Intel® SoC FPGA Embedded Development Suite User Guide

Updated for Intel® Quartus® Prime Design Suite: **19.1**

Subscribe
Send Feedback

ug-1137 | 2019.05.16
Latest document on the web: PDF | HTML

Contents

1. Introduction to the Intel® SoC FPGA EDSSoC FPGA Embedded Development Suite

The Intel® SoC FPGA Embedded Development Suite (SoC EDS) is a comprehensive tool suite for embedded software development on Intel FPGA SoC devices.

The SoC EDS contains development tools, utility programs, run-time software, and application examples that enable firmware and application software development on Intel SoC hardware platforms.

The SoC EDS is offered in two editions:

- **SoC EDS Standard Edition**—The SoC EDS Standard Edition targets the Arria® V SoC, Cyclone® V SoC, and Intel Arria 10 SoC and must be used only with FPGA projects created in Intel Quartus® Prime Standard Edition. Although the Intel FPGA SoC devices and Intel Quartus Prime Standard Edition support Intel Arria 10 SoC, it is recommended to use the SoC EDS Professional Edition of both tools for new Intel Arria 10 SoC Projects.
- **SoC EDS Professional Edition**—The SoC EDS Professional Edition targets the Intel Arria 10 SoC and the Intel Stratix® 10 SoC and must be used only with FPGA projects created in Intel Quartus Prime Pro Edition.

  Note: The SoC EDS Professional Edition does not support Cyclone V SoC and Arria V SoC, as they are not supported by Intel Quartus Prime Pro Edition.

Related Information

- SoCEDSGettingStarted
  For more information about getting started, refer to the SoCEDSGettingStarted page on the Intel FPGA Wiki.
- Intel SoC FPGA EDS Licensing on page 15

1.1. Tool Versions

Table 1. Tool Versions

Refer to this table when you want to determine the tool versions.

Tool	Version
Mentor Graphics* BareMetal GCC Compiler	6.2.0 (Sourcery CodeBench Lite 2016.11-88)
Linux* Compiler	4.8.3 (Linaro* GCC 4.8-2014.04)
Arm* Compiler 5	5.06 update 6
Arm Compiler 6	6.10.1
Arm Development Studio 5* (DS-5*) Intel SoC FPGA Edition	5.29.2

1.2. Differences Between Standard and Professional Editions

The main difference between the two different editions exists in the SoC devices that are supported, as mentioned in the previous section.

Things to know about installation and licensing for the two editions:

- The default installation paths are different between editions. For details on the default installation paths, refer to the "Installing the Intel SoC FPGA EDS" chapter.
- There are no licensing differences between SoC EDS editions. The same DS-5 Intel SoC FPGA Edition License works for both Standard and Pro editions. For details about licensing, refer to the "Intel SoC FPGA EDS Licensing" chapter.

Related Information

- Installing the Intel SoC FPGA EDS on page 12
- Intel SoC FPGA EDS Licensing on page 15
- Linux Compiler on page 100

1.3. Overview

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

- Board bring-up
- Device driver development
- Operating system (OS) porting
- BareMetal 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 Intel SoC FPGA Edition AE Toolkit
- Compiler tool chains:
  - BareMetal GNU Compiler Collection (GCC) tool chain from Mentor Graphics
  - Arm BareMetal compiler 5
  - Arm BareMetal compiler 6
  - Linux GCC compiler tool chain from Linaro
- 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 Bootloaders
    - Preloader and bootloader for Arria V SoC and Cyclone V SoC
    - Bootloader for Intel Arria 10 SoC
    - Bootloader for Intel Stratix 10 SoC
- Embedded command shell allowing easy invocation of the included tools
- SD Card Boot Utility
- Intel Quartus Prime Programmer and Signal Tap

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 *Golden System Reference Design User Manual* available on the Rocketboards website.

Related Information

- Golden System Reference Design User Manual
- GitHub Repository

1.3.1. Linux Device Tree Binary

SoC EDS includes Linux Device Tree Binary (DTB) files that enable Linux to load the drivers for the IP located in the FPGA fabric:

- Cyclone V SoC: `<SoC EDS installation directory>/examples/hardware/cv_soc_devkit_ghrd/soc_system.dtb`
- Arria V SoC: `<SoC EDS installation directory>/examples/hardware/av_soc_devkit_ghrd/ghrd_5astfd5k3.dtb`
- Intel Arria 10 SoC: `<SoC EDS installation directory>/examples/hardware/a10_soc_devkit_ghrd/ghrd_10as066n2.dtb`

Note: These files are created with the aid of the Linux Device Tree Generator.

Note: A DTB file is not provided for Intel Stratix 10 SoC, as it is not supported by the Linux Device Tree Generator.

Related Information

Linux Device Tree Generator on page 101

1.3.2. Intel Stratix 10 SoCStratix 10 Golden Hardware Reference Design

The GHRD included with the SoC EDS v19.1 targets the device **1SX280LU2F50E2VG**, while the customer-available Intel Stratix 10 SoC Development Kits are populated with the device **1SX280LU2F50E2VGS2**. The GHRD can be updated to match the device on the Intel Stratix 10 SoC Development Kit with the following commands executed from an Embedded Command Shell on a Linux computer where both SoC EDS Professional Edition 19.1 and Intel Quartus Prime Pro Edition 19.1 are installed:

```
mkdir s10_soc_devkit_ghrd
cd s10_soc_devkit_ghrd
tar xf $SOCEDS_DEST_ROOT/examples/hardware/ \
s10_soc_devkit_ghrd/tgz/*.tar.gz
sed -i 's/QUARTUS_DEVICE := 1SX280LU2F50E2VG/QUARTUS_DEVICE \
:= 1SX280LU2F50E2VGS2/g' Makefile
make clean
make scrub_clean
rm -rf output_files qsys_top subsys_periph subsys_jtg_mst \
ip/qsys_top/ ip/subsys_jtg_mst/ ip/subsys_periph/ *.qsys *.qpf *.qsf
make generate_from_tcl
make sof
```

1.4. 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 **Intel SoC FPGA Embedded Development Suite** page.

Table 2. 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	BareMetal Developer	RTOS Developer	Linux Kernel and Driver Developer	Linux Application Developer
Arm DS-5 Intel SoC FPGA Edition Debugging	√	√	√	√	√
Arm DS-5 Intel SoC FPGA Edition Tracing		√	√	√	
Arm DS-5 Intel SoC FPGA Edition Cross Triggering		√	√	√	
Hardware Libraries		√	√	√	
Second Stage Bootloader Generator	√	√	√	√	
Flash Programmer		√	√	√	√
BareMetal Compiler	√	√	√	√	
Linux Compiler				√	√
Linux Device Tree Generator				√	

Related Information

SoC Embedded Development Suite web page

1.4.1. Hardware Engineer

As a hardware engineer, you typically design the FPGA hardware in Platform Designer (Standard). You can use the debugger of Arm DS-5 Intel SoC FPGA Edition 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 Intel SoC FPGA Edition 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 Arm DS-5 Intel SoC FPGA Edition, which is part of the paid Intel Intel SoC FPGA EDS license.

For more information, refer to the *Intel SoC FPGA EDS Licensing* section.

Related Information

- Intel SoC FPGA EDS Licensing on page 15
- Hardware – Software Development Flow on page 10

1.4.2. BareMetal and RTOS Developer

As a BareMetal or a RTOS developer, you need JTAG debugging and low-level visibility into the system. The following tasks require JTAG debugging, which is enabled only in the Arm DS-5 Intel SoC FPGA Edition.

- To compile your code and the SoC Hardware Library to control the hardware in a convenient and consistent way, use the BareMetal compiler.
- To program the flash memory on the target board, use the Flash Programmer.

For more information, see the "Licensing" section.

Related Information

Intel SoC FPGA EDS Licensing on page 15

1.4.3. 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 BareMetal 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 Arm DS-5 Intel SoC FPGA Edition.

For more information, see the "Licensing" section.

Related Information

Intel SoC FPGA EDS Licensing on page 15

1.4.4. 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 about the threads that are running and the drivers that are loaded.

These tasks do not require JTAG debugging. You can perform these tasks with both the paid and free SoC EDS FPGA licenses. For more information, see the "Licensing" section.

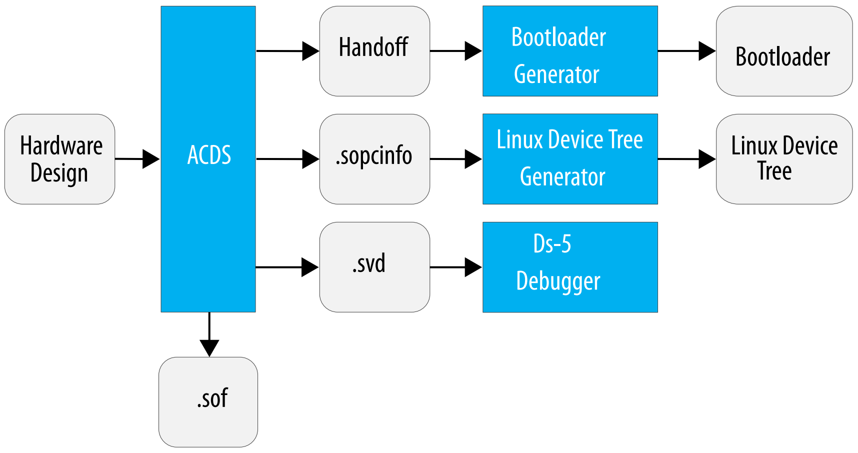
Related Information

Intel SoC FPGA EDS Licensing on page 15

1.5. Hardware – Software Development Flow

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

Figure 1. Intel 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. 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" section.
- **.svd** file – contains descriptions of the HPS registers and of the soft IP registers on the FPGA side implemented in the FPGA portion of the device. The **.svd** file contains register descriptions for the HPS peripheral registers and soft IP components in the FPGA portion of the SoC. This file is used by the Arm DS-5 Intel SoC FPGA Edition Debugger to allow you to inspect and modify these registers.
- **.sopcinfo** file – contains a description of the entire system. 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" section.

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

  Note: For Intel Stratix 10 SoC, the handoff information is part of the configuration bitstream, and the bootloader has direct access to it. Because of that there is no need for a Bootloader Generator tool in this case.

Related Information

- BSP Generation Flow on page 56
- SoC Embedded Development Suite Download Page
- Linux Device Tree Generator on page 101

1.6. Introduction to the Intel SoC FPGA EDS Revision History

Document Version	Changes
2019.05.16	• *Tool Versions* section: Updated the Arm Development Studio 5 (DS-5) Intel SoC FPGA Edition version. • *Intel Stratix 10 SoC SoC Golden Hardware Reference Design* section: Updated the SoC EDS Professional Edition and Intel Quartus Prime Pro Edition versions.
2018.09.24	Updated the Tool versions for Arm Compiler 6 and Arm Development Studio 5 (DS-5) Intel SoC FPGA Edition
2018.06.18	• Updated chapter to include support for Intel Stratix 10 SoC • Replaced "Second Stage Bootloader" with "Bootloader". • For the Golden Hardware Reference Designs (GHRD), added the precompiled bootloader for Intel Stratix 10 SoC • Added the "Intel Stratix 10 SoC Golden Hardware Reference Design" section
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Renamed the Web and Subscription Editions to align with Quartus Prime naming
2016.02.17	Maintenance release
2015.08.06	Added Intel Arria 10 SoC support

2. Installing the Intel SoC FPGA EDS

You must install the SoC EDS and the Arm DS-5 Intel SoC FPGA Edition in order to run the SoC EDS on an Intel SoC hardware platform.

2.1. Installation Folders

The default installation folder for SoC EDS, referred to as <SoC FPGA EDS installation directory> throughout this document, is:

- Standard Edition:
  - `c:\intelFPGA\19.1\embedded` on Windows
  - `~/intelFPGA/19.1/embedded` on Linux
- Professional Edition:
  - `c:\intelFPGA_pro\19.1\embedded` on Windows
  - `~/intelFPGA_pro/19.1/embedded` on Linux

The default installation folder for the Intel Quartus Prime Programmer, referred to as <Quartus Prime installation directory> throughout this document, is:

- Standard Edition:
  - `c:\intelFPGA\19.1\qprogrammer` on Windows
  - `~/intelFPGA/19.1/qprogrammer` on Linux
- Professional Edition:
  - `c:\intelFPGA_pro\19.1\qprogrammer` on Windows
  - `~/intelFPGA_pro/19.1/qprogrammer` on Linux

2.2. Installing the SoC EDS on Windows

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

1. Download the latest installation program from the *Intel SoC FPGA Embedded Development Suite Download Center* page of the Intel FPGA website.
2. Run the installer to open the **Installing Intel SoC FPGA Embedded Development 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: It is recommended to accept the default installation paths for both Intel Quartus Prime and SoC EDS software, to allow them to properly 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 Intel SoC FPGA Edition installation, and click **Finish**.

Related Information

SoC Embedded Development Suite Download Center

2.2.1. Installing Cygwin on Windows*

Starting with SoC EDS v19.1, the Windows* version of SoC EDS requires the Cygwin component to be manually installed. You need to have administrative privileges on the PC in order to perform the Cygwin installation.

The procedure for installing Cygwin is:

1. Go to the official Cygwin website https://cygwin.com/.
2. Download the 64bit installer (`setup-x86_64.exe`)
3. Open a Windows Command Prompt.
4. Run the installer in the Command Prompt:
    a. Copy the following command in a text editor:

```
<Path to Cygwin Installer>\setup-x86_64.exe --wait --quiet-mode --root
      C:\intelFPGA_pro\19.1\embedded\host_tools\cygwin --site http://
cygwin.mirrors.hoobly.com
      --packages
      make,gcc-core,gcc-g+
+,ncurses,inetutils,openssh,mosh,patch,flex,bison,tar,bzip2,zip,unzip,ut
il-linux,git,subversion,vim,xxd,m4,wget,dos2unix,libintl-
devel,diffutils,libncurses-devel,iperf,xorg-server,xinit,mingw64-x86_64-
gcc-core,mingw64-x86_64-gcc-g++
```

    b. Make sure the command is all on a single line. Copying and pasting from this document may split the command in multiple lines.
    c. Replace `<Path to Cygwin Installer>` with the full path to where you downloaded the installer.
    d. Replace `C:\intelFPGA_pro\19.1\embedded\host_tools\cygwin` with the equivalent path if SoC EDS was installed in a non-default location.
    e. Copy the command and paste it in the Command Prompt.
5. Press Enter to issue the command and wait until everything is automatically installed

2.3. Installing the SoC EDS on Linux

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 *Intel SoC FPGA Embedded Development Suite Download Center* page on the Intel FPGA website.

2.4. Installing the Arm Development Studio 5 (DS-5) Intel SoC FPGA EditionArm DS-5 Intel SoC FPGA Edition Toolkit

For the last step of the SoC EDS installation process, start the Arm Development Studio 5 Intel SoC FPGA Edition 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 Development Studio 5 Intel SoC FPGA Edition, 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 Development Studio 5 Intel SoC FPGA Edition installation is complete.
8. Click **Finish**.

2.5. Installing the Intel SoC FPGA EDS Document Revision History

Document Version	Changes
2019.05.16	• *Installation Folders* section: Updated the path names for the 19.1 release • Created two new sections called *Installing Cygwin on Windows* and *Installing the SoC EDS on Windows*.
2018.09.24	Updated the path names for the 18.1 release.
2018.06.18	Maintenance release
2018.05.07	Maintenance release
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Updated the installation path to 15.1
2015.08.06	Added Intel Arria 10 SoC support

3. Intel SoC FPGA EDS Licensing

The only SoC EDS component that requires a license is the Arm DS-5 Intel SoC FPGA Edition. The rest of the SoC EDS components, such as tools and software, do not require a license. This is the same for both Standard and Professional editions of SoC EDS.

The Arm DS-5 Intel SoC FPGA Edition has three different license options:

- Arm DS-5 Intel SoC FPGA Edition License
- Arm DS-5 Intel SoC FPGA Edition Community Edition License
- 30-day Evaluation of Arm DS-5 Intel SoC FPGA Edition License

The following table shows the different debugging capabilities of the Arm DS-5 Intel SoC FPGA Edition Licenses:

Table 3. Comparing Arm DS-5 Intel SoC FPGA Edition Licenses

Licensing Option	Debugging Scenarios Enabled
Arm DS-5 Intel SoC FPGA Edition Community Edition License	• Linux application debugging over ethernet
Arm DS-5 Intel SoC FPGA Edition License, including the 30 day evaluation version	• JTAG-based BareMetal Debugging • JTAG-based Linux Kernel and Driver Debugging • Linux Application Debugging over Ethernet

Based on the specific Arm DS-5 Intel SoC FPGA Edition License that is used, the SoC FPGA EDS has two different licensing options:

- SoC EDS Paid license - using Arm DS-5 Intel SoC FPGA Edition license
- SoC EDS Free license - using Arm DS-5 Intel SoC FPGA Edition Community Edition license

Note: For a detailed comparison chart, refer to the "Compare Editions" section on the *Intel SoC FPGA Embedded Development Suite* web page.

Related Information

SoC Embedded Development Suite web page

3.1. Getting the License

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

Arm DS-5 Intel SoC FPGA Edition License - If you have purchased a SoC development kit or a stand-alone license for DS-5 Intel SoC FPGA Edition AE, then you have already received an Arm license serial number. This is a 15-digit alphanumeric string with two dashes in between. Use this serial number to activate your license in DS-5 Intel SoC FPGA Edition, as shown in the "Activating the License" section.

Note: This license contains one-year of Support and Maintenance from Arm starting at the date of purchase or renewal.

DS-5 Intel SoC FPGA Edition Community Edition License - If you are using the SoC EDS Lite Edition, you are able to use the DS-5 Intel SoC FPGA Edition perpetually to debug Linux applications over an Ethernet connection. Get your Arm license activation code from the "DS-5 Intel SoC FPGA Edition Community Edition" section on the DS-5 Intel SoC FPGA Edition Development Studio page on the Arm website and then activate your license in DS-5 Intel SoC FPGA Edition, as shown in the "Activating the License" section.

30-Day Evaluation of DS-5 Intel SoC FPGA Edition License - If you want to evaluate the DS-5 Intel SoC FPGA Edition, you can get a 30-Day Evaluation activation code from the "DS-5 Intel SoC FPGA Edition 30-Day Evaluation" section on the DS-5 Intel SoC FPGA Edition Development Studio page on the Arm website and then activate your license in Arm DS-5 Intel SoC FPGA Edition, as shown in the "Activating the License" section.

Related Information

- SoC FPGA EDS Download Page
- Activating the License on page 16
- DS-5 Community Edition License
  For more information about obtaining your ARM license activation code for the DS-5 Community Edition license
- DS-5 Intel FPGA Edition 30-Day Evaluation
  For more information about obtaining your activation code for the DS-5 Intel FPGA Edition 30-Day Evaluation license

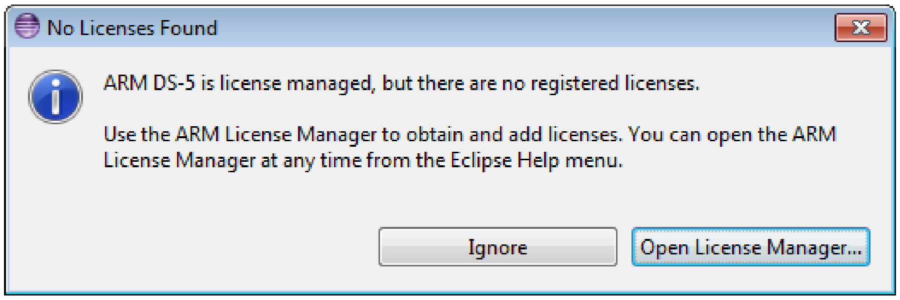
3.2. Activating the License

This section presents the steps required for activating the license in the Arm DS-5 Intel SoC FPGA Edition 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 Arm DS-5 Intel SoC FPGA Edition 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 Intel SoC FPGA Edition is run, it notifies you that it requires a license. Click the **Open License Manager** button.

