

# Using the TMS320C642x Bootloader

Daniel Allred

## ABSTRACT

This document describes the functionality of the C642x 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 that is explained in detail in [Section 3](#) of this document.

Project collateral and source code discussed in this application report can be downloaded from the following URL: <http://www.ti.com/lit/zip/SPRAAK5>.

## Contents

|                                      |  |    |
|--------------------------------------|--|----|
| 1                                    | Introduction .....   | 3  |
| 2                                    | Boot Mode Description.....   | 3  |
| 3                                    | Application Image Script .....                                       | 17 |
| 4                                    | Bootting Operating Systems ( Linux®/ DSP/BIOS™,etc.) .....           | 28 |
| 5                                    | ROM Bootloader RAM Memory Requirements and Code/Data Placement ..... | 29 |
| 6                                    | ROM Bootloader Cache Considerations.....                             | 29 |
| 7                                    | AIS Generation Tool, genAIS.....                                     | 29 |
| 8                                    | Sample AIS Boot Images.....  | 31 |
| 9                                    | Debugging Boot Failures .....  | 39 |
| 10                                   | Determining On-Chip Bootloader Version.....                          | 39 |
| 11                                   | Calculating CRC .....  | 40 |
| 12                                   | References.....  | 40 |
| Appendix A Calculating the CRC ..... |  | 41 |

## List of Figures

|    |  |    |
|----|--|----|
| 1  | Signal Connections for I2C EEPROM Boot Mode .....  | 10 |
| 2  | SPI Transfer With CLKSTP = 11 and CLKXP = 0 .....  | 13 |
| 3  | 24x8 Bit SPI EEPROM Read Timing .....              | 16 |
| 4  | DM643x 24x8 Bit Address SPI Boot .....             | 17 |
| 5  | Basic Structure of Application Image Script.....   | 18 |
| 6  | Structure of SET Command.....                      | 19 |
| 7  | Valid SET Command Data Types .....                 | 20 |
| 8  | Structure of GET Command .....                     | 21 |
| 9  | Structure of Section Load Command .....            | 21 |
| 10 | Structure of Section Fill Command .....            | 22 |
| 11 | Structure of Jump Command .....                    | 22 |
| 12 | Structure of Jump_Close Command.....               | 23 |
| 13 | Structure of Enable CRC/Disable CRC Commands ..... | 24 |
| 14 | Structure of Request CRC Command.....              | 25 |
| 15 | Structure of Function Execute Command .....        | 26 |
| 16 | UART AIS Boot Image.....                           | 37 |

## List of Tables

|    |   |    |
|----|---|----|
| 1  | Terms and Abbreviations .....   | 3  |
| 2  | Non-Fastboot Modes (FASTBOOT = 0) .....   | 4  |
| 3  | User-Select Multiplier Fastboot Modes (FASTBOOT = 1).....   | 5  |
| 4  | PLL Multiplier Selection (PLLMS[2:0]) in User-Select Multiplier Fastboot Modes (FASTBOOT = 1) ..... | 5  |
| 5  | PLL Multiplier Based on Value AEM and PLLMS[2:0] Pins.....  | 6  |
| 6  | PLL1 and PLL2 Multiplier Ranges.....  | 7  |
| 7  | PLL1 and PLL2 Multiplier Ranges.....  | 7  |
| 8  | PLL2 Clock Frequency Ranges .....   | 7  |
| 9  | PLL2 Clock Frequency Ranges .....   | 7  |
| 9  | CPU Frequency During FASTBOOT .....   | 7  |
| 10 | I2C EEPROM Layout for PCI Autoinitialization Data.....  | 9  |
| 11 | I2C Timing Register Settings.....   | 11 |
| 12 | SPI Master Clock Frequencies for FASTBOOT = 1 .....   | 12 |
| 13 | SPI Master Boot Modes .....   | 13 |
| 14 | SPI 16x8 EEPROM-to-DSP McBSP0 Connection .....  | 13 |
| 15 | Supported NAND Device Types .....   | 14 |
| 16 | UART Connection Attributes for Boot.....  | 15 |
| 17 | SPI EEPROM and DSP Pin Connections for 24 Bit SPI Mode .....  | 17 |
| 18 | AIS Version 2.0 Supported Opcodes .....   | 19 |
| 19 | Numeric Formats That Can Be Used in SET Command.....  | 20 |
| 20 | Valid SET Command Data Types .....  | 20 |
| 21 | Valid SET Command Data Types Field Descriptions .....   | 20 |
| 22 | Pre-Defined ROM Functions .....   | 26 |
| 23 | Sample Function Execute Command .....   | 26 |
| 24 | C642x Program Options .....   | 30 |
| 25 | EMIFA ROM Fast Boot AIS Boot Image Example .....  | 32 |
| 26 | I2C AIS Boot Image Example.....   | 33 |
| 27 | AIS Image in I2C EEPROM Memory.....   | 34 |
| 28 | SPI AIS Boot Image Example .....  | 34 |
| 29 | AIS Image in SPI EEPROM Memory .....  | 35 |
| 30 | NAND Boot AIS Boot Image Example.....   | 37 |
| 31 | Debugging Boot Failures.....  | 39 |

## 1 Introduction

**Table 1. Terms and Abbreviations**

| Term       | Description  |
|------------|--|
| Bootloader | SW/Code for ROM C642x Bootloader                           |
| AIS        | Application Image Script                                   |
| BL         | Boot Loader (referring to the bootloader in this text)     |
| DSP        | Digital Signal Processor (referring to C642x 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                     |
| 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](#).  
**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 3](#).

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.

DSP/BIOS is a trademark of Texas Instruments.

Linux is a registered trademark of Linux Torvalds in the U.S. and/or other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries.

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

| DEVICE BOOT AND CONFIGURATION PINS |         | BOOT DESCRIPTION <sup>(1)</sup>                 | C642x DMP (Master/Slave) | PLL1 CLOCK SETTING AT BOOT |                                  |                            | DSPBOOTADDR (DEFAULT) <sup>(1)</sup> |
|------------------------------------|---------|---|--------------------------|----------------------------|----------------------------------|----------------------------|--------------------------------------|
| BOOTMODE[3:0]                      | PCIE1EN |   |                          | PLL MODE <sup>(2)</sup>    | 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] <sup>(3)</sup>         | 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                          |

<sup>(1)</sup> 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.

