Intel® SoC FPGA Embedded Development Suite User Guide

Updated for Intel® Quartus® Prime Design Suite: 19.1 Standard and 19.3 Pro
1. Introduction to the Intel® SoC FPGA EDS

1.1. Tool Versions

1.2. Differences Between Standard and Professional Editions

1.3. Overview

1.4. Hardware and Software Development Roles

1.4.1. Hardware Engineer

1.4.2. BareMetal and RTOS Developer

1.4.3. Linux Kernel and Driver Developer

1.4.4. Linux Application Developer

1.5. Hardware – Software Development Flow

1.6. Introduction to the Intel SoC FPGA EDS Revision History

2. Installing the Intel SoC FPGA EDS

2.1. Installation Folders

2.2. Installing the SoC EDS on Windows

2.2.1. Installing Cygwin on Windows*

2.3. Installing the SoC EDS on Linux

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

2.5. Installing the Intel SoC FPGA EDS Document Revision History

3. Arm Development Studio 5 Intel SoC FPGA Edition

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

3.2. BareMetal Project Management

3.2.1. BareMetal Project Management Using Makefiles

3.2.2. GCC-Based BareMetal Project Management

3.2.3. Arm Compiler 5 BareMetal Project Management

3.3. Debugging

3.3.1. Accessing Debug Configurations

3.3.2. Creating a New Debug Configuration

3.3.3. Debug Configuration Options

3.3.4. DTSL Options

3.4. Arm Development Studio 5 Intel SoC FPGA Edition Revision History

4. Intel SoC FPGA EDS Licensing

4.1. Getting the License

4.2. Activating the License

4.3. Intel SoC FPGA EDS Licensing Revision History

5. Embedded Command Shell

5.1. Embedded Command Shell Revision History


6.1. Introduction

6.2. BSP Generator

6.2.1. BSP Generation Flow

6.2.2. BSP Generator Graphical User Interface

6.2.3. BSP Generator Command Line Interface

6.2.4. BSP Files and Folders

6.2.5. BSP Settings
6.3. Bootloader Image Tool (mkpimage) ................................................................. 60
   6.3.1. Operation .................................................................................................. 60
   6.3.2. Header File Format .................................................................................. 61
   6.3.3. Tool Usage ............................................................................................... 62
   6.3.4. Output Image Layout .............................................................................. 63
6.4. U-Boot Image Tool (mkimage) ......................................................................... 65
   6.4.1. Tool Options ............................................................................................ 66
   6.4.2. Usage Examples ...................................................................................... 66
6.5. Building the Bootloader .................................................................................. 66
   6.5.1. Building the Cyclone V SoC and Arria V SoC Preloader ....................... 66
   6.5.2. Building the Intel Arria 10 SoC Bootloader ............................................ 66
   6.5.3. Building the Intel Stratix 10 SoC and Intel Agilex Devices .................... 67
6.6. Boot Tools User Guide Revision History ....................................................... 68

7. Hardware Library .............................................................................................. 69
   7.1. Feature Description ..................................................................................... 70
      7.1.1. SoC Abstraction Layer (SoCAL) .......................................................... 70
      7.1.2. HW Manager ...................................................................................... 70
   7.2. Hardware Library Reference Documentation ............................................. 71
   7.3. System Memory Map .................................................................................. 71
   7.4. Hardware Library Revision History ............................................................ 72

8. HPS Flash Programmer User Guide ................................................................. 73
   8.1. HPS Flash Programmer Command-Line Utility ......................................... 74
   8.2. How the HPS Flash Programmer Works ..................................................... 74
   8.3. Using the Flash Programmer from the Command Line ............................. 74
      8.3.1. HPS Flash Programmer ........................................................................ 74
      8.3.2. HPS Flash Programmer Command Line Examples .............................. 76
   8.4. Supported Memory Devices ...................................................................... 77
   8.5. HPS Flash Programmer User Guide Revision History ............................. 78

9. BareMetal Compilers ....................................................................................... 79
   9.1. Arm BareMetal Compilers ......................................................................... 79
   9.3. BareMetal Compilers Revision History ..................................................... 80

10. SD Card Boot Utility ........................................................................................ 82
   10.1. Usage Scenarios ...................................................................................... 82
      10.2. Tool Options ....................................................................................... 83
      10.3. SD Card Boot Utility Revision History .................................................. 84

11. Linux Software Development Tools ............................................................... 85
   11.1. Linux Compiler ....................................................................................... 85
   11.2. Linux Device Tree Generator .................................................................... 86
   11.3. Linux Software Development Tools Revision History ......................... 86

12. Support and Feedback .................................................................................... 88
   12.1. Support and Feedback Revision History ................................................. 88
1. Introduction to the Intel® SoC FPGA EDS

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 Professional Edition**—The SoC EDS Professional Edition targets the Intel Arria 10 SoC, Intel Stratix® 10 SoC, and Intel Agilex™ 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

Intel SoC FPGA EDS Licensing on page 45

1.1. Tool Versions

Table 1.  Tool Versions

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

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mentor Graphics* BareMetal GCC Compiler</td>
<td>6.2.0 (Sourcery CodeBench Lite 2019.05-16) which includes GCC version 8.3.1 20190416</td>
</tr>
<tr>
<td>Linux* Compiler</td>
<td>4.8.3 (Linaro* GCC 7.2-2017.11)</td>
</tr>
<tr>
<td>Arm* Compiler 5</td>
<td>5.06 update 6</td>
</tr>
<tr>
<td>Arm Compiler 6</td>
<td>6.10.1</td>
</tr>
<tr>
<td>Arm Development Studio 5* (DS-5*) Intel SoC FPGA Edition</td>
<td>5.29.2</td>
</tr>
</tbody>
</table>

