000
6   {
7     * (+ISR,+RO)
8   }
9 }
10
11 DATA +0 ALIGN 32 0x00080000 ; max_size=512K
12 {
13   EILM 0x00000000
14   {
15     LOADADDR NEXT __data_lmastart
16     ADDR NEXT __data_start
17     * (.vector_table)
18     * (+RW,+ZI)
19     STACK = 0x00280000
20   }
21 }

```

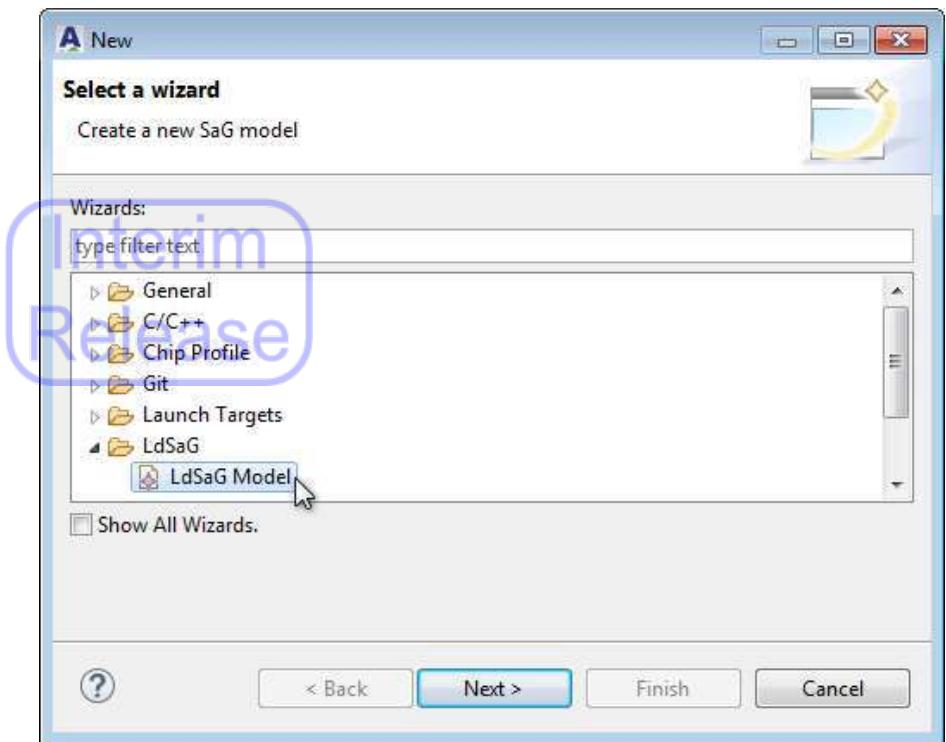
Drag-and-Drop Source Code

Figure 4. Source code editing interface of the SaG editor

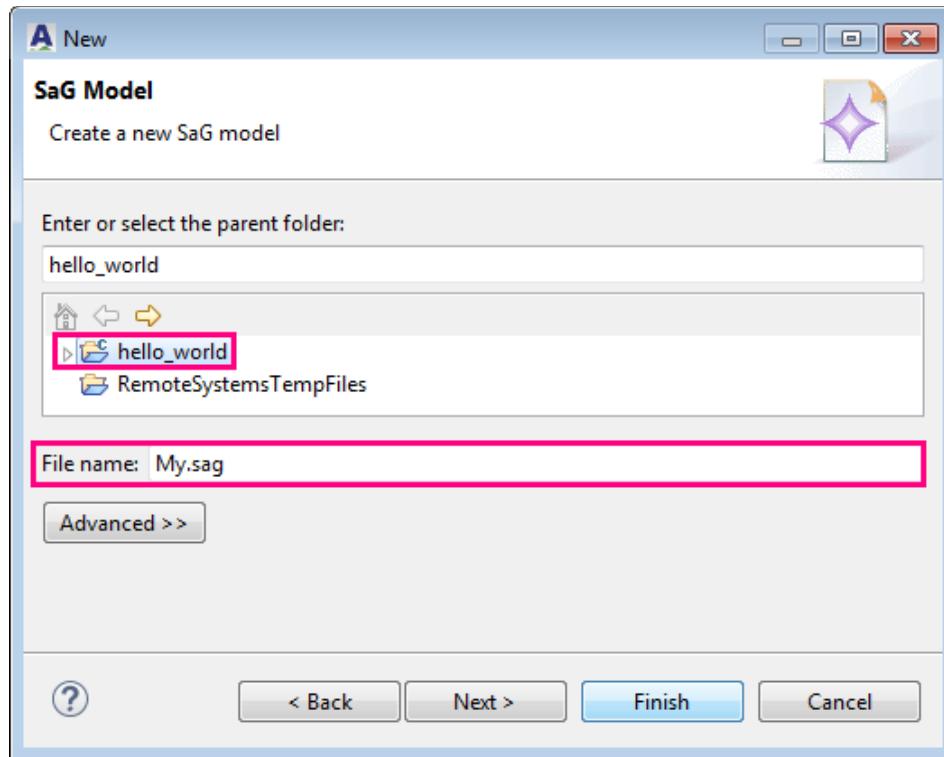
In the following, the drag-and-drop interface is used to illustrate the step-by-step creation and modification of a SaG file in AndeSight.

#### Step 1 Creating or opening a SaG file in the SaG editor:

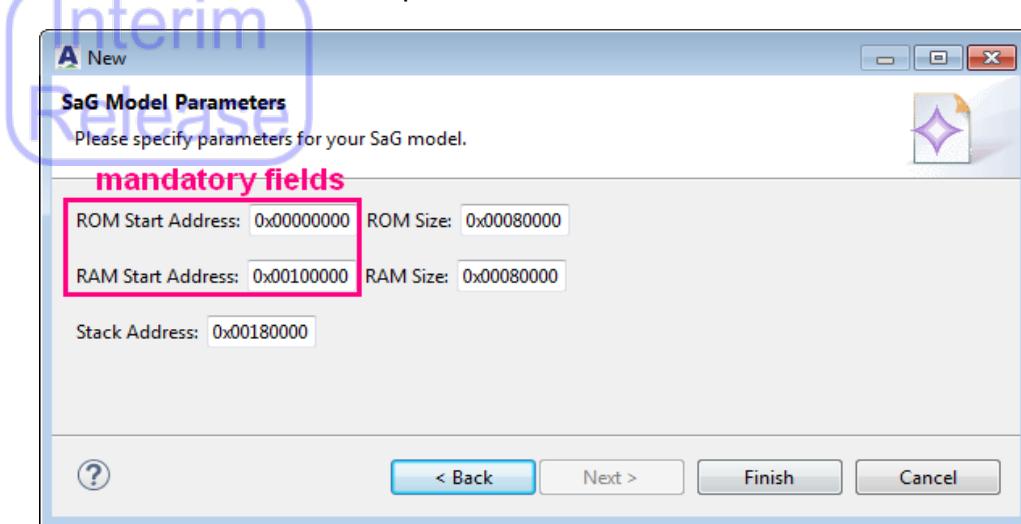
- To modify an existing SaG file, double-click the desired .sag file to open it in the SaG editor.
- A SaG model is used to create a new SaG file. On the AndeSight main menu, select “File > New > Others ...”. At the **Select a Wizard** prompt, select “LdSaG > LdSaG Model” and click “Next.”



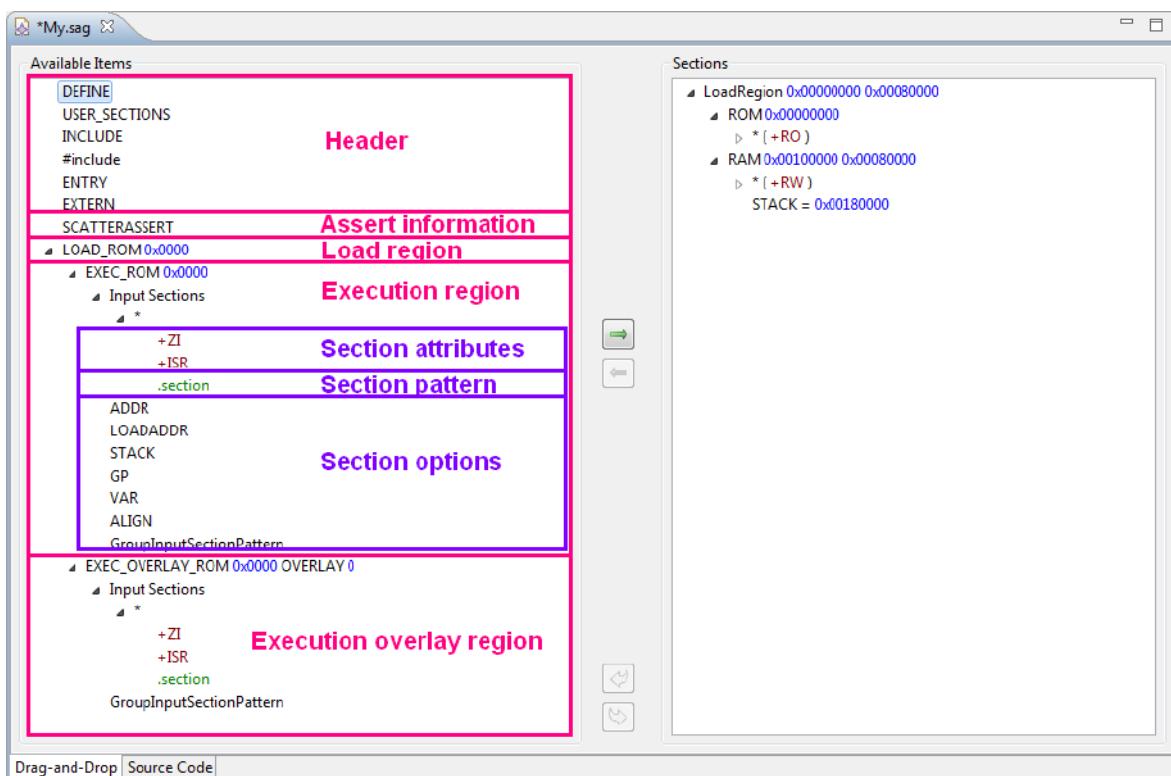
Enter or select a project folder into which the SaG file will be saved, and specify a file name for the SaG file. Click “Next.”



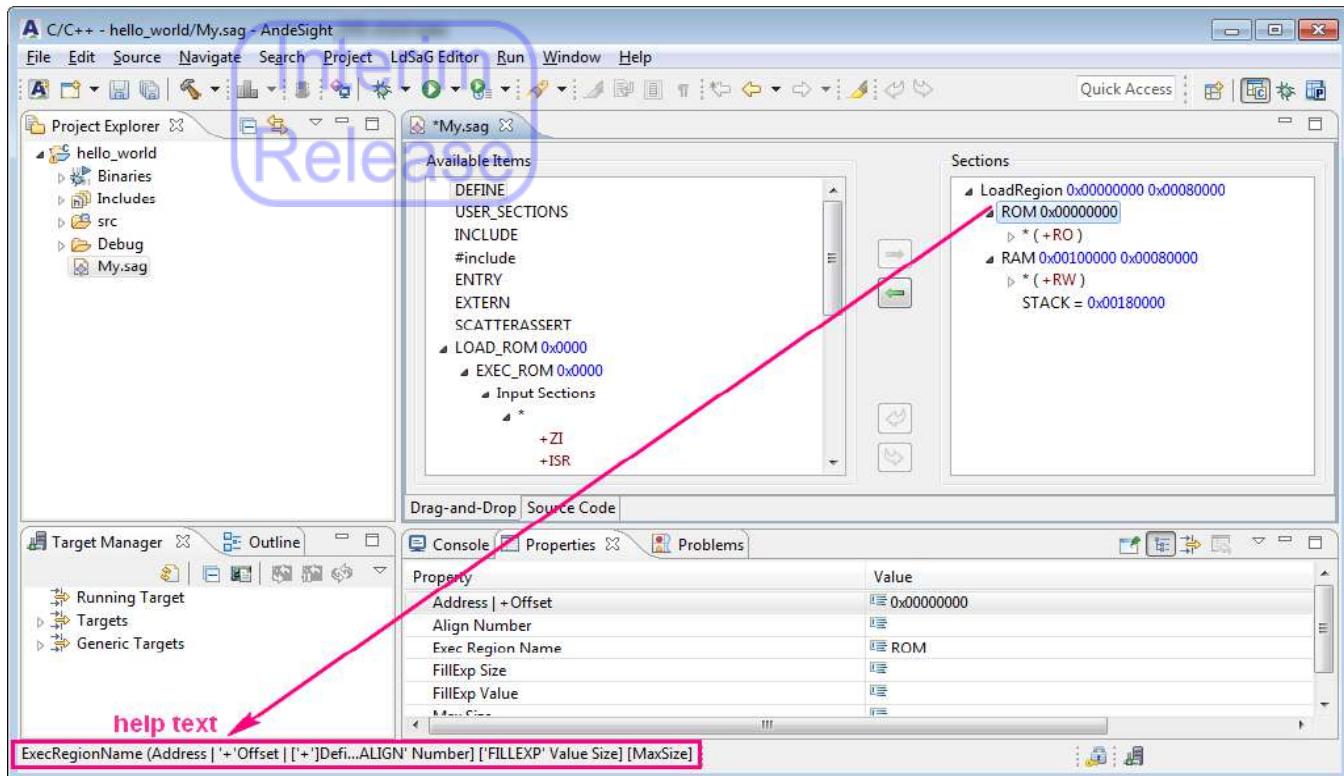
Under **SaG Model Parameters**, specify the parameters for your SaG model and click “Finish” to open the file in the SaG editor. Note that among the SaG model parameters, ROM start address and RAM start address are required fields.



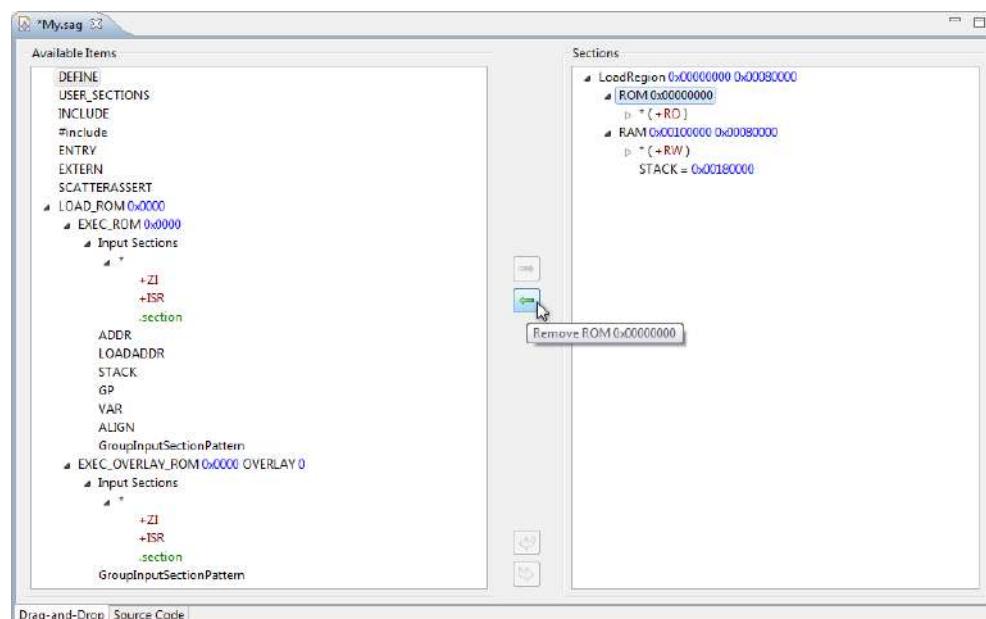
**Step 2** In the SaG editor, “Available Items” on the left lists SaG syntax elements for header, assert information, load region, execution region, and execution overlay region.



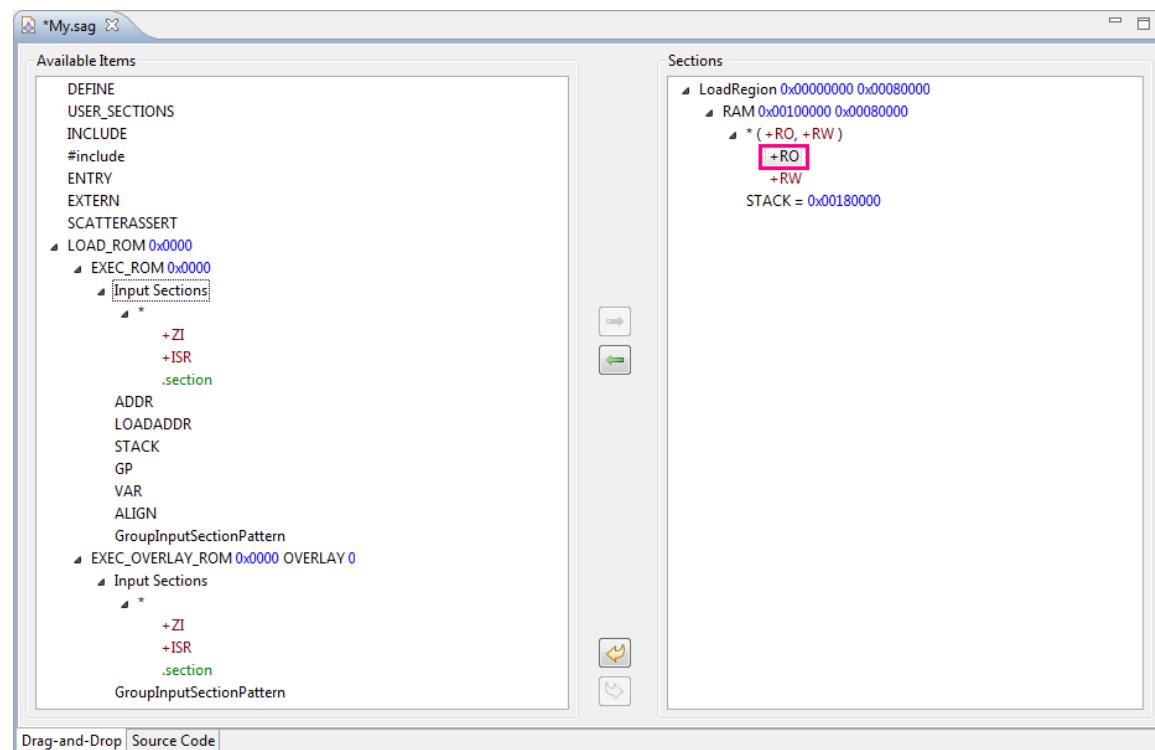
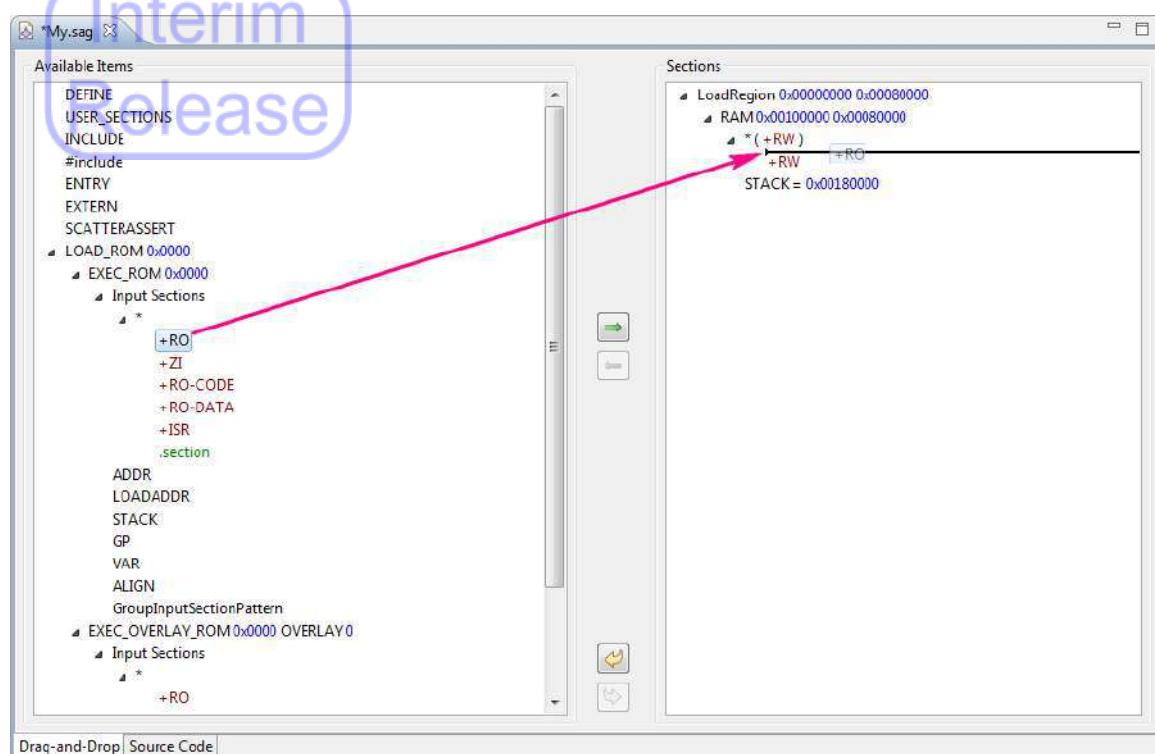
“Sections” on the right displays elements in the SaG file. When selecting an element in the right box, the help text appears at the bottom of AndeSight giving hints concerning syntax construction.



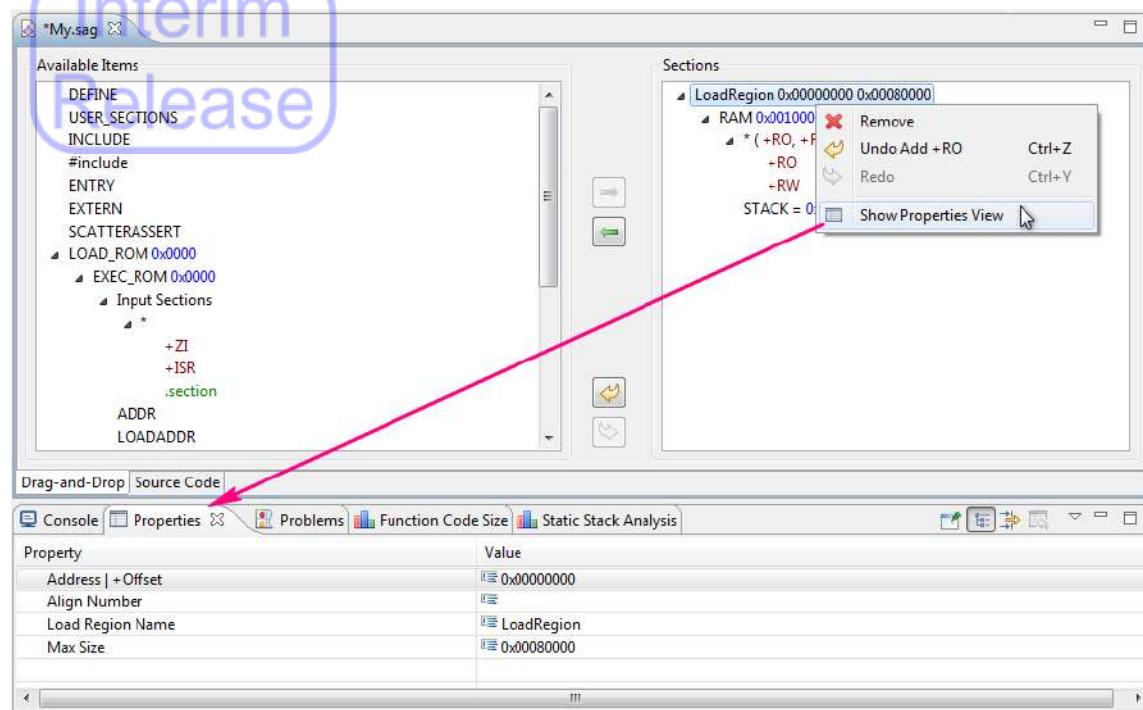
**Step 3** You can remove unwanted elements from your SaG file by selecting them in the right box and clicking  (remove).



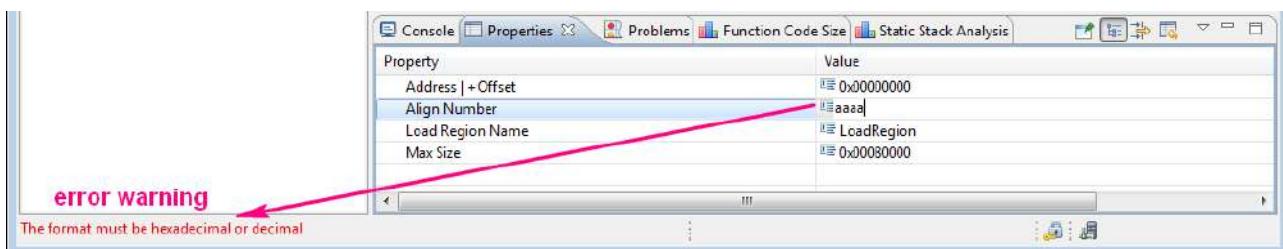
**Step 4** To organize the hierarchy of your SaG file, elements of interest can be dragged from the left box to their desired positions in the right box. Note that components of the execution region and components of the execution overlay region are mutually exclusive.



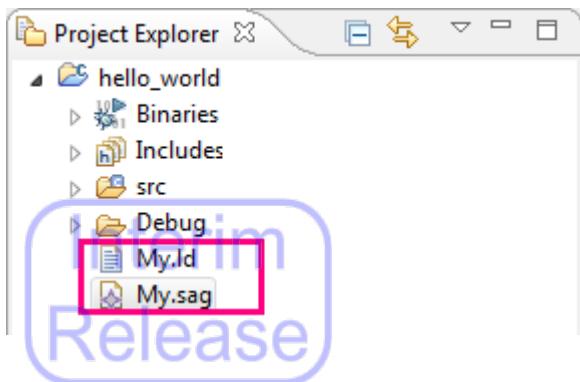
**Step 5** To configure section settings, right-click the desired components in the right box and select “Show Properties View” in its pull-down menu. Then, specify or modify the values in the table cells (Value fields) of the **Properties** view.



The **Properties** view used with the SaG editor includes an error-checking feature, which issues a warning when the wrong format is entered.



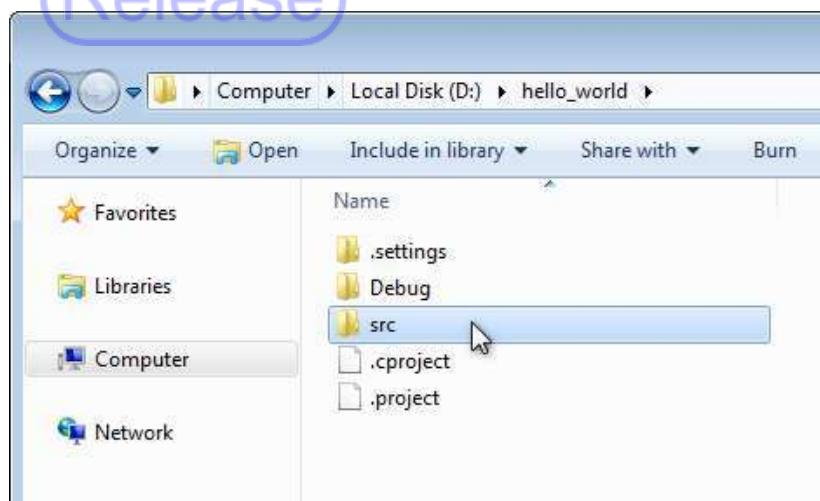
**Step 6** Repeat Steps 3 to 5 to edit the SaG file. Then, press “Ctrl + S” to save the changes. The corresponding linker script is generated with the SaG file within the same folder.



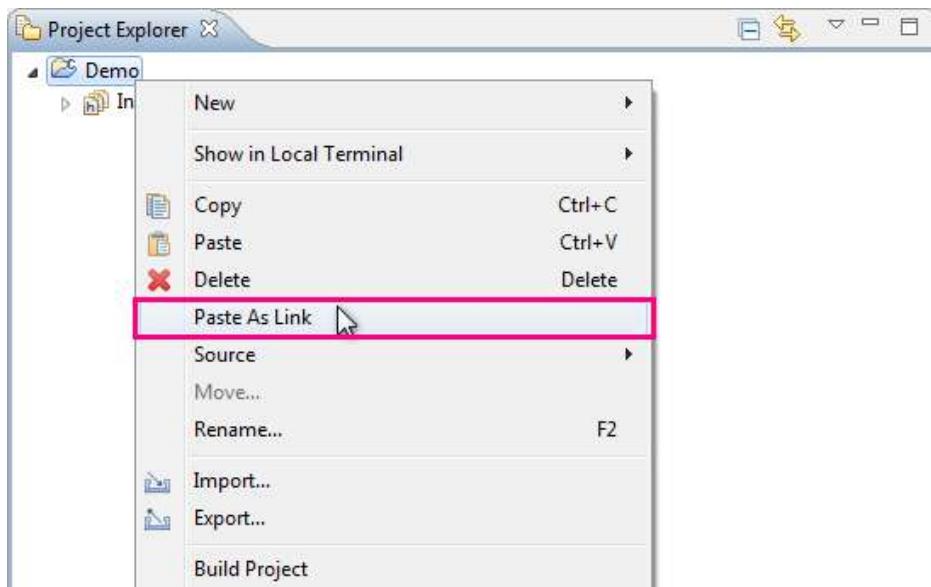
#### 2.1.2.4 Linking to files or folders outside the project

Apart from adding physical files to your AndeSight project, you can include pointers that virtually link to files or folders stored outside the project in the file system.

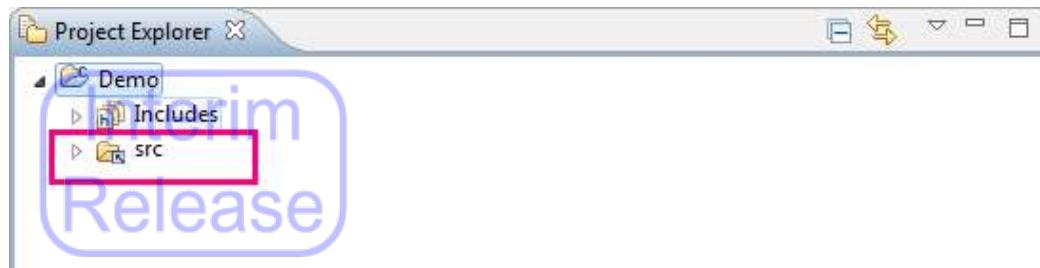
**Step 1** Select a desired file or folder in the file system and copy it by pressing “Ctrl + C”.



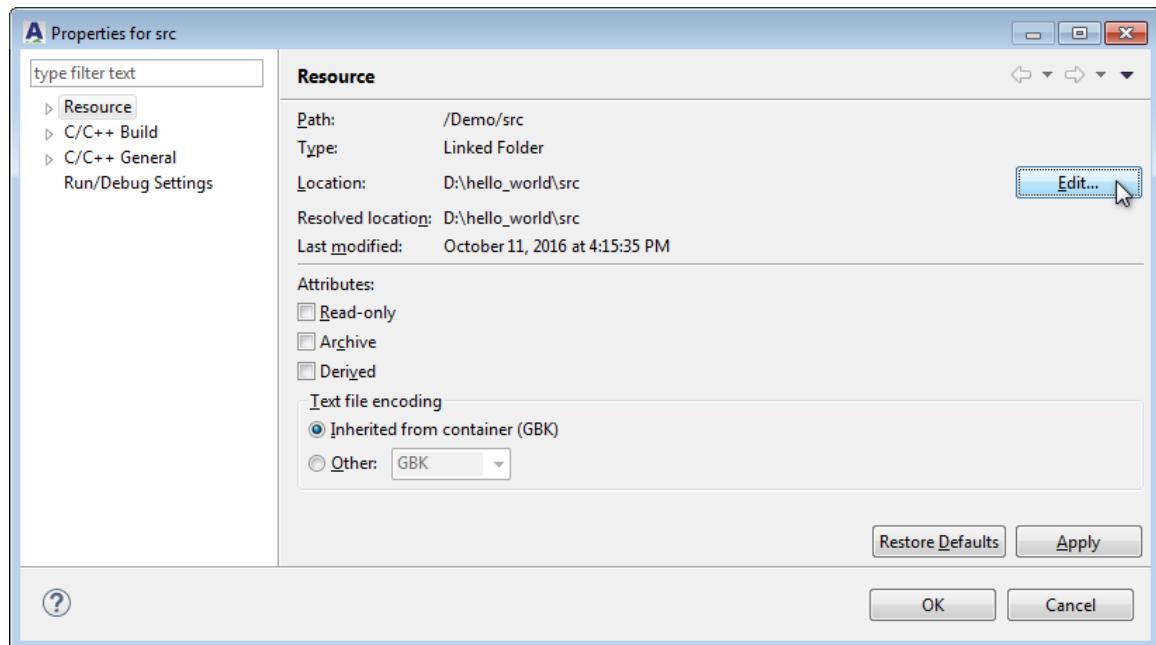
**Step 2** In the **Project Explorer** view of the AndeSight IDE, right-click on the project folder you want to create a link to reference the file or folder you selected in Step 1. Next, select “Paste As link” from the pull-down menu.



**Step 3** A file/folder pointer is created and denoted by a small arrow on its icon.



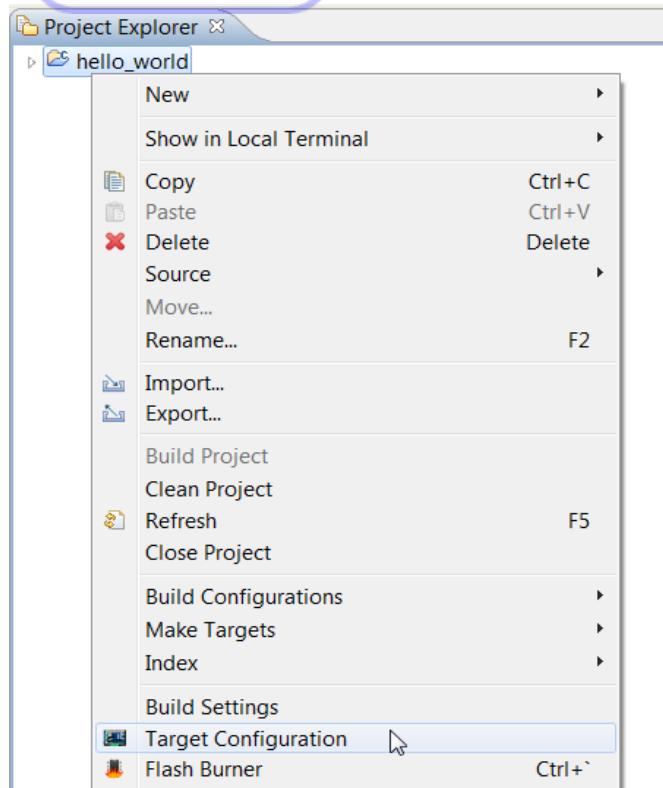
To change the resource that the pointer links to, right-click on the pointer in **Project Explorer**, select “Properties” from the pull-down menu, and modify the setting on the invoked **Properties** dialog.



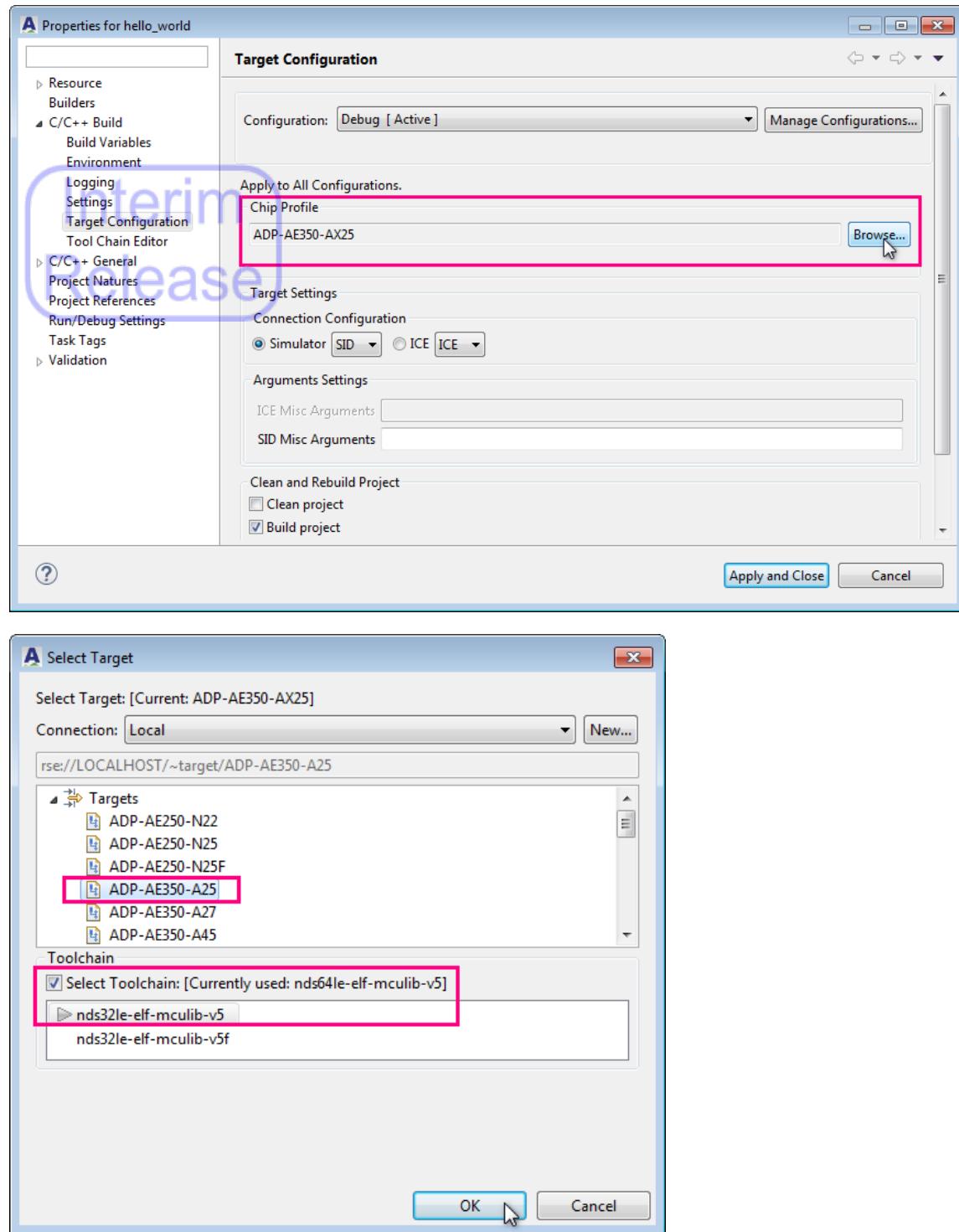
### 2.1.3. Target configuration for projects

The target configuration of a project overwrites target management default settings for the entire workspace (see Section 2.2.4). To configure target settings for a project, follow the steps below:

- Step 1** Right-click your current project folder and select “Target Configuration” in the pull-down menu.



- Step 2** In the **Chip Profile** section of the **Target Configuration** page, click “Browse...” to select a pre-defined target in the invoked dialog. You may also select the toolchain to be used with the pre-defined target by clicking the “Select Toolchain” option and selecting from the list below. Click “OK.”

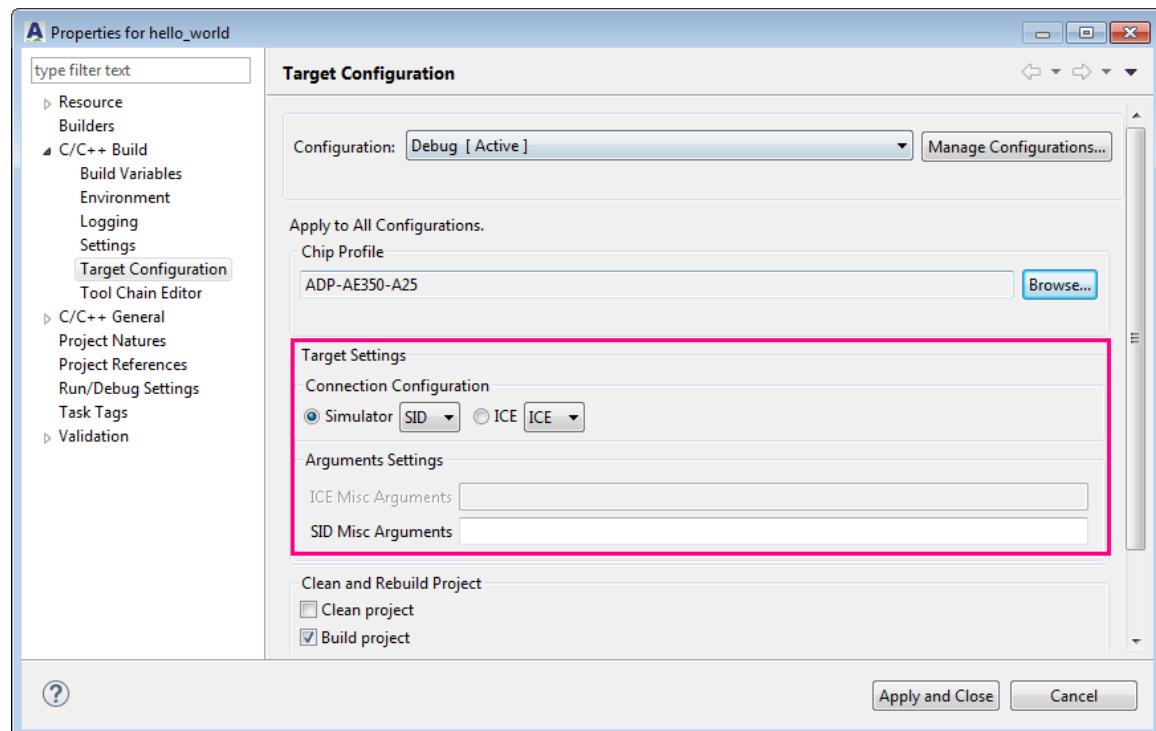


**Step 3** Move down to the **Target Settings** section to specify a connection configuration and enter SID/ICE arguments to be applied for the connection in **Argument Settings** section.

**NOTE**

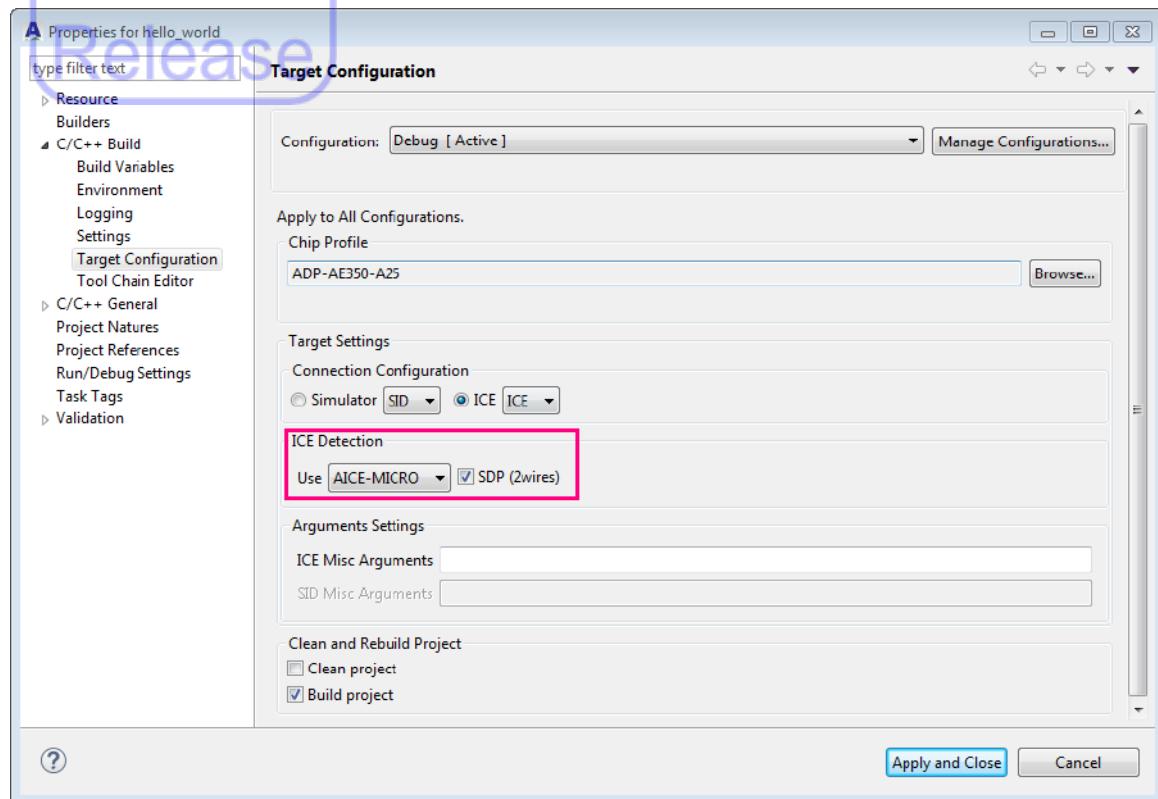
1. The ICEman options “**--port**” and “**-p**” (for specifying a port number for GDB connection) should be avoided since it is not supported by AndeSight.
2. For a multicore system with SMP (Symmetric Multiprocessing) support, make sure to specify the ICEman option “**--smp**”.
3. To use a Corvette-F1 simulator target with fixed-configuration N22 RTL in Andes FreeStart program, specify the “SID” connection configuration first and enter the following in the SID Misc Arguments field:

```
-e "set cpu cpu-option \\\" --conf-is-a-e on --conf-pmp 0
--conf-pfm off --conf-powerbrake off --conf-i cache off
--conf-user-mode off --conf-mul radix4 --conf-btb off \\\""
```



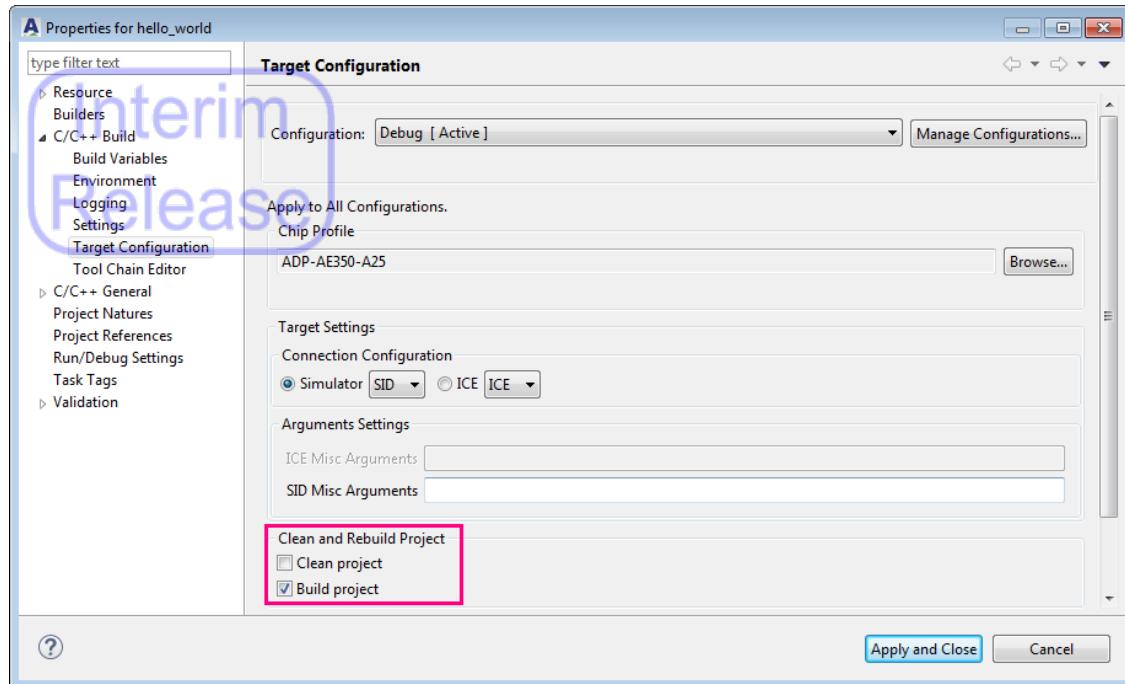
When an ICE configuration is selected for a V5 target, AndeSight automatically detects and displays the connected ICE device(s) in the **ICE Detection** section. ICE devices that can be recognized by

AndeSight include AICE-MINI+ and AICE-MICRO. Other ICE devices for Andes V5 targets such as Olimex FTDI will be listed as “Other” in this section. For AICE-MINI+ or AICE-MICRO that’s connected to a target board using 2-wire debug interface, please manually select the option “SDP (2 wires)” in this section to ensure the connection.



The **Arguments Settings** section has a priority over the **ICE Detection section**. Therefore, make sure the ICE arguments you specify are for the selected ICE device.

**Step 4** In the **Clean and Rebuild Project** section, specify whether to clean or debug the project after completing configuration.



**Step 5** Click “Apply and Close” to complete target configuration for your project.

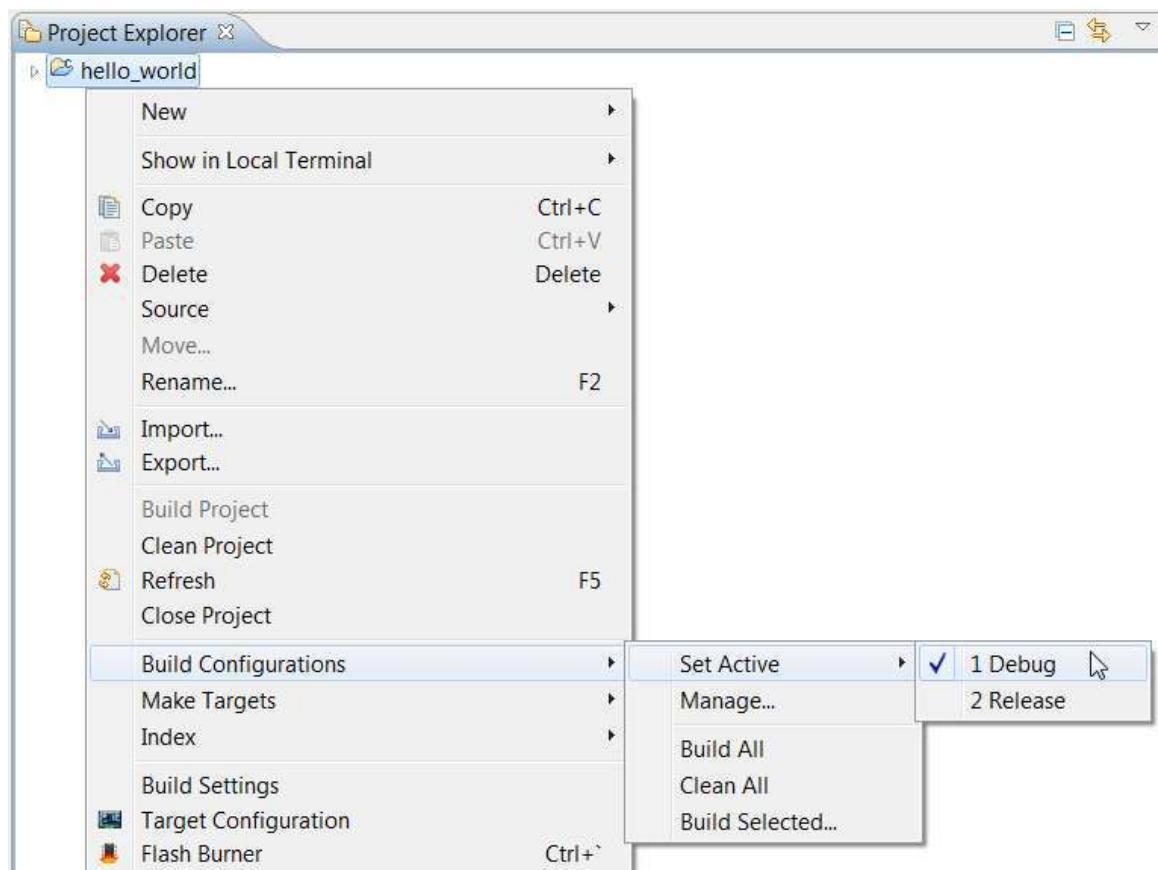
## 2.1.4. Managed build system for Andes projects

The managed build system in AndeSight enables you to make selections concerning the build settings of your project using self-explanatory graphical user interfaces.

### 2.1.4.1 Specifying the build configuration

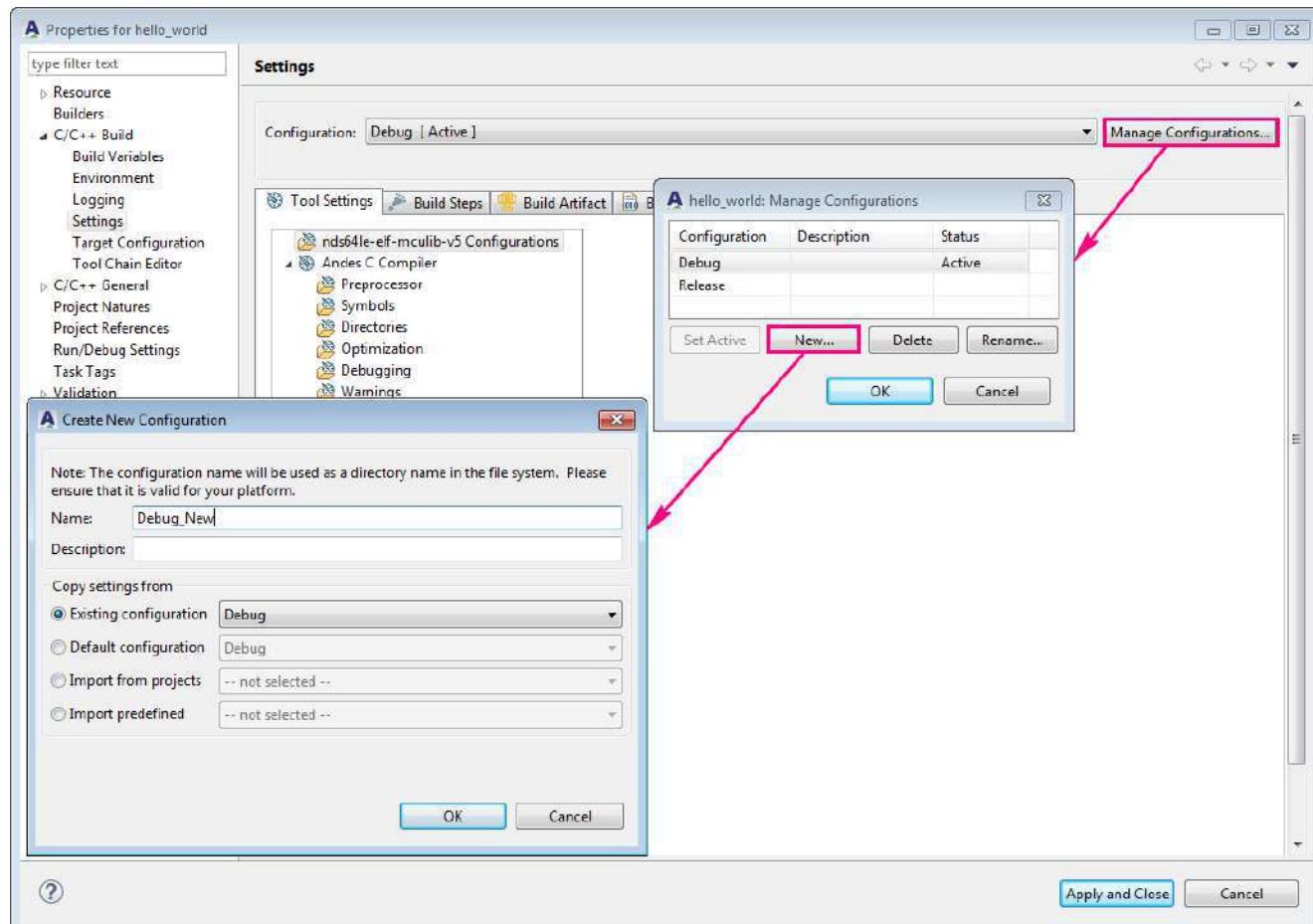
Before utilizing the managed build system, it is necessary to specify a build configuration for your project:

**Step 1** In the **Project Explorer** view, right-click your current project folder and select “Build Configurations > Set Active” in the pull-down menu. Make a selection from available build configurations and set it as active.



**Step 2** To create, delete, or rename the build configuration of an existing project, right-click the project folder and select “Build Settings” from the pull-down menu. The **Settings** page opens. Click the “Manage Configurations...” button.

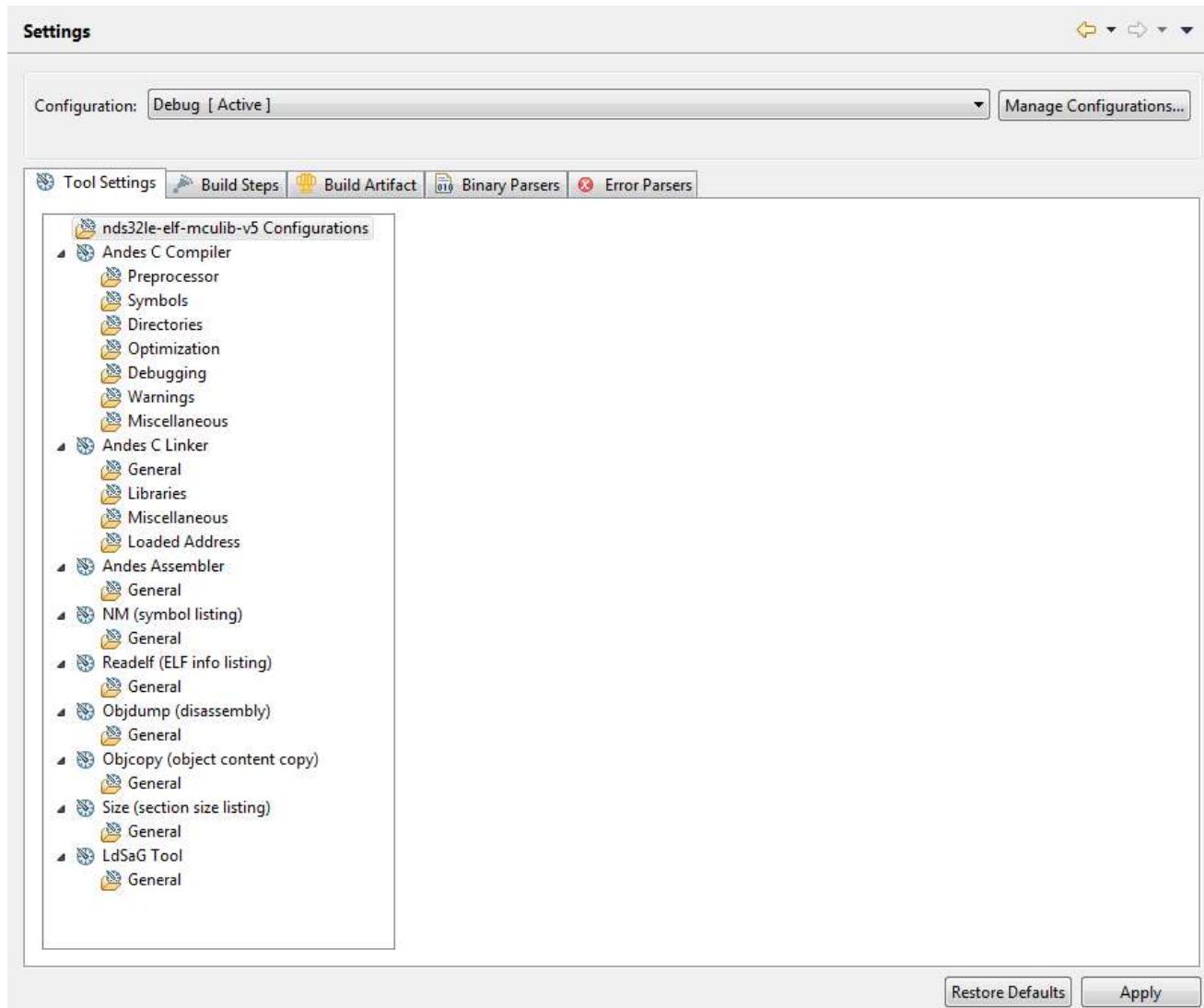
Then, click “New...,” “Delete”, or “Rename...” in the invoked dialog. For example, creating a build configuration would proceed as outlined in the figure below:



### 2.1.4.2 Configuring build settings

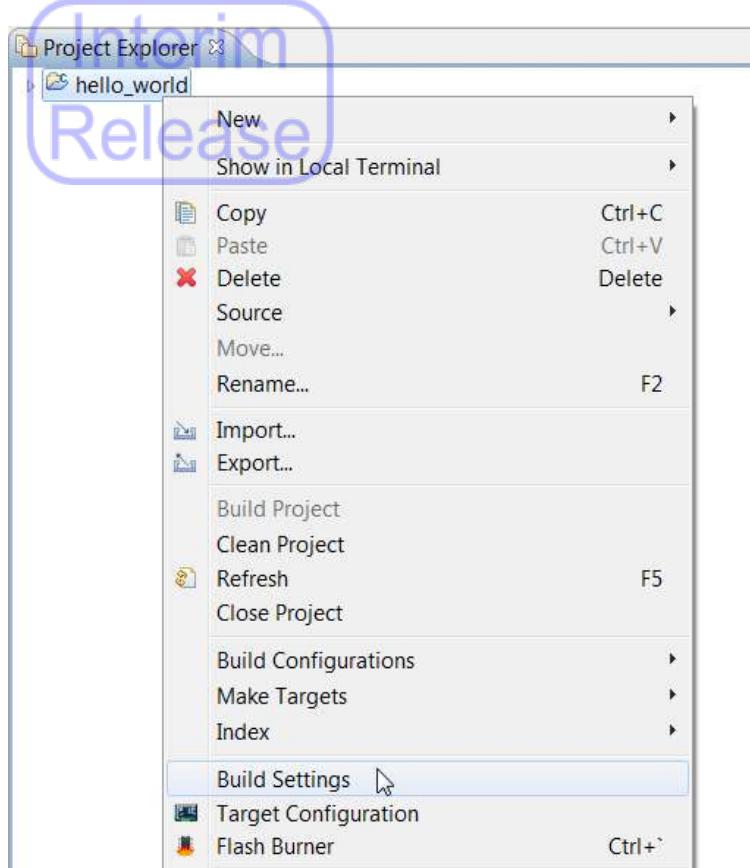
The managed build system includes the following tools and options pertaining to build configurations:

1. Compiler options allow you to set compilation flags, such as optimization level and debug level.
2. Linker options allow you to disable the default library and start file, enable relaxation on branches, specify a program start address, data address, and stack address, and browse for linker scripts.
3. Assembler options allow you to set assembler flags.
4. nm/readelf/objdump/objcopy/size/LdSaG options allow you to enable/disable output and control output content.

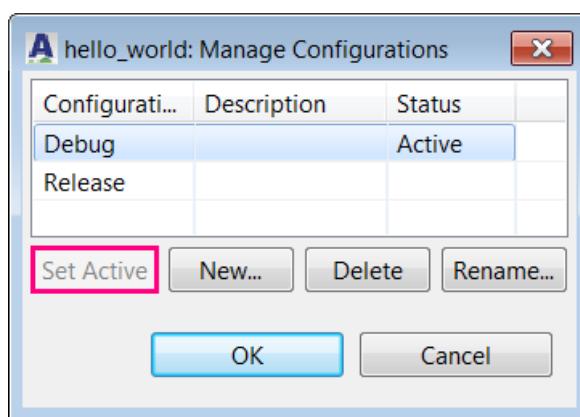
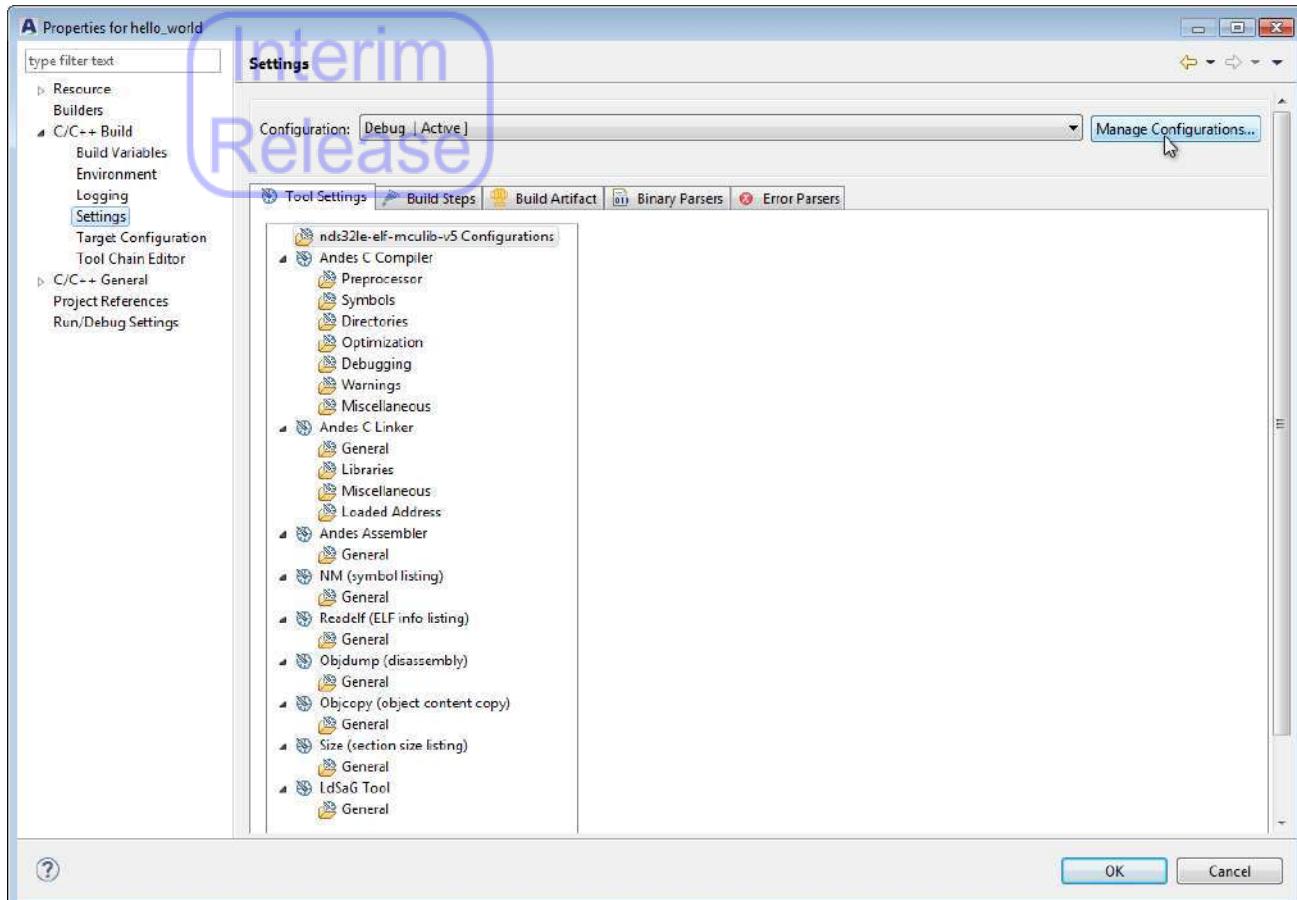


To configure build settings for a project, follow the steps below:

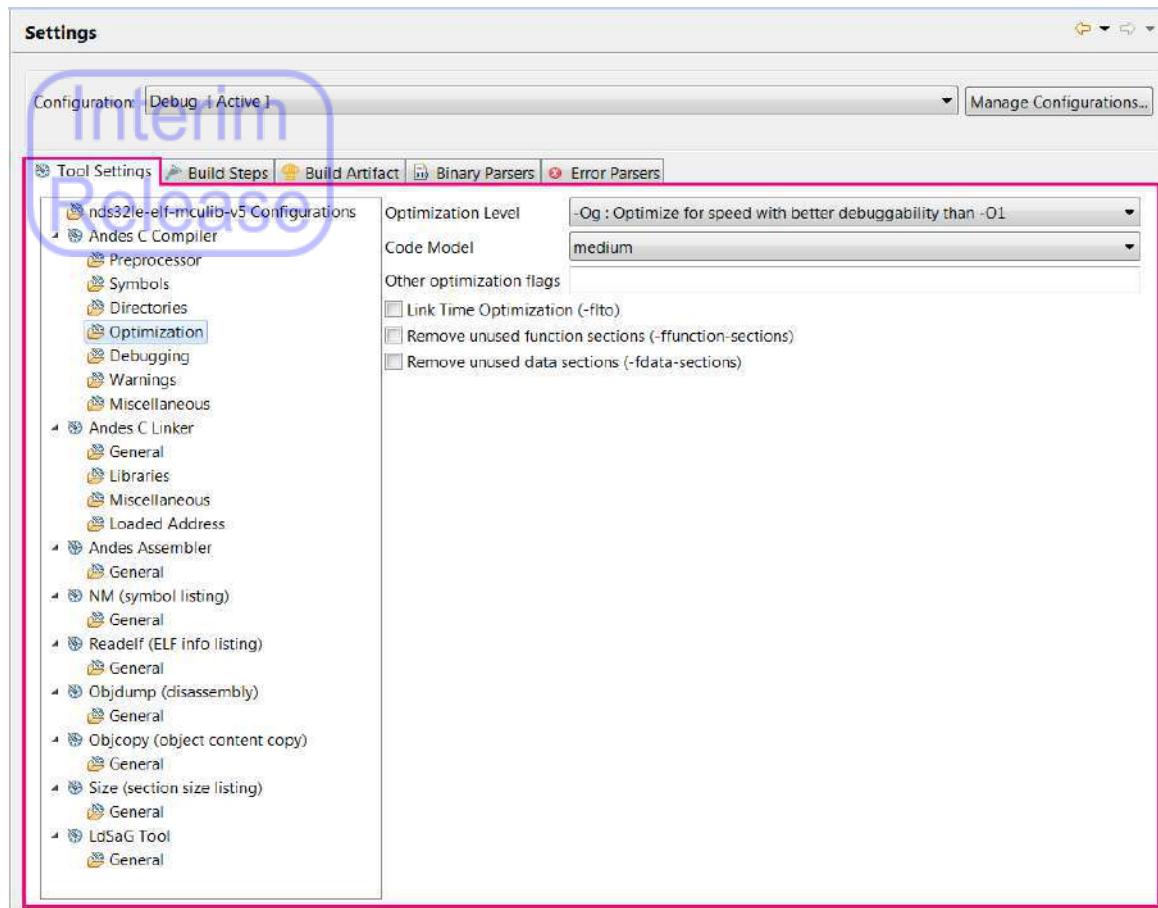
**Step 1** In the **Project Explorer** view, right-click the current project folder and select “Build Settings” from the pull-down menu.



**Step 2** On the **Settings** page, click “Manage Configurations...” to specify a build configuration that meets your project objectives and set it active in the invoked dialog.



**Step 3** Under the **Tool Settings** tab are listed the available build options. Select and make the desired modifications.



### ■ “Andes C Compiler > Optimization” page

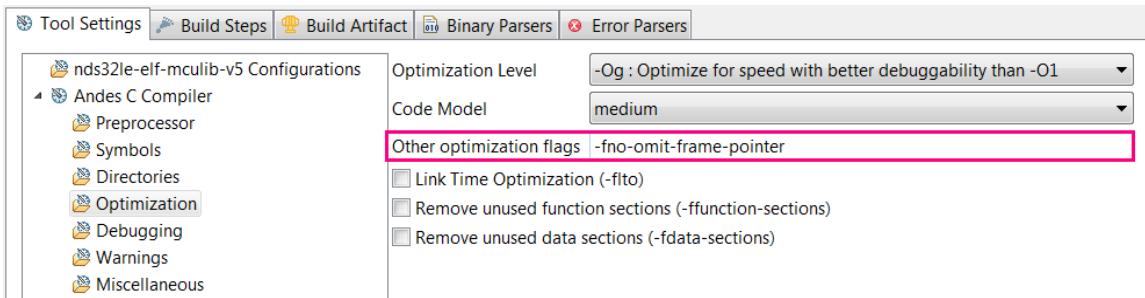
This page allows you to configure optimization levels, optimization options, and the code model for your projects. For details concerning the optimization techniques supported by the Andes compiler, please refer to the *Andes Programming Guide*.

- **Optimization level:** You can select from eight optimization levels (from **-O0** to **-Os3**) to control the trade-off between performance and code size according to your needs.
  
- **Code model:** Use the tables below to specify your code model as well as the scale of your programs and data:

Table 4. Supported code models for V5 RV32 and RV64 toolchains

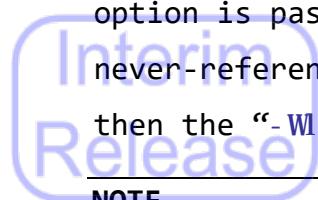
	Section Code model	text	data	rodata
RV32	small (medlow)	Full 32-bit addressing space		
	medium (medlow)			
	large (medlow)			
RV64	small (medlow)	The highest 2 GiB and the lowest 2 GiB		
	medium (medany)		± 2 GiB	
	large	Full 64-bit addressing space		

- Other optimization flags: This page allows you to manually turn off default optimization options applicable to a particular optimization level. For example, `-fomit-frame-pointer` is applicable to `-Og` and other levels by default. To prevent this default setting, simply enter `-fno-omit-frame-pointer` as follows:



- Link Time Optimization (-flto): This option makes it possible to apply aggressive link time optimization to your programs. For requirements and limitations pertaining to this optimization option, please refer to the *Andes Programming Guide*.
- Remove unused function sections (-ffunction-sections) and unused data sections (-fdata-sections): These options allow

the removal of never-referenced sections. When both options are selected, the compiler places each function/data item into its own section of the output file, whereupon the “`--gc-sections`” option is passed to the linker for the removal of never-referenced functions/data. If GCC is a linker program, then the “`-Wl, --gc-sections`” option is used instead.



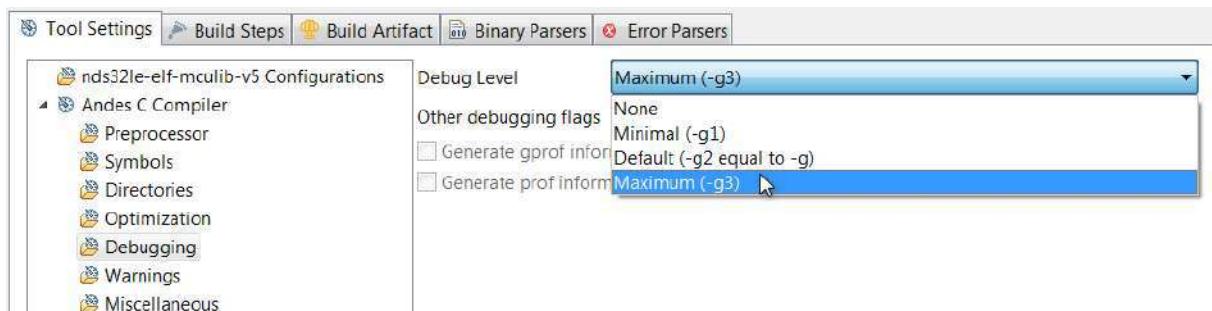
#### NOTE

The option “`-gc-sections`” removes used but not directly-referenced sections, such as interruption vector tables or code that is called or jumped to by addresses instead of symbol names. To keep these sections requires that you include the keyword “`KEEP`” in the linker script file, as follows:

```
. nds32_init : { KEEP(*(.nds32_init)) }
```

#### ■ “Andes C Compiler > Debugging” page

This page allows you to configure the debugging level for your project.



#### ■ “Andes C Compiler > Miscellaneous” page

- Other flags: “`-fmessage-length=N`” is an option that allows the formatting of error messages to ensure that they fit on lines of N characters in length. This option has been included in the compiler settings of all Andes projects. The default value is

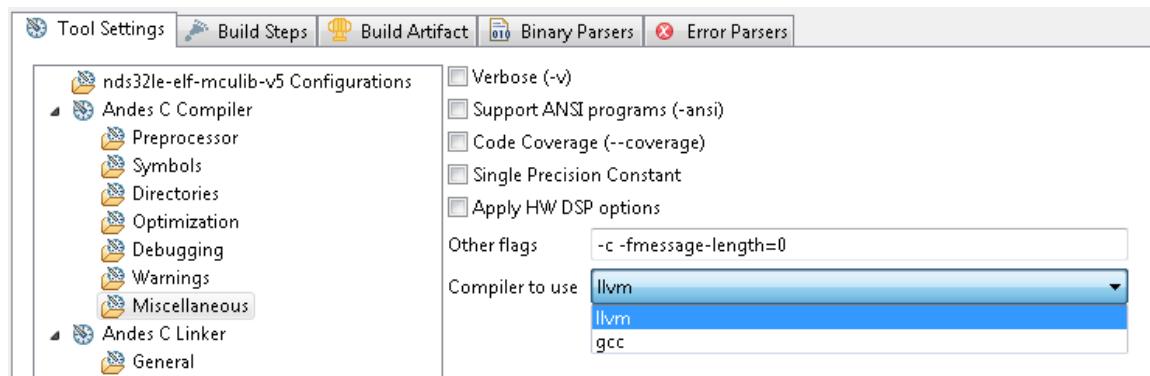
**0**, which means that no line-wrapping is performed for error parsing and each error message appears on a single line in the **Problem** view. To configure the format of error messages, simply modify the value of this option.

- **Code coverage** (`--coverage`): This option enables code coverage analysis. For details concerning code coverage analysis in AndeSight, please refer to Section 2.7.

#### NOTE

Andes LLVM compiler has not supported the code coverage feature yet. Please use the GCC compiler to build programs that need code coverage analysis.

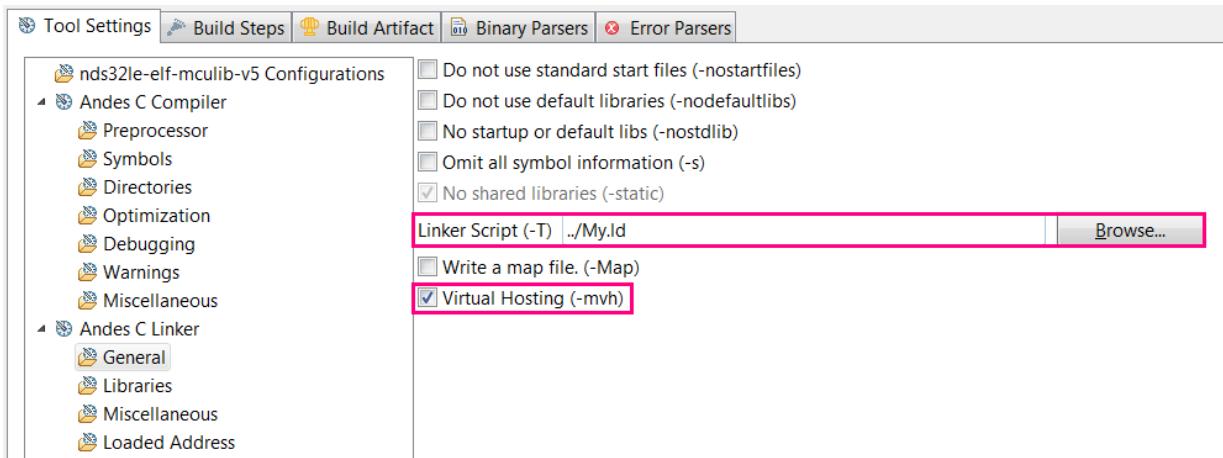
- **Single Precision Constant**: This option treats a floating point constant as a single precision constant, rather than converting it into a double-precision constant.
- **Apply HW DSP options**: This option, when selected, enables compiler and linker to be applied with options that enable the DSP ISA extension supported by the target.
- **Compiler to use**: This option is only present for Andes V5 elf toolchains that include both LLVM and GCC compilers. For building programs with a V5 elf toolchain, this combo box allows you to specify which compiler to use.



### ■ “Andes C Linker > General” page

This page allows you to configure general linker options.

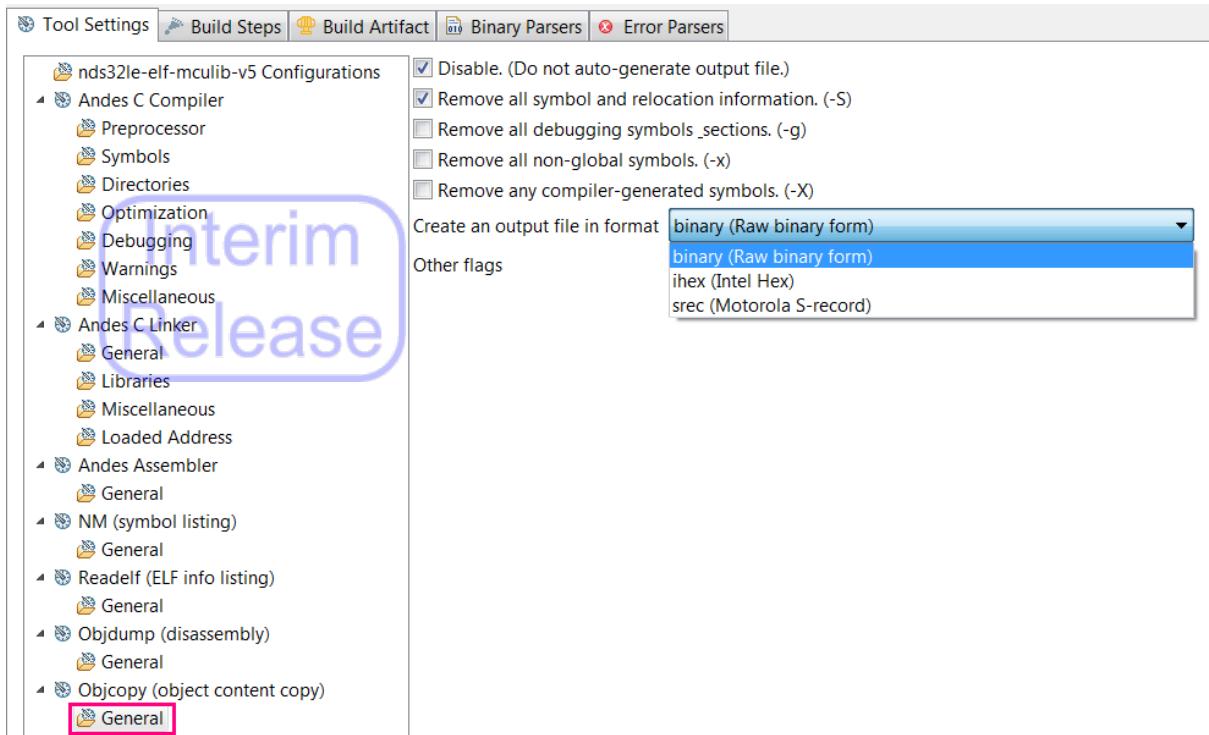
- **Linker Script (-T)**: Assign a linker script to your project. This setting is overwritten and appears as `$(LDSAG_OUT)` when the LdSaG tool is enabled. If the LdSaG tool is disabled (i.e., the “Generate linker script” option in “LdSaG Tool > General” is unselected), you have to specify a linker script here.
- **Virtual Hosting (-mvh)**: Select this option to activate Virtual Hosting support. When Virtual Hosting support is enabled, I/O requests from target boards without I/O devices can be directed to GDB on the host side, thereby accelerating the development process and shortening the development cycle. For system calls supported by Virtual Hosting, please refer to the *Andes Programming Guide*.



### ■ “Objcopy (object content copy) > General” page

This page allows you to copy the content of one object file to another object file.

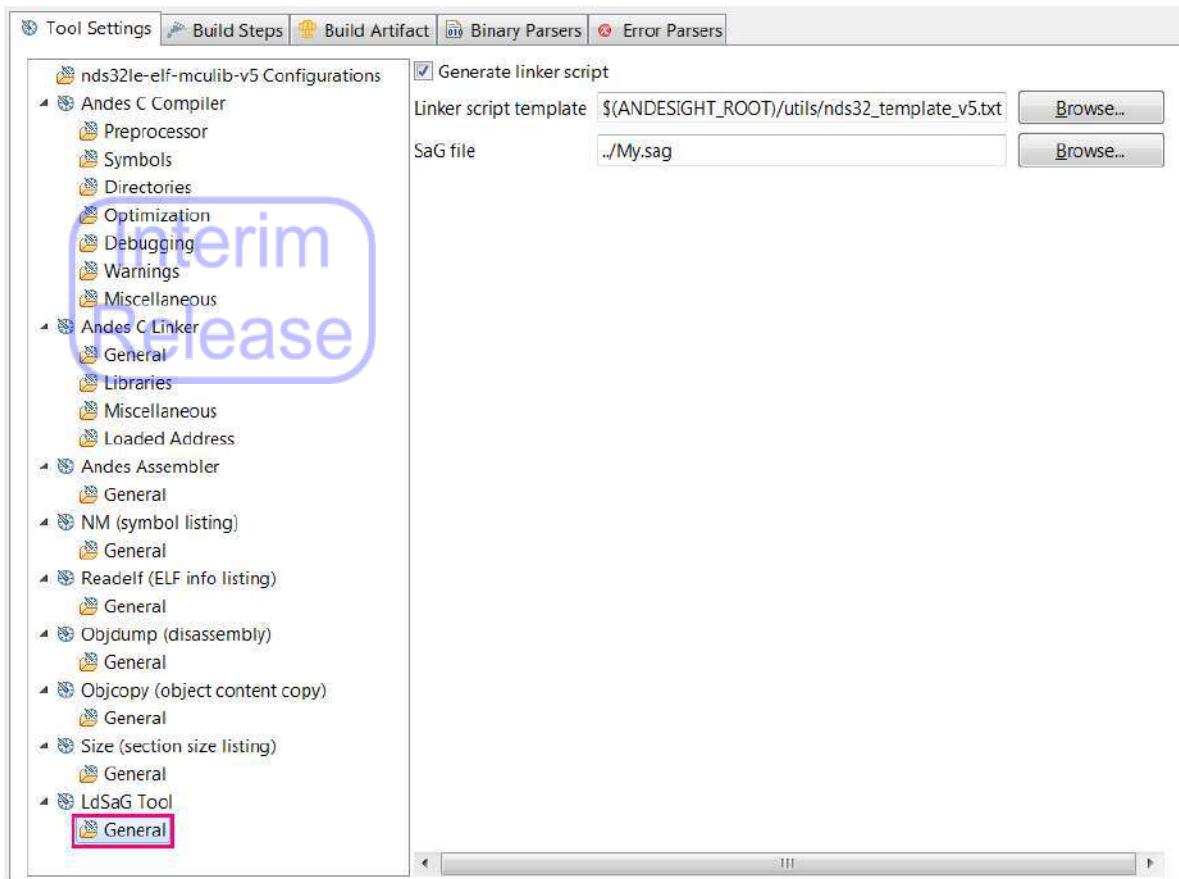
- **Create an output file**: Specify the format of your destination object file. Three industry-standard formats are supported: binary (raw binary form), ihex (Intel Hex format), and srec (Motorola S-record format).



### ■ “LdSaG Tool > General” page

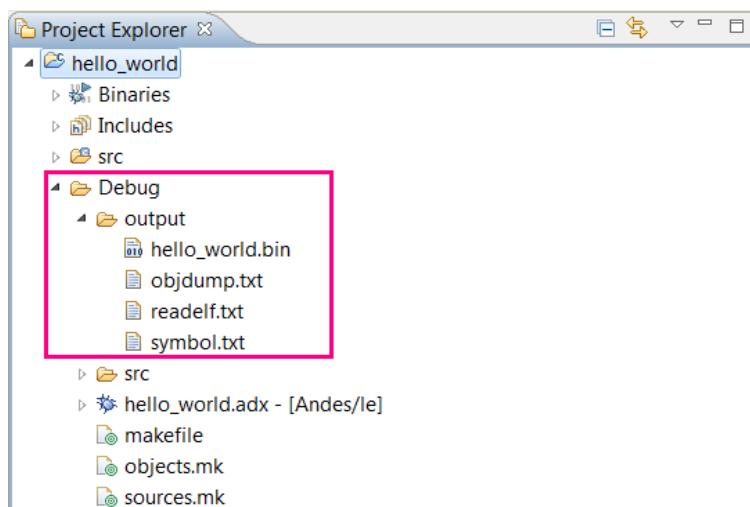
This page allows you to generate a linker script for your program.

- Generate linker script: Select this option to generate a linker script under **PROJECT\Debug\output** and use it for your application. When this option is selected, the “Linker Script (-T):” setting in “Andes C Linker > General” page is overwritten and presented as “**`$(LDSAG_OUT)`**”.
- Linker script template: Specify a template to generate linker scripts. The default template is **`nds32_template_v5.txt`** for V5 cores.
- SaG file: Specify a SaG-formatted script that describes the memory map of the executable image. The **.sag** file can be created or edited using the SaG editor (see Section 2.1.2.3).



**Step 4** After configuring the build options, click “Apply and Close.”

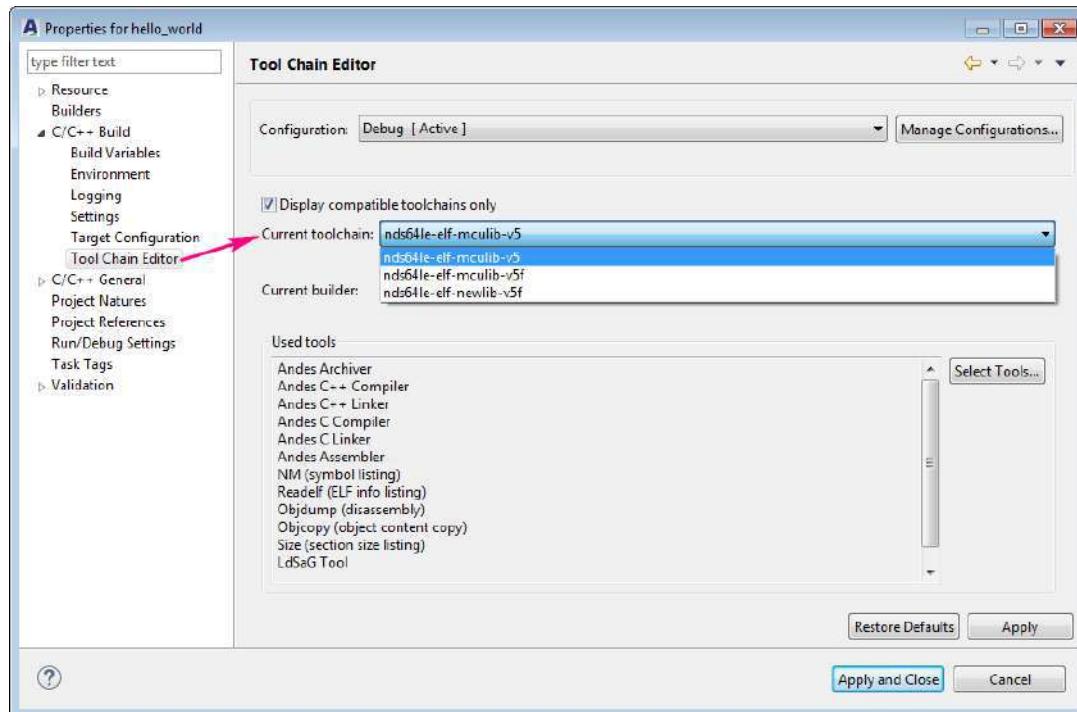
**Step 5** When binutils (NM/Readelf/Objdump/Objcopy/Size) are enabled, the output is generated in **PROJECT\Debug\output** after the project is built.



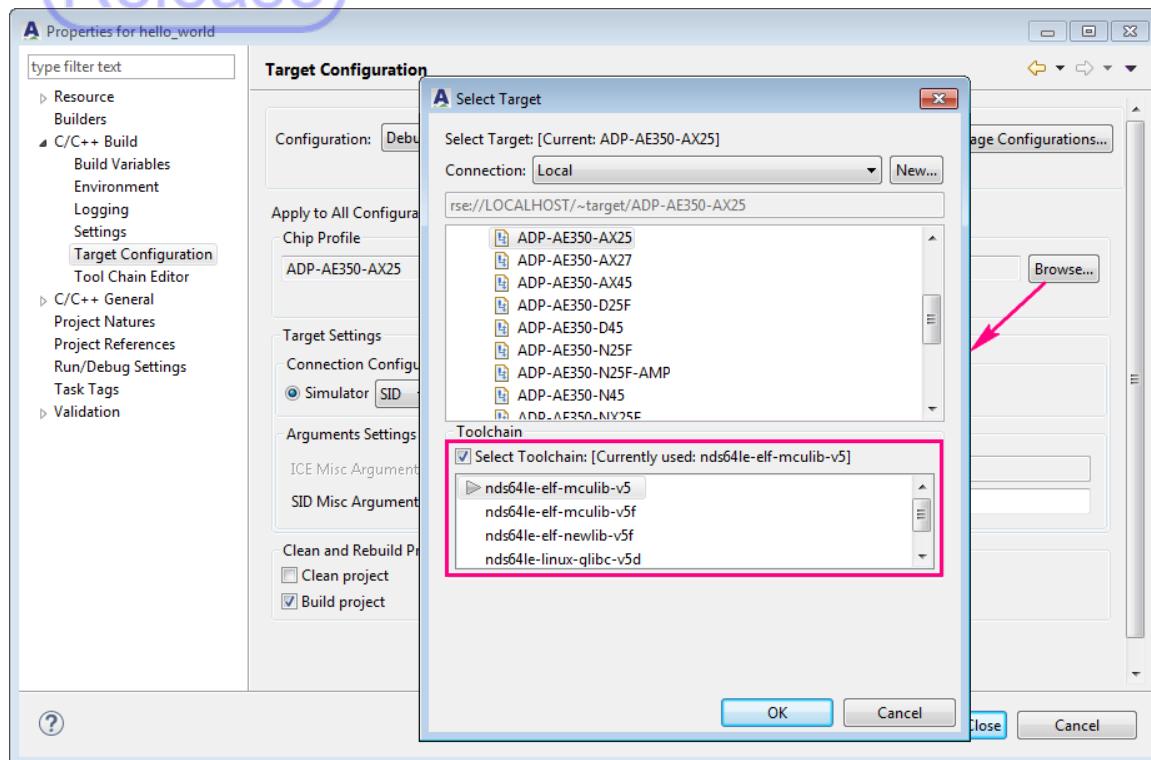
#### 2.1.4.3 Selecting/switching toolchains

Andes toolchains are built with respect to the core ISA, CPU version (32 or 64 bit), library, and endianness. Please consult Table 2 for toolchains suggested for development using Andes cores. The following describes the process used to select/switch a toolchain for an Andes project. To change global toolchain defaults for a specific type of project, please refer to “**Changing default toolchain(s) for specific project types**” at the end of this section. Changing the preferred toolchain(s) for a specific chip profile requires modifications to its Chip Profile Property file (ChipFile.atd). Please refer to Step 4 in Section 2.2.1.2 to find out where the Chip Profile Property file resides. Then, refer to Step 5 in the same section to find out how to evoke the chip profile editor and configure toolchain settings.

**Step 1** Right-click the project folder and select “Properties” to invoke the **Properties** dialog. Select “C/C++ Build > Tool Chain Editor”. On the drop-down list of “Current toolchain”, select an appropriate toolchain.



**Step 2** You may also switch the toolchain when configuring the target. On the **Properties** dialog, select “C/C++ Build > “Target Configuration” on the pull-down menu. Click “Browse...” in the Chip Profile section to invoke the **Select Target** wizard. Select your desired target, check the “Select Toolchain” option, and make further selection(s) from the toolchain list. Click “OK.”

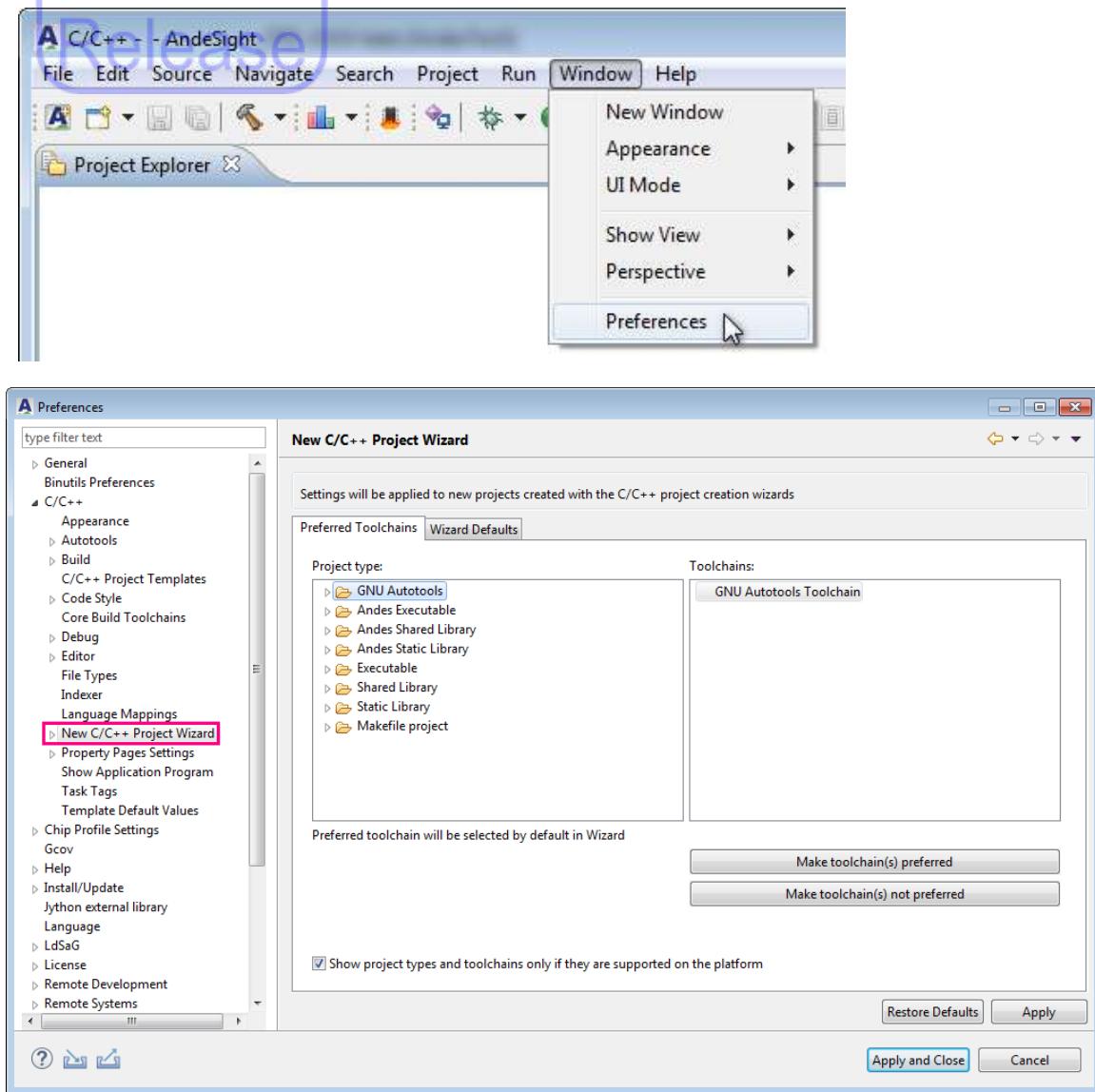


**Step 3** Click “Apply and Close” to complete the process.

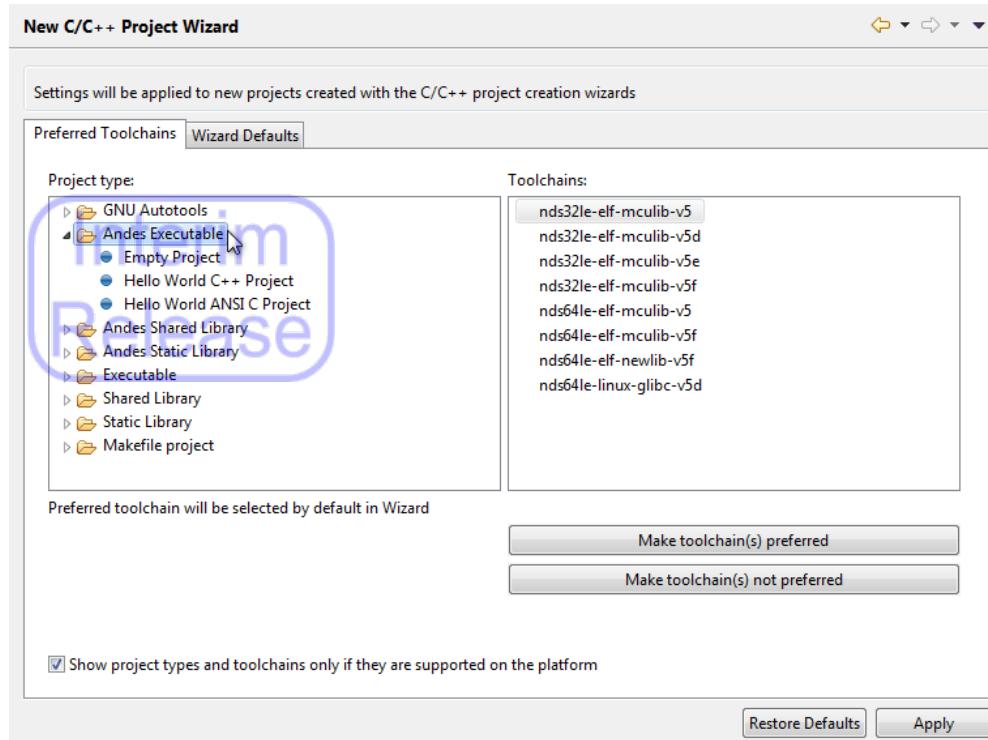
## Changing default toolchain(s) for specific project types

To set/change the default toolchains for specific project types, follow the steps below:

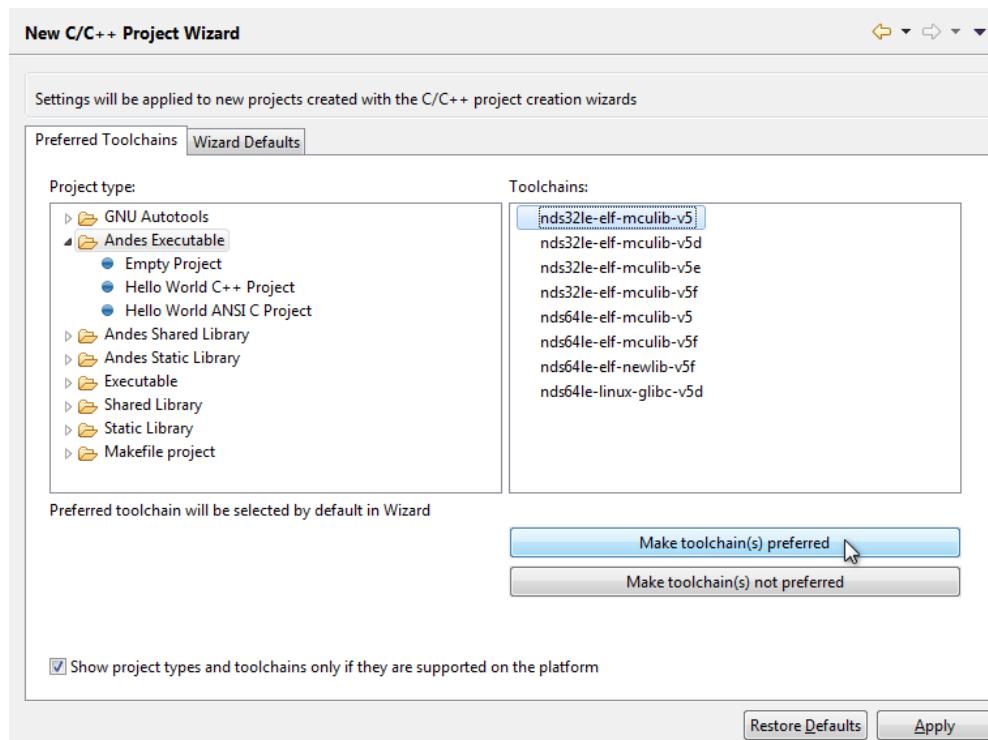
**Step 1** Select “Windows > Preferences” on the main menu of AndeSight. On the Preferences dialog, select “C/C++ > New C/C++ Project Wizard.”



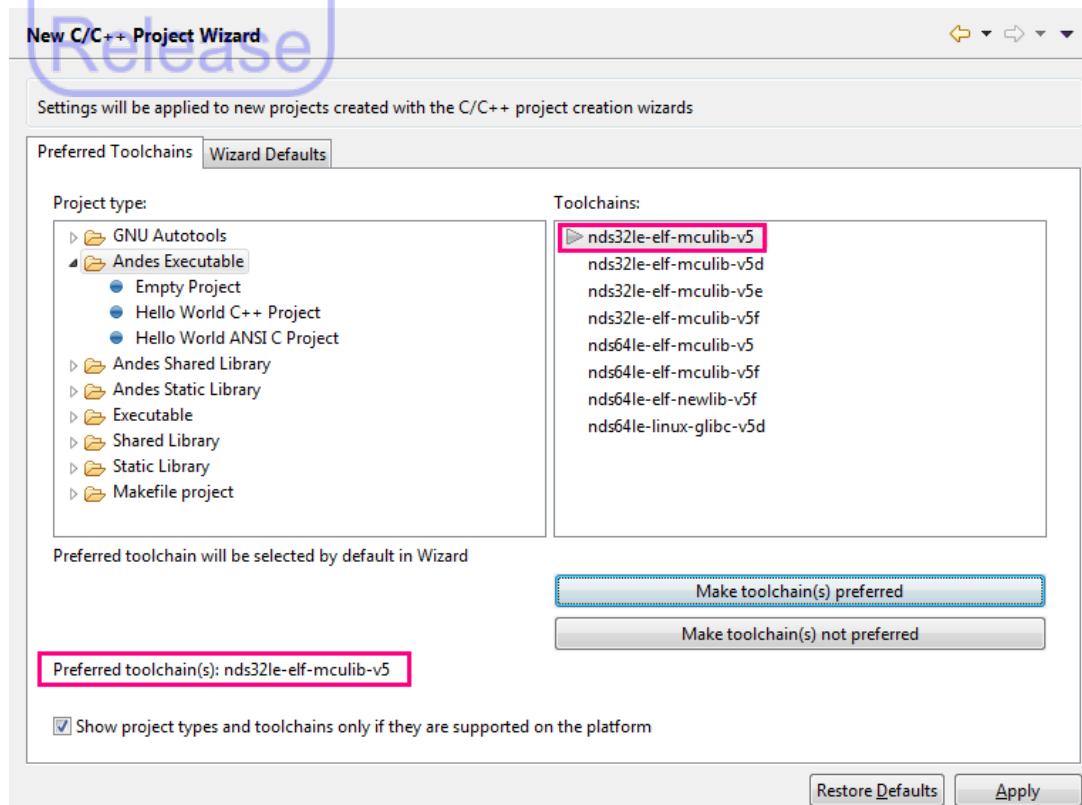
**Step 2** On the **New C/C++ Project Wizard** page, select a project type from “Andes Executable”, “Andes Shared Library”, “Andes Static Library”, and “Makefile project.”



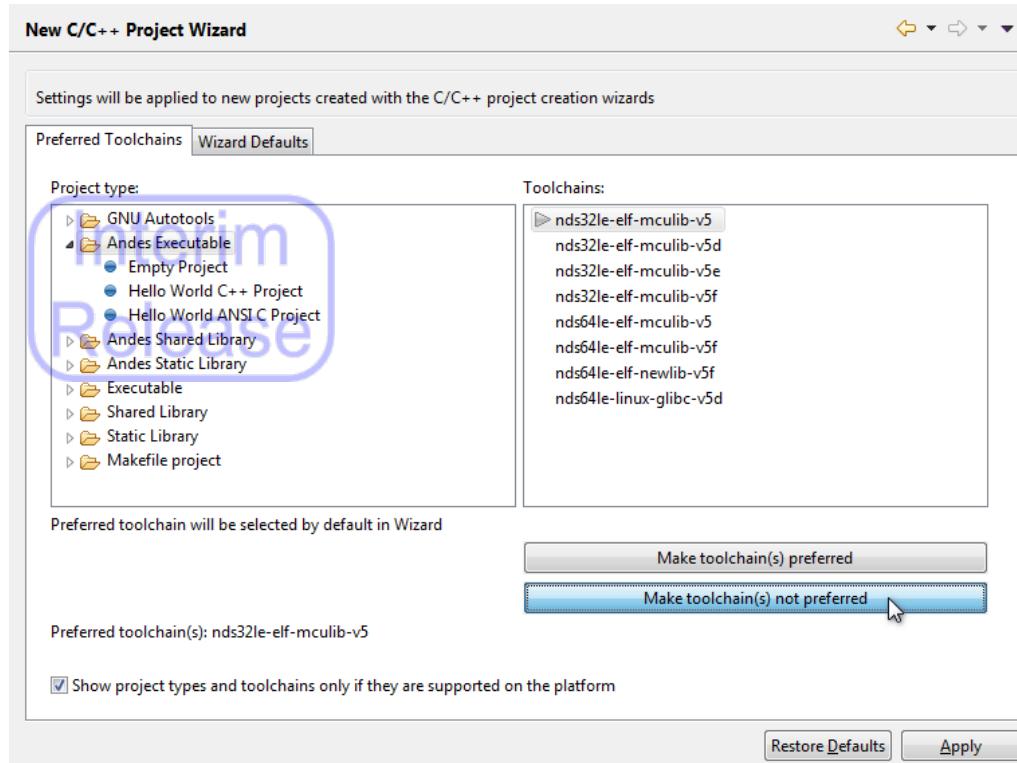
**Step 3** Select one or multiple toolchains from the “Toolchains” column and click “Make toolchain(s) preferred.”



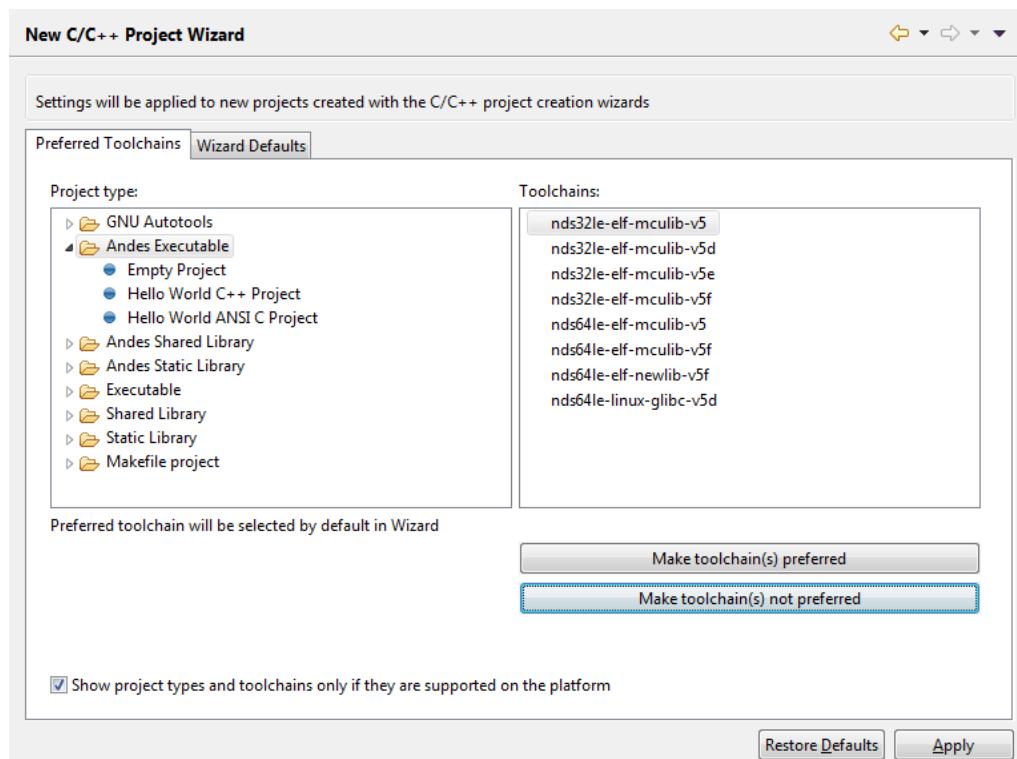
**Step 4** The toolchain(s) preceded by a preference marker ➤ in the “Toolchains” column is included in the list of “Preferred Toolchain(s)” at the bottom of the page. This indicates that it has been designated as a default toolchain for that type of project. Click “Apply”.



**Step 5** To remove a toolchain from the default list for a particular type of project, select the desired project type in the “Project type” column and then select a toolchain preceded by the preference marker ➤ in the “Toolchains” column. Click the “Make toolchain(s) not preferred” button.



- Step 6** The selected toolchain is no longer preceded by the preference marker  in the “Toolchains” column. This indicates that it has been removed from the default toolchain(s) list for that type of project.

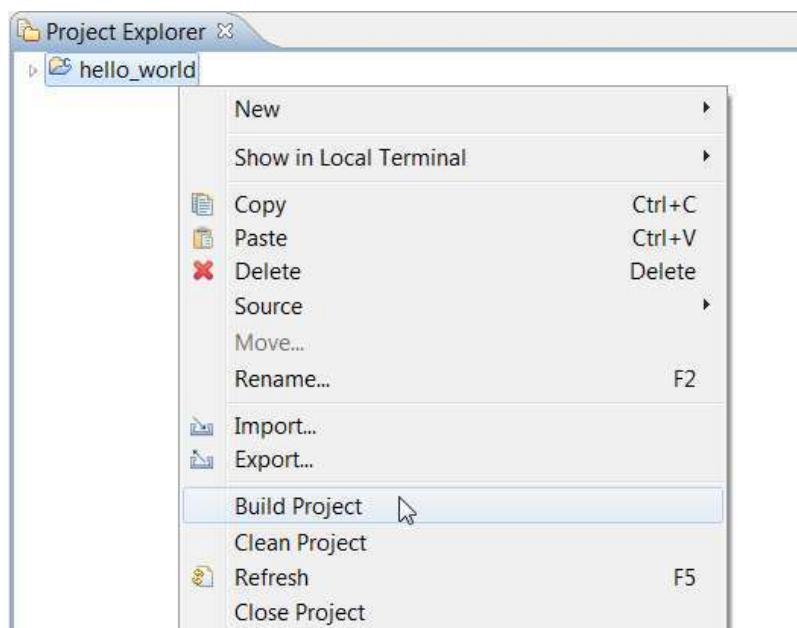


## 2.1.5. Building a project

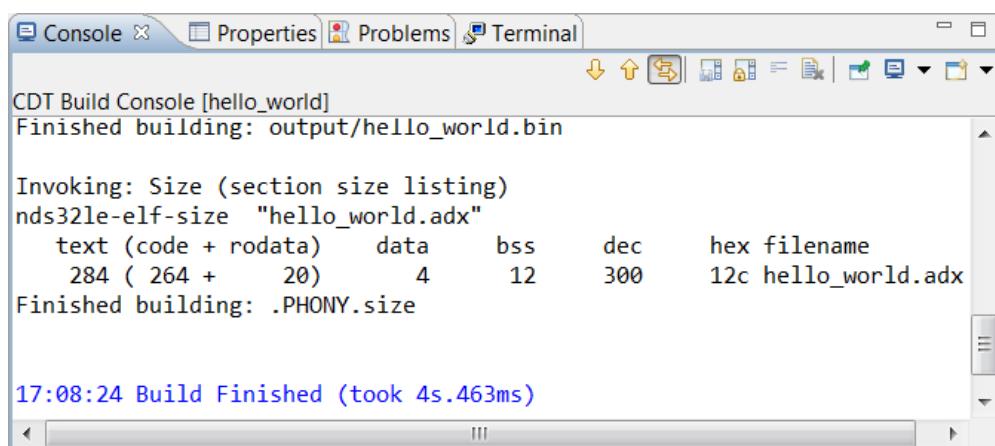
Building a project involves translating source code into an executable for a specific target system. To build a project in AndeSight, follow the steps below:

**Step 1** Configure the build system as outlined in Section 2.1.4.

**Step 2** Click  (Build) on the AndeSight toolbar to commence the build process. You may also right-click the project folder to trigger the pull-down menu and select “Build Project”.



**Step 3** Verify the results of the build in the **Console** view.



```

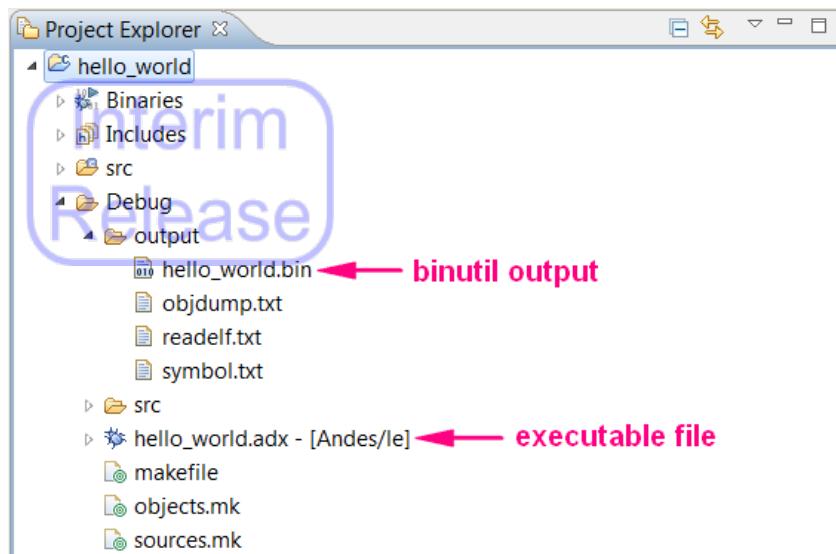
CDT Build Console [hello_world]
Finished building: output/hello_world.bin

Invoking: Size (section size listing)
nds32le-elf-size "hello_world.adx"
    text (code + rodata)      data      bss      dec      hex filename
      284 ( 264 +      20)        4       12      300      12c hello_world.adx
Finished building: .PHONY.size

17:08:24 Build Finished (took 4s.463ms)

```

**Step 4** The executable file and object file are by default generated under **PROJECT\Debug** and the binutil output is under **PROJECT\Debug\output**.

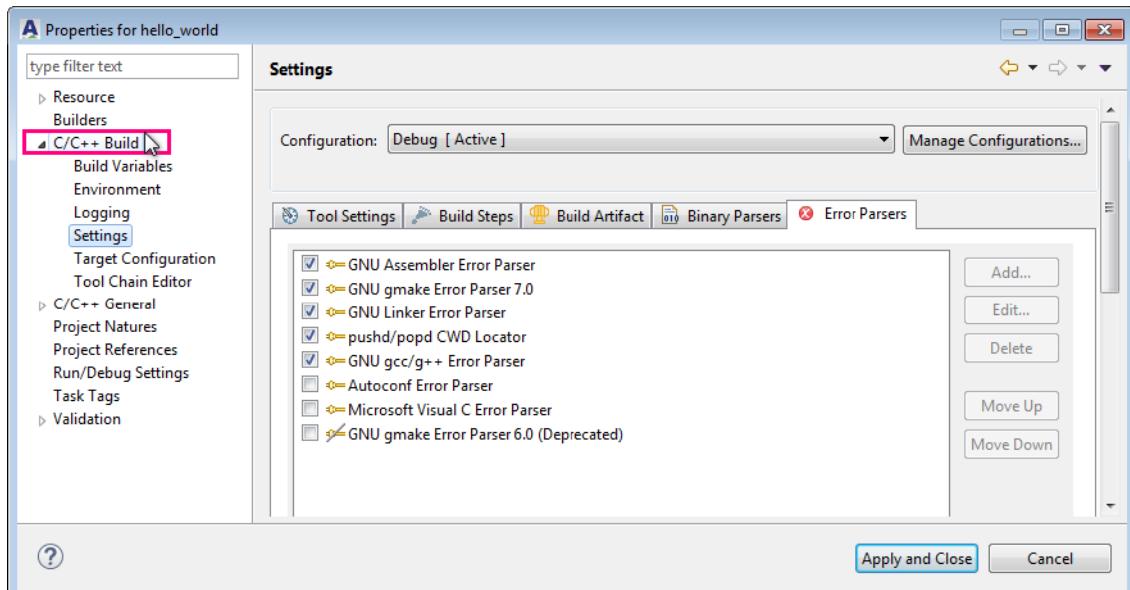



---

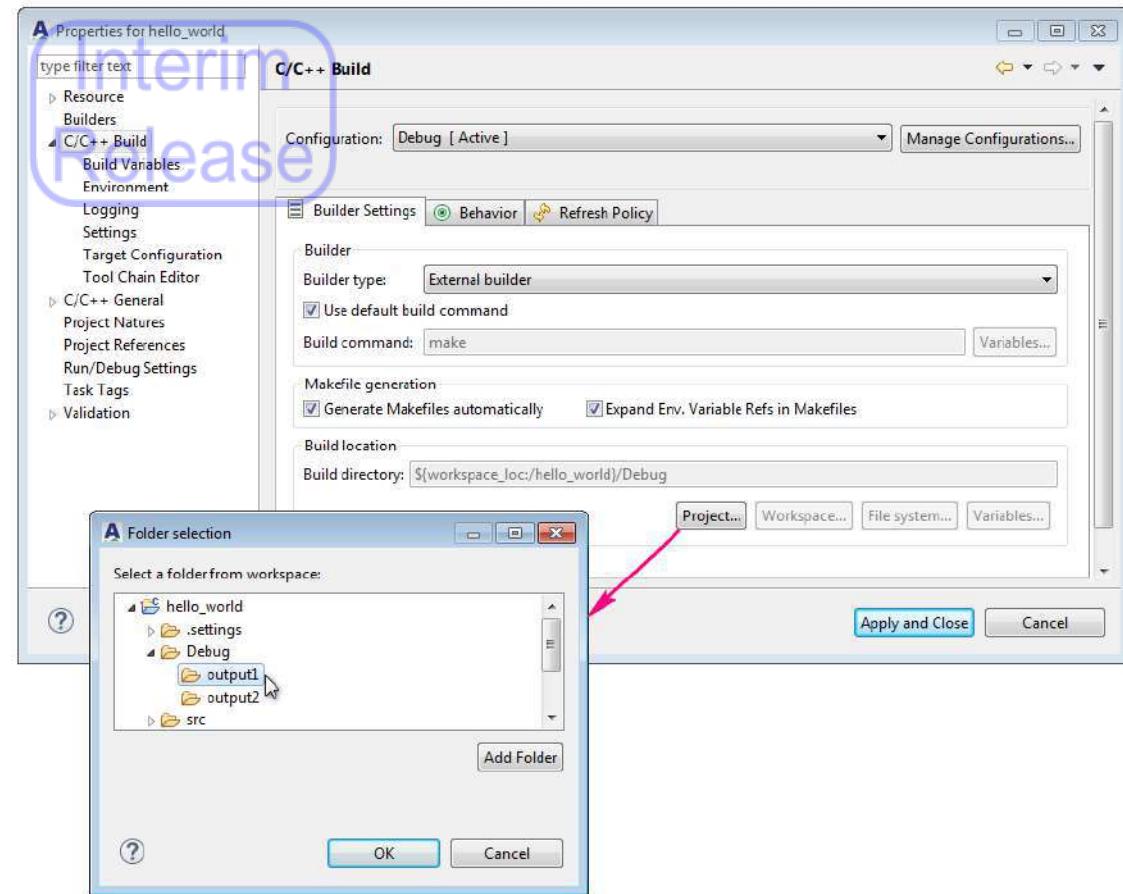
#### NOTE

If you want to change the default build location of the executable and object files, follow below before commencing the build:

1. Select the project in **Project Explorer** and right-click to select “Build Settings” on the pull-down menu.
2. On the **Properties** dialog, select “C/C++ Build” on the navigation pane.



3. On the C/C++ Build page, click on the “Project...” button in the Build location section to invoke the **Folder selection** dialog and specify the desired build location on it.



## 2.1.6. Static analysis

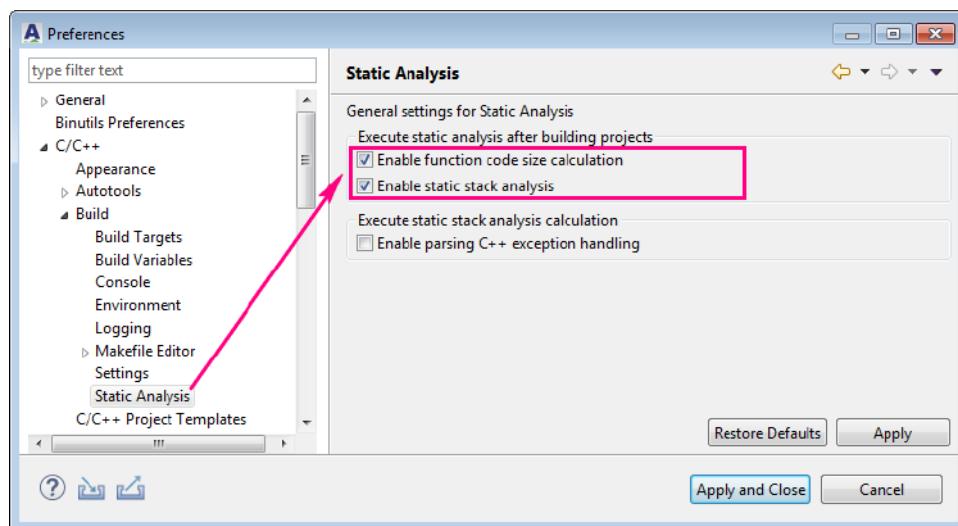
AndeSight allows you to perform two types of static program analysis, function code size calculation and static stack analysis. Both forms of analysis are based on project binary executables. Function code size provides information concerning the code size of individual functions and the total code size of user-defined functions, whereas static stack analysis indicates the stack size for each function and the entire program.

Automatic static analysis can be performed whenever a project is built. Before building a project, follow the steps below in the **Preferences** settings to enable the functionality:

**Step 1** From AndeSight main menu, click “Window > Preferences” to invoke the **Preferences** dialog.

**Step 2** Select “C/C++ > Build > Static Analysis” in the navigation pane.

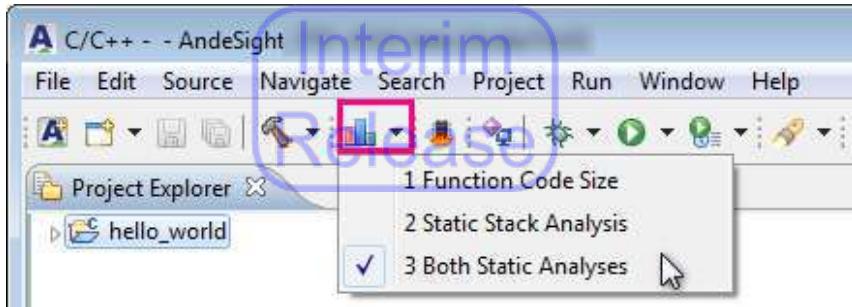
**Step 3** On the **Static Analysis** page, select the checkbox indicating the static analysis that you would like to conduct and click “OK.”



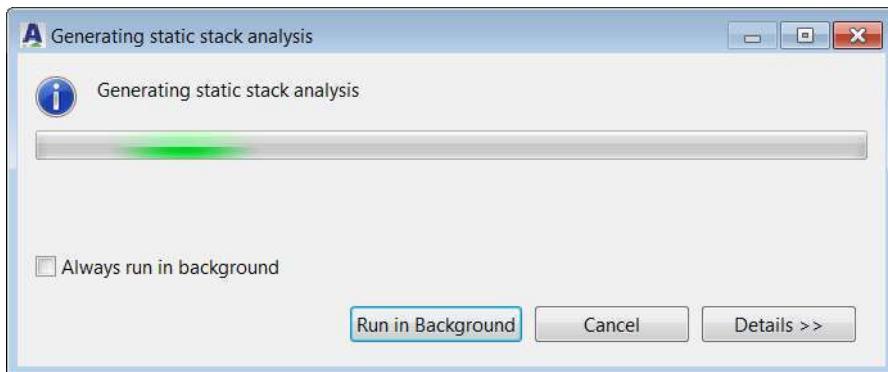
Note that the default static analysis for C++ program does not cover code relating to exception handling. If you want the analyzer to parse all the code, please specify the option “Enable parsing C++ exception handling” on this page.

In the event that your preferred static analysis is not enabled by default, it can be executed for a built project via a button on the AndeSight toolbar. After building a project, click the drop-down

arrow  (Static Analysis) and make a selection from the pull-down menu.



Whenever static analysis is conducted, a dialog pops up notifying you that analysis is in progress. Once analysis is completed, the results are displayed in the invoked **Function Code Size** view and/or the **Static Stack Analysis** view. The following sections respectively introduce the two views.



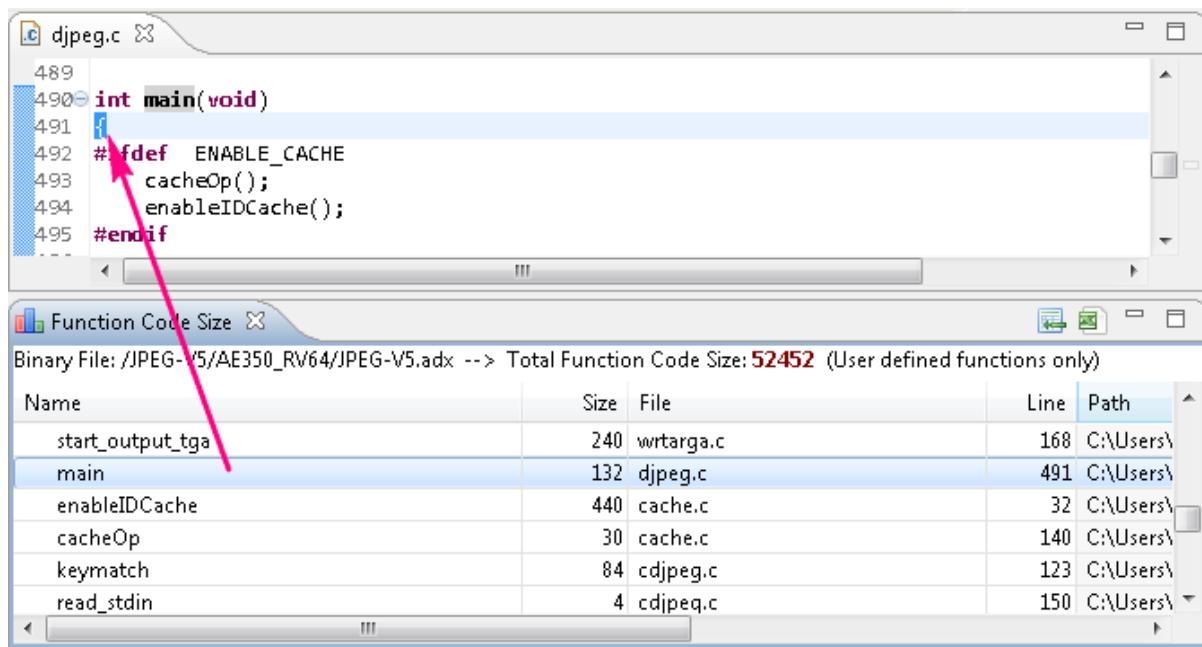
### 2.1.6.1 Function Code Size view

The **Function Code Size** view allows you to investigate information pertaining to the function size of built projects. This view displays the total size of user-defined functions as well as the sizes of individual functions analyzed from project executable(s).



Name	Size	File	Line	Path
jpeg_getc	64	djpeg.c	419	C:\Users\
print_text_marker	132	djpeg.c	434	C:\Users\
usage	296	djpeg.c	96	C:\Users\
parse_switches	992	djpeg.c	176	C:\Users\
default_decompress_parms	352	jdapimin.c	115	C:\Users\
output_pass_setup	180	jdapistd.c	96	C:\Users\
init_destination	36	jdatadst.c	47	C:\Users\

This view also presents source details of each function. Functions can be sorted by name or size simply by clicking the column headers. You may also double-click a function of interest to find it in the source code.



The screenshot illustrates the integration between the Function Code Size view and the source code editor. The top window shows the source code for `djpeg.c`, specifically the `main` function. A red arrow points from the entry for `main` in the Function Code Size view below to the `#ifdef ENABLE_CACHE` directive in the source code. The Function Code Size view displays the total size of 52452 and lists other functions like `start_output_tga`, `cacheOp`, and `keymatch`.

Name	Size	File	Line	Path
start_output_tga	240	wrtarga.c	168	C:\Users\
main	132	djpeg.c	491	C:\Users\
enableIDCache	440	cache.c	32	C:\Users\
cacheOp	30	cache.c	140	C:\Users\
keymatch	84	cdjpeg.c	123	C:\Users\
read_stdin	4	cdjpeg.c	150	C:\Users\

Note that this view does not provide code size information for functions that have been optimized out during the build process. Furthermore, the “Total Function Code Size” value in this view accounts only for the .text section. To obtain the exact function size of the entire program requires that you add up the .text, .rodata and .nd32\_init sections.

## Toolbar for the Function Code Size view

### Include external libraries



Click this button to include library functions in calculations on function code size. When this button is not selected, only user-defined functions are counted.

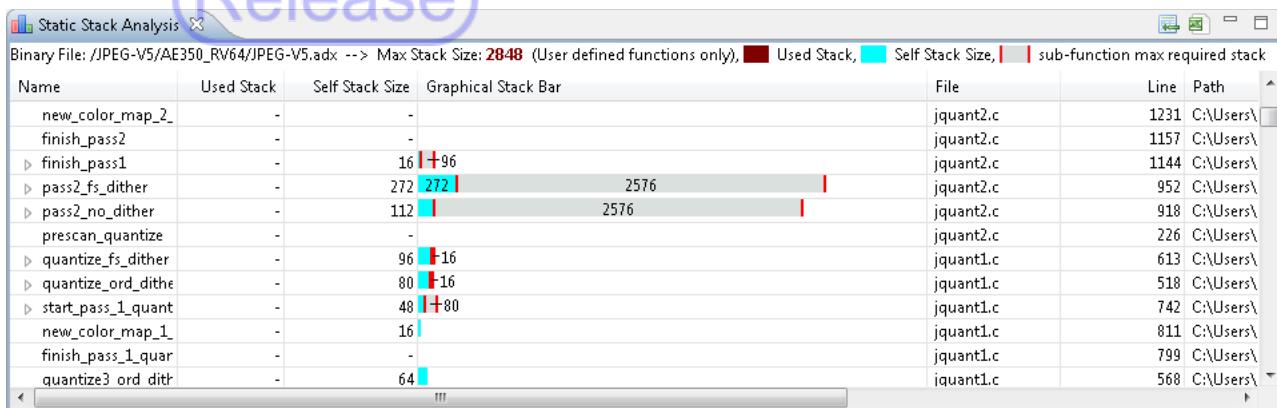
### Export to CSV file



Click this button to export function code size data to a .csv file.

### 2.1.6.2 Static Stack Analysis view

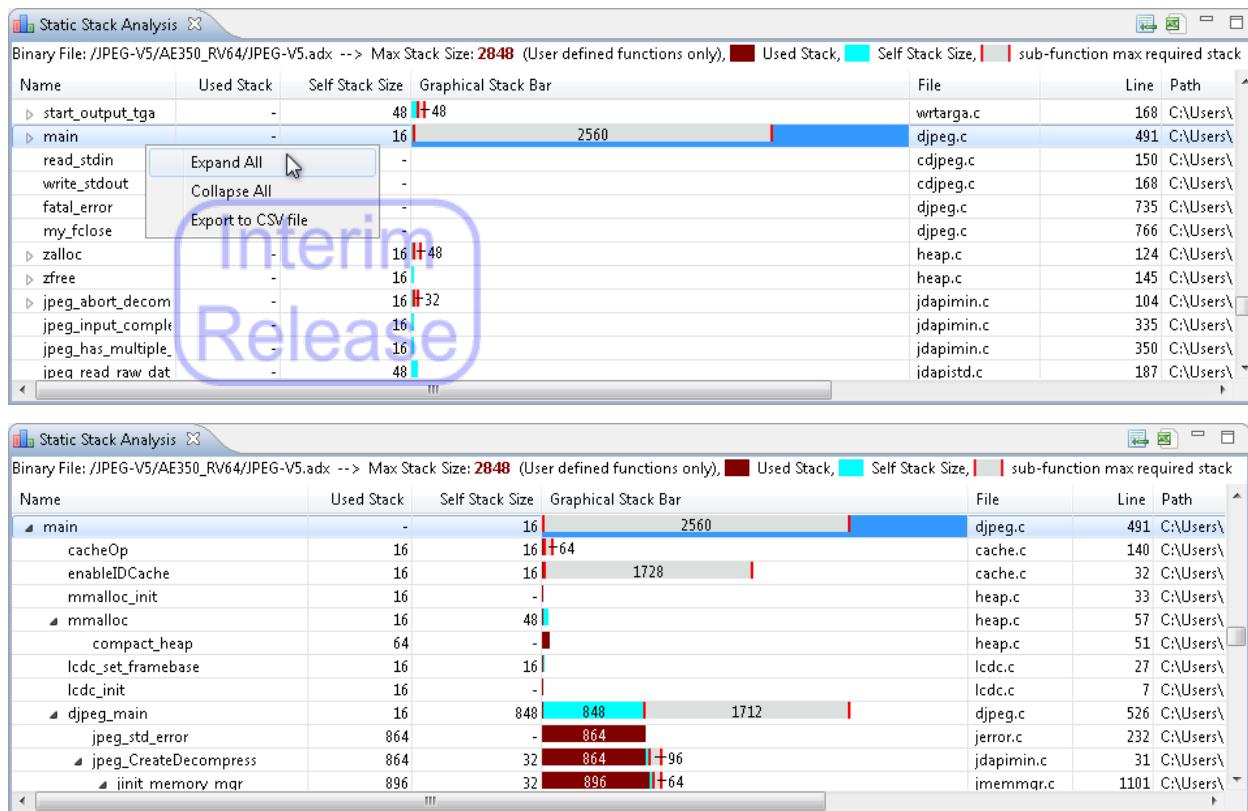
The **Static Stack Analysis** view displays the size of static stacks for built projects. After generating analysis results, the maximum size of stacks required for the entire program are listed at the top of this view, and the functions with their stack information and source details are listed in a call graph. Note that the size of stacks for interrupt service routines and indirectly-called functions are not counted in the maximum size of required stacks.



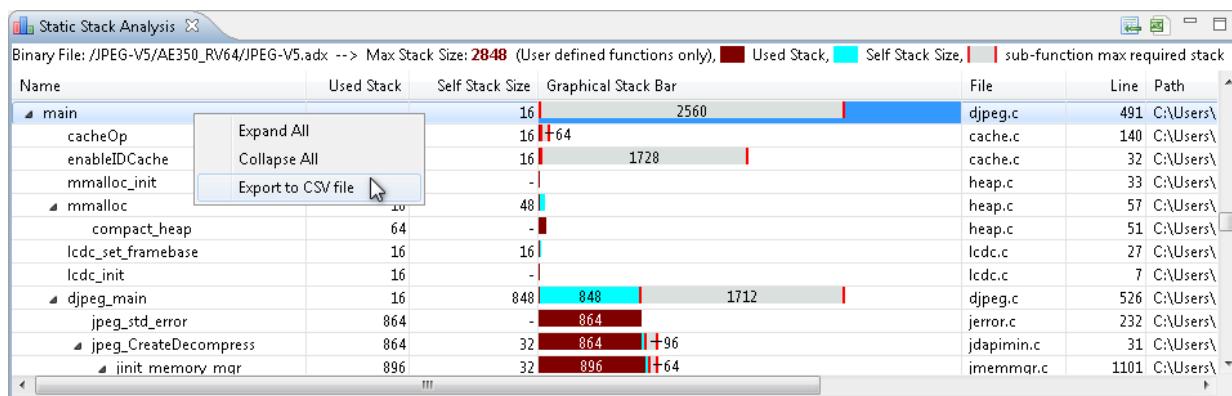
The following stack information is provided for each function:

- Self Stack Size – the size of stacks used for the current function
- Used Stack – the size of stacks used before the current function is called
- Graphical Stack Bar – a graphical representation of cumulative stack usage for the current function. This bar lists the size of the self stack and used stack as well as the maximum size of stacks used by the callees of the function. The three stack sizes are presented as blocks of various colors (red for used stack, cyan for self stack size, and grey for sub-function maximum required stack).

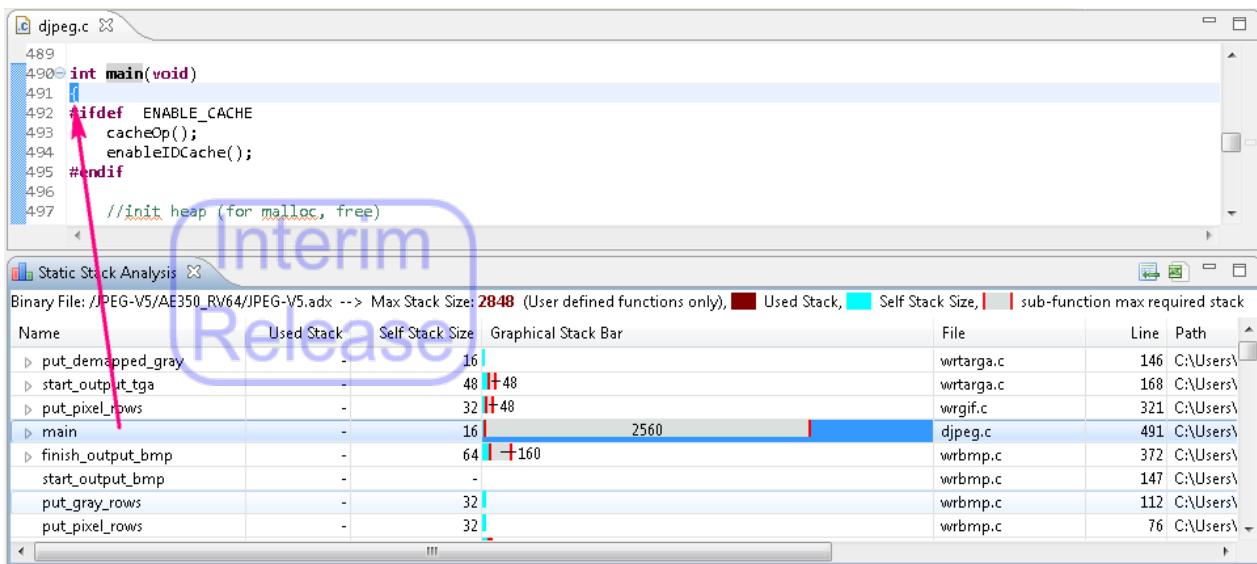
When first invoked, this view presents stack information of root functions, including entry functions, interrupt service routines, and indirectly-called functions. Right-click a root function of interest and select “Expand all” in the pull-down menu to display all sub-functions and their respective stack information. This call graph of stack information is used to identify the execution path that consumes the most stacks.



You may also right-click a root-function of interest and select “Export to CVS file” in the pull-down menu to export its stack information, including stack analysis of its sub-functions, to a .cvs file.



In the **Static Stack Analysis** view, functions can be sorted by name, self stack size, or used stack size by clicking the column headers. You may also double-click a function of interest to locate it in the source code.



### Toolbar for the Static Stack Analysis view

#### Include external libraries



Click this button to include library functions in static stack analysis. When this button is not selected, only user-defined functions are counted.

#### Export to CSV file



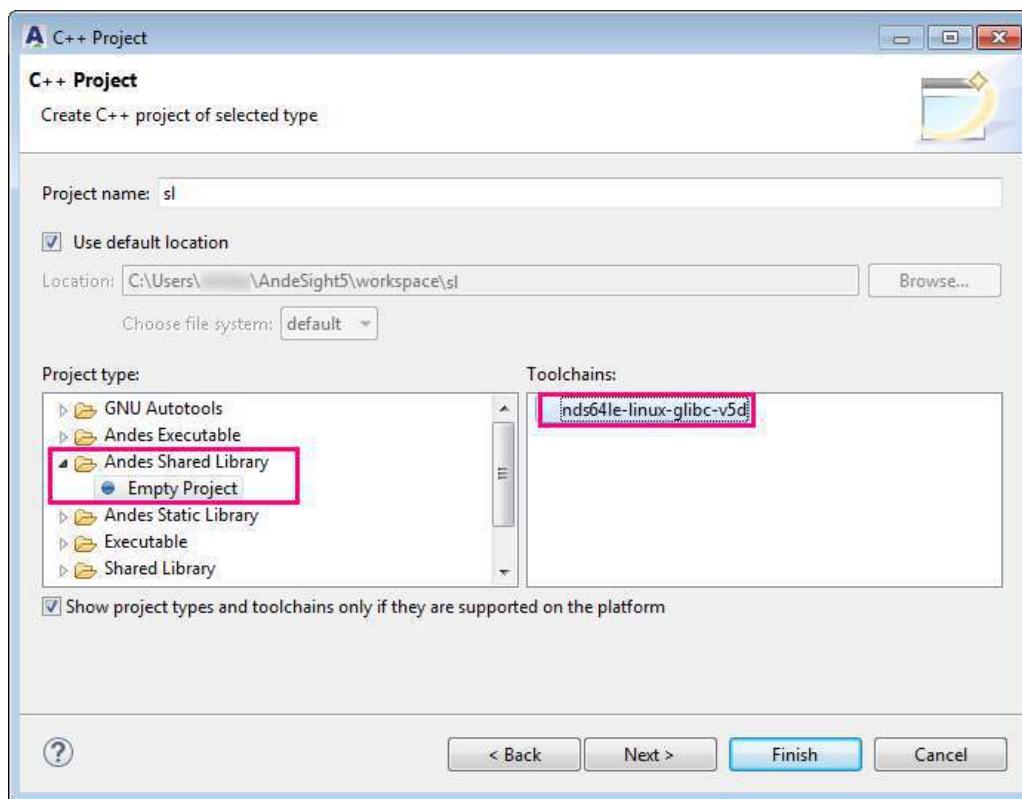
Click this button to export static stack analysis of root functions and their expanded nodes in the view to a .csv file.

### 2.1.7. Creating and building a shared/static library project

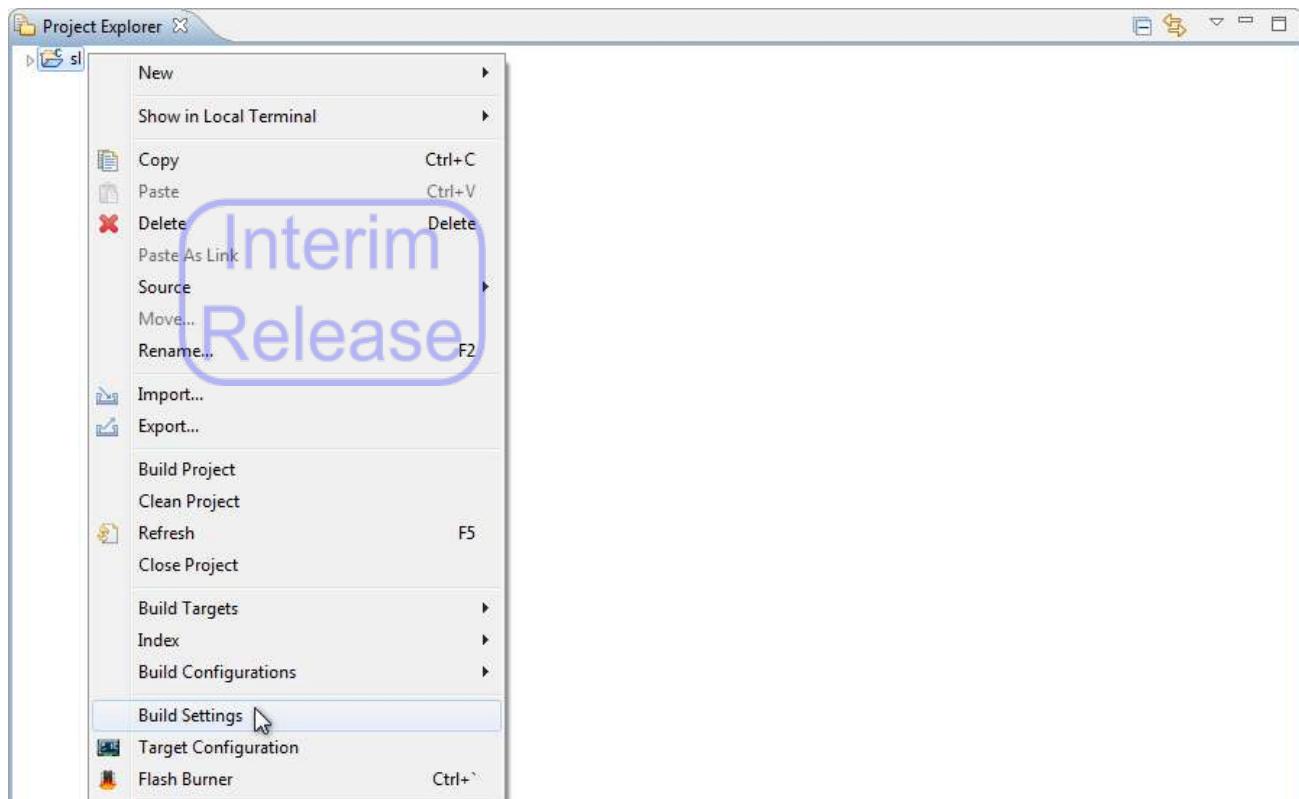
Proceed as follows to create a shared/static library project using the regular project creation procedure.



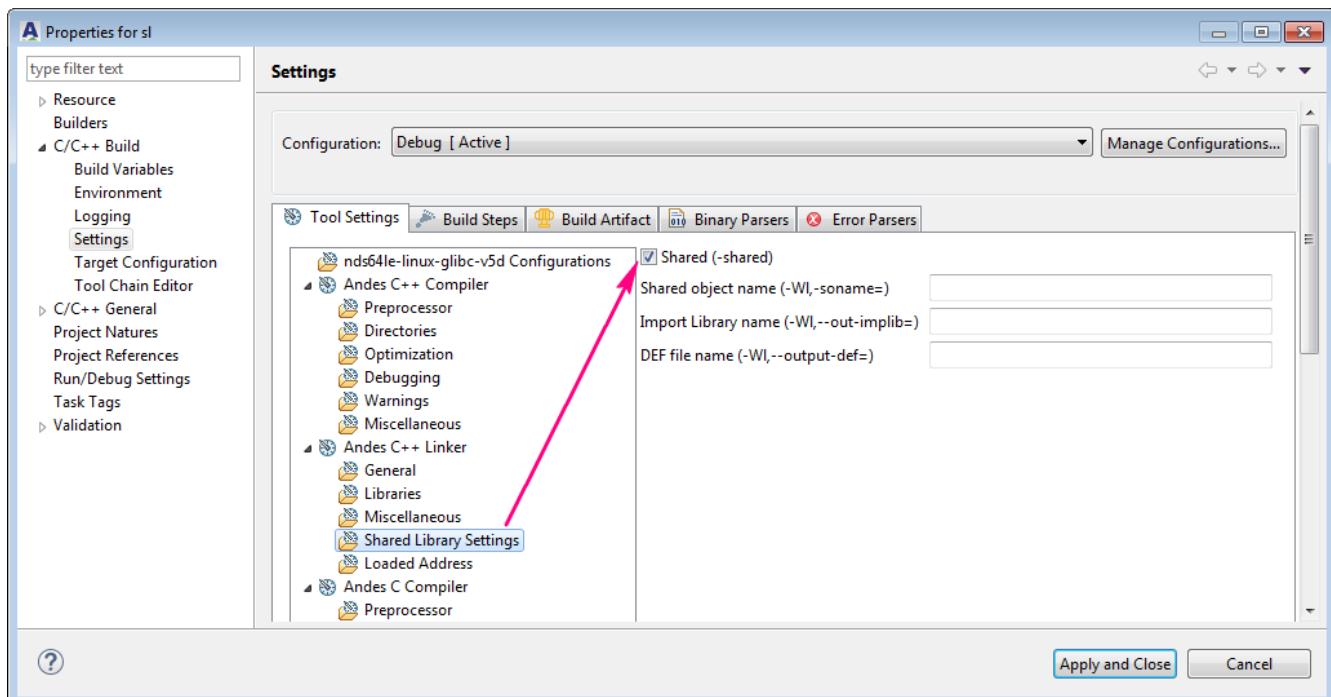
**Step 1** Please refer to Section 2.1.1.2 for instructions on creating a project for generic targets. At the **C or C++ Project** prompt, select “[Andes Shared Library|Andes Static Library] > Empty Project” as the project type. To create a shared library project, be sure to specify a glibc toolchain. In the example below, “nds64le-linux-glibc-v5d” is selected as the toolchain for the shared library project “sl.”



**Step 2** The shared/static library project is created in the **Project Explorer** view. For a static library project, jump to Step 4. For a shared library project, right-click the project folder and select “Build Settings” in the pull-down menu.



**Step 3** In the **Tool Settings** tab, select “Andes C Linker > Shared Library Settings” to ensure that the “Shared (-shared)” checkbox is checked. Click “Apply and Close” to exit project settings.



**Step 4** Please refer to Section 2.1.2.2 to add header files and source files.

For example, add the following “sl.h” and “sl.c” to output the shared library “libsl.so.”

```
#ifndef SL_H_
#define SL_H_

int sum(int a, int b);

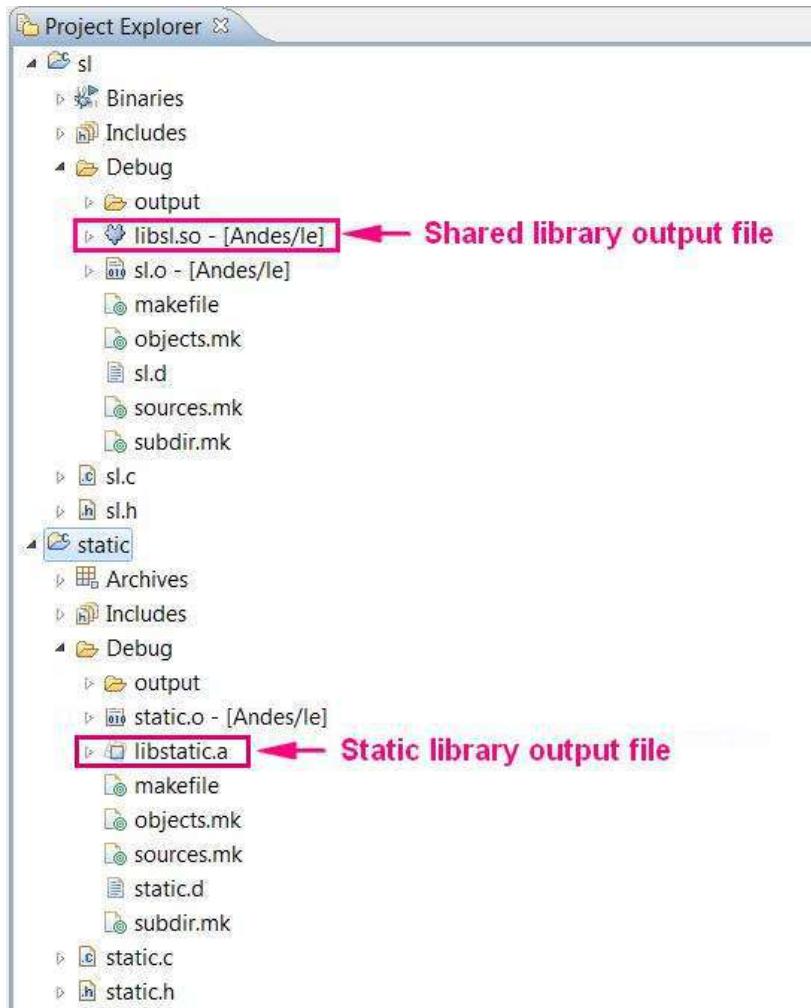
#endif /*SL_H_*/
```

sl.h

```
int sum(int a, int b) {
    return a + b;
}
```

sl.c

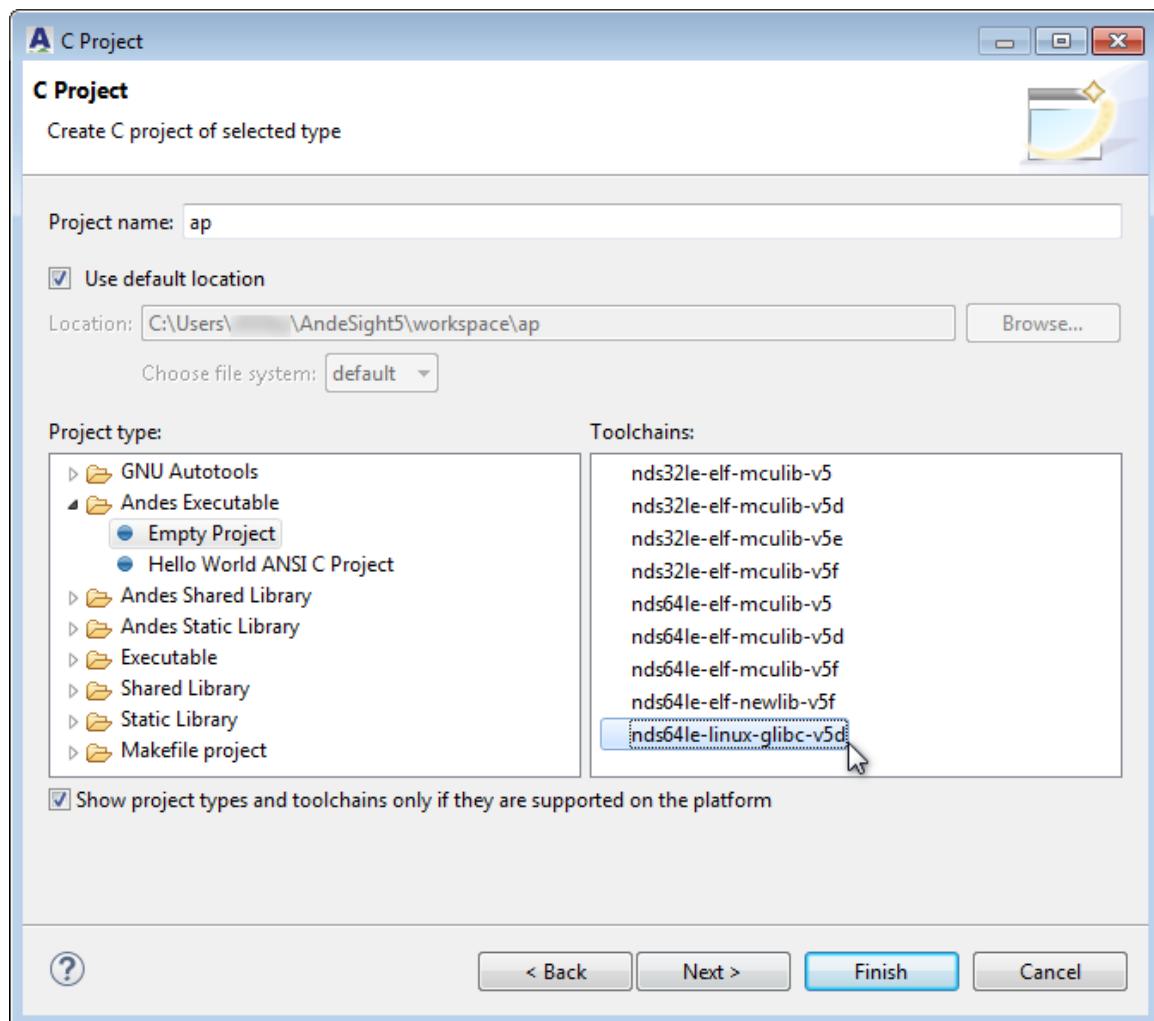
**Step 5** Please refer to Section 2.1.5 to build the project and generate shared library output files (\*.so) or static library output files (\*.a).



## 2.1.8. Creating and building a project that uses shared/static libraries

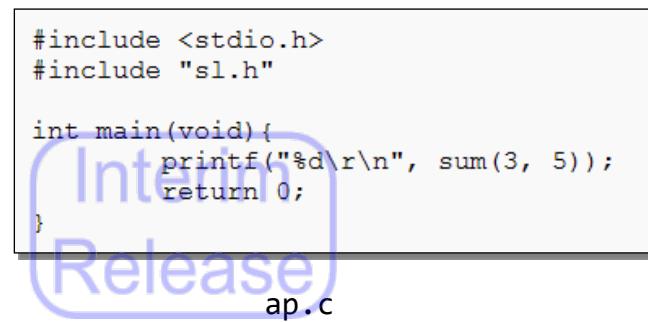
In this section, the shared/static project created in Section 2.1.7 and a new project “ap” are used to demonstrate how to create a project using a shared/static library.

**Step 1** Please refer to Section 2.1.1.2 to create a project for generic targets. At the **C or C++ Project** prompt, be sure to specify a toolchain that matches the toolchain of the shared library project. In the example below, `nds64le-linux-glibc-v5d` (the toolchain of the shared library project “sl”) is specified for the project “ap”.



**Step 2** You can find the created project in the **Project Explorer** view. Please refer to Section 2.1.2.2 to add header files and source files to the project. For example, add the following source file “ap.c” to the

“ap” project.

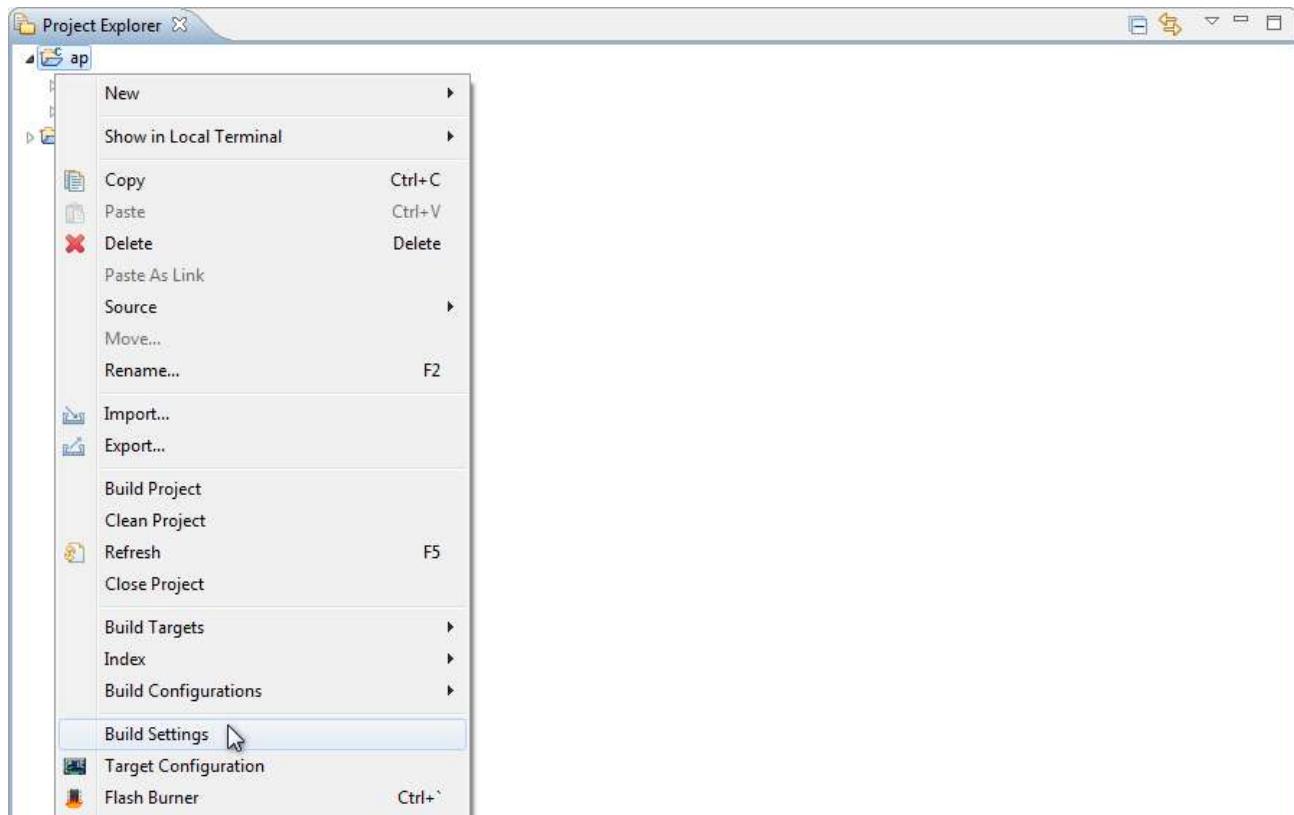


```
#include <stdio.h>
#include "sl.h"

int main(void){
    printf("%d\r\n", sum(3, 5));
    return 0;
}
```

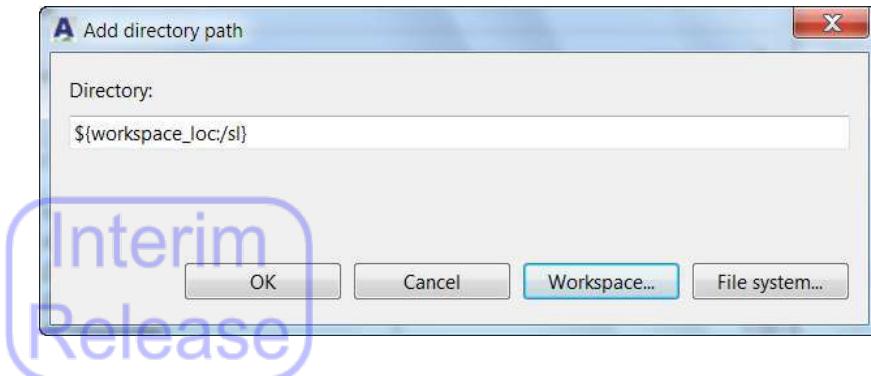
ap.c

**Step 3** In the **Project Explorer** view, right-click the project folder and select “Build Settings” from the pull-down menu.

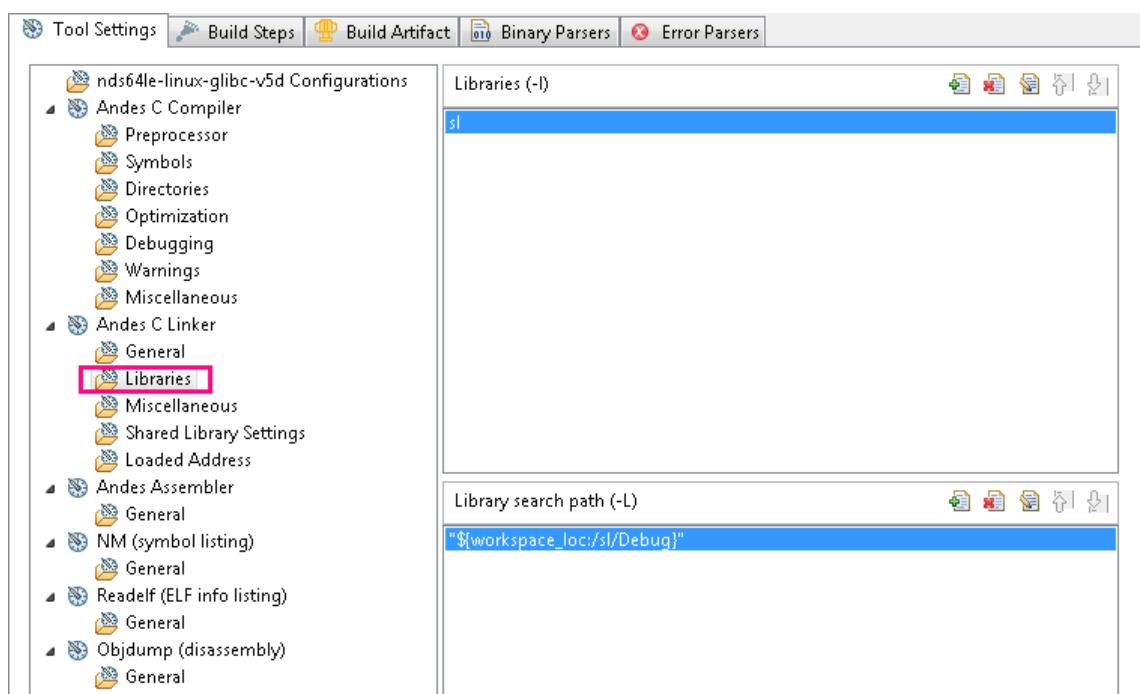


**Step 4** In the **Tool Settings** tab, configure the build settings as follows:

- **Andes C Compiler > Directories:** Click “” (Add ...) to select the shared/static library header file path. For example,

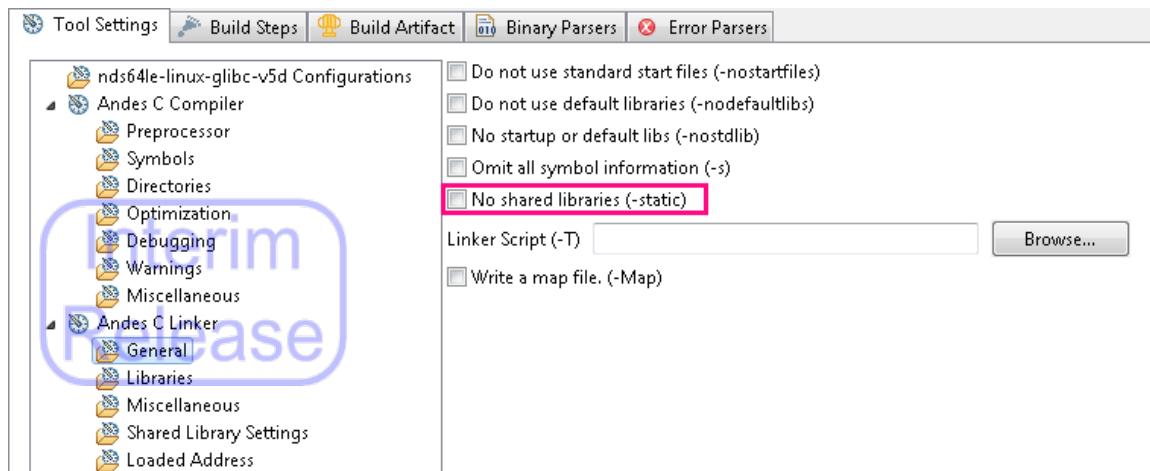


- **Andes C Linker > Libraries:** In the Libraries (-l) column, click  (Add ...) to enter the name of the shared/static library. Under Library search path (-L), click  to specify the path of the shared/static library.



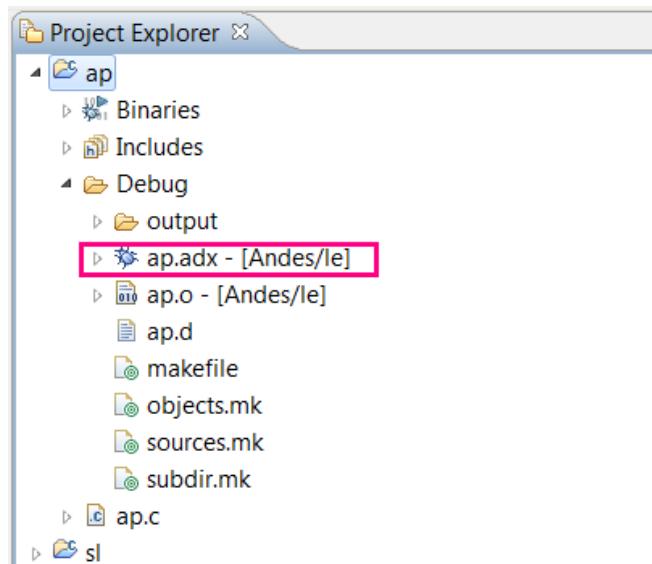
Using shared libraries in a C project requires the following setting as well:

- **Andes C Linker > General:** Uncheck “No shared libraries (-static)”.



Then click “Apply and Close” on the dialog.

**Step 5** Please refer to Section 2.1.5 to build the project and generate an executable/output file.



### 2.1.9. Project migration

The project types “Executable (ANDES32 N9, N10, N12 cross-platform)” available in AndeSight versions prior to v1.4 are no longer supported. AndeSight v5.0 supports project migration only from AndeSight v1.4 to AndeSight v5.0.

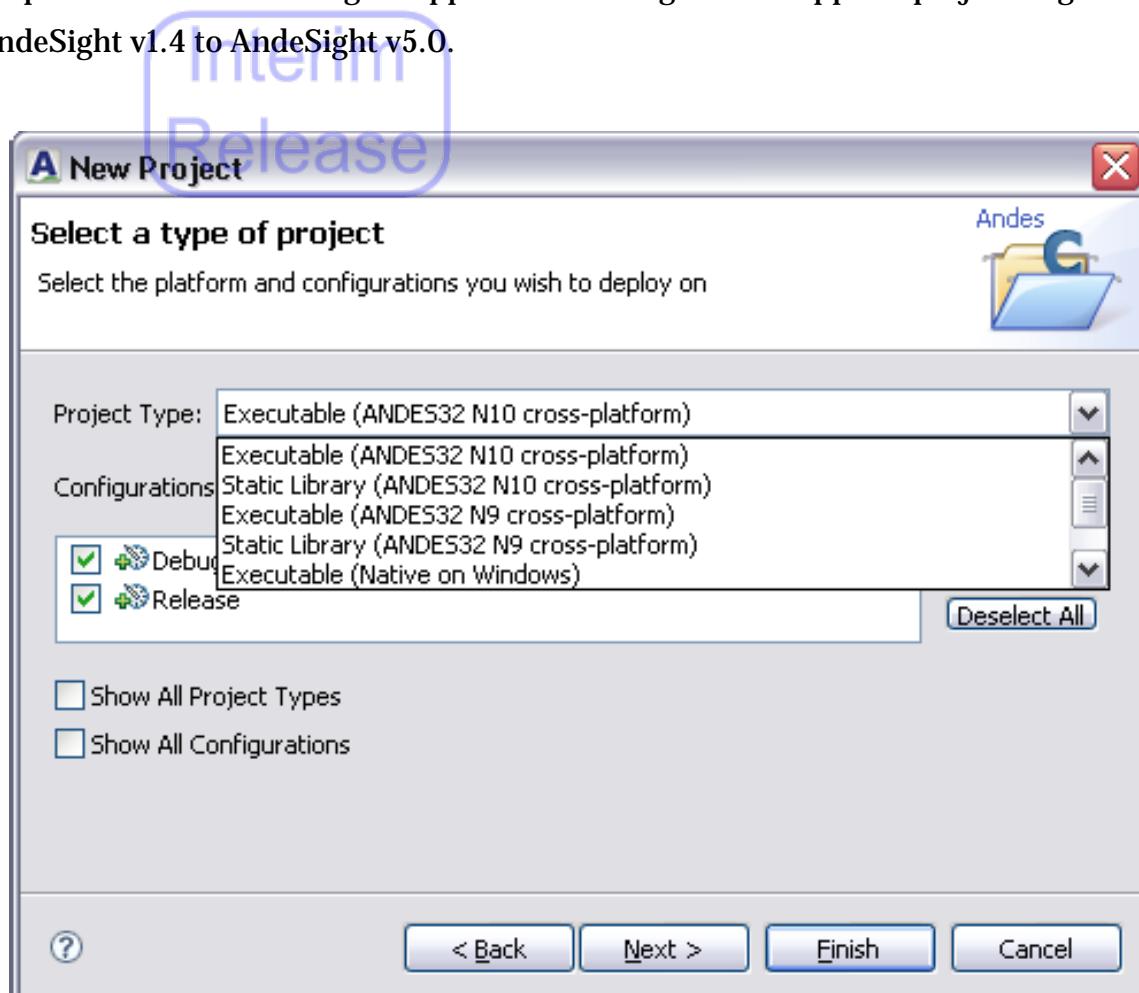
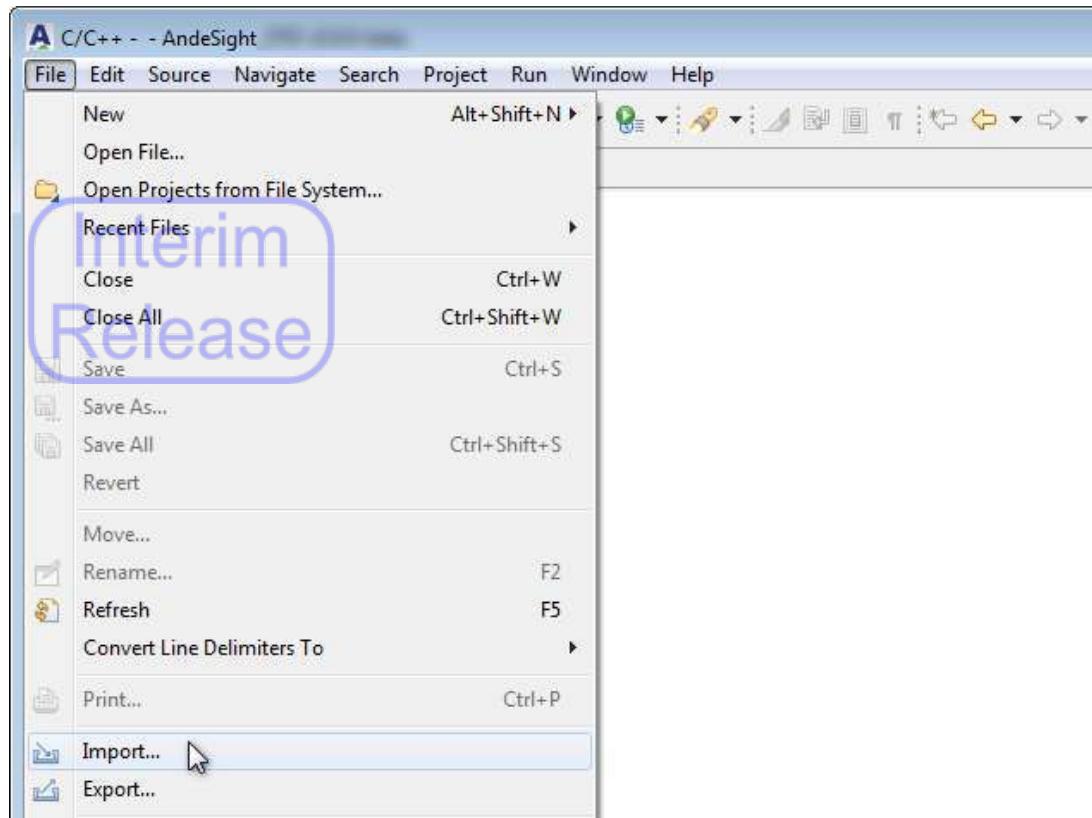


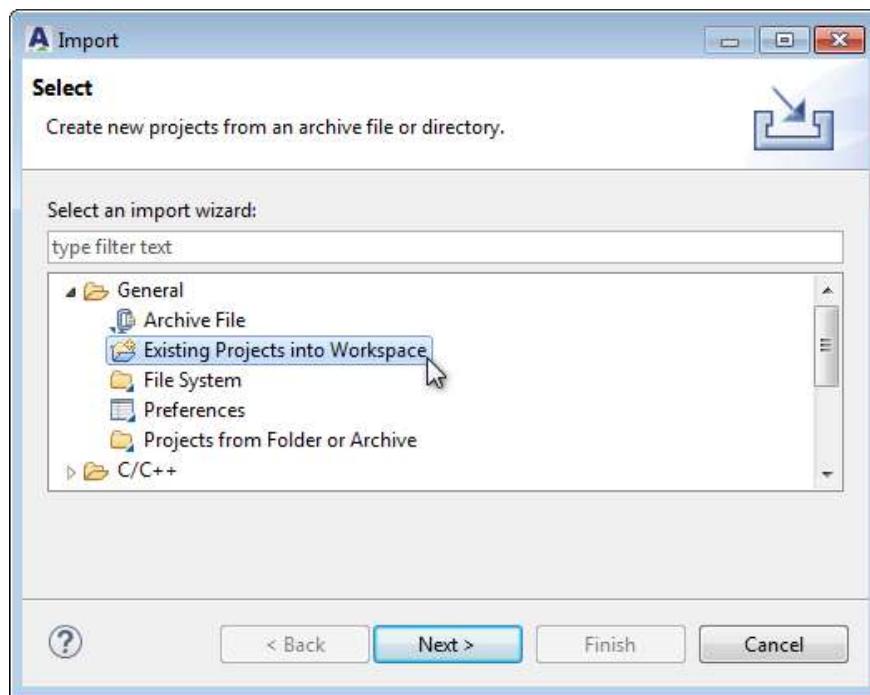
Figure 5. Selecting project type in versions prior to AndeSight v1.4

The importing of an AndeSight v2.1 JPEG project is used as an example to demonstrate the migration process.

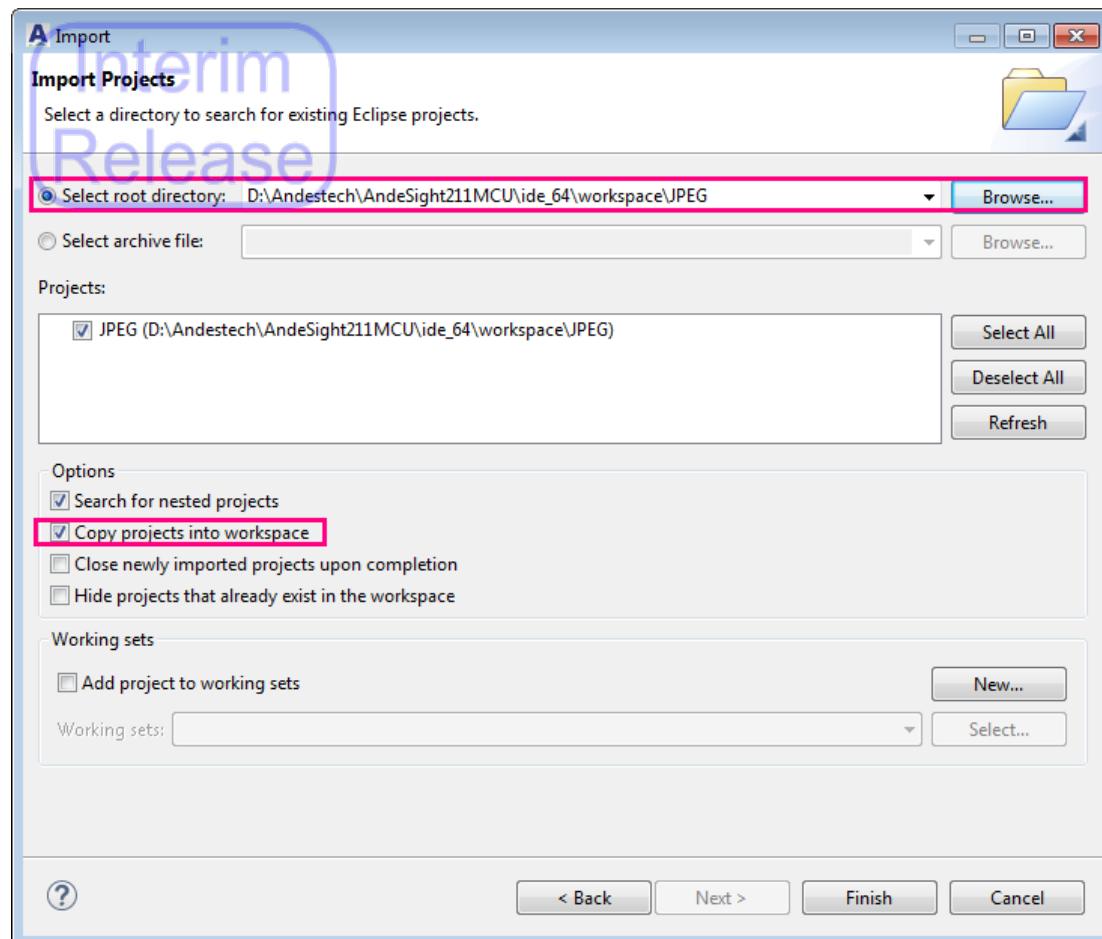
**Step 1** On the AndeSight main menu, select “File > Import...”. You may also right-click anywhere in the **Project Explorer** view and select “Import” on the pop-up menu.



**Step 2** On the **Import** dialog, select “General > Existing Projects into Workspace” and click “Next.”



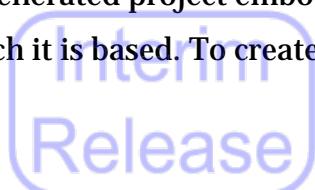
**Step 3** Click “Browse...” to select the AndeSight v2.1 JPEG project. Then, make sure the option “Copy projects into workspace” is selected and click “Finish” to import the project into the current workspace.



**Step 4** The AndeSight v2.1 JPEG project is then imported to the **Project Explorer** view. The target configuration and build settings can be modified as outlined in Sections 2.1.3 and 2.1.4. The project can be built as outlined in Section 2.1.5.

### 2.1.10. Project template

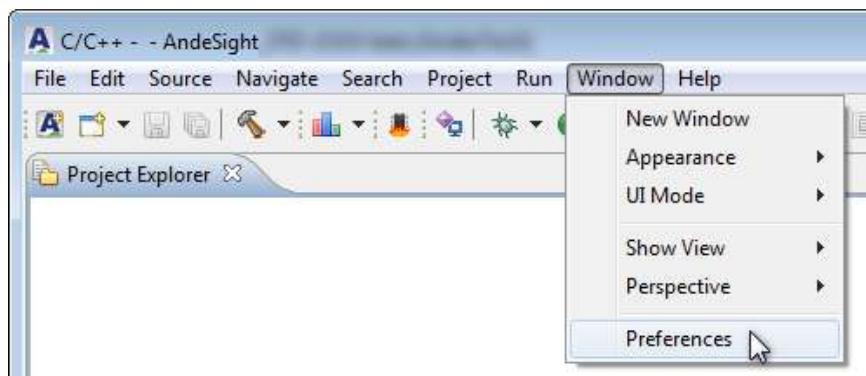
A user-defined project can serve as a template for the creation of subsequent projects. In other words, a newly generated project embodies the source codes and settings specific to the project template on which it is based. To create a project template for subsequent projects, follow the steps below:



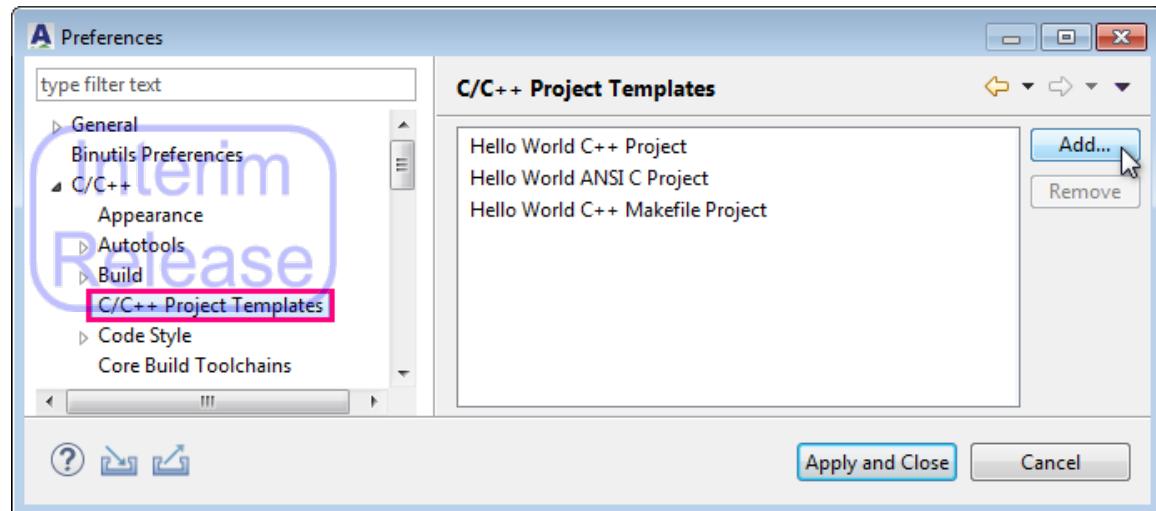
**Step 1** Refer to Sections 2.1.1 to 2.1.2 to create a project and its source/header files. You can locate the created project in the **Project Explorer** view. Refer to Section 2.1.4 to configure the build settings and Sections 2.4.3.1 and 2.4.7.2 for the debug settings. The project “Demo” is presented below as an example.



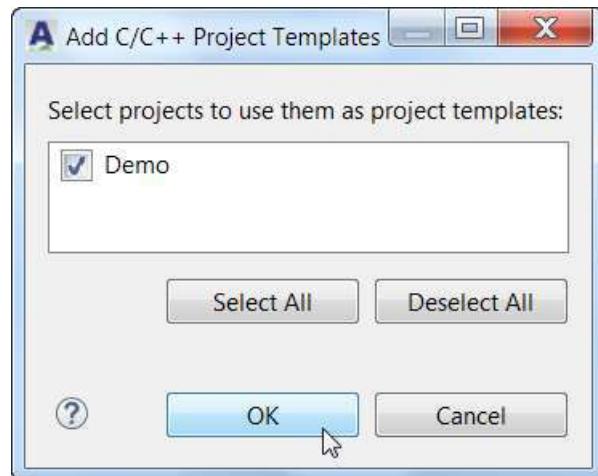
**Step 2** From the AndeSight main menu, select “Window > Preferences.”



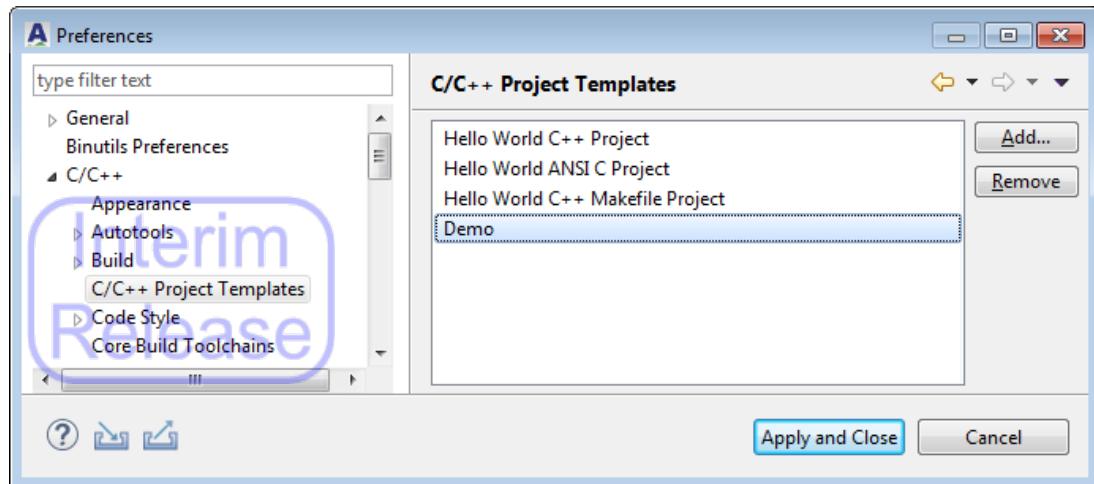
**Step 3** Enter the “C/C++ > C/C++ Project Templates” page and click “Add...”.



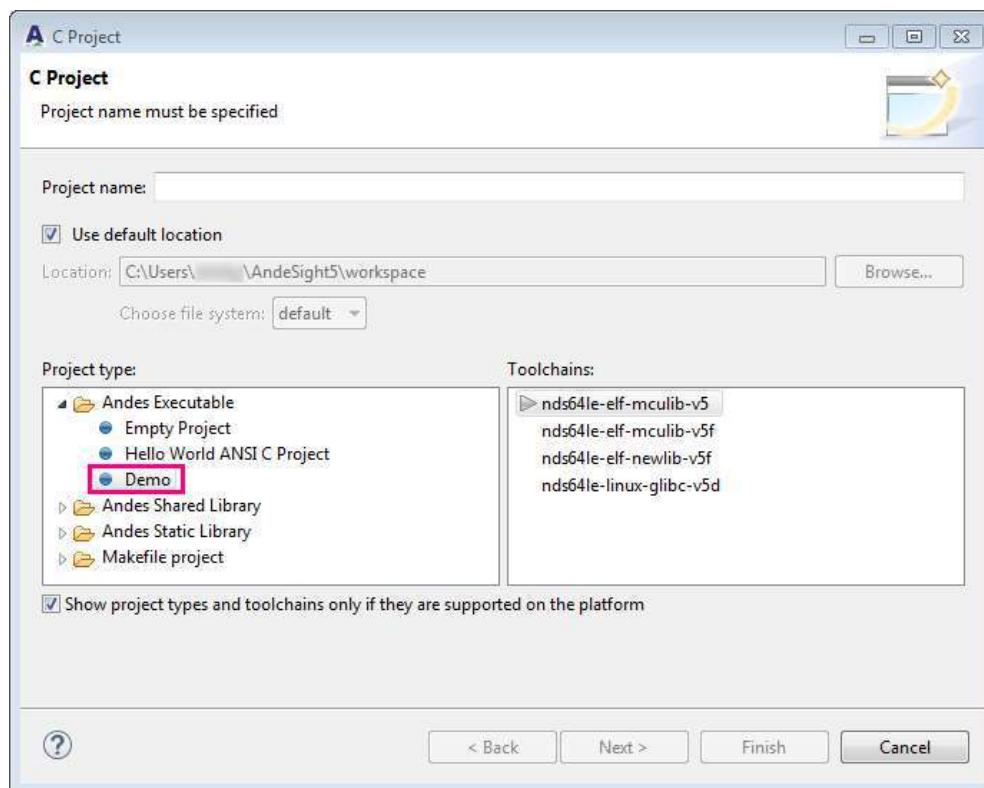
**Step 4** In the Add C/C++ Project Templates wizard, select the project created in Step 1 and click “OK.”



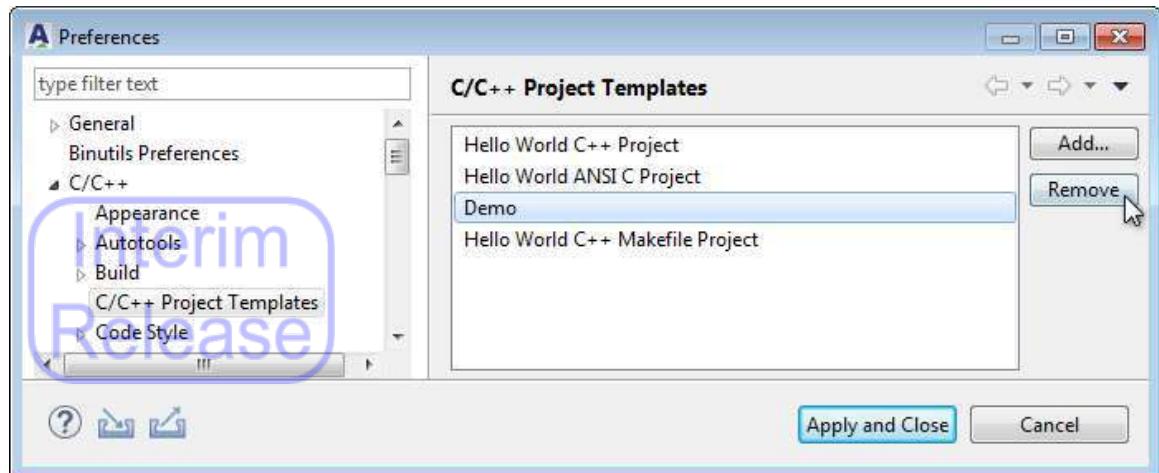
**Step 5** The created project serving as a project template is now listed in the C/C++ Project Templates box. Click “Apply and Close” to exit the Preferences dialog.



**Step 6** You may now select the new project template (“Demo” in the figure below) for the creation of new projects.



**Step 7** You may also remove a project template. Select “Window > Preferences” from the AndeSight main menu and click “C/C++ Project Templates”. Select the unwanted project template on the right and click the “Remove” button.



### 2.1.11. Restoring a project to its default settings

Projects created using a project template have default build and debug configuration settings.

You can refer to Section 2.1.4 to change the build settings and Section 2.4.3.1 and 2.4.7.2 to change the debug settings. You can also restore the changed configurations to its default settings as follows:

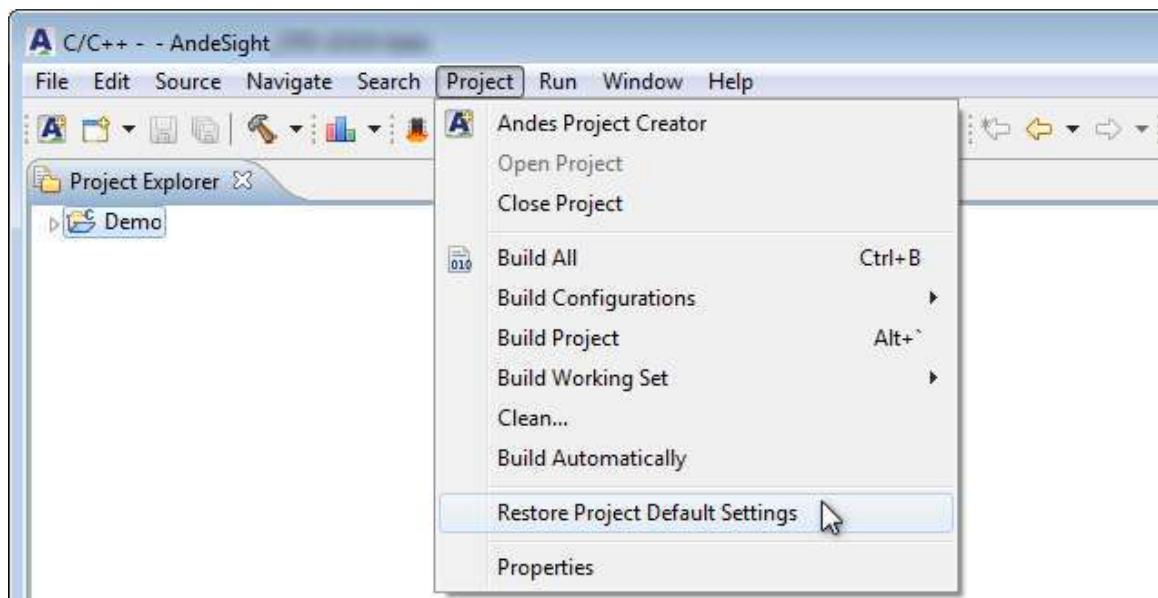
---

**NOTE**

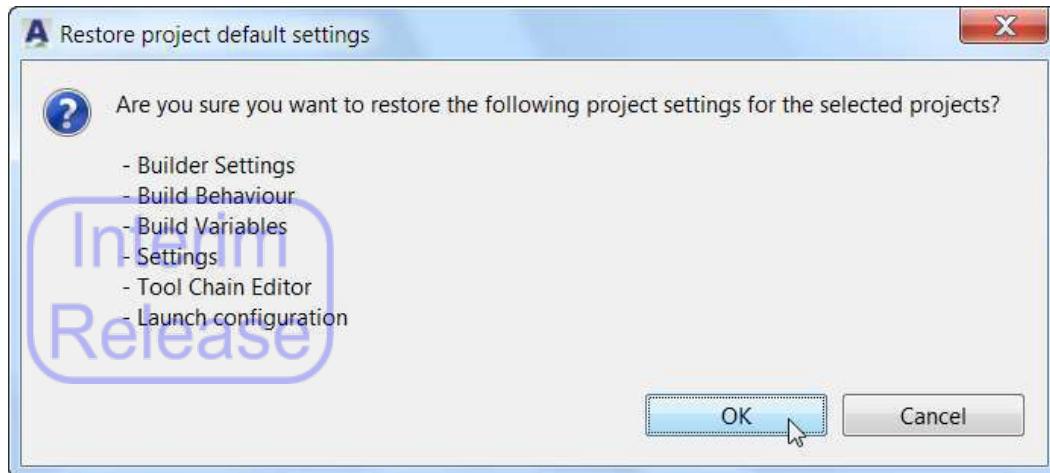
Default debug configuration settings are defined in the launch configuration file (\*.launch) specified in a project template. To generate a launch configuration file for your project template, refer to Section 2.4.3.1, “Common tab”.

---

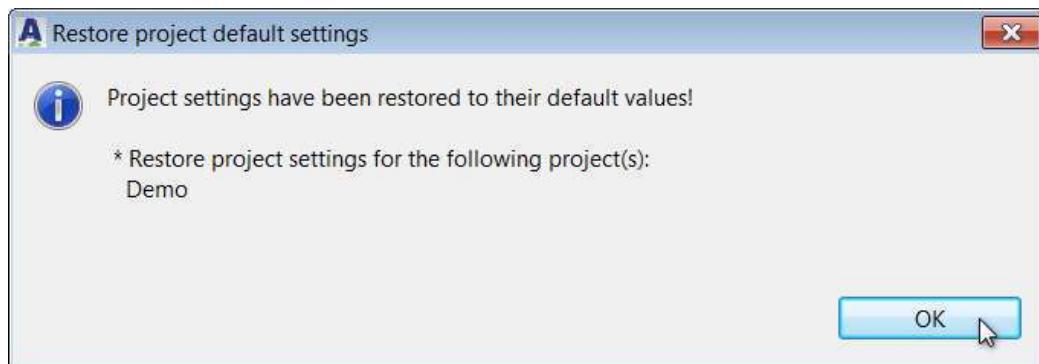
**Step 1** Select the desired project folder in the **Project Explorer** view and click “Project > Restore Project Default Settings” on the AndeSight main menu.



**Step 2** A prompt indicating which settings will be reverted to default values is displayed. Click “OK” to continue.



**Step 3** Verify the restoration results from the popup dialog. Click “OK” to complete the process.



## 2.2. Target management

In the AndeSight IDE, program development (e.g., running, debugging, and/or profiling) deals with either a predefined target or a generic target. The two target systems are introduced below:

- Pre-defined Targets: A pre-defined target is defined by a chip profile, which contains the specifications of a specific chip/board/product. These targets in AndeSight can be found in the Chip Profile list under the **Andes Project Creator** view as well as in the “Target” groups of the **Target Manager** view. In addition to these targets based on Andes cores, you can define your own targets by creating chip profiles for your ready-made chips/boards/products (see Sections 2.2.1.1 and 2.2.1.2 for more details).
- Generic Targets: The configuration of a generic target is based on the Andes Instruction Set Architecture, endian type, library, and executable file format. The generic targets available in AndeSight are listed in the “Generic Targets” group of the **Target Manager** view. These targets depend on the installed toolchains. Choosing a generic target for your C or C++ project means that any target that supports the project toolchain can be launched in a debug session.

Open the **Andes Project Creator** view to see all of the pre-defined targets for program development. They are grouped according to platform in the **Chip Profile** section. Expand the group tags (i.e., AE250, AE350, and Corvette) to identify the chip profile of the respective target.

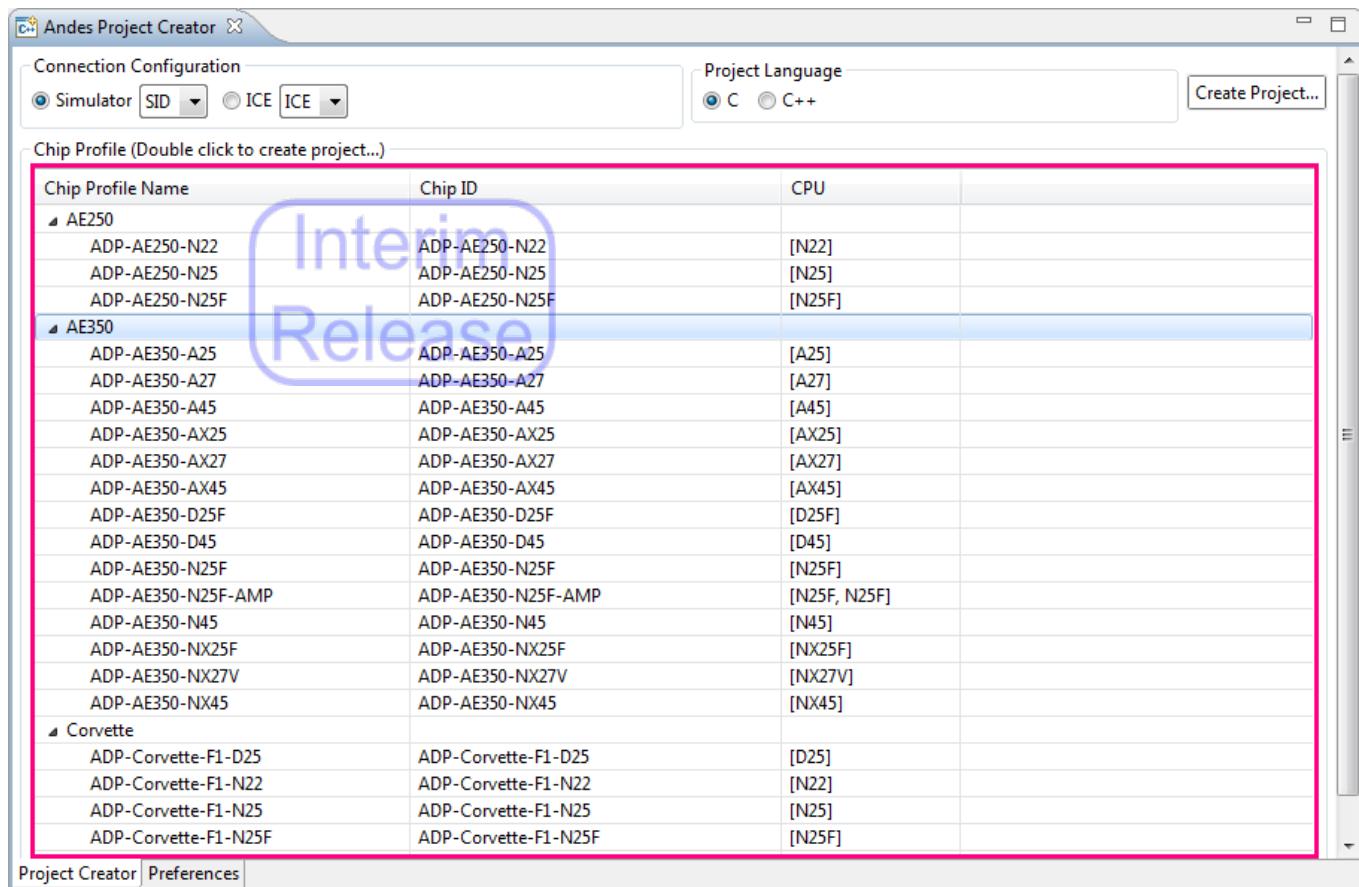


Figure 6. Predefined targets in AndeSight STD v5.0

Each target can be connected to a local debugging host through an ICE device or a simulator, depending on the connection configuration to which it is assigned. A target can also be connected to a remote host as long as a connection configuration is available. To examine the prepared connection configurations in the IDE, select “Windows > Preferences” on the AndeSight main menu to invoke the **Preferences** dialog and click “Target Management Default Settings > Connection Configurations” in the navigation pane to view the **Connection Configurations** page. Refer to Section 2.2.4.1 to edit an existing connection configuration or create a new one if needed.

Project creation using pre-defined targets is different from that using generic targets. Projects using pre-defined targets are created via the **Andes Project Creator** view (Section 2.1.1.1) and those created using generic targets follow the regular C/C++ project creation flow (Section 2.1.1.2).

---

**NOTE**

Andes FreeStart program provides two types of N22 RTL: fully-configurable N22 and fixed-configuration N22. The chip profile “ADP-Corvette-F1-N22”, though shared among Corevette-F1 targets pre-integrated with configurable/unconfigurable N22 RTL, contains settings mainly for fully-configurable N22. To execute your project on a Corvette-F1 target with fixed-configuration N22 RTL in Andes FreeStart program, you’ll have to specify the chip profile “ADP-Corvette-F1-N22” and then change the toolchain to nds32le-elf-[newlib|mcuLIB]-v5e. If the “SID” connection configuration is intended for the project, be sure to also specify the following simulator arguments:

```
-e "set cpu cpu-option \\\" --conf-i sa-e on --conf-pmp 0 --conf-pfm off  
--conf-powerbrake off --conf-i cache off --conf-user-mode off --conf-mul radixx4  
--conf-btb off \\\""
```

For details about changing the toolchain for a selected chip profile and specifying simulator arguments for a project, please refer to Section 2.1.3, Step 2 and Step 3.

---

The AndeSight IDE manages target connections to a local host (via an ICE device or a simulator) by means of the **Target Manager** view (Section 2.2.5). The typical approach to target management in AndeSight is outlined in the following:

**Step 1** Set up the target system on which the program is to run.

**Step 2** Create a project using a pre-defined target in the **Andes Project Creator** view (Section 2.1.1.1), or create a project using a generic target adhering to the regular C/C++ project creation flow (Section 2.1.1.2).

**Step 3** Build an executable for the project (Section 2.1.5).

**Step 4** Run/Debug/Profile the desired program. The target will be initiated automatically and managed via the **Target Manager** view.

The flow of target management in AndeSight is illustrated in the figure below.

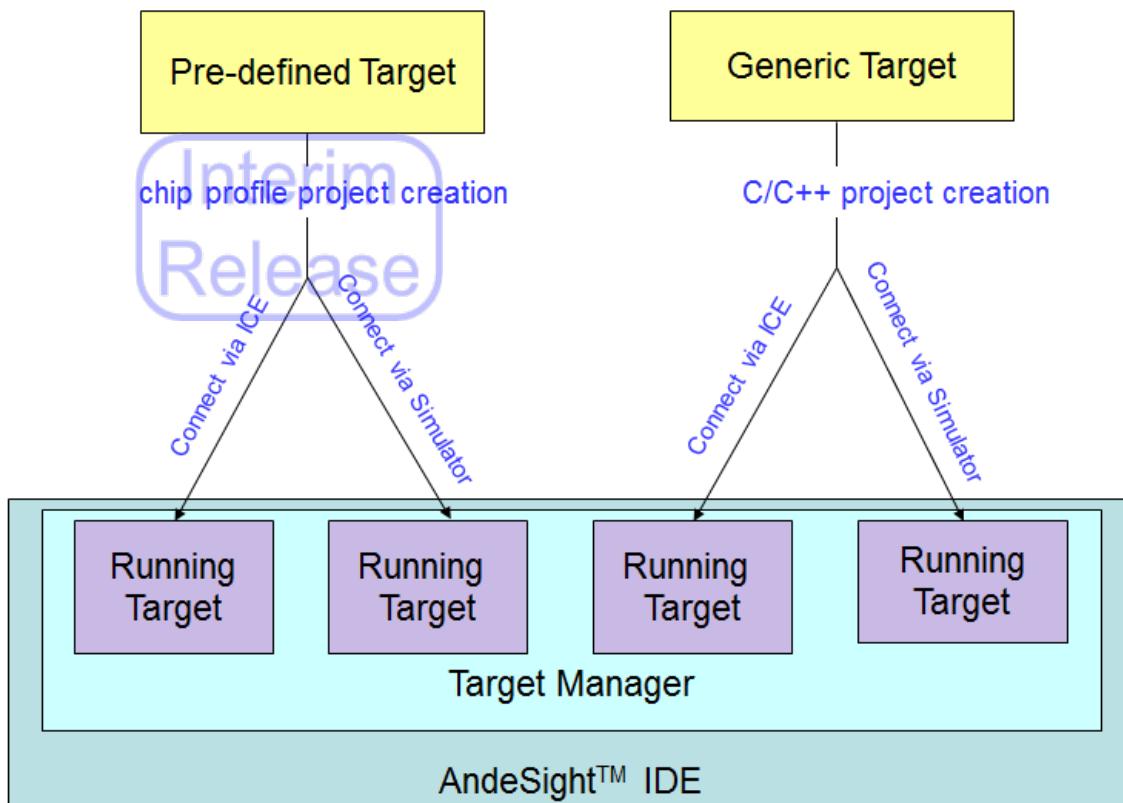


Figure 7. Flowchart of target management in AndeSight

### 2.2.1. Chip profiles for defining targets

To facilitate program development in the AndeSight IDE, chip profiles can be created to define your chips/boards/products as targets. A chip profile contains all of the software settings necessary for a ready-made chip or chip prototype. These settings include the chip name, toolchain, flash driver, SoC register file, CPU register file, memory map, linker script, and simulator configuration. Once a chip profile has been created, the defined target appears in the **Andes Project Creation** view similar to a project creation short-cut. This can save considerable time in the creation of projects sharing the same target.

In AndeSight, a chip profile is packed with associated software setting files in a folder named by the target under **ANDESI GHT\_ROOT\target**. In some cases, these files may be delivered as a binary file with the .ext file extension.

A target can be defined in a chip profile by manually writing a targetboard.properties file (Section 2.2.1.1) or taking advantage of the chip profile editor (Section 2.2.1.2). Note that the defined target will not function properly if the toolchains or software settings files (such as the SoC register file) designated in the profile are unavailable. Thus, be sure to install the specified toolchain(s) and prepare the associated files along with the chip profile.

### 2.2.1.1 Defining a target using a targetboard.properties file

**Step 1** Create a directory under the “target” directory of AndeSight:

```
%mkdi r ANDESIGHT_ROOT/target/TARGET.
```

The **TARGET** directory is where chip profiles should be located. In naming the **TARGET** directory, you are designating the chip/board/product you want to define. The characters in the name may include the following: [A-Z,0-9,-,\_].

**Step 2** Write a targetboard.properties text file under the directory **ANDESIGHT\_ROOT\target\TARGET**. You may reference **README.txt** under **ANDESIGHT\_ROOT\target** for the file format or use the targetboard.properties file of a pre-defined target in the same folder as the template. The following exemplifies the format with the targetboard.properties file of the chip profile ADP-AE350-AX25.