Figure 2. No License Found



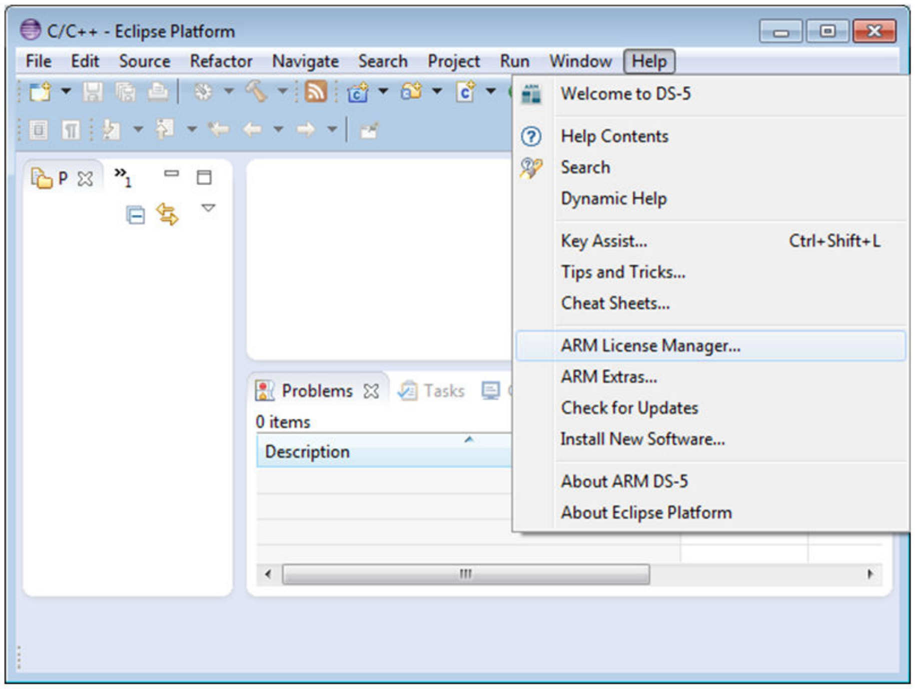
No Licenses Found

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.

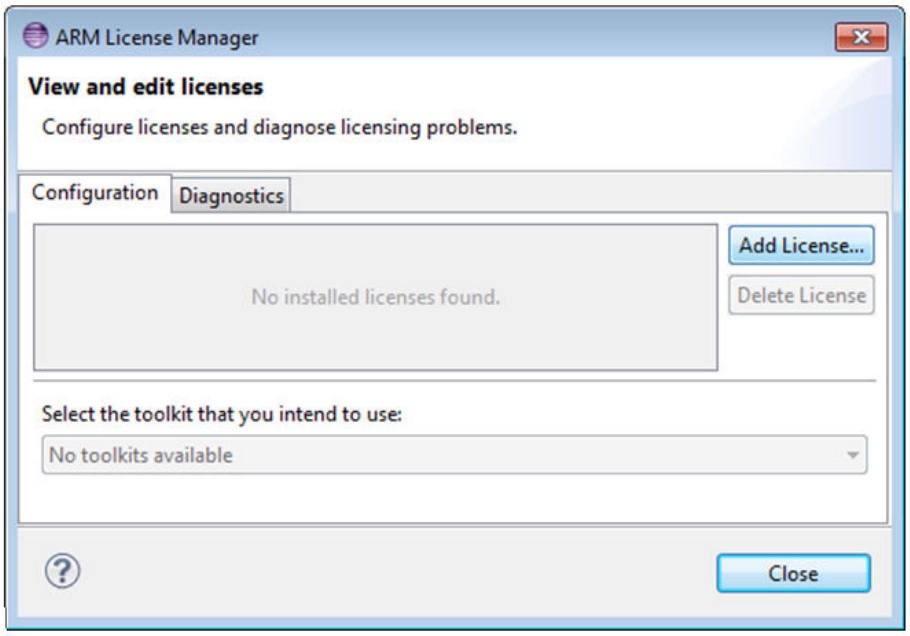
2. If at any time it is required to change the license, select **Help ➤ Arm License Manager** to open the **License Manager**.

Figure 3. 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 4. Arm License Manager



View and edit licenses

Configure licenses and diagnose licensing problems.

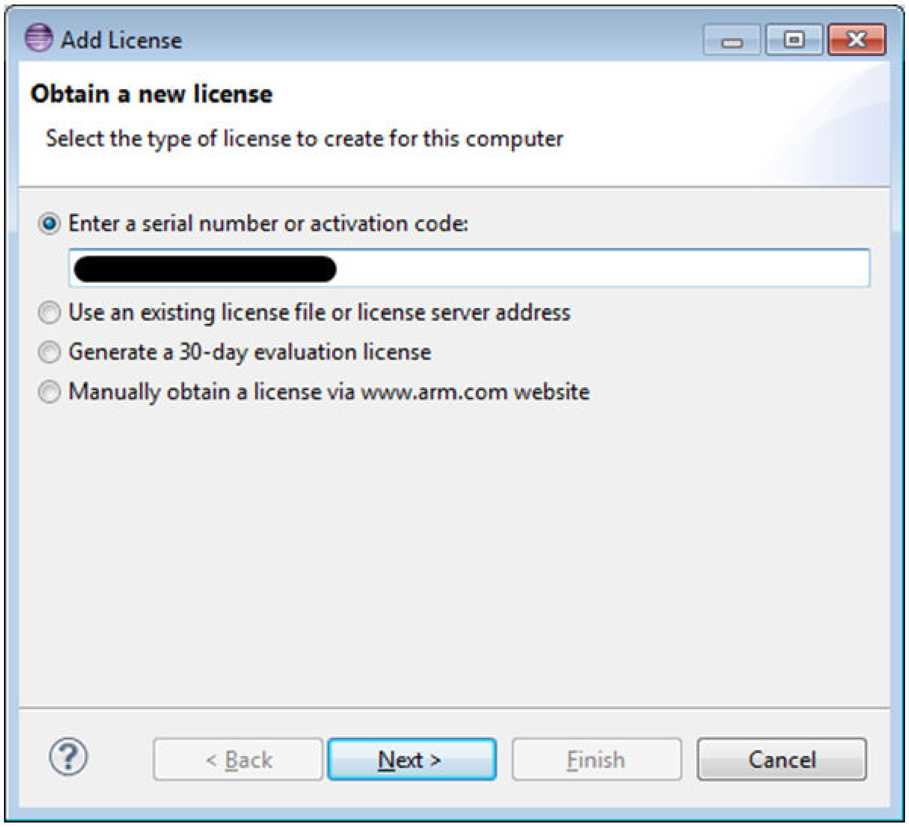
No installed licenses found.

Select the toolkit that you intend to use:

No toolkits available

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 DS-5 Intel SoC FPGA Edition—Enter the Arm License Number.
   b. Arm DS-5 Intel SoC FPGA Edition Community Edition and 30-day Evaluation of Arm DS-5 Intel SoC FPGA Edition—Enter the Arm License Activation Code.

Figure 5. Add License - Obtain a New License



Obtain a new license

Select the type of license to create for this computer

(●) Enter a serial number or activation code:

[redacted]

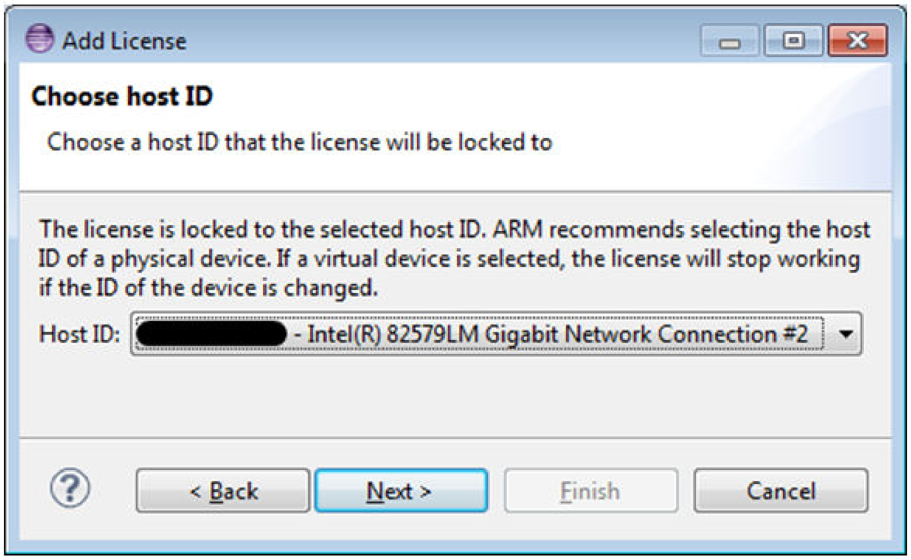
( ) Use an existing license file or license server address

( ) 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 6. Add License - Choose host ID



Add License

Choose host ID

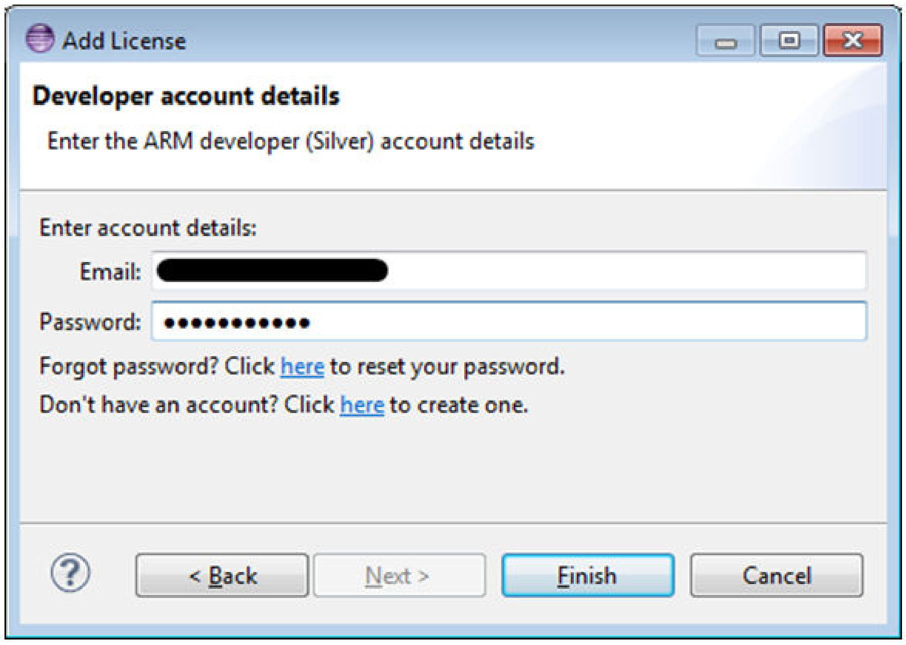
Choose a host ID that the license will be locked to

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 7. 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.

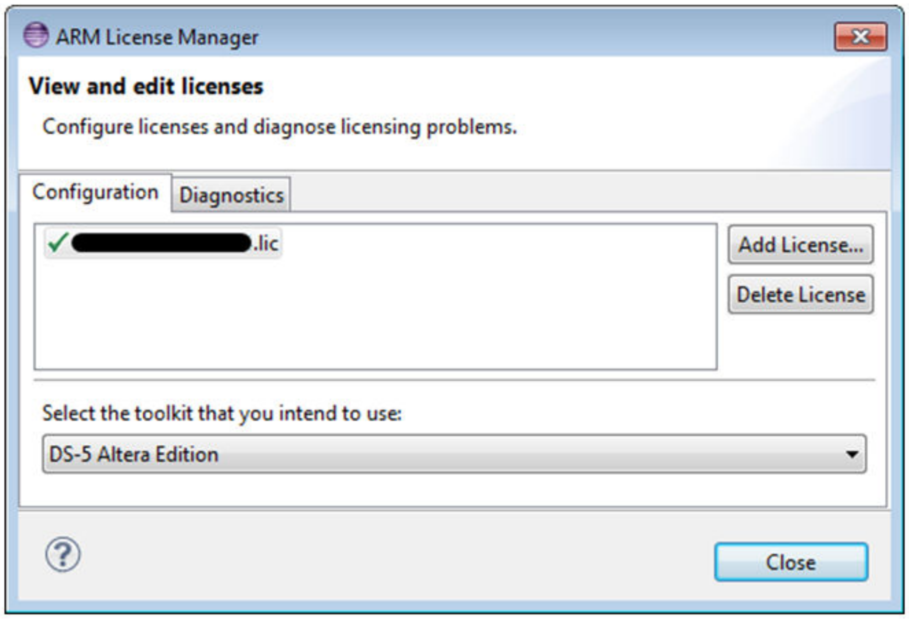
Note: The License Manager needs to be able to connect to the Internet in order to activate the license. If you do not have an Internet connection, you need to write down your Ethernet MAC address and generate the license directly from the *Arm Self-Service* web page on the Arm website, then select the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

Note: Only the Arm DS-5 Intel SoC FPGA Edition, with an associated license number can be activated this way. The Arm DS-5 Intel SoC FPGA Edition Community Edition and 30-day Evaluation of Arm DS-5 Intel SoC FPGA 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 Intel SoC FPGA Edition` activates the license and display it in the **License Manager**. Click **Close**.

Figure 8. Arm License Manager


ARM License Manager

View and edit licenses

Configure licenses and diagnose licensing problems.

✓ [redacted].lic

Select the toolkit that you intend to use:

DS-5 Altera Edition

Related Information

- ARM website
- Getting the License on page 15

3.3. Intel SoC FPGA EDS Licensing Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Maintenance release
2018.06.18	Maintenance release

continued...

Document Version	Changes
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Renamed the Web and Subscription Editions to align with Quartus Prime naming
2016.02.17	Maintenance release
2015.08.06	Added Intel Arria 10 SoC support

4. Embedded Command Shell

The purpose of the embedded command shell is to enable 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 from the **Start** menu or by running:

```
<SoC EDS installation directory>\Embedded_Command_Shell.bat
```

On Linux, the embedded command shell is started by running:

```
<SoC EDS installation directory>/embedded_command_shell.sh
```

4.1. Embedded Command Shell Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Maintenance release
2018.06.18	Maintenance release
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Maintenance release
2015.08.06	Added Intel Arria 10 SoC support

5. Getting Started Guides

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

- Board Setup
- 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
For more information about getting started, refer to the SoCEDSGettingStarted page on the Intel FPGA Wiki.

5.1. Getting Started Guides Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Maintenance release
2018.06.18	Maintenance release
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Maintenance release
2015.08.06	Removed content from this section and moved it to the Intel FPGA Wiki

6. Arm Development Studio 5 Intel SoC FPGA EditionArm DS-5 Intel SoC FPGA Edition

The Arm DS-5 Intel SoC FPGA Edition is a device-specific exclusive offering from Intel; and is a powerful Eclipse-based comprehensive Integrated Development Environment (IDE).

For the current version, refer to Table 1 on page 5.

Some of the most important provided features are:

- File editing, supporting syntax highlighting and source code indexing
- Build support, based on makefiles
- BareMetal 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 Intel FPGA Download Cable II

The Arm Development Studio 5 Intel SoC FPGA Edition 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 Development Studio 5 Intel SoC FPGA Edition reference material from Eclipse, by navigating to **Help ➤ Help Contents ➤ Arm DS-5 Intel SoC FPGA Edition Documentation** or on the Arm website.

Related Information

- Online ARM DS-5 Documentation
  The ARM DS-5 for Intel SoC FPGA reference material can be accessed online on the documentation page of the ARM website.
- Tool Versions on page 5

6.1. Starting Eclipse Arm Development Studio 5 Intel SoC FPGA Edition

Arm Development Studio 5 Intel SoC FPGA Edition 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 Development Studio 5 Intel SoC FPGA Edition uses, you must type **eclipse &** at the command line.

6.2. BareMetal Project Management

Arm Development Studio 5 Intel SoC FPGA Edition enables convenient project management for BareMetal projects using two different methods:

- Using Makefiles
- Using the Arm Development Studio 5 Intel SoC FPGA Edition 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 DS-5 Intel SoC FPGA Edition Compiler BareMetal projects.

Method	Advantages	Compiler Toolchain Support
Makefile	Scripted compilation	GCC
Arm Development Studio 5 Intel SoC FPGA Edition graphical interface	Multiple toolchain support	GCC, Arm DS-5 Intel SoC FPGA Edition Compiler

6.2.1. BareMetal Project Management Using Makefiles

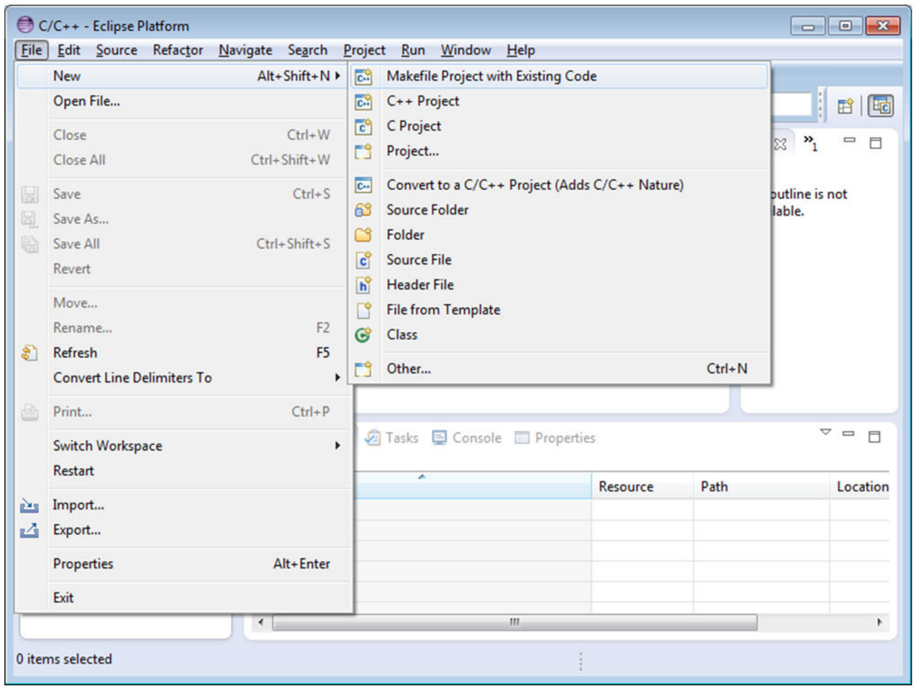
The Arm Development Studio 5 Intel SoC FPGA Edition enables convenient project management using makefiles. The sample projects that are provided with SoC EDS use makefiles to manage the build process.

Note: This option refers to just the DS-5 Intel SoC FPGA Edition specific aspects. If you are not familiar with defining and using makefiles, please use the Arm Development Studio 5 Intel SoC FPGA Edition GUI option detailed in the next section.

To allow Arm Development Studio 5 Intel SoC FPGA Edition to manage a makefile-based project, create a project, as follows:

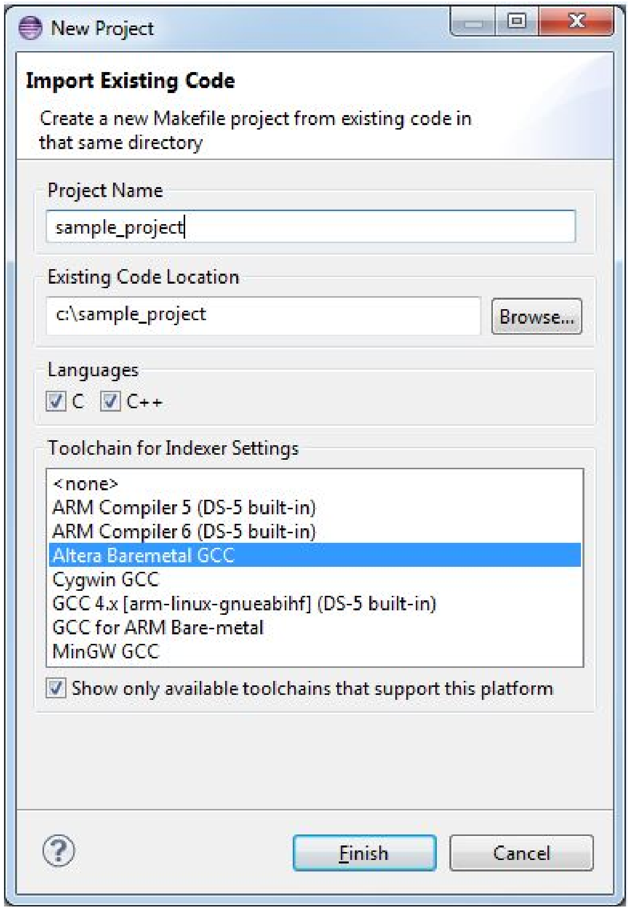
1. Create a folder on the disk.
2. Create the project by selecting **File ➤ New ➤ Makefile Project with Existing Code**.

