Booting and Configuration Introduction

A

2014.02.28

av_5400a

This topic describes the booting of the hard processor system (HPS) and the configuration of the FPGA portion of the Altera system-on-a-chip (SoC) device.

The HPS boot starts when a processor is released from reset (for example, on power up) and executes code in the internal boot ROM at the reset exception address. The boot process ends when the code in the boot ROM jumps to the next stage of the boot software. This next stage of boot software is referred to as the preloader. The preloader can be customized and is typically stored external to the HPS in a nonvolatile flash-based memory.

The processor can boot from the following sources:

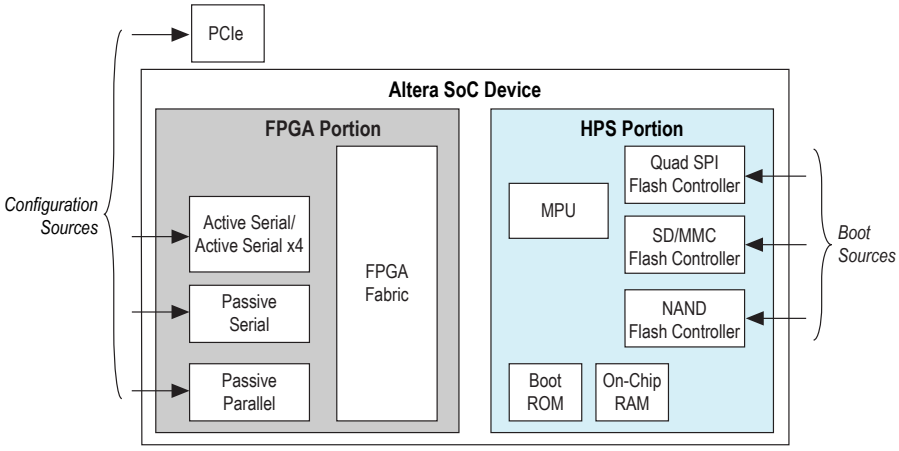
- NAND flash memory through the NAND flash controller
- Secure Digital/MultiMediaCard (SD/MMC) flash memory through the SD/MMC flash controller
- Serial peripheral interface (SPI) and quad SPI flash memory through the quad SPI flash controller using Slave Select 0
- FPGA fabric

The HPS boot supports indirect or direct execution of the preloader depending on the boot device. With indirect execution, the boot ROM code copies the preloader from the boot device into the on-chip RAM and jumps to it. Indirect execution is used for flash memory boot sources. With direct execution, the boot ROM code jumps to the preloader located in the FPGA fabric.

Configuration of the FPGA portion of the device starts when the FPGA portion is released from the reset state (for example, on power-on). The control block (CB) in the FPGA portion of the device is responsible for obtaining an FPGA configuration image and configuring the FPGA. The FPGA configuration ends when the configuration image has been fully loaded and the FPGA enters user mode. The FPGA configuration image is provided by users and is typically stored in non-volatile flash-based memory. The FPGA CB can obtain a configuration image from the HPS through the FPGA manager or from any of the sources supported by the Arria V FPGAs family.

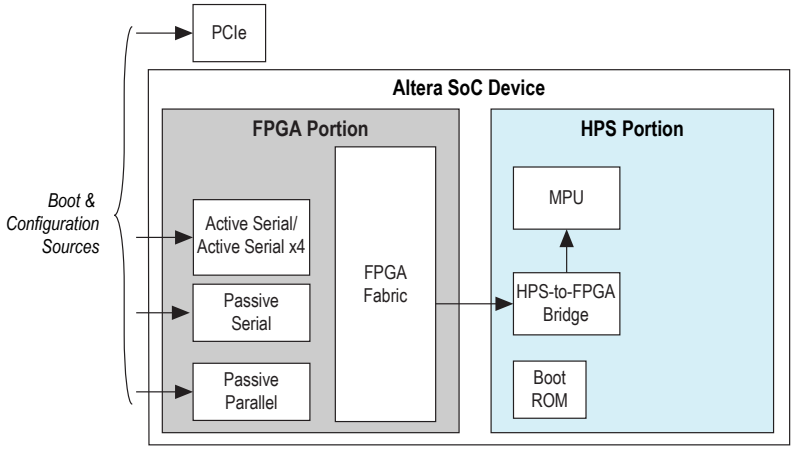
The following three figures illustrate the possible HPS boot and FPGA configuration schemes. The arrows in the figures denote the data flow direction. The following figure shows that the FPGA configuration and HPS boot occur independently. The FPGA configuration obtains its configuration image from a non-HPS source, while the HPS boot obtains its preloader from a non-FPGA fabric source.

Figure A-1: Independent FPGA Configuration and HPS Booting



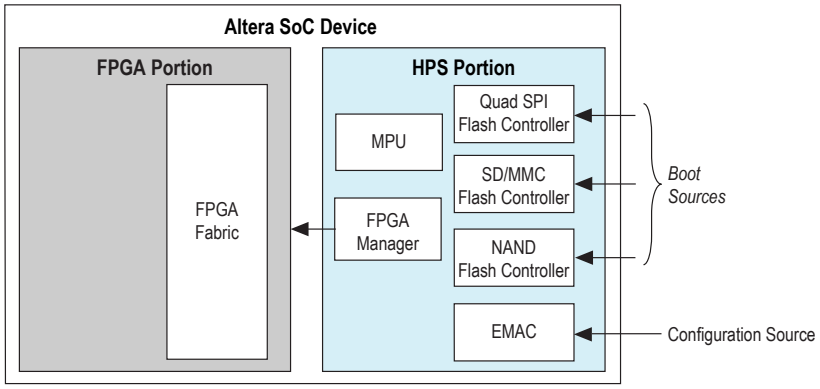
The following figure shows that the FPGA is first configured through one of its non-HPS configuration sources and then the HPS boots from the FPGA fabric. The HPS boot waits for the FPGA fabric to be powered on and in user mode before executing. The HPS boot ROM code executes the preloader from the FPGA fabric over the HPS-to-FPGA bridge. The preloader can be obtained from the FPGA RAM or by accessing an external interface, depending on your design and implementation.

