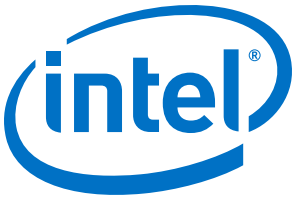


Intel® Stratix® 10 SoC Remote System Update (RSU) User Guide

Updated for Intel® Quartus® Prime Design Suite: **19.3**

Subscribe
Send Feedback

UG-20197 | 2020.01.09
Latest document on the web: **PDF** | **HTML**

Contents

1. Overview

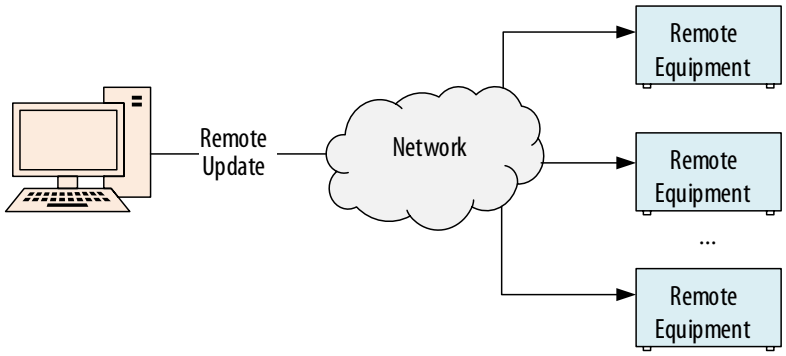
In the remote system update (RSU) implementation for active configuration schemes for all Intel® Stratix® 10 SoC devices, either the HPS drives the RSU or the FPGA drives the RSU. When the HPS drives the RSU, the RSU allows you to reconfigure the QSPI configuration bitstream for an Intel Stratix 10 SoC device once a new version is available on the target equipment.

Note: It is your responsibility to deploy the image over the network to the target remote equipment.

The RSU performs configuration error detection during and after the reconfiguration process. If errors in the application images prevent reconfiguration, then the configuration reverts to the factory image and provides error status information.

If the configuration bitstream in the QSPI flash is corrupted rendering the device non-functional, the only method to recover the device is to connect to the device over JTAG and re-program the QSPI flash. However, this method may not be available if the system does not have a JTAG connector or if the target equipment is in a remote or hard to access location.

Figure 1. Typical Remote System Update Usage



This document details the Intel Stratix 10 HPS remote system update solution and provides an update example using the *Intel Stratix 10 SX SoC Development Kit*.

For more information about when the FPGA drives the RSU, refer to the *Intel Stratix 10 Configuration User Guide*.

Related Information

- Intel Stratix 10 Configuration User Guide
- Intel Stratix 10 SX SoC Development Kit User Guide

1.1. Features

The remote system update solution:

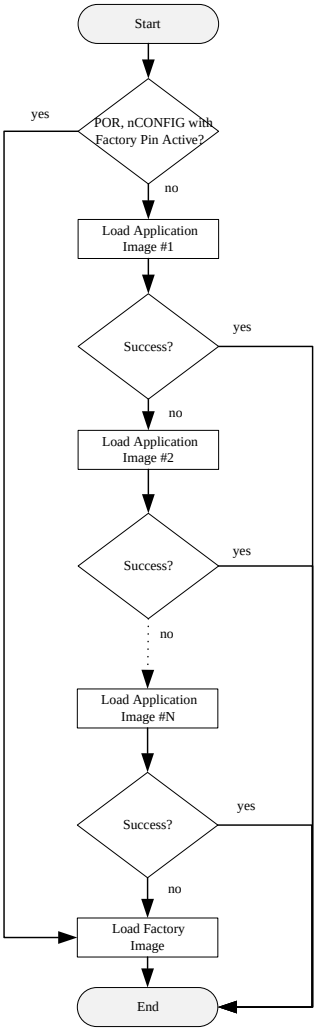
- Provides support for creating the initial flash image for a system to support RSU. The image is created offline, before the device is deployed in the field.
- Allows a set of application images to be tried in a specific order until one is successful. When configuration succeeds, the First Stage Bootloader (FSBL) is loaded and the HPS is taken out of reset (or just the HPS and HPS EMIF I/Os when in HPS first mode). Optionally an HPS watchdog timeout can be treated as a RSU failure too.

  Note: A maximum of seven application images can be specified at image creation time, but more can be added later. The maximum number of application images is 126; but typically only a few are used because flash memory size is limited.
- Loads a factory image if no application image is available, or all application images failed.
- Provides you the option to select a pin so that when it is asserted during a configuration caused by a POR or `nCONFIG` event, it forces the SDM to load the factory image instead of the highest priority application image.
- Provides you with the ability to add and remove application images.
- Provides you with the ability to change the order that the application images are loaded.
- Provides you with the ability to load a specific application image or the factory image from flash.
- Provides you with information about the image that is currently running, and errors that the RSU is encountering.
- Provides you with an API to notify the SDM of the state of the HPS software as a numeric value. The state is reported back after the next image is successfully loaded if the previous image failed due to an HPS watchdog timeout.
- Provides the `max retry` option, which allows each application image and the factory image the option to retry up to three times.
- Provides the option to update the factory image.

The factory and application images are also called bitstreams and they typically contain the FPGA fabric configuration, the SDM firmware, and the HPS FSBL. For HPS first mode, the FPGA fabric configuration is omitted.

Figure 2 on page 7 describes the image selection flow that occurs when the RSU enabled device is configured as a result of a power-up or `nCONFIG` event.

Figure 2. RSU Image Selection Flow



The HPS watchdog timeouts can be optionally configured to be treated as configuration failures. When that happens, the algorithm above behaves as if the image failed to configure, and it moves to the next image. If the factory image is the last image to run, then there is no other image available. When this happens, the SDM clears the FPGA and HPS and remains not configured.

For more information, refer to the *Remote System Update Watchdog* and *Enabling HPS Watchdog to Trigger RSU* sections.

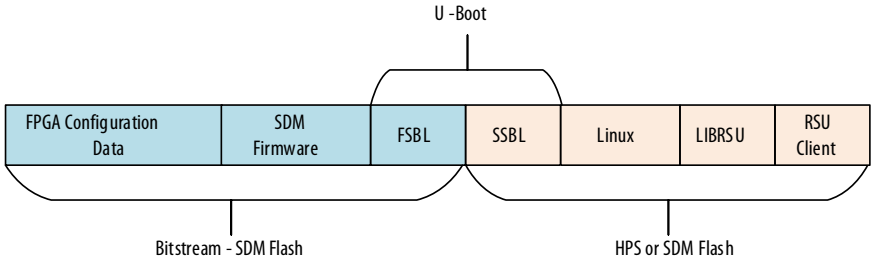
Related Information

- Intel Stratix 10 Configuration User Guide
- Remote System Update Watchdog on page 13
- Enabling HPS Watchdog to Trigger RSU on page 27

1.2. System Components

Figure 3 on page 8 describes the typical components of an Intel Stratix 10 SoC based system using RSU.

Figure 3. System Components



The bitstream is stored in the configuration flash device (QSPI) connected to the SDM pins. The HPS software is typically stored in a mass storage flash device (SD/eMMC/NAND) connected to the HPS pins but can also be stored in the flash device connected to the SDM pins.

Table 1. System Components

Component	Description
FPGA Configuration Data	When using FPGA first mode, this component of the bitstream contains the full FPGA and I/O configuration data. When using HPS first mode, it contains just the HPS and HPS EMIF I/Os .
SDM Firmware	Firmware for the Secure Device Manager: • Provides commands for managing RSU. These commands are not directly accessible to you. Instead both U-Boot and Linux* offer services for accessing the functionality offered by the SDM commands indirectly.
FSBL	HPS First Stage Bootloader: • Initializes hardware and loads SSBL.
SSBL	HPS Second Stage Bootloader: • Loads and boots the operating system • Uses SDM commands to provide you RSU APIs and command line capabilities. • Provides a resident Secure Monitor Call (SMC)(1) handler to allow Linux to indirectly use the SDM commands.
Linux Drivers	Intel RSU driver uses the functionality provided by U-Boot SMC and makes services available to applications running on Linux.
LIBRSU	Linux user space library providing APIs for managing RSU.
RSU Client	Linux sample application which uses LIBRSU for managing RSU.

(1) On Cortex-A53, there are four execution levels: EL0-Application, EL1-OS, EL2-Hypervisor, and EL3-Secure Monitor. Interacting with SDM is only allowed for software running at EL3. U-Boot runs at EL3 while Linux runs at EL1. For Linux to communicate with the SDM, it has to issue an SMC trap to a handler left resident by U-Boot. Then that handler runs at EL3 and is able to communicate with the SDM.

Intel provides an RSU solution that focuses on the reliable update of the components that are part of the configuration bitstream, and are located in SDM flash. It is your responsibility to devise a scheme for the reliable update of the rest of the system components.

For information about version compatibility requirements of various system components, refer to the *Version Compatibility Considerations* section.

Related Information

Version Compatibility Considerations on page 69

1.3. Glossary

Terms	Description
Firmware	Firmware that runs on SDM. Implements various functions including: • FPGA configuration • Voltage regulator configuration • Temperature measurement • HPS software load • HPS Reset • RSU • Device security Each application image and the factory image have their firmware at offset zero in the image.
Decision firmware	Special SDM firmware used to identify and load the highest priority image. Previous versions of this user guide refer to decision firmware as DCMF. Four identical copies reside in flash, starting at address zero.
Decision firmware data	Data structure stored in flash containing the following information used by the decision firmware: • Factory load pin assignment • PLL settings for external clock source • Quad SPI pin assignments • Number of retries (`max retry`) each image has to be loaded This data structure is populated by the Intel Quartus® Prime Programming File Generator with the information retrieved from the factory image SOF file.
Configuration pointer block (CPB)	List of application image addresses, in order of priority.
Sub-partition table (SPT)	Data structure to facilitate the management of the flash storage.
Application image	Configuration bitstream that implements your design.
Factory image	The fallback configuration bitstream that the RSU loads when all attempts to load an application image fail.
Initial RSU flash image	Contains the factory image, the application images, the decision firmware, and the associated RSU data structures.
Factory update image	An image that updates the following RSU-related items in flash: • Factory image • Decision firmware • Decision firmware data

2. Use Cases

This section describes the main use cases for the Remote System Update.

2.1. Manufacturing

At manufacturing time, you must provide the flash partitioning information, a factory image and one or more application images. A tool called "Programming File Generator" uses this input to create an initial RSU flash image file. You must program the flash with this image file to prepare the device for remote system update. Both FPGA first and HPS first image types are supported.

2.2. Application Image Boot

The application image performs your application function. Remote system update allows you to safely switch a system from one application image to the next without restarting and risk of failure if one of the application images is corrupted or contains serious bugs.

The device maintains, in flash, a list of the addresses where application images are present in flash. This list is known as the configuration pointer block.

When attempting to load an application image the SDM traverses the configuration pointer block in reverse order. It attempts to load the first image, and if this load is successful then the image is now in control of the device.

If loading the first image is unsuccessful, then the SDM attempts to load the second image. If this image also fails, the SDM continues to advance to the next image until it reaches a successful image or all images fail to load. If no image is successful then the SDM loads the factory image.

2.3. Factory Image Boot

The purpose of the factory image is to provide enough functionality to allow a device whose application images have all been corrupted (or replaced with broken application images) to obtain a new application image and program that image into QSPI.

Once the factory image is loaded, new application images could be programmed in QSPI to replace the failed ones, from either U-Boot or Linux,

The SDM loads the factory image in the following situations:

- You assign the function LOADFACTORY to an SDM pin and assert the pin soon after a POR or nCONFIG release.
- All SDM attempts to load the application images fail.
- You request the factory image to be loaded.

When loading the factory image, the configuration system treats it in the same way as it does an application image.

2.4. Modifying the List of Application Images

The SDM uses the configuration pointer block to determine priority of application images.

The pointer block operates taking into account the following characteristics of quad SPI flash memory:

- On a sector erase, all the sector flash bits become 1's.
- A program operation can only turn 1's into 0's.

The pointer block contains an array of values which have the following meaning:

- All 1's – the entry is unused. The client can write a pointer to this entry. This is the state after a quad SPI erase operation occurs on the pointer block.
- All 0's – the entry has been previously used and then canceled.
- A combination of 1's and 0's – a valid pointer to an application image.

When the configuration pointer block is erased, all entries are marked as unused. To add an application image to the list, the client finds the first unused location and writes the application image address to this location. To remove an application image from the list, the client finds the application image address in the pointer block list and writes this address to 0s.

If the configuration pointer block runs out of space for new application images, the client compresses the pointer block by completing the following actions:

- Read all the valid entries from the configuration
- Erasing the pointer block
- Adding all previously valid entries
- Adding the new image

When using HPS to manage RSU, both the U-Boot and LIBRSU clients implement the block compression. For designs that drive RSU from FPGA logic, you can implement pointer block compression many different ways, including Nios® II code, a scripting language, or a state machine.

Pointer block compression does not occur frequently because the pointer block has up to 508 available entries.

There are two configuration pointer blocks: a primary (CPB0) and a backup (CPB1). Two blocks enable the list of application images to be protected if a power failure occurs just after erasing one of them. When a CPB is erased and re-created, the header is written last. The CPB header is checked prior to use to prevent accidental use if a power failure occurred. For more information, refer to the *Configuration Pointer Block Layout* topic. When compressing, the client compresses (erases and rewrites) the primary CPB completely. Once the primary CPB is valid, it is safe to modify the secondary CPB. When rewriting, the magic number at the start of a CPB is the last word written in the CPB. (After this number is written only image pointer slot values can be changed.)

When the client writes the application image to flash, it ensures that the pointers within the main image pointer of its first signature block are updated to point to the correct locations in flash. When using HPS to manage RSU, both the U-Boot and LIBRSU clients implement the required pointer updates. For more information, refer to the *Application Image Layout* topic.

Related Information

- Configuration Pointer Block Layout on page 23
- Application Image Layout on page 23

2.5. Querying RSU Status

The SDM firmware offers a command for querying the RSU status, providing the following information:

- RSU status code
- Address of the currently running image
- Address of the last failing image (sticky)
- Error code for last failing image (sticky)
- Error location for last failing image (sticky)
- Decision firmware RSU interface version (added in Intel Quartus Prime Pro Edition software version 19.3)
- Current image (application or factory) firmware RSU interface version (added in Intel Quartus Prime Pro Edition software version 19.3)
- `retry` counter value (added in Intel Quartus Prime Pro Edition software version 19.3)

The SDM firmware also offers commands for the following:

- Clearing the sticky fields related to last failing image (added in Intel Quartus Prime Pro Edition software version 19.3)
- Resetting the retry counter value to zero (added in Intel Quartus Prime Pro Edition software version 19.3)

The functionality offered by all of the above SDM commands is available from both U-Boot and LIBRSU.

For more information, refer to the RSU Status and Error Codes on page 78.

Related Information

RSU Status and Error Codes on page 78

2.6. Loading a Specific Image

The SDM firmware provides a command to load a specific image from flash. The image can be the factory image or one of the application images. You have access to this functionality from both U-Boot and LIBRSU.

If the requested image fails to load, the SDM tries to reload the currently running image at the time the command is issued. If the currently running image also fails to load, the SDM tries to load the highest priority image in the CPB. If the highest priority

image in the CPB is the failing image, then it tries to load the second image in the CPB and continues to traverse the pointer table in priority order. If there are no more images left to try in the CPB, then the SDM tries to load the factory image.

When U-Boot issues this command , the SDM immediately resets and wipes both the FPGA and HPS, then it proceeds to load the specified image. When LIBRSU calls the equivalent API, the command is not immediately sent to SDM as this causes Linux to lock up. Instead, the U-Boot SMC handler makes a note to call the command on the next `reboot` request from the HPS, so that the SDM command only gets issued when the Linux `reboot` command is run. This causes the kernel to shut down first and the HPS `reboot` request is sent to the U-Boot SMC handler.

Note: The effect of loading a new image on HPS is similar with a cold reset. However, the SDM HPS_COLD_nRESET pin (if selected) is not asserted.

Calling the SDM command to load a specific image causes the following sticky fields from the state reported by SDM to be cleared:

- `failed_image`
- `error_details`
- `error_location`
- `state`

For more information, refer to the *RSU Status and Error Codes* section.

Related Information

RSU Status and Error Codes on page 78

2.7. Protected Access to Flash

After the HPS acquires QSPI flash ownership in the bootloader, the HPS has full access to QSPI and can potentially corrupt the flash.

In order to minimize the risk of rendering the system non-operational, decision firmware, decision firmware data, and factory image areas are not exposed as Linux Memory technology device (MTD) devices.

Another option to protect the flash may be to use QSPI vendor specific commands to mark the decision firmware, decision firmware data, and factory image areas as read-only. However, that may result in a more complex procedure for updating the decision firmware, decision firmware data, and factory image areas.

2.8. Remote System Update Watchdog

The HPS watchdog timer can be used to trigger a reset if they are not serviced periodically. The desired behavior with respect to the RSU flow on such a reset can be selected from Intel Quartus Prime tools to be one of the following:

- Trigger a cold reset – the SDM loads the HPS FSBL from the current image again.
- Trigger a warm reset – the SDM causes the HPS to re-start the HPS FSBL, without reloading it.
- Trigger a cold reset and a remote system update – the SDM considers the last loaded application image a failure and then loads the next application image in the CPB, or loads the factory image, if the list is exhausted. If the watchdog timer expires in this mode while executing the factory image, then the HPS is wiped (same as for a regular cold reset). However, the SDM HPS_COLD_nRESET pin (if selected) is not asserted.

For more information about how to select the HPS watchdog behavior, refer to the *Intel Quartus Prime Pro Edition* section.

Currently, the FSBL automatically enables the watchdog as one of the first things it does, and then services the watchdog periodically. Then the SSBL disables the watchdog as one of the very first things it does.

It is your responsibility to enable and service the watchdog for SSBL and Linux, if desired.

Note: Servicing the watchdog timer requires a deeper understanding of how it works, and is beyond the scope of this document.

Note: The current application image is not removed from the CPB when a watchdog timeout occurs. This means that the image is not permanently marked as unusable and can try again (for example after a POR or nCONFIG even happens).

Related Information

Intel Quartus Prime Pro Edition Software on page 25

2.9. RSU Notify

The SDM offers a command called RSU `Notify` which the HPS software uses to let the SDM know the current HPS software state that has a 16-bit numerical value.

If the HPS watchdog timeout triggers an RSU failure, the next FPGA application or factory image is loaded. When the HPS queries the RSU state, the top 16 bits are `0xf006` and the bottom 16 bits contain the last RSU `Notify` value reported to the SDM before the watchdog timeout.

The RSU `Notify` command is called with a value of:

- 1—from FSBL just before SSBL is entered
- 2—from SSBL just before the control is passed to the operating system
- Custom—from both U-Boot and LIBRSU

RSU `Notify` values reported when a watchdog timeout occurs:

Value	Description
0	FSBL either did not complete or did not reach the point of launching SSBL.
1	FSBL did complete, but SSBL either did not complete or did not reach the point of launching the operating system.
2	Both FSBL and SSBL are complete and attempting to launch the operating system.
Custom	You reported this custom value from either U-Boot or LIBRSU.

2.10. Updating the Factory Image

Support for updating the factory image was added in Intel Quartus Prime Pro Edition software version 19.2.

Note: In addition to the factory image, the update procedure also updates the decision firmware and decision firmware data.

Note: The factory update process requires new U-Boot or LIBRSU functionality. You may need to update your U-Boot or LIBRSU to be able to update the factory image.

In order to make your system ready to support updating the factory image you need to have an available slot in flash that is the size of the maximum factory image you anticipate using plus 512 KB.

You may temporarily use an application image slot for the update procedure, because application images are typically larger than factory images. However, that means one less application image slot is available during the factory image update procedure.

The procedure for updating the factory image and decision firmware is as follows:

1. Create a factory update image with the Programming File Generator, using the new factory image SOF file as input. The factory update image contains the new factory image, new decision firmware, new decision firmware data, and special firmware to perform the actual update.
2. Deploy the factory update image on the remote system.
3. Write the factory update image to the flash and make it the highest priority, with either U-Boot or LIBRSU. Because the factory update image has a different format than application images, use the new LIBRSU APIs: `rsu_slot_program_factory_update_buf` and `rsu_slot_program_factory_update_file`. There are also U-Boot equivalents.
4. Pass control to the factory update image by requesting it to be loaded or by toggling nCONFIG.
5. The factory update image proceeds and replaces the factory image, decision firmware and decision firmware data with the new copies, then erases itself from the CPB.
6. At the end, the factory update image loads the new highest priority image in the CPB, or the factory image if the CPB is empty.

The factory update flow is resilient to power loss. If the power is lost during the update, the next time the power is back up, the factory update image resumes the update process from where it stopped.

For examples on how to perform the factory image update from both U-Boot and Linux, refer to the *Remote System Update Example* section.