Figure 9. Creating a Project with Existing Code



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

Figure 10. 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

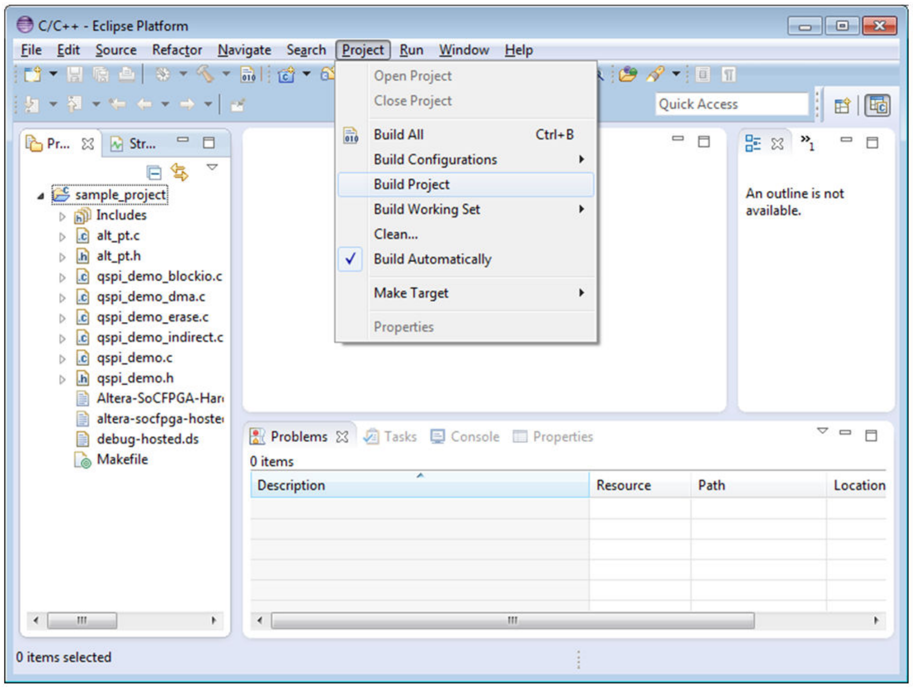
Languages: [✓] C [✓] C++

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 Development Studio 5 Intel SoC FPGA Edition 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 11. Eclipse IDE - Build Process Invoked - Building a Software Project in Arm Development Studio 5 Intel SoC FPGA Edition



If the compilation tools issue errors, Arm Development Studio 5 Intel SoC FPGA Edition parses and formats them for you and displays them in the Problems view.

5. Build a software project in Arm Development Studio 5 Intel SoC FPGA Edition.

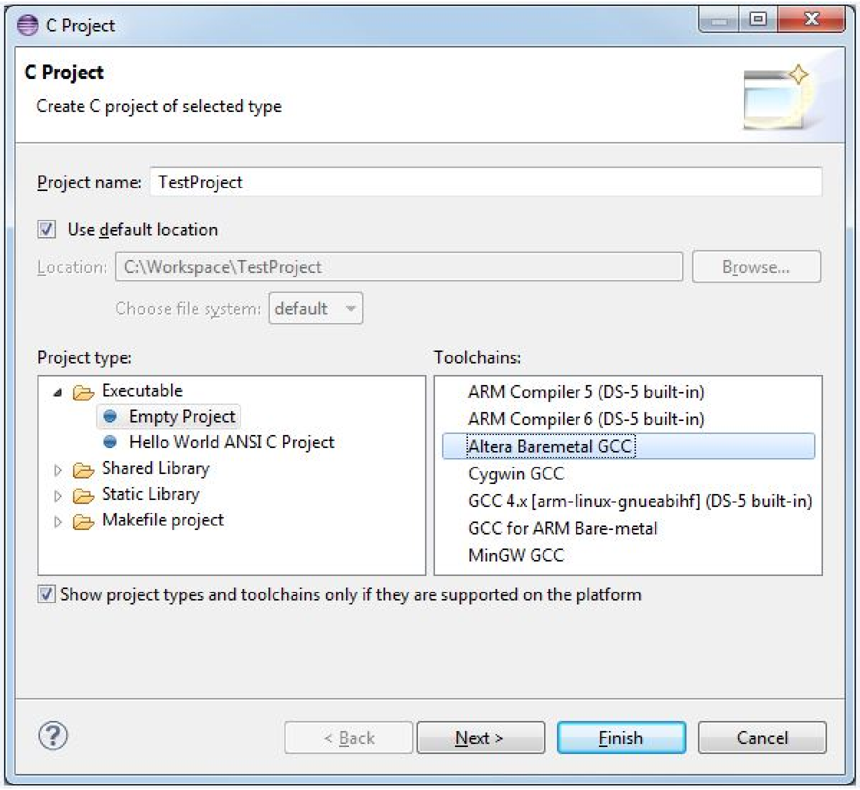
6.2.2. GCC-Based BareMetal Project Management

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

6.2.2.1. Creating Project

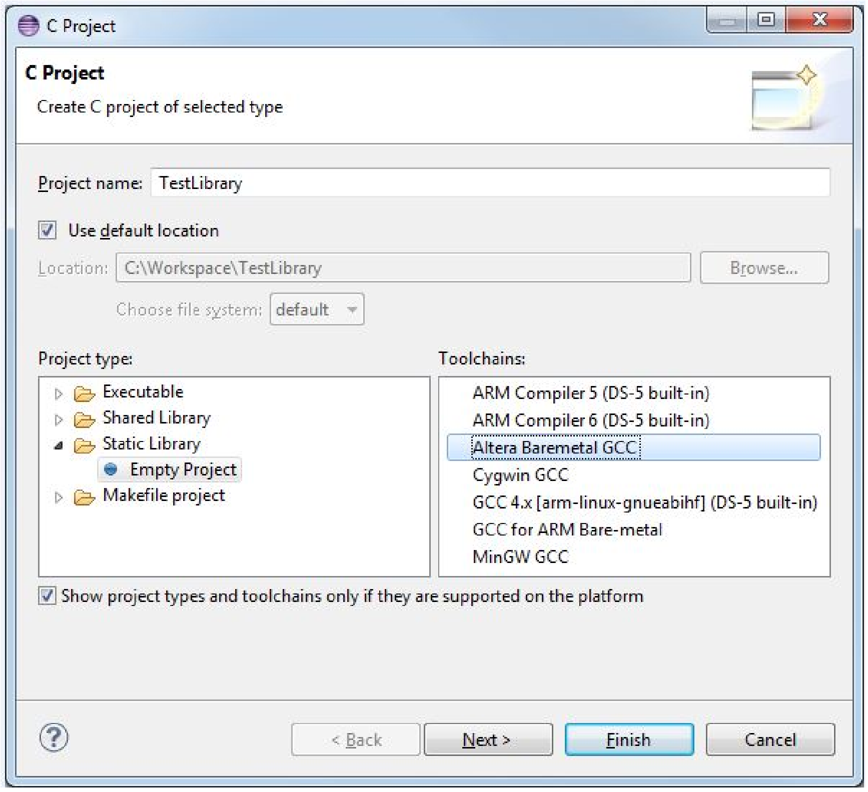
1. Start Eclipse.
2. Go to **File ➤ New C Project**.
3. Determine if you want to create an **Executable** empty project or a **BareMetal library** empty project.
   a. **BareMetal executable**
   Select **Project Type** to be **Executable ➤ Empty Project** then **Toolchain** to be **Intel FPGA Baremetal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

Figure 12. BareMetal 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 **BareMetal Library ➤ Empty Project** and click Finish.

Figure 13. BareMetal Library Project Type


C Project

C Project

Create C project of selected type

Project name: TestLibrary

☑ Use default location

Location: C:\Workspace\TestLibrary Browse...

Choose file system: default

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

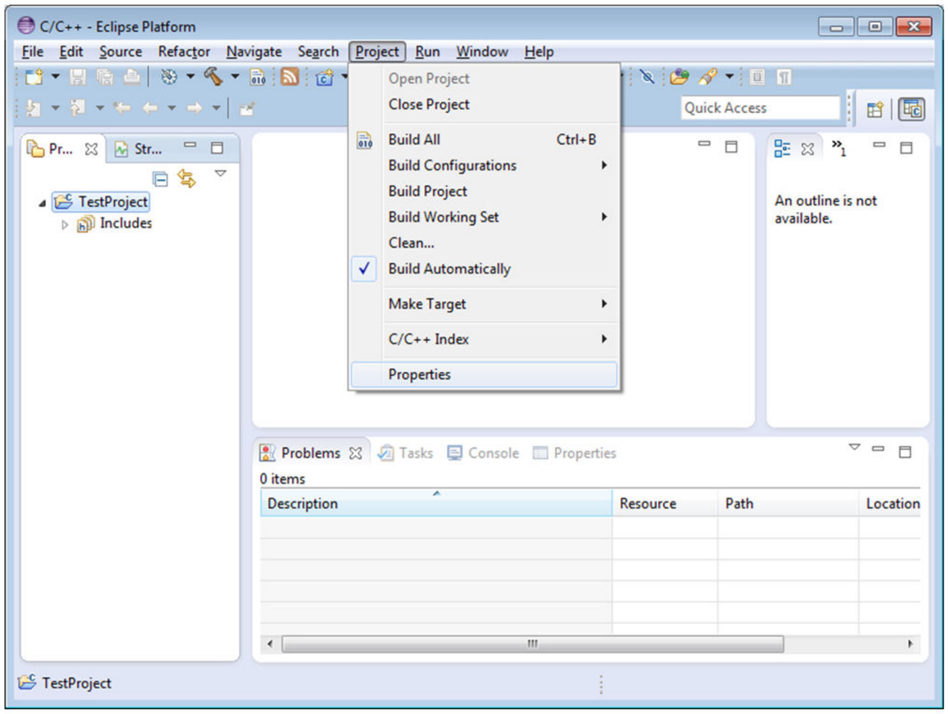
Toolchains:
- ARM Compiler 5 (DS-5 built-in)
- ARM Compiler 6 (DS-5 built-in)
- Altera Baremetal GCC
- Cygwin GCC
- GCC 4.x [arm-linux-gnueabihf] (DS-5 built-in)
- GCC for ARM Bare-metal
- MinGW GCC

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

6.2.2.2. Build Settings

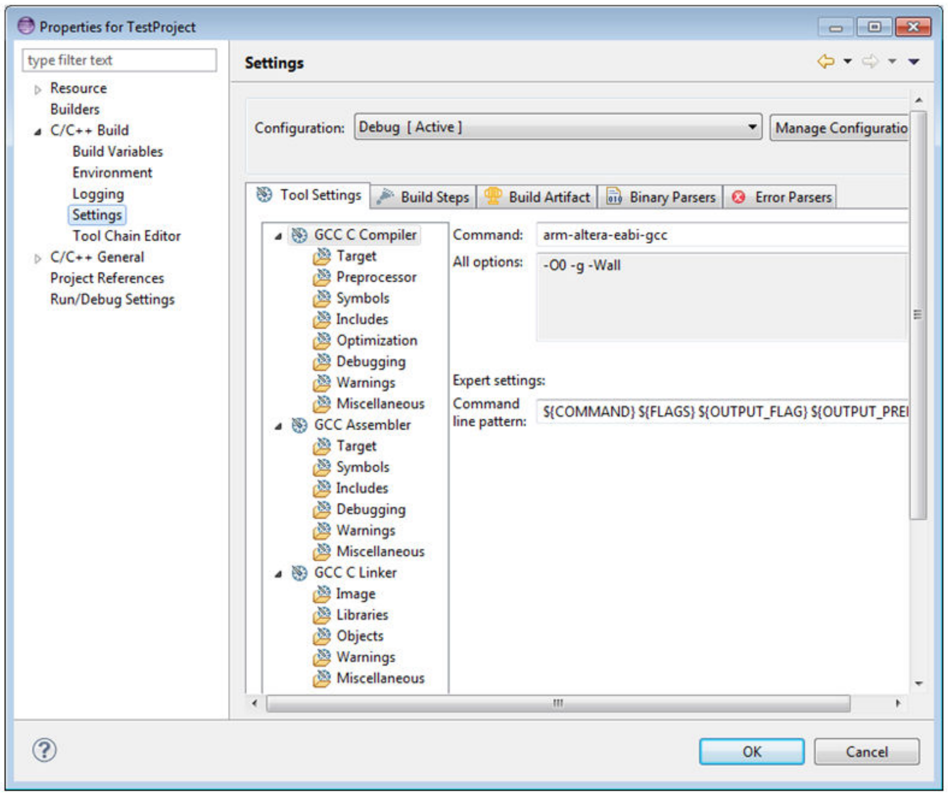
Once the project is created, the project properties can be accessed by going to **Project ➤ Properties**.

Figure 14. Project Properties



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

Figure 15. 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_PREI

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

- Compiler
- Assembler
- Linker

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

Related Information

Getting Started Guides on page 24

6.2.3. Arm Compiler 5 BareMetal Project Management

The Arm Compiler 5 is shipped with the SoC EDS.

For the current version, refer to Table 1 on page 5. The Arm DS-5 Intel SoC FPGA Edition can be used to manage Arm compiler projects in a GUI environment.

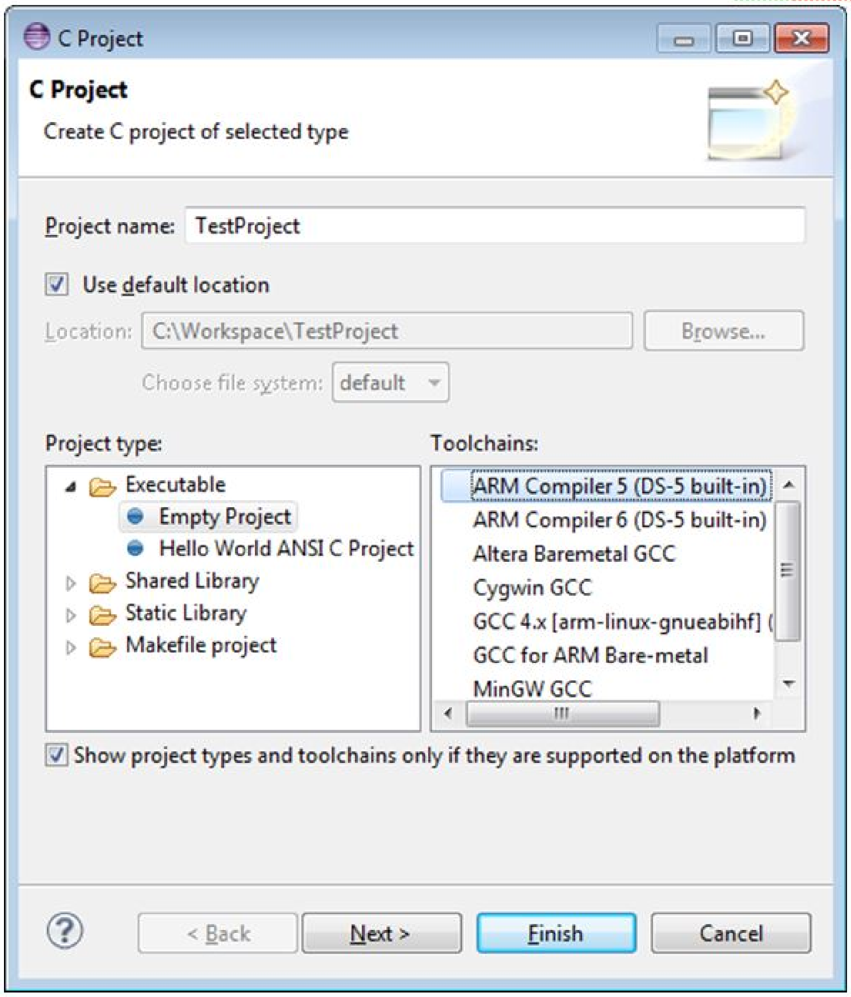
Related Information

Tool Versions on page 5

6.2.3.1. 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 16. Create an Empty Arm Compiler BareMetal Executable Project



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

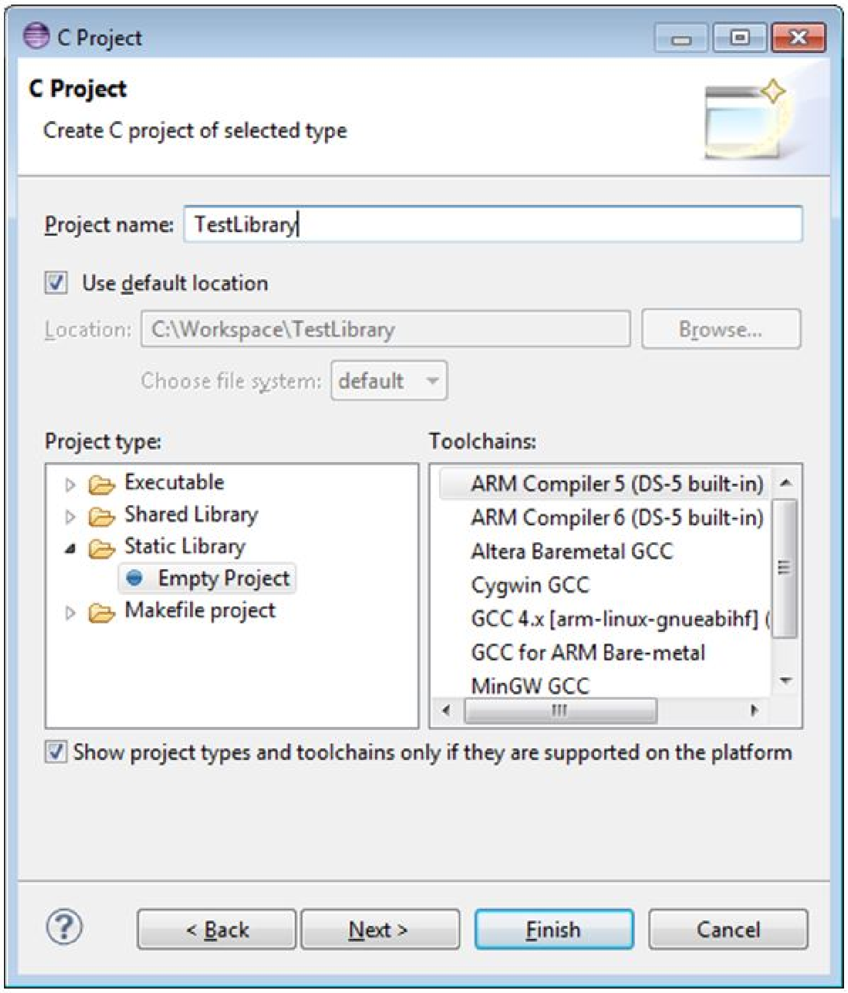
Project type:
- Executable
  - Empty Project
  - Hello World ANSI C Project
- Shared Library
- Static Library
- Makefile project

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] (
- 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 17. Create an Empty Arm Compiler BareMetal Library Project


C Project

C Project

Create C project of selected type

Project name: TestLibrary

☑ Use default location

Location: C:\Workspace\TestLibrary Browse...