Figure A-2: FPGA Configures First



The following figure shows that the HPS boots first through one of its non-FPGA fabric boot sources and then software running on the HPS configures the FPGA fabric through the HPS FPGA manager. The software on the HPS obtains the FPGA configuration image from any of its flash memory devices or communication interfaces, for example, the Ethernet media access controller (EMAC). The software is provided by users and the boot ROM is not involved in configuring the FPGA fabric.

Figure A-3: HPS Boots First



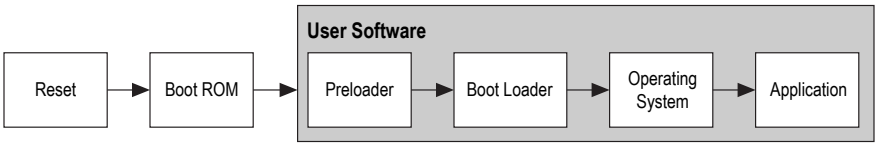
HPS Boot

Booting software on the HPS is a multi-stage process. Each stage is responsible for loading the next stage. The first software stage is the boot ROM. The boot ROM code locates and executes the second software stage, called the preloader. The preloader locates, and if present, executes the next software stage. The preloader and subsequent software stages (if present) are collectively referred to as user software.

Only the boot ROM code is located in the HPS. Subsequent software is located external to the HPS and is provided by users. The boot ROM code is only aware of the preloader and not aware of any potential subsequent software stages.

The figure below illustrates the typical boot flow. However, there may be more or less software stages in the user software than shown and the roles of the software stages may vary.

Figure A-4: Typical Boot Flow



Boot Process Overview

Reset

The boot process begins when a CPU in the MPU exits from the reset state. When a CPU exits from reset, it starts running code at the reset exception address. In normal operation, the boot ROM is mapped at the reset exception address so code starts running in the boot ROM.

It is possible to map the on-chip RAM or SDRAM at the reset exception address and run code other than the boot ROM code. However, this chapter assumes that the boot ROM maps to the reset exception address.

Boot ROM

The boot ROM contains software code that executes after a reset exits. If running on CPU0, the boot ROM code reads the boot select (BOOTSEL) and clock select (CLKSEL) values from the bsel and csel fields of the boot information register (boot info) in the system manager. This is done to determine the boot source and to set up the clock manager. The bsel and csel field values come from the BOOTSEL and CLKSEL pins which are sampled by the system manager coming out of reset.

The user software in CPU0 is responsible to release CPU1 from reset. If CPU1 is released from reset with the boot ROM mapped to the reset exception address, the boot ROM code jumps to the value in the CPU1 start address (`cpu1startaddr`) in the boot ROM code register group (`romcodegrp`) in the system manager.

BOOTSEL Field Values and Flash Device Selection

Table A-1: BOOTSEL Field Values and Flash Device Selection

BOOTSEL Field Value	Flash Device
0x0	Reserved
0x1	FPGA (HPS-to-FPGA bridge)
0x2	1.8 V NAND flash memory
0x3	3.3 V NAND flash memory
0x4	1.8 V SD/MMC flash memory with external transceiver
0x5	3.3 V SD/MMC flash memory with internal transceiver
0x6	1.8 V SPI or quad SPI flash memory
0x7	3.3 V SPI or quad SPI flash memory

Note:

If the BOOTSEL value is set 0x1 to Boot from FPGA, then the CLKSEL values are ignored. PLLs are bypassed so that OSC1 drives all the clocks.

For indirect execution from flash memory boot sources, the boot ROM code loads the preloader image from the flash device to the on-chip RAM and passes software control to the preloader in the on-chip RAM. For direct execution from the FPGA fabric boot source, the boot ROM code waits until the FPGA portion of the device is in user mode, and is ready to execute code and then passes software control to the preloader in the FPGA RAM.

Preloader

The function of the preloader is user-defined. However, typical functions include initializing the SDRAM interface and configuring the HPS I/O pins. Initializing the SDRAM allows the preloader to load the next stage of the boot software (that might not fit in the 60 kilobytes (KB) available in the on-chip RAM). A typical next software stage is the open source boot loader, U-boot.

The preloader is allowed to load the next boot software stage from any device available to the HPS. Typical sources include the same flash device that contains the preloader, a different flash device, or a communication interface such as an EMAC.

Boot Loader

The boot loader loads the operating system and passes software control to the operating system.

Boot ROM

The function of the boot ROM code is to determine the boot source, initialize the HPS after a reset, and jump to the preloader. In the case of indirect execution, the boot ROM code loads the preloader image from the flash memory to on-chip RAM. The boot ROM performs the following actions to initialize the HPS:

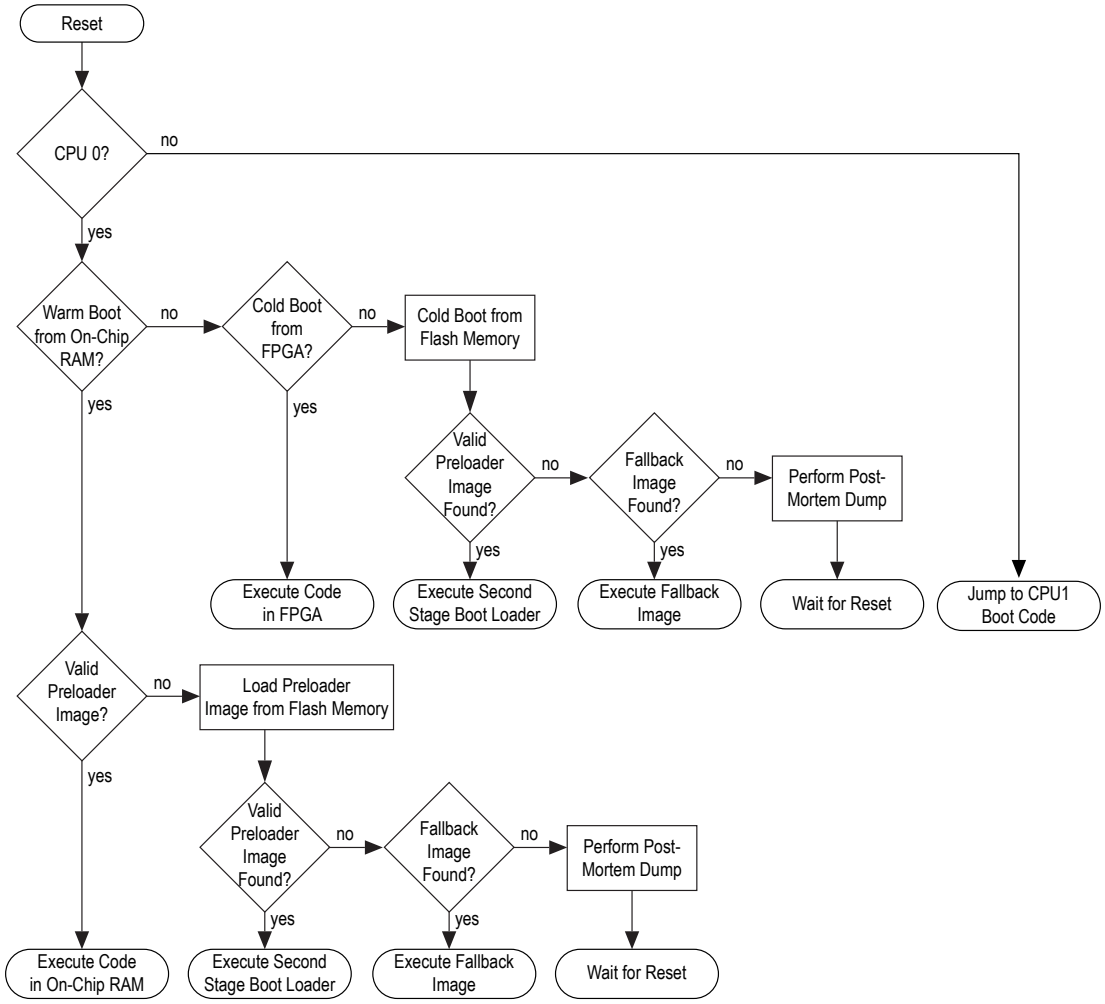
- Enable instruction cache, branch predictor, floating point unit, NEON vector unit
- Sets up the level 4 (l4) watchdog 0 timer
- Configures the main PLL and peripheral PLL based on the CLKSEL value
- Configures I/O elements and pin multiplexing based on the BOOTSEL value
- Initializes the flash controller to default settings

When booting from flash memory, the boot ROM code uses the top 4 KB of the on-chip RAM as data workspace. This area is reserved for the boot ROM code after a reset until the boot ROM code passes software control to preloader. This limits the maximum size of the preloader for indirect execution to 60 KB. For a warm boot or cold boot from FPGA, the boot ROM code does not reserve the top 4KB of the on-chip RAM, and the user may place user data in this area without being overwritten by boot ROM.

Boot ROM Flow

This section describes the software flow from reset until the boot ROM code passes software control to the preloader. The following figure illustrates that the boot ROM code can perform a warm boot from on-chip RAM, a cold boot from the FPGA portion of the device, or a cold boot from flash memory.

Figure A-5: BOOT ROM FLOW



During a cold boot from the FPGA portion of the device, the boot ROM code waits until the FPGA is ready and then attempts direct execution at address 0xC0000000 across the HPS-to-FPGA bridge. For example, the boot software could be provided by initialized on-chip RAM in the FPGA portion of the device at address 0xC0000000 (offset 0x0 from HPS-to-FPGA bridge). During a cold boot from flash memory, the boot ROM code attempts to load the first preloader image from flash memory to on-chip RAM and pass control to the preloader. If the image is invalid, the boot ROM code attempts to load up to three subsequent images from flash memory. If there is still no valid image found after the subsequent loads, the boot ROM code checks the FPGA portion of the device for a fallback image.

Warm boot from on-chip RAM has the highest priority to execute if the `warmramgrp` registers in the `romcodegrp` group in the system manager has been configured to support booting from on-chip RAM on a warm reset. When the enable in `warmramgrp` is enabled and length is set to 0x0, boot ROM will not perform CRC on the preloader image and immediately jump to the address in execution. If the length is not 0x0, boot ROM will ensure the preloader image passes the CRC check before the preloader image is executed.

If a valid preloader image cannot be found in the on-chip RAM, or the preloader in the on-chip RAM fails the CRC check, boot ROM code attempts to load the last valid preloader image loaded from the flash memory,

identified by the `index` field of the initial software last image loaded register (`initswlastld`) in the `romcodegrp` group in the system manager. If the image is invalid, boot ROM code attempts to load up to three subsequent images from flash memory. If a valid preloader image cannot be found in the on-chip RAM or flash memory, the boot ROM code checks the FPGA portion of the device for a fallback image.

Loading the Preloader

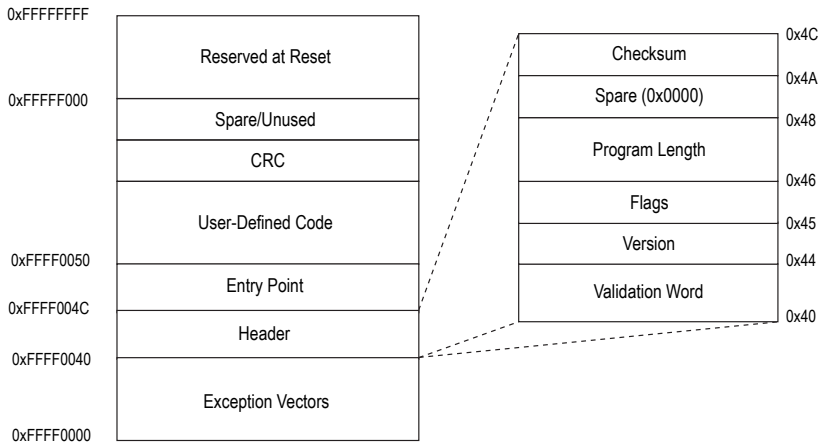
The boot ROM code loads the preloader image from flash memory into the on-chip RAM and passes control to the preloader. The boot ROM code checks for a valid image by verifying the header and cyclic redundancy check (CRC) in the preloader image.

The boot ROM code checks the header for the following information:

- Validation word—validates the preloader image. The validation word has a fixed value of 0x31305341.
- Version—indicates the header version.
- Program length—the total length of the image (in 32-bit words) from offset 0x0 to the end of code area, including exception vectors and CRC.
- Checksum—a checksum of all the bytes in the header, from offset 0x40 to 0x49.