For API reference information, refer to the *LIBRSU Reference Information* section, located in the *LIBRSU Reference Information* appendix.

Related Information

- Remote System Update Example on page 37
- LIBRSU Reference Information on page 80
- Remote System Update Example on page 37
- LIBRSU Reference Information on page 80

2.11. Retrying when Configuration Fails

The global option, `max retry`, new to Intel Quartus Prime Pro Edition software version 19.3, allows you the option to configure the number of retries for both application images and the factory image, if any configuration failure occurs:

- Flash corruptions
- Authentication and encryption errors
- RSU watchdog timeouts

The `max retry` option is selected in Intel Quartus Prime for the factory image project and is stored in the decision firmware data structure in flash. The default value for `max retry` is 1, which means each image is tried only once, the standard behavior before Intel Quartus Prime software version 19.3. The maximum value for `max retry` is 3, which means each image is tried up to three times.

The SDM maintains the `retry counter`, a counter that counts the number of retries for the current image and then goes to the next image when the maximum is reached. The counter is reset to zero when a new image is loaded, and its maximum value is [`max retry` - 1]. You can query the value of the `retry counter` and also request it to be reset to zero from both U-Boot and LIBRSU.

Both the decision firmware and the currently running application or factory image firmware must be generated with Intel Quartus Prime Programmer software version 19.3 or above so that all of the `max retry` functionality is available. If only the decision firmware version is 19.3 or above, then the images are tried the specified number of times, but querying the `retry counter` or resetting it to zero is not available.

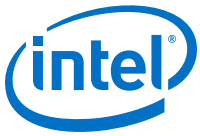
The retry behavior does not apply when there is a request to load a specific image. For more information, refer to the *Loading a Specific Image* section.

For information about how to query the decision firmware and application firmware versions, refer to the *RSU Status and Error Codes* section.

For examples on how `max retry` works for both U-Boot and Linux, refer to the *Remote System Update Example* section.

Related Information

- RSU Status and Error Codes on page 78



- Loading a Specific Image on page 12
- Remote System Update Example on page 37

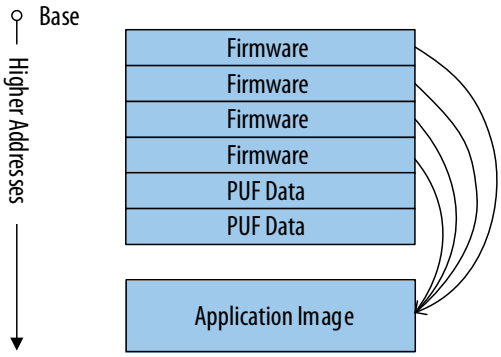
3. Quad SPI Flash Layout

3.1. High Level Flash Layout

3.1.1. Standard (non-RSU) Flash Layout

In the standard (non-RSU) case, the flash contains four firmware images and the application image. To guard against possible corruption, there are four The firmware contains a pointer to the location of the highest priority application image in flash. The Intel Quartus Prime Software creates a physically unclonable function (PUF) data partition if you select **QSPI Intrinsic ID PUF-wrapped** for the **Encryption key storage select** parameter on the **Assignments ➤ Device ➤ Device and Pin Options ➤ Security ➤** menu. (2)

Figure 4. Flash Layout - Non-RSU



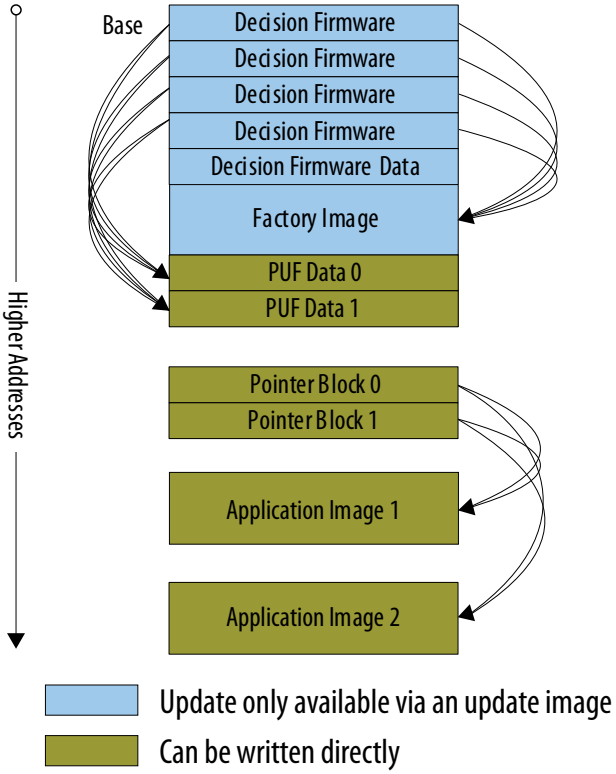
3.1.2. RSU Flash Layout – SDM Perspective

In the RSU case, decision firmware replaces the standard firmware. The decision firmware copies have pointers to the following structures in flash:

- Decision data
- One factory image
- Two Pointer Blocks (CPBs)

(2) tab.

Figure 5. RSU Flash Layout - SDM Perspective



The decision firmware data stores basic settings, including the following:

- The clock and pins that connect to quad SPI flash memory
- The **Direct to Factory Image** pin that forces the SDM to load the factory image

  Note: You can set this pin on the following menu in the factory image project: **Assignments ➤ Device ➤ Device and Pin Options ➤ Configuration ➤ Configuration Pin Options**

The pointer blocks contain a list of application images to try until one of them is successful. If none are successful, the SDM loads the factory image. To ensure reliability, the pointer block includes a main and a backup copy in case an update operation fails.

Both the factory image and the application images start with firmware. First, the decision firmware loads the firmware. Then, that firmware loads the rest of the image. These implementation details are not shown in the figure above. For more information, refer to the *Application Image Layout* section.

Related Information

Application Image Layout on page 23

3.1.3. RSU Flash Layout – Your Perspective

The sub-partition table (SPT) manages the quad SPI flash.

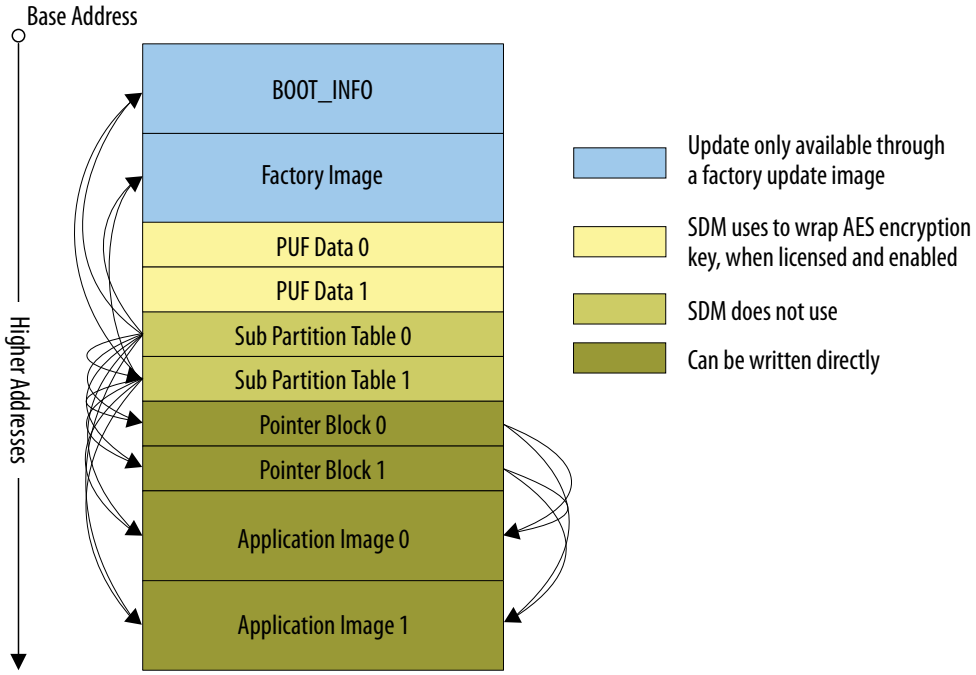
The Intel Quartus Prime Programming File Generator creates the SPT when creating the initial manufacturing image. To ensure reliable operation, the Programming File Generator creates two copies of the sub-partition table and the configuration pointer block, SPT0 and SPT1 and CPB0 and CPB1

The initial RSU image stored in flash typically contains the following partitions:

Table 2. Typical Sub-Partitions of the Initial RSU Image

Sub-partition Name	Contents
BOOT_INFO	Decision firmware and decision firmware data
FACTORY_IMAGE	Factory Image
PUF	Physically Unclonable Function
SPT0	Sub-partition table 0
SPT1	Sub-partition table 1
CPB0	Pointer block 0
CPB1	Pointer block 1
P1 (you enter)	Application image 1
P2 (you enter)	Application image 2

Figure 6. RSU Flash Layout - Your Perspective



To summarize, your view of flash memory is different from SDM view in two ways:

- You do not need to know the addresses of the decision firmware, decision firmware data, and factory image.
- You have access to the sub-partition tables. The sub-partition tables provide access to the data structures required for remote system update.

3.2. Detailed Quad SPI Flash Layout

3.2.1. RSU Sub-Partitions Layout

The *Flash Sub-Partitions Layout* table shows the layout of RSU flash images.

Table 3. Flash Sub-Partitions Layout

Flash Offset	Size (in bytes)	Contents	Sub-Partition Name
0 K	256 K	Decision firmware	BOOT_INFO
256 K	256 K	Decision firmware	
512 K	256 K	Decision firmware	
768 K	256 K	Decision firmware	
1M	8 K + 24 pad	Decision firmware data	
1M+32 K	32 K	Reserved for SDM	
2M+32	32 K	PUF data copy 1	PUF.
2M+32	32 K	PUF data copy 2	
2M+128 K	varies	Factory image	FACTORY_IMAGE
Next	4 K + 28 K pad	Sub-partition table (copy 0)	SPT0
Next	4 K + 28 K pad	Sub-partition table (copy 1)	SPT1
Next	4 K + 28 K pad	Pointer block (copy 0)	CPB0
Next	4 K + 28 K pad	Pointer block (copy 1)	CPB1
Next	varies	Application image 1	You assign
Next	varies	Application image 2	You assign

The Intel Quartus Prime Programming File Generator allows you to create many user partitions. These partitions can contain application images and other items such as the Second Stage Boot Loader (SSBL), Linux kernel, or Linux root file system.

When you create the initial flash image, you can create up to seven partitions for application images. There are no limitations on creating empty partitions.

3.2.2. Sub-Partition Table Layout

The following table shows the structure of the sub-partition table. The Intel Quartus Prime software supports up to 126 partitions. Each sub-partition descriptor is 32 bytes.

Table 4. Sub-partition Table Layout

Offset	Size (in bytes)	Description
0x000	4	Magic number 0x57713427
0x004	4	Version number (0 for this document)
0x008	4	Number of entries
0x00C	20	Reserved

continued...

Offset	Size (in bytes)	Description
0x020	32	Sub-partition Descriptor 1
0x040	32	Sub-partition Descriptor 2
0xFE0	32	Sub-partition Descriptor 126

Each 32-byte sub-partition descriptor contains the following information:

Table 5. Sub-partition Descriptor Layout

Offset	Size	Description
0x00	16	Sub-partition name, including a null string terminator
0x10	8	Sub-partition start offset
0x18	4	Sub-partition length
0x1C	4	Sub-partition flags

Two flags are currently defined:

- bit 0: system/reserved
- bit 1: read only

The Intel Quartus Programming File Generator sets these flags as follows at image creation time, then they are not changed afterward:

Table 6. Flags Specifying Contents and Access

Partition	System	Read Only
BOOT_INFO	1	1
FACTORY_IMAGE	1	1
PUF data copy 1	1	1
PUF data copy 2	1	1
SPT0	1	0
SPT1	1	0
CPB0	1	0
CPB1	1	0
P1	0	0
P2	0	0

3.2.3. Configuration Pointer Block Layout

The configuration pointer block contains a list of application image addresses. The SDM tries the images in sequence until one of them is successful or all fail. The structure contains the following information:

Table 7. Pointer Block Layout

Offset	Size (in bytes)	Description
0x00	4	Magic number 0x57789609
0x04	4	Size of pointer block header (0x18 for this document)
0x08	4	Size of pointer block (4096 for this document)
0x0C	4	Reserved
0x10	4	Offset to image pointers (IPTAB)
0x14	4	Number of image pointer slots (NSLOTS)
0x18	—	Reserved
IPTAB	8	First (lowest priority) image pointer slot
	8	Second (2nd lowest priority) image pointer slot
	8	…
	8	Last (highest priority) image pointer

The configuration pointer block can contain up to 508 application image pointers. The actual number is listed as NSLOTS. A typical configuration pointer block update procedure consists of adding a new pointer and potentially clearing an older pointer. Typically, the pointer block update uses one additional entry. Consequently, you can make 508 remote system updates before the pointer block must be erased. The erase procedure is called *pointer block compression*. This procedure is safe. There are two copies of pointer block. The copies are in different flash erase sectors. While one copy is being updated the other copy is still valid.

3.2.4. Application Image Layout

The application image comprises SDM firmware and the configuration data. The configuration data includes up to four sections. The SDM firmware contains pointers to those sections. The table below shows the location of the number of sections and the section pointers in a application image.

Table 8. Application Image Section Pointers

Offset	Size (in bytes)	Description
0x1F00	4	Number of sections
	…	
0x1F08	8	Address of 1st section
0x1F10	8	Address of 2nd section
0x1F18	8	Address of 3rd section

continued...

Offset	Size (in bytes)	Description
0x1F20	8	Address of 4th section
	...	
0x1FFC	4	CRC32 of 0x1000 to 0x1FFB

The section pointers must match the actual location of the FPGA image in flash. Two options are available to meet this requirement:

- You can generate the application image to match the actual location in quad SPI flash memory, Because different systems may have a different set of updates. This option may not be practical.
- You can generate the application image as if it is located at address zero, then update the pointers to match the actual location.

Note: When using HPS to manage RSU, both U-Boot and LIBRSU clients implement the above procedure to relocate application images targeting address zero in the actual destination slot address.

Here is the procedure to update the pointers from an application image created for `INITIAL_ADDRESS` to `NEW_ADDRESS`:

1. Create the application image, targeting the `INITIAL_ADDRESS`.
2. Read the 32-bit value from offset 0xF100 of the application image to determine the number of sections.
3. For `<s>= 1 to number_of_sections`:
   a. `section_pointer` = read the 64-bit section pointer from 0xF100 + (s * 8)
   b. Subtract `INITIAL_ADDRESS` from `section_pointer`
   c. Add `NEW_ADDRESS` to `section_pointer`
   d. Store updated `section_pointer`
4. Recompute the CRC32 for addresses 0x1000 to 0x1FFB. Store the new value at offset 0x1FFC. The CRC32 value must be computed on a copy of the data using the following procedure:
   a. Swap the bits of each byte so that the bits occur in reverse order and compute the CRC.
   b. Swap the bytes of the computed CRC32 value to appear in reverse order.
   c. Swap the bits in each byte of the CRC32 value.
   d. Write the CRC32 value to flash.

Note: The factory update image has a different format. Refer to the *Generating a Factory Update Image* topic for the correct procedure to generate the factory update image.

4. Intel Quartus Prime Software and Tool Support

This section lists the Intel Quartus Prime tools you can use with the Intel Quartus Prime Pro Edition software for the RSU scenarios. For more information about each tool, refer to the corresponding documentation.

4.1. Intel Quartus Prime Pro Edition Software

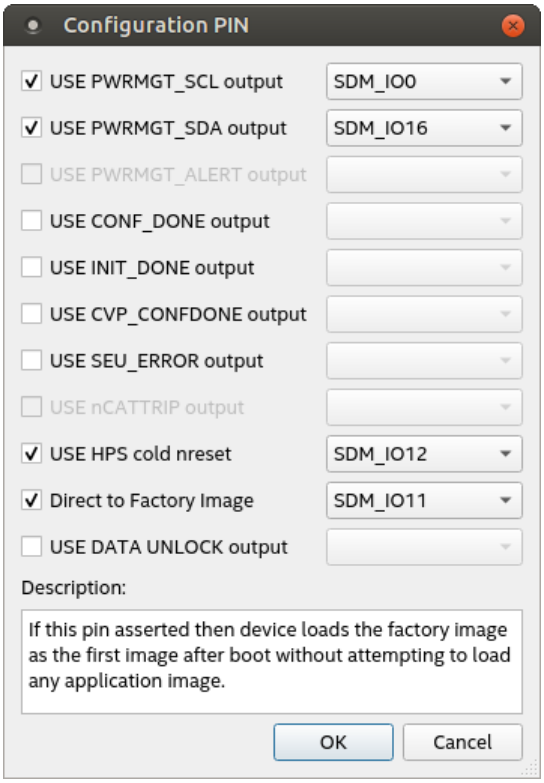
You must use the Intel Quartus Prime Pro Edition software to compile the hardware projects you use for remote system update.

4.1.1. Selecting Factory Load Pin

Platform Designer offers the option to select the pin to use to force the factory application to load on a reset. ,

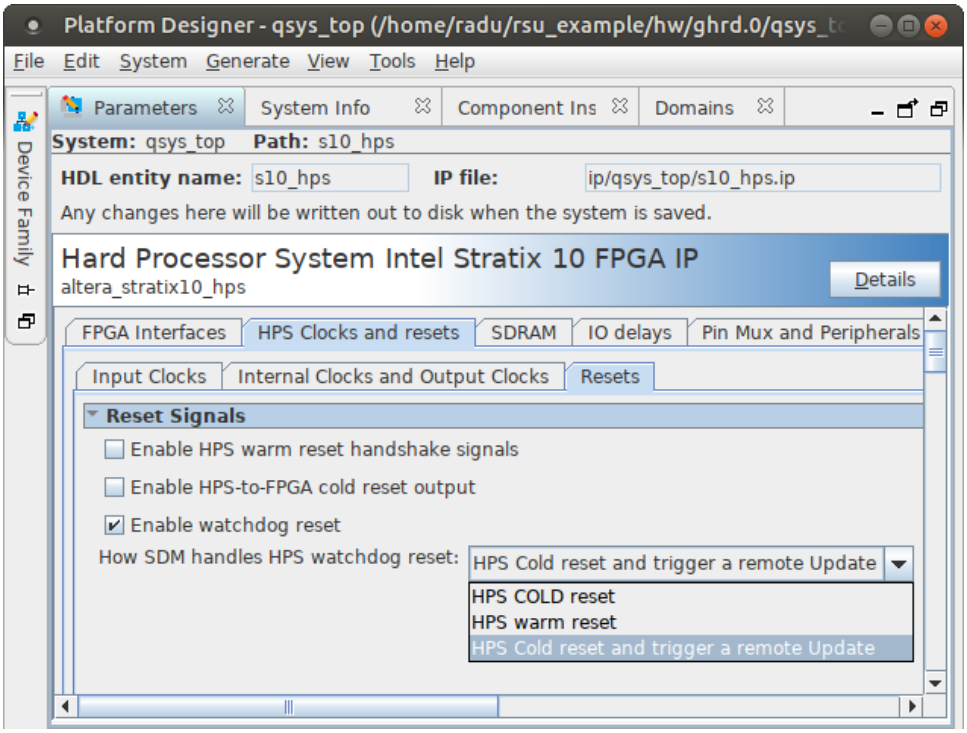
1. Navigate to **Assignments ➤ Device ➤ Device and Pin Options ➤ Configuration ➤ Configuration Pin Options**
2. Check the **Direct to Factory Image** check box.
3. Select the desired pin from the drop box.

