Explore more resources
Altera® Design Hub

altera
An Intel Company

Ashling* RiscFree* Integrated Development Environment (IDE) for Altera® FPGAs User Guide

Updated for Quartus® Prime Design Suite: **25.1**

Online Version
Send Feedback

730783
2025.04.11

Contents

1. About this Document..... 4

2. Ashling RiscFree IDE for Altera FPGAs..... 5

2.1. About the RiscFree IDE for Altera FPGAs IDE..... 5
2.1.1. Supported Devices..... 6
2.1.2. Quartus Prime Software Support..... 6
2.1.3. Intel Simics® Simulator for Intel FPGAs Support..... 6
2.2. Getting Started with the Ashling RiscFree IDE for Altera FPGAs..... 7
2.2.1. Installing RiscFree IDE for Altera FPGAs..... 7
2.2.2. Getting Started with RiscFree IDE for Altera FPGAs..... 10
2.2.3. Creating the Project..... 12
2.2.4. Building the Application..... 13
2.2.5. Run and Debug Configurations in the RiscFree IDE for Altera FPGAs..... 13
2.2.6. Debug Information in the RiscFree IDE for Altera FPGAs..... 15
2.3. Using Ashling RiscFree IDE for Altera FPGAs with Nios V Processor System..... 18
2.3.1. Importing Nios V Processor Project..... 18
2.3.2. Building Nios V Processor Project..... 19
2.3.3. Running a Nios V Processor Project..... 20
2.3.4. Debugging a Nios V Processor Project..... 23
2.3.5. Debugging Tools..... 33
2.4. Using Ashling RiscFree IDE for Altera FPGAs with Arm Hard Processor System..... 38
2.4.1. Pre-requisite for Debugging Arm HPS Project..... 38
2.4.2. Importing Arm HPS Project..... 38
2.4.3. Setting Debug Configurations and Downloading Arm HPS Project Using RiscFree IDE..... 39
2.5. Debugging Features with RiscFree IDE for Altera FPGAs..... 41
2.5.1. Debug Features in RiscFree IDE..... 41
2.5.2. Processor System Debug..... 43
2.5.3. Heterogeneous Multicore Debug..... 44
2.5.4. Debugging µC/OS-II Application..... 51
2.5.5. Debugging FreeRTOS Application..... 51
2.5.6. Debugging Zephyr Application..... 52
2.5.7. Arm HPS On-Chip Trace..... 54
2.5.8. Debugging the Arm Linux Kernel..... 60
2.5.9. Debugging Target Software in an Intel Simics Simulator Session..... 69

3. Ashling Visual Studio Code Extension for Altera FPGAs..... 70

3.1. About the Ashling Visual Studio Code Extension..... 70
3.1.1. Supported Devices..... 70
3.1.2. Quartus Prime Software Support..... 71
3.2. Getting Started with Ashling Visual Studio Code Extension..... 71
3.2.1. Installing Ashling Visual Studio Code Extension..... 71
3.2.2. Getting Started with Ashling Visual Studio Code Extension..... 75
3.2.3. Debug Configurations using Ashling Visual Studio Code Extension for Altera FPGAS..... 79
3.2.4. Debug Information in the Ashling Visual Studio Code Extension for Altera FPGAs..... 80
3.3. Using Ashling Visual Studio Code Extension with Nios V Processor System..... 83

3.3.1. Creating Nios V Processor BSP using Nios V Processor BSP Generator..... 83
3.3.2. Creating Nios V Processor Application Project using Nios V App Generator 85
3.3.3. Importing Nios V Processor Project.....88
3.3.4. Building Nios V Processor Project..... 89
3.3.5. Debugging a Nios V Processor Project..... 90
3.3.6. Debugging Tools..... 97
3.4. Using Ashling Visual Studio Code Extension with Arm Hard Processor System.....102
3.4.1. Pre-requisite for Debugging Arm HPS Project..... 102
3.4.2. Importing Arm HPS Project.....102
3.4.3. Setting Debug Configurations and Downloading Arm HPS Project Using Visual Studio Code..... 103
3.5. Debugging Features in Ashling Visual Studio Code Extension.....106
3.5.1. Debug Features in Ashling Visual Studio Code Extension.....106
3.5.2. Nios V Processor System Debug..... 110
3.5.3. Heterogeneous Multicore Debug..... 111

4. Ashling RiscFree Integrated Development Environment (IDE) for Altera FPGAs User Guide Archives.....115

5. Document Revision History for the Ashling RiscFree Integrated Development Environment (IDE) for Altera FPGAs User Guide.....116

A. Appendix..... 119
A.1. Optional Toolchain Configuration in Ashling RiscFree IDE for Altera FPGAs.....119
A.1.1. Toolchain for Application and BSP.....119
A.1.2. Toolchain for Arm Processor.....119
A.1.3. Modifying Toolchain Configuration.....120
A.2. Performing External Tools Configuration in Ashling RiscFree IDE for Altera FPGAs.....121
A.3. Debugging with Command-Line Interface.....122
A.3.1. Running Ashling GBD Server.....122
A.3.2. Running GDB.....124
A.3.3. Nios V Processor Example..... 124
A.3.4. ARM HPS Core Example..... 126
A.4. Help Content.....128
A.5. Open-Source Components in RiscFree IDE.....128

730783 | 2025.04.11

Send Feedback

1. About this Document

This document outlines the Ashling* RiscFree* IDE for Altera® FPGAs in with the Visual Studio Code Extension, focusing on debugging HPS and Nios® V processor applications projects. It details the capabilities and features of both tools, including their interfaces, debugging functionalities, and integration methods. Additionally, the document provides instructions for configuring and utilizing these tools to effectively manage and debug application projects on HPS and Nios V processors.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

2. Ashling RiscFree IDE for Altera FPGAs

2.1. About the RiscFree IDE for Altera FPGAs IDE

RiscFree IDE for Altera FPGAs is Ashling's Eclipse* C/C++ Development Toolkit (CDT) based integrated development environment (IDE) for Altera FPGAs Arm*-based HPS and RISC-V based Nios V processors.

The RiscFree IDE provides a complete, seamless environment for C and C++ software development and has the following features:

- Eclipse CDT based IDE with full source and project creation, editing, build, and debug support using the RISC-V GNU compiler collection (GCC) toolchain.
- Project Manager and Build Manager including Make and CMake support with rapid import, build, and debug of application frameworks created using the Quartus® Prime software.
- RISC-V GNU GCC toolchain with support for newlib or picolibc run-time libraries using the Nios V Hardware Abstraction Layer (HAL) API for hardware access.
- Integrated support for Intel® FPGA Download Cable II JTAG debug probe.
- ROM or RAM based debugging support, for example, hardware breakpoints for flash-based support.
- High-level Register Viewer based on industry standard System View Description (SVD) files.
- Integrated serial terminal.

Related Information

- Nios V Embedded Processor Design Handbook
  Describes how to most effectively use the tools, recommends design styles, and practices for developing, debugging, and optimizing embedded systems using the Nios V processor and Intel-provided tools (Quartus Prime Pro Edition User Guide).
- Nios V Processor Reference Manual
  Provides information about the Nios V processor performance benchmarks, processor architecture, the programming model, and the core implementation (Quartus Prime Pro Edition User Guide).
- Nios V Processor Software Developer Handbook
  Describes the Nios V processor software development environment, the tools that are available, and the process to build software to run on Nios V processor (Quartus Prime Pro Edition User Guide).
- AN 980: Nios® V Processor Intel® Quartus® Prime Software Support
- Eclipse Documentation

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

2.1.1. Supported Devices

The following devices support Nios V processor core debugging:

- Agilex™ 5
- Agilex 7
- Stratix® 10
- Stratix V
- Stratix IV
- Arria® 10
- Arria II GX
- Arria II GZ
- Arria V
- Arria V GZ
- Cyclone® 10 GX
- Cyclone IV E
- Cyclone IV GX
- Cyclone V
- Cyclone 10 LP
- MAX® 10

The following devices with SoC feature support Arm HPS debugging:

- Agilex 5
- Agilex 7
- Stratix 10
- Arria 10
- Cyclone V

2.1.2. Quartus Prime Software Support

You can use Ashling RiscFree IDE for Intel FPGAs with Quartus Prime Standard Edition or Quartus Prime Pro Edition software.

2.1.3. Intel Simics® Simulator for Intel FPGAs Support

The Intel Simics® Simulator for Intel FPGAs is a full-system simulator supporting the software development process. It provides capabilities that include hardware and software inspection, dynamic system configuration, hardware modeling tools, scripting, and other features.

Ashling RiscFree IDE for Altera FPGAs provides a graphical interface to the Intel Simics simulator for Intel FPGAs software that allows you to:

- Control the simulation progress with a mouse and buttons in a GUI instead of entering commands on a CLI (which is also supported).
- Debug target software using the Ashling RiscFree IDE for Altera FPGAs debugger to view progress through your source code, set breakpoints, examine variable values and register contents, and view stack traces.
- Integrate Ashling RiscFree IDE for Altera FPGAs and Intel Simics simulator as a complete development environment for your target software. You can write your target program, test it, run simulations, and debug it all in one location on your target hardware.

Related Information

Intel® Simics® Simulator for Intel® FPGAs: User Guide

2.2. Getting Started with the Ashling RiscFree IDE for Altera FPGAs

RiscFree IDE for Altera FPGAs provides a consistent development platform that works for all Nios V embedded processor systems. You can accomplish all Nios V processor software development tasks within RiscFree IDE, including creating, editing, building, running, and debugging programs. RiscFree IDE for Altera FPGAs supports Nios V processor and Arm core debugging in multiple host operating system (OS) platforms.

Note: If you install both the Quartus Prime software and the Quartus Prime Programmer and Tools in the same location as RiscFree IDE for Altera FPGAs IDE, it refers to the required binaries from the Quartus Prime software installation directory by default.

2.2.1. Installing RiscFree IDE for Altera FPGAs

RiscFree IDE has two types of installation options:

- Bundled installation option when installing the Quartus Prime software
- Standalone installation which requires you to install one of the following to use the debugger:
  - The Quartus Prime Programmer and Tools
  - The Quartus Prime software

Related Information

Operating System Support

2.2.1.1. Bundled Installation with the Quartus Prime Software

The RiscFree IDE is available in the Quartus Prime software installation package as an optional installation component. For a bundled installation, select the **RiscFree IDE for Intel FPGAs** component during the Quartus Prime Pro Edition software installation.

Note: The RiscFree IDE for Altera FPGAs is not bundled with the 22.1std release of Quartus Prime Standard Edition. Refer to the following sections for the RiscFree IDE for Altera FPGAs standalone installation.

Figure 1. Selecting Components During Bundled Installation

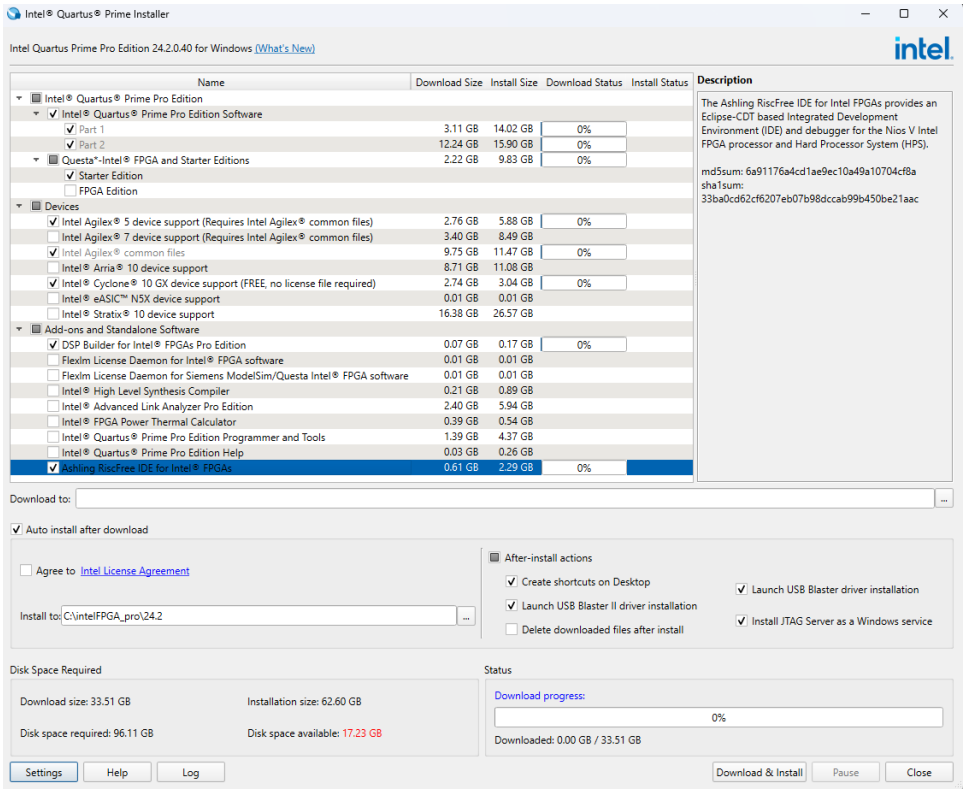


Table 1. Installation Directory for Bundled Installation with the Quartus Prime Software

Operating System	Linux*	Windows*
RiscFree IDE Installation Directory	`<Intel Quartus Prime installation directory>/riscfree/`	`<Intel Quartus Prime installation directory>/riscfree/`
Quartus Prime Binary Files Directory	`<Intel Quartus Prime installation directory>/quartus/linux64/…`	`<Intel Quartus Prime installation directory>/quartus/bin64/…`

Related Information

FPGA Software Download Center
Provides the latest version of the Quartus Prime Design Software.

2.2.1.2. Standalone Installation

You can download the RiscFree IDE for Altera FPGAs from the *FPGA Software Download Center*.

1. Select the latest version of Quartus Prime design software.
2. Scroll down to *Downloads*, and click the *Additional Software* tab.
3. Scroll down to *Stand-Alone Software*.
4. Find RiscFree IDE for Altera FPGAs and click *Download*.

Related Information

FPGA Software Download Center
Provides the latest version of the Quartus Prime Design Software.

2.2.1.2.1. Standalone Installation with Supported Quartus Prime Software

You can install the standalone RiscFree IDE for your system that has a pre-installed Quartus Prime software.

The standalone RiscFree IDE debug utility requires the Quartus Prime binaries from the installed Quartus Prime software directory: `../<Intel Quartus Prime installation directory>`. Install the RiscFree IDE in the same path as the Quartus Prime software. If you select a different path, you receive the following message:

Figure 2. Installation Warning Message

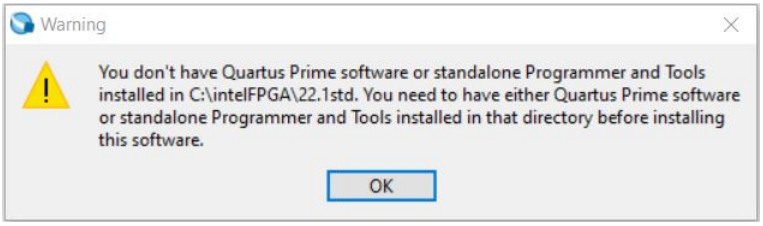


Table 2. Installation Directory for Standalone Installation with the Quartus Prime Software

Operating System	Linux	Windows
RiscFree IDE Installation Directory	`<Intel Quartus Prime installation directory>/riscfree/`	`<Intel Quartus Prime installation directory>/riscfree/`
Quartus Prime Software Directory	`<Intel Quartus Prime installation directory>`	`<Intel Quartus Prime installation directory>`

2.2.1.2.2. Standalone Installation with the Quartus Prime Programmer and Tools

You can install the standalone RiscFree IDE with the Quartus Prime Programmer and Tools.

The standalone RiscFree IDE debug utility requires the Quartus Prime Programmer and Tools binaries from the installed Quartus Prime Programmer and Tools directory. Install the RiscFree IDE in the same path as the Quartus Prime Programmer and Tools: `../<Quartus_programmer_installation_directory>`. If you select a different path, you receive an error message. Refer to Figure Installation Warning Message for the error message.

Table 3. Installation Directory for Standalone Installation with the Quartus Prime Programmer and Tools

Operating System	Linux	Windows
RiscFree IDE Installation Directory	*<Quartus_programmer_installation_directory>*/riscfree/	*<Quartus_programmer_installation_directory>*/riscfree/
Quartus Prime Programmer Directory	*<Quartus_programmer_installation_directory>*/qprogrammer/quartus/linux64/…	*<Quartus_programmer_installation_directory>*/qprogrammer/quartus/bin64/…

Note: Nios V processor tools are not available for standalone installation with the Quartus Prime Programmer and Tools.

2.2.2. Getting Started with RiscFree IDE for Altera FPGAs

Writing software for the Nios V processor is like writing software for any other microcontroller family. An easy way to start designing effectively is to purchase an Altera FPGA development kit that includes documentation, a ready-made evaluation board, a getting-started reference design, and all the development tools necessary to write Nios V processor programs.

This section guides you through the most fundamental operations in the RiscFree IDE for Altera FPGAs and the following aspects:

- How to create an application project for Nios V processor.
- How to import an application project and the board support package (BSP) project required to interface with your hardware.
- How to build the application in Ashling RiscFree IDE for Altera FPGAs.
- How to run and debug the software on an FPGA development board.

Related Information

Nios® V Processor Software Developer Handbook: Nios® V Processor Software Development Flow
For more information about Nios® V Processor Software Development Flow. The chapter describes BSP generation from the hardware system and CMakeLists file generation for application before importing the binaries into Ashling RiscFree IDE for Altera FPGAs.

2.2.2.1. Ashling RiscFree IDE for Altera FPGAs Workbench

The term workbench refers to the RiscFree IDE for Altera FPGAs desktop development environment. The workbench is where you edit, compile, and debug your programs in RiscFree IDE for Altera FPGAs.

2.2.2.1.1. Perspective, Editors, and Views

Each workbench window contains one or more perspectives. Each perspective provides a set of capabilities for accomplishing a specific type of task.

Most perspectives in the workbench comprise an editor area and one or more views. An editor allows you to open and edit a project resource (i.e., a file, folder, or project). Views support editors and provide alternative presentations and ways to navigate the information in your workbench.

Any number of editors can be open at once, but only one can be active at a time. The main menu bar and toolbar for the workbench window contain operations that are applicable to the active editor. Tabs in the editor area indicate the names of resources that are currently open for editing. An asterisk (*) indicates that an editor has unsaved changes. Views can also provide their own menus and toolbars, which, if present, appear along the top edge of the view. To open the menu for a view, click the drop-down arrow icon at the right of the view's toolbar or right-click in the view. A view might appear on its own or stacked with other views in a tabbed notebook

For detailed information about the RiscFree IDE for Altera FPGAs workbench, perspectives, and views, refer to the RiscFree IDE for Altera FPGAs help system.

2.2.2.1.2. Starting a Project

You can start a project in Ashling RiscFree IDE for Altera FPGAs, either by creating a new project or import an existing project. The **Project Explorer** offers a list of popular approaches to kick start your project

2.2.2.1.3. Navigating the Project

When you created a Nios V processor project, it appears in the **Project Explorer** view. You can expand each project to examine its folders and files.

2.2.2.1.4. Building the Project

When building a Nios V project, you can refer to the following views:

- **Console** view displays the build progress.
- **Make Targets** view displays the makefile action.
- **Problems** view displays the compile warnings or errors.

2.2.2.1.5. Configuring the FPGA

To configure the FPGA, use the Quartus Prime Programmer from the Quartus Prime software.

The project directory for your hardware design contains an SRAM Object File (`.sof`) and the `.qsys` file. The `.sof` file contains the hardware design to be programmed in the FPGA.

Related Information

- Quartus® Prime Pro Edition User Guide: Programmer
  For more information about programming an FPGA with Quartus Prime Pro Edition Programmer.
- Quartus® Prime Standard Edition User Guide: Programmer
  For more information about programming an FPGA with Quartus Prime Standard Edition Programmer.

2.2.2.1.6. Running the Project

You can run a Nios V processor program using the RiscFree IDE for Altera FPGAs on Run Configurations, such as an Intel FPGA development board.

To run a software project, navigate to **Run** and click **Run Configurations**. This command conducts the following actions:

- Creates an Ashling RISC-V Hardware Debugging.
- Builds the project executable. If all target files are up to date, nothing is built.
- Establishes communications with the target, and verifies that the FPGA is configured with the correct hardware design.
- Downloads the Executable and Linking Format File (.elf) to the target memory
- Starts execution at the .elf entry point.

To disconnect the terminal from the target, click the **Terminate** icon in the **Console** view. Terminating only disconnects the host from the target. The target processor continues executing the program.

2.2.2.1.7. Debugging the Project

To debug a software project, navigate to **Debug** and click on **Debug Configurations**. This command conducts the following actions:

- Creates an Ashling RISC-V Hardware Debugging.
- Builds the project executable. If all target files are up to date, nothing is built.
- If debugging on hardware, establish communications with the target and verify that the FPGA is configured with the correct hardware design.
- Downloads the `.elf` to the target memory.
- Sets a breakpoint at the top of main().
- Starts execution at the `.elf` entry point.

The RiscFree IDE for Altera FPGAs debugger with the Nios V processor plugins provides a perspective, allowing you to perform many common debugging tasks. Debugging a Nios V processor program with the Nios V plugins is generally the same as debugging any other C/C++ program with RiscFree IDE for Altera FPGAs and the CDT plugins.

For information about debugging with RiscFree IDE for Altera FPGAs and the CDT plugins, refer to the RiscFree IDE for Altera FPGAs help system.

2.2.3. Creating the Project

2.2.3.1. Specifying the Project Location

The project location is the parent directory in which the RiscFree IDE for Altera FPGAs creates the project folder. By default, it is in the same directory as the RiscFree IDE for Altera FPGAs's workspace directory.

To place your application project in the created folder, turn off **Use default location**, and specify the path in the **Location** field.

2.2.3.2. Specifying the Project Name

Select a descriptive name for your project. The RiscFree IDE for Altera FPGAs imports the folder created previously to contain the application project files. You can still rename the project if needed. Follow the naming convention below when specifying the project name:

- The only valid project name characters are letters, numbers, and the underscore (_) symbol.
- Project names cannot contain spaces or special characters.
- The first character in the project name must be a letter or underscore. The maximum filename length is 250 characters.

2.2.3.3. Specifying the Project Type

Select the project type that is appropriate to your project. The project type determine the toolchain, data, and tab that the IDE uses.

2.2.3.4. Specifying the Configurations

Select the configurations that you want to deploy on your project. By default, the items in the list of all build configurations to be created are checked.

2.2.4. Building the Application

To build a Nios V project in the RiscFree IDE for Altera FPGAs, right-click the project name and click **Build Project**. Depending on the speed of the host machine, the build process can take a minute or two for a simple project, while building a complex project takes longer. You can monitor the build status using the progress bar. During the build process, you can view the build commands and command-line output in the Eclipse **Console** view. The following message appears in the **Console** view when the build process is complete.

If the project has a dependency on another project, such as a BSP or a user library, the IDE builds the dependency project first. This feature allows you to build an application and its BSP with a single command.

2.2.5. Run and Debug Configurations in the RiscFree IDE for Altera FPGAs

2.2.5.1. Opening the Run and Debug Configuration Dialog Boxes

You can open the run and debug configuration dialog boxes in two options:

- Right-click an application, point to **Run As**, and click **Run Configurations**.
- Right-click an application, point to **Debug As**, and click **Debug Configurations**.

Depending on which option you use, the dialog box title is either **Run Configurations** or **Debug Configurations**. However, both views show the same run configurations. Each run configuration appears on several tabs. The following table describes each tab.

Table 4. Run (Debug) Configuration

Tab	Description
Main	• This tab allows you to specify the application project to run. • You can control the runtime parameters in the following ways: — Specify the name for the configuration — Specify the name of the project — Specify the path of the application — Select build configuration based on the location of the application *Note:* The RiscFree IDE for Altera FPGAs sets these parameters to reasonable defaults. Do not modify them unless you have a clear understanding of their effects.
Debugger	• This tab allows you to control the connection between the host machine and the target hardware. • You can control the connection in the following ways: — Select the **Debug Probe** in the **Debug Probe Configuration**, if there is more than one debug probe available. — Click **Auto-detect Scan Chain** to detect JTAG scan chain information of the target device automatically. — Use GDB Server Setup to debug the software application remotely. — Use GDB Client Setup to communicate with the Ashling GDBServer via a TCP connection using GDB Remote Serial Protocol. — To modify GDB Server Setup and GDB Client Setup using the command line, you can refer to: • Running Ashling GBD Server for GDB Server Setup • Running GDB for GDB Client Setup *Note:* By default, the setup is generic. It can support all environments.
Startup	• This tab allows you to configure options related to debugging your program. Example of the options are loading images or symbols and setting breakpoints, which help you analyze and troubleshoot your program during execution. • You can use load image and symbol specifically for booting. — If you want to download your program to analyze and troubleshoot your program during execution, turn on both load image and symbols. — If your program is in booting mode and you want to analyze and troubleshoot your program during execution, turn off both load image and symbols. *Note:* Nios V processor supports hart ID of 0 only.
OS Awareness	This tab allows you to enable OS-aware debugging and specify the operating system to use to debug your design. The operating system are: • FreeRTOS • Linux, • Zephyr • µC/OS-II You can also specify the location of the Python script for the RTOS.
Source	You can specify the location of source files that are used when running or debugging a C/C++ application. By default, the source files are taken from your project.
Common	When a run configuration is created, it is saved with the extension `.launch`. You can select an alternate place for storing your run configuration, how to access it, and which perspective to use while executing an application.
SVD Path	The RiscFree IDE for Altera FPGAs supports Device-specific System View Description (SVD) JSON format files. The files provide the peripheral registers viewer with detailed definitions. Typically, they are obtained via the CMSIS Packs or xPacks that have been installed. You can use ZIP to compress the file.

2.2.6. Debug Information in the RiscFree IDE for Altera FPGAs

Debug perspective allows you to manage the debugging and execution of a program in the Workbench. You can control how your program is executed by establishing breakpoints, pausing started programs, walking through your code, and analyzing the contents of variables.

The Debug perspective provides the following information:

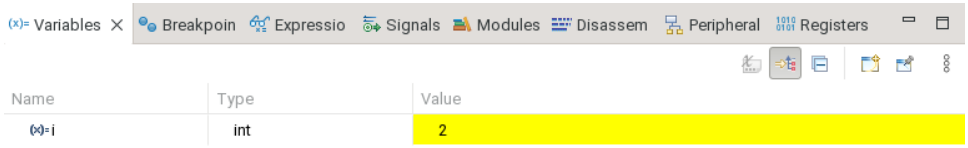
- Every target that you are debugging has its stack frame for the suspended threads
- Every thread in your program is represented as a node in the tree
- The process for each program that you are running

The Debug view also controls the C/C++ Editor. The C/C++ Editor highlights the execution pointer's location as you review your code.

2.2.6.1. Variables

The Variables view displays information about the variables in a specified stack frame. When execution is paused, the updated values are by default indicated in yellow. As with the other debug-related views, the view does not refresh when you run your program. When the execution is terminated, it is refreshed.

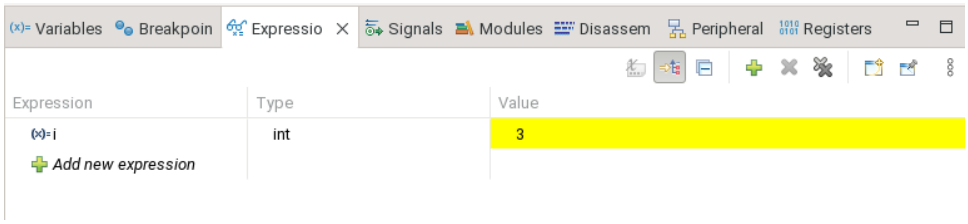
Figure 3. Variables View



2.2.6.2. Expressions

An expression is a code snippet that can be evaluated to yield an outcome. The specific debug model determines the context of an expression. Some expressions might need to be evaluated at a specific area in the program for the variables to be referenced. The **Expressions** view contains information about expressions.

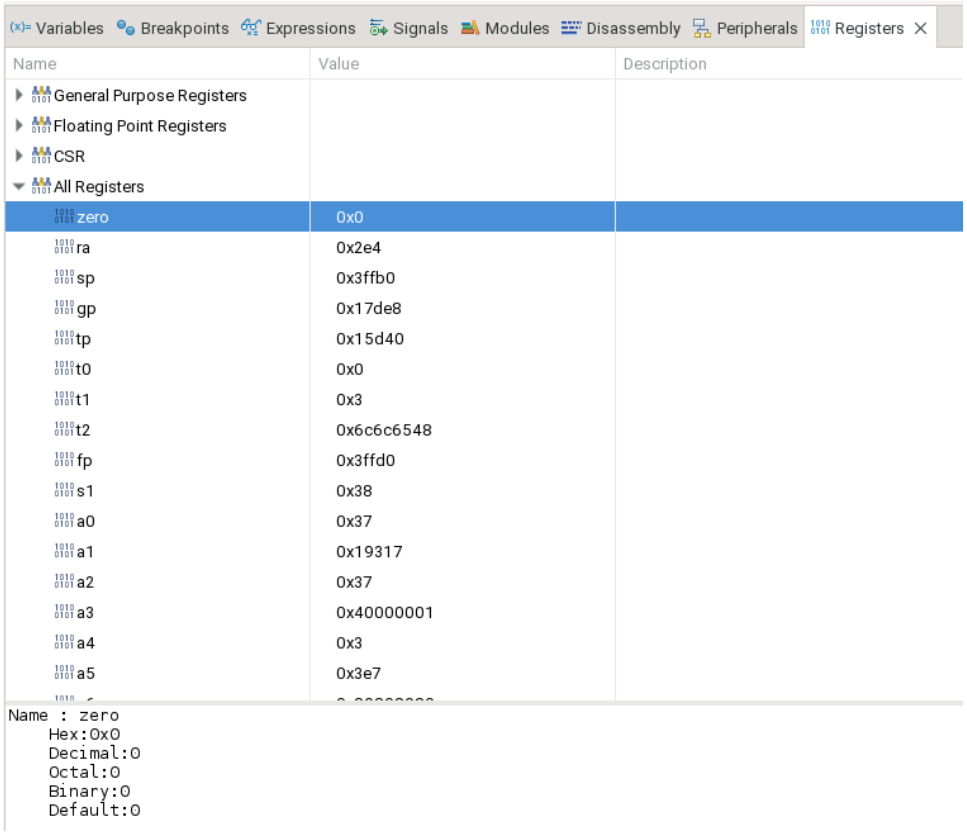
Figure 4. Expressions View



2.2.6.3. Registers

The Registers View feature allows you to see details about the registers in a chosen stack frame. When your program pauses or ends, the Registers view highlights the modified values.

Figure 5. Registers View



2.2.6.4. Memory

In **Memory** view, you can observe and modify the memory used by your processes. A list of so-called memory monitors represents the process memory. Every monitor depicts a memory region identified by its base address. Each memory monitor supports a variety of predetermined data forms known as memory renderings. The debugger supports five rendering types: signed integer, unsigned integer, ASCII, and hexadecimal (default). The monitor creation automatically displays the default rendering.

Figure 6. Memory View

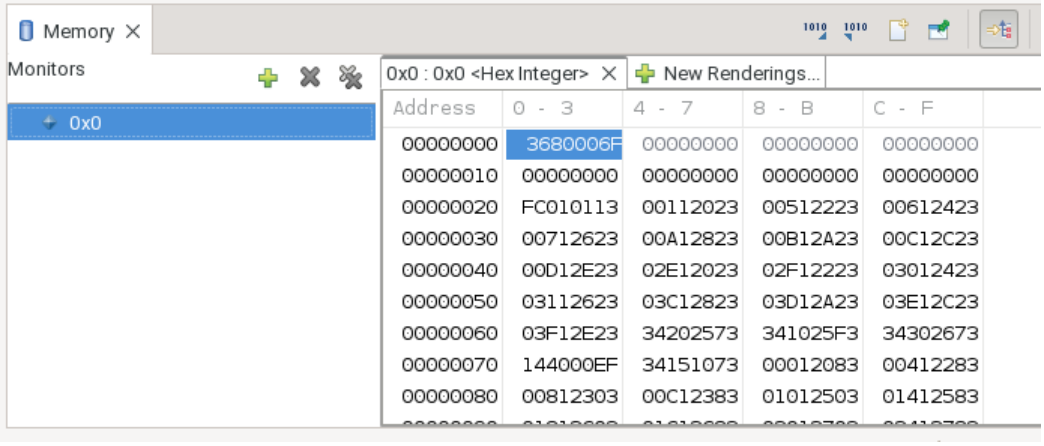


Table 5. Memory View and Description

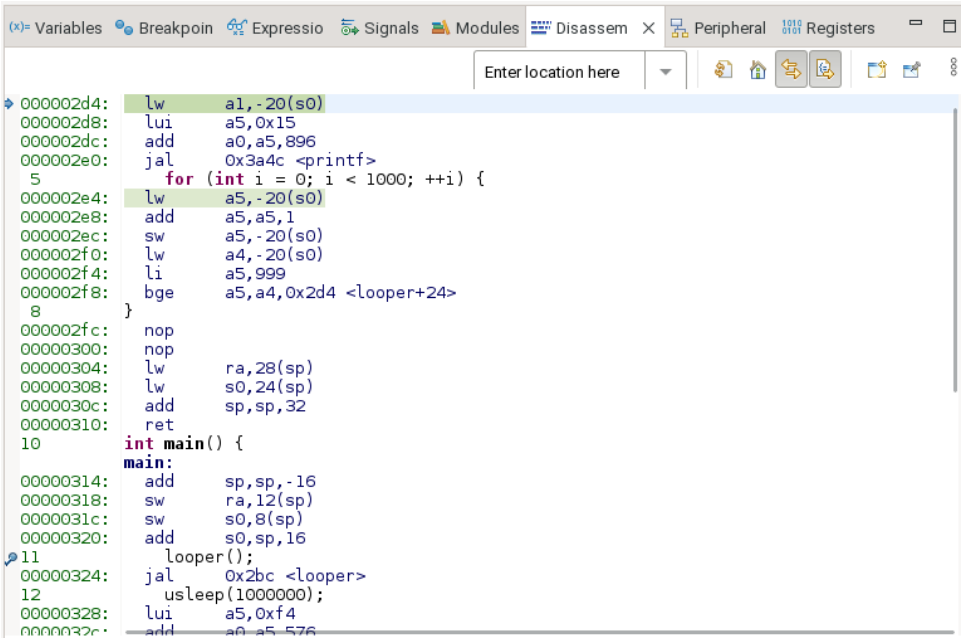
View	Description
Memory Monitors	Shows the list of memory monitors that have been added to the debug session that is currently chosen in the Debug view.
Memory Renderings	• Comprises the tabs that display renderings and are controlled by the selection in the Memory Monitors pane. • You can set it up to show two renderings simultaneously.

2.2.6.5. Disassembly

The Disassembly view displays the loaded program as assembler instructions mixed with source code for comparison. An arrow marker highlighted in the view shows the line that is currently running. It allows you to perform the following tasks:

- Set breakpoints at the start of any assembler instruction
- Enable and turn off breakpoints and set their properties
- Step through the disassembly instructions of your program
- Jump to specific instructions in the program

Figure 7. Disassembly View



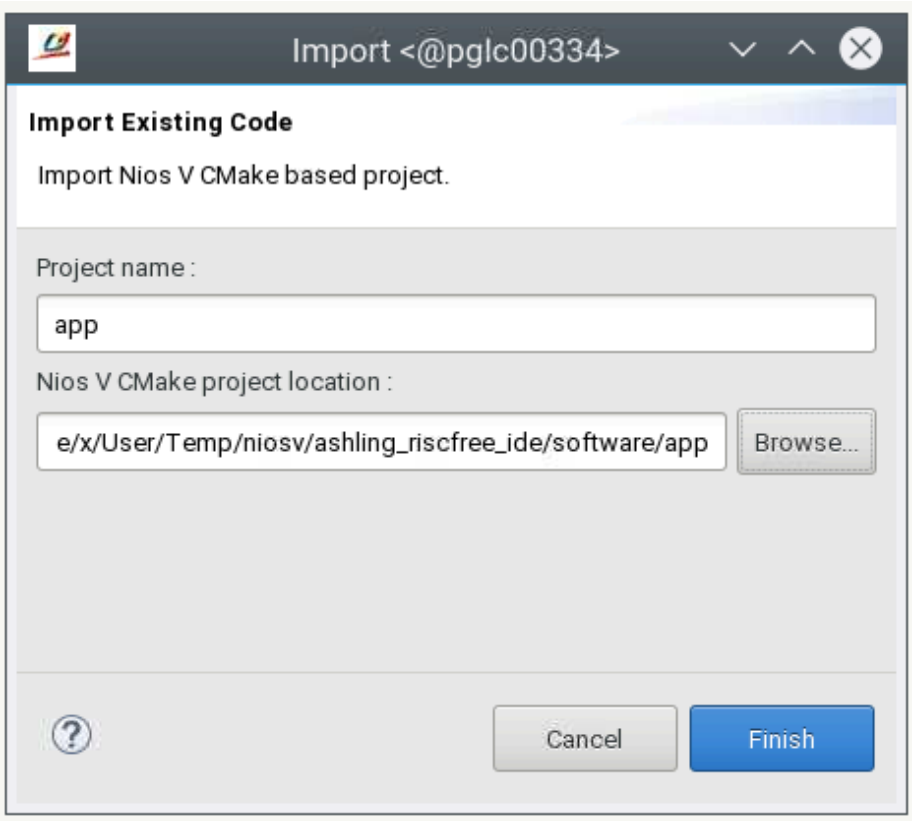
2.3. Using Ashling RiscFree IDE for Altera FPGAs with Nios V Processor System

2.3.1. Importing Nios V Processor Project

To import Nios V processor BSP and application project to RiscFree IDE, follow these steps:

1. Click **File ➤ Import Nios V CMake Project**.
2. In the **Import Nios V CMake Project** window, browse and select the location of the Nios V processor BSP or application project.
3. Click **Finish**. The CMake driven application or BSP project is added to the Project Explorer.
4. After you specify the location path, the project name is automatically filled in. However, you can still rename the project if needed.

Figure 8. Importing Nios V Processor Sample Project



Related Information

- AN 985: Nios® V Processor Tutorial
  A quick start guide about creating a simple Nios Vprocessor system and running the Hello World application. Refer to the tutorial for more information about generating BSP project and APP CMakeLists files.
- Nios® V Processor Software Developer Handbook: Nios® V Processor Software Development Flow
  For more information about Nios® V Processor Software Development Flow. The chapter describes BSP generation from the hardware system and CMakeLists file generation for application before importing the binaries into Ashling RiscFree IDE for Altera FPGAs.

2.3.2. Building Nios V Processor Project

You can browse the project files for Nios V processor project in RiscFree IDE Project Explorer tab. To build the project, follow these steps:

1. Right-click the project folder and click **Build Project**.
2. When the build is complete, you can find the console prints as shown in the following diagram.

