Altera SoC Embedded Design Suite User Guide

Subscribe

Send Feedback

ug-1137
2016.02.17

101 Innovation Drive
San Jose, CA 95134
www.altera.com

now part of Intel

Contents

Introduction to SoC Embedded Design Suite 1

2016.02.17

ug-1137 Subscribe Send Feedback

The Altera® system on a chip (SoC) Embedded Design Suite (EDS) provides the tools needed to develop embedded software for Altera's SoC devices.

The Altera SoC EDS is a comprehensive tool suite for embedded software development on Altera SoC devices. The Altera SoC EDS contains development tools, utility programs, run-time software, and application examples that enable firmware and application software development on the Altera SoC hardware platform.

Overview

The Altera SoC EDS enables you to perform all required software development tasks targeting the Altera SoCs, including:

- Board bring-up
- Device driver development
- Operating system (OS) porting
- Bare-metal application development and debugging
- OS- and Linux-based application development and debugging
- Debug systems running symmetric multiprocessing (SMP)
- Debug software targeting soft IP residing on the FPGA portion of the device

The major components of the SoC EDS include:

- ARM® Development Studio 5 (DS-5™) AE (AE) Toolkit
- Compiler tool chains:
  - Bare-metal GNU Compiler Collection (GCC) tool chain from Mentor Graphics®
  - Bare-metal compiler
  - Linux GCC compiler tool chain from Linaro®
- Pre-built Linux package including:
  - Linux kernel executable
  - U-boot image
  - Bootloader Device Tree —Device tree blob consumed by u-boot for A10
  - Linux Device Tree—Device tree blob consumed by Linux
  - Secure Digital (SD) card image
  - Script to download Linux source code from the GitHub repository(1)
- SoC Hardware Library (HWLIB)

- Hardware-to-software interface utilities:
  - Second stage bootloader generator
  - Linux device tree generator
- Sample applications
- Golden Hardware Reference Designs (GHRD) including:
  - FPGA hardware project
  - FPGA hardware SRAM Object File (.sof) file
  - Precompiled Second Stage Bootloaders
    - Preloader for Arria V and Cyclone V
    - Bootloader for Arria 10
- Embedded command shell allowing easy invocation of the included tools
- SD Card Boot Utility
- Quartus® Prime Programmer and SignalTap II

Note: The Linux package included in the SoC EDS is not an official release and is intended to be used only as an example. Use the official Linux release described in the *Golden System Reference Design (GSRD) User Manual* available on the Rocketboards website or a specific release from the Git trees located on the GitHub repository for development.

Note: The SoC EDS is tested only with the Linux release that comes with it. Newer Linux releases may not be fully compatible with this release of SoC EDS.

Note: The Golden Hardware Reference Design (GHRD) included with the SoC EDS is not an official release and is intended to be used only as an example. For development purposes, use the official GHRD release described in the *GSRD User Manual* available on the Rocketboards website.

Related Information

- **Golden System Reference Design User Manual**
- **GitHub Repository**

Linux Device Tree Binary

There are two different Linux device tree binary (DTB) file versions delivered as part of the SoC EDS:

- The version from the **prebuilt_images** folder: **socfpga_cyclone5.dtb** and **socfpga_arria10.dtb** are generic DTB files which do not have any dependency on soft IP. FPGA programming and bridge releasing are not required before Linux starts running using this DTB.

  This DTB file is intended for customers interested in bringing up a new board or just wanting to simplify their boot flow until they get to the Linux prompt. If what is being developed or debugged does not involve the FPGA, it is better to remove the FPGA complexities.
- The version from the hardware design folder: **soc_system.dtb**, **ghrd_5astfd5k3.dtb**, and **ghrd_10as066n2.dtb** are based on the GHRD design, which is part of the GSRD. Since the GHRD does contain soft IPs, these DTB file versions notify Linux to load the soft IP drivers. Therefore, the FPGA needs to be programmed and the bridges released before booting Linux.

(1) The script downloads the sources corresponding to the pre-built Linux package.

Hardware and Software Development Roles

Depending on your role in hardware or software development, you need a different subset of the SoC EDS toolkit. The following table lists some typical engineering development roles and indicates the tools that each role typically requires.

For more information about each of these tools, refer to the **SoC Embedded Design Suite** page.

Table 1-1: Hardware and Software Development Roles

This table lists typical tool usage, but your actual requirements depend on your specific project and organization.

Tool	Hardware Engineer	Bare-Metal Developer	RTOS Developer	Linux Kernel and Driver Developer	Linux Application Developer
ARM DS-5 Debugging	√	√	√	√	√
ARM DS-5 Tracing		√	√	√	
ARM DS-5 Cross Triggering		√	√	√	
Hardware Libraries		√	√	√	
Second Stage Bootloader Generator	√	√	√	√	
Flash Programmer		√	√	√	√
Bare-Metal Compiler	√	√	√	√	
Linux Compiler				√	√
Linux Device Tree Generator				√	

Hardware Engineer

As a hardware engineer, you typically design the FPGA hardware in Qsys. You can use the debugger of ARM DS-5 AE to connect to the ARM cores and test the hardware. A convenient feature of the DS-5 debugger is the soft IP register visibility, using Cortex Microcontroller Software Interface Standard (CMSIS) System View Description (**.svd**) files. With this feature, you can easily read and modify the soft IP registers from the ARM side.

As a hardware engineer, you may generate the Preloader for your hardware configuration. The Preloader is a piece of software that configures the HPS component according to the hardware design.

As a hardware engineer, you may also perform the board bring-up. You can use ARM DS-5 debugger to verify that they can connect to the ARM and the board is working correctly.

These tasks require JTAG debugging, which is enabled only in the Subscription Edition. For more information, see the *Licensing* section.

Related Information

- Licensing on page 3-1
- Hardware – Software Development Flow on page 1-5
  For more information about **.svd** files, refer to the Hardware - Software Development Flow section.

Bare-Metal and RTOS Developer

As either a bare-metal or a RTOS developer, you need JTAG debugging and low-level visibility into the system.

Use the bare-metal compiler to compile your code and the SoC Hardware Library to control the hardware in a convenient and consistent way.

Use the Flash Programmer to program the flash memory on the target board.

These tasks require JTAG debugging, which is enabled only in the Subscription Edition. For more information, see the *Licensing* section.

Related Information

Licensing on page 3-1

Linux Kernel and Driver Developer

As a Linux kernel or driver developer, you may use the same tools the RTOS developers use, because you need low-level access and visibility into the system. However, you must use the Linux compiler instead of the bare-metal compiler. You can use the Linux device tree generator (DTG) to generate Linux device trees.

These tasks require JTAG debugging, which is enabled only in the Subscription Edition.

For more information, see the "Licensing" chapter.

Related Information

Licensing on page 3-1

Linux Application Developer

As a Linux application developer, you write code that targets the Linux OS running on the board. Because the OS provides drivers for all the hardware, you do not need low-level visibility over JTAG. DS-5 offers a very detailed view of the OS, showing information such as which threads are running and which drivers are loaded.

These tasks do not require JTAG debugging. You can perform them both in the Web and Subscription editions. For more information, see the *Licensing* section.

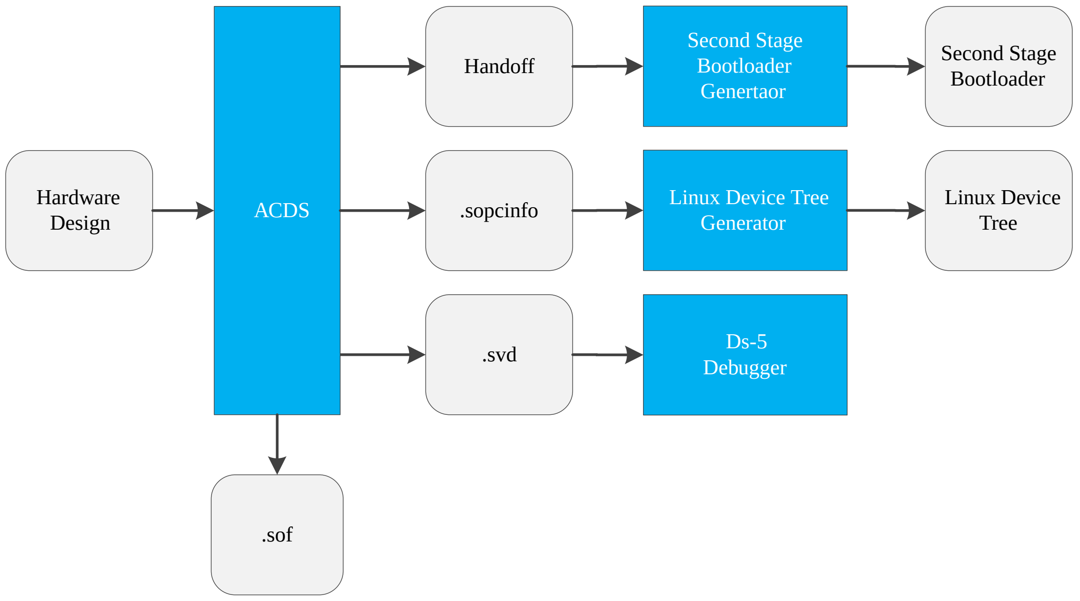
Related Information

Licensing on page 3-1

Hardware – Software Development Flow

The Altera hardware-to-software handoff utilities allow hardware and software teams to work independently and follow their respective familiar design flows.

Figure 1-1: Altera Hardware-to-Software Handoff



The following handoff files are created when the hardware project is compiled:

- **Handoff** folder – contains information about how the HPS component is configured, including things like which peripherals are enabled, the pin MUXing and IOCSR settings, and memory parameters
- **.svd** file – contains descriptions of the HPS registers and of the soft IP registers on FPGA side implemented in the FPGA portion of the device
- **.sopcinfo** file – contains a description of the entire system

The handoff folder is used by the second stage bootloader generator to create the preloader.

For more information about the handoff folder, refer to the *BSP Generation Flow* chapter.

The **.svd** file contains the description of the registers of the HPS peripheral registers and registers for soft IP components in the FPGA portion of the SoC. This file is used by the ARM DS-5 Debugger to allow these registers to be inspected and modified by the user.

The SOPC Information (**.sopcinfo**) file, containing a description of the entire system, is used by the Linux device tree generator to create the device tree used by the Linux kernel.

For more information, refer to the "Linux Device Tree Generator" chapter.

Note: The soft IP register descriptions are not generated for all soft IP cores.

Related Information

- **BSP Generation Flow** on page 7-2
- **Linux Device Tree Generator**
- **SoC Embedded Design Suite Download Page**

SoC EDS Introduction Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Maintenance release.
August 2015	2015.08.06	Added Arria 10 support.

Installing the Altera SoC Embedded Design Suite

2

2016.02.17

ug-1137 Subscribe Send Feedback

You must install the Altera SoC Embedded Design Suite (EDS) and the ARM DS-5 AE to run the SoC EDS on an Altera SoC hardware platform.

Installation Folders

The default installation folder for SoC EDS is:

- *<SoC EDS installation directory>*
  - **c:\altera\15.1\embedded** on Windows
  - **~/altera/15.1/embedded** on Linux

The default installation folder for Quartus Prime Programmer is:

- *<Quartus installation directory>*
  - **c:\altera\15.1\qprogrammer** on Windows
  - **~/altera/15.1/qprogrammer** on Linux

Note: The installation directories are defined, as follows:

- *<Altera installation directory>* to denote the location where Altera tools are installed.
- *<SoC EDS installation directory>* to denote the location where SoC EDS is installed.

Installing the SoC EDS

Perform the following steps to install the SoC EDS Tool Suite in a Windows-based system:

1. Download the latest installation program from the *SoC Embedded Design Suite Download Center* page of the Altera website.
2. Run the installer to open the **Installing SoC Embedded Design Suite (EDS)** dialog box, and click **Next** to start the **Setup Wizard**.
3. Accept the license agreement, and click **Next.**
4. Accept the default installation directory or browse to another installation directory, and click **Next**.

   Note: If you have previously installed the Quartus Prime software, accept the default *<SoC EDS installation directory>* to allow the Quartus Prime software and the SoC EDS Tool Suite to operate together.

5. Select **All** the components to be installed, and click **Next**. The installer displays a summary of the installation.
6. Click **Next** to start the installation process. The installer displays a separate dialog box with the installation progress of the component installation.
7. When the installation is complete, turn on **Launch DS-5 Installation** to start the ARM DS-5 installation, and click **Finish**.

Note: On some Linux-based machines, you can install the SoC EDS with a setup GUI similar to the Windows-based setup GUI. Because of the variety of Linux distributions and package requirements, not all Linux machines can use the setup GUI. If the GUI is not available, use an equivalent command-line process. Download the Linux installation program from the *SoC Embedded Design Suite Download Center* page on the Altera website.

Related Information

SoC Embedded Design Suite Download Center

Installing the ARM DS-5 Altera Edition Toolkit

Before you begin

For the last step of the SoC EDS installation process, start the ARM DS-5 AE Toolkit installer.

Note: Make sure you have the proper setting to access the internet.

1. When the **Welcome** message is displayed, click **Next**.
2. Accept the license agreement and click **Next**.
3. Accept the default installation path, to ensure proper interoperability between SoC EDS and ARM DS-5 AE, and click **Next**.
4. Click **Install** to start the installation process. The progress bar is displayed.
5. When a driver installation window appears, click **Next**.
6. Accept the driver installation and click **Install**.
7. After successful installation, click **Finish**. ARM DS-5 AE installation is complete.
8. Click **Finish**.

Installing Altera SoC Embedded Design Suite Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Updated the installation path to 15.1.
August 2015	2015.08.06	Added Arria 10 support.

Licensing 3

2016.02.17

ug-1137 Subscribe Send Feedback

The SoC EDS is available with three different licensing options:

- Subscription edition
- Free web edition
- 30-day evaluation of subscription edition

The only tool impacted by the selected licensing option is the ARM DS-5 AE. All the other tools offer the same level of features in all licensing options; for example, the second stage bootloader generator and the bare-metal compiler offer the same features no matter which licensing option is used.

The main difference between the licensing options depends on which types of debugging scenarios are enabled:

Licensing Option	Debugging Scenarios Enabled
Web edition	• Linux application debugging over ethernet
Subscription edition 30-day evaluation of the subscription edition	• JTAG-based Bare-Metal Debugging • JTAG-based Linux Kernel and Driver Debugging • Linux Application Debugging over Ethernet

Getting the License

Depending on the licensing option, it is necessary to follow the steps detailed for each option to obtain the license.

Subscription Edition - If you have purchased the SoC EDS Subscription Edition, then you have already received an ARM license serial number. This is a 15-digit alphanumeric string with two dashes in between. You will need to use this number to activate your license in DS-5, as shown in the *Activating the License* section.

Free Web Edition - For the free SoC EDS Web Edition, you will be able to use DS-5 perpetually to debug Linux applications over an Ethernet connection. Get your ARM license activation code from the SoC Embedded Design Suite download page on the Altera website and then activate your license in DS-5, as shown in the *Activating the License* section.

30-Day Evaluation of Subscription Edition - If you want to evaluate the SoC EDS Subscription Edition, you can get a 30-Day Evaluation activation code from the SoC Embedded Design Suite download page on the Altera website and then activate your license in DS-5, as shown in the *Activating the License* section.

Related Information

- SoC EDS Download Page
- Activating the License on page 3-2

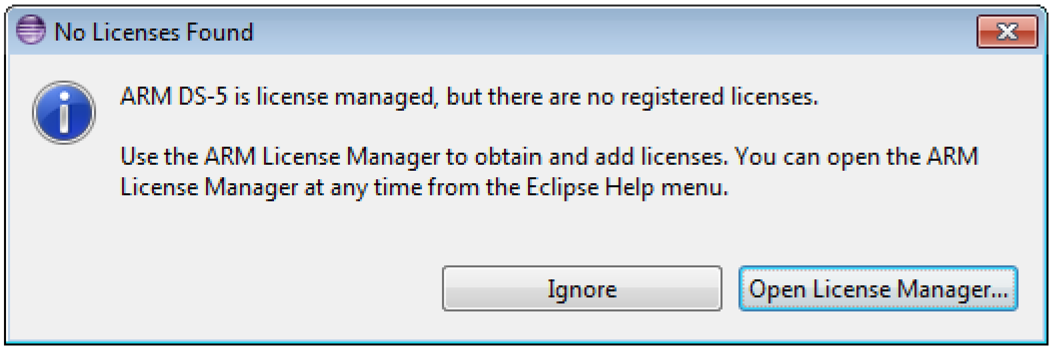
Activating the License

This section presents the steps required for activating the license in the ARM DS-5 AE by using the serial license number or activation code that were mentioned in the "Getting the License" chapter.

Note: An active user account is required to activate the DS-5 AE license. If you do not have an active user account, it can be created on the ARM *Self-Service* page available on the ARM website.

1. The first time the ARM DS-5 is run, it notifies you that it requires a license. Click the **Open License Manager** button.

Figure 3-1: No License Found

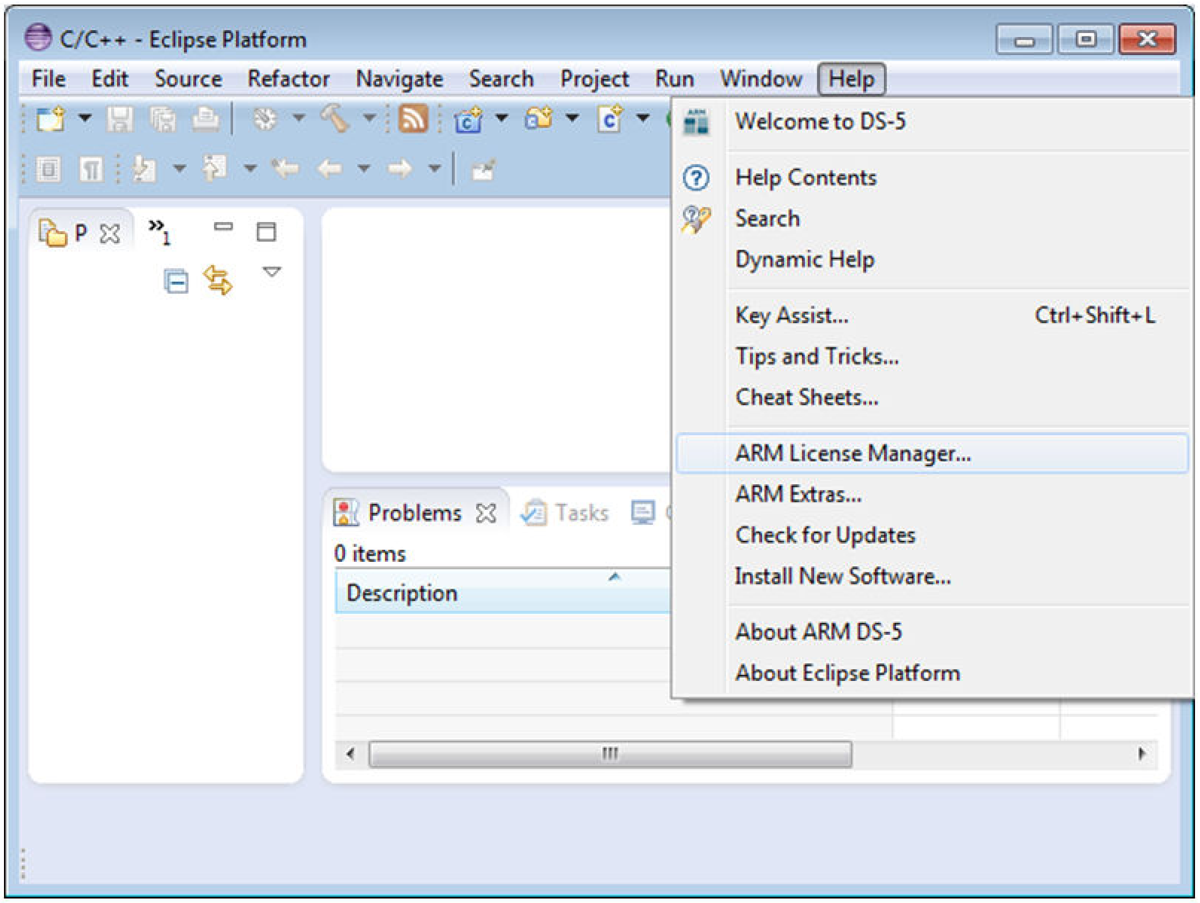

No Licenses Found

ARM DS-5 is license managed, but there are no registered licenses.

Use the ARM License Manager to obtain and add licenses. You can open the ARM License Manager at any time from the Eclipse Help menu.

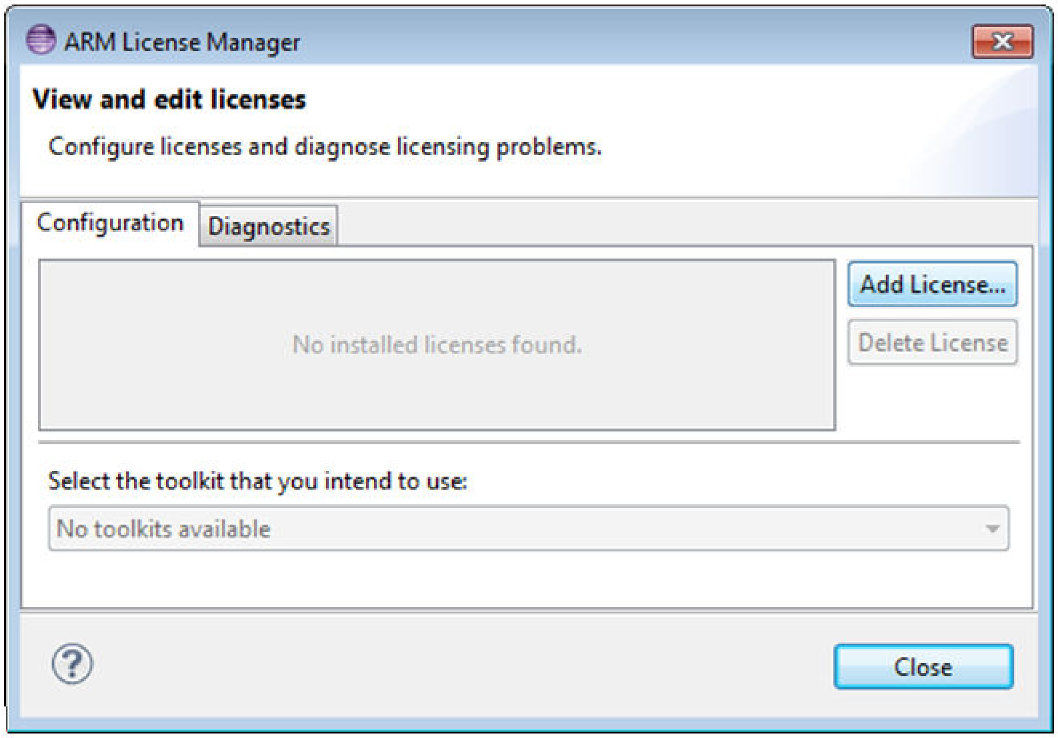
2. If at any time it is required to change the license, select **Help** > **ARM License Manager** to open the **License Manager**.

Figure 3-2: Accessing ARM License Manager



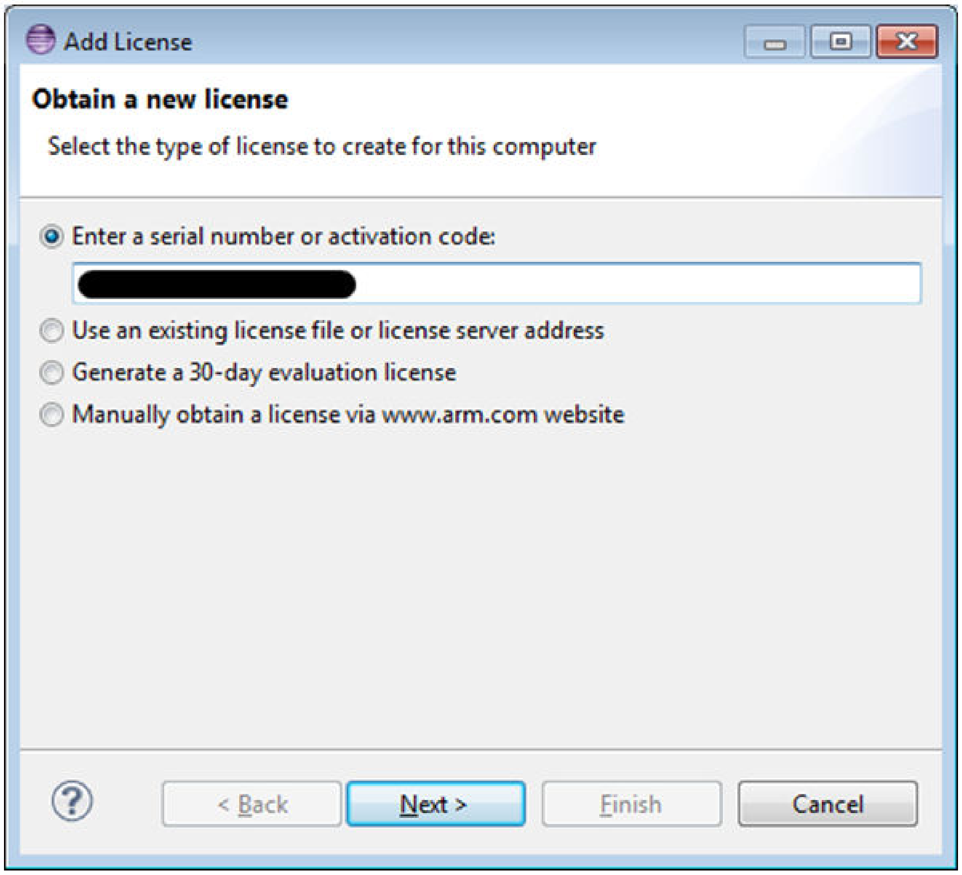
3. The **License Manager - View and edit licenses** dialog box opens and shows that a license is not available. Click the **Add License** button.

Figure 3-3: ARM License Manager


ARM License Manager
View and edit licenses
Configure licenses and diagnose licensing problems.
Configuration
Diagnostics
No installed licenses found.
Add License...
Delete License
Select the toolkit that you intend to use:
No toolkits available
Close

4. In the **Add License** - **Obtain a new licenses** dialog box, select the type of license to enter. In this example, select the radio button, **"Enter a serial number or activation code to obtain a license"** to enter the choices listed, below. When done, click **Enter**.
   a. ARM License Number for **Subscription Edition**.
   b. ARM License Activation Code for **Web Edition** and **30-Day Evaluation**.

Figure 3-4: Add License - Obtain a New License


Add License

Obtain a new license

Select the type of license to create for this computer

(●) Enter a serial number or activation code: [redacted]

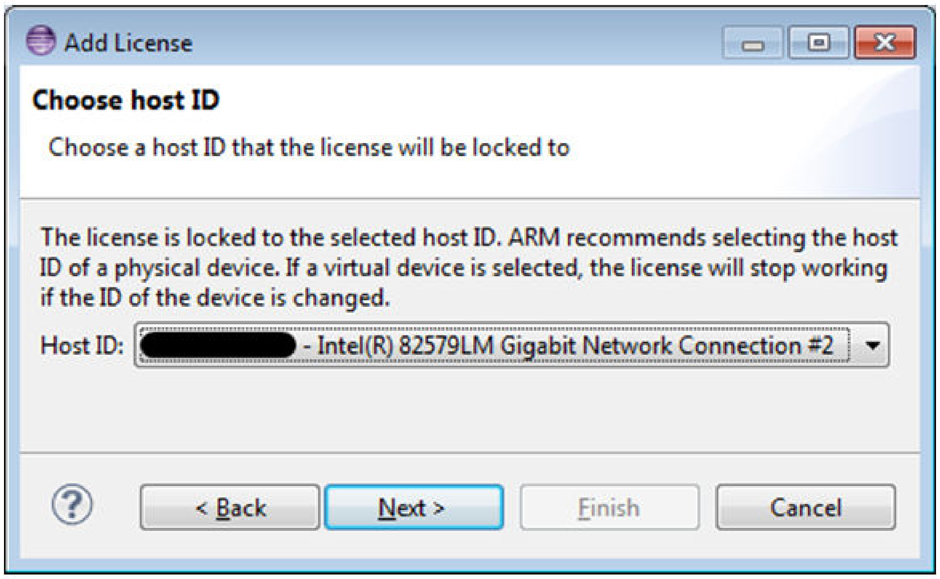
( ) Use an existing license file or license server address

( ) Generate a 30-day evaluation license

( ) Manually obtain a license via www.arm.com website

5. Click **Next**.
6. In the **Add License - Choose Host ID** dialog box, select the Host ID (Network Adapter MAC address) to tie the license to. If there are more than one option, select the one you desire to lock the license to, and click **Next**.

Figure 3-5: Add License - Choose host ID



Add License

Choose host ID

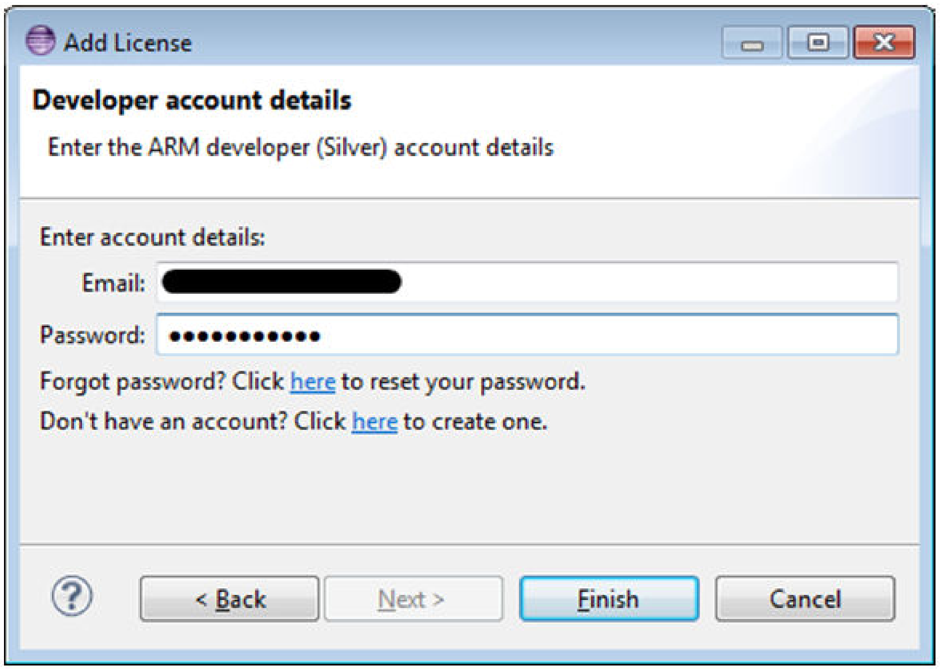
Choose a host ID that the license will be locked to

The license is locked to the selected host ID. ARM recommends selecting the host ID of a physical device. If a virtual device is selected, the license will stop working if the ID of the device is changed.

