C66x DSP Bootloader

User Guide

Literature Number: SPRUGY5
November 2010

Release History

Release	Date	Chapter/Topic	Description/Comments
1.0	November 2010	All	Initial Release

Contents

List of Tables

Preface

About This Manual

This document describes the features of the on-chip bootloader provided with C66x_Digital Signal Processors (DSP).

This document should be used in conjunction with the device-specific data manuals 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
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
Serial Peripheral Interface (SPI) for KeyStone Devices User Guide	SPRUGP2
Serial RapidIO (SRIO) for KeyStone Devices User Guide	SPRUGW1

Trademarks

Chapter 1

Introduction

This document describes the features of the on-chip bootloader provided with C66x Digital Signal Processors (DSP).

This document should be used in conjunction with the device-specific data manuals and user guides for peripherals used during the boot. This document supports only non-secure boot mode.

1.1 Bootloader Features

The bootloader is DSP code that transfers application code from slow non-volatile external memory into high-speed internal memory for execution or interacts with the host to transfer the application to the DSP after the DSP is taken out of reset. The bootloader is permanently stored in the internal ROM of the DSP starting at the base address of ROM 0x20B00000.

To accommodate different system requirements, the bootloader offers a variety of boot modes to transfer the application into the DSP memory. The various boot modes and their operations are listed below.

- EMIF16 boot (available only in TCI6608\C667x): The code is executed from the external asynchronous memory through a 16-bit interface.
- Serial Rapid IO (SRIO) boot: An external host loads the application through the SRIO peripheral either in messaging mode or in directIO mode.
- Ethernet boot: An external host loads the application through an Ethernet peripheral with the packet accelerator driven by the core reference clock or the SerDes reference clock during this boot process.
- PCIe boot: The external host loads the application into the on-chip memory through PCI Express. The code execution begins when the host indicates to the bootloader that the application has been loaded.
- Master I^2C boot: The application is loaded from an I^2C EEPROM one section at a time by using a boot table to determine the length and the starting address for each section.
- Slave I^2C boot: An external I^2C master sends the application to the DSP in the form of a boot table. This mode is typically used to support single I^2C EEPROM with multiple DSPs to minimize the boot time.
- SPI boot: The application is loaded from a serial flash memory device using the boot parameter table.
- HyperLink boot: This is a passive boot mode wherein the host takes the responsibility of configuring the memory and loading the code directly to boot the DSPs.

1.2 Terms and Abbreviations

Term	Definition
I²C	Inter-Integrated Circuit
MSMC	Multicore Shared Memory Controller
POR	Power on Reset
SPI	Serial Peripheral Interface
SRIO	Serial Rapid Input/Output

Chapter 2

Reset Types

2.1 Introduction

Boot is driven on a device reset by CorePac0. To determine the type of reset, the PLL controller register RESET_TYPE and status register RESET_STAT are queried. There are four types of reset supported in the KeyStone architecture:

- Power-on reset (POR)
- Chip 0 Reset (Hard Reset)
- Chip 1 Reset (Soft Reset)
- Local Reset

Bootloader initialization varies according to the type of reset.

2.2 Bootloader Initialization After POR

Power-on-Reset is a cold reset, which means that the part was not currently powered on. Therefore, all logic on the device must be reset to its default state. POR can be asserted by the PORz pin or the RESETFULLz pin. The PORz pin asserts the device when the system power supplies are ramped and the RESETFULLz pin asserts the device completely—including the reset-isolated logic—after the device is powered on. Most initializations, like local power/sleep control (LPSC) numbers, are done during POR. Bootstrapping is also done during the POR. After the POR is detected, the bootloader reads the DEVSTAT register, which is tied to the bootstrap pins, to identify the type of boot, then executes the boot procedure accordingly. The bootloader also initializes the interface of the booting device. The initialization procedures are listed below:

- The ROM code enables the reset isolation in all peripherals that support it. The power state of these peripherals is not changed.

  Table 2-1 lists the peripherals that have reset isolation and their corresponding LPSCs.

Table 2-1 Reset Isolation Configuration

Peripheral	TCI6608/C667x LPSC number	TCI661x/C6670 LPSC number
Smart Reflex	1	1
DDR3 EMIF	2	2
Embedded Trace	6	6
Ethernet SGMIIs and switch	8	8
SRIO	11	11
AIF	N/A	18

- The ROM code also ensures that the power and clock domains are enabled for any peripherals that are required for boot.
- The boot code configures the system PLL to set the device speed based on the three PLL selection bits in the DEVSTAT register. A post divider of two is always applied (postdiv value = 1).

  Table 2-2 lists PLL configurations that are achievable by different combinations of the PLL selection bits for different device speeds. Clkr is the predivider value (minus 1); clkf is the multiplier value (minus 1).

  The resulting multiplier is: (clkf + 1)/(2*clkr + 1)

  Based on the required clock setting, the boot PLL select is set in the DEVSTAT register through the bootstrap pin settings.

Table 2-2 PLL Clock Configuration

Boot Pll Select	Input Clock Freq (MHz)	Core = 800 MHz		Core = 1000MHz		Core = 1200MHz	
		Clkr	Clkf	Clkr	Clkf	Clkr	Clkf
0	50.00	0	31	0	39	0	47
1	66.67	0	23	0	29	0	35
2	80.00	0	19	9	24	0	29
3	100.00	0	15	0	19	0	23
4	156.25	24	255	4	63	24	383
5	250.00	4	31	0	7	4	47
6	312.50	24	127	4	31	24	191
7	122.88	47	624	28	471	31	624

- The ROM boot code also configures the PLLs for other peripherals that are used in the boot mode selected. For further PLL configuration, see the sections for specific boot modes.
- The ROM code also initializes a portion of the local L2 RAM for CorePac0. For the TCI661x/C6670, the reserved L2 memory ranges from 0x8F2E00 to 0x8FFFFF. For the TCI6608/C667x, the reserved L2 memory ranges from 0x872E00 to 0x87FFFF in all the cores. This area of the RAM is used to store the initial configuration of the boot process like the boot parameter table. Table 2-3 lists the configuration inputs stored in the local L2 for CorePac0 in C66xx devices.

Table 2-3 CorePac0 Memory Usage by the ROM Bootloader

TCI6608/C667x Address	TCI661x/C6670 Address	Size	Description
0x872E00	0x8F2E00	0x0040	ROM boot version string, (Unreserved)
0x872E40	0x8F2E40	0x0400	Boot Code Stack
0x873240	0x8F3240	0x0020	Boot Progress Register Stack (copies of boot program on mode change)
0x873260	0x8F3260	0x0100	Boot Internal Stats
0x873360	0x8F3360	0x0100	Boot variables (FAR data)
0x873460	0x8F3460	0x0100	DDR Configuration table
0x873560	0x8F3560	0x0080	RAM table functions
0x8735E0	0x8F35E0	0x0080	Boot Parameter table
0x873660	0x8F3660	0x3600	Clear text packet scratch
0x876C60	0x8F6C60	0x7f80	Ethernet/SRIO packet/message/descriptor memory
0x87EBE0	0x8FEBE0	0x80	Small Stack

Note that for EMIF16 boot in the TCI6608/C667x, no memory is reserved by the bootloader; memory usage depends entirely on the image stored in, and executed from, the NOR flash. Also, the 32k block of memory specified for the Ethernet/SRIO boot modes corresponds to two memory pages in the TCI6608/C667x and one memory page in the TCI661x/C6670. During the boot process, the bootloader executes an IDLE command on the secondary CorePacs and keep the secondary CorePacs waiting on an interrupt. The boot image to be executed in the different cores are downloaded using different boot method to the local L2s of the respective CorePacs. After the download is completed, the BOOT_MAGIC_ADDRESS of CorePac0 is written with the start address of the

image, which moves the program counter to that address and starts executing. The application that is executed in CorePac0 should write to the BOOT_MAGIC_ADDRESS of the secondary CorePacs with the start address of the application that has to be run on these CorePacs; it should also write into the IPC register to trigger an interrupt for that particular CorePac. This brings the secondary CorePacs from IDLE and starts executing the stored application from the start address specified in the BOOT_MAGIC ADDRESS. The BOOT_MAGIC ADDRESS for each core is the last address in the respective local L2. For TCI6608/C667x it is 0x807FFF and for TCI661x/C6670 it is 0x80FFFF.