Figure 9. Console Prints

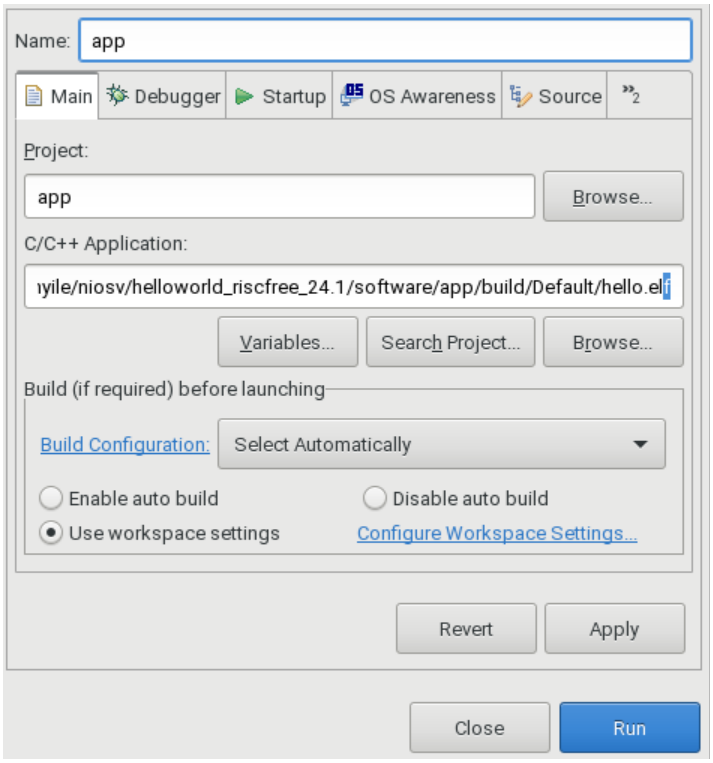


2.3.3. Running a Nios V Processor Project

You can download and run Nios V processor software project on the targeted Intel FPGA using the RiscFree IDE for Altera FPGAs. To run the project, follow these steps:

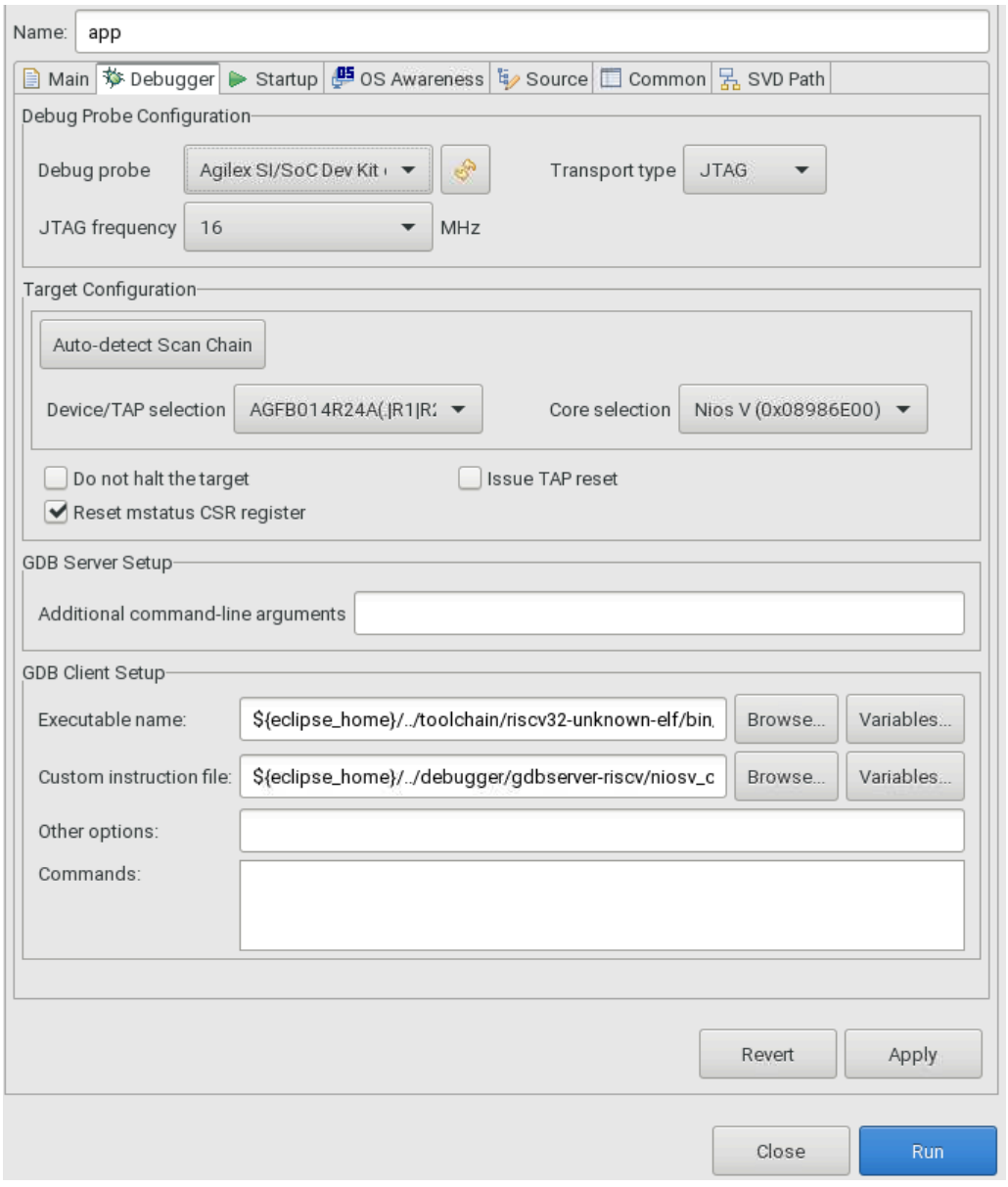
1. Right-click the project folder (application or BSP) in the project explorer and select **Run As ➤ Run configurations**.
2. Select **Ashling RISC-V Hardware Debugging ➤ <Project Name>**. Ensure the **Project** and **C/C++ Application** match your project name and your project `.elf` file, respectively.
3. In the **Main** tab, for **C/C++ Application**, browse to select the application build `.elf` file. For example: `hello.elf`.

Figure 10. Run Configurations for Nios V Processor—Main Tab



4. Under the **Debugger** tab, set these settings:
   - **Debug probe**: Agilex SI/SoC Dev Kit (Name of the FPGA board/development kit)
   - **Transport type**: JTAG
   - **JTAG frequency**: 16 MHz

Figure 11. Run Configurations for Nios V Processor—Debugger Tab

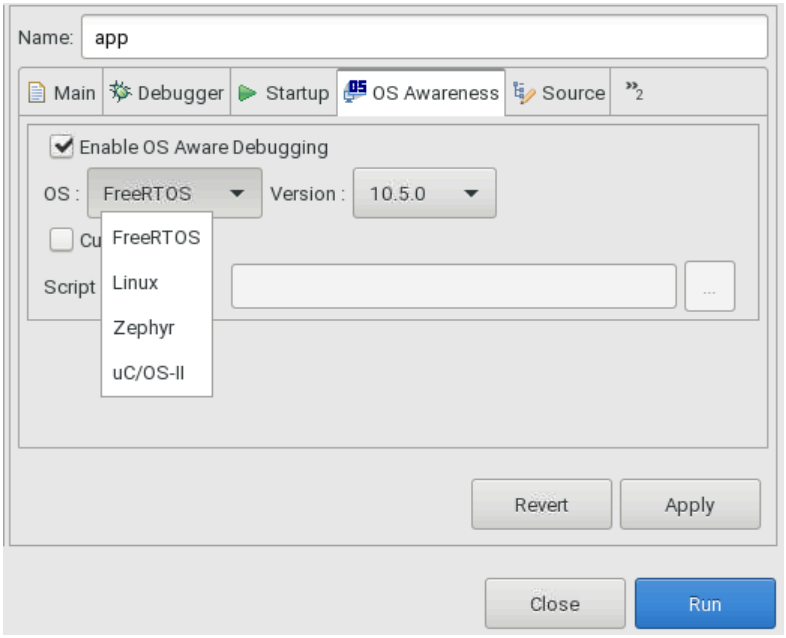


5. Click **Auto-detect Scan Chain** to automatically detect JTAG scan chain information of the target device. Select the options from **Device/Tap selection** and **Core selection**.
6. Based on the OS you use, configure the OS Awareness settings as follows:

- Intel HAL: No OS Awareness configuration is required.
- Other OS: Under the **OS Awareness** tab, turn on **Enable OS Aware Debugging**, and select the OS version applicable to you as listed below:
  - **OS**: µC/OS-II and **Version**: 2.93.0
  - **OS**: FreeRTOS and **Version**: 10.5.0
  - **OS**: Zephyr and **Version**: 3.2.0

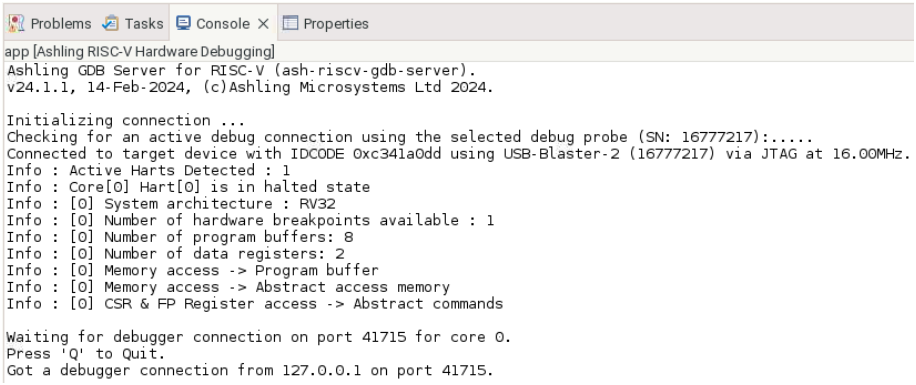
  Note: Nios V processor does not support Linux OS.

Figure 12. Enabling OS Aware Run in RISC-V Hardware Debugging



7. Click **Run**. RiscFree IDE for Altera FPGAs downloads the program to the target and you can find the console prints as shown in the following diagram.

Figure 13. Console Prints after Run Connection is Successful



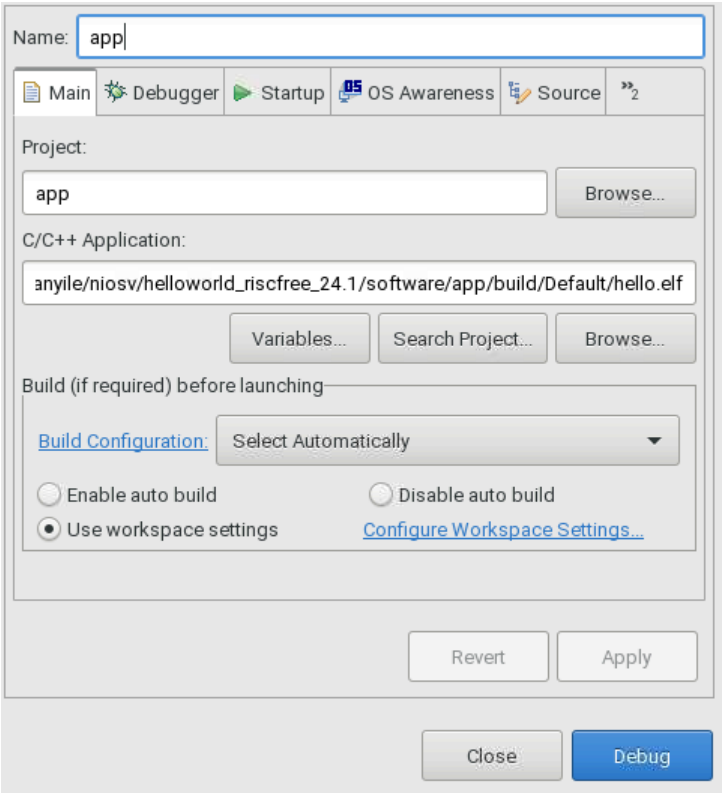
2.3.4. Debugging a Nios V Processor Project

2.3.4.1. Download and Debug Nios V Processor Application

You can download and debug the Nios V processor software project on the targeted Intel FPGA using the RiscFree IDE for Altera FPGAs. To debug the project, follow these steps:

1. Right-click the project folder (application or BSP) in the project explorer and select **Debug As ➤ Debug configurations**.
2. Select **Ashling RISC-V Hardware Debugging ➤ <Project Name>**. Ensure the **Project** and **C/C++ Application** match your project name and project `.elf` file, respectively.
3. Under the **Main** tab, for **C/C++ Application**, browse to select the application build `.elf` file. For example: `hello.elf`

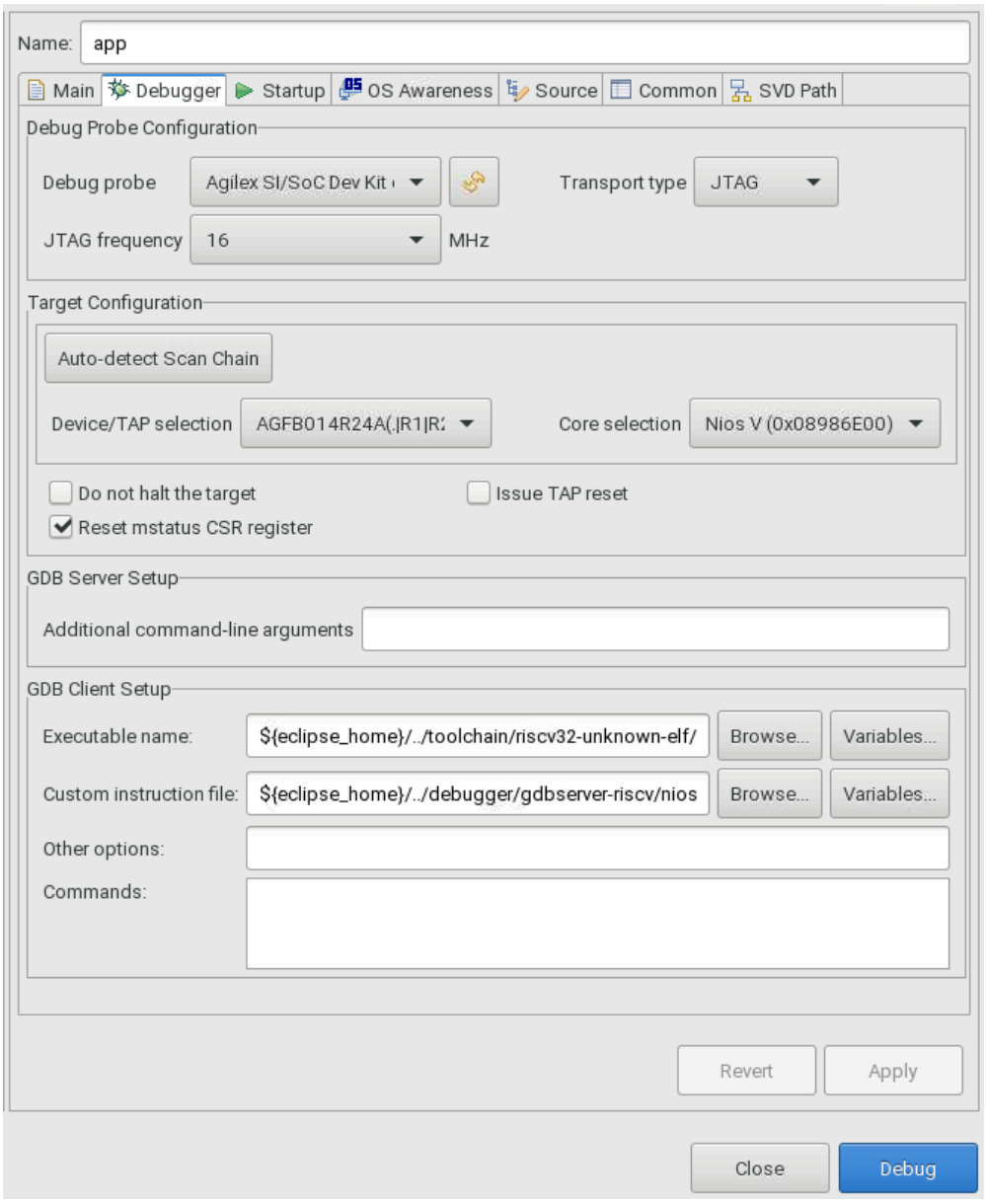
Figure 14. Debug Configurations for Nios V Processor—Main Tab



.

4. Under the **Debugger** tab, set these settings:
   - **Debug probe**: Agilex SI/SoC Dev Kit (Name of the FPGA board/development kit)
   - **Transport type**: JTAG
   - **JTAG frequency**: 16 MHz

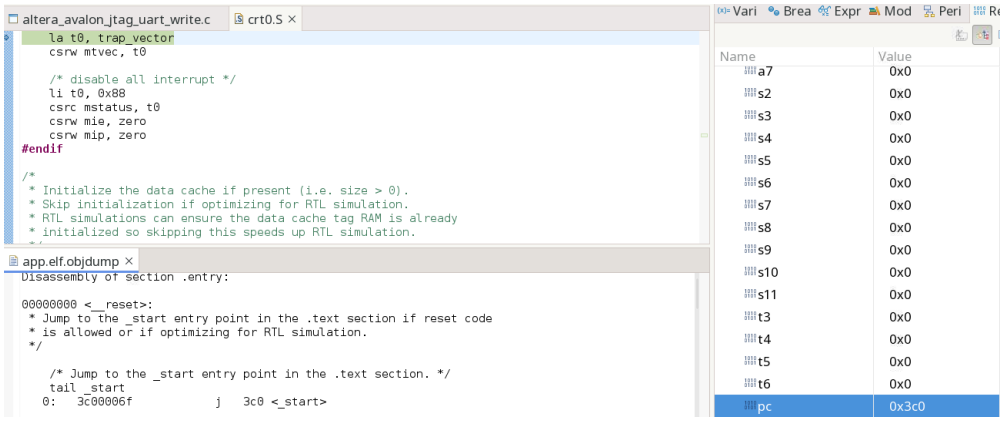
Figure 15. Debug Configurations for Nios V Processor—Debugger Tab



5. Click **Auto-detect Scan Chain** to automatically detect JTAG scan chain information of the target device. Select the options from **Device/Tap selection** and **Core selection**.
6. Leave all other settings as they are, including those in the **Startup** tab, as sensible defaults have been set.

   Note: If you turn off **Resume**, you can debug from the processor initialization.

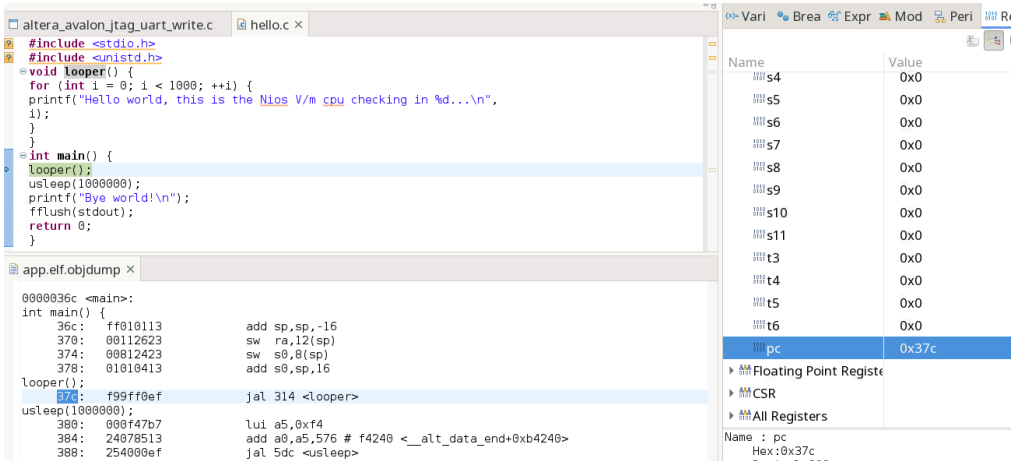
Figure 16. Debug Configurations for Nios V Processor — Turn off Resume



The example above shows that the program counter is at `0x3c0`, which is displayed in the disassembly just after the reset vector. The processor jumps to `0x3c0`, which is `_start` (the beginning of the Nios V processor initialization).

If you turn on **Resume**, you can debug from the user application.

Figure 17. Debug Configurations for Nios V Processor — Turn on Resume

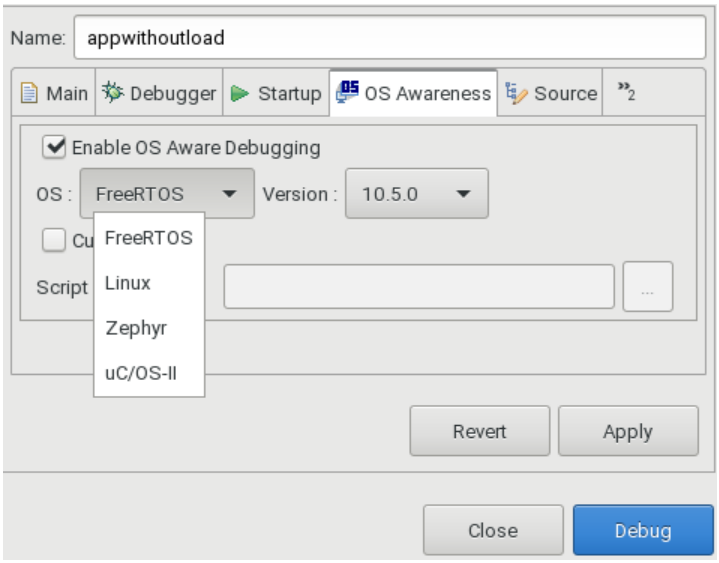


The example above shows that the program counter is at `0x37c`, which is paused at the main function and is shown in the disassembly. The processor then jumps to `0x37c`, which is paused at `looper()`.

7. Based on the OS you use, configure the OS Awareness settings as follows:
   - Intel HAL: No OS Awareness configuration is required.
   - Other OS: Under the **OS Awareness** tab, turn on **Enable OS Aware Debugging**, and select the OS version applicable to you as listed below:
     — **OS**: µC/OS-II and **Version**: 2.93.0
     — **OS**: FreeRTOS and **Version**: 10.5.0
     — **OS**: Zephyr and **Version**: 3.2.0

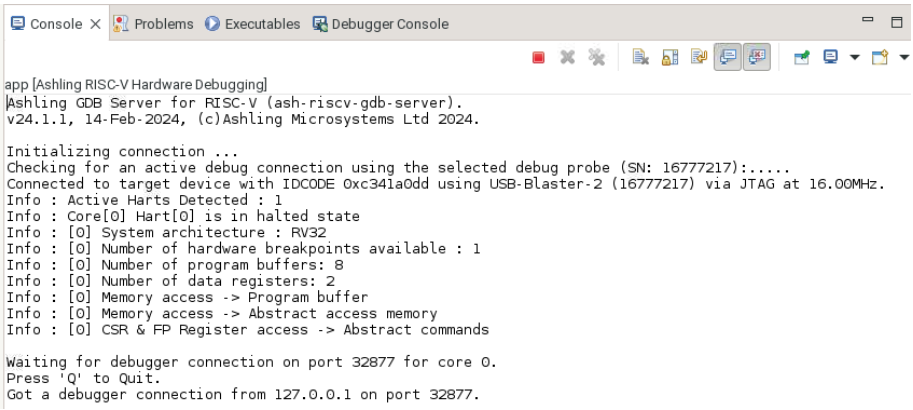
     Note: Nios V processor does not support Linux OS.

Figure 18. Enabling OS Aware Debugging in RISC-V Hardware Debugging



8. Click **Debug**. RiscFree IDE for Altera FPGAs downloads the program to the target and you can find the console prints as shown in the following diagram.

Figure 19. Console Prints after Debug Connection is Successful



9. Refer to the *Debugging Features with RiscFree IDE* section for further debugging.

Note: You can issue a debug reset using `niosv-download -r` command. This command only resets the Nios V processor if the debug reset interface is connected to the Nios V processor IP's reset input in your Platform Designer.

Note: `niosv-download` (under `<Intel Quartus Prime installation directory>/niosv/bin directory`) is only available for the Quartus Prime software. This tool is not available for standalone RiscFree IDE for Altera FPGAsRiscFree IDE installation with the Quartus Prime Programmer and Tools

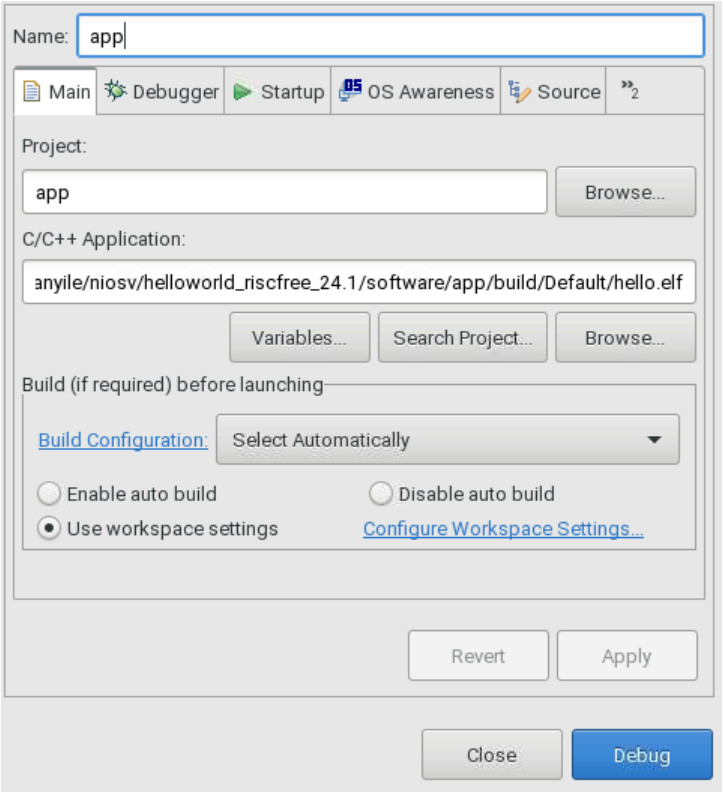
2.3.4.2. Connect to a Running Nios V Processor Application

You can connect to a running Nios V processor software project on the targeted Altera FPGA using Ashling RiscFree IDE for Altera FPGAs.

Follow these steps to connect to a running project:

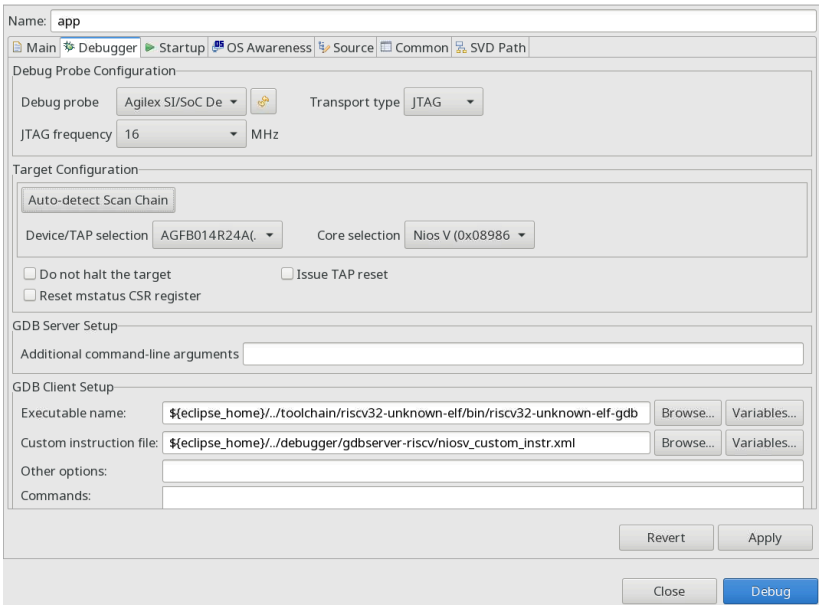
1. Right-click the project folder (application or BSP) in the project explorer and select **Debug As ➤ Debug configurations**.
2. Select **Ashling RISC-V Hardware Debugging ➤ <Project Name>**. Ensure the **Project** and **C/C++ Application** match with your project name and your project `.elf` file respectively.
3. Under the **Main** tab, for **C/C++ Application**, browse to select the application build `.elf` file. For example: `hello.elf`.

Figure 20. Debug Configurations for Nios V Processor—Main Tab



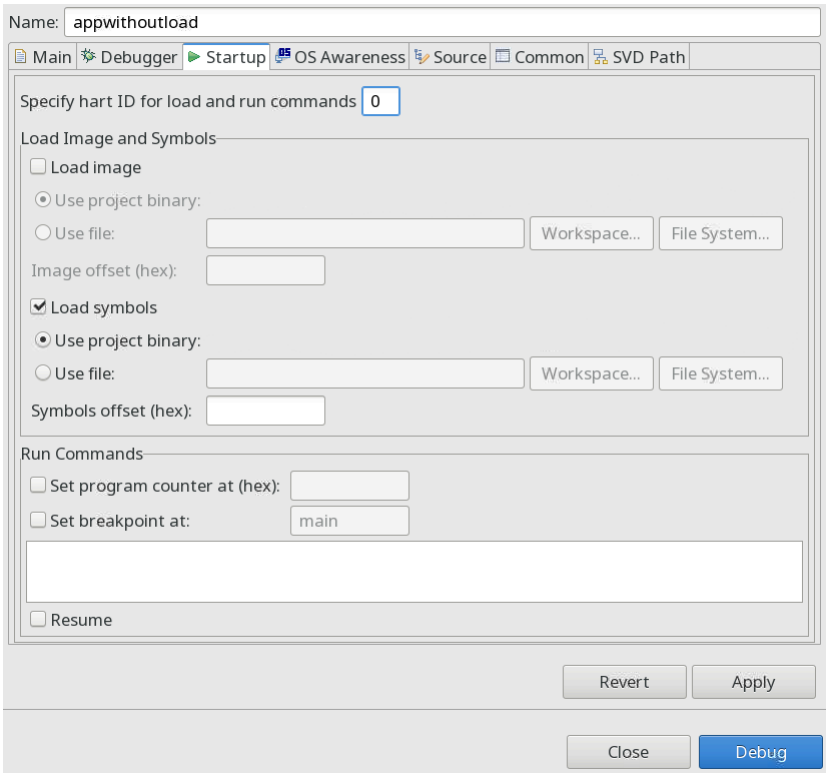
4. Under the **Debugger** tab, set these settings:
   - **Debug probe**: Agilex SI/SoC Dev Kit (Name of the FPGA board/development kit)
   - **Transport type**: JTAG
   - **JTAG frequency**: 16 MHz

Figure 21. Debug Configurations for Nios V Processor—Debugger Tab



5. Click **Auto-detect Scan Chain** to automatically detect JTAG scan chain information of the target device. Select the options from **Device/Tap selection** and **Core selection**.
6. Turn off **Reset mstatus CSR register**
7. Navigate to **Startup** tab, go to **Load Image and Symbols** and turn off the following features:,
   a. **Load Image**
   b. **Set Breakpoint**
   c. **Resume**

Figure 22. Debug Configurations for Nios V Processor—Startup Tab

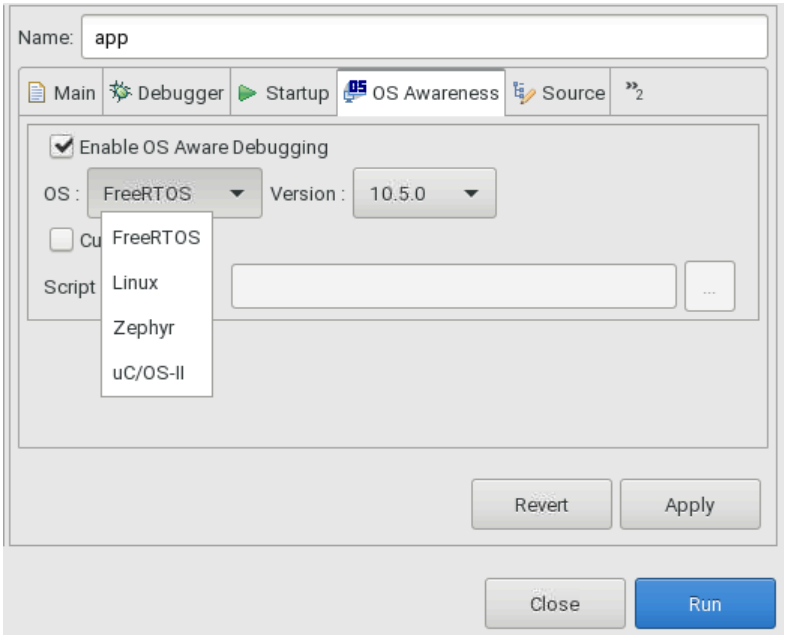


8. Based on the OS you use, configure the OS Awareness settings as follows:

- Intel HAL: No OS Awareness configuration is required.
- Other OS: Under the **OS Awareness** tab, turn on **Enable OS Aware Debugging**, and select the OS version applicable to you as listed below:
  - **OS**: µC/OS-II and **Version**: 2.93.0
  - **OS**: FreeRTOS and **Version**: 10.5.0
  - **OS**: Zephyr and **Version**: 3.2.0

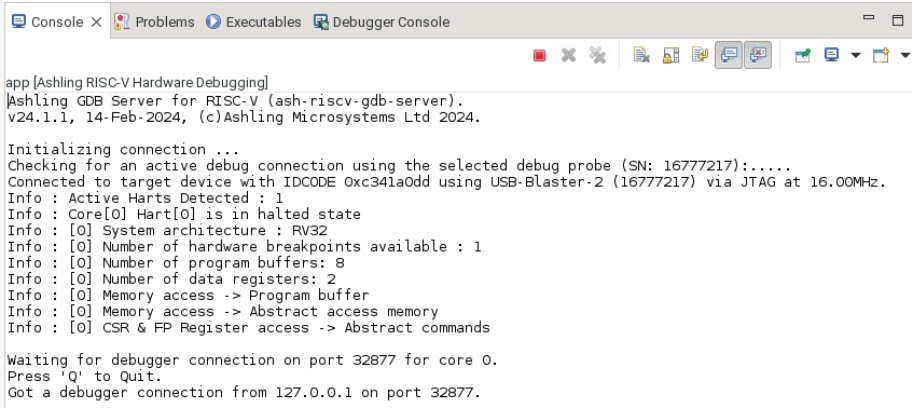
  Note: Nios V processor does not support Linux OS.

Figure 23. Enabling OS Aware Debugging in RISC-V Hardware Debugging



9. Click **Debug**. RiscFree IDE for Altera FPGAs downloads the program to the target and you can find the console prints as shown in the following diagram.

Figure 24. Console Prints after Debug Connection is Successful



Currently, the program is running in debug mode. You can check the Debug tab on the left side of the RiscFree IDE for Altera FPGAs.

Figure 25. Debug state - Running



Note: Before you proceed to debug, make sure the debugger is in **Suspended** state. For more information on how to suspend the debugger, refer to *GUI for Debugging*.

Figure 26. Debug state - Suspended

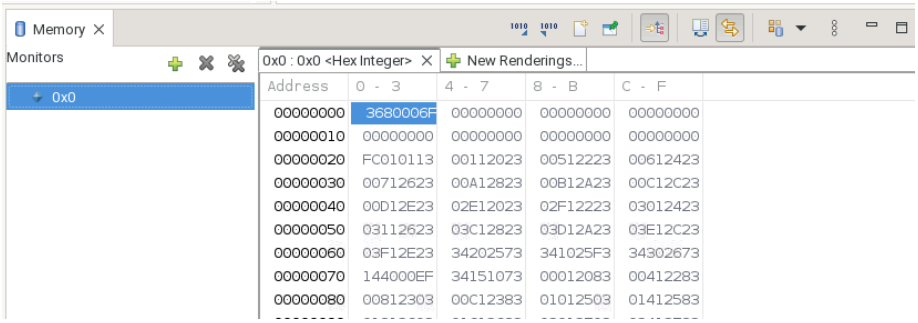


10. Click **Suspend** icon in the Debug bar.

    To verify the boot copier has copied the application successfully, continue with the following steps:

11. Go to **Window ➤ Show View ➤ Memory Browser**.
12. Select **Add Memory Monitor**.
13. Provide the memory address 0x0 and click **OK**.

Figure 27. Memory Browser at Address 0x0



14. Go to `<Working directory>/software/app/build/Default` folder.
15. Open the `hello.elf.objdump` file.
16. Search for **Disassembly** of section `.entry`.
17. The disassembly shows that the information at starting address 0 is 0x3680006f, which is exactly the same as in the **Memory Browser**.
18. You can continue to verify section `.exceptions`.

Figure 28. Disassembly of an Application

```
Disassembly of section .entry:

00000000 <__reset>:
 * Jump to the _start entry point in the .text section if reset code
 * is allowed or if optimizing for RTL simulation.
 */

    /* Jump to the _start entry point in the .text section. */
    tail _start
   0:   3680006f            j   368 <_start>
    ...
```

2.3.4.3. Considerations when Debugging ROM-based Designs

A ROM-based Nios V processor design refers to the Nios V processor software developed specifically to be stored in involatile memories, such as ROM or Flash. The ROM-based designs are closely related to the booting implementation in *Nios V Embedded Processor Design Handbook.*

- Nios V Processor Booting from On-Chip Flash (UFM)
- Nios V Processor Booting from General Purpose QSPI Flash
- Nios V Processor Booting from Configuration QSPI Flash
- Nios V Processor Booting from On-Chip Read-Only Memory (OCROM

Any write attempt by the debugger into a ROM or Flash is not supported because of the ROM's read-only characteristics and the Flash's write operation complexity, respectively.

Thus, debugging ROM-based designs come with the following limitations:

1. The debugger is unable to download the application ELF into ROM/Flash. This is affecting the **Load Image** feature in **Startup** tab.
2. The debugger is unable to set software breakpoint, which replaces instructions with ebreak. This condition is affecting the **Set breakpoint at** feature in **Startup** tab, stepping actions in **Debug Bar** and **Toggle Breakpoint**.

Once the application is booting successfully, Altera recommends implementing Connect to a Running Nios V Processor Application along with setups in Unable to Set Software Breakpoint to connect and debug the application.

2.3.4.3.1. Unable to download (Load Image)

You can program the ROM/Flash using specialized programming methods to load the image into the involatile memories. For more information, please refer to *Nios V Embedded Processor Design Handbook - Nios® V Processor Configuration and Booting Solutions*.

Table 6. Programming Methods

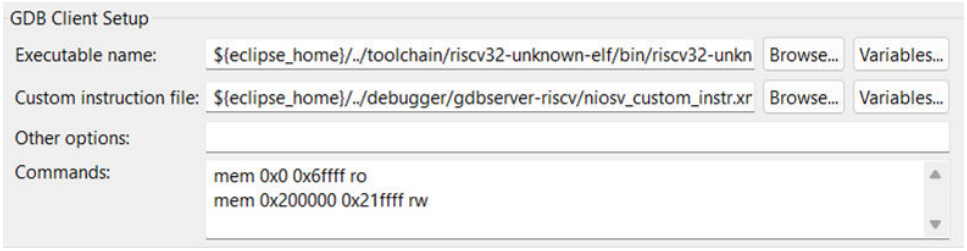
Involatile Memories	Programming File	Programming Methods
UFM	POF file	Using Quartus Programmer under Internal Configuration mode.
General Purpose QSPI Flash	POF file	Using Quartus Programmer under Passive Serial or AVST mode with Parallel Flash Loader.
Configuration QSPI Flash	JIC file	Using Quartus Programmer under Active Serial mode.
On-Chip Read-Only Memory	HEX file	Memory initialized during Quartus project compilation.