<sup>(2)</sup> 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.

<sup>(3)</sup> 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. User-Select Multiplier Fastboot Modes (FASTBOOT = 1)**

| DEVICE BOOT AND CONFIGURATION PINS |         | BOOT DESCRIPTION <sup>(1)</sup>                 | C642x DMP (Master/Slave) | PLL1 CLOCK SETTING AT BOOT |                                  |                            | DSPBOOTADDR (DEFAULT) <sup>(1)</sup> |
|------------------------------------|---------|---|--------------------------|----------------------------|----------------------------------|----------------------------|--------------------------------------|
| BOOTMODE[3:0]                      | PCIE1EN |   |                          | PLL MODE <sup>(2)</sup>    | 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 4                    | /2                               | Table 4                    | 0x0010 0000                          |
| 0010                               | 0       | HPI Boot  | Slave                    | Table 4                    | /2                               | Table 4                    | 0x0010 0000                          |
|                                    | 1       | PCI Boot with Auto Initialization               | Slave                    | Table 4                    | /2                               | Table 4                    | 0x0010 0000                          |
| 0011                               | 0 or 1  | Reserved  | –                        | –                          | –                                | –                          | –                                    |
| 0100                               | 0 or 1  | EMIFA ROM FASTBOOT with AIS                     | Master                   | Table 4                    | /2                               | Table 4                    | 0x0010 0000                          |
| 0101                               | 0 or 1  | I2C Boot [FAST MODE] <sup>(3)</sup>             | Master                   | Table 4                    | /2                               | Table 4                    | 0x0010 0000                          |
| 0110                               | 0 or 1  | 16-bit SPI Boot [McBSP0]                        | Master                   | Table 4                    | /2                               | Table 4                    | 0x0010 0000                          |
| 0111                               | 0 or 1  | NAND Flash Boot                                 | Master                   | Table 4                    | /2                               | Table 4                    | 0x0010 0000                          |
| 1000                               | 0 or 1  | UART Boot without Hardware Flow Control [UART0] | Master                   | Table 4                    | /2                               | Table 4                    | 0x0010 0000                          |
| 1001                               | 0 or 1  | EMIFA ROM FASTBOOT without AIS                  | Master                   | Table 4                    | /2                               | Table 4                    | –                                    |
| 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 4                    | /2                               | Table 4                    | 0x0010 0000                          |
| 1111                               | 0 or 1  | 24-bit SPI Boot (McBSP0 + GP[97])               | Master                   | x20                        | /2                               | CLKIN x20 / 2              | 0x0010 0000                          |

<sup>(1)</sup> 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.

<sup>(2)</sup> Any *supported* PLL MODE is available. [See Table 4 for supported C642x PLL MODE options].

<sup>(3)</sup> 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. PLL Multiplier Selection (PLLMS[2:0]) in User-Select Multiplier Fastboot Modes (FASTBOOT = 1)**

| DEVICE BOOT AND CONFIGURATION PINS |          | PLL1 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. Because 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 5](#). For more detailed description of these settings and associated timings, see the device-specific data sheet.

**Table 5. PLL Multiplier Based on Value AEM and PLLMS[2:0] Pins**

| FASTBOOT | AEM    | PLLMS[2:0] | PLL <sub>M</sub>   |
|----------|--------|------------|--|
| 1        | 001    | N/A        | If ((BOOTMODE[3:0] == 0001) && (PCIEN==0)) Then PLL <sub>M</sub> = 26 (CLKIN × 27)<br>If ((BOOTMODE[3:0]==0011) && (PCIEN==0)) Then PLL <sub>M</sub> =14 (CLKIN × 15)<br>PLL <sub>M</sub> = 19 (CLKIN × 20) for all other values of BOOTMODE[3:0], PCIEN |
| 1        | != 001 | 000        | PLL <sub>M</sub> = 19 (CLKIN × 20) for all values of BOOTMODE[3:0], PCIEN  |
|          |        | 001        | PLL <sub>M</sub> = 14 (CLKIN × 15) for all values of BOOTMODE[3:0], PCIEN  |
|          |        | 010        | PLL <sub>M</sub> = 15 (CLKIN × 16) for all values of BOOTMODE[3:0], PCIEN  |
|          |        | 011        | PLL <sub>M</sub> = 17 (CLKIN × 18) for all values of BOOTMODE[3:0], PCIEN  |
|          |        | 100        | PLL <sub>M</sub> = 21 (CLKIN × 22) for all values of BOOTMODE[3:0], PCIEN  |
|          |        | 101        | PLL <sub>M</sub> = 24 (CLKIN × 25) for all values of BOOTMODE[3:0], PCIEN  |
|          |        | 110        | PLL <sub>M</sub> = 26 (CLKIN × 27) for all values of BOOTMODE[3:0], PCIEN  |
|          |        | 111        | PLL <sub>M</sub> = 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 6. PLL1 and PLL2 Multiplier Ranges**

| PLL Multiplier (PLLM) | MIN | MAX |
|-----------------------|-----|-----|
| PLL1 Multiplier       | x14 | x32 |
| PLL2 Multiplier       | x14 | x32 |

**Table 7. PLLC1 Clock Frequency Ranges**

| Clock Signal Name        | MIN                        | MAX               | UNIT |     |
|--------------------------|----------------------------|-------------------|------|-----|
| MXI/CLKIN                | 15                         | 30 <sup>(1)</sup> | MHz  |     |
| PLLOUT                   | At 1.2-V CV <sub>DD</sub>  | 300               | 600  | MHz |
|                          | At 1.05-V CV <sub>DD</sub> | 300               | 520  | MHz |
| SYSCLK1 (CLKDIV1 Domain) | -600 devices               |                   | 600  | MHz |
|                          | -500 devices               |                   | 500  | MHz |
|                          | -400 devices               |                   | 400  | MHz |
|                          | -300 devices               |                   | 300  | MHz |

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

**Table 8. PLLC2 Clock Frequency Ranges**

| Clock Signal Name          | MIN                        | MAX | UNIT |     |
|----------------------------|----------------------------|-----|------|-----|
| MXI/CLKIN <sup>(1)</sup>   | 15                         | 30  | MHz  |     |
| PLLOUT                     | At 1.2-V CV <sub>DD</sub>  | 300 | 900  | MHz |
|                            | At 1.05-V CV <sub>DD</sub> | 300 | 666  | MHz |
| PLL2_SYSCLK1 (to DDR2 PHY) |                            | 266 | MHz  |     |

<sup>(1)</sup> 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 9](#) lists the resulting CPU frequencies based on the PLLM values selected by FASTBOOT options.

