1. Boot Flow Overview ........................................................................................................ 3

2. System Requirements ..................................................................................................... 5
   2.1. Minimum Hardware Requirements ............................................................................ 5
   2.2. Minimum Software Requirements ............................................................................. 5
   2.3. Optional Virtual Platforms ....................................................................................... 5

3. Getting Started ............................................................................................................... 6
   3.1. Installing Software Components .............................................................................. 6
       3.1.1. Installing the Intel SoC EDS ........................................................................ 6
       3.1.2. Installing the Compiler Toolchain ................................................................. 6
   3.2. Building the Secure Monitor .................................................................................... 6
       3.2.1. User Configuration ..................................................................................... 7
       3.2.2. Getting the Arm Trusted Firmware Source Code ............................................. 7
       3.2.3. Building the BL31 image ............................................................................. 8
   3.3. Building the UEFI Boot Loader ................................................................................. 9
       3.3.1. Prerequisites ............................................................................................. 9
       3.3.2. Obtaining the UEFI Source Code................................................................ 10
       3.3.3. Compiling the UEFI Source Code with the Linaro Tool Chain........................... 10
       3.3.4. UEFI Generated Files ................................................................................ 12

4. Running UEFI on Intel Stratix 10 Hardware .................................................................. 13
   4.1. Running on a Physical Board with ATF and UEFI Bootloader ................................. 13
       4.1.1. Generate a .sof file with ATF ...................................................................... 13
       4.1.2. Creating an SD Card Image with PEI.ROM and DXE.ROM ............................... 13
       4.1.3. Running the Secure Monitor ...................................................................... 14
       4.1.4. Debugging with DS-5 ............................................................................... 14
   4.2. Booting Linux ...................................................................................................... 17
       4.2.1. Booting from the DXE Console ................................................................... 18

1. Boot Flow Overview

This document provides comprehensive information on Unified Extensible Firmware Interface (UEFI) boot loader for Intel Stratix 10 SoC.

The Intel Stratix 10 SoC provides a secure boot flow, consisting of:

- The boot ROM
- The secure device manager (SDM)
- The Secure Monitor
- The UEFI boot loader

The Intel Stratix 10 SoC secure boot flow ensures that the system boot loader is signed with a cryptographic key, validated by the firmware.

The Secure Monitor stage also implements the TrustZone* model of secure partitioning. This model divides the software environment into two isolated partitions, called the secure world and the non-secure world. The two worlds can only communicate with each other through the Secure Monitor.

The binary image of the UEFI boot loader can be stored on Quad SPI flash, NAND flash, or an SD/MMC card. On board power-up, the secure device manager (SDM) loads the Secure Monitor directly onto Hard Processor System (HPS) on-chip RAM. Then the Secure Monitor loads the UEFI boot loader in HPS DDR memory.

The Secure Monitor tasks include:

- Initializing DDR SDRAM memory
- Configuring low level hardware, such as PLL, IOs, and pin MUXes, needed by nonsecure world software

The UEFI boot loader tasks include:

- Providing Ethernet support
- Supporting basic hardware diagnostic features
- Fetching subsequent boot software such as the operating system package or kernel image.

Note: For non-secure boot, the operating system package can include kernel image, device tree blob and filesystem. For secure boot it can be a secure kernel.
Figure 1. UEFI Boot Flow Overview

- EL0: App
- EL1: OS
- EL2: UEFI PEI & Dxe (SSBL), DDR
- EL3: Arm Trusted FW (FSBL), OCRAM
- SDM BootROM
- Configuration Mgmt FW
2. System Requirements

To load and execute the Intel Stratix 10 SoC Unified Extensible Firmware Interface (UEFI) boot loader, your system must meet the following requirements.

2.1. Minimum Hardware Requirements

- Windows PC or Linux workstation with the following configuration:
  - Serial terminal, such as Minicom for Linux or Tera Term for Windows
  - microSD card slot or microSD card writer or SD capable writer with SD to microSD converter

Table 1. Platform Capabilities

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Able to compile the UEFI boot loader</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Able to compile the Secure Monitor</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

2.2. Minimum Software Requirements

- Intel SoC FPGA Embedded Development Suite (SoC EDS) v18.1
- Linaro aarch64-linux-gnu-gcc toolchain version 4.8.3 20140401

Related Information
https://releases.linaro.org/archive/14.04/components/toolchain/binaries/
Linaro toolchain binaries

2.3. Optional Virtual Platforms

Virtual platforms are available for the Intel Stratix® 10 SoC. A virtual platform enables you to develop and test software before target hardware is available.