2.3.4.3.2. Unable to Set Software Breakpoint

The Ashling RiscFree IDE setups breakpoints as software breakpoints by default. This behavior applies to the **Set breakpoint at:** feature in the **Startup** tab, stepping actions in the **Debug Bar**, and **Toggle Breakpoints**. You must perform the following setups to switch from software breakpoint to hardware breakpoint.

Setups	Effects	Related Breakpoint Features
Define the ROM memory region in **Debugger tab ➤ GDB Client Setup ➤ Command**.	RiscFree IDE for Altera FPGAs automatically switches to hardware breakpoint when it identifies any breakpoint within a Read-Only memory region.	• **Set breakpoint at:** feature in **Startup** tab • Stepping actions in **Debug Bar**. • Any breakpoints within the Read-Only region
When adding breakpoints, click **Toggle Hardware Breakpoint**. Refer to Hardware Breakpoint.	Manually setup a hardware breakpoint.	Any user-added breakpoints within the Read-Only region
Follow these steps when adding breakpoints: 1. Click **Toggle Breakpoint**. 2. Right-click on the breakpoint symbol. 3. Navigate to **Breakpoint Properties**. 4. In **Common**, change **Type** from **Regular** to **Hardware**.	Manually configure a software breakpoint into hardware breakpoint.	Any user-added breakpoints within the Read-Only region

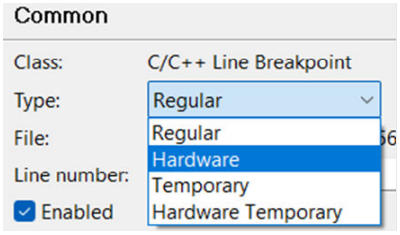
Figure 29. Defining ROM Memory Region



The following are the settings in this example:

- Address 0 to 0x6FFFF is a Read-Only region. Ashling RiscFree IDE for Altera FPGAs IDE automatically switched to hardware breakpoint when it identifies any breakpoint within this region.
- Address 0x200000 to 0x21FFFF is a Read-Write region. The breakpoint in this region remains as software breakpoints.

Figure 30. Type Option in Breakpoint Properties

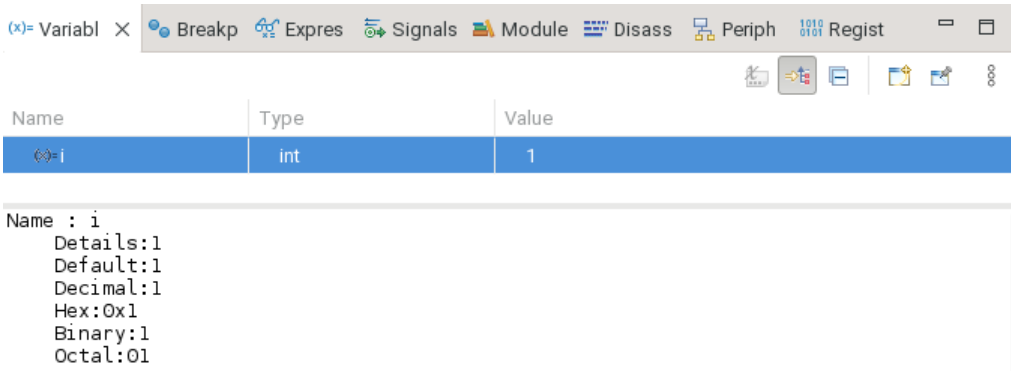


2.3.5. Debugging Tools

2.3.5.1. View Variables

To launch the variables view, go to **Window ➤ Show View ➤ Other... ➤ Variables**.

Figure 31. Variables View

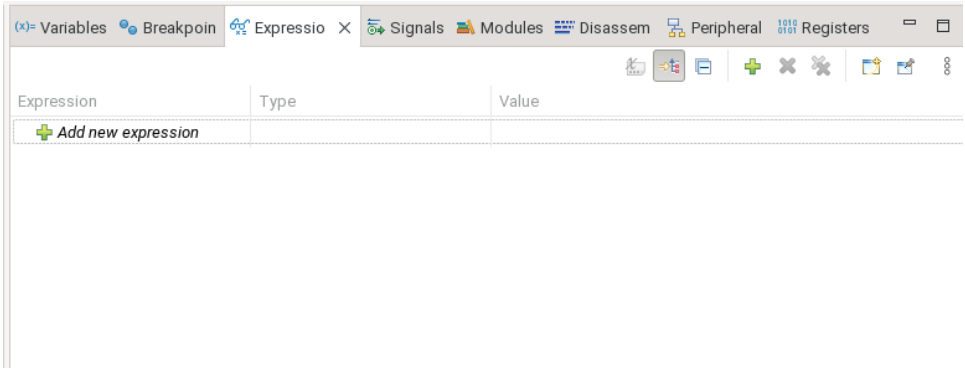


In this view, you can view the variables in your program. When you click on a variable, it displays the number format for the selected variable.

2.3.5.2. Adding Expressions

To launch the expressions view, go to **Window ➤ Show View ➤ Expressions**.

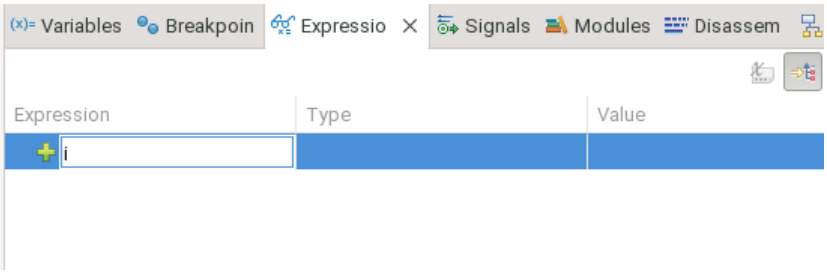
Figure 32. Expressions View



To add an expression in the view, follow these steps:

1. Click **Add new expression** in the view.
2. Type the expression that you want to display. Press **Enter**.

Figure 33. Expressions View – Add new expression



3. The expression and its value are displayed in the Expressions view. When the execution of a program is suspended, all expressions are reevaluated.

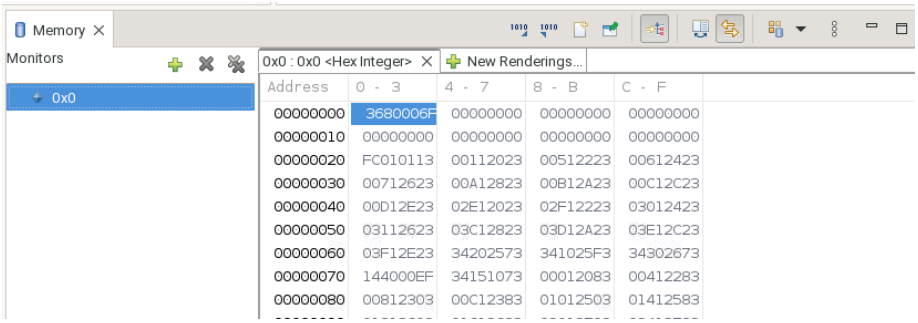
2.3.5.3. View Memory

Ashling RiscFree IDE for Altera FPGAs supports memory browsers. You can view the content of On-Chip Memory (RAM) or other memory devices. This example targets the start address of On-Chip Memory (RAM) where an application begins.

To launch the memory browser, follow these steps:

1. Go to **Window ➤ Show View ➤ Memory Browser**.
2. Select **Add Memory Monitor**.
3. Provide the memory address 0x0 and click **OK**.

Figure 34.



4. Go to `<Working directory>/software/app/build/Default` folder.
5. Open the `hello.elf.objdump` file.
6. Search for **Disassembly** of section `.entry`.
7. The disassembly shows that the information at starting address 0 is 0x36c006f, which is exactly the same as in the **Memory Browser**.
8. You can continue to verify section `.exceptions`.

Figure 35. Disassembly of an Application

```
Disassembly of section .entry:

00000000 <__reset>:
 * Jump to the _start entry point in the .text section if reset code
 * is allowed or if optimizing for RTL simulation.
 */

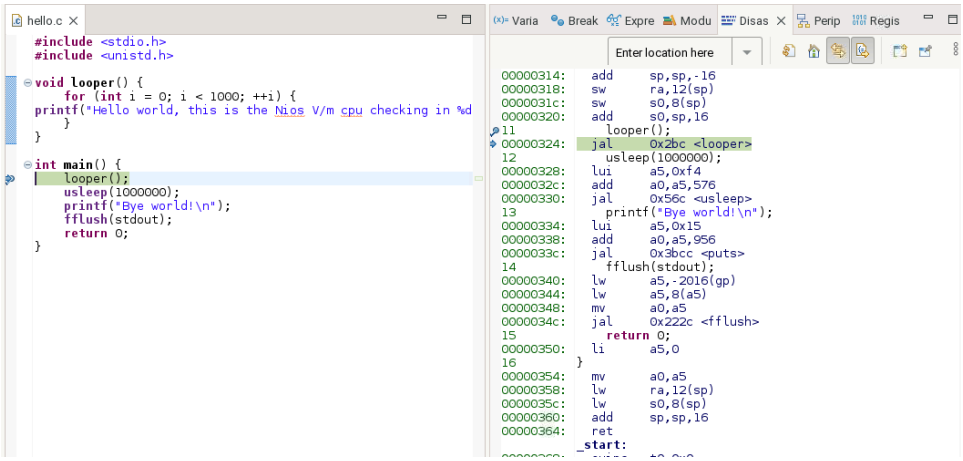
    /* Jump to the _start entry point in the .text section. */
    tail _start
   0:   3680006f            j    368 <_start>
    ...
```

2.3.5.4. Disassembly View with Registers View/ Stepping into assembler functions

Disassembly view allows you to observe your program as it is disassembled. This view is helpful when the instruction pointer enters a function for which it does not have the source.

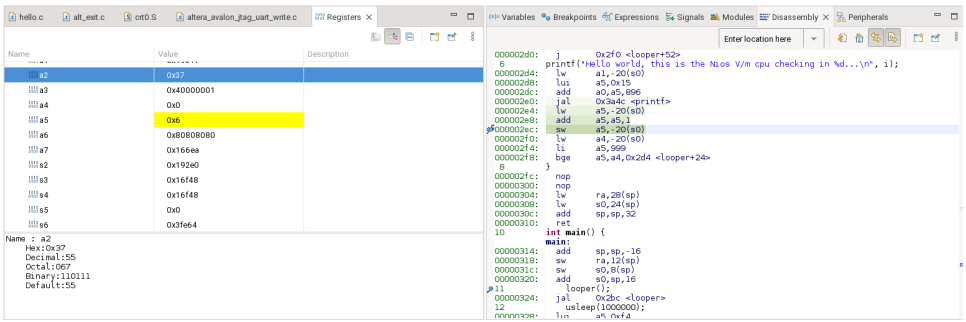
To launch the disassembly view, go to **Window ➤ Show View ➤ Disassembly**.

Figure 36. Disassembly View



To debug using the disassembled code in the **Disassembly** view, set a breakpoint at the line you wish to pause. Once the debugger step into the line that you have set the breakpoint, you can observe the change in the **Registers** view.

Figure 37. Disassembly View with Register View

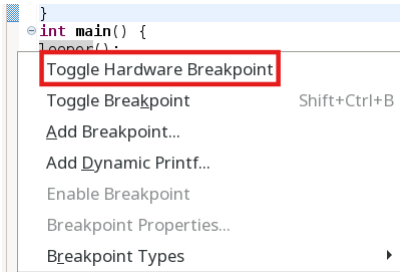


2.3.5.5. Hardware Breakpoint

RiscFree IDE for Altera FPGAs supports hardware breakpoint when debugging on On-Chip ROM or flash. Follow these steps to use hardware breakpoint:

1. Set a hardware breakpoint at `main()` by right click on the left-margin of the line containing `main()`.

Figure 38. Set Hardware Breakpoint



2. To make sure the hardware breakpoint is in the correct location, you can check the value in the `tdata2` register in the **Window ➤ Show View ➤ Other... ➤ Registers ➤ CSR**.

Figure 39. Value in `tdata2` Register

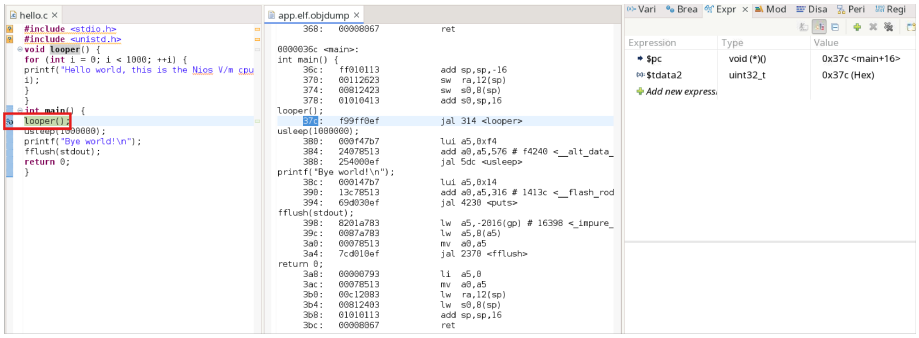
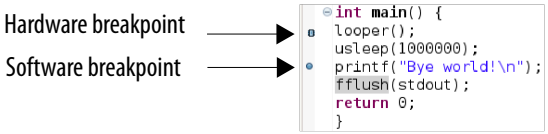


Figure 40. Difference between Hardware and Software Breakpoints



2.3.5.6. Configuring Program Counter Manually

The RiscFree IDE for Altera FPGAs provide the feature to change the processor's program counter during debugging. Note that, this feature must be used with caution.

Changing the program counter can lead to inconsistent program states, unexpected behaviour, or crashes if not done carefully. It is generally recommended for advanced users who have a deep understanding of the program's logic and state.

1. Navigate to **Startup tab ➤ Run Commands ➤ Set program counter at (hex)**.
2. Provide any instruction memory address in HEX.
3. You may refer to the objdump file to locate the interested instruction memory address.
4. Start debugging, and the processor jumps to the specified instruction memory address.

Figure 41. Set program counter at (hex) in Startup tab



2.4. Using Ashling RiscFree IDE for Altera FPGAs with Arm Hard Processor System

This section provides guidance on debugging Arm HPS core using the RiscFree IDE debugging tool. This tutorial is based on the Altera FPGA example design from Ashling.

Note: RiscFree IDE supports Arm HPS standalone debug.

2.4.1. Pre-requisite for Debugging Arm HPS Project

This example uses the Agilex 7 board. To prepare the platform you are planning to debug, follow these steps:

1. Prepare a golden system reference design (GSRD) HPS `.sof` file or use a prebuilt HPS `.sof` file.
2. Download the HPS `.sof` file to the Agilex 7 board using the Quartus Prime Programmer and Tools.

Related Information

- Agilex 7 SoC Golden System Reference Design (GSRD)
  Provides the prebuilt `.sof` files at the Release Contents section for download.
- Cyclone V SoC Golden System Reference Design (GSRD)
  Provides the prebuilt `.sof` files at the Release Contents section for download.
- Stratix 10 SoC Golden System Reference Design (GSRD)
  Provides the prebuilt `.sof` files at the Release Contents section for download.
- Arria 10 SoC Golden System Reference Design (GSRD)
  Provides the prebuilt `.sof` files at the Release Contents section for download.

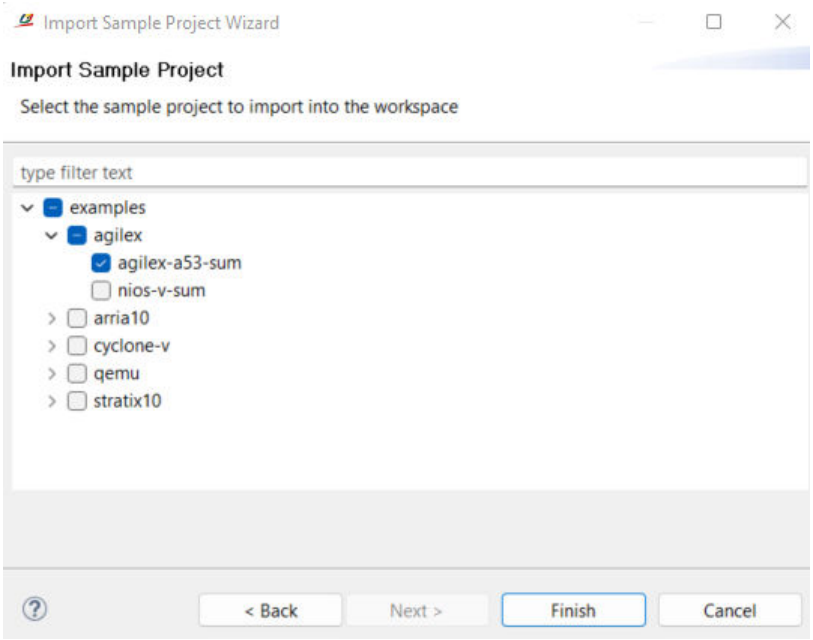
2.4.2. Importing Arm HPS Project

To import the Agilex 7 Arm Cortex*-A53 sample project and a pre-built `.elf` file, follow these steps:

Note: Arm toolchain is not included by default. Follow the steps in the *Toolchain for Arm Processor* section to modify toolchain configuration to include Arm GNU GCC toolchain.

1. To open the **Import** wizard, click **File ➤ Import**. In the **Import** wizard, select **Ashling Sample Projects ➤ C/C++ Sample Projects** and click **Next**.
2. In the **Import Sample Project Wizard**, select **examples ➤ agilex ➤ cortex-a53-sum** and click **Finish**. The project is listed in the Project Explorer.

Figure 42. Importing Cortex-A53 Sample Project to RiscFree IDE



Related Information

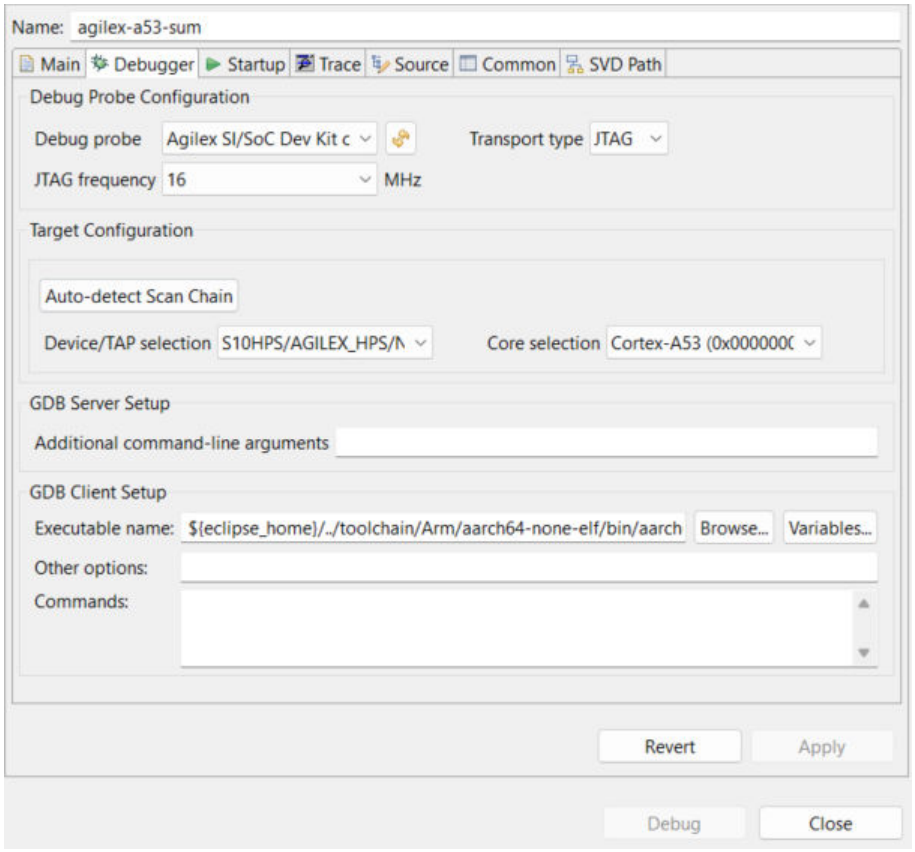
Toolchain for Arm Processor on page 119

2.4.3. Setting Debug Configurations and Downloading Arm HPS Project Using RiscFree IDE

To debug the project, follow these steps:

1. Right-click the project directory and select **Debug ➤ Debug Configurations**.
2. Select **Ashling Arm Hardware Debugging ➤ cortex-a53-sum**. Ensure the **Project** and **C/C++ Application** match with your project name and your project `.elf` file respectively.
3. Under the Debugger tab, set these settings:
   - Debug probe: **Agilex SI/SoC Dev Kit** — Select the connected device
   - Transport type: **JTAG**
   - JTAG/SWD frequency: **16 MHz**
   - Core selection: Can select any core between **0** to **3**

Figure 43. Debugger Settings for Agilex 7 Arm Cortex-A53 Core



Note: The default settings for the current example design in the RiscFree IDE are configured based on Agilex 7 device.

Note: If you are using a different Arm architecture, select the Arm GNU debugger (GDB) (select **GDB Client Setup ➤ Executable name**) manually.

- 64-bit ELFs: `${eclipse_home}/../toolchain/Arm/aarch64-none-elf/bin/aarch64-none-elf-gdb.exe`
- 32-bit ELFs: `${eclipse_home}/../toolchain/Arm/arm-none-eabi/bin/arm-none-eabi-gdb.exe`

4. Click **Debug**. RiscFree IDE downloads the program to the target and you can find the console prints as shown in the following diagram.

Figure 44. RiscFree IDE after Program is Downloaded and System is Ready for Debug



Related Information

Debugging Features with RiscFree IDE for Altera FPGAs on page 41

2.5. Debugging Features with RiscFree IDE for Altera FPGAs

2.5.1. Debug Features in RiscFree IDE

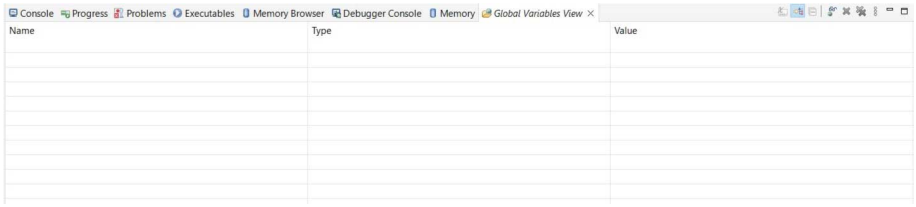
2.5.1.1. Viewing Global Variables

RiscFree IDE has dedicated global variables.

To view the global variables in the application (`.elf`), follow these steps:

1. Go to **Window ➤ Show View ➤ Other... ➤ Debug ➤ Global Variables View**.

Figure 45. Global Variables View



2. To list all available global variables in the application (`.elf`), in the **Global Variables View** tab, click the **Add Global Variables** icon. Select the required variables to view in the **Global Variables View**.

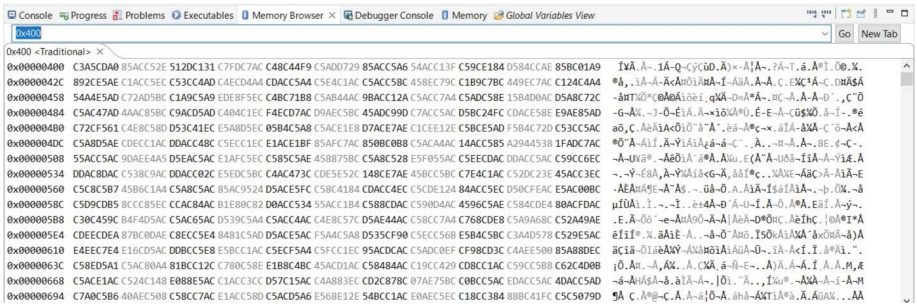
2.5.1.2. Launching Memory Browser

RiscFree IDE supports memory browser.

To launch the memory browser, follow these steps:

1. Go to **Window ➤ Show View ➤ Memory Browser**.
2. View the target memory in the memory browser.

Figure 46. Memory Browser showing Target Memory



2.5.1.3. Debugger Console

RiscFree IDE supports debugger console.

You can launch the debugger console from **Window ➤ Show View ➤ Debugger Console**.

You can enter debug commands and view output in the debug console. The GNU GDB syntax is fully supported in the console.

2.5.1.4. Specifying the SVD File

RiscFree IDE supports device-specific System View Description (SVD) JSON format files which provide a high-level, textual description of registers and the associated bits. Intel provides the SVD files as part of the Software Development Kit.

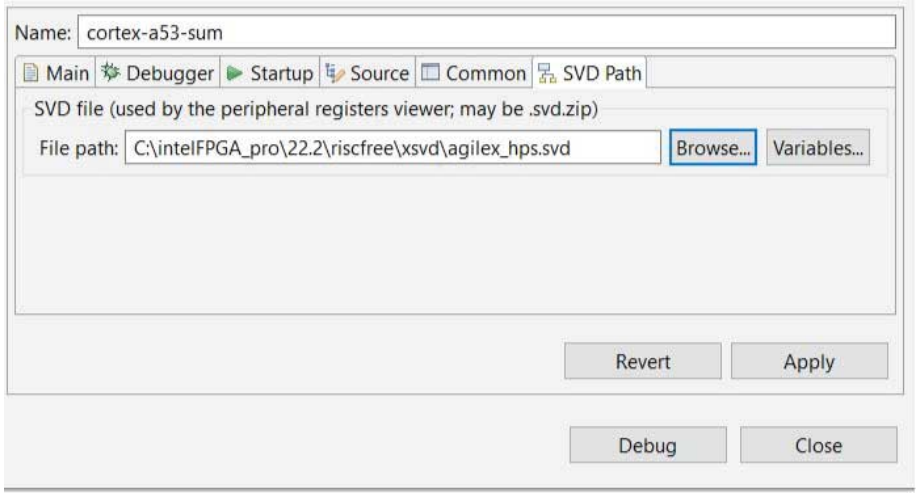
Table 7. SVD File Path

Operating System	SVD File Path
Linux	*<Intel Quartus Prime installation directory>*/riscfree/xsvd
Windows	*<Intel Quartus Prime installation directory>*/riscfree/xsvd

To specify the SVD file in Debug Configuration, follow these steps:

1. Open the **Debug Configuration** window.

Figure 47. Specifying the SVD File



2. Under the **SVD Path** tab, browse and select the SVD file.
3. Click **Debug** to start debugging.

2.5.1.5. Breakpoint Settings

RiscFree IDE supports SoC-wide breakpoint and core-specific software breakpoints in shared code.

SoC-wide Breakpoint

A single breakpoint can halt all cores in active debug launches. To enable SoC-wide breakpoint, go to **Run ➤ Enable SoC Wide Breakpoint**.

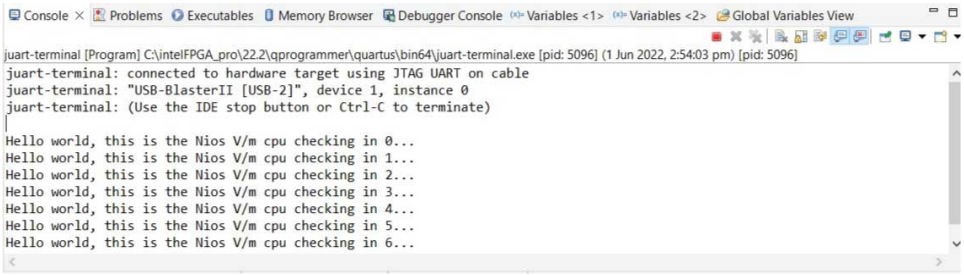
Core-specific Software Breakpoints

This breakpoint can only halt a specified core if debug is running for common code shared between multiple cores. To enable this feature, go to **Window ➤ Preferences ➤ C/C++ ➤ Debug ➤ GDB** and select **Set breakpoint for active debug context only**. This option is unchecked by default. If this is unchecked, any breakpoint set in shared code is applicable for all the cores sharing the code.

2.5.1.6. JTAG UART Output Terminal

You can view the output of JTAG UART (juart) terminal in the RiscFree IDE via the external tool configuration.

Figure 48. JTAG UART Terminal Console Print



Related Information

Performing External Tools Configuration in Ashling RiscFree IDE for Altera FPGAs on page 121

2.5.2. Processor System Debug

This section is a generic debugging section where the debugging steps are the same for Nios V processor and Arm HPS debugging.

Follow the guidelines in this section to start debugging the program using GUI or the GDB console after completing the steps from the following sections:

- *Setting Debug Configurations and Downloading Nios V Processor Project Using RiscFree IDE*
- *Setting Debug Configurations and Downloading Arm HPS Project Using RiscFree IDE*

Related Information

Setting Debug Configurations and Downloading Arm HPS Project Using RiscFree IDE on page 39

2.5.2.1. GUI for Debugging

You can control the debugging process using the Debug Bar.

Figure 49. Execution Control in the Debug Bar

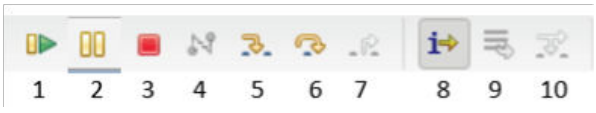


Table 8. Execution Control Using the Debug Bar

Debug Bar GUI	Function	Description
1	Resume	Continues the debugging process until the next breakpoint, or until the end of the debugging process.
2	Suspend	Halts the running code while debugging.
3	Terminate	Terminates the debug session. You need to relaunch the debug session after terminated.
4	Disconnect	Disconnects the debug tool connected to the RiscFree IDE.
5	Step into	Steps into the next method call at the currently executing line of code.
6	Step over	Steps over the next method call (without entering it) at the currently executing line of code. The method is still executed.
7	Step return (out)	Returns from a method which has been stepped into. The remainder of the code that was skipped by returning is still executed.
8	Instruction stepping mode	Switches to instruction stepping mode.
9	Drop to frame	Pops the current stack frame and puts control back out to the calling method, resetting any local variables.
10	Use step filters	Changes whether step filters should be used in the current Debug View.

2.5.3. Heterogeneous Multicore Debug

Heterogeneous multicore debugging is a method where you can debug two or more cores that differ in architecture or micro-architecture (Nios V processor core and Arm HPS core) simultaneously in the same integrated development environment.

By using heterogeneous multicore debugging, you can do the following:

- Simultaneously debug the entire integrated system in a single IDE.
- Real-time control of multiple processor cores that interact with each other.

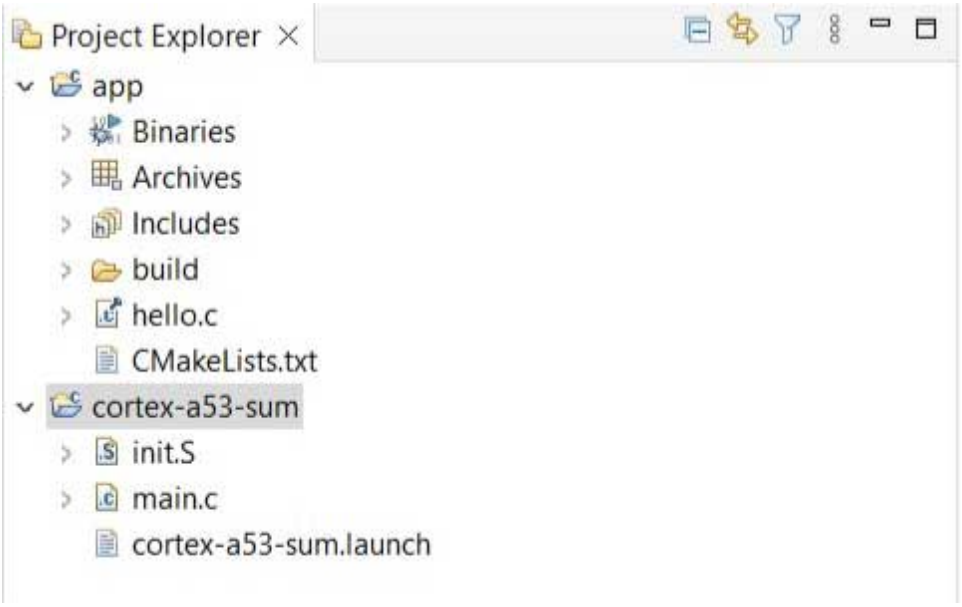
The example uses Nios V processor "Hello World" project and HPS design example provided by Ashling.

2.5.3.1. Importing Nios V Processor and Arm HPS Projects

To import the Nios V processor and Arm HPS projects to the RiscFree IDE workspace, follow these steps:

1. Follow the steps in the *Importing Nios V Project* and *Importing Arm HPS Project* sections.
2. After you import the projects, you can find the project directory in the Project Explorer.

Figure 50. Nios V Processor and Arm Cortex-A53 Ashling Samples Imported into RiscFree IDE Workspace



Related Information

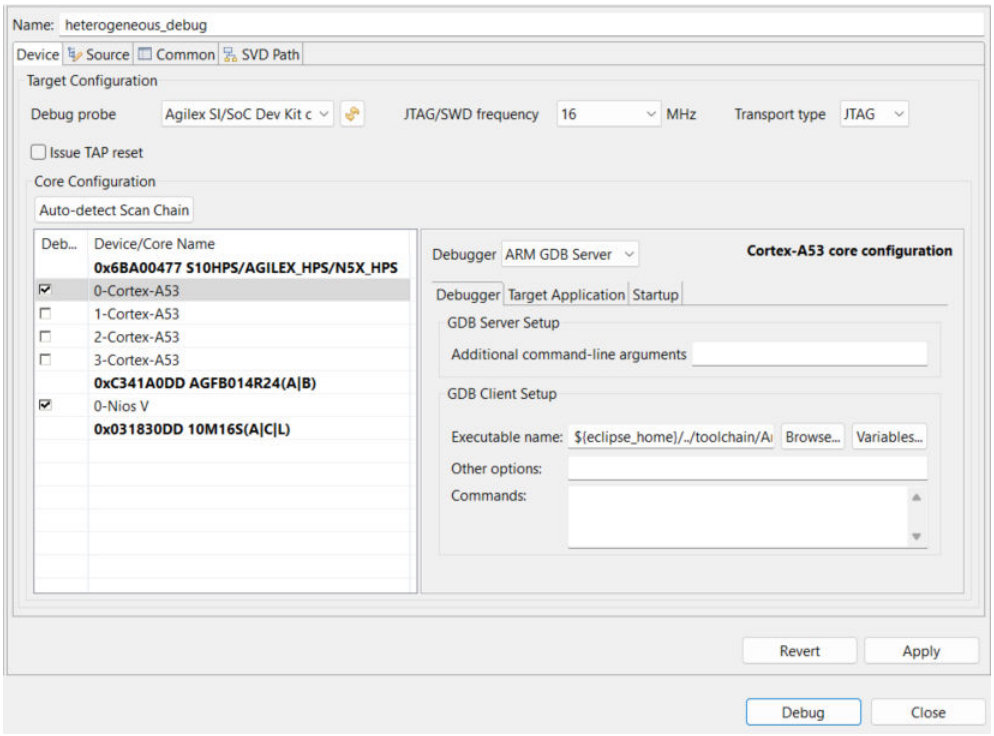
- Importing Nios V Processor Project on page 18
- Importing Arm HPS Project on page 38

2.5.3.2. Setting Core Configuration

To set the core configuration, follow these steps:

1. Right click project directory (either HPS or Nios V application) and select **Debug ➤ Debug Configurations**.
2. Select **Ashling Heterogenous Multicore Debugging ➤ cortex-a53-sum.elf**.
3. Under the **Device** tab, set these settings:
   - Debug probe: **Agilex SI/SoC Dev Kit**
   - JTAG/SWD frequency: **16 MHz**
   - Transport type: **JTAG**
4. Click the **Auto-detect Scan Chain** to list out all the available cores.
5. Under **Core Configuration**, select **Cortex-A53** and **Nios V/m**.

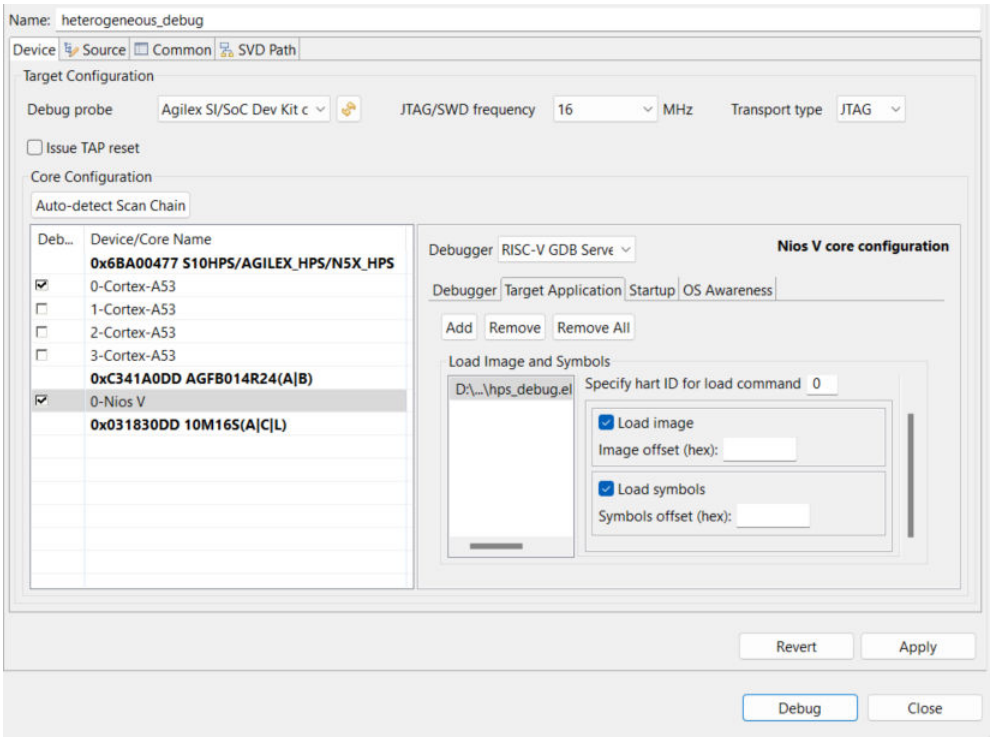
Figure 51. List of Available Cores for the Selected Target Device in RiscFree IDE



6. Set the core specific configuration for each core under the **Debugger** tab.
7. Specify the `.elf` file for each core under the **Target Application** tab. In this example, the target applications are added as `nios-v-sum.elf` for **Nios V/m** core and `cortex-a53-sum.elf` for **Cortex-A53** core.