**Table 9. CPU Frequency During FASTBOOT**

| PLLM | CPU Frequency (MHz) |
|------|---------------------|
| 19   | 270                 |
| 14   | 202.5               |
| 15   | 216                 |
| 17   | 243                 |
| 21   | 297                 |
| 24   | 337.5               |
| 26   | 364.5               |
| 29   | 405                 |

### 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 5](#).
2. Configures any HPI register that may be required.
3. Clears the DSP Boot Address Register (DSPBOOTADDR). Clears boot error code field (BOOTCMPLT.ERR) and boot complete bit (BOOTCMPLT.BC) in the Boot Complete Register (BOOTCMPLT).
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

C642x 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.

#### 2.5.1 PCI Boot With No Auto-Initialization

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 5](#). (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 FASTBOOT 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.

## 2.5.2 PCI With Auto-Initialization

When auto-initialization is used, the PCI configuration registers are programmed by the on-chip ROM Boot Loader (RBL) with the values stored in an I2C EEPROM connected to the I2C interface of the device.

PCI I2C EEPROM auto-initialization is enabled when `BOOTMODE[3:0] = 0010b`, `PCIEN = 1`, and `FASTBOOT = 1`. If auto-initialization is not enabled, the PCI configuration registers are left with their default values and the I2C EEPROM is not accessed for PCI configuration purposes. The function of the `BOOTMODE[3:0]`, `PCIEN`, and `FASTBOOT` pins is fully described in the device data manual, refer to that document for more details.

When auto-initialization is enabled, the `CONFIG_DONE` bit in the configuration done register (`PCICFGDONE`) takes a default value of 0. This prevents the PCI from responding to any requests. When auto-initialization is completed, the RBL sets the `CONFIG_DONE` bit to 1 to allow the PCI to respond to requests.

### 2.5.2.1 I2C EEPROM Memory Map

The on-chip ROM Bootloader requires big-endian format for the data stored in the I2C EEPROM. Byte addresses 400h through 41Bh of the I2C EEPROM are reserved for auto-initialization of PCI configuration registers. The remaining locations are not used for auto-initialization and can be used for storing other data. [Table 10](#) summarizes the I2C EEPROM memory layout, as required for PCI auto-initialization.

The data as stored in I2C EEPROM must begin at I2C EEPROM address `0x400` and is formatted as shown in [Table 10](#).

**Table 10. I2C EEPROM Layout for PCI Autoinitialization Data**

| 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        | Byte Checksum              |
| 0x41b        | Reserved (must use 0xAA)   |

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

### 2.5.2.2 I2C EEPROM Checksum for Autoinitialization Data

The PCI configuration data contained in the I2C EEPROM is checked against a checksum. The configuration data is treated as an array of bytes, starting from the lowest I2C address (400h) to the highest (419h), for a total of 26 bytes.

Byte Checksum = AAh XOR Byte0(400h) XOR Byte1(401h)... XOR Byte25(419h)

If the I2C EEPROM is not accessed for PCI configuration purposes (that is, PCI\_EEAI = 0 at reset), then the checksum is not performed. If the checksum fails, the on-chip ROM bootloader defaults to the UART boot and it does not set the CONFIG\_DONE bit in the configuration done register (PCICFGDONE).

### 2.5.2.3 DSP I2C EEPROM Interface

For PCI auto-initialization, the DSP supports I2C EEPROMs or devices operating as I2C slaves with the following features:

- The memory device complies with Philips I2C Bus Specification v 2.1
- The memory device uses two bytes for internal addressing; that is, the read/write bit followed by two bytes for addressing
- The memory device has the capability to autoincrement its internal address counter such that the contents of the memory device can be read sequentially

During PCI auto-initialization, 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. The slave address of the I2C EEPROM slave address must be set to 50h.

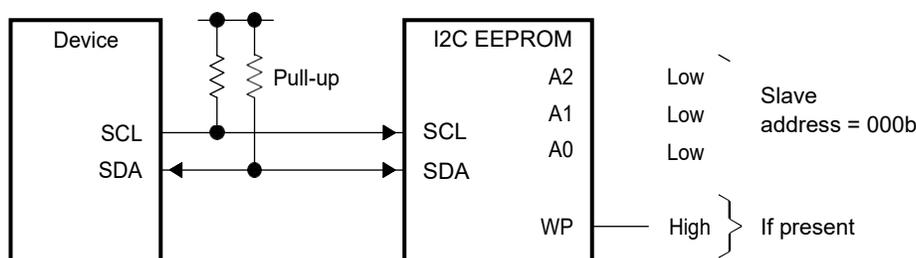


Figure 1. Signal Connections for I2C EEPROM Boot Mode

Some I2C EEPROMs have a write-protect (WP) feature that prevents unauthorized writes to memory. This feature is not needed for auto-initialization because the DSP will only read data from the I2C EEPROM. The write protect feature can be enabled or disabled.

For detailed information on the I2C, see the *TMS320C642x DSP Inter-Integrated Circuit (I2C) Module User's Guide* ([SPRUEN0](#)).

