KeyStone II Architecture
ARM Bootloader

User Guide

Literature Number: SPRUHJ3
July 2013

Release History

Release	Date	Description
SPRUGHJ3	July 2013	Initial Release

Contents

List of Tables

List of Figures

Preface

About This Manual

This document describes the features of the on-chip bootloader provided with the ARM Cortex-A15 processor.

IMPORTANT NOTE—This document should be used in conjunction with the device-specific data manuals, KeyStone Architecture Bootloader user guide and user guides for peripherals used during the boot. This document supports only non-secure boot mode.

Notational Conventions

This document uses the following conventions:

- Commands and keywords are in **boldface** font.
- Arguments for which you supply values are in *italic* font.
- Terminal sessions and information the system displays are in `screen font`.
- Information you must enter is in **`boldface screen font`**.
- Elements in square brackets ([ ]) are optional.

Notes use the following conventions:

Note—Means reader take note. Notes contain helpful suggestions or references to material not covered in the publication.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

CAUTION—Indicates the possibility of service interruption if precautions are not taken.

WARNING—Indicates the possibility of damage to equipment if precautions are not taken.

Related Documentation from Texas Instruments

C66x CorePac User Guide	SPRUGW0
DDR3 Memory Controller for KeyStone Devices User Guide	SPRUGV8
External Memory Interface (EMIF16) for KeyStone Devices User Guide	SPRUGZ3
Gigabit Ethernet (GbE) Switch Subsystem (1GB) for KeyStone Devices User Guide	SPRUGV9
HyperLink for KeyStone Devices User Guide	SPRUGW8
Inter Integrated Circuit (I^2C) for KeyStone Devices User Guide	SPRUGV3
Multicore Shared Memory Controller (MSMC) for KeyStone Devices User Guide	SPRUGW7
Peripheral Component Interconnect Express (PCIe) for KeyStone Devices User Guide	SPRUGS6
Phase Locked Loop (PLL) Controller for KeyStone Devices User Guide	SPRUGV2
Power Sleep Controller (PSC) for KeyStone Devices User Guide	SPRUGV4
Serial Peripheral Interface (SPI) for KeyStone Devices User Guide	SPRUGP2
Serial RapidIO (SRIO) for KeyStone Devices User Guide	SPRUGW1
KeyStone Architecture Bootloader User Guide	SPRUGY5B
Universal Asynchronous Receiver/Transmitter (UART) for KeyStone Devices User Guide	SPRUGP1

Trademarks

Chapter 1

Introduction

IMPORTANT NOTE—The information in this document should be used in conjunction with information in the device-specific KeyStone II Architecture data manual that applies to the part number of your device.

This document describes the features of the on-chip ROM Boot Loader (RBL) provided with KeyStone II devices with ARM cortex support.

This document should be used in conjunction with the device-specific data manuals and user guides for peripherals used during the boot. This document applies for all ARM master boot modes on non-secure devices only.

1.1 Bootloader Features

The ARM ROM Boot Loader (RBL) is software that resides in on-chip read-only memory (ROM) that assists the customer in transferring and executing their application code. The start address of the RBL is 0x00000000. In order to accommodate various system scenarios, the RBL supports several boot modes. These boot modes can be broadly classified as host or memory boot modes.

During a host boot, the device is configured to receive code from a host via the selected interface. Either the host writes the application code directly into internal memory or the RBL receives the application code on the selected interface and stores it in internal memory.

During a memory boot, the device transfers code from non-volatile memory to internal memory for execution. Because different KeyStone II devices support different sets of boot modes, see the device-specific data manual to obtain the list of boot modes supported in that device.

The boot operation can be divided into two sections: initialization and boot process. During initialization, the RBL configures the device resources as needed to support the boot process. The resources used depend on the boot mode requirements. During the boot process the boot image is loaded into device memory and executed. The actions performed during the initialization and boot processes depend on the following factors:

- The trigger that initiated the boot operation
- The boot mode and location of the boot image (host or memory) as defined by the configuration specified on the boot mode pins of the device

This document discusses the different triggers that can initiate the boot operation, the initialization process, and the boot process for the various boot modes.

The ARM in various KeyStone II devices supports the following boot modes:

- **Sleep boot:** Basic initialization is performed but no user application is loaded or executed
- **I²C Master boot:** Basic initialization is performed and the user application is read from an I²C slave nonvolatile memory
- **SPI boot:** Basic initialization is performed and the user application is read from an SPI slave nonvolatile memory
- **EMIF boot:** Basic initialization is performed and the user application is copied from NOR flash connected to the EMIF hardware block of the device
- **NAND boot:** Basic initialization is performed and the user application is read from an external NAND flash connected to the EMIF hardware block of the device
- **SRIO boot:** Basic initialization is performed and the user application is received via the SRIO interface
- **Ethernet boot:** Basic initialization is performed and the user application is received with the TFTP protocol over the Ethernet interface
- **PCIe boot:** Basic initialization is performed and the RBL then waits while an external host writes the user application directly into internal memory via the PCIe interface

- **HyperLink boot:** Basic initialization is performed and the RBL then waits while an external host writes the user application directly into internal memory via HyperLink
- **UART boot:** Basic initialization is performed and the RBL then waits to receive data from an external host via the UART serial connection

