Application Report
SPRAAG0C–July 2007

Using the TMS320DM643x Bootloader

Baldwin, Karen

ABSTRACT

This document describes the functionality of the DM643x ROM bootloader software. Please note that the ROM bootloader requires use of Application Image Script (AIS) as the primary data format for loading code/data. AIS is a Texas Instruments, Inc. proprietary data format. AIS is explained in detail in Section 3 of this document.

This application report contains project code that can be downloaded from this link. http://www-s.ti.com/sc/techlit/spraag0.zip

Contents

List of Figures

List of Tables

1 Introduction

The ROM bootloader resides in the ROM of the device beginning at ROM address 0x00100000. The ROM boot loader (RBL) implements methods for booting in the listed modes. It reads the contents of the BOOTCFG register to determine boot mode and performs appropriate commands to effect boot of device. If an improper boot mode is chosen or if for some reason an error is detected during boot from a slave device, the RBL communicates this through UART as default boot device.

When booting in master mode, the boot loader reads the boot information from the slave device as and when required. When booting in slave mode, the boot loader depends on the master device to feed the boot information as and when required. Please note that for all boot modes, the ROM bootloader disables the watchdog timer for a duration of boot. All applications MUST avoid configuring the watchdog timer during the boot process. (No AIS commands or code should change this during boot). Figure 15 shows a list of terms and abbreviations used in this application report

- Emulation boot
- HPI
- PCI (DSP as slave)
- EMIFA ROM direct boot
- EMIFA ROM fast boot with AIS
- EMIFA ROM fast boot without AIS
- NAND

- I2C (DSP as master)
- SPI 16×8 (DSP as master, 16 bits of address per SPI operation, supporting upto 64Kx8 devices)
- SPI 24×8 (DSP as master, 24 bits of address transmitted per SPI operation supporting upto 16Mx8 devices)
- UART (DSP as slave), no flow control
- UART (DSP as slave), with flow control
- VLYNQ (DSP as slave)

Table 1. Terms and Abbreviations

Term	Description
Bootloader	SW/Code for ROM DM643x Bootloader
AIS	Application Image Script
BL	Boot Loader (referring to the bootloader in this text)
DSP	Digital Signal Processor (referring to DM643x in this text)
EMIF	External Memory Interface
GPIO	General-Purpose Input/Output
HPI	Host Port Interface
I2C	Inter Integrated Circuit
NAND	Inverted AND Gate Not AND
OFD	Object File Display
OS	Op-Code Synchronization
PCI	Peripheral Component Interconnect
POS	Ping Op-Code Synchronization
ROM	Read Only Memory
SPI	Serial Peripheral Interface
SRGR	Sample Rate Generator Control Register
SWS	Start-Word Synchronization
UART	Universal Asynchronous Receiver/Transmitter

2 Boot Mode Description

The selection of the following boot modes depend upon the status of boot device pins. The status of these pins is captured on the rising edge of device $\overline{\text{POR}}$ reset into the BOOTCFG register. The bootloader reads the contents of the BOOTCFG register and branches to the appropriate code to implement the selected boot.

The boot modes are grouped into three categories — Non-Fastboot Modes, Fixed-Multiplier Fastboot Modes, and User-Select Multiplier Fastboot Modes.

- **Non-Fastboot Modes (FASTBOOT = 0):** The device operates in default phased-locked loop (PLL) bypass mode during boot. The Non-Fastboot bootmodes are shown in Table 2.
- **Fixed-Multiplier Fastboot Modes (FASTBOOT = 1, AEM[2:0] = 001b):** The bootloader code speeds up the device during boot according to the fixed PLL multipliers. The Fixed-Multiplier Fastboot bootmodes are shown in Table 3.

  NOTE: The PLLMS[2:0] configurations have no effect on the Fixed-Multiplier Fastboot Modes, as these pins function as AEAW[2:0] to select the EMIFA address width when AEM[2:0] = 001b.
- **User-Select Multiplier Fastboot Modes (FASTBOOT = 1, AEM[2:0] = 000b,011b,100b,101b):** The bootloader code speeds up the device during boot. The PLL multiplier is selected by the user via the PLLMS[2:0] pins. The User-Select Multiplier Fastboot bootmodes are shown in Table 4.

If an invalid boot mode is specified, the bootloader writes an error code to the ERR field of the BOOTCMPLT register and then defaults to UART boot for all non-host boot modes (for example, I2C, SPI).

Boot device pins must be configured to one of the valid modes. A description of each valid mode is given in subsequent sections.

All other modes not shown in these tables are reserved and invalid settings.

Table 2. Non-Fastboot Modes (FASTBOOT = 0)

DEVICE BOOT AND CONFIGURATION PINS		BOOT DESCRIPTION[1]	DM643x DMP (Master/Slave)	PLLC1 CLOCK SETTING AT BOOT			DSPBOOTADDR (DEFAULT)[1]
BOOTMODE[3:0]	PCIEN			PLL MODE[2]	CLKDIV1 DOMAIN (SYSCLK1 DIVIDER)	DEVICE FREQUENCY (SYSCLK1)	
0000	0 or 1	No Boot (Emulation Boot)	Master	Bypass	/1	CLKIN	0x0010 0000
0001	0 or 1	Reserved	–	–	–	–	–
0010	0	HPI Boot	Slave	Bypass	/1	CLKIN	0x0010 0000
	1	Reserved	–	–	–	–	–
0011	0 or 1	Reserved	–	–	–	–	–
0100	0 or 1	EMIFA ROM Direct Boot [PLL Bypass Mode]	Master	Bypass	/1	CLKIN	0x4200 000
0101	0 or 1	I2C Boot [STANDARD MODE][3]	Master	Bypass	/1	CLKIN	0x0010 0000
0110	0 or 1	16-bit SPI Boot [McBSP0]	Master	Bypass	/1	CLKIN	0x0010 0000
0111	0 or 1	NAND Flash Boot	Master	Bypass	/1	CLKIN	0x0010 0000
1000	0 or 1	UART Boot without Hardware Flow Control [UART0]	Master	Bypass	/1	CLKIN	0x0010 0000
1001	0 or 1	Reserved	–	–	–	–	–
1010	0 or 1	VLYNQ Boot	Slave	Bypass	/1	CLKIN	0x0010 0000
1011	0 or 1	Reserved	–	–	–	–	–
1100	0 or 1	Reserved	–	–	–	–	–
1101	0 or 1	Reserved	–	–	–	–	–
1110	0 or 1	UART Boot with Hardware Flow Control [UART0]	Master	Bypass	/1	CLKIN	0x0010 0000
1111	0 or 1	24-bit SPI Boot (McBSP0 + GP[97])	Master	Bypass	/1	CLKIN	0x0010 0000

(1) For all boot modes that default to DSPBOOTADDR = 0x0010 0000 (i.e., all boot modes except the EMIFA ROM Direct Boot, BOOTMODE[3:0] = 0100, FASTBOOT = 0), the bootloader code disables all C64x+ cache (L2, L1P, and L1D) so that upon exit from the bootloader code, all C64x+ memories are configured as all RAM. If cache use is required, the application code must explicitly enable the cache.

(2) The PLL MODE for Non-Fastboot Modes is fixed as shown in this table; therefore, the PLLMS[2:0] configuration pins have no effect on the PLL MODE.

(3) I2C Boot (BOOTMODE[3:0] = 0101b) is *only* available if the MXI/CLKIN frequency is between 21 MHz to 30 MHz. I2C Boot is not available for MXI/CLKIN frequencies less than 21 MHz.

Table 3. Fixed-Multiplier Fastboot Modes (FASTBOOT = 1, AEM[2:0] = 001b)

DEVICE BOOT AND CONFIGURATION PINS		BOOT DESCRIPTION[1]	DM643x DMP (Master/Slave)	PLLC1 CLOCK SETTING AT BOOT			DSPBOOTADDR (DEFAULT)[1]
BOOTMODE[3:0]	PCIEN			PLL MODE[2]	CLKDIV1 DOMAIN (SYSCLK1 DIVIDER)	DEVICE FREQUENCY (SYSCLK1)	
0000	0 or 1	No Boot (Emulation Boot)	Master	Bypass	/1	CLKIN	0x0010 0000
0001	0	HPI Boot with PLL Multiplier x27 at boot	Slave	x27	/2	CLKIN x27 / 2	0x0010 0000
	1	Reserved	–	–	–	–	–
0010	0	HPI Boot with PLL Multiplier x20 at boot	Slave	x20	/2	CLKIN x20 / 2	0x0010 0000
	1	Reserved	–	–	–	–	–
0011	0	HPI Boot with PLL Multiplier x15 at boot	Slave	x15	/2	CLKIN x15 / 2	0x0010 0000
	1	Reserved	–	–	–	–	–
0100	0 or 1	EMIFA ROM FASTBOOT with Application Image Script (AIS)	Master	x20	/2	CLKIN x20 / 2	0x0010 000
0101	0 or 1	I2C Boot [FAST MODE][3]	Master	x20	/2	CLKIN x20 / 2	0x0010 0000
0110	0 or 1	16-bit SPI Boot [McBSP0]	Master	x20	/2	CLKIN x20 / 2	0x0010 0000
0111	0 or 1	NAND Flash Boot	Master	x20	/2	CLKIN x20 / 2	0x0010 0000
1000	0 or 1	UART Boot without Hardware Flow Control [UART0]	Master	x20	/2	CLKIN x20 / 2	0x0010 0000
1001	0 or 1	EMIFA ROM FASTBOOT without AIS	Master	x20	/2	CLKIN x20 / 2	0x0010 0000
1010	0 or 1	VLYNQ Boot	Slave	x20	/2	CLKIN x20 / 2	0x0010 0000
1011	0 or 1	Reserved	–	–	–	–	–
1100	0 or 1	Reserved	–	–	–	–	–
1101	0 or 1	Reserved	–	–	–	–	–
1110	0 or 1	UART Boot with Hardware Flow Control [UART0]	Master	x20	/2	CLKIN x20 / 2	0x0010 0000
1111	0 or 1	24-bit SPI Boot (McBSP0 + GP[97])	Master	x20	/2	CLKIN x20 / 2	0x0010 0000

(1) For all boot modes that default to DSPBOOTADDR = 0x0010 0000, the bootloader code disables all C64x+ cache (L2, L1P, and L1D) so that upon exit from the bootloader code, all C64x+ memories are configured as all RAM. If cache use is required, the application code must explicitly enable the cache.

(2) The PLL MODE for Fixed-Multiplier Fastboot Modes is fixed as shown in this table; therefore, the PLLMS[2:0] configuration pins have no effect on the PLL MODE.

(3) I2C Boot (BOOTMODE[3:0] = 0101b) is *only* available if the MXI/CLKIN frequency is between 21 MHz to 30 MHz. I2C Boot is not available for MXI/CLKIN frequencies less than 21 MHz.

Table 4. User-Select Multiplier Fastboot Modes (FASTBOOT = 1, AEM[2:0] = 000b, 011b, 100b, or 101b)

DEVICE BOOT AND CONFIGURATION PINS		BOOT DESCRIPTION[1]	DM643x DMP (Master/Slave)	PLLC1 CLOCK SETTING AT BOOT			DSPBOOTADDR (DEFAULT)[1]
BOOTMODE[3:0]	PCIEN			PLL MODE[2]	CLKDIV1 DOMAIN (SYSCLK1 DIVIDER)	DEVICE FREQUENCY (SYSCLK1)	
0000	0 or 1	No Boot (Emulation Boot)	Master	Bypass	/1	CLKIN	0x0010 0000
0001	0	Reserved	–	–	–	–	–
	1	PCI Boot without Auto Initialization	Slave	Table 5	/2	Table 5	0x0010 0000
0010	0	HPI Boot	Slave	Table 5	/2	Table 5	0x0010 0000
	1	PCI Boot with Auto Initialization	Slave	Table 5	/2	Table 5	0x0010 0000
0011	0 or 1	Reserved	–	–	–	–	–
0100	0 or 1	EMIFA ROM FASTBOOT with AIS	Master	Table 5	/2	Table 5	0x0010 0000
0101	0 or 1	I2C Boot [FAST MODE][3]	Master	Table 5	/2	Table 5	0x0010 0000
0110	0 or 1	16-bit SPI Boot [McBSP0]	Master	Table 5	/2	Table 5	0x0010 0000
0111	0 or 1	NAND Flash Boot	Master	Table 5	/2	Table 5	0x0010 0000
1000	0 or 1	UART Boot without Hardware Flow Control [UART0]	Master	Table 5	/2	Table 5	0x0010 0000
1001	0 or 1	EMIFA ROM FASTBOOT without AIS	Master	Table 5	/2	Table 5	–
1010	0 or 1	VLYNQ Boot	Slave	x20	/2	CLKIN x20 / 2	0x0010 0000
1011	0 or 1	Reserved	–	–	–	–	–
1100	0 or 1	Reserved	–	–	–	–	–
1101	0 or 1	Reserved	–	–	–	–	–
1110	0 or 1	UART Boot with Hardware Flow Control [UART0]	Master	Table 5	/2	Table 5	0x0010 0000
1111	0 or 1	24-bit SPI Boot (McBSP0 + GP[97])	Master	x20	/2	CLKIN x20 / 2	0x0010 0000

(1) For all boot modes that default to DSPBOOTADDR = 0x0010 0000, the bootloader code disables all C64x+ cache (L2, L1P, and L1D) so that upon exit from the bootloader code, all C64x+ memories are configured as all RAM. If cache use is required, the application code must explicitly enable the cache.

