TMS320C645x/C647x DSP Bootloader

User's Guide

Texas Instruments

Literature Number: SPRUEC6G
March 2006–Revised June 2011

Contents

List of Figures

List of Tables

Read This First

About This Manual

This document describes the features of the on-chip bootloader provided with the following TMS320C645x/C647x Digital Signal Processors (DSPs): C6454, C6455, C6457, C6472, and C6474.

This document contains preliminary data current as of the publication date and is subject to change without notice.

Important Notice Regarding Bootloader Program Contents:

Texas Instruments may periodically update the bootloader code supplied in the ROM to correct known problems, provide additional features or improve functionality. These changes may be made without notice as needed. Although changes to the ROM code preserve functional compatibility with prior versions, the locations of functions within the main bootloader code may change. You should avoid calling these functions directly as the code may change in the future.

Related Documentation From Texas Instruments

The following documents describe the TMS320C6000™ devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

SPRU732 — ***TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide.*** Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C64x and TMS320C64x+ digital signal processors (DSPs) of the TMS320C6000 DSP family. The C64x/C64x+ DSP generation comprises fixed-point devices in the C6000 DSP platform. The C64x+ DSP is an enhancement of the C64x DSP with added functionality and an expanded instruction set.

SPRU198 — ***TMS320C6000 Programmer's Guide.*** Describes ways to optimize C and assembly code for the TMS320C6000™ DSPs and includes application program examples.

SPRU871 — ***TMS320C64x+ Megamodule Reference Guide.*** Describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

SPRU974 — ***TMS320C645x DSP Inter-Integrated Circuit (I2C) Module User's Guide.*** This document describes the inter-integrated circuit (I2C) module in the TMS320C645x digital signal processors (DSPs).

SPRU976 — ***TMS320C645x DSP Serial RapidIO (SRIO) User's Guide.*** This document describes the Serial RapidIO® (SRIO) peripheral on the TMS320C645x digital signal processors (DSPs).

SPRU975 — ***TMS320C645x DSP Ethernet Media Access Controller (EMAC)/ Management Data Input/Output (MDIO) User's Guide.*** This document provides a functional description of the Ethernet Media Access Controller (EMAC) and Physical layer (PHY) device Management Data Input/Output (MDIO) module integrated with TMS320C645x digital signal processors (DSPs).

SPRUGK3 — ***TMS320C6457 DSP Inter-Integrated Circuit (I2C) Module User's Guide.*** This document describes the inter-integrated circuit (I2C) module in the TMS320C6457 digital signal processors (DSPs).

SPRUGK4 — ***TMS320C6457 DSP Serial RapidIO (SRIO) User's Guide.*** This document describes the Serial RapidIO (SRIO) peripheral on the TMS320C6457 digital signal processors (DSPs).

SPRUGK9 —***TMS320C6457 DSP Ethernet Media Access Controller (EMAC)/ Management Data Input/Output (MDIO) User's Guide.*** This document provides a functional description of the Ethernet Media Access Controller (EMAC) and Physical layer (PHY) device Management Data Input/Output (MDIO) module integrated with the TMS320C6457 digital signal processors (DSPs).

SPRUFX0 — ***TMS320C6472 DSP Technical Reference Manual.*** This document provides a functional description of the peripherals on the TMS320C6472 digital signal processors (DSPs).

SPRUG22 — ***TMS320C6474 DSP Inter-Integrated Circuit (I2C) Module User's Guide.*** This document describes the inter-integrated circuit (I2C) module in the TMS320C6474 digital signal processors (DSPs).

SPRUG23 — ***TMS320C6474 DSP Serial RapidIO (SRIO) User's Guide.*** This document describes the Serial RapidIO (SRIO) on the TMS320C6474 digital signal processors (DSPs).

SPRUG08 — ***TMS320C6474 DSP Ethernet Media Access Controller (EMAC)/ Management Data Input/Output (MDIO) User's Guide.*** This document provides a functional description of the Ethernet Media Access Controller (EMAC) and Physical layer (PHY) device Management Data Input/Output (MDIO) module integrated with the TMS320C6474 digital signal processors (DSPs).

TMS320C6000, Code Composer Studio are trademarks of Texas Instruments.
RapidIO is a registered trademark of RapidIO Trade Association.
All other trademarks are the property of their respective owners.

TMS320C645x/C647x Bootloader

1 Introduction

This section provides a description of the features of the on-chip bootloader provided with the following TMS320C645x/C647x Digital Signal Processors (DSPs): C6454, C6455, C6457, C6472, and C6474.

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

1.1 Bootloader Features

The bootloader is DSP code that transfers application code from an external source into internal or external program memory after the DSP is taken out of reset. The bootloader allows application code to reside in slow non-volatile external memory and be transferred to high-speed internal memory for execution, or to be transferred from a host processor to the DSP after the DSP is taken out of reset. The bootloader is permanently stored in the ROM of the DSP starting at byte address 0x00100000 for C6454, C6455 and C6472, 0x3C000000 for C6457 and C6474 devices.

To accommodate different system requirements, the bootloader offers a variety of methods (boot modes) to transfer the application into DSP memory. The following is a list of the available boot modes and a summary of their functional operation:

- Direct execution from internal memory (L2), No boot: The CPU executes directly out of L2 (operation is invalid if there is no valid code in L2). Typically only used during debug on emulator.
- Host boot (HPI/PCI): In this mode, the bootloader waits until the code to be executed is loaded into on-chip memory by a host device via the HPI or the PCI. Code execution begins when the host indicates to the bootloader that the application has been loaded.
- EMIF boot from 8-bit external asynchronous memory: The code executes out of external memory.
- Master I2C boot: The application is loaded from an I2C EEPROM, one section at a time, using a boot table to determine the length and the starting address for each section.
- Slave I2C boot: An external I2C master sends the application to the DSP, also using a boot table.
- Serial RapidIO® (SRIO) boot: An external host loads the application via the SRIO peripheral, using directIO protocol. A doorbell interrupt is used to indicate that the code has been loaded.
- EMAC boot: An external host loads the application via EMAC peripheral.
- UTOPIA boot: An external host loads the application via UTOPIA peripheral.

1.2 Terms and Abbreviations

1X mode — 1 Tx and 1 Rx differential pair

CPPI — Communications Port Programming Interface

DDR2 — Double Data Rate 2

DIX — Digital, Intel, and Xerox

DSK — Developer Starter Kit

EEPROM — Electrically Erasable Programmable ROM

EMAC — Ethernet Media Access Control

EMIF — External Memory Interface

GEL — Gain Extension Language used in Code Composer Studio™

GMII — Gigabit Media Independent Interface

HPI — Host Port Interface

I2C —Inter-Integrated Circuit

L2 — Level 2

MAC — Media Access Control

MII — Media Independent Interface

PCI — Peripheral Component Interconnect

PDMA — Packet Direct Memory Access

PHY — Physical Layer Device

RMII — Reduced Media-Independent Interface

SERDES — Serializer/Deserializer

S3MII — Source Synchronous Serial Media Independent Interface

SGMII — Serial Gigabit Media Independent Interface

SMII — Serial Media Independent Interface

SRIO — Serial RapidIO

UTOPIA — Universal Test and Operations Physical Interface for ATM

2 C6454/C6455 Bootloader Operation

The structure and operation of the C6454/55 bootloader are described in the following sections.

2.1 Bootloader Initialization

When the bootloader begins execution, the program performs some initialization of the DSP prior to loading code. Table 1 describes the DSP resources that are configured by the bootloader. Table 2 lists the cache settings while L2 cache is always disabled.

Table 1. C6454/55 Bootloader Initialization

Resource	Initialization Value
Interrupts	Interrupts are left enabled (GIE=1). PCI and SRIO boot interrupts are enabled, with the interrupt mux registers configured to route the PCI and SRIO interrupts to the core. ISTP (the vector table pointer) is set to point into the ROM at 0x103c00. After the interrupt arrives, the ISTP and interrupt mux registers are restored to their default values.
Memory	L2 Memory, from 009F F080h to 009F FFFFh for C6455 and from 008F F080h to 008F FFFFh for C6454, is reserved during the boot and can be reclaimed by application after the boot.
PLL1 Controller	For SRIO and PCI boot, the bootloader configures the PLL1 Controller such that CLKIN1 is multiplied by 15. For all other boot modes, the PLL1 Controller is in Bypass mode. The PLL1 Controller configuration is not changed when the bootloader exits.
Peripheral Powerup	Peripherals are enabled as required by the bootloader in the selected mode. Peripherals are not disabled when the bootloader exits.
Registers	The state of all CPU registers, with the exception of the PC, must be considered random on bootloader exit.

Table 2. C6454/55 Cache Settings

BOOTMODE[3:0]	Boot Description	L1P/L1D Cache Settings
0	No boot	If PCI_EEAI = 1, 0 KB If PCI_EEAI = 0, 32 KB by default. However, without valid code in L2, PC may branch into the ROM and cache defaults settings may be overwritten.
1	HPI/PCI boot	32KB
2	EMIFA fast boot	32KB
3	Reserved	N/A
4	EMIFA ROM boot	32KB
From 5 to 15	Other boots	0 KB

After the initialization is performed, the bootloader loads the on-chip RAM according to the boot mode selected, and then causes the DSP to begin execution of the loaded code. At that point, the boot load process is complete. Whenever the DSP is reset, the CPU starts execution of the bootloader again, and the entire boot load process is repeated.

The following sections describe the various C6454/55 boot modes and boot tables in detail.

2.2 Boot Mode Selection

The desired boot mode is selected by setting the four boot mode select pins BOOTMODE[3:0], which are sampled during reset. The BOOTMODE pins are shared with EMIFA address pins [19:16].

Table 3 describes the available boot mode options and their corresponding BOOTMODE pin configurations.

Table 3. Boot Mode Selection Options

BOOTM3	BOOTM2	BOOTM1	BOOTM0	Boot Mode Source	See
0	0	0	0	No boot, execution begins from the base of L2 (0x800000)	Section 2.3.1
0	0	0	1	Host boot (HPI)	Section 2.3.2
0	0	1	0	Reserved	-
0	0	1	1	Reserved	-
0	1	0	0	EMIFA ROM boot	Section 2.3.3
0	1	0	1	Master I2C boot	Section 2.3.4
0	1	1	0	Slave I2C boot	Section 2.3.5
0	1	1	1	Host boot (PCI)	Section 2.3.2
1	X	0	0	Serial RapidIO boot, Configuration 0 (4 1x ports)[1]	Section 2.3.6
1	X	0	1	Serial RapidIO boot, Configuration 1 (1 4x port)[1]	Section 2.3.6
1	X	1	0	Serial RapidIO boot, Configuration 2 (1 4x port)[1]	Section 2.3.6
1	X	1	1	Serial RapidIO boot, Configuration 3 (1 4x port)[1]	Section 2.3.6

[1] SRIO boot applies to C6455 only, reserved for C6454.

2.3 Boot Mode Options

2.3.1 Direct Execution From External Asynchronous Memory - No Boot

When BOOTMODE[3:0] = 0000b, the no boot option is selected. In this mode, the bootloader program does not execute. The DSP begins execution at the base of internal L2 memory at byte address 0x800000.

2.3.2 Host Boot Mode (HPI/PCI)

In host boot mode, an external host can load code and data directly into the DSP memory while the CPU waits. Host boot does not use a boot table. The code and/or data sections are directly loaded to the desired locations by the host. When the host has finished loading the application, it generates a host interrupt (HPI or PCI), the CPU then begins executing at the base of L2.

An HPI host can use the DSPINT bit of the Host Port Interface Control register (HPIC) to generate an interrupt to the CPU while a PCI host can use the DSPINT bit of the PCI Status Set register (PCISTATSET). In the case of PCI, the DSPINT bit of the Back End Application Interrupt Enable register must also be set to 1.

If PCI boot is selected, the bootloader configures the PLL1 Controller such that CLKIN1 is multiplied by 15. More specifically, PLLM is set to 0Eh (x15) and RATIO is set to 0 (÷1) in the PLL1 Multiplier Control register (PLLM) and PLL1 Pre-Divider register (PREDIV), respectively.

2.3.2.1 PCI Auto-Initialization

If PCI auto-initialization through I2C EEPROM is enabled through the PCI_EEAI configuration pin, regardless of the selected boot mode, the on-chip bootloader also powers-on the PCI and I2C peripherals, initializes the I2C interface, and reads the PCI configuration values and a 16-bit checksum from an I2C EEPROM (starting at I2C address 0x400). Table 4 shows the register layout in I2C EEPROM. If the checksum verification passes, the bootloader writes the values read from I2C EEPROM into the appropriate PCI Hook Configuration registers and PCI Back End Configuration registers (back end registers written to only after power-on reset). Additional details about auto-initialization and PCI reset can be found in the *TMS320C645x DSP Peripheral Component Interconnect (PCI) User's Guide* (SPRUE60).

Table 4. I2C EEPROM Memory Layout

Byte Address	Contents
0x400	Vendor ID [15:8]
0x401	Vendor ID [7:0]
0x402	Device ID [15:8]
0x403	Device ID [7:0]
0x404	Class code [7:0]
0x405	Revision ID [7:0]
0x406	Class code [23:16]
0x407	Class code [15:8]
0x408	Subsystem vendor ID [15:8]
0x409	Subsystem vendor ID [7:0]
0x40a	Subsystem ID [15:8]
0x40b	Subsystem ID [7:0]
0x40c	Max_Latency
0x40d	Min_Grant
0x40e	Reserved (use 0x00)
0x40f	Reserved (use 0x00)
0x410	Reserved (use 0x00)
0x411	Reserved (use 0x00)
0x412	Reserved (use 0x00)
0x413	Reserved (use 0x00)
0x414	Reserved (use 0x00)
0x415	Reserved (use 0x00)
0x416	Reserved (use 0x00)
0x417	Reserved (use 0x00)
0x418	Reserved (use 0x00)
0x419	Reserved (use 0x00)
0x41a	Checksum [15:8]
0x41b	Checksum [7:0]

2.3.3 EMIFA Boot Mode

In EMIFA boot mode, the bootloader simply branches to the base address of EMIFA CE3 (0xB000 0000). Interrupts are disabled.

2.3.4 I2C EEPROM Boot Mode

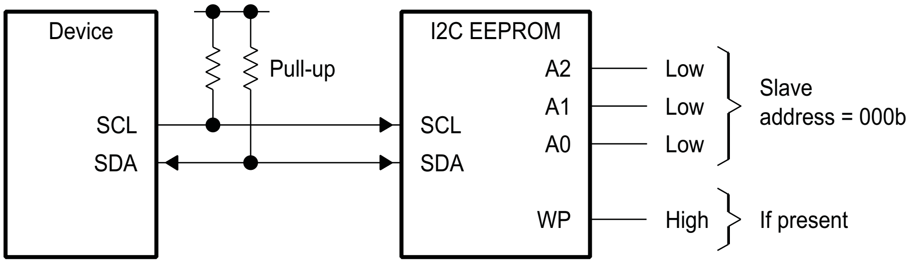
This section assumes familiarity with the I2C operation using the master receiver and master transmitter modes. For detailed information on the I2C, see the *TMS320C645x DSP Inter-Integrated Circuit (I2C) Module User's Guide* (SPRU974).

The bootloader supports boot from I2C EEPROMs or devices operating as I2C slaves that emulate the appropriate format. The bootloader has the following requirements for the I2C EEPROMs:

- The memory device complies with Philips I2C Bus Specification v 2.1.
- The memory device uses two bytes for internal addressing.
- The memory device has the capability to auto-increment its internal address counter such that the contents of the memory device can be read sequentially.