1.2 Terms and Abbreviations

Term	Definition
I²C	Inter-Integrated Circuit
MSMC	Multicore Shared Memory Controller
PCIe	Peripheral Component Interconnect Express
POR	Power on Reset
RBL	ROM Boot Loader
SPI	Serial Peripheral Interface
SRIO	Serial Rapid Input/Output

Chapter 2

Reset Types and Boot Configurations

2.1 Introduction

The actions performed by the RBL vary based on the what triggers boot and the selected boot mode. Resets are used to trigger boot in KeyStone II devices. There are four types of reset supported in the KeyStone II architecture:

- Power-On reset (POR)
- Hard reset
- Soft reset
- Local reset

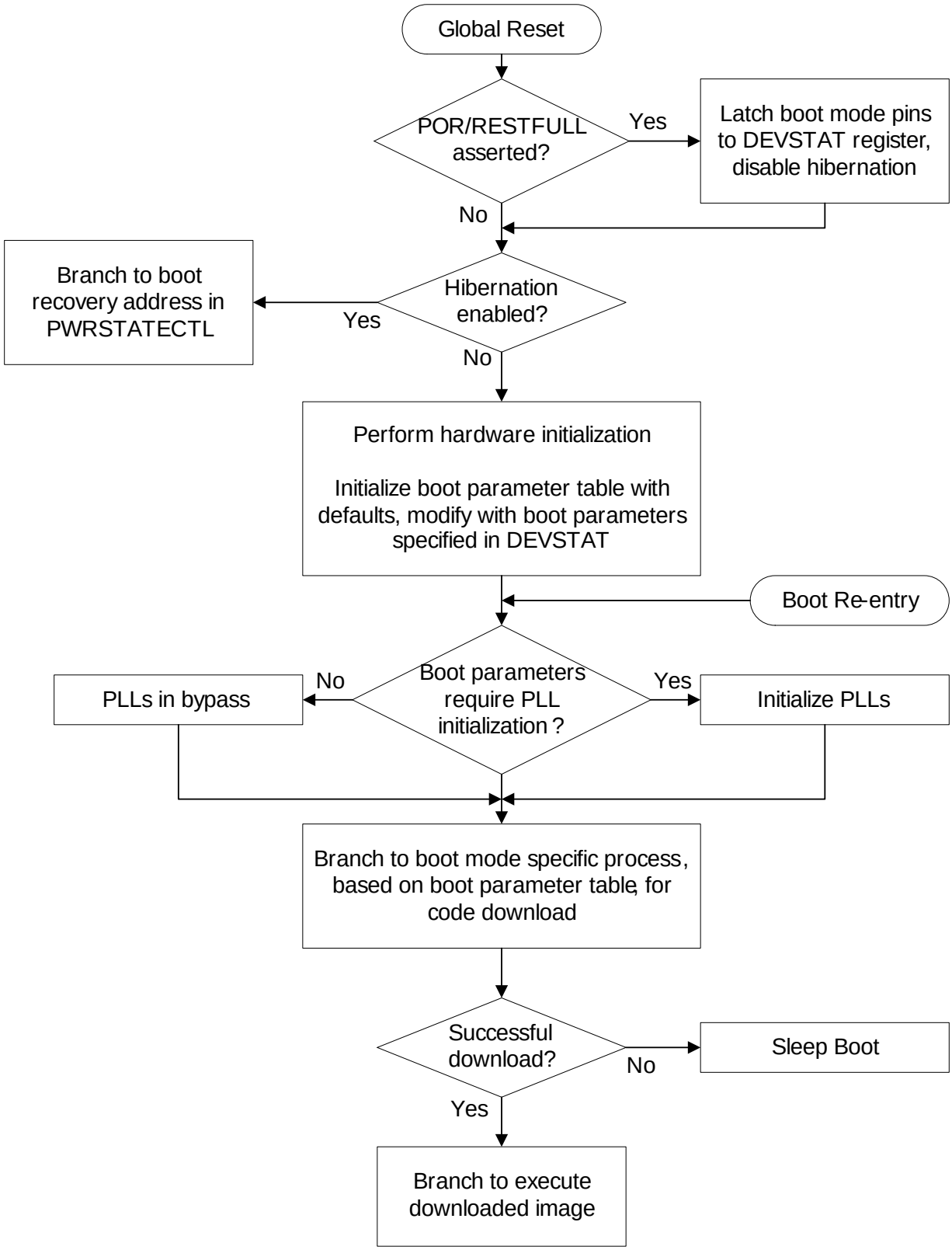
POR, hard, and soft resets are global resets that affect the entire device. A local reset is used to reset only a single processor. The ARM may perform local reset of the DSPs, but the ARM cannot be reset by a local reset. It is in reset when the full device is in reset. For further details on the reset types, see the device-specific data manual.

External pins on the device are used at device power-up to allow customers to select the desired boot mode. There are also pins that specify some of the configuration parameters used during that boot mode. Some boot modes also include a *minimum boot pin select* pin that can be used to select configurable parameters instead of selecting them with the boot configuration pins. The value of these pins is latched into the device status register during POR. The number of pins used to select the boot mode and the parameter varies depending on the device. See the device-specific data manual. for more information.