(2) Any *supported* PLL MODE is available. [See Table 5 for supported DM643x PLL MODE options].

(3) I2C Boot (BOOTMODE[3:0] = 0101b) is *only* available if the MXI/CLKIN frequency is between 21 MHz to 30 MHz. I2C Boot is not available for MXI/CLKIN frequencies less than 21 MHz.

Table 5. PLL Multiplier Selection (PLLMS[2:0]) in User-Select Multiplier Fastboot Modes (FASTBOOT = 1; AEM[2:0] = 000b, 011b, 100b, or 101b)

DEVICE BOOT AND CONFIGURATION PINS	PLLC1 CLOCK SETTING AT BOOT		
PLLMS[2:0]	PLL MODE	CLKDIV1 DOMAIN (SYSCLK1 DIVIDER)	DEVICE FREQUENCY (SYSCLK1)
000	x20	/2	CLKIN x20 / 2
001	x15	/2	CLKIN x15 / 2
010	x16	/2	CLKIN x16 / 2
011	x18	/2	CLKIN x18 / 2
100	x22	/2	CLKIN x22 / 2
101	x25	/2	CLKIN x25 / 2
110	x27	/2	CLKIN x27 / 2
111	x30	/2	CLKIN x30 / 2

2.1 Boot Requirements, Constraints, and Default Settings

Please make note of the following requirements:

- FASTBOOT is required for all PCI boot modes.
- Bootloader only supports 16-bit address width for I2C EEPROMs.
- For PCI boot with auto-initialization, an I2C EEPROM must be connected to I2C of the device.
- Please note that all boot timings are optimized for a 27 MHZ input clock frequency.
- I2C, SPI, UART, NAND, and EMIFA FASTBOOT (BOOTMODE[3:0]=0100b) requires data for boot to be stored in AIS Format. AIS is a Texas Instruments, Inc proprietary format for boot images. A detailed description of AIS is given in Section 3 of this document. Any formats used for HOST modes such as HPI and PCI is solely at the discretion of the user.
- When FASTBOOT is selected, the bootloader configures the PLL. The value of the PLL multiplier depends on the status of the AEM and PLLMS[2:0] pins latched at reset into the BOOTCFG register. This document bases all timing calculations assuming a 27 MHZ input clock to the device. For more detailed information and presentation of a wider range of operating frequencies for FASTBOOT, see the device-specific data sheet.
- The ROM bootloader does not support any NAND devices which specifically require the toggle of chip select signal for operation.
- For NAND boot, the NAND device must be connected to EMIFA CS2.
- The bootloader disables CACHE during the boot process, regardless of boot mode chosen. The only boot mode exception is, direct EMIF boot, in which the boot loader is not invoked; therefore, CACHE is in power on the default state.
- The bootloader supports SPI EEPROMS with data arrangement x8 bits for all SPI boot modes. Becasue the bootloader only provides enough clocks to retrieve 8 bits of data, it cannot support devices with x16 bit data arrangement.

2.2 FASTBOOT Mode

With the exception of emulation bootmode (BOOTMODE[3:0]==0000b), when FASTBOOT option is selected, the bootloader software programs the PLL. The PLL multiplier used depends on the value of the AEM and PLLMS[2:0] pins latched at reset. The bootloader reads the value of these pins as latched into the BOOTCFG register at device $\overline{\text{POR}}$ reset and selects PLL multiplier according to Table 6. For more detailed description of these settings and associated timings, see the device-specific data sheet.

Table 6. PLL Multiplier Based on Value AEM and PLLMS[2:0] Pins

FASTBOOT	AEM	PLLMS[2:0]	PLLM
1	001	N/A	If ((BOOTMODE[3:0] == 0001) && (PCIEN==0)) Then PLLM = 26 (CLKIN × 27)
			If ((BOOTMODE[3:0]==0011) && (PCIEN==0)) Then PLLM=14 (CLKIN × 15)
			PLLM = 19 (CLKIN × 20) for all other values of BOOTMODE[3:0], PCIEN
1	!= 001	000	PLLM = 19 (CLKIN × 20) for all values of BOOTMODE[3:0], PCIEN
		001	PLLM = 14 (CLKIN × 15) for all values of BOOTMODE[3:0], PCIEN
		010	PLLM = 15 (CLKIN × 16) for all values of BOOTMODE[3:0], PCIEN
		011	PLLM = 17 (CLKIN × 18) for all values of BOOTMODE[3:0], PCIEN
		100	PLLM = 21 (CLKIN × 22) for all values of BOOTMODE[3:0], PCIEN
		100	PLLM = 24 (CLKIN × 25) for all values of BOOTMODE[3:0], PCIEN
		110	PLLM = 26 (CLKIN × 27) for all values of BOOTMODE[3:0], PCIEN
		111	PLLM = 29 (CLKIN × 30) for all values of BOOTMODE[3:0], PCIEN

Note that the bootloader does not generate an error condition for invalid selections of the PLL multiplier. Therefore, care must be taken to ensure that the selected PLL multiplier does not exceed the timing constraints and operating frequency for boot peripheral or the PLL. Please see the following tables for constraints on PLL multipliers and clock frequencies. For more detailed information on these requirements, see the device-specific data sheet.

Table 7. PLL1 and PLL2 Multiplier Ranges

PLL MULTIPLIER (PLLM)	MIN	MAX
PLL1 Multiplier	x14	x30
PLL2 Multiplier	x14	x32

Table 8. PLLC1 Clock Frequency Ranges

CLOCK SIGNAL NAME		MIN	MAX	UNIT
MXI/CLKIN		20	30 (1)	MHz
PLLOUT	At 1.2-V CV_{DD}	400	600	MHz
	At 1.05-V CV_{DD}	400	520	MHz
SYSCLK1 (CLKDIV1 Domain)	-600 devices		600	MHz
	-500 devices		500	MHz
	-400 devices		400	MHz
	-300 devices		300	MHz

(1) MXI/CLKIN input clock is used for both PLL controllers (PLLC1 and PLLC2).

Table 9. PLLC2 Clock Frequency Ranges

CLOCK SIGNAL NAME		MIN	MAX	UNIT
MXI/CLKIN (1)		20	30	MHz
PLLOUT	At 1.2-V CV_{DD}	400	900	MHz
	At 1.05-V CV_{DD}	400	666	MHz
PLL2_SYSCLK1 (to DDR2 PHY)			333	MHz
PLL2_SYSCLK2 (to VPBE)			54	MHz

(1) MXI/CLKIN input clock is used for both PLL controllers (PLLC1 and PLLC2).

2.2.1 CPU Frequency With FASTBOOT Options

The boot loader software uses a fixed PLL divider of 1 (divide by 2), for deriving CPU clock. Assuming an input oscillator frequency of 27 MHz, Table 10 lists the resulting CPU frequencies based on the PLLM values selected by FASTBOOT options.

Table 10. CPU Frequency During FASTBOOT

PLLM	CPU Frequency
19	270 MHz
14	202 MHz
15	216 MHz
17	243 MHz
21	297 MHz
24	337 MHz
26	364 MHz
27	405 MHz

2.3 Emulation Boot (BOOTMODE[3:0] = 0000b, FASTBOOT = 0 or 1)

In this boot mode the ROM boot loader software executes a software loop. The emulation software has responsibility for performing any code download and controlling the device. All FASTBOOT options are ignored for this boot mode. The PLL operates in bypass mode, yielding a CPU timing of 27 MHZ.

2.4 HPI Boot (BOOTMODE[3:0] = 0001b or 0010b, or 0011b, PCIEN = 0, FASTBOOT = 0 or 1)

In HPI boot mode, the device bootloader hardware module branches to the start of the ROM bootloader software. Then, the ROM bootloader code performs the following sequence:

1. When FASTBOOT = 1, the bootloader programs the PLL based on PLL multiplier settings latched at reset, as discussed in Table 6.
2. Configures any HPI register that may be required.
3. Clears the DSPBOOTADDR register. Clears boot error code field (BOOTCMPLT.ERR) and boot complete bit (BOOTCMPLT.BC) in BOOTCMPLT register.
4. Posts HINT to the HOST device, signaling that the DSP is awake and ready for code download.
5. Enters a software loop waiting for non-zero value in the BOOTCMPLT.BC register.
6. When download of application is complete, the HOST writes the application start address into the DSPBOOTADDR register and then sets the boot complete bit in BOOTCMPLT register.
7. Once BOOTCMPLT.BC has been set by HOST, the ROM bootloader software branches to the address set by HOST in DSPBOOTADDR.

2.5 PCI Boot (BOOTMODE[3:0] = 0001b or 0010b, PCIEN = 1, FASTBOOT = 1)

DM643x supports the PCI boot with DSP as PCI slave only. The bootloader implements the PCI boot with and without auto-initialization. When the PCI boot with auto-initialization is selected, the bootloader expects auto-init data to be stored in I2C EEPROM connected to I2C of the device. Please note that although the bootloader attempts boot when FASTBOOT mode is not enabled, this is NOT the recommended mode for the PCI boot. Please enable FASTBOOT with any PCI boot mode to ensure PCI timing meets requirements.

In PCI boot mode with no auto-initialization, the ROM bootloader performs the following steps:

1. Bootloader configures PLL using the PLL multiplier selected based on the value of AEM and PLLMS[2;0] according to Table 6. (Please note that if FASTBOOT is not selected, the bootloader still attempts to complete boot. However, the PCI operating frequency may not meet minimal PCI requirements of 33 MHZ).
2. Clears the DSPBOOTADDR and BOOTCMPLT register fields.
3. When boot mode = 0001b, the ROM bootloader sets PCI CONFIG_DONE bit in the PCI Configuration Done Register (PCICFGDONE) and the PCI Slave Control Register (PCISLVCNTL) to 1. When boot mode = 0010b, PCI auto-init mode is enabled and the ROM bootloader programs the PCI wrapper registers setting CONFIG_DONE = 1 after this is complete.
4. The bootloader then enters a software loop polling for BOOTCMPLT.BC. Once boot complete is detected, the ROM bootloader software branches to the address set by the HOST in DSPBOOTADDR register.

When FASTOOT mode is selected along with the PCI boot, as the first step, the ROM bootloader software configures the PLL prior to clearing DSPBOOTADDR and BOOTCMPLT registers.

When the PCI boot with auto-initialization is selected, the bootloader reads the PCI configuration data stored in I2C EEPROM connected to the I2C of the device. The data as stored in I2C EEPROM must begin at I2C EEPROM address 0×400 and is formatted as shown in Table 11.

Table 11. PCI Configuration Data for Auto-Init

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]

The PCI initialization data is expected to be stored in BIG-ENDIAN format.

2.6 EMIFA ROM Direct Boot (BOOTMODE[3:0] = 0100b, FASTBOOT = 0)

EMIFA direct boot does not require intervention from the ROM bootloader software. The DSP hardware boot module causes direct branch to start of EMIFA memory at address 0×42000000.

2.7 EMIFA ROM Fast Boot Without AIS (BOOTMODE[3:0] = = 1001b, FASTBOOT == 1)

In this boot mode, the ROM bootloader configures the PLL based on values of AEM and PLLMS[2:0] pins latched at reset. Then, it branches directly to address 0×42000000. This boot mode effectively operates the same as EMIFA direct boot (BOOTMODE[3:0]==0100b, FASTBOOT==0), with exception that the PLL is now configured. This enables faster EMIF clock to speed boot from an external device.