The preloader image has a maximum size of 60 KB. This size is limited by the on-chip RAM size of 64 KB, where 4 KB is reserved as a workspace for the boot ROM data and stack. The preloader can use this 4 KB region (for its stack and data, for example) after the boot ROM code passes control to the preloader. This 4 KB region is overwritten by the boot ROM code on a subsequent reset. The following figure shows the preloader image layout in the on-chip RAM after being loaded from the boot ROM.

Figure A-6: Preloader Image Layout



Exception vectors—Exception vectors are located at the start of the on-chip RAM. Typically, the preloader remaps the lowest region of the memory map to the on-chip RAM (from the boot ROM) to create easier access to the exception vectors.

Header—contains information such as validation word, version, flags, program length, and checksum for the boot ROM code to validate the preloader image before passing control to the preloader.

Entry point—contains the preloader image address. After the boot ROM code validates the header, the boot ROM code jumps to this address.

User-defined code—typically contains the program code of the preloader.

CRC—contains a CRC of data from address 0xFFFF0000 to 0xFFFF0000+(Program Length*4)–0x0004. The polynomial used to validate the preloader image is *x 32 + x 26 + x 23 + x 22 + x 16 + x 12 + x 11 + x 10 + x 8 + x 7 + x 5 + x 4 + x 2 + x + 1*. There is no reflection of the bits. The initial value of the remainder is 0xFFFFFFFF and the final value is XORed with 0xFFFFFFFF.

Reserved at reset—the top 4 KB is reserved for the boot ROM code after a reset. The boot ROM code uses this area for internal structures, workspace, and post-mortem dump. This area includes the shared memory where the boot ROM code passes information to the preloader.

Shared Memory

The shared memory contains information that the boot ROM code passes to the preloader. The boot ROM code passes the location of shared memory to the preloader in register `r0`, as described in **HPS State on Entry to the Preloader** on page A-11.

The shared memory contains the following information:

- Common—contains non-flash-specific settings used by the boot ROM code.
- Saved hardware register—contains hardware register values that the boot ROM code saves before the registers are modified by the boot ROM code.
- Flash device-specific—contains flash device-specific settings used by the boot ROM code that the preloader may use to continue using the flash device without reinitialization.

Table A-2: Shared Memory Block

Information Type	Content	Description
Common	Flash image	Indicates which preloader image (0-3) the boot ROM code loaded from the flash device. A nonzero value indicates that there was an error loading image 0 and the boot ROM code loaded another image.
	CLKSEL value used	Indicates the CLKSEL value used by the boot ROM code. Typically, this value is the value read from the `clksel` field of the `bootinfo` register in the `romcodegrp` group in the system manager. However, if the PLL fails to lock, the boot ROM code ignores the `clksel` field and uses zero to indicate PLL bypass mode.
	BOOTSEL value used	Indicates the BOOTSEL value used by the boot ROM code. Typically, this value is the value read from the `bootsel` field of the `bootinfo` register in the `romcodegrp` group in the system manager.
	Last page	Indicates the number of the last page read from the flash device.
	Page size	Indicates the page size (in bytes) used by the flash device. • For NAND flash memory, the boot ROM code reads this value from the NAND flash controller. • For SPI and quad SPI flash memory, the boot ROM code configures the page size through the quad SPI flash controller. • For SD/MMC flash memory, the boot ROM code configures the page size through the SD/MMC flash controller.

Information Type	Content	Description
	Flash device type	Indicates the flash device used by the boot ROM code.
	Step complete	Track the completion state of up to 64 individual major steps during the boot process. The 64 bit value has one bit set for each major step completed in the boot process.
	CPU0 and CPU1 crash data	Indicates the CPU0 and CPU1 crash data. These values are pointers to where the boot ROM code saves the crash dump in an event of a crash.
	Flags	The last bit indicates if warm boot with CRC check is used.
Saved hardware registers	Status register (`stat`) of the reset manager	Contains reset source and event timeout information.
	Control (`ctrl`) register in the `romcodegrp` group in the system manager	Contains information used to control the boot ROM code.
	`initswstate` register in the `romcodegrp` group in the system manager	Contains the magic value 0x49535756 when the preloader has reached a valid state.
Flash device- specific (SD/MMC)	is_sd_card	Indicates the card type. 1 indicates SD card; 0 indicates MMC.
	is_sector_mode	Indicates the addressing mode of card. 1 indicates sector addressing mode; 0 indicates byte addressing mode.
	rca	Contains the relative card address, which is used for host-card communication during card identification
	partition_start_sector	Indicates the partition start offset in unit sectors. 0 indicates raw mode.
	partition_size	Size of the partition in unit sectors. 0 indicates raw mode.

Related Information

HPS State on Entry to the Preloader on page A-11

L4 Watchdog 0 Timer

The L4 watchdog 0 timer is reserved for boot ROM use. While booting, if a watchdog reset happens before software control passes to the preloader, boot ROM code attempts to load the last valid preloader image, identified by the `initswlastld` register in the `romcodegrp` group in the system manager.

If the watchdog reset happens after the preloader has started executing but before the preloader writes a valid value to `initswstate` register, the boot ROM increments `initswlastld` and attempts to load that image. If the watchdog reset happens after the preloader writes a valid value to `initswstate` register, boot ROM code attempts to load the image indicated by `initswlastld` register.

HPS State on Entry to the Preloader

When the boot ROM code is ready to pass control to the preloader, the processor (CPU0) is in the following state:

- Instruction cache is enabled
- Branch predictor is enabled
- Data cache is disabled
- MMU is disabled
- Floating point unit is enabled
- NEON vector unit is enabled
- Processor is in ARM secure supervisor mode

The boot ROM code sets the ARM® Cortex™-A9 MPCore™ registers to the following values:

- r0—contains the pointer to the shared memory block, which is used to pass information from the boot ROM code to the preloader. The shared memory block is located in the top 4 KB of on-chip RAM.
- r1—contains the length of the shared memory.
- r2—unused and set to 0x0.
- r3—reserved.

All other MPCore registers are undefined.

Note: When booting CPU0 using the FPGA boot, or when booting CPU1 using any boot source, all MPCore registers, caches, the MMU, the floating point unit, and the NEON vector unit are undefined. HPS subsystems and the PLLs are undefined.