The RBL also handles hibernation recovery and provides a multi-stage boot mechanism.

Figure 2-1shows a high level overview of the EBL functionality.

Figure 2-1 Boot High Level Overview



2.2 Bootloader Initialization After Power-On Reset

Power-on reset (POR) resets the entire device. Everything on the device is reset to its default state. POR is initiated by either the $\overline{\text{POR}}$ or $\overline{\text{RESETFULL}}$ external pins. The $\overline{\text{POR}}$ pin is asserted during power up of the device while the $\overline{\text{RESETFULL}}$ pin can be asserted by a host to reset the device after it is already powered up. $\overline{\text{POR}}$ causes all peripherals to be initialized to their default states and boot processing to be started. The state of the boot configuration pins are latched into the device status register (DEVSTAT) at POR. The RBL uses the values in DEVSTAT to select how boot is performed. All initialization and boot processing is performed by ARM core 0.

The RBL uses the boot configuration information from the DEVSTAT register to determine what initialization to perform. Initialization executed by the RBL includes:

- The RBL enables reset isolation in all peripherals that support it. The power state of these peripherals is not changed. The device-specific data manual lists the peripherals that support reset isolation.
- The RBL enables the power and clock domains for any peripherals required during boot.
- Various PLLs may be configured depending on values specified in DEVSTAT. DEVSTAT will contain information indicating the PLL, the input clock frequency, and when that PLL should be configured.
  - When the ARM PLL CONFIG value is specified in DEVSTAT, the ARM PLL is programmed so that the output of the ARM PLL is the frequency indicated by the ARMSPEED field in the DEVSPEED register.
  - When the SYS PLL CONFIG value is specified in DEVSTAT, the system PLL is programmed so that the output of the system PLL is the frequency indicated by the DEVSPEED field in the DEVSPEED register.
  - When information concerning the PA PLL clock input is specified, the PA PLL is programmed so that the output of the PA PLL is the frequency indicated in the data manual for the device.
- All interrupts are disabled except for IPC interrupts and the host interrupts that are used for an external host boot modes (PCIe, SRIO, and HyperLink).
- All secondary ARM cores are held in reset during the boot process. All DSP cores execute an IDLE command.
- All cache is disabled.
- The RBL uses the boot configuration information in DEVSTAT to setup and initialize a boot parameter table that is used to control the boot process. This table is stored in MSMC SRAM. Some information in the table is initialized based on the configuration parameters in DEVSTAT while the remaining information is default values based only on the boot mode. The format of the table varies depending on the boot mode. All start with a few entries that are common to all boot modes. Information about the boot parameter table can be found in the device-specific data manual.

2.3 Bootloader Initialization Process After Hard or Soft Reset

A warm reset of the non-isolated portions of the device can be initiated by the external RESET pin, the watchdog timer, or by software writing to a PLL Controller register. The reset can be configured as a hard reset or a soft reset. By default, it will be configured as a hard reset. When a hard reset is performed, all modules except the test logic, emulation logic, and reset-isolated modules will be reset. When a soft reset is performed, some of the MMRs and memory are preserved in addition to the modules that are not reset during a hard reset. See the data manual for the specific device for more details about hard and soft reset.

After the hardware portion of the reset is performed, the RBL will determine if hibernation was enabled. If hibernation was not enabled, the initialization listed in section Section 2.2 will be performed. This means the main difference between POR and hard/soft reset is that for hard/soft reset the boot configuration pins are not latched into DEVSTAT and any modules set for reset isolation are not reset.

2.4 Bootloader Initialization after Hibernation

Hibernation can be used to reduce power consumption when processing is not needed while allowing for a fast recovery. The RBL may be used to put the device into hibernation. Or, the device can be put into the hibernation state directly by the user application code. The RBL will be involved with exiting the hibernation state.

Before shutting down the cores, the user application must set the hibernation mode, hibernation enable bits, and the recovery address into the PWRSTATECTL Register. The contents of PWRSTATECTL are maintained through hard and soft reset. After hard or soft reset, the RBL samples the PWRSTATECTL register to determine if hibernation recovery should be performed. If hibernation is enabled, the RBL resets some peripherals (based on hibernation mode), and branches to the hibernation recovery address specified in PWRSTATECTL.

There are two hibernation modes.

Hibernation 1 mode should be used when critical user application code and data are stored in MSMC SRAM. The RBL provides a callable function to enter this hibernation mode. When making the call, the user application code must provide the desired hibernation mode and recovery address to the function. The recovery address must be 10-bit aligned. It may be in the MSMC or DDR. The function will set DDR3 for self-refresh and configure the Chip Miscellaneous Control register (CHIP_MISC_CTL0) to block the reset of the parity SRAM when hibernation recovery is performed. It will then shut down all module and power domains of the device except for MSMC to enter Hibernation 1 state. In this hibernation mode the contents of the MSMC SRAM are preserved, but the MSMC MMRs are not preserved. This means the contents of these MMRs should be saved in MSMC SRAM before entering hibernation. The values will then be available to restore to the MMRs by the recovery code. When Hibernation 1 recovery is performed, the RBL will disable DDR self-refresh, clear the hibernation enable bit in PWRSTATECTL, and branch to the hibernation recovery address.