In I2C boot mode, the DSP acts as the master and the I2C EEPROM acts as the slave. Figure 1 shows the minimum connection required between the DSP and one I2C EEPROM. The required pull-ups must be placed on SDA and SCL to ensure that the I2C EEPROM interface works correctly.

Figure 1. Signal Connections for I2C EEPROM Boot Mode



Some I2C EEPROMs have a write-protect (WP) feature that prevents unauthorized writes to memory. This feature is not needed for boot loading purposes because the DSP only reads data from the I2C EEPROMs. The write protect feature can be enabled or disabled without impacting bootloader operation.

The bootloader requires the I2C EEPROM slave address to be 0x50. Other EEPROM slave addresses can then be used as specified in the boot parameter table loaded from the initial EEPROM.

The frequency of the I2C bus is initially set to CLKIN1/6600. For example, with a CLKIN1 frequency of 50 MHz, the serial clock frequency would be set to 7.57 kHz. Subsequently, the bootloader reads the actual CPU frequency and desired I2C frequency values from boot parameters (see Table 6) and reprograms the I2C.

2.3.4.1 *I2C EEPROM Data Blocking*

All data stored on the I2C EEPROM are stored in blocks. Each block has a maximum length of 128 bytes, including the 4 byte block header. shows the format of the block.

Table 5. C6454/55 I2C EEPROM Block Format

Offset (bytes)	Size (bytes)	Name	Value
0	2	Block size	The size of the block, including the header.
2	2	Checksum	The ones complement check sum, including the block size and checksum fields. Valid checksum values are 0 and -0.
4-126	0-124	Data	Data

The bootloader reads data from the I2C EEPROM in blocks. If the checksum shows a failure, the block is re-read until the checksum is valid. A value of 0 in the checksum field disables the checksum check.

2.3.4.2 C6454/55 Boot Parameter Structure

The I2C boot sequence begins with the DSP reading a block of boot parameters from the I2C EEPROM. The boot parameter table complies with I2C EEPROM data blocking format. The boot parameters begin at EEPROM address 0, and a boot parameter block consists of 128 bytes. The DSP calculates the address of boot parameter block to load based on the value of the CFGGP[2:0] bits of the Device Status Register (DEVSTAT) as follows: address = 0x80 * CFGGP[2:0]. This allows the DSP to read one of eight possible boot parameter blocks. The values in this block determine how the boot process proceeds. Table 6 shows the structure of these boot parameters. Each value is 2 bytes, and the bytes must be stored in big endian format (most significant byte at the lowest address), regardless of the endianness setting of the processor. The structure has a total length of 26 bytes.

Table 6. C6454/55 I2C Boot Parameter Table

Offset (byte)	Field	Value	
4	Boot mode	The boot mode that is used by the bootloader	
		0x0-0xF	See Table 3.
		0x100	I2C Master Write (extended mode)
6	Options	I2C options:	
		Bits 01-00	Boot table type
		00	Boot parameter mode
		01	Boot table mode
		10	Boot configuration mode
		11	Slave receive boot
		Bits 04-02	EEPROM type
		00	24C00 to 24C16A
		01	24C32 to 24C1024
		10	24CW256
		11	Reserved
8	Device address (LSW)	For further I2C boot, the lower 16 bits of the address in the I2C EEPROM	
10	Device address (MSW)	For further I2C boot, the upper 16 bits of the address in the I2C EEPROM. This is an abstraction; this value is used as the I2C EEPROM slave address (default 0x50).	
12	Broadcast address	If the boot mode is master broadcast, the boot tables are read from the I2C EEPROM and are re-sent to this slave address (default 0x0).	
14	Slave address	The address used by the DSP as its local slave address	
16	CPU frequency (MHz)	The actual CPU frequency as measured from SYSREFCLK of the PLL1 Controller. This parameter configures the I2C clock. Note that the PLL1 Controller is programmed by the bootloader for some boot modes (see Table 1).	
18	I2C clock frequency (kHz)	The desired I2C serial clock frequency	
20	Next device address (LSW)	If options indicate a boot configuration table load, this address is used as the address in the I2C ROM to find the next boot parameter table to load.	
22	Next device address (MSW)	The upper 16 bits of the address, used as the slave address of the I2C EEPROM	
24	Address delay	Delay between address write and read from I2C EEPROM (in cycles)	

The two LSBs of the options field (offset 6 bytes) define what the bootloader expects to find in the I2C EEPROM at the offset specified by the Device Address (LSW), and how it should proceed. A value of 0 indicates another boot parameter table. A value of 1 indicates a boot table (i.e., a table which contains initialized code and data sections, see Section 6.2.2). A value of 2 indicates a boot configuration table (table used to configure registers, see Section 6.2.3). A value of 3 is reserved, and the remaining bits must be set to 0.

After the DSP reads and remembers the boot parameter table, the bootloader performs a boot re-entry. On this pass the code executes based on the values provided by the boot parameter table.

If the options indicate a boot table is loading, then the bootloader reads from the I2C EEPROM address specified in the boot table until the end of the table is reached (i.e., until all code and data sections are loaded), and immediately begin execution of the loaded code by branching to the entry point specified at the beginning of the boot table (see Section 6.2.2.1).

If the options indicate a boot configuration table is loading, then the bootloader reads from the specified I2C EEPROM address until the end of the boot configuration table. This typically initializes various registers (see Section 6.2.3). The next device address previously read from the boot parameter block is copied into device address, the boot options are cleared, and the bootloader performs a boot re-entry. This directs the ROM to read the I2C EEPROM at the specified device address, which should have another boot parameter table.

2.3.5 I2C Slave Boot Mode

In the I2C slave boot mode, the DSP is programmed as an I2C slave and waits for an I2C Master to send data, using a standard boot table format.

The procedure is the same as Section 2.3.4, except for how the DSP receives data. In this case, rather than performing a master address write followed by a master data read, the DSP configures its I2C interface for slave reads, its device address to the default slave address (0x4), and polls for receiver ready, reading one byte at a time.

This boot mode can be used when it is desired to boot multiple DSPs at the same time from the same I2C EEPROM. One DSP is configured for I2C Master Write (in the Options boot parameters field), and the remaining DSPs are configured for I2C slave boot mode. The slave DSPs come out of reset first, followed by the master DSP. The master DSP reads from I2C EEPROM and re-transmits to the broadcast address (default 0x4). Therefore, all DSPs boot at the same time. In some cases, if in the design, another I2C master is booting the slave C6454/55 DSP through the I2C. Then, the I2C master needs to act similarly to a C6454/55 DSP during I2C boot. First, the I2C bus on the master side needs to be configured to have the exact same frequency as the I2C module within the DSP. Second, the I2C master needs to send 6 bytes (count does not include slave address) to the slave DSP before sending the boot table:

04 xx xx yy yy zz zz

Where:

04 = the slave address for the C6454/55 DSP in slave I2C boot mode

xx xx = length

yy yy = checksum

zz zz = boot option

For example:

04 00 06 00 00 00 01 ← Order: slave address, length, checksum, boot option

Length = 6

Checksum = 0 (not used)

Boot option = 1 (for telling the slave DSP that the next coming data is the boot table); for other options, see the Options field in Table 6.

2.3.6 SRIO Boot Mode

In the SRIO boot mode, an external host can load code and data directly into the DSP memory while the CPU waits, similar to the HPI/PCI boot mode. The code and/or data sections are directly loaded to the desired locations, using the directIO model. When the host has finished loading the application, it signals through a doorbell interrupt and the CPU then begins executing at the base of L2.

The on-chip bootloader first configures the PLL1 Controller such that CLKIN1 is multiplied by 15. It then determines the SRIO boot configuration (0, 1, 2 or 3) and initializes and enables required blocks of the SRIO peripheral.

The SRIO boot configuration is determined by the BOOTMODE[1:0] pins (see Table 3). In boot configuration 0, the SRIO peripheral is configured as four 1x ports and the boot is performed on port 0. In the remaining configurations, the SRIO peripheral is configured as a single 1x/4x port.

The logical layer local device ID within the Serial RapidIO network is configured using BOOTMODE[2] as the address MSB and configuration pins CFGGP[2:0] as the three LSBs. If BOOTMODE[2]:CFGGP[2:0] is 0b1111, the default value of the device ID is 0xFF or 0xFFFF, and it must be configured/assigned by the host (maintenance packet to DEVICEID_REG1); otherwise, device ID = 2 + BOOTMODE[2]:CFGGP[2:0].

In the SRIO boot configuration 0, the bootloader configures the SRIO peripheral in the following sequence:

1. Peripheral blocks (registers BLK*n*_EN, GBL_EN): Blocks 0, 1, 2 and 5 are enabled (MMRs, LSU, MAU and Port 0, respectively). Global enable is also set.
2. PLL (register CFG0_CNTL): Multiplier is set to 10x, loop bandwidth to 1/12th of the reference clock, and the PLL is enabled. A delay of 1usec follows PLL enabling.
3. Receiver (registers CFGRX*n*_CNTL): Half rate, 10-bit width, set comma aligned, set normal polarity, etc.
4. Transmitter (registers CFGTX*n*_CNTL): Half rate, 10-bit width, set common mode, set output swing, set emphasis, set normal polarity, etc.
5. Processing Element Feature CAR (register PE_FEAT): 34-bit address, extended features pointer valid; common transport large systems support.
6. Source/Destination Operations CAR (registers SRC_OP, DEST_OP): All source operations, all destination operations, except atomic.
7. Maintenance block header (register SP_MB_HEAD): General endpoint device with software assisted error recovery.
8. Input/Output Port Enable (registers SP*n*_CTL).
9. Base ID: Same as the local Device ID.
10. IP-Level Port Mode (register SP_IP_MODE): 4 ports (1x mode each), Port-Write disabled, Packet accepted by the Phy layer with any DestID and forwarded to the logical layer.
11. Packet Forwarding (registers PF_16B_CNTL*n*, PF_8B_CNTL*n*): Disabled.
12. Doorbell interrupt routing (register DOORBELL0_ICRR): Routed to INTDST4.
13. Prescaler for physical layer timers (register IP_PRESCAL): 333 MHz.
14. Check Port OK: Continue if the port is initialized and is exchanging error-free control symbols with the attached device; otherwise, record an error in setting up the peripheral.
15. Boot Complete (register SP_PER_SET_CNTL): Writes to read-only registers are disabled after this point.
16. Assert Peripheral Enable (PCR register) to enable logical layer data flow.
17. Clear any pending interrupts (register ICRR).

Registers not mentioned above retain their default values.

In the SRIO boot configurations 1-3, the initialization is the same as in SRIO boot configuration 0, except for the registers that configure the SERDES ports.

Table 7 lists the required SerDES reference clock and associated link rate settings.

Table 7. C6454/55 SRIO Boot Configurations

Boot Configuration	SerDes Reference Clock	SRIO Link Rate
SRIO boot configuration 0	125 MHz	1.25 Gbps
SRIO boot configuration 1	156.25 MHz	3.125 Gbps
SRIO boot configuration 2	125 MHz	3.125 Gbps
SRIO boot configuration 3	312.5 MHz	3.125 Gbps

2.4 C6454/55 Bootloader Versions

2.4.1 Determining the Bootloader Version

The boot ROM can be identified by reading the ROM signature. The ROM address may very from one bootloader address to the next (see Table 8). The signature is a string of characters, stored one byte per 32 bit word. If the ROM boot executes this signature is copied as a packed byte string (order depends on the endianness configuration) to L2 at 0x9ff040.

2.4.2 Differences Between Bootloader Versions

Different versions of the on-chip bootloader exit on silicon revision 1.1 and 2.0 of the C6454/55 DSP. Table 8 summarizes the differences between two versions.

Table 8. Differences Between C6454/55 Bootloader Versions

Feature	C6454/55 Silicon Revision 1.1	C6454/55 Silicon Revision 2.0
Signature	v2.2 Wed Jul 27 17:25:44 2005 i2c pci_eeai Serial RapidIO	v2.3 Wed Feb 01 13:37:24 2006 i2c pci_eeai Serial RapidIO
Starting address of the signature string in ROM	0x103aa0	0x103b30
I2C address delay	Not programmable	Programmable (see Table 6)
SRIO PLL	Function not available to second stage bootloaders or boot configuration tables	Function available to second stage bootloaders and boot configuration tables
EMIF boot	EMIF boot with PCI auto-initialization enabled is not supported	EMIF boot with PCI auto-initialization enabled is supported
SERDES in SRIO boot configuration 0	Only SERDES 0 configured	All SERDES (0-3) configured
Default I2C frequency (see Section 2.3.4)	15*(CLKIN1/6600) if PCI auto-initialization is enabled, CLKIN1/6600 otherwise	CLKIN1/6600

3 C6457 Bootloader Operation

This section describes the structure and operation of the C6457 bootloader. The bootloader code is executed in the L3 ROM. All boot modes on C6457 are ROM based; i.e., execution is transferred to the L3 ROM base when the DSP is released from reset and, depending on the boot pins strapped, the relevant boot mode is executed.

3.1 Bootloader Operation

When the bootloader begins execution, the program performs some DSP initialization prior to loading the code. Table 9 describes the DSP resources that are configured by the bootloader.

Table 9. C6457 Bootloader Initialization

Resource	Initialization Value
Interrupts	Interrupts are left disabled (GIE=0) except for NO BOOT, HPI, and SRIO boot modes. When interrupts are enabled, the interrupt mux registers are configured to route the correct system event to the core. ISTP (vector table pointer) is set to point to the start of L3 ROM (0x3C000000) and the IER mask is set to enable the correct interrupts (Reset and NMI interrupts are always enabled). After interrupt, the ISTP and GIE are restored to their default values.
Memory	L1P/L1D is set to all cache. L2 memory from byte address 0x009FA000 - 0x009FFFFF (20480 bytes) is reserved for use as scratch memory during the boot process and can be reclaimed by the application after the boot.
PLL1 Controller	For I2C, HPI, SRIO, and MAC boot modes, the PLL1 is set to x16 mode.
Peripheral Powerup	Peripherals are powered up as required by the bootloader in the selected mode. Peripherals are not powered down when the bootloader exits. HPI, EMAC, and EMIFA modules need to be powered on.
Registers	The state of all CPU registers, with the exception of PC, must be considered random on bootloader exit.

L2 is configured as RAM only (no cache). The boot code uses a portion of L2 during the boot process (see the shaded area in Table 10). When the boot is complete, the application can reclaim this reserved memory.

Table 10. L2 Memory Map for Bootloader Code

Memory Range	Size (bytes)	Description
0x800000 - 0x9F9FFF		Not used by boot code. Application can use this space to download code and data sections.
0x9FA000 - 0x9FDFFF	16K	Allocated for storing received EMAC packets. In addition, it is used to store SRIO and HPI boot packets when boot table processing is selected for these modes.
0x9FE000 - 0x9FE7FF	2K	Allocated as stack by boot code.
0x9FE800 - 0x9FE8FF	256	Used by boot code to store boot parameters.
0x9FE900 - 0x9FEAFF	512	Allocated for un-initialized data structures used by the boot code.
0x9FEB00 - 0x9FFAFF	4K	Used for boot table processing by the boot code
0x9FFB00 - 0x9FFBFF	256	Boot version string
0X9FFC00 - 0x9FFDFF	512	Boot Statistics/diagnostics
0X9FFF00 - 0x9FFF0F	16	Boot ROM table pointers
0X9FFFFC - 0x9FFFFF	4	Application entry point (boot exit address)

3.2 Boot Mode Selection

The boot mode is selected by setting the boot mode pins BOOTMODE[3:0], which are latched during reset. The configuration pins GPIO[13:9] are mapped to CFGGP[4:0] which are are referred to as CFG[3:0] in Table 11 and throughout this chapter for the C6457 device.

