Intel® Stratix® 10 SoC FPGA Boot User Guide

Updated for Intel® Quartus® Prime Design Suite: **22.4**

Online Version
Send Feedback

UG-20101

ID: 683847
Version: 2023.01.20

Contents

1. Introduction.. 4
1.1. Glossary...4
1.2. Intel Stratix 10 SoC FPGA Boot Overview..5

2. FPGA Configuration First Mode.. 6
2.1. Boot Flow Overview for FPGA Configuration First Mode...............................6
2.1.1. Power-On Reset (POR).. 7
2.1.2. Secure Device Manager.. 8
2.1.3. First-Stage Bootloader..8
2.1.4. Second-Stage Bootloader..9
2.1.5. Operating System...9
2.1.6. Application...9
2.2. System Layout for FPGA Configuration First Mode.....................................10
2.2.1. External Configuration Host Only...10
2.2.2. External Configuration Host with HPS Flash.. 11
2.2.3. Single SDM Flash 11
2.2.4. FPGA Configuration - First Dual Flash System.................................... 12

3. HPS Boot First Mode.. 14
3.1. Boot Flow Overview...14
3.1.1. Power-On Reset (POR)..16
3.1.2. Secure Device Manager... 16
3.1.3. First-Stage Bootloader...17
3.1.4. Second-Stage Bootloader...18
3.1.5. Operating System... 18
3.1.6. Application..19
3.2. System Layout for HPS Boot First Mode.. 19
3.2.1. External Configuration Host Only...19
3.2.2. External Configuration Host with HPS Flash.. 20
3.2.3. Single SDM Flash 20
3.2.4. HPS Boot First - Dual Flash System... 22

4. Creating the Configuration Files...23
4.1. Overview.. 23
4.2. Intel Quartus Prime Hardware Project Compilation......................................24
4.2.1. Device and Pin Options.. 24
4.2.2. Platform Designer Options..25
4.3. Bootloader Software Compilation... 25
4.4. Programming File Generator...26
4.5. Configuration over JTAG...27
4.5.1. FPGA Configuration First.. 27
4.5.2. HPS Boots First..27
4.6. Configuration from QSPI.. 28
4.6.1. Supported QSPI Devices.. 29
4.6.2. FPGA Configuration First.. 29
4.6.3. HPS Boot First... 33
4.7. Configuration over AVST.. 38
4.7.1. FPGA Configuration First.. 38

Send Feedback

4.7.2. HPS Boot First.......... 41
4.8. Configuration via Protocol.......... 44
4.8.1. Creating Configuration Files from Command Line..........45
4.8.2. Creating Configuration Files Using Graphical Interface..........46
4.9. Remote System Update.......... 49
4.10. Partial Reconfiguration.......... 50

5. Golden System Reference Design and Design Examples..........53
5.1. Golden System Reference Design.......... 53
5.2. Bootloader Examples..........53
5.3. Additional Design Examples..........53

6. Configuring the FPGA Fabric from HPS Software..........55
6.1. Configuring the FPGA Fabric from U-Boot.......... 55
6.2. Configuring the FPGA Fabric from Linux.......... 56
6.3. FPGA Partial Reconfiguration from Linux..........57

7. Debugging the Intel Stratix 10 SoC FPGA Boot Flow.......... 61
7.1. Reset.......... 61
7.1.1. HPS Reset Pin..........62
7.1.2. L4 Watchdog Timer 0..........63
7.2. Debugging the HPS Bootloader Using the Arm DS Intel SoC FPGA Edition.......... 63
7.3. Other Debug Considerations..........63

8. Intel Stratix 10 SoC FPGA Boot User Guide Archives.......... 67

9. Document Revision History for Intel Stratix 10 SoC FPGA Boot User Guide..........68

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.

1.1. Glossary

Table 1. Intel Stratix 10 SoC FPGA Boot Glossary

Term	Definition
EMIF	External memory interface
`*.pof`	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.
FW	Firmware; Controls and monitors software stored in Secure Device Manager's (SDM) read-only memory.
HPS	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.
HPS EMIF I/O section	Part of the raw binary file (`*.rbf`) that configures the EMIF I/O used by the HPS
FPGA I/O section	Part of the `*.rbf` that configures the I/O assigned to the FPGA core
Core `*.rbf`	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.
FSBL	First-stage Bootloader for HPS
`*.jic`	JTAG Indirect Configuration file that allows programming through JTAG
OS	Operating system
`*.rbf`	Raw binary file representing the FPGA bitstream
`*.rpd`	Raw Programming Data file for AS devices

continued...

Term	Definition
`*.sof`	SRAM object file which contains the bitstream for the primary FPGA design. Firmware is not part of the `*.sof`.
SSBL	Second-stage Bootloader for HPS
SDM	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.

1.2. 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—When you select the FPGA First option, the SDM fully configures the FPGA, then configures the HPS SDRAM pins, loads the HPS first-stage bootloader (FSBL) and takes the HPS out of reset.

  Note: 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—When you select the HPS First option, the SDM first configures the HPS SDRAM pins, loads the HPS FSBL and takes the HPS out of reset. Then the HPS configures the FPGA I/O and FPGA fabric at a later time.

  Note: 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.

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

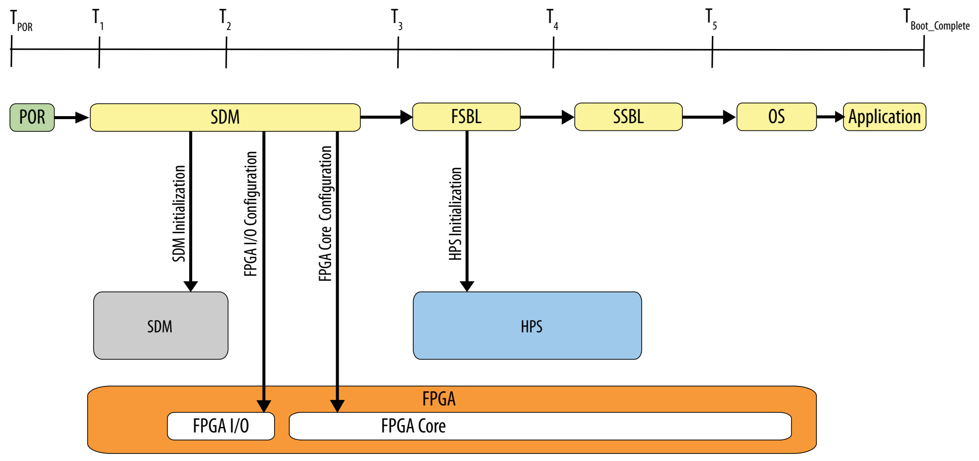


Table 2. FPGA Configuration First Stages

The sections following this table describe each stage in more detail.

Time	Boot Stage	Device State
T_{POR} to T_1	POR	Power-on reset
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 Bootloader (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 Bootloader (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.

Related Information

- Intel Stratix 10 Device Data Sheet
  For information about the POR delay specification
- AN 692: Power Sequencing Considerations for Intel Cyclone® 10 GX, Intel Arria® 10, Intel Stratix 10, and Intel Agilex™ Devices

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

SDM Boot Source	Details
Avalon-ST (x8/x16/x32)	Supported
JTAG	Supported
Active Serial (AS)/ Quad SPI	Supported. SDM only boots in x4 mode for active serial flash and Micron* MT25Q flash. Other supported quad SPI flash devices boot in x1 mode. After the configuration firmware loads into the SDM, the SDM can switch the flash into x4 mode.

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

- System Layout for HPS Boot First Mode on page 19
- Intel Stratix 10 Configuration User Guide

2.1.3. First-Stage Bootloader

The first-stage bootloader (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. 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 control to the SSBL.

[1] The `MSEL[2:0]` pins are multiplexed with the `SDM_IO[9]`, `SDM_IO[7]`, `SDM_IO[5]` pins respectively.

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

- U-Boot secondary program loader (SPL)
  - Intel provides the source code for U-Boot on GitHub.
- Arm Trusted Firmware
  - Intel provides the source code for the Arm Trusted Firmware on GitHub.

Related Information

- Creating the Configuration Files on page 23
- U-Boot Source Code on GitHub
- Arm Trusted Firmware Source Code on GitHub

2.1.4. Second-Stage Bootloader

The second-stage bootloader (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 the source code for U-Boot on GitHub.
- UEFI
  - Intel provides the source code for UEFI on GitHub.
- RTOS
- Bare Metal application

Related Information

- UEFI Source Code on GitHub
- U-Boot Source Code on GitHub
- Arm Trusted Firmware Source Code on GitHub

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 the Golden System Reference Design (GSRD) which includes the Linux kernel and a root filesystem built with Yocto recipes.

Related Information

Golden System Reference Design and Design Examples on page 53

2.1.6. Application

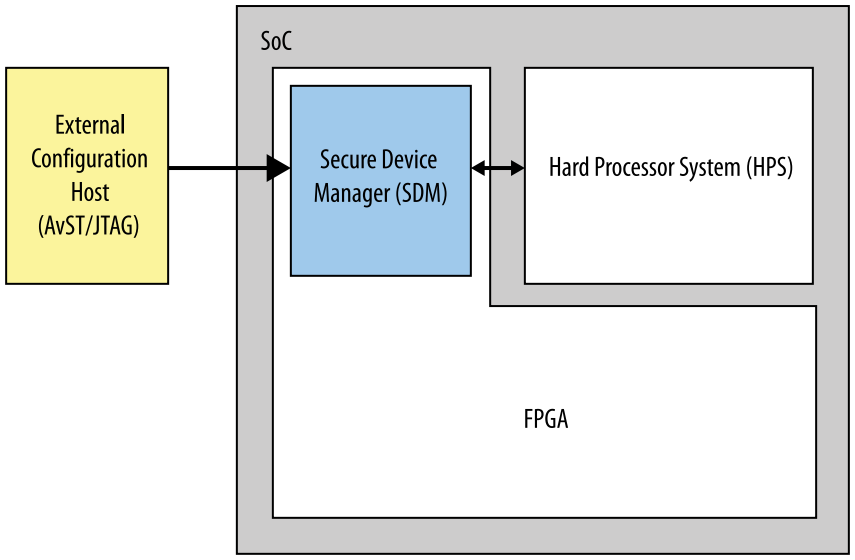
The application that runs on the OS is the last boot stage. The application can also replace the OS stage as a dedicated, Bare Metal runtime code.

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


SoC
External Configuration Host (AvST/JTAG)
Secure Device Manager (SDM)
Hard Processor System (HPS)
FPGA

In this example, the external configuration host (Avalon® streaming 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) 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).

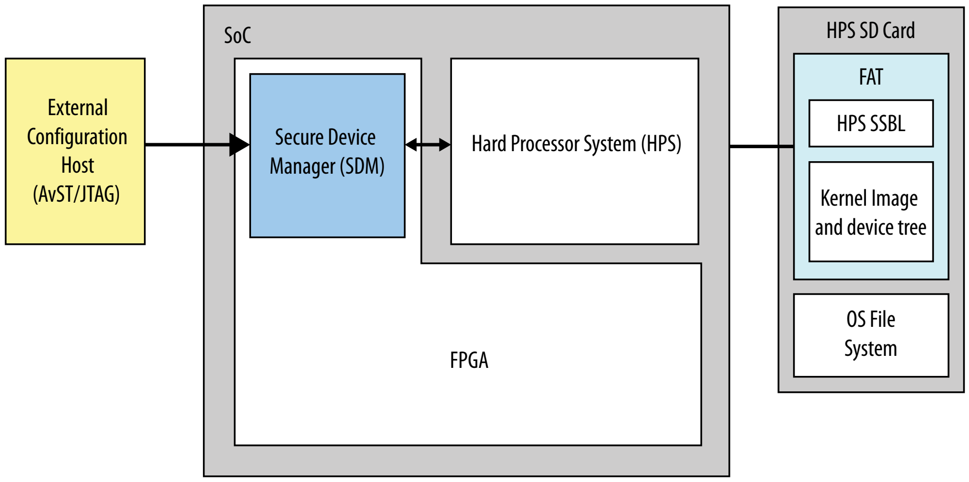
You can use the FSBL to retrieve the SSBL from other sources, such as through the HPS Ethernet MAC interface. To implement these modes of access, you must create a working Ethernet software stack in the FSBL.

Table 4. Supported Configuration Boot and SSBL Source

SDM Configuration Host	SSBL Source	Details
Avalon streaming	HPS Ethernet	Not supported in U-Boot FSBL code provided by Intel.
JTAG		

2.2.2. External Configuration Host with HPS Flash

Figure 3. External Configuration Host with HPS Flash


External Configuration Host (AvST/JTAG)
SoC
Secure Device Manager (SDM)
Hard Processor System (HPS)
FPGA
HPS SD Card
FAT
HPS SSBL
Kernel Image and device tree
OS File System

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.

All HPS flash devices are supported:

- SD Card
- eMMC
- NAND

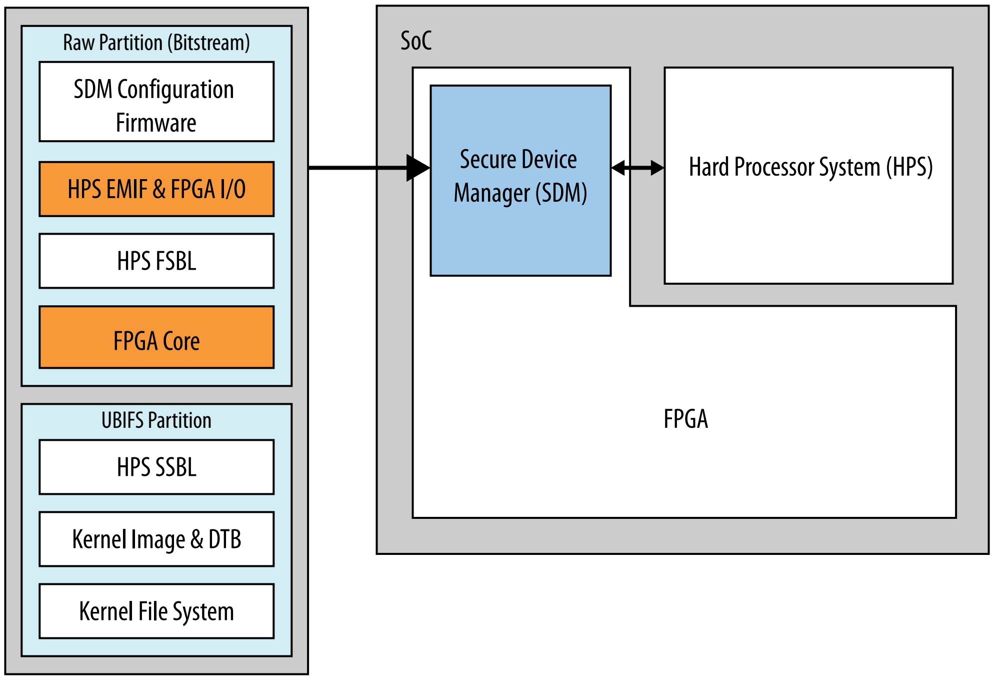
2.2.3. Single SDM Flash

In this case, the Quad SPI flash connected to the SDM contains all the data required for configuring and booting the system, including the configuration bitstream, bootloader and OS files.

Note: When you use the HPS to access the SDM Quad SPI, it operates at a lower bandwidth of ~4-6 MB/s. This is due to the high latency of the PSI link between HPS and SDM, and the fact that all transfers are done in Programmed IO (PIO) mode, instead of DMA mode.

Software running on the HPS must request permission from the SDM to get exclusive access to the QSPI before using it. This is already implemented in the U-Boot and UEFI bootloaders supported by Intel.

Figure 4. FPGA Configuration First Layout with Quad SPI


Raw Partition (Bitstream)
SDM Configuration Firmware
HPS EMIF & FPGA I/O
HPS FSBL
FPGA Core
UBIFS Partition
HPS SSBL
Kernel Image & DTB
Kernel File System
SoC
Secure Device Manager (SDM)
Hard Processor System (HPS)
FPGA

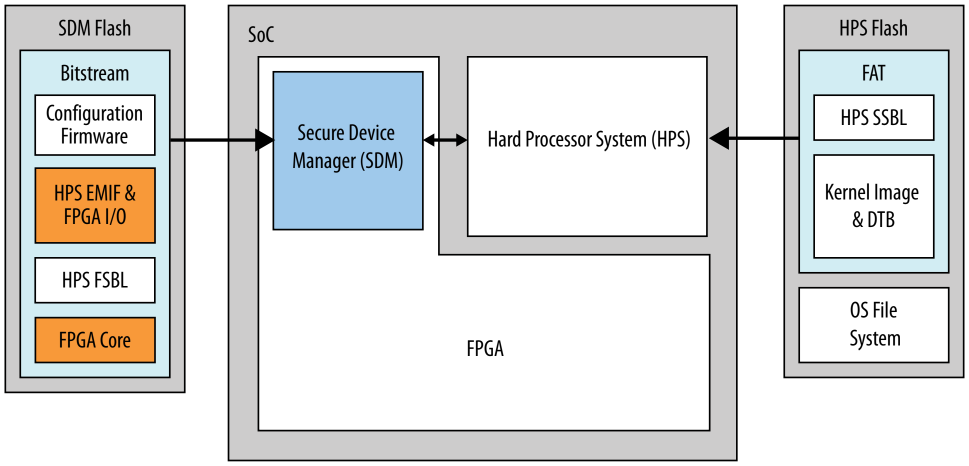
Related Information

- Intel Stratix 10 Hard Processor System Technical Reference Manual
  For more information, refer to the Booting and Configuration Appendix
- Intel Stratix 10 SoC FPGA First Single QSPI Flash Boot

2.2.4. FPGA Configuration - First Dual Flash System

In a dual flash system, the SDM flash stores the configuration bitstream, while the HPS flash stores the HPS SSBL and the rest of the OS files.