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

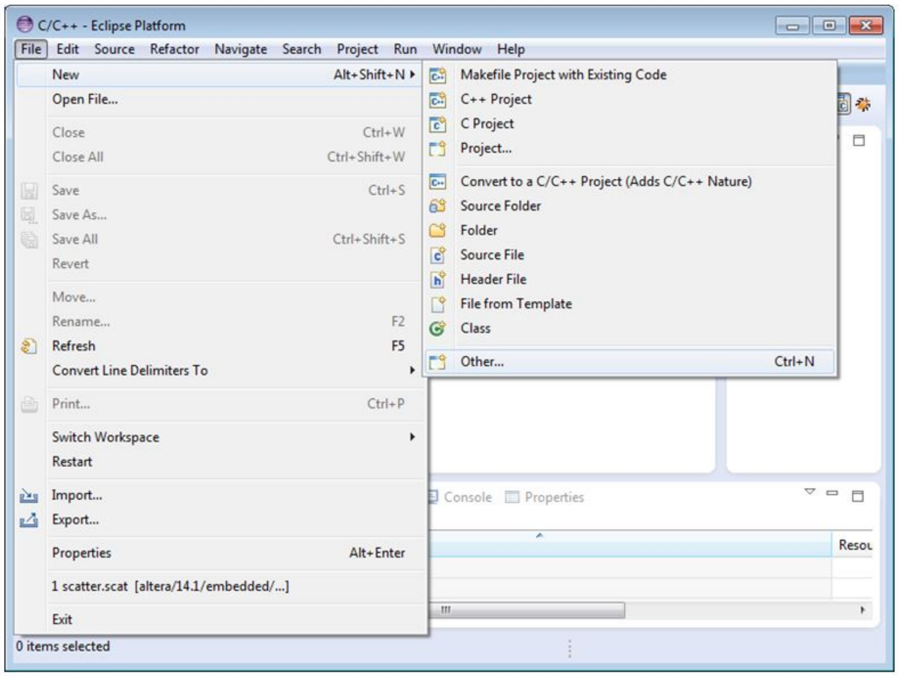
☑ Show project types and toolchains only if they are supported on the platform

6.2.3.2. Linker Script

Arm Development Studio 5 Intel SoC FPGA Edition offers a visual tool to help create linker scripts.

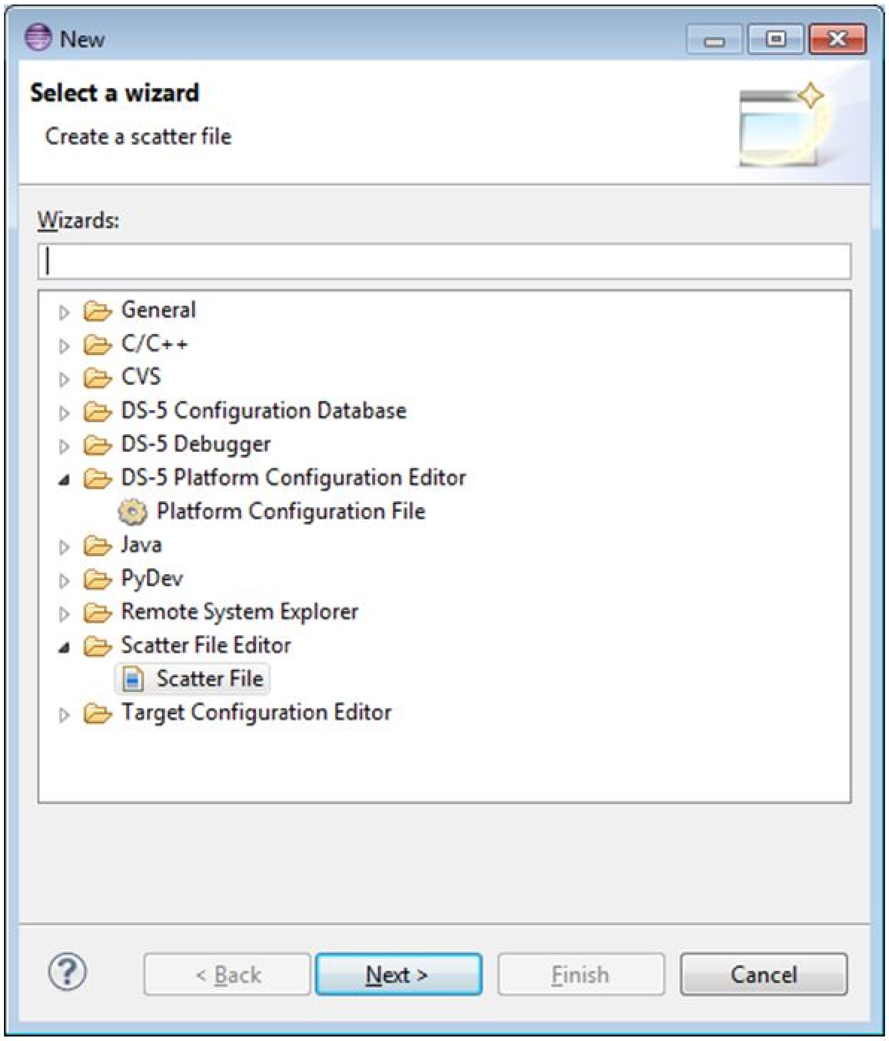
1. Go to **File ➤ New ➤ Other...**

Figure 18. Creating a Linker Script



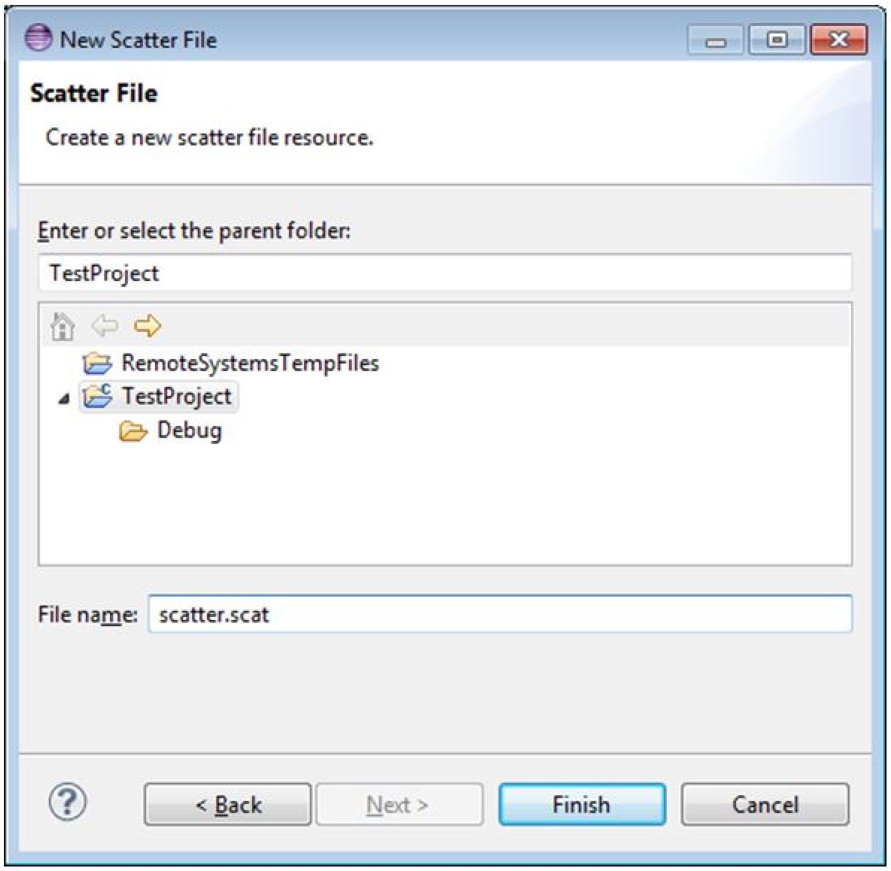
2. Select **Scatter File Editor ➤ Scatter File** and press **Next**.

Figure 19. Creating a Scatter File



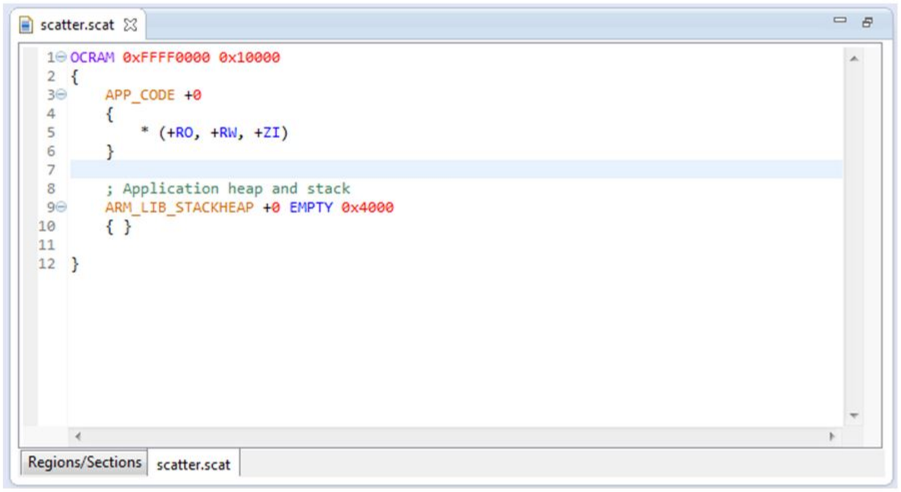
3. Select the location of the new file, type in the file name and press **Finish**.

Figure 20. Create a New Scatter File Resource



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

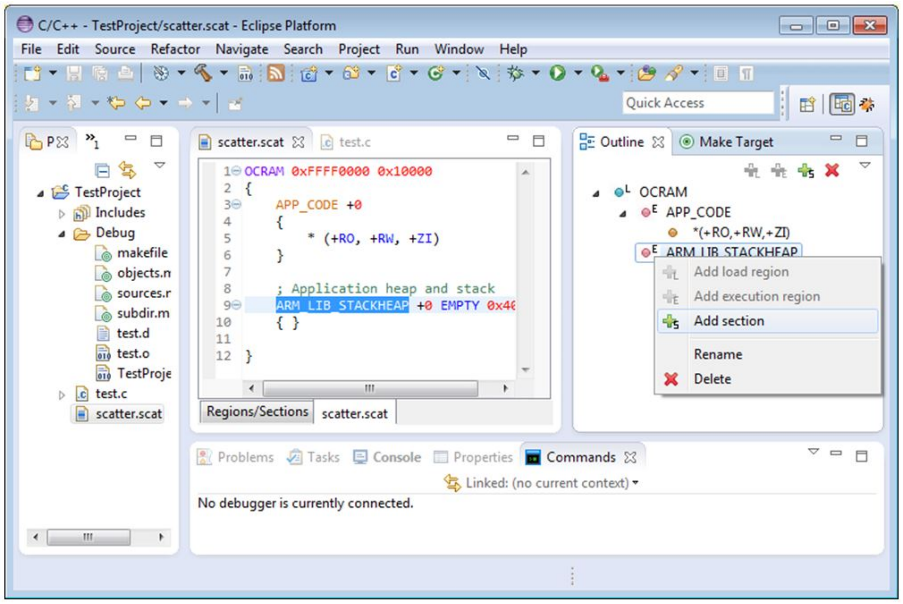
Figure 21. Linker Script Example



```
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 22. Editing "Scatter.scat" File Using Tools on the Outline View



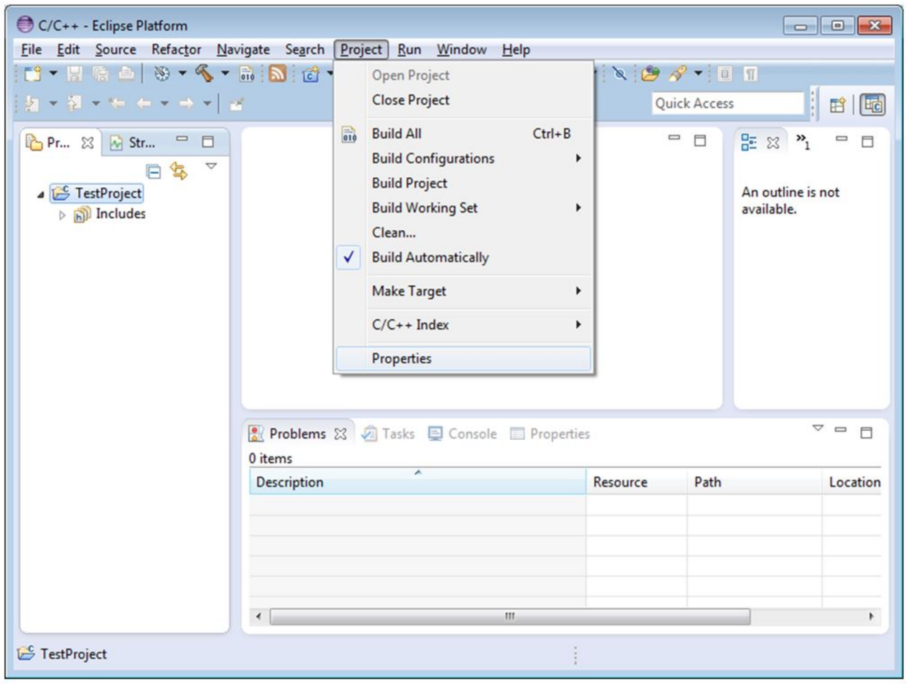
```
OCRAM 0xFFFF0000 0x10000
{
    APP_CODE +0
    {
        * (+RO, +RW, +ZI)
    }

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

No debugger is currently connected.

6.2.3.3. Build Settings

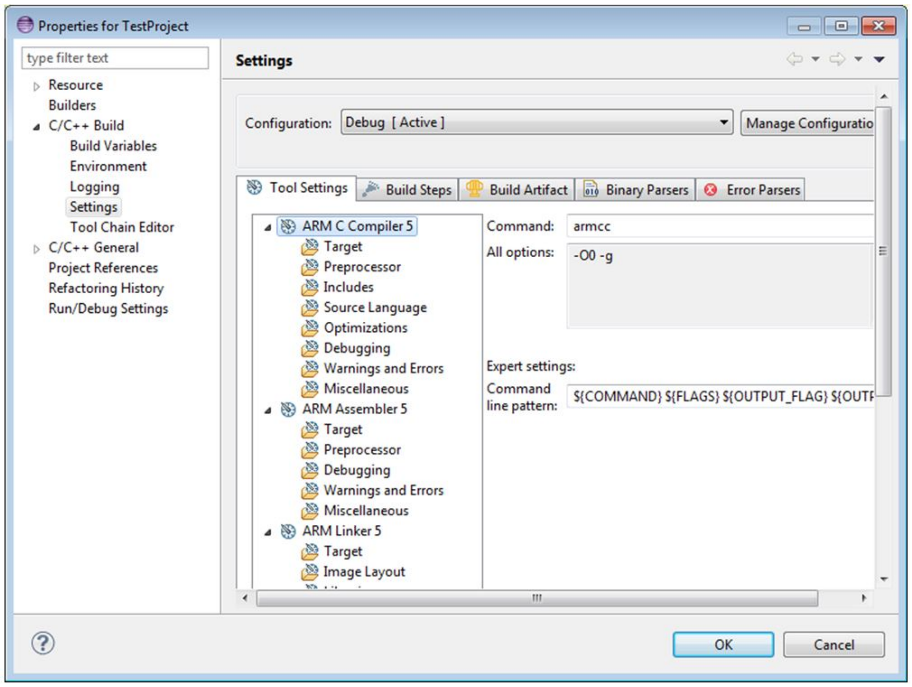
1. Once the project is created, the project properties can be accessed by going to **Project ➤ Properties**.

Figure 23. Project Properties



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

Figure 24. Project Settings



Configuration: Debug [Active]

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 DS-5 Intel SoC FPGA Edition 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 Intel SoC development board.

Related Information

Getting Started Guides on page 24

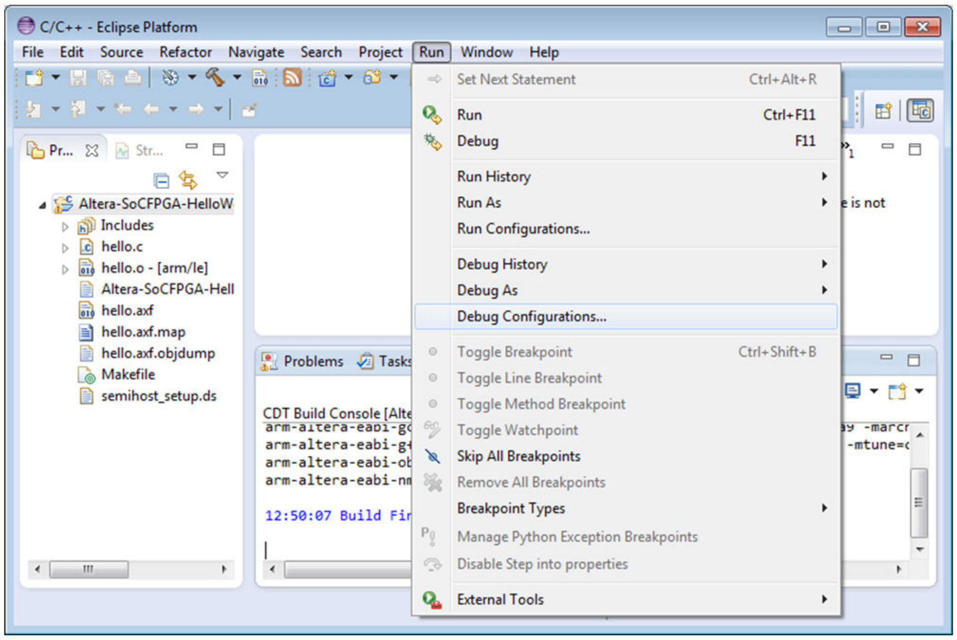
6.3. Debugging

The Arm Development Studio 5 Intel SoC FPGA Edition offers you a variety of debugging features.

6.3.1. 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.

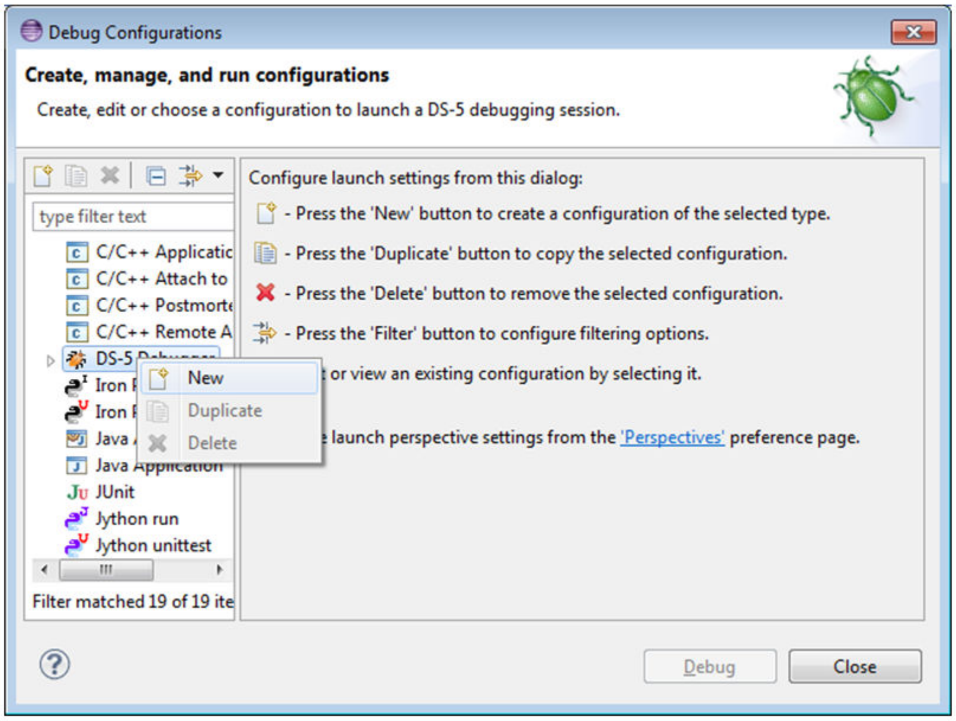
Figure 25. Accessing Debug Configurations



6.3.2. 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 26. Create New Debug Configuration


Debug Configurations

Create, manage, and run configurations

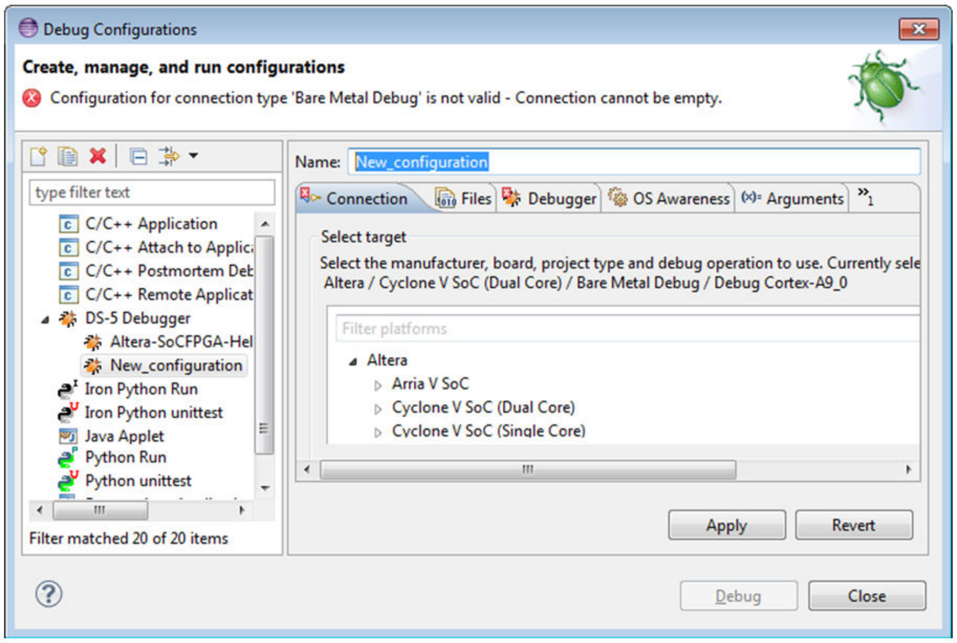
Create, edit or choose a configuration to launch a DS-5 debugging session.