Figure 7. Configuration PIN GUI



4.1.2. Enabling HPS Watchdog to Trigger RSU

The tool also offers the option of selecting what happens when you enable an HPS watchdog, but do not service it, and produces an HPS reset. The option is available as an HPS component property in Platform Designer, as shown below:



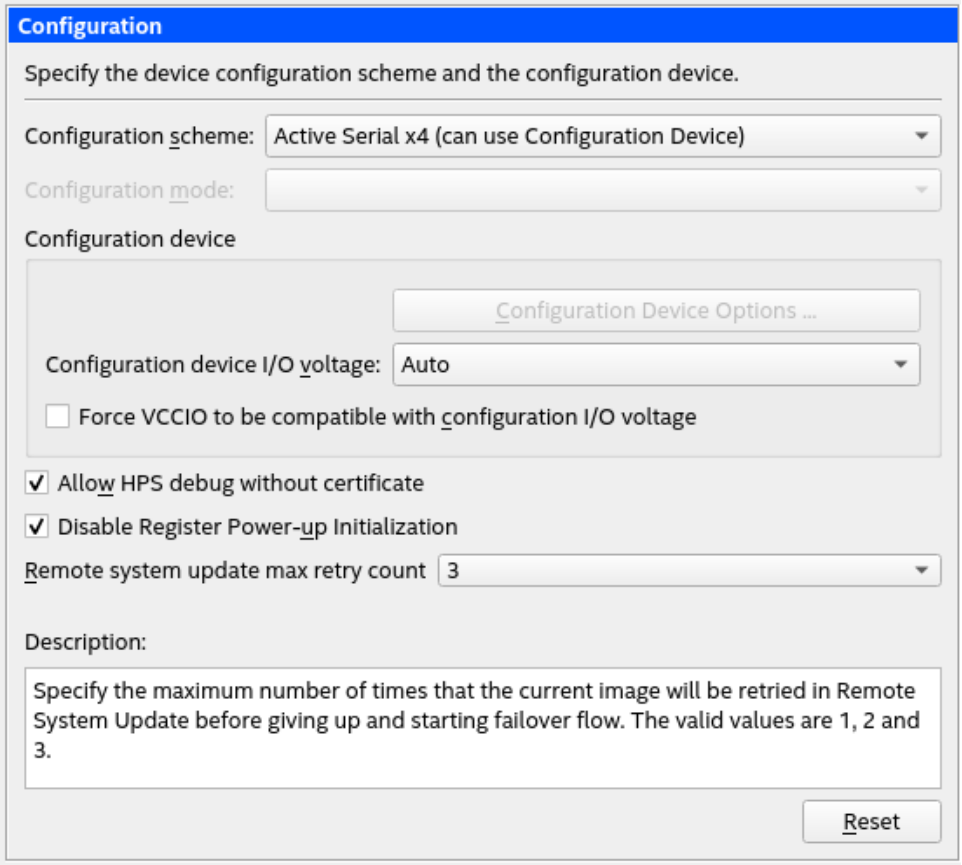
When the highlighted option is selected and an HPS watchdog is enabled and times out because it is not serviced, an HPS cold reset is issued, and SDM considers the current application image as a failure. Then SDM tries to load the next application image in the configuration pointer block, or the factory image if all application images fail.

If the factory image is the last image to run, then there is no other image available. When this happens, the SDM clears the FPGA and HPS and remains not configured.

4.1.3. Setting the `Max Retry` Parameter

The `max retry` parameter specifies how many times the application and factory image are able to retry when configuration failures occur. The default value is one, which means each image is tried only once. The maximum possible value is three, which means each image can retry up to three times.

The `max retry` parameter is specified for the hardware project used to create the factory image, from the Intel Quartus Prime GUI by navigating to **Assignments ➤ Device ➤ Device and Pin Options ➤ Configuration** and selecting the value for the "Remote system update max retry count" field.



The parameter can also be specified by directly editing the project Intel Quartus Prime settings file (.qsf) and adding the following line or changing the value if it is already there:

```
set_global_assignment -name RSU_MAX_RETRY_COUNT 3
```

4.2. Programming File Generator

Intel Quartus Prime Programming File Generator, a new Intel Quartus Prime tool became available in Intel Quartus Prime Pro Edition software version 18.0. The tool creates programming files for Intel Stratix 10, including the Intel Stratix 10 SoC RSU initial flash images, application images, and factory update images.

For usage examples, refer to the *Creating the Initial Flash Image*, *Creating the Initial Flash Image*, and *Creating the Factory Update Image* sections.

For more information about the tool, refer to *Intel Quartus Prime Pro Edition User Guide: Programmer*.

Related Information

- Creating the Initial Flash Image on page 40
- Creating the Application Image on page 49
- Creating the Factory Update Image on page 49

- Intel Quartus Prime Pro Edition User Guide: Programmer

4.2.1. Programming File Generator File Types

The most important file types created by the Programming File Generator are listed in the following table:

File Extension	File Type	Description
`.jic`	JTAG Indirect Configuration File	These files are intended to be written to the 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 application image, or a factory update image.
`.map`	Memory Map File	These files contain details about where the input data was placed in the output file. This file is human readable.
`.rbf`	Raw Binary File	These files are binary files which can be used typically to configure the FPGA fabric for HPS first use cases. They can also contain configuration images for other types of flash than QSPI (for example parallel NOR flash).

4.2.2. `Bitswap` Option

The Intel Quartus Prime Programmer assumes by default that the binary files have the bits in the reversed order for each byte. Because of this, the "`bitswap=on`" option needs to be enabled as follows:

- For each input binary file (`.bin` and `.hex` files are supported).
- For each output RPD file (full flash images, application images and factory update images).

The `bitswap` option is used accordingly in the examples presented in this document.

4.3. Intel Quartus Prime Programmer

You can use the Intel Quartus Prime Programmer to program the initial flash image. For an example, refer to the *Flashing the Initial Image to QSPI* section.

Related Information

Flashing the Initial RSU Image to QSPI on page 52

5. Software Support

This section presents details about the software support for RSU. The information refers to the following versions of software and tools:

- Intel Quartus Prime Pro Edition software version 19.3
- U-Boot from GitHub tag: ACDS19.3_REL_GSRD_PR
- Linux from GitHub tag: ACDS19.3_REL_GSRD_PR
- LIBRSU and RSU Client from GitHub tag: ACDS19.3_REL_GSRD_PR

5.1. SDM RSU Support

Besides implementing the actual RSU configuration flow, the SDM offers commands to interact with RSU:

- Get the flash address of the currently running image.
- Get details on the errors that occurred when trying to load an image which failed.
- Clear the sticky error information to be able to record new error information.
- Get the locations of SPTs.
- Load a specific image.
- Report the state of HPS software.
- Query the value of the current retry counter.
- Reset the value of the current retry counter.

The SDM commands need to be called from EL3, the highest execution level on Cortex-A53.

The SDM commands are not publicly documented. Full support is offered for accessing the relevant services from both U-Boot and LIBRSU.

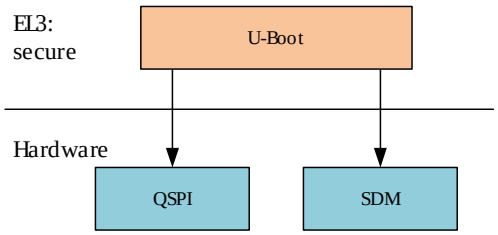
5.2. U-Boot RSU Support

U-Boot provides the following RSU-related features:

- Enables you to access the functionality of all SDM RSU commands from U-Boot command line.
- Enables you to access the functionality of all SDM RSU commands from U-Boot source code, using an interface similar with LIBRSU.
- Implements a resident SMC (secure Monitor Call) handler which is then used by Linux to interact with the relevant SDM services.

U-Boot runs at EL3, the highest execution level, so it is allowed to access the SDM APIs. U-Boot also accesses the QSPI flash to read the contents of the SPTs and CPBs.

Figure 8. U-Boot RSU



Related Information

Remote System Update Example on page 37

5.2.1. U-Boot SMC Handler

The U-Boot SMC handler runs at EL3, the highest execution level, and allows software running at lower execution levels to access services offered by the SDM such as FPGA configuration and SDM APIs.

Getting the location of the SPTs is not offered as a service, because the Linux device tree ensures that the Linux view of the QSPI flash starts from the SPT location. For the actual interface and implementation refer to the file: `arch/arm/mach-socfpga/smc_rsu_s10.c`.

Note: This information is provided for reference only, as you do not need to access the U-Boot SMC handler directly. Instead, the LIBRSU and RSU client enable you to get access to all the RSU services.

Related Information

U-Boot Source Code Details on page 74

5.2.2. U-Boot RSU APIs

U-Boot offers a full set of APIs to manage RSU, similar with the LIBRSU APIs.

Refer to the source code file `arch/arm/mach-socfpga/include/mach/rsu.h` for details on the APIs. Refer to source code `arch/arm/mach-socfpga/rsu_s10.c` for the U-Boot `rsu` command implementation, which are implemented using the APIs.

Refer to LIBRSU Reference Information on page 80 for more details about LIBRSU.

Related Information

LIBRSU Reference Information on page 80

5.2.3. U-Boot RSU Commands

U-Boot offers the `rsu` command, with a full set of options for managing the RSU, similar with the functionality offered by the RSU client.

You can see all the `rsu` command options by running it without any parameters:

```
SOCFPGA_STRATIX10 # rsu
rsu - SoCFPGA Stratix10 SoC Remote System Update

Usage:
rsu dtb   - Update Linux DTB qspi-boot parition offset with spt0 value
list  - List down the available bitstreams in flash
slot_by_name <name> - find slot by name and display the slot number
slot_count - display the slot count
slot_disable <slot> - remove slot from CPB
slot_enable <slot> - make slot the highest priority
slot_erase <slot> - erase slot
slot_get_info <slot> - display slot information
slot_load <slot> - load slot immediately
slot_load_factory - load factory immediately
slot_priority <slot> - display slot priority
slot_program_buf <slot> <buffer> <size> - program buffer into slot, and make it
highest priority
slot_program_buf_raw <slot> <buffer> <size> - program raw buffer into slot
slot_program_factory_update_buf <slot> <buffer> <size> - program factory update
buffer into slot, and make it highest priority
slot_rename <slot> <name> - rename slot
slot_size <slot> - display slot size
slot_verify_buf <slot> <buffer> <size> - verify slot contents against buffer
slot_verify_buf_raw <slot> <buffer> <size> - verify slot contents against raw
buffer
status_log - display RSU status
update <flash_offset> - Initiate firmware to load bitstream as specified by
flash_offset
notify <value> - Let SDM know the current state of HPS software
clear_error_status - clear the RSU error status
reset_retry_counter - reset the RSU retry counter
```

Refer to the *Exercising U-Boot RSU Commands* section for examples of how to use the U-Boot `rsu` command.

The following commands do not have an RSU client equivalent:

- `list`
- `update`
- `dtb`

The `rsu list` command:

- Queries the SDM about the location of the SPT in flash, reads and displays it.
- Reads the CMF pointer block from flash and displays the relevant information.
- Queries SDM about the currently running image, RSU state and the encountered errors and displays the information.

The following is an example of the `rsu list` command being used:

```
SOCFPGA_STRATIX10 # rsu list
RSU: Remote System Update Status
Current Image     : 0x01000000
Last Fail Image   : 0x00000000
State             : 0x00000000
Version           : 0x00000101
Error location    : 0x00000000
Error details     : 0x00000000
Retry counter     : 0x00000000
RSU: Sub-partition table 0 offset 0x00910000
RSU: Sub-partition table 1 offset 0x00918000
SF: Detected mt25qu02g with page size 256 Bytes, erase size 4 KiB, total 256 MiB
```

```
RSU: Sub-partition table content
       BOOT_INFO    Offset: 0x0000000000000000    Length: 0x00110000    Flag :
0x00000003
   FACTORY_IMAGE    Offset: 0x0000000000110000    Length: 0x00800000    Flag :
0x00000003
              P1    Offset: 0x0000000001000000    Length: 0x01000000    Flag :
0x00000000
            SPT0    Offset: 0x0000000000910000    Length: 0x00008000    Flag :
0x00000001
            SPT1    Offset: 0x0000000000918000    Length: 0x00008000    Flag :
0x00000001
            CPB0    Offset: 0x0000000000920000    Length: 0x00008000    Flag :
0x00000001
            CPB1    Offset: 0x0000000000928000    Length: 0x00008000    Flag :
0x00000001
              P2    Offset: 0x0000000002000000    Length: 0x01000000    Flag :
0x00000000
              P3    Offset: 0x0000000003000000    Length: 0x01000000    Flag :
0x00000000
RSU: CMF pointer block offset 0x00920000
RSU: CMF pointer block's image pointer list
Priority 1 Offset: 0x0000000001000000 nslot: 0
```

The `rsu update` command is used to tell the SDM to load an image from a specific address. The following is an example of the `rsu update` command:

```
SOCFPGA_STRATIX10 # rsu update 0x03000000
RSU: RSU update to 0x0000000003000000
```

The `rsu dtb` command is used to let U-Boot update the first QSPI partition information in the Linux DTB so that it starts immediately after the BOOT_INFO partition. This way the decision firmware, decision firmware data, and the factory image are not accessible from Linux, as this reduces the risk of accidental corruption for them. The size of the partition is also reduced accordingly.

For more information, refer to the *Protected Access to Flash* section.

The `rsu dtb` operates on the DTB loaded in memory by U-Boot, before passing it to Linux. The sequence to use is:

1. Load DTB.
2. Run `rsu dtb` command.
3. Boot Linux

Related Information

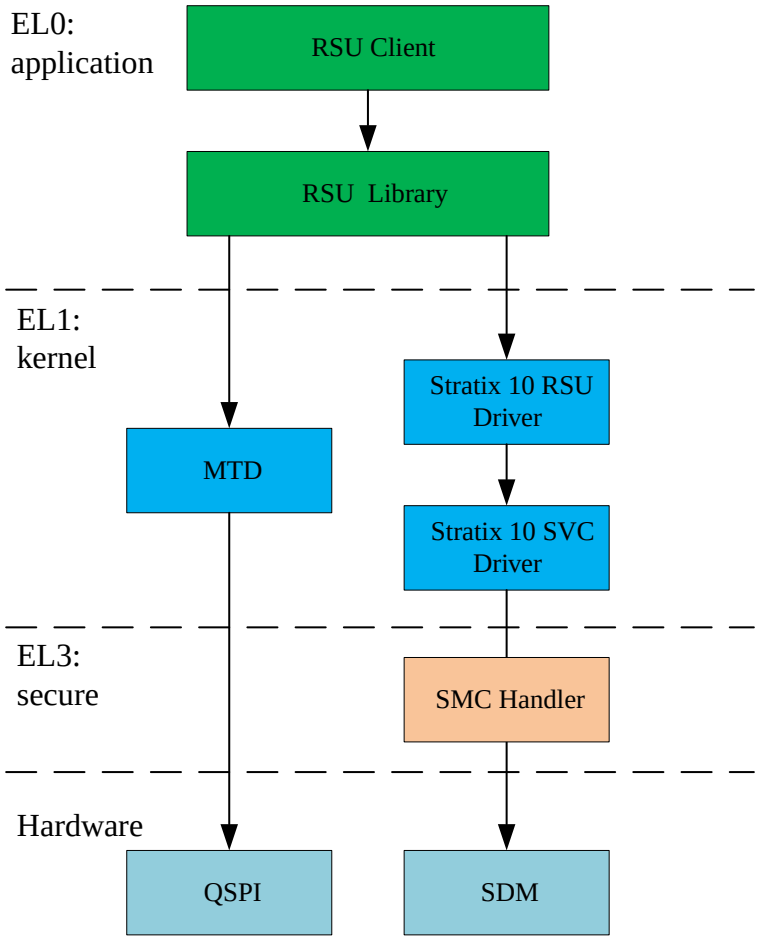
- Protected Access to Flash on page 13
- Exercising U-Boot RSU Commands on page 61

5.3. Linux RSU Support

The following RSU facilities are offered on Linux:

- LIBRSU user-space library which allows you to perform a complete set of RSU operations.
- RSU client example application which exercises the LIBRSU APIs.

Figure 9. Linux RSU Overview



5.3.1. Intel Service Driver

The SMC handler is left resident by U-Boot and is running at EL3, the highest execution level, therefore it can access the SDM services. The Intel Service Driver allows the kernel to execute SMCs in order to access the services offered by the SMC handler.

The default kernel configuration defines `CONFIG_INTEL_STRATIX10_SERVICE=y`, which means that this driver is part of the kernel. Besides RSU, the Intel Service Driver also provides the FPGA configuration services offered by the U-Boot SMC handler.

The driver does not need to be used directly, so its interface is not documented here. For more information, refer to the source code in the `drivers/firmware/stratix10-svc.c` file for details.

Note: Linux is a rapidly changing community project. The information from this section is valid for Linux kernel version 4.14.130-ltsi, was changed from version 4.9.78-ltsi, and may change again in the future.

5.3.2. Intel RSU Driver

The Intel RSU driver exports the RSU services using the **sysfs** interface. The source code is located in the `drivers/firmware/stratix10-rsu.c` file.

The default kernel configuration defines `CONFIG_INTEL_STRATIX10_RSU=m` which means it is configured as a loadable module which needs to be loaded with `insmod`. For information about how to use this driver, refer to the *Exercising the RSU Client* section, located in the *LIBRSU Reference Information* appendix.

The driver offers its services through the following `sysfs` files located in the `/sys/devices/platform/stratix10-rsu.0` folder. This path is a configurable parameter for LIBRSU. For more information, refer to the *Configuration File* section, located in the *LIBRSU Reference Information* appendix.

Table 9. SysFS Entries for Intel RSU Driver

File	R/W	Description
`current_image`	RO	Location of currently running image in SDM QSPI flash.
`fail_image`	RO	Location of failed image in SDM QSPI flash.
`error_details`	RO	Opaque error code, with no meaning to users.
`error_location`	RO	Location of error within the image that failed.
`state`	RO	State of the RSU system.
`version`	RO	Version of the RSU system.
`notify`	WO	Used to notify SDM of the HPS software execution stage, also for clearing the error status, and resetting the retry counter.
`retry_counter`	RO	Current value of the `retry_counter`, indicating how many times the current image retries to be loaded. A value of zero means this is the first time.
`reboot_image`	WO	Address of image to be loaded on next `reboot` command.

Note: This information is provided as reference only, as using the RSU driver directly is not recommended or supported. Instead, use the LIBRSU or RSU client directly to perform the RSU tasks.

Note: Linux is a rapidly changing community project. The information from this section is valid for Linux kernel version 4.14.130-ltsi, was changed from version 4.9.78-ltsi, and may change again in the future.

Related Information

- RSU Status and Error Codes on page 78
- Exercising the RSU Client on page 53
- Configuration File on page 80

5.3.3. LIBRSU Library

The RSU library offers a complete set of RSU APIs which are callable from your applications. The library is built on top of the Intel RSU driver and it uses the Linux MTD framework to access the QSPI flash.

The LIBRSU Library uses the term "slot" to refer to a sub-partition which is intended to contain an application image. It also uses the term "priority" to refer to the fact that the images are loaded by SDM in the order defined by the configuration pointer block.

For information about LIBRSU configuration file, data types, and APIs, refer to the *LIBRSU Reference Information* section.

Related Information

LIBRSU Reference Information on page 80

5.3.4. RSU Client

The RSU Client is an example Linux application which is built on top of the LIBRSU and exercises the APIs offered by the library. For more information about the RSU client command option and also usage examples, refer to the *Exercising the RSU Client* section, located in the *LIBRSU Reference Information* appendix.

The RSU Client can be used as-is, but Intel recommends that you write your own application to manage RSU according to your specific requirements.

Related Information

Exercising the RSU Client on page 53

6. Remote System Update Example

This chapter presents a complete Remote System Update example, including the following:

- Creating the initial flash image containing the bitstreams for a factory image, one application image, and two empty slots to contain additional application images.
- Creating an SD card with the SSBL, Linux, LIBRSU, RSU client and an application image and a factory update image to be deployed.
- Exercising the Linux RSU client application.
- Exercising the U-Boot RSU commands.
- Exercising the Intel RSU Linux driver.

6.1. Prerequisites

The following items are required to run the RSU example:

- Host PC running Ubuntu 16.04 LTS (other Linux versions may work too)
- Minimum 48GB of RAM, required for compiling the hardware designs
- SoC EDS Professional Edition software version 19.3 – for the hardware projects
- Intel Quartus Prime Pro Edition software version 19.3 – for compiling the hardware projects, generating the flash images and writing to flash
- Access to Internet – to clone the git trees for U-Boot, Linux, zlib and LIBRSU and to build the Linux rootfs using Yocto.
- Intel Stratix 10 SoC Development kit, production version – for running the example.

6.2. Git Source Versions

The following Git tags were used to test the example presented in this chapter:

Item	Git Tree	Branch	Tag
U-Boot	https://github.com/altera-opensource/u-boot-socfpga	`origin/socfpga_v2019.04`	ACDS19.3_REL_GSRD_PR
Linux Kernel	https://github.com/altera-opensource/linux-socfpga	`origin/socfpga-4.14.130-ltsi`	ACDS19.3_REL_GSRD_PR
Intel RSU	https://github.com/altera-opensource/intel-rsu	`origin/master`	ACDS19.3_REL_GSRD_PR

The tags identify the Intel Quartus Prime Pro Edition software version 19.3 release. Use the branch names to enable getting the latest features. For more information, refer to the *Version Compatibility Considerations*.

Note: Intel policy specifies that only the current and immediately previous U-Boot and Linux branches are kept on GitHub. As a new branch is supported, the oldest branch is removed. You must keep a local copy of the sources used to build your binaries in case you need to reproduce the build or make changes in the future.