Hibernation 2 mode is the same as Hibernation 1 except that it will also shut down the MSMC module and power domains on entry to the hibernation state. This hibernation mode is intended for use when critical user application code and data are stored in external DDR. The recovery address must be in external DDR.

After hibernation, all secondary ARM cores and all DSPs function as they do on initial power up.

Before entering hibernation mode, the user can enable or disable reset isolation for the SRIO. When the SRIO has reset-isolation enabled before entering hibernation, the user should also make the LPSC for SRIO active because packet forwarding requires the VBUS clock to function. When SRIO has reset-isolation disabled before entering hibernation, the SRIO block must be disabled by the user. This includes stopping the VBUS clock and disabling the PHY layer. Stopping the VBUS clock without disabling the PHY layer can cause system congestion and system hang.

When the PCIe is used in EP mode, the user must put the PCIe in the L1 powerdown mode before it enters the hibernation mode. The PCIe power domain should be kept on during hibernation mode. When the device exits hibernation mode, the RC device can issue a reset request to all the EP points to bring the PCIe endpoint alive.

To avoid resetting the DDR during hard reset, reset isolation is provided for the DDR. The boot code enables the DDR reset isolation by default and the user has the option to turn off the reset isolation feature if it is not needed (See the *Power Sleep Controller (PSC) for KeyStone Devices User Guide* in "Related Documentation from Texas Instruments" on page ø-viii for disabling the reset isolation for DDR3.) Because the DDR contents are preserved, the PLL for DDR3 EMIF must stay locked and DDR PHY must be active to preserve the DDR3 content. This avoids full calibration when resuming the normal operation; full calibration can corrupt the DDR3 content. In summary, the DDR is alive during both hibernation modes and the DDR3 can be put into self-refresh mode to save power.

2.5 Bootloader Operation on Secondary Cores

On device power up with ARM as the boot master, all secondary ARM cores are held in reset. When a secondary ARM core is needed, any active core may load code for the secondary core to the execution address, write the execution address to the boot magic register, generate an IPC interrupt to the core, and take the core out of reset. The secondary core will then come out of reset and begin executing the code provided to it.

On device power up with ARM as the boot master, all DSP cores perform basic initialization and then enter the idle state. When a DSP is needed, any active core may load code from the DSP core to the execution address, write the execution address to the boot magic address, and generate an IPC interrupt to the core. The DSP will then come out of the idle stat and begin executing the code provided to it.

2.6 Multi-Stage Boot

The RBL also provides a multi-stage boot mechanism. This feature can be used when a single stage boot does not provide the desired flexibility. The typical use case for a multi-stage boot is to perform a basic boot using the boot mode and parameter settings available in DEVSTAT, let the downloaded code modify the boot parameter table to select new boot parameters or a new boot mode, and then branch to the boot re-entry point to perform a second boot. The initial boot parameter table is built by the RBL as described in Section 2.2. The boot parameter table is not modified by the by the RBL when the re-entry point is used.

An example of why a multi-stage boot may be desired is for SPI boot. When a default SPI boot is performed it does not program any of the device PLLs; they are left in bypass mode. When the ARM PLL is in bypass, the ARM will be clocked by the input clock instead of the fast output of the PLL. This means that code download across the SPI interface will be slower than if the ARM was running at full speed. Using a multi-stage boot, a very small program can be downloaded using the default SPI boot. When executed, this small program can modify the boot parameter table so that it directs the RBL to program the ARM PLL. After modifying the boot parameter table, the program simply branches to the boot re-entry location to perform a secondary SPI boot with the device now running at full speed.

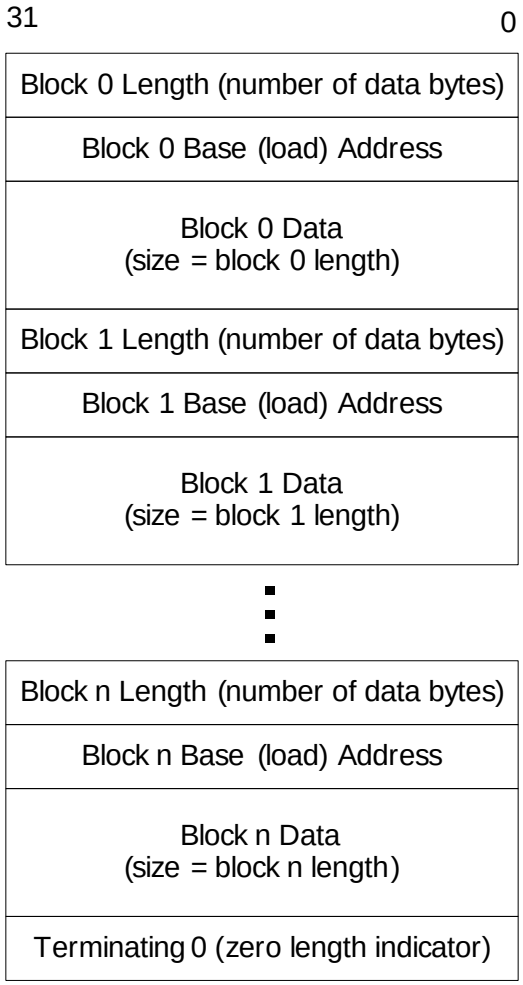
2.7 Boot Image Formats

Two boot image formats are used for ARM master boot modes. Each boot mode expects the image it downloads to be in one of these formats. The following sections describe the two formats. See the specific boot mode description to determine the boot image format used for each boot mode.