2.8 EMIFA ROM Fast Boot With AIS (BOOTMODE[3:0] = 0100b, FASTBOOT = 1)

During EMIFA FAST ROM boot mode the DSP hardware boot module transfers control to the ROM bootloader software. This boot mode operates differently than the EMIFA direct boot. The ROM bootloader controls the boot process, first programming the PLL to operate at faster CPU speeds, then reading code/data starting at EMIFA address 0×42000000. The data stored in the FLASH/ROM must be in AIS format. A description of AIS is given in Section 3 of this document. The AIS boot image consists of AIS commands and data necessary to load the application code into the DSP memory. Using the AIS format, eliminates requirement for user defined secondary boot loader to load code. The ROM bootloader processes AIS commands from the EMIFA ROM until an AIS JUMP_CLOSE instruction is encountered. The JUMP_CLOSE instruction contains the application code start address. This command signals that the application has been fully loaded and all AIS commands have been processed for the boot. The ROM bootloader clears its internal state and then branches to the start of the application code. EMIF FASTBOOT sequence:

1. Programs the PLL using the PLL multiplier selected by the value of the AEM and PLLMS[2:0] pins as shown in Table 6.
2. Reads the value of the 8_16 pin as latched into the BOOTCFG register and sets the EMIF data width accordingly.
3. Fetches the AIS data from the external memory and processes the AIS commands until the JUMP_CLOSE command is encountered.
4. Branches to the application start address given in the JUMP CLOSE command.

2.9 I2C Master Mode Boot (BOOTMODE[3:0] = 0101b, FASTBOOT = 0 or 1)

The DM643x supports the I2C boot with DSP as I2C master only. The DSP hardware boot module transfers control to the ROM bootloader software at device reset. The ROM bootloader configures the I2C peripheral device, and begins read of data from the I2C EEPROM. The data stored in the I2C EEPROM is expected to be in AIS format. The first 32 bits are ignored by the bootloader; this location is currently reserved. The second 32 bit word must contain the AIS magic number. The remaining data in the I2C EEPROM must be in AIS format. Please refer to Section 3 for details of AIS. Boot sequence for I2C is as follows:

1. When FASTBOOT = 1, bootloader programs PLL using PLL miltiplier selected by values of AEM and PLLMS[2:0] as seen in Table 6.
2. Configures I2C for master mode with slave address register set to 0x50, and own address register set to 0×29.
3. Processes each AIS command until JUMP CLOSE command is encountered.
4. Branches to application start address
5. If an error occurs during the I2C boot process, the bootloader writes an error condition in the ERR field of the BOOTCMPLT register. Then, it attempts to perform boot through UART.

2.9.1 I2C Master Boot Timing

The bootloader sets the following values for I2C clock pre-scale and clock low hold/clock high hold registers:

Table 12. I2C Timing Register Settings

FASTBOOT	Register	Value
0	IPSC	0x2
	ICCLKH	0x2D
	ICLKL	0x2D
1	IPSC	0x2
	ICCLKH	0x8
	ICCLKL	0x8

The frequency of the I2C master clock is derived by:

$$\text{I2C master clock frequency} = \frac{\text{I2C peripheral clock frequency}}{(\text{IPSC} + 1) * \left[(\text{ICCLKL} + \text{D}) + (\text{ICCLKH} + \text{D})\right]}$$

For DM643x, the I2C input clock is directly derived from CLKIN. The value of the quantity represented by "D" is determined by the IPSC value (IPSC > 1, D = 5, IPSC = 1, D = 6, IPSC = 0, D = 7). For purposes of determining the I2C master clock used for boot, D = 5, since the boot loader software always programs a value of 2 for IPSC. Assuming an input oscillator frequency of 27 MHz, the settings for IPSC< ICLKH, ICCLKL, in Table 12 yields the following I2C master clock frequencies:

FASTBOOT == 1

I2C Master clock ~ 310 Khz

FASTBOOT == 0

I2C Master clock ~87 Khz

Please note that the input clock for the I2C module bypasses the PLL, therefore any FASTBOOT option settings using AEM and PLLMS[2:0] and subsequent PLLM selection has no effect on the frequency of the I2C master clock.

2.10 SPI 16x8 Master Mode Boot (BOOTMODE[3:0] = 0110b, FASTBOOT = 0 or 1)

SPI 16x8 boot is implemented by configuring MCBSP0 of the device to operate in SPI mode. This mode supports SPI EEPROMS that require 16 bits of address and fetch/receive 8 bits of data. Sixteen bits of address allows boot from SPI devices with sizes upto 64Kx8. The bootloader only supports DSP as SPI master for this boot mode. The bootloader software configures MCBSP0 for SPI mode with 32 bit data transmit/receive. The SPI read command and 16 bit address are packed into the upper 3 bytes of the 32 bit word. The fourth empty byte provides the clock cycles needed to retrieve 8 bits of data out from the SPI EEPROM. The boot flow is as follows:

1. If FASTBOOT is enabled, the bootloader programs PLL using PLLM value selected according to Table 6.
2. The bootloader then reads AIS formatted boot image from EEPROM.
3. When last AIS command is encountered (JUMP CLOSE command) the bootloader branches to application entry address given in the command.

2.10.1 SPI 16x8 Master Boot Timing

The SPI master clock frequency is derived from the internal clock provided to the MCBSP0. The peripheral clock is derived by a fixed divide of 1/6 the CPU clock. The master clock is then further divided by the value of the CLKGDV field of the MCBSP's SRGR. The bootloader software fixes the CLKGDV at a value of 0x2. This provides a divide down ratio of 1/3 of the MCBSP input clock. Table 13 shows the derived master clock frequency based on the possible PLLM values.

Table 13. SPI Master Clock Frequencies for FASTBOOT = 1

PLLM	PLLOUT (MHz)	CPU (MHz)	MCBSP (MHz)	SPI Master (MHz)
19	540	270	45	15
14	404	202	33.7	11
15	432	216	36	12
17	486	243	40.5	13.5
21	594	297	49.5	16.5
24	675	337	56	18.7
26	729	364	60.7	**20.2**
29	810	405	67.5	**22.5**

Please note that the timings given in the table are based on 27Mhz input clock frequency. Please check the datasheet for your particular device and the datasheet for the your SPI EEPROM to determine proper timiing and frequency range.

2.10.2 SPI 16x8 Master Boot Signal Polarity

MCBSP0 is configured for SPI Master boot with following modes selected:

Table 14. SPI Master Boot Modes

Register	Value
PCR	Field Values Set
	FSXM = 1, FSRP = 1, CLKXM = 0, FSXP = 1
SPCR	Field Values Set
	CLKSTP = 3, (11b)
RCR	Field Values Set
	RDATDLY = 1
XCR	Field Value Set
	XDATDLY = 1

With these modes selected, the SPI master clock polarity is inactive high, and a MCBSP begins data transfer one-half clock cycle prior to first rising edge of clock amd samples input data on the rising edge of the clock. This operation supports SPI EEPROMS that sample data on rising edge of clock, and send data out on falling edge of clock.

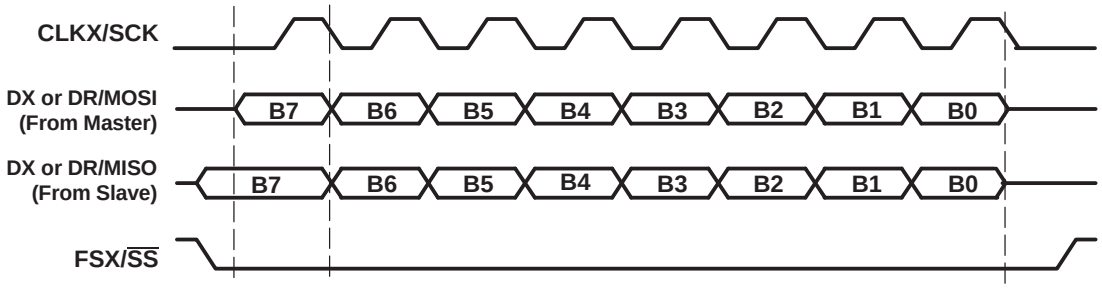

Figure 1. SPI Transfer With CLKSTP = 11 and CLKXP = 0

For further details on how the MCBSP operates when in SPI mode, please refer to the *TMS320C6000 Multi-Channel Buffered Serial Port User's Guide* (SPRU580).

2.10.3 Connecting SPI EEPROM for SPI 16x8 Boot

To enable boot from 16x8 SPI EPPROM, the EEPROM should be connected to McbSP0 pins in accordacne with Table 15

Table 15. SPI 16x8 EEPROM-to-DSP McBSP0 Connection

SPI EEPROM	McBSP0
Sn	FSX0
C	CLKX0
D	DX0
Q	DR0

2.11 NAND Flash Boot (BOOTMODE[3:0] = 0111b, FASTBOOT = 0 or 1)

NAND Flash boot mode is currently supported via secondary boot. There is a race condition in the polling for NAND ready in the ROM bootloader for Rev 1.10 and 1.20 of the device. Therefore this boot mode is not fully supported in ROM. Workaround for this problem is to boot from any of the other supported boot methods, such as I2C or SPI, and allow secondary bootloader to then load code from NAND flash. A sample secondary boot loader and the code needed to flash the secondary code to I2C or SPI is given in the attachment to this document. A list of devices supported by the secondary NAND boot is given in Table 16. Please note that the secondary bootloader does not support any NAND devices which specifically require toggle of chip select signal for operation. NAND device used for boot must be connected to EMIFA CS2.

Table 16. Supported NAND Device Types

Device ID	Page Size	Total Memory Size
0×E3	512+16	4 MB
0×E5	512+16	4 MB
0×E6	512+16	8 MB
0×39	512+16	8 MB
0×6B	512+16	8 MB
0×73	512+16	16 MB
0×33	512+16	16 MB
0×75	512+16	32 MB
0×35	512+16	32 MB
0×43	512+16	16 MB
0×45	512+16	32 MB
0×53	512+16	16 MB
0×55	512+16	32 MB
0×76	512+16	64 MB
0×36	512+16	64 MB
0×79	512+16	128 MB
0×71	512+16	256 MB
0×46	512+16	64 MB
0×56	512+16	64 MB
0×74	512+16	128 MB
0×F1	2048+64	128 MB

Table 16. Supported NAND Device Types (continued)

Device ID	Page Size	Total Memory Size
0×A1	2048+64	128 MB
0xAA	2048+64	256 MB
0×DA	2048+64	256 MB
0×AC	2048+64	512 MB
0×DC	2048+64	512 MB
0×B1	2048+64	128 MB
0×C1	2048+64	128 MB

The secondary bootloader expects the data in NAND to be in AIS format. Please note that AIS data is considered to be a serial stream, therefore all AIS data must be contained in contiguous pages/blocks within the FLASH. Currently, the secondary bootloader makes no attempt to bypass bad blocks. Once it has determined location for start of AIS data, it assumes remaining data is in contiguous 'good' blocks of memory. The secondary bootloader does perform 1 bit error correction, when ECC 1 bit error is detected. The bootloader begins search of AIS data from block 1 of the memory and searches the all remaining blocks to find the AIS magic number. Block 0 is reserved for use of the application.

2.12 UART Boot (BOOTMODE[3:0] = 1000b, 1110b, FASTBOOT = 0 or 1)

UART boot differs from the other boot modes in that the bootloader software performs some communication with the HOST during the boot process. The bootloader performs the following sequence when UART boot is selected.

1. When FASTBOOT = 1, the bootloader programs the PLL using the PLL mulitplier selected according to Table 6.
2. Bootloader configures UART registers as required by chosen mode.
3. Bootloader sends message BOOTME through the serial interface to the HOST.
4. Bootloader waits response from the HOST in the form of the AIS magic number. The bootloader will continually loop, waiting for response.
5. When response is received from HOST< the bootloader processes AIS commands as read from the serial interface until a JUMP CLOSE command is encountered.
6. When JUMP CLOSE command is read, the bootlaoder sends message, DONE, to the HOST and then branches to the application start address.

Please note that the AIS commands are expected to be in ASCII representation, hence to send the AIS magic word , 0x41504954, the character sequence "41","50","49","54", is expected to be recieved by the bootloader. A sample AIS stream for UART boot is given in Section 3.

2.12.1 UART Boot Timing

Operationally, UART boot via BOOTMODES[3:0]==1000b and 1110b are essentially the same. The difference is in the management of data flow. When BOOTMODE[3:0] = 1000b, UART boot is executed without use of hardware flow control. UART BOOTMODE[3:0]=1110b is selected, then the UART is configured to use the hardware flow control module. In both modes the UART FIFO is enabled, and is set for the maximum FIFO size of 14.