Table 11. C6457 Boot Mode Selection

BOOTMODE[3:0]	Description	CFG[3:0]	See
0	No boot	None	Section 3.3.1
1	I2C Master (device address 0x50)	First EEPROM boot parameter block address	Section 3.3.2
2	I2C Master (device address 0x51)	First EEPROM boot parameter block address	Section 3.3.2
3	I2C Slave	3:0 + 1= Device ID	Section 3.3.3
4	HPI	3:0 = 2 boot table loading mode, otherwise direct loading mode	Section 3.3.4
5	EMIFA	None	Section 3.3.5
6	EMAC Master	3:0 Device ID	Section 3.3.6
7	EMAC Slave	3:0 Device ID	Section 3.3.6
8	EMAC Forced	3:0 Device ID	Section 3.3.6
9	Reserved	N/A	-
10	RapidIO Configuration 0	3:0 Node (0xF for default)	Section 3.3.7
11	RapidIO Configuration 1	3:0 Node (0xF for default)	Section 3.3.7
12	RapidIO Configuration 2	3:0 Node (0xF for default)	Section 3.3.7
13	RapidIO Configuration 3	3:0 Node (0xF for default)	Section 3.3.7
14-15	Reserved	N/A	-

3.3 Boot Mode Options

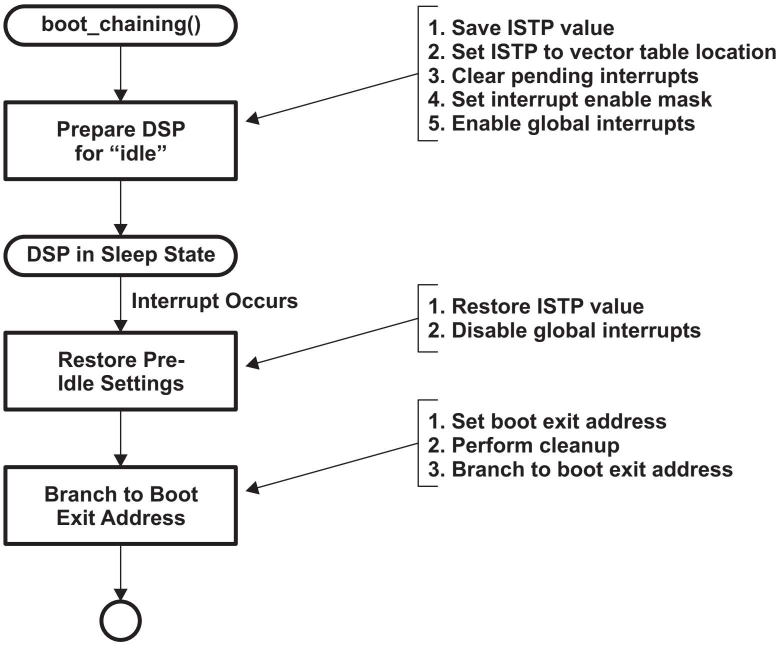
3.3.1 No Boot Mode

The C6457 device no boot mode is similar to the C6454/55 device's process, described in Section 2.3.1, with the following differences:

1. No-boot mode is ROM based instead of completely hardware based.
2. The bootloader puts the DSP in sleep state until an interrupt arrives. The boot code sets the IER mask to 0xFF and GIE=1 prior to entering sleep state.

The boot ROM code flow is shown in Figure 2.

Figure 2. Program Flow for No-Boot Mode


boot_chaining()
Prepare DSP for "idle"
1. Save ISTP value
2. Set ISTP to vector table location
3. Clear pending interrupts
4. Set interrupt enable mask
5. Enable global interrupts
DSP in Sleep State
Interrupt Occurs
Restore Pre-Idle Settings
1. Restore ISTP value
2. Disable global interrupts
Branch to Boot Exit Address
1. Set boot exit address
2. Perform cleanup
3. Branch to boot exit address

The following points should be noted for no-boot mode usage:

- No-boot mode is usually selected when the user wants to download the application using an emulator in conjunction with Code Composer Studio (CCStudio). In this usage case, when CCStudio connects to the DSP, it brings the DSP out of the sleep state and PC execution is halted. CCStudio downloads the application image in DSP memory and the start address for execution is set to the application entry point (CCStudio uses the default entry point _c_int00). Application code execution starts when the user invokes the CCStudio Run command.
- The other usage option is a host that downloads the application to L2 (using an emulation protocol like RTDX) and provides an NMI to the DSP so that the application execution can start. The entry point to the application is always the L2 base. The application must be linked so that the entry point for the application is the L2 base (_cint_00 should be mapped to 0x0080000).

3.3.2 I2C Master Mode

The C6457 device I2C master boot process is similar to the C6454/55 device's process, described in Section 2.3.4 and Section 2.3.5, with the following differences:

1. The PLL1 multiplier is set to 16.
2. The C6457 device supports both slave device addresses 0x50 and 0x51
3. Changes in the I2C boot parameter block format.

3.3.2.1 I2C Boot Parameter Structure

Parameter blocks read from the EEPROM override the default settings stored in ROM. In addition, they allow the boot modes to be chained (e.g., I2C master boot followed by I2C master write boot or I2C master boot followed by SRIO boot mode). This gives the user flexibility to choose the settings. BOOTMODE[3:0] pins must be set to I2C master boot mode for a parameter block to be read. It is mandatory that the first block read in I2C master boot mode is a parameter block.

There are three different formats for parameter blocks: I2C, SRIO, and EMAC. These allow the user to customize settings for I2C, SRIO, and EMAC boot modes.

Table 12. I2C Boot Parameter Block Format

Address Offset (bytes)	Name	Value	Default Value (in ROM)
0	Length	Length of the table, in bytes.	30 bytes (size of I2C param block)
2	Checksum	1s complement checksum, 0 for disabled checksum.	0
4	Boot mode	Boot modes as shown in Table 13.	BOOTMODE[3:0] translated to values in Table 13.
6	Port	Physical port number, always 0.	0
8	Software PLL	Software PLL multiply factor, not used.	16
10	Option	0b000 - Boot Parameter mode	0b000 when BOOTMODE[3:0] is I2C master boot mode
		0b001 - Boot Table mode	0b001 when BOOTMODE[3:0] is I2C slave boot mode
		0b010 - Boot Config mode	
		0b011 - Slave boot	
		0b100 - Master broadcast boot	
		0b101-0b111 - Reserved	
12	Device address (low)	I2C data address, LSW.	DEVNUM[3:0] * 0x80
14	Device address (high)	Slave device address on I2C bus address.	0x50 or 0x51 depending on I2C boot mode
16	Broadcast address	In I2C master write mode, the I2C data is sent to this address (after I2C EEPROM read).	0
18	Device ID	The address of this device on the I2C Bus (used for slave boot only).	0x04 when BOOTMODE[3:0] is I2C master boot mode
			CFG[3:0] + 1 when BOOTMODE[3:0] is I2C slave boot mode
20	Core frequency, MHz	The frequency of the CPU core.	800
22	I2C bus frequency, kHz	The desired I2C bus frequency. Used only if the device is an I2C master.	10
24	Next device address (low)	Used only if options specifiy boot configuration mode. The address of the next boot parameter table on the I2C EEPROM.	0
26	Next device address (high)	Slave device address on I2C bus address.	0
28	Address delay	Delay to use (usec) between when the address field is written to the I2C EEPROM and the subsequent data read.	0x200

Table 13. C6457 Extended Boot Mode

Extended Boot Mode Value	Boot Type
0x100	Reserved
0x101	I2C Master
0x102	I2C Slave
0x103	I2C Master Write
0x104	Reserved
0x105	EMAC
0x106	RapidIO
0x107	No boot
0x108	HPI
0x109	EMIFA

3.3.3 I2C Slave Boot Mode

The C6457 device I2C slave boot process is similar to the C6454/55 device's process, described in Section 2.3.5, with the following difference:

1. Device ID, which is I2C device address as a slave, is CFG[3:0] + 1.

3.3.4 HPI Boot Mode

The C6457 device HPI boot process is similar to the C6454/55 device's process, described in Section 2.3.2, with the following differences:

1. The HPI is ROM based instead of completely hardware based.
2. The PLL1 multiplier is set to 16.
3. Direct and boot-table loading are supported.

The boot code allows two bootloading modes for HPI: direct bootloading and boot-table processing. The HPI boot mode is used by the host as follows:

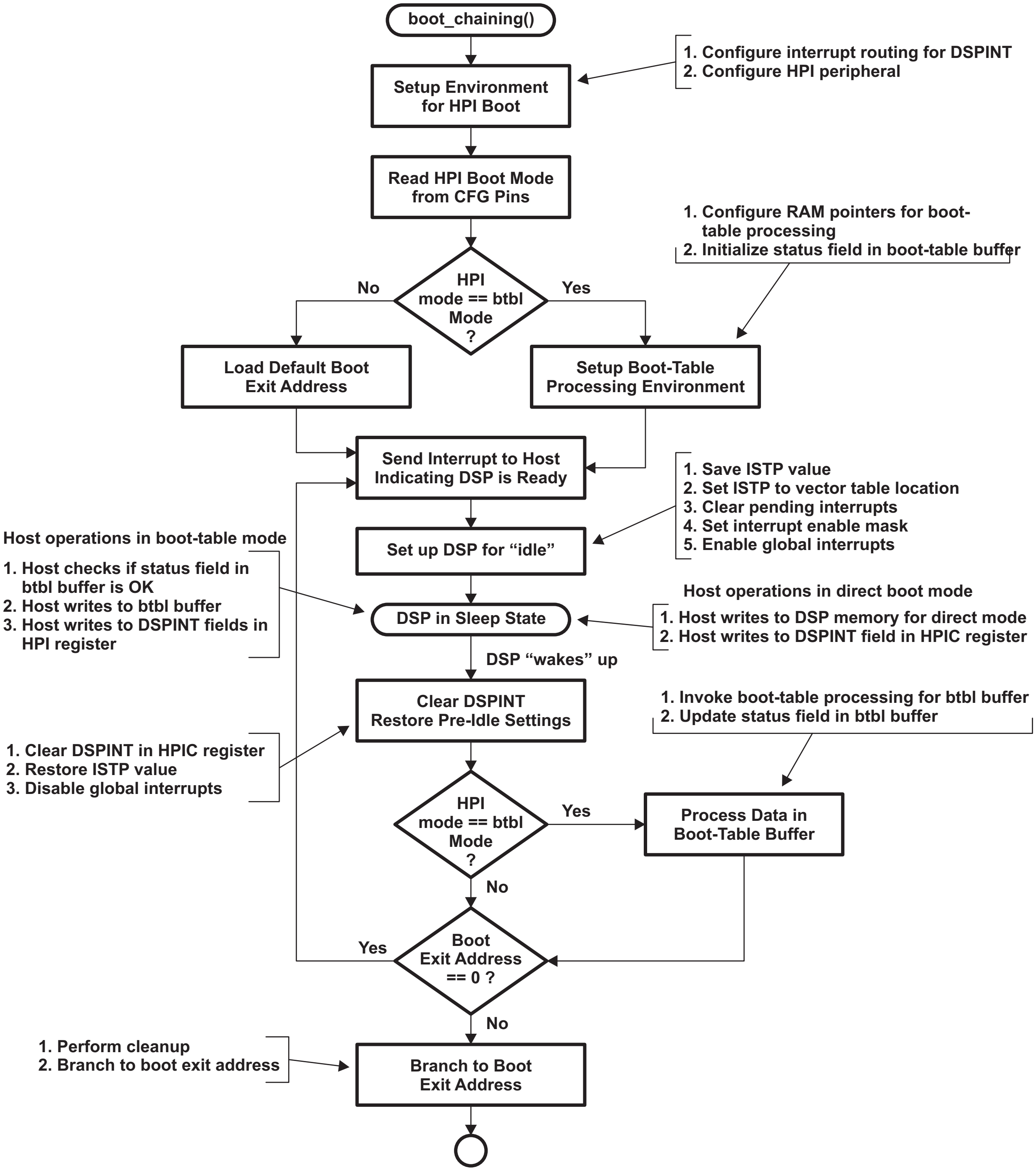
- Direct HPI bootloading mode involves the host making direct write accesses to DSP memory. The DSP de-asserts the HINT line to indicate that it is ready and then enters the sleep state. The host then starts to make direct writes of the application code and data sections to DSP memory. The host can also modify the default entry point of the application (default entry point is 0x800000) by writing to the boot exit address in scratch memory (for details, see Table 10). When the host is finished writing the application to DSP memory, it asserts the DSPINT event. This brings the DSP out of the sleep state and starts execution at the address specified in the boot exit address. Since the HPI can only make 32-bit aligned accesses, the application data and code sections need to be 32-bit aligned.
- HPI boot-table loading mode involves the host writing boot-table blocks to the DSP boot-table buffer located in scratch memory (for details, see Table 10) and the DSP processing the boot-table blocks. The host and DSP follow a handshake procedure in order to ensure that the boot-table blocks are correctly processed. The DSP de-asserts the HINT line to indicate to the host that it is ready, clears the status bit in the boot-table buffer, and then enters the sleep state. The host then writes a boot-table block to the boot-table buffer and asserts the DSPINT line. The DSPINT event wakes the DSP from the sleep state and processes the boot-table block in the boot-table buffer. When processing on the boot-table block is complete, the DSP updates the status field in the boot-table buffer (for status values, see Table 14). The DSP de-asserts the HINT line indicating to the host that it is ready. The host should check the status of the previous block and, if the status is OK, it writes the next boot-table block and the process repeats. The sequence is terminated when the last boot-table block containing the termination sequence is passed to the DSP for processing. The DSP updates the boot exit address to the entry point of the application, updates the status field, performs cleanup, and branches to the boot exit address.

Table 14. Boot-Table Processing Status Values and Enumeration

Status Value	Status Enumeration
0x0000	Previous boot-table block OK.
0x0001	Previous boot-table block checksum failed.
0x0002	Previous boot-table block had invalid header length.

- The boot-table mode requires the DSP to process every boot-table block. Therefore, the direct HPI bootloading mode is faster. However, the boot-table loading mode can accommodate sections that are aligned on any byte boundary.
- The HPI boot mode cannot be customized (using parameter block read using I2C master boot mode) since the boot code does not configure the HPI peripheral.