Host ID: [redacted] - Intel(R) 82579LM Gigabit Network Connection #2

7. In the **Add License - Developer account details** dialog box, enter an ARM developer (Silver) account. If you do not have an account, it can be created easily by clicking the provided link. After entering the account information, click **Finish**.

Figure 3-6: Add License - Developer Account Details



Add License

Developer account details

Enter the ARM developer (Silver) account details

Enter account details:

Email: [redacted]

Password: ●●●●●●●●●●●●

Forgot password? Click here to reset your password.

Don't have an account? Click here to create one.

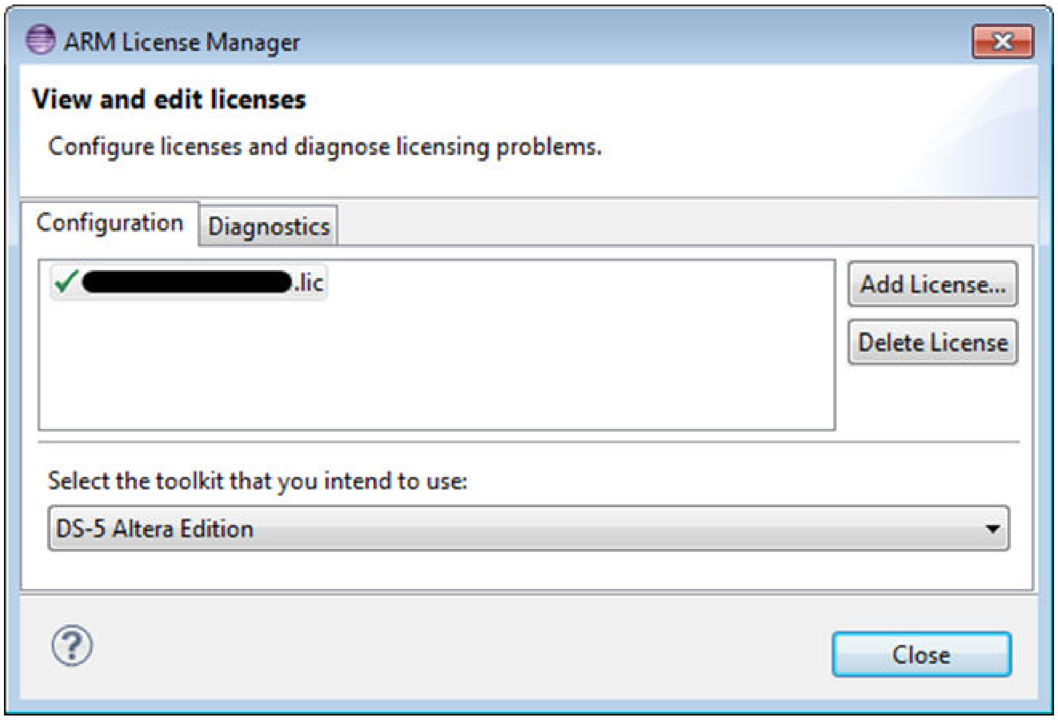
Note: The License Manager needs to be able to connect to the Internet in order to activate the license. If you do not have an Internet connection, you will need to write down your Ethernet MAC address and generate the license directly from the *ARM Self-Service* web page on the ARM website, then select the "**Already have a license**" option in the License Manger.

Note: Only the Subscription Edition, with an associated license number can be activated this way. The Web Edition and Evaluation edition are based on activation codes, and these codes cannot be used on the *ARM Self-Service* web page on the ARM website. They need to be entered directly in the License Manager; which means an Internet connection is a requirement for licensing.

The ARM License Manager uses the Eclipse settings to connect to the Internet. The default Eclipse settings use the system-wide configuration for accessing the Internet. In case the License Manager cannot connect to the Internet, you can try to change the Proxy settings by going to **Window** > **Preferences** > **General** > **Network Connections**. Ensure that "HTTPS" proxy entry is configured and enabled.

8. After a few moments, the `ARM DS-5` will activate the license and display it in the **License Manager**. Click **Close**.

Figure 3-7: ARM License Manager



Related Information

- ARM website
- Getting the License on page 3-1

SoC EDS Licensing Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Maintenance release.
August 2015	2015.08.06	Added Arria 10 support.

Embedded Command Shell 4

2016.02.17

ug-1137 Subscribe Send Feedback

The purpose of the embedded command shell is to provide an option for you to invoke the SoC EDS tools. It enables you to invoke the SoC EDS tools without qualifying them with the full path. Commands like '`eclipse`', '`bsp-editor`', or '`arm-altera-eabi-gcc`' can be executed directly.

On Windows, the embedded command shell is started by running **<SoC EDS installation directory>\Embedded_Command_Shell.bat**.

On Linux, the embedded command shell is started from the **Start** menu or by running <*SoC EDS installation directory*>**/embedded_command_shell.sh**.

Embedded Command Shell Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Maintenance release.
August 2015	2015.08.06	Added Arria 10 support.

Getting Started Guides 5

2016.02.17

ug-1137 Subscribe Send Feedback

The Getting Started Guides chapter provides instructions on how to access complete Getting Started instructions including the following:

- Board Setup
- Running Linux
- Running the Tools
- Second Stage Bootloader
- Baremetal Debugging
- Hardware Libraries (HWLibs)
- Peripheral Register Visibility
- Baremetal Project Management
- Linux Application Debugging
- Linux Kernel and Driver Debugging
- Tracing

Related Information

SoCEDSGettingStarted Wiki Page

For more information on how to access the Getting Started Guides.

Getting Started Guides Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Maintenance release.
August 2015	2015.08.06	Removed content from this section and moved it to the Altera Wiki.

Related Information

SoCEDSGettingStarted

For more information about getting started, refer to the SoCEDSGettingStarted page on the Altera Wiki.

ARM DS-5 AE 6

2016.02.17

ug-1137 Subscribe Send Feedback

The ARM Development Studio 5 Toolkit AE (ARM DS-5 AE) is a device-specific exclusive offering from Altera.

The ARM DS-5 AE is a powerful Eclipse-based comprehensive Integrated Development Environment (IDE). Some of the most important provided features are:

- File editing, supporting syntax highlighting and source code indexing
- Build support, based on makefiles
- Bare-metal debugging
- Linux application debugging
- Linux kernel and driver debugging
- Multicore debugging
- Access to HPS peripheral registers
- Access to FPGA soft IP peripheral registers
- Tracing of program execution through Program Trace Macrocells (PTM)
- Tracing of system events through System Trace Macrocells (STM)
- Cross-triggering between HPS and FPGA
- Connecting to the target using Altera USB Blaster™ II

The ARM DS-5 AE is a complex tool with many features and options. This chapter only describes the most common features and options and provides getting started scenarios to help you get started, quickly.

You can access the ARM DS-5 AE reference material from Eclipse, by navigating to **Help** > **Help Contents** > **ARM DS-5 Documentation** or online.

Related Information

Online ARM DS-5 Documentation

The ARM DS-5 AE reference material can be accessed online on the documentation page of the ARM website.

Starting Eclipse ARM DS-5 AE

ARM DS-5 AE must be started from the Embedded Command Shell.

Eclipse needs to be started from the Embedded Command Shell so that all the utilities are added to the search path, and they can be used directly from the makefiles without the full path.

To start the Eclipse IDE that the ARM DS-5 AE uses, you must type **eclipse &** at the command line.

Bare-Metal Project Management

ARM DS-5 AE enables convenient project management for bare-metal projects using two different methods:

- Using Makefiles
- Using the ARM DS-5 AE graphical interface

Some users prefer Makefiles because they allow the option for the project compilation to be performed from scripts. Other users prefer to use a GUI to manage the project, and this is available for both GCC and ARM Compiler bare-metal projects.

Method	Advantages	Compiler Toolchain Support
Makefile	Scripted compilation	GCC
ARM DS-5 AE graphical interface	Multiple toolchain support	GCC, ARM Compiler

Bare-Metal Project Management Using Makefiles

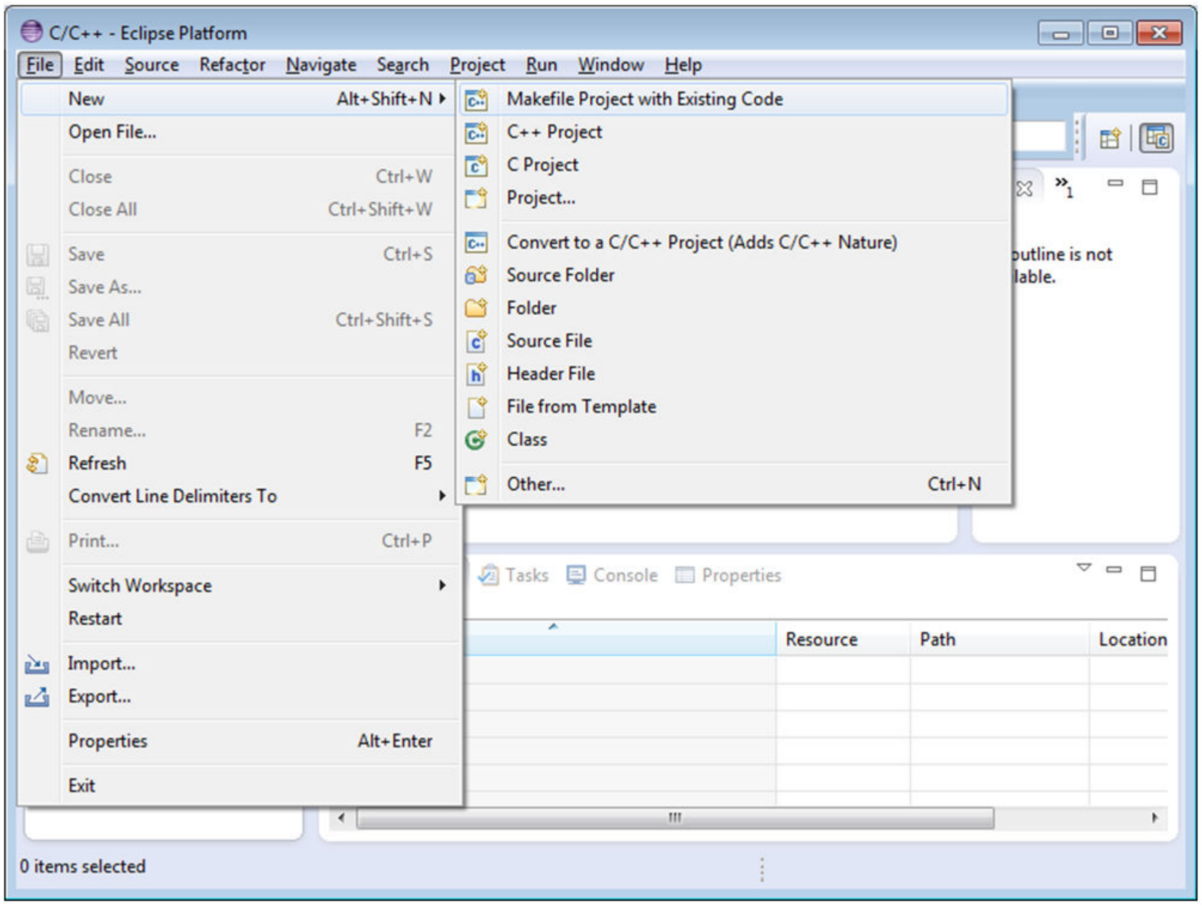
The ARM DS-5 AE enables convenient project management using makefiles. The sample projects that are provided with SoC EDS use makefiles to manage the build process.

Note: This option refers to just the DS-5 specific aspects. If you are not familiar with defining and using makefiles, please use the ARM DS-5 AE GUI option detailed in the next section.

To allow ARM DS-5 AE to manage a makefile-based project, create a project, as follows:

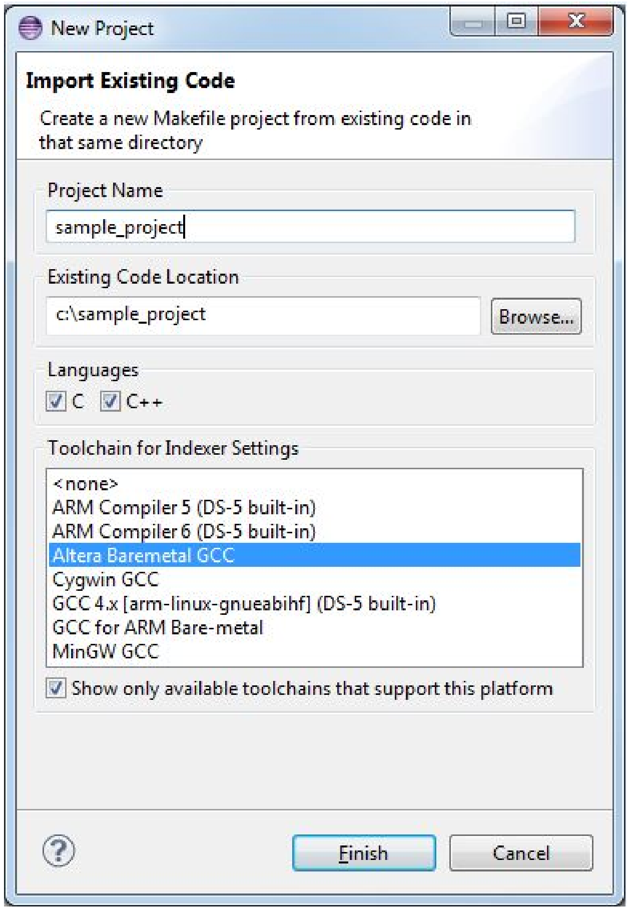
1. Create a folder on the disk.
2. Create the project by selecting **File > New > Makefile Project with Existing Code**.

Figure 6-1: Creating a Project with Existing Code



3. Type the folder name in the **Existing Code Location** edit box and then click **Finish**.

Figure 6-2: Import Existing Code


New Project

Import Existing Code

Create a new Makefile project from existing code in that same directory

Project Name: sample_project

Existing Code Location: c:\sample_project

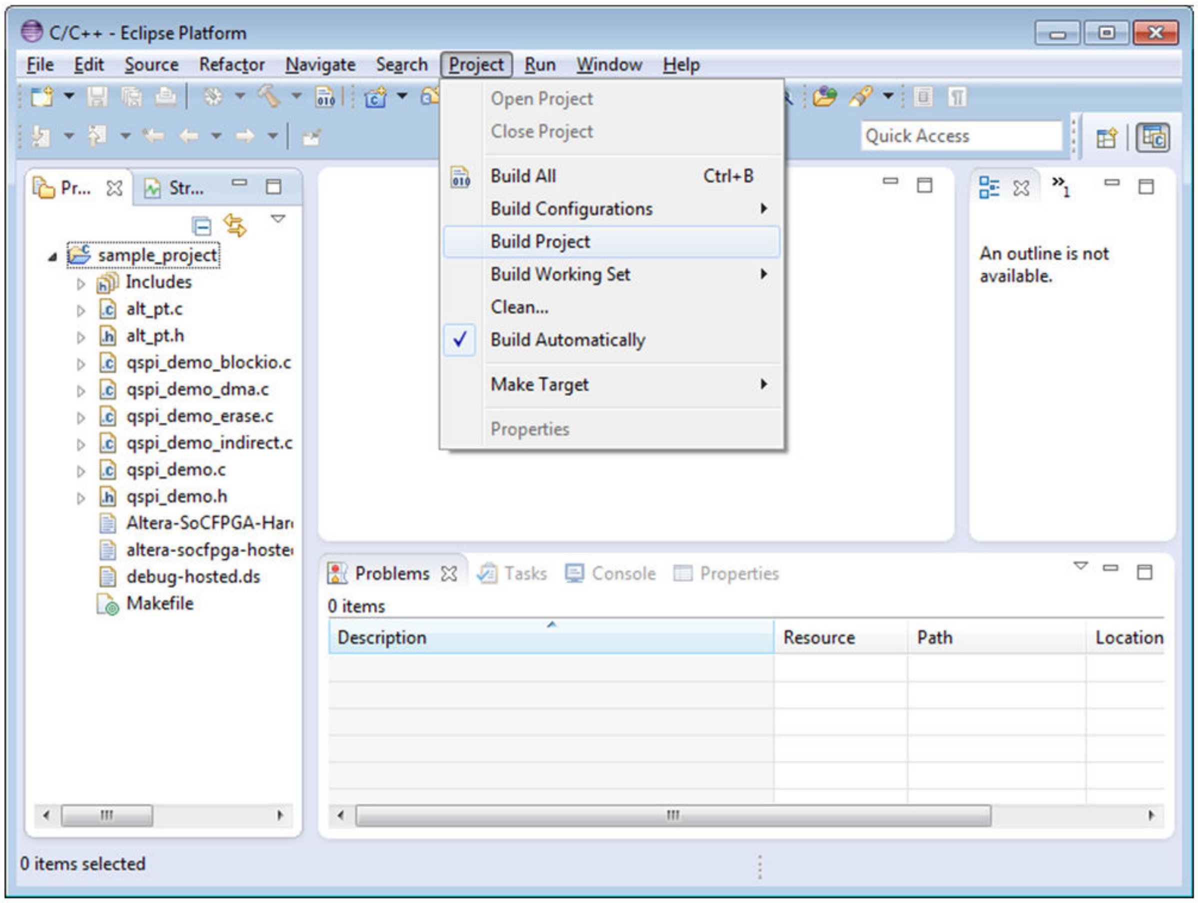
Languages: [✓] C [✓] C++

Toolchain for Indexer Settings:
- <none>
- ARM Compiler 5 (DS-5 built-in)
- ARM Compiler 6 (DS-5 built-in)
- [✓ Altera Baremetal GCC]
- Cygwin GCC
- GCC 4.x [arm-linux-gnueabihf] (DS-5 built-in)
- GCC for ARM Bare-metal
- MinGW GCC

[✓] Show only available toolchains that support this platform

4. Create a Makefile in that folder, and define the rules required for compiling the code. Make sure it has the **all** and the **clean** targets.
   ARM DS-5 AE now offers the possibility of invoking the build process from the IDE and allows you to build your project, as shown in the following figure:

Figure 6-3: Eclipse IDE - Build Process Invoked - Building a Software Project in ARM DS-5 AE



If the compilation tools issue errors, ARM DS-5 AE parses and formats them for you and displays them in the Problems view.

5. Build a software project in ARM DS-5 AE.

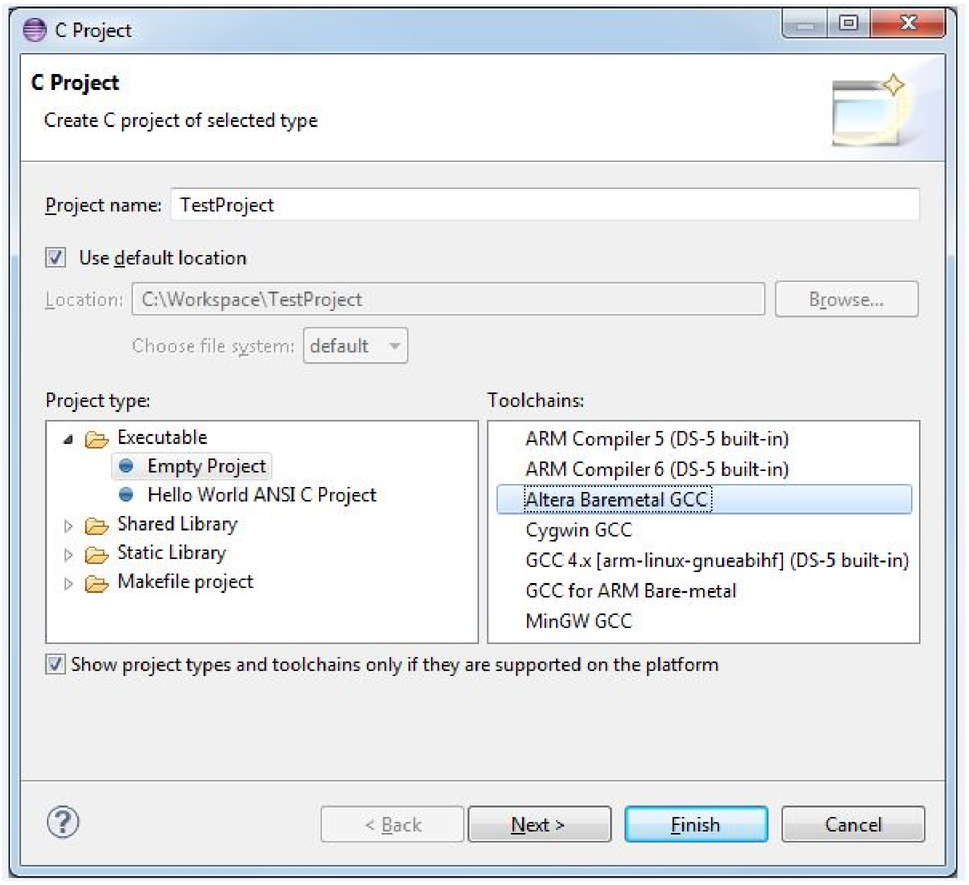
GCC-Based Bare-Metal Project Management

This section shows how the Bare-metal toolchain plugin can be used to manage GCC-based projects in a GUI environment.

Creating Project

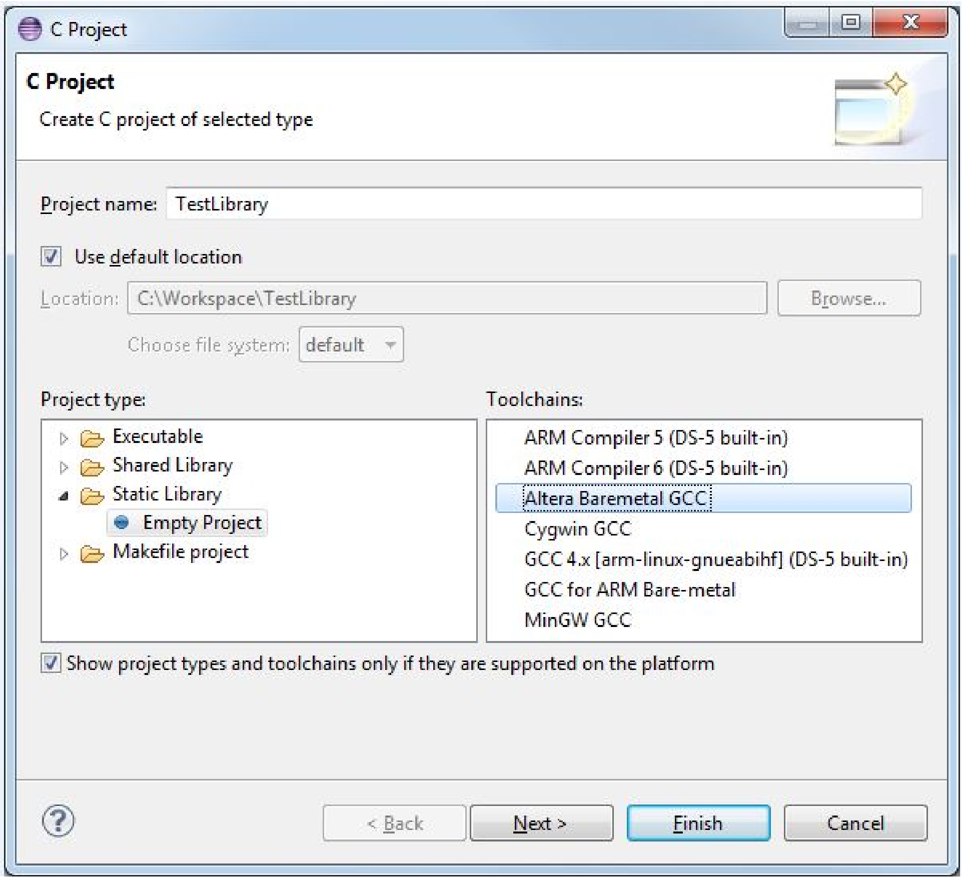
1. Start Eclipse.
2. Go to **File** > **New C Project**.
3. Determine if you want to create an **Executable** empty project or a **Bare-metal library** empty project.
   a. **Bare-metal executable**
      Select **Project Type** to be **Executable** > **Empty Project** then **Toolchain** to be **Altera Baremetal GCC** then click **Finish**.

Figure 6-4: Bare-Metal Executable Project Type


C Project
C Project
Create C project of selected type
Project name: TestProject
Use default location
Location: C:\Workspace\TestProject
Browse...
Choose file system: default
Project type:
Executable
Empty Project
Hello World ANSI C Project
Shared Library
Static Library
Makefile project
Toolchains:
ARM Compiler 5 (DS-5 built-in)
ARM Compiler 6 (DS-5 built-in)
Altera Baremetal GCC
Cygwin GCC
GCC 4.x [arm-linux-gnueabihf] (DS-5 built-in)
GCC for ARM Bare-metal
MinGW GCC
Show project types and toolchains only if they are supported on the platform
< Back
Next >
Finish
Cancel

b. **Static Library**
Select **Bare-metal Library** > **Empty Project** and click Finish.

Figure 6-5: Bare-Metal Library Project Type


C Project

C Project

Create C project of selected type

Project name: TestLibrary

☑ Use default location

Location: C:\Workspace\TestLibrary

Choose file system: default

Project type:
- Executable
- Shared Library
- Static Library
  - Empty Project
- Makefile project

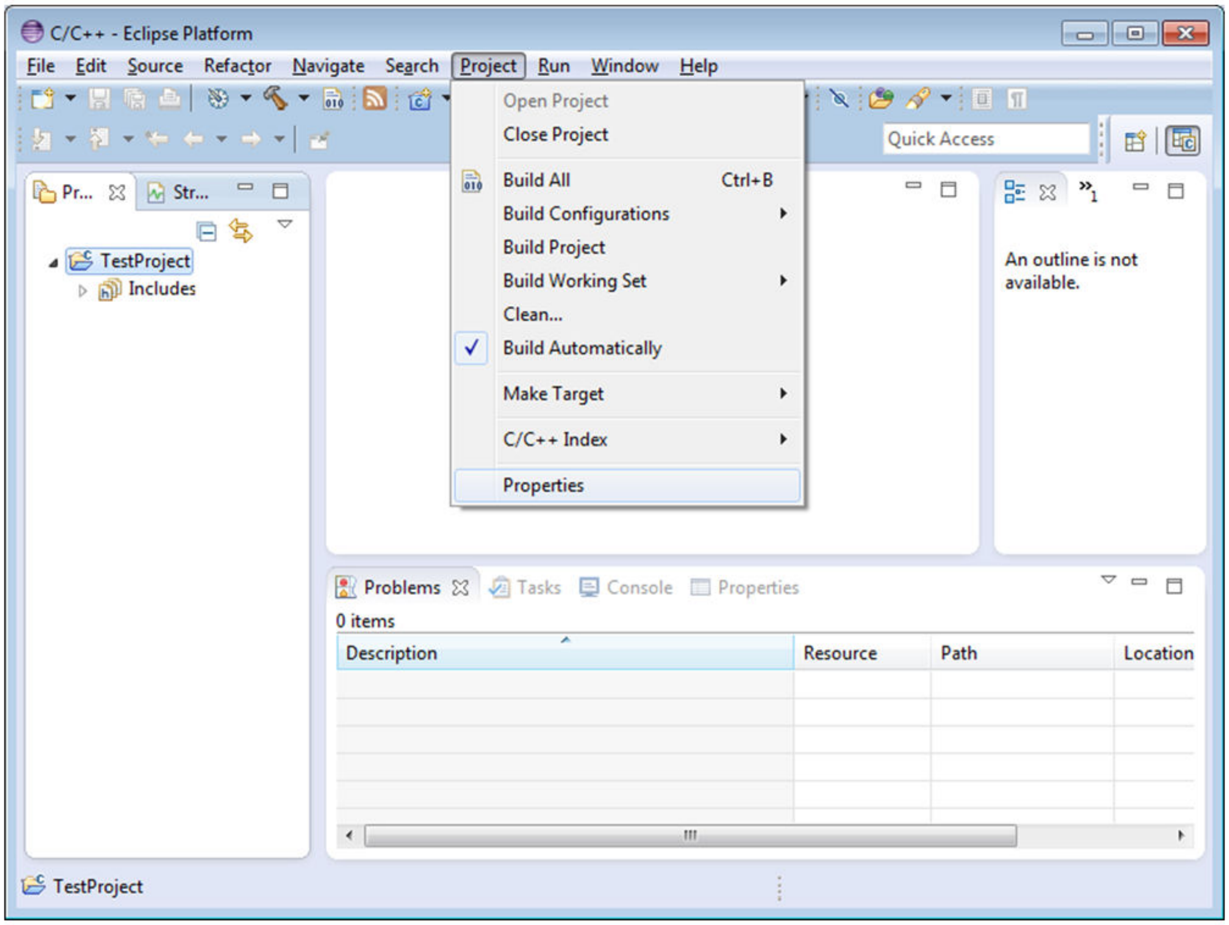
Toolchains:
- ARM Compiler 5 (DS-5 built-in)
- ARM Compiler 6 (DS-5 built-in)
- Altera Baremetal GCC
- Cygwin GCC
- GCC 4.x [arm-linux-gnueabihf] (DS-5 built-in)
- GCC for ARM Bare-metal
- MinGW GCC

☑ Show project types and toolchains only if they are supported on the platform

Build Settings

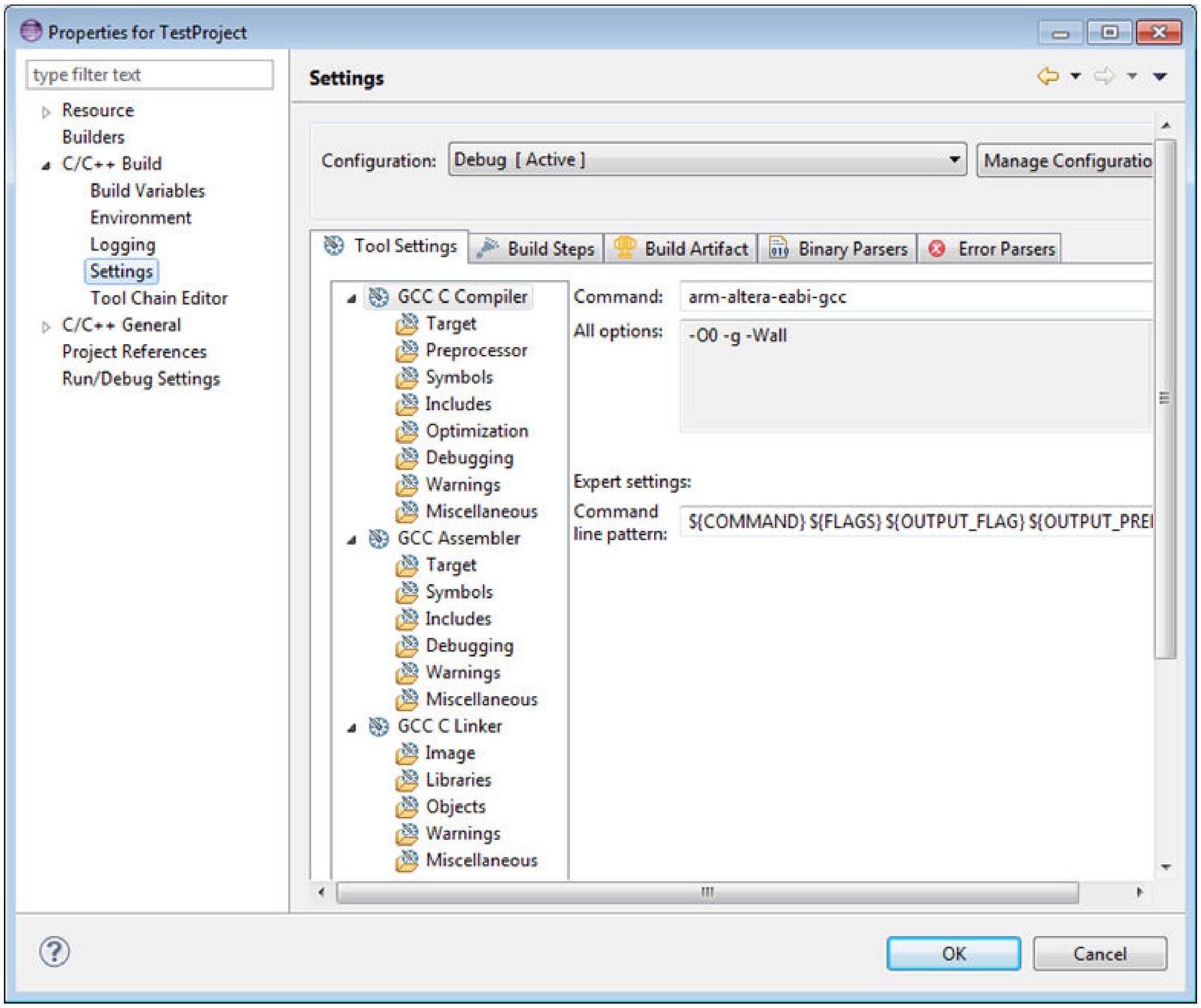
Once the project is created, the project properties can be accessed by going to **Project** > **Properties**.