When the boot ROM code passes control to the preloader, the following conditions also exist:

- The boot ROM is still mapped to address 0x0.
- The L4 watchdog 0 timer is active and has been toggled.

Preloader

The preloader typically performs the following actions:

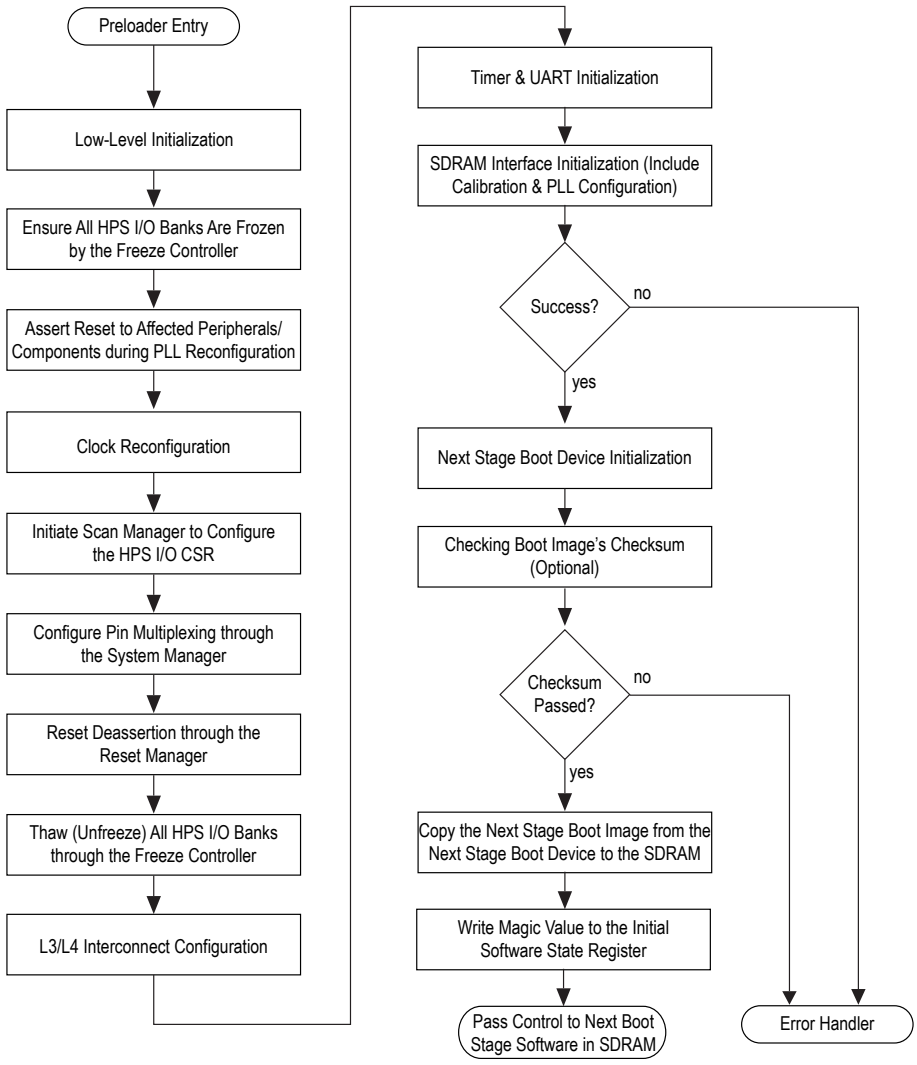
- Initialize the SDRAM interface.

- Configure the `remap` register to map the on-chip RAM to address 0x0 so that exceptions are handled by the preloader.
- The on-chip RAM is also accessible with the alias 0x0.
- Configure the HPS I/O through the scan manager.
- Configure pin multiplexing through the system manager.
- Configure HPS clocks through the clock manager.
- Initialize the flash controller (NAND, SD/MMC, or quad SPI) that contains the next stage boot software.
- Load the next stage boot software into the SDRAM and pass control to it.

Typical Preloader Boot Flow

This section describes a typical software flow from the preloader entry point until the software passes control to the next stage boot software.

Figure A-7: Typical Preloader Bootflow



Low-level initialization steps include reconfiguring or disabling the L4 watchdog 0 timer, invalidating the instruction cache and branch predictor, remapping the on-chip RAM to the lowest memory region, and setting up the data area.

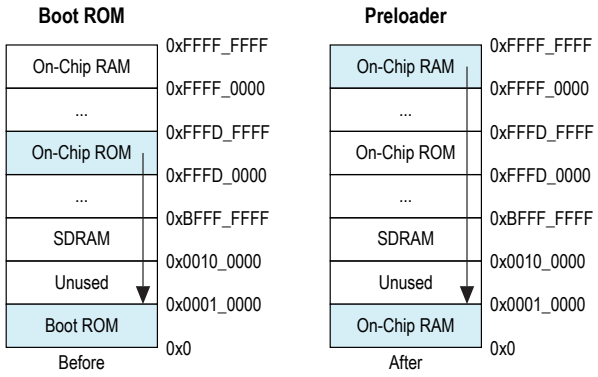
Upon entering the preloader, the L4 watchdog 0 timer is active. The preloader can either disable, reconfigure, or leave the watchdog timer unchanged. Once enabled after reset, the watchdog timer cannot be disabled, only paused.

The instruction cache and branch predictor, which were previously enabled by the boot ROM code, need to be invalidated.

The preloader needs to remap the exception vector table because the exception vectors are still pointing to the exception handler in the boot ROM when the preloader starts executing. By setting the L3 interconnect remap bit 0 to high, the on-chip RAM mirrors to the lowest region of the memory map. After this remap, the exception vectors will use the exception handlers in the preloader image.

The figure below shows the memory map before and after remap.

Figure A-8: Remapping the On-Chip RAM



The preloader can reconfigure all HPS clocks. During clock reconfiguration, the preloader asserts reset to the peripherals in the HPS affected by the clock changes.

The preloader configures HPS I/O pins through the scan manager and pin multiplexing through the system manager. The preloader initiates the freeze controller in the scan manager to freeze all the I/O pins and put them in a safe state during I/O configuration and pin multiplexing.

The SDRAM goes through full initialization for cold boot or a partial initialization for warm boot. For full initialization, the preloader configures the SDRAM PLL before releasing the SDRAM interface from reset. SDRAM calibration adjusts I/O delays and FIFO settings to compensate for any board skew or impairment in the board, FPGA portion of the device, or memory device. For partial initialization, SDRAM PLL configuration and SDRAM calibration is not necessary.

