1. Introduction

This user guide describes the Intel® Stratix® 10 SoC FPGA boot flow, boot sources, and how to generate a bitstream required for successful booting of the device. The details provided in this boot user guide include:

- The typical boot flows and boot stages of the Intel Stratix 10 SoC FPGA.
- The supported system layout for different hard processor system (HPS) boot modes.
- How to use Intel Quartus® Prime Pro Edition to generate the configuration bitstream.
- Examples of how to generate the HPS first-stage boot loader (FSBL) and second stage boot loader (SSBL).

1.1. Glossary

Table 1. Intel Stratix 10 SoC FPGA Boot Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMIF</td>
<td>External memory interface</td>
</tr>
<tr>
<td>*.pof</td>
<td>Programming object file; contains data to configure the FPGA portion of the SoC and additionally, may contain the HPS first-stage payload. This file is typically stored in external flash such as quad serial peripheral interface (Quad SPI) or common flash interface (CFI) flash.</td>
</tr>
<tr>
<td>FW</td>
<td>Firmware; Controls and monitors software stored in Secure Device Manager’s (SDM) read-only memory.</td>
</tr>
<tr>
<td>EPCQ-L</td>
<td>Intel Enhanced Programmable Configuration Quad memory, low voltage. Both this memory and Quad SPI flash can interface to the SDM’s Quad SPI controller.</td>
</tr>
<tr>
<td>HPS</td>
<td>Hard Processor System; the SoC portion of the device, consisting of a quad core Arm® Cortex-A53 processor, hard IPs, and HPS I/Os in the Intel Stratix 10 SoC FPGA.</td>
</tr>
<tr>
<td>HPS EMIF I/O section</td>
<td>Part of the raw binary file (*.rbf) that configures the EMIF I/O used by the HPS</td>
</tr>
<tr>
<td>FPGA I/O section</td>
<td>Part of the *.rbf that configures the I/O assigned to the FPGA core</td>
</tr>
<tr>
<td>Core *.rbf</td>
<td>Core raw binary file; FPGA core image file that includes logic array blocks (LABs), digital signal processing (DSP), and embedded memory. The core image consists of a single reconfigurable region, or both static and reconfigurable regions.</td>
</tr>
<tr>
<td>FSBL</td>
<td>First-stage Boot Loader for HPS</td>
</tr>
<tr>
<td>*.jic</td>
<td>JTAG Indirect Configuration file that allows programming through JTAG</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>*.rbf</td>
<td>Raw binary file representing the FPGA bitstream</td>
</tr>
<tr>
<td>*.rpd</td>
<td>Raw Programming Data file for AS (EPCQ-L) devices</td>
</tr>
</tbody>
</table>

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

- Intel Quartus Prime Pro Edition version 18.0 or later
- Intel SoC FPGA Embedded Development Suite (SoC EDS) version 18.0 or later
  
  *Note:* The versions of Intel Quartus Prime Pro Edition and SoC EDS must match.
- Arm Development Studio 5* Intel SoC FPGA Edition version 5.28 or later

### Related Information

Boot-inge and Configuration Appendix


### 1.3. Intel Stratix 10 SoC FPGA Boot Overview

The Intel Stratix 10 SoC FPGA combines an FPGA with a hard processor system (HPS) that is capable of booting Bare Metal applications or operating systems such as Linux*.

When booting the device from a power-on reset, you can choose between two different methods of booting:

- **FPGA Configuration First Mode**
  - The FPGA and all of the I/Os are fully configured before the HPS is released from reset. Thus, when the HPS boots, the FPGA is in user mode and is ready to interact with the HPS. Optionally, the SDM can hold the HPS in reset until instructed by the user. To release the HPS from reset, you can use soft IP such as a mailbox client to send a mailbox request to the SDM.

- **HPS Boot First Mode**
  - This mode is also referred to as Early I/O Release Mode or Early I/O Configuration. After power-on, the device configures a minimal amount of I/O required by the HPS before releasing the HPS from reset. This mode allows the HPS to boot quickly without having to wait for the full configuration to complete. Subsequently, the HPS may trigger an FPGA configuration request during the SSBL or OS stage.

### Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.sof</td>
<td>SRAM object file which contains the bitstream for the primary FPGA design. Firmware is not part of the *.sof.</td>
</tr>
<tr>
<td>SSBL</td>
<td>Second-stage Boot Loader for HPS</td>
</tr>
<tr>
<td>SDM</td>
<td>Secure Device Manager; a triple-redundant processor-based block that manages FPGA configuration and hard processor system (HPS) secure boot process in Intel Stratix 10 devices.</td>
</tr>
</tbody>
</table>
2. FPGA Configuration First Mode

2.1. Boot Flow Overview for FPGA Configuration First Mode

You can program the Intel Stratix 10 SoC device to configure the FPGA first and then boot the HPS. The available configuration data sources configure the FPGA core and periphery first in this mode. After completion, you may optionally boot the HPS. All of the I/O, including the HPS-allocated I/O, are configured and brought out of tri-state. If the HPS is not booted:

- The HPS is held in reset.
- HPS-dedicated I/O are held in reset.
- HPS-allocated I/O are driven with reset values from the HPS.

If the FPGA is configured before the HPS boots, the boot flow looks like the example figure below. The flow includes the time from power-on-reset ($T_{POR}$) to boot completion ($T_{Boot\_Complete}$).

Figure 1. Typical FPGA Configuration First Boot Flow
Table 2. FPGA Configuration First Stages
The sections following this table describe each stage in more detail.

<table>
<thead>
<tr>
<th>Time</th>
<th>Boot Stage</th>
<th>Device State</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_{POR} to T_1</td>
<td>POR</td>
<td>Power-on reset</td>
</tr>
</tbody>
</table>
| T_1 to T_2    | Secure Device Manager (SDM)-Boot ROM           | 1. SDM samples the MSEL pins to determine the configuration scheme and boot source.  
|               |                                                 | 2. SDM establishes the device security level based on eFuse values.   
|               |                                                 | 3. SDM initializes the device by reading the configuration firmware (initial part of the bitstream) from the boot source.  
|               |                                                 | 4. SDM authenticates and decrypts the configuration firmware (this process occurs as necessary throughout the configuration).  
|               |                                                 | 5. SDM starts executing the configuration firmware.                          |
| T_2 to T_3    | SDM-configuration firmware                     | 1. SDM I/O are enabled.                                                     
|               |                                                 | 2. SDM configures the FPGA I/O and core (full configuration) and enables the rest of your configured SDM I/O.                     
|               |                                                 | 3. SDM loads the FSBL from the bitstream into HPS on-chip RAM.              
|               |                                                 | 4. SDM enables HPS SDRAM I/O and optionally enables HPS debug.              
|               |                                                 | 5. FPGA is in user mode.                                                   
|               |                                                 | 6. HPS is released from reset. CPU1-CPU3 are in a wait-for-interrupt (WFI) state. |
| T_3 to T_4    | First-Stage Boot Loader (FSBL)                 | 1. HPS verifies the FPGA is in user mode.                                    
|               |                                                 | 2. The FSBL initializes the HPS, including the SDRAM.                       
|               |                                                 | 3. HPS loads SSBL into SDRAM.                                              
|               |                                                 | 4. HPS peripheral I/O pin mux and buffers are configured. Clocks, resets, and bridges are also configured.                     
|               |                                                 | 5. HPS I/O peripherals are available.                                       |
| T_4 to T_5    | Second-Stage Boot Loader (SSBL)                | 1. HPS bootstrap completes.                                                 
|               |                                                 | 2. OS is loaded into SDRAM.                                                |
| T_5 to T_{Boot_Complete} | Operating System (OS) | The OS boots and applications are scheduled for runtime launch.            |

Note: The location of the source files for configuration, FSBL, SSBL, and OS can vary and are described in the System Layout for FPGA Configuration First Mode section.

Related Information
System Layout for FPGA Configuration First Mode on page 10

2.1.1. Power-On Reset (POR)
Ensure you power each of the power rails according to the power sequencing consideration until they reach the required voltage levels. In addition, the power-up sequence must meet either the standard or the fast power-on reset (POR) delay time.
2.1.2. Secure Device Manager

Once the Intel Stratix 10 SoC FPGA exits POR, the SDM samples the MSEL[2:0] pins\(^{(1)}\) to determine the boot source. Next, the device configures the SDM I/Os according to the selected boot source interface and the SDM retrieves the configuration bitstream through the interface. SDM can boot from the following boot sources listed in the following table.

Table 3. Available SDM Boot Sources for the Intel Stratix 10 SoC FPGA

<table>
<thead>
<tr>
<th>SDM Boot Source</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon-ST (x8/x16/x32)</td>
<td>Supported</td>
</tr>
<tr>
<td>JTAG</td>
<td>Supported</td>
</tr>
<tr>
<td>Active Serial (AS)/ Quad SPI</td>
<td>Supported. SDM only boots in x4 mode for active serial flash and Micron* MT25Q flash. Other serial flash discoverable parameters (SDFP) JEDEC-compliant flash boot in x1 mode. After the configuration firmware loads into the SDM, the SDM can switch the flash into x4 mode.</td>
</tr>
<tr>
<td>SD/MMC x4/x8</td>
<td>Supported in a future version of Intel Quartus Prime Pro Edition.</td>
</tr>
</tbody>
</table>

The typical configuration bitstream for FPGA configuration first contains:
1. Configuration firmware for the SDM
2. FPGA I/O and HPS external memory interface (EMIF) I/O configuration data
3. FPGA core configuration data
4. HPS FSBL code and FSBL hardware handoff binary data

The SDM completes the configuration of the FPGA core and I/O, and then copies the HPS FSBL code and HPS FSBL hardware handoff binary to the HPS on-chip RAM.