Figure 3. Boot Code Flow for HPI Boot Mode


boot_chaining()
Setup Environment for HPI Boot
1. Configure interrupt routing for DSPINT
2. Configure HPI peripheral
Read HPI Boot Mode from CFG Pins
1. Configure RAM pointers for boot-table processing
2. Initialize status field in boot-table buffer
HPI mode == btbl Mode ?
No
Yes
Load Default Boot Exit Address
Setup Boot-Table Processing Environment
Send Interrupt to Host Indicating DSP is Ready
1. Save ISTP value
2. Set ISTP to vector table location
3. Clear pending interrupts
4. Set interrupt enable mask
5. Enable global interrupts
Set up DSP for "idle"
Host operations in boot-table mode
1. Host checks if status field in btbl buffer is OK
2. Host writes to btbl buffer
3. Host writes to DSPINT fields in HPI register
DSP in Sleep State
Host operations in direct boot mode
1. Host writes to DSP memory for direct mode
2. Host writes to DSPINT field in HPIC register
DSP "wakes" up
Clear DSPINT Restore Pre-Idle Settings
1. Invoke boot-table processing for btbl buffer
2. Update status field in btbl buffer
1. Clear DSPINT in HPIC register
2. Restore ISTP value
3. Disable global interrupts
HPI mode == btbl Mode ?
Yes
Process Data in Boot-Table Buffer
No
Boot Exit Address == 0 ?
Yes
No
1. Perform cleanup
2. Branch to boot exit address
Branch to Boot Exit Address

3.3.5 EMIFA Boot Mode

The C6457 device EMIFA boot process is similar to the C6454/55 device's process, described in Section 2.3.3, with the following difference:

1. EMIFA is powered up by the boot code by programming the power-sleep controller (PSC).

3.3.6 EMAC Boot Mode

The bootloader configures the EMAC peripheral if it is enabled in bit 5 of Options in the EMAC boot parameter table, opens a transmit and receive channel, configures Rx Communications Port Programming Interface (CPPI), and also routes EMAC Rx interrupt to 4, and Tx interrupt to 5. Then the bootloader transmits an Ethernet-ready frame out if it is enabled in bit 4 of Options. When an EMAC Rx interrupt happens, the bootloader processes the received packet and passes the boot tables into memory. As a result of Efusing, the MAC address is stored in registers 0x02880834 and 0x02880838. When the end of the boot table is reached, it clears the Rx Channel 0 head description pointer to disable reception and exit boots, and jumps to application.

The Ethernet peripheral is configured to accept a combination of a single MAC address and broadcast packets, as defined by the Ethernet boot parameter table. The peripheral rejects packets not matching the MAC addresses selected without a record of the drop.

Local L2 memory is allocated for the received packets. The 16K bytes start from 0x9FA000.

TI DSP Ethernet boot can be used in two modes: default mode and I2C customized mode. Both modes have the same fundamental boot protocols. The default mode is used as a first level bootloader and resides in internal ROM. The I2C customized mode resides in internal ROM, but the MAC parameters reside on the EEPROM and are easily modified, which means you first boot through I2C and then jump to EMAC boot. The I2C customized mode refers to using the EMAC parameter from the I2C EEPROM EMAC parameter table values, which are customized, rather than the values inside the boot ROM.

In boot parameter tables, there is a common field called bootmode. By using the bootmode field, you can first run the I2C master boot, and then jump to other boot modes (see Table 35 and Table 36).

Table 15. EMAC Boot Configuration

EMAC Boot Configuration	Advertised Ability	SGMII Auto-negotiation	Reference Clock (RIOSGMIICLK Pin)	SERDES Line Rate
Master mode	1Gbps, full duplex	Enabled	125 MHz	1.25 Gbps
Slave mode	1Gbps, full duplex	Enabled	125 MHz	1.25 Gbps
Forced mode	1Gbps, full duplex	Disabled	125 MHz	1.25 Gbps

Table 16. C6457 EMAC Boot Parameter Structure

Address Offset (bytes)	Name	Value	Default Value in ROM
0	Length	Length of the table, in bytes.	56 bytes (size of MAC param block)
2	Checksum	1s complement checksum, 0 for disabled checksum.	0
4	Boot mode	Boot modes, as shown in Table 13, should be 0x0105 for MAC boot.	BOOTMODE[4:0] translated to values shown in Table 13
6	Port	Physical port number, always 0.	0
8	Software PLL	Software PLL multiply factor, not used.	16

Table 16. C6457 EMAC Boot Parameter Structure (continued)

Address Offset (bytes)	Name	Value		Default Value in ROM
10	Options	Bits 2:0	110: SGMII mode	110
			Others: reserved	
		Bit 3	0: Full duplex	0
			1: Half duplex	
		Bit 4	0: Send Ethernet ready	0
			1: Suppress Ethernet ready	
		Bit 5	0: Initialize MAC peripheral	0
			1: No peripheral initialization	
		Bit 6	0: Flow control disabled	0
			1: Flow control enabled	
12	MAC addr high	16 most significant bits of MAC address.		E-fuse
14	MAC addr med	16 next most significant bits of MAC address.		E-fuse
16	MAC addr low	16 least significant bits of MAC address.		E-fuse
18	Multi addr high	16 most significant bits of multicast MAC address.		0XFFFF
20	Multi addr med	16 next most significant bits of multicast MAC address.		0XFFFF
22	Multi addr low	16 least significant bits of multicast MAC address.		0XFFFF
24	UDP src port	16 bit UDP source port to accept during boot. 0 indicates accept any source port.		0
26	UDP dest port	Destination UDP port used for Ethernet-ready frame.		9 (discard)
28	Dev ID 12	ASCII characters (digits) specifying device ID.		ASCII 00
30	Dev ID 34	ASCII characters (digits) specifying device ID.		ASCII 0, ASCII CFG[3:0]
32	Host MAC high	16 most significant bits of host MAC address used in Ethernet-ready frame.		0XFFFF
34	Host MAC med	16 next most significant bits of host MAC address.		0XFFFF
36	Host MAC low	16 least significant bits of host MAC address.		0XFFFF
38	CPSGMII configuration	Bit 3:0	CPSGMII configuration index to point to table already stored in the ROM.	Default value in ROM
		Bit 4	0: Use configuration index in bit 3:0	0
			1: Use direct configurations	
		Bit 5	0: Configure CPSGMII	0
			1: Do not configure CPSGMII	
40	CPSGMII control	Control register, bit 15:0.		0x0000
42	CPSGMII MR_ADV_ABILITY	MR_ADV_ABILITY register, bit 15:0.		0x0000
44	CPSGMII TX_CFG high	TX_CFG register, bit 31:16.		0x0000
46	CPSGMII TX CFG low	TX_CFG register, bit 15:0.		0x0000
48	CPSGMII RX_CFG high	RX_CFG register, bit 31:16.		0x0000
50	CPSGMII RX_CFG low	RX_CFG register, bit 15:0.		0x0000
52	CPSGMII AUX_CFG high	AUX_CFG register, bit 31:16.		0x0000
54	CPSGMII AUX_CFG low	AUX_CFG register, bit 15:0.		0x0000

3.3.6.1 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 in such a way that most if not all BOOTP and DHCP servers discard it. The announcement frame is sent only once; no re-transmission 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 = 0TTL = 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
```

3.3.6.2 EMAC Boot Table Frame Format

The Ethernet boot table frame has a format as shown in Table 17.

Table 17. EMAC Boot Table Frame 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)

The boot table frame header has the following format.

Word Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Magic Number (0x544B)															
1	Opcode (0x01)								Sequence Number							

The boot table format is encapsulated in Ethernet frames with IPV4 and UDP headers. The following paragraphs describe the Ethernet frames which are accepted. Frames not matching the criteria specified below are silently discarded and subsequent frames are 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 params is accepted. VLAN fields (other than type/len) are ignored. If 802.3 format MAC format is used, the SNAP/LLC header is verified and skipped. The type field selects IPV4 type (0x0800).
- The IPV4 header validates the Version 4 (IPV6 is not supported), flag and fragment fields, and protocol (UDP) field. The header length field is parsed in order to properly 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 then the source port is accepted. The UDP header length is sanity tested against the (appropriately adjusted) frame length. If the UDP length is too long for the frame or is not a multiple of 2, 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 it increments by one for each processed frame. The sequence number must be allowed to flip over when the maximum is reached, meaning that the sequence number value 0 is expected after reception of sequence number 256.
- The boot table frame payload (which is a multiple of 4 bytes in length) is processed by the boot-table processing function.

3.3.7 SRIO Boot Mode

In the SRIO boot mode, an external host can load code and data directly into the DSP memory while the CPU waits. The code and/or data sections are directly loaded to the desired locations, using the directIO model. When the host has finished loading the application, it signals, through a doorbell interrupt, and the CPU then begins executing at the base of L2. The C6457 device SRIO boot process is similar to the C6454/55 device's process, described in Section 2.3.4 and Section 2.3.5, with the following differences:

1. The PLL1 multiplier is set to 16.
2. The C6457 device supports customized SRIO boot parameters tables through I2C.
3. The prescaler setting (IP_PRESCAL register) for physical layer timers changed from 0x21 to 0x08.
4. The SERDES reference clock and SRIO link rates are different for different SRIO boot configurations (see Table 18).
5. The C6457 device uses CFG[3:0] + 2 as the device ID.
6. SRIO lane 0 must be connected to another SRIO device for SRIO configurations 0-3.
7. Doorbell interrupt routing (register DOORBELL0_ICRR): Routed to INTDST0.

Table 18. C6457 SRIO Boot Configuration

Boot Configuration	Port Configuration	SerDes Reference Clock (RIOSGMIICLK Pin)	Link Rate
SRIO boot configuration 0	4 1x ports	125 MHz	1.25 Gbps
SRIO boot configuration 1	1 4x port	125 MHz	3.125 Gbps
SRIO boot configuration 2	1 4x port	156.25 MHz	1.25 Gbps
SRIO boot configuration 3	1 4x port	156.25 MHz	3.125 Gbps

3.3.7.1 SRIO Boot Parameter Structure

Table 19. C6457 SRIO Boot Parameter Block

Offset	Name	Value		Default Value in ROM
0	Length	Length of the table, in bytes.		24 bytes (size of RIO param block)
2	Checksum	1s complement checksum, 0 for disabled checksum.		0
4	Boot mode	Boot modes, as shown in Table 13, should be 0x0106 for SRIO boot.		BOOTMODE[4:0] translated to values shown in Table 13
6	Port	Physical port number, always 0.		0
8	Software PLL	Software PLL multiply factor, not used.		16
10	Options	Bit 0	0: Transmit disabled	0
			1: Transmit enabled	
		Bit 1	0: Master Mode	0
			1: Boot Table mode	
		Bit 2	0: Configure port	0
			1: Don't configure port	
12	Config index	Base configuration index.		0:1x port, 1:4x port
14	Node ID	8-/16-bit node identification.		CFG[3:0] + 2
16	SERDES reference clock (MHz)	SERDES reference clock frequency.		From Boot mode
18	Link rate (Mbps)	Data link rate (megabits per second).		From Boot mode
20	Packet forward low	Packet forward range low value.		0
22	Packet forward high	Packet forward range high.		0

3.4 C6457 Bootloader Version

The bootloader version can be identified at the address reserved in scratch memory (see Table 10). The bootloader version for silicon v1.0 is *v1.5 Wed Dec 12 14:57:46 2007 i2c mac rapidio*. The bootloader version for silicon v1.2 is *v1.6 Tue Oct 28 11:15:10 2008 i2c mac rapidio*.

There are two known issues in the v1.5 bootloader:

1. The BOOTP packet is not sent out in the EMAC boot mode.
2. In SRIO boot mode when 4x mode falls back to 1x mode in certain hardware configurations, the boot does not operate correctly. The workaround is: after the host sends the application image, the host needs to write an application entry point to 0x9FFFFC and then, send the doorbell interrupt.

These problems have been fixed in the v1.6 bootloader.

4 C6472 Bootloader Operation

This section describes the structure and operation of the C6472 bootloader.

4.1 Bootloader Operation

When the bootloader begins execution, the program performs some initialization of the DSP prior to loading code. Table 20 describes the DSP resources that are configured by the bootloader. Table 21 lists the cache settings while L2 cache is always disabled.

Table 20. C6472 Bootloader Initialization

Resource	Initialization Value
Interrupts	Interrupts are left disabled (GIE=0). For Serial RapidIO and UTOPIA boot interrupts are enabled, with the interrupt mux registers configured to route the Serial RapidIO interrupt to the core. ISTP (the vector table pointer) is set to point into the ROM. After the interrupt arrives the ISTP and interrupt mux registers are restored to their default values. ISTP (the vector table pointer) is set to the base of ROM address, 0x00100000.
Memory	L2 Memory from byte address 0x8969c0 to 0x897fff is reserved for core 0 during the boot and can be reclaimed by application after the boot.
PLL1 Controller	PLL1 setting is based on the configuration pin CFGGP4, refer to
Peripheral Powerup	Peripherals are powered up as required by the boot loader in the selected mode. Peripherals are not powered down when the boot loader exits. I2C is on always on power domain, HPI, UTOPIA, EMAC and SRIO need to be powered on.
Registers	The state of all CPU registers, with the exception of the PC, must be considered random on boot loader exit.

Table 21. C6472 Cache Settings

BOOTMODE[3:0]	Boot Description	L1P/L1D Cache Settings
0	No boot	32 KB
1	HPI	32 KB
From 2 to 14	Other boots	L1P 32 KB, L1D 16 KB
15	Reserved	N/A

After the initialization is performed, the bootloader loads the on-chip RAM according to the boot mode selected, and then causes the DSP to begin execution of the loaded code. At that point, the boot load process is complete. Whenever the DSP is reset, the CPU starts execution of the bootloader again, and the entire boot load process is repeated.

A chip-level register, BOOT_COMPLETE_STAT, in the C6472 device serves two purposes:

1. Controlling the reset release of the individual cores by boot software, boot controller, or host in both global boot and local boot.
2. Signaling the boot process completion. BOOT_COMPLETE_STAT is cleared by $\overline{\text{POR}}$, $\overline{\text{RESET}}$ and other device resets. $\overline{\text{LRESET}}$ **does not** affect BOOT_COMPLETE_STAT. For the BCx bits, writing 1 sets the bit and writing a 0 has no effect.

For global boot after $\overline{\text{POR}}$/$\overline{\text{RESET}}$/other device resets, the output of the 6 LSBs of BOOT_COMPLETE_STAT drives the status of the BOOTACTIVE pin, which indicates when all of the cores have been released from reset. BCx bits indicate when the individual core has been released from reset.

The BOOTACTIVE output is used to indicate all cores have been advanced from the reset state to some operation but does not indicate whether the operation beyond that is good. Additional ways of signaling for all resets ($\overline{\text{POR}}$, $\overline{\text{RESET}}$, $\overline{\text{LRESET}}$, etc.) should be defined by the application such as using GPIOs to communicate to a host that the core(s) has advanced past reset and is ready for the next step or has completed initialization and is ready to receive commands.

For local boot after local reset, BOOT_COMPLETE_STAT may not be as useful as for global boot. When $\overline{\text{LRESET}}$ is applied to a single core there is typically no loading of the memory as is performed when $\overline{\text{POR}}$, $\overline{\text{RESET}}$ and other device resets (except no-boot mode) are applied and there is not an analogous boot active state. When the core that has been given an $\overline{\text{LRESET}}$ is released and begins execution, it executes from previously initialized memory. There are two cases to consider at this point: immediate boot (no boot local mode) and host boot (local mode). For immediate boot, if it is important for the system to know that the core is proceeding normally following an $\overline{\text{LRESET}}$. Then the code that is executed should include some signaling methods such as using a GPIO to indicate that it is executing "normally." This is system dependent as a message on an application interface (like Ethernet) may be more appropriate than a GPIO assertion. For host boot, since the host may want to initialize the local memory of the core before releasing the reset, it is important to understand the reset state by the host when $\overline{\text{LRESET}}$ has been applied to a core. The response to the assertion of the $\overline{\text{LRESET}}$ is given through RESET_STAT by the core. In this case, the host should monitor RESET_STAT through HPI.

The following sections describe the various C6472 boot modes and boot tables in detail.

4.2 Boot Mode Selection

The desired boot mode is selected by setting the three boot mode select pins BOOTMODE[3:0], which are sampled during reset. The BOOTMODE pins are shared with GP[9:6], and configuration pins CFGGP[4:0] are shared with GP[14:10]. CFGGP[4:0] is referred to as CFG[4:0] in Table 22.

Table 22 shows the available boot mode options and their corresponding BOOTMODE pin configurations for the C6472 device.

Table 22. C6472 Boot Mode Selection

BOOTMODE[3:0]	Description	CFG[4:0]	See
0	No boot	None	Section 4.3.2
1	Host (HPI)	None	Section 4.3.2
2	I2C (address 0x50)	4: x9(0), x19(1) 3:0: Boot param index	Section 4.3.3
3	I2C (address 0x51)	4: x9(0), x19(1) 3:0: Boot param index	Section 4.3.3
4	I2C Slave	4: x9(0), x19(1) 3:0:Unused	Section 4.3.3
5	UTOPIA 8 bit Pllx10	Phy ID	Section 4.3.4
6	UTOPIA 8 bit Pllx20	Phy ID	Section 4.3.4
7	UTOPIA 16 bit Pllx10	Phy ID	Section 4.3.4
8	UTOPIA 16 bit Pllx20	Phy ID	Section 4.3.4
9	MAC Port 0	4: x10(0), x20(1) 3:0: Device ID (when RMII is selected, 3: controls speed - 1 for 100 Mbs, 0 for 10 Mbps - and Device ID[3] is 0)	Section 4.3.5
10	MAC Port 1	4: x10(0), x20(1) 3:0: Device ID (when RMII is selected, 3: controls speed - 1 for 100 Mbs, 0 for 10 Mbps - and Device ID[3] is 0)	Section 4.3.5
11	RapidIO boot configuration 0	4: x10(0), x20(1) 3:0: Node (0xf for default)	Section 4.3.6
12	RapidIO boot configuration 1	4: x10(0), x20(1) 3:0: Node (0xf for default)	Section 4.3.6
13	RapidIO boot configuration 2	4: x10(0), x20(1) 3:0: Node (0xf for default)	Section 4.3.6
14	RapidIO boot configuration 4	4: x10(0), x20(1) 3:0: Node (0xf for default)	Section 4.3.6
15	Reserved	N/A	N/A

4.3 Boot Mode Options

4.3.1 No Boot Mode

The C6472 device no boot option is the same as the C6454/55 device's process described in Section 2.3.1.

4.3.2 HPI Boot Mode

If HPI boot is selected after global reset, all C64x+ megamodule cores are internally "held in reset" while the remainder of the device (including all memory subsystems of the C64x+ megamodule) is released from reset. During this period, an external host can initialize the C6472 device memory space (shared memory as well as the C64x+ megamodule memory), as necessary through an HPI interface, including internal configuration registers such as those that control the DDR2 or other peripherals. Once the host is finished with all necessary initialization, it must write a 1 to bit fields BC0 through BC5 of the BOOT_COMPLETE_STAT register (inside the Boot Controller) indicating boot complete of the corresponding C64x+ megamodule. This transition causes the Boot Controller to bring the C64x+ megamodule core out of the "held-in-reset" state. The CPU then begins execution from the internal L2 SRAM address programmed in the DSP_BOOT_ADDRx register (default is base L2). All memory may be written to and read by the host. This allows for the host to verify what it sends to the DSP, if required.

For the C6472 device, only the HPI peripheral can be used for host boot. PLL1, which provides CPU/6 clock to the HPI module, will initially be running in bypass mode. Therefore, the HPI interface will be very slow and $\overline{\text{HRDY}}$ must be observed. Initial HPI accesses can configure PLL1 for full-speed operation to make HPI accesses shorter.

4.3.3 I2C EEPROM Boot Mode

The C6472 device I2C EEPROM boot mode is similar to the C6454/55 device's process, described in Section 2.3.4 and Section 2.3.5, with the following differences:

1. PLL1 is set up to either *9 or *19 mode.
2. The C6472 device supports both I2C address 0x50 and 0x51.
3. Changes in I2C EEPROM common block and I2C boot parameter tables.
4. The C6472 device has 6 cores while the C6454/55 device only has one. Core 0 executes the ROM boot code. Secondary cores can be loaded by using the global address for download. Secondary cores can be booted by setting the execution start address in the boot address registers (one per core), then setting the boot complete bits for the cores to begin execution. The execution address register requires that the execution start be aligned on a 22 bit boundary. Writing to the boot address and boot complete registers can be done using standard boot tables.

Writing to the boot address and boot complete registers can be done using standard boot tables or core 0 to execute a subprogram at the beginning of the application. In Example 1, core 0 sets the boot address for core 1 and core 2, then indicates that boot is complete for core 1 and core 2.

Example 1. Booting With Core 0