- All L1D and L1P memory is configured by the boot code as cached memory. But L2 memory is configured as addressable memory.
- All the interrupts are disabled except host interrupts that are needed for the PCIe, SRIO (DirectIO), and HyperLink boot modes.
- The bootloader also has a DDR configuration table that, by default, is initialized to all zeros. During the bootloader process , the parameter table is polled after every boot table section is complete. The DDR3 is configured if the bootloader finds that the enable bitmap field is non-zero. This allows a single boot table to configure the DDR table, then load data to DDR. The DDR configuration boot parameter table is shown in Table 2-4.

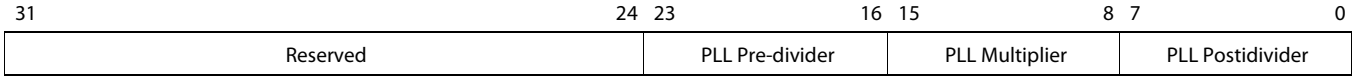
Table 2-4 DDR Configuration (Part 1 of 2)

Byte offset	Name	Description
0	Enable bitmap	One bit per configuration value. Bit 0 corresponds to the PLL config entry, bit 1 to the SDRAM config entry, etc. The corresponding value will only be set if the bit is set in this bitmap.
4	PLL config	See Figure 2-1
8	config	SDRAM Config Register
12	config 2	SDRAM Config 2 Register
16	Refresh ctl	SDRAM Refresh Control Register
20	Timing 1	SDRAM Timing 1 Register
24	Timing 2	SDRAM Timing 2 Register
28	Timing 3	SDRAM Timing 3 Register
32	Nvm timing	LPDDR2-NVM Timing Register
36	Pwr management	Power Management Control Register
40	IODFT_TLGC	IODFT Test Logic Global Control Register
44	Perf ctl cfg	Performance Counter Config Register
48	Perf ctl sel	Performance Counter Master Region Select Register
52	Read idle ctl	Read Idle Control Register
56	Irq enable	System VBUSM Interrupt Enable Set Register
60	Zq config	SDRAM Output Impedance Calibration Config Register
64	Temp alert cfg	Temperature Alert Config Register
68	Phy ctrl 1	DDR PHY Control 1 Register
72	Phy ctrl 2	DDR PHY Control 2 Register
76	Pri cos map	Priority to Class of Service Mapping Register
80	Mst id cos map 1	Master ID to Class of Service 1 Mapping Register
84	Mst id cos map 2	Master ID to Class of Service 2 Mapping Register
88	Ecc ctrl	ECC Control Register
92	Ecc addr rng 1	ECC Address Range 1 Register

Table 2-4 DDR Configuration (Part 2 of 2)

Byte offset	Name	Description
96	Ecc addr rng 2	ECC Address Range 2 Register
100	Rw/exc thresh	Read Write Execution Threshold Register
End of Table 2-4		

Figure 2-1 DDR PLL Configuration



The boot mode pin configuration in the DEVSTAT register is used to decode the boot mode. Table 2-5 shows the boot mode pin layout. The SR ID is mentioned only for legacy pupose and it is removed in bootloader designed for Keystone Architecture devices.

Table 2-5 Boot Mode Pins

12	11	10	9	8	7	6	5	4	3	2	1	0
PLL Mult I2C/SPI Ext Dev. Cfg			Device Configuration (with bits 5-3 used for SR ID)							Boot Device		

Table 2-6 lists the boot device values.

Table 2-6 Device Configuration Pin Values

Boot Mode Pins: Boot Device Values	
Value	**Boot Device**
0	Sleep / EMIF16 (TCI6608/C667x only)
1	Serial RapidIO
2	Ethernet (SGMII) (PA driven from core clk)
3	Ethernet (SGMII) (PA driver from PA clk)
4	PCI
5	I2C
6	SPI
7	HyperLink

Based on the different boot modes, the device configuration and the PLL bit fields vary. The initial configurations above are guided by the boot parameter table stored in the fixed address in local L2 of CorePac0. The boot parameter table begins with five parameters irrespective of the boot mode selected. Table 2-7 lists these parameters.

Table 2-7 Boot Parameter Common Values (Part 1 of 2)

Byte Offset	Name	Description
0	Length	The length of the table, including the length field, in bytes.
2	Checksum	The 16 bits ones complement of the ones complement of the entire table. A value of 0 will disable checksum verification of the table by the boot ROM.
4	Boot Mode	0-7: Specifies the boot device.
6	Port Num	Identifies the device port number to boot from, if applicable

Table 2-7 Boot Parameter Common Values (Part 2 of 2)

Byte Offset	Name	Description
8	SW PLL, MSW	PLL configuration, MSW
10	SW PLL, LSW	PLL configuration, LSW
End of Table 2-7		

Table 2-8 shows the format of the PLL configuration field and the bit values.

Table 2-8 PLL Configuration

21	30	29	16	15	8	7	0
PLL Config Ctl 00 - PLL not configured 01 - PLL configured only if it is already in bypass or disable mode 10 - PLL is configured 11- PLL is disabled and put into bypass mode		Pll Multiplier (can be between 0-255		Pll Pre-Divider (can be between 0-255)		Pll Post-Divider (can be between 0-255)	

For information about the different types of tables supported by the bootloader, see Section 3.5.7. The BOOTCOMPLETE register, which controls the BOOTCOMPLETE pin status, is used to indicate the completion of the ROM boot process. Based on the boot mode selected, the rest of the boot parameter table varies. The other CorePacs are all in IDLE and it is up to the boot image to take the CorePac out of IDLE.

2.3 Bootloader Initialization After Chip Reset 0 (Hard Reset)

This reset type is similar to POR, but assumes that the device was already powered up and has already gone through a power-on -reset sequence. This reset was meant to reset all logic on the chip with the exception of any test logic, emulation logic, or any other IP that needs reset isolation. Hard Reset is also commonly referred to as warm reset or device reset. Hard reset can be applied either externally by assertion of active-low RESETz pin or internally by writing to a register “RSCTRL” in PLLCTL or from the Watchdog TimersReset initiated through reset_req_pi_n[] port of PLLCTL are blockable by setting “RSCFG” register in PLLCTL. All other reset sources in this category are nonblockable. Hard reset can be applied to the device anytime during operation to reset all logic except for test, emulation, and IPs that need reset isolation. While in this reset, only part of the device is initialized. Some of the longer initialization activities such as EFUSE scan and PSC initialization do not happen. Bootstrapping is also not done for this type of reset. All tristate outputs are also placed in the high-impedance state except for the reset isolated modules; all other outputs are driven to their inactive level. By default, the PLL goes to bypass mode and clock settings are reset to default values. To avoid this, Reset Isolation Register (RSISO) in the PLLCTL must be enabled by Boot ROM software. This retains the state of the PLL, the dividers, or both in the PLLCTL. For the Secondary PLL, such as DDR PLL and PA_SS PLL, this is different as the PLL is always enabled and clocking. Hard reset does not reset any test or emulation logic. It is intended that an emulator session will continue unaffected. It is also a requirement that the Reset isolation must be supported for SRIO, SGMII, TETB, DDR3_EMIF, and Smart Reflex for TCI661x/C6670 and TCI6608/C667x and AIF2 for TCI661x/C6670.

The bootstrapping is not done. The eFuse scan autoload sequence is not initiated. PORz / RESETFULLz should remain deasserted during this time. A Hard reset takes the PSC to its default state. From the PSC point of view, hard reset causes the PSM state machine to reset, so all the power domains except the “always on and reset isolated’ domains will be turned off before it gets turned on again.