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 11
- Intel SoC FPGA EDS Licensing on page 45
- Linux Compiler on page 85

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
- Bare-Metal application development and debugging
- OS- and Linux-based application development and debugging
- Debug systems running symmetric multiprocessing (SMP)
- Debug software targeting soft IP residing on the FPGA portion of the device

The major components of the SoC EDS include:

- Arm Development Studio 5 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
- 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 Designs (GHRD) included with the SoC EDS are not official releases and are intended to be used only as examples. For development purposes, use the official GHRD releases described in the Golden System Reference Design User Manual available on the Rocketboards website.

Related Information
- GitHub Repository

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.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Hardware Engineer</th>
<th>BareMetal Developer</th>
<th>RTOS Developer</th>
<th>Linux Kernel and Driver Developer</th>
<th>Linux Application Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arm DS-5 Intel SoC FPGA Edition Debugging</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Arm DS-5 Intel SoC FPGA Edition Tracing</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arm DS-5 Intel SoC FPGA Edition Cross Triggering</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Libraries</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Second Stage Bootloader Generator</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash Programmer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BareMetal Compiler</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux Compiler</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Linux Device Tree Generator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Note: This tool is obsolete. For more information, refer to section Linux Device Tree Generator.

Related Information
- SoC Embedded Development Suite web page
- Linux Device Tree Generator on page 86

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 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 45
- Hardware – Software Development Flow on page 8

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 45

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.

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 SoC FPGA EDS license.

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

For information about how to create a device tree, refer to the "HOWTO Create a Device Tree" web page on Rocketboards.org.

Related Information
- Intel SoC FPGA EDS Licensing on page 45
- HOWTO Create a Device Tree
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 45

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**

![Diagram of Intel Hardware-to-Software Handoff](image)
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 and Intel Agilex 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 55
- SoC Embedded Development Suite Download Page
- Linux Device Tree Generator on page 86

1.6. Introduction to the Intel SoC FPGA EDS Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
• Added Intel Agilex support.  
• Removed references to SoCEDSGettingStarted wiki.  
• Updated tool versions,  
• Removed the following chapters from the *Introduction to the Intel SoC FPGA Embedded Development Suite:*  
  — Linux Device Tree Binary  
  — Intel Stratix 10 SoC Golden Hardware Reference Design  
  — Removed the Getting Started Guides section from the document.  
• Linux Kernel and Driver Developer section: Added a link to HOWTOCreateADeviceTree on the Rocketboards.org website. |
| 2019.05.16       | • Tool Versions section: Updated the Arm Development Studio 5 (DS-5) Intel SoC FPGA Edition version.  
| 2018.09.24       | Updated the Tool versions for Arm Compiler 6 and Arm Development Studio 5 (DS-5) Intel SoC FPGA Edition |

*continued...*
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 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\19.3\embedded on Windows
  - ~/intelFPGA_pro/19.3/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\19.3\qprogrammer on Windows
  - ~/intelFPGA_pro/19.3/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**.
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 version 19.1, 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/](https://cygwin.com/).
2. Download the 64bit installer (**setup-x86_64.exe**)
3. Open a Windows Command Prompt as Administrator.
4. In the Command Prompt, change directory to `<SoC EDS Installation Folder>\cygwin_setup`.
5. Run the command:

   ```bash
   soceds-cygwin-setup.bat <path to cygwin installer>\setup-x86_64.exe
   ```

   where `<path to cygwin installer>` is the location where the **setup-x86_64.exe** was downloaded.

### 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. You can download the Linux installation program from the **Download Center for FPGAs** web page on the Intel FPGA website.

**Related Information**

Download Center for FPGAs
2.4. Installing the Arm Development Studio 5 (DS-5) Intel SoC FPGA Edition Toolkit

- Ensure that you have the proper setting to access the internet.
- Ensure that the Arm DS-5 Intel SoC FPGA Edition installer starts automatically as the last step of the SoC EDS installer.
- Intel recommends that you do not start the Arm DS-5 Intel SoC FPGA Edition installer manually.
- The installation folder is defined as `<SoC FPGA EDS installation directory>/embedded` for Windows and `<SoC FPGA EDS installation directory>/embedded` for Linux.
- `$QUARTUS_ROOTDIR` needs to be set to `<quartus install>/quartus` before launch.

For more information about the Intel SoC FPGA EDS and Intel Quartus Prime installation folders, refer to the Installation Folders section.

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. At this point, Arm Development Studio 5 Intel SoC FPGA Edition installation is complete.
8. Click Finish.

Related Information

Installation Folders on page 11

2.5. Installing the Intel SoC FPGA EDS Document Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Updated steps in the Installing Cygwin on Windows section.</td>
</tr>
</tbody>
</table>
| 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 |

continued...
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016.11.07</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2016.05.27</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2016.02.17</td>
<td>Updated the installation path to 15.1</td>
</tr>
<tr>
<td>2015.08.06</td>
<td>Added Intel Arria 10 SoC support</td>
</tr>
</tbody>
</table>
3. Arm Development Studio 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 4.