Related Information
Intel Stratix 10 Configuration User Guide
For more information about the SDM

2.1.3. First-Stage Boot Loader

The first-stage boot loader (FSBL) is the first boot stage for the HPS. In FPGA Configuration First mode, the SDM extracts and loads the FSBL into the on-chip RAM of the HPS. The SDM releases the HPS from reset after the FPGA has entered user mode.

---

\(^{(1)}\) The MSEL[2:0] pins are multiplexed with the SDM_IO[9], SDM_IO[7], SDM_IO[5] pins respectively.
mode. After the HPS exits reset, it uses the FSBL hardware handoff file to setup the clocks, HPS dedicated I/Os, and peripherals. Typically, the FSBL then loads the SSBL into HPS SDRAM and passes the control to the SSBL.

Note: The SDM can optionally hold the HPS in reset until instructed by the user. Refer to the Compiling the SRAM Object File section to enable this setting.

You can create the FSBL from one of the following sources:
- U-Boot secondary program loader (SPL)
  - Intel provides a pre-built U-Boot FSBL *.ihex as well as source code as part of the SoC EDS package.
- Arm Trusted Firmware
  - Intel provides the source code for the Arm Trusted Firmware on public git.
- Real-time operating system (RTOS)
- Bare Metal application

The steps to generate, update, and integrate the HPS FSBL into the configuration bitstream are discussed in the Bitstream Creation and Programming Overview section.

Related Information
- Compiling the SRAM Object File on page 26
- Bitstream Creation and Programming Overview on page 25
- Arm Trusted Firmware Source Code on git

2.1.4. Second-Stage Boot Loader

The second-stage boot loader (SSBL) is the second boot stage for the HPS. The FSBL initiates the copy of the SSBL to the HPS SDRAM. The SSBL typically enables more advanced peripherals such as Ethernet and supports command line interface.

You can create the SSBL from one of the following sources:
- U-Boot
  - Intel provides a pre-built U-Boot SSBL binary as well as source code as part of the SoC EDS package.
- UEFI
  - Intel provides the source code for UEFI SSBL on public git.
- RTOS
- Bare Metal application

Intel provides the U-boot source code for the Intel Stratix 10 HPS SSBL as part of the SoC EDS. The latest source code is also available on the Intel public git repository.

Related Information
- UEFI Source Code on git
2.1.5. Operating System

Typically, the SSBL loads the operating system (OS) stage into SDRAM. The OS executes from SDRAM. Depending on your application requirements, you may implement a conventional OS or an RTOS.

Intel provides an example of the Intel Stratix 10 SoC Linux kernel and Ångström recipe that is available on the Intel public git repository. Refer to the Generating the Linux Kernel Image section for more information.

Related Information
Generating the Linux Kernel Image on page 37

2.1.6. Application

The application that runs on the OS is the last boot stage. The application can also replace the OS stage as a dedicated, Bare Metal runtime code.

2.2. System Layout for FPGA Configuration First Mode

The following sections describe the supported system layout for FPGA Configuration First mode. The OS is assumed to be Linux in the following examples, but you may replace Linux with other supported operating systems.

2.2.1. External Configuration Host Only

Figure 2. External Configuration Host Only
In this example, the external configuration host (AvST or JTAG) provides the SDM with a configuration bitstream that consist of:

- SDM configuration firmware
- FPGA I/O and HPS EMIF I/O configuration data
- FPGA core configuration data
- HPS FSBL code and HPS FSBL hardware handoff binary

Because the HPS SSBL (or subsequent OS files) is not part of the bitstream, the HPS can only boot up to the FSBL stage. This setup is applicable if you are using the FSBL to run simple applications (for example, Bare Metal applications).

You can use the FSBL to retrieve the SSBL from other sources, such as through the HPS Ethernet MAC interface. To implement these modes of access, you must create a working Ethernet software stack in the FSBL.

<table>
<thead>
<tr>
<th>SDM Configuration Host</th>
<th>SSBL Source</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvST</td>
<td>HPS Ethernet</td>
<td>Not supported in U-Boot FSBL code provided by Intel.</td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.2.2. External Configuration Host with HPS Flash

An external configuration host with HPS flash provides an SDM configuration bitstream containing:

- SDM configuration firmware
- FPGA I/O and HPS EMIF I/O configuration data
- FPGA core configuration data
- HPS FSBL code and HPS FSBL hardware handoff binary
In this system layout, you can use the HPS flash to store the HPS SSBL, Linux image device tree information and OS file system. This layout enables the device to boot into an OS such as Linux.

**Table 5. Supported Configuration Boot Source and HPS Flash Combinations**

<table>
<thead>
<tr>
<th>SDM Configuration Host</th>
<th>HPS Flash</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvST</td>
<td>SD/MMC</td>
<td>Pre-built HPS SD image (for booting the SSBL and OS) and *.sof (for JTAG Programming) are included in the SoC EDS.</td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AvST</td>
<td>NAND</td>
<td>Supported in future release of the SoC EDS.</td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**2.2.3. Single SDM Flash**

In a single flash attached to SDM layout, the flash contains all of the files required for booting, including the configuration bitstream and OS files.

**Table 6. Single SDM Flash Sources**

<table>
<thead>
<tr>
<th>SDM Flash Type</th>
<th>Details</th>
</tr>
</thead>
</table>
| Active serial/Quad SPI | Refer to the Intel Stratix 10 SoC Single QSPI Flash Boot example on RocketBoards.  
*Note: When you use the HPS to access the SDM Quad SPI, the speed access is reduced due to the additional delays between the path from the HPS to SDM.* |
| SD/MMC               | Supported in a future version of the SoC EDS. |

Software running on the HPS (for example, FSBL) must request permission from the SDM to access the flash attached to the SDM.

In the SD/MMC example below, the SSBL, OS, and OS file system reside in the FAT and EXT4 partitions. In the Quad SPI flash example, the SSBL, OS and file system reside in the Unsorted Block Image File System (UBIFS).
Figure 4. FPGA Configuration First Layout with SD/MMC

Figure 5. FPGA Configuration First Layout with Quad SPI
2.2.4. FPGA Configuration First Dual Flash System

In a dual flash system, the SDM flash (for example, active serial flash) stores the configuration bitstream, while the HPS flash (for example, an SD card) stores the HPS SSBL and the rest of the OS files.

<table>
<thead>
<tr>
<th>SDM Flash Type</th>
<th>HPS Flash Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active serial/Quad SPI</td>
<td>SD/MMC</td>
<td>Refer to the Intel Stratix 10 GSRD on RocketBoards website for implementation support.</td>
</tr>
<tr>
<td>Active serial/Quad SPI</td>
<td>NAND</td>
<td>Supported in a future version of the</td>
</tr>
<tr>
<td>SD/MMC</td>
<td>SD/MMC or NAND</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. FPGA Configuration First Dual SDM and HPS Flash

Related Information
RocketBoards: Intel Stratix 10 GSRD
3. HPS Boot First Mode

3.1. Boot Flow Overview

You can boot the HPS and HPS EMIF I/O first before configuring the FPGA core and periphery. The $\text{MSEL}[2:0]$ settings determine the source for booting the HPS. In this mode, any of the I/O allocated to the FPGA remain tri-stated while the HPS is booting. The HPS can subsequently request the SDM to configure the FPGA core and periphery, excluding the HPS EMIF I/O. Software determines the configuration source for the FPGA core and periphery. In HPS boot first mode, you have the option of configuring the FPGA core during the SSBL stage or after the OS boots.

*Note:* The initial configuration of HPS EMIF I/O and loading of HPS FSBL is called "Phase 1 configuration". The subsequent configuration of FPGA core and periphery by HPS is called "Phase 2 configuration".

A typical HPS Boot First flow may look like the following figure. You can use U-Boot, Unified Extensible Firmware Interface (UEFI), or a custom boot loader for your FSBL or SSBL. An example of an OS is Linux or an RTOS. The flow includes the time from power-on-reset ($T_{POR}$) to boot completion ($T_{Boot\_Complete}$).