Figure 6-6: Project Properties



Then, in the **Project Properites** window, the **Compilation** settings can be accessed by selecting **C/C++ Build** > **Settings**.

Figure 6-7: Project Settings



Configuration: Debug [Active]

Command: arm-altera-eabi-gcc

All options: -O0 -g -Wall

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PRE

The **Build Settings** incldue detailed settings for all tools:

- Compiler
- Assembler
- Linker

The *Getting Started Guides* chapter in this document contains a link to complete instructions on how to create a project from scratch, compile it and run it on an Altera SoC development board.

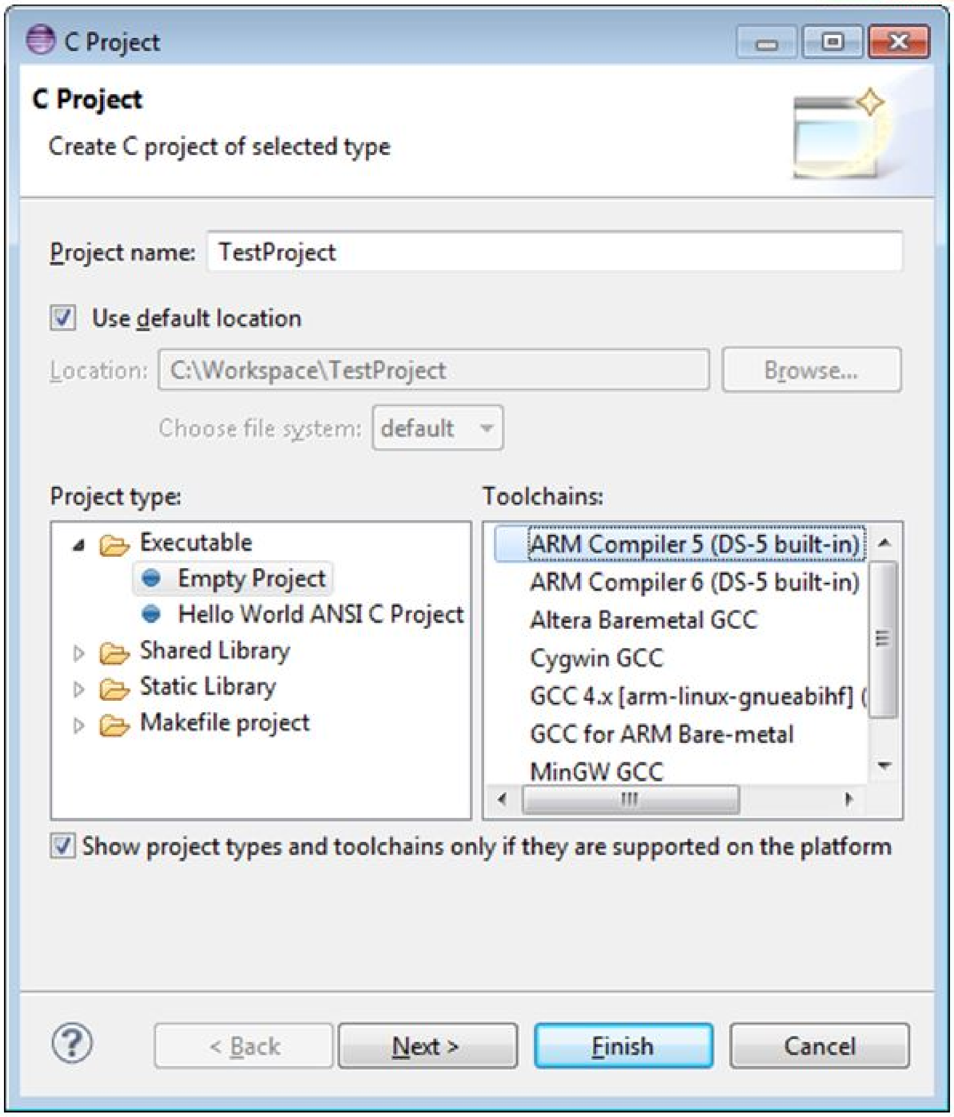
ARM Compiler Bare-Metal Project Management

This section shows ARM DS-5 can be used to manage ARM compiler projects in a GUI environment.

Creating a Project

1. Start Eclipse.
2. Go to **File** > **New C Project**.
3. Select one of the following options:
   a. Select "Project Type" as **Executable** > **Empty Project** and then for "Toolchains", select **ARM Compiler 5**. Click **Finish**.

Figure 6-8: Create an Empty ARM Compiler Bare-Metal Executable Project



Project name: TestProject

[✓] Use default location

Location: C:\Workspace\TestProject Browse...

Choose file system: default

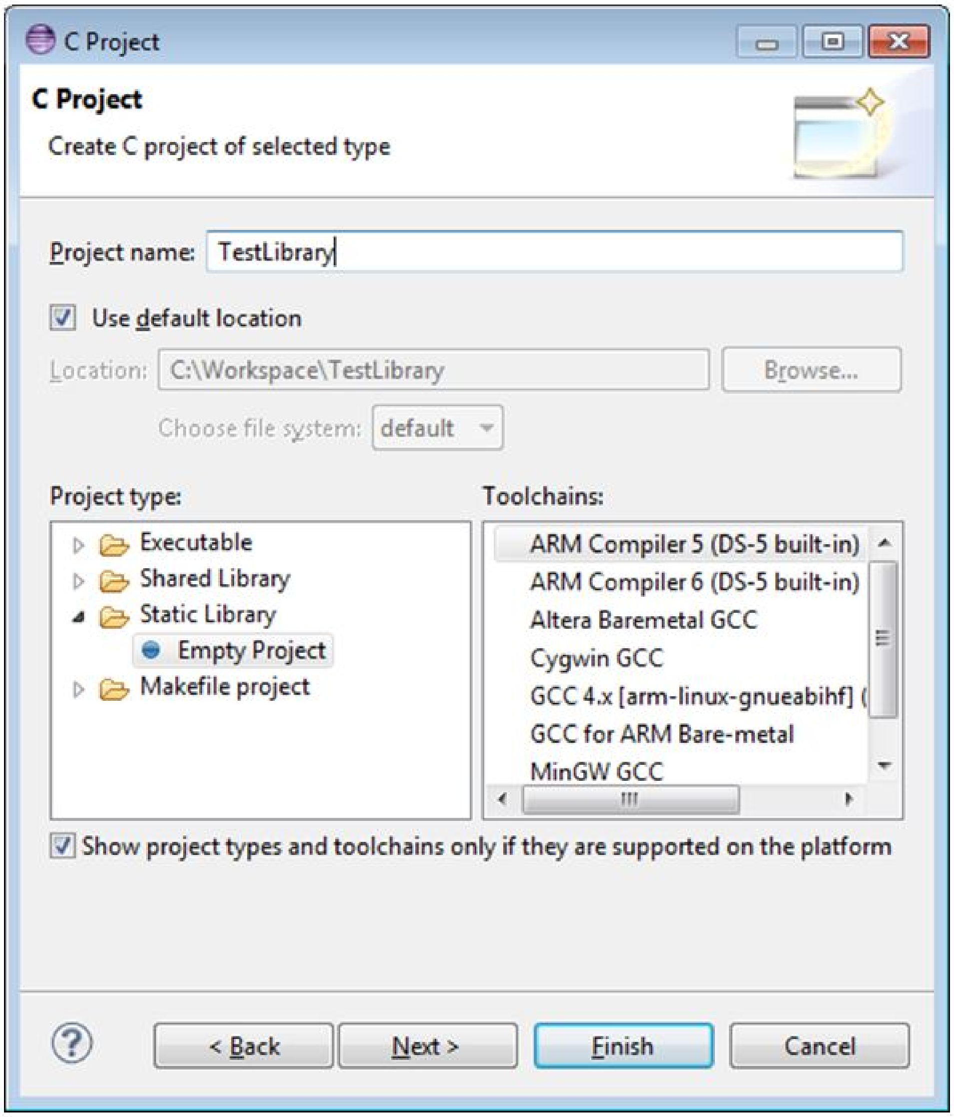
Project type:
- Executable
  - Empty Project
  - Hello World ANSI C Project
- Shared Library
- Static Library
- Makefile project

Toolchains:
- ARM Compiler 5 (DS-5 built-in)
- ARM Compiler 6 (DS-5 built-in)
- Altera Baremetal GCC
- Cygwin GCC
- GCC 4.x [arm-linux-gnueabihf] (
- GCC for ARM Bare-metal
- MinGW GCC

[✓] Show project types and toolchains only if they are supported on the platform

   b. Select **Static Library** > **Empty Project** and then for "Toolchains", select **ARM Compiler 5**. Click **Finish**.

Figure 6-9: Create an Empty ARM Compiler Bare-metal Library Project



C Project

C Project

Create C project of selected type

Project name: TestLibrary

[✓] Use default location

Location: C:\Workspace\TestLibrary

Choose file system: default

Project type:

- Executable
- Shared Library
- Static Library
  - Empty Project
- Makefile project

Toolchains:

- ARM Compiler 5 (DS-5 built-in)
- ARM Compiler 6 (DS-5 built-in)
- Altera Baremetal GCC
- Cygwin GCC
- GCC 4.x [arm-linux-gnueabihf] (
- GCC for ARM Bare-metal
- MinGW GCC

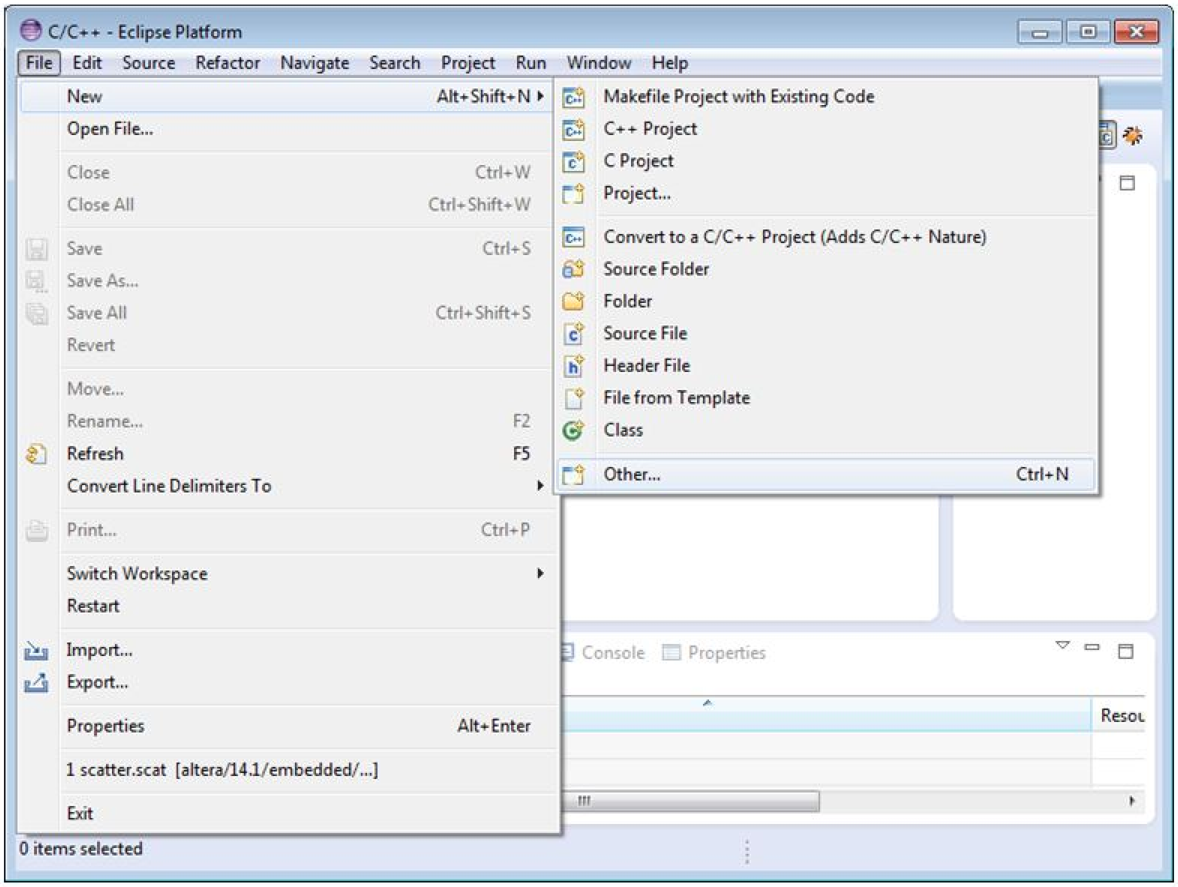
[✓] Show project types and toolchains only if they are supported on the platform

Linker Script

ARM DS-5 AE offers a visual tool to help create linker scripts.

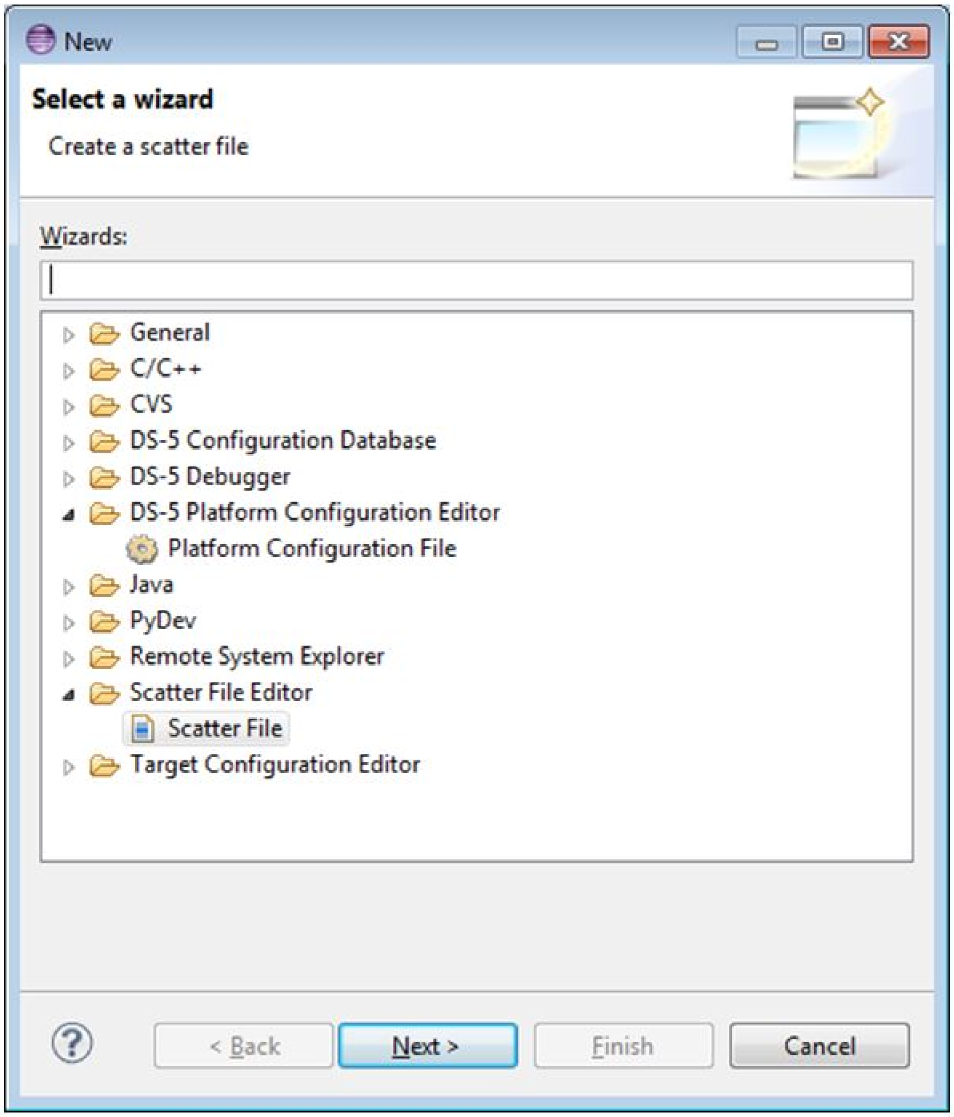
1. Go to **File** > **New** > **Other...**

Figure 6-10: Creating a Linker Script



2. Select **Scatter File Editor** > **Scatter File** and press **Next**.

Figure 6-11: Creating a Scatter File



Select a wizard

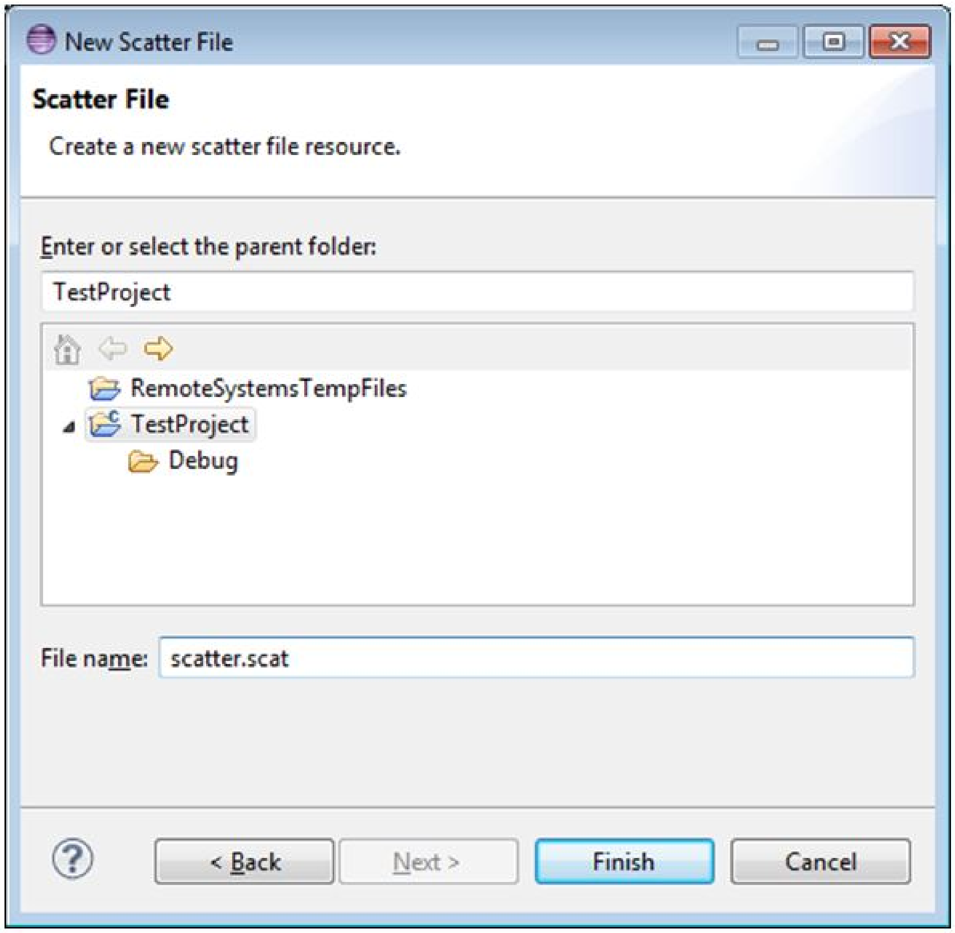
Create a scatter file

Wizards:

- General
- C/C++
- CVS
- DS-5 Configuration Database
- DS-5 Debugger
- DS-5 Platform Configuration Editor
  - Platform Configuration File
- Java
- PyDev
- Remote System Explorer
- Scatter File Editor
  - Scatter File
- Target Configuration Editor

3. Select the location of the new file, type in the file name and press **Finish**.

Figure 6-12: Create a New Scatter File Resource


New Scatter File

Scatter File

Create a new scatter file resource.

Enter or select the parent folder:

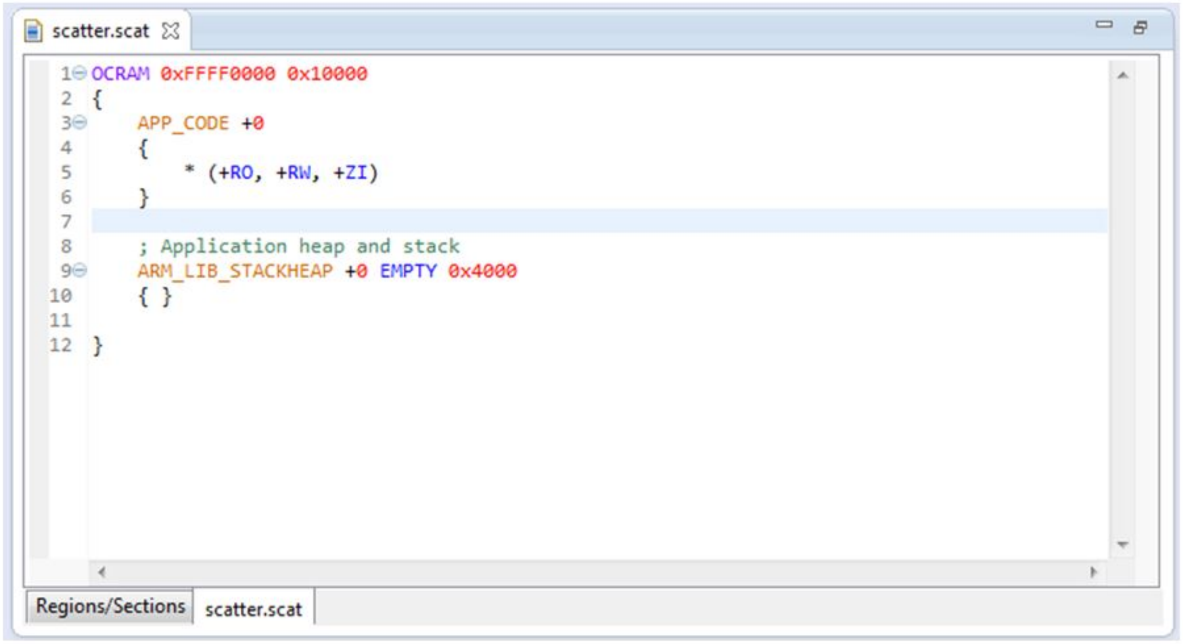
TestProject

- RemoteSystemsTempFiles
- TestProject
  - Debug

File name: scatter.scat

4. The linker script file can be edited directly as shown in the example below.

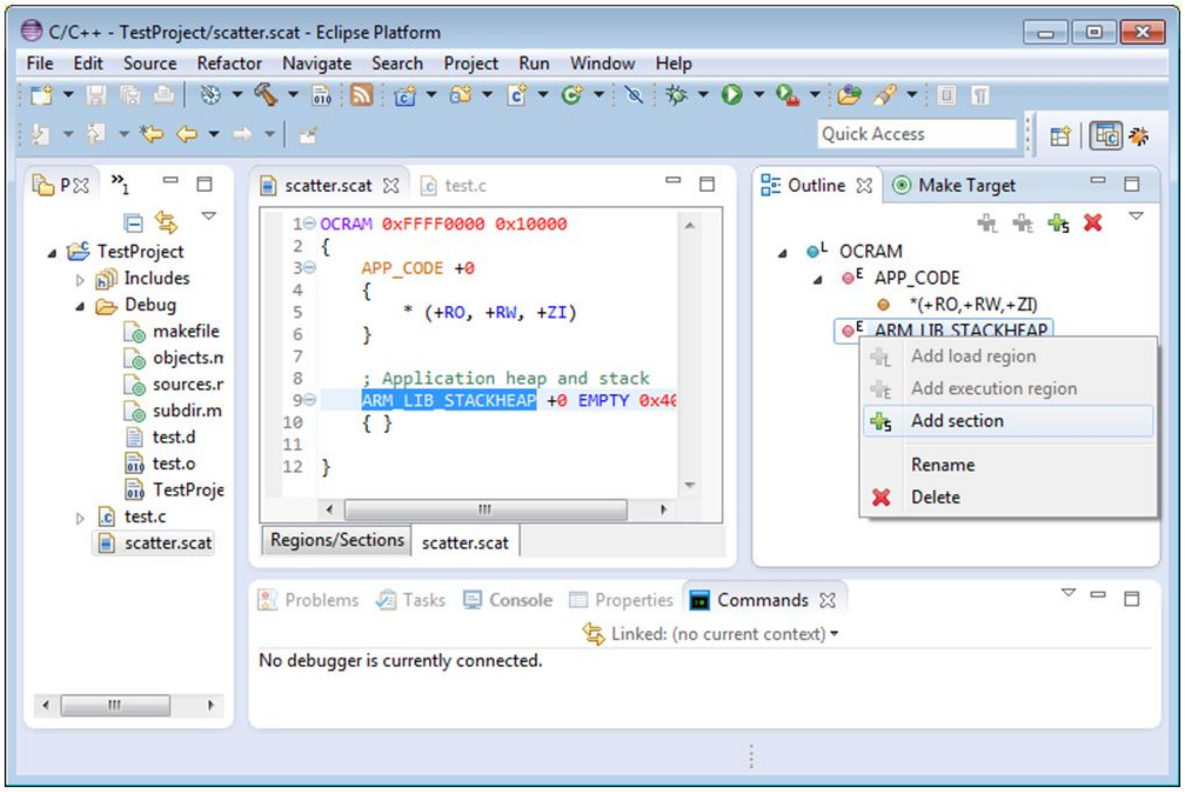
Figure 6-13: Linker Script Example



```
OCRAM 0xFFFF0000 0x10000
{
    APP_CODE +0
    {
        * (+RO, +RW, +ZI)
    }

    ; Application heap and stack
    ARM_LIB_STACKHEAP +0 EMPTY 0x4000
    { }

}
```