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. It is the only SoC EDS component that requires a license.

For information about the licensing options and how to obtain them, refer to the Intel SoC FPGA EDS Licensing section.

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.
- Intel SoC FPGA EDS Licensing on page 45
- Tool Versions on page 4
3.1. 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.

3.2. 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Advantages</th>
<th>Compiler Toolchain Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>Scripted compilation</td>
<td>GCC</td>
</tr>
<tr>
<td>Arm Development Studio 5 Intel SoC FPGA Edition graphical interface</td>
<td>Multiple toolchain support</td>
<td>GCC, Arm DS-5 Intel SoC FPGA Edition Compiler</td>
</tr>
</tbody>
</table>

3.2.1. 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 2. Creating a Project with Existing Code

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

3.2.2. 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.

3.2.2.1. 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.
b. **Static Library**

Select **BareMetal Library ➤ Empty Project** and click Finish.
3.2.2.2. Build Settings

Once the project is created, the project properties can be accessed by going to Project ➤ Properties.
Then, in the **Project Properties** window, the **Compilation** settings can be accessed by selecting **C/C++ Build ➤ Settings**.
Figure 8. Project Settings

![Project Settings](image)

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

- Compiler
- Assembler
- Linker

### 3.2.3. 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 4. 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 4

#### 3.2.3.1. 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 9.** Create an Empty Arm Compiler BareMetal Executable Project

b. Select **Static Library ➤ Empty Project** and then for "Toolchains", select **Arm Compiler 5**. Click **Finish**.
3.2.3.2. 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...
2. Select **Scatter File Editor ➤ Scatter File** and press **Next**.
3. Select the location of the new file, type in the file name and press **Finish**.
4. The linker script file can be edited directly as shown in the example below.
Figure 14. Linker Script Example

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

Figure 15. Editing "Scatter.scat" File Using Tools on the Outline View

3.2.3.3. Build Settings

1. Once the project is created, the project properties can be accessed by going to Project ➤ Properties.
2. Then, in the **Project Properties** window, the "Compilation" settings can be accessed by selecting **C/C++ Build ➤ Settings**.

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.

3.3. Debugging

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

3.3.1. 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 18. Accessing Debug Configurations

For more information, refer to the Debugging the HPS Boot Loader Using the Arm DS-5 Intel SoC FPGA Edition section in the Intel Stratix 10 SoC FPGA Boot User Guide.

Related Information
Intel Stratix 10 SoC FPGA Boot User Guide
3.3.2. 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 19. Create New Debug Configuration**

In the Arm Development Studio 5 Intel SoC FPGA Edition, you can assign a default name to the configuration, which you can edit.
3.3.3. 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

Online ARM DS-5 Documentation

For more information, refer to the DS-5 reference documentation located on the ARM website.

3.3.3.1. 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 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 FPGA Download Cable or the DSTREAM instance.

**Figure 21. Connection Options for BareMetal, Linux Kernel and Device Driver Debug**

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 22. Linux Application Debugging – Connect to a Running GDB Server

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 gdb:

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 23. Linux Application Debugging – Download And Debug Application

Note: For the Linux Application Debug, the Connection needs to be configured in the Remote System Explorer view.

Related Information
- DTSL Options on page 40
  For more information about the option on the Connections tab, refer to the DTSL Options section.
- Debugger Options on page 37

3.3.3.2. 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.
3.3.3.3 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 25. Debugger Settings

![Debugger Settings Figure]

3.3.3.4. RTOS Awareness

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

Figure 26. RTOS Awareness Settings

![RTOS Awareness Settings Figure]
Related Information
Keil Website
For more information about RTOS Awareness, refer to the Embedded Development Tools page on the Keil™ website.

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

Figure 27. Arguments Settings

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

Figure 28. Environment Settings
3.3.4. 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 29. Debug Configurations - DTSL Options - Edit

3.3.4.1. 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
### 3.3.4.2. 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 31.** DTSL Configuration Editor - Trace Capture > Trace Capture Method

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

**Figure 32.** DTSL Configuration Editor - Trace Capture > Timestamp Frequency
3.3.4.3. Cortex-A9 Settings

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

Figure 33. DTSL Configuration Editor - Cortex-A9

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)
3.3.4.4. STM Settings

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

**Figure 34. DTSL Configuration Editor - STM**

![DTSL Configuration Editor - STM]

Only one option is available:
- **Enable STM Trace** – check to enable STM tracing.

3.3.4.5. 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 35. DTSL Configuration Editor - ETR**

![DTSL Configuration Editor - ETR]

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.

3.3.4.6. 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.
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).

### 3.4. Arm Development Studio 5 Intel SoC FPGA Edition Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
|                  | • Removed references to the Getting Started Guides section.  
|                  | • Accessing Debug Configurations: Added a link to the Debugging the HPS Boot Loader Using the Arm DS-5 Intel SoC FPGA Edition section in the Intel Stratix 10 SoC FPGA Boot User Guide |
| 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 |
4. 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>
<thead>
<tr>
<th>Licensing Option</th>
<th>Debugging Scenarios Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arm DS-5 Intel SoC FPGA Edition Community Edition License</td>
<td>Linux application debugging over ethernet</td>
</tr>
</tbody>
</table>
| 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

4.1. Getting the License

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

*Other names and brands may be claimed as the property of others.
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.