```
; Boot complete & boot address registers
BOOT_COMPLETE    .equ 0x02AB0004
DSP_BOOT_ADDR1   .equ 0x02AB0224
DSP_BOOT_ADDR2   .equ 0x02AB0244
DSP_BOOT_ADDR3   .equ 0x02AB0264
DSP_BOOT_ADDR4   .equ 0x02AB0284
DSP_BOOT_ADDR5   .equ 0x02AB02A4

; Boot entry address for core 1 & 2
BOOT_ENTRY_ADDR1 .equ 0x810000
BOOT_ENTRY_ADDR2 .equ 0x820000

        MVKL DSP_BOOT_ADDR1, A1
        MVKH DSP_BOOT_ADDR1, A1
        MVKL BOOT_ENTRY_ADDR1, B1
        MVKH BOOT_ENTRY_ADDR1, B1
              SHR B1,10,B1
        STW  B1, *A1

        MVKL DSP_BOOT_ADDR2, A1
        MVKH DSP_BOOT_ADDR2, A1
        MVKL BOOT_ENTRY_ADDR2, B1
        MVKH BOOT_ENTRY_ADDR2, B1
              SHR B1,10,B1
        STW  B1, *A1

        MVKL BOOT_COMPLETE, A1
        MVKH BOOT_COMPLETE, A1
        MVK  0x07, B1  ; core 0, 1, 2 complete , 0x3f for all 6 cores
        STW  B1, *A1
```

4.3.3.1 I2C EEPROM Data Blocking

Table 23. C6472 I2C EEPROM Common Block Format

Offset (bytes)	Size (bytes)	Name	Value
0	2	Block size	The size of the block including the header
2	2	Checksum	The ones complement checksum, including the block size and checksum fields. Valid checksum values are 0 and -0
4	2	Bootmode	Extended boot mode as shown in Table 24
6	2	Port	Physical port number. Refer to the example used for EMAC boot.
8	2	sw PLL	Software PLL multiplier factor
10 - 126	0 - 118		Data

Table 24. C6472 Extended Boot Mode

Extended Boot Mode Value	Boot Type
0x100	Reserved
0x101	I2C Master
0x102	I2C Slave
0x103	I2C Master Write
0x104	UTOPIA
0x105	MAC
0x106	Serial RapidIO
0x107	Sleep
0x108	HPI

The extended boot modes applies to all boot modes.

4.3.3.2 I2C Boot Parameter Structure

Table 25. C6472 I2C Boot Parameter Table

Offset (bytes)	Name	Value	Default Value (Cold Boot)
10	Option	0b000 - Boot parameter mode 0b001 - Boot table mode 0b010 - Boot configuration mode 0b011 - Slave boot 0b100 - Master broadcast boot 0b101-0b111 - Reserved	0b000
12	Dev addr (low)	I2C data address, LSW	CFG[3:0] *0x80
14	Dev addr (high)	I2C data address, MSW (I2C bus address)	0x50 or 0x51 (from boot mode)
16	Broadcast address	If I2C master mode, the I2C data is sent to this address (after I2C EEPROM read).	0
18	Device ID	The address of this device on the I2C Bus (used for slave boot only)	0x4
20	Core freq MHz	The frequency of the CPU core	50
22	I2C bus freq kHz	The desired I2C bus frequency. Used only if the device is an I2C master.	10
24	Next dev addr (low)	Used only if options specify boot configuration mode. The address of the next boot parameter table on the I2C EEPROM.	0
26	Next dev addr (high)	The most significant word of the next boot parameter table	0
28	Address delay	Delay to use (usec) between when the address field is written to the I2C EEPROM and the subsequent data read	0x200

4.3.4 UTOPIA Boot Mode

The bootloader configures the port, setup timer and then sleep until interrupt. Check for cell, if a cell is found then send the payload to boot table process, if end of boot table reached, disable all Rx PDMA channels, disable timer and then exit boot loader.

If hardware configuration is enabled in boot params, the peripheral is configured as a UTOPIA slave (which is the only option available) and big Endian. All received cells are accepted and routed to PDMA channel 8. PDMA channel 8 routes the cells to the 16 kByte of addressable memory in L1D. The actual number of cells which can be buffered varies by the cell size parameter in the boot parameter table, and ranges from 256 to 292 cells. UTOPIA PDMA interrupts are routed to the DSP at full and half buffer marks. The DSP configures local timer 0 to interrupt the DSP every 50 μs for 16-bit mode, 100 μs for 8-bit mode. The timer counters are based on the boot parameter field core_freq_MHz.

Each UTOPIA cell contains 32 byte payload, immediately followed by the 32-bit magic value 0x54492164. The cell is searched from the end of the cell payload towards the start. Cells which do not have the magic number are discarded (and counted). The payload is passed to the boot table processing function.

4.3.4.1 UTOPIA Boot Parameter Structure

Table 26. UTOPIA Boot Parameter Table

Offset (bytes)	Name	Value		Default
10	Options	Bit 0	0 - multi PHY 1 - single PHY	0
		Bit 1	0 - 8 bit UTOPIA 1 - 16 bit UTOPIA	From boot mode
		Bit 2	0 - initialize Port 1 - Skip port initialization	0
12	Cell size (bytes)	Physical cell size		53 (8 bit port) 54 (16 bit port)
14	Bus Width (bits)	UTOPIA bus width		From boot mode
16	Slave ID	Slave ID used in multi PHY mode		CFG[4:0]
18	Core freq (MHz)	Core frequency (after PLL configuration)		500

4.3.5 EMAC Boot Mode

The bootloader configures the EMAC peripheral if it is enabled in bit 5 of Options in the EMAC boot parameter table, opens a transmit and receive channel, configures Rx Communications Port Programming Interface (CPPI), and also routes EMAC Rx interrupt to 4, and Tx interrupt to 5. Then the bootloader transmits an Ethernet-ready frame out if it is enabled in bit 4 of Options. When an EMAC Rx interrupt happens, the bootloader processes the received packet and passes the boot tables into memory. As a result of Efusing, the MAC address is stored in registers 0x02A8 0700 and 0x02A8 0704. When the end of the boot table is reached, it clears the Rx Channel 0 head description pointer to disable reception and exit boots, and jumps to application.

The Ethernet peripheral is configured to accept a combination of a single MAC address and broadcast packets, as defined by the Ethernet boot parameter table. The peripheral rejects packets not matching the MAC addresses selected without a record of the drop.

The boot code configures the selected MAC port as determined by the macsel pins read from the devstat register.

Local L1D memory is allocated for the received packets. The 16K bytes start from 0xf00000 are divided into 10 packet buffers, each of size 1600 bytes. The CPPI is allocated in the dedicated CPPI memory area for the port.

TI DSP Ethernet boot can be used in two modes: default mode and I2C customized mode. Both modes have the same fundamental boot protocols. The default mode is used as a first level bootloader and resides in internal ROM. The I2C customized mode resides in internal ROM, but the MAC parameters reside on the EEPROM and are easily modified, which means you first boot through I2C and then jump to EMAC boot. The I2C customized mode refers to using the EMAC parameter from the I2C EEPROM EMAC parameter table values, which are customized, rather than the values inside the boot ROM.

In boot parameter tables, there is a common field called bootmode. By using the bootmode field, you can first run the I2C master boot, and then jump to other boot modes (see Table 23 and Table 24).

4.3.5.1 EMAC Boot Parameter Structure

Table 27. C6472 EMAC Boot Parameter Table

Offset (bytes)	Name	Value		Default
10	Options	Bits 2:0	000: MII mode	
			001: RMII mode auto-negotiation	
			010: GMII mode	From MACSEL
			011: RGMII mode	
			100: SMII mode	
			101: S3MII mode	
			110: RMII mode 10 Mbps	
			111: RMII mode 100 Mbps	
		Bit 3	0: Full duplex	0
			1: Half duplex	
		Bit 4	0: Send Ethernet ready	0
			1: Suppress Ethernet ready	
		Bit 5	0: Initialize MAC peripheral	0
			1: No peripheral initialization	
		Bit 6	0: Flow control disabled	0
			1: Flow control enabled	
12	MAC addr high	16 Most significant bits of MAC address[1]		e-fuse
14	MAC addr med	16 next most significant bits of MAC address		e-fuse
16	MAC addr low	16 least significant bits of MAC address		e-fuse
18	Multi addr high	16 most significant bits of multicast MAC address[2]		0xffff
20	Multi addr med	16 next most significant bits of multicast MAC address		0xffff
22	Multi addr low	16 least significant bits of multicast MAC address		0xffff
24	UDP src port	16 bit UDP source port to accept during boot. 0 indicates to accept any source port		0
26	UDP dest port	Destination UDP port used for Ethernet-ready frame		9 (discard)
28	Dev id 12	ASCII characters (digits) specifying device id		ASCII "00"
30	Dev id 34	ASCII characters (digits) specifying device id		ASCII "0", ASCII cfg[3:0]
32	Host MAC high	16 most significant bits of host MAC address used in Ethernet-ready frame		0xffff
34	Host MAC med	16 next most significant bits of host MAC address		0xffff
36	Host MAC low	16 least significant bits of host MAC address		0xffff

[1] A value of 0 for MAC address indicates that the e-fuse value is used.

[2] A value of 0 for Multi cast address means the e-fuse multicast value is used (if available).

4.3.5.2 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 in such a way that most if not all BOOTP and DHCP servers discard it. The announcement frame is sent only once; no re-transmission 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 = 0TTL = 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
```

4.3.5.3 EMAC Boot Table Frame Format

The Ethernet boot table frame has a format as shown in Table 28.

Table 28. EMAC Boot Table Frame 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)

The boot table frame header has the following format.

Word Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Magic Number (0x544B)															
1	Opcode (0x01)								Sequence Number							

The boot table format is encapsulated in Ethernet frames with IPV4 and UDP headers. The following paragraphs describe the Ethernet frames which are accepted. Frames not matching the criteria specified below are silently discarded and subsequent frames are 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 params is accepted. VLAN fields (other than type/len) are ignored. If 802.3 format MAC format is used, the SNAP/LLC header is verified and skipped. The type field selects IPV4 type (0x0800).
- The IPV4 header validates the Version 4 (IPV6 is not supported), flag and fragment fields, and protocol (UDP) field. The header length field is parsed in order to properly 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 then the source port is accepted. The UDP header length is sanity tested against the (appropriately adjusted) frame length. If the UDP length is too long for the frame or is not a multiple of 2, 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 it increments by one for each processed frame. The sequence number must be allowed to flip over when the maximum is reached, meaning that the sequence number value 0 is expected after reception of sequence number 256.
- The boot table frame payload (which is a multiple of 4 bytes in length) is processed by the boot-table processing function.

4.3.6 SRIO Boot Mode

The C6472 device SRIO boot mode is similar to the C6454/55 device's process, described in Section 2.3.6, with the following differences:

1. PLL1 is set up to either *10 or *20 mode.
2. Peripheral is configured as two serial ports, 1X mode.
3. The C6472 device supports customized SRIO boot parameter tables through I2C.
4. Prescaler setting (Register IP_PRESCAL) for physical layer timers changed from 0x21 to 0x08.
5. The C6472 device uses CFG[3:0] + 2 as the device ID, but the C6454/55 device uses BOOTMODE[2]:CFG[2:0] + 2 as the device ID.
6. The C6472 device has 6 cores while the C6454/55 device has only one. Core 0 executes the ROM boot code. Secondary cores can be loaded by using the global address for download. Secondary cores can be booted by setting the execution start address in the boot address registers (one per core), then setting the boot complete bits for the cores to begin execution. The execution address register requires that the execution start be aligned on a 22 bit boundary. Writing to the boot address and boot complete registers can be done using standard boot tables.
7. Doorbell interrupt routing (register DOORBELL0_ICRR): Routed to INTDST0.