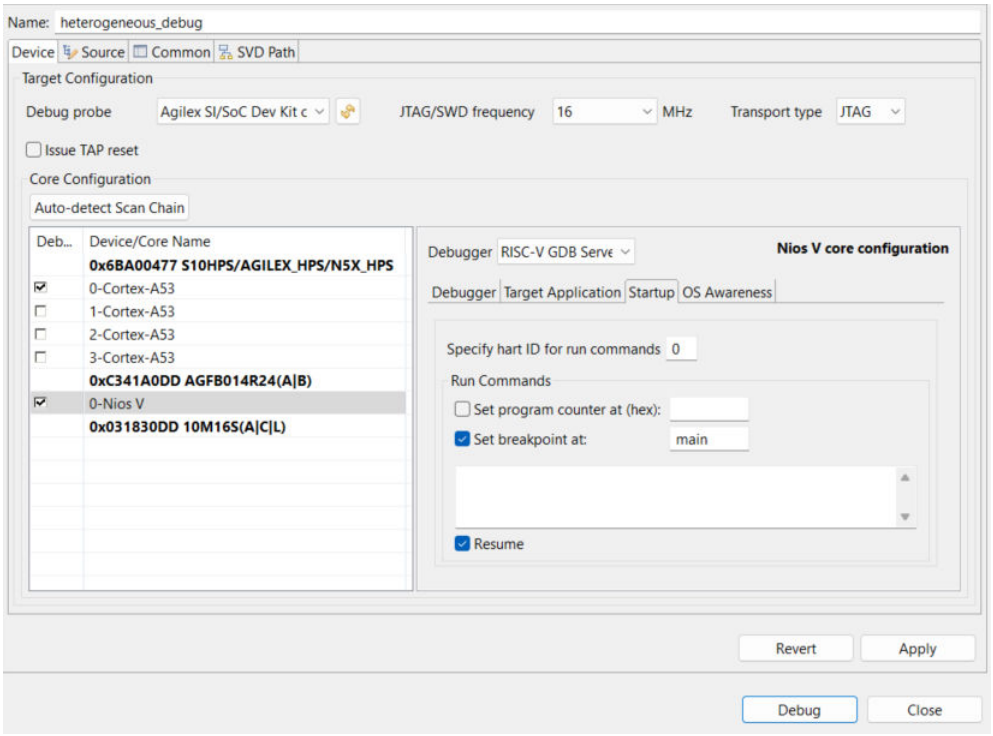
   Note: You can specify multiple `.elf` file for a single core.

Figure 52. Target Application Tab Settings



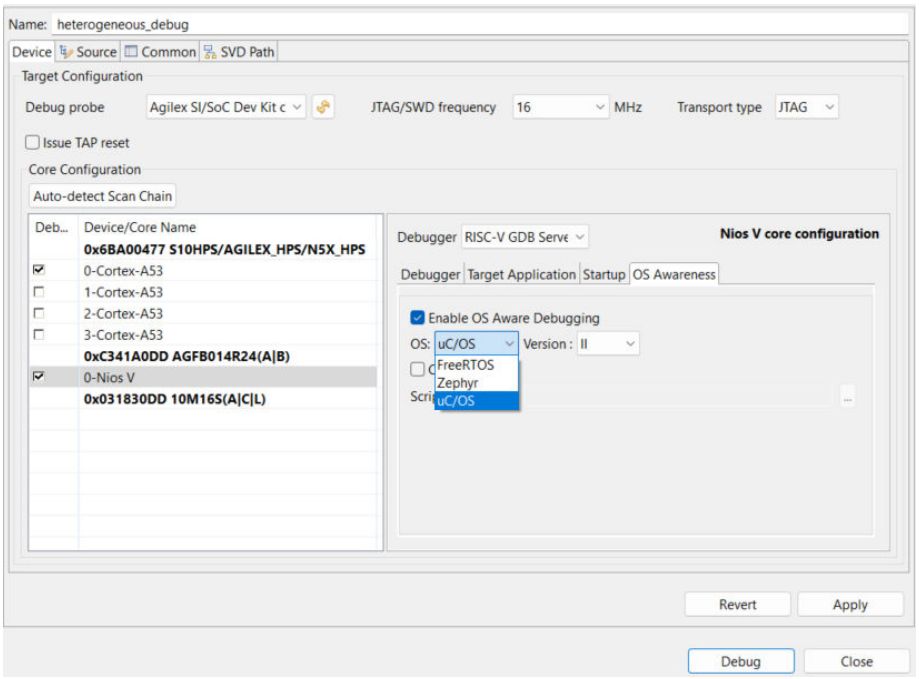
8. Set the breakpoint at a specific function under the **Startup** tab. This debug example uses the default settings.

Figure 53. Startup Tab Settings



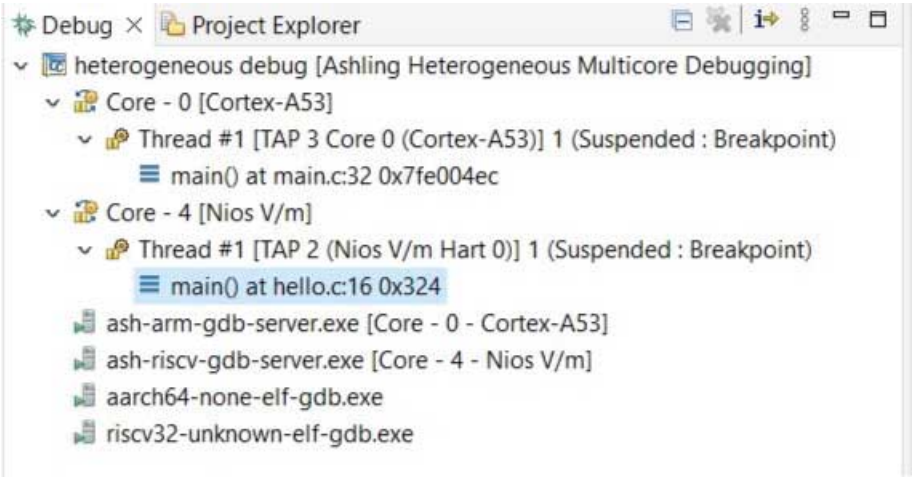
9. Based on the OS you use, configure the OS Awareness settings as follows:
   - Intel HAL: No OS Awareness configuration is required.
   - Other OS: Under the **OS Awareness** tab, turn on **Enable OS Aware Debugging**, and select the OS as listed below:
     — **µC/OS** and Version: **II**
     — **FreeRTOS** and Version: **10.4.1**
     — **Zephyr** and Version: **3.1.0**

Figure 54. Enabling OS Aware Debugging in Heterogeneous Debugging



10. Click **Debug** when all configuration is complete. All cores selected for debug are launched.

Figure 55. Heterogeneous Multicore Debug View in RiscFree IDE



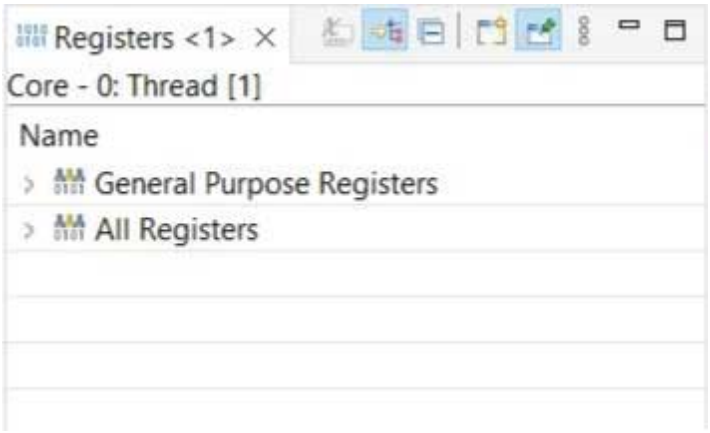
11. To debug two cores simultaneously, go to **Window ➤ Show View ➤ Registers**. Then, make a copy of the existing debug view using the **Open new View** icon.

Figure 56. Register Window



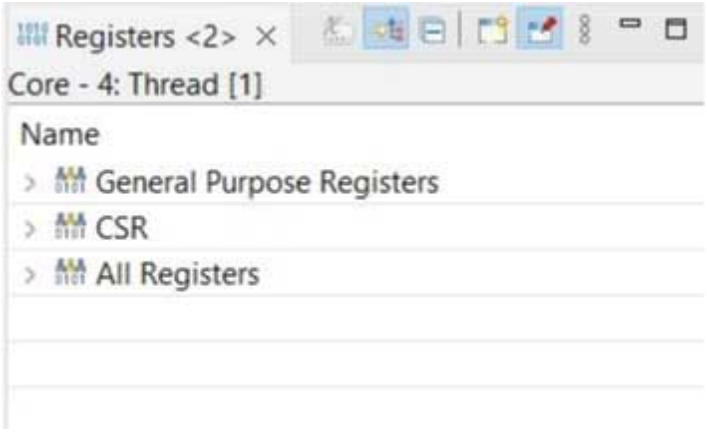
12. In the **Heterogenous Multicore Debug View** pane, select **Thread #1 [TAP 3 Core 0 (Cortex-A53)]**. Pin one copy of the debug view to Arm processor using the **Pin to Debug Context** icon.

Figure 57. Arm Processor (Core 0) Register View



13. In the **Heterogenous Multicore Debug View** pane, select **Thread #1 [TAP 2 (Nios V/m Hart 0)]**. Pin second copy of the debug view to Nios V processor using the **Pin to Debug Context** icon.

Figure 58. Nios V Processor (Core 4) Register View



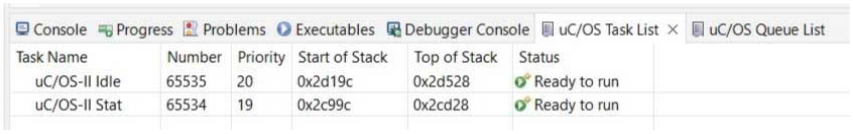
2.5.4. Debugging µC/OS-II Application

When you debug the µC/OS-II application, you can view the number of tasks, task details, and queue list.

To debug µC/OS-II application, follow these steps:

1. Launch the µC/OS-II application.
   - To open the µC/OS Task List, go to **Window ➤ Show View ➤ Other ➤ µC/OS ➤ µC/OS Task List**

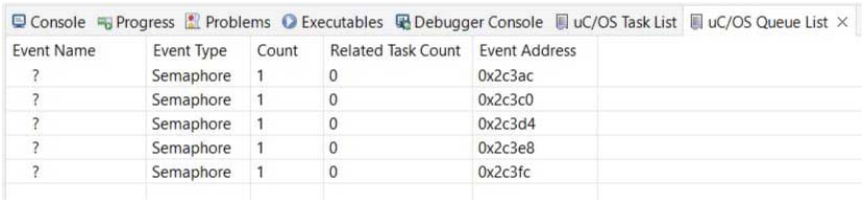
Figure 59. µC/OS Task List View



The µC/OS Task List shows task details including task name, priority, status, and stack details.

- To open the µC/OS Queue List, go to **Window ➤ Show View ➤ Other ➤ µC/OS ➤ µC/OS Queue List**.

Figure 60. µC/OS Queue List View



The µC/OS Queue List shows details of all events and queues, including tasks related to each event.

2.5.5. Debugging FreeRTOS Application

When you debug the FreeRTOS application, you can view the number of tasks, task details, and queue list.

To display details of FreeRTOS Task List, enable the macro as below in the `FreeRTOSConfig.h`.

Note: If you use the FreeRTOS BSP generated from BSP Editor, you do not need to manually configure `configUSE_TRACE_FACILITY` in `FreeRTOSConfig.h`. The setting depends on `system.h` (part of `FreeRTOSConfig.h #include`), which can be configured using BSP Editor GUI or `niosv-bsp` command.

Table 9. FreeRTOS Debug Configuration

Debug Information	Macros	Configuration	Description
Task List	`configUSE_TRACE_FACILITY`	`1`	To display task number of the project

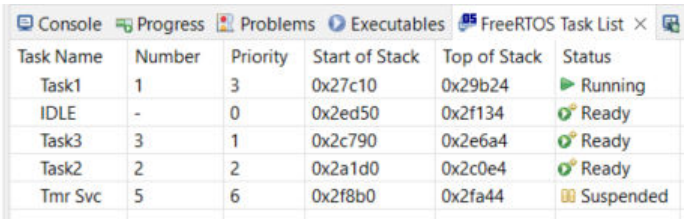
The configuration is as shown below:

```
#define configUSE_TRACE_FACILITY 1
```

To debug FreeRTOS application, follow these steps:

1. Launch the FreeRTOS application debug.
   - To open the FreeRTOS Task List, go to **Window ➤ Show View ➤ Other ➤ FreeRTOS ➤ FreeRTOS Task List**.

Figure 61. FreeRTOS Task List View



The **FreeRTOS Task List** shows task details including task name, priority, status, and stack details.

   - To open the FreeRTOS Task List, go to **Window ➤ Show View ➤ Other ➤ FreeRTOS ➤ FreeRTOS Task List**

Figure 62. FreeRTOS Queue List View



The **FreeRTOS Queue List View** shows details of all events and queues, including tasks related to each event.

Related Information

FreeRTOS* Real-Time Operating System section, Nios V Processor Software Developer Handbook

For more information about FreeRTOS*.

2.5.6. Debugging Zephyr Application

When you debug the Nios V processor application, you can view the Zephyr Mutex (Mutual Exclusion Object) List, Zephyr Semaphore List, and Zephyr Thread List.

To display details of Zephyr Mutex, Semaphore and Thread lists, enable the macros as below in the `prj.conf` file.

Table 10. Zephyr Debug Configuration

Debug Information	Macros	Configuration	Description
Enable Trace	`CONFIG_TRACING`	y	To enable all the tracing function.

continued...

Debug Information	Macros	Configuration	Description
			Note: All tracing macros have dependency to CONFIG_TRACING.
Mutex List	CONFIG_TRACING_OBJECT_TRACKING	y	To enable object tracking for mutex.
Semaphore List			To enable object tracking for semaphore.
Thread List	CONFIG_DEBUG_THREAD_INFO	y	To display thread details.

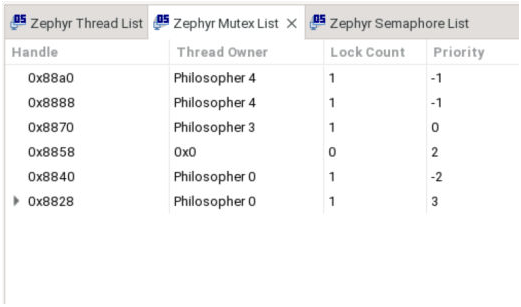
The configuration is as shown below:

```
CONFIG_TRACING=y
CONFIG_TRACING_OBJECT_TRACKING=y
CONFIG_DEBUG_THREAD_INFO=y
```

To debug Zephyr application, follow these steps:

1. Launch the Zephyr application debug.
   - To open the Zephyr Mutex List view, go to **Window ➤ Show View ➤ Other ➤ Zephyr Mutex List**.

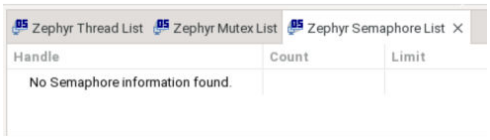
Figure 63. Zephyr OS Mutex List View



The **Zephyr Mutex List** tab shows details of all mutexes including handle, thread owner, lock count, priority, along with any wait threads related to each mutex.

- To open the Zephyr Semaphore List view, go to **Window ➤ Show View ➤ Other ➤ Zephyr Sempahore List**.

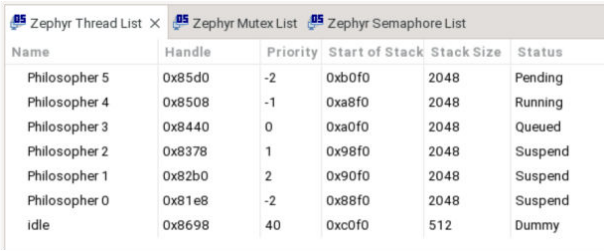
Figure 64. Zephyr Sempahore List View



The **Zephyr Semaphore List** tab shows details of all semaphores including handle, count, limit along with any wait threads related to each semaphore.

- To open the Zephyr Thread List view, go to **Window ➤ Show View ➤ Other ➤ Zephyr Thread List**.

Figure 65. Zephyr OS Thread List View



The **Zephyr Thread List** tab shows thread details including name, handle, priority, start of stack, stack size, and status details.

Related Information

Nios V Processor Zephyr Design Example RocketBoards

2.5.7. Arm HPS On-Chip Trace

You can configure and capture the Arm HPS On-Chip Trace with Intel FPGA Download Cable II. The following table shows the Arm trace sources that generate instruction trace:

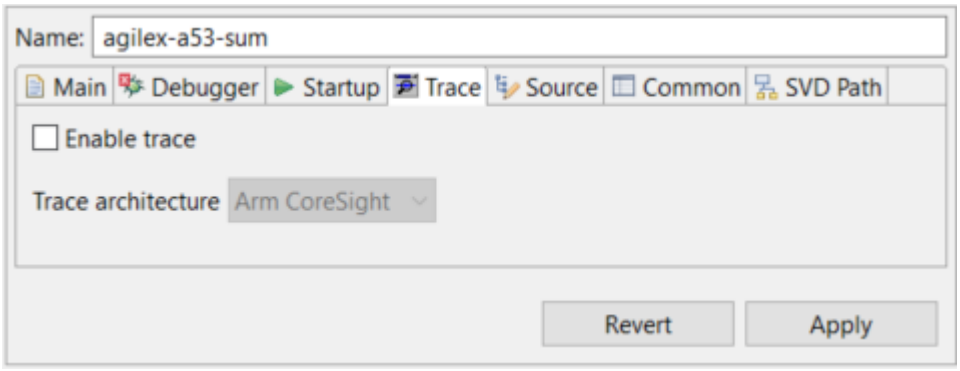
Table 11. Arm Trace Source

Arm Trace Source	Encoding Type	Description	Supported Device
Program Trace Macrocell (PTM)	PTM	A real-time trace module providing instruction tracing of a processor	• Arria 10 • Intel Cyclone V
Embedded Trace Macrocell (ETM)	ETMv4	The Arm trace source that captures data and instruction trace	• Agilex 7 • Stratix 10

Follow the steps to enable and configure the **Trace** feature:

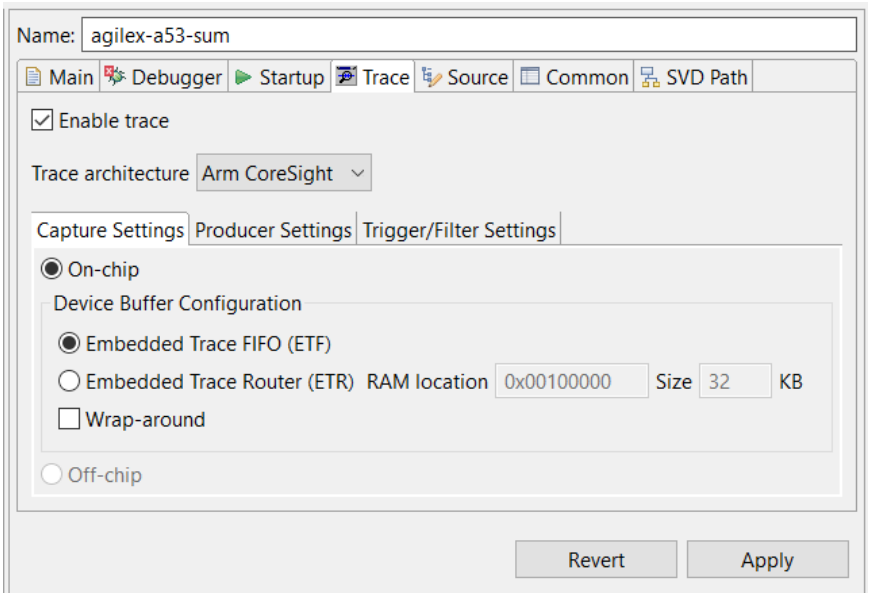
1. Launch **Ashling Arm Hardware Debugging**, and click the **Trace** tab.

Figure 66. Trace Tab



2. Click the **Enable trace** checkbox to enable the trace selected settings. The **Trace architecture** defaults to **Arm Coresight**. The trace selected settings appear in the following tab:
   a. **Capture Settings**
   b. **Producer Settings**
   c. **Trigger/Filter Settings**

Figure 67. Capture Settings Sub-Tab



3. Under the **Capture Settings** tab, set this settings:
   a. Select **On-chip**.
   b. Select one of the following Device Buffer Configuration:

i. **Embedded Trace FIFO (ETF)** — stores trace data in small and fixed size trace buffer.

ii. **Embedded Trace Router (ETR)** — routes the trace data to the system memory. You need to specify the RAM location and Size .

c. **Wrap-around** — enables overwriting old trace data with the new trace data when the trace buffer is full.

Note: **Wrap-around** is optional. If you do not select **Wrap-around**, trace capture stops when the trace buffer is full.

4. Under **Producer Settings** tab, set this settings:

a. Select your preferred settings:

i. **Cycle Accurate Trace** — captures trace on all cycles even if there is no trace to output on that cycle.

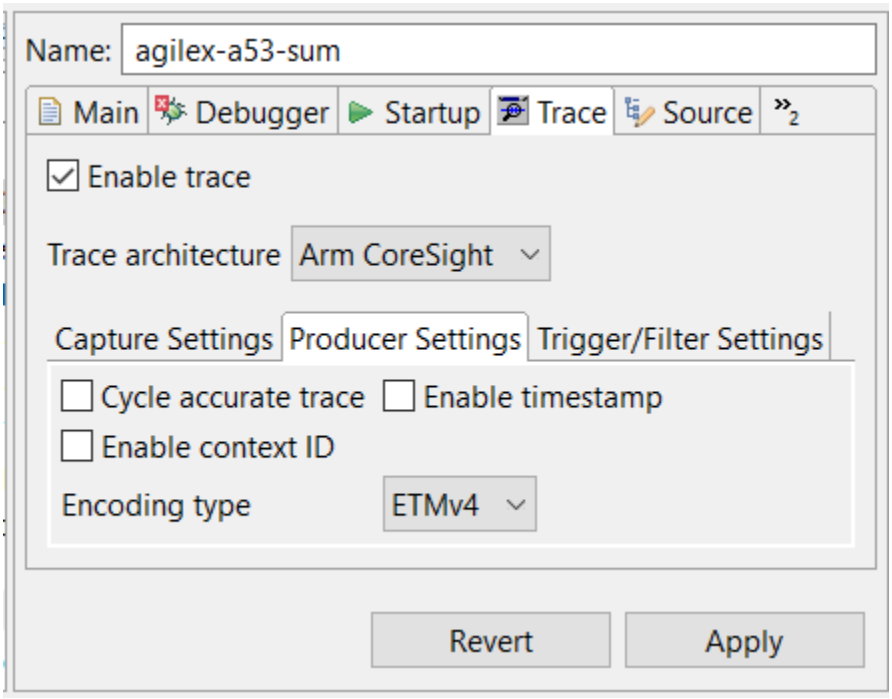
ii. **Enable context ID** — enables the tracing of the ASID and the current process ID.

iii. **Enable timestamp** — enables timestamps in your trace.

b. For **Encoding type**, you can select **ETMv4** or **PTM**. Refer to Table Arm Trace Source for more details.

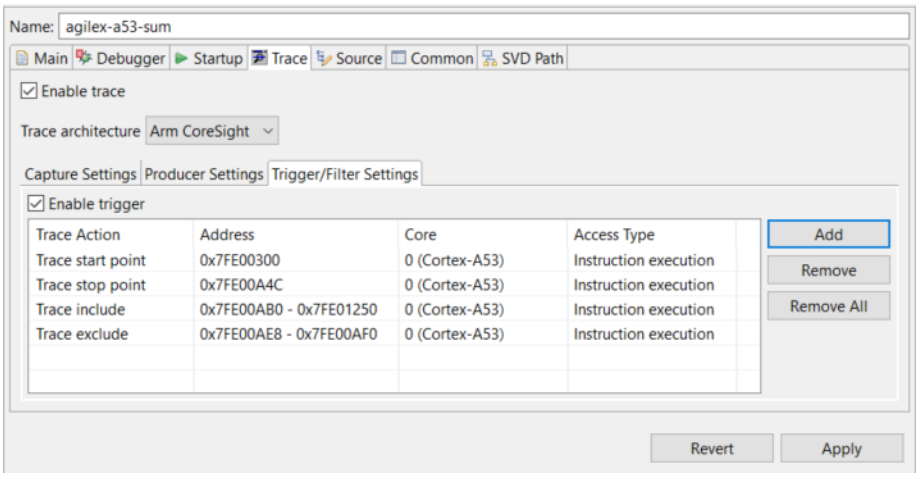
Note: **ETMv3** is an older version of **ETMv4** and not supported by Intel devices.

Figure 68. Producer Settings Sub-tab



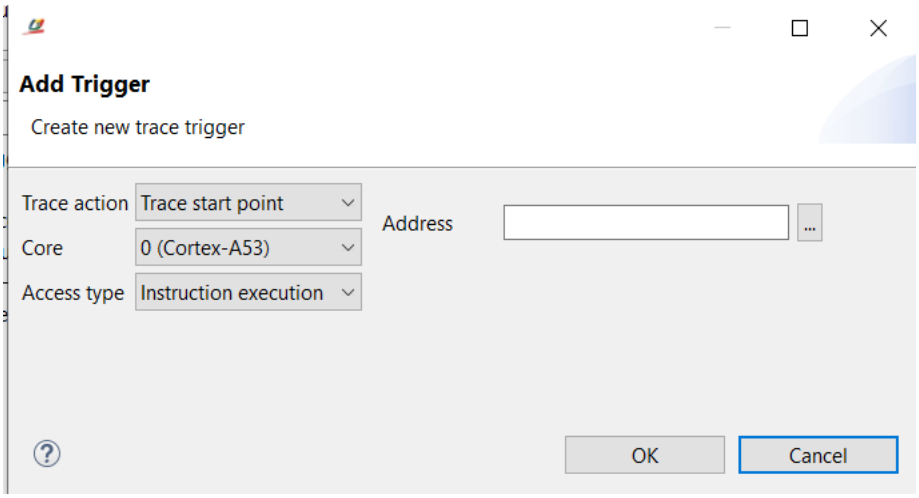
5. Under **Trigger/Filter Settings** tab, click the **Enable trigger** checkbox.

Figure 69. Trigger/Filter Settings Tab



6. Click **Add** to bring up the **Add Trigger** dialog box as shown below:

Figure 70. Add Trigger Dialog Box



a. For **Trace action**, choose one of the following options:

i. **Trace include**— limits trace to the selected address range

ii. **Trace exclude**— excludes trace from the selected address range

iii. **Trace start point**— starts tracing when the selected address is executed

iv. **Trace stop point**— stops tracing when the selected address is executed

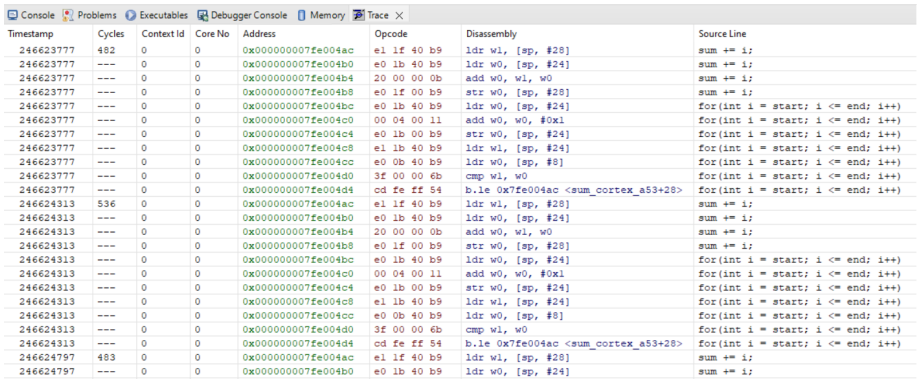
b. For **Core**, use the same core that you select in Debugger tab.

c. For **Access type**, select the only available option — **Instruction execution** as instruction trace and to support trigger.

7. Once you complete the configuration, click **Debug**.

8. Open **Trace** view via **Window ➤ Show ➤ View ➤ Other ➤ Debug ➤ Trace**. Click **Fetch, decode and populate trace data** button on the top right of the window to fetch trace data from the target and display in the Trace view as shown below.

Figure 71. Trace View



2.5.7.1. Trace Bar

You can control the trace process using the **Trace Bar**.

Figure 72. Trace View Bar Options



Table 12. Trace Control Using the Trace Bar

Trace Bar	Function	Description
1	Fetch Trace	Fetches the trace data from the target trace buffer, decode and populate it in the **Trace** view.
2	Clear Trace	Clears the trace view and the target trace buffer data.
3	First Page	Navigates the trace data when there are multiple pages of trace information.
4	Previous Page	
5	Next Page	
6	Last Page	
7	Start Capture	Starts the trace capturing. Applicable only when you manually stop the trace using Stop Capture.
8	Stop Capture	Stops the trace.

2.5.7.2. Trace View

Table 13. Trace View Information and Description

Trace Information	Description
Timestamp	The timestamp value received from the target
Cycles	The CPU cycle count information. The cycle count is available from the target for a set of instructions only. Cycle count against an instruction represents the number of CPU cycles consumed for the set of instructions starting from the last cycle count received.
Context ID	Context ID of the current execution. Context ID gives the information about the ASID and the current Process ID.
Core No	Core index of the core executing the trace data.
Address	The instruction address executed by the CPU.
Opcode	Opcode of the executed instruction.
Disassembly	Disassembly of the executed instruction.
Source Line	Source Line corresponding to the instruction execution.

Example Use Case: Disassembly View

Double click the trace row to highlight the corresponding source line and disassembly in **Source** and **Disassembly** view. Refer Figure Trace View and the following figures for more details.

Figure 73. Source Highlighting — Debug and Disassembly Tab

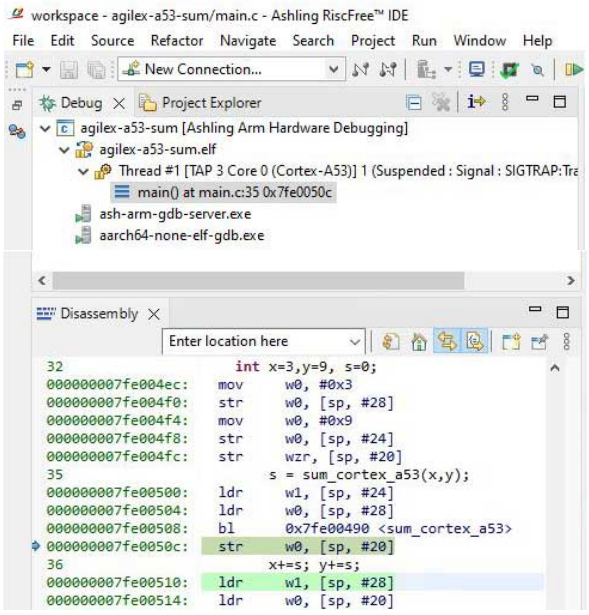
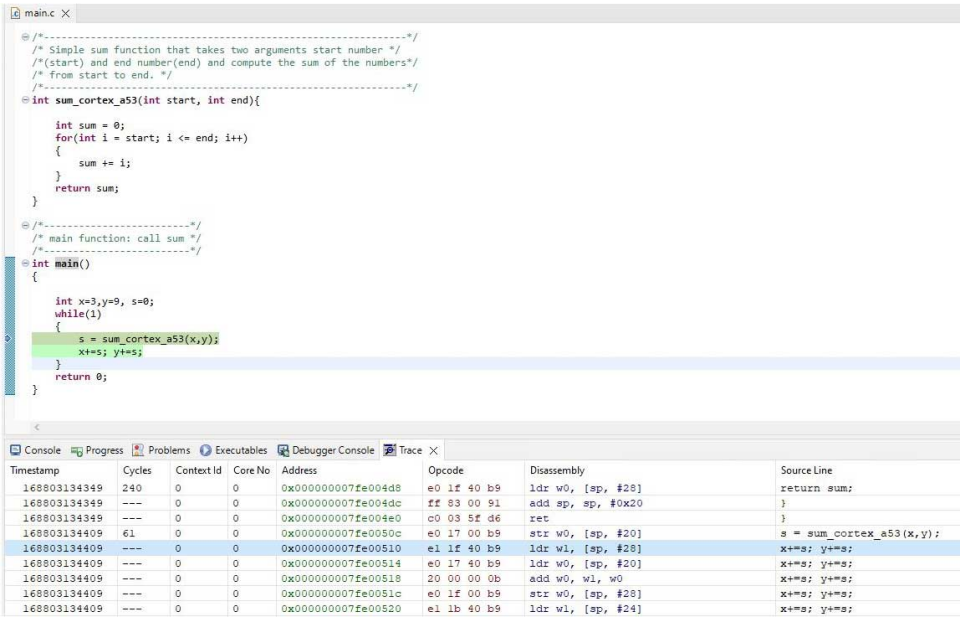


Figure 74. Source Highlighting — main.c Tab



2.5.8. Debugging the Arm Linux Kernel

You can use the Arm Linux kernel for debugging on the Agilex 7 SoC devices using Ashling RiscFree IDE for Altera FPGAs. Currently, specifically supported kernel versions are 5.15.100-lts and 6.1.20-lts. Other versions can work with reduced functionality.

2.5.8.1. Preparing Linux for Kernel Debugging

To allow kernel and module debugging, you must enable the `CONFIG_DEBUG_INFO` and `CONFIG_GDB_SCRIPTS` configurations during the Linux kernel's build process. Use the following commands to enable the configuration parameters on top of the default kernel configuration.

```
make defconfig
./scripts/config --set-val CONFIG_DEBUG_INFO  y
./scripts/config --set-val CONFIG_GDB_SCRIPTS y
make oldconfig
```

You can also change these options using the `make menuconfig` command.

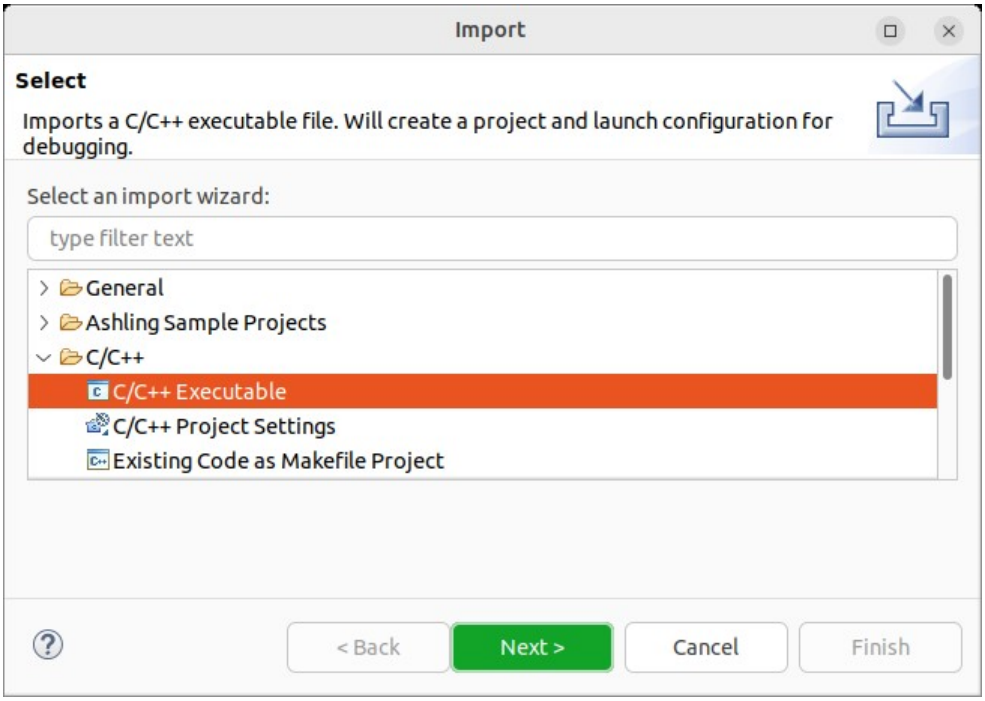
Note: Newer kernel versions define the `CONFIG_DEBUG_INFO` automatically based on other debugging options in the kernel. Refer to your kernel documentation for more details if the above command does not successfully set `CONFIG_DEBUG_INFO=y` in the resulting kernel `.config` file.

2.5.8.2. Debugging the Arm HPS Linux Kernel

Refer to the following steps to debug the Arm HPS Linux Kernel:

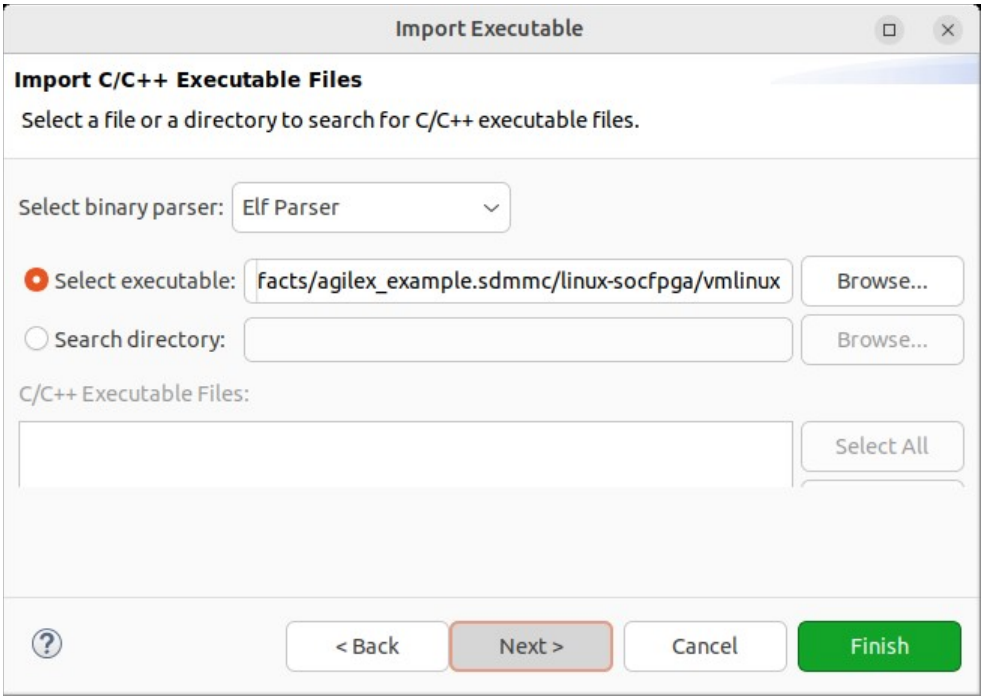
1. Boot Linux on the Agilex development board using a Linux kernel ready for debugging.
2. Go to **File ➤ Import ➤ C/C++ Executable** then click **Next**.