Related Information
- SoC FPGA EDS Download Page
- Activating the License on page 46
- 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

4.2. 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.
2. If at any time it is required to change the license, select Help ➤ Arm License Manager to open the 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.
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.
   
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**.
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**.

*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 Manager.
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 43. Arm License Manager

Related Information
- ARM website
- Getting the License on page 45

4.3. Intel SoC FPGA EDS Licensing Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>

continued...
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.05.16</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2018.09.24</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2018.06.18</td>
<td>Maintenance release</td>
</tr>
</tbody>
</table>
| 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 |
5. 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:

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

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

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

### 5.1. Embedded Command Shell Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.05.16</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2018.09.24</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2018.06.18</td>
<td>Maintenance release</td>
</tr>
</tbody>
</table>
| 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 |

6.1. Introduction

The boot flow for all the Intel SoC devices includes a bootloader. The bootloader has two stages called First Stage Bootloader (FSBL) and Second Stage Bootloader (SSBL). The FSBL is also known as Preloader.

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.

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

- **BSP Generator** – enables you to get the Intel Quartus Prime handoff information and use it for the initial configuration of 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 includes instructions on how to build the bootloader for Intel Stratix 10 SoC and Intel Agilex devices. For instructions on how to build the bootloader for Cyclone V SoC, Arria V SoC and Intel Arria 10 SoC, please visit "Building Bootloader" on RocketBoards.org.
6.2. BSP Generator

The BSP Generator is used for the initial configuration of the Bootloader based on the Intel Quartus Prime handoff information for the following Intel SoC devices: Cyclone V SoC, Arria V SoC and Intel Arria 10 SoC.

Note: The BSP Generator is not used for the Intel Stratix 10 SoC and Intel Agilex devices. For Intel Stratix 10 SoC and Intel Agilex 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 create a new BSP with the default settings, based on the handoff information from Intel Quartus Prime.

The generated BSP includes a makefile, which outputs a message pointing you to visit "Building Bootloader" on RocketBoards.org to get details on how to build the bootloader.

6.2.1. BSP Generation Flow

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

6.2.1.1. Cyclone V SoC and Arria V SoC Flow

For Cyclone V SoC and Arria V SoC, the BSP Generator is required to be run first, to convert some of the handoff files into C source code. After that, you need to clone the U-Boot source code from github. Then you use an U-Boot script called qts-filter to extract the information from the handoff folder, and the set of files created by the BSP Generator, and put them in the U-Boot source code folder. Then you use the U-Boot makefile to build the SPL image. The SPL image can then be downloaded to a Flash device or FPGA RAM to be used for booting HPS.
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.

For complete instructions on how to build the bootloader for Cyclone V SoC and Arria V SoC, please visit "Building Bootloader" on RocketBoards.org.

### 6.2.1.2. Intel Arria 10 SoC Flow

For Intel Arria 10 SoC, you run the BSP Generator first, and it creates a Device Tree with default settings based on the Intel Quartus Prime handoff information. There is no source code generated, because all customization is encapsulated in the Bootloader Device Tree.

**Note:** The device tree is board specific and may need to be edited to reflect customer designs.

The next step is to clone the U-Boot source code from GitHub, and copy the generated device tree into U-Boot source code. For complete instructions, go to the Building Bootloader web page on RocketBoards.org. Then the U-Boot is compiled using the make utility and it creates the combined bootloader image, which contains both the bootloader executable and the bootloader device tree. You can download the combined image to a Flash device or FPGA RAM to use for booting the HPS.

#### Figure 47. 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. 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.

However, you do 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.

6.2.2. BSP Generator Graphical User Interface

The BSP generator GUI is deprecated, and you do not need to use it anymore. The settings are now customized directly in the bootloader files, and not from the BSP Generator. Only the `bsp-create-settings` command line tool needs to be ran, but without customized options, as they are not taken into consideration.

6.2.3. BSP Generator Command Line Interface

The BSP command-line tools can be invoked from the embedded command shell, and provide the following feature:

`bsp-create-settings`—a tool that converts the Intel Quartus Prime handoff information to source code for Cyclone V SoC and Arria V SoC, and to a Device Tree for Intel Arria 10 SoC.

**Note:** The following tools are depreciated and not used anymore:

• `bsp-update-settings`
• `bsp-query-settings`
• `bsp-generate-files`

6.2.3.1. bsp-create-settings

The `bsp-create-settings` tool converts the Intel Quartus Prime handoff information to source code for Cyclone V SoC and Arria V SoC, and to a Device Tree for Intel Arria 10 SoC.
Table 4. **User Parameters: bsp-create-settings**

<table>
<thead>
<tr>
<th>Option</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--type &lt;bsp-type&gt;</td>
<td>Yes</td>
<td>This option specifies the type of BSP. Allowed values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;spl&quot; for Cyclone V SoC/Arria V SoC Preloader</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;uboot&quot; and &quot;uefi&quot; for Intel Arria 10 SoC Bootloader</td>
</tr>
<tr>
<td>--settings &lt;filename&gt;</td>
<td>Yes</td>
<td>This option specifies the path to a BSP settings file. The file is created</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with default settings. Intel recommends that you name the BSP settings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>file settings.bsp.</td>
</tr>
<tr>
<td>--preloader-settings-dir &lt;directory&gt;</td>
<td>Yes</td>
<td>This option specifies the path to the hardware handoff files.</td>
</tr>
<tr>
<td>--bsp-dir &lt;directory&gt;</td>
<td>Yes</td>
<td>This option specifies the path where the BSP files are generated. When</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specified, bsp-create-settings generates the files after the settings file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>has been created. Intel recommends that you always specify this parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with bsp-create-settings.</td>
</tr>
<tr>
<td>--set &lt;name&gt; &lt;value&gt;</td>
<td>No</td>
<td>This option sets the BSP setting &lt;name&gt; to the value &lt;value&gt;. Multiple</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instances of this option can be used with the same command. Refer to</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>BSP Settings</strong> for a complete list of available setting names and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>descriptions.</td>
</tr>
</tbody>
</table>