## 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 0x42000000.

## 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 0x42000000. 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 0x42000000. 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 5](#).
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 C642x 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 multiplier selected by values of AEM and PLLMS[2:0] as seen in [Table 5](#).
2. Configures I2C for master mode with slave address register set to 0x50, and own address register set to 0x29.
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 11. I2C Timing Register Settings**

| FASTBOOT | Register | Value |
|----------|----------|-------|
| 0        | IPSC     | 0x2   |
|          | ICCLKH   | 0x2D  |
|          | ICLKL    | 0x2D  |
| 1        | IPSC     | 0x2   |
|          | ICCLKH   | 0x8   |
|          | ICLKL    | 0x8   |

The frequency of the I2C master clock is derived by:

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

For C642x, 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 11](#) 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 5](#).
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 12](#) shows the derived master clock frequency based on the possible PLLM values.

**Table 12. 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        | <b>20.2</b>      |
| 29   | 810          | 405       | 67.5        | <b>22.5</b>      |

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

### 2.10.2 SPI 16x8 Master Boot Signal Polarity

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

**Table 13. SPI Master Boot Modes**

| Register | Value   |
|----------|---|
| PCR      | Field Values Set<br>FSXM = 1, FSRP = 1, CLKXM = 0, FSXP = 1 |
| SPCR     | Field Values Set<br>CLKSTP = 3, (11b)                       |
| RCR      | Field Values Set<br>RDATDLY = 1                             |
| XCR      | Field Value Set<br>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 and 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 2. 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 EEPROM, the EEPROM should be connected to MCBSP0 pins in accordance with [Table 14](#)

**Table 14. 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 is fully supported on Rev 1.30 of the device. For Rev 1.0 and Rev 1.20, this boot mode was supported through secondary boot only. On Rev 1.30 of the device, the bootloader supports boot using AIS image script stored in NAND starting at Block 1 or above. The bootlaoder searches for the AIS magic number in the first 32 bits of the of the page 0 of the block. If the magic number is not found, it continues searching through all of the NAND blocks until the last block is searched. If the magic number is not found it will abort the boot. The bootloader code skips bad blocks. A bad block is indicated when the bootloader detects the value 0xBADDBADD in starting in byte offset 0x4 in the NAND block spare memory area. When a bad block is detected, the bootloader skips this block and proceeds to the next block. The bootloader also performs 1 bit ECC correction when errors are detected in data read from the block. The bootloader uses the hardware ECC calculation provided by the DMP's EMIF module to detect and correct ECC errors.

On Rev 1.0 and Rev 1.20 of the device, there is a race condition in the polling for NAND ready in the ROM bootloader that prevents successful boot from NAND. 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 a secondary bootloader to 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 15](#). Please note that the secondary bootloader does not support any NAND devices that specifically require toggle of chip select signal for operation. The NAND device used for boot must be connected to EMIFA CS2.

**Table 15. Supported NAND Device Types**

| Device ID | Page Size | Total Memory Size |
|-----------|-----------|-------------------|
| 0xE3      | 512+16    | 4 MB              |
| 0xE5      | 512+16    | 4 MB              |
| 0xE6      | 512+16    | 8 MB              |
| 0x39      | 512+16    | 8 MB              |
| 0x6B      | 512+16    | 8 MB              |
| 0x73      | 512+16    | 16 MB             |
| 0x33      | 512+16    | 16 MB             |
| 0x75      | 512+16    | 32 MB             |
| 0x35      | 512+16    | 32 MB             |
| 0x43      | 512+16    | 16 MB             |
| 0x45      | 512+16    | 32 MB             |
| 0x53      | 512+16    | 16 MB             |
| 0x55      | 512+16    | 32 MB             |
| 0x76      | 512+16    | 64 MB             |
| 0x36      | 512+16    | 64 MB             |
| 0x79      | 512+16    | 128 MB            |
| 0x71      | 512+16    | 256 MB            |
| 0x46      | 512+16    | 64 MB             |
| 0x56      | 512+16    | 64 MB             |
| 0x74      | 512+16    | 128 MB            |
| 0xF1      | 2048+64   | 128 MB            |
| 0xA1      | 2048+64   | 128 MB            |
| 0xAA      | 2048+64   | 256 MB            |
| 0xDA      | 2048+64   | 256 MB            |
| 0xAC      | 2048+64   | 512 MB            |
| 0xDC      | 2048+64   | 512 MB            |
| 0xB1      | 2048+64   | 128 MB            |
| 0xC1      | 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 the start of AIS data, it assumes that the 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 the search of AIS data from block 1 of the memory and searches all of the 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 multiplier selected according to [Table 5](#).
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 16](#)

**Table 16. 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),<br>or none (BOOTMODE[3:0] == 1000b) |

## 2.13 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 address bits [23:16].
5. Transfer SPI address bits [5:8].
6. Transfer SPI address 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).

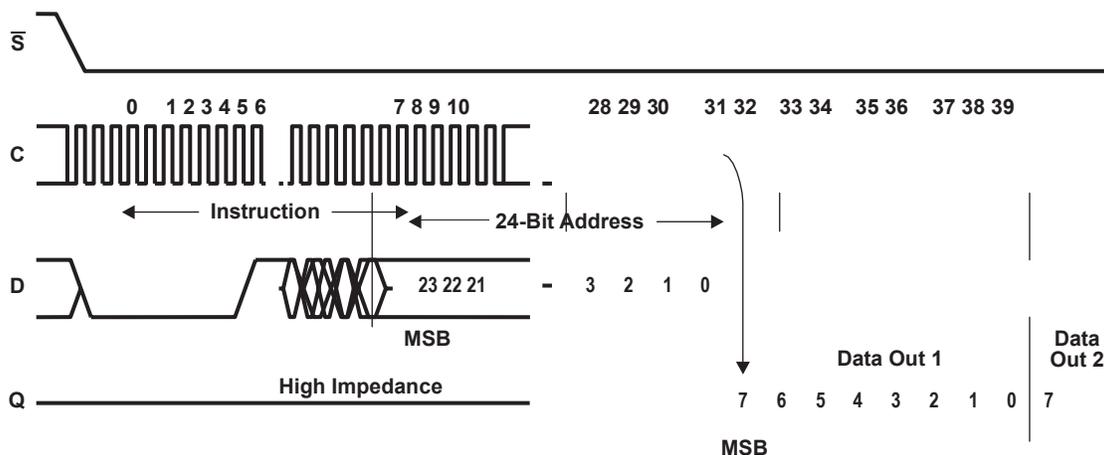
### 2.13.1 SPI 24x8 Master Boot Timings

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](#)

### 2.13.2 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 3](#) shows example 24x8 bit SPI EEPROM read timing.



**Figure 3. 24x8 Bit SPI EEPROM Read Timing**

Figure 4 shows how the DM643x 24x8 bit SPI boot works. Table 17 shows the pin connection between the DM643x and the 24x8 SPI EEPROM.