Figure 75. Selecting Import Wizard



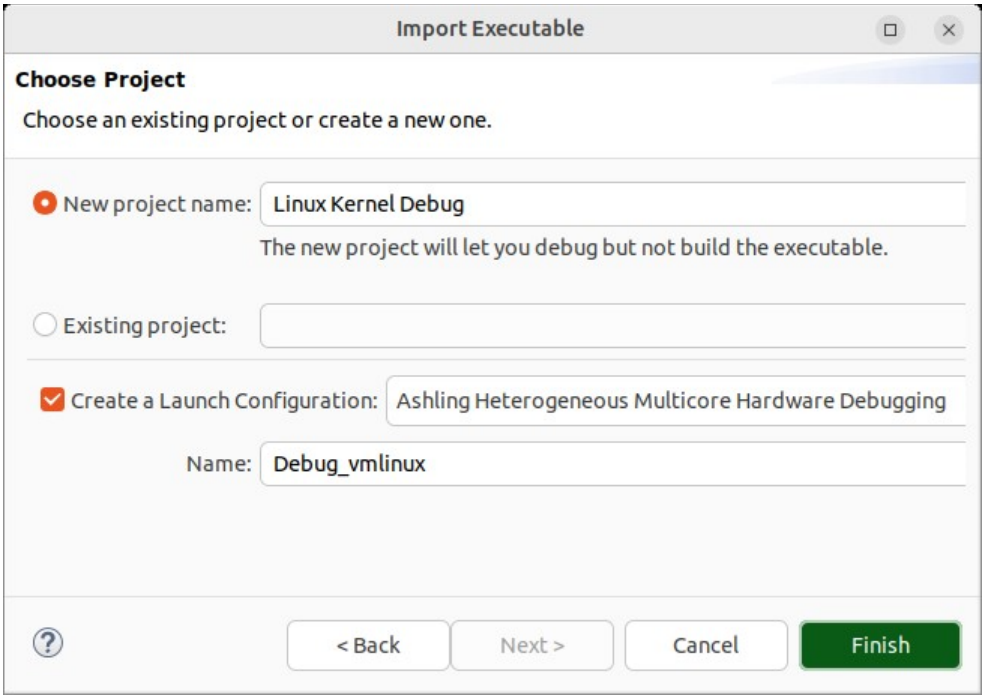
3. In the **Import Executable** window, click **Browse...** and select the **vmlinux** executable location on your computer. Click **Next**.

Figure 76. Importing C/C++ Executable Files



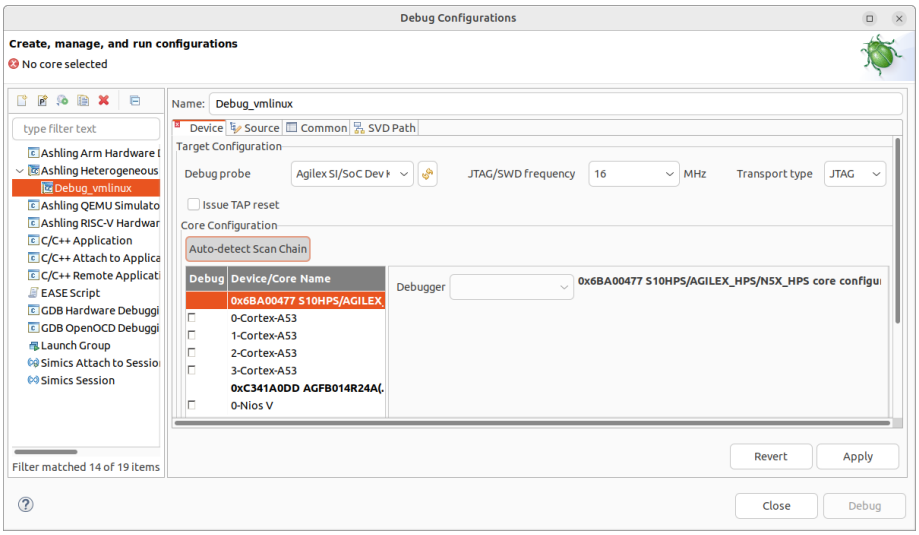
4. Use `Linux Kernel Debug` for **New project name**. Select **Ashling Heterogeneous Multicore Hardware Debug** for **Create a Launch Configuration**, then click **Finish**.

Figure 77. Choosing Project



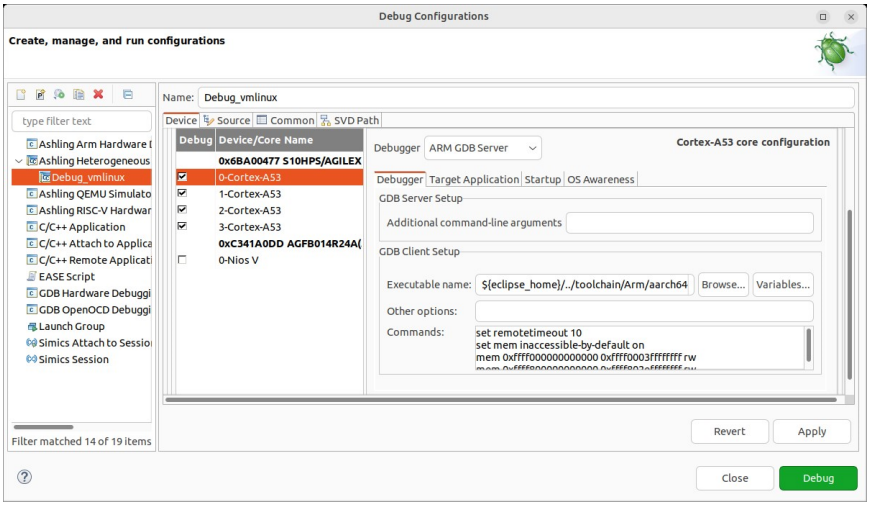
5. The **Debug Configurations** window appears. Use the **Auto-detect Scan Chain** button, if you need to populate the device list in the **Device** tab in the launch configuration. Perform and repeat the following steps for all four Cortex-A53 cores.

Figure 78. Creating, Managing, and Running Configuration



a. For the Linux debug session, check all the cores that Linux is running on as shown below.

Figure 79. Linux Cores



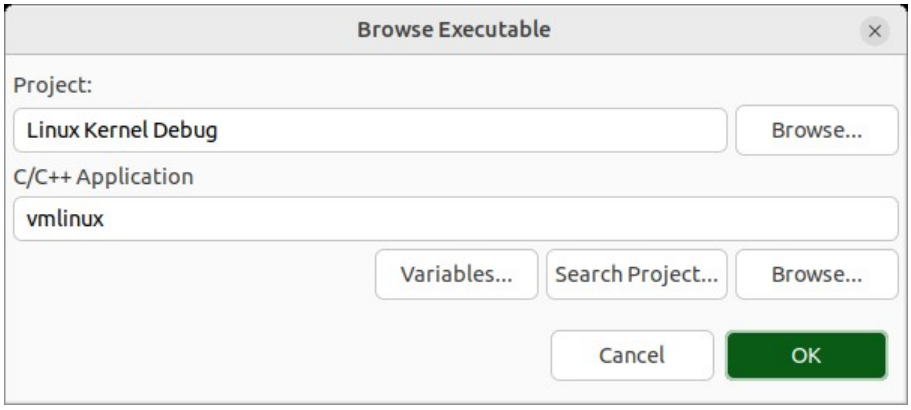
Note: Altera recommends you specify a proper memory access attributes to avoid unwanted or illegal memory access during the debug session. For example, the memory access region configured for Agilex platform is:

```
set remotetimeout 10
set mem inaccessible-by-default on
mem 0xffff000000000000 0xffff0003ffffffff rw
mem 0xffff800000000000 0xffff802effffffff rw
```

Ensure you select the default path and then select `aarch64-none-linux-gnu-gdb` as the GDB Executable name.

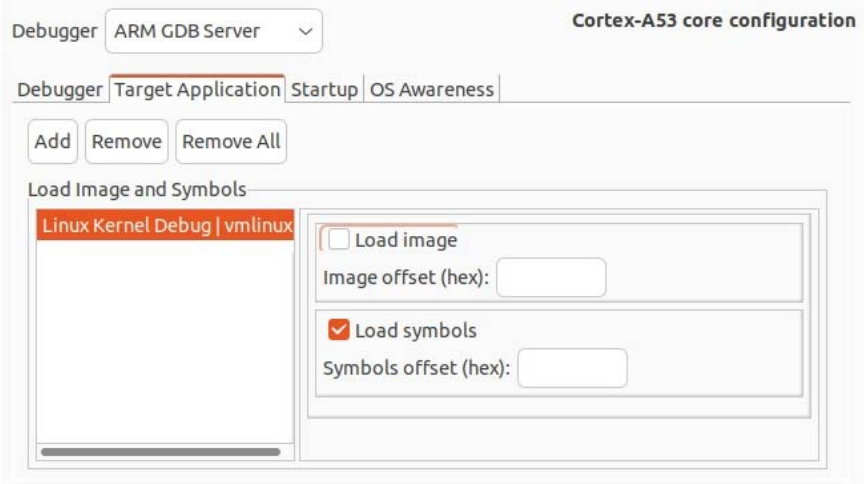
b. Click **Target Application** tab. Click **Add**. The **Browse Executable** window appears. For **Project**, click **Browse...** and select the current project. For **C/C++ Application**, click **Browse...** and select **vmlinux** as the executable. Click **OK**.

Figure 80. Browse Executable Window



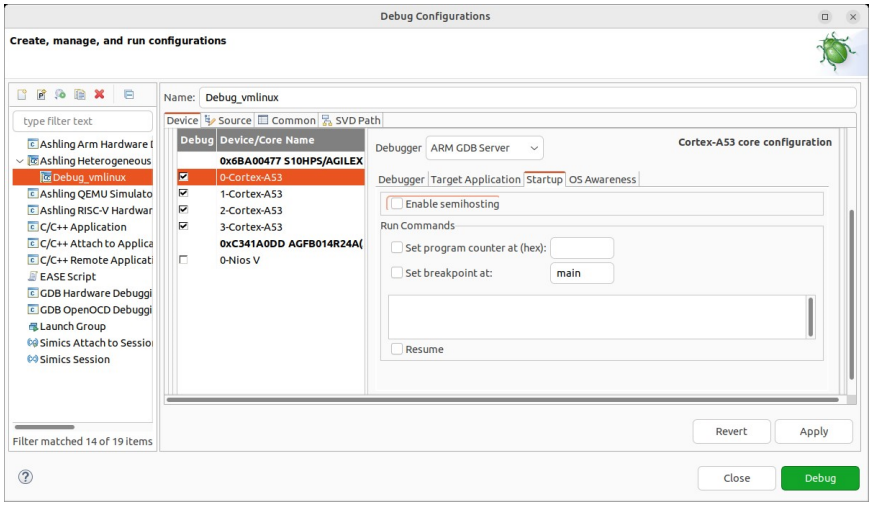
c. After adding the executable, ensure you turn off **Load image** as the image is already loaded to the target.

Figure 81. Target Application Tab



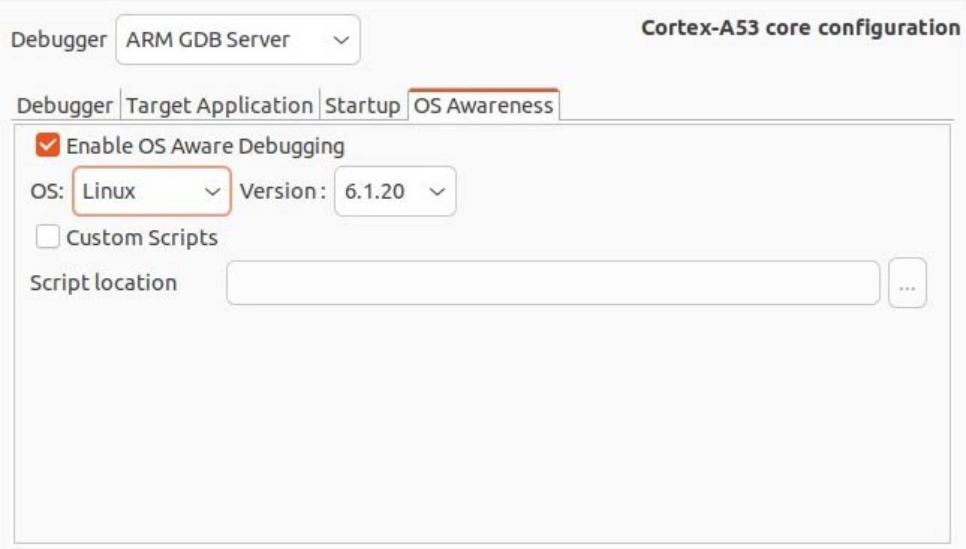
d. Click the **Startup** tab. Ensure the configuration is as shown below.

Figure 82. Startup Tab



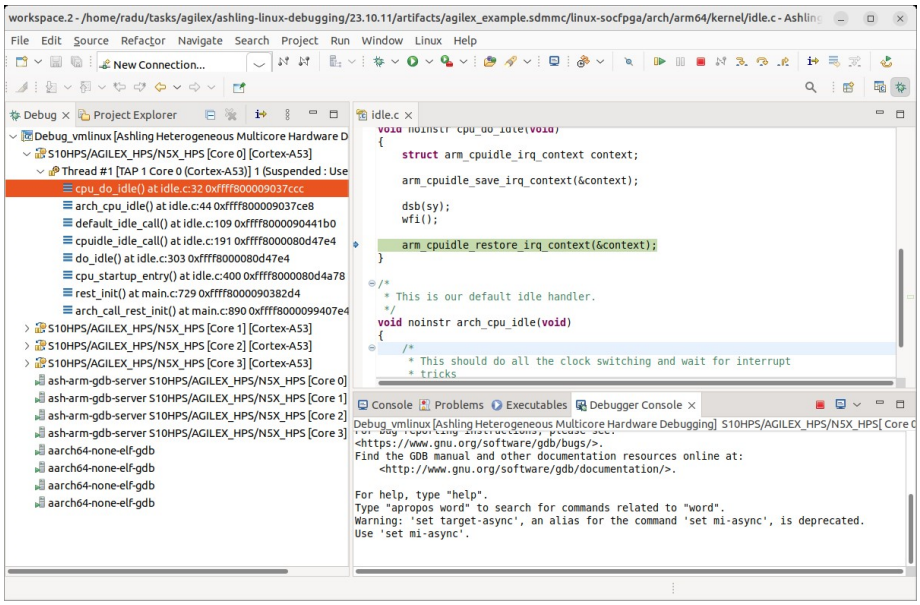
e. Go to **OS Awareness** tab. Turn on the **OS Aware Debugging** checkbox and select appropriate Linux kernel version.

Figure 83. OS Awareness Tab



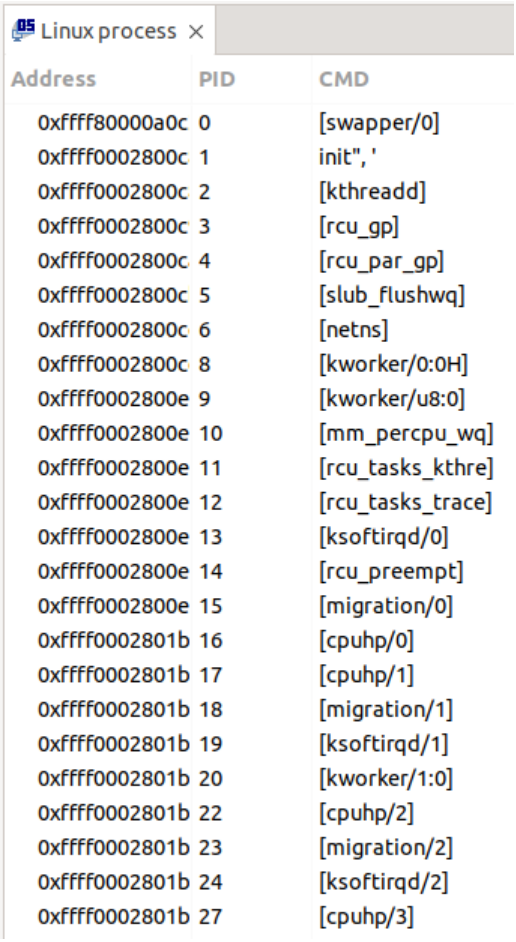
6. After you complete the configurations for all of the Cortex-A53 cores, click **Apply** then click **Debug**. The debugger connects to the Linux running on the board, then it stops the cores, and display the current code, as shown below.

Figure 84. idle.c tab



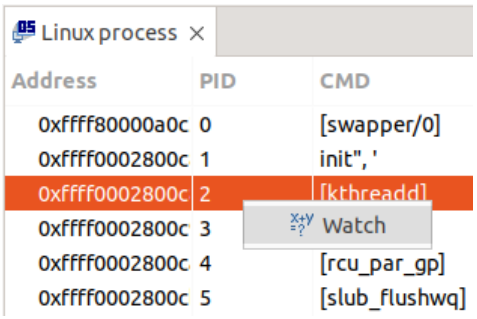
7. Go to **Linux ➤ Processes ➤ List Running Processes**, and then the debugger shows the processes.

Figure 85. Linux Process



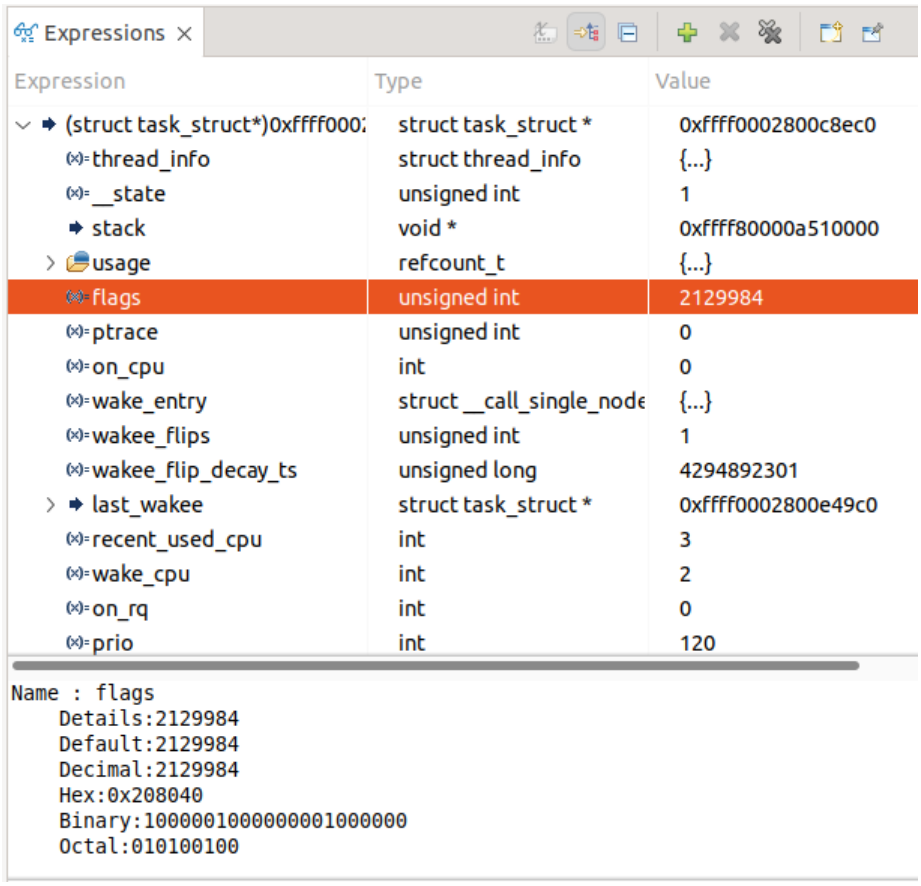
8. Right-click a process in the list, then select the **Watch** option. The debugger opens the process in the **Watch** window. You can now inspect its properties.

Figure 86. Inspecting the Process Properties



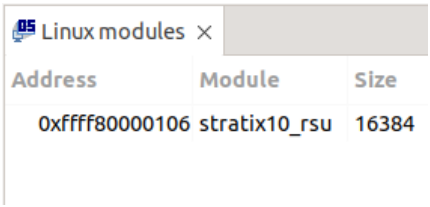
The following figure shows an example of the Linux process properties.

Figure 87. Process Properties



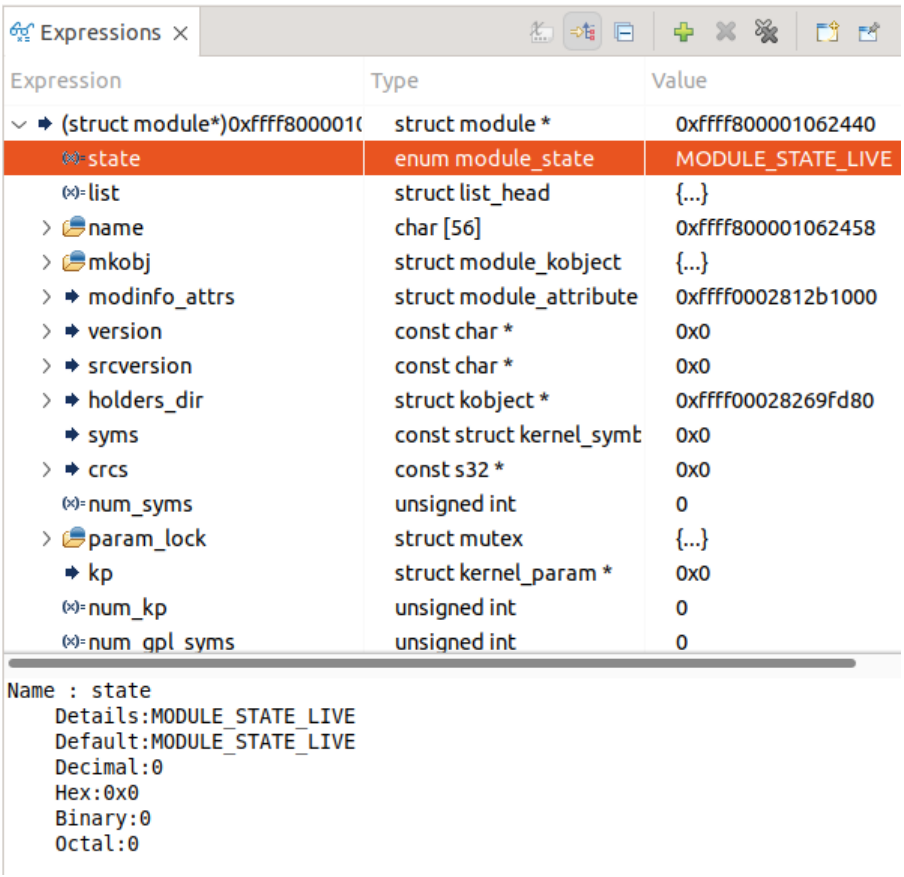
9. Go to **Linux ➤ Modules ➤ List Loaded Modules** to show the list of modules. In the example below only one module was loaded:

Figure 88. Linux Modules



10. Right-click a module in the list, then select the **Watch** option. You can inspect the module properties as shown in the figure below:

Figure 89. Module Properties



2.5.9. Debugging Target Software in an Intel Simics Simulator Session

Ashling RiscFree IDE for Altera FPGAs provides several debug capabilities that allow you to perform the target software debugging more easily than using the resources of the Intel Simics simulator for Intel FPGAs. The Ashling RiscFree IDE for Altera FPGAs GUI provides a friendly interaction with the target software, allowing you to inspect source code, set breakpoints, and observe and update variable values and memory content, among other features.

The Ashling RiscFree IDE for Altera FPGAs also allows you to perform a remote debug session connected to an Intel Simics session that can run on the same host PC or a remote system on the same network.

Related Information

- Debugging Target Software with Ashling* RiscFree* Debugger
  For more information about debugging target software.
- Use Case: Debug Bare-metal Code Using Simics-RiscFree
  For more information about the setup and procedure to perform debugging using Riscfree IDE over target software in an Intel Simics simulation.

3. Ashling Visual Studio Code Extension for Altera FPGAs

3.1. About the Ashling Visual Studio Code Extension

Ashling Visual Studio Code Extension is a set of code that runs in Visual Studio Code and provides new or improved features for Altera FPGAs Arm*-based HPS and RISC-V based Nios V processors. Ashling Visual Studio Code Extension provides a complete, seamless Visual Studio Code based C and C++ software development and has the following features:

- GUI-based debug configurations for Altera FPGA Arm HPS and Nios V soft cores such as probe selection, device selection, core selection etc.
- Auto-detect feature displaying all the devices and cores in the FPGA, allowing user to select the required core for the debug session.
- CMake based project management support, allowing for the direct import and build of Nios V processor APP and BSP projects.
- FreeRTOS and Zephyr RTOS aware debug support including tasks and event views.
- Nios V GCC compiler toolchain fully integrated with support for newlib or picolibc run-time libraries using the Nios V Hardware Abstraction Layer (HAL) API for hardware access.
- Integrated support for Intel FPGA Download Cable II JTAG debug probe.
- Custom instruction support and extensions for the Nios V processor.
- Assembly level instruction stepping support.
- ROM or RAM based debugging support (e.g., hardware breakpoints for flash-based support).

3.1.1. Supported Devices

Table 14. Supported Devices

Feature	Supported Devices
Nios V processor core debugging	• Agilex 5 • Agilex 7 • Stratix 10 • Arria 10 • Cyclone 10 GX • Stratix V • Arria V • Arria V GZ • Cyclone IV E • Cyclone IV GX

continued...

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Feature	Supported Devices
	• Cyclone V • Cyclone 10 LP • MAX 10
Arm HPS debugging	• Agilex 5 • Agilex 7 • Stratix 10 • Arria 10 • Cyclone V

3.1.2. Quartus Prime Software Support

You can use Ashling Visual Studio Code Extension with Quartus Prime Standard Edition or Quartus Prime Standard Edition.

3.2. Getting Started with Ashling Visual Studio Code Extension

Ashling Visual Studio Code Extension provides a consistent development platform that works for all Nios V embedded processor systems. You can accomplish all software development tasks within Visual Studio Code, including creating, editing, building, and debugging programs. Ashling Visual Studio Code Extension supports Nios V processor and Arm core debugging in multiple host operating system (OS) platforms.

3.2.1. Installing Ashling Visual Studio Code Extension

3.2.1.1. Prerequisites

Install the following software components before you install the Ashling Visual Studio Code Extension:

- Quartus Prime
- Ashling RiscFree IDE for Altera FPGAs
- Microsoft Visual Studio Code

Note: The Ashling Visual Studio Code Extension supports only Quartus Prime Pro Edition 25.1 and Quartus Prime Standard Edition 24.1, including their latest versions.

3.2.1.2. Installing Ashling Visual Studio Code Extension for Altera FPGAs

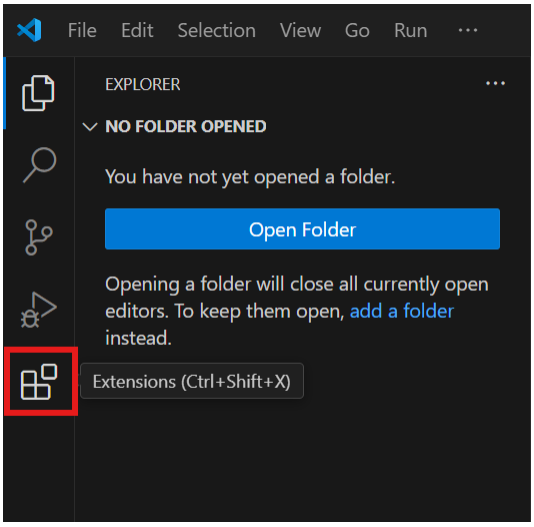
Note: To select a specific Quartus Prime version, you can either modify the default Quartus Prime environment variables or open Ashling Visual Studio Code Extension through the Nios V Command Shell. Launching Ashling Visual Studio Code Extension via the Nios V Command Shell ensures that the `Ashling: InstallationPath` is automatically configured to the desired Quartus Prime version based on the environment settings defined in the Nios V Command Shell.

1. Open the **Start Menu** and search for Nios V Command Shell. Then, launch it.
2. Type the following code to launch Ashling Visual Studio Code Extension.

```
$ code.
```

3. Click **Extensions** icon in the **Activity bar**.

Figure 90. Extensions Icon in Activity Bar



4. Navigate to **Views and More Actions ➤ Primary Side Bar ➤ Install from VSIX...**.

Figure 91. Views and More Actions in Primary Side Bar

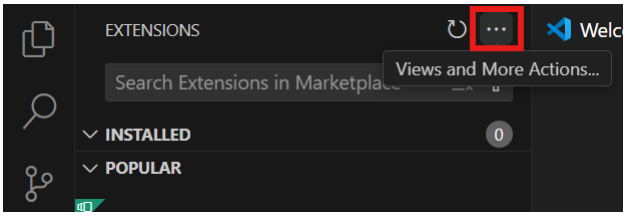
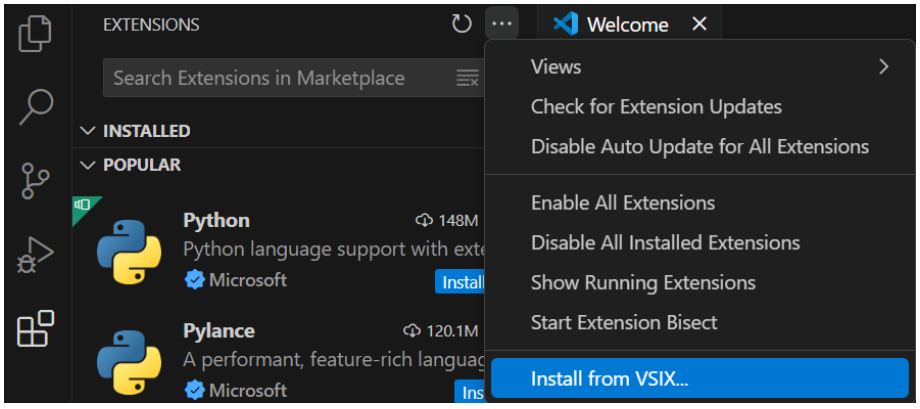


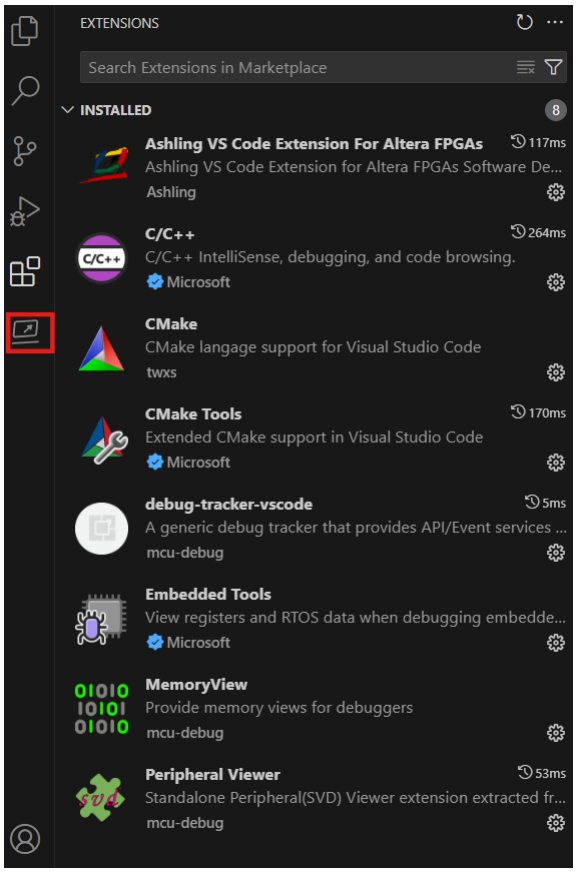
Figure 92. Install from VSIX



5. Browse to `<Quartus Prime Installation directory>/riscfree/vscode/<Ashling VSCode For Altera FPGA.vsix extension` and click **Install**.
6. Once the extension is installed, the Ashling icon appears in the **Activity** bar.

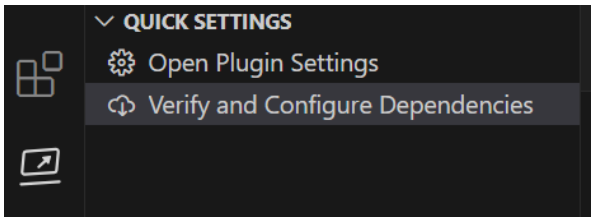
Note: After the installation process, ensure that the **C/C++**, **CMake**, **CMake Tools**, **debug-tracker-vscode**, **Embedded Tools**, **Peripheral Viewer**, and **Memory View** extensions are also installed.

Figure 93. Installed Ashling Visual Studio Code Extension for Altera FPGAs



7. Click the Ashling icon under **QUICK SETTINGS** and click **Verify and Configure Dependencies** to set the debug environment. Ashling Visual Studio Code Extension verifies the presence of Quartus Prime and Ashling RiscFree IDE for Altera FPGAs in the machine and configures the path for build tools (Make, CMake, etc.), compiler (Nios V GNU GCC toolchain), and debugger (GDB, Ashling GDB Server, Intel JTAG Server, etc.).

Figure 94. Verify and Configure Dependencies



If the extension found the Quartus Prime and Ashling RiscFree IDE for Altera FPGAs path successfully, the Ashling Visual Studio Code Extension reports successful verification in the **OUTPUT** view.

Note: Ensure that all dependencies have passed the verification process. If any dependency fails, please consult with support.

Figure 95. Output of Verify and Configure Dependencies

```
[3:57:46 PM] Starting Quartus installation search...
[3:57:48 PM] Checking environment variables:
[3:57:48 PM]   QUARTUS_ROOTDIR: C:\altera_pro\25.1\quartus
[3:57:48 PM]   QUARTUS_ROOT: not set
[3:57:48 PM]   ALTERA_ROOT: not set
[3:57:48 PM]  Found Quartus installation(s) at: C:\altera_pro\25.1\quartus
[3:57:48 PM] Searching for RiscFree installation...
[3:57:50 PM] Checking for RiscFree at: C:\altera_pro\25.1\riscfree
[3:57:50 PM]  Valid RiscFree installation found at: C:\altera_pro\25.1\riscfree
[3:57:50 PM] Verifying RiscFree toolchain...
[3:57:52 PM] Setting up VS Code configuration...
[3:57:53 PM]  Setup completed successfully!
[3:57:53 PM]
Checking toolchain components...

[3:57:53 PM] Checking Ashling SDK Installation
==============================
[3:57:53 PM] Verifying Build Tools
------------------------------
   Checking CMake........
      Location: C:\altera_pro\25.1\riscfree\build_tools\cmake\bin\cmake.exe
      Getting version........ 3.30.2
   Checking Make........
      Location: C:\altera_pro\25.1\riscfree\build_tools\bin\make.exe
   Checking Ashling CLI........
      Location: C:\altera_pro\25.1\riscfree\cli\ashling-cli.exe

[3:57:55 PM] Verifying Debug Tools
------------------------------
   Checking ARM GDB Server........
      Location: C:\altera_pro\25.1\riscfree\debugger\gdbserver-arm\ash-arm-gdb-
server.exe
   Checking RISC-V GDB Server........
      Location: C:\altera_pro\25.1\riscfree\debugger\gdbserver-riscv\ash-riscv-
gdb-server.exe
   Checking QEMU RISC-V........
      Location: C:\altera_pro\25.1\riscfree\qemu\qemu-system-riscv32.exe
      Getting version........ 9.0.2
[3:57:56 PM] Verifying RISC-V Toolchain
----------------------------------
   Checking RISC-V GCC (32-bit)........
      Location: C:\altera_pro\25.1\riscfree\toolchain\riscv32-unknown-elf\bin
\riscv32-unknown-elf-gcc.exe
      Getting version........ 13.2.0
   Checking RISC-V G++ (32-bit)........
      Location: C:\altera_pro\25.1\riscfree\toolchain\riscv32-unknown-elf\bin
\riscv32-unknown-elf-g++.exe
   Checking RISC-V GDB (32-bit)........
      Location: C:\altera_pro\25.1\riscfree\toolchain\riscv32-unknown-elf\bin
\riscv32-unknown-elf-gdb.exe
      Getting version........ Version unknown

[3:57:58 PM] Verifying ARM Toolchain
--------------------------------
   Checking ARM GDB (AArch64 ELF)........
      Location: C:\altera_pro\25.1\riscfree\toolchain\Arm\aarch64-none-elf\bin
\aarch64-none-elf-gdb.exe
      Getting version........ Version unknown

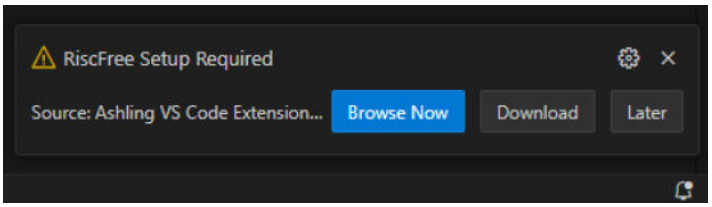
[3:57:58 PM] Summary
=======
Components Found: 10/10

 All components verified successfully
```

If the extension couldn't locate the Quartus Prime and Ashling RiscFree IDE for Altera FPGAs path automatically, the Ashling Visual Studio Code Extension extension prompts the **Browse Now** dialogue box. You are required to browse and select the location of the missing component(s).

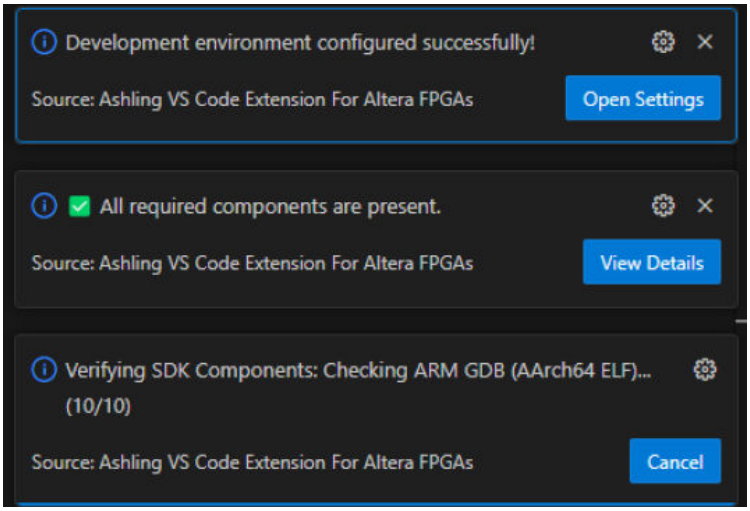
For example, the Ashling RiscFree IDE for Altera FPGAs folder can be found within the Quartus Prime installation folder. (`<Quartus Installation Directory>\altera_pro\XX.Y\riscfree`).

Figure 96. RiscFree Setup



The following messages appear when the missing component is found and configured successful by the extension.

Figure 97. Verify and Configure Dependencies Successful



3.2.2. Getting Started with Ashling Visual Studio Code Extension

Writing software for the Nios V processor is like writing software for any other microcontroller family. An easy way to start designing effectively is to purchase a development kit from Altera FPGA that includes documentation, a ready-made evaluation board, a getting-started reference design, and all the development tools necessary to write Nios V processor programs.

This section guides you through the most fundamental operations in the Ashling Visual Studio Code Extension and the following aspects:

- How to start a software project for Nios V processor.
- How to import an application (APP) project and the Board Support Package (BSP) project required to interface with your hardware.
- How to build the application in the Ashling Visual Studio Code Extension.
- How to run and debug the software on an FPGA development board.

3.2.2.1. Ashling Visual Studio Code Extension Workbench

The term workbench refers to the Ashling Visual Studio Code Extension desktop development environment. You can use the workbench to edit, compile and debug your programs in Ashling Visual Studio Code Extension.