The bootloader software does not use auto-baud detect. The UART clock divide registers are set for a total divide down value of 15. With a 27 MHZ input clock, this yields an approximate baud rate of 115 kbps (kilobits per second). The input clock supplied to the UART bypasses the PLL, therefore this baud rate is unaffected by PLL configuration and advantage can be taken of the FASTBOOT modes for faster CPU clock. The required connection settings for UART boot are given in Table 17

Table 17. UART Connection Attributes for Boot

Attribute	Value
Baud Rate	115 kbps
Data Bits	8
Stop Bits	1
Parity	None
Flow Control	Hardware flow control (BOOTMODE[3:0} ==1110b), or none (BOOTMODE[3:0] == 1000b)

2.13 VLYNQ Boot (BOOTMODE[3:0]=1010b, FASTBOOT = 0 or 1)

The ROM bootloader supports boot via VLYNQ with DSP as VLYNQ slave. The bootloader ensures that VLYNQ is enabled and then polls for BOOTCMPLT.CMPLT flag in the BOOTCMPLT register to indicate that the download of application by VLYNQ Host is complete. The bootloader will then branch to the start address in DSPBOOTADDR as written by the VLYNQ Host. The boot process for VLYNQ then is as follows:

1. If FASTBOOT is enabled bootloader configures PLL according using appropriate PLL multiplier.
2. Bootloader makes sure VLYNQ is enabled.
3. Bootloader executes empty loop polling for BOOTCMPLT.CMPLT flag.
4. VLYNQ Host downloads application to DSP.
5. VLYNQ Host writes application start address to DSPBOOTADDR register.
6. VLYNQ Host writes 1 to BOOTMCPLT.CMPLT register flag.
7. Bootloader detects BOOTCMPLT.CMPLT and branches to start address in DSPBOOTADDR.

2.13.1 VLYNQ Boot Timing

There are two independent clocks that should be considered when configuring the DSP for boot from VLYNQ. There are 1) the VLYNQ clock (data clock) and 2) the internal VLYNQ module clock. Because the DSP is acting as VLYNQ slave for purposes of boot, the VLYNQ clock (data clock) will be provided externally by a VLYNQ master. This is the clock that determines the VLYNQ data rate. The VLYNQ clock frequency should be determined by the system requirements and is independent of the internal VLYNQ module clock frequency.

The internal VLYNQ module clock is provided by the clock module within the DSP and is affected by CLKIN/PLLOUT depending on PLL mode. When PLL is in bypass mode, the internal VLYNQ module clock is derived from CLKIN and has value CLKIN/6. When any of the FASTBOOT options are chosen, then PLL is no longer in bypass mode, and internal VLYNQ module clock timing is derived from PLLOUT, and has value SYSCLK1/6 (SYSCLK1 is CPU clock). The internal VLYNQ module clock should not exceed 99Mhz. So care should be taken when choosing PLL multipliers for FASTBOOT option that the value of SYSCLK1 does not exceed 594Mhz, keeping internal VLYNQ module clock at or below the rated operating frequency of 99Mhz. Again, the 99Mhz limitation is on internal VLYNQ module clock ONLY and has no effect on the external VLYNQ clock (data clock) provided by the VLYNQ Master.

2.14 SPI 24×8 Master Mode Boot (BOOTMODE[3:0]=1111b, FASTBOOT = 0 or 1)

This bootmode supports SPI EEPROMs that require 24 bits of address and transfer 8 bits of data per read/write cycle. Twenty-four bits of address allows support of SPI EEPROMS with size upto 16Mx8. As in SPI 16 bit mode, data stored in the SPI EEPROM is expected to be formatted as a valid AIS image and follows the same boot flow:

1. If FASTBOOT is enabled, bootloader configures PLL
2. The bootloader then reads AIS formatted boot image from EEPROM.
3. When last AIS command is encountered (JUMP CLOSE command) the bootloader branches to application entry address given in the command.

To enable communication with SPI EEPROMs that require 24 bits of address, the McBSP is configured differently for 24 bit SPI Master mode boot, than in 16x8 Master mode. The McBSP has a native limitation of 32 bits for transmit/receive when operating in SPI mode. To effectively address SPI EEPROMs requiring 24 bits of address, 40 bits are actually needed; 8 bits of command + 24 bits of address + 8 bits to clock data in/out of the EEPROM. So in SPI 24x8 Master boot, a GPIO pin is used as chip select signal, instead of FSX0 which is normally connected for this purpose. When using this boot mode the FSX0 pin should not be connected and is unused. McBSP0 is then configured to transmit/receive 8 bit data. Please note that chip select is assumed inactive high for this boot mode. The 40 bits needed to communicate with the SPI device are transmitted in separate bytes. The transmit sequence from the DSP to EEPROM is:

1. Assert GPIO (drive GPIO high initially before McBSP0 is released to avoid premature chip select toggle).
2. De-assert GPIO (driving chip select low).
3. Transfer 8 bits of SPI command.
4. Transfer SPI addres bits [23:16].
5. Transfer SPI address bits [5:8].
6. Transfer SPI addres bits [7:0].
7. Send 8 dummy bits to enable clock out of data from EEPROM.
8. Bootloader reads 8 bits from DR0.
9. Assert GPIO (drive chip select high).

2.14.1 SPI 24x8 Master Boot Timings

Timings for SPI 24x8 Master Boot is determined by system PLL settings and McBSP0 divide down clock. These are described in detail in Section 2.10.1

2.14.2 SPI 24x8 Boot Signal Polarity

The McBSP SPI mode settings for 24x8 mode are the same as for 16x8 SPI Master mode, CLKSTP = 11b, CLKXP = 0. In this mode, data is transmitted from DSP one-half cycle prior to rising edge of clock and received on rising edge of clock. This operational mode is compatible with SPI EEPROMS which sample data in on rising edge of clock and clock data out on the falling edge.

The read timings for the 24-bit SPI mode are given below. Figure 2 shows example 24x8 bit SPI EEPROM read timing.

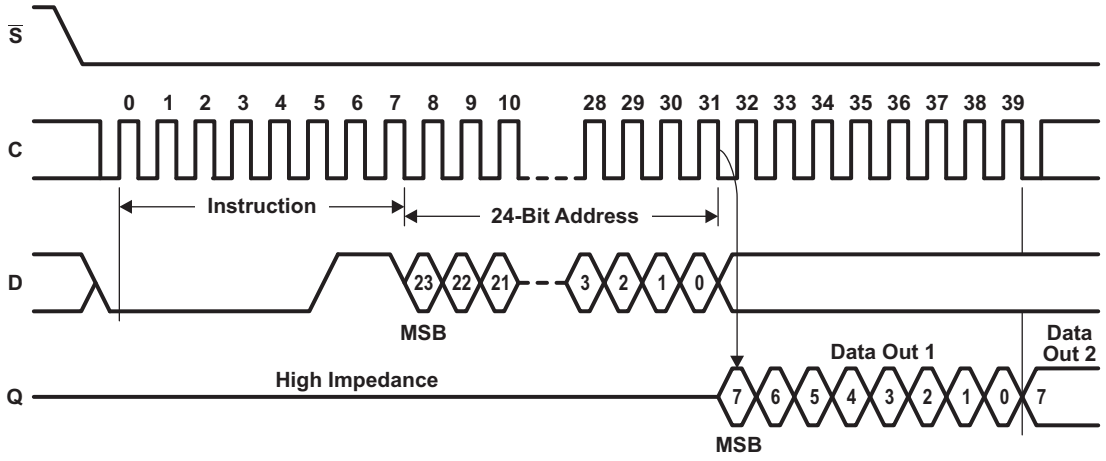

Figure 2. 24x8 Bit SPI EEPROM Read Timing

Figure 3 shows how the DM643x 24x8 bit SPI boot works. Table 18 shows the pin connection between the DM643x and the 24x8 SPI EEPROM.

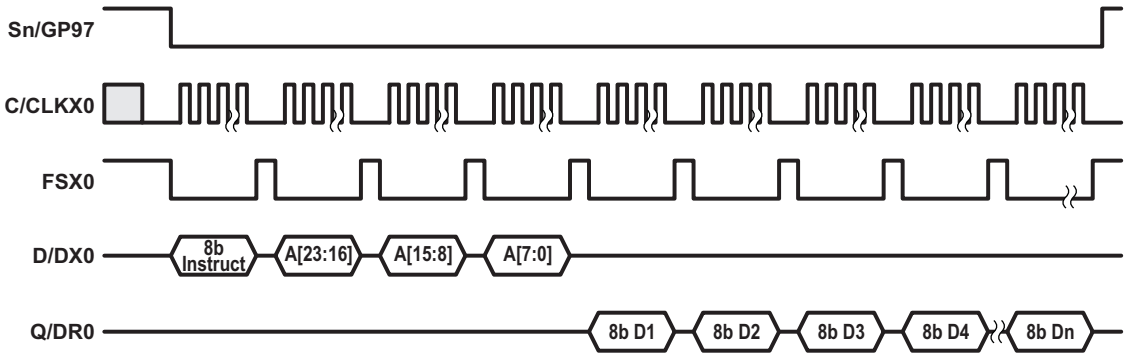

Figure 3. DM643x 24x8 Bit Address SPI Boot

2.14.3 Connecting SPI EEPROM for SPI 24x8 Boot

For 24×8 bit SPI mode, the McBSP0 pins must be connected to the SPI EEPROM according to Table 18

Table 18. SPI EEPROM and DSP Pin Connections for 24 Bit SPI Mode

SPI EEPROM	DSP	Comment
Sn	GPIO97	Connects GPIO97 to chip select of SPI EEPROM
C	CLKX0	Connects CLKX0 to clock of SPI EEPROM
D	DX0	Connects DX0 to data in of SPI EEPROM
Q	DR0	Connects DR0 to data out of SPI EEPROM
-	FSX0	Leave uncconected, this signal is not used

3 Application Image Script

The bootloader accepts boot information in the form of a script, called application image script (AIS). Application image script is a Texas Instruments, Inc. proprietary application image transfer format. This script is a binary file consisting of a script header followed by various commands that can be interpreted and executed by the boot loader. Each command contains an op-code, followed by optional additional data required to execute the op-code. The bootloader currently supports AIS version 1.99; all commands and data are assumed to be 32 bits in width.

The AIS starts a header that consists of a magic word (0×41504954); the header is then followed by a series of commands as shown in Figure 4. Each command consists of an op-code followed by optional additional data. All AIS command streams are terminated with a JUMP_CLOSE command which causes transfer of control to the loaded application code and terminates execution of the ROM bootloader.

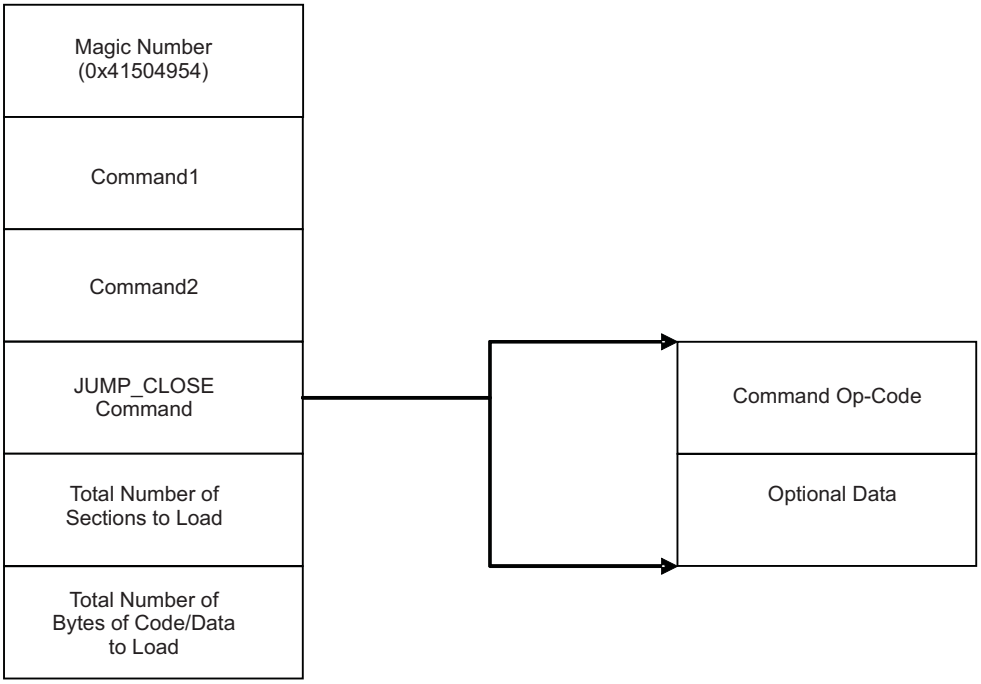

Figure 4. Basic Structure of Application Image Script

The bootloader only accepts data in AIS format for all modes except HPI ad PCI. The following sections define each command with appropriate op-code, structure and placement in AIS. Table 19 lists the various opcodes that are supported by AIS 1.0.