```
#==== CPU Description ====
cpu.cores=1
cpu0.type=AX25
cpu0.isa.reduceregs=N
cpu0.registers=../../CPU/V5-64.crgs
#cpu0.c_compiler_opt=
#cpu0.cpp_compiler_opt=
#cpu0.linker_opt=

#==== SOC Registers Description File ====
soc.registers=../../SoC/AE350-4GB regs

#==== Memory Map Description File ====
memory.map=../../MemoryMap/AE350-4GB-memory.mem

#==== Preferred Tool chain ====
preferred.tool chain=nds64le-elf-mculib-v5

#==== Linker Script File ====
#linker.script=
```

```

==== Flash Burner Settings ====
flash.driver=target_burn_frontend
flash.driver.custom.ui=N
flash.target=ae350
flash.start.address=0x0
flash.controller.address=
flash.misarguments=
flash.target.burn.bin=target_SPI_v5_64.bin

==== ICE miscellaneous arguments ====
iceman.misc.args=-Z v5

==== Simulator Configuration File ====
vep=../../vep/tmp/ADP-AE350-AX25.vep

==== VEP Front-End Filter ====
#vep.vepiofilter=SDC:CFC

==== Simulator VIOS/Raw mode ====
vep.vepiomode=RAW

==== Simulator miscellaneous arguments ====
#sid.misc.args=

==== Removable (Y/N) ====
#removable=Y

==== vep2conf arguments ====
vep2conf.misarguments=-a v5

```

---

**NOTE**

1. Arguments for connection configurations can also be specified in the targetboard.properties file. AndeSight comes with two default connection configurations, “SID” and “ICE”, and their arguments are specified using the property names “sid.misc.args” and “iceman.misc.args” respectively. As to arguments for other connection configurations, they must be defined using a property

name in the following format:

**Configuration name + Connection type (simulator/ice) + misc. args**

The configuration name in the format is case insensitive. For example, the following defines arguments for a simulator connection configuration “QEMU”:

**qemu.simulator.msc.args=--machine andes\_ae350 256M -nographic -S**

2. In addition to Andes-defined .regs files, AndeSight chip profiles also accepts SoC registers files of CMSIS System View Description format (\*.svd). To define a .svd file for a target in a targetboard.properties file, you will need to specify both the CMSIS-SVD file and the svd format with properties “**soc. registers**” and “**soc. registers.type**” in the SoC Registers Description File section. For example,

...

```
#== Soc Registers Description File ==
soc. registers=. /STM32F21x.svd
soc. registers.type=svd
```

...

---

- Step 3** The files pertaining to software settings that are specified in the targetboard.properties file (e.g., SoC register file or Memory Map file) must be prepared in the same directory for the sake of file packing. The file structure of the pre-defined targets in AndeSight should be as follows:

#### **ANDESIGHT\_ROOT**

```
`-- target
  |-- TARGET_1
  |   |-- targetboard.properties
  |   `-- ASSOCIATED_FILES (e.g. SoC register file, Memory Map file ...)
  |
  |-- TARGET_2
```

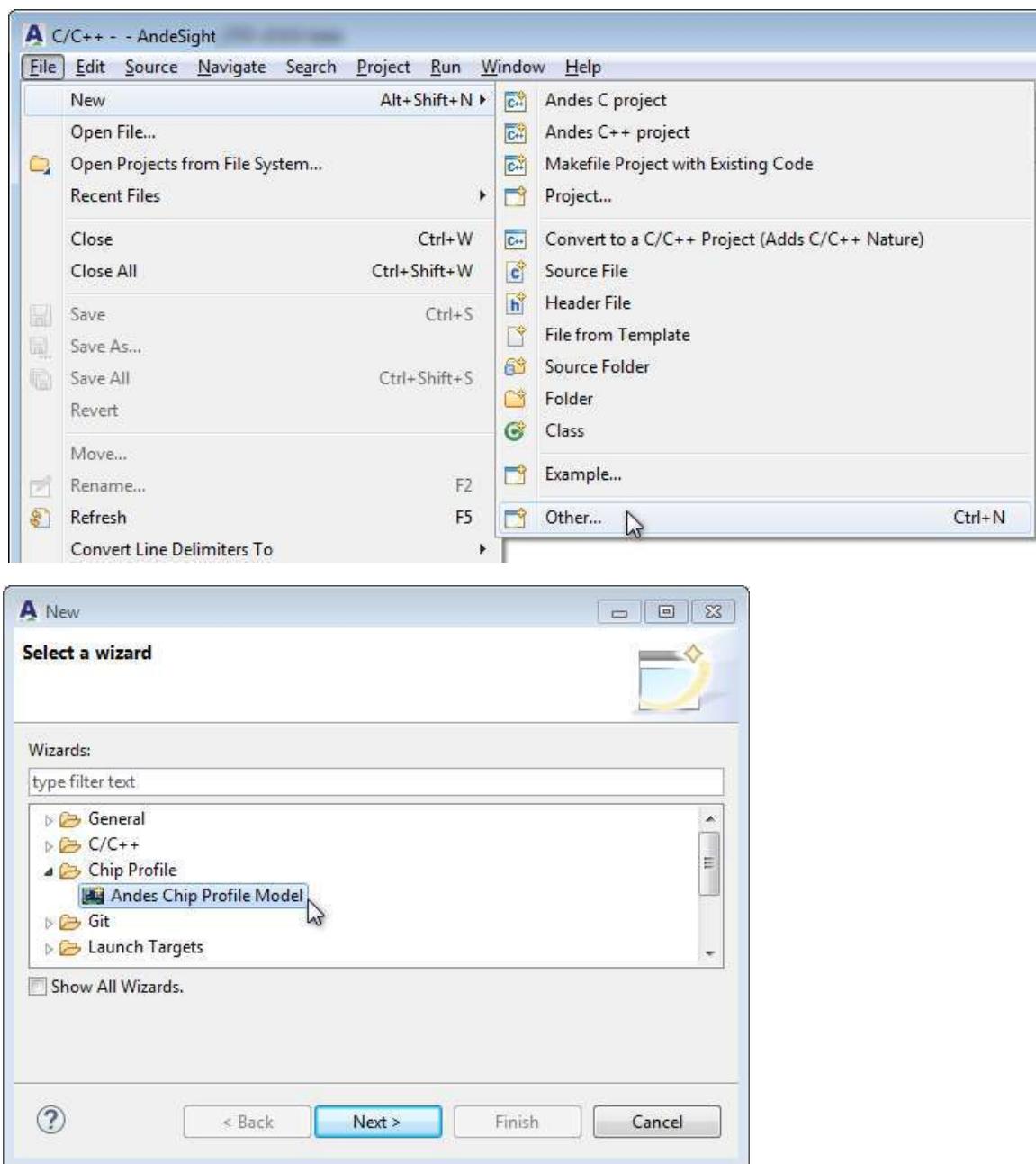
```
|   |-- targetboard.properties  
|   '-- ASSOCIATED_FILES (e.g. SoC register file, Memory Map file ...)  
|
```



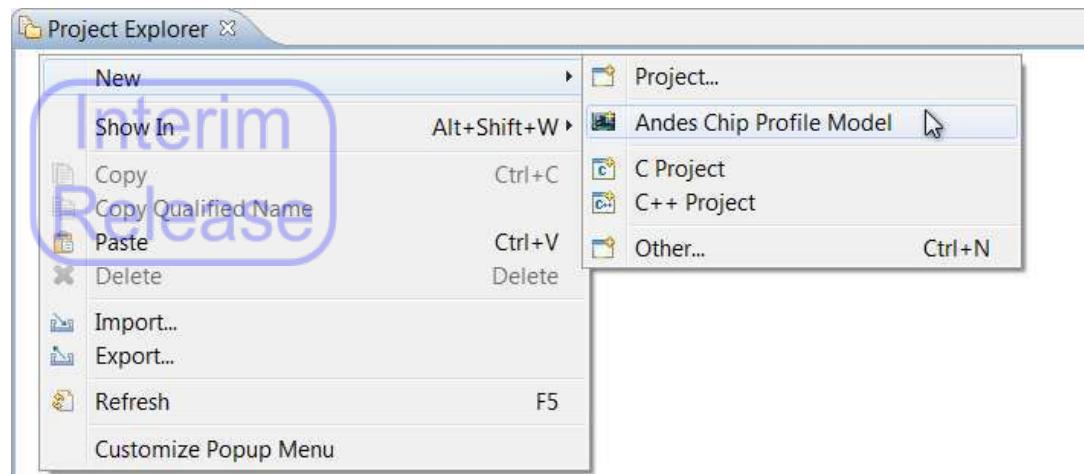
### 2.2.1.2 Defining a target using the chip profile editor

The chip profile editor in the AndeSight IDE is an easy-to-use graphical interface used in creation or modification of chip profiles for defining or configuring targets. The following gives step-by-step instructions on the creation of chip profiles using the chip profile editor.

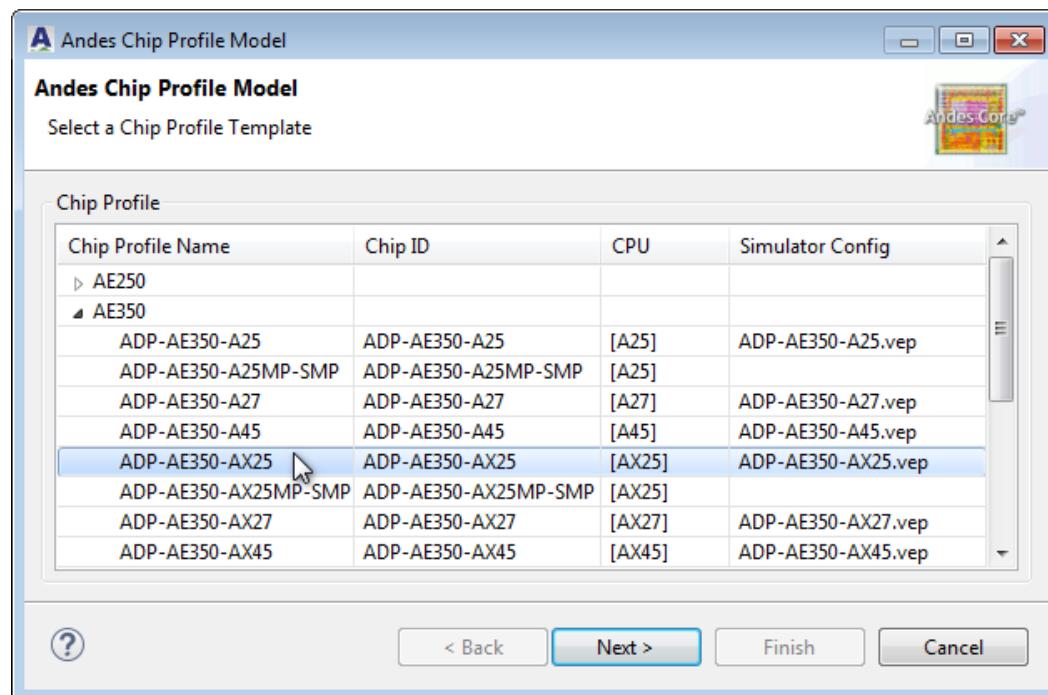
**Step 1** On the AndeSight main menu, select “File > New > Other ...”. On the invoked wizard, select “Chip Profile > Andes Chip Profile Model” and click “Next.”



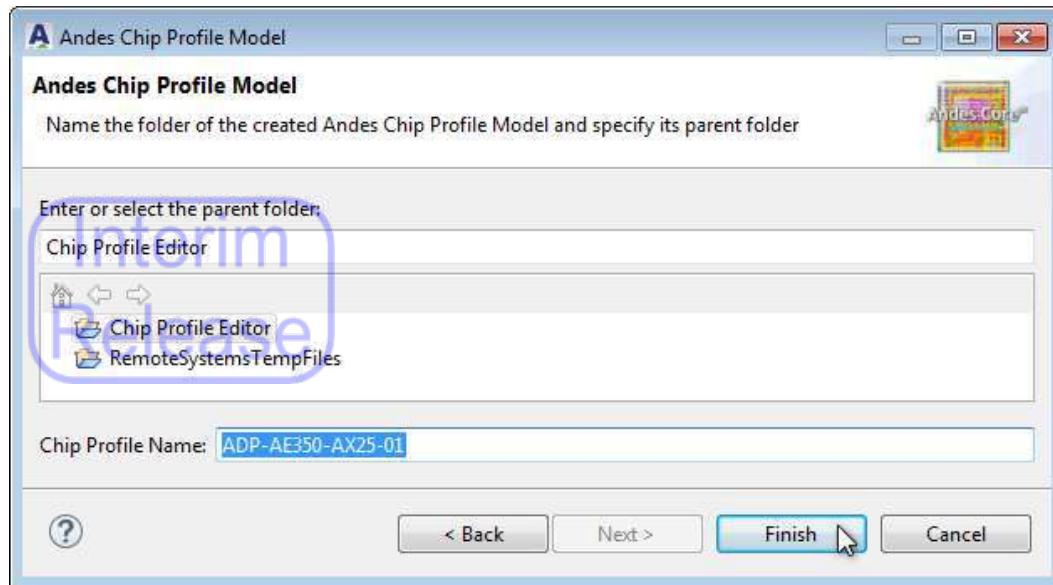
You may also right-click anywhere in the **Project Explorer** view and select “New > Andes Chip Profile Model” on the pop-up menu.



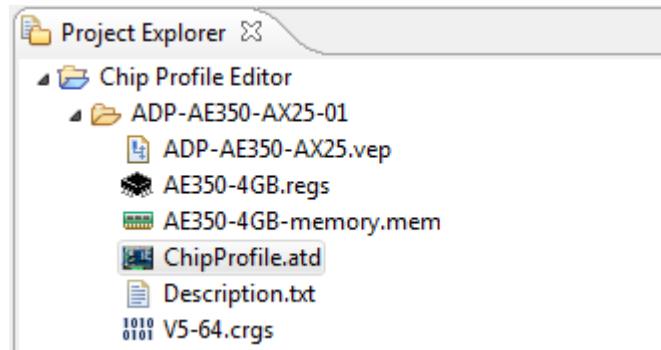
**Step 2** At the **Andes Chip Profile Model** prompt, select a reference chip profile and click “Next.”



**Step 3** Enter a name for your target in the “Chip Profile Name” field and click “Finish.”



**Step 4** You can find your target in the **Project Explorer** view. The chip profile and associated software setting files can be found in the expanded tree. You can also manually copy software setting files to the chip profile folder in **Project Explorer**.



A chip profile and associated files can be distinguished by their file extensions. A file can be modified by double-clicking it to invoke the corresponding editor. A summary of these files is presented in the table below.

Table 5. Chip profile and associated files used in defining a target

File attribute	File extension	Function of editor
Chip Profile Properties file	atd	The editor appears in the <b>Properties</b> view and is used to configure chip profile options (e.g., preferred toolchain and project template).
SoC Registers file	regs	The editor allows configuration of name, type, width, descriptions address, values, reset values, masks, and bit field values of SoC registers. The configuration can be examined in the <b>SoC Registers</b> view during a debug session (see Section 2.4.2.8).
	svd	AndeSight chip profiles also allow SoC registers files in CMSIS-SVD format. However, .svd files are read-only in the editor and can't be modified.
CPU Registers file	crgs	The editor allows you to configure descriptions of CPU registers. The configuration can be examined in the <b>CPU Registers</b> view during a debug session (see Section 2.4.2.9).
Memory Map file	mem	The editor allows configuration of address ranges, usages, access widths (see Step 6), cacheability, and descriptions of memory regions. The configuration can be examined in the <b>Memory Map</b> view during a debug session (see Section 2.4.2.4).
Virtual Evaluation Platform file	vep	Though vep files define the settings of assembled IP models, its editor, called

File attribute	File extension	Function of editor
		<b>CPU Dialog</b> , only allows you to configure CPU architecture settings. For detailed introduction to the configuration settings, please refer to the CPU Dialog part at the end of this section.
Description file	txt	The editor allows you to include information about the target, which then appears as a tooltip in the <b>Andes Project Creator</b> view.

**Step 5** Double-click ChipProfile.atd. The configurable options of the chip profile appear in the **Properties** view. Configure the desired settings in table cells (i.e., **Value** fields).

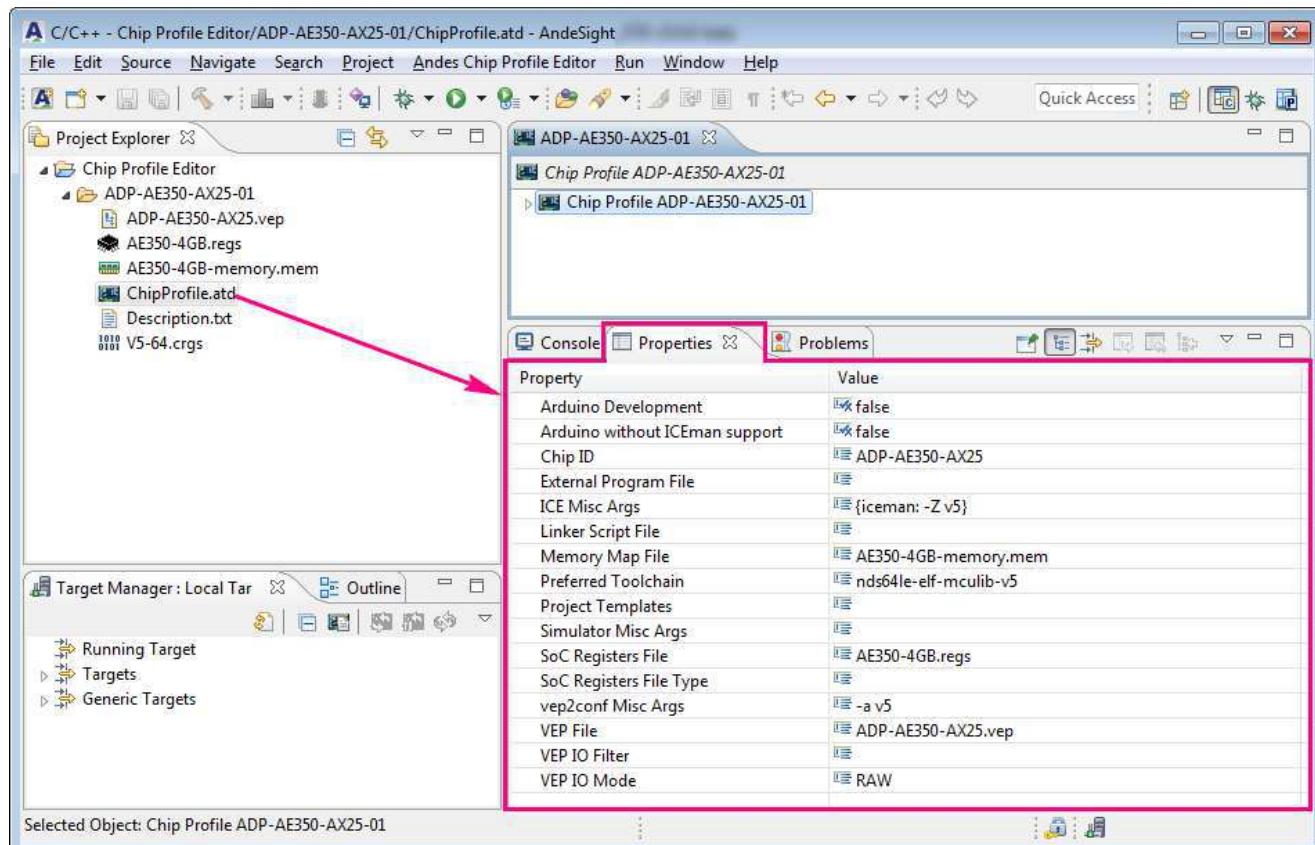
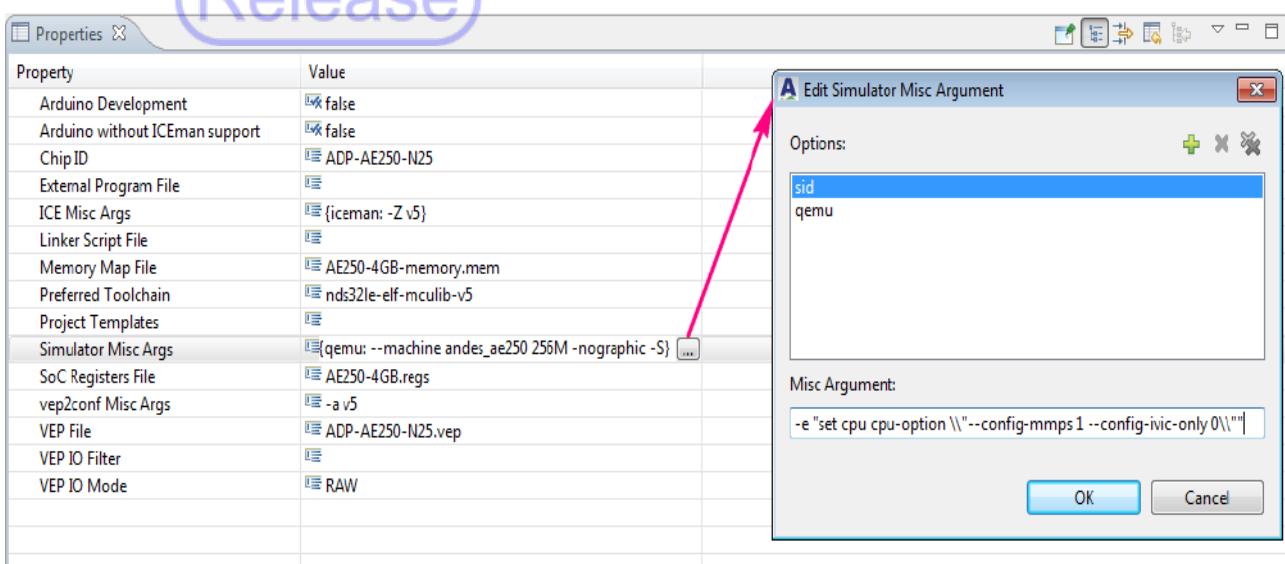
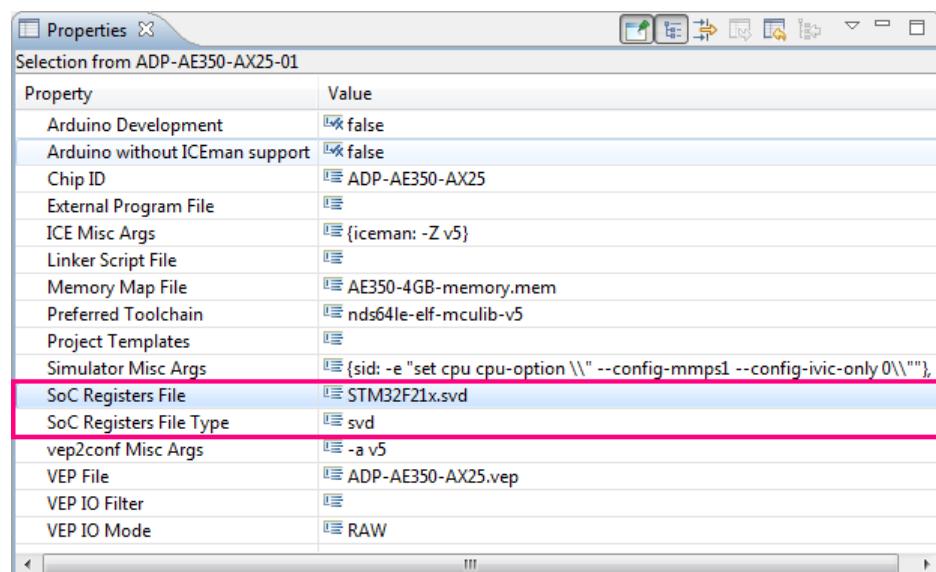


Figure 8. Chip profile properties file and the corresponding editor

Among the configuration options of ChipProfile.atd, “ICE Misc Args” and “Simulator Misc Args” are for you to specify arguments for certain connection configurations, including the default ones “ICE” and “SID” in AndeSight. Just click their value field to invoke the **Edit Simulator/ICE Misc Argument dialog** and edit arguments for specific connection configurations there.



Also note that Andes chip profiles allow SoC registers files in either Andes-defined .regs or CMSIS-SVD .svd format. To select a .svd file for a chip profile, be sure to specify both the CMSIS-SVD file and its format in the editor of chip profile properties, as shown below.



Double-clicking ChipProfile.atd also invokes a view with a name matching the target. The new chip profile in the view can be expanded to show the CPU settings and flash driver settings. Selecting them reveals their configurable options in the Properties view. Configure the settings directly in the Value fields.

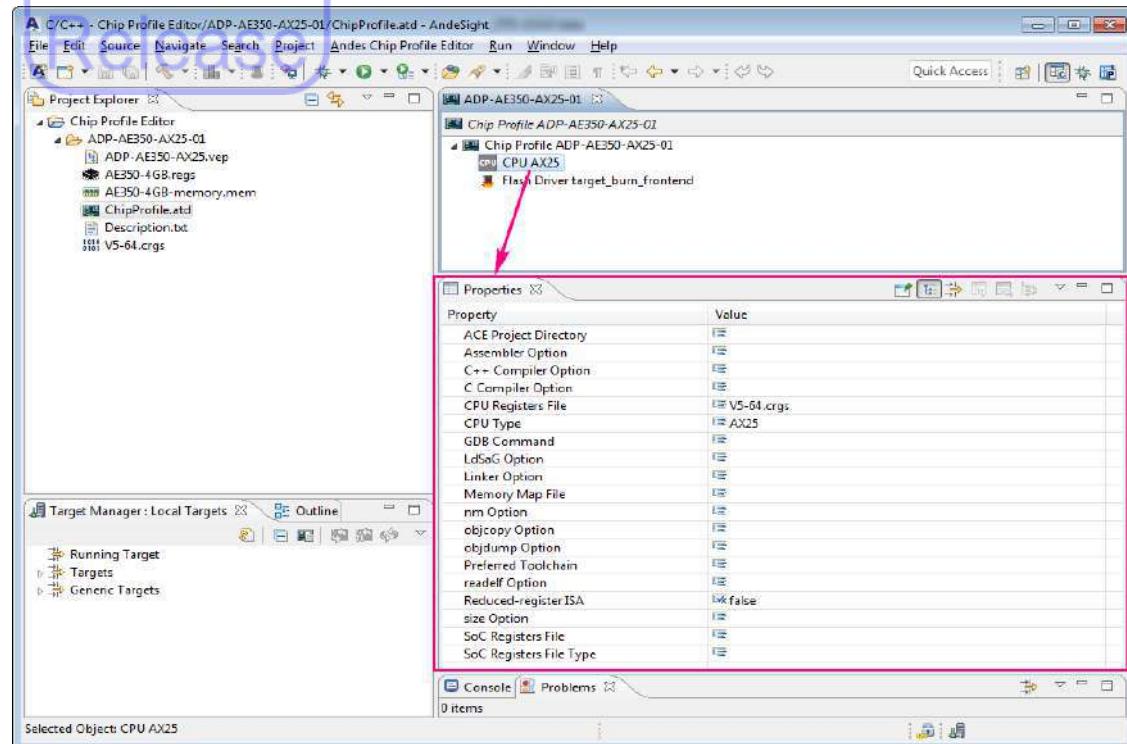


Figure 9. CPU settings in a chip profile editor

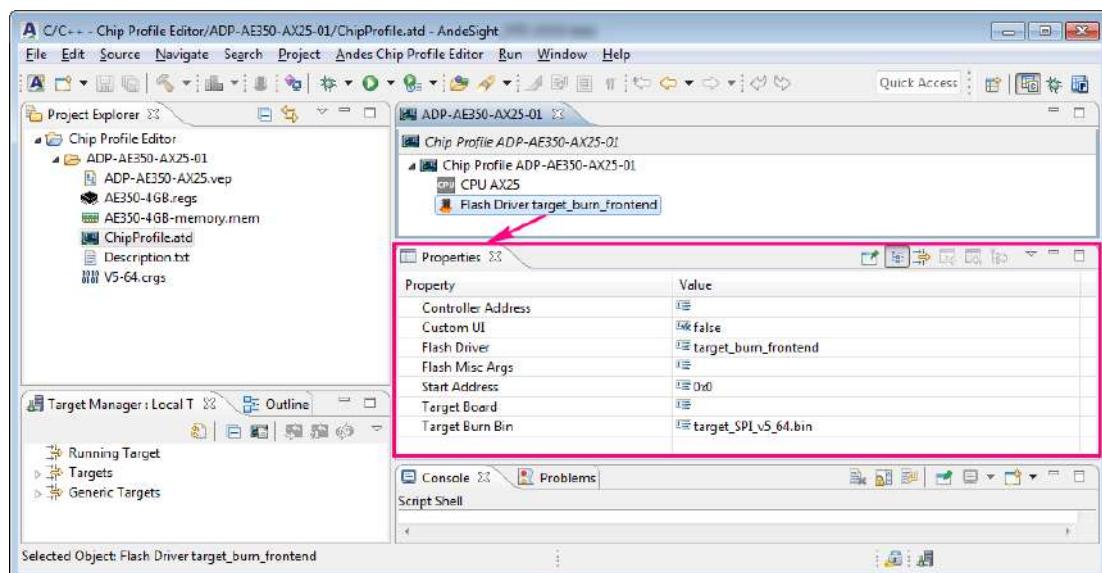
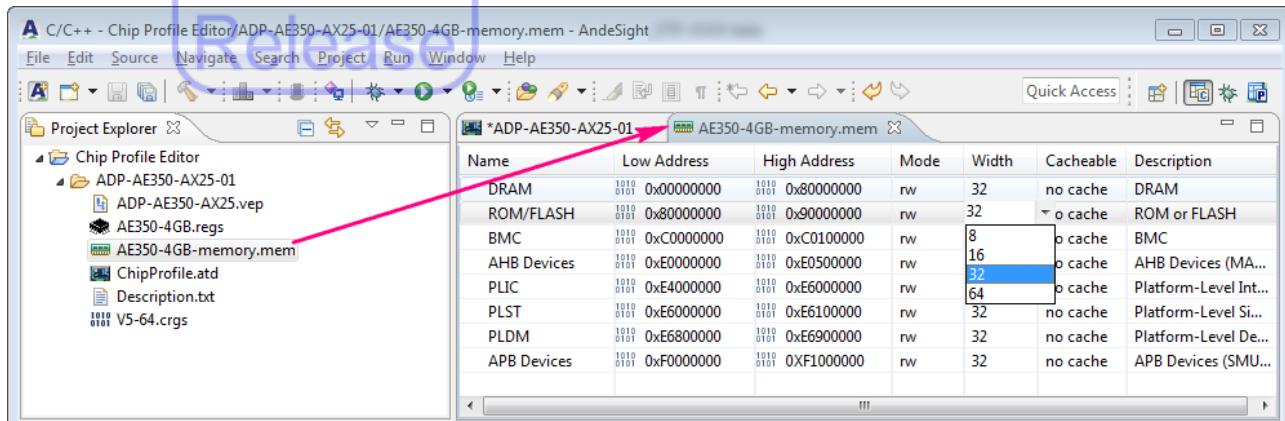


Figure 10. Flash driver settings in a chip profile editor

**Step 6** In Project Explorer, double-click the files associated with the target (SoC registers/CPU registers/Memory Map/VEP/Description file) and work through the configuration in the invoked views. For example, you can configure the access width of memory-mapped device registers in the Memory Map Editor:

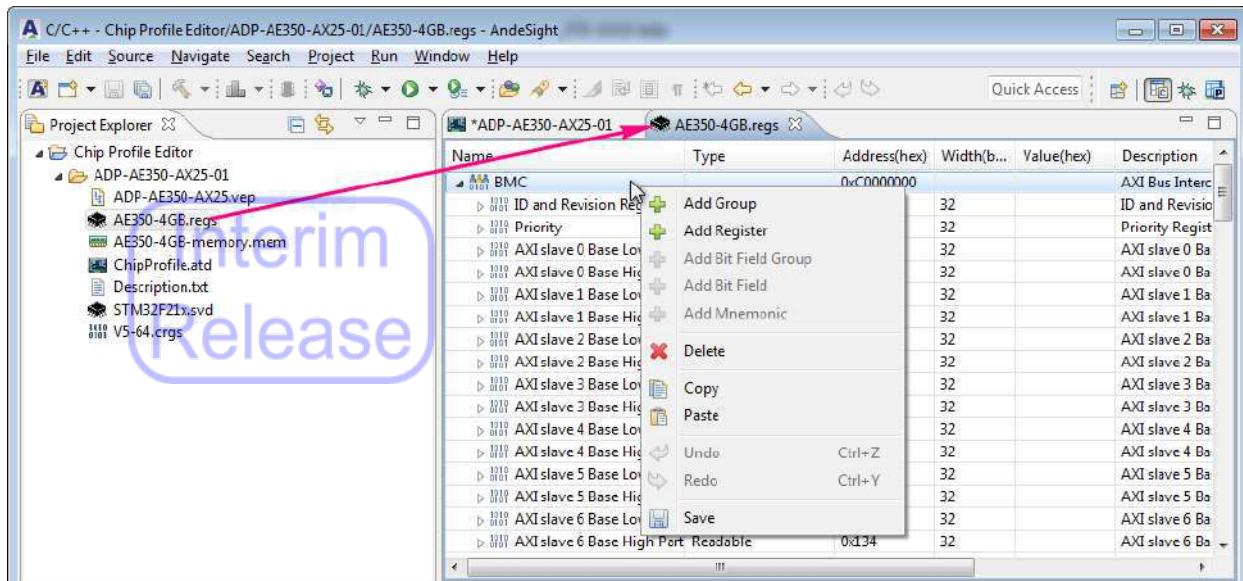


The access sizes are indicated by the values in the “Width” column:

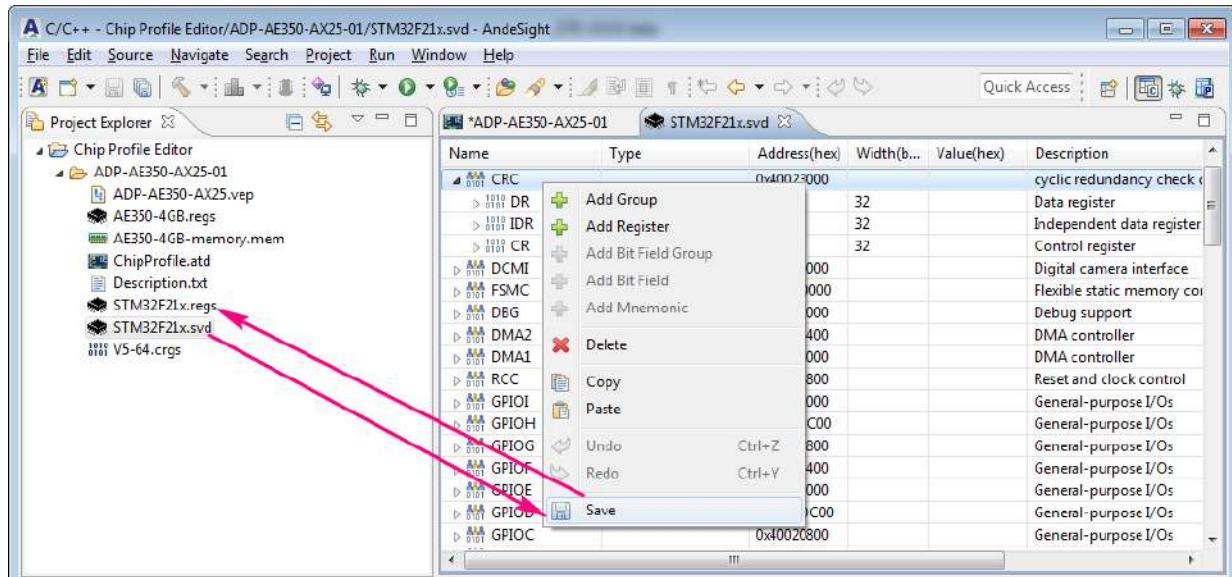
<u>Width value</u>	<u>Access size</u>
8	8-bit (one-byte) memory accesses
16	16-bit (two-byte, half-word) memory accesses
32	32-bit (four-byte, a word) memory accesses
64*	64-bit (eight-byte, a quad word) memory accesses

(\*The width value “64” also indicates 32-bit memory access for 32-bit cores.)

Likewise, information pertaining to peripheral registers and their bit fields can be modified for specific SoC using the **SoC Registers Editor**:

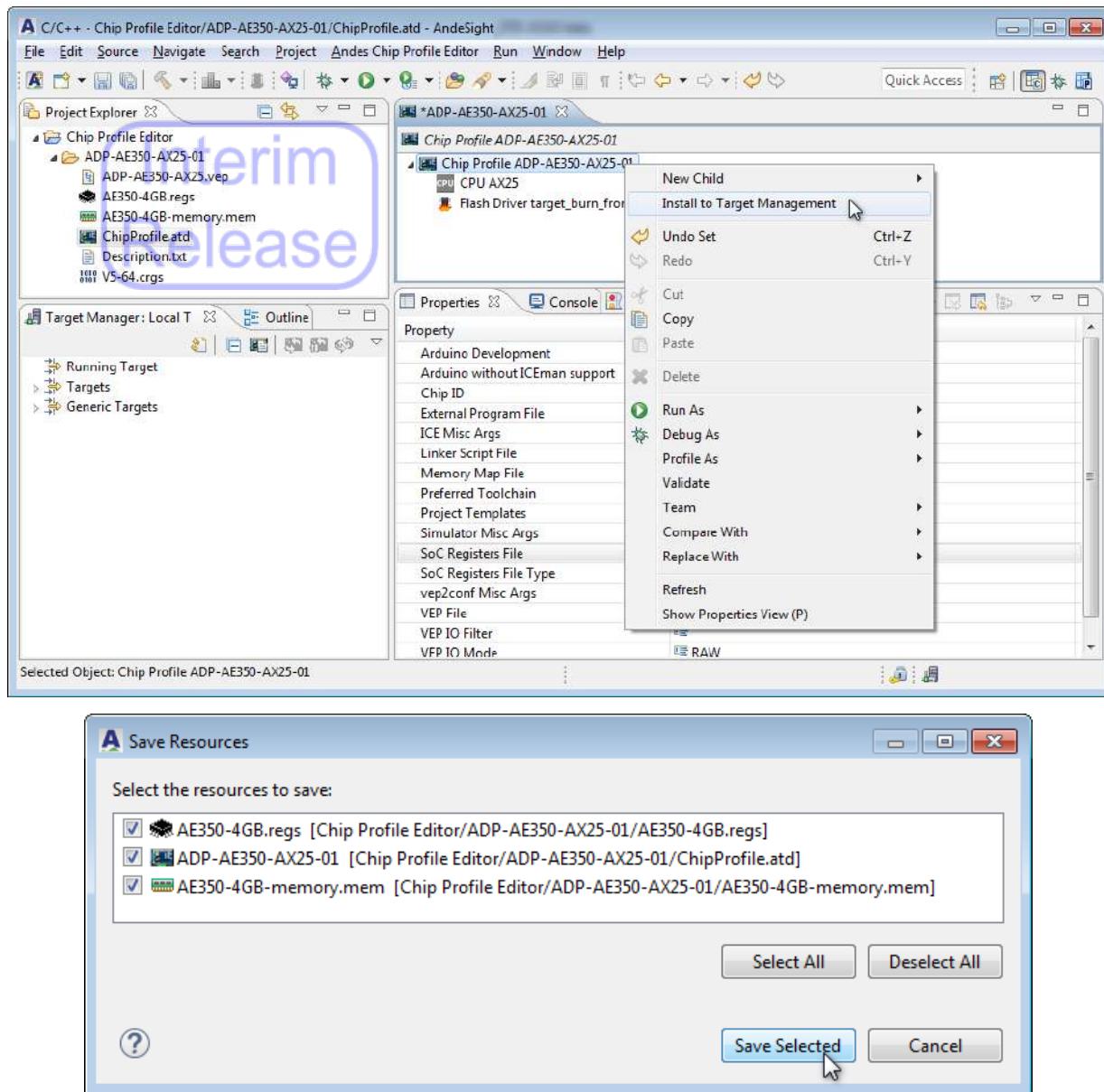


Note that SoC registers files in CMSIS-SVD format (.svd) are read-only in the AndeSight IDE and cannot be edited in the **Soc Registers Editor**. To modify system settings in a .svd file, you may save it as a .regs file like below and edit the converted file in the **Soc Registers Editor**.

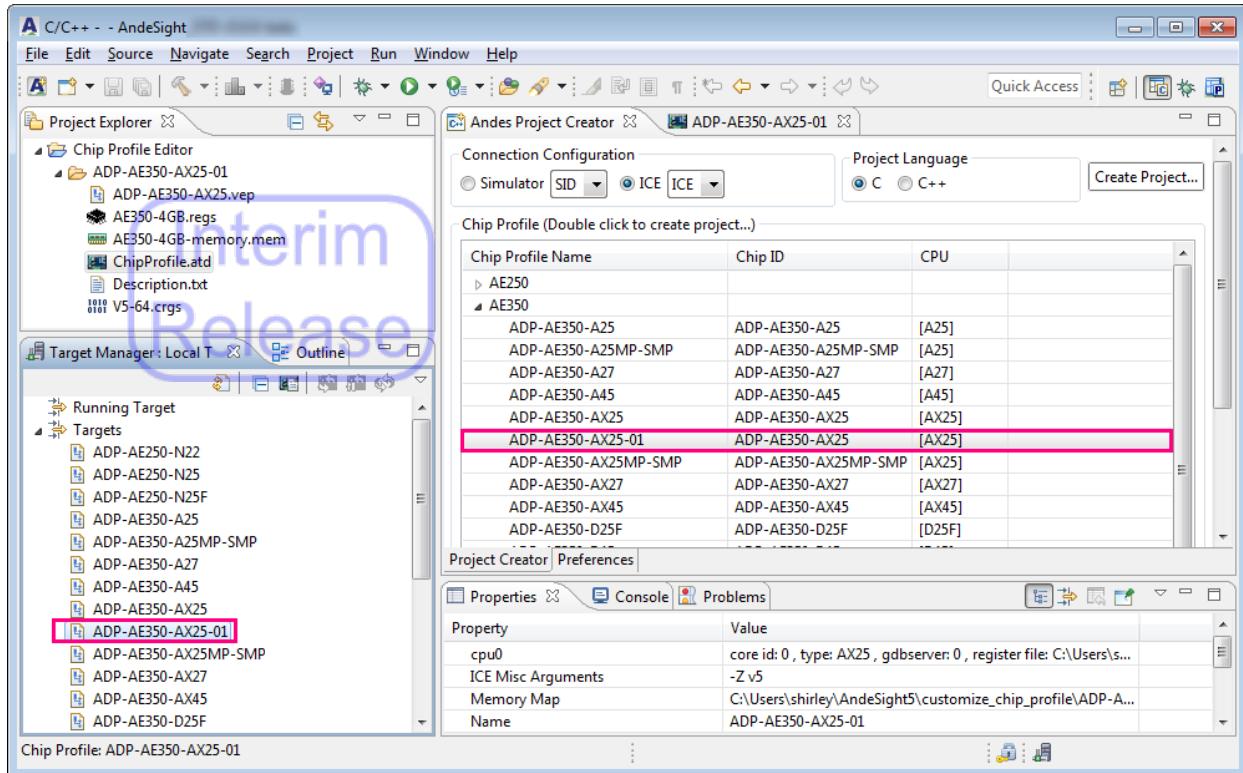


**Step 7** Upon the completion of configuration, right-click the target name in the invoked view and select “Install to Target Management” in the pull-down menu. At the Save Resources prompt, select all the change items made to the chip profile/associated files and click “Save

Selected” to save the changes.



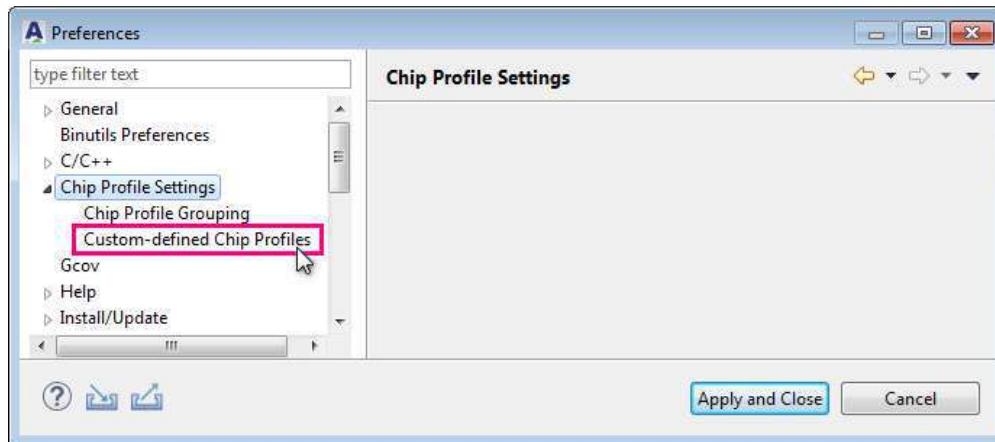
**Step 8** The target should now be integrated within the “Targets” group of the **Target Manager** view as well as the Chip Profile section of the **Andes Project Creator** view. At this point, your target has been successfully defined in AndeSight.



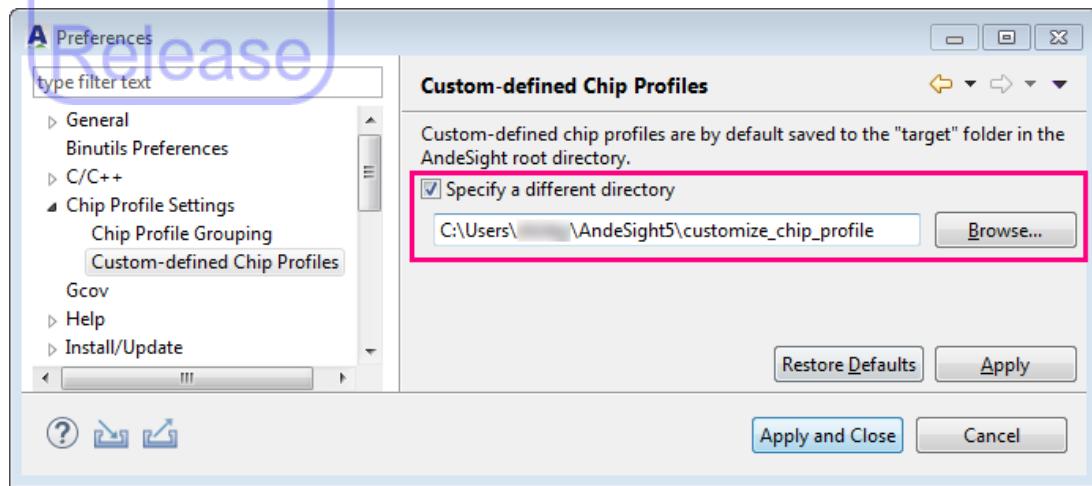
## NOTE

Targets that are defined through the chip profile editor are stored to **ANDESI GHT\_ROOT\target** by default. To change the default save location of custom-defined chip profiles, just proceed as follows:

**Step 1** On the AndeSight main menu, select “Window > Preferences” to invoke the **Preferences** dialog and click “Chip Profile Settings > Custom-defined Chip Profiles” in the navigation pane.

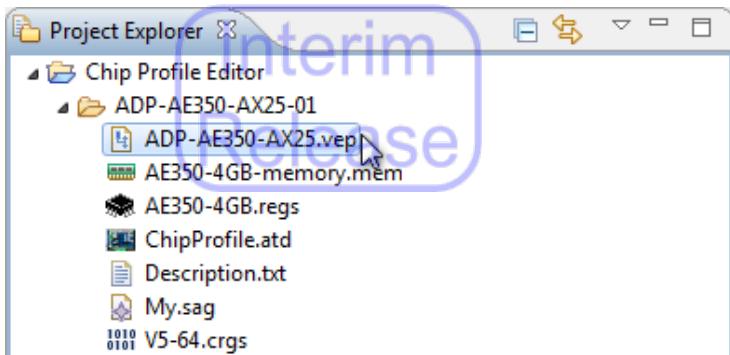


**Step 2** On the Custom-defined Chip Profiles page, select the option “Specify a different directory”. Accept the alternative save location that shows on this page (i.e., “**USER\_ROOT\AndeSight5\customize\_chip\_profile**”) or click “Browse...” to select another folder in your file system. Then, click on “Apply and Close” to complete the configuration.



## CPU Dialog

The **CPU Dialog** is a graphical interface by which to configure the CPU architecture. It can be invoked by double-clicking a vep file under a chip profile folder.



The configuration settings within the **CPU Dialog** vary according to the CPU specified.

The following walk you through each tab and illustrate respective options using the AX25 CPU Dialog as an example.

### ■ Cache tab

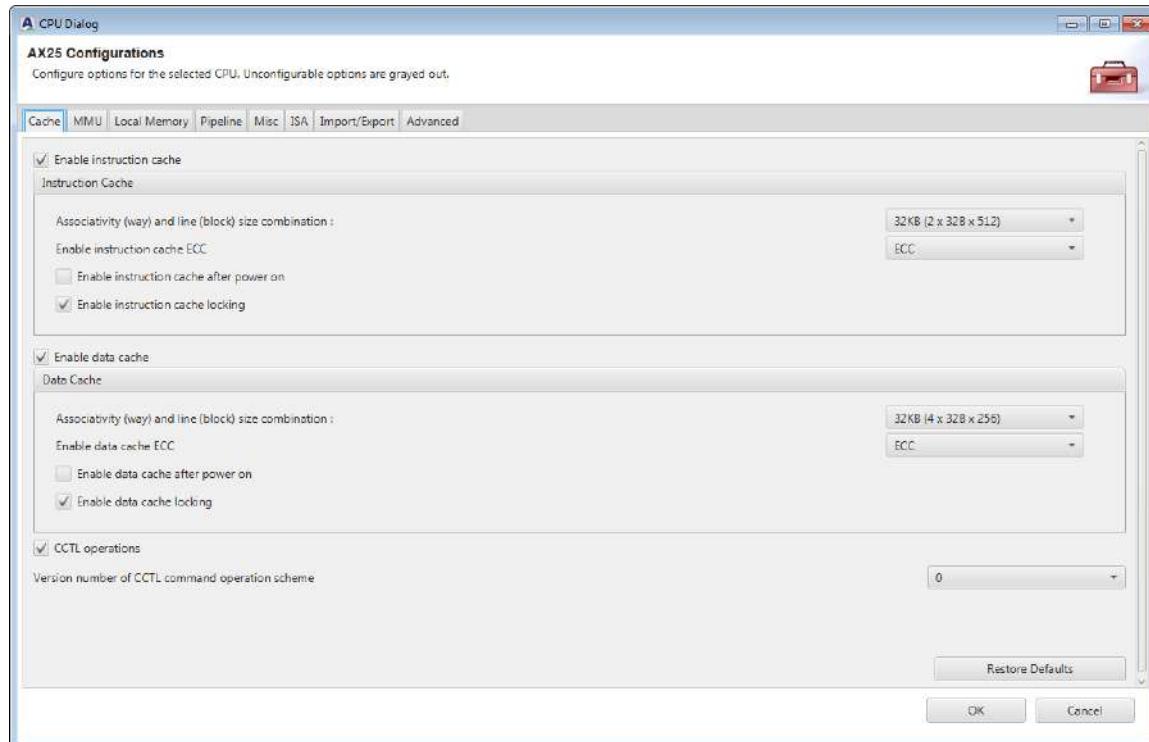


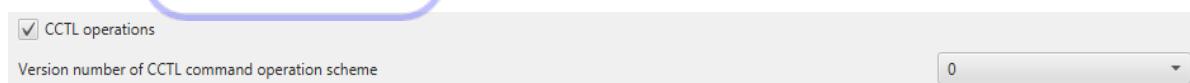
Figure 11. AX25 CPU Dialog > Cache

- **(Enable) instruction/data cache**

These two sections allow you to specify the cache size according to the N-way set-associativity cache scheme and the cache line size combination.

- **CCTL operations**

This section allows you to configure CCTL (cache control) operations and the version of CCTL command operation scheme.



## ■ Memory Management Unit (MMU)

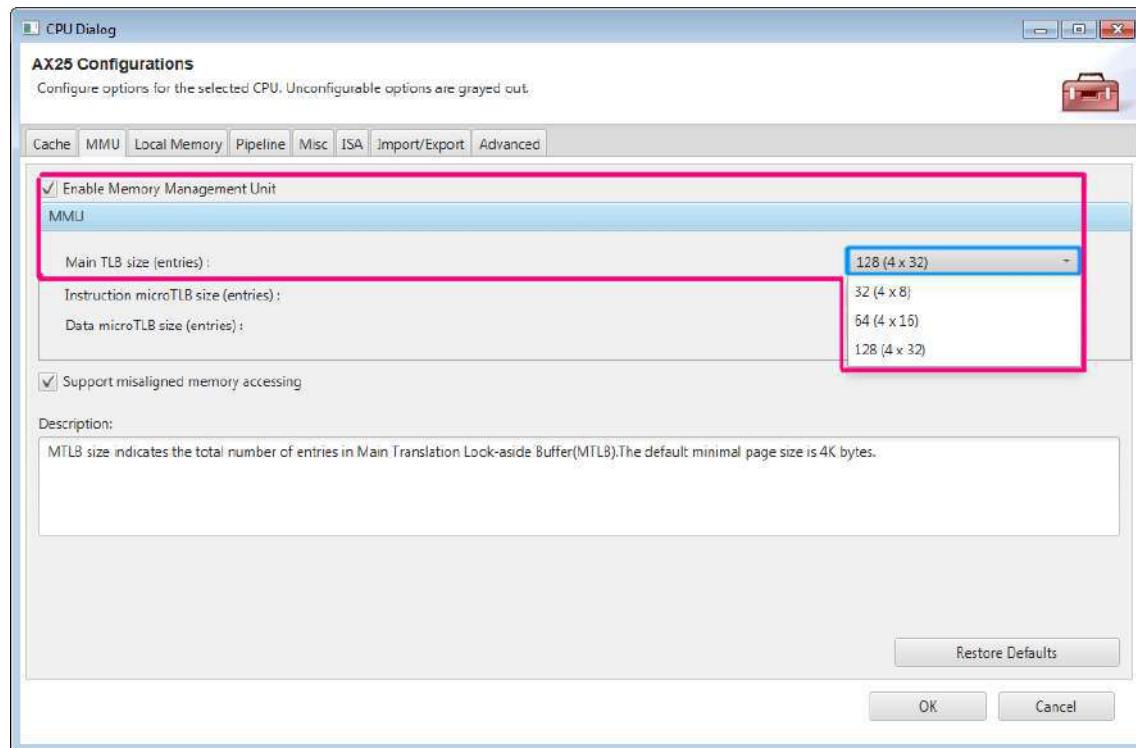


Figure 12. AX25 CPU Dialog > Memory Management Unit

To enable the memory management feature for a V5 core, simply select the option “Enable Memory Management Unit” and set the Main TLB size to 32, 64, or 128 entries. You can also configure whether a V5 CPU supports misaligned memory accesses in this tab.

## ■ Local Memory

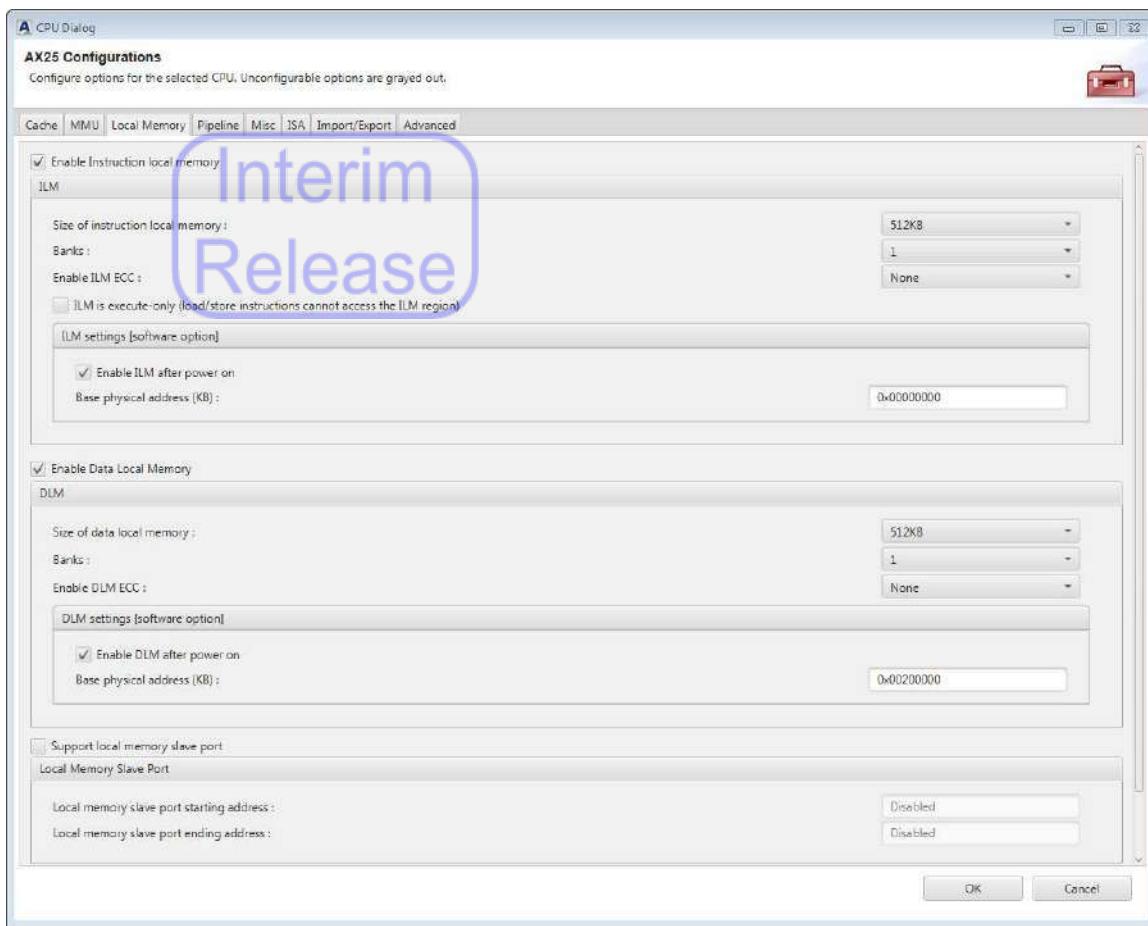


Figure 13. AX25 CPU Dialog > Local Memory

- **Enable instruction/data local memory**

These two sections allow you to specify the sizes of the Instruction Local Memory (ILM) and Data Local Memory (DLM). The base physical address is valid over a range of 0 ~ 4095 and it must be 1 MB or 1 KB aligned based on the ICM\_CFG.BSAV setting. You can also specify the error-checking scheme (i.e. none, parity, or ECC) for the ILM/DLM.

- **Support local memory slave port**

This section allows you to enable the local memory slave port and configure its address range.

## ■ Pipeline tab

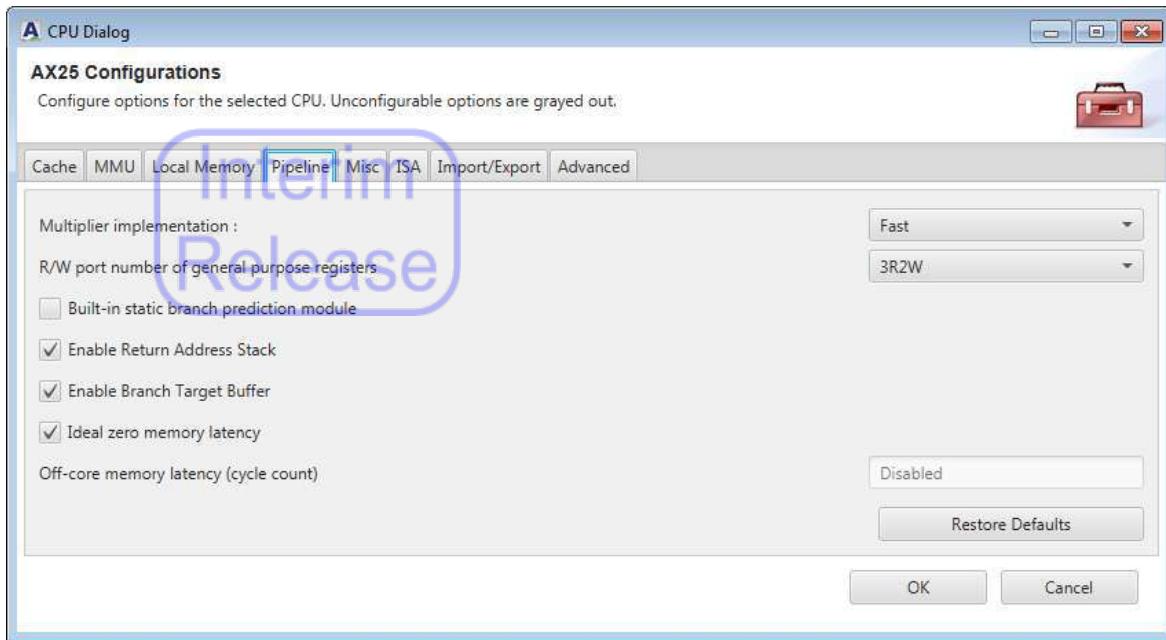


Figure 14. AX25 CPU Dialog > Pipeline

The following pipeline settings allow you to optimize processor performance:

- **Multiplier implementation**

The multiplier types include fast, radix2, radix4, radix16, radix256 and slow for V5 cores.

- **R/W port number of general purpose registers**

V5 cores can be programmed to have three read registers and two registers (3R2W) or three read registers and one register (3R1W).

- **Built-in static branch prediction module**

This option allows you to determine whether the built-in static branch prediction module is present.

- **Enable Return Address Stack**

This option allows you to determine whether the Return Address Stack feature is present.

- **Enable Branch Target Buffer**

This option allows you to determine whether the Branch Target Buffer feature is present.

- **Ideal zero memory latency**

This configuration sets memory delay to zero and ignores all the latency resulted from cache miss or cache-off load/store.

- **Off-core memory latency**

This option is disabled when the option “Ideal zero memory latency” is selected. When the option “Ideal zero memory latency” is deselected, this option is configurable and must be set to a value greater than 0.



## ■ Miscellaneous

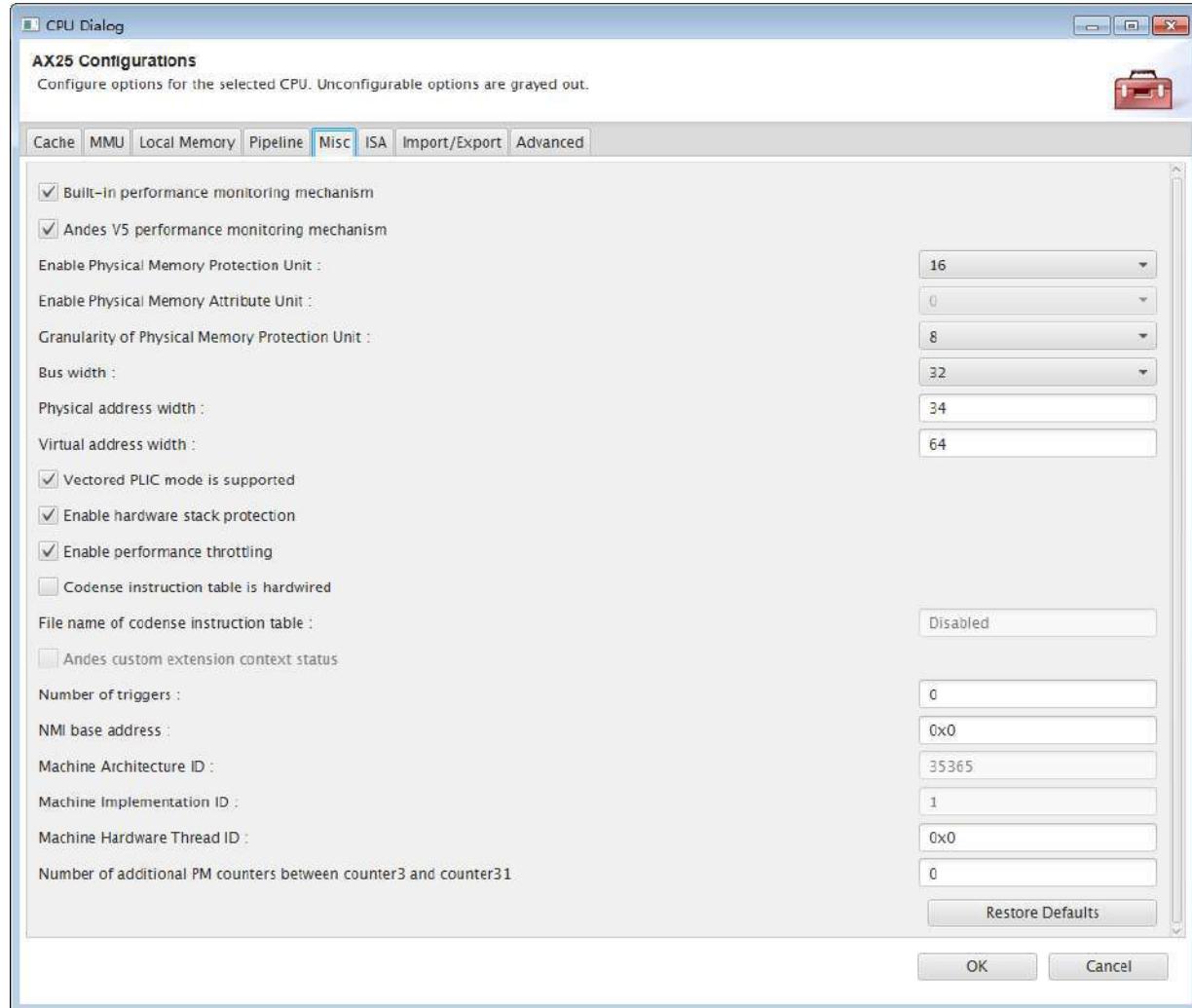


Figure 15. AX25 CPU Dialog > Miscellaneous

- **Built-in performance monitoring mechanism**

This option allows you to determine whether the built-in performance monitoring feature is present.

- **Andes V5 performance monitoring mechanism**

This option allows you to determine whether Andes-enhanced performance monitoring feature is present.

- **Enable Physical Memory Protection Unit**

This option allows you to specify the number of supported entries in the physical memory protection unit.

- **Enable Physical Memory Attribute Unit**

This option allows you to specify the number of supported entries in the physical memory attribute unit.

- **Condense instruction table is hardwired**

This option allows you to determine whether the Xcodense instruction table is in memory or hardwired. If the instruction table is hardwired, software does not have to set up UITB.ADDR to use the Xcodense instruction.

- **Andes custom extension context status**

This option allows you to determine whether Andes custom extension context status is configurable.

- **Number of additional PM counters between counter 3 and counter31**

Refer to the table below and enter a value to specify the number of additional performance monitoring counter from counter3 to counter 31.

Value	Meaning
0	4 additional counters
1	3 additional counters
2	2 additional counters
3	1 additional counters
4	0 additional counters
30 > N > 4	N additional counters

## ■ Instruction Set Architecture

The built-in instructions, extensions and ISA options configurable for an AndesCore CPU are listed under this tab. For details of the options, please refer to the following two documents: *The RISC-V Instruction Set Manual Volume I: User-Level ISA* and *AndeStar V5 Instruction Extension Specifications*.

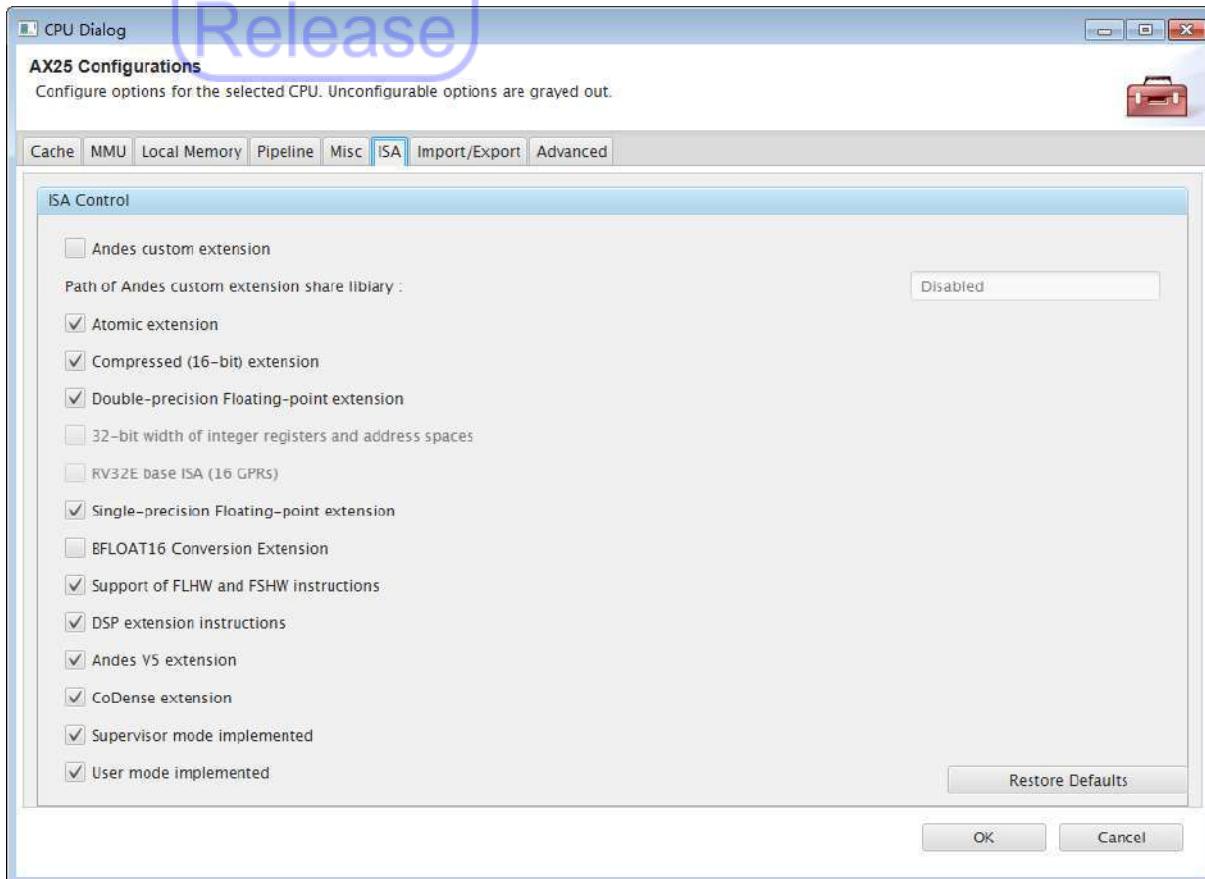


Figure 16. AX25 CPU Dialog > ISA

## ■ Import/Export

You may import an .opt CPU configuration file or export the current CPU settings to an .opt CPU configuration file.

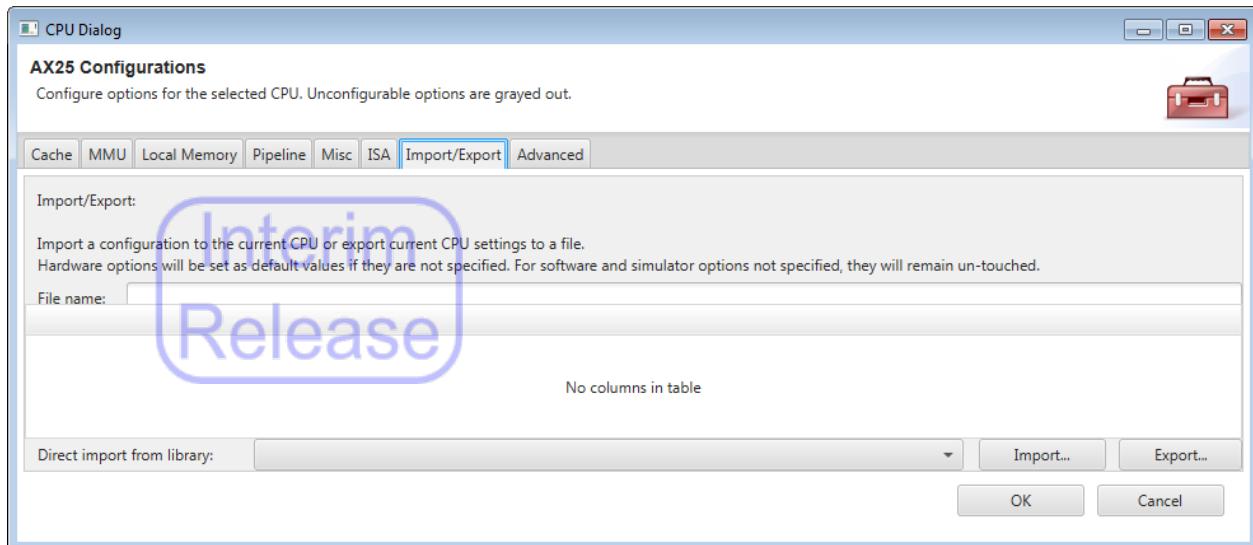


Figure 17. AX25 CPU Dialog > Import/Export

## Importing a .opt file

To import an .opt file, click “Import” to evoke a dialog box to browse for the path of the CPU configuration file. The import function incorporates the following error-checking mechanisms to ensure consistency in the configuration of the CPU:

1. Unless hardware dependent CPU configuration options are specified in the imported .opt file, they are assigned the default values.
2. Unless simulator dependent CPU configuration options are specified in the imported .opt file, the corresponding settings in the current .vep file are applied.
3. The import would be aborted under the following conditions:
  - An option that accepts only a hexadecimal value is entered with a decimal value.
  - The specified option values do not fall within the valid range.

## Exporting an .opt file

To export a CPU configuration to an .opt file, click “Export” and specify the destination and name of the exported file in the evoked dialog.

### ■ Advanced

Select the option “Allow advanced or extended options” and enter the desired CPU options in the editor below. The options entered in this editor will override the specifications in the **CPU Dialog**.

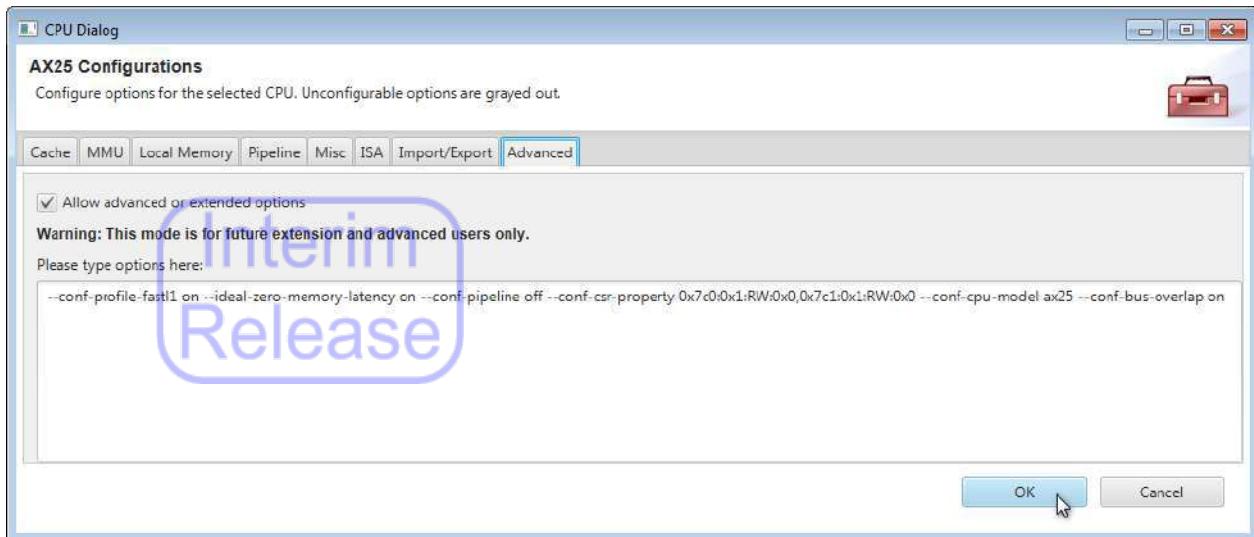
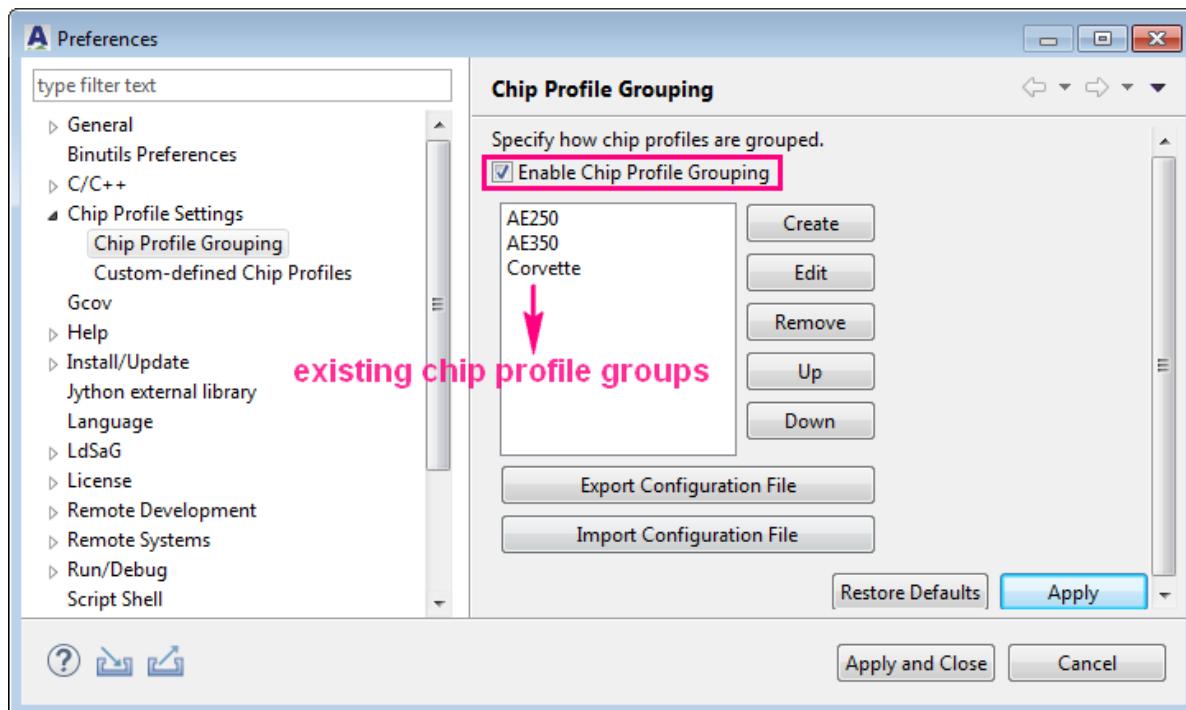


Figure 18. AX25 CPU Dialog > Advanced

### 2.2.1.3 Grouping defined targets

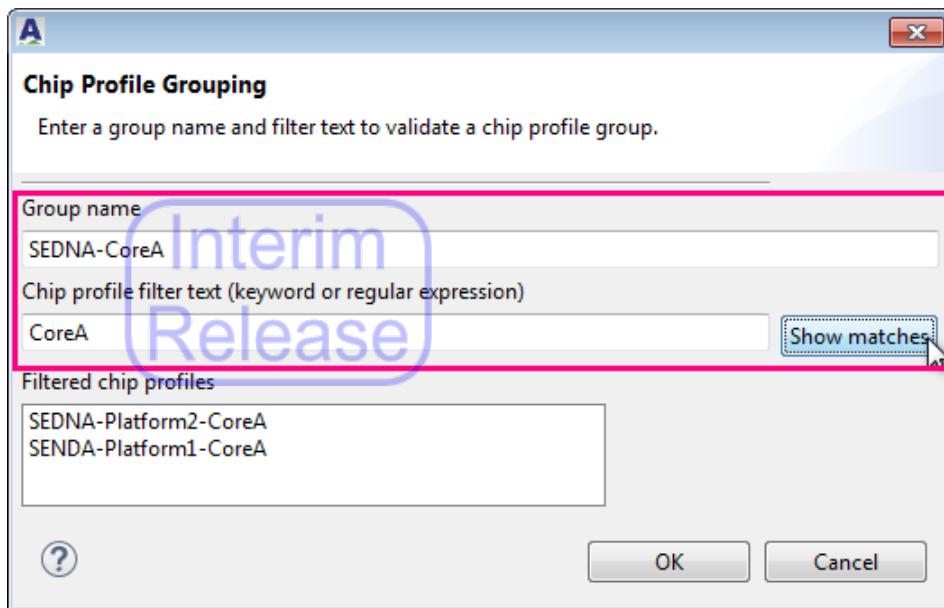
Defined targets can be grouped and sorted to make them easier to locate during project creation or target management. The grouping panel can be found on the **Chip Profile Grouping** page. On the AndeSight main menu, select “Window > Preferences” and click “Chip Profile Settings > Chip Profile Grouping” in the navigation pane of the **Preferences** dialog. Select the option “Enable Chip Profile Grouping” to turn on the grouping function. The existing chip profile groups will be listed below.



Use the buttons on this page to manage the grouping and then click “Apply and Close” to complete configuration:

- Creating a group:

Click “Create” to invoke the **Chip Profile Grouping** dialog. Specify a group name and a filter text to sort chip profiles within the group. Then, click “Show matches” to check the sorting results, and click “OK”.



- Modifying a group configuration:

Select a chip profile group and click “Edit”. Modifications are made on the **Chip Profile Grouping** dialog.

- Deleting a group:

Select a chip profile group and click “Remove”.

- Reordering a grouping:

Select a chip profile group and click “Up” or “Down” to move it to the desired position in the group list.

- Exporting/Importing a grouping configuration:

Click “Export Configuration File” to export the chip profile grouping settings to a file or “Import Configuration File” to import a configuration file to define the grouping. The default export file is “group\_setting.prefs”.

Chip profile grouping is reflected in **Andes Project Creator** as well as in **Target Management Default Settings** (Section 2.2.4). Targets are grouped according to the grouping configuration in the **Chip Profile** section. Some targets are not sorted into any defined groups in the configuration; therefore, they are identified as “other chip profiles” and listed under the tag “Other Chip Profile.”

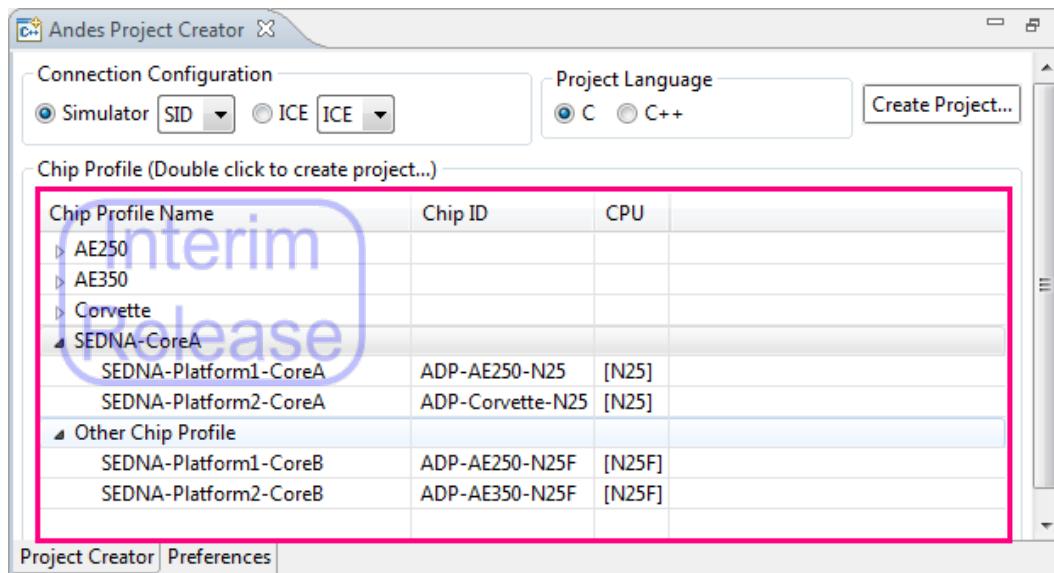


Figure 19. Chip profile grouping in Andes Project Creator

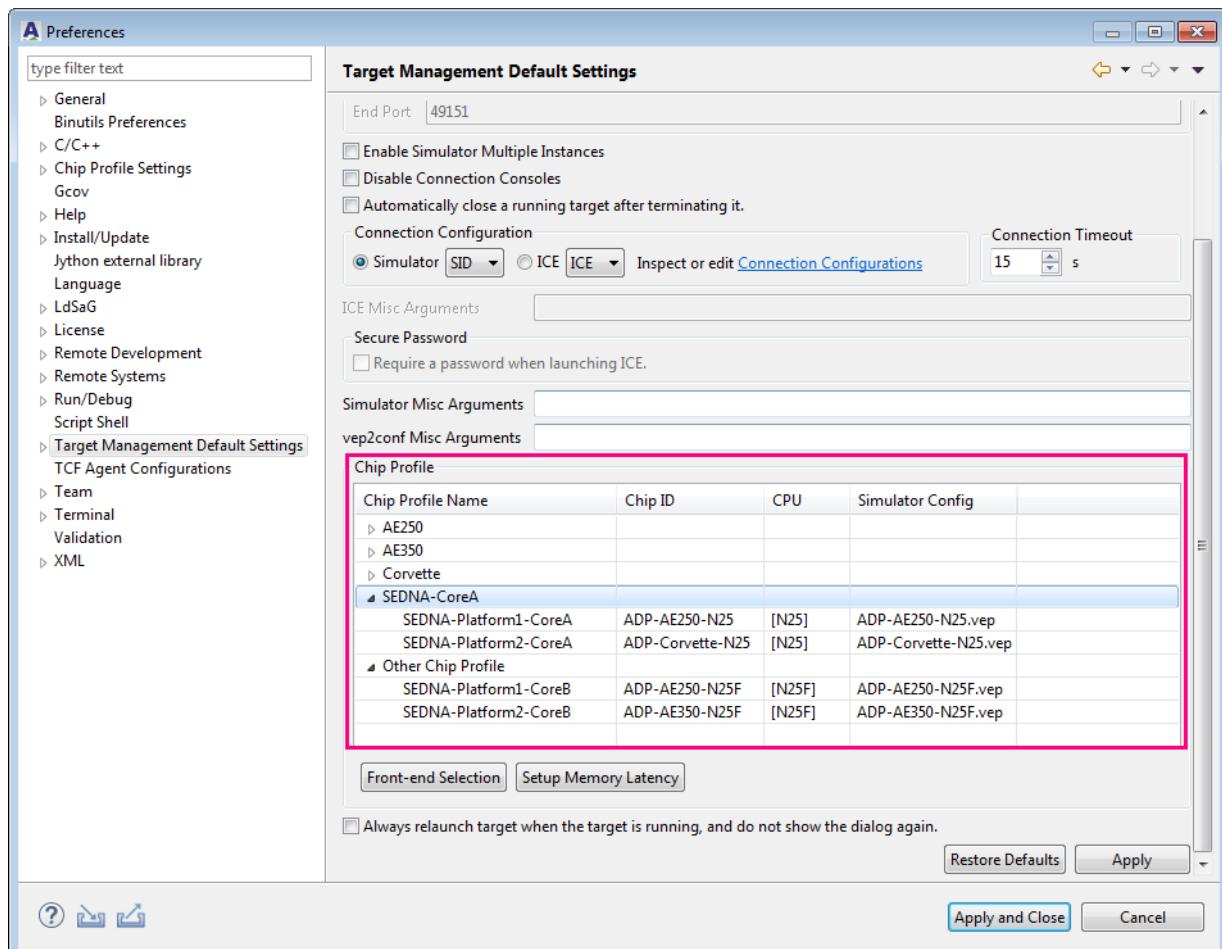
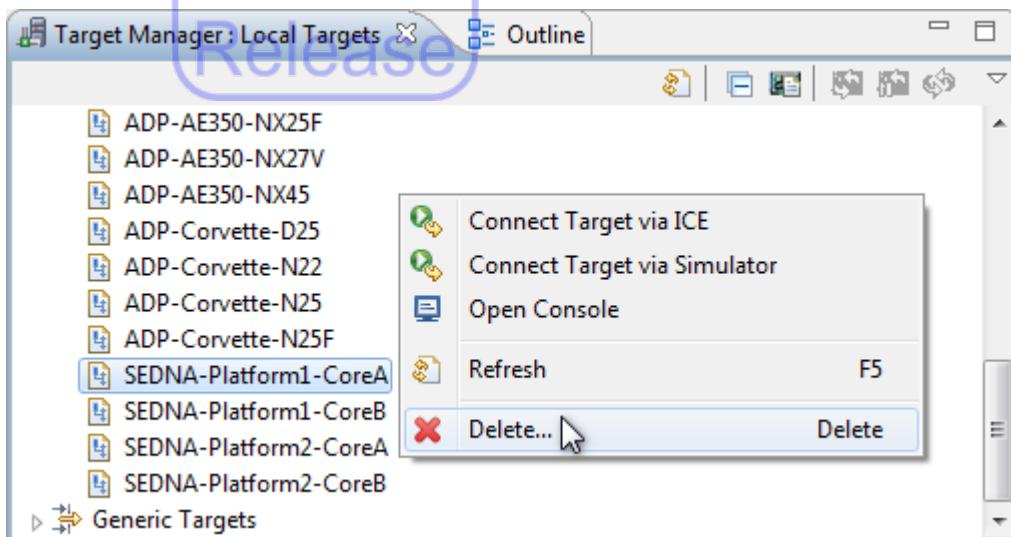


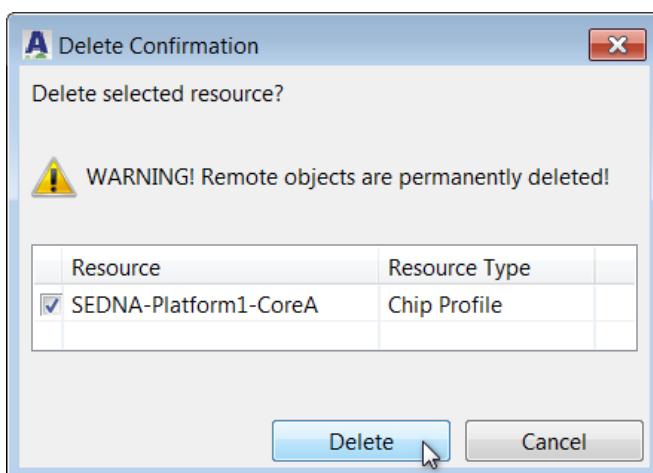
Figure 20. Chip profile grouping in Target Management Default Settings

#### 2.2.1.4 Deleting a target defined by a chip profile

The pre-defined targets that come with the AndeSight installation are not deletable; however, you can delete the targets that you define. This can be achieved by locating the desired target in the **Target Manager** view, right-clicking, and then selecting “Delete ...” from the pull-down menu.

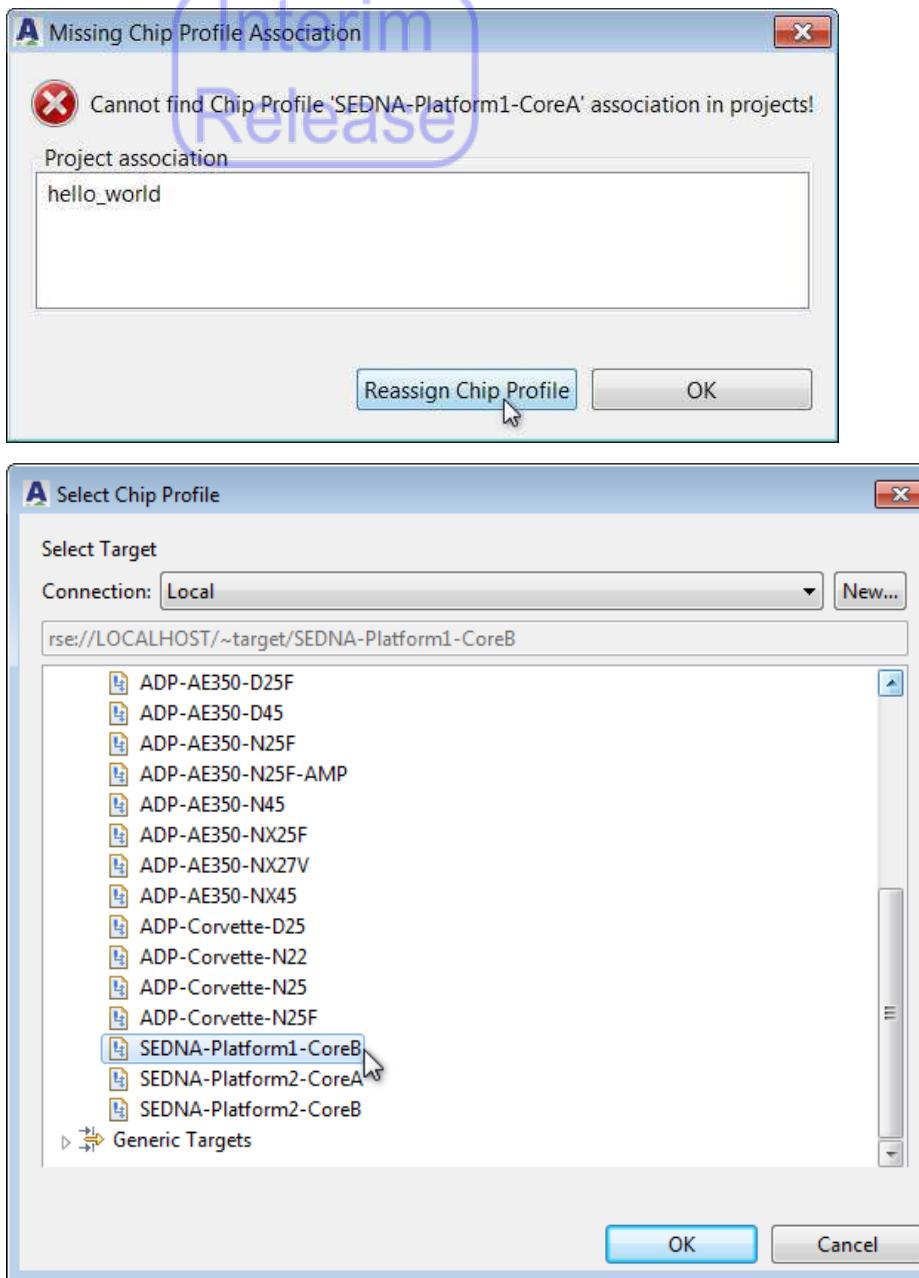


Click “Delete” on the dialog that appears to confirm the deletion, and continue.



The target is immediately removed from the **Target Manager** view and **Andes Project Creator** view. Its chip profile and associated software settings files are also deleted under **ANDES\GHT\_ROOT\target**.

If the deleted chip profile had been assigned to a project, then the **Missing Chip Profile Association** dialog pops up to inform you of the loss of chip profile association. Simply click “Reassign Chip Profile” in the dialog and assign an available target to the project in the **Select Chip Profile** wizard.



### 2.2.2. Setting up the target system

The target system of an AndeSight project can be a simulator target or an ICE target. In other words, it may be a virtual evaluation platform (VEP) connected to a debugging host through an SoC simulator specified by a VEP configuration file. It may also be a real evaluation platform (REP) connected to a debugging host through an ICE debugger. In addition to the above two target systems, AndeSight STD IDE allows connection to a target system of Linux applications via a network. Before setting up the target systems, follow the illustration below to build the physical network between AndeSight IDE and the target systems (simulator target/ICE target/Linux application).

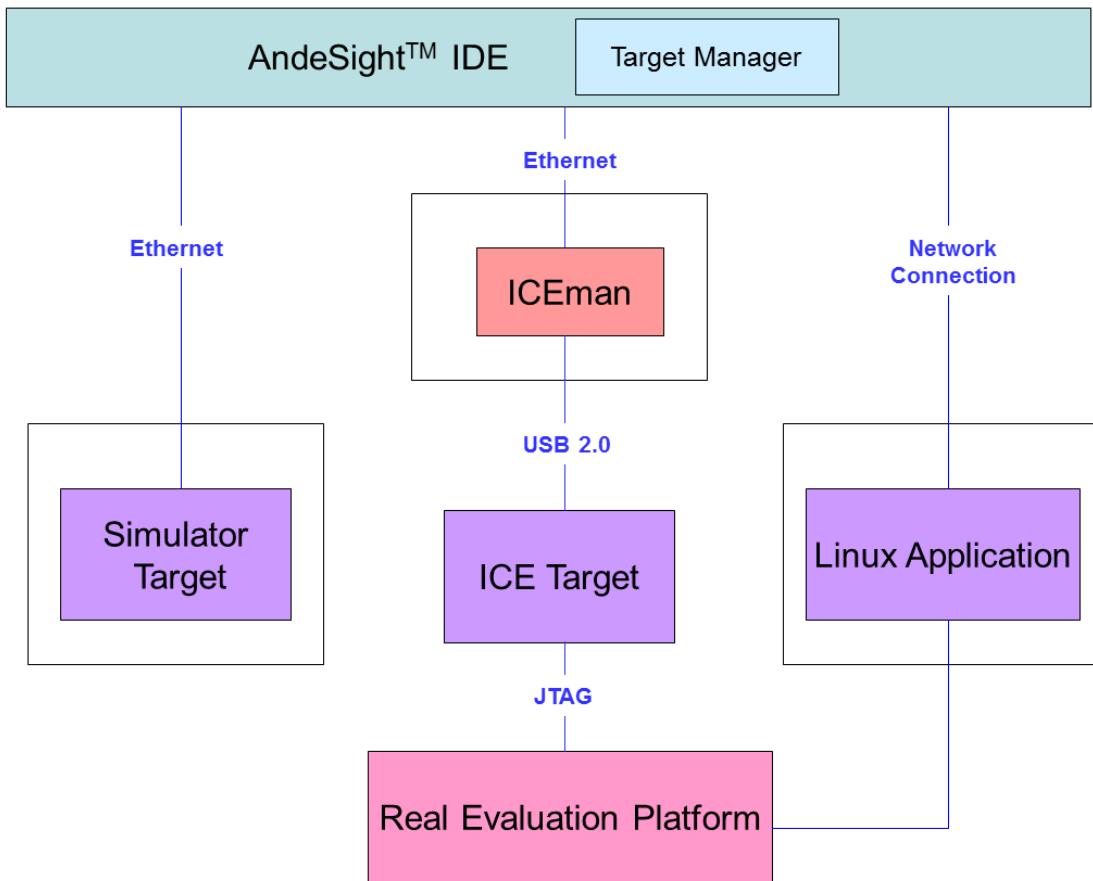


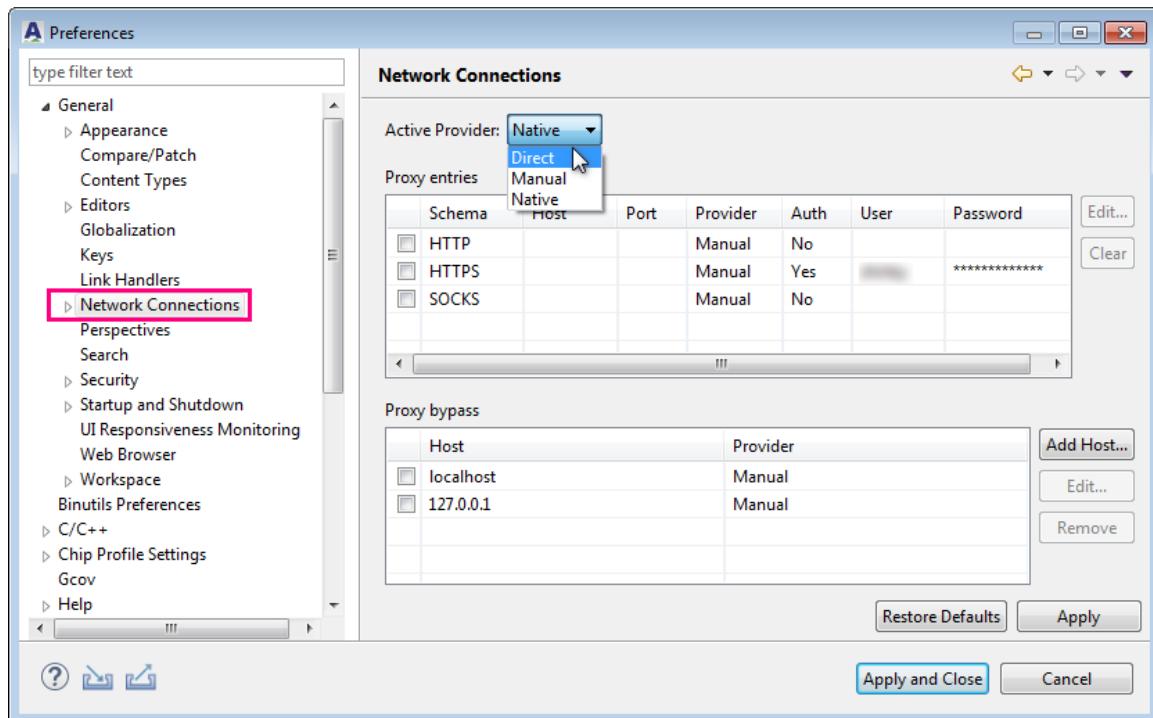
Figure 21. Physical network between AndeSight STD IDE and target systems

### 2.2.2.1 Setting up a simulator target

Simulator targets do not have to be set up manually. However, for Ubuntu 18.04 with proxy settings, it is necessary to modify the network connection settings in AndeSight to ensure the successful launch of simulator targets. This is done as follows:

**Step 1** On the AndeSight main menu, select “Window > Preferences” and click “General > Network Connections” in the navigation pane of the **Preferences** dialog.

**Step 2** Set the Active Provider option to “Direct”. Click “Apply and Close.”



### 2.2.2.2 Setting up an ICE target

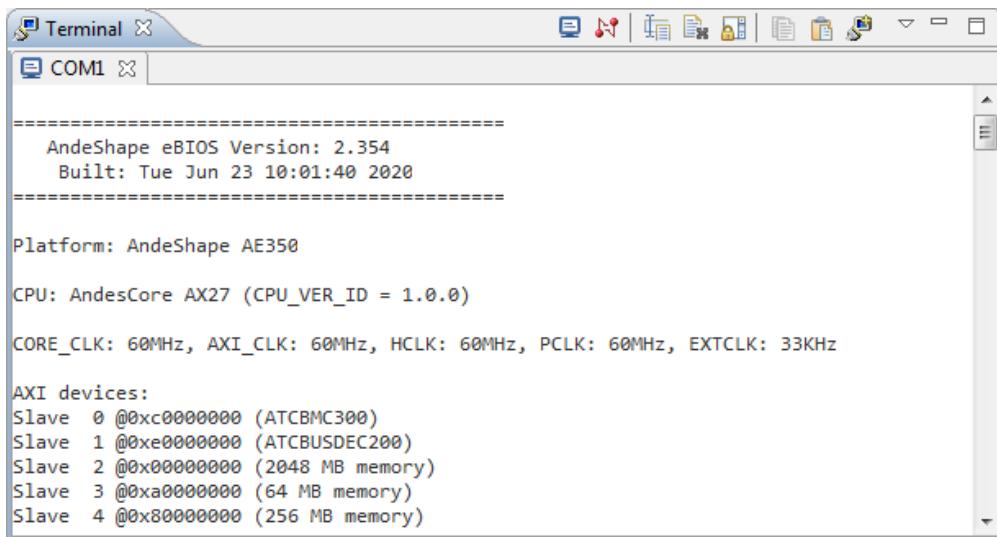
**Step 1** Install the target board (Network Cable, Ethernet, Power cord) and connect it to an ICE device.

**Step 2** Toggle the power switch of the target board to power on the Real Evaluation Platform (REP) and check that the on-board LED is illuminated.

**Step 3** Press the “PWR” button to start the target system.

### 2.2.2.3 Setting up a Linux application

**Step 1** Refer to Section 2.2.3 to build a terminal connection with a Linux application.



The screenshot shows a terminal window titled "Terminal" with a tab labeled "COM1". The window displays the following boot log:

```

=====
AndeShape eBIOS Version: 2.354
Built: Tue Jun 23 10:01:40 2020
=====

Platform: AndeShape AE350

CPU: AndesCore AX27 (CPU_VER_ID = 1.0.0)

CORE_CLK: 60MHz, AXI_CLK: 60MHz, HCLK: 60MHz, PCLK: 60MHz, EXTCLK: 33KHz

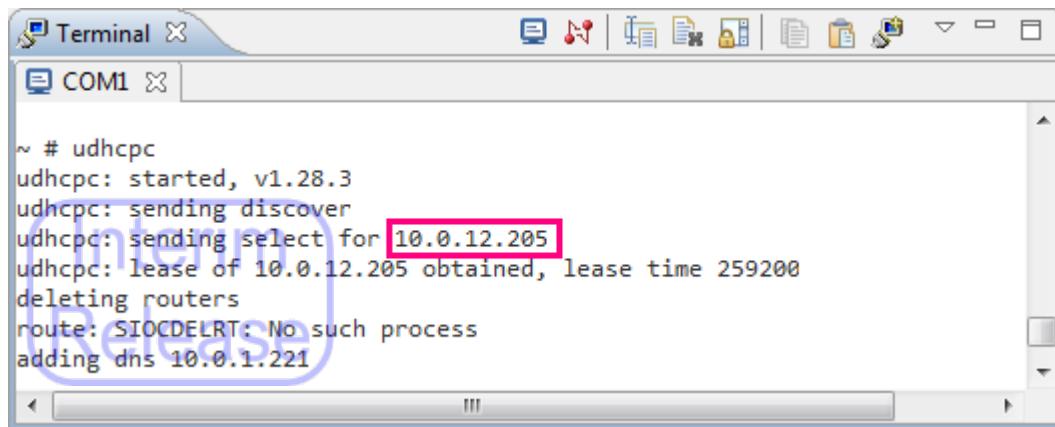
AXI devices:
Slave 0 @0xc0000000 (ATCBMC300)
Slave 1 @0xe0000000 (ATCBUSDEC200)
Slave 2 @0x00000000 (2048 MB memory)
Slave 3 @0xa0000000 (64 MB memory)
Slave 4 @0x80000000 (256 MB memory)

```

**Step 2** Consult *Linux Development User Manual* to boot up the Linux kernel via GDB for a V5 target.

**Step 3** Obtain an IP address from a DHCP server. In the following instance, 10.0.12.205 was selected.

```
# udhcpc
```



The screenshot shows a terminal window titled "Terminal" with a tab labeled "COM1". The terminal output displays the following log from the "udhcpc" command:

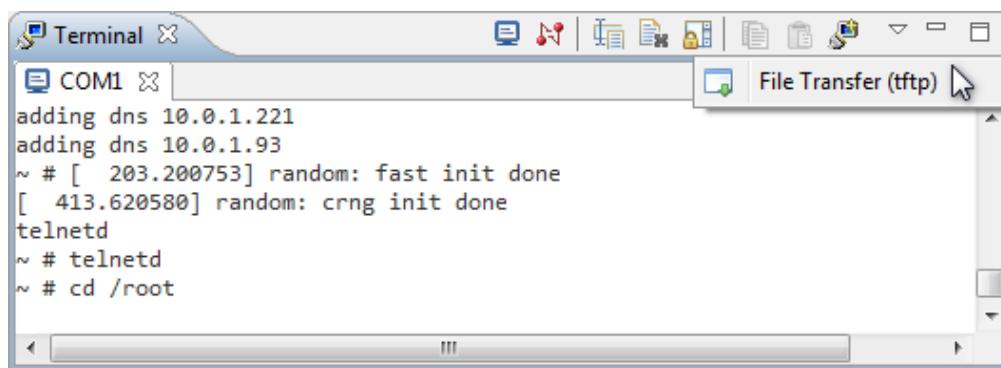
```
~ # udhcpc
udhcpc: started, v1.28.3
udhcpc: sending discover
udhcpc: sending select for 10.0.12.205
udhcpc: lease of 10.0.12.205 obtained, lease time 259200
deleting routers
route: SIOCDELRT: No such process
adding dns 10.0.1.221
```

**Step 4** Initiate the telnet service.

```
# telnetd
```

**Step 5** Consult the *Linux Development User Manual* to build a gdbserver in the Linux environment. The source files of gdbserver (**gdb.tgz**) are placed under **ANDESIGHT\_ROOT\Linux\** in AndeSight on Linux environment.

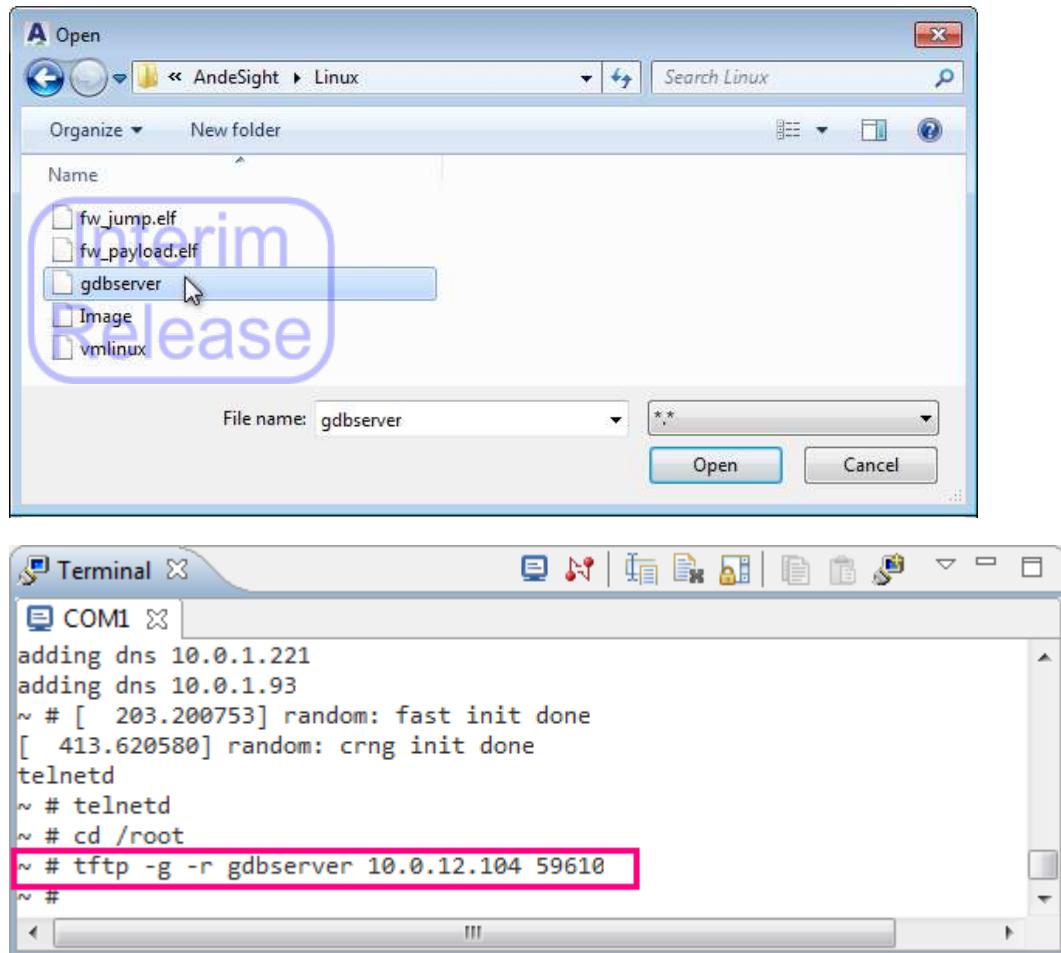
**Step 6** Return to the **Terminal** view in AndeSight. Change the current directory to **/root**, click  and select “File Transfer (tftp)”. Specify gdbserver in the invoked dialog and click “Open” to transfer it to the Linux kernel.



The screenshot shows a terminal window titled "Terminal" with a tab labeled "COM1". The terminal output shows the user navigating to the root directory and starting the telnetd service:

```
adding dns 10.0.1.221
adding dns 10.0.1.93
~ # [ 203.200753] random: fast init done
[ 413.620580] random: crng init done
telnetd
~ # telnetd
~ # cd /root
```

A file transfer dialog is open in the background, with the "File Transfer (tftp)" option selected.

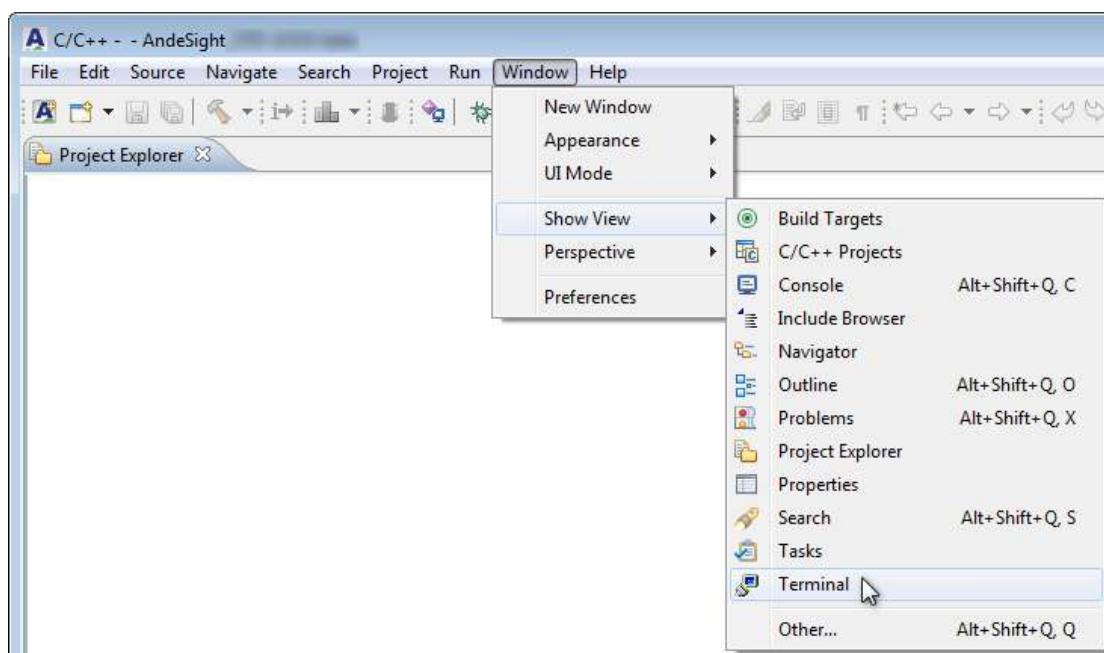


### 2.2.3. Terminal connection to ICE targets or Linux applications

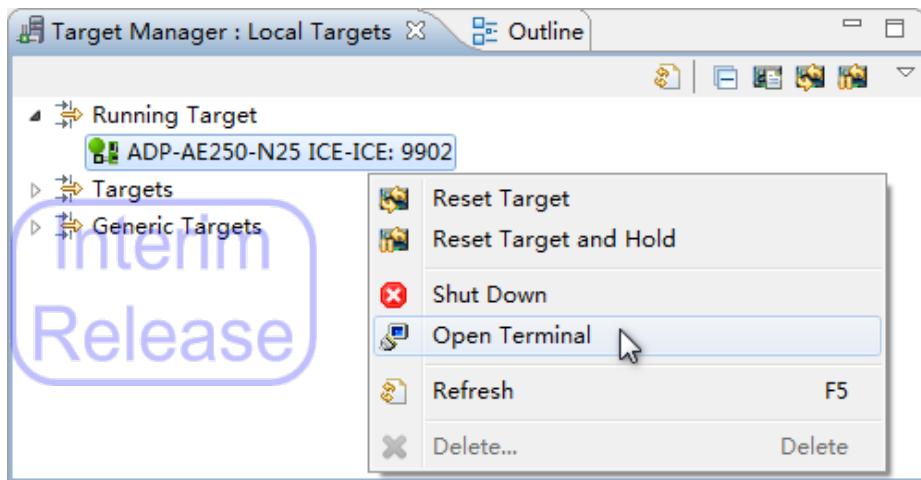
From the **Terminal** view in AndeSight, it is possible to build a connection to ICE targets or Linux applications and issue shell commands directly to targets. To establish a terminal connection, proceed as follows:

**Step 1** Install the target board (Network Cable, Ethernet, Power cord) and connect it to an ICE device. If you are using an Andes ICE (AICE), you may refer to the *AICE Quick Start Guide* for information on the installation procedure.

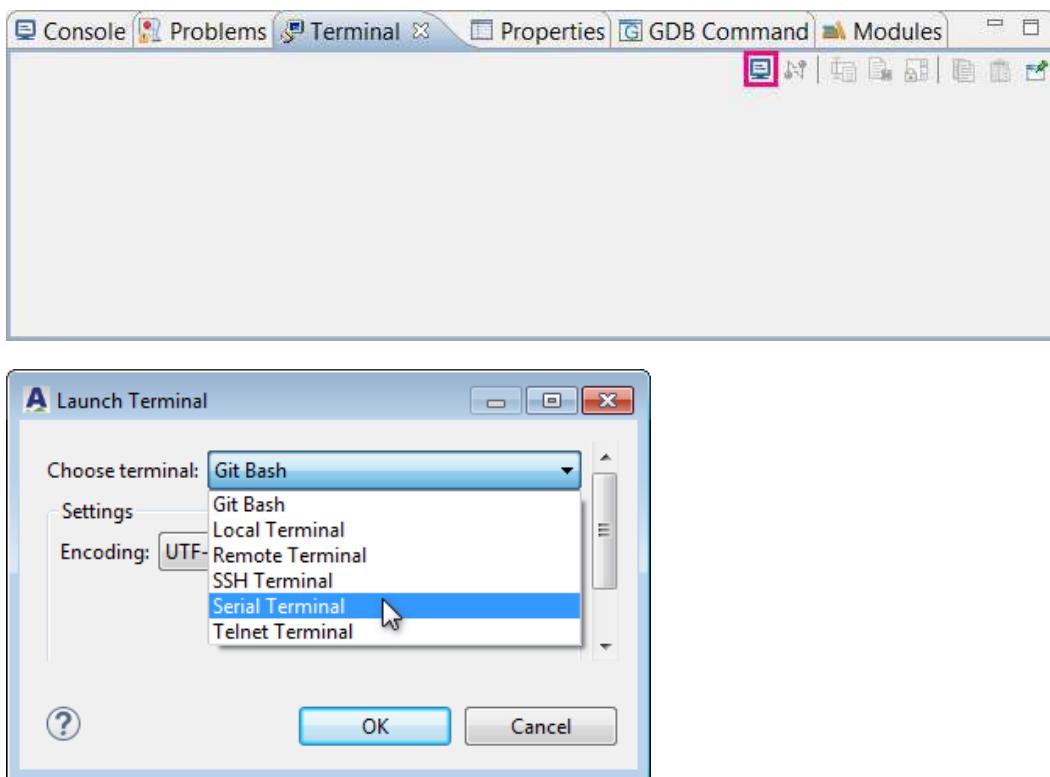
**Step 2** On the AndeSight main menu, select “Window > Show View > Terminal” to open a terminal.



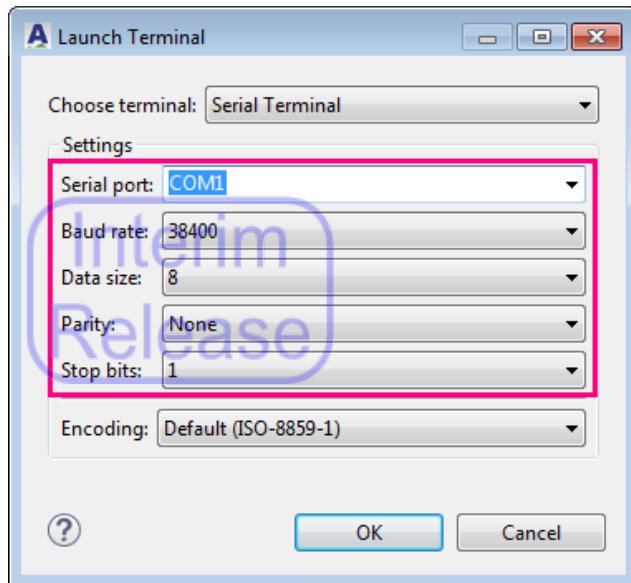
If your ICE target has been launched, you may also invoke the **Terminal** view from the **Target Manager** view (See Section 2.2.5). Simply right-click the ICE target in the Running Target group of the **Target Manager** view and select “Open Terminal” from the pull-down menu.



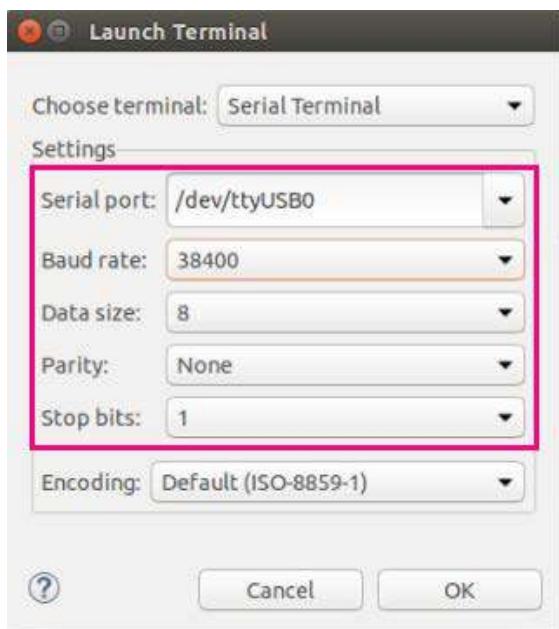
**Step 3** Find the **Terminal** view in AndeSight. Click  (open a terminal) to invoke the **Launch Terminal** prompt and select “Serial Terminal.”



Then, specify the COM port or serial port to enable a host-target connection and configure the port settings as follows:

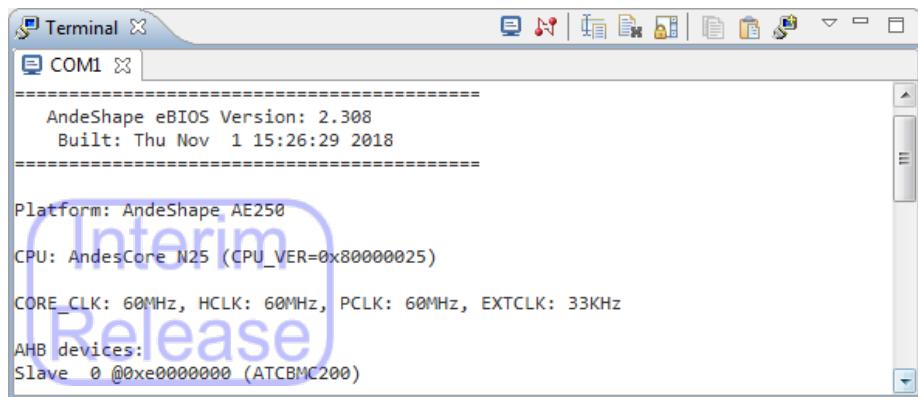


(Specifying the COM port and configuring port settings in Windows)



(Specifying the serial port and configuring port settings in Linux)

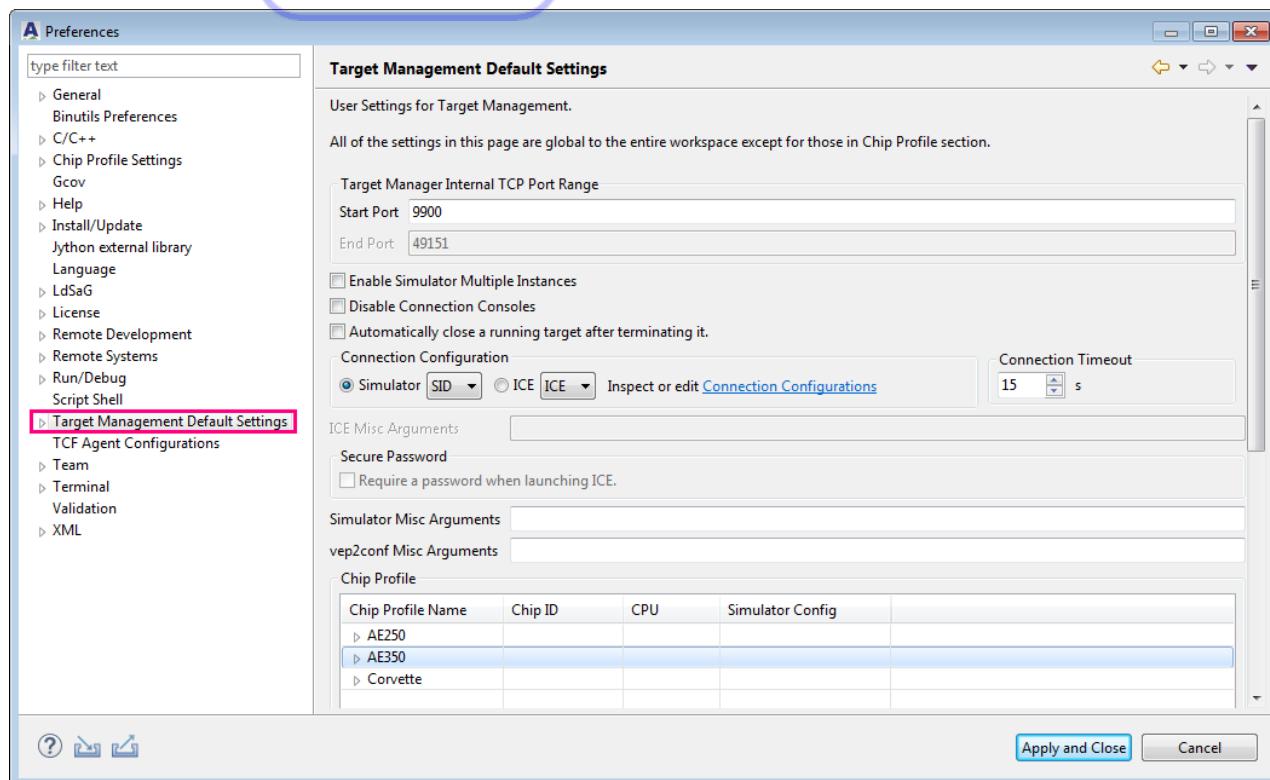
**Step 4** Toggle on the power switch of the target board. The boot message will appear in the **Terminal** view.



At this point, the terminal connection is established.

## 2.2.4. Target management default settings

The **Target Management Default Settings** page makes it possible to manage the default environment settings for target platforms. On the AndeSight main menu, select “Window > Preferences” to invoke the **Preferences** dialog. Then, click “Target Management Default Settings” in the navigation pane to enter the page and configure the settings. The settings on this page apply to the entire workspace with the exception of the Chip Profile section.

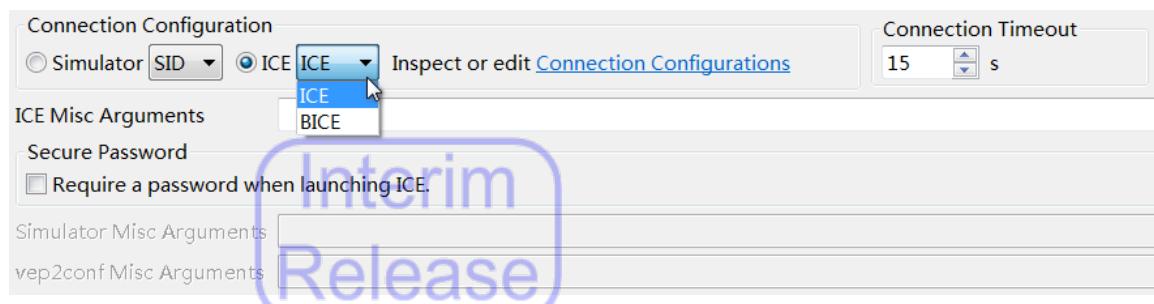


### Target Manager Internal TCP Port Range

Target Manager Internal TCP Port Range			
Start Port	9900		
End Port	49151		

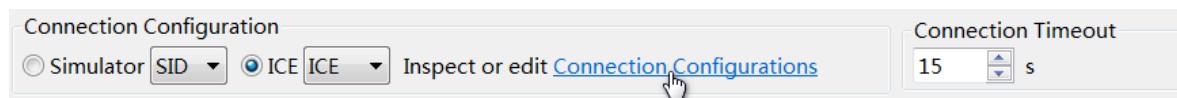
Specify the start port number to enable automatic port allocation.

## Connection Configuration

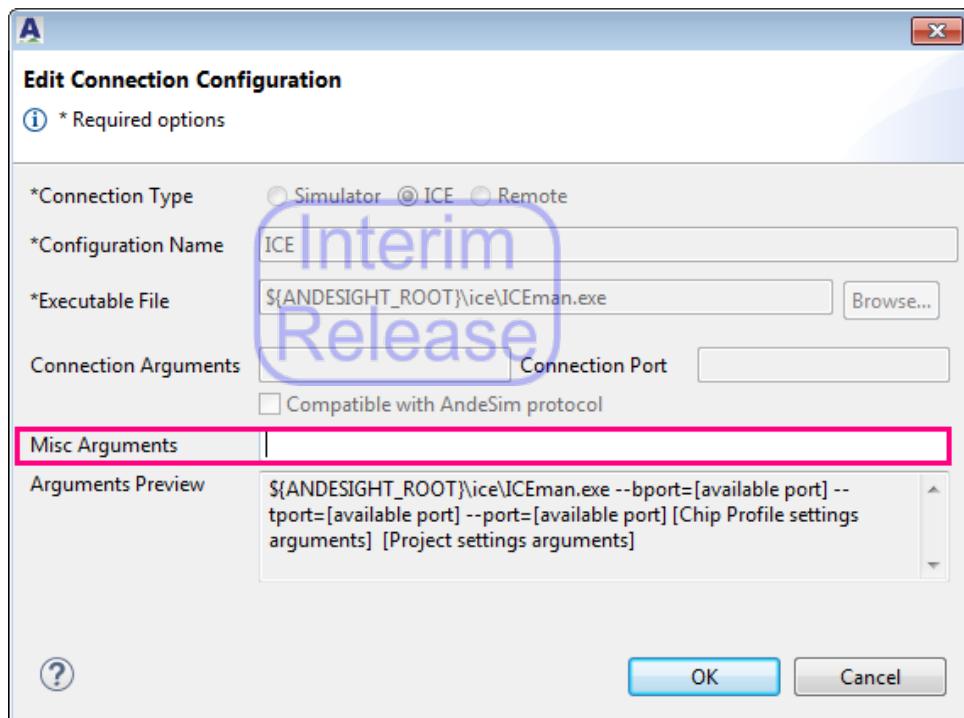


Select a connection configuration from the combo boxes. The combo box on the left includes configurations for targets connected to a local debugging host through a simulator. The combo box on the right includes configurations for targets connected to a local debugging host through an ICE device.

The Simulator/ICE Misc Arguments field displays the connection options specified for a given configuration (if there are any). Click the “Connection Configurations” string to link to the **Connection Configurations** page (Section 2.2.4.1) to examine the available connection configurations and options.



You may also click the Simulator/ICE Misc Arguments field to enter arguments in the invoked **Edit Connection Configuration** dialog. For ICE configurations, make sure that you do not use the ICEman option “`--port`” or “`-p`” (for specifying a port number for GDB connection), as this is not supported in AndeSight. Also, be sure to specify the ICEman option “`--smp`” for a V5 target system with SMP (Symmetric Multiprocessing) support.

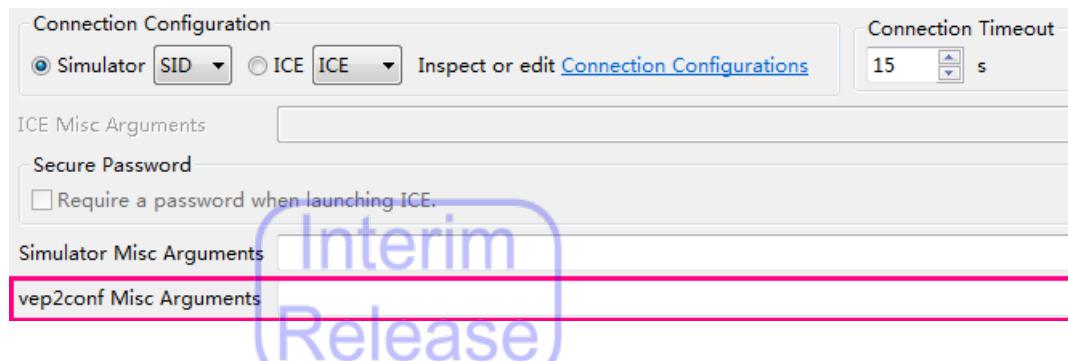


The timeout period for a target connection can be adjusted in the Connection Timeout combo box. The default timeout duration is 15 seconds.

To use an ICE target with an Andes Security processor containing an EDM that requires a password for program development, make sure that you select the checkbox “Require a password when launching ICE”. This will trigger a **Secure Password** wizard (shown below) prompting you to enter a passcode whenever your ICE is launched. For the passcode format, please refer to the *Andes ICE Management Software (ICEmain) User Manual*.



To use a simulator target, you may also specify options for VEP conversion in the vep2conf Misc Arguments field.



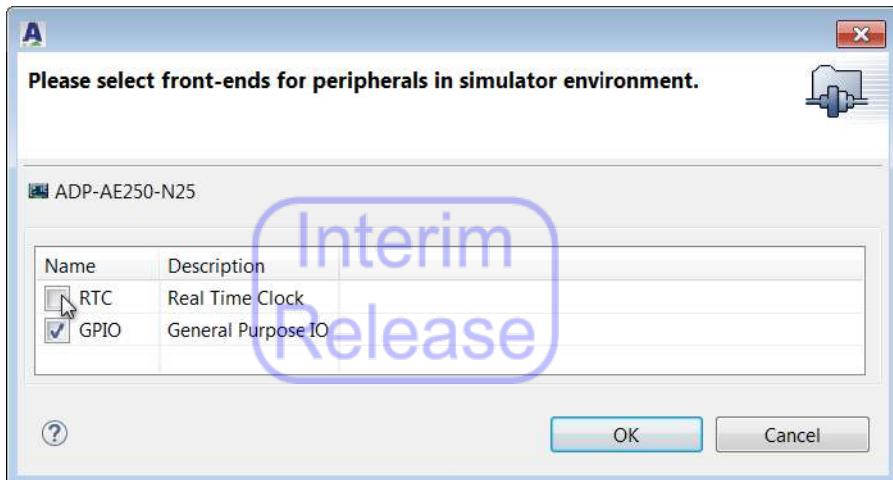
## Chip Profile

Chip Profile

Chip Profile Name	Chip ID	CPU	Simulator Config
<b>AE250</b>			
ADP-AE250-N22	ADP-AE250-N22	[N22]	ADP-AE250-N22.vep
ADP-AE250-N25	ADP-AE250-N25	[N25]	ADP-AE250-N25.vep
ADP-AE250-N25F	ADP-AE250-N25F	[N25F]	ADP-AE250-N25F.vep
<b>AE350</b>			
ADP-AE350-A25	ADP-AE350-A25	[A25]	ADP-AE350-A25.vep
ADP-AE350-A27	ADP-AE350-A27	[A27]	ADP-AE350-A27.vep
ADP-AE350-A45	ADP-AE350-A45	[A45]	ADP-AE350-A45.vep
ADP-AE350-AX25	ADP-AE350-AX25	[AX25]	ADP-AE350-AX25.vep
ADP-AE350-AX27	ADP-AE350-AX27	[AX27]	ADP-AE350-AX27.vep
ADP-AE350-AX45	ADP-AE350-AX45	[AX45]	ADP-AE350-AX45.vep
ADP-AE350-D25F	ADP-AE350-D25F	[D25F]	ADP-AE350-D25F.vep
ADP-AE350-D45	ADP-AE350-D45	[D45]	ADP-AE350-D45.vep
ADP-AE350-N25F	ADP-AE350-N25F	[N25F]	ADP-AE350-N25F.vep
ADP-AE350-N25F-AMP	ADP-AE350-N25F-AMP	[N25F, N25F]	ADP-AE350-N25F-AMP.vep
ADP-AE350-N45	ADP-AE350-N45	[N45]	ADP-AE350-N45.vep
ADP-AE350-NX25F	ADP-AE350-NX25F	[NX25F]	ADP-AE350-NX25F.vep
ADP-AE350-NX27V	ADP-AE350-NX27V	[NX27V]	ADP-AE350-NX27V.vep
ADP-AE350-NX45	ADP-AE350-NX45	[NX45]	ADP-AE350-NX45.vep
<b>Corvette</b>			
ADP-Corvette-F1-D25	ADP-Corvette-F1-D25	[D25]	ADP-Corvette-F1-D25.vep
ADP-Corvette-F1-N22	ADP-Corvette-F1-N22	[N22]	ADP-Corvette-F1-N22.vep
ADP-Corvette-F1-N25	ADP-Corvette-F1-N25	[N25]	ADP-Corvette-F1-N25.vep

Front-end Selection    Setup Memory Latency

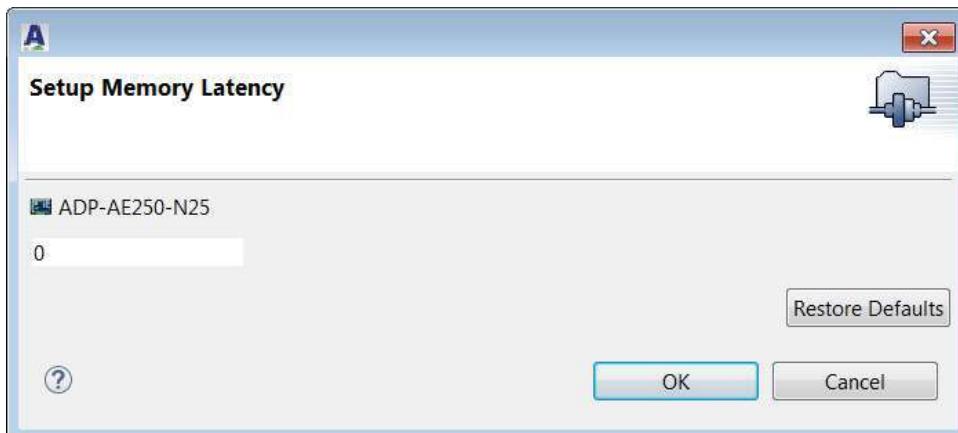
This section allows you to configure front-ends and memory latency for specific simulator targets. From the **Chip Profile** list, select a desired target and click the **Front-end Selection** or **Setup Memory Latency** button to make further selections in the wizards that pop up. The **Front-end Selection** wizard lists the peripherals available for the designated simulator target. The VEP front-ends that you select for a target in this wizard must be evoked via the **Target Manager** view. For details on how to evoke VEP front-ends, please refer to Section 2.2.5.1. To filter out a front-end (i.e., to prevent it from being initiated as the simulator target launches), simply uncheck its check box on the **Front-end Selection** wizard.



## NOTE

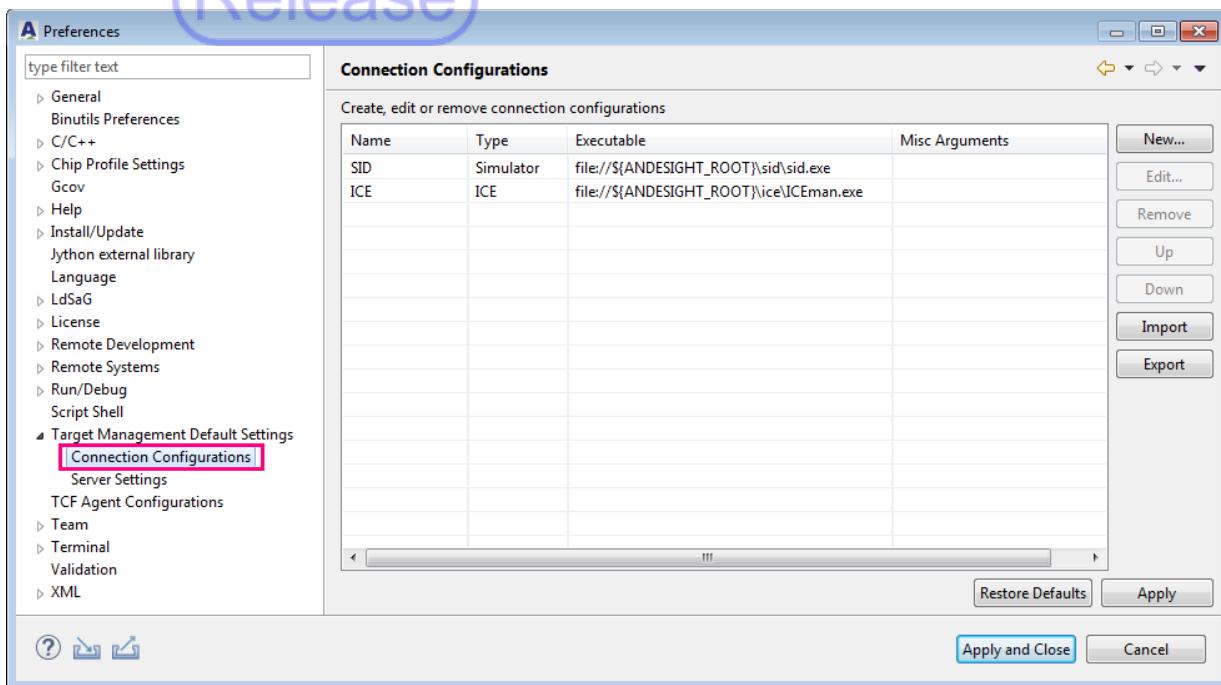
If the VEP front-end filter setting has been specified in the chip profile of the target (targetboard.properties file), then it is not possible to use this wizard to select front-ends.

The **Setup Memory Latency** wizard allows the configuration of memory latency for the simulator target that is selected. Andes simulator targets have a default memory latency of 0 cycle. You may disable or reduce memory latency for your simulator using this wizard.



#### 2.2.4.1 Connection configurations

The **Connection Configurations** page lists the configurations available for target connection. This page can also be used to create, modify, or delete connection configurations. On the AndeSight main menu, select “Window > Preferences” and click “Target Management Default Settings > Connection Configurations” in the navigation pane of the **Preferences** dialog to view this page.



The connection configurations defined on this page facilitate the management of connection settings for project targets. They are displayed in **Connection Configuration** combo boxes of the following AndeSight entries:

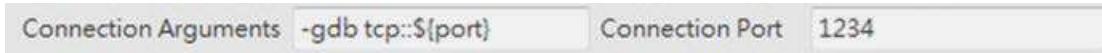
- **Andes Project Creator** view (Section 2.1.1.1)
- **Andes C/C++ Project** wizard (which can be accessed via “AndeSight main menu > File > New > Andes [C/C++] Project)
- **Target Management Default Settings** page (Section 2.2.4)
- **Target Configuration** page (Section 2.1.3)

AndeSight comes with two default connection configurations: “ICE” for ICE targets and “SID” for simulator targets. The buttons on this page enable the creation of new configurations, the removal of unwanted configurations, and the modification of existing configurations.

■ Creating a connection configuration:

Click “New...” to invoke the **New Connection Configuration** dialog and specify the following attributes for new configurations:

- Connection Type:
  - ICE: Targets are connected to a local host via an ICE device.
  - Simulator: Targets are connected to a local host via a simulator.
  - Remote: Targets are connected to a remote host.
- Configuration Name: Enter a name for the new connection configuration.
- Executable File: For ICE or Simulator connection types, enter the path to the ICE or simulator executable file or click “Browse...” to locate the file in the invoked dialog.
- Connection Arguments: For ICE and Simulator connection types, enter options to enable gdb communication via a socket port.  `${port}` can be used as a placeholder for the connection port number in this field.



- Connection Port: For ICE and Simulator connection types, enter a port number for gdb to connect to.
- Compatible with AndeSim protocol: Select this option for connection with a SystemC simulator that has the protocol compatibility with AndeSim™. This will enable the SystemC simulator to run/debug/profile like AndeSim™ though having no VEP file involved in its launch process.
- Misc Arguments: For ICE and Simulator connection types, specify options that you wish to pass to the ICE management software or simulator. If this connection configuration is selected, then these options will be presented in the ICE/Simulator Misc. Argument field of the **Target Management Default Settings** page.

For the options of Andes ICE management software (ICEman), please refer to the *Andes ICE Management Software (ICEman) User Manual*. Avoid using the option “`--port`” or “`-p`” (for specifying a port number for GDB connection), as this is not supported by AndeSight. In addition, for a V5 target with SMP cores, make sure you specify the ICEman option “`--smp`” to enable the SMP support.

- Argument Preview: For ICE and Simulator connection types, this field displays the

order of arguments defined in chip profiles, in this connection configuration and in the project settings.

- Host Name: For the Remote connection type, specify the name or IP address of the remote host to which targets will be connected.
- Host Port: For the Remote connection type, specify a port number for gdb to connect to.

---

**NOTE**

The host name and port number specified here are used to launch a run/debug/profile session for a project using this remote configuration, unless you manually modify them in the run/debug/profile configuration. For more information concerning remote connection settings in run/debug/profile configurations, see Section 2.4.3.1, Debugger tab.

---

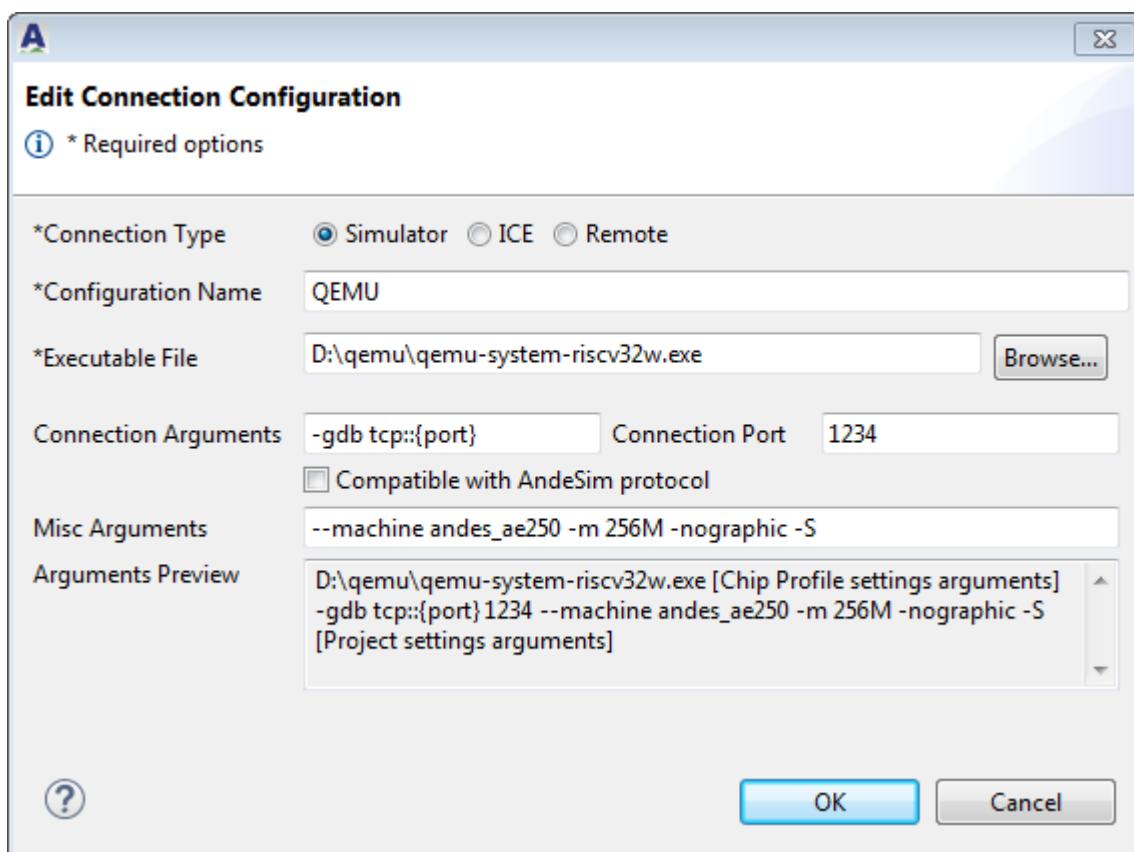


Figure 22. Editing a connection configuration with Simulator connection type

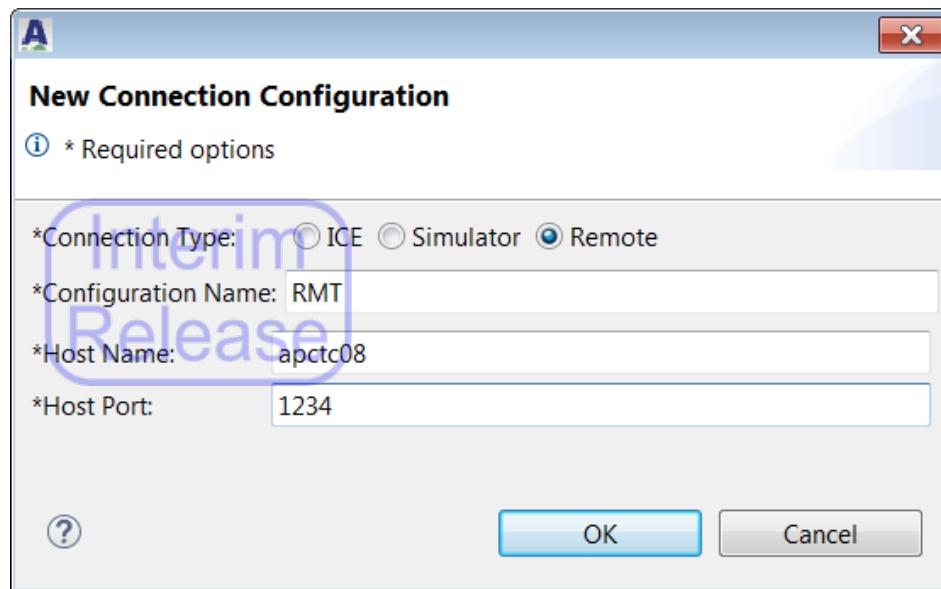


Figure 23. Creating a new connection configuration with Remote connection type

■ **Modifying a connection configuration:**

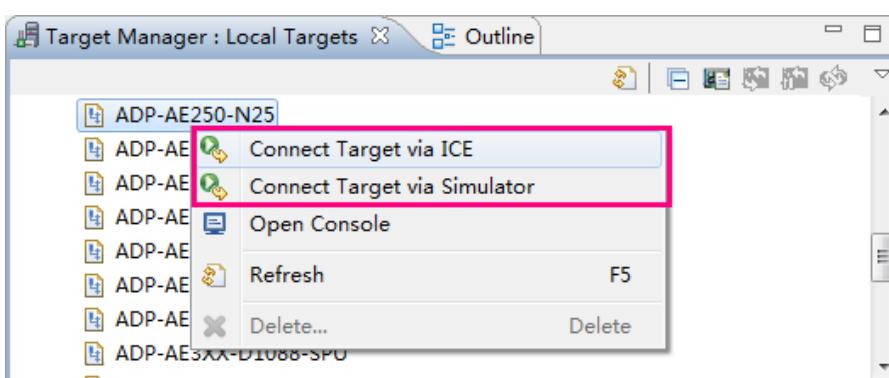
Select a connection configuration and click “Edit” to modify it on the **Edit Connection Configuration** dialog.

■ **Deleting a connection configuration:**

Select a connection configuration and click “Remove”.

■ **Reordering connection configurations:**

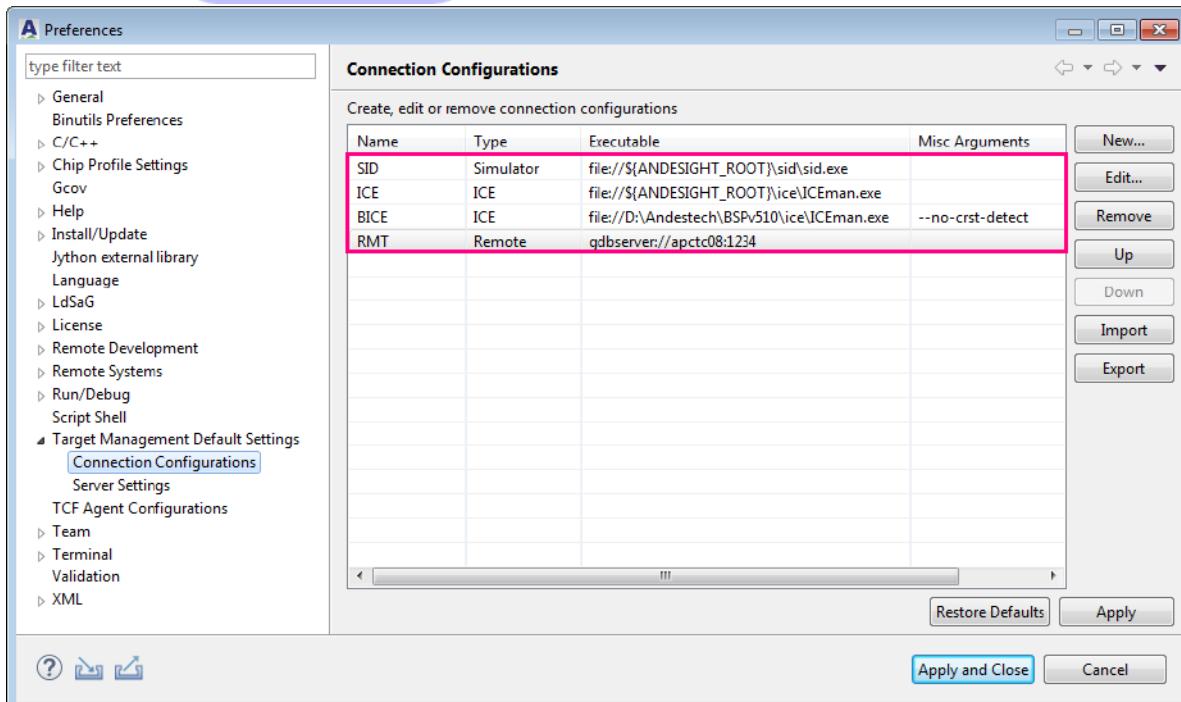
Select a connection configuration and click “Up” or “Down” to move it to the desired position in the list. The highest-ranking connection configuration with an ICE or Simulator connection type is applied when target connection of that type is established via the **Target Manager** view.



■ Importing/Exporting a connection configuration:

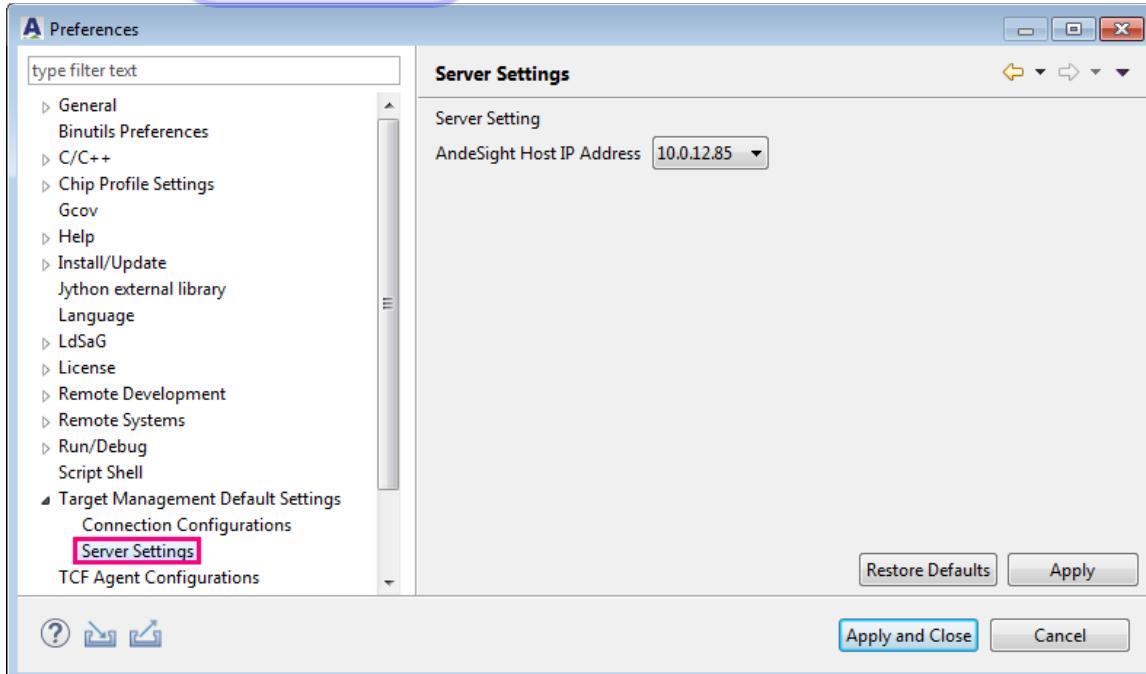
Click “Import” to import a .json file that defines connection configurations or click “Export” to export defined connection configurations to a .json file.

The connection configurations that you define are listed in the Connection Configurations table on the **Preferences** dialog. Make sure you click “Apply” or “Apply and Close” to save your changes.



#### 2.2.4.2 Server settings

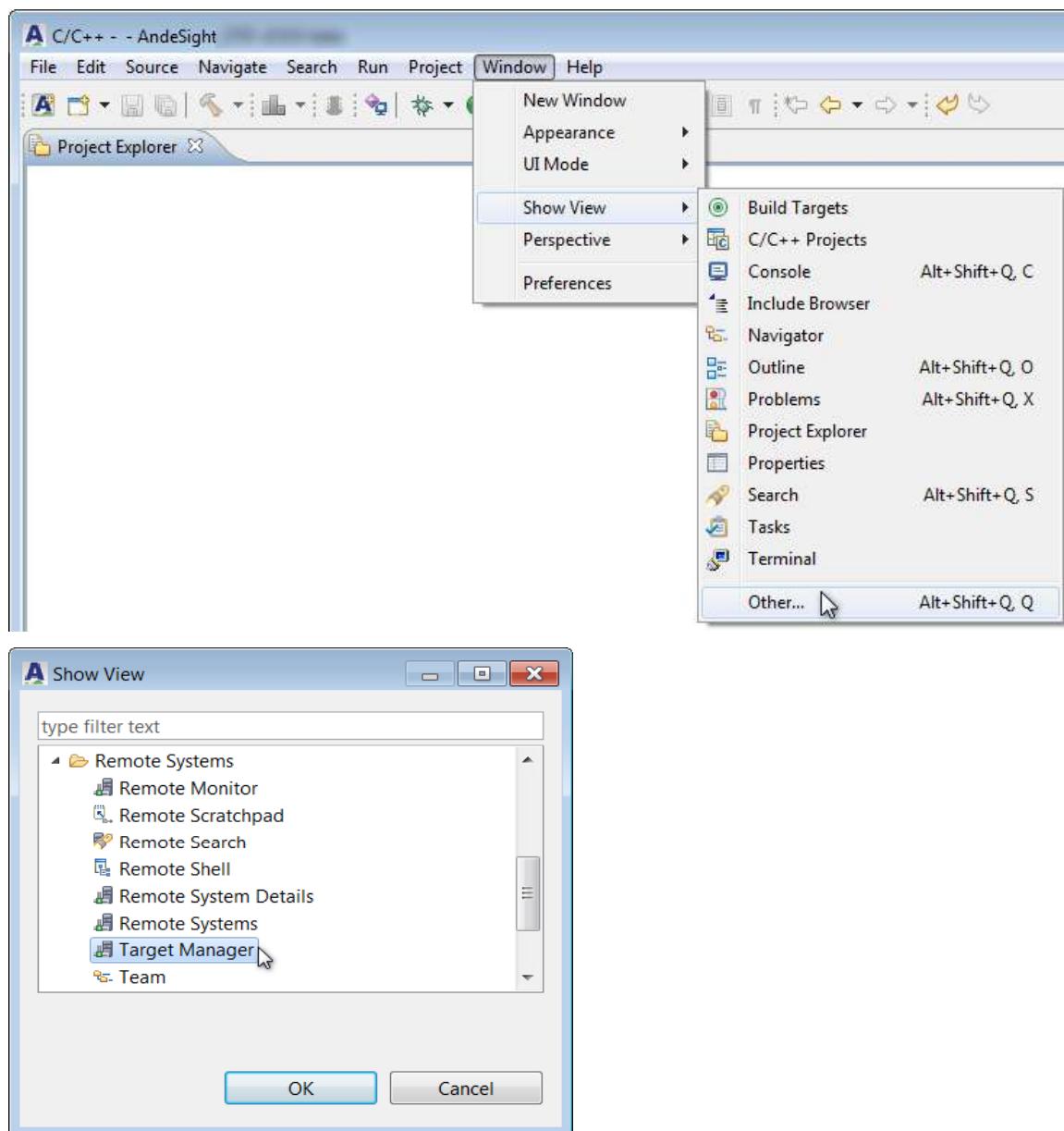
The **Server Settings** page allows you to specify the IP address of AndeSight in cases where the host system has multiple IP addresses. On the AndeSight main menu, select “Window > Preferences” to invoke the **Preferences** dialog and then select “Target Management Default Settings > Server Settings” in the navigation pane to find the page and specify the IP address.



## 2.2.5. Target Manager

AndeSight uses the **Target Manager** to administer AndeSight connections to target systems. The **Target Manager** automatically establishes and manages communications between hosts and targets in the background. It is also possible to manually connect to a variety of target systems and manage them using the **Target Manager** view.

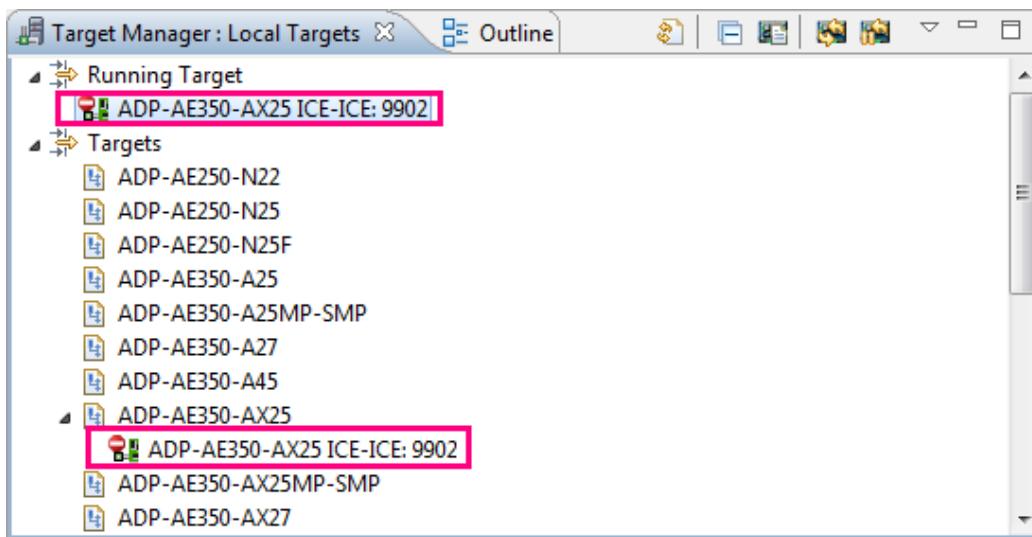
By default, the **Target Manager** view appears in the lower left of the AndeSight workbench. If you cannot locate it in the current perspective, then you may evoke this view by selecting “Window > Show View > Other...” from the AndeSight main menu and then selecting “Remote Systems > Target Manager” in the **Show View** dialog.



Target systems supported by the current AndeSight installation are listed under Targets or Generic Targets in the **Target Manager** view.



After a target is launched, its connection type, connection configuration, and the port number for the gdb connection appear in its expanded tree. The launched target may also be found in the Running Target group.



The **Target Manager** view also displays the connection status of each target, including the following:

### Available



The target is available for connection and program development.

## Launched



The target is connected to the host but not loaded with a program yet.

## Busy



The target is launched, loaded with a program and engaged in program development.

A launched target is also linked to a target button  in the lower right of the AndeSight workbench. This button also provides information concerning target properties and connections.

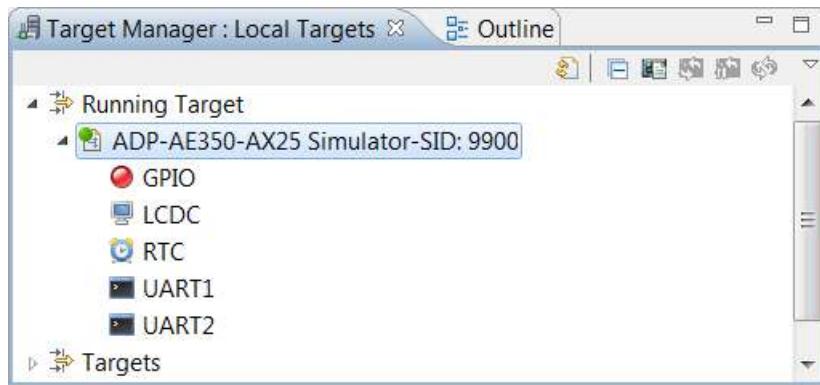


### 2.2.5.1 VEP front-ends in Target Manager view

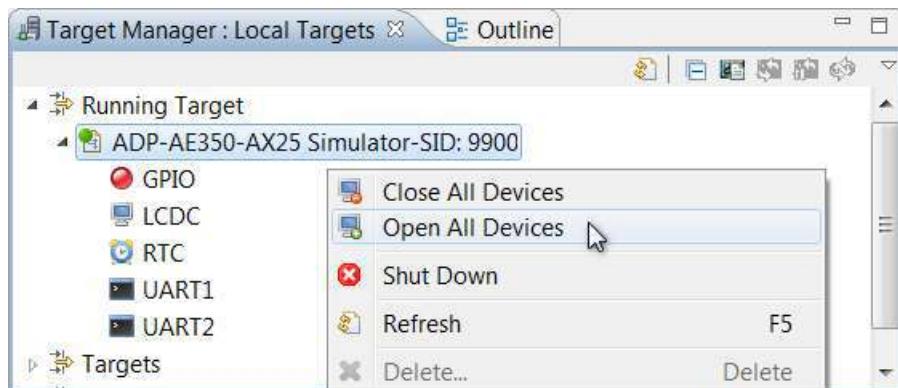
The front-ends of a VEP file simulate the peripherals connected to a simulator target. They are listed with the simulator target in the **Target Manager** view for ease of monitoring.

Follow the steps below to evoke VEP front-ends from the **Target Manager** view:

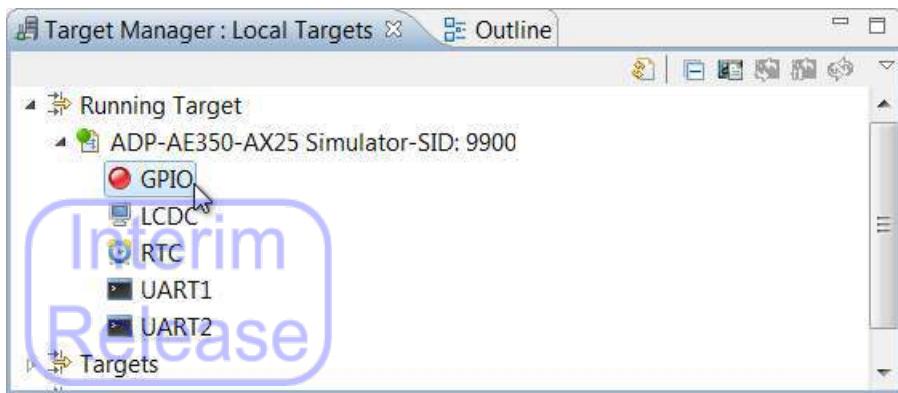
- Step 1** After a simulator target is successfully launched, locate it in the “Running Target” group of the **Target Manager** view.



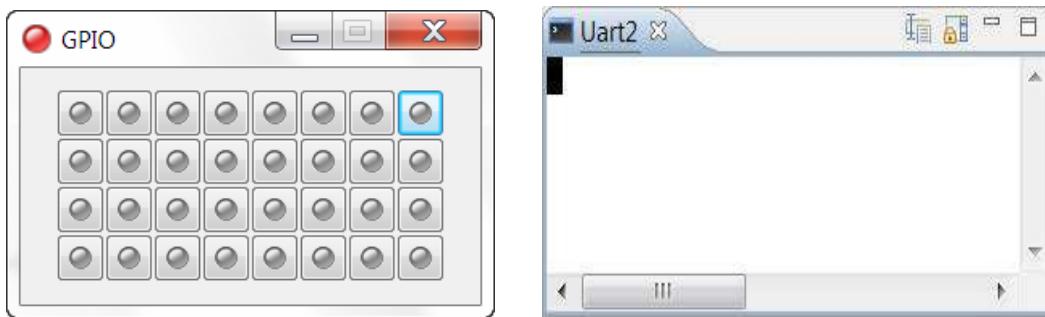
- Step 2** To evoke all VEP front-ends of the simulator target, right-click the running target and select “Open All Devices” in the pull-down menu.



To evoke a specific VEP front-end, double-click the running target and find its front-ends in the expanded tree of the target. Select the front-end of interest and double-click it.



**Step 3** The evoked VEP front-ends will appear either as pop-up dialog or an attached view on the AndeSight workbench.



---

**NOTE**

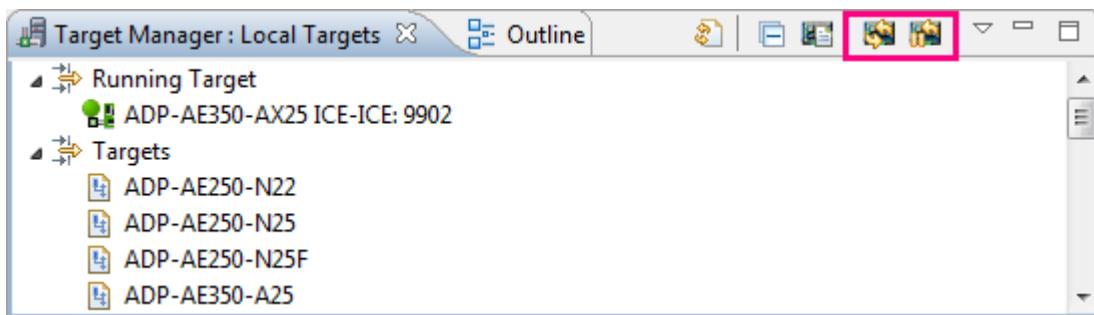
Evoked VEP front-ends are visible only in the perspective from which they were evoked. Monitoring the same virtual I/Os after a switch of perspective requires the evocation of VEP front-ends from the **Target Manager** again.

---

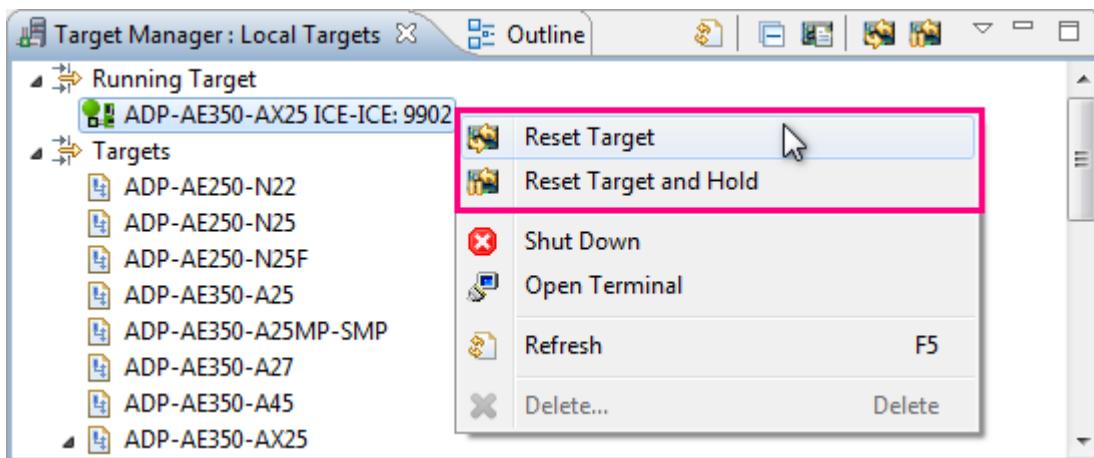
## 2.2.6. Resetting a target board or an ICE

AndeSight provides a handy UI to reset your target board in the event of abnormal behavior.

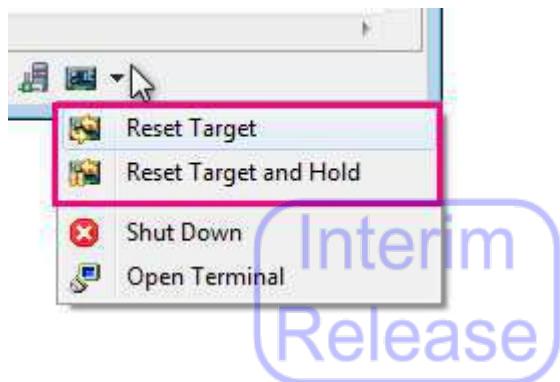
To reset an ICE target or ICE, select a running ICE target in the **Target Manager** view. On the toolbar of this view, click  (Reset Target) to reset the real board or  (Reset Target and Hold) to reset the board and hold the CPU at PC=0 position. The debug session will shut down if any of these buttons are clicked during execution of the program.



An ICE target or ICE can also be reset from the pull-down menu of the target. Right-click the target to evoke a pull-down menu and make a selection between “Reset Target” and “Reset Target and Hold”.

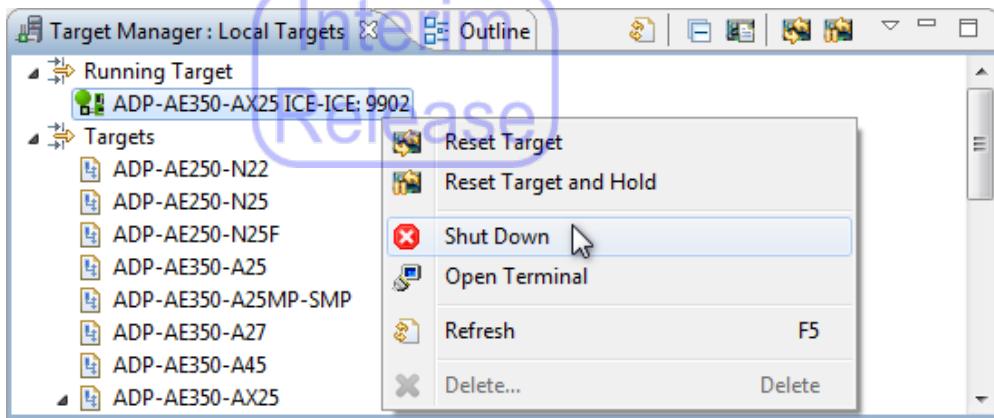


Target buttons also make it possible to reset targets. Locate the desired target button  in the lower right of the AndeSight workbench and click  to select between “Reset Target” and “Reset Target and Hold” in the pull-down menu.



### 2.2.7. Shutting down a target

To terminate a running target, select it in the **Target Manager** view, right-click and select “Shut Down” from the pull-down menu.



Target buttons can also be used to shut down targets. Locate the desired target button  in the lower right of the AndeSight workbench, click  and select “Shut Down” from the pull-down menu.

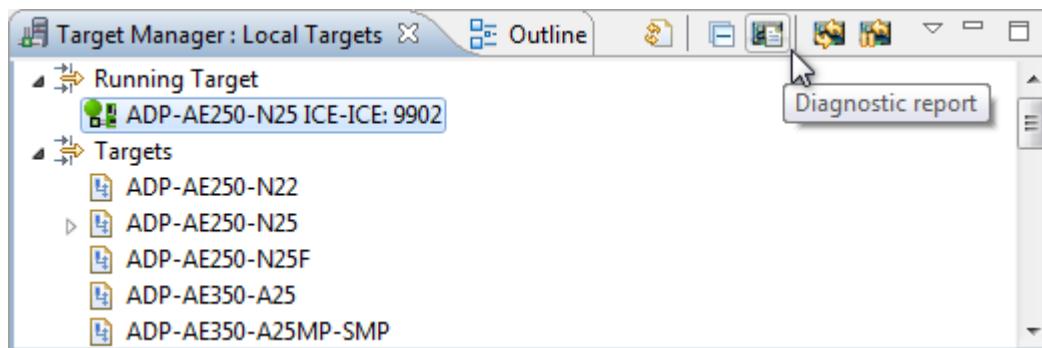


## 2.2.8. Diagnosing your target board or ICE

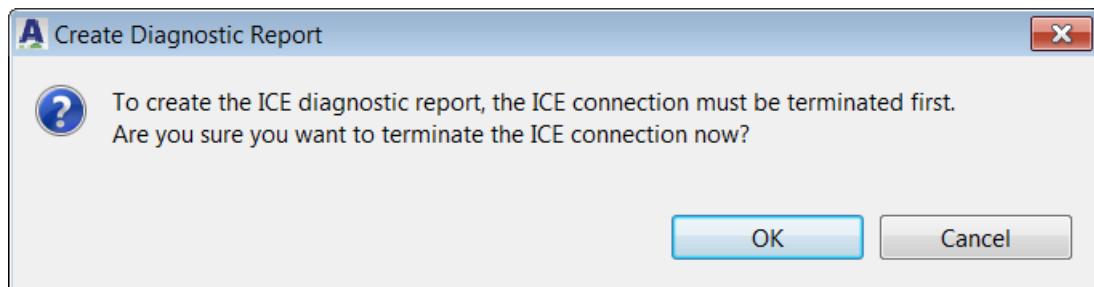
In the event that the debugger is not executed as expected, AndeSight provides functions to easily diagnose the problem. Follow the steps below to obtain a diagnostic report for your ICE target or ICE:



- Step 1** In the Target Manager view, select the desired running ICE target and click the “Diagnostic report” button  on the toolbar.



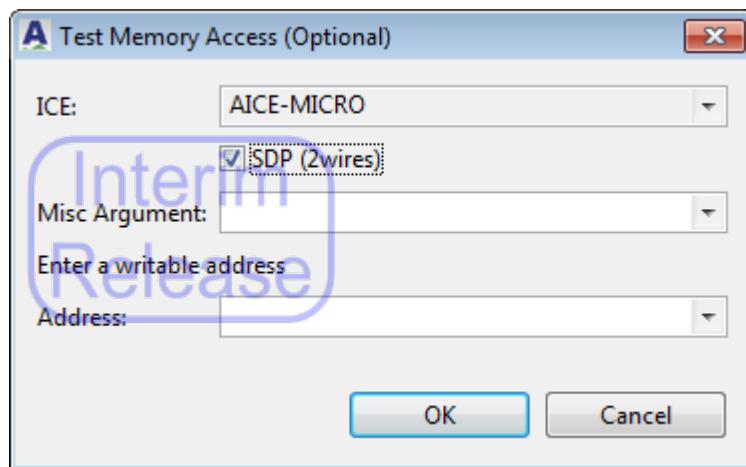
- Step 2** At the prompt, click “OK” to terminate the target-ICE connection.



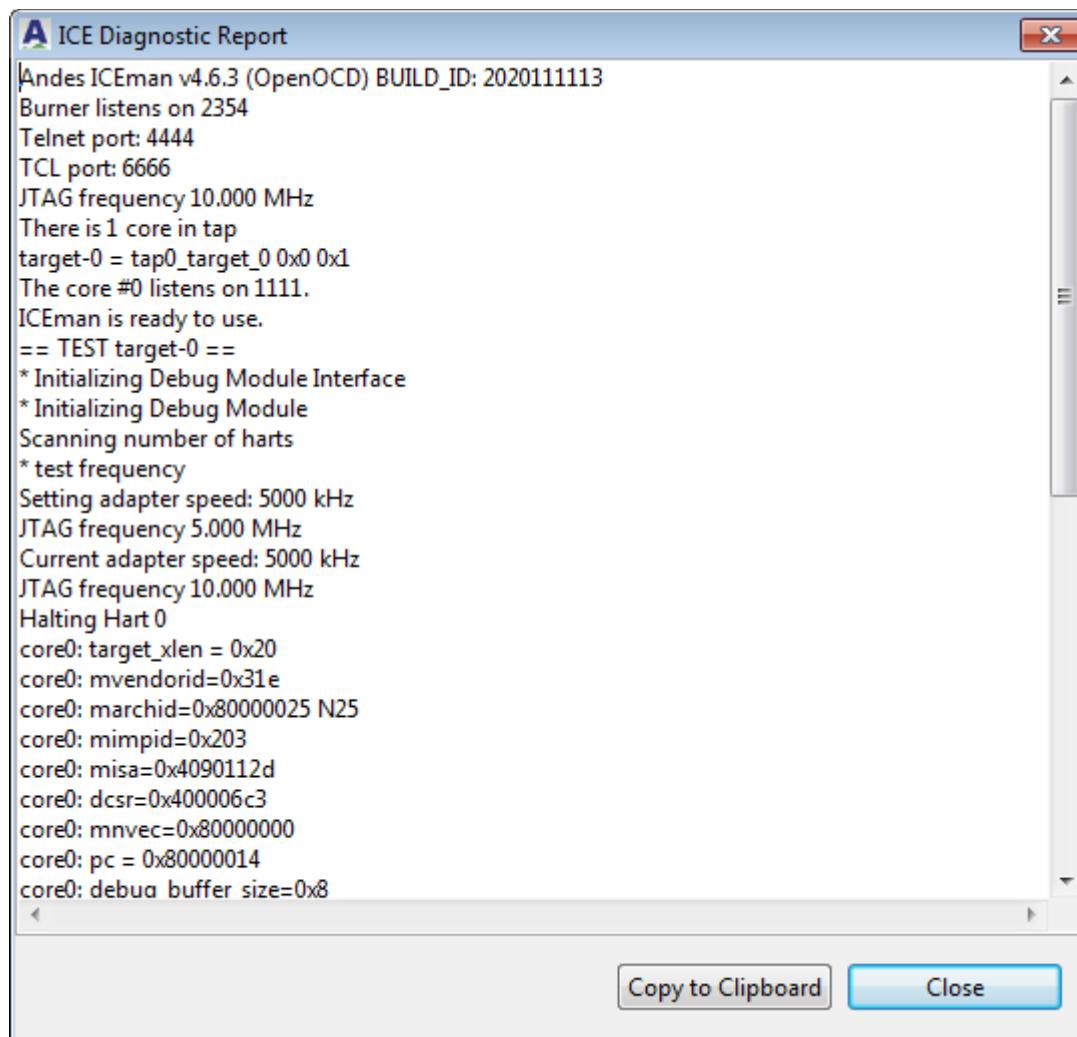
- Step 3** The invoked dialog shows the ICE device in use. AndeSight can recognize AICE-MINI+ and AICE-MICRO. As to other ICE devices for Andes V5 targets such as Olimex FTDI, they will be shown as “Other” in the ICE combo box. For AICE-MICRO or AICE-MINI+ that’s connected to a target board of 2-wire debug interface, make sure to select the option “SDP (2 wires)” for the detected ICE device.

Next, optionally enter connection arguments for your ICE device and

input a writable address to test the memory access.



- Step 4** The diagnostic report for the debugger will be generated and displayed in the dialog that pops up.



## 2.3. Flash programming

AndeSight provides two approaches to facilitate programming and reprogramming flash memory of microcontrollers on target systems. The first approach requires a flash burner, PAR\_burn or SPI\_burn, to communicate with ICEman using socket protocols.

The other programming approach requires a flash burner, target\_burn\_frontend, to communicate with ICEman using telnet protocols and a target application, target\_burn, to program the flash memory directly. Compared with the first approach, this approach accelerates the programming process.

Section 2.3.1 and 2.3.2 below explain the two programming methods in details and describe how to build the required files (i.e. flash burner and target application) for the programming. Section 2.3.3 introduces how to change flash programming settings in your chip profile. If you have had a built burner executable and target\_burn application, follow instructions in Section 2.3.4 to perform flash programming directly using AndeSight UI.

### 2.3.1. Flash programming using a flash burner

This programming method requires a flash burner, SPI\_burn or PAR\_burn, to communicate with ICEman using socket protocols. ICEman reads from/writes to the target via Debug Module using Andes-defined API. The programming method is illustrated below.

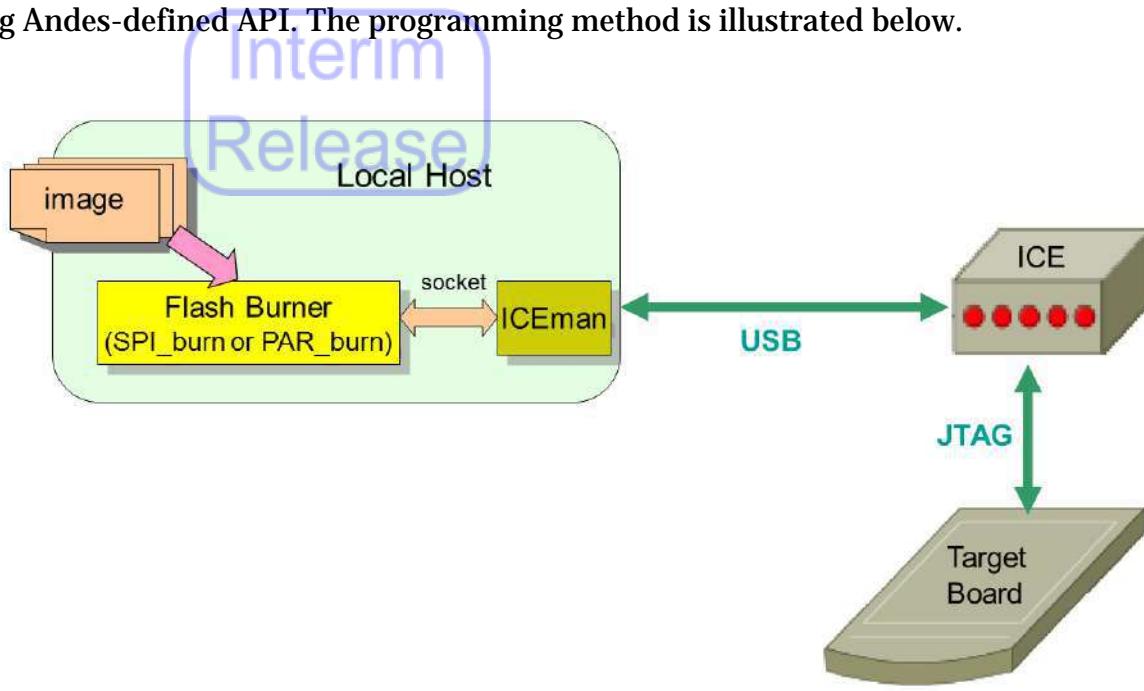


Figure 24. In-system programming via PAR\_burn or SPI\_burn

Andes-defined API functions required for communication between a burner utility and ICEman are outlined in the following:

- **outw/outh/outb(ADDRESS, DATA)** : Write word/half word/byte data to the target.
- **i nw/i nh/i nb(ADDRESS)** : Read word/half word/byte data from the target.
- **fastin(ADDRESS, SIZE, BUFFER)** : Use ICEman to fill the buffer with multi-word data read from the target. When Bit 1 of memory addresses from which the data is read has a value of 0, the addresses increment automatically by 4 (i.e., a word size) after each read access.
- **fastout(ADDRESS, SIZE, BUFFER)** : Write multi-word data from the target by ICEman. When Bit 1 of memory addresses from which the data is written has a value of 0, the addresses increment automatically by 4 (i.e., a word size) after each write access.
- **fastbytein(ADDRESS, SIZE, BUFFER, IF\_CONST\_ADDR)** : Use ICEman to fill the buffer with multi-byte data read from the target. When **IF\_CONST\_ADDR** is set to 0, the addresses increment automatically by 1 (i.e., a byte size) after each read access.
- **fastbyteout(ADDRESS, SIZE, BUFFER, IF\_CONST\_ADDR)** : Write multi-byte data from the

target by ICEman. When **IF\_CONST\_ADDR** is set to **0**, the addresses increment automatically by 1 (i.e., a byte size) after each write access.

Use the following API function to specify actions that ICEman must apply to your target or ICE:

- **send\_cmd (char CMD)**: Send a command to ICEman. The **CMD** command can be any of the following:
  - **RESET\_TARGET**: Reset the target board.
  - **RESET\_HOLD**: Reset the target board and make it stop at \$IVB. In other words, halt the CPU before the boot code is executed.

#### 2.3.1.1 Building PAR\_burn or SPI\_burn burner

AndeSight provides pre-built PAR\_burn and SPI\_burn burners, **PAR\_burn.exe** and **SPI\_burn.exe**, under **ANDESIGHT\_ROOT\flash\bin**.

To meet your own programming needs, you can proceed with the following steps to modify the source of the PAR\_burn/SPI\_burn burner in AndeSight and build a new burner.

**Step 1** Go to **ANDESIGHT\_ROOT\flash\src-[PAR\_burn|SPI\_burn]**.

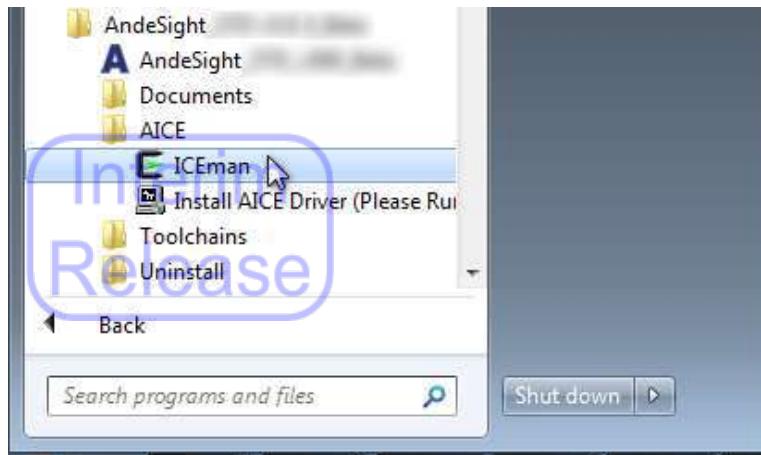
**Step 2** Open **main.c**. Look for all **TODO** comments in the file and configure values relevant to your target system.

**Step 3** Open **[smcflash-Intel J3|smcflash-Micron|spi flash-MXIC].c**. Modify the file for your flash programming following the instructions in its **TODO** comments.

**Step 4** For SPI\_burn only, if the SPI-ROM of your target system has slower working frequency and can't be accessed successfully, open **platform.c**, look for the macros **SPI\_TX\_FIFO** and **SPI\_RX\_FIFO** (counters to access the data port of the SPI controller), and decrease their values.

**Step 5** On the Windows Start Menu, click "Start > All Programs > Andestech

> AndeSight VERSION > AICE > ICEman" to invoke the Cygwin environment.



**Step 6** Export the location of the toolchain to the shell PATH environment.

For example:

```
$ export
PATH=/cygdrive/d/AndesTech/AndeSight/toolchains/nds32elf-mculib-v5/bin:$PATH
```

**Step 7** Change the current directory to where the source code of the flash burner locates, **ANDESIGHT\_ROOT\flash\src-[SPI\_burn|PAR\_burn]**.

**Step 8** Execute the build script, **build\_[SPIburn|PARburn].sh**, to build the burner utility.