2.7.1 GP Header Boot Image Format

The GP (General Purpose) header image format is blocks of data, each preceded with an 8-byte header. The 8-byte header consists of a 4-byte length and a 4-byte base address. A block of data follows each header. The data block must contain the number of bytes specified in the block length field. Each GP header image must contain at least one data block, but may contain more. Additional blocks are appended to the previous blocks. The image ends with a a trailing 4-byte length field containing 0. This image format is usually used for boot modes in which the image is read from non-volatile memory. The general format of a GP header image is shown in Figure 2-2.

Figure 2-2 GP Header Boot Image Format



The data in a GP header formatted image is processed as it is read from the boot source. The length field in each header tells the boot data processor how many bytes to expect in that block and the base address tells it where to store the data. Data processing continues until a terminating length of 0 is encountered. After all blocks have been read, the RBL branches to the last base address specified in the image to begin execution of the image.

A tool that can take a standard binary image and convert it to GP header boot image format is provided in the MCSDK.

2.7.2 Blob Boot Image Format

The blob image format is simply a binary load image with the entry point being the first byte of the image. This image format is generally used for boot modes where the image is received from a host.

In boot modes in which the host does not have direct access to device memory, the image will be stored at a pre-determined address, typically the base address of MSMC SRAM. In these cases, the image must be linked to execute from that address. After the complete image has been received and stored, the RBL branches to the image storage address for execution.

In boot modes in which the host has direct access to device memory, the host may store the image anywhere in device memory. After writing the image into device memory, the host must write the load address at the *boot magic* address location. When the RBL sees a non-zero value as the boot magic address data, it branches to the address written.

Chapter 3

Boot Modes

3.1 Sleep Boot

The sleep boot mode does not result in any image being loaded for execution. In the ARM master sleep boot mode, the ARM simply performs ARM and system PLL initialization as specified by the boot mode pins and then executes a polling loop of its magic address. This mode is typically used in debug and development environments in which code images are loaded using an emulator.

3.2 I²C Slave Boot

The I²C slave boot mode is used to receive the boot image from an I²C master connected to one of the device I²C ports. In this boot mode, the device acts as the I²C slave. The image received in this boot mode must be in GP Header format.

The boot pins provide a way to specify:

- KeyStone I²C slave bus address
- I²C port number

See the device-specific data manual for the specific boot mode pins and definitions.

The RBL will initialize the I²C hardware and then start polling the specified I²C port for data. As data is read from the I²C port, it will be stored at the address specified in the GP header. The RBL will continue polling for and reading data until a complete image has been read. Data will be read and processed in 2-Kbyte chunks. The I²C master should pad the image with zeroes to be a multiple of this size. When the complete image has been read, the RBL will branch to the base address of the last GP data block in the image.

3.3 I²C Master Boot

The I²C master boot mode is used to read the boot image from a EEPROM connected to one of the device I²C ports. In this boot mode the device acts as the I²C master. The image read in this boot mode must be in GP Header format.

The boot pins provide a way to specify:

- EEPROM I²C slave bus address
- I²C port number
- Base address offset to use when accessing the EEPROM

The base address offset value can be used to specify an offset to the chip select base address of where to start reading on the EEPROM device. See the device-specific data manual for the specific boot mode pins and definitions for your device.

The RBL will start reading from the I²C EEPROM slave at the specified I²C bus address on the specified I²C port. This read will be done beginning at the specified base address offset. Data will be read in 2-Kbyte chunks. The data will be stored at the address specified in the GP header. It will continue reading GP header formatted data from the EEPROM and storing it as directed by the headers until a complete image has been read. When the complete image has been read, the RBL will branch to the base address of the last GP data block in the image.

3.4 SPI Boot

The SPI boot mode is used to read the boot image from NOR memory connected to the device through the SPI interface. The image read from the NOR memory must be in GP Header format.

The boot pins provide a way to specify:

- SPI port to be used
- Address width for accessing the NOR memory (16-bit or 24-bit)
- Clock phase and polarity for data latching
- Number of pins to drive (3 when no CS is needed, or 4)
- Chip select to use (when used)
- Base address offset to use when accessing the NOR memory

The base address offset value can be used to specify an offset to the chip select base address of where to start reading on the NOR memory. See the device-specific data manual for the specific boot mode pins and definitions for your device.

The RBL will start reading from the NOR memory at the chip select base address plus the base address offset specified by the boot mode pins. The data will be stored at the address specified in the GP Header. It will continue reading GP Header formatted data from the NOR memory and storing it as directed by the headers until a complete image has been read. When the complete image has been read, the RBL will branch to the base address of the last GP data block in the image.