5. The file can also be edited by using the tools on the Outline view for the file.

Figure 6-14: Editing "Scatter.scat" File Using Tools on the Outline View



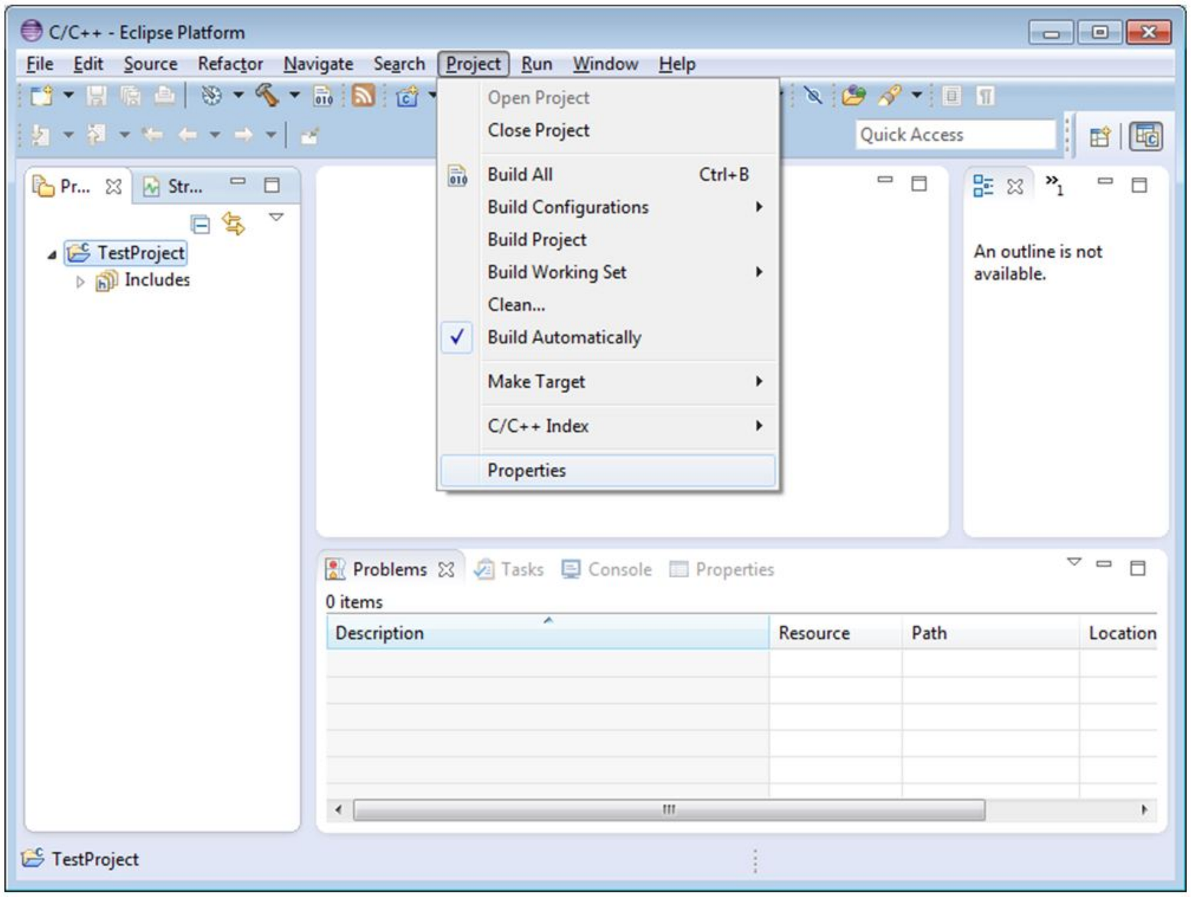
```
OCRAM 0xFFFF0000 0x10000
{
    APP_CODE +0
    {
        * (+RO, +RW, +ZI)
    }

    ; Application heap and stack
    ARM_LIB_STACKHEAP +0 EMPTY 0x4(
    { }

}
```

```
No debugger is currently connected.
```

Build Settings

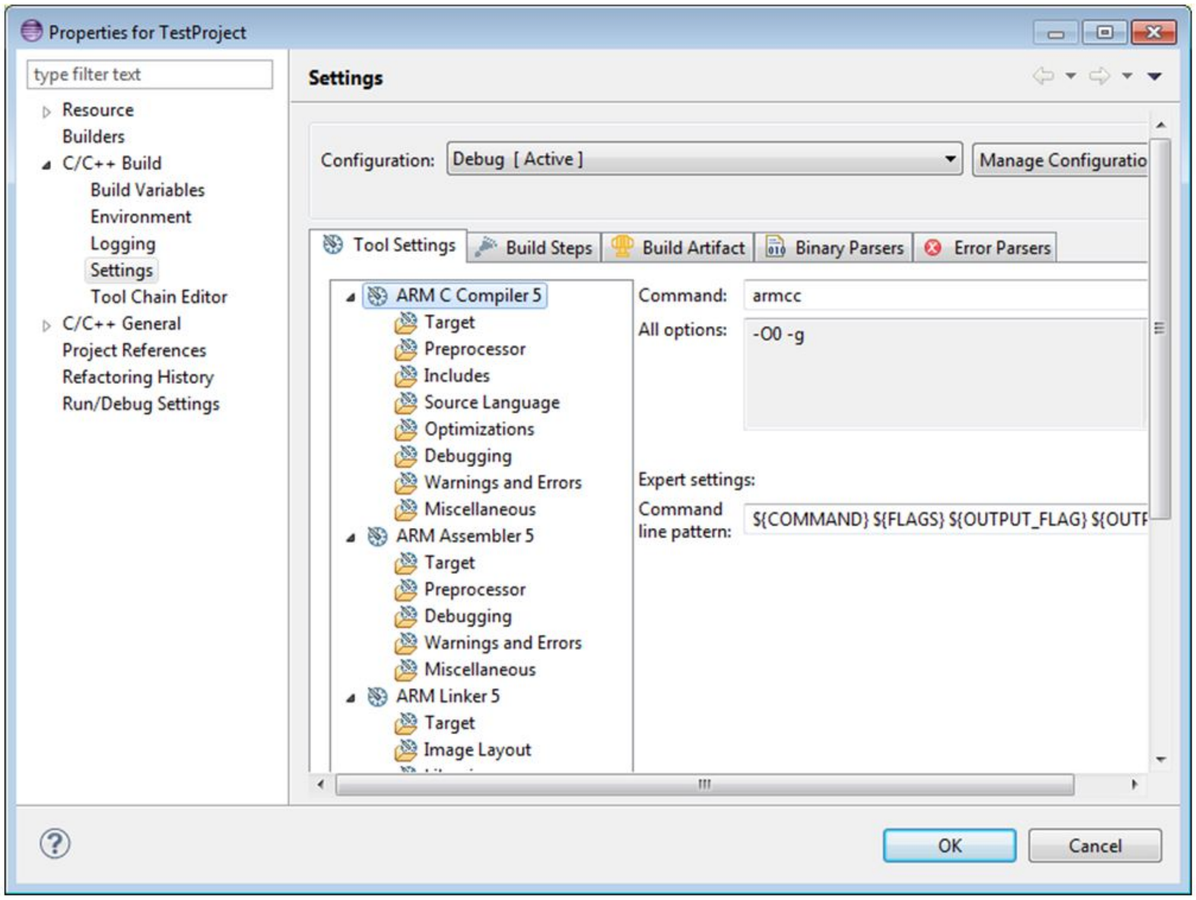
1. Once the project is created, the project properties can be accessed by going to **Project** > **Properties**.

Figure 6-15: Project Properties



2. Then, in the **Project Properties** window, the "Compilation" settings can be accessed by selecting **C/C++ Build** > **Settings**.

Figure 6-16: Project Settings



Properties for TestProject

Settings

Configuration: Debug [Active]

Tool Settings | Build Steps | Build Artifact | Binary Parsers | Error Parsers

- ARM C Compiler 5
  - Target
  - Preprocessor
  - Includes
  - Source Language
  - Optimizations
  - Debugging
  - Warnings and Errors
  - Miscellaneous
- ARM Assembler 5
  - Target
  - Preprocessor
  - Debugging
  - Warnings and Errors
  - Miscellaneous
- ARM Linker 5
  - Target
  - Image Layout

Command: armcc

All options: -O0 -g

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTP

The build settings include detailed settings for all tools:

- Compiler
- Assembler
- Linker

The "Getting Started with ARM Compiler Bare Metal Project Management" contains a link to complete instructions on how to create a project from scratch, compile it and run it on an Altera SoC development board.

Related Information

Getting Started Guides on page 5-1

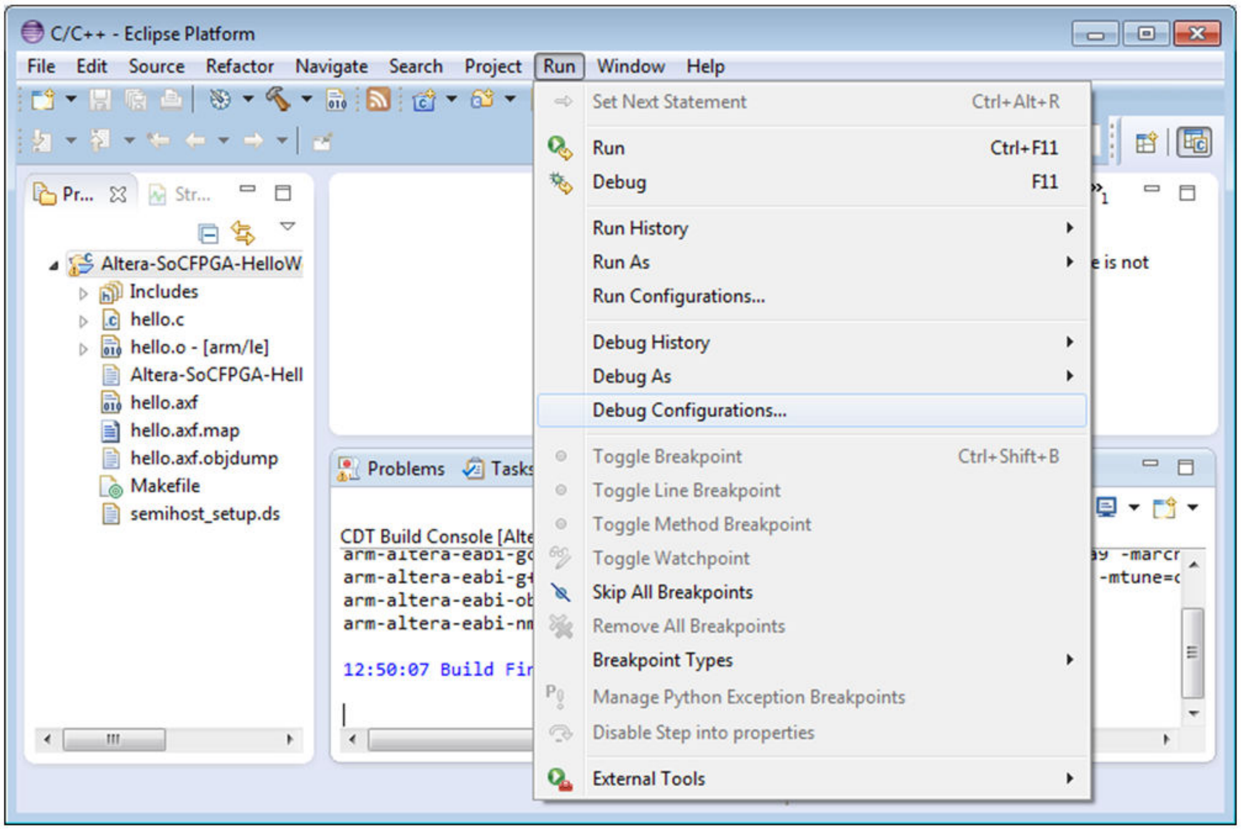
Debugging

The ARM DS-5 AE offers you a variety of debugging features.

Accessing Debug Configurations

The settings for a debugging session are stored in a **Debug Configuration**. The **Debug Configurations** window is accessible from the **Run** > **Debug Configurations** menu.

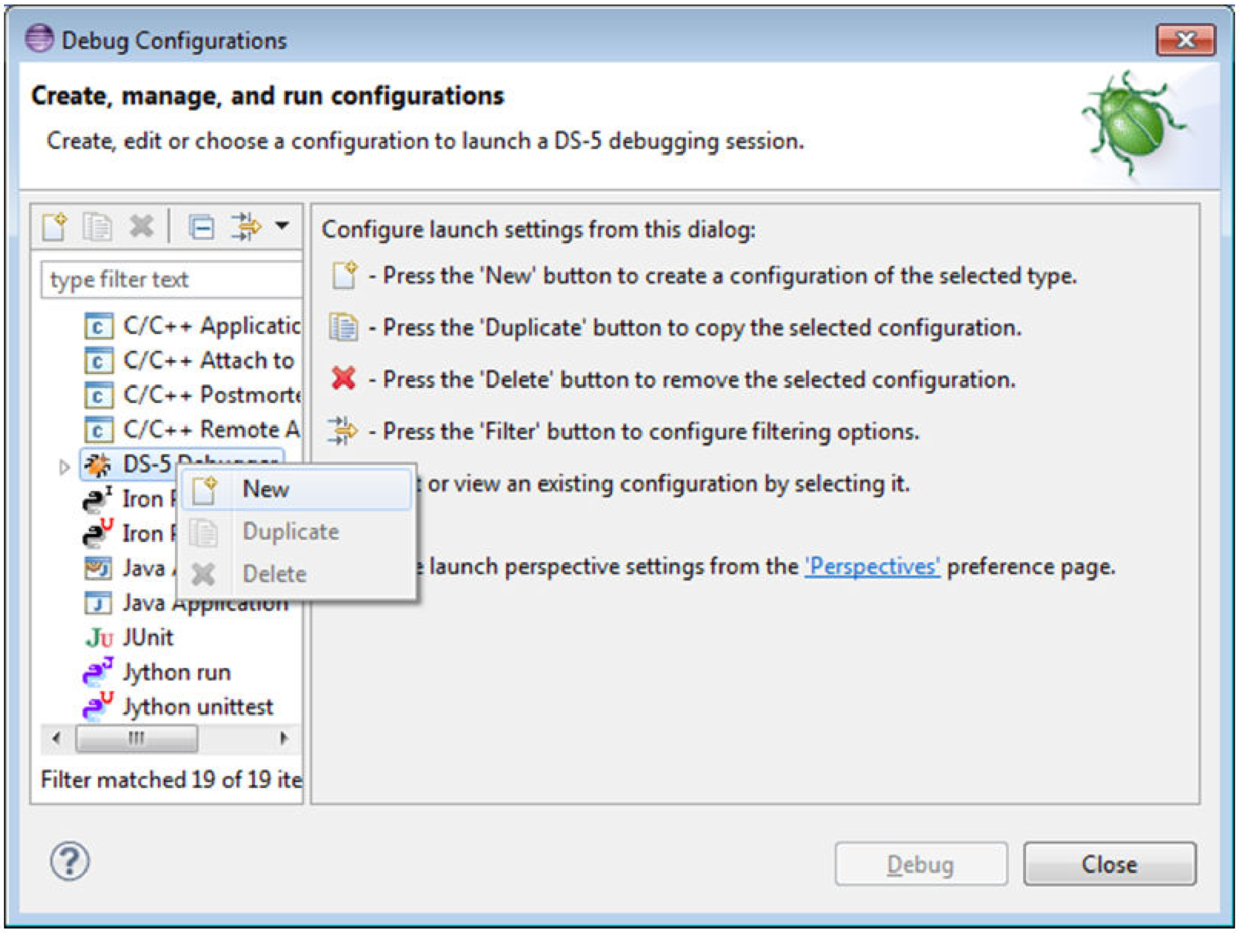
Figure 6-17: Accessing Debug Configurations



Creating a New Debug Configuration

A **Debug Configuration** is created in the **Debug Configurations** window by selecting **DS-5 Debugger** as the type of configuration in the left panel and then right-clicking with the mouse and selecting the **New** menu option.

Figure 6-18: Create New Debug Configuration


Debug Configurations

Create, manage, and run configurations

Create, edit or choose a configuration to launch a DS-5 debugging session.

type filter text

C/C++ Applicatic
C/C++ Attach to
C/C++ Postmorte
C/C++ Remote A
DS-5 Debugger
Iron
Iron
Java
Java Application
JUnit
Jython run
Jython unittest

Filter matched 19 of 19 ite

Configure launch settings from this dialog:

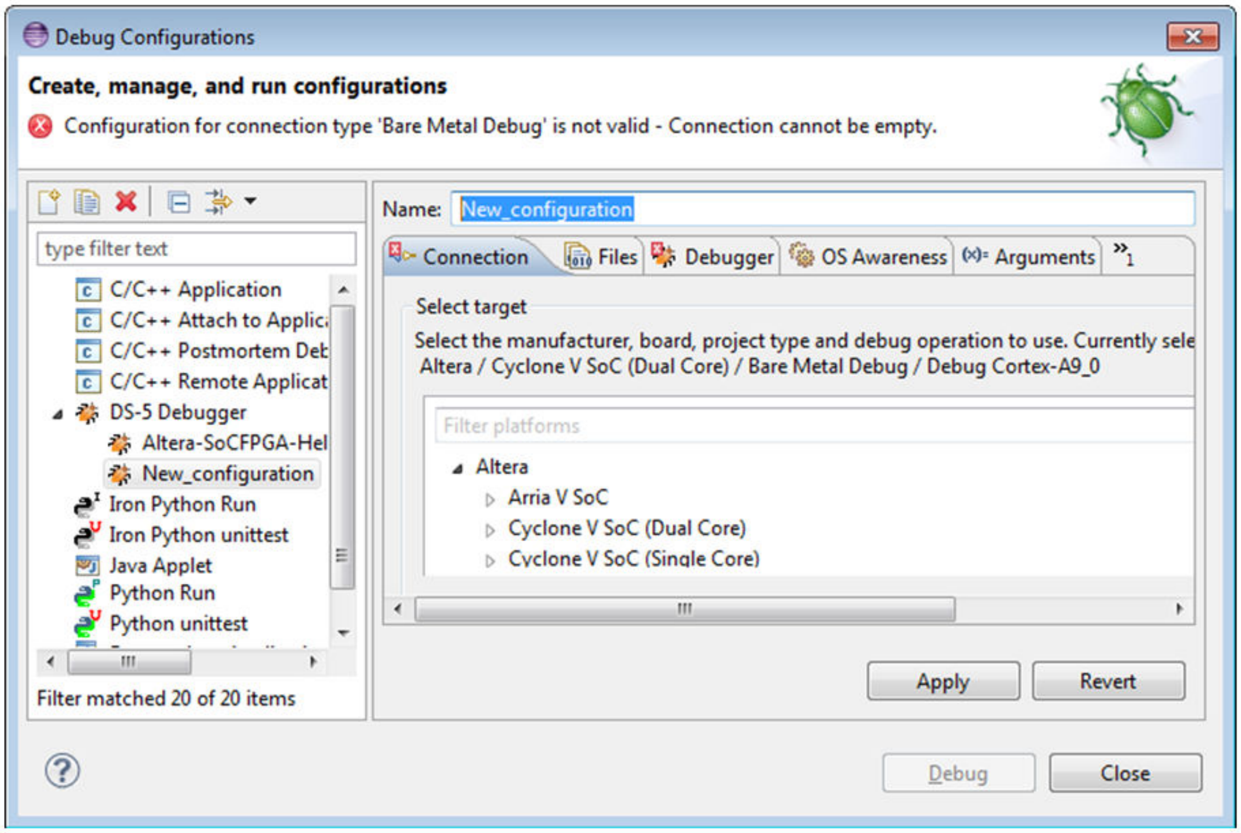
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.

or view an existing configuration by selecting it.

launch perspective settings from the 'Perspectives' preference page.

In the ARM DS-5 AE, you can assign a default name to the configuration, which you can edit.

Figure 6-19: Rename Debug Configuration


Debug Configurations
Create, manage, and run configurations
Configuration for connection type 'Bare Metal Debug' is not valid - Connection cannot be empty.
type filter text
C/C++ Application
C/C++ Attach to Applic
C/C++ Postmortem Del
C/C++ Remote Applicat
DS-5 Debugger
Altera-SoCFPGA-Hel
New_configuration
Iron Python Run
Iron Python unittest
Java Applet
Python Run
Python unittest
Filter matched 20 of 20 items
Name: New_configuration
Connection
Files
Debugger
OS Awareness
Arguments
Select target
Select the manufacturer, board, project type and debug operation to use. Currently sele
Altera / Cyclone V SoC (Dual Core) / Bare Metal Debug / Debug Cortex-A9_0
Filter platforms
Altera
Arria V SoC
Cyclone V SoC (Dual Core)
Cyclone V SoC (Single Core)
Apply
Revert
Debug
Close

Debug Configuration Options

This section lists the **Debug Configuration** options, which allows you to specify the desired debugging options for a project:

- Connection Options
- File Options
- Debugger Options
- RTOS Awareness
- Arguments
- Environment
- Event Viewer

Related Information

- **Getting Started Guides**
  For examples on how to use the ARM DS-5 AE debugging features.
- **Online ARM DS-5 Documentation**
  Refer to the DS-5 reference documentation for the complete details.

Connection Options

The **Connection** tab allows the user to select the desired target. The following targets are available for the Altera platforms:

Arria 10 SoC:

- Bare Metal Debug
    - Debug Cortex-A9_0
    - Debug Cortex-A9_1
    - Debug Cortex-A9x2 SMP
- Linux Application Debug
    - Connect to already running **gdbserver**
    - Download and debug application
    - Start **gdbserver** and debug target resident application
- Linux Kernel and/or Device Driver Debug
    - Debug Cortex-A9_0
    - Debug Cortex-A9_1
    - Debug Cortex-A9x2 SMP

Arria V SoC:

- Bare Metal Debug
    - Debug Cortex-A9_0
    - Debug Cortex-A9_1
    - Debug Cortex-A9x2_SMP
- Linux Application Debug
    - Connect to already running **gdbserver**
    - Download and debug application
    - Start **dbgserver** and debug target resident application
- Linux Kernel and/or Device Driver Debug
    - Debug Cortex-A9_0
    - Debug Cortex-A9_1
    - Debug Cortex-A9x2_SMP

Cyclone V SoC (Single Core):

- Bare Metal Debug
    - Debug Cortex-A9_0
- Linux Application Debug
    - Connect to already running **gdbserver**
    - Download and debug application
    - Start **dbgserver** and debug target resident application
- Linux Kernel and/or Device Driver Debug
    - Debug Cortex-A9_0

Cyclone V SoC (Dual Core):

- Bare Metal Debug
    - Debug Cortex-A9_0
    - Debug Cortex-A9_1
    - Debug Cortex-A9x2_SMP
- Linux Application Debug
    - Connect to already running **gdbserver**
    - Download and debug application
    - Start **dbgserver** and debug target resident application
- Linux Kernel and/or Device Driver Debug
    - Debug Cortex-A9_0
    - Debug Cortex-A9_1
    - Debug Cortex-A9x2_SMP

Dual Arria V SoC (Two Dual Core SoCs):

- Bare Metal Debug
    - Debug HPS0 Cortex-A9_0
    - Debug HPS0 Cortex-A9_1
    - Debug HPS0 Cortex-A9x2_SMP
    - Debug HPS1 Cortex-A9_0
    - Debug HPS1 Cortex-A9_1
    - Debug HPS1 Cortex-A9x2_SMP
- Linux Application Debug
    - Connect to already running **gdbserver**
    - Download and debug application
    - Start **dbgserver** and debug target resident application
- Linux Kernel and/or Device Driver Debug
    - Debug HPS0 Cortex-A9_0
    - Debug HPS0 Cortex-A9_1
    - Debug HPS0 Cortex-A9x2_SMP
    - Debug HPS1 Cortex-A9_0
    - Debug HPS1 Cortex-A9_1
    - Debug HPS1 Cortex-A9x2_SMP

Dual Cyclone V SoC (Two Dual Core SoCs):

- Bare Metal Debug
    - Debug HPS0 Cortex-A9_0
    - Debug HPS0 Cortex-A9_1
    - Debug HPS0 Cortex-A9x2_SMP
    - Debug HPS1 Cortex-A9_0
    - Debug HPS1 Cortex-A9_1
    - Debug HPS1 Cortex-A9x2_SMP
- Linux Application Debug
    - Connect to already running **gdbserver**
    - Download and debug application
    - Start **dbgserver** and debug target resident application
- Linux Kernel and/or Device Driver Debug
    - Debug HPS0 Cortex-A9_0
    - Debug HPS0 Cortex-A9_1
    - Debug HPS0 Cortex-A9x2_SMP
    - Debug HPS1 Cortex-A9_0
    - Debug HPS1 Cortex-A9_1
    - Debug HPS1 Cortex-A9x2_SMP

Android Application Debug:

- Native Application/Library Debug
    - APK Native Library Debug via `gdbserver`
        - Attach to a running Android application
        - Download and debug an Android application

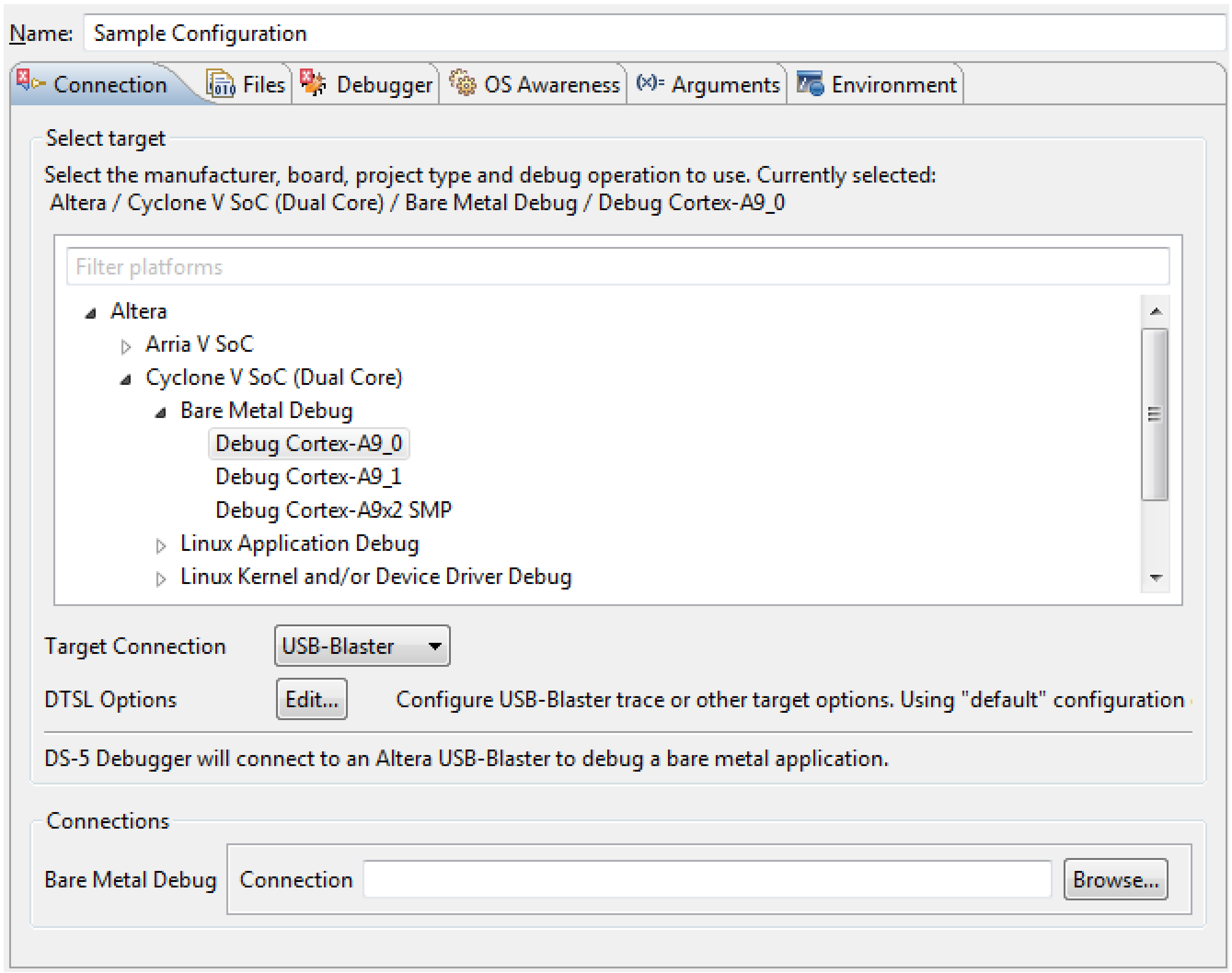
Linux Application Debug:

- Application Debug
    - Connections via `gdbserver`
        - Connect to already running **gdbserver**
        - Download and debug application
        - Start **dbgserver** and debug target resident application

Depending on the selected Target, the Connections panel will look different. For Bare Metal Debug and Linux Kernel and/or Device Driver Debug target types:

- A Target Connection option appears and it allows the user to select the type of connection to the target. Altera USB-Blaster and DSTREAM are two of the most common options.
- A DTSL option appears, allowing the user to configure the Debug and Traces Services Layer (detailed later).
- A Connections Browse button appears, allowing the user to browse and select either of the specific instances for the connection— Altera USB-Blaster or the DSTREAM instance.