```
$ ./build_[SPIburn|PARburn].sh
```

---

#### NOTE

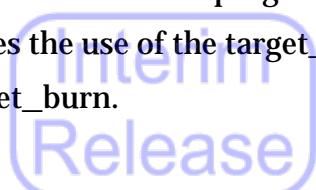
To ensure that the burner utility is operable in Windows, make sure that the x86 MinGW toolchain is installed before building the burner.

**Step 9** The executable file of the flash burner, **SPI\_burn.exe** or **PAR\_burn.exe**, is built in the same folder. To obtain help messages for the burner utility, issue:

```
./[PAR_burn.exe|SPI_burn.exe] -help
```

### 2.3.2. Flash programming using a flash burner and a target application

The programming method illustrated in Figure 24 does not always ensure efficient programming on V5 targets. To accelerate the programming process, a different approach is suggested. This approach requires the use of the target\_burn\_frontend burner utility along with a target application, target\_burn.



The target\_burn\_frontend utility serves as a pure interface in the programming process to obtain and output information. The communication between target\_burn\_frontend and ICEman is carried out using telnet protocols. The target\_burn application provides the basic flash programming functions. It runs on the target system and reads from/writes to the memory/flash directly. According to the commands designated by target\_burn\_frontend and the image file for programming, ICEman fills the command buffer in the target\_burn application and control the programming flow. The programming method is illustrated below.

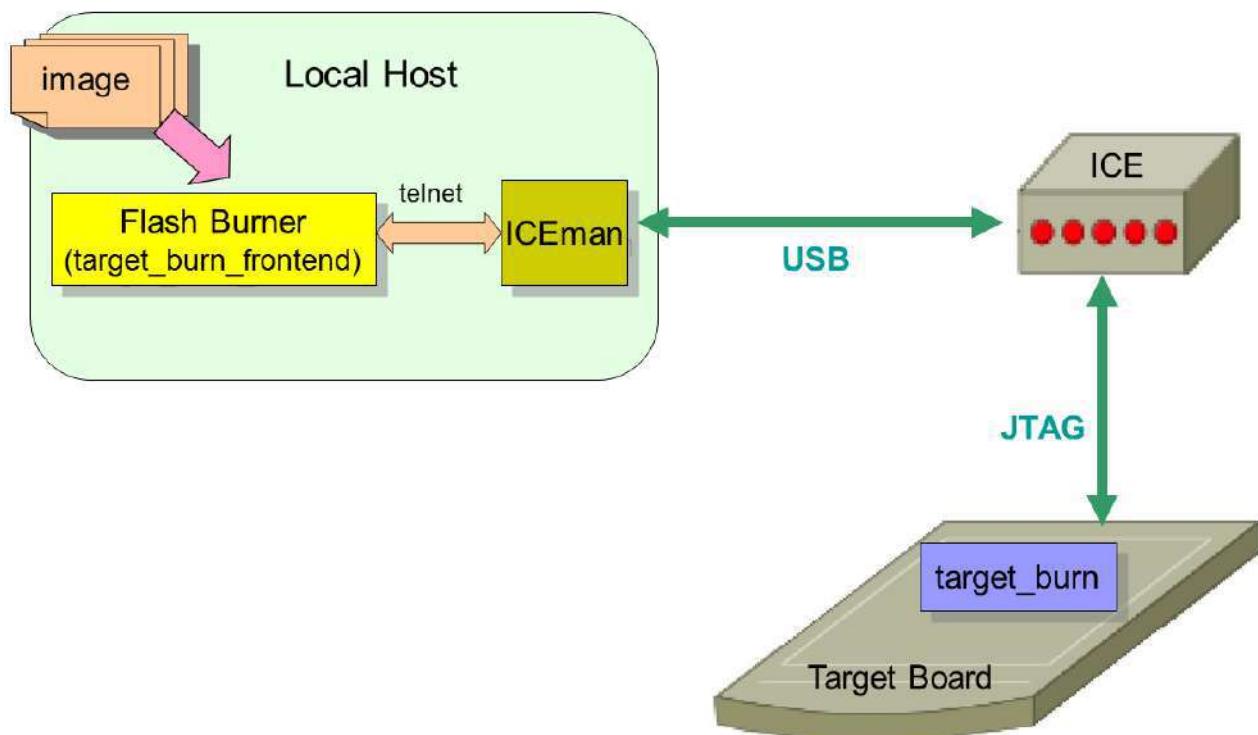


Figure 25. In-system Programming by target\_burn\_frontend and target\_burn

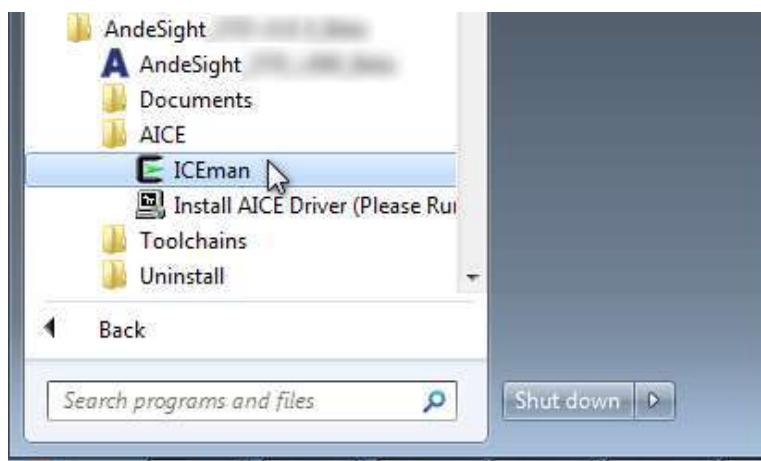
### 2.3.2.1 Building target\_burn\_frontend burner

AndeSight includes a ready-to-use target\_burn\_frontend burner, **target\_burn\_frontend.exe**, under **ANDESIGHT\_ROOT\flash\bin** to work in conjunction with a target\_burn application for flash programming.

You may also build the burner on your own following the steps below.

**Step 1** Unzip the compressed target\_burn source (**target\_burn.zip**) under **ANDESIGHT\_ROOT\flash\src-target\_burn** to generate a “target\_burn” directory. This directory contains the source code of the target\_burn\_frontend burner in the subdirectory “target\_burn\_frontend.”

**Step 2** On the Windows Start Menu, click “Start > All Programs > Andestech > AndeSight VERSION > AICE > ICEman” to invoke the Cygwin environment.



**Step 3** Export the location of the toolchain to the shell PATH environment.

For example:

```
$ export  
PATH=/cygdrive/d/Andestech/AndeSight/toolchains/nds32le-elf-mculibv5/bin:$PATH
```

**Step 4** Change the current directory to where the source code of the target\_burn\_frontend burner locates.

**Step 5** Execute the build script, `build_target_burn_frontend.sh`, to build the burner utility.

```
$ ./build_target_burn_frontend.sh
```

---

**NOTE**

To ensure that the burner utility is operable in Windows, make sure that the x86 MinGW toolchain is installed before building.

**Step 6** The executable file of the flash burner, `target_burn_frontend.exe`, is built in the same folder. To obtain help messages for the burner utility, issue:

```
./target_burn_frontend.exe -help
```

### 2.3.2.2 Building target\_burn application

#### Using target\_burn for SPI flash programming

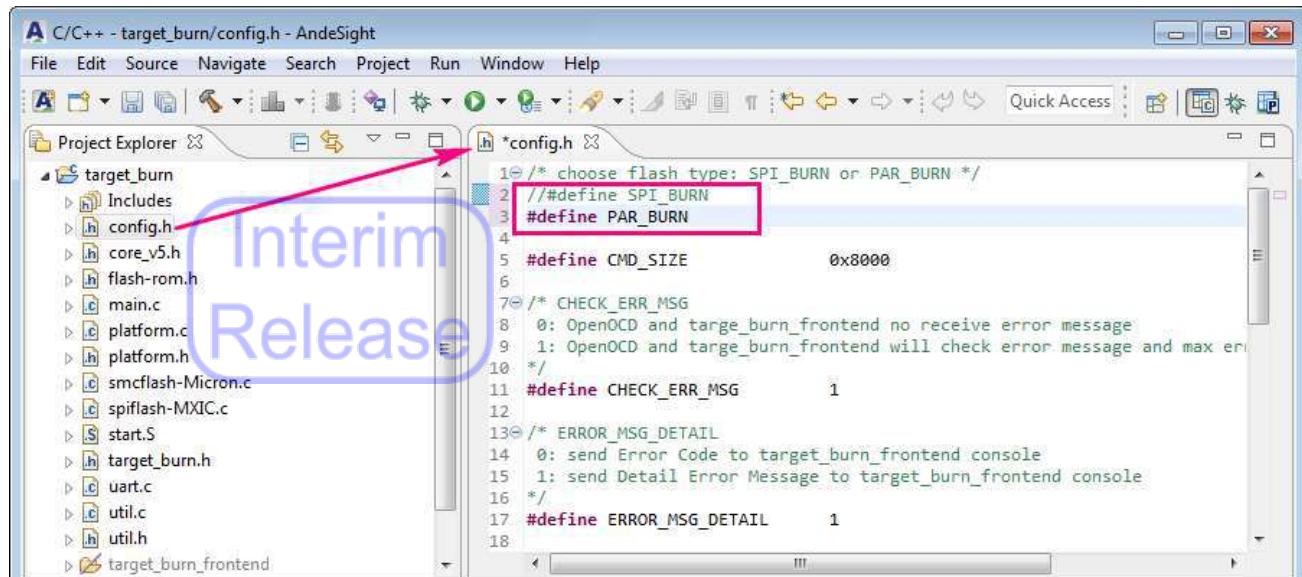
For SPI flash programming on Andes 32/64-bit V5 targets, you can use the pre-built target\_burn application in AndeSight directly. The binaries of the target\_burn application `target_SPI_v5_[32|64].bin` are located under `ANDESIGHT_ROOT\flash\target_bin`.

#### Building target\_burn for parallel flash programming

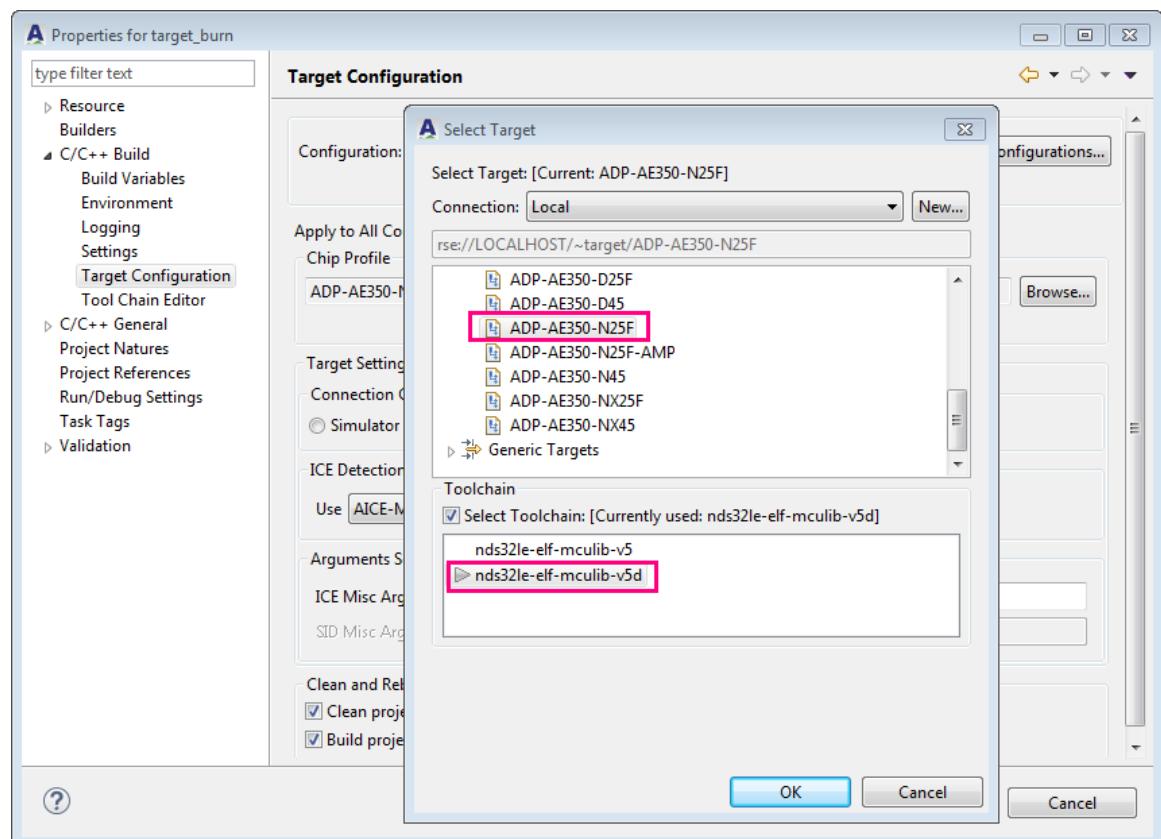
If you intend to program a parallel flash on Andes V5 targets, proceed with the following steps to build a new target\_burn application:

**Step 1** Follow Section 2.1.1.3 to import the compressed target\_burn project (`target_burn.zip`) from `ANDESIGHT_ROOT\flash\src-target_burn` into Project Explorer.

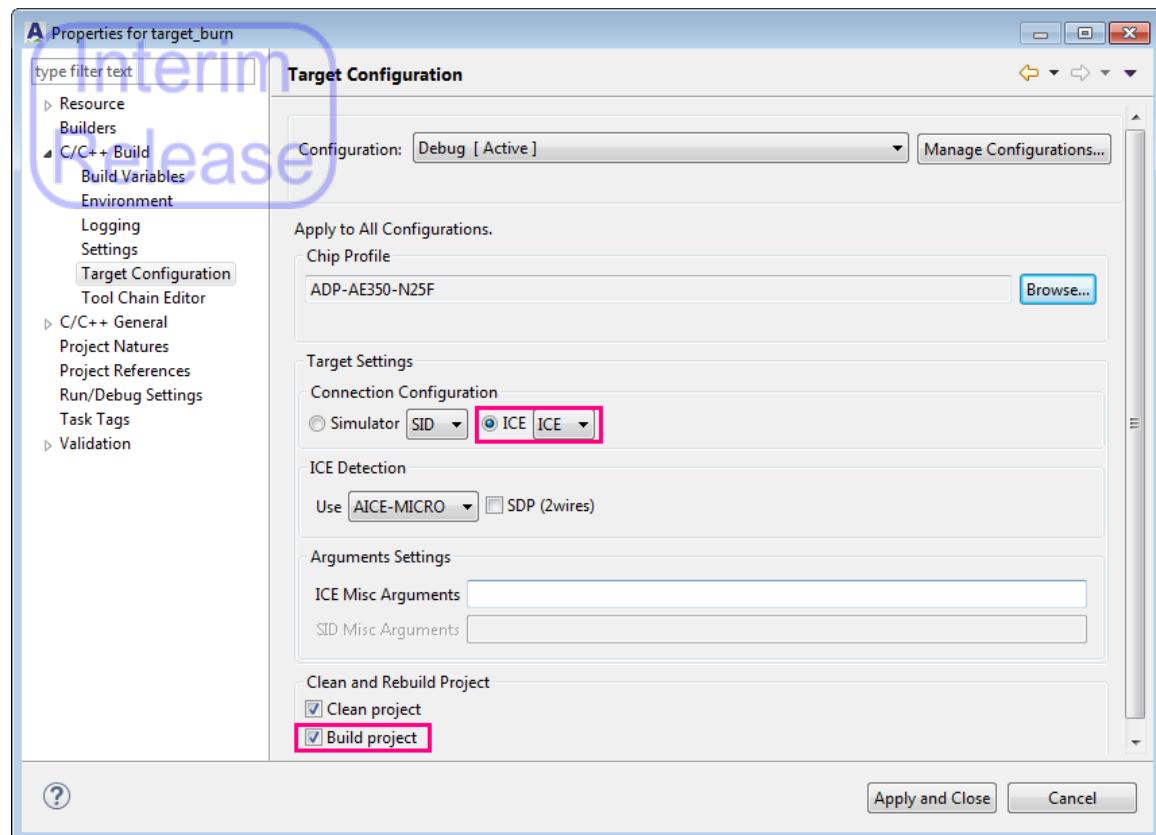
**Step 2** In Project Explorer, double-click `config.h` under the target\_burn project to open it in the editor. Comment the macro “`#define SPI_BURN`” and uncomment “`#define PAR_BURN`” in the file.



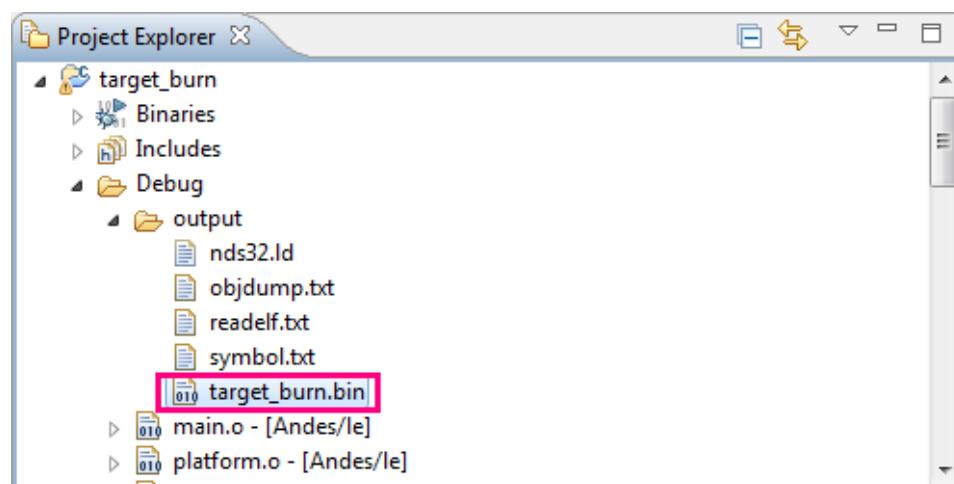
**Step 3** Right-click the project folder and select “Target Configuration” from the pull-down menu. In the invoked dialog, click “Browse...” in the **Chip Profile** section to specify a chip profile and a toolchain that is compatible with your target system.



**Step 4** In the **Properties** dialog, specify “ICE” as the connection type and select the “Build project” option. Then, click “Apply and Close” to commence the build process.



**Step 5** The binary of the target application, `target_burn.bin`, is built under `target_burn\Debug\output`. It is the program you need to load to a V5 target system for parallel flash programming.



### Building target\_burn for programming other flash

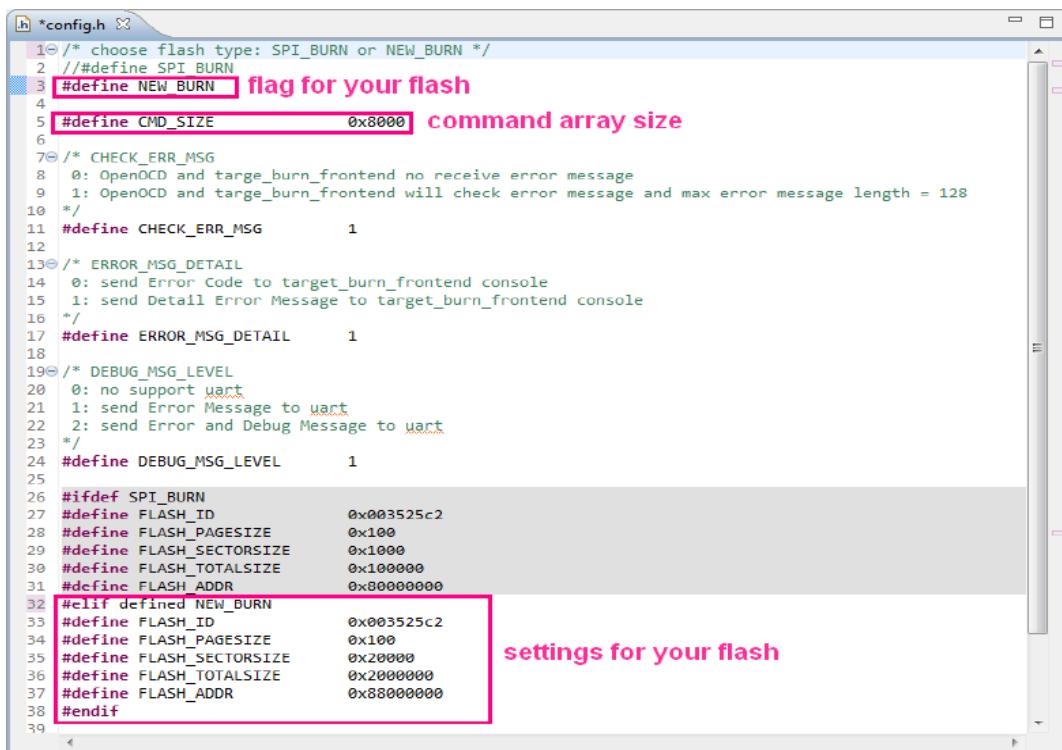
The default target\_burn application supports SPI or parallel flash programming. If you want to use this application to program a different flash type, please proceed as follows to modify its configuration and build a new binary:

**Step 1** Follow Section 2.1.1.3 to import the compressed target\_burn project (`target_burn.zip`) from `ANDESIGHT_ROOT\flash\src-target_burn` into **Project Explorer**.

**Step 2** In **Project Explorer**, double-click `config.h` under the target\_burn project to open it in the editor. Define the following in the file:

- A flag for your flash
- Settings for your flash, including its page size, sector size, total size and memory address
- `CMD_SIZE`, which is the size of the command buffer that ICEman fills to control the programming flow. Its value must satisfy the following constraint:

$$\text{CMD\_SIZE} \geq (\text{FLASH_PAGESIZE} + 8 \text{ bytes})$$

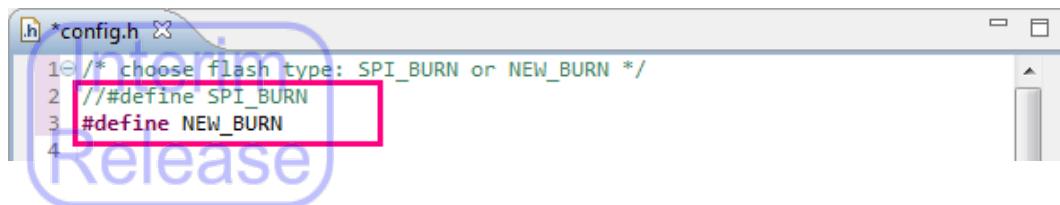


```

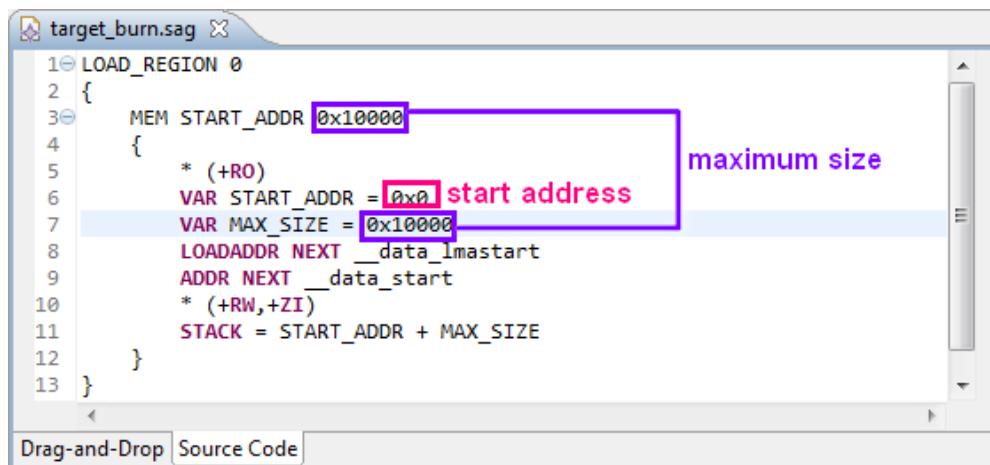
1 /* choose flash type: SPI_BURN or NEW_BURN */
2 // #define SPI_BURN
3 #define NEW_BURN flag for your flash
4
5 #define CMD_SIZE          0x8000  command array size
6
7 /* CHECK_ERR_MSG
8  0: OpenOCD and target_burn_frontend no receive error message
9  1: OpenOCD and target_burn_frontend will check error message and max error message length = 128
10 */
11 #define CHECK_ERR_MSG      1
12
13 /* ERROR_MSG_DETAIL
14  0: send Error Code to target_burn_frontend console
15  1: send Detail Error Message to target_burn_frontend console
16 */
17 #define ERROR_MSG_DETAIL    1
18
19 /* DEBUG_MSG_LEVEL
20  0: no support uart
21  1: send Error Message to uart
22  2: send Error and Debug Message to uart
23 */
24 #define DEBUG_MSG_LEVEL     1
25
26 #ifdef SPI_BURN
27 #define FLASH_ID           0x003525c2
28 #define FLASH_PAGESIZE       0x100
29 #define FLASH_SECTORSIZE     0x1000
30 #define FLASH_TOTALSIZE      0x100000
31 #define FLASH_ADDR          0x80000000
32 #elif defined NEW_BURN
33 #define FLASH_ID           0x003525c2
34 #define FLASH_PAGESIZE       0x100
35 #define FLASH_SECTORSIZE     0x200000
36 #define FLASH_TOTALSIZE      0x20000000
37 #define FLASH_ADDR          0x88000000
38#endif
39

```

Also, make sure that you comment the macro “#define SPI\_BURN”, enable the flag for your flash by uncommenting it, and press “Ctrl + S” to save the changes.



**Step 3** Open `target_burn.sag` under the project. Specify the start address (`START_ADDR`) and maximum size (`MAX_SIZE`) for flash programming in the file, and press “Ctrl + C” to save the changes. The maximum size is defined in two parts in the file, as shown below.



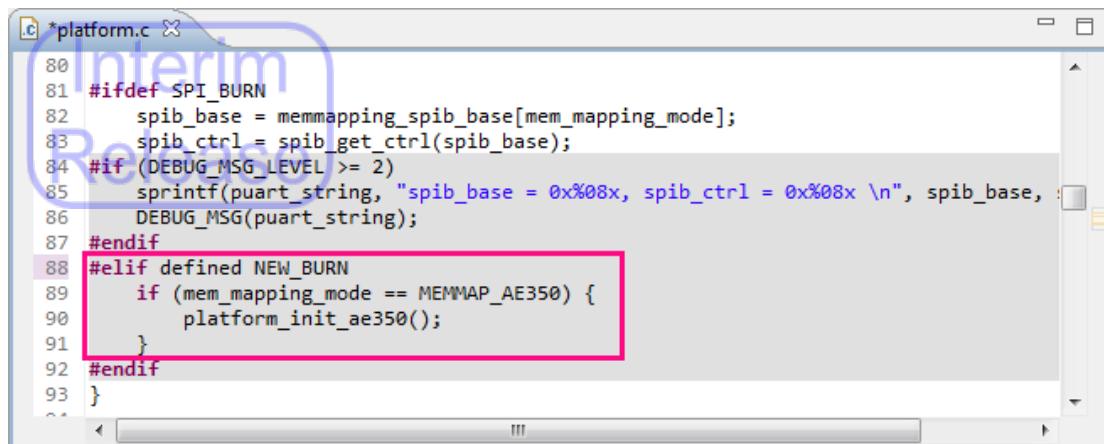
Also note that the value of the maximum size must satisfy the following constraint:

`MAX_SIZE >= total size of (target_burn binary + data + bss + stack)`

Where

1. The text and rodata sections are used to account for the size of the target\_burn binary.
2. The data section contains the command array, whose size is defined by the `CMD_SIZE` macro in `config.h`.
3. The size of the stack should be greater than `0x180` (i.e., 384 bytes).

**Step 4** Open `platform.c` under the project. Edit the file to add functions for initialization of your flash controller and press “Ctrl + C” to save the changes.

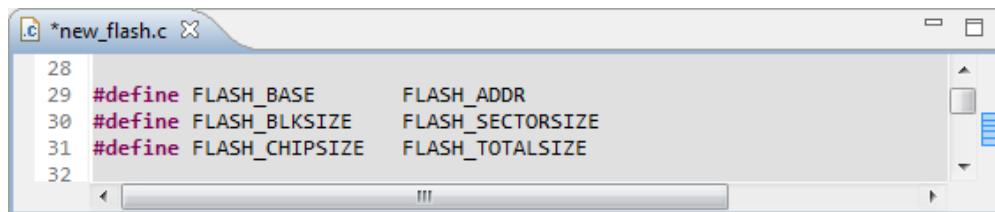


```

80
81 #ifdef SPI_BURN
82     spib_base = memmapping_spib_base[mem_mapping_mode];
83     spib_ctrl = spib_get_ctrl(spib_base);
84 #if (DEBUG_MSG_LEVEL >= 2)
85     sprintf(puart_string, "spib_base = 0x%08x, spib_ctrl = 0x%08x \n", spib_base, spib_ctrl);
86     DEBUG_MSG(puart_string);
87 #endif
88 #elif defined NEW_BURN
89     if (mem_mapping_mode == MEMMAP_AE350) {
90         platform_init_ae350();
91     }
92 #endif
93 }

```

**Step 5** Follow Section 2.1.2.2 to create a source file named by your flash under the target\_burn project. Edit the file to add macros that point to your flash settings defined in `config.h` (see Step 2) as follows:



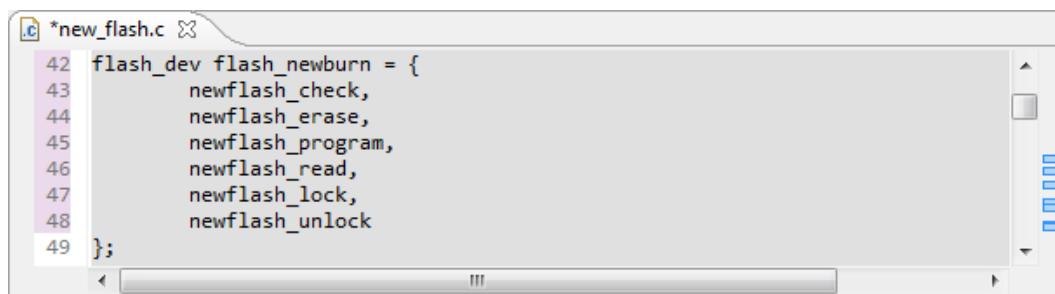
```

28
29 #define FLASH_BASE           FLASH_ADDR
30 #define FLASH_BLKSIZE        FLASH_SECTORSIZE
31 #define FLASH_CHIPSIZE       FLASH_TOTALSIZE
32

```

Next, write functions to check, erase, program, read, lock and unlock your flash. To define these functions necessary for flash programming, you can reference `spi_flash-MXIC.c` under the target\_burn project and look up its `TODO` comments for detailed instructions.

Then, declare these functions inside the `flash_dev` struct.



```

42 flash_dev flash_newburn = {
43     newflash_check,
44     newflash_erase,
45     newflash_program,
46     newflash_read,
47     newflash_lock,
48     newflash_unlock
49 };

```

**Step 6** Open `main.c` under the project. Edit the file to include the `flash_dev` struct for your flash so that functions and macros defined in Step 5 for flash programming can be called.

```
12 extern void nds_ebreak(void);
13 extern void uart_init(unsigned int baudrate);
14 #ifdef SPI_BURN
15 extern flash_dev flash_MXIC;
16 flash_dev *nds_flash_dev = (flash_dev *)&flash_MXIC;
17 #elif defined NEW_BURN
18 extern flash_dev flash_newburn;
19 flash_dev *nds_flash_dev = (flash_dev *)&flash_newburn;
20 #endif
21 unsigned char *current_command_index;
```

This file comes with functions to debug the target\_burn program through single stepping and designates an index for each function as follows:



```
*main.c x
5 #define STEP_LOCK 1
6 #define STEP_EXIT 2
7 #define STEP_UNLOCK 3
8 #define STEP_TX 4
9 #define STEP_INIT 6
10 #define STEP_ERASE 8
```

Reference these functions and their indexes to specify verification commands in the command array(`cmd_arr[CMD_SIZE]`). Or, simply use existing commands in the array to debug the program directly.

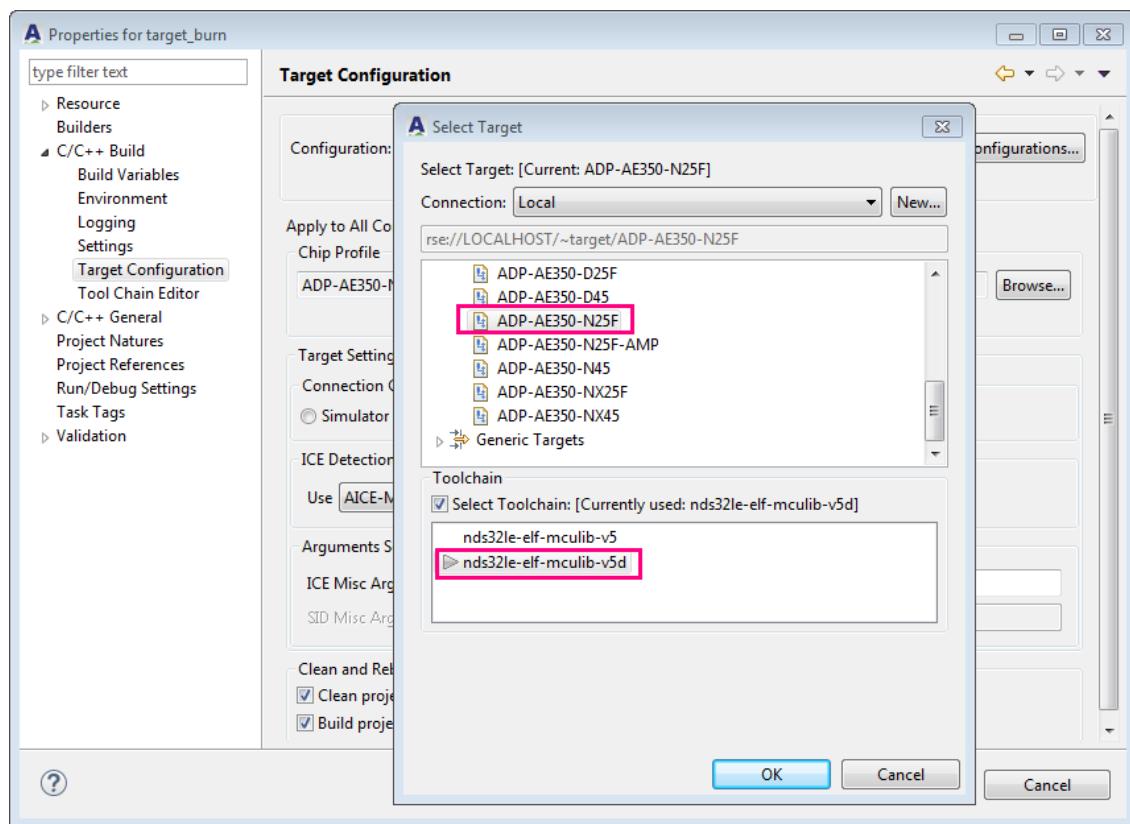
The following explains what commands in the given array are specified:

2. The following sequence { ... 8, 0, 0, 1, 0 ...} is used to execute the `nds_erase` function and erase the flash from Sector 0 to Sector 1.

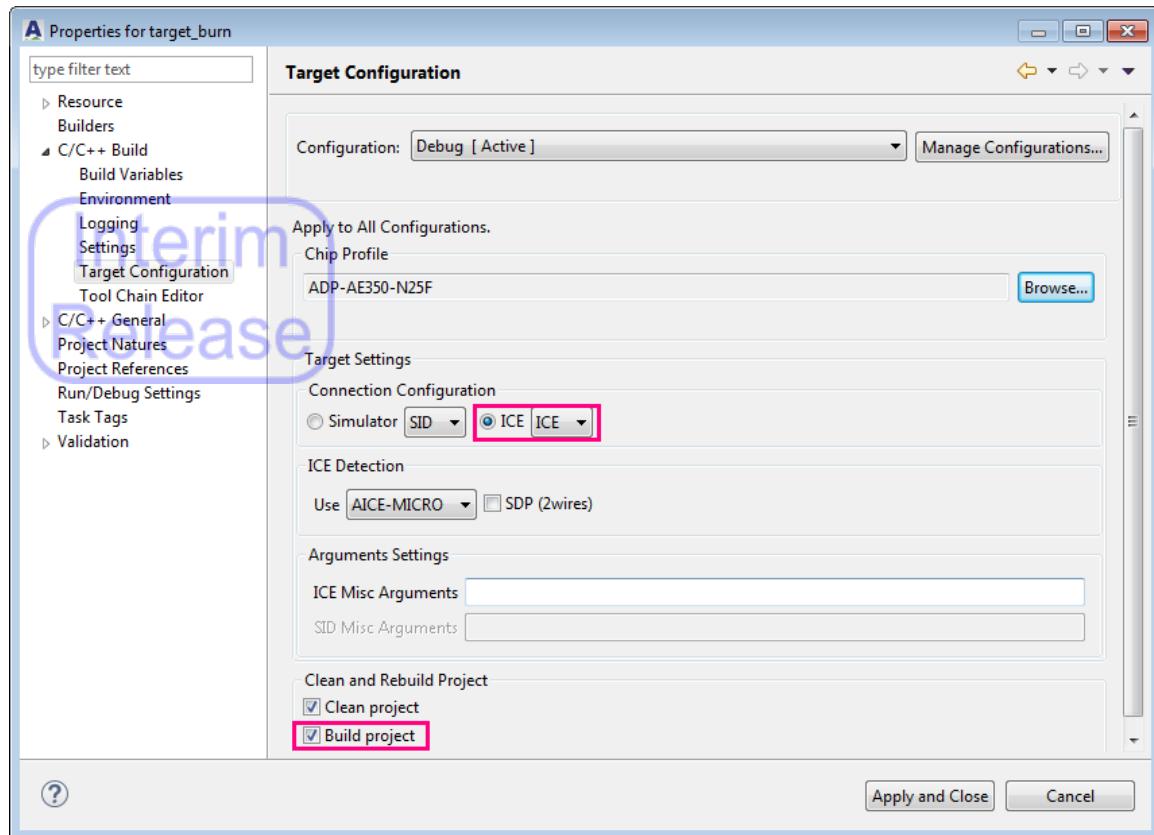
3. The last sequence { ...

4, 0, 0, 0, 0, 32, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1  
9, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32} is used to execute the `nds_tx` function and write data 0~32 to the offset address 0 of the flash.

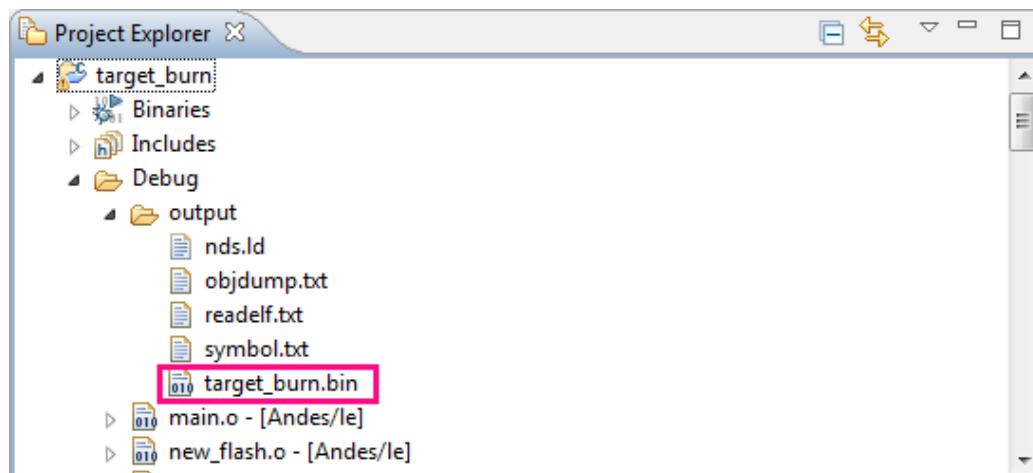
**Step 7** Right-click the project folder and select “Target Configuration” from the pull-down menu. In the invoked dialog, click “Browse...” in the **Chip Profile** section to specify a chip profile and a toolchain that is compatible with your target system.



**Step 8** In the **Properties** dialog, specify “ICE” as the connection type and select the “Build project” option. Then, click “OK” to commence the build process.



**Step 9** The binary of the target application, `target_burn.bin`, is built under `target_burn\Debug\output`. It is the program to be loaded to a V5 target system for flash programming.

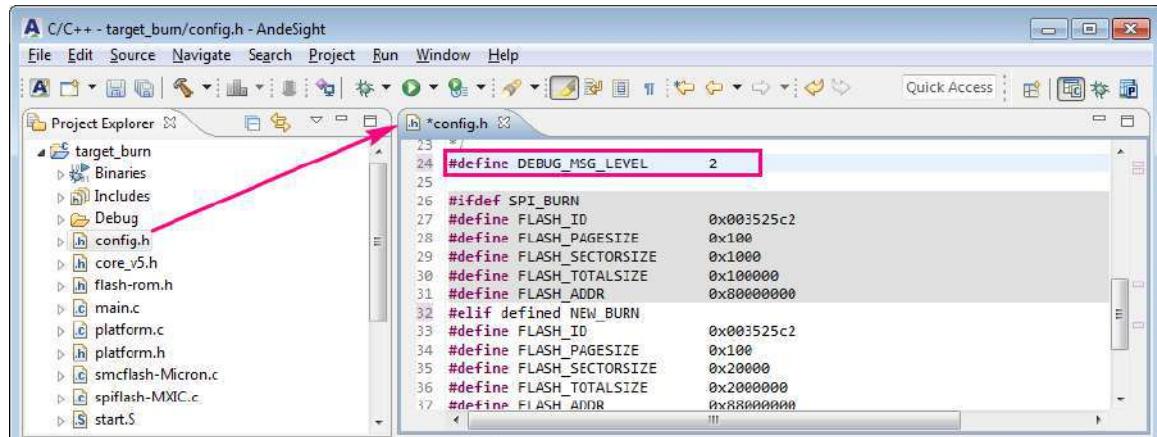


**Step 10** If you encounter problems programming flash with the `target_burn` application, follow below to debug the application using functions in the command array of `main.c` (See Step 6 for details about the

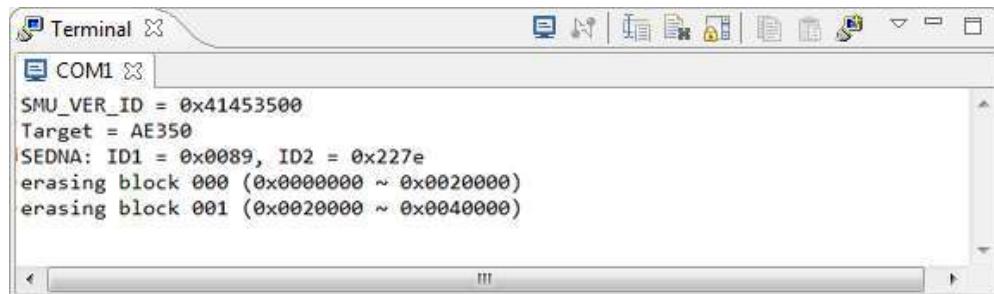
command array):

- Follow Section 2.2.2.2 and Section 2.2.3 to set up your target system and build the terminal connection with the system.

- In Project Explorer, open `config.h` under the `target_burn` project and change the value for the `DEBUG_MSG_LEVEL` macro to 2. This is to enable the display of error and debug messages.

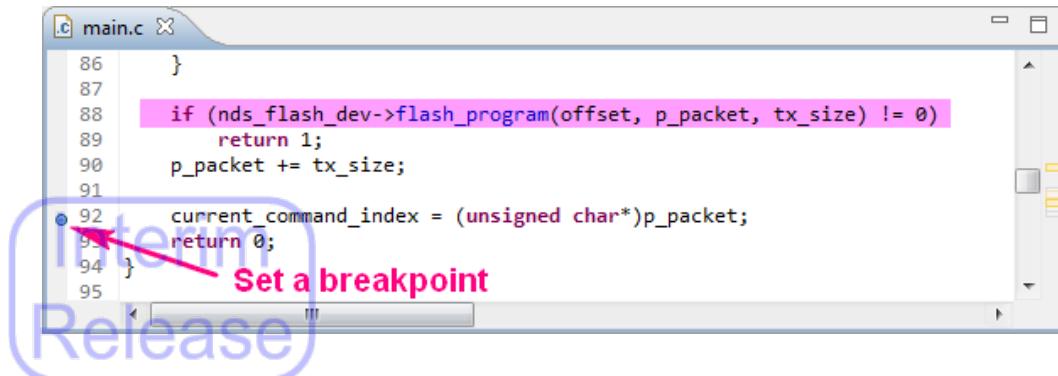


- Follow Section 2.4.3 to commence an Application Program debugging for this project. After the debug session is launched, click (Resume), (Step Into) or (Step Over) on the toolbar of the Debug view and observe the debugging process in the Terminal view.

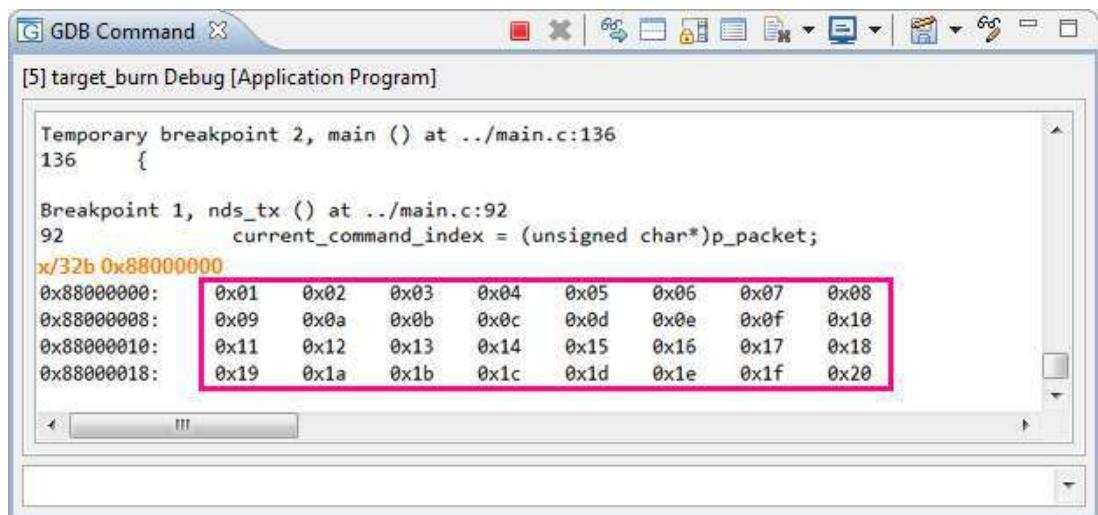


### Step 11 Verify if the target\_burn application programs the flash properly:

- Open `main.c` in the editor and set a breakpoint somewhere after the line "`nds_flash_dev->flash_program(offset, p_packet, tx_size);`" in the `nds_tx` function.



2. Resume the program until it hits the breakpoint.
3. In the **GDB Command** view, issue a GDB command to read the memory address to which the data is written. Examine the data read from the address and see if it matches the data to be programmed in the command array of `main.c`. For example, if you use the given command array that specifies 1~32 as the data to be written, the data read from the flash offset address should be as follows:

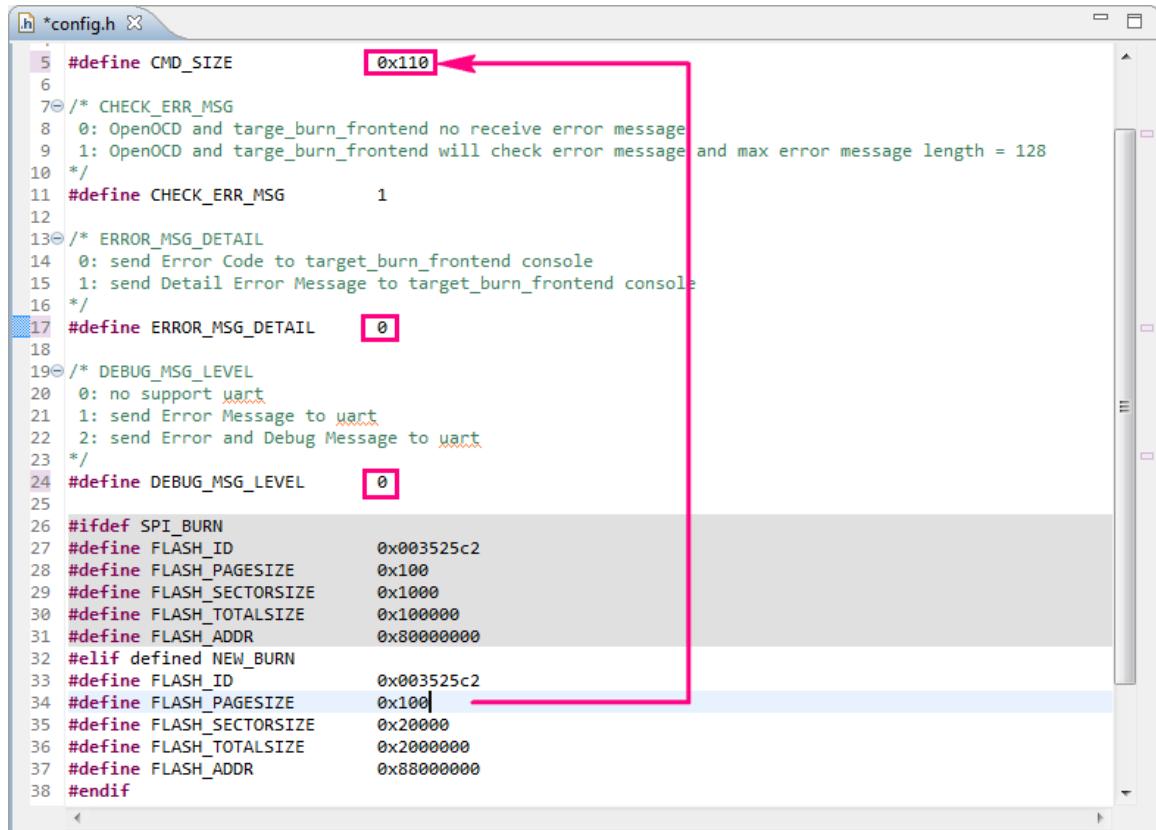


## NOTE

1. Albeit with a lower efficiency, programming on a V5 target with limited RAM is possible using a `target_burn` binary with a reduced size of 4KB. To build such a `target_burn` binary, you will need to make the following changes on the building procedure:

**Step 1** For `config.h` under the `target_burn` project, change the value of

**ERROR\_MSG\_DETAIL** and of **DEBUG\_MSG\_LEVEL** to 0 so that the application will output only error code for each error that occurs and print no debug message. Also, specify a minimum command array size in the flash by modifying **CMD\_SIZE** to a value no less than the flash page size plus 8 bytes. In this example, **CMD\_SIZE** is set to **0x110** as the page size of the flash is defined as **0x100**. If the RAM on your target still has spare space, you may specify a larger value to **CMD\_SIZE** to improve the burning speed.

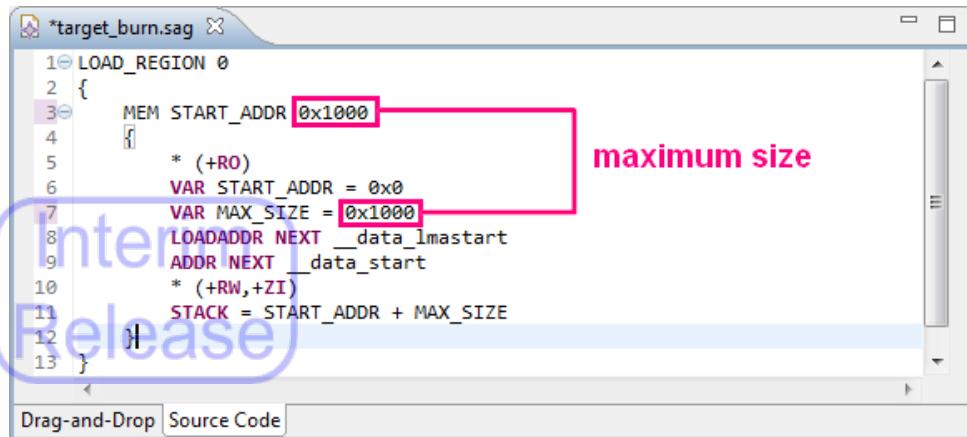


```

1 *config.h x
2
3 5 #define CMD_SIZE      0x110
4
5 /* CHECK_ERR_MSG
6 0: OpenOCD and targe_burn_frontend no receive error message
7 1: OpenOCD and targe_burn_frontend will check error message and max error message length = 128
8 */
9
10 #define CHECK_ERR_MSG     1
11
12
13 /* ERROR_MSG_DETAIL
14 0: send Error Code to target_burn_frontend console
15 1: send Detail Error Message to target_burn_frontend console
16 */
17 #define ERROR_MSG_DETAIL   0
18
19 /* DEBUG_MSG_LEVEL
20 0: no support uart
21 1: send Error Message to uart
22 2: send Error and Debug Message to uart
23 */
24 #define DEBUG_MSG_LEVEL    0
25
26 #ifdef SPI_BURN
27 #define FLASH_ID          0x003525c2
28 #define FLASH_PAGESIZE    0x100
29 #define FLASH_SECTORSIZE  0x1000
30 #define FLASH_TOTALSIZE   0x100000
31 #define FLASH_ADDR        0x80000000
32 #elif defined NEW_BURN
33 #define FLASH_ID          0x003525c2
34 #define FLASH_PAGESIZE    0x100
35 #define FLASH_SECTORSIZE  0x20000
36 #define FLASH_TOTALSIZE   0x2000000
37 #define FLASH_ADDR        0x88000000
38#endif

```

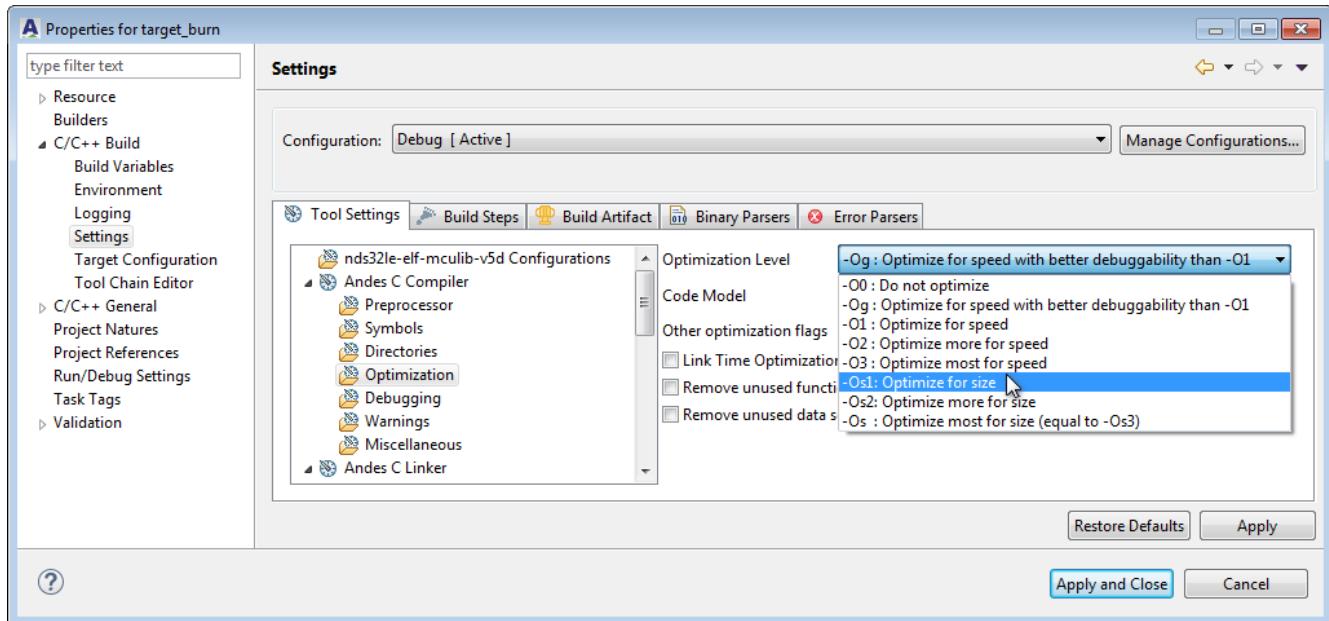
**Step 2** For **target\_burn.sag** under the project, follow Step 3 to specify a least maximum size for flash programming. In this example, the maximum size is set to **0x1000**.



```
*target_burn.sag
1 LOAD_REGION 0
2 {
3     MEM START_ADDR 0x1000
4     * (+RO)
5     VAR START_ADDR = 0x0
6     VAR MAX_SIZE = 0x1000
7     LOADADDR NEXT __data_lmastart
8     ADDR NEXT __data_start
9     * (+RW,+ZI)
10    STACK = START_ADDR + MAX_SIZE
11
12 }
13
```

Drag-and-Drop Source Code

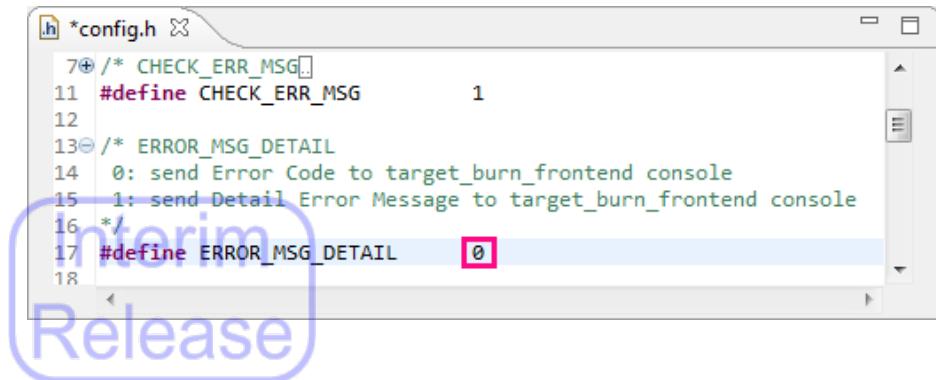
**Step 3** Follow Section 2.1.4.2 to find the optimization level setting of the target\_burn project and change it to **-Os1**.



2. If needed, you can define a custom error code or error message to indicate a specific programming problem. Just proceed as follows:

■ Using a custom error code to indicate a certain error scenario

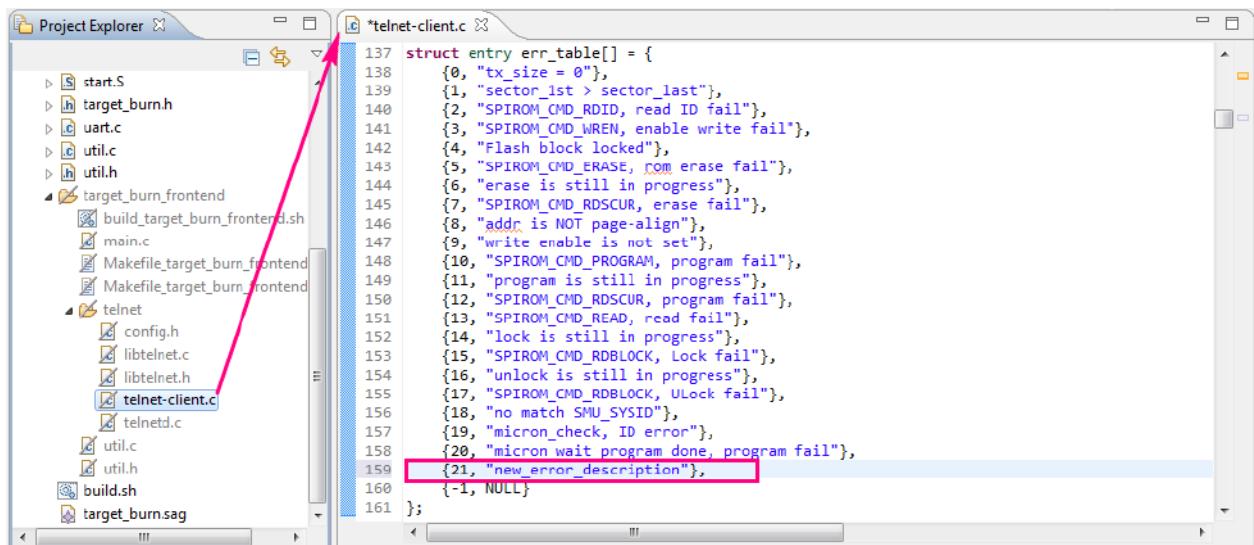
**Step 1** Open `config.h` under the target\_burn project and set the value of `ERROR_MSG_DETAIL` to 0.



**Step 2** Open the source file associated with the programming problem under the target\_burn project and insert the function “**“ERR\_CODE\_EXIT(CUSTOM\_ERROR\_CODE)**” as below.

```
42 #if (ERROR_MSG_DETAIL == 0)
43     ERR_CODE_EXIT(21);
44 #endif
```

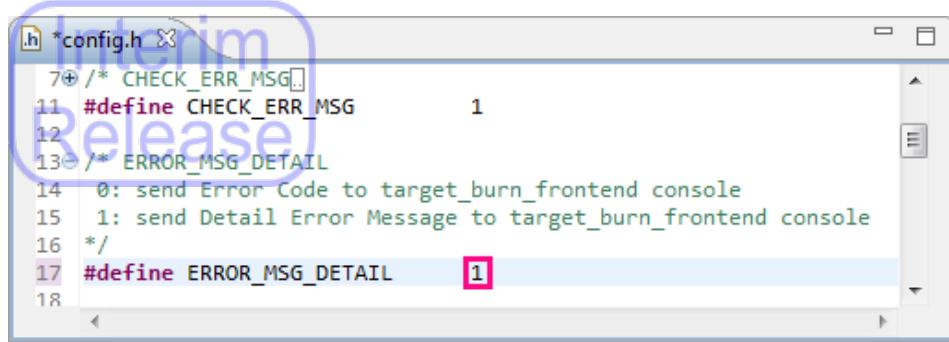
**Step 3** Open **telnet-client.c** under **TARGET\_BURN\_PROJECT\target\_burn\_frontend\telnet** and add your custom error code and corresponding error description to the struct “**entry\_err\_table**”.



**Step 4** Build the target\_burn binary as well as the target\_burn\_frontend executable to compile the newly-defined error code.

■ Using a custom error message to indicate a certain error scenario

**Step 1** Open `config.h` under the `target_burn` project and set the value of `ERROR_MSG_DETAIL` to 1.



```
7 /* CHECK_ERR_MSG */
11 #define CHECK_ERR_MSG      1
12
13 /* ERROR_MSG_DETAIL
14  0: send Error Code to target_burn_frontend console
15  1: send Detail Error Message to target_burn_frontend console
16 */
17 #define ERROR_MSG_DETAIL 1
18
```

**Step 2** Open the source file associated with the programming problem under the `target_burn` project and insert the macro “`ERR_EXIT(CUSTOM_ERROR_DESCRIPTION)`” as below.

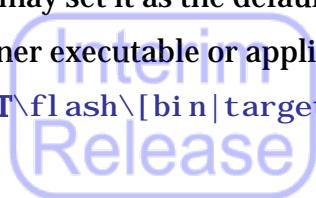
```
42 #if (ERROR_MSG_DETAIL > 0)
43     ERR_EXIT("new_error_description\n");
44 #endif
```

**Step 3** Build the `target_burn` binary to compile the newly-defined error message.

### 2.3.3. Changing the flash burner or target\_burn application in your chip profile

After following Section 2.3.1.1, 2.3.2.1 and 2.3.2.2 to generate a new flash burner or target\_burn application, you may set it as the default one for programming your projects. This requires that you save the burner executable or application binary to

**ANDESI GHT\_ROOT\flash\[bin|target\_bin]** first and specify it in the chip profile of your project.



You may modify the flash driver settings of a chip profile directly in the targetboard.properties file or via the chip profile editor in AndeSight. For details on accessing targetboard.properties files or the chip profile editor, please refer to Sections 2.2.1.1 and 2.2.1.2. The following figures show the flash driver settings in a targetboard.properties file and chip profile editor.

```
.....  
#==== Flash Burner Settings ====  
flash.driver=target_burn_frontend.exe  
flash.driver.custom.ui=N  
flash.target=a350  
flash.start.address=0x0  
flash.controller.address=  
flash.misarguments=  
flash.target.burn.bin=target_SPI_v5_32.bin  
...
```

Figure 26. Flash burner settings in a targetboard.properties file

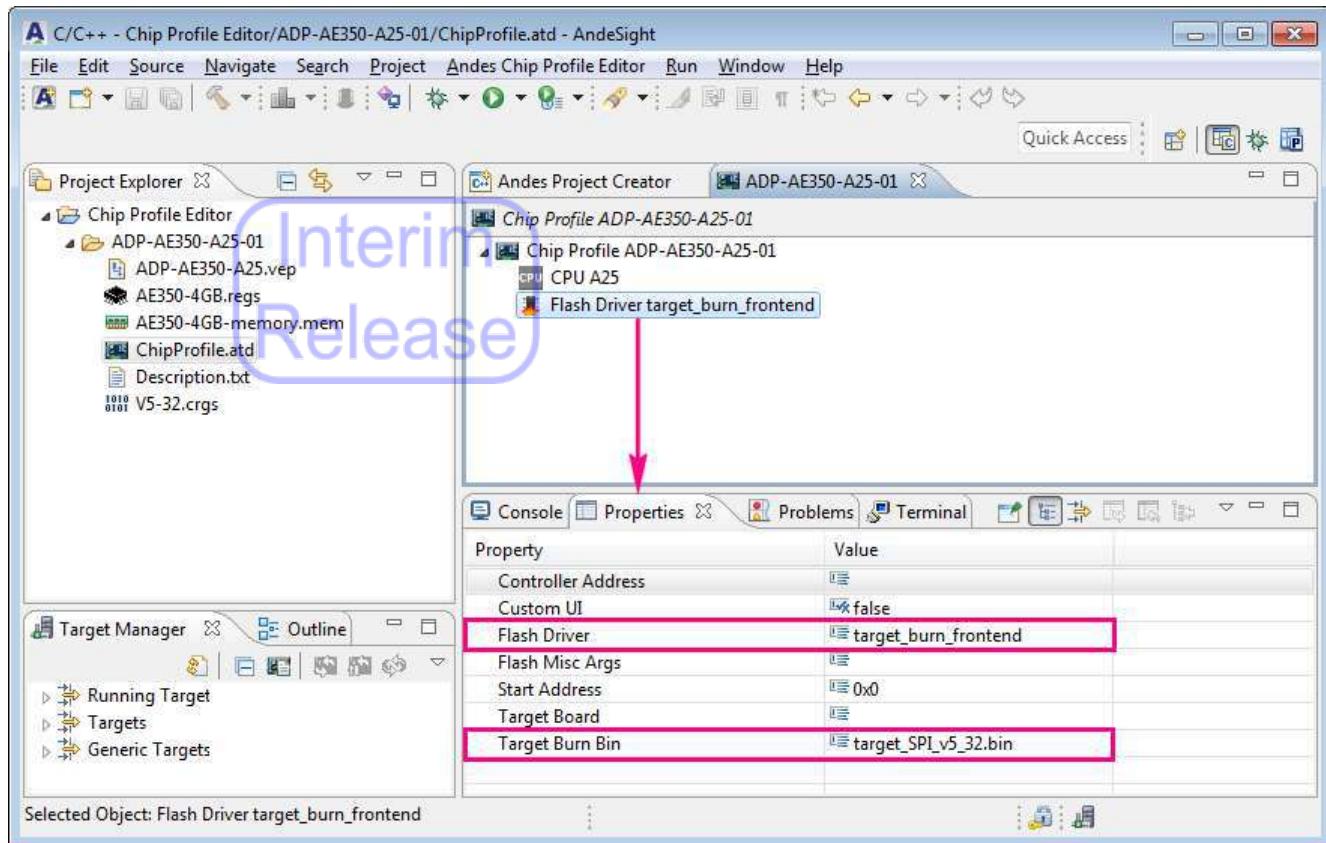
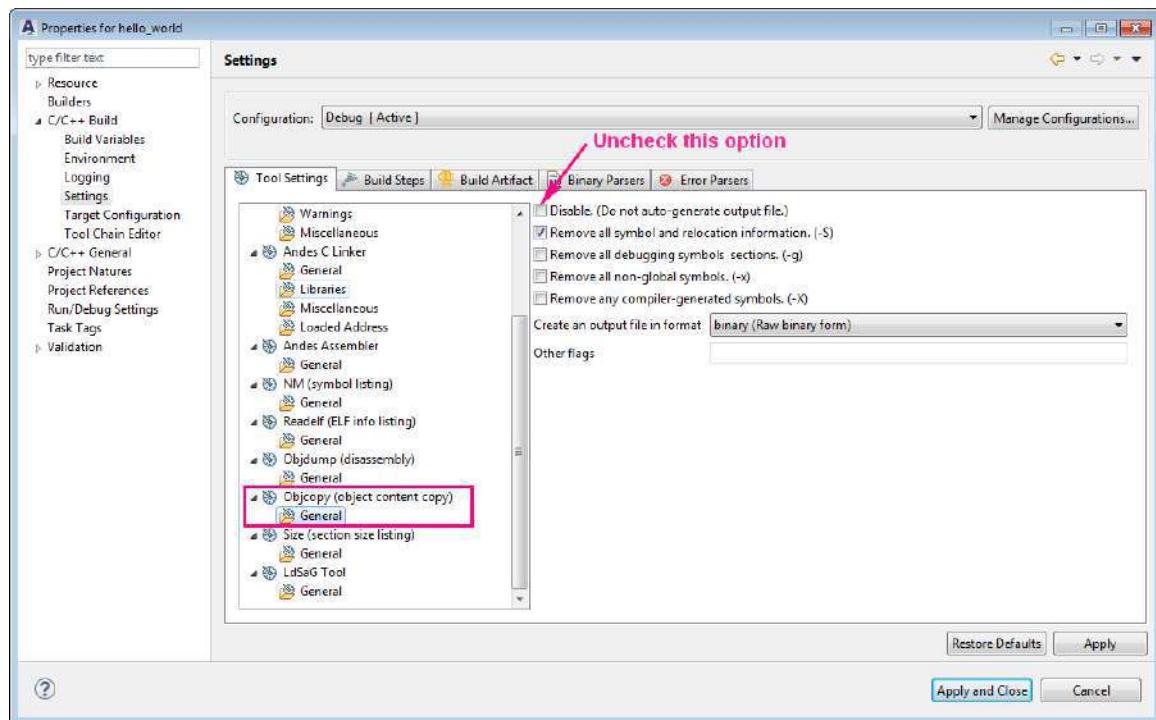


Figure 27. Flash burner settings in chip profile editor

### 2.3.4. Flash programming UI in AndeSight

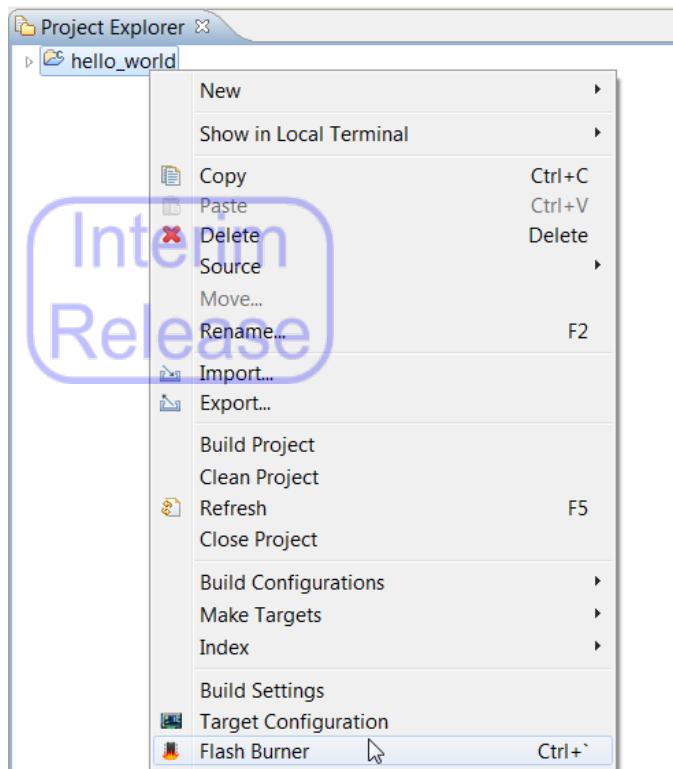
The **Flash Programming Wizard** in AndeSight provides a friendly interface by which to configure settings for in-system programming on flash memory. To perform flash programming using AndeSight, follow the steps below.

- Step 1** Follow Section 2.1 to create and build a project. Before building the project, enable the Andes Objcopy Tool for the generation of image files (binary type).



- Step 2** Follow Section 2.2.2 to set up the target system.

- Step 3** Right-click the project folder and select “Flash Burner” in the pull-down menu to invoke the **Flash Programming Wizard**.



**Step 4** Specify the parameters in the **Flash Programming Wizard**:

Frequently-used parameters for ICE targets and simulator targets are summarized below.

- Programming parameters for ICE targets

Field or option	Usage	Notes
Flash driver	<p>Specify the path of the burner utility. The target_burn_frontend utility is recommended as it can speed up the programming process when used with the target_burn application. Otherwise, select the PAR_burn utility to program parallel flash and the SPI_burn utility to program SPI flash. These burner programs are all placed under</p>	

Field or option	Usage	Notes
	<b>ANDESIGHT_ROOT\flash\bin\.</b>	
Flash Image	Specify the path of the image file to be burned onto flash memory.	
Target Burn	<p>Select this option to perform in-system programming on the target directly. You will need to specify a target_burn program to be loaded to the target system for the programming. The default target_burn program for SPI flash on target systems with an Andes 32-bit V5 CPU core is <b>target_SPI_v5_32.bin</b>, whereas that for SPI flash on systems with an Andes 64-bit V5 CPU core is <b>target_SPI_v5_64.bin</b>. Both programs are placed under <b>ANDESIGHT_ROOT\flash\target_bin\.</b></p>	<p>This field is specified only when the target_burn_frontend utility is in use. And, when this field is entered, remote flash programming is not supported.</p>
Target management	Select this option if the target designated during project creation is your desired target.	The option is mandatory when a target_burn program is used for programming.
Host	Specify the name or IP address of the remote host.	If the target designated for your project is not your desired target and you want to connect your target to a remote host, then uncheck the “Target management” option and specify these two options.
Burner port	Specify a socket port to which the burner will connect.	

Field or option	Usage	Notes
Programming Start Address	Specify the start address to be programmed. This field may not be left empty, and the default value for ADP-XC7K is <b>0x0</b> .	
Flash controller address	Specify the address of the flash memory to be programmed.	
Target board	Specify the type of board to map the ROM start address to <b>0x0</b> . This field is optional.	

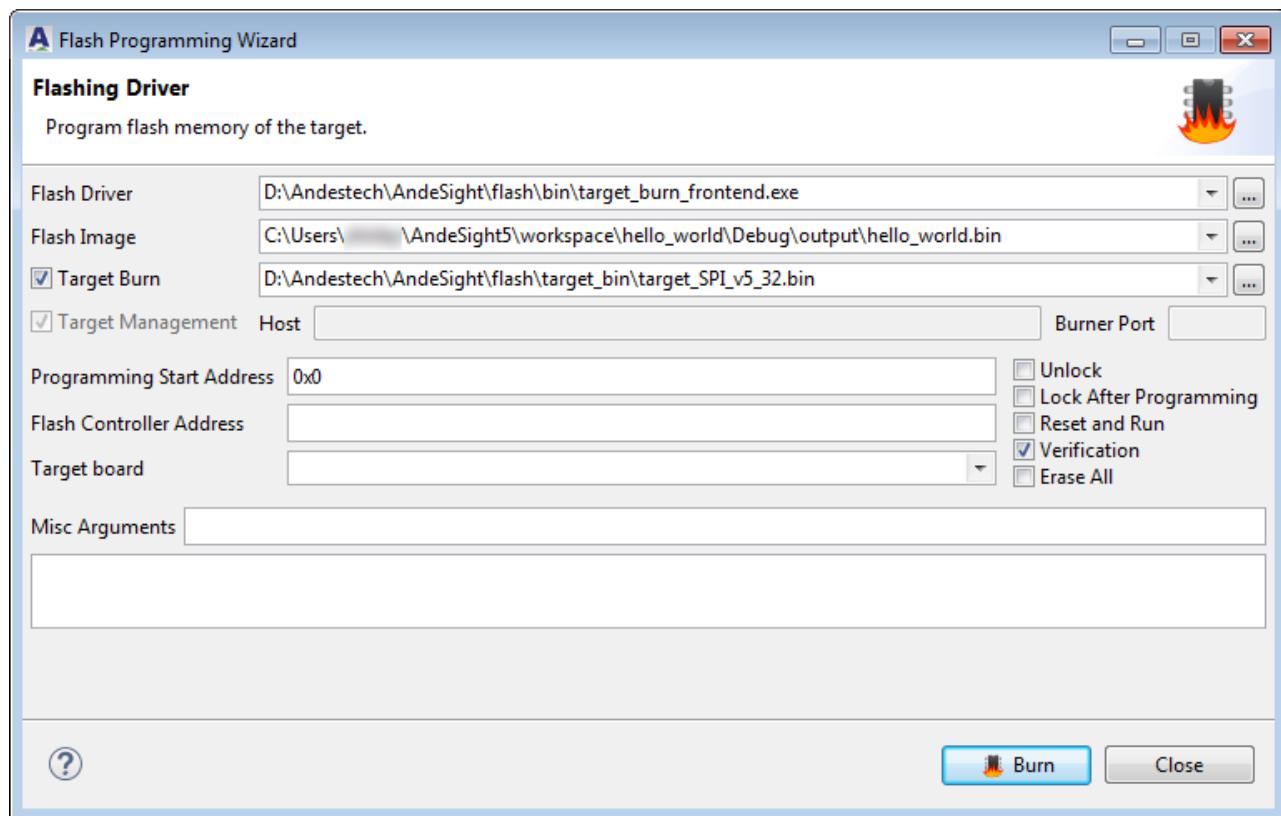


Figure 28. Flash Programming Wizard for projects using target\_burn on V5 ICE targets

- Programming parameters for simulator targets

Section	Field or option	Usage	Notes
Flash image	Location	Specify the path of the image file to be burned onto flash memory.	
	Programming Start Address	Specify the start address to be programmed. This field may not be left empty.	
Connection setting	Target management	Select this option if the target designated during project creation is your desired target.	The option is selected by default.
	Host name or IP address	Specify the name or IP address of the remote host on which the simulator will run.	If the target designated for your project is not your desired target and you want to connect your target to a remote host, then uncheck the “Target management” option and specify these options.
	Port number	Specify the port number to which the simulator will connect.	
	GDB Executable	Specify the GDB executable under your toolchain folder.	For Andes C/C++ projects, the GDB executable and initiation script are auto-detected.
	GDB Command File	Specify a script to perform the necessary initiation for your target.	

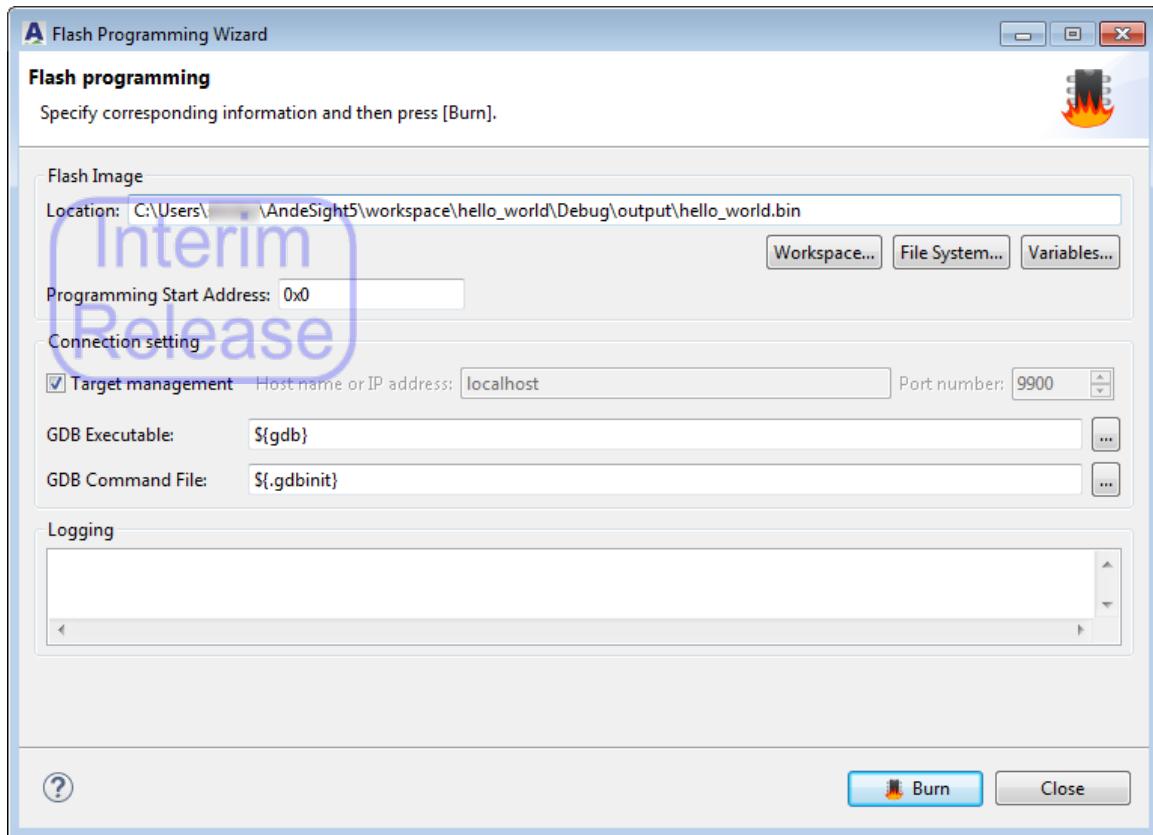


Figure 29. Flash Programming Wizard for projects on simulator targets

**Step 5** Click “Burn” to commence flash memory programming.

### 2.3.5. Replacing Andes flash burner(s) and AndeSight flash programming UI

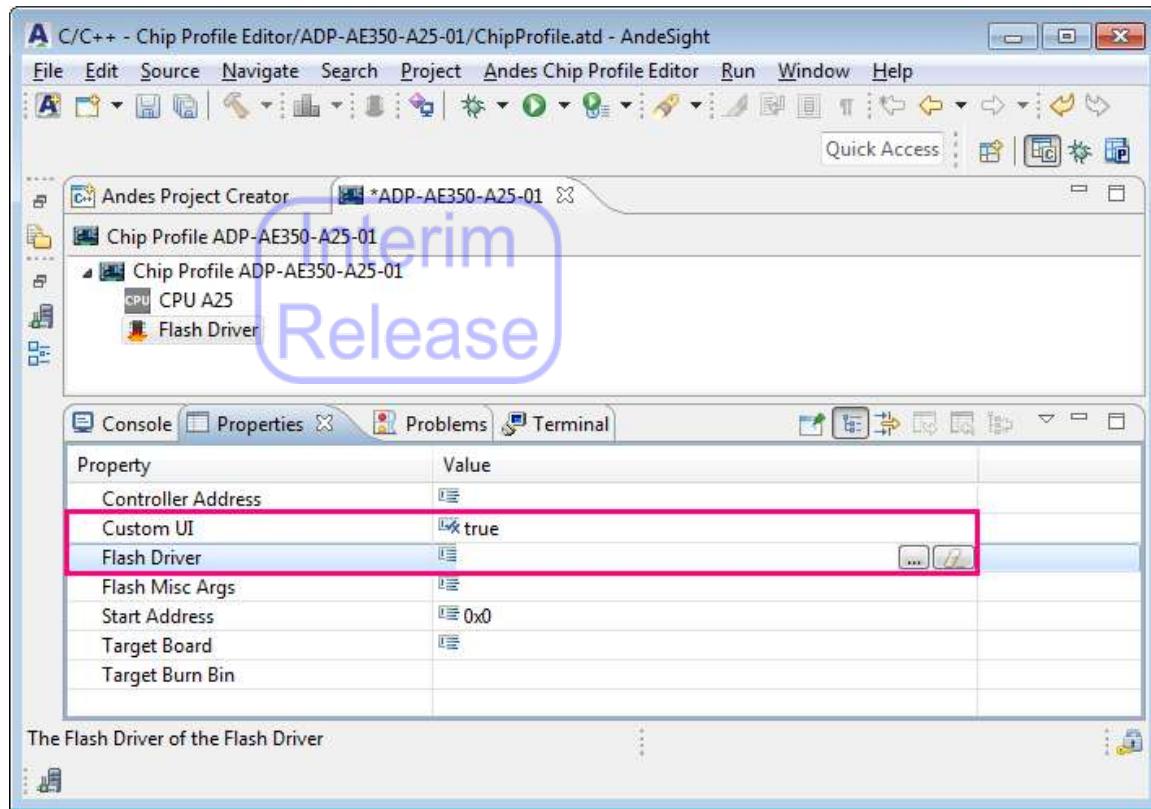
In case that you prefer your own flash burner and/or programming wizard to what are inherent in AndeSight, you may specify them in the chip profile for your projects. This enables you to use your own burner utility and programming wizard for flash programming.

To make the specifications in the chip profile, edit the corresponding targetboard.properties file directly or use the chip profile editor in AndeSight. For the targetboard.properties file, look for the “Flash Burner Settings” section and set the attribute “`flash.driver`” to the absolute path to your burner. To replace the **Flash Programming Wizard** in AndeSight with your designated programming interface, make sure you also assign “`Y`” to the attribute

“`flash.driver.custom.ui`”. Once you complete the specifications, relaunch AndeSight for the changes to take effect. For details on targetboard.properties files, please refer to Section 2.2.1.1.

```
...
===== Flash Burner Settings ====
flash.driver=ABSOLUTE_PATH_TO_YOUR_BURNER
flash.driver.custom.ui=Y
flash.target=
flash.start.address=
flash.controller.address=
flash.mscarguments=
flash.target.burn.bin=
...
...
```

Likewise, for specifications via the chip profile editor, set the property “Flash Driver” to the absolute path to your flash burner and the property “Custom UI” to “`true`” if you would like to replace the programming wizard in AndeSight. Once the specifications are completed, make sure you install the modified chip profile to Target Management. For details on specifying target settings via the chip profile editor, please refer to Section 2.2.1.2.



## NOTE

1. The following are the commands that AndeSight issues in the background to execute your burner:

```
$ YOUR_BURNER --image *.bin --p BURNER_PORT
```

The argument “`--port BURNER_PORT`” will always be sent and the other argument “`--image *.bin`” will be sent whenever there is a binary file in your project. Therefore, make sure your burner can handle the two arguments if it will be used to replace Andes flash burner.

2. The following lists other settings related to flash programming in chip profiles. If any of them is specified in targetboard.properties files or chip profile editor, make sure your burner knows how to process the corresponding argument. Otherwise, simply leave them unspecified in the chip profile.

Attributes in targetboard.properties	Properties in Chip Profile Editor	Argument
flash.target	Target Board	
flash.start.address	Start Address	<code>--addr</code>

Attributes in targetboard.properties	Properties in Chip Profile Editor	Argument
flash.controller.address	Controller Address	--base
flash.miscarguments	Flash Misc Args	YOUR_ARGUMENTS
flash.target.burn.bin	Target Burn Bin	-P TELNET_PORT -A TARGET_BURN_BIN

Internal  
Release

## 2.4. Debugging/Running programs

### 2.4.1. Debug perspective

The **Debug** perspective comprises a code editor, the **Target Manager** view, and several debug views, which enable you to control the program execution and conduct diagnoses in each step.

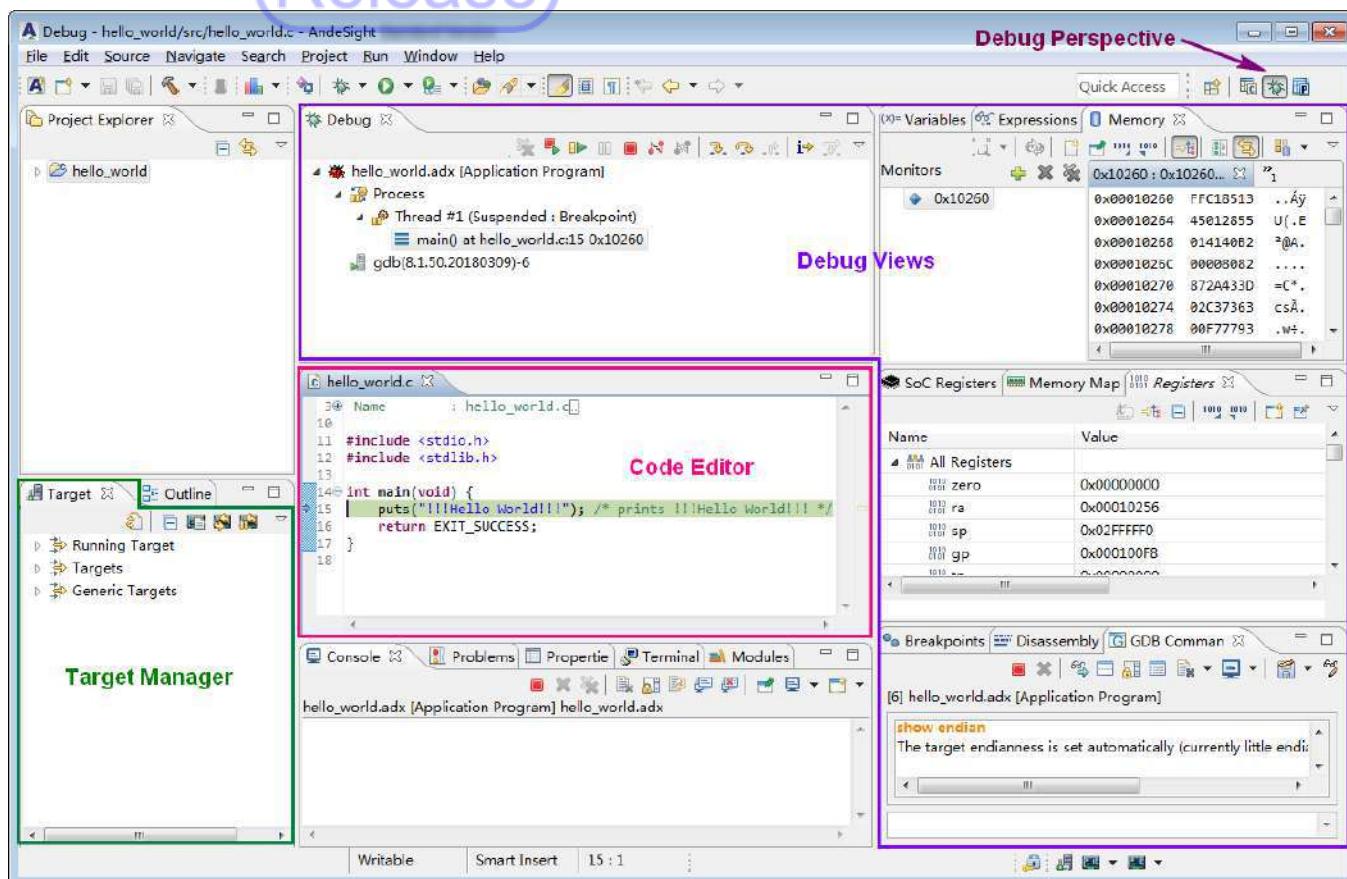
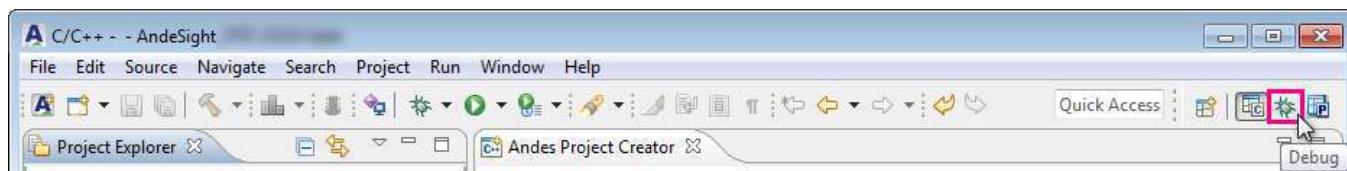


Figure 30. Debug perspective

The **AndeSight IDE** will lead you to the **Debug** perspective when a debug session is launched. From the AndeSight perspective toolbar, you may open the **Debug** perspective manually by clicking  (Debug).

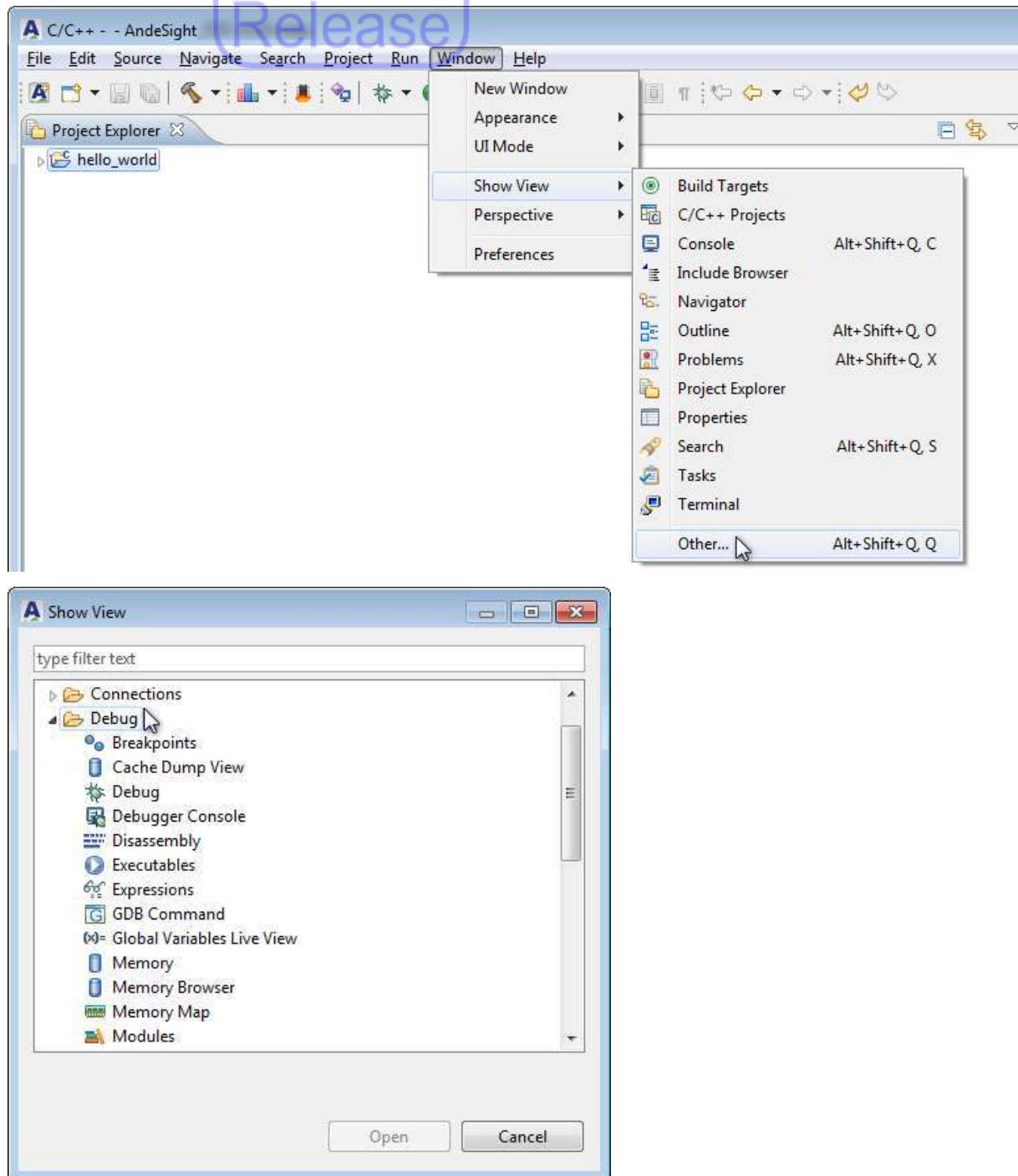


From the AndeSight main menu, you may also evoke the **Debug** perspective by selecting “Window > Perspective > Open Perspective > Debug”.



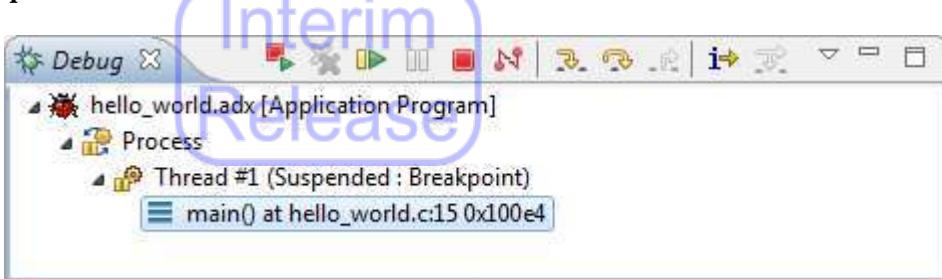
## 2.4.2. AndeSight debug views

Some of the AndeSight debug views are invoked automatically when a debug session is launched. To invoke a specific debug view, go to the AndeSight main menu and select “Window > Show View> Other...”. Then, click “Debug” in the **Show View** dialog to select a desired debug view in the expanded tree.



### 2.4.2.1 Debug view

This view allows you to control program execution. It also displays the execution stack that outlines suspended and active threads for the target undergoing debugging. Each thread is presented as a node in the tree.



#### Toolbar for the Debug view

##### Terminate and relaunch



End the selected debug thread and start it again.

##### Resume



Resume the suspended debug thread.

##### Suspend



Stop the selected debug thread temporarily.

##### Terminate



End the selected debug thread.

##### Disconnect



Detach the debugger from the selected process.

##### Step into



Execute the current statement of the selected debug thread and step into function calls.

##### Step over



Execute the current statement of the selected debug thread and step over function calls.

##### Step return

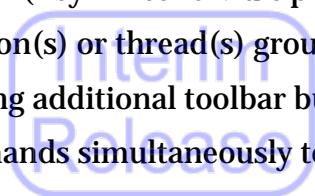


Execute the current statement of the selected debug thread until it returns to its caller.

**Instruction stepping mode** 

Move through assembly instructions in the **Disassembly** view.

For an AMP (Asymmetric Multiprocessing) multi-core project or a project with its configuration(s) or thread(s) grouped into a Core Group configuration (see Section 2.4.11), the following additional toolbar buttons are present in the **Debug** view. They are used to send commands simultaneously to each core within the multicore project or Core Group.

**Group resume** 

Resume each suspended debug thread within an AMP project or Core Group.

**Group suspend** 

Temporarily stop each debug thread within an AMP project or Core Group.

**Group terminate** 

End each debug thread within an AMP project or Core Group.

**Group step into** 

Execute the current statement of each debug thread within an AMP project or Core Group and step into function calls.

**Group step over** 

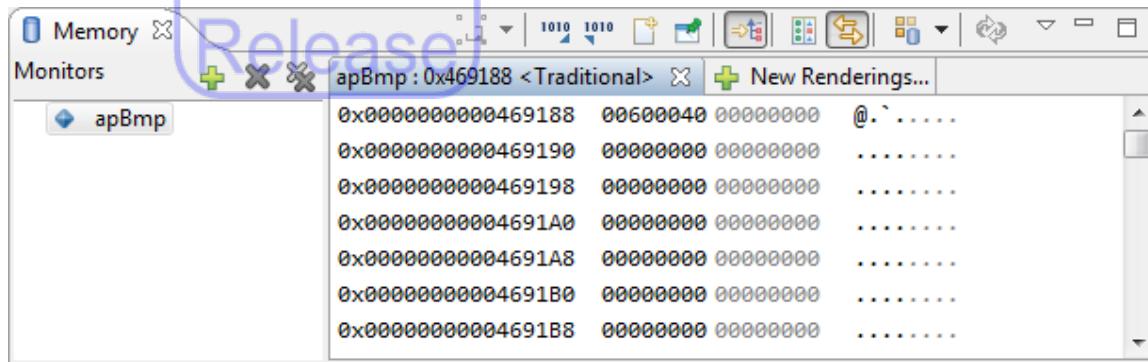
Execute the current statement of each debug thread within an AMP project or Core Group and step over function calls.

**Group step return** 

Execute the current statement of each debug thread within an AMP project or Core Group until it returns to its caller.

#### 2.4.2.2 Memory view

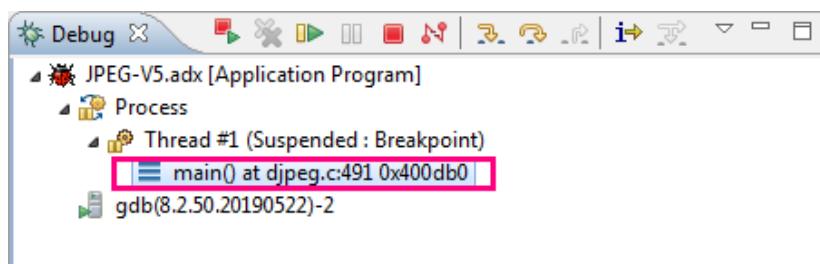
This view allows you to examine and modify your process memory, which is presented as a list of memory monitors. Each memory monitor refers to a section of memory specified by its base address or an expression. The data in each memory monitor can be rendered in several predefined formats.



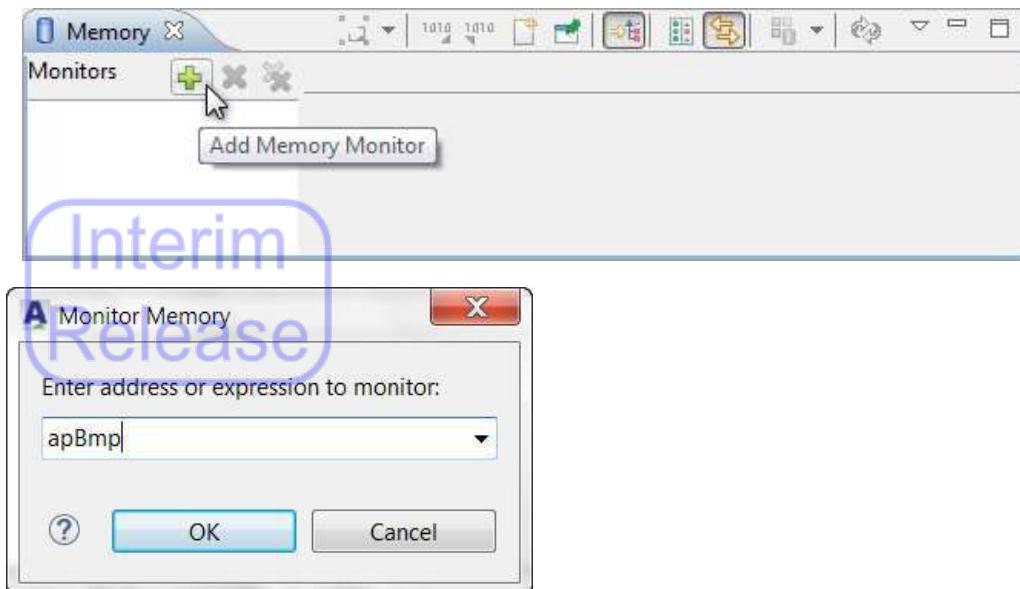
#### Setting a memory monitor

The following procedure is illustrated using the **Memory** view in a debug session of the JPEG-V5 demo.

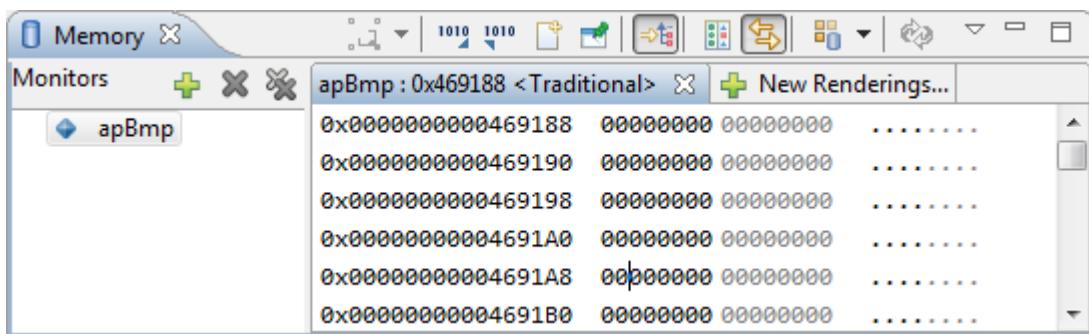
**Step 1** After program execution is suspended, select a desired thread or stack frame in the **Debug** view.



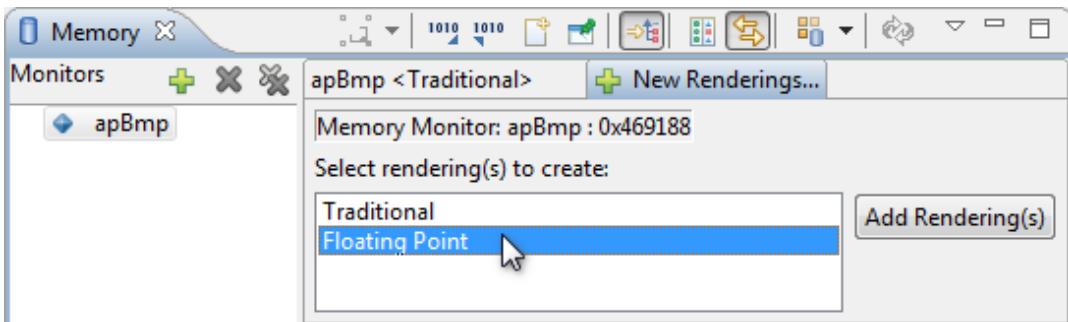
**Step 2** In the **Memory** view, click **+** (Add Memory Monitor) under the monitors pane to invoke the **Monitor Memory** dialog. Then, enter a memory address or expression in the dialog and click “OK.”



**Step 3** A traditional memory rendering is displayed by default.



To create a floating point memory rendering, click on the **New Renderings** tab in the right pane, select “Floating Point” and click the button “Add rendering(s)”.



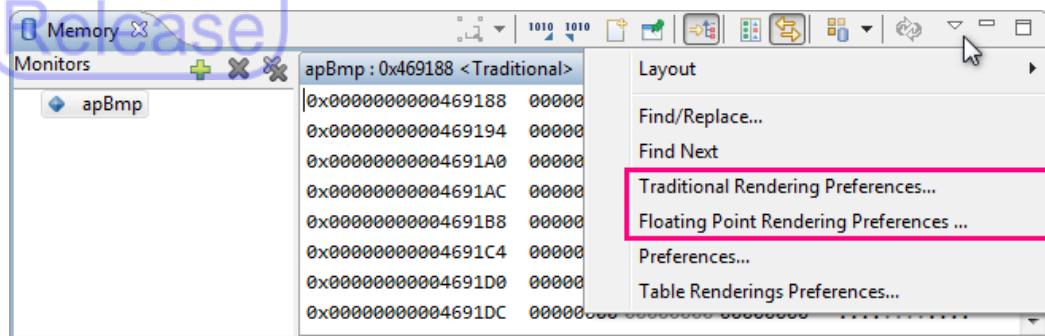

---

#### NOTE

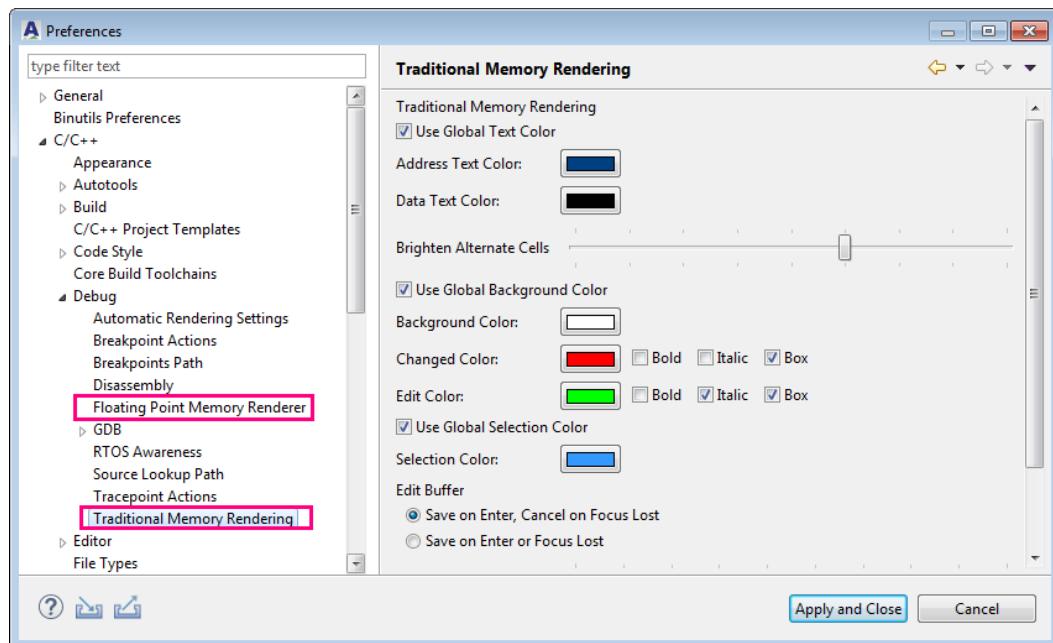
The renderings are default displayed in black texts. You can also

use different text colors to differentiate addresses from data in the renderings. Just follow below:

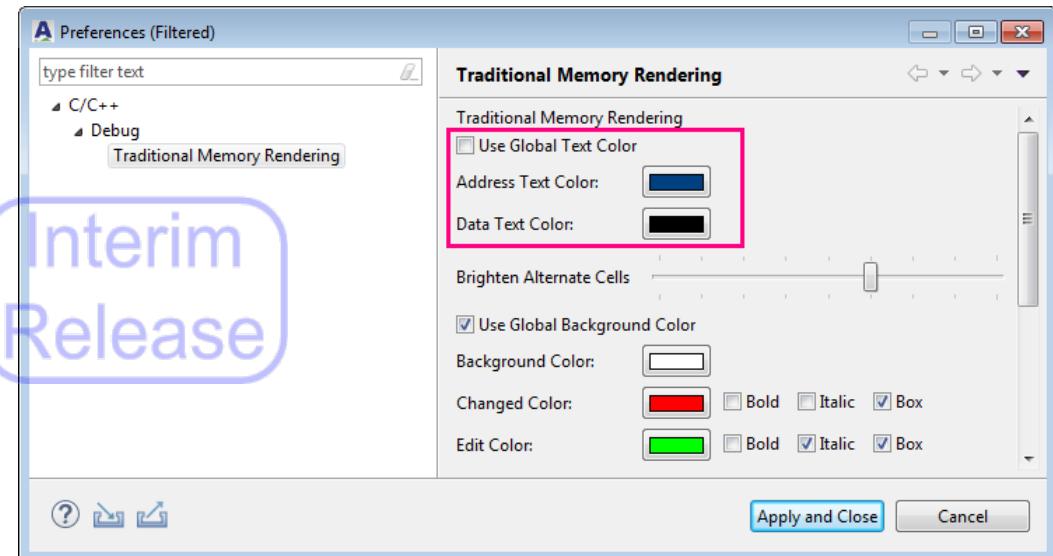
1. Click  (View Menu) on the toolbar and select “Traditional/Floating Point Rendering Preferences” in the pull-down menu to open the preference panel.



The preference panel for the traditional or floating point memory renderings can also be found in the AndeSight Preferences settings. To open it, just select “AndeSight main menu > Window > Preferences > C/C++ > Debug > [Traditional Memory Rendering|Floating Point Memory Renderer]”.

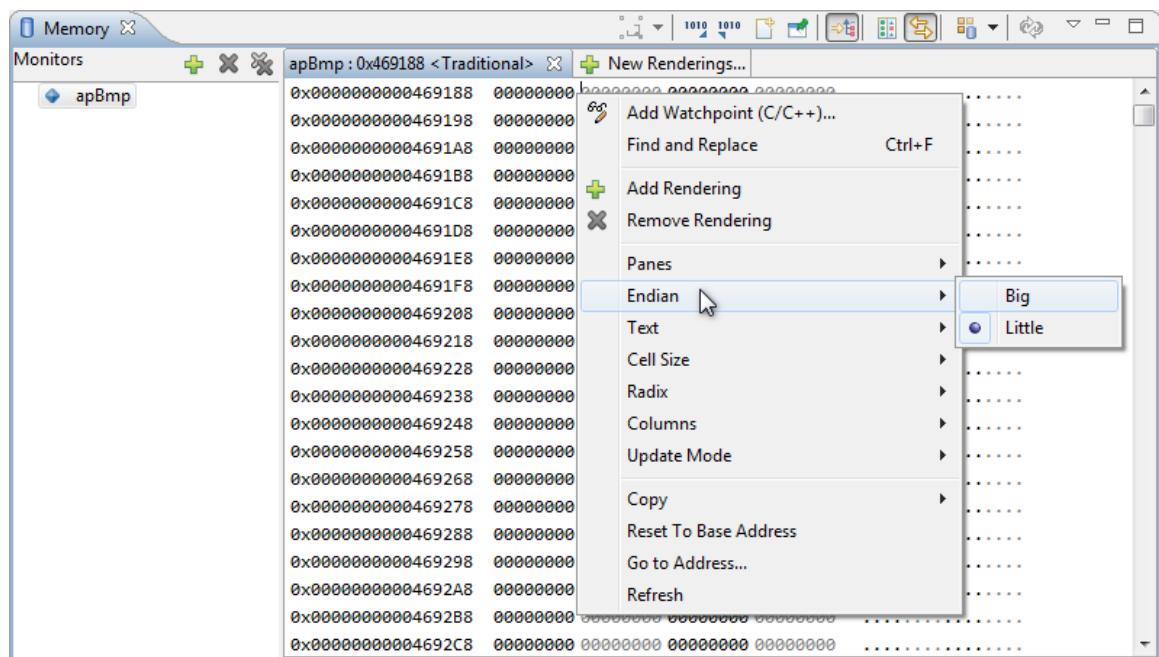


2. Uncheck the option “Use Global Text Color” first and define the respective text color for addresses and data.



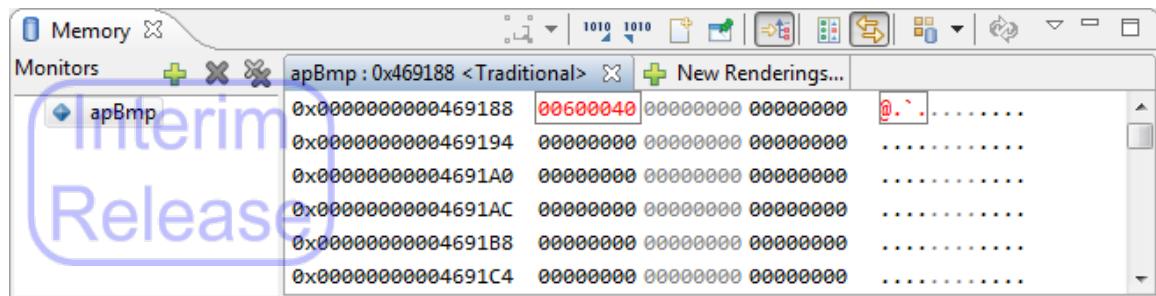
3. Click “Apply and Close” on the dialog to save the changes.

**Step 4** You may change the endianness or addressable size of the displayed data by right-clicking the rendering to invoke a pull-down menu and making your selections.

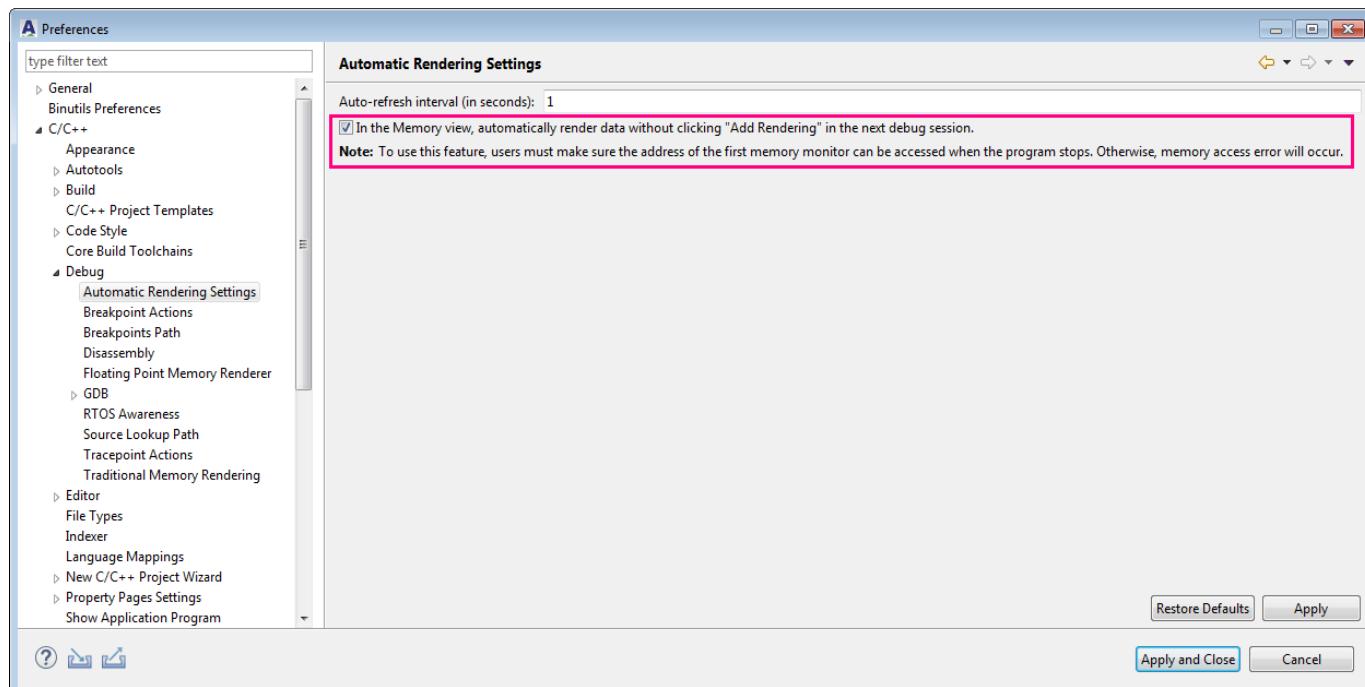


You may also edit the memory values in the cells; however, the program will crash if you enter inappropriate values.

**Step 5** During program execution, the deviation of memory content is marked in red.



**Step 6** To render data automatically for the same memory monitor(s) after re-launching a debug session, specify it in the **Preferences** settings by clicking “AndeSight main menu > Window > Preferences > C/C++ > Debug > Automatic Rendering Settings” and selecting the checkbox that reads, “In the Memory view, automatically render data without clicking “Add Renderings” in the next debug session.” Note that this feature functions only if the address of the first memory monitor can be accessed when the program stops. Otherwise, a memory access error will occur.



### Toolbar for the Memory view

#### Switch source between BUS and CPU



This button enables you to specify whether memory content is provided by the CPU or BUS.

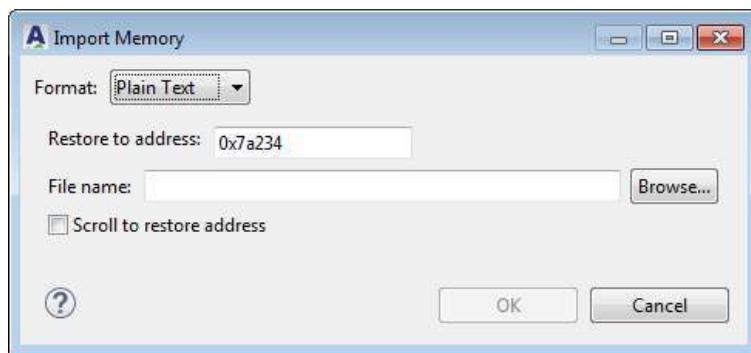
The CPU mode provides content in the system or cache memory. The BUS mode provides content only in system memory.



#### Import memory values



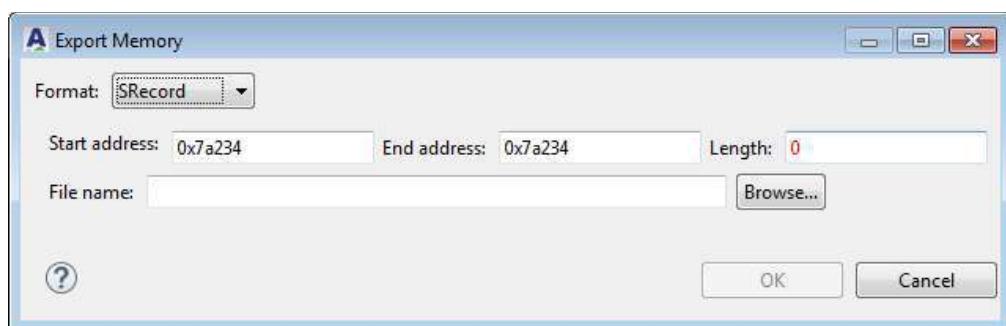
Import a prescribed set of memory values into the **Memory** view.



#### Export memory values



Export the memory values shown in the **Memory** view to files.



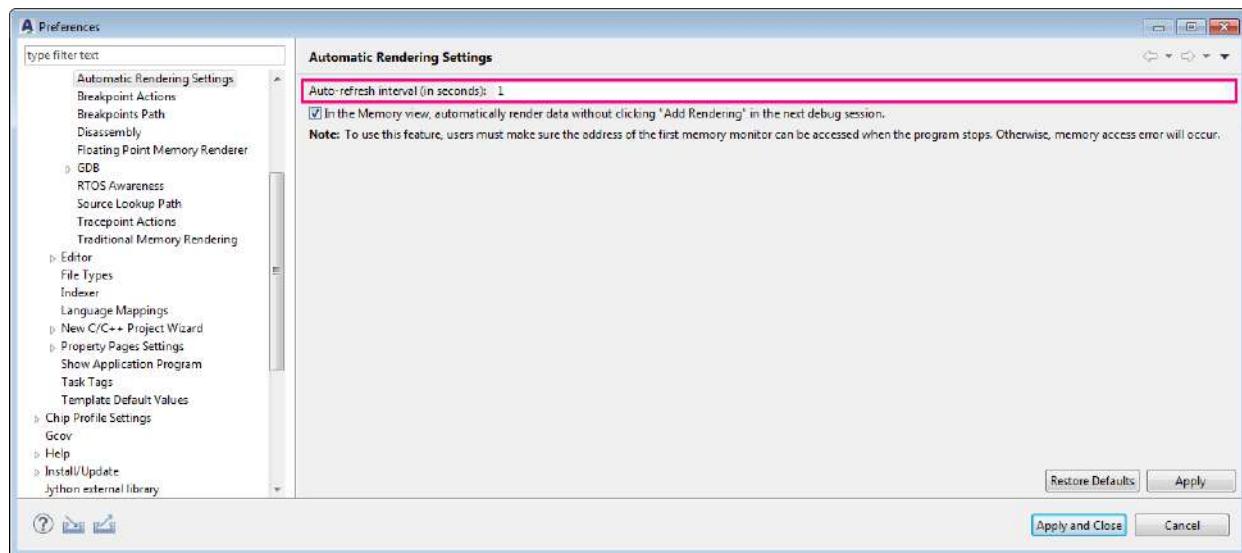
AndeSight supports three formats for importing/exporting memory data: S-record, plain text, and raw binary. Plain text and raw binary are more common. Plain text data contain hexadecimal values and store two hex characters in a byte, whereas raw binary data contain binary values.

## Auto refresh



This button enables periodical refreshing of the memory content; however, it is applicable only to ICE targets. For V5 targets with debug module capable of auto-refresh, you may select this button to enable the auto-refresh feature.

The default refresh interval is 1 second, but you can change it in the **Automatic Rendering Settings** page under the **Preferences** settings, which can be opened as follows: “AndeSight main menu > Window > Preferences > C/C++ > Debug > Automatic Rendering Settings”.



### 2.4.2.3 Memory Browser view

The **Memory Browser** view is an alternative to the **Memory** view for examining and modifying process memory.

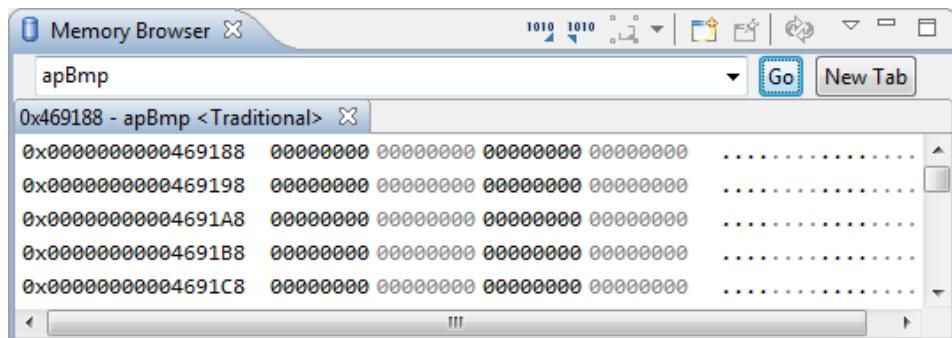
#### Setting a memory monitor

The following procedure is illustrated using the **Memory Browser** view from a debug session of JPEG-V5 demo.

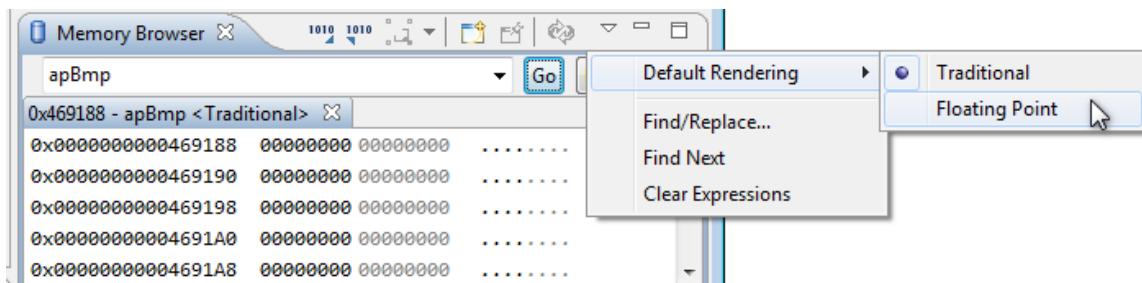
**Step 1** After program execution is suspended, enter the desired address or expression in the text field of **Memory Browser** and click “Go.”

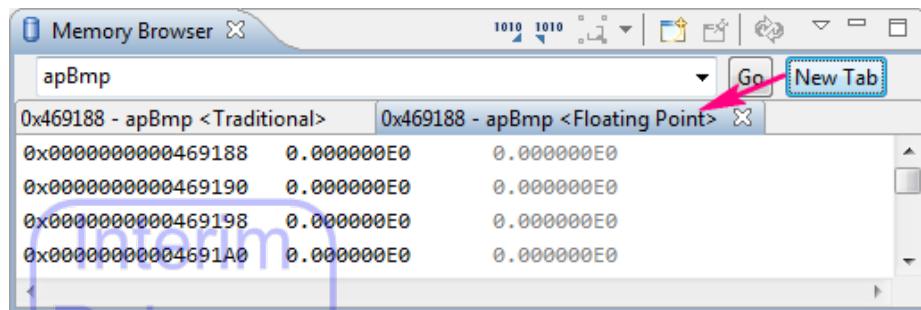


**Step 2** A traditional memory rendering is displayed by default.



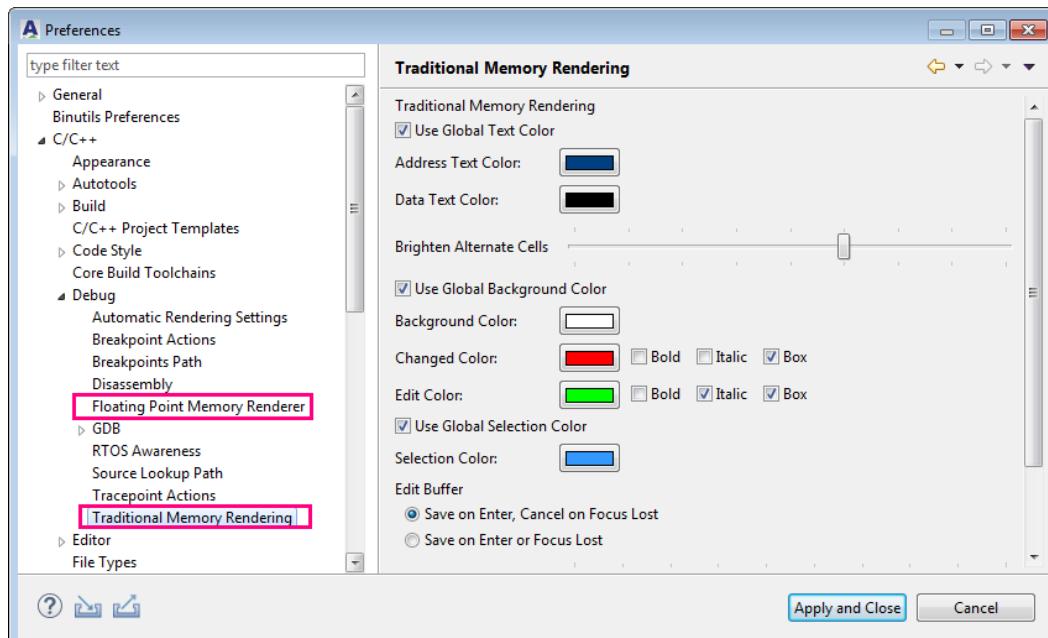
To create a floating point memory rendering, click  on the toolbar, select “Default Rendering > Floating Point” in the pull-down menu, and then click the New Tab button in the view.



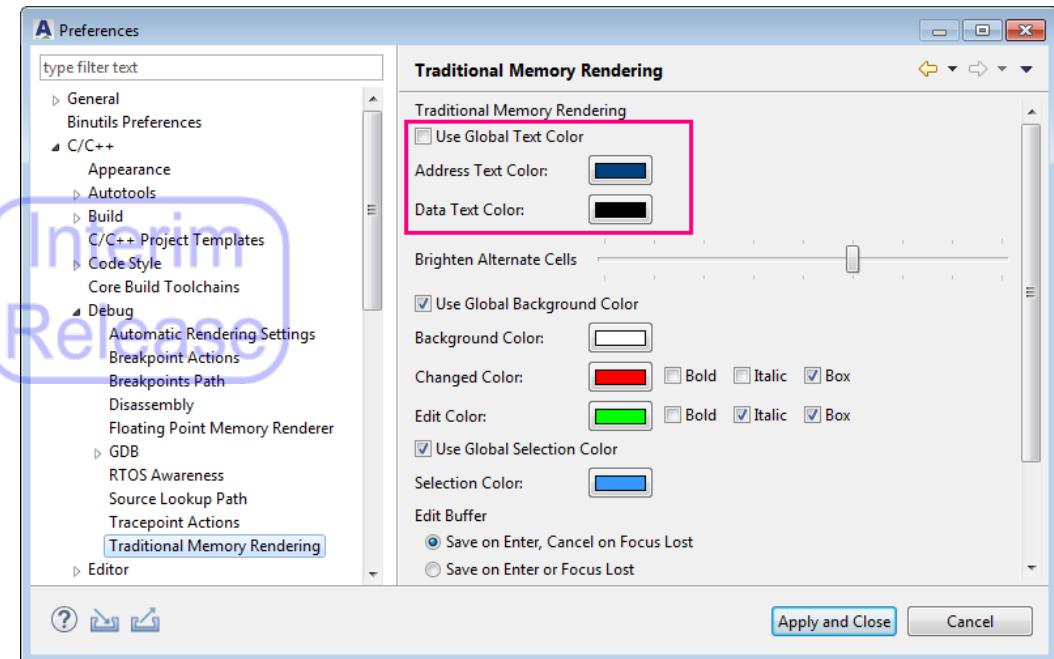
**NOTE**

The renderings are default displayed in black texts. You can also use different text colors to differentiate addresses from data in the renderings. Just follow below:

1. On the AndeSight main menu, select “Window > Preferences > C/C++ > Debug > [Traditional Memory Rendering|Floating Point Memory Rendering]” to invoke the **Traditional Memory Rendering** or **Floating Point Renderer** page.

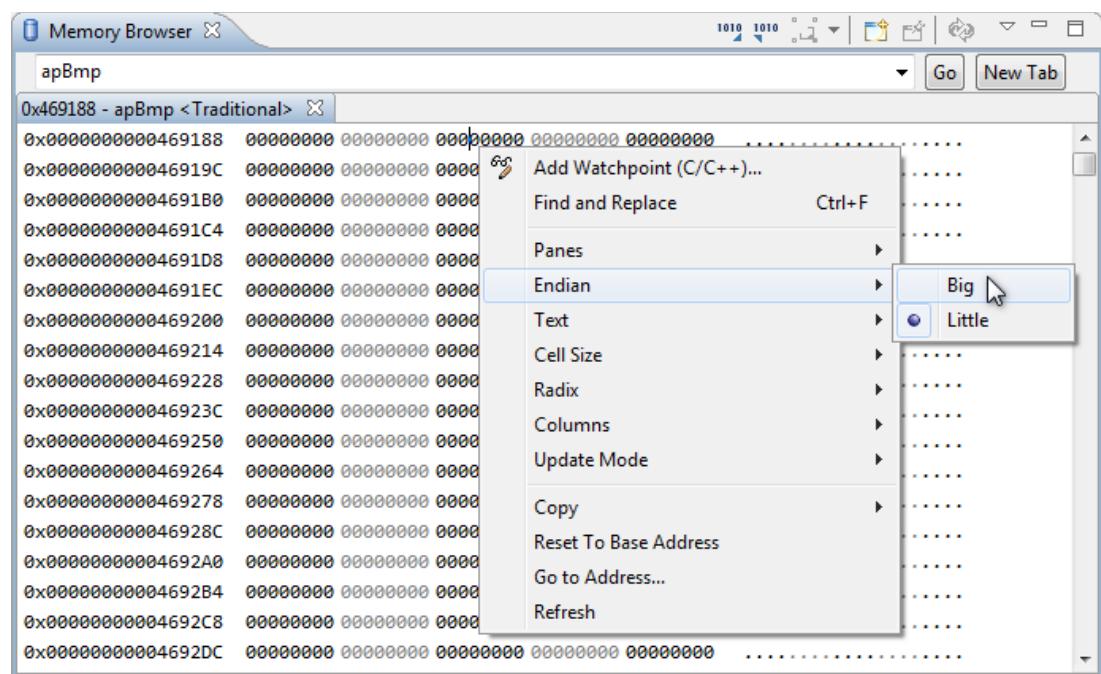


2. Uncheck the option “Use Global Text Color” first and define the respective text color of addresses and data.

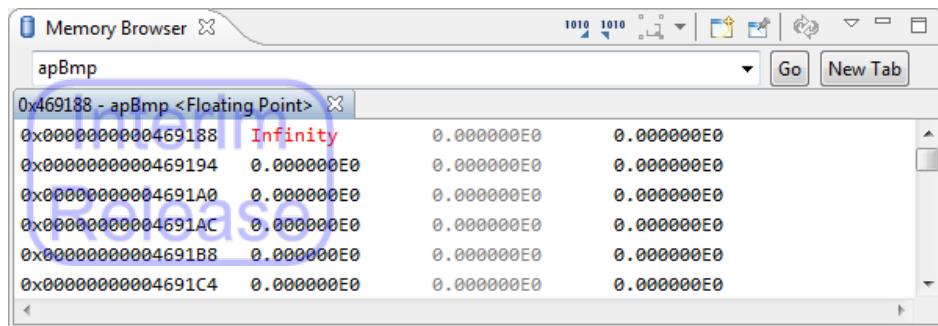


3. Click “Apply and Close” on the dialog to save the changes.

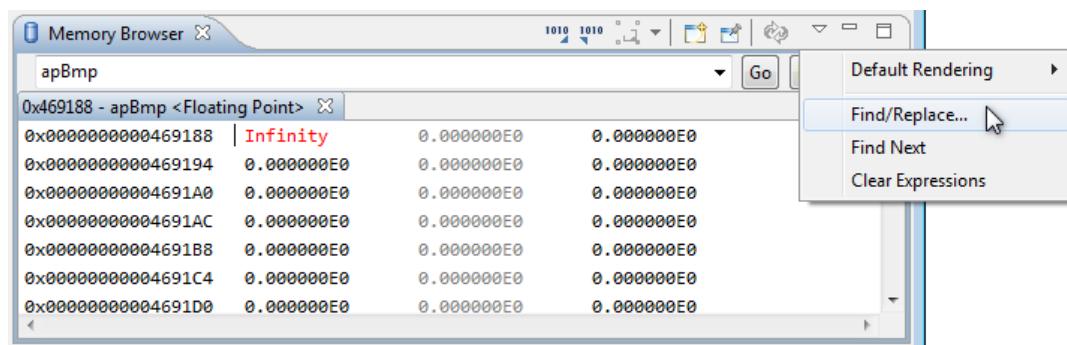
**Step 3** You may change the endianness or addressable size of the displayed data by right-clicking the rendering to invoke a pull-down menu and making your selections.



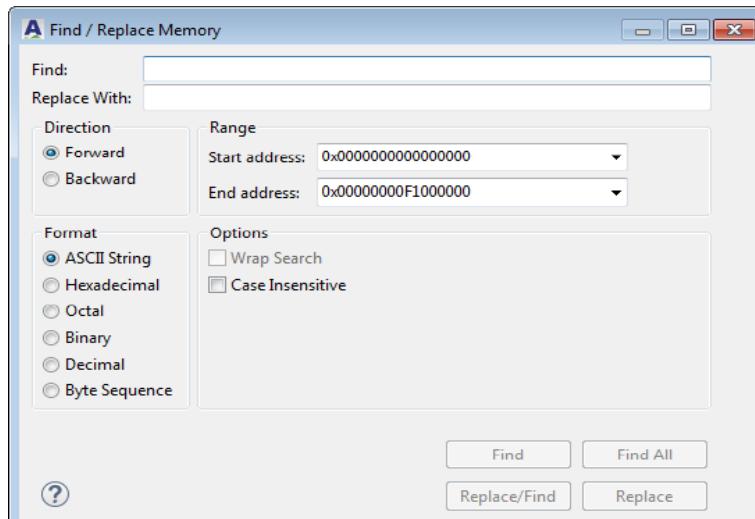
**Step 4** During program execution, the deviation of memory content is marked in red.



**Step 5** To find and replace specific data words in the selected memory monitor, click  (View Menu) on the toolbar and select “Find/Replace...” in the pull-down menu.



In the invoked **Find/Replace Memory** dialog, enter the data words to be found and substituted. Then, click the **Find** or **Replace** button.

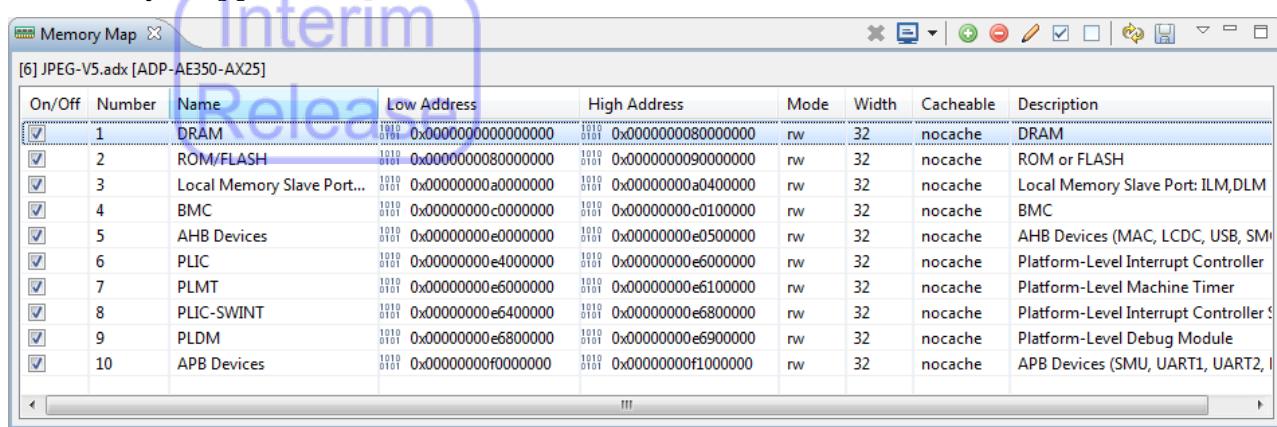


**Step 6** For ICE targets with a debug module supporting auto-refresh, you can select  (Auto Refresh) to periodically refresh the memory content during program execution.

Interim  
Release

#### 2.4.2.4 Memory Map view

The **Memory Map** view is an interface by which to examine or add/modify/delete memory regions, memory attributes (read-only, write-only, read/write), and descriptions of memory-mapped devices.



The screenshot shows the Memory Map view for the project [6] JPEG-V5.adx [ADP-AE350-AX25]. The table lists 10 memory regions:

On/Off	Number	Name	Low Address	High Address	Mode	Width	Cacheable	Description
<input checked="" type="checkbox"/>	1	DRAM	0x0000000000000000	0x0000000008000000	rw	32	nocache	DRAM
<input checked="" type="checkbox"/>	2	ROM/FLASH	0x0000000008000000	0x0000000009000000	rw	32	nocache	ROM or FLASH
<input checked="" type="checkbox"/>	3	Local Memory Slave Port...	0x00000000a0000000	0x00000000a0400000	rw	32	nocache	Local Memory Slave Port: ILM,DLM
<input checked="" type="checkbox"/>	4	BMC	0x00000000c0000000	0x00000000c0100000	rw	32	nocache	BMC
<input checked="" type="checkbox"/>	5	AHB Devices	0x00000000e0000000	0x00000000e0500000	rw	32	nocache	AHB Devices (MAC, LCDC, USB, SMI)
<input checked="" type="checkbox"/>	6	PLIC	0x00000000e4000000	0x00000000e6000000	rw	32	nocache	Platform-Level Interrupt Controller
<input checked="" type="checkbox"/>	7	PLMT	0x00000000e6000000	0x00000000e6100000	rw	32	nocache	Platform-Level Machine Timer
<input checked="" type="checkbox"/>	8	PLIC-SWINT	0x00000000e6400000	0x00000000e6800000	rw	32	nocache	Platform-Level Interrupt Controller S
<input checked="" type="checkbox"/>	9	PLDM	0x00000000e6800000	0x00000000e6900000	rw	32	nocache	Platform-Level Debug Module
<input checked="" type="checkbox"/>	10	APB Devices	0x00000000f0000000	0x00000000f1000000	rw	32	nocache	APB Devices (SMU, UART1, UART2, I

Notice that the read-only “Width” column in this view indicates the access size in the memory regions. The values in the “Width” column and the access sizes they represent are as follows:

<u>Width value</u>	<u>Access size in memory region</u>
8	8-bit (one-byte) memory access
16	16-bit (two-byte, half-word) memory access
32	32-bit (four-byte, a word) memory access
64*	64-bit (eight-byte, a quad word) memory access

(\*Width value “64” indicates 32-bit memory access for 32-bit CPU cores.)

#### Adding/Modifying/Deleting a user-defined memory region

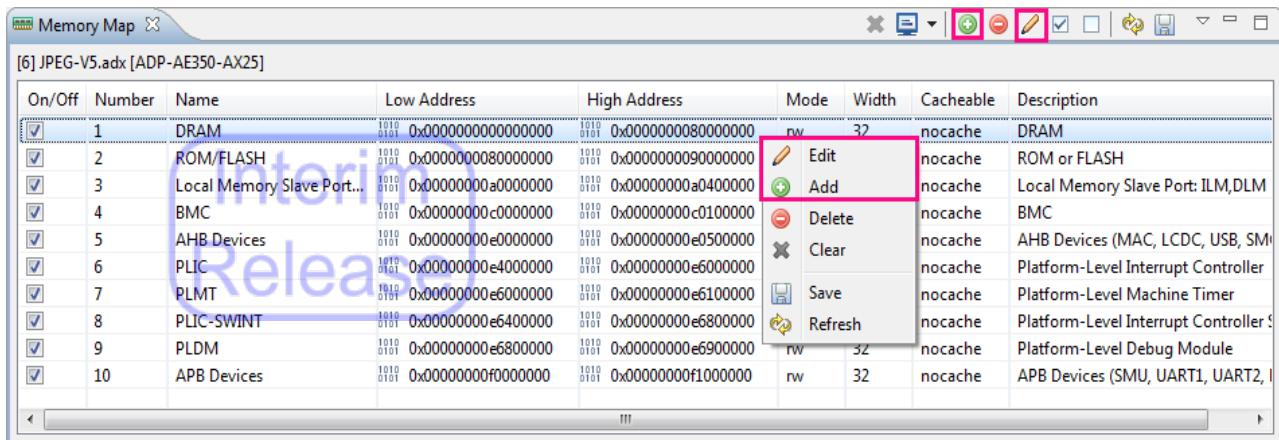
The following procedure is illustrated using the **Memory Map** view from a debug session of the JPEG-V5 project.

##### Step 1 Invoke the **Memory Map Region** wizard:

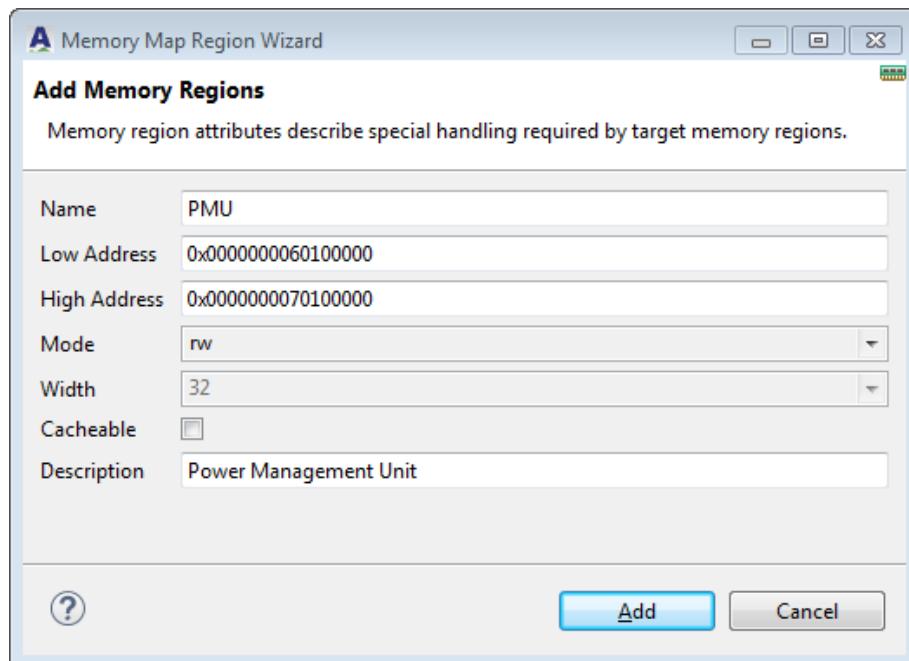
To add a memory region, click  (Add Memory Region) on the **Memory Map** view toolbar or right-click anywhere in the **Memory Map** view to select “Add” on the pop-up menu.

To modify a memory region, select a desired row in the **Memory Map** view and then click  (Modify Memory Regions) on the toolbar or

right-click and select “Edit” on the menu that pops up.



**Step 2** In the **Memory Map Region Wizard**, configure the options and click “Add” or “Change”:



- **Name:** Assign a name for the memory region.
- **Low Address:** Specify the inclusive lower bound of the memory region.
- **High Address:** Specify the exclusive upper bound of the memory region.
- **Description:** Provide a detailed description of the memory region.

■ Mode (Memory Access Mode):

Specify the mode by which GDB accesses the memory region. Memory access modes include the following:

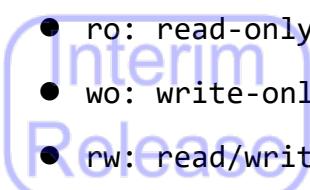
- ro: read-only access to a memory region
- wo: write-only access to a memory region
- rw: read/write access to a memory region (default mode)

■ Cacheable (Data Cache)

Specify whether GDB caches target memory. Enabling this attribute improves performance by reducing overhead associated with the debug protocol; however, it can lead to erroneous results because GDB is unaware of volatile variables or memory-mapped device registers.

- Cacheable (checked box): Enable GDB to cache target memory.
- Non-cacheable (unchecked box): GDB is not enabled to cache target memory (default setting).

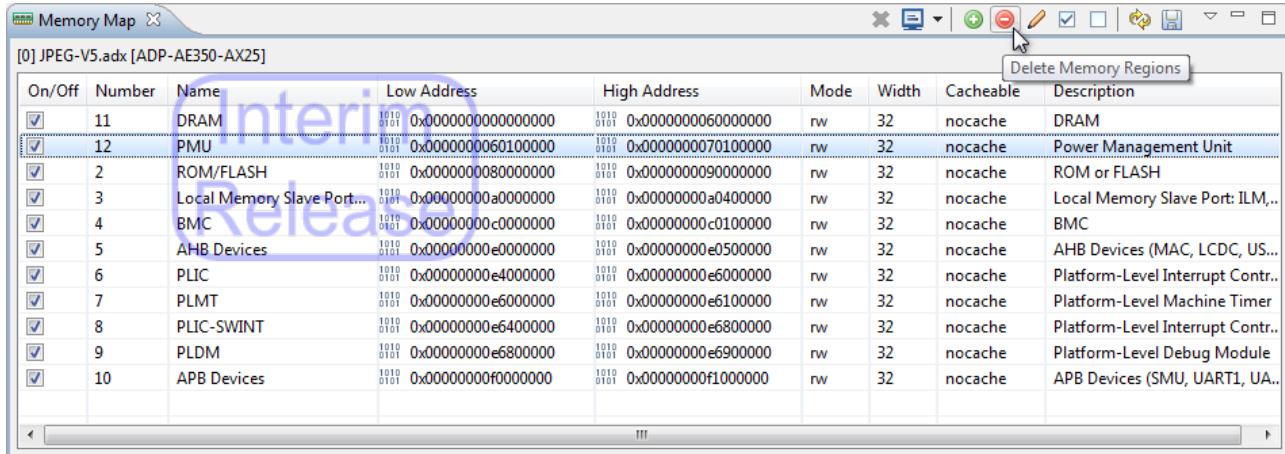
**Step 3** The created/modified memory region appears immediately in the **Memory Map** view. You may further enable/disable the region by selecting/unselecting the check box in the on/off column. To define additional memory regions, repeat Steps 1 to 3.



A screenshot of the AndeSight IDE's Memory Map window. The window title is "Memory Map". The current project is "[0] JPEG-V5.adx [ADP-AE350-AX25]". The table lists memory regions with the following columns: On/Off, Number, Name, Low Address, High Address, Mode, Width, Cacheable, and Description. Rows 11 and 12 are highlighted with a red border. The table contains 14 rows of memory regions, each with its address range, mode (rw), width (32), and cacheability status (nocache). The descriptions provide details about the memory mapped to specific components like DRAM, PMU, ROM/FLASH, BMC, AHB Devices, PLIC, PLMT, PLIC-SWINT, PLDM, and APB Devices.

On/Off	Number	Name	Low Address	High Address	Mode	Width	Cacheable	Description
<input checked="" type="checkbox"/>	11	DRAM	1010 0x0000000000000000	1010 0x0000000060000000	rw	32	nocache	DRAM
<input checked="" type="checkbox"/>	12	PMU	1010 0x0000000060100000	1010 0x0000000070100000	rw	32	nocache	Power Management Unit
<input checked="" type="checkbox"/>	2	ROM/FLASH	1010 0x0000000080000000	1010 0x0000000090000000	rw	32	nocache	ROM or FLASH
<input checked="" type="checkbox"/>	3	Local Memory Slave Port...	1010 0x00000000a0000000	1010 0x00000000a0400000	rw	32	nocache	Local Memory Slave Port: ILM,..
<input checked="" type="checkbox"/>	4	BMC	1010 0x00000000c0000000	1010 0x00000000c0100000	rw	32	nocache	BMC
<input checked="" type="checkbox"/>	5	AHB Devices	1010 0x00000000e0000000	1010 0x00000000e0500000	rw	32	nocache	AHB Devices (MAC, LCDC, US...
<input checked="" type="checkbox"/>	6	PLIC	1010 0x00000000e4000000	1010 0x00000000e6000000	rw	32	nocache	Platform-Level Interrupt Contr..
<input checked="" type="checkbox"/>	7	PLMT	1010 0x00000000e6000000	1010 0x00000000e6100000	rw	32	nocache	Platform-Level Machine Timer
<input checked="" type="checkbox"/>	8	PLIC-SWINT	1010 0x00000000e6400000	1010 0x00000000e6800000	rw	32	nocache	Platform-Level Interrupt Contr..
<input checked="" type="checkbox"/>	9	PLDM	1010 0x00000000e6800000	1010 0x00000000e6900000	rw	32	nocache	Platform-Level Debug Module
<input checked="" type="checkbox"/>	10	APB Devices	1010 0x00000000f0000000	1010 0x00000000f1000000	rw	32	nocache	APB Devices (SMU, UART1, UA..

**Step 4** To delete a memory region, select the row of the memory region and click  (Delete Memory Regions) on the toolbar.



#### 2.4.2.5 Cache Dump View

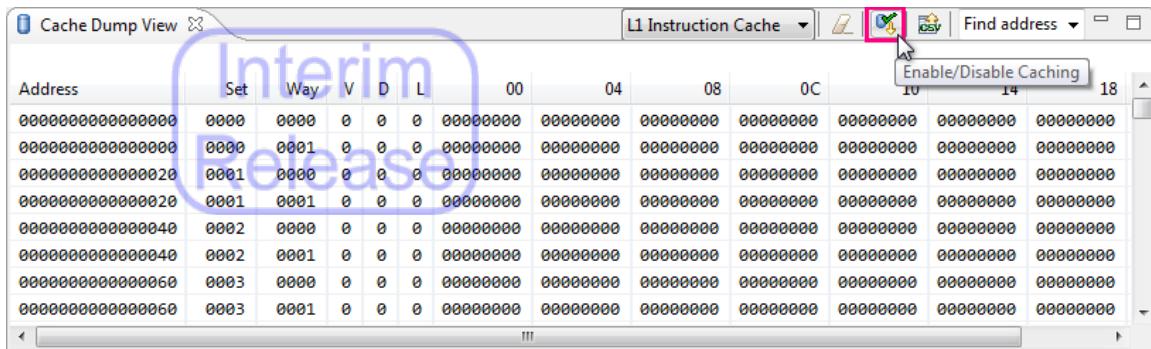
The **Cache Dump** view allows you to monitor CPU caches after program execution suspends. For each cache line, the view displays its current information including the address, set index, way number, valid state, dirty state, lock state and stored data.

Address	Set	Way	V	D	L	00	04	08	0C	10	14	18
0000000000000000	0000	0000	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000000	0000	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000020	0001	0000	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000020	0001	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000040	0002	0000	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000040	0002	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000060	0003	0000	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000060	0003	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000

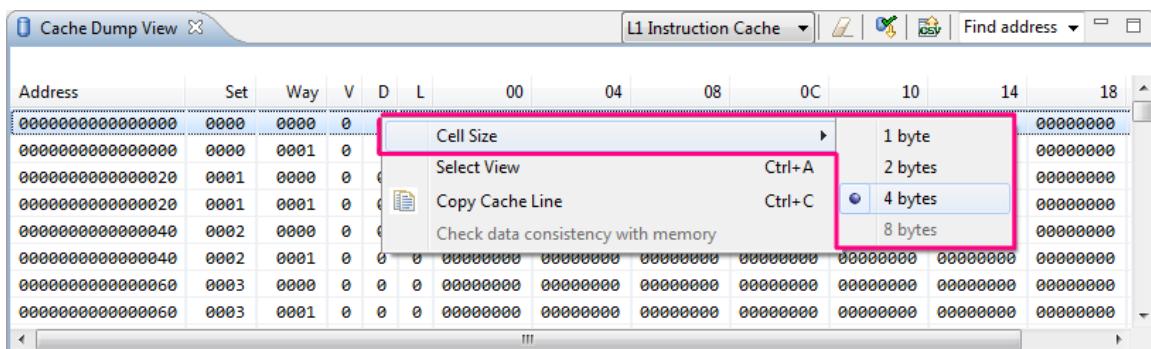
Whenever there are divergences from previous cache content after program execution or cache operation, the changes in table cells can be identified by a distinctive color.

Address	Set	Way	V	D	L	00	04	08	0C	10	14	18
0000000000405A00	0000	0000	1	0	0	FCDF06F	24843703	F207879B	07DB0799	639C0EF7	97828522	BF4DF905
0000000000409A00	0000	0001	1	0	0	9E878793	06C7EC63	00797613	0933C601	092140C9	7C0437DB	7F634705
0000000000401A20	0001	0000	1	0	0	00F93423	644260E2	690264A2	80826105	EC061101	E426E822	842AE04A
0000000000409A20	0001	0001	1	0	0	47390009	B783D798	D7C00009	0009B783	854E639C	000780E7	01890593
0000000000401A40	0002	0000	1	0	0	04078863	0717741C	07130000	EB98F2A7	00000717	F2C70713	0717EF98
0000000000409A40	0002	0001	1	0	0	84AA4920	04050063	098A3783	07E197CA	08FA3C23	045B0439	641C0E8A
0000000000401A60	0003	0000	1	0	0	F398F927	00004717	00C70713	0717F798	07130000	FB98F0A7	B423FF84
0000000000409A60	0003	0001	1	0	0	B8230124	E4040004	01848513	740270A2	694264E2	6A0269A2	80826145

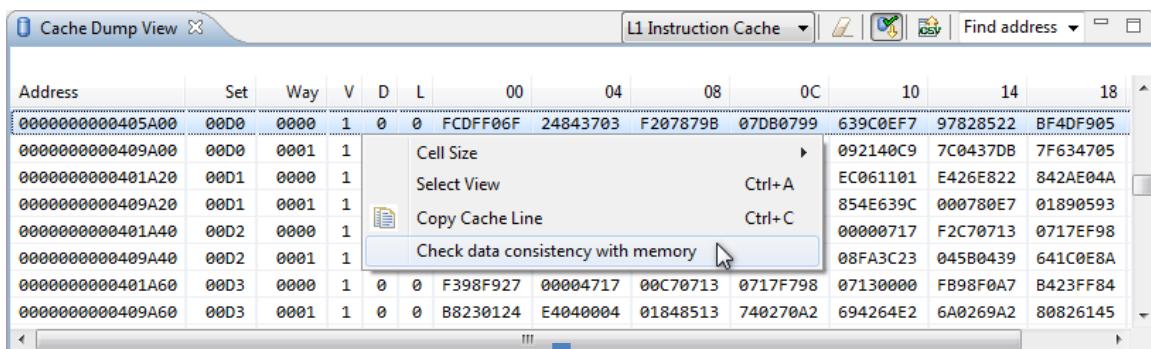
In addition to displaying cache information, this view also allows you to enable or disable CPU caching using the toggle button  on the toolbar. If CPU caching is enabled, data are accessed efficiently from the cache; if not, the CPU fetches data from the main memory.



You can adjust the display of cache content by changing the cell size (1/2/4/8 bytes). Just right-click cache information to invoke a pull-down menu and make a selection.



To check the consistency of data between cache and memory through this view, right-click on a cache line labelled as valid and select “Check data consistency with memory” from its pull-down menu. The result of whether the data in the cache line is identical with that in the corresponding memory shows on the top of the view right away.



**Cache Dump View**

All data in the cache line "0x405A00" matches that in the corresponding memory.

Address	Set	Way	V	D	L	00	04	08	0C	10	14	18
0000000000405A00	00D0	0000	1	0	0	FCDDFF06F	24843703	F207879B	07DB0799	639C0EF7	97828522	BF4DF905
0000000000409A00	00D0	0001	1	0	0	9E878793	06C7EC63	00797613	0933C601	092140C9	7C0437DB	7F634705
0000000000401A20	00D1	0000	1	0	0	00F93423	644260E2	690264A2	80826105	EC061101	E426E822	842AE04A
0000000000409A20	00D1	0001	1	0	0	47390009	B783D798	D7C00009	0009B783	854E639C	000780E7	01890593
0000000000401A40	00D2	0000	1	0	0	04078863	0717741C	07130000	EB98F2A7	00000717	F2C70713	0717EF98
0000000000409A40	00D2	0001	1	0	0	84AA4920	04050063	098A3783	07E197CA	08FA3C23	045B0439	641C0E8A

Interim  
Release

If a data mismatch is found on the selected cache line, you can hover your cursor over table cells of the addresses with data consistency problem and check the popup tooltips for the current data in the corresponding memory.

**Cache Dump View**

The data at the address(es) "0x40,0x44,0x48,0x4C,0x50,0x54,0x58,0x5C" of the selected cache line does not match that in the corresponding memory.

Address	Set	Way	V	D	L	00	04	08	0C	10	14	18	1C
0000000000000020	0001	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000040	0002	0000	1	0	0	80E70000	000070160	00008082	A00132A0	00001517	8C108082	00001617	29450513
0000000000000040	0002	0001	0	0	0	00000000	Current 00970160	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000060	0003	0000	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000060	0003	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000080	0004	0000	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000000000000080	0004	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

## Toolbar for the Cache Dump view

### Switch between instruction cache and data cache

L1 Instruction Cache ▾

In the combo box, specify whether to monitor the content of the instruction cache or that of the data cache.

**Cache Dump View**

L1 Instruction Cache

L1 Data Cache

Address	Set	Way	V	D	L	00	04	08	0C	10	14	18
0000000000405A00	00D0	0000	1	0	0	FCDDFF06F	24843703	F207879B	07DB0799	639C0EF7	97828522	BF4DF905
0000000000409A00	00D0	0001	1	0	0	9E878793	06C7EC63	00797613	0933C601	092140C9	7C0437DB	7F634705
0000000000401A20	00D1	0000	1	0	0	00F93423	644260E2	690264A2	80826105	EC061101	E426E822	842AE04A
0000000000409A20	00D1	0001	1	0	0	47390009	B783D798	D7C00009	0009B783	854E639C	000780E7	01890593
0000000000401A40	00D2	0000	1	0	0	04078863	0717741C	07130000	EB98F2A7	00000717	F2C70713	0717EF98
0000000000409A40	00D2	0001	1	0	0	84AA4920	04050063	098A3783	07E197CA	08FA3C23	045B0439	641C0E8A

### Invalidate all



This button allows you to flush entire cache and is thus often used during a context switch. After this button is pressed, the entire cache will be unlocked and set to invalid, forcing the CPU to retrieve all data from the main memory.

### Enable/disable cache

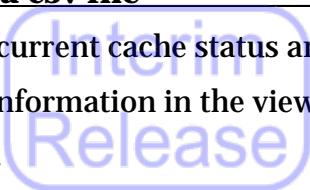


Determine whether the cache can be looked up or not.

### Export to a csv file



Export the current cache status and data to a CSV file. If you only need to keep the visualized information in the view, select and copy it using right-click menu and paste to a desired file.



Address	Set	Way	V	D	L	00	04	08	0C	10	14
00000000000404000	0000	0000	0	0	0	B8030FD7					
00000000000000000	0000	0001	0	0	0	00000000					
00000000000414020	0001	0000	0	0	0	43017159					
00000000000408020	0001	0001	0	0	0	842AE022					
00000000000414040	0002	0000	0	0	0	F85AFD00					
00000000000408040	0002	0001	0	0	0	0717E118					

Select View Ctrl+A  
Copy Cache Line Ctrl+C  
Check data consistency with memory

### Locate memory data in the cache

Find address ▾

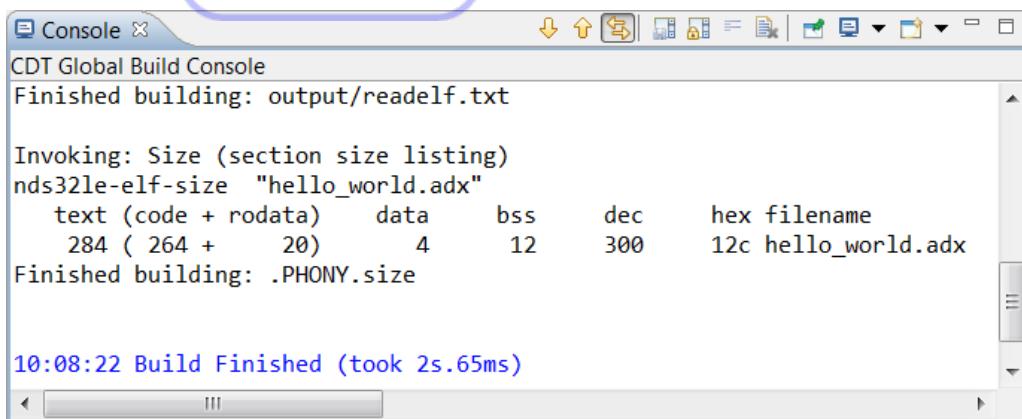
To determine if the data at a specific memory location is valid in the cache, enter its address in this search bar and press the enter key. If the data is valid in the cache, the cache line containing that address will be highlighted in the view.

Address	Set	Way	V	D	L	00	04	08	0C	10	14
00000000000404000	0000	0000	1	0	0	B8030FD7	B7830007	87DB070F	638C0FD7	B7C94701	00008067 22
00000000000000000	0000	0001	0	0	0	00000000	00000000	00000000	00000000	00000000	00000000 00
00000000000414020	0001	0000	1	0	0	A01D4881	00D7873B	0EE6075B	0ED605DB	FF85B583	2785E30C 00
00000000000408020	0001	0001	1	0	0	842AE022	6B9C611C	85229782	FA5F80EF	00100513	7A1050EF 00
00000000000404040	0002	0000	1	0	0	03132885	00010603	DD635D1C	278304F8	270300C3	87BB0243 B6
00000000000408040	0002	0001	1	0	0	0717E118	07130000	E518EBA7	00000717	FA870713	0717E918 07

#### 2.4.2.6 Console view

The **Console** view shows the output of a process while providing a command line interface for the input of commands. Multiple consoles can be open simultaneously, and you may

switch between them by clicking the drop-down arrow of  (Display Selected Console) on the toolbar for this view and selecting a desired console from the pull-down menu. The following is a CDT Build Console showing the build result of the hello\_world project.



The screenshot shows the 'Console' view of the AndeSight IDE. The title bar says 'Console'. The main area displays the output of a build process:

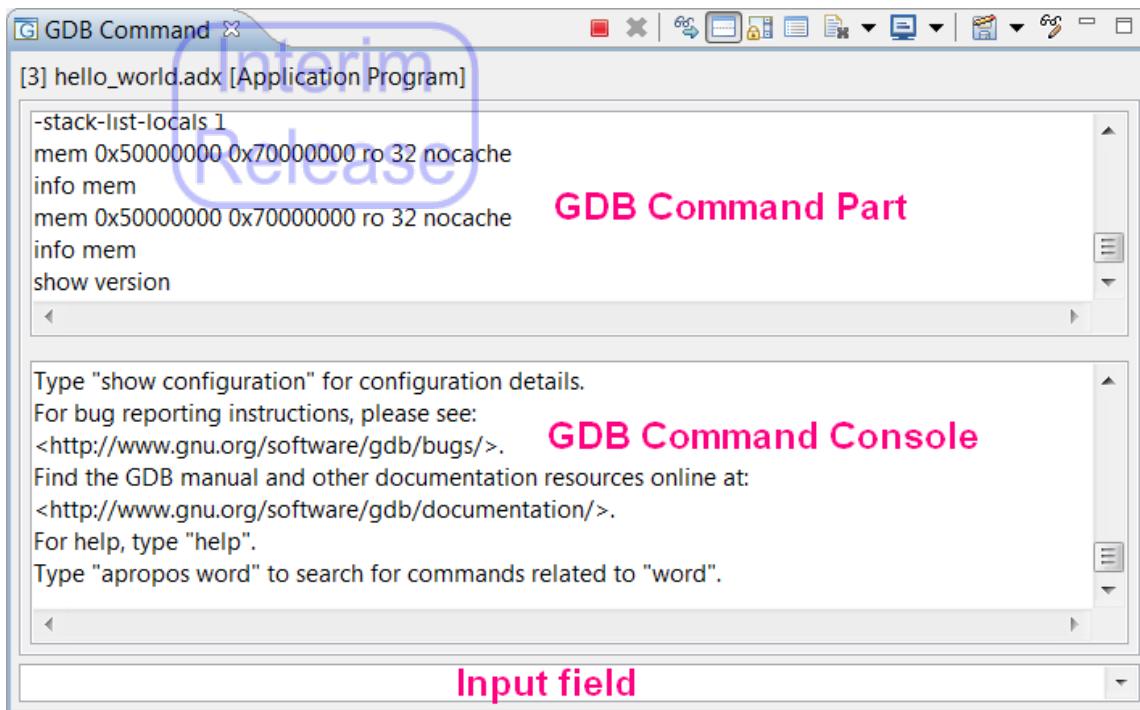
```
CDT Global Build Console
Finished building: output/readelf.txt