3.5 EMIF Boot

The EMIF boot mode is used to read the boot image from NOR memory connected to the EMIF interface. When the ARM is the boot master, the NOR memory must be connected to the device using EMIF CS2 (CE0). The image read from the NAND must be in GP Header format.

The boot pins provide a way to specify:

- NOR memory width (8-bit or 16-bit)
- Extended wait mode (enable or disable)
- Base address offset to use when accessing the NOR memory

The base address offset value can be used to specify an offset to the chip select base address of where to start reading on the NOR memory. See the device-specific data manual for the specific boot mode pins and definitions for your device.

The RBL will start reading from the NOR memory at the chip select base address plus the base address offset specified by the boot mode pins. The data will be stored at the address specified in the GP header. It will continue reading GP Header formatted data from the NOR memory and storing it as directed by the headers until a complete image has been read. When the complete image has been read, the RBL will branch to the base address of the last GP data block in the image.

3.6 NAND Boot

The NAND boot mode is used to read the boot image from NAND memory connected to the device through the EMIF interface. When the ARM is the boot master, the NAND memory must be connected to the device using EMIF CS2 (CE0). The image read from the NAND memory must be in GP Header format.

To read from a NAND memory device, the ARM must know some information about the geometry of the device. The required information includes:

- Data transfer size: 8-bit or 16-bit
- Page size
- Block size (pages per block)
- Number of blocks
- Number of spare bytes per page (used for ECC data)
- Number of address cycles

Once the geometry is known, the RBL can decide if the NAND memory is compatible with the NAND read algorithms embedded in the RBL. If not, boot halts.

If the NAND is compatible, the boot image will be read from the NAND memory and the data blocks stored in the location(s) specified in the headers ending with the RBL branching to the base address of the last block to begin execution of the image.

The RBL uses has several methods of determining the NAND device geometry. The first method is to *ask* the device if it is ONFI-compliant. To determine if the device is ONFI compliant, a standard read ID command is issued to device address 0x20 and four bytes read. If the four bytes contain 0x49464E4F (ASCII for ONFI), the RBL knows the device is ONFI-compliant. It will then read the ONFI parameter table from the device that contains the geometry information.

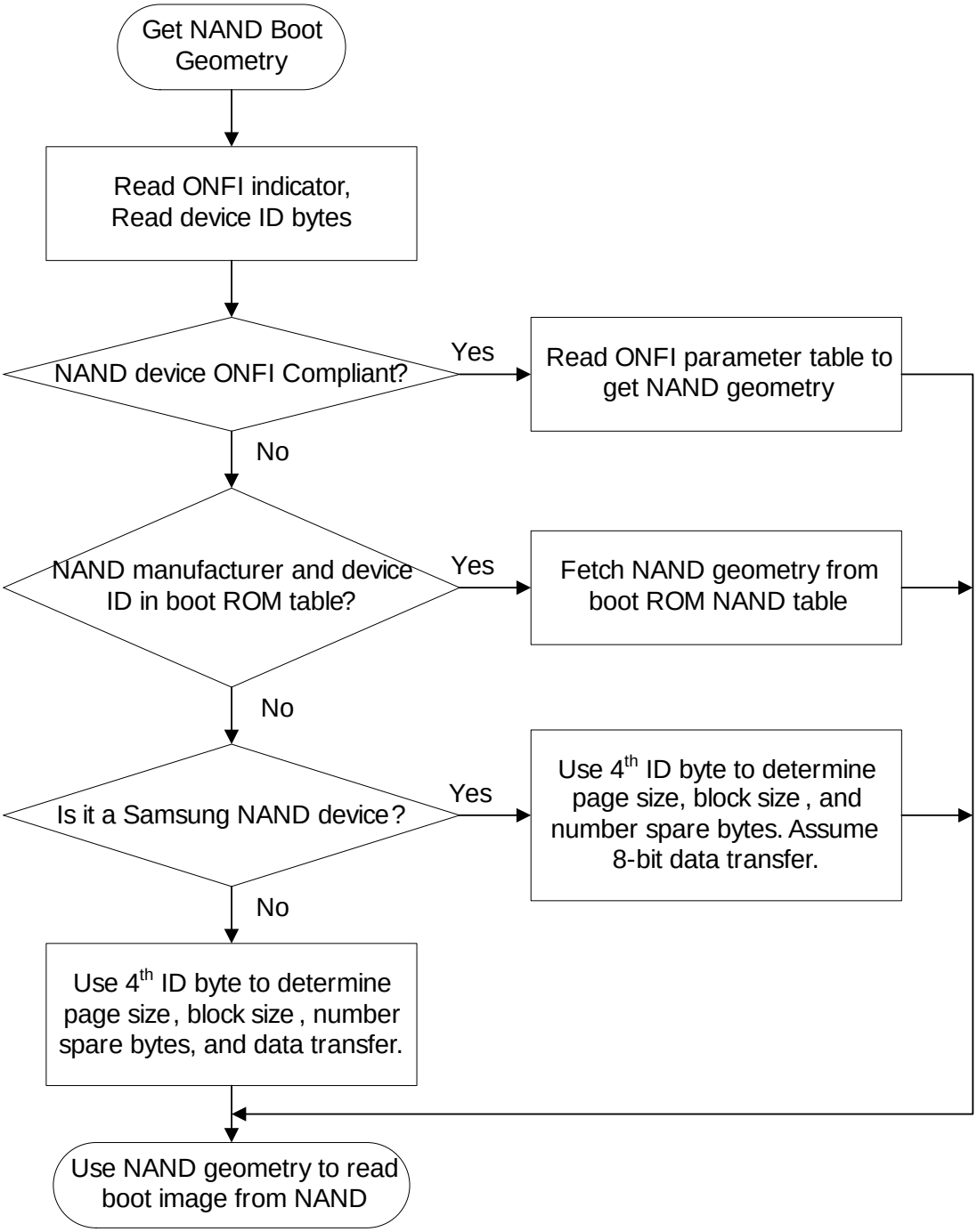
The manufacturer and device ID will always be read from the device by issuing a read ID command to device address 0x00 and four bytes read. The first byte is the manufacturer ID. The second byte is the device ID. The third and fourth bytes imply information about the device geometry.