Figure 6-20: Connection Options for Bare-metal and Linux Kernel and/or Device Driver Debug


Name: Sample Configuration

Select target

Select the manufacturer, board, project type and debug operation to use. Currently selected:
Altera / Cyclone V SoC (Dual Core) / Bare Metal Debug / Debug Cortex-A9_0

Filter platforms

- Altera
  - Arria V SoC
  - Cyclone V SoC (Dual Core)
    - Bare Metal Debug
      - Debug Cortex-A9_0
      - Debug Cortex-A9_1
      - Debug Cortex-A9x2 SMP
    - Linux Application Debug
    - Linux Kernel and/or Device Driver Debug

Target Connection: USB-Blaster

DTSL Options: Edit... Configure USB-Blaster trace or other target options. Using "default" configuration

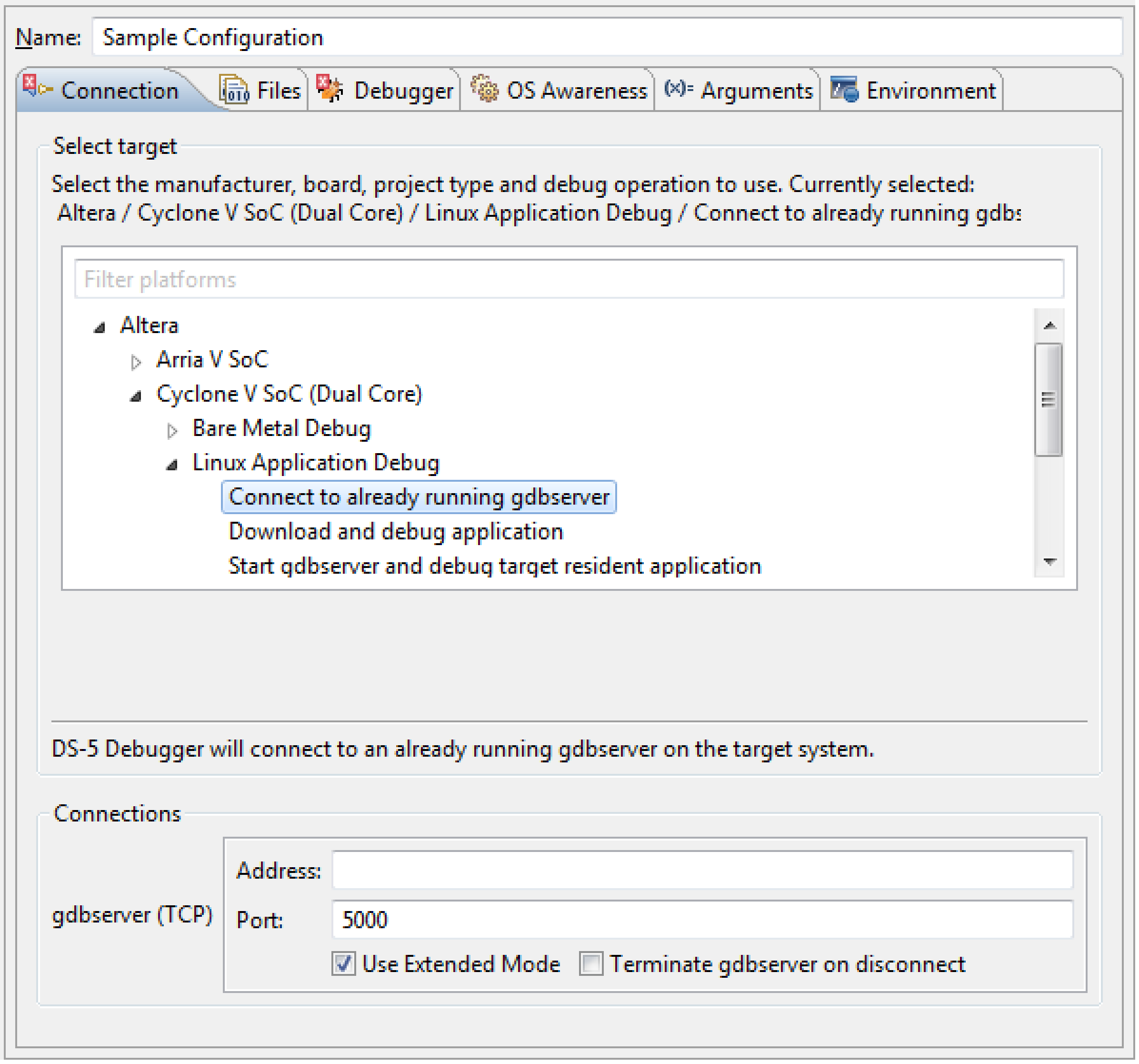
DS-5 Debugger will connect to an Altera USB-Blaster to debug a bare metal application.

Connections

Bare Metal Debug: Connection [ ] Browse...

For the **Linux Application Debug** targets, the connection parameters will be different depending on which type of connection was selected. The following two pictures illustrate the options.

Figure 6-21: Linux Application Debugging – Connect to a Running GDB Server



Name: Sample Configuration

Select target

Select the manufacturer, board, project type and debug operation to use. Currently selected:
Altera / Cyclone V SoC (Dual Core) / Linux Application Debug / Connect to already running gdbs

- Altera
  - Arria V SoC
  - Cyclone V SoC (Dual Core)
    - Bare Metal Debug
    - Linux Application Debug
      - Connect to already running gdbserver
      - Download and debug application
      - Start gdbserver and debug target resident application

DS-5 Debugger will connect to an already running gdbserver on the target system.

Connections

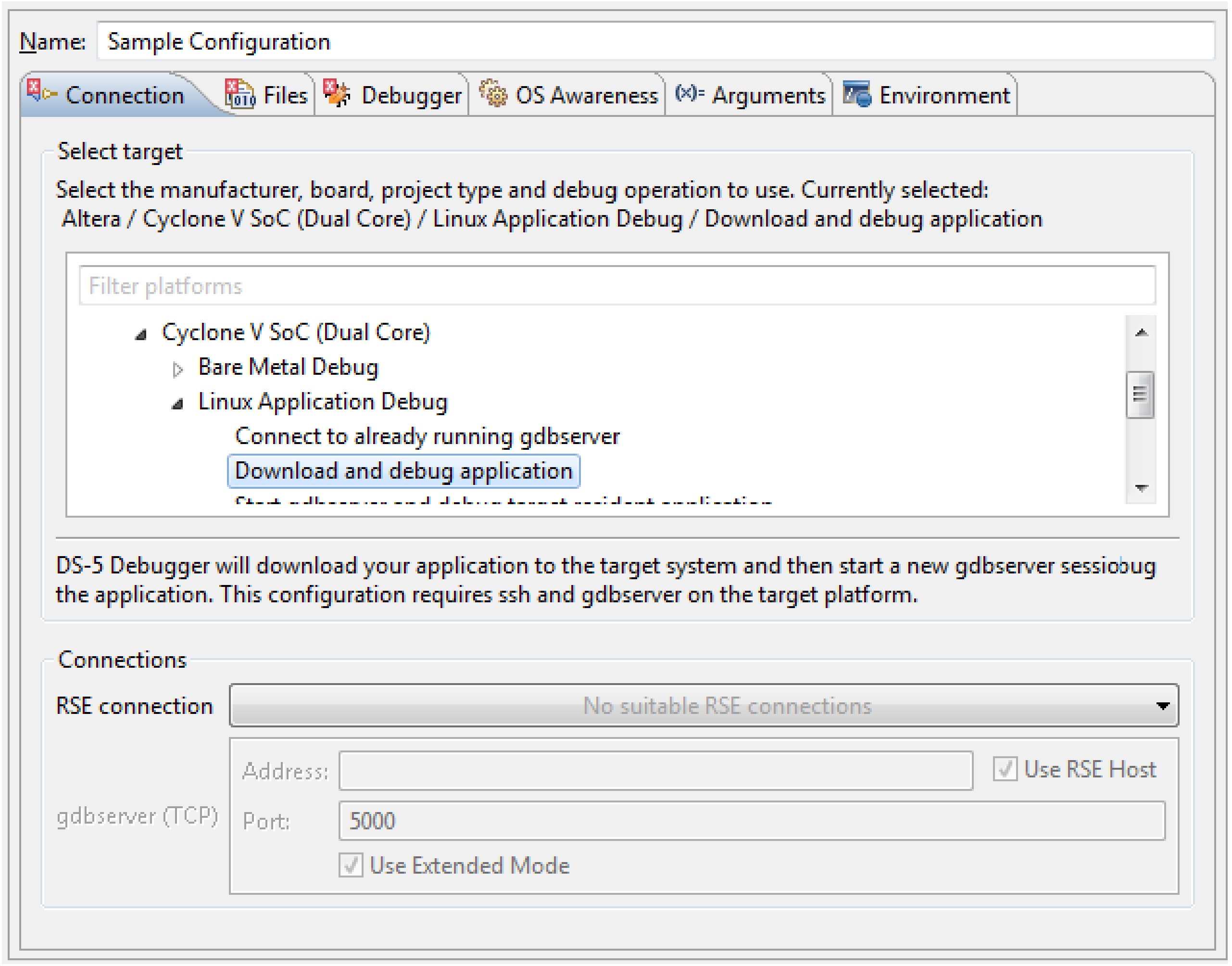
gdbserver (TCP)

Address:

Port: 5000

[✓] Use Extended Mode [ ] Terminate gdbserver on disconnect

Figure 6-22: Linux Application Debugging – Download And Debug Application


Name: Sample Configuration

Connection | Files | Debugger | OS Awareness | Arguments | Environment

Select target

Select the manufacturer, board, project type and debug operation to use. Currently selected:
Altera / Cyclone V SoC (Dual Core) / Linux Application Debug / Download and debug application

Filter platforms

- Cyclone V SoC (Dual Core)
  - Bare Metal Debug
  - Linux Application Debug
    - Connect to already running gdbserver
    - Download and debug application

DS-5 Debugger will download your application to the target system and then start a new gdbserver sessiobug the application. This configuration requires ssh and gdbserver on the target platform.

Connections

RSE connection: No suitable RSE connections

gdbserver (TCP)
Address:
Use RSE Host ✓
Port: 5000
Use Extended Mode ✓

Note: For the **Linux Application Debug**, the **Connection** needs to be configured in the **Remote System Explorer** view, as shown in *Getting Started with Linux Application Debugging*.

Related Information

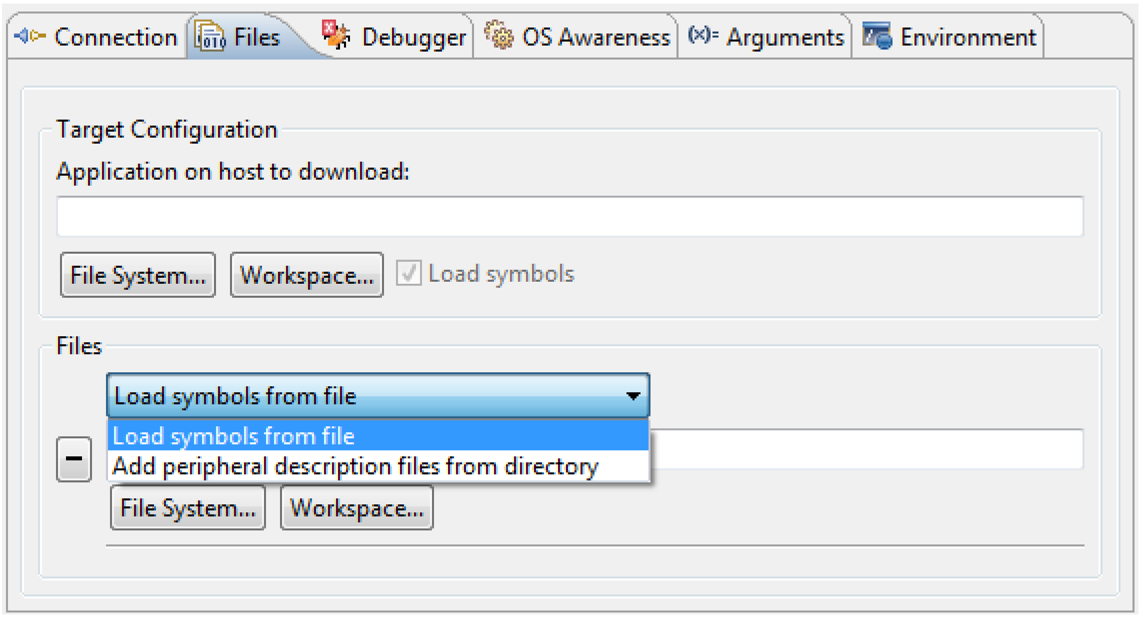
- **DTSL Options** on page 6-31
  For more information about the option on the Connections tab, refer to the DTSL Options section.
- **Debugger Options** on page 6-28
- **Getting Started Guides** on page 5-1

Files Options

The **Files** tab allows the following settings to be configured:

- **Application on host to download** – the file name of the application to be downloaded to the target. It can be entered directly in the edit box or it can be browsed for in the **Workspace** or on the **File System**.
- **Files** – contains a set of files. A file can be added to the set using the "+" button, and files can be removed from the set using the "–" button. Each file can be one of the following two types:
  - **Load symbols from file** – the debugger will use that file to load symbols from it,
  - **Add peripheral description files from directory** – the debugger to load peripheral register descriptions from the .SVD files stored in that directory. The SVD file is a result of the compilation of the hardware project.

Figure 6-23: Files Settings

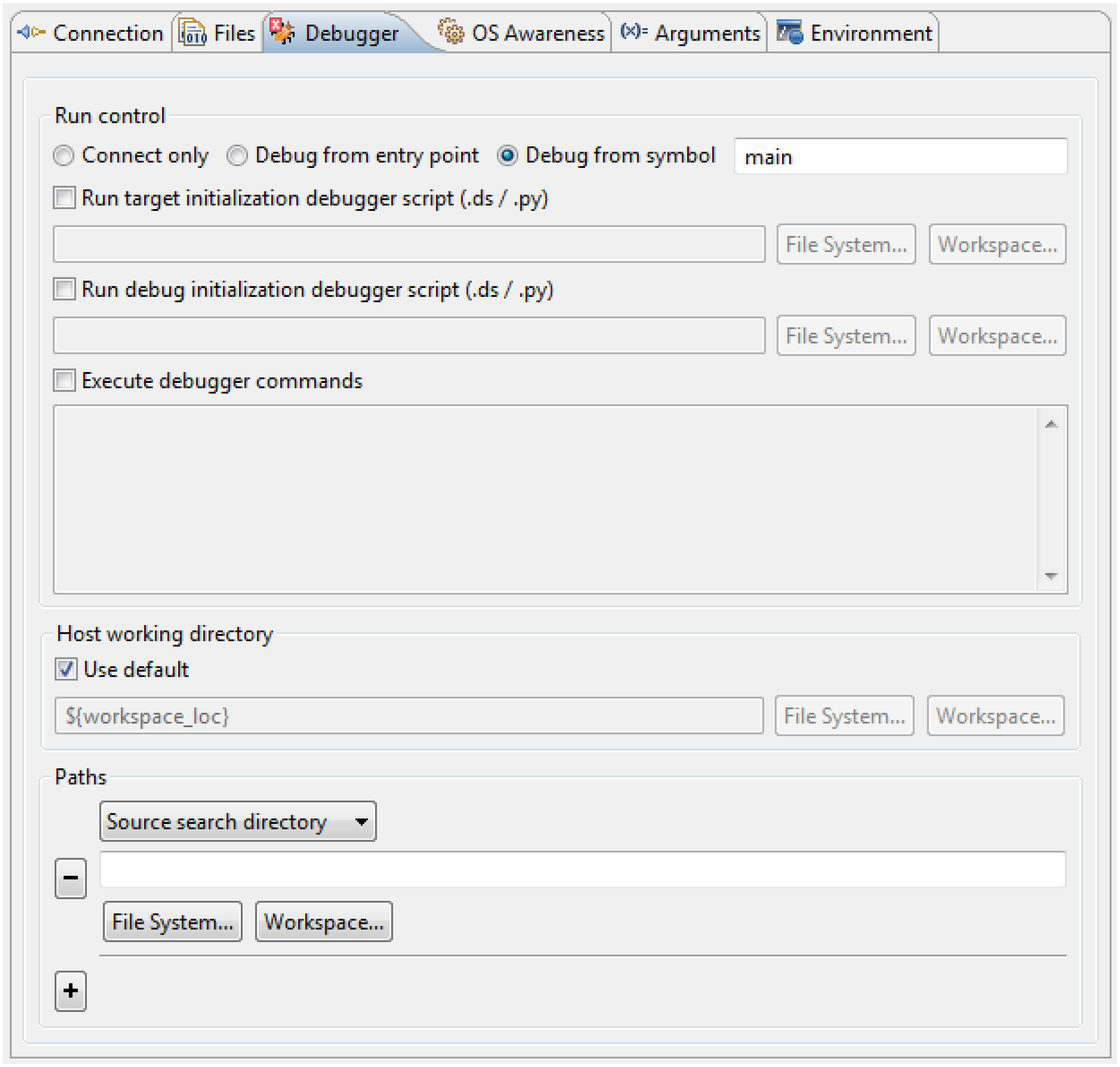


Debugger Options

The **Debugger** tab offers the following configurable options

- **Run Control** Options
  - Option to connect only, debug from entry point or debug from user-defined symbol,
  - Option to run user-specified target initialization script,
  - Option to run user-specified debug initialization script,
  - Option to execute user-defined debugger commands
- **Host working directory** – used by semihosting
- **Paths** – allows the user to enter multiple paths for the debugger to search for sources. Paths can be added with "+" button and removed with "-" button.

Figure 6-24: Debugger Settings



Run control

Connect only [✓ Debug from symbol] Debug from entry point: main

☐ Run target initialization debugger script (.ds / .py)

☐ Run debug initialization debugger script (.ds / .py)

☐ Execute debugger commands

Host working directory

☑ Use default

${workspace_loc}

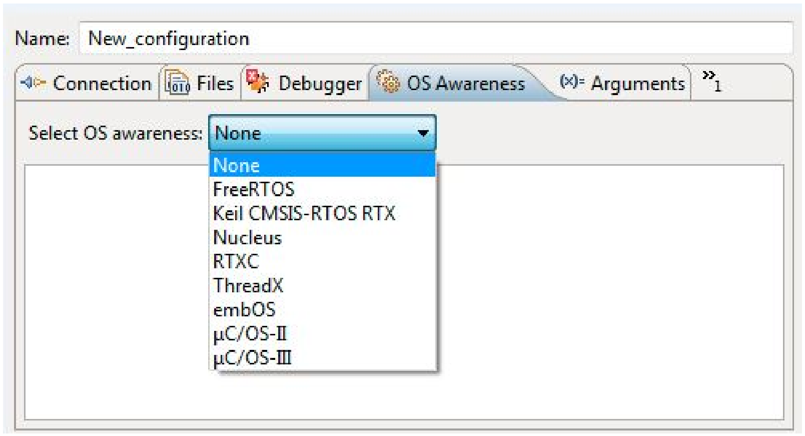
Paths

Source search directory

RTOS Awareness

The **RTOS Awareness** tab allows the user to enable RTOS awareness for the debugger.

Figure 6-25: RTOS Awareness Settings


Name: New_configuration

Select OS awareness: None

Related Information

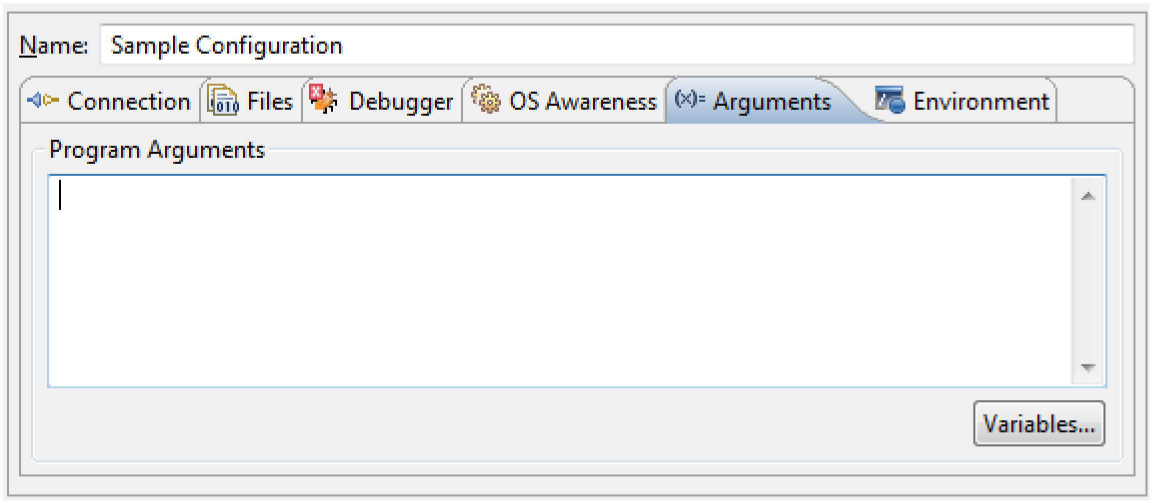
Keil Website

For more information about RTOS Awareness, refer to the Embedded Development Tools page on the KeilTM™ website.

Arguments

The **Arguments** tab allows the user to enter program arguments as text.

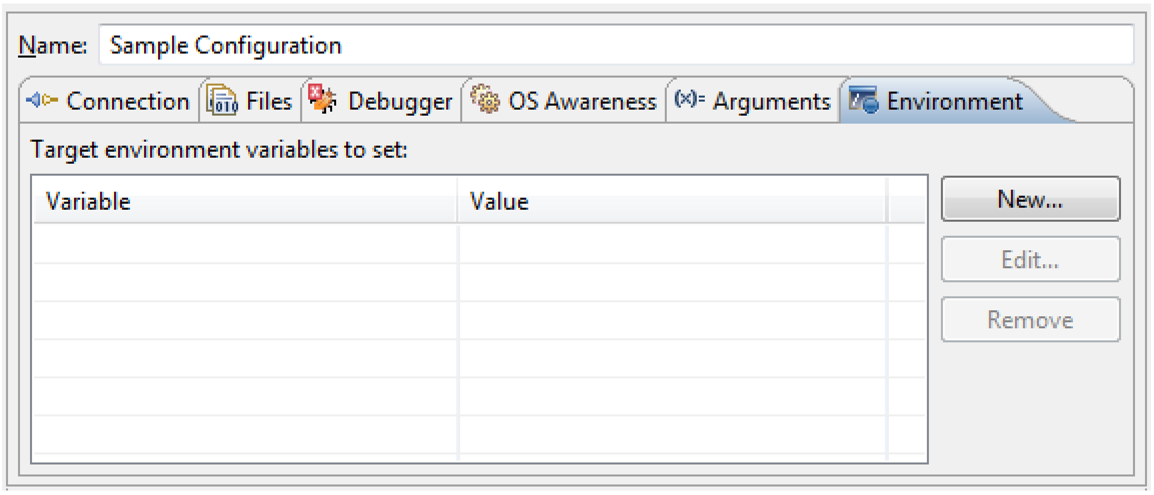
Figure 6-26: Arguments Settings


Name: Sample Configuration

Environment

The **Environment** tab allows the user to enter environment variables for the program to be executed.

Figure 6-27: Environment Settings


Name: Sample Configuration

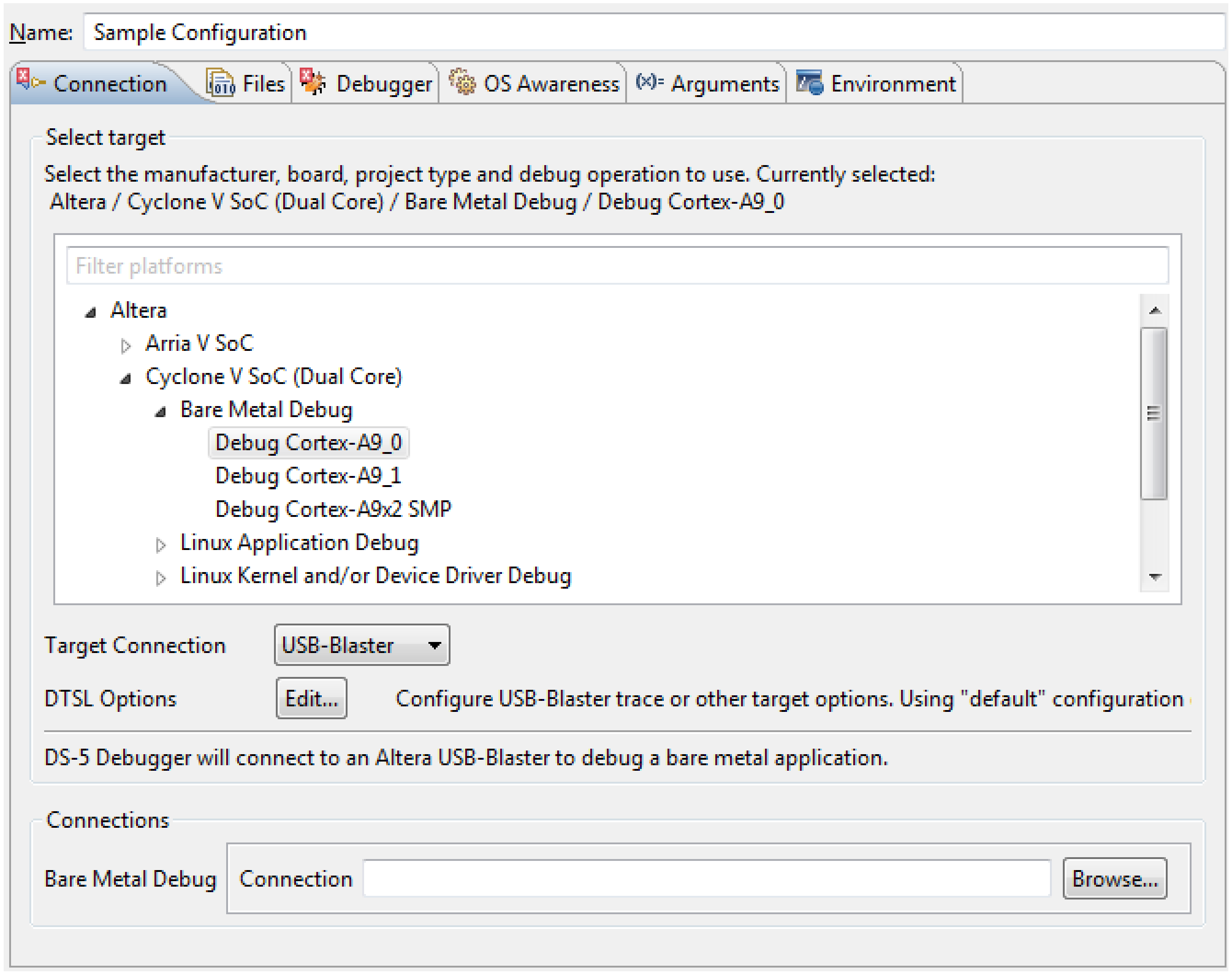
Target environment variables to set:

Variable	Value

DTSL Options

The Debug and Trace Services Layer (DTSL) provides tracing features. To configure trace options, in your project's **Debug Configuration** window, in the "Connection" tab, click the **Edit** button to open the **DTSL Configuration** window.

Figure 6-28: Debug Configurations - DTSL Options - Edit


Name: Sample Configuration

Connection | Files | Debugger | OS Awareness | Arguments | Environment

Select target

Select the manufacturer, board, project type and debug operation to use. Currently selected:
Altera / Cyclone V SoC (Dual Core) / Bare Metal Debug / Debug Cortex-A9_0

Filter platforms

- Altera
  - Arria V SoC
  - Cyclone V SoC (Dual Core)
    - Bare Metal Debug
      - Debug Cortex-A9_0
      - Debug Cortex-A9_1
      - Debug Cortex-A9x2 SMP
    - Linux Application Debug
    - Linux Kernel and/or Device Driver Debug

Target Connection: USB-Blaster

DTSL Options: Edit... Configure USB-Blaster trace or other target options. Using "default" configuration

DS-5 Debugger will connect to an Altera USB-Blaster to debug a bare metal application.

Connections

Bare Metal Debug | Connection | Browse...

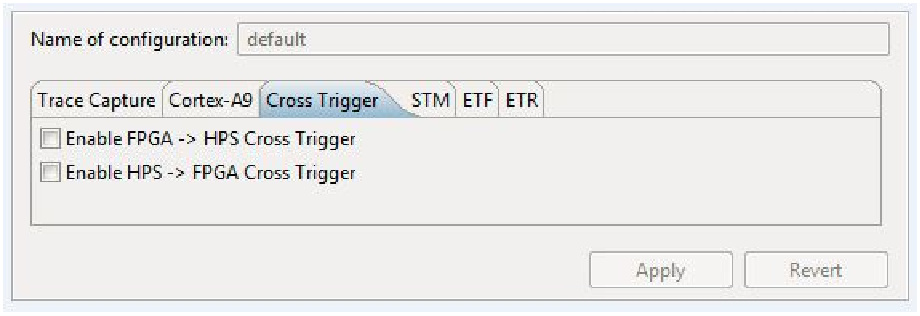
Cross Trigger Settings

The **Cross Trigger** tab allows the configuration of the cross triggering option of the SoC FPGA.

The following options are available:

- **Enable FPGA > HPS Cross Trigger** – for enabling triggers coming from FPGA to HPS
- **Enable HPS > FPGA Cross Trigger** – for enabling triggers coming from HPS to FPGA

Figure 6-29: DTSL Configuration Editor - Cross Trigger


Name of configuration: default

Tabs: Trace Capture | Cortex-A9 | Cross Trigger | STM | ETF | ETR

☐ Enable FPGA -> HPS Cross Trigger
☐ Enable HPS -> FPGA Cross Trigger

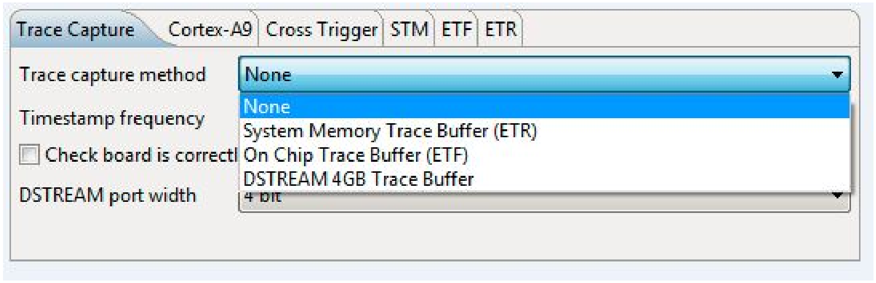
Trace Capture Settings

The **Trace Capture** tab allows the selection of the destination of the trace information. As mentioned in the introduction, the destination can be one of the following:

- **None** – meaning the tracing is disabled
- **ETR** – using any memory buffer accessible by HPS
- **ETF** – using the 32KB on-chip trace buffer
- **DSTREAM** – using the 4GB buffer located in the DSTREAM

The DSTREAM option is available only if the **Target** connection is selected as **DSTREAM** in the **Debug Configuration**.