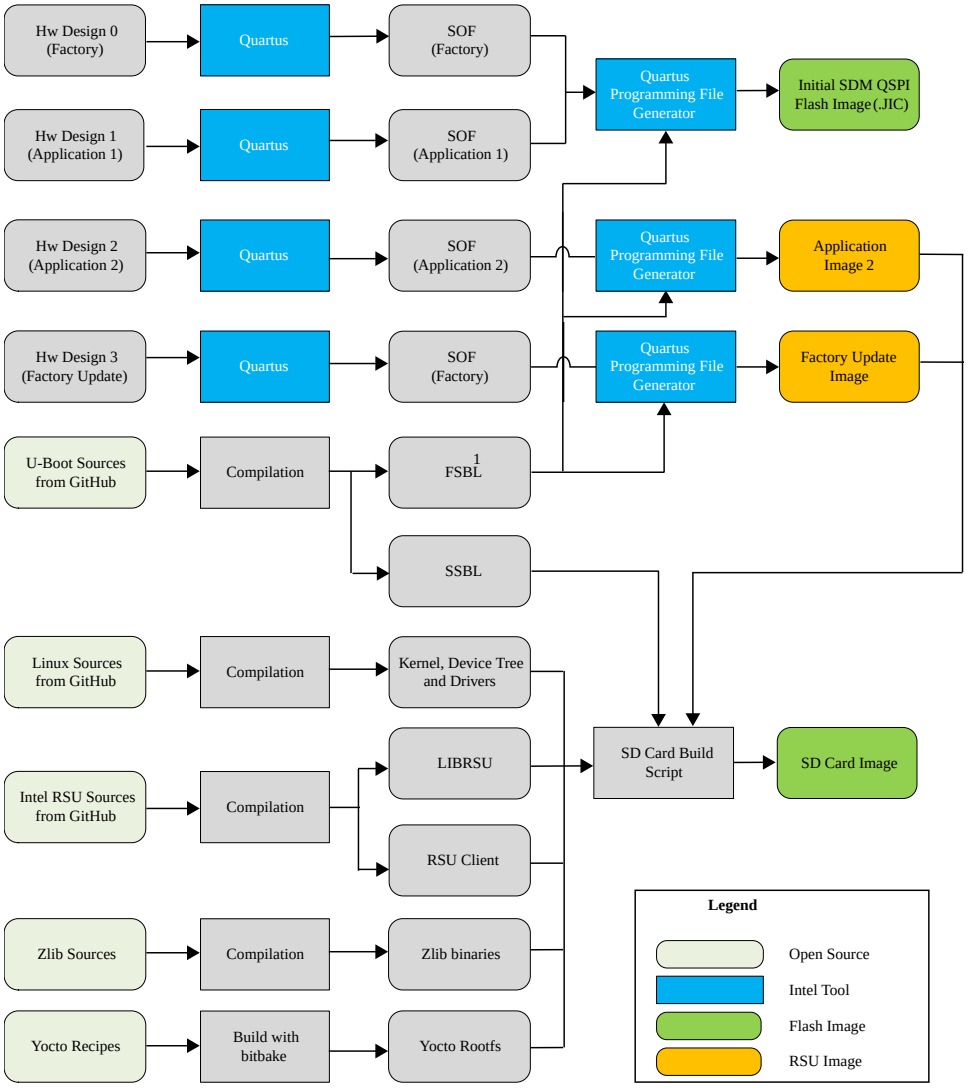
Related Information

Version Compatibility Considerations on page 69

6.3. Building RSU Example Binaries

The diagrams below illustrate the build flow used for this example.

Figure 10. RSU Example Build Flow



(1) It is not mandatory to use the same FSBL for each image. If the user has some low level recovery code in the factory image (for example, then the actual binary may differ.)

The end results of the build flow are:

- Initial flash image: contains the factory image, an application image and two empty application image partitions aka slots.
- SD card image: contains SSBL (U-Boot), Linux device tree, Linux kernel, Linux rootfs with the Intel RSU driver, LIBRSU, RSU Client, two application images and the factory update image

6.3.1. Setting up the Environment

All the commands listed in this chapter are run from the context of an Embedded Command Shell, and also use a specific toolchain from Linaro*. Follow the instructions below to set up the required environment:

```
# create the top folder used to store all the example files
sudo rm -rf ~/rsu_example && mkdir ~/rsu_example && cd ~/rsu_example
export TOP_FOLDER=`pwd`
# start an embedded command shell
~/intelFPGA_pro/19.3/embedded/embedded_command_shell.sh
# retrieve and setup the toolchain
wget https://releases.linaro.org/components/toolchain/binaries/7.4-2019.02/\
aarch64-linux-gnu/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
tar xf gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
export PATH=`pwd`/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin/:$PATH
# setup the environment variables used to compile Linux and LIBRSU
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-
```

6.3.2. Building the Hardware Projects

Create four different hardware projects, based on the GHRD from the Intel SoC FPGA Embedded Development Suite (SoC EDS), with a few changes:

- Use a different ID in the `SystemID` component, to make the binaries for each project slightly different.
- Change the behavior of watchdog timeout, to issue a cold reset and trigger an RSU event.
- Set the `max retry` parameter to three, so that each application image and the factory image have up to three retries when configuration failures occur.

The commands to create and compile the projects are listed below:

```
cd $TOP_FOLDER
# compile hardware designs: 0-factory, 1,2-applications, 3-factory update
rm -rf hw && mkdir hw && cd hw
for version in {0..3}
do
rm -rf ghrd.$version && mkdir ghrd.$version && cd ghrd.$version
tar xf $SOCEDS_DEST_ROOT/examples/hardware/s10_soc_devkit_ghrd/tgz/*.tar.gz
make clean
make scrub_clean
rm -rf *.qpf *.qsf *.txt *.bin *.qsys ip/qsys_top/ ip/subsys_jtg_mst/ ip\
/subsys_periph/
sed -i 's/0xACD5CAFE/0xABAB000'$version'/g' create_ghrd_qsys.tcl
sed -i 's/W_RESET_ACTION ./W_RESET_ACTION 2/g' construct_hps.tcl
make generate_from_tcl
echo "set_global_assignment -name RSU_MAX_RETRY_COUNT 3" \
>> ghrd_1sx280lu2f50e2vg.qsf
make sof
```

```
cd ..
done
cd ..
```

After completing the above steps, the following SOF files are created:

- `hw/ghrd.0/output_files/ghrd_1sx280lu2f50e2vg.sof`
- `hw/ghrd.1/output_files/ghrd_1sx280lu2f50e2vg.sof`
- `hw/ghrd.2/output_files/ghrd_1sx280lu2f50e2vg.sof`
- `hw/ghrd.3/output_files/ghrd_1sx280lu2f50e2vg.sof`

6.3.3. Building U-Boot

The following commands can be used to get the U-Boot source code and compile it:

```
cd $TOP_FOLDER
rm -rf u-boot-socfpga
git clone https://github.com/altera-opensource/u-boot-socfpga
cd u-boot-socfpga
#git checkout -b test ACDS19.3_REL_GSRD_PR
git checkout -t -b test origin/socfpga_v2019.04
make clean && make mrproper
make socfpga_stratix10_defconfig
make -j 24
cd ..
```

After completing the above steps, the following files are created:

- `u-boot-socfpga/spl/u-boot-spl-dtb.hex` — FSBL (U-boot SPL) hex file:
- `u-boot-socfpga/u-boot.img` — SSBL (U-Boot) image file

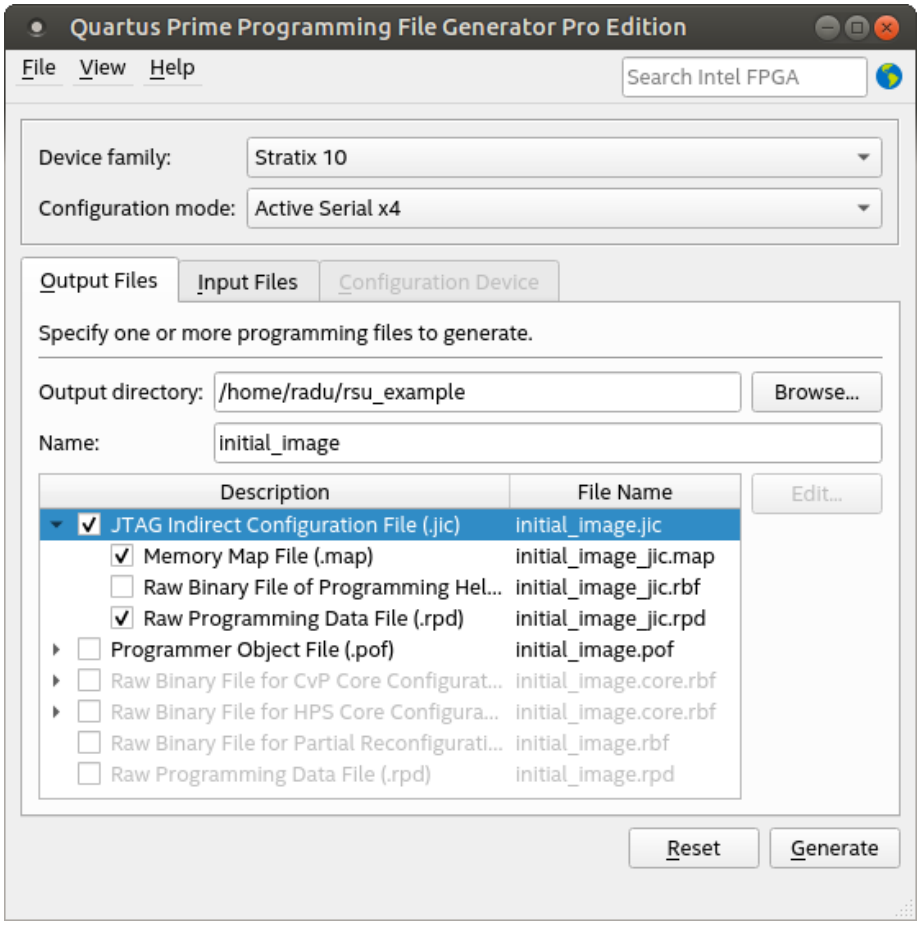
6.3.4. Creating the Initial Flash Image

Create an initial flash image containing the factory and application images and the associated RSU data structures.

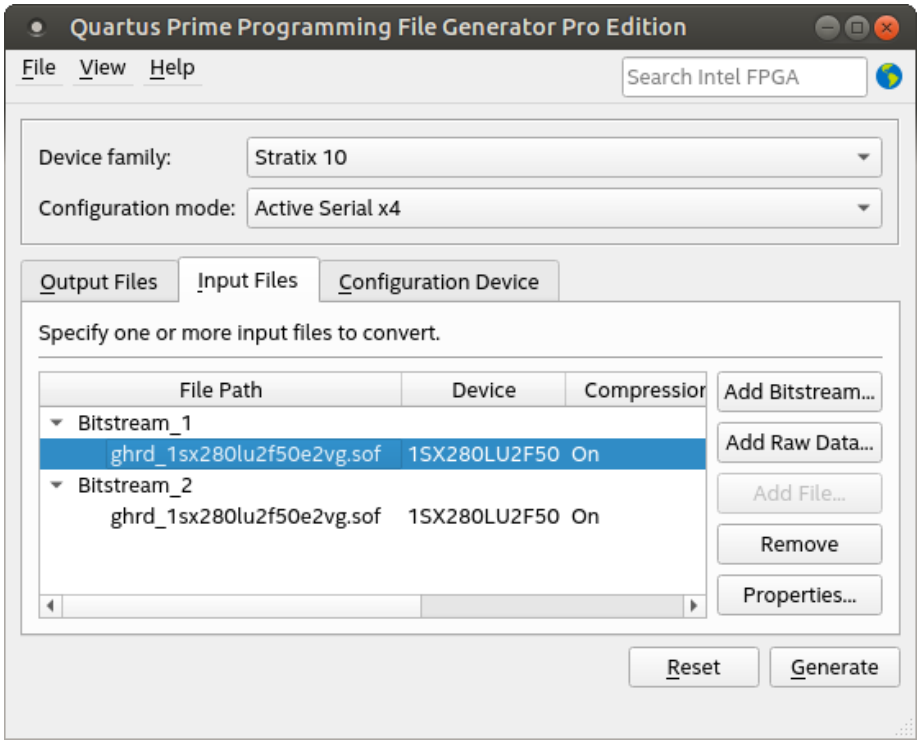
1. Start the **Programming File Generator** tool by running the **qpfgw** command:

   ```
   cd $TOP_FOLDER
   qpfgw &
   ```

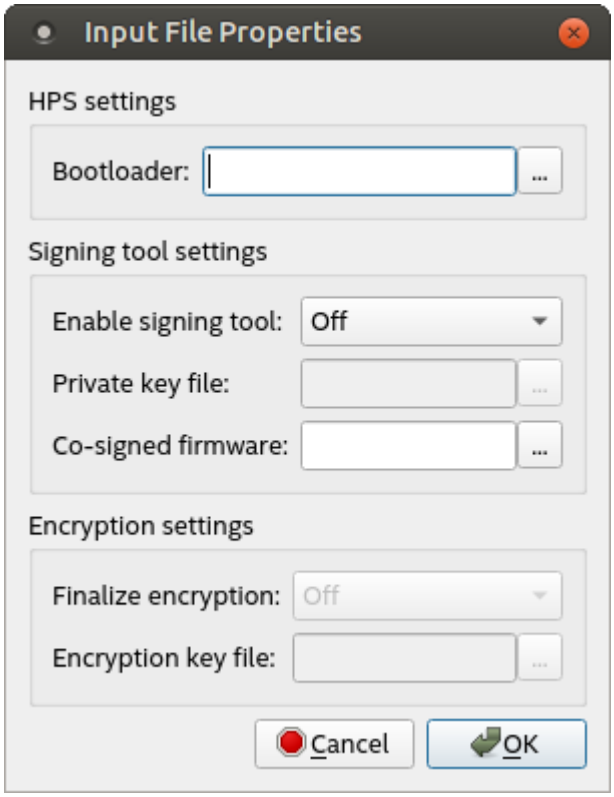
2. Select the **Device family** as **Intel Stratix 10**, and **Configuration mode** as **Active Serial x4**.
3. Change the **Name** to "initial_image".
4. Select the output file type as **JTAG Indirect Configuration File (. jic)**, which is the format used by the Intel Quartus Prime Programmer tool for writing to the QSPI flash.
5. Select the optional **Memory Map File (.map)** file so that it is also generated. The `.map` file contains information about the resulted flash layout.
6. Select the optional **Raw Programming Data File (.rpd)** file so that it is also generated. This file contains the binary flash content, without anything else added. The window looks similar to this:



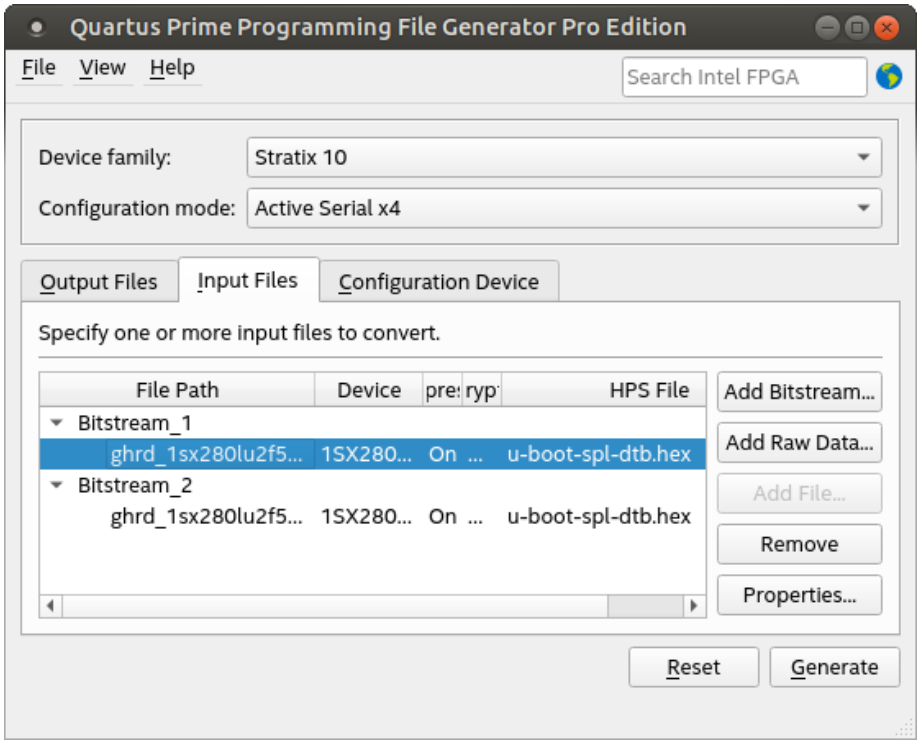
7. Click the **Raw Programming Data File (.rpd)** file to select it. Then click the **Edit ...** button and select the `Bitswap` option to be "On". This enables the RPD file to be usable by HPS software like U-Boot and Linux if needed.
8. Once the output type was selected, click the **Input Files** tab.
9. In the **Input Files** tab click the **Add Bitstream** button, then browse to `$TOP_FOLDER/hw/ghrd.0/output_files`, select the file `ghrd_1sx280lu2f50e2vg.sof`, and then click **Open**. This is the initial factory image. Do the same for the `$TOP_FOLDER/hw/ghrd.1/output_files/ghrd_1sx280lu2f50e2vg.sof` image. This is the initial application image. The tab now looks like below:



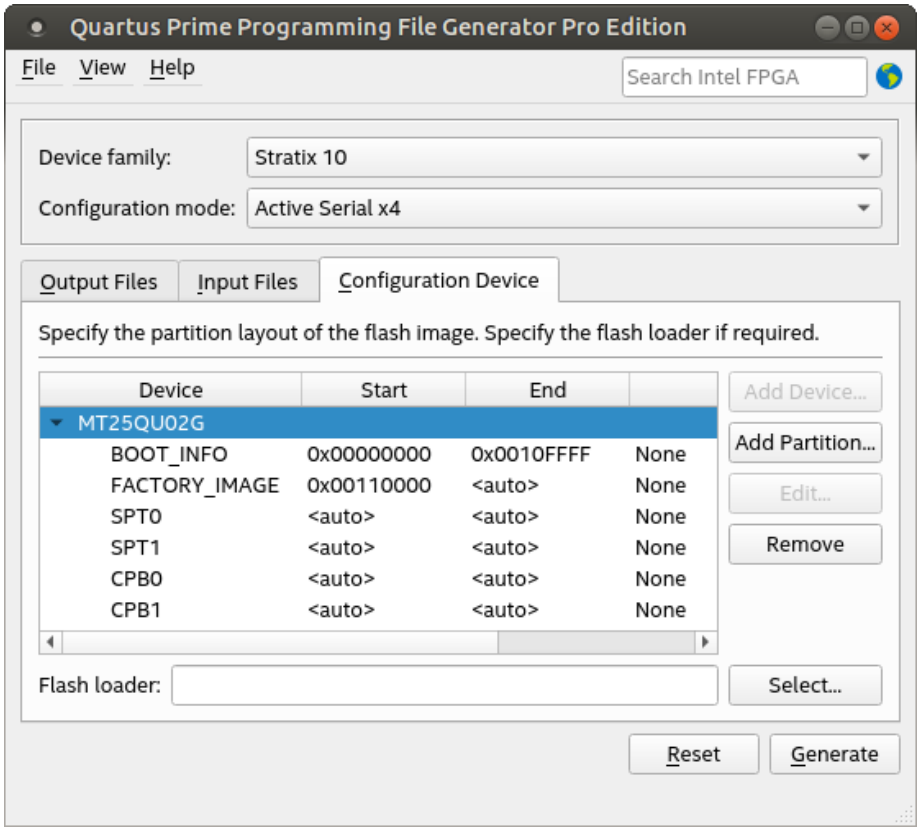
10. Click the first `.sof` file, then click the **Properties** button on the right side. This opens the window to browse for the first stage bootloader and select authentication and encryption settings.



11. Click the **Bootloader ... (Browse)** button and select the file `$TOP_FOLDER/u-boot-socfpga/spl/u-boot-spl-dtb.hex`, then click OK.
12. Click the second `.sof` file and add the same first stage bootloader file to it. The **Input Files** tab now looks like shown below:

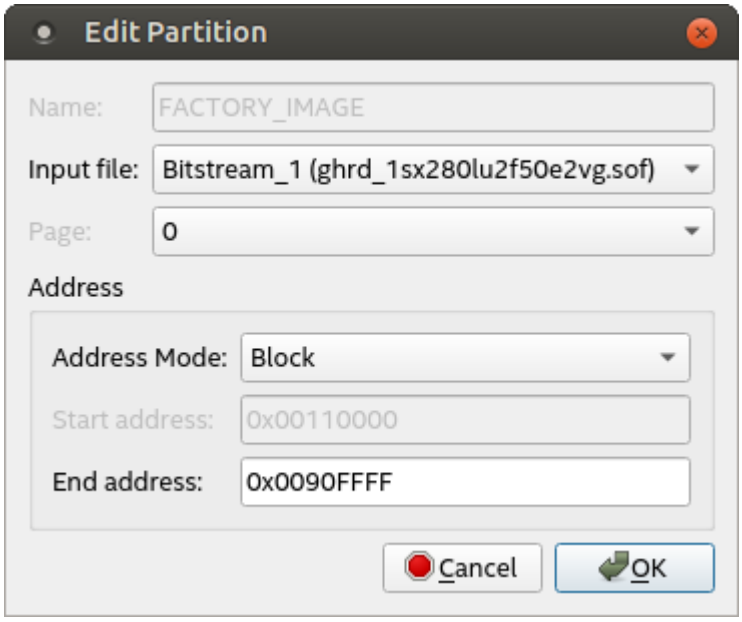


13. Click the **Configuration Device** tab. Note that the tab is only enabled once at least one input file was added in the **Input Files** tab.
14. Because more than one input file was added in the **Input Files** tab, it displays the options for remote system update. Otherwise, it only enables the standard configuration flow.
15. In the **Configuration Device** tab, click **Add Device**, select the **MT25QU02G** in the dialog box window, then click **OK**. Once that is done, the window displays the default initial partitioning for RSU:



Note: You can also use another supported flash device, because this example only needs 512Mb of flash space. The Intel Quartus Prime Programmer displays some warnings in case another supported 512 Mb or larger flash is used, but the example works.