The following virtual platforms are available for the Intel Stratix 10 SoC:

- Wind River Simics*. The Simics virtual platform is available under license, free of charge. To get the Simics virtual platform and documentation, contact Wind River at www.windriver.com/products/simics.
- Intel Stratix 10 SoC Virtual Platform v. 1.3 from Mentor Graphics*. The Intel Stratix 10 SoC Virtual Platform and documentation are available at Mentor* Embedded for Intel Stratix FPGAs and SoCs.

---

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

3.1. Installing Software Components

3.1.1. Installing the Intel SoC EDS

You must install the Intel SoC EDS v18.1 on your machine.

Download the SoC EDS from the Download Center for FPGAs.

3.1.2. Installing the Compiler Toolchain

You compile the UEFI boot loader and the Secure Monitor with the GNU Toolchain (EABI Release) for Arm* Processors.

The supported compiler toolchain is Linaro* aarch64-linux-gnu-gcc (crosstool-NG linaro-1.13.1-4.8-2014.04 - Linaro GCC 4.8-2014.04) 4.8.3 20140401 (prerelease).

You can download the toolchain from https://releases.linaro.org/archive/14.04/components/toolchain/binaries/.

- Linux: gcc-linaro-aarch64-linux-gnu-4.8-2014.04_linux.tar
- Windows: gcc-linaro-aarch64-linux-gnu-4.8-2014.04_win32.zip

Related Information

https://releases.linaro.org/archive/14.04/components/toolchain/binaries/
Linaro toolchain downloads

3.2. Building the Secure Monitor

As security becomes more and more important, a secured boot solution becomes a requirement in the embedded world. To ensure comprehensive security and a trusted platform, secure partitioning is required. The Intel Stratix 10 device achieves secure partitioning by implementing the TrustZone model with Arm Trusted Firmware (ATF). The TrustZone model splits the computing environment into two isolated worlds, the secure world and normal world, which are linked by a software monitor called the Secure Monitor. The two worlds have separated logical address space and peripherals. Communication between the two worlds is only possible by calling the privileged Secure Monitor call (SMC) instruction.

The full secure boot solution is:

- BootRom
- Secure Device Manager
- Secure Monitor
Secure Monitor mode is a privileged mode and is always secure regardless of the state of the NS bit. The Secure Monitor is code that runs in Secure Monitor mode and processes switches to and from the Secure world. The overall security of the software relies on the security of this code along with the Secure boot code.

**Related Information**

www.trustedfirmware.org
- General information about Arm Trusted Firmware

### 3.2.1. User Configuration

You can find all platform configurations in arm-trusted-firmware/plat/intel/soc/stratix10/platform_def.h.

For user configuration, you should only modify the first part of this file.

```c
/******************************************
User configuration
******************************************/
#ifndef EMULATOR
#define EMULATOR
#endif
#ifndef VIRTUAL_PLATFORM
#define VIRTUAL_PLATFORM
#endif
#ifndef ENABLE_HANDOFF
#define ENABLE_HANDOFF
#endif
#ifndef PLAT_SEMIHOSTING_ENABLE
#define PLAT_SEMIHOSTING_ENABLE
#endif
#ifndef PLAT_NS_IMAGE_OFFSET
#define PLAT_NS_IMAGE_OFFSET 0x50000
#endif
#ifndef PLAT_HANDOFF_OFFSET
#define PLAT_HANDOFF_OFFSET 0xFFE3F000
#endif
#ifndef BOOT_SOURCE
#define BOOT_SOURCE BOOT_SOURCE_SDMMC
#endif
```

**Note:**

To change the boot filename or offset, you can change the `#define` in this file.

### 3.2.2. Getting the Arm Trusted Firmware Source Code

The ATF source is at https://github.com/altera-opensource/arm-trusted-firmware. To get the ATF source code, simply run the following steps:

1. Open a terminal.
2. Create a new directory to check out the ATF source code from GitHub.
3. Change to this working directory and clone the ATF source from the Git trees as follows:

   ```bash
   $ git clone https://github.com/altera-opensource/arm-trusted-firmware
   ```

4. When completed, change to the arm-trusted-firmware folder and perform a Git check out as follows:

   ```bash
   $ cd arm-trusted-firmware
   $ git checkout -t -b test_atf origin/socfpga_v1.4
   ```

**Related Information**

- [Building the BL31 image](#) on page 8
- [Compiling the UEFI Source Code with the Linaro Tool Chain](#) on page 10
3.2.3. Building the BL31 image

This section describes how to build the ATF with the Linaro GCC compiler.