Table 29 lists the required SerDes reference clock and associated link rate settings.

Table 29. C6472 SRIO Boot Configurations

Boot Configuration	SerDes Reference Clock	SRIO Link Rate
SRIO boot configuration 0	125 MHz	1.25 Gbps
SRIO boot configuration 1	125 MHz	3.125 Gbps
SRIO boot configuration 2	156.25 MHz	1.25 Gbps
SRIO boot configuration 3	156.25 MHz	3.125 Gbps

4.3.6.1 SRIO Boot Parameter Structure

Table 30. C6472 SRIO Boot Parameter Table

Offset (bytes)	Name	Value		Default
10	Options	Bit 0	0: Transmit disabled	0
			1: Transmit enabled	
		Bit 1	0: Master Mode	0
			1: Reserved	
		Bit 2	0: Configure port	0
			1: Don't configure port	
12	Configuration index	Base configuration index		From boot mode
14	Node ID	8/16 bit node identification		CFG[3:0] + 2
16	SERDES Ref Clock (MHz*100)	SERDES reference clock frequency in units of hundredths of megahertz; i.e., a value of 1 MHz would have a value of 100.		From boot mode
18	Link rate (Mbps)	Data link rate (mega bits per second)		From boot mode
20	Packet forward low	Packet forward range low value		0
22	Packet forward high	Packet forward range high		0

4.4 C6472 Bootloader Version

The bootloader version can be identified at byte address 0x0010 9190. The signature for C6472 silicon revision 1.0 is *v3.10 Thu Nov 16 16:17:31 2006 i2c mac utopia rapidio*.

5 C6474 Bootloader Operation

This section describes the structure and operation of the C6474 bootloader.

5.1 Bootloader Initialization

When the bootloader begins execution, the program performs some initialization of the DSP prior to loading code. Table 31 describes the DSP resources that are configured by the bootloader. Table 32 lists the cache settings while L2 cache is always disabled.

Table 31. C6474 Bootloader Initialization

Resource	Initialization Value
Interrupts	Interrupts are left disabled (GIE=0). For Serial RapidIO boot interrupts are enabled, with the interrupt mux registers configured to route the RapidIO interrupt to the core. ISTP (the vector table pointer) is set to point into the ROM. After the interrupt arrives the ISTP and interrupt mux registers are restored to their default values. ISTP (the vector table pointer) is set to the L3 ROM base address, 0x3C000000.
Memory	L2 Memory from byte address 0x10880000 to 0x10885FFF and 0x108FFFF8 to 0x108FFFFF is reserved for core 0 during the boot and can be reclaimed by application after the boot.
PLL1 Controller	For all bootmodes, the PLL is set to x16 mode.
Peripheral Powerup	Peripherals are powered up as required by the boot loader in the selected mode. Peripherals are not powered down when the boot loader exits. I2C and EMAC are on always on power domain, only SRIO need to be powered on.
Registers	The state of all CPU registers, with the exception of the PC, must be considered random on boot loader exit.

Table 32. C6474 Cache Settings

BOOTMODE[3:0]	Boot Description	L1P/L1D Cache Settings
0	No boot	32 KB
From 1 to 6	I2C and EMAC boots	32 KB
7	Reserved	N/A
From 8 to 11	SRIO boots	32 KB
From 12 to 15	Reserved	N/A

After the initialization is performed, the bootloader loads the on-chip RAM according to the boot mode selected, and then causes the DSP to begin execution of the loaded code. At that point, the boot load process is complete. Whenever the DSP is reset, the CPU starts execution of the bootloader again, and the entire boot load process is repeated.

L2 is configured as RAM only (no cache). The boot code uses a portion of L2 during the boot process (see the shaded area in Table 33). When the boot is complete, the application can reclaim this reserved memory.

Table 33. L2 Memory Map for Bootloader Code

Memory Range	Size (bytes)	Description
0x10880000 - 0x10883FFF	16K	Allocated for storing received EMAC packets. In addition, it is used to store SRIO boot packets when boot table processing is selected for these modes.
0x10884000 - 0x108847FF	2K	Allocated as stack by boot code.
0x10884800 - 0x108848FF	256	Used by boot code to store boot parameters.
0x10884900 - 0x10884AFF	512	Allocated for un-initialized data structures used by the boot code.
0x10884B00 - 0x10885AFF	4K	Used for boot table processing by the boot code.
0x10885B00 - 0x10885BFF	256	Boot version string.
0X10885C00 - 0x10885DFF	512	Boot statistics/diagnostics.
0X10885E00 - 0x10885EFF	256	Boot progress.
0x10885F00 – 0x10885F0F	16	Boot ROM table pointers.
0x108FFFF8 - 0x108FFFFB	4	Flag for core 0 to bring core 1 and 2 out of reset.
0X108FFFFC - 0x108FFFFF	4	Application entry point (boot exit address).

5.2 Multicore Considerations of Bootloading

The C6474 device contains three C64x+ Megamodule cores, and during boot, Core 0 begins executing first after reset and keeps Core 1 and Core 2 in reset. Following reset, Core 0 begins executing from the L3 ROM base address, and is responsible for performing the boot process (e.g., from I2C ROM, Ethernet, or RapidIO), after which Core 0 brings the other C64x+ Megamodule cores out of reset by setting to 1 the EVTPULSE4 bit (bit 4) of the C64x+ Megamodule Core 0's EVTASRT register. This process is valid only once: writing 1, then writing 1 again will not bring Core 1 and 2 out of reset again. Then, the C64x+ Megamodule Core 0 begins execution from the entry address defined in the boot table. Core 1 and 2 begin execution from their L2 RAMs' base address, 0x800000. If the content in 0x800000 for Core 1 and Core 2 is zero, Core 0 puts in IDLE code instead to make sure Core 1 and 2 stay in IDLE when those cores are released out of reset.

5.3 Boot Mode Selection

The desired boot mode is selected by setting the three boot mode select pins BOOTMODE[3:0], which are sampled during reset.

Table 34 shows the available boot mode options and their corresponding BOOTMODE pin configurations for the C6474 device.

Table 34. C6474 Boot Mode Selection

BOOTMODE[3:0]	Description	DEVNUM[3:0]	See
0	No boot	None	Section 5.4.1
1	I2C (address 0x50)	Boot param index	Section 5.4.2
2	I2C (address 0x51)	Boot param index	Section 5.4.2
3	I2C Slave	None	Section 5.4.2
4	EMAC (Master)	Device ID	Section 5.4.3
5	EMAC (Slave)	Device ID	Section 5.4.3
6	EMAC (Forced Mode)	Device ID	Section 5.4.3
7	Reserved	N/A	-
8	RapidIO boot configuration 0	3:0: Node (0xf for default)	Section 5.4.4
9	RapidIO boot configuration 1	3:0: Node (0xf for default)	Section 5.4.4
10	RapidIO boot configuration 2	3:0: Node (0xf for default)	Section 5.4.4
11	RapidIO boot configuration 3	3:0: Node (0xf for default)	Section 5.4.4
12 to 15	Reserved	N/A	-

5.4 Boot Mode Options

5.4.1 No Boot

The C6474 device no boot option is the same as the C6454/55 device's process described in Section 2.3.1.

5.4.2 I2C EEPROM Boot Mode

The boot process is similar to the C6454/55 process, described in Section 2.3.4 and Section 2.3.5, with the following differences:

1. PLL1 is set up to *16 mode.
2. The C6474 device supports both I2C address 0x50 and 0x51.
3. Changes in I2C EEPROM common block and I2C boot parameter tables.
4. The C6474 device has 3 cores while the C6454/55 devices only have one. For the C6474 device, core 0 executes the ROM boot code, secondary cores can be loaded by using the global address for download. At the end of boot, for the silicon revision 1.x device, core 0 always releases core 1 and 2 out of reset; for the silicon revision 2.x device, the bootloader checks address 0x108FFFF8, if it is 0 (the default value), core 0 releases core 1 and 2 out of reset, if it has no 0 values, core 0 does not take core 1 and core 2 out of reset. Afterwards, core 0 runs from its base of L2 RAM. Core 1 and core 2 run from the base address of L2 RAM (i.e., 0x800000), if they are taken out of reset. If the content in 0x800000 for core 1 or 2 is zero; core 0 puts IDLE code in core 1 or core 2 local L2 base addresses to make sure core 1 or 2 stay in IDLE when that core is released out of reset.

5.4.2.1 I2C EEPROM Data Blocking

All data stored on the I2C EEPROM are stored in blocks. Each block has a maximum length of 128 bytes, including the 10 byte block header. Table 35 shows the format of the block.

Table 35. C6474 I2C EEPROM Common Block Format

Offset (bytes)	Size (bytes)	Name	Value
0	2	Block size	The size of the block including the header.
2	2	Checksum	The ones complement checksum, including the block size and checksum fields. Valid checksum values are 0 and -0.
4	2	Bootmode	Extended boot mode as shown in Table 36
6	2	Port	Physical port number, always 0 for C6474.
8	2	sw PLL	Software PLL multiplier factor. Ignored, and multiplier is always set to x16.
10 - 126	0 - 118		Data

The bootloader reads data from the I2C EEPROM in blocks. If the checksum shows a failure, the block is re-read until the checksum is valid. A value of 0 in the checksum field disables the checksum check.

Table 36. C6474 Extended Boot Mode

Extended Boot Mode Value	Boot Type
0x100	Reserved
0x101	I2C Master
0x102	I2C Slave
0x103	I2C Master Write
0x104	Reserved
0x105	EMAC
0x106	Serial RapidIO

The extended boot modes applies to EMAC and SRIO boot as well.

5.4.2.2 I2C Boot Parameter Structure

The I2C boot sequence begins with the DSP reading a block of boot parameters from the I2C EEPROM. The boot parameter table complies with I2C EEPROM data blocking format. The boot parameters begin at EEPROM address 0, and a boot parameter block consists of 128 bytes. The DSP calculates the address of boot parameter block to load based on the value of the DEVNUM[3:0] bits of the Device Status Register (DEVSTAT) as follows: address = 0x80 * DEVNUM[3:0]. This allows the DSP to read one of eight possible boot parameter blocks. The values in this block determine how the boot process proceeds. Table 37 shows the structure of these boot parameters. Each value is 2 bytes, and the bytes must be stored in big endian format (most significant byte at the lowest address), regardless of the endianness setting of the processor. The structure has a total length of 30 bytes.

Table 37. C6474 I2C Boot Parameter Table

Offset (bytes)	Name	Value	Default Value (Cold Boot)
10	Option	0b000 - Boot Parameter mode 0b001 - Boot Table mode 0b010 - Boot Configuration mode 0b011 - Slave boot 0b100 - Master broadcast boot 0b101-0b111 - Reserved	0b000
12	Dev addr (low)	I2C data address, LSW	DEVNUM[3:0]*0x80
14	Dev addr (high)	I2C data address, MSW (I2C bus address)	0x50 or 0x51 (from boot mode)
16	Broadcast address	If I2C master mode, the I2C data is sent to this address (after I2C EEPROM read).	0
18	Device ID	The address of this device on the I2C Bus (used for slave boot only)	0x4
20	Core freq MHZ	The frequency of the CPU core	800
22	I2C bus freq kHz	The desired I2C bus frequency. Used only if the device is an I2C master.	10

Table 37. C6474 I2C Boot Parameter Table (continued)

Offset (bytes)	Name	Value	Default Value (Cold Boot)
24	Next dev addr (low)	Used only if options specify boot configuration mode. The address of the next boot parameter table on the I2C EEPROM.	0
26	Next dev addr (high)	The most significant word of the next boot parameter table.	0
28	Address delay	Delay to use (usec) between when the address field is written to the I2C EEPROM and the subsequent data read.	0x200

The three LSBs of the option field (offset 10 bytes) define what the bootloader expects to find in the I2C EEPROM at the offset specified by the Device Address (LSW), and how it should proceed. A value of 0 indicates another boot parameter table. A value of 1 indicates a boot table (i.e., a table which contains initialized code and data sections, see Section 6.2.2). A value of 2 indicates a boot configuration table (table used to configure registers, see Section 6.2.3). A value of 3 indicates slave boot and a value of 4 indicates master broadcast mode.

After the DSP reads and remembers the boot parameter table, the bootloader performs a boot re-entry. On this pass the code executes based on the values provided by the boot parameter table.

If the options indicate a boot table is loading, then the bootloader reads from the I2C EEPROM address specified in the boot table until the end of the table is reached (i.e., until all code and data sections are loaded), and immediately begin execution of the loaded code by branching to the entry point specified at the beginning of the boot table (see Section 6.2.2.1).

If the options indicate a boot configuration table is loading, then the bootloader reads from the specified I2C EEPROM address until the end of the boot configuration table. This typically initializes various registers (see Section 6.2.3). The next device address previously read from the boot parameter block is copied into device address, the boot options are cleared, and the bootloader performs a boot re-entry. This directs the ROM to read the I2C EEPROM at the specified device address, which should have another boot parameter table.

5.4.3 EMAC Boot Mode

The EMAC boot process is similar to the C6457 process, described in Section 3.3.6, with the following differences:

1. PLL1 is set up to *16 mode.
2. The C6474 device supports SGMII only and has a different interrupt handler.
3. Changes in EMAC boot parameter tables.
4. The C6474 device allocates 16K bytes start from L2 memory 0x10880000 received packets.
5. The C6474 device has only one EMAC port.
6. The C6474 device has 3 cores while the C6457 device only has one and the boot controller is different. For the C6474 device, core 0 executes the ROM boot code, secondary cores can be loaded by using the global address for download. At the end of boot, for the silicon revision 1.x device, core 0 always releases core 1 and 2 out of reset; for the silicon revision 2.x device, the bootloader checks address 0x108FFFF8, if it is 0 (the default value), core 0 releases core 1 and 2 out of reset, if it has no 0 values, core 0 does not take core 1 and core 2 out of reset. Afterwards, core 0 runs from its base of L2 RAM. Core 1 and core 2 run from the base address of L2 RAM (i.e., 0x800000), if they are taken out of reset. If the content in 0x800000 for core 1 or 2 is zero; core 0 puts IDLE code in core 1 or core 2 local L2 base addresses to make sure core 1 or 2 stay in IDLE when that core is released out of reset.
7. As a result of Efusing, the MAC address is stored in register 0x0288 0834 and 0x0288 0838 instead.

5.4.3.1 EMAC Boot Parameter Structure

Table 38. C6474 EMAC Boot Parameter Table

Offset (bytes)	Name	Value		Default
10	Options	Bits 2:0	110: SGMII mode	110
			Others: Reserved	
		Bit 3	0: Full duplex	0
			1: Half duplex	
		Bit 4	0: Send Ethernet ready	0
			1: Suppress Ethernet ready	
		Bit 5	0: Initialize MAC peripheral	0
			1: No peripheral initialization	
		Bit 6	0: Flow control disabled	0
			1: Flow control enabled	
12	MAC addr high	16 Most significant bits of MAC address[1]		e-fuse
14	MAC addr med	16 next most significant bits of MAC address		e-fuse
16	MAC addr low	16 least significant bits of MAC address		e-fuse
18	Multi addr high	16 most significant bits of multicast MAC address[2]		0xffff
20	Multi addr med	16 next most significant bits of multicast MAC address		0xffff
22	Multi addr low	16 least significant bits of multicast MAC address		0xffff
24	UDP src port	16 bit UDP source port to accept during boot. 0 indicates to accept any source port		0
26	UDP dest port	Destination UDP port used for Ethernet-ready frame		9 (discard)
28	Dev id 12	ASCII characters (digits) specifying device id		ASCII "00"
30	Dev id 34	ASCII characters (digits) specifying device id		ASCII "0", ASCII DEVNUM[3:0]
32	Host MAC high	16 most significant bits of host MAC address used in Ethernet-ready frame		0xffff
34	Host MAC med	16 next most significant bits of host MAC address		0xffff
36	Host MAC low	16 least significant bits of host MAC address		0xffff

[1] A value of 0 for MAC address indicates that the e-fuse value is used.