16. Select the **FACTORY_IMAGE** entry, and click the **Edit...** button. The **Edit Partition** window pops up. Select the **Input file** as **Bitstream_1 (ghrd_1sx280lu2f50e2vg.sof)**. Change **Address Mode** to **Block** because you want to make sure you are leaving enough space for the biggest factory image you anticipate using. Set the **End Address** to **0x0090FFFF** in order to reserve 8MB for the factory image. This end address was calculated by adding 8MB to the end of the **BOOT_INFO** partition. Click **OK**.



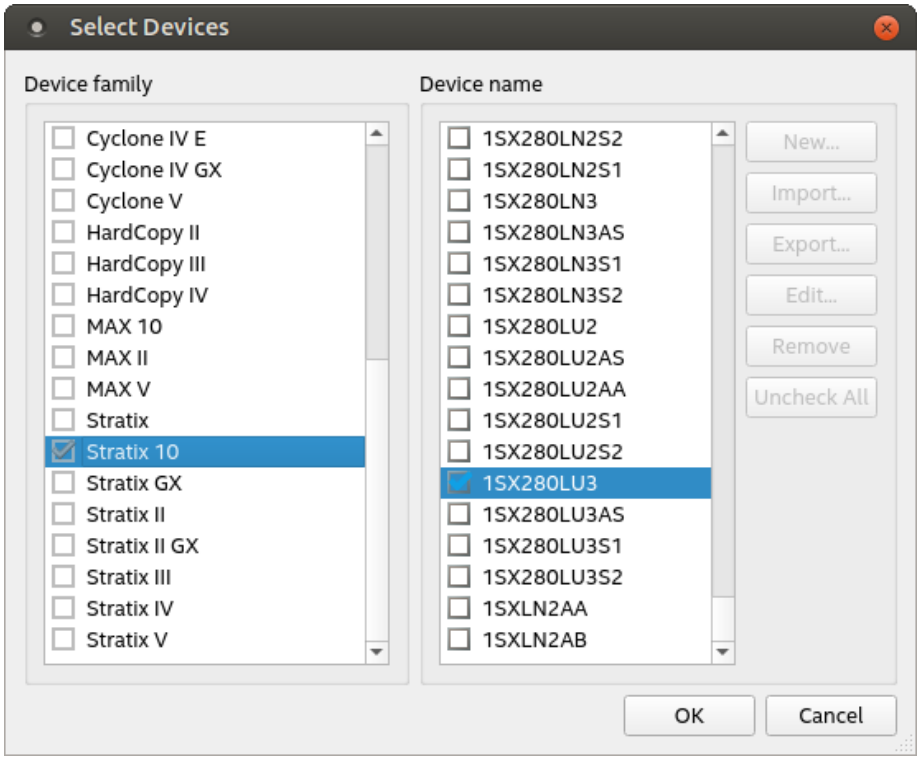
Note: The **Page** property for FACTORY_IMAGE partition must always be set to 0. This means that the FACTORY_IMAGE can retry only after all the application images failed.

17. Select the **MT25QU02G** flash device in the **Configuration Device** tab by clicking it, then click the **Add Partition...** button to open the **Add Partition** window. Leave the **Name** as **P1** and select the **Input file** as **Bitstream_2(ghrd_1sx280lu2f50e2vg.sof)**. This becomes the initial application image. Select the **Page** as **1** – this means it has the highest priority of all application images. Select the **Address Mode** as **Block** and allocate 16MB of data by setting **Start Address** = **0x01000000** and **End Address** = **0x01FFFFFF**.

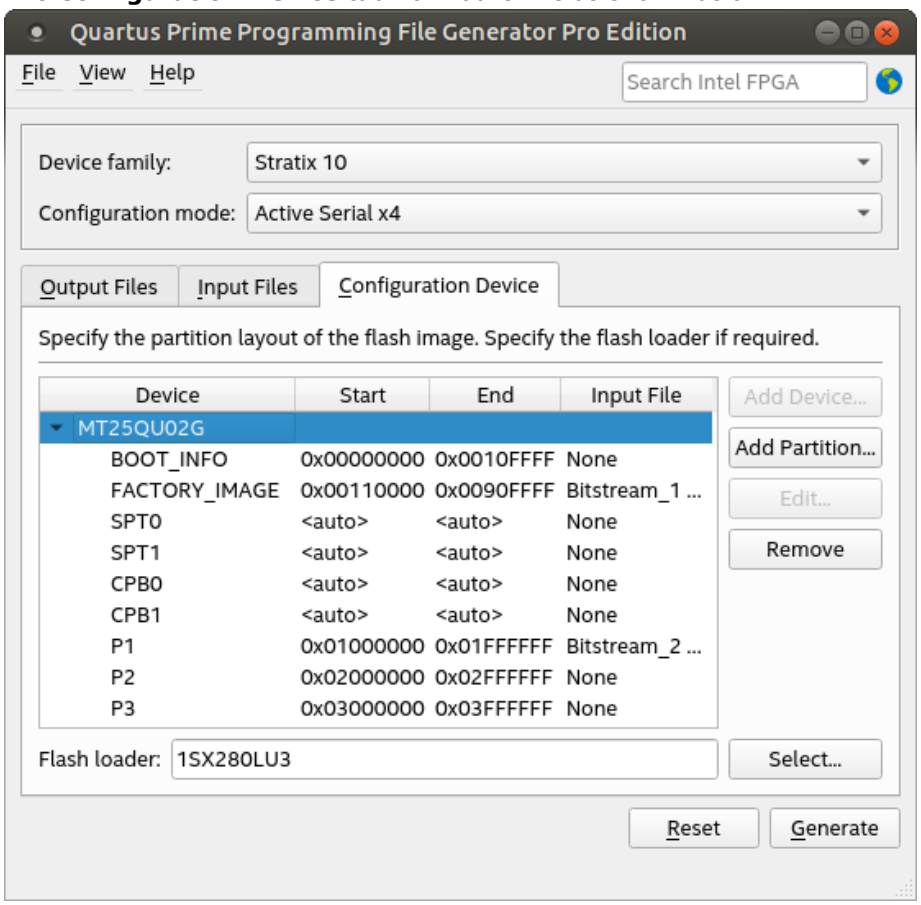
    The Page property is used by the Programming File Generator to determine the order in which images appear initially in the configuration pointer block. The highest priority is the one with Page=1, then the one with Page=2, and so on. The Programming File Generator issues an error if there are multiple partitions with the same page number, or if there are any "gaps" as in having a Page=1 then a Page=3, without a Page=2 for example.

    Only up to seven partitions can contain application images at initial flash image creation time. This limitation does not have adverse effects, as typically at creation time it is expected to have just a factory image and one application image.

18. Create two more partitions P2 and P3 using the same procedure as for the previous step, except set the **Input file** to **None**, leave **Page** unchanged (it does not matter for empty partitions) and set the start and end addresses as follows:
    - P2: **Start Address** = **0x02000000** and **End Address** = **0x02FFFFFF**.
    - P3: **Start Address** = **0x03000000** and **End Address** = **0x03FFFFFF**.
19. Click **Select ...** to select the Flash loader. The flash loader becomes part of the JIC file and is used by the Flash Programmer tool. Select the desired **Device family** and **Device name** as shown below**:**


Select Devices
Device family
Cyclone IV E
Cyclone IV GX
Cyclone V
HardCopy II
HardCopy III
HardCopy IV
MAX 10
MAX II
MAX V
Stratix
Stratix 10
Stratix GX
Stratix II
Stratix II GX
Stratix III
Stratix IV
Stratix V
Device name
1SX280LN2S2
1SX280LN2S1
1SX280LN3
1SX280LN3AS
1SX280LN3S1
1SX280LN3S2
1SX280LU2
1SX280LU2AS
1SX280LU2AA
1SX280LU2S1
1SX280LU2S2
1SX280LU3
1SX280LU3AS
1SX280LU3S1
1SX280LU3S2
1SXLN2AA
1SXLN2AB
New...
Import...
Export...
Edit...
Remove
Uncheck All
OK
Cancel

The **Configuration Device** tab now looks like as shown below:



20. Click **File ➤ Save As ..** and save the file as $TOP_FOLDER/`initial_image.pfg`. This file can be useful later, if you wanted to re-generate the initial image by using the command:

```
cd $TOP_FOLDER
quartus_pfg -c initial_image.pfg
```

Note: The created pfg file is actually an XML file which can be manually edited to replace the absolute file paths with relative file paths. You cannot directly edit the `.pfg` file for other purposes. The `.pfg` file can be opened from Programming File Generator, if changes are needed.

21. Click the **Generate** button to generate the initial flash image as $TOP_FOLDER/`initial_image.jic` and the map file as $TOP_FOLDER/`initial_image_jic.map`. A dialog box opens indicating the files were generated successfully.

6.3.5. Creating the Application Image

The RSU application images can be created from the Programming File Generator GUI tool, but the easiest way is to create them from command line. The following commands are used to create the file used in this example:

```
cd $TOP_FOLDER
mkdir -p images
rm -rf images/application2.rpd
quartus_pfg -c hw/ghrd.2/output_files/ghrd_1sx280lu2f50e2vg.sof \
    images/application2.rpd \
    -o hps_path=u-boot-socfpga/spl/u-boot-spl-dtb.hex \
    -o mode=ASX4 -o start_address=0x00000 -o bitswap=ON
```

The following application image is created: `images/application2.rpd`.

6.3.6. Creating the Factory Update Image

The factory update image can be created from the Programming File Generator GUI tool, but the easiest way is to create it from command line. The following commands are used to create the file used in this example:

```
cd $TOP_FOLDER
mkdir -p images
rm -f images/factory_update.rpd
quartus_pfg -c hw/ghrd.3/output_files/ghrd_1sx280lu2f50e2vg.sof \
    images/factory_update.rpd \
    -o hps_path=u-boot-socfpga/spl/u-boot-spl-dtb.hex \
    -o mode=ASX4 -o start_address=0x00000 -o bitswap=ON \
    -o rsu_upgrade=ON
```

The following factory update image is created: `images/factory_update.rpd`.

6.3.7. Building Linux

The following commands can be used to obtain the Linux source code and build Linux:

```
cd $TOP_FOLDER
rm -rf linux-socfpga
git clone https://github.com/altera-opensource/linux-socfpga
cd linux-socfpga
#git checkout -b test ACDS19.3_REL_GSRD_PR
git checkout -t -b origin/socfpga-4.14.130-ltsi
make clean && make mrproper
make defconfig
make -j 24 Image dtbs modules
make -j 12 modules_install INSTALL_MOD_PATH=modules_install
cd ..
```

After completing the above steps, the following files are created in the `$TOP_FOLDER/linux-socfpga` :

- `arch/arm64/boot/Image` — kernel image
- `arch/arm64/boot/dts/altera/socfpga_stratix10_socdk.dtb`— kernel device tree
- `modules_install/`— folder containing the loadable modules for the kernel

Note: The Intel SoC FPGA Linux releases on GitHub have a retention policy described at Linux Git Guidelines. At some point, the current Linux branch is removed and the above tag does not work anymore. In such an event, you can either move to the latest release of all the components, or try using the latest release of Linux, as that should also work.

6.3.8. Building ZLIB

The ZLIB is required by LIBRSU. The following steps can be used to compile it:

```
cd $TOP_FOLDER
rm -rf zlib-1.2.11
wget http://zlib.net/zlib-1.2.11.tar.gz
tar xf zlib-1.2.11.tar.gz
rm zlib-1.2.11.tar.gz
cd zlib-1.2.11/
export LD=${CROSS_COMPILE}ld
export AS=${CROSS_COMPILE}as
export CC=${CROSS_COMPILE}gcc
./configure
make
export ZLIB_PATH=`pwd`
cd ..
```

Note: The version of zlib mentioned above is the one that was tested with this release. You may want to use the latest zlib version, as it may contain updates and bug fixes.

After the above steps are completed, the following items are available:

- `~/rsu_example/zlib-1.2.11/zlib.h` — header file, used to compile files using zlib services
- `~/rsu_example/zlib-1.2.11/libz.so*` — shared objects, used to run executables linked against zlib APIs

6.3.9. Building LIBRSU and RSU Client

The following commands can be used to build the LIBRSU and the example client application:

```
cd $TOP_FOLDER
export ZLIB_PATH=`pwd`/zlib-1.2.11
rm -rf intel-rsu
git clone https://github.com/altera-opensource/intel-rsu
cd intel-rsu
#git checkout -b test ACDS119.3_REL_GSRD_PR
git checkout -t -b test origin/master
cd lib
# add -I$(ZLIB_PATH) to CFLAGS
sed -i 's/\(CFLAGS := .*\)$/\1 -I\$\(ZLIB_PATH\)/g' makefile
make
cd ..
cd example
# add -L$(ZLIB_PATH) to LDFLAGS
sed -i 's/\(LDFLAGS := .*\)$/\1 -L\$\(ZLIB_PATH\)/g' makefile
make
cd ..
cd ..
```

The following files are created:

- `$TOP_FOLDER/intel-rsu/lib/librsu.so` — shared object required at runtime for running applications using `librsu`
- `$TOP_FOLDER/intel-rsu/etc/qspi.rc` — resource file for `librsu` configuration
- `$TOP_FOLDER/intel-rsu/example/rsu_client` — example client application using `librsu`

6.3.10. Building the Root File System

A root file system is required to boot Linux. There are a lot of ways to build a root file system, depending on your specific needs. This section shows how to build a small root file system using Yocto.

1. Various packages may be needed by the build system. On a Ubuntu 16.04 machine the following command was used to install the required packages:

```
sudo apt-get install git chrpath g++ gcc make texinfo
```

2. Run the following commands to build the root file system:

```
cd $TOP_FOLDER
rm -rf yocto && mkdir yocto && cd yocto
git clone -b warrior git://git.yoctoproject.org/poky.git
git clone -b master git://github.com/kraj/meta-altera.git
source poky/oe-init-build-env ./build
echo 'MACHINE = "stratix10"' >> conf/local.conf
echo 'BBLAYERS += " ${TOPDIR}/../meta-altera "' >> conf/bblayers.conf
bitbake core-image-minimal
```

After the build completes, which can take a few hours depending on your host system processing power and Internet connection speed, the following root file system archive is created: `$TOP_FOLDER/yocto/build/tmp/deploy/images/stratix10/core-image-minimal-stratix10.tar.gz`

For more information about building Yocto, refer to Rocketboards Building Yocto or Angstrom web page. For more information about building Linux, including building the Angstrom rootfs using Yocto, refer to the Rocketboards Getting Started web page.

6.3.11. Building the SD Card

The following commands can be used to create the SD card image used in this example:

```
cd $TOP_FOLDER
sudo rm -rf sd_card && mkdir sd_card && cd sd_card
wget https://releases.rocketboards.org/release/2018.10/gsrd/\
tools/make_sdimage.py
chmod +x make_sdimage.py
# prepare the fat partition contents
mkdir fat &&  cd fat
cp ../../u-boot-socfpga/u-boot.img .
cp ../../linux-socfpga/arch/arm64/boot/Image .
cp ../../linux-socfpga/arch/arm64/boot/dts/altera/socfpga_stratix10_socdk.dtb .
cp ../../images/*.rpd .
cd ..
# prepare the rootfs partition contents
mkdir rootfs && cd rootfs
sudo tar xf ../../yocto/build/tmp/deploy/images/stratix10/\
core-image-minimal-stratix10.tar.gz
sudo rm -rf lib/modules/*
```

```
sudo cp ../../linux-socfpga/modules_install/lib/modules/*/kernel/drivers/\
firmware/stratix10-rsu.ko home/root/
sudo cp ../../images/*.rpd home/root
sudo cp ../../intel-rsu/example/rsu_client home/root/
sudo cp ../../intel-rsu/lib/librsu.so lib/
sudo cp ../../intel-rsu/etc/qspi.rc etc/librsu.rc
sudo cp ../../zlib-1.2.11/libz.so* lib/
cd ..
# create sd card image
sudo ./make_sdimage.py -f \
-P fat/*,num=1,format=vfat,size=100M \
-P rootfs/*,num=2,format=ext3,size=100M \
-s 256M \
-n sdcard_s10_rsu.img
cd ..
```

This creates the SD card image as `$TOP_FOLDER/sd_card/sdcard_s10_rsu.img`.

The following items are included in the rootfs on the SD card:

- U-Boot
- Linux RSU driver
- ZLIB shared objects
- LIBRSU shared objects and resource files
- RSU client application
- Application image
- Factory update image

6.4. Flashing the Binaries

6.4.1. Flashing the Initial RSU Image to QSPI

1. Make sure to install the QSPI SDM bootcard on the Intel Stratix 10 SoC Development Kit
2. Configure the Intel Stratix 10 SoC Development Kit as follows:
   - SW1: 1:OFF, rest:ON
   - SW2: 1:ON 2:ON 3: ON 4: OFF
   - SW3: all OFF
   - SW4: 1:ON 2:OFF 3:OFF 4:ON
3. Run the following command to write the image to SDM QSPI by using the command line version of the Intel Quartus Prime Programmer:

```
cd $TOP_FOLDER
quartus_pgm  -c 1 -m jtag -o "pvi;./initial_image.jic"
```

6.4.2. Writing the SD Card Image

1. Write the SD card image `$TOP_FOLDER/sd_card/sdcard_s10_rsu.img` to a microSD card. You can use an USB micro SD card writer and the Linux `dd` command on your host PC to achieve this. Exercise caution when using the `dd` command, as incorrect usage can lead to your host Linux system becoming corrupted and non-bootable.
2. Insert the micro SD card in the slot on the Intel Stratix 10 version HPS daughtercard.

6.5. Exercising the RSU Client

6.5.1. Basic Operation

This section demonstrates how to use the RSU client to perform the following basic operations:

- Querying the RSU status.
- Querying the number of slots and the information about them.
- Adding a new application image.
- Verifying that the application image was written correctly.
- Requesting a specific application image to be loaded.