Table 19. AIS Version 2.0 Supported Opcodes

Opcode	Value
Section Load	0×58535901
Request CRC	0×58535902
Enable CRC	0×58535903
Disable CRC	0×58535904
Jump	0×58535905
Jump_Close	0×58535906
Set	0×58535907
Start Over	0×58535908
Reserved	0×58535909
Section Fill	0×5853590A
Get	0×5853590C
Function Execute	0x5853590D

3.1 SET Command

The SET command is a simple mechanism that enables you to write 8-bit, 16-bit or 32-bit data to any address in DSP address space. One of the arguments to this command implements a delay after the memory write happens. This can be used for memory mapped register write to take effect. Set commands may be used to configure various peripherals of the DSP. This includes PLL and EMIF at minimum and can configure more peripherals if required.

When DSP comes up from reset, the PLL is in bypass mode. As a result, the CPU is clocked at the same frequency as connected crystal/CLK IN, which is generally very low. This results in slow communication and high boot time. Selecting FASTBOOT mitigates this by programming the PLL with a slightly higher multiplier of 0×C, but this does not change default EMIF wait states, etc. In order to reduce boot time, the PLL and EMIF registers can be re-configured at the very beginning of the boot process using a series of SET commands. For this reason, all SET commands for configuring EMIF and setting PLL should be placed at the beginning of the AIS boot image as shown in Figure 5.

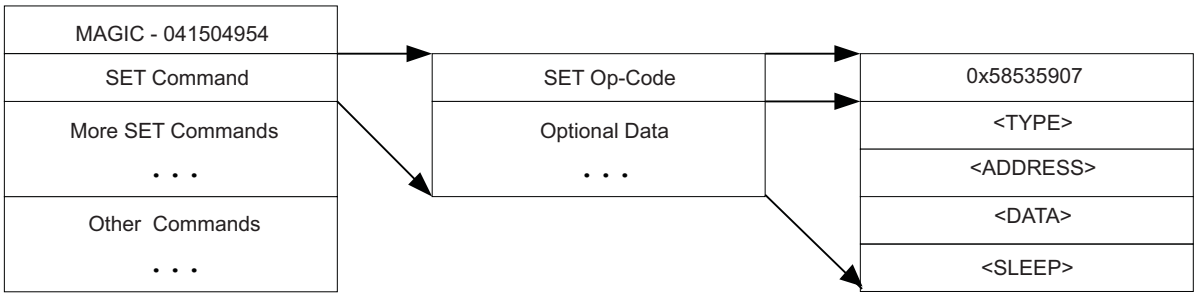

Figure 5. Structure of SET Command

Each set command consists of SET (0×58535907) op-code, followed by four words of additional data as shown. SET command entries in AIS can be explained using the following representation:

```
<Address> =
<Data><Type>::<Sleep>
```

The above command instructs bootloader to write <Data> to address <Address> in DSP address space and then sleep for <Sleep> * CPU clocks. The data-type field <Type> decides whether <Data> should be written as 8 bit (B), 16-bit (S) or 32-bit (I)All other fields can be in any numeric format as described in Table 20.

Table 20. Numeric Formats That Can Be Used in SET Command

Name	Format	Example 1	Example 2	Example 3
Hexadecimal 1	0[xX][0-9a-fA-F]+	0×1234abCD	0×1000	0×5a
Octal	0[0-7]+	02215125715	010000	0132

The data-type field <Type> determines the size of the data item such as 8-bit (B), 16-bit (S) or 32-bit (I). Data-type also can be a *field* or *bits*. This allows the setting of a particular range of bits within the data at the specified address. For *field* and *bits* data-types, the <Type> field also encodes the *start* and *stop* bit positions that define the field to be modified. Table 21 gives a full list of the data-types that can be used.

Table 21. Valid SET Command Data Types

Data Type	Value
8 bit	1
16 bits	2
32 bits	3
Field (1-32 bits)	4
Bits(1-32 bits)	5

The *field* and *bits* data-types are handled similarly by the bootloader. The difference between these types are that with a specifier of *field*, the bootloader performs a read/modify write operation at the given address. The *bits* data type results in a read of the address, followed by a write of the new value to the address. The <Type> specification is a 32-bit word that contains fields for data type (shown above), *start bit*, and *stop bit*. The *start bit* and *stop bit* fields are required only if a data-type of *field(3)* or *(bits(4)* is used. These fields delimit the number of bits that are affected by the instruction. Table 19 shows the encoding of the 32 bit <Type>.

3.1.1 Valid SET Command Data Types

Figure 6. Valid SET Command Data Types

31	24	23	16
Reserved		Stop Bit	

15	8	7	0
Start Bit		Data Type	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 22. Valid SET Command Data Types Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-16	Stop Bit		Stop bit (for *bits* and *fields* data type) last bit position in word that delimits field
15-8	Start Bit		Start bit (for *bits* and *fields* data type) first bit position in word for start of field
7-0	Data Type		Data Type (1,2,3,4,5), specifies type of data to write

3.2 Get Command

The GET command enables fetch of a value stored at any read accessible DSP memory address. The GET command has the same format as the SET command described in Section 3.1, with the exception that delay is not required. All data formatting rules described in the SET command are valid for the GET command. The GET command always transmits full 32 bits even if relevant data is only 8- or 16-bits wide. Data is zero-filled and right-justified (for example, MSBs are zero for all data that is less than 32 bits in length). Figure 7 shows the structure of the GET command.

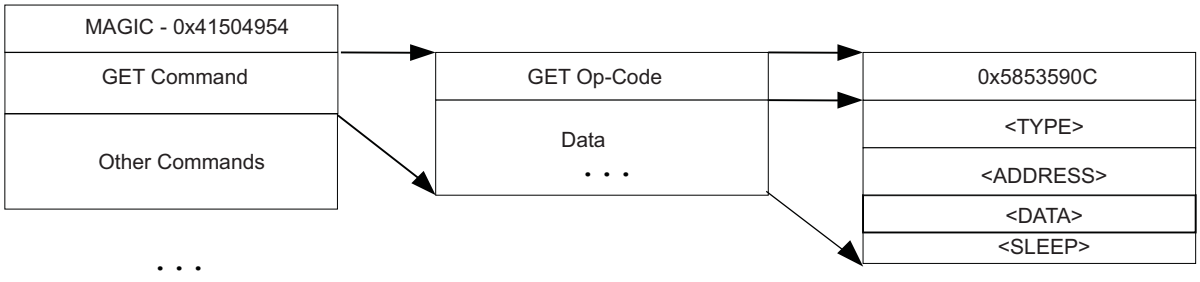

Figure 7. Structure of GET Command

3.3 Section Load Command

Section load command is used to load a chunk of code/data to DSP memory. All initialized sections of application are loaded to DSP memory using Section Load commands. These commands are placed after all SET commands in AIS. Figure 8 shows the structure of the section load command.

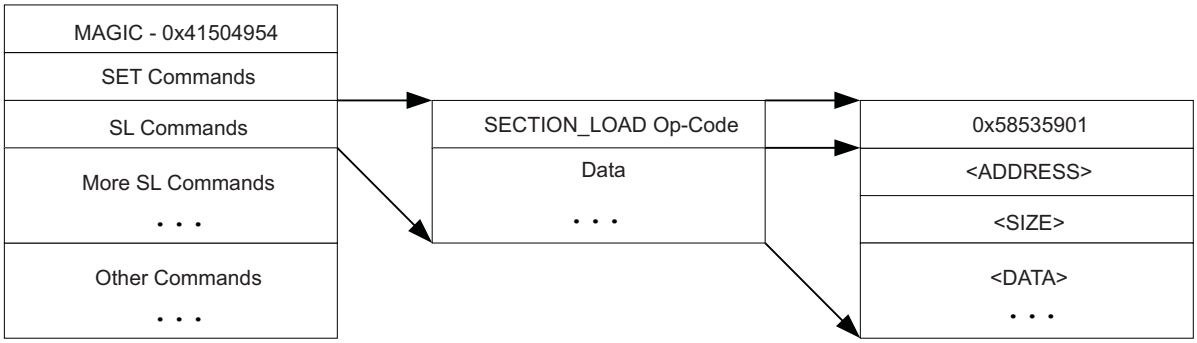

Figure 8. Structure of Section Load Command

Each section load command consists of SECTION_LOAD (0×58535901) op-code, followed by section's start address, size and contents.

3.4 Section Fill Command

Section fill command is used when a particular section is to be filled with a specific pattern. For example, a section that contains all zeros can be initialized with the section fill command. These commands can be placed anywhere where a regular section load command can occur. Figure 9 shows the structure of the section fill command.

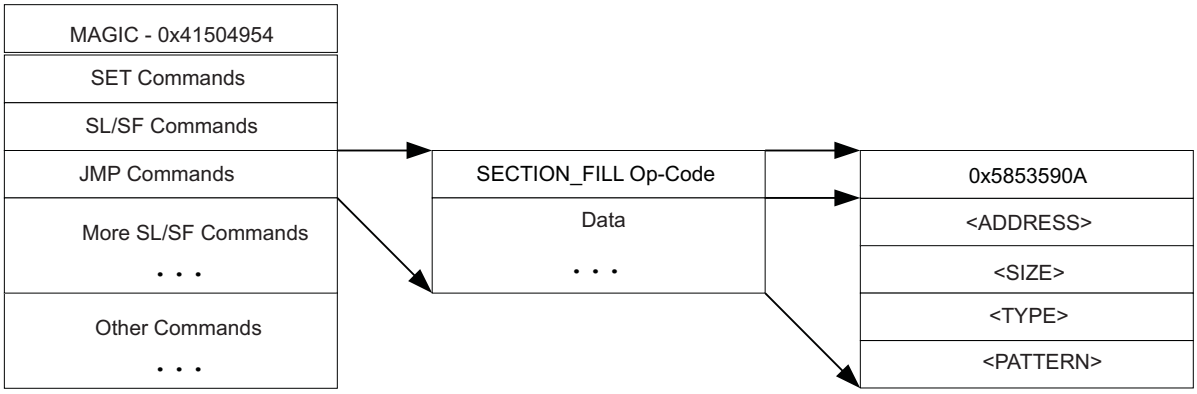

Figure 9. Structure of Section Fill Command

Each section fill command consists of SECTION_FILL (0×5853590A) op-code, followed by section's start address, size, pattern-type (8/16/32 bit) and pattern to be filled.

3.5 Jump Command

This command instructs the DSP to jump to start address of earlier loaded application. It consists of JUMP (0×58535905) op-code, followed by the jump address. Figure 10 shows the structure of the jump command.

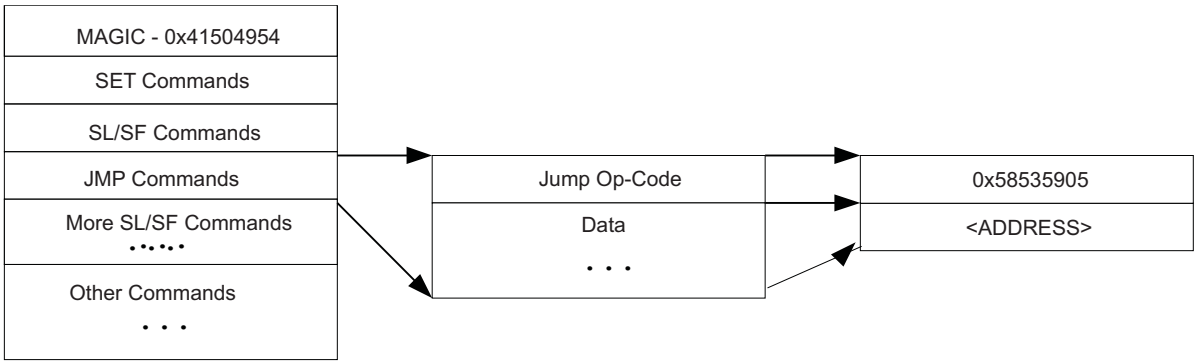

Figure 10. Structure of Jump Command

This command is used to implement bootloader 2. To achieve this, bootloader 2 is loaded through the section load and section fill commands. Once this is done a jump command is issued to start execution from the start address of bootloader2. Once bootloader2 execution is over, normal AIS interpretation and execution continues.