The PLLCTL module accepts the reset and steps through the reset sequence as follows:

1. The RESETz pin is pulled active low or internal requestor asserts reset. The reset signals flow to the modules reset, resetting anything that uses reset asynchronously, and sends a tri-state signal to most I/O pads to prevent off-chip contention. IOs associated with reset isolated modules would not be tri-stated. For a list of the IOs tri-stated, see the pin list.
2. Any logic that was using reset synchronously is now reset.
3. The RESETz pin is released or other internal requestor deasserts reset (driven inactive high).
4. Reset is stretched for an additional 16 slow-sysclk cycles.
5. The chip synchronous reset is released and, after waiting eight Main PLL clock cycles, all the system clocks that were allowed to pause are started again. The clock pauses to allow the reset to propagate across the chip and make it easier to meet timing requirements.
6. When the clocks restart after the pause, the device is out of reset and starts executing.

2.4 Bootloader Initialization After Chip Reset 1 (Soft Reset)

This resets only part of the device, a subset of the hard reset. It is usually called a Soft Reset and can work several different ways. In one use model, some modules are reset completely while other modules are not. Another use model is modules that may have a separate reset input in which the logic is reset, but the previous configuration or any memory states may be saved. It can also be a combination of the two.

Soft Reset is an internal reset. This can also be an external reset coming from RESETz pin. The Reset config register (RSCFG) of the PLLCTL needs to be set to configure various initiators for soft reset. Reset initiated through reset_req_pi_n[] port of PLLCTL are blockable by setting the RSCFG register in PLLCTL. All other reset sources in this category are nonblockable. Bootstrapping is not done for this type of reset. Soft reset behaves like hard reset except that PCIe MMRs, EMIF16 MMRs, DDR3 EMIF MMRs, and External memory contents are retained. For DDR3 SDRAM memory contents to be retained, the software must first ensure that the SDRAM is put in self-refresh mode. By default, the PLL multiplier and PLL controller settings return to default. The Reset Isolation Register (RSISO) in the PLL controller must be set in Boot ROM code to block this behavior. This setting is mandatory for reset isolation to work.

In Chip Reset1, it does not affect the clock logic or the power control logic of the peripherals. PSC modules do not get reset on a soft reset thereby it does not cause a PSM-state-machine reset. Also, it does not reset test or emulation circuitry. It is also required that reset isolation must be supported for SRIO, SGMII, TETB, and the Smart Reflex subsystem for both TCI661x/C6670 and TCI6608/C667x and AIF2 for TCI661x/C6670 and DDR3_EMIF for TCI6608/C667x. Bootstrapping is not done. The eFuse scan autoload sequence is also not initiated. Soft reset behaves like hard reset except that PCIe MMRs and External Memory Contents for TCI661x/C6670 and TCI6608/C667x and EMIF16 MMRs, DDR3 EMIF MMRs are retained. The Reset config register (RSCFG) of the PLLCTL should be set to configure various initiators for soft reset.

The PLLCTL module accepts the reset and steps through the reset sequence as follows:

1. The internal requestor asserts reset or RESETz pin is pulled active low. The reset controller latches this, then activates the signals necessary to reset everything except the modules that reset only on hard reset, such as the PCIe for both TCI661x/C6670 and TCI6608/C667x, DDR3 EMIF, and EMIF16 modules for TCI6608/C667x. The reset signals flow to the modules reset, resetting anything that uses reset asynchronously, and send a tristate signal to most of the I/O pads to prevent off-chip contention. IOs associated with reset-isolated modules would not be tri-stated. For a list of the IOs tri-stated, see the pin list.
2. Any logic that was using reset synchronously is now reset.
3. RESETz pin is released (driven inactive high) or other internal requestor deasserts reset.
4. Reset is stretched for an additional 16 slow-sysclk cycles.
5. The chip synchronous reset is released, and after waiting eight Main PLL clock cycles, all the system clocks that were allowed to pause are started again. This clock pausing is to allow reset to propagate across the chip to make it easier to meet timing requirements.
6. When clocks restart after the pause, the device is out of reset, and starts executing.

2.5 The Effect of Hibernation on Bootloader Initialization During Chip Reset0 and Chip Reset1

There are two hibernation modes: Hibernation 1 and Hibernation 2.

Hibernation1 requires chip reset to exit; the chip reset is triggered through the pin. The critical status and information is stored in the MSMC SRAM. A chip-level register is added to control the reset MSMC parity RAM. Before entering Hibernation1 mode, the software must correctly configure the chip-level register (chip_misc_msmc_block_parity_rst) so the parity SRAM will not reset when the device applies chip_0_rst to exit Hibernation1. For TCI661x/C6670, the response time is less than 100 ms. For TCI6608/C667x, the response time is less than 100 ms.

Hibernation2 requires chip_0_rst to be triggered by the chip reset pin, MSMC SRAM turned off. The critical data is stored in DDR in Hibernation2 mode. When the device exits from Hibernation2 mode through chip reset, the MSMC is reset. So the MSMC configuration is set to the default value after reset, so only the lower 2 GB of 4 GB space is visible to the DDR. Therefore, the branching address for exiting Hibernation2 mode should be set at the lower 2 GB boundary) between 0x8000_0000 to 0xFFFF_FFFF. For TCI661x/C6670, the response time is less than a few seconds. For TCI6608/C667x, the response time is less than 1~2 seconds.

Before the TCI661x/C6670 enters hibernation mode, SRIO can enable reset isolation (default) or disable reset isolation. When SRIO has reset isolation enabled before entering hibernation, the LPSC for SRIO should be 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; 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 PCIe is used as EP mode, the software must put PCIe in the L1 powerdown mode before it enters the hibernation mode and PCIe power domain should be kept on during the 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.

Chip_0_rst triggered by the reset pin is the only way to exit hibernation; therefore, to preserve the DDR content while exiting hibernation, DDR should not be hard reset by chip_0_rst. To support this, reset-isolation is added to DDR. The boot code enables the DDR reset isolation by default and the software has the option to turn off the reset isolation feature if it is not needed.

During Hibernation1, only MSMC SRAM content is preserved; the MSMC MMR is not preserved. There is a requirement for the ROM software to recover from hibernation.

Because the MSMC register is set to default after chip reset exits Hibernation1 mode, the software must save the MSMC MMR values to a known memory location inside the MSMC SRAM. When the device exits Hibernation1 mode, the software can access the memory contents under the default MSMC setting. Therefore, the MSMC register values can be restored. Before entering the hibernation state, an indication of hibernation and the specific mode (1 or 2) must be stored in the boot-cfg MMR PWRSTATECTL.

Because it is a requirement to maintain the DDR3 RAM contents when the device wakes up from both hibernation modes, 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.

MSMC on this device does not support the PSC Disable interface. Therefore, MSMC could not detect the status of DDR3 EMIF when DDR3 EMIF is disabled or enabled by PSC. Accesses to disabled DDR3 EMIF would hang the device. The solution is to tie off the PSC control to DDR_EMIF to enable it on the chip level (always on). Accesses made to this empty space will not hang when there are no external DDR memories attached to the controller. The tie-off of the PSC control plus the reset isolation of DDR EMIF make it impossible to reset EMIF4F independently of the rest of the chip. To reset EMIF4F independently—only for debugging purposes—a bit called DDR3_PSC_LOCK_n is created inside the chip-level MMR CHIP_MISC_CTL. It controls the DDR3 EMIF PSC to support a mode reset to the DDR3 EMIF. When set to 1, it allows the DDR3 EMIF to be reset independently of the rest of the chip.

After Chip Reset0 and Chip Reset1, the bootloader performs a standard boot in CorePac0 if hibernation is not enabled; the other CorePacs will be in the IDLE state and wake up, then branch to the boot magic address.