**Figure 7. Typical HPS Boot First Flow**
### Table 8. HPS Boot First Stages

<table>
<thead>
<tr>
<th>Time</th>
<th>Boot Stage</th>
<th>Device State</th>
</tr>
</thead>
<tbody>
<tr>
<td>T&lt;sub&gt;POR&lt;/sub&gt;</td>
<td>POR</td>
<td>Power-on reset</td>
</tr>
</tbody>
</table>
| T<sub>1</sub> to T<sub>2</sub> | SDM- Boot ROM      | 1. SDM samples the MSEL pins to determine the configuration and boot source. It also establishes the device security level based on eFuse values.  
2. SDM firmware initializes the device.  
3. SDM authenticates and decrypts the bitstream (this process occurs as necessary throughout the configuration). |
| T<sub>2</sub> to T<sub>3</sub> | SDM- Configuration Firmware | 1. SDM configures the HPS EMIF I/O and the rest of the user-configured SDM I/O.  
2. SDM loads the FSBL from the bitstream into HPS on-chip RAM.  
3. SDM enables HPS SDRAM I/O and optionally enables HPS debug.  
4. HPS is released from reset. |
| T<sub>3</sub> to T<sub>4</sub> | FSBL               | 1. The FSBL initializes the HPS, including the SDRAM.  
2. FSBL obtains the SSBL from HPS flash or by requesting flash access from the SDM.  
3. FSBL loads the SSBL into SDRAM.  
4. HPS peripheral I/O pin multiplexer and buffers are configured. Clocks, resets and bridges are also configured.  
5. HPS I/O peripherals are available. |
| T<sub>4</sub> to T<sub>5</sub> | SSBL               | HPS bootstrap completes. After bootstrap completes, any of the following steps may occur:  
1. The FPGA core configuration loads into SDRAM from one of the following sources:  
   • SDM flash  
   • HPS alternate flash  
   • EMAC interface  
2. HPS requests SDM to configure the FPGA core. (2)  
3. FPGA enters user mode.  
4. OS is loaded into SDRAM. |
| T<sub>5</sub> to T<sub>Boot_Complete</sub> | OS                 | 1. OS boot occurs and applications are scheduled for runtime launch  
2. (Optional step) OS initiates FPGA configuration through a secure monitor call (SMC) to the resident SMC handler (typically SSBL), which then initiates the request to the SDM. |

**Note:** The location of the source files for configuration, FSBL, SSBL and OS can vary. Refer to the *System Layout for HPS Boot First Mode* section for more information.

**Related Information**

*System Layout for HPS Boot First Mode* on page 19

---

(2) FPGA I/O and FPGA core configuration can occur at the SSBL or OS stage, but is typically configured during the SSBL stage.
3.1.1. Power-On Reset (POR)

Ensure you power each of the power rails according to the power sequencing consideration until they reach the required voltage levels. In addition, the power-up sequence must meet either the standard or the fast power-on reset (POR) delay time.

Related Information
- Intel Stratix 10 Device Datasheet
  For referencing the POR delay specification
- AN 692: Power Sequencing Considerations for Intel Cyclone 10 GX, Intel Arria 10, and Intel Stratix 10 Devices

3.1.2. Secure Device Manager

After the Intel Stratix 10 SoC FPGA exits POR, the SDM samples the MSEL[2:0] pins to determine the boot source. Next, the device configures the SDM I/Os according to the selected boot source interface and the SDM retrieves the configuration bitstream through the interface.

**Table 9. Available SDM Boot Sources for the Intel Stratix 10 SoC FPGA**

<table>
<thead>
<tr>
<th>SDM Boot Source</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon-ST (x8/x16/x32)</td>
<td>Supported</td>
</tr>
<tr>
<td>JTAG</td>
<td>Supported</td>
</tr>
<tr>
<td>Active Serial (AS)/ Quad SPI</td>
<td>Supported. SDM only boots in x4 mode for active serial flash and Micron MT25Q flash. Other serial flash discoverable parameters (SDFP) JEDEC-compliant flash devices boot in x1 mode. Once the device initialization code loads into the SDM, the SDM can switch these flash into x4 mode.</td>
</tr>
<tr>
<td>SD/MMC x4/x8</td>
<td>Supported in a future version of Intel Quartus Prime Pro Edition.</td>
</tr>
</tbody>
</table>

The typical configuration bitstream for HPS boot first mode contains:
1. SDM configuration firmware
2. HPS external memory interface (EMIF) I/O configuration data
3. HPS FSBL code and HPS FSBL hardware handoff binary

The SDM completes the configuration of the HPS EMIF I/O and then copies the HPS FSBL to the HPS on-chip RAM.

Related Information
- Intel Stratix 10 Configuration User Guide
  For more information about the SDM

3.1.3. First-Stage Boot Loader

In HPS Boot First mode, the SDM copies the FSBL code to the HPS on-chip RAM and releases the HPS from reset after the HPS EMIF I/O are configured. This method allows the HPS to boot faster because it does not need to wait for the entire FPGA configuration to complete.
After the SDM releases the HPS from reset, the FSBL initializes the HPS. Initialization includes configuring clocks, HPS dedicated I/Os and peripherals. Finally, the FSBL loads the SSBL into HPS SDRAM and passes the control to the SSBL.

You can create the FSBL from one of the following sources:

- U-Boot secondary program loader (SPL)
  - Intel provides a pre-built U-Boot FSBL *.hex file as well as source code as part of the SoC EDS package.
- Arm Trusted Firmware
  - Intel provides the source code for the Arm Trusted Firmware FSBL on the public git.
- Real-time operating system (RTOS)
- Bare Metal application

The latest source code is also available on the Intel public git repository.

The Bitstream Creation and Programming Overview section describes the steps to generate, update and integrate the HPS FSBL into the configuration bitstream.

**Related Information**
- Bitstream Creation and Programming Overview on page 25
- Arm Trusted Firmware Source Code on git

### 3.1.4. Second-Stage Boot Loader

The second-stage boot loader (SSBL) is the second boot stage for the HPS. The FSBL initiates the copy of the SSBL to the HPS SDRAM. The SSBL typically enables more advance peripherals such as Ethernet and supports command line interface.

You can create the HPS SSBL from one of the following sources:

- U-Boot
  - Intel provides a pre-built U-Boot SSBL binary as well as source code as part of the SoC EDS package.
- UEFI
  - Intel provides the source code for UEFI SSBL on public git.
- RTOS
- Bare Metal application

Intel provides the U-boot source code for the Intel Stratix 10 HPS SSBL as part of the SoC EDS. The latest source code is also available on the Intel public git repository.

You can optionally perform FPGA core and I/O configuration in during the SSBL stage. The SSBL copies the FPGA configuration files from one of the following sources to the HPS SDRAM:

- HPS Flash
- SDM Flash
- External host via the HPS Ethernet (for example, TFTP)
After the SSBL copies the FPGA configuration files to the HPS SDRAM, the SSBL can initiate a configuration request to the SDM to begin the configuration process. Refer to the Configuring the FPGA from SSBL and OS section for more details.

Related Information
- Configuring the FPGA from SSBL and OS on page 39
- UEFI Source Code on git

3.1.5. Operating System

The SSBL loads the operating system (OS) stage into SDRAM. The OS executes from SDRAM. Depending on your application requirements you may implement a conventional OS or an RTOS.

Intel provides an example of the Intel Stratix 10 SoC Linux kernel and Ångström recipe that is available on the Intel public git repository. Refer to the Generating the Linux Kernel Image section for more information.

Related Information
Generating the Linux Kernel Image on page 37

3.1.6. Application

The application that runs on the OS is the last boot stage. The application can also replace the OS stage as a dedicated, Bare Metal runtime code.

3.2. System Layout for HPS Boot First Mode

3.2.1. External Configuration Host Only

Figure 8. External Configuration Host Only
In this example, the external configuration host (AvST or JTAG) provides the SDM with a configuration bitstream that consist of:

- SDM configuration firmware
- HPS EMIF I/O configuration data
- HPS FSBL code and HPS FSBL hardware handoff binary

However, because the HPS SSBL (or subsequent OS files) are not part of the bitstream, the HPS can only boot up to the FSBL stage. This setup is applicable if you are using the FSBL to run simple applications (for example, Bare Metal applications).

Because the FPGA core is not configured, the FSBL must retrieve the SSBL from external sources, for example the HPS EMAC interface. The U-Boot FSBL source code that is generated through the SoC EDS does not include source code to support SSBL retrieval through the HPS EMAC interface. You must implement the Ethernet software stack in the FSBL separately. Similarly, after the SSBL loads, the SSBL must retrieve the FPGA core and I/O configuration file from an external source as well.

### Table 10. Supported Configuration Boot and SSBL Sources

<table>
<thead>
<tr>
<th>SDM Configuration Host</th>
<th>SSBL Source</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvST</td>
<td>HPS Ethernet</td>
<td>Not supported in U-Boot FSBL code provided by Intel.</td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.2.2. External Configuration Host with HPS Flash

**Figure 9. External Configuration Host with HPS Flash**

An external configuration host provides an SDM configuration bitstream containing:
• SDM configuration firmware
• HPS EMIF I/O configuration data
• HPS FSBL code and HPS FSBL hardware handoff binary

In this system layout, the HPS flash (for example, SD card), contains the HPS SSBL, Linux image device tree information and OS file system.

Depending on the boot stage that performs the FPGA configuration, the FPGA core and I/O configuration file can be stored either in the HPS Flash storage partition (SSBL initiates configuration), or integrated as part of the OS file system (OS initiates configuration).

Table 11. Supported Configuration Boot Source and HPS Flash Combinations

<table>
<thead>
<tr>
<th>SDM Configuration Host</th>
<th>HPS Flash</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvST</td>
<td>SD/MMC</td>
<td>Not supported in Intel Stratix 10 GSRD. You must recompile your design with the HPS First option in Intel Quartus Prime Pro Edition and manually create their FPGA configuration file in the HPS SD/MMC.</td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AvST</td>
<td>NAND</td>
<td>Supported in future release of the SoC EDS.</td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2.3. Single SDM Flash

In a single flash attached to SDM layout, the flash contains all of the files required for booting, including the configuration bitstream and the OS files.