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.

In the Arm Development Studio 5 Intel SoC FPGA Edition, you can assign a default name to the configuration, which you can edit.

Figure 27. Rename Debug Configuration


Debug Configurations

Create, manage, and run configurations

Configuration for connection type 'Bare Metal Debug' is not valid - Connection cannot be empty.

Name: New_configuration

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

Send Feedback

6.3.3. 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

- SoCEDSGettingStarted
  For examples on how to use the ARM DS-5 for Intel SoC FPGA debugging features, refer to the SoCEDSGettingStarted page on the Intel FPGA Wiki.
- Online ARM DS-5 Documentation
  For more information, refer to the DS-5 reference documentation located on the ARM website.

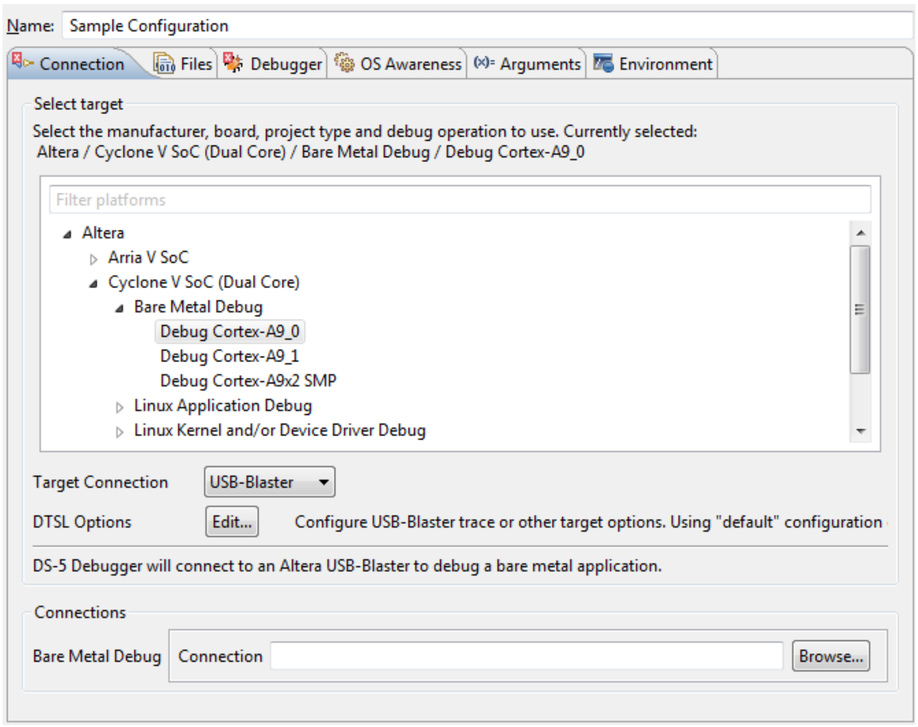
6.3.3.1. Connection Options

The **Connection** tab allows you to select the desired target. You must select the debugging target from the provided options, which cover BareMetal applications, Linux kernel and Linux application debugging for all Intel SoC FPGA devices.

The **Connections** panel looks differently, based on the selected Target. For BareMetal Debug, Linux Kernel, and Device Driver Debug target types, the following items appear:

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

Figure 28. Connection Options for BareMetal, Linux Kernel and Device Driver Debug



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

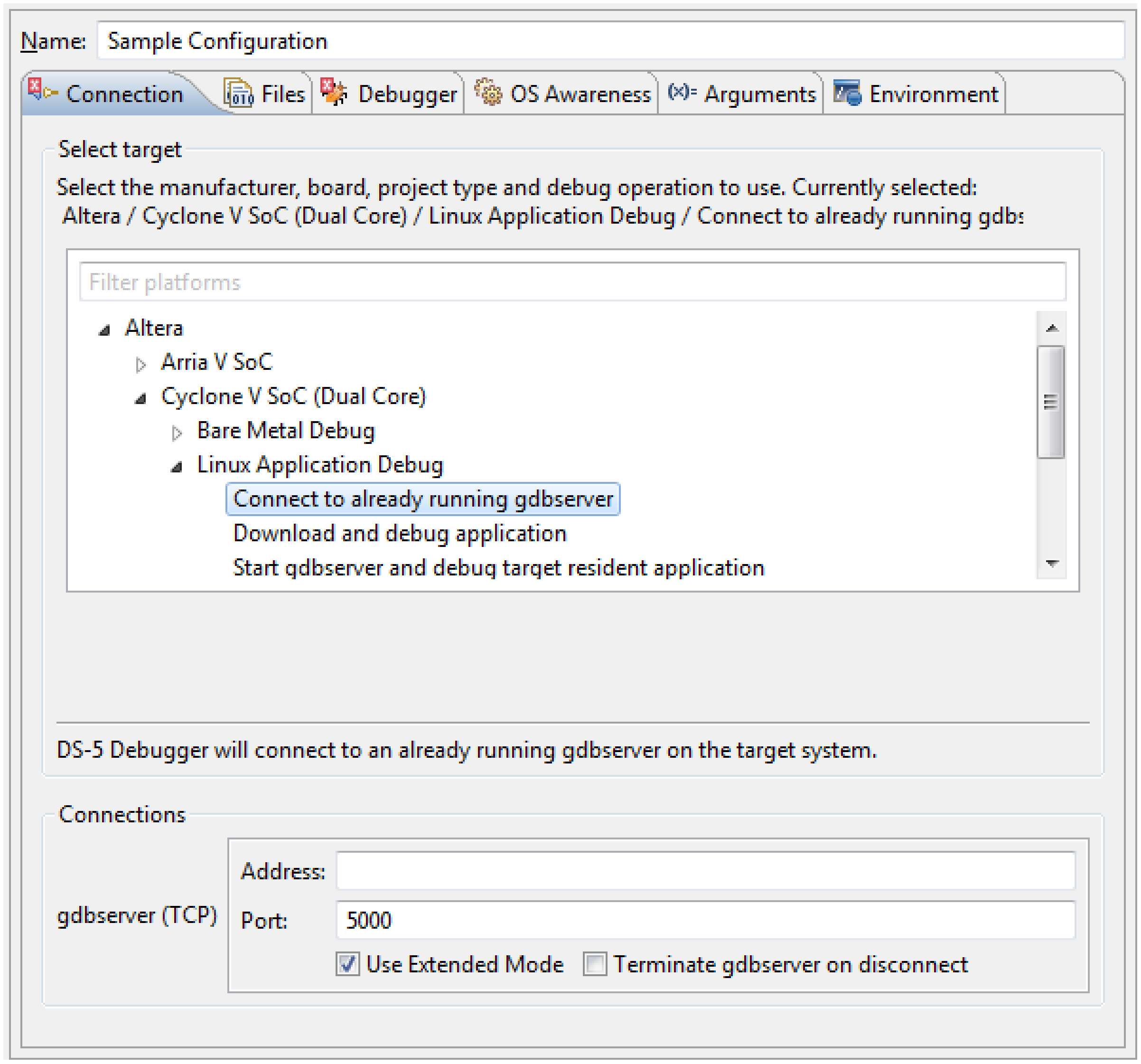
DS-5 Debugger will connect to an Altera USB-Blaster to debug a bare metal application.

Connections

Bare Metal Debug | Connection | Browse...

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

Figure 29. Linux Application Debugging – Connect to a Running GDB Server



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 / Connect to already running gdbs

Filter platforms

- 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

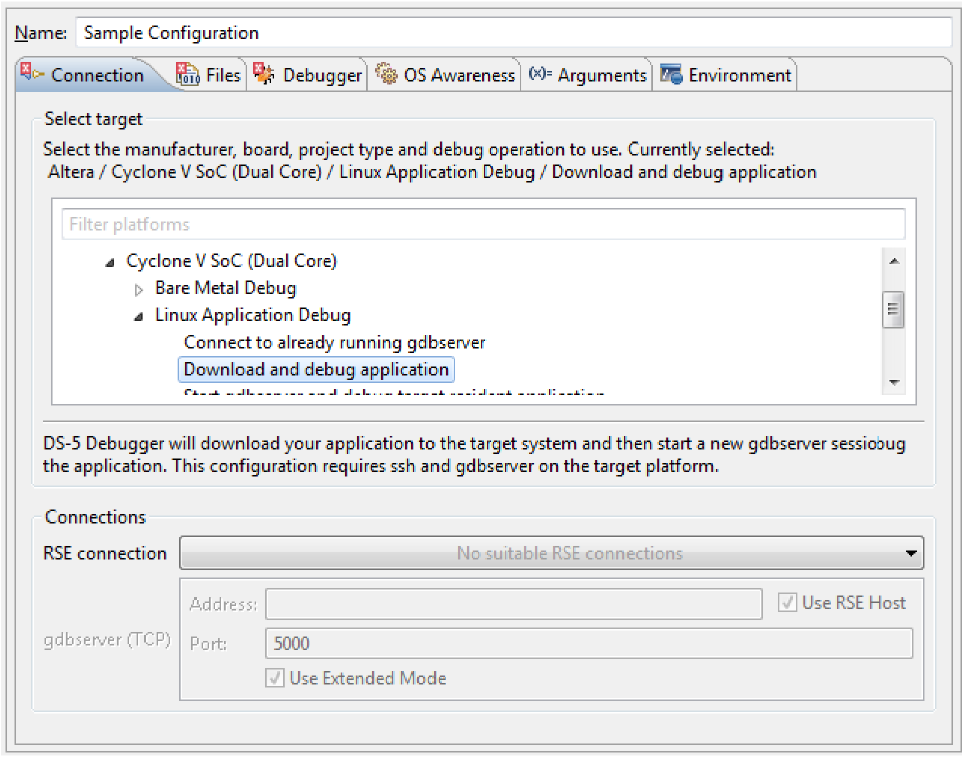
DS-5 Debugger will connect to an already running gdbserver on the target system.

Connections

gdbserver (TCP)

Address:
Port: 5000
[✓] Use Extended Mode [ ] Terminate gdbserver on disconnect

Figure 30. Linux Application Debugging – Download And Debug Application



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

- 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

Port: 5000

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

Related Information

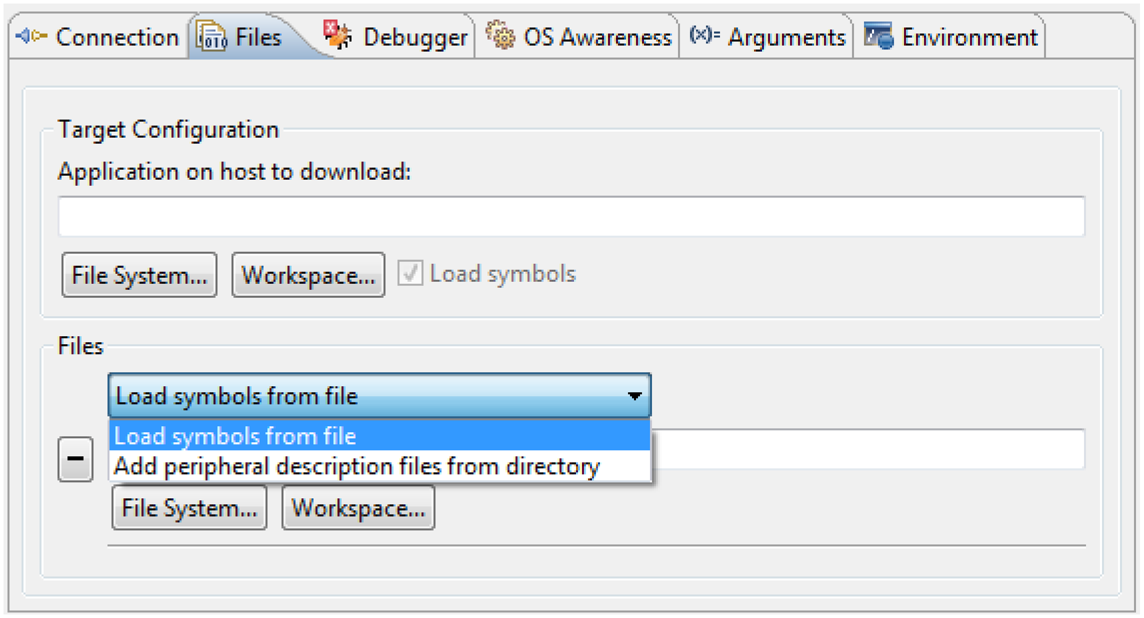
- DTSL Options on page 50
  For more information about the option on the Connections tab, refer to the DTSL Options section.
- Debugger Options on page 47
- Getting Started Guides on page 24

6.3.3.2. 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 uses 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 31. Files Settings

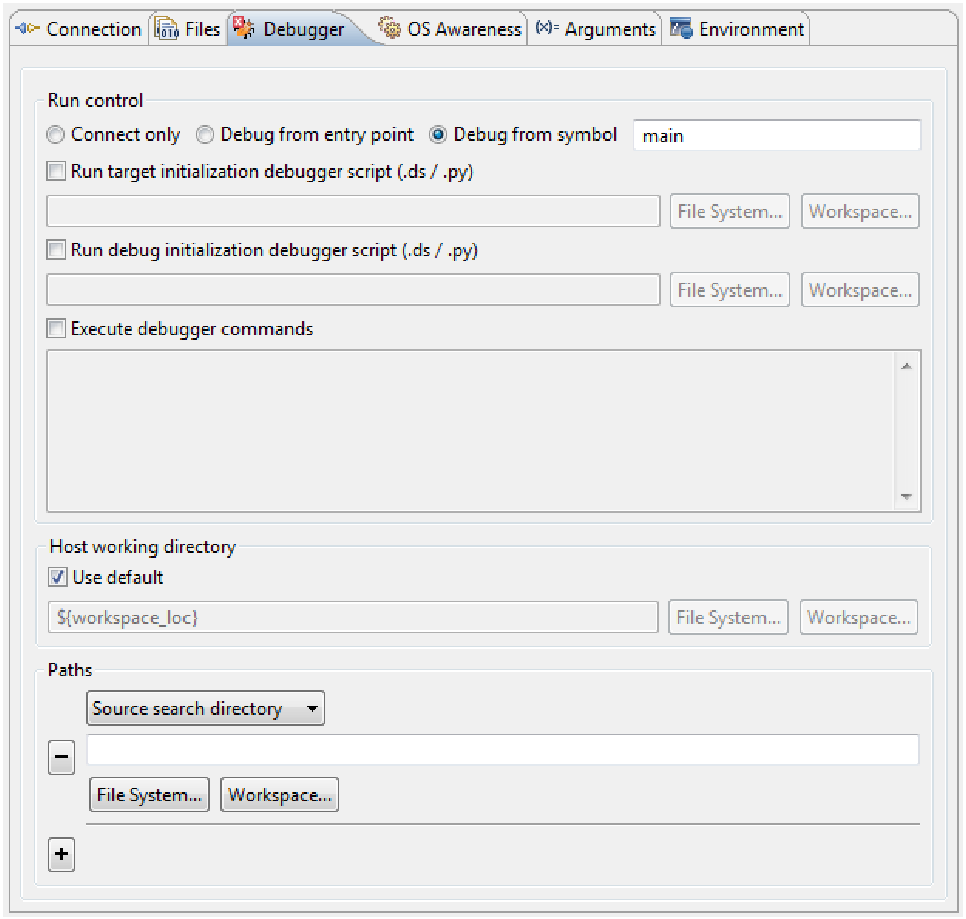


6.3.3.3. Debugger Options

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 you to enter multiple paths for the debugger to search for sources. Paths can be added with "+" button and removed with "-" button.