Invoking: Size (section size listing)
nds32le-elf-size "hello_world.adx"
text (code + rodata)    data      bss      dec      hex filename
 284 ( 264 +     20)      4       12      300      12c hello_world.adx
Finished building: .PHONY.size

10:08:22 Build Finished (took 2s.65ms)
```

#### 2.4.2.7 GDB Command view

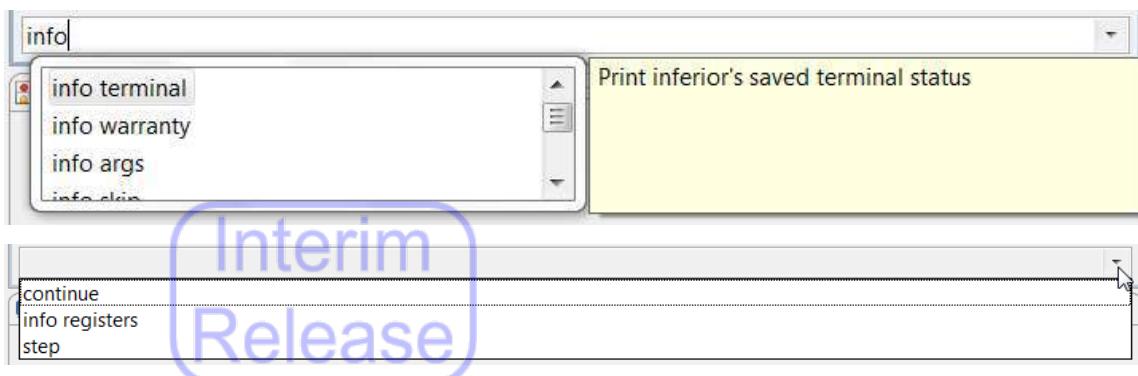
The **GDB Command** view serves as a GDB command interface. The figure below shows the **GDB Command** view from a debug session of the hello\_world project.



The **GDB Command** view embodies three parts. The upper column is a read-only log view that shows all GDB commands issued in a debug session. This column is hidden by default, but you may invoke it by clicking  (Show Command Part) on the toolbar.

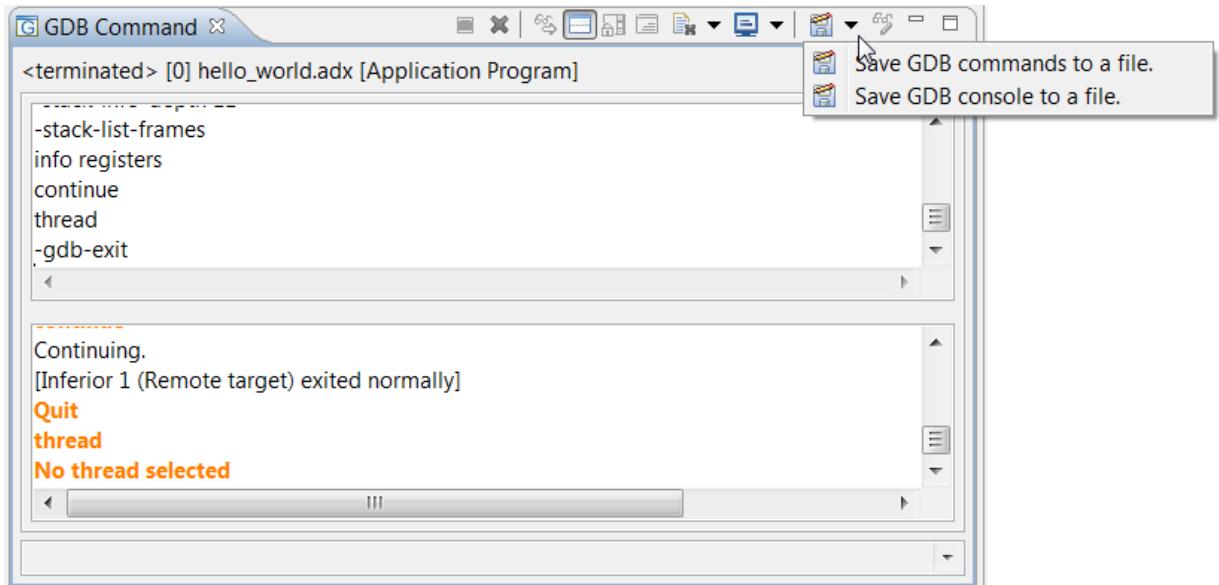
The central column is a GDB command console that displays the GDB commands and results. Different elements are printed in different colors: orange messages are GDB commands; blue are program outputs; black are the results of GDB commands.

The bottom part of the **GDB Command** view is an input field in which to issue GDB commands. This field provides command suggestions and descriptions when you enter a command. It also saves up to 20 records of GDB commands entered for a given project. After launching an existing project, you can check your GDB command history by clicking the pull-down arrow at the right or using the up and down arrow keys.



You may export the messages in the GDB command part or GDB command console to a file.

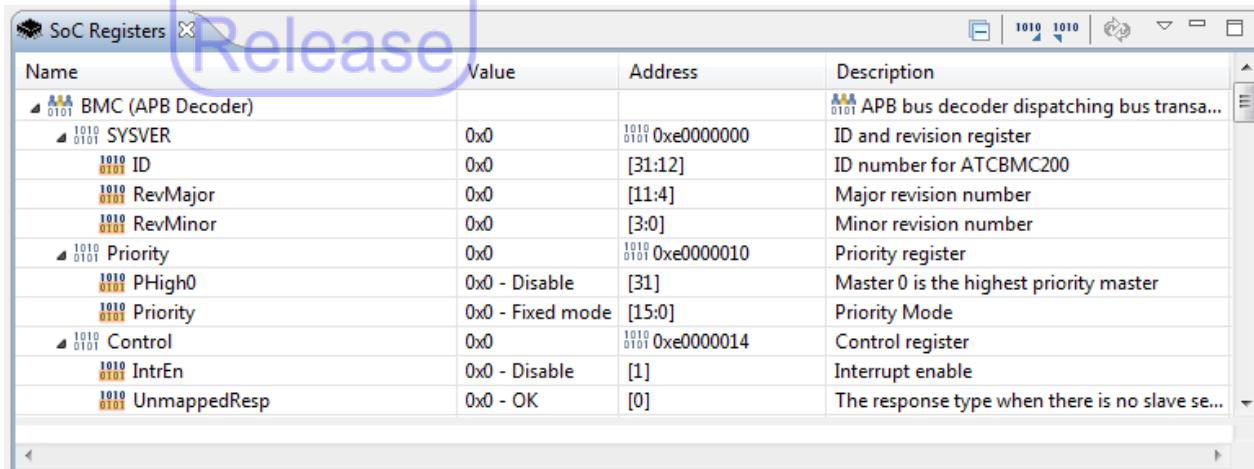
Simply click the drop-down arrow of  (Save GDB Context) on the toolbar and make the desired selection. To reload exported GDB commands, click  (Reload GDB Command from File) on the toolbar and select the exported file.



#### 2.4.2.8 SoC Registers view

The **SoC Registers view** allows you to read from and write to SoC register values down to the bit field level. This view is evoked automatically as part of the debug perspective. The predefined SoC description files in AndeSight are placed under the directory

**ANDESI GHT\_ROOT\SoC**.



Name	Value	Address	Description
APB Decoder			APB bus decoder dispatching bus transaction
SYSPWR	0x0	0xe0000000	ID and revision register
ID	0x0	[31:12]	ID number for ATCBMC200
RevMajor	0x0	[11:4]	Major revision number
RevMinor	0x0	[3:0]	Minor revision number
Priority	0x0	0xe0000010	Priority register
PHigh0	0x0 - Disable	[31]	Master 0 is the highest priority master
Priority	0x0 - Fixed mode	[15:0]	Priority Mode
Control	0x0	0xe0000014	Control register
IntrEn	0x0 - Disable	[1]	Interrupt enable
UnmappedResp	0x0 - OK	[0]	The response type when there is no slave selected

#### Configuring the SoC Registers view

The SoC registers that appear in this view are defined using a customizable meta file. You may create and modify a SoC description file by editing the file via the **SoC Registers Editor** (see Section 2.2.1.2, Step 6) or following the steps below:

- Step 1** Install Python 2.7.x from the official Python website (<http://www.python.org/>) and set the folder path of Python installation in your OS system environment. For example, set the Python installation folder to **PATH=C:\Python27;%PATH%** for Windows.
- Step 2** Based on a given .csv file in **ANDESI GHT\_ROOT\SoC\SoCGenerator**, create a description file for your SoC registers and save it under your SoC name (e.g. AE250-4GB-NEW.csv). Open the description file to find the SoC registers organized in groups. The rows that are defined only by their base address list the group names, whereas the rows defined using an offset address or other values under the groups are class-binding SoC registers.

1	Name	Description	Address(h)	Width(bit)	Value(hex)	Reset Valu	Readable	Writable	Mask(hex)
2	BMC (APB Decoder)	APB bus decoder dispatching bus transactions from	0xE0000000						
3	SYSVER	— Group — Register	ID and revision register	0x00	32	Hardwired	True	False	
4	bit field:ID	ID number for ATCBMC200	[31:12]	20	0x00002	True	False		
5	bit field:RevMajor	Major revision number	[11:4]	8		True	False		
6	bit field:RevMinor	Minor revision number	[3:0]	4		True	False		
7	Priority	— Register	Priority register	0x10	32	0xF	True	True	
8	bit field:PHigh0	Master 0 is the highest priority master	[31]	1	0x0	True	True		
9	mnemonic:Disable	regardless of the Priority mode(Disable)			0x0	True	True		
10	mnemonic:Enable	regardless of the Priority mode(Enable)			0x1	True	True		
11	bit field:Priority	Priority Mode	[15:0]	16	0xFFFF	True	True		
12	mnemonic:Fixed mode	The priority from high to low is Master 0, Master 1, Master 2...Master 15 0x0				True	True		
13	Control	— Register	Control register	0x14	32	0x0	True	True	
14	bit field:IntrEn	Interrupt enable	[1]	1	0x0	True	True		
15	mnemonic:Disable	Disable interrupt			0x0	True	True		

The bit field information of each SoC register is organized hierarchically from bit field groups to bit fields and mnemonics.

APB slave base/size register 1	— Register	APB slave base/size register	0x20	32	Depends on	True	False
bitFieldGroup:bitFields	— Bit field group	Expression:32bit					
bit field:Base	— Bit fields	Base address	[31:20]	12	Depends on	True	False
bit field:Size		Size of the address space	[3:0]	4	Depends on	True	False
mnemonic:Invalid slave		Invalid slave		0x0		True	False
mnemonic:1K		1M		0x1		True	False
mnemonic:2K		2M		0x2		True	False
mnemonic:4K		4M		0x3		True	False
mnemonic:8K		8M		0x4		True	False
mnemonic:16K		16M		0x5		True	False
mnemonic:32K		32M		0x6		True	False
mnemonic:64K		64M		0x7		True	False
mnemonic:128K		128M		0x8		True	False
mnemonic:256K		256M		0x9		True	False
mnemonic:512K		512M		0xA		True	False
mnemonic:1M		1G		0xB		True	False
mnemonic:2M		2G		0xC		True	False
mnemonic:Reserved		Reserved		0xD		True	False
mnemonic:Reserved		Reserved		0xE		True	False
mnemonic:Reserved		Reserved		0xF		True	False
bitFieldGroup:bitFields1		Expression:24bit					
bit field:Base		Base address	[23:10]	14	Depends on	True	False

**Step 3** Edit the description file in accordance with the descriptions of peripheral registers in the datasheet of your SoC and save your changes. Note that the mask column has a default value of 0xFFFFFFFF when it is left blank.

**Step 4** Double-click `main.py` under `ANDESIGHT_ROOT\SoC\SoCGenerator` to generate a corresponding .regs file of your SoC description file (e.g. AE250-4GB-NEW.regs) in the folder `ANDESIGHT_ROOT\SoC`.

**Step 5** As shown below, you may update the SoC Registers file name in the associated targetboard.properties file (under **ANDESIGHT\_ROOT\target\TARGET**, see Section 2.2.1.1), or ChipProfile.atd file (Chip Profile Property File, see Section 2.2.1.2). The peripheral registers will then be displayed in the **SoC Registers** view.

(Interim Release)

```

1  #--- Chip Profile ===
2  chip.id=ADP-AE250-N25
3
4  #--- CPU Description ===
5  cpu.cores=1
6  cpu0.type=N25
7  cpu0.isa.reduceregs=N
8  cpu0.registers=../../CPU/V5-32.crgs
9  #cpu0.c_compiler_opt=
10 #cpu0.cpp_compiler_opt=
11 #cpu0.linker_opt=
12
13 #--- SoC Registers Description File ===
14 soc.registers=../../SoC/AE250-4GB-NEW.regs
15
16 #--- Memory Map Description File ===
17 memory.map=../../MemoryMap/AE250-4GB-memory.mem

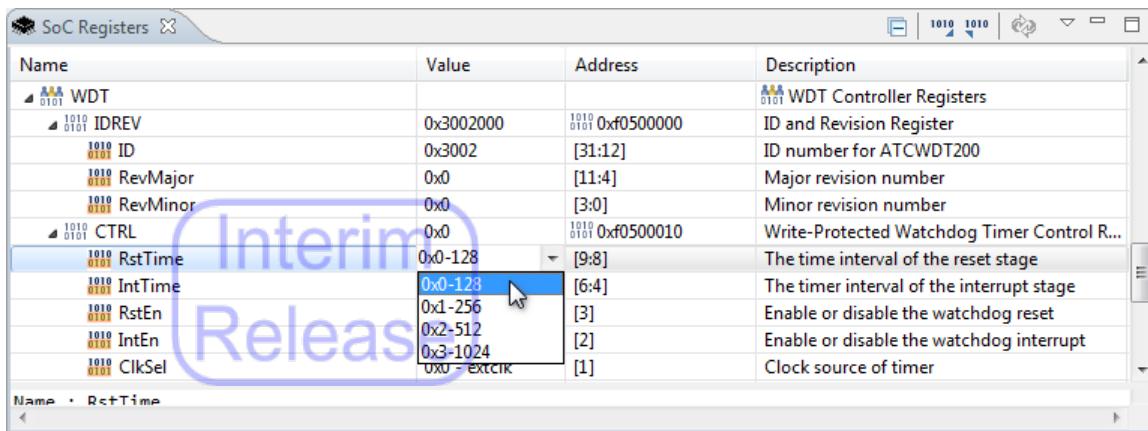
```

## Importing/exporting register values

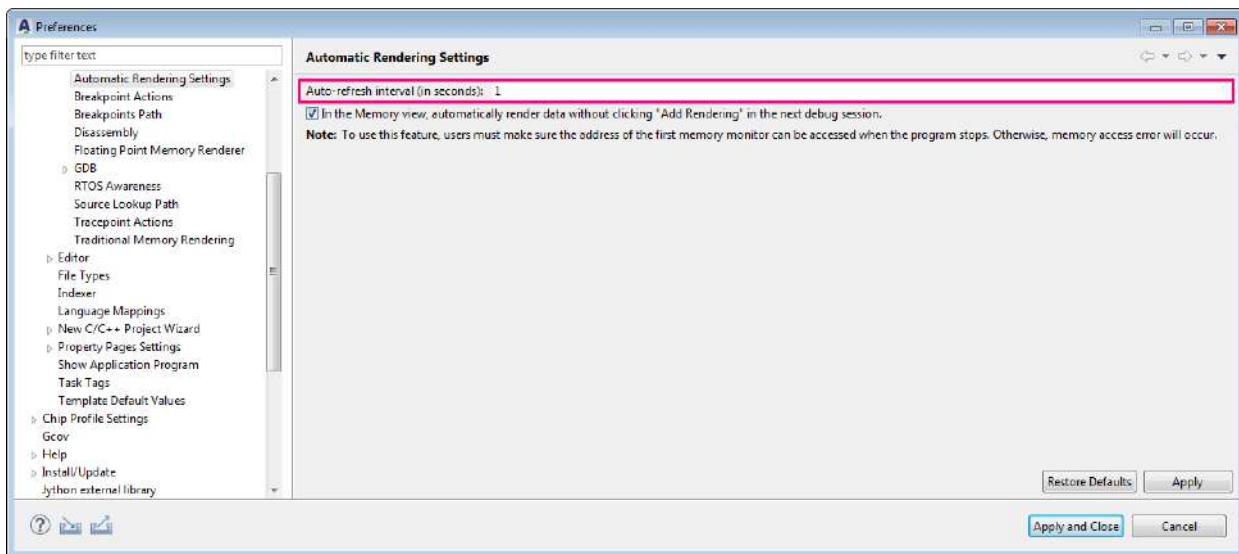
You can export values of specific SoC registers at runtime to a file. To do so, select the SoC components or registers of interest (multiple selections can be performed by holding the “Ctrl”) and click  (Export) on the toolbar. By clicking  (Import) on the toolbar, you may import exported SoC register data for further investigation.

## Modifying register values

You may modify the register or bit field values directly within the table cells. Bit field values may also be modified by making a selection from the combo box of mnemonics.

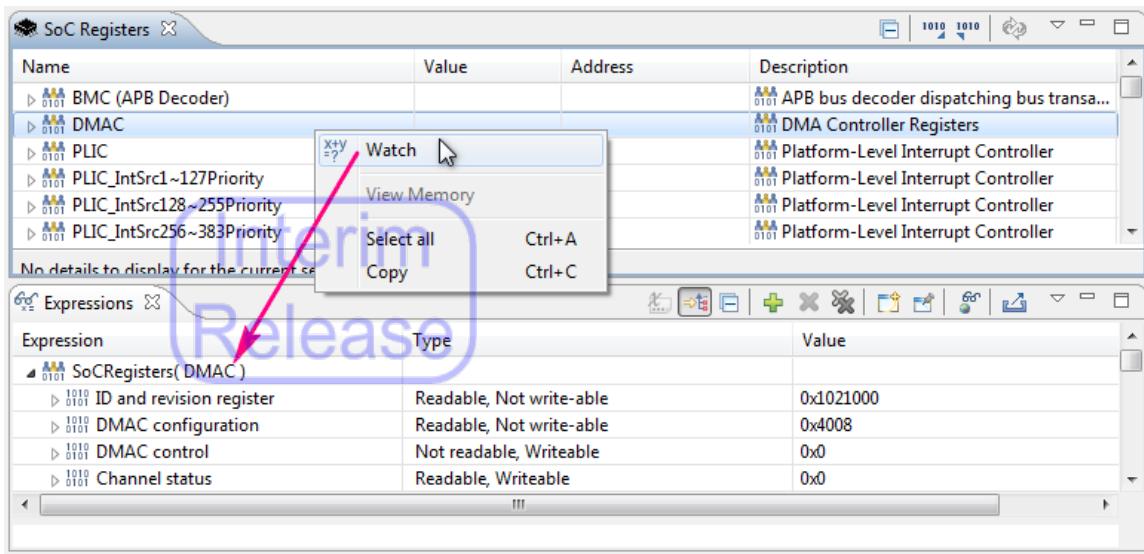


Even during runtime, you may click  (Auto Refresh) on the toolbar of the **SoC Registers** view to update the changes made to the register or bit field values. The default refresh interval is 1 second; however, you can change it in the **Automatic Rendering Settings** page of the **Preferences** settings, which can be opened as follows: “AndeSight main menu > Window > Preferences > C/C++ > Debug > Automatic Rendering Settings”.



## Watching SoC registers

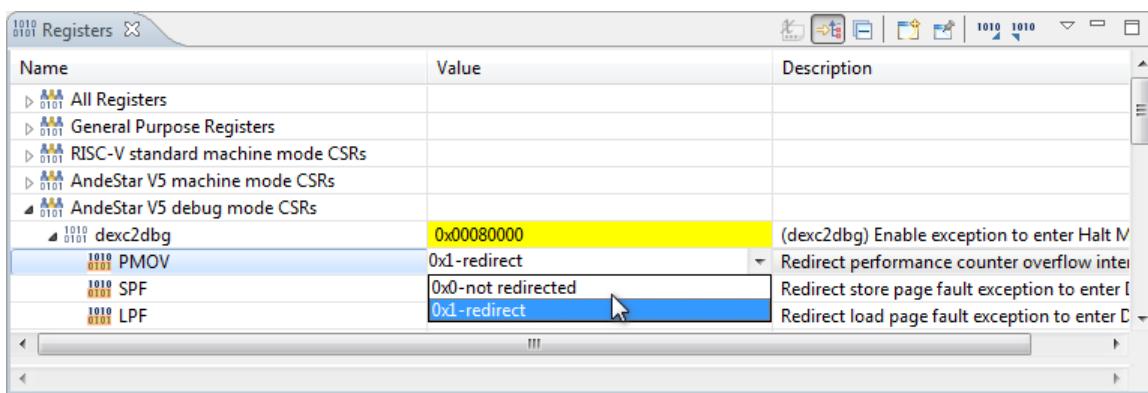
SoC registers can be watched by groups or individually. In the **SoC Registers** view, select the desired group or register and right-click to select “Watch” on the pull-down menu. The selected SoC registers appear in the **Expressions** view for inspection.



#### 2.4.2.9 Registers view

This view displays information about CPU registers down to the bit field level. You may modify registers or bit field values directly within the table cells. Bit field values can also be configured by making a selection from the combo box of mnemonics. To export or import register values, simply click  (Export) or  (Import) on the toolbar of this view and the register values will be exported or imported in key-value format. Like the **SoC Registers** view, the grouping of registers in this view is also configurable.

(Interim Release)



The screenshot shows the 'Registers' view in the AndeSight IDE. The window title is 'Registers'. The table has three columns: 'Name', 'Value', and 'Description'. The 'Value' column for the 'dexc2dbg' register is highlighted in yellow. A dropdown menu is open over the 'Value' cell for the 'PMOV' register, showing options: '0x0-not redirected', '0x1-redirect', and '0xd-redirect'. The 'dexc2dbg' register is described as '(dexc2dbg) Enable exception to enter Halt M'. The 'PMOV' register is described as 'Redirect performance counter overflow inter'. The 'SPF' register is described as 'Redirect store page fault exception to enter I'. The 'LPF' register is described as 'Redirect load page fault exception to enter C'.

Name	Value	Description
All Registers		
General Purpose Registers		
RISC-V standard machine mode CSRs		
AndeStar V5 machine mode CSRs		
AndeStar V5 debug mode CSRs		
dexc2dbg	0x00080000	(dexc2dbg) Enable exception to enter Halt M
PMOV	0x1-redirect	Redirect performance counter overflow inter
SPF	0x0-not redirected	Redirect store page fault exception to enter I
LPF	0xd-redirect	Redirect load page fault exception to enter C

#### 2.4.2.10 Variables view

This view lists information about the variables associated with the stack frame selected in the **Debug** view.



Name	Type	Value
▷  cinfo	j_decompress_ptr	<optimized out>
▷  compptr	jpeg_component_info *	<optimized out>
▷  coef_block	JCOEFPTR	<optimized out>
▷  output_buf	JSAMPARRAY	0x730118
(x)= output_col	JDIMENSION	0x98
(x)= tmp0	INT32	<optimized out>

When program execution is suspended, changes in variable values are highlighted.

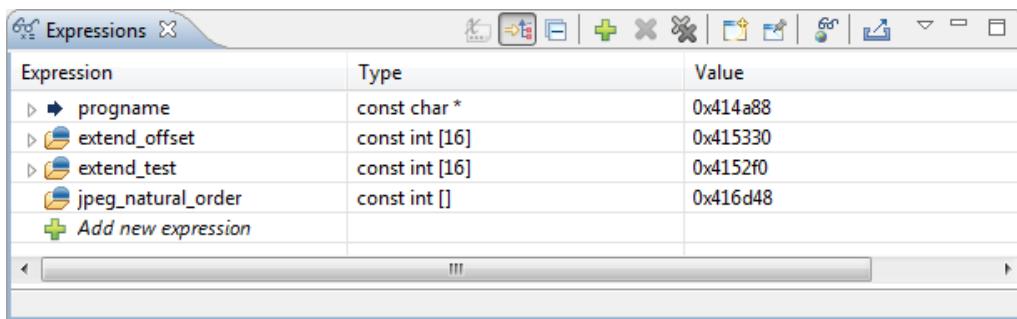


Name	Type	Value
▷  cinfo	j_decompress_ptr	0xeffd48
▷  dinfo	djpeg_dest_ptr	0x72da60
(x)= rows_supplied	JDIMENSION	<optimized out>
▷  dest	bmp_dest_ptr	0x72da60
▷  image_ptr	JSAMPARRAY	0x7306f8
▷  inptr	JSAMPROW	0x731b39

#### 2.4.2.11 Expressions view (for displaying global variables)

Global variables are not displayed in the **Variables** view, but rather in the **Expressions** view or **Global Variables Live** view (Section 2.4.2.12). Updated values of monitored global variables are highlighted in yellow. The **Expressions** view shows the updated values only after the program is suspended. The **Global Variables Live** view shows the values of global variables during runtime as long as the program runs on an ICE target with a V5 core.

The **Expressions** view also allows you to export the values of global variables for further inspection. The figure below shows the **Expressions** view from a debug session of a JPEG-V5 demo program.



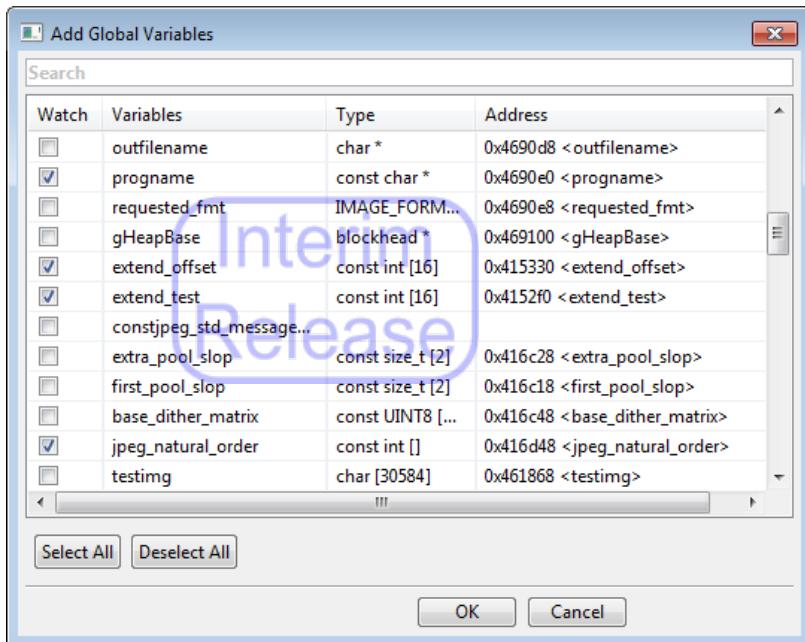
Expression	Type	Value
▶ programe	const char *	0x414a88
▶ extend_offset	const int [16]	0x415330
▶ extend_test	const int [16]	0x4152f0
▶ jpeg_natural_order	const int []	0x416d48
+ Add new expression		

Toolbar for the Expressions view

#### Add global variables



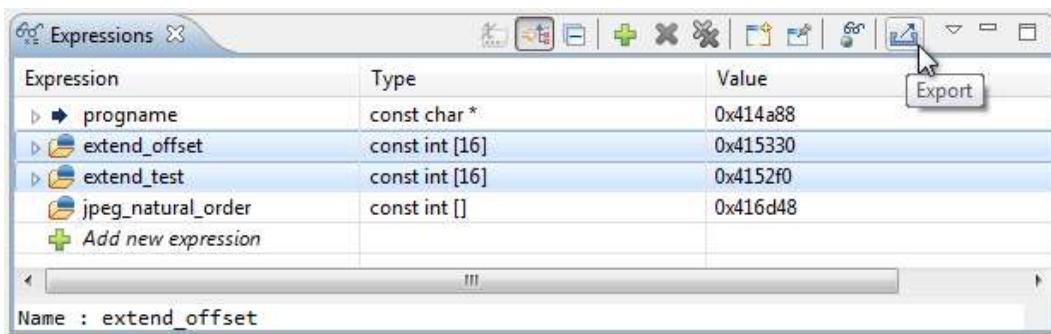
After suspending program execution, click this button to invoke the **Add Global Variables** wizard and select the global variables you wish to monitor. You may use the search bar in the upper part of the wizard to search for specific global variables. The selected global variables and their values are displayed in the **Expressions** view.



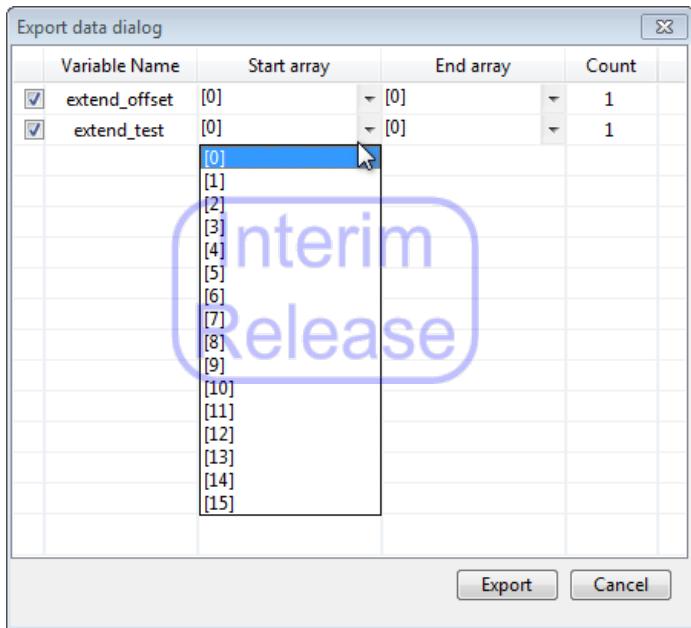
## Export



This button enables you to export global variables of interest. In the **Expressions** view, simply select the global variables that you wish to export and click  (Export) to invoke the **Export Data Dialog**.



In the dialog, specify the start and end arrays for the export of variables. Then, click "Export" to begin the export process.



Interim  
Release

#### 2.4.2.12 Global Variables Live View

The **Global Variables Live View** makes it possible to inspect global variables during runtime as long as the program is running on an ICE target. You can monitor selected global variables and examine variable members in the expanded tree after the program is suspended. The values of global variables or structure members updated during program execution are highlighted in yellow in this view.

When using a simulator target, this view shows the updated values of global variables only after program execution is suspended.

The figure below shows the **Global Variables Live View** from a debug session of an MP3 demo.

Name	Type	Value	Address
audio	struct audio	{...}	0x534960 <audio>
> start	unsigned char *	0x01000000	0x534960 <audio>
data_sz	unsigned long	262656	0x534964 <audio+4>
samplerate_inited	unsigned char	1	0x534968 <audio+8> """"
samplerate	unsigned short	48000	0x53496a <audio+10>
samplesize_inited	unsigned char	1	0x53496c <audio+12> """"
samplesize	unsigned short	16	0x53496e <audio+14>
channels_inited	unsigned char	1	0x534970 <audio+16> """"
channels	unsigned short	1	0x534972 <audio+18>

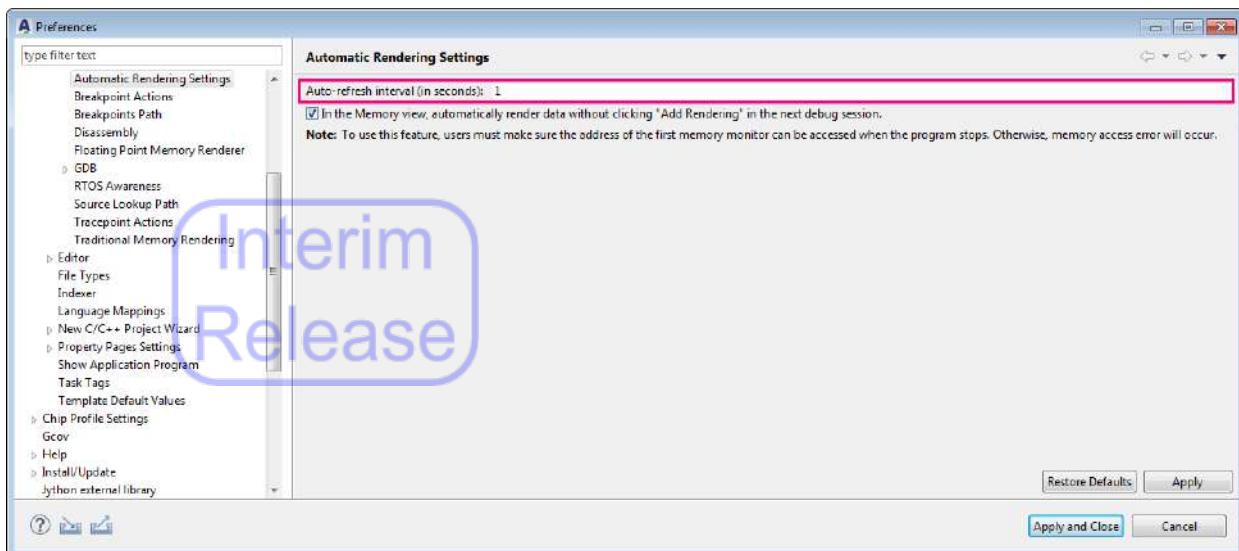
#### Toolbar for the Global Variables Live View

##### Auto refresh



This button makes it possible to refresh global variables in real time. This option is selected by default for ICE targets with a debug module supporting auto-refresh.

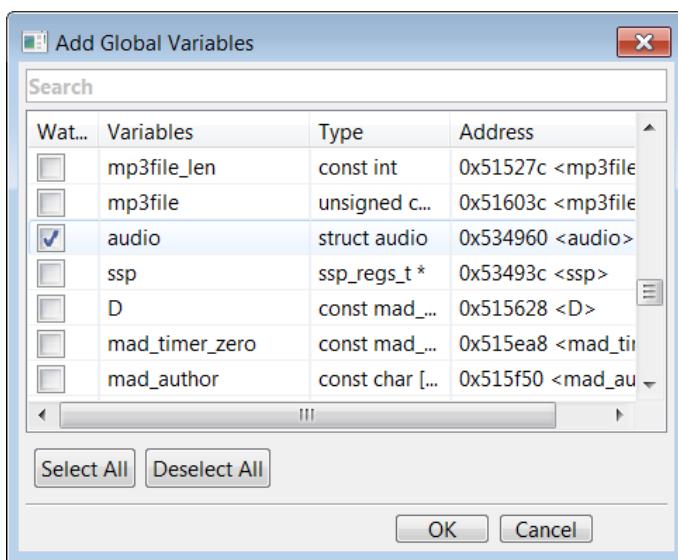
The default refresh interval is 1 second; however, you may change this on the **Automatic Rendering Settings** page under the **Preferences** settings, which can be opened as follows: “AndeSight main menu > Window > Preferences > C/C++ > Debug > Automatic Rendering Settings”.



## Add global variables



After program execution is suspended, click this button to invoke the **Add Global Variables** wizard and select the global variables that you wish to monitor. Use the search bar in the upper part of the wizard to search for specific global variables. Selected global variables and their information (types, values and addresses) are displayed in the **Global Variables Live View**.



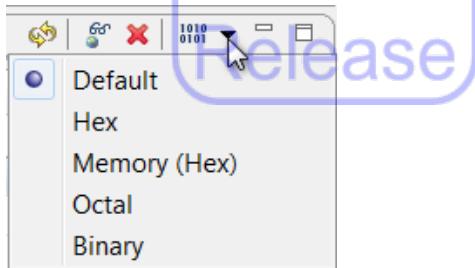
## Delete selected variables



A monitored global variable can be removed by selecting it in the **Global Variables Live View** and clicking this button.

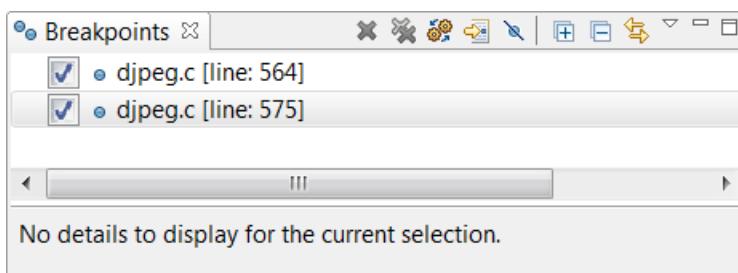
### **Select format for values**

Click the drop-down arrow of  and from the pull-down menu select a display format for the values of monitored global variables. Note that the format “Memory (hex)” refers to the hex values stored in memory, the order of which varies according to the endianness of the target system.



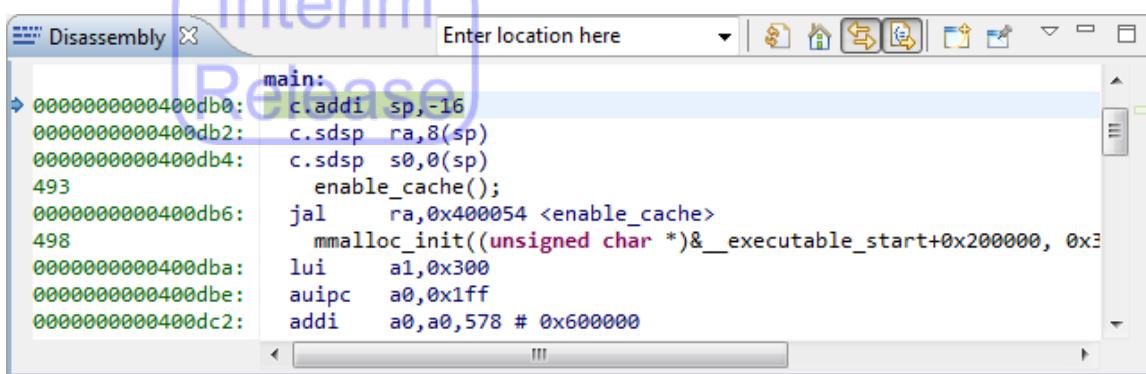
#### 2.4.2.13 Breakpoints view

The **Breakpoints** view lists all of the breakpoints you have set for your project. You may also add, cancel, enable, or disable breakpoints in this view. The figure below presents the **Breakpoints** view, indicating where breakpoints are set in the source code of a JPEG demo.



#### 2.4.2.14 Disassembly view

The **Disassembly** view shows high-level source code with its associated assembler code for use in identifying instruction issues during debugging. The figure below shows the **Disassembly** view from a debug session of the JPEG-V5 demo.



The screenshot shows the AndeSight IDE's Disassembly view. The window title is "Disassembly". At the top, there is a search bar labeled "Enter location here" and a toolbar with various icons. The main area displays assembly code for the "main" function. The code includes instructions like c.addi, c.sdsp, enable\_cache(), jal, mmalloc\_init, lui, auipc, and addi. The assembly code is color-coded, and the memory addresses are shown in green.

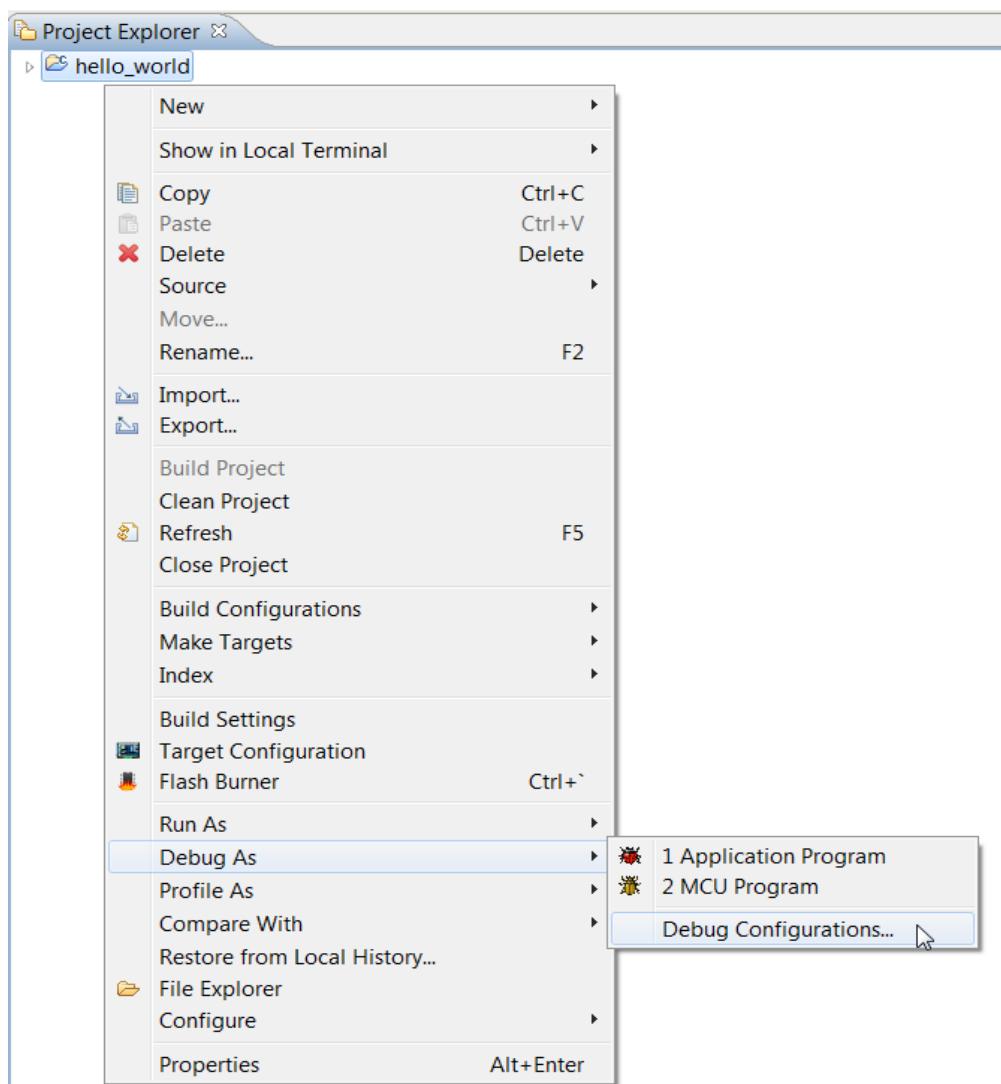
```
Disassembly X Enter location here
main:
0000000000400db0: c.addi sp,-16
0000000000400db2: c.sdsp ra,8(sp)
0000000000400db4: c.sdsp s0,0(sp)
493
enable_cache();
0000000000400db6: jal    ra,0x400054 <enable_cache>
498
mmalloc_init((unsigned char *)&_executable_start+0x200000, 0xE
0000000000400dba: lui    a1,0x300
0000000000400dbe: auipc a0,0x1ff
0000000000400dc2: addi   a0,a0,578 # 0x600000
```

### 2.4.3. Engaging a run/debug session

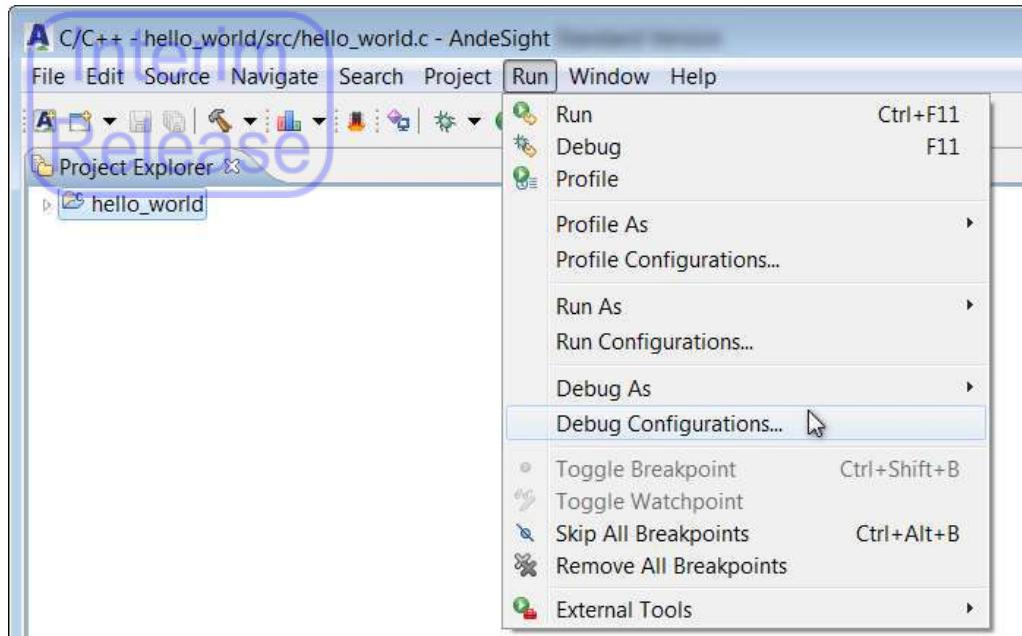
A run or debug session is launched with a set of configuration settings specifying the program executable, debugger behaviors, associated debugging tools, and other relevant information.

After creating a configuration for your project, a run/debug session can be launched simply by clicking  (Run) or  (Debug) on the AndeSight toolbar. To specify a configuration for your run/debug session or modify the settings, proceed as follows:

- Step 1** Right-click the project of interest in the **Project Explorer** view and select “[Run|Debug] as > [Run|Debug] Configurations...” on the pull-down menu to evoke the run/debug configurations.



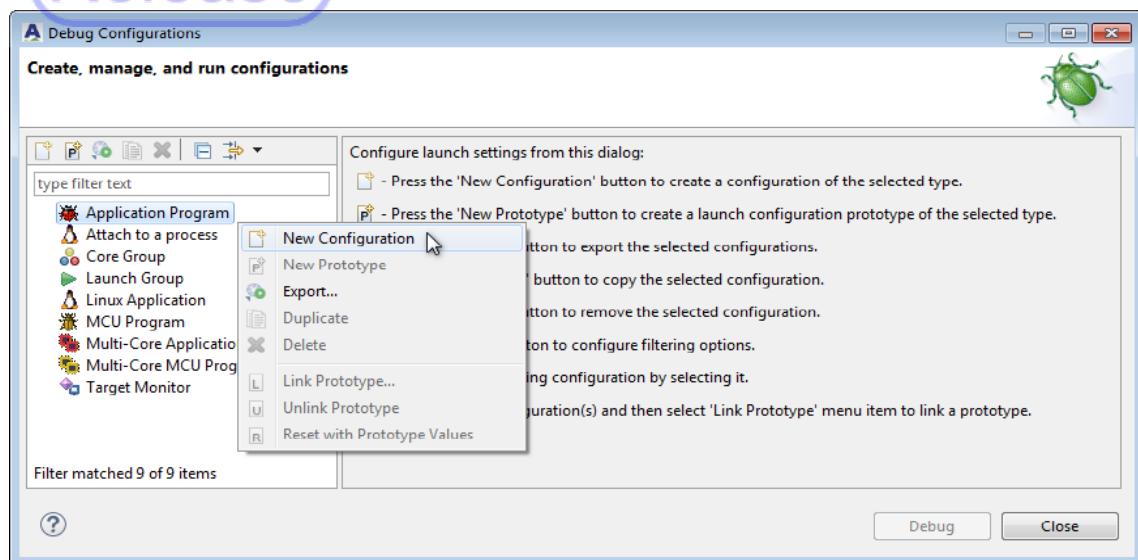
Or, you may evoke the run/debug configurations by selecting the desired project in the **Project Explorer** view and then selecting “Run > [Run|Debug] Configurations...” on the AndeSight main menu.



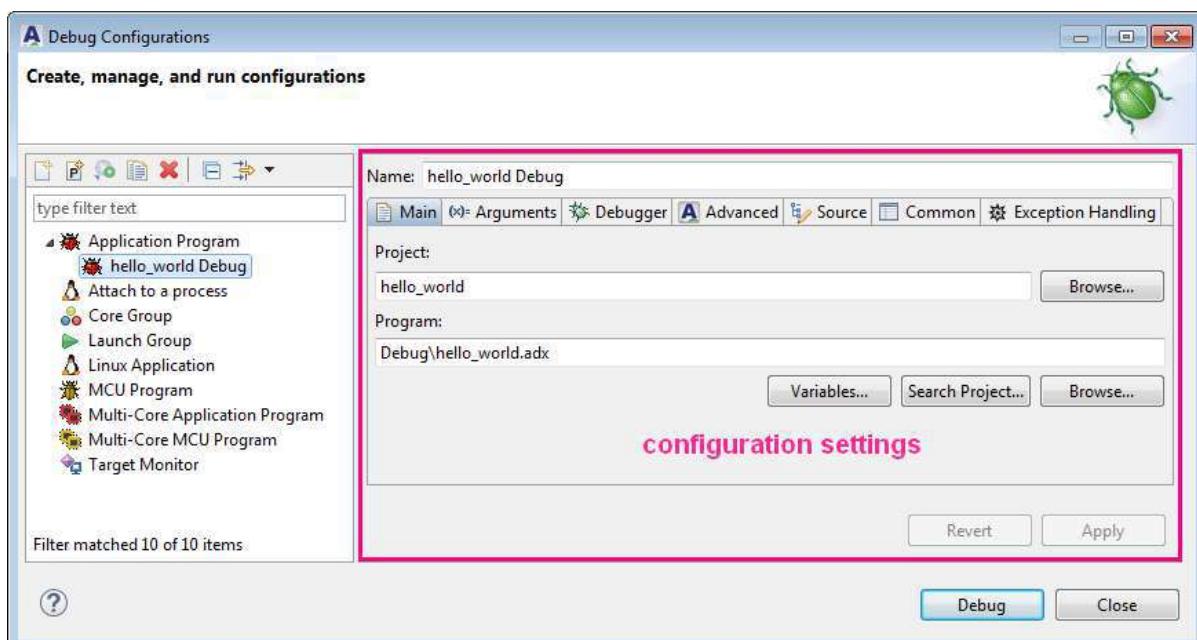
**Step 2** In the invoked **Run/Debug Configurations** dialog, reference the following to right-click a configuration type in the navigation pane and select “New Configuration” from the pull-down menu. A configuration is created for your debug/run session.

- (Regular) Application Program: for development on a target that provides system service
- Multi-Core Application Program: for development on a multi-core target that is without SMP support and provides system service
- (Regular) MCU Program: for development on a target that does not provide system service
- Multi-Core MCU Program: for development on a multi-core target that is without SMP support and does not provide system service
- Target Monitor: for checking the default state of a target
- Linux Application: for development on a target with TCP/IP network capability and a Linux kernel running on it

- Attach to Process: for debugging processes running on a Linux application
- Launch Group: for launching multiple debug configurations at the same time
- Core Group: for launching multiple debug configurations at the same time and send debug commands simultaneously to grouped cores.

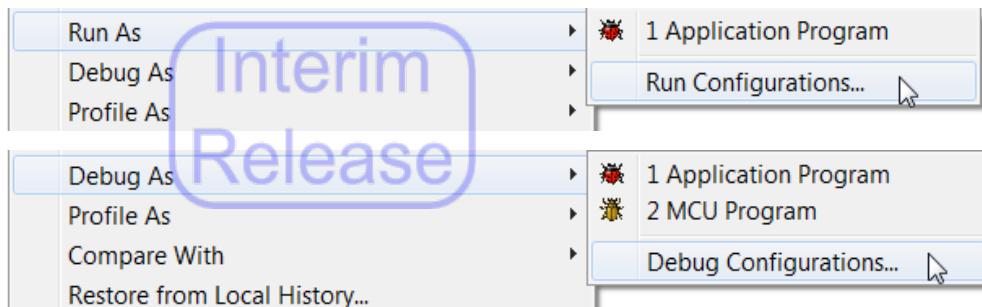


**Step 3** Work through the configuration settings in each tab and then click “Apply” and “[Run|Debug]” to launch the run/debug session.



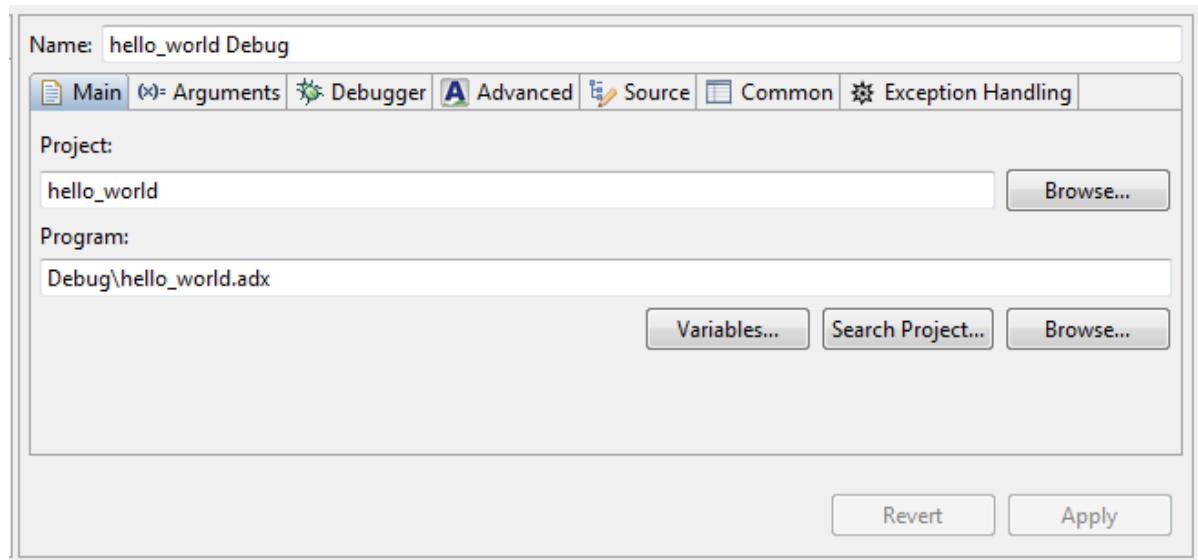
#### 2.4.3.1 Configuration settings for run/debug sessions

To invoke the **Run/Debug Configurations** dialog, right-click the project folder and select “[Run|Debug] As > [Run|Debug] Configurations ...” from the pull-down menu.



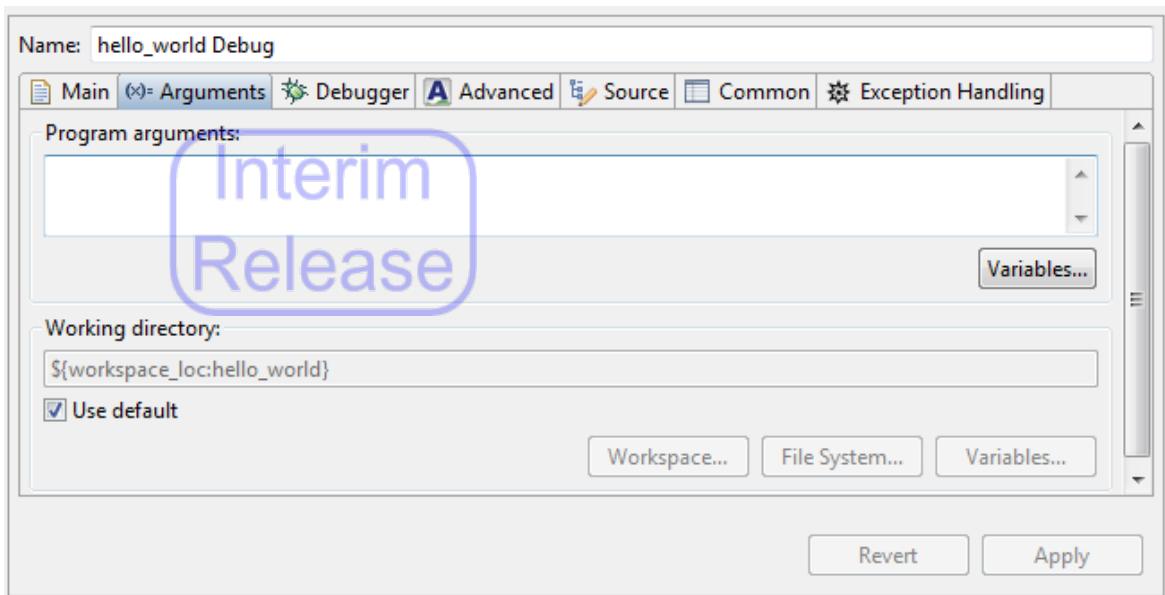
The **Run/Debug Configurations** dialog includes tabs in which to configure settings for a run/debug session. The tabs and options within the tabs vary with the type of configuration. The figures below show the tabs for the configuration type “Application Program”. Many of these tabs are also used for other configuration types, with same or variant options.

#### Main tab



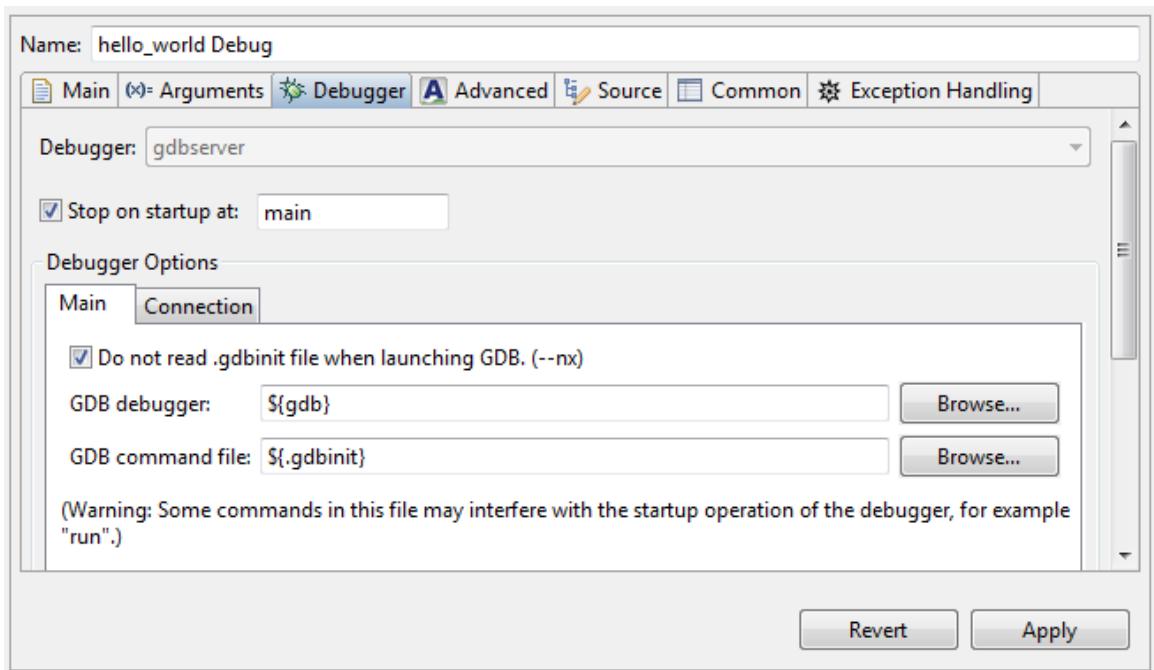
- Project: Select a project for the run/debug configuration.
- Program: Specify the path of the program executable (\*.adx).

## Arguments tab



- Program arguments: Set program arguments.
- Working directory: Select a workspace for the program.

## Debugger tab



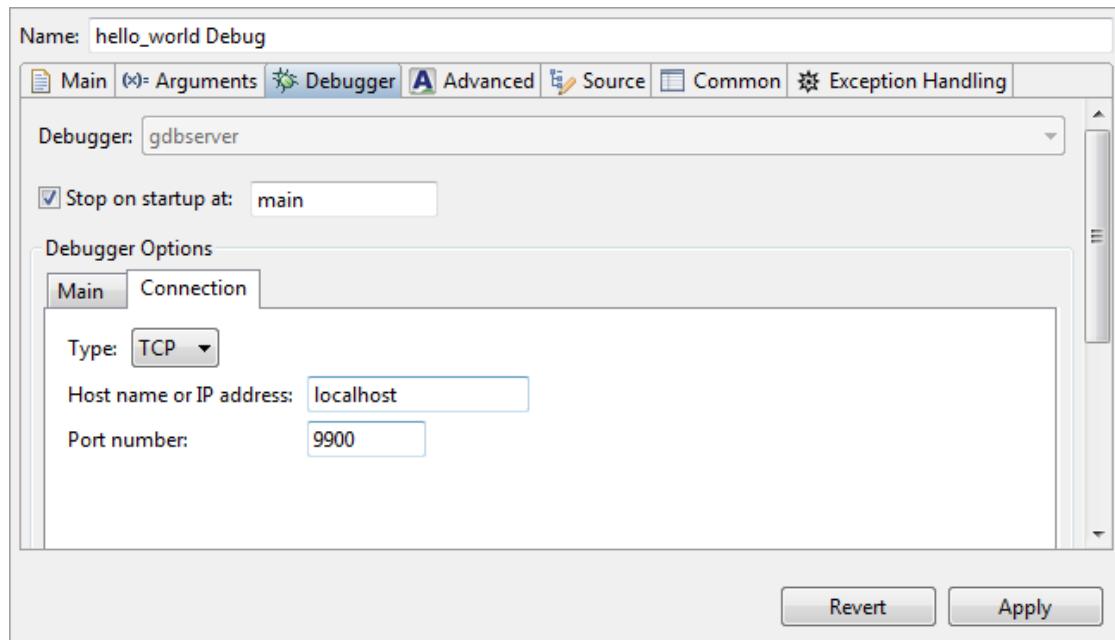
- Stop on startup at: Select this option and specify a function for the debugger to set a temporary breakpoint.

**NOTE**

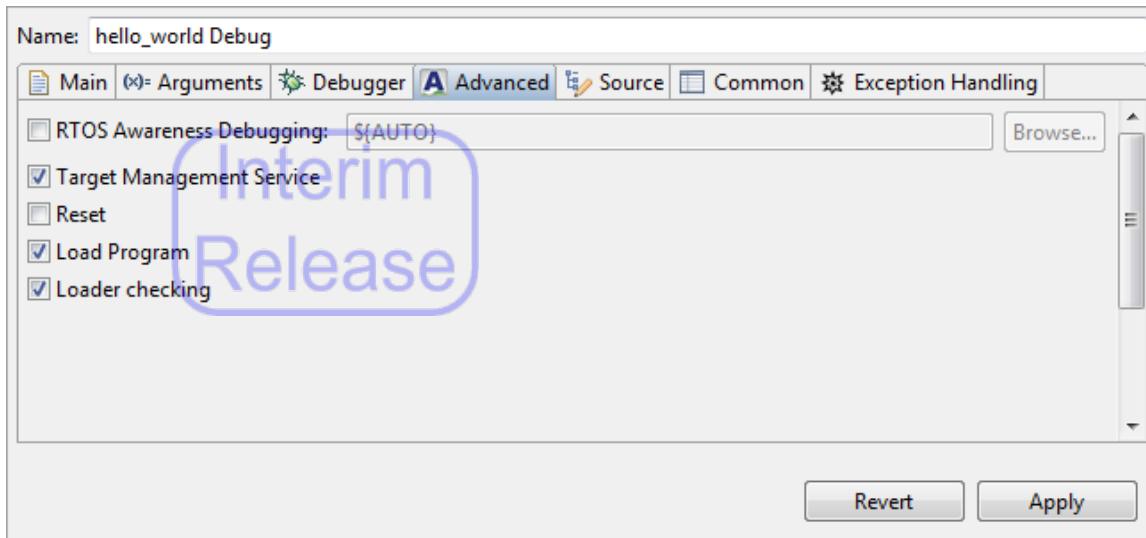
This setting will affect the number of available hardware breakpoints.

## ■ Debugger options

- Main: Specify the default GDB debugger binary and command file used in the run/debug session. You may change the GDB tools, albeit at the risk of interfering with startup operations.
- Connection: Specify the settings for the target connection if “Target Management Service” is disabled (i.e., unchecked). The “Target Management Service” option is in the **Advanced** tab of regular and Multi-Core Application Program configurations; for regular and Multi-Core MCU Program configurations, this option is in the **Main** tab.
  - ◆ Host name or IP address: Enter the name or IP address of the remote host to which your target will be connected.
  - ◆ Port number: Enter the port number to which GDB will connect. Port information will be displayed with the launched target in “Target Manager > Running Target”.



## Advanced tab



- **RTOS Awareness Debugging:** Select this option to create tables displaying RTOS information. This feature auto-detects the version of the kernel from symbols defined in the project source code. You can also click “Browse...” to manually specify a Python script that defines GDB commands.

---

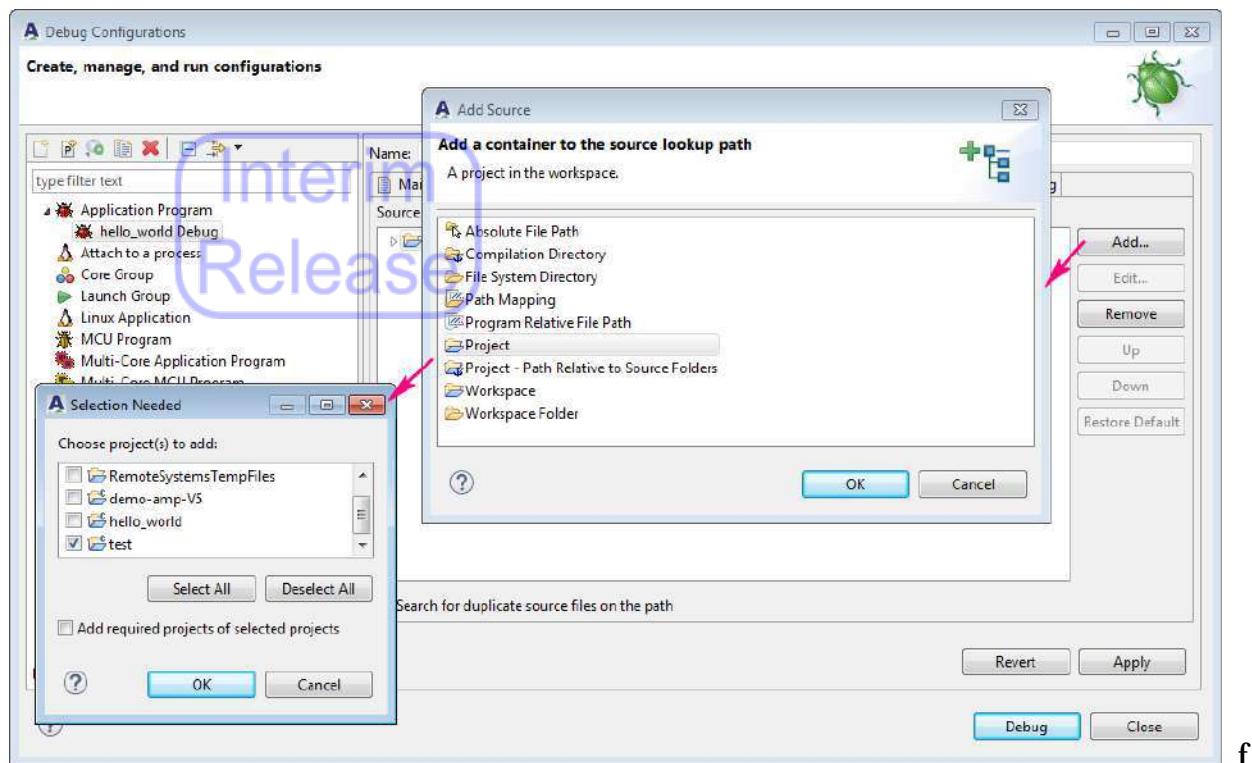
### NOTE

RTOS Awareness auto-detection may malfunction if `--gc-sections` is applied.

---

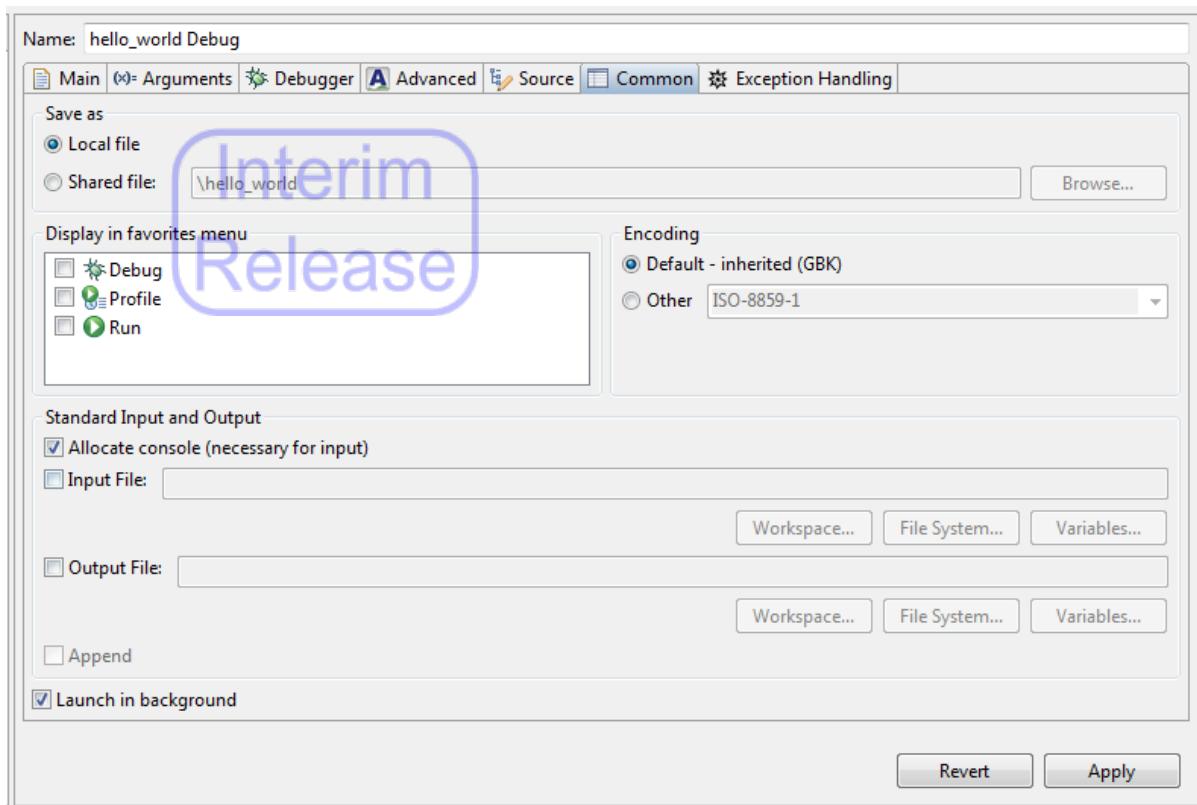
- **Target Management Service:** Select this option to enable the Target Manager to take care of target management. You may also uncheck the option and specify the target connection setting on your own (from “Debugger tab > Debugger Options - Connection”).
- **Reset:** Select this option to reset the target board every time a debug session is launched.
- **Load Program:** Select this option to load the project executable file “\*.adx” to the target board.
- **Loader Checking:** Select this option to verify that the project executable is compatible with the target configuration. An error message pops up if incompatibility is detected.

## Source tab



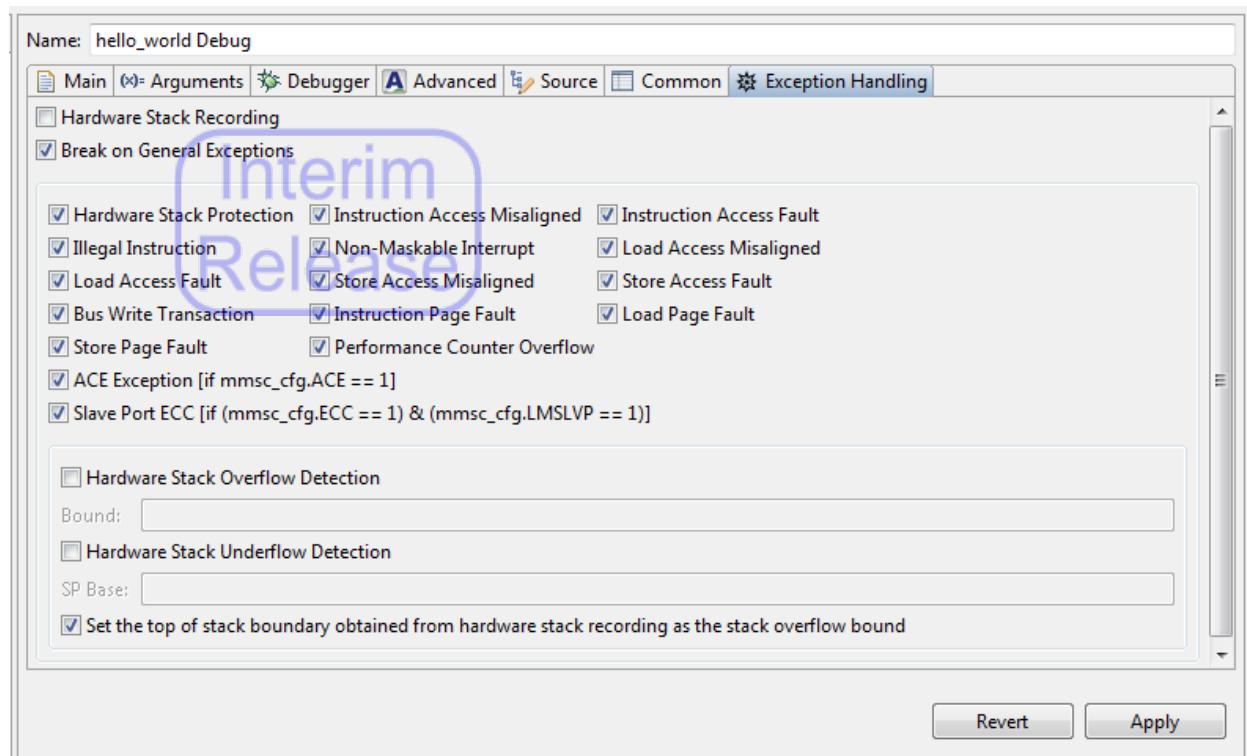
This tab allows you to specify the source files used for debugging a C or C++ application. For example, you can specify a project that contains a shared library and add it to the lookup path.

## Common tab



- **Save as:** Save your launch configurations as files. To generate a launch configuration file (\*.launch) for later use or for sharing with team members, check the “Shared file:” radio box and click “Browse...” to specify a location in which the file is to be generated.
- **Standard Input and Output:**
  - **Allocate console (necessary for input):** Set the **Console** view (see Section 2.4.2.5) as the prompt for inputs or outputs.
  - **Input File:** Assign stdin from a project resource (via the button “Workspace...”) or a file (via the button “File system...”). Likewise, you may click “Variables” to select and configure a variable for inputs.
  - **Output File:** Redirect outputs to a project resource (via the button “Workspace...”) or a file (via the button “File system...”). Likewise, you may click the button “Variables...” to select and configure a variable for outputs.
  - **Append:** Append output data from each launch to the selected output file. If this option is unchecked, new output data will overwrite previous output data.

## Exception Handling tab



This tab allows you to enable hardware stack recording and specify types of exception that must be caught during program execution. For further details, please refer to Section 2.5. Note that the two options “Hardware Stack Recording” and “Hardware stack protection” cannot be set at the same time.

## 2.4.4. Breakpoints and watchpoints

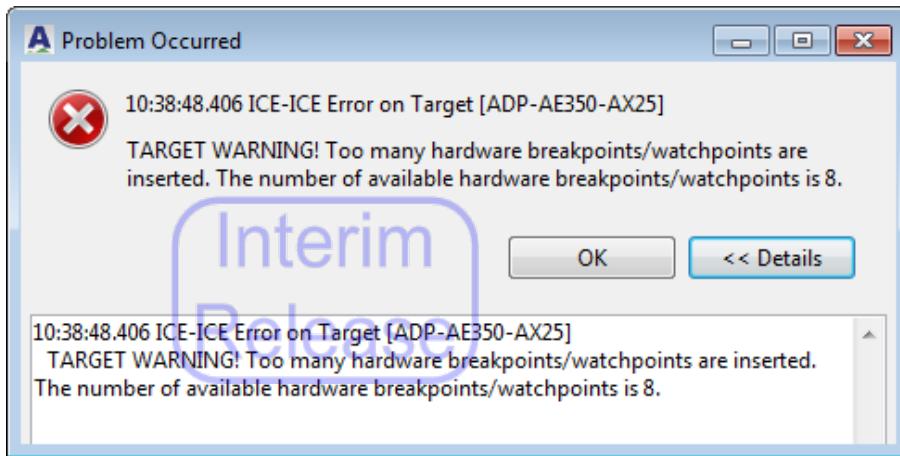
The number of watchpoints that can be set depends on the number of hardware breakpoints supported by the target. Please refer to the target specifications to determine the number of hardware breakpoints or watchpoints that are supported.

### 2.4.4.1 Setting breakpoints

Please take note of the following before setting breakpoints for your programs:

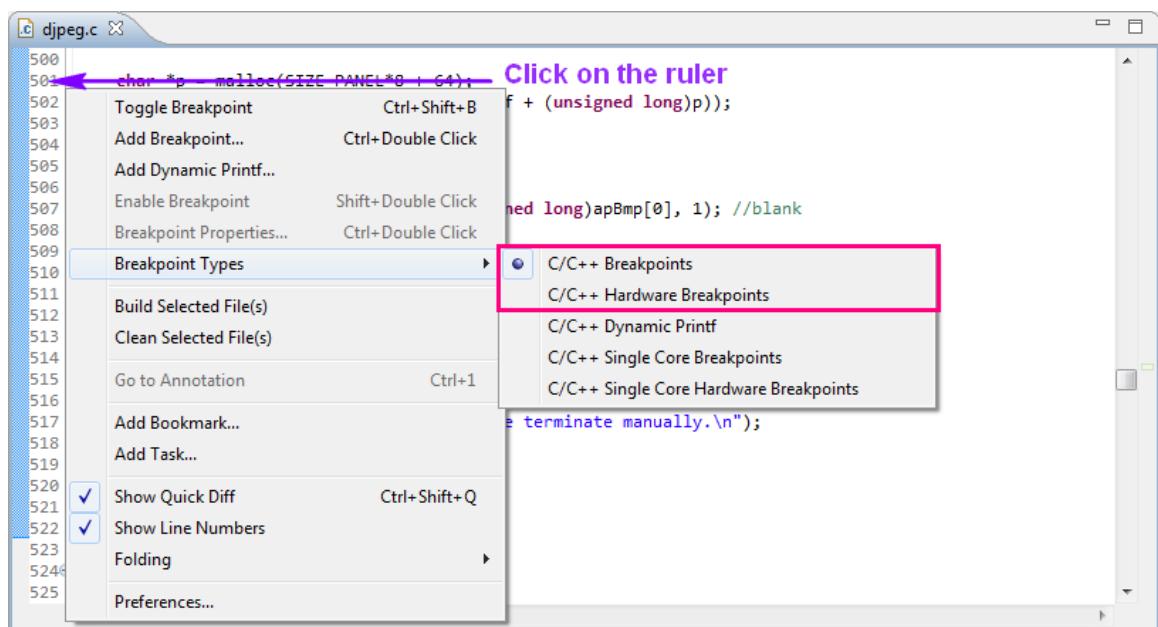
- If a for-loop is in the code lines for further examination, then set the breakpoint on the line of the first statement inside the for-loop.
- If programs are debugged on ROM, then software breakpoints will be automatically converted to hardware breakpoints and the debugger will use hardware breakpoints during Step In, Step Over, or Run to Line.
- When programs are debugged on ROM in a regular/Multi-Core Application Program configuration, the debugger will by default use one hardware breakpoint temporarily. To prevent this, go to “Debug Configurations Dialog > Debugger tab” and uncheck the option “Stop on startup at:” before debugging. Or, launch a regular/Multi-Core MCU program debug session (see Section 2.4.7.1) for the program instead.
- The available hardware breakpoints for use in a debug session are affected by breakpoint settings in the debug configuration, as follows:
  - Regular/Multi-Core Application Program debug configuration > Debugger tab > the option “Stop on startup at:” (see Debugger tab in Section 2.4.3.1).
  - Regular/Multi-Core MCU Program debug configuration > Startup tab > Runtime Options section > the option “Set breakpoints at:” (see Step 2 of Section 2.4.7.2).

As shown in the figure below, a warning message will pop up if the number of inserted hardware breakpoints exceeds the hardware limit.

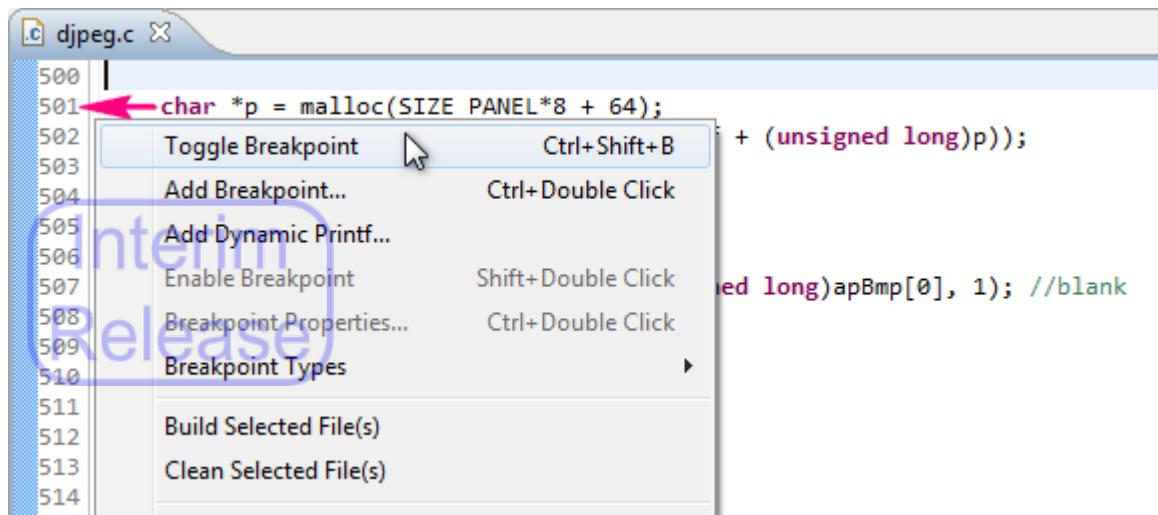


Then, proceed with the following procedure to set breakpoints:

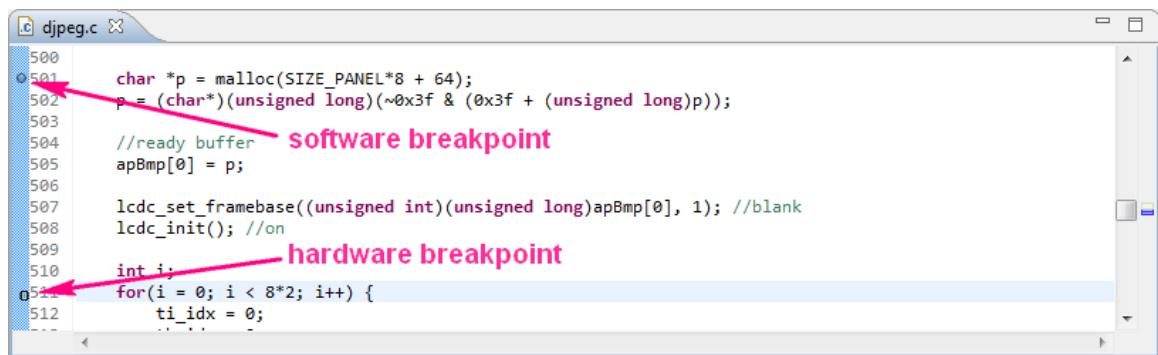
- Step 1** Open a source file in a code editor.
- Step 2** Click the ruler at the left margin of the code editor to invoke a pull-down menu. Specify a desired breakpoint type by selecting “Breakpoint Types” and choosing between “C/C++ Hardware Breakpoints” and “C/C++ Breakpoints” (software breakpoints).



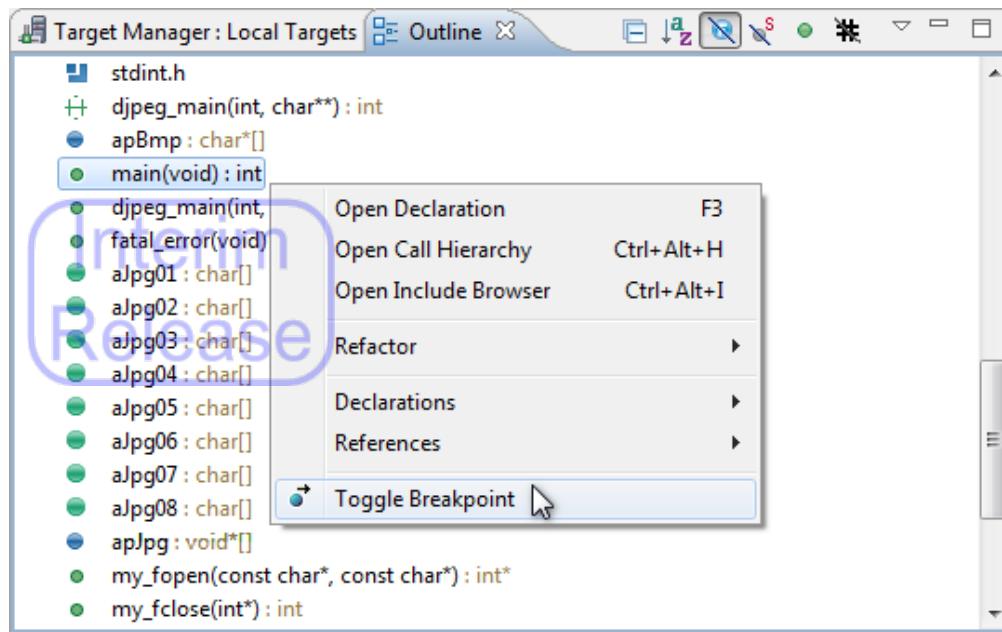
- Step 3** To the left of the line where you want to set a breakpoint, double-click the ruler or right-click and select “Toggle Breakpoint” from the pull-down menu.



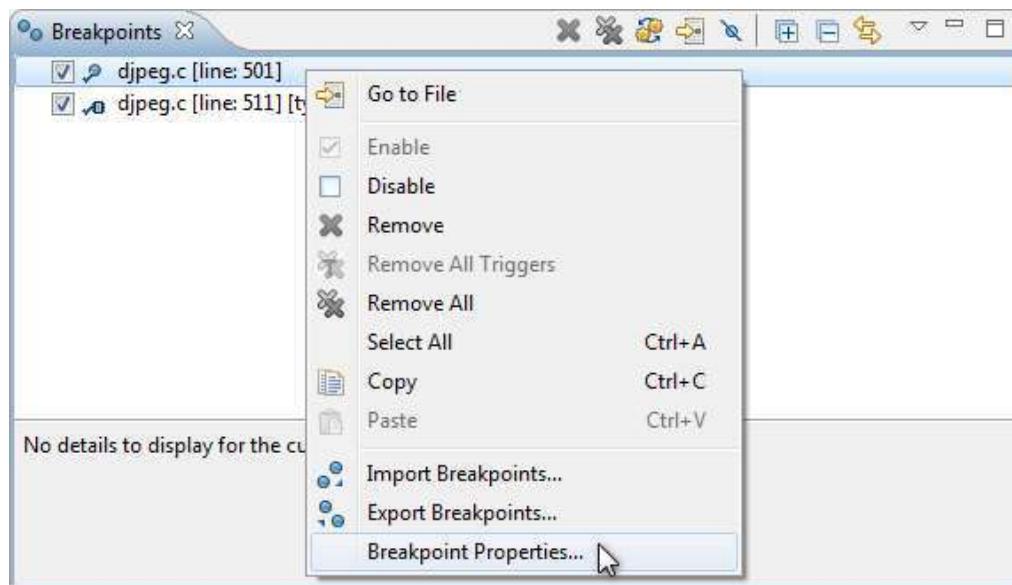
Breakpoint markers appearing on the ruler indicate where you want to suspend the program execution.



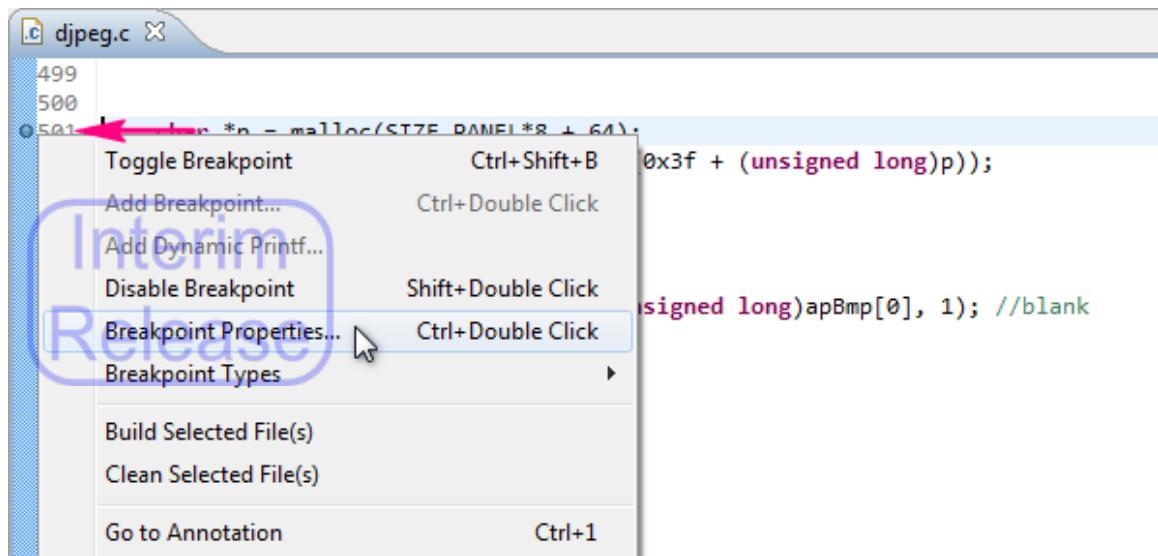
You may also set a function breakpoint in the **Outline** view by right-clicking a desired function in the view and selecting “Toggle Breakpoint” from the pull-down menu.



**Step 4** To specify the triggering condition for a breakpoint, right-click the desired breakpoint either in the **Breakpoints** view or on the ruler of the code editor and then select “Breakpoint Properties...” from the pull-down menu.

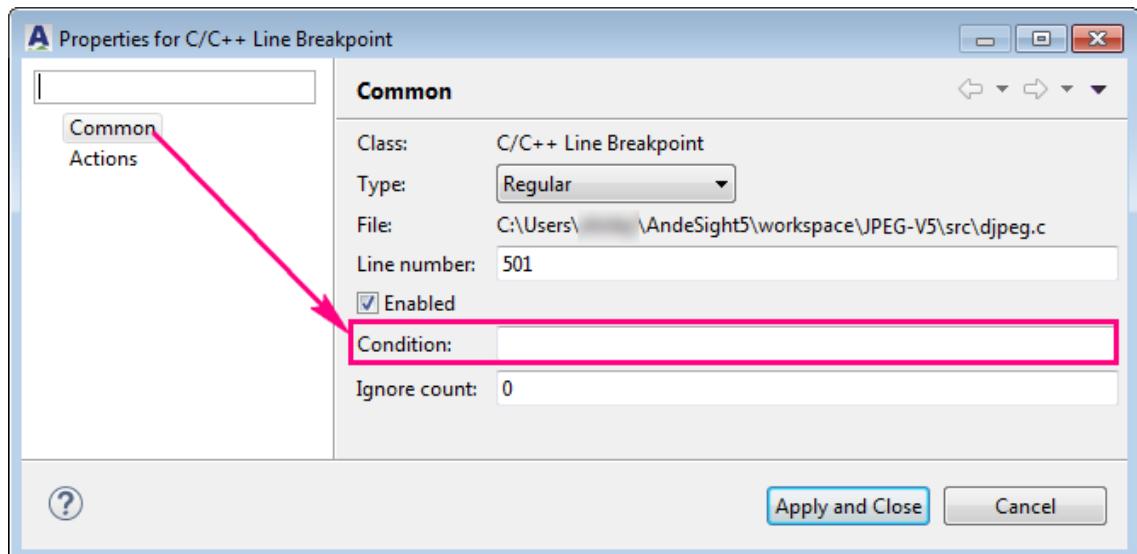


(select “Breakpoint Properties...” from the **Breakpoints** view)



(select “Breakpoint Properties...” from a code editor)

In the invoked dialog, select “Common” in the navigation pane and enter the conditional expression in the Condition field. Then, click “OK.”



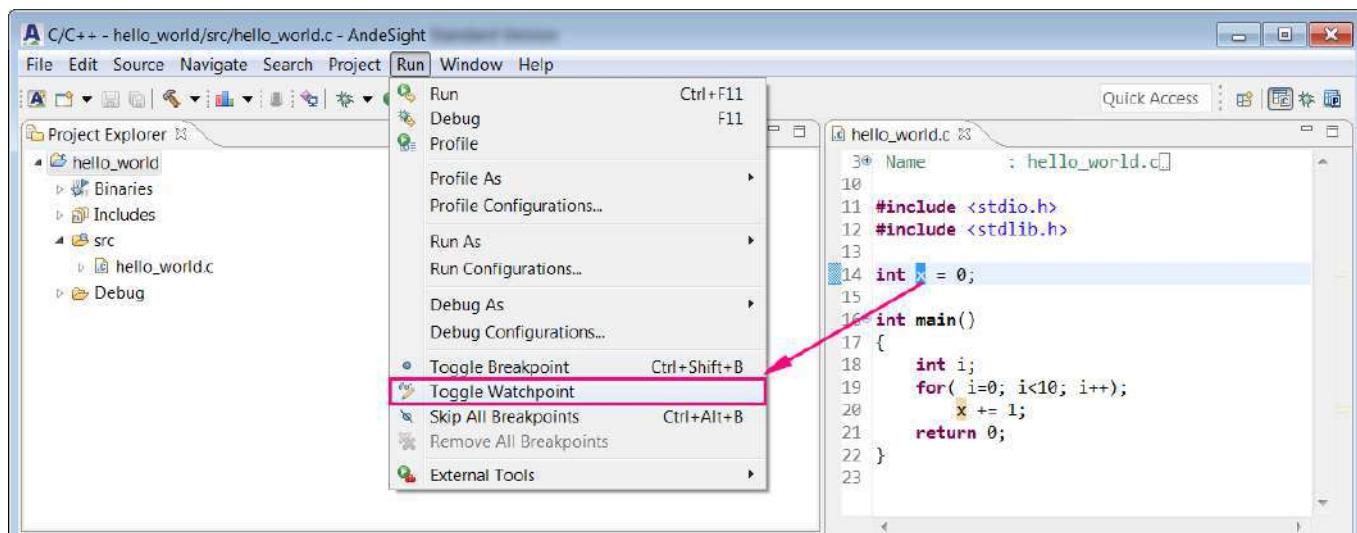
**Step 5** To clear a breakpoint, simply double-click its breakpoint marker in the code editor.

#### 2.4.4.2 Setting watchpoints

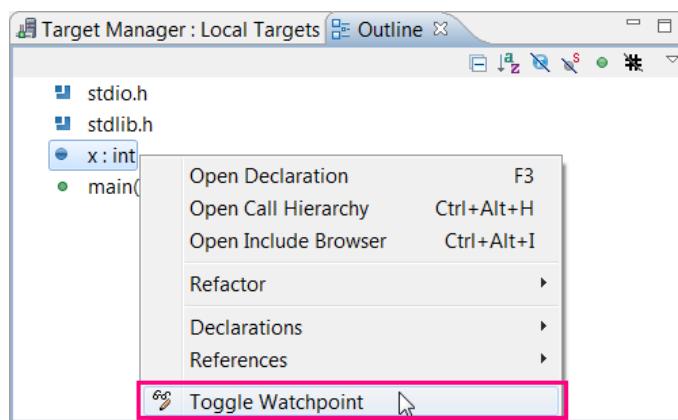
Unlike breakpoints, which are line-specific, a watchpoint stops the program execution only in the event that the value of a variable or expression changes – regardless of when or where it occurs. To set a watchpoint for an application on AndeSight, follow the procedure below:

**Step 1** You can set a watchpoint from one of the following entries:

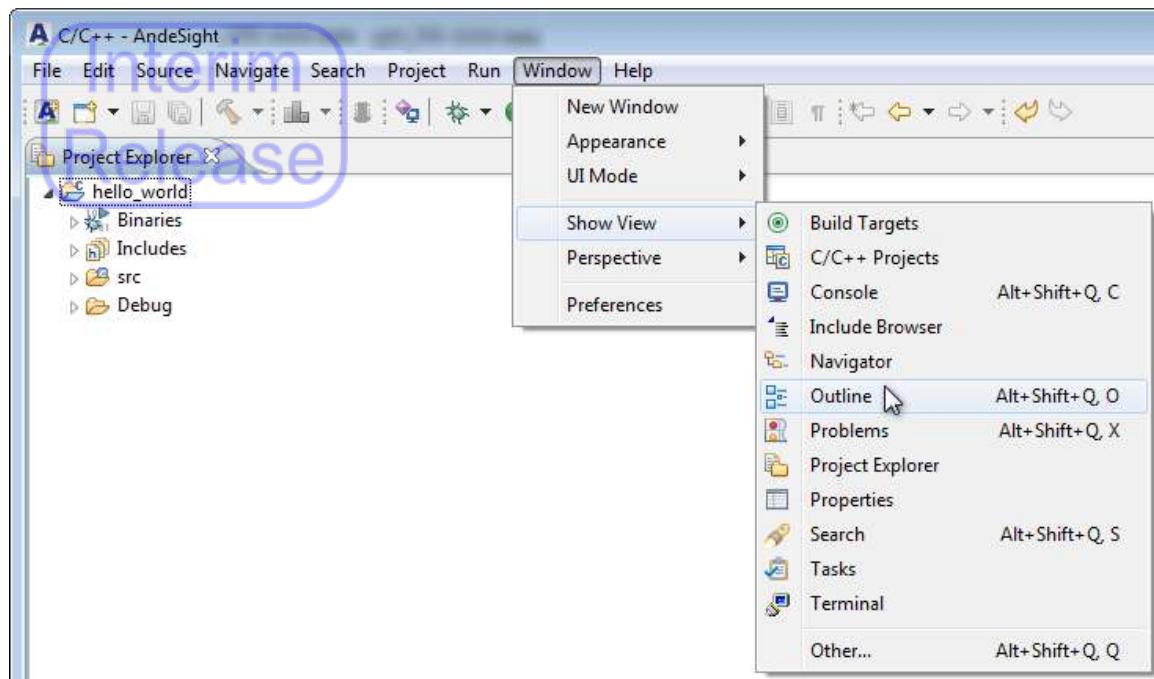
- From a code editor: Select the global variable of interest in a code editor and select “Run > Toggle Watchpoint” from the AndeSight main menu.



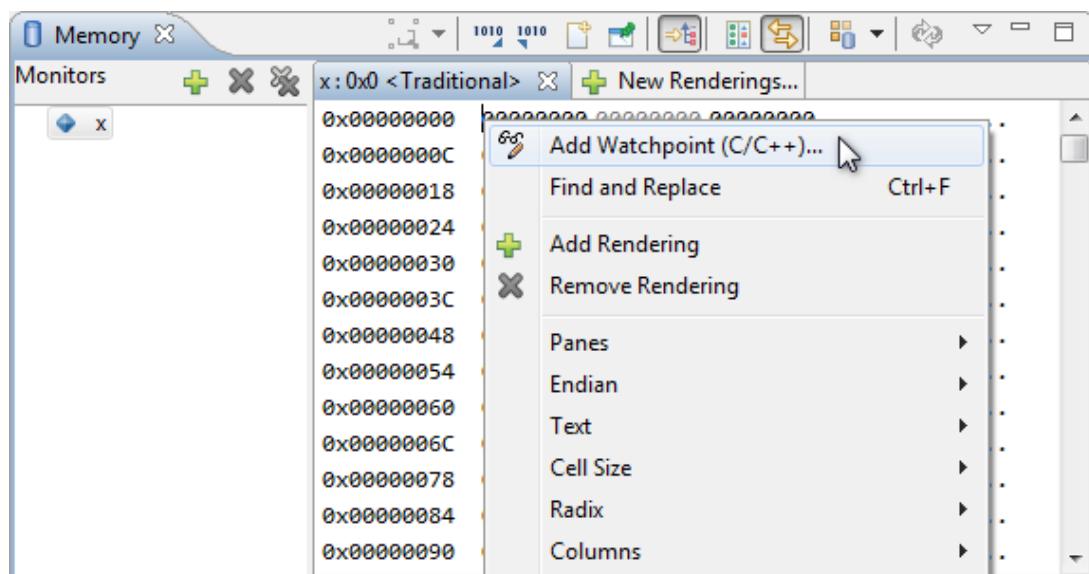
- From the Outline view: In the Outline view, select a global variable of interest, right-click and select “Toggle Watchpoint” from the pull-down menu.



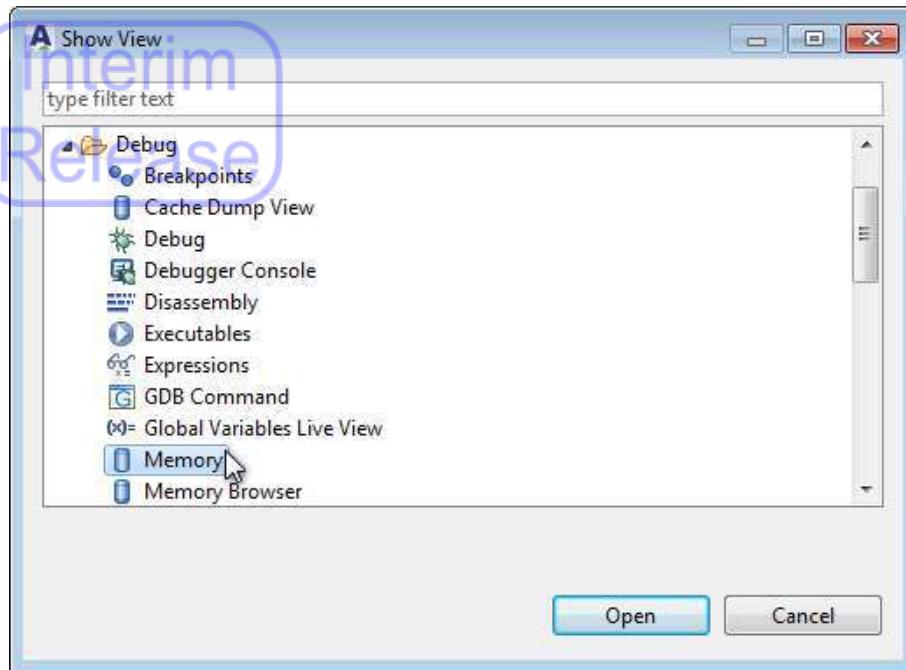
The **Outline** view is opened automatically in a debug session. You can also invoke it by selecting “Window > Show View > Outline” from the AndeSight main menu.



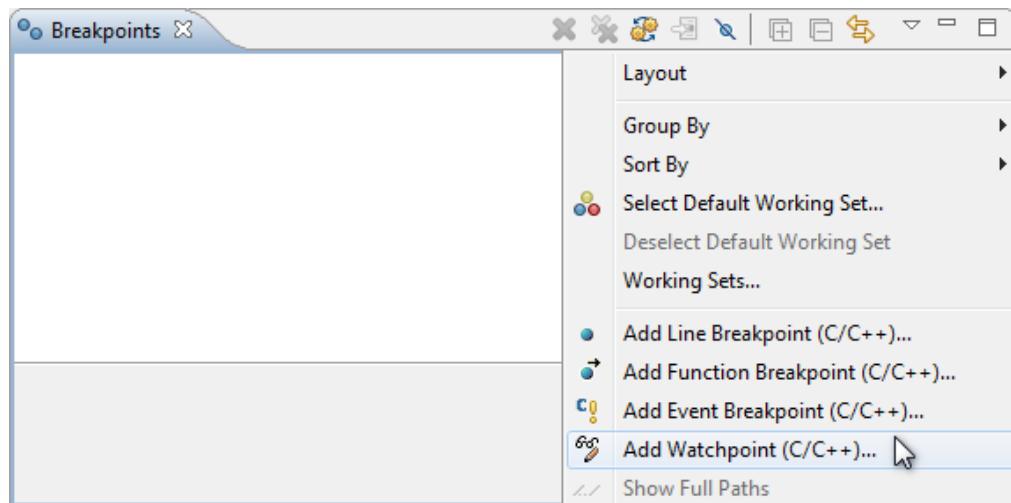
- From the **Memory** view: In the **Memory** view, select a desired addressable unit or range of units on the memory rendering, right-click and select “Add Watchpoint (C/C++)...” from the pull-down menu.



The **Memory** view can be invoked by selecting “Window > Show View > Others...” from the AndeSight main menu and then “Debug > Memory” in the **Show View** dialog.

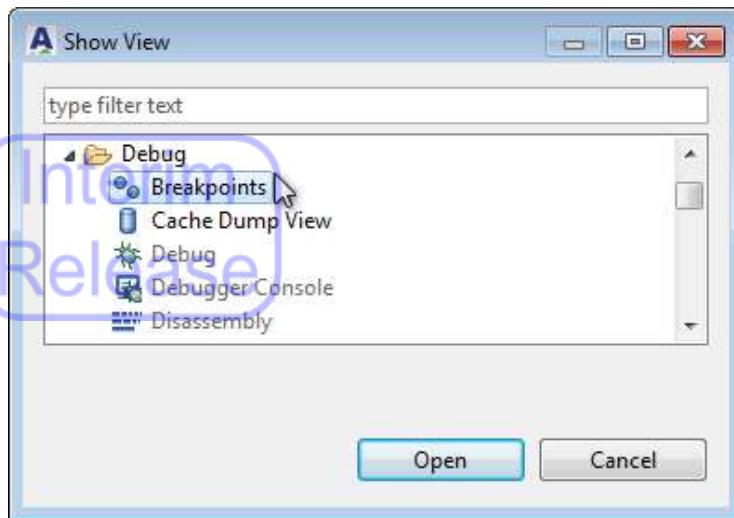


- From the **Breakpoints** view: On the toolbar of the **Breakpoints** view, click  (View Menu) and select “Add Watchpoint (C/C++)...” from the pull-down menu.



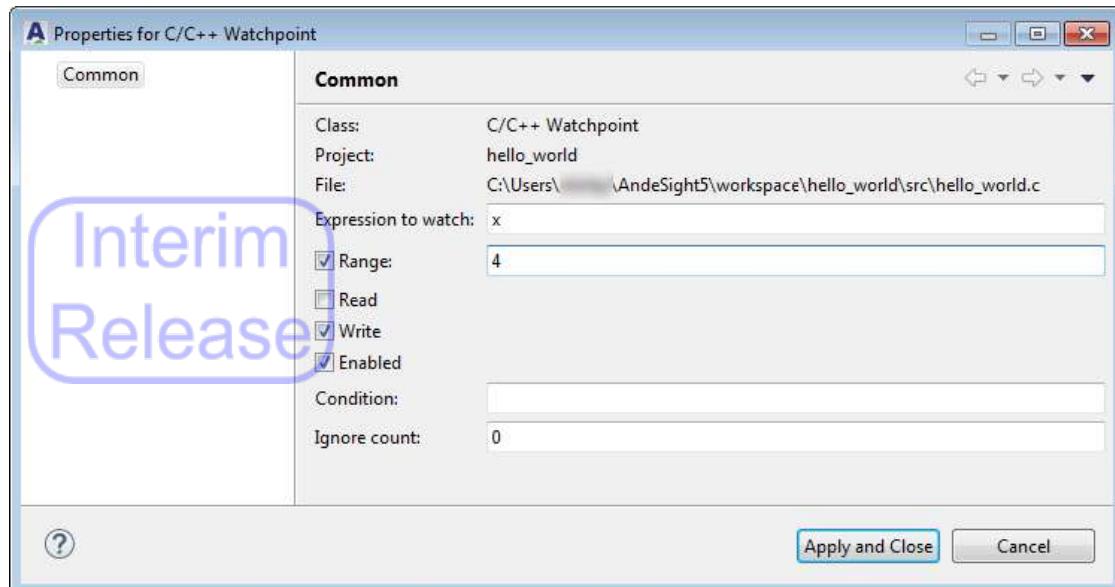
The **Breakpoints** view can be invoked by selecting “Window > Show View > Others...” from the AndeSight main menu and then “Debug >

Breakpoints” in the **Show View** dialog.

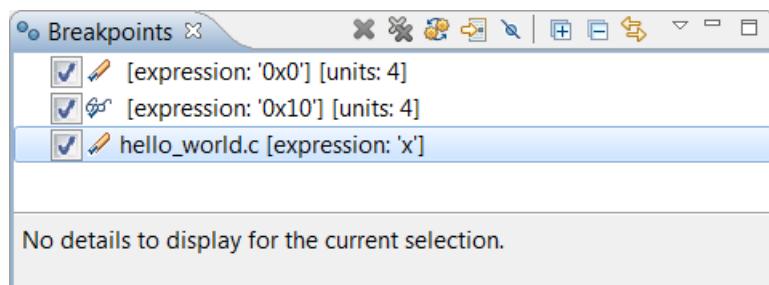


**Step 2** Specify the settings in the **Properties** dialog and click “Apply and Close”:

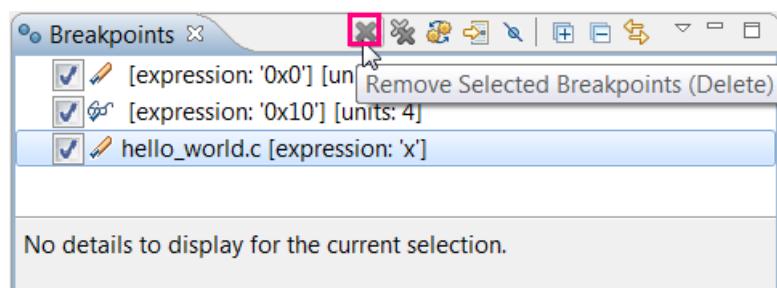
- Expression to watch: Enter a constant address or the address of a variable (&i) here if you are going to specify the watched region in the “range” option below.
- Range: Select this option to specify the addressable units monitored by a watchpoint.
- Read: Select this option to stop the program execution when the watched expression is read.
- Write: Select this option to stop the program execution when the watched expression is written.
- Enabled: Select this option to enable the watchpoint.
- Condition: Specify an expression to be evaluated when the watchpoint is hit.
- Ignore count: If you want the program to suspend after the value of the watched expression has changed a certain number of times, specify the number of times here. The ignore count must be an integer and its default value is zero.



**Step 3** The watchpoints you set are listed in the **Breakpoints** view. The watchpoint marker  denotes write watchpoints and  denotes read watchpoints.

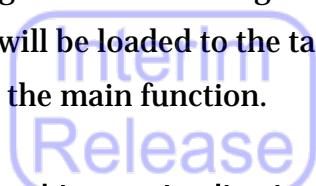


**Step 4** To clear a watchpoint, select it in the **Breakpoints** view and click  (Remove Selected Breakpoints) on the toolbar.



## 2.4.5. Application Program debugging

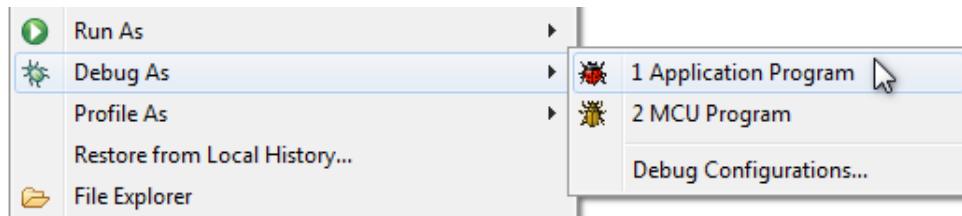
If you use a target system that provides system services such as BIOS or boot code, then select “Application Program” as the configuration type for your debug session. In the debug session, the applications will be loaded to the target’s RAM in ELF file format and stop at specified symbols, such as the main function.



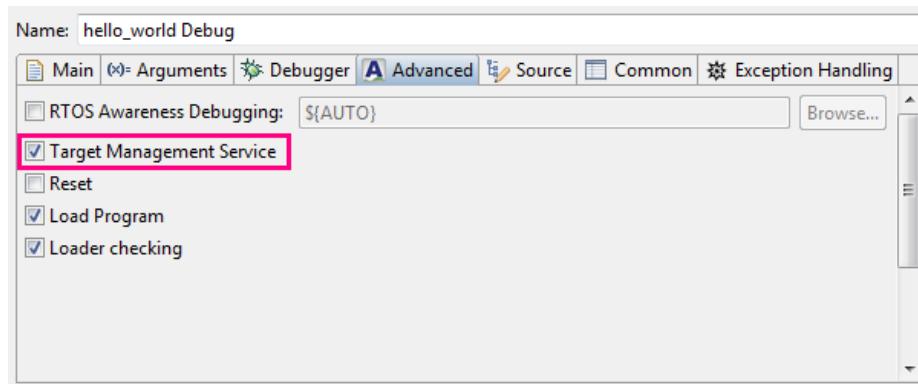
### 2.4.5.1 Launching an Application Program run/debug session

**Step 1** See Section 2.1 for instruction on creating and building a project.

In **Project Explorer**, right-click the project and select “[Run|Debug] As > Application Program” from the pull-down menu to run or debug the executable with an **Application Program** configuration.



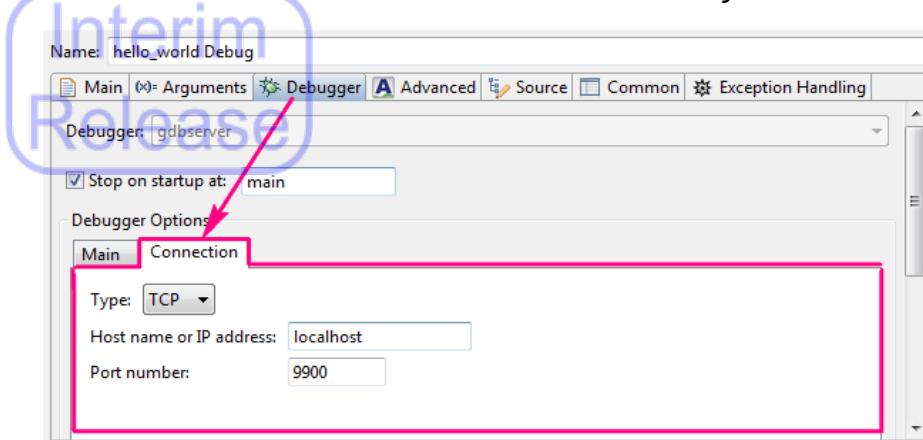
By default, the option “Target Management Service” is enabled to set the local target designated during project creation as the project target for the run/debug session and allow the **Target Manager** to manage the target.



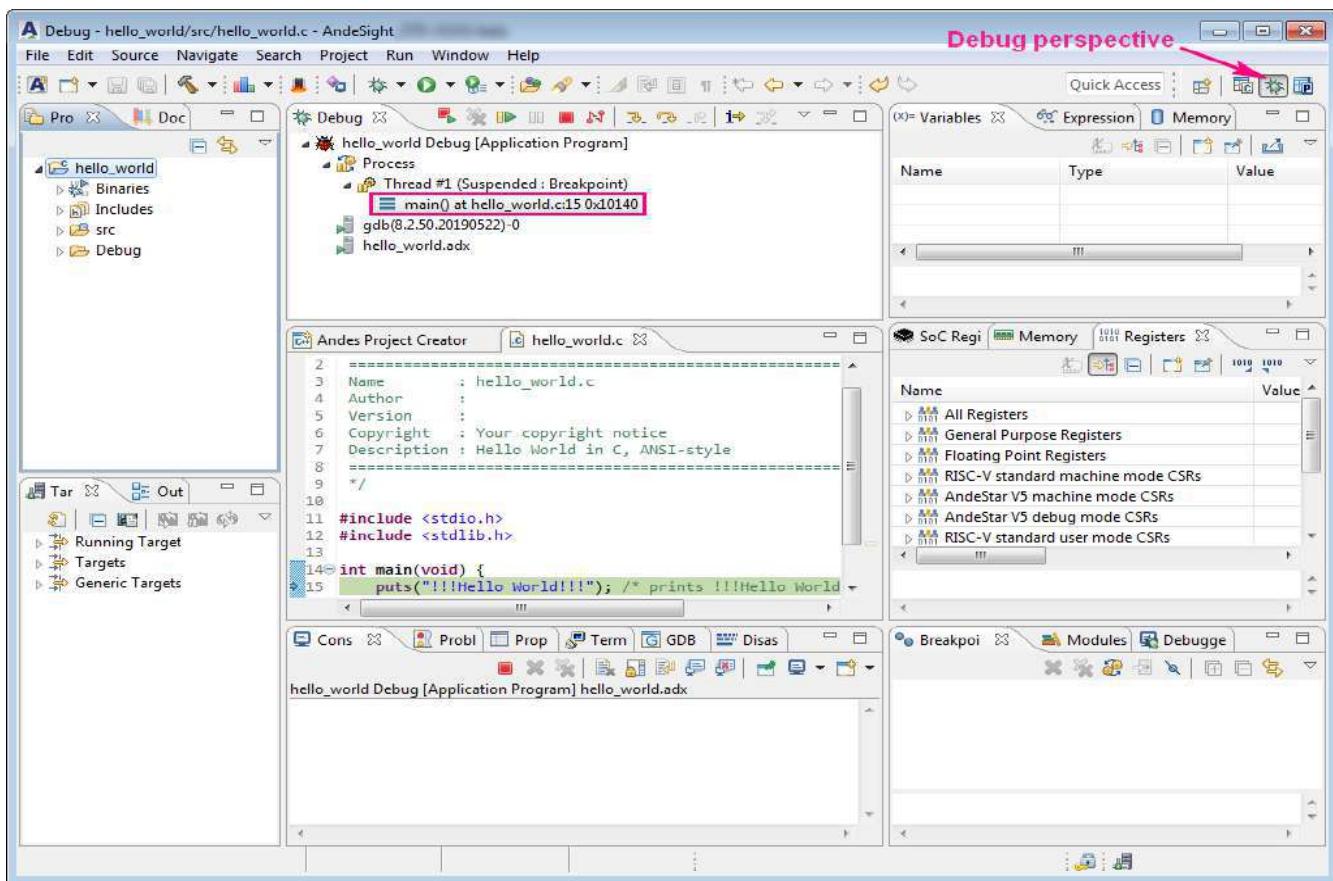
If the target designated for the project is not your desired target, then you may connect to a remote host as follows:

1. Invoke the **Run/Debug Configurations** dialog. Under the **Advanced** tab,

- uncheck the option “Target Management Service”.
2. Turn to “Debugger tab > Debugger Options - Connection” and specify the settings for target connection.
  3. Run the ICE device or simulator manually.



**Step 2** Click “Apply” and “Debug” to launch a debug session. The **Debug perspective** is invoked automatically. By default, the program is suspended at the main function.



**Step 3** In the **Debug** view, click  (Resume) on the toolbar to continue program execution and observe the results in other debug views (the **Variables** view, the **Registers** view ...).

Interim  
Release

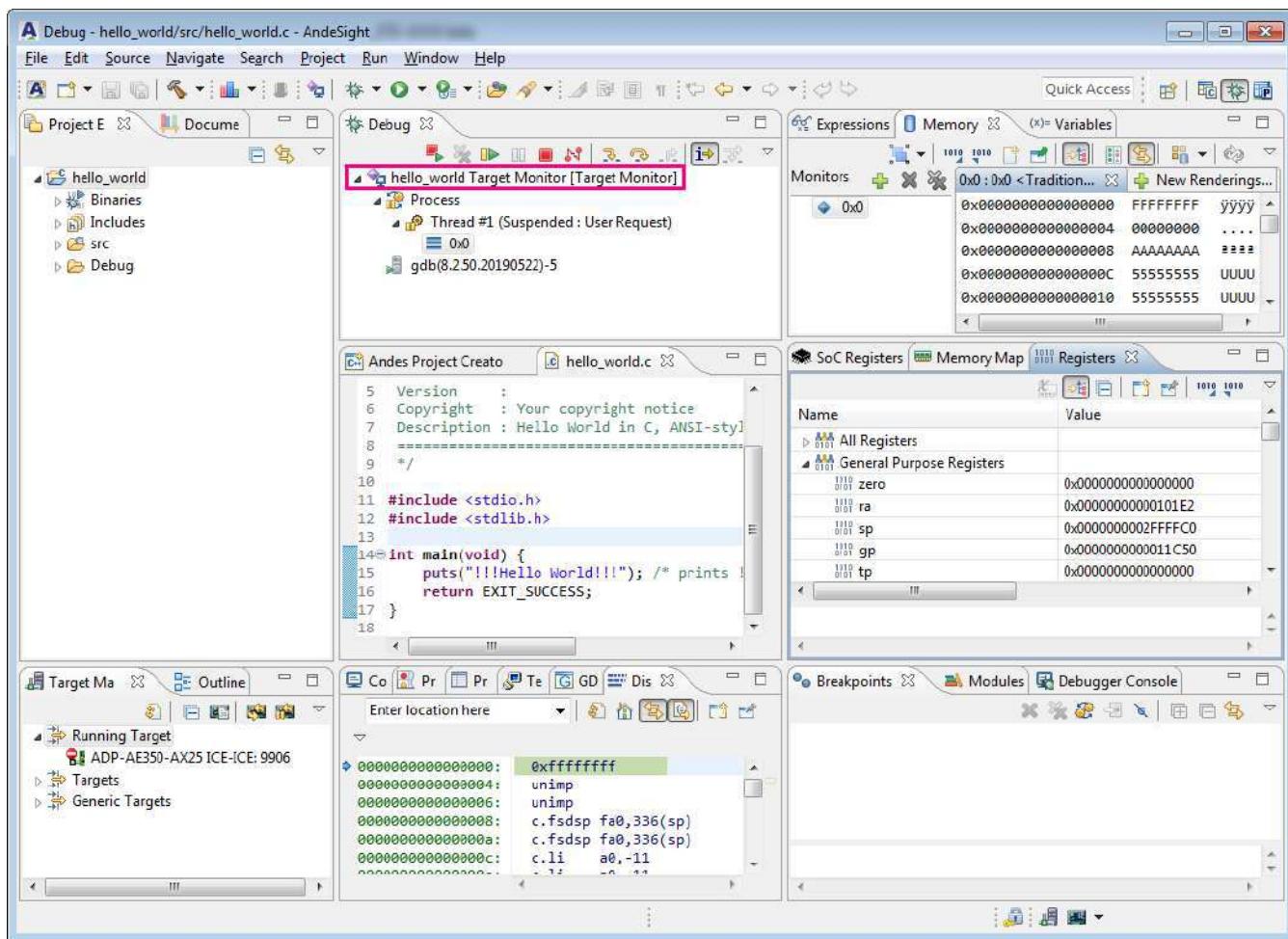
## 2.4.6. Target Monitor

Target Monitor sessions allow you to examine and modify the default state of your target system, such as registers or memory content, prior to commencing a debug session. This type of session is meant for targets connected to a host via an ICE device (ICE targets).

### 2.4.6.1 Launching a Target Monitor session

**Step 1** Follow Section 2.1 to create and build a project. Select the project in **Project Explorer** and click  (Monitor Target) on the AndeSight toolbar.

**Step 2** The **Target Monitor** session is launched in the **Debug** perspective. You may examine default system values in various debug views (the **Registers** view, the **SoC Registers** view...).



## 2.4.7. MCU Program debugging

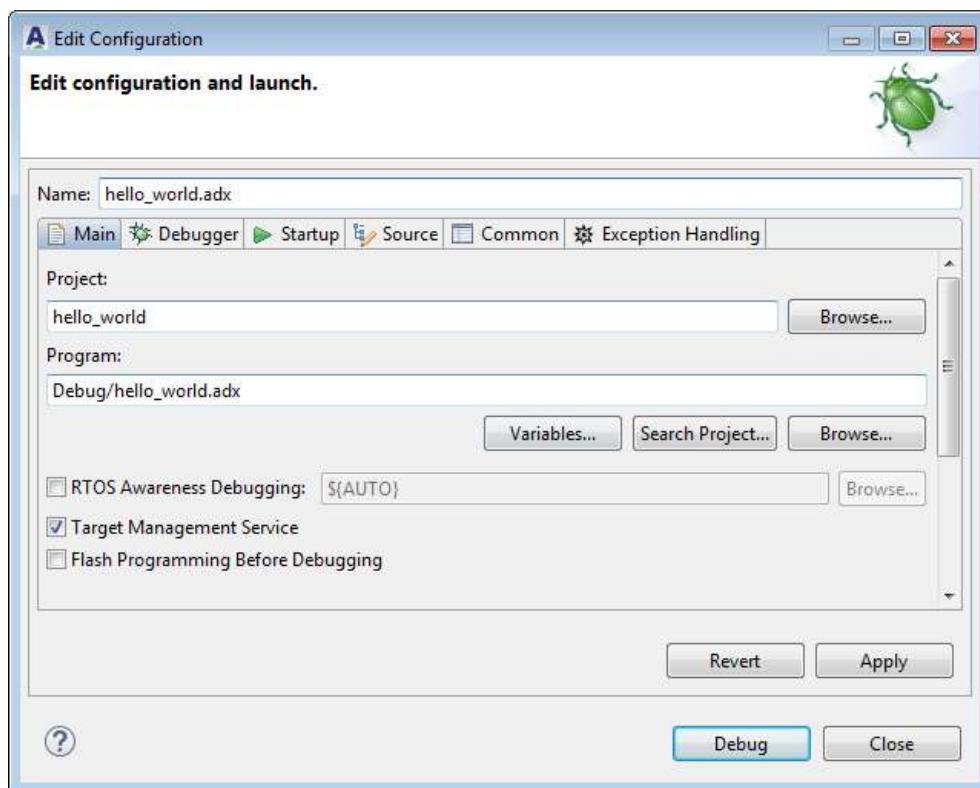
Select “MCU Program” as the configuration type for your debug session if a target system without system services is in use, such that the application handles everything from system boot to the application level. During the debug session, the MCU program (binary files; \*.BIN) in ROM will be debugged directly or loaded to RAM on the target.

### 2.4.7.1 Launching an MCU Program debug session

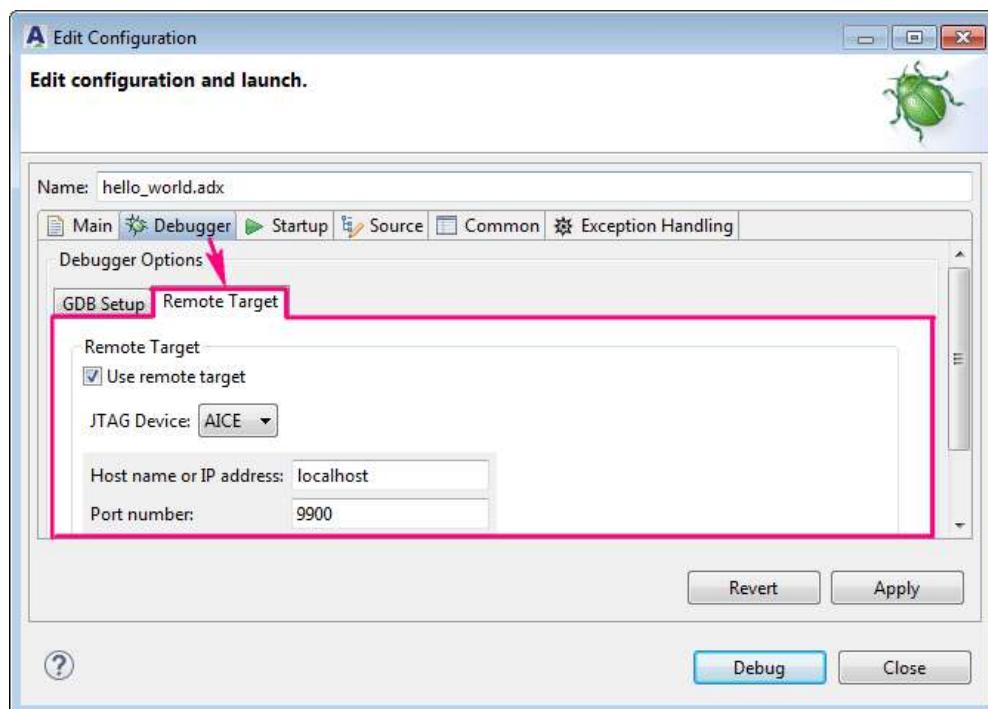
**Step 1** Follow Section 2.1 to create and build a project. In **Project Explorer**, right-click the project and select “Debug As > MCU Program” from the pull-down menu.



**Step 2** In the **Edit Configuration dialog**, proceed through the options under the **Main** tab.



- Project: Specify a project for the debug session.
- Program: Specify the boot code file with the symbol information.
- RTOS Awareness Debugging: Select this option to perform OS aware debugging.
- Target Management Service: Select this option to have the **Target Manager** take care of target management. If this option is unchecked, then configure the target connection settings in “Debugger tab > Remote Target.”



- Flash Programming Before Debugging: Select this option to perform in-system programming before MCU program debugging.

**Step 3** Click “Apply” and “Debug” to launch the MCU Program debug session.

#### 2.4.7.2 Reset-and-Hold configuration

In the MCU Program debug configuration, the Reset-and-Hold option is tailored for ICE targets. It enables the debugger to hold the CPU immediately after the target is reset. This feature is particularly useful in boot code development. The following outlines the launching of a debug session with the Reset-and-Hold configuration.

**Step 1** Follow Steps 1 and 2 in Section 2.4.7.1 to set up the MCU Program debug configuration.

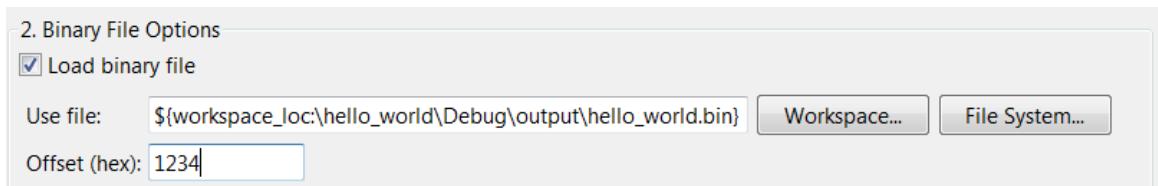
**Step 2** In the **Edit Configuration** dialog, click on the “Startup” tab and work through the options:

##### 1. GDB Initialization Commands



- **Reset-and-Hold:** Select this option to hold the CPU after target reset and set the program counter to 0x0.
- **Box:** Enter GDB commands for startup.

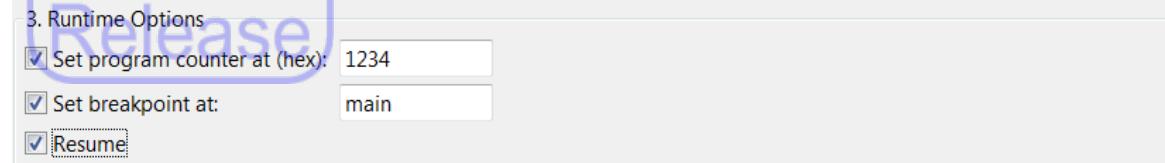
##### 2. Binary File Options



- **Load binary file:** Select this option if you want to load a file (\*.bin) to RAM at the beginning of a debug session.
- **Use file:** Click “Workspace” or “File System” to browse for the binary file to be loaded into RAM.

- Offset (hex): Specify a memory address to which the file is to be loaded. In the example above, the selected binary file “hello\_world.bin” is loaded to 0x1234.

### 3. Runtime Options (optional)



- Set program counter at (hex): Select this option to specify a memory address to which PC will move to begin debugging. In this example, the program counter moves from 0x0 to 0x1234 during runtime.
- Set breakpoint at: Select this option to enter a function name, line number, or memory address (\*address) as the breakpoint. In this example, the initial execution is suspended when PC hits the main function.

---

#### NOTE

The number of available hardware breakpoints is affected by this setting.

---

- Resume: Select this option to resume a halted execution.

### 4. GDB Run Commands (optional)



Specify GDB commands that run after the debug process is suspended in this box.

**Step 3** Click “Apply” and “Debug” to launch a debug session using the Reset-and-Hold configuration.

**NOTE**

For a project to be debugged or run on a SMP-capable target system, it is necessary to use the MCU Program configuration and have the following settings in the Startup tab:

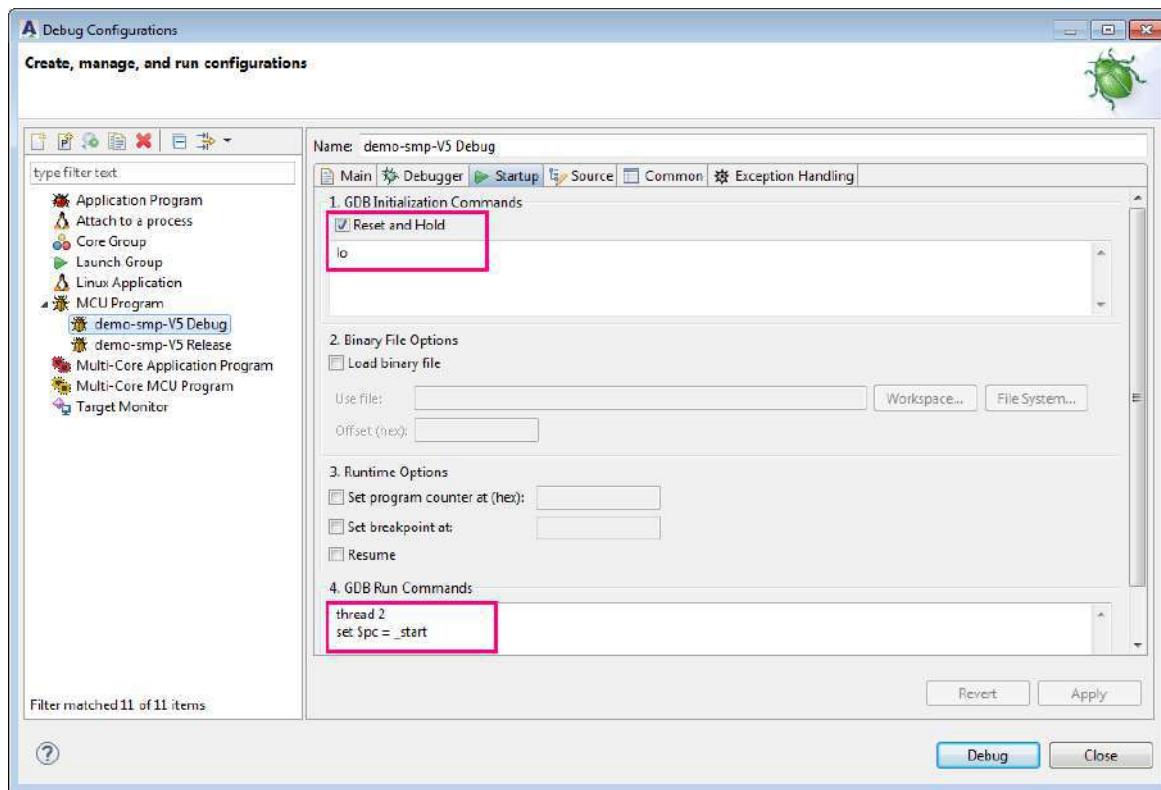
- In the GDB Initialization Commands section, check the “Reset and Hold” option and enter the command “`lo`”.
- In the GDB Run Commands section, enter the following commands to set the program counter of cores other than core 1 to the entry point of the ELF executable –

`thread N`

`set $pc=ELF_ENTRY`

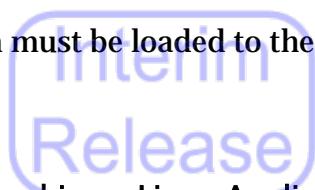
where the command “`thread`” is to switch control among cores.

The following example is the debug configuration of demo-smp-V5 for a dual-core SMP system. The program counter of core 2 is set to the `_start` symbol, which is the entry point of the program executable.



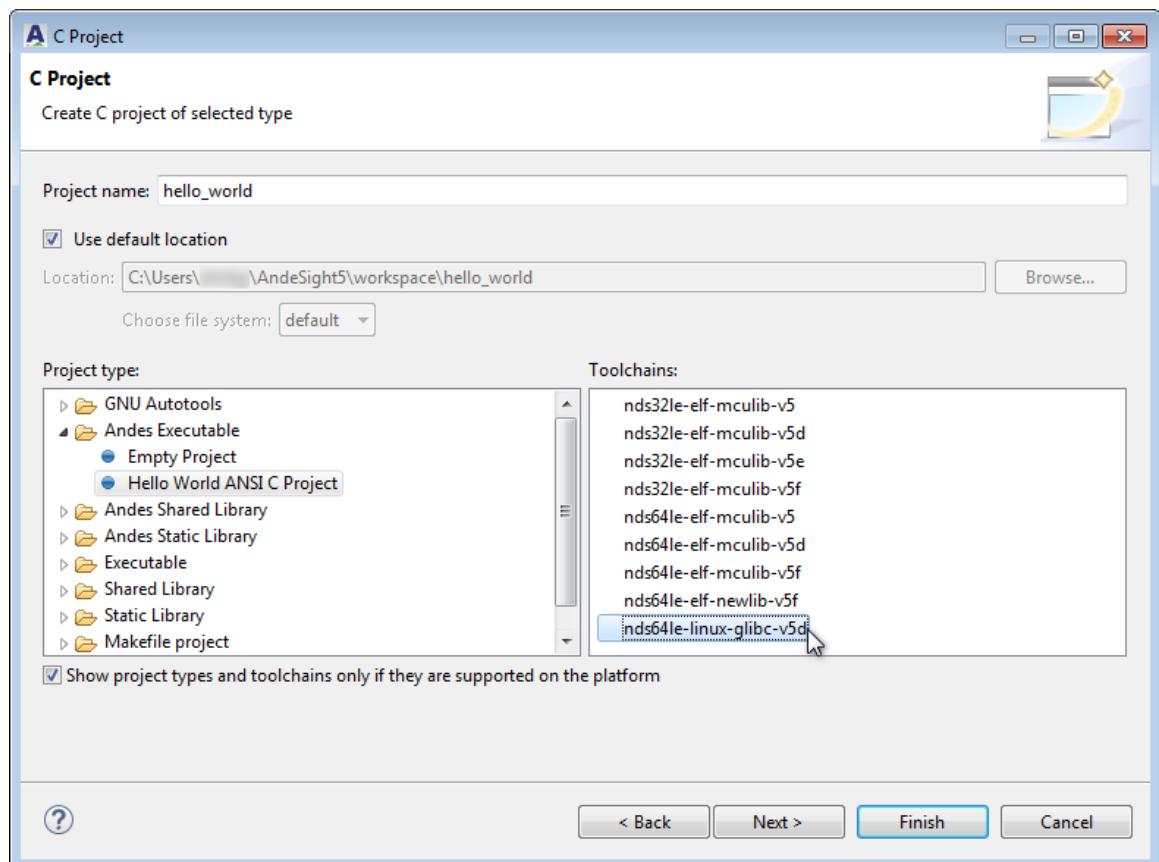
## 2.4.8. Linux Application debugging

Select “Linux Application” as the configuration type for your debug session if using a target system that comes with TCP/IP network capability and has a Linux kernel running on it. Note that the program must be loaded to the Linux kernel before being debugged via a network connection.



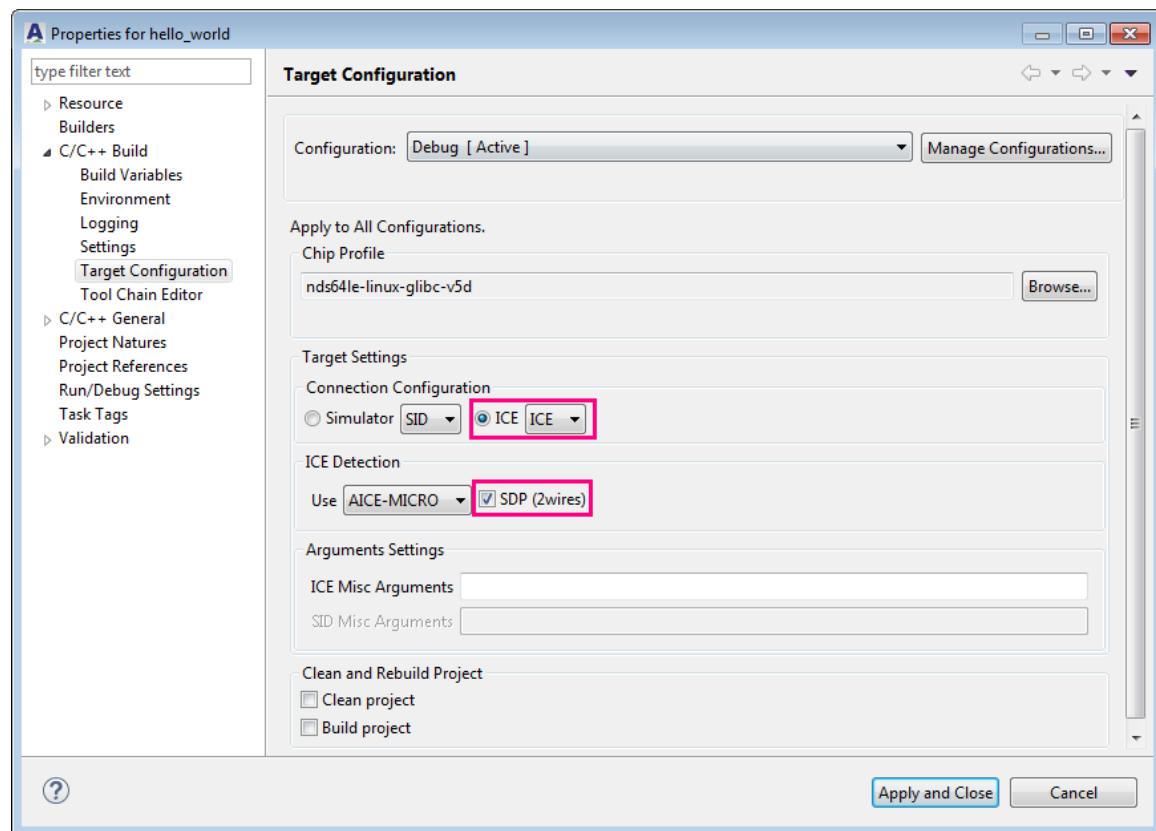
### 2.4.8.1 Launching a Linux Application debug session

**Step 1** Follow Section 2.1.1.2 to create a C project for your Linux application. Be sure to select a toolchain compatible with the Linux kernel for your project. In this example, the toolchain “nds64le-linux-glibc-v5d” is selected for the project.



If needed, refer to Section 2.1.2.2 to add header files and source files to the project.

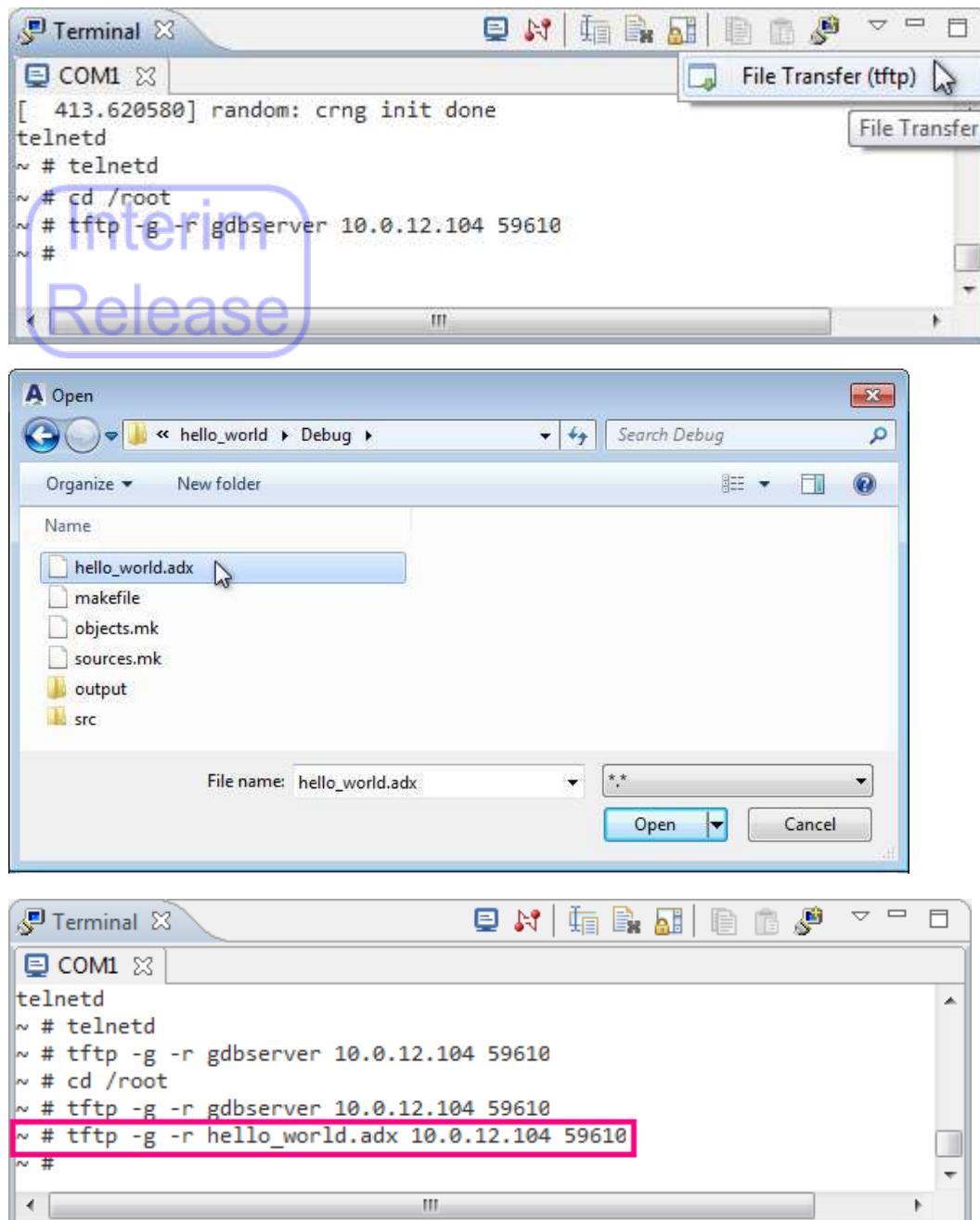
**Step 2** In Project Explorer, right-click the project and select “Target Configuration” from the pull-down menu. In the invoked dialog, specify “ICE” as the connection configuration and click “Apply and Close”. If you are using a V5 target board with 2-wire debug interface in conjunction with the ICE device AICE-MICRO or AICE-MINI+, make sure to check the option “SDP (2 wires)” in the ICE detection section as well.



**Step 3** On the AndeSight toolbar, click  (Build) to build the project.

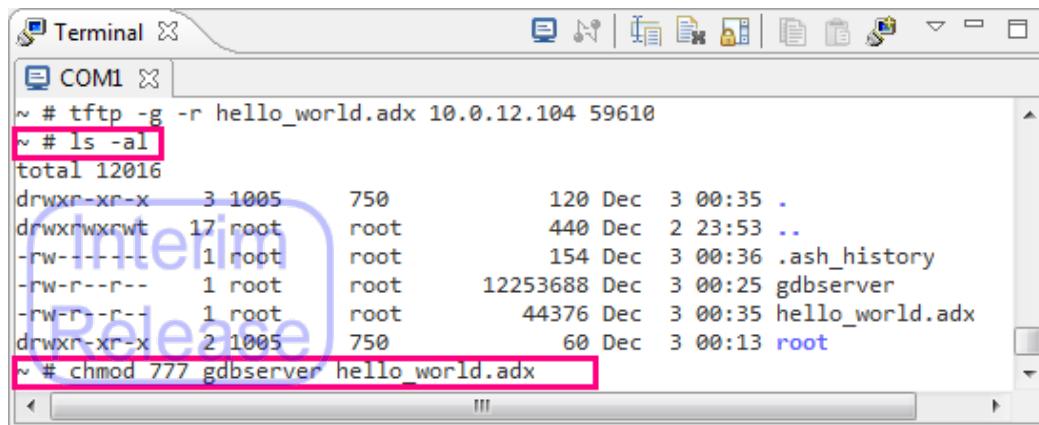
**Step 4** Follow Section 2.2.2.3 to set up the Linux application.

**Step 5** In the Terminal view, click  and select “File Transfer (tftp)”. Specify the program executable in the invoked wizard and click “Open” to transfer it to the Linux kernel. In the example below, `hello_world.adx` is transferred.



**Step 6** Verify that gdbserver and the executable are both under the root directory, and then change their modes.

```
#ls -al  
#chmod 777 GDBSERVER PROJECT_EXECUTABLE
```



Terminal window showing terminal session:

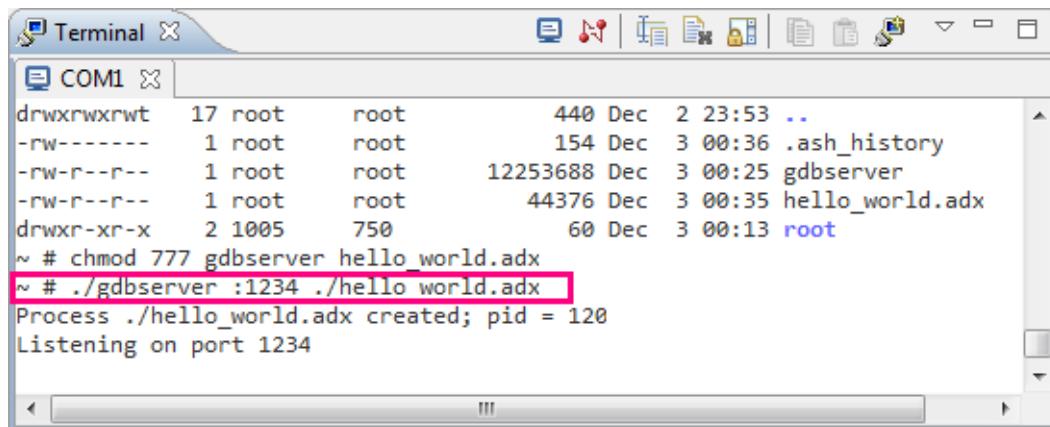
```

~ # tftp -g -r hello_world.adx 10.0.12.104 59610
~ # ls -al
total 12016
drwxr-xr-x  3 1005      750          120 Dec  3 00:35 .
drwxrwxrwt 17 root       root        440 Dec  2 23:53 ..
-rw-----  1 root       root       154 Dec  3 00:36 .ash_history
-rw-r--r--  1 root       root    12253688 Dec  3 00:25 gdbserver
-rw-r--r--  1 root       root     44376 Dec  3 00:35 hello_world.adx
drwxr-xr-x  2 1005      750          60 Dec  3 00:13 root
~ # chmod 777 gdbserver hello_world.adx

```

**Step 7** Execute gdbserver, assign a port number to which gdbserver will connect, and specify the program executable to be debugged.

**#. ./GDBSERVER :PORT\_NUMBER PROJECT\_EXECUTABLE**



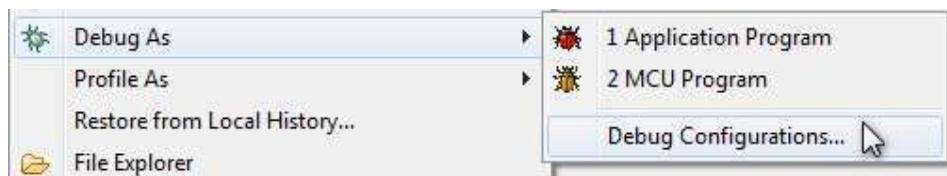
Terminal window showing terminal session:

```

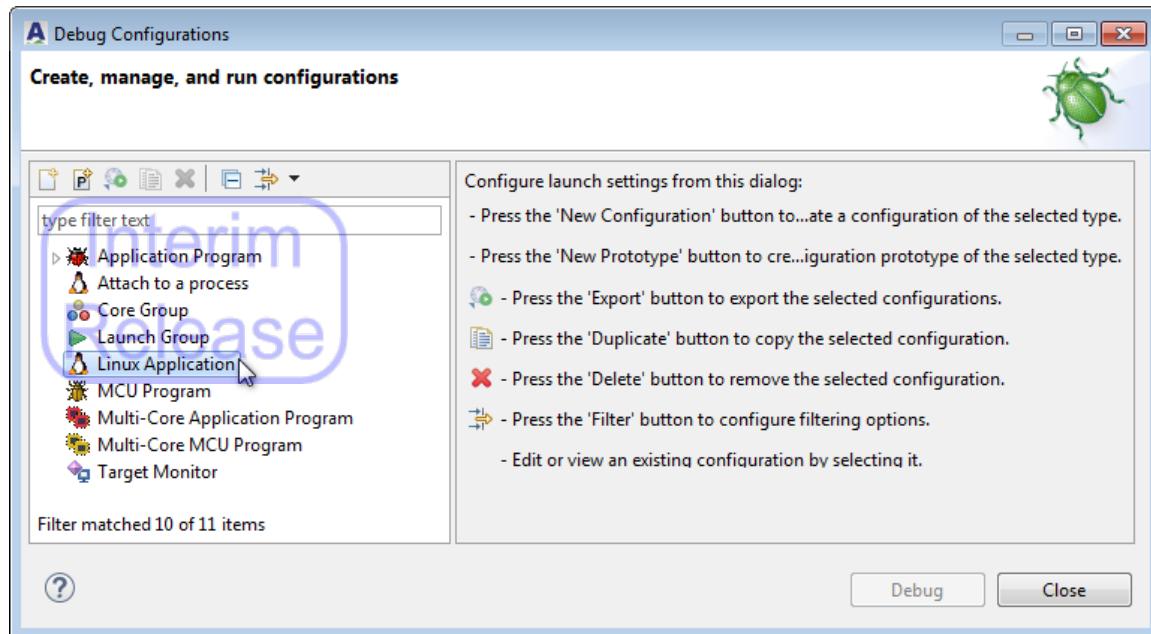
drwxrwxrwt 17 root       root        440 Dec  2 23:53 ..
-rw-----  1 root       root       154 Dec  3 00:36 .ash_history
-rw-r--r--  1 root       root    12253688 Dec  3 00:25 gdbserver
-rw-r--r--  1 root       root     44376 Dec  3 00:35 hello_world.adx
drwxr-xr-x  2 1005      750          60 Dec  3 00:13 root
~ # chmod 777 gdbserver hello_world.adx
~ # ./gdbserver :1234 ./hello_world.adx
Process ./hello_world.adx created; pid = 120
Listening on port 1234

```

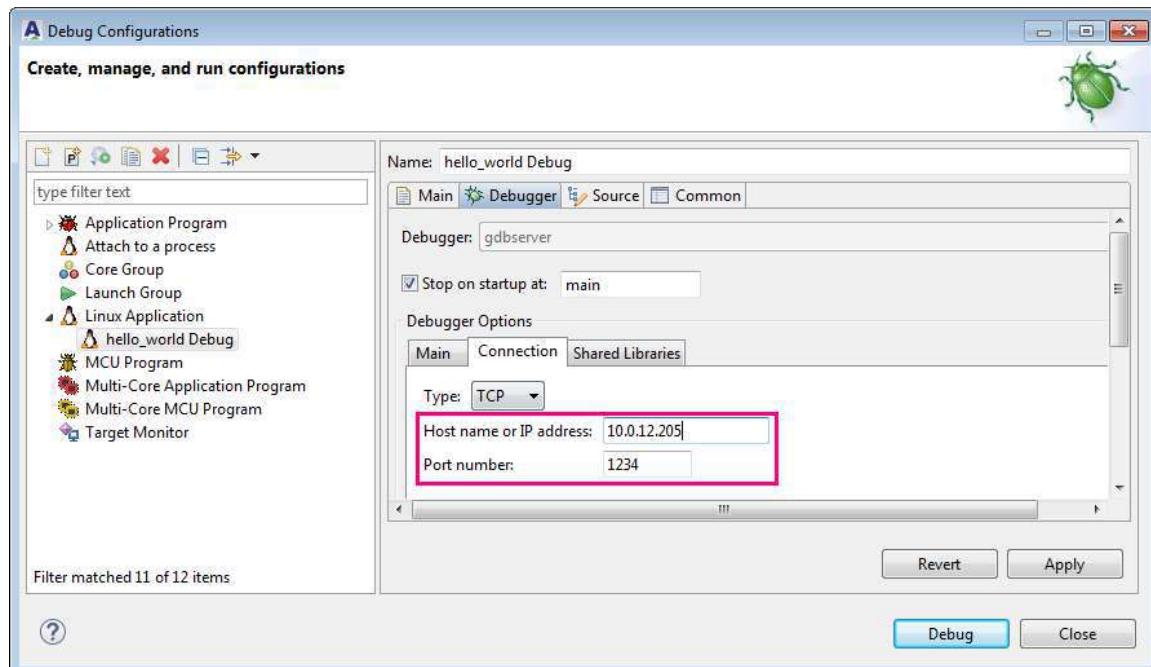
**Step 8** In **Project Explorer**, right-click the project and select “Debug as > Debug Configurations...” from the pull-down menu.



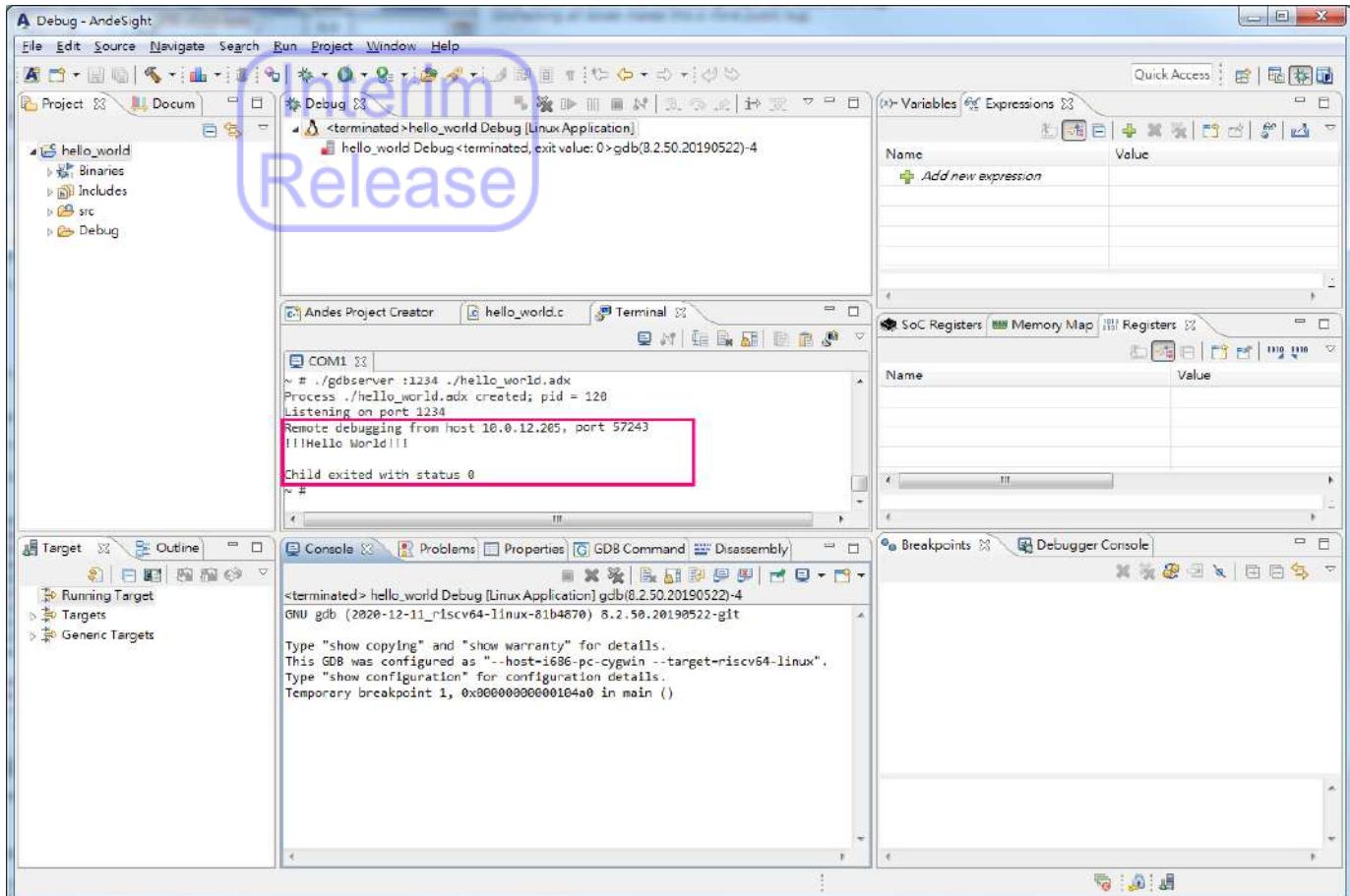
**Step 9** In the **Debug Configurations** dialog, double-click “Linux Application” to create a debug configuration for the project.



**Step 10** Go to “Debugger tab > Connection” to specify the IP address of the target system (which can be obtained via Step 3 in Section 2.2.2.3), as well as the port number to which gdb will connect (see Step 7). Click “Apply” and “Debug” to launch the Linux Application debug session.



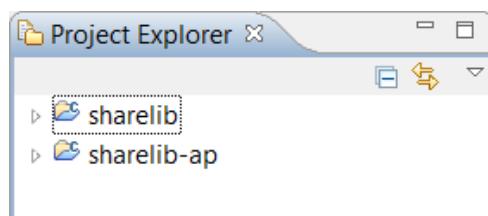
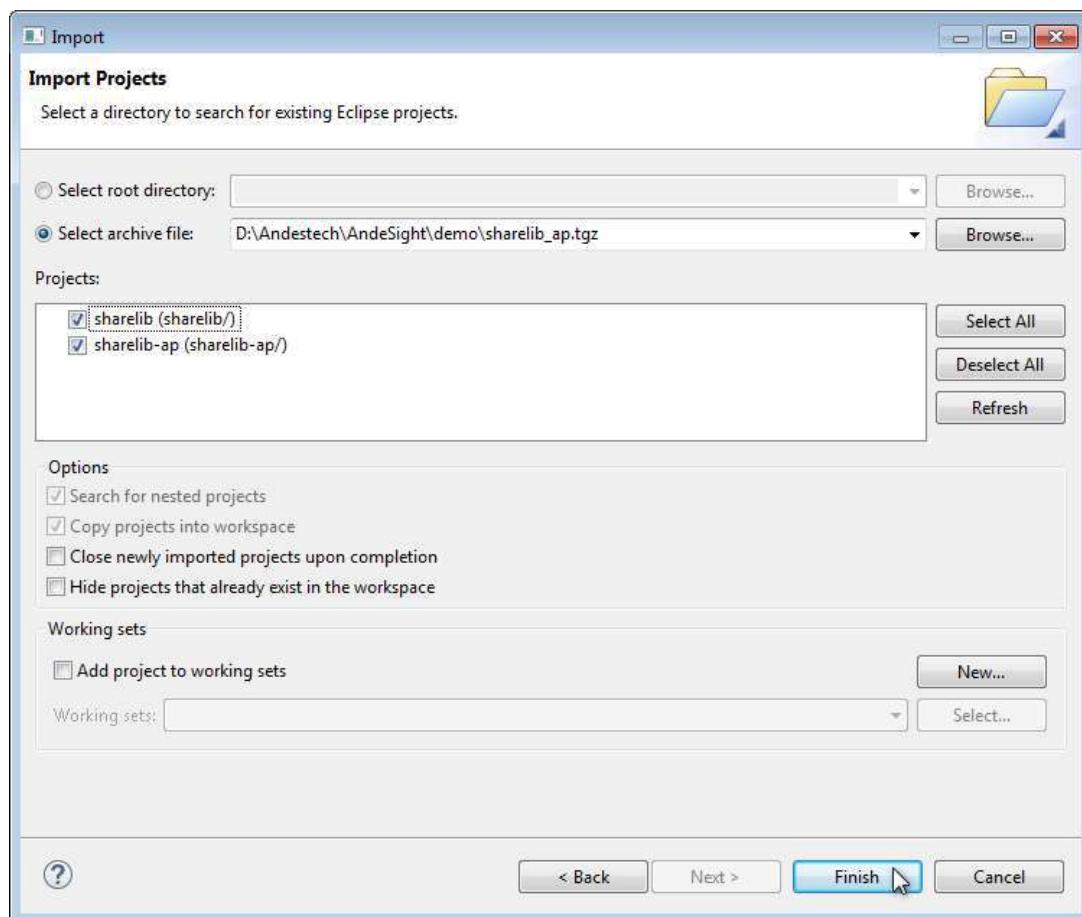
**Step 11** In the Debug perspective, click  (Resume) in the Debug view to observe the debugging process in the Terminal view.



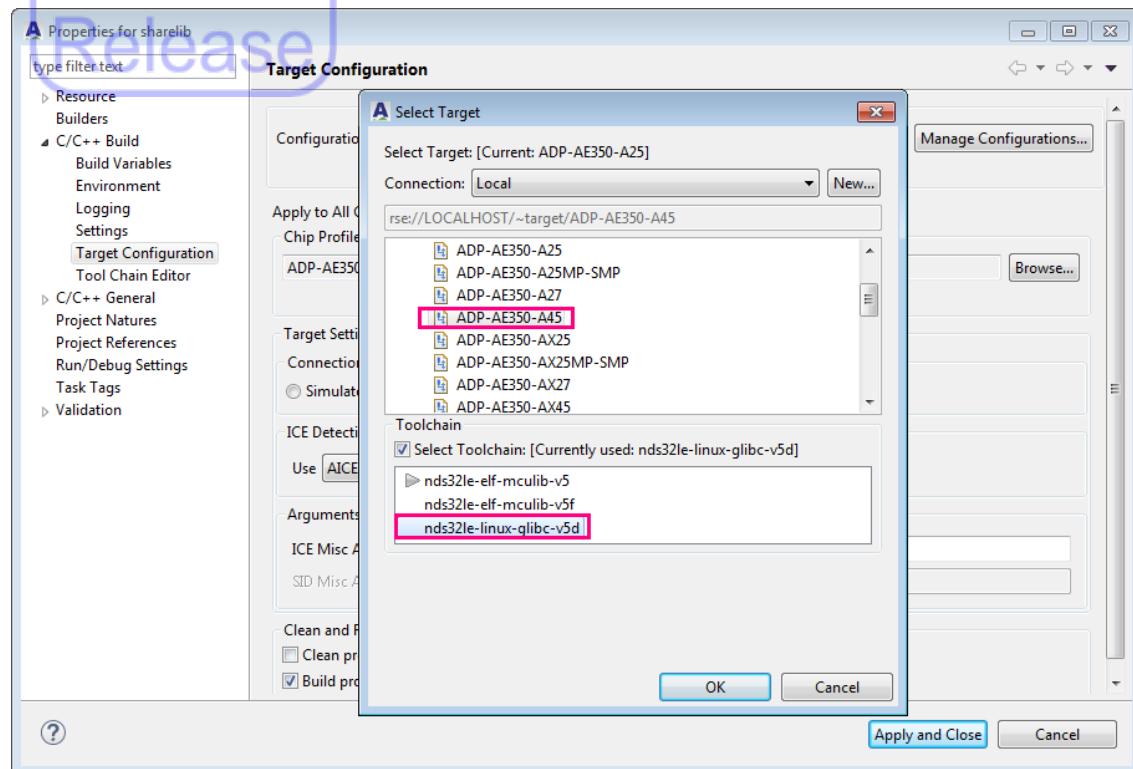
#### 2.4.8.2 Launching a Linux Application debug session with shared library

See Sections 2.1.7 and 2.1.8 to create a shared library project and C project using the shared library. In the following, “sharelib” (shared library project) and “sharelib-ap” (project that use the shared library) in AndeSight are utilized as sample projects to demonstrate the debugging of a project using a shared library.

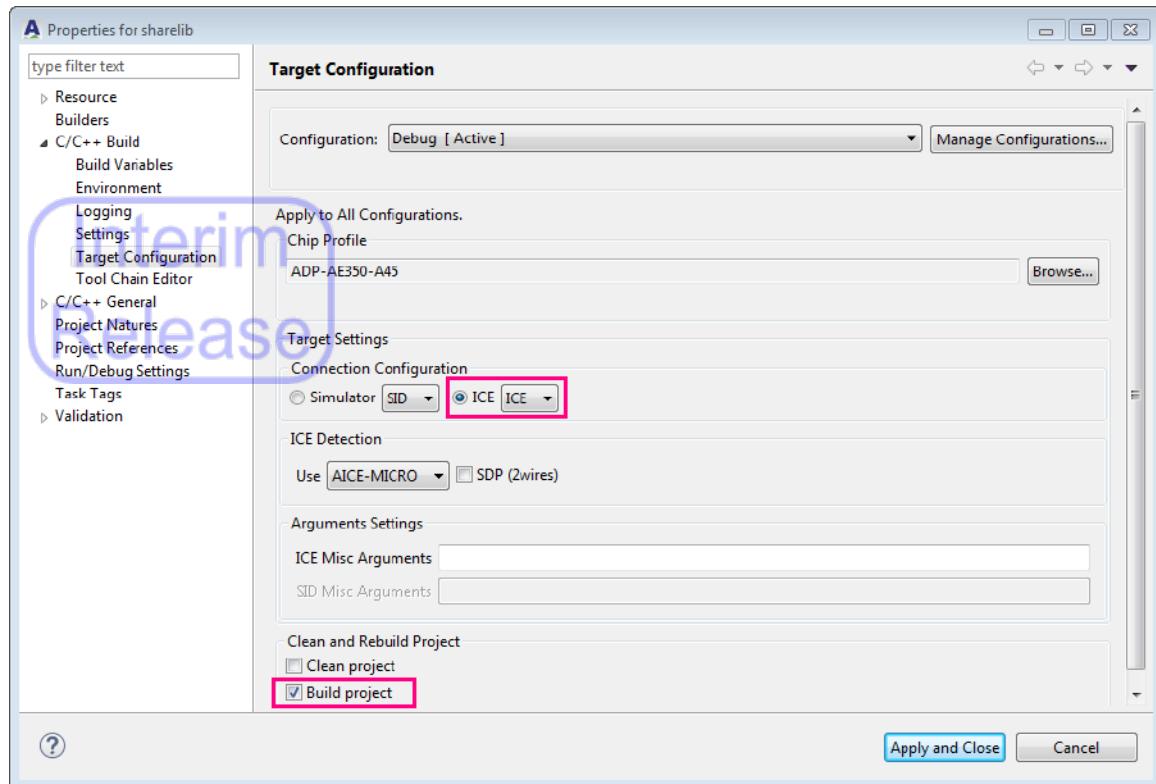
**Step 1** Follow Section 2.1.1.3 to select the compressed file “sharelib\_ap.tgz” in **ANDESIGHT\_ROOT\demo\** and import “sharelib” and “sharelib-ap” into the **Project Explorer** view.



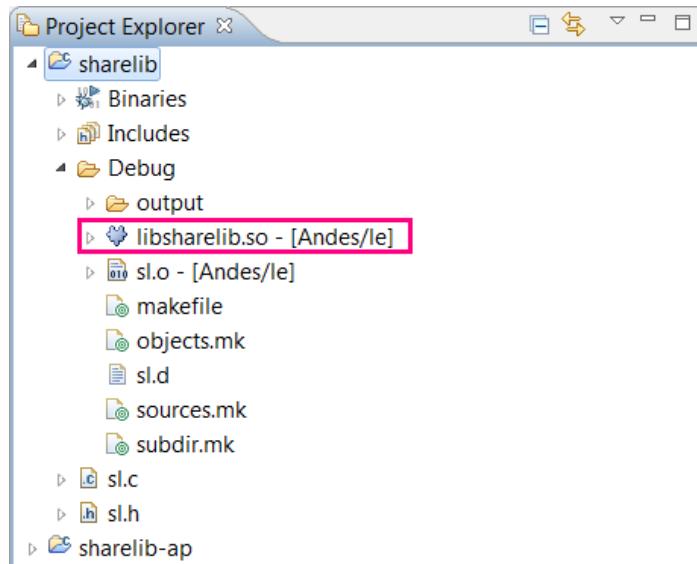
**Step 2** Right-click the “sharelib” project and select “Target Configuration” from the pull-down menu. On the **Target Configuration** page, click “Browse” to invoke the **Select Target** dialog, then select a chip profile corresponding to your target system and a toolchain compatible with your Linux kernel, and click “OK”.



**Step 3** Make sure you select “ICE” as the connection configuration and specify the “Build project” option. If you are using a V5 target board with 2-wire debug interface in conjunction with the ICE device AICE-MICRO or AICE-MINI+, be sure to also check the option “SDP (2 wires)” in the ICE detection section.



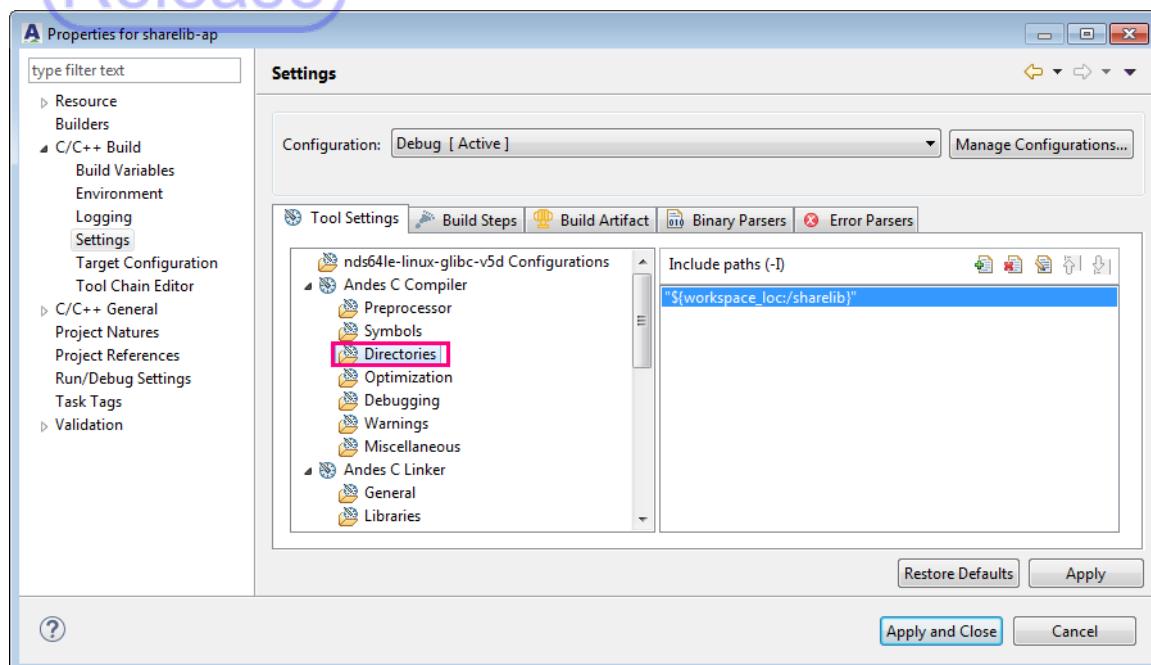
**Step 4** The build process for the project starts. The output file is generated under the \Debug folder.



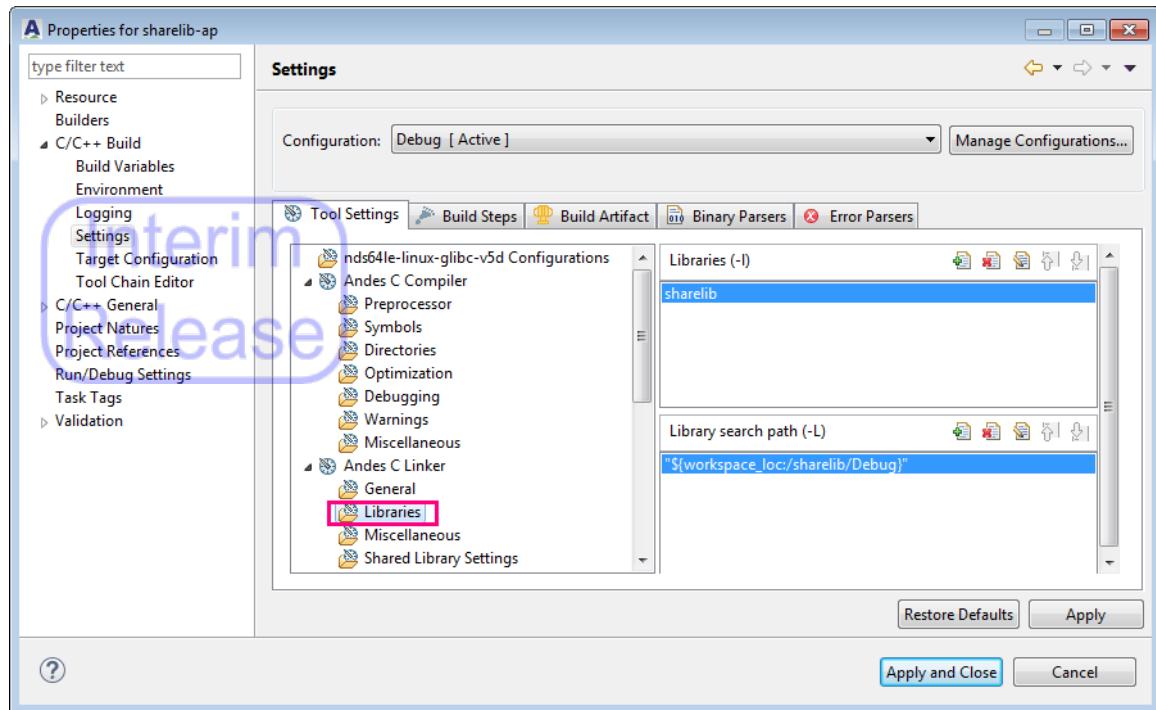
**Step 5** For a “sharelib-ap” project, follow Step 2 and Step 3 to specify a chip profile, toolchain and connection configuration matching those of the “sharelib” project.

**Step 6** In the **Properties** dialog, click “C/C++ Build > Settings > Tool Settings tab” to configure the build setting outlined below, and then click “Apply and Close”:

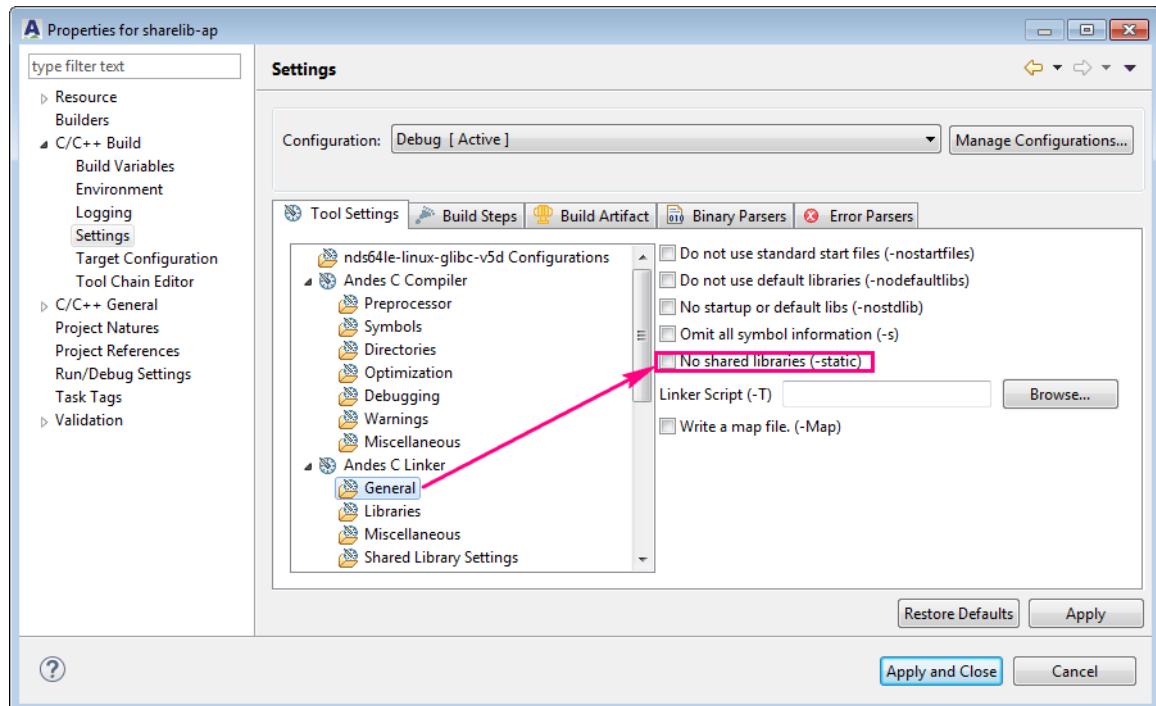
- **Andes C Compiler > Directories:** Make sure that the header file path of the shared library is added. Otherwise, click “ (Add ...)” to specify it.



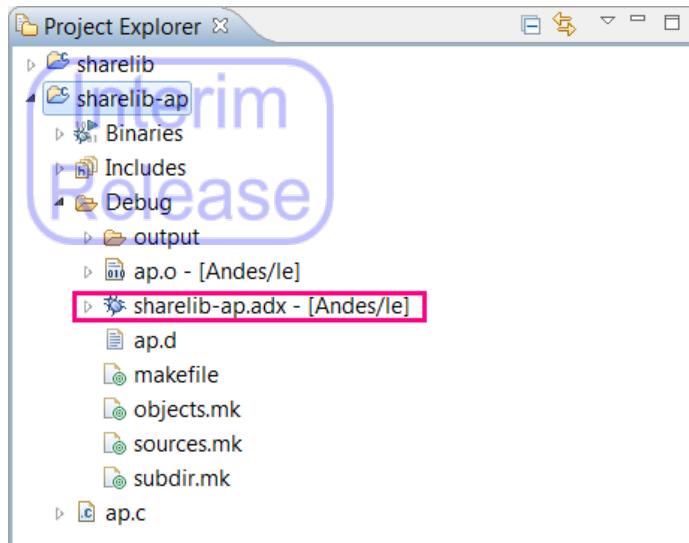
- **Andes C Linker > Libraries:** Make sure that the name and path of the shared library are added to the Libraries (-l) and the Library search path (-L) column, respectively. Otherwise, click  (Add ...) to specify them.



- **Andes C Linker > General:** Make sure the option “No shared libraries (-static)” is deselected.

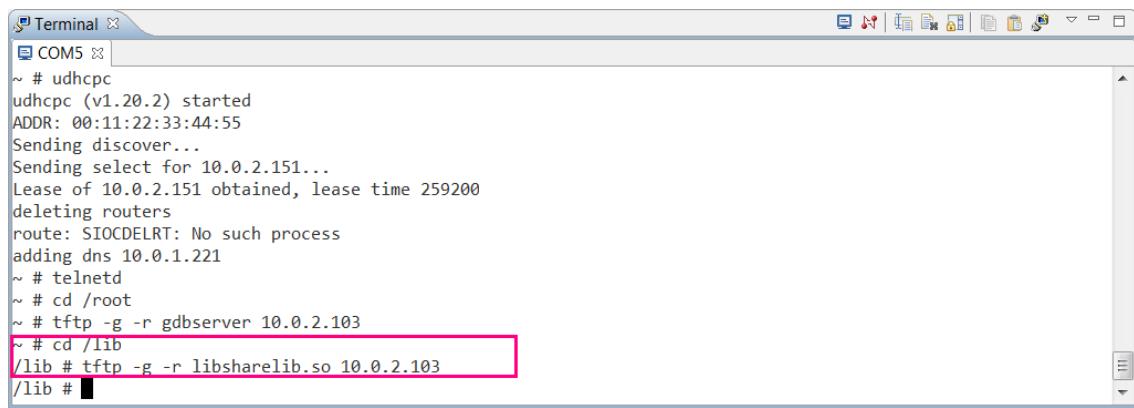


**Step 7** On the AndeSight toolbar, click  (Build) to build the “sharelib-ap” project. The output file is generated under the [/Debug](#) folder.

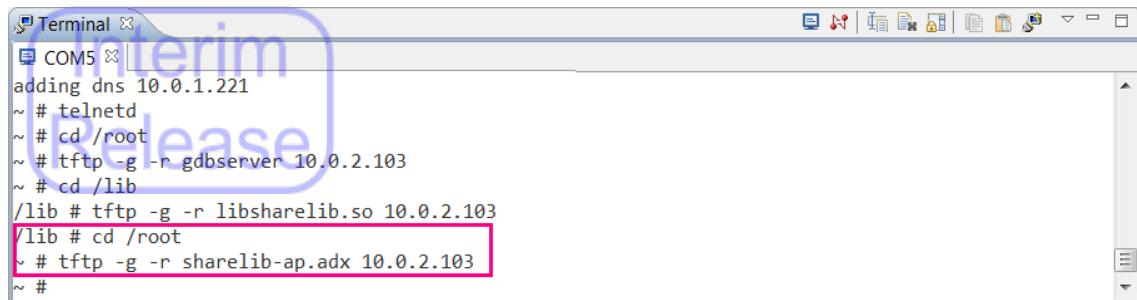


**Step 8** Follow Section 2.2.2.3 to set up the Linux application.

**Step 9** In the **Terminal** view, change the current directory to [/lib](#). Then, click  and select “File Transfer (tftp)” to transfer the shared library to the Linux kernel. In the example, **libsharelib.so** is transferred.



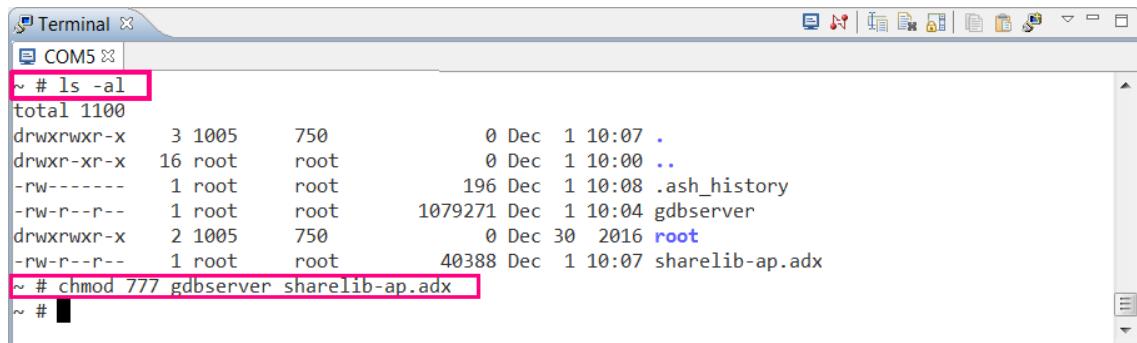
**Step 10** Change the current directory to `/root`. Click  and select “File Transfer (tftp)” again to transfer the program executable to the Linux kernel. In the example, `sharelib-ap.adx` is transferred.