Figure 5. FPGA Configuration First - Dual Flash devices (SDM and HPS)



Related Information

- RocketBoards: Intel Stratix 10 SoC GSRD
- Golden System Reference Design and Design Examples on page 53

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 `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 First Boot mode, you have the option of configuring the FPGA core during the SSBL stage or after the operating system boots.

Note: Configuring the HPS EMIF I/O for the first time and then loading the HPS FSBL is called "Phase 1 configuration". The subsequent configuration of FPGA core and periphery by HPS is called "Phase 2 configuration". The phase 1 and phase 2 configuration files must be generated from the same Intel Quartus Prime Pro Edition software version, this includes patches installed if applicable.

A typical HPS First Boot flow may look like the following figure. You can use U-Boot, Unified Extensible Firmware Interface (UEFI), or a custom bootloader 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 6. Typical HPS Boot First Flow

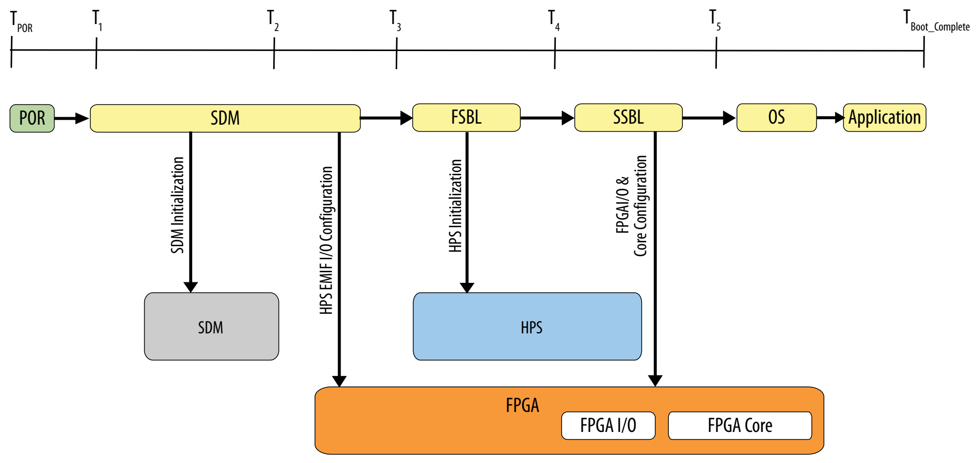


Table 5. HPS Boot First Stages

Time	Boot Stage	Device State
T_{POR}	POR	Power-on reset
T_1 to T_2	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_2 to T_3	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_3 to T_4	First Stage Bootloader (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. 6. HPS bootstrap completes.
T_4 to T_5	Second Stage Bootloader (SSBL)	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 that the SDM configure the FPGA core.(2) 3. FPGA enters user mode. 4. OS is loaded into SDRAM.
T_5 to $T_{Boot\_Complete}$	Operating System (OS)	1. OS boot occurs and the OS schedules applications for runtime launch. 2. (Optional step)The 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. For more information, refer to the *System Layout for HPS Boot First Mode* section.

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

Note: To avoid configuration failures, the Intel Stratix 10 device requires clocks for the PCIe* and all E-tile transceiver reference clocks. You must provide the input reference clock, `refclk`, and it must be free-running and stable at device power up for a successful device configuration.

L- and H-tile (does not apply to E-tile)—the `refclk` requirement is mandatory if you are configuring your Intel Stratix 10 device over a PCIe link; otherwise the `refclk` requirement is not mandatory for a non PCIe use case.

- For PCIe use case, the firmware waits for the PLL calibration code to ensure the PLL is calibrated properly in order to release the device for entering into user mode. Therefore, `refclk` is mandatory for PLL calibration.
- For non PCIe use case, without `refclk` supply during configuration, the firmware does not gate device configuration without a proper PLL calibration code. You can calibrate the XCVR PLL in user mode for XCVR channels to operate properly.

E-tile (does not apply to L- and H-tile)—the `refclk` requirement is mandatory. E-tile does not support the PCIe use case.

- The E-tile `refclk` is needed to load the firmware (from the FPGA configuration bit stream) into the E-Tile.

Related Information

- System Layout for HPS Boot First Mode on page 19
- Intel Stratix 10 Configuration User Guide

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 Data Sheet
  For information about the POR delay specification
- AN 692: Power Sequencing Considerations for Intel Cyclone® 10 GX, Intel Arria® 10, Intel Stratix 10, and Intel Agilex™ 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 6. Available SDM Boot Sources for the Intel Stratix 10 SoC FPGA

SDM Boot Source	Details
Avalon-ST (x8/x16/x32)	Supported
JTAG	Supported
Active Serial (AS)/ Quad SPI	Supported. SDM only boots in x4 mode for active serial flash and Micron MT25Q flash. Other supported quad SPI flash devices boot in x1 mode. Once the device initialization code loads into the SDM, the SDM can switch these flash into x4 mode.

The typical configuration bitstream for HPS boot first mode contains:

- SDM configuration firmware
- HPS external memory interface (EMIF) I/O configuration data
- 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

- System Layout for HPS Boot First Mode on page 19
- Intel Stratix 10 Configuration User Guide

3.1.3. First-Stage Bootloader

After the SDM releases the HPS from reset, the FSBL initializes the HPS. Initialization includes configuring clocks, HPS dedicated I/Os, and peripherals.

Note: In HPS first boot mode, the SDM, HPS OSC and HPS EMIF clocks must be running stable and set at the correct frequency before you begin any part of the configuration sequence.

In HPS first boot mode, phase 1 configuration is successful as long as HPS OSC and HPS EMIF clocks are running stable.

For a generic transceiver use case, if the XCVR ref clock is not running during phase 2 configuration, the phase 2 configuration still succeeds.

For a PCIe use case, if the PCIe ref clock is not running during phase 2 configuration, the configuration fails.

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

- U-Boot secondary program loader (SPL)
  — Intel provides the source code for U-Boot on GitHub.
- Arm Trusted Firmware
  — Intel provides the source code for the Arm Trusted Firmware on GitHub.

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

Related Information

- Creating the Configuration Files on page 23

- U-Boot Source Code on GitHub
- Arm Trusted Firmware Source Code on GitHub

3.1.4. Second-Stage Bootloader

The second-stage bootloader (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 the command line interface.

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

- U-Boot
  — Intel provides the source code for U-Boot on GitHub.
- UEFI
  — Intel provides the source code for UEFI on GitHub.
- RTOS
- Bare Metal application

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 Fabric from HPS Software on page 55
- UEFI Source Code on GitHub
- U-Boot Source Code on GitHub
- Arm Trusted Firmware Source Code on GitHub

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 the Golden System Reference Design (GSRD) which includes the Linux kernel and a root filesystem built with Yocto recipes.

Related Information

Golden System Reference Design and Design Examples on page 53

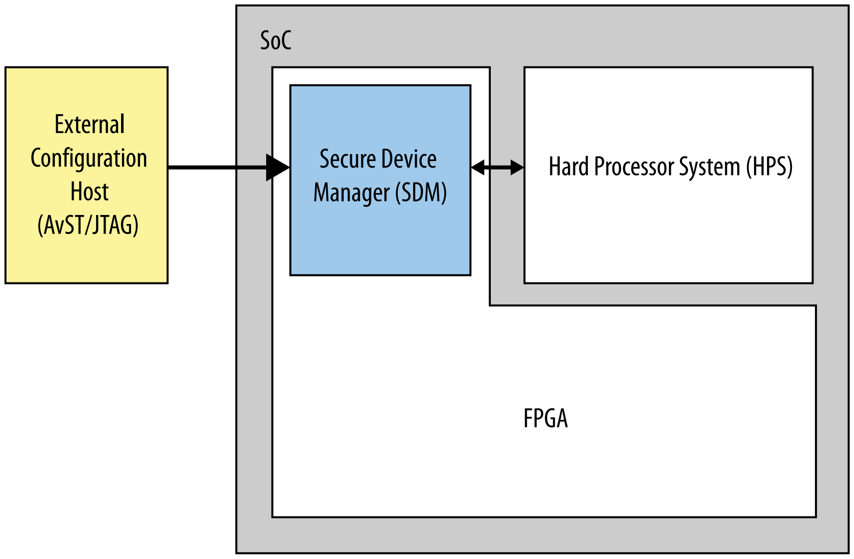
3.1.6. Application

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 7. External Configuration Host Only


SoC
External Configuration Host (AvST/JTAG)
Secure Device Manager (SDM)
Hard Processor System (HPS)
FPGA

In this example, the external configuration host (Avalon Streaming or JTAG) provides the SDM a configuration bitstream that consists of the following components:

- 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 such as Bare Metal applications.

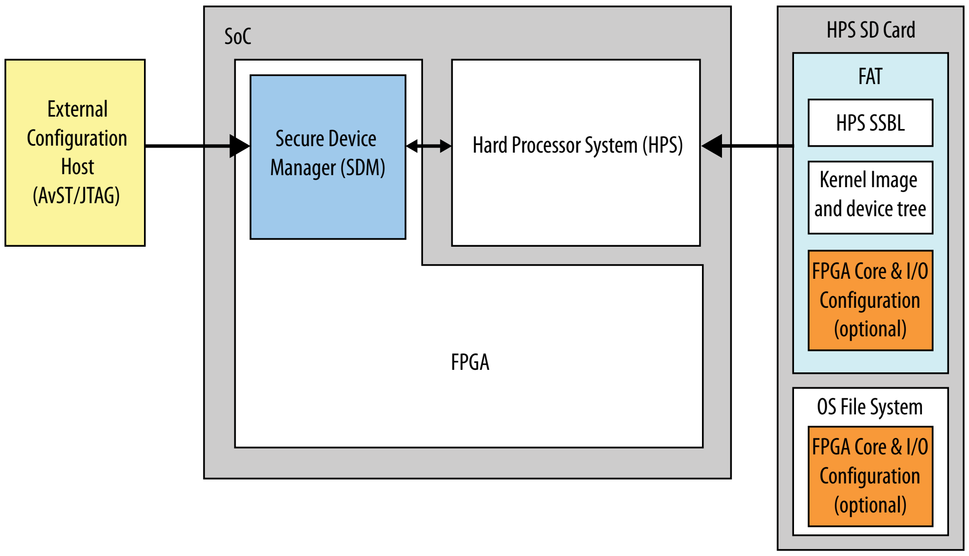
Because the FPGA core is not configured, the FSBL must retrieve the SSBL from external sources, such as the HPS EMAC interface. The U-Boot FSBL source code that Intel provides does not include source code to support SSBL retrieval through the HPS EMAC interface. You must implement the Ethernet software 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 7. Supported Configuration Boot and SSBL Sources

SDM Configuration Host	SSBL Source	Details
Avalon streaming	HPS Ethernet	Not supported in U-Boot FSBL code that Intel provides.
JTAG		

3.2.2. External Configuration Host with HPS Flash

Figure 8. External Configuration Host with HPS Flash


External Configuration Host (AvST/JTAG)
SoC
Secure Device Manager (SDM)
Hard Processor System (HPS)
FPGA
HPS SD Card
FAT
HPS SSBL
Kernel Image and device tree
FPGA Core & I/O Configuration (optional)
OS File System
FPGA Core & I/O Configuration (optional)

An external configuration host provides an SDM configuration bitstream containing the following components:

- 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, contains the HPS SSBL, Linux image device tree information, and the OS file system.

Depending on the boot stage that performs the FPGA configuration, you have the following options for storing the FPGA core and I/O configuration file:

- In the HPS flash partition—The SSBL initiates configuration.
- In the OS file system—The OS initiates configuration

All HPS flash devices are supported:

- SD Card
- eMMC
- NAND

3.2.3. Single SDM Flash

In this case, the Quad SPI flash connected to the SDM contains all the data required for configuring and booting the system, including the configuration bitstream, bootloader and OS files.

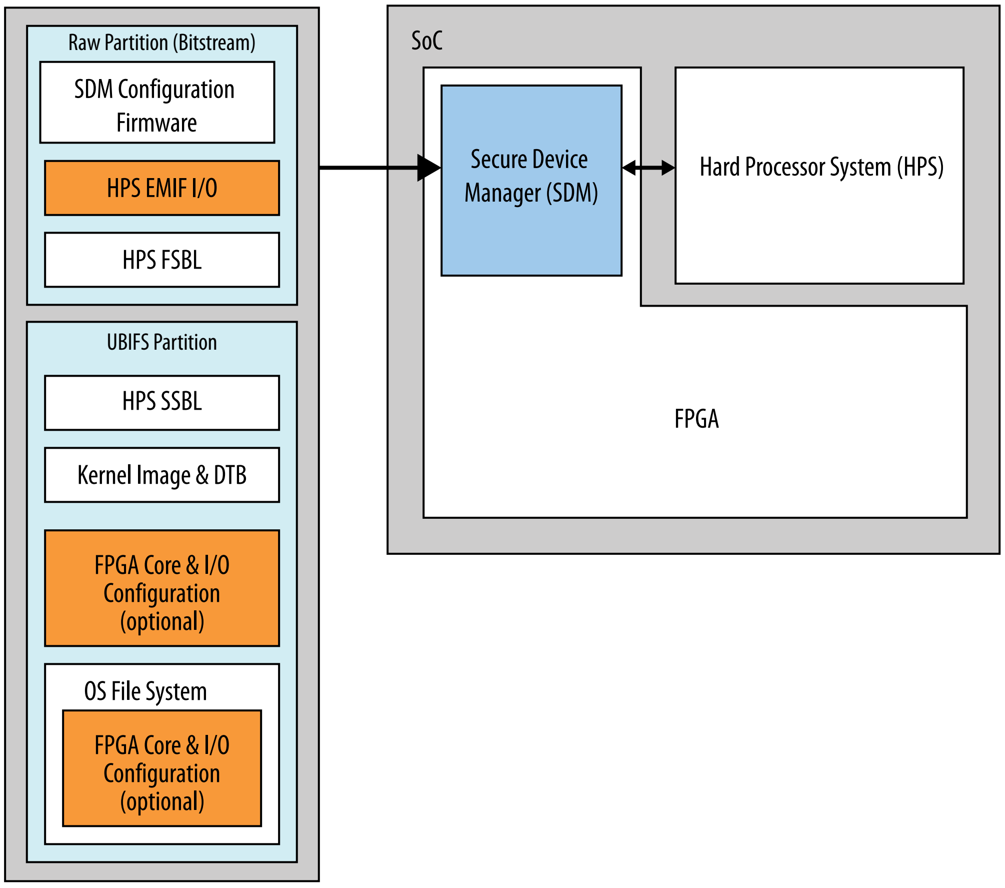
Note: When you use the HPS to access the SDM Quad SPI, it operates at a lower bandwidth of ~4-6 MB/s. This is due to the high latency of the PSI link between HPS and SDM, and the fact that all transfers are done in Programmed IO (PIO) mode, instead of DMA mode.

Software running on the HPS must request permission from the SDM to get exclusive access to the QSPI before using it. This is already implemented in the U-Boot and UEFI bootloaders supported by Intel.

Depending on the boot stage that performs the FPGA configuration, you have the following options for storing the FPGA core and I/O configuration file:

- An SDM flash storage partition—In this case the SSBL initiates configuration
- In the OS file system—In this case the OS initiates configuration

Figure 9. HPS Boot First Layout with Quad SPI


Raw Partition (Bitstream)
SDM Configuration Firmware
HPS EMIF I/O
HPS FSBL
UBIFS Partition
HPS SSBL
Kernel Image & DTB
FPGA Core & I/O Configuration (optional)
OS File System
FPGA Core & I/O Configuration (optional)
SoC
Secure Device Manager (SDM)
Hard Processor System (HPS)
FPGA

Related Information

- Intel Stratix 10 Hard Processor System Technical Reference Manual
  For more information, refer to the Booting and Configuration Appendix
- Intel Stratix 10 SoC FPGA First Single QSPI Flash Boot
- Creating the Configuration Files on page 23
- Intel Stratix 10 Hard Processor System Technical Reference Manual

3.2.4. HPS Boot First - Dual Flash System

In a dual flash system, the SDM flash such as an active serial flash stores the configuration bitstream. The HPS flash such as an SD card stores the HPS SSBL and the rest of the OS files.