If Hibernation1 is enabled, CorePac0 will disable DDR self-refresh. Branch-to-address specified in PWRSTATECTL (Do not clear standby and hibernation fields) and the other CorePacs will be in the IDLE state and after wake-up verify that PWRSTATECTL's standby and hibernation fields have become 0, then branch to the boot magic address.

If Hibernation2 is enabled, CorePac0 will also disable DDR self-refresh and reset MSMC parity. Branch to address specified in PWRSTATECTL (Do not clear standby and hibernation fields) and the other CorePacs will be in the IDLE state and after wake up will verify that PWRSTATECTL's standby and hibernation fields have become 0, then branch to the boot magic address.

Chapter 3

Boot Modes

3.1 EMIF16 Boot

3.1.1 Basic Boot Operation

EMIF16 boot mode is available to boot from NOR flash in the TCI6608/C667x device only. In this mode, the ROM code configures the EMIF16 interface and sets the boot complete, then branches to the EMIF CS2 data memory at 0x70000000. No return is expected. No memory is reserved by the bootloader. The device configuration for EMIF16 boot mode is shown in Table 3-1.

Table 3-1 EMIF16 Boot Mode Device Configurations

6	5	4	3	2	1	0
Reserved		Wait Enable 0 - Wait disabled 1- Wait enabled	submode 00 - Sleep boot 01 - EMIF16 boot 10 - 11 - Reserved		SR Index	

3.2 SRIO Bootloader Operation

3.2.1 Basic Boot Operation

The SRIO boot operates in two modes: the Messaging mode and the DirectIO mode. Both the message mode and DirectIO mode are enabled by default, but mailbox can be disabled using the I²C boot parameter table. In the Messaging mode, the bootloader receives the boot image through the SRIO messaging interface in the form of the boot table; transmission is terminated after the end-of-boot packet is received. This signals the completion of the boot process and also wakes up the DSP to execute the boot image from the entry point specified in the boot table. For the DirectIO mode, the DSP is in IDLE state till the host sends the entry point address to the boot magic address. The bootloader keeps polling the boot magic address and when the value is non-zero, the bootloader wakes up the DSP, which then executes the boot image from the address in the boot magic address. Because there is no provision in messaging mode to handle the messaging output, the host application should provide a delay between message transmissions to avoid an out-of-order message scenario.

The SRIO boot mode parameters and the bit field descriptions are listed in Table 3-2.

Table 3-2 SRIO Boot Mode Parameter Table

Byte Offset	Name	Description
12	Options	Bit 0 Tx enable 0 SRIO Transmit disable 1 SRIO Transmit Enable Bit 1 Mailbox Enable 0 Mailbox mode disabled. (SRIO boot is in DirectIO mode). 1 Mailbox mode enabled. (SRIO boot is in Messaging mode). Bit 2 Bypass Configuration 0 Configure the SRIO 1 Bypass SRIO configuration Bit 3-15 Reserved
14	Lane Setup	0b0000 - SRIO configured as four 1x ports 0b0001 - SRIO configured as 3 ports (2x, 1x, 1x) 0b0010 - SRIO configured as 3 ports (1x, 1x, 2x) 0b0011 - SRIO configured as 2 ports (2x, 2x) 0b0100 - SRIO configured as 1 4x port 0b 0101 - 0bffff - Reserved
16	Config Index	Specifies the template used for RapidIO configuration. Must be 0 for KeyStone Architecture
18	Node ID	The node ID value to set for this device
20	SERDES ref clk	The SERDES reference clock frequency, in 1/100 MHZ
22	Link Rate	Link rate, MHz
24	PF Low	Packet forward address range, low value
End of Table 3-2		

Apart from some of the parameters in the table above that are latched from the bootstrap settings, all the other parameters are populated with default values by the bootloader. These parameters can be modified only through the I²C boot parameter table through the I²C master boot and reenter the bootloader in SRIO mode. For example, although there are five possible lane setups, only two lane setups are possible from the bootstrap pin configuration for primary SRIO boot and other lane setups can

be configured only through the I²C boot parameter table. The lane setups and the reference clock setups can be derived from the SRIO device configuration field of the DEVSTAT register. The SRIO device configuration field is described in the Table 3-3 and Table 3-4.

Table 3-3 SRIO Boot Mode Device C

6	5	4	3	2	1	0
Lane Setup	Data Rate		Ref Clock		SR ID	

Table 3-4 SRIO Boot Mode Device Configuration Field Descriptions

Bit Field	Value	Description
SR ID	0-3	Smart Reflex ID
Ref Clock	0	Reference Clock = 156.25 MHz
	1	Reference Clock = 250 MHz
	2	Reference Clock = 312.5 MHz
Data Rate	0	Data Rate = 1.25 Gbps
	1	Data Rate = 2.5 Gbps
	2	Data Rate = 3.125 Gbps
	3	Data Rate = 5.0 Gbps
Lane Setup	0	Port Configured as 4 ports each 1 lane wide (4x1ports)
	1	Port Configured as 2 ports 2 lanes wide (2x2 ports)
End of Table 3-4		

Before configuring the SRIO, SerDes should be configured. The bootloader initializes the PLL and the receive and the transmit channel registers. After configuring SerDes and SRIO, for message mode the boot code then initializes the QMSS to configure transmit and receive queues. The boot code allocates a 32k block of local L2 memory on CorePac0 for packet reception. This is divided into seven buffers of 4096 bytes each with the remainder for the packet descriptors. Because the accumulation and interrupt functions are not used during the message mode, it should not be necessary to download and run the QM PDSPs. After the boot table processing is complet, e the boot code resets the queue manager, sets the boot complete bit and branches to the address specified by the boot table; the interrupt maps are restored to their default values.

The boot messages are simply a segmented boot table pre-pended with the header shown in Table 3-5. The application that is loading the message from the host device should be responsible to package the message packets with the header before sending to the device.

Table 3-5 SRIO Message Mode Boot Header

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
block Size = 6															
Block Checksum one's Complement															

3.3 Ethernet Bootloader Operation

3.3.1 Basic Boot Operation

The bootloader configures SerDes, SGMII, and the switch plus the PASS and the Multicore Navigator (Packet DMA and the QMSS) to prepare to receive the boot table from the host connected to the device through the Ethernet interface. These initial configurations are derived by querying the boot mode pins of the DEVSTAT register and the boot parameter table for the Ethernet boot. Based on the boot mode selected, the PA subsystem can be driven either by the main PLL reference clock or by the SerDes reference clock (see Table 3-6 for the boot mode selection). If the main PLL reference clock is used, the PA multiplier configuration can be derived from the three PLL selection bits in the DEVSTAT and the input clock as shown in the table.

Table 3-6 PA Pll Clock Configuration

Boot PLL Select [2:0]	Input Clock Freq (MHz)	PA = 350 MHz	
		Clkr	Clkf
0	50.00	0	41
1	66.67	1	62
2	80.00	3	104
3	100.00	0	20
4	156.25	24	335
5	250.00	4	41
6	312.50	24	167
7	122.88	11	204
End of Table 3-6			

If the SerDes reference clock is used, the PA PLL clock configuration is restricted to the option 4, 5, and 6 in Table 3-6. If the SerDes clock is used the bootloader first configures the SerDes PLL and the transmit and receive channels. The default values for these registers are configured in the boot parameter table for the Ethernet boot as shown in Table 3-9. The external connection from the DSP to the host is selected through the device configuration bits in the boot mode pins of the DEVSTAT register. The device configuration bit fields and the values for these fields are described in Table 3-7 and Table 3-8.

Table 3-7 Ethernet(SGMII) Boot Mode Device Configuration

6	5	4	3	2	1	0
SERDES Clock Mult		Ext Connection		DEV ID	DEV ID (SR ID)	

Table 3-8 Ethernet Boot Mode Device Configuration Descriptions (Part 1 of 2)

Bit field	Value	Description
Ext connection	0	Mac to Mac connection, master with auto negotiation
	1	Mac to Mac connection, slave, and Mac to Phy
	2	Mac to Mac, forced link
	3	Mac to fiber connection
Device ID	0-7	This value is used in the device ID field of the Ethernet ready frame. Bits 1:0 are use for the SR ID.