Figure 6-30: DTSL Configuration Editor - Trace Capture > Trace Capture Method

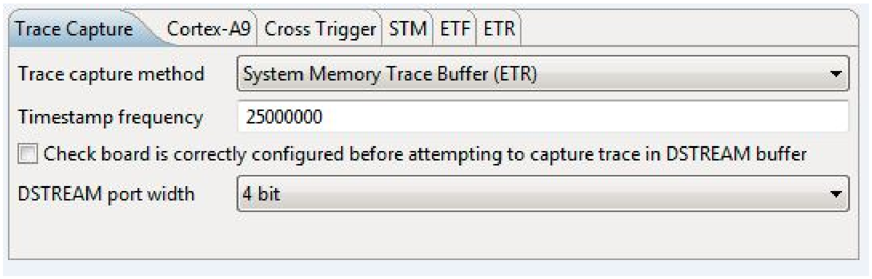

Tabs: Trace Capture | Cortex-A9 | Cross Trigger | STM | ETF | ETR

Trace capture method: None
Timestamp frequency
☐ Check board is correctl
DSTREAM port width: 4 bit

Dropdown options:
None
System Memory Trace Buffer (ETR)
On Chip Trace Buffer (ETF)
DSTREAM 4GB Trace Buffer

The **Trace Buffer** tab provides the option of selecting the timestamp frequency.

Figure 6-31: DTSL Configuration Editor - Trace Capture > Timestamp Frequency

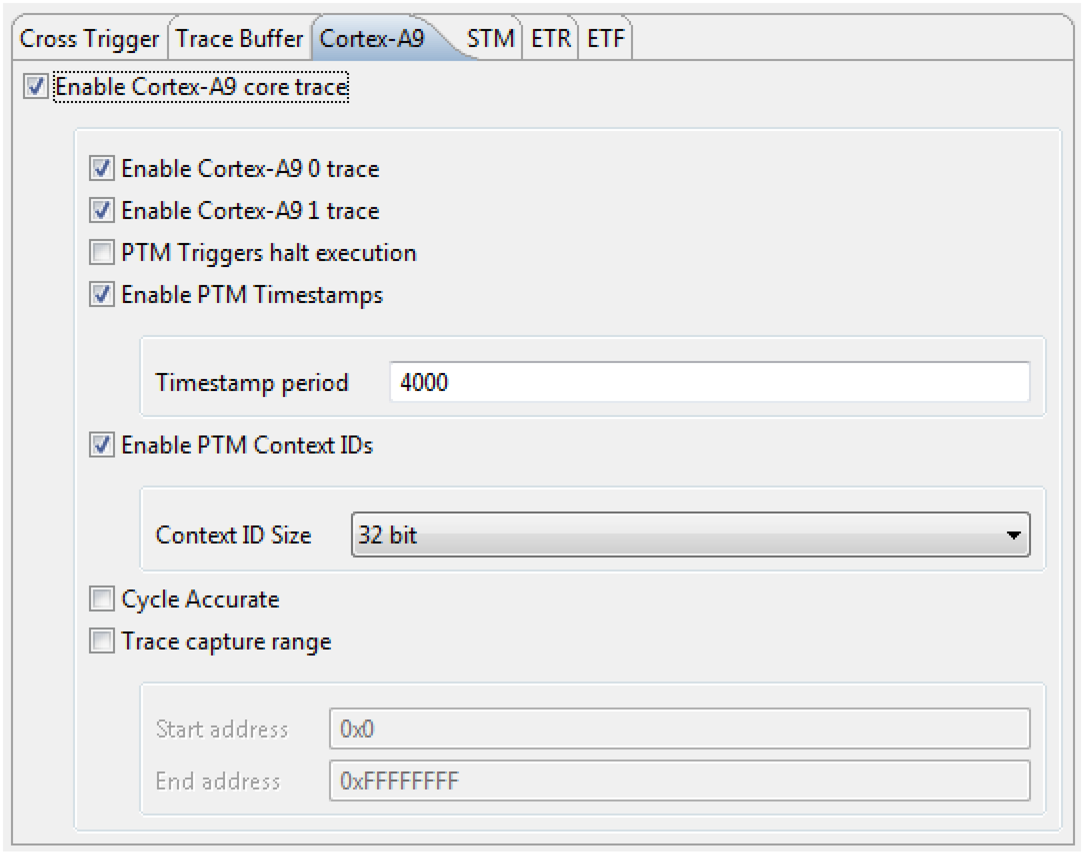

Tabs: Trace Capture | Cortex-A9 | Cross Trigger | STM | ETF | ETR

Trace capture method: System Memory Trace Buffer (ETR)
Timestamp frequency: 25000000
☐ Check board is correctly configured before attempting to capture trace in DSTREAM buffer
DSTREAM port width: 4 bit

Cortex-A9 Settings

The **Cortex-A9** tab allows the selection of the desired core tracing options.

Figure 6-32: DTSL Configuration Editor - Cortex-A9



[✓] Enable Cortex-A9 core trace
[✓] Enable Cortex-A9 0 trace
[✓] Enable Cortex-A9 1 trace
[ ] PTM Triggers halt execution
[✓] Enable PTM Timestamps
Timestamp period: 4000
[✓] Enable PTM Context IDs
Context ID Size: 32 bit
[ ] Cycle Accurate
[ ] Trace capture range
Start address: 0x0
End address: 0xFFFFFFFF

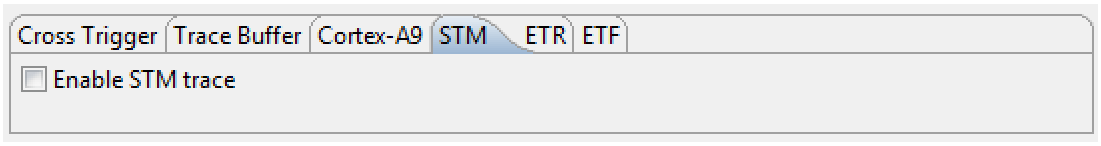
The following **Core Tracing Options** are available:

- **Enable Cortex-A9 0 core trace** – check to enable tracing for core #0
- **Enable Cortex-A9 1 core trace** – check to enable tracing for core #1
- **PTM Triggers halt execution** – check to cause the execution to halt when tracing
- **Enable PTM Timestamps** – check to enable time stamping
- **Enable PMT Context IDs** – check to enable the context IDs to be traced
- **Context ID Size** – select 8-, 16- or 32-bit context IDs. Used only if Context IDs are enabled
- **Cycle Accurate** – check to create cycle accurate tracing
- **Trace capture range** – check to enable tracing only a certain address interval
- **Start Address, End Address** – define the tracing address interval (Used only if the Trace Capture Range is enabled)

STM Settings

The **STM** tab allows you to configure the System Trace Macrocell (STM).

Figure 6-33: DTSL Configuration Editor - STM



☐ Enable STM trace

Only one option is available:

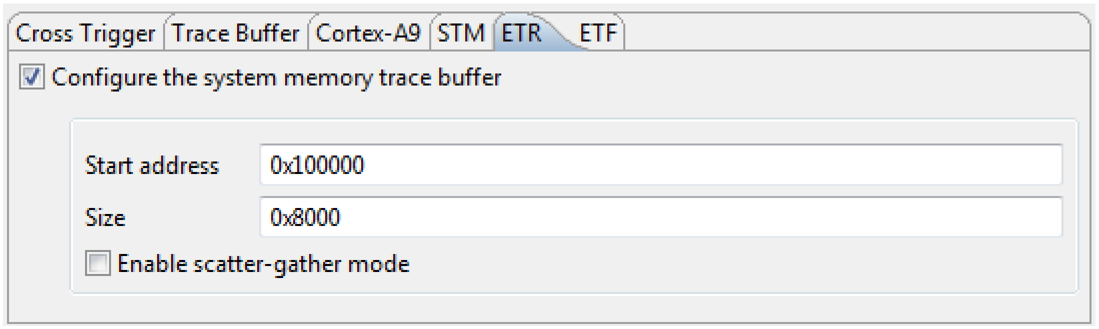
- **Enable STM Trace** – check to enable STM tracing.

ETR Settings

The **ETR** settings allow the configuration of the Embedded Trace Router (ETR) settings.

The Embedded Trace Router is used to direct the tracing information to a memory buffer accessible by HPS.

Figure 6-34: DTSL Configuration Editor - ETR



☑ Configure the system memory trace buffer

Start address 0x100000

Size 0x8000

☐ Enable scatter-gather mode

The following options are available:

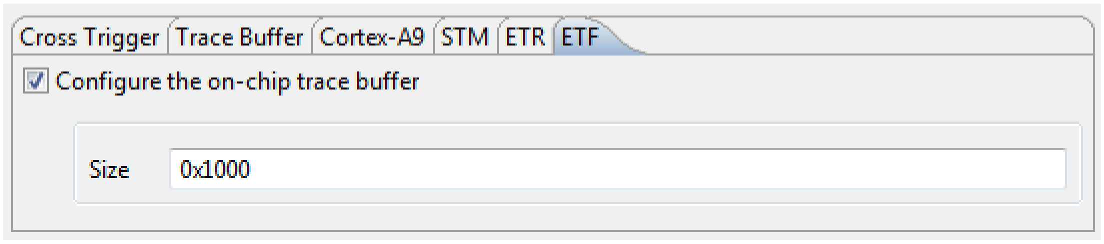
- **Configure the system memory trace buffer** – check this if the **ETR** is selected for trace destination on the **Trace Capture** tab.
- **Start Address, Size** – define the trace buffer location in system memory and its size.
- **Enable scatter-gather mode** – use when the OS cannot guarantee a contiguous piece of physical memory. The scatter-gather table is setup by the operating system using a device driver and is read automatically by the ETR.

ETF Settings

The **ETF** tab allows the configuration of the Embedded Trace FIFO (ETF) settings.

The Embedded Trace FIFO is a 32KB buffer residing on HPS that can be used to store tracing data to be retrieved by the debugger, but also as an elastic buffer for the scenarios where the tracing data is stored in memory through ETR or on the external DSTREAM device using TPIU.

Figure 6-35: DTSL Configuration Editor - ETF



☑ Configure the on-chip trace buffer

Size 0x1000

The following options are available:

- **Configure the on-chip trace buffer** – check this if ETF is selected for trace destination on the **Trace Capture** tab.
- **Size** – define the ETF size. The default size is set to 0x8000 (32KB).

ARM DS-5 AE Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Updated default size in ETF Settings.
August 2015	2015.08.06	Added Arria 10 support.

Boot Tools User Guide 7

2016.02.17

ug-1137 Subscribe Send Feedback

Introduction

The boot flow for all Altera SoC devices includes the second stage bootloader (SSBL). The SSBL is usually referred to as "preloader" in the context of Cyclone V and Arria V devices, and "bootloader" in the context of Arria 10 devices.

The SSBL is loaded by the boot ROM into the on-chip RAM (OCRAM) and has the task of bringing up the SDRAM, and loading and executing the next stage in the boot process.

Figure 7-1: Cyclone V and Arria V Typical Boot Flow

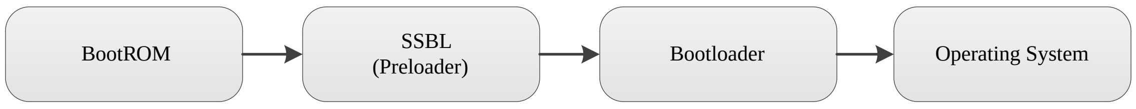
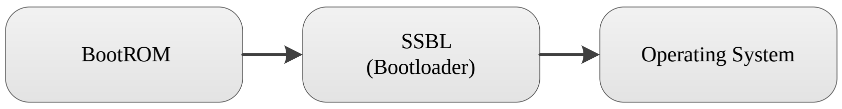


Figure 7-2: Arria 10 Typical Boot Flow



For Cyclone V and Arria V devices, the OCRAM size is 64 KB, which limits the SSBL size and typically requires an additional bootloader stage, which then is used to load an operating system.

For Arria 10 devices, there is no need for an additional bootloader stage, because the OCRAM size is 256 KB and the required functionality is included in the SSBL.

This chapter presents the tools that are used to enable the SSBL management:

- **SSBL Support Package Generator** – enables the user to create and manage the SSBL
- **SSBL Image Tool (mkpimage)** – enables to user to add the boot ROM-required header on top of the SSBL
- **U-Boot Image Tool (mkimage)**: enables the user to add the SSBL-required header on top of the files loaded by SSBL

Second Stage Bootloader Support Package Generator

The Second Stage Bootloader (SSBL) Support Package Generator provides an easy, safe, and reliable way to customize the SSBL for the Altera SoC devices.

The SSBL Support Package is referred to as “BSP” and the SSBL Support Package Generator is referred to as “BSP Generator” for the remainder of this document.

The BSP Generator allows you to perform the following tasks:

- Create a new BSP
- Report BSP settings
- Modify BSP settings
- Generate BSP files

The generated BSP includes a makefile, which can be used to build the corresponding bootable Preloader or Bootloader image.

The BSP Generator functionality can be accessed from a Graphical User Interface or by using a set of command line tools, which allow complete scripting of the flow.

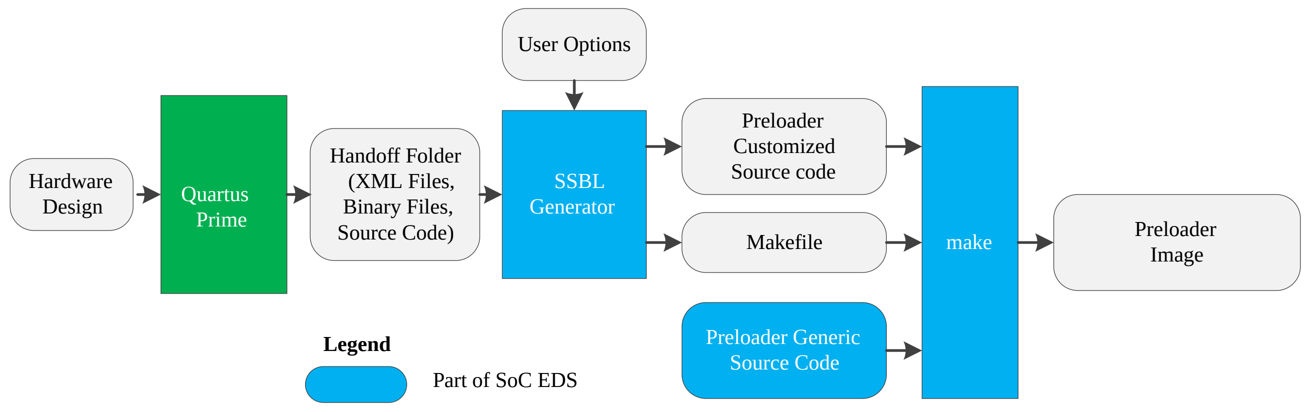
BSP Generation Flow

This section presents the BSP generation flow for both the Cyclone V and Arria V Preloader and Arria 10 Bootloader. While the flows are similar, there are some important differences.

Cyclone V and Arria V Flow

For Cyclone V and Arria V, the BSP Generator creates a customized BSP with preloader generic source files and board-specific SoC FPGA files. The generator consolidates the hardware settings and user inputs to create the BSP. The BSP files include a makefile to create the preloader image. The preloader image can then be downloaded to a Flash device or FPGA RAM to be used for booting HPS.

Figure 7-3: Arria V/ Cyclone V BSP Generator Flow


User Options
Hardware Design
Quartus Prime
Handoff Folder (XML Files, Binary Files, Source Code)
SSBL Generator
Preloader Customized Source code
Makefile
Preloader Generic Source Code
make
Preloader Image
Legend
Part of SoC EDS

The hardware handoff information contains various settings that the user entered when creating the hardware design in Qsys and Quartus Prime. These include the following:

- Pin-muxing for the HPS dedicated pins
- I/O settings for the HPS dedicated pins:
    - Voltage
    - Slew rate
    - Pull up/ down
- State of HPS peripherals:
    - Enabled
    - Disabled
- Configuration of the bridges between HPS and FPGA
- Clock tree settings:
    - PLL settings
    - Clock divider settings
    - Clock gatting settings
- DDR settings:
    - Technology
    - Width
    - Speed

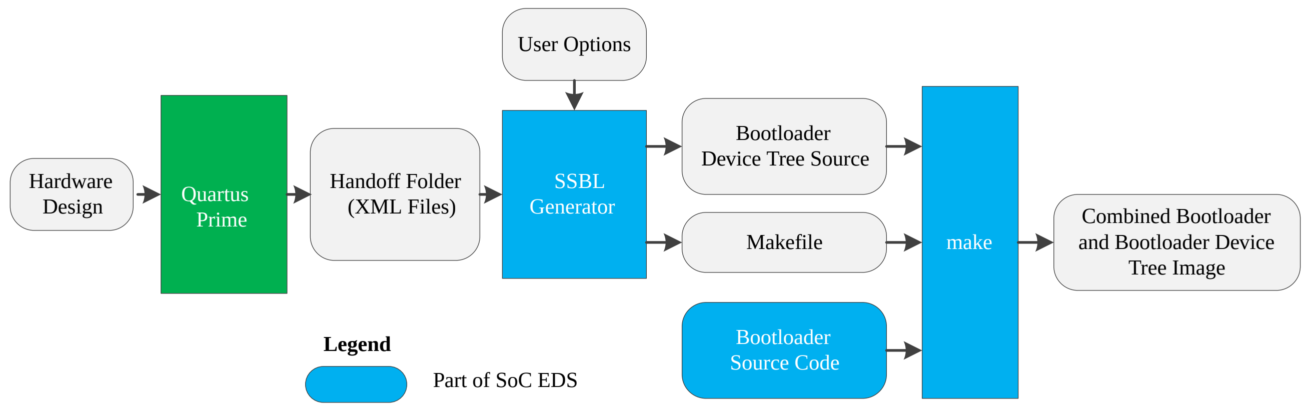
The handoff settings are output from the Quartus Prime compilation and are located in the **<quartus project directory>/hps_isw_handoff/<hps entity name>** directory (where <**hps entity name** > is the HPS component name in Qsys).

You must update the hardware handoff files and regenerate the BSP each time a hardware change impacts the HPS, such as after pin multiplexing or pin assignment changes.

Arria 10 Flow

For Arria 10, the BSP Generator creates a customized BSP consisting of a makefile and a Bootloader Device Tree. There is no source code generated, as all customization is encapsulated in the Bootloader Device Tree and makefile settings. The makefile can be used to create the combined bootloader image, which contains both the bootloader executable and the bootloader device tree. The combined image can then be downloaded to a Flash device or FPGA RAM to be used for booting HPS.

Figure 7-4: Arria 10 SSBL Support Package Generator Flow



The hardware handoff information contains various settings that the user entered when creating the hardware design in Qsys and Quartus Prime. These include the following:

- Pin-muxing for the HPS dedicated pins
- I/O settings for the HPS dedicated pins:
  - Voltage
  - Slew rate
  - Pull up/ down
- Pin-muxing for the shared pins
- State of HPS peripherals:
  - Enabled
  - Disabled
- Configuration of the bridges between HPS and FPGA
- Clock tree settings:
  - PLL settings
  - Clock divider settings
  - Clock gatting settings

The handoff settings are output from the Quartus Prime compilation and are located in the **<quartus project directory>/hps_isw_handoff** directory.

The user must run the BSP Generator and re-generate the Bootloader device tree each time a hardware change results in a change of the above parameters.

However, the user does not have to always recompile the Bootloader whenever a hardware setting is changed. The Bootloader needs to be recompiled only when changing the boot source.

BSP Generator Graphical User Interface

You must perform the following steps to use the BSP Generator GUI, **bsp -editor**:

1. Start an embedded command shell.

2. Run the **bsp -editor** command in the embedded command shell to launch the BSP Generator GUI.

3. To open and modify an existing BSP project, click **File** > **Open** and browse to an existing .bsp file.

4. To create a new BSP project, click **File** > **New HPS BSP** to open the **New BSP** dialog box. The **New BSP** dialog box includes the following settings and parameters:

- **Preloader settings directory** – the path to the hardware handoff files. The generator inspects the handoff files to verify the validity of this path.
- **Operating system** – Select the type of SSBL from the following two options:
  - U-Boot SPL Preloader (Cyclone V/Arria V HPS)
  - U-Boot Bootloader (Arria 10 HPS)
- **Version** – the SSBL version to use. This release only supports the default 1.0 version.
- **Use default locations** – checked by default, to derive the location of the BSP from the location of the hardware handoff folder. Uncheck if a custom path is desired instead.

- **BSP target directory** – the destination folder for new BSP files created by the generator. This document refers to this as < **bsp directory**>. The default directory name is “spl_bsp” for the Arria V/ Cyclone V Preloader and “uboot_bsp” for the Arria 10 Bootloader. The directory name can be modified when **Use default locations** is unchecked.
- **BSP settings file name** – the location and filename of the **. bsp** file containing the BSP settings.
- **Additional .tcl scripting** – the location and filename of a .tcl script for overriding the default BSP settings.

5. You can customize the BSP. After creating or opening a .bsp file, access the **Settings** in the **BSP Editor** dialogue box. The Settings are divided into Common and Advanced settings. When you select a group of settings, the controls for the selected settings appear on the right side of the dialogue box.

When you select a single setting, the setting name, description and value are displayed. You can edit these settings in the **BSP Editor** dialogue box.

6. Click **Generate** to generate the BSP.

7. Click **Exit** to exit the BSP Generator GUI.

Using .tcl Scripts

Instead of using the default settings, you can create a tcl script file (**.tcl**) to define custom settings during BSP creation.

`set_setting` is the only available **.tcl** command.

Refer to BSP Settings for a list of available settings.

The following shows example commands that are used to set parameters in the BSP settings file:

```
set_setting spl.boot.BOOT_FROM_QSPI true
set_setting spl.boot.QSPI_NEXT_BOOT_IMAGE 0x50000
```

Related Information

BSP Settings on page 7-9

BSP Generator Command Line Interface

The BSP command-line tools can be invoked from the embedded command shell, and provide all the features available in the BSP Generator GUI:

- The **bsp-create-settings** tool creates a new BSP settings file.
- The **bsp-update-settings** tool updates an existing BSP settings file.
- The **bsp-query-settings** tool reports the setting values in an existing BSP settings file.
- The **bsp-generate-files** tool generates a BSP from the BSP settings file.

Note: Help for each tool is available from the embedded command shell. To display help, type the following command:`<name of tool> --help`.

bsp-create-settings

The **bsp-create-settings** tool creates a new BSP settings file with default settings. You have the option to modify the BSP settings or generate the BSP files as shown in the following example.

Example 7-1: Creating a New BSP Settings File

```
The following example creates a new Preloader BSP settings file, based on
the hardware handoff information and using the default BSP settings:
bsp-create-settings --type spl --bsp-dir . \
 --settings settings.bsp \
 --preloader-settings-dir ../../hps_isw_handoff/<hps_entity_name>
```

Table 7-1: User Parameters: bsp-create-settings

Option	Required	Description
`--type <bsp type>`	Yes	This option specifies the type of BSP. Allowed values are "spl" for Cyclone V/Arria V Preloader, and "uboot" for Arria 10 Bootloader.
`--settings <filename>`	Yes	This option specifies the path to a BSP settings file. The file is created with default settings.Altera recommends that you name the BSP settings file “settings.bsp”.
`--preloader-settings-dir <directory>`	Yes	This option specifies the path to the hardware handoff files.
`--bsp-dir <directory>`	Yes	This option specifies the path where the BSP files are generated.When specified, **bsp -create-settings** generates the files after the settings file has been created.Altera recommends that you always specify this parameter with **bsp -create-settings**.
`--set <name> <value>`	No	This option sets the BSP setting <name> to the value <value>. Multiple instances of this option can be used with the same command. Refer to **BSP Settings** for a complete list of available setting names and descriptions.

bsp-update-settings

The **bsp-update-settings** tool updates the settings stored in the BSP settings file, as shown in the following example.

Example 7-2: Updating a BSP Settings File

```
The following command changes the value of a parameter inside the file
"settings.bsp":
bsp-update-settings --settings settings.bsp --set spl.debug.SEMIHOSTING 1
```

Table 7-2: User Parameters: bsp-update-settings

Option	Required	Description
`--settings <settings-file>`	Yes	This option specifies the path to an existing BSP settings file to update.
`--bsp-dir <bsp-dir>`	No	This option specifies the path where the BSP files are generated.When this option is specified, **bsp -create-settings** generates the BSP files after the settings file has been created.
`--set <name> <value>`	No	This option sets the BSP setting <name> to the value <value>. Multiple instances of this option can be used with the same command. Refer to **BSP Settings** for a complete list of available setting names and descriptions.

bsp-query-settings

The **bsp-query-settings** tool queries the settings stored in BSP settings file, as shown in the following example.

Example 7-3: Querying a BSP Settings File

```
The following command will retrieve all the settings from "settings.bsp" and
displays the setting names and values:
bsp-query-settings --settings settings.bsp --get-all --show-names
```

Table 7-3: User Parameters: bsp-query-settings

Option	Required	Description
`--settings <settings-file>`	Yes	This option specifies the path to an existing BSP settings file.

Option	Required	Description
`--get <name>`	No	This option instructs **bsp -query-settings** to return the value of the BSP setting <name>.
`--get-all`	No	This option shows all the BSP settings values.When using --get-all,you must also use --show-names.
`--show-names`	No	This option only takes effect when used together with --get <name> or --get-all. When used with one of these options, names and values of the BSP settings are shown side- by-side.

bsp-generate-files

The **bsp-generate-files** tool generates the files and settings stored in BSP settings file, as shown in the following examples.

Example 7-4: Generating Files After BSP Creation

```
The following commands create a settings file based on the handoff folder,
and then generate the BSP files based on those settings:
bsp-create-settings --type spl --bsp-dir . \
--settings settings.bsp \
--preloader-settings-dir \
../../hps_isw_handoff/<hps_entity_name>
bsp-generate-files --settings settings.bsp --bsp-dir
```

Example 7-5: Generating Files After BSP Updates

```
The following commands update the settings of a an existing BSP settings
file, and then generate the BSP files based on those settings:
bsp-update-settings --settings settings.bsp --set \
spl.debug.SEMIHOSTING 1
bsp-generate-files --settings settings.bsp --bsp-dir
```

Use the **bsp-generate-files** tool when BSP files need to be regenerated in one of the following conditions:

- **bsp-create-settings** created the BSP settings, but the --bsp-dir parameter was not specified, so BSP files were not generated.
- **bsp-update-settings** updated the BSP settings, but the --bsp-dir parameter was not specified, so the files were not updated.
- You want to ensure the BSP files are up-to-date.

Table 7-4: User Parameters: bsp-generate-files

Option	Required	Description
`--settings <settings-file>`	Yes	This option specifies the path to an existing BSP settings file.
`--bsp-dir <bsp-dir>`	Yes	This option specifies the path where the BSP files are generated.

BSP Files and Folders

The files and folders created with the BSP Generator are stored in the location you specified in **BSP target directory** in the **New BSP** dialog box.

For Cyclone V/Arria Preloader BSPs, the generated files include the following:

- **settings.bsp** – file containing all BSP settings
- **Makefile** – makefile used to compile the Preloader and create the preloader image; for more information, refer to Preloader Compilation
- **preloader.ds** – ARM DS-5 script, that can be used to download and debug the Preloader on a target device
- **generated** – folder containing files generated from the hardware handoff files

For Arria 10 Bootloader BSPs, the generated files include the following:

- **settings.bsp** – file containing all BSP settings
- **Makefile** – makefile used to compile the Bootloader, convert the Bootloader device tree file to binary, and create the combined Bootloader and Bootloader Device Tree image; for more information, refer to Preloader Compilation
- **config.mk** – makefile configuration file, containing the boot source selection and whether the Bootloader compilation is selected
- **devicetree.dts** – Bootloader device tree, containing the Bootloader customization details, derived from the handoff files and the user settings
- **uboot .ds** – ARM DS-5 script, that can be used to download and debug the Bootloader on a target device

BSP Settings

This section lists all the available BSP settings, which can be accessed from either the Graphical User Interface application (**bsp-editor**) or from the command line tools (**bsp-create-settings**, **bsp-update-settings**, **bsp-query-settings**).

The available BSP settings are different between Cyclone V and Arria V SSBL (Preloader) and Arria 10 SSBL (Bootloader).

Cyclone V and Arria V BSP Settings

Table 7-5: Cyclone V and Arria V BSP Settings

BSP Setting	Type	Default Value	Description
`spl.PRELOADER_TGZ`	String	`"<SoC EDS installation directory>/host_tools/altera/preloader/uboot-socfpga.tar.gz"`	This setting specifies the path to archive file containing the preloader source files.
`spl.CROSS_COMPILE`	String	`"arm-altera-eabi-"`	This setting specifies the cross compilation tool chain for use.
`spl.boot.BOOT_FROM_QSPI`	Boolean	False	Select the source for the subsequent boot image. Note that only one source can be active at a time. When using **bsp-create-settings** or **bsp-update-settings**, you must turn off the boot option that is currently turned on before you can turn on a different boot option.
`spl.boot.BOOT_FROM_SDMMC`	Boolean	True	
`spl.boot.BOOT_FROM_RAM`	Boolean	False	
`spl.boot.BOOT_FROM_NAND`	Boolean	False	
`spl.boot.QSPI_NEXT_BOOT_IMAGE`	Hexadecimal	0x60000	This setting specifies the location of the subsequent boot image in QSPI.
`spl.boot.SDMMC_NEXT_BOOT_IMAGE`	Hexadecimal	0x40000	This setting specifies the location of the subsequent boot image in SD/MMC.
`spl.boot.NAND_NEXT_BOOT_IMAGE`	Hexadecimal	0xC0000	This setting specifies the location of the subsequent boot image in NAND.
`spl.boot.FAT_SUPPORT`	Boolean	False	Enable FAT partition support when booting from SD/MMC.
`spl.boot.FAT_BOOT_PARTITION`	Decimal-Number	1	When FAT partition support is enabled, this specifies the FAT partition where the boot image is located.
`spl.boot.FAT_LOAD_PAYLOAD_NAME`	String	`"u-boot.img"`	When FAT partition supported is enabled, this specifies the boot image filename to be used.