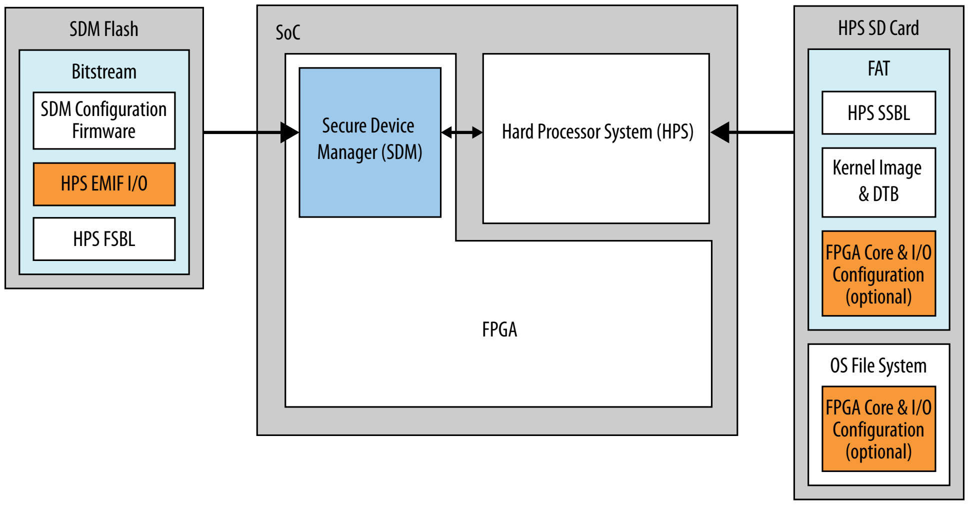
All the HPS flash devices are supported:

- SD Card
- eMMC
- NAND

Depending on the boot stage that performs the FPGA configuration, you have the following options for storing the FPGA core and I/O configuration file:

- In the HPS flash partition—The SSBL initiates configuration.
- In the OS file system—The OS initiates configuration

Figure 10. HPS Boot First - Dual Flash Devices (SDM and HPS)


SDM Flash
Bitstream
SDM Configuration Firmware
HPS EMIF I/O
HPS FSBL
SoC
Secure Device Manager (SDM)
Hard Processor System (HPS)
FPGA
HPS SD Card
FAT
HPS SSBL
Kernel Image & DTB
FPGA Core & I/O Configuration (optional)
OS File System
FPGA Core & I/O Configuration (optional)

4. Creating the Configuration Files

There are several different configuration methods available:

- Configuration over JTAG
- Configuration from QSPI
- Configuration over AVST
- Configuration via Protocol
- Remote System Update (RSU)

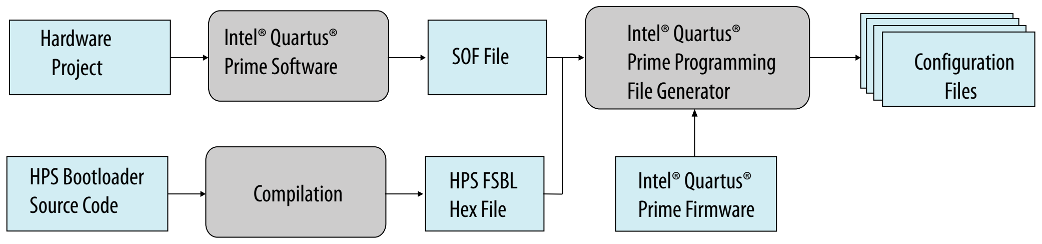
This chapter describes how to create the configuration files for all these configuration methods.

4.1. Overview

The configuration files are created by the Intel Quartus Prime Programming File Generator, using the following inputs:

- SOF file resulted from compilation of the hardware project in Intel Quartus Prime Software
- HPS First Stage Bootloader (FSBL) hex file resulted from compiling an HPS bootloader
- Intel Quartus Prime Firmware, which ends up running on the SDM

Figure 11. Overview of Configuration File Generation


Hardware Project
Intel® Quartus® Prime Software
SOF File
Intel® Quartus® Prime Programming File Generator
Configuration Files
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Firmware

The resulted configuration files contain the following components which are required for configuring the device:

Table 8. Components for Configuring the Device

Source	Component	Description
SOF File	HPS IO Configuration Data	Used for configuring HPS IOs including HPS DDR
	FPGA IO Configuration Data	Used for configuring FPGA IOs
	FPGA Fabric Configuration Data	Used for configuring the FPGA fabric

continued...

Source	Component	Description
	Handoff Data for SDM Firmware	Used to pass parameters to SDM firmware
	Handoff Data for HPS FSBL	Used to pass parameter to HPS FSBL
Intel Quartus Prime	SDM Firmware	Located at the beginning of configuration bitstream and can be executed as part of the configuration.
Your HPS Software	HPS FSBL	First software ran on HPS after SDM takes it out of reset

4.2. Intel Quartus Prime Hardware Project Compilation

There are various settings that need to be provided in Intel Quartus Prime for the hardware project, which impact the device configuration, and HPS booting. This section presents a summary of these options.

4.2.1. Device and Pin Options

The device and pin options can be accessed from Intel Quartus Prime, by going to **Assignments ➤ Device ➤ Device and Pin Options**. The most important options related to configuration and HPS boot are:

- **General ➤ Configuration Clock Source** : allows using an internal oscillator or an external input clock for configuration purposes.
- **Configuration ➤ Configuration Scheme**: allows selecting the configuration source:
  - Active Serial x4
  - AVST x8
  - AVST x16
  - AVST x32
- **Configuration ➤ Active Serial Clock Source**: allows selecting the QSPI clock speed when Active Serial x4 mode is selected
- **Configuration ➤ Configuration Pin Options**: allows selecting SDM pin behavior for configuration purposes
- **Configuration ➤ HPS/FPGA Configuration Order**: allows selecting FPGA Configuration First (called `After INIT_DONE`) or HPS Boot First (called `HPS First`) modes.
- **Configuration ➤ HPS Debug Access Port (DAP)**: allows the HPS JTAG port to be connected to **HPS Pins**, **FPGA Pins** or **Disabled**. It is typically connected to SDM pins, so you can have a single JTAG connection covering both SDM and HPS.
- **CvP Settings ➤ Configuration via Protocol**: can be selected as **Initialization and update** or **Off**.

For more information about these options, refer to the *Intel Stratix 10 Configuration User Guide*.

Related Information

- System Layout for HPS Boot First Mode on page 19
- Intel Stratix 10 Configuration User Guide

4.2.2. Platform Designer Options

The HPS component instantiated in Platform Designer has various selectable HPS settings. You can access them by doing the following:

1. Open the hardware project in the Intel Quartus Prime GUI.
2. In the Intel Quartus Prime GUI, to **Tools ➤ Platform Designer** to open the Platform Designer.
3. When asked by Platform Designer, select and open the file instantiating the HPS component.
4. In Platform Designer, click the HPS component and access the **Parameters** panel.

The available settings are grouped as follows:

- FPGA Interfaces
  - General
  - Bridges
  - DMA Requests
  - Interrupts
- HPS Clocks and Resets
  - Input Clocks
  - Internal Clocks and Output Clocks
  - Resets
- SDRAM
- IO Delays
- Pin Mux and Peripherals

For more information about these settings, refer to the *Intel Stratix 10 HPS Component User Guide*.

Related Information

Intel Stratix 10 Hard Processor System Component Reference Manual

4.3. Bootloader Software Compilation

Intel supports the following bootloaders:

- U-Boot
- UEFI

For more information about the bootloaders, including how to configure and build for the various booting scenarios, refer to the BuildingBootloader web page on RocketBoards.

Related Information

RocketBoards: BuildingBootloader

4.4. Programming File Generator

The tool that is used for creating the configuration files is called Programming File Generator. This tool is part of the Intel Quartus Prime Programmer, which is included with the full Intel Quartus Prime software installation. The Intel Quartus Prime Programmer can also be downloaded and installed separately as a standalone application, in which case less disk space would be required.

The Programming File Generator tool can be called from command line using the `quartus_pfg` command, or as a graphical user interface application using `qpfgw` command. Both command line version and graphical interface version offer the same functionality with a couple of exceptions:

- Adding an HPS FSBL hex file to a SOF file is only supported by the command line version.
- Creating the initial flash contents for a Remote System Update (RSU) system is only supported by the graphical interface version.

The command line version is typically more convenient to use, as it requires less parameters to be entered than the graphical interface version. The graphical interface version also has the capability of saving all the selected options in a `.pfg` file, by going to **File ➤ Save** or **File ➤ Save As** menus. The saved `.pfg` file can later be used to re-create the configuration files by running the `quartus_pfg -c filename.pfg` command. The `.pfg` file is an XML file, typically not editable by hand. One useful edit though is to replace absolute filenames with relative ones, so that the tool could be ran in a different folder than the one where the `.pfg` file was originally created.

The following table summarizes the file types handled by the Intel Quartus Prime Programming File Generator:

Table 9. Intel Quartus Prime Programming File Generator File Types

File Extension	File Type	Description
`.jic`	JTAG Indirect Configuration File	These files are intended to be written to QSPI flash by using the Intel Quartus Prime Programmer tool. They contain the actual flash data, and also a flash loader, which is a small FPGA design used by the Intel Quartus Prime Programmer to write the data.
`.rpd`	Raw Programming Data File	These files contain actual binary content for the flash and no additional metadata. They can contain the full content of the flash, similar with the `.jic` file—this is typically used in the case where an external tool is used to program the initial flash image. They can also contain an RSU application image, or RSU factory update image.
`.rbf`	Raw Binary File	These files are binary files which can be used typically to configure the FPGA fabric for HPS configuration first mode. They can also be used for passively configuring the FPGA device through Avalon Streaming Interface. They are also used for configuring the FPGA fabric for CvP case. The maximum size of the `.core.rbf` file for the FPGA core fabric from the HPS does not exceed the bit-stream size for the FPGA from a configuration device. The maximum size of bit-stream can be found in the *Configuration Bit Stream Sizes* section in the *Intel Stratix 10 Device Data Sheet*.
`.map`	Memory Map File	These files contain details about where the input data was placed in the output file. This file is human readable.

Related Information

Intel Stratix 10 Device Data Sheet

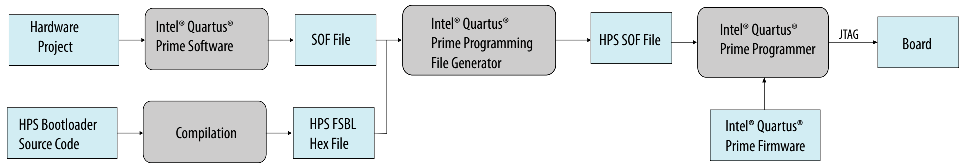
4.5. Configuration over JTAG

In this case, the configuration bitstream is sent to the device over JTAG with the help of the Intel Quartus Prime Programmer.

4.5.1. FPGA Configuration First

The following figure shows an overview of the process:

Figure 12. Configuration over JTAG with FPGA Configuration First


Hardware Project
Intel® Quartus® Prime Software
SOF File
Intel® Quartus® Prime Programming File Generator
HPS SOF File
Intel® Quartus® Prime Programmer
JTAG
Board
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Firmware

The following steps are involved:

1. Compile hardware project with Intel Quartus Prime to obtain the SOF file.
2. Compile the HPS FSBL source code to obtain the HPS FSBL `hex` file or use a precompiled HPS FSBL `hex` file.
3. Add the HPS FSBL `hex` file to the SOF file to obtain the HPS SOF File. SOF files resulted from compiling hardware designs that have HPS instantiated cannot be used directly to configure the device.
4. Use the Intel Quartus Prime Programmer to configure the device over JTAG with the resulted HPS SOF file. The required firmware to run on the SDM must be downloaded on the device by the Intel Quartus Prime Programmer.

Run the following command to add the HPS FSBL `hex` file to the SOF file to create the HPS SOF file:

```
quartus_pfg -c design.sof design_hps.sof -o hps_path=fsbl.hex
```

The input and output files for this command are:

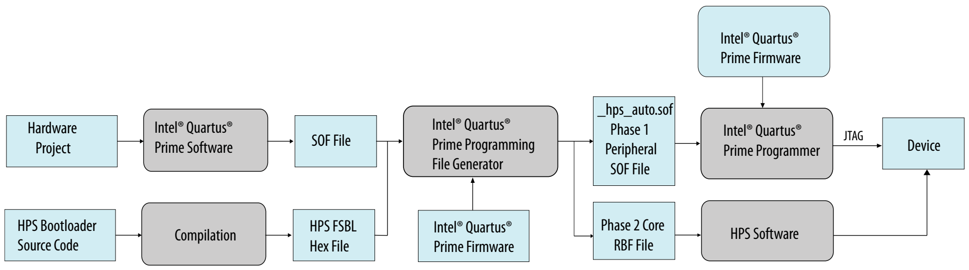
- Input Files:
  - `design.sof`
  - `fsbl.hex`
- Output File:
  - `design_hps.sof`

4.5.2. HPS Boots First

In order to obtain the SOF file to be used to configure the device through JTAG with the HPS Boots First option, you need to generate a bitstream intended for another configuration method, such as AVST. An SOF file called `design_hps_auto.sof` is expected to be created automatically, and it can be used to configure the device. You discard or ignore the Phase 1 AVST bitstream, and use the Phase 2 bitstream to later configure the FPGA fabric from HPS software.

The following figure shows an overview of the process:

Figure 13. Configuration over JTAG with HPS Boots First


Intel® Quartus® Prime Firmware
Hardware Project
Intel® Quartus® Prime Software
SOF File
Intel® Quartus® Prime Programming File Generator
_hps_auto.sof Phase 1 Peripheral SOF File
Intel® Quartus® Prime Programmer
JTAG
Device
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Firmware
Phase 2 Core RBF File
HPS Software

The following steps are involved:

1. Compile hardware project with Intel Quartus Prime to obtain the SOF file.
2. Compile the HPS FSBL source code to obtain the HPS FSBL `hex` file, or use a precompiled one.
3. Use Programming File Generator to create the following files:
    - Raw Binary File (`RBF`): contains the small phase 1 configuration bitstream. Discard this file.
    - `hps_auto` SOF File: contains the phase 1 configuration data, and the HPS FSBL.
    - `Core` RBF File: contains the typically much larger phase 2 configuration bitstream. To be used by HPS software later to configure the fabric.
4. Use Intel Quartus Prime Programmer to configure the device using the phase 1 `hps_auto` SOF file. HPS software starts running, beginning with HPS FSBL.
5. At a later time, HPS software configures the FPGA fabric by using the phase 2 Core RBF bitstream.

The following example creates the files for HPS boot first:

```
quartus_pfg -c design.sof design.rbf -o hps_path=fsbl.hex -o hps=on
```

The input and output files for this command are:

- Input Files:
    — `design.sof`
    — `fsbl.hex`
- Output Files:
    — `design.hps.rbf`—to be discarded
    — `design_hps.auto.sof`—Phase 1 SOF
    — `design.core.rbf`—Phase 2 Core RBF

4.6. Configuration from QSPI

The device configures itself with the bitstream which it reads from QSPI flash. This configuration method is also called “Active Serial x4” or “ASx4”.

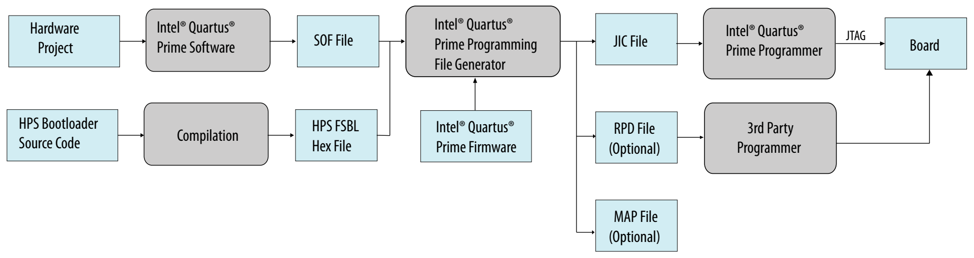
4.6.1. Supported QSPI Devices

For the list of supported QSPI devices, refer to the Intel Supported Configuration Devices web page.

4.6.2. FPGA Configuration First

The following figure shows an overview of configuring from QSPI when using FPGA configuration first:

Figure 14. Configuration from QSPI using FPGA Configuration First


Hardware Project
Intel® Quartus® Prime Software
SOF File
Intel® Quartus® Prime Programming File Generator
JIC File
Intel® Quartus® Prime Programmer
JTAG
Board
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Firmware
RPD File (Optional)
3rd Party Programmer
MAP File (Optional)

The following steps are involved:

1. Compile hardware project with Intel Quartus Prime to obtain the SOF file.
2. Compile the HPS FSBL source code to obtain the HPS FSBL hex file or use a precompiled one.
3. Use Programming File Generator to create the following files:
    - JTAG Indirect Configuration (JIC) File: contains the configuration bitstream to be written to flash and SDM helper image used by the Intel Quartus Prime Programmer to write the bitstream to flash.
    - [Optional] Raw Programming Data (RPD) File: contains the configuration bitstream in plain binary format. Can be written to flash with a 3rd party programmer, such as U-Boot.
    - [Optional] Map File: describes the actual flash usage in human-readable text format.
4. Use the Intel Quartus Prime Programmer to write the JIC image to QSPI flash. Alternatively, use a 3rd party programmer to write the RPD image to flash.
5. Set MSEL to QSPI, then power up, power cycle or toggle `nCONFIG` to cause the device to configure itself from QSPI.
6. FPGA device is configured, and after `INIT_DONE` the HPS FSBL is also executed.