If the device was found to not be ONFI-compliant, the RBL will compare the manufacturer and device ID to a table of known devices stored in the RBL. This table includes geometry information for each device listed in the table. Please note that given how fast the NAND devices evolve, it is impossible for the table to have information about all devices on the market. The stored geometry information was current when the RBL was created. If manufacturers reuse device IDs, the information in the RBL table may not match the actual device.

If the NAND device is not ONFI-compliant and information about it was not found in the RBL table, the RBL will make a *best guess* of the geometry based on the manufacturer ID and the fourth ID byte read from the device. For NAND devices manufactured by any company except Samsung, the fourth ID byte can be use to determine page size, block size, number of spare bytes per page, and the data transfer size. For Samsung manufactured NANDs, the fourth ID byte can be used to determine only page size, block size, and number of spare bytes per page. An 8-bit data transfer size will be assumed for Samsung NAND devices.

The process the RBL uses to determine the device geometry is shown in Figure 3-1.

Figure 3-1 NAND Geometry Determination Process



Once the NAND geometry is known, the GP Header formatted image can be read from the device. The RBL reads the first page of the NAND block indicated in the boot parameter table. Bad blocks are detected by reading the spare bytes of pages 0, 1, and the last page of a block. If the first six bytes of all three spare areas contain 0xFF, the

block is assumed to be good. If any of those bytes contains a value other than 0xFF, the block is assumed to be bad. If a bad block is encountered, the RBL discards any partial boot image already read and goes to the next NAND block to start reading a new boot image.

Data read from the NAND is error-corrected using ECC data stored in the spare bytes for the page. Each page must have at least 16 spare bytes for each 512 data bytes on the page. The RBL uses the 4-bit hardware ECC build into the EMIF interface of the device for ECC calculation. The ECC data for the page is stored in the spare bytes of the page as shown in Figure 3-2. When a 512-byte data segment of a page is read from the device it is error corrected using the ECC data for that segment. If the data cannot be corrected, the RBL discards any partial boot image already read and goes to the next NAND block to start reading a new boot image.

Figure 3-2 NAND ECC Layout in Spare Bytes

Bytes 0 through 5	Bad block marking (no ECC)
Bytes 6 through 15	ECC for data segment 0 of page
Bytes 16 through 21	Unused
Bytes 22 through 31	ECC data for segment 1 of page
Bytes 32 through 37	Unused
Bytes 38 through 47	ECC data for segment 3 of page

⋮

Bytes n*16 through (n*16)+5	Unused
Bytes (n*16)+6 through (n*16)+15	ECC data for segment n of page
Bytes (n*16)+16 through end	Possible unused spare bytes

The RBL will continue reading GP Header formatted data from the NAND and storing it as directed by the headers until a complete image has been read. When the complete image has been read, the RBL will branch to the base address of the last GP data block in the image.

3.7 SRIO Boot

The SRIO boot mode transfers a boot image into internal memory via the Serial Rapid I/O interface. The RBL sets up the QMSS and queries the boot configuration parameter pins for additional interface set up. Rapid I/O, by default, is set up with mailbox mode and messaging mode enabled, but Direct I/O mode is available for writing an image directly into memory.

After peripheral configuration, the bootloader continuously polls the host boot data address. The next step depends on the mode of transfer the host uses.

If the host uses Direct I/O mode, it should write the image directly into the MSMC. It is the host's responsibility to update the host boot data address with the execution address of the image after transfer is complete. Once the bootloader polls a non-zero value in the host boot data address, it will exit the boot function and begin execution at the provided address. The image must be in Blob Format when this mode is used.

If the host uses Messaging mode, the boot image must be in GP Format. The host is responsible for segmenting the boot image into SRIO packets. Each packet must begin with the SRIO message mode boot header that includes: two bytes packet size (including the header) followed by a two-byte checksum for verifying accuracy of transferred data. When the complete image has been transferred, the RBL will branch to the base address of the last GP data block in the image.