Table 3-8 Ethernet Boot Mode Device Configuration Descriptions (Part 2 of 2)

Bit field	Value	Description
SERDES Clock Mult The output frequency of the PLL must be 1.25 GBs.	0	x8 for input clock of 156.25 MHz
	1	x5 for input clock of 250 MHz
	2	x4 for input clock of 312.5 MHz
	3	Reserved
End of Table 3-8		

Table 3-9 Ethernet Boot Parameter Table (Part 1 of 2)

Byte Offset	Name	Description
12	Options	Bit 4 Skip TX 0 Send Ethernet Ready Frame every 3 seconds 1 Don't send Ethernet Ready Frame Bits 06 - 05 Initialize Config 00 Switch, SERDES, SGMII and PASS are configured 01 Only SGMII and PASS are configured 10 Reserved 11 None of the Ethernet system is configured.
14	MAC High	The 16 MSBs of the MAC address to receive during boot
16	MAC Med	The 16 middle bits of the MAC address to receive during boot
18	MAC Low	The 16 LSBs of the MAC address to receive during boot
20	Multi MAC High	The 16 MSBs of the multi-cast MAC address to receive during boot
22	Multi MAC Med	The 16 middle bits of the multi-cast MAC address to receive during boot
24	Multi MAC Low	The 16 LSBs of the multi-cast MAC address to receive during boot
26	Source Port	The source UDP port to accept boot packets from. A value of 0 will accept packets from any UDP port
28	Dest Port	The destination port to accept boot packets on.
30	Device ID 12	The 1st two bytes of the device ID. This is typically a string value, and is sent in the Ethernet ready frame
32	Device ID 34	The 2nd two bytes of the device ID.
34	Dest MAC High	The 16 MSBs of the MAC destination address used for the Ethernet ready frame. Default is broadcast.
36	Dest MAC Med	The 16 middle bits of the MAC destination address
38	Dest MAC Low	The 16 LSBs of the MAC destination address
40	Sgmii Config	Bits 0-3 are the config index, bit 4 set if direct config used, bit 5 set if no configuration done
42	Sgmii Control	The SGMII control register value
44	Sgmii Adv Ability	The SGMII ADV Ability register value
46	Sgmii TX Cfg High	The 16 MSBs of the sgmii Tx config register
48	Sgmii TX Cfg Low	The 16 LSBs of the sgmii Tx config register
50	Sgmii RX Cfg High	The 16 MSBs of the sgmii Rx config register
52	Sgmii RX Cfg Low	The 16 LSBs of the sgmii Rx config register
54	Sgmii Aux Cfg High	The 16 MSBs of the sgmii Aux config register
56	Sgmii Aux Cfg Low	The 16 LSBs of the sgmii Aux config register

Table 3-9 Ethernet Boot Parameter Table (Part 2 of 2)

Byte Offset	Name	Description
58	SERDES PLL Cfg MSW	The SGMII_SERDES_CFG_PLL register, MSW. Default value (all 32 bits) 0x00000229. Includes the PLL multiplier in bits 7:1
60	SERDES PLL CFG LSW	The SGMII_SERDES_CFG_PLL register, LSW.
62	SERDES RXCfg MSW	The SGMII_SERDES_CFGRX register, MSW. Default value (all 32 bits) 0x00000700. This value is used in both SERDES lanes.
64	SERDES RX CFG LSW	The SGMII_SERDES_CFGRX register, LSW.
66	SERDES TXCfg MSW	The SGMII_SERDES_CFGTX register, MSW. Default value (all 32 bits) 0x00000100. This value is used in both SERDES lanes.
68	SERDES TX CFG LSW	The SGMII_SERDES_CFGTX register, LSW.
70	PKT PLL Cfg MSW	The packet subsystem PLL configuration, MSW
72	PKT PLL CFG LSW	The packet subsystem PLL configuration, LSW
End of Table 3-9		

After initializing the clock according to mode, the Ethernet boot code enables the SGMII in full-duplex gigabit rate. The SGMII is also configured to receive broadcast packets as specified in the boot parameter table (Table 3-9). The external PHY is also initialized to come up in standard learning mode. The Ethernet boot code then initializes the QMSS to configure transmit and receive queues. The boot code allocates a 32k block of local L2 memory on CorePac0 for packet reception. This is divided into 20 buffers each of 1536 bytes, with the remainder for the packet descriptors.

Custom firmware is loaded to operate the PA subsystem, which enables the PASS to direct all the received traffic to the CPDMA using the above-configured flow. The boot code also configures a transmit channel to optionally send an Ethernet-ready packet which is sent at the end of the configuration described above. When it is enabled to send, the Ethernet-ready packet is sent every three seconds—the time interval between the packets varies with the input clock—until the first valid boot packet is received by polling the receive queue. After it is detected, the received packet is verified, then sent to the boot table processing functions. When boot-table processing is completed, the boot code resets the PA and the queue manager before branching to the address specified in the boot table. The interrupt maps are also restored to their default values and start executing the application loaded.

3.3.2 Ethernet-Ready Announcement Format

The Ethernet-ready announcement frame is made in the form of a BOOTP request so it can use a standard format. No response is processed for this message and it is constructed so that most if not all BOOTP and DHCP servers will discard it. The announcement frame is sent every three seconds (the time interval between the packets varies with the clock input); no retransmission is done.

The frame uses the DIX MAC Header format. The MAC header contains:

- Destination MAC address = H-MAC addr (from boot params, normally FF:FF:FF:FF:FF:FF)
- Source MAC address == this devices MAC addr (from boot params)
- Type = IPV4 (0x800)

The IPV4 header contains:

- Version = 4
- Header length = 0
- TOS = 0
- Len = 328 (300 BOOTP + 8 UDP + 20 IP)
- ID = 0x0001
- Flags + Fragment offset = 0
- TTL = 0x10
- Protocol = UDP (17)
- Header checksum = 0xA9A5
- SRC IP = 0.0.0.0
- DEST IP = 0.0.0.0

The UDP header contains:

- Source port = BOOTP client (68 decimal)
- Destination port = BOOTP server (67 decimal).
- Length = 308 (300 BOOTP + 8 UDP)
- Checksum = 0 (not calculated)

The BOOTP Payload contains:

- Opcode = Request (1)
- HW Type = Ethernet (1)
- HW Addr Len = 6
- Hop Count = 0
- Transaction ID = 0x12345678
- Number of seconds = 1
- 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 Addr = this device's MAC address
- Server hostname = "ti-boot-table-svr"
- Filename = 'ti-boot-table-XXXX" (where XXXX is the 4 character device ID from boot params)
- Vendor info = all zeros

3.3.3 Ethernet Boot Packet Format

The boot table format is encapsulated in Ethernet frames with IPV4 and UDP headers. The following paragraphs describe the Ethernet frames that are accepted. Frames not matching the specified criteria are silently discarded and subsequent frames processed.

Frames using both DIX and 802.3 MAC header formats are accepted as are frames with and without VLAN tags. Any source MAC address is acceptable. A destination MAC address of this device (as specified in boot params) or the M-MAC specified in the boot parameters are accepted. VLAN fields (other than type/len) are ignored. If 802.3 format MAC format is used, the SNAP/LLC header will be verified and skipped. The type field will select IPV4 type (0x0800).

The IPV4 header validates the Version (4), flag and fragment fields, and protocol (UDP) field. The header length field is parsed to skip header option words. Any source and destination IP addresses are accepted.

The UDP header validates that the source and destination port numbers match those specified in the boot parameters. If the boot parameter source port field is 0, any source port will be accepted. The UDP header length is sanity-tested against the adjusted frame length. If the UDP length is too long for the frame or is not a multiple of two, the frame is discarded. The UDP checksum is verified and the frame with incorrect UDP checksum is discarded if the UDP checksum field is non-zero.