Depending on the boot stage that performs the FPGA configuration, the FPGA core and I/O configuration file can be stored either in the SDM flash storage partition (SSBL initiates configuration), or integrated as part of the OS file system (OS initiates configuration).

Table 12. Supported Single SDM Flash Sources

<table>
<thead>
<tr>
<th>SDM Flash Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active serial/Quad SPI</td>
<td>Not supported in Intel Stratix 10 GSRD. You must recompile your design with the HPS First option in Intel Quartus Prime Pro Edition and manually create the flash image for SDM Quad SPI. Refer to the Creating the *.jic File for System Layout with a Single SDM Flash for more information.</td>
</tr>
<tr>
<td>SD/MMC</td>
<td>Supported in a future version of the SoC EDS.</td>
</tr>
</tbody>
</table>

In order for the software running in the HPS (for example, the FSBL and SSBL) to access the flash attached to the SDM, it must request permission from the SDM.

Note: HPS accessing SDM flash is only supported for Quad SPI in Intel Quartus Prime Pro Edition 18.1. Other memories will be supported in future versions of Intel Quartus Prime Pro Edition.

In the SD/MMC example, the SSBL, OS and OS file system reside in the FAT and EXT4 partitions. In the Quad SPI flash example, the SSBL, OS and file system reside in the Unsorted Block Image File System (UBIFS).
Figure 10. HPS Boot First Layout with SD/MMC
### 3.2.4. HPS Boot First Dual Flash System

In a dual flash system, the SDM flash (for example, active serial flash) stores the configuration bitstream, while the HPS flash (for example, an SD card) stores the HPS SSBL and the rest of the OS files.

#### Table 13. Supported Dual SDM and HPS Flash Combinations

<table>
<thead>
<tr>
<th>SDM Flash Type</th>
<th>HPS Flash Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active serial/Quad SPI</td>
<td>SD/MMC</td>
<td>Not supported in Intel Stratix 10 GSRD. You must recompile your design with the HPS First Option in Intel Quartus Prime and manually update the SDM Quad SPI flash image and the HPS SD card.</td>
</tr>
<tr>
<td>Active serial/Quad SPI</td>
<td>NAND</td>
<td>Supported in a future version of the SoC EDS.</td>
</tr>
<tr>
<td>SD/MMC</td>
<td>SD/MMC or NAND</td>
<td></td>
</tr>
</tbody>
</table>
Depending on the boot stage that performs the FPGA configuration, the FPGA core and I/O configuration file can be stored either in the HPS Flash storage partition (SSBL initiate configuration), or integrated as part of the OS file system (OS initiates configuration).

Figure 12.  HPS Boot First Dual SDM and HPS Flash
4. Creating a Configuration Bitstream for Intel Stratix 10 SoC FPGA

4.1. Bitstream Creation and Programming Overview

This section describes how to create the configuration bitstream for Intel Stratix 10 FPGA and the steps to write the resulting files to memory. Steps listed in this section apply to both the FPGA Configuration First mode and HPS Boot First mode unless explicitly mentioned.

Figure 13. Bitstream Creation and Programming Overview

The numbers in the figure correspond to the following sections:
1. Compiling the SRAM Object File on page 26
2. Compiling U-Boot FSBL and SSBL on page 29
3. Creating the *.sof + Handoff and FSBL File on page 31
4. Creating the Raw Binary File (*.rbf) for FPGA Configuration Using HPS on page 32
5. Creating the Raw Programming Data (*.rpd) File For Flash Programming on page 33

Related Information
Intel Stratix 10 SoC Golden Hardware Reference Design
For more information about updating the GHRD to match the device on the Intel Stratix 10 SoC Development Kit.

4.2. Compiling the SRAM Object File

1. Create an Intel Quartus Prime Pro Edition project that uses the HPS component in the Platform Designer tool. If you are using the Intel Stratix 10 SoC development kit, you can navigate to the Intel Stratix 10 SoC Golden Hardware Reference Design (GHRD) project that is preinstalled with the SoC EDS. The default path for the GHRD project is shown below.

   On Windows:
   ```
c:\intelFPGA_pro\18.0\embedded\examples\hardware\s10_soc_devkit_ghrd
   ```
   On Linux:
   ```
~/intelFPGA_pro/18.0/embedded/hardware/s10_soc_devkit_ghrd
   ```

2. Ensure that the SSBL boot source, HPS dedicated I/O and clock settings are set correctly in the HPS component within the Platform Designer tool. These settings are part of the HPS handoff data that is consumed by the HPS FSBL.

   a. Set the HPS boot source to indicate the location of the HPS SSBL.

   Note: The current version of the Platform Designer only supports the Use HPS SDMMC flash option in the FSBL.

   b. Configure the HPS clock settings in the HPS Clocks and resets tab.

   ![Figure 14. Selecting HPS Boot Source]

   ![Image of Selecting HPS Boot Source]
Configure HPS Clock Settings

Figure 15.

- Update the pin multiplexing configuration in the Pin Mux and Peripherals tab.

Configure Pin Multiplexing

Figure 16.

3. Save your settings and run Generate HDL... in Platform Designer to complete the HPS component instantiation process before returning to the main Intel Quartus Prime Pro Edition.
4. Open the project in Intel Quartus Prime Pro Edition and go to Assignments ➤ Device and Pin Options.

5. Specify the boot mode through the HPS/FPGA configuration order pulldown in the Configuration project category.
   - **After INIT_DONE**: This option selects FPGA Configuration First. The SDM configures the FPGA core and all the periphery I/O before loading the FSBL into the HPS on-chip RAM and releasing the HPS from reset. If any errors exist during initial configuration, the HPS is not released from reset.
   - **HPS First**: This option selects HPS Boot First. The SDM only configures the I/O required for the HPS SDRAM, and then loads the FSBL into the HPS on-chip RAM before releasing the HPS from reset. The FPGA core and the other unused I/O, remain unconfigured.
   - **When requested by FPGA**: This option is similar to FPGA Configuration First, except that the SDM does not release the HPS from reset. You can implement an IP in the FPGA, or use the mailbox client IP, in order to initiate the request to SDM to release the HPS from reset.

When you select **HPS First** or **After INIT_DONE**, you must ensure that the Use configuration device option is unchecked because you must include the FSBL into the bitstream before the completed bitstream is generated.

**Figure 17. Configuration Scheme and Boot Mode Selection**

6. After you complete your design, run the Start Compilation option. After the compilation is complete, the resultant output files, including the *.sof, are stored in the `<Project_Directory>/output_files` folder.

   **Note**: If you are using the GHRD project, the pre-built *.sof file found in the output_files sub-directory is named `ghrd_1sx280lu2f50e2vg.sof`. 
4.3. Compiling the First Stage Boot Loader (FSBL) and the Second Stage Boot Loader (SSBL)

4.3.1. Compiling U-Boot FSBL and SSBL

For the latest U-Boot FSBL and SSBL compilation information, always refer to the Rocketboards.org website.

SoC EDS includes prebuilt U-Boot FSBL and SSBL files located in the following directory:

<table>
<thead>
<tr>
<th>Location of Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;GHRD Project_Directory&gt;/software/u-boot/</td>
<td>Prebuilt U-Boot SSBL files in *.elf, *.dtb, *.bin and *.img format</td>
</tr>
</tbody>
</table>

You can only compile the Intel Stratix 10 SoC FPGA U-Boot in Linux. To begin, follow the steps below.

**Note:** These steps are tested in Ubuntu 16.04 LTS and may not be similar when used in other Linux distributions.

1. Launch the SoC EDS command shell.

   `<SoC_FPGA_EDS installation directory>/embedded_command_shell.sh`

2. Download and extract the Linaro* toolchain and configure the environment.

   ```
   $ tar xf gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar.xz
   $ export PATH=$PWD/gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu/bin:/$PATH
   $ export CROSS_COMPILE=aarch64-linux-gnu-
   $ export ARCH=arm64
   ```

3. Obtain the U-Boot source code from one of the following methods.

   - SoC EDS source archive.
     
     Copy and extract `<SoC_EDS_installation_path>/embedded_host_tools/altera/boot_loaders/stratix10/u-boot/uboot-socfpga.tar.gz` in your working directory.
     
     ```
     $ tar xzf uboot-socfpga.tar.gz
     ```

   - Clone git using the script included in the SoC EDS.
This method ensures that you have the same code base as the one used to generate the prebuilt U-Boot FSBL and SSBL. Copy and run the `<SoC_EDS_installation_path>\embedded\host_tools\altera\boot_loaders\stratix10\u-boot\git_clone.sh` in your working directory.

```
$ ./git_clone.sh
```

— Clone git using Intel public git repository.

Use this method if you require the latest U-Boot code that has been upstreamed to the git.

_Note:_ This version may include newer functionality but may not be fully compatible with the rest of the components.

```
$ git clone https://github.com/altera-opensource/u-boot-socfpga
$ cd u-boot-socfpga
$ git checkout socfpga_v2017.09
$ cd..
```

4. If needed, update the settings for console, environment and configurations within U-Boot by editing the configuration header file at `include/configs/stratix10_socdk.h`. Save the file after you complete your edits.