4.6.2.1. Creating Configuration Files from Command Line

The following example command creates the QSPI configuration files for FPGA configuration first mode:

```
quartus_pfg -c design.sof design.jic design.rpd design.map \
-o hps_path=fsbl.hex \
-o device=MT25QU128 \
-o flash_loader=1SX280LU2 \
-o mode=ASX4 \
-o bitswap=on
```

The input and output files for this command are:

- Input Files:
  - `design.sof`
  - `fsbl.hex`
- Output Files:
  - `design.jic`
  - `design.rpd`
  - `design.map`

The command parameters are listed below:

Table 10. Command Parameters

Parameter	Description
`hps_path`	Location of HPS FSBL file in hex format
`device`	Target QSPI device. Use a device listed in *Supported QSPI Devices* or use the graphical interface to determine available options.
`flash_loader`	Which helper image to be used for writing JIC to flash. It is typically a prefix of your FPGA part number. Use the graphical interface mode to determine available options.
`mode`	ASX4 for QSPI
`bitswap`	Set to "on" to create RPD with plain binary format, usable by 3rd party tools.

Note: If your design is small, you can target a QSPI device that is smaller than what you have on board. When programming the resulted JIC file, a warning is displayed, but the resulted file size and erasing and programming times are reduced accordingly.

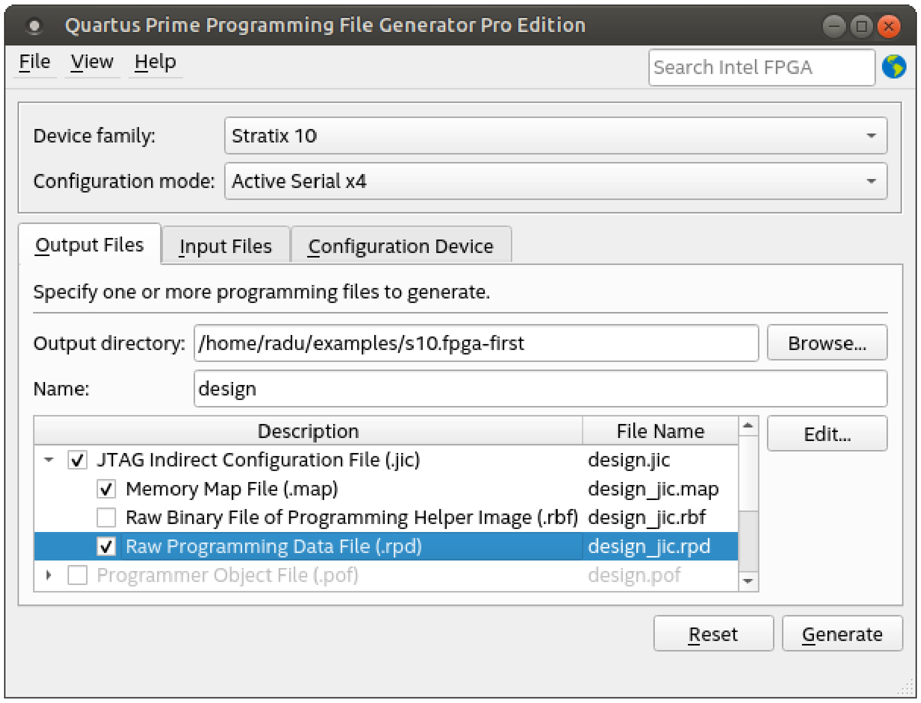
4.6.2.2. Creating Configuration Files Using Graphical Interface

The following example creates the QSPI configuration files for FPGA configuration first mode using the Programming File Generator in GUI mode:

1. Start the Programming File Generator in GUI mode by running the `qpfgw` command.
2. Select the **Device Family** to be Intel Stratix 10.
3. Select the **Configuration mode** to be **Active Serial x4**.
4. In the **Output Files** tab:
   a. Change the output file **Name** to "design"
   b. Check **JTAG Indirect Configuration File (.jic)** option – the others are grayed out.
   c. Optionally check the **Memory Map File (.map)** sub-option
   d. Optionally check the **Raw Programming Data (.rpd)** sub-option.
   e. Click the **Raw Programming Data (.rpd)** sub-option (if checked above) then click on **Edit** button and select the **Bit swap** option to be "on". This ensures the file uses the natural byte format that can be used by 3rd party tools like U-Boot.

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 15. Intel Quartus Prime Programming File Generator Pro Edition Window: Output Files


Quartus Prime Programming File Generator Pro Edition

Device family: Stratix 10
Configuration mode: Active Serial x4

Output Files | Input Files | Configuration Device

Specify one or more programming files to generate.

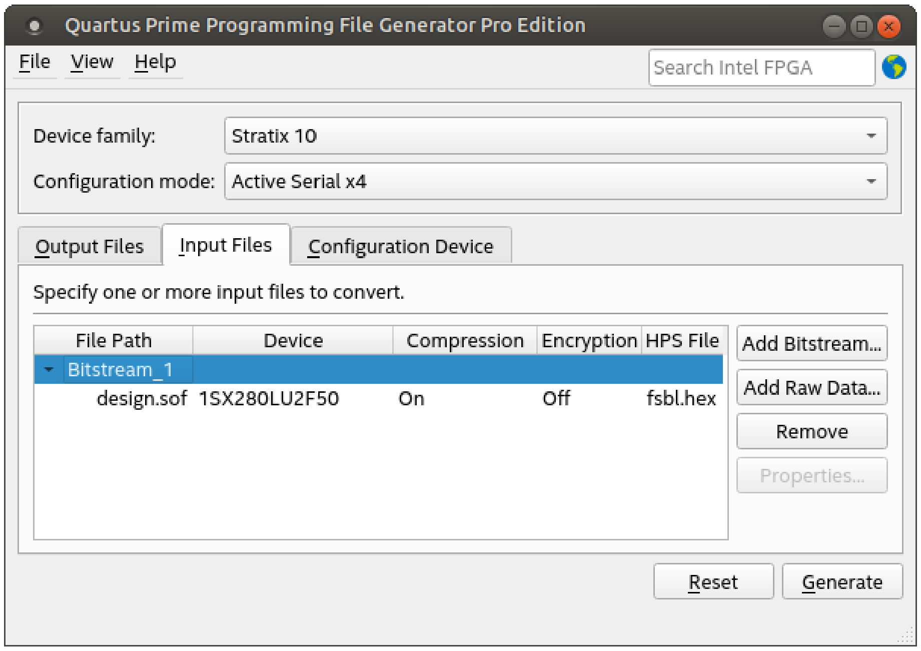
Output directory: /home/radu/examples/s10.fpga-first
Name: design

Description	File Name
☑ JTAG Indirect Configuration File (.jic)	design.jic
☑ Memory Map File (.map)	design_jic.map
☐ Raw Binary File of Programming Helper Image (.rbf)	design_jic.rbf
☑ Raw Programming Data File (.rpd)	design_jic.rpd
☐ Programmer Object File (.pof)	design.pof

5. Switch to **Input Files** tab by clicking it. In the **Input Files** tab, do the following:
   a. Click the **Add Bitstream** button, browse to your SOF file, then click **Open**.
   b. Click the newly added `design.sof` file, then click **Properties**. In the **HPS settings ➤ Bootloader** section, click the **".."** browse button, go to the location of your HPS FSBL hex file, select it and click **Open**.

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 16. Intel Quartus Prime Programming File Generator Pro Edition Window: Input Files



Device family: Stratix 10

Configuration mode: Active Serial x4

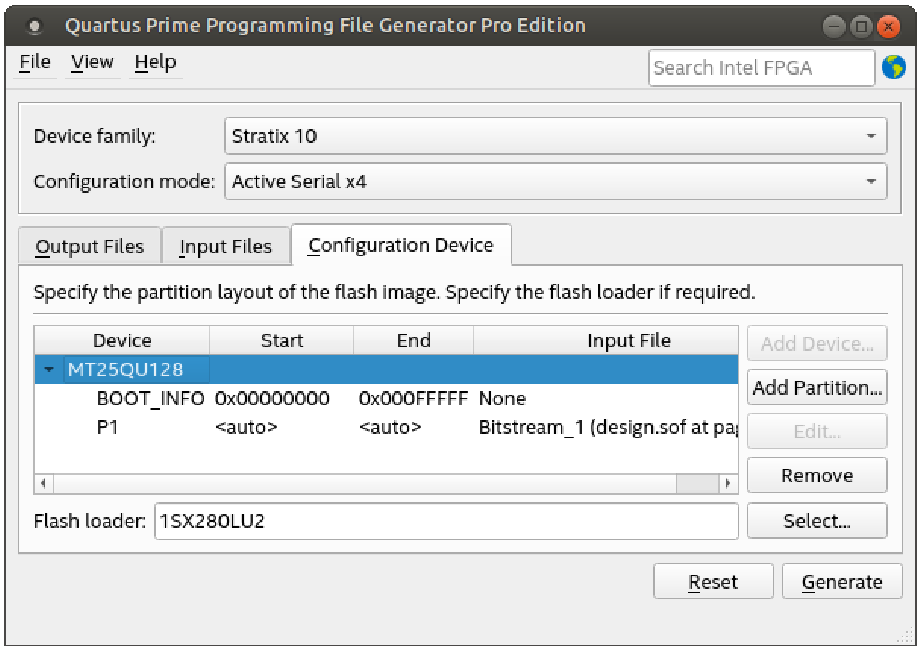
Specify one or more input files to convert.

File Path	Device	Compression	Encryption	HPS File
Bitstream_1				
design.sof	1SX280LU2F50	On	Off	fsbl.hex

6. Switch to the **Configuration Device** tab by clicking it. In the **Configuration Device** tab, do the following:
   a. Click **Add device**, select your desired flash device (in this example **MT25QU128**) then click **OK**
   b. Click the newly added device, then click **Add Partition**. In the partition window, leave **Name** as "P1", **select Input file** to be "Bitstream_1 (design.sof)" and leave **Page** as "0", **Address Mode** as "Auto", then click **OK**.
   c. Under **Flash Loader**, click the **Select..** button, then select **Intel Stratix 10** under **Device Family**, and **1SX280LU2** under **Device Name**. Click **OK**.

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 17. Intel Quartus Prime Programming File Generator Pro Edition Window: Configuration Device



Device family: Stratix 10
Configuration mode: Active Serial x4

Specify the partition layout of the flash image. Specify the flash loader if required.

Device	Start	End	Input File
MT25QU128			
BOOT_INFO	0x00000000	0x000FFFFF	None
P1	<auto>	<auto>	Bitstream_1 (design.sof at pa

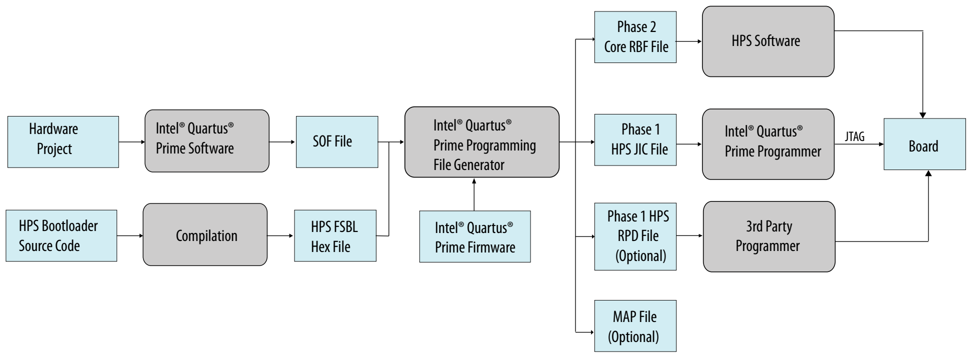
Flash loader: 1SX280LU2

7. Click the **Generate** button. Once the files are generated, a confirmation message is received.
8. Optionally, go to **File ➤ Save** or **File ➤ Save As** to save the configuration in a `.pfg` file. You can generate the output again by applying the same options by running the command line version of the tool like this: "`quartus_pfg -c <filename.pfg>`".

4.6.3. HPS Boot First

In this case, the device reads a small phase 1 bitstream from QSPI flash and uses it to configure the HPS IO, HPS DDR, and bring up the HPS software. Then later the HPS software can configure the FPGA fabric using the typically much larger phase 2 configuration bitstream. The following figure shows an overview of the process:

Figure 18. Configuration from QSPI using HPS Boot First


Hardware Project
Intel® Quartus® Prime Software
SOF File
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Programming File Generator
Intel® Quartus® Prime Firmware
Phase 2 Core RBF File
Phase 1 HPS JIC File
Phase 1 HPS RPD File (Optional)
MAP File (Optional)
HPS Software
Intel® Quartus® Prime Programmer
3rd Party Programmer
JTAG
Board

The following steps are involved:

1. Compile hardware project with Intel Quartus Prime to obtain the SOF file.
2. Compile the HPS FSBL source code to obtain the HPS FSBL hex file, or use a precompiled one.
3. Use Programming File Generator to create the following files:
    - HPS JTAG Indirect Configuration (JIC) file: contains the small phase 1 configuration bitstream and a small SDM helper firmware image used by the Intel Quartus Prime Programmer to write the bitstream to flash.
    - [Optional] HPS Raw Programming Data (RPD) File: contains the small phase 1 configuration bitstream in simple binary format. Can be written to flash with a 3rd party programmer, such as U-Boot.
    - Core RBF file: contains the phase 2 configuration bitstream, to be used by HPS software to configure the FPGA fabric.
    - [Optional] Map file: describes the flash placement and usage in human-readable text format.
4. Use the Intel Quartus Prime Programmer to write the JIC image to QSPI flash. Alternatively use a 3rd party programmer to write the RPD image to flash.
5. Set MSEL to QSPI, then power up, power cycle or toggle nCONFIG to cause the device to configure itself from QSPI.
6. HPS software starts running, beginning with HPS FSBL.
7. Later HPS software configures the FPGA fabric by using the phase 2 Core RBF bitstream.

Note: The phase 1 and phase 2 configuration bitstreams must be created by the exact same Intel Quartus Prime Programming File Generator version, including the same firmware patches. Also, the phase 1 and phase 2 configuration bitstreams must have the same HPS IO settings, including the HPS DDR settings. If these conditions are not both met, the phase 2 configuration fails.

4.6.3.1. Creating Configuration Files from Command Line

The following example creates the QSPI configuration files for HPS boot first mode:

```
quartus_pfg -c design.sof design.jic design.rpd design.map \
-o hps_path=fsbl.hex \
-o device=MT25QU128 \
-o flash_loader=1SX280LU2 \
-o mode=ASX4 \
-o hps=on \
-o bitswap=on
```

The input and output files for this command are:

- Input Files:
  - `design.sof`
  - `fsbl.hex`
- Output Files:
  - `design.hps.jic`
  - `design.core.rbf`
  - `design.rpd` (optional)
  - `design.map` (optional)

The command parameters are listed below:

Table 11. Command Parameters

Parameter	Description
`hps_path`	Location of HPS FSBL file in hex format
`device`	Target QSPI device. Use a device listed in *Supported QSPI Devices* or use the graphical interface to determine available options.
`flash_loader`	Which helper image to be used for writing JIC to flash. It is typically a prefix of your FPGA part number. Use the graphical interface mode to determine available options.
`mode`	ASX4 for QSPI
`bitswap`	Set to "on" to create RPD with plain binary format, usable by 3rd party tools.
`hps`	Set to "on" to enable HPS boot first mode, omit for FPGA configuration first mode

Note: When using HPS boot first your JIC is small, and you can target a QSPI device that is smaller than what you have on board. When programming the resulted JIC file, a warning is displayed, but the resulted file size and erasing and programming times are reduced accordingly.