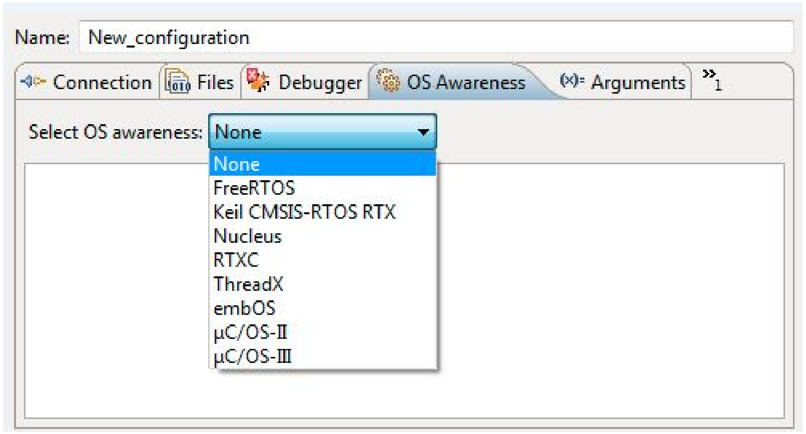
Figure 32. Debugger Settings



6.3.3.4. RTOS Awareness

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

Figure 33. RTOS Awareness Settings



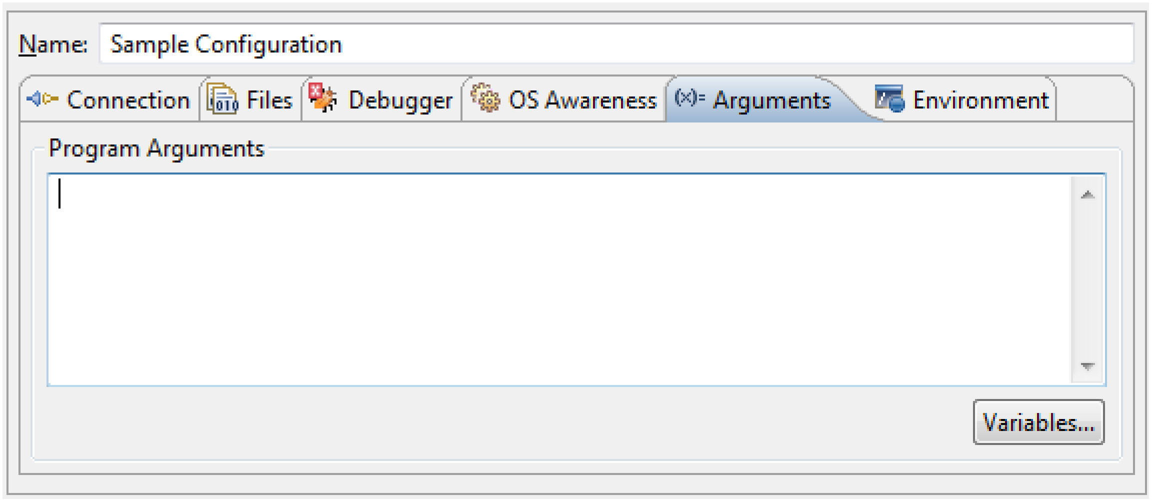
Related Information

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

6.3.3.5. Arguments

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

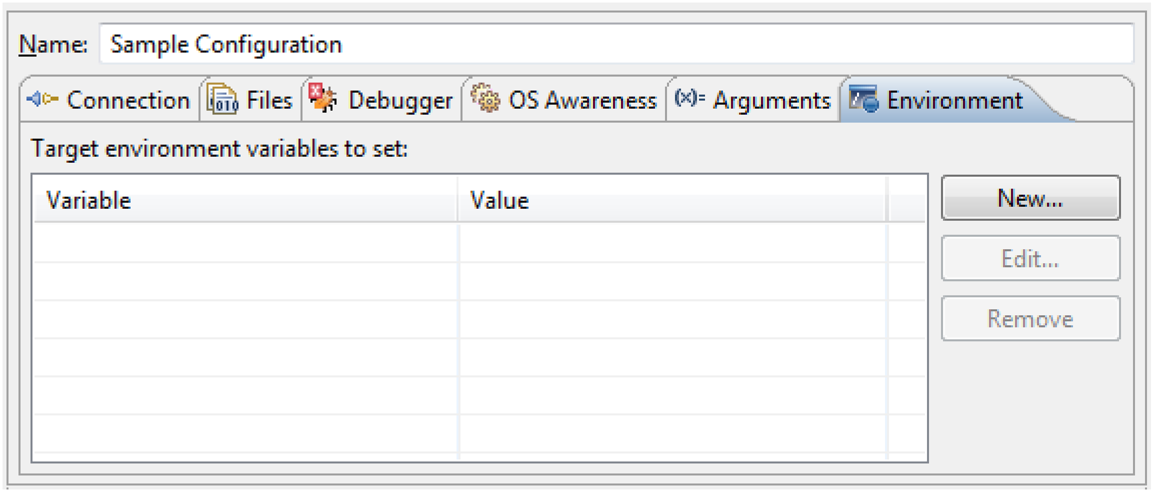
Figure 34. Arguments Settings



6.3.3.6. Environment

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

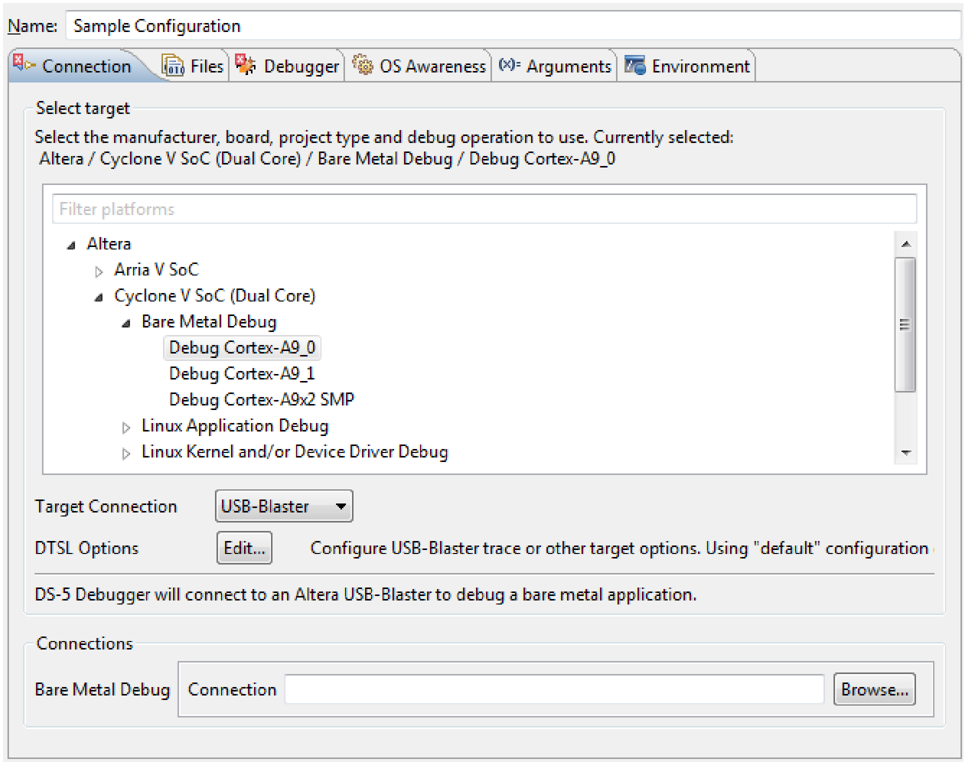
Figure 35. Environment Settings



6.3.4. 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 36. Debug Configurations - DTSL Options - Edit



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

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.

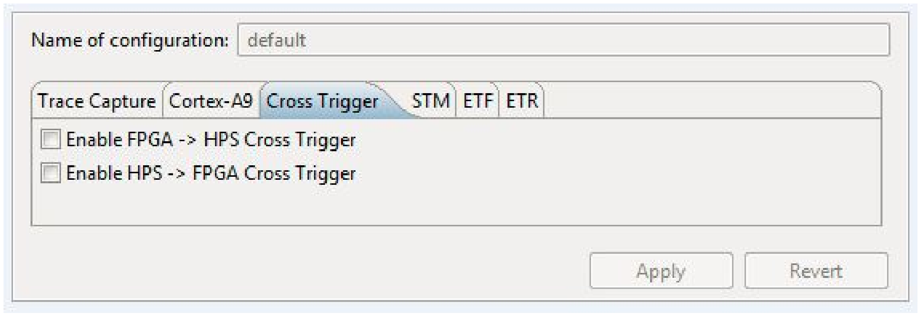
6.3.4.1. Cross Trigger Settings

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 37. 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

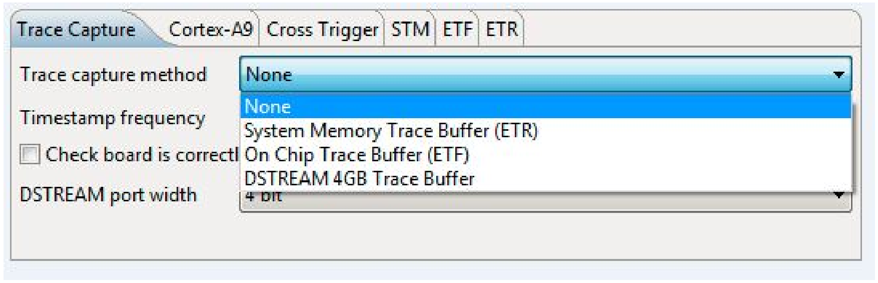
6.3.4.2. Trace Capture Settings

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 38. DTSL Configuration Editor - Trace Capture > Trace Capture Method

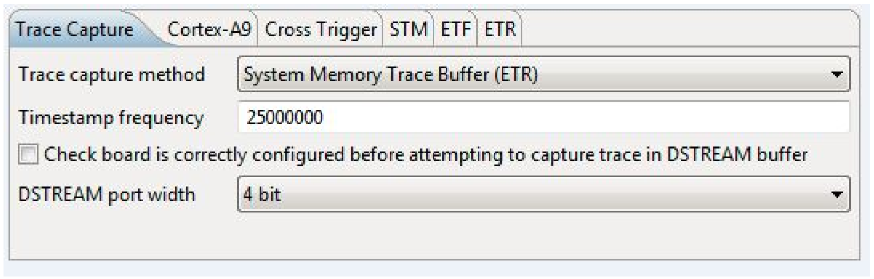


Tabs: Trace Capture | Cortex-A9 | Cross Trigger | STM | ETF | ETR

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

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

Figure 39. DTSL Configuration Editor - Trace Capture > Timestamp Frequency



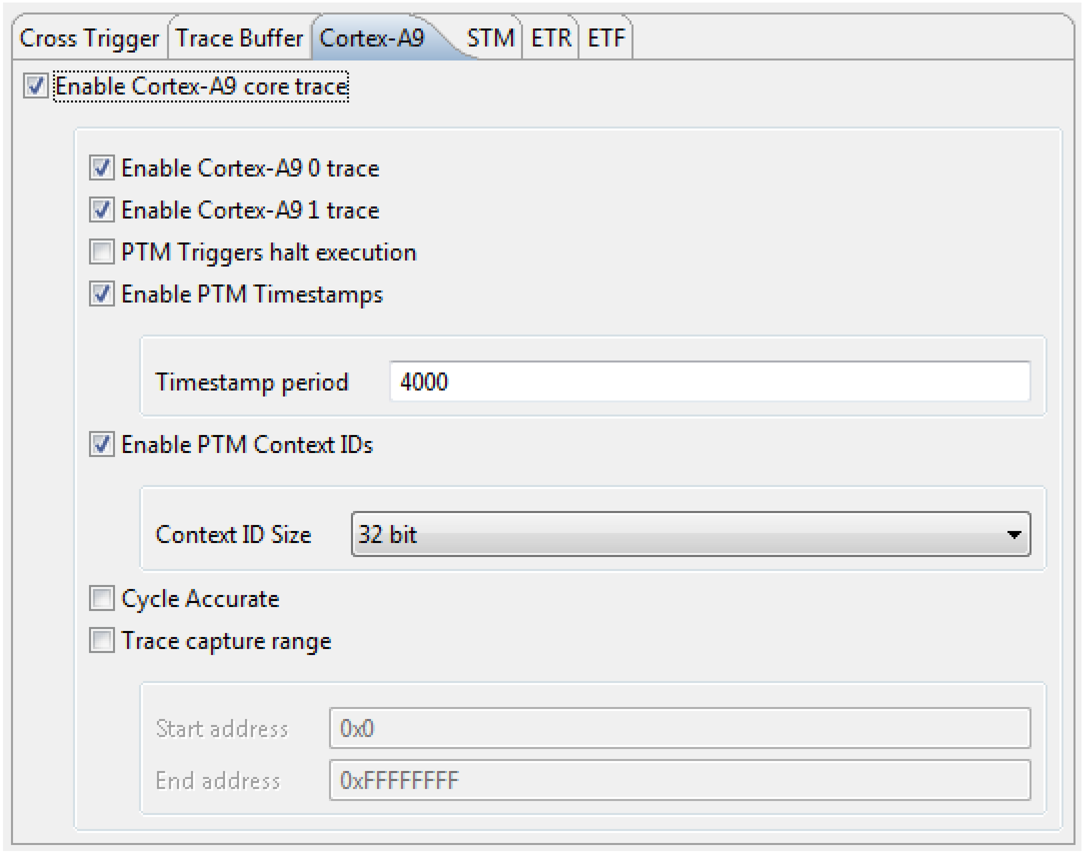
Tabs: Trace Capture | Cortex-A9 | Cross Trigger | STM | ETF | ETR

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

6.3.4.3. Cortex-A9 Settings

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

Figure 40. 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

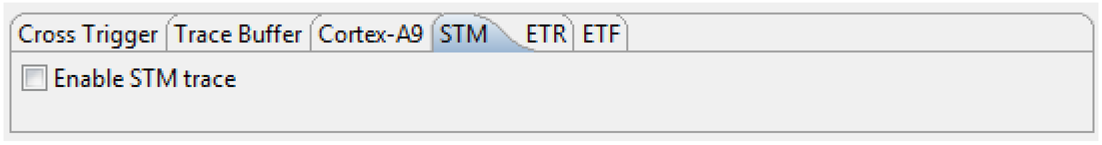
The following **Core Tracing Options** are available:

- **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)

6.3.4.4. STM Settings

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

Figure 41. DTSL Configuration Editor - STM



☐ Enable STM trace

Only one option is available:

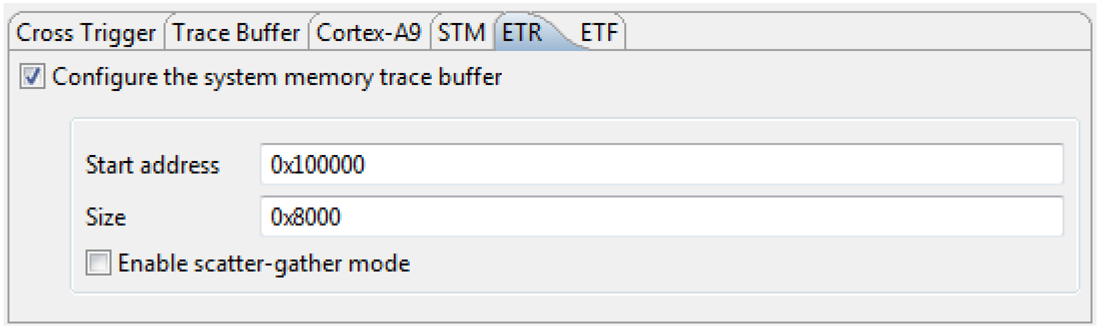
- **Enable STM Trace** – check to enable STM tracing.

6.3.4.5. 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 42. DTSL Configuration Editor - ETR



☑ Configure the system memory trace buffer

Start address 0x100000

Size 0x8000

☐ Enable scatter-gather mode

The following options are available:

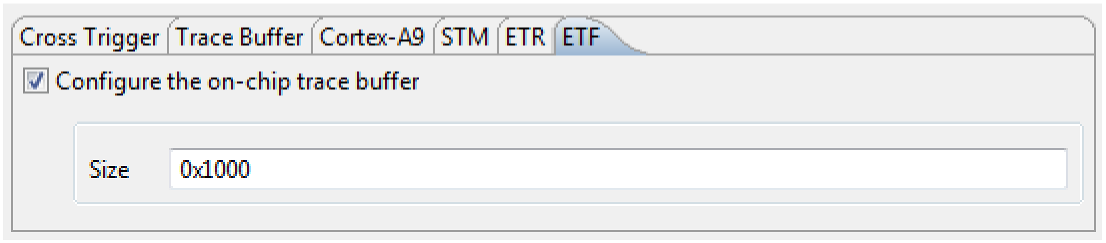
- **Configure the system memory trace buffer** – check this if the **ETR** is selected for trace destination on the **Trace Capture** tab.
- **Start Address, Size** – define the trace buffer location in system memory and its size.
- **Enable scatter-gather mode** – use when the OS cannot guarantee a 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.

6.3.4.6. 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 43. 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).

6.4. Arm Development Studio 5 Intel SoC FPGA EditionArm DS-5 Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Maintenance release
2018.06.18	Maintenance release
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Updated default size in ETF Settings
2015.08.06	Added Intel Arria 10 SoC support

7. Boot Tools User Guide

7.1. Introduction

The boot flow for all the Intel SoC devices includes a bootloader. The bootloader can have one or two stages. These stages are called FSBL (first stage bootloader) and SSBL (second stage bootloader).

The FSBL needs to fit into the on-chip RAM (OCRAM) and has limited functionality. It performs the tasks of initializing the HPS, bringing up the DDRAM and then loading and executing the next stage in the boot process. The next stage can be the SSBL, the end application, or an Operating System (OS).

The SSBL resides in DDRAM and therefore can have a larger size. Besides the ability to load and execute the next stage in the booting process, it typically offers a lot more functionality such as filesystem support, networking services, and command line interface.

For Cyclone V SoC, Arria V SoC, and Intel Stratix 10 SoC devices, the bootloader typically has two stages, although for simpler scenarios the first stage is sufficient. For Intel Arria 10 SoC devices, the bootloader typically consists of a single stage.

Figure 44. Cyclone V SoC and Arria V SoC Typical Boot Flow

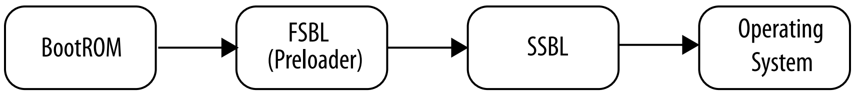


Figure 45. Intel Arria 10 SoC Typical Boot Flow

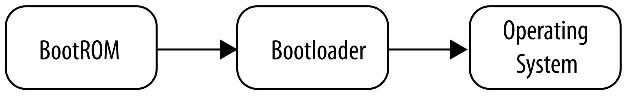
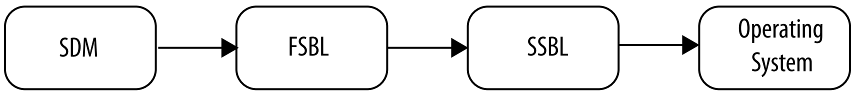


Figure 46. Intel Stratix 10 SoC Typical Boot Flow



The Cyclone V SoC and Arria V SoC FSBL is also known as Preloader.

On Cyclone V SoC, Arria V SoC and Intel Arria 10 SoC devices, the very first code run by HPS is the BootROM. For Intel Stratix 10 SoC devices, the very first code run by HPS is the FSBL which is loaded by Secure Device Manager (SDM).

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

- **BSP Generator** – enables you to create and manage the bootloader for Cyclone V SoC, Arria V SoC and Intel Arria 10 SoC devices.
- **Bootloader Image Tool (** `mkpimage` **)** — enables you to add the BootROM-required header on top of the bootloader for Cyclone V SoC, Arria V SoC and Intel Arria 10 SoC devices.
- **U-Boot Image Tool (mkimage)**— enables you to add the bootloader-required header on top of the files loaded by bootloader.

This chapter also presents instructions on how to build the bootloader for all the Intel SoC FPGA devices.

7.2. BSP Generator

The BSP Generator provides an easy, safe, and reliable way to customize the Bootloader for the Intel SoC devices: Cyclone V SoC, Arria V SoC and Intel Arria 10 SoC.

Note: There is no equivalent tool for the Intel Stratix 10 SoC devices. For Intel Stratix 10 SoC devices, the settings are available in the Intel Quartus Prime project, and passed along to the Bootloader by the SDM.

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.

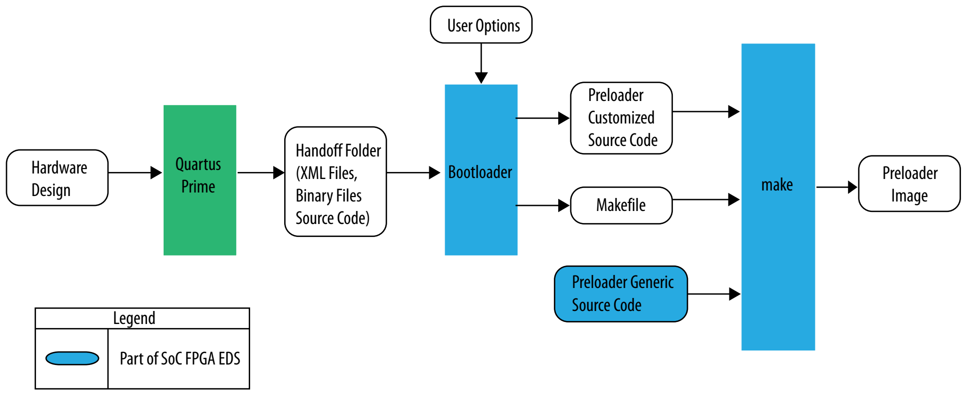
7.2.1. BSP Generation Flow

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

7.2.1.1. Cyclone V SoC and Arria V SoC Flow

For Cyclone V SoC and Arria V SoC, 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 47. Arria V SoC/ Cyclone V SoC BSP Generator Flow


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

The hardware handoff information contains various settings that you entered when creating the hardware design in Platform Designer (Standard) and Intel Quartus Prime Standard Edition. 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 gating settings
- DDR settings:
  - Technology
  - Width
  - Speed

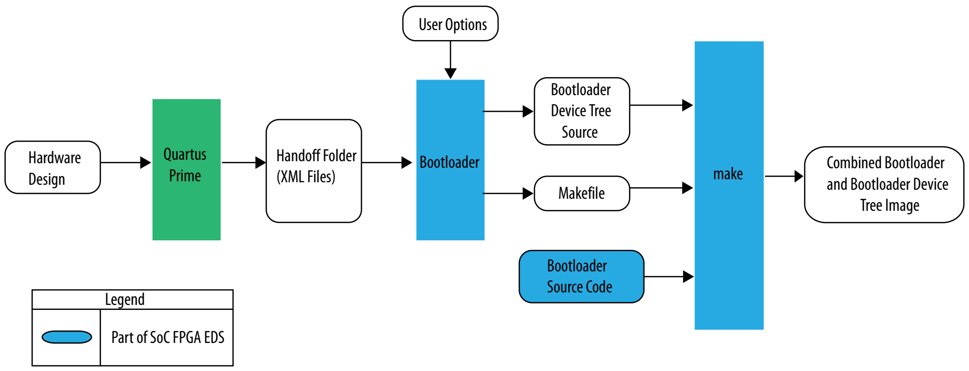
The handoff settings are output from the Intel Quartus Prime Standard Edition compilation and are located in the **<quartus project directory>/ hps_isw_handoff/<hps entity name>** directory (where <**hps entity name** > is the HPS component name in Platform Designer (Standard)).

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.

7.2.1.2. Intel Arria 10 SoC Flow

For Intel Arria 10 SoC, 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 48. Intel Arria 10 SoC BSP Generator Flow



The hardware handoff information contains various settings that you entered when creating the hardware design in Platform Designer (Standard) and Intel Quartus Prime Standard Edition. 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 gating settings

The handoff settings are output from the Intel Quartus Prime Standard Edition 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.