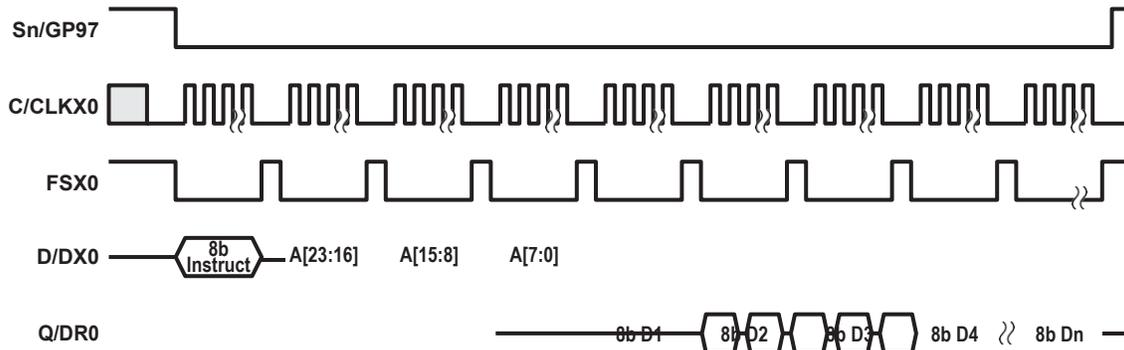


Figure 4. DM643x 24x8 Bit Address SPI Boot

### 2.13.3 Connecting SPI EEPROM for SPI 24x8 Boot

For 24x8 bit SPI mode, the McBSP0 pins must be connected to the SPI EEPROM according to Table 17

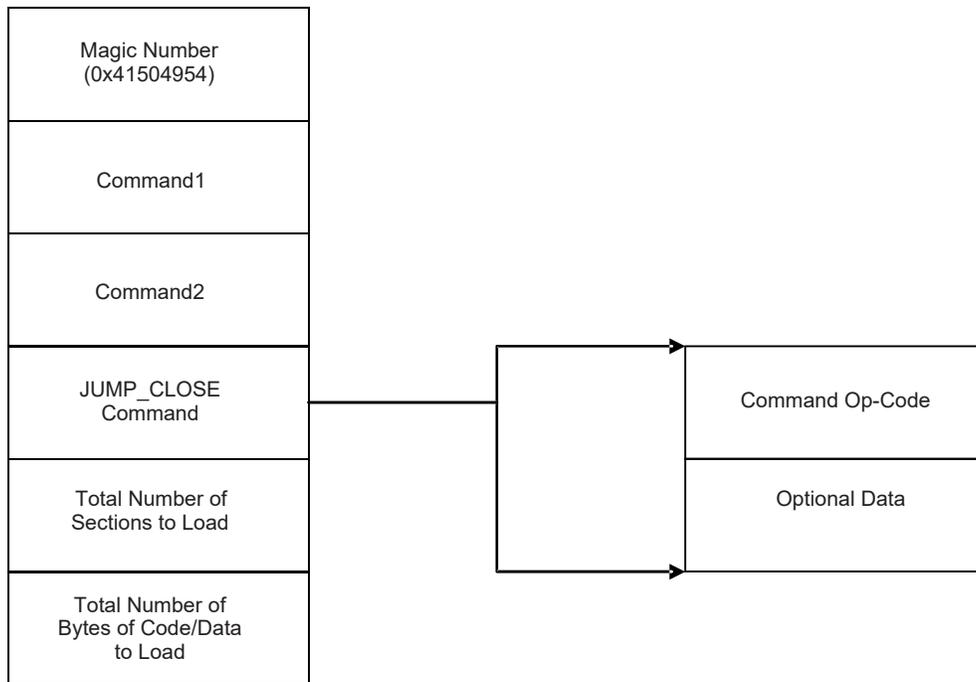
Table 17. SPI EEPROM and DSP Pin Connections for 24 Bit SPI Mode

| 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 unconnected, 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 (0x41504954); the header is then followed by a series of commands as shown in Figure 5. 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 5. Basic Structure of Application Image Script**

The bootloader only accepts data in AIS format for all modes except HPI and PCI. The following sections define each command with appropriate op-code, structure and placement in AIS. Table 18 lists the various opcodes that are supported by AIS 1.0.

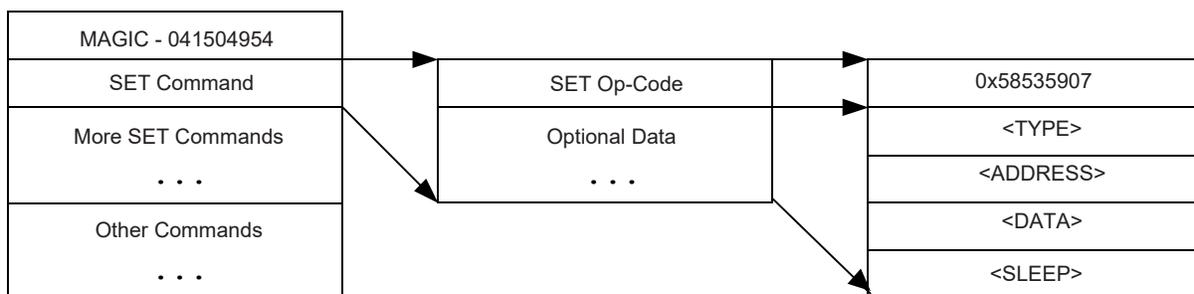
**Table 18. AIS Version 2.0 Supported Opcodes**

| Opcode           | Value      |
|------------------|------------|
| Section Load     | 0x58535901 |
| Request CRC      | 0x58535902 |
| Enable CRC       | 0x58535903 |
| Disable CRC      | 0x58535904 |
| Jump             | 0x58535905 |
| Jump_Close       | 0x58535906 |
| Set              | 0x58535907 |
| Start Over       | 0x58535908 |
| Reserved         | 0x58535909 |
| Section Fill     | 0x5853590A |
| Get              | 0x5853590C |
| 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 0xC, 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 6.