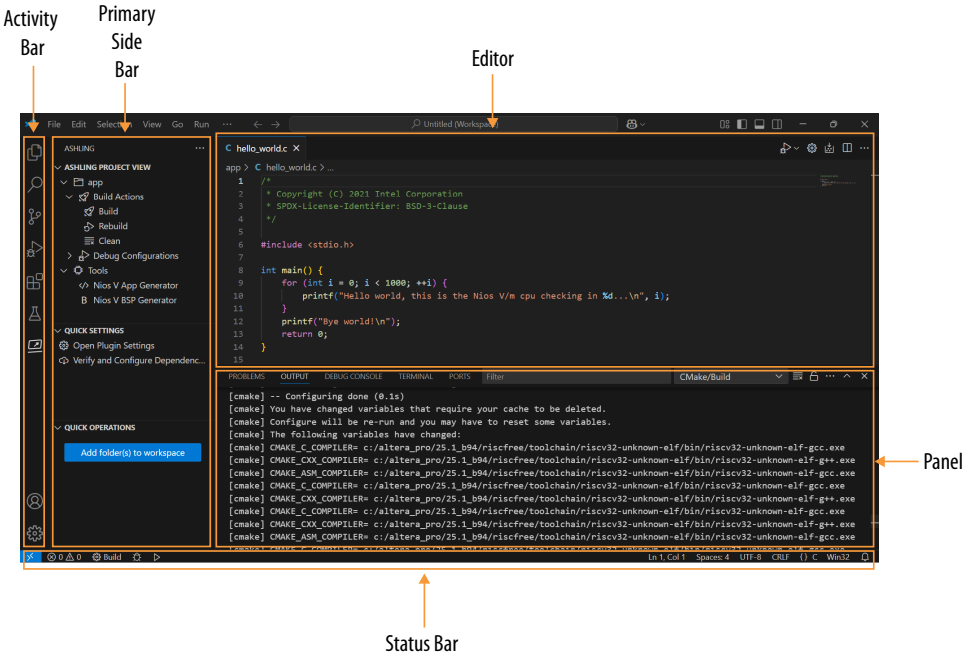
3.2.2.1.1. Perspective, Editors, and Views

Visual Studio Code has a simple layout that optimizes editor space while letting you navigate your project effectively. The user interface is divided into five main areas.

Table 15. User Interface Descriptions

User Interface	Description
Editor	• Main area for editing your files. • You can have many editors open side by side, both vertically and horizontally.
Primary Side Bar	• Contains different perspectives such as the Explorer to help you work on your project.
Status Bar	• Provides information about the currently open project and the files you are editing.
Activity Bar	• Located on the far left-hand side. • Allows you to navigate between views and displays extra context-specific indicators. • You can change the position of the Activity Bar.
Panel	• An additional space for views below the editor region. • By default, it contains output, debug information, errors and warnings, and an integrated terminal. • You can change the position to the left or right for more vertical space.

Figure 98. User Interface



A dot (•) indicates that an editor has unsaved changes. Views can also provide their own menus and toolbars, which, if present, appear along the top edge of the view. To open the menu for a view, click the drop-down arrow icon at the right of the view's toolbar or right-click in the view. A view might appear on its own or stacked with other views in a tabbed notebook.

Related Information

Visual Studio Code User Interface

3.2.2.1.2. Starting a Project

You can start a project in Ashling Visual Studio Code Extension by importing an existing project.

3.2.2.1.3. Navigating the Project

After importing a Nios V processor project, it shows up in the **Explorer** and **Ashling Project View**.

- To examine the software project's folders, files, and source codes, refer to the **Explorer**.
- To build and debug the software project, refer to the **Ashling Project View**.

Figure 99. Explorer View

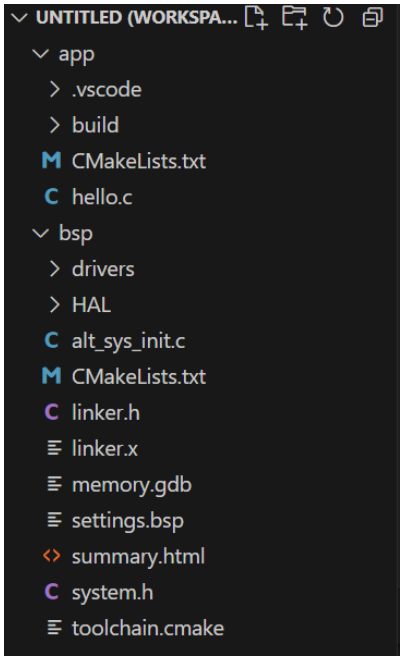
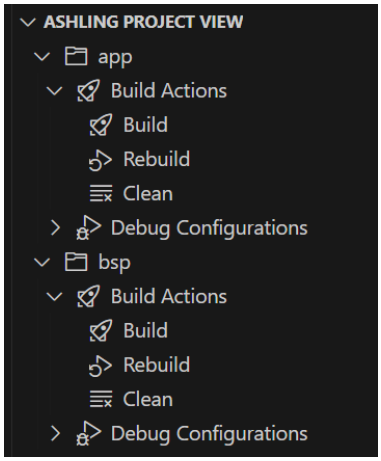


Figure 100. Ashling Project View

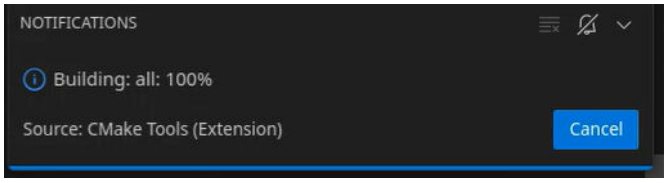


3.2.2.1.4. Building the Project

To build a Nios V processor project in the Visual Studio Code, click **Build Project**. A **NOTIFICATIONS** window shows you the build status at the bottom right corner. The build process can take a minute or two for a simple project, depending on the speed of the host machine. Building a complex project takes longer. During the build process, you can view the build commands and command-line output in the **OUTPUT** view.

If the project is dependent on another project, such as a BSP or a user library, Visual Studio Code builds the dependency project first. This feature allows you to build an application and its BSP with a single command.

Figure 101. Build Project



3.2.2.1.5. Configuring the FPGA

Before running your software, you must ensure the correct hardware design runs on the FPGA. To configure the FPGA, use the Quartus Prime Programmer from the Quartus Prime software.

The project directory for your hardware design contains an SRAM Object File (`.sof`) and the `.qsys` file. The `.sof` file contains the hardware design to be programmed in the FPGA.

3.2.2.1.6. Debugging the Project

To debug a software project, click on **Debug Configurations** and select **Nios V Hardware Debugging**. This command carries out the following actions:

- Creates a **Nios V Hardware Debugging**.
- Builds the project executable. If all target files are up to date, nothing is built.
- If you are debugging on hardware, it establishes communications with the target and verifies that the FPGA is configured with the correct hardware design.
- Downloads the `.elf`to the target memory.
- Sets a breakpoint at the top of `main()`.
- Starts execution at the `.elf`entry point.

The Visual Studio Code debugger allows you to perform many common debugging tasks. Debugging a Nios V program with the Nios V plugins is generally the same as debugging any other C/C++ program with Visual Studio Code.

3.2.3. Debug Configurations using Ashling Visual Studio Code Extension for Altera FPGAS

You can click the **+** icon beside the **Debug Configuration** to open the debug configuration dialog box. Each debug configuration is presented on several tabs. This section describes each tab.

Table 16. Run (Debug) Configuration

Tab	Description
Main	• Specify the application project to run.
Debugger	• Control the connection between the host machine and the target hardware. • You can control the connection in the following ways: — Select the **Debug Probe** in the **Debug Probe Configuration**, if there is more than one debug probe available. — Click **Auto-detect Scan Chain** to detect JTAG scan chain information of the target device automatically. — Use **GDB Server Setup** to debug the software application remotely. — Use **GDB Client Setup** to communicate with the Ashling GDBServer via a TCP connection using GDB Remote Serial Protocol. • To modify GDB Server Setup and GDB Client Setup using the command line, you can refer to: — Running Ashling GBD Server for GDB Server Setup — Running GDB for GDB Client Setup *Note:* By default, the setup is generic. It can support all environments.
Startup	• Configure options related to software debugging. • Such as loading images or symbols, and setting breakpoints, which facilitates software analysis and troubleshooting. • You can use load image and symbol specifically for booting. • If you want to download your program to analyze and troubleshoot your program during execution, turn on both load image and symbols. • If your program is in booting mode and you want to analyze and troubleshoot your program during execution, turn off load image. *Note:* Nios V processor supports hart ID of 0 only
Source	• Specify the source code when the source code is not present in the compilation path. • This allows mapping of the compile-time paths for source to local source locations. It is an object of key/value pairs and resolves the first string-matched path. For example, `"sourceFileMap": { "/mnt/c": "c:\\" }` maps any path returned by the debugger that begins with `/mnt/c` and convert it to `c:\\`.

3.2.4. Debug Information in the Ashling Visual Studio Code Extension for Altera FPGAs

Debug perspective allows you to manage the debugging and execution of a program in the Workbench. You can control how your program is executed by establishing breakpoints, pausing started programs, walking through your code, and analyzing the contents of variables.

You can find the following information in the Debug perspective,:

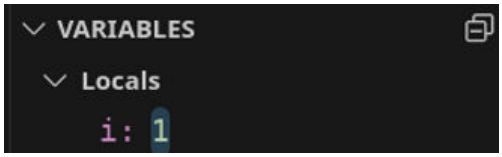
- Each target you are debugging has its own stack frame for suspended threads.
- Every thread in your program is displayed as a node in a tree structure.
- Each program you are running has a corresponding process.

Additionally, the Debug view interacts with the C/C++ Editor, highlighting the location of the execution pointer as you navigate through your code.

3.2.4.1. Variables

The **Variables** section displays information about the variables in a specified stack frame. When execution is paused, the updated values are, by default, indicated in blue. As with the other debug-related views, the view does not refresh when you run your program; it is refreshed when the execution is terminated.

Figure 102. Variables Section



3.2.4.2. Expressions

An expression is a code snippet that produces a result when evaluated. The specific debug model determines the context of an expression. Some expressions might need to be evaluated at a specific area in the program for the variables to be referenced. The Expressions section contains information about expressions.

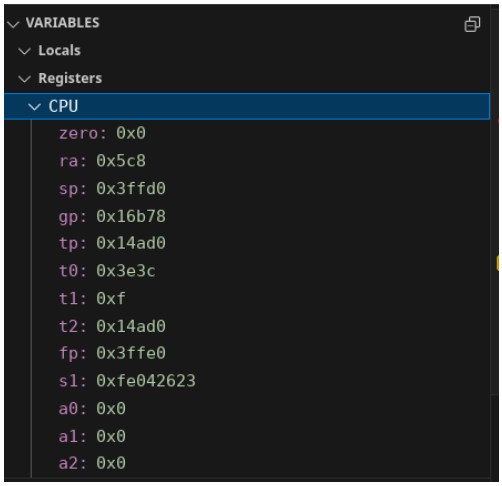
Figure 103. Expressions Section



3.2.4.3. Registers

You can see details regarding the registers in a chosen stack frame by using the **Registers View** feature. When your program pauses or ends, the modified values are highlighted in the **Registers** section.

Figure 104. Registers Section

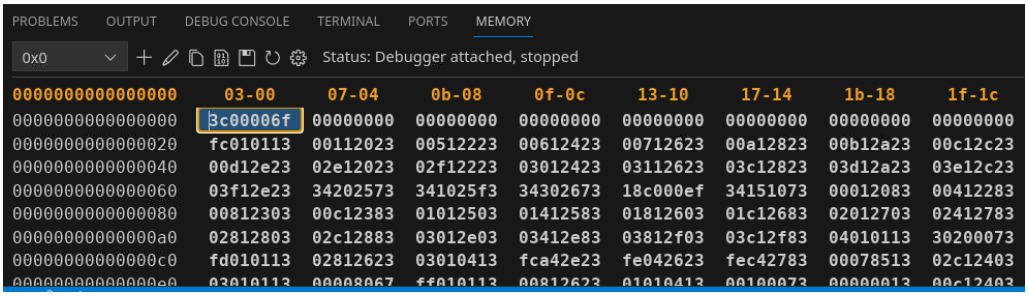


3.2.4.4. Memory

You can observe and modify the memory used by your processes in **MemoryView**. A list of so-called memory monitors represents the process memory. Every monitor depicts a memory region identified by its base address.

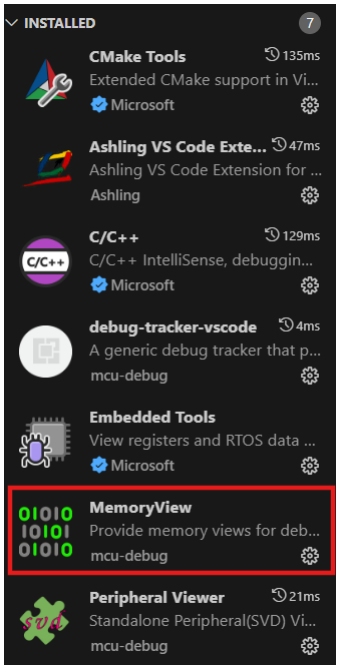
Note: You must select little-endian in the **MemoryView** properties because the Nios V processor is a little-endian processor.

Figure 105. Memory Viewer



Note: To view memory, make sure the **MemoryView** extension is installed. This extension is automatically installed when you install the Ashling Visual Studio Code Extension.

Figure 106. MemoryView



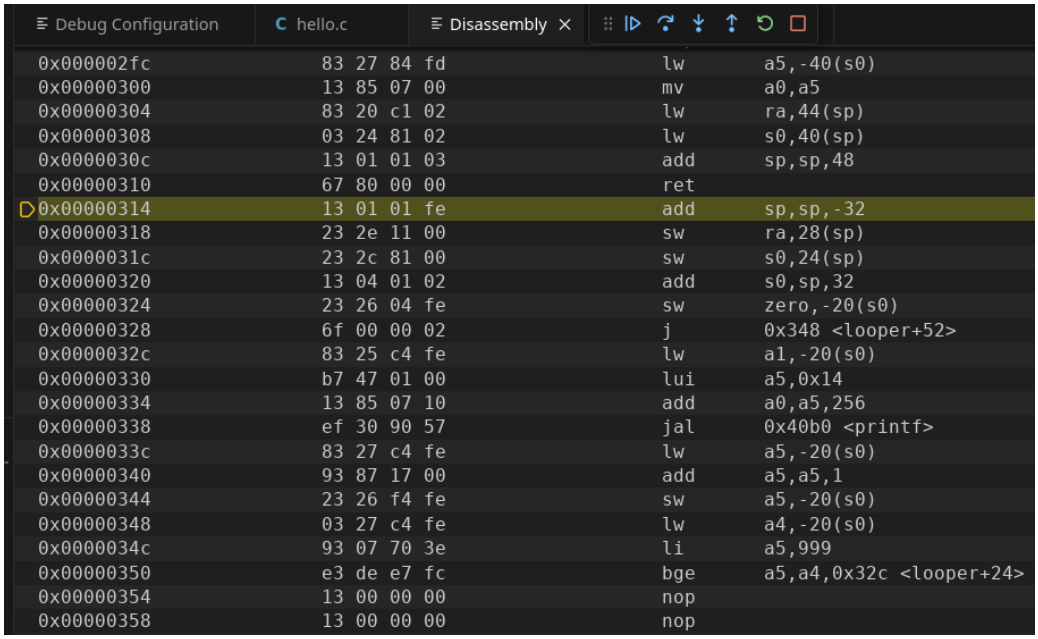
3.2.4.5. Disassembly

The **Disassembly** view shows the loaded program as assembly instructions alongside the source code for easy comparison. An arrow marker and a highlight in the view show the line that is currently running.

This view enables you to perform the following tasks:

- Set breakpoints at the start of any assembler instruction
- Enable and disable breakpoints and their set their properties
- Step through the disassembly instructions of your program
- Jump to specific instructions in the program

Figure 107. Dissassembly View



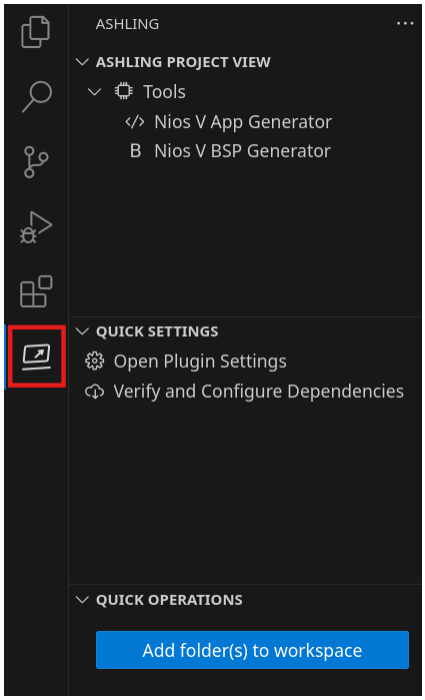
3.3. Using Ashling Visual Studio Code Extension with Nios V Processor System

3.3.1. Creating Nios V Processor BSP using Nios V Processor BSP Generator

To create a Nios V processor BSP to Visual Studio Code, follow these steps:

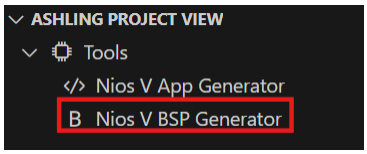
1. Click the **Ashling** icon to open the **Ashling Sidebar**.

Figure 108. Ashling Sidebar



2. Go to **Ashling Project View ➤ Tools ➤ Nios V BSP Generator**.

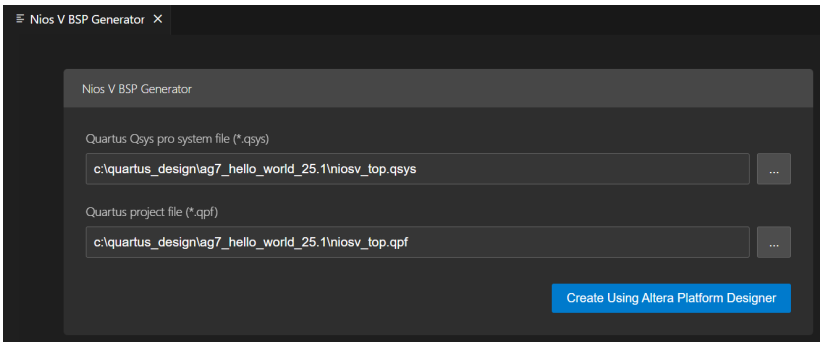
Figure 109. Nios V App Generator



3. The method to launch the BSP Editor differs based on the Quartus Prime edition. The tables below illustrate the differences between both versions. Follow the steps corresponding to your version.

- Quartus Prime Pro Edition
  a. A **Nios V BSP Generator** window appears. Click the **...** icon to browse and select the `.qsys` file for **Quartus Qsys pro system file (*qsys)**.
  b. Click the **...** icon to browse and select the `.qpf` file for **Quartus project file (*qpf)**.

Figure 110. Nios V BSP Generator Window



  c. Click **Create Using Altera Platform Designer** to launch the Platform Designer.
  d. In Platform Designer, go to **File ➤ New BSP**.
- Quartus Prime Standard Edition
  — Launch directly from **BSP Editor**.

Related Information

AN 980: Nios® V Processor Quartus® Prime Software Support
For more information about the subsequent steps to create the BSP.

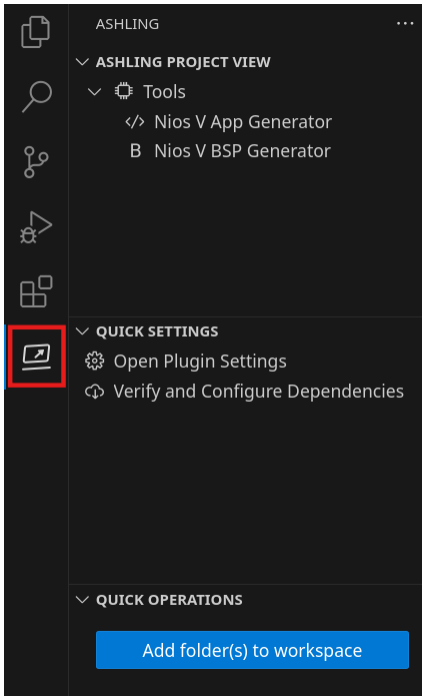
3.3.2. Creating Nios V Processor Application Project using Nios V App Generator

Note: You have to generate your BSP before generating the Nios V Processor Application Project.

Follow these steps to create a Nios V processor application project for Visual Studio Code.

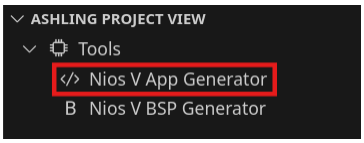
1. Click the **Ashling** icon to open the **Ashling Sidebar**.

Figure 111. Ashling Sidebar



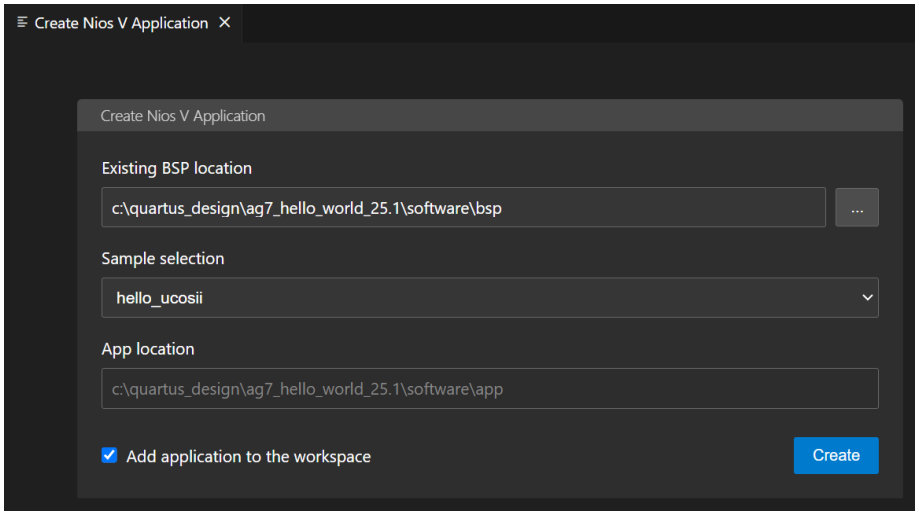
2. Go to **Ashling Project View ➤ Tools ➤ Nios V APP Generator**.

Figure 112. Nios V App Generator



3. A **Create Nios V Application** window appears. For **Existing BSP location**, click the **...** icon to identify the App location automatically.

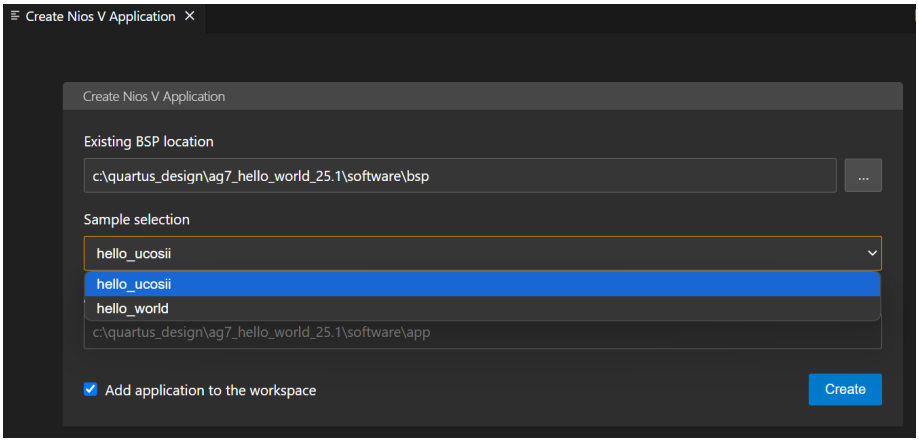
Figure 113. Create Nios V Application - Existing BSP location and App Location



4. Select the sample code in the **Sample selection**.

Figure 114. Create Nios V Application – Sample Selection

The default sample selections are `hello_ucosii` and `hello_world`.



Note: You can manually add your custom sample code folder to the following directory: `<Quartus Installation Directory>\niosv\examples\software`. Once you place the sample code folder in this directory, it is available in the **Sample selection** drop-down menu.

Figure 115. Manually Adding Custom Sample Code to Sample Selection Menu (in VS Code)

This figure shows the manually added sample code, which is `hello_from_niosvg`.

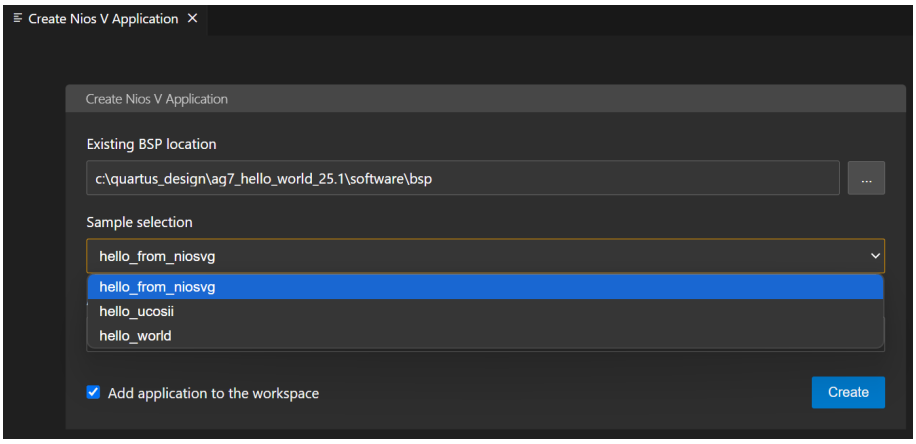
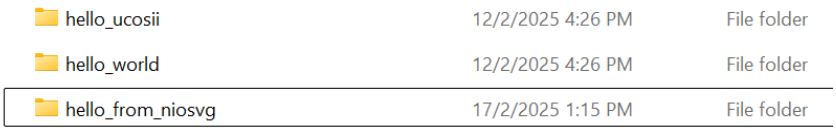
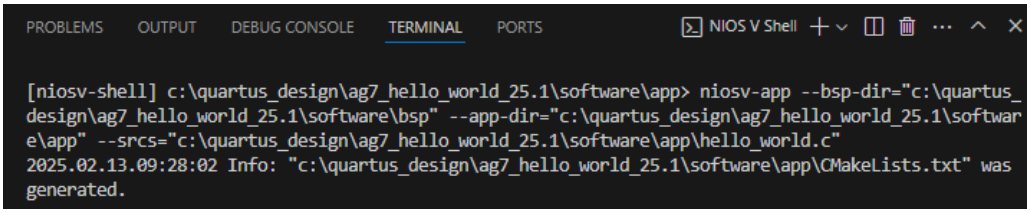


Figure 116. Manually Adding Custom Sample Code to Sample Selection Menu (in Quartus Prime Installation Folder)



5. Leave the **Add application to the workspace** option as default and click **Create**. Once the application is successfully generated, you can find the prints in the terminal as shown in the following diagram.

Figure 117. Terminal Prints



Note: If you want to include more sources into the CMakeLists.txt, you can use the up arrow key to retrieve the previous command (`niosv-app`) in the terminal.

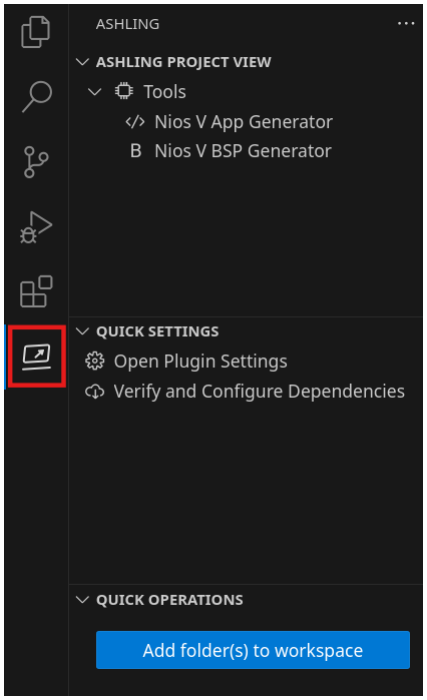
3.3.3. Importing Nios V Processor Project

Note: To run Visual Studio Code, use the Nios V Command Shell to execute it.

To import Nios V processor BSP and application project to Visual Studio Code, follow these steps:

1. Click the **Ashling** icon to open the **Ashling Sidebar**.

Figure 118. Ashling Sidebar



2. Under **QUICK OPERATIONS**, click **Add Folder(s) to workspace** and browse to the folder containing the CMake based Nios V processor application project (`<Project Directory>\software\app`).
3. Repeat the same steps to import CMake based Nios V processor BSP project (`<Project Directory>\software\bsp`).

Related Information

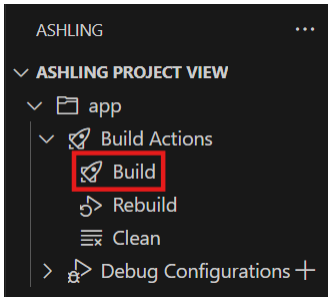
Nios® V Processor Software Developer Handbook: Nios® V Processor Software Development Flow
For more information about Nios® V Processor Software Development Flow. The chapter describes BSP generation from the hardware system and CMakeLists file generation for application before importing the binaries into Ashling Visual Studio Code.

3.3.4. Building Nios V Processor Project

Follow these steps to build the project,:

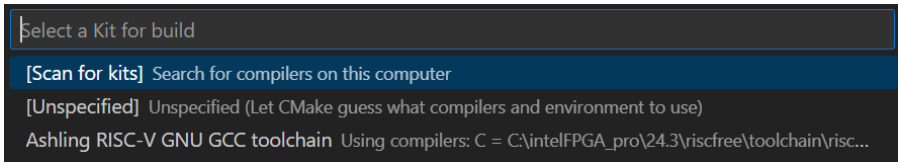
1. Go to **ASHLING PROJECT VIEW ➤ app ➤ Build Actions ➤ Build**.

Figure 119. Build Project



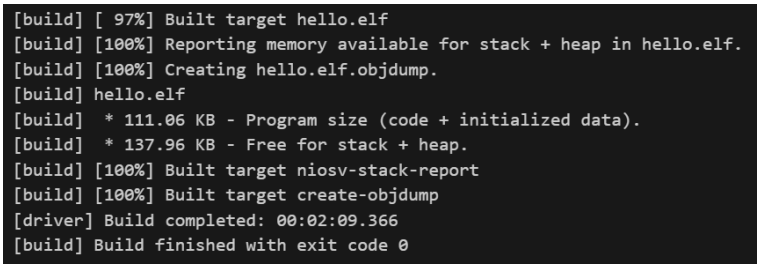
2. Select **Ashling RISC-V GNU GCC toolchain**.

Figure 120. Build Project—Select Kit



3. Once the application is successfully built, you can find the console prints as shown in the following diagram.

Figure 121. Console Prints



3.3.5. Debugging a Nios V Processor Project

3.3.5.1. Downloading and Debugging Nios V Processor Application

You can download and debug a Nios V processor software project on the targeted Altera FPGA using the Visual Studio Code.

Follow these steps to download and debug the project,:

1. Go to **ASHLING PROJECT VIEW ➤ app ➤ Debug Configurations**. Click the **+** icon beside **Debug Configurations** to list down the supported launch configurations.
2. Click **Nios V Hardware Debugging** and specify a name for the launch.

Figure 122. Debug Configurations

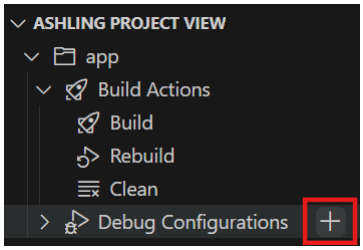
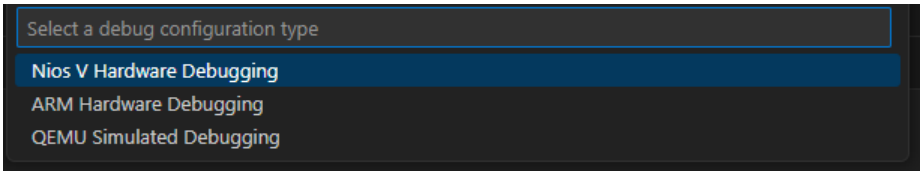
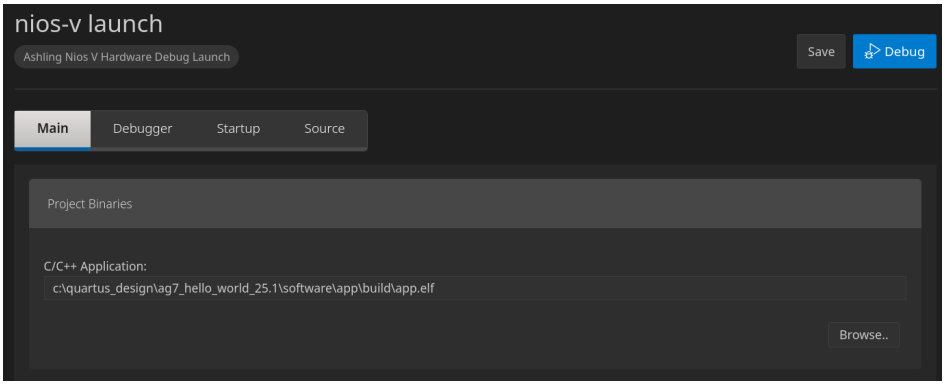


Figure 123. Create Nios V Hardware Debugging



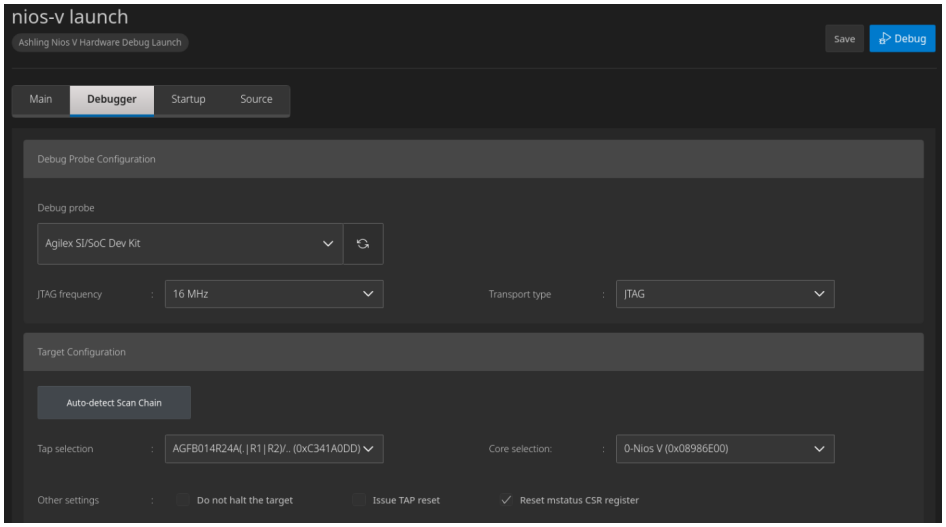
3. A debug configuration window appears. An `.elf` file is auto selected from the project in the **Main** tab. You can select a different `.elf` by using the **Browse** feature.

Figure 124. Debug Configurations for Nios V Processor — Main Tab



4. Click **Debugger** tab, click the drop-down icon to select appropriate debug probe for the debug session.
5. Under the **Debugger** tab, set these settings:
   - Debug probe: **Agilex SI/SoC Dev Kit (Name of the FPGA board/ development kit)**
   - Transport type: **JTAG**.
   - JTAG frequency: **16 MHz**.
6. Click **Auto-detect Scan Chain** to automatically detect JTAG scan chain information of the target device. Make your selection for **Tap selection** and **Core selection**.

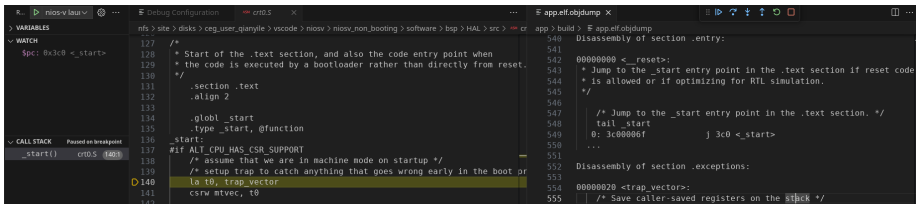
Figure 125. Debug Configurations for Nios V Processor — Debugger Tab



7. Leave all other settings as they are, including those in the **Startup** tab, as the sensible defaults have been set.

   Note: If you turn off the **Resume** feature, you can start debugging from the processor initialization stage.

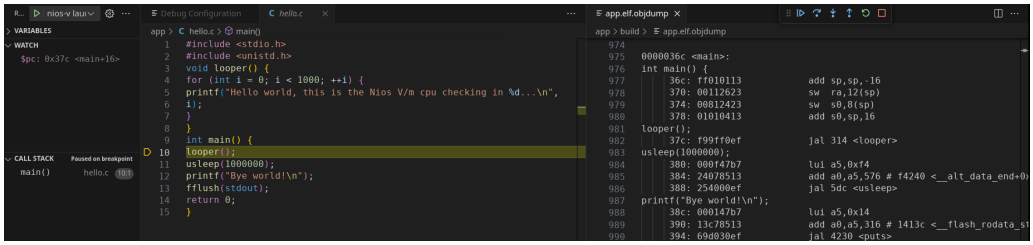
Figure 126. Debug Configurations for Nios V Processor — Turn off Resume



This example shows that the program counter is at `0x3c0`, which is displayed in the disassembly just after the reset vector. The processor jumps to `0x3c0`, which is `_start` (the beginning of the Nios V processor initialization).

If you turn on the **Resume** feature, you can debug from the user application.

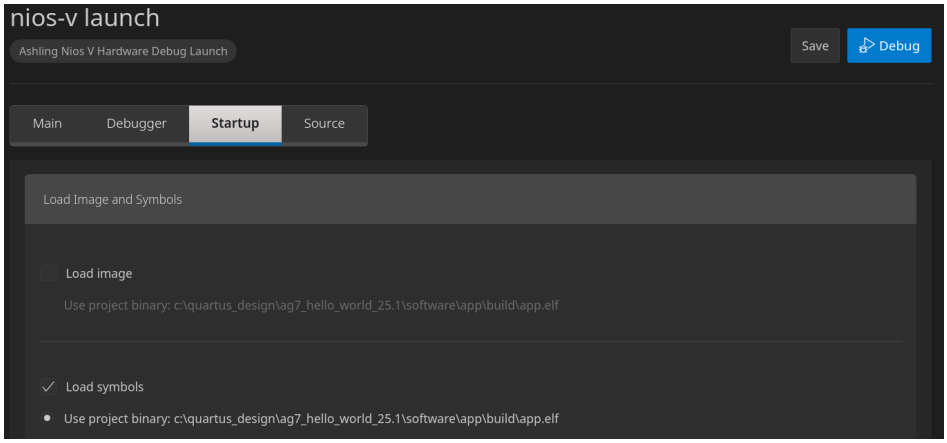
Figure 127. Debug Configurations for Nios V Processor — Turn on Resume



This example shows that the program counter is at 0x37c, which is paused at the main function and is shown in the disassembly. The processor then jumps to 0x37c, which is paused at `looper()`.