The following checks are performed on the boot table frame header. The magic number field and opcode fields are compared to the expected values. The sequence number field is compared to the expected value. The expected value for sequence number is 0 for the first frame processed and increments by one for each processed frame.

The Boot table frame payload (which is a multiple of four bytes in length) is processed by the boot-table processing function.

Table 3-10 Ether Boot Packet Format

Ethernet Header, one of the following types:
DIX Ethernet (DMAC, SMAC, type: 14 bytes) 802.3 w/ SNAP/LLC (DMAC, SMAC, len, LLC, SNAP: : 22 bytes) DIX Ethernet w/ VLAN (18 bytes) 802.3 w/ VLAN and SNAP/LLC (26 bytes)
IPV4 (20 to 84 bytes) UDP (8 bytes)
Boot Table Frame Header (4 bytes) Boot Table Frame Payload (min 4 bytes, max limited by max Ethernet frame - previous headers)

Table 3-11 Boot Table Frame Header

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Boot Magic Number = 0x544b															
Opcode = 0x01								Sequence number							

3.4 PCI Express (PCIe) Bootloader Operation

3.4.1 Basic Boot Operation

In the PCIe boot mode, the host configures memory and loads all the sections directly to the memory. The bootloader configures the base address registers, the number of windows, and their size. The PCIe power-up is configured through the external pin PCIESSEN. PCIe boot code can configure the PCIe registers either by getting the values from the I²C or the default values from the boot parameter table (Table 3-12).

Table 3-12 PCIe Boot Parameter Table

Byte Offset	Name	Description
12	Config	PCI configuration options
14	Address Width	PCI address width, can be 32 or 64
16	Serdes Frequency	Serdes frequency, in Mbps. Can be 2500 or 5000
18	Reference clock	Reference clock frequency, in units of 10kHz. Value values are 10000 (100MHz), 12500 (125MHz), 15625 (156.25Mhz), 25000 (250MHz) and 31250 (312.5 MHz). A value of 0 means that value is already in the SerDes cfg parameters and will not be computed by the boot ROM.
20	Window 1 Size	Window 1 size. The size is 32 MB * 2n, where n is the parameter. Valid are 0-7.
22	Window 2 Size	Window 2 size.
24	Window 3 Size	Window 3 size. Valid only if address width is 32.
26	Window 4 Size	Window 4 Size. Valid only if the address width is 32.
28	Window 5 Size	Window 5 Size. Valid only if the address width is 32.
30	Serdes cfg msw	PCIe serdes config word, MSW
32	Serdes cfg lsw	PCIe serdes config word, LSW
34	Serdes lane 0 cfg msw	Serdes lane config word, msw lane 0
36	Serdes lane 0 cfg lsw	Serdes lane config word, lsw, lane 0
38	Serdes lane 1 cfg msw	Serdes lane config word, msw, lane 1
40	Serdes lane 1 cfg lsw	Serdes lane config word, lsw, lane 1
End of Table 3-12		

If the PCIe boot is the primary boot, the BAR size configuration is driven by the BAR config fields listed in Table 3-13 and Table 3-14.

Table 3-13 PCIe Boot Mode Device Configuration

6	5	4	3	2	1	0
Reserved	BAR Config				SR ID	

Table 3-14 PCIe Boot Mode Device Configuration Bits Description

Bit Field	Value	Description
SR ID	0-3	Smart Reflex ID
Bar Config	0-0xf	See Table 1-14

Table 3-15 PCIe Window Sizes

		32 bit Address Translation					64 bit Address Translation	
BAR cfg	**BAR0**	**BAR1**	**BAR2**	**BAR3**	**BAR4**	**BAR5**	**BAR1/2**	**BAR3/4**
0b0000	PCIe MMRs	32	32	32	32	Clone of BAR4		
0b0001		16	16	32	64			
0b0010		16	32	32	64			
0b0011		32	32	32	64			
0b0100		16	16	64	64			
0b0101		16	32	64	64			
0b0110		32	32	64	64			
0b0111		32	32	64	128			
0b1000		64	64	128	256			
0b1001		4	128	128	128			
0b1010		4	128	128	256			
0b1011		4	128	256	256			
0b1100							256	256
0b1101							512	512
0b1110							1024	1024
0b1111							2048	2048

End of Table 3-15

If the I²C EEPROM is used to configure the PCI registers, the BAR configurations are driven by the parameter table downloaded through I²C. On the boot reentry, the PCIe boot code is executed with these BAR configurations. In this mode of operation, the I²C peripheral is also configured by the bootloader.

The bootloader code also configures the interrupt subsystem by configuring the chip-level interrupt controller, then executes the IDLE instruction. After the host transfers the boot table directly to the memory location and if the BOOT_ADDRESS register is non-zero, the ROM code wakes up the DSP and branches to the start memory address to which the boot table is copied by the host after restoring the default interrupt configuration. If the BOOT_ADDRESS register is still zero the ROM code keeps the DSP in idle till the value is non-zero.

3.5 I2C Bootloader Operations

3.5.1 I2C Boot Basic Operations

I2C is the only peripheral that is configured during I2C boot. Because the data is transferred using I2C from EEPROM, the interrupt subsystem and the EDMA are not enabled. A seven-bit address and eight-bit data modes are the only supported transfer configuration. I2C can operate either in master mode or slave mode.

3.5.2 Master Mode—Boot Parameter Table

This is the default I2C mode. The BOOT-ADDRESS register is cleared and the data is read from the I2C in blocks of data starting at address (0x80 * parameter index). The parameter index can be read from the I2C master mode device configuration bit fields shown in the Table 3-14 and Table 3-14.

Table 3-16 I2C Master Mode Device Configuration

9	8	7	6	5	4	3	2	1	0
Reserved	Speed	Address	Mode(0)	Parameter Index					
								SR index	

Table 3-17 I2C Master Mode Field Description

Bit Field	Value	Description
Mode	0	Master Mode
	1	Passive Mode
Address	0	Boot From I2C EEPROM at I2C bus address 0x50
	1	Boot From I2C EEPROM at I2C bus address 0x51
Speed	0	I2C data rate set to approximately 20 kHz
	1	I2C fast mode. Data rate set to approximately 400 kHz (will not exceed)
Parameter Index	0-63	Identifies the index of the configuration table initially read from the I2C EEPROM

This block contains a boot parameter table for configuring the DSP to boot through any of the boot modes like Ethernet, SPI, etc. After reading the table and verifying the checksum, the boot code is reinitiated and reenters the boot code based on the boot mode.

3.5.3 Master Boot—Boot Table Mode

In this mode, the data is read from the EEPROM through the I2C and the four-byte block header is stripped and the remaining data is passed to the boot table processing function. If the BOOT_ADDRESS register is ever found to be non-zero, the boot code terminates, sets boot complete, and branches to that address. Rereads are attempted for blocks with checksum errors. Rereads are attempted for I2C errors detected during reads/writes.

3.5.4 Master Boot—Config Table Mode

In this mode, the data read from the I²C contains configuration tables. Each element in the table has three 32-bit fields, as shown in Table 3-18.

Table 3-18 Config Table Layout

Entry 0	Address
	Set Mask
	Clear Mask
Entry 1	Address
	Set Mask
	Clear Mask
…	
Entry N, Table Termination	Address = 0
	Set Mask = 0
	Clear Mask = 0

This mode is typically used to poke registers needed before boot can be run or to execute functions from a previously-loaded boot. Each entry in the table falls into one of three types described below.

3.5.4.1 Standard Entry

A standard entry has address ≠ 0, and set mask or clear mask ≠ 0. The ROM code reads the 32-bit value at address, modifies the value as shown in Table 3-19 on a bit-by-bit basis, and writes the value back.

Table 3-19 Standard Boot Config Table Options

Set Mask Bit	Clear Mask Bit	Operation
0	0	Bit value is unchanged
1	0	Bit value is set
0	1	Bit value is cleared
1	1	Bit value is toggled

3.5.4.2 Branch Entry

A branch entry has address ≠ 0, set mask = 0, clear mask = 0. The boot ROM makes a function call to the address. On return (if there is one) the table processing continues.