To start building the ATF with the Linaro GCC compiler, simply run the following steps:

1. Change your directory to the ATF source code location as follows:

   $ cd arm-trusted-firmware

2. Set the GCC path and environment variable CROSS_COMPILE to Linaro cross compile as follows:

   $ export PATH=<your gcc directory>/gcc-linaro-aarch64-linux-gnu-4.8-2014.04_linux/bin:$PATH
   $ export CROSS_COMPILE=aarch64-linux-gnu-

3. Remove the build tree completely as follows:

   $ make realclean

4. Build the ATF by using the following command:

   $ make PLAT=stratix10 DEBUG=1

   **Note:** You must enable debugging GCC 4.8. You can disable debugging for GCC 4.9 and above.

5. The following messages appear when the ATF build is successful:

   AS drivers/console/console.S
   AS drivers/ti/uart/16550_console.S
   AS lib/cpus/aarch64/aem_generic.S
   AS lib/cpus/aarch64/cortex_a53.S
   AS lib/semihosting/aarch64/semihosting_call.S
   AS plat/common/aarch64/platform_mp_stack.S
   AS plat/intel/soc/stratix10/aarch64/plat_helpers.S
   AS bl31/aarch64/bl31_cadpoint.S
   AS bl31/aarch64/runtime_exceptions.S
   AS bl31/aarch64/crash_reporting.S
   AS lib/el3_runtime/aarch64/cpu_data.S
   AS lib/cpus/aarch64/cpu_helpers.S
   AS lib/locks/exclusive/aarch64/spinlock.S
   AS lib/pci/aarch64/pci helpers.S
   AS lib/el3_runtime/aarch64/context.S
   AS common/aarch64/debug.S
   AS lib/aarch64/cache helpers.S
   AS lib/aarch64/misc helpers.S
   AS plat/common/aarch64/platform_helpers.S
   PP bl31/bl31.ld.S
   LD build/stratix10/debug/bl31/bl31.elf
   BIN build/stratix10/debug/bl31.bin

   Built build/stratix10/debug/bl31.bin successfully

6. The table below lists the Secure Monitor output files.
Table 2. **Descriptions of Secure Monitor Files**

<table>
<thead>
<tr>
<th>File Path and Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\build\stratix10\release\bl31.bin</td>
<td>Generated binary file</td>
</tr>
<tr>
<td>\build\stratix10\release\bl31\bl31.elf</td>
<td>Generated elf file</td>
</tr>
<tr>
<td>\build\stratix10\debug\bl31.bin</td>
<td>Generated debug binary file</td>
</tr>
<tr>
<td>\build\stratix10\debug\bl31\bl31.elf</td>
<td>Generated debug elf file</td>
</tr>
</tbody>
</table>

**Note:** The first two files in the table above are generated if you run make PLAT=stratix10 without the DEBUG option.

### 3.3. Building the UEFI Boot Loader

To build a UEFI boot loader, you obtain the UEFI source code and compile the UEFI source with the supported toolchain.

The Unified Extensible Firmware Interface (UEFI) is a standardized firmware specification that simplifies and secures platform initialization and firmware bootstrap operations. UEFI is currently developed and supported by representatives from more than 250 industry-leading technology companies. Arm and the Linaro Enterprise Group are also promoting the use of UEFI on Arm architecture, because the UEFI specification helps standardize the boot process for Arm processor-based platforms.

UEFI technology is future-proofed through standardization of firmware design rather than proprietary firmware design. UEFI specifications promote business and technological efficiency, improve performance and security, facilitate interoperability between devices, platforms and systems and comply with next-generation technologies. The UEFI specification is peer-reviewed and published, allowing developers to write firmware once per platform and reuse it without much modification. This reuse results in cost and time savings during boot loader development.

This framework uses the BSD license, permitting you to optionally commercialize your implementation with minimal legal issues.

You can compile the UEFI source code either in a Windows or in a Linux system.

#### 3.3.1. Prerequisites

**For Windows System**

If you are using a Windows system, you must have Git installed. You can download Git from [www.git-scm.com/download/win](http://www.git-scm.com/download/win).