The preloader looks for a valid next stage boot image in the next stage boot device by checking the boot image validation data and checksum in the mirror image. Once validated, the preloader copies the next stage boot image from the next stage boot device to the SDRAM.

Before software passes control to the next stage boot software, the preloader can write a valid value (0x49535756) to the preloader `initswstate` register under the `romcodegrp` group in the system manager. This value indicates that there is a valid boot image in the on-chip RAM. When a warm reset occurs, the

boot ROM code can check the `initswstate` register for the magic value to determine if it needs to reload the preloader image into the on-chip RAM.

Flash Memory Devices

The flash memory devices available for HPS boot are the NAND, SD/MMC, SPI, and quad SPI. The flash device can store the following kinds of files for booting purposes:

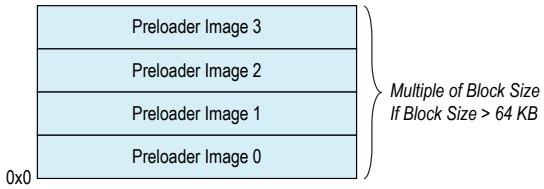
- Preloader binary file (up to four copies)
- Boot loader binary file
- Operating system binary file
- Application file
- FPGA programming files

Note: The preloader file must be stored in a partition with no file system.

NAND Flash Devices

The following figure shows that the preloader image is located at offsets which are multiples of the block size. For NAND device with block size equal or greater than 64kB, the preloader images are located in the first 4 blocks of the device. For NAND device with less than 64kB block size, preloader image needs to be placed in multiple of blocks. Since a block is the smallest area used for erase operation, any update to a particular image does not affect other images.

Figure A-9: NAND Flash Image Layout



Related Information

http://www.altera.com/literature/hb/arria-v/av_54010.pdf

For more information about the NAND flash memory, refer to the *NAND Flash Controller* chapter in the Arria V Device Handbook, Volume 3.

NAND Flash Driver Features Supported in the Boot ROM Code

Table A-3: NAND Flash Support Features

Feature	Driver Support
Device	Open NAND Flash Interface (ONFI) 1.0 raw NAND or electronic signature devices, single layer cell (SLC)
Chip select	CS0 only. Only CS0 is available to the HPS, the other three chip selects are routed out to the FPGA portion of the device.
Bus width	x8 only

Feature	Driver Support
Page size	512 bytes, 2 KB, 4 KB, or 8 KB.
Page per block	32, 64, 128, 384 or 512
ECC	512-bytes with 8-bit correction

CLKSEL Pin Settings for the Quad SPI Controller

Table A-4: NAND Controller CLKSEL Pin Settings

Setting	CSEL Pin			
	0	1	2	3
osc1_clk (EOSC1 pin) range	10–50 MHz	10–12.5 MHz	12.5–25 MHz	25–50 MHz
Device frequency (nand_x_clk/25)	osc1_clk/25, 2 MHz max	osc1_clk*20/25, 9.6 MHz max	osc1_clk*10/25, 9.6 MHz max	osc1_clk*5/25, 9.6 MHz max
controller clock (nand_x_clk)	osc1_clk, 50 MHz max	osc1_clk*20, 240 MHz max	osc1_clk*10, 240 MHz max	osc1_clk*5, 240 MHz max
mpu_clk	osc1_clk, 50 MHz max	osc1_clk*32, 400 MHz max	osc1_clk*16, 400 MHz max	osc1_clk*8, 400 MHz max
PLL modes	Bypassed	Locked	Locked	Locked

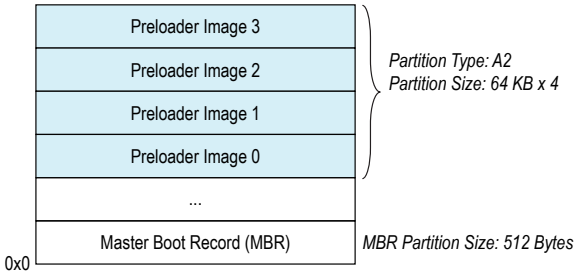
SD/MMC Flash Devices

The following figure shows the SD/MMC flash image layout. The master boot record (MBR) is located at the first 512 bytes of the memory. The MBR contains information of partitions (address and size of partition). The preloader image is always stored in partition A2. Partition A2 is a custom raw partition with no file system.

The start address of each image is based on the following formula:

Start address = partition start address + (<n> * 64 K), where <n> is the image number

Figure A-10: SD/MMC Flash Image Layout



The SD/MMC controller supports two booting modes:

- MBR (partition) mode
- The boot image is read from a custom partition (0xA2)
- The first image is located at the beginning of the partition, at offset 0x0
- Start address = partition start address
- Raw mode
- If the MBR signature is not found, SD/MMC driver assumes it is in raw mode.
- The boot image data is read directly from sectors in the user area and is located at the first sector of the SD/MMC.
- The first image is located at the start of the memory card, at offset 0
- Start address = 0

The MBR contains the partition table, which is always located on the first sector (LBA0) with a memory size of 512 bytes. The MBR consists of executable code, four partition entries, and the MBR signature. A MBR can be created by specific tools like the FDISK program.

Table A–5 lists the MBR structure.

Table A-5: MBR Structure

Offset	Size (Byte)	Description
0x000	446	Code area
0x1BE	16	Partition entry for partition 1
0x1CE	16	Partition entry for partition 2
0x1DE	16	Partition entry for partition 3
0x1EE	16	Partition entry for partition 4
0x1FE	2	MBR signature: 0xAA55

The standard MBR structure contains a partition with four 16-bytes entries. Thus, memory cards using this standard table cannot have more than four primary partitions or up to three primary partitions and one extended partition.

Each partition type is defined by the partition entry. The boot images are stored in a primary partition with custom partition type (0xA2). The SD/MMC flash driver does not support a file system, so the boot images are located in partition A2 at fixed locations.

Table A-6: Partition Entry

Offset	Size (Byte)	Description
0x0	1	Boot indicator. 0x80 indicates that is it bootable.
0x1	3	Starting CHS value
0x4	1	Partition type
0x5	3	Ending CHS value
0x8	4	LBA of first section in partition
0xB	4	Number of sectors in partition

The boot ROM code configures the SD/MMC controller to default settings for the supported SD/MMC flash memory.