**Figure 6. Structure of SET Command**

Each set command consists of SET (0x58535907) 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 19.

**Table 19. 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]+ | 0x1234abCD  | 0x1000    | 0x5a      |
| Octal         | 0[0-7]+           | 02215125715 | 010000    | 0132      |

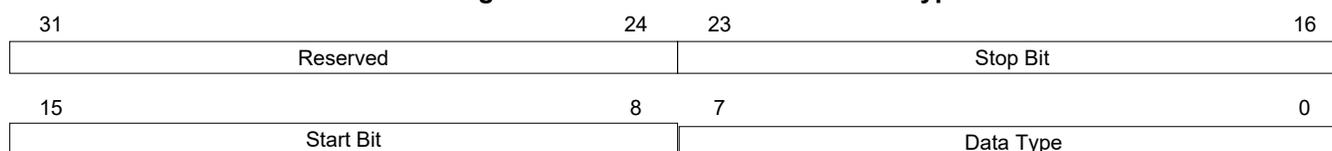
The data-type field <Type> determines the size of the data item such as 8-bit (B), 16-bit (S) or 32-bit (I). Data-type also can be a *field* or *bits*. This allows the setting of a particular range of bits within the data at the specified address. For *field* and *bits* data-types, the <Type> field also encodes the *start* and *stop* bit positions that define the field to be modified. Table 20 gives a full list of the data-types that can be used.

**Table 20. 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 18 shows the encoding of the 32 bit <Type>.

### 3.1.1 Valid SET Command Data Types

**Figure 7. Valid SET Command Data Types**


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

**Table 21. Valid SET Command Data Types Field Descriptions**

| Bit   | Field     | Value | Description   |
|-------|-----------|-------|---|
| 31-24 | Reserved  | 0     | Reserved  |
| 23-16 | Stop Bit  |       | Stop bit (for <i>bits</i> and <i>fields</i> data type) last bit position in word that delimits field  |
| 15-8  | Start Bit |       | Start bit (for <i>bits</i> and <i>fields</i> 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 8 shows the structure of the GET command.

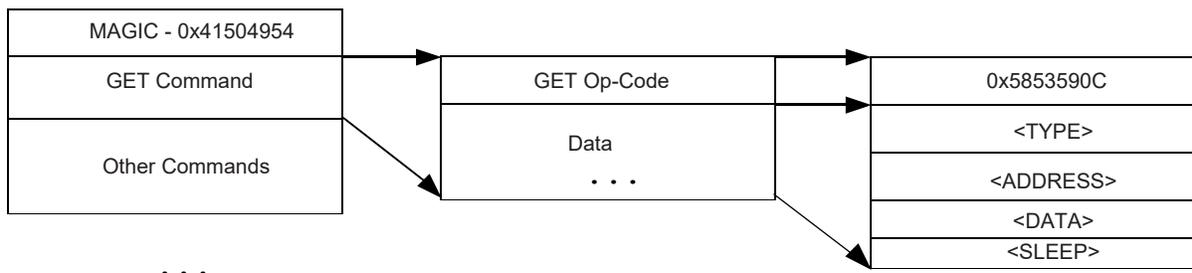


Figure 8. Structure of GET Command

### 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 9 shows the structure of the section load command.

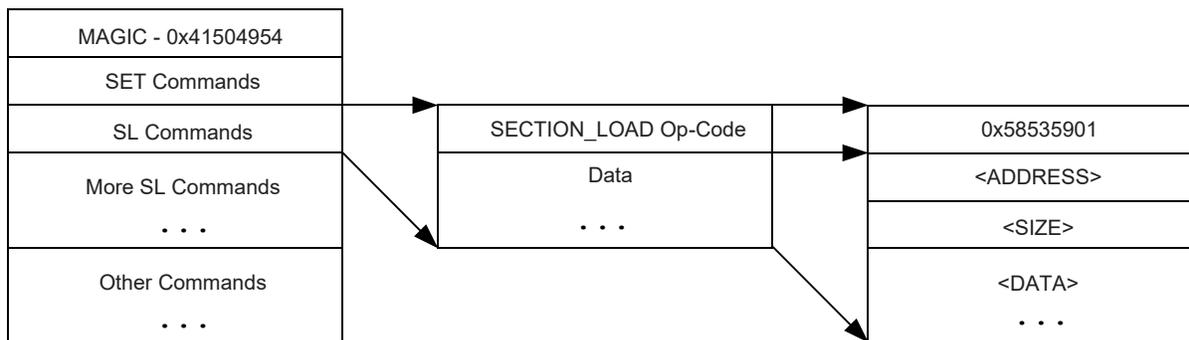
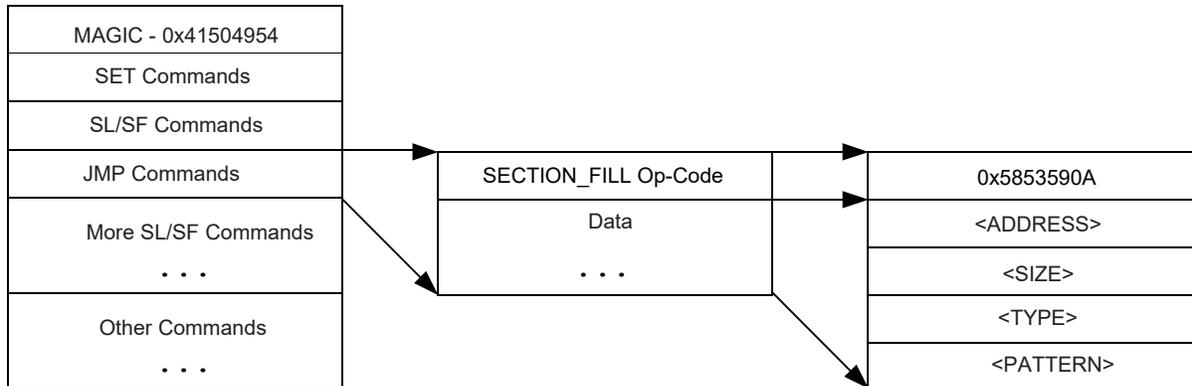


Figure 9. Structure of Section Load Command

Each section load command consists of SECTION\_LOAD (0x58535901) 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 10 shows the structure of the section fill command.