### 6.2.4. 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** – deprecated, and not used
- **generated** – folder containing files generated from the hardware handoff files

For Intel Arria 10 SoC Bootloader BSPs the generated files include the following:

- **settings.bsp** – file containing all BSP settings
- **Makefile** – makefile used to compile the Bootloader, convert the Bootloader device tree file to binary, and create the combined Bootloader and Bootloader Device Tree image; for more information, refer to Preloader Compilation
- **config.mk** – deprecated, and not used
- **devicetree.dts** – Bootloader device tree, containing the Bootloader customization details, derived from the handoff files and you settings
6.2.5. BSP Settings

The BSP settings are deprecated and are not used anymore. Instead, the bootloader is configured manually, by selecting a different configuration, changing the selected configuration through "make menuconfig", editing the Device Tree or editing the source code.

6.3. 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 48. Basic Operation of the mkpimage Tool

```
<table>
<thead>
<tr>
<th>SSBL</th>
<th>+</th>
<th>CRC Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>(empty)</td>
<td></td>
<td>Header</td>
</tr>
<tr>
<td>Exception Vector</td>
<td></td>
<td>SSBL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CRC Checksum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Header</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exception Vector</td>
</tr>
</tbody>
</table>
```

6.3.1. 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.

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.

6.3.2. 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 49. Header Format Version 0**

<table>
<thead>
<tr>
<th>0x4A</th>
<th>Header Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x48</td>
<td>Reserved (0x0)</td>
</tr>
<tr>
<td>0x46</td>
<td>Program length in 32-bit words</td>
</tr>
<tr>
<td>0x45</td>
<td>Flags</td>
</tr>
<tr>
<td>0x44</td>
<td>Version</td>
</tr>
<tr>
<td>0x40</td>
<td>Validation word (0x31305341)</td>
</tr>
</tbody>
</table>

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 50. Header Format Version 1

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x52</td>
<td>Header Checksum</td>
</tr>
<tr>
<td>0x50</td>
<td>Reserved (0x0)</td>
</tr>
<tr>
<td>0x4C</td>
<td>Program entry offset</td>
</tr>
<tr>
<td>0x48</td>
<td>Program length in bytes</td>
</tr>
<tr>
<td>0x46</td>
<td>Header length in bytes</td>
</tr>
<tr>
<td>0x45</td>
<td>Flags</td>
</tr>
<tr>
<td>0x44</td>
<td>Version</td>
</tr>
<tr>
<td>0x40</td>
<td>Validation word (0x31305341)</td>
</tr>
</tbody>
</table>

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:

\[ x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1 \]

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.

6.3.3. 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

6.3.4. Output Image Layout

6.3.4.1. 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.

6.3.4.2. 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.
6.3.4.3. 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 5. Typical Arria V SoC/Cyclone V SoC Preloader Image Layout

<table>
<thead>
<tr>
<th>Offset</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x30000</td>
<td>Preloader Image 3</td>
</tr>
<tr>
<td>0x20000</td>
<td>Preloader Image 2</td>
</tr>
<tr>
<td>0x10000</td>
<td>Preloader Image 1</td>
</tr>
<tr>
<td>0x00000</td>
<td>Preloader Image 0</td>
</tr>
</tbody>
</table>

Table 6. Typical Intel Arria 10 SoC Bootloader Image Layout

<table>
<thead>
<tr>
<th>Offset</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xC0000</td>
<td>Bootloader Image 3</td>
</tr>
<tr>
<td>0x80000</td>
<td>Bootloader Image 2</td>
</tr>
<tr>
<td>0x40000</td>
<td>Bootloader Image 1</td>
</tr>
<tr>
<td>0x00000</td>
<td>Bootloader Image 0</td>
</tr>
</tbody>
</table>

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

6.3.4.3.1. 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 if the NAND block is 128 KB, then the Preloader images will need to be located at 128 KB intervals.

6.3.4.3.2. 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.
6.3.4.3.3. 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.

6.3.4.4. 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

  For more information, please refer to the Booting and Configuration chapter.

  For more information, please refer to the Booting and Configuration chapter.

  For more information, please refer to the Booting and Configuration chapter.

6.4. 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 51. 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
6.4.1. 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
```

6.4.2. Usage Examples

**Example 1. 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 2. 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
```

6.5. Building the Bootloader

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

For complete details on how to build the Cyclone V SoC and Arria V SoC bootloaders, please visit "Building Bootloader" on RocketBoards.org.

6.5.2. Building the Intel Arria 10 SoC Bootloader

For complete details on how to build the Intel Arria 10 SoC bootloaders, please visit "Building Bootloader" on RocketBoards.org.
6.5.3. Building the Intel Stratix 10 SoC and Intel Agilex Devices

On Intel Stratix 10 SoC and Intel Agilex 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.