5. Build U-Boot. In the `u-boot-socfpga` folder, run make:

```
$ cd u-boot-socfpga
$ make mrproper
$ make socfpga_stratix10_defconfig
$ make
```


<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-boot.elf</td>
<td>U-Boot SSBL ELF file</td>
</tr>
<tr>
<td>u-boot-dtb.bin</td>
<td>U-Boot SSBL binary file</td>
</tr>
<tr>
<td>u-boot-dtb.img</td>
<td>U-Boot SSBL image file</td>
</tr>
<tr>
<td>spl\u-boot-spl-dtb.bin</td>
<td>U-Boot FSBL binary file</td>
</tr>
</tbody>
</table>

7. Create the U-Boot FSBL *.ihex file.

```
$ aarch64-linux-gnu-objcopy -I binary -O ihex --change-addresses 0xfee00000 spl/u-boot-spl-dtb.bin spl/u-boot-spl-dtb.ihex
```

### 4.3.2. Compiling the Arm Trusted Firmware FSBL and UEFI SSBL

An alternative to U-boot is to use the Arm Trusted Firmware for FSBL, followed by UEFI as the SSBL. Intel provides the source code for both the Arm Trusted Firmware and UEFI on the public git repository. Follow the steps listed in the _Intel Stratix 10 SoC UEFI Boot Loader User Guide_ document in order to generate the FSBL and SSBL files required for your system.

**Related Information**
- [Arm Trusted Firmware Source Code on git](https://github.com/altera-opensource/arm-trusted-firmware)
4.4. Creating the *.sof + Handoff and FSBL File

After you have both the *.sof file from the Intel Quartus Prime Pro Edition compilation, and the FSBL *.hex file, you can use the Intel Quartus Prime Pro Edition Convert Programming File tool (quartus_cpf) to merge these two together. The tool creates a single *.sof file that has both the hardware design information, as well as the FSBL content.

The bitstream section that contains the HPS FSBL full image is shown below.

**Figure 18.** HPS FSBL Full Image Components

During full compilation, the Intel Quartus Prime Pro Edition generates the data in the HPS FSBL handoff binary based on the settings inside the Platform Designer. The file contains information on:

- HPS pin multiplexing configuration, including HPS dedicated I/O settings and export to FPGA settings
- HPS clock configuration
- HPS SSBL boot source

During configuration, the SDM copies the entire HPS FSBL image into the HPS on-chip memory. When the HPS FSBL executes, it uses the information on the HPS FSBL handoff binary to set the hardware accordingly.

1. To merge the *.sof file and the HPS FSBL *.hex file, type the following quartus_cpf command:

```
$ quartus_cpf --bootloader=<hex_file> <input_sof> <output_sof>
```

Where the `<hex_file>` is the FSBL *.hex file, the `<input_sof>` is the existing *.sof file to insert, and the `<output_sof>` is the final *.sof containing the first-stage boot image. If you compile a new FSBL *.hex, you can re-run the command above to overwrite the existing FSBL image in the output *.sof file.

The Intel Quartus Prime Pro Edition can also use this merged file to program the Intel Stratix 10 SoC FPGA using JTAG. Using JTAG enables you to test your design quickly without having to write to the SDM Flash first. After programming is successful without any error, the HPS attempts to run the FSBL code that is embedded in the bitstream.
Note: If you are using the GHRD, the prebuilt *.sof + Handoff and FSBL file is ghrd_1sx2801u2f50e2vg_hps.sof.

4.5. Creating the Raw Binary File (*.rbf) for FPGA Configuration Using HPS

You can use the *.rbf with a third-party programmer, Configuration via Protocol (CvP) or HPS to initiate the FPGA programming.

For example, in HPS Boot First mode, you can store the *.rbf containing the FPGA core and I/O configuration file in an external storage such as a TFTP server or a USB device. During boot, the HPS retrieves this file and initiates the FPGA configuration during the booting process.

To create the *.rbf file, type the Intel Quartus Prime Pro Edition Convert Programmer tool command as shown below:

```bash
$ quartus_cpf -c -o bitstream_compression=on <input_sof> <output_rbf>
```

If you are using the GHRD output ghrd_1sx2801u2f50e2vg_hps.sof, the command is:

```bash
$ quartus_cpf -c -o bitstream_compression=on ghrd_1sx2801u2f50e2vg_hps.sof ghrd_1sx2801u2f50e2vg_hps.rbf
```

In the previous section, if you create the *.sof + Handoff and FSBL file for HPS Boot First mode, you must add the --hps option as shown below:

```bash
$ quartus_cpf -c --hps -o bitstream_compression=on <input_sof> <output_rbf>
```

Adding the --hps option generates two resultant *.rbf.

Table 15. Generated *.rbf Using --hps Option

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ghrd_1sx2801u2f50e2vg_hps.core.rbf</td>
<td>*.rbf containing the FPGA core and I/O configuration file. Depending on your system layout, you may store this file in the SDM Flash, HPS Flash, or external storage such as a TFTP server.</td>
</tr>
<tr>
<td>ghrd_1sx2801u2f50e2vg_hps.hps.rbf</td>
<td>*.rbf containing the initial bitstream to set the HPS EMIF I/O and HPS FSBL. The external configuration host can perform the initial configuration of the device using this file.</td>
</tr>
</tbody>
</table>

Refer to the Configuring the FPGA from SSBL and OS section for more information on using the HPS to initiate FPGA configuration.

Related Information

Configuring the FPGA from SSBL and OS on page 39
4.6. Creating the Raw Programming Data (*.rpd) File For Flash Programming

You can use the quartus_cpf tool to generate a raw programming data file (*.rpd) that can be used for flash programming with a third-party programming tool. Alternatively, you can use the U-Boot software running in the HPS to program the SDM flash device.

You can create the *.rpd file during *.jic file creation or directly from the *.sof file.

**Note:** If you are using the *.rpd in the GHRD, the pre-built *.rpd file name is ghrd_1sx280lu2f50e2vg_hps_auto.rpd.

**Generating the *.rpd File During *.jic Creation**

You can automatically generate the *.rpd file while creating a *.jic file by adding the auto_create_rpd=on command line option.

```
$ quartus_cpf -c -d EPCQL1024 -s 1SX280LU2 -o auto_create_rpd=on -o rpd_little_endian=off ghrd_1sx280lu2f50e2vg_hps.sof
ghrd_1sx280lu2f50e2vg_hps.jic
```

If you are using the GUI option in Intel Quartus Prime Pro Edition to generate the *.rpd file, you can set the bit-level endianness of the *.rpd file from the Options/Boot Info... selection.

In the example above, the resultant *.rpd file is named ghrd_1sx280lu2f50e2vg_hps_auto.rpd. This file contains only the relevant content from the design, and does not necessarily have a file size equal to the size of the flash device.

**Generating the *.rpd File from the *.sof File**

You can create an *.rpd file from the *.pof file that is generated from the *.sof file.

1. Create a *.pof file by typing the command below:

```
$$ quartus_cpf -c -d EPCQL1024 ghrd_1sx280lu2f50e2vg_hps.sof
ghrd_1sx280lu2f50e2vg_hps.pof
```

2. Create the *.rpd file from the *.pof file by typing the commands below:

```
$$ quartus_cpf -c -o rpd_little_endian=off ghrd_1sx280lu2f50e2vg_hps.pof
ghrd_1sx280lu2f50e2vg_hps.rpd
```

This *.rpd creation results in a large programming image that contains data for the entire Quad SPI device. This image is 128 MB in the example provided. The required data for this example is less than 8 MB.
The *.sof data is placed at the beginning of the *.rpd file and is padded with 0xFF's to the end of the image. You can write a script to strip the unnecessary trailing 0xFF's or you can use the Linux dd command to grab enough of the *.rpd file to capture the required data as shown in the example below.

Info: Command: quartus_cpf -c ghrd_1sx2801u2f50e2vg_hps.pof 
ghrd_1sx2801u2f50e2vg_hps.rpd
Info: Quartus Prime Convert_programming_file was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 420 megabytes
Info: Processing ended: Wed Sep 27 13:10:00 2017
Info: Elapsed time: 00:00:11
Info: Total CPU time (on all processors): 00:00:11

$$ ls -l
total 266940
-rw-rw-r--. 1 XXXXXXX XXXXXXX 134217970 Sep 27 13:08 ghrd_1sx2801u2f50e2vg_hps.pof
-rw-rw-r--. 1 XXXXXXX XXXXXXX 134217728 Sep 27 13:10 ghrd_1sx2801u2f50e2vg_hps.rpd
-rw-r--r--. 1 XXXXXXX XXXXXXX 4900067 Sep 27 12:42 ghrd_1sx2801u2f50e2vg_hps.sof

$$ hexdump -s 0x793898 ghrd_1sx2801u2f50e2vg_hps.rpd
0793898 431b 5895 357e 8632 702a 24fd 6156 d0f8
07938a8 d6d8 aa44 f1c3 b4a9 5381 e887 b3f2 c60f
07938b8 c786 65b3 1f56 4d8d 8a66 3f9c 954c 7e38
07938c8 2b98 fc10 0000 0000 0000 0000 0000 0000
07938d8 0000 0000 0000 0000 0000 0000 0000 0000
*0793ff8 0000 0000 0000 0000 ffff ffff ffff ffff
0794008 ffff ffff ffff ffff ffff ffff ffff ffff