4.6.3.2. Creating Configuration Files Using Graphical Interface

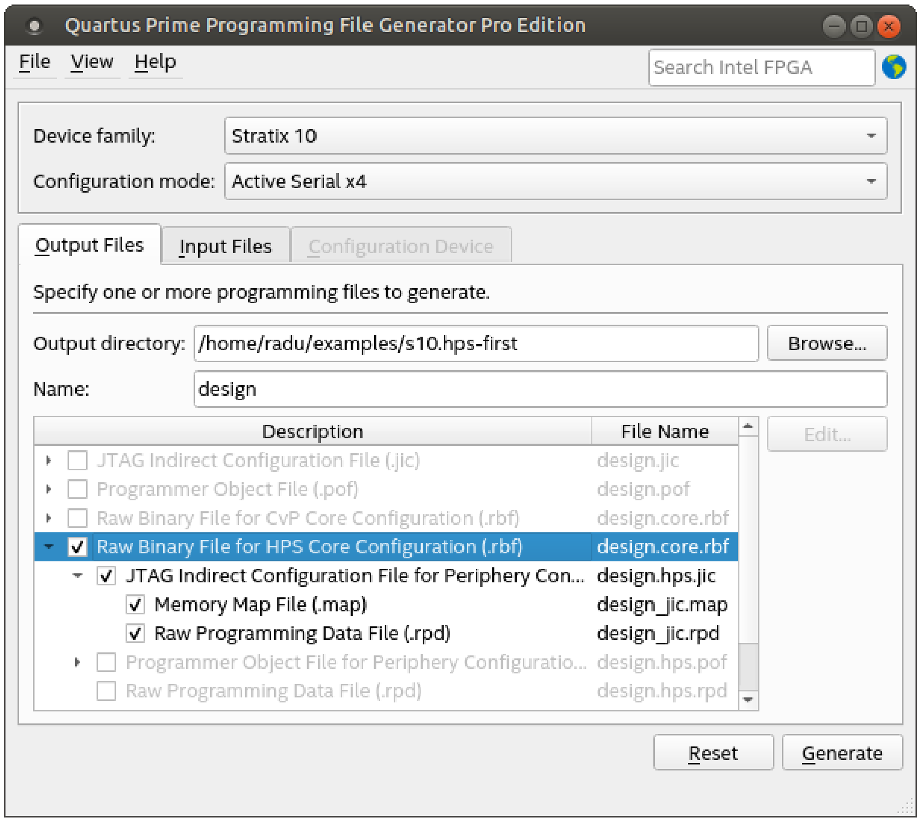
The following example creates the QSPI configuration files for HPS boot first mode using the Programming File Generator in GUI mode:

1. Start the Programming File Generator in GUI mode by running the `qpfgw` command.
2. Select the **Device Family** to be Intel Stratix 10.
3. Select the **Configuration mode** to be **Active Serial x4**.
4. In the **Output Files** tab:
   a. Change the output file **Name** to “design”.
   b. Check **Raw Binary File for HPS Core Configuration (.rbf)** option – the others are grayed out.
   c. Check the **JTAG Indirect Configuration File for Periphery Configuration (.jic)** sub-option

d. Optionally check the **Memory Map File (.map)** sub-option
e. Optionally check the **Raw Programming Data (.rpd)** sub-option.
f. Click the **Raw Programming Data (.rpd)** sub-option (if checked above) then click on **Edit** button and select the **Bit swap** option to be "on". This ensures the file uses the natural byte format that can be used by 3rd party tools like U-Boot.

The Intel Quartus Prime Programming File Generator window is displayed:

Figure 19. Intel Quartus Prime Programming File Generator Pro Edition Window: Output Files



Device family: Stratix 10
Configuration mode: Active Serial x4

Specify one or more programming files to generate.

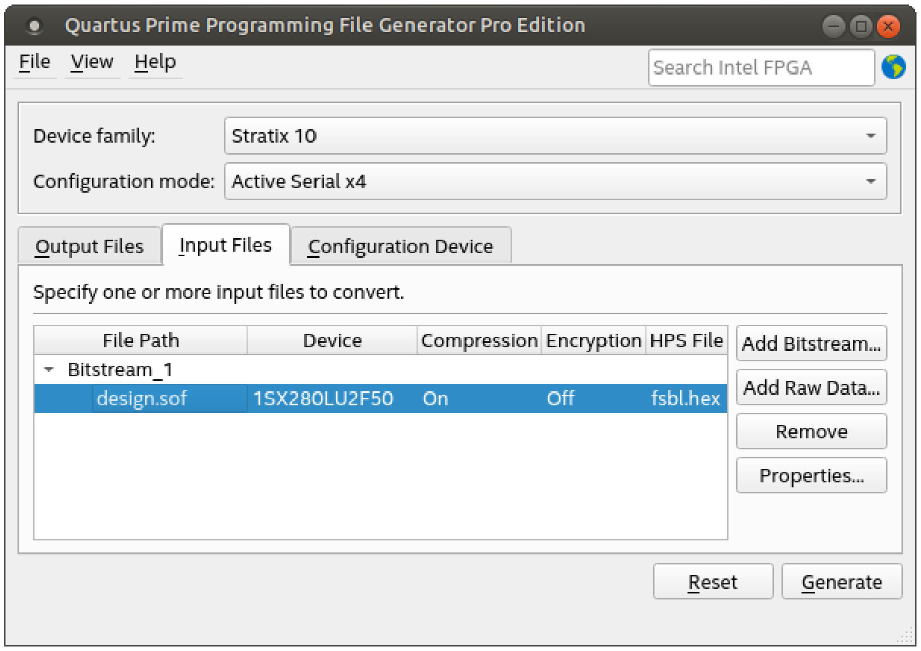
Output directory: /home/radu/examples/s10.hps-first
Name: design

Description	File Name
☐ JTAG Indirect Configuration File (.jic)	design.jic
☐ Programmer Object File (.pof)	design.pof
☐ Raw Binary File for CvP Core Configuration (.rbf)	design.core.rbf
✓ Raw Binary File for HPS Core Configuration (.rbf)	design.core.rbf
✓ JTAG Indirect Configuration File for Periphery Con...	design.hps.jic
✓ Memory Map File (.map)	design_jic.map
✓ Raw Programming Data File (.rpd)	design_jic.rpd
☐ Programmer Object File for Periphery Configuratio...	design.hps.pof
☐ Raw Programming Data File (.rpd)	design.hps.rpd

5. Switch to **Input Files** tab by clicking it. In the **Input Files** tab, do the following:
   a. Click the **Add Bitstream** button, browse to your SOF file, then click **Open**.
   b. Click the newly added `design.sof` file, then click **Properties**. In the **HPS settings ➤ Bootloader** section, click the **".."** browse button, go to the location of your HPS FSBL hex file, select it and click **Open**.

The Intel Quartus Prime Programming File Generator window is displayed:

Figure 20. Intel Quartus Prime Programming File Generator Pro Edition Window: Input Files



Device family: Stratix 10
Configuration mode: Active Serial x4

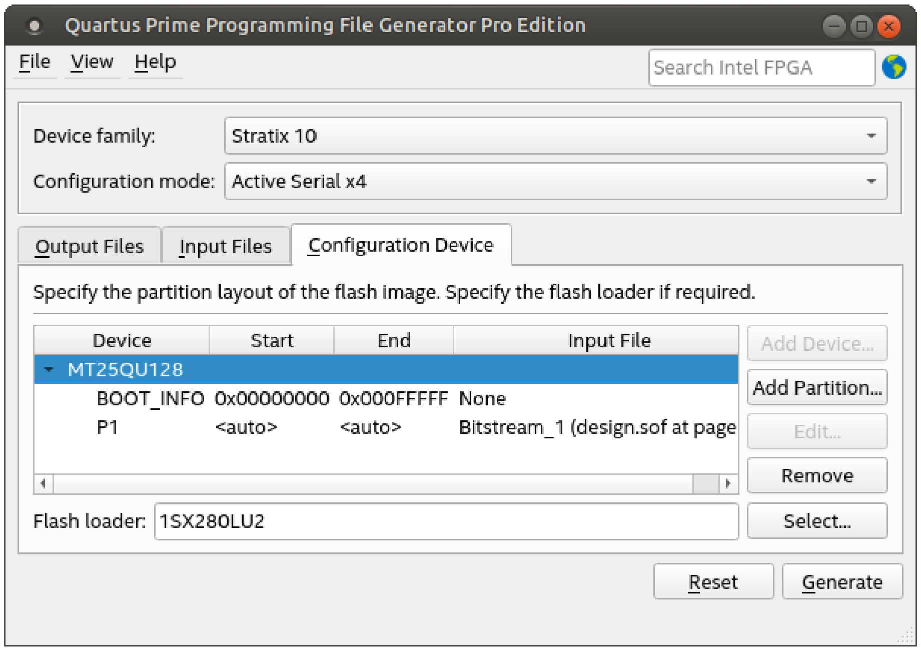
Specify one or more input files to convert.

File Path	Device	Compression	Encryption	HPS File
Bitstream_1				
design.sof	1SX280LU2F50	On	Off	fsbl.hex

6. Switch to the **Configuration Device** tab by clicking it. In the **Configuration Device** tab, do the following:
   a. Click **Add device**, select your desired flash device (in this example **MT25QU128**) then click **OK**
   b. Click the newly added device, then click **Add Partition**. In the partition window, leave **Name** as "P1", **select Input file** to be "Bitstream_1 (design.sof)" and leave **Page** as "0", **Address Mode** as "Auto", then click **OK**.
   c. Under **Flash Loader**, click the **Select..** button, then select **Intel Stratix 10** under **Device Family**, and **1SX280LU2** under **Device Name**. Click **OK**.

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 21. Intel Quartus Prime Programming File Generator Pro Edition Window: Configuration Device



Device family: Stratix 10

Configuration mode: Active Serial x4

Specify the partition layout of the flash image. Specify the flash loader if required.

Device	Start	End	Input File
MT25QU128			
BOOT_INFO	0x00000000	0x000FFFFF	None
P1	<auto>	<auto>	Bitstream_1 (design.sof at page

Flash loader: 1SX280LU2

7. Click the **Generate** button. Once the files are generated, a confirmation message is received.

8. Optionally, go to **File ➤ Save** or **File ➤ Save As** to save the configuration in a `.pfg` file. You can generate the output again by applying the same options by running the command line version of the tool like this: "`quartus_pfg -c <filename.pfg>`".

4.7. Configuration over AVST

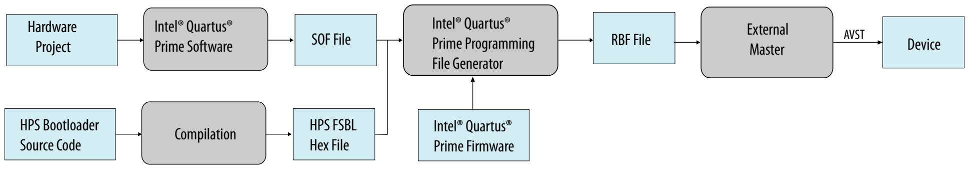
An external master sends the configuration data to the device over an Avalon Streaming Interface bus. There are three different supported widths for the Avalon Streaming Interface bus: AVST x8, AVST x16 and AVST x32.

Note: On Configuration over AVST, the QSPI access from the HPS software is not supported and the QSPI component must be disabled in both the bootloader (for example: U-Boot configuration and device tree) and the OS (for example: Linux device tree).

4.7.1. FPGA Configuration First

The following figure shows an overview of the process:

Figure 22. Configuration over Avalon Streaming Using FPGA Configuration First


Hardware Project
Intel® Quartus® Prime Software
SOF File
Intel® Quartus® Prime Programming File Generator
RBF File
External Master
AVST
Device
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Firmware

The following steps are involved:

1. Compile hardware project with Intel Quartus Prime to obtain the SOF file.
2. Compile the HPS FSBL source code to obtain the HPS FSBL hex file, or use a precompiled file.
3. Use Programming File Generator to create the following files:

   Raw Binary File (RBF): contains the configuration bitstream in binary format.
4. Set MSEL to the AVST mode.
5. Power up, power cycle or toggle `nCONFIG` on the device.
6. Use an external master connected over AVST to configure the device using the RBF File.

4.7.1.1. Creating Configuration Files from Command Line

The following example creates the RBF file for FPGA configuration first:

```
quartus_pfg -c design.sof design.rbf -o hps_path=fsbl.hex
```

The input and output files for this command are:

- Input Files:
  - `design.sof`
  - `fsbl.hex`
- Output File:
  - `design.rbf`

The command parameter is listed below:

Table 12. Command Parameters

Parameter	Description
`hps_path`	Location of HPS FSBL file in hex format

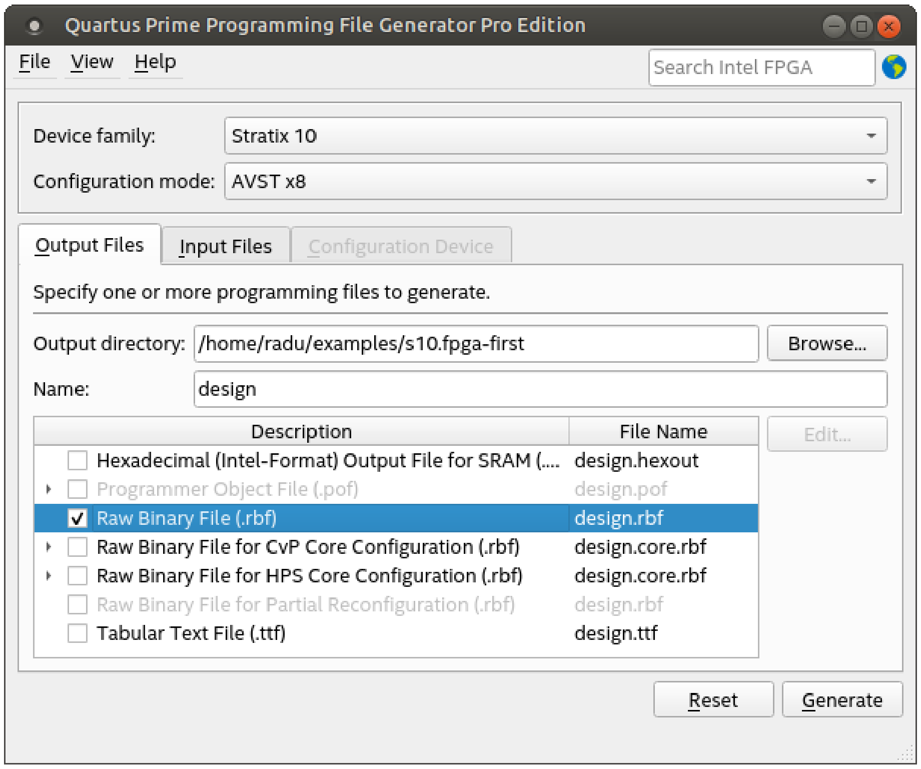
4.7.1.2. Creating Configuration Files Using Graphical Interface

The following example creates the AVST configuration files for FPGA first mode using the Programming File Generator in GUI mode:

1. Start the Programming File Generator in GUI mode by running the `qpfgw` command.
2. Select the **Device Family** to be Intel Stratix 10.
3. Select the **Configuration mode** to be **AVST x8**, **AVST x16** or **AVST x32**.
4. In the **Output Files** tab:
   a. Change the output file **Name** to "design".
   b. Check **Raw Binary File (.rbf)** option – the others are grayed out.

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 23. Intel Quartus Prime Programming File Generator Pro Edition Window: Output Files


Quartus Prime Programming File Generator Pro Edition

Device family: Stratix 10
Configuration mode: AVST x8

Output Files | Input Files | Configuration Device

Specify one or more programming files to generate.

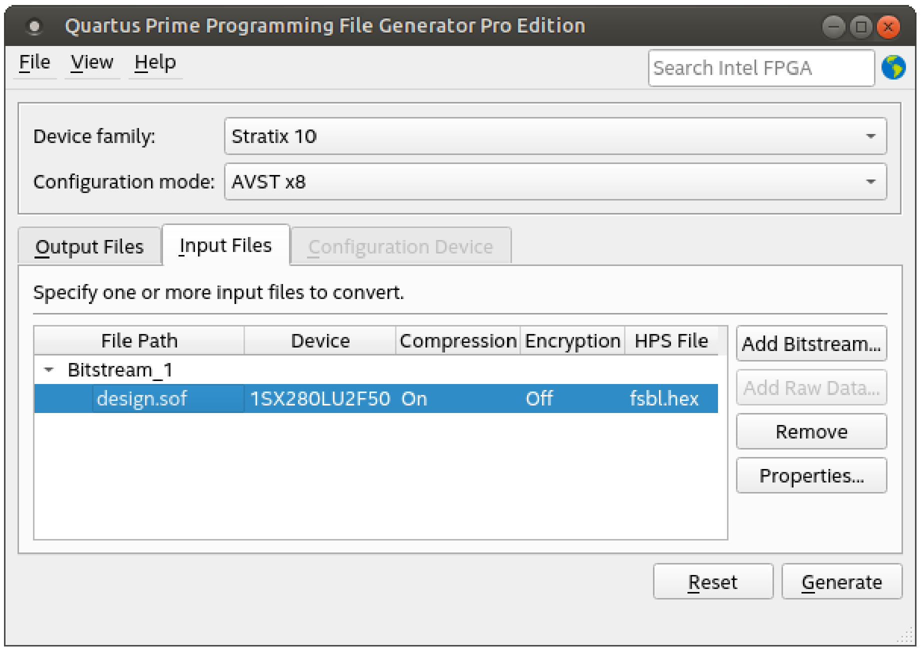
Output directory: /home/radu/examples/s10.fpga-first
Name: design

	Description	File Name
☐	Hexadecimal (Intel-Format) Output File for SRAM (....	design.hexout
☐	Programmer Object File (.pof)	design.pof
✓	Raw Binary File (.rbf)	design.rbf
☐	Raw Binary File for CvP Core Configuration (.rbf)	design.core.rbf
☐	Raw Binary File for HPS Core Configuration (.rbf)	design.core.rbf
☐	Raw Binary File for Partial Reconfiguration (.rbf)	design.rbf
☐	Tabular Text File (.ttf)	design.ttf

5. Switch to **Input Files** tab by clicking it. In the **Input Files** tab, do the following:
   a. Click the **Add Bitstream** button, browse to your SOF file, then click **Open**.
   b. Click the newly added `design.sof` file, then click **Properties**. In the **HPS settings ➤ Bootloader** section, click the **".."** browse button, go to the location of your HPS FSBL hex file, select it and click **Open**.