BSP Setting	Type	Default Value	Description
`spl.boot.WATCHDOG_ENABLE`	Boolean	True	This setting enables the watchdog during the preloader execution phase. The watchdog remains enabled after the preloader exits.
`spl.boot.CHECKSUM_NEXT_IMAGE`	Boolean	True	This setting enables the preloader to validate the checksum in the subsequent boot image header information.
`spl.boot.EXE_ON_FPGA`	Boolean	False	This setting executes the preloader on the FPGA. Select `spl.boot.EXE_ON_FPGA` when the preloader is configured to boot from the FPGA.
`spl.boot.STATE_REG_ENABLE`	Boolean	True	This setting enables writing the magic value to the **INITSWSTATE** register in the system manager when the preloader exists; this indicates to the boot ROM that the preloader has run successfully.
`spl.boot.BOOTROM_HANDSHAKE_ CFGIO`	Boolean	True	This setting enables handshake with boot ROM when configuring the IOCSR and pin multiplexing. If `spl.boot.BOOTROM_ HANDSHAKE_ CFGIO` is enabled and warm reset occurs when the preloader is configuring IOCSR and pin multiplexing, the boot ROM will reconfigure IOCSR and pin multiplexing again. This option is enabled by default.
`spl.boot.WARMRST_SKIP_CFGIO`	Boolean	True	This setting enables the preloader to skip IOCSR and pin multiplexing configuration during warm reset. `spl.boot.WARMRST_ SKIP_CFGIO` is only applicable if the boot ROM has skipped IOCSR and pin multiplexing configura-tion.
`spl.boot.SDRAM_INITIALIZA-TION`	Boolean	False	Initialize the SDRAM to initialize the ECC bits.

BSP Setting	Type	Default Value	Description
`spl.boot.SDRAM_ECC_INIT_ BOOT_REGION_START`	Hexadecimal	0x1000000	The start address of the memory region within the SDRAM to be initialized.
`spl.boot.SDRAM_ECC_INIT_ BOOT_REGION_END`	Hexadecimal	0x2000000	The end address of the memory region within SDRAM to be initialized.
`spl.boot.SDRAM_ECC_INIT_ REMAIN_REGION`	Boolean	True	Initialize the remaining SDRAM, during the flash accesses to load the image.
`spl.debug.DEBUG_MEMORY_WRITE`	Boolean	False	This setting enables the preloader to write debug information to memory for debugging, useful when UART is not available. The address is specified by `spl.debug.DEBUG_ MEMORY_ADDR`.
`spl.debug.SEMIHOSTING`	Boolean	False	This setting enables semihosting support in the preloader, for use with a debugger tool. `spl.debug.SEMIHOSTING` is useful when UART is unavailable.
`spl.debug.SKIP_SDRAM`	Boolean	False	The preloader skips SDRAM initialization and calibration when this setting is enabled.
`spl.performance.SERIAL_ SUPPORT`	Boolean	True	This setting enables UART print out support, enabling preloader code to call printf() at runtime with debugging information. stdout output from printf() is directed to the UART. You can view this debugging information by connecting a terminal program to the UART specified peripheral.

BSP Setting	Type	Default Value	Description
`spl.reset_assert.DMA`	Boolean	False	When enabled, this setting will force the corresponding peripheral to remain in reset. You must ensure the debugger does not read registers from these components.
`spl.reset_assert.GPIO0`	Boolean	False	
`spl.reset_assert.GPIO1`	Boolean	False	
`spl.reset_assert.GPIO2`	Boolean	False	
`spl.reset_assert.L4WD1`	Boolean	False	
`spl.reset_assert.OSC1TIMER1`	Boolean	False	
`spl.reset_assert.SDR`	Boolean	False	
`spl.reset_assert.SPTIMER0`	Boolean	False	
`spl.reset_assert.SPTIMER1`	Boolean	False	
`spl.warm_reset_handshake.FPGA`	Boolean	True	This setting enables the reset manager to perform handshake with the FPGA before asserting a warm reset.
`spl.warm_reset_handshake.ETR`	Boolean	True	This setting enables the reset manager to request that the Embedded Trace Router (ETR) stalls the Advanced eXtensible Interface (AXI) master and waits for the ETR to finish any outstanding AXI transactions before asserting a warm reset of the L3 interconnect or a debug reset of the ETR.

BSP Setting	Type	Default Value	Description
`spl.warm_reset_handshake.SDRAM`	Boolean	False	This option allows the SDRAM contents to be preserved accros warm resets. **Note:** When using this option, the SDRAM controller is not completely re-initialized when coming back from warm reset. This may be a problem whenever the warm reset is generated by the Watchdog as a consequence of an SDRAM controller failure. **Note:** Also the SDRAM PLL is not re-initialized when this option is enabled and the system comes out of the warm reset.
`spl.boot.FPGA_MAX_SIZE`	Hexadecimal	0x10000	This setting specifies the maximum code (**.text** and **.rodata**) size that can fit within the FPGA. If the code build is bigger than the specified size, a build error is triggered.
`spl.boot.FPGA_DATA_BASE`	Hexadecimal	0xFFFF0000	This setting specifies the base location for the data region (**.data**, **.bss**, **heap** and **stack**) when execute on FPGA is enabled.
`spl.boot.FPGA_DATA_MAX_SIZE`	Hexadecimal	0x10000	This setting specifies the maximum data (**.data**, **.bss**, **heap** and **stack**) size that can fit within FPGA. If the code build is bigger than the specified size, a build error is triggered.
`spl.debug.DEBUG_MEMORY_ADDR`	Hexadecimal	0xFFFFFD00	This setting specifies the base address for storing preloader debug information enabled with the `spl.debug.DEBUG_MEMORY_WRITE` setting.

BSP Setting	Type	Default Value	Description
`spl.debug.DEBUG_MEMORY_SIZE`	Hexadecimal	0x200	This setting specifies the maximum size used for storing preloader debug information.
`spl.debug.DEBUG_MEMORY_ADDR`	Hexadecimal	0xFFFFFD00	This setting specifies the base address for storing preloader debug information enabled with the `spl.debug.DEBUG_MEMORY_WRITE` setting.
`spl.debug.HARDWARE_DIAGNOSTIC`	Boolean	False	Enable hardware diagnostic support. To enable this, at least 1GB of memory is needed, otherwise hardware diagnostic will fail to run properly.
`spl.boot.RAMBOOT_PLLRESET`	Boolean	True	Execute RAM Boot PLL reset code on warm reset when CSEL = 00. This option is required to enable correct warm reset functionality when using CSEL = 00. When enabling this option, the upper 4 KB of OCRAM are reserved and must not be modified by the user software. **Note:** Enabling this feature with CSEL != 00 does not have any effect, as the impacted code checks for this.

Arria 10 BSP Settings

The Arria 10 BSP Settings are divided in four different groups:

- Main Group
- MPU Firewall
- L3 Firewall Group
- F2S Firewall Group

The following table presents the settings prefix to be added in front of the setting name for each of the groups. They are presented in a table to make the rest of this section more readable.

Table 7-6: Arria 10 BSP Firewall Setting Prefixes

Settings Group	Setting Group Prefix
Main Group	N/A
MPU Firewall	`mpu_m0.noc_fw_ddr_mpu_fpga2sdram_ddr_scr.*`

Settings Group	Setting Group Prefix
L3 Firewall Group	`mpu_m0.noc_fw_ddr_l3_ddr_scr.*`
F2S Firewall Group	`mpu_m0.noc_fw_ddr_mpu_fpga2sdram_ddr_scr.*`

Aria 10 Main BSP Settings Group

Table 7-7: Arria 10 Main BSP Settings Group

BSP Setting	Type	Default Value	Description
`uboot.boot_device`	String	`"SD/MMC"`	Select the source for the boot image. Possible values are SD/MMC and QSPI.
`uboot.model`	String	`"SOCFPGA Arria10 Dev Kit"`	Name of the board to be displayed when bootloader starts.
`uboot.external_fpga_config`	Boolean	False	Configure Uboot to wait early on in the boot sequence to allow the FPGA to be brought to user mode by either a JTAG download or an externally connected flash.
`uboot.rbf_filename`	String	`socfpga.rbf`	Full FPGA `.rbf` filename. This setting is ignored when the `uboot.external_fpga_config` setting is enabled.
`uboot.rbf_offset`	Hexadecimal	`0x720000`	RBF offset address in QSPI Flash.
`uboot.disable_uboot_build`	Boolean	False	Can be used to disable building Uboot, and just generate the Bootloader Device Tree file.
`uboot.secureboot.enable_bootloader_encryption`	Boolean	0	Encrypt Bootloader using key file specified.
`uboot.secureboot.enable_bootloader_signing`	Boolean	0	Sign Bootloader using the key pair file specified.
`uboot.secureboot.encryption_key_file`	String	`encrypt.key`	Key file used for Bootloader encryption.
`uboot.secureboot.encryption_key_name`	String	`key1`	Key Name to use within Key File for Bootloader Encryption.
`uboot.secureboot.signing_key_fpga_offset`	Hexadecimal	`0x0`	Offset from H2F Bridge Base Address (`0xC0000000`) to location of root-public-key.

BSP Setting	Type	Default Value	Description
`uboot.secureboot.signing_key_pair_file`	String	`root_key.pem`	Key Pair File to use when signing is enabled. You can generate this file with the command: `'make generate-signing-key-pair-file'`.
`uboot.secureboot.signing_key_type`	String	`user`	Sign Bootloader using key pair file specified.

Arria 10 Bootloader MPU Firewall BSP Settings Group

Table 7-8: Arria 10 Bootloader MPU Firewall BSP Settings Group

BSP Setting	Type	Default Value	Description
`enable.mpuregion0enable`	Boolean	True	Enable MPU Firewall Regions
`enable.mpuregion1enable`		False	
`enable.mpuregion2enable`		False	
`enable.mpuregion3enable`		False	
`mpuregion0addr.base`	Hexadecimal	0x0	MPU Firewall region bases
`mpuregion1addr.base`			
`mpuregion2addr.base`			
`mpuregion3addr.base`			
`mpuregion0addr.limit`	Hexadecimal	0xffff	MPU Firewall region limits
`mpuregion1addr.limit`			
`mpuregion2addr.limit`			
`mpuregion3addr.limit`			

Arria 10 Bootloader L3 Firewall BSP Settings Group

Table 7-9: Arria 10 Bootloader L3 Firewall BSP Settings Group

BSP Setting	Type	Default Value	Description
`enable.hpsregion0enable`	Boolean	True	Enable L3 Firewall regions
`enable.hpsregion1enable`		False	
`enable.hpsregion2enable`		False	
`enable.hpsregion3enable`		False	
`enable.hpsregion4enable`		False	
`enable.hpsregion5enable`		False	
`enable.hpsregion6enable`		False	
`enable.hpsregion7enable`		False	
`hpsregion0addr.base`	Hexadecimal	0x0	L3 Firewall region bases
`hpsregion1addr.base`			
`hpsregion2addr.base`			
`hpsregion3addr.base`			
`hpsregion4addr.base`			
`hpsregion5addr.base`			
`hpsregion6addr.base`			
`hpsregion7addr.base`			
`hpsregion0addr.limit`	Hexadecimal	0xffff	L3 Firewall region limits
`hpsregion1addr.limit`			
`hpsregion2addr.limit`			
`hpsregion3addr.limit`			
`hpsregion4addr.limit`			
`hpsregion5addr.limit`			
`hpsregion6addr.limit`			
`hpsregion7addr.limit`			

Arria 10 Bootloader F2S Firewall BSP Settings Group

Table 7-10: Arria 10 Bootloader F2S Firewall BSP Settings Group

BSP Setting	Type	Default Value	Description
`enable.fpga2sdram0region0`	Boolean	True	Enable F2S Firewall regions
`enable.fpga2sdram0region1`		True	
`enable.fpga2sdram0region2`		True	
`enable.fpga2sdram0region3`		False	
`enable.fpga2sdram1region0`		False	
`enable.fpga2sdram1region1`		False	
`enable.fpga2sdram1region2`		False	
`enable.fpga2sdram1region3`		False	
`enable.fpga2sdram2region0`		False	
`enable.fpga2sdram2region1`		False	
`enable.fpga2sdram2region2`		False	
`enable.fpga2sdram2region3`		False	
`fpga2sdram0region0addr.base`	Hexadecimal	0x0	F2S Firewall region bases
`fpga2sdram0region1addr.base`			
`fpga2sdram0region2addr.base`			
`fpga2sdram0region3addr.base`			
`fpga2sdram1region0addr.base`			
`fpga2sdram1region1addr.base`			
`fpga2sdram1region2addr.base`			
`fpga2sdram1region3addr.base`			
`fpga2sdram2region0addr.base`			
`fpga2sdram2region1addr.base`			
`fpga2sdram2region2addr.base`			
`fpga2sdram2region3addr.base`			

BSP Setting	Type	Default Value	Description
`fpga2sdram0region0addr.limit`	Hexadecimal	0xffff	F2S Firewall region limits
`fpga2sdram0region1addr.limit`			
`fpga2sdram0region2addr.limit`			
`fpga2sdram0region3addr.limit`			
`fpga2sdram1region0addr.limit`			
`fpga2sdram1region1addr.limit`			
`fpga2sdram1region2addr.limit`			
`fpga2sdram1region3addr.limit`			
`fpga2sdram2region0addr.limit`			
`fpga2sdram2region1addr.limit`			
`fpga2sdram2region2addr.limit`			
`fpga2sdram2region3addr.limit`			

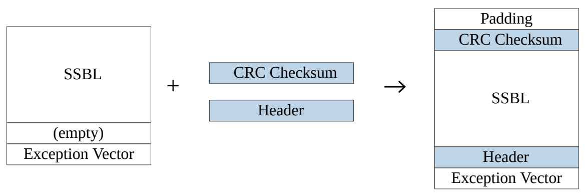
Second Stage Bootloader Image Tool (mkpimage)

The Second Stage Bootloader (SSBL) Image Tool (mkpimage) creates an Altera BootROM-compatible image of the Arria V and Cyclone V Preloader or Arria 10 Bootloader. The tool can also decode the header of previously generated images.

The mkpimage tool makes the following assumptions:

- The input file format is raw binary. You must use the **objcopy** utility provided with the GNU Compiler Collection (GCC) tool chain from the Mentor Graphics website to convert other file formats, such as Executable and Linking Format File (**.elf**), Hexadecimal (Intel-Format) File (**.hex**), or S-Record File (**.srec**), to a binary format.
- The output file format is binary.
- The tool always creates the output image at the beginning of the binary file. If the image must be programmed at a specific base address, you must supply the address information to the flash programming tool.
- The output file contains only Preloader or Bootloader images. Other images such as Linux, SRAM Object File (.sof) and user data are programmed separately using a flash programming tool or related utilities in the U-boot on the target system.

Figure 7-5: Basic Operation of the mkpimage Tool



Operation

The mkpimage tool runs on a host machine. The tool generates the header and CRC checksum and inserts them into the output image with the SSBL program image and its exception vector.

For certain flash memory tools, the position of the SSBL images must be aligned to a specific block size; the mkpimage tool generates any padding data that may be required.

The mkpimage tool optionally decodes and validates header information when given a pre-generated SSBL image.

As illustrated, the binary SSBL image is an input to the mkpimage tool. The compiler leaves an empty space between the SSBL exception vector and the program. The mkpimage tool overwrites this empty region with header information and calculates a checksum for the whole image.

When necessary, the mkpimage tool appends the padding data to the output image.

The mkpimage tool can operate with either one or four input files. Operation on four input files consists in processing each file individually, then concatenating the four resulted images.

Header File Format

The mkpimage header file format has two versions:

- Version 0, used for Cyclone V and Arria V SSBL (Preloader)
- Version 1, used for Arria 10 SSBL (Bootloader)

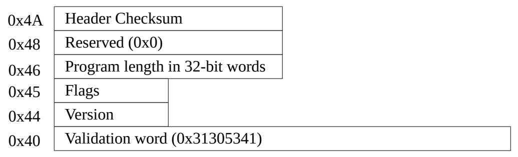
For Version 0, used for Cyclone V and Arria V Preloader, the header includes the following:

- Validation word (0x31305341)
- Version field (set to 0x0)
- Flags field (set to 0x0)
- Program length measured by the number of 32 bit words in the Preloader program
- 16-bit checksum of the header contents (0x40 – 0x49)

Table 7-11: Header Format Version 0

0x4A	Header checksum
0x48	Reserved (0x0)
0x46	Program length in 32-bit words (n)
0x45	Flags
0x44	Version
0x40	Validation word (0x31305341)

Figure 7-6: Header Format Version 0


0x4A Header Checksum
0x48 Reserved (0x0)
0x46 Program length in 32-bit words
0x45 Flags
0x44 Version
0x40 Validation word (0x31305341)

For Version 1, used for Arria 10 Bootloader, the header includes the following:

- Validation word (0x31305341).
- Version field (set to 0x1).
- Flags field (set to 0x0).
- Header length, in bytes, set to 0x14 (20 bytes).
- Total program length (including the exception vectors and the CRC field) in bytes. For an image to be valid, length must be a minimum of 0x5C (92 bytes) and a maximum of 0x32000 (200KiB).
- Program entry offset relative to the start of header (0x40) and should be 32-bit word-aligned. Default is 0x14, any value smaller than that is invalid.
- 16-bit checksum of the header contents (0x40 – 0x51):

Figure 7-7: Header Format Version 1

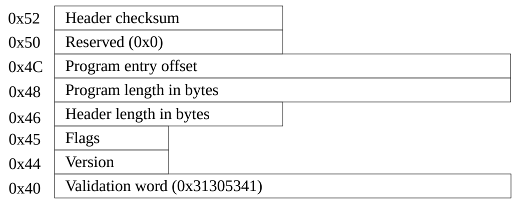

0x52 Header checksum
0x50 Reserved (0x0)
0x4C Program entry offset
0x48 Program length in bytes
0x46 Header length in bytes
0x45 Flags
0x44 Version
0x40 Validation word (0x31305341)

Table 7-12: Header Format Version 1

0x52	Header checksum
0x50	Reserved (0x0)
0x4c	Program entry offset (x)
0x48	Program length in bytes (n)
0x46	Header length in bytes
0x45	Flags
0x44	Version
0x40	Validation word (0x31305341)

The header checksum for both versions of the mkpimage header is the CRC checksum of the byte value from offset 0x0 to (n*4)-4 bytes where n is the program length.

The CRC is a standard CRC32 with the polynomial:

```
x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1
```

. There is no reflection of the bits and the initial value of the remainder is 0xFFFFFFFF and the final value is exclusive OR-ed with 0xFFFFFFFF.

Tool Usage

The mkimage tool has three usage models:

- Single image creation
- Quad image creation
- Single or quad image decoding

If an error is found during the make image process, the tool stops and reports the error. Possible error conditions include:

- For Cyclone V and Arria V Preloaders: input image is smaller than 81 bytes or larger than 60 KB.
- For Arria 10 Bootloaders: input image is smaller than 92 bytes or larger than 200 KB.

mkpimage invokes the tool; invoking the tool with the --help option provides a tool description and tool usage and option information.

```
$ mkpimage --help
mkpimage version 15.1 (build 189)

Description: This tool creates an Altera BootROM-compatible image of Second
Stage Boot Loader (SSBL). The input and output files are in binary format.
It can also decode and check the validity of previously generated image.

Usage:
 Create quad image:
     mkpimage [options] -hv <num> -o <outfile> <infile> <infile> <infile> <infile>
 Create single image:
     mkpimage [options] -hv <num> -o <outfile> <infile>
 Decode single/quad image:
     mkpimage -d [-a <num>] <infile>

Options:
 -a (--alignment) <num>       : Address alignment in kilobytes for output image
                                (64, 128, 256, etc.), default to 64 for header
                                version 0 and 256 for header version 1,
                                override if the NAND flash has a different
                                block size. If outputting a single image, value
                                of '0' is permitted to specify no flash block
                                padding (needed for SSBL image encryption).
 -d (--decode)                : Flag to decode the header information from
                                input file and display it
 -f (--force)                 : Flag to force decoding even if the input file
                                is an unpadded image
 -h (--help)                  : Display this help message and exit
 -hv (--header-version) <num> : Header version to be created (Arria/Cyclone V =
                                0, Arria 10 = 1)
 -o (--output) <outfile>      : Output file, relative and absolute path
                                supported
 -off (--offset) <num>        : Program entry offset relative to start of
                                header (0x40), default to 0x14. Used for header
                                version 1 only
 -v (--version)               : Display version and exit
```

Output Image Layout

Base Address

The bootable SSBL image must be placed at 0x0 for NAND and QSPI flash. For SD/MMC the image can also be placed at 0x0, but typically the image is placed at offset 0x0 in a custom partition of type 0xA2. The custom partition does not have a filesystem on it. The boot ROM is able to locate the partition using the MBR (Master Boot Record) located at 0x0 on the SD/MMC card.

The mkpimage tool always places the output image at the start of the output binary file, regardless of the target flash memory type. The flash programming tool is responsible for placing the image at the desired location on the flash memory device.

Size

For Cyclone V and Arria V, a single Preloader has a maximum 60 KB image size. You can store up to four preloader images in flash. If the boot ROM does not find a valid preloader image at the first location, it attempts to read an image from the next location and so on. To take advantage of this feature, program four preloader images in flash.

For Arria 10, a single Bootloader has a maximum 200 KB image size. You can store up to four Bootloader images in flash. If the boot ROM does not find a valid Bootloader image at the first location, it attempts to read the next one and so on. To take advantage of this feature, program four Bootloader images in flash.

Address Alignment

For Cyclone V and Arria V, every Preloader image has to be aligned to a 64KB boundary, except for NAND devices. For NAND devices, each Preloader image has to be aligned to the greater of 64 KB or NAND block size.

For Arria 10, every Bootloader image has to be aligned to a 256 KB boundary, except for NAND devices. For NAND devices, each Bootloader image has to be aligned to the greater of 256 KB or NAND block size.

The following tables present typical image layouts, that are used for QSPI, SD/MMC and NAND devices with block size equal or less to 64 KB (for Cyclone V/Arria V) or 256 KB (for Arria 10).

Table 7-13: Typical Arria V/Cyclone V Preloader Image Layout

Offset	Image
0x30000	Preloader Image 3
0x20000	Preloader Image 2
0x10000	Preloader Image 1
0x00000	Preloader Image 0

Table 7-14: Typical Arria 10 Bootloader Image Layout

Offset	Image
0xC0000	Bootloader Image 3

Offset	Image
0x80000	Bootloader Image 2
0x40000	Bootloader Image 1
0x00000	Bootloader Image 0

The mkpimage tool is unaware of the target flash memory type. If you do not specify the block size, the default is 64 KB.

NAND Flash

Each Preloader or Bootloader image occupies an integer number of blocks. A block is the smallest entity that can be erased, so updates to a particular boot image do not impact the other images.

For example for Cyclone V and Arria V, a single Preloader image has a maximum size of 64 KB. But it the NAND block is 128 KB, then the Preloader images will need to be located at 128 KB intervals.

Serial NOR Flash

Each QSPI boot image occupies an integer number of sectors unless subsector erase is supported; this ensures that updating one image does not affect other images.

SD/MMC

The master boot record, located at the first 512 bytes of the device memory, contains partition address and size information. The Preloader and Bootloader images are stored in partitions of type 0xA2. Other items may be stored in other partition types according to the target file system format.

You can use the fdisk tool to set up and manage the master boot record. When the fdisk tool partitions an SD/MMC device, the tool creates the master boot record at the first sector, with partition address and size information for each partition on the SD/MMC.

Padding

The mkimage tool inserts a CRC checksum in the unused region of the image. Padding fills the remaining unused regions. The contents of the padded and unused regions of the image are undefined.

Related Information

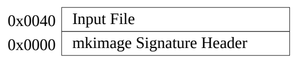
- **Arria V Hard Processor System Technical Reference Manual**
  For more information, please refer to the Booting and Configuration chapter.
- **Cyclone V Hard Processor System Technical Reference Manual**
  For more information, please refer to the Booting and Configuration chapter.
- **Arria 10 Hard Processor System Technical Reference Manual**
  For more information, please refer to the Booting and Configuration chapter.

U-Boot Image Tool (mkimage)

Both the Preloader (for Arria V/ Cyclone V) and the Bootloader (for Arria 10) require the presence of the U-Boot image header at the beginning of the next stage boot image. Depending on usage scenario, other items that are loaded by either Preloader or Bootloader may also require the presence of the U-Boot image header.

The mkimage utility is delivered with SoC EDS and can be used to append the U-Boot image header to the next stage boot image or any other required files.

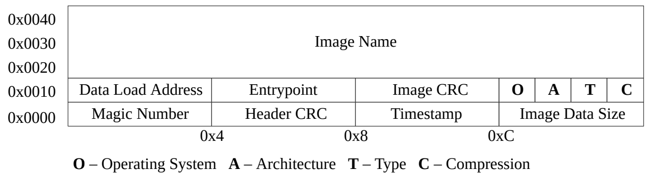
Figure 7-8: mkimage Header



The header consists of the following items:

- Image magic number - determines if the image is a valid boot image
- Image data size - the length of the image to be copied
- Data load address - the entry point of the boot image (not used for items that are not bootable images)
- Operating system - determines the type of image
- Image name - the name of the boot image
- Image CRC - the checksum value of the boot image

Figure 7-9: mkimage Header Layout



Tool Options

mkimage invokes the mkimage tool and the --help option provides the tool description and option information.

```
$ mkimage --help
Usage: mkimage -l image
          -l ==> list image header information
       mkimage [-x] -A arch -O os -T type -C comp -a addr -e ep -n name -d
data_file[:data_file...] image
          -A ==> set architecture to 'arch'
          -O ==> set operating system to 'os'
          -T ==> set image type to 'type'
          -C ==> set compression type 'comp'
          -a ==> set load address to 'addr' (hex)
          -e ==> set entry point to 'ep' (hex)
          -n ==> set image name to 'name'
          -d ==> use image data from 'datafile'
          -x ==> set XIP (execute in place)
       mkimage [-D dtc_options] [-f fit-image.its|-F] fit-image
          -D => set options for device tree compiler
          -f => input filename for FIT source
Signing / verified boot not supported (CONFIG_FIT_SIGNATURE undefined)
       mkimage -V ==> print version information and exit
```

Usage Examples

Example 7-6: Creating a U-boot Image

```
mkimage –A arm –T firmware –C none –O u-boot –a 0x08000040 –e 0 –n "U-Boot
2014.10 for SOCFGPA board" –d u-boot.bin u-boot.img
```

Example 7-7: Creating a Bare-metal Application Image

```
mkimage -A arm -O u-boot -T standalone -C none -a 0x02100000 -e 0 -n
"baremetal image" -d hello_world.bin hello_world.img
```

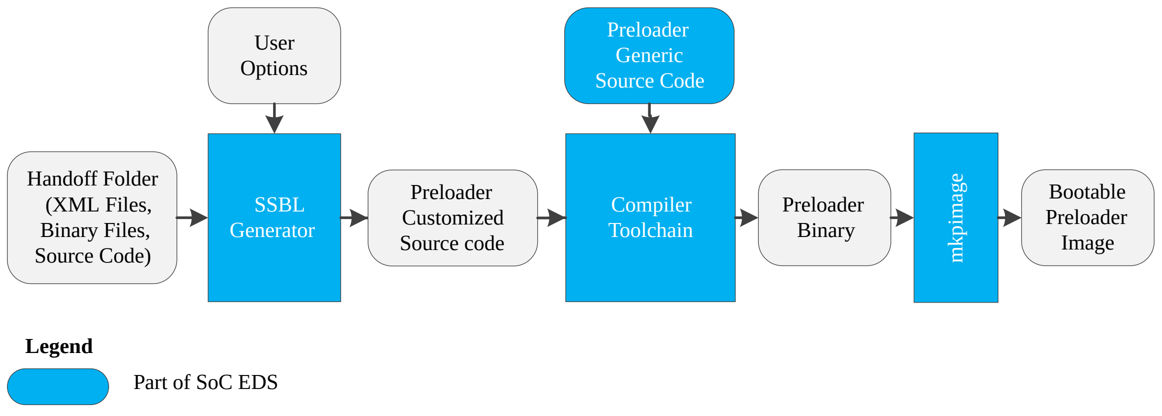
Building the Second Stage Bootloader

This section presents the details of how the bootable SSBL image is created, with the aid of the generated Makefile, for both Cyclone V and Arria V Preloader, and Arria 10 Bootloader.

Building the Cyclone V and Arria V Preloader

The following figure presents the Cyclone V and Arria V Preloader build flow, as performed by the generated Makefile when invoking **make**. In order to keep the illustration simple, the generated Makefile itself is not shown.