3.5.4.3 Terminate Entry

The table termination field has address, set mask, and clear mask all set to 0. When this entry is found, the boot ROM modifies the current active boot parameter table. The boot mode is changed to I²C master boot parameter table mode, the address is changed to the current next address value, and the boot is rerun.

3.5.5 Master Boot - Transmit Mode

Master transmit mode cannot be initiated from bootstrapped pins. To enter this mode, the device is pin-strapped as I²C Master mode boot. The boot parameter table loaded sets the new boot mode to I²C master transmit mode.

When it is in master transmit mode, the boot ROM reads a block of data from the I²C EEPROM exactly as it does in master mode, supporting both boot tables and boot configuration tables. But in addition to processing the I²C block, the boot ROM retransmits the block from the I²C bus to the address specified in the broadcast address field of the boot parameter table.

Before sending the first data block, the boot ROM must send the following block to the broadcast address on the I²C bus. The slave boot processors use this field to set their options flag. This is required to allow the slaves to know if the next received data blocks contain boot tables data or boot configuration tables.

Table 3-20 I²C Master Transmit Mode Broadcast Options

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
block Size = 6															
Checksum															
I²C Slave Boot mode Options filed															

3.5.6 Slave Boot

In I²C slave mode, the boot ROM operates the I²C device in receiver mode. Blocks received from the I²C Master are expected to have the standard I²C block header. The first block received must be of the form shown in Table 3-28. This allows the slave to know if the subsequent blocks contain boot table data or boot configuration tables. For boot configuration tables, after the end of the table is reached, the boot ROM reexecutes in I²C slave mode, and once again expects the first block received from the I²C bus to be the options block.

3.5.7 Secondary Stage Bootloader

For each boot mode, the bootloader uses a default boot parameter table for that mode and the values for certain parameter from the bootstrap pin configuration by reading the DEVSTAT register. Even though this is convenient, it does not give a lot of flexibility in terms of the possible configurations for a given boot mode. Also, there is no provision to configure additional peripherals that might apply to some scenarios. For example, it might be usefule to configure DDR to store an application that was downloaded during the Ethernet boot mode.

To facilitate such unique scenarios, the I²C boot mode provides two-stage booting. When the KeyStone devices boot up in I²C mode, they always enter the boot code twice. The first stage might be used to configure the additional peripheral using a boot config table, then point to a second parameter table to be loaded. In the second stage, the second parameter table loaded is used by the bootloader to boot in the specified mode. The first stage might also be used to load a boot application using a boot table; at the end of the boot table the bootloader transfers control to the entry point of the downloaded boot application to execute. To know which type of table to use in a scenario, it is important to understand the table structure. The section below describes how each of these three types of tables are generated and combined into one I²C EEPROM image.

3.5.7.1 Boot Parameter Table

The structure of the boot parameter table depends on the boot mode. The first offset in any boot parameter table is the length of the parameter table. The next offset is used for check sum. The third offset denotes the extended boot mode. Based on the boot mode selected, the rest of the boot parameter table is set. Table 3-21 shows an example of an I²C boot mode parameter table.

Note—The Next device address is not used in the boot parameter table.

Table 3-21 Boot Parameter Table

Offset	Field	Value
0	Length	0x20(32 bytes)
2	Checksum	0 (Checksum disabled)
4	Boot Mode	5 (I²C)
6	Port Num	0
8	PLL configuration (MSW)	0
10	PLL configuration (LSW)	0
12	Option	1 (Boot Table)
14	Boot Dev Addr	0x80
16	Boot Dev Addr Ext	0x50
18	Broadcast Addr	0
20	Local Address	0
22	Device Freq	320(800MHz)
24	Bus Frequency	64(100kHz)
26	Next Dev Addr	0(not used)
28	Next Dev Addr Ext	0(not used)
30	Address Delay	0
End of Table 3-21		

3.5.7.2 Boot Configuration Table

A boot configuration table is used if certain peripherals must be programmed with values that differ from their reset values before loading an application. For example, if the application needs to be loaded into DDR memory, the boot configuration load can be used to program DDR registers and enable the DDR peripheral. Table 3-22 shows an example of a boot parameter table.

Table 3-22 Boot Parameter Table For Calling a Boot Config Portion (Part 1 of 2)

Offset (Byte)	Field	Value
Field		
0	Length	0x20(32 bytes)
2	Checksum	0 (Checksum disabled)
4	Boot Mode	5 (I²C)
6	Port Num	0
8	PLL configuration (MSW)	0
10	PLL configuration (LSW)	0

Table 3-22 Boot Parameter Table For Calling a Boot Config Portion (Part 2 of 2)

Offset (Byte)	Field	Value
12	Option	2 (Boot Config)
14	Boot Dev Addr	0x400
16	Boot Dev Addr Ext	0x50
18	Broadcast Addr	0
20	Local Address	0
22	Device Freq	320(800MHz)
24	Bus Frequency	64(100kHz)
26	Next Dev Addr	100
28	Next Dev Addr Ext	50
30	Address Delay	0
End of Table 3-22		

According to Table 3-22, the config table is available at the start address 0x400 in the EEPROM. After processing the config table a second set of parameter tables are loaded from the start address of 0x80 in the EEPROM.

Each table entry in the boot config table has three elements. The first element is the address to be modified, the second element is the set mask, and the third element is the clear mask. The bootloader reads the specified address, then sets any bits that are set in the set mask element and clears any bits that are set in the clear mask element. If both the set and clear mask elements are 0, the value in the address field is branched via a standard call with the return address stored in register B3. The boot configuration table is terminated when all three elements are 0. Table 3-23 shows an example boot configuration.

Table 3-23 Boot Config Table Format

Offset	Data	Operation
0x0	0x0093001C	Set 16 bits MSBs and clear 16LSBs at address 0x0093001C
0x4	0xFFFF0000	
0x8	0x0000FFFF	
0xC	0x00000000	Termination
0x10	0x00000000	
0x14	0x00000000	

3.5.7.3 Boot Table

The boot table is a block of data that contains the code and data sections to be loaded by the bootloader, as well as other information such as the entry point address. The boot table is created by the hex conversion utility (a standard component of the TMS320C6000 Assembly Language Tools), based on the COFF (common object file format) output of the linker for the application code. The hex conversion utility provides several output options, including industry-standard ASCII formats that can be used to program parallel or serial EEPROMs, and formats that can be used in code for a host to transmit the boot table to the DSP.

Code and data sections are inserted into the boot table automatically by the hex conversion utility. The hex conversion utility uses information embedded by the linker in the .out file to determine each section's destination address and length. Adding these sections to the boot table requires no special intervention by the user. The hex conversion utility adds all initialized sections in the application to the boot table. The remaining information included in this section describes the format of the sections in the boot table.

Each section is added to the boot table in the same format. The first entry is a 32-bit count representing the length of the section in bytes. The next entry is a 32-bit destination address, where the first byte of the section is copied.

The bootloader continues to read and copy these sections until it encounters a section whose byte count is zero. This indicates the end of the boot table; then, the bootloader branches to the entry point address (specified at the beginning of the boot table) and begins executing the application.

The boot table format is as follows:

- 32-bit header record indicating where the bootloader should branch after it has completed copying the data
- For each COFF section
  - 32-bit Section byte count
  - 32-bit Section address (destination address for the copy)
  - The data to be copied
- A 32-bit Termination record (0x00000000)

To create the boot table, use the following steps:

1. Use the hex conversion utility (hex6x.exe) revision 6.0A or later.
2. Use the -boot option to cause the hex conversion utility to create a boot table.
3. Specify the romwidth and memwidth to both be 32 bits.
4. Specify the entry point using the -e entry_point_address option. The entry point is the address to which the bootloader transfers execution when the boot load is complete.
5. Specify the output filename using the -o output_filename option. If you do not specify an output filename, the hex conversion utility creates a default filename based on the output format.
6. Specify the endianness to match the compilation, -order L for little endian, -order M for big endian.
7. Correct any sections that are not multiples of 32 bits. The C compiler always generates sections whose lengths are multiples of 32 bits. This may not be the case for any sections declared in assembly. For little endian systems, the byte order must be swapped for these remaining bytes.