1. Start an embedded command shell.
2. Download and setup the required toolchain:

   ```
   wget https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
   tar xf gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
   export PATH=`pwd`/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/.
   export ARCH=arm64
   export CROSS_COMPILE=aarch64-linux-gnu-
   ```

3. Retrieve the U-Boot source code from github, and the currently supported branch:

   ```
   git clone https://github.com/altera-opensource/u-boot-socfpga
   cd u-boot-socfpga
   git checkout -t -b test origin/socfpga_v2019.04
   make clean && make mrproper
   ```

4. Configure the U-Boot using the default configuration for the development kit:

   - For Intel Stratix 10 SoC:
     ```
     make socfpga_stratix10_defconfig
     ```
   - For Intel Agilex:
     ```
     make socfpga_agilex_defconfig
     ```

5. Build both U-Boot and SPL:

   ```
   make -j 24
   ```

   The following files are created:
   - FSBL hex file: `u-boot-socfpga/spl/u-boot-spl-dtb.hex`
   - SSBL image: `u-boot-socfpga/u-boot.img`

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

For 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
# 6.6. Boot Tools User Guide Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
|                  | • Added support for Intel Agilex  
|                  | • Provided instruction on how to build the bootloader now that the BSP generator GUI is deprecated  
|                  | • Removed the following sections:  
|                  | — Using .tcl Scripts  
|                  | — Cyclone V SoC and Arria V SoC BSP Settings  
|                  | — U-Boot Build Flow  
|                  | — UEFI Build Flow  
|                  | • Moved the following sections to the Boot Tools User Guide Professional Edition:  
|                  | — Intel Arria 10 SoC BSP Settings  
|                  | — Intel Arria 10 SoC Main BSP Settings Group  
|                  | — Intel Arria 10 SoC Bootloader MPU Firewall BSP Settings Group  
|                  | — Intel Arria 10 SoC Bootloader L3 Firewall BSP Settings Group  
|                  | — Intel Arria 10 SoC Bootloader FPGA-to-SDRAM Firewall BSP Settings Group  
|                  | • Removed the following BSP Generator Command Line Interfaces that were deprecated:  
|                  | — bsp-update-settings  
|                  | — bsp-query-settings  
|                  | — bsp-generate-files  
|                  | • Provided instruction on how to configure the bootloader manually now that the BSP settings are deprecated  
|                  | • Provided details on how to build the bootloaders by pointing to "Building Bootloader" on RocketBoards.org |
| 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  

**Related Information**  
7. 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 53. 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

### 7.1. 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

#### 7.1.1. 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.

#### 7.1.2. 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).

7.2. 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.