[2] A value of 0 for Multicast address means the e-fuse multicast value is used (if available).

Table 38. C6474 EMAC Boot Parameter Table (continued)

Offset (bytes)	Name	Value		Default
38	CPSGMII Config	Bit 3:0	CPSGMII configuration index to point to table already stored in the ROM.[3]	From boot table
		Bit 4	0: Use configuration index in bit 3:0	0
			1: Use direct configurations	
		Bit 5	0: Configure PSGMII	0
			1: Don't configure CPSGMII	
40	CPSGMMI control	Control register, bit 15:0		0x0000
42	CPSGMII Mr_Adv_Ability	Mr_Adv_Ability register, bit 15:0		0x0000
44	CPSGMII Tx_Cfg high	TX_Cfg register, bit 31:16		0x0000
46	CPSGMII Tx_Cfg low	Tx_Cfg register, bit 15:0		0x0000
48	CPSGMII Rx_Cfg high	Rx_Cfg register, bit 31:16		0x0000
50	CPSGMII Rx_Cfg low	Rx_Cfg register, bit 15:0		0x0000
52	CPSGMII Aux_Cfg high	Aux_Cfg register, bit 31:16		0x0000
54	CPSGMII Aux_Cfg low	Aux_Cfg register, bit 15:0		0x0000

[3] The following table has been defined:

```
const sgmiiConfig_t sgmiiConfigTbl[DEVICE_SGMII_N_CFG_TABLES] = {
  {
     0x00000021, /* Control: enable auto-negotiation and master mode */
     0x00009801, /* Mr_Adv_Ability: advertise fullduplex gigabit */
     0x00000a21, /* Tx_Cfg */
     0x00081021, /* Rx_Cfg */
     0x0000000b /* Aux_Cfg */
  },
  {
     0x00000001, /* Control: enable auto-negotiation and slave mode */
     0x00000001, /* Mr_Adv_Ability: MAC to PHY Configuration reg */
     0x00000a21, /* Tx_Cfg */
     0x00081021, /* Rx_Cfg */
     0x0000000b /* Aux_Cfg */
  },
  {
     0x00000020, /* Control: force link, no auto-negotiation */
     0x00009801, /* Mr_Adv_Ability: advertise fullduplex gigabit */
     0x00000a21, /* Tx_Cfg */
     0x00081021, /* Rx_Cfg */
     0x0000000b /* Aux_Cfg */
  }
 };
```

The EMAC master boot uses table 0, slave boot uses table 1, and force mode uses table 2.

5.4.4 SRIO Boot Mode

In the SRIO boot mode, an external host can load code and data directly into the DSP memory while the CPU waits. The code and/or data sections are directly loaded to the desired locations, using the directIO model. When the host has finished loading the application, it signals through a doorbell interrupt and the CPU then begins executing at the base of L2. The C6474 device SRIO boot process is similar to the C6454/55 device's process, described in Section 2.3.6, with the following differences:

1. PLL1 is set up to *16 mode.
2. The peripheral is configured as two serial ports, 1X mode.
3. The C6474 device supports customized SRIO boot parameter tables through I2C.
4. Prescaler setting (register IP_PRESCAL) for physical layer timers changed from 0x21 to 0x08.

5. The C6474 device has 3 cores while the C6454/55 devices only have one. For the C6474 device, core 0 executes the ROM boot code, secondary cores can be loaded by using the global address for download. At the end of boot, for the silicon revision 1.x device, core 0 always releases core 1 and 2 out of reset; for the silicon revision 2.x device, the bootloader checks address 0x108FFFF8, if it is 0 (the default value), core 0 releases core 1 and 2 out of reset, if it has no 0 values, core 0 does not take core 1 and core 2 out of reset. Afterwards, core 0 runs from its base of L2 RAM. Core 1 and core 2 run from the base address of L2 RAM (i.e., 0x800000), if they are taken out of reset. If the content in 0x800000 for core 1 or 2 is zero; core 0 puts IDLE code in core 1 or core 2 local L2 base addresses to make sure core 1 or 2 stay in IDLE when that core is released out of reset.
6. The recommended SERDES reference clock and SRIO link rate is different.
7. The C6474 device uses DEVNUM[3:0] + 2 as the default node ID when DEVNUM[3:0] is not 0xF. When DEVNUM[3:0] is 0xF, then the default node ID is either 0xFF or 0xFFFF.
8. Doorbell interrupt routing (register DOORBELL0_ICRR): Routed to INTDST0.

Table 39 lists the required SERDES reference clock and associated link rate settings.

Table 39. C6474 SRIO Boot Configurations

Boot Configuration	SerDes Reference Clock	SRIO Link Rate
SRIO boot configuration 0	125 MHz	1.25 Gbps
SRIO boot configuration 1	125 MHz	3.125 Gbps
SRIO boot configuration 2	156.25 MHz	1.25 Gbps
SRIO boot configuration 3	156.25 MHz	3.125 Gbps

TI DSP SRIO boot can be used in two modes: default mode and I2C customized mode. Both modes have the same fundamental boot protocols. The default mode is used as a first-level bootloader and resides in internal ROM. The I2C customized mode resides in internal ROM, but the SRIO parameters reside on the EEPROM and are easily modified, which means you first boot through I2C and then jump to SRIO boot. The I2C customized mode refers to using the SRIO parameter from the I2C EEPROM SRIO parameter table values, which are customized, rather than the values inside the boot ROM. In boot parameter tables, there is a common field called bootmode. By using the bootmode field, you can first run the I2C master boot, and then jump to other boot modes (see Table 35 and Table 36).

5.4.4.1 SRIO Boot Parameter Structure

Table 40. C6474 SRIO Boot Parameter Table

Offset (bytes)	Name	Value		Default
10	Options	Bit 0	0: Transmit disabled	0
			1: Transmit enabled	
		Bit 1	0: Master Mode	0
			1: Boot Table Mode	
		Bit 2	0: Configure port	0
			1: Don't configure port	
12	Configuration index	Base configuration index		From boot mode
14	Node ID	8/16 bit node identification		DEVNUM[3:0] + 2
16	SERDES Ref Clock	SERDES reference clock frequency in MHz		From boot mode
18	Link rate (Mbps)	Data link rate (mega bits per second)		From boot mode
20	Packet forward low	Packet forward range low value		0
22	Packet forward high	Packet forward range high		0

5.5 C6474 Bootloader Version

The bootloader version can be identified at byte address 0x3C0075F0 from ROM. The signatures for C6474 silicon revisions 1.2, 1.3, and 2.1 are shown in Table 41.

Table 41. Differences Between C6474 Bootloader Versions

	Silicon Revision	
	1.2 and 1.3	**2.1**
Signature	v1.5 Tue Feb 27 13:47:44 2007 i2c mac rapidio	v1.7 Fri Aug 08 11:57:28 2008 i2c mac rapidio
Signature Address	0x3C0075F0	0x3c006780
Big Endian I2C boot	Working	Working
BOOTP packet in the EMAC boot	Ethernet-ready announcement packet was sent out after link is up.	Ethernet-ready announcement packet is sent out after link is up also after correct PLL lock status of SGMII register is obtained.
I2C Slave boot	Won't work	Working
Core 1 and core 2 reset control	Core 0 takes core 1 and core 2 out of reset at the end of boot.	Adding options so that core 0 does not take core 1 and core 2 out of reset after boot. If address 0x108FFFF8 has non-zero values, then core 0 does not take core 1 and core 2 out of reset at the end of boot. Otherwise, core 0 takes core 1 and core 2 out of reset at the end of boot.
SRIO DEV_ID	Default values	Programmed SRIO register offset 0x1000 and 0x1004 so that SRIO host can differentiate different endpoints during system exploration.

6 Creating Boot Images

6.1 Host/PCI Boot

If the DSP is configured for Host or PCI boot, the host is expected to load code and initialized data directly into the DSP's memory space, and to send an interrupt to the DSP indicating the completion of the load. This causes the on-chip bootloader to transfer program control to the base of L2.

The application image which is to be loaded by the host is typically in a C header format. For each section, the following information exists:

- Destination address in the DSP's memory space.
- Length of the section.
- Actual section data, in a form of an initialized array.

The method of creating such an image, starting with an .out file, is described in *Using OFD Utility to Create a DSP Boot Image* (SPRAA64). The method involves using the hex6x and ofd6x tools, which are part of the C6000 code generation tool suite.

6.1.1 PCI Auto-Initialization

As previously discussed in , if PCI auto-initialization through EEPROM is enabled (PCI_EEAI configuration pin), then it is also necessary to program the required initialization values into an I2C EEPROM. This section of I2C EEPROM is called the PCI Configuration Block. It starts at I2C EEPROM byte address 0x400. describes the memory map for the I2C EEPROM.

6.2 I2C Boot

In the I2C EEPROM boot process, as described in Section 2.3.4, the bootloader starts by reading one of the eight possible boot parameter blocks (at I2C EEPROM address 0x80*CFGGP[2:0] for C645x/C6472 devices; 0x80*DEVNUM[3:0] for C6474 device). Depending on the value of the Options field in this boot parameter block (see Table 6 and Table 37), the bootloader continues either by reading and processing a boot configuration (to reprogram certain registers and memory locations), or by reading the actual application in the form of a boot table. In the first case, the initial boot parameter set is typically configured to point to a second boot parameter set which is loaded after boot configuration is processed. This second parameter set typically indicates a boot table load (meaning the actual application), after which the bootloader transfers control to the application code starting at the entry point specified by the boot table itself.

This section describes how each of the three types of tables (boot parameters, boot table and boot configuration table) are generated and how they are combined into a single I2C EEPROM image. Examples of a few typical boot scenarios are also given.

6.2.1 Boot Parameter Table

Table 6 and Table 37 show the required format for the boot parameter table. The following sections describe some examples.

6.2.1.1 Boot Parameter Example for Setting Up Boot Table Download

Table 42 shows an example of boot parameter entries used for boot table load. The options field is set to 01b, indicating the boot table mode. The LSW portion of the device address (offset 8) points to the start address of the boot table at EEPROM address 0x80. This example assumes that the I2C EEPROM only contains the parameter set at EEPROM address 0x0, and the actual application boot table which starts at the next 128-byte block; i.e., at EEPROM address 0x80. The MSW portion of the device address is the default I2C EEPROM bus address, 0x50.

Table 42. Boot Parameter Table Example for Boot Table Load

Offset (byte)	Field	Value
0	Length	26
2	Checksum	0 (check disabled)
4	Boot mode	5 (I2C boot)
6	Options	01 (Boot Table Mode)
8	Device address (LSW)	0x80
10	Device address (MSW)	0x50
12	Broadcast address	0 (not used)
14	Slave address	0 (not used)
16	CPU frequency (MHz)	50 (PLL1 Controller is in Bypass mode)
18	I2C clock frequency (kHz)	50
20	Next device address (LSW)	0 (not used)
22	Next device address (MSW)	0 (not used)
24	Address delay	0

6.2.1.2 *Boot Parameter Example for Setting Up Boot Configuration Table Download*

A boot configuration is used if certain peripherals must be programmed with values that differ from their reset values prior to loading an application. For example, if the application needs to be loaded into DDR memory, then a boot configuration load can be used to program DDR registers and enable the DDR peripheral. Table 43 shows an example of boot parameter entries used for boot configuration load. This example assumes that the boot configuration starts at address 0x400 in the EEPROM (at the end of the parameter block), and that the boot parameters for the subsequent boot table block start at address 0x80.

Table 43. Boot Parameter Table Example for Boot Configuration Load

Offset (byte)	Field	Value
0	Length	26
2	Checksum	0 (check disabled)
4	Boot mode	5 (I2C boot)
6	Options	02 (Boot Configuration Mode)
8	Device address (LSW)	0x400
10	Device address (MSW)	0x50
12	Broadcast address	0 (not used)
14	Slave address	0 (not used)
16	CPU frequency (MHz)	50 (PLL1 Controller is in Bypass mode)
18	I2C clock frequency (kHz)	50
20	Next device address (LSW)	0x80
22	Next device address (MSW)	0x50
24	Address delay	0

6.2.2 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. Section 6.2.2.2 and Section 6.2.2.3 detail the role of the hex conversion utility in creating the boot table.

6.2.2.1 Boot Table Structure

The boot table format is as follows:

- A 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)

6.2.2.2 Code and Data Sections in the Boot Table

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 with 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 is the indication of the end of the boot table and the bootloader then branches to the entry point address (specified at the beginning of the boot table) and begins execution of the application.

6.2.2.3 Creating the Boot Table

To create the boot table, use the following steps:

1. Use the hex conversion utility (hex6x.exe) revision 6.0A or later. Earlier versions may not support the boot table features correctly.
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 desired output format. See the *TMS320C6000 Assembly Language Tools v 6.1 User's Guide* (SPRU186) for detailed information on the available hex conversion utility output formats.
6. 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.
7. Specify the endianness to match the compilation, -order L for little endian, -order M for big endian.
8. 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.

Section 6.2.2.3.1 shows an example of how to set the hex conversion utility options to create a boot table.