Related Information

http://www.altera.com/literature/hb/arria-v/av_54011.pdf

For more information about the SD/MMC, refer to the *SD/MMC Controller* chapter in the Arria V Device Handbook, Volume 3.

Default settings of the SD/MMC controller.

Table A-7: SD/MMC Controller Default Settings

Parameter	Default	Register Value
Card type	1 bit	The card type register (`ctype`) in the SD/MMC controller registers (`sdmmc`) = 0x0
Timeout	Maximum	The timeout register (`tmout`) = 0xFFFFFFFF
FIFO threshold RX watermark level	1	The RX watermark level field (`rx_wmark`) of the FIFO threshold watermark register (`fifoth`) = 0x1
Clock source	0	The clock source register (`clksrc`) = 0x0

Parameter		Default	Register Value
Block size		512	The block size register (blksiz) = 0x200
Clock divider	Identification mode	32	The clock divider register (clkdiv)= 0x10 (2*16=32)
	Data transfer mode	Bypass	The clock divider register (clkdiv)= 0x00

CLKSEL Pin Settings for the SD/MMC Controller

Table A-8: SD/MMC Controller CLKSEL Pin Settings

Setting		CLKSEL Pin			
		0	1	2	3
osc1_clk (EOSC1 pin) range		10–50 MHz	10–12.5 MHz	12.5–25 MHz	25–50 MHz
ID mode	Device clock (sdmmc_cclk_out)	osc1_clk/128, 391 KHz max	osc1_clk/32, 391 KHz max	osc1_clk/64, 391 KHz max	osc1_clk/128, 391 KHz max
	Controller baud rate divisor	32	32	32	32
Data transfer mode	Device clock (sdmmc_cclk_out)	osc1_clk/4, 12.5 MHz max	osc1_clk*1, 12.5 MHz max	osc1_clk/2, 12.5 MHz max	osc1_clk/4, 12.5 MHz max
	Controller baud rate divisor (even numbers only)	1 (bypass)	1 (bypass)	1 (bypass)	1 (bypass)
Controller clock (sdmmc_clk)		osc1_clk, 50 MHz max	osc1_clk, 50 MHz max	osc1_clk, 50 MHz max	osc1_clk*2, 50 MHz max
mpu_clk		osc1_clk, 50 MHz max	osc1_clk*32, 400 MHz max	osc1_clk*16, 400 MHz max	osc1_clk*8, 400 MHz max
PLL modes		Bypassed	Locked	Locked	Locked

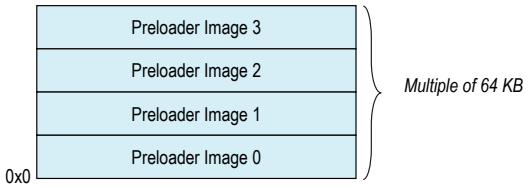
SPI and Quad SPI Flash Devices

The figure below shows the SPI and quad SPI flash image layout. The preloader image is always located at offsets which are multiples of 64 KB. The boot ROM code only supports QSPI devices with Mode Reset Command. Common manufacturers for Mode Reset Command are Spansion and Winbond.

The first image is located at offset 0 and followed by subsequent images. The start address of each image is based on the following formula:

Start address = (N * 64K), where N is the image number

Figure A-11: SPI and Quad SPI Flash Image Layout



The boot ROM code configures the quad SPI controller to default settings for the supported SPI or quad SPI flash memory.

Related Information

http://www.altera.com/literature/hb/arria-v/av_54012.pdf
For more information about the quad SPI flash memory, refer to the *Quad SPI Flash Controller* chapter in the Arria V Device Handbook, Volume 3.

Quad SPI Controller Default Settings

Table A-9: Quad SPI Controller Default Settings

Parameter	Default Setting	Register Value
SPI baud rate.	Divide by 4	The master mode baud rate divisor field (`bauddiv`) of the quad SPI configuration register (`cfg`) in the quad SPI controller registers (`qspiregs`) = 1.
Read opcode.	CLKSEL = 9 or 1: Normal read CLKSEL = 2 or 3: Fast read	The read opcode in non-XIP mode field (`rdopcode`) in the device read instruction register (`devrd`) = 0x3 (for normal read) and 0xB (for fast read).
Instruction type.	Single I/O (1 bit wide)	The address transfer width field (`addrwidth`) and data transfer width field (`datawidth`) of the `devrd` register = 0.
Delay in master reference clocks for the length that the master mode chip select outputs are deasserted between words when the clock phase is zero.	200 ns	The clock delay for chip select deassert field (`nss`) in the quad SPI device delay register (`delay`). Refer to `delay[31:24]` in Table SPI and Quad SPI Flash Delay Configuration.

Parameter	Default Setting	Register Value
Delay in master reference clocks between one chip select being deactivated and the activation of another. This delay ensures a quiet period between the selection of two different slaves and requires the transmit FIFO to be empty.	0 ns	The clock delay for chip select deactivation field (`btwn`) in the `delay` register = 0x0.
Delay in master reference clocks between the last bit of the current transaction and the first bit of the next transaction. If the clock phase is zero, the first bit of the next transaction refers to the cycle in which the chip select is deselected.	20 ns	The clock delay for last transaction bit field (`after`) in the `delay` register. Refer to `delay[15:8]` in Table SPI and Quad SPI Flash Delay Configuration.
Added delay in master reference clocks between setting `qspi_n_ss_out` low and first bit transfer.	20 ns	The clock delay with `qspi_n_ss_out` field (`init`) in the `delay` register. Refer to `delay[7:0]` in Table SPI and Quad SPI Flash Delay Configuration.
Number of address bytes.	3 bytes	The number of address bytes field (`numaddrbytes`) of the device size register (`devsz`) = 2. Note: Before a reset, you must ensure that the QSPI flash device is configured to 3 bytes address mode for the boot ROM to function properly.