// NOTE: The start of the PAD is design specific
7ffff8 ffff ffff ffff ffff
8000000

$$ dd bs=1M count=8 if=ghrd_1sx2801u2f50e2vg_hps.rpd of=ghrd_1sx2801u2f50e2vg_hps_shorter.rpd
8+0 records in
8+0 records out
8388608 bytes (8.4 MB) copied, 0.00619553 s, 1.4 GB/s

$$ hexdump -s 0x793898 ghrd_1sx2801u2f50e2vg_hps_shorter.rpd
0793898 431b 5895 357e 8632 702a 24fd 6156 d0f8
07938a8 d6d8 aa44 f1c3 b4a9 5381 e887 b3f2 c60f
07938b8 c786 65b3 1f56 4d8d 8a66 3f9c 954c 7e38
07938c8 2b98 fc10 0000 0000 0000 0000 0000 0000
07938d8 0000 0000 0000 0000 0000 0000 0000 0000
*0793ff8 0000 0000 0000 0000 ffff ffff ffff ffff
0794008 ffff ffff ffff ffff ffff ffff ffff ffff

* // NOTE: Much, much less 0xFFFF fill
7ffff8 ffff ffff ffff ffff
08000000

$$ ls -l
total 275140
-rw-rw-r--. 1 XXXXXXX XXXXXXX 134217970 Sep 27 13:08 ghrd_1sx2801u2f50e2vg_hps.pof
-rw-rw-r--. 1 XXXXXXX XXXXXXX 134217728 Sep 27 13:10 ghrd_1sx2801u2f50e2vg_hps.rpd
-rw-r--r--. 1 XXXXXXX XXXXXXX 8388608 Sep 27 13:15 ghrd_1sx2801u2f50e2vg_hps_shorter.rpd
-rw-r--r--. 1 XXXXXXX XXXXXXX 4900067 Sep 27 12:42 ghrd_1sx2801u2f50e2vg_hps.sof
4.7. Creating the JTAG Indirect Configuration (*.jic) File for Flash Programming

The Intel Quartus Prime Pro Edition Programmer tool uses the JTAG Indirect Configuration file (*.jic) to program Active Serial (AS) flash such as the EPCQ-L flash, and a number of third-party flash such as Micron MT25Q.

To generate the *.jic file, use the following command:

```
$ quartus_cpf -c -s <flash loader device> -d <flash device> <input sof file> <output jic file>
```

Note: If you are using the Intel Quartus Prime Pro Edition version 18.0 or earlier, refer to this knowledge database entry to enable support for the Micron MT25 flash.

You can also use the GUI option in the Intel Quartus Prime Pro Edition to perform this step. Go to File ➤ Convert Programming File... Select your *.sof file, flash device, and flash loader device from the options listed in the Convert Programming File window. If you decide to autogenerate the *.rpdm file, clicking the Options/Boot info.. opens a separate window that allows you to edit the bit-level endianness of the *.rpdm file.

Figure 19. Convert Programming File Window

After you program the AS or Quad SPI device with the *.jic file, the flash contains the following sections depending on the boot mode:
• FPGA Configuration First
  — SDM Configuration Firmware
  — FPGA I/O and HPS external memory interface (EMIF) I/O configuration data
  — FPGA core configuration data
  — HPS FSBL code and FSBL hardware handoff binary

• HPS Boot First
  — SDM Configuration Firmware
  — HPS external memory interface (EMIF) I/O configuration data
  — HPS FSBL code and HPS FSBL hardware handoff binary

*Note:* If you are using the GHRD, the pre-built *.jic* file for EPCQ-L1024 is named ghrd_1sx280lu2f50e2vg_hps.jic. Refer to Rocketboards.org for a step-by-step process for programming the SDM Quad SPI flash using the *.jic* file.


For system layout that uses a single SDM Flash, you have the option to include OS files (SSBL, Linux kernel image and OS file system) to the *.jic* file before you program the active serial or Quad SPI flash device. Refer to the *Intel Stratix 10 SoC Single QSPI Flash Boot* example on RocketBoards for step-by-step guide to enable Single QSPI Flash Boot.

**Related Information**

RocketBoards: Intel Stratix 10 SoC Single QSPI Flash Boot Example
5. Generating the Linux Kernel Image

The Intel Stratix 10 SoC FPGA Golden System Reference Design (GSRD) provides the Ångström meta layer for building the U-Boot, Linux kernel, and root file system. This design provides you a working example for booting the system into Linux. Refer to the Rocketboards.org website for more information about using the Ångström build recipes. Refer to the RocketBoards website for latest Linux Kernel branch information.

Note: If you are using the Intel Stratix 10 SoC FPGA Development Kit, the SoC EDS has a pre-built SD Card image that contains the complete U-Boot SSBL, kernel image, device tree, and Linux file system. You can obtain the SD card image from the following location: <SoC_EDS_installation path>\embedded\embeddedsw\socfpga\prebuilt_images\Stratix10_Linux_SDCard.tar.gz.

5.1. Compiling the Linux Kernel Image

You can retrieve the Linux kernel sources from the Intel public git repository to build the kernel image separately. Assuming that you have set up the tool chain as described in the Compiling U-Boot FSBL and SSBL section, type the following commands to build the Linux kernel.

```
$ cd ~
$ git clone https://github.com/altera-opensource/linux-socfpga
$ cd linux-socfpga
$ git checkout socfpga-4.9.78-ltsi // Refer to the RocketBoards website for latest Linux Kernel branch information.
$ make s10_devkit_defconfig
$ make
```

You may make changes to the default configuration (for example, adding GPIO drivers) by typing make menuconfig after the make s10_devkit_defconfig command.

```
$ make s10_devkit_defconfig
$ make menuconfig
```

Make your desired changes and remember to save the *.config file before proceeding to the final make command.
Completing these commands creates the following files:

Table 16. Resultant Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmlinux</td>
<td>Linux kernel ELF file</td>
</tr>
<tr>
<td>arch/ARM64/boot/Image</td>
<td>Linux kernel image</td>
</tr>
<tr>
<td>arch/ARM64/boot/dts/altera/socfpga_stratix10_socdk.dtb</td>
<td>Device tree blob for Linux</td>
</tr>
</tbody>
</table>

Note: If you are using the Intel Stratix 10 SoC development kit, the default location of the kernel image (file name: Image) and the device tree blob (filename: socfpga_stratix10_socdk.dtb) is located in the FAT partition of the HPS SD Card.
6. Configuring the FPGA from SSBL and OS

In the HPS Boot First mode, you can use either the U-Boot SSBL or OS to initiate the FPGA configuration.

6.1. Configuring the FPGA Using the U-Boot SSBL

1. Boot the device until U-Boot SSBL stage.
2. Load the FPGA Configuration *.rbf from either SDM flash, HPS flash, or through HPS Ethernet to HPS SDRAM. By using a simple command, fpga load, the HPS requests the SDM to begin the FPGA configuration using the *.rbf file that has been loaded to the HPS SDRAM.

Below is the example of the serial output when using the SSBL to initiate the FPGA configuration. The FPGA configuration file is located in the HPS SD Card FAT partition.

```
SOCFPGA_STRATIX10 # fatls mmc 0:1
12827136 image
18490 socfpga_stratix10_socdk.dtb
398447 u-boot-dtb.img
6979584 ghrd_1sx280lu2f50e2vg_hps.core.rbf
4 file(s), 0 dir(s)
SOCFPGA_STRATIX10 # fatload mmc 0:1 0x8000 ghrd_1sx280lu2f50e2vg_hps.core.rbf
reading ghrd_1sx280lu2f50e2vg_hps.core.rbf
6979584 bytes read in 477 ms (14 MiB/s)
SOCFPGA_STRATIX10 # fpga load 0x8000 $filesize
.......
FPGA reconfiguration OK!
```

The following code is a similar example, except that the FPGA configuration file is being transferred through the TFTP server. The dcache flush command is required after the transfer from TFTP server to ensure content is loaded onto the HPS SDRAM.

```
SOCFPGA_STRATIX10 # dhcp 0x8000 /xxx/ghrd_1sx280lu2f50e2vg_hps.core.rbf
Speed: 100, full duplex
DHCP client bound to address xxx.xxx.xxx.xxx (1004 ms)
Using ethernet@ff800000 device
TFTP from server xxx.xxx.xxx.xxx; our IP address is xxx.xxx.xxx.xxx
Filename '/xxx/ghrd_1sx280lu2f50e2vg_hps.core.rbf'.
Load address: 0x8000
Loading: #################################################################
#################################################################
#################################################################
#################################################################
#################################################################
#################################################################
#################################################################
#              608.4 Kib/s
done
Bytes transferred = 6979584 (6a8000 hex)
```
6.2. Configuring the FPGA Using the Linux Operating System

**Note:** This feature is supported with Linux kernel v4.9 LTSI and onwards. Refer to RocketBoards.org and the Intel public git repository for the latest information regarding this feature.

The Linux kernel for Intel Stratix 10 SoC FPGA allows you to enable the programming of FPGA from within the OS.

**Testing FPGA Reconfiguration at Kernel Level**

If you want to test the FPGA reconfiguration at kernel level, make the following changes to the kernel source code:

1. In the file `/arch/arm64/boot/dts/altera/Makefile`, add a second `.dtb` file. For example:
   ```
   dtb-$(CONFIG_ARCH_STRATIX10) += socfpga_stratix10_socdk.dtb
   socfpga_stratix10_ovl1.dtb
   ```

2. Create a new `.dts` file (for example: `socfpga_stratix10_ovl1.dts`) and add the overlay information of the RBF file into the file as shown below:
   ```
   /dts-v1/;
   /plugin/;
   /
   | fragment80 {
   |     target-path = "/soc/base_fpga_region";
   |     #address-cells = <1>;
   |     #size-cells = <1>;
   |     __overlay__ {
   |         #address-cells = <1>;
   |         #size-cells = <1>;
   |         firmware-name = "soc_s10_ovl1.rbf";
   |         config-complete-timeout-us = <30000000>;
   |     };
   | }
   ```

When you build the Linux kernel for this feature, the build generates two `*.dtb` files.