6.2.2.3.1 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
```

For detailed information on the C6x hex conversion utility, see the *TMS320C6000 DSP Assembly Language Tools User's Guide* (SPRU186).

6.2.3 Boot Configuration Table

The boot configuration tables provide a read/modify/write capability to any memory on the DSP. Each table entry has three elements that are each 32 bits wide. 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. Any bits that are set in the set mask are set, any that are set in the clear mask are cleared, and any that are set in both are toggled. If both the set mask and the clear mask are 0, then the value in the address field is branched via a standard function call, with the return address stored in register B3.

Boot configuration tables are typically used to configure additional peripherals prior to the I2C boot process, but they can be used to poke a small program into memory and execute it as well.

The boot configuration table is terminated when all three fields are zero.

An example of a boot configuration table with three entries is shown in Table 44.

Table 44. Boot Configuration Table Example

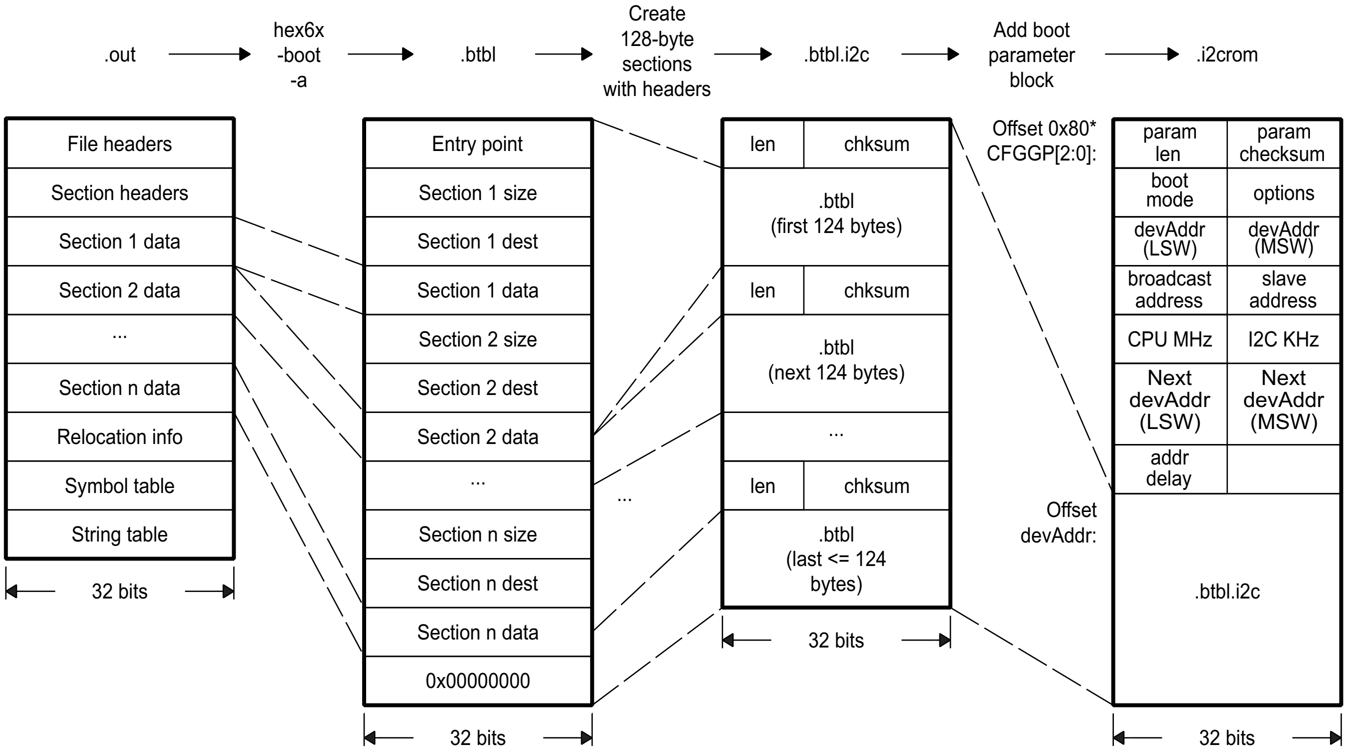
Offset	Data	Operation
0x0	0x0093001C	Set 16 MSBs and clear 16 LSBs at address 0x0093001c
0x4	0xFFFF0000	
0x8	0x0000FFFF	
0xC	0x00930010	Toggle bits 0,8,16 and 24 at address 0x930010
0x10	0x01010101	
0x14	0x01010101	
0x18	0x00930018	Branch to function at address 0x00930018
0x1C	0x00000000	
0x20	0x00000000	
0x24	0x00000000	Termination
0x28	0x00000000	
0x2C	0x00000000	

6.2.4 Creating the Combined EEPROM Image

As seen in the previous sections, the overall I2C EEPROM image consists of a single or multiple boot parameter sets, a boot table, and optionally, single or multiple boot configuration blocks. When the individual files are merged, the offsets in the combined file need to be consistent with the values of Device Address and Next Device Address. A few examples are shown in Figure 4 and Figure 5.

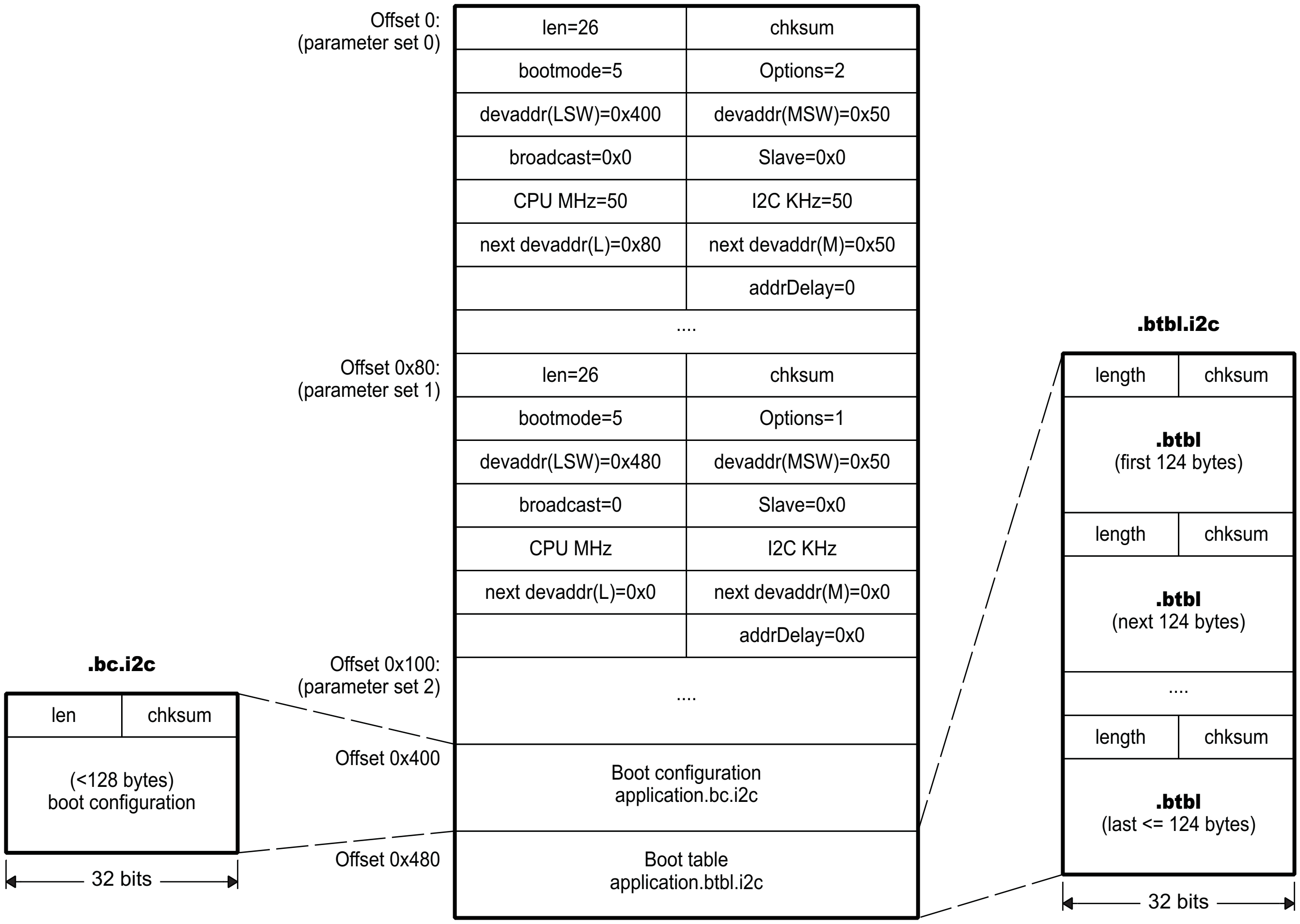
For the most part, this process can be done using existing tools (such as hex6x) or manually (such as creating boot parameters or boot configuration, or determining proper file offsets), although the process can be tedious if too many components are involved. The one step that is more involved is segmentation of boot tables (or an exceptionally long boot configuration) into 128-byte sections. This is required for I2C bootloader operation, as per Section 2.3.4.1 and Section 5.4.2.1. Also, a tool is needed to compute checksums, if used.

Figure 4. Creating an I2C EEPROM Image Containing the Application Program



A The boot parameter table in this figure applies to the C6454/55 device only.

Figure 5. Creating an I2C EEPROM Image Containing Boot Configuration and Application Program



A The boot parameter table in this figure applies to the C6454/55 device only.

6.3 *EMAC Boot*

Boot image should be created according to Table 17.

7 Bootloader Expansion

7.1 Second Stage Bootloader

In some cases, you may want to boot the DSP through a peripheral which is not supported by the available boot modes and the on-chip bootloader. For example, you may want to do an Ethernet boot via an EMAC peripheral. First, an Ethernet bootloader is loaded via a Master I2C boot and then it executes, causing the rest of the application to be loaded via the EMAC peripheral. Alternatively, there may be a need to customize the functionality provided by the on-chip ROM bootloader. This can be accomplished using a Second Stage bootloader, which can be loaded via any of the supported bootmodes. The second stage bootloader should use the same structures (boot parameters tables and boot tables) as the on-chip ROM bootloader.

7.2 Boot to DDR2 Memory

In some cases, user may want to boot the application into DDR2 memory. This can be done by using a boot configuration table. The basic idea is to enable DDR2 and program DDR2 memory controller registers before the actual boot table process happens. Similar ideas apply to the C6474 device, but note that the memory map may be different. The following code is an example that works on the C6454/55 DSK. For more details, if needed, see Section 6.2.

7.2.1 Create a Boot Table Mapped to DDR2 Memory

This is application dependent. The following is an example that helps for a quick test.

```
   .data
   .def     someData
someData   .word 01234ABCDh

   .def  byte1
   .sect ".byte1"
byte1:   .byte 0x12

   .def  byte2
   .sect ".byte2"
byte2:   .byte 0x12, 0x34

   .text
   .def _c_int00

myConst    .equ   011223344h

_c_int00:

MVKL.S1        myConst, A1
MVKH.S1        myConst, A1

MVKL.S1        byte1, A2
MVKH.S1        byte1, A2
LDB.D1            *A2, B2

MVKL.S1        byte2, A3
MVKH.S1        byte2, A3
LDB.D1            *A3++, B3
LDB.D1            *A3, B4

etrap:

BNOP.S1 etrap, 5
```

The corresponding link command file is as follows:

```
/* Object file */
example.obj

/* Linker options */
-c
-a

MEMORY
{
   PAGE 0:

     TEXT    :   origin = 0xE0000000, length = 0x0040
     DATA    :   origin = 0xE0000100, length = 0x0004
    BYTE1    :   origin = 0xE0000200, length = 0x0001
    BYTE2    :   origin = 0xE0000300, length = 0x0002

}

SECTIONS
{
     .text  >  TEXT PAGE 0
     .data  >  DATA PAGE 0
     .byte1 >  BYTE1 PAGE 0
     .byte2 >  BYTE2 PAGE 0
}
```

7.2.2 Create a DDR2 Configuration Table

DDR2 memory controller configuration may need to be changed based on the real DDR2 memory used.

```
0x00640000      /* length, checksum */
0x02AC002C      /* PERCFG1 *
0x00000002   /* DDR2CTL =1, enable DDR2 */
0xFFFFFFFD   /* clear mask */
0x78000008   /* SDCFG */
0x00D38822   /* Unlock boot + timing, CAS4, 4 banks, 10 bit column */
0xFF2C77DD   /* clear mask */
0x7800000C   /* SDRFC */
0x000007A2   /* Refresh */
0xFFFFF85D   /* clear mask */
0x78000010   /* SDTIM1 */
0x3EDB4B91   /* Timing 1 */
0xC124B46E   /* clear mask */
0x78000014   /* SDTIM2 */
0x00A2C722   /* Timing 2 */
0xFF5D38DD   /* clear mask */
0x780000E4   /* DMCCTL */
0x00000005   /* PHY read latency for CAS 4 is 4 + 2 - 1 */
0xFFFFFFFA   /* clear mask */
0x78000008   /* SDCFG */
0x00538822   /* Lock, CAS4, 4 banks, 10 bit column, lock timing */
0xFFAC77DD   /* clear mask */
0x00000000   /* last element */
0x00000000   /* last element */
0x00000000   /* last element */
```

The functionality of the above boot configuration table is exactly the same as the following code which can be put into a .GEL file.

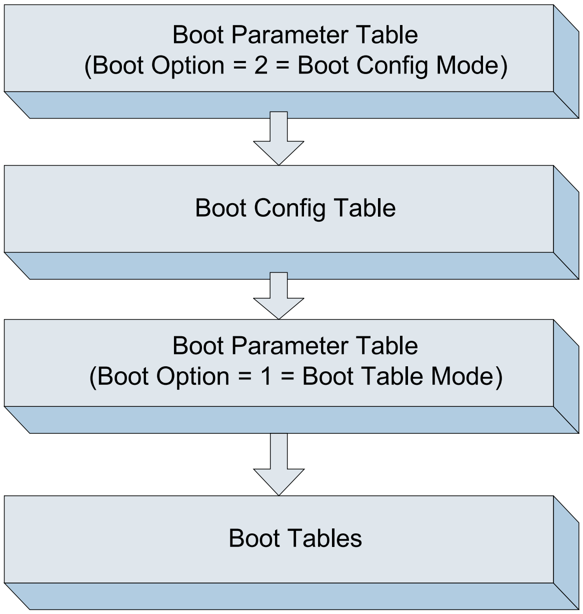
```
#define PERCFG1     0x02AC002C    /* Peripheral configuration 1 */

#define DDR_BASE_ADDR (0x78000000)

#define DDR_MIDR     (*(int*)(DDR_BASE_ADDR + 0x00000000))
#define DDR_SDCFG    (*(int*)(DDR_BASE_ADDR + 0x00000008))
#define DDR_SDRFC    (*(int*)(DDR_BASE_ADDR + 0x0000000C))
#define DDR_SDTIM1   (*(int*)(DDR_BASE_ADDR + 0x00000010))
#define DDR_SDRIM2   (*(int*)(DDR_BASE_ADDR + 0x00000014))
#define DDR_DDRPHYC  (*(int*)(DDR_BASE_ADDR + 0x000000E4))

init_DDR2()
{
   /* Enable the DDR2 Memory Controller */
    *(int *)PERCFG1 = 0x00000002;

   /* Unlock boot+timing,CAS4,4 banks, 10 bit column */
   DDR_SDCFG    = 0x00D38822;
   DDR_SDRFC    = 0x000007A2; /* Refresh */
   DDR_SDTIM1   = 0x3EDB4B91; /* Timing 1 */
   DDR_SDRIM2   = 0x00A2C722; /* Timing 2 */
   DDR_DDRPHYC  = 0x00000005; /* PHY read latency for CAS 4 is 4 + 2 - 1 */
   /* Lock, CAS4, 4 banks, 10 bit column, lock timing*/
   DDR_SDCFG = 0x00538822;
}
```

NOTE: This example assumes DDR2 is implemented as 128M bytes starting from 0xE0000000. For different boards with different DDR2 memory, suggest first use .GEL file to verify whether the DDR2 memory is programmed correctly or not by writing the DDR2 memory in Code Composer Studio.

7.2.3 Create the Combined EEPROM Image

Create the final I2C EEPROM data file, which includes boot parameter table, boot configuration table and finally boot tables. For example, logically it is something like Figure 6.

Figure 6. I2C Tables Illustration


Boot Parameter Table
(Boot Option = 2 = Boot Config Mode)
Boot Config Table
Boot Parameter Table
(Boot Option = 1 = Boot Table Mode)
Boot Tables

7.2.4 Perform the Boot Test

To perform the boot test, program I2C EEPROM with the image in Figure 6, power off the target board, select boot mode to be 5, and then power on. The boot tables should be in DDR2 memory now and the expected result is:

```
A1 = 0x11223344
B2 = 0x12
B3 = 0x12
B4 = 0x34
```

Appendix A Revision History

This revision history highlights the technical changes made to the document in this revision.

Table 45. C645x/C647x Bootloader Revision History

See	Additions/Modifications/Deletions
Section 2.3.5	Modified I2C slave address to 0x4
Section 3.2	Modified first paragraph
Table 25	Modified Default Value for DeviceID to 0x4
Table 37	Modified Default Value for DeviceID to 0x4

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

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