```
Terminal X
COM5
adding dns 10.0.1.221
~ # telnetd
~ # cd /root
~ # tftp -g -r gdbserver 10.0.2.103
~ # cd /lib
/lib # tftp -g -r libsharelib.so 10.0.2.103
/lib # cd /root
~ # tftp -g -r sharelib-ap.adx 10.0.2.103
~ #
```

**Step 11** Verify that `gdbserver` and the executable are under `/root` directory. Then, change their modes.

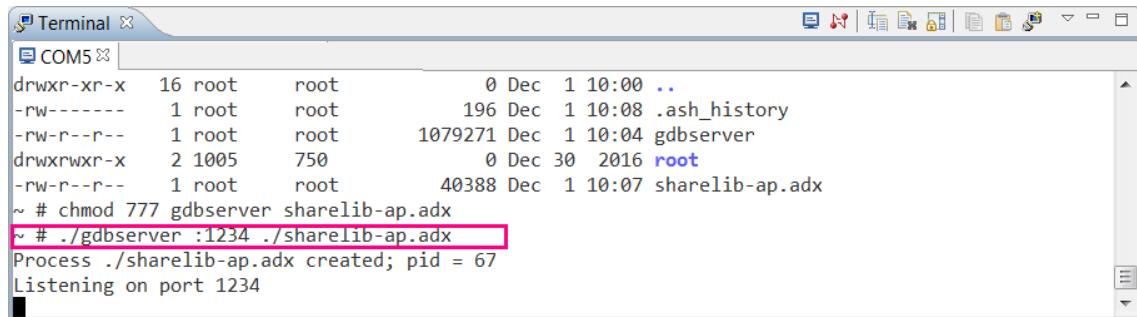
```
#ls -al
#chmod 777 GDBSERVER PROJECT_EXECUTABLE
```



```
Terminal X
COM5
~ # ls -al
total 1100
drwxrwxr-x  3 1005    750          0 Dec  1 10:07 .
drwxr-xr-x  16 root     root         0 Dec  1 10:00 ..
-rw-----  1 root     root        196 Dec  1 10:08 .ash_history
-rw-r--r--  1 root     root      1079271 Dec  1 10:04 gdbserver
drwxrwxr-x  2 1005    750          0 Dec 30 2016 root
-rw-r--r--  1 root     root      40388 Dec  1 10:07 sharelib-ap.adx
~ # chmod 777 gdbserver sharelib-ap.adx
~ #
```

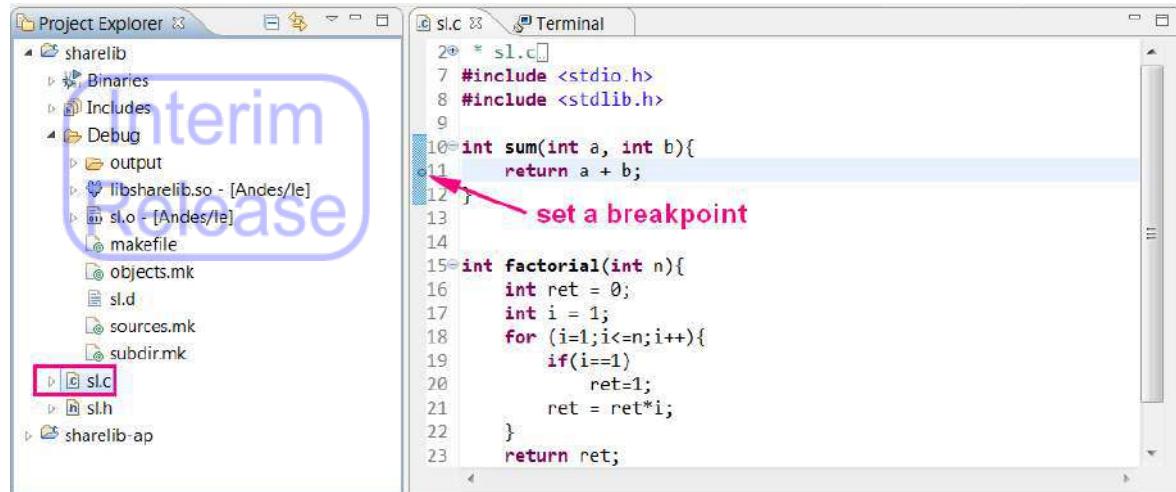
**Step 12** Execute `gdbserver`, assign a port number to which `gdbserver` will connect, and specify the program executable to be debugged.

```
./GDBSERVER :PORT_NUMBER PROJECT_EXECUTABLE
```



```
Terminal X
COM5
drwxr-xr-x  16 root     root         0 Dec  1 10:00 ..
-rw-----  1 root     root        196 Dec  1 10:08 .ash_history
-rw-r--r--  1 root     root      1079271 Dec  1 10:04 gdbserver
drwxrwxr-x  2 1005    750          0 Dec 30 2016 root
-rw-r--r--  1 root     root      40388 Dec  1 10:07 sharelib-ap.adx
~ # chmod 777 gdbserver sharelib-ap.adx
~ # ./gdbserver :1234 ./sharelib-ap.adx
Process ./sharelib-ap.adx created; pid = 67
Listening on port 1234
```

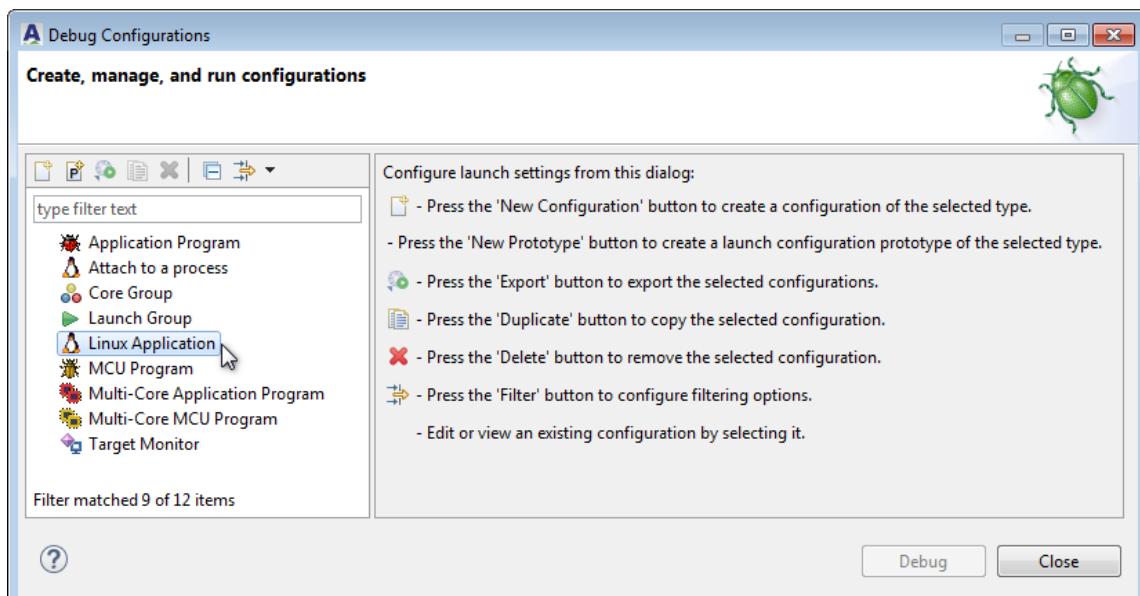
**Step 13** Follow Section 2.4.4.1 to set breakpoints in the source code of the “sharelib” project.



**Step 14** Right-click the “sharelib-ap” project and select “Debug as > Debug Configurations...” from the pull-down menu.

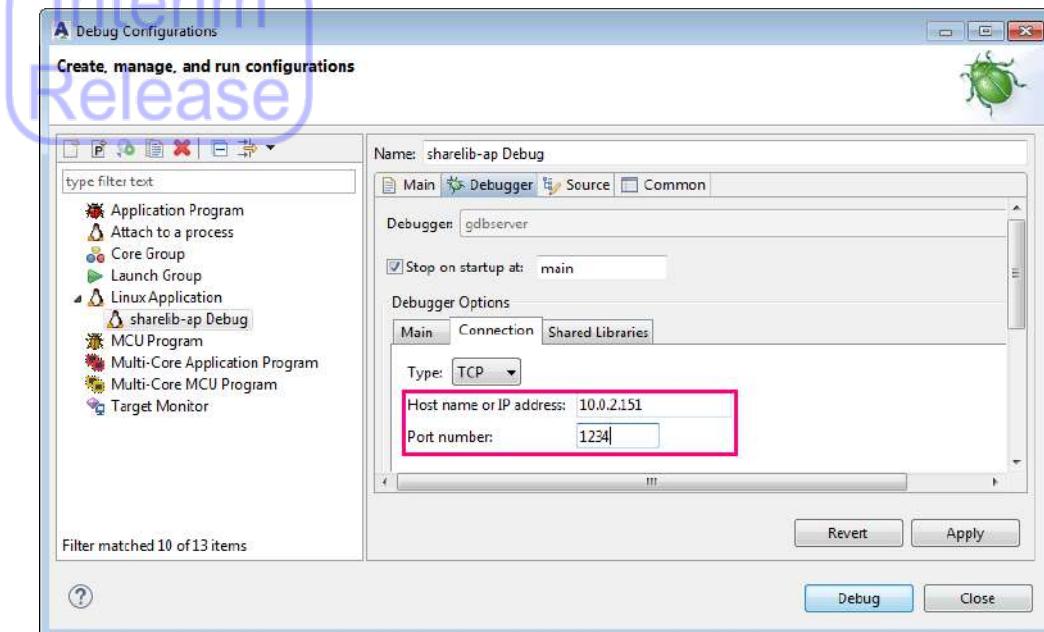


**Step 15** On the **Debug Configurations** dialog, double-click “Linux Application” to create a debug configuration for the project.

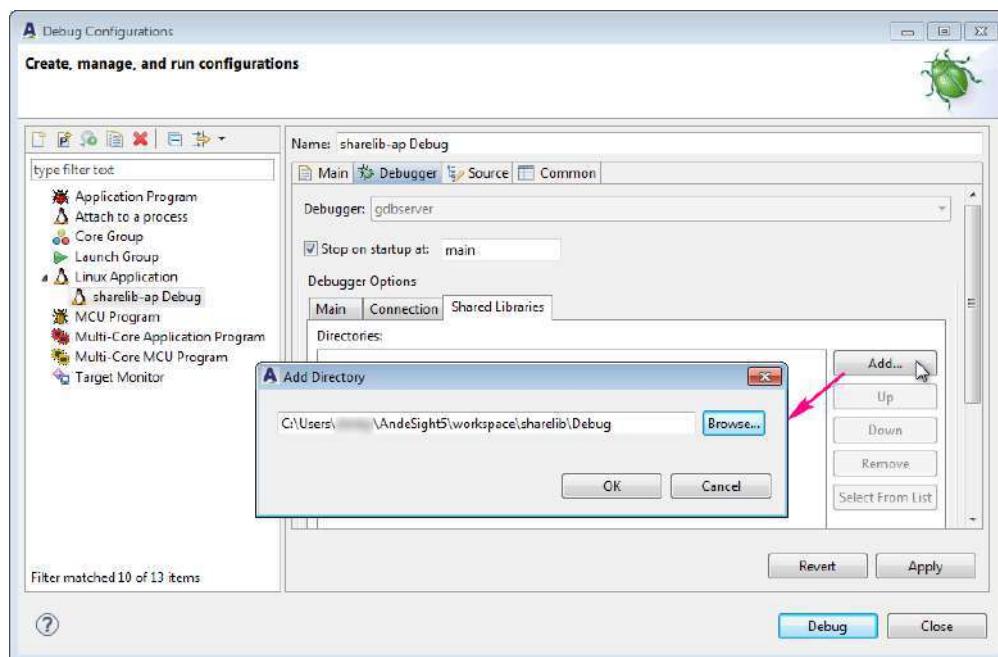


**Step 16** Work through the configuration tabs as outlined below and then click “Apply” and “Debug” to launch a debug session:

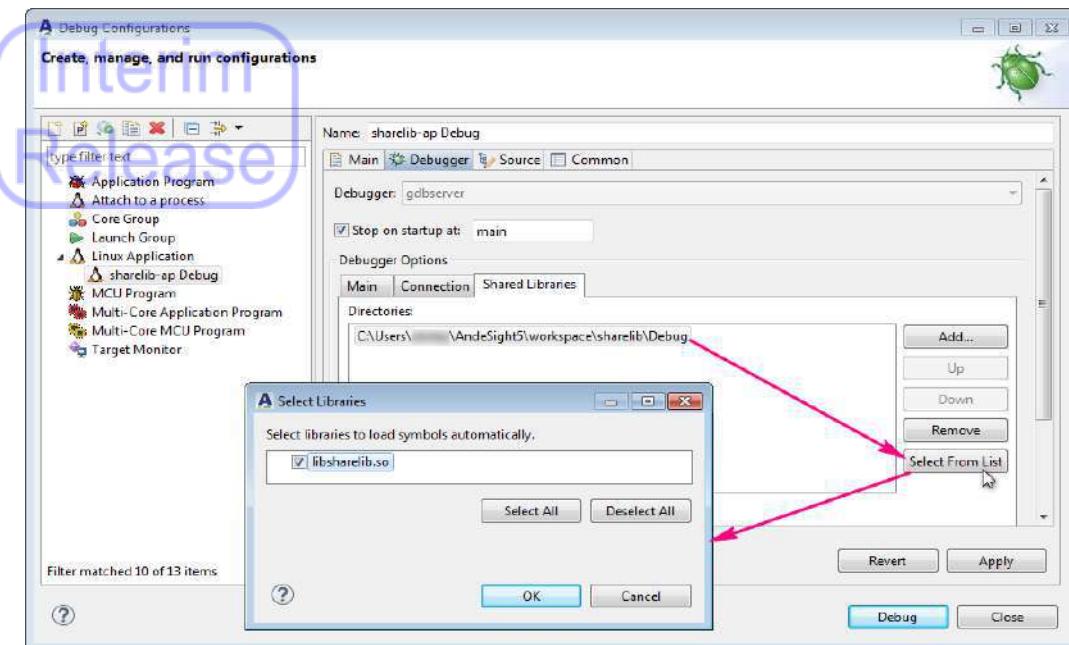
- **Debugger tab > Connection tab:** Specify the IP address of the target system and the port number to which GDB will connect.



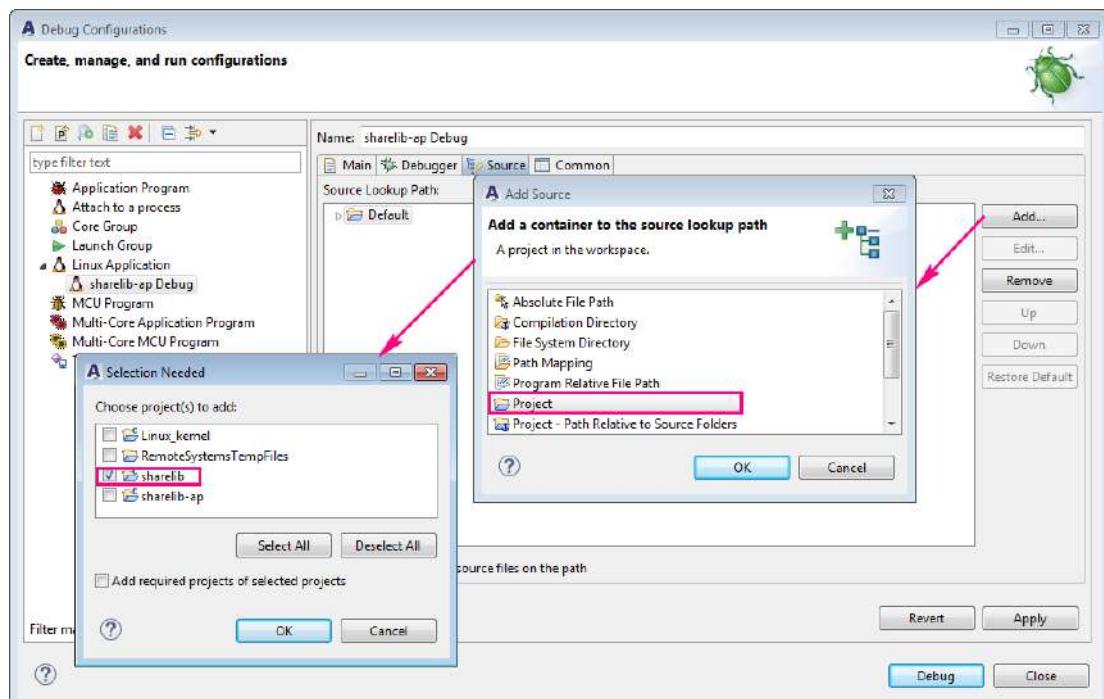
- **Debugger tab > Shared Libraries tab:** Click the “Add...” button to invoke the **Add Directory** dialog. Click “Browse...” to select the folder that includes the shared library and press “OK.”



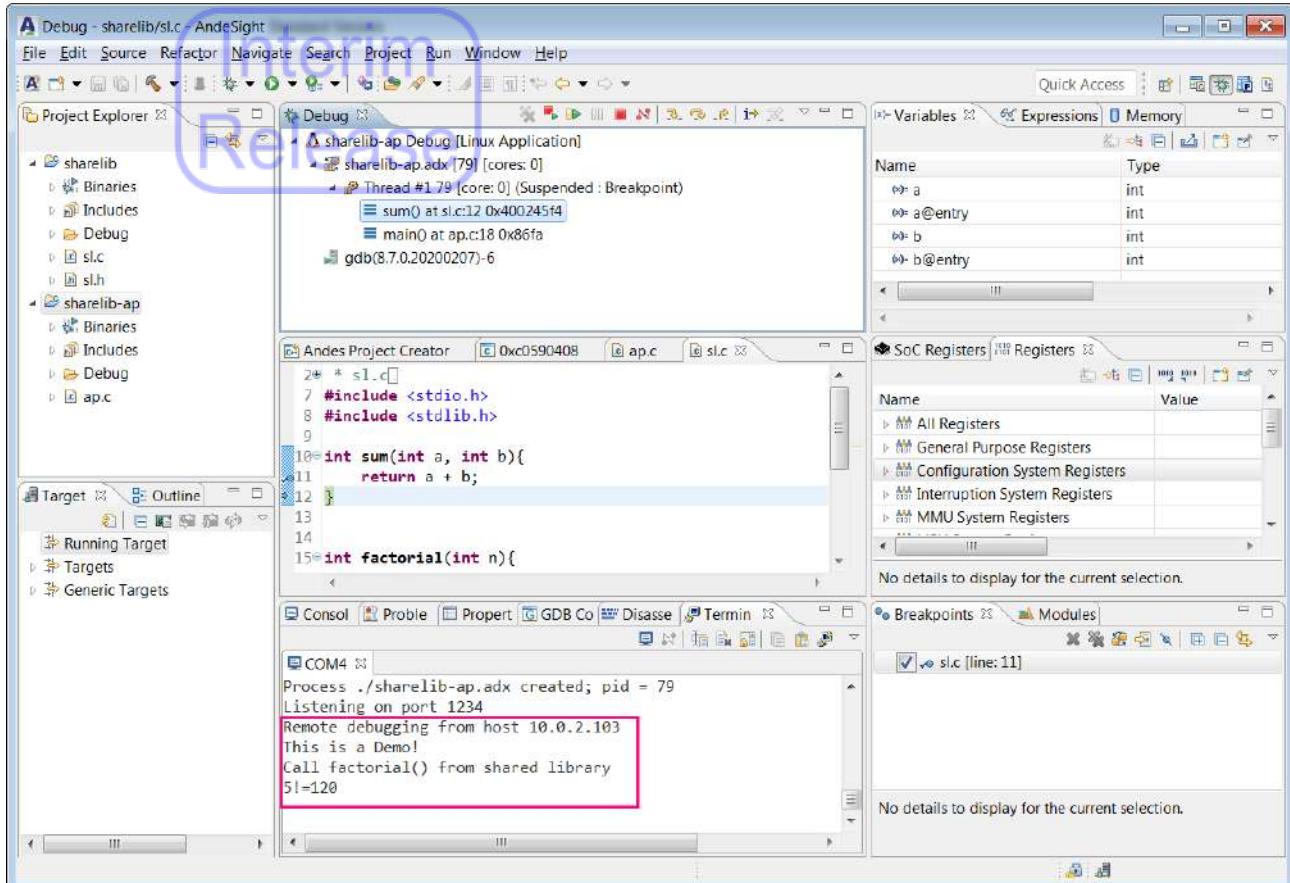
Click the directory that you just added, and press the “Select From List” button to select the shared library of interest, and then click “OK.”



- **Source tab:** Click the “Add...” button to invoke the **Add Source dialog**. Select “Project” and click “OK”. Check the shared library project in the invoked **Selection Needed** dialog and click “OK.”

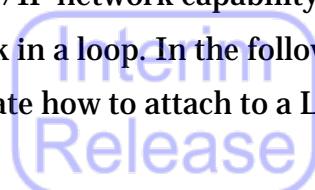


**Step 17** The **Debug** perspective is invoked automatically. Click  (Resume) in the **Debug** view to observe the debugging process in the **Terminal** View.



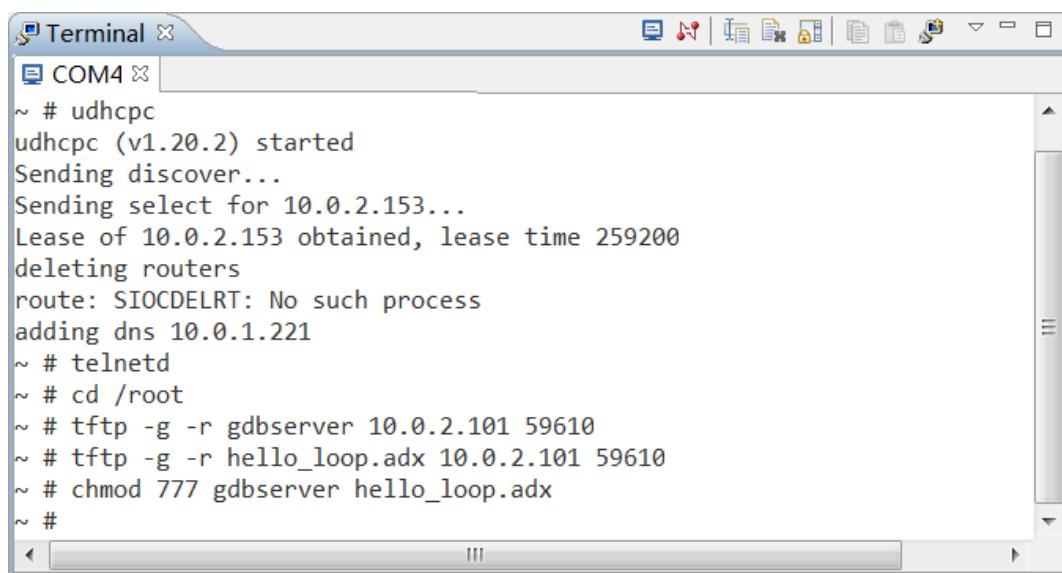
## 2.4.9. Attach to Process

The “Attach to Process” configuration is used to debug a Linux process running on a target system with TCP/IP network capability. This approach is useful for debugging programs that hang or are stuck in a loop. In the following subsection, a hello\_world project falling into a loop is used to illustrate how to attach to a Linux process in a debug session.



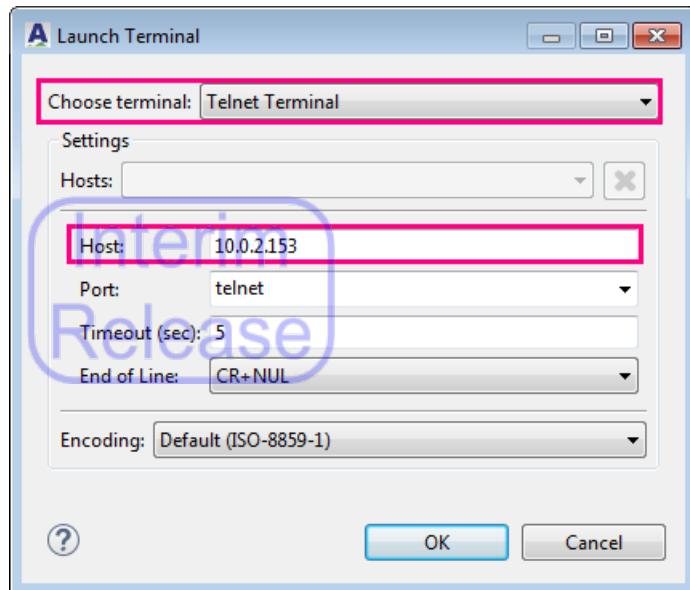
### 2.4.9.1 Launching an Attach-to-Process debug session

**Step 1** Follow Step 1 to Step 6 in Section 2.4.8.1 to create and build a C project, set up a Linux application, initiate a terminal connection, transfer gdbserver and executable program ([gdbserver](#) and [hello\\_loop.adx](#) respectively in the example) to the Linux kernel, and change their modes.



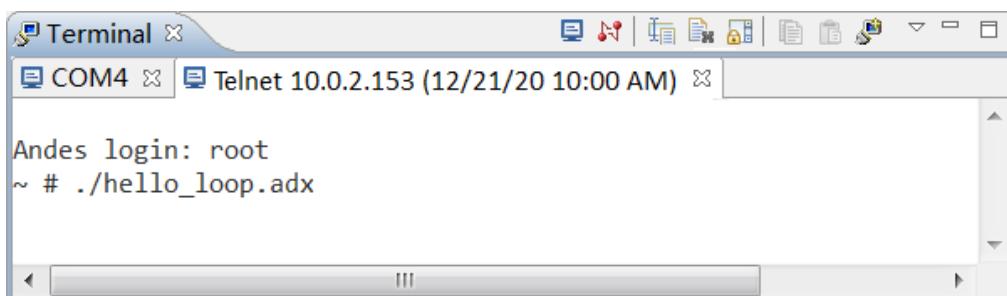
```
Terminal x
COM4 x
~ # udhcpc
udhcpc (v1.20.2) started
Sending discover...
Sending select for 10.0.2.153...
Lease of 10.0.2.153 obtained, lease time 259200
deleting routers
route: SIOCDELRT: No such process
adding dns 10.0.1.221
~ # telnetd
~ # cd /root
~ # tftp -g -r gdbserver 10.0.2.101 59610
~ # tftp -g -r hello_loop.adx 10.0.2.101 59610
~ # chmod 777 gdbserver hello_loop.adx
~ #
```

**Step 2** Click  (Open a terminal) on the **Terminal** toolbar. In the invoked dialog, select “Telnet Terminal” as the terminal type, enter the IP address of the target system for the host (which can be obtained via Step 3 in Section 2.2.2.3), and click “OK” to launch a new terminal.



**Step 3** Type in the account name to login and execute the program.

```
#. ./PROJECT_EXECUTABLE
```



**Step 4** Follow Step 2 to initiate another terminal connection. Start gdbserver and specify a port number in this terminal.

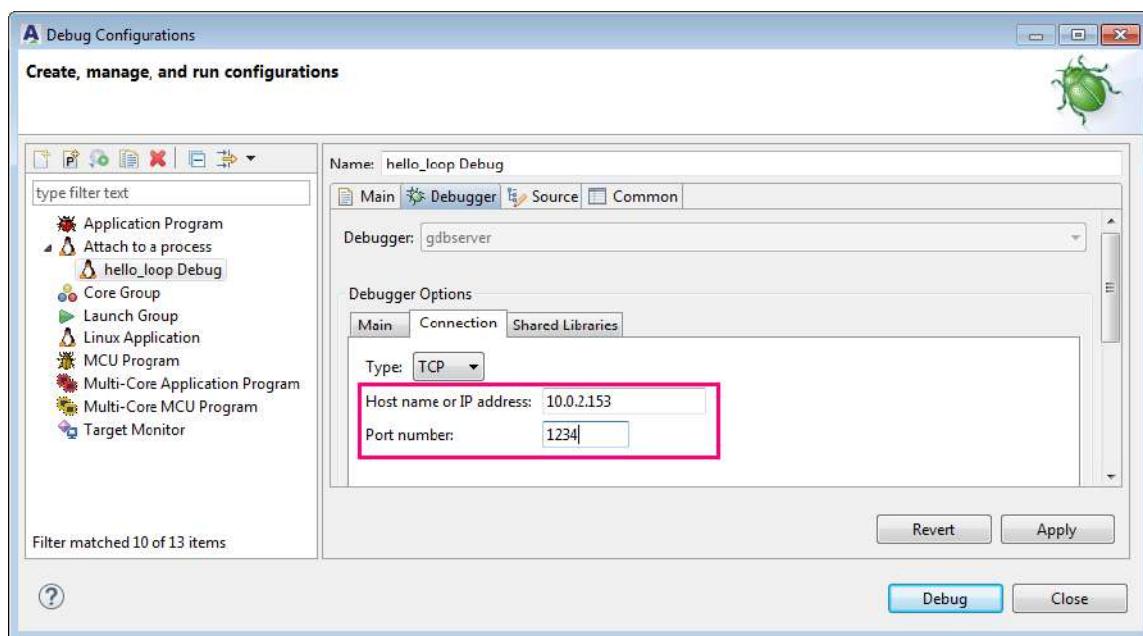
```
#. ./GDBSERVER --multi :PORT_NUMBER
```



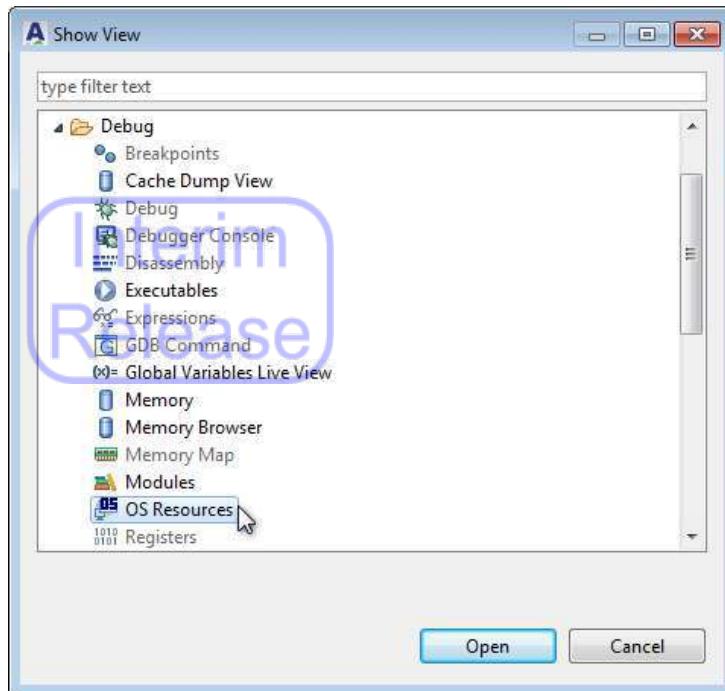
**Step 5** In **Project Explorer**, right-click the project and select “Debug as > Debug Configurations...”



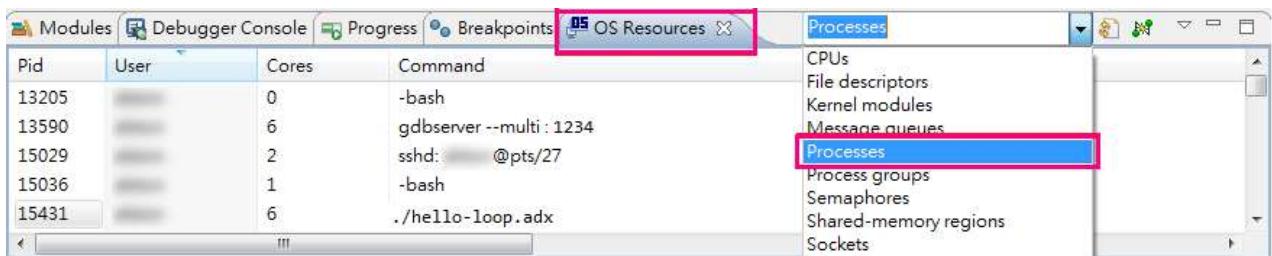
**Step 6** In the **Debug Configurations** dialog, double-click “Attach to a process” to create a debug configuration for the project. Click “**Debugger tab** > **Connection tab**” to specify the IP address of the target system and the port number to which GDB connects (see Step 4). Then, click “**Apply**” and “**Debug**” to launch a debug session. The **Debug perspective** is invoked automatically.



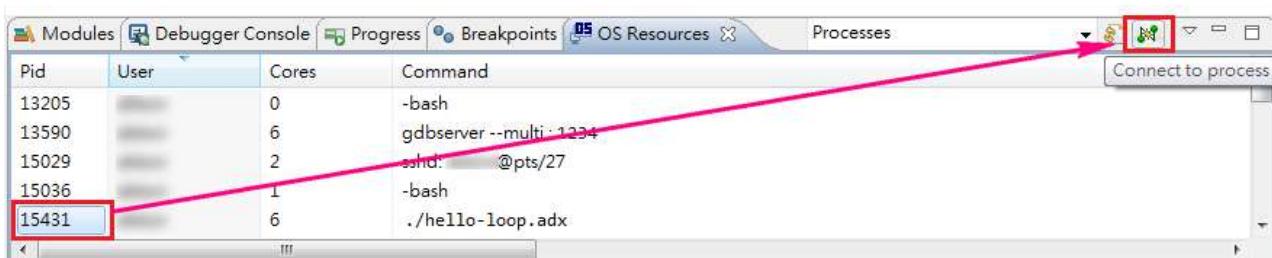
**Step 7** Invoke the **Open Resource** view by clicking “Window > Show View > Other...” from the AndeSight main menu and then selecting “Debug > OS resources” in the invoked dialog.



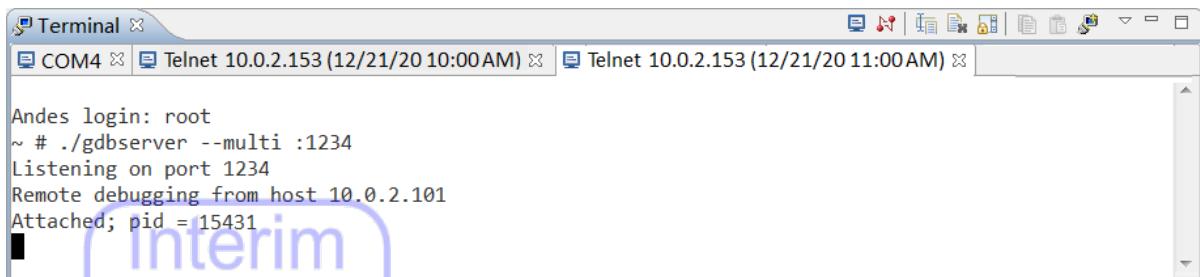
**Step 8** In the OS Resources view, select “Processes” from the combo box to display all the running processes in the view.



Next, select the PID of the running program and click on the "Connect to process" button to attach the process.



In the Terminal view, verify that the specified process is attached.



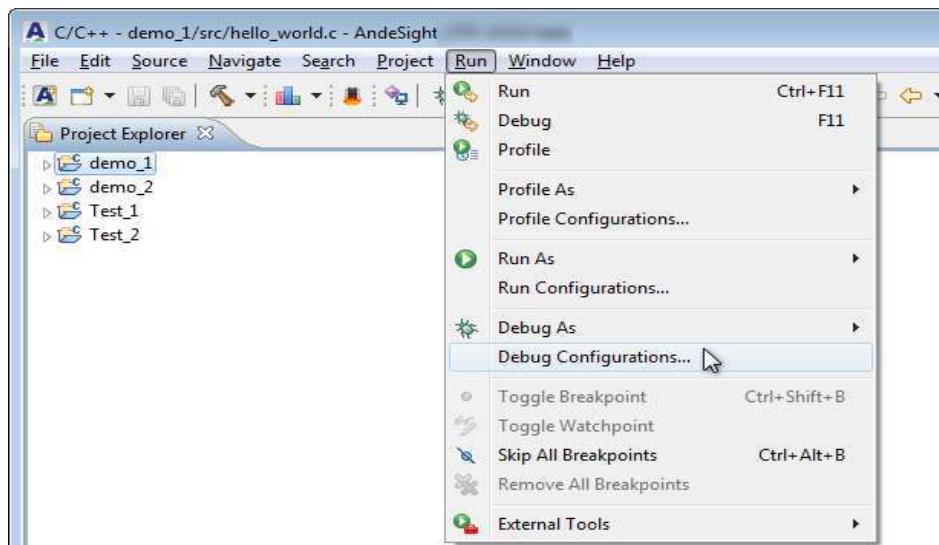
**Step 9** Proceed with debugging by clicking  (Resume),  (Step Into) or  (Step Over) on the toolbar of the **Debug** view.

## 2.4.10. Launch Group

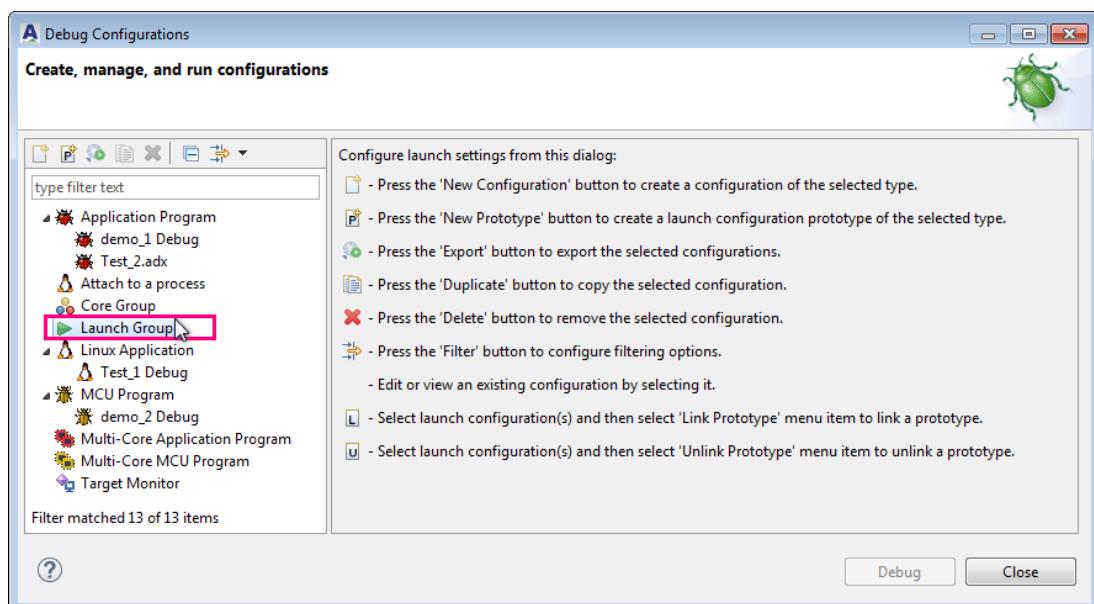
Launch Group is a handy function for launching several Run/Debug configurations simultaneously for a multi-purpose task.

### 2.4.10.1 Launching a Launch Group debug session

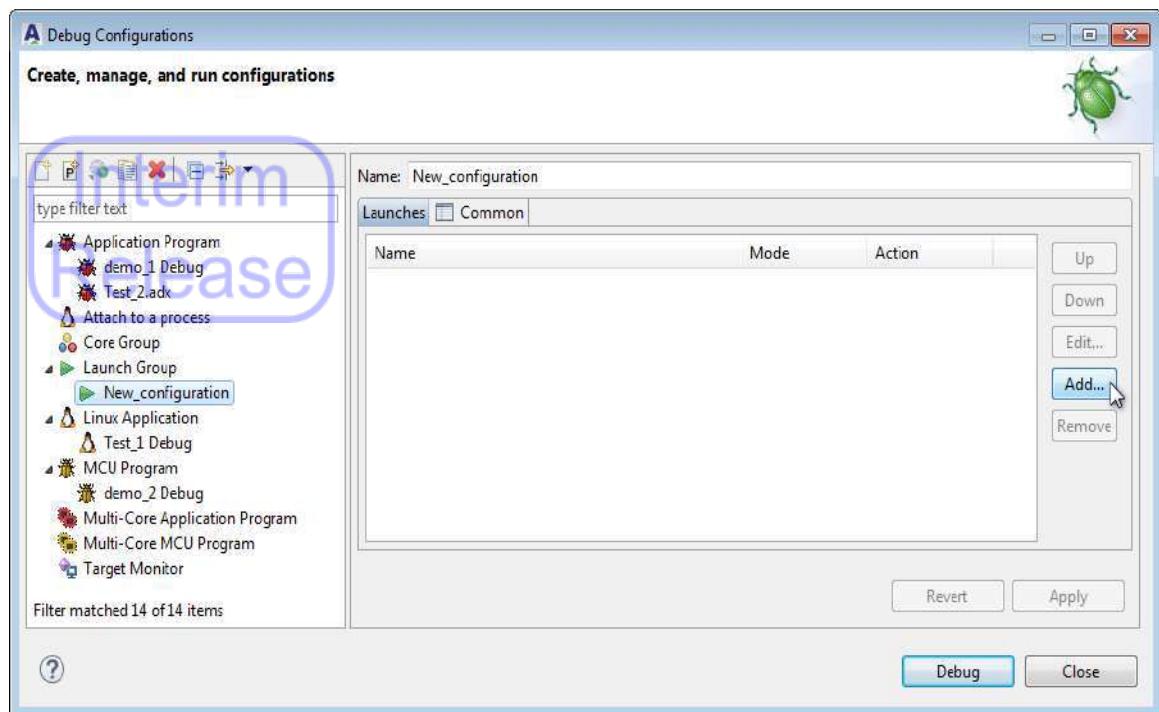
**Step 1** On the AndeSight main menu, select “Run > Debug Configurations...” to invoke the **Debug Configurations** dialog.



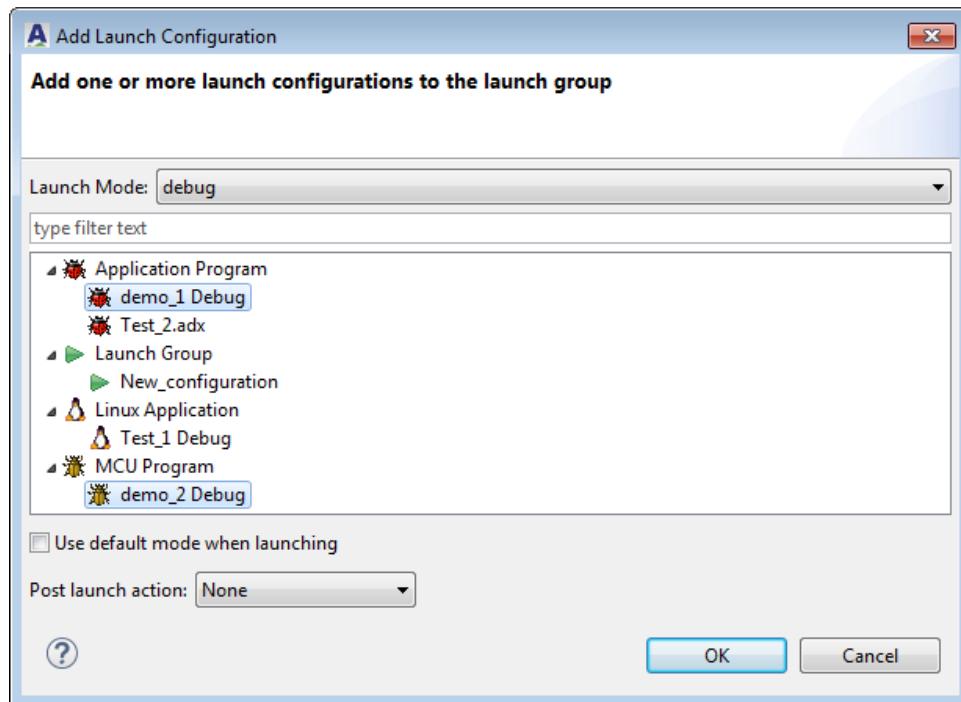
**Step 2** Double-click “Launch Group” to create a multi-task debug configuration.



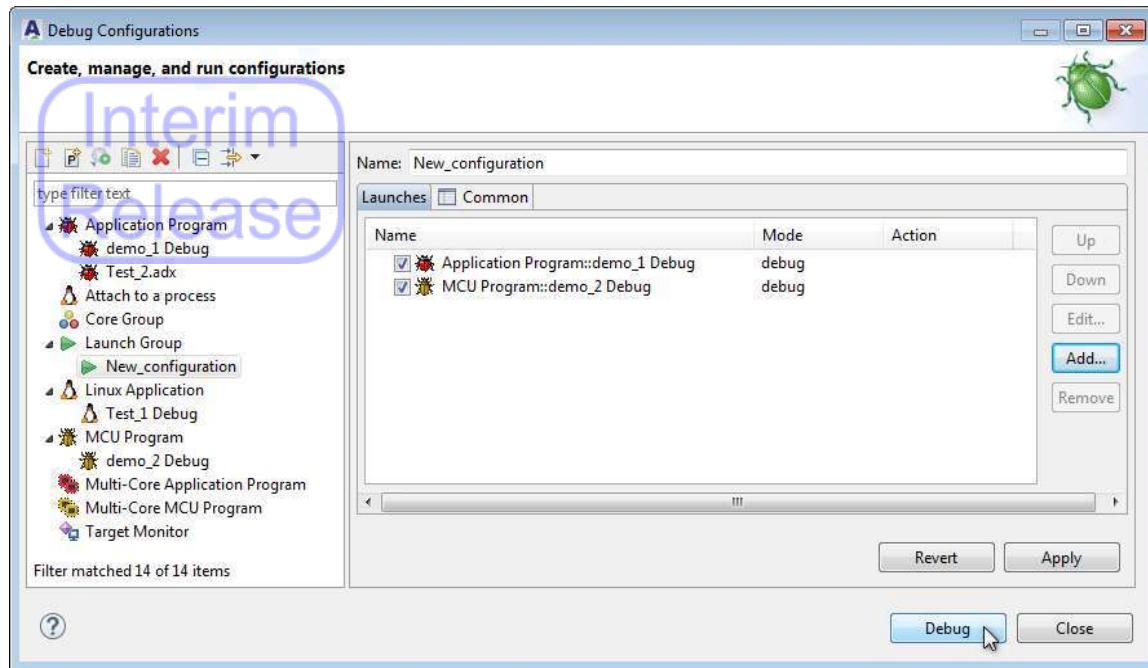
**Step 3** Name the multi-task configuration and click “Add.”



**Step 4** In the invoked **Add Launch Configuration** dialog, expand the nodes of the various debug configuration types and select at least one debug configuration for the Launch Group. Click “OK.”



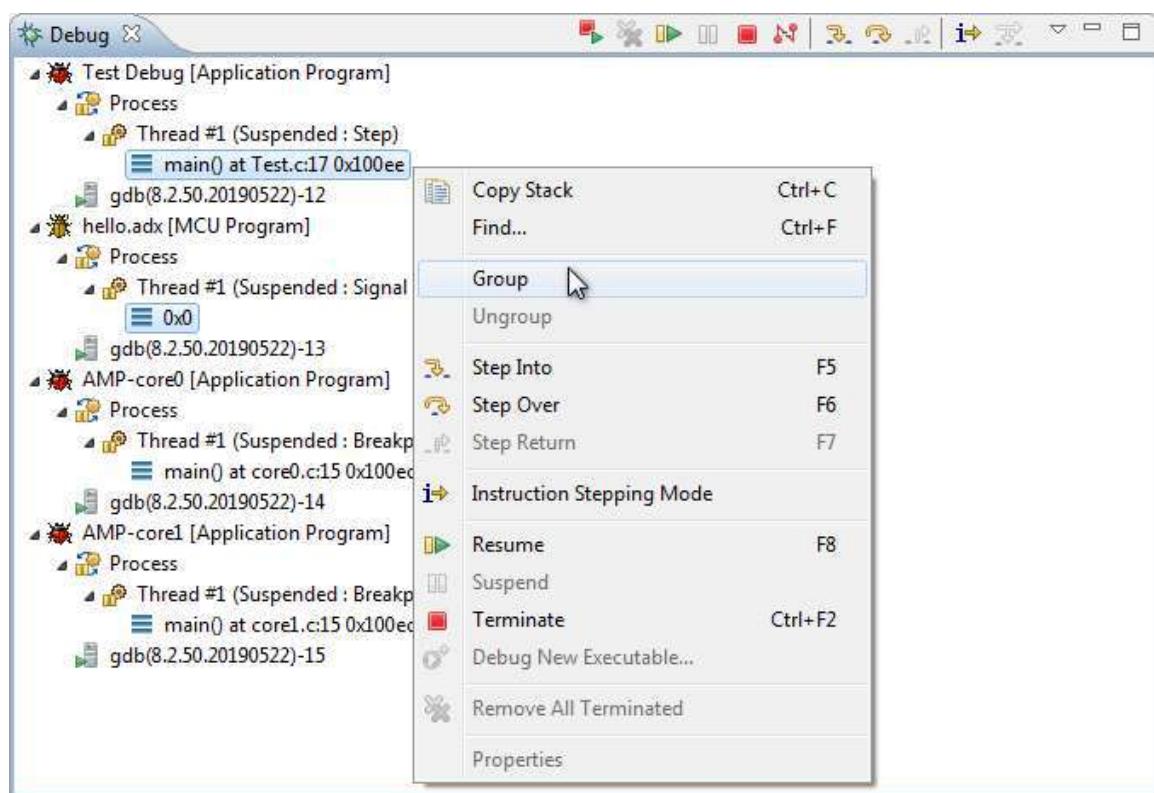
**Step 5** Click “Apply” and “Debug” to initiate the debug session using the Launch Group configuration.



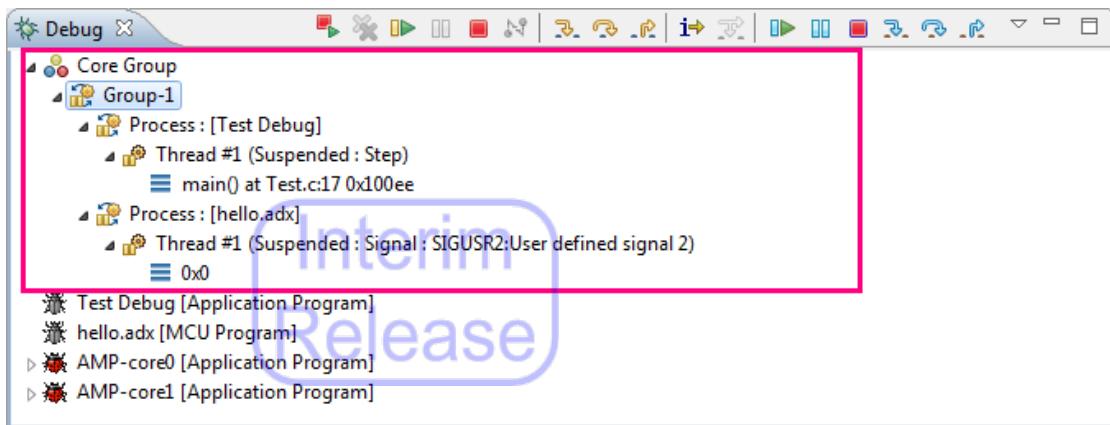
### 2.4.11. Core Group

The Core Group feature enables you to send debug commands simultaneously to a set of cores in a debug session. By grouping a number of active threads in a debug session, you can request their execution cores to perform the same debug action with one click. This saves you the time and trouble of individual operations and is particularly useful for development with targets integrated with multiple cores.

You can dynamically create a group of specific cores/threads during runtime. In the **Debug** view that displays execution stacks of debugging targets, multi-select interested active processes or threads and right-click to select “Group” in the pull-down menu.



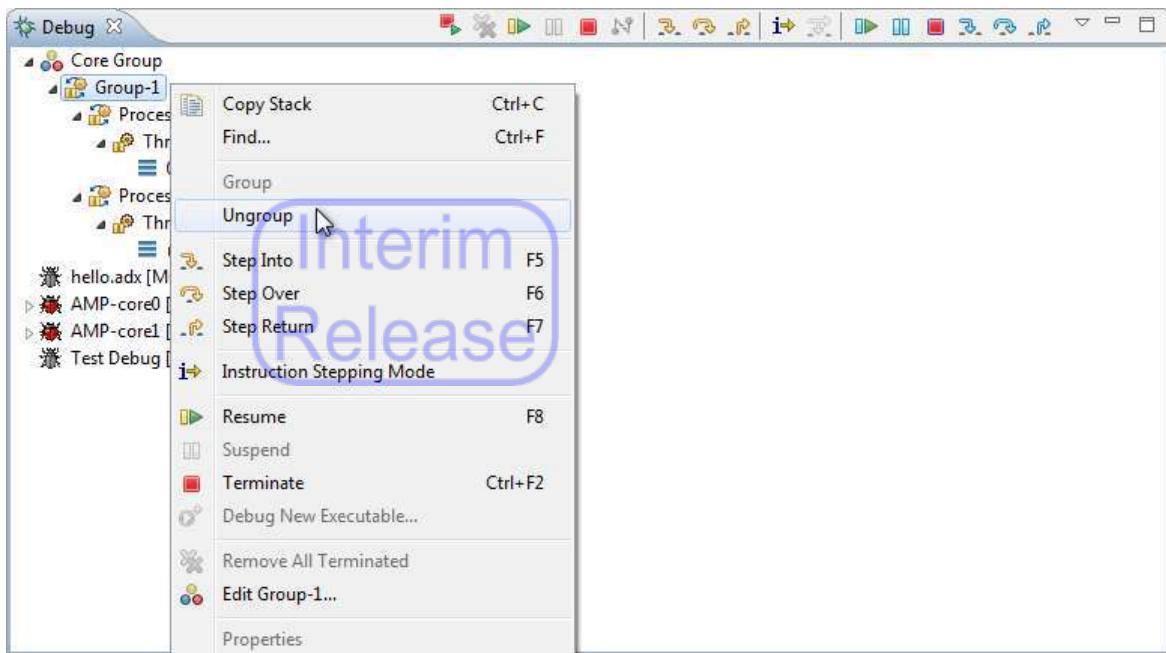
This will generate a new group (here, “Group-1”) under the “Core Group” node in the **Debug** view. This group contains the threads you just selected as its members and allows you to use toolbar buttons to send debug commands to all the cores executing the threads all at once.



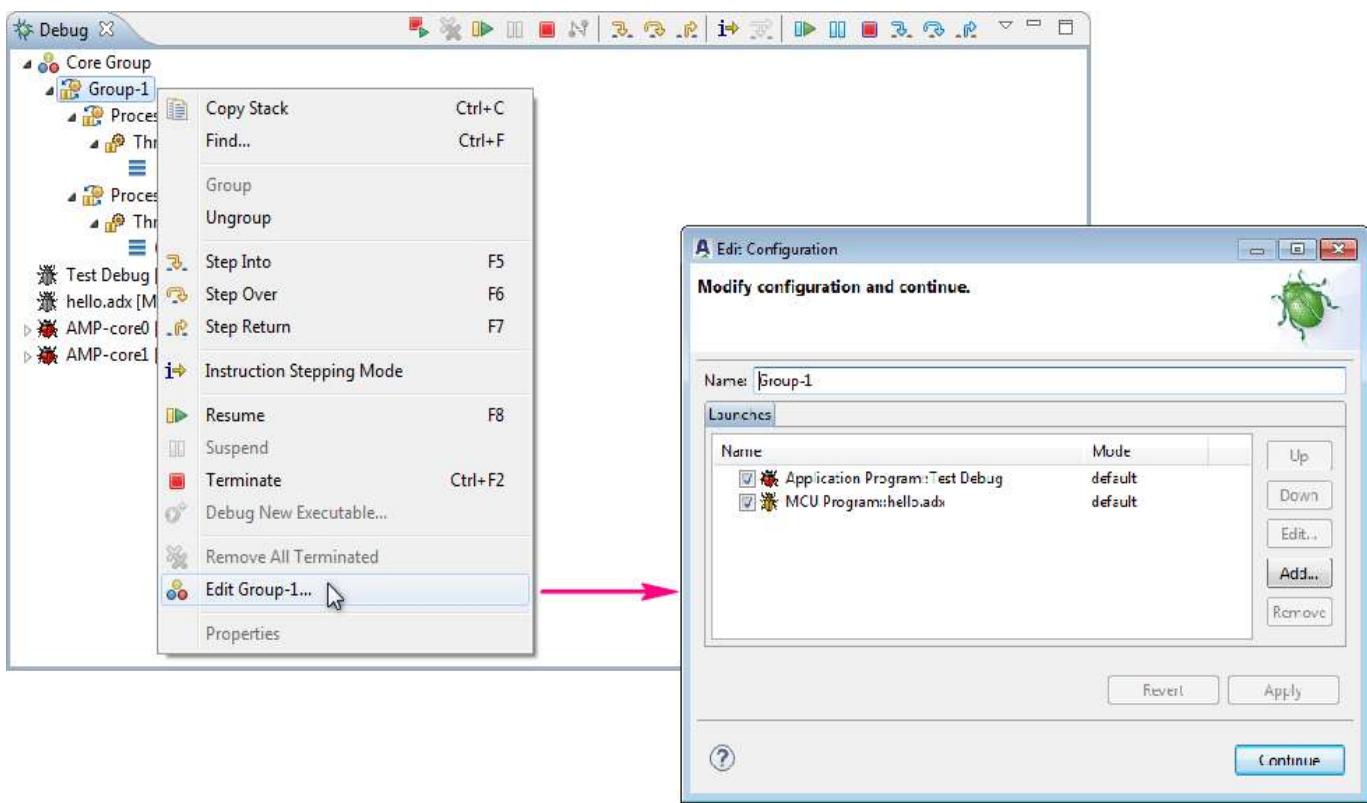
Take the above figure as an example. With Thread #1 of the process “Test Debug” and Thread #1 of the process “hello.adx” grouped together, you can have both threads resume the execution by selecting their group (“Group-1”) first and clicking the toolbar button  (Resume) or  (Group Resume).

Note that when only a single member thread is selected, commands can be sent to the core of that thread alone or all the execution cores inside the group depending on which set of toolbar buttons is used. The buttons  on the left of the toolbar will request the selected thread to resume/suspend/terminate/step into/step over/step return while the toolbar buttons  on the right will have the thread as well as its sibling threads in the group to perform the debug actions.

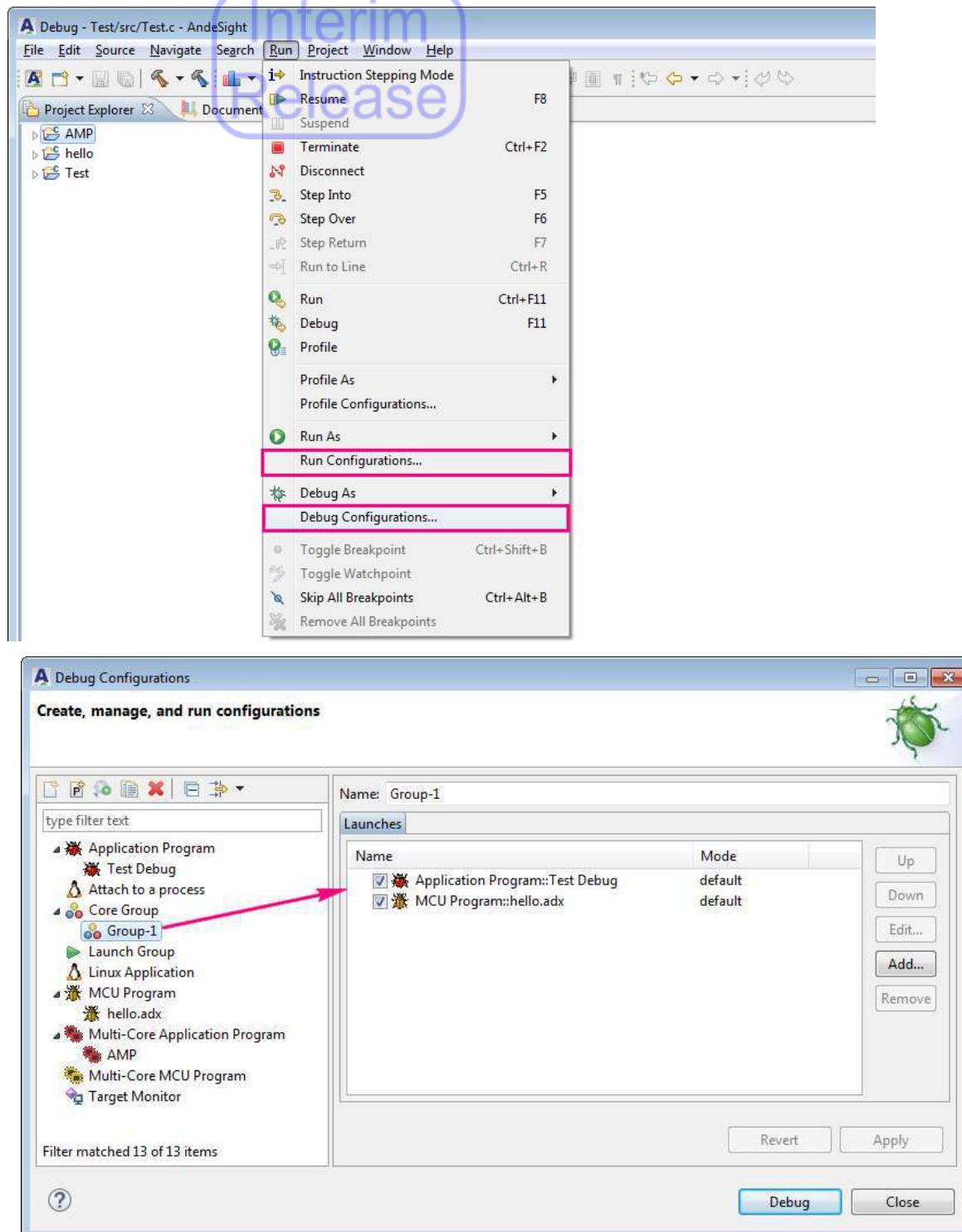
To ungroup threads/cores inside a group, simply select the group in the **Debug** view and right-click to select “Ungroup” in the pull-down menu.



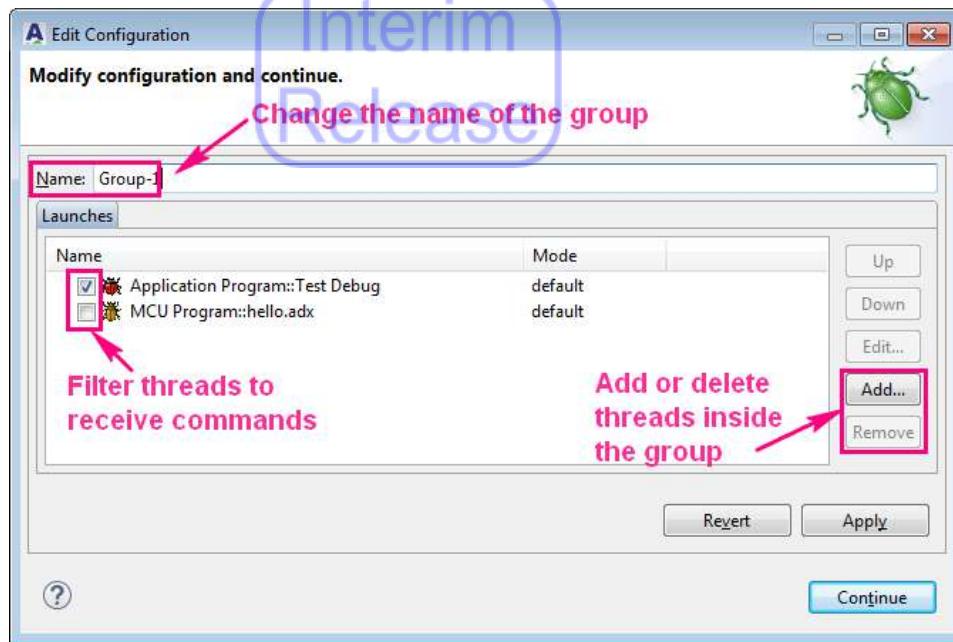
You can manage threads/cores in a group by editing the configuration of the group. To access the configuration, just select the desired group in the **Debug** view and right-click to select “Edit **GROUP\_NAME**” in the pull-down menu. A **Edit Configuration** dialog will then appear to outline the settings of the group.



You can also access the configuration of the group through the **Run/Debug Configurations dialog**, which can be invoked via “AndeSight main menu > Run > [Run|Debug] Configurations...”. In the navigation pane of the dialog, select the desired group under the Core Group node. The configuration settings of the group will then be displayed on the right.

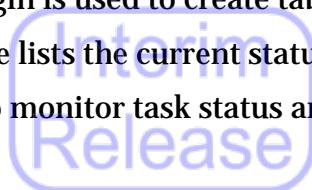


In the configuration settings of the group, you can rename the group, filter member threads/cores to receive commands (using check boxes), and add/delete threads/cores into/from the group (using the Add/Remove button). The changes on the configuration of the group will be realized in the **Debug** view after a debug session is relaunched for the group.



#### 2.4.12. RTOS awareness debugging

AndeSight supports RTOS awareness debugging on FreeRTOS or Zephyr executables. A script-based plugin is used to create tables and display RTOS information. The RTOS information table lists the current status of each task and internal information of each IPC, making it easy to monitor task status and OS kernel activities.



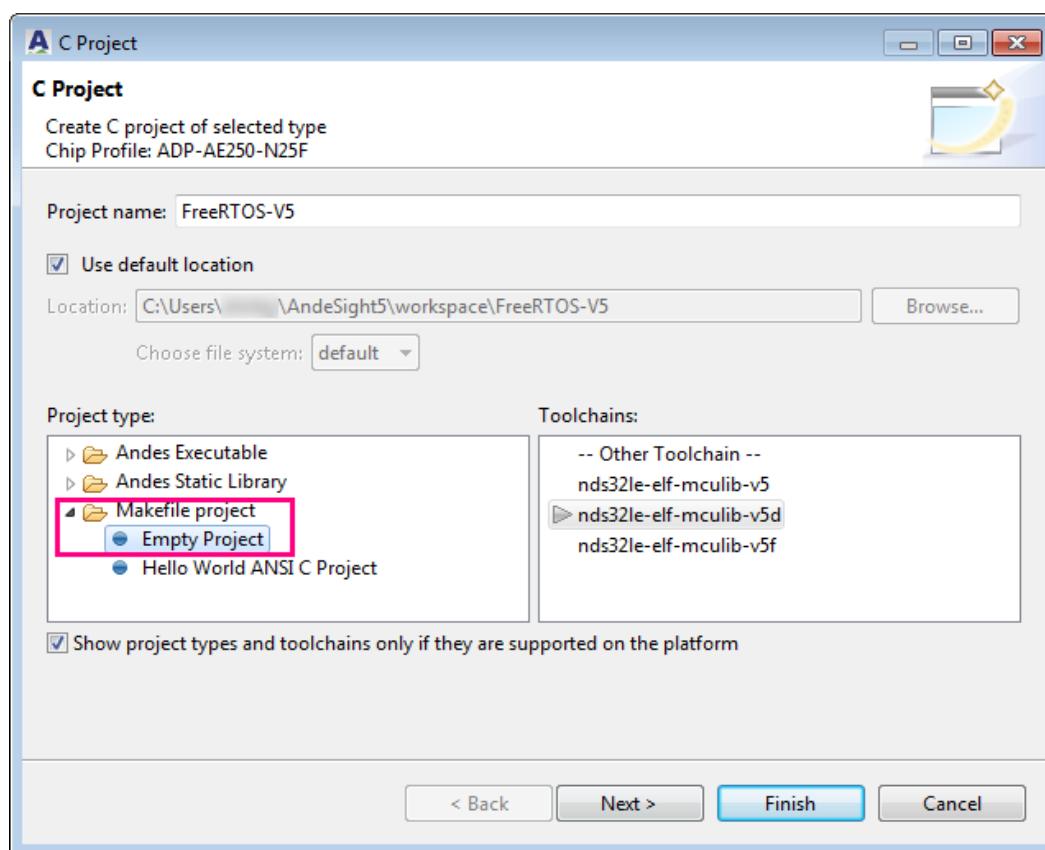
You may use FreeRTOS and Zephyr applications in AndeSight (i.e., [freertos-V5.tgz](#) and [zephyr-V5.tgz](#) in [ANDESIGHT\\_ROOT\RTOS\\_source](#)) to perform RTOS awareness debugging. For detailed instructions on the debugging procedure, please refer to Section 2.4.12.1 or Section 2.4.12.2.

To learn more about FreeRTOS and its features, please visit its [official website](#). For Zephyr feature and capabilities, reference the [official documentation](#) of Zephyr project.

#### 2.4.12.1 RTOS awareness debugging for FreeRTOS

This section outlines using AndeSight to perform RTOS awareness debugging for FreeRTOS. AndeSight provides compressed source code ([freertos-V5.tgz](#)) of two standard FreeRTOS demos, “Blinky” and “Full”. Make sure you use the Blinky application for RTOS awareness debugging as the Full demo is not suitable due to a strong real-time requirement.

**Step 1** Follow Section 2.1.1.1 to create a project with AndeSight. Make sure you choose “Makefile project > Empty Project” as the project type and select a pertinent toolchain. Click “Finish”.

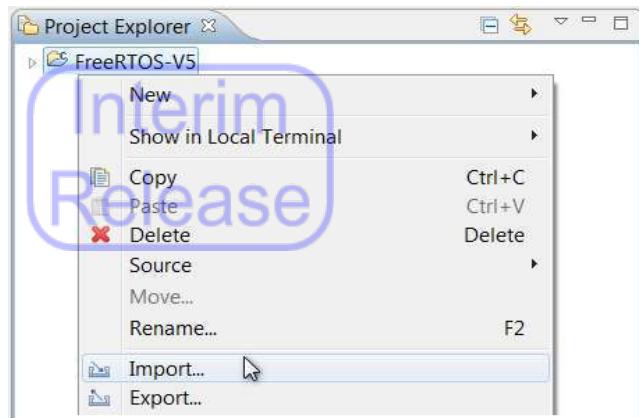



---

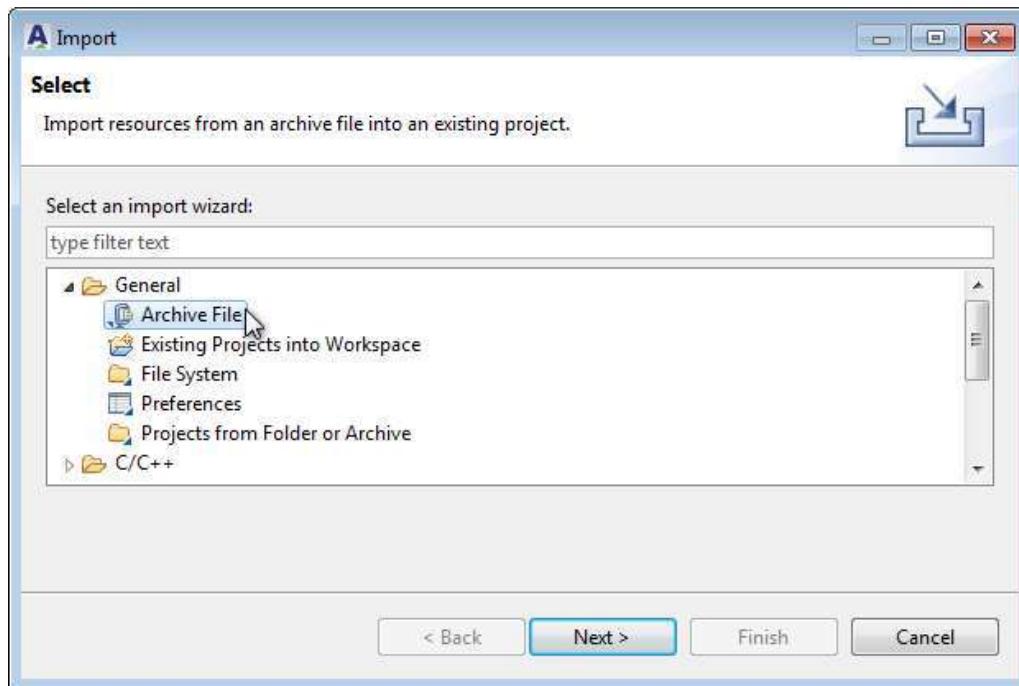
#### NOTE

To run the FreeRTOS demo on a Corvette-F1 target pre-integrated with fixed-configuration N22 RTL in Andes FreeStart program, make sure you specify the chip profile “ADP-Corvette-F1-N22” and select the toolchain “nds32le-elf-[newlib|mculib]-v5e”.

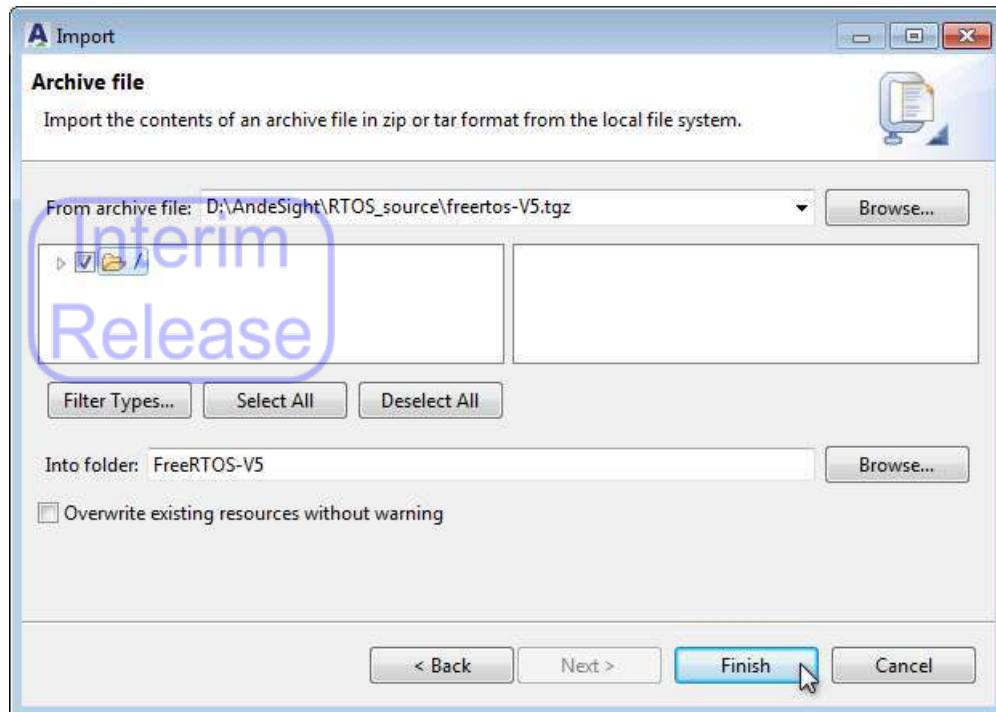
**Step 2** Find the created project in **Project Explorer**. Right-click the project folder and select “Import...” from the pull-down menu.



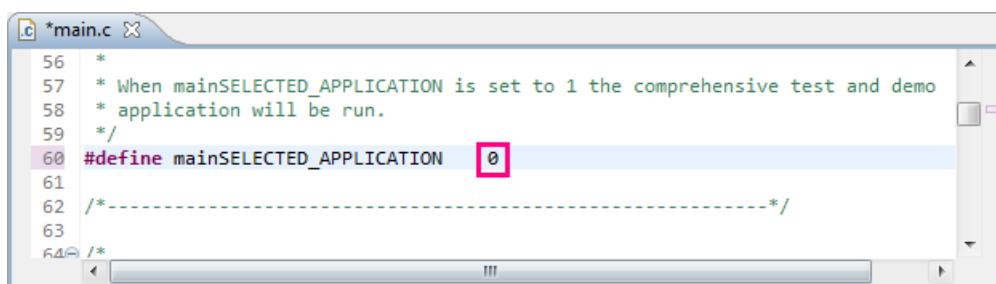
**Step 3** In the **Import** dialog, select “General > Archive File” and click “Next.”



**Step 4** Click “Browse...” to select the compressed Andes FreeRTOS source code “**freertos-V5.tgz**” for V5 targets from **ANDESIGHT\_ROOT\RTOS\_sources**. Click “Finish” to commence importing.

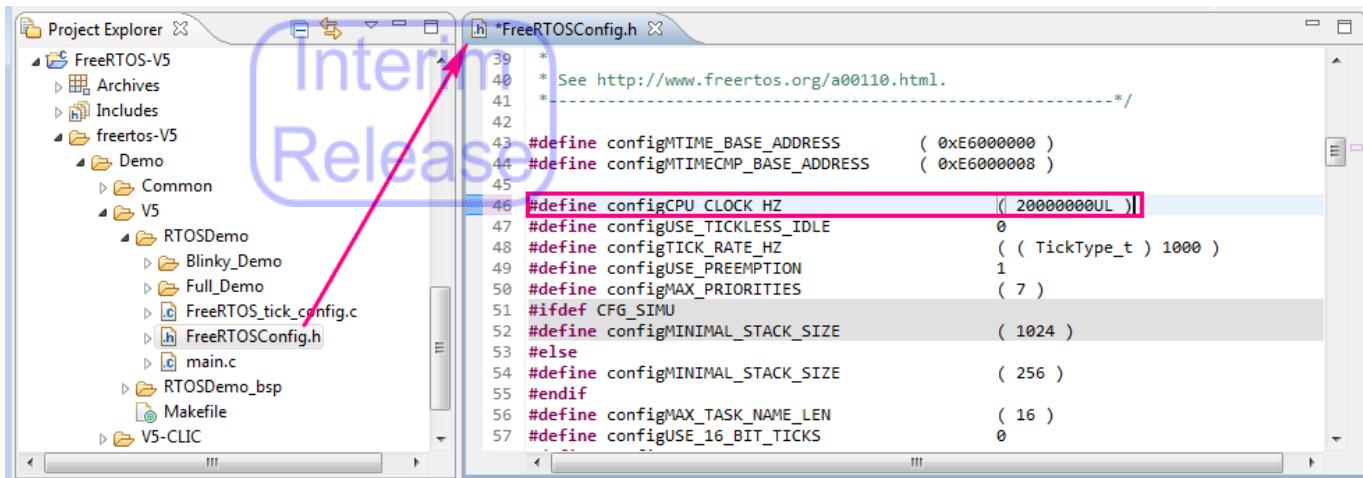


**Step 5** Specify the FreeRTOS Blinky application to be built by editing `main.c` under `PROJECT\freertos-V5\Demo\[V5|V5-CLIC]\RTOSDemo` and modifying the value of the definition “`mainSELECTED_APPLICATION`” to 0.



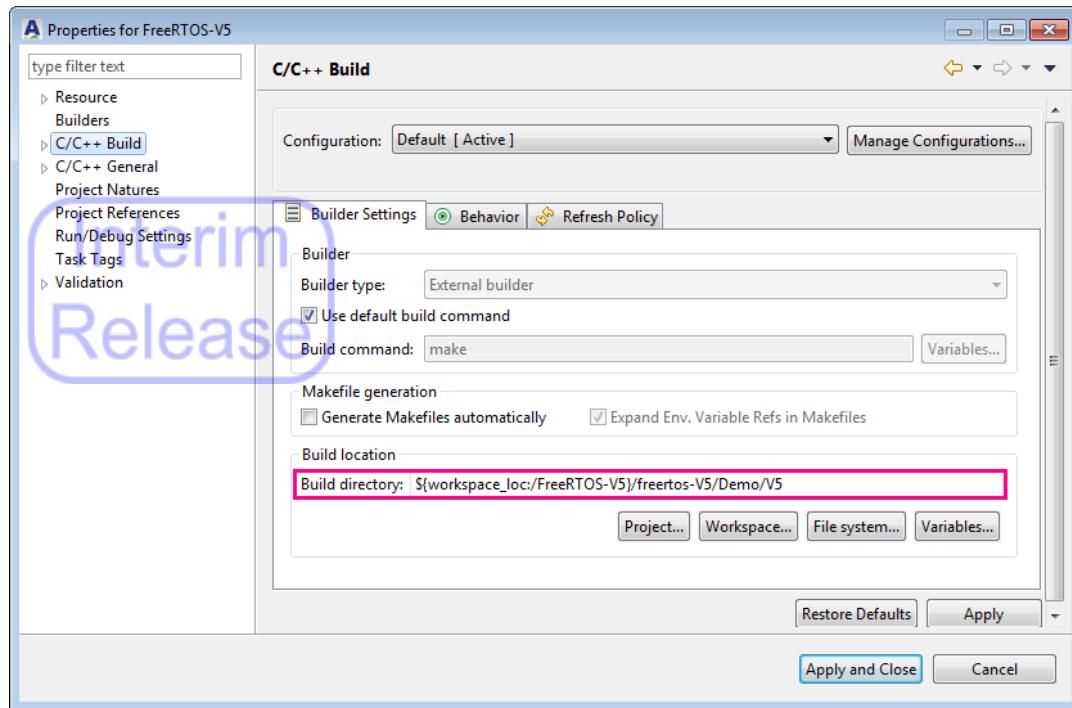
**Step 6** The CPU frequency required by the demo program is defined by `configCPU_CLOCK_HZ` in the configuration file `FreeRTOSConfig.h`, which is placed under `PROJECT\freertos-V5\Demo\[V5|V5-CLIC]\RTOSDemo`. Edit the value of `configCPU_CLOCK_HZ` if it is different from the CPU speed of your target, which can be obtained from boot messages or associated data sheet.

The following example is to change the configCPU\_CLOCK\_HZ value to **20000000UL** for running a FreeRTOS demo program on a ADP-XC7K target of AE250 platform pre-integrated with N22.



**Step 7** In **Project Explorer**, right-click the project folder and select “Properties” from the pull-down menu. In the **Properties** dialog, select “C/C++ Build” in the navigation pane and click “Builder Settings tab > Build location”. Change the build directory to the folder that Makefile resides as follows:

- For V5 targets pre-integrated with CLIC (Core-Local Interrupt Controller), change the build directory to  
 `${workspace_loc: /PROJECT/freertos-V5/Demo/V5-CLIC}`.
- For other V5 targets, change the build directory to  
 `${workspace_loc: /PROJECT/freertos-V5/Demo/V5}`, as shown below.



**Step 8** Click the **Behavior** tab and add the following commands in the Build and Clean field. Then, click “Apply”.

In the Build field, enter commands as follows to enable a verbose build using a 32-bit/64-bit toolchain, specify the platform of your target, and generate an executable:

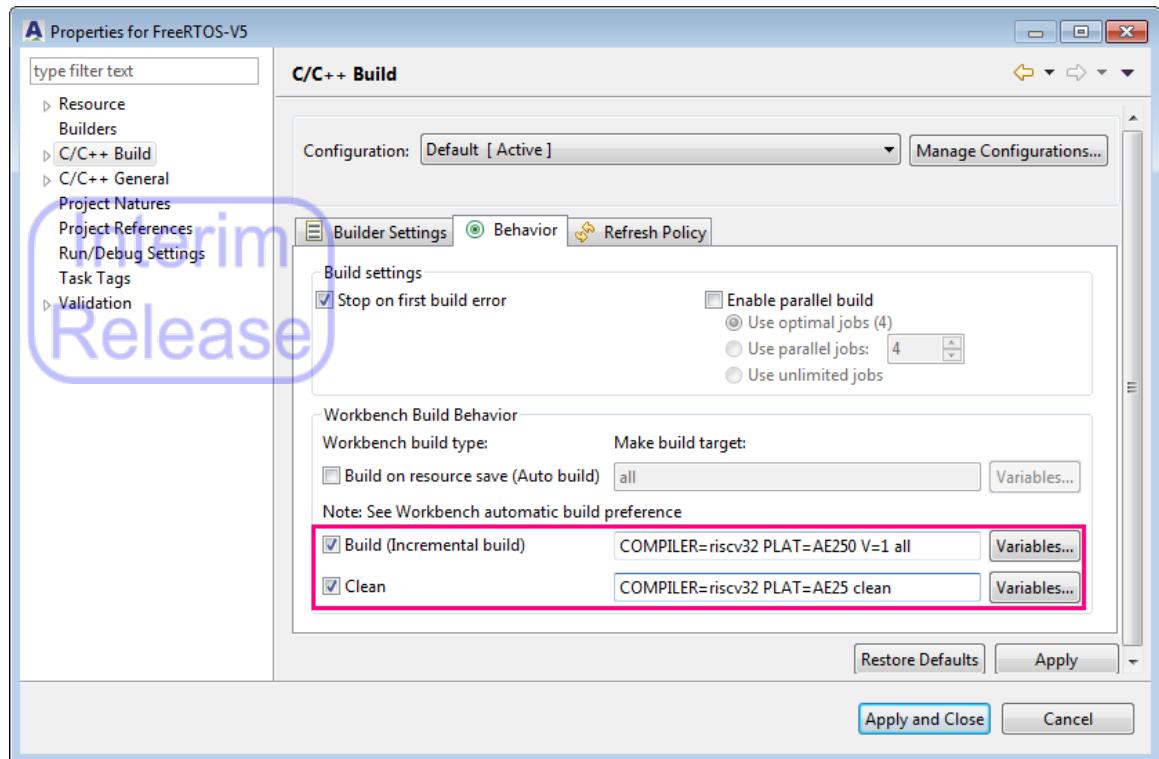
```
COMPILER=[ri scv32|ri scv64|ri scv32-ll vm|ri scv64-ll vm]
PLAT=[AE250|AE350|CF1-AE250] V=1 all
```

For an AE350 target, make sure you also specify **USE\_CACHE=1** to enable caching.

In the Clean field, enter the following:

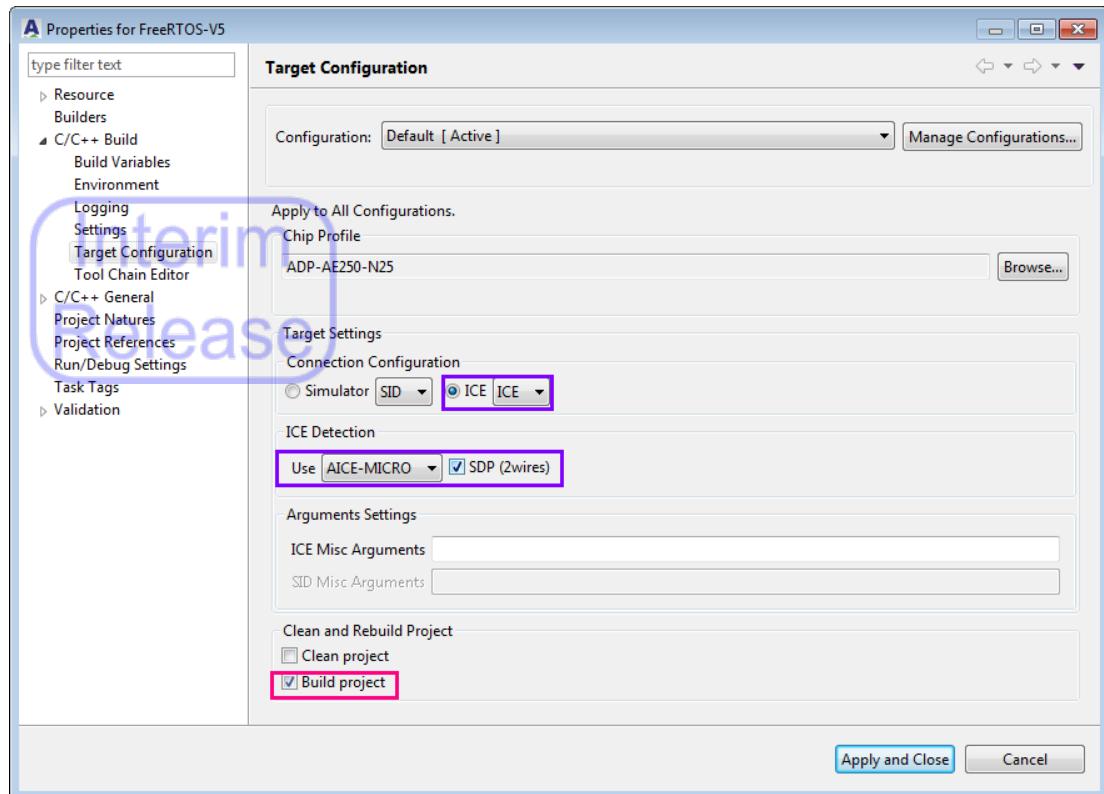
```
COMPILER=[ri scv32|ri scv64|ri scv32-ll vm|ri scv64-ll vm]
PLAT=[AE250|AE350|CF1-AE250] clean
```

For example, enter build and clean commands as follows for an ADP-AE250-N25F target:



**Step 9** In the **Properties** dialog, click “C/C++ Build > Target Configuration” in the navigation pane to select the option “Build project” on the Target Configuration page.

To run the demo on a target board with 2-wire debug interface in conjunction with the ICE device AICE-MICRO or AICE-MINI+, make sure to select “ICE” as the connection configuration and check the option “SDP (2 wires)” in the ICE detection section.



To run the demo on a Corvette-F1 simulator target pre-integrated with fixed-configuration N22 RTL in Andes FreeStart program, make sure the connection configuration “SID” is selected and enter the following in the SID Misc Arguments field as well:

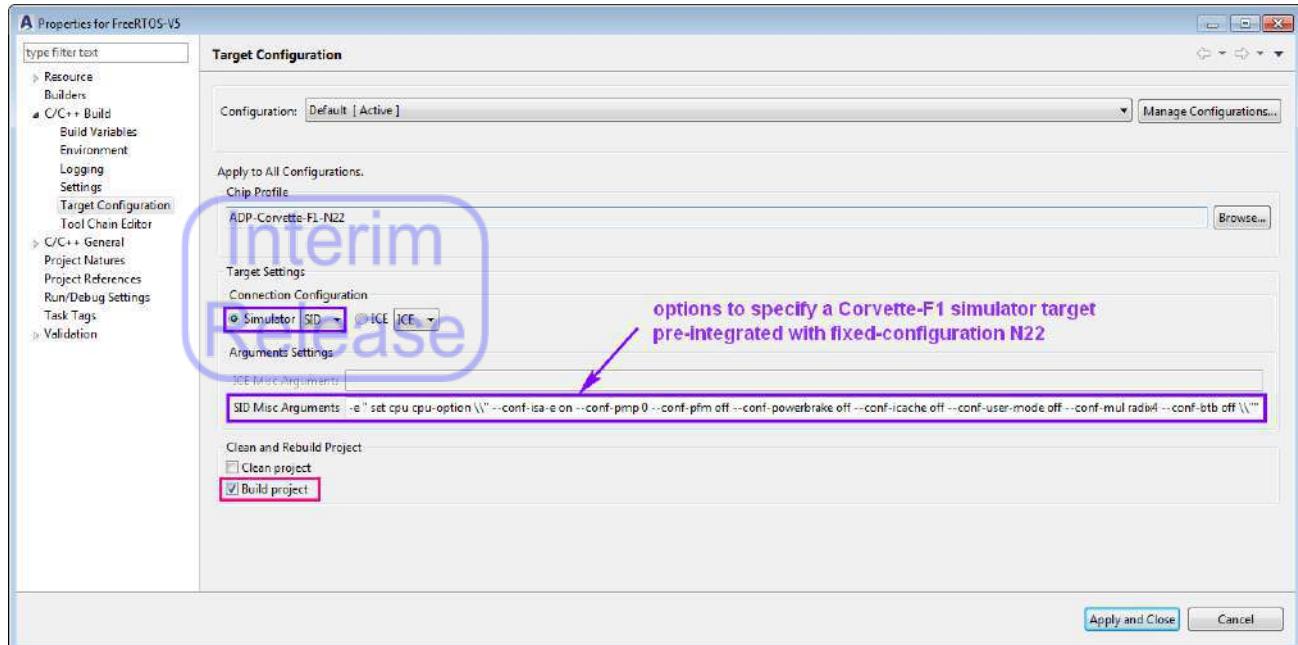
```
-e " set cpu cpu-option \\\" --conf-is-a-e on --conf-pmp 0 --conf-pfm
off --conf-powerbrake off --conf-i-cache off --conf-user-mode off
--conf-mul radix4 --conf-btb off \\\""
```

---

#### NOTE

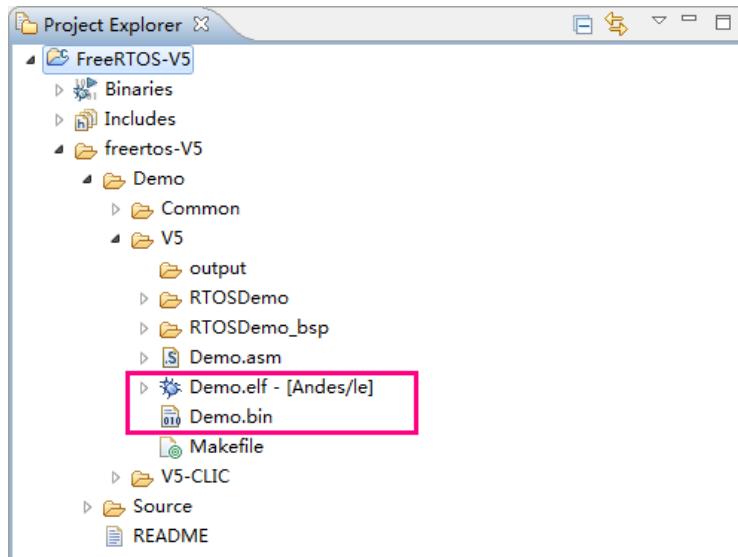
The above commands are for the Windows environment. For Linux, replace the escape character “\\” with “\”.

---

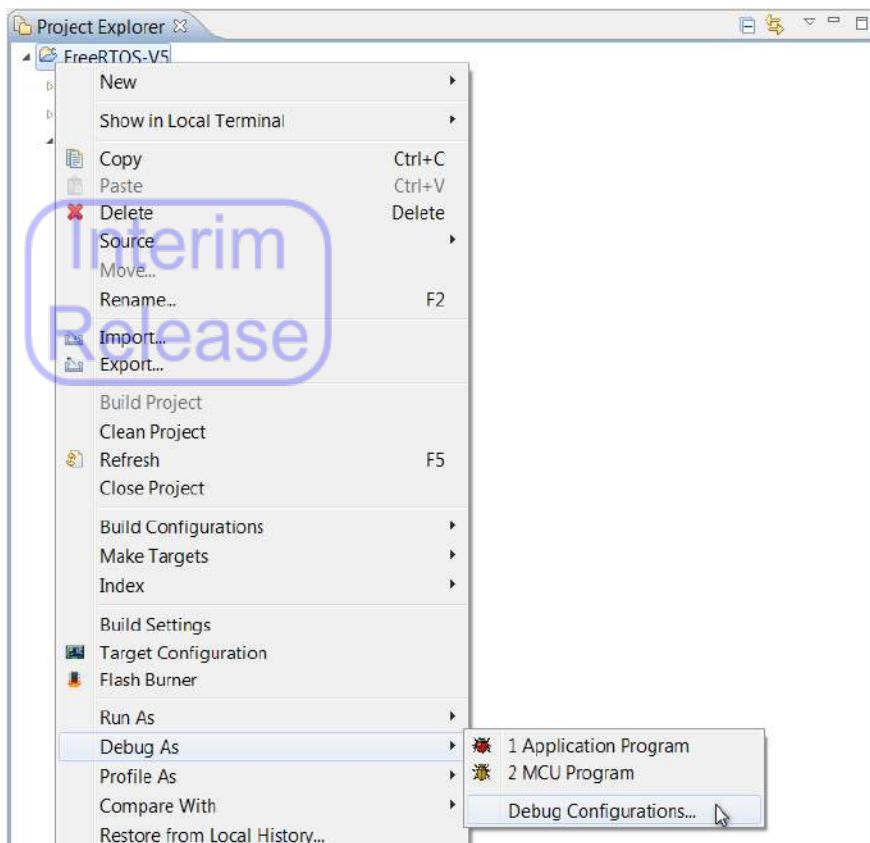


Click “OK” on the **Properties** dialog.

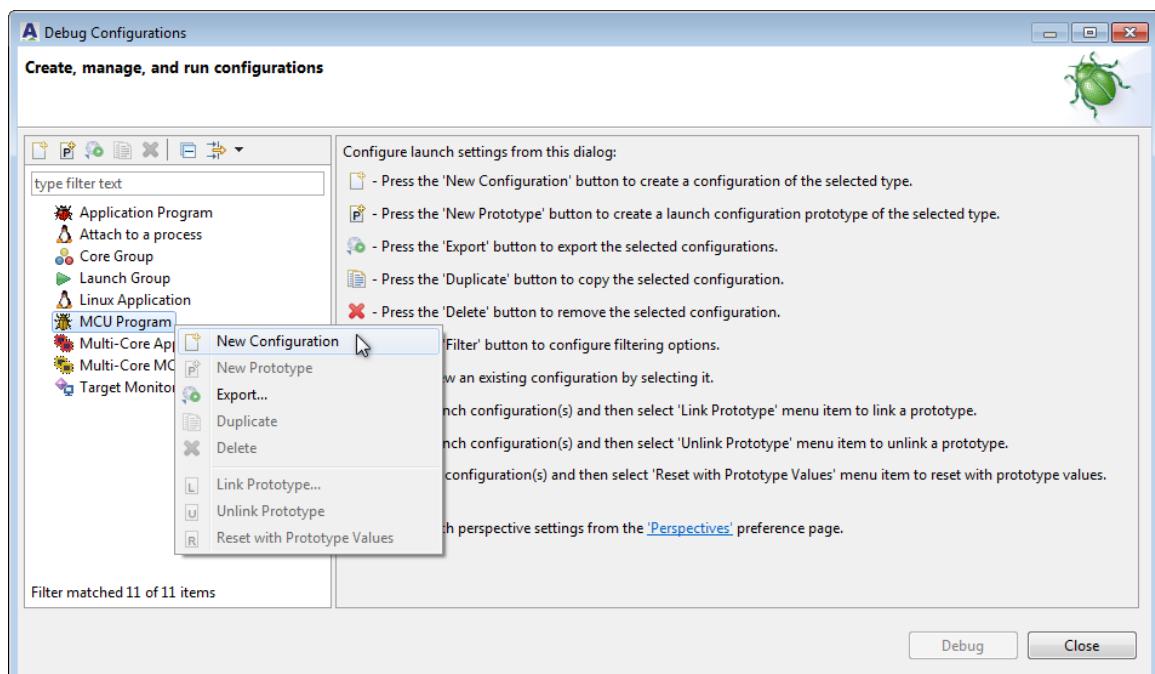
**Step 10** In the **Project Explorer** view, verify that **Demo.elf** and **Demo.bin** are built under the build directory specified in Step 7.



**Step 11** Right-click the project folder and select “Debug As > Debug Configurations...” from the pull-down menu.

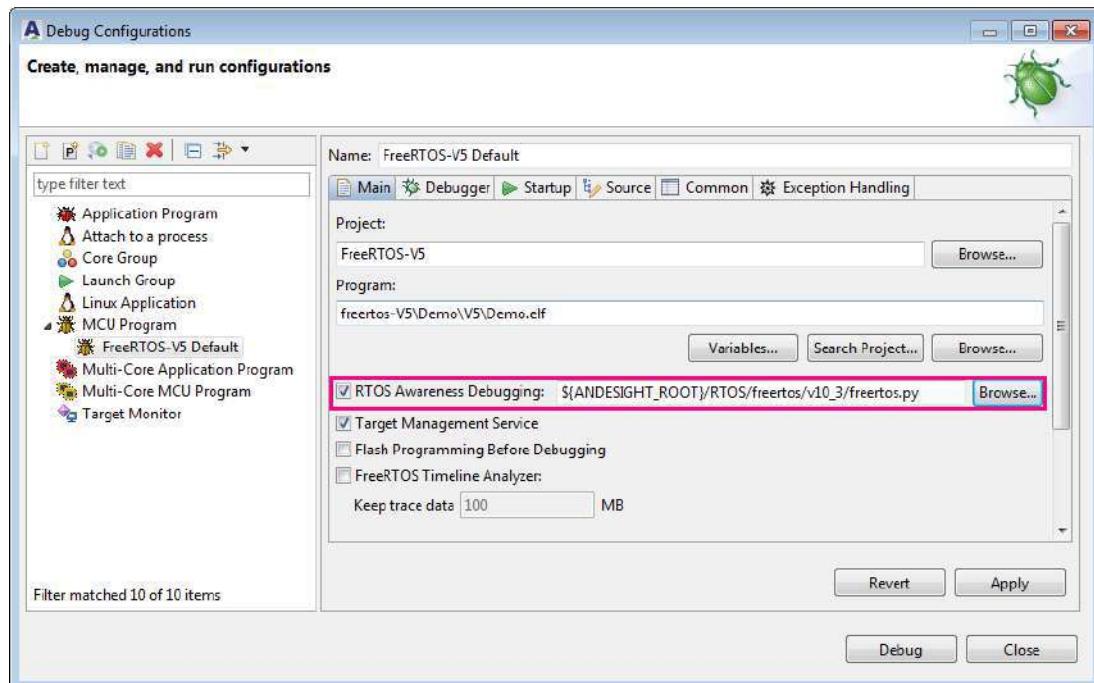


**Step 12** In the invoked **Debug Configurations** dialog, right-click “MCU Program”, and select “New Configuration” from the pull-down menu to create a debug configuration.



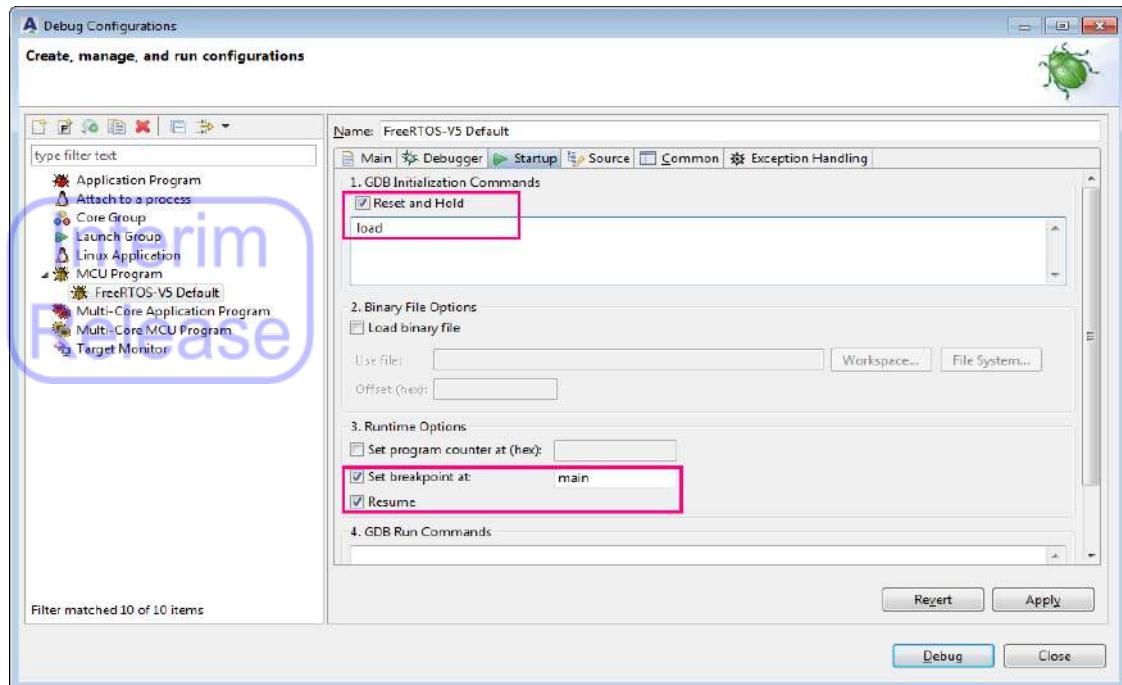
**Step 13** In the **Main** tab of the **MCU Program** debug configuration, select the “RTOS Awareness Debugging” option to enable OS awareness debugging. This option auto-detects python script under the project folder. Note that the RTOS Awareness auto-detection may be affected if `--gc-sections` is applied.

You can also click “Browse...” to specify a python script (`freertos.py`) corresponding to your version of FreeRTOS from `ANDESIGHT_ROOT\RTOS\FreeRTOS\VERSION\`. Debugging FreeRTOS demos from AndeSight v5.0 requires that you select the python script for FreeRTOS v10.3.

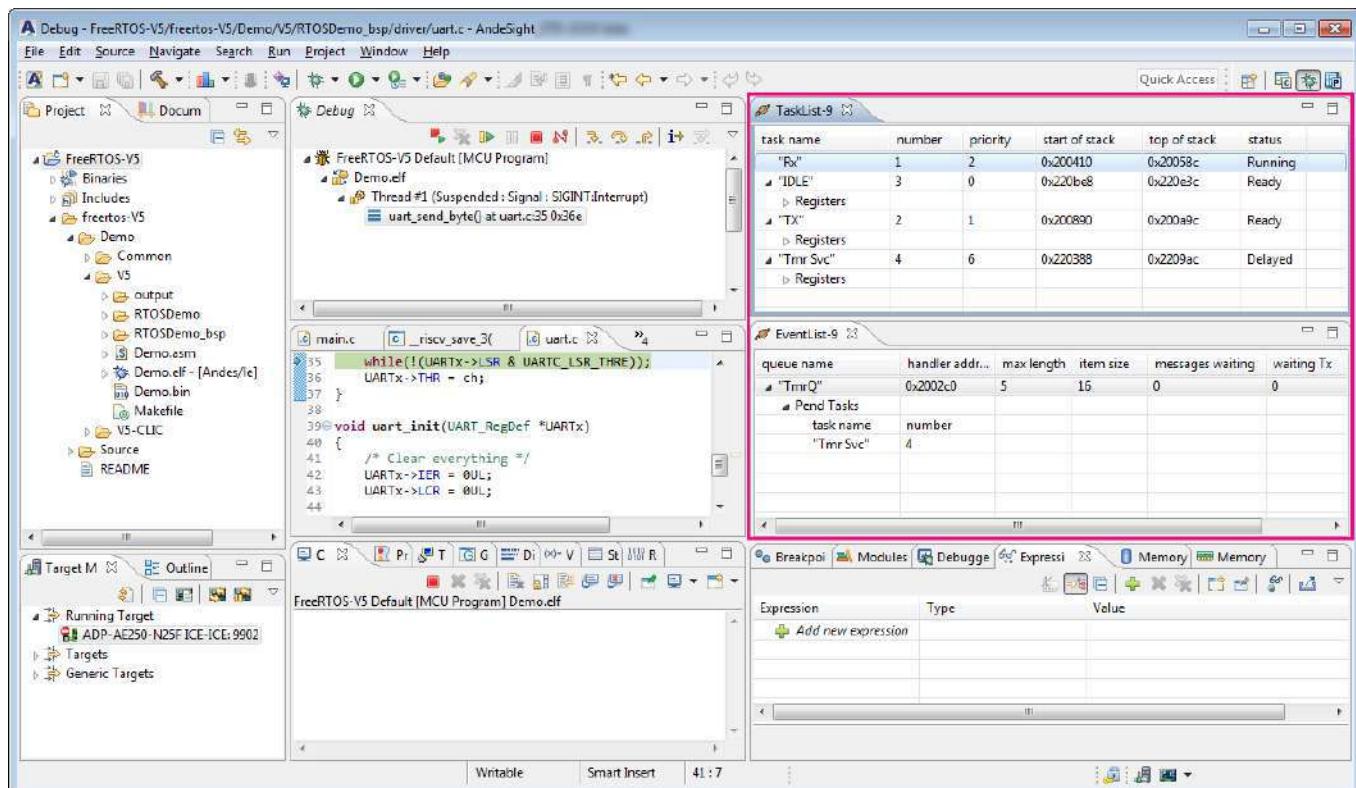


**Step 14** In the **Startup** tab of the **MCU Program** debug configuration, configure the following settings and click “Apply” and “Debug” to launch a debug session. The **Debug** perspective is invoked automatically.

- In the GDB Initialization Commands section, check the “Reset and Hold” option and enter the command “`load`”.
- In the Runtime Options section, set a breakpoint at the `main` function and select the “Resume” option.



**Step 15** Proceed with debugging by clicking (Resume), (Suspend), (Step Into) or (Step Over) on the toolbar of the Debug view and observe the system resources in the Event List and Task List views.

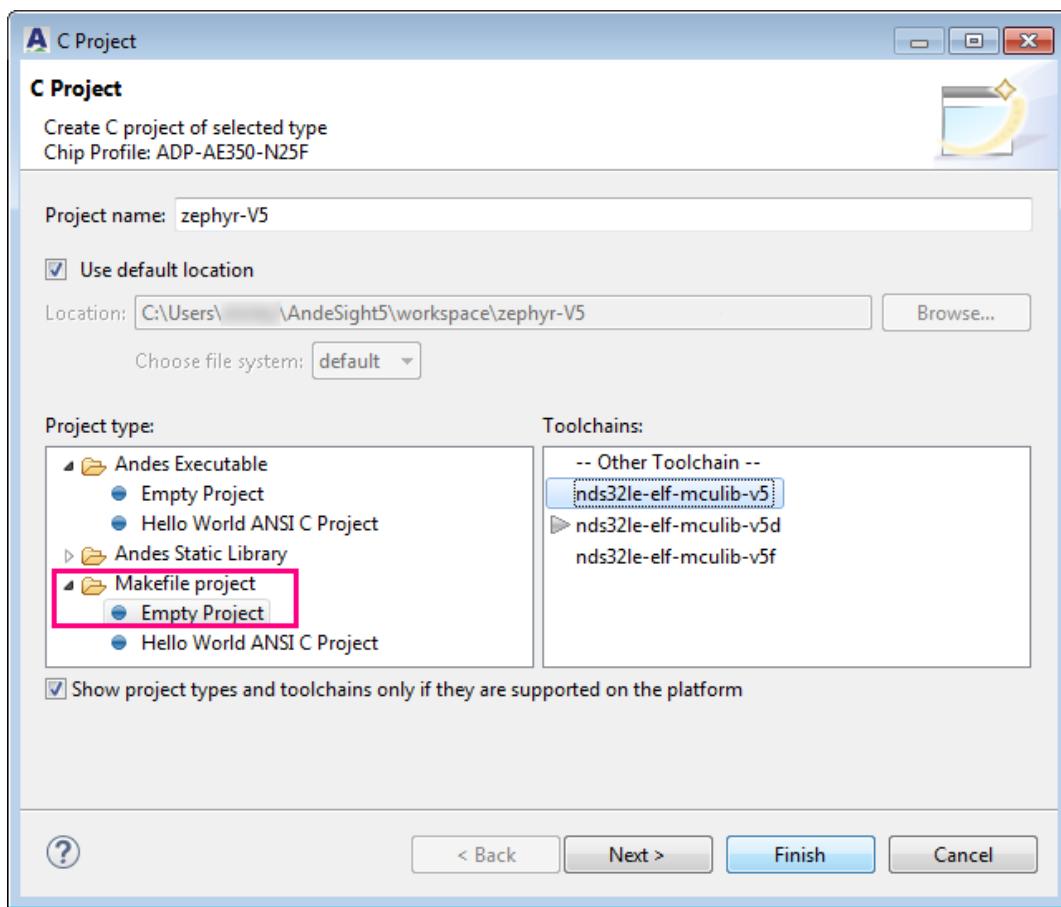


### 2.4.12.2 RTOS awareness debugging for Zephyr

This section outlines using AndeSight to perform RTOS awareness debugging for Zephyr.

AndeSight provides Zephyr source package ([zephyr-V5.tgz](#)) including sample applications for targets of ISA V5. For details about Zephyr samples, please refer to the [Zephyr official website](#).

**Step 1** Follow Section 2.1.1.1 to create a project with AndeSight. Make sure you choose “Makefile project > Empty Project” as the project type and select a pertinent toolchain. Click “Finish”.



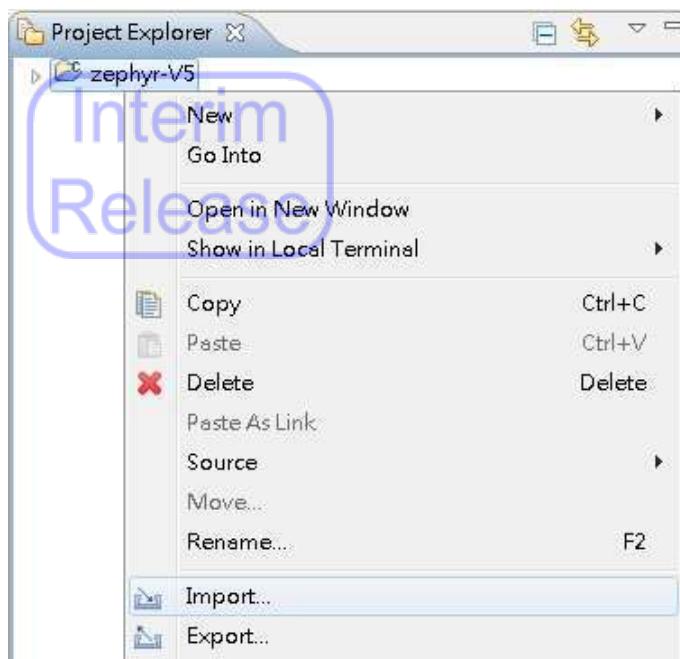

---

#### NOTE

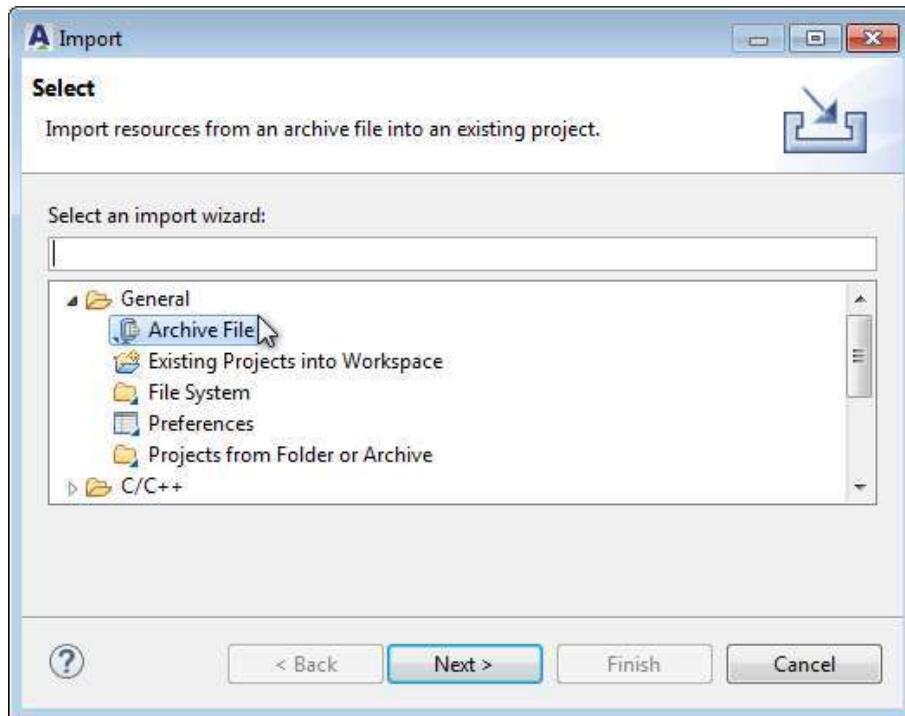
Zephyr only supports standard RISC-V PLIC (Platform Level Interrupt Controller), so make sure do not run it on the ICE/SID target which uses CLIC (Core Level Interrupt Controller).

---

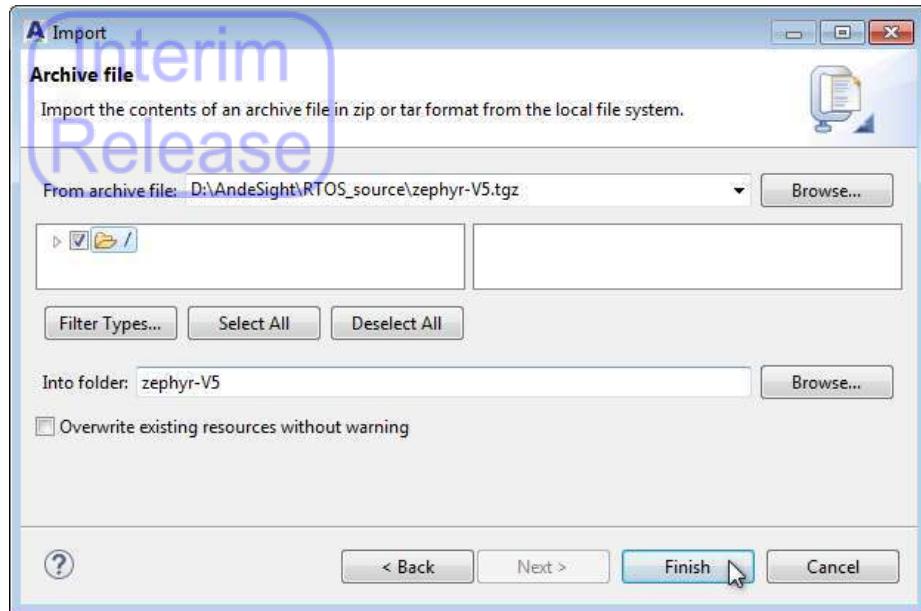
**Step 2** Find the created project in **Project Explorer**. Right-click the project folder and select “Import...” from the pull-down menu.



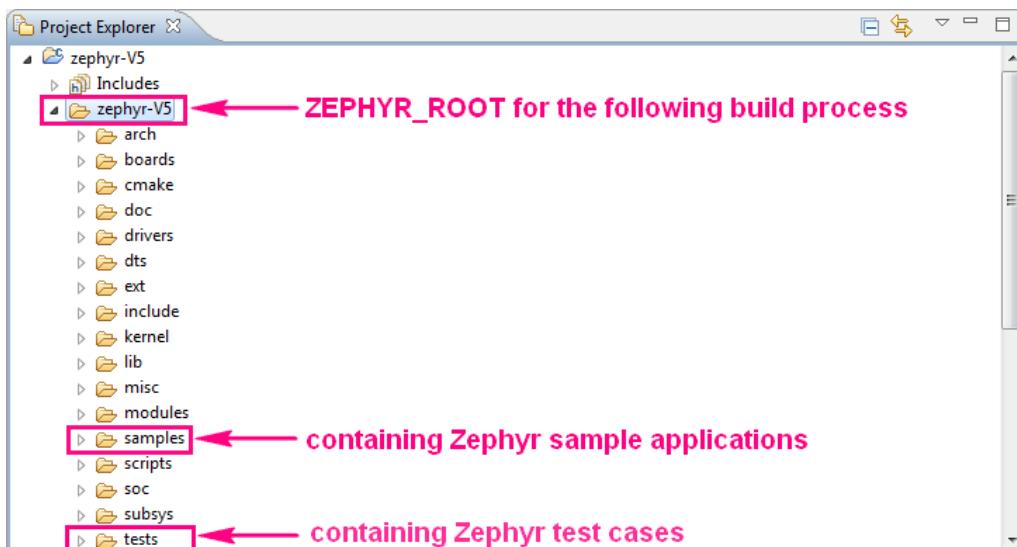
**Step 3** In the **Import** dialog, select “General > Archive File” and click “Next.”



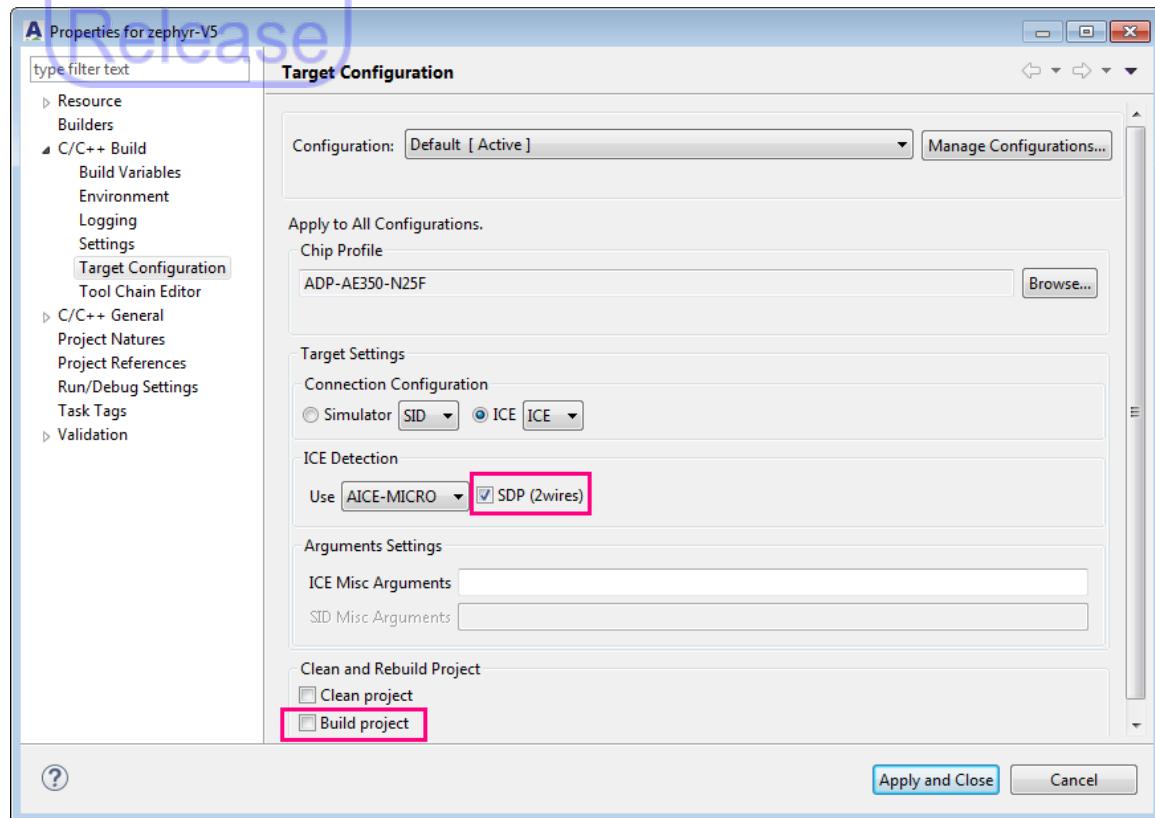
**Step 4** Click “Browse...” to select the compressed Andes Zephyr source code “**zephyr-V5.tgz**” for V5 targets from **ANDESIGHT\_ROOT\RTOS\_sources**. Click “Finish” to commence importing.



In **Project Explorer**, a directory containing Zephyr sources is generated under the project you created. This directory serves as the base directory for the following build steps and is indicated as **ZEPHYR\_ROOT**. The source code of Zephyr sample applications and test cases can be found in **ZEPHYR\_ROOT\samples** and **ZEPHYR\_ROOT\tests** respectively.



**Step 5** Right-click the project in **Project Explorer** and select “Target Configuration” from the pull-down menu. In the invoked dialog, uncheck the option “Build project”. If you are using an ICE target with 2-wire debug interface connected to the ICE device AICE-MICRO or AICE-MINI+, check the option “SDP (2 wires)” on the dialog. Click “Apply and Close”.



**Step 6** Build a Zephyr application in a command line environment:

1. On Windows, invoke the cygwin environment from the AndeSight package. On Linux, reference the "[Getting Started Guide](#)" on the Zephyr official website to set up a Zephyr development environment.
2. Change the current directory to **ZEPHYR\_ROOT** in workspace, set environment variables and specify an adequate toolchain to build Zephyr. Note that Zephyr applications can only be built with GCC compiler in elf toolchains. The following example is to specify

the toolchain “nds32le-elf-mculib-v5” for the build.

```
$ cd ZEPHYR_ROOT
$ source zephyr-env.sh
$ export ZEPHYR_TOOLCHAIN_VARIANT='cross-compile'
$ export CROSS_COMPILE=ANDESIGHT_ROOT/toolchains/nds32le-elf-mculib-v5
/bin/riscv32-elf-
```

3. Change the current directory to the base directory of a desired Zephyr application, **ZEPHYR\_APP**. For example, in the case of the mutex API test sample, **ZEPHYR\_APP** refers to **ZEPHYR\_ROOT/tests/kernel/mutex/mutex\_api**.

```
$ cd ZEPHYR_APP
```

4. Create a build directory in **ZEPHYR\_APP** and change the current directory to it.

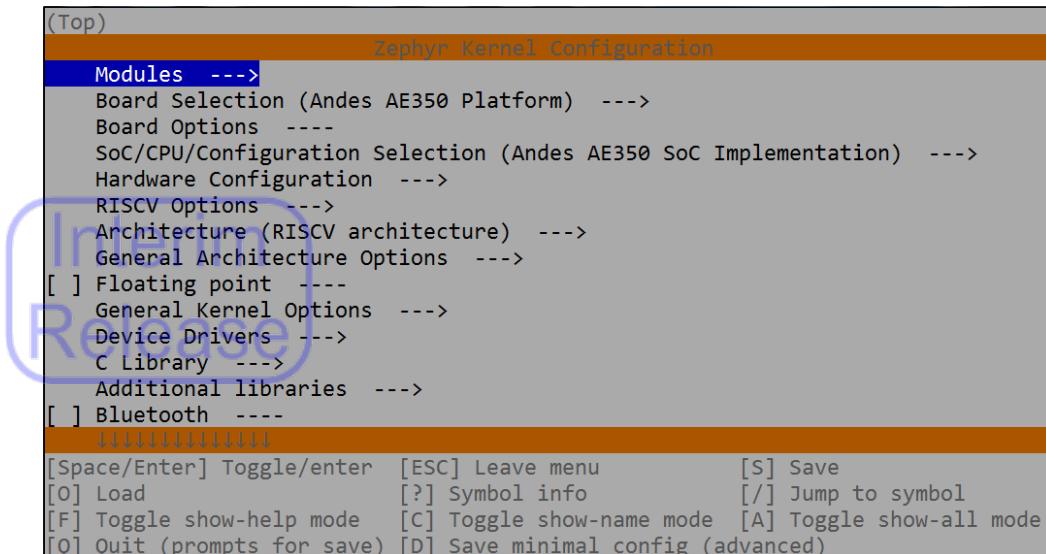
```
$ mkdir build && cd build
```

5. Execute the **cmake** command and specify an Andes platform (i.e., AE250, AE350 or Corvette-F1) to generate build files in the build directory.

```
$ cmake -DBOARD=[ae250|ae350|cf1_ae250] .. /
```

6. Issue as follows to open an interactive interface and temporarily change the kernel configuration in the application build directory (i.e., **ZEPHYR\_APP/build/zephyr/.config**):

```
$ make menuconfig
```



7. Enable the configurations for kernel debugging through the following menuconfig paths:

```
- Option for kernel debugging  
(Top) -> Debugging Options -> Kernel object tracing  
(Top) -> General Kernel Options -> Kernel Debugging and Metrics -> Thread monitoring  
(Top) -> General Kernel Options -> Kernel Debugging and Metrics -> Thread name
```

8. Skip this step if the default kernel configuration for the board and the sample application will be used directly. In the case that the kernel configuration needs to be modified to meet your Andes V5 platform, specify options associated with Andes V5 hardware through the following menuconfig paths:

- Option for supporting RV32/RV64 kernel
  - (Top) -> Hardware Configuration -> CPU Architecture of SoC
- Option for supporting RISC-V hard float
  - (Top) -> Floating point
  - (Top) -> RISCV Options -> RISCV Processor Options -> Use double precision floating point (needs RISC-V 'D' extension)
- Option for supporting RV32E
  - (Top) -> RISCV Options -> RISCV Processor Options -> Use RV32 Embedded ISA, a reduced version of RV32I with only 16 registers
- Option for other HW configuration
  - (Top) -> Hardware Configuration -> Enable cache
  - (Top) -> Hardware Configuration -> Support Hardware DSP
  - (Top) -> Hardware Configuration -> Support performance throttling
- Option for SMP configuration
  - (Top) -> General Kernel Options -> SMP Options -> Use new-style \_\_arch\_switch instead of \_\_swap
  - (Top) -> General Kernel Options -> SMP Options -> Enable symmetric multithreading support
  - (Top) -> General Kernel Options -> SMP Options -> Number of CPUs/cores

For more information about the kernel configuration options, refer to the following page on the Zephyr official website:  
<https://docs.zephyrproject.org/2.0.0/application/index.html#kernel-configuration>



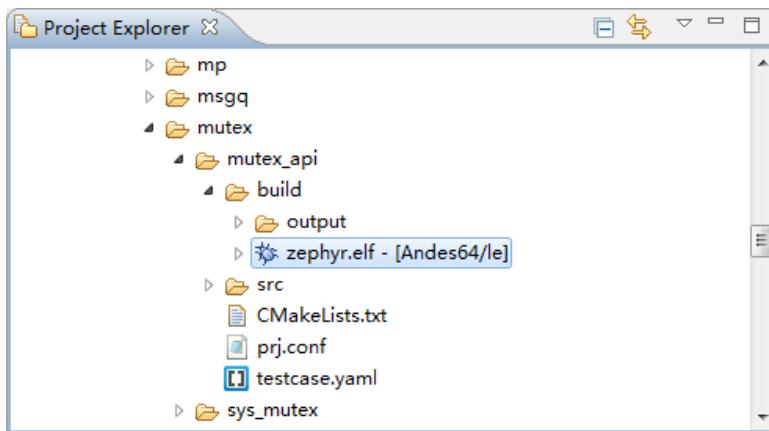
If you want to change the kernel configuration for the application permanently, follow instructions in the following page:  
<https://docs.zephyrproject.org/2.0.0/application/index.html#setting-application-configuration-values>

#### 9. Execute the `make` command to build the application image.

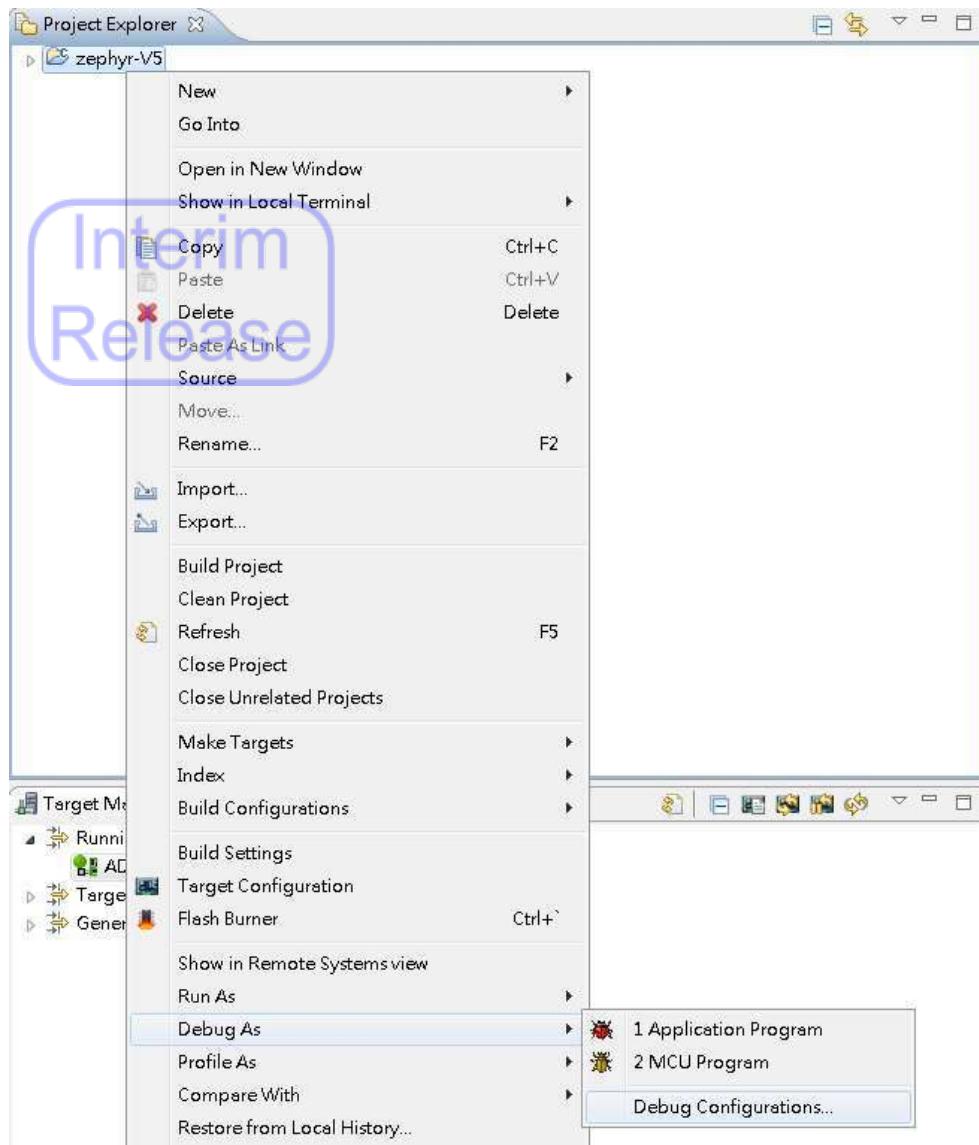
`$ make`

The Zephyr sample application `zephyr.elf` is generated in `ZEPHYR_APP/build/zephyr/`.

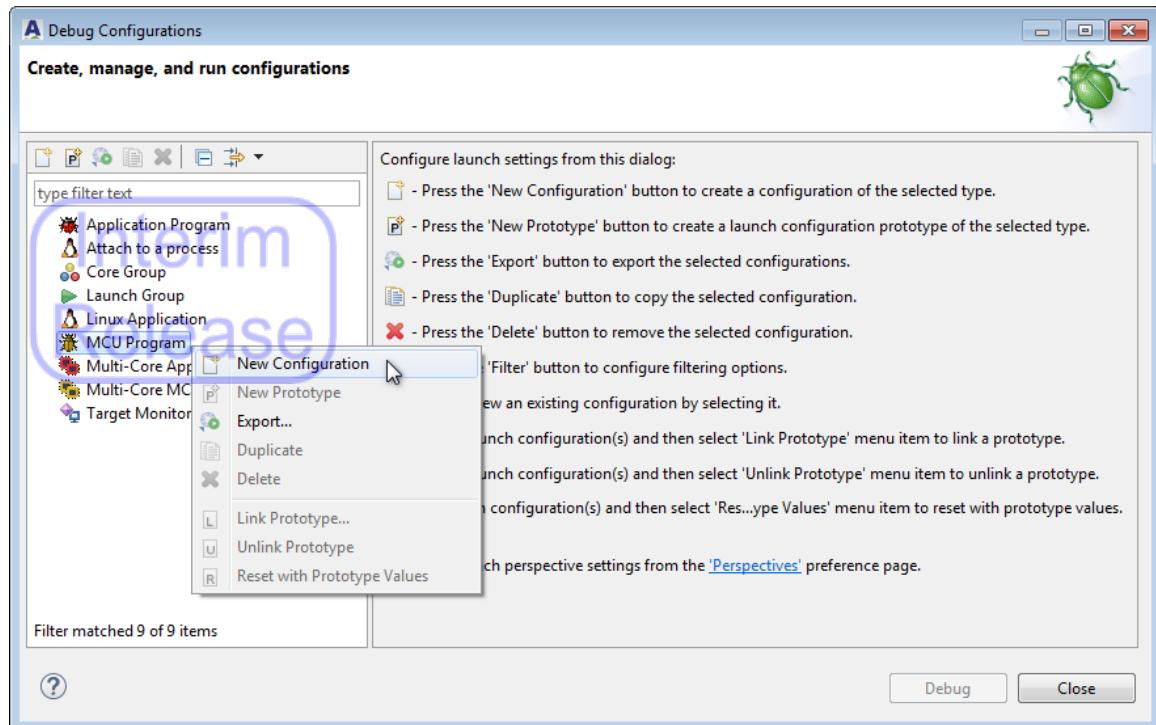
**Step 7** Switch back to the AndeSight environment. In the **Project Explorer** view, verify that the ELF executable `zephyr.elf` is built under the build directory you specified in the Zephyr project.



**Step 8** Right-click the project folder and select “Debug As > Debug Configurations...” from the pull-down menu.

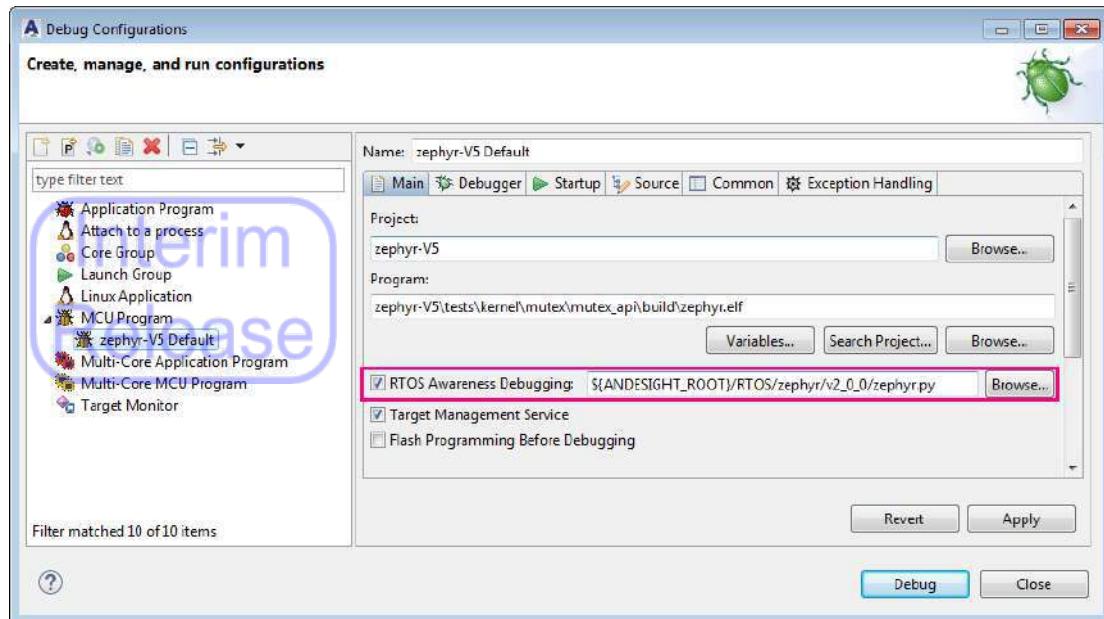


**Step 9** In the invoked **Debug Configurations** dialog, right-click “MCU Program” and select “New Configuration” from the pull-down menu to create a debug configuration.



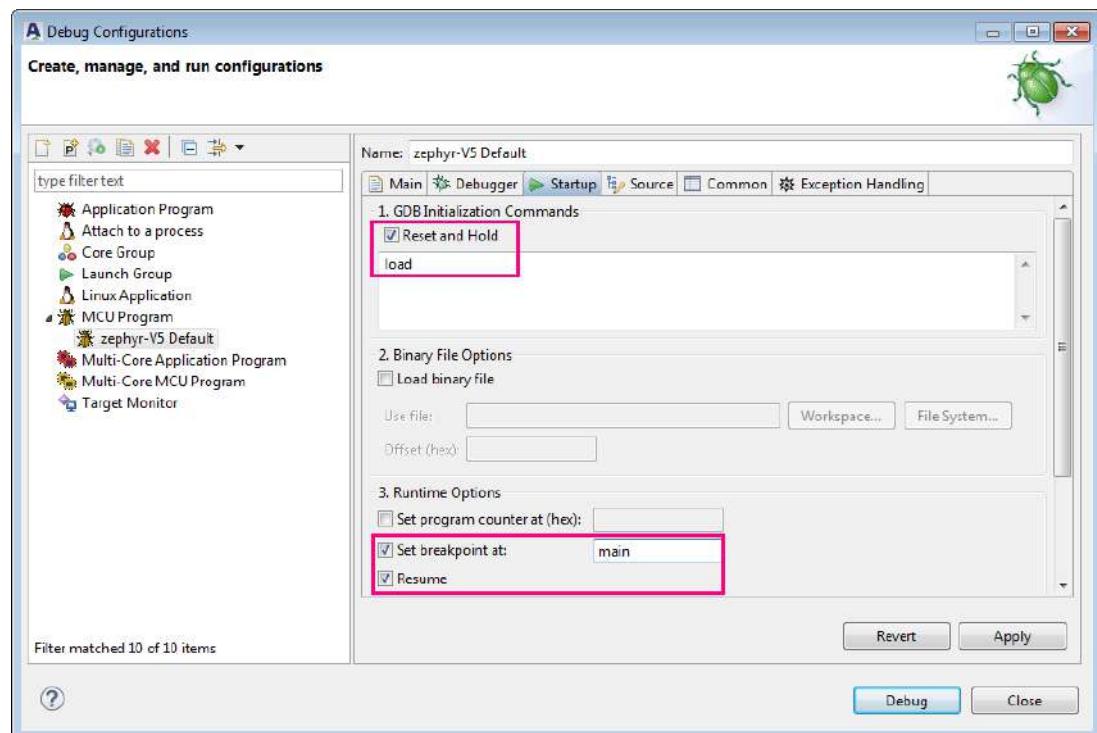
**Step 10** On the **Main** tab of the **MCU Program** debug configuration, select the “RTOS Awareness Debugging” option to enable OS awareness debugging. This option auto-detects python script under the project folder. Note that the RTOS Awareness auto-detection may be affected if `--gc-sections` is applied.

You can also click “Browse...” to specify a python script (`zephyr.py`) corresponding to your version of Zephyr from `ANDESIGHT_ROOT\RTOS\zephyr\VERSION\`. Debugging Zephyr samples from AndeSight v5.0 requires a python script for Zephyr v2.0.0.

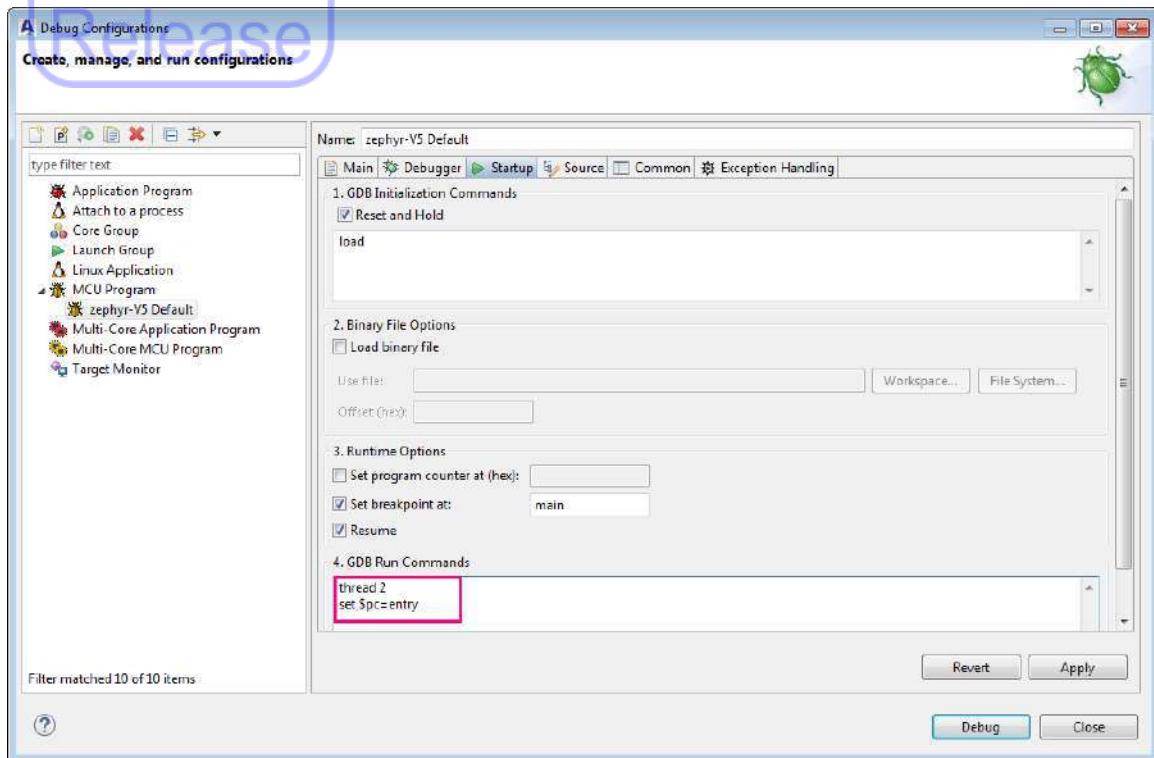


## Step 11 Configure the settings in the Startup tab as follows:

- In the GDB Initialization Commands section, check the “Reset and Hold” option and enter the command “**load**”.
- In the Runtime Options section, set a breakpoint at the main function and select the “Resume” option.

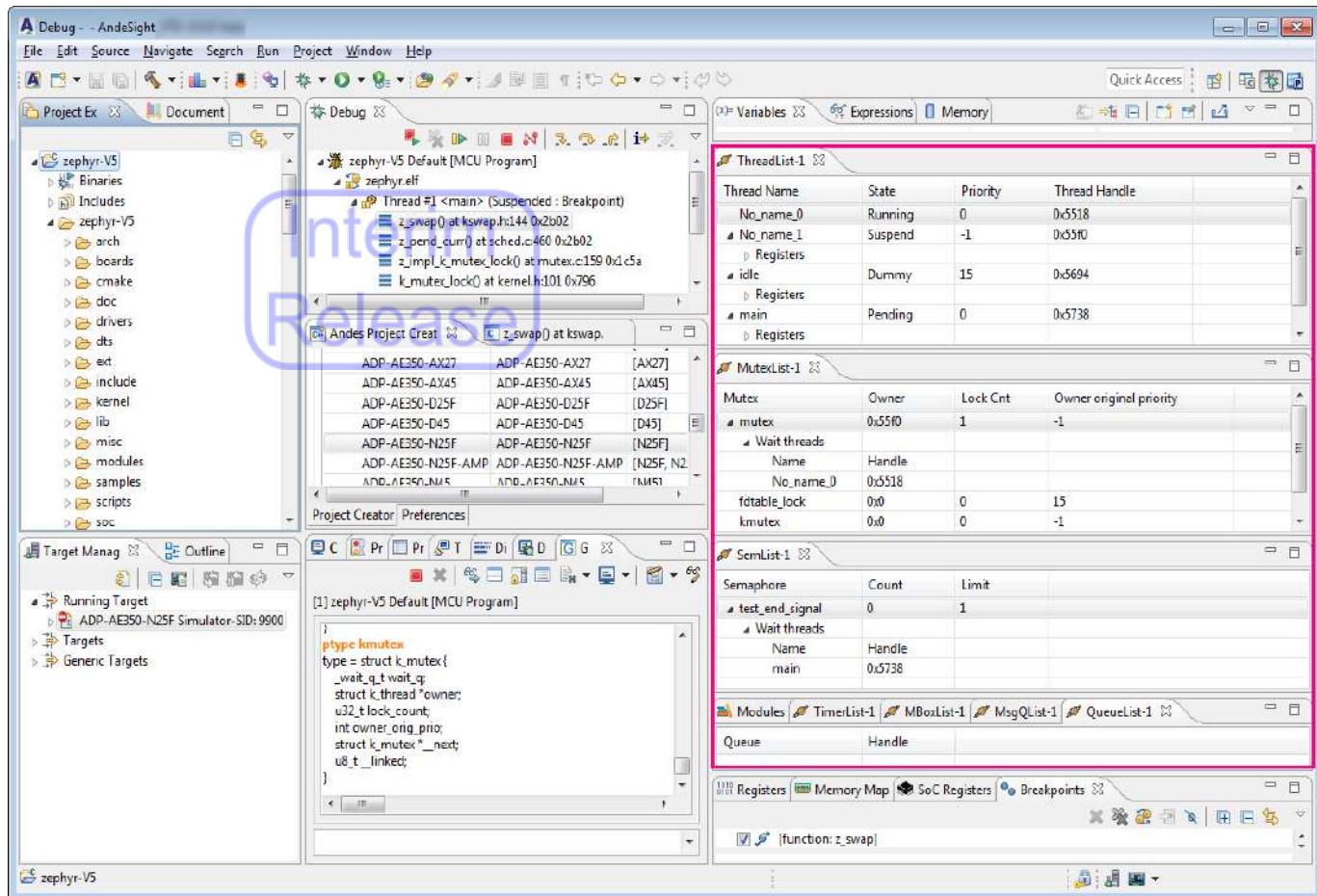


If you are going to run the application on a SMP system, enter commands in the GDB Run Commands to set the program counter of cores other than core 1 to the start address of the program executable, which is at the symbol “`entry`” for Zephyr applications. To switch control among cores, use the command “`thread`”.



**Step 12** In the **Debug Configurations** dialog, click “Apply” and “Debug” to launch a debug session. The **Debug** perspective is invoked automatically.

**Step 13** Proceed with debugging by clicking  (Resume),  (Suspend),  (Step Into) or  (Step Over) on the toolbar of the **Debug** view and observe the system resources in the **Thread List**, **Sem List**, **Mutex List**, **Timer List**, **MBox List**, **MsgQ List**, and **Queue List** views.



### 2.4.12.3 RTOS applications for FreeRTOS

#### Blinky Demo

This demo includes two tasks, one software timer, and one queue. One task in conjunction with the software timer repeatedly sends values of 100 and 200 to the other task via the queue. The sending task writes to the queue every 200 milliseconds and the software timer writes to the queue every 2 seconds. The receiving task prints a message to the UART port each time it receives the values.

The output message of this demo application is as follows:

```
Bl i nky Demo
Message received from task
Message received from software timer
Message received from task
Message received from task
Message received from task
Message received from task
```

#### Full Demo

Supported only by ICE targets, this demo application tests the RTOS port and illustrates the following:

- Creation of RTOS objects using statically and dynamically allocated memory
- Direct to task notifications
- Event groups
- Software timers
- Queues
- Semaphores
- Mutexes

In addition to these standard demo tasks, two other tasks are also created in the demo.

- Register test tasks – Register test tasks test the RTOS context switch mechanism by filling each Andes CPU register (including floating point registers) with a known, unique value, then repeatedly verifying that the value originally written to the register remains in the register throughout the lifetime of the task.
- Check task – The check task periodically (every 5 seconds) queries standard demo tasks and register test tasks to ensure they are functioning as intended. It then prints a status message to the UART port.

The output message of this demo application is as follows:

#### Full Demo

```
Pass, status code = 0, tick count = 5000
Pass, status code = 0, tick count = 10000
Pass, status code = 0, tick count = 15000
Pass, status code = 0, tick count = 20000
Pass, status code = 0, tick count = 25000
Pass, status code = 0, tick count = 30000
Pass, status code = 0, tick count = 35000
Pass, status code = 0, tick count = 40000
Pass, status code = 0, tick count = 45000
Pass, status code = 0, tick count = 50000
Pass, status code = 0, tick count = 55000
Pass, status code = 0, tick count = 60000
```

#### 2.4.12.4 RTOS applications for Zephyr

For detailed introduction to Zephyr sample applications, please visit the [Zephyr official website](#).

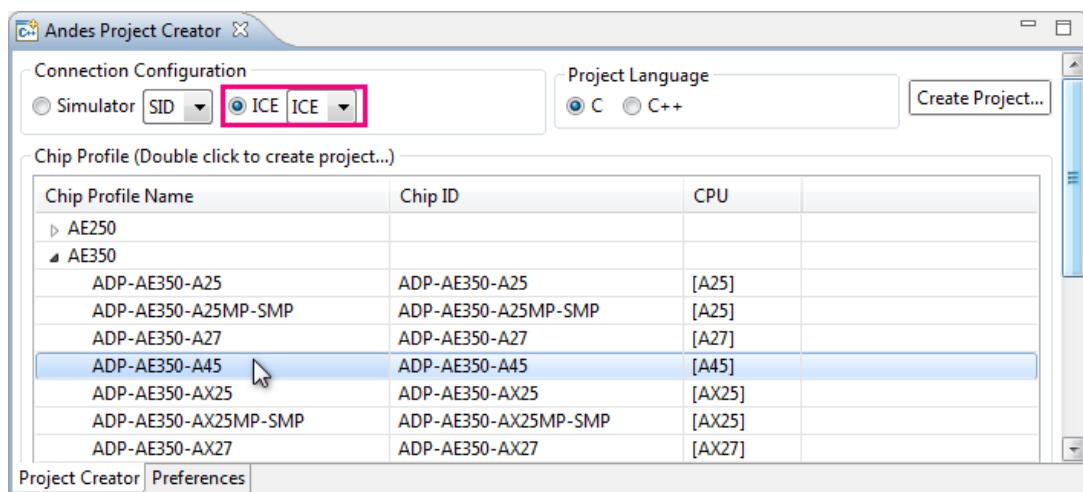
### 2.4.13. Linux kernel debugging

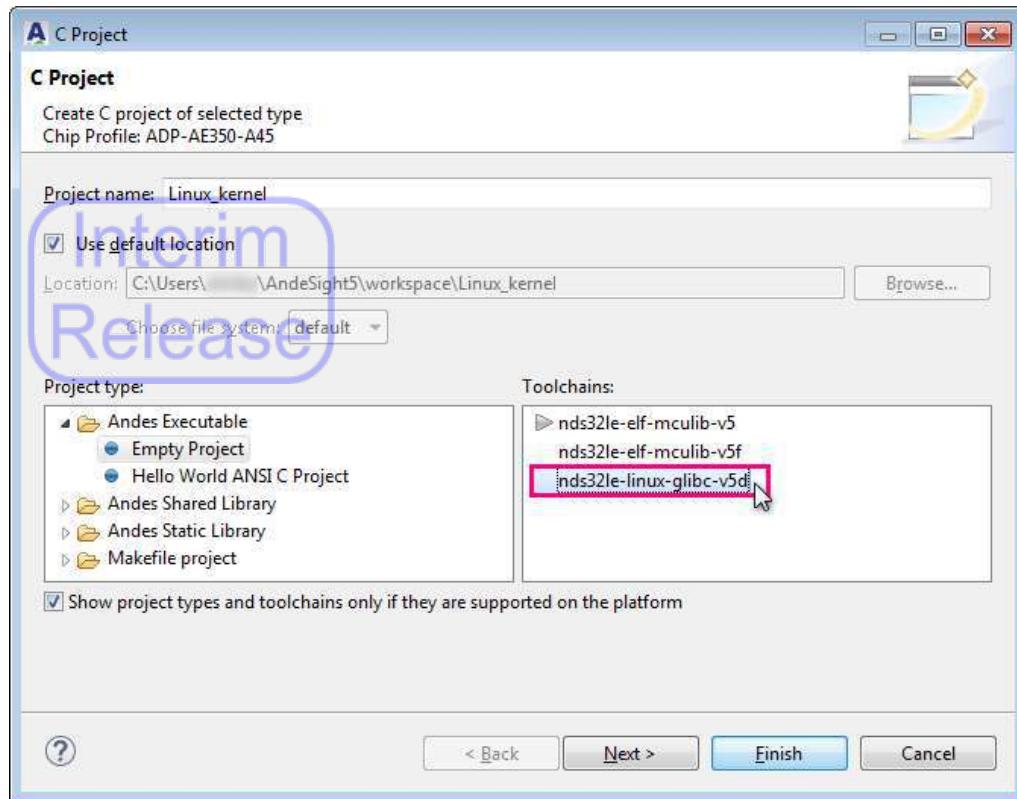
You can use AndeSight IDE to debug a Linux kernel on a V5 target system. What you need to do first is to create a C project that contains files associated with the Linux kernel, including the Linux kernel source, the source package of the M-mode firmware OpenSBI, the ELF and raw files of the kernel image (vmlinux and Image) and OpenSBI image (fw\_jump.elf and fw\_jump.bin), and a device tree blob (DTB) file for booting the Linux kernel. Then, you can launch a MCU Program debug session for the project and perform regular debugging tasks.

For the procedure to boot up the Linux kernel and preparations before that, please refer to *Linux Development User Manual*.

#### 2.4.13.1 Debugging Linux kernel with AndeSight

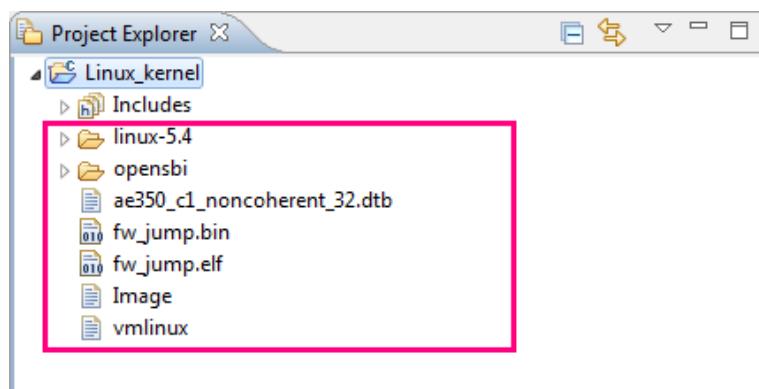
**Step 1** Follow Section 2.1.1.1 to create a project with AndeSight. Make sure you choose “ICE” as the connection configuration and “Andes Executable > Empty Project” as the project type and select a pertinent Linux toolchain. Click “Finish.”



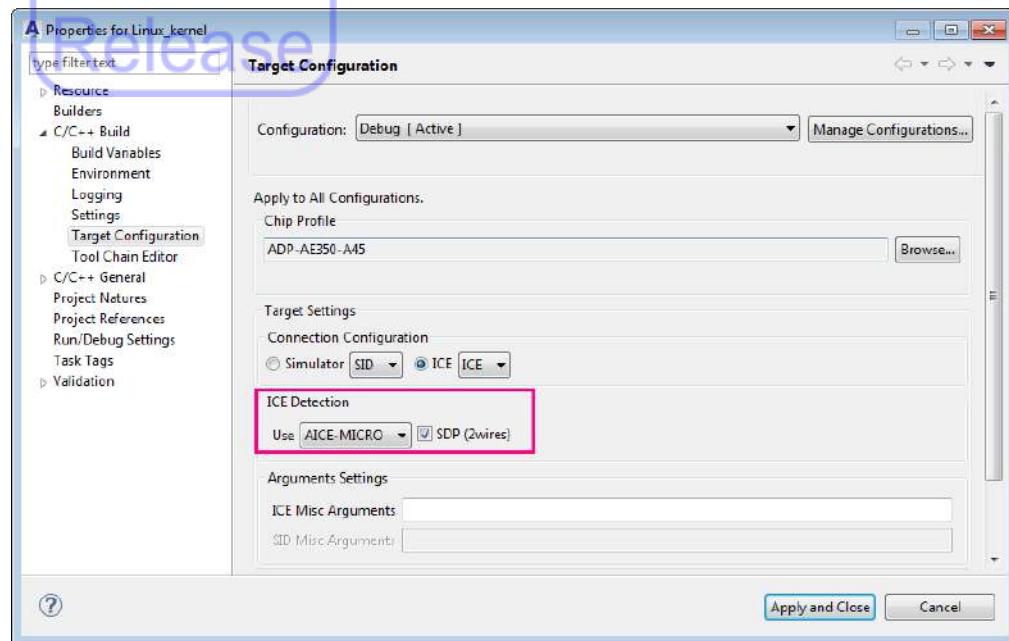


**Step 2** Reference *Linux Development User Manual* to obtain or build the following files and copy them to the newly-created project:

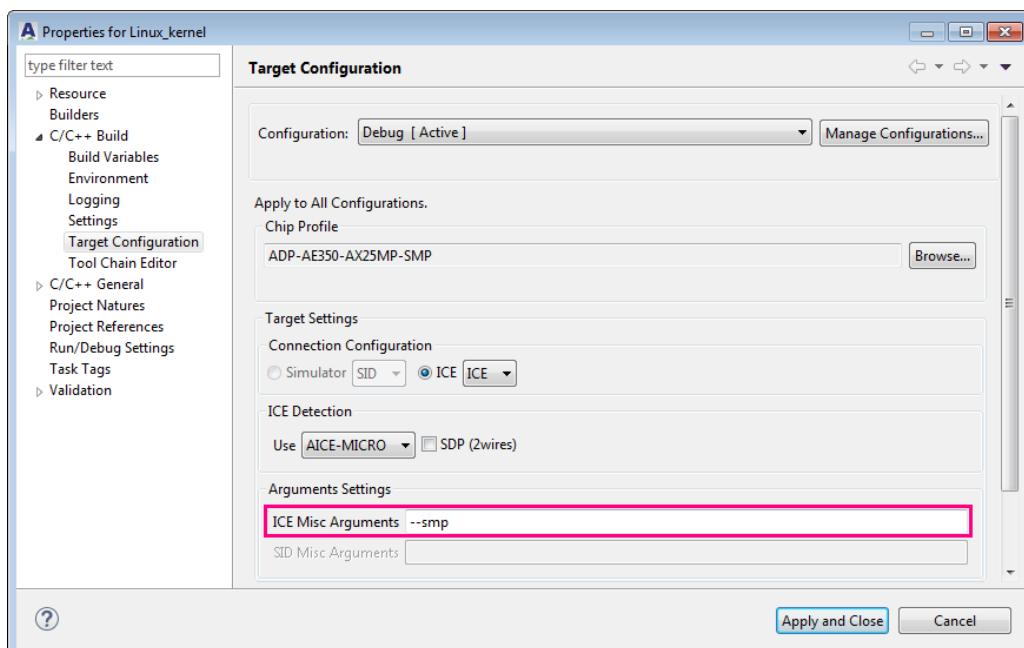
- Source package of the Linux kernel
- Source package of the M-mode firmware OpenSBI
- ELF and raw binary files of Linux kernel image, [vmlinux](#) and [Image](#)
- ELF and raw binary files of OpenSBI image, [fw\\_jump.elf](#) and [fw\\_jump.bin](#)
- DTB file corresponding to your target



**Step 3** If you are using a target board with 2-wire debug interface in conjunction with the ICE device AICE-MICRO or AICE-MINI+, right-click the project in **Project Explorer**, select “Target Configuration” from the pull-down menu to invoke the **Properties** dialog, and select the option “SDP (2 wires)” in the ICE detection section.

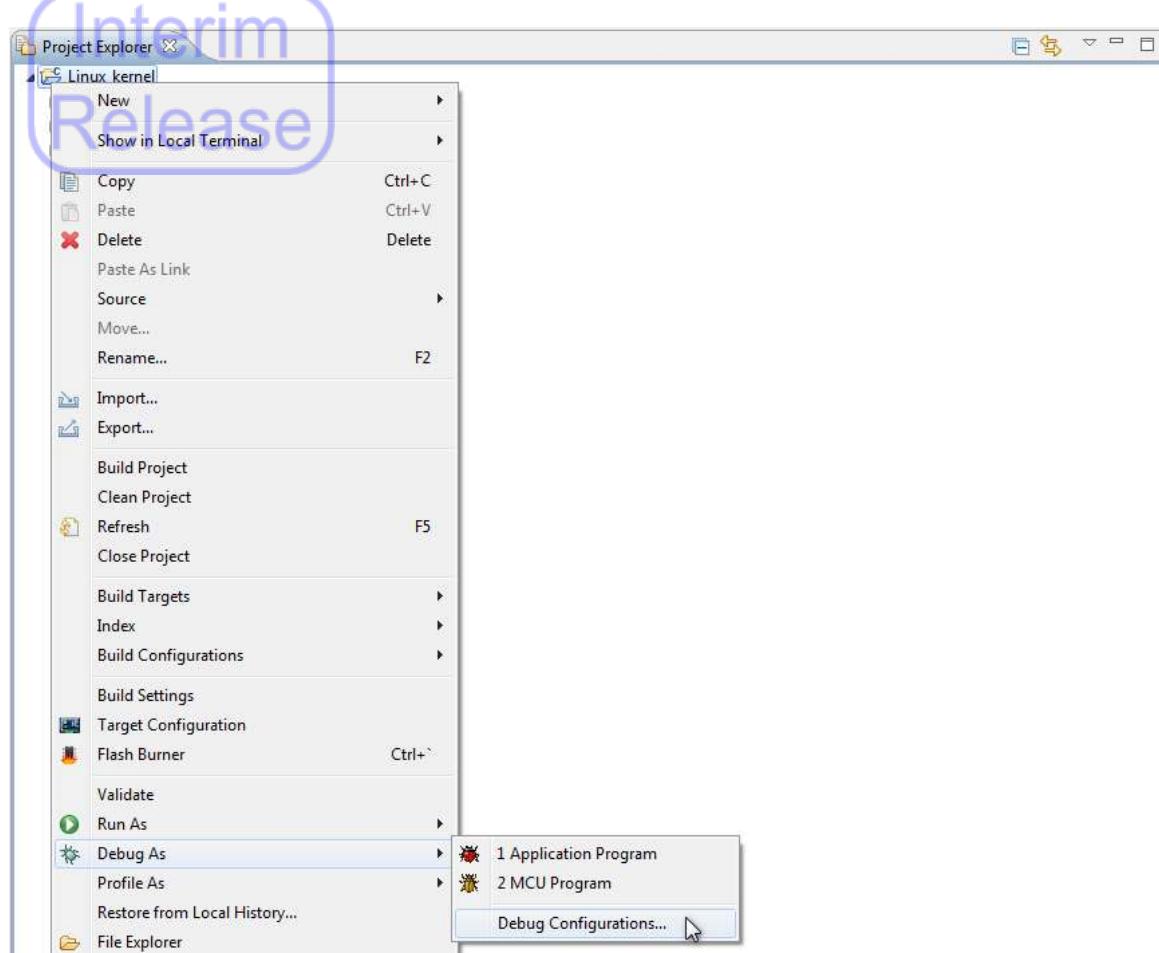


To debug the Linux kernel on a SMP system, specify “`--smp`” in the ICEman Misc Arguments field.

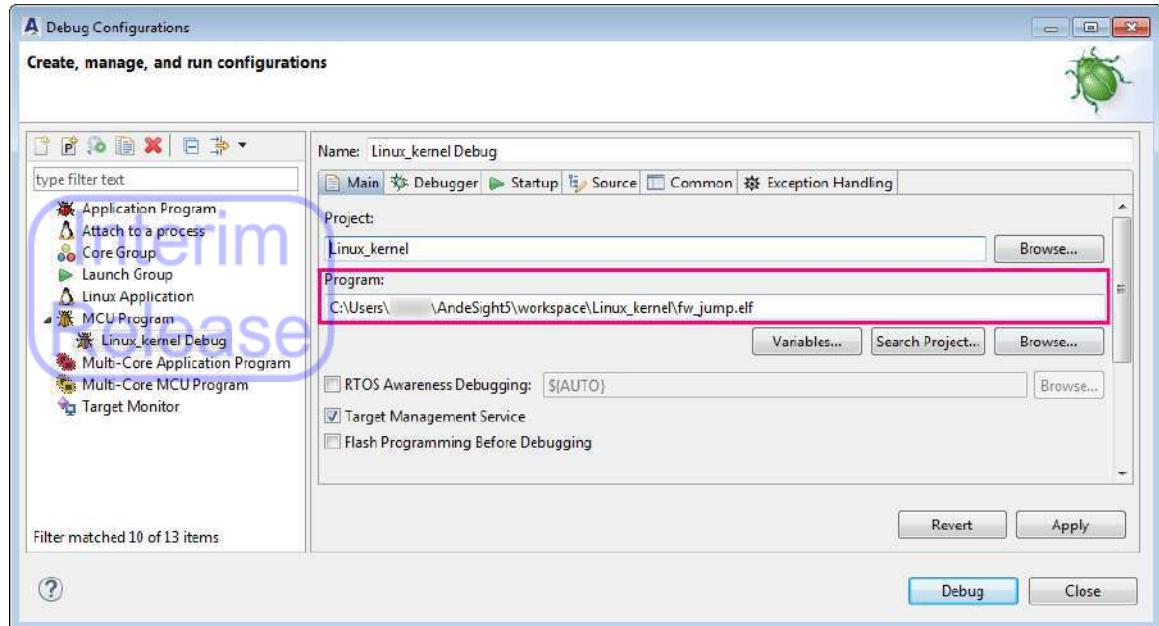


Otherwise, jump to the next step.

**Step 4** Right-click the project in the **Project Explorer** view and select “Debug as > Debug Configurations...” on the pull-down menu.



**Step 5** On the **Debug Configurations** dialog, double-click the configuration type “MCU Program” in the navigation pane to create a debug configuration. Go to the **Main** tab to specify the OpenSBI firmware file `fw_jump.elf` under the project for the **Program** field.



**Step 6** Go to the **Startup** tab. Select the option “Reset and Hold” first. Next, select the option “Load binary file” and specify to use the raw binary file of OpenSBI firmware (**fw\_jump.bin**) under the project. Then, enter GDB Run Commands for your target as follows:

```
restore DTB_FILE binary 0x20000000
restore Image binary [0x200000|0x400000]
thread apply all set $pc=0x0
thread apply all set $a1=0x20000000
thread apply all set $a0=$mhartid
add-symbol-file vmlinux
(set substitute-path KERNEL_SOURCE_PATH KERNEL_LOCAL_PATH)
(set substitute-path OPENSBI_SOURCE_PATH OPENSBI_LOCAL_PATH)
delete mem
```

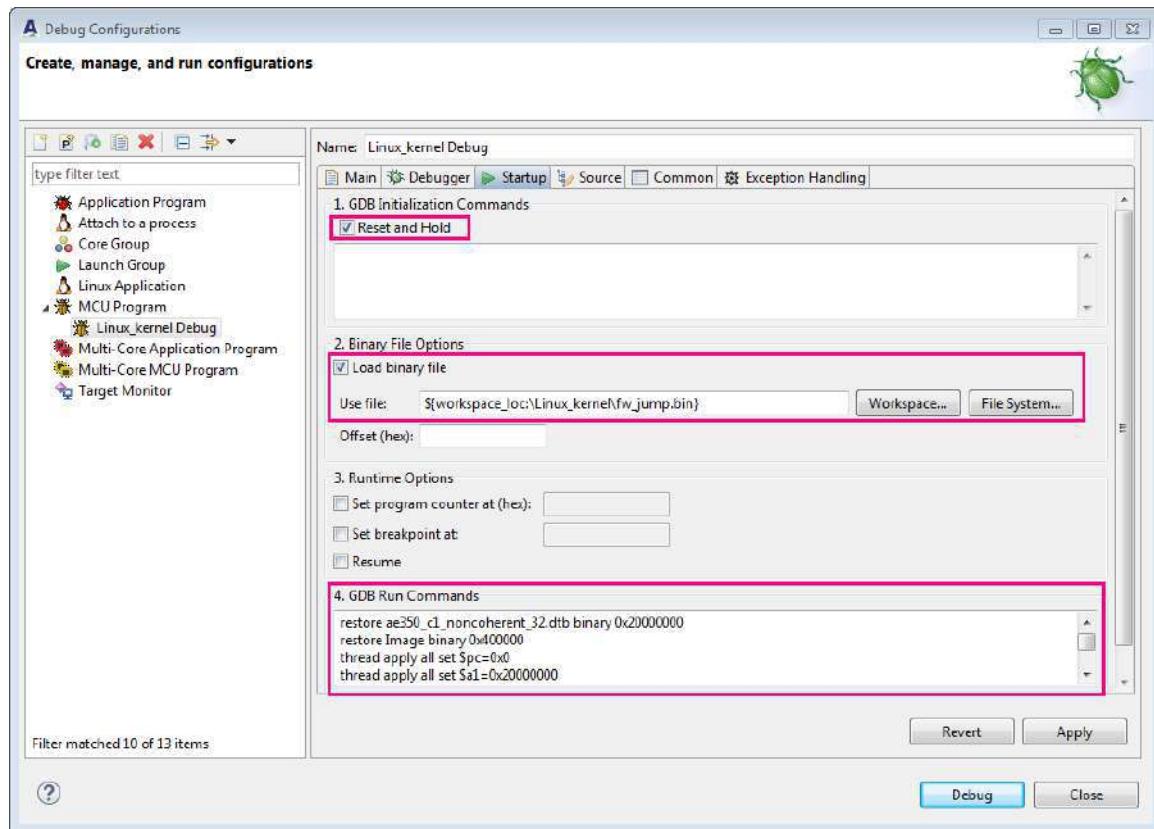
These commands are used to perform the following tasks:

1. Loading the device tree blob to the address 0x20000000.
2. Loading the Linux kernel to the address 0x200000 on a 64-bit system or to 0x400000 on a 32-bit system.
3. Enabling CPU registers to recognize the dtb location
4. Loading symbols from vmlinux
5. Enabling GDB to access the entire memory region

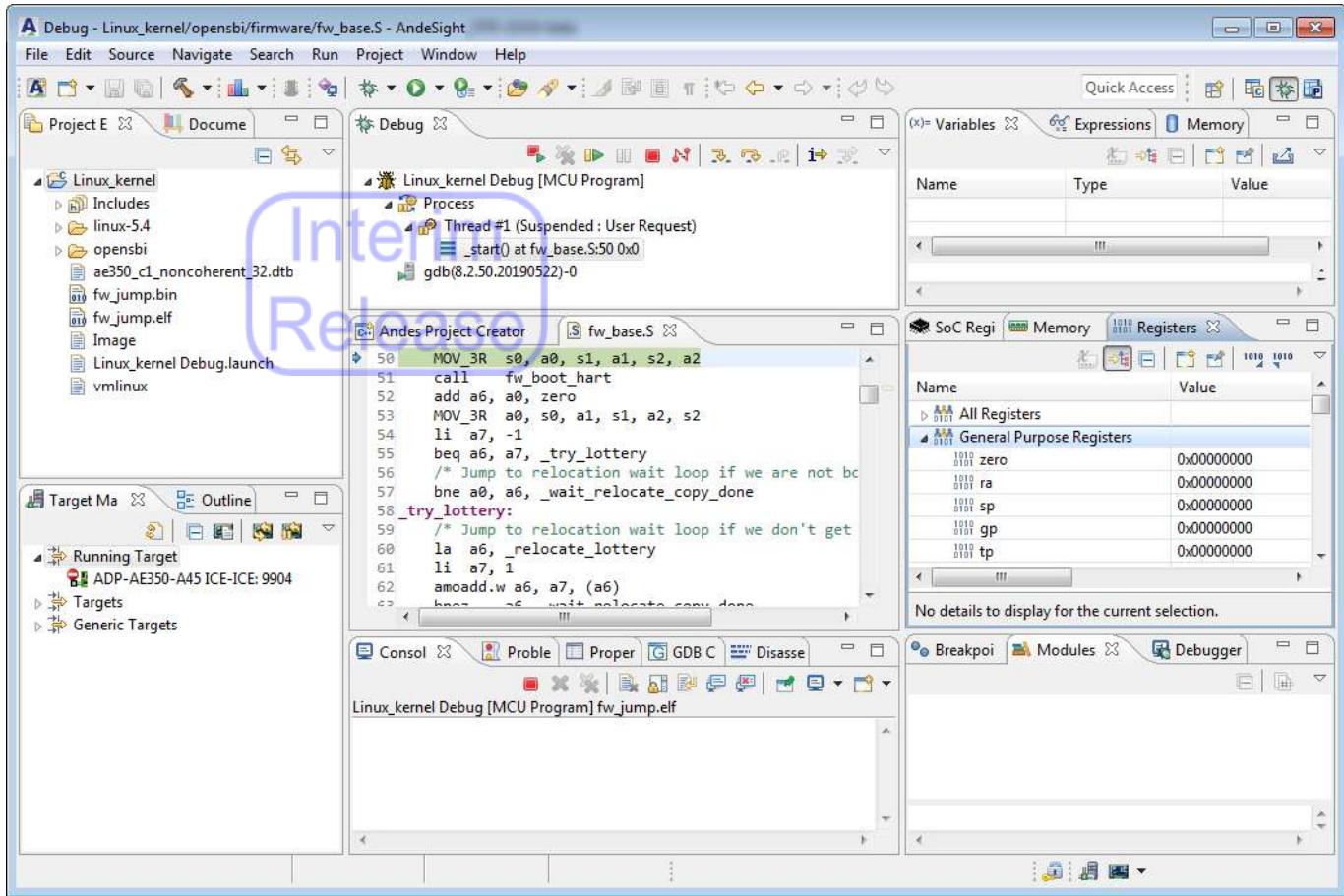
Note that the `substitute_path` command is only used when debugging the Linux kernel on Windows. The command is to map the original paths of kernel and OpenSBI sources with their local cygwin paths after symbols are loaded from `vmlinux`. The following is an example of specifying cygwin paths for kernel and OpenSBI sources.

`set substitute-path /home/AndeSight/linux/linux-5.4  
/cygdrive/c/users/USER/AndeSight5/workspace/PROJECT/linux-5.4`

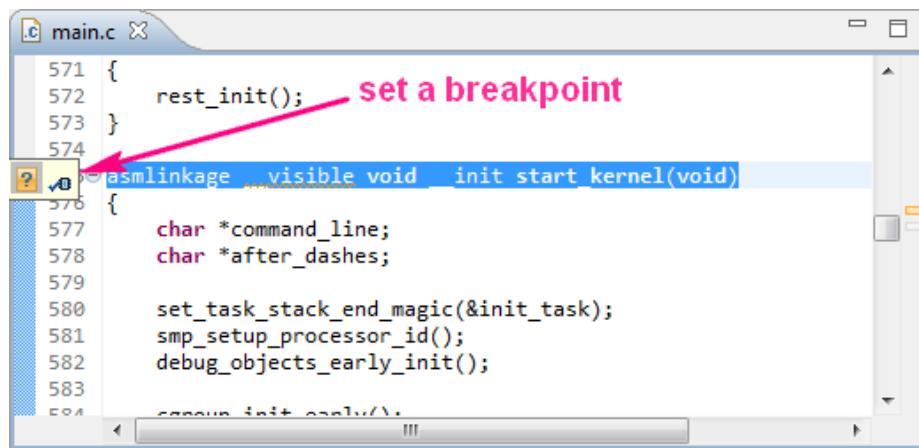
`set substitute-path /home/AndeSight/linux/opensbi  
/cygdrive/c/users/USER/AndeSight5/workspace/PROJECT/opensbi`



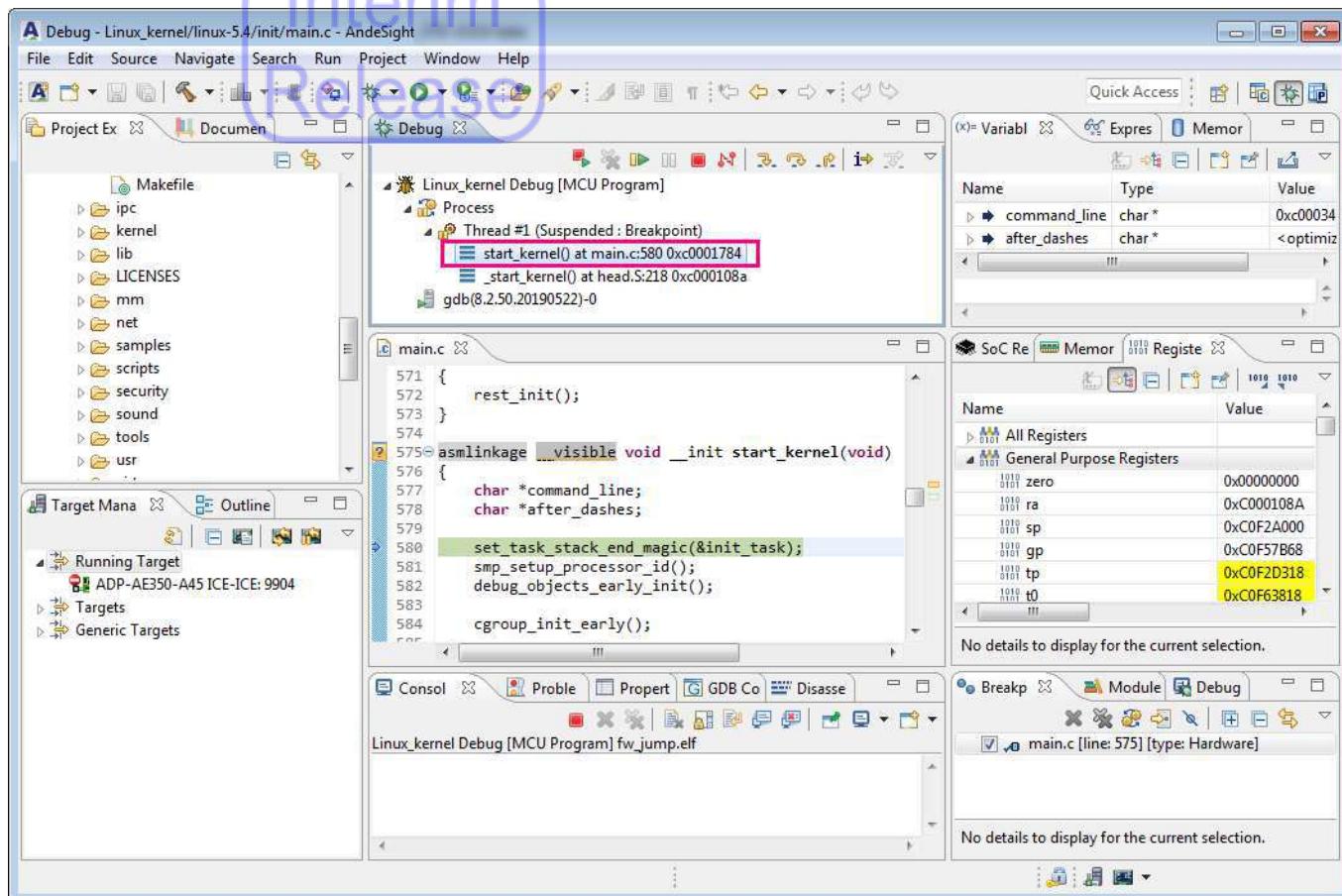
**Step 7** Click “Debug” on the **Debug Configurations** dialog to launch a debug session for the project. The **Debug** perspective is invoked automatically. Please allow time to load the OpenSBI binary, which will take several minutes. For a SMP system, the stack frame in the **Debug** view will display multiple threads for respective cores on the system.



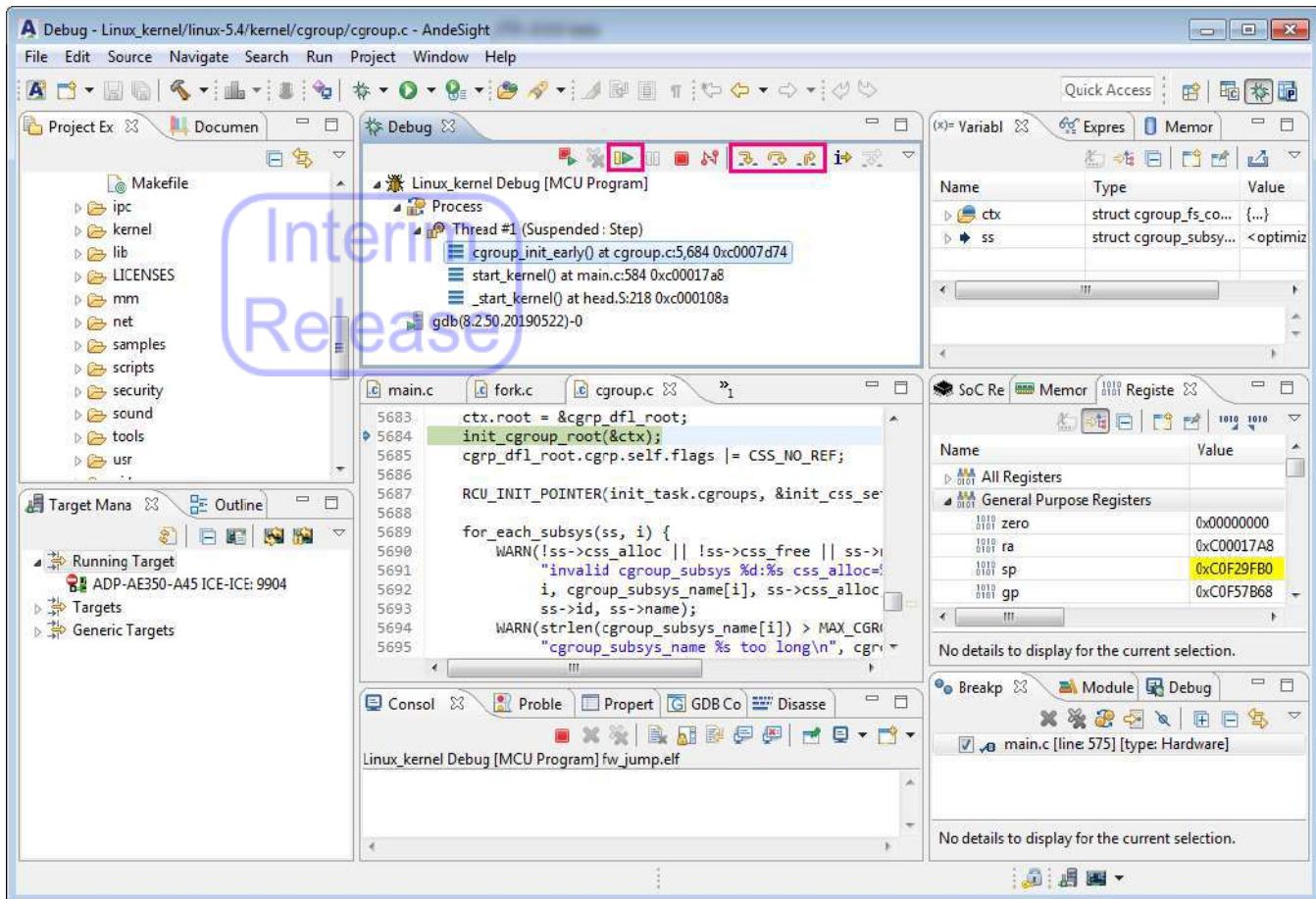
**Step 8** Double-click `main.c` under `PROJECT\linux-5.4\init\` to open it in the editor and follow Section 2.4.4.1 to set a hardware breakpoint at `start_kernel()`, the entry point of the kernel. Then, click (Resume) on the toolbar of the **Debug** view to resume program execution.



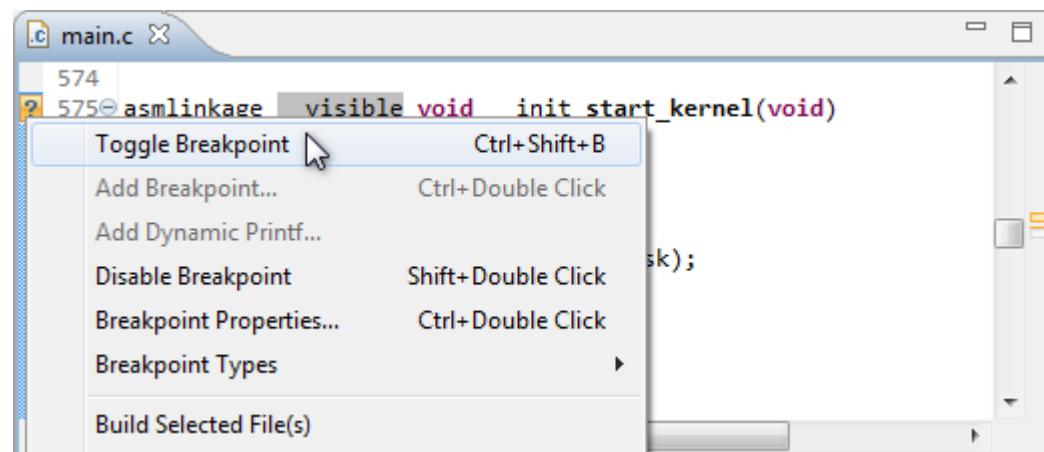
**Step 9** The program suspends at `start_kernel()`. For a SMP target, one of the threads is suspended, showing that one of the SMP cores has hit the breakpoint. Now you can set a breakpoint, hardware or software, at any other location of the project.



**Step 10** In the **Debug** view, click  (Resume),  (Step Into),  (Step Over) or  (Step Return) on the toolbar to execute statements of interest, observe the changes on the stack frames in the **Debug** view, and examine variables or other values in other debug views (**Variables** view, **Registers** view...).



Note that breakpoints you set will be invalid after program execution is terminated. For a re-launched debug session, please set the hardware breakpoint at `start_kernel()` and other breakpoints again by toggling off and on them in code editors after the program execution suspends due to the reset-and-hold operation.



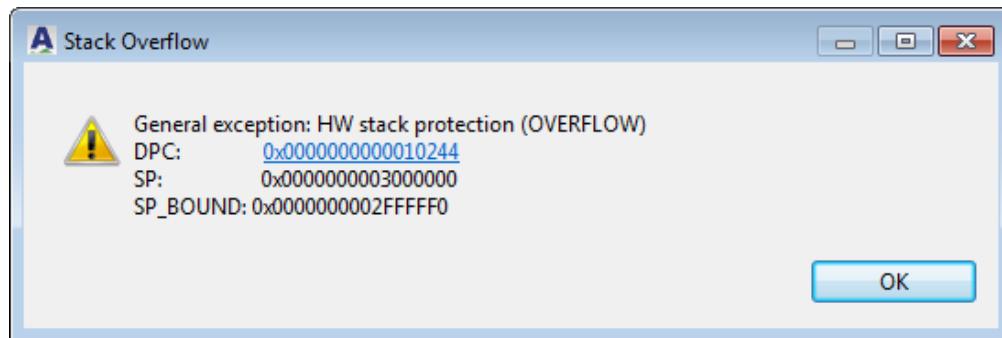
## 2.5. General exception handling and stack recording/protection

### 2.5.1. Stack recording and stack protection

For CPU cores that support hardware stack protection, AndeSight allows hardware top of stack recording or hardware stack protection; however, the two schemes cannot be used at the same time. The hardware top of stack recording scheme records the top of the stack boundary at runtime, helping you to find the required stack memory size for your applications. The hardware stack protection scheme reports stack overflow or underflow problems. It is subject to the following conditions:

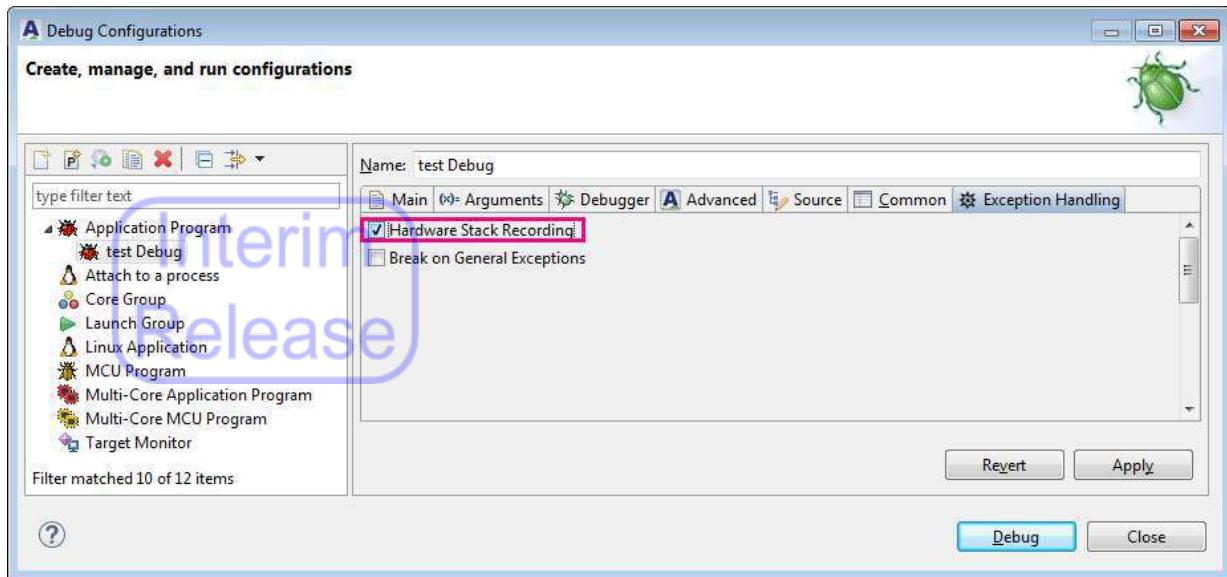
- The values of system registers for hardware stack protection cannot be modified using program code.
- From the limited number of available hardware breakpoints, one must be set aside for use by the stack protection handler.

To perform hardware stack protection, please refer to the previous section (Section 2.5.2). In the event of an exception, a dialog pops up providing relevant information.

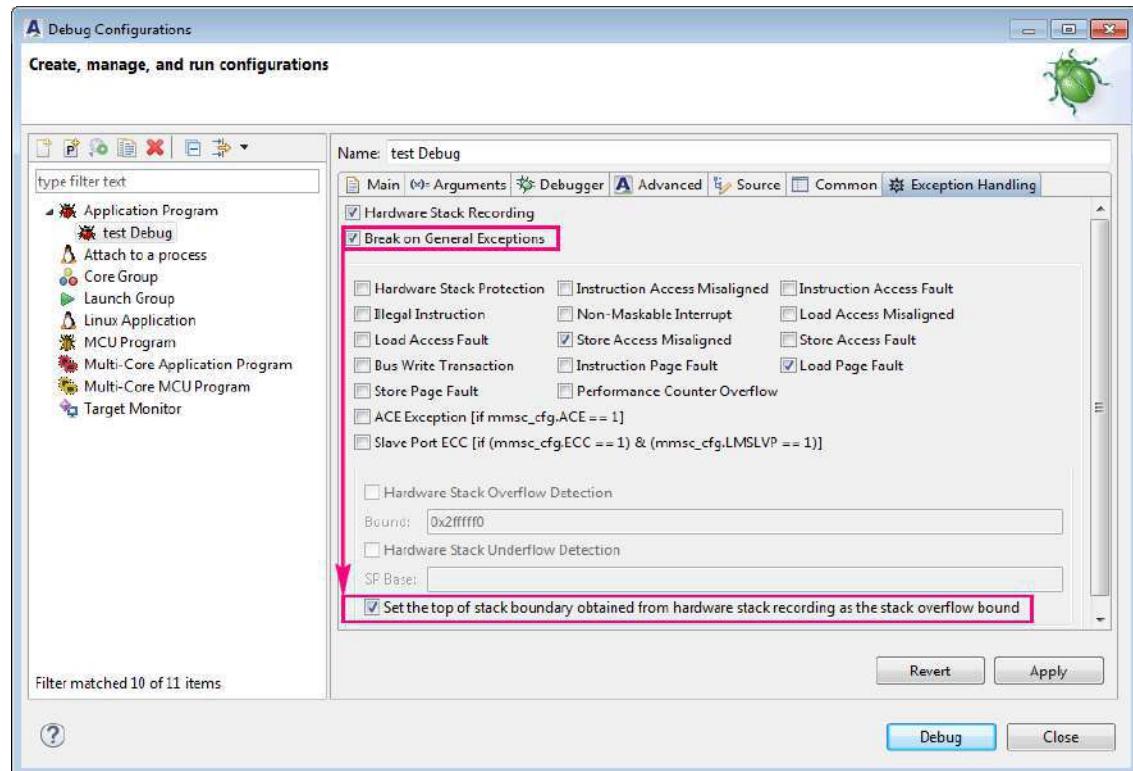


To enable hardware stack recording, proceed as follows:

**Step 1** Follow Section 2.4.3, Steps 1 to 2 to create a debug configuration for your project and click the **Exception Handling** tab on the **Debug Configurations** dialog. Then, select the option “Hardware Stack Recording”.



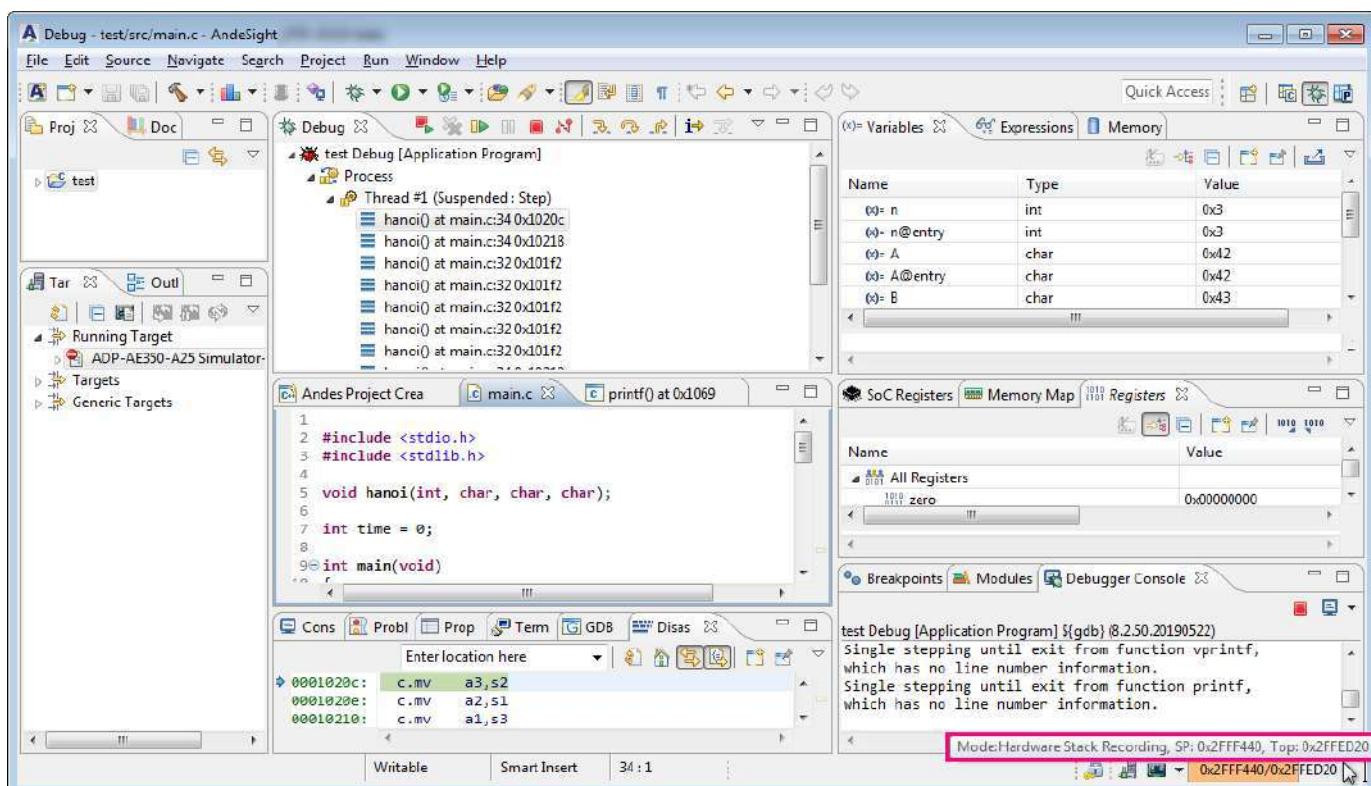
If you want to set the maximum top of stack boundary recorded from the upcoming debug session as the bound for later stack overflow detection, move on to select “Break on General Exceptions > Set the top of stack boundary obtained from hardware stack recording as the stack overflow bound” on the Exception Handling tab to enable the setting.



**Step 2** On the **Debug Configurations** dialog, click “Apply” and “Debug” to launch a debug session.

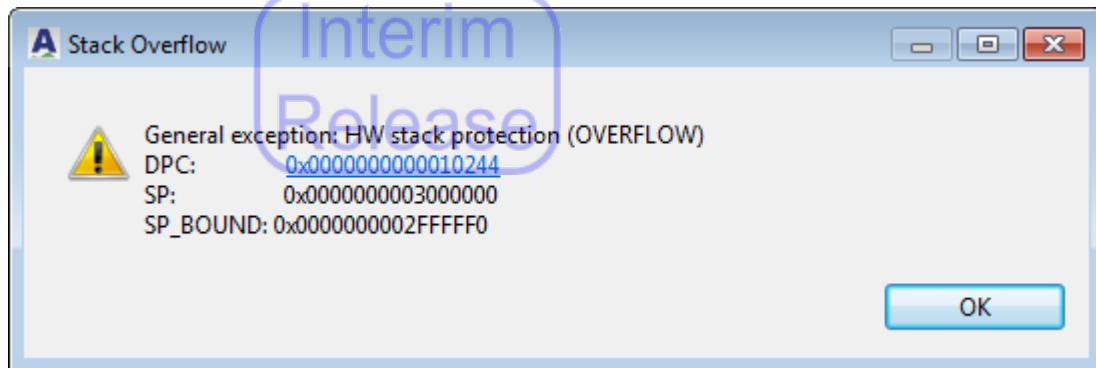
**Step 3** The **Debug perspective** is invoked automatically. Click  (Resume) on the toolbar of the **Debug view** to resume program execution.

The status bar in the lower right corner of the AndeSight IDE shows the top of the stack boundary at runtime with the “Top” value. In the example below, the top of stack boundary at runtime is **0x2FFED20**.



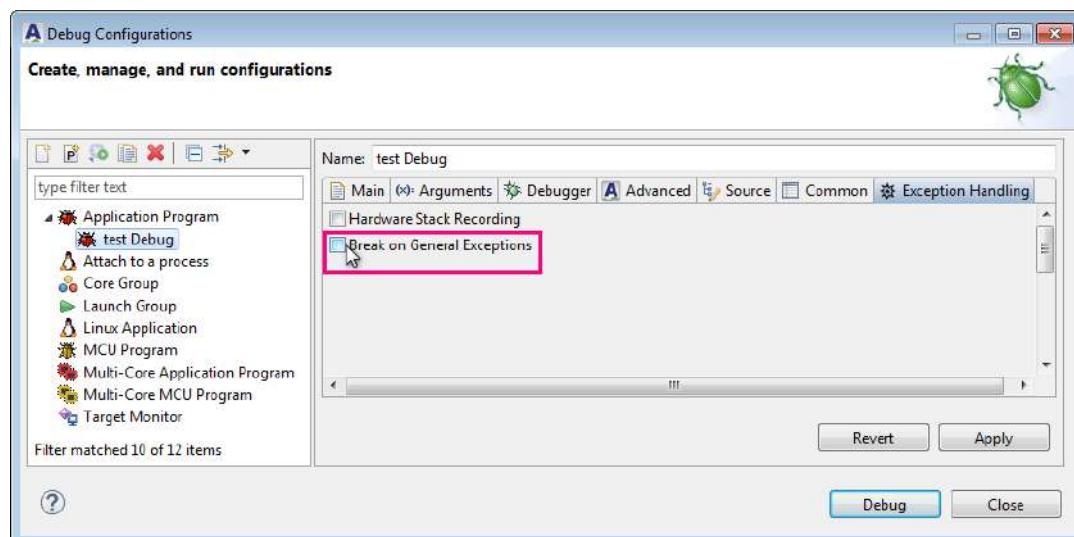
### 2.5.2. General exception handling

AndeSight allows you to stop program execution when a selected type of general exception occurs. A pop-up dialog with information about the exception will then appear as shown below:

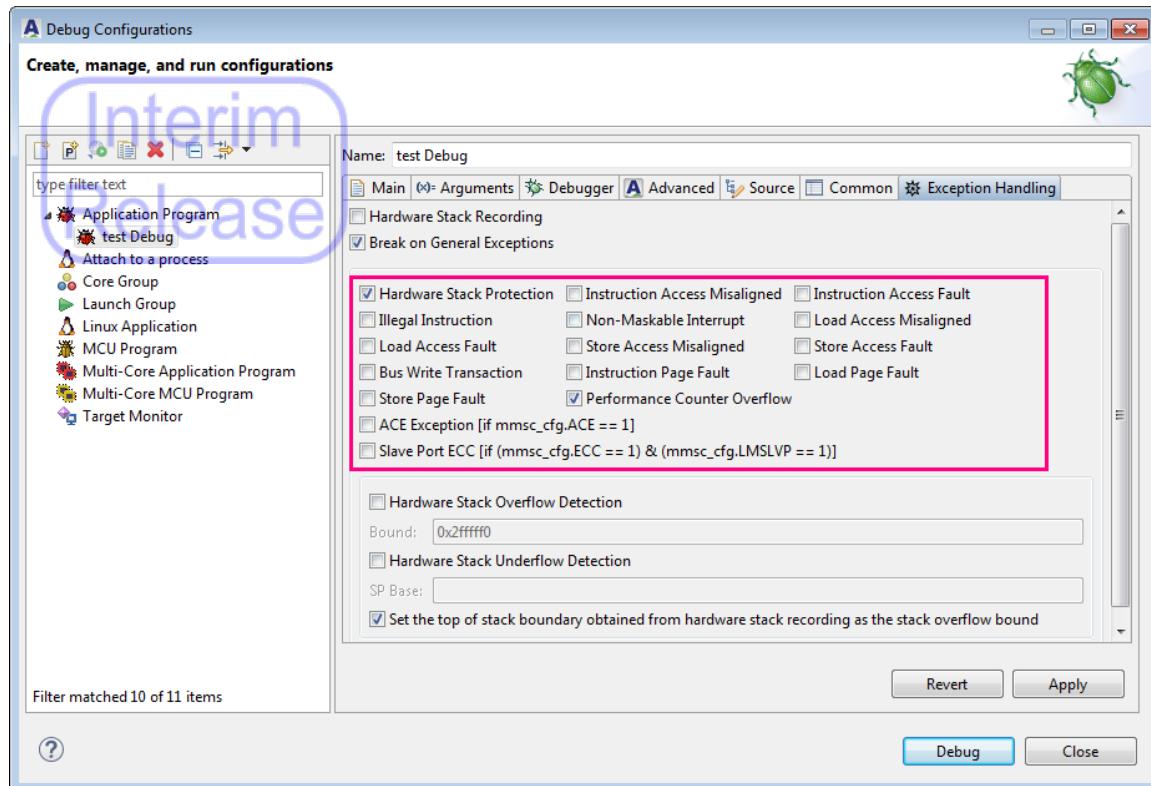


The following steps outline how to use AndeSight to catch certain general exception types in a debug session. Such a “handler” of specified exception types during program execution consumes a hardware breakpoint if the interrupt vector is in ROM or flash. In this case, make sure there are sufficient hardware breakpoints available first.

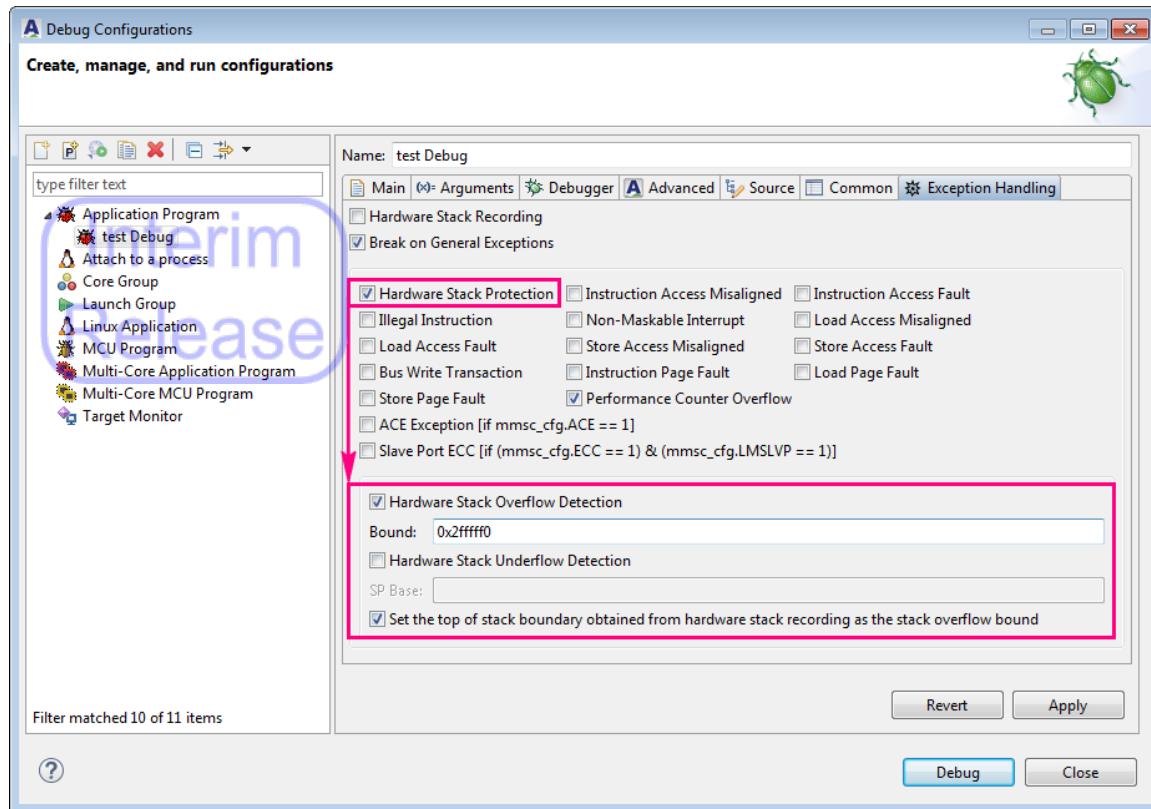
**Step 1** Follow Section 2.4.3, Steps 1 to 2 to create a debug configuration for your project and click the **Exception Handling** tab on the **Debug Configurations** dialog. Then, select the option “Break on General Exceptions.”



**Step 2** Specify one or multiple exception types that need to be caught during program execution.



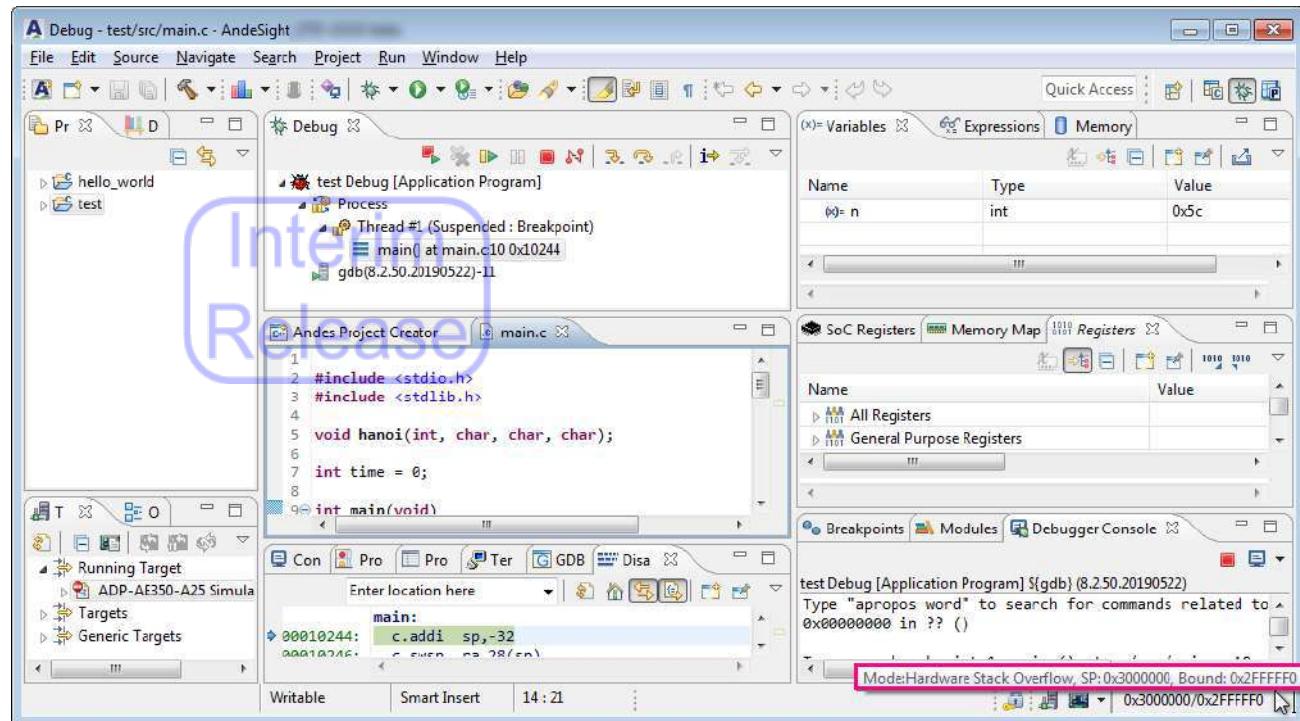
For the exception type “Hardware stack protection”, you also have to specify whether to perform “Hardware Stack Overflow Detection” and/or “Hardware Stack Underflow Detection”. For hardware stack overflow detection, enter the value of the stack bound register to be compared with the updated value of the Stack Pointer (SP) register. For hardware stack underflow detection, enter the value of the stack base register to be compared with the updated SP value. If you have performed hardware stack recording and specified the option “Set the top of stack boundary obtained from hardware stack recording as the stack overflow bound” for that, the bound field for hardware stack overflow detection will automatically be filled with the maximum top value of stack boundary recorded earlier.



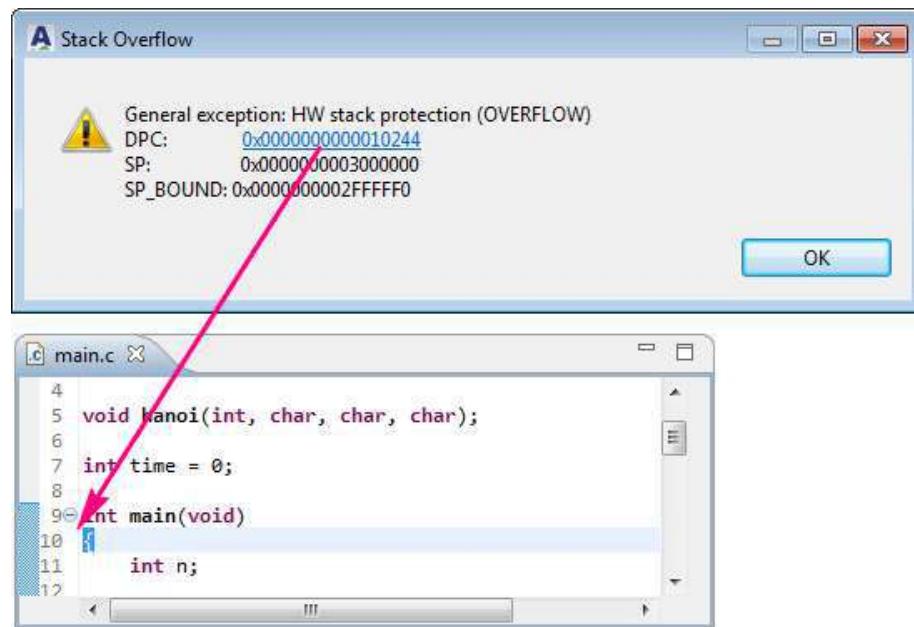
**Step 3** On the **Debug Configurations** dialog, click “Apply” and “Debug” to launch a debug session.

**Step 4** The **Debug** perspective is invoked automatically. Optionally, you may set breakpoints in the source code and click  (Resume) on the toolbar of the **Debug** view to resume program execution.

If the type of “Hardware stack protection” is specified, then the stack status during runtime will be displayed in the progress bar in the lower right corner of the AndeSight IDE.



**Step 5** Whenever a specified exception occurs, a dialog pops up providing relevant information. To locate where the exception occurs in source code, just click the IPC value on the dialog.



## 2.6. Profiling

AndeSight allows the real-time profiling of projects built with GCC compiler. It renders and summarizes informative profiling statistics immediately after your program is suspended due to a breakpoint, or stopped after single stepping.

---

**NOTE**

1. Andes LLVM compiler has not supported the profile feature yet.
  2. The Virtual Hosting configuration may affect program profiling. For programs enabled to have the Virtual Hosting support, the profiling results can be inaccurate.
-

## 2.6.1. Profile perspective

The **Profile** perspective contains profile views that display profiling data.

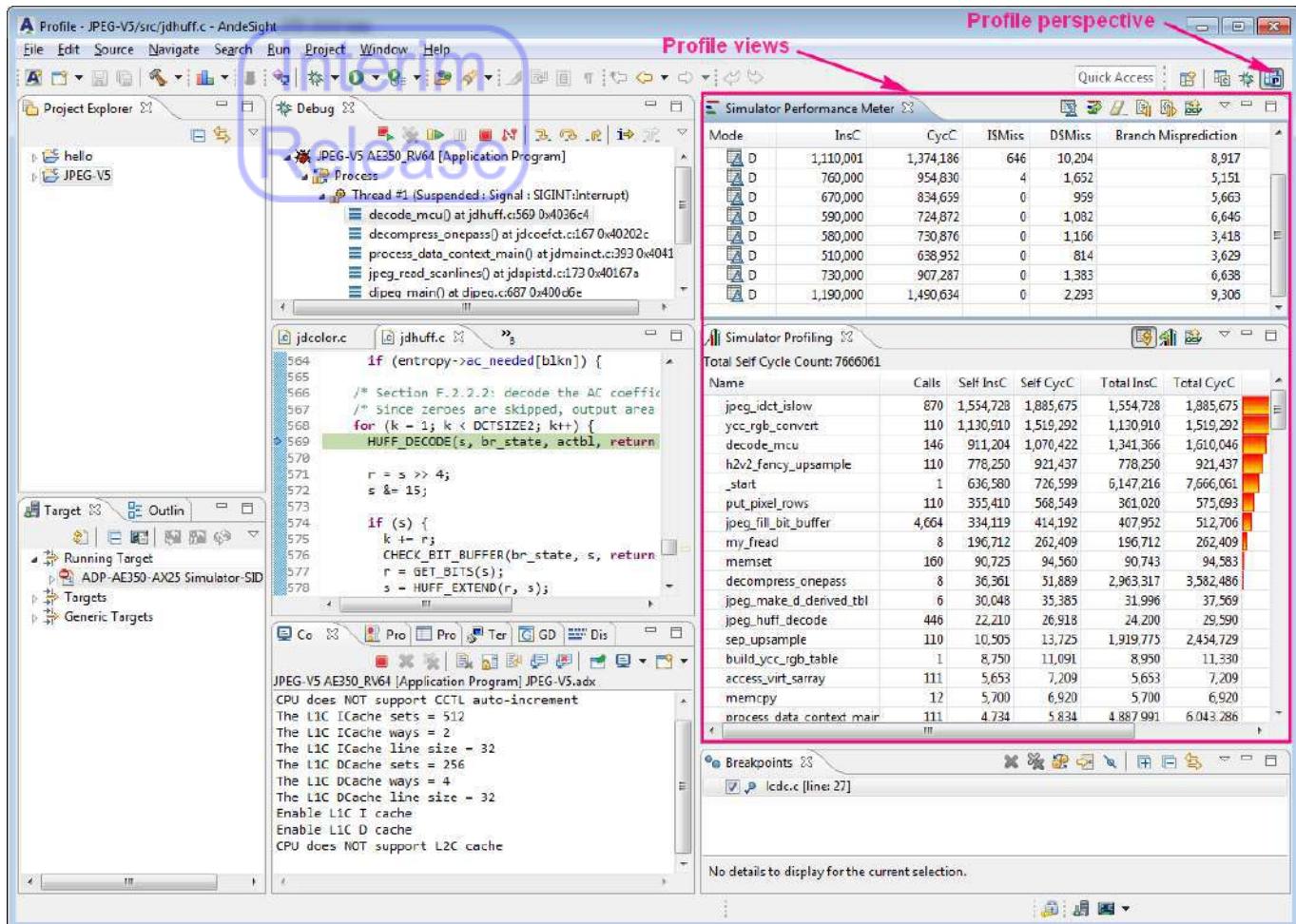
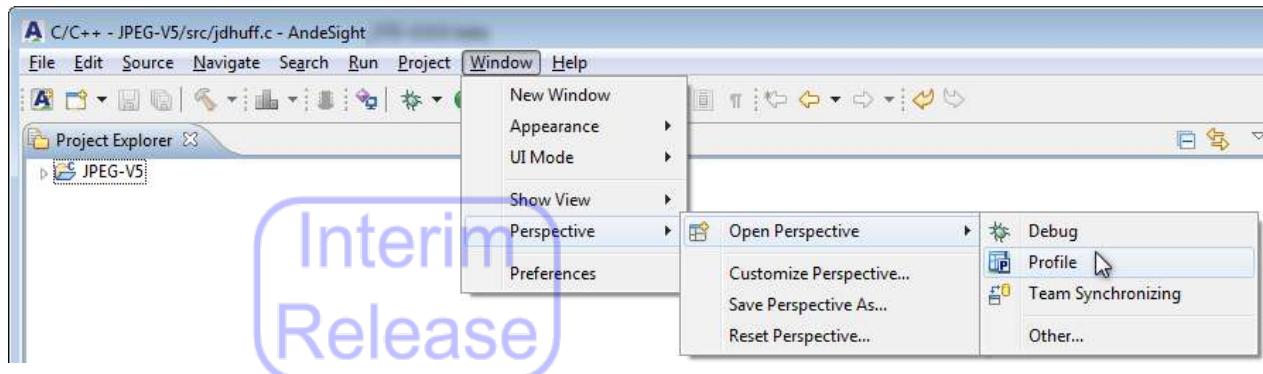
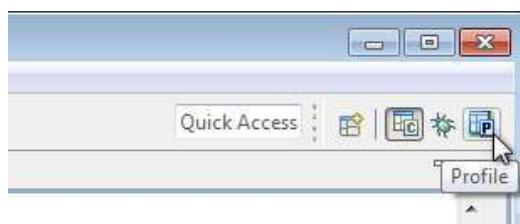


Figure 31. Profile perspective of AndeSight IDE

The **Profile** perspective is evoked automatically when a profile session is launched. You can also evoke the profile perspective manually by selecting “Window > Perspective > Open Perspective > Profile” from the AndeSight main menu.

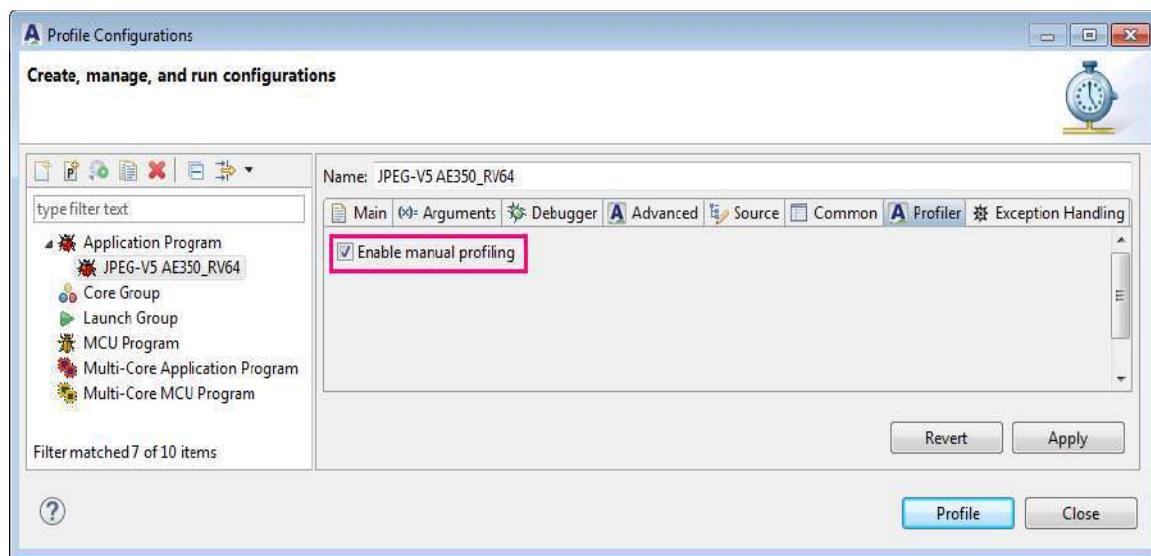


Likewise, you may switch to the **Profile** perspective from the AndeSight perspectives toolbar.



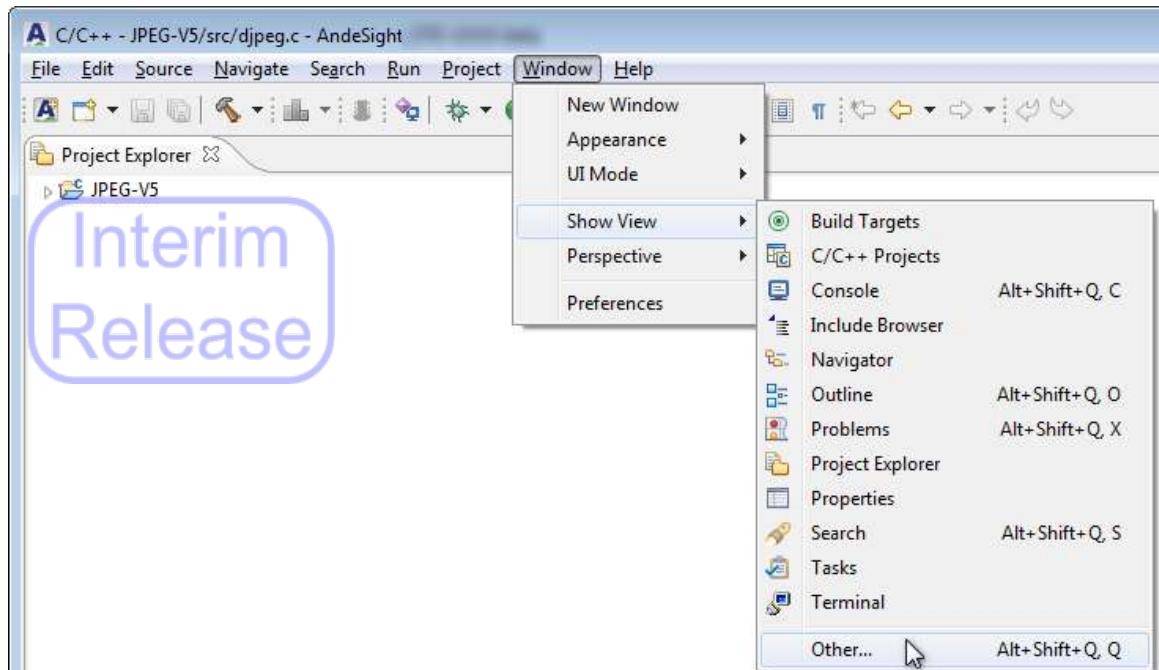
## 2.6.2. Profile views

Profile views provide a real-time interface to track and investigate bottlenecks during program execution. For projects using default profile configuration, profiling data are generated and displayed in profile views automatically after the execution suspends or terminates. For projects configured to have manual profiling (as shown below), you need to start a profile session by clicking the button  from the AndeSight toolbar during runtime so that the profiling data will be generated and displayed. For details about how to control the start and end of a profile session during program execution, please refer to Section 2.6.3.

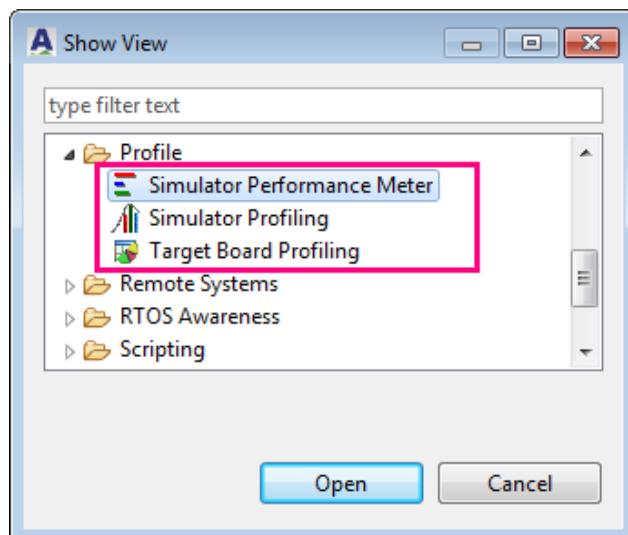


For programs with a simulator target, the profile views triggered during a profile session are the **Simulator Performance Meter** view and the **Simulator Profiling** view. For programs with an ICE target, the profile view is the **Target Board Profiling** view. These views open automatically as part of the **Profile** perspective after launching a profile session. You can also evoke them manually as follows:

**Step 1** From the AndeSight main menu, select “Window > Show View > Other...”.



**Step 2** In the **Show View** dialog, click “Profile” to select a desired profile view from the expanded tree and click “Open.”



### 2.6.2.1 Simulator Performance Meter view

For programs with a simulator target only, the **Simulator Performance Meter** view shows the profiling data between any two program stop points in columns listed in Table 6.

Simulator Performance Meter						
Mode	InsC	CycC	I\$Miss	D\$Miss	Branch Misprediction	
Δ D	1,550	1,606	0	0	17	
Δ D	5	7	0	0	1	
Δ D	2	2	0	0	0	
Δ D	155	297	0	0	33	
Δ D	1	1	0	0	0	
Δ D	176	258	0	0	7	
Δ D	3	9	0	0	2	
Δ D	5	8	0	0	1	

Table 6. Columns in Simulator Performance Meter view

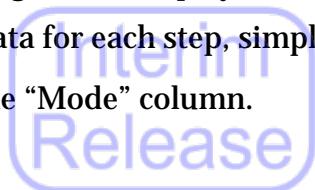
Available columns		Description
Mode	Delta	profiling data for each step in program execution is displayed
	Cumulative	profiling data for each step is accumulated in current step
InsC		aggregated instruction count up to a particular function with respect to time
CycC		aggregated cycle count up to a particular function with respect to time
I\$ Miss		the number of times an instruction cache miss occurs
D\$ Miss		the number of times a data cache miss occurs
Branch Misprediction		the number of times that the branch predictor fails to predict the correct fetched address
File Name		file name for source code under investigation
Line Number		source code line number
Source Code		source code under investigation
Address		memory address of corresponding source code
Instruction		assembly code derived from source code

### Toolbar for the Simulator Performance Meter view

#### Cumulative mode



The profiling data is displayed in Delta mode ( D) by default. To accumulate the profiling data for each step, simply click this button. The cumulative data is denoted by C in the “Mode” column.



#### Reset performance meter



Click this button to set the cumulative data to zero.

#### Clear data



Click this button to erase all profiling data in the **Simulator Performance Meter** view.

#### Go to the previous source



Click this button to go back to the row of profiling data generated in the last step.

#### Go to the next source



Click this button to go forward to the row of profiling data generated in the next step.

#### Export to csv



Click this button to export the profiling data to a .csv file.

#### View menu



Click this button to choose a desired layout. You may also filter the columns in the **Simulator Performance Meter** view by clicking “ (View Menu) > Layout > Select Columns...” and making selections on the evoked dialog. For a complete list of columns in the **Simulator Performance Meter** view, please refer to Table 6.

### 2.6.2.2 Simulator Profiling view

For programs with a simulator target only, the **Simulator Profiling** view lists all executed functions for the program being profiled. The statistics are updated in real time with respect to each step executed when the “Auto Refresh Mode” is enabled. You may sort the functions in a particular order by clicking a desired column header. For instance, by clicking the column header “Time Percentage”, the functions in this view are sorted according to their runtime percentage. For a detailed list of columns in this view, please refer to Table 7.

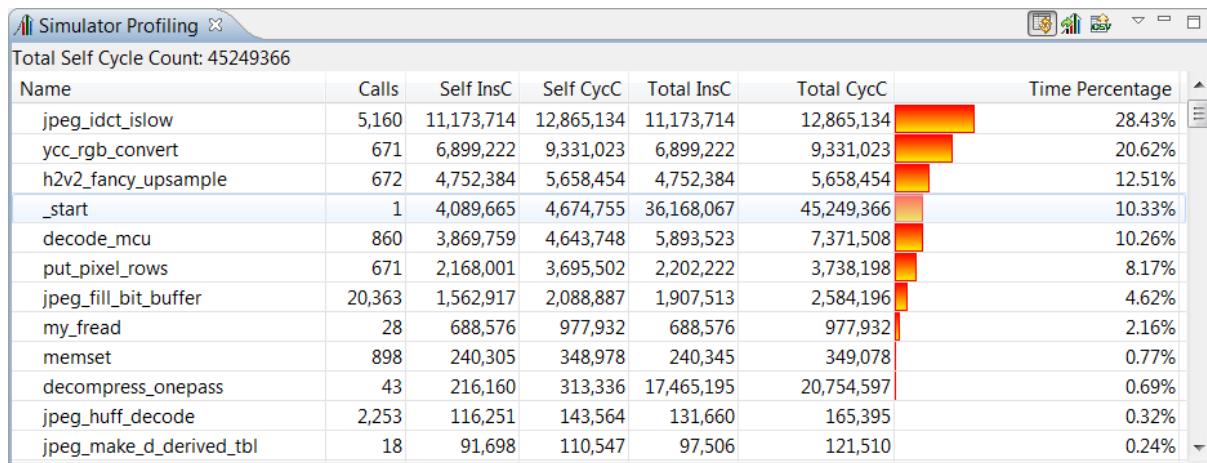


Table 7. Columns in the Simulator Profiling view

Available columns	Description
Name	name of function
FuncCall PC	memory address of function
Calls	total number of times a function is called up – a function neither called nor compiled is not displayed
Self InsC	Self Instruction Count – total number of instructions per function
Self CycC	Self Cycle Count – total number of cycles per function
Total InsC	Total Instruction Count – total number of instructions per function plus total number of instructions per function's children function per call
Total CycC	Total Cycle Count – total number of cycles per function per call plus total number of cycles per function's children function per call

<b>Available columns</b>	<b>Description</b>
Time Percentage	runtime percentage
Avg Self InsC	Average Self-instruction Count – average number of instructions per function
Avg Self CycC	Self Cycle Count – average number of cycles per function
Avg Total CycC	Average Total Cycle Count – average total number of cycles per function per call plus total number of cycles per child function per call
Avg Total InsC	Average Total Instruction Count – average total number of instructions per function per call plus total number of instructions per child function per call
CPI	Self Cycles Per Self Instruction
Accumulated Percentage	accumulation of runtime percentage
Max Total InsC	Maximum Total Instruction Count - maximum total number of instructions per function per call plus total number of instructions per child function per call
Max Total CycC	Maximum Total Cycle Count - maximum total number of cycles per function per call plus total number of cycles per child function per call
Min Total InsC	Minimum Total Instruction Count - minimum total number of instructions per function per call plus total number of instructions per child function per call
Min Total CycC	Minimum Total Cycle Count - minimum total number of cycles per function per call plus total number of cycles per child function per call
Max Self InsC	Maximum Self Instruction Count - maximum number of instructions per function
Max Self CycC	Maximum Self Cycle Count - maximum number of cycles per function
Min Self InsC	Minimum Self Instruction Count - minimum number of instructions per function
Min Self CycC	Minimum Self Cycle Count - minimum number of cycles per function

**Toolbar for the Simulator Profiling view****Auto refresh mode**

Click this button to display profiling data in real time.

**Reset profiling statistics**

Click this button to set the profiling statistics to zero.

**Export to csv**

Click this button to export the profiling data to a .csv file.

**View menu**

Click this button to choose a desired layout. You may also filter the columns in the

**Simulator Profiling** view by clicking “ (View Menu) > Layout > Select Columns...” and selecting from the evoked dialog. For a complete list of columns in the **Simulator Profiling** view, please refer to Table 7.

### 2.6.2.3 Target Board Profiling view

For programs with an ICE target only, the **Target Board Profiling** view shows which parts of the program consume most of the execution time. It also provides call graph information for each function and allows you to create charts based on profiling data.

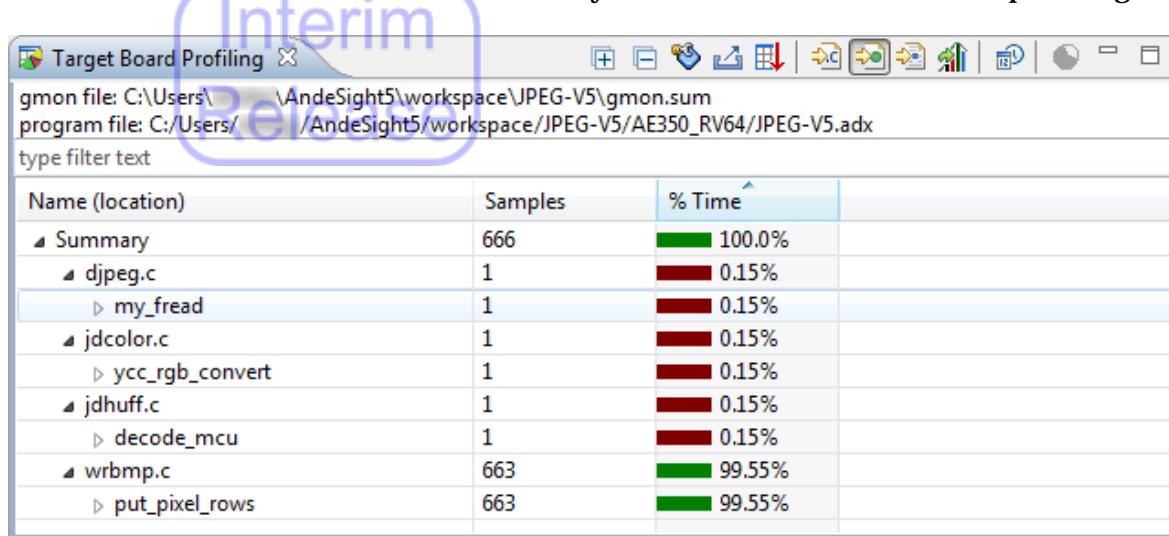


Table 8. Columns in Target Board Profiling view

Available columns	Description
Name (location)	List of profiled source files and their instructions
Samples	Sampling count of an instruction. Instruction calls are sampled every 10 milliseconds
%Time	Percentage of execution time of a particular instruction with respect to the total execution time of the program

#### Toolbar for the Target Board Profiling view

##### Show/hide columns



Click this button to invoke a dialog to specify the columns that you want to display in the **Target Board Profiling** view.

##### Export to csv



Click this button to export profiling data to a .csv file.

**Sorting**

Click this button to invoke a dialog to specify the sorting order of your profiling data in the **Target Board Profiling** view.

**Sort samples per file**

Click this button to sort the profiling data by file.

**Sort samples per function**

Click this button to sort the profiling data by function.

**Sort samples per line**

Click this button to sort the profiling data by line.

**Reset profiling data**

Click this button to set the profiling data to zero.

**Switch samples/time**

Click this button to switch the data from sample information to time information or vice versa.

**Create chart...**

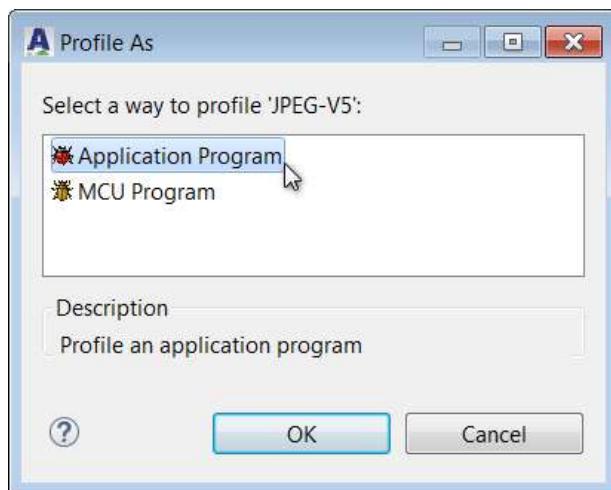
Click this button to create a bar graph, vertical bars, or a pie chart based on the profiling data specified in the **Target Board Profiling** view.

### 2.6.3. Engaging a profile session

This section uses the JPEG demo to demonstrate how to launch a profile session for a project:

**Step 1** See Section 2.1 for instructions on creating and building a project.

**Step 2** In **Project Explorer**, select the project and click  (Profile) on the AndeSight toolbar. Select a configuration from the invoked **Profile As** dialog and click “OK”. This starts the profile session automatically from the beginning of program execution.

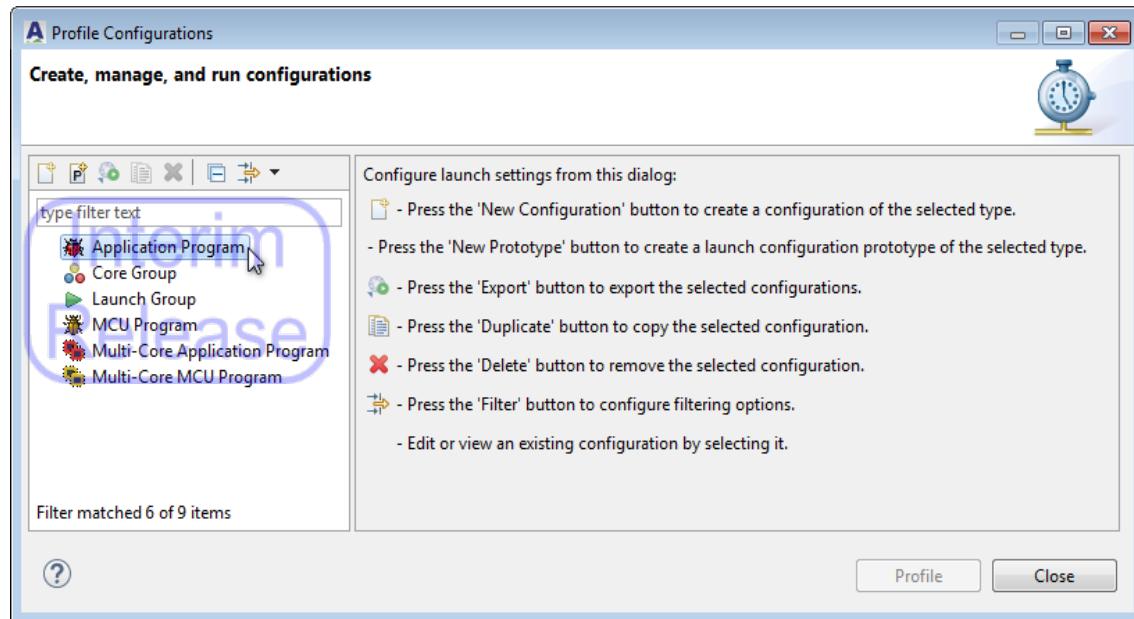


If you prefer to control the start of the profile session at the runtime, specify it in the profile configuration as follows:

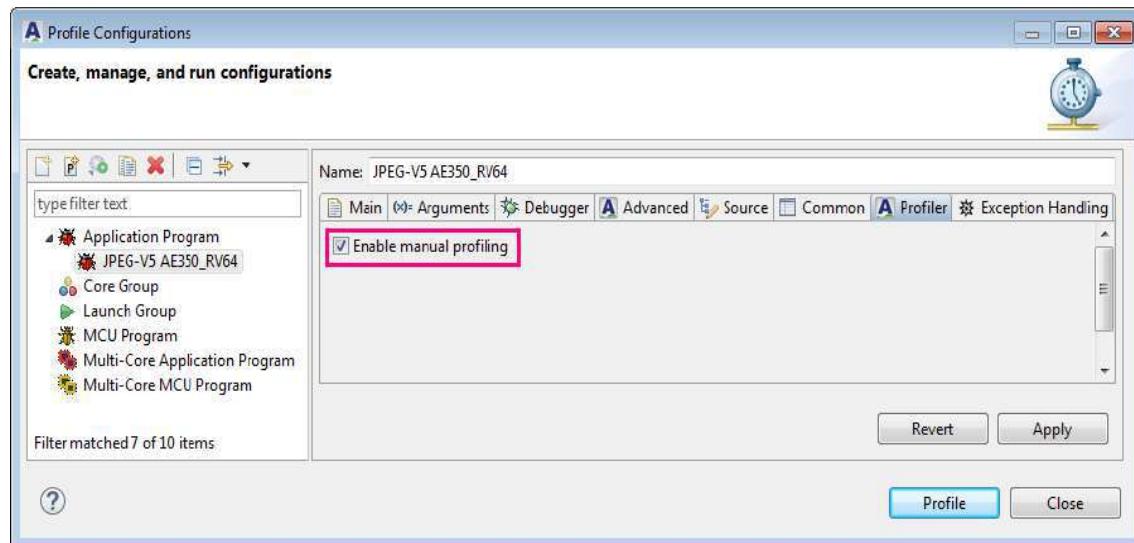
1. In **Project Explorer**, right-click the project and select “Profile As > Profile Configurations...” from the pull-down menu.



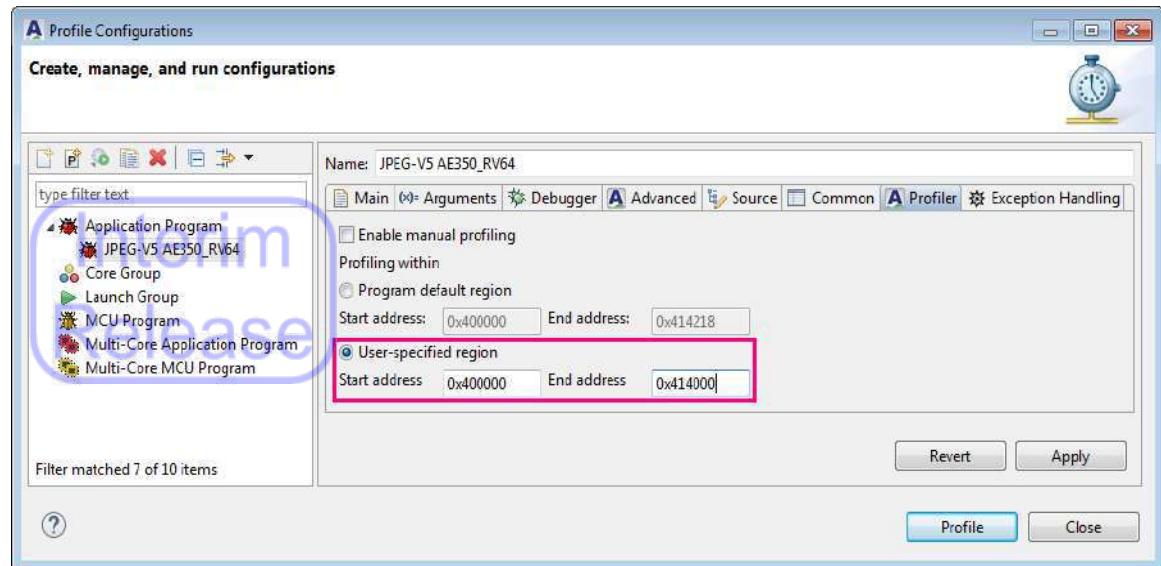
2. In the **Profile Configurations** dialog, double-click on a configuration type to create a profile configuration.



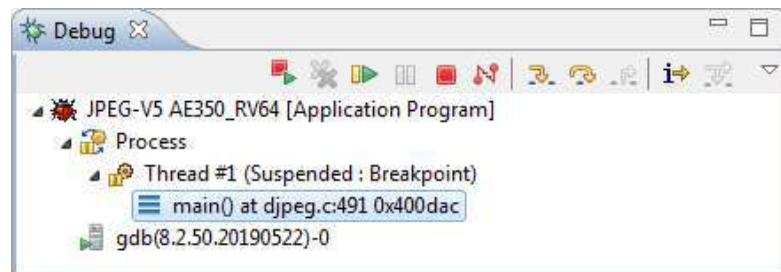
- Under the **Profile** tab of the newly-created configuration, select the option “Enable manual profiling” and click “Profile” on the dialog.



For ICE targets, the default section to be profiled is the .text section. If you want to change the profiled region, select the radio button “User-specified region” under the **Profile** tab and specify the start and end address of the profile region below before clicking “Profile” to launch the profile session.



**Step 3** The Profile perspective is invoked automatically. In the **Debug** view, notice that program execution is suspended at the main function. Click toolbar buttons like (Resume), (Step Into) or (Step Over) to proceed with the profiling.



For projects using default profile configuration, profiling data will be generated and displayed right after the execution suspends or terminates. For projects with a simulator target, profiling data is displayed in the **Simulator Profiling** view. Auto Refresh Mode is enabled by default to allow the display of profiling data in real time.

**Simulator Profiling**

Total Self Cycle Count: 21131688

Name	Calls	Self InsC	Self CycC	Total In...	Total CycC	Time Percentage
jpeg_idct_islow	2,430	4,273,895	5,194,185	4,273,895	5,194,185	24.58%
ycc_rgb_convert	318	3,269,358	4,392,111	3,269,358	4,392,111	20.78%
decode_mcu	405	2,393,716	2,819,463	3,526,272	4,239,323	13.34%
h2v2_fancy_upsample	318	2,249,850	2,663,868	2,249,850	2,663,868	12.61%
_start	1	1,838,702	2,097,783	16,897,127	21,131,688	9.93%
put_pixel_rows	318	1,027,458	1,643,698	1,043,676	1,664,311	7.78%
jpeg_fill_bit_buffer	12,239	877,201	1,086,007	1,074,089	1,348,686	5.14%
my_fread	18	442,602	590,397	442,602	590,397	2.79%

For projects with an ICE target, profiling data appears in the **Target Board Profiling** view.

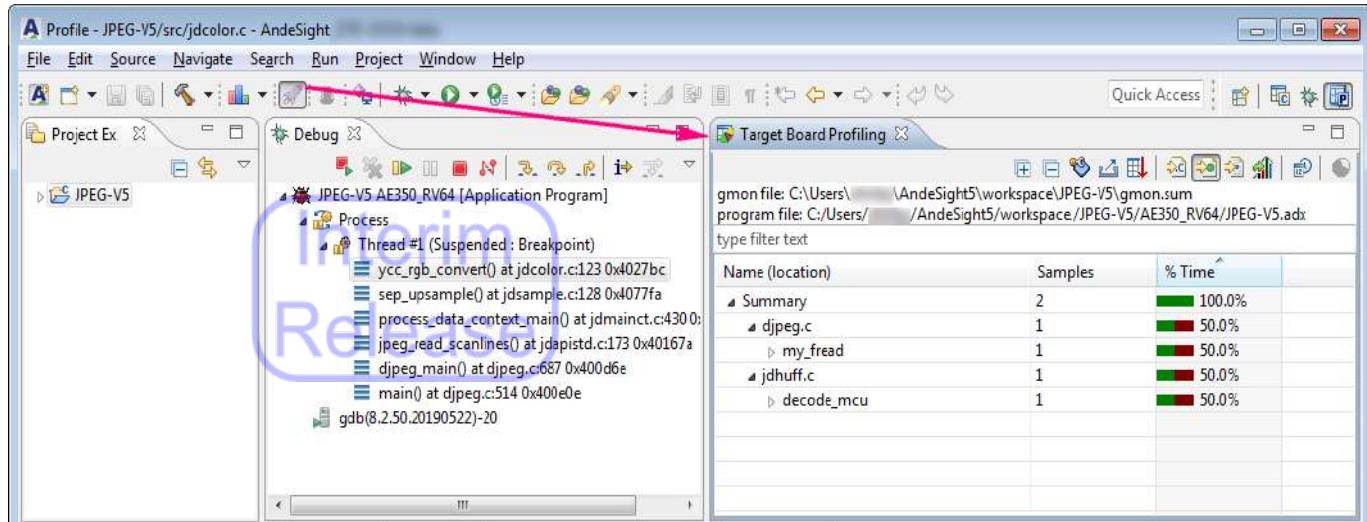
**Target Board Profiling**

gmon file: C:\Users\ AndeSight5\workspace\JPEG-V5\gmon.sum  
 program file: C:/Users/ AndeSight5/workspace/JPEG-V5/AE350\_RV64/JPEG-V5.adx

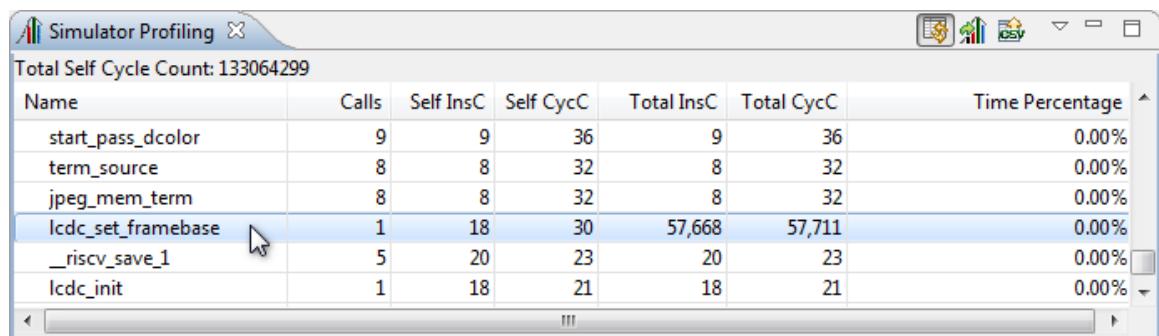
type filter text

Name (location)	Samples	% Time
Summary	4	100.0%
jdcolor.c	1	25.0%
ycc_rgb_convert	1	25.0%
jdsample.c	1	25.0%
h2v2_fancy_upsample	1	25.0%
jdhuff.c	2	50.0%
decode_mcu	1	25.0%
jpeg_fill_bit_buffer	1	25.0%

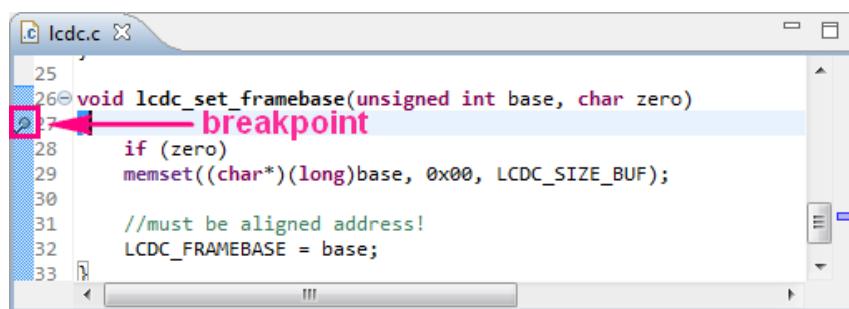
If you have specified to profile manually in the configuration, follow Section 2.4.4.1 to set a breakpoint at where you want to commence the profile session. Then, resume the program and click the button  (start profiling) on the AndeSight toolbar after the program hits the breakpoint and suspends. Profiling data will be displayed in profile views right away. To stop profiling during program execution, simply click the button  again.



**Step 4** Based on the profiling data, identify the bottleneck and double-click a function of interest to find it in the source code. Please note that the “Go-to-Source” feature functions only if the source code is available.



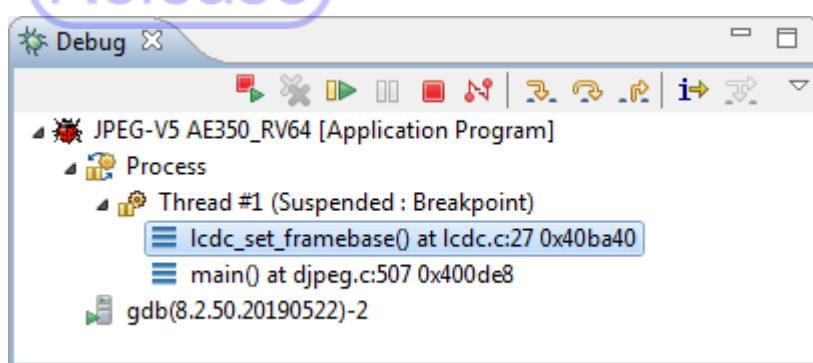
**Step 5** In the source code, follow Section 2.4.4.1 to set breakpoints on function statements of interest.



**Step 6** On the AndeSight toolbar, click  (Profile) to re-launch the

profile session. The program execution is suspended at the main function.

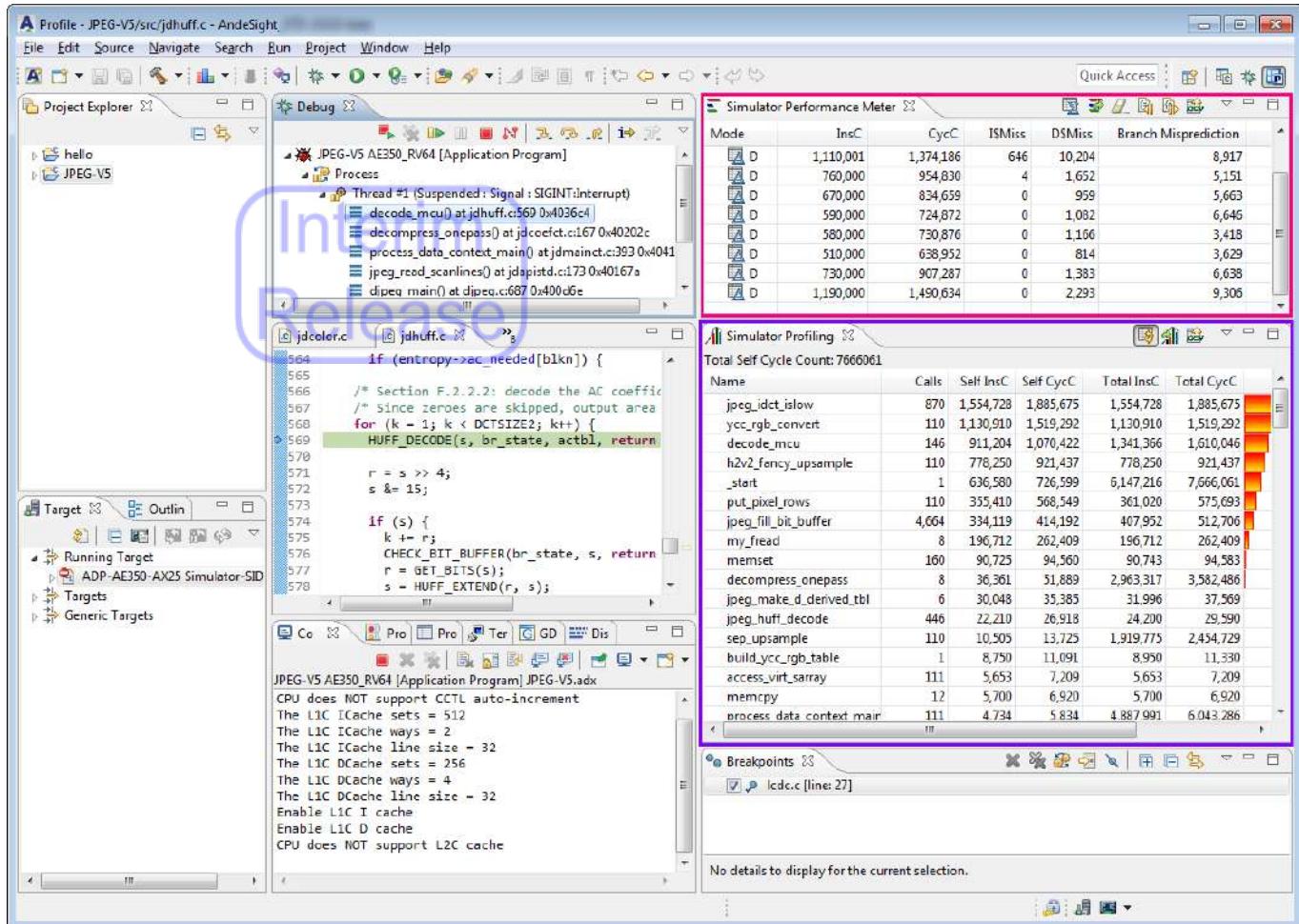
- Step 7** On the toolbar of the **Debug** view, click  (Resume) to continue the program execution. The program then suspends at where your first breakpoint is set.



- Step 8** On the toolbar of the **Debug** view, select  (Step Into) or  (Step Over) to execute function statements of interest. If the profile session needs to be started manually, click the button  on the AndeSight toolbar whenever you want to profile the program.

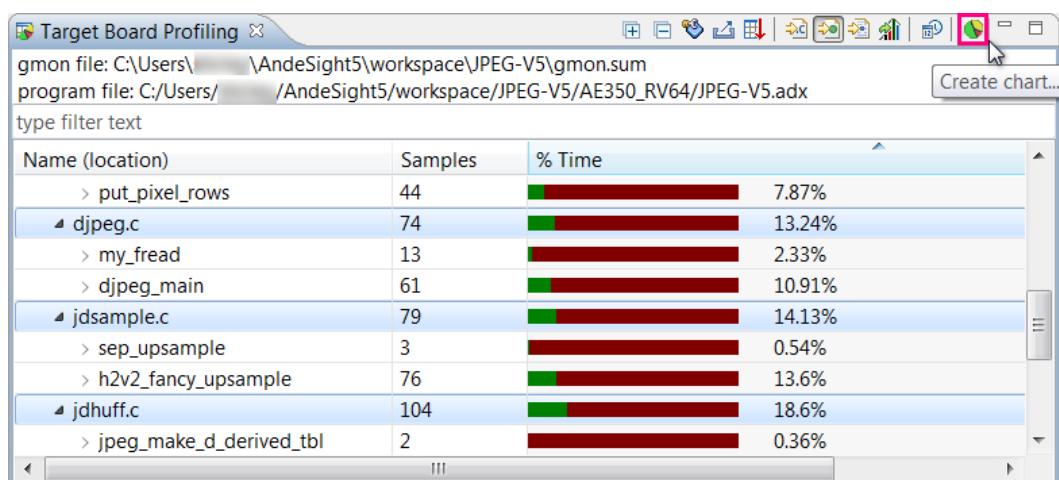
- Step 9** Check the profiling data again in the **Simulator Profiling** view (for simulator targets) or the **Target Board Profiling** view (for ICE targets).

For projects with a simulator target, the profiling results can be checked with respect to each execution step in the **Simulator Performance Meter** view.

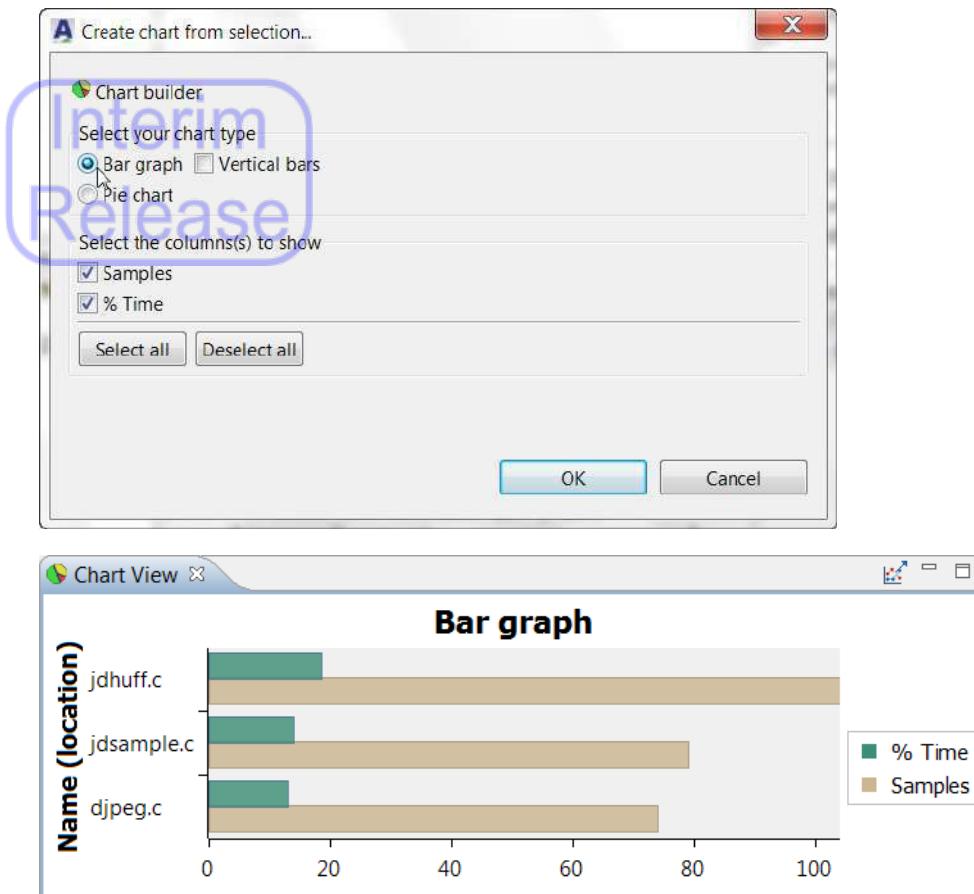


For projects with an ICE target, a graphic representation of the profiling information can be obtained as follows:

1. In the **Target Board Profiling** view, select the files or functions of interest and click (Create chart...) on the toolbar.



2. In the invoked **Create Chart from Selection** dialog, make a selection and click “OK” to create a chart.



**Step 10** Whenever assembly code is available, click “Instruction Stepping Mode [i→]” on the toolbar of the **Debug** view to switch to the **Disassembly** view and step through each instruction.

## 2.7. Code coverage

With the code coverage feature enabled, the execution percentage of your code can be checked in the **gcov** view. This makes it possible to detect unexecuted parts of your program and tune your code according to the coverage data.

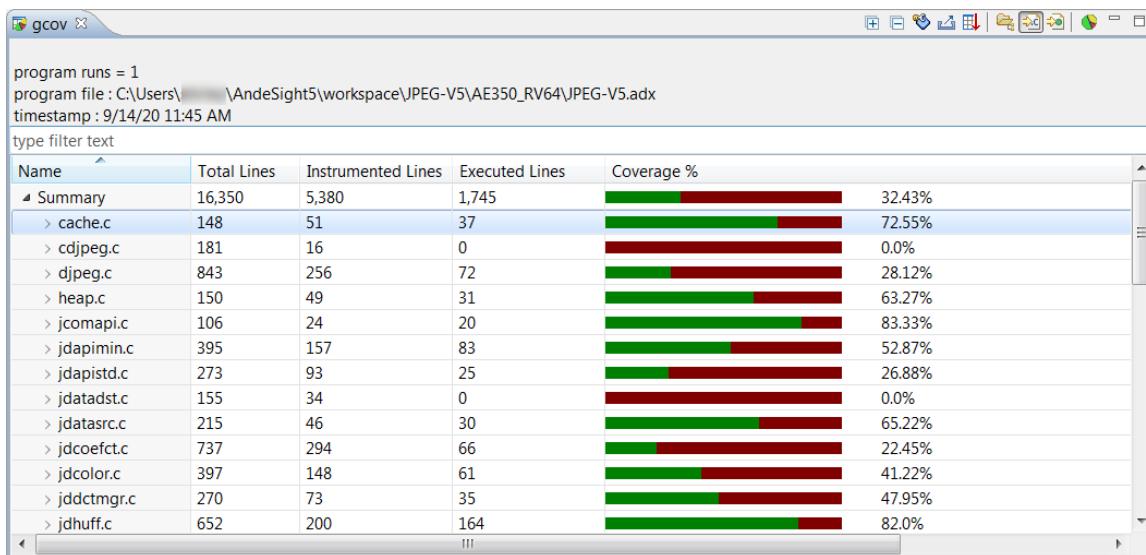
Note that Andes LLVM compiler has not supported the code coverage feature yet. Please use the GCC compiler to build programs that need code coverage analysis.

### 2.7.1. Code coverage views

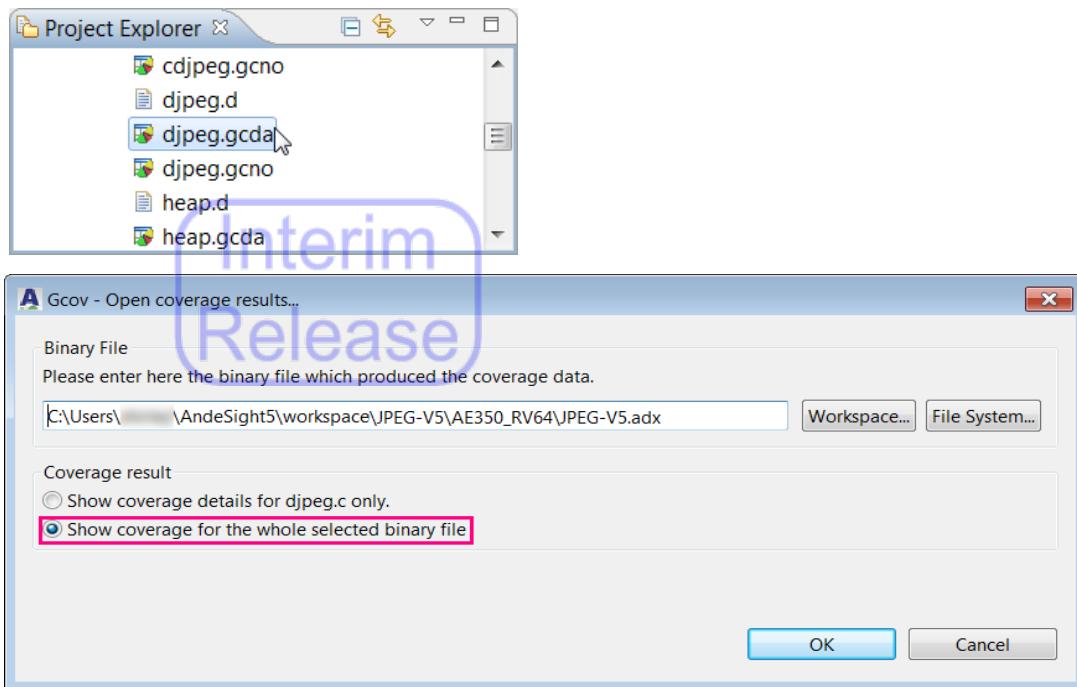
#### 2.7.1.1 Gcov view

The **gcov** view displays a summary of code coverage results in terms of files and functions.

The coverage data can be sorted into ascending or descending order by clicking a column header.



The view is launched automatically after execution of a project enabled with the coverage feature. To see the coverage result for whole program in the gcov view, you may also open it manually by double-clicking any .gcda file under your project and selecting the option “Show coverage for the whole selected binary file” in the invoked dialog. .gcda files usually can be found in **PROJECT\debug\src\**.



## Toolbar for the Gcov view

### Show/hide columns



Click this button to invoke a dialog to specify columns to display in the gcov view.

### Export to csv



Click this button to export the coverage data to a .csv file.

### Sorting



Click this button to invoke a dialog to specify the order in which coverage data in the **gcov** view is sorted.

### Sort coverage per folder



Click this button to sort the coverage result by folder.

### Sort coverage per file



Click this button to sort the coverage result by file.

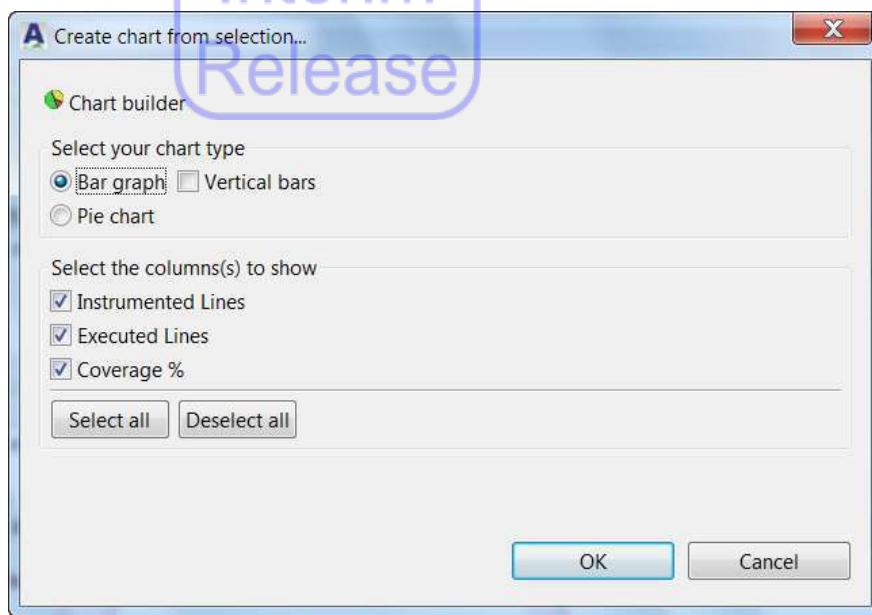
### Sort coverage per function



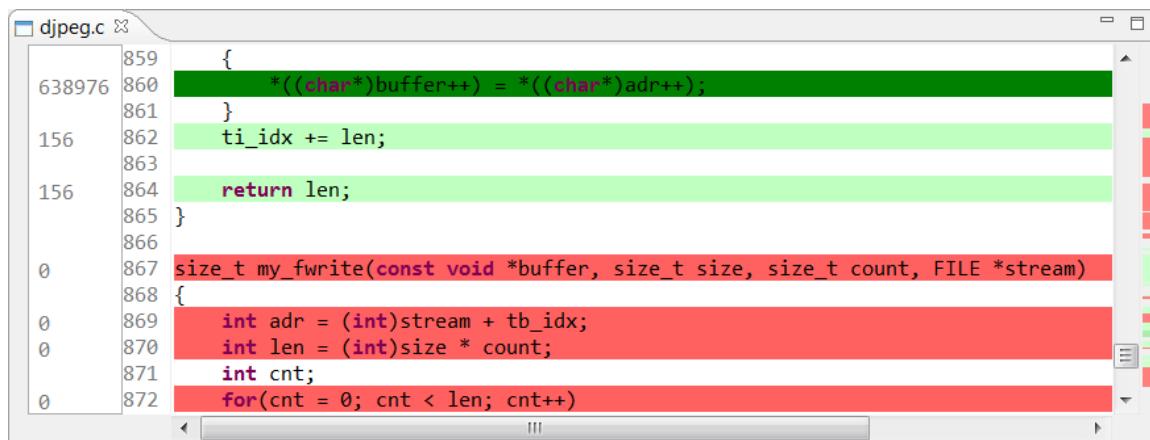
Click this button to sort the coverage result by function.

### Create chart...

Click this button to generate a bar graph, vertical bars, or pie chart based on coverage data of the selected files or functions. You may select the chart type and the information to be displayed in the invoked **Create Chart from Selection** dialog.



#### 2.7.1.2 Code editor with coverage highlighting



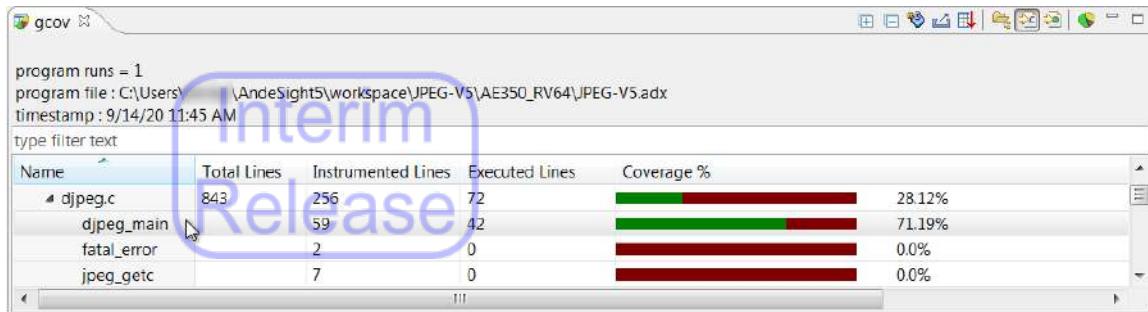
```
djpeg.c
859 {
638976 860   *((char*)buffer++) = *((char*)adr++);
861 }
156 862   ti_idx += len;
863
156 864   return len;
865 }
866
0 867 size_t my_fwrite(const void *buffer, size_t size, size_t count, FILE *stream)
868 {
0 869   int adr = (int)stream + tb_idx;
0 870   int len = (int)size * count;
871   int cnt;
0 872   for(cnt = 0; cnt < len; cnt++)

```

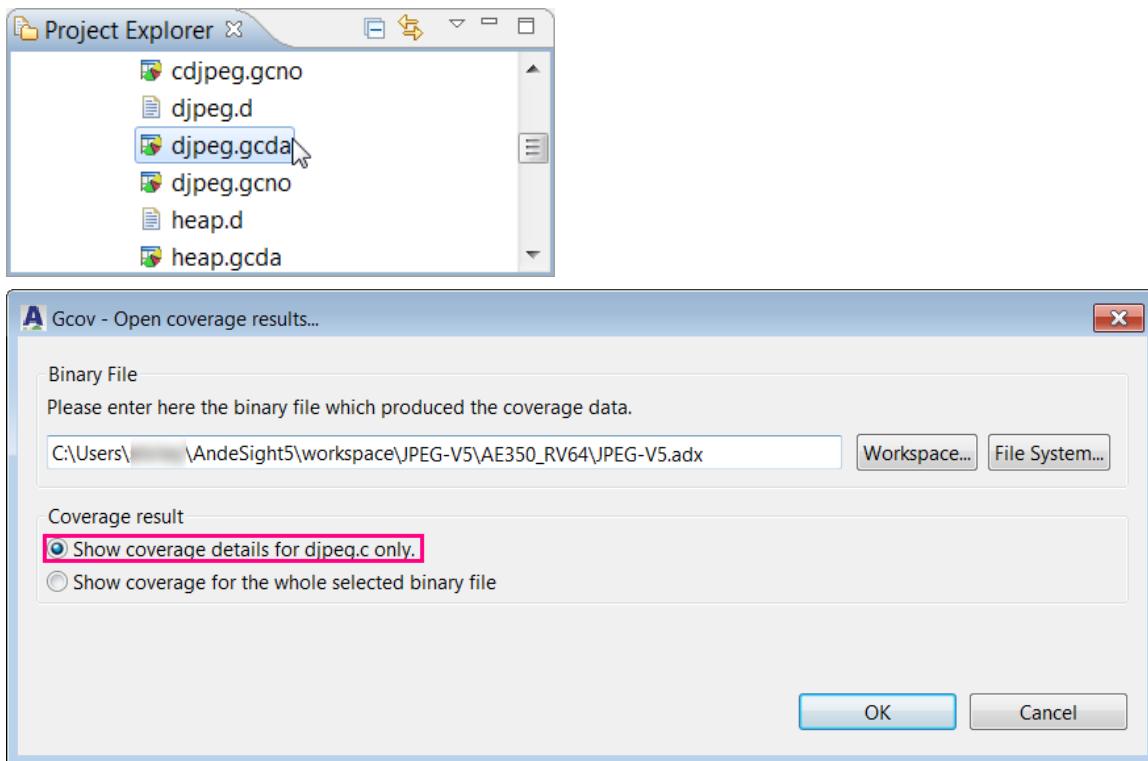
AndeSight also enables the highlighting of source code coverage in the code editor. The coverage is classified into three levels with corresponding color highlights:

- - Lines that are partially executed
- - Lines that are fully or frequently executed
- - Lines that are seldom or never executed

To examine source code with coverage highlighting, simply double-click the files or functions of interest in the **gcov** view.



You may also double-click a .gcda file under your project and select the option “Show coverage details for “XXX.c” only” in the invoked dialog. .gcda files usually can be found in **PROJECT\debug\src\**.



## 2.7.2. Engaging a coverage session

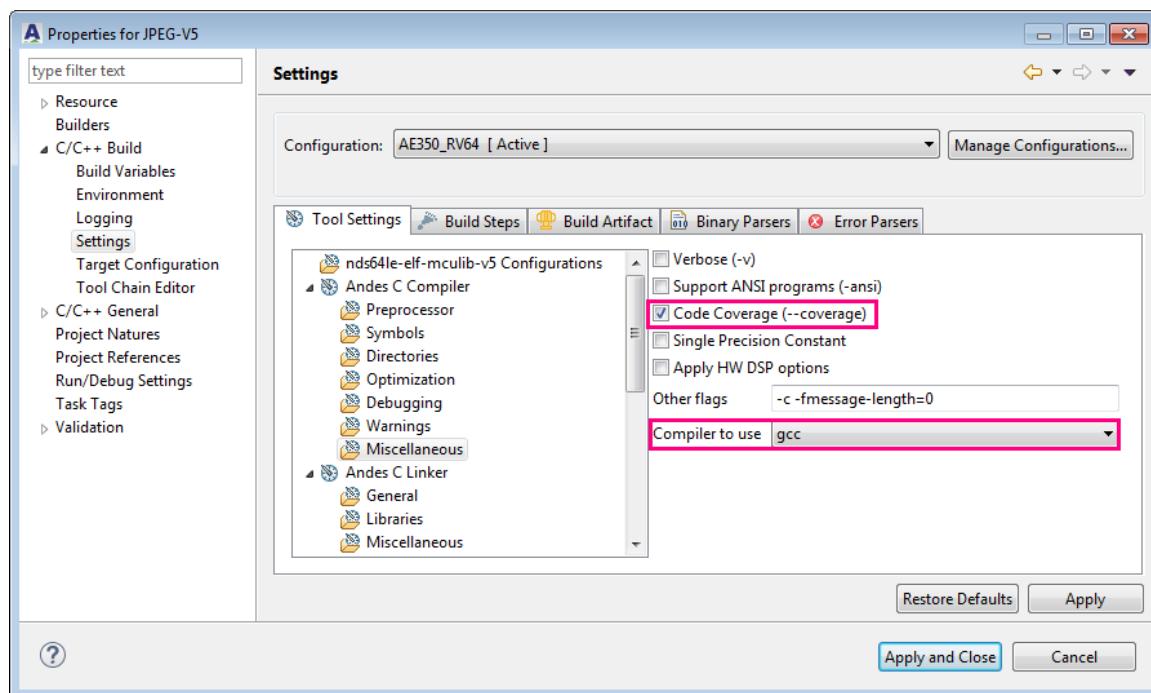
**Step 1** Follow Section 2.1.4 to configure the build settings of a project in **Project Explorer**. Be sure to turn to “Andes C Compiler > Miscellaneous” and select the “Code Coverage (--coverage)” option.

---

**NOTE**

The code coverage feature has not been supported by Andes LLVM compiler yet. Please use the GCC compiler for the feature.

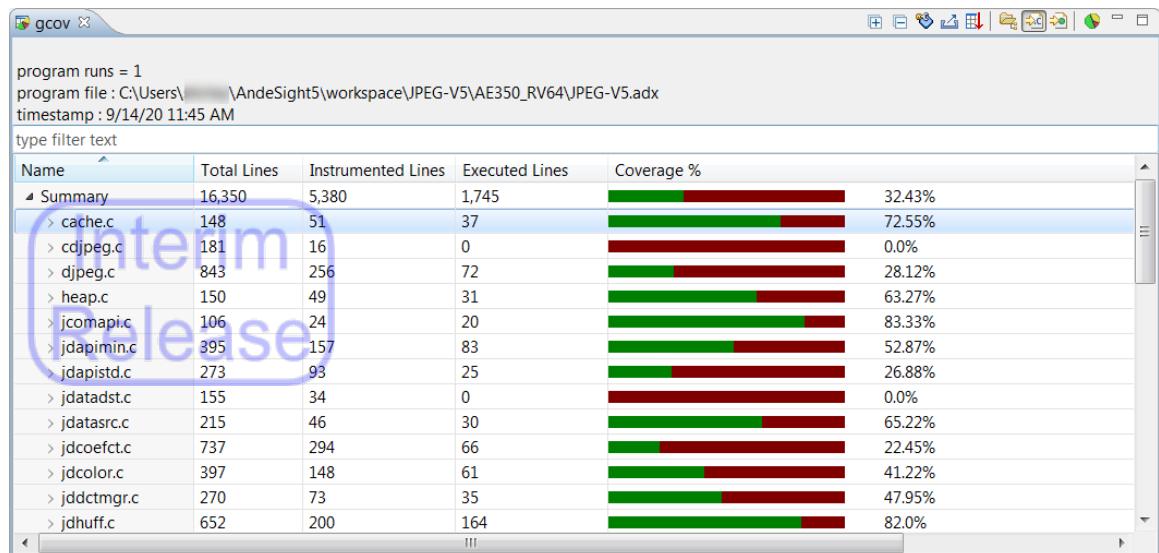
---



**Step 2** Click  (Build) on the AndeSight toolbar to build the project.

**Step 3** Click  (run) or  (Debug) on the AndeSight toolbar to run or debug the project.

**Step 4** Upon the completion or termination of program execution, the **gcov** view containing the coverage data is presented in AndeSight automatically.

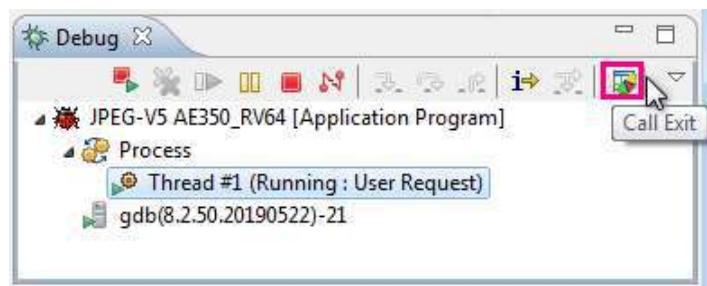


If you encounter any problems during debugging and would like to terminate the debug session and keep the coverage statistics, just click  (Call Exit) on the toolbar of the Debug view, as shown below. The coverage information will then be displayed in the pop-up gcov view. For startup demos with a simulator target, click Bit 1 of GPIO front-ends to display code coverage data during program execution. For startup demos with an ICE target, press the SW2 button on the board.

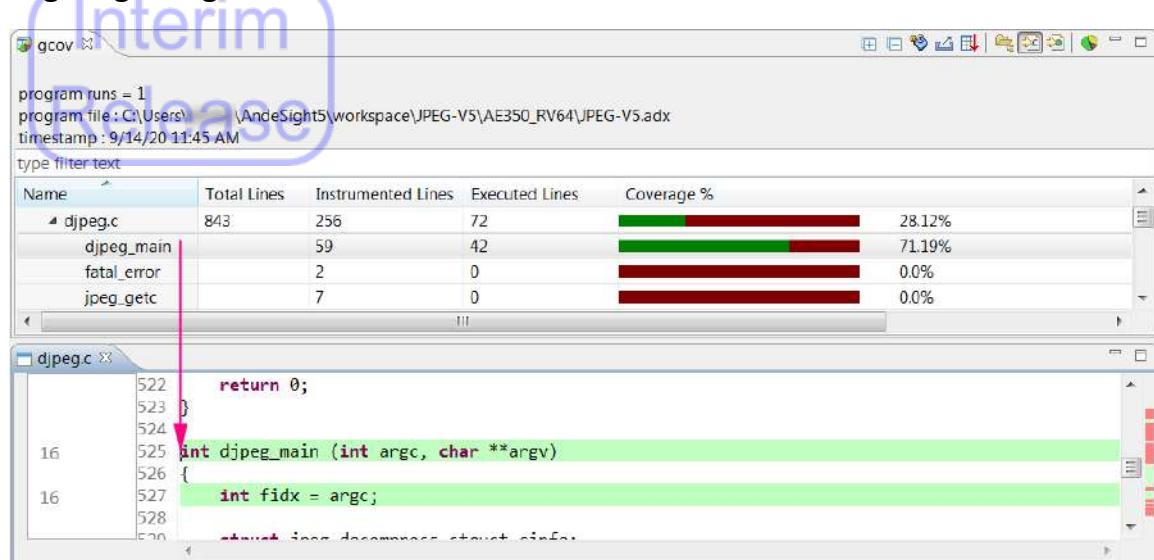
---

#### NOTE

The “call exit” method does not necessarily generate coverage information in all cases (e.g., prior to execution of global constructors or during critical sections of a program).



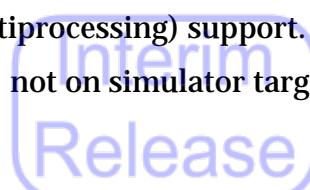
**Step 5** You can investigate line coverage in the source code by double-clicking a file or function of interest in the **gcov** view. The code editor then appears showing the source code with coverage highlighting.



**Step 6** On the toolbar of the **Debug** view, click  (run) or  (Debug) again to restart the program. Coverage data in the **gcov** view will be updated following termination of program execution.

## 2.8. Development on a multicore system without SMP support

The subsections below describe how to debug a project on a multicore system without SMP (Symmetric Multiprocessing) support. Such multi-core debugging is currently supported on ICE targets only (i.e., not on simulator targets).



### 2.8.1. Creating a multi-core chip profile

Follow Section 2.2.1.1 or Section 2.2.1.2 to create a multi-core chip profile by writing a targetboard.properties file or using the chip profile editor in AndeSight, but take note of the following:

- To define a targetboard.properties file for a multi-core chip, make sure you specify the number of CPU cores along with the type, CPU registers file, and whether RISC ISA is used for each core. For the preferred toolchain, software settings and configuration files that will be applied by default to this chip, assign them in their respective section. As to settings or files specific to a particular CPU core, designate them separately with the label of that CPU (e.g., CPU1). The following example is a targetboard.properties file for a chip pre-integrated with two N25F cores.

```

==== Chip Profile ====
chip. id=ADP-AE350-N25F-Multicore

==== CPU Description ====
cpu. cores=2                                #number of cores on the chip

cpu0. type=N25F                             #descriptions for CPU0
cpu0. isa. reducereg=1
cpu0. registers=../../CPU/V5-32. crgs

cpu1. type=N25                               #descriptions for CPU1
cpu1. isa. reducereg=1
cpu1. registers=../../CPU/V5-32. crgs

cpu1. linker. script=ae350-hello-loop.ld    #linker script specific to CPU1

==== SoC Registers Description File ====
soc. registers=../../SoC/AE350-4GB. regs

```

```

==== Memory Map Description File ====
memory.map=..../MemoryMap/AE350-4GB-memory.mem

==== Preferred Tool chain ====
preferred.toolchain=nds32le-elf-mculib-v5d

==== Linker Script File ====
linker.script=ae350-hello.ld

==== Flash Burner Settings ====
flash.driver=target_burn_frontend
flash.driver.custom.ui=N
flash.target=ae350
flash.start.address=0x0
flash.controller.address=
flash.miscarguments=
flash.target.burn.bin=target_SPI_v5_32.bin