**For Linux System**

Building the UEFI requires additional Linux packages. Depending on your Linux distribution, the command to install the packages is different:

If you are using a Ubuntu distribution, type:

```
$ sudo apt-get install uuid-dev build-essential
```
If you using a Fedora distribution, type:

```bash
$ sudo yum install uuid-devel libuuid-devel
```

For building UEFI, the Python package is required. If Python is not already available on your system, running the commands from the SoC EDS Embedded Command Shell provides the required Python dependency.

### 3.3.2. Obtaining the UEFI Source Code

The UEFI source code is located in GitHub. The following steps show you how to get the UEFI source code.

1. Open a terminal.
2. Clone the UEFI source from the Git trees.
   ```bash
   $ git clone https://github.com/altera-opensource/uefi-socfpga
   ```
3. When completed, change to the `uefi-socfpga` folder and perform a Git checkout.
   ```bash
   $ cd uefi-socfpga
   $ git checkout -t -b test_uefi origin/socvp_socfpga_udk2015
   ```

### 3.3.3. Compiling the UEFI Source Code with the Linaro Tool Chain

#### 3.3.3.1. Windows System

To compile the UEFI source code with the Linaro toolchain in a Windows system:

1. Open the command prompt.
2. Go to your working directory and set `SOCEDS_DEST_ROOT` to the location of your SoC EDS.
   ```bash
   $ cd <your_working_directory>\uefi-socfpga
   $ set SOCEDS_DEST_ROOT=<your_SOCEDS_location>
   ```
3. Set the GCC path to the location of the compiler toolchain.
   ```bash
   $ set PATH=<your_Linaro GCC Toolchain_location>%PATH%
   ```

   **Note:** If you encountered a GCC error while compiling the UEFI source code after setting the path, you can edit the `setup.bat` file manually by entering the following command to use the full compiler path.

   ```bash
   set GCC48_ARM_PREFIX=%DS5_ROOT%\sw\gcc\bin\arm-linux-gnueabihf-
   set GCC48_AARCH64_PREFIX=<your working directory>\uefi_17v0_window
   \gcc-linaro-aarch64-linux-gnu-4.8-2014.04_win32\bin\aarch64-linux-
   gnu-
   ```
4. Run the setup command.
   ```bash
   $ setup.bat
   ```
5. Build the UEFI by entering the following command:
   ```bash
   $ make device=s10
   ```
6. The command prompt displays **Build Done** after the UEFI compilation is successful.

**Figure 2. Command Prompt Display**

![Command Prompt Display](image)

### 3.3.3.2. Linux System

This section explains how to compile the UEFI source code with the Linaro toolchain in a Linux system:

1. Open a terminal and enter the following command:

   ```
   $ cd <your_uefi_directory>/uefi-socfpga
   $ export PATH=<your gcc directory>/gcc-linaro-aarch64-linux-gnu-4.8-2014.04_linux/bin:$PATH
   ```

2. Clean the entire `<your_uefi_directory>/uefi-socfpga/Build/` folder and `BaseTools` folder by entering the following command:

   ```
   $ make clean
   ```

3. Compile the UEFI boot loader for the Intel Stratix 10 SoC device by entering the following command:

   ```
   $ make DEVICE=s10
   ```

4. Your terminal displays a "Build Done" message after the UEFI is successfully compiled.
### 3.3.4. UEFI Generated Files

Compiling the UEFI source code creates the following files in the `/uefi-socfpga/Build/Stratix10SoCPkg/RELEASE_GCC48` folder:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEI.ROM</td>
<td>This is the UEFI Pre-EFI Initialization (PEI) phase image which acts as a second stage boot loader. This file is programmed onto the flash daughter card.</td>
</tr>
<tr>
<td>load_uefi_fw.ds</td>
<td>This is the DS-5 script template. It is imported to the DS-5 tool and loads the UEFI firmware for debug and development purposes. This script loads the debug symbols. It supports the GCC compiler. ARMCC is not supported.</td>
</tr>
<tr>
<td>DXE.ROM</td>
<td>This file loads the optional second stage of the UEFI boot loader when you want to boot the UEFI shell and utilize the TFTP feature or run a UEFI application.</td>
</tr>
</tbody>
</table>
4. Running UEFI on Intel Stratix 10 Hardware

4.1. Running on a Physical Board with ATF and UEFI Bootloader

This section describes how to run the Secure Monitor on a physical board.