7.3. 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.
The following example demonstrates how to use the tool to generate only the file that shows the HPS A9 Core 0 point of view:

```bash
goal sopc-create-header-files ghrd_10as066n2.sopcinfo --module\n   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.

### 7.4. Hardware Library Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
   • Maintenance release |
| 2019.05.16       | Maintenance release |
| 2018.09.24       | Updated the location and format to find the reference documentation for the SoCAL and HW Manager.  
   • Updated chapter to include support for Intel Stratix 10 SoC  
   • Replaced "Second Stage Bootloader" with "Bootloader".  
   • Updated information about the Toolchains |
| 2018.06.18       | • Intel FPGA rebranding  
   • Rebranded paths and tools for the Standard and Professional versions |
| 2017.05.08       |  
   • Updated the HW Library figure  
   • Added a new section called "System Memory Map" |
| 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 |
8. 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.
8.1. 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.

8.2. 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 54. 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.

8.3. Using the Flash Programmer from the Command Line

8.3.1. 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 7. HPS Flash Programmer Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Short Option</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--addr</td>
<td>-a</td>
<td>Yes (if the start address is not 0)</td>
<td>This option specifies the start address of the operation to be performed.</td>
</tr>
<tr>
<td>--cable</td>
<td>-c</td>
<td>Yes</td>
<td>This option specifies what download cable to use. To obtain the list of programming cables, run the command &quot;jtagconfig&quot;. 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 &quot;-c&quot; parameter can be the number of the programming cable, or its name. The following are valid examples for the above case: • -c 1 • -c &quot;Intel FPGA Download Cable [USB-2]&quot;</td>
</tr>
<tr>
<td>--device</td>
<td>-d</td>
<td>Yes</td>
<td>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.</td>
</tr>
<tr>
<td>--boot</td>
<td>N/A</td>
<td>Yes</td>
<td>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</td>
</tr>
<tr>
<td>--exit_xip</td>
<td>N/A</td>
<td>Yes (if the QSPI flash device has been put into XIP mode)</td>
<td>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 &lt;cable&gt; -o &lt;operation&gt; --exit_xip=0x80.</td>
</tr>
<tr>
<td>--operation</td>
<td>-o</td>
<td>Yes</td>
<td>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 &lt;BlankCheck&gt; flash</td>
</tr>
</tbody>
</table>

*continued...*
### Option Descriptions

<table>
<thead>
<tr>
<th>Option</th>
<th>Short Option</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PV</td>
<td>--program</td>
<td>Yes</td>
<td>Program and verify flash</td>
</tr>
<tr>
<td>BPV</td>
<td>--BPV</td>
<td>Yes</td>
<td>Program (blank-check) and verify flash</td>
</tr>
<tr>
<td>X</td>
<td>--X</td>
<td>Yes</td>
<td>Examine flash</td>
</tr>
</tbody>
</table>

- **Note:** The program begins with erasing the flash operation before programming the flash by default.

---

<table>
<thead>
<tr>
<th>Option</th>
<th>Short Option</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--size</td>
<td>-s</td>
<td>No</td>
<td>This option specifies the number of bytes of data to be performed by the operation. size is optional.</td>
</tr>
</tbody>
</table>

- **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** | No | repeat—specifies the number of duplicate images for the operation to perform. |
- **interval** | No | interval—specifies the repeated address. The default value is 64 kilobytes (KB). |

### 8.3.2. 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 3. 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 4. 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 5. 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 6. 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 7. 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 8. 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 9. 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 10. 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 $M$.

8.4. Supported Memory Devices

<table>
<thead>
<tr>
<th>Flash Device</th>
<th>Manufacturer</th>
<th>Device ID</th>
<th>DIE #</th>
<th>Density (Mb)</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>M25P40</td>
<td>Micron</td>
<td>0x132020</td>
<td>1</td>
<td>4</td>
<td>3.3</td>
</tr>
<tr>
<td>N25Q064</td>
<td>Micron</td>
<td>0x17BA20</td>
<td>1</td>
<td>64</td>
<td>3.3</td>
</tr>
<tr>
<td>N25Q128</td>
<td>Micron</td>
<td>0x18BA20</td>
<td>1</td>
<td>128</td>
<td>3.3</td>
</tr>
<tr>
<td>N25Q128</td>
<td>Micron</td>
<td>0x18BB20</td>
<td>1</td>
<td>128</td>
<td>1.8</td>
</tr>
<tr>
<td>N25Q256</td>
<td>Micron</td>
<td>0x19BA20</td>
<td>1</td>
<td>256</td>
<td>3.3</td>
</tr>
<tr>
<td>N25Q256</td>
<td>Micron</td>
<td>0x19BB20</td>
<td>1</td>
<td>256</td>
<td>1.8</td>
</tr>
<tr>
<td>MT25QL512</td>
<td>Micron</td>
<td>0x20BA20</td>
<td>1</td>
<td>512</td>
<td>3.3</td>
</tr>
<tr>
<td>N25Q512</td>
<td>Micron</td>
<td>0x20BA20</td>
<td>2</td>
<td>512</td>
<td>3.3</td>
</tr>
<tr>
<td>MT25QU512</td>
<td>Micron</td>
<td>0x20BB20</td>
<td>1</td>
<td>512</td>
<td>1.8</td>
</tr>
<tr>
<td>N25Q512A</td>
<td>Micron</td>
<td>0x20BB20</td>
<td>2</td>
<td>512</td>
<td>1.8</td>
</tr>
<tr>
<td>N25Q00AA</td>
<td>Micron</td>
<td>0x21BA20</td>
<td>4</td>
<td>1024</td>
<td>3.3</td>
</tr>
<tr>
<td>MT25QU01G</td>
<td>Micron</td>
<td>0x21BB20</td>
<td>2</td>
<td>1024</td>
<td>1.8</td>
</tr>
<tr>
<td>N25Q00AA</td>
<td>Micron</td>
<td>0x21BB20</td>
<td>4</td>
<td>1024</td>
<td>1.8</td>
</tr>
<tr>
<td>MT25QL02G</td>
<td>Micron</td>
<td>0x22BA20</td>
<td>4</td>
<td>2048</td>
<td>3.3</td>
</tr>
<tr>
<td>MT25QU02G</td>
<td>Micron</td>
<td>0x22BB20</td>
<td>4</td>
<td>2048</td>
<td>1.8</td>
</tr>
<tr>
<td>S25FL128S</td>
<td>Cypress</td>
<td>0x182001</td>
<td>1</td>
<td>128 (64KB Sectors)</td>
<td>3.3</td>
</tr>
<tr>
<td>S25FL128S</td>
<td>Cypress</td>
<td>0x182001</td>
<td>1</td>
<td>128 (256KB Sectors)</td>
<td>3.3</td>
</tr>
<tr>
<td>S25FL256S</td>
<td>Cypress</td>
<td>0x190201</td>
<td>1</td>
<td>256 (64KB Sectors)</td>
<td>3.3</td>
</tr>
<tr>
<td>S25FL256S</td>
<td>Cypress</td>
<td>0x190201</td>
<td>1</td>
<td>256 (256KB Sectors)</td>
<td>3.3</td>
</tr>
<tr>
<td>S25FL512S</td>
<td>Cypress</td>
<td>0x200201</td>
<td>1</td>
<td>512</td>
<td>3.3</td>
</tr>
<tr>
<td>MX25L1283SE</td>
<td>Macronix</td>
<td>0x1820C2</td>
<td>1</td>
<td>128</td>
<td>3.3</td>
</tr>
</tbody>
</table>

*continued...*
## Table 9. ONFI Compliant NAND Flash

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>MFC ID</th>
<th>Device ID</th>
<th>Density (Gb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micron</td>
<td>0x2C</td>
<td>0x68</td>
<td>32</td>
</tr>
<tr>
<td>Micron</td>
<td>0x2C</td>
<td>0x48</td>
<td>16</td>
</tr>
<tr>
<td>Micron</td>
<td>0x2C</td>
<td>0xA1</td>
<td>8</td>
</tr>
<tr>
<td>Micron</td>
<td>0x2C</td>
<td>0xF1</td>
<td>8</td>
</tr>
</tbody>
</table>

**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

# 8.5. HPS Flash Programmer User Guide Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Maintenance release</td>
</tr>
<tr>
<td>2019.05.16</td>
<td>HPS Flash Programmer: Documented --boot=18 for QSPI programming</td>
</tr>
<tr>
<td>2018.09.24</td>
<td>Added supported memory devices in the &quot;QSPI Flash&quot; table; and updated voltages for several flash devices.</td>
</tr>
<tr>
<td>2018.06.18</td>
<td>• Updated chapter to include support for Intel Stratix 10 SoC</td>
</tr>
<tr>
<td></td>
<td>• Updated chapter to include support for Intel Quartus Prime Pro Edition</td>
</tr>
<tr>
<td>2017.05.08</td>
<td>• Intel FPGA rebranding</td>
</tr>
<tr>
<td></td>
<td>• Rebranded paths and tools for the Standard and Professional versions</td>
</tr>
<tr>
<td>2016.11.07</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2016.05.27</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2016.02.17</td>
<td>Added QSPI Flash part number to the QSPI Flash table in the &quot;Supported Memory Devices&quot; chapter</td>
</tr>
<tr>
<td>2015.08.06</td>
<td>Added Intel Arria 10 SoC support</td>
</tr>
</tbody>
</table>
9. BareMetal Compilers

The SoC EDS provides the following BareMetal compilers:

- Arm BareMetal Compiler 5 (ARMCC)
- Arm BareMetal Compiler 6 (LLVM-based)

Related Information
Mentor Graphics

9.1. 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.


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 4.

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
- Libraries Manual

**Related Information**

Differences Between Standard and Professional Editions on page 4

### 9.3. BareMetal Compilers Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
• Removed reference to SoC EDS Getting Started Guides |
| 2019.05.16       | Maintenance release |
| 2018.09.24       | Maintenance release |
| 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 |

*continued...*
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016.05.27</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2016.02.17</td>
<td>Updated the compiler version for Mentor Graphics Sourcery CodeBench Lite Edition</td>
</tr>
<tr>
<td>2015.08.06</td>
<td>Added Intel Arria 10 SoC support</td>
</tr>
</tbody>
</table>
10. 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 85

10.1. 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.

10.2. 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 10. Command Line Options

<table>
<thead>
<tr>
<th>Command line Argument</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p filename</td>
<td>Optional</td>
<td>Specifies Preloader file to write</td>
</tr>
<tr>
<td>-b filename</td>
<td>Optional</td>
<td>Specifies Cyclone V or Arria V SoC Bootloader file to write</td>
</tr>
<tr>
<td>-B filename</td>
<td>Optional</td>
<td>Specifies Intel Arria 10 SoC Bootloader file to write</td>
</tr>
<tr>
<td>-a write</td>
<td>Required</td>
<td>Specifies action to take. Only &quot;write&quot; action is supported. Example: &quot;-a write&quot;</td>
</tr>
<tr>
<td>disk_file</td>
<td>Required (unless -d option is used)</td>
<td>Specifies disk image file or physical disk to write. 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.</td>
</tr>
<tr>
<td>-d</td>
<td>Optional</td>
<td>specify disk drive letter to write to. Example: &quot;-d E&quot;. When using this option, the disk_file option cannot be specified.</td>
</tr>
<tr>
<td>-h</td>
<td>Optional</td>
<td>Displays help message and exits</td>
</tr>
<tr>
<td>--version</td>
<td>Optional</td>
<td>Displays script version number and exits</td>
</tr>
</tbody>
</table>

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

## 10.3. SD Card Boot Utility Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
                   • Maintenance release |
| 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 |
11. 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 82

### 11.1. Linux Compiler

The Linaro Linux compiler is shipped with the SoC EDS. For the current version, refer to Table 1 on page 4.

*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:

```
```

The documents are provided as HTML files. Some of the provided documents are:

- Compiler manual
- Assembler manual
- Linker manual
11.2. 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. 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.

**Warning:** The Linux Device Tree Generator is obsolete and does not support Intel Stratix 10 SoC and newer devices. However, it is tested with and supports only the Linux kernel version targeted by the associated GSRD. Intel does not recommend that you use the Linux Device Tree Generator if your design targets a different Linux kernel version. Instead, Intel recommends that you 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.

Refer to *HOWTO Create A Device Tree*, which supports all kernels.

**Related Information**
- Linux Device Tree Generator User Guide
  For more details, navigate to the Device Tree Generator User Guide located on the "Device Tree Generator Documentation" web page on the Rocketboards website.
- HOWTO Create a Device Tree

11.3. Linux Software Development Tools Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>2019.05.16</th>
<th>Maintenance release</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.09.24</td>
<td>Maintenance release</td>
</tr>
</tbody>
</table>

*continued...*
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 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 |
12. 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
myAltera Account Sign In

12.1. Support and Feedback Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Removed reference to SoC EDS Getting Started Guides</td>
</tr>
<tr>
<td>2019.05.16</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2018.09.24</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2018.06.18</td>
<td>Updated chapter to include support for Intel Stratix 10 SoC</td>
</tr>
</tbody>
</table>
| 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 |