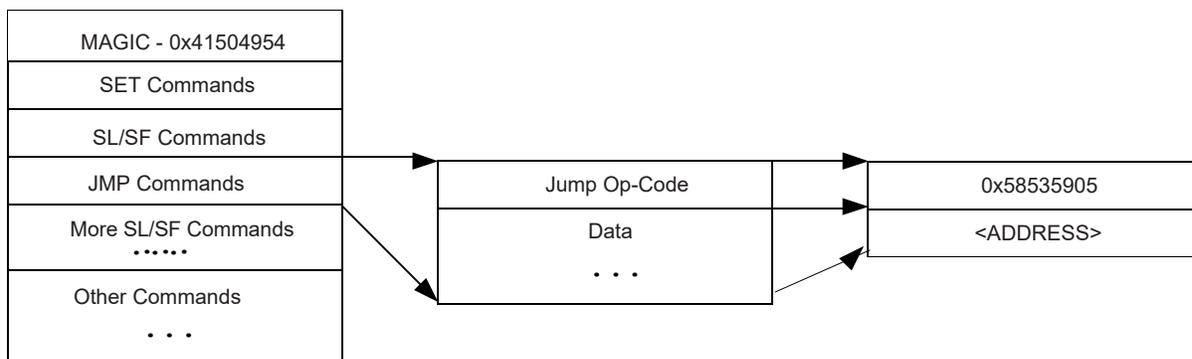


**Figure 10. Structure of Section Fill Command**

Each section fill command consists of SECTION\_FILL (0x5853590A) 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 (0x58535905) op-code, followed by the jump address. Figure 11 shows the structure of the jump command.



**Figure 11. 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 12 shows the structure of the Jump\_Close command.

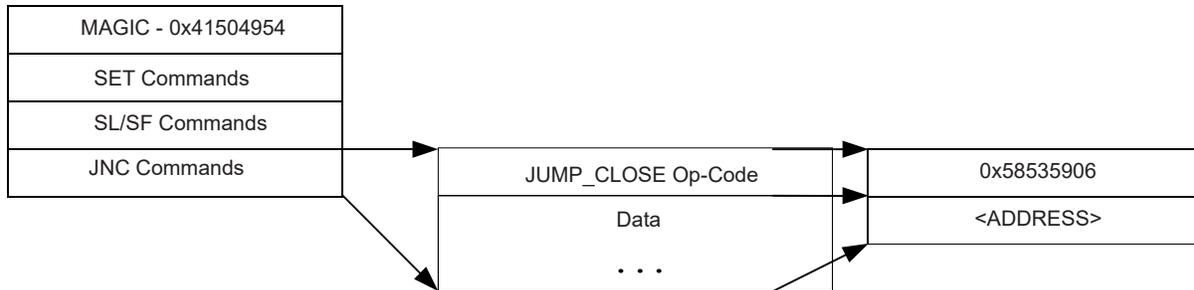


Figure 12. Structure of Jump\_Close Command

This command is placed at the end of AIS, after all other commands. It consists of JUMP\_CLOSE (0x58535906) 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 an 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:

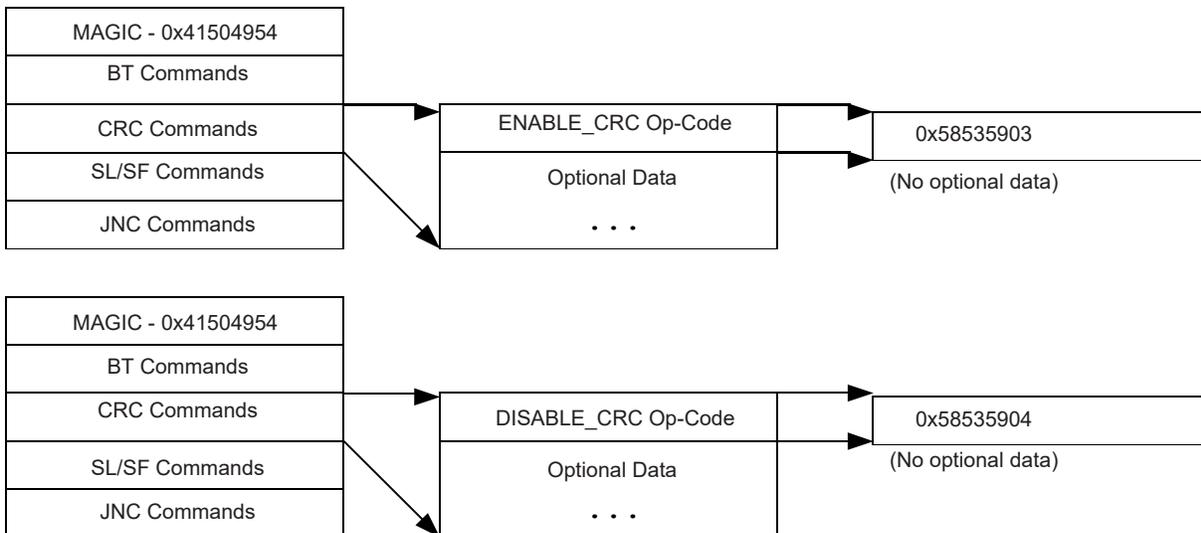
**No CRC**— CRC computation is disabled and there is no way to detect or correct any error.

**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 13 shows the structure of the enable CRC/disable CRC commands.



**Figure 13. Structure of Enable CRC/Disable CRC Commands**

These commands consist of only a single ENABLE\_CRC (0x58535903) or DISABLE\_CRC (0x58535904) op-code. There is no additional data required.