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 24. Intel Quartus Prime Programming File Generator Pro Edition Window: Input Files



Device family: Stratix 10
Configuration mode: AVST x8

Specify one or more input files to convert.

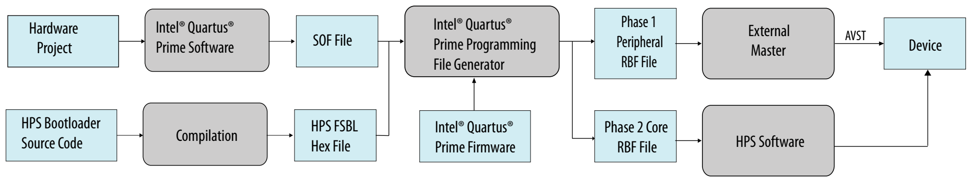
File Path	Device	Compression	Encryption	HPS File
Bitstream_1				
design.sof	1SX280LU2F50	On	Off	fsbl.hex

6. Click the **Generate** button. Once the files are generated, a confirmation message is received.
7. Optionally, go to **File ➤ Save** or **File ➤ Save As** to save the configuration in a `.pfg` file. You can generate the output again by applying the same options by running the command line version of the tool like this: `quartus_pfg -c <filename.pfg>`.

4.7.2. HPS Boot First

In this case, the device gets first configured with a small phase 1 bitstream from QSPI flash, the HPS IO, HPS DDR are configured and HPS starts running the FSBL. Then at a later time the HPS software can configure the FPGA fabric using the typically much larger phase 2 configuration bitstream. The following figure shows an overview of the process:

Figure 25. Configuration over Avalon Streaming Using HPS Boot First


Hardware Project
Intel® Quartus® Prime Software
SOF File
Intel® Quartus® Prime Programming File Generator
Phase 1 Peripheral RBF File
External Master
AVST
Device
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Firmware
Phase 2 Core RBF File
HPS Software

1. Compile hardware project with Intel Quartus Prime to obtain the SOF file.
2. Compile the HPS FSBL source code to obtain the HPS FSBL hex file, or use a precompiled one.
3. Use Programming File Generator to create the following files:

    - Raw Binary File (RBF): contains the small phase 1 configuration bitstream.
    - Core RBF File: contains the typically much larger phase 2 configuration bitstream, to be used by HPS software later to configure the fabric.

   a. At a later time HPS software configures the FPGA fabric by using the phase 2 Core RBF bitstream.

4. Set MSEL to the AVST mode.
5. Power up, power cycle or toggle `nCONFIG` on the device.
6. Use an external master connected over AVST to configure the device using the phase 1 bitstream.

   HPS software starts running, beginning with HPS FSBL.

   At a later time HPS software configures the FPGA fabric by using the phase 2 Core RBF bitstream.

4.7.2.1. Creating Configuration Files from Command Line

The following example creates the RBF file for HPS boot first:

```
quartus_pfg -c design.sof design.rbf -o hps_path=fsbl.hex -o hps=on
```

The input and output files for this command are:

- Input Files:
  — `design.sof`
  — `fsbl.hex`
- Output Files:
  — `design.hps.rbf`
  — `design.core.rbf`

The command parameter is listed below:

Table 13. Command Parameters

Parameter	Description
`hps_path`	Location of HPS FSBL file in hex format
`hps`	Set to "on" to enable HPS boot first mode, omit for FPGA configuration first mode

4.7.2.2. Creating Configuration Files Using Graphical Interface

The following example creates the AVST configuration files for HPS boot first mode using the Programming File Generator in GUI mode:

1. Start the Programming File Generator mode by running the `qpfgw` command.
2. Select the **Device Family** to be Intel Stratix 10.
3. Select the **Configuration mode** to be **AVST x8**, **AVST x16** or **AVST x32**.
4. In the **Output Files** tab:

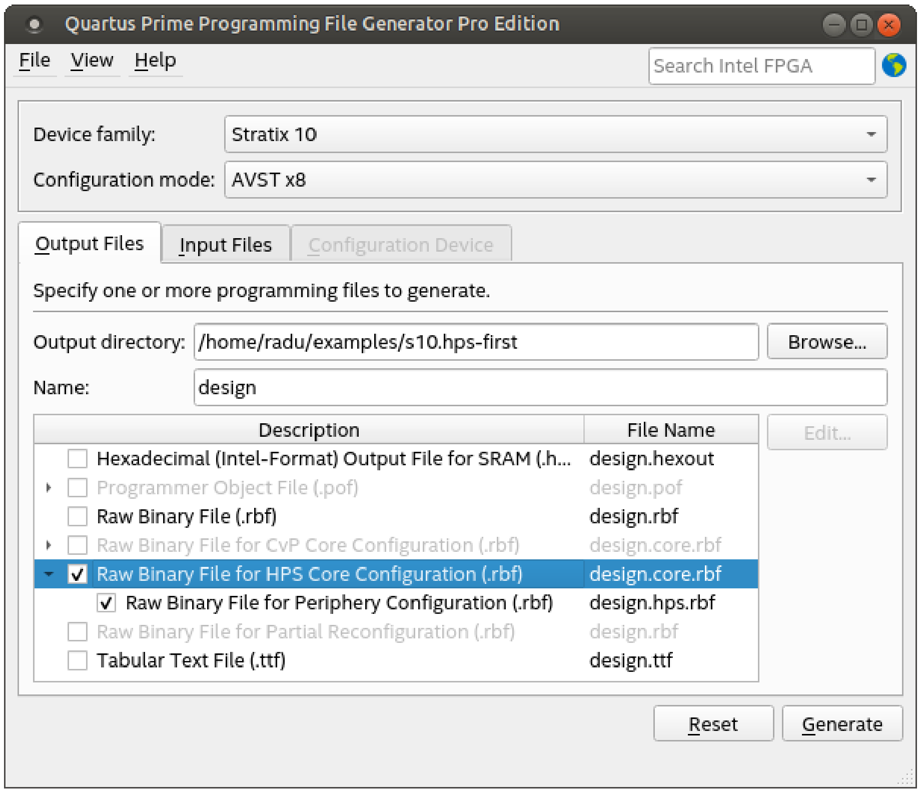
a. Change the output file **Name** to “design”.

b. Check **Raw Binary File for HPS Core Configuration (.rbf)** option – the others are grayed out.

c. Check **Raw Binary File for Periphery Configuration (.rbf)** sub-option.

The Intel Quartus Prime Programming File Generator window is displayed:

Figure 26. Intel Quartus Prime Programming File Generator Pro Edition Window: Output Files



Device family: Stratix 10
Configuration mode: AVST x8

Specify one or more programming files to generate.

Output directory: /home/radu/examples/s10.hps-first
Name: design

	Description	File Name
☐	Hexadecimal (Intel-Format) Output File for SRAM (.h...	design.hexout
☐	Programmer Object File (.pof)	design.pof
☐	Raw Binary File (.rbf)	design.rbf
☐	Raw Binary File for CvP Core Configuration (.rbf)	design.core.rbf
✓	Raw Binary File for HPS Core Configuration (.rbf)	design.core.rbf
✓	Raw Binary File for Periphery Configuration (.rbf)	design.hps.rbf
☐	Raw Binary File for Partial Reconfiguration (.rbf)	design.rbf
☐	Tabular Text File (.ttf)	design.ttf

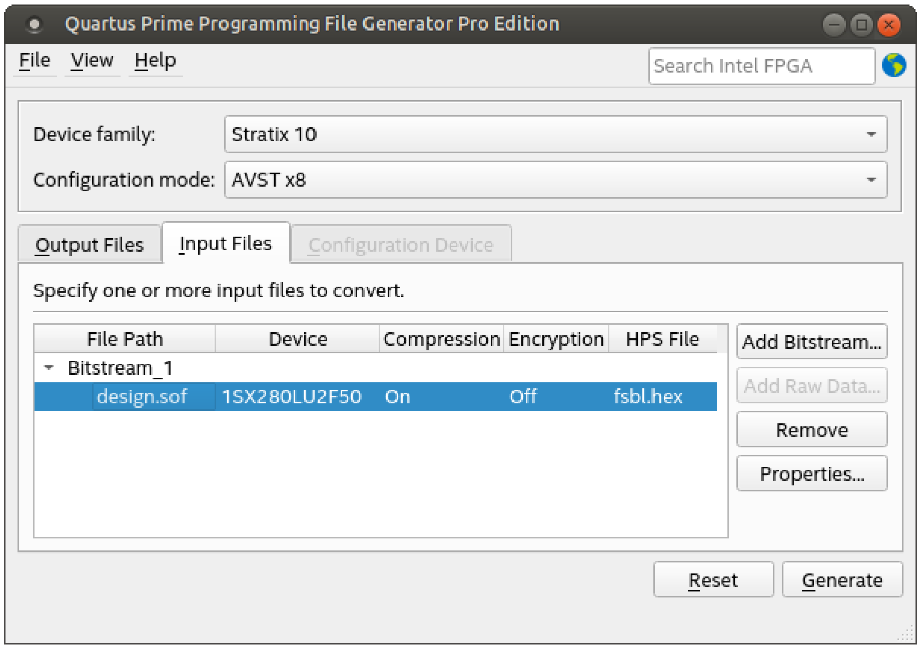
5. Switch to **Input Files** tab by clicking it. In the **Input Files** tab, do the following:

a. Click the **Add Bitstream** button, browse to your SOF file, then click **Open**.

b. Click the newly added `design.sof` file, then click **Properties**. In the **HPS settings ➤ Bootloader** section, click the **“..”** browse button, go to the location of your HPS FSBL hex file, select it and click **Open**.

The Intel Quartus Prime Programming File Generator window is displayed:

Figure 27. Intel Quartus Prime Programming File Generator Pro Edition Window: Input Files



Device family: Stratix 10
Configuration mode: AVST x8

Specify one or more input files to convert.

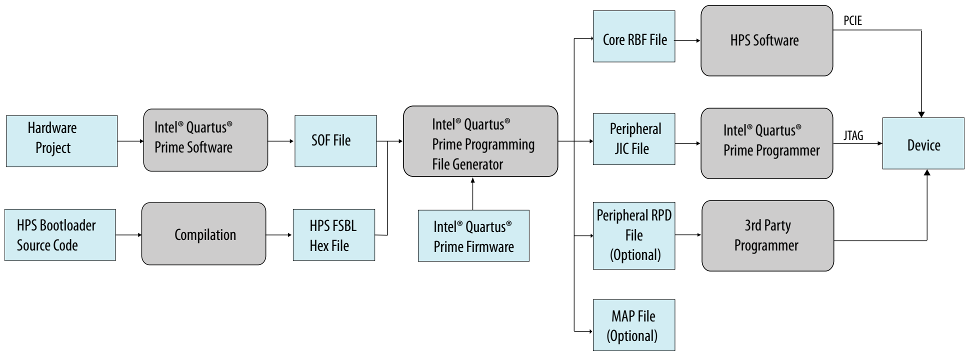
File Path	Device	Compression	Encryption	HPS File
Bitstream_1				
design.sof	1SX280LU2F50	On	Off	fsbl.hex

6. Click the **Generate** button. Once the files are generated, a confirmation message is received.
7. Optionally, go to **File ➤ Save** or **File ➤ Save As** to save the configuration in a `.pfg` file. You can generate the output again by applying the same options by running the command line version of the tool like this: “`quartus_pfg -c <filename.pfg>`”.

4.8. Configuration via Protocol

In the Configuration via Protocol (CvP) case, a small QSPI flash image is configured first, which brings up the PCIe interface quickly. Then, later, the PCIe host computer configures the fabric with the Core RBF file.

Figure 28. Configuration over Protocol


Hardware Project
Intel® Quartus® Prime Software
SOF File
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Programming File Generator
Intel® Quartus® Prime Firmware
Core RBF File
Peripheral JIC File
Peripheral RPD File (Optional)
MAP File (Optional)
HPS Software
Intel® Quartus® Prime Programmer
3rd Party Programmer
PCIE
JTAG
Device

The following steps are involved:

1. Compile hardware project with Intel Quartus Prime to obtain the SOF file.
2. Compile the HPS FSBL source code to obtain the HPS FSBL hex file, or use a precompiled file.
3. Use Programming File Generator to create the following files:
   - Peripheral JIC File: contains the initial configuration bitstream (including peripheral configuration data and SDM firmware) and a small SDM helper firmware image used by the Intel Quartus Prime Programmer to write the bitstream to flash.
   - [Optional] Peripheral RPD File: contains the same initial configuration bitstream as the Peripheral JIC file, in simple binary format. Can be written to flash with a 3rd party programmer, such as U-Boot.
   - Core RBF File: contains the FPGA configuration data, to be used by PCIe host software later to configure the fabric. The HPS FSBL is included in the Core RBF file.
   - [Optional] Map File: describes the actual flash usage in human-readable text format.
4. The FPGA device is configured from the initial peripheral bitstream from QSPI flash, which brings up the PCIe interface.
5. The PCIe host later configures the core fabric over PCIe. This includes downloading and running HPS FSBL.

Note: When using CvP, only the FPGA configuration first mode is supported.

4.8.1. Creating Configuration Files from Command Line

The following example creates the configuration files for CvP:

```
quartus_pfg -c design.sof design.jic design.rpd design.map \
-o hps_path=fsbl.hex \
-o device=MT25QU128 \
-o flash_loader=1SX280LU2 \
-o mode=ASX4 \
-o bitswap=on \
-o cvp=on
```

The input and output files for this command are:

- Input Files
  — `design.sof`
  — `fsbl.hex`
- Output Files:
  — `design.periph.jic`
  — `design.core.rbf`
  — `design.rpd` (optional)
  — `design.map` (optional)

The command parameters are listed below:

Table 14. Command Parameters

Parameter	Description
`hps_path`	Location of HPS FSBL file in hex format
`device`	Target QSPI device. Use a device listed in *Supported QSPI Devices* or use the graphical interface to determine available options.
`flash_loader`	Which helper image to be used for writing JIC to flash. It is typically a prefix of your FPGA part number. Use the graphical interface mode to determine available options.
`mode`	ASX4 for QSPI
`bitswap`	Set to "on" to create RPD with plain binary format, usable by 3rd party tools.
`cvp`	Set to "on" to enable Configuration via Protocol.

Note: When using HPS boot first your JIC is small, and you can target a QSPI device that is smaller than what you have on board. When programming the resulted JIC file, a warning is displayed, but the resulted file size and erasing and programming times are reduced accordingly.

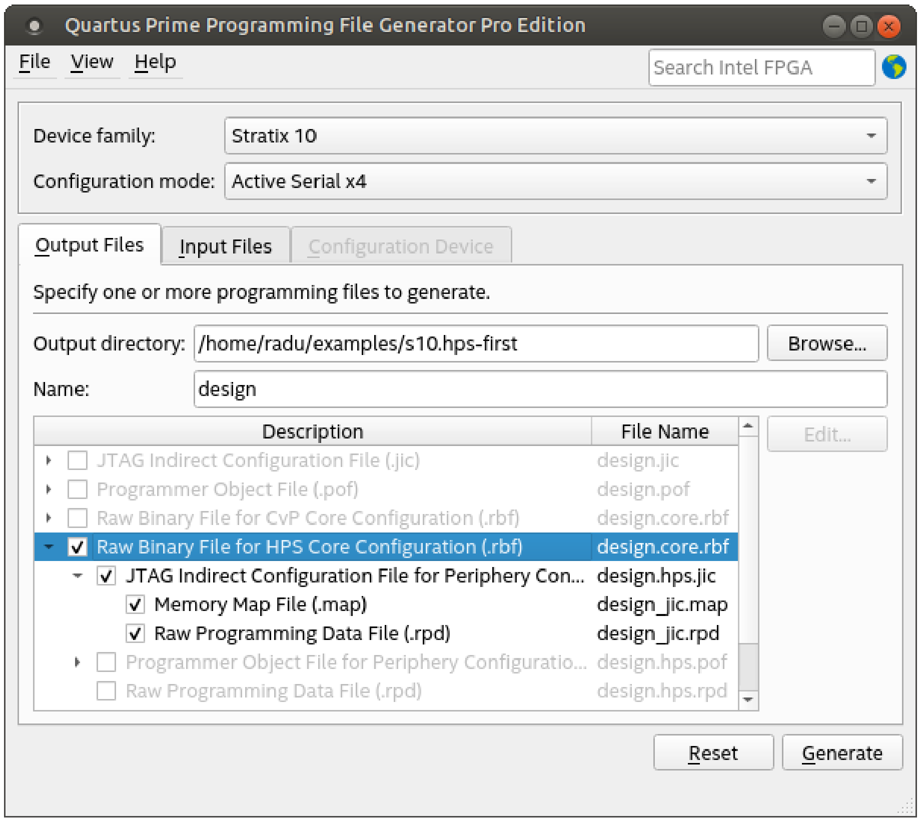
4.8.2. Creating Configuration Files Using Graphical Interface

The following example creates the CvP configuration files using the Programming File Generator in GUI mode:

1. Start the Programming File Generator in GUI mode by running the `qpfgw` command.
2. Select the **Device Family** to be Intel Stratix 10.
3. Select the **Configuration mode** to be **Active Serial x4**.
4. In the **Output Files** tab:
   a. Change the output file **Name** to “design”.
   b. Check **Raw Binary File for CvP Core Configuration (.rbf)** option – the others are grayed out.
   c. Check the **JTAG Indirect Configuration File for Periphery Configuration (.jic)** sub-option
   d. Check the **Memory Map File (.map)** sub-option
   e. Optionally check the **Raw Programming Data (.rpd)** sub-option.
   f. Click the **Raw Programming Data (.rpd)** sub-option

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 29. Intel Quartus Prime Programming File Generator Pro Edition Window: Output Files


Quartus Prime Programming File Generator Pro Edition

File View Help

Device family: Stratix 10
Configuration mode: Active Serial x4

Output Files | Input Files | Configuration Device

Specify one or more programming files to generate.