3.6 Jump_Close Command

This command is used at the end of the boot process to start execution of the loaded application. it instructs the DSP to terminate the boot process and jump to start address of loaded application. Figure 11 shows the structure of the Jump_Close command.

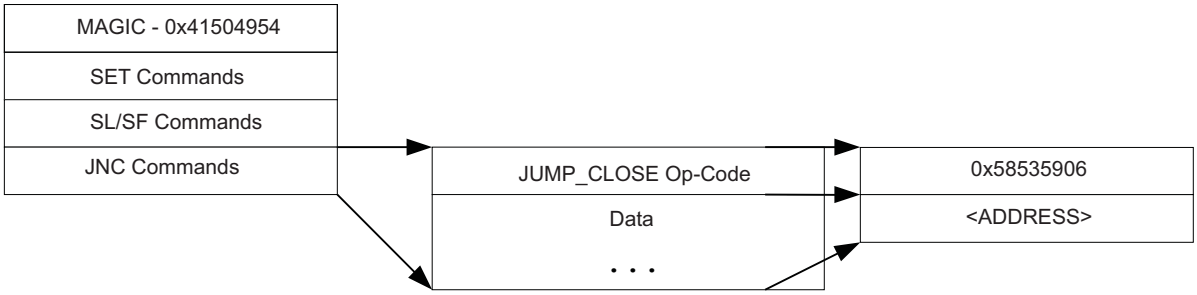

Figure 11. Structure of Jump_Close Command

This command is be placed at the end of AIS, after all other commands. It consists of JUMP_CLOSE (0×58535906) op-code, followed by the start address of the application where the boot loader should jump. In addition to the application entry point address, two words, the 1) total number of sections that should have been loaded during boot, and 2) the total number of bytes which should have been loaded during boot are placed as the last two words of the image.

3.7 CRC Options

There is a possibility of error in communication when the DSP is booting up. Execution of a corrupted application image may result in instability or malfunction. In order to avoid such problems, AIS supports opcodes to verify the validity of data loaded through section load/section fill commands. A proprietary 32-bit CRC computation algorithm is used for verification. The CRC options are implemented by invoking the AIS generation tool with the appropriate option. The tool inserts the CRC enable and CRC requests commands necessary to implement each of the following options:

No CRC— CRC computation is disabled and there is no way to detect or correct any error.

Single CRC— Single CRC is computed for all the sections. Verification is done at the end, just before Jump N Close command. In case of error, all the sections are loaded again; CRC is recalculated and re-verified again at the end.

Section-Wise CRC— CRC is computed for each section. Verification is done at the end of each section and attempt to reload the section is made in case of error.

3.7.1 Enable/Disable CRC Commands

These commands are used to enable/disable computation of the CRC for sections loaded through section load/section fill commands. Figure 12 shows the structure of the enable CRC/disable CRC commands.

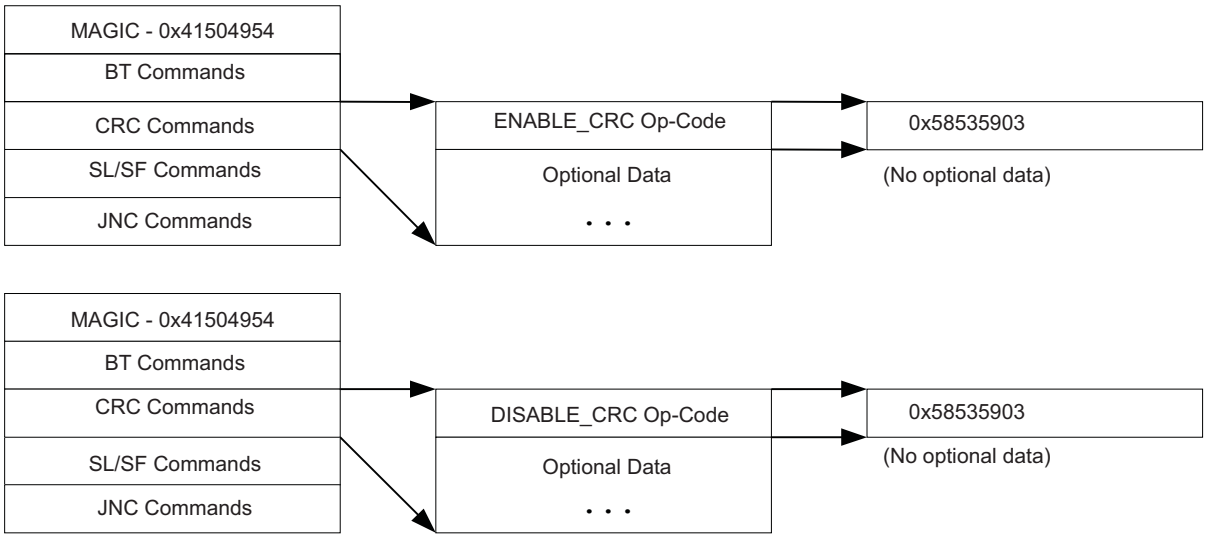

Figure 12. Structure of Enable CRC/Disable CRC Commands

These commands consist of only a single ENABLE_CRC (0×58535903) or DISABLE_CRC (0×58535904) op-code. There is no additional data required.

3.7.2 Request CRC Command

This command is used to request and validate the current value of the CRC computed by the DSP. Using this command requires that the enable CRC command be issued earlier in AIS. This command consists of the REQUEST_CRC (0×58535902) op-code, followed by the expected CRC value and seek-value; the CRC of loaded/filled section(s) are compared with the expected CRC value. If the CRC is correct, seek-value is ignored and execution continues to next command.

A mismatch in the CRC indicates that the data loaded to the DSP memory using earlier section load/section fill commands is corrupted. AIS has to be re-executed from the last known error-free point to load the data again. In order to locate that point, a seek-value is made available as part of the request CRC command. This value is to be interpreted as a negative number and should be added to the current address in AIS. On doing this, the address points to the last error-free point in AIS; execution should be continued as normal from this updated address.

In case of CRC error, the host should indicate the same to the DSP using the start-over command described in Section 3.7.3. After doing that, it should add the seek-value to the AIS address pointer and start executing AIS from that point onwards.

On receiving the start-over command, the DSP knows that the CRC error has occurred. It resets its CRC computation and becomes ready to accept more commands from the host.

Figure 13 shows the structure of the request CRC command.

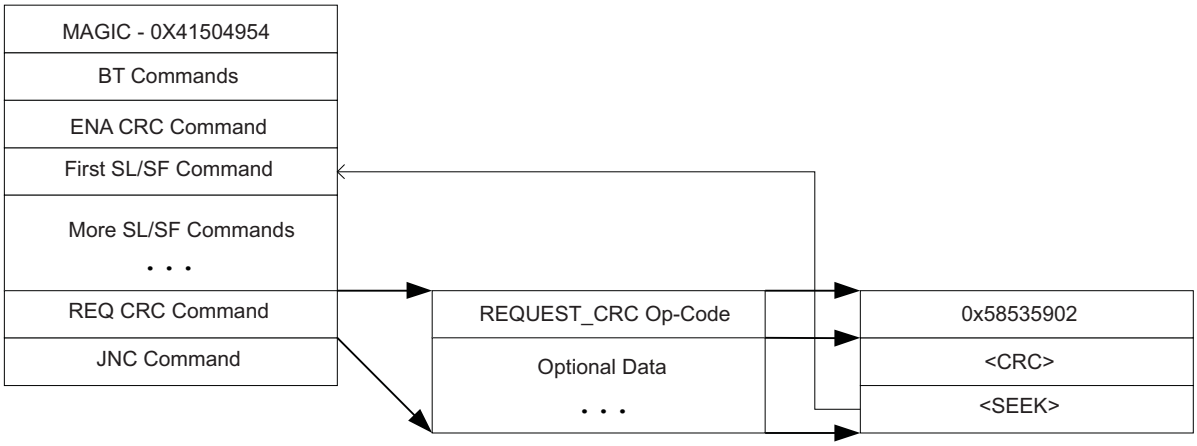

Figure 13. Structure of Request CRC Command

For a single CRC option, this command appears only once in AIS, after the last section load/section fill command. The seek value is interpreted as a negative number, which when added to the current offset in AIS, makes offset point to the start of the first section load/section fill command as shown in Figure 13.

For section-wise CRC option, this command appears after each section load/section fill commands. The seek value is interpreted as a negative number, which when added to the current offset in AIS, makes offset point to the start of the previous section load/section fill command as shown in Figure 13.

3.7.3 Start-Over Command

The start-over command consists of a STARTOVER (0×58535908) op-code with no additional data. This instructs the bootloader to reset its computed CRC value to 0. This command is normally issued by host on its own when it detects a CRC mismatch for slave modes. For master modes, this is taken care of by the bootloader state machine.

3.8 Function Execute Command

Figure 14 shows the structure of the function execute command.

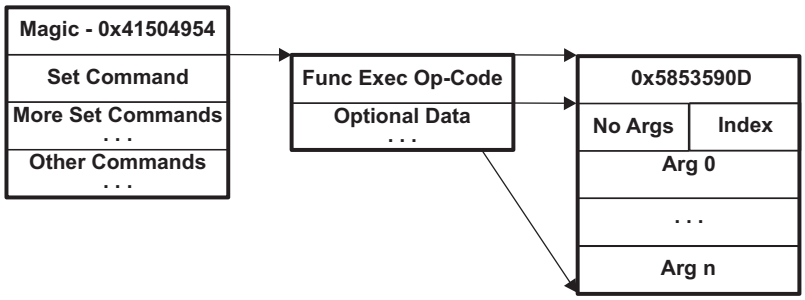

Figure 14. Structure of Function Execute Command

The function execute command allows execution of pre-defined functions that are present in the bootloader ROM code. For the DM643x, the following functions have been pre-defined to enable configurability of the PLL, DDR memory controller, and EMIFA, during the boot process. Please note that the PLL configuration using this command overwrites any PLL configuration that is performed by the bootloader when the FASTBOOT option is selected. Table 23 shows an example pre-defined ROM functions.

Table 23. Pre-Defined ROM Functions

Function	Index	Number of Arguments	Description
PLL Config	0	3	Programs PLL
EMIFA Config	1	5	Programs EMIF control registers
DDR Config	2	9	Programs DDR PLL and DDR memory controller sets the DDR control registers

When creating the command sequence for the function execute command, the upper 16 bits of the word immediately following the command opcode contains the number of arguments required by the function, and the lower 16 bits must contain the function index.

3.8.1 PLL Config Function

The PLL config function enables re-programming of the PLL beyond what is selectable by the FASTBOOT option. The PLL config function requires three arguments and they must be given in the order shown:

1. PLL multiplier
2. PLL divide 1 (divide down for CPU/system clock)
3. Oscillator source (0-internal , 1 - external)

Table 24 shows an example function execute command for PLL config.

Table 24. Sample Function Execute Command

AIS Data	Description
0x5853590D	Function Execute Opcode
0x00030000	3 arguments, Function index = 0
0x00000019	PLLM = 0x19
0x00000001	PLLDIV1 (CPU/Sysclk) = 1, divide of 2
0x00000000	Internal oscillator

3.8.2 EMIFA Config Function

The EMIFA config function takes five arguments, given in the following order:

1. AB1CR control register value
2. AB2CR control register value
3. AB3CR control register value
4. AB4CR control register value
5. NANDFCR control register value

Please note that this function does not override the AEM and AEAW pin settings that are latched at device reset.

3.8.3 DDR Config Function

The DDR memory controller config function requires nine arguments. The arguments must be given in the order shown:

1. DDR PLLM value
2. DDR CLK divide down
3. Video processing back end (VPBE) CLK divde down
4. DDR clock source (0-internal, 1-external)
5. DDR control register value
6. Synchronous dynamic random access memory (SDRAM) config register value
7. SDRAM timer 0 register value
8. SDRAM timer 1 register value
9. SDRAM refresh control register value

The DDR memory controller config function, programs the DRR PLL and then configures the DDR memeory controller using the register settings given in the function execute command. It performs a single write/read to the start of the DDR memory controller space to confirm the DDR memory controller operation.

4 Booting Operating Systems (Linux®/DSP/BIOS™,etc.)

The ROM bootloader operates independent of boot modes provided by specific operating systems. The boot-startup code for any operating system must be in a format in compliance with the ROM bootmodes described in the previous sections. The ROM bootloader views all operating system start-up code no different than any other application code. Therefore, if the operating system requires any specialized formats to boot the preponderance of its code, this must be done via secondary boot. The secondary bootloader for the operating system must be presented in the appropriate format for the ROM bootloader to properly load its code. After loading the operating system boot code (secondary boot, if necessary), the ROM bootloader branches to the operating system startup/boot-up. If a secondary bootloader was required, the secondary bootloader then completes the download of the rest of the operating system and begins execution.