Quad SPI Controller CLKSEL Pin Settings

Table A-10: Quad SPI Controller CLKSEL Pin Settings

Setting	CSEL Pin			
	0	1	2	3
`osc1_clk` (`EOSC1` pin) range	10–50 MHz	20–50 MHz	25–50 MHz	10–25 MHz
Device clock (`sclk_out`)	`osc1_clk`/4, 12.5 MHz max	`osc1_clk`/2, 25 MHz max	`osc1_clk`*1, 50 MHz max	`osc1_clk`*2, 50 MHz max
Controller clock (`qspi_clk`)	`osc1_clk`, 50 MHz max	`osc1_clk`*2, 100 MHz max	`osc1_clk`*4, 200 MHz max	`osc1_clk`*8, 200 MHz max
Controller baud rate divisor (even numbers only)	4	4	4	4

Setting	CSEL Pin			
	0	1	2	3
Flash read instruction (1 dummy byte for READ_FAST)	READ	READ	READ_FAST	READ_FAST
mpu_clk	osc1_clk, 50 MHz max	osc1_clk*8, 400 MHz max	osc1_clk*8, 400 MHz max	osc1_clk*16, 400 MHz max
PLL modes	Bypassed	Locked	Locked	Locked

SPI and Quad SPI Flash Delay Configuration

The delay register in the quad SPI controller registers (qspiregs) configures relative delay into the generation of the master output signals.

The SPI or quad SPI flash memory needs to meet the following timing requirements:

- T_{SLCH} (delay[7:0]): 20 ns
- T_{CHSH} (delay[15:8]): 20 ns
- T_{SHSL} (delay[31:24]): 200 ns

Table A-11: SPI and Quad SPI Flash Delay Configuration

CSEL Pin	T_{ref_clk} (ns)	T_{sclk_out} (ns)	Device Delay Register		
			delay[7:0]	delay[15:8]	delay[31:24]
0	20	80	1	1	6
1	10	40	2	2	16
2–3	5	20	4	4	36

The formula to calculate the delay is

delay[7:0] = T_{SLCH}/T_{qspi_clk}

delay[15:8] = T_{CHSH}/T_{qspi_clk}

delay[31:24] = $(T_{SHSL} - T_{sclk\_out})/T_{qspi\_clk}$

FPGA Configuration

You can configure the FPGA portion of the SoC device with non-HPS sources or by utilizing the HPS. Software executing on the HPS configures the FPGA by writing the configuration image to the FPGA manager in the HPS. Software can control the configuration process and monitor the FPGA status by accessing the control and status register (CSR) interface in the FPGA manager.

Related Information

- **Configuration, Design Security, and Remote System Upgrades**
  For more information about configuring the FPGA in general, refer to the *Configuration, Design Security, and Remote System Upgrade* appendix in the Arria V Device Handbook, Volume 1.

- **http://www.altera.com/literature/hb/arria-v/av_54013.pdf**
  For more information about configuring the FPGA through the HPS FPGA manager, refer to the *FPGA Manager* chapter in the Arria V Device Handbook, Volume 3.

Full Configuration

The HPS uses the FPGA manager to configure the FPGA portion of the device. The following sequence suggests one way for software to perform a full configuration:

- CONF_DONE = 1 and nSTATUS = 1 indicates successful configuration.
- CONF_DONE = 0 or nSTATUS = 0 indicates unsuccessful configuration. Complete steps 12 and 13, then go back and repeat steps 3 to 10 to reload the configuration image.
- If DCLK is unused, write a value of 4 to the DCLK count register (dclkcnt).
- If DCLK is used, write a value of 20,480 (0x5000) to the dclkcnt register.

If the HPS resets in the middle of a normal configuration data transfer before entering user mode, software can assume that the configuration is unsuccessful. After the HPS resets, software must repeat the steps for full configuration.

1. Set the cdratio and cfgwdth bits of the ctrl register in the FPGA manager registers (fpgamgrregs) to match the characteristics of the configuration image. These settings are dependant on the MSEL pins input.
2. Set the nce bit of the ctrl register to 0 to enable HPS configuration.
3. Set the en bit of the ctrl register to 1 to give the FPGA manager control of the configuration input signals.
4. Set the nconfigpull bit of the ctrl register to 1 to pull down the nCONFIG pin and put the FPGA portion of the device into the reset phase.
5. Poll the mode bit of the stat register and wait until the FPGA enters the reset phase.
6. Set the nconfigpull bit of the ctrl register to 0 to release the FPGA from reset.
7. Read the mode bit of the stat register and wait until the FPGA enters the configuration phase.
8. Clear the interrupt bit of nSTATUS (ns) in the gpio interrupt register (fpgamgr-regs.mon.gpio_porta_eoi).
9. Set the axicfgen bit of the ctrl register to 1 to enable sending configuration data to the FPGA.
10. Write the configuration image to the configuration data register (data) in the FPGA manager module configuration data registers (fpgamgrdata). You can also choose to use a DMA controller to transfer the configuration image from a peripheral device to the FPGA manager.
11. Use the fpgamgrregs.mon.gpio_ext_porta registers to monitor the CONF_DONE (cd) and nSTATUS (ns) bits.
12. Set the axicfgen bit of the ctrl register to 0 to disable configuration data on AXI slave.
13. Clear any previous DONE status by writing a 1 to the dcntdone bit of the DCLK status register (dclkstat) to clear the completed status flag.
14. Send the DCLKs required by the FPGA to enter the initialization phase.
15. Poll the dcntdone bit of the DCLK status register (dclkstat) until it changes to 1, which indicates that all the DCLKs have been sent.
16. Write a 1 to the dcntdone bit of the DCLK status register to clear the completed status flag.
17. Read the mode bit of the stat register to wait for the FPGA to enter user mode.
18. Set the en bit of the ctrl register to 0 to allow the external pins to drive the configuration input signals.

Document Revision History

Table A-12: Document Revision History

Date	Version	Changes
February 2014	2014.02.28	Correction to "Leading the Preloader" section
December 2013	2013.12.30	• Updated figures in the Booting and Configuration Introduction section. • Updated the Rest and Boot ROM sections. • Updated the Shared Memory Block table. • Updated register names in the Full Configuration section.
November 2012	1.3	• Expanded shared memory block table. • Added CLKSEL tables. • Additional minor updates.
June 2012	1.2	Updated the HPS boot and FPGA configuration sections.
May 2012	1.1	• Updated the HPS boot section. • Added information about the flash devices used for HPS boot. • Added information about the FPGA configuration mode.
January 2012	1.0	Initial release.