**Table 17. Resultant Files**

<table>
<thead>
<tr>
<th>Device Tree File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socfpga_stratix10_socdk.dtb</td>
<td>The default <code>*.dtb</code> file used with the kernel image to boot the system.</td>
</tr>
<tr>
<td>socfpga_stratix10_ovl1.dtb</td>
<td>The <code>*.dtb</code> file used to trigger FPGA configuration in OS.</td>
</tr>
</tbody>
</table>
In your compilation output folder, rename the FPGA configuration file (*.rbf) to the following name: soc_s10_ovl1.rbf. Then, copy both the FPGA configuration file (*.rbf) and the socfpga_stratix10_ovl1.dtb file to the following location in your Root File System:

```
$ mkdir <your_ROOTFS>/lib/firmware
$ cp socfpga_stratix10_ovl1.dtb <your_ROOTFS>/lib/firmware/
$ cp soc_s10_ovl1.rbf <your_ROOTFS>/lib/firmware/
```

The changes above allow you to program the FPGA in Linux by applying an overlay on the system. After you boot to Linux and log in with root privilege, use the following command to begin FPGA configuration:

```
# mkdir /sys/kernel/config/device-tree/overlays/0
# echo socfpga_stratix10_ovl1.dtb >
/sys/kernel/config/device-tree/overlays/0/path/
```

If you want to re-apply the overlay, you have to first remove the existing overlay, and then re-run the previous steps:

```
# rmdir /sys/kernel/config/device-tree/overlays/0
# mkdir /sys/kernel/config/device-tree/overlays/0
# echo socfpga_stratix10_ovl1.dtb >/sys/kernel/config/device-tree/overlays/0/
path
```
7. Debugging the Intel Stratix 10 SoC FPGA Boot Flow

To debug the Intel Stratix 10 SoC FPGA boot flow, you must understand the different conditions that may impact the system, such as reset and hardware configuration settings. In addition, you may also use debug tools such as Arm Development Studio 5 Intel SoC FPGA Edition to load and debug the boot loader software used in your design.

7.1. Reset

Table 18. Reset Effects on Booting and Configuration

<table>
<thead>
<tr>
<th>Reset Type</th>
<th>Initiated By</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power-on Reset</td>
<td>An external event</td>
<td>• The entire HPS and FPGA are reset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the device is released from POR, SDM begins initialization. A POR is the only way initialization can begin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• POR is the only way to recover from a tamper event.</td>
</tr>
<tr>
<td>nCONFIG Reset</td>
<td>nCONFIG pin</td>
<td>An SoC device-wide reset input that cold resets the HPS and reconfigures the FPGA.</td>
</tr>
<tr>
<td>Cold Reset</td>
<td>SDM, HPS_COLD_nRESET pin, Watchdog Timeout Event (calls SDM)</td>
<td>• All of HPS, except the HPS I/O, Clock Manager, Reset Manager, the boot scratch registers in the System Manager, and the TAP controller, are reset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An HPS cold reset does not impact the FPGA core and FPGA I/O (the device is not reconfigured).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The SDM reloads the FSBL into on-chip RAM.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the HPS_COLD_nRESET pin asserts, the SDM begins the reset sequence.</td>
</tr>
<tr>
<td>Cold and Trigger Remote Update Reset</td>
<td>Watchdog Timeout Event (calls SDM)</td>
<td>• SDM requests reset manager to assert or de-assert cold reset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the HPS_COLD_nRESET pin asserts, the SDM begins the reset sequence.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The SDM loads the FSBL from the next bitstream or factory bitstream into on-chip RAM.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The FPGA is first erased and then loaded with an image from the next bitstream or factory bitstream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>There must always be a factory image present.</td>
</tr>
</tbody>
</table>

continued...
### Reset Type

<table>
<thead>
<tr>
<th>Reset Type</th>
<th>Initiated By</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warm Reset</td>
<td>• FSBL or any software that makes a warm reset request through the EL3 register(^3) software write to the RMR_EL3 register to trigger a warm reset to the CPUs. You can still use debug tools after a warm reset because it does not reset the debug modules. The SDM does not reload the FSBL on a warm reset. You must ensure that your FSBL can support reentry, or that your on-chip RAM contains the FSBL or minimum required to boot the system. • Watchdog Timeout Event (calls SDM)</td>
<td>• Before you can write to the RMR_EL3 register, CPU0 must ensure that the other CPUs are in WFI mode. Immediately after the RMR_EL3 register is written, the WFI instruction must be executed. • All of the HPS, except debug, MPU debug, and the boot scratch registers in the System Manager are reset. • A warm reset does not reload the FSBL into the HPS. The FSBL remains in the on-chip RAM during a warm reset. • Warm reset of the HPS does not impact the FPGA core and I/O (the device is not reconfigured). • In the case of a single configuration and boot source, warm reset returns flash control from HPS to SDM.</td>
</tr>
<tr>
<td>Software Reset</td>
<td>A software write to the Reset Manager</td>
<td>• A software reset of a CPU does not affect SDM functionality.</td>
</tr>
<tr>
<td>Watchdog Reset</td>
<td>Timeout from a user configurable watchdog timer register.</td>
<td>• Each CPU has a dedicated watchdog timer. • L4 Watchdog Timer 0 is configured during HPS initialization. • Use the Intel Quartus Prime Pro Edition to configure the watchdog reset for an HPS cold, HPS warm, or HPS cold and trigger remote update reset. See other entries in this table for details on each reset type.</td>
</tr>
<tr>
<td>Debug Reset</td>
<td>JTAG SRST pin</td>
<td>• To control reset, you must connect the HPS_COLD_nRESET pin to the JTAG SRST pin. • A debug reset initiates a cold reset.</td>
</tr>
<tr>
<td>JTAG Reset</td>
<td>JTAG SRST pin</td>
<td>• JTAG can reset the entire device regardless of the MSEL[2:0] settings and reload a new configuration or test through JTAG.</td>
</tr>
</tbody>
</table>

When the device is reset in secure mode, all system warm and watchdog resets are treated as a POR reset. When the HPS is released from reset, all CPUs begin executing the FSBL. The FSBL ensures that CPUs 1 through 3 are in wait-for-interrupt (WFI) mode.

### 7.1.1. HPS Reset Pin

You can configure this pin through Intel Quartus Prime Pro Edition.

---

\(^3\) The Cortex*-A53 has four levels of exception from EL3 to EL0. EL3 is the highest privilege and EL0 is the lowest.
### Table 19. HPS Reset Pins

<table>
<thead>
<tr>
<th>Pin Function</th>
<th>Possible Settings</th>
<th>Functional Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPS cold nreset</td>
<td>SDM_IO0, SDM_IO10-16</td>
<td>Assert this pin to trigger cold reset to the HPS. If the HPS is cold reset via software, this pin becomes an output pin and remain low until the HPS cold reset sequence is complete.</td>
</tr>
</tbody>
</table>

#### 7.1.2. L4 Watchdog Timer 0

Each CPU has its own L4 Watchdog Timer. The HPS FSBL enables L4 Watchdog Timer 0 for CPU0. L4 Watchdog Timer 0 issues a reset when a timeout occurs because of a corrupted bitstream or HPS image or any other issue that causes the HPS to hang.

This watchdog is active until the second-stage boot loader indicates that it has started correctly and taken control of the exception vectors. The timeout is configurable at the FSBL source. U-Boot SPL default is 3 seconds for timeout.

#### 7.2. Debugging the HPS Boot Loader Using the Arm DS-5* Intel SoC FPGA Edition

##### 7.2.1. Selecting the HPS Debug Interface

In Intel Stratix 10 SoC FPGA, you have the option to use split FPGA and HPS JTAG, or combined JTAG. To set this in Intel Quartus Prime Pro Edition, go to Assignments ➤ Device ➤ Device and Pin Options... Select the Configuration category and choose from the HPS debug access port (DAP) drop down options.

![Selecting Debug Interface from Configuration Window](image)

The available options are:
7.2. Setting the Debug Configuration

Before debugging, you must create a Debug Configuration for the Intel Stratix 10 SoC FPGA device in the Arm DS-5* Intel SoC FPGA Edition tool.

1. Connect the device to the host PC using the Intel FPGA Download Cable or Arm DSTREAM interface connector.
   
   Note: The Intel FPGA Download Cable requires Arm DS-5 Intel SoC FPGA Edition version 5.29 or later.

2. Start Arm DS-5 Intel SoC FPGA Edition and select the workspace when prompted.

3. Launch the Debug Configuration windows by selecting Run ➤ Debug Configuration from the main menu.

4. Create a new Debug Configuration by right clicking DS-5 Debugger on the left pane and select New from the displayed menu.

5. Rename the new Debug Configuration accordingly (example: Debug Boot Loader).

6. From the Connection tab, select the target as Intel SoC FPGA ➤ Stratix 10 SoC ➤ Bare Metal Debug ➤ Debug Cortex A53_0.