4.1.1. Generate a .sof file with ATF

1. Get a .sof file from the $SOCEDS_DEST_ROOT installation directory.
2. Convert the binary file bl31.bin, generated in Building the BL31 Image.

```
$ aarch64-linux-gnu-objcopy -I binary -O ihex - \\
--change-addresses 0xffe0000 bl31.bin bl31.hex
```
3. Include the bootloader into the .sof file as follows:

```
$ quartus_cpf - -bootloader=bl31.hex \\
ghrd_1sx2801u3f5013vg.sof ghrd_1sx2801u3f5013vg_hps.sof
```

4.1.2. Creating an SD Card Image with PEI.ROM and DXE.ROM

1. Generate PEI.ROM and DXE.ROM as in Building the UEFI Boot Loader.
3. Program the prebuilt SD card image into SD card.
4. Copy PEI.ROM and DXE.ROM to the FAT partition of the SD card.

Related Information

Building the BL31 image on page 8
4.1.3. Running the Secure Monitor

1. Power up the board after the SD card is inserted.
2. Open Quartus programmer and program the board with the .sof file generated in *Generating a .sof File with ATF*.

![Quartus Programmer](image)

The board boots up from the ATF and automatically loads PEI.ROM and DXE.ROM from SDMMC to boot to the DXE phase.

**Related Information**

Generate a .sof file with ATF on page 13

4.1.4. Debugging with DS-5

This section describes how to load ATF and the UEFI bootloader to the physical board through the DS-5 debugger.

1. Launch eclipse by using the following command:

   ```bash
   $ eclipse &
   ```

2. Switch to the DS-5 Debug perspective as follows:
3. Create an S10 Separate JTAG DSTREAM debug configuration as shown in the following figure.

4. Connect to the target when the configuration is complete.

5. At the DS-5 command console, run the following commands:

   $ interrupt
   $ restore <Path to bl31.bin> binary 0xffe00000
   $ set var $pc=0xffe00000
   $ continue

   Note: If semihosting is enabled in platform_def.h for building the bl31 image, DS-5 must enable it too with the command set semihosting enable true before the command continue to continue the boot up process.

   This step loads the ATF to the board and continues the boot up process. It loads PEI.ROM and DXE.ROM from SDMMC automatically.
6. If you want to use the debugger to load PEI.ROM and DXE.ROM to memory, modify the configuration in your platform_def.h file and bl31_plat_setup.c file before building the bl31 image.

7. Open the platform_def.h file located in arm-trusted-firmware/plat/intel/soc/stratix10/platform_def.h and enable semihosting as follows:

   # define PLAT_SEMIHOSTING_ENABLE

8. Open arm-trusted-firmware/plat/intel/soc/stratix10/bl31_plat_setup.c and comment out the line shown below:

   //LoadBootImageFile (PLAT_NS_IMAGE_NAME, PLAT_NS_IMAGE_OFFSET);

9. Regenerate the bl31 image and follow Step 5 on page 15 to load the ATF to the physical board. Interrupt the processor through the DS-5 debugger after memory initialization succeeds, as follows:

   $ interrupt
10. Load PEI.ROM and DXE.ROM to the physical board with the DS-5 debugger by using the following command:

```
$ restore <Path_to_PEI.ROM> binary 0x50000
$ restore <Path_to_DXE.ROM> binary 0x02000000
$ continue
```

4.2. Booting Linux

This section shows you how to boot Linux after UEFI enters the DXE phase.
4.2.1. Booting from the DXE Console

1. Boot the board up to the DXE phase, as described in Running the Secure Monitor.
2. Once the DXE phase is loaded, enter the following command to boot Linux:

```
$ Linuxloader fs1:Image -d fs1:soxfpga_stratix10_socdk.dtb -c "console=ttyS0,115200 root=/dev/mmcblk0p2 rwrootwait"
```

Note: Make sure Linux image is stored in the SD card.

Related Information
Running the Secure Monitor on page 14
5. Document Revision History for Intel Stratix 10 SoC
UEFI Boot Loader User Guide

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| March 2019 | 2019.03.28 | • Added new section, [Optional Virtual Platforms](#) on page 5  
• Added new section, [Building the Secure Monitor](#) on page 6, to describe new boot stage and secure boot  
• Updated [Boot Flow Overview](#) on page 3 for the Secure Monitor  
• Updated [UEFI Generated Files](#) on page 12  
• Removed sections [Intel Stratix 10 SoC Virtual Platform](#) and [Booting PXE using Intel Stratix 10 SoC Virtual Platform](#), and other detailed documentation of the Intel Stratix 10 SoC Virtual Platform.  
• Added new section, [Running UEFI on Intel Stratix 10 Hardware](#) on page 13 |
| June 2017  | 2017.06.19 | Initial release                                                                                                                                 |

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