3.8 Ethernet Boot

Ethernet booting is performed by transferring an image by bootp/TFTP protocol over the Ethernet interface and executing it. The boot loader configures the SerDes, SGMII and the switch plus the PASS and Multicore Navigator (Packet DMA and QMSS). At power-on reset, the boot parameter pins are queried to configure the Ethernet for transfer. Configuration options are outlined in detail in the device-specific data manual. With ARM as the boot master, the image must be in Blob Format

After device configuration, the bootloader performs a standard bootp/TFTP boot. The device sends a bootp request with its MAC address to a host TFTP server to be assigned an IP from a pool of addresses. After this connection is established, the device is able to receive image data encapsulated in Ethernet packets. Data received is stripped of its network headers and the boot data is stored in the MSMC. After transfer is complete, the bootloader will begin executing the image at the base of MSMC. File size is limited to 5M bytes and has a max transfer time of 60 seconds. The device will broadcast for one minute at times 0, 4, 12, 28, and 60 seconds.

Ethernet packets are accepted only in DIX (Ethernet II) frames with IPv4 and UDP headers. The only UDP port accepted is hard coded to 1234, but any source port is accepted. Frames not matching this criteria are discarded and subsequent frames are processed.

The DIX frame contains:

- Destination MAC address = Default broadcast address (FF:FF:FF:FF:FF:FF)
- Source MAC address = Device MAC address from OTP
- Type = IPv4 (0x800)

The IPv4 header contains:

- Version = 4
- Header length = 0
- TOS = 0
- Len = Computed during operation
- ID = 0x001
- Flags + Fragment offset = 0
- TTL = 0x10
- Protocol = UDP (17)
- Header checksum = Computed during operation
- SRC IP = 0.0.0.0
- DEST IP = 255.255.255.255

The UDP header contains:

- Source port = BOOTP client (68)
- Destination port = BOOTP server (67)
- Length = Computed during operation
- Checksum = Computed during operation

The BOOTP payload contains:

- Opcode = Request (1)
- HW Type = Ethernet (1)

- HW Address Length = 6
- Hop Count = 0
- Transaction ID = 1
- Number of seconds = 0
- Client IP = 0.0.0.0
- Your IP = 0.0.0.0
- Server IP = 0.0.0.0
- Gateway IP = 0.0.0.0
- Client HW Address = Device MAC address, from OTP
- Server Hostname = NULL
- Filename = NULL
- Option 60, vendor ID string, from boot parameter table
- Option 61, client ID string, from boot parameter table

3.9 PCI Express (PCIe) Boot

The PCIe boot mode transfers a boot image via the PCIe interface into MSMC for execution. This image must be in Blob Format. The bootloader queries the device boot configuration pins for set up of the BARs, windows, and their sizes as desired to provide memory access to the host. Additional information about these configurations and other registers can be found in the device-specific data manual.

After configuration of the peripheral, the ARM core executes a WFI instruction that causes the ARM to suspend execution waiting for an interrupt. While the ARM is suspended, the PCIe host can write the boot image into MSMC. After the image is written, the PCIe host must write the execution starting address of the image to the host boot data address register.

MSI interrupts are enabled and routed to break out of the suspended state. Information on MSI interrupts can be found in the *Peripheral Component Interconnect Express (PCIe) for KeyStone Devices User Guide* in "Related Documentation from Texas Instruments" on page ø-viii. Each time an interrupt is triggered, the ARM core will poll the host boot data address register to check if the boot image is ready. If the host boot data address register indicates the boot image is ready, the bootloader will begin executing the boot image at the address specified in the register.

3.10 HyperLink Boot

The HyperLink boot mode transfers a boot image via HyperLink interface into MSMC for execution. This image must be in Blob Format. The bootloader queries the device boot configuration pins for set up of the data rate and port as desired to provide memory access to the host. Additional information about these configurations and other registers can be found in the device-specific data manual.

After configuration of the peripheral, the ARM core executes a WFI instruction that causes the ARM to suspend execution waiting for an interrupt. While the ARM is suspended, the HyperLink host can write the boot image into the MSMC. After the image is written, the HyperLink host must write the execution starting address of the image to the host boot data address register.

The HyperLink host must interrupt the ARM to break out of the suspended state. Each time an interrupt is triggered, the ARM core will poll the host boot data address register to check if the boot image is ready. If the host boot data address register indicates the boot image is ready, the bootloader will begin executing the boot image at the address specified in the register.

3.11 UART Boot

The UART boot mode transfers a boot image from an external host over a UART interface. The host must be capable of sending data via the X-modem protocol. While the ARM is the boot master, the image must be in Blob Format. The host must match the default boot serial configurations, which include:

- No parity
- One stop bit
- 115,200 baud rate
- 8-bit data length
- No flow control

If multiple device UART ports are available, the boot configuration pins for the desired port number must be set; this setting can be found in the device-specific data manual. After initialization is complete, the device sends pings to the host to indicate it is ready to receive the boot image. The pings consist of an ASCII capital *C* character. The host must begin an X-modem file transfer. The bootloader is available to receive data for 30 seconds after initialization. If 30 seconds elapse with no transfer, the boot will be considered failed and the next step will be taken in the boot process flow.

The bootloader stores data at the base of MSMC. Once the complete boot image is transferred and stored in the MSMC, the bootloader will begin execution from the base of MSMC.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265