1. Power cycle the board and let Linux boot.
2. Log in using 'root' as user name and an empty password

```
Tue Sep 24 23:28:28 UTC 2019
INIT: Entering runlevel: 5
Configuring network interfaces...
[    2.682073] socfpga-dwmac ff800000.ethernet eth0: IEEE 1588-2008
Advanced Timestamp supported
[    2.690867] socfpga-dwmac ff800000.ethernet eth0: registered PTP clock
udhcpc: started, v1.30.1
[    6.796166] socfpga-dwmac ff800000.ethernet eth0: Link is Up - 1Gbps/
Full - flow control rx/tx
udhcpc: sending discover
udhcpc: sending select for 192.168.1.115
udhcpc: lease of 192.168.1.115 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
done.
Starting syslogd/klogd: done

Poky (Yocto Project Reference Distro) 2.7.1 stratix10 /dev/ttyS0

stratix10 login: root
root@stratix10:~#
```

3. Load the stratix10-rsu kernel driver by running the command:

```
root@stratix10:~# insmod stratix10-rsu.ko
```

4. Run the `rsu_client` without parameters, to display its help message:

```
root@stratix10:~# ./rsu_client
--- RSU app usage ---
-c|--count                          get the number of slots
-l|--list slot_num                  list the attribute info from the selected
slot
-z|--size slot_num                  get the slot size in bytes
```

```
-p|--priority slot_num            get the priority of the selected slot
-E|--enable slot_num              set the selected slot as the highest
priority
-D|--disable slot_num             disable selected slot but to not erase it
-r|--request slot_num             request the selected slot to be loaded
after the next reboot
-R|--request-factory              request the factory image to be loaded
after the next reboot
-e|--erase slot_num               erase app image from the selected slot
-a|--add file_name -s|--slot slot_num  add a new app image to the selected
slot
-u|--add-factory-update file_name -s|--slot slot_num  add a new factory
update image to the selected slot
-A|--add-raw file_name -s|--slot slot_num  add a new raw image to the
selected slot
-v|--verify file_name -s|--slot slot_num  verify app image on the selected
slot
-V|--verify-raw file_name -s|--slot slot_num  verify raw image on the
selected slot
-f|--copy file_name -s|--slot slot_num  read the data in a selected slot
then write to a file
-g|--log                          print the status log
-n|--notify value                 report software state
-C|--clear-error-status           clear errors from the log
-Z|--reset-retry-counter          reset current retry counter
-h|--help                         show usage message
```

5. Exercise the `rsu_client` command that displays the current status, it shows the application image from slot 0 (partition P1) is loaded with no errors:

```
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000001000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

6. Run the RSU client commands that display information about the slots:

```
root@stratix10:~# ./rsu_client --count
number of slots is 3
Operation completed
root@stratix10:~# ./rsu_client --list 0
      NAME: P1
    OFFSET: 0x0000000001000000
      SIZE: 0x01000000
  PRIORITY: 1
Operation completed
root@stratix10:~# ./rsu_client --list 1
      NAME: P2
    OFFSET: 0x0000000002000000
      SIZE: 0x01000000
  PRIORITY: [disabled]
Operation completed
root@stratix10:~# ./rsu_client --list 2
      NAME: P3
    OFFSET: 0x0000000003000000
      SIZE: 0x01000000
  PRIORITY: [disabled]
Operation completed
```

7. Add the `application2.rpd` application image to slot 1 (partition P2):

```
root@stratix10:~# ./rsu_client --add application2.rpd --slot 1
Operation completed
```

8. Verify that the application image was written correctly to flash:

```
root@stratix10:~# ./rsu_client --verify application2.rpd --slot 1
Operation completed
```

9. List again the slots, it shows the most recently written partition P2 image having the highest priority (lowest priority number that is):

```
root@stratix10:~# ./rsu_client --list 0
      NAME: P1
    OFFSET: 0x0000000001000000
      SIZE: 0x01000000
  PRIORITY: 2
Operation completed
root@stratix10:~# ./rsu_client --list 1
      NAME: P2
    OFFSET: 0x0000000002000000
      SIZE: 0x01000000
  PRIORITY: 1
Operation completed
root@stratix10:~# ./rsu_client --list 2
      NAME: P3
    OFFSET: 0x0000000003000000
      SIZE: 0x01000000
  PRIORITY: [disabled]
Operation completed
```

10. Power cycle the board, boot Linux, load the RSU module and display the status – it shows the image from partition P2 running:

```
root@stratix10:~# insmod stratix10-rsu.ko
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000002000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

11. Instruct the RSU client to request slot 0 (partition P1) from SDM on next `reboot` command:

```
root@stratix10:~# ./rsu_client --request 0
Operation completed
```

12. Restart Linux by running the `reboot` command:

```
root@stratix10:~# reboot
```

13. Log into Linux, load the kernel driver and display the RSU status:

```
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000001000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

The status shows that the image from partition P1 was loaded, as requested.

6.5.2. Watchdog Timeout and `max retry` Operation

This section uses the RSU client to demonstrate the following:

- RSU handling of watchdog timeouts.
- `max retry` feature, which allows each image up to three times to load.
- RSU notify, which allows the HPS software state to be reported and retrieved after a watchdog timeout.
- Clearing the RSU status error fields.
- Resetting the current `retry` counter value.

Note: The commands listed in this section rely on the commands from the *Basic Operation*section running first, specifically adding image P2 to the flash.

1. Power cycle the board, boot Linux, load the RSU module and display the status – it shows the P2 image running, as it is the highest priority:

```
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000002000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

   What do the version fields mean:

   - `Version field` is `0x00000101`—both decision firmware and application firmware RSU interface versions are 1, and the system supports the `max retry` feature.
   - `Retry` counter is `0x00000000`—first attempt to load this image.
   - `State` is `0x00000000`—No errors to report

2. Enable the watchdog but do not service it, as this produces a timeout, and restarts Linux:

```
root@stratix10:~# echo "something" > /dev/watchdog
[  603.649746] watchdog: watchdog0: watchdog did not stop!
```

3. Wait for Linux to restart after the watchdog timeout, then load the RSU kernel driver and display the log:

```
root@stratix10:~# insmod stratix10-rsu.ko
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000002000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000001
Operation completed
```

   The same P2 image is loaded, but the `retry` counter value is now one, which means this is the second retry for this image to be loaded. The state and error fields are all clear, as this is not considered an error.

4. Enable another watchdog timeout and wait for Linux to restart. After the restart, query the RSU log and observe that the `retry` counter is now two:

```
root@stratix10:~# insmod stratix10-rsu.ko
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000002000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000002
Operation completed
```

5. Notify the SDM of the HPS execution stage as a 16bit number:

```
root@stratix10:~# ./rsu_client --notify 0x1234
Operation completed
```

6. Enable another watchdog timeout and watch for Linux to restart. After the restart, query the RSU log:

```
root@stratix10:~# insmod stratix10-rsu.ko
root@stratix10:~# ./rsu_client --log
      VERSION: 0x0ACF0101
        STATE: 0xF0061234
CURRENT IMAGE: 0x0000000001000000
   FAIL IMAGE: 0x0000000002000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation complete
```

   The SDM loaded the next application image in the CPB (P1), and it reports that the image P2 failed. The state indicates that a watchdog timeout occurred (upper 16 bits = `0xF006`) and that the notify value reported by HPS software was `0x1234`. The upper 16 bits of the version are set to `0x0ACF` which means the previous error was reported by the application image firmware. For more information, refer to RSU Status and Error Codes *on page 78*.

7. Clear the errors, and display again the status - it shows no errors:

```
root@stratix10:~# ./rsu_client --clear-error-status
Operation completed
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000001000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

8. Enable a watchdog timeout and display the status - it shows a `retry` counter value of one:

```
root@stratix10:~# insmod stratix10-rsu.ko
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000001000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
```

```
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000001
Operation completed
```

9. Use the RSU client to reset the current retry counter value to zero, and query the status again to confirm it:

```
root@stratix10:~# ./rsu_client --reset-retry-counter
Operation completed
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000001000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

Related Information

Basic Operation on page 53

6.5.3. Updating the Factory Image

This section demonstrates how to use the RSU client to update the factory image.

Note: The commands listed in this section rely on the commands from the *Basic Operation* section running first, specifically adding an application image to the P2 flash partition.

1. Power cycle the board, boot Linux, load the RSU module and display the status – it shows the P2 image running, as it is the highest priority:

```
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000002000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

2. Confirm that slot 2 (partition P3) is not used, write the factory update image to it, and verify it was written correctly:

```
root@stratix10:~# ./rsu_client --list 2
      NAME: P3
    OFFSET: 0x0000000003000000
      SIZE: 0x01000000
  PRIORITY: [disabled]
Operation completed
root@stratix10:~# ./rsu_client --add-factory-update factory_update.rpd --
slot 2
Operation completed
root@stratix10:~# ./rsu_client --verify factory_update.rpd --slot 2
Operation completed
```

3. Confirm that slot 2 is now the highest priority in the CPB:

```
root@stratix10:~# ./rsu_client --list 2
      NAME: P3
    OFFSET: 0x0000000003000000
```

```
      SIZE: 0x01000000
  PRIORITY: 1
Operation completed
```

4. Instruct the RSU client to request slot 2 (partition P3) to be loaded on next `reboot` command:

```
root@stratix10:~# ./rsu_client --request 2
Operation completed
```

5. Restart Linux by running the `reboot` command:

```
root@stratix10:~# reboot
```

6. Linux shuts down, then the factory update image updates the decision firmware, decision firmware data and factory image in flash. Then it removes itself from the CPB and loads the now highest priority image in the CPB. Confirm that P2 is now loaded and P3 is disabled:

```
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000002000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
root@stratix10:~# ./rsu_client --list 2
      NAME: P3
    OFFSET: 0x0000000003000000
      SIZE: 0x01000000
  PRIORITY: [disabled]
Operation completed
```

Related Information

Basic Operation on page 53

6.5.4. Fallback on Flash Corruption

This section uses the RSU client to demonstrate falling back in case of configuration errors caused by flash corruption.

Note: The commands listed in this section rely on the commands from the *Basic Operation* section running first, specifically adding image P2 to the flash.

1. Power cycle the board, boot Linux, load the RSU module and display the status – it shows the P2 image running, as it is the highest priority:

```
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000002000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

What do the version fields mean:

- `Version field` is `0x00000101`—both decision firmware and application firmware RSU interface versions are 1, and the system supports the `max retry` feature.
- `Retry` counter is `0x00000000`—first attempt to load this image.
- `State` is `0x00000000`—No errors to report

2. Erase slot 1, which also takes it out of CPB:

```
root@stratix10:~# ./rsu_client --erase 1
Operation completed
```

3. Create a file with random data, and write it to the P2 slot:

```
root@stratix10:~# dd if=/dev/urandom of=corrupt.rpd bs=1M count=1
1+0 records in
1+0 records out
root@stratix10:~# ./rsu_client --add-raw corrupt.rpd --slot 1
Operation completed
```

4. Enable the P2 slot, which puts it as the highest priority in the CPB:

```
root@stratix10:~# ./rsu_client --enable 1
Operation completed
```

5. Confirm that P2 is now the highest priority in the CPB:

```
root@stratix10:~# ./rsu_client --list 1
      NAME: P2
    OFFSET: 0x0000000002000000
      SIZE: 0x01000000
  PRIORITY: 1
Operation completed
```

6. Power cycle the board, boot Linux, load RSU driver and query the RSU log:

```
root@stratix10:~# insmod stratix10-rsu.ko
root@stratix10:~# ./rsu_client --log
      VERSION: 0x0DCF0101
        STATE: 0xF004D003
CURRENT IMAGE: 0x0000000001000000
   FAIL IMAGE: 0x0000000002000000
    ERROR LOC: 0x00000000
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

The current image is P1, and the P2 shows as failed. The top 16bit of the version field are set as `0x0DCF` which means the error was caused by the decision firmware, because it was not able to load the image. The top 16 bits of the state field are set to `0xF004`, which indicates an internal error. For more information, refer to RSU Status and Error Codes *on page 78*.

7. Clear the error status and display the log again to confirm it was cleared:

```
root@stratix10:~# ./rsu_client --clear-error-status
Operation completed
root@stratix10:~# ./rsu_client --log
      VERSION: 0x00000101
        STATE: 0x00000000
CURRENT IMAGE: 0x0000000001000000
   FAIL IMAGE: 0x0000000000000000
    ERROR LOC: 0x00000000
```

```
ERROR DETAILS: 0x00000000
RETRY COUNTER: 0x00000000
Operation completed
```

Related Information

Basic Operation on page 53

6.6. Exercising U-Boot RSU Commands

6.6.1. Basic Operation

This section demonstrates how to use U-Boot to perform the following basic operations:

- Querying the RSU status.
- Querying the number of slots and the information about them.
- Adding a new application image.
- Verifying that an application image was written correctly.
- Requesting a specific application image to be loaded.

Note: This section assumes that the flash contains the initial RSU image. If that is not true, you need to re-flash the initial image, as shown in the *Flashing the Initial RSU Image to QSPI* .

1. Power cycle the board and press any key when prompted, to get to the U-Boot command prompt:

```
U-Boot 2019.04-00201-g82d3e01 (Oct 04 2019 - 13:23:59
+0000)socfpga_stratix10

CPU:   Intel FPGA SoCFPGA Platform (ARMv8 64bit Cortex-A53)
Model: SoCFPGA Stratix 10 SoCDK
DRAM:  4 GiB
MMC:   dwmmc0@ff808000: 0
Loading Environment from MMC... *** Warning - bad CRC, using default
environment

In:    serial0@ffc02000
Out:   serial0@ffc02000
Err:   serial0@ffc02000
Net:
Warning: ethernet@ff800000 (eth0) using random MAC address -
02:18:a0:c1:2c:b8
eth0: ethernet@ff800000
Hit any key to stop autoboot:  0
SOCFPGA_STRATIX10 #
```

2. Run the `rsu` command without parameters, to display its help message, and usage options:

```
SOCFPGA_STRATIX10 # rsu
rsu - SoCFPGA Stratix10 SoC Remote System Update

Usage:
rsu dtb   - Update Linux DTB qspi-boot parition offset with spt0 value
list  - List down the available bitstreams in flash
slot_by_name <name> - find slot by name and display the slot number
slot_count - display the slot count
slot_disable <slot> - remove slot from CPB
slot_enable <slot> - make slot the highest priority
```

```
slot_erase <slot> - erase slot
slot_get_info <slot> - display slot information
slot_load <slot> - load slot immediately
slot_load_factory - load factory immediately
slot_priority <slot> - display slot priority
slot_program_buf <slot> <buffer> <size> - program buffer into slot, and
make it highest priority
slot_program_buf_raw <slot> <buffer> <size> - program raw buffer into slot
slot_program_factory_update_buf <slot> <buffer> <size> - program factory
update buffer into slot, and make it highest priority
slot_rename <slot> <name> - rename slot
slot_size <slot> - display slot size
slot_verify_buf <slot> <buffer> <size> - verify slot contents against buffer
slot_verify_buf_raw <slot> <buffer> <size> - verify slot contents against
raw buffer
status_log - display RSU status
update <flash_offset> - Initiate firmware to load bitstream as specified by
flash_offset
notify <value> - Let SDM know the current state of HPS software
clear_error_status - clear the RSU error status
reset_retry_counter - reset the RSU retry counter
```

3. Run the `rsu list` command to display the RSU partitions, CPBs, the currently running image and the status:

```
SOCFPGA_STRATIX10 # rsu list
RSU: Remote System Update Status
Current Image    : 0x01000000
Last Fail Image   : 0x00000000
State             : 0x00000000
Version           : 0x00000101
Error location    : 0x00000000
Error details    : 0x00000000
Retry counter    : 0x00000000
RSU: Sub-partition table 0 offset 0x00910000
RSU: Sub-partition table 1 offset 0x00918000
SF: Detected mt25qu02g with page size 256 Bytes, erase size 4 KiB, total
256 MiB
RSU: Sub-partition table content
       BOOT_INFO    Offset: 0x0000000000000000    Length: 0x00110000
Flag : 0x00000003
   FACTORY_IMAGE    Offset: 0x0000000000110000    Length: 0x00800000
Flag : 0x00000003
            P1    Offset: 0x0000000001000000    Length: 0x01000000
Flag : 0x00000000
          SPT0    Offset: 0x0000000000910000    Length: 0x00008000
Flag : 0x00000001
          SPT1    Offset: 0x0000000000918000    Length: 0x00008000
Flag : 0x00000001
          CPB0    Offset: 0x0000000000920000    Length: 0x00008000
Flag : 0x00000001
          CPB1    Offset: 0x0000000000928000    Length: 0x00008000
Flag : 0x00000001
            P2    Offset: 0x0000000002000000    Length: 0x01000000
Flag : 0x00000000
            P3    Offset: 0x0000000003000000    Length: 0x01000000
Flag : 0x00000000
RSU: CMF pointer block offset 0x00920000
RSU: CMF pointer block's image pointer list
Priority 1 Offset: 0x0000000001000000 nslot: 0
```

Note: The `rsu list` U-Boot command does not have a RSU client equivalent. Instead, the same information can be retrieved using other commands, as shown next.

4. Run the `rsu status_log` command to display the RSU status:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x01000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

   Application image P1 is loaded, as it is the highest priority in the CPB. There are no errors.

5. Display information about the slots:

```
SOCFPGA_STRATIX10 # rsu slot_count
Number of slots = 3.
SOCFPGA_STRATIX10 # rsu slot_get_info 0
NAME: P1
OFFSET: 0x0000000001000000
SIZE: 0x01000000
PRIORITY: 1
SOCFPGA_STRATIX10 # rsu slot_get_info 1
NAME: P2
OFFSET: 0x0000000002000000
SIZE: 0x01000000
PRIORITY: [disabled]
SOCFPGA_STRATIX10 # rsu slot_get_info 2
NAME: P3
OFFSET: 0x0000000003000000
SIZE: 0x01000000
PRIORITY: [disabled]
SOCFPGA_STRATIX10 # rsu slot_size 0
Slot 0 size = 16777216.
SOCFPGA_STRATIX10 # rsu slot_size 1
Slot 1 size = 16777216.
SOCFPGA_STRATIX10 # rsu slot_size 2
Slot 2 size = 16777216.
```

6. Add the `application2.rpd` image to slot 1:

```
SOCFPGA_STRATIX10 # load mmc 0:1 $loadaddr application2.rpd
3338240 bytes read in 152 ms (20.9 MiB/s)
SOCFPGA_STRATIX10 # rsu slot_program_buf 1 $loadaddr $filesize
Slot 1 was programmed with buffer=0x0000000002000000 size=3338240.
```

7. Verify that the application image was written correctly:

```
SOCFPGA_STRATIX10 # rsu slot_verify_buf 1 $loadaddr $filesize
Slot 1 was verified with buffer=0x0000000002000000 size=3338240.
```

8. Confirm that slot 1 (partition P2) contains now the highest priority image:

```
SOCFPGA_STRATIX10 # rsu slot_get_info 1
NAME: P2
OFFSET: 0x0000000002000000
SIZE: 0x01000000
PRIORITY: 1
```

9. Power-cycle the board, stop U-Boot and check the RSU status log:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x02000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
```

```
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

The application image from slot 1 (partition P2) was loaded, because it is marked as the highest priority in the CPB.

10. Load the application image from slot 0 (partition P1) by running any of the following two commands:

```
SOCFPGA_STRATIX10 # rsu update 0x01000000
RSU: RSU update to 0x0000000001000000
```

or

```
SOCFPGA_STRATIX10 # rsu slot_load 0
Slot 0 loading.
```

11. Load the newly requested image. Stop at U-Boot prompt and check the status log to confirm it:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x01000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

Note: In U-Boot, the effect of requesting a specific image is immediate. On Linux, it only takes effect on the next `reboot` command.

Related Information

- Flashing the Initial RSU Image to QSPI on page 52
- Watchdog and Max Retry Operation on page 64
- Updating the Factory Image on page 66
- Fallback on Flash Corruption on page 67

6.6.2. Watchdog and Max Retry Operation

This section uses U-Boot to demonstrate the following:

- RSU handling of watchdog timeouts.
- `max retry` feature, which allows each image up to three times to load.
- RSU notify, which allows the HPS software state to be reported before and retrieved after a watchdog timeout.
- Clearing the RSU status error fields.
- Resetting the current retry counter value.

Note: The commands listed in this section rely on the commands from the *Basic Operation* section running first, specifically adding an application image to the P2 flash partition.

1. Power-cycle the board, stop U-Boot and check the RSU status log:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x02000000
Last Fail Image : 0x00000000
```

```
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

The application image from slot 1 (partition P2) was loaded, since it is marked as the highest priority in the CPB.

What do the fields mean:

- `Version field` is `0x00000101`—both decision firmware and application firmware RSU interface versions are 1, and the system supports the `max retry` feature.
- `Retry` counter is `0x00000000`—first attempt to load this image.
- `State` is `0x00000000`—No errors to report

2. Enable the watchdog without servicing it, in order to cause a timeout:

```
SOCFPGA_STRATIX10 # mw.l 0xffd00200 1
```

3. After a minute, the watchdog times out, and SDM reloads the same application image, since the `max retry` parameter is set to three. Look at the U-Boot console and check the status log:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x02000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000001
```

The `retry` counter value is now one, which means this is the second retry for this image to be loaded.

4. Enable another watchdog timeout. At the U-Boot prompt, query the RSU log and observe that the `retry` counter is now two:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x02000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000002
```

5. Use the notify command to let SDM know the state of HPS software as a 16bit value:

```
SOCFPGA_STRATIX10 # rsu notify 0x1234
```

6. Enable a watchdog timeout one more time and display the RSU status log after the restart:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x01000000
Last Fail Image : 0x02000000
State           : 0xf0061234
Version         : 0x0acf0101
```

```
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

The SDM loaded the next application image in the CPB (P1), and it reports that the image P2 failed. The state indicates that a watchdog timeout occurred (upper 16 bits = 0xF006) and that the notify value reported by HPS software was 0x1234. The upper 16 bits of version are set to 0x0ACF which means the previous error was reported by the application image firmware. For more information, refer to RSU Status and Error Codes *on page 78*.

7. Clear the errors and display the status - it shows no errors:

```
SOCFPGA_STRATIX10 # rsu clear_error_status
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x01000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

8. Enable a watchdog timeout, boot to U-Boot, and display the status - it shows the `retry` counter is one:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x01000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000001
```

9. Reset the current `retry` counter value to zero and query the status again to confirm it:

```
SOCFPGA_STRATIX10 # rsu reset_retry_counter
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x01000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

6.6.3. Updating the Factory Image

This section demonstrates how to use U-Boot to update the factory image.

Note: The commands listed in this section rely on the commands from the *Basic Operation* section running first, specifically adding an application image to the P2 flash partition.