Please note for this scenario, only the secondary bootloader MUST follow the appropriate ROM bootloader protocol for the boot mode chosen. The rest of the operating system code/data may be in any format required for the secondary boot to complete load of the system.

For example, if using universal boot for the uCLinux operating system, only the code for u-boot itself would need to be in AIS format, if booting from SPI/I2C, Fast EMIF, etc. The remaining code/data for the uCLinux operating system would be in the compressed format expected by u-boot. u-boot would then uncompress and load the remainder of the uCLinux code to DSP memory.

5 ROM Bootloader RAM Memory Requirements and Code/Data Placement

The ROM bootloader uses a small amount of RAM in the internal memory space of the device for stack and temporary buffer/data storage space. Memory is allocated in the first lower 20 K bytes of the L1D (data) CACHE for this purpose. Applications MUST NOT link any initialized code/data sections into this area of memory. Doing so may result in overwriting of essential data used by the bootloader to effect boot; this causes boot to fail. Un-initialized sections such as the compiler generated sections , .bss, and .far, can be allocated to this area, since these are not populated until after the boot process is complete and the application starts to run. Also, note that the bootloader uses CPU writes to copy downloaded code/data to memory. Becasue of this, the bootlaoder cannot directly load code into the L1P program space. The application must actively populate this space once it has been downloaded.

6 ROM Bootloader Cache Considerations

The ROM bootloader disables all cache for L2 RAM and L1 RAM (both L1 Data and L1 Program) during the boot process. If cache is enabled during the boot process via AIS commands, then be aware that the bootloader code disables cache once again after the application code is fully loaded and prior to the branch to application start. Therefore, the application code must explicitly enable cache, if cache use is required. The application cannot assume cache is in default power on state, especially if cache was enabled during boot. The bootloader does not restore cache registers to their power on defaults; it simply disables the cache upon exit.

7 AIS Generation Tool , DM643x

The DM643x is a Perl script that converts a linked executable for the DM643x to the appropriate format for the given boot mode and data/memory widths. The DM643x is a command line tool and may be invoked as part of a larger script or Make file. The current version of DM643x was developed using Active Perl V5.8.6.

A simple invocation of the DM643x includes the name of the application executable file, the name of the AIS output file, the type of the output file, the boot mode, and the data or address/memory width of the device where the image is stored.

For example:

DM643x –i MyApplication.out –o MyApplication.ais –bootmode spi –otype ascii –addrsz 16

This invocation would produce a converted ASCII AIS file formated for the SPI boot. The AIS generation tool can produce either an ASCII, binary, plain text, or asm output file. The asm output file contains the AIS image in the form of assembly .word directives. This assembly file may then be assembled/linked and passed to the Hex Conversion Utility for use with an EEPROM burner. A list of available options for the DM643x tool is shown in Table 25.

WARNING

Please note that the genAIS tool has a dependency on the OFD utility (ofd6x,ofd6x.exe) that is provided with the *TMS320C6000 Code Generation Tools Installation Instructions* (SPRU237). The genAIS utility currently requires ofd6x v6.1.0A06333 or above. genAIS uses the Perl *system()* function to invoke the OFD utility. Some verions of the Windows®NT and Windows95 operating systems, may not suppport use of the *system()* function. In this situation, the OFD utility must be run prior to invoking genAIS and the resulting XML file must be specified as an input on the genAIS command line. The options shown below in invoking the OFD utility are optimal for usage with genAIS, and represent the minimal set required. Therefore, it is recommended to use this set of options when invoking the OFD.

for example,

ofd6x -x --obj_display=none,header,optheader,sections,symbols myApp.out -o myApp.xml

genAIS -i myApp.out -x myApp.xml -o myAIS.txt -bootmode uart -otype txt

(For more information about the OFD, please refer to *TMS320C6000 Assembly Language Tools User's Guide*, SPRU186).

Table 25. DM643x Program Options

Option	Description
-i filename	Specifies input executable file
-o filename	Specifies name of the AIS output file
-x filename	Specifies name of XML output from the OFD tool (ofd6x)
-crc N (N = 0,1,2)	Selects CRC generation:
	N = 0 - no CRC generation
	N = 1 - CRC generated for each section load
	N = 2 - single CRC generated for entire load
-bootmode N (N=i2c, spi, uart. nand, raw)	Specifies boot mode for which conversion is to be generated:
	Please note that *raw* generates an AIS image that is mode independent
-otype N (N=ascii, binary, txt, asm)	Specifies content format for AIS output
-memwidth N (N=8,16,24)	Specifies memory/address width for external memories associated with the I2C and SPI bootmodes. Please note that the memory width of 16 bits is the only valid memory type for the I2C for the DM643x and C6423/C6421 devices.
-datawidth N (N=8,16)	Specifies NOR flash data access width for the EMIF FASTBOOT option. Please note that selecting this option is NOT a substitute for setting the proper EMIF 8_16-bit pin on the device when booting from EMIF.
-cfg	Specifies the name of an optional configuration file that contains a sequence of set or function execute commands to be included at the beginning of the AIS output file.
-addrsz	SPI EEPROM addres width in bits, i.e., 16, 24

8 Sample AIS Boot Images

AIS data streams are required for fast EMIFA, SPI, I2C, NAND Flash, and UART boot modes. A sample AIS stream for each of these modes is presented in this section. The AIS boot images in this section were created using a single tool called, DM643x. The DM643x is a Perl script that converts an application linked executable file to an AIS boot image file for the bootmode selected. DM643x is discussed in the next section. All boot images generated in this section use the same sample assembly source shown in Example 1.

Example 1. Sample Source Code for AIS Examples

```
;======================================
;  Sample Assembly Source File
;  a = 6;
;  while(1) {
;    b = a + 1;
;    c = b + 2;
;  }
;
;======================================
           .global     _a,_b,_c
           .sect       "myData"
_a         .word 0xA
_b         .word 0xB
_c         .word 0xC

        .text
             .global Start
Start:
                  MVKL     .S1       _a,A3
                  MVKL     .S1       _c,A5
                  MVKL     .S1       _b,A4
                  MVKH     .S1       _a,A3
||                MVK      .S2       6,B4
                  STW      .D1T2     B4,*A3
```

Example 1. Sample Source Code for AIS Examples (continued)

```
||                MVKH     .S1       _c,A5
                  MV       .L2X      A3,B5
||                MVKH     .S1       _b,A4

loop:
                  LDW      .D2T2    *B5,B4
                  NOP                4
                  ADD      .L2       1,B4,B4
                  STW      .D1T2     B4,*A4
                  NOP                2
                  LDW      .D1T1     *A4,A3
                  NOP                4
                  ADD      .L1       2,A3,A3
                  STW      .D1T1     A3,*A5
                  NOP                2
                  B        .S1       loop
                  NOP                5
```

8.1 AIS Boot Image for EMIFA ROM Boot

The first 8-bit byte in the FLASH/ROM accessed via EMIFA MUST contain the EEPROM size. Valid values are 0×00 → 8 bit, 0×01 → 16 bit. The next three bytes are reserved. The first valid AIS word begins on the next 32-bit word boundary. This word MUST contain the AIS magic word, 0×41504954. Any valid AIS command may appear after the magic word. Table 20 shows the sample data stream for a 16-bit FLASH, using the sample source included at the start of this section

Table 26. EMIFA ROM Fast Boot AIS Boot Image Example

Data	Explanation
0x00000001	First byte of word specifies external memory data width
0x41504954	AIS Magic Number
0x58535903	Enable CRC Command
0x58535901	Section Load Command
0x10800000	Section Load Address
0x00000040	Section Size in Bytes
0x01802028	Start of Raw Section Data
0x02802428	
0x02002228	
0x01884069	
0x0200032A	
0x020C0277	
0x02884068	
0x028C1FDB	
0x02084068	
0x6C6E10CD	
0x10442641	
0x003C2C6E	
0x45B06C6E	
0x2C6E00B4	
0x8C6E008A	
0xEFC08000	End of Raw Section Data
0x58535902	Request CRC Command
0x0E85A97B	Expected CRC Value
0xFFFFFFA8	Negative Pointer to Last Valid Command in Stream
0x58535901	Section Load Command
0x10800040	Section Load Address
0x0000000C	Section Size in Bytes
0x0000000A	Start of Section Raw Data
0x0000000B	
0x0000000C	End of Section Raw Data
0x58535902	Request CRC Command
0x8434A250	Expected CRC Value
0xFFFFFFDC	Negative Pointer to Last Valid Command in Stream
0x58535906	Jump Close Command
0x10800000	Application Entry Point Address
0x00000002	Total number of sections that should have been loaded
0x0000004C	Total number of bytes that should have been loaded

8.2 AIS Boot Image for I2C Boot

The first 32-bit word on the AIS header for the I2C boot mode is reserved and is ignored by the bootloader. The second 32-bit word MUST contain the AIS magic number. A sample AIS image for I2C is shown in Table 26.

Table 27. I2C AIS Boot Image Example

Data	Explanation
0x00000002	Reserved for DM643x – boot loader ignores
0x41504954	AIS Magic Number
0x58535903	AIS Magic Number
0x58535901	Section Load Command
0x10800000	Section Load Address
0x00000040	Section Size in Bytes
0x01802028	Start Section Raw Data
0x02802428	
0x02002228	
0x01884069	
0x02884068	
0x028C1FDB	
0x02084068	
0x6C6E10CD	
0x10442641	
0x003C2C6E	
0x45B06C6E	
0x2C6E00B4	
0x8C6E008A	
0xEFC08000	End Section Raw Data
0x58535902	Request CRC Command
0x0E85A97B	Expected CRC Value
0xFFFFFFA8	Negative Pointer to Last Valid Command
0x58535901	Section Load Command
0x10800040	Section Load Address
0x0000000C	Section Size in Bytes
0x0000000A	Start of Section RAW Data
0x0000000B	
0x0000000C	End Section Raw Data
0x58535902	Request CRC Value
0x8434A250	Expected CRC Value
0xFFFFFFDC	Negative Pointer to Last Valid Command
0x58535906	Jump Close Command
0x10800000	Application Entry Point Address
0x00000002	Total number of sections that should have been loaded
0x0000004C	Total number of bytes that should have been loaded

Table 28 details the expected byte arrangement of the AIS boot image in the I2C EEPROM.

Table 28. AIS Image in I2C EEPROM Memory

Byte Address	Byte0	Byte1	Byte2	Byte3	32-Bit AIS Data	Explanation
0x0000	0x02	0x00	0x00	0x00	0x00000002	First byte contains address size in bytes – IGNORED by bootloader for this device
0x0004	0x54	0x49	0x50	0x41	0x41504954	AIS Magic Word
0x0008	0x03	0x59	0x53	0x58	0x58535903	Enable CRC Command
0x000C	0x01	0x59	0x53	0x58	0x58535901	Section Load Command
0x0010	0x00	0x00	0x80	0x10	0x10800000	Section Load Address
0x0014	0x40	0x00	0x00	0x00	0x00000040	Section Size in Bytes
0x001C	0x28	0x20	0x80	0x01	0x01802028	Start Section Raw Data
0x0020	0x28	0x24	0x80	0x02	0x02802428	
0x0024	0x28	0x22	0x00	0x02	0x02002228	
0x0028	0x69	0x40	0x88	0x01	0x01884069	
0x008C					0x58535906	JUMP CLOSE Command
0x0090					0x10800000	Application Entry Point Address
0x0094					0x00000002	Total Number of Sections
0x0098					0x0000004C	Total Number of Bytes

8.3 AIS Boot Image for SPI Boot

The AIS boot image for SPI is exactly the same as I2C with the exception that the first 32-bit word in the AIS image must contain the address width of the the SPI EEPROM expressed in bytes. The byte containing the address width MUST be located at address *0* of the EEPROM. This address width byte is included for internal use of the bootloader.