Figure 7-10: Cyclone V/Arria V Preloader Build Flow


User Options
Preloader Generic Source Code
Handoff Folder (XML Files, Binary Files, Source Code)
SSBL Generator
Preloader Customized Source code
Compiler Toolchain
Preloader Binary
mkpimage
Bootable Preloader Image
Legend
Part of SoC EDS

The makefile performs the following tasks:

- Copies the generic preloader source code into < **bsp directory**>/ **uboot-socfpga**
- Copies the generated BSP files and hardware handoff files to the source directory in < **bsp_directory** >/ **uboot-socfpga /board/ altera / socfpga _<device>**
- Configures the compiler tools to target an SoC FPGA
- Compiles the source files in < **bsp directory**>/ **uboot-socfpga** with the user-specified cross compiler (specified in the BSP settings) and stores the generated preloader binary files in < **bsp_directory** >/ **uboot - socfpga / spl**
- Converts the preloader binary file to a preloader image, < **bsp_directory** >/**preloader- mkpimage.bin**, with the **mkpimage** tool

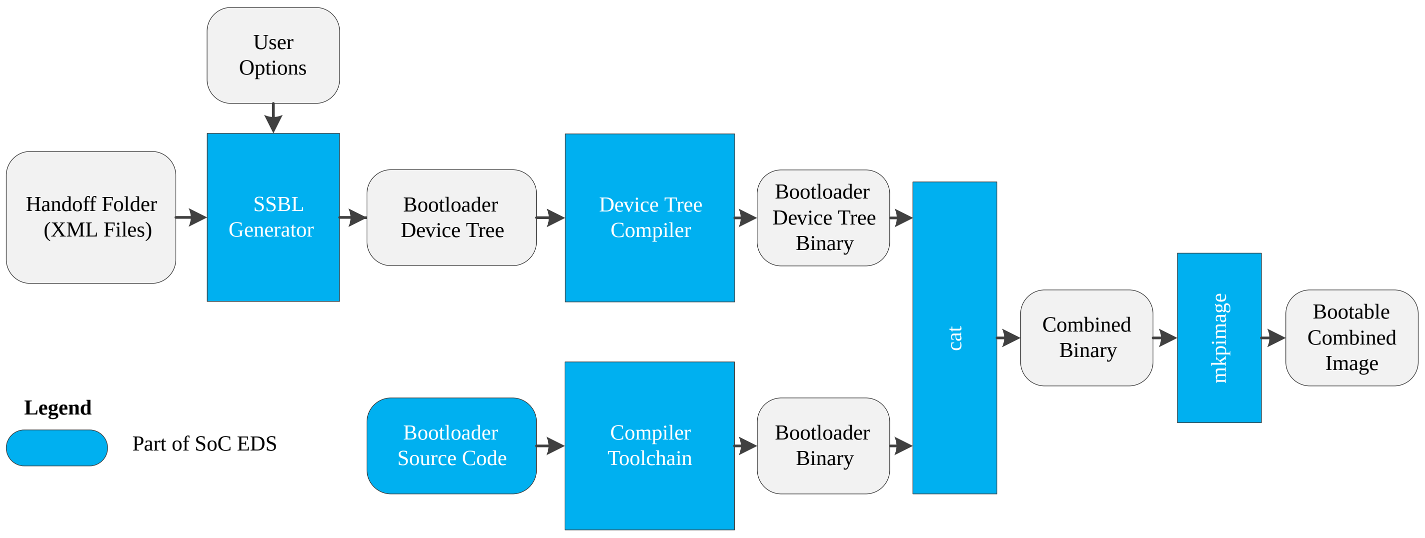
The makefile contains the following targets:

- make all – compiles the preloader
- make clean – deletes preloader-mkpimage.bin from the <bsp directory>
- make clean-all – deletes <bsp directory>, including the source files in the directory

Building the Arria 10 Bootloader

The following figure presents the Arria 10 Bootloader build flow, as performed by the generated Makefile when invoking **make**. In order to keep the illustration simple, the generated Makefile itself is not shown.

Figure 7-11: Arria 10 Bootloader Build Flow



Warning: For this release of the tools, the Bootloader compilation is only supported on Linux host machines. The Bootloader compilation It is not supported on Windows host machines. The rest of the flow is supported on both Linux and Windows host machines.

Note: The Bootloader needs to be recompiled only when the boot source is changed (SD/MMC and QSPI are currently supported). The Bootloader does not need to be recompiled when other settings are changed, as those settings are encapsulated in the Bootloader Device Tree.

The makefile performs the following tasks:

- Copies the bootloader source code into < **bsp directory**>/ **uboot-socfpga**
- Configures the bootloader to target the Arria 10 SoC
- Compiles the source files in < **bsp directory**>/ **uboot-socfpga** and creates the Bootloader binary < **bsp directory**>/ **uboot-socfpga /u- boot .bin**
- Creates the bootloader device tree file based on the hardware handoff information and user settings into < **bsp directory**>/ **devicetree.dts**
- Compiles the bootloader device tree source file by using **dtc** (Device Tree Compiler) into < **bsp directory**>/ **devicetree.dtb**
- Concatenates the bootloader device tree binary and the bootloader binary files into a combined binary file < **bsp directory**>/ **u- boot_w_dtb.bin**
- Converts the combined binary file to a bootable combined image, < **bsp directory**>/ **uboot_w_dtb-mkpimage.bin**, with the **mkpimage** tool

The makefile contains the following targets:

- make all – builds everything
- make src – copies the bootloader source code folder
- make uboot – builds the bootloader binary
- make dtb – compiles the bootloader device tree source to device tree binary
- make clean – deletes all built files
- make clean-all – deletes all built files, and the bootloader source code folder

Boot Tools User Guide Document Revision History

Date	Version	Changes
February 2016	2016.02.17	• Updated the help definition for `mkpimage` in the "Tools Usage" and "Tool Options" sections. • Updated the Arria 10 Main BSP Settings Group table.
August 2015	2015.08.06	Added Arria 10 support.

Hardware Library

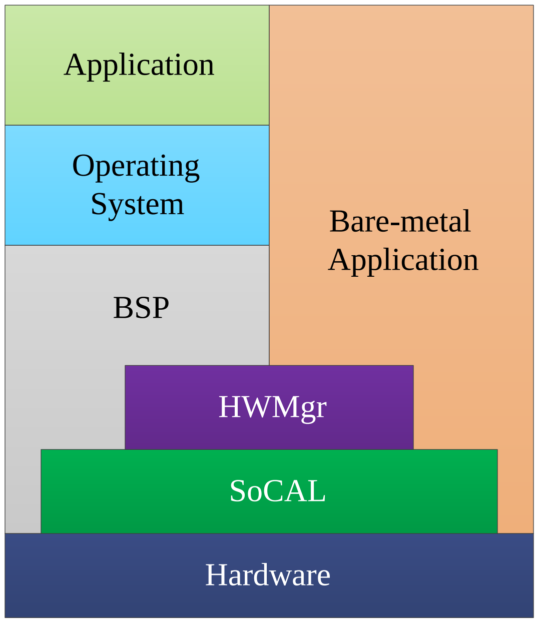
8

2016.02.17

ug-1137 Subscribe Send Feedback

The Altera SoC FPGA Hardware Library (HWLIB) was created to address the needs of low-level software programmers who require full access to the configuration and control facilities of SoC FPGA hardware. An additional purpose of the HWLIB is to mitigate the complexities of managing the operation of a sophisticated, multi-core application processor and its integration with hardened IP peripheral blocks and programmable logic in a SoC architecture.

Figure 8-1: HW Library



Within the context of the SoC HW/SW ecosystem, the HWLIB is capable of supporting software development in conjunction with full featured operating systems or standalone bare-metal programming environments. The relationship of the HWLIB within a complete SoC HW/SW environment is illustrated in the above figure.

The HWLIB provides a symbolic register abstraction layer known as the SoC Abstraction Layer (SoCAL) that enables direct access and control of HPS device registers within the address space. This layer is necessary for enabling several key stakeholders (boot loader developers, driver developers, board support package developers, debug agent developers, and board bring-up engineers) requiring a precise degree of access and control of the hardware resources.

The HWLIB also deploys a set of Hardware Manager (HW Manager) APIs that provides more complex functionality and drivers for higher level use case scenarios.

The HWLIB has been developed as a source code distribution. The intent of this model is to provide a useful set of out-of-the-box functionality and to serve as a source code reference implementation that a user can tailor accordingly to meet their target system requirements.

The capabilities of the HWLIB are expected to evolve and expand over time particularly as common use case patterns become apparent from practical application in actual systems.

In general, the HWLIB assumes to be part of the system software that is executing on the Hard Processor System (HPS) in privileged supervisor mode and in the secure state.

The anticipated HWLIB clients include:

- Bare-Metal application developers
- Custom preloader and boot loader software developers
- Board support package developers
- Diagnostic tool developers
- Software driver developers
- Debug agent developers
- Board bring-up engineers
- Other developers requiring full access to SoC FPGA hardware capabilities

Feature Description

This section provides a description of the operational features and functional capabilities present in the HWLIB. An overview and brief description of the HWLIB architecture is also presented.

The HWLIB is a software library architecturally comprised of two major functional components:

- SoC Abstraction Layer (SoCAL)
- Hardware Manager (HW Manager)

SoC Abstraction Layer (SoCAL)

The SoC Abstraction Layer (SoCAL) presents the software API closest to the actual HPS hardware. Its purpose is to provide a logical interface abstraction and decoupling layer to the physical devices and registers that comprise the hardware interface of the HPS.

The SoCAL provides the benefits of:

- A logical interface abstraction to the HPS physical devices and registers including the bit-fields comprising them.
- A loosely coupled software interface to the underlying hardware that promotes software isolation from hardware changes in the system address map and device register bit field layouts.

Hardware Manager (HW Manager)

The Hardware Manager (HW Manager) component provides a group of functional APIs that address more complex configuration and operational control aspects of selected HPS resources.

The HW Manager functions have the following characteristics:

- Functions employ a combination of low level device operations provided by the SoCAL executed in a specific sequence to effect a desired operation.
- Functions may employ cross functional (such as from different IP blocks) device operations to implement a desired effect.
- Functions may have to satisfy specific timing constraints for the application of operations and validation of expected device responses.
- Functions provide a level of user protection and error diagnostics through parameter constraint and validation checks.

The HW Manager functions are implemented using elemental operations provided by the SoCAL API to implement more complex functional capabilities and services. The HW Manager functions may also be implemented by the compound application of other functions in the HW Manager API to build more complex operations (for example, software controlled configuration of the FPGA).

Hardware Library Reference Documentation

Reference documentation for the SoCAL API and HW Manager API is distributed as part of the SoCEDS Toolkit. This reference documentation is provided as online HTML accessible from any web browser.

The locations of the online SoC FPGA Hardware Library (HWLIB) Reference Documentation are:

- SoC Abstraction Layer (SoCAL) API Reference Documentation:
    - Cyclone V and Arria V: <*SoC EDS installation directory*>**/ip/altera/hps/altera_hps/doc/soc_cv_av/socal/html/index.html**
    - Arria 10: <*SoC EDS installation directory*>**/ip/altera/hps/altera_hps/doc/soc_a10/socal/html/index.html**
- Hardware Manager (HW Manager) API Reference Documentation: <*SoC EDS installation directory*>**/ip/altera/hps/altera_hps/doc/hwmgr/html/index.html**

System Memory Map

The addresses of the HPS hard IP modules are accessible through the provided SoCAL macros. SoCAL also provides macros for accessing the individual registers and register fields of the HPS hard IP modules.

For the FPGA IP modules, the macros for accessing IP registers and register fields are usually part of the IP deliverables. However, the actual IP modules-based addresses are often changed at system integration time, in the Qsys tool.

The tool "sopc-create-header-files" can be used to create a C include file with the bases addresses of all the IP modules residing in the FPGA fabric.

Note: The tool is part of Quartus Prime, and not of SoC EDS. The tool can be invoked from the SoC EDS Embedded Command Shell once Quartus Prime is installed.

The basic usage of the tool is to invoke it with the .sopcinfo file as a single parameter. For example, in order to generate the include files for the Arria 10 GHRD, the following command can be executed in that folder: `sopc-create-header-files ghrd_10as066n2.sopcinfo`.

The tool creates a separate include file for each of the masters in the system, showing the system addresses from that master's point of view. Use the file **<hps_component_name>_a9_0.h** for HPS software development, as it shows the system addresses from the HPS point of view.

The following example demonstrates how to use the tool to generate only the file that shows the HPS A9 Core 0 point of view:

```
sopc-create-header-files ghrd_10as066n2.sopcinfo --module\
arria10_hps_0_arm_a9_0 --single arria10_hps_0_arm_a9_0.h
```

This creates just one file, called "arria10_hps_0_arm_a9_0.h".

You can also run "`sopc-create-header-files --help`" for more details about the tool, or refer to Quartus Prime documentation.

Hardware Library Document Revision History

Date	Version	Changes
February 2016	2016.02.17	• Updated the HW Library figure. • Added a new section called "System Memory Map".
August 2015	2015.08.06	Added Arria 10 support.

HPS Flash Programmer User Guide

2016.02.17

ug-1137 Subscribe Send Feedback

The Altera Quartus Prime software and Quartus Prime Programmer include the HPS flash programmer. Hardware designs, such as HPS, incorporate flash memory on the board to store FPGA configuration data or HPS program data. The HPS flash programmer programs the data into a flash memory device connected to an Altera SoC. The programmer sends file contents over an Altera download cable, such as the USB-Blaster™ II, to the HPS and instructs the HPS to write the data to the flash memory.

The HPS flash programmer programs the following content types to flash memory:

- HPS software executable files — Many systems use flash memory to store non-volatile program code or firmware. HPS systems can boot from flash memory.

  Note: The HPS Flash Programmer is mainly intended to be used for programming the Preloader image to QSPI or NAND flash. Because of the low speed of operation, it is not recommended to be used for programming large files.
- FPGA configuration data — At system power-up, the FPGA configuration controller on the board or HPS read FPGA configuration data from the flash memory to program the FPGA. The configuration controller or HPS may be able to choose between multiple FPGA configuration files stored in flash memory.
- Other arbitrary data files — The HPS flash programmer programs a binary file to any location in a flash memory for any purpose. For example, a HPS program can use this data as a coefficient table or a sine lookup table.

The HPS flash programmer programs the following memory types:

- Quad serial peripheral interface (QSPI) Flash
- Open NAND Flash Interface (ONFI) compliant NAND Flash

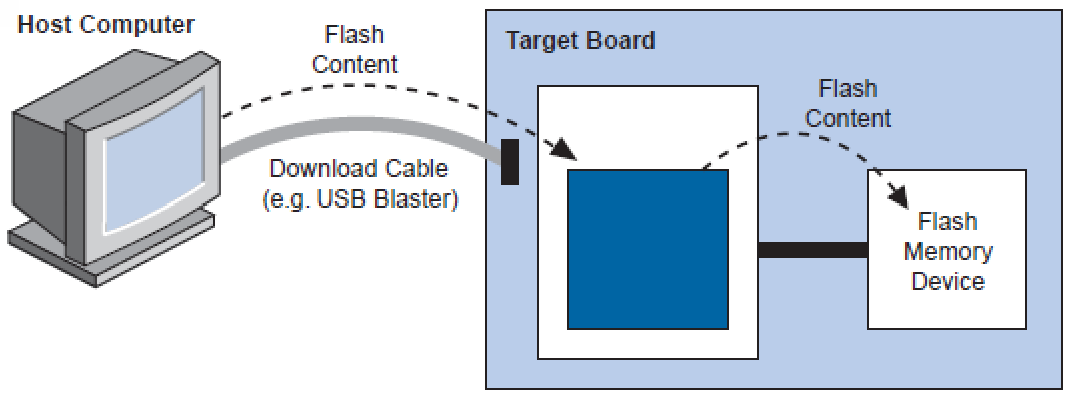
HPS Flash Programmer Command-Line Utility

You can run the HPS flash programmer directly from the command line. For the Quartus Prime software, the HPS flash programmer is located in <*Altera installation directory*>**/quartus/bin**. For the Quartus Prime Programmer, the HPS flash programmer is located in <*Altera installation directory*>**/qprogrammer/bin**.

How the HPS Flash Programmer Works

The HPS flash programmer is divided into a host and a target. The host portion runs on your computer and sends flash programming files and programming instructions over a download cable to the target. The target portion is the HPS in the SoC. The target accepts the programming data flash content and required information about the target flash memory device sent by the host. The target writes the data to the flash memory device.

Figure 9-1: HPS Flash Programmer



The HPS flash programmer determines the type of flash to program by sampling the boot select (BSEL) pins during cold reset; you do not need to specify the type of flash to program.

Using the Flash Programmer from the Command Line

HPS Flash Programmer

The HPS flash programmer utility can erase, blank-check, program, verify, and examine the flash. The utility accepts a Binary File with a required "**.bin**" extension.

The HPS flash programmer command-line syntax is:

```
quartus_hps <options> <file.bin>
```

Note: The HPS flash programmer uses byte addressing.

Table 9-1: HPS Flash Programmer Parameters

Option	Short Option	Required	Description
`--cable`	`-c`	Yes	This option specifies what download cable to use. To obtain the list of programming cables, run the command "jtagconfig". It will list the available cables, like in the following example: jtagconfig 1) USB-Blaster [USB-0] 2) USB-Blaster [USB-1] 3) USB-Blaster [USB-2] The "-c" parameter can be the number of the programming cable, or its name. The following are valid examples for the above case: -c 1 -c "USB-Blaster [USB-2]"
`--device`	`-d`	Yes (if there are multiple HPS devices in the chain)	This option specifies the index of the HPS device. The tool will automatically detect the chain and determine the position of the HPS device; however, if there are multiple HPS devices in the chain, the targeted device index must be specified.

Option	Short Option	Required	Description
`--operation`	`-o`	Yes	This option specifies the operation to be performed. The following operations are supported: • I: Read IDCODE of SOC device and discover Access Port • S: Read Silicon ID of the flash • E: Erase flash • B: Blank-check flash • P: Program flash • V: Verify flash • EB: Erase and blank-check flash • BP: Program <*BlankCheck*> flash • PV: Program and verify flash • BPV: Program (blank-check) and verify flash • X: Examine flash **Note:** The program begins with erasing the flash operation before programming the flash by default.
`--addr`	`-a`	Yes (if the start address is not 0)	This option specifies the start address of the operation to be performed.
`--size`	`-s`	No	This option specifies the number of bytes of data to be performed by the operation. `size` is optional.
`--repeat`	`-t`	No	These options must be used together. The HPS BOOT flow supports up to four images where each image is identical and these options duplicate the operation data; therefore you do not need eSW to create a large file containing duplicate images. `repeat` specifies the number of duplicate images for the operation to perform. `interval` specifies the repeated address. The default value is 64 kilobytes (KB). `repeat`and `interval`are optional.
`--interval`	`-i`		

HPS Flash Programmer Command Line Examples

Type `quartus_hps --help` to obtain information about usage. You can also type `quartus_hps --help=<option>` to obtain more details about each option. For example "quartus_hps --help=o".

Example 9-1: Program File to Address 0 of Flash

`quartus_hps -c 1 -o P input.bin` programs the input file (**input.bin**) into the flash, starting at flash address 0 using a cable *M*.

Example 9-2: Program First 500 Bytes of File to Flash (Decimal)

`quartus_hps -c 1 -o PV -a 1024 -s 500 input.bin` programs the first 500 bytes of the input file (**input.bin**) into the flash, starting at flash address 1024, followed by a verification using a cable *M*.

Note: Without the prefix 0x for the flash address, the tool assumes it is decimal.

Example 9-3: Program First 500 Bytes of File to Flash (Hexadecimal)

`quartus_hps -c 1 -o PV -a 0x400 -s 500 input.bin` programs the first 500 bytes of the input file (**input.bin**) into the flash, starting at flash address 1024, followed by a verification using a cable *M*.

Note: With the prefix 0x, the tool assumes it is hexadecimal.

Example 9-4: Program File to Flash Repeating Twice at Every 1 MB

`quartus_hps -c 1 -o BPV -t 2 -i 0x100000 input.bin` programs the input file (**input.bin**) into the flash, using a cable *M*. The operation repeats itself twice at every 1 megabyte (MB) of the flash address. Before the program operation, the tool ensures the flash is blank. After the program operation, the tool verifies the data programmed.

Example 9-5: Erase Flash on the Flash Addresses

`quartus_hps -c 1 -o EB input.bin` erases the flash on the flash addresses where the input file (**input.bin**) resides, followed by a blank-check using a cable *M*.

Example 9-6: Erase Full Chip

`quartus_hps -c 1 -o E` erases the full chip, using a cable *M*. When no input file (**input.bin**) is specified, it will erase all the flash contents.

Example 9-7: Erase Specified Memory Contens of Flash

`quartus_hps -c 1 -o E -a 0x100000 -s 0x400000` erases specified memory contents of the flash. For example, 4 MB worth of memory content residing in the flash address, starting at 1 MB, are erased using a cable *M*.

Example 9-8: Examine Data from Flash

`quartus_hps -c 1 -o X -a 0x98679 -s 56789 output.bin` examines 56789 bytes of data from the flash with a 0x98679 flash start address, using a cable *M*.

Supported Memory Devices

Table 9-2: QSPI Flash

Flash Device	Mode	Device ID	DIE #	Density (Mb)
N25Q128A	Micron	0x18BA20	1	128
N25Q256	Micron	0x19BA20	1	256
N25Q512	Micron	0x20BA20	2	512
N25Q00	Micron	0x21BA20	4	1024
N25Q128A	Micron	0x18BB20	1	128
N25Q256	Micron	0x19BB20	1	256
N25Q512	Micron	0x20BB20	2	512
N25Q00	Micron	0x21BB20	4	1024
	Micron	0x132020	1	
S25FL128S	Spansion	0x182001	1	128 (Sector size of 64 KB)
S25FL128S	Spansion	0x182001	1	128 (Sector size of 256 KB)
S25FL256S	Spansion	0x190201	1	256 (Sector size of 64 KB)
S25FL256S	Spansion	0x190201	1	256 (Sector size of 256 KB)
S25FL512S	Spansion	0x200201	1	512

Table 9-3: ONFI Compliant NAND Flash

Manufacturer	MFC ID	Device ID	Density (Gb)
Micron	0x2C	0x68	32
Micron	0x2C	0x48	16

Manufacturer	MFC ID	Device ID	Density (Gb)
Micron	0x2C	0xA1	8
Micron	0x2C	0xF1	8

Note: The HPS Flash Programmer supports all ONFI compliant NAND flash devices that are supported by the HPS QSPI Flash Controller.

Note: The HPS Flash Programmer supports the Cyclone V, Arria V and Arria 10 SoC devices.

HPS Flash Programmer User Guide Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Added QSPI Flash part number to the QSPI Flash table in the "Supported Memory Devices" chapter.
August 2015	2015.08.06	Added Arria 10 support.

Bare-Metal Compiler 10

2016.02.17

ug-1137 Subscribe Send Feedback

The bare-metal compiler that is shipped with the SoC EDS is the Mentor Graphics Sourcery™ CodeBench Lite Edition, version 4.9.2.

For more information on the Sourcery CodeBench Lite Edition and to download the latest version of the tools, refer to the Mentor Graphics website.

The compiler is a GCC-based **arm-altera-eabi** port. It targets the ARM processor, it assumes bare-metal operation, and it uses the standard ARM embedded-application binary interface (EABI) conventions.

The bare-metal compiler is installed as part of the SoC EDS installation in the following folder: <*SoC EDS installation directory*>**/host_tools/mentor/gnu/arm/baremetal**.

The Embedded Command Shell, opened by running the script from the *SoC EDS installation directory*, sets the correct environment PATH variables for the bare-metal compilation tools to be invoked. After starting the shell, commands like **arm-altera-eabi-gcc** can be invoked directly. When the ARM DS-5 AE environment is started from the embedded command shell, it inherits the environment settings, and it can call these compilation tools directly.

Alternatively, the full path to the compilation tools can be used: <*SoC EDS installation directory*>**/host_tools/mentor/gnu/arm/baremetal/bin**.

The bare-metal compiler comes with full documentation, located at <*SoC EDS installation directory*>**/host_tools/mentor/gnu/arm/baremetal/share/doc/sourceryg++-arm-altera-eabi**. The documentation is offered in four different formats to accommodate various user preferences:

- HTML files
- Info files
- Man pages
- PDF files

Among the provided documents are:

- Compiler manual
- Assembler manual
- Linker manual
- Binutils manual
- GDB manual
- Getting Started Guides[2]
- Libraries Manual

[2] The Getting Started Guides are located on the **SoCEDSGettingStarted** page on the Altera Wiki.

Related Information

- Mentor Graphics
- SoC EDS Getting Started Guides

Bare Metal Compiler Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Updated the compiler version for Mentor Graphics® Sourcery CodeBench Lite Edition.
August 2015	2015.08.06	Added Arria 10 support.

SD Card Boot Utility 11

2016.02.17

ug-1137 Subscribe Send Feedback

The SoC EDS SD card boot utility is a tool for updating the boot software on an SD card.

The Preloader is typically stored in a custom partition (with type = 0xA2) on the SD card. Optionally the next boot stage (usually the Bootloader) can also be stored on the same custom partition.

Since it is a custom partition, without a file-system, the Preloader and/or Bootloader cannot be updated by copying the new file to the card; and a software tool is needed.

The SD card boot utility allows the user to update the Preloader and/or Bootloader on a physical SD card or a disk image file. The utility is not intended to create a new bootable SD card or disk image file from scratch. In order to do that, it is recommended to use fdisk on a Linux host OS.

Related Information

Linux Software Development Tools on page 12-1

Usage Scenarios

This utility is intended to update boot software on that resides on an existing:

- Existing SD card
- Existing disk image file

The tool supports updating the following:

- Cyclone V and Arria V Preloader
- Cylone V anfd Arria V Bootloader
- Arria 10 Bootloader

Note: For Cyclone V and Arria V, the Preloader file needs to have the mkpimage header, as required by the boot ROM, and the Bootloader file needs to have the mkimage header, as required by the Preloader.

Note: For Arria 10, the Bootloader needs to have the mkpimage header, as required by BootROM.

Note: Both mkpimage and mkimage tools are delivered as part of SoC EDS.

The tool only updates the custom partition that stores the Altera SoC boot code. The rest of the SD card or disk image file is not touched. This includes the Master Boot Record (MBR) and any other partitions (FAT, EXT3 etc) and free space.

Warning: The users of this tool need administrative or root access to their computer to use this tool to write to physical SD cards. These rights are not required when only working with disk image files. Please contact the IT department if you do not have the proper rights on your PC.

Tool Options

The utility is a command line program. The table describes all the command line options; and the figure shows the `--help` output from the tool.

Table 11-1: Command Line Options

Command line Argument	Required?	Description
-p filename	Optional	Specifies Preloader file to write
-b filename	Optional	Specifies Cyclone V or Arria V Bootloader file to write
-B filename	Optional	Specifies Arria 10 Bootloader file to write
-a write	Required	Specifies action to take. Only "write" action is supported. Example: "-a write"
disk_file	Required(unless -d option is used)	Specifies disk image file or physical disk to write to. A disk image file is a file that contains all the data for a storage volume including the partition table. This can be written to a physical disk later with another tool. For physical disks in Linux, just specify the device file. For example: /dev/mmcblk0 For physical disks in Windows, specify the physical drive path such as \\.\physicaldrive2 or use the drive letter option(-d) to specify a drive letter. The drive letter option is the easiest method in Windows
-d	Optional	specify disk drive letter to write to. Example: "-d E". When using this option, the disk_file option cannot be specified.
-h	Optional	Displays help message and exits
--version	Optional	Displays script version number and exits

Sample Output from Utility

```
$ alt-boot-disk-util --help
Altera Boot Disk Utility
Copyright (C) 1991-2014 Altera Corporation

Usage:
#write preloader to disk
    alt-boot-disk-util -p preloader -a write disk_file

#write bootloader to disk
```

```
        alt-boot-disk-util -b bootloader -a write disk_file

    #write BOOTloader and PREloader to disk
        alt-boot-disk-util -p preloader -b bootloader -a write disk_file

    #write Arria10 Bootloader to disk
        alt-boot-disk-util -B a10bootloader -a write disk_file

    #write BOOTloader and PREloader to disk drive 'E'
        alt-boot-disk-util -p preloader -b bootloader -a write -d E



    Options:
      --version             show program's version number and exit
      -h, --help            show this help message and exit
      -b FILE, --bootloader=FILE
                            bootloader image file'
      -p FILE, --preloader=FILE
                            preloader image file'
      -a ACTION, --action=ACTION
                            only supports 'write' action'
      -B FILE, --a10-bootloader=FILE
                            arria10 bootloader image file
      -d DRIVE, --drive=DRIVE
                            specify disk drive letter to write to
```

SD Card Boot Utility Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Maintenance release.
August 2015	2015.08.06	Added Arria 10 support.

Linux Software Development Tools 12

2016.02.17

ug-1137 Subscribe Send Feedback

The Linux software development tools are listed below. The Linux compiler and the linux device tree generator are described in the following sections of this chapter.

- Linux compiler
- SD card boot utility
- Linux device tree generator (DTG)

Related Information

SD Card Boot Utility on page 11-1

Linux Compiler

The Linaro Linux compiler, version 4.8.3, is shipped with the SoC EDS.

For more information about the Linux compiler and to download the latest version of the tools, refer to the download page at the Linaro website.

The compiler is a GCC-based **arm-linux-gnueabihf** port. It targets the ARM processor, it assumes the target platform is running Linux, and it uses the GNU embedded-application binary interface (EABI) hard-float (HF) conventions.

The Linux compiler is installed as part of the ARM DS-5 AE, which is installed as part of the SoC EDS. The compilation tools are located at <*SoC EDS installation directory*>**/ds-5/bin**.

The Linux compiler comes with full documentation, located at <*SoC EDS installation directory*>**/ds-5/documents/gcc**. The documents are provided as HTML files. Some of the provided documents are:

- Compiler manual
- Assembler manual
- Linker manual
- Binutils manual
- GDB manual
- Getting Started Guide

Related Information

Linaro Website

Linux Device Tree Generator

A device tree is a data structure that describes the underlying hardware to an operating system - primarily Linux. By passing this data structure to the OS kernel, a single OS binary may be able to support many variations of hardware. This flexibility is particularly important when the hardware includes an FPGA.

The Linux Device Tree Generator (DTG) tool is part of Altera SoC EDS and is used to create linux device trees for SoC systems that contain FPGA designs created using Qsys. The generated device tree describes the HPS peripherals, selected FPGA Soft IP, and peripherals that are board dependent.

Note: The Arria 10 Bootloader also has an associated Device Tree called Bootloader Device Tree that is created and managed by the BSP Editor tool.

Related Information

- Altera Wiki
  For more information about the Linux DTG, refer to the Altera Wiki website.
- Linux Device Tree Generator User Guide
  For more details, navigate to the **Device Tree Generator User Guide** located on the **Device Tree Generator Documentation** page on the Rocketboards website.

Linux Software Development Tools Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Maintenance release.
August 2015	2015.08.06	Added Arria 10 support.

Support and Feedback 13

2016.02.17

ug-1137 Subscribe Send Feedback

Altera values your feedback. Please contact your Altera TSFAE or submit a service request at myAltera to report software bugs, potential enhancements, or obtain any additional information.

Related Information

myAltera Account Sign In

Support and Feedback Document Revision History

Date	Version	Changes
February 2016	2016.02.17	Maintenance release.
August 2015	2015.08.06	Added Arria 10 support.