==== ICEman miscellaneous arguments ====
iceman.misc.args=-Z v5

==== Simulator Configuration File ====
vep=..../vep/tmp/ADP-AE350-N25F.vep

==== VEP Front-End Filter ====
#vep.vepifilter=SDC:CFC

==== Simulator VIOS/Raw mode ====
vep.vepiomode=RAW

==== Simulator miscellaneous arguments ====
#sid.misc.args=

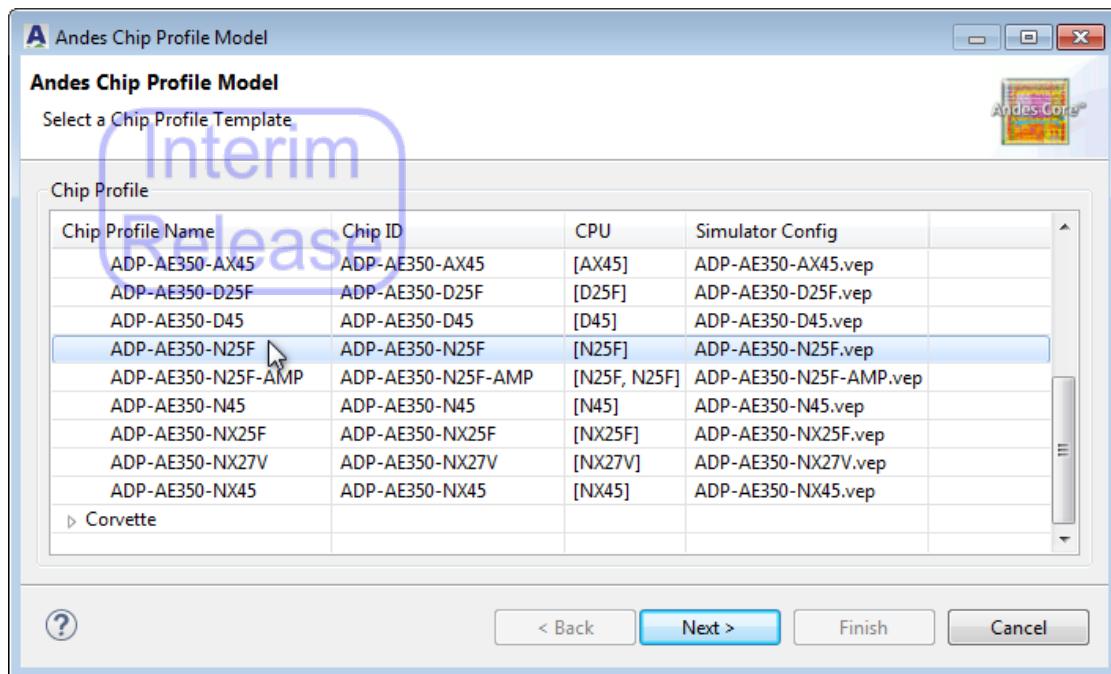
==== Removable (Y/N) ====
#removable=Y

==== vep2conf arguments ====
vep2conf.miscarguments=-a v5

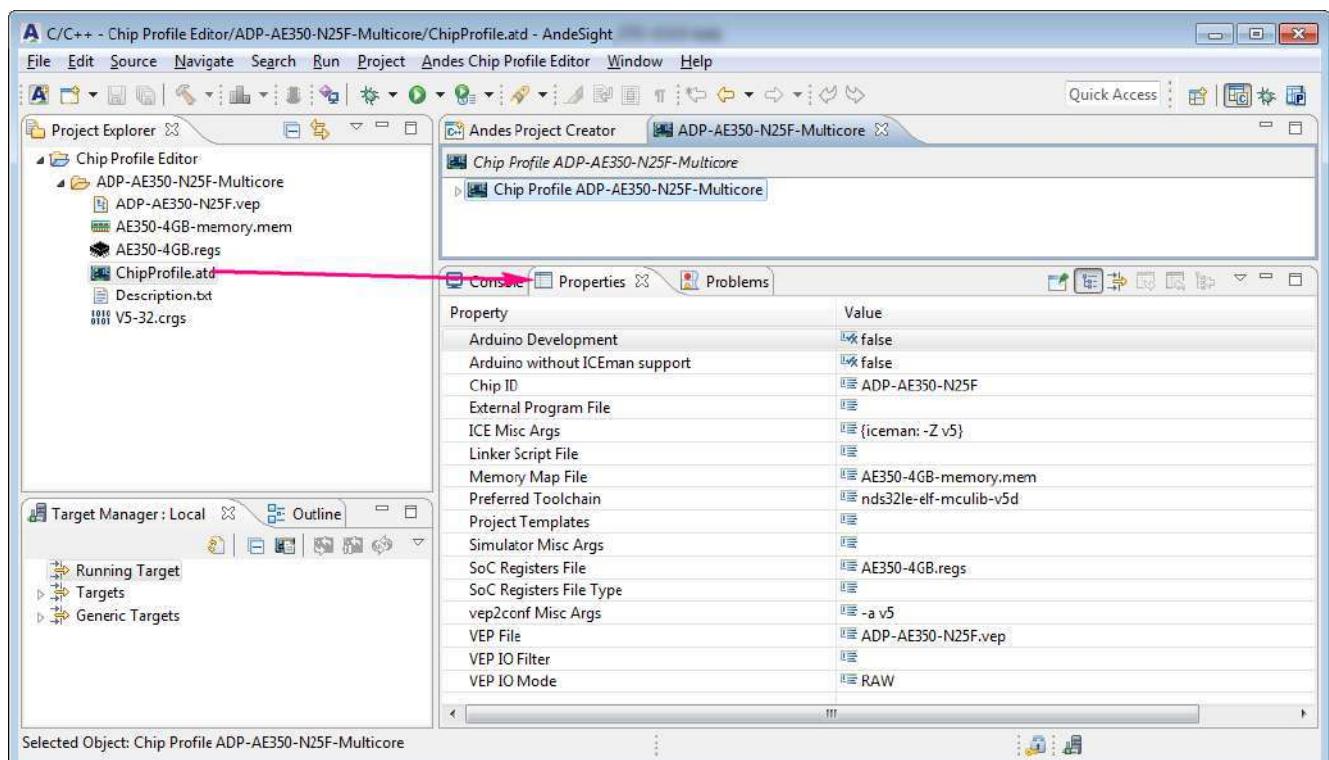
```

- When creating a multi-core chip profile using AndeSight chip profile editor, select the first CPU core (CPU0) of the chip on the **Andes Chip Profile Model** dialog and specify the default settings and configuration files for the chip by double-clicking **ChipProfile.atd** and

editing it in the **Properties** view.



(Specifying the first CPU for a multi-core chip profile)

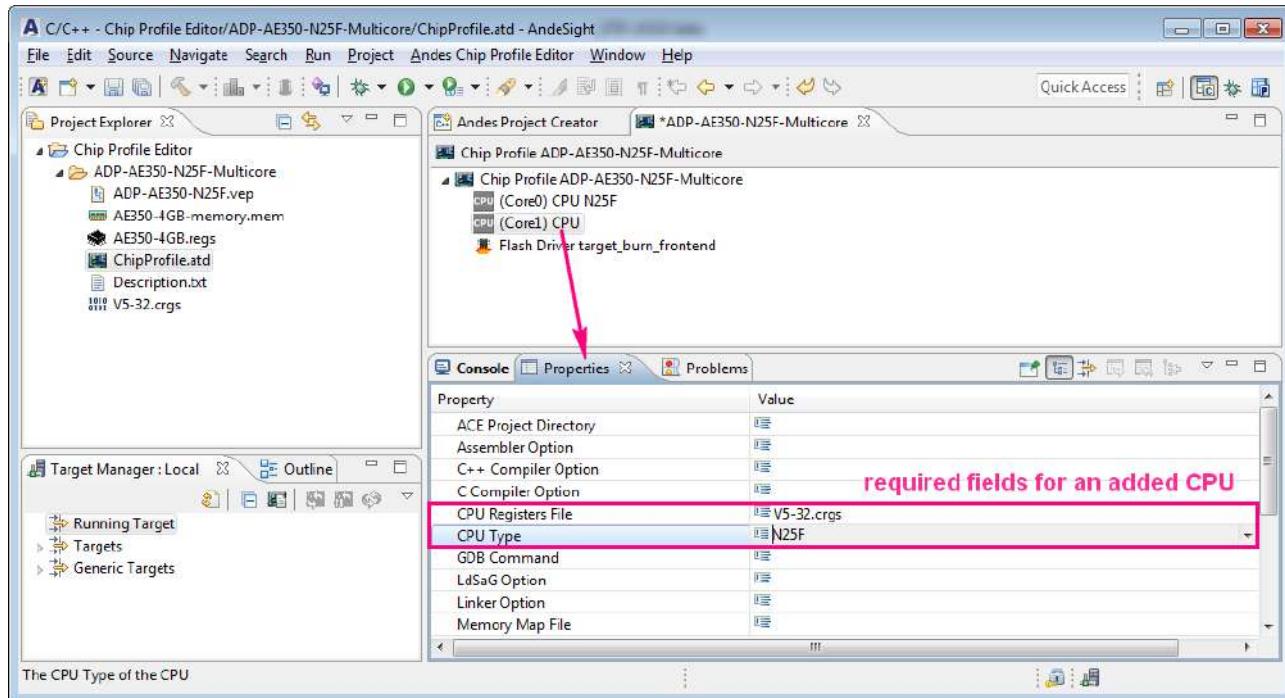


(Editing ChipProfile.atd to specify default settings and files for a multi-core chip)

To add a core to the chip profile, right-click the chip profile in an invoked view and select “New Child > CPU”.

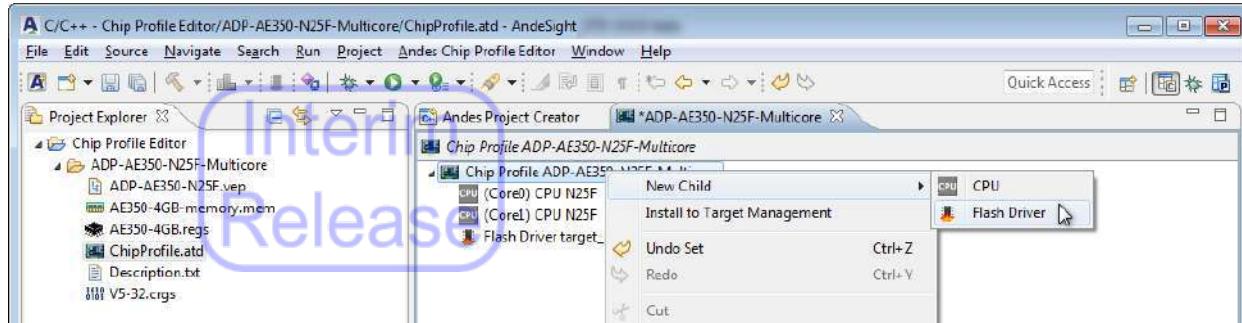


When a CPU is added, all cores in the chip profile will be numbered and labelled (e.g., Core0, Core1) automatically in AndeSight. Specify the CPU type and CPU registers file for the added CPU core in the **Properties** view. If the core needs any specific software settings or configuration files that are different from the default values defined in **ChipProfile.atd**, specify them in the **Properties** view as well.

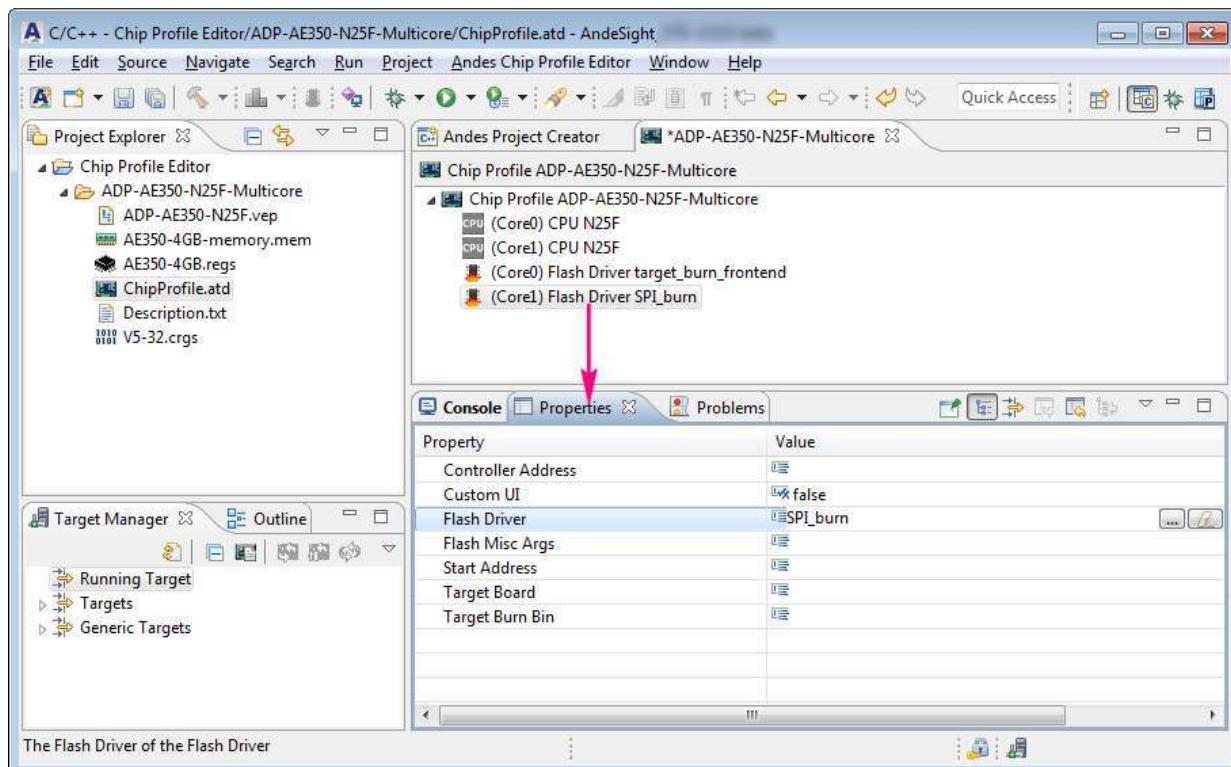


If the core uses a flash driver or burner settings distinct from those for Core0, create a dedicated flash driver configuration by right-clicking the chip profile in the invoked view, selecting “New Child > Flash Driver”, and specifying the flash driver settings in the

**Properties** view. Multiple flash driver configurations in a chip profile will automatically bear CPU information in AndeSight.



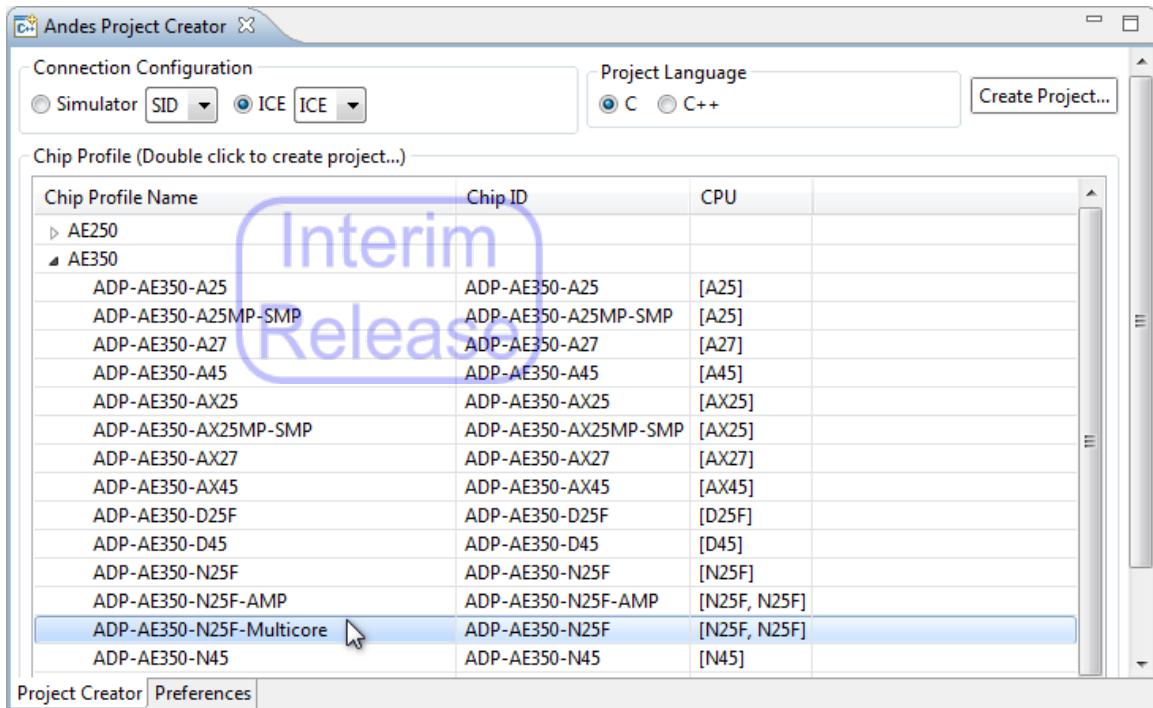
(Adding a flash driver configuration to a chip profile)



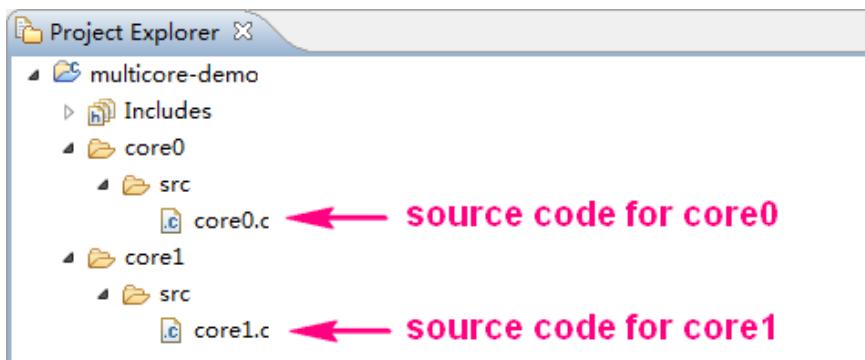
(Specifying properties for the added flash driver configuration)

### 2.8.2. Creating a multi-core project

After a multi-core chip project is created, it will be listed among pre-defined targets in the **Andes Project Creator** view. Follow Section 2.1.1.1 to specify the chip profile and create a project to run on the multi-core target.



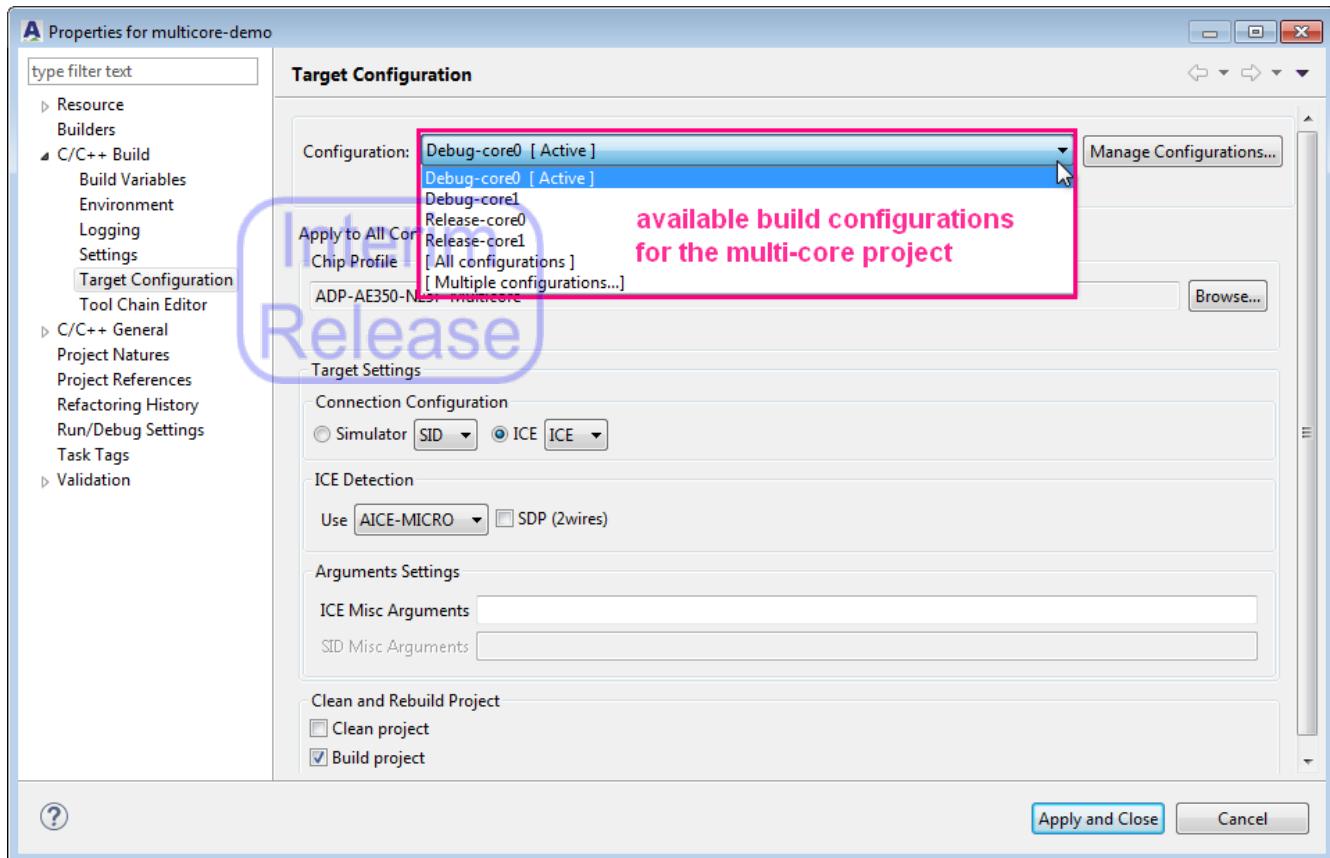
Find the created multi-core project in the **Project Explorer** view. The project contains sub-folders for respective cores. Please follow Section 2.1.2 to include source files, header files or linker scripts for each core in its source folder (**PROJECT\CORE\src**).



### 2.8.3. Configuring build settings and building the project

A multi-core project contains default Debug and Release build configurations for each core.

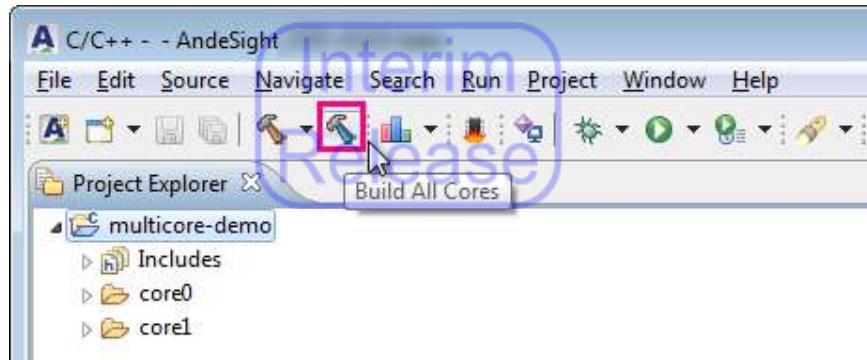
Follow Section 2.1.4.2 to specify a build configuration and configure its settings on the **Settings** page of the **Properties** dialog.



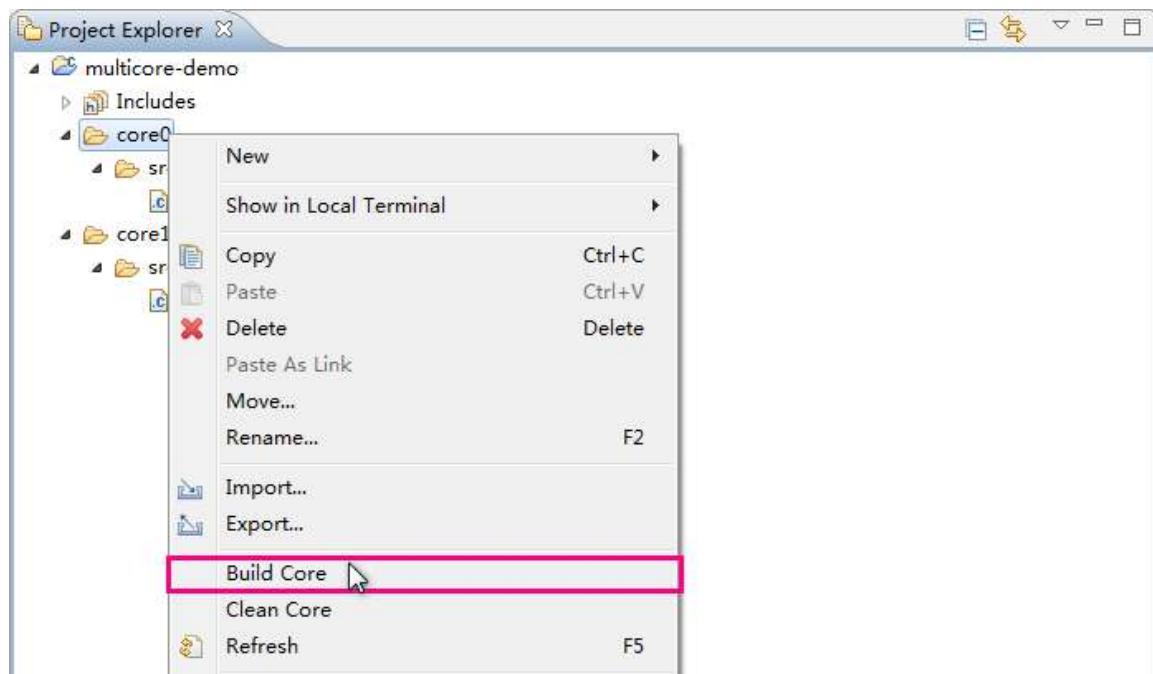
If you want to change build settings for a specific core afterwards, right-click on the core folder under the project in the **Project Explorer** view, and select “Build Settings” from the pull-down menu to access the settings and make modifications.



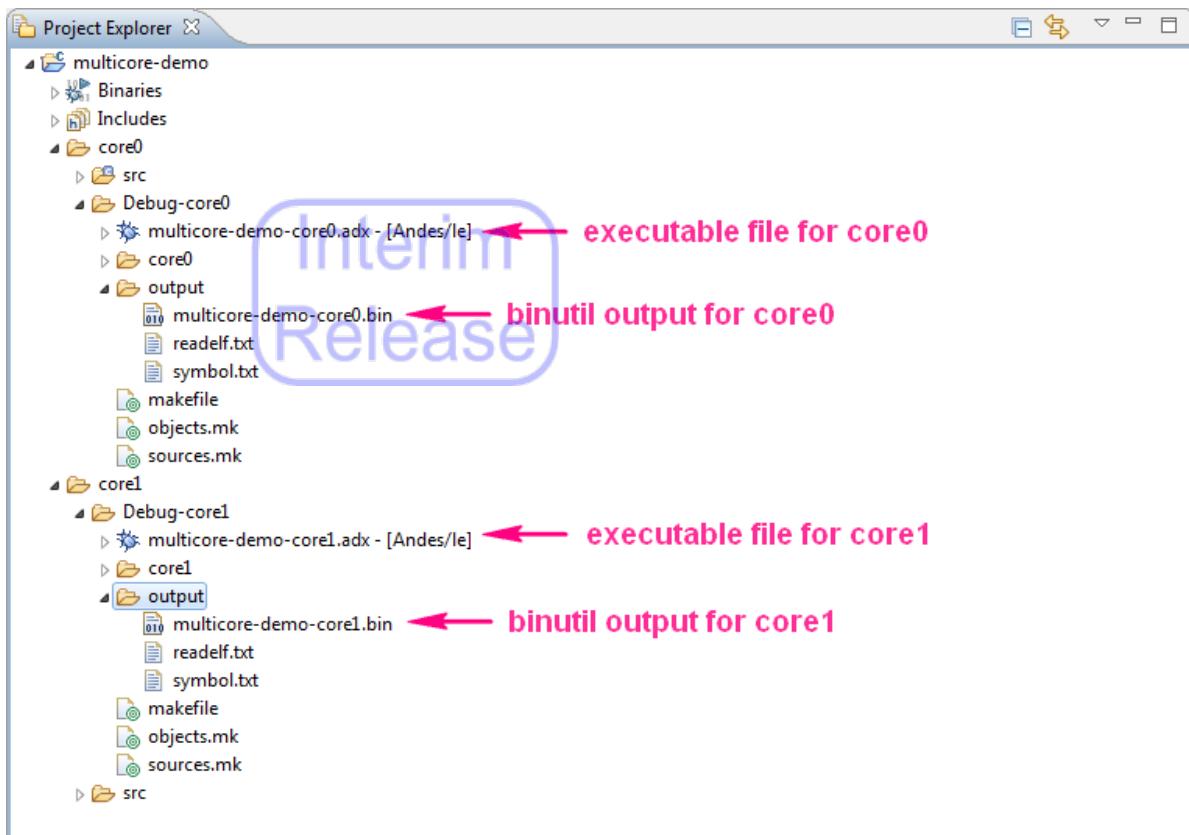
To commence the build process for all cores associated to the project, select the project in the **Project Explorer** view and click on  (Build All Cores) on the AndeSight toolbar. You may also right-click the project folder to trigger the pull-down menu and select “Build Project”.



If you intend to commence a single-core build, right-click the folder for that core under the project and select “Build Core” from the pull-down menu.



After a successful build, the executable file and object file for each core are generated under **PROJECT\CORE\[ Debug | Release ] - CORE** and the binutil output under **PROJECT\CORE\[ Debug | Release ] - CORE\output**.

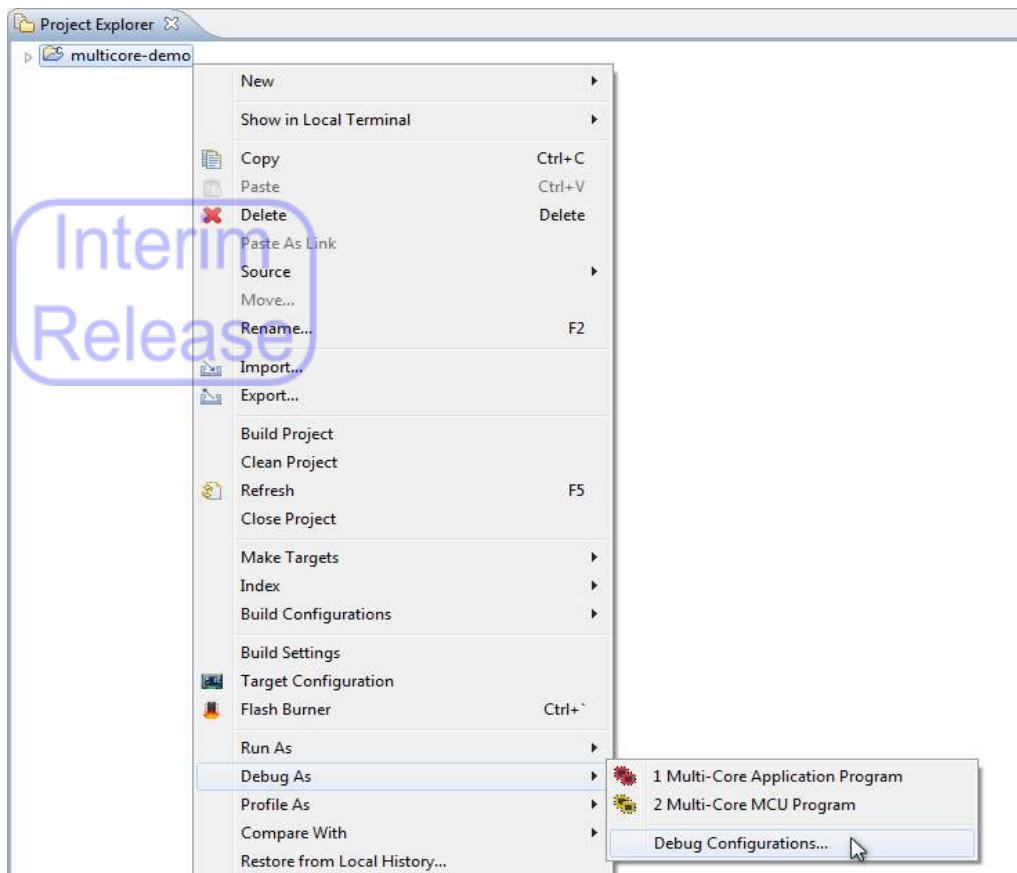


## 2.8.4. Launching a debug session for the multi-core project

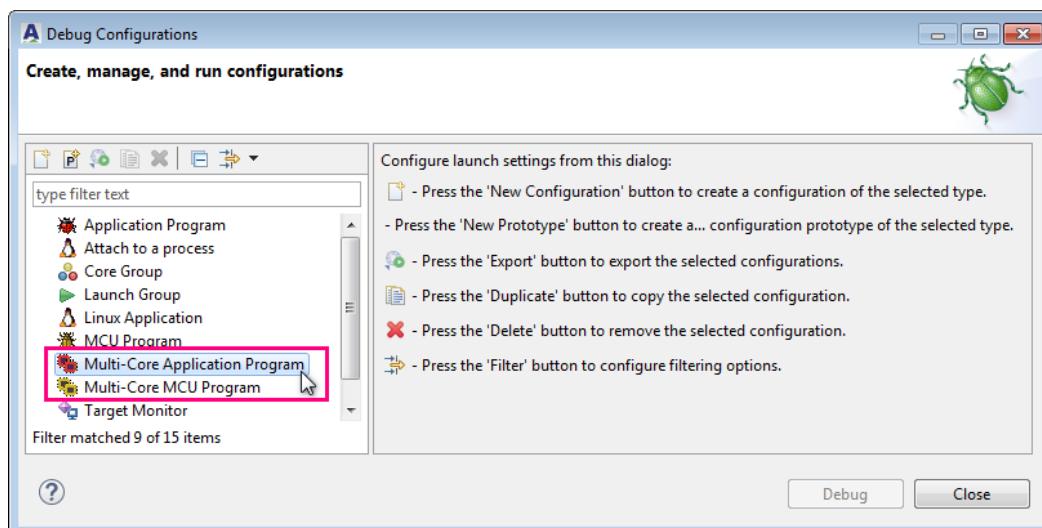
The following introduce two methods to launch a debug session for your multi-core project: standard launch and quick launch. The standard launch allows you to create a multicore debug configuration and specify detailed settings for your debug session whereas the quick launch enables you to efficiently launch a multicore debug configuration with default settings.

### 2.8.4.1 Standard launch

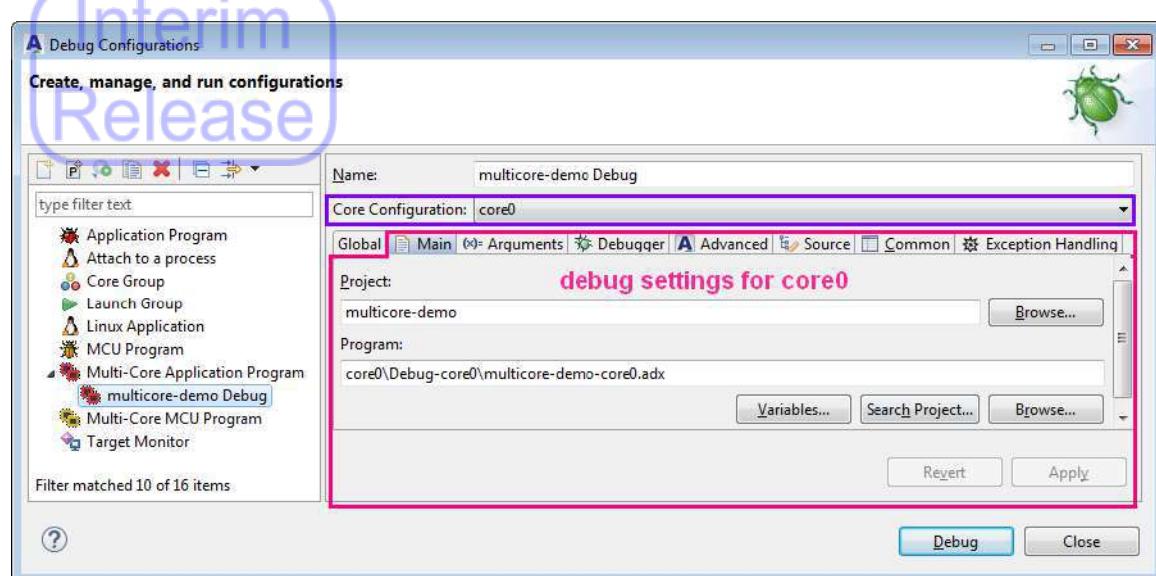
**Step 1** Right-click a multi-core project in the **Project Explorer** view and select “Debug as > Debug Configurations...” on the pull-down menu.



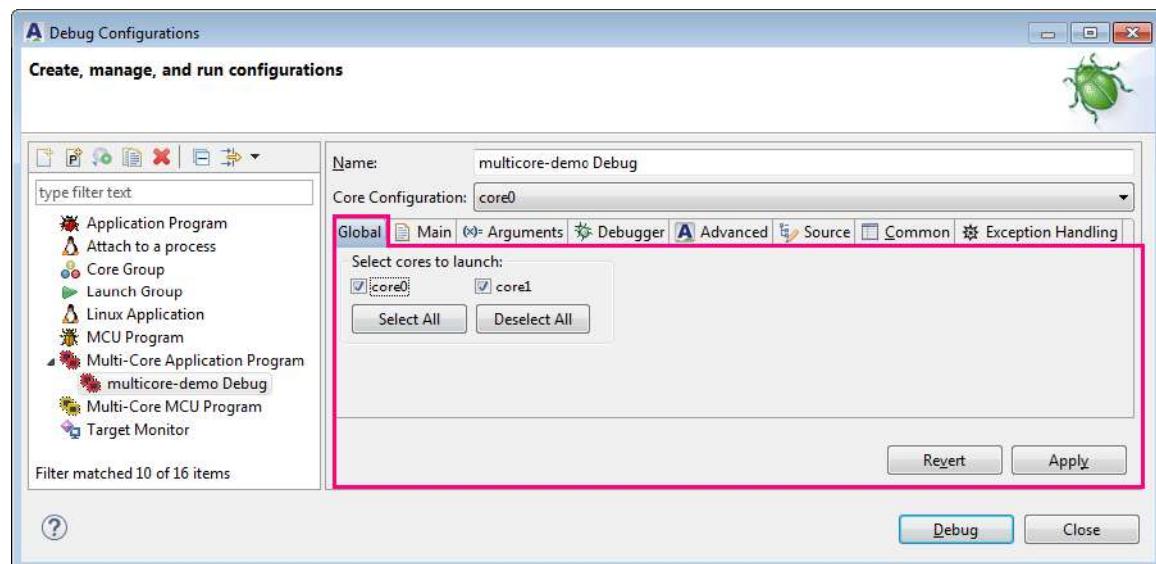
**Step 2** On the **Debug Configurations** dialog, double-click the configuration type “Multi-Core Application Program” (for multi-core targets with system service) or “Multi-Core MCU Program” (for multi-core targets without system service) in the navigation pane. A debug configuration is created for the multi-core project.



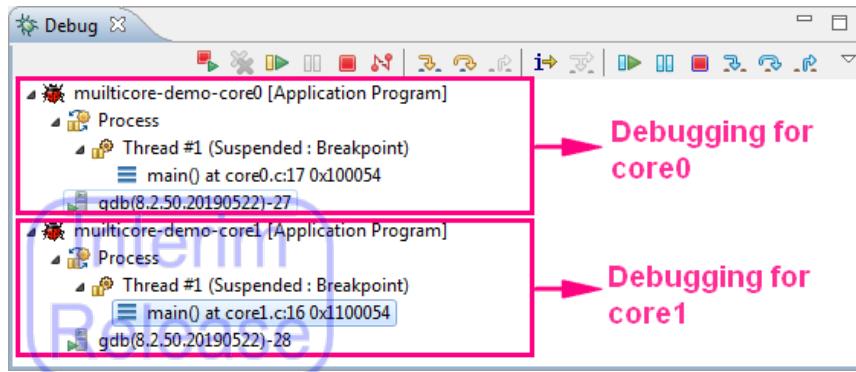
**Step 3** Configure debug settings for each core by specifying a core from the **Core Configuration** drop-down list first and referencing Section 2.4.3.1 to modify debug settings from the **Main** tab to the **Exception Handling** tab.



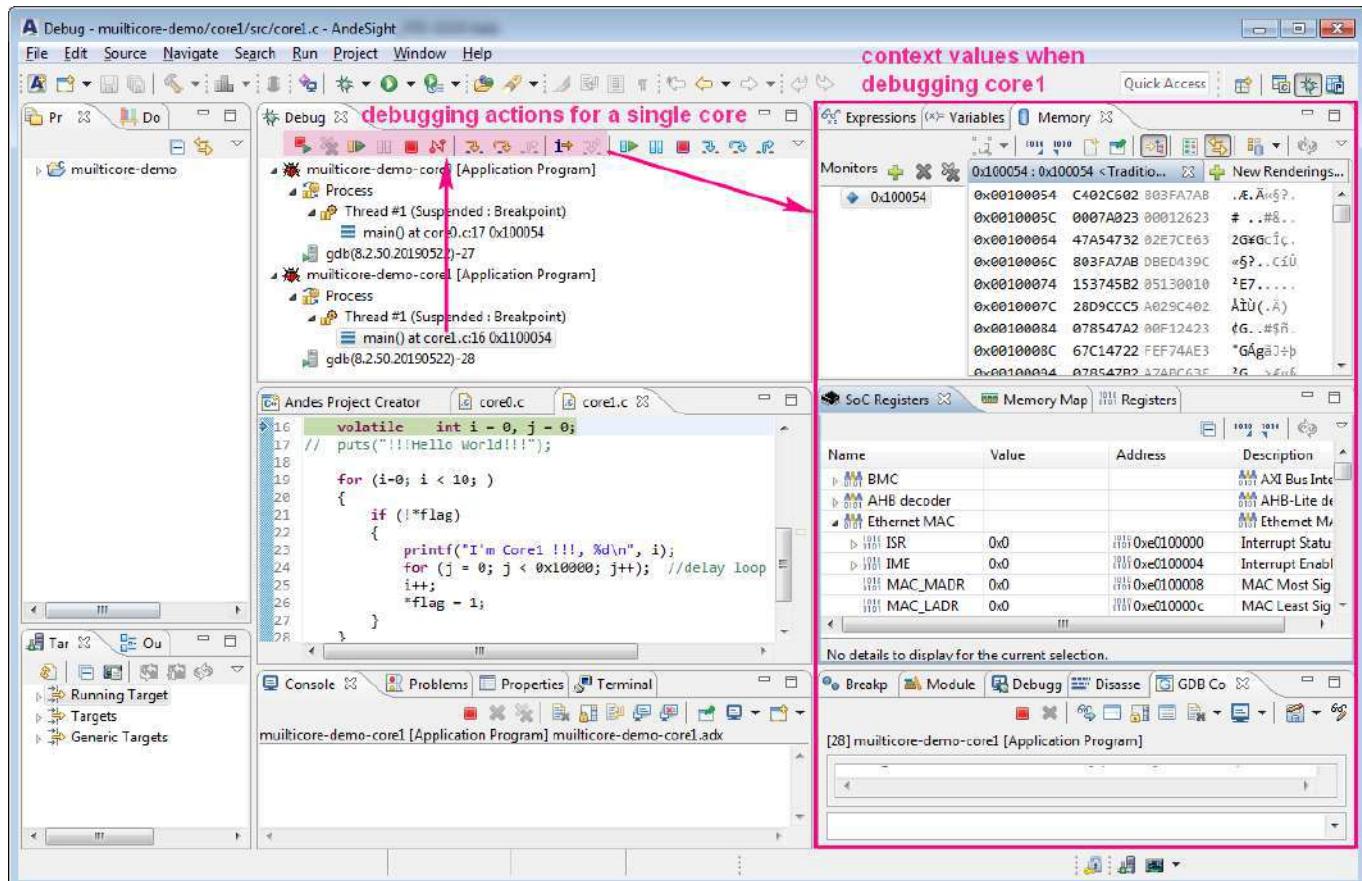
**Step 4** Click the **Global** tab and specify which core(s) the debug session is launched for. Make sure you select at least one core and then click “Apply” and “Debug” to launch the debug session.



**Step 5** In the invoked **Debug** perspective, the **Debug** view displays the stack frame of suspended threads for the core(s) you are debugging.

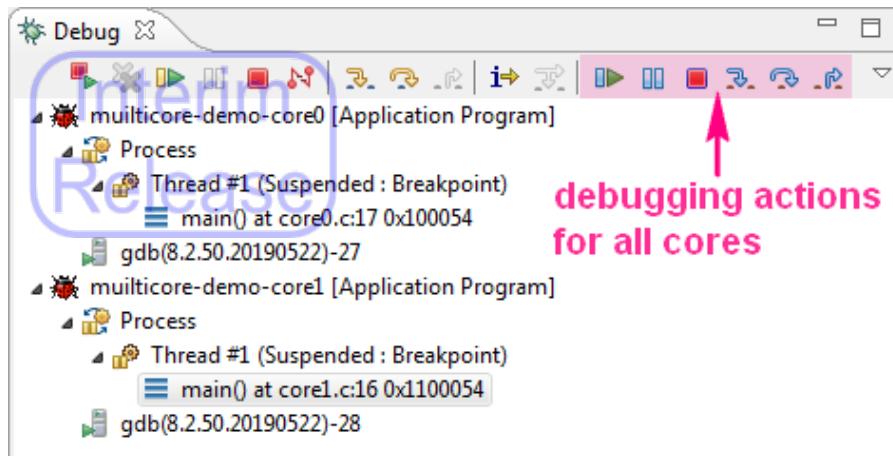


**Step 6** To debug one specific core, select the stack frame for the core in the **Debug** view and click toolbar buttons like (Resume), (Step Into) or (Step Over) in this view. The updated context values for the specified core during program execution can be inspected from other debug views (e.g., **Registers** view, **Variables** view, **Memory** view) in the perspective.

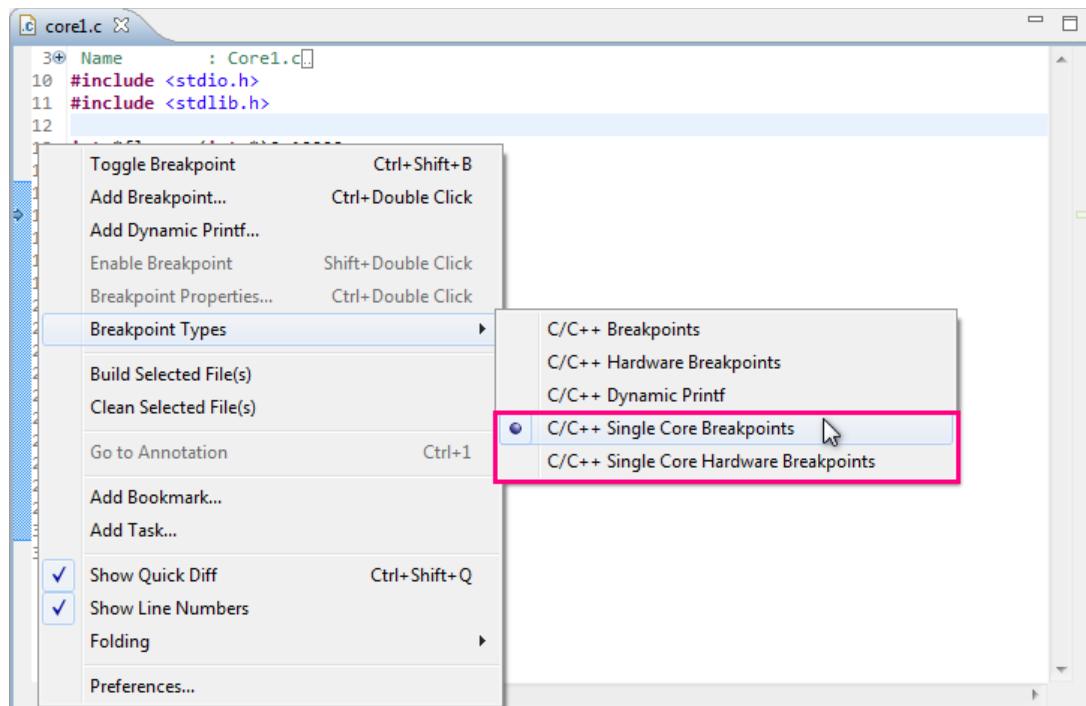


If you want to perform debug actions across all cores, click toolbar

buttons like  (Group Resume),  (Group Step Into),  (Group Step Over) or  (Group Step Return) in the Debug view.

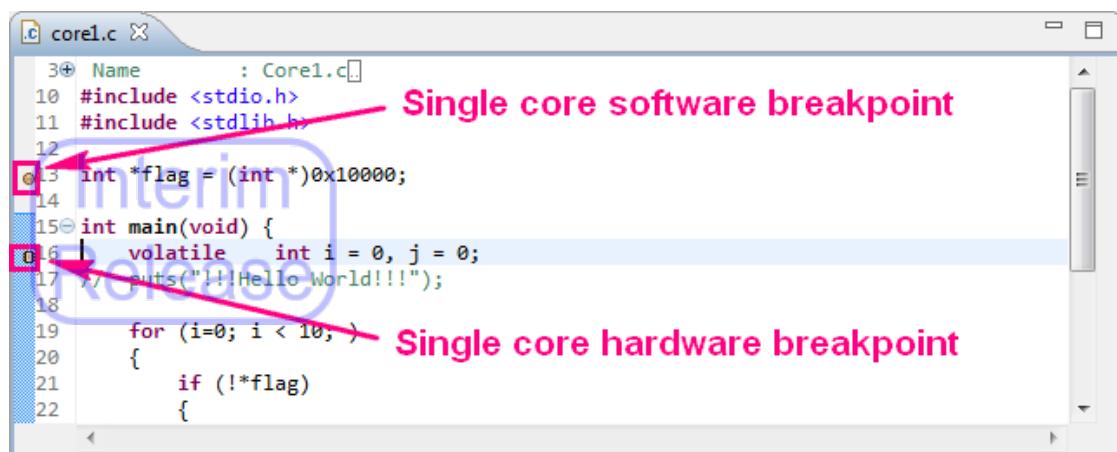


**Step 7** To set a breakpoint for all cores, just follow instructions in Section 2.4.4.1. If you need to apply a breakpoint to a specific core, specify the breakpoint type “C/C++ Single Core (Software) Breakpoints” or “C/C++ Single Core Hardware Breakpoints” before toggling the breakpoint.



The toggled single core breakpoints are denoted by yellow icons in

code editors and the **Breakpoints** view.



---

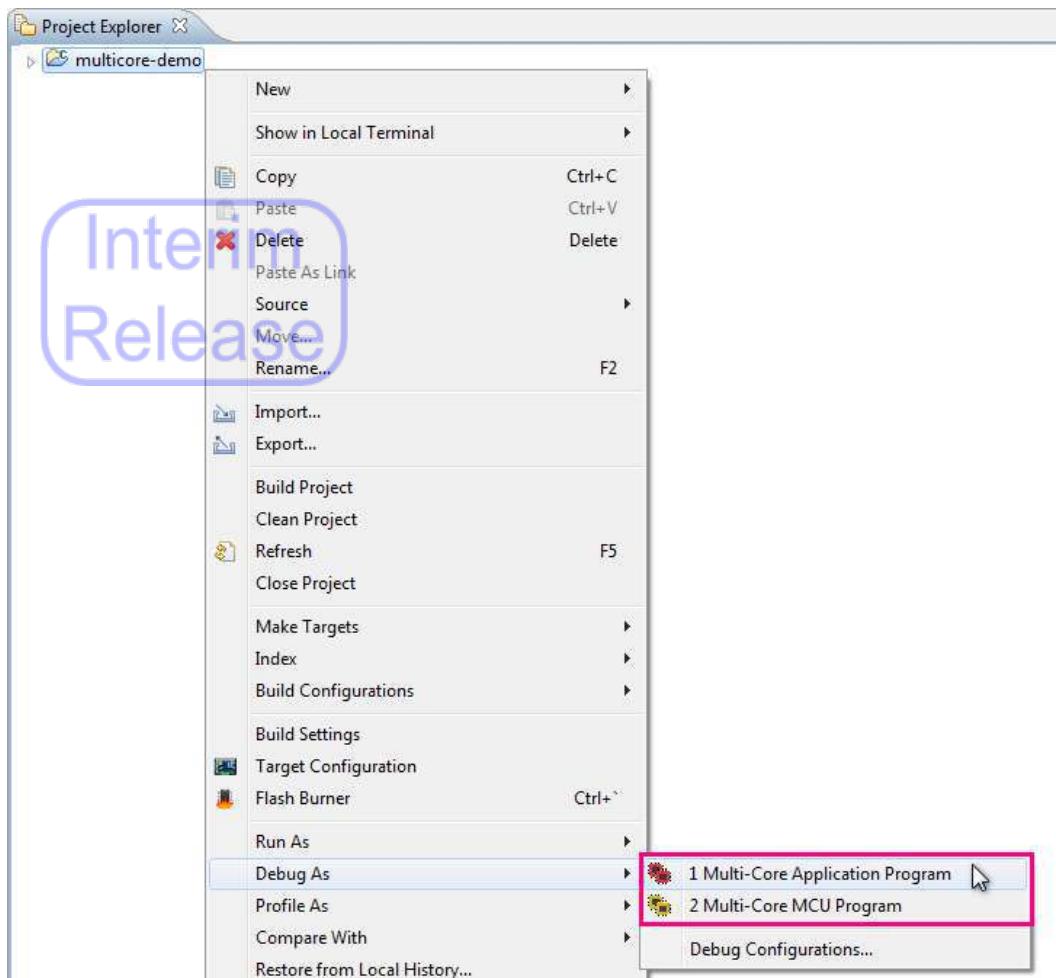
#### NOTE

Single core software breakpoints require that each core uses a separate memory space to run its code. If you want to set a single-core breakpoint on shared code executed by multiple cores, make sure to specify the type “C/C++ Single Core Hardware Breakpoints” as the software counterpart may not work properly.

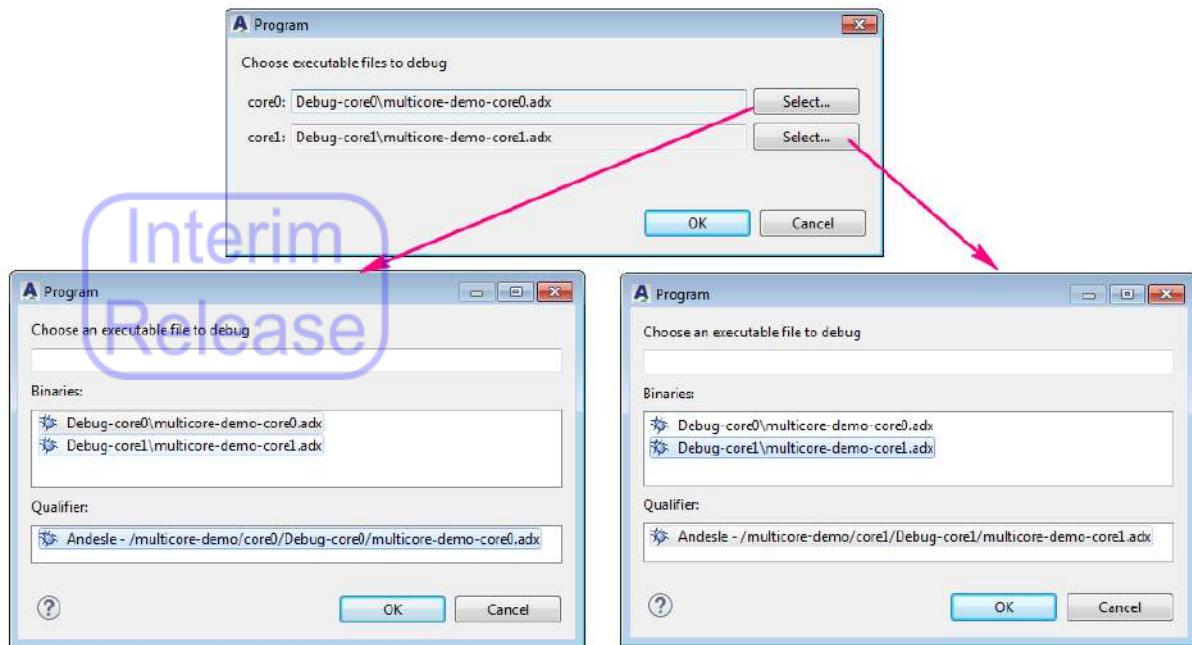
---

#### 2.8.4.2 Quick launch

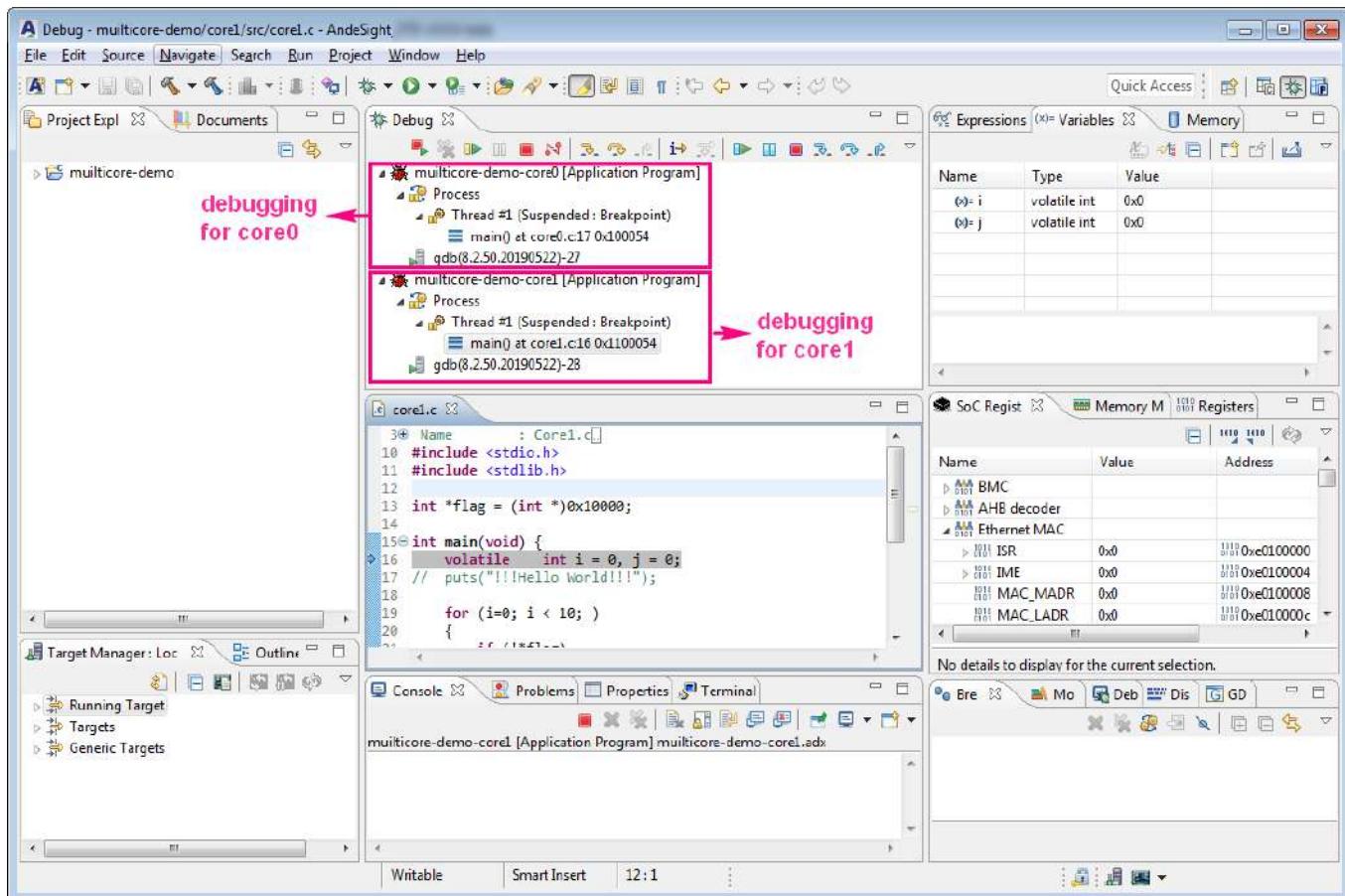
**Step 1** Right-click the multi-core project in the **Project Explorer** view and select “Debug as > [Multi-core Application Program|Multi-Core MCU Program]” on the pull-down menu.



**Step 2** In the invoked **Program** wizard, click “Select...” to specify a program executable for each core and click “OK”.



**Step 3** The debug sessions for respective cores are launched simultaneously.  
 Use toolbar buttons in the **Debug** view to proceed the debugging.



## 2.9. Tracing RTOS applications

The AndeSight IDE is capable to trace the execution of RTOS applications and provide insights into their runtime behaviors against time. By enabling the trace functionality in the debug configuration of a RTOS application, the execution of tasks, interrupts and events within the specified period of time will be collected, recorded and visualized in the AndeSight IDE. This facilitates you to inspect program execution on the system, identify program issues and obtain information to optimize applications.

The RTOS trace feature is available for FreeRTOS applications on ICE targets with a netlist supporting bus mode and quick access. For target platforms using local memory, the bus interface must also support slave port so that the RTOS trace feature can function properly.

---

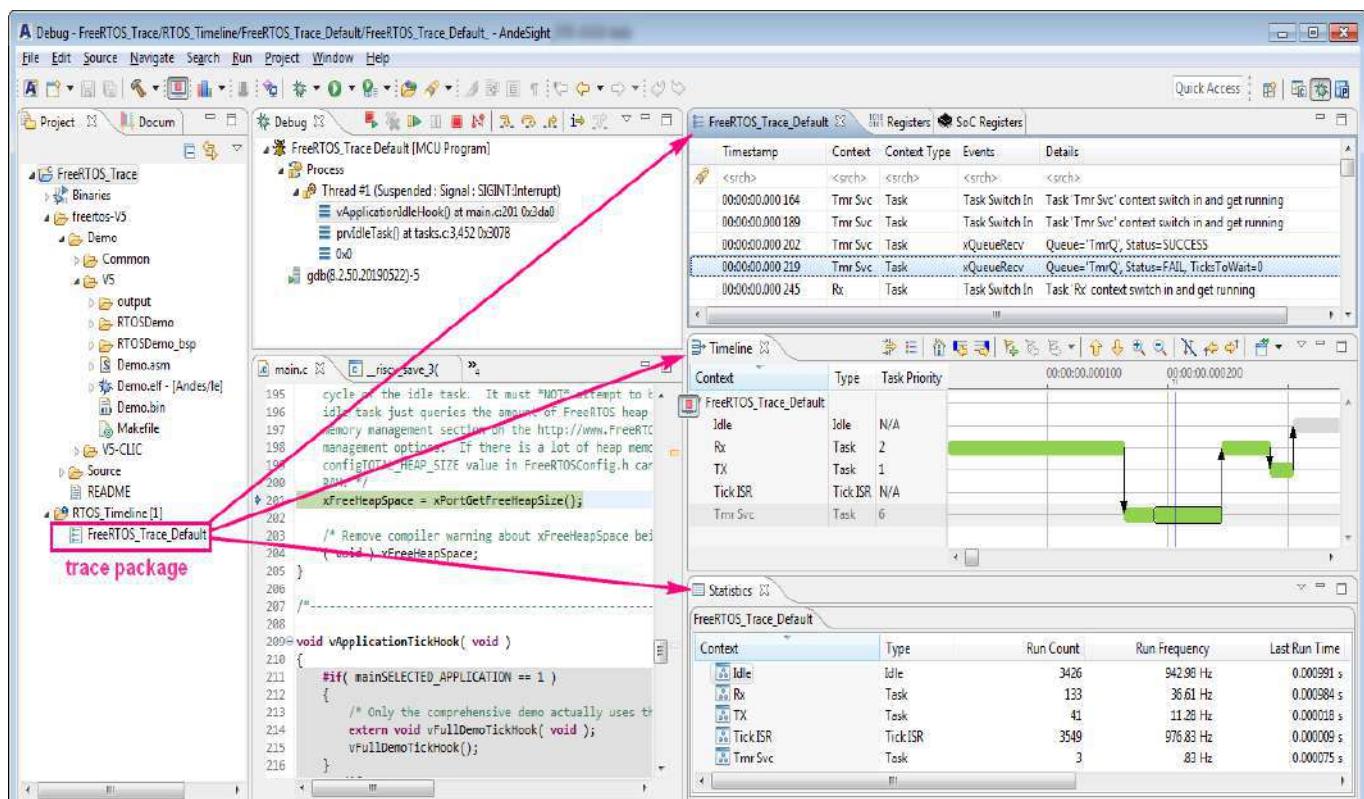
### NOTE

The RTOS trace feature in AndeSight v5.0 beta release is currently not supported on targets enabled with cache.

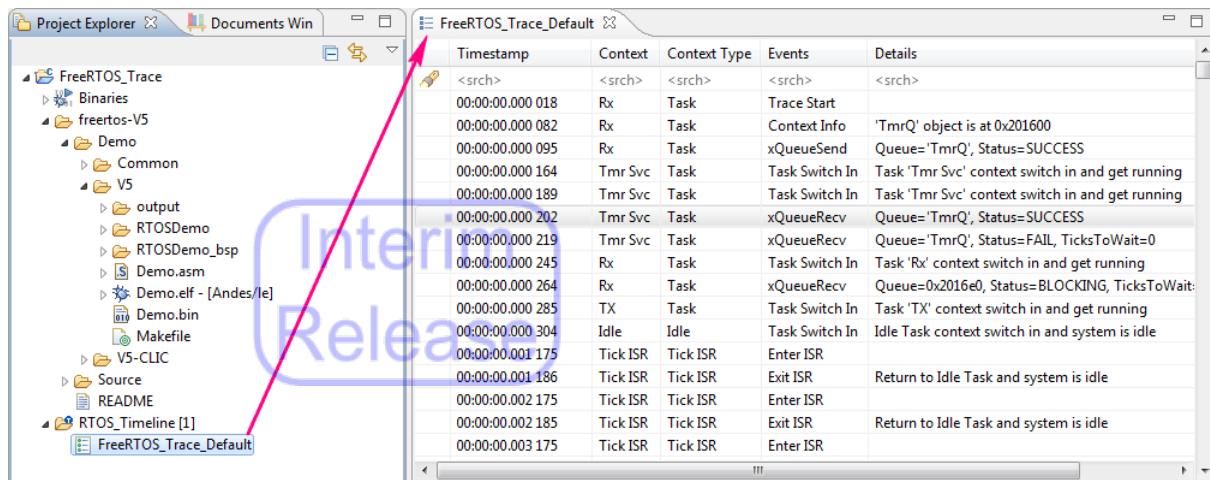
---

### 2.9.1. RTOS trace views

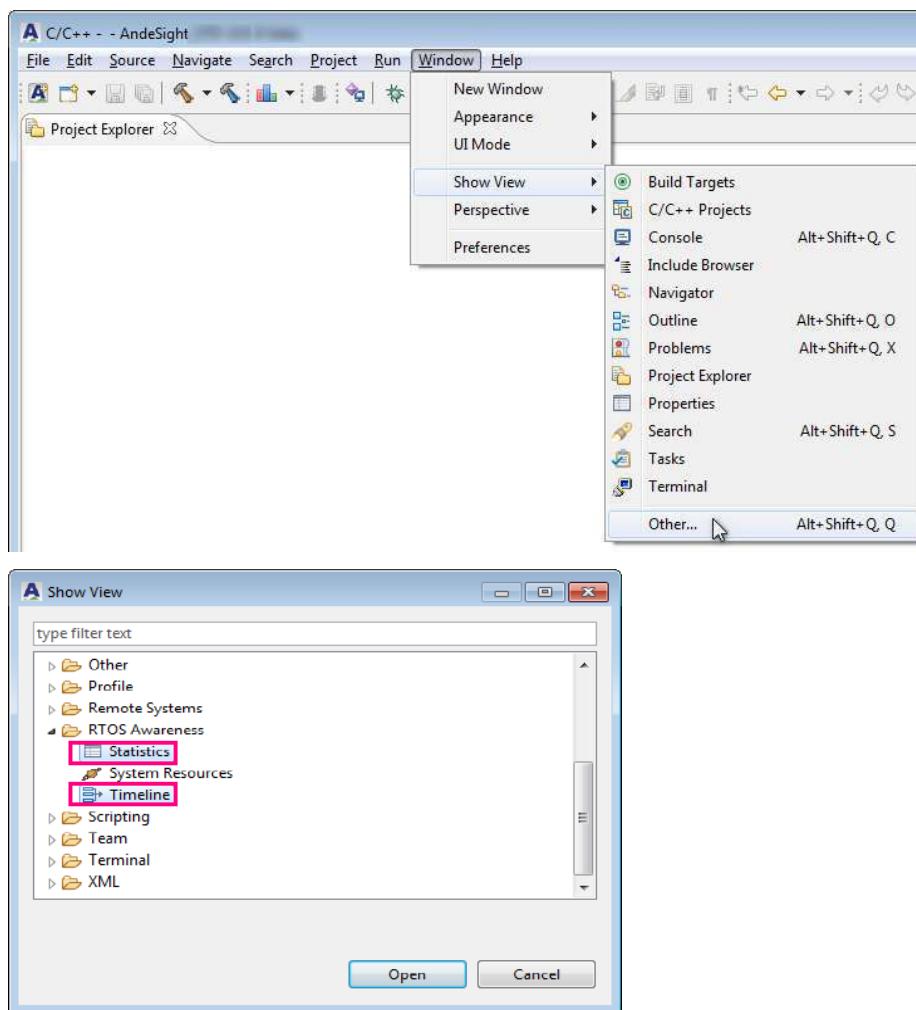
The data captured for a FreeRTOS application in a trace session is saved to a package file named as the applied launch configuration under **PROJECT\FreeRTOS\_Timeline**. When the trace package file is generated upon program suspension/termination or the end of a trace session, the **Event**, **Timeline** and **Statistics** views are invoked automatically to render and visualize trace data. These RTOS trace views serve as the interfaces to monitor and investigate program execution. If there are updates on the trace data, the three views will be refreshed synchronously to present the latest information.



The **Event** view, also shown in the name of the applied launch configuration, lists all the events recorded for the trace session. It can be opened manually by double-clicking the package file in the **Project Explorer**.

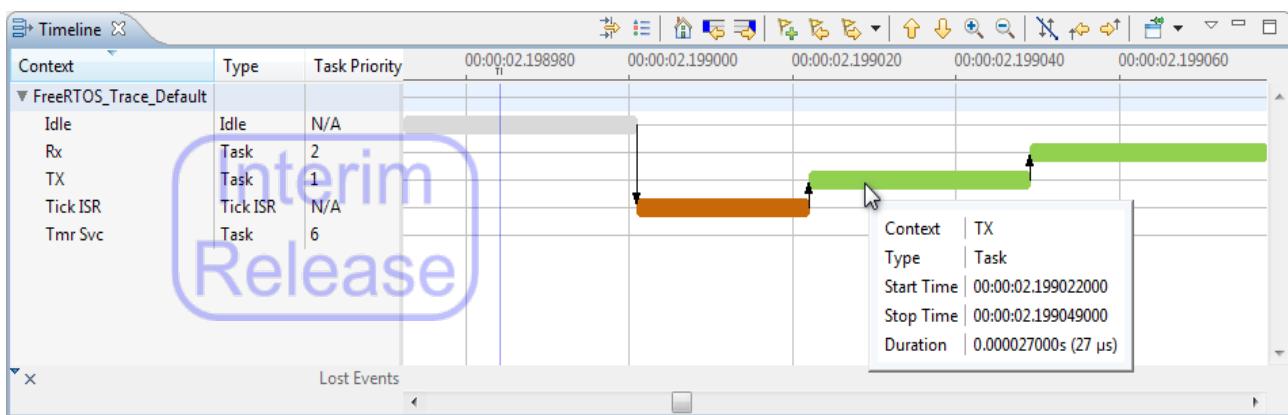


The **Timeline** view visualizes the recorded runtime behavior in a timeline and the **Statistics** view provides statistical information of the trace data. The two views can be invoked manually by selecting “Window > Show View > Other...” from the AndeSight main menu and then selecting “RTOS Awareness > [Statistics|Timeline]” in the **Show View** dialog.



The information contained herein is the exclusive property of Andes Technology Co. and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of Andes Technology Corporation.

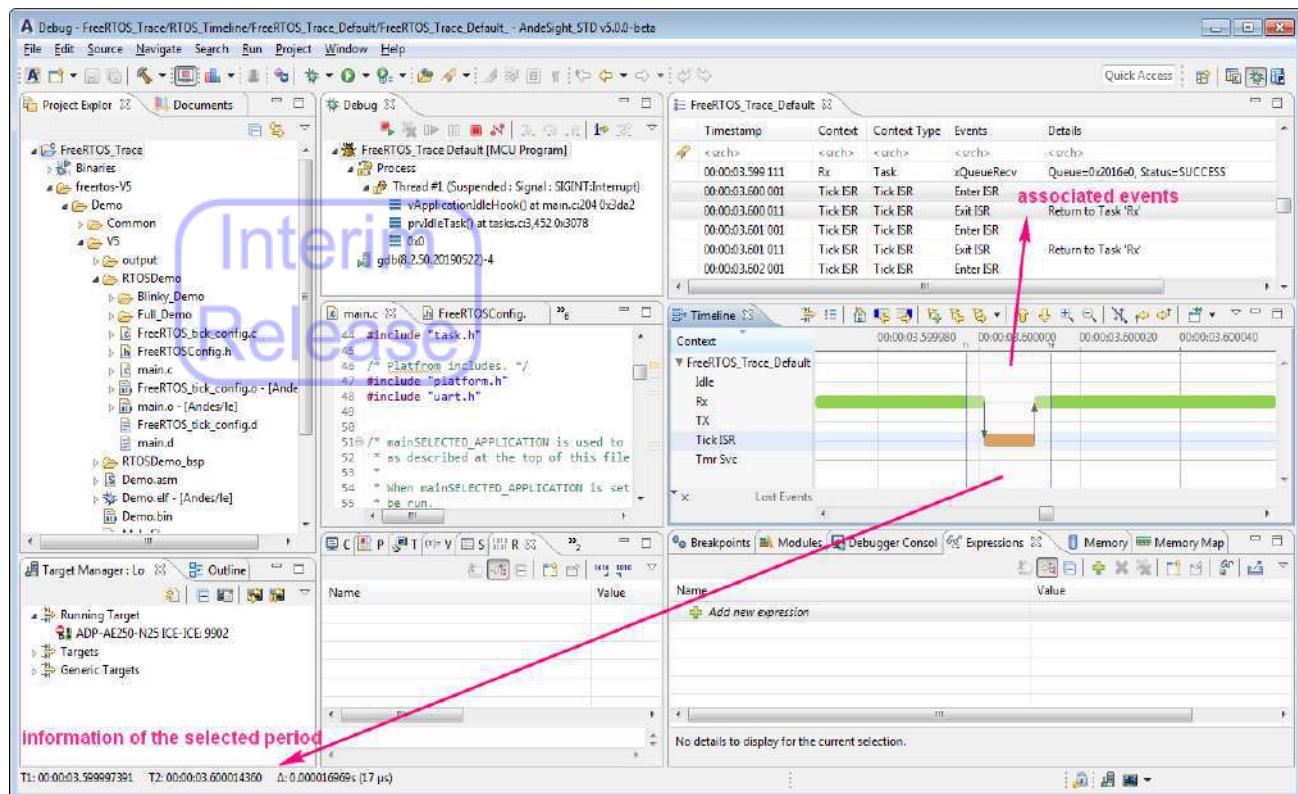
### 2.9.1.1 Timeline view



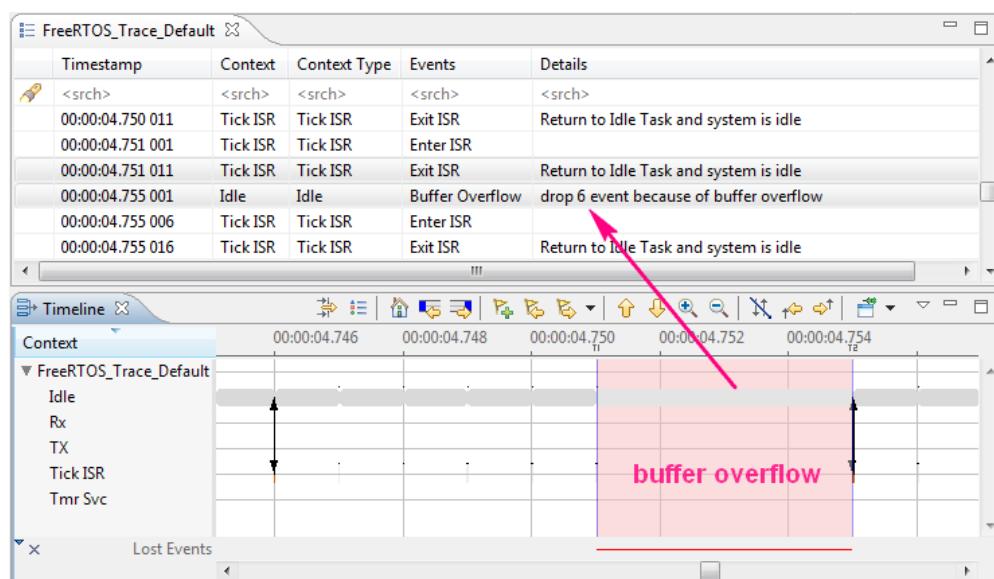
The **Timeline** view illustrates system behaviors for a RTOS application by depicting the sequence of contexts and CPU transitions in a timeline. Each row in the view stands for a specific context in the trace session. The occurrence of a context is denoted by a colored bar and the CPU transition upon a context switch is default represented as an arrow. Whenever you hover the mouse over a colored bar or an arrow on the timeline, a tooltip reveals to provide detailed information of the pointed context or CPU transitions.

On the left of this view is a context information table that gives the name, type and task priority of each context used in the application. The context rows in the **Timeline** view can be reordered by clicking on column headers in this table.

The T1 vertical line in this view indicates the time that's currently selected. To specify a time point or the start of a time span, left-click the specific time on the timeline. To specify the ending time of a time span, drag the cursor to the desired position and left-click again. The ending time is then denoted by the T2 vertical line and the interval between T1 and T2 lines is shaded to denote the selected time range. The details of the selected range, including the start time, ending time and the duration, is shown at the bottom of the AndeSight IDE. In addition, the event(s) associated with the selected time or range will be highlighted in shaded background in the **Event** view.

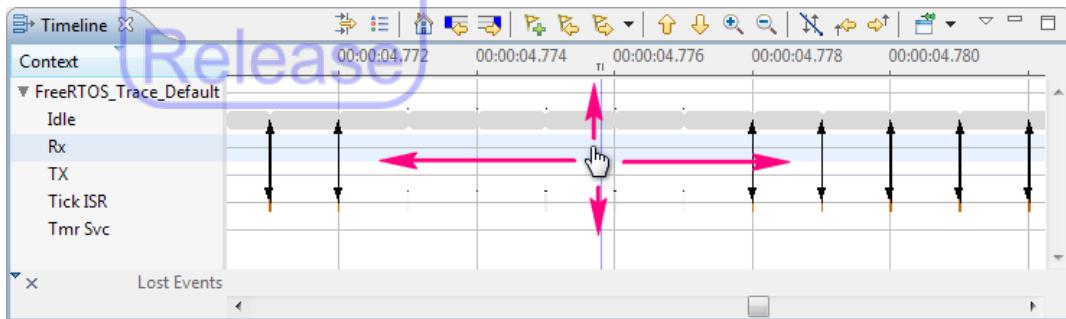


If buffer overflow occurs during the trace session, the RTOS tracer will continue recording but drop events until there is sufficient buffer space. In that case, the **Timeline** view will highlight affected time segments in a colored background to signal that some events are lost there for being dropped by the RTOS tracer. By selecting one of the segments on the timeline, you will be able to examine how many events are dropped within the period from the **Event** view.



The display of the timeline graph and the navigation in this view can be manipulated using toolbar buttons or mouse operations. For example, you can use the mouse and keyboard to perform the following functions:

- Panning: Press the Ctrl key and the left mouse button on the mouse to turn the cursor into a hand icon and hold it to scroll around.



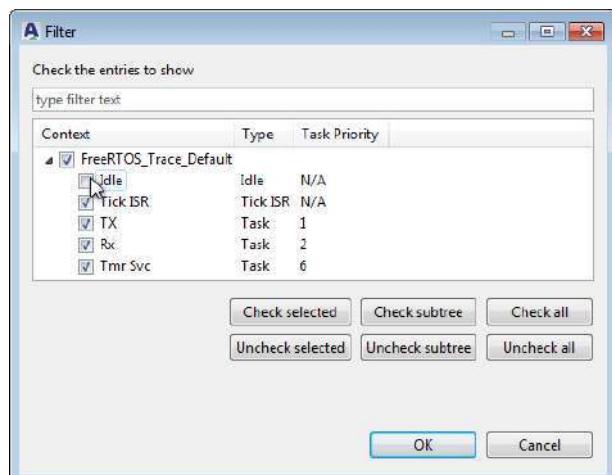
- Zooming into a range: Press the right mouse button and drag over the interested region.
- Zooming in and out horizontally: Press the Ctrl key and roll the mouse wheel up and down
- Zooming in and out vertically: Press the Shift and Ctrl keys and roll the mouse wheel up and down
- Reset the timeline graph to a full range: Double-click on the time ruler in the view

### Toolbar for the Timeline view

#### Show view filters

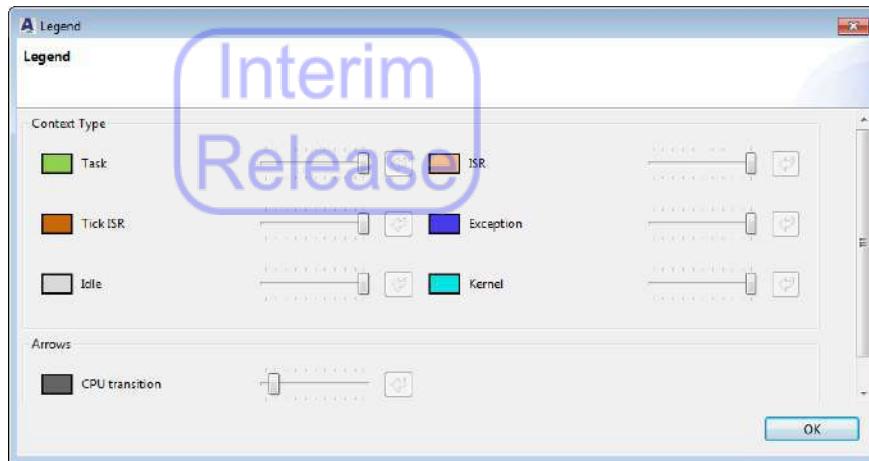


Click on this button to invoke the **Filter** dialog and specify what context to show or hide on the timeline.



Show legends

Click on this button to invoke the **Legend** dialog and define the color and width for the graphical representation of each context and CPU transition.

Reset the time scale to default

Click on this button to reset the timeline graph to the full range.

Select previous/next state change

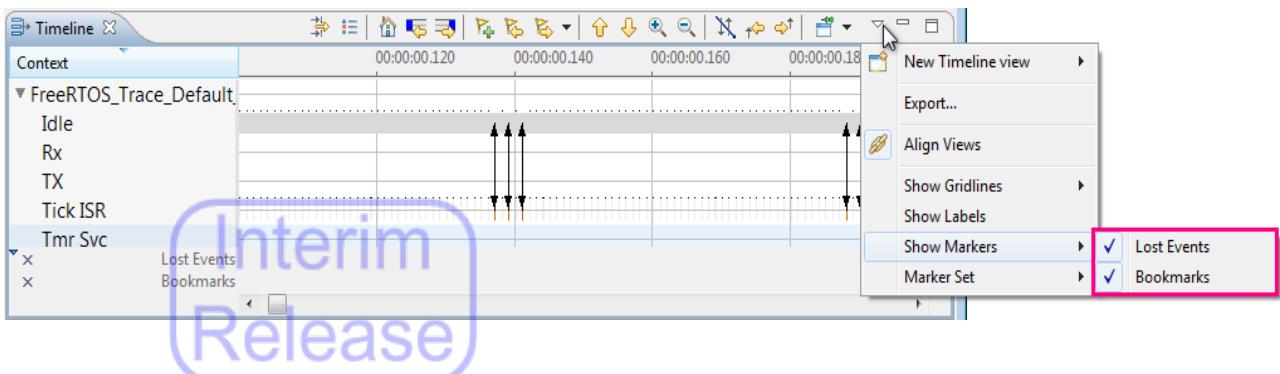
Select a context row in this view and click on this button to jump to its last/next switch.

Add/Delete bookmark

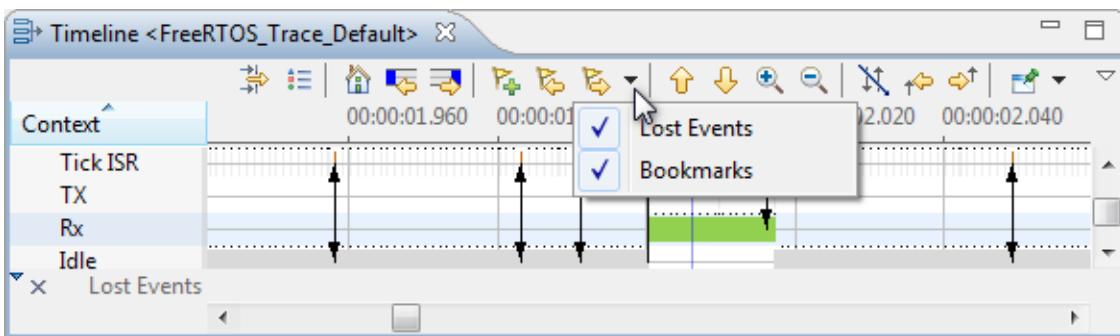
Specify a time point or range on the timeline graph and click on the button to set it as a bookmark. To clear a bookmarked time point/range on the timeline, select it in the view and click on the button .

Previous/Next marker

A marker on the timeline can be a bookmark you've set and/or a period with lost events. To specify whether to show lost events or bookmarks as markers in this view, click on the toolbar button (View Menu) and click on “Show Markers” to make a selection in the pull-down menu.



To locate a marker on the timeline, first click on the drop-down arrow  next to the button  (Next Marker) to specify the marker type(s) (i.e., lost events and/or bookmarks) for navigation and then click on  or  to jump to the previous or next marker on the timeline.



### Previous/Next element



Click on the button to select the previous/next context row.

### Zoom in/out



Click on the button to zoom into/out from the timeline graph.

### Hide arrows



Click on the button to hide the arrows that indicate CPU transitions on the timeline.

### Follow CPU backward/forward

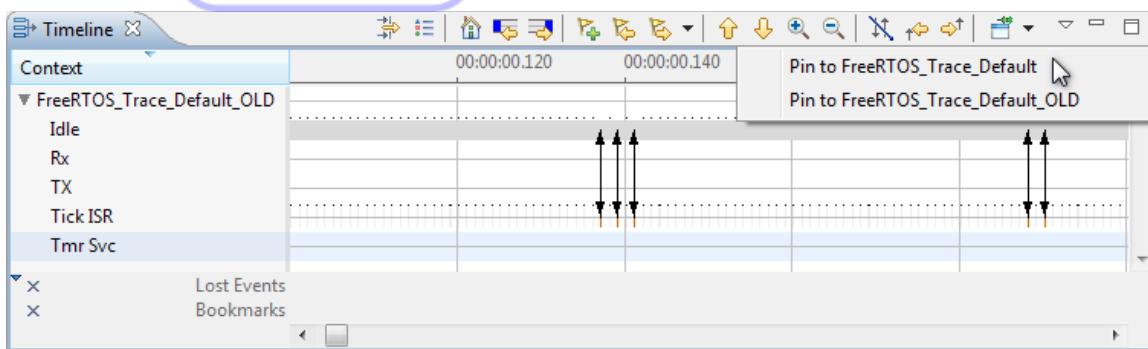


Click on the button to jump to the last/next time when the CPU is assigned to a new context.

### Pin/Unpin view



Click on the button  (Pin View) to pin the **Timeline** view to the trace data when the view is pinned. For a pinned **Timeline** view, clicking on the button  (Unpin View) enables it to refresh and render the timeline graph for the current trace session. To specify a trace session to pin the **Timeline** view to, click on the drop down arrow  next to the Pin/Unpin button and make a selection.



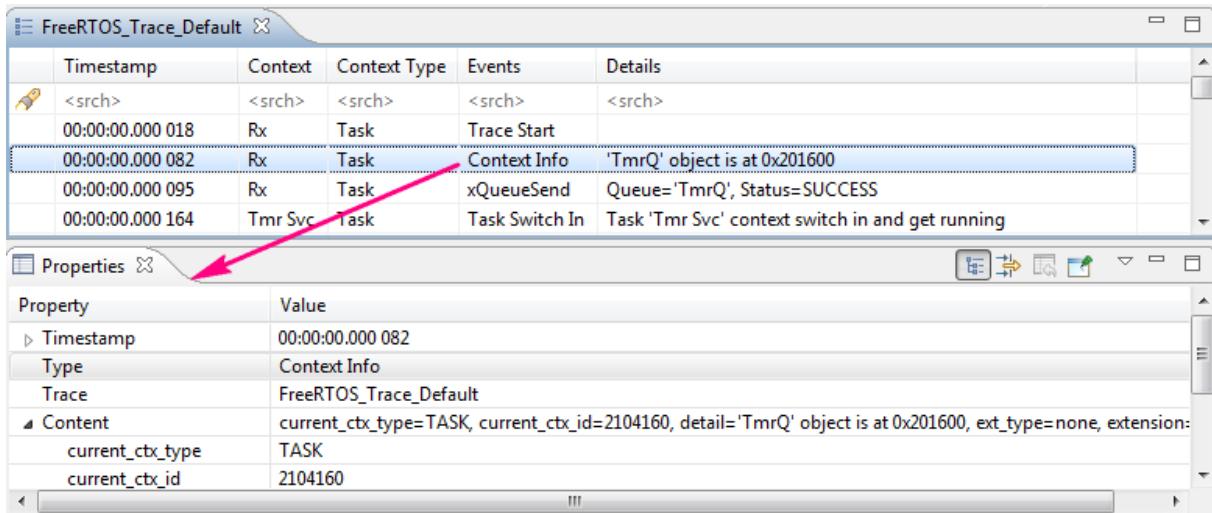
### 2.9.1.2 Event view

Interim  
Release

Timestamp	Context	Context Type	Events	Details
<srch>	<srch>	<srch>	<srch>	<srch>
00:00:00.000 018	Rx	Task	Trace Start	
00:00:00.000 082	Rx	Task	Context Info	'TmrQ' object is at 0x201600
00:00:00.000 095	Rx	Task	xQueueSend	Queue='TmrQ', Status=SUCCESS
00:00:00.000 164	Tmr Svc	Task	Task Switch In	Task 'Tmr Svc' context switch in and get running
00:00:00.000 189	Tmr Svc	Task	Task Switch In	Task 'Tmr Svc' context switch in and get running
00:00:00.000 202	Tmr Svc	Task	xQueueRecv	Queue='TmrQ', Status=SUCCESS
00:00:00.000 219	Tmr Svc	Task	xQueueRecv	Queue='TmrQ', Status=FAIL, TicksToWait=0
00:00:00.000 245	Rx	Task	Task Switch In	Task 'Rx' context switch in and get running
00:00:00.000 264	Rx	Task	xQueueRecv	Queue=0x2016e0, Status=BLOCKING, TicksToWait=4294967295
00:00:00.000 285	TX	Task	Task Switch In	Task 'TX' context switch in and get running

The **Event** view, shown in the name of the applied launch configuration (e.g., “FreeRTOS\_Trace\_Default” in the above figure), displays all the collected events from a trace session in a tabular format and indicates the event name, timestamp, context, context type and description for each event occurrence within your FreeRTOS application.

In contrast to the **Timeline** view that provides a high-level perspective of system activities in terms of contexts, this view gives you an in-depth understanding of events taken place in the contexts. For further details of a selected event, you may refer to the **Properties** view.

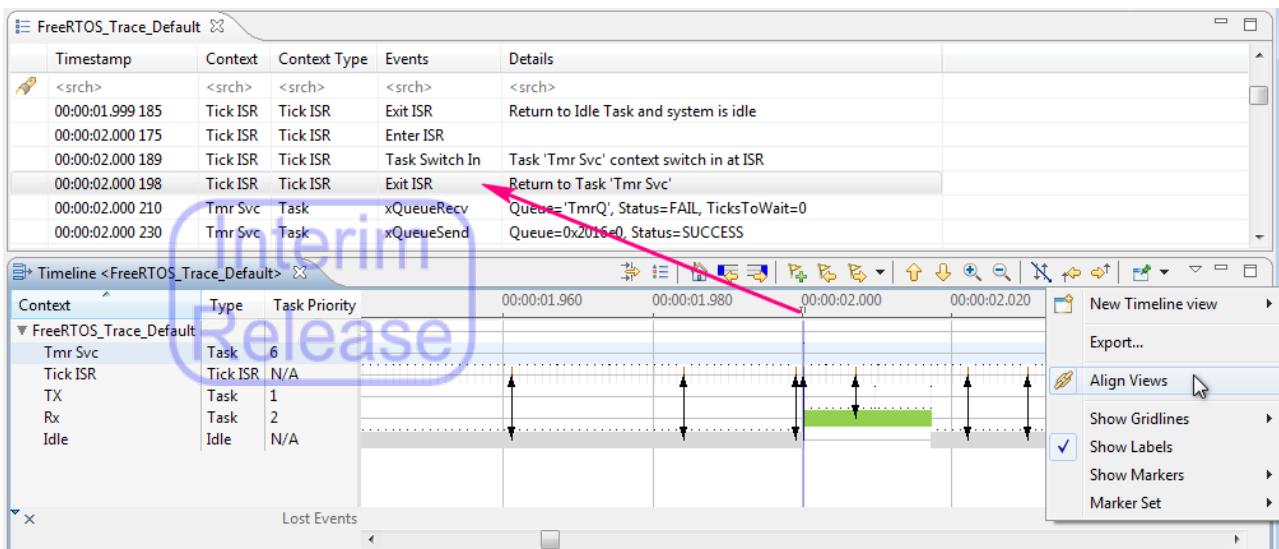


Timestamp	Context	Context Type	Events	Details
<srch>	<srch>	<srch>	<srch>	<srch>
00:00:00.000 018	Rx	Task	Trace Start	
00:00:00.000 082	Rx	Task	Context Info	'TmrQ' object is at 0x201600
00:00:00.000 095	Rx	Task	xQueueSend	Queue='TmrQ', Status=SUCCESS
00:00:00.000 164	Tmr Svc	Task	Task Switch In	Task 'Tmr Svc' context switch in and get running

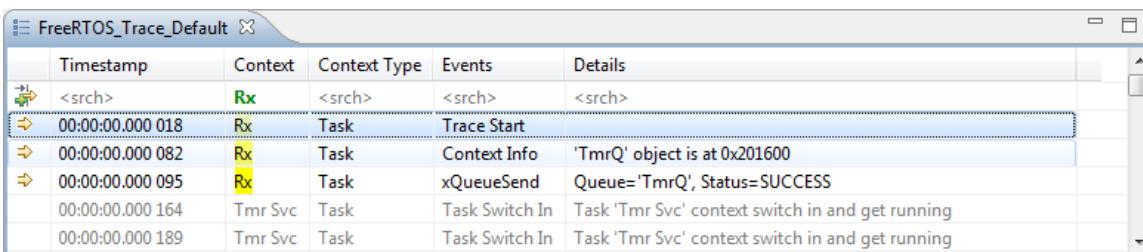
  

Properties	
Property	Value
Timestamp	00:00:00.082
Type	Context Info
Trace	FreeRTOS_Trace_Default
Content	current_ctx_type=TASK, current_ctx_id=2104160, detail='TmrQ' object is at 0x201600, ext_type=none, extension:
current_cbt_type	TASK
current_cbt_id	2104160

The navigation between the **Timeline** view and **Event** view can be synchronized. Just click on  (View menu) on the toolbar of the **Timeline** view and select “Align Views” on the pull-down menu.



The **Event** view comes with the search and filtering functionality. Specify your search conditions with a regular expression or text string in column cell(s) of the first row and press the Enter key to apply the condition(s). The view will be refreshed to mark all matching event with an arrow ➤ in their left margin and the first matching event will be selected. To jump to the next matching event, press the Enter key again.



The screenshot shows the AndeSight IDE interface with the 'Event' view. A search condition 'Rx' is applied to the first row of the table. The table displays the following events:

Timestamp	Context	Context Type	Events	Details
00:00:00.000 018	Rx	Task	Trace Start	
00:00:00.000 082	Rx	Task	Context Info	'TmrQ' object is at 0x201600
00:00:00.000 095	Rx	Task	xQueueSend	Queue='TmrQ', Status=SUCCESS
00:00:00.000 164	Tmr Svc	Task	Task Switch In	Task 'Tmr Svc' context switch in and get running
00:00:00.000 189	Tmr Svc	Task	Task Switch In	Task 'Tmr Svc' context switch in and get running

To set the search condition(s) as a filter, click on  (Add as Filter) in the first row or press Ctrl + Enter. This will lead the view to display only the matching events and provide the number of matching events in the second row. The filtering conditions are shown in the tags on the top of the table. To remove a filtering condition, click on ✖ of the corresponding tag. To remove all filters at once, right-click any matching event and select "Clear Filters" in the pull-down menu.

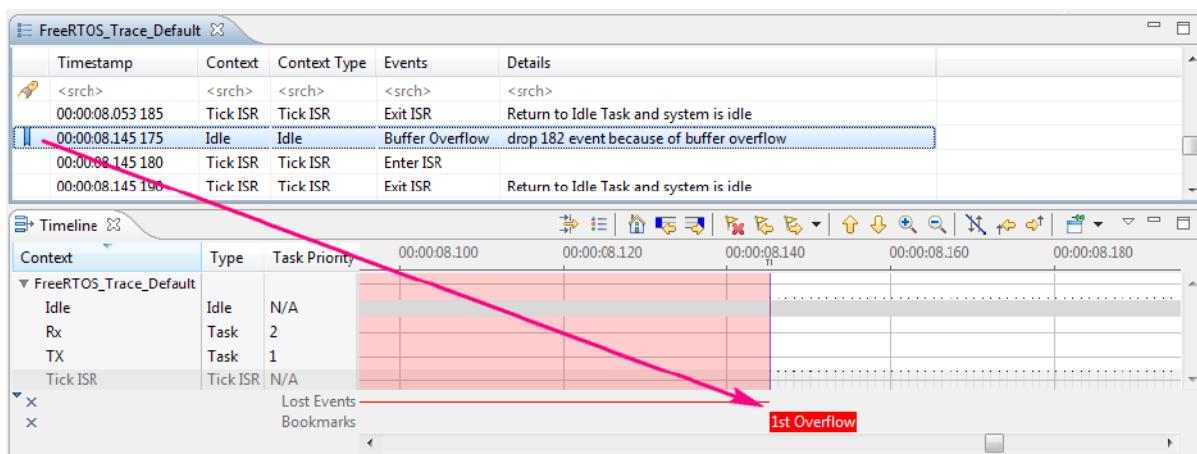
*(Interim Release)*

Context/current_ctx_id matches "Rx"				
Timestamp	Context	Context Type	Events	Details
<srch>	<srch>	<srch>	<srch>	<srch>
128/14730				
00:00:00.000 018	Rx	Task	Trace Start	
00:00:00.000 082	Rx	Task	Context Info	'TmrQ' object is at 0x201600
00:00:00.000 095	Rx	Task	xQueueSend	Queue='TmrQ', Status=SUCCESS
00:00:00.000 245	Rx	Task	Task Switch In	Task 'Rx' context switch in and get running

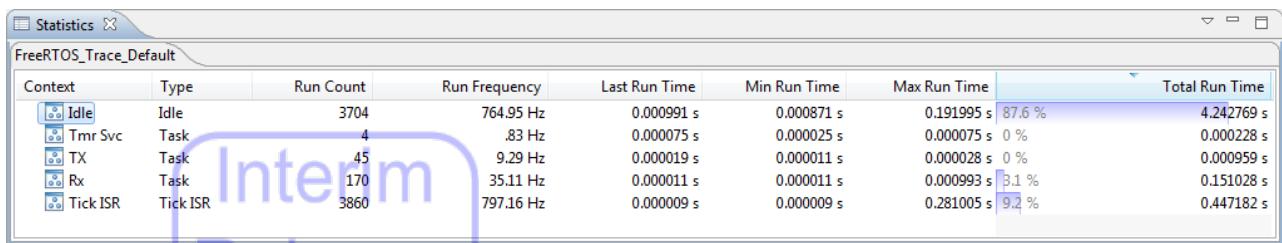
You can also create a bookmark for an event of interest in this view. Just double-click on the left margin of an event and enter bookmark descriptions in the invoked **Add Bookmark** dialog.

Timestamp	Context	Context Type	Events	Details
<srch>	<srch>	<srch>	<srch>	<srch>
00:00:08.145 175	Idle	Idle	Buffer Overflow	drop 182 event because of buffer overflow
00:00:08.145 180	TICK ISR	TICK ISR	Enter ISR	Return to Idle Task and system is idle
00:00:08.145 185	Tick ISR	Tick ISR	Exit ISR	Return to Idle Task and system is idle
00:00:08.150 175	Tick ISR	Tick ISR	Enter ISR	Return to Idle Task and system is idle
00:00:08.150 180	Tick ISR	Tick ISR	Exit ISR	Return to Idle Task and system is idle
00:00:08.150 185	Tick ISR	Tick ISR	Enter ISR	Return to Idle Task and system is idle
00:00:08.150 187	Tick ISR	Tick ISR	Exit ISR	Return to Idle Task and system is idle
00:00:08.150 188	Tick ISR	Tick ISR	Enter ISR	Return to Idle Task and system is idle
00:00:08.150 189	Tick ISR	Tick ISR	Exit ISR	Return to Idle Task and system is idle

The bookmarked event in the **Event** view will be marked with the icon  on its left margin. It will also be added to the **Timeline** view right away, allowing you to investigate or compare interested events on a timeline.



### 2.9.1.3 Statistics view



Context	Type	Run Count	Run Frequency	Last Run Time	Min Run Time	Max Run Time	Total Run Time
Idle	Idle	3704	764.95 Hz	0.000991 s	0.000871 s	0.191995 s	4.242769 s
Tmr Svc	Task	4	.83 Hz	0.000075 s	0.000025 s	0.000075 s	0.000228 s
TX	Task	45	9.29 Hz	0.000019 s	0.000011 s	0.000028 s	0.000959 s
Rx	Task	170	35.11 Hz	0.000011 s	0.000011 s	0.000993 s	0.151028 s
Tick ISR	Tick ISR	3860	797.16 Hz	0.000009 s	0.000009 s	0.281005 s	0.447182 s

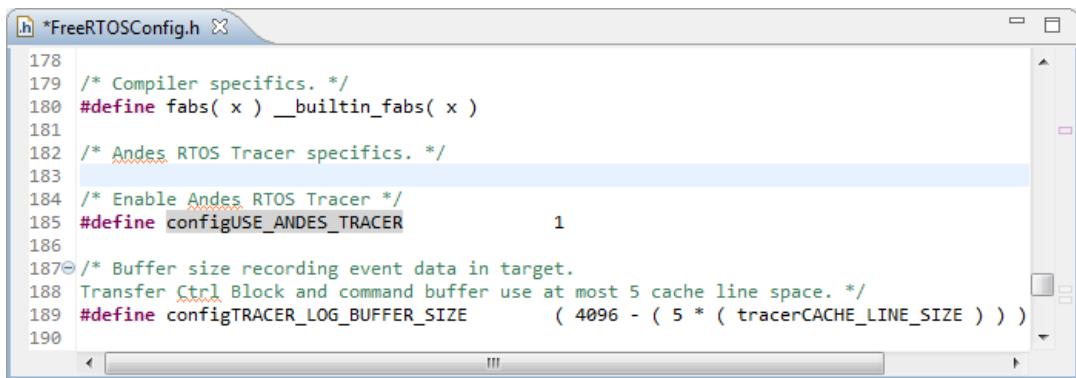
The **Statistics** view gives statistical information of contexts in the trace session. For each context of the FreeRTOS application in the recorded time range, the **Statistic** view shows its name and type, the number and frequency of its occurrence, its last/minimum/maximum/total duration in seconds, and the percentage of its total duration against the time measured. The information in this view can be sorted by clicking on the column headers.

## 2.9.2. Tracing an FreeRTOS application

The FreeRTOS Blinky application is used in this section to illustrate how to initiate a trace session for a RTOS application and examine its runtime behaviors in the AndeSight IDE.

**Step 1** Follow Section 2.4.12.1, Step 1 ~ Step 12 to create, configure and build a project for FreeRTOS Blinky application and create a MCU Program debug configuration for it. To trace the FreeRTOS application, two additional modifications need to be made on **FreeRTOSConfig.h** placed under **PROJECT\freertos-V5\Demo\V5\RTOSDemo**:

- Changing the value of **configTICK\_RATE\_HZ** to **100**
- Setting the value of **configUSE\_ANDES\_TRACER** to **1**, as shown below.



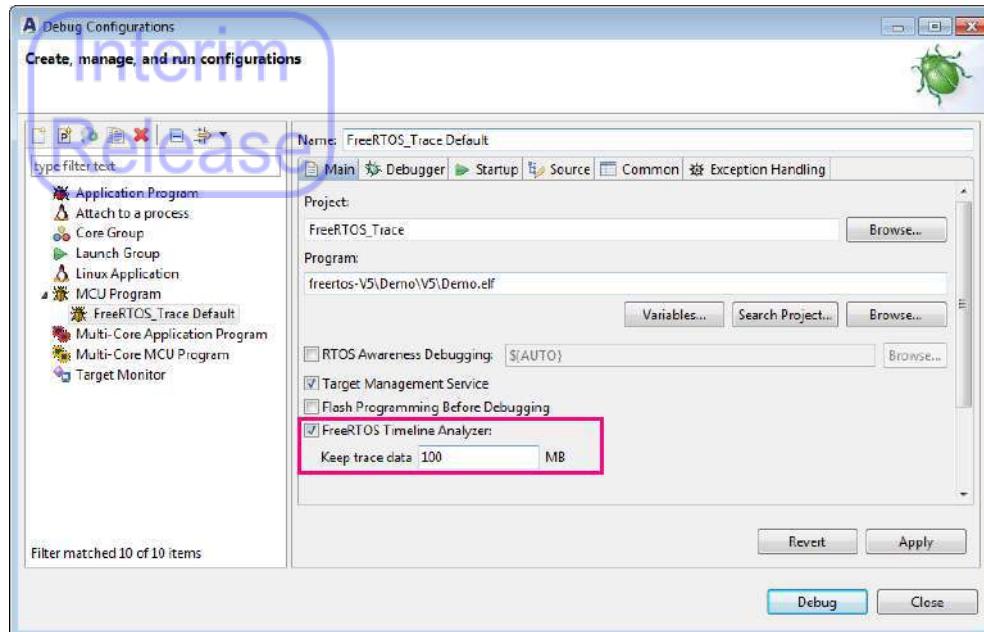
```
*FreeRTOSConfig.h
178
179 /* Compiler specifics. */
180 #define fabs( x ) __builtin_fabs( x )
181
182 /* Andes RTOS Tracer specifics. */
183
184 /* Enable Andes RTOS Tracer */
185 #define configUSE_ANDES_TRACER           1
186
187/* Buffer size recording event data in target.
188 Transfer Ctrl Block and command buffer use at most 5 cache line space.*/
189 #define configTRACER_LOG_BUFFER_SIZE    ( 4096 - ( 5 * ( tracerCACHE_LINE_SIZE ) ) )
190
```

Also note that the RTOS trace feature has the following target requirements and make sure the target configuration of this project satisfies these conditions:

- It is restricted to ICE targets with netlist supporting bus mode and quick access.
- For platforms using local memory, the bus interface must also support slaveport.
- Targets enabled with cache is currently not supported.

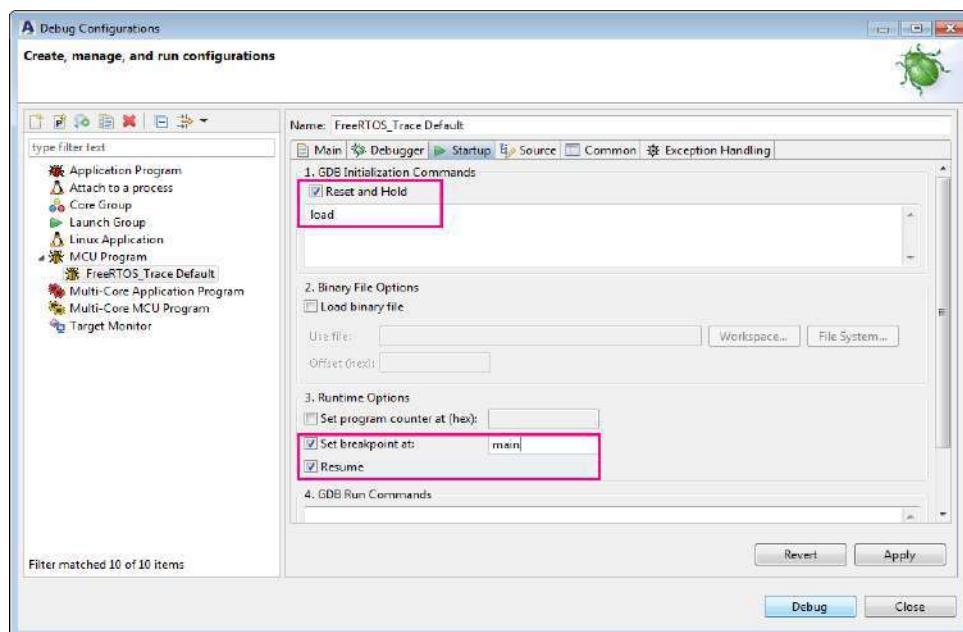
In this section, an example project “RTOS-trace” is created for the Blinky application and an ICE target of ADP-AE250-N25F is used.

**Step 2** In the main tab of the **MCU Program** debug configuration, select the “FreeRTOS Timeline Analyzer” option to enable the RTOS tracer and specify a trace data size no less than 100 MB.



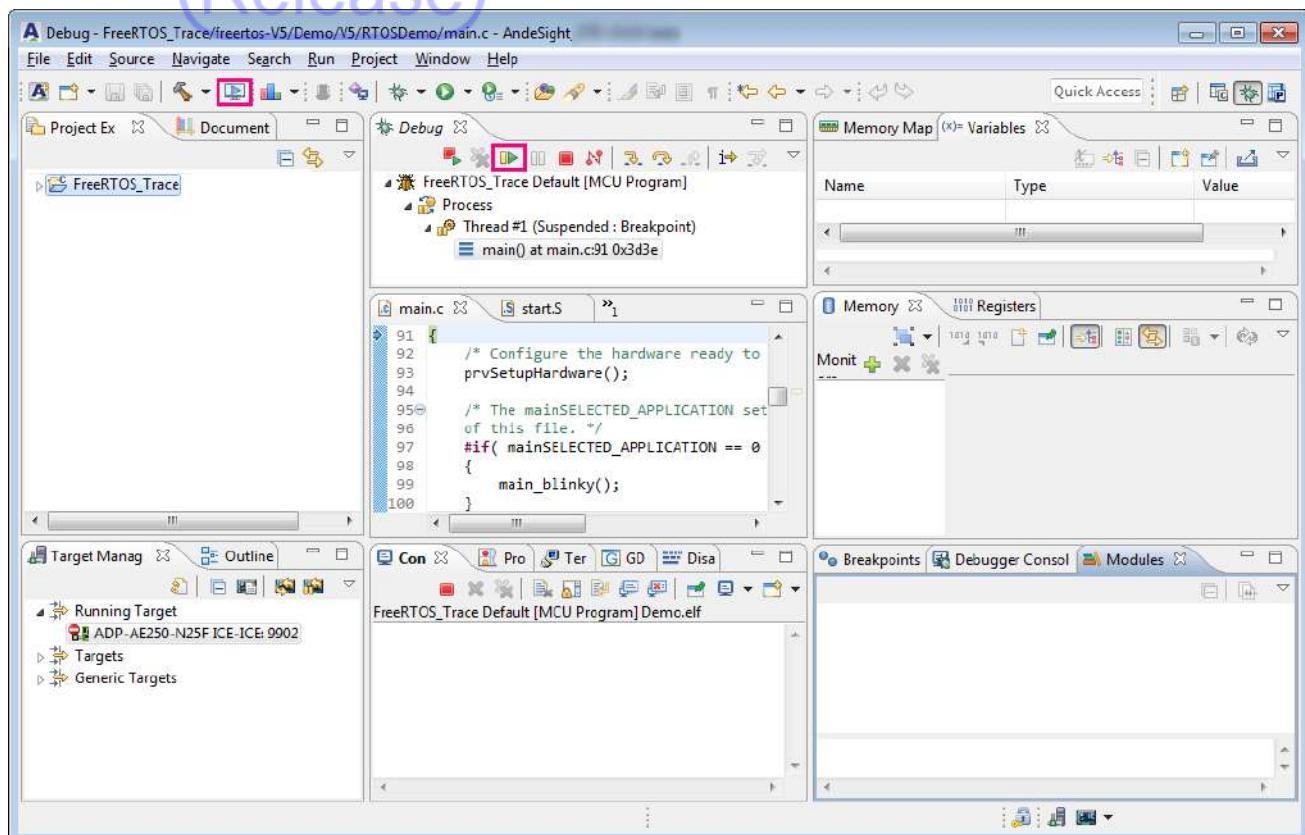
**Step 3** In the Startup tab, configure the settings as follows:

- In the GDB Initialization Commands section, check the “Reset and Hold” option and enter the command “`load`”.
- In the Runtime Options section, set a breakpoint at the main function and select the “Resume” option.

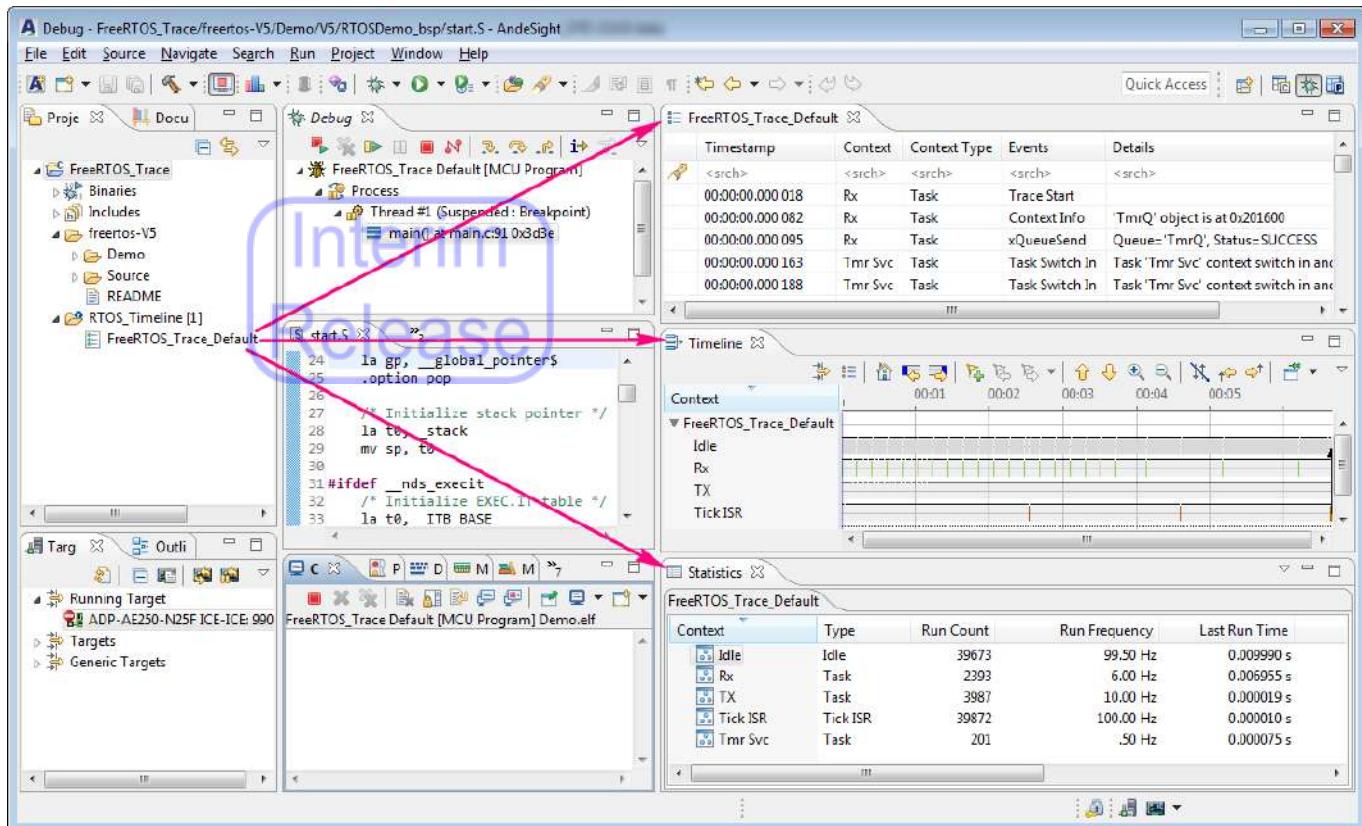


Then, click “Apply” and “Debug” to launch the debug session.

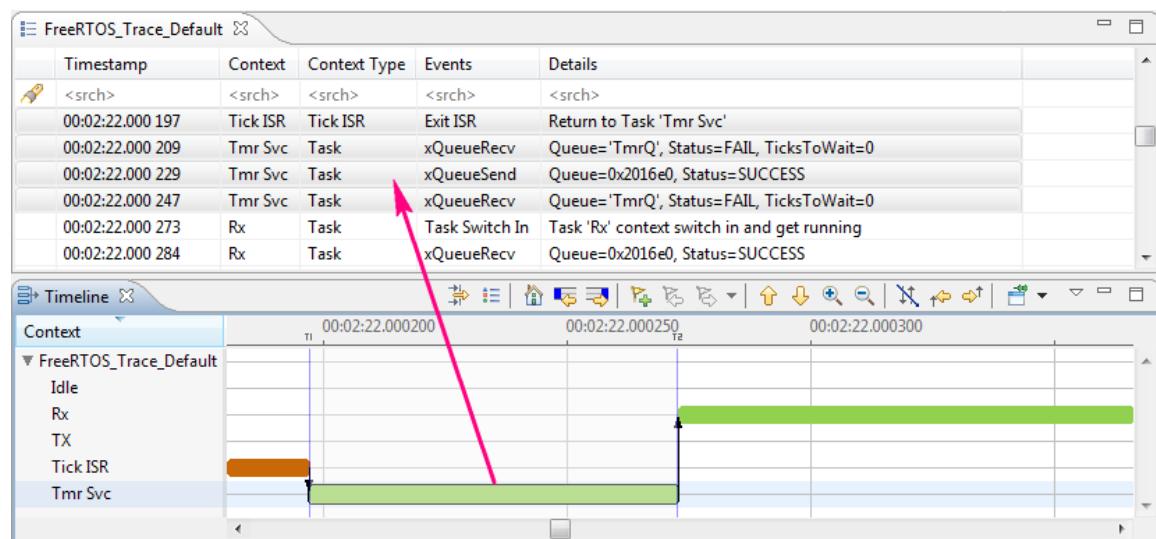
**Step 4** The **Debug** perspective is invoked automatically and the program suspends at the main function. Click on  (Start FreeRTOS trace) on the AndeSight toolbar to initiate the trace session and click on  (Resume) in the Debug view to continue the program execution.



**Step 5** The RTOS tracer starts to record system activities of the FreeRTOS application and save them to a package file named as the current launch configuration under **PROJECT\FreeRTOS\_Timeline**. Whenever the program is suspended/terminated or if the trace session is ended with the click of the toolbar button  (Stop FreeRTOS trace), the captured data in the package file will be rendered to the **Event**, **Timeline** and **Statistics** views. Note that the **Event** view is displayed in the name of the applied debug configuration.



Navigate in the **Timeline** view to inspect the sequence of contexts and their switches with respect to time. For an interested time point, period or context, select them on the timeline and examine their associated events in the **Event** view. The figure below shows that the selected context “TmrSvc” involves four events.



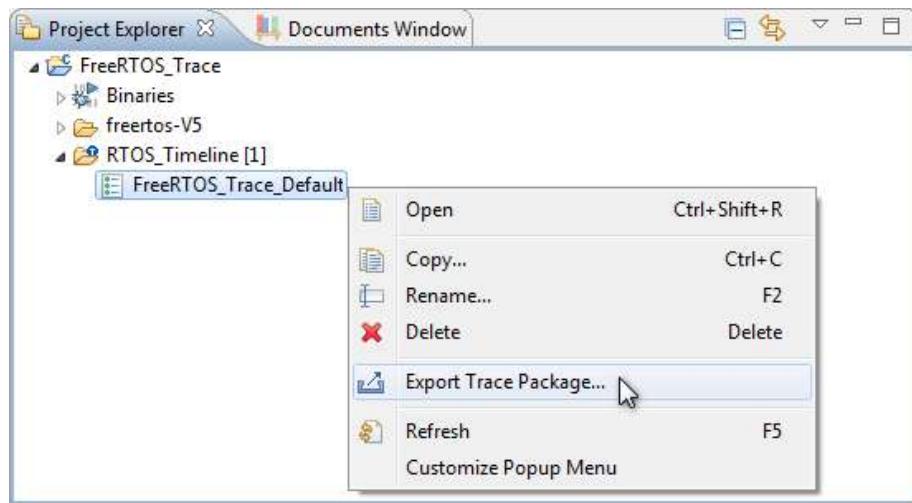
---

**NOTE**

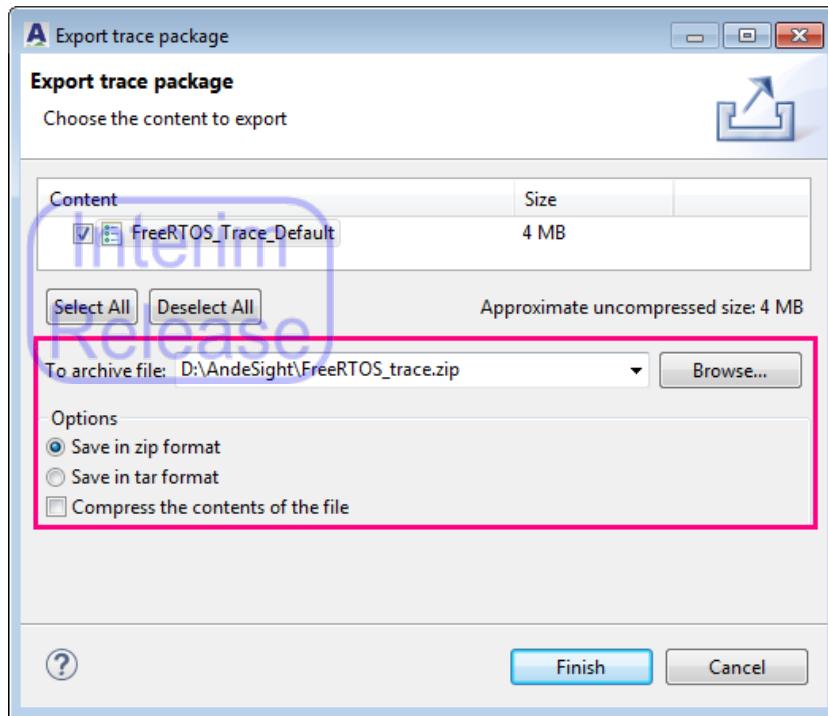
The following debug functions in the AndeSight IDE will be affected by the current RTOS tracer and should be avoided during a RTOS trace session:

- The toolbar button  (Auto refresh) in the **Memory view**, **Memory Browser view**, **SoC Registers view** and **Global Variables Live view**.
  - The toolbar button  (Switch source between BUS and CPU) in the **Memory view** and **Memory Browser view**.
- 

**Step 6** To export the trace data, right-click on the trace package file under **PROJECT\FreeRTOS\_Timeline** and select “Export Trace Package...” in the pull-down menu.

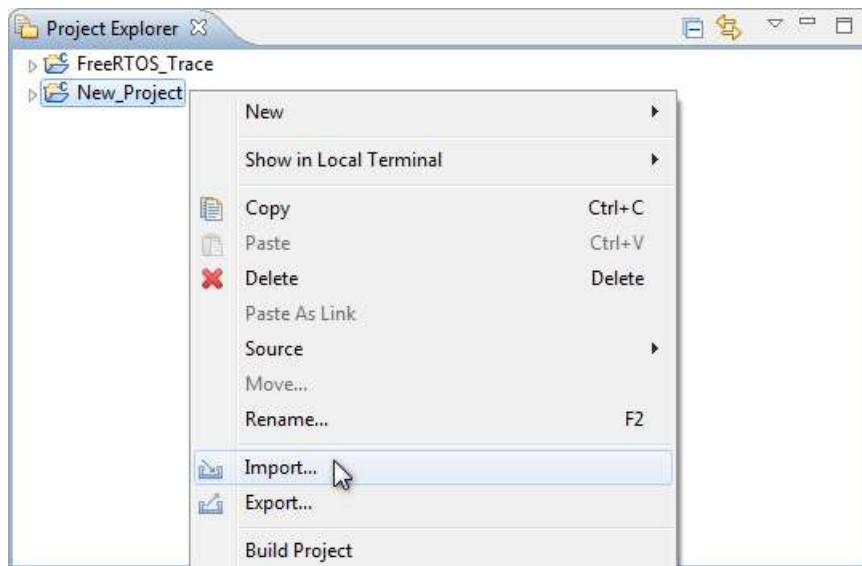


In the invoked dialog, specify the name, location and format for the exported file and then click “Finish” to complete the export process. In the figure below, the trace package “FreeRTOS\_Trace\_Default” is exported to **FreeRTOS\_trace.zip** under the AndeSight root directory.

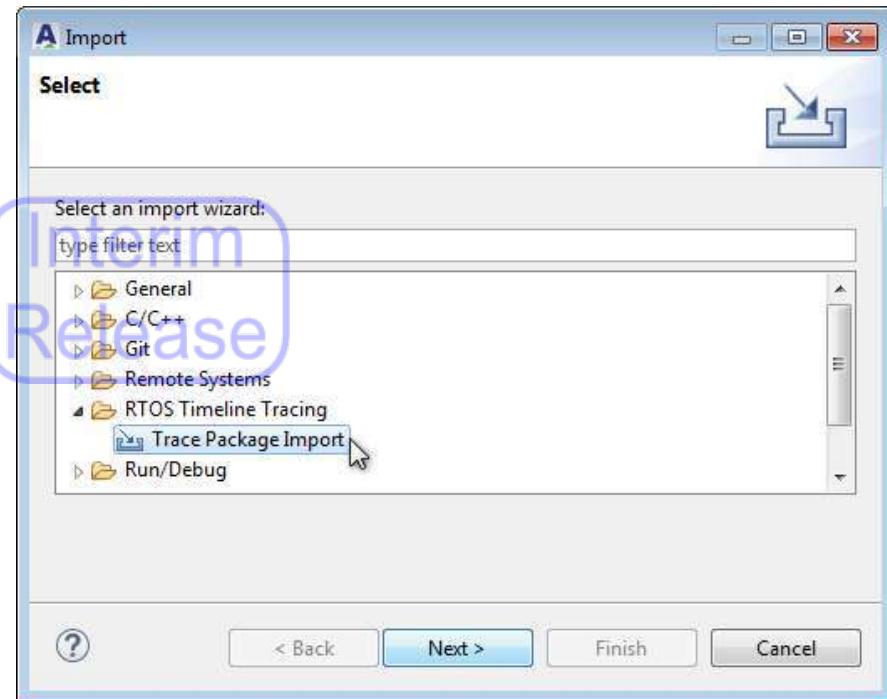


**Step 7** To import the exported trace package to another project, proceed as follows:

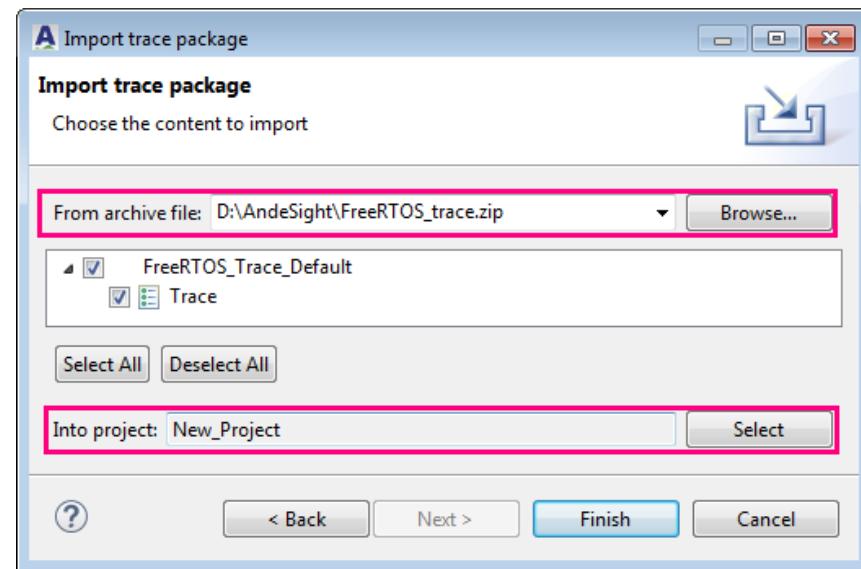
1. Right-click on the desired project (e.g. “New\_Project” in the figure below) and select “Import” from its pull-down menu.



2. In the **Import** dialog, select “RTOS Timeline Tracing > Trace Package Import” and click on “Next”.

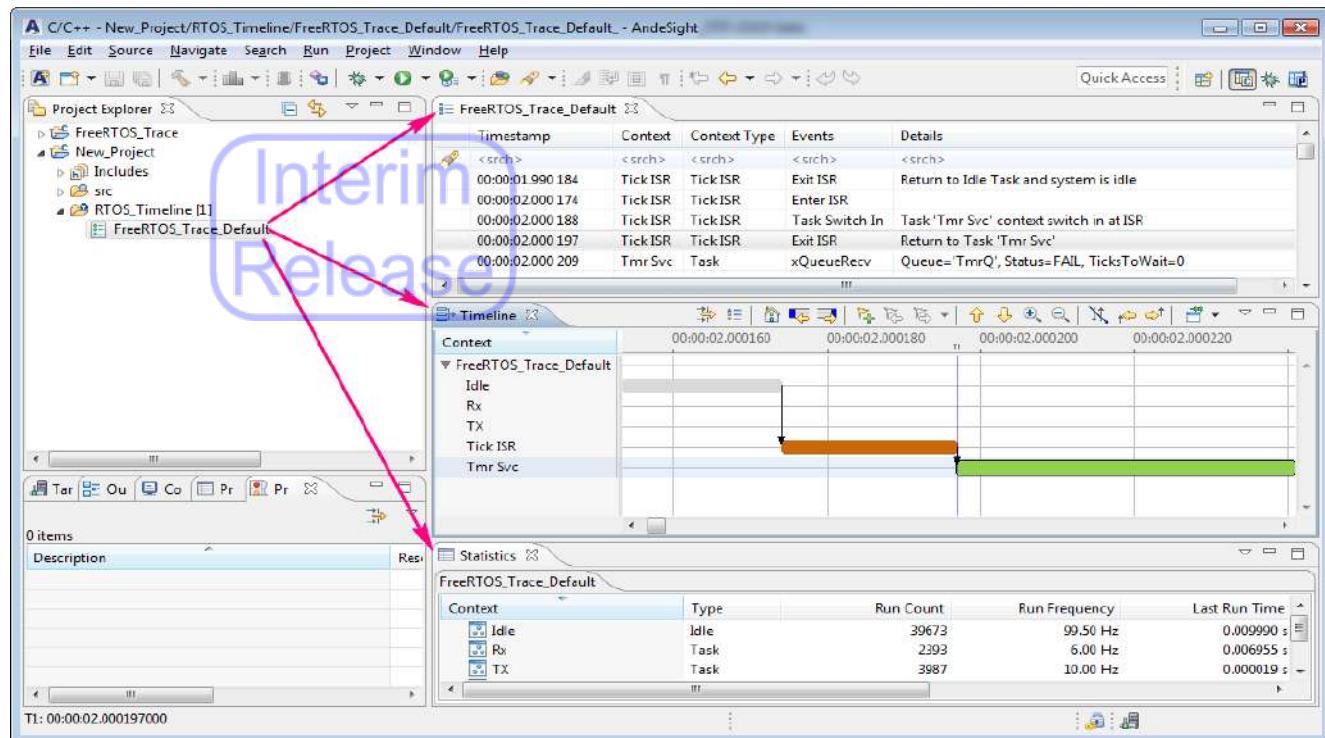


3. In the **Import trace package** dialog, click “Browse” to navigate for the exported trace file. In the “Into project” field, select the project for the file to be imported to. Then, click “Finish” to complete the import process.



The trace package file is exported to the designated project.  
Double-click on it to render trace data in the **Event, Timeline**

and **Statistics** view for further inspection.



## 3. Demo projects

AndeSight contains a number of demos specifically for target systems using V5 cores. The demos can be found under `ANDESIGHT_ROOT\demo\`. Make sure to select a demo from a directory corresponding to the platform and configuration of your target system. The following sections walk you through the demo projects and describe a set of AndeSight functions along the way.

(*Interim Release*)

### 3.1. JPEG decoder and MP3 demo

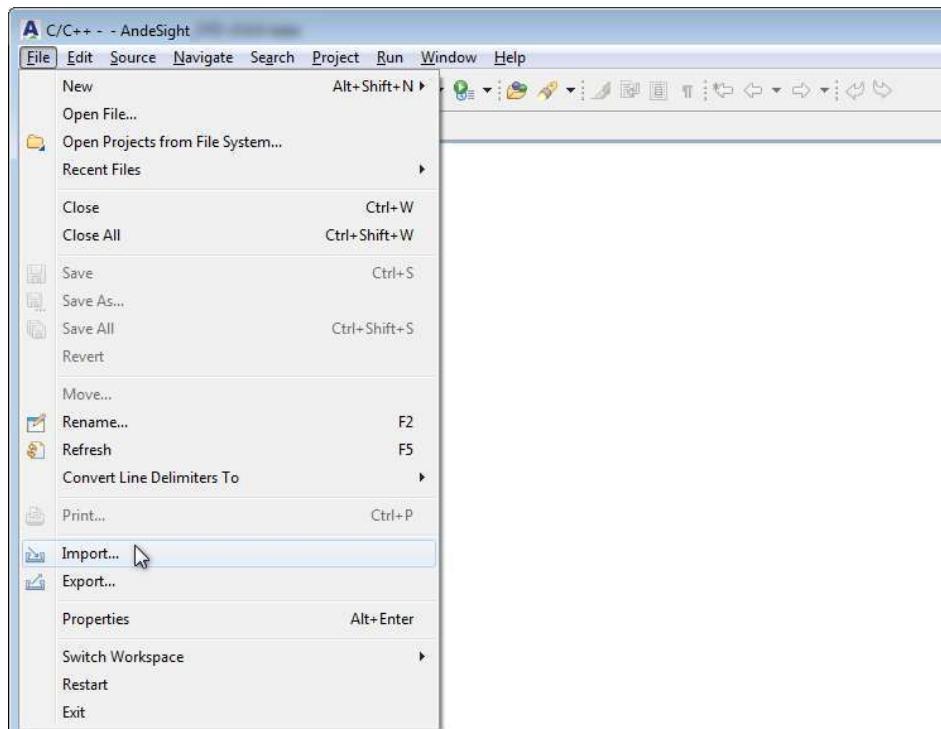
The JPEG decoder demonstrates the decoding and display of a series of jpg files. The MP3 demo demonstrates the playing of audio files in polling mode and interrupt mode. Both of these applications run only on AE350 targets.

The two demos can be executed in the same procedure. In the sections below, the JPEG project is adopted as an example for step-by-step instructions on how to run the two standalone demo programs. The operation focuses on the following:

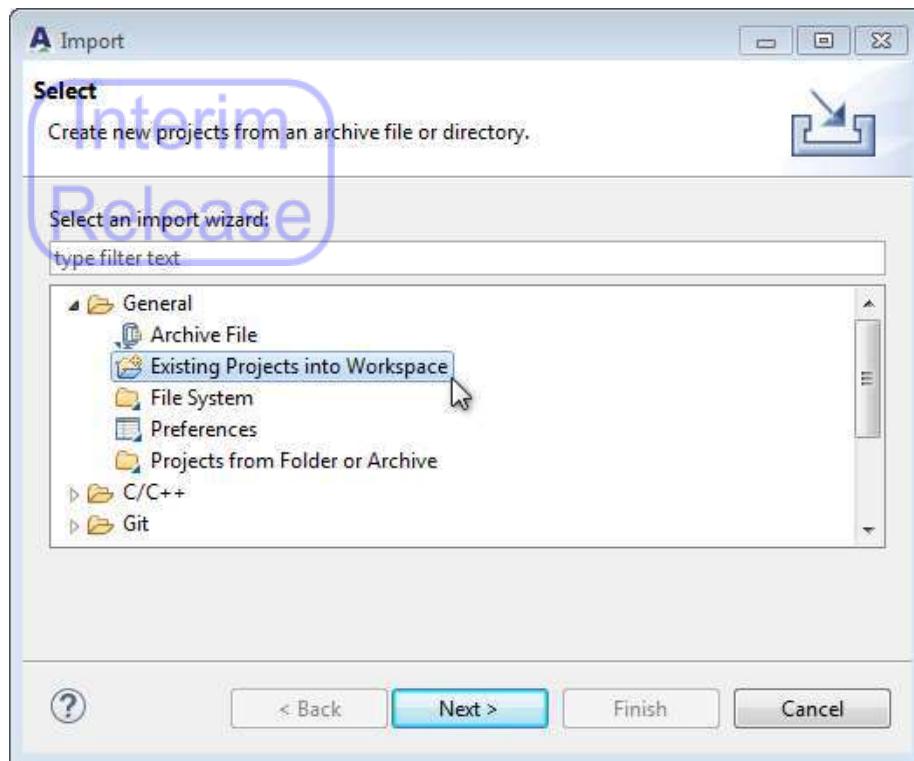
- How to import an existing project
- How to specify a build configuration for a project
- How to specify target configurations for a project
- How to build a project
- How to run a project

#### 3.1.1. Importing a demo project

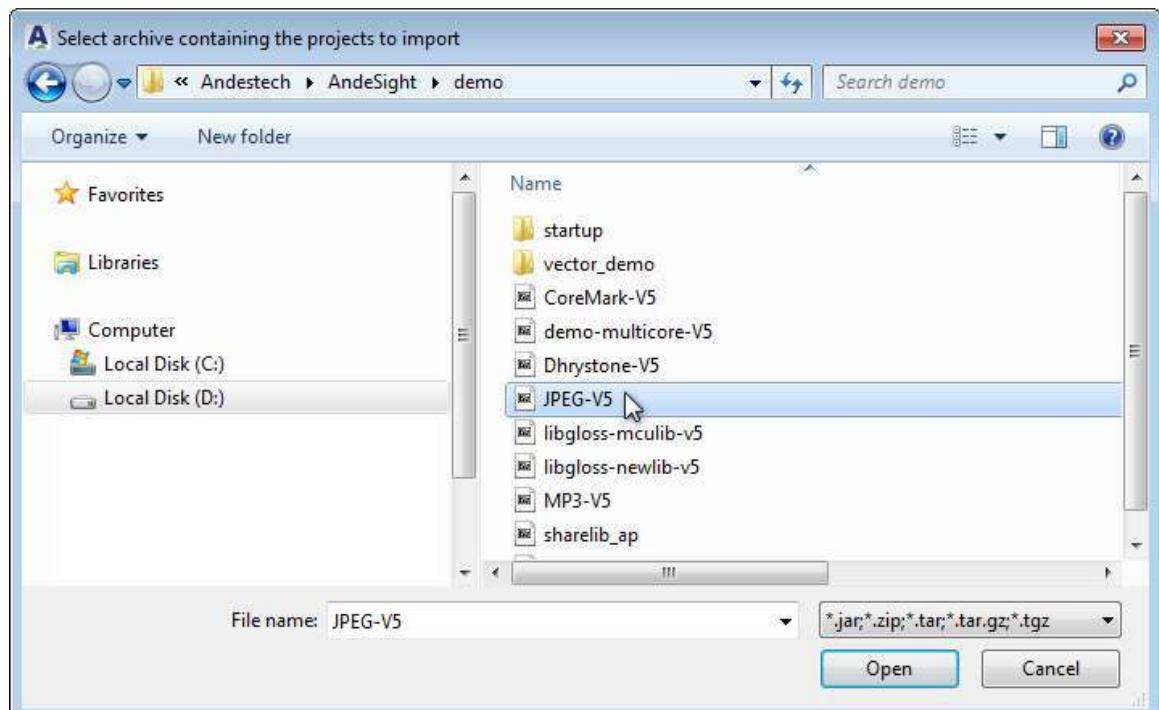
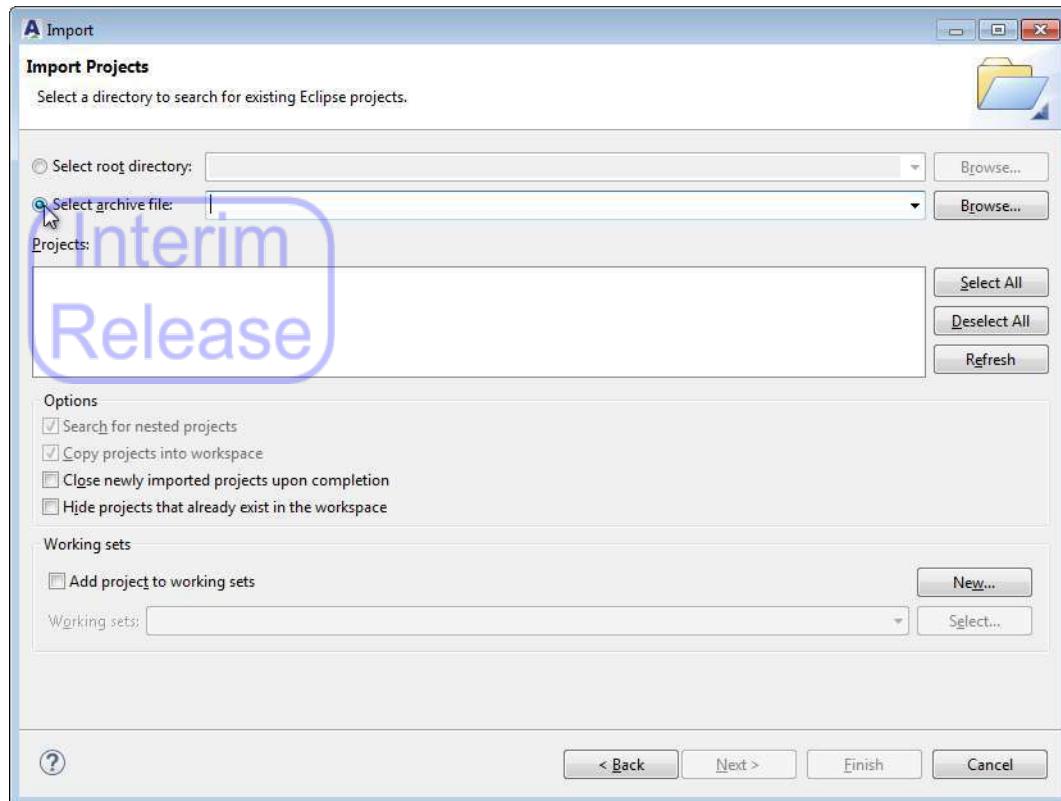
**Step 1** On the AndeSight main menu, click “File” and select “Import...” from the pull-down menu.



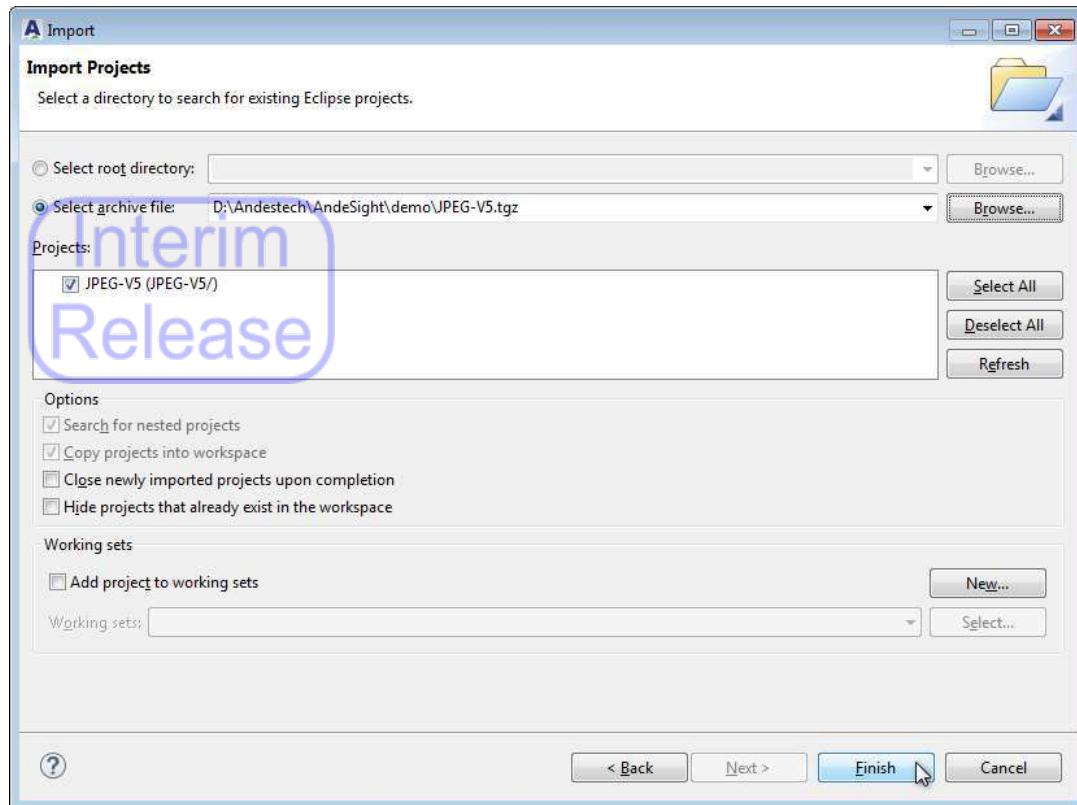
**Step 2** On the invoked **Import** dialog, select “General > Existing Projects into Workspace” and click “Next.”



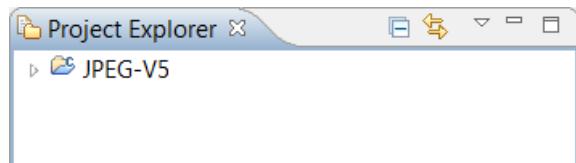
**Step 3** The JPEG/MP3 demo is provided as a tarball in AndeSight. To import the compressed file, select the “Select archive file” radio button and click “Browse...” to search for the file in the invoked wizard. The JPEG/MP3 tarball is [\[JPEG|MP3\]-V5.tgz](#) under [ANDESI GHT\\_ROOT\demo\](#). Then, click “Open”.



**Step 4** Back in the Import dialog, click “Finish” to complete project importation.

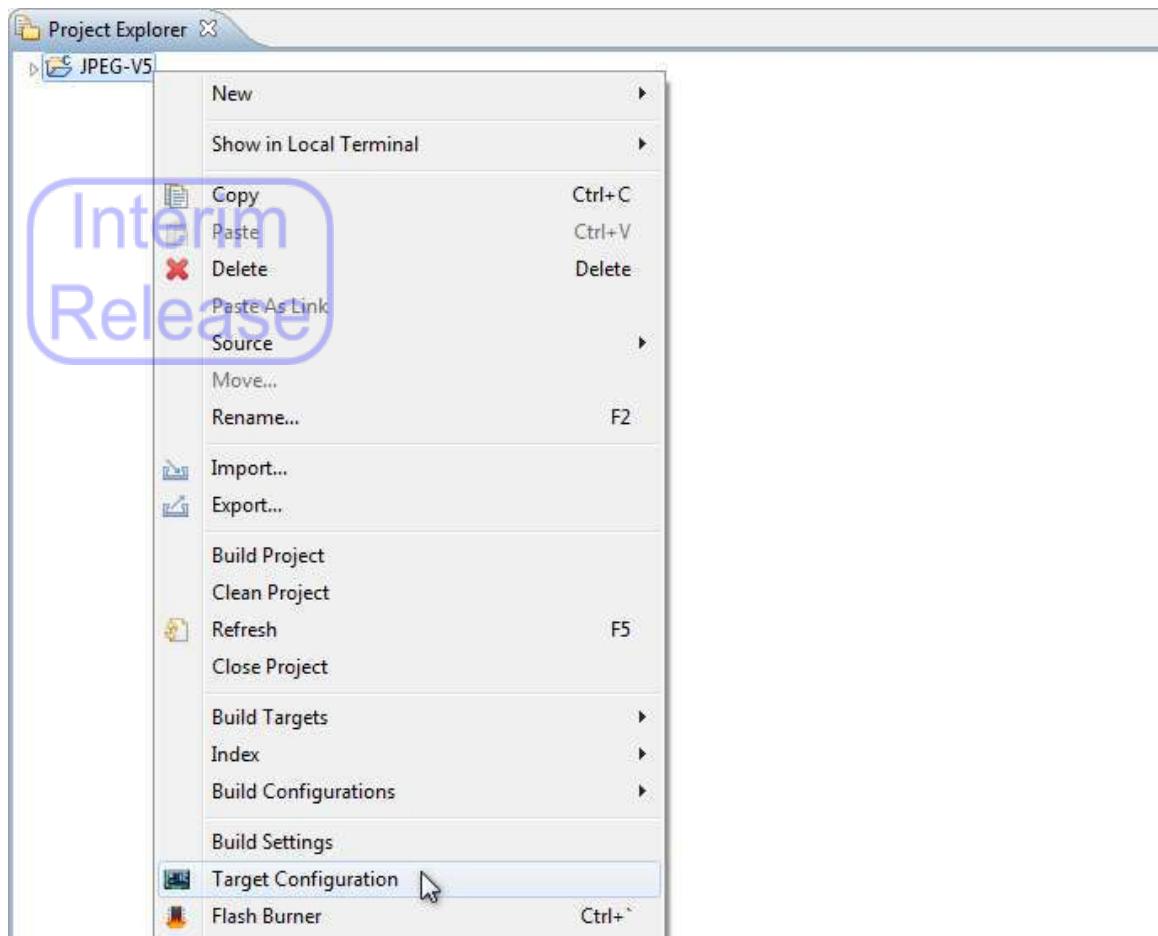


**Step 5** Find the imported JPEG demo project in the **Project Explorer** view.

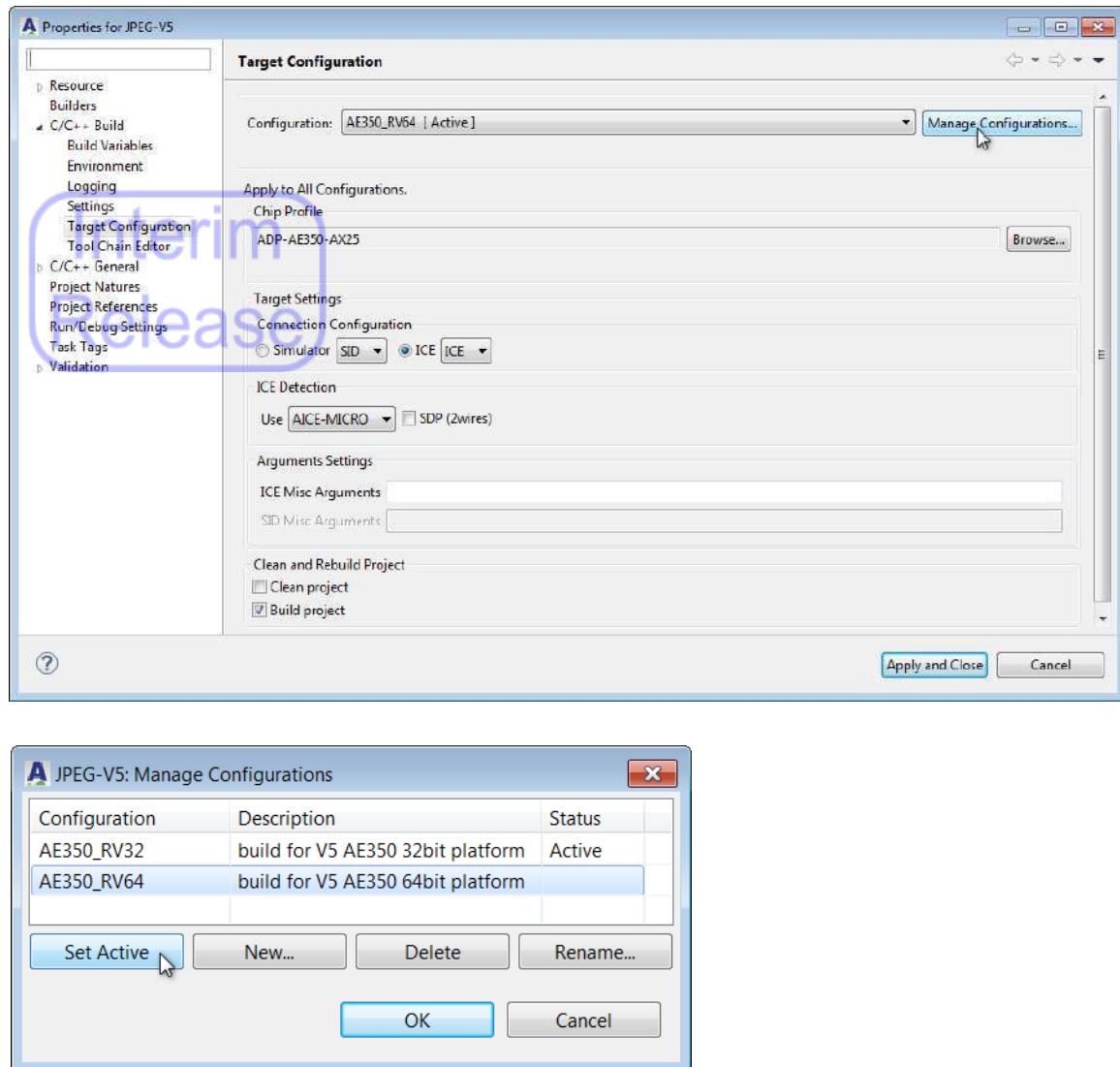


### 3.1.2. Specifying build configuration and target configuration

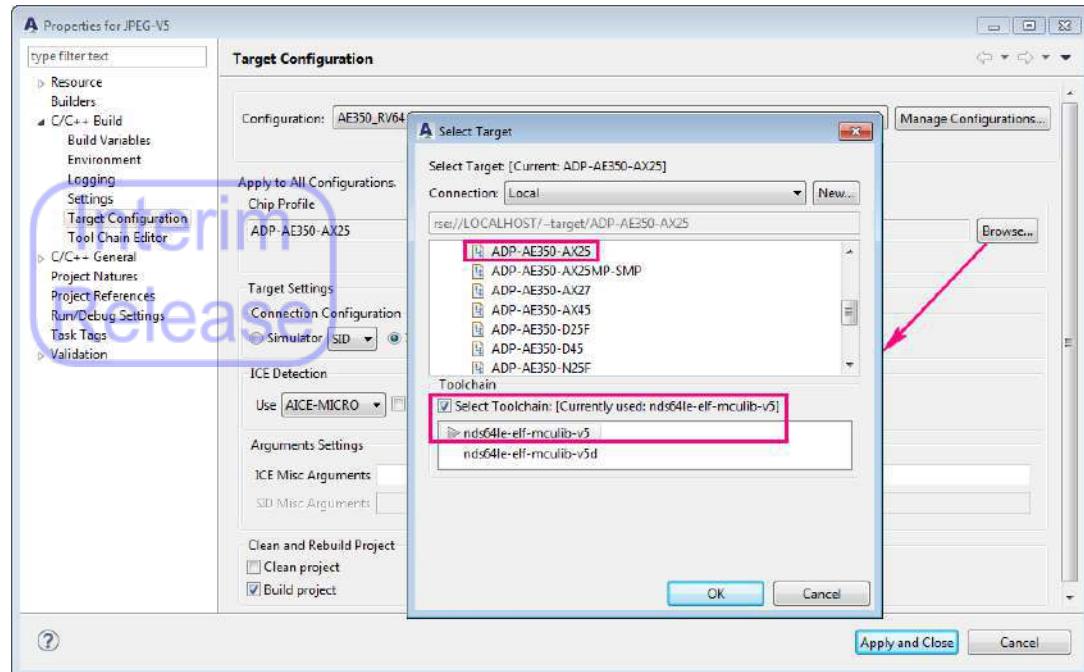
**Step 1** Right-click the project folder and select “Target Configuration” from the pull-down menu.



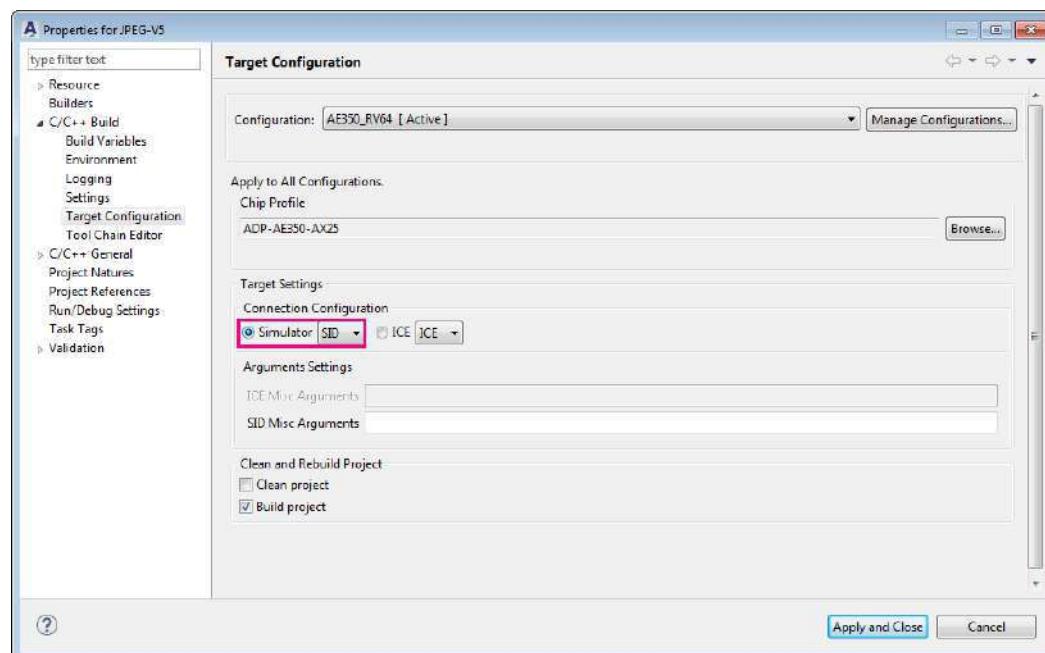
**Step 2** On the **Target Configuration** page of the **Properties** dialog, click the **Manage Configurations** button. In the invoked dialog, specify a build configuration corresponding to your target and set it as active.



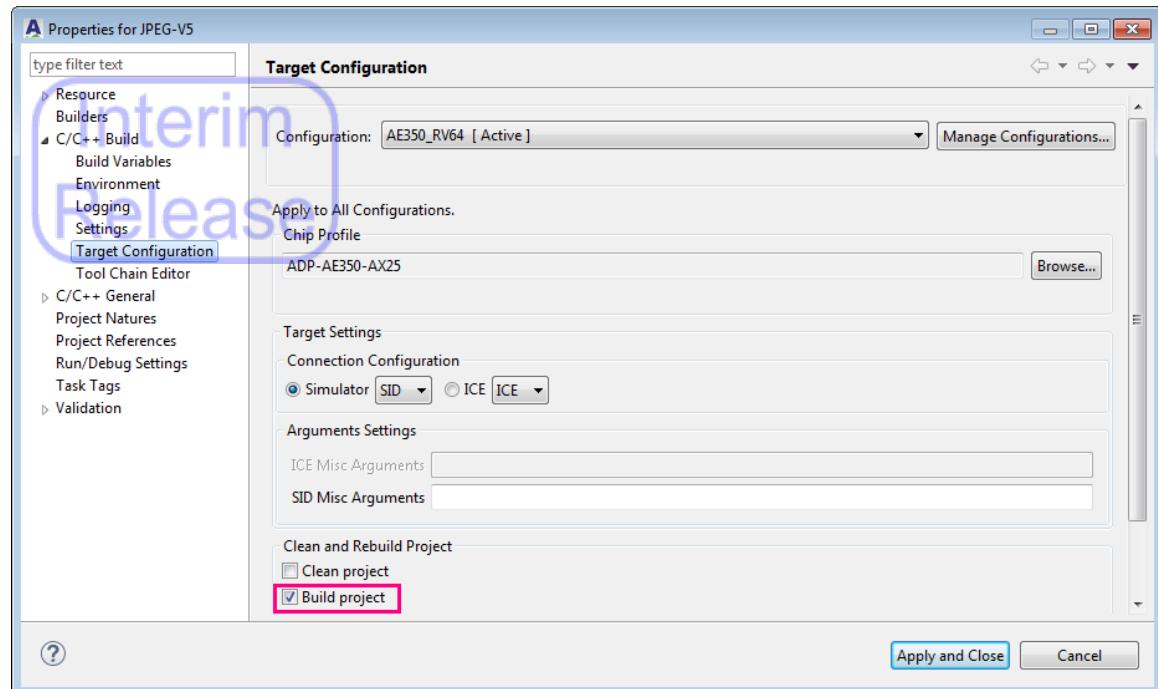
**Step 3** In the **Chip Profile** section, click “Browse...” to invoke the **Select Target** wizard. Designate a target of AE350 platform. Check the “Select Toolchain” option to specify a toolchain from the toolchain list and click “OK”. The example here sets ADP-AE350-AX25 as the target and nds32le-elf-mculib-v5 as the working toolchain.



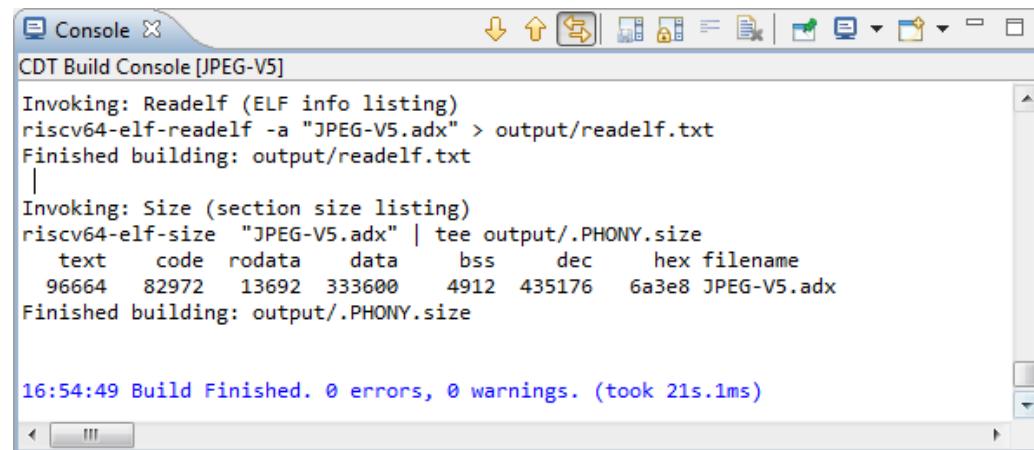
**Step 4** Specify a connection configuration. If you are using a V5 target board with 2-wire debug interface in conjunction with the ICE device AICE-MICRO or AICE-MINI+, make sure to select “ICE” as the connection configuration and check the option “SDP (2 wires)” in the ICE detection section. The example here uses “SID” as the connection configuration.



**Step 5** Select the option “Build project” and click “Apply and Close” on the dialog to build the project.



**Step 6** Verify that the build is successful in the **Console** view.

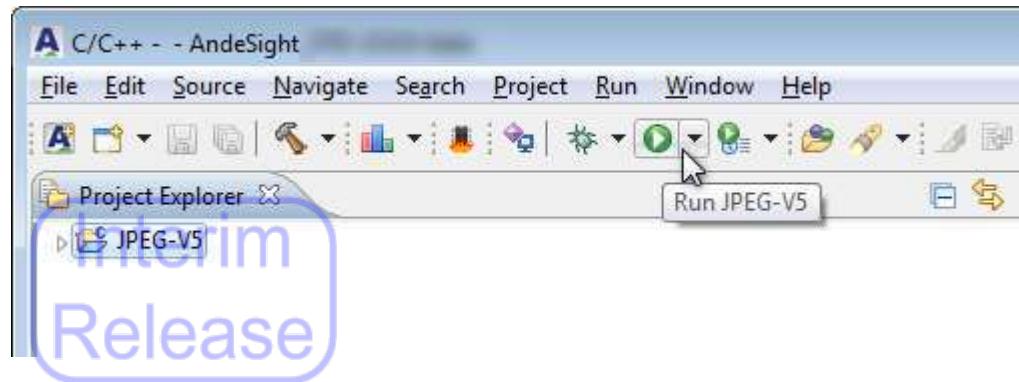


```
CDT Build Console [JPEG-V5]
Invoking: Readelf (ELF info listing)
riscv64-elf-readelf -a "JPEG-V5.adx" > output/readelf.txt
Finished building: output/readelf.txt
|
Invoking: Size (section size listing)
riscv64-elf-size "JPEG-V5.adx" | tee output/.PHONY.size
  text    code   rodata   data   bss   dec   hex filename
  96664   82972   13692  333600   4912  435176  6a3e8 JPEG-V5.adx
Finished building: output/.PHONY.size

16:54:49 Build Finished. 0 errors, 0 warnings. (took 21s.1ms)
```

### 3.1.3. Running the demo project

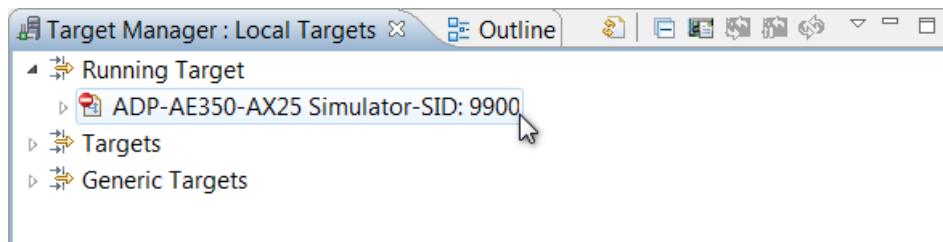
**Step 1** In **Project Explorer**, select the demo project and click  (Run) on the AndeSight toolbar to run the application.



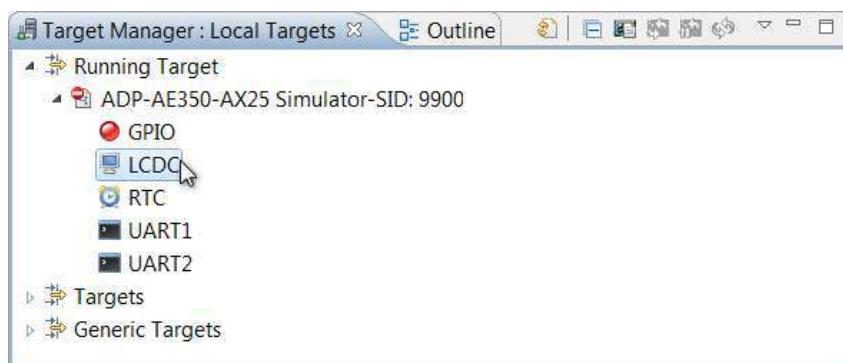
**Step 2** For a JPEG project running on an ICE target, the image files included in the demo project are rendered on the LCD panel. For an MP3 project running on an ICE target, the audio files included in the demo project are rendered over a speaker.

For a JPEG project running on a simulator target, the image files are rendered on the LCDC front-end, which can be invoked as follows:

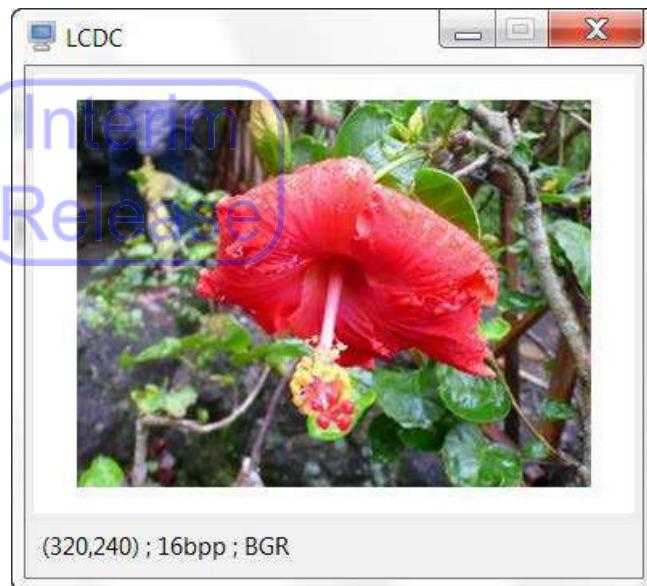
1. Find the launched target from “Target Manager > Running Target” and double-click on it.



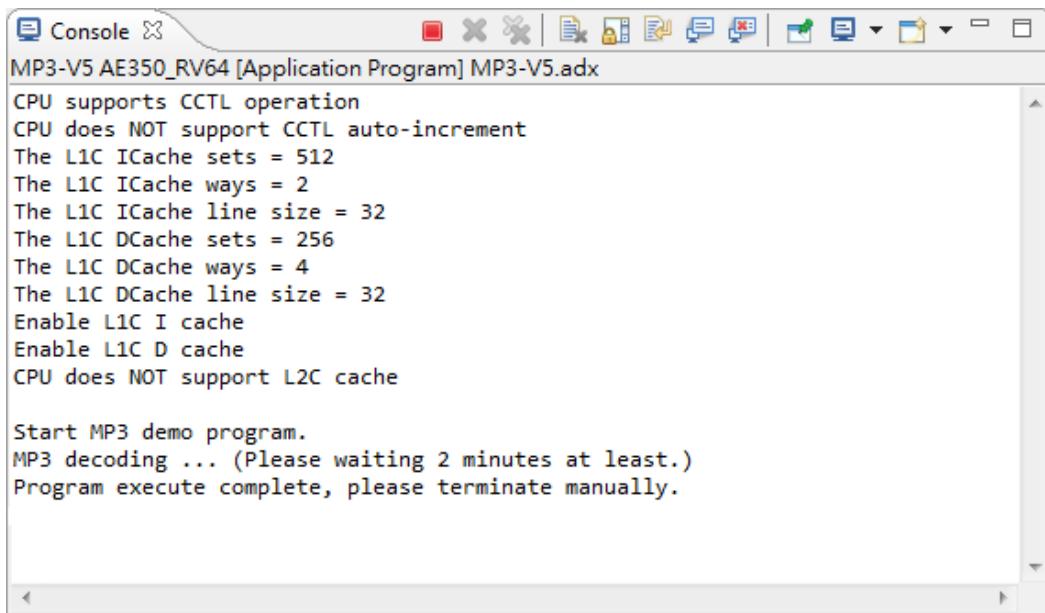
2. Find the devices associated with the target in the expanded tree and double-click “LCDC.”



3. The image files included in the JPEG demo project are now rendered on the virtual LCD panel.



For the MP3 application running on a simulator target, please allow time to render the audio. The **Console** view also informs you that at least 2 minutes are required to decode the MP3 file.



```
Console X
MP3-V5 AE350_RV64 [Application Program] MP3-V5.adx
CPU supports CCTL operation
CPU does NOT support CCTL auto-increment
The L1C ICache sets = 512
The L1C ICache ways = 2
The L1C ICache line size = 32
The L1C DCache sets = 256
The L1C DCache ways = 4
The L1C DCache line size = 32
Enable L1C I cache
Enable L1C D cache
CPU does NOT support L2C cache

Start MP3 demo program.
MP3 decoding ... (Please waiting 2 minutes at least.)
Program execute complete, please terminate manually.
```

### 3.2. Startup demo programs

The Andes startup demo programs demonstrate frequently-used programming patterns for V5 targets. These startup demos placed under

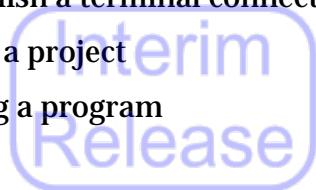
**ANDESI GHT\_ROOT\demo\startup\[ Corvette- F1 | AE250 | AE350 ]\[ clic | plic ]** are as follows:

- demo-plic-V5: sample program for machine interrupts and vectored interrupts related to PLIC (Platform-Level Interrupt Controller)
- demo-plic-novector-V5: sample program for machine interrupts and non-vectored interrupts related to PLIC
- demo-clic-V5: sample program for machine interrupts and interrupts related to CLIC (Core-Local Interrupt Controller)
- demo-printf-V5: sample program for overwriting weak functions
- demo-slaveport-V5: sample program for accessing ILM/DLM via a slave port
- demo-wfi-V5: sample program for low-power control
- demo-pfm-V5: sample program for performance monitoring
- demo-ecc-V5: sample program for ECC (error checking and correction)
- demo-hsp-V5: sample program for hardware stack protection
- demo-powerbrake-V5: sample program for hardware performance throttling
- demo-cache-V5: sample program for cache operations
- demo-cache-lock-V5: sample program for hardware cache lock mechanism
- demo-cplusplus-V5 : sample program for C++ programing
- demo-pmp-V5: sample program for physical memory protection in machine and user mode
- demo-dsp-V5: sample program for using DSP library functions
- demo-hibernate-V5: sample program simulating the wake-up of CPU from deep-sleep mode
- demo-mmu-V5: sample program for using MMU (memory management unit) in machine, supervisor and user mode
- demo-smp-V5: sample program for SMP (symmetric multiprocessing)
- demo-pma-V5: sample program for PMA (physical memory attribute)

In the following subsections, demo-printf-V5, a startup demo for V5 targets, is adopted as an example to illustrate the launching of a debug session for startup demo programs. The operation focuses on the following:

- How to import an existing project

- How to configure build settings
- How to specify a target configuration for a project
- How to set breakpoints
- How to establish a terminal connection with an ICE target
- How to build a project
- How to debug a program



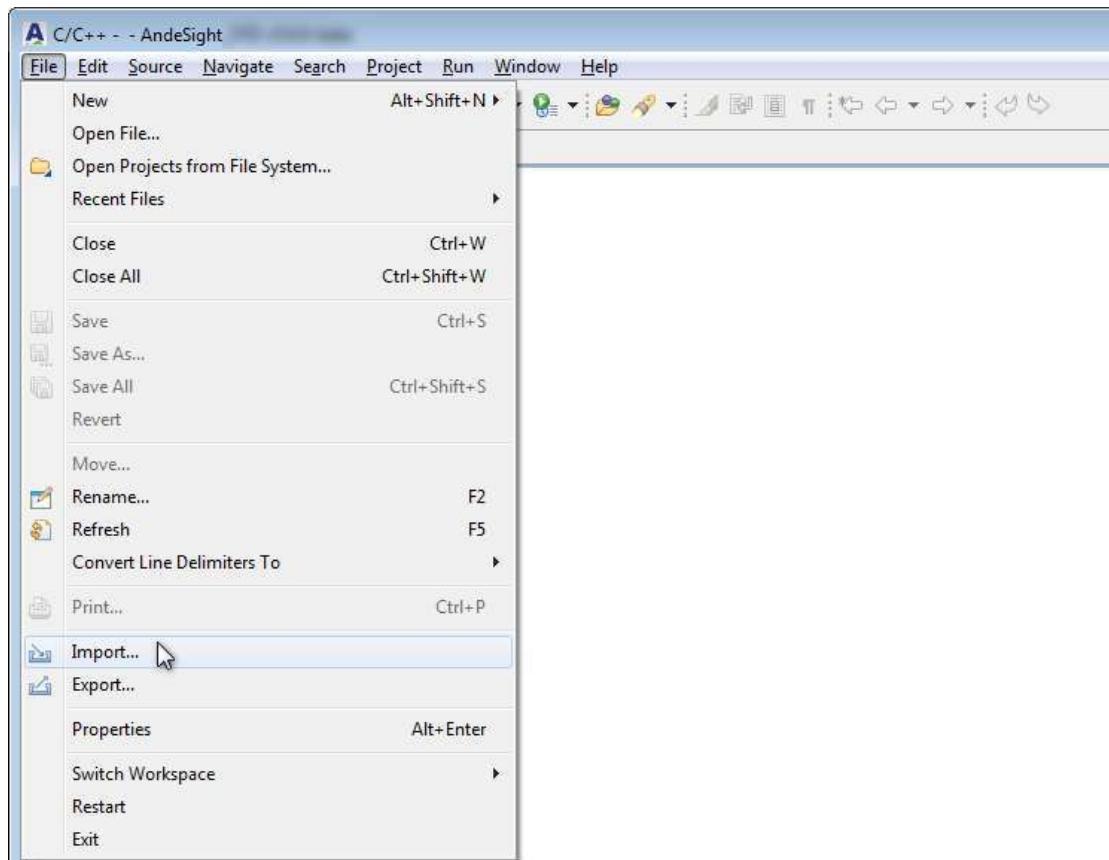
For a detailed introduction to the respective startup demo programs and files in their packages, please refer to Sections 3.2.2 to 3.2.3.

### 3.2.1. Debugging a startup demo project

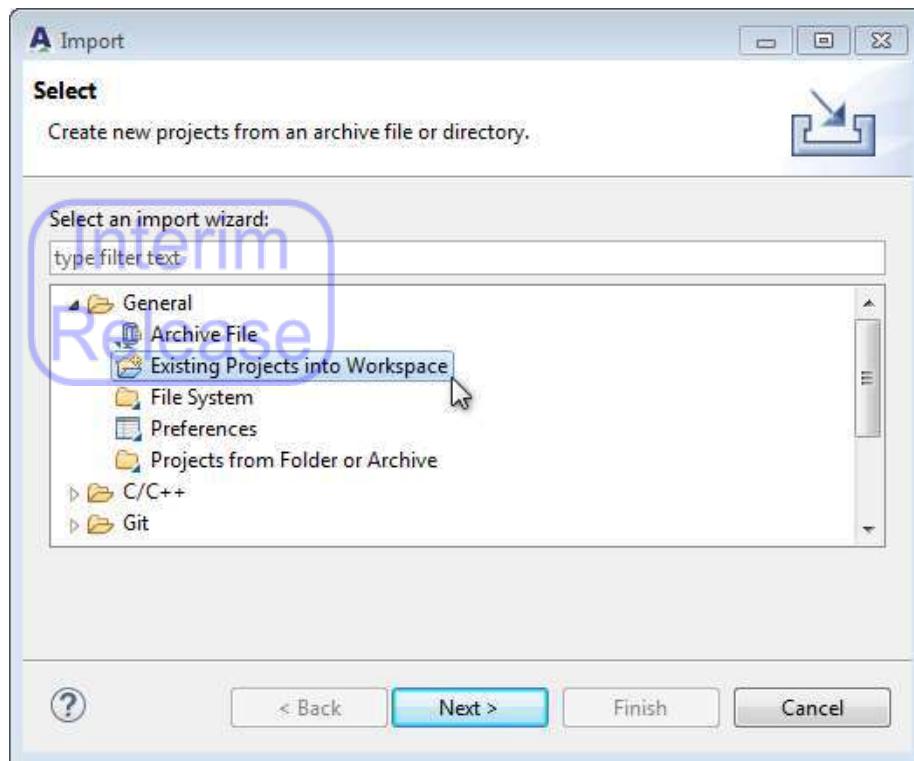
Note that V5 startup demos under **ANDESIGHT\_ROOT\demo\startup\AE250\clic** are currently restricted to the ADP-AE250 targets pre-integrated with N22 and CLIC (Core-Local Interrupt Controller); V5 demos under **ANDESIGHT\_ROOT\demo\startup\Corvette-F1\clic** are restricted to the Corvette-F1 targets pre-integrated with N22 and CLIC. In addition, as N22 targets pre-integrated with PLIC (Platform-Level Interrupt Controller) do not support the vectored PLIC mode, they can't be used for applications under **ANDESIGHT\_ROOT\demo\startup\AE250\plib** except demo-plic-novector-V5.

#### 3.2.1.1 Importing a startup demo project

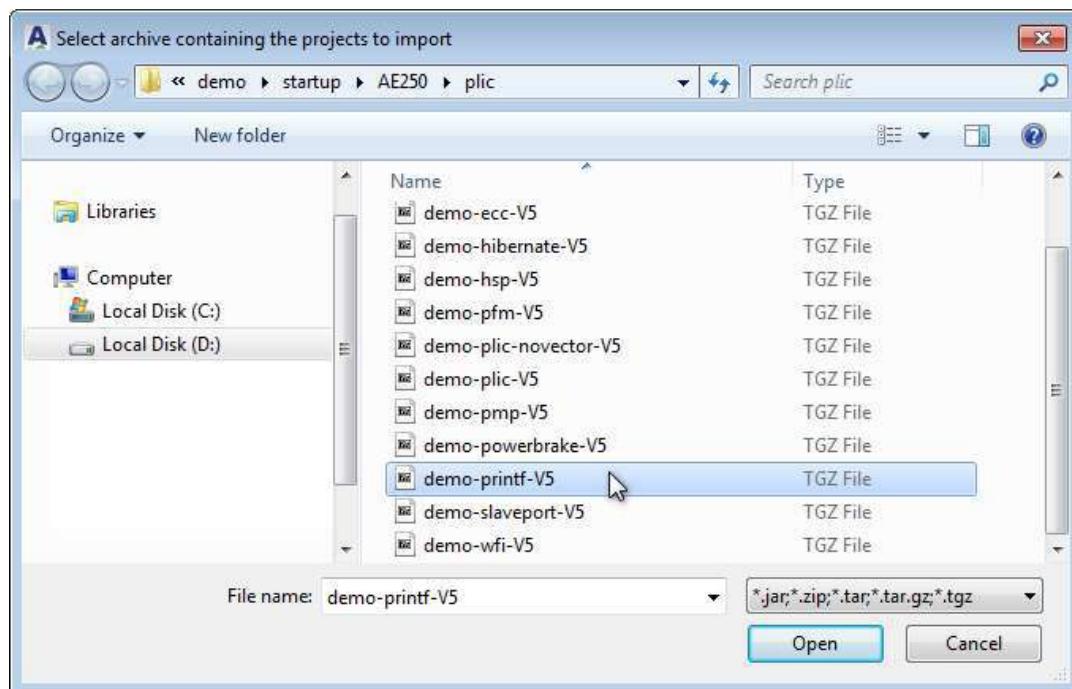
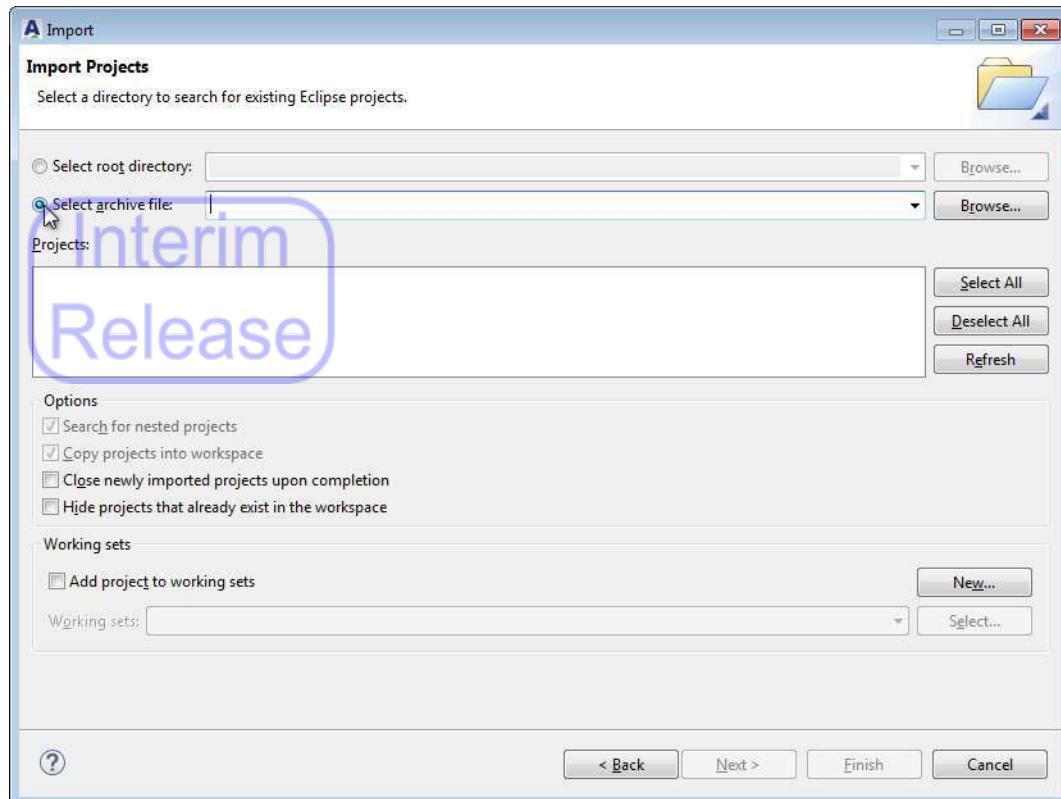
**Step 1** On the AndeSight main menu, select “File > Import...”.



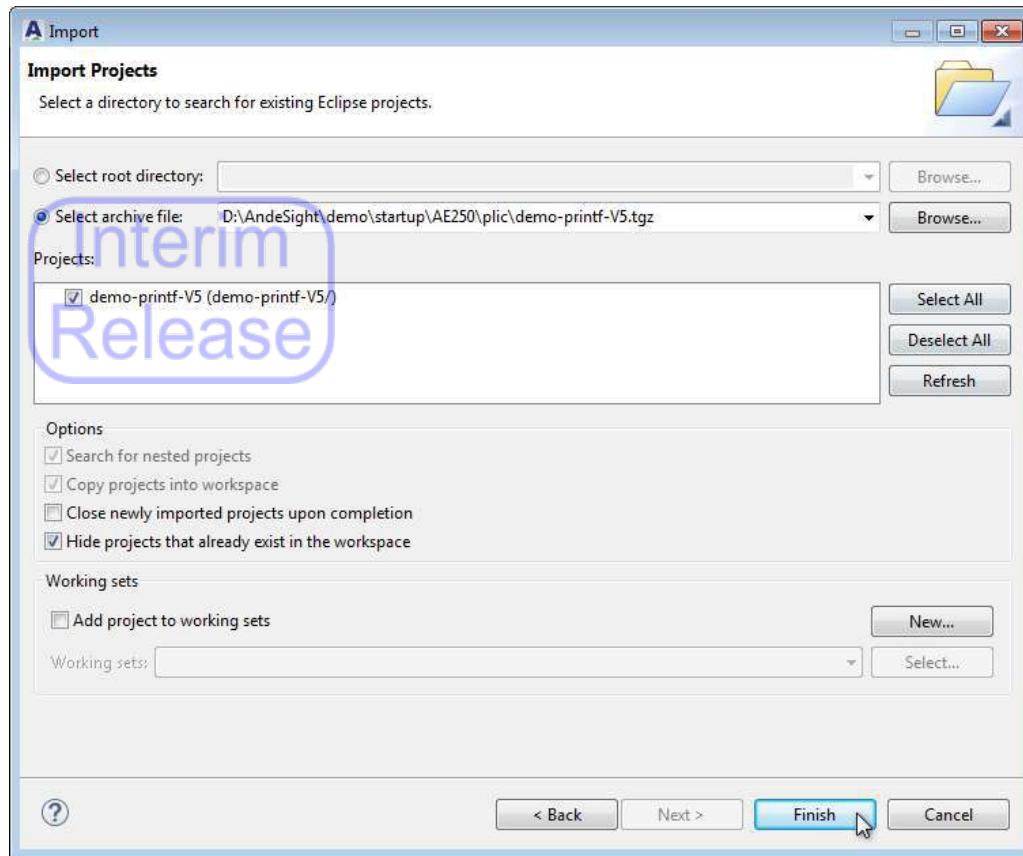
**Step 2** In the invoked **Import** dialog, select “General > Existing Projects into Workspace” and click “Next.”



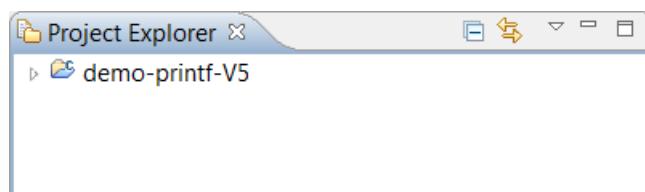
- Step 3** Startup demos are provided as tarballs in AndeSight; therefore, choose the “Select archive file” radio button. Depending on the platform of your target system, click “Browse...” to search for a startup demo project in the invoked wizard and click “Open”. The following summarize the directories of startup demos:
- Startup demos for ADP-AE250 targets pre-integrated with N22 and CLIC (Core-Local Interrupt Controller):  
**ANDESI GHT\_ROOT\demo\startup\AE250\cli c**.
  - Startup demos for Corvette-F1 targets pre-integrated with N22 and CLIC: **ANDESI GHT\_ROOT\demo\startup\Corvette- F1\cli c**
  - Startup demos for targets pre-integrated with PLIC (Platform-Level Interrupt Controller):  
**ANDESI GHT\_ROOT\demo\startup\[ AE250 | AE350 ]\pl i c**



**Step 4** Back in the Import dialog, click “Finish” to complete the process.



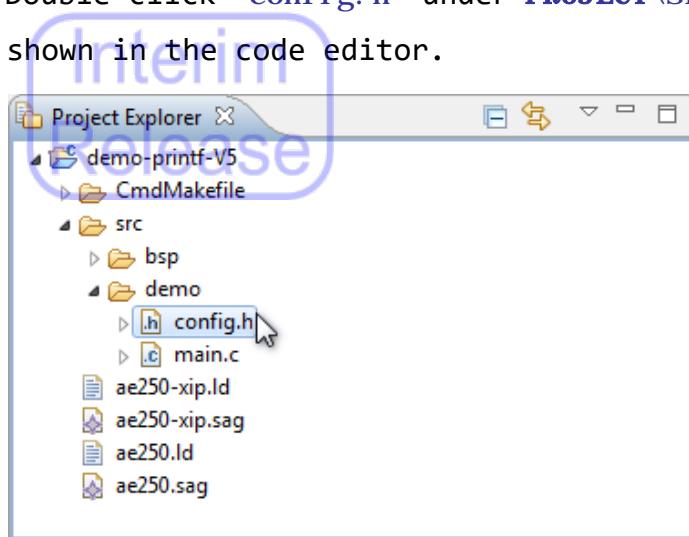
**Step 5** You will find the imported startup demo project in the **Project Explorer** view.



### 3.2.1.2 Modifying the configuration file

Modification of configuration files is necessary before building a startup demo for targets.

- Step 1** Double-click “config.h” under **PROJECT\src\demo** to find its content shown in the code editor.

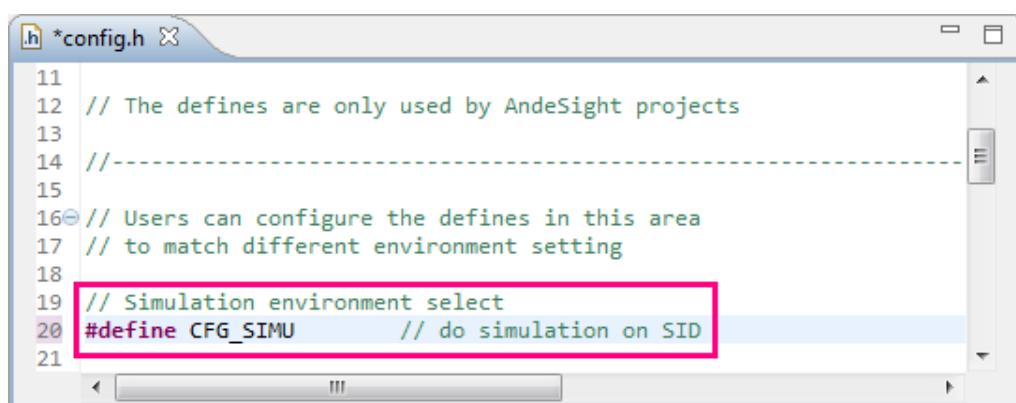


- Step 2** For simulator targets, enable the simulator macro “#define CFG\_SIMU” by removing the comment. Skip this step if you want to debug the startup demo on an ICE target.

---

#### NOTE

1. demo-slaveport-V5, demo-mmu-V5, demo-powerbrake-V5, demo-cache-lock-V5 and demo-smp-V5 are not supported on simulator targets.
  2. demo-pma-V5 is currently supported on ICE/simulator targets of A(X)27 CPU cores or ICE targets of 45 series CPU cores.
- 



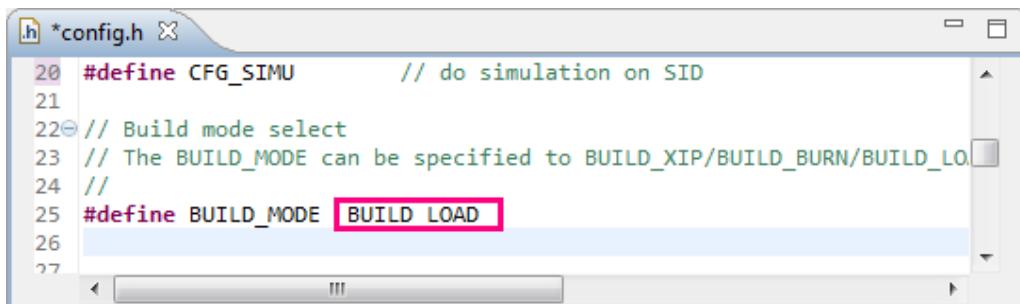
```

11
12 // The defines are only used by AndeSight projects
13
14 //-----
15
16// Users can configure the defines in this area
17// to match different environment setting
18
19// Simulation environment select
20#define CFG_SIMU      // do simulation on SID
21

```

**Step 3** Specify a build mode for your demo project:

- Specify **BUILD\_LOAD**, if the program will be loaded by GDB or eBIOS.
- Specify **BUILD\_BURN**, if the program will be burned to flash but run in RAM.
- Specify **BUILD\_XIP**, if the program will be burned to flash and run in flash.



```

20 #define CFG_SIMU      // do simulation on SID
21
22// Build mode select
23// The BUILD_MODE can be specified to BUILD_XIP/BUILD_BURN/BUILD_L
24//
25#define BUILD_MODE BUILD LOAD
26
27
  
```

---

**NOTE**

1. The following demos are restricted to specific modes.
    - demo-ecc-V5 supports LOAD and BURN modes only.
    - demo-pmp-V5 and demo-mmu-V5 support the LOAD mode only.
  2. There are SaG files dedicated to each mode. For instructions on specifying a SaG file, please refer to Section 3.2.1.4, Step 5.
- 

**Step 4** Save the changes by pressing “Ctrl + S”.

### 3.2.1.3 Configuring CPU frequency required by startup demo project

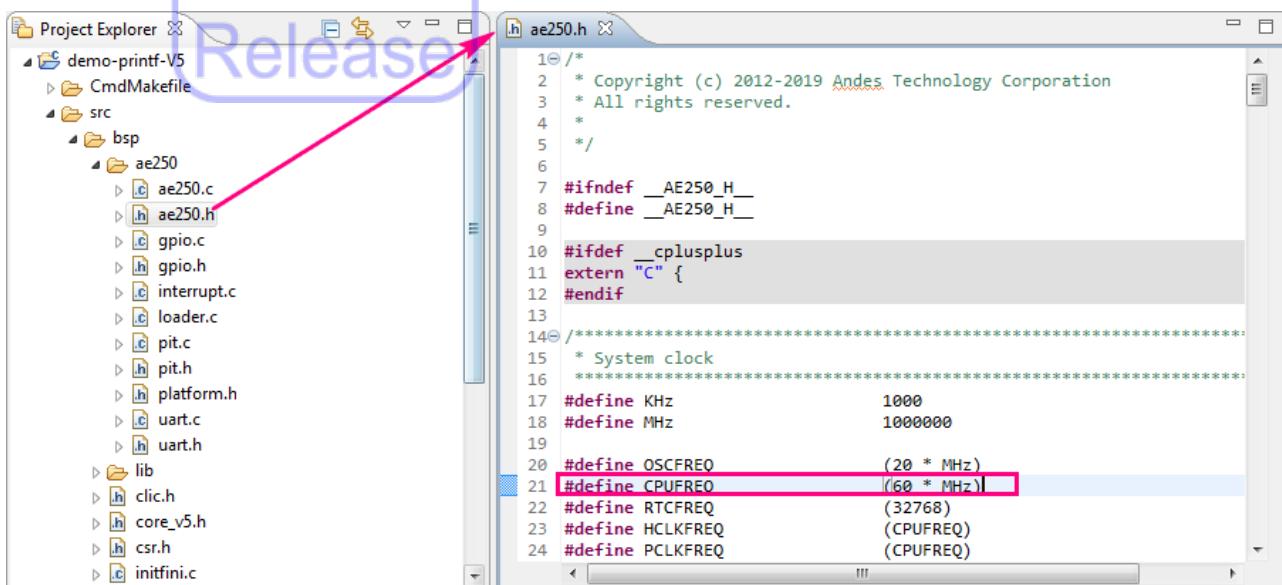
**Step 1** Check boot messages or associated data sheet for the CPU speed of your target. For example, the boot messages of an ADP-XC7 target board of AE250 platform pre-integrated with N22 shows that its CPU frequency is 20MHz.

**Step 2** From **PROJECT\src\bsp\PLATFORM**, double-click the platform header file

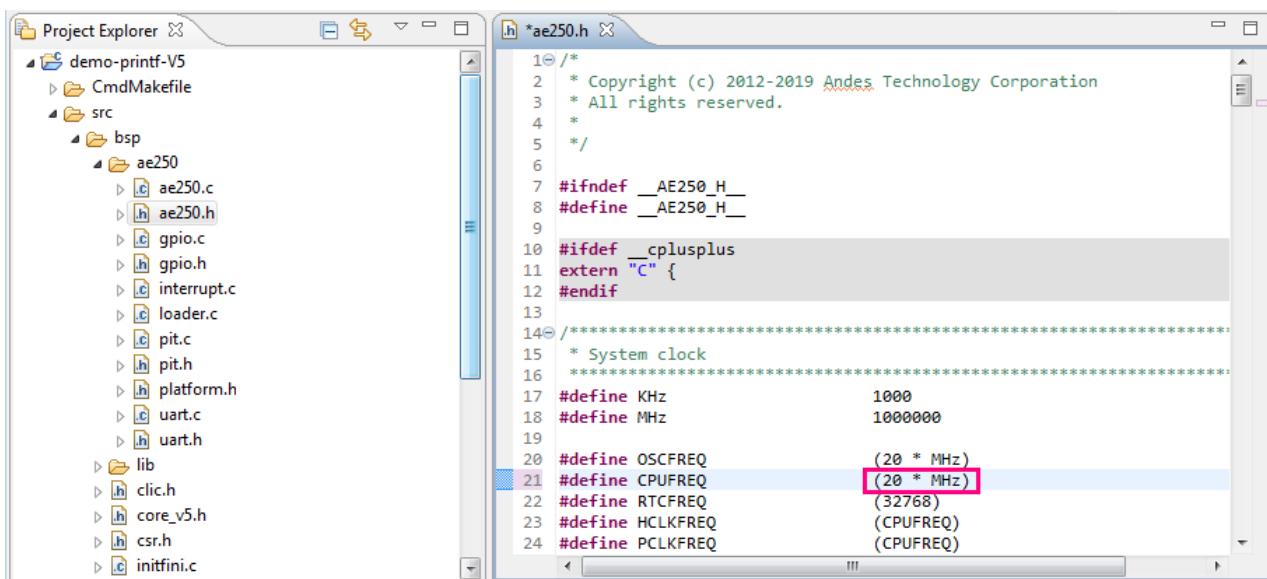
**PLATFORM.h** for your startup program to open it in the code editor.

Check **CPUFREQ** in the file for required CPU frequency of the program.

The figure below shows the required CPU frequency defined in **ae250.h** for running demo-printf-V5 as 60MHz.



**Step 3** Edit the value of **CPUFREQ** in the file if it does not match the CPU speed of your target obtained in Step 1. For example, to debug demo-printf-V5 on a N22 target with CPU frequency of 20MHz, change the value of **CPUFREQ** to “**20 \* MHz**”, as illustrated below.

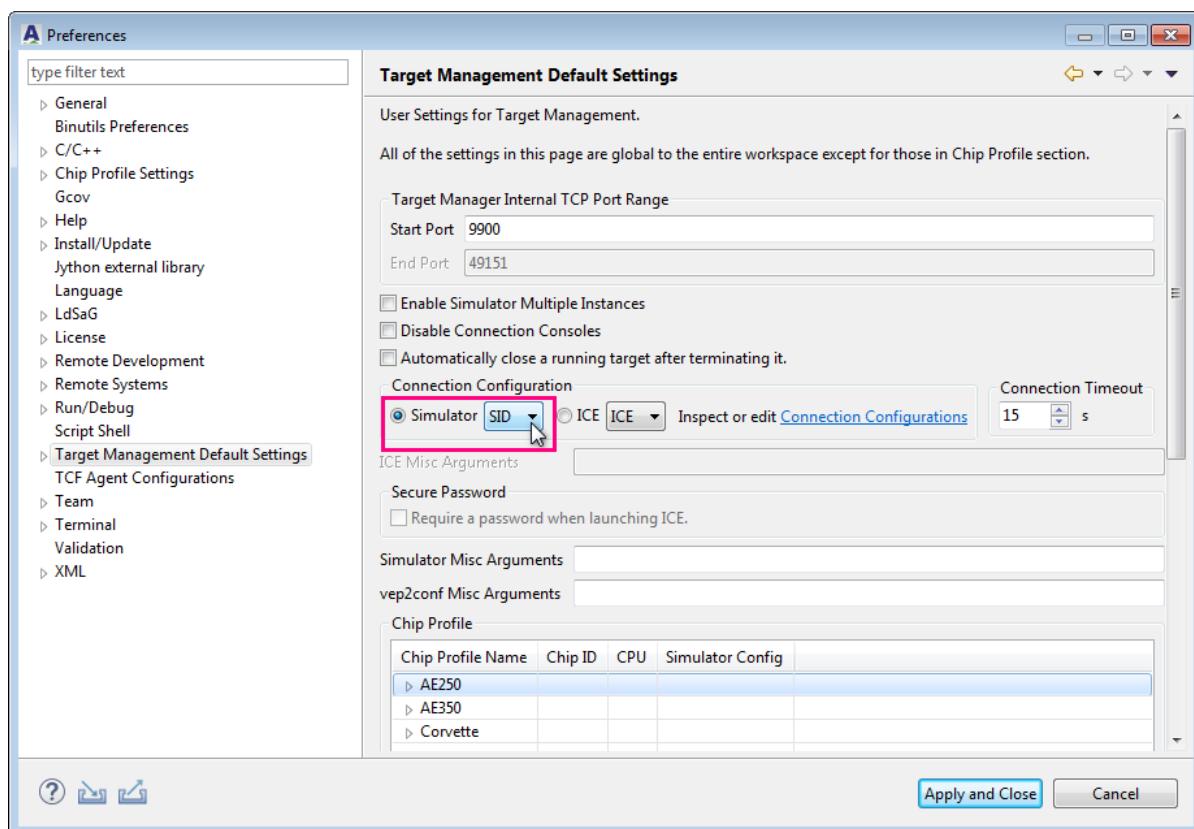


**Step 4** Save the changes by pressing “Ctrl + S”.

### 3.2.1.4 Specifying target configuration and SaG file for a build

**Step 1** For demos other than demo-mpu, jump to the next step. To run demo-mpu on a simulator target, follow the instructions below on specifying simulator arguments:

1. Invoke the **Preferences** dialog by clicking “AndeSight main menu > Window > Preferences”.
2. Select “Target Management Default Settings” in the navigation pane of the **Preferences** dialog to enter the **Target Management Default Settings** page. Then, specify “SID” as the connection configuration.



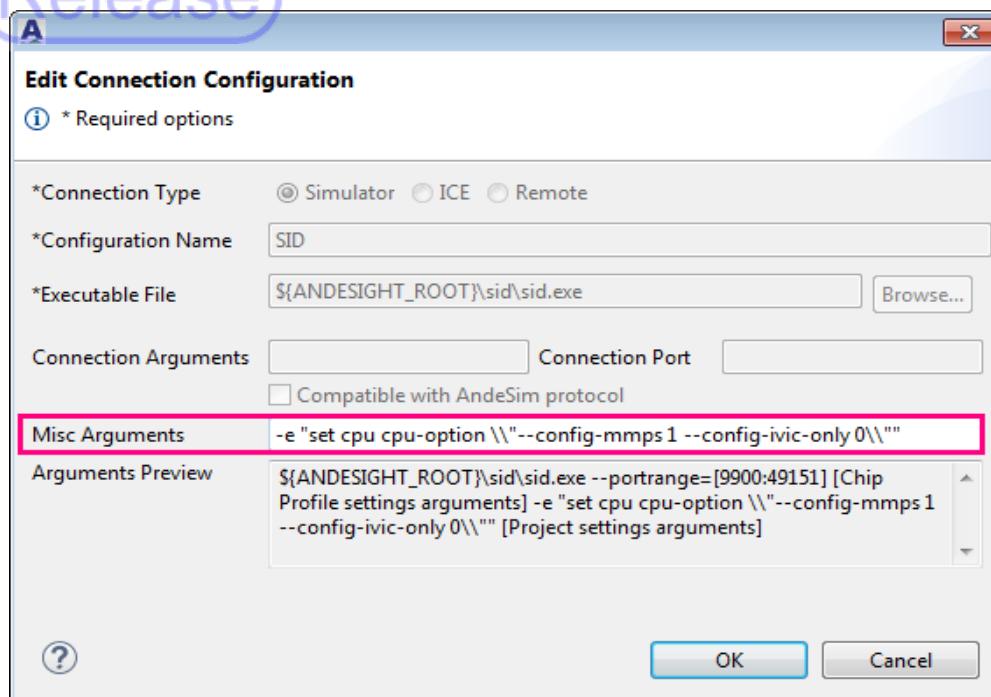
3. Click the “Simulator Misc Arguments” field. In the invoked **Edit Connection Configuration** dialog, enter the following argument and

click “OK”:

```
-e "set cpu cpu-option \\\"--config-mmmps 1 --config-ivic-only 0\\\""
```

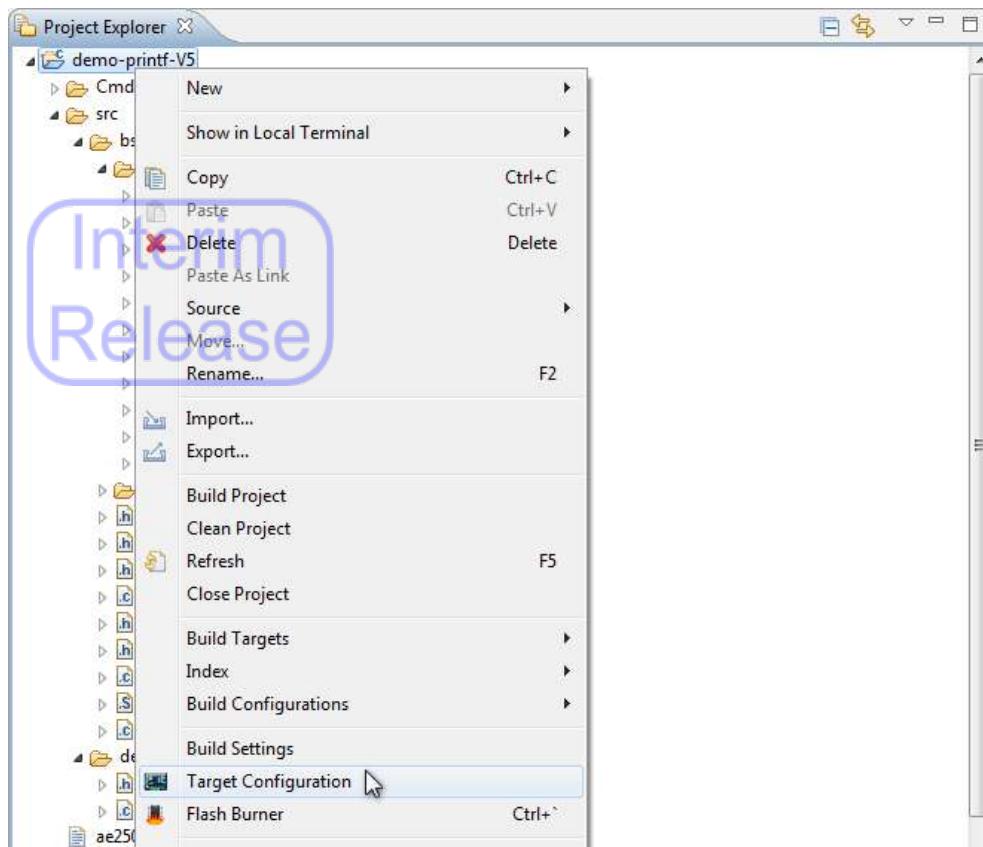
#### NOTE

The above commands are for the Windows environment. For Linux, replace the escape character “\\\" with “\\”.

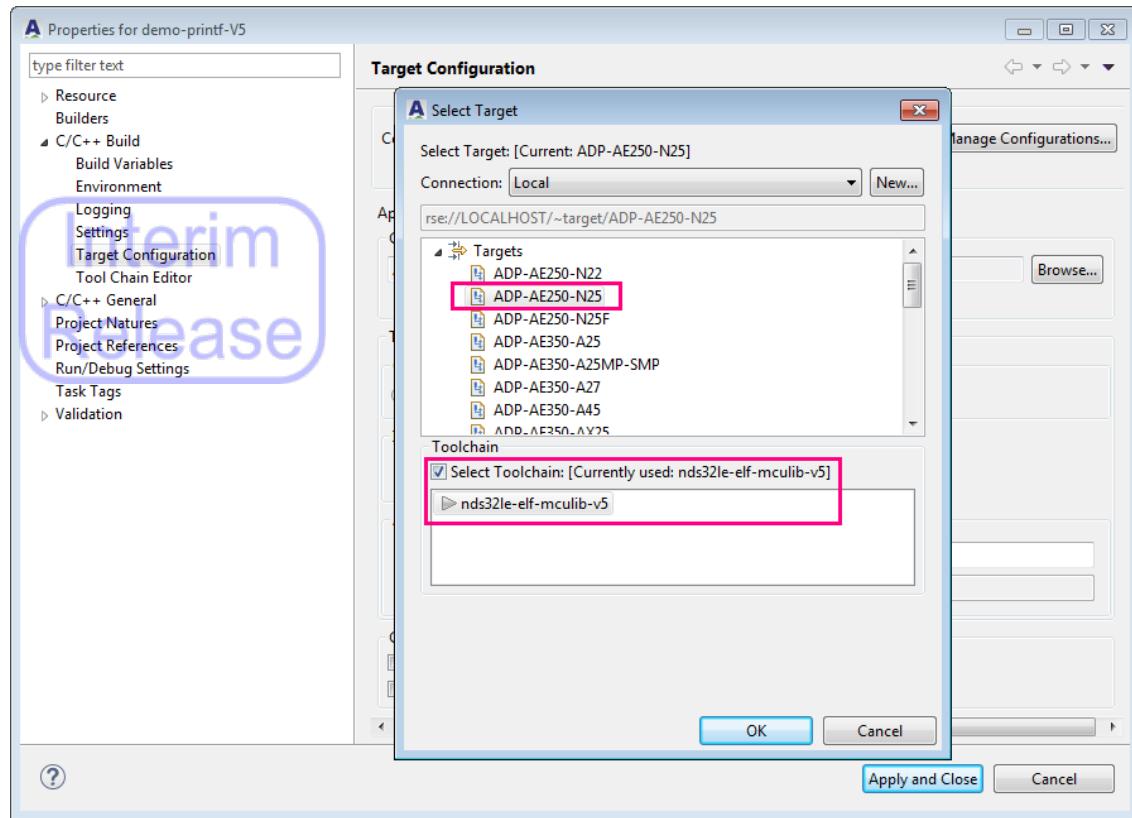


4. Click “Apply and Close” in the **Preferences** dialog.

**Step 2** Right-click the project in **Project Explorer** and select “Target Configuration” from the pull-down menu to enter the **Target Configuration** page.



**Step 3** In the **Chip Profile** section, click “Browse...” to designate a supported target for the startup demo, specify a desired toolchain in the invoked dialog, and click “OK”. The example here sets ADP-AE250-N25 as the target for the demo-printf-V5 project and selects nds32le-elf-mculib-v5 as the working toolchain.




---

**NOTE**

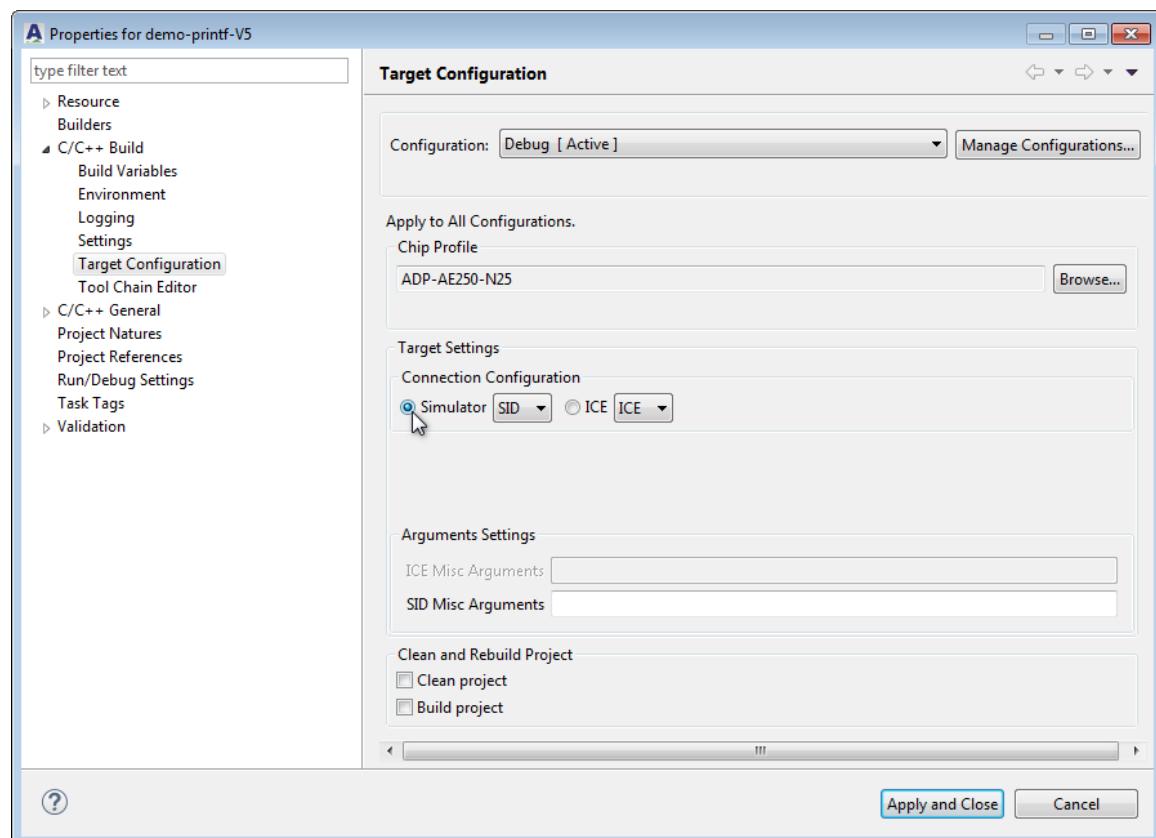
1. demo-mmuv5 is restricted to AE350 targets pre-integrated with PLIC and the MMU feature.
2. demo-cache-lock-V5 is restricted to AE350 targets with PLIC and the cache-lock feature.
3. demo-smp-V5 is restricted to AE350 targets pre-integrated with SMP dual cores and PLIC; demo-pma-V5 is restricted to AE350 targets pre-integrated with PLIC and the PMA feature.
4. Startup demos under **ANDESIGHT\_ROOT\demo\startup\AE250\clic** are restricted to ADP-XC7 targets of AE250 platform pre-integrated with N22 and CLIC (i.e., ADP-AE250-N22); startup demos under **ANDESIGHT\_ROOT\demo\startup\Corvette-F1\clic** are restricted to the Corvette-F1 targets pre-integrated with N22 and CLIC (i.e., ADP-Corvette-F1-N22).
5. demo-plic-V5, demo-powerbrake-V5, demo-hibernate-V5,

and demo-slaveport-V5 are not supported on N22 targets pre-integrated with PLIC.

6. To use a Corvette-F1 target pre-integrated with fixed-configuration N22 RTL in Andes FreeStart program, make sure you select the chip profile “ADP-Corvette-F1-N22” and the toolchain “nds32le-elf-[newlib|mcuLIB]-v5e”.

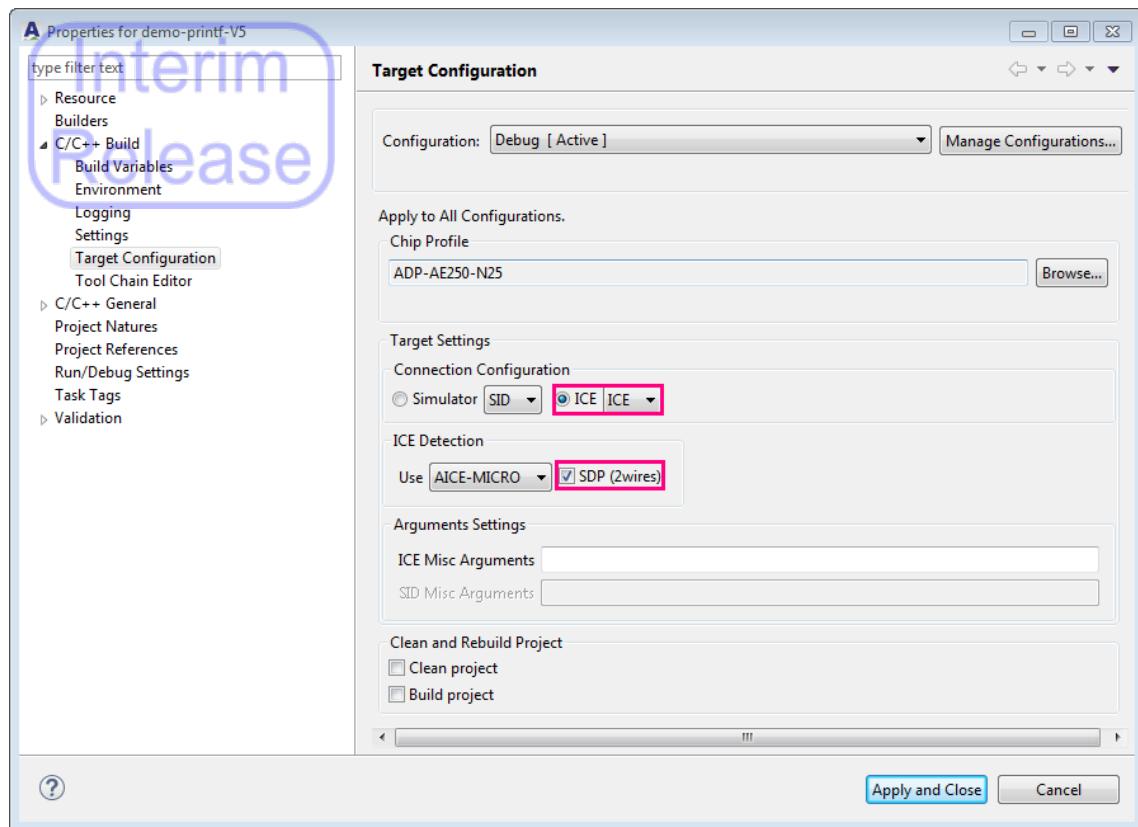
#### Step 4 Select a connection configuration for your target.

For demo-slaveport-V5, demo-mmu-V5, demo-powerbrake-V5, demo-cache-lock-V5 or demo-smp-V5, make sure you select the configuration “ICE” as these applications are not supported on simulator targets. Also, please note that demo-pma-V5 is currently supported on ICE/simulator targets of A(X)27 CPU cores or ICE targets of 45 series CPU cores.



If you are using a V5 target board with 2-wire debug interface in

conjunction with the ICE device AICE-MICRO or AICE-MINI+, make sure to select “ICE” as the connection configuration and check the option “SDP (2 wires)” in the ICE detection section.



To run a startup demo on a Corvette-F1 simulator target pre-integrated with fixed-configuration N22 RTL in Andes FreeStart program, make sure to select the connection configuration “SID” and enter the following in the SID Misc Arguments field:

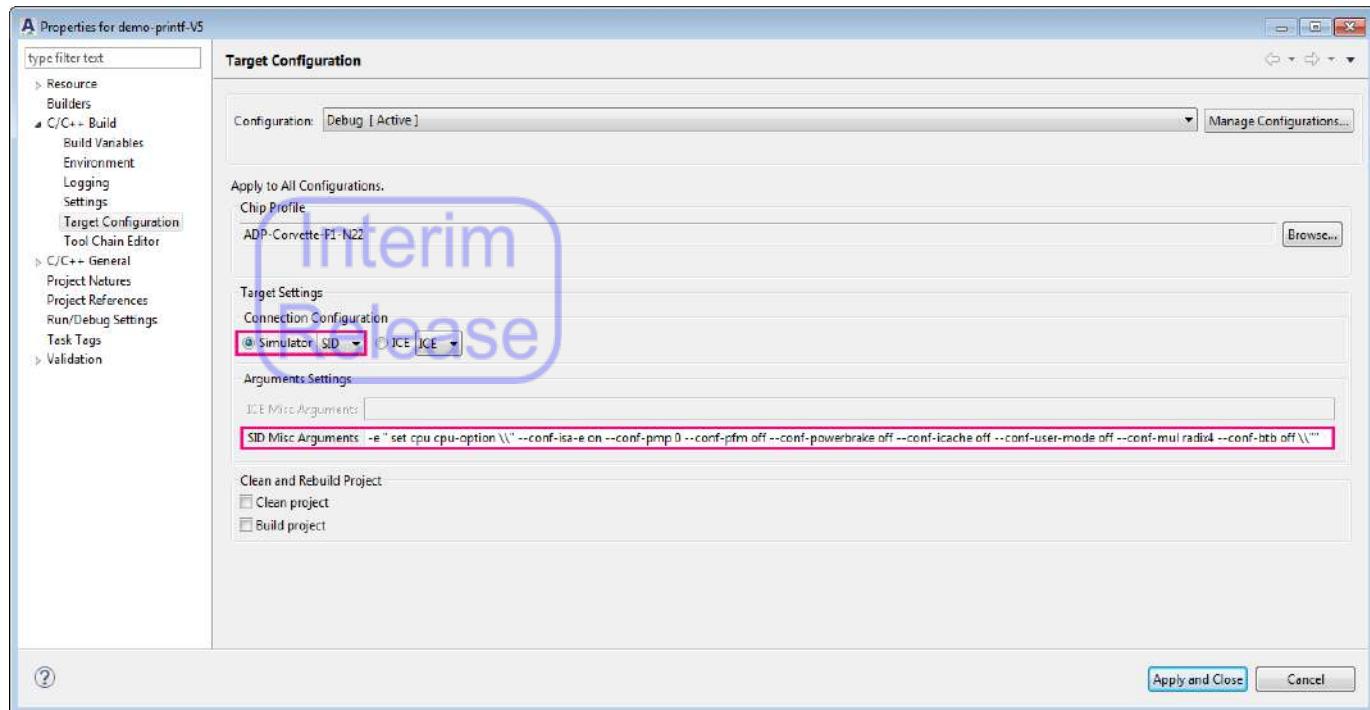
```
-e " set cpu cpu-option \\\" --conf-is-a-e on --conf-pmp 0 --conf-pfm
off --conf-powerbrake off --conf-i-cache off --conf-user-mode off
--conf-mul radix4 --conf-btb off \\\""
```

---

#### NOTE

The above commands are for the Windows environment. For Linux, replace the escape character “\\\" with “\”.

---



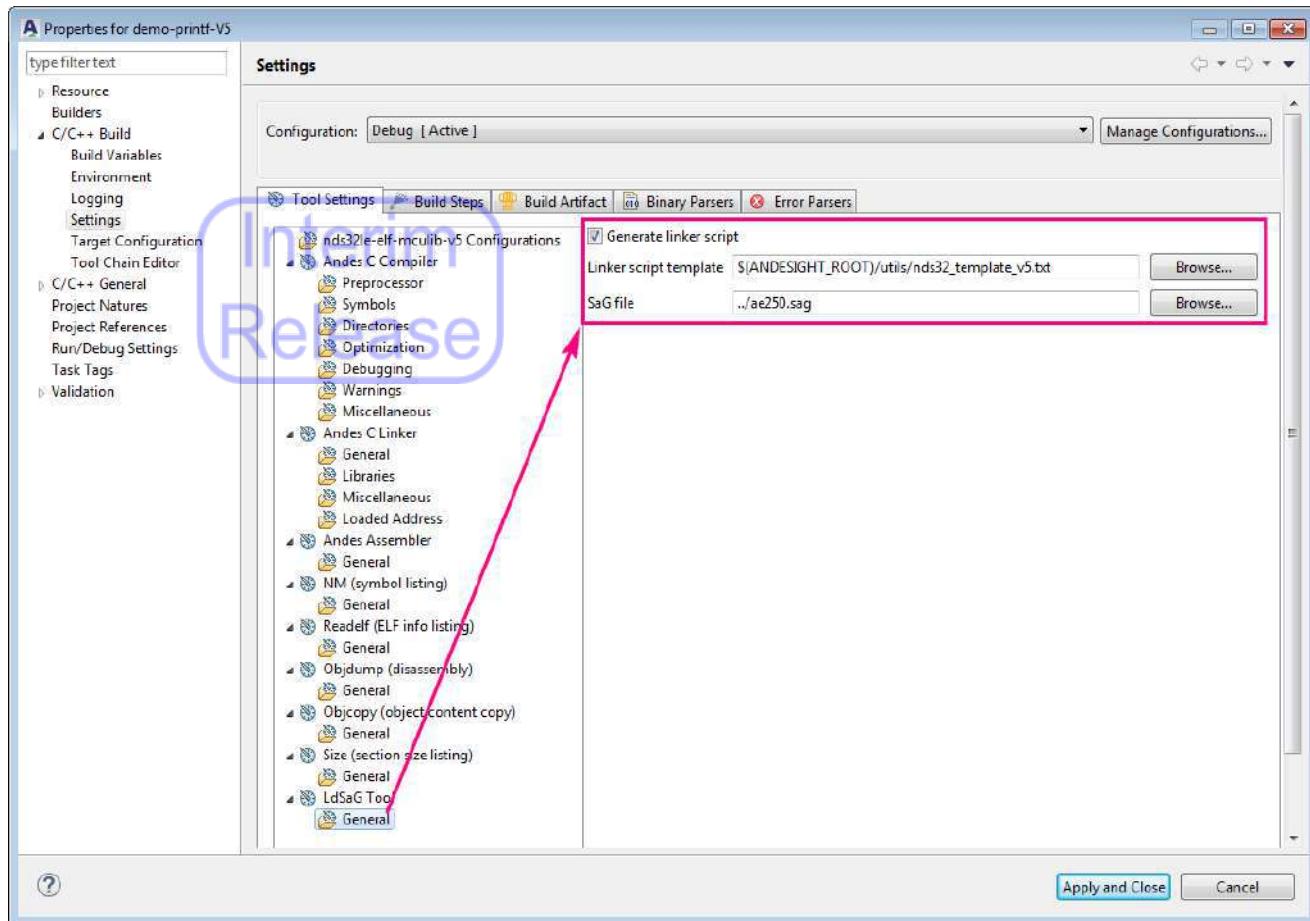
**Step 5** Check the table below to find the SaG file corresponding to your project, platform, build mode, and addressing.

**Table 9. SaG files and linker scripts for demos of various configurations**

Demo	Plat-form	Mode	Address-ing	SaG file and linker script				
demo-cache-V5	AE250	LOAD/BURN	32/64 bit	ae250. [ sag ld]				
demo-cache-lock-V5								
demo-cplusplus-V5		XIP		ae250-xip. [ sag ld]				
demo-dsp-V5								
demo-hibernate-V5								
demo-hsp-V5								
demo-plic-V5		LOAD/BURN		ae350. [ sag ld]				
demo-plic-novector-V5								
demo-pfm-V5	AE350							
demo-pma-V5								

Demo	Plat-form	Mode	Address-ing	SaG file and linker script
demo-powerbrake-V5		XIP		ae350-xip. [sag ld]
demo-printf-V5		XIP		ae250. [sag ld]
demo-slaveport-V5		XIP		ae350. [sag ld]
demo-wfi-V5		XIP		ae350. [sag ld]
demo-ecc-V5	AE250	LOAD/BURN		ae250. [sag ld]
	AE350			ae350. [sag ld]
demo-mmU-V5	AE350	LOAD		ae350. [sag ld]
demo-pmp-V5	AE250	LOAD		ae250. [sag ld]
	AE350			ae350. [sag ld]
demo-smp-V5	AE350	LOAD/BURN		ae350. [sag ld]
		XIP		ae350-xip. [sag ld]
demo-clic-V5	AE250	LOAD/BURN		ae250. [sag ld]
		XIP		ae250-xip. [sag ld]

Select “C/C++ Build > Settings” in the navigation pane of the **Properties** dialog and select “LdSaG Tool > General” in the **Tool Settings** tab. Check the “Generate linker script” option, specify the linker script template `nds32_template_v5.txt` for V5 targets and select the appropriate SaG file for your project. The following example selects the SaG file `ae250.sag` to build `demo-printf-V5` for running on ADP-AE250-N25 under LOAD mode.



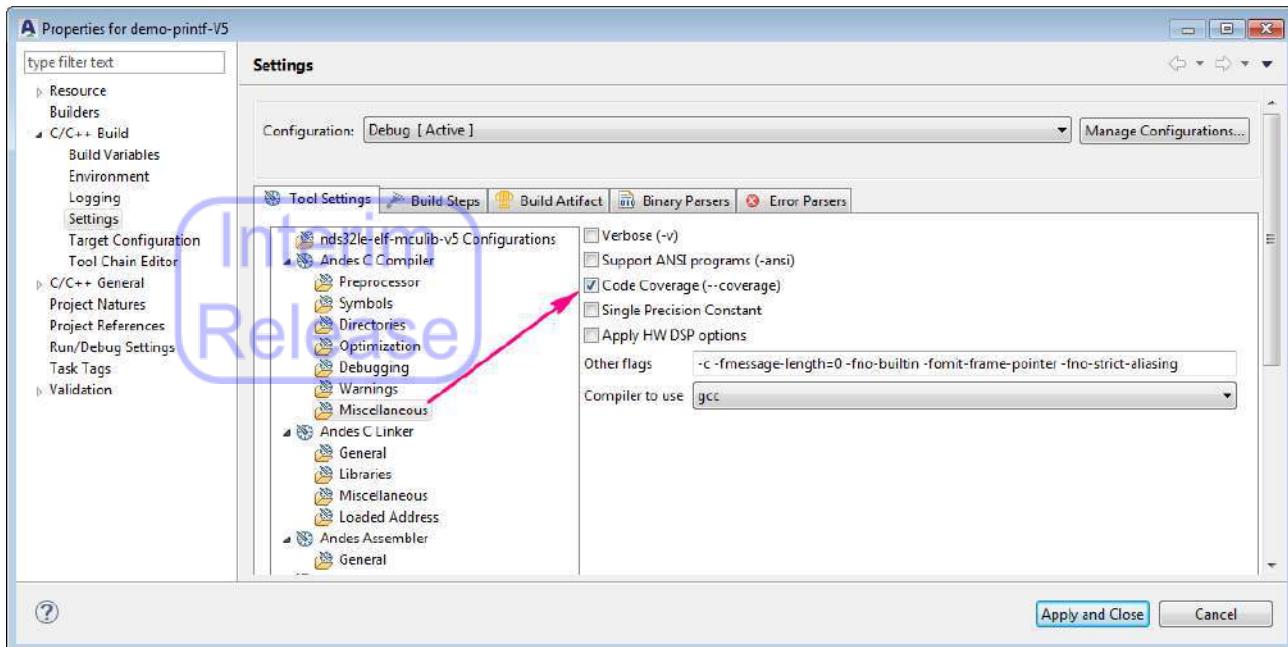
**Step 6** Skip this step to forego code coverage analysis for startup demos. For code coverage analysis, click “Andes C Compiler > Miscellaneous” and select the “Code Coverage (–coverage)” option.

---

#### NOTE

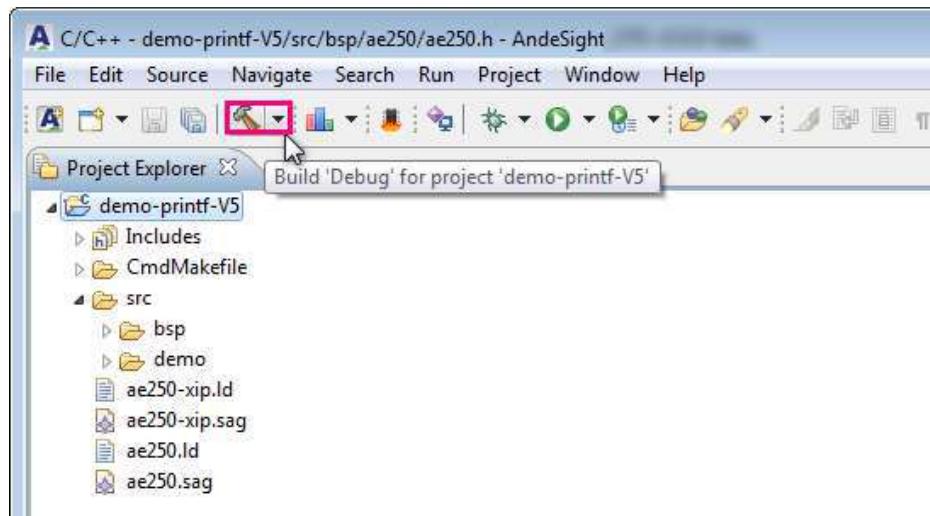
V5 startup demos using a N22 target pre-integrated with PLIC do not support code coverage analysis

---

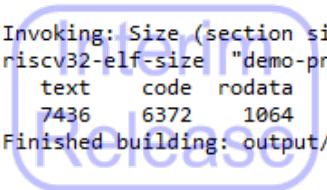


**Step 7** Click “Apply and Close” to complete the configuration.

**Step 8** Click  (Build) on the AndeSight toolbar to build the project.



**Step 9** Verify that the build is successful in the **Console** view.



```
Console X
CDT Build Console [demo-printf-V5]
Invoking: Readelf (ELF info listing)
riscv32-elf-readelf -a "demo-printf-V5.adx" > output/readelf.txt
Finished building: output/readelf.txt

Invoking: Size (section size listing)
riscv32-elf-size "demo-printf-V5.adx" | tee output/.PHONY.size
text code rodata data bss dec hex filename
7436 6372 1064 4 16 7456 1d20 demo-printf-V5.adx
Finished building: output/.PHONY.size

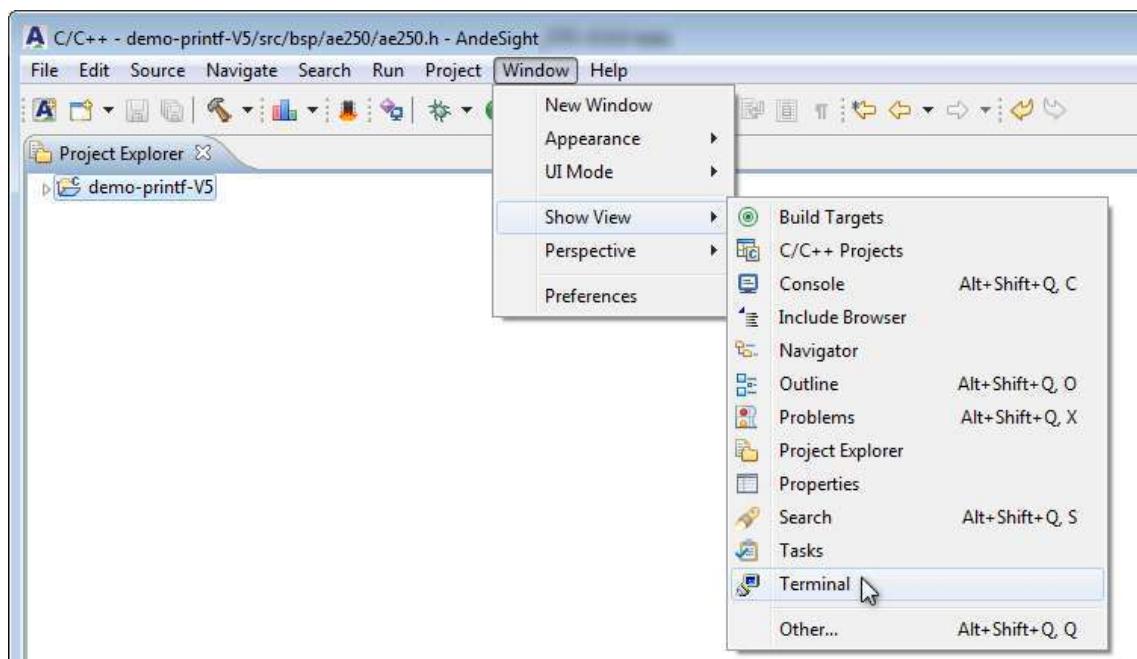
14:01:51 Build Finished. 0 errors, 0 warnings. (took 19s.308ms)
```

### 3.2.1.5 Building terminal connections with ICE targets

Skip this section if you are planning to debug the startup demo on a simulator target.

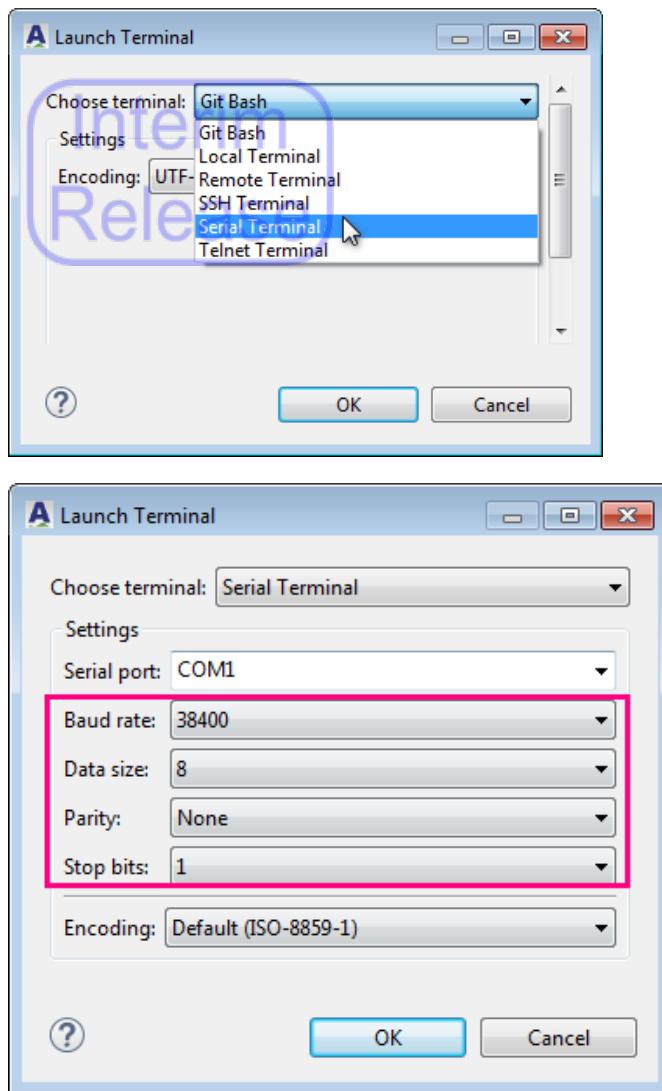
**Step 1** Install the target board and connect it to an ICE device.

**Step 2** From the AndeSight main menu, select “Window > Show View > Terminal” to invoke the **Terminal** view.



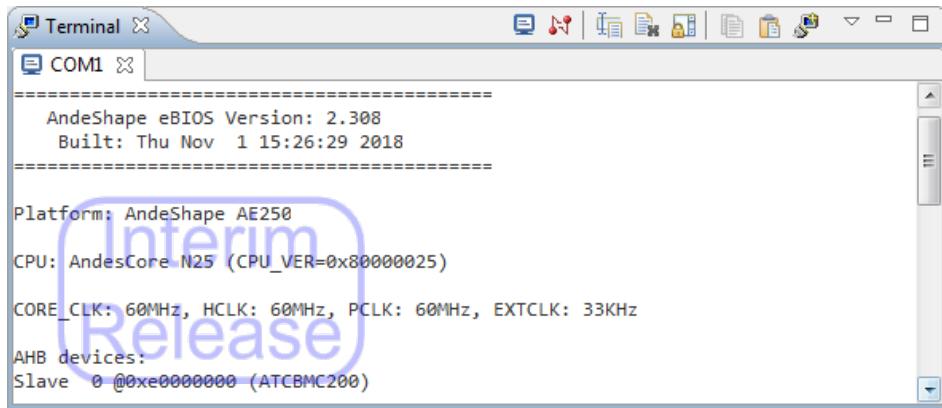
**Step 3** Click  (Open a Terminal) on the toolbar of the **Terminal** view to invoke the **Launch Terminal** wizard. Select “Serial Terminal”, specify

the com port for your target system, and configure port settings as follows:



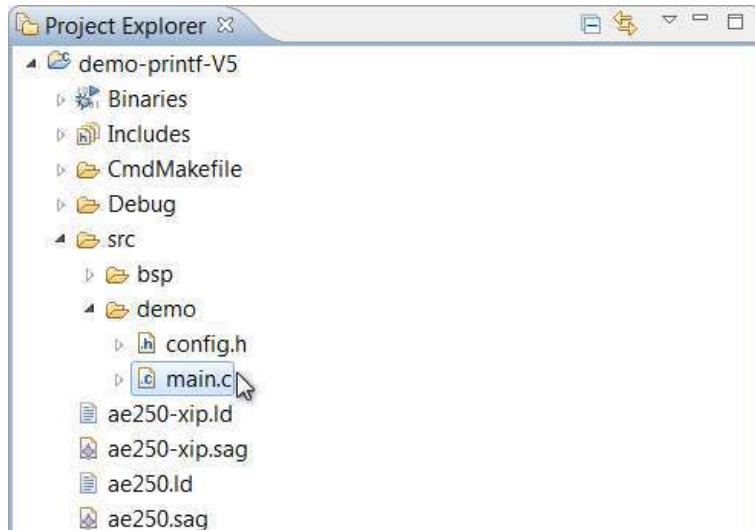
Then, click “OK” to establish the connection.

**Step 4** Toggle on the power switch of the target board and press the “PWR ON” button. The boot message is displayed in the Terminal view.

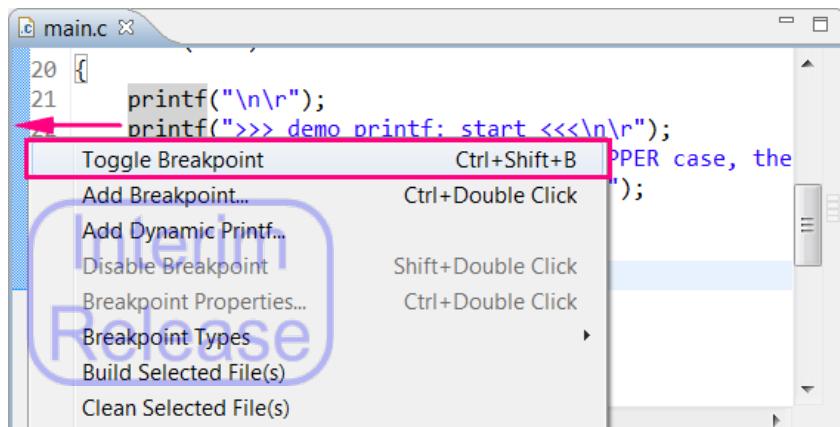


### 3.2.1.6 Setting breakpoint(s) in source files

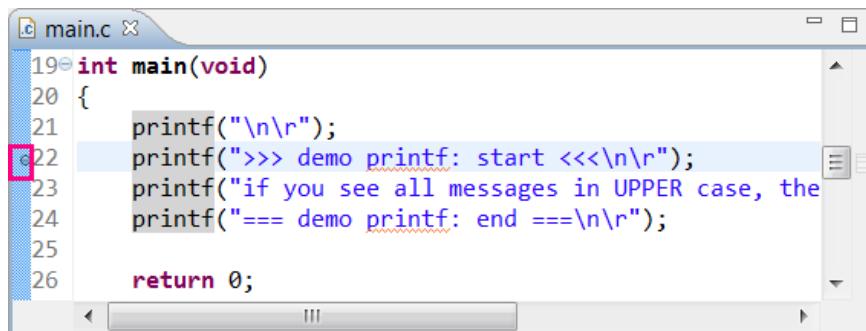
**Step 1** Double-click the source file “`main.c`” under `PROJECT\src\demo` to display its content in the code editor.



**Step 2** Right-click to the left of the line where you want to add the breakpoint, and select “Toggle Breakpoint” from the pull-down menu. You may also configure the breakpoint type by clicking “Breakpoint Types” from the pull-down menu.

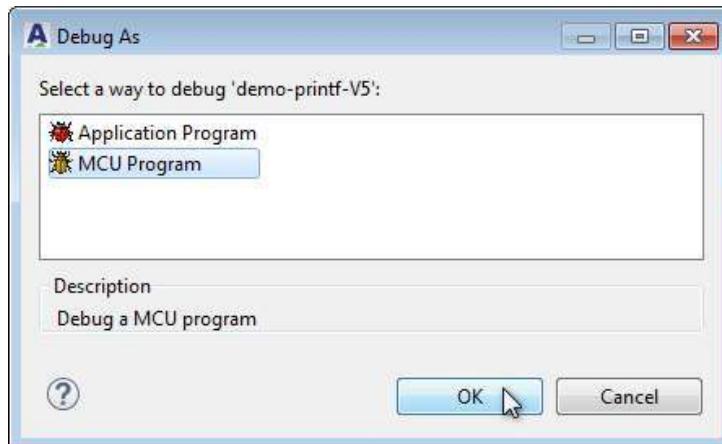


**Step 3** Find the breakpoint marker on the line where you want to suspend the execution of the startup demo program.



### 3.2.1.7 Launching a debug session for startup demo projects

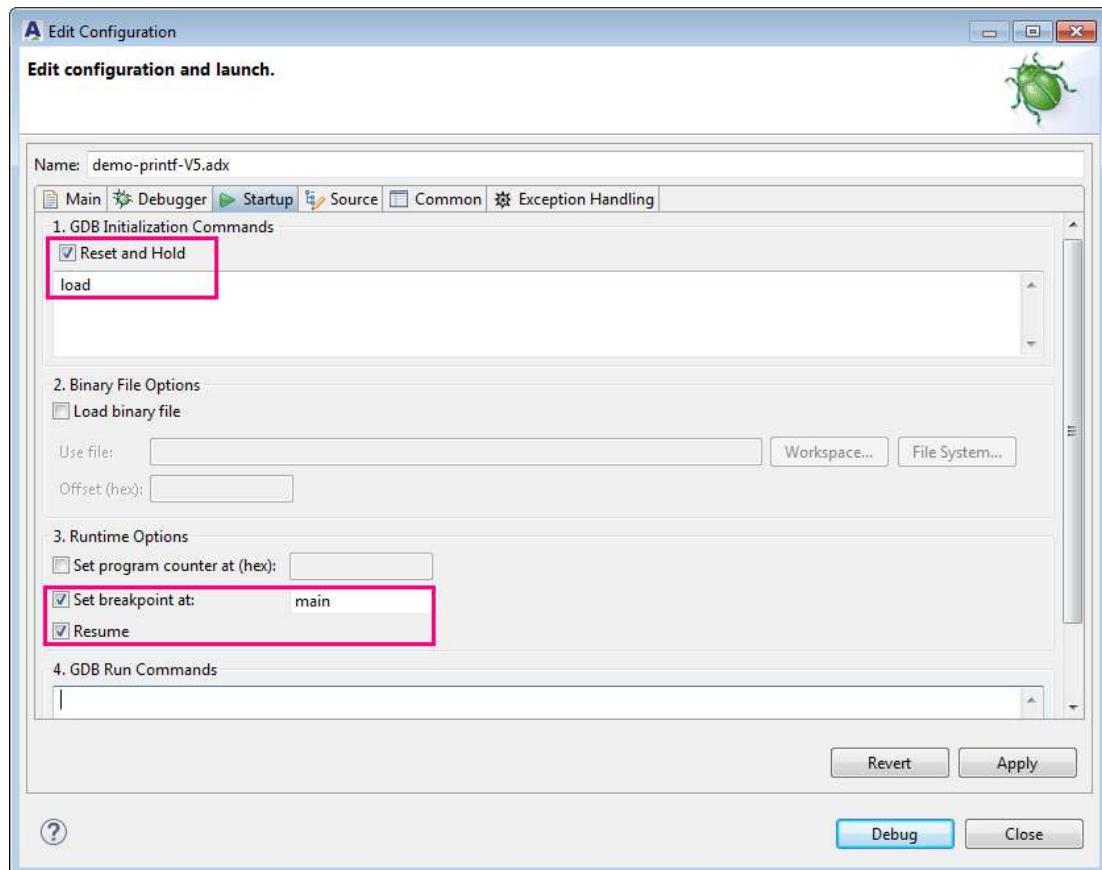
**Step 1** In Project Explorer, select the project and click  (Debug) on the AndeSight toolbar. In the invoked dialog, select the debug configuration “MCU Program” and click “OK”.



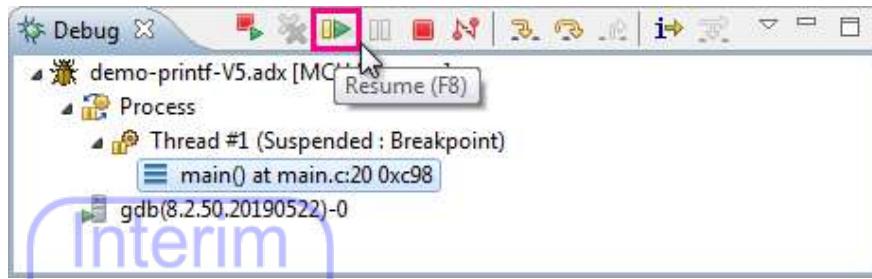
**Step 2** On the invoked **Edit Configuration** dialog, select the Startup tab and configure the settings as follows:

- In the GDB Initialization Commands section, check the “Reset and Hold” option and enter the command “**load**”.
- In the Runtime Options section, set a breakpoint at the **main** function and select the “Resume” option.

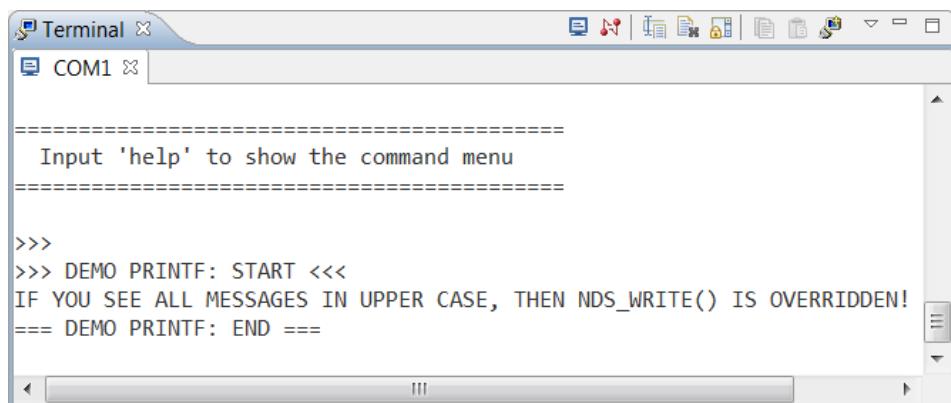
Then, click “Apply” and “Debug” to launch the debug session.



**Step 3** Program execution is suspended at the main function first. Click  (Resume) on the toolbar of the **Debug** view to suspend the program at where your breakpoint is set.

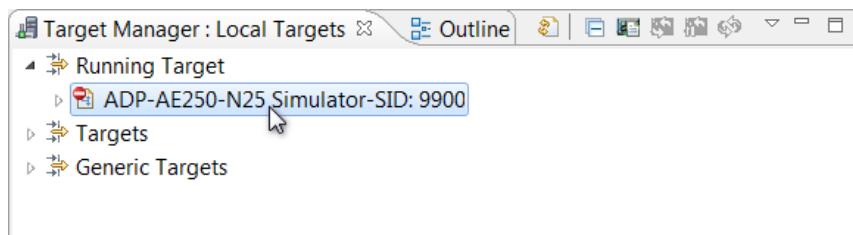


**Step 4** Click  (Resume),  (Step Into), or  (Step Over) on the toolbar of the **Debug** view to execute statements of interest. Check the results in the **Terminal** view if your startup demo is debugged on an ICE target.

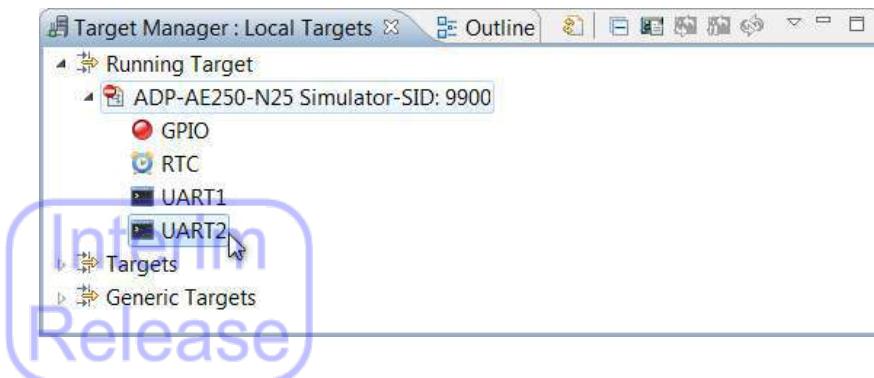


The results of startup projects with a simulator target are displayed in a UART console, which can be invoked as follows:

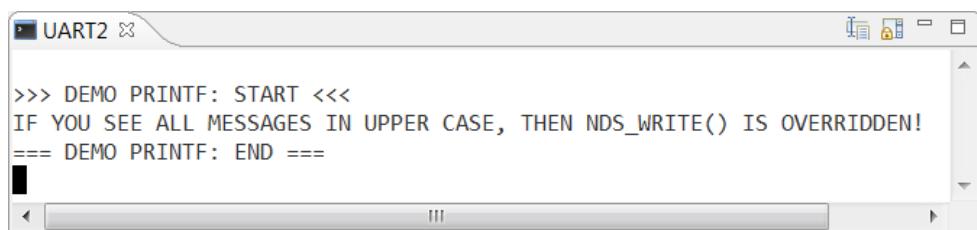
1. Double-click the launched target in the **Target Manager** view and find its front-ends in the expanded tree.



2. Find the pertinent UART and double-click on it.



### 3. Check the results in the invoked **UART** console.



### 3.2.2. Packages of V5 startup demo programs

Tarballs of startup demo programs for V5 targets pre-integrated with PLIC can be found under **ANDESIGHT\_ROOT\demo\startup\[AE250|AE350]\plib** whereas those for V5 targets pre-integrated with CLIC are under **ANDESIGHT\_ROOT\demo\startup\[AE250|Corvette-F1]\clib**. The directories of each startup demo package are organized as follows:

- **CmdMakefile/**

This directory includes Makefile, which is used to make the demo program.

- **src/**

This directory contains the following two sub-directories –

- **bsp/**

This directory contains BSP source files, including **start.S**, for target systems of Andes CPU cores, platforms, and boards.

- **demo/**

This directory contains source files and a configuration file (**config.h**) for this program.

In addition to the files mentioned above, each startup demo application contains SaG files and linker scripts in its root directory. For details about linker scripts for specific platforms and build modes, please refer to Section 3.2.2.2.

The following lists make arguments and **config.h** macros that can be specified for running these demos.

- Make arguments:

Option	Argument	Description
MODE	LOAD	The demo program is loaded to memory and executed in RAM directly.
	BURN	The demo program is burned to flash and stored in ROM/flash before being copied to RAM at startup for execution.
	XIP	“XIP” stands for “eXecute In Place”. In this mode, the demo program is burned to flash and then executed directly in ROM/flash.
SIMU	1	Enables simulation tweaks

Option	Argument	Description
	0	Disables simulation tweaks; default setting
DEBUG	1	Enables debug tweaks (Equal to <code>CFLAGS+= -g -O0</code> )
	0	Disables debug tweaks; default setting
GCOV	1	Enables code coverage
	0	Disables code coverage

■ config.h macros:

- `CFG_SIMU` (equivalent to make argument `SIMU=1`)
- `CFG_GCOV` (equivalent to make argument `GCOV=1`)

In the command line terminal, type “`make COMPILER=riscv[32|64|32-l1vm|64-l1vm]`” under `PROJECT\CmdMakefile\` to compile the source codes of Andes startup demo program and generate the following binary files:

`Demo.elf`: executable file in ELF format

`Demo.bin`: executable file in binary format

`Demo.asm`: disassembly file

To run a startup demo program, you may choose either BURN/XIP (eXecute In Place) or LOAD mode. In BURN/XIP mode, you burn `Demo.bin` to ROM/flash and then reset/power-on the target to run it. In LOAD mode, you load `Demo.elf` to RAM via GDB using Andes ICE and ICEman before running the program.

---

### NOTE

The execution of demo-ecc-V5 or demo-slaveport-V5 requires ILM and DLM, which are disabled by default on AE350. Thus, to run the two programs in LOAD mode, you need to enable the ILM and DLM first. For 32-bit V5 cores, issue “`set $milmb=0x1`” and “`set $mdlmb=0x200001`” in the GDB console as shown below. For 64-bit V5 cores, issue “`set $milmb=0x11`” and “`set $mdlmb=0x2000011`” (i.e., adding a lower case “L” to the above commands) instead.