Send Feedback

However, you 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.

7.2.2. 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 SoC/Arria V SoC HPS)
  - — U-Boot Bootloader or UEFI Bootloader (Intel Arria 10 SoC 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 SoC/ Cyclone V SoC Preloader and `uboot_bsp` or `uefi_bsp` for the Intel Arria 10 SoC 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** dialog 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.

7.2.2.1. 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.

For more information about the list of available settings, refer to the "BSP Settings" section.

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 63

7.2.3. 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`.

7.2.3.1. 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 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 4. 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 SoC/Arria V SoC Preloader • "uboot" and "uefi" for Intel Arria 10 SoC Bootloader
`--settings <filename>`	Yes	This option specifies the path to a BSP settings file. The file is created with default settings.Intel 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.Intel 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.

7.2.3.2. bsp-update-settings

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

Example 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 5. 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.

7.2.3.3. bsp-query-settings

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

Example 3. Querying a BSP Settings File

The following command retrieves all the settings from `settings.bsp` and displays the setting names and values:

```
bsp-query-settings --settings settings.bsp --get-all --show-names
```

Table 6. User Parameters: bsp-query-settings

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

7.2.3.4. 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 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 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. 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.

7.2.4. 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 SoC/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 Intel SoC FPGA Edition 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 Intel Arria 10 SoC Bootloader BSPs, for both U-Boot and UEFI, 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 you settings

7.2.5. 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 SoC and Arria V SoC SSBL (Preloader) and Intel Arria 10 SoC SSBL (Bootloader).

7.2.5.1. Cyclone V SoC and Arria V SoC BSP Settings

Table 8. Cyclone V SoC and Arria V SoC BSP Settings

BSP Setting	Type	Default Value	Description
`spl.PRELOADER_TGZ`	String	*<SoC EDS installation directory>*/ host_tools/ altera/	This setting specifies the path to archive file containing the preloader source files.

continued...

BSP Setting	Type	Default Value	Description
		preloader/ uboot- socfpga.tar.gz"	
`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`	DecimalNumber	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.
`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 BootROM that the preloader has run successfully.
`spl.boot.BOOTROM_HANDSHAKE_CFGIO`	Boolean	True	This setting enables handshake with BootROM 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 BootROM reconfigures 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.

continued...

Send Feedback

BSP Setting	Type	Default Value	Description
			`spl.boot.WARMRST_SKIP_CFGIO` is only applicable if the boot ROM has skipped IOCSR and pin multiplexing configuration.
`spl.boot.SDRAM_INITIALIZATION`	Boolean	False	Initialize the SDRAM to initialize the ECC bits.
`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.
`spl.reset_assert.DMA`	Boolean	False	When enabled, this setting forces 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

continued...

BSP Setting	Type	Default Value	Description
			outstanding AXI transactions before asserting a warm reset of the L3 interconnect or a debug reset of the ETR.
`spl.warm_reset_handshake.SDRAM`	Boolean	False	This option allows the SDRAM contents to be preserved across 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.
`spl.debug.DEBUG_MEMORY_SIZE`	Hexadecimal	0x200	This setting specifies the maximum size used for storing preloader debug information.

continued...

BSP Setting	Type	Default Value	Description
`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 fails 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 you software. *Note:* Enabling this feature with CSEL != 00 does not have any effect, as the impacted code checks for this.

7.2.5.2. Intel Arria 10 SoC BSP Settings

The Intel Arria 10 SoC 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 9. Intel Arria 10 SoC BSP Firewall Setting Prefixes

Settings Group	Setting Group Prefix
Main Group	N/A
MPU Firewall	For U-Boot: `altera_arria10_soc_noc_arria10_uboot_driver.I_NOC.mpu_m0.noc_fw_ddr_mpu_fpga2sdram_ddr_scr.` For UEFI: `altera_arria10_soc_noc_arria10_uefi_driver.I_NOC.mpu_m0.noc_fw_ddr_mpu_fpga2sdram_ddr_scr.`
L3 Firewall Group	For U-Boot: `altera_arria10_soc_noc_arria10_uboot_driver.I_NOC.mpu_m0.noc_fw_ddr_l3_ddr_scr.` For UEFI: `altera_arria10_soc_noc_arria10_uefi_driver.I_NOC.mpu_m0.noc_fw_ddr_l3_ddr_scr.`
F2S Firewall Group	For U-Boot: `altera_arria10_soc_noc_arria10_uboot_driver.I_NOC.mpu_m0.noc_fw_ddr_mpu_fpga2sdram_ddr_scr.` For UEFI: `altera_arria10_soc_noc_arria10_uefi_driver.I_NOC.mpu_m0.noc_fw_ddr_mpu_fpga2sdram_ddr_scr.`

7.2.5.2.1. Intel Arria 10 SoC Main BSP Settings Group

Table 10. Intel Arria 10 SoC Main BSP Settings Group for U-Boot

BSP Setting	Type	Default Value	Description
`uboot.boot_device`	String	SD/MMC	Select the source for the boot image. Possible values are SD/MMC, QSPI, and NAND.
`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.
`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.

Table 11. Intel Arria 10 SoC Main BSP Settings Group for UEFI

BSP Setting	Type	Default Value	Description
`uefi.boot_device`	String	SD/MMC	Select the source for the boot image. Possible values are SD/MMC, QSPI, and NAND.
`uefi.model`	String	SOCFPGA Arria10 Dev Kit	Name of the board to be displayed when bootloader starts.

continued...

BSP Setting	Type	Default Value	Description
`uefi.external_fpga_config`	Boolean	False	Configure UEFI 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.
`uefi.rbf_filename`	String	socfpga.rbf	Full FPGA `.rbf` filename. This setting is ignored when the `uefi.external_fpga_config` setting is enabled.
`uefi.rbf_offset`	Hexadecimal	`0x720000`	RBF offset address in QSPI Flash.
`uefi.disable_uefi_build`	Boolean	False	Can be used to disable building UEFI, and just generate the Bootloader Device Tree file.

For more information about authentication and encryption, refer to the *Intel Arria 10 SoC Secure Boot User Guide*.

Related Information

Arria 10 SoC Secure Boot User Guide
For more information about authentication and encryption

7.2.5.2.2. Intel Arria 10 SoC Bootloader MPU Firewall BSP Settings Group

Table 12. Intel Arria 10 SoC Bootloader MPU Firewall BSP Settings Group

BSP Setting[1]	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`			

[1] Use the Intel Arria 10 SoC BSP Firewall Setting Prefixes. Table 9 on page 67

7.2.5.2.3. Intel Arria 10 SoC Bootloader L3 Firewall BSP Settings Group

Table 13. Intel Arria 10 SoC Bootloader L3 Firewall BSP Settings Group

BSP Setting[2]	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`			

[2] Use the Intel Arria 10 SoC BSP Firewall Setting Prefixes. Table 9 on page 67

7.2.5.2.4. Intel Arria 10 SoC Bootloader FPGA-to-SDRAM Firewall BSP Settings Group

Table 14. Intel Arria 10 SoC Bootloader FPGA-to-SDRAM Firewall BSP Settings Group

BSP Setting[3]	Type	Default Value	Description
`enable.fpga2sdram0region0`	Boolean	True	Enable FPGA-to-SDRAM 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	FPGA-to-SDRAM 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`			
`fpga2sdram0region0addr.limit`	Hexadecimal	0xFFFF	FPGA-to-SDRAM Firewall region limits
`fpga2sdram0region1addr.limit`			
`fpga2sdram0region2addr.limit`			
`fpga2sdram0region3addr.limit`			
`fpga2sdram1region0addr.limit`			
`fpga2sdram1region1addr.limit`			

continued...

[3] Use the Intel Arria 10 SoC BSP Firewall Setting Prefixes. Table 9 on page 67

BSP Setting[3]	Type	Default Value	Description
fpga2sdram1region2addr.limit			
fpga2sdram1region3addr.limit			
fpga2sdram2region0addr.limit			
fpga2sdram2region1addr.limit			
fpga2sdram2region2addr.limit			
fpga2sdram2region3addr.limit			

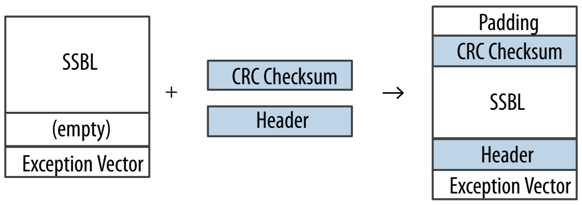
7.3. Bootloader Image Tool (`mkpimage`)

The Bootloader Image Tool (`mkpimage`) creates an Intel BootROM-compatible image of the Arria V SoC and Cyclone V SoC Preloader or Intel Arria 10 SoC 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 49. Basic Operation of the `mkpimage` Tool



7.3.1. 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 bootloader program image and its exception vector.

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

(3) Use the Intel Arria 10 SoC BSP Firewall Setting Prefixes. Table 9 on page 67

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

As illustrated, the binary bootloader image is an input to the `mkpimage` tool. The compiler leaves an empty space between the bootloader 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.

7.3.2. Header File Format

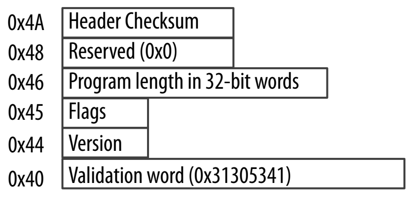
The `mkpimage` header file format has two versions:

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

For Version 0, used for Cyclone V SoC and Arria V SoC 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)

Figure 50. Header Format Version 0

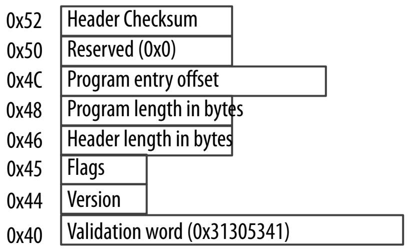


For Version 1, used for Intel Arria 10 SoC Bootloader, the header includes the following:

- 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 51. 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)

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.

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

Type `mkpimage` to launch the tool. The `--help` option provides a tool description and tool usage and option information.

```
$ mkpimage --help
mkpimage version 19.1

Description: This tool creates an Intel 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
SoC =
                                0, Intel
          Arria 10 SoC = 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
```

7.3.4. Output Image Layout

7.3.4.1. 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 BootROM 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.

7.3.4.2. Size

For Cyclone V SoC and Arria V SoC, a single Preloader has a maximum 60 KB image size. You can store up to four preloader images in flash. If the BootROM 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 Intel Arria 10 SoC, a single Bootloader has a maximum 200 KB image size. You can store up to four Bootloader images in flash. If the BootROM 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.

7.3.4.3. Address Alignment

For Cyclone V SoC and Arria V SoC, 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 Intel Arria 10 SoC, 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 NAND erase block size equal or less to 64 KB (for Cyclone V SoC/Arria V SoC) or 256 KB (for Intel Arria 10 SoC).

Table 15. Typical Arria V SoC/Cyclone V SoC Preloader Image Layout

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

Table 16. Typical Intel Arria 10 SoC Bootloader Image Layout

Offset	Image
0xC0000	Bootloader Image 3
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.

7.3.4.3.1. 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 SoC and Arria V SoC, 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.

7.3.4.3.2. 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.

7.3.4.3.3. 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.

7.3.4.4. 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

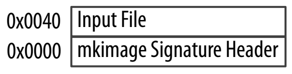
- Arria V Hard Processor System Technical Reference Manual
  For more information, please refer to the Booting and Configuration chapter.
- Cyclone V Hard Processor System Technical Reference Manual
  For more information, please refer to the Booting and Configuration chapter.
- Arria 10 Hard Processor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

7.4. U-Boot Image Tool (mkimage)

Both the FSBL and SSBL 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.

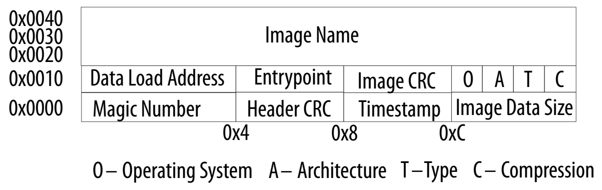
Figure 52. mkimage Header



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 53. mkimage Header Layout



7.4.1. 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
```

7.4.2. Usage Examples

Example 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. Creating a BareMetal Application Image

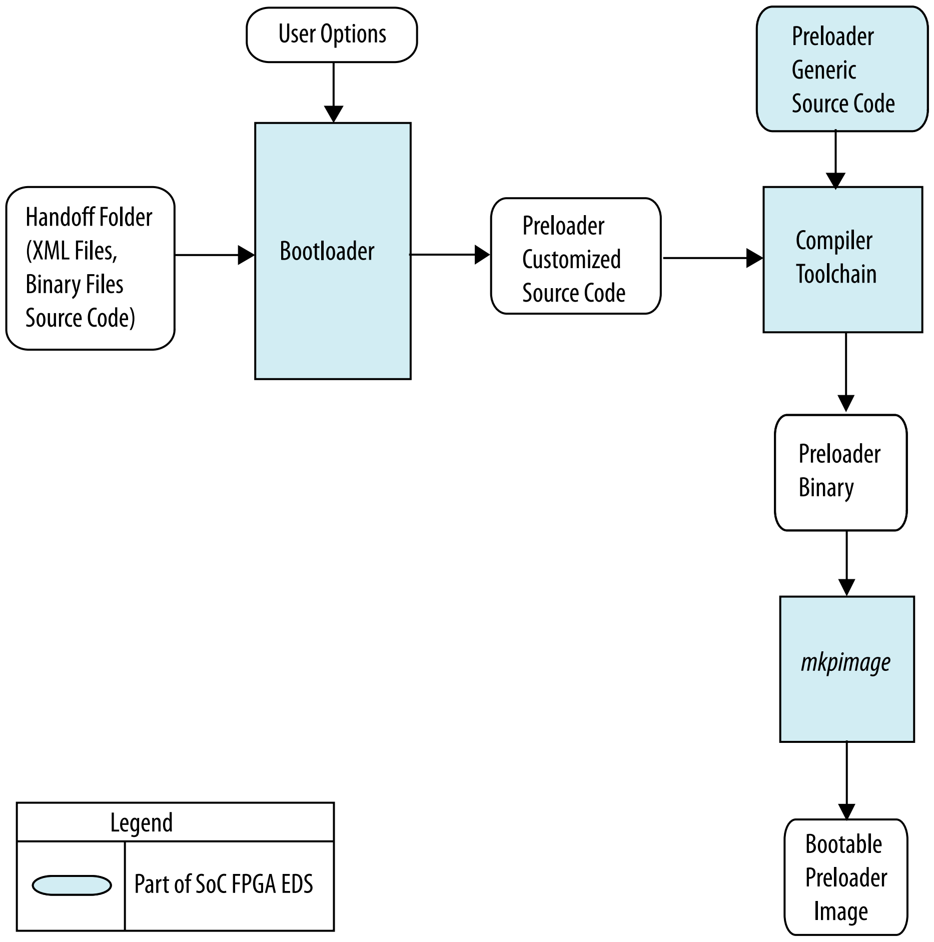
```
mkimage -A arm -O u-boot -T standalone -C none -a 0x02100000 -e 0 -n "baremetal 
image" -d hello_world.bin hello_world.img
```

7.5. Building the Bootloader

7.5.1. Building the Cyclone V SoC and Arria V SoC Preloader

The following figure presents the Cyclone V SoC and Arria V SoC 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 54. Cyclone V SoC and Arria V SoC Preloader Build Flow



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 you-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

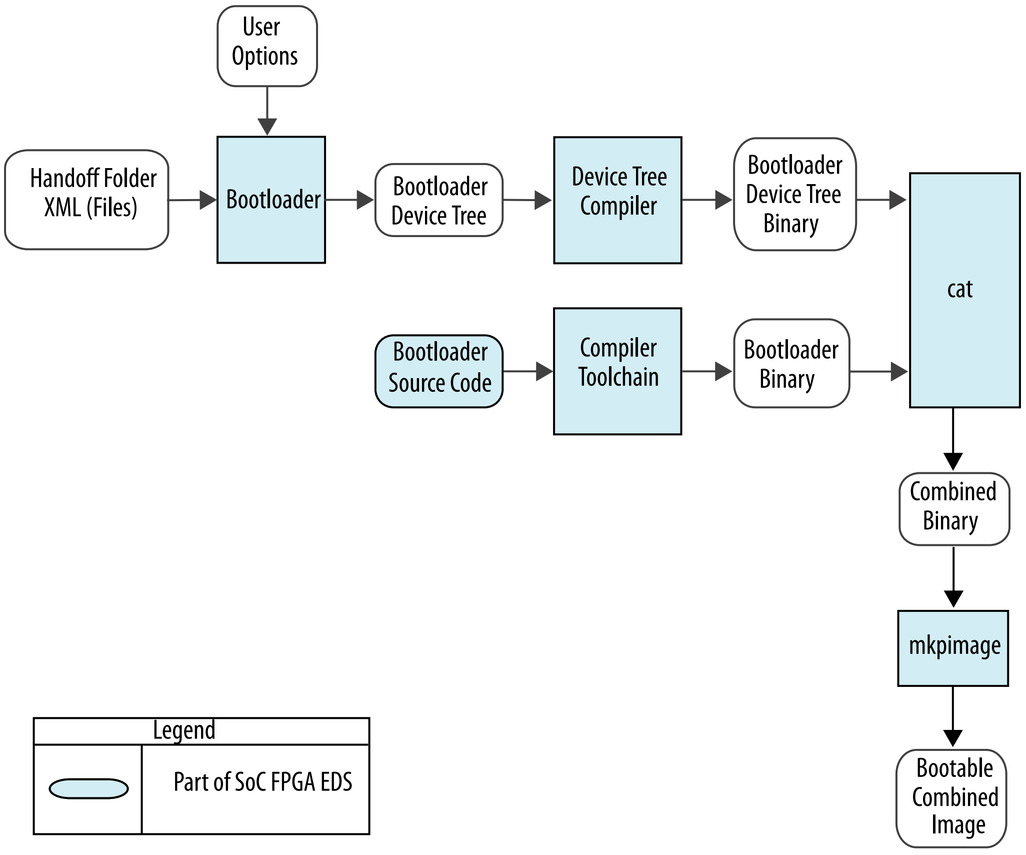
The makefile contains the following targets:

- `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

7.5.2. Building the Intel Arria 10 SoC Bootloader

The following figure presents the Intel Arria 10 SoC 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 55. Intel Arria 10 SoC Bootloader Build Flow



Note: For this release of the tools, the U-Boot compilation is only supported on Linux host machines. On Windows machines, the U-Boot compilation is not supported, and the bootloader must be generated with compilation disabled (with `--set uboot.disable_uboot_build` set to true or from `bsp-editor` interface). In this case, a pre-built version of U-Boot is used. The only limitation is that the U-Boot sources cannot be modified. Since the U-Boot configuration is contained in the bootloader device tree, this should not be a problem in most situations. However, if U-Boot source customization is required, a Linux host machine needs to be used.

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.

7.5.2.1. U-Boot Build Flow

The U-Boot makefile performs the following tasks:

- Copies the bootloader source code into `<bsp_directory>/uboot-socfpga`.
- Configures the bootloader to target the Intel Arria 10 SoC.
- Compiles the source files in `<bsp_directory>/uboot-socfpga` and creates the Bootloader binary `<bsp_directory>/uboot-socfpga/u-boot.bin`.
- Compiles the bootloader device tree source file by using the Device Tree Compiler (**dtc**) into `<bsp_directory>/devicetree.dtb`.
- Concatenates the bootloader device tree binary and the bootloader binary files into a combined binary file `<bsp_directory>/uboot_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 U-Boot 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

7.5.2.2. UEFI Build Flow

The makefile performs the following tasks:

- Copies the UEFI source code into `<bsp_directory>/uefi-socfpga`.
- Configures the UEFI to target the Intel Arria 10 SoC.
- Compiles the bootloader device tree source file by using the Device Tree Compiler (dtc) into `<bsp_directory>/devicetree.dtb`.
- Compiles the UEFI bootloader, concatenates the device tree and creates a bootable image in `<bsp_directory>/uefi-socfpga/Build/PEI.ROM`. This is the equivalent of `<bsp_directory>/uboot_w_dtb-mkpimage.bin`.

For more information about UEFI bootloader, refer to the "UEFI Bootloader" page on the Intel FPGA Wiki.

Related Information

Altera Wiki
For more information about UEFI bootloader, refer to the "UEFI Bootloader" page.

7.5.3. Building the Intel Stratix 10 SoC BootloaderBuilding the S10 Bootloader

On Intel Stratix 10 SoC devices, the SDM loads the FSBL from the configuration bitstream. The configuration bitstream contains the hardware handoff data, which is made available to the FSBL. Because of this, there is no need for a Bootloader Generator tool to pass additional information to the Bootloader building process.

The SoC EDS includes the source code for U-Boot, Arm Trusted Firmware (ATF) and UEFI bootloaders in the folder `<SoC EDS installation folder>/host_tools/altera/bootloaders/stratix10`. For each bootloader, the source code is provided as an archive, but also as a shell script which fetches the same source code version from the altera-opensource repository on github.

The SoC EDS includes the pre-built U-Boot FSBL and SSBL binaries in the folder `<SoC EDS installation folder>/host_tools/altera/bootloaders/stratix10/u-boot/prebuilt`. When no special customizations are needed, the prebuilt binaries can be used directly.

The U-Boot FSBL and SSBL can be rebuilt from source code on a Linux host by running the following steps:

1. Start an embedded command shell:

   ```
   ~/intelFPGA_pro/19.1/embedded/embedded_command_shell.sh
   ```

2. Download and setup the required toolchain:

   ```
   wget https://releases.linaro.org/components/toolchain/binaries/7.2-2017.11\
   /aarch64-linux-gnu/gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar.xz
   tar xf gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar.xz
   export PATH=`pwd`/gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu/bin/
   :$PATHexport CROSS_COMPILE=aarch64-linux-gnu-
   ```

3. Extract the U-Boot source code:

   ```
   tar xf $SOCEDS_DEST_ROOT/host_tools/altera/bootloaders/stratix10/
   u-boot/uboot-socfpga.tar.gz
   cd uboot-socfpga
   ```

4. Configure U-Boot with the default Intel Stratix 10 SoC configuration, intended for the Intel Stratix 10 SoC Development Kit:

   ```
   make socfpga_stratix10_defconfig
   ```

5. Build U-Boot:

   ```
   make
   ```

   The following files are built:

   - FSBL binary: `uboot-socfpga/spl/u-boot-spl-dtb.bin`
   - SSBL binary: `uboot-socfpga/u-boot-dtb.img`

6. Create the hex version of FSBL, to be added to the configuration bitstream:

```
aarch64-linux-gnu-objcopy -I binary -O ihex --change-addresses 0xffe00000 \
spl/u-boot-spl-dtb.bin spl/u-boot-spl.hex
```

For more details about ATF and UEFI, refer to *Intel Stratix 10 SoC UEFI Boot Loader User Guide*.

For more details about the Intel Stratix 10 SoC bootloaders, refer to the *Intel Stratix 10 SoC FPGA Boot User Guide*.

Related Information

- Intel® Stratix® 10 SoC UEFI Boot Loader User Guide
- Intel® Stratix® 10 SoC FPGA Boot User Guide

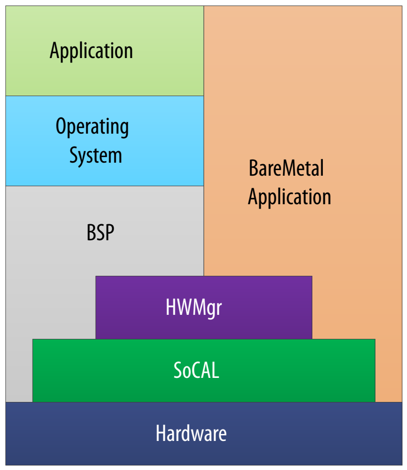
7.6. Boot Tools User Guide Revision History

Document Version	Changes
2019.05.16	• *Tool Usage* section: Updated the `mkpimage` version to 19.1. • *Building the Intel Stratix 10 SoC Bootloader* section: In step 1, updated the path for the 19.1 release.
2018.09.24	Updated the path names, in the "Building the Intel Stratix 10 SoC Bootloader" section, to coincide with the 18.1 release.
2018.06.18	• Updated chapter to include support for Intel Stratix 10 SoC • Replaced "Second Stage Bootloader" with "Bootloader". • Replaced "SSBL" with "FSBL" • Added the Intel Stratix 10 SoC Typical Boot Flow figure • Added the "Building the Intel Stratix 10 SoC Bootloader" section
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.7	Added an "UEFI Build Flow" section
2016.05.27	Added a link to the Intel Arria 10 SoC Secure Boot User Guide
2016.02.17	• Updated the help definition for `mkpimage` in the "Tools Usage" and "Tool Options" sections • Updated the Intel Arria 10 SoC Main BSP Settings Group table
2015.08.06	Added Intel Arria 10 SoC support

8. Hardware Library

The Intel 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 56. Hardware Library



Within the context of the SoC hardware and software ecosystem, the HWLIB is capable of supporting software development in conjunction with full featured operating systems or standalone BareMetal 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, BSP 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:

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

8.1. 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)
- HW Manager

8.1.1. 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.

8.1.2. HW Manager

The 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).

8.2. Hardware Library Reference Documentation

Reference documentation for the SoCAL and HW Manager are distributed as part of the Intel SoC FPGA EDS. The documentation is in HTML format and is located at `<SoC EDS installation directory>/ip/altera/hps/doc`. You can view this documentation with any web browser.

8.3. 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 Platform Designer (Standard) 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 Intel Quartus Prime Standard Edition, and not of Intel SoC FPGA EDS. The tool can be invoked from the SoC EDS Embedded Command Shell once Intel Quartus Prime Standard Edition 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 Intel Arria 10 SoC 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.

Send Feedback

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 Intel Quartus Prime Standard Edition documentation.

8.4. Hardware Library Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Updated the location and format to find the reference documentation for the SoCAL and HW Manager.
2018.06.18	• Updated chapter to include support for Intel Stratix 10 SoC • Replaced "Second Stage Bootloader" with "Bootloader". • Updated information about the Toolchains
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	• Updated the HW Library figure • Added a new section called "System Memory Map"
2015.08.06	Added Intel Arria 10 SoC support

9. HPS Flash Programmer User Guide

The Intel Quartus Prime software and Intel 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 Intel FPGA SoC. The programmer sends file contents over an Intel download cable, such as the Intel FPGA Download Cable 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

Note: The HPS Flash Programmer does not support the Intel Stratix 10 SoC devices. It is your responsibility to program the flash connected to HPS. Several options are possible:

- Use a bus switch to route the EPCQ-L/QSPI signals to an external master that does the programming.
- Use software running on HPS to do the programming. For example, U-Boot can be loaded with an Arm debugger or System Console, and then used to program the flash.

Note: On Intel Stratix 10 SoC, the HPS can have access to the QSPI flash memory connected to SDM. This flash is programmed using the Intel Quartus Prime Programmer tool that is part of both the Intel Quartus Prime Pro Edition and Intel SoC FPGA Embedded Development Suite.

9.1. HPS Flash Programmer Command-Line Utility

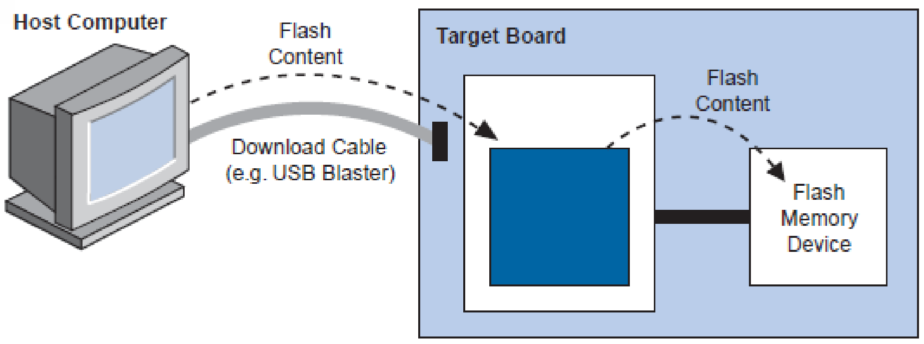
You can run the HPS flash programmer directly from the command line.

- For the Quartus Prime software, the HPS flash programmer is located in the <Quartus Prime installation directory>/quartus/bin directory.
- For the SoC EDS software, the HPS flash programmer is located in the <SoC EDS installation directory>/ qprogrammer/bin directory.

9.2. 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 57. 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.

9.3. Using the Flash Programmer from the Command Line

9.3.1. 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 17. HPS Flash Programmer Parameters

Option	Short Option	Required	Description
`--addr`	`-a`	Yes (if the start address is not 0)	This option specifies the start address of the operation to be performed.
`--cable`	`-c`	Yes	This option specifies what download cable to use. To obtain the list of programming cables, run the command "jtagconfig". It lists the available cables, like in the following example: `jtagconfig` • Intel FPGA Download Cable [USB-0] • Intel FPGA Download Cable [USB-1] • Intel FPGA Download Cable [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` "Intel FPGA Download Cable [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 automatically detects 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.
`--boot`	N/A	Yes	Option to reconfigure the HPS IOCSR and PINMUX before starting flash programming. The Intel Quartus Prime HPS Flash Programmer options are: • Warm/cold reset HPS (BootROM) so that BootROM can reconfigure the setting. FPGA (for Intel Arria 10 SoC) is `nconfig`. • Explicitly configure dedicated I/O and PINMUX Available options are: • 1 - Set Breakpoint to halt CPU, warm reset HPS [not recommended] • 2 - Set Watchpoint to halt CPU, warm reset HPS • 3 - Explicitly configure dedicated IO and PINMUX • 16 - Cold reset HPS Cyclone V SoC and Intel Arria 10 SoC support values: 1, 2 and 16. Intel Arria 10 SoC supports values: 2, 3 and 16 *Note:* For the first three options, add up integer of 16, so that the HPS cold reset is performed
`--exit_xip`	N/A	Yes (if the QSPI flash device has been put into XIP mode)	This option exits the QSPI flash device from XIP mode. A non-zero value has to be specified for the argument. For example, `quartus_hps -c <cable> -o <operation> --exit_xip=0x80`.
`--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 Sislicon 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

continued...

Option	Short Option	Required	Description
			• 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.
`--size`	`-s`	No	This option specifies the number of bytes of data to be performed by the operation. `size` is optional.
Note: The following options: `repeat` and `interval` must be used together and are optional. The HPS BOOT flow supports up to four images where each image is identical. These options duplicate the operation data; therefore you do not need embedded software to create a large file containing duplicate images.			
`--repeat`	`-t`	No	`repeat`—specifies the number of duplicate images for the operation to perform.
`--interval`	`-i`	No	`interval`—specifies the repeated address. The default value is 64 kilobytes (KB).

9.3.2. 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 8. 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. 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 10. 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 11. 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 12. 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 13. 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 erases all the flash contents.

Example 14. Erase Specified Memory Contents 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 15. 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*.

9.4. Supported Memory Devices

Table 18. QSPI Flash

Flash Device	Manufacturer	Device ID	DIE #	Density (Mb)	Voltage
M25P40	Micron	0x132020	1	4	3.3
N25Q064	Micron	0x17BA20	1	64	3.3
N25Q128	Micron	0x18BA20	1	128	3.3
N25Q128	Micron	0x18BB20	1	128	1.8
N25Q256	Micron	0x19BA20	1	256	3.3
N25Q256	Micron	0x19BB20	1	256	1.8
MT25QL512	Micron	0x20BA20	1	512	3.3
N25Q512	Micron	0x20BA20	2	512	3.3
MT25QU512	Micron	0x20BB20	1	512	1.8
N25Q512A	Micron	0x20BB20	2	512	1.8
N25Q00AA	Micron	0x21BA20	4	1024	3.3
MT25QU01G	Micron	0x21BB20	2	1024	1.8
N25Q00AA	Micron	0x21BB20	4	1024	1.8
MT25QL02G	Micron	0x22BA20	4	2048	3.3
MT25QU02G	Micron	0x22BB20	4	2048	1.8
S25FL128S	Cypress	0x182001	1	128 (64KB Sectors)	3.3
S25FL128S	Cypress	0x182001	1	128 (256KB Sectors)	3.3
S25FL256S	Cypress	0x190201	1	256 (64KB Sectors)	3.3
S25FL256S	Cypress	0x190201	1	256 (256KB Sectors)	3.3
S25FL512S	Cypress	0x200201	1	512	3.3
MX25L12835E	Macronix	0x1820C2	1	128	3.3

continued...

Send Feedback

Flash Device	Manufacturer	Device ID	DIE #	Density (Mb)	Voltage
MX25L25635	Macronix	0x1920C2	1	256	3.3
MX66L51235F	Macronix	0x1A20C2	1	512	3.3
MX66L1G45	Macronix	0x1B20C2	1	1024	3.3
MX66U51235	Macronix	0x3A25C2	1	512	1.8
GD25Q127C	GigaDevice	0x1840C8	1	128	3.3

Table 19. ONFI Compliant NAND Flash

Manufacturer	MFC ID	Device ID	Density (Gb)
Micron	0x2C	0x68	32
Micron	0x2C	0x48	16
Micron	0x2C	0xA1	8
Micron	0x2C	0xF1	8

Note: The above table contains just examples of supported devices. 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 SoC, Arria V SoC and Intel Arria 10 SoC devices. For more information, refer to the "Supported Flash Devices for Cyclone V SoC and Arria V SoC" and the "Supported Flash Devices for Intel Arria 10 SoC" web pages.

Related Information

- Supported Flash Devices for Cyclone V and Arria V SoC
- Supported Flash Devices for Arria 10 SoC

9.5. HPS Flash Programmer User Guide Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Added supported memory devices in the "QSPI Flash" table; and updated voltages for several flash devices.
2018.06.18	• Updated chapter to include support for Intel Stratix 10 SoC • Updated chapter to include support for Intel Quartus Prime Pro Edition
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Added QSPI Flash part number to the QSPI Flash table in the "Supported Memory Devices" chapter
2015.08.06	Added Intel Arria 10 SoC support

10. BareMetal Compilers

The SoC EDS provides the following BareMetal compilers:

- Arm BareMetal Compiler 5 (ARMCC)
- Arm BareMetal Compiler 6 (LLVM-based)
- Mentor Graphics Sourcery™ CodeBench Lite Edition (GCC based)

Related Information

- Mentor Graphics
- SoC EDS Getting Started Guides

10.1. Arm BareMetal Compilers

The Arm BareMetal compilers are part of the Arm DS-5 Intel SoC FPGA Edition.

The Arm BareMetal Compiler 5 targets 32-bit platforms (Cortex-A9 on Cyclone V SoC, Arria V SoC and Intel Arria 10 SoC).

The Arm BareMetal Compiler 6 targets both 32-bit platforms and 64-bit platforms (Cortex-A53 on Intel Stratix 10 SoC).

The Arm BareMetal compilers documentation is integrated in Arm DS-5 Intel SoC FPGA Edition, like with all the Arm DS-5 Intel SoC FPGA Edition components.

10.2. Mentor Code Sourcery Compiler

The BareMetal compiler that is shipped with the SoC EDS is the Mentor Graphics Sourcery CodeBench Lite. For the current version, refer to Table 1 on page 5.

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 GCC-based compiler, **arm-altera-eabi**, has the following features:

- Targets the Arm processor
- Assumes BareMetal operation
- Uses the standard Arm embedded-application binary interface (EABI) conventions.

The BareMetal 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 sets the correct environment PATH variables for the BareMetal compilation tools to be invoked. You can open the shell from the *<SoC EDS installation directory>*. After starting the shell, commands like **arm-altera-eabi-gcc** can be invoked directly. When the Arm Development Studio 5 Intel SoC FPGA Edition environment is started from the embedded command shell, it inherits the environment settings, and it can call these compilation tools directly.

You can also use the full path to the compilation tools:

```
<SoC EDS installation directory>/host_tools/mentor/gnu/arm/baremetal/bin
```

The BareMetal compiler is installed 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[4]
- Libraries Manual

Related Information

- Differences Between Standard and Professional Editions on page 6
- SoCEDSGettingStarted

10.3. BareMetal Compilers Revision History

Document Version	Changes
2019-05-16	Maintenance release
2018.09.24	Maintenance release

continued...

(4) The Getting Started Guides are located on the "SoCEDSGettingStarted" web page on the Intel FPGA Wiki.

Document Version	Changes
2018.06.18	• Updated chapter to include support for Intel Stratix 10 SoC • Added information that the Arm BareMetal Compiler 6 targets both 32-bit and 64-bit platforms
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Added a "Mentor Code Sourcery Compiler" section
2016.05.27	Maintenance release
2016.02.17	Updated the compiler version for Mentor Graphics Sourcery CodeBench Lite Edition
2015.08.06	Added Intel Arria 10 SoC support

11. SD Card Boot Utility

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, Bootloader, or both cannot be updated by copying the new file to the card; and a software tool is needed.

The SD card boot utility allows you to update the Preloader, Bootloader, or both 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.

Note: The SD Card Boot Utility tool is not needed for Intel Stratix 10 SoC devices.

Related Information

Linux Software Development Tools on page 100

11.1. 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 SoC and Arria V SoC Preloader—The Preloader file needs to have the `mkpimage` header, as required by the bootROM.
- Cyclone V SoC and Arria V SoC Bootloader—The Bootloader file needs to have the `mkimage` header, as required by the Preloader.
- Intel Arria 10 SoC Bootloader—The Bootloader needs to have the `mkpimage` header, as required by the bootROM.

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

The tool only updates the custom partition that stores the Intel 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 (such as FAT and EXT3) 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.

11.2. 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 20. 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 SoC Bootloader file to write
-B filename	Optional	Specifies Intel Arria 10 SoC 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
```

11.3. SD Card Boot Utility Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Maintenance release
2018.06.18	• Updated chapter to indicate that the SD Card Boot Utility is not needed for Intel Stratix 10 SoC devices
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Maintenance release
2015.08.06	Added Intel Arria 10 SoC support

12. Linux Software Development Tools

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 97

12.1. Linux Compiler

The Linaro Linux compiler is shipped with the SoC EDS. For the current version, refer to Table 1 on page 5.

Note: For Intel Stratix 10 SoC devices, it is recommended to use the Linaro compiler from: `https://releases.linaro.org/components/toolchain/binaries/7.2-2017.11/aarch64-linux-gnu/gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar.xz` instead of the Linux compiler 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 Intel SoC FPGA Edition, 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 is installed 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
- Differences Between Standard and Professional Editions on page 6

12.2. 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 SoC EDS and is used to create Linux device trees for the Cyclone V SoC, Arria V SoC, and Intel Arria 10 SoC systems that contain FPGA designs created using Platform Designer (Standard). The generated device tree describes the HPS peripherals, selected FPGA Soft IP, and peripherals that are board dependent.

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

Note: The Linux Device Tree Generator does not support the Intel Stratix 10 SoC devices.

Warning: The Linux Device Tree Generator is tested with and supports only the Linux kernel version targeted by the associated GSRD. It is not recommended to use the Linux Device Tree Generator if your design targets a different Linux kernel version. Instead, it is recommended to manage the Device Tree manually by using the Device Tree files provided by the kernel as a baseline, and by adding the FPGA IP and board information manually.

Related Information

- Intel FPGA Wiki
  For more information about the Linux DTG, refer to the Intel FPGA 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.

12.3. Linux Software Development Tools Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Maintenance release

continued...

Document Version	Changes
2018.06.18	• Updated chapter to include support for Intel Stratix 10 SoC • Added information about using the Linaro compiler instead of the Linux compiler • Updated the chapter to indicate that the Linux Device Tree Generator does not support the Intel Stratix 10 SoC devices.
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Maintenance release
2015.08.06	Added Intel Arria 10 SoC support

13. Support and Feedback

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

Related Information

- SoCEDSGettingStarted
  For more information about getting started, refer to the SoCEDSGettingStarted page on the Intel FPGA Wiki.
- myAltera Account Sign In

13.1. Support and Feedback Revision History

Document Version	Changes
2019.05-16	Maintenance release
2018.09.24	Maintenance release
2018.06.18	Updated chapter to include support for Intel Stratix 10 SoC
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Maintenance release
2015.08.06	Added Intel Arria 10 SoC support