Example **Creating a Boot Table for ASCII Output**

To create a boot table for the application my_app.out with the following conditions:

- Little endian compilation
- Desired output is ASCII format in a file called my_app.hex

Use the following options on the hex conversion utility command line or command file:

```
-boot; option to create a boot table
-a; ASCII format
-e _c_int00; Standard entry point for C library
-order L; Little endian format
-memwidth32; memory width
-romwidth32; rom width
-o my_app.hex; specify the output filename
my_app.out; specify the input file
```

3.6 SPI Boot Operation

3.6.1 Basic Boot Operation

The boot parameter table (Table 3-24) is used to configure SPI boot. Because SPI is operated through direct register reads and writes, no EDMA configurations are required.

Table 3-24 SPI Boot Parameter Table

Byte Offset	Name	Description
12	Options	Bits 0 & 1 Modes 00 Load a boot parameter table from the SPI 01 Load boot records from the SPI (boot tables) 10 Load boot config records from the SPI (boot config tables) 11 Reserved Bits 2 - 15 Reserved
14	Mode	SPI mode, 0-3
16	Address Width	The number of bytes in the SPI device address. Can be 2 or 3 (16 or 24 bit)
18	Data Width	The data width of the device. Can be 8 or 16
20	NPin	The operational mode, 3 or 4 pin
22	Chipsel	The chip select used (valid in 4 pin mode only). Can be 0-3.
24	Read Addr MSW	The first address to read from, MSW (valid for 24 bit address width only)
26	Read Addr LSW	The first address to read from, LSW
28	CPU Freq MHz	The speed of the CPU, in MHz
30	Bus Freq, MHz	The MHz portion of the SPI bus frequency. Default = 5MHz
32	Bus Freq, kHz	The kHz portion of the SPI buf frequency. Default = 0
End of Table 3-24		

The SPI boot configuration derives the parameters to fill the boot parameter table from the boot device configuration as shown in Table 3-25 and Table 3-26.

Table 3-25 SPI Boot Mode Device Configuration

9	8	7	6	5	4	3	2	1	0
Mode (Clk Pol/Phase)		4, 5 Pin	Addr Width	Chip Select		Parameter Index (1 - 0 bits also used for SR ID)			

Table 3-26 SPI Boot Mode Device Configuration Description (Part 1 of 2)

Bit Field	Value	Description
Mode	0	Data is output on the rising edge of SPICLK. Input data is latched on the falling edge.
	1	Data is output one half-cycle before the first rising edge of SPICLK and on subsequent falling edges. Input data is latched on the rising edge of SPICLK.
	2	Data is output on the falling edge of SPICLK. Input data is latched on the rising edge.
	3	Data is output one half-cycle before the first falling edge of SPICLK and on subsequent rising edges. Input data is latched on the falling edge of SPICLK.
4,5 pin	0	4 pin mode used
	1	5 pin mode used
Addr Width	0	16 bit address values are used
	1	24 bit address values are used

Table 3-26 SPI Boot Mode Device Configuration Description (Part 2 of 2)

Bit Field	Value	Description
Chip Select	0-3	The chip select field value
Parameter Table Index	0-3	Specifies which parameter table is loaded
SR Index	0-3	Smart Reflex Index
End of Table 3-26		

After initializing the peripherals by the boot parameter table shown in Table 3-21, the boot code reads either a boot parameter table or boot config table that is at the address specified by the first boot parameter table and executes it directly. Similar to the I²C in the boot config table mode, the boot code consist of data in blocks with the first four bytes of data as header which is stripped and the rest of the data blocks are used for booting process.

In the Boot Parameter mode, SPI reads the data block from the EEPROM into the internal boot parameter table and reenters the bootloader. The table loaded can contain a boot parameter table for any boot mode.

After the boot process completion, the interrupt maps are restored to their default values.

3.7 HyperLink Boot Operation

3.7.1 Basic Boot Operation

The HyperLink boot through the ultra-short-range (HyperLink) connection is configured using the four SerDes lanes. The SerDes clock configurations are obtained from the boot configuration field bits as shown in Table 3-27 and Table 3-28.

Table 3-27 HyperLink Boot Mode Device Configuration

6	5	4	3	2	1	0
Reserved	Date Rate		Ref Clock		SR Index	

Table 3-28 HyperLink Boot Device Configuration Description

Bit Field	Value	Description
SR Index	0-3	Smart Reflex Index
Ref Clock	0	156.25 MHz
	1	250 MHz
	2	312.5 MHz
Data Rate	0	1.25 Gbaud
	1	3.125 Gbaud
	2	6.25 Gbaud
	3	12.5 Gbaud

HyperLink boot code initializes the chip-level interrupt controller to interrupt the DSP after the boot. After setting up the interrupt configuration, the boot ROM executes an idle instruction. The remote device loads the memory directly and generates the HyperLink interrupt. When the boot code is awoken, the interrupt maps are restored to their default values and the DSP branches to the address in DSP_BOOT_ADDR0. As with PCI, if the value in DSP_BOOT_ADDR0 is still zero after wakeup, the ROM again executes an idle.

HyperLink mapping can be configured by the master, but the bootloader sets up the following initial mappings. In all cases, the selector size is 4 MB. Table 3-29 and Table 3-30 show the TCI661x/C6670 and TCI6608/C667x mappings for the HyperLink segment, respectively.

Table 3-29 TCI661x/C6670 Boot ROM HyperLink Mapping (Part 1 of 2)

Segment	Size	Translated Address	Description
0	512kB	0x10800000	CorePac0 local L2
1	512kB	0x11800000	CorePac1 Local L2
2	512kB	0x12800000	CorePac2 Local L2
3	512kB	0x13800000	CorePac3 Local L2
4	512kB	0x14800000	CorePac 4 Local L2
5	512kB	0x15800000	CorePac5 Local L2
6	512kB	0x16800000	CorePac6 Local L2
7	512kB	0x17800000	CorePac7 Local L2
8	128kB	0x08000000	XMC config
9	1MB	0x0bc00000	MSMC Config
10	4MB	0x0c000000	MSMC Memory
11	4MB	0x01c00000	Config Regs

Table 3-29 TCI661x/C6670 Boot ROM HyperLink Mapping (Part 2 of 2)

Segment	Size	Translated Address	Description
12	4MB	0x02000000	Config Regs
13	4MB	0x02400000	Config Regs
14	2MB	0x02800000	Config Regs
11	512 Bytes	0x21000000	DDR Config
12 - 63	4MB	0x80000000 (4MB steps)	DDR Memory
End of Table 3-29			

Table 3-30 TCI6608/C667x Boot ROM HyperLink Mapping

Segment	Size	Translated Address	Description
0	512kB	0x10800000	Core 0 local L2
1	512kB	0x11800000	Core 1 Local L2
2	512kB	0x12800000	Core 2 Local L2
3	512kB	0x13800000	Core 3 Local L2
4	512kB	0x14800000	Core 4 Local L2
5	512kB	0x15800000	Core 5 Local L2
6	512kB	0x16800000	Core 6 Local L2
7	512kB	0x17800000	Core 7 Local L2
8	128kB	0x08000000	XMC config
9	1MB	0x0bc00000	MSMC config
10	4MB	0x0c000000	MSMC memory
11	4MB	0x01c00000	Config Regs
12	4MB	0x02000000	Config Regs
13	4MB	0x02400000	Config Regs
14	2MB	0x02800000	Config Regs
15	512 Bytes	0x21000000	DDR Config
16 - 63	4MB	0x80000000 (4MB steps)	DDR Memory
End of Table 3-30			

Index

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps

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