```
(gdb) target remote :1111
(gdb) set $milmb=0x1
(gdb) set $mdlmb=0x200001
(gdb) l o
```

Loading section .init, size 0x4e lma 0x0  
Loading section .text, size 0x137c lma 0x50  
Loading section .rodata, size 0x2fa lma 0x1400  
Loading section .eh\_frame, size 0x144 lma 0x1700  
Loading section .data, size 0x400 lma 0x1848  
Start address 0xc, Load size 7176  
Transfer rate: 19 KB/sec, 1435 bytes/write.  
(gdb) c

Interim  
Release

### 3.2.2.1 Start.S for V5 startup demos

The file `start. S` in V5 startup demo projects includes the following components:

- trap entry examples,
- vector table for interruptions (vectored external PLIC or CLIC interrupts), and
- low-level initialization for C programs.

The trap entry examples illustrate dispatch handling for trap events, from assembly code to C functions. You may override the `trap_handler()` weak function, depending on your requirements.

The vector table `_vectors` and interrupt dispatch examples show the vector service routine for vectored external PLIC or CLIC interrupts, from assembly code to C functions. You may override any weak function of an interrupt vector service routine depending on your needs. For example, you may use your own handler to replace the vector service routine to the PLIC or CLIC interrupt source 1 by overriding the `entry_irq1()` weak function in `PROJECT\src\PLATFORM\interrupt. c`. Please note that if the PLIC/CLIC device supports a number **N** of interrupt sources, a minimum alignment of " $2^{\lceil \log_2(N) \rceil} \times 4$ " bytes is required for the vector table `_vectors`. In `startup. S`, the number of interrupt sources for PLIC is set to 32 and that for CLIC to 83 by default, so the vector table `_vectors` is aligned to 128 and 512 bytes for PLIC and CLIC respectively. If your PLIC/CLIC supports a different number of interrupt sources, make sure to modify the alignment for the vector table `_vectors` in `start. S`.

In addition, `start. S` also invokes a code segment of low-level initialization for C programs. The following bullets explain the initialization code segment in `start. S`, including the special code sequence, the used symbols, and their meanings:

- Symbol `_global_pointer$`

The instruction sequence relating to `_global_pointer$` is to initialize the global data pointer register `gp (x3)` with the value of `_global_pointer$`. The code is as follows:

```
.option push
.option norelax
la gp, _global_pointer$
.option pop
```

The symbol `__global_pointer$` is the address in the middle of the data sections. The linker places scalar data around to enable easy access using gp-based load/store instructions and the calculation of addresses using gp-based add instructions.

- Symbol `_ITB_BASE`

The instruction sequence related to `_ITB_BASE` is used to initialize the instruction table register `uitb` (user mode CSR) with the value of `_ITB_BASE`. The symbol `_ITB_BASE` is the base address of the instruction table required for the instruction `exec. it`. When the linker performs code size optimization, it automatically assigns the value of `_ITB_BASE`, fills the corresponding table with useful instructions, and generates `exec. it`.

- Symbol `_stack`

The instruction sequence related to `_stack` is used to initialize the stack pointer register `sp (x2)` with the value of `_stack`. The symbol `_stack` is the start address of the stack used by the C compiler to pass function parameters, local variables, and return values. The following code is used to avoid loading the address into `sp` directly:

```
la t0, _stack  
mv sp, t0
```

The linker obtains its value from linker script. The stack proceeds from high addresses to low addresses when performing function calls; therefore, the initial stack address is normally set to the highest address in the program data memory.

### 3.2.2.2 Linker scripts for V5 startup demos

Andes V5 startup demos can be built and executed using either of the following modes:

- XIP mode: boot up and execute on external ROM/flash or ILM
- BURN mode: boot up on ROM/flash and then jump to execute on memory
- LOAD mode: execute on ILM or memory directly.

When running these startup demos, make sure to select a linker script corresponding to the platform in use and the mode specified for the demo application. The execution of the program in each mode along with linker scripts specific to AE250 or AE350 is summarized in the following.

#### XIP mode: Boot up and execute in flash

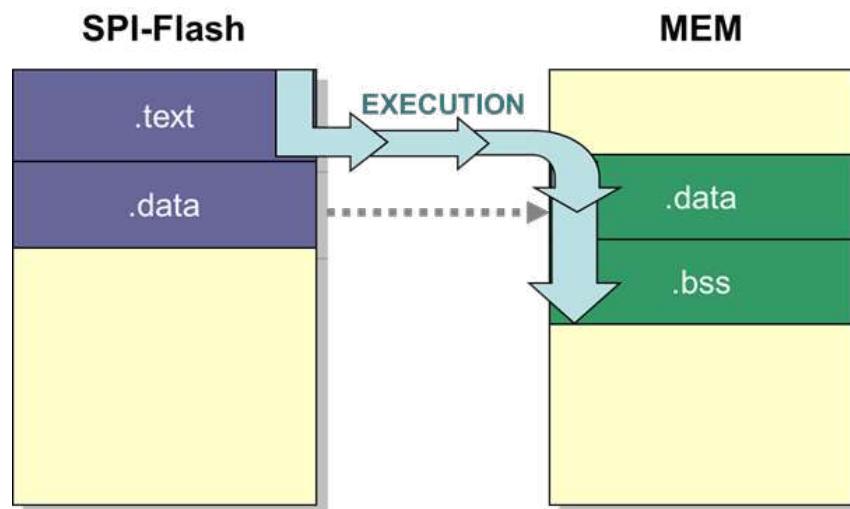


Figure 32. XIP mode: Boot-n-run on flash

The linker scripts used for XIP mode are [ae250-xi p. 1 d](#) on AE250 and [ae350-xi p. 1 d](#) on AE350. They enable booting up and execution in SPI-Flash. As shown in Figure 32, the linker scripts specify [\\_start](#) in the .text section of SPI-Flash as the entry point of the boot code, and copy the .data section from SPI-Flash to MEM during program execution. Note that for XIP mode on AE250, the .data section is copied to DLM.

**BURN mode: Boot up in flash and then jump to execute in MEM**

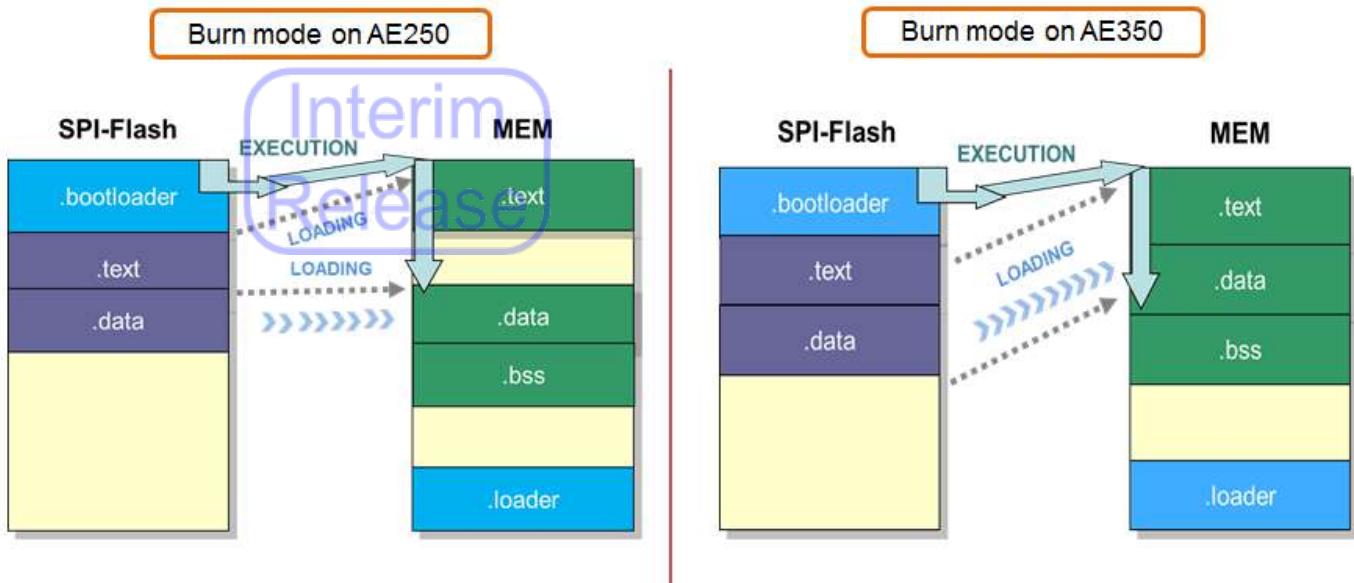


Figure 33. Burn mode: Boot in flash, then load to MEM for execution

The linker scripts for BURN mode are [ae250.1d](#) on AE250 and [ae350.1d](#) on AE350. They enable booting up using SPI-Flash and then jumping to MEM for execution. As shown in Figure 33, the linker scripts specify `boot_loader` in the `.bootloader` section of SPI-Flash as the entry point for boot code and relocate `loader` in the `.loader` section from SPI-Flash to MEM. `loader` is then executed in MEM, and `.text` and `.data` are loaded next from SPI-Flash to MEM for execution. Note that for BURN mode on AE250, `loader` is executed in ILM, and `.text` and `.data` are loaded to ILM and DLM respectively.

#### LOAD mode: Execute in MEM directly

The linker scripts for LOAD mode, same as those used for BURN mode, are [ae250.1d](#) on AE250 and [ae350.1d](#) on AE350. When they are used for LOAD mode, the program is executed directly in MEM without loading `.text` or `.data` sections from ROM to MEM. That is, the `.bootloader` and `.loader` sections in the linker scripts are not applied in this mode. Note that for LOAD mode on AE250, the program is executed in ILM.

### 3.2.3. Description and execution result of each V5 startup demo program

#### 3.2.3.1 PLIC related interrupt

This demo program (demo-plic-V5) demonstrates how to activate the following machine-interrupts and interrupts related to PLIC (Platform-Level Interrupt Controller):

- Machine *mtime* interrupt

The interrupt interval is set to two seconds. In the event that the machine *mtime* interrupt occurs, then the handler sets the machine timer flag and the machine software trigger flag.

- PLIC PIT interrupt

The interrupt interval is set to 200 milliseconds. In the event of a PLIC PIT interrupt, then the handler sets the PLIC PIT timer flag and the machine software trigger flag.

- Machine software interrupt

This interrupt is activated by the machine *mtime* or PLIC PIT interrupt. A UART message is then output periodically (every two seconds/200 milliseconds) to indicate that the machine *mtime* or PLIC PIT is alive.

The main function of the application checks the machine software trigger flag to determine whether to trigger a machine software interrupt.

The output message of this demo application is as follows:

Andes V5 demo program

```
Message triggered from PLIC PIT interrupt
Message triggered from machine time interrupt
Message triggered from PLIC PIT interrupt
Message triggered from PLIC PIT interrupt
```

....

**NOTE**

This demo program (demo-plic-V5) is not supported on N22 targets pre-integrated with PLIC.

### 3.2.3.2 Non-vectored PLIC related interrupt

This demo program (demo-plic-novector-V5) is similar to "demo-plic-V5" except that interrupts activated by this demo are machine and non-vectored external PLIC interrupts.

The output message of this demo application is as follows:

```
Andes V5 demo- plic- novector program
Message triggered from PLIC PIT interrupt
Message triggered from machine time interrupt
Message triggered from PLIC PIT interrupt
Message triggered from PLIC PIT interrupt
...
...
```

### 3.2.3.3 CLIC Related interrupt

This demo program (demo-clic-V5) demonstrates how to activate the following machine interrupts and interrupts related to CLIC (Core-Local Interrupt Controller):

- Machine *mtime* interrupt

The interrupt interval is set to two seconds. In the event that the machine *mtime* interrupt occurs, then the handler sets the machine timer flag and triggers the machine software interrupt.

- CLIC PIT interrupt

The interrupt interval is set to 200 milliseconds. In the event that the CLIC PIT interrupt occurs, then the handler sets the CLIC PIT timer flag and triggers the machine

**software interrupt.**

- Machine software interrupt

This interrupt is activated by the machine *mtime* or CLIC PIT interrupt. A UART message is then output periodically (every two seconds/200 milliseconds) to indicate that the machine timer or CLIC PIT timer is alive.

The machine software handler function of the demo application checks the machine timer or CLIC PIT timer flag to determine whether to output the UART message.

The output message of this demo application is as follows:

Andes V5 demo- cl i c program

Message triggered from CLIC PIT interrupt

Message triggered from machine time interrupt

Message triggered from CLIC PIT interrupt

Message triggered from CLIC PIT interrupt

. . .

#### 3.2.3.4 Low power control

This demo program (demo-wfi-V5) illustrates the use of the Wait-for-Interrupt (WFI) instruction and how to wake up from stalled mode even if interrupts are disabled. This program executes WFI instructions for the core to enter stalled mode and sets an external interrupt in the main function. To wake up a system from stalled mode, simply press a button on the target board. The system will then execute the instruction next to the WFI instruction and wake itself up.

The output message of this demo application is as follows:

Andes V5 demo-wfi program

Entering StandBy mode. . . . .

Wake up from standby mode. . . . .

### 3.2.3.5 Printf function redirection

This demo program (demo-printf-V5) illustrates how to redirect the output to a UART port using the standard `printf()` function in order to overwrite the weak functions `_fstat()` and `_write()`.

The output message of this demo application is as follows:

```
>>> DEMO PRINTE: START <<<
IF YOU SEE ALL MESSAGES IN UPPER CASE, THEN NDS_WRITE() IS OVERRIDDEN!
==> DEMO PRINTE: END ==>
```

### 3.2.3.6 Local memory slave port

This demo program (demo-slaveport-V5) illustrates how to access local memory (ILM/DLM) using a slave port. After checking the local memory configured at the beginning, the program begins moving data from ILM/DLM to DDR and then reading back. During data transfer, the program alternates between CPU and BUS addresses of local memory to show that the slave port is active.

The output message of this demo application is as follows:

```
Andes V5 demo-slaveport program
MCM_CFG = 0x51000
The system has ILM..
MDCM_CFG = 0x51000
The system has DLM..
ILM Size = 0x80000, DLM Size = 0x80000
Move data from ILM to DDR
Checking data... OK
Move data from DLM to DDR
Checking data... OK
Complete
```

---

#### **NOTE**

This demo program (demo-slaveport-V5) is not supported on simulator targets and N22 targets pre-integrated with PLIC.

### 3.2.3.7 Power brake

This demo program (demo-powerbrake-V5) illustrates the dynamic adjustment of CPU performance using 16 throttling levels. After confirming that a CPU is integrated with performance throttles, the program uses a number sequence to test the number of CPU cycles that are consumed when the CPU is throttled down to the lowest performance level or when increased to the highest performance throttles. The results show that the lowest performance throttle results in a greater number of CPU cycles than the highest performance throttle.

The program then configures the CPU to the lowest performance throttle and runs the number sequence again for fast interrupt mode. The number of cycles consumed by the CPU in ISR when fast interrupt mode is disabled exceeds the number of cycles when fast interrupt mode is enabled.

The output message of this demo application is as follows:

Andes V5 demo- powerbrake program

---

[Part 1: Run the number game with LOWEST/HIGHEST performance]

CPU is configured to LOWEST performance (T\_LEVEL: 15)  
 Go number game: 0 --> 9.  
 0...1...2...3...4...5...6...7...8...9...  
 consumed cycles: 31541518.

CPU is configured to HIGHEST performance (T\_LEVEL: 0)  
 Go number game: 0 --> 9.  
 0...1...2...3...4...5...6...7...8...9...  
 consumed cycles: 2470233.

Consumed cycles (LOWEST performance) > Consumed cycles (HIGHEST performance)  
 => OK

---

[Part 2: Run the number game in ISR while ENABLE/DISABLE fast interrupt mode (FAST\_INT) ]

CPU is configured to LOWEST performance (T\_LEVEL: 15)

```
<ENABLE fast interrupt mode>
Go number game in ISR: 0 --> 9.
0...1...2...3...4...5...6...7...8...9...
consumed cycles: 2470223.
```

```
<DI SABLE fast interrupt mode>
Go number game in ISR: 0 --> 9.
0...1...2...3...4...5...6...7...8...9...
consumed cycles: 31541412.
```

Consumed cycles(DI SABLE FAST\_INT) > Consumed cycles(ENABLE FAST\_INT) => OK

demo-powerbrake PASS!

---

## NOTE

This demo program (demo-powerbrake-V5) is not supported on simulator targets and N22 targets pre-integrated with PLIC.

---

### 3.2.3.8 Error checking and correction

This demo program (demo-ecc-V5) illustrates the Andes ECC (Error Checking and Correction) feature. Using this program, the system enables ECC (if the feature is integrated in the target) and tests it sequentially on protected ILM and DLM. Each test generates correctable as well as uncorrectable error exceptions for ECC by injecting single-bit errors (SBE) and double-bit errors (DBE), respectively.

The output message of this demo application is as follows:

Andes V5 demo-ecc program

```
INFO: ECC ILM demo started.
INFO: Initialize and enable ECC checking for ILM
INFO: Injecting SBE ...
INFO: Error injected at 0nxn
INFO: Execute ILM error injected instruction ...
[ILM Parity/ECC Error]
ILM: Correctable error detected!
INFO: Injecting DBE ...
```

INFO: Error injected at 0nxn  
INFO: Execute ILM error injected instruction ...  
[ILM Parity/ECC Error]  
ILM: Uncorrectable error detected!  
INFO: ECC ILM demo succeeded  
  
INFO: ECC DLM demo started.  
INFO: Initialize and enable ECC checking for DLM  
INFO: Injecting SBE ...  
INFO: Error injected at 0nxn  
INFO: Access DLM error injected data ...  
[DLM Load Parity/ECC Error]  
DLM: Correctable error detected!  
INFO: Injecting DBE ...  
INFO: Error injected at 0nxn  
INFO: Access DLM error injected data ...  
[DLM Load Parity/ECC Error]  
DLM: Uncorrectable error detected!  
INFO: ECC DLM demo succeeded

Parity/ECC demo successful

### 3.2.3.9 Hardware stack protection

This demo program (demo-hsp-V5) illustrates hardware stack protection. In the first run of this program, the system moves a HANOI tower using recursion and the hardware stack operating scheme is enabled to record the stack size required for the move. In the second run of the program, the HANOI tower is moved again; however, the stack size is reduced on purpose. Since the hardware stack protection is enabled, the next-precise stack overflow exception is thrown when a stack of insufficient size is detected.

The output message of this demo application is as follows:

Andes V5 demo-hsp program

The initial stack pointer : 0x27ffd0

[HW Stack Recording]

HANOI Tower with 4 disks :

Move disk 1 from A to B

Move disk 2 from A to C  
 Move disk 1 from B to C  
 Move disk 3 from A to B  
 Move disk 1 from C to A  
 Move disk 2 from C to B  
 Move disk 1 from A to B  
 Move disk 4 from A to C  
 Move disk 1 from B to C  
 Move disk 2 from B to A  
 Move disk 1 from C to A  
 Move disk 3 from B to C  
 Move disk 1 from A to B  
 Move disk 2 from A to C  
 Move disk 1 from B to C  
 HANOI (4) = 15 moves  
 Top of Stack : 0x27fa80

[HW Stack Overflow Detection]  
 Set stack top bound : 0x27fc00  
 Retest...  
 HANOI Tower with 4 disks :  
 HSP stack overflow : sp = 0x27fbb0, sp\_bound = 0x27fc00

### 3.2.3.10 Cache operations

This demo program (demo-cache-V5) illustrates runtime code patching on cacheable memory to ensure data coherency. The demo program first executes the `selfModifyCode()` function and calculates the size of patch code in the `dummyTextContext()` function. It then copies the patch code to the address of the `buf` symbol in the `selfModifyCode()` function. As soon as the patch code is copied, the demo code writes back D-Cache to invalidate it. Meanwhile, I-Cache is also invalidated in the `cacheOp()` function to ensure memory coherency. The program then executes the patch code that modifies the value of the global variable `g_selfmodify` within D-Cache and checks the correctness of `g_selfmodify` at the end.

The output message of this demo application on a CPU with L2 cache is as follows:

Andes V5 demo- cache program  
 CPU supports CCTL operation

CPU supports CCTL auto-increment

The L1C ICache sets = 256

The L1C ICache ways = 4

The L1C ICache line size = 32

The L1C DCache sets = 256

The L1C DCache ways = 4

The L1C DCache line size = 32

Enable L1C I cache

Enable L1C D cache

CPU supports L2C cache

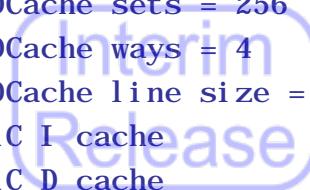
The L2C cache size = 256KB

Execute original self modify code.

I/D cache flush

Run selfModifyCode Pass.

Completed.



The output message of this demo application on a CPU without L2 cache is as follows:

Andes V5 demo-cache program

CPU supports CCTL operation

CPU supports CCTL auto-increment

The L1C ICache sets = 256

The L1C ICache ways = 4

The L1C ICache line size = 32

The L1C DCache sets = 256

The L1C DCache ways = 4

The L1C DCache line size = 32

Enable L1C I cache

Enable L1C D cache

CPU does NOT support L2C cache

Execute original self modify code.

I/D cache flush

Run selfModifyCode Pass.

Completed.

### 3.2.3.11 Cache lock operations

This demo program (demo-cache-lock-V5) demonstrates hardware cache lock mechanism with CCTL commands.

This demo program starts with a code patching on an unlocked cache and then performs a code patching on a locked cache. In the unlock scenario, the main function checks the

cache feature and enable I/D cache if the feature is available. The program then runs the patch code “A” to modify the value of a global variable named `g_selfmodify` and uses CCTL commands to ensure memory coherency while the patch code is unlocked in the cache. Next, it runs NOP instructions with a size of 64K to flush the patch code “A” out of the I cache and proceed with another patch code “B” that also modifies `g_selfmodify`. The value of `g_selfmodify` is examined in the end to verify if the unlocked cache works.

The program follows to run the cache lock scenario where the patch code “A” is executed and locked into I cache. Similarly, the patch code “A” is flushed out with 64K NOP instructions and another patch code “B” is executed to modify `g_selfmodify`. The value of `g_selfmodify` is examined at last to verify if the locked cache works.

The output message of this demo application is as follows:

Andes V5 demo-cache-lock program  
CPU supports cache lock feature.

=====demo cache unlock=====

Run unlock self modify code, expect `g_selfmodify=8888888`. . . . .

Run self modify code to set `g_selfmodify = 7777777`. . . . .

Run 64k NOP buf to full-fill Icache to flush the unlocked Icache line of self modify code. . . . .

Run self modify code to set `g_selfmodify = 8888888`. . . . .

The `g_selfmodify = 8888888`

Run cache unlock scenario PASS.

=====demo cache lock=====

Run lock self modify code, expect `g_selfmodify=7777777`. . . . .

Run self modify code to set `g_selfmodify = 7777777`. . . . .

Run 64k NOP buf to full-fill Icache to flush the unlocked Icache line of self modify code. . . . .

Run self modify code to set `g_selfmodify = 8888888`. . . . .

The `g_selfmodify = 7777777`

Run cache lock scenario PASS.

Compl eted.

---

## NOTE

This demo program (demo-cache-lock-V5) is not supported on simulator targets.

---

### 3.2.3.12 Performance monitor

This demo program (demo-pfm-V5) illustrates the use of basic profiling counters, “mcycle” (the number of processor cycles) and “minstret” (the number of retired instructions). A simple factorial program is run twice to measure performance. In the first run, the mcycle and minstret counters at the end are compared with those at the beginning, and the difference between the counters is used to evaluate performance. In the second run, the two counters at the beginning are both cleared and their values at the end of the program are used to evaluate performance.

The output message of this demo application is as follows:

Andes V5 demo-pfm program

Run factorial (100) mathematics ...

Demo 1: Using Counter Differences.

Loop 0: Retired 416 instructions in 451 cycles

Loop 1: Retired 416 instructions in 429 cycles

Loop 2: Retired 416 instructions in 426 cycles

Demo 2: Clearing Counters, Using Values Directly.

Loop 0: Retired 415 instructions in 450 cycles

Loop 1: Retired 415 instructions in 441 cycles

Loop 2: Retired 415 instructions in 441 cycles

### 3.2.3.13 Physical memory protection

This demo program (demo-pmp-V5) shows how to protect physical memory accesses with proper privilege mode. It first initializes a PA (physical address) memory region with a PMP (physical memory protection) configuration, which allows memory accesses only in machine mode. Next, the program runs in user mode to perform the following tasks: storing a word to a PA region that allows accesses either in machine or user mode, storing a word to a region that allows accesses only in machine mode, and fetching the code of the trap handler from a region which requires machine mode. The first task succeeds whereas the latter two fail due to the lack of a write/execute privilege and trigger a store/instruction access exception.

The output message of this demo application on a CPU implemented with RISC-V

N-extension is as follows:

PMP uses NAPOT scheme!.

Andes V5 demo-pmp program runs in user mode

Try to store a word to a region allowing in user or machine mode.

You should see the [Store Pass] message ...

[Store Pass]

Next, try to store a word to a region requiring in machine mode.

You should see the [Data Store Access Fault] message ...

[N-Extension U-Mode: Data Store Access Fault]

Finally, try to execute machine mode code.

You should see the [Instruction Fetch Fault] message ...

[N-Extension U-Mode: Instruction Fetch Fault]

PMP-Pass!

The output message of this demo application on a CPU implemented without RISC-V

N-extension is as follows:

PMP uses NAPOT scheme!.

Andes V5 demo-pmp program runs in user mode

Try to store a word to a region allowing in user or machine mode.

You should see the [Store Pass] message ...

[Store Pass]

Next, try to store a word to a region requiring in machine mode.

You should see the [Data Store Access Fault] message ...

[U-Mode: Data Store Access Fault]

Finally, try to execute machine mode code.

You should see the [Instruction Fetch Fault] message ...

[U-Mode: Instruction Fetch Fault]

PMP-Pass!

## NOTE

This demo program (demo-pmp-V5) is not supported on target boards booted from flash.

### 3.2.3.14 Memory management for VA to PA translation

This demo program (demo-mmu-V5) shows the use of a PTE (page table entry) table to translate virtual addresses (VA) to physical addresses (PA). It first initializes a PTE table which stores the VA=PA mapping for programs in machine mode and the VA!=PA mapping for programs in supervisor or user mode. Next, the program sets SV32(RV32)/SV48(RV64) to turn on the translation mechanism, switches to run in

supervisor and in user mode, and causes page fault exceptions after trying to store an unregistered VA. It then dynamically stores the mapping between the VA and a dedicated PA into the PTE table and tries to store the same VA again. This time, it accesses the VA successfully with a corresponding entry in the PTE table.

The output message of this demo application is as follows:

setup PTE table!

Andes V5 demo-mmu program

Try to store a word to a supervisor VA without MMU mapping.

You should see the [Supervisor: Data Store Page Fault] message ...

[Supervisor: Data Store Page Fault]

Supervisor: Data Store Success

Try to store a word to a user VA without MMU mapping.

You should see the [User: Data Store Page Fault] message ...

[User: Data Store Page Fault]

User: Data Store Success

MMU-Pass!

## NOTE

This demo program (demo-mmu-V5) is restricted to AndesCore A-prefixed series (i.e., A25/AX25/A27/AX27/A45/AX45) with the AE350 netlist that has the MMU feature. In addition, it is not supported on target boards booted from flash.

### 3.2.3.15 DSP library

This demo program (demo-dsp-V5) illustrates the use of DSP library functions. To call specific library functions, you should include the corresponding header files in `main.c`. For the header file specific to each function category, please refer to *Andes DSP Library V5 User Manual*.

The output message of this demo application is as follows:

Andes V5 demo-dsp program

-----  
after CFFT\_RD2, maxdiff= 0x0080 [0.00390816]

-----  
MAE is 0.00079375, RMSD is 0.00176055, NRMSD is 0.00005502, MAXDIFF is 0.00390816, SNR is 67.12823486

CFFT\_RD2 out scaled up by 64

```
after CI FFT_RD2, maxdiff= 0x0009 [0. 00029564]
```

```
-----  
MAE is 0. 00012192, RMSD is 0. 00014900, NRMSD is 0. 00007450, MAXDIFF is  
0. 00029564, SNR is 73. 52618408
```

```
CI FFT_RD2 out scaleup by 2
```

```
CI FFT_RD2 PASS
```

### 3.2.3.16 CPU entering SLEEP mode and waking up

This demo program (demo-hibernate-V5) demonstrates the working of CPU status registers in a power-saving mechanism. It simulates a power-saving mechanism in which the system enters SLEEP mode after a trigger event (GPIO) occurs and wakes up after a wakeup event (Timer). CPU status registers are saved to and restored from storage area during this process.

At the start of this program, the system displays messages continuously. When GPIO buttons are pressed, the GPIO ISR saves the current CPU status and triggers the system to enter SLEEP mode. This suspends the program, such that all messages are frozen. A timer is set to wake up the system after four seconds. When this period has elapsed, the system wakes up, executes Timer ISR, and restores the CPU status, whereupon the program resumes displaying messages.

The output message of this demo application is as follows:

#### Sleep and Wakeup example

```
A  
BA  
CBA  
DCBA  
EDCBA  
FEDCBA  
GFEDCBA  
HGFEDCBA  
I HGFEDCBA  
JI HGFEDCBA  
JI HGFEDCBA  
I HGFEDCBA  
HGFEDCBA  
GFEDCBA
```

FEDCBA

EDCBA

DCBA

CBA

BA

A

0

10

210

3210

43210

543210

6543210

76543210

876543210

9876543210

9876543210

876543210

76543210

6543210

543210

43210

3210

210

10

0

..

..



---

**NOTE**

This demo program (demo-hibernate-V5) is not supported on N22 targets pre-integrated with PLIC.

---

### 3.2.3.17 C++ programming

This demo program (demo-cplusplus-V5) demonstrates how to implement a static C++ program. When compiling C++ code with static objects, the compiler inserts a call to `__cxa_atexit()` with `__dso_handle` as one of its arguments. This program provides a

---

dummy version of these symbols for your reference.

In this demo program, the main function creates a C++ object and uses `cout` from the `iostream` library to output object messages including constructor and destructor ones. The `cout` function retargets functions for `printf()` and redirects messages to a UART port.

The output message of this demo application is as follows:

Andes V5 C++ demo program  
I am Object constructor  
Length of line : 6.000000  
I am Object destructor

### 3.2.3.18 Symmetric multiprocessing

This demo program (demo-smp-V5) demonstrates activation of machine time, machine software and PIT related interrupts in a dual-core SMP system as well as interaction between cores through IPI (Inter-Processor Interrupt) mechanism. During the bootstrap, the main core (mhart0) handles platform initialization tasks like relocating data section, clearing bss section, and initializing UART and PLIC.

In this demo program, the main function checks the core IDs (mhart IDs) and performs core-by-core initialization operations for interrupts like machine time, machine software and PIT related interrupts. The machine software interrupt on the mhart0 core is activated to output machine timer alive message every 2 seconds with IPI notification from the mhart1 core. On the other hand, the machine software interrupt on the mhart1 core is also activated to output PIT timer alive message every 0.2 seconds with IPI notification from the mhart0 core.

The output message of this demo application is as follows:

Andes V5 demo-smp program  
[Mhart1]: Message triggered from PLIC PIT interrupt  
[Mhart1]: Message triggered from PLIC PIT interrupt

[Mhart1]: Message triggered from PLIC PIT interrupt  
[Mhart1]: Message triggered from PLIC PIT interrupt  
[Mhart0]: Message triggered from machine time interrupt  
[Mhart1]: Message triggered from PLIC PIT interrupt  
[Mhart0]: Message triggered from machine time interrupt

**NOTE**

This demo program (demo-smp-V5) is not supported on simulator targets.

**3.2.3.19 Physical memory attribute**

This demo program (demo-pma-V5) demonstrates how to use PMA to configure memory attributes. It first checks whether the PMA feature is supported, then initializes the PMA configuration of three memory regions by setting the A and B regions as non-cacheable and the C region as cacheable, and turns on the cache. Next, it performs the following operations in sequence:

1. It reads the A region with non-cacheable attribute, then changes the memory attribute to cacheable and reads the region again. The consumed cycles for the A region with non-cacheable and cacheable attributes are measured respectively and compared.
2. It reads the A region with cacheable attribute and the B region with non-cacheable attribute, and compares the consumed cycles for the two regions that have different cacheability attributes.
3. It reads the C region with cacheable attribute, then flushes the cache and changes the memory attribute to non-cacheable. The consumed cycles for the C region with cacheable and non-cacheable attributes are measured respectively and compared.

The output message of this demo application is as follows:

Andes V5 demo- pma program.

[Reconfigurable PMA]: Try to read A region from non-cacheable memory.  
consumed cycles: 661360.

[Background PMA]: Try to read A region from cacheable memory.  
consumed cycles: 147465.

PMA-Pass of the same region A with cacheable and non-cacheable attribute!

[Reconfigurable PMA]: Try to read B region from non-cacheable memory.  
consumed cycles: 661322.

PMA-Pass of the different regions A and B with cacheable and non-cacheable attribute!

[Reconfigurable PMA]: Try to read C region from cacheable memory.  
consumed cycles: 147465.

[Reconfigurable PMA]: Try to read C region from non-cacheable memory.  
consumed cycles: 661320.

PMA-Pass of the same region C with cacheable and non-cacheable attribute!

---

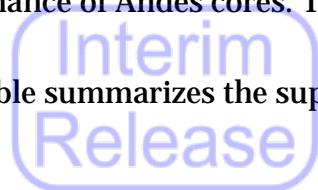
## NOTE

This demo program (demo-pma-V5) is currently supported on ICE/simulator targets of A(X)27 CPU cores or ICE targets of 45 series CPU cores .

---

### 3.3. Dhystone, CoreMark and Whetstone

Dhystone, CoreMark and Whetstone are benchmark applications that can be used to measure the CPU performance of Andes cores. These demos are placed under [ANDESI GHT\\_ROOT\demo\](#).



The following table summarizes the supported platforms and project settings for these demo programs.

Table 10. Verified platforms and project settings for Dhystone, CoreMark and Whetstone programs

Project	Target platform			Compatible project setting				
	AE250	Corvette-F1	AE350	Project Type	CPU Core	Endian	Library	ISA
<b>Dhystone</b>	√	√	√	AndeSight Project/ Makefile C Project	32/64 bit	little	newlib/ mcullib	v5/ v5e
<b>CoreMark</b>	√	√	√					
<b>Whetstone</b>	√		√					

√: Supported

#### NOTE

1. A simple implementation of `my_malloc()` and `scanf()` could be found in Dhystone project. If you don't require these two functions for your programs, then simply comment them in `utils.c` and `config.h`. Then, `malloc()` and `scanf()` in the Andes library are used for the project; however, this increases the code size.
2. If compiler option “`-ffunction-sections -fdata-sections`” is applied for reduction of code size, then make sure your functions are called (directly or indirectly) by the main function. Otherwise, unused functions will not be included in the generated binary.

### 3.3.1. Running Dhystone-V5, CoreMark-V5 or Whetstone-V5 demos

The following build configurations addressing different platforms and memory sizes are provided for these benchmark programs:

- **AE250\_Release** and **AE250\_LLVM\_Release**: for CPU cores with more than 512KB ILM and 512KB DLM on ADP-XC7 targets of AE250 platform
- **AE350\_Release** and **AE350\_LLVM\_Release**: for CPU cores with more than 512KB ILM and 512KB DLM on ADP-XC7 targets of AE350 platform
- **CF1-AE250\_Release** and **CF1-AE250\_LLVM\_Release**: for CPU cores with more than 128KB ILM and 128KB DLM on Corvette targets of F1 platform

The availability of build configurations for respective demo and the compiler to be used with are as follows:

Build configurations	Dhystone-V5	CoreMark-V5	Whetstone-V5	Compiler to be used with
AE250_Release	V	V	V	• For Dhystone-V5 and Whetstone-V5: GCC or LLVM
AE350_Release	V	V	V	
CF1-AE250_Release	V	V		• For CoreMark-V5: GCC
AE250_LLVM_Release		V		LLVM
AE350_LLVM_Release		V		
CF1-AE250_LLVM_Release		V		

To run the Dhystone-V5/CoreMark-V5/Whetstone-V5 project, follows the instructions below:

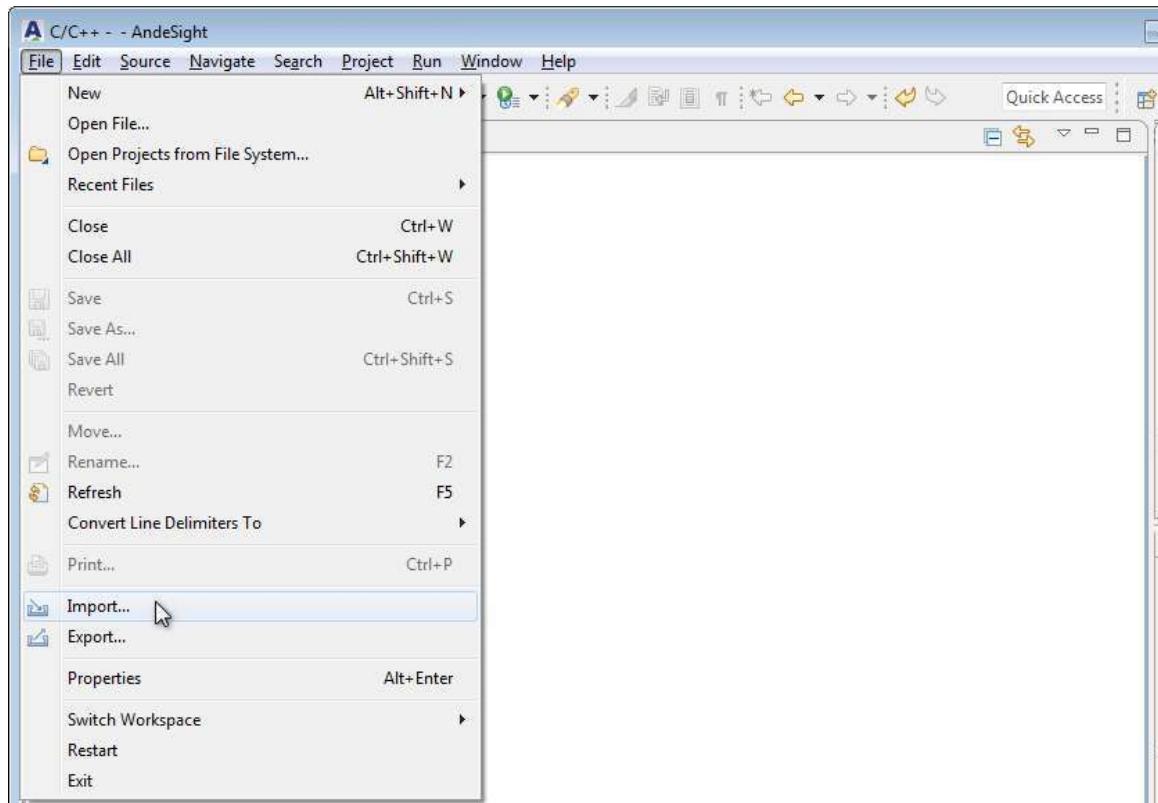
The three demo applications are executed in a similar manner. The Dhystone-V5 project is used as an example in the following subsections for step-by-step instructions on running the three demo applications. The operation focuses on the following:

- How to import an existing project
- How to specify a build configuration and a target configuration

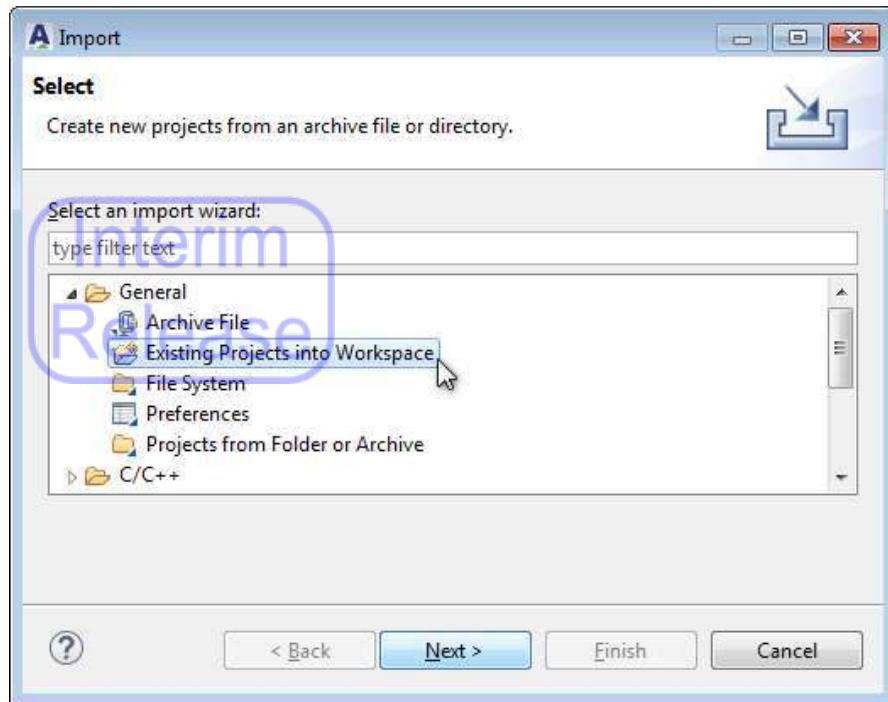
- How to configure build settings
- How to build a project
- How to establish a terminal connection to an Andes ICE target
- How to run a project

### 3.3.1.1 Importing a demo project

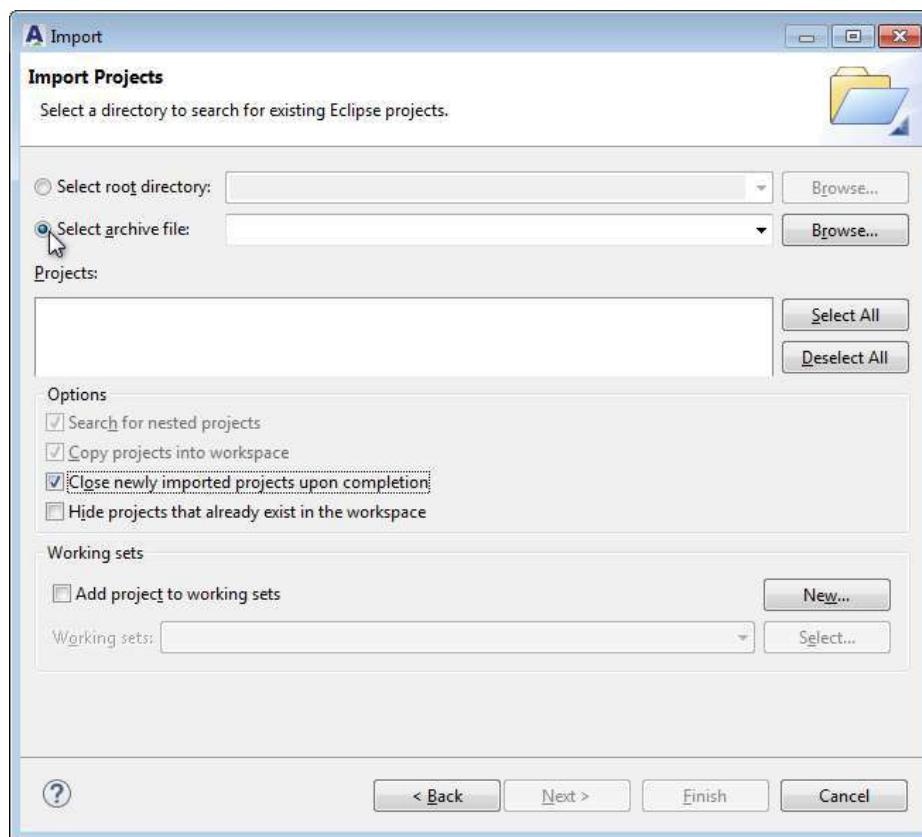
**Step 1** On the AndeSight main menu, click “File” and select “Import...” from the pull-down menu.



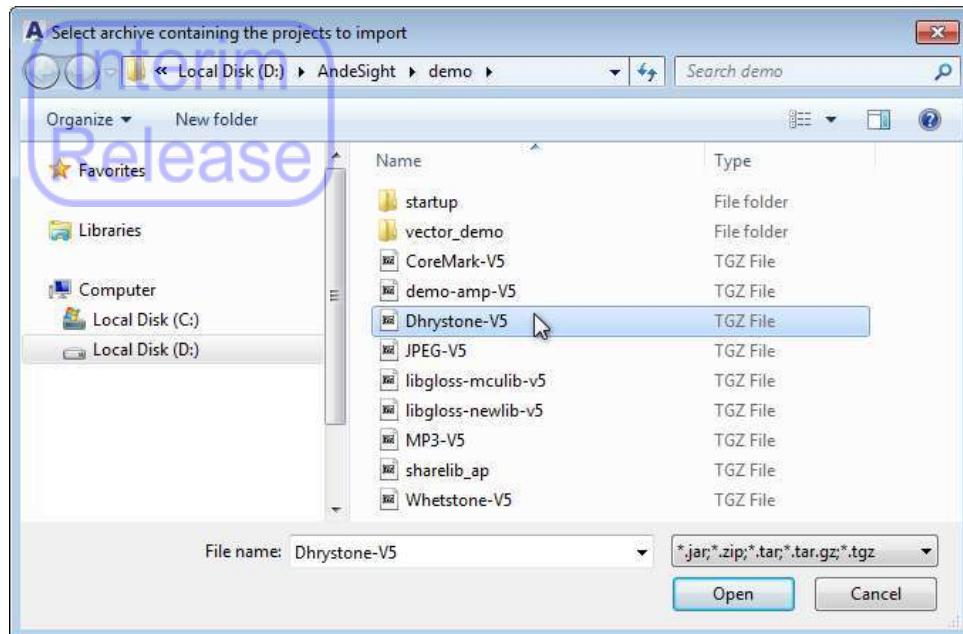
**Step 2** In the invoked **Import** dialog box, select “General > Existing Projects into Workspace” and click “Next.”



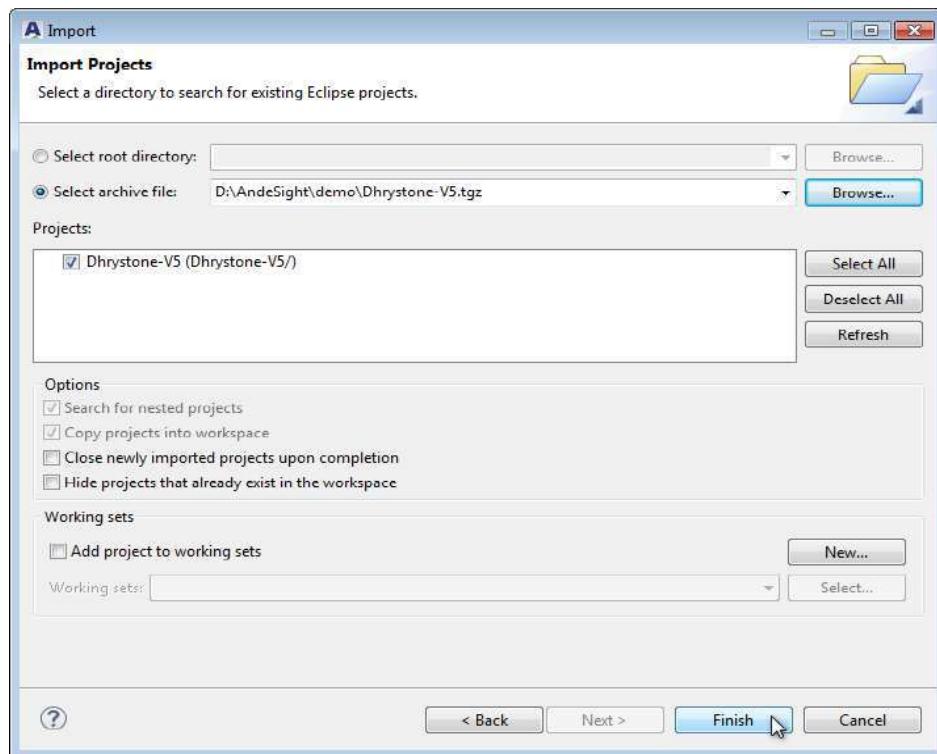
**Step 3** Dhystone-V5, CoreMark-V5 and Whetstone-V5 demos are provided as tarballs in AndeSight; therefore, you have to check the “Select archive file” radio box to import the compressed file.



**Step 4** In the invoked wizard, click “Browse...” to find the Dhrystone, CoreMark or Whetstone project ([Dhrystone- V5. tgz](#), [CoreMark- V5. tgz](#) or [Whetstone- V5. tgz](#)) under **ANDESIGHT\_ROOT\demo\** and click “Open.”



**Step 5** In the **Import** dialog, click “Finish” to complete importing the project.



**Step 6** In the **Project Explorer** view, find the imported demo project.

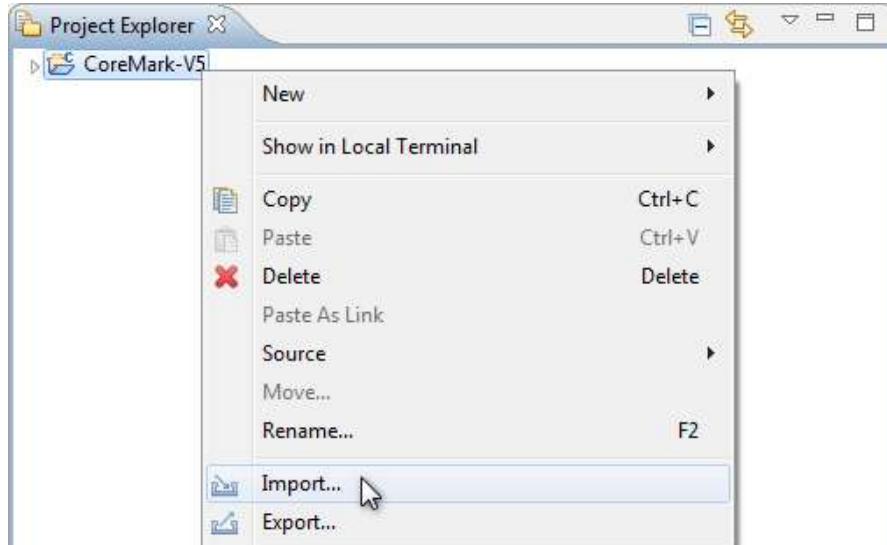


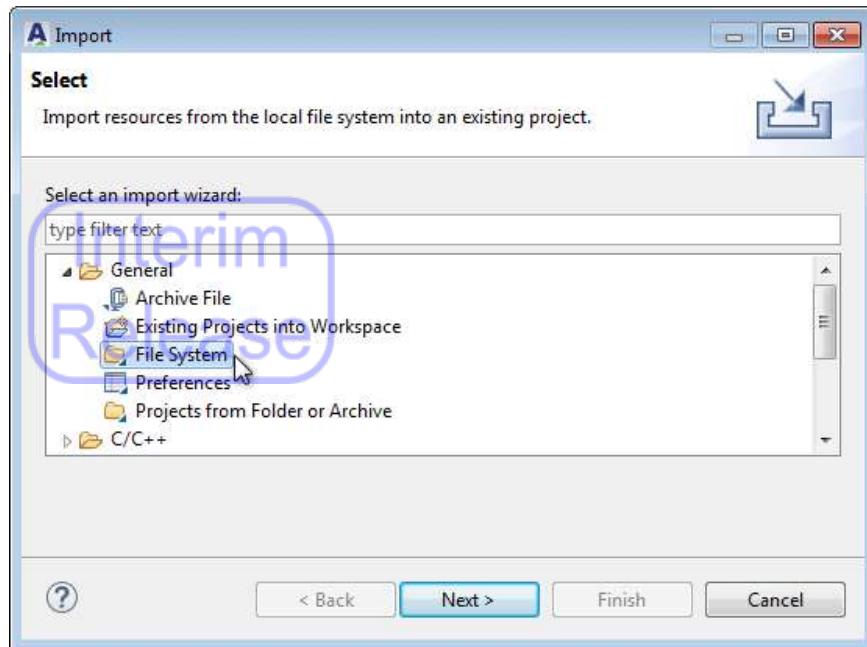
### 3.3.1.2 Porting benchmark software on Andes platforms (for CoreMark-V5 projects only)

Skip this section if you want to run a Dhrystone-V5 or Whetstone-V5 demo.

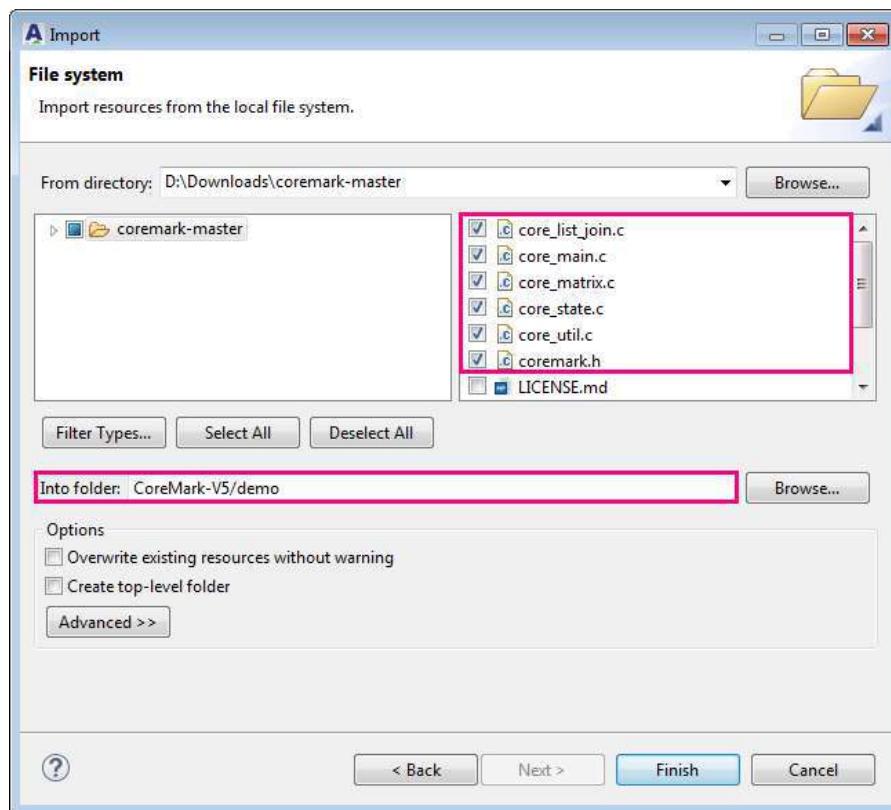
**Step 1** Download the CoreMark software package `coremark_master.zip` from the EEMBC github page (<https://github.com/eembc/CoreMark>) and un-compress it to create the folder `coremark_master`.

**Step 2** Return to AndeSight. In **Project Explorer**, right-click the imported CoreMark project and select “Import” from the pull-down menu. In the invoked dialog, select “General > File System” and click “Next.”



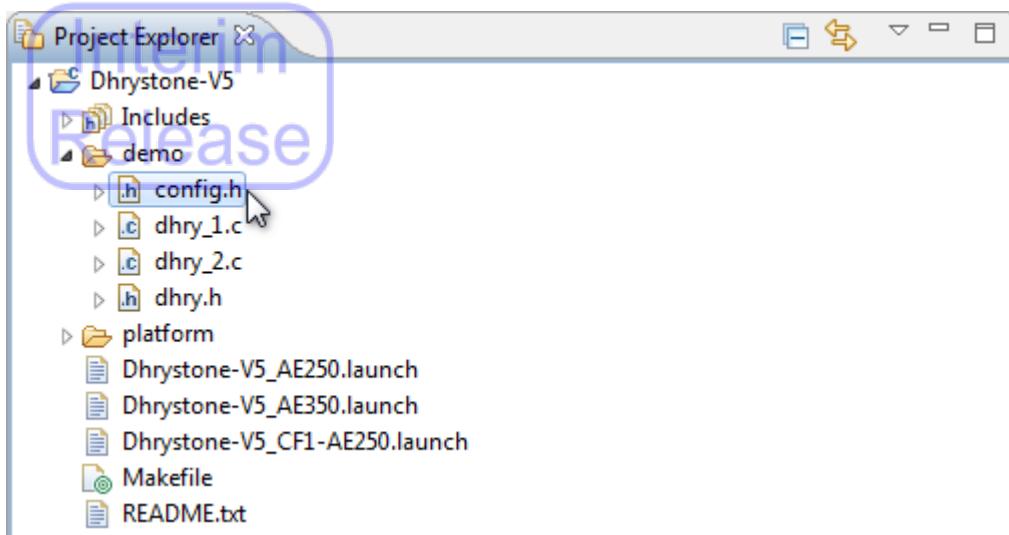


**Step 3** Click “Browse...” to search for the `coremark_master` folder. Select `coremark.h`, `core_list_join.c`, `core_main.c`, `core_matrix.c`, `core_state.c` and `core_util.c` and click “Finish” to import the files into `CoreMark-V5\demo`.



### 3.3.1.3 Modifying configuration files

**Step 1** In Project Explorer, double-click `config.h` under `PROJECT\demo` to display its content in the editor.



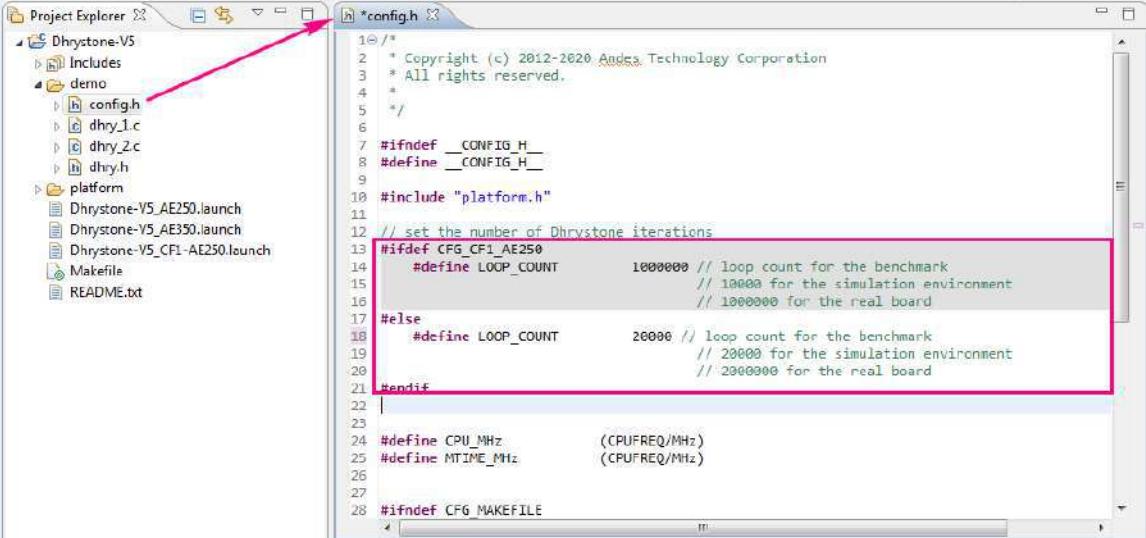
**Step 2** specify the following desired configuration if it's required, and press “**Ctrl + S**” to save the changes:

- **CFG\_VH**: Uncomment this macro if you want to enable the Virtual Hosting support for the project. Note that the support also requires a corresponding setting in the project's build configuration. Please reference Section 3.3.1.4, Step 7 to specify the Virtual Hosting option.
- **CFG\_SIMU**: For the Whetstone-V5 project on a simulator target only, uncomment this macro to reduce the iteration times and accelerate the simulation speed.
- **CFG\_CACHE\_ON**: Uncomment this macro if you want to enable cache.
- **LOOP\_COUNT/ITERATIONS**: To obtain valid benchmark results, the Dhystone-V5 and Whetstone-V5 projects need to be iterated for at least 2 seconds and the CoreMark-V5 project needs iterations of 10 seconds at minimum. For the Whetstone-V5 project, its calibration has ensured iterations of more than 2 seconds. For the other two projects, modify the **LOOP\_COUNT/ITERATIONS** value to

enable sufficient iterations:

- Dhrystone-V5 project on a target of Corvette-F1 platform:  
Change the **LOOP\_COUNT** value to **10000** for a simulator target or **1000000** for an ICE target.
- Dhrystone-V5 project on a target of ADP-[AE250|AE350] platform:  
Change the **LOOP\_COUNT** value to **20000** for a simulator target or **2000000** for an ICE target.
- CoreMark-V5 project on a target of ADP-[AE250|ADP-AE350|Corvette-F1] platform: Change the **ITERATIONS** value to **350** for a simulator target or **3500** for an ICE target.

The following **config.h** illustrates the **LOOP\_COUNT** values for running Dhrystone-V5 on a V5 simulator target.

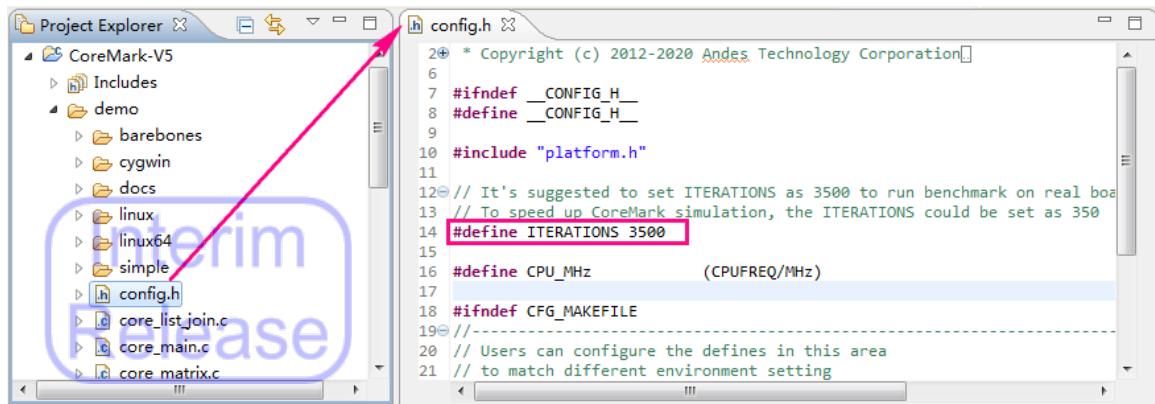


```

1 /* 
2  * Copyright (c) 2012-2020 Andes Technology Corporation
3  * All rights reserved.
4  */
5
6
7 #ifndef __CONFIG_H__
8 #define __CONFIG_H__
9
10 #include "platform.h"
11
12 // set the number of Dhrystone iterations
13 #ifdef CFG_CF1_AE250
14     #define LOOP_COUNT      1000000 // loop count for the benchmark
15                                // 10000 for the simulation environment
16                                // 1000000 for the real board
17 #else
18     #define LOOP_COUNT      20000 // loop count for the benchmark
19                                // 20000 for the simulation environment
20                                // 2000000 for the real board
21 #endif
22
23
24 #define CPU_MHz          ((CPUFREQ/MHz))
25 #define MTIME_MHz        ((CPUFREQ/MHz))
26
27
28 #ifndef CF6_MAKEFILE

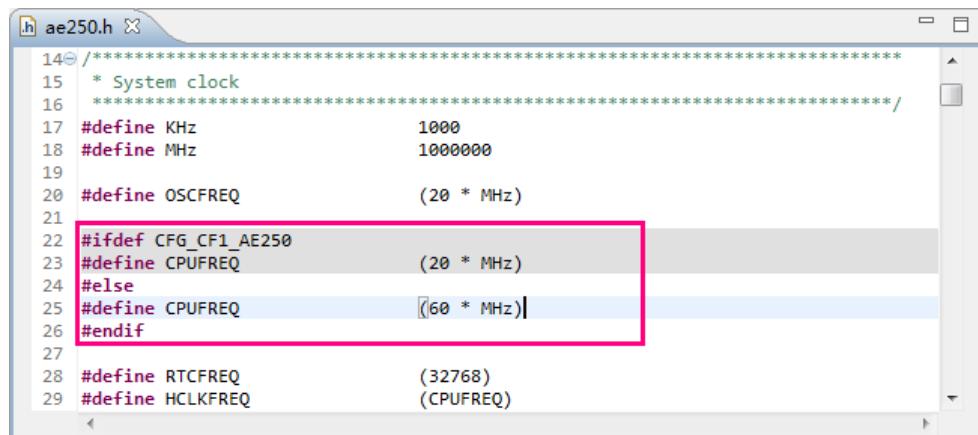
```

The **config.h** example below shows the **ITERATIONS** values for running CoreMark-V5 on a V5 ICE target.



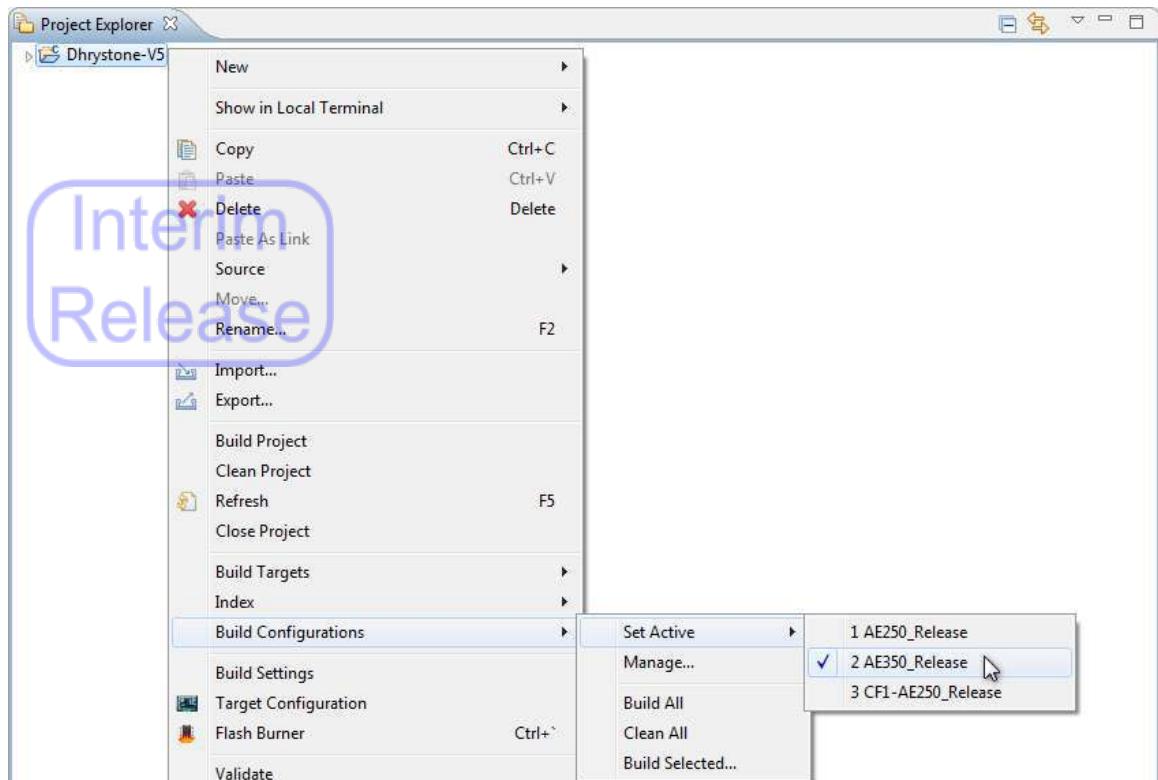
**Step 3** The required CPU frequency is defined by **CPUFREQ** in the corresponding platform header file (i.e. **ae250.h** and **ae350.h**) under **PROJECT\platform\[ae250|ae350]**. Make sure the **CPUFREQ** value matches the CPU speed defined in the data sheet of the target board, as listed below:

- For Corvette-F1 targets: The **CPUFREQ** value must be **20** MHz.
- For AE250 targets: The **CPUFREQ** value must be **20** MHz for N22 and **60** MHz for other CPU cores.
- For AE350 targets: The **CPUFREQ** value must be **60** MHz.

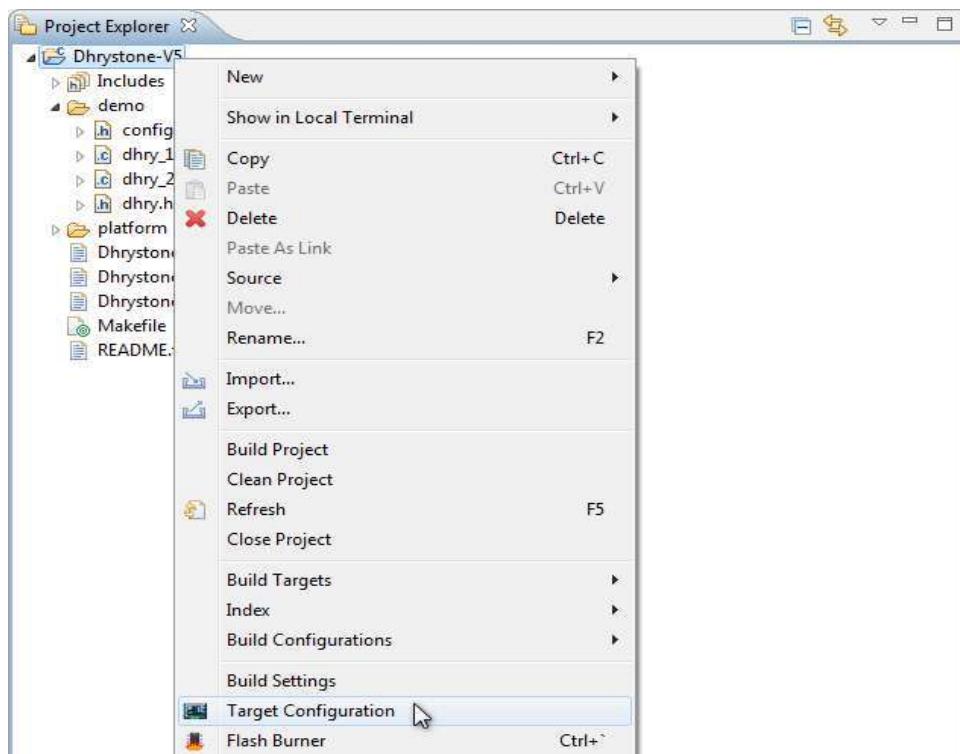


### 3.3.1.4 Specifying build configurations and target configurations and building the program

**Step 1** In Project Explorer, right-click the project folder and click “Build Configurations > Set Active” to select a build configuration that fits your target.



**Step 2** From the pull-down menu of the project, select “Target Configuration” to enter the **Target Configuration** page.

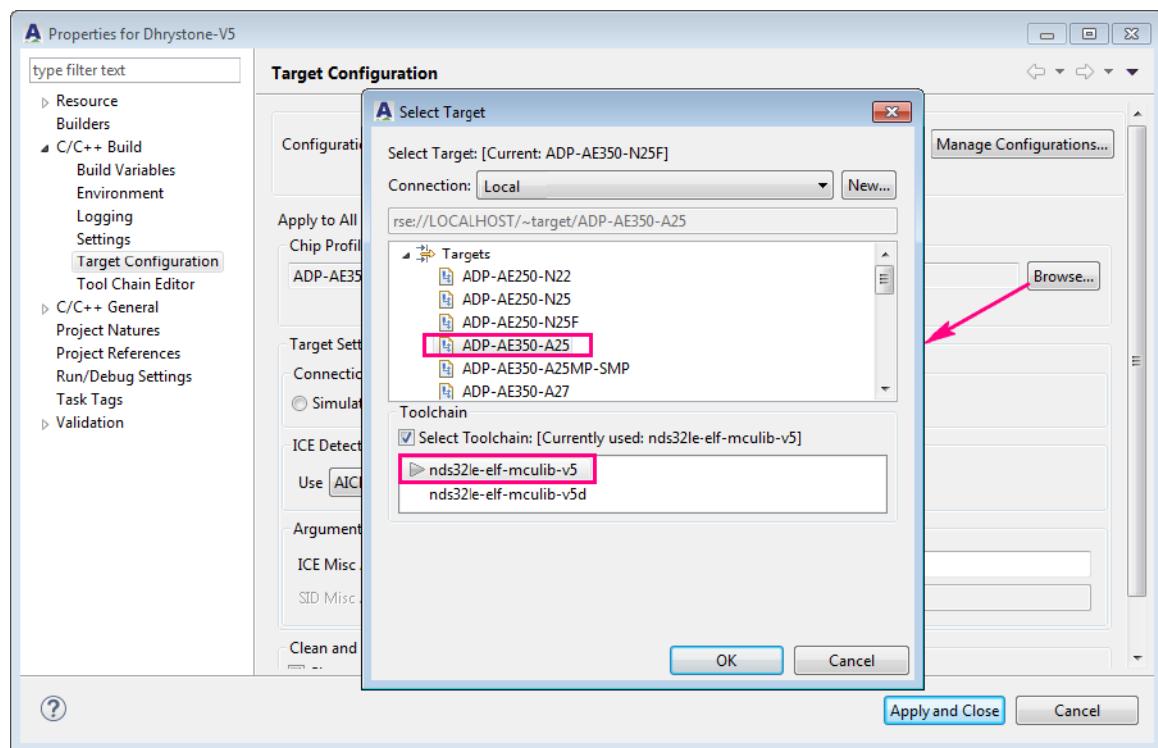


### Step 3 Specify a target and a toolchain for the project:

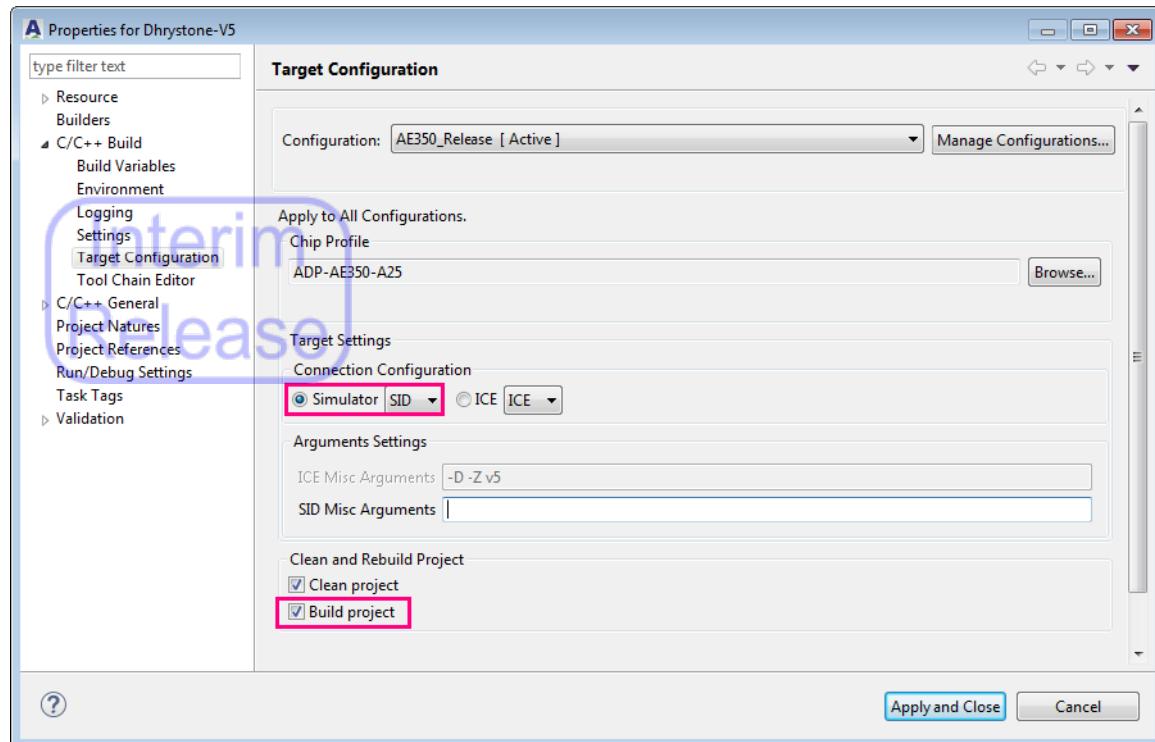
In the Chip Profile section , click “Browse...” in the Chip Profile section, specify a V5 target and a toolchain in the invoked **Select Target** wizard, and then click “OK”.

#### NOTE

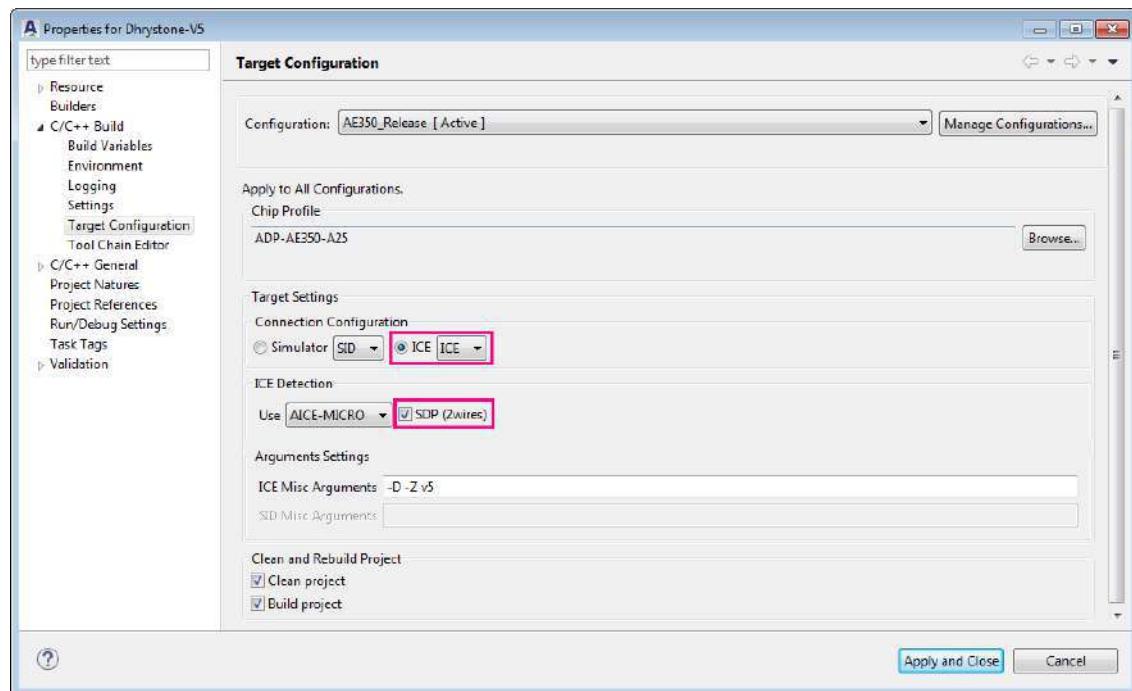
To use a Corvette-F1 target pre-integrated with fixed-configuration N22 RTL in Andes FreeStart program, make sure you select the chip profile “ADP-Corvette-F1-N22” and the toolchain “nds32le-elf-[newlib|mcullib]-v5e”.



### Step 4 On the Target Configuration page, specify a connection configuration and select the “Build Project” option.



If you are using a V5 target board with 2-wire debug interface in conjunction with the ICE device AICE-MICRO or AICE-MINI+, make sure to select “ICE” as the connection configuration and check the option “SDP (2 wires)” in the ICE detection section.



**Step 5** Jump to the next step if the project will run on an ICE target. If “SID” is selected as the connection configuration for the project, specify the following commands in the SID Misc Arguments field to enable the pipeline model. Please be aware that enabling the pipeline model may also affect performance.

- For a Corvette-F1 simulator target with fixed-configuration N22 RTL in Andes FreeStart program, enter

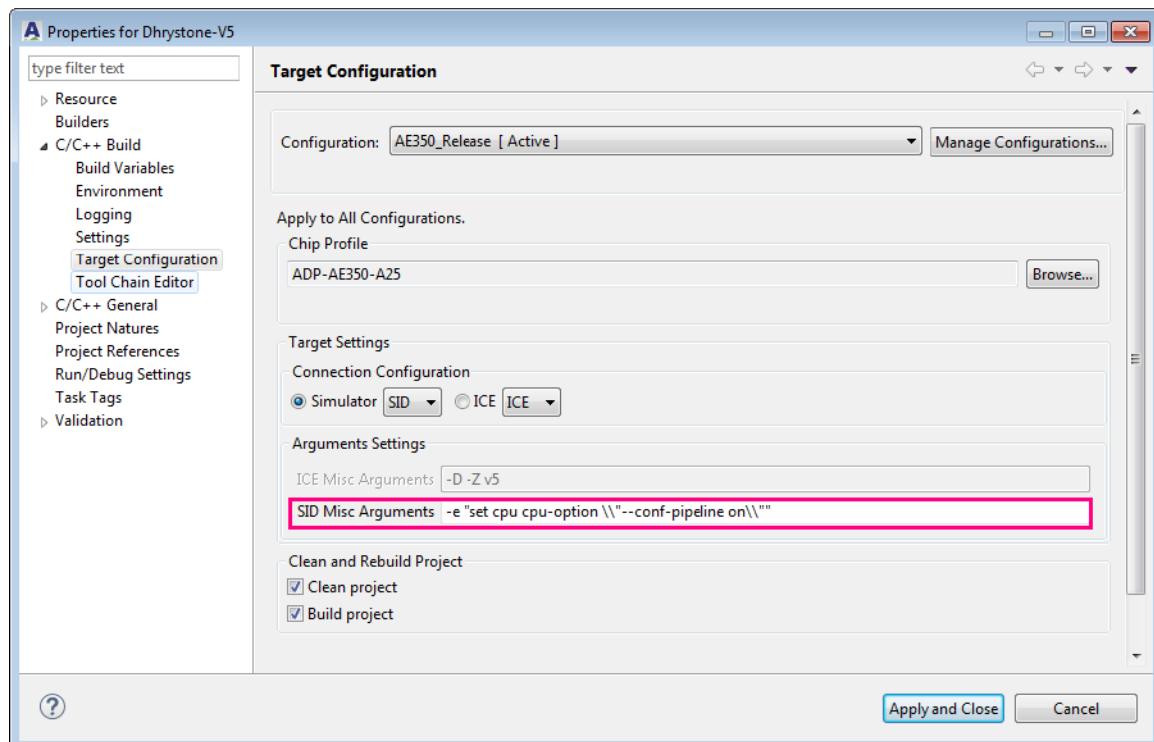
```
-e "set cpu cpu-option \\\"--conf-pipeline on --conf-gprport
3r1w --conf-is-a-e on --conf-pmp 0 --conf-pfm off
--conf-powerbrake off --conf-i-cache off --conf-user-mode off
--conf-mul radix4 --conf-btb off\\\""
```

- For other simulator targets, enter

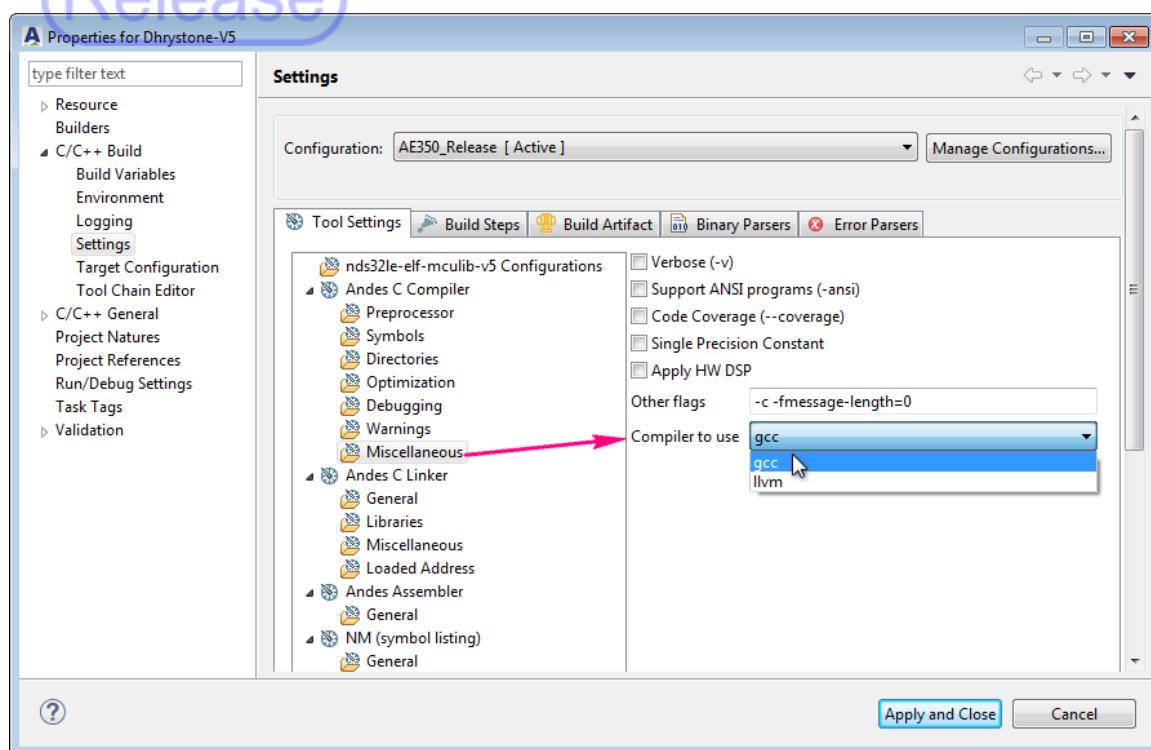
```
-e "set cpu cpu-option \\\"--conf-pipeline on\\\""
```

#### NOTE

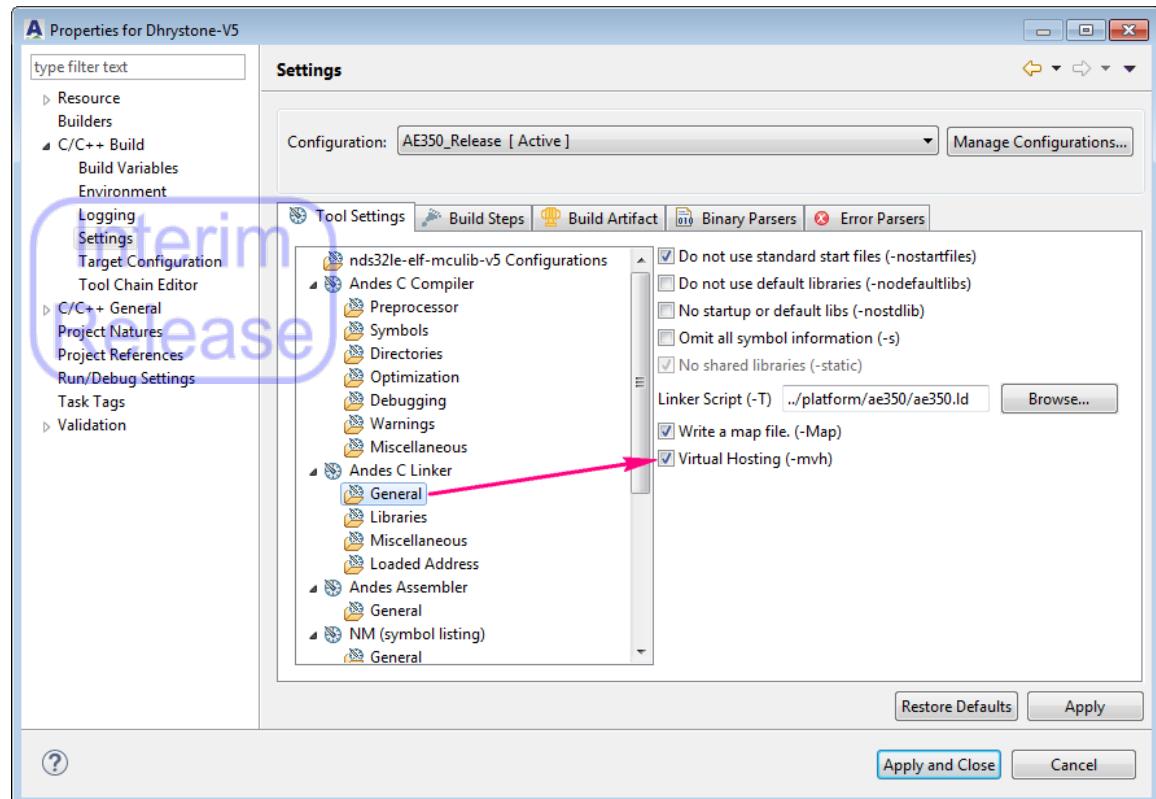
The above commands are for a Windows environment. For a Linux environment, replace the escape character “\\\" with “\\”.



**Step 6** For CoreMark-V5, jump to the next step. For Dhystone-V5 or Whetstone-V5, select a compiler between GCC and LLVM. To do so, click on “C/C++ Build > Settings” in the navigation pane of the **Properties** dialog, next click on the Tool Settings tab and select “Andes C compiler > Miscellaneous”, and then specify a compiler from the combo box “Compiler to use”.

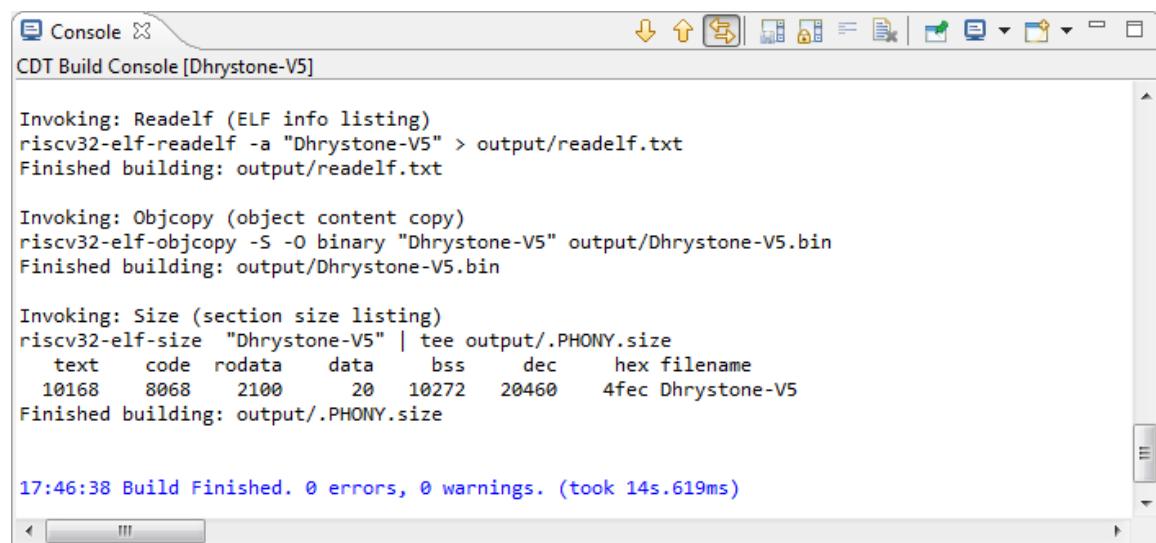


**Step 7** If output messages for the project need to be redirected to the GDB console, click on “Andes C Linker > General” in the Tool Settings tab and select the option “Virtual Hosting (-mvh)”. Note that the Virtual Hosting configuration must also be specified in **config.h** of the project (see Section 3.3.1.3, Step 2).



**Step 8** Click “Apply and Close” on the **Properties** dialog to build the project.

**Step 9** Verify the success of the build process in the **Console** view.



```

Console X
CDT Build Console [Dhrystone-V5]

Invoking: Readelf (ELF info listing)
riscv32-elf-readelf -a "Dhrystone-V5" > output/readelf.txt
Finished building: output/readelf.txt

Invoking: Objcopy (object content copy)
riscv32-elf-objcopy -S -O binary "Dhrystone-V5" output/Dhrystone-V5.bin
Finished building: output/Dhrystone-V5.bin

Invoking: Size (section size listing)
riscv32-elf-size "Dhrystone-V5" | tee output/.PHONY.size
      text   code  rodata   data    bss   dec   hex filename
  10168     8068    2100     20  10272   20460   4fec Dhrystone-V5
Finished building: output/.PHONY.size

17:46:38 Build Finished. 0 errors, 0 warnings. (took 14s.619ms)

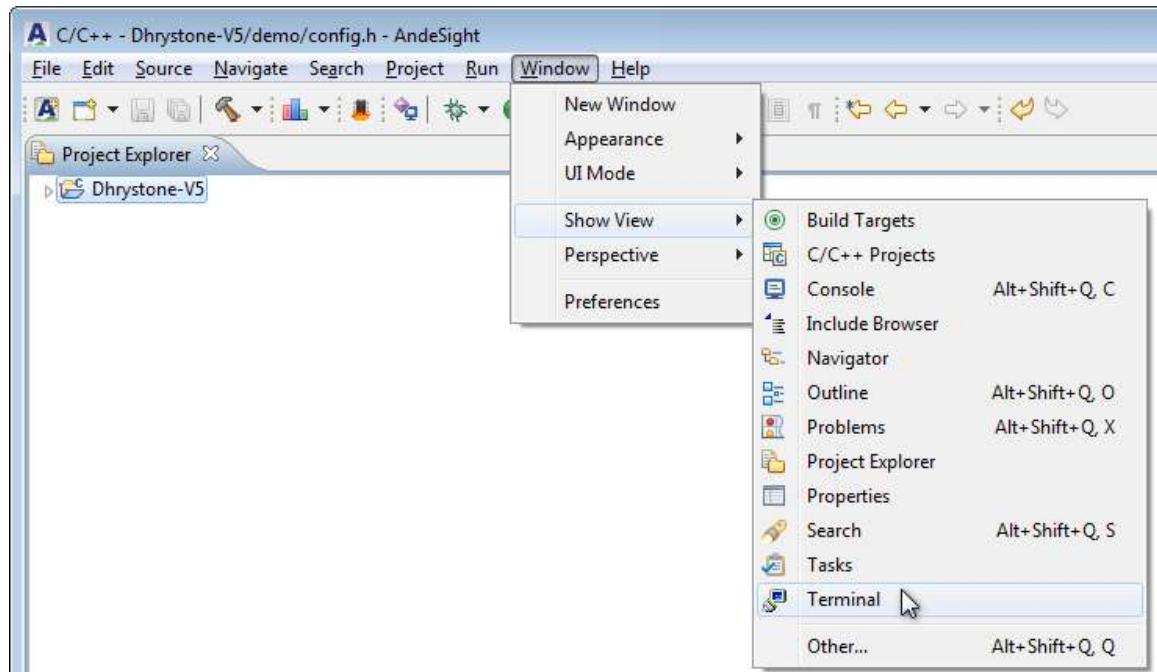
```

### 3.3.1.5 Building a terminal connection (for ICE targets only)

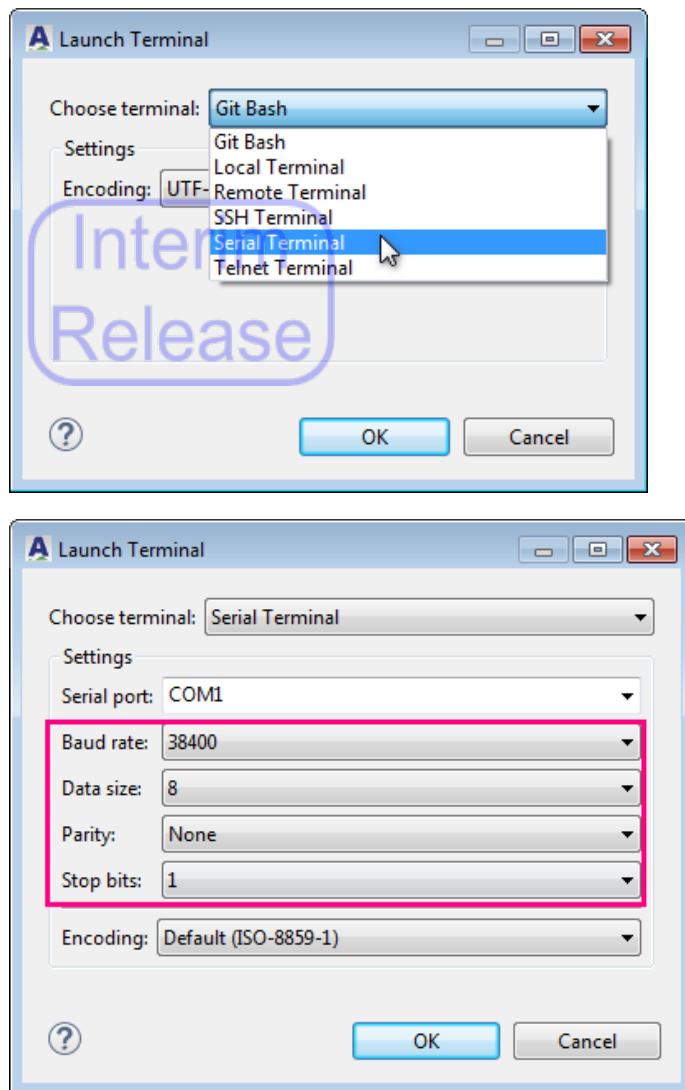
Skip this section if your Dhystone-V5/CoreMark-V5/Whetstone-V5 project runs on a simulator target.

**Step 1** Install the target board and connect an ICE device to it. For installation of an Andes ICE, please refer to the *AICE Quick Start Guide*.

**Step 2** On the AndeSight main menu, select “Window > Show View > Terminal” to invoke the **Terminal** view.

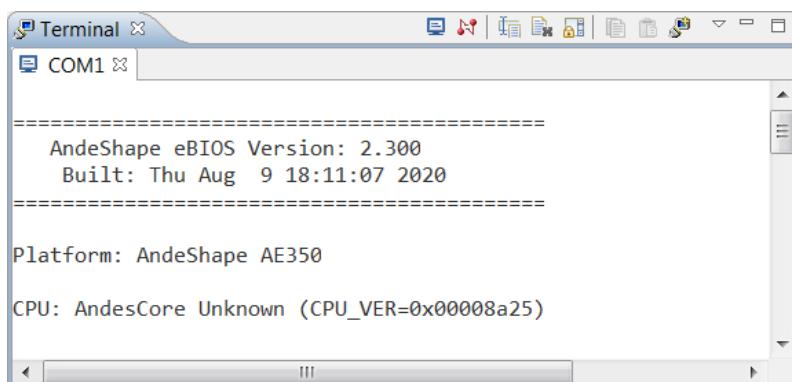


**Step 3** In the toolbar of the **Terminal** view, click  (Open a Terminal) to invoke the **Launch Terminal** wizard. Choose “Serial Terminal” and then specify the com port for target connection, and configure the port settings as follows:



Click “OK” to build the connection.

**Step 4** Toggle the power switch of the target board and press the “PWR ON” button to see the boot message in the **Terminal** view.



### 3.3.1.6 Launching a debug session for Dhystone-V5, CoreMark-V5 or Whetstone-V5

**demo**

**Step 1** In Project Explorer, right click the project folder and select “Debug As > Debug Configuration...” from the pull-down menu.



**Step 2** In the navigation pane of the invoked dialog, expand “MCU program” and click the configuration you specified for your project in Section 3.3.1.4, Step 1. Then, go to the **Startup** tab, make sure that the settings on this tab are as follows and click “Debug” to run the project.

- The “Reset and Hold” option must be checked.
- GDB commands:
  - “`set $mi1mb=0x1L`” is specified for the ILM configuration.

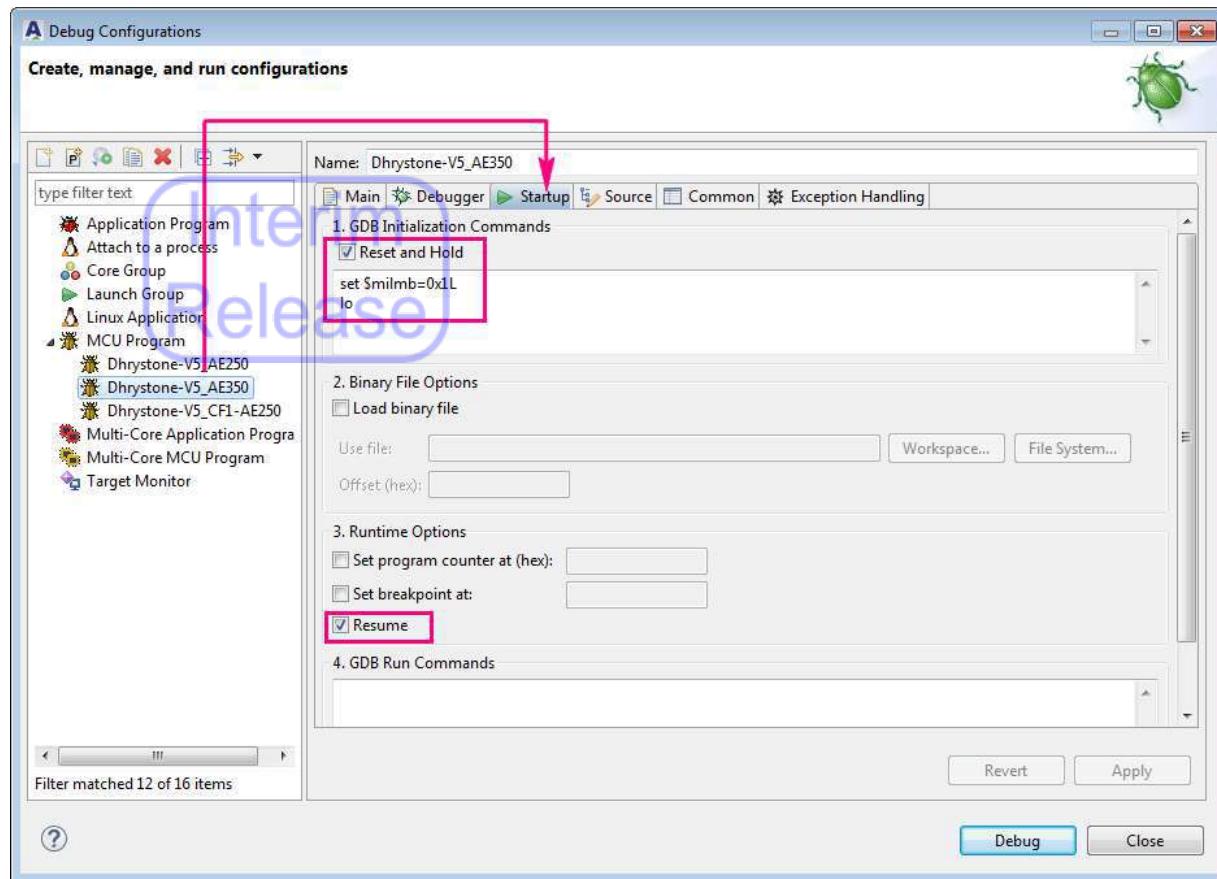
---

**NOTE**

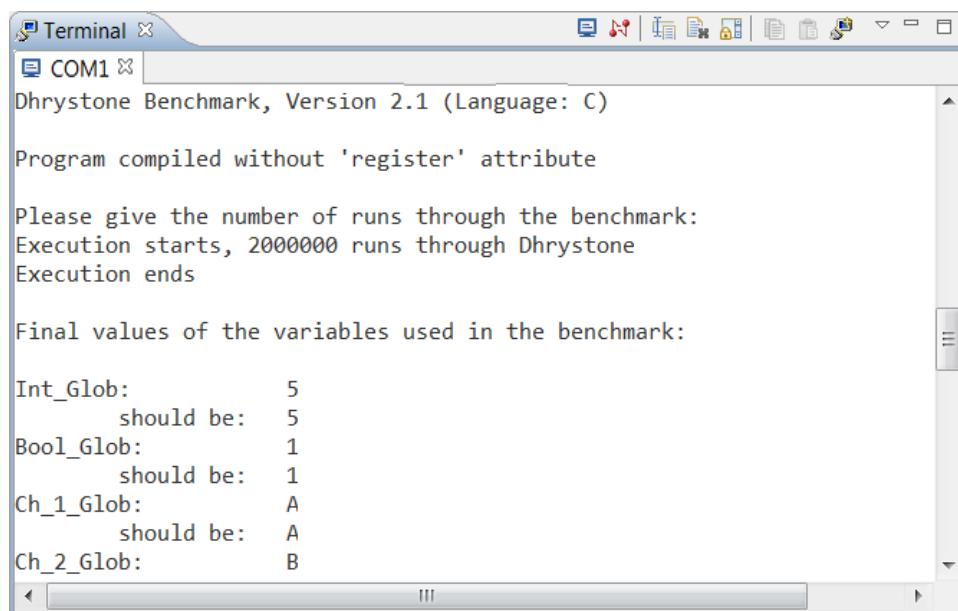
The instructions and data in local memory are not cached by the cache controller. If you want to run the demo on cache, make sure to disable ILM before turning on the cache by modifying this GDB command to “`set $mi1mb=0x0L`”. DLM will be turned off later by the benchmark program.

---

- The “`lo`” option is used to load ELF executables to the launch setting defined in the **Main** tab.
- The “`Resume`” option is used to resume program execution.

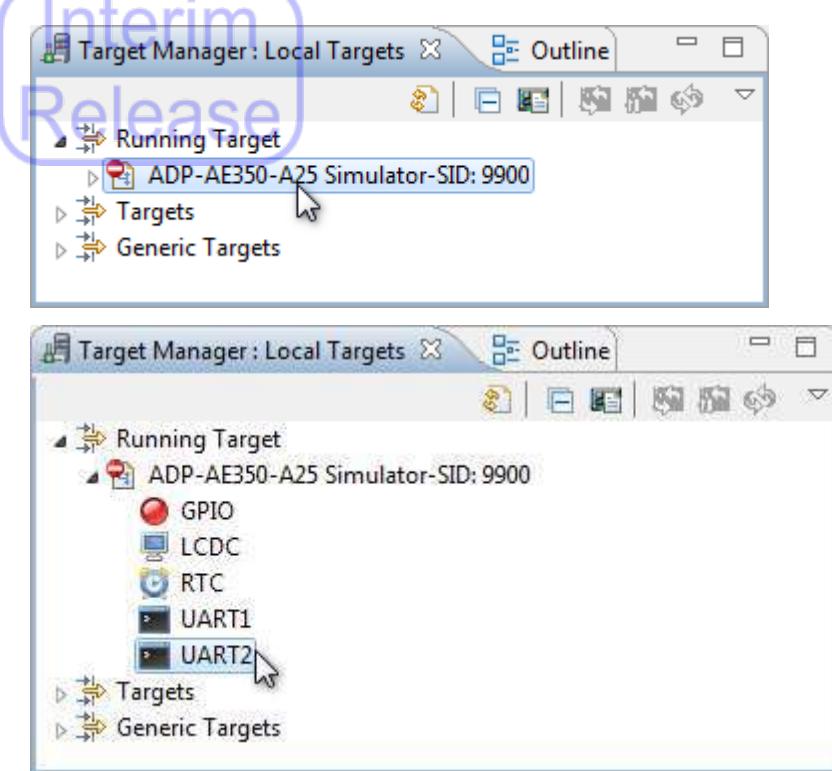


### Step 3 For an ICE target, check the results in the Terminal view.

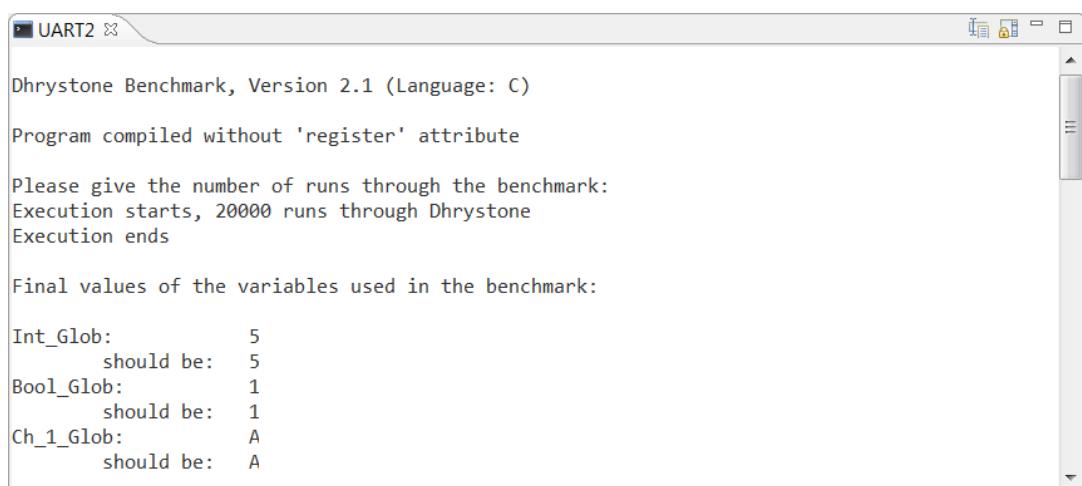


For a project with simulator target, the results are displayed in a UART console, which can be invoked as follows:

1. In the **Target Manager** view, double-click the launched target and double-click “UART2” in the expanded tree.



2. Check the result in the invoked **UART** console.




---

#### NOTE

1. An error message "ERROR! Must execute for at least 10 secs for

a valid result!" shows when CoreMark-V5 project runs on a simulator target with 350 iterations. This message can be ignored as it is designed to get a valid CoreMark result on ICE targets.

2. The CoreMark score can be obtained from "Iterations/Sec" of the CoreMark-V5 result. To measure the single thread performance of the CPU, the score needs to be divided by the CPU frequency (i.e., CoreMark/MHz). For CPU frequency of respective targets, please refer to Step 3.

The following is a CoreMark-V5 result from an AE350 target, which has a CPU frequency of 60MHz. It shows to deliver a CoreMark score of 212.30 per second. Thus, its CoreMark/MHz value is known as 3.54 ( $212.30/60$ ).

```
2K performance run parameters for coremark.
CoreMark Size      : 666
Total ticks        : 98916105
Total time (secs) : 1.648602
Iterations/Sec     : 212.301121 ← CoreMark score
ERROR! Must execute for at least 10 secs for a valid result!
Iterations         : 350
Compiler version   : GCC7.3.0
Compiler flags     : -O3 -mcmode=medium -funroll-all-loops -finline-limit=600 -ftree-dominator-opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting
                     -g3 -mcpu=n25f -fmessage-length=0
Memory location    : STACK
seedcrc            : 0xe9f5
[0]crclist         : 0xe714
[0]crcmatrix       : 0x1fd7
[0]crcstate        : 0x8e3a
[0]crcfinal        : 0x6621
Errors detected
```

### 3.3.2. Replacing Dhrystone with other benchmarks

You can also modify the Dhrystone code to make a new benchmark for the evaluation of CPU performance. This requires removing the source files of the Dhrystone project ([dhry\\_1.c](#), [dhry\\_2.c](#) and [dhry.h](#)) and adding source files to adopt the new benchmark (see Section 2.1.2.2). Note that the size of the added code and data should not exceed 256/512KB.

If any of the options defined in [config.h](#) (such as [CPU\\_MHz](#) or [LOOP\\_COUNT](#)) are to be used for the new benchmark, then make sure to include [config.h](#) in the new source files. To determine the time spent on the new benchmark, utilize the following code:

```
extern long    time();  
long    Begin_Time, End_Time, User_Time;  
  
Begin_Time = time( (long *) 0 );  
... the loop of benchmark...  
End_Time = time( (long *) 0 );  
User_Time = End_Time - Begin_Time;
```

In the above code, [User\\_Time](#) refers to the time spent in the loop of the benchmark.

### 3.4. AMP (Asymmetric multiprocessing) demo program

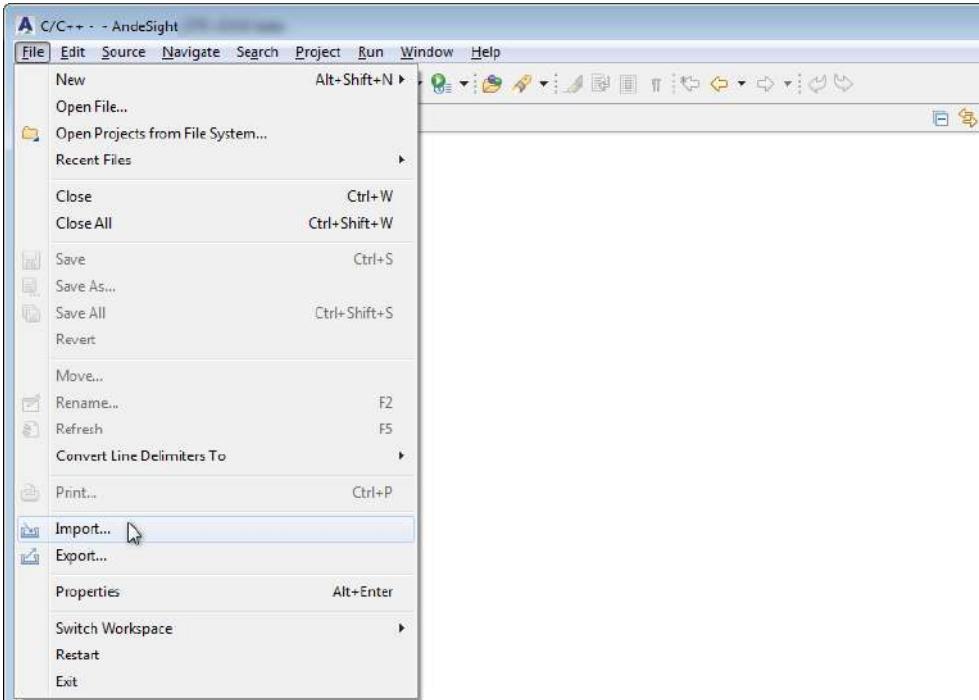
The demo program “demo-amp-V5” demonstrates inter-processor communication (IPC) using a mailbox and shared memory on a dual-core target without SMP support. It kicks off the two cores on the target in the beginning. Core0 outputs an alive message and then notifies Core1 via a flag on shared memory. Core1 follows to output an alive message after receiving the notification and informs Core0 via the flag too. The demo scenario goes on with two cores taking turns to output an alive message through the same method.

The following provide step-by-step instructions on how to run the AMP demo program. The operation focuses on the following:

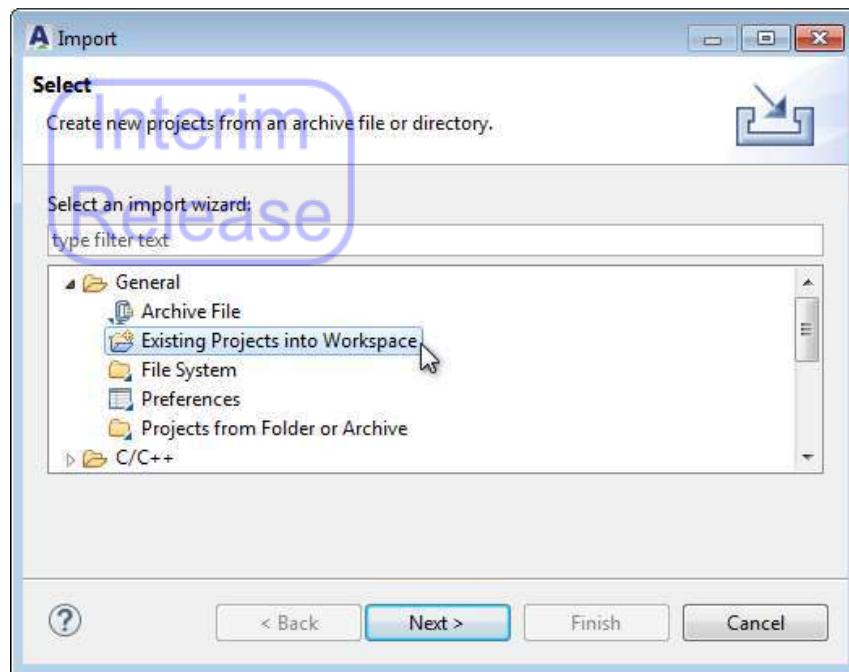
- How to import an existing project
- How to specify the target configuration for the project
- How to build an AMP project
- How to debug an AMP project and examine the execution results on respective cores

#### 3.4.1. Importing the AMP demo project

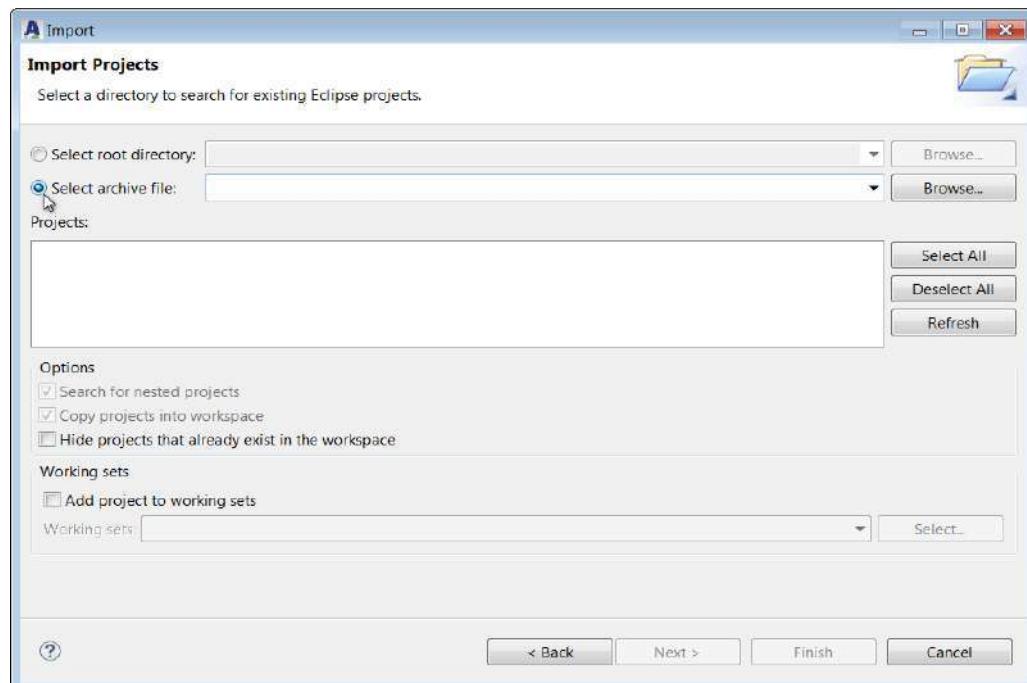
**Step 1** On the AndeSight main menu, click “File” and select “Import...” from the pull-down menu.

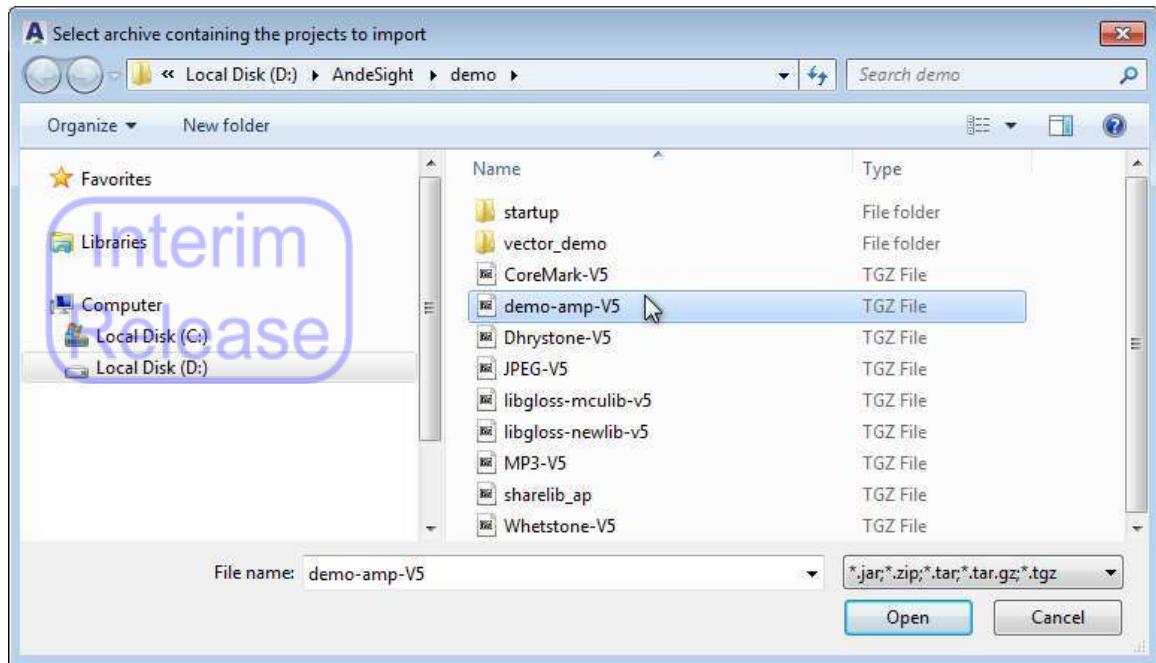


**Step 2** On the invoked **Import** dialog, select “General > Existing Projects into Workspace” and click “Next.”

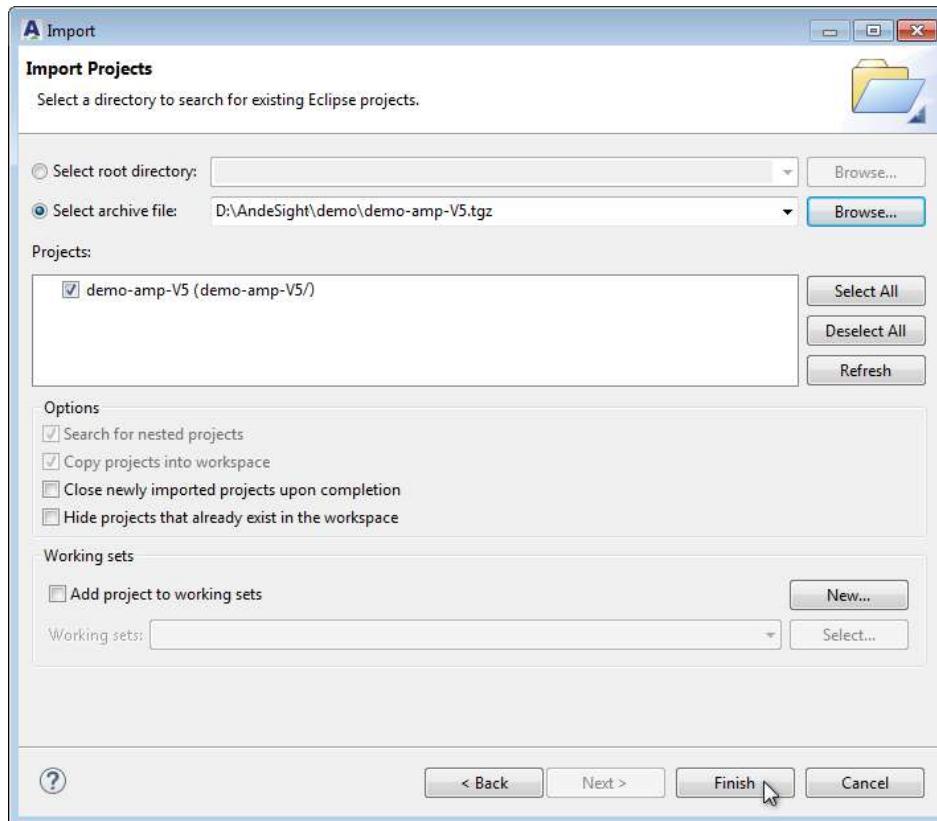


**Step 3** The AMP demo program is provided as a tarball ([demo-amp-V5.tgz](#)) in **ANDESIGHT\_ROOT\demo\**. To import the compressed file, select the “Select archive file” radio button and click “Browse...” to search for the file in the invoked wizard. Then, click “Open”.

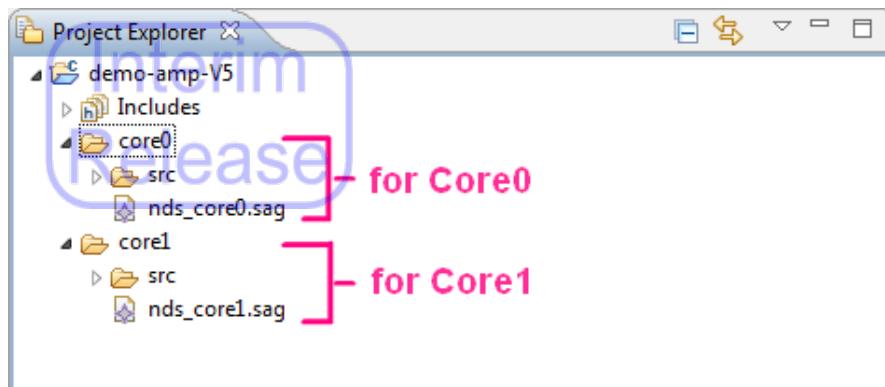




**Step 4** Back in the **Import** dialog, click “Finish” to complete project importation.

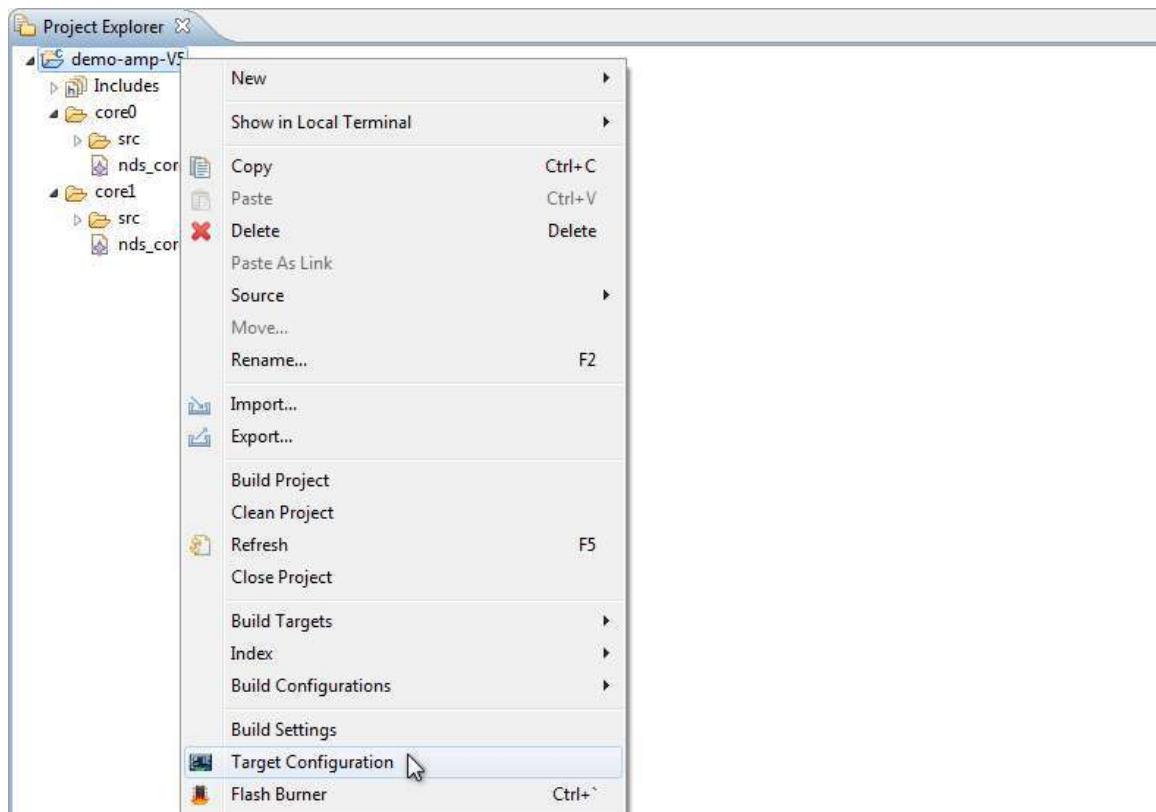


**Step 5** Find the imported AMP demo project “demo-amp-V5” in the **Project Explorer** view. The project contains sub-folders that provide associated files for each core.

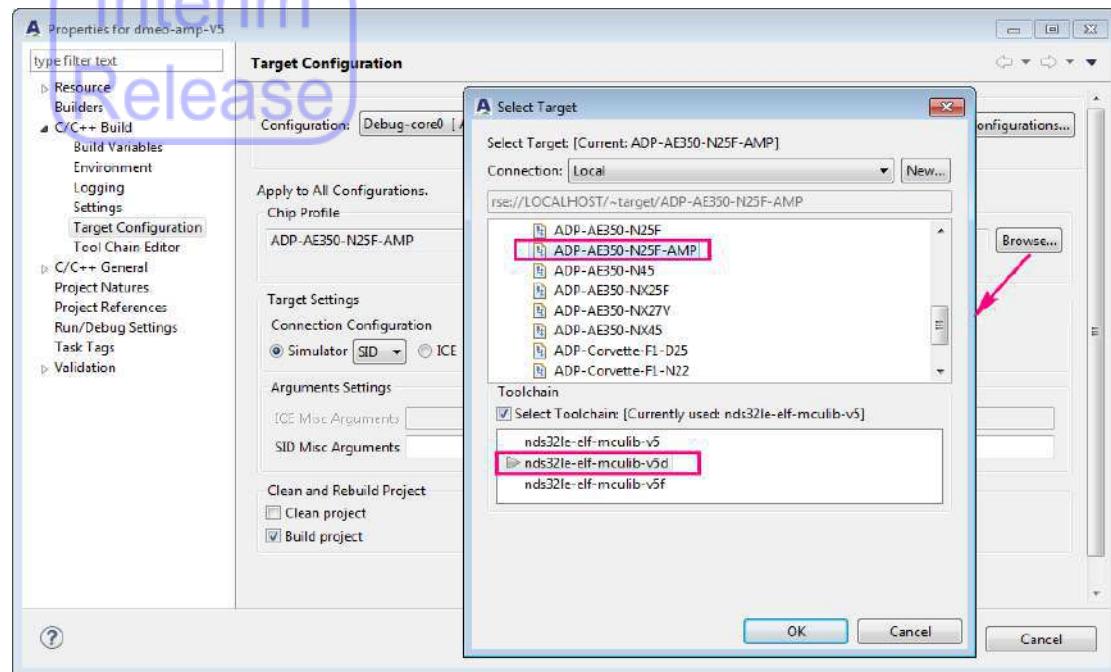


### 3.4.2. Specifying the target configuration and building the AMP project

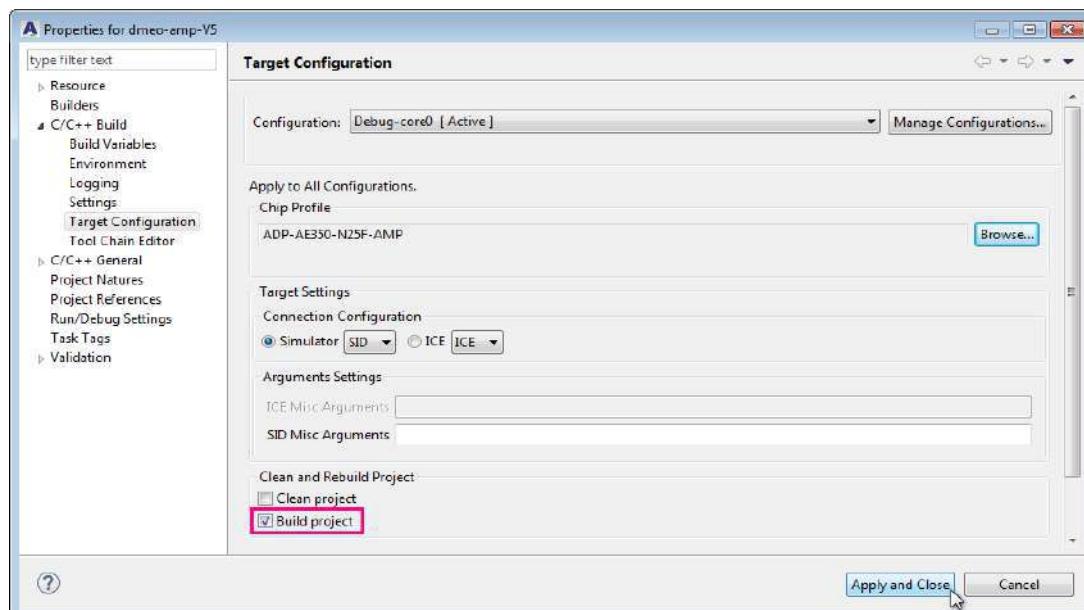
**Step 1** In the **Project Explorer** view, right-click the project folder to trigger the pull-down menu and select “Target Configuration”.



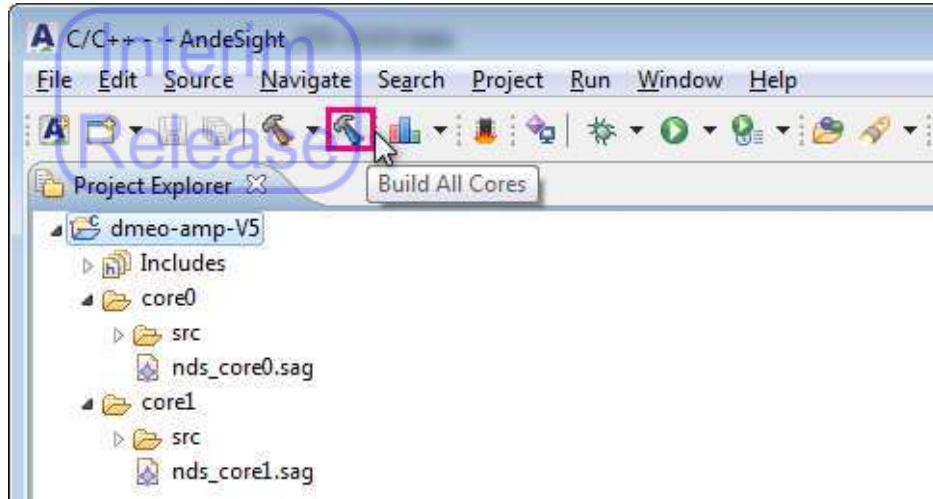
**Step 2** In the Chip Profile section of the invoked dialog, check if a chip profile of dual-core AMP system is selected. If not, click “Browse” to search one in the **Select Target** dialog and specify a compatible toolchain.



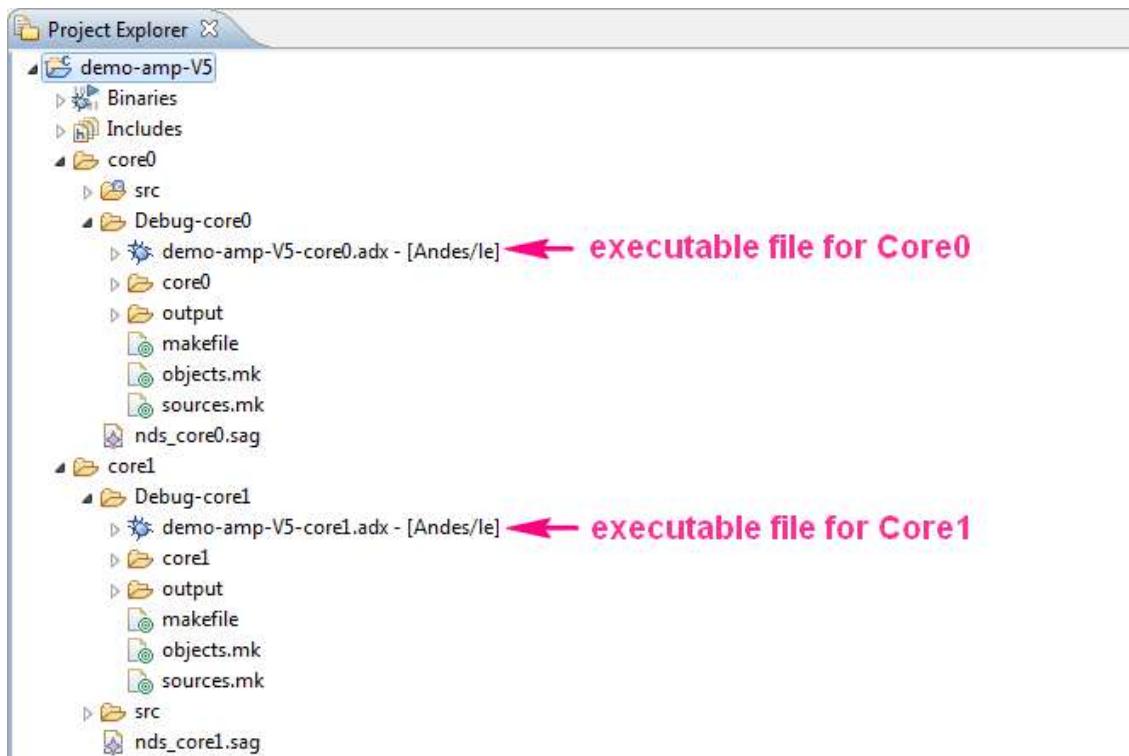
**Step 3** On the **Properties** dialog, make sure that the “Build project” option is selected and click “Apply and Close” to commence the build process for both cores.



To build program executables for AMP cores at once, you can also select the project in the **Project Explorer** view and click on  (Build All Cores) on the AndeSight toolbar.



**Step 4** After a successful build, the executables for respective cores are generated under **PROJECT\[\ core0|core1\]\Debug- [\ core0|core1]**.

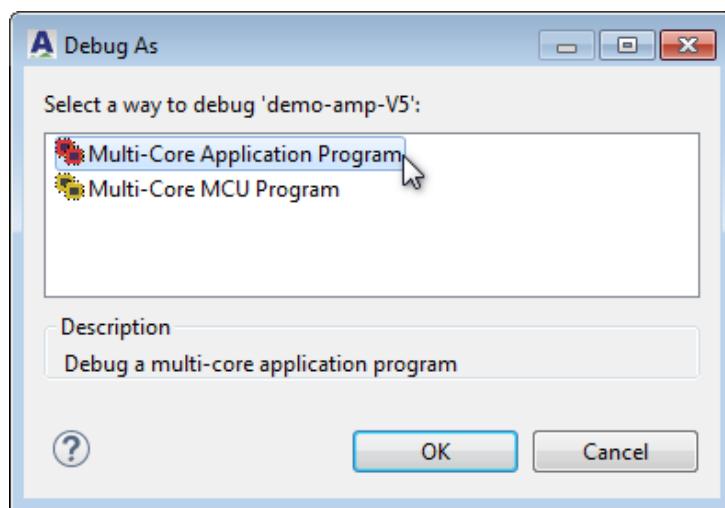


### 3.4.3. Debugging the AMP project

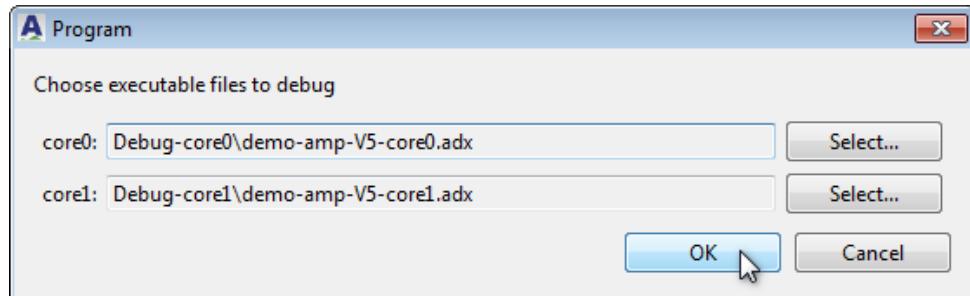
**Step 1** Select the demo project in the **Project Explorer** view and click  (Debug As...) on the AndeSight toolbar.



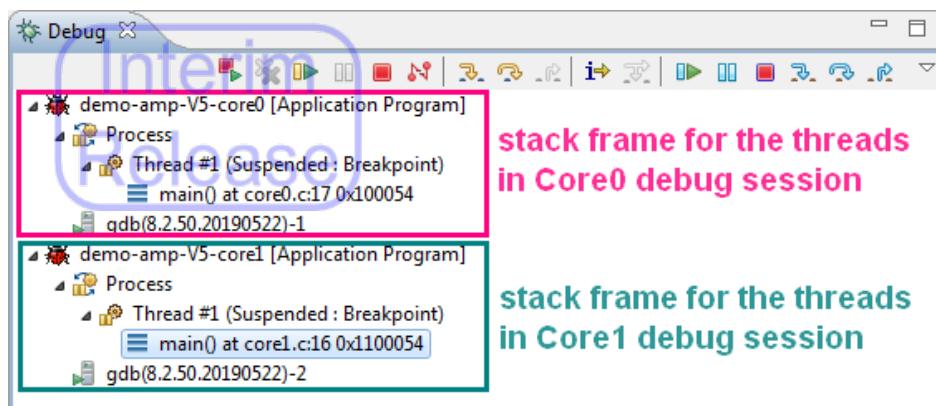
**Step 2** In the **Debug As** wizard, select a multi-core debug configuration and click “OK.”



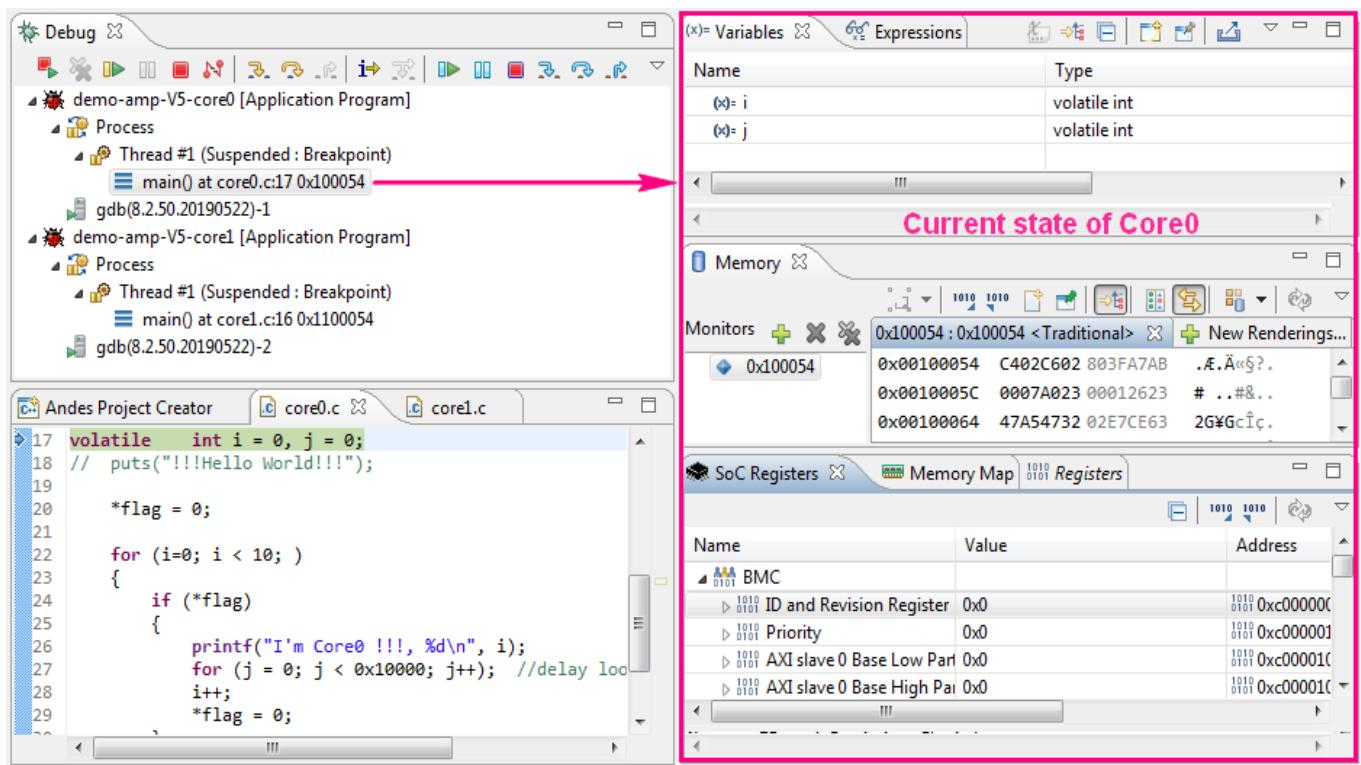
**Step 3** In the **Program** wizard, make sure that each core is specified with its corresponding executable file and click “OK” to launch the debug sessions for Core0 and for Core1 simultaneously.



**Step 4** The **Debug** view displays stack frames for threads in Core0 and Core1 debug sessions. It shows that the program execution in either debug session is suspended at the main function.

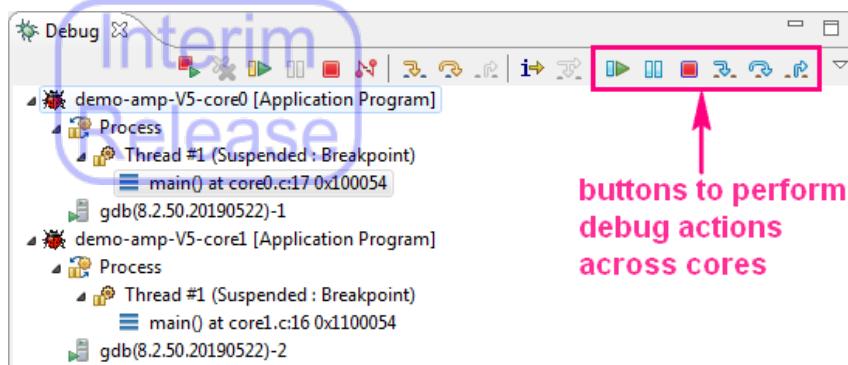


You may select the stack frame of a specific core in the **Debug** view to inspect the current state of the core from other debug views (e.g., **Registers** view, **Variables** view, **Memory** view).

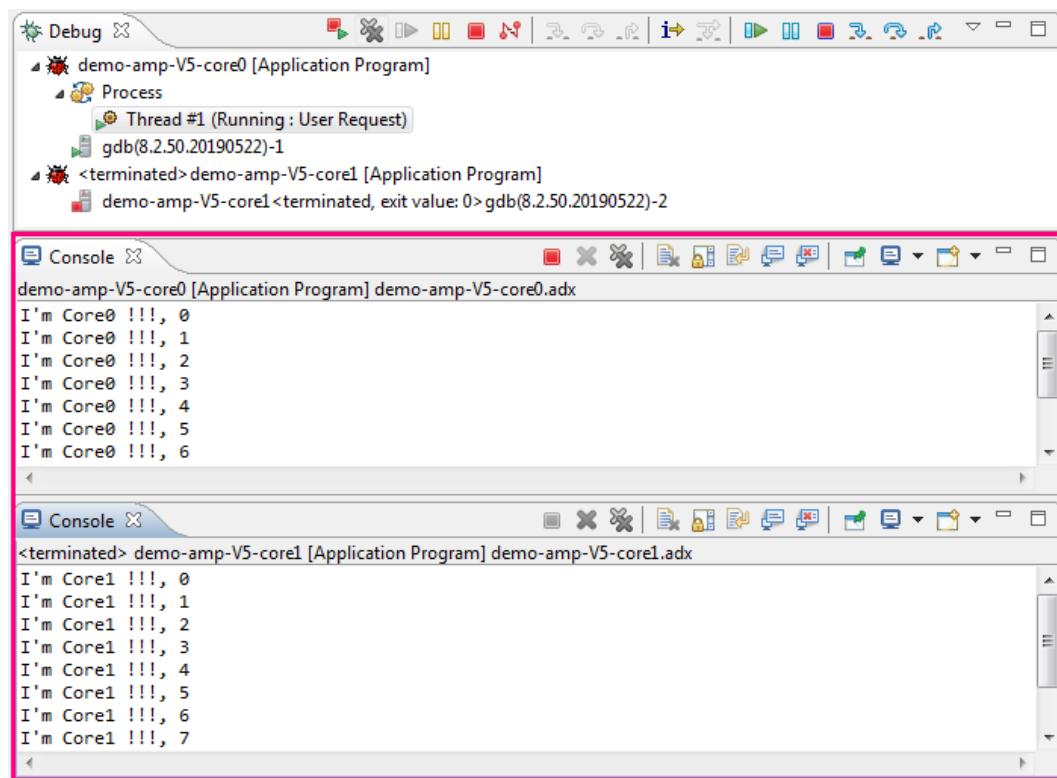


**Step 5** To resume the program execution on both Core0 and Core1 at the same time, click the toolbar button  (Group Resume) in the **Debug** view.

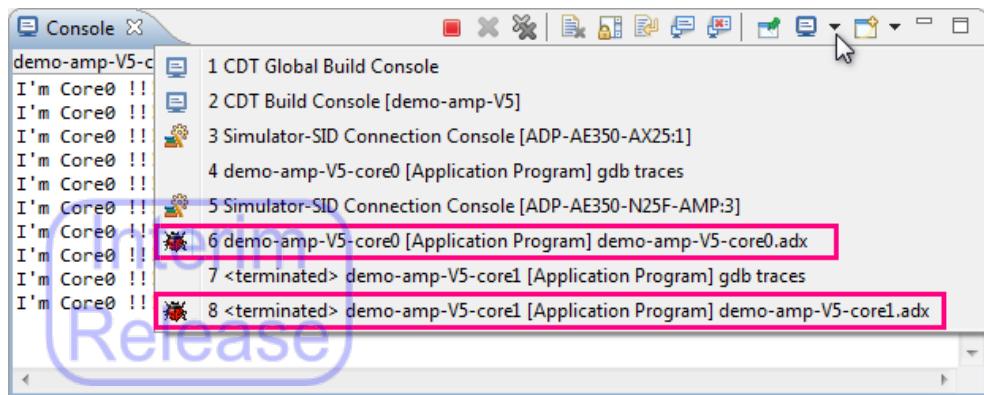
You can also click buttons like  (Group Step Into),  (Group Step Over) or  (Group Step Return) to perform debug actions across both cores.



**Step 6** Observe the execution results on Core0 and on Core1 in the **Console** view(s).



The toolbar button  (Display Selected Console) on the **Console** view allows you to switch between consoles of Core0 and Core1 execution results.



### 3.5. Vector demos

Andes provides four vector demo programs covering simple and sophisticated functions to demonstrate the acceleration with RISC-V Vector ISA Extension in Andes RISC-V vector processors. These vector demo programs and their respective descriptions are as follows:

#### ■ demo\_v\_f32\_add

This demo program reads values from an array, adds a constant to them, and then stores the results back to the array.

#### ■ demo\_mat\_mul\_f32

This demo program demonstrates a specific matrix multiplication in which, given that both ROW and COL are a multiple of 8, a ROW \* COL matrix is multiplied by a COL \* COL2 matrix and the multiplication results are stored in a ROW \* COL2 matrix.

#### ■ demo\_nn\_conv\_HWC\_q7

This demo program convolves an input tensor with a kernel weight, and stores the convolution results to an output tensor.

#### ■ demo\_nn\_relu\_q7

This demo program reads values from an array, filters the values with ReLU (Rectified Linear Unit), and then stores the results back to the array.

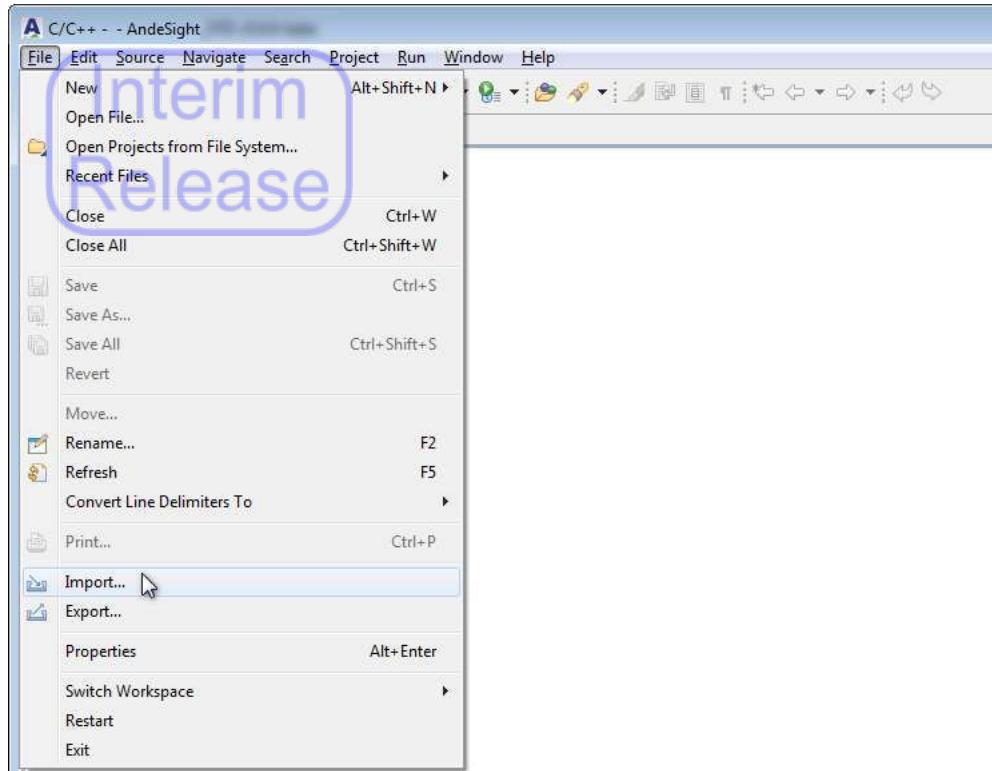
To present the speedup achieved by using vector extension algorithms, pure C algorithms are also implemented in these vector demo programs for comparison.

The following provide step-by-step instructions on how to run a vector demo program. The operation focuses on the following:

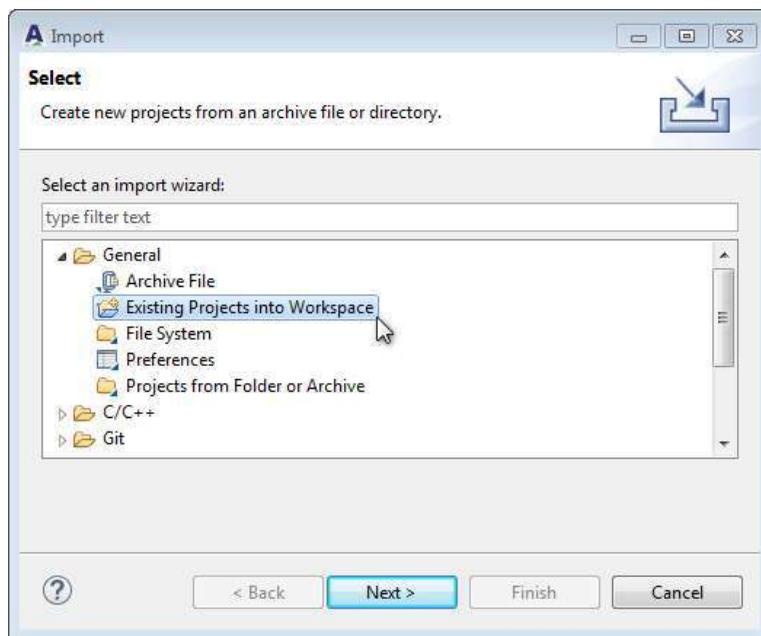
- How to import an existing project
- How to specify the target configuration for the project
- How to build a project
- How to execute and validate the vector demo project

### 3.5.1. Importing a vector demo project

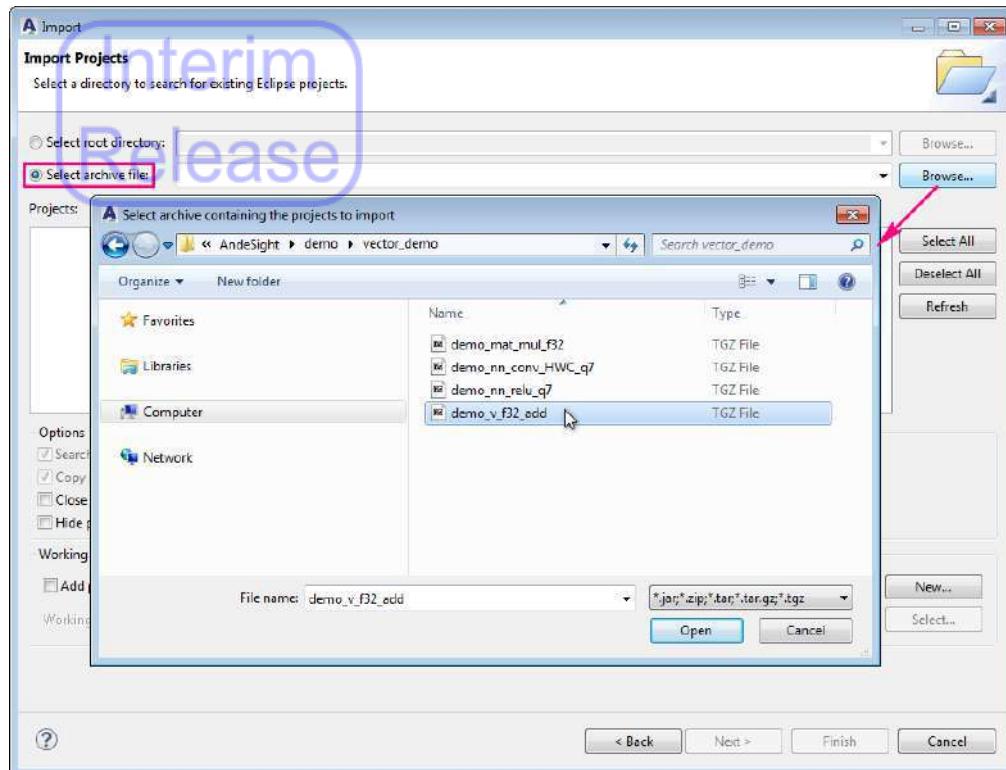
**Step 1** On the AndeSight main menu, select “File > Import”.



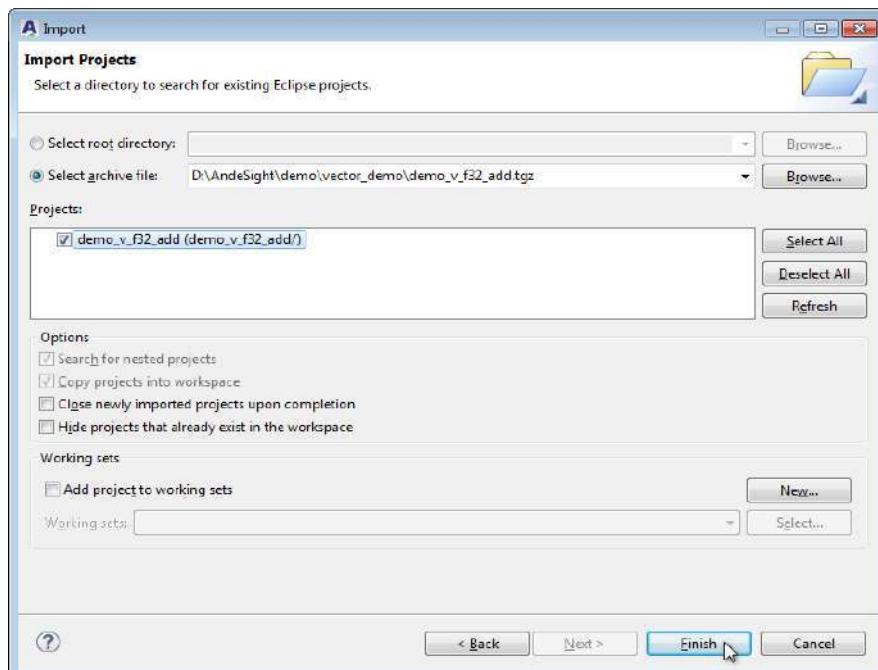
**Step 2** Select “General > Existing Project into Workspace” on the invoked dialog and click “Next”.



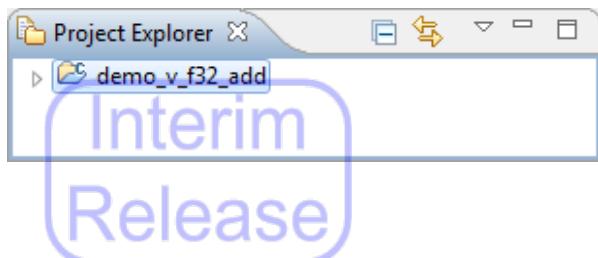
**Step 3** Select the “Select archive file” radio button, click “Browse...” to specify a desired vector demo project under **ANDESIGHT\_ROOT/demo/vector\_demo**, and click “Open”.



**Step 4** Back in the Import dialog, click “Finish.”

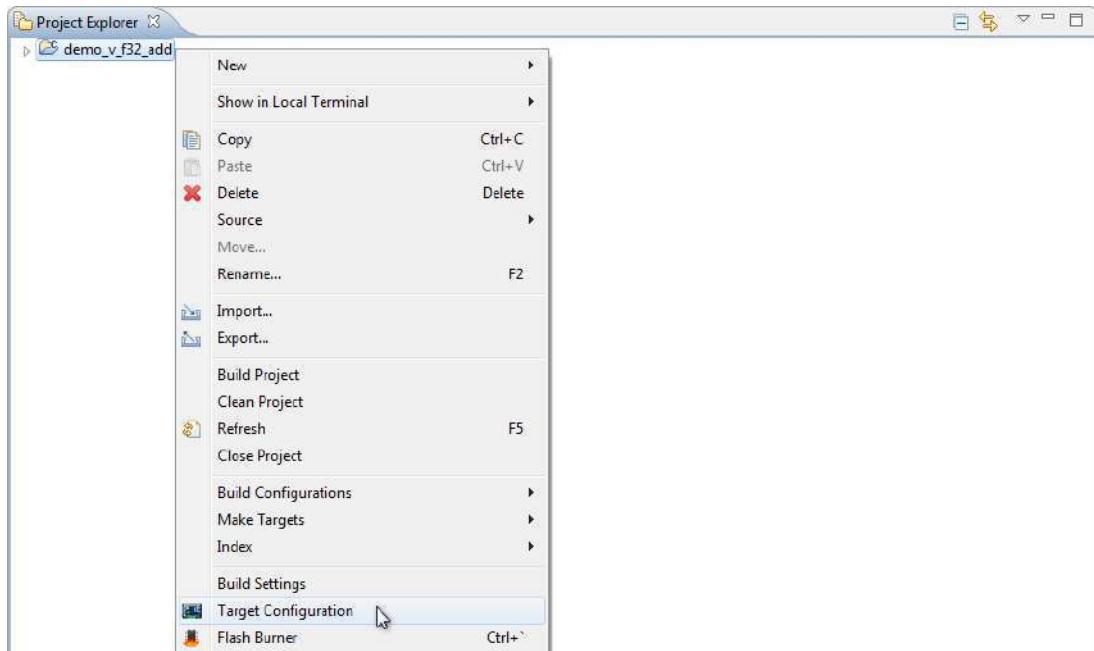


**Step 5** The imported vector demo project is shown in the **Project Explorer** view in AndeSight.

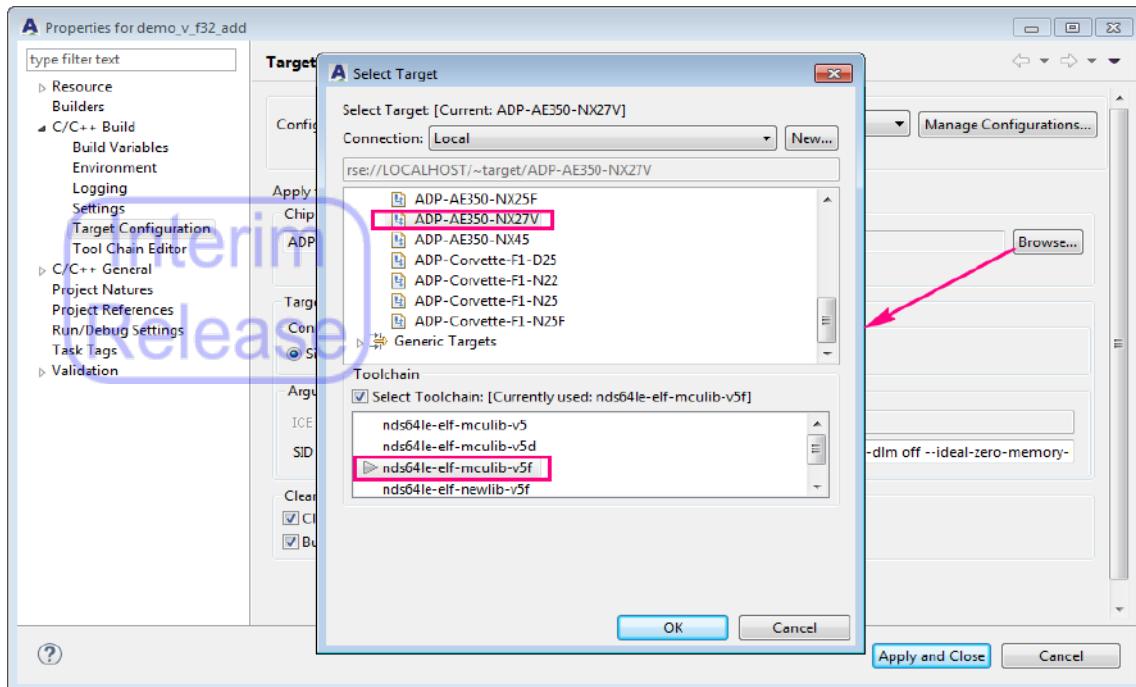


### 3.5.2. Specifying the target configuration and building the project

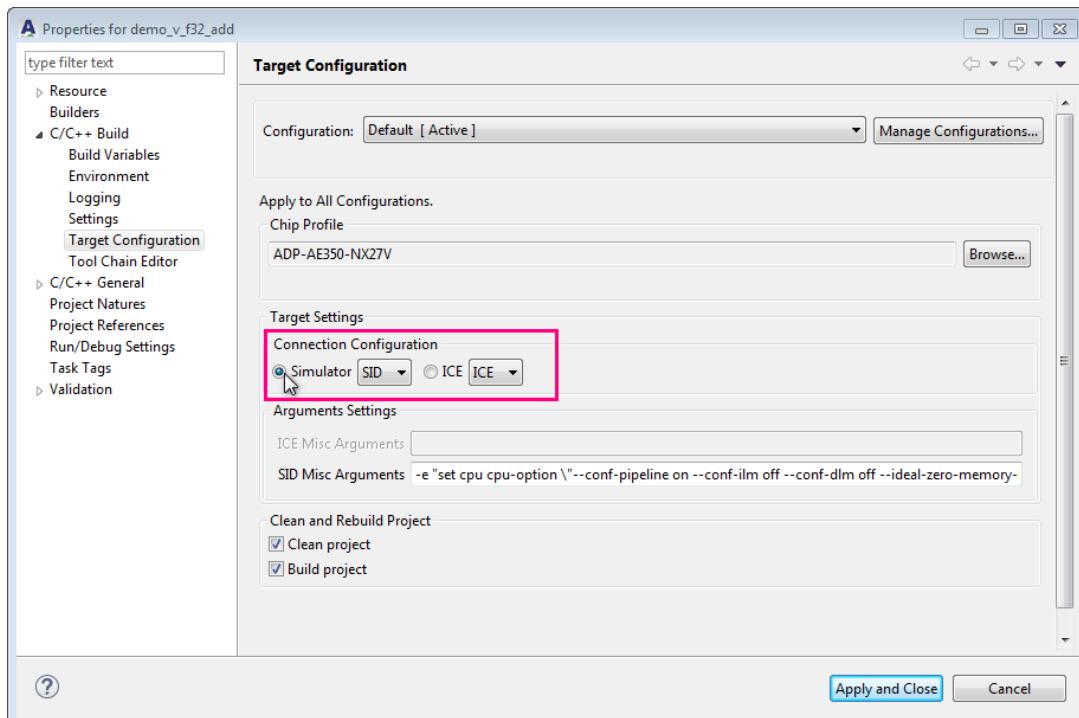
**Step 1** Right-click on the vector demo project in **Project Explorer** and select “Target Configuration” from the pull-down menu.



**Step 2** On the Target Configuration page of the **Properties** dialog, click on “Browse...” in the Chip Profile section to invoke a Select Target dialog. Specify a chip profile that supports vector extension, select a working toolchain, and then click “OK”. In the following example, the chip profile “ADP-AE350-NX27V” and the toolchain “nds64le-elf-mculib-v5f” are selected.

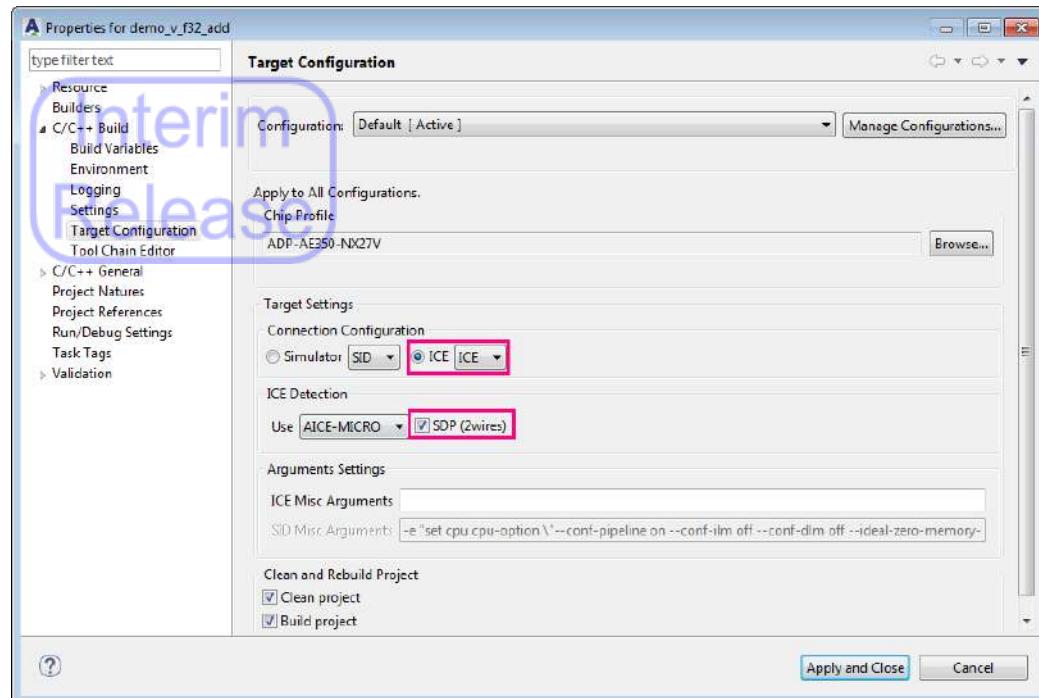


**Step 3** Back in the **Properties** dialog, select a connection configuration for your target.

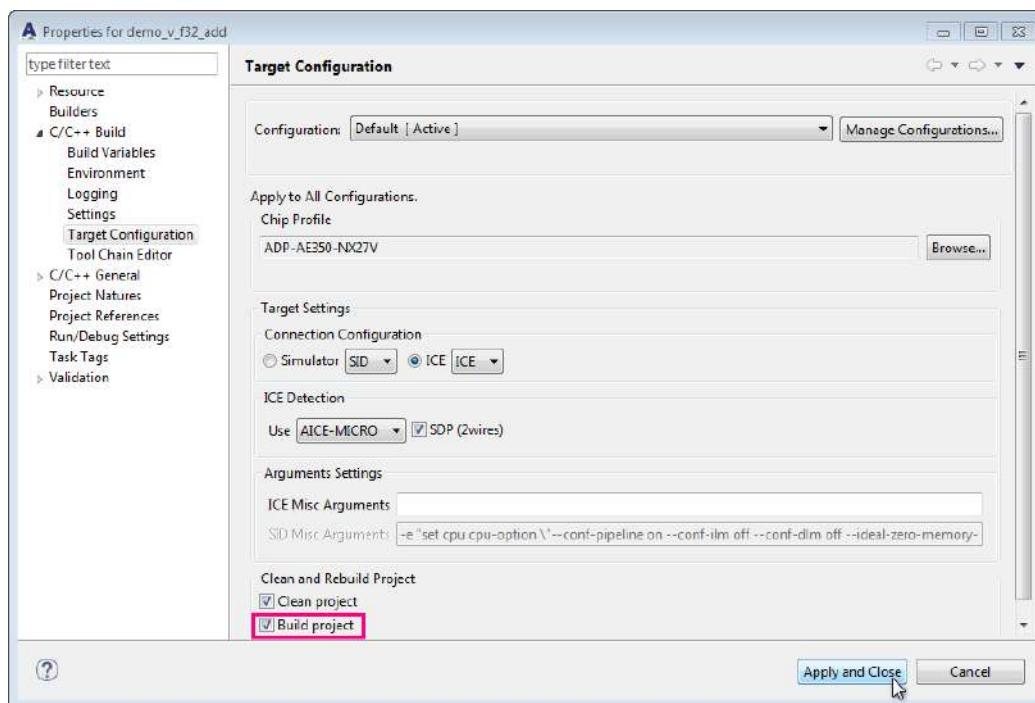


If you are using a V5 target with 2-wire debug interface in conjunction with the ICE device AICE-MICRO or AICE-MINI+, make sure to select

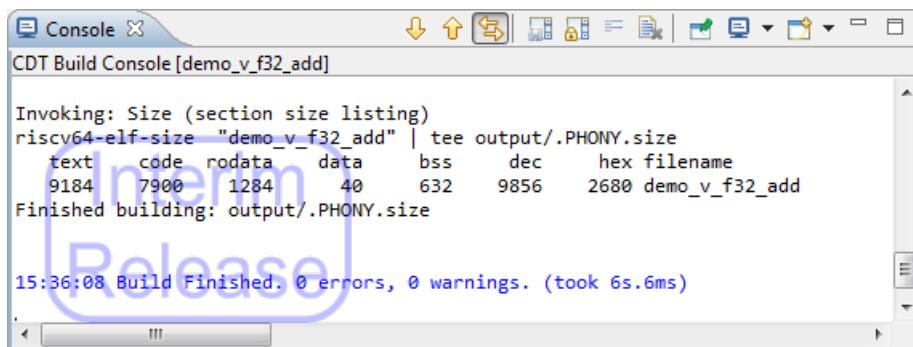
“ICE” as the connection configuration and check the option “SDP (2 wires)” in the ICE detection section.



**Step 4** Make sure that the option “Build project” is selected and click on “Apply and Close” to commence the build process.



**Step 5** Verify that the build is successful in the **Console** view.



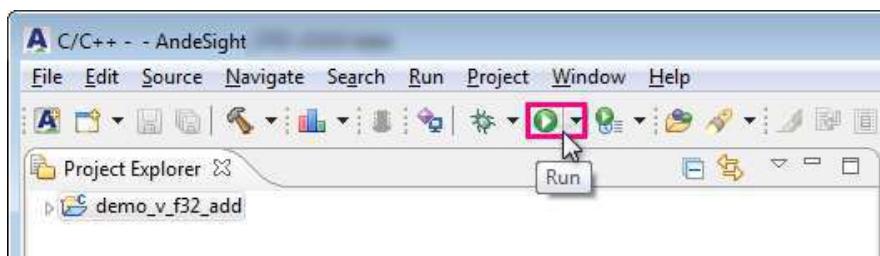
```
CDT Build Console [demo_v_f32_add]

Invoking: Size (section size listing)
riscv64-elf-size "demo_v_f32_add" | tee output/.PHONY.size
text    code   rodata  data    bss    dec    hex filename
9184    7900   1284    40     632    9856   2680 demo_v_f32_add
Finished building: output/.PHONY.size

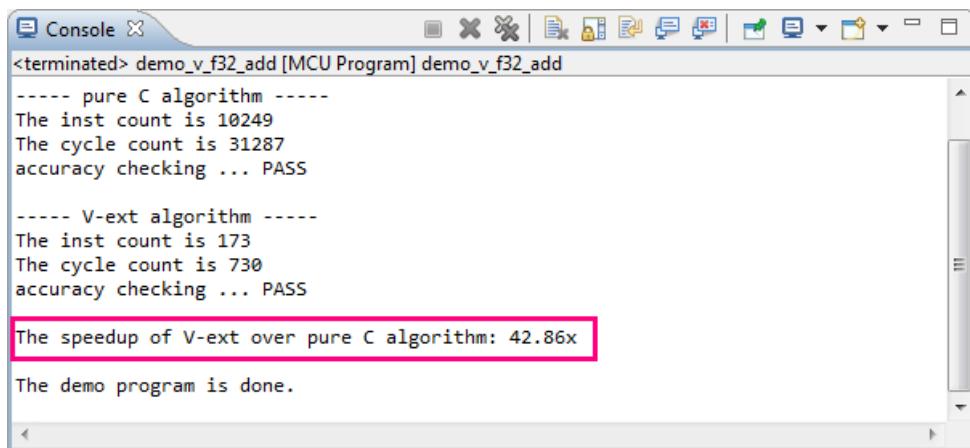
15:36:08 Build Finished. 0 errors, 0 warnings. (took 6s.6ms)
```

### 3.5.3. Running the vector demo project

**Step 1** On the AndeSight toolbar, click  (Run) to run the demo project.



**Step 2** Check the **Console** view for the respective execution result of pure C algorithms and V extension algorithms, which includes the instruction count, cycle count, and outcome of accuracy check to validate the execution. You can also obtain the speedup ratio achieved by vector extension algorithms over pure C algorithms in the view.



```
<terminated> demo_v_f32_add [MCU Program] demo_v_f32_add
----- pure C algorithm -----
The inst count is 10249
The cycle count is 31287
accuracy checking ... PASS

----- V-ext algorithm -----
The inst count is 173
The cycle count is 730
accuracy checking ... PASS

The speedup of V-ext over pure C algorithm: 42.86x

The demo program is done.
```

## 4. Program development using the GDB command line console

For development using Andes toolchains without the aid of an IDE graphical user interface, follow the instructions in this chapter to perform debugging and profiling via the GDB command line console.



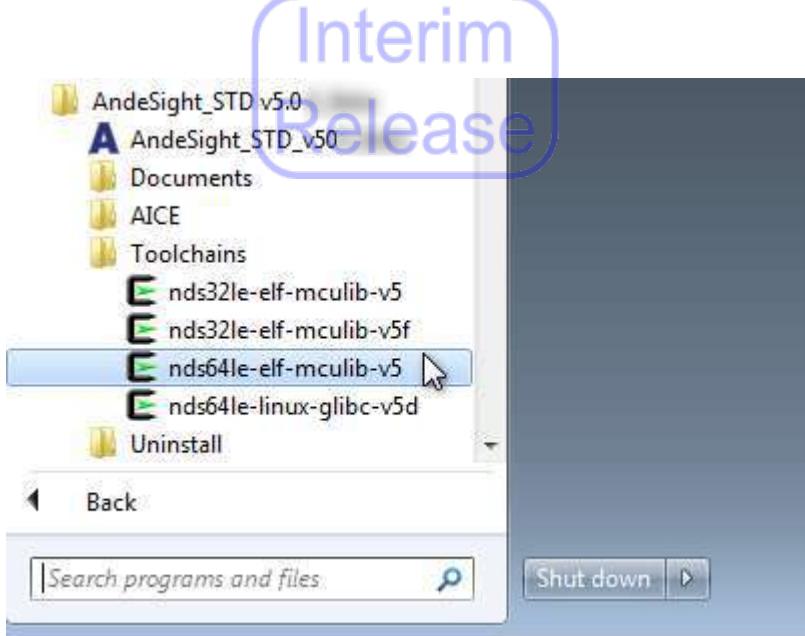
---

### NOTE

1. Altering the bit value of the Processor Status Word register via the debugger interface is error-prone; therefore, this is not recommended. For instance, changing the values of the registers, POM, IT, and DT may cause the debug function to halt. Please consult *AndeStar™ System Privileged Architecture Manual* for the definition of the Processor Status Word register.
  2. The function `(gdb) target sim (x)` is not supported.
  3. Altering endianness via a console GDB interface is not recommended.
-

#### 4.1. Invoking a console for debugging using Andes toolchains

On the **Windows Start Menu**, click “Start > All Programs > Andestech > AndeSight\_STD VERSION > Toolchains > **TOOLCHAIN**” to invoke the console.



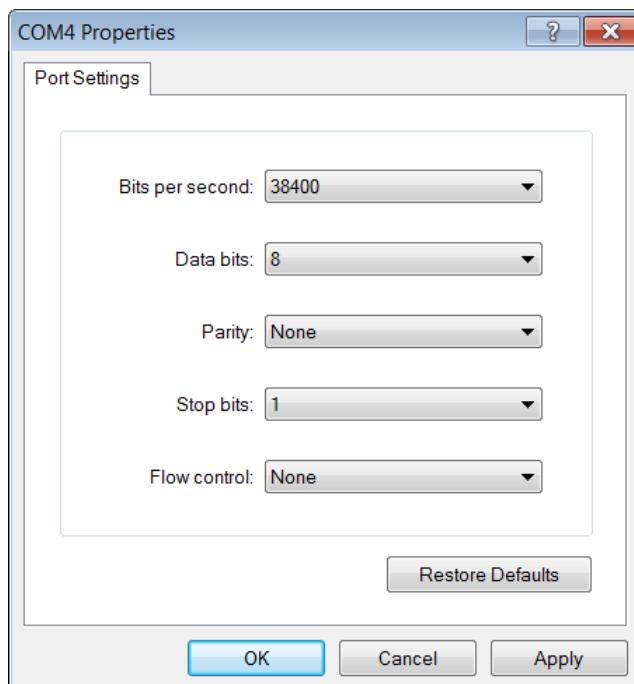
## 4.2. Debugging on ICE targets

### 4.2.1. Connecting the debug host to an ICE target

**Step 1** Install the target board (Network Cable, Ethernet, Power cord) and connect it to an ICE device. If you are using an Andes ICE (AICE), please refer to the *AICE Quick Start Guide* for instructions on installation.

**Step 2** Open a serial terminal program, such as Hyper Terminal or Minicom, and specify the com port for connection.

**Step 3** Set the port settings as follows:



**Step 4** Toggle on the power switch of the target board and press the “PWR ON” button.

**Step 5** Initiate the stand-alone Andes ICE controller program ICEman on the debug host and specify a port number for it. For a V5 target, you

will also need to specify the target using the option “**-Z v5**”, such as the following:

```
$ ./ICEman -Z v5 --port PORT_NUMBER
```

---

**NOTE**

1. To show the help messages for ICEman, enter \$ **ICEman -h**.
2. For V5 targets integrated with a Symmetric Multi-Processing cores, the option “**--smp**” must be applied too.

```
$ ./ICEman -Z v5 --port PORT_NUMBER --smp
```

---

#### 4.2.2. Compiling source programs

To compile your program, issue the following commands:

```
$ ./[riscv32|riscv64]-elf-gcc -static -o  
WORKSPACE_ROOT/PROJECT/PROJECT_EXECUTABLE  
WORKSPACE_ROOT/PROJECT/PROJECT_SOURCE_PROGRAM
```

where **-static** refers to static linking.

#### 4.2.3. Debugging with GDB

**Step 1** Open a new console and initiate GDB.

```
$ ./[riscv32|riscv64]-elf-gdb  
WORKSPACE_ROOT/PROJECT/PROJECT_EXECUTABLE
```

**Step 2** Enter the port number assigned for ICEman to build the host-target connection (available port: 1025-9999, excluding 1234, 4444, 9897).

```
(gdb) target remote : PORT_NUMBER
```

**Step 3** Link **PROJECT\_EXECUTABLE** and load its symbols for debugger use.

```
(gdb) load
```

**Step 4** For target systems without SMP support, jump to the next step. For V5 targets with SMP support, set the program counter of each core except core 1 to the entry point of the ELF executable. For example,

in a dual-core SMP system, you may follow below to set the program counter of core 2 to the ELF entry point.

```
(gdb) thread 2  
(gdb) set $pc=ELF_ENTRY  
(gdb) thread 1
```

---

**NOTE**

In the case of demo-smp-V5 (Section 3.2.3.18), the ELF entry point is the `_start` symbol; for Zephyr demo enabled with the SMP feature (Section 2.4.12.2), the ELF entry point is the `entry` symbol.

---

**Step 5 Execute the program.**

```
(gdb) c
```

#### 4.2.4. Typical debugging commands

- Set a breakpoint at the main function: `(gdb) b main`
- Continue program execution after suspension at the breakpoint: `(gdb) c`
- Pause a running program (applicable to ICE targets only): `Ctrl +c`

For a complete list of GDB command options, consult the online GDB manual.

### 4.3. Debugging on simulator targets

**Step 1** Copy the exported configuration file **PROJECT.conf** from **WORKSPACE\_ROOT\.metadata\vep.conf** to a directory of your choice, such as **ANDESIGHT\_ROOT\CONF\_DIRECTORY\**. Make sure that the toolchain of the configuration file matches that of the desired application program.

**Step 2** Under the AndeSight root directory, compile your project program:

```
$ ./[riscv32|riscv64]-elf-gcc -static -o
WORKSPACE_ROOT/PROJECT/PROJECT_EXECUTABLE
WORKSPACE_ROOT/PROJECT/PROJECT_SOURCE_PROGRAM
```

**Step 3** Open a new console and initiate the simulator.

```
$ ./sid ANDESIGHT_ROOT/CONF_DIRECTORY/PROJECT.conf
```

In the message that the simulator generates upon activation, take note of the port number.

```
socketiobase: server at 0.0.0.0: PORT_NUMBER
```

For example,

```
$ ./sid .. /toolchains/nds32le-elf-mculib-v5/config/nds32-target-config.conf
```

```
socketiobase: using fd 4
```

```
socketiobase: server at 0.0.0.0: 9898
```

```
GDB init ...
```

```
VEPsocketio: init ...
```

```
socketiobase: using fd 5
```

```
socketiobase: server at 0.0.0.0: 9899
```

In this example, the port number for the simulator is **9898**.

**Step 4** Open another console and initiate GDB.

```
$ ./[riscv32|riscv64]-elf-gdb
WORKSPACE_ROOT/PROJECT/PROJECT_EXECUTABLE
```

**Step 5** Enter the port number for the simulator to build a connection to the simulator.

(gdb) target remote [HOST\_IP]: PORT\_NUMBER

For example,

(gdb) target remote :9898

Remote debugging using: 9898

0x00000000 in ?? ()

**Step 6** Link **PROJECT\_EXECUTABLE** and load its symbols for debugger use.

(gdb) load

**Step 7** Set a breakpoint at the main function:

(gdb) b main

**Step 8** Continue program execution after suspension at the breakpoint:

(gdb) c

For other command options, please consult online GCC and GDB manuals.

## 4.4. Headless build

The headless build feature enables you to run the AndeSight IDE to build pre-configured, uncompressed workspace projects from command line. This is useful if you want to automate the builds.

To perform a headless build for an AndeSight project, issue the following commands:

```
. ./AndeSight --launcher.suppressErrors \
-nosplash \
-application org.eclipse.cdt.managedbuilder.core.headlessbuild \
-data ${WORKSPACE_PATH} \
-import ${PROJECT_PATH} \
-cleanBuild ${PROJECT_NAME}/${BUILD_CONFIGURATION_NAME} \
```

The following is an example to build a JPEG-V5 project on Linux:

```
. ./AndeSight --launcher.suppressErrors -nosplash -application
org.eclipse.cdt.managedbuilder.core.headlessbuild -data
/test/AST50-workspace/ -import /test/AST50-workspace/JPEG-V5/ -cleanBuild
JPEG-V5/AE350_RV32
```

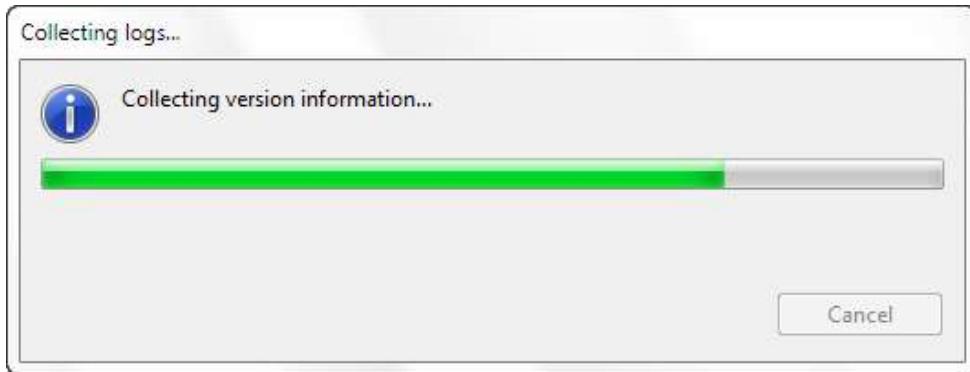
On Windows, you will need one more option “**-console**” to print out the build process. For example,

```
. ./AndeSight --launcher.suppressErrors -console -nosplash -application
org.eclipse.cdt.managedbuilder.core.headlessbuild -data
D:\ASTv50-workspace\ -import D:\ASTv50-workspace\JPEG-V5\ -cleanBuild
JPEG-V5/AE350_RV32
```

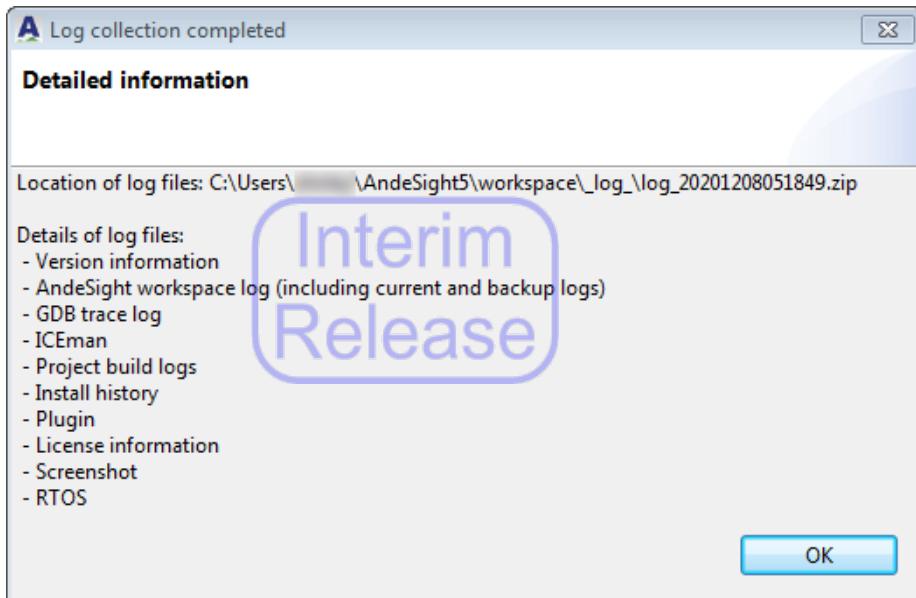
## 5. Tips and tricks

### 5.1. Collecting diagnostic information via a hotkey

A number of unexpected behaviors may occur when you use AndeSight with a development board or AndeSight simulator. The reasons for these could be an unstable development board, cable connection issues, incomplete installation, incorrect settings in AndeSight, outdated components, unsuccessful compilation, and so on. If you run into a reproducible problem, then use the hotkey “Ctrl+Shift+F10” to generate diagnostic information. The information can be used by Andes support personnel to identify the cause of your problem and thereby expedite the troubleshooting process.



When the collection of information has been completed, a dialog such as the one below pops up showing that the relevant logs and the screenshot are compressed within a zip file under **WORKSPACE\_ROOT\log\**. Provide Andes support personnel with this zip file or upload it to the Andes issue-tracking system. You will be contacted promptly with a response.

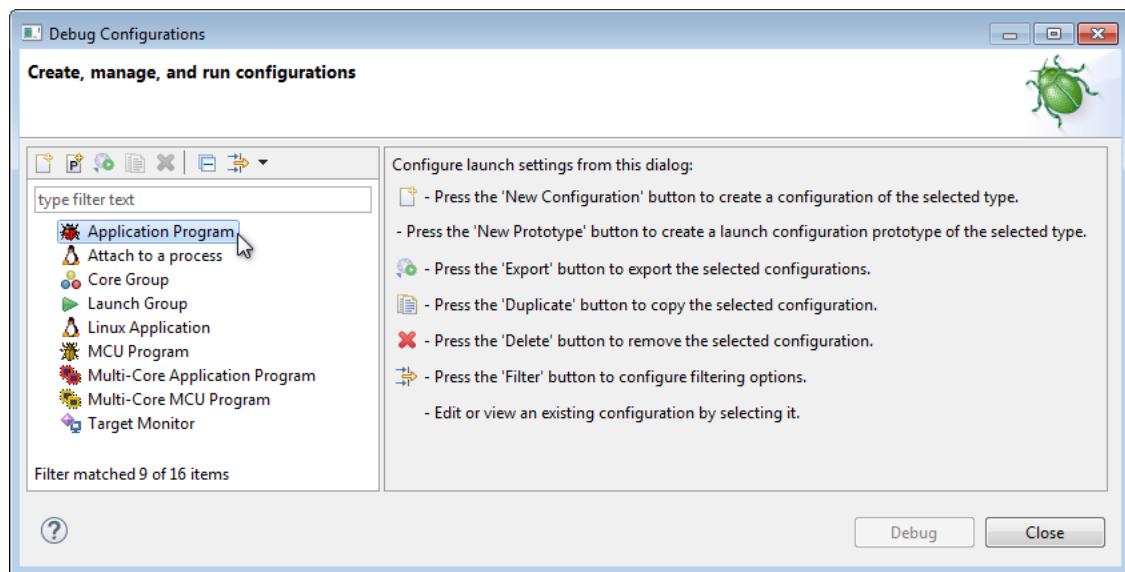


## 5.2. Directing console outputs to text files

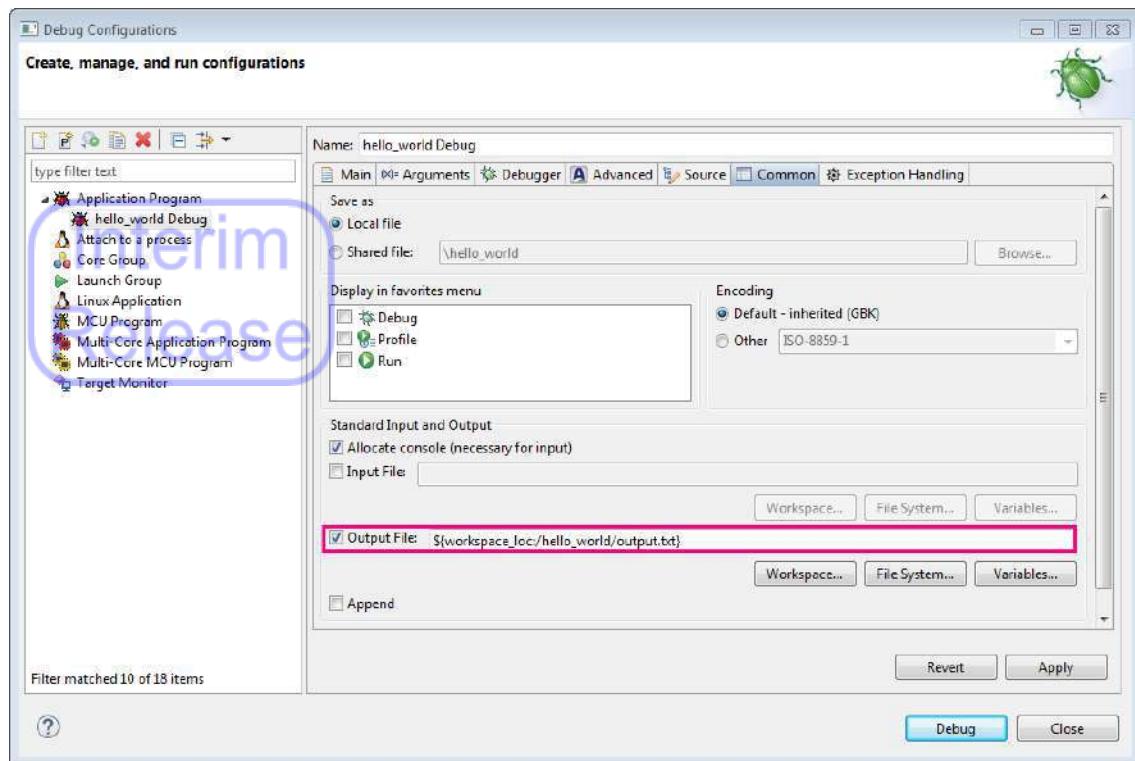
**Step 1** Right-click the desired project and select “[Run|Debug|Profile] As > [Run|Debug|Profile] Configurations...” from the pull-down menu to invoke the Run/Debug/Profile Configurations dialog.



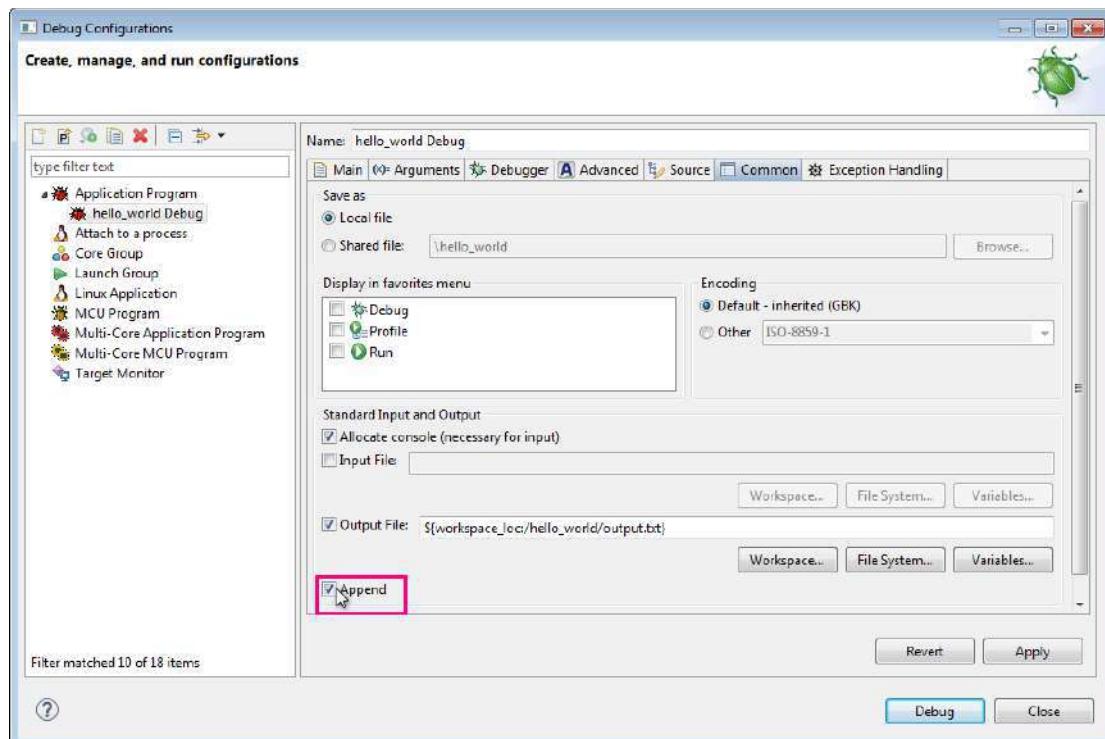
**Step 2** In the **Debug Configurations** dialog, double-click on a configuration type to create a debug configuration.



**Step 3** Go to the **Common** tab. Select the “Output File” option from the “Standard Input and Output” section. Click “Workspace...” or “File Systems ...” to assign a text file for the re-directed output.



**Step 4** To avoid overwriting existing output data with new output data, select the “Append” option to append them to a designated output file and click “Apply.”



### 5.3. Changing default compiler/linker options in chip profiles

If compiler or linker options configured in your selected chip profile fail to meet your project requirements, then you may edit the targetboard.properties file to include the desired compiler/linker options in the “CPU Description” section and click Ctrl+S to save the changes. For example, you can specify compiler and linker options for compression optimization in the targetboard.properties file of the chip profile “ADP-AE250-N25” as follows:

```
#== Chip Profile ==
chip.id=ADP-AE250-N25

#== CPU Description ==
cpu.cores=1
cpu0.type=N25
cpu0.isa.reducereg=1
cpu0.registers=../../CPU/N25.crgs
cpu0.c_compiler_opt=-mexecit
cpu0.cpp_compiler_opt=-mexecit
cpu0.linker_opt=--mexecit
```

The targetboard.properties file of a chip profile is placed under **ANDESIGHT\_ROOT\target\TARGET**. For detailed descriptions of targetboard.properties files, please refer to Section 2.2.1.1.