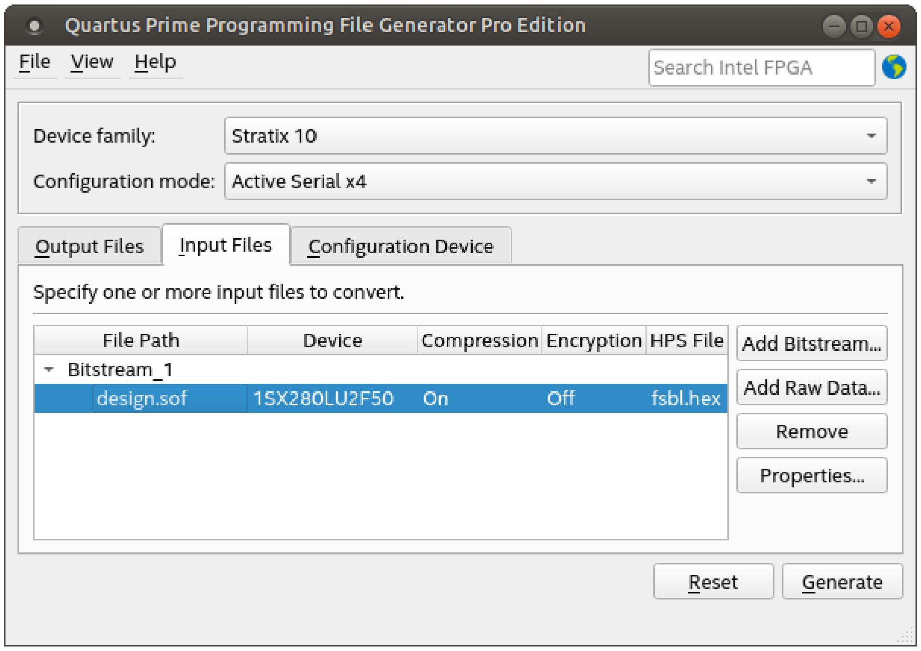
Output directory: /home/radu/examples/s10.hps-first
Name: design

Description	File Name
☐ JTAG Indirect Configuration File (.jic)	design.jic
☐ Programmer Object File (.pof)	design.pof
☐ Raw Binary File for CvP Core Configuration (.rbf)	design.core.rbf
✓ Raw Binary File for HPS Core Configuration (.rbf)	design.core.rbf
✓ JTAG Indirect Configuration File for Periphery Con...	design.hps.jic
✓ Memory Map File (.map)	design_jic.map
✓ Raw Programming Data File (.rpd)	design_jic.rpd
☐ Programmer Object File for Periphery Configuratio...	design.hps.pof
☐ Raw Programming Data File (.rpd)	design.hps.rpd

5. Switch to **Input Files** tab by clicking it. In the **Input Files** tab, do the following:
   a. Click the **Add Bitstream** button, browse to your SOF file, then click **Open**.
   b. Click the newly added `design.sof` file, then click **Properties**. In the **HPS settings ➤ Bootloader** section, click the **".."** browse button, go to the location of your HPS FSBL hex file, select it and click **Open**.

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 30. Intel Quartus Prime Programming File Generator Pro Edition Window: Input Files



Device family: Stratix 10
Configuration mode: Active Serial x4

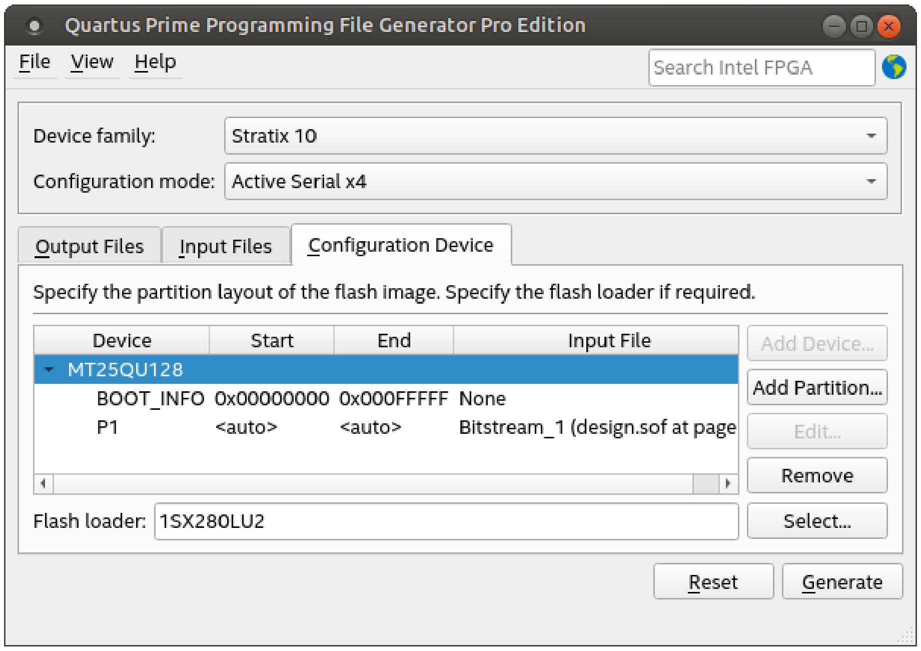
Specify one or more input files to convert.

File Path	Device	Compression	Encryption	HPS File
Bitstream_1				
design.sof	1SX280LU2F50	On	Off	fsbl.hex

6. Switch to the **Configuration Device** tab by clicking it. In the **Configuration Device** tab, do the following:
   a. Click **Add device**, select your desired flash device (in this example **MT25QU128**) then click **OK**
   b. Click the newly added device, then click **Add Partition**. In the partition window, leave **Name** as "P1", **select Input file** to be "Bitstream_1 (design.sof)" and leave **Page** as "0", **Address Mode** as "Auto", then click **OK**.
   c. Under **Flash Loader**, click the **Select..** button, then select **Intel Stratix 10** under **Device Family**, and **1SX280LU2** under **Device Name**. Click **OK**.

   The Intel Quartus Prime Programming File Generator window is displayed:

Figure 31. Intel Quartus Prime Programming File Generator Pro Edition Window: Configuration Device



Device family: Stratix 10

Configuration mode: Active Serial x4

Specify the partition layout of the flash image. Specify the flash loader if required.

Device	Start	End	Input File
MT25QU128			
BOOT_INFO	0x00000000	0x000FFFFF	None
P1	<auto>	<auto>	Bitstream_1 (design.sof at page

Flash loader: 1SX280LU2

7. Click the **Generate** button. Once the files are generated, a confirmation message is received.
8. Optionally, go to **File ➤ Save** or **File ➤ Save As** to save the configuration in a `.pfg` file. You can generate the output again by applying the same options by running the command line version of the tool like this: "`quartus_pfg -c <filename.pfg>`".

4.9. Remote System Update

In the Remote System Update (RSU) case, the FPGA is configured from QSPI flash, using several application images and a factory image. First each application image is tried in order of priority, until one of them is successfully configured to the device. If no application image succeeds in configuring, the factory image is tried last. Procedures are provided for updating application image and the factory image in a safe way, so that the system integrity is not compromised. This is very important for devices deployed in the field, especially when there are a lot of them, or they are deployed in hard to reach places.

For more information about RSU, refer to the *Intel Stratix 10 Hard Processor System Remote System Update User Guide*.

Related Information

Intel Stratix 10 Hard Processor System Remote System Update (RSU) User Guide

4.10. Partial Reconfiguration

Partial reconfiguration (PR) allows you to reconfigure a portion of the FPGA dynamically, while the rest of the FPGA design continues to function. The portion that can be reconfigured is referred to as **PR region** while the FPGA design that is to be configured in the PR region is referred to as **PR persona**. All areas not occupied by PR regions in your project is referred to as **static region**. You can associate the static region with the top-level partition of the design. The static region contains both the core and periphery locations of the device. The static region bitstream configures this region.

You can define multiple PR personas for a particular region in your design (for example: PR persona0, PR persona1, etc.), without impacting operation in areas outside the region. This methodology is effective in systems with multiple functions that time-share the same FPGA device resources. PR enables the implementation of more complex FPGA systems.

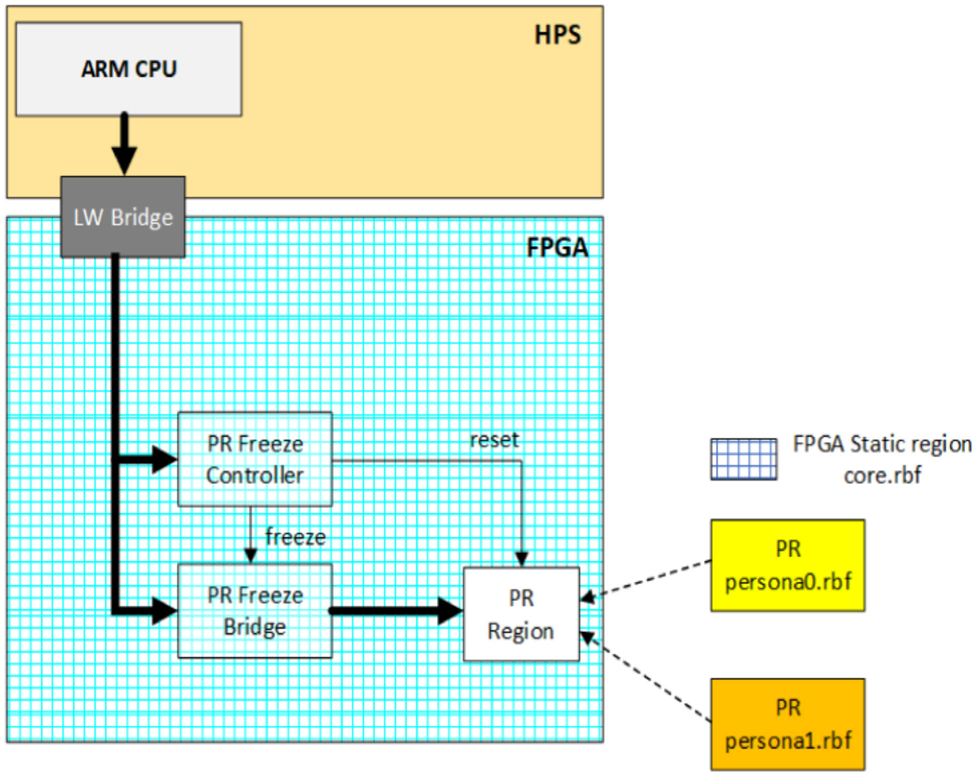
When PR is used, the FPGA hardware design needs to be built enabling PR support. For this, the following components are needed as part of the static FPGA region:

- **PR Freeze Bridge**—Before performing the PR, all the interfaces between the PR region and the rest of the FPGA must be isolated (frozen). This module is used for this purpose. Depending on the design in the PR, more than one bridge may be needed.
- **PR Freeze Controller**—This handles the freezing mechanism. Before PR is performed, this module freezes all the PR bridges and after PR is done, this unfreezes the interfaces and takes the PR region out of reset.

In use cases where partial reconfiguration is controlled by the HPS, the PR process is initiated by the HPS software sending the PR bitstream (`persona .rbf` file) to the SDM. This can be done from Linux using the device tree overlay mechanism described in the *FPGA Partial Reconfiguration from Linux* section.

The following figure describes an example of the PR concept. This also shows the associated IPs added to the FPGA design and how these are connected to the HPS.

Figure 32. Example of the PR Concept

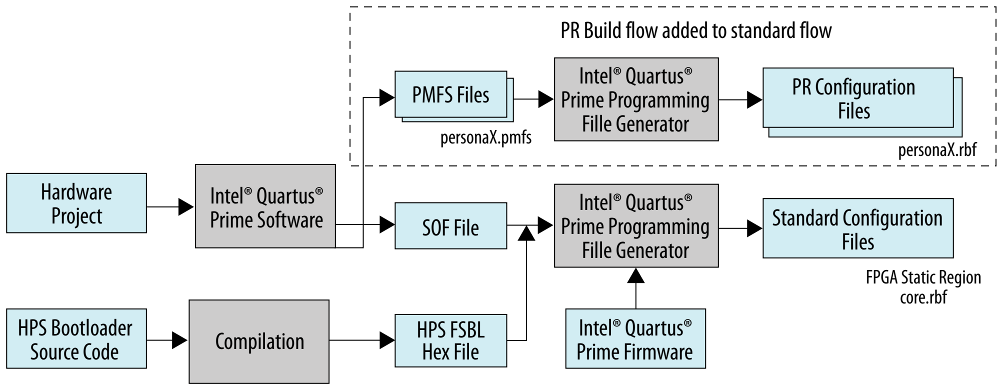


The process to create the RBF file is shown in the following figure and for this, the Intel Quartus Prime Software is used. This flow extends from the standard configuration flow to generate the `.pmsf` files for each one of the persona PR designs which are then converted in to the corresponding `personaX.rbf` file. The PR design for each persona are included as part of the hardware project. The FPGA design to be loaded into the static region is built as part of the standard build flow.

The command used to create the RBF files is shown next:

```
# Command to create personaX.rbf
quartus_pfg -c personaX.pmsf personaX.rbf
```

Figure 33. Process to Create the RBF File


PR Build flow added to standard flow
PMFS Files
personaX.pmfs
Intel® Quartus® Prime Programming Fille Generator
PR Configuration Files
personaX.rbf
Hardware Project
Intel® Quartus® Prime Software
SOF File
Intel® Quartus® Prime Programming Fille Generator
Standard Configuration Files
FPGA Static Region core.rbf
HPS Bootloader Source Code
Compilation
HPS FSBL Hex File
Intel® Quartus® Prime Firmware

Related Information

FPGA Partial Reconfiguration from Linux on page 57

5. Golden System Reference Design and Design Examples

5.1. Golden System Reference Design

The Golden System Reference Design (GSRD) is a thoroughly tested known good design showcasing a system using both HPS and FPGA resources, intended to be used as a baseline project.

The GSRD is comprised of the following components:

- Golden Hardware Reference Design (GHRD)
- Reference HPS software including:
  - Arm Trusted Firmware
  - U-Boot
  - Linux Kernel
  - Linux drivers
  - Sample applications

The current GSRD uses HPS First configuration mode, with U-Boot configuring the FPGA fabric.

The full documentation for the GSRD includes details on both how to use the pre-built binaries and also how to rebuild them if needed. The GSRD includes a complete PR example, including recipes to build the PR binaries and instructions used to exercise different PR scenarios from Linux.

The documentation can be referenced at the following link: Intel Stratix 10 SoC GSRD.

5.2. Bootloader Examples

The Building Bootloader web page on the RocketBoards website contains examples of booting the HPS from all supported flash devices (SD card, Quad SPI Flash, eMMC, NAND). It also contains guides for running the bootloaders from Arm Development Studio* (DS*) Intel SoC FPGA Edition command line debugger and debugging the bootloaders from Arm DS Intel SoC FPGA Edition Eclipse tool.

Related Information

RocketBoards: BuildingBootloader

5.3. Additional Design Examples

Additional design examples are available in the Project web page on the RocketBoards website. Several examples including booting Linux from various flash devices can be accessed from the following list:

- Intel Stratix 10 SoC FPGA First Single QSPI Flash Boot
- Intel Stratix 10 SoC with eMMC Storage on HPS

6. Configuring the FPGA Fabric from HPS Software

Configuring the FPGA fabric from HPS software is supported only when using the HPS Boot First mode.

When creating the configuration files, you obtain a phase 1 HPS configuration file, and a phase 2 FPGA fabric configuration file. Typically, the two files come from running the Intel Quartus Prime Programming File Generator on a SOF file resulted from compiling a hardware project.

The phase 1 and phase 2 configuration files can also be obtained from different projects, or modified projects, if the following conditions are met:

- The same Programming File Generator version is used for generating both files, as this ensures both have the same SDM firmware version.
- Both SOF files that are used have the same HPS IO settings and HPS DDR settings.
- The FPGA fabric can be configured from both U-Boot and Linux.