1. Power-cycle the board, stop U-Boot and check the RSU status log:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x02000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
```

```
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x0000000a
```

The application image from slot 1 (partition P2) was loaded, because it is marked as the highest priority in the CPB.

2. Confirm that slot 2 is not used, write the factory update image to it, and verify it was written correctly:

```
SOCFPGA_STRATIX10 # rsu slot_get_info 2
NAME: P3
OFFSET: 0x0000000003000000
SIZE: 0x01000000
PRIORITY: [disabled]
SOCFPGA_STRATIX10 # load mmc 0:1 $loadaddr factory_update.rpd
3489792 bytes read in 158 ms (21.1 MiB/s)
SOCFPGA_STRATIX10 # rsu slot_program_factory_update_buf 2 $loadaddr
$filesize
Slot 2 was programmed with buffer=0x0000000002000000 size=3489792.
SOCFPGA_STRATIX10 # rsu slot_verify_buf 2 $loadaddr
$filesize
Slot 2 was verified with buffer=0x0000000002000000 size=3489792.
```

3. Confirm that slot 2 is now the highest priority in the CPB:

```
SOCFPGA_STRATIX10 # rsu slot_get_info 2
NAME: P3
OFFSET: 0x0000000003000000
SIZE: 0x01000000
PRIORITY: 1
```

4. Instruct the SDM to load the factory update image from slot 2:

```
SOCFPGA_STRATIX10 # rsu slot_load 2
Slot 2 loading.
```

5. The factory update image runs and updates the decision firmware, decision firmware data and factory image in flash. Then it removes itself from the CPB and loads the now highest priority image in the CPB. At the U-Boot prompt, confirm that P2 is now loaded and P3 is disabled:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x02000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
SOCFPGA_STRATIX10 # rsu slot_get_info 2
NAME: P3
OFFSET: 0x0000000003000000
SIZE: 0x01000000
PRIORITY: [disabled]
```

6.6.4. Fallback on Flash Corruption

This section uses U-Boot to demonstrate falling back in case of configuration errors caused by flash corruption.

Note: The commands listed in this section rely on the commands from the *Basic Operation* section running first, specifically adding an application image to the P2 flash partition.

1. Power-cycle the board, stop U-Boot and check the RSU status log:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x02000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

   The application image from slot 1 (partition P2) was loaded, because it is marked as the highest priority in the CPB.

   What do the fields mean:

   - `Version field` is `0x00000101`—both decision firmware and application firmware RSU interface versions are 1, and the system supports the `max retry` feature.
   - `Retry` counter is `0x00000000`—first attempt to load this image.
   - `State` is `0x00000000`—No errors to report

2. Corrupt the image in the slot 1 by erasing some of it:

```
SOCFPGA_STRATIX10 # sf probe
SF: Detected mt25qu02g with page size 256 Bytes, erase size 4 KiB, total
256 MiB
SOCFPGA_STRATIX10 # sf erase 0x02000000 0x4000
SF: 16384 bytes @ 0x2000000 Erased: OK
```

3. Power cycle the board, stop at U-Boot prompt, and query the RSU log:

```
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x01000000
Last Fail Image : 0x02000000
State           : 0xf004d003
Version         : 0x0dcf0101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

   The current image is P1, and the P2 shows as a failure. Note that SDM tried to load the image three times from flash, as specified by the `max retry` option. The top 16 bits of the version field are set as `0x0DCF` which means the error was caused reported by the decision firmware, as it was not able to load the image. The top 16 bits of the state field are set to `0xF004`, which indicate an internal error. For more information, refer to RSU Status and Error Codes *on page 78*.

4. Clear the error status and display the log again to confirm it was cleared:

```
SOCFPGA_STRATIX10 # rsu clear_error_status
SOCFPGA_STRATIX10 # rsu status_log
Current Image   : 0x01000000
Last Fail Image : 0x00000000
State           : 0x00000000
Version         : 0x00000101
Error location  : 0x00000000
Error details   : 0x00000000
Retry counter   : 0x00000000
```

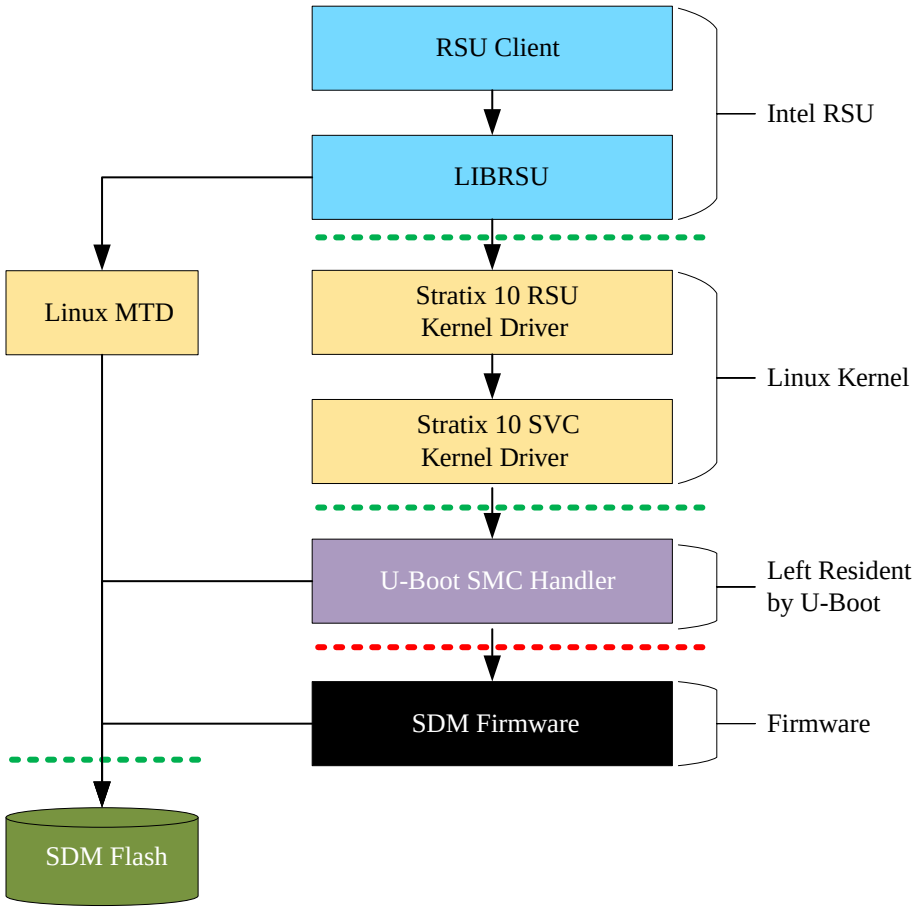
7. Version Compatibility Considerations

7.1. API Version Compatibility

The SDM provides an API which is used by U-Boot. U-Boot exports the services offered by the SDM API to the Linux kernel through the SMC interface. The Linux kernel has drivers which then export the services offered through the SMC interface to Linux applications. Each of these interfaces are different, although they offer similar functionality. The following figure identifies these interfaces.

Note: SDM Firmware, U-Boot, and LIBRSU directly access the SDM Flash, so an interface is shown there, as well.

Figure 11. API Version Compatibility



The API version compatibility rules are:

- The SDM Firmware and U-Boot interface (shown as a red line above) are allowed to change between versions. Although efforts are made to keep the changes compatible, you must always use a U-Boot (FSBL and SSBL) version compatible with the SDM Firmware version. The FSBL is not shown in the above figure for brevity. It is used only to load SSBL and it does not remain resident.
- All other interfaces (shown as green lines above) either never change, or always change in a way in which any version of a service client should work with any version of a service provider.

The following table presents the versions for the software running on HPS.

Note: The "Current Branch" column is valid for the Intel Quartus Prime Pro Edition software version 19.3 release. In the future, both U-Boot and Linux will use newer branches.

Table 10. Software Versions

Item	Git Tree	Current Branch	Tag
U-Boot	https://github.com/altera-opensource/u-boot-socfpga	`origin/socfpga_v2019.04`	Pre 19.3: `ACDS<major>.<minor>_REL_S10_GSRD_PR` 19.3+: `ACDS<major>.<minor>_REL_GSRD_PR`
Linux Kernel	https://github.com/altera-opensource/linux-socfpga	`origin/socfpga-4.14.130-ltsi`	`ACDS<major>.<minor>_REL_GSRD_PR`
Intel RSU	https://github.com/altera-opensource/intel-rsu	`origin/master`	`ACDS<major>.<minor>_REL_GSRD_PR`

Note: Intel policy specifies that only the current and immediately previous U-Boot and Linux branches are kept on GitHub. As a new branch is supported, the oldest branch is removed. You must keep a local copy of the sources used to build your binaries if you need to reproduce the build or make changes in the future.

The SDM firmware is added to the bitstream by the Intel Quartus Prime Programming File Generator, which is part of a Intel Quartus Prime release. An Intel Quartus Prime release is identified by its <major>.<minor> number, such as "18.1" or "19.3". The correct version of U-Boot to use with a certain Intel Quartus Prime Programming File Generator release is given by the tag from the above table. Both FSBL and SSBL must be built from the same version of U-Boot.

It can be sometimes useful to use the latest on the U-Boot branch instead of the official GSRD tag described above to get access to new functionality and bug fixes which are posted on GitHub between releases. Care must be exercised when doing so, as once a new Intel Quartus Prime software version is released, the interface between SDM firmware and U-Boot may change. Intel recommends always using the GSRD official tag, unless explicitly instructed to do otherwise.

Updates to Intel Quartus Prime can sometimes be released which can impact the interface between SDM firmware and U-Boot. In such cases, instructions are provided along with the update to enable the selection of the proper U-Boot version.

Note: Typically the same version of tools is used for both Intel Quartus Prime (which creates the FPGA configuration data as a SOF file) and Intel Quartus Prime Programming File Generator (which creates the bitstreams containing the SDM firmware). Generally, you may be able to generate a bitstream using a newer version of Intel Quartus Prime Programming File Generator with an older version of SOF. In this case, your SDM firmware may have a newer version than your FPGA configuration data. However, this is not guaranteed to always work.

7.2. API Version Compatibility Testing

The interface between U-Boot and Linux is designed as version compatible. That is, any version of Linux can work with any version of U-Boot.

Not all combinations of versions are tested, but sufficient tests are performed before each release to ensure that incompatibilities that can be introduced in the U-Boot/ Linux interface or the QSPI flash interface are caught and fixed before each release.

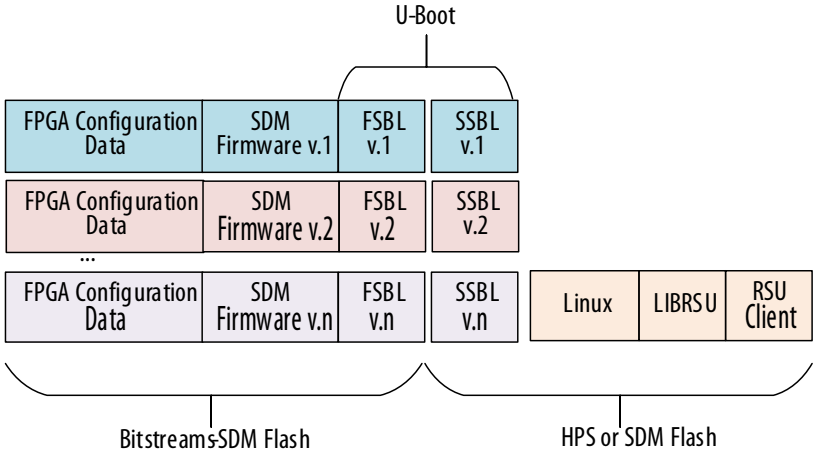
7.3. Using Multiple Intel Quartus Prime Software Versions for Bitstreams

As mentioned, the version of U-Boot (FSBL & SSBL) needs to be compatible with the version of SDM firmware being used, which is tied to the Intel Quartus Prime software version.

If different bitstreams (application or factory) are generated with different Intel Quartus Prime software versions, then each of these bitstreams need to use a compatible version of FSBL. Furthermore each of the FSBLs requires a compatible SSBL. This results in the requirement of having a separate SSBL for each bitstream.

A typical system using multiple Intel Quartus Prime software versions for the bitstreams can look as shown in the following figure:

Figure 12. RSU System Using Multiple Intel Quartus Prime Software Versions



Some users may even want to duplicate Linux, LIBRSU and RSU Client for each bitstream, but that is not required, as those interfaces are forward and backward compatible. It is only the U-Boot (FSBL and SSBL) which needs to be compatible with the SDM firmware API.

7.4. Updating U-Boot to Support Multiple Intel Quartus Prime Software Versions

By default U-Boot does not support loading a separate SSBL for each bitstream. Instead, the SSBL location is fixed as an address when loading SSBL from QSPI, and as a file name when loading SSBL from a FAT partition.

This section gives guidance on how you may update U-Boot to support multiple Intel Quartus Prime software versions by having a different SSBL for each bitstream.

7.4.1. Using Multiple SSBLs with SD/MMC

This section presents the recommended approach to support multiple SSBLs when they are stored in the HPS SD/MMC.

No changes required in the Programming File Generator, at initial image creation time.

Changes required in FSBL:

- Query SDM and read flash to determine all the partition information, and the currently running bitstream location in flash.
- Look up the currently running bitstream location in the list of the SPT partitions to determine the partition containing the currently running bitstream.
- Instead of using a hardcoded file name for the SSBL, use a name derived from the name of the partition containing the currently running bitstream.

If the SSBL is configured with read-only environment, then no SSBL code changes are needed. If the SSBL is configured with a modifiable environment, then the following changes are recommended:

- Make sure there is enough space between the MBR and the first SD card partition to store the environment for all the bitstreams.
- Change the environment location from the hardcoded value to an address which is different for each bitstream.

The recommended application image update procedure also changes:

1. Use LIBRSU to erase the application image partition. This also disables it, removing it from the CPB.
2. Replace the corresponding U-Boot image file on the FAT partition with the new version, using a filename derived from the partition name.
3. In the case of a modifiable environment, erase the sector(s) associated with the application image partition.
4. Use LIBRSU to write the new application image. This also enables it, putting it as the highest priority in the CPB.

7.4.2. Using Multiple SSBLs with QSPI

This section presents the recommended approach to support multiple SSBLs when they are stored in the SDM QSPI flash:

Changes required in the Programming File Generator, at initial image creation time:

- Use the default naming scheme for the bitstream partitions:
  - FACTORY_IMAGE for factory image
  - P1, P2, and so on for application images
- Make copies of the U-Boot image files (`u-boot-socfpga/u-boot-dtb.img`) to have the .bin extension, as that is what the Programming File Generator requires for binary files. For example name them `u-boot-socfpga/u-boot-dtb.img.bin`
- In the Input Files tab, click Add Raw button, then select the .bin file filter at the bottom and browse for the renamed U-Boot image file. Once added, click the file to select it, then click Properties on the right. Make sure to change the "Bit-swap" option from "off" to "on". This is telling the PFG that it is a regular binary file.
- Create new partitions to contain SSBLs, one for each bitstream:
  - SSBL partitions are large enough for the U-Boot image (512KB can suffice for most applications)
  - Name the partition with a name that is derived from the bitstream partition name, and is less than 15 characters long (limit for SPT partition name excluding null terminator). For example "FACTORY.SSBL", "P1.SSBL", etc.
  - For the initially loaded SSBL partitions, select the corresponding U-Boot binary image files as Input file so that they are loaded with the SSBLs.
- Generate the initial image.

Changes required in FSBL:

- Query SDM and read flash to determine all the partition information, and the currently running bitstream location in flash.
- Look up the currently running bitstream location in the list of the SPT partitions to determine the partition containing the currently running bitstream.
- Add the ".SSBL" to the name of the currently running partition and find the SSBL partition using that name. Treat "FACTORY_IMAGE" differently as adding ".SSBL" to it can make it longer than the maximum allowable of 15 characters.
- Instead of loading the SSBL for a hardcoded address, load it from the partition found at the previous step.

If the SSBL is configured with read-only environment, then no changes are needed. If the SSBL is configured with a modifiable environment, then the following changes are recommended:

- Make the SSBL partitions larger than the maximum anticipated U-Boot image, to accommodate the environment. 512KB can still suffice, as typical U-Boot image is smaller, and typical environment size is 4KB.
- Change the hardcoded value for the SSBL environment address to query SDM for partitioning information and currently running image to determine the name of the current SSBL partition, and use its top portion as environment.

The recommended application image update procedure also changes:

- Use LIBRSU to erase the application image partition. This also disables it, removing it from the CPB.
- Use MTD to erase SSBL partition instead of LIBRSU because it does not support erasing raw partitions yet, just bitstreams.
- Use MTD to write the new contents of the SSBL partition instead of LIBRSU because it does not support writing raw partitions yet, just bitstreams.
- Use LIBRSU to write the new application image. This also enables it, putting it as highest priority in the CPB..

7.4.3. U-Boot Source Code Details

This section presents some details about the U-Boot source code, to aid in the implementation of support for multiple SSBLs.

The U-Boot code which queries SDM for the SPT partitioning information and currently running image and displays them when the `run list` U-Boot command is executed is located in the file `arch/arm/mach-socfpga/rsu_s10.c`. This code can be used as a starting point for implementing the FSBL changes to allow it to load a different SSBL for each bitstream.

File name for the SSBL binary when it is loaded from SD card is defined in `include/configs/socfpga_stratix10_socdk.h`:

```
#define CONFIG_SYS_MMCSD_FS_BOOT_PARTITION    1
#define CONFIG_SPL_FS_LOAD_PAYLOAD_NAME        "u-boot-dtb.img"
```

Location and size of U-Boot environment when it is stored in SD/MMC is defined in `include/configs/socfpga_stratix10_socdk.h`:

```
#define CONFIG_ENV_SIZE            0x1000
#define CONFIG_SYS_MMC_ENV_DEV        0    /* device 0 */
#define CONFIG_ENV_OFFSET         512    /* just after the MBR */
```

Location of SSBL in QSPI flash is defined in `include/configs/socfpga_stratix10_socdk.h`:

```
#define CONFIG_SYS_SPL_MALLOC_START    (CONFIG_SPL_BSS_START_ADDR \
                    - CONFIG_SYS_SPL_MALLOC_SIZE)
#define CONFIG_SPL_SPI_LOAD
#define CONFIG_SYS_SPI_U_BOOT_OFFS      0x3C00000
```

Location and size of U-Boot environment in QSPI flash is defined in `include/configs/socfpga_stratix10_socdk.h`:

```
#ifdef CONFIG_ENV_IS_IN_SPI_FLASH
#undef CONFIG_ENV_OFFSET
#undef CONFIG_ENV_SIZE
#define CONFIG_ENV_OFFSET        0x710000
#define CONFIG_ENV_SIZE            (4 * 1024)
#define CONFIG_ENV_SECT_SIZE        (4 * 1024)
#endif /* CONFIG_ENV_IS_IN_SPI_FLASH */
```

8. Using RSU With HPS First

In the HPS First use case, the initial bitstream does not configure the FPGA fabric, and only the HPS FSBL is loaded and executed. Then at a later time the HPS configures the FPGA fabric – for example from U-Boot or Linux.

The potential advantages of using HPS First are:

- HPS can be booted faster.
- The bitstreams are much smaller, requiring smaller SDM QSPI size.
- The FPGA fabric configuration can reside on larger HPS flash or even be accessed remotely over the network.

The RSU fully supports both FPGA first, and HPS first use cases.

The following changes are required to the example presented in the *Remote System Update Example* section to use HPS first instead of FPGA first.

Related Information

Remote System Update Example on page 37

8.1. Update Hardware Designs to use HPS First

When creating the hardware projects, enable HPS first for each project, either from the Intel Quartus Prime GUI, or as shown in bold below:

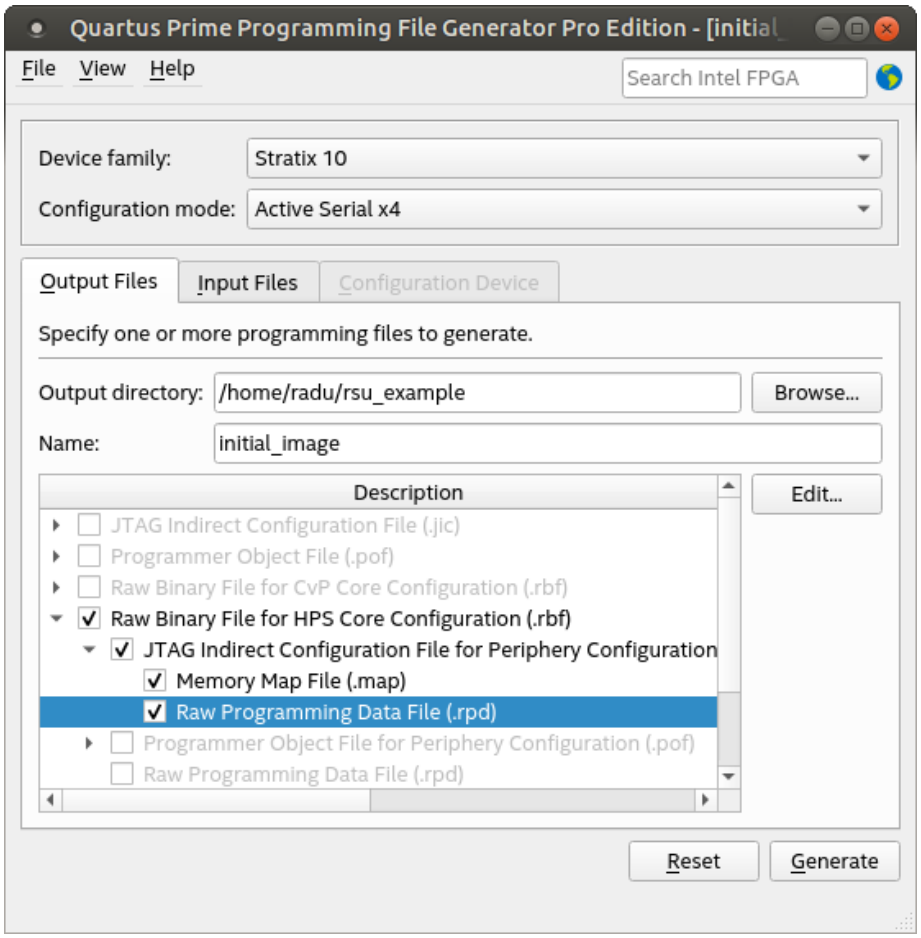
```
cd $TOP_FOLDER
# compile hardware designs: 0-factory, 1,2-applications, 3-factory update
rm -rf hw && mkdir hw && cd hw
for version in {0..3}
do
rm -rf ghrd.$version && mkdir ghrd.$version && cd ghrd.$version
tar xf $SOCEDS_DEST_ROOT/examples/hardware/s10_soc_devkit_ghrd/tgz/*.tar.gz
make clean
make scrub_clean
rm -rf *.qpf *.qsf *.txt *.bin *.qsys ip/qsys_top/ ip/subsys_jtg_mst/ ip\
/subsys_periph/
# enable hps first option
sed -i 's/BOOTS_FIRST := .*/BOOTS_FIRST := hps/g' Makefile
sed -i 's/0xACD5CAFE/0xABAB000'$version'/g' create_ghrd_qsys.tcl
sed -i 's/W_RESET_ACTION ./W_RESET_ACTION 2/g' construct_hps.tcl
make generate_from_tcl
echo "set_global_assignment -name RSU_MAX_RETRY_COUNT 3" \
>> ghrd_1sx280lu2f50e2vg.qsf
make sof
cd ..
done
cd ..
```

8.2. Creating Initial Flash Image for HPS First

The following changes are required:

- Instead of **JTAG Indirect Configuration File (.jic)**, select the **Raw Binary File for HPS Core Configuration (.rbf)** then check the **JTAG Indirect Configuration File for Periphery Configuration (. jic)**, **Memory Map File (.map)**, and the **Raw Programming Data File (.rpd)** options.

Figure 13. Creating JIC File for HPS First



- You can make the factory image and application image partitions smaller, because they do not store the FPGA fabric configuration data.

After clicking the **Generate** button the following additional files are generated in the $TOP_FOLDER folder:

- `initial_image_FACTORY_IMAGE.core.rbf` – containing the FPGA fabric configuration data for the factory image.
- `initial_image_P1.core.rbf` – containing FPGA fabric configuration data for the application image P1.

8.3. Creating Application and Factory Update Images for HPS First

The RSU application image is created with the same command as for FPGA first, with an additional parameter specified, as shown below, in bold:

```
cd $TOP_FOLDER
mkdir -p images
rm -rf images/application2.rpd
quartus_pfg -c hw/ghrd.2/output_files/ghrd_1sx280lu2f50e2vg.sof \
    images/application2.rpd \
    -o hps_path=u-boot-socfpga/spl/u-boot-spl-dtb.hex \
    -o mode=ASX4 -o start_address=0x00000 -o bitswap=ON \
    -o hps=1
```

The factory update image is created with the same command as for FPGA first, with the same additional parameter specified, as shown below, in bold:

```
cd $TOP_FOLDER
mkdir -p images
rm -f images/factory_update.rpd
quartus_pfg -c hw/ghrd.3/output_files/ghrd_1sx280lu2f50e2vg.sof \
    images/factory_update.rpd \
    -o hps_path=u-boot-socfpga/spl/u-boot-spl-dtb.hex \
    -o mode=ASX4 -o start_address=0x00000 -o bitswap=ON \
    -o rsu_upgrade=ON \
    -o hps=1
```

A. RSU Status and Error Codes

The RSU status can be checked from U-Boot and Linux and contains the following 32-bit fields:

Table 11. RSU Status Fields

Field	Description
current_image	Location of currently running image in flash.
failed_image	Address of failed image.
error_details	Opaque error code, with no meaning to users.
error_location	Location of error in the image that failed.
state	State of RSU system.
version	RSU interface version and error source.
retry counter	Current value of the retry counter.

The failed_image, error_details, error_location, state fields and the error_source bit field of the version field have a sticky behavior: they are set when an error occurs, then they are not updated on subsequent errors, and they are cleared when one of the following events occur: POR, nCONFIG, a specific image is loaded, or the error status is specifically cleared from either U-Boot or Linux.

The state field has two bit fields:

Table 12. State fields

Bit Field	Bits	Description
major_error_code	31:16	Major error code, see below for possible values.
minor_error_code	15:0	Minor error code, opaque value

The following major error codes are defined:

Table 13. RSU Major Error Codes

Major Error Code	Description
0xF001	BITSTREAM_ERROR
0xF002	HARDWARE_ACCESS_FAILURE
0xF003	BITSTREAM_CORRUPTION
0xF004	INTERNAL_ERROR

continued...

Major Error Code	Description
0xF005	DEVICE_ERROR
0xF006	HPS_WATCHDOG_TIMEOUT
0xF007	INTERNAL_UNKNOWN_ERROR

The minor error code is typically an opaque value, with no meaning for you. The only exception is for the case where the major error code is 0xF006 (HPS_WATCHDOG_TIMEOUT), in which case the minor error code is the value reported by the HPS to SDM through the RSU Notify command before the watchdog timeout occurred.

The version component has the following bit fields:

Table 14. Version fields

Bit Field	Bits	Description
error_source	31:16	Source of the recorded error: • 0x0000: for no error • 0x0ACF: if the error was produced by an application or factory image firmware • 0x0DCF: if the error was produced by the decision firmware
acmf_version	15:8	Current image firmware RSU interface version
dcmf_version	7:0	Decision firmware RSU interface version

Prior to Intel Quartus Prime version 19.3 the version field was 0x00000000. Intel Quartus Prime version 19.3 sets both the acmf_version and dcmf_version to 1. When both are reported as 1, the following features are available:

- Trying an image multiple times (max retry).
- Querying and resetting the retry counter.
- Clearing the error status.

If only the dcmf_version is set to 1, the images still can try multiple times (if enabled), but the retry counter cannot be queried or reset.

Note: When combining application and factory images, and decision firmware with different RSU firmware interface versions, the acmf_version and dcmf_version fields may be reported inconsistently. Make sure to have all from Intel Quartus Prime version 19.3 or newer for reliable use of the new features.

B. LIBRSU Reference Information

B.1. Configuration File

The LIBRSU library relies on the resource file `/etc/librsu.rc` to set the logging level and the MTD QSPI partition to be used for RSU purposes.

Table 15. LIBRSU Configuration File Elements

Element	Description
Element: `# COMMENT` **Usage:** `// COMMENT` **Options:** None	Single line comments **Required?**: No
Element: `root` **Usage:** `root {type} {path}` **Options:** `type: Storage type = [qspi, datafile]`	Specifies the storage containing the RSU data region that LIBRSU manages. The datafile type is provided for testing purposes and treats an ordinary file as the RSU data region. **Required?**: Yes
Element: `rsu-dev` **Usage:** `rsu-dev {path}` **Options:** • `path : Path to the RSU entries in Linux sysfs`	Specifies the path for the RSU sysfs entries in Linux. **Required?**: No. When not specified, it defaults to `/sys/devices/platform/stratix10-rsu.0`.
Element: `log` **Usage:** `log {level} [stderr|path]` **Options:** • `level : Verbose level = [off, low, medium, high]` • `path : Path to a logfile for debug information` • `stderr (defaults to stderr)`	Instruct LIBRSU to open a logfile and append debug information as commands are performed. Three levels of verbosity are allowed. The log is directed to stderr by default. **Required?**: No
Element: `write-protect` **Usage:** `write-protect {slot}` **Options:** `slot : Slot number`	Instruct LIBRSU to block any attempts to modify the specified slot. The priority of the selected slot might change based on changes to other slots. This option can be used multiple times. **Required?**: No

The default values are:

```
# cat /etc/librsu.rc
log med stderr
root qspi /dev/mtd0
rsu-dev /sys/devices/platform/stratix10-rsu.0
```

Note: Previous versions of LIBRSU did not allow complete configuration of the `rsu-dev` option. When moving from 4.9.78-ltsi kernel to the 4.14.130-ltsi kernel, you need to also move to the new LIBRSU version. However, the new LIBRSU version can be configured to work with the older 4.9.78-ltsi by specifying the following in the configuration file:

```
rsu-dev /sys/devices/platform/soc:firmware:svc/soc:firmware:svc:rsu/
```

The U-Boot SMC handler, Linux SVC driver and Linux RSU driver do not export the SDM API for determining the SPT addresses. Therefore, the MTD QSPI partition to be used by LIBRSU must start at the location of the SPT0, in order for LIBRSU to be able to determine the flash partitioning information. This can either be hardcoded in the device tree, or U-Boot can edit the device tree with the appropriate information before passing it to Linux using the `rsu dtb` command.

B.2. Error Codes

In case of success, the LIBRSU APIs return the value 0; otherwise, the LIBRSU APIs return the values shown below, as negative values:

```
#define ELIB          1
#define ECFG          2
#define ESLOTNUM      3
#define EFORMAT       4
#define EERASE        5
#define EPROGRAM      6
#define ECMP          7
#define ESIZE         8
#define ENAME         9
#define EFILEIO      10
#define ECALLBACK    11
#define ELOWLEVEL    12
#define EWRPROT      13
#define EARGS        14
```

B.3. Macros

The following macros are available to extract the fields from the version field of the `rsu_status_info` structure:

```
#define RSU_VERSION_ERROR_SOURCE(v) (((v) & 0xFFFF0000) > 16)
#define RSU_VERSION_ACMF_VERSION(v) (((v) & 0xFF00) > 8)
#define RSU_VERSION_DCMF_VERSION(v) ((v) & 0xFF)
```

B.4. Data Types

B.4.1. `rsu_slot_info`

This structure contains slot (SPT entry) information.

```
struct rsu_slot_info {
    char name[16];
    __u64 offset;
    int size;
    int priority;
};
```

B.4.2. rsu_status_info

This structure contains the RSU status information.

```
struct rsu_status_info {
    __u64 version;
    __u64 state;
    __u64 current_image;
    __u64 fail_image;
    __u64 error_location;
    __u64 error_details;
    __u64 retry_counter;
};
```

B.4.3. rsu_data_callback

This is a callback used in a scheme to provide data in blocks as opposed to all at once in a large buffer, which may minimize memory requirements.

```
/*
 * rsu_data_callback - function pointer type for data source callback
 */
typedef int (*rsu_data_callback)(void *buf, int size);
```

B.5. Functions

B.5.1. librsu_init

Prototype	`int librsu_init(char *filename);`
Description	Load the configuration file and initialize internal data by reading SPT and CPB data from flash. • If SPT0 is corrupted, SPT1 is loaded instead. If one SPT is corrupted (no magic number) and one is good, the corrupted SPT is recovered with information from the good SPT. • If CPB0 is corrupted, CPB1 is loaded instead. If one CPB is corrupted (no magic number) and one is good, the corrupted CPB is recovered with information from the good CPB.
Parameters	`filename`: configuration file to load. If Null or empty string, the default is `/etc/librsu.rc`
Return Value	0 on success, or error code

B.5.2. librsu_exit

Prototype	`void librsu_exit(void);`
Description	Cleanup internal data and release librsu.
Parameters	None
Return Value	None

B.5.3. rsu_slot_count

Prototype	`int rsu_slot_count(void);`
Description	Get the number of slots defined.
Parameters	None
Return Value	The number of defined slots

B.5.4. rsu_slot_by_name

Prototype	`int rsu_slot_by_name(char *name);`
Description	Retrieve slot number based on name.
Parameters	`name`: name of slot
Return Value	Slot number on success, or error code

B.5.5. rsu_slot_get_info

Prototype	`int rsu_slot_get_info(int slot, struct rsu_slot_info *info);`
Description	Retrieve the attributes of a slot.
Parameters	`slot`: slot number `info`: pointer to info structure to be filled in
Return Value	0 on success, or error code

B.5.6. rsu_slot_size

Prototype	`int rsu_slot_size(int slot);`
Description	Get the size of a slot.
Parameters	`slot`: slot number
Return Value	The size of the slot in bytes, or error code

B.5.7. rsu_slot_priority

Prototype	`int rsu_slot_priority(int slot);`
Description	Get the load priority of a slot. Priority of zero means the slot has no priority and is disabled. The slot with priority of one has the highest priority.
Parameters	`slot`: slot number
Return Value	The priority of the slot, or error code

B.5.8. rsu_slot_erase

Prototype	`int rsu_slot_erase(int slot);`
Description	Erase all data in a slot to prepare for programming. Remove the slot if it is in the CPB.
Parameters	`slot`: slot number
Return Value	0 on success, or error code

B.5.9. rsu_slot_program_buf

Prototype	`int rsu_slot_program_buf(int slot, void *buf, int size);`
Description	Program a slot using FPGA config data from a buffer and enter slot into CPB as highest priority.
Parameters	`slot`: slot number `buf`: pointer to data buffer `size`: bytes to read from buffer
Return Value	0 on success, or error code

B.5.10. rsu_slot_program_factory_update_buf

Prototype	`int rsu_slot_program_factory_update_buf(int slot, void *buf, int size);`
Description	Program a slot using a factory update image data from a buffer and enter slot into CPB as highest priority.
Parameters	`slot`: slot number `buf`: pointer to data buffer `size`: bytes to read from buffer
Return Value	0 on success, or error code

B.5.11. rsu_slot_program_file

Prototype	`int rsu_slot_program_file(int slot, char *filename);`
Description	Program a slot using FPGA config data from a file and enter slot into CPB as highest priority.
Parameters	`slot`: slot number `filename`: input data file
Return Value	0 on success, or error code

B.5.12. rsu_slot_program_factory_update_file

Prototype	`int rsu_slot_program_factory_update_file(int slot, char *filename);`
Description	Program a slot using a factory update image data from a file and enter slot into CPB as highest priority.
Parameters	`slot`: slot number `filename`: input data file
Return Value	0 on success, or error code

B.5.13. rsu_slot_program_buf_raw

Prototype	`int rsu_slot_program_buf_raw(int slot, void *buf, int size);`
Description	Program a slot using raw data from a buffer. The slot is not entered into the CPB.
Parameters	`slot`: slot number `buf`: pointer to data buffer `size`: bytes to read from buffer
Return Value	0 on success, or error code

B.5.14. rsu_slot_program_file_raw

Prototype	`int rsu_slot_program_file_raw(int slot, char *filename);`
Description	Program a slot using raw data from a file. The slot is not entered into the CPB.
Parameters	`slot`: slot number `filename`: input data file
Return Value	0 on success, or error code

B.5.15. rsu_slot_verify_buf

Prototype	`int rsu_slot_verify_buf(int slot, void *buf, int size);`
Description	Verify FPGA config data in a slot against a buffer.
Parameters	`slot`: slot number `buf`: pointer to data buffer `size`: bytes to read from buffer
Return Value	0 on success, or error code

B.5.16. rsu_slot_verify_file

Prototype	`int rsu_slot_verify_file(int slot, char *filename);`
Description	Verify FPGA config data in a slot against a file.
Parameters	`slot`: slot number `filename`: input data file
Return Value	0 on success, or error code

B.5.17. rsu_slot_verify_buf_raw

Prototype	`int rsu_slot_verify_buf_raw(int slot, void *buf, int size);`
Description	Verify raw data in a slot against a buffer.
Parameters	`slot`: slot number `buf`: pointer to data buffer `size`: bytes to read from buffer
Return Value	0 on success, or error code

B.5.18. rsu_slot_verify_file_raw

Prototype	`int rsu_slot_verify_file_raw(int slot, char *filename);`
Description	Verify raw data in a slot against a file.
Parameters	`slot`: slot number `filename`: input data file
Return Value	0 on success, or error code

B.5.19. rsu_slot_program_callback

Prototype	`int rsu_slot_program_callback(int slot, rsu_data_callback callback);`
Description	Program and verify a slot using FPGA config data provided by a callback function. Enter the slot into the CPB as highest priority.
Parameters	`slot`: slot number `callback`: callback function to provide input data
Return Value	0 on success, or error code

B.5.20. rsu_slot_program_callback_raw

Prototype	`int rsu_slot_program_callback_raw(int slot, rsu_data_callback callback);`
Description	Program and verify a slot using raw data provided by a callback function. The slot is not entered into the CPB.
Parameters	`slot`: slot number `callback`: callback function to provide input data
Return Value	0 on success, or error code

B.5.21. rsu_slot_verify_callback

Prototype	`int rsu_slot_verify_callback(int slot, rsu_data_callback callback);`
Description	Verify a slot using FPGA configuration data provided by a callback function.
Parameters	`slot`: slot number `callback`: callback function to provide input data
Return Value	0 on success, or error code

B.5.22. rsu_slot_verify_callback_raw

Prototype	`int rsu_slot_verify_callback_raw(int slot, rsu_data_callback callback);`
Description	Verify a slot using raw data provided by a callback function.
Parameters	`slot`: slot number `callback`: callback function to provide input data
Return Value	0 on success, or error code

B.5.23. rsu_slot_copy_to_file

Prototype	`int rsu_slot_copy_to_file(int slot, char *filename);`
Description	Read the data in a slot and write to a file.
Parameters	`slot`: slot number `filename`: input data file
Return Value	0 on success, or error code

B.5.24. rsu_slot_enable

Prototype	`int rsu_slot_enable(int slot);`
Description	Set the selected slot as the highest priority. This is the first slot attempted after a power-on reset.
Parameters	`slot`: slot number
Return Value	0 on success, or error code

B.5.25. rsu_slot_disable

Prototype	`int rsu_slot_disable(int slot);`
Description	Remove the selected slot from the priority scheme, but do not erase the slot data so that it can be re-enabled.
Parameters	`slot`: slot number
Return Value	0 on success, or error code

B.5.26. rsu_slot_load_after_reboot

Prototype	`int rsu_slot_load_after_reboot(int slot);`
Description	Request that the selected slot be loaded after the next restart, no matter the priority. A power-on reset ignores this request and uses slot priority to select the first slot.
Parameters	`slot`: slot number
Return Value	0 on success, or error code

B.5.27. rsu_slot_load_factory_after_reboot

Prototype	`int rsu_slot_load_factory_after_reboot(void);`
Description	Request that the factory image be loaded after the next restart. A power-on reset ignores this request and uses slot priority to select the first slot.
Parameters	None
Return Value	0 on success, or error code

B.5.28. rsu_slot_rename

Prototype	`int rsu_slot_rename(int slot, char *name);`
Description	Rename the selected slot.
Parameters	`slot`: slot number `name`: new name for slot
Return Value	0 on success, or error code

B.5.29. rsu_slot_status_log

Prototype	`int rsu_status_log(struct rsu_status_info *info);`
Description	Copy the SDM status log to info struct.
Parameters	`info`: pointer to info struct to fill in
Return Value	0 on success, or error code

B.5.30. rsu_notify

Prototype	`int rsu_notify(int value);`
Description	Report HPS software execution stage as a 16-bit number.
Parameters	`value`: HPS software execution stage - only 16-bit lower bits are used
Return Value	0 on success, or error code

B.5.31. rsu_clear_error_status

Prototype	`int rsu_clear_error_status(void);`
Description	Clear the sticky error fields from the current status log.
Parameters	None
Return Value	0 on success, or error code

B.5.32. rsu_reset_retry_counter

Prototype	`int rsu_reset_retry_counter(void);`
Description	Reset the `retry` counter, so that the currently running image can try again after a watchdog timeout, if the value of `max retry` is greater than one.
Parameters	None
Return Value	0 on success, or error code

11. Document Revision History for the Intel Stratix 10 SoC Remote System Update User Guide

Table 16. Document Revision History for the Intel Stratix 10 SoC Remote System Update User Guide

Document Version	Changes
2020.01.09	Updated the `git checkout` command in section *Building U-Boot*.
2019.12.20	Updated for Intel Quartus Prime software version 19.3 release: • Added factory image update procedure, examples and reference information. • Added new description for error handling related to loading a specific image • Full U-Boot support for RSU • LIBRSU support for RSU notify Updated for 19.3 release: • Added the max retry option, including examples and reference information. • Added clearing the error status Updated all examples to latest tools and software.
2019.02.27	Initial release