8. Click **Debug** in the top right corner of the launch configuration dialog.

Figure 128. Debug Configurations for Nios V Processor — Startup Tab



3.3.5.2. Connecting to a Running Nios V Processor Application

You can connect to a running Nios V processor software project on the targeted Altera FPGA using the Visual Studio Code, and continue with the debugging session.

Follow these steps to connect to a running application and debug:

1. Go to **ASHLING PROJECT VIEW ➤ app ➤ Debug Configurations**. Click the **+** icon beside **Debug Configurations** to list down the supported launch configurations. Click **Nios V Hardware Debugging** and specify a name for the launch.

Figure 129. Debug Configurations

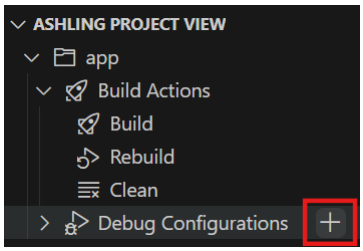
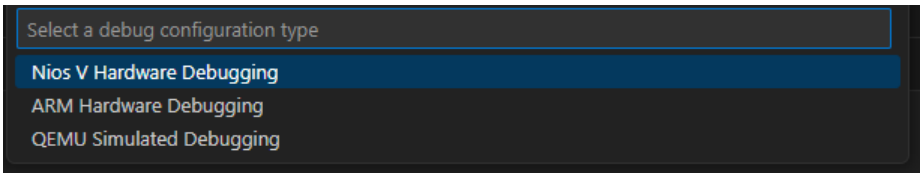
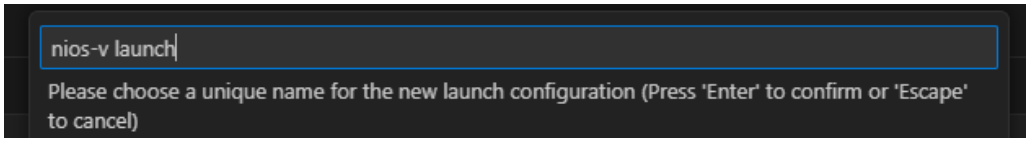


Figure 130. Create Nios V Hardware Debugging



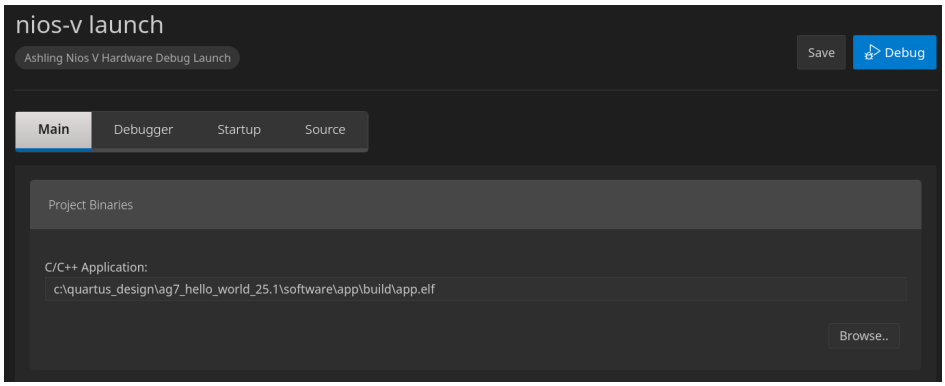
2. Specify a name for the new launch configuration.

Figure 131. New Launch Configuration



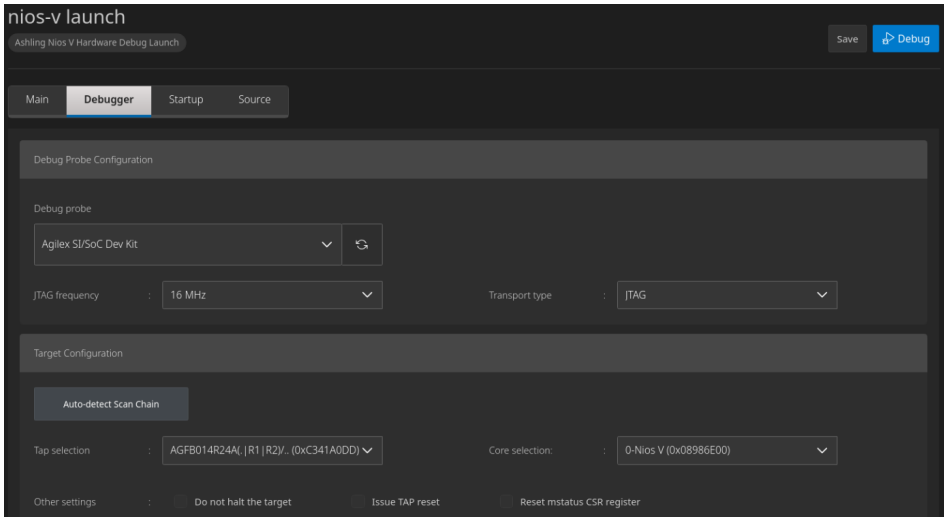
3. A debug configuration window appears. An `.elf` file is auto selected from the project in the **Main** tab. You can select a different `.elf` by using **Browse..** feature.

Figure 132. Debug Configurations for Nios V Processor — Main Tab



4. Click **Debugger** tab, click the drop-down icon to select appropriate debug probe for the debug session.
5. Under the **Debugger** tab, set these settings:
   - Debug probe: **Agilex SI/SoC Dev Kit (Name of the FPGA board/ development kit)**
   - Transport type: **JTAG**.
   - JTAG frequency: **16 MHz**.
6. Click **Auto-detect Scan Chain** to automatically detect JTAG scan chain information of the target device. Make your selection for **Tap selection** and **Core selection**.
7. Turn off **Reset mstatus CSR register**.

Figure 133. Debug Configurations for Nios V Processor — Debugger Tab



8. Turn off **Load Image**, turn off **Set Breakpoint** and turn off **Resume**.
9. Click **Debug** in the top right corner of the launch configuration dialog.

Figure 134. Debug Configurations for Nios V Processor — Startup Tab (Part 1)

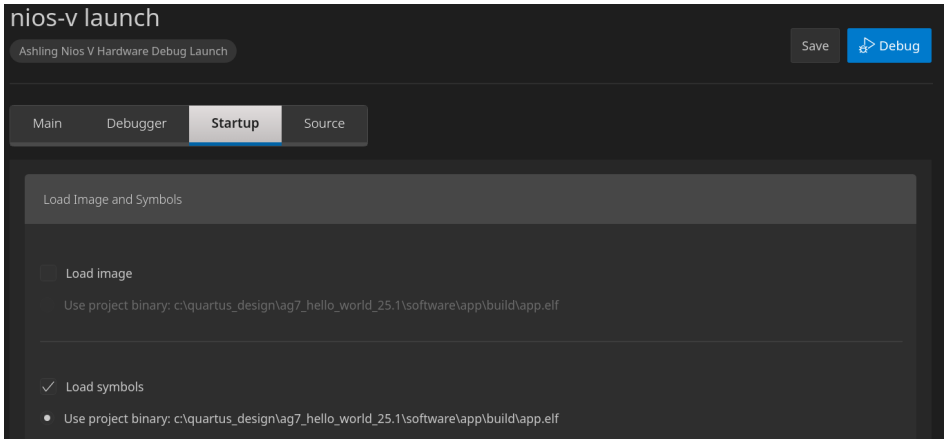
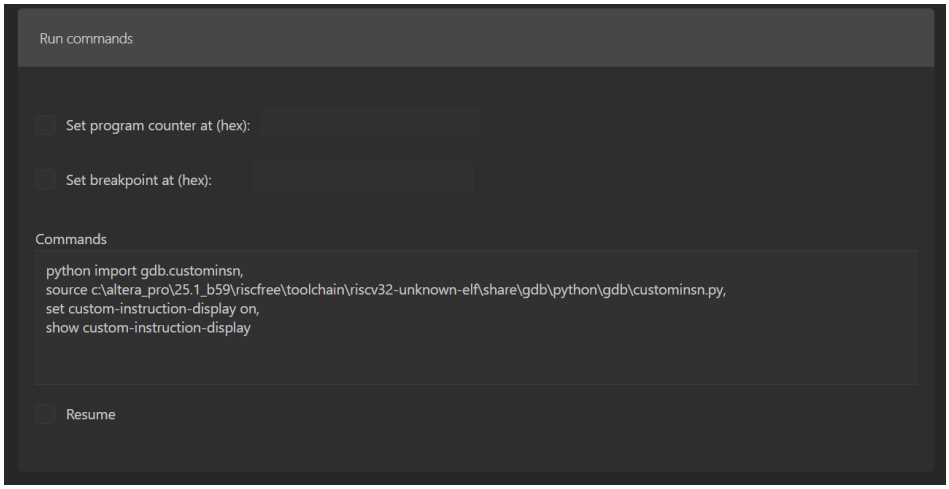


Figure 135. Debug Configurations for Nios V Processor — Startup Tab (Part 2)



3.3.5.3. Considerations when Debugging ROM-based Designs

A ROM-based Nios V processor design refers to the Nios V processor software is developed specifically to be stored in involatile memories, such as ROM or Flash. The ROM-based designs are closely related to the booting implementation in *Nios V Embedded Processor Design Handbook*.

- Nios V Processor Booting from On-Chip Flash (UFM)
- Nios V Processor Booting from General Purpose QSPI Flash
- Nios V Processor Booting from Configuration QSPI Flash
- Nios V Processor Booting from On-Chip Read-Only Memory (OCROM)

Any write attempt by the debugger into a ROM or Flash is not supported, because of the ROM's read-only characteristics and the Flash's write operation complexity respectively.

Thus, debugging a ROM-based designs come with the following limitations:

1. The debugger is unable to download the application ELF into ROM/Flash.
2. The debugger is unable to set software breakpoint, which replace instructions with ebreak. This is affecting the **Set breakpoint at** feature in **Startup** tab, stepping actions in **Debug Bar** and any user-added breakpoints.

Once the application is booting successfully, Altera recommends implementing Connecting to a Running Nios V Processor Application along with setups in Unable to Set Software Breakpoint to connect and debug the application.

3.3.5.3.1. Unable to download (Load Image)

You can program the ROM/Flash using specialized programming methods to load the image into the involatile memories.

Table 17. Programming Methods

Involatile Memories	Programming File	Programming Methods
UFM	POF file	Using Quartus Prime Programmer under Internal Configuration mode.
General Purpose QSPI Flash	POF file	Using Quartus Prime Programmer under Passive Serial or AVST mode with Parallel Flash Loader.
Configuration QSPI Flash	JIC file	Using Quartus Prime Programmer under Active Serial mode.
On-Chip Read-Only Memory	HEX file	Memory initialized during Quartus Prime project compilation.

Related Information

Nios® V Embedded Processor Design Handbook: Nios® V Processor Configuration and Booting Solutions

3.3.5.3.2. Unable to Set Software Breakpoint

The Ashling RiscFree IDE for Altera FPGAs setup breakpoints as software breakpoint by default. This behaviour applies to **Set breakpoint at** feature in **Startup** tab, stepping actions in **Debug Bar** and any user-added breakpoints. You must perform the following setups to switch from software breakpoint into hardware breakpoint.

Table 18. Switch to Hardware Breakpoints

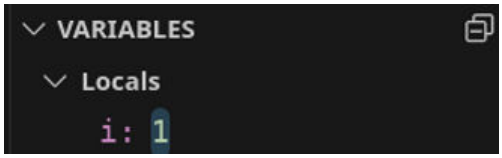
Setups	Effects	Related Breakpoint Features
Enable **Use hardware breakpoint only**. Refer to 3.3.6.5. Hardware Breakpoint.	Ashling RiscFree IDE for Altera FPGAs automatically switched to hardware breakpoint when it identifies any breakpoint.	• **Set breakpoint at** feature in **Startup** tab • Stepping actions in **Debug Bar** • Any breakpoints

3.3.6. Debugging Tools

3.3.6.1. View Variables

To observe the Variables view, go to **Activity Bar ➤ VARIABLES ➤ Locals**.

Figure 136. Variables View



In this view, you can see the variables in your program. When you click on a variable, it displays the number format for that selected variable.

3.3.6.2. Adding Expressions

To observe the Expressions view, go to **Activity Bar ➤ WATCH**.

Figure 137. Expressions View



To add an expression in the view

1. Highlight the expression
2. Right click on the highlighted expression and click **Add to Watch**.

The **WATCH** displays the expression and its value.. When the execution of a program is suspended, all expressions are reevaluated.

3.3.6.3. View Memory

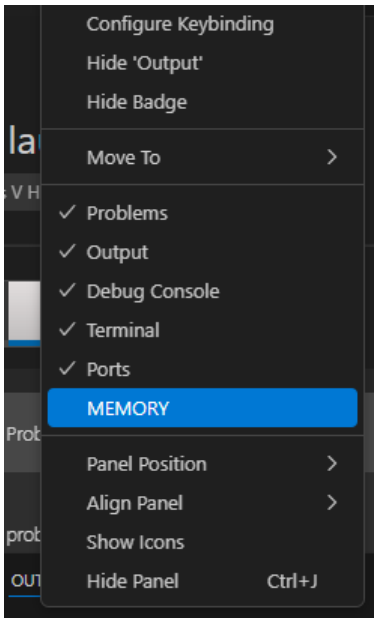
Visual Studio Code supports memory browser. You can view the content of On-Chip Memory (RAM), external memory, or other memory-mapped devices. This example targets the start address of On-Chip Memory (RAM), where an application begins.

Note: You must select little-endian in the Memory View properties because Nios V processor is a little-endian processor.

Follow these steps to launch the memory browser:

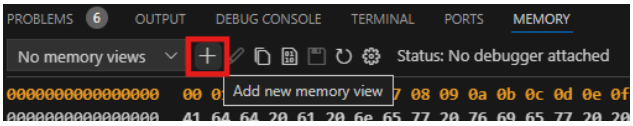
1. Navigate to **OUTPUT** and right click on it.

Figure 138. Enable Memory



2. Click **MEMORY**.
3. Click **Add new memory view** to add memory address.

Figure 139. Add Memory



4. Click **Edit memory view properties**, set these settings:
   - Address: Hex/decimal constant or expression: Leave it as default
   - Display Name: Leave it as default
   - Format: `4-Byte`
   - Endianness: `Little`

Figure 140. Edit Memory View Properties

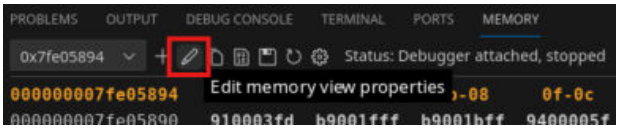


Figure 141. Edit Memory View Properties-Settings

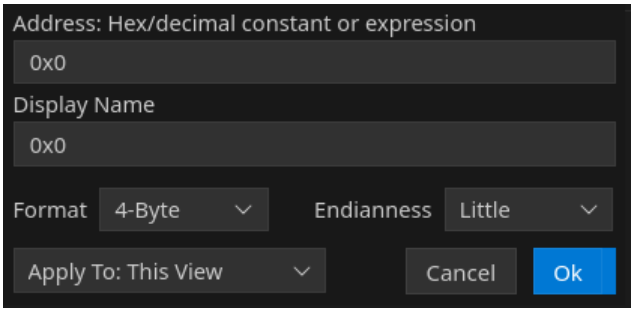
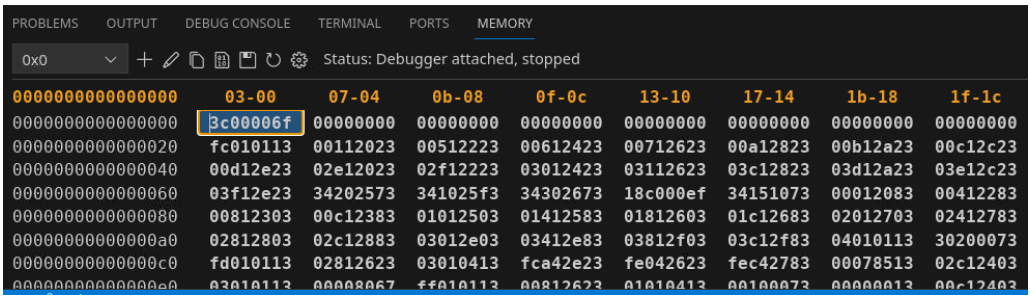
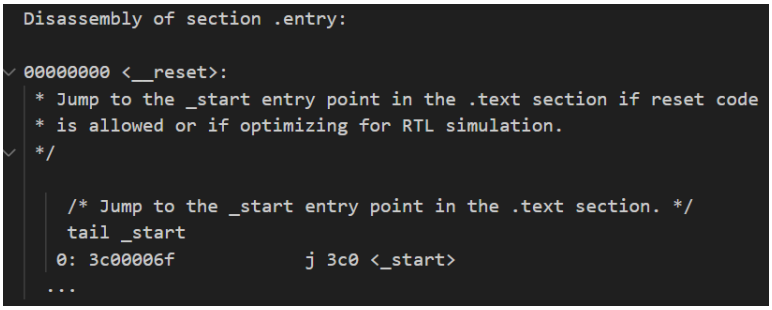


Figure 142. Memory View



5. Go to `<Working directory>/software/app/out/build/` folder.
6. Open the `hello.elf.objdump` file.
7. Search for `Disassembly of section .entry`.
8. The disassembly shows that the information at starting address 0 is `0x3c00006f`, which is exactly the same as in the **Memory Browser**.
9. You can continue to verify `section .exceptions`.

Figure 143. Disassembly of an Application



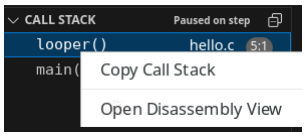
3.3.6.4. Disassembly View with Registers View Stepping into Assembler Functions

Disassembly view allows you to observe your program as it is disassembled. This is helpful when the instruction pointer enters a function for which it does not have the source.

Follow these steps to launch the disassembly view:

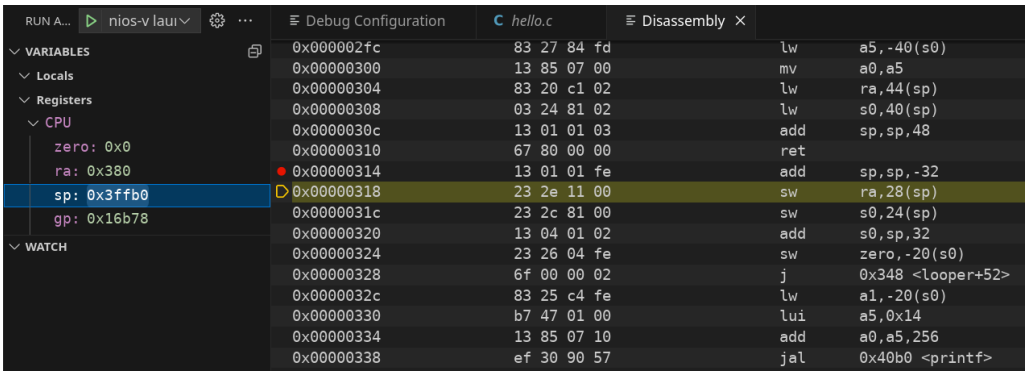
1. Navigate to **Activity Bar ➤ CALL STACK**.
2. Right-click on the item in **CALL STACK** and click **Open Disassembly View**.

Figure 144. Turn On Disassembly View



To debug using the disassembled code in the **Disassembly** view, set a breakpoint at the line you wish to pause. Once the debugger steps into the line that you have set the breakpoint, you can observe the change in the **Registers** view.

Figure 145. Disassembly View with Register View



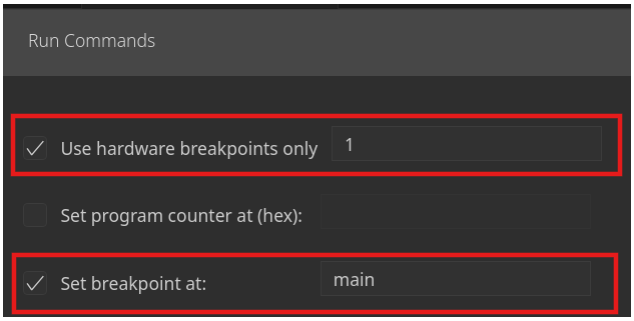
3.3.6.5. Hardware Breakpoint

Follow these steps to use hardware breakpoint:

1. In the **Debug Configuration** window, click **Startup** tab.
2. Under **Run Commands**, enable **Use hardware breakpoints only** and set **1** hardware breakpoint.

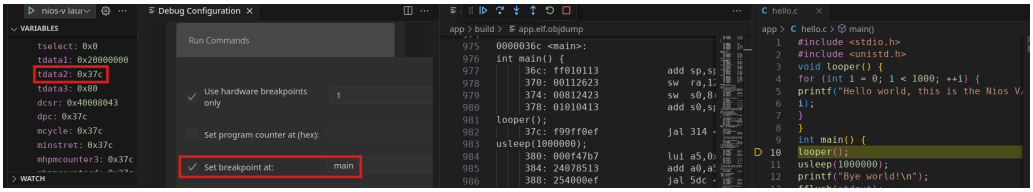
   Note: Nios V processor only supports one trigger.
3. Enable **Set breakpoint at:** and set the location you want to set the breakpoint.

Figure 146. Set Hardware Breakpoint



4. Click **Debug**.
5. To make sure the hardware breakpoint is in the correct location, you can check the value in the `tdata2` register in the **VARIABLES ➤ Registers ➤ CPU**.

Figure 147. Value in `tdata2` Register



This example shows how to use the hardware breakpoint with set the breakpoint. The breakpoint is set at main, which is `0x37c` that shown in `tdata2`. The breakpoint pauses the execution at the first code within the `main`.

3.3.6.6. Configuring the Program Counter Manually

The Ashling Visual Studio Code Extension provides the feature to change the processor's program counter during debugging. Use this feature with caution.

If not done carefully, changing the program counter can lead to inconsistent program states, unexpected behaviour, or crashes. Altera recommends this option for you if you have advanced knowledge and a thorough understanding of the program's logic and state.

1. Navigate to **Startup** tab. Click **Run Commands ➤ Set program counter at (hex)**.
2. Provide any instruction memory address in HEX.
3. You may refer to the `objdump` file to locate the interested instruction memory address.
4. Start debugging, and the processor jumps to the specified instruction memory address.

Figure 148. Set Program Counter at (hex) in Startup Tab



3.4. Using Ashling Visual Studio Code Extension with Arm Hard Processor System

The example in this tutorial is based on the Intel FPGA example design from Ashling.

Note: Visual Studio Code supports Arm HPS standalone debug.

3.4.1. Pre-requisite for Debugging Arm HPS Project

This example uses the Agilex 7 board. To prepare the platform you are planning to debug, follow these steps:

1. Prepare a golden system reference design (GSRD) HPS `.sof` file or use a prebuilt HPS `.sof` file.
2. Download the HPS `.sof` file to the Agilex 7 board using the Quartus Prime Programmer and Tools.

Related Information

- Agilex 7 SoC Golden System Reference Design (GSRD)
  Provides the prebuilt .sof files at the Release Contents section for download.
- Cyclone V SoC Golden System Reference Design (GSRD)
  Provides the prebuilt .sof files at the Release Contents section for download.
- Stratix 10 SoC Golden System Reference Design (GSRD)
  Provides the prebuilt .sof files at the Release Contents section for download.
- Arria 10 SoC Golden System Reference Design (GSRD)

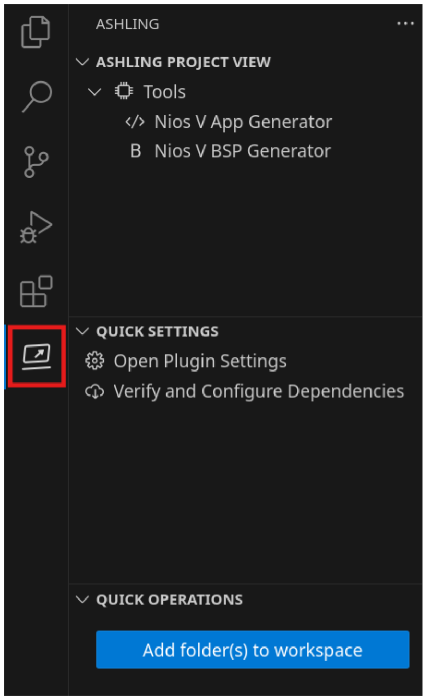
3.4.2. Importing Arm HPS Project

Note: If you want to test the pre-built `.elf` file, refer to the link provided in the Related Information section. This extension does not provide the pre-built `.elf` file.

Follow these steps to import the Agilex 7 Arm Cortex*-A53 sample project and a pre-built `.elf` file:

1. Click the **Ashling** icon to open the **Ashling Sidebar**.

Figure 149. Ashling Sidebar



2. Under **QUICK OPERATIONS**, click **Add Folder(s) to workspace** and browse to the folder.

Related Information

Importing Arm* HPS Project

3.4.3. Setting Debug Configurations and Downloading Arm HPS Project Using Visual Studio Code

Follow these steps to debug the project:

1. Go to **ASHLING PROJECT VIEW ➤ agilex7-a53-picojpeg ➤ Debug Configurations**. Click the **+** icon beside **Debug Configurations** to list down the supported launch configurations. Click **ARM Hardware Debugging** and specify a name for the launch.

Figure 150. Debug Configurations

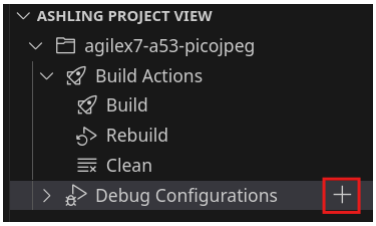
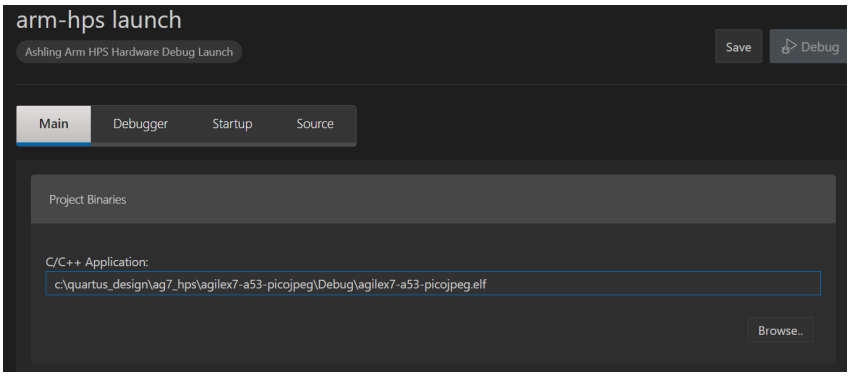


Figure 151. Create ARM Hardware Debugging



2. A debug configuration window appears, `.elf` file is auto selected from the project in the **Main** tab. You can select a different `.elf` by using **Browse** feature.

Figure 152. Debug Configurations for ARM Processor—Main Tab



3. Click **Debugger** tab, and click the drop-down icon to select the appropriate debug probe for the debug session.
4. Under the **Debugger** tab, set these settings:
   - Debug probe: **Agilex SI/SoC Dev Kit** (Name of the FPGA board/development kit)
   - Transport type: **JTAG**
   - JTAG frequency: **16 MHz**
   - Core selection: Can select any core between 0 to 3
5. Click **Auto-detect Scan Chain** to automatically detect JTAG scan chain information of the target device. Select the options from **Tap selection** and **Core selection**.

Figure 153. Debug Configurations for ARM Processor — Debugger Tab (Part 1)

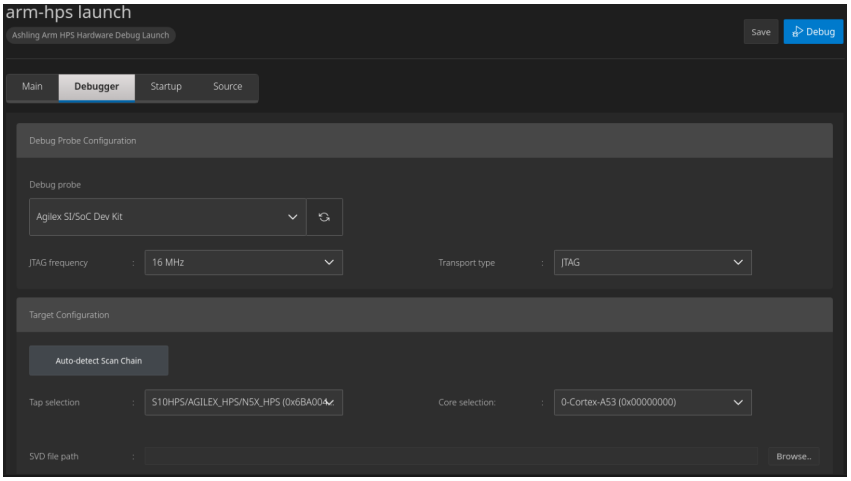
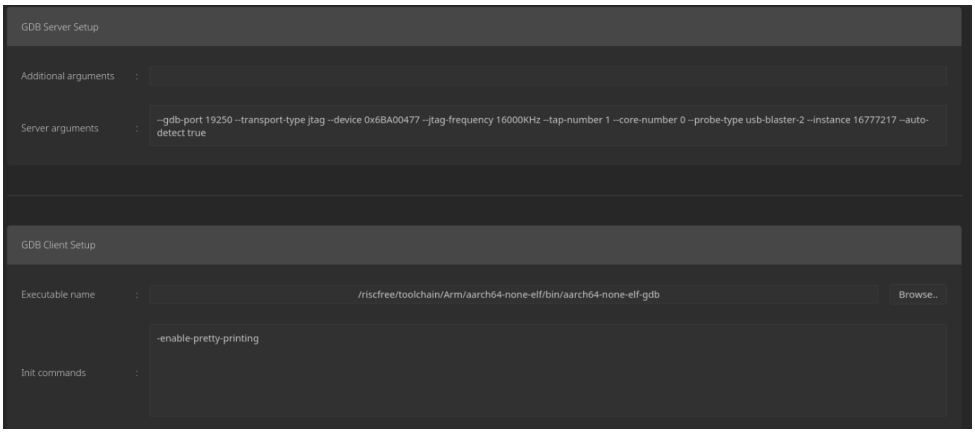


Figure 154. Debug Configurations for ARM Processor — Debugger Tab (Part 2)



Note: The default settings for the current example design in the Visual Studio Code are configured based on the Agilex 7 device.

Note: If you are using a different Arm architecture, select the Arm GNU debugger (GDB) (select **GDB Client Setup ➤ Executable name**) manually.

- 64-bit ELFs: `${eclipse_home}/../toolchain/Arm/aarch64-none-elf/bin/aarch64-none-elf-gdb.exe`
- 32-bit ELFs: `${eclipse_home}/../toolchain/Arm/arm-noneeabi/ bin/arm-none-eabi-gdb.exe`

6. Leave all other settings, including those in the **Startup** tab, as sensible defaults have been set.
7. Click **Debug** in the top right corner of the launch configuration dialog.
8. Visual Studio Code downloads the program to the target SoC and display its progress through the console prints.

Figure 155. Debug Configurations for ARM Processor — Startup Tab and Debug Button

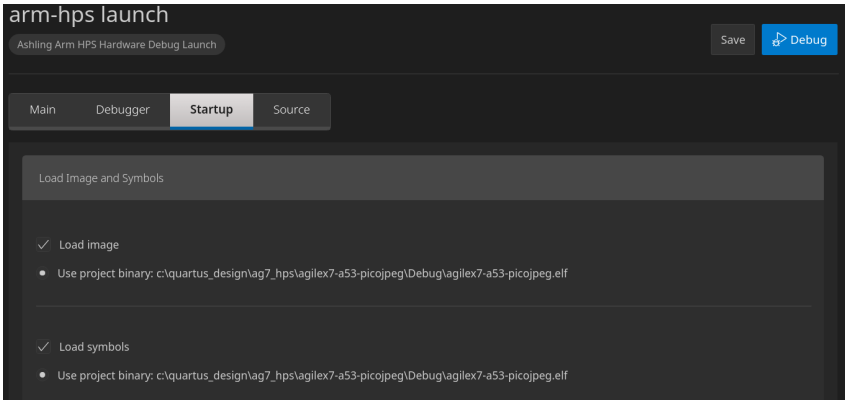
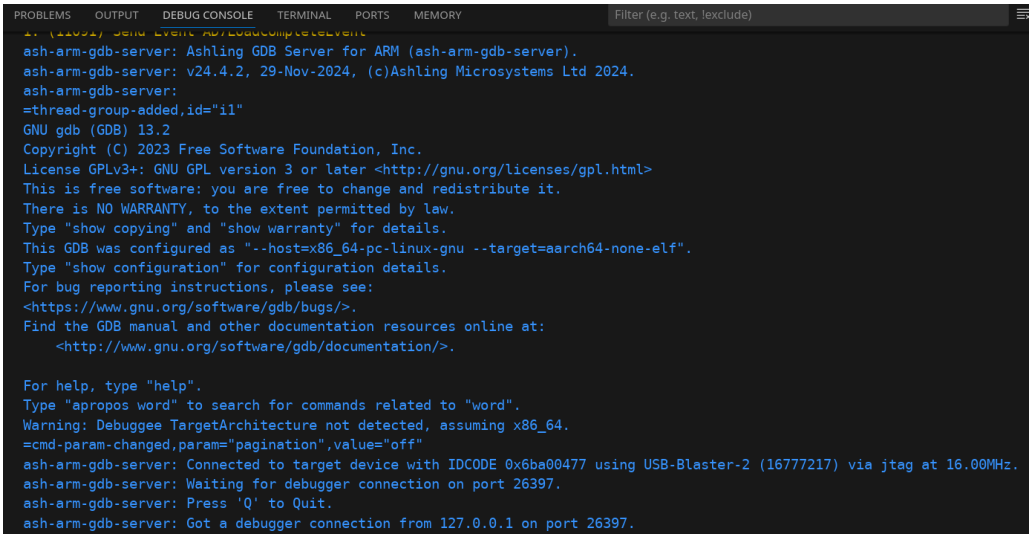


Figure 156. Visual Studio Code after Program is Downloaded and System is Ready for Debug



3.5. Debugging Features in Ashling Visual Studio Code Extension

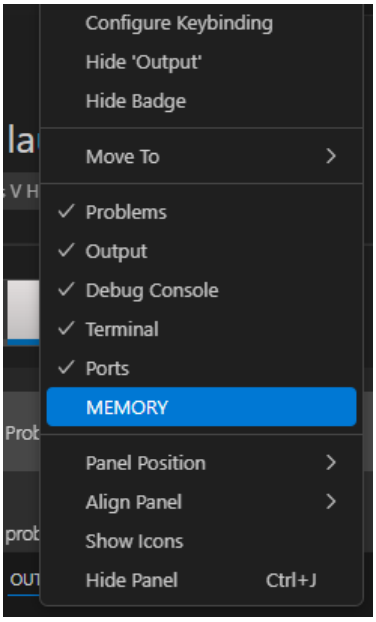
3.5.1. Debug Features in Ashling Visual Studio Code Extension

3.5.1.1. Launching Memory Browser

Visual Studio Code supports memory view. Follow these steps to launch the memory browser:

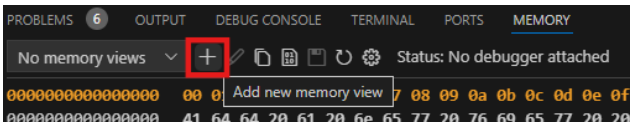
1. Navigate to **OUTPUT** and right-click on it.

Figure 157. Enable Memory



2. Click **MEMORY**.
3. Click **Add new memory view** to add memory address.

Figure 158. Add Memory



4. Click **Edit memory view properties**, set these settings:

- Address: Hex/decimal constant or expression: Leave it default
- Display Name: Leave it default
- Format: **4-Byte**
- Endianness: **Little**

Figure 159. Edit Memory View Properties

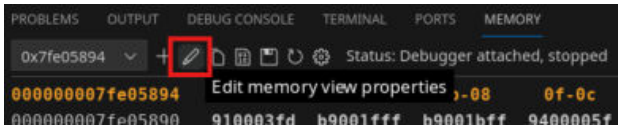


Figure 160. Edit Memory View Properties- Settings

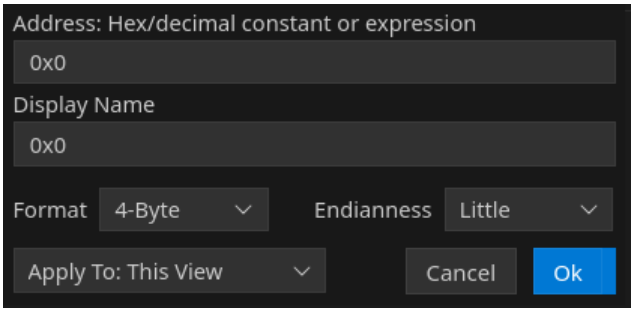
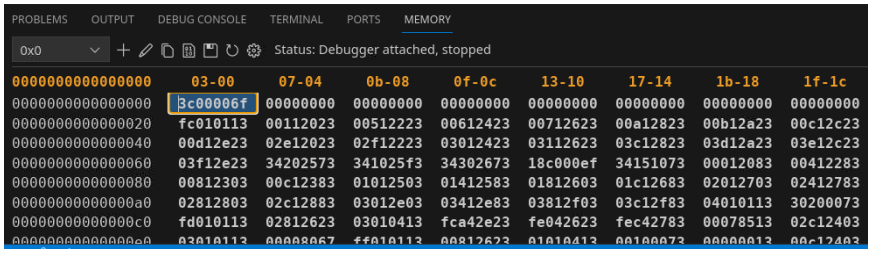


Figure 161. Memory View



3.5.1.2. Debugger Console

Visual Studio Code supports debugger console.

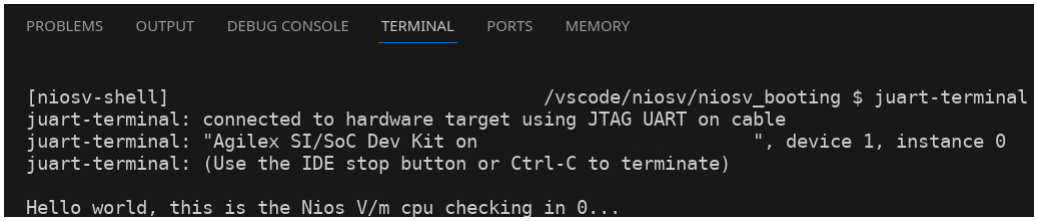
You can launch the debugger console from **View ➤ Debugger Console**.

You can enter debug commands and view output in the debug console. The GNU GDB syntax is fully supported in the console.

3.5.1.3. JTAG UART Output Terminal

You can view the output of the JTAG UART (juart) terminal in the Visual Studio Code via launching `juart-terminal` in the **Terminal** window.

Figure 162. JTAG UART Terminal Console Print



3.5.1.4. Debugging RTOS

Note: µC/OS-II RTOS is not supported.

FreeRTOS

When you debug the FreeRTOS application, you can view the number of tasks, task details, and queue list. To display details of FreeRTOS Task List, enable the macro as below in the `FreeRTOSConfig.h`.