6.1. Configuring the FPGA Fabric from U-Boot

The FPGA fabric is configured from U-Boot using data from a RAM buffer with the `fpga load <device> <address> <size>`.

The GSRD uses HPS boots first mode, with the FPGA fabric being configured by U-Boot running the following script before booting Linux:

```
setexpr.b reg *0xFFD120DC;
if itest $reg -eq 3;
then bridge enable;
else load mmc 0:1 $loadaddr ghrd.core.rbf; dcache flush; fpga load 0 $loadaddr
$filesize; bridge enable;
fi
```

The script does the following:

1. Checks if the FPGA fabric is already configured. This can be done with the Intel Quartus Prime Programmer, or before hand, on a previous boot before cold resetting the HPS.
2. If the FPGA fabric is already configured, it enables the bridges by running the `brigde enable` command, then exits.
3. If the FPGA fabric is not configured:

a. It loads the phase 2 core fabric configuration file from the SD card to DDR.
b. It flushes the data caches so that the data can be accessed by the SDM.
c. It calls the `fpga load <device> <address> <size>` command to configure FPGA fabric.
d. It calls `bridge enable` to configure the bridges.

The GSRD is documented on the Intel Stratix 10 Soc GSRD web page on the RocketBoards website.

Note: In boot flow that uses ATF, the first 1 MB of SDRAM is configured as secure region, you must use the address range between 0x100000 (1 MB offset) to 0x20000000 (512 MB offset) for the FPGA configuration file (`.rbf`).

6.2. Configuring the FPGA Fabric from Linux

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.

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-$(CONFIG_ARCH_STRATIX10) += socfpga_stratix10_socdk.dtb
dtb-$(CONFIG_ARCH_STRATIX10) += overlay.dtb
```

2. Create the new `overlay.dts` file and add the overlay information of the RBF file into the file as shown below:

```
/dts-v1/;
/plugin/;
/ {
    fragment@0 {
        target-path = "/soc/base_fpga_region";
        #address-cells = <1>;
        #size-cells = <1>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <1>;

            firmware-name = "overlay.rbf";
            config-complete-timeout-us = <30000000>;
        };
    };
};
```

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

Table 15. Resultant Files

Device Tree File	Description
`socfpga_stratix10_socdk.dtb`	The default `*.dtb` file used with the kernel image to boot the system.
`overlay.dtb`	The `*.dtb` file used to trigger FPGA configuration in OS.

In your compilation output folder, rename the FPGA configuration file (`*.rbf`) to the following name: `overlay.rbf`. Then, copy both the FPGA configuration file (`*.rbf`) and the `overlay.dtb` file to the following location in your Root File System:

```
$ mkdir <your_ROOTFS>/lib/firmware

$ cp overlay.dtb <your_ROOTFS>/lib/firmware/

$ cp overlay.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 overlay.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 overlay.dtb >/sys/kernel/config/device-tree/overlays/0/path
```

6.3. FPGA Partial Reconfiguration from Linux

The Partial Reconfiguration started from Linux relies on the device tree overlay mechanism which allows you to modify the original device tree at run time.

The components needed to perform the Platform Reconfiguration from Linux are listed next. These items are already provided as part of the Partial Reconfiguration example provided in the following RocketBoards page: Intel Stratix 10 SoC GSRD: Partial Reconfiguration

- **dtbt**—This is a script used to manage the overlay. This script provides the `dtbt` command that can be called from the Linux shell providing the ability to add or remove an overlay to the device tree and also list the overlays currently applied. This is located under Linux file system at `/sbin` directory. The source code for this script can be seen from the altera-opensource / dtbt web page.
- **pr_fpga_static_region.dtbo**—This is the overlay for the static region. This is generated from the `pr_fpga_static_region.dts` file and creates the new device tree nodes that describe the PR region and the freeze controller. The attributes for the PR region node are expected to be updated by the PR persona overlays. This overlay needs to be applied before persona overlays. This file is located under Linux file system at the `/boot/devicetree` directory. Next is shown the structure of the static region device tree overlay DTS.

  Note: The static region and PR region can also be defined directly from the initial Linux device tree as alternative to use an overlay for this purpose.

```
fragment@0 {
    target-path = "/soc/base_fpga_region";  /* Device tree hierarchy */
    :
    __overlay__ {
         external-fpga-config;
           :
        /* PR region node */
           fpga_pr_regionN{
             compatible = "fpga-region";
             fpga-bridges = <&freeze_controller_0>;
             :
            };
            /* PR freeze Controller node */
            freeze_controller@offset {
              compatible = "altr,freeze_controller-ver", "altr,freeze-bridge-
controller";
              :
            };
       };
};
```

- **pr_personaX.dtbo**—This is the overlay for the PR Persona X and extends the **fpga_pr_regionN** node created in the PR Static Region overlay. This overlay is generated from the `pr_personaX.dts` file. Here is where it is indicated which is the RBF file that corresponds to the FPGA design to be loaded in the PR region (referred to as the `personaX.rbf`). This overlay also defines additional nodes and specific attributes that applies for this persona in the PR region node in the device tree. There is a PR Persona overlay file for each persona FPGA design to be loaded in the PR region. These overlay files are located under Linux file system at the `/boot/devicetree` directory. Next is shown the structure of the PR Persona Overlay DTS.

```
fragment@1 {
    target-path = "/soc/base_fpga_region/fpga_pr_regionN";
     :
    __overlay__ {
       partial-fpga-config;
       firmware-name = "personaX.rbf";  /* RBF file for this persona */
       :
       /* Specific attributes for this PR Persona in fpga_pr_regionN  node*/
       :
       pr_region_N_nodeA:{
            /* Specific attributes for this PR Persona in sub node A */

       };
       pr_region_N_nodeB:{
            /* Specific attributes for this PR Persona in sub node B */

       };
    };
};
```

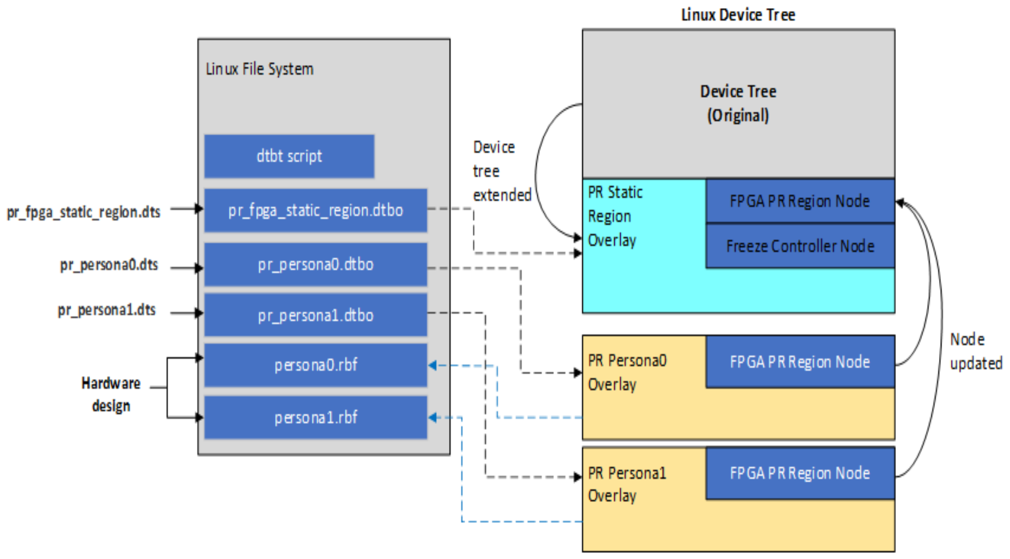
- **personaX.rbf**—This is the bitstream used by personaX overlay and corresponds to the FPGA design to be loaded into the PR region.

  During the partial reconfiguration started by Linux, this accesses the PR RBF files under its file system at the `/lib/firmware` directory.

  Note: In the hardware design, it is not allowed enabling both SGMII and PR at the same time. Because of this, the hardware design needs to be recompiled with SGMII disabled and PR enabled to allow PR to be used.

The following figure shows the components involved in Partial Reconfiguration for 2 Persona using the Linux device tree overlays mechanism:

Figure 34. Components Involved



As indicated before, to perform the partial reconfiguration from Linux, we make use of the `dtbt` command which allows the device tree overlays handling. At the time that the overlay is applied, the PR is performed automatically, and the Linux driver associated to the overlay is loaded. The following table lists some of the functionality that this command provides:

Description	Command Usage
List overlays applied	`dtbt -l`
Add an overlay	`dtbt -a <.dtbo to apply> -p <.dtbo search path>`
Remove an overlay	`dtbt -r <.dtbo to remove > -p <.dtbo search path>`

Note: When the PR overlay is removed, the corresponding Linux driver is removed.

To perform the PR successfully, the following sequence should be followed from the Linux shell:

1. Load static region overlay and confirm that this was applied by listing the overlays.

   ```
   # dtbt -a pr_fpga_static_region.dtbo -p /boot/devicetree
   # dtbt -l
   ```

   Only one static region overlay should be shown.

2. Perform PR by loading the desired overlay and confirm that this was applied by listing the overlays applied.

   ```
   # dtbt -a pr_persona0.dtbo -p /boot/devicetree
   # dtbt -l
   ```

   Static region and persona0 overlays should be shown.

3. If another PR persona wants to be loaded, the current PR persona overlay needs to be removed and then the new PR Persona overlay should be applied.

```
root@agilex:~# dtbt -r pr_persona0.dtbo -p /boot/devicetree
root@agilex:~# dtbt -a pr_persona1.dtbo -p /boot/devicetree
root@agilex:~# dtbt -l
```

   Static region and persona1 overlays should be shown.

Related Information

- Intel Stratix 10 SoC GSRD: Partial Reconfiguration
- altera-opensource / dtbt web page

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 Intel SoC FPGA Edition to load and debug the bootloader software used in your design.

7.1. Reset

Table 16. Reset Effects on Booting and Configuration

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

continued...

Reset Type	Initiated By	Details
Warm Reset	• 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)	• 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.
Software Reset	A software write to the Reset Manager	• A software reset of a CPU does not affect SDM functionality.
Watchdog Reset	Timeout from a user configurable watchdog timer register.	• 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.
Debug Reset	JTAG SRST pin	• To control reset, you must connect the HPS_COLD_nRESET pin to the JTAG SRST pin. • A debug reset initiates a cold reset.
JTAG Reset	JTAG SRST pin	• JTAG can reset the entire device regardless of the `MSEL[2:0]` settings and reload a new configuration or test through JTAG.

7.1.1. HPS Reset Pin

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

Table 17. HPS Reset Pins

Pin Function	Possible Settings	Functional Description
`HPS cold nreset`	`SDM_IO0`, `SDM_IO10-16`	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.

(3) The Cortex*-A53 has four levels of exception from EL3 to EL0. EL3 is the highest privilege and EL0 is the lowest.

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 bootloader 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 Bootloader Using the Arm DS Intel SoC FPGA Edition

You can debug the bootloader by using Arm DS Intel SoC FPGA Edition. In order to do that, you need a JTAG connection, so you must enable the HPS Debug Access Port to be accessible through either the SDM or HPS pins.

For more information, refer to the *Device and Pin Options* section.

For instructions about debugging the Bootloader with Arm DS Intel SoC FPGA Edition, refer to the following web pages on RocketBoards.org:

- Debugging U-Boot with Arm DS Intel SoC FPGA Edition
- Debugging UEFI with Arm DS Intel SoC FPGA Edition

Related Information

- Device and Pin Options on page 24
- Debugging U-Boot with Arm DS Intel SoC FPGA Edition
- Debugging UEFI with Arm DS Intel SoC FPGA Edition

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 18. Data Trace Test ("Walking Ones") Example

Written Value	Read Value	Failure Detected
00000001	00000001	No failure detected.
00000010	00000000	Error, most likely the second data bit, `D[1]`, is stuck low or shorted to ground.
00000100	00000100	No failure detected, confirmed `D[1]` is stuck low or shorted to another trace that is not listed in this table.
00001000	00001000	No failure detected.
00010000	00010000	No failure detected.
00100000	01100000	Error, most likely `D[6]` and `D[5` are short circuited.
01000000	01100000	Error, confirmed that `D[6]` and `D[5]` are short circuited.
10000000	10000000	No failure detected.

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 19. Address Trace Test (Powers of Two) Example

Address	Written Value	Read Value	Failure Detected
00000000	1	1	No failure detected.
00000001	2	2	No failure detected.
00000010	3	1	Error, the second address bit, A[1], is stuck low.
00000100	4	4	No failure detected.
00001000	5	5	No failure detected.
00010000	6	6	No failure detected.
00100000	7	6	Error, A[5] and A[4] are short circuited.
01000000	8	8	No failure detected.
10000000	9	9	No failure detected.

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

- External Memory Interfaces Intel Stratix 10 FPGA IP User Guide
- Intel Stratix 10 SX SoC Development Kit User Guide

8. Intel Stratix 10 SoC FPGA Boot User Guide Archives

For the latest and previous versions of this user guide, refer to Intel Stratix 10 SoC FPGA Boot User Guide. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

9. Document Revision History for Intel Stratix 10 SoC FPGA Boot User Guide

Document Version	Intel Quartus Prime Version	Changes
2023.01.20	22.4	Made the following changes: • Added a note to the *Configuration over AVST* section. • Updated information in *Configuration over JTAG* section. • Added information about Partial Reconfiguration
2022.07.26	21.4	Made the following changes: • Added description of the REBOOT_HPS in the *Reset* section. • Added information about the maximum size of the `.core.rbf` and bit-stream in the *Programming File Generator* section. • Added the *Intel Stratix 10 SoC FPGA Boot User Guide Archives* section • Made changes to *Document Revision History for Intel Stratix 10 SoC FPGA Boot User Guide* — Added the Intel Quartus Prime Version column — Converted this section from an appendix to a chapter
2021.11.10	21.1	Replaced the Supported QSPI Devices table with a link to the Intel Supported Configuration Devices web page.
2021.05.28	21.1	• Replaced the *Creating a Configuration Bitstream for Intel Stratix 10 SoC FPGA* section with the new *Creating the Configuration Files* section • Replaced the *Generating the Linux Kernel Image* section with the new *Golden System Reference Design and Design Examples* section • Renamed section *Configuring the FPGA from SSBL and OS* section to *Configuring the FPGA Fabric from HPS Software* • Renamed section *Configuring the FPGA Using the U-Boot SSBL* section to *Configuring the FPGA Fabric from U-Boot* • Renamed section *Configuring the FPGA Using the Linux Operating System* section to *Configuring the FPGA Fabric from Linux* • Replaced the tool: Arm Development Studio 5* Intel SoC FPGA Edition with Arm Development Studio Intel SoC FPGA Edition • Removed the following sections from *Debugging the HPS Boot Loader Using the Arm Development Studio Intel SoC FPGA Edition*: — *Selecting the HPS Debug Interface* — *Configuring the FPGA and run the Debug FSBL* — *Creating the Debug Configuration* — *Debugging the First Stage Boot Loader (FSBL)* — *Debugging U-Boot*
2020.12.04	20.2	Added a restriction to the *Boot Flow Overview* section that the phase 1 and phase 2 configuration files must be generated from the same Intel Quartus Prime Pro Edition software version.
2020.09.15	19.2	• Added information about the differences between H- and L-tile regarding clock requirements prior to configuration in *Boot Flow Overview*. • Added information clock requirements for HPS first boot mode; and what is required for phase 1 and phase 2 in *First-Stage Bootloader*.

continued...

Document Version	Intel Quartus Prime Version	Changes
2020.06.30	19.2	Removed support for the SD/MMC configuration scheme in Intel Stratix 10 devices.
2019.12.19	19.2	Made the following changes: • Removed all references to EPCQ-L. This flash device is obsolete. • Replaced EPCQL1024 with MT25QU02G in `quartus_pfg` commands. • Removed all references to prebuilt binaries or sources inside the SoC EDS. These files are no longer included in the distribution. • Corrected directory path in *Compiling the SRAM Object File*. It should include 19.3. • Revised *Compiling U-Boot FSBL and SSBL* to get the source code from the GitHub repository. • Updated the version of `socfpga` in *Compiling the Linux Kernel Image* • Revised the debugging section.
2019.12.16	19.2	Made the following changes: • Changed all commands to convert programming files to use the `quartus_pfg` instead of `quartus_cpf`. The `quartus_cpf` command does not handle some of the advanced security features that Intel Stratix 10 devices support. • Added the following note to the *Single SDM Flash* topic: *Due to a problem in the Intel Quartus Prime Pro Edition Software, if you specify the* ***HPS boot from FPGA*** *parameter on the* ***FPGA Interfaces*** *tab of the Intel Arria® 10 Hard Processor System Intel Arria 10 FPGA IP GUI, this information has no effect on HPS behavior.* • Removed statement in *Creating the Raw Programming Data (*.rpd) File For Flash Programming* topic saying that the `.rpd` file generated is the same size as the configuration device. This statement is not true for Intel Stratix 10 and later device families.
2019.07.25	19.2	Note added to explain HPS First "Phase 1 configuration" and "Phase 2 configuration" terminology in Boot Flow Overview on page 14 .
2018.09.24	18.0	• 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.