7. Select the Target Connection as USB-Blaster or DSTREAM accordingly.

8. Click the Browse button in the Connections group and select the appropriate connection.

7.2.3. Debugging the First Stage Boot Loader (FSBL)

1. In the Debugger tab from the Run Control option, select Connect Only.

2. In the same tab, check the Run Debug Initialization Debugger Script option and then click File System. Select the uboot.ds file from the Boot loader Source folder.

3. Click the Debug button to close the Debug Configurations Window and start a debug session. After this step, you can apply standard debugging techniques.
7.2.4. Loading U-Boot

Launch the Arm DS-5 Intel SoC FPGA Edition tool and connect to the DStream debugger. You can load the U-boot image using the following command from the DS-5 Intel SoC FPGA Edition command window:

```
restore C:\intelFPGA_pro\18.0\embedded\examples\hardware/s10_soc_devkit_ghrd\software\u-boot\u-boot-dtb.bin 0x1000
set $PC=0x1000
```

Note: The Disassembly Window must show memory location EL3: 0x000001000.

7.2.5. Debugging U-Boot

This task describes how to debug the first-stage boot loader using the GHRD.

1. Set MSEL[2:0] to 0x7 to enable JTAG as a boot source.
2. Build the SoC EDS GHRD with FPGA Configuration First enabled.
3. Perform a power-on cycle. You must complete a power-on cycle for every debugging cycle.
   This *.sof includes the hps_debug ELF (.axf) file from the<ghrd_directory>/software/hps_debug/ folder. This first stage boot example HPS *.elf provides the bare minimum required to get the HPS in a known initial state. This *.sof programs the entire FPGA so that the HPS does not need to take further action to program the FPGA. You can now load the secondary program loader (SPL) in a controlled way using the debugger.
   This script loads the SPL and the SPL device tree to the low address of the on-chip RAM. This script also sets a breakpoint in the SPL code right before the code attempts to read from its next boot source. The script runs code on the CPU until the breakpoint is hit and then it loads the U-Boot *.elf and the U-Boot device tree to DDR. The script continues until you reach the U-Boot prompt on the serial port.

7.3. Other Debug Considerations

Peripherals

The first step in the board bring-up process is peripheral testing. Add one interface at a time to your design. After a peripheral passes the tests you create for it, remove it from the test design. Avoid leaving peripherals that pass testing in your design as you move to other peripheral tests. Multiple peripherals can create instability due to noise or crosstalk. By testing peripherals in a system individually, you can isolate issues in your design to a particular interface.

A common failure in any system involves memory. The most problematic memory devices operate at high speeds, which can result in timing failures. High performance memory also requires many board traces to transfer data, address, and control signals, which cause failures if they are not routed properly. You can use the Nios® II
processor to verify your memory devices using verification software or a debugger such as the Intel Stratix 10 EMIF Toolkit. The Nios II processor is not capable of stress testing your memory but you can use it to detect memory address and data line issues.

**Data Trace Failure**

If your board fabrication facility does not perform bare board testing, you must perform these tests. To detect data trace failures on your memory interface, use a "walking ones" pattern. The "walking ones" pattern shifts a logical 1 through all of the data traces between the FPGA and the memory device.

The pattern can be increasing or decreasing; the important factor is that only one data signal is 1 at any given time. The increasing version of this pattern is as follows: 1, 2, 4, 8, 16, and so on. Using this pattern, you can detect a few issues with the data traces such as short or open circuit signals. A signal is short circuited when it is accidentally connected to another signal. A signal is open circuited when it is accidentally left unconnected. Open circuits can have a random signal behavior unless you connect a pull-up or pull-down resistor to the trace. If you use a pull-up or pull-down resistor, the signal drives a 0 or 1; however, the resistor is weak relative to a signal being driven by the test, so that test value overrides the pull-up or pull-down resistor. To avoid mixing potential address and data trace issues in the same test, test only one address location at a time. To perform the test, write the test value out to memory, and then read it back. After verifying that the two values are equal, proceed to testing the next value in the pattern. If the verification stage detects a variation between the written and read values, a bit failure has occurred. The table below provides an example of the process used to find a data trace failure. It makes the simplifying assumption that sequential data bits are routed consecutively on the PCB.

**Table 20. Data Trace Test ("Walking Ones") Example**

<table>
<thead>
<tr>
<th>Written Value</th>
<th>Read Value</th>
<th>Failure Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>00000001</td>
<td>No failure detected.</td>
</tr>
<tr>
<td>00000010</td>
<td>00000000</td>
<td>Error, most likely the second data bit, D[1], is stuck low or shorted to ground.</td>
</tr>
<tr>
<td>00000100</td>
<td>00000100</td>
<td>No failure detected, confirmed D[1] is stuck low or shorted to another trace that is not listed in this table.</td>
</tr>
<tr>
<td>00010000</td>
<td>00010000</td>
<td>No failure detected.</td>
</tr>
<tr>
<td>10000000</td>
<td>10000000</td>
<td>No failure detected.</td>
</tr>
</tbody>
</table>

**Address Trace Failure**

The address trace test is similar to the "walking ones" test used for data with one exception. For this test, you must write to all the test locations before reading back the data. Using address locations that are powers of two, you can quickly verify all the address traces of your circuit board. The address trace test detects the aliasing effects that short or open circuits can have on your memory interface. For this reason, it is important to write to each location with a different data value so that you can detect
the address aliasing. You can use increasing numbers such as 1, 2, 3, 4, and so on while you verify the address traces in your system. The table below shows how to use powers of two in the process of finding an address trace failure.

Table 21. Address Trace Test (Powers of Two) Example

<table>
<thead>
<tr>
<th>Address</th>
<th>Written Value</th>
<th>Read Value</th>
<th>Failure Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>1</td>
<td>1</td>
<td>No failure detected.</td>
</tr>
<tr>
<td>00000001</td>
<td>2</td>
<td>2</td>
<td>No failure detected.</td>
</tr>
<tr>
<td>00000010</td>
<td>3</td>
<td>1</td>
<td>Error, the second address bit, A[1], is stuck low.</td>
</tr>
<tr>
<td>00001000</td>
<td>4</td>
<td>4</td>
<td>No failure detected.</td>
</tr>
<tr>
<td>00010000</td>
<td>5</td>
<td>5</td>
<td>No failure detected.</td>
</tr>
<tr>
<td>00100000</td>
<td>6</td>
<td>6</td>
<td>No failure detected.</td>
</tr>
<tr>
<td>10000000</td>
<td>9</td>
<td>9</td>
<td>No failure detected.</td>
</tr>
</tbody>
</table>

**Device Isolation**

Using device isolation techniques, you can disable features of devices on your PCB that cause your design to fail. Typically, designers use device isolation for early revisions of the PCB, and then remove these capabilities before shipping the product. Most designs use crystal oscillators or other discrete components to create clock signals for the digital logic. If the clock signal is distorted by noise or jitter, failures may occur. To guard against distorted clocks, you can route alternative clock pins to your FPGA. If you include SMA connectors on your board, you can use an external clock generator to create a clean clock signal. Having an alternative clock source is very useful when debugging clock-related issues.

Sometimes the noise generated by a particular device on your board can cause problems with other devices or interfaces. Having the ability to reduce the noise levels of selected components can help you determine the device that is causing issues in your design. The simplest way to isolate a noisy component is to remove the power source for the device in question. For devices that have a limited number of power pins, if you include 0 ohm resistors in the path between the power source and the pin, you can cut the power to the device by removing the resistor. This strategy is typically not possible with larger devices that contain multiple power source pins connecting directly to a board power plane. Instead of removing the power source from a noisy device, you can often put the device into a reset state by driving the reset pin to an active state. Another option is to simply not exercise the device so that it remains idle. A noisy power supply or ground plane can create signal integrity issues. With the typical voltage swing of digital devices frequently below a single volt, the power supply noise margin of devices on the PCB can be as little as 0.2 volts. Power supply noise can cause digital logic to fail. For this reason, it is important to be able to isolate the power supplies on your board. You can isolate your power supply by using fuses that are removed so that a stable external power supply can be substituted temporarily in your design.

**JTAG**
FPGAs use the JTAG interface for programming, communication, and verification. Designers frequently connect several components, including FPGAs, discrete processors, and memory devices, communicating with them through a single JTAG chain. Sometimes the JTAG signal is distorted by electrical noise, causing a communication failure for the entire group of devices. To guarantee a stable connection, you must isolate the FPGA under test from the other devices in the same JTAG chain.

**Related Information**
- Intel Stratix 10 SoC Development Kit User Guide
- Intel Stratix 10 External Memory Interfaces IP User Guide
## A. Document Revision History for Intel Stratix 10 SoC FPGA Boot User Guide

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.07.25</td>
<td>19.2</td>
<td>Note added to explain HPS First &quot;Phase 1 configuration&quot; and &quot;Phase 2 configuration&quot; terminology in Boot Flow Overview on page 15.</td>
</tr>
</tbody>
</table>
| 2018.09.24       | 18.1                                   | • Added references to the Single QSPI Flash Boot Example.  
• Updated the filenames in accordance to the SoC EDS 18.1 version.  
• Added reference to the Micron MT25Q Support knowledge base.  
• Added the information about Testing FPGA Reconfiguration at Kernel Level. |
| 2018.05.01       | 18.0                                   | Initial release. |