Note: If you use the FreeRTOS BSP generated from BSP Editor, you do not need to manually configure `configUSE_TRACE_FACILITY` in `FreeRTOSConfig.h`. The setting depends on `system.h` (part of `FreeRTOSConfig.h #include`), which can be configured using BSP Editor GUI or `niosv-bsp` command

Table 19. FreeRTOS Debug Configuration

Debug Information	Macros	Configuration	Description
Task List	`configUSE_TRACE_FACILITY`	1	To display task number of the project

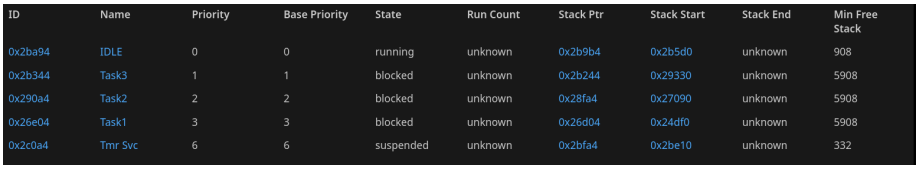
The configuration is as shown below:

```
#define configUSE_TRACE_FACILITY 1
```

To debug FreeRTOS application, follow these steps:

- Launch the FreeRTOS application debug. The RTOS related view is automatically appeared in **Panel**.

Figure 163. RTOS Related View



Zephyr

When you debug the Nios V processor application, you can view the Zephyr Mutex (Mutual Exclusion Object) List, Zephyr Semaphore List, and Zephyr Thread List.

To display details of Zephyr Mutex, Semaphore and Thread lists, enable the macros as below in the `prj.conf` file.

Debug Information	Macros	Configuration	Description
Enable Trace	CONFIG_TRACING	Y	To enable all the tracing function. *Note:* All tracing macros have dependency to CONFIG_TRACING.
Mutex List	CONFIG_TRACING_OBJECT_TRACKING	Y	To enable object tracking for mutex.
Semaphore List			To enable object tracking for semaphore.
Thread List	CONFIG_DEBUG_THREAD_INFO	Y	To display thread details.

The configuration is as shown below:

```
CONFIG_TRACING=y
CONFIG_TRACING_OBJECT_TRACKING=y
CONFIG_DEBUG_THREAD_INFO=y
```

Follow these steps to debug the Zephyr application:

- Launch the Zephyr application debug. The RTOS-related view automatically appears in the **Panel**.

ID	Name	Priority	Base Priority	State	Run Count	Stack Ptr	Stack Start	Stack End	Min Free Stack
0x2ba94	IDLE	0	0	running	unknown	0x2b9b4	0x2b5d0	unknown	908
0x2b344	Task3	1	1	blocked	unknown	0x2b244	0x29330	unknown	5908
0x290a4	Task2	2	2	blocked	unknown	0x28fa4	0x27090	unknown	5908
0x26e04	Task1	3	3	blocked	unknown	0x26d04	0x24df0	unknown	5908
0x2c0a4	Tmr Svc	6	6	suspended	unknown	0x2bfa4	0x2be10	unknown	332

3.5.2. Nios V Processor System Debug

This section is a generic debugging section where the debugging steps are the same for Nios V processor and Arm HPS debugging. Use the guidelines in this section to start debugging the program with the GUI or GDB console after completing the following steps:

- Debugging a Nios V Processor Project
- Setting Debug Configurations and Downloading Arm HPS Project Using Visual Studio Code on page 103

3.5.2.1. GUI for Debugging

You can control the debugging process using the Debug Toolbar on the top of the window.

Figure 164. Execution Control in the Debug Toolbar

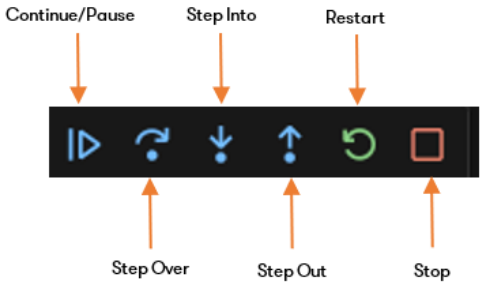


Table 20. Execution Control Using the Debug Toolbar

Function	Description
Continue / Pause	**Continue**: Resume normal program/script execution (up to the next breakpoint). **Pause**: Inspect code executing at the current line and debug line-by-line.
Step Over	Execute the next method as a single command without inspecting or following its component steps.
Step Into	Enter the next method to follow its execution line-by-line.
Step Out	When inside a method or subroutine, return to the earlier execution context by completing remaining lines of the current method as though it were a single command.
Restart	Terminate the current program execution and start debugging again using the current run configuration.
Stop	Terminate the current program execution.

3.5.3. Heterogeneous Multicore Debug

Heterogeneous multicore debugging is a method where you can debug two or more cores that differ in architecture (such as Nios V multicore or Nios V processor core with Arm HPS core) simultaneously in the same integrated development environment.

By using heterogeneous multicore debugging, you can do the following:

- Simultaneously debug the entire integrated system in a single IDE.
- Real-time control of multiple processor cores that interact with each other.

3.5.3.1. Importing Projects

To import the Nios V processor and Arm HPS projects to the Visual Studio Code workspace, follow these steps:

- Follow the steps in the Importing Nios V Processor Project and Importing Arm HPS Project sections.
- After you import the projects, you can find the project directory in the **Explorer**.

Figure 165. Explorer View (Nios V Multicore)

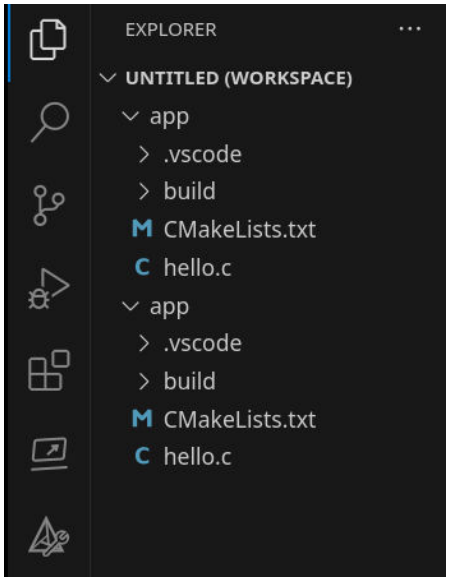
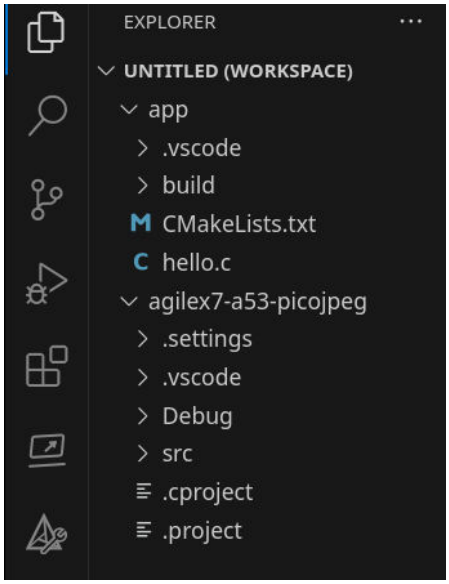


Figure 166. Explorer view (Nios V Processor and ARM HPS Core)



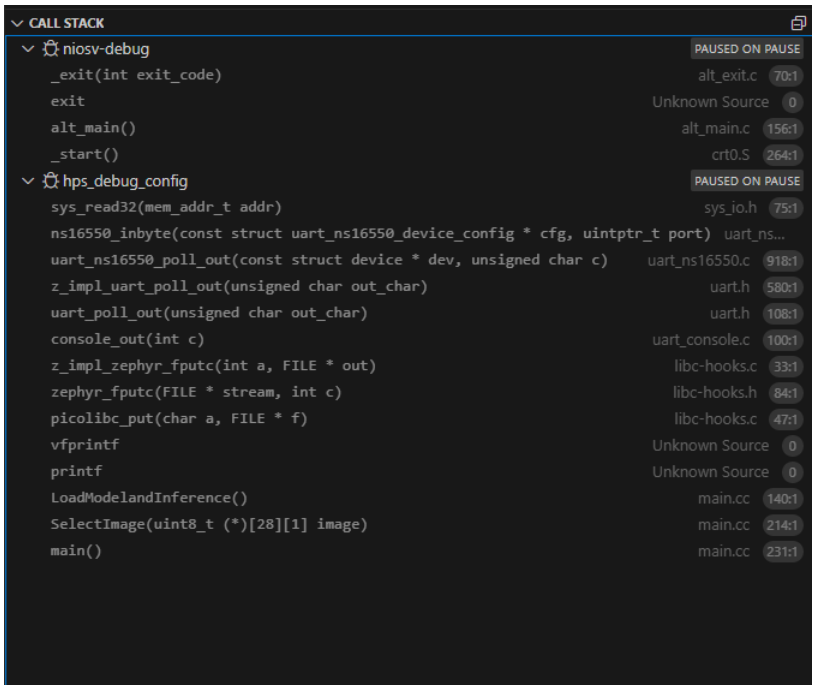
3.5.3.2. Debugging Multicore Projects (Multiple Nios V Processor cores)

Follow these steps to set the core configuration:

1. Follow the steps in Using Ashling Visual Studio Code Extension with Nios V Processor System for both Nios V processors.
2. The **CALL STACK** section, all cores selected for debug are launched.
3. To debug two cores simultaneously, go to **Activity Bar ➤ VARIABLES**.

Note: Visual Studio Code only display the debug information of a single core at any given moment. It automatically switch between the cores based on your selection.

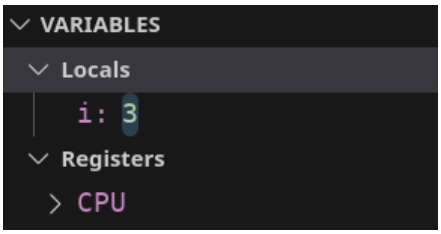
Figure 167. Heterogeneous Multicore Debug View in CALL STACK



4. To debug two cores simultaneously, go to **Activity Bar ➤ VARIABLES**.

   Note: Visual Studio Code only displays the debug information of a single core at any given moment. It automatically switches between the cores based on your selection.

Figure 168. Register Window



3.5.3.3. Debugging Multicore project (Nios V processor core and ARM HPS)

Ensure you have completed the steps in Using Ashling Visual Studio Code Extension with Nios V Processor System and Setting Debug Configurations and Downloading Arm HPS Project Using Visual Studio Code. Then, follow these steps to set the core configuration:

1. In the **CALL STACK** section, all cores selected for debugging are launched
2. To debug two cores simultaneously, go to **Activity Bar ➤ VARIABLES**

Note: Visual Studio Code only displays the debug information of a single core at any given moment. It automatically switches between the cores based on your selection.

4. Ashling RiscFree Integrated Development Environment (IDE) for Altera FPGAs User Guide Archives

For the latest and previous versions of this user guide, refer to Ashling RiscFree Integrated Development Environment (IDE) for Altera FPGAs User Guide. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

5. Document Revision History for the Ashling RiscFree Integrated Development Environment (IDE) for Altera FPGAs User Guide

Document Version	Quartus Prime Version	Changes
2025.04.11	25.1	• Changed the title and related mentions of *Ashling RiscFree Integrated Development Environment (IDE) for Intel FPGAs* to *Ashling RiscFree Integrated Development Environment (IDE) for Altera FPGAs User Guide*. • Revamped chapter *Ashling RiscFree IDE for Altera FPGAs.* • Added new chapter *Ashling Visual Studio Code Extension for Altera FPGAs*. • Moved the topic *Debugging with Command-Line Interface* to the *Appendix*.
2025.01.27	24.3.1	• Updated Table: *RiscFree IDE Open-Source Components* in *Open-Source Components in RiscFree IDE*.
2024.11.25	24.3	• Updated Table: *RiscFree IDE Open-Source Components* in *Open-Source Components in RiscFree IDE*.
2024.07.26	24.2	• Added Agilex 5 in the topic *Supported Devices*. • Added new chapter *Getting started with the Ashling RiscFree IDE for Intel FPGAs*. Moved the chapter *Installation and Setup* to be part of the new chapter. • Retitled *Debug Setup for Nios V Processor System* to *Using Ashling RiscFree IDE for Intel FPGAs with Nios V Processor System* and added the following subtopics: — *Setting Run Configuration to Download Nios V Processor Project* — *Setting Debug Configuration to Debug Nios V Processor Project* — *Setting Debug Configuration to Debug a Booting Nios V Processor Project* — *Debugging Tools* • Retitled *Debug Setup for Arm Hard Processor System* to *Using Ashling RiscFree IDE for Intel FPGAs with Arm Hard Processor System*. • Updated Appendix *Performing External Tools Configuration*. • Updated Table: *RiscFree IDE Open-Source Components*.
2024.05.13	24.1	• Added a note in topic *Graphical User Interface*. • Updated topic *Importing Nios V Processor Project*. • Updated Table: *Options and Description*. • Updated Table: *RiscFree IDE Open-Source Components*.
2023.12.14	23.4	• Updated Table: *RiscFree IDE Open-Source Components* in *Open-Source Components in RiscFree IDE*.
2023.12.01	23.3	Added new topics: • *Intel Simics Simulator Support*. • *Ashling RiscFree IDE for Altera FPGAs Interaction with Intel Simics Simulator for FPGAs*. • *Debugging the Linux Kernel*.

continued...

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Document Version	Quartus Prime Version	Changes
		• *Debugging in Intel Simics.* • *Debugging with Command-Line Interface*. • Updated Table: *RiscFree IDE Open-Source Components* in *Open-Source Components in RiscFree IDE*.
2023.07.20	23.2	• Added a link to *AN 980: Nios® V Processor Intel® Quartus® Prime Software Support* in the topic *Intel® Quartus® Prime Software Support*. • Updated Table: *RiscFree IDE Open-Source Components* in *Open-Source Components in RiscFree IDE*.
2023.04.10	23.1	• Edited topic *Debug Setup for Nios V Processor System* to add the mention of Zephyr and FreeRTOS. • Updated the following figures: — *Debug Configurations for Nios V Processor —Debugger Tab* — *Debugger Settings for Intel Agilex® 7 Arm Cortex-A53 Core* — *List of Available Cores for the Selected Target Device in RiscFree IDE* — *Startup Tab Settings* — *Enabling OS Aware Debugging in Heterogeneous Debugging* • Added new topics: — *Debugging FreeRTOS Application*. — *Debugging Zephyr Application*. • Added Ashling RiscFree IDE for Intel FPGAs and OpenCSD to the table *RiscFree IDE Open-Source Components*. • Updated product family name to Intel Agilex 7.
2022.12.19	22.4	• Added new topics in *Debugging with RiscFree IDE*. — *Arm HPS On-Chip Trace* — *Trace Toolbar* — *Trace View* • Updated Table: *RiscFree IDE Open-Source Components* in *Open-Source Components in RiscFree IDE*.
2022.10.31	22.1std	• Added more devices to the *Supported Devices* section. • Added links to the documentations supporting *Quartus Prime Standard Edition* in *Supported Devices* section. • Updated references from *Intel Quartus Prime Pro Edition* to *Intel Quartus Prime* to indicate support for both Pro and Standard Edition. • Added steps to download Ashling RiscFree IDE for Intel FPGAs in the *Standalone Installation* section. • Added a figure *Installation Warning Message* in *Standalone Installation with Supported Intel Quartus Prime Software* section.
2022.09.26	22.3	• Updated the document title from *RiscFree Integrated Development Environment (IDE) for Intel FPGAs User Guide* to *Ashling RiscFree Integrated Development Environment (IDE) for Intel FPGAs User Guide*. • Updated the features in the *About the RiscFree IDE* section. • Added a note in the *Standalone Installation with the Quartus Prime Pro Edition Programmer and Tools* session. • Updated the RiscFree IDE launching method and added a note to avoid the `niosv-stack-report` warning for standalone installation in the *Command Line Interface* section. • Added information on Nios V applications OS support in the *Debug Setup for Nios V Processor System* section. • Update the *Setting Debug Configurations and Downloading Nios V Project Using RiscFree IDE* section. — Updated the *Debug Configurations for Nios V—Main Tab* diagram. — Added step on OS Awareness settings.

continued...

Document Version	Quartus Prime Version	Changes
		• Added note to state that RiscFree IDE supports Arm HPS standalone debug in the *Debug Setup for Arm Hard Processor System* section. • Updated the *Setting Core Configuration* section. — Updated the *Target Application Tab Settings* diagram. — Added step on OS Awareness settings. • Added the following sections: — *Debugging µC/OS-II Application* — *Help Content* — *Open-Source Components in RiscFree IDE*
2022.06.21	22.2	Initial release.

A. Appendix

A.1. Optional Toolchain Configuration in Ashling RiscFree IDE for Altera FPGAs

A.1.1. Toolchain for Application and BSP

The `risc32-unknown-elf` toolchain is included in RiscFree IDE by default. Toolchain configuration is not required to build the software for current Nios V processor system.

To build an application or BSP with a toolchain other than `risc32-unknown-elf`, perform these configuration steps to include the other toolchain to the build path:

Note: If you configure the BSP setting file to use the RiscFree IDE toolchain (`risc32-unknown-elf`), do not perform this configuration.

Follow the steps below to include the toolchain in path of the RiscFree IDE builder:

1. Download the open source GNU RISC-V Embedded GCC package version v10.2.0-1.2 or later that supports the necessary RISC-V Instruction-Set Architecture (ISA).
2. Extract the package in `<Intel Quartus Prime installation directory>/niosv/` directory. Use the same directory for Linux and Windows.
3. Right click the project under the **Project Explorer** section and click **Properties**.
4. Go to **C/C++ Build ➤ Environment**. Double click **PATH variable** to add the toolchain path. Example on adding `xpack-riscv-none-embed-gcc` toolchain:
   - Linux (add ":" as a separator for multiple path variables):

     `<Intel Quartus Prime installation directory>/niosv/xpack-riscv-none-embed-gcc-10.2.0-1.2/bin`
   - Windows (add ";" as a separator for multiple path variables):

     `<Intel Quartus Prime installation directory>/niosv/xpack-riscv-none-embed-gcc-10.2.0-1.2/bin`
5. Click **OK** and **Apply & Close**.

A.1.2. Toolchain for Arm Processor

Only the Nios V toolchain is included in RiscFree IDE. The CDT plugins from Eclipse support cross-building for Arm.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

To use CDT plugins for Arm using your own toolchain, follow these steps:

1. Go to **File ➤ New ➤ C/C++ Project** and click **Next**.
2. Select **C Managed Build** and click **Next**.
3. Select project type. The project type can be completely managed by CDT, CMake, or make-based.
4. Enter Project Name as **Test Bare-Metal** or your preferred name. Select Project type as **Executable ➤ Hello World C++ Project**, and Toolchain as **Cross GCC**. Click **Next**.
5. Enter Author, Copyright notice, and other details. Click **Next**. This step is optional.
6. Ensure **Debug** and **Release** options are enabled. Click **Next**.
7. Enter the cross GCC details as follows:

Devices	Arria 10 and Cyclone V (Non-SDM Devices)	Stratix 10 and Agilex 7 (SDM Devices)
Cross compiler prefix	`arm-none-linux-gnueabihf-`	`aarch64-none-linux-gnu-`
Cross compiler path	`<path_to_toolchain_directory>/gcc-arm-11.2-2022.02-x86_64-arm-none-linux-gnueabihf/bin/`	`<path_to_toolchain_directory>/gccarm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin/`

Note: The toolchain varies for SDM and non-SDM devices. For details on the toolchain, refer to GSRD of the device you use.

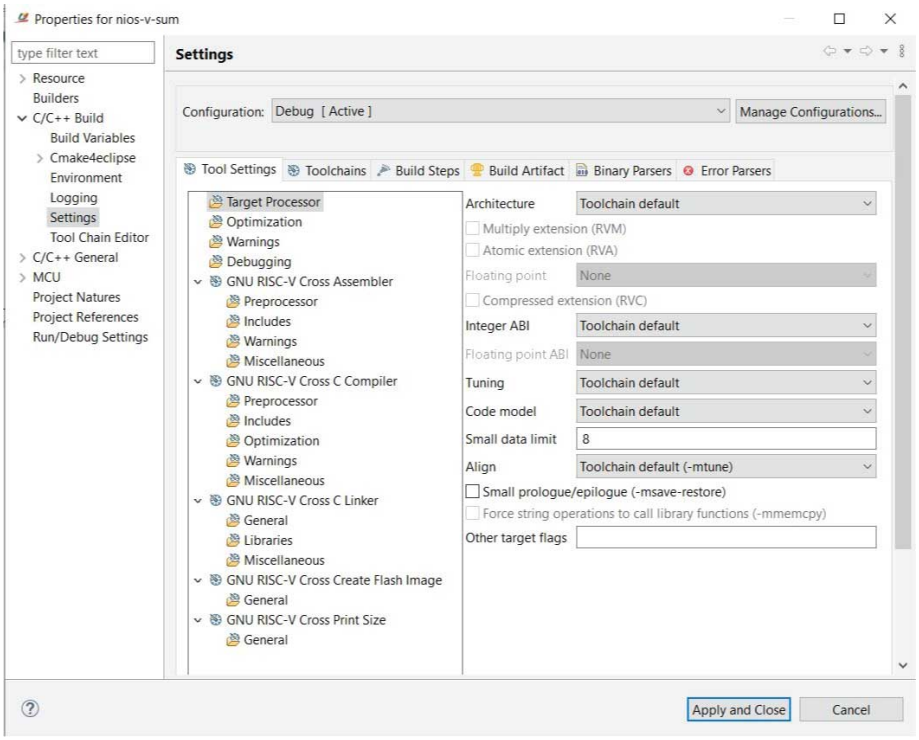
8. Click **Finish**. The project is created.

A.1.3. Modifying Toolchain Configuration

To change the toolchain options, follow these steps:

1. Right click project files (application or BSP) and select **Properties**.
2. Click **C/C++ Build ➤ Settings**.

Figure 169. Modifying Toolchain Configuration



A.2. Performing External Tools Configuration in Ashling RiscFree IDE for Altera FPGAs

External tool configuration is a generic feature where you can configure the RiscFree IDE to include tools based on your requirement.

To perform external tool configuration for juart terminal, follow these steps:

1. Go to **Run ➤ External Tools ➤ External Tools Configurations...**. Double click **Program** to open a **New_configuration** window.
2. Rename the configuration as **Nios V JTAG UART Output**.
3. Browse and select the `juart-terminal` file in the following paths:

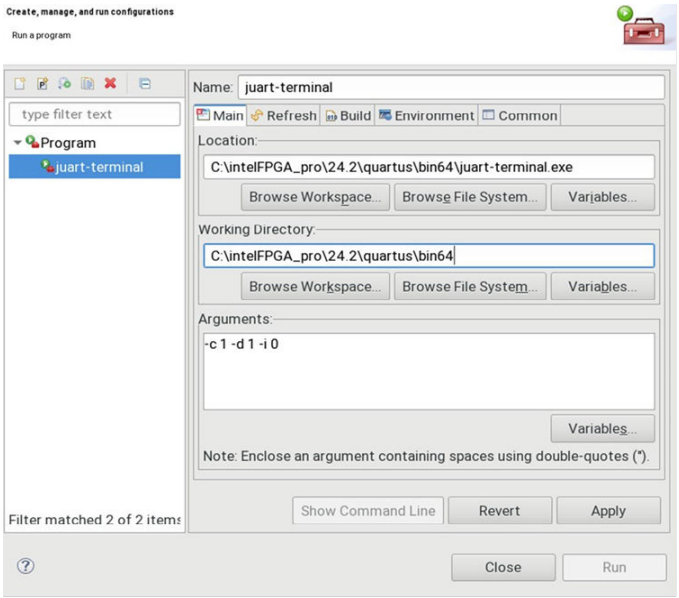
Operating System		Path
Windows	Quartus Prime Software	*<Intel Quartus Prime installation directory>*/quartus/bin64/juart-terminal.exe
	Quartus Prime Programmer and Tools	*<Intel Quartus Prime installation directory>*/qprogrammer/quartus/bin64/juart-terminal.exe
Linux	Quartus Prime Software	*<Intel Quartus Prime installation directory>*/quartus/linux64/juart-terminal
	Quartus Prime Programmer and Tools	*<Intel Quartus Prime installation directory>*/qprogrammer/quartus/linux64/juart-terminal

4. Set the preferred arguments. In this example, set the arguments as follows:

- Connect to JTAG UART at cable 1: **-c 1**
- Connect to device 0: **-d 1**
- Connect to instance 0: **-i 0**

Note: JTAG device chain index varies according to your setup. Run `jtagconfig --debug` to obtain the latest device chain index numbering.

Figure 170. External Tools Configuration



Note: The steps are also applicable to other Nios V tools (`niosv-app` and `niosv-bsp`) under *<Intel Quartus Prime installation directory>*`/niosv/bin` directory. These tools are not available for standalone RiscFree IDE installation with the Quartus Prime Programmer and Tools.

A.3. Debugging with Command-Line Interface

You can perform core debugging on the Nios V processor or the Arm HPS core using Ashling GDBServer and GDB (GNU debugger) with a command-line interface. You need to invoke the compatible executables based on the core architecture.

A.3.1. Running Ashling GBD Server

The Ashling GDBServer connects to the processor core with a remote GDB, which allows you to debug the software applications remotely. The Ashling GDBServer must be ready before running the GDB.

Table 21. Ashling GDBServer

Processor Core	Executables File
Nios V Processor	`<Intel Quartus Prime installation directory>/riscfree/debugger/gdbserver-riscv/ash-riscv-gdb-server.exe`
Arm HPS core	`<Intel Quartus Prime installation directory>/riscfree/debugger/gdbserver-arm/ash-arm-gdb-server.exe`

By invoking the Ashling GDBServer with `--help`, you can get more information from the executables' internal documentation about the supported GDB MONITOR commands. The table below lists the common options for starting the Ashling GDBServer.

Table 22. Options and Description

Options	Mandatory	Description
`--device <IDCODE>`	Yes	Specifies the IDCODE of the target device. Use `jtagconfig` to display the device IDCODE.
`--autodetect true`	Yes	Enable JTAG scan chain auto-detection on the specified IDCODE.
`--probe-type usb-blaster-2`	Yes	Select Intel FPGA Download Cable II (USB-Blaster 2) as the debug probe.
`--tap-number <number>`	No	Select the TAP (test access port) with the specified IDCODE (when there is more than one TAP with the same IDCODE).
`--core-number <number>`	No	Select the processor core within the same device. • Use `jtagconfig` to display the Nios V processor core number. • Refer to the *Hard Processor System Technical Reference Manual* for the number of Arm cores in the device.
`--jtag-frequency <frequency>`	No	Specifies the JTAG frequency. The following are the valid inputs: • 1 MHz, 2 MHz, 3 MHz, 4 Mhz, 4.8 Mhz, 6 MHz, 16 MHz, 24 MHz • `auto-adjust` (to specify the same frequency as jtagserver)
`--list-probes`	No	List out all the serial number for the connected debug probes.
`--instance <number>`	No	Select the probe based on the serial number (persistent ID). Apply Ashling GDBServer with `--list-probes` to determine the serial number.
`--gdb-port <port>`	No	Specifies the TCP port to connect to GDB. Default TCP at 2331.

Related Information

- Agilex 7 Hard Processor System Technical Reference Manual
- Stratix 10 Hard Processor System Technical Reference Manual
- Arria 10 Hard Processor System Technical Reference Manual
- Cyclone V Hard Processor System Technical Reference Manual

A.3.2. Running GDB

Once the Ashling GDBServer is ready, the GDB can communicate with the Ashling GDBServer via a TCP connection using GDB Remote Serial Protocol. Start the GDB with the application ELF file as an argument. To establish a TCP connection, use the `target remote:<port>` command in the GDB.

Table 23. GDB in Ashling RiscFree IDE for Altera FPGAs

Processor Core	Executables File
Nios V Processor	`<Intel Quartus Prime installation directory>/riscfree/toolchain/riscv32-unknown-elf/bin/riscv32-unknown-elf-gdb.exe`

Arm HPS core	64-bit ELFs:
	`<Intel Quartus Prime installation directory>/riscfree/toolchain/Arm/aarch-none-elf/bin/aarch64-none-elf-gdb.exe`
	32-bit ELFs:
	`<Intel Quartus Prime installation directory>/riscfree/toolchain/Arm/arm-none-eabi/bin/arm-none-eabi-gdb.exe`

A.3.3. Nios V Processor Example

The following is the example of running Ashling GDBServer at port 54321 on Agilex 7 FPGA - Hello World Design on Nios V/m Processor example design.

```
$ jtagconfig.exe -d
1) AGF FPGA Development Kit on 10.107.209.7:3590 [3-1.1]
   (JTAG Server Version 22.1.0 Build 174 03/30/2022 SC Pro Edition)
  C341A0DD   AGFB014R24A(.|R1|R2)/.. (IR=10)
    Design hash    74DFB795DF11A555CEDF
    + Node 00486E00  Source/Probe #0
    + Node 08986E00  Nios V #0
    + Node 0C006E00  JTAG UART #0
  031830DD   10M16S(A|C|L) (IR=10)

  Captured DR after reset = (30D068376063061BB020D10DD) [98]
  Captured IR after reset = (006AAD55) [32]
  Captured Bypass after reset = (0A) [5]
  Captured Bypass chain = (00) [5]
  JTAG clock speed auto-adjustment is enabled. To disable, set
JtagClockAutoAdjust parameter to 0
  JTAG clock speed 24 MHz
```

Refer to the following steps to debug the Nios V processor application ELF in the GDB:

1. On the first terminal or command prompt, start the Ashling GDBServer. For the rest of the walkthrough, this terminal is called as Ashling GDBServer.

```
$ ash-riscv-gdb-server.exe --device C341A0DD \
--auto-detect true --probe-type usb-blaster-2 \
--core-number 0 --jtag-frequency auto-adjust \
--gdb-port 54321

Ashling GDB Server for RISC-V (ash-riscv-gdb-server).
v23.3.2, 01-Sep-2023, (c)Ashling Microsystems Ltd 2023.

Initializing connection ...
Checking for an active debug connection using the selected debug probe (SN:
16777217):
Connected to target device with IDCODE 0xc341a0dd using USB-Blaster-2 via
JTAG at adaptive speed.
Info : Active Harts Detected : 1
Info : [0] System architecture : RV32
```

```
Info : [0] Number of hardware breakpoints available : 1
Info : [0] Number of program buffers: 8
Info : [0] Number of data registers: 2
Info : [0] Memory access -> Program buffer
Info : [0] Memory access -> Abstract access memory
Info : [0] CSR & FP Register access -> Abstract commands

Waiting for debugger connection on port 54321 for core 0.
Press 'Q' to Quit.
```

2. On another terminal or command prompt, start the GDB and connect to port 54321. For the rest of the walkthrough, this terminal is called as GBD.

```
$ riscv32-unknown-elf-gdb.exe app_hal.elf

GNU gdb (GDB) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv32-
unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from app_hal.elf...

(gdb) target remote:54321
Remote debugging using :54321
0x00000008 in ?? ()
```

Upon success, the Ashling GDBServer responds with the following messages.

```
Got a debugger connection from 127.0.0.1 on port 54321.
```

3. You can start debugging the Nios V processor application ELF in the GDB with GDB connected to the Ashling GDBServer.

```
(gdb) load
Loading section .entry, size 0x20 lma 0x0
Loading section .exceptions, size 0x29c lma 0x20
Loading section .text, size 0x18bf4 lma 0x2bc
Loading section .rodata, size 0xf70 lma 0x18eb0
Loading section .rwdata, size 0x1bd8 lma 0x1b9f8
Start address 0x00000354, load size 113144
Transfer rate: 6 KB/sec, 927 bytes/write.

(gdb) b main
Breakpoint 1 at 0x324: file sw/app_hal/hello.c, line 16
(gdb) continue
Continuing.
Breakpoint 1, main () at sw/app_hal/hello.c:16
```

4. To quit the GDB, use the `exit` command, and answer `y` to detach from the processor.

```
(gdb) exit
A debugging session is active.

        Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) y
```

```
Detaching from program: app_hal.elf, Remote target
Ending remote debugging.
[Inferior 1 (Remote target) detached]
```

5. Upon quitting GDB, the Ashling GDBServer responds with the following messages. Continue to quit the Ashling GDBServer by pressing Q.

```
Debugger disconnected, closed connection to 127.0.0.1 on port 54321.
Q
Quitting...
```

Related Information

- Agilex 7 FPGA - Hello World Design on Nios® V/m Processor
- Agilex 7 FPGA F-Series Development Kit (P-Tile and E-Tile)

A.3.4. ARM HPS Core Example

The following example shows how to run Ashling GDBServer at port 12345 on Agilex 7 FPGA F-Series Transceiver-SOC Development Kit (Production 1 P-Tiles and E-Tiles) GHRD (DK-SI-AGF014EA) with **Ashling Sample Project**.

1. Locate the **Ashling Sample Project** for Agilex 7 SoC Development Kit in `<Intel Quartus Prime installation directory>/riscfree/examples/agilex/agilex-a53-sum.zip`.

```
$ jtagconfig.exe -d
1) Agilex SI/SoC Dev Kit on 10.107.209.0:3560 [3-1.4.1]
   (JTAG Server Version 22.1.0 Build 174 03/30/2022 SC Pro Edition)
  6BA00477   S10HPS/AGILEX_HPS/N5X_HPS (IR=4)
  C341A0DD   AGFB014R24A(.|R1|R2)/.. (IR=10)
  031830DD   10M16S(A|C|L) (IR=10)

  Captured DR after reset = (6BA00477C341A0DD031830DD) [96]
  Captured IR after reset = (100555) [24]
  Captured Bypass after reset = (0) [3]
  Captured Bypass chain = (0) [3]
  JTAG clock speed auto-adjustment is enabled. To disable, set
JtagClockAutoAdjust parameter to 0
  JTAG clock speed 24 MHz
```

2. On the first terminal or command prompt, start the Ashling GDBServer. For the rest of the walkthrough, this terminal is called as Ashling GDBServer.

```
$ ash-arm-gdb-server.exe --device 6BA00477 \
--auto-detect true --probe-type usb-blaster-2 \
--core-number 0 --jtag-frequency auto-adjust \
--gdb-port 12345

Ashling GDB Server for ARM (ash-arm-gdb-server).
v23.3.2, 01-Sep-2023, (c)Ashling Microsystems Ltd 2023.

Checking for an active debug connection using the selected debug probe (SN:
16777217):
Connected to target device with IDCODE 0x6ba00477 using USB-Blaster-2 via
JTAG at adaptive speed.
Waiting for debugger connection on port 12345 for core 0.
Press 'Q' to Quit.
```

3. On another terminal or command prompt, start the GDB and connect to port 12345. For the rest of the walkthrough, this second terminal is called as GDB.

```
$ aarch64-none-elf-gdb.exe agilex-a53-sum.elf

GNU gdb (GDB) 13.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=aarch64-none-
elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from agilex-a53-sum.elf...

(gdb) target remote:12345
Remote debugging using :12345
0x00000000ffe00000 in ?? ()
```

   Upon success, the Ashling GDBServer responds with the following messages.

```
Got a debugger connection from 127.0.0.1 on port 12345.
```

4. With GDB connected to the Ashling GDBServer, you can start debugging the Arm HPS core application ELF in the GDB.

```
(gdb) load
Loading section .text, size 0x718 lma 0x7fe00300
Loading section .init, size 0x34 lma 0x7fe00a18
Loading section .fini, size 0x34 lma 0x7fe00a4c
Loading section .rodata, size 0x58 lma 0x7fe00a80
Loading section .eh_frame, size 0x4 lma 0x7fe00ad8
Loading section .data, size 0x758 lma 0x7fe00ae0
Loading section .init_array, size 0x8 lma 0x7fe01238
Loading section .fini_array, size 0x8 lma 0x7fe01240
Start address 0x000000007fe00468, load size 3908
Transfer rate: 592 bytes/sec, 488 bytes/write.

(gdb) b main
Breakpoint 1 at 0x7fe004ec: file ../main.c, line 32.
(gdb) continue
Continuing.

Breakpoint 1, main () at ../main.c:32
```

5. To quit the GDB, use the `exit` command, and answer `y` to detach from the processor.

```
(gdb) exit
A debugging session is active.

        Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) y
Detaching from program: agilex-a53-sum.elf, Remote target
Ending remote debugging.
[Inferior 1 (Remote target) detached]
```

6. Upon quitting GDB, the Ashling GDBServer responds with the following messages. Continue to quit the Ashling GDBServer by pressing Q.

```
Debugger disconnected, closed connection to 127.0.0.1 on port 12345.
Q
Quitting...
```

Related Information

- GSRD for Agilex 7 F-Series Transceiver-SoC DevKit (P-Tile and E-Tile)
- Agilex 7 FPGA F-Series Transceiver-SoC Development Kit (P-Tile and E-Tile)

A.4. Help Content

For more information about RiscFree IDE, go to **Help ➤ Help Content**. You can use the search bar to find the information about a specific topic.

A.5. Open-Source Components in RiscFree IDE

The following table shows the open-source components bundled with the RiscFree IDE.

Table 24. RiscFree IDE Open-Source Components

Open-Source Component	Version Number	Source Code Link	Tag
Eclipse CDT (C/C++ Development Tooling)	11.6.1	https://github.com/AshlingMicrosystems/cdt	cdt-RFI-v25.1.1
Eclipse Embedded CDT	6.6.0	https://github.com/AshlingMicrosystems/eclipse-plugins	eclipse-plugin-RFI-v25.1.1
Eclipse SWT (Standard Widget Tooling	4.32	https://github.com/AshlingMicrosystems/eclipse.platform.swt	eclipse-platform-swt-RFI-v25.1.1
cmake4eclipse	2.1.4	https://github.com/AshlingMicrosystems/cmake4eclipse	cmake4eclipse-RFI-v25.1.1
RISC-V GNU GCC	13.2	https://github.com/AshlingMicrosystems/gcc	gcc-RFI-v25.1.1
RISC-V GNU Binutils	2.41	https://github.com/AshlingMicrosystems/binutils-gdb	gdb-RFI-v25.1.1
RISC-V GNU GDB	13.2	https://github.com/AshlingMicrosystems/binutils-gdb	gdb-RFI-v25.1.1
RISC-V Newlib	4.4.0	https://github.com/AshlingMicrosystems/newlib	newlib-RFI-v25.1.1
RISC-V Picolibc	1.8.5	https://github.com/AshlingMicrosystems/picolibc	picolibc-RFI-v25.1.1
QEMU (Arm and RISC-V)	9.0.2	https://github.com/AshlingMicrosystems/qemu	qemu-RFI-25.1.1
OpenCSD	1.4.0	https://github.com/AshlingMicrosystems/OpenCSD	opencsd-RFI-25.1.1
Eclipse IDE	2024-06	—	—
GNU Make	4.4.1	—	—
CMake	3.31.1	—	—

continued...

Open-Source Component	Version Number	Source Code Link	Tag
GNU Arm 32-bit GDB (arm-none-eabi 32-bit)	13.2	—	—
GNU Arm 64-bit GDB (aarch64-elf 64-bit)	13.2	—	—
Java Runtime Environment (OpenJDK)	21	—	—