Table 29. SPI AIS Boot Image Example

Data	Explanation
0x00000002	EEPROM Address Width in Bytes - Please note this value will be 0x00000003 in case of 24 Bit SPI
0x41504954	AIS Magic Number
0x58535903	Request CRC Command
0x58535901	Section Load Command
0x10800000	Section Load Address
0x00000040	Section Size in Bytes
0x01802028	Start Section Raw Data
0x02802428	
0x02002228	
0x01884069	
0x0200032A	
0x020C0277	
0x02884068	
0x028C1FDB	
0x02084068	
0x6C6E10CD	
0x10442641	
0x003C2C6E	

Table 29. SPI AIS Boot Image Example (continued)

Data	Explanation
0x45B06C6E	
0x2C6E00B4	
0x8C6E008A	
0xEFC08000	End Section Raw Data
0x58535902	Request CRC Command
0x0E85A97B	Expected CRC Value
0xFFFFFFA8	Negative Pointer to Last Valid Command
0x58535901	Section Load Command “myData section”
0x10800040	Section Load Address
0x0000000C	Section Size in Bytes
0x0000000A	Start Section Raw Data
0x0000000B	
0x0000000C	End Section Raw Data
0x58535902	Request CRC Command
0x8434A250	Expected CRC Value
0xFFFFFFDC	Negative Pointer to Last Valid Command
0x58535906	Jump Close Command
0x10800000	Application Entry Point Address
0x00000002	Total number of sections that should have been loaded
0x0000004C	Total number of bytes that should have been loaded

Please note that the byte ordering of data as stored in the EEPROM should be as follows using the AIS data from Table 28 as an example.

Table 30. AIS Image in SPI EEPROM Memory

Byte Address	Byte0	Byte1	Byte2	Byte3	32-Bit AIS Data	Explanation
0x0000	0x02	0x00	0x00	0x00	0x00000002	First byte contains address size in bytes
0x0004	0x54	0x49	0x50	0x41	0x41504954	AIS Magic Word
0x0008	0x03	0x59	0x53	0x58	0x58535903	Enable CRC Command
0x000C	0x01	0x59	0x53	0x58	0x58535901	Section Load Command
0x0010	0x00	0x00	0x80	0x10	0x10800000	Section Load Address
0x0014	0x40	0x00	0x00	0x00	0x00000040	Section Size in Bytes
0x001C	0x28	0x20	0x80	0x01	0x01802028	Start Section Raw Data
0x0020	0x28	0x24	0x80	0x02	0x02802428	
0x0024	0x28	0x22	0x00	0x02	0x02002228	
0x0028	0x69	0x40	0x88	0x01	0x01884069	
0x008C					0x58535906	JUMP CLOSE Command
0x0090					0x10800000	Application Entry Point Address
0x0094					0x00000002	Total Number of Sections
0x0098					0x0000004C	Total Number of Bytes

8.4 AIS Boot Image for UART Boot

UART boot mode differs from the previous modes in that some communication is carried out between the DSP and HOST in addition to transfer of AIS commands. The DSP UART acts as slave in the boot process. But, to alert the HOST that the DSP is alive and ready to receive, it sends the initial message BOOT ME to the HOST. As acknowledgment, the HOST then begins sending the AIS boot image, beginning with the AIS magic number. The AIS data is sent as ASCII text. The bootloader software converts to the equivalent hexadecimal constant.

The bootloader continues to process AIS commands transmitted by the HOST until the JUMP CLOSE command is encountered. After the JUMP CLOSE command is received, the bootloader sends the message DONE to the HOST. This signals the HOST that boot has successfully completed.

DSP					HOST
SENDS	→	“BOOT ME”		→	
	←		“41”	←	SENDS first byte of AIS Magic #
	←		“50”	←	SENDS second byte of AIS Magic #
	←		“49”	←	SENDS third byte of AIS Magic #
	←		“54”	←	SENDS last byte of AIS Magic #
	←		“58”	←	SENDS first byte of AIS command
	←		“53”	←	SENDS second byte of AIS command
	←		“59”	←	SENDS third byte of AIS command
	←		“03”	←	SENDS last byte of AIS command
				←	HOST continues to SEND commands and data until JUMP CLOSE command is issued
	←		“58”	←	SENDS first byte of JUMP CLOSE
	←		“53”	←	SENDS second byte of JUMP CLOSE
	←		“59”	←	SENDS third byte of JUMP CLOSE
	←		“06”	←	SENDS last byte of JUMP CLOSE
	←		“10”	←	SENDS first byte of entry point address
	←		“80”	←	SENDS second byte of entry point address
	←		“00”	←	SENDS third byte of entry point address
	←		“00”	←	SENDS last byte of entry point address
	←		“00”	←	SENDS first bye of section count
	←		“00”	←	SENDS second byte of section count
	←		“00”	←	SENDS third byte of section count
	←		“02”	←	SENDS last byte of section count
	←		“00”	←	SENDS first byte of byte count
	←		“00”	←	SENDS second byte of byte count
	←		“00”	←	SENDS third byte of byte count
	←		“4C”	←	SENDS last byte of byte count
SENDS	→	“ DONE”		→	

At this point the boot process is complete and the bootloader branches to the application start address. If an error occurs, for example a CRC error, the bootloader issues a message CORRUPT to the host and places an error condition in the ERR field of the BOOTCMPLT register. It then re-attempts boot.

The AIS boot image for UART is an ASCII string with no spaces or carriage returns between elements (see Figure 15).

```
41504954585359035853590110800000000000400180202802802428020022280188406902000
32A020C027702884068028C1FDB020840686C6E10CD10442641003C2C6E45B06C6E2
C6E00B48C6E008AEFC08000585359020E85A97BFFFFFFA858535901108000400000000
C0000000A0000000B0000000C585359028434A250FFFFFFDC5853590610800000000000000
20000004C
```

Figure 15. UART AIS Boot Image

8.5 AIS Boot Image for NAND Boot

AIS boot image for NAND boot is very similar to all the others seen so far, with exceptions for three words that define the starting block and number of pages where AIS image is stored. Since this is not known until the data is actually written to the NAND device, it is your responsibility to fill in these three fields in the AIS data. The DM643x tool leaves space for these in Table 31 generated as place holders for real values to be encoded later, when image is finally written to the NAND.

Table 31. NAND Boot AIS Boot Image Example

Data	Explanation
0x41504954	AIS magic number
0x00000000	Place holder reserved for number of pages over which image spans
0x00000000	Place holder for block where image starts
0x00000000	Place holder for page on which image starts
0x58535903	Enable CRC command
0x58535901	Section load command
0x10800000	Section load address
0x00000040	Section size in bytes
0x01802028	Start of section raw data
0x02802428	
0x02002228	
0x01884069	
0x0200032A	
0x020C0277	
0x02884068	
0x028C1FDB	
0x02084068	
0x6C6E10CD	
0x10442641	
0x003C2C6E	
0x45B06C6E	
0x2C6E00B4	
0x2C6E00B4	

Table 31. NAND Boot AIS Boot Image Example (continued)

Data	Explanation
0xEFC08000	End of section raw data
0x58535902	Request CRC command
0x0E85A97B	Expected CRC value
0xFFFFFFA8	Negative pointer to last valid command
0x58535901	Section load command
0x10800040	Section load address
0x0000000C	Section size in bytes
0x0000000A	Start section raw data
0x0000000B	
0x0000000C	End section raw data
0x58535902	Request CRC command
0x8434A250	Expected CRC value
0xFFFFFFDC	Negative pointer to last valid command
0x58535906	JUMP CLOSE command
0x10800000	Application entry point address
0x00000002	Total number of sections that should have been loaded
0x0000004C	Total number of sections that should have been loaded

8.6 Configuration Data File

By using the –cfg option, a sequence of set or function execute commands can be included at the beginning of the AIS output data file. This allows the option to configure the DDR memory controller, EMIF, or PLL to enable proper boot from/to external memories. The commands in this file precede any other AIS data that is generated. Please note that the data in the configuration file is not parsed by the genAIS tool; it is simply passed directly through to the output file. Care must be taken to ensure that a correct data sequence appears in the file. A sample configuration file that calls the ROMed configuration functions for the PLL, EMIF, and DDR memory controller is shown below.

```
0x5853590D  #   Function Execute Command
0x00030000  #   Selects PLL configuration function, with 3 arguments
0x00000015  #   PLLM value
0x00000000  #   PLLDIV 0
0x00000000  #   Clock source
0x5853590D  #   Function Execute Command
0x00050001  #   Selects EMIFA configuration, with 5 arguments
0x3FFFFFFC  #   AB1CR control register mask
0x3FFFFFFC  #   AB2CR control register mask
0x3FFFFFFC  #   AB3CR control register mask
0x3FFFFFFC  #   AB4CR control register mask
0x00000000  #   NANDFCR control register mask
0x5853590D  #   Function Execute Command
0x00090002  #   Selects DDR memory configuration, with 9 arguments
0x00000017  #   DDR PLLM
0x00000001  #   PLL SRC
0x0000000B  #   DDR CLLK DIV
0x00000000  #   VPBE CLK DIV
0x50006405  #   DDR Control register mask
0x00138822  #   SDRAM Config register mask
0x16492148  #   SDRAM Timer 0 register mask
0x000CC702  #   SDRAM Timer 1 register mask
0x000004EF  #   SDRAM Refresh control register mask
```

9 Determining On-Chip Bootloader Version

The bootloader version can be found by reading the ROM location 0×0101A00. More than one ROM version is extant at this time. ROM version *0×27B2A120* supports EMIFA direct ROM boot only. No other boot mode should be selected, when using this version. FASTBOOT option is also NOT supported by that ROM version. ROM versions *0×00010200* and *0×0010300* support all the features delineated within this document, including FASTBOOT.

10 Calculating CRC

The on-chip bootloader uses a 32-bit CRC. Code for calculating the CRC is given in the Appendix A. The CRC as calculated for the on-chip bootloader requires three calls to the BL_updateCrc function. The first call is made sending the section load address as the data word. The second call uses the section size in bytes as the data word. The third call sends the actual section data, calculating a CRC across all the data elements in the section. So the final CRC is a combination of the CRC's calculated for section address, section size and section data. A sample set of calls to the function to create the expected CRC value is shown below:

```
 unsigned int
crc;
 unsigned int sectionAddr;
 unsigned int sectionSize;

unsigned int *sectionData;
 crc =
BL_updateCRC(&sectionAddr, 4, 0);
 crc =
BL_updateCRC(&sectionSize, 4, crc);
 crc =
BL_updateCRC(sectionData, sectionSize, crc);
```

The last calculated *crc* value should be written as the expected CRC for the REQUEST_CRC command. If calculating a single CRC for the entire application load, simply pass each successive *crc* value into the subsequent calls to BL_updateCRC.

```
typedef struct {
 unsigned int sectionAddr;

unsigned int sectionSize;
 unsigned int
*sectionData;
 } SectionDatObj;
 SectionDataObj mySections[10];

unsigned int crc;
 crc = 0;

for(i=0;i<10;i++) {
 crc =
BL_updateCRC(&(mySections[i].sectionAddr), 4, crc);
 crc = BL_updateCRC(&(mySections[i].sectionSize), 4,
crc);
 crc = BL_updateCRC(mySections[i].sectionData,
mySections[i].sectionSize, crc);

}
```

Appendix A Calculating the CRC

The CRC calculated to process the REQUEST_CRC command is based on the following algorithm, where *data_ptr* points to the first data element in the current section, *section_size* is the size of the section expressed in 8-bit bytes, and *crc* is current CRC value.

```
unsigned int updateCRC(unsigned int *data_ptr, unsigned int section_size, unsigned int crc)
{
            unsigned int n, crc_poly = 0x04C11DB7; /* CRC - 32 */
            unsigned int msb_bit;
            unsigned int residue_value;
            int bits;

            for( n = 0; n < (section_size>>2); n++ )
            {
                        bits = 32;
                        while( --bits >= 0 )
                        {
                                    msb_bit = crc & 0x80000000;
                                    crc = (crc << 1) ^ ( (*data_ptr >> bits) & 1 );
                                    if ( msb_bit )
                        crc = crc ^ crc_poly;
                        }
                        data_ptr ++;
            }

            switch(section_size & 3)
            {
                        case 0:
                                    break;
                        case 1:
                                    residue_value = (*data_ptr & 0xFF) ;
                                    bits = 8;
                                    break;
                        case 2:
                                    residue_value = (*data_ptr & 0xFFFF) ;
                                    bits = 16;
                                    break;
                        case 3:
                                    residue_value = (*data_ptr & 0xFFFFFF) ;
                                    bits = 24;
                                    break;
            }

            if(section_size & 3)
            {

                        while( --bits >= 0 )
                        {
                                    msb_bit = crc & 0x80000000;
                                    crc = (crc << 1) ^ ( (residue_value >> bits) & 1 );
                                    if ( msb_bit ) crc = crc ^ crc_poly;
                        }
            }
            return( crc );
}
```

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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