### 3.7.2 Request CRC Command

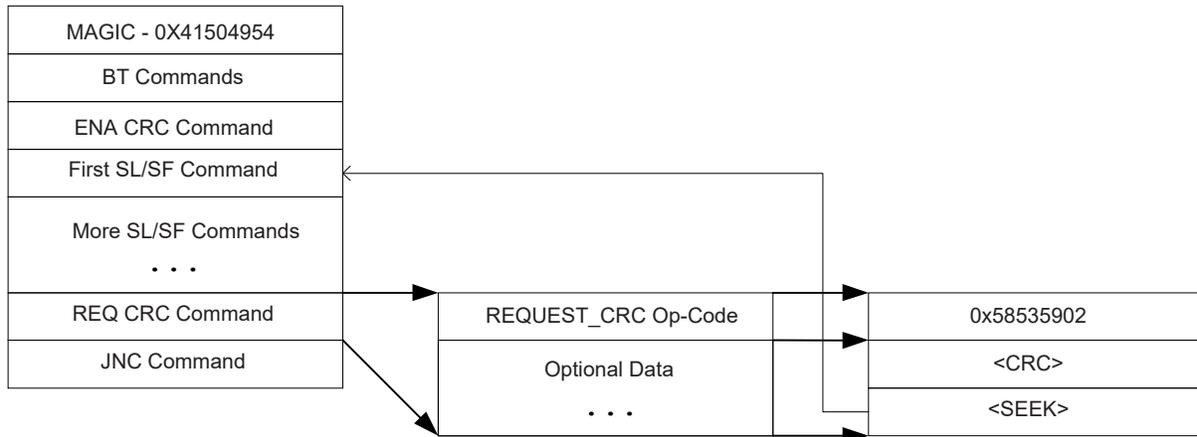
This command is used to request and validate the current value of the CRC computed by the DSP. Using this command requires that the enable CRC command be issued earlier in AIS. This command consists of the REQUEST\_CRC (0x58535902) op-code, followed by the expected CRC value and seek-value; the CRC of 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 14 shows the structure of the request CRC command.



**Figure 14. 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 14.

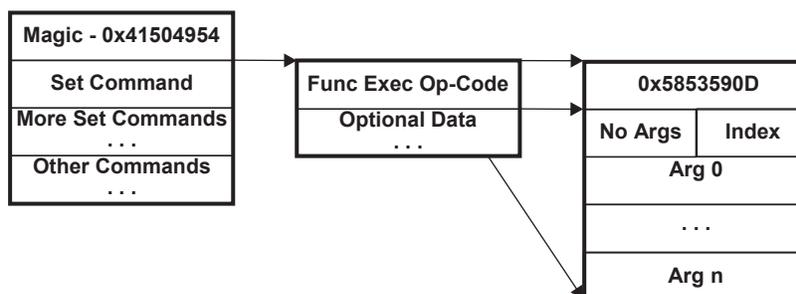
For section-wise CRC option, this command appears after each section load/section fill commands. The seek value is interpreted as a negative number, which when added to the current offset in AIS, makes offset point to the start of the previous section load/section fill command as shown in Figure 14.

### 3.7.3 Start-Over Command

The start-over command consists of a STARTOVER (0x58535908) 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 15 shows the structure of the function execute command.



**Figure 15. 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 C642x, 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 22 shows an example pre-defined ROM functions.

**Table 22. 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 23 shows an example function execute command for PLL config.

**Table 23. 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 five arguments for the EMIFA config function are 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 (sets PLLM field of PLLM register of PLL2)
2. DDR CLK divide down (sets RATIO field of PLLDIV1 register of PLL2)
3. Reserved - set to 0x00000000
4. DDR clock source (0-internal, 1-external) (sets CLKMODE field of PLLCTL register of PLL2)
5. DDR control register value (sets DDRPHYCR of DDR2 Memory Controller)
6. Synchronous dynamic random access memory (SDRAM) config register value (sets SDBCR register of DDR2 Memory Controller)
7. SDRAM timer 0 register value (sets SDTIMR register of DDR2 Memory Controller)
8. SDRAM timer 1 register value (sets SDTIMR2 register of DDR2 Memory Controller)
9. SDRAM refresh control register value (sets SDRCR register of DDR2 Memory Controller)

The DDR memory controller config function, programs the DRR PLL and then configures the DDR memory 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 can 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. Because of this, the bootloader 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, genAIS

The genAIS.pl script is a Perl script that converts a linked executable for the C642x to the appropriate format for the given boot mode and data or memory widths. The genAIS script is a command line tool and may be invoked as part of a larger script or Make file. The current version of genAIS was developed using Active Perl V5.8.6. A simple invocation of the genAIS script 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 and memory width of the device where the image is stored.

For example:

```
genAIS.pl -i MyApplication.out -o MyApplication.ais -bootmode spi -otype ascii -addrsz 16
```

This invocation would produce a converted ASCII AIS file formatted 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 genAIS tool is shown in [Table 24](#).

A simple invocation of the C642x 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:

```
genAIS.pl -i MyApplication.out -o MyApplication.ais -bootmode spi -otype ascii -addrsz 16
```

This invocation would produce a converted ASCII AIS file formatted 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 C642x tool is shown in [Table 24](#).

## 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 versions of the Windows®NT and Windows95 operating systems, may not support 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 24. C642x 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:<br>N = 0 - no CRC generation<br>N = 1 - CRC generated for each section load<br>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:<br>Please note that <i>raw</i> 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 C642x 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 address 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, C642x. The C642x is a Perl script that converts an application linked executable file to an AIS boot image file for the bootmode selected. C642x 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
||              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 0x00 → 8 bit, 0x01 → 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, 0x41504954. Any valid AIS command may appear after the magic word. [Table 19](#) shows the sample data stream for a 16-bit FLASH, using the sample source included at the start of this section

**Table 25. 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                                      |
| 0xFFFFFA8  | 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                                      |
| 0xFFFFFDC  | 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 25](#).

**Table 26. I2C AIS Boot Image Example**

| Data       | Explanation   |
|------------|---|
| 0x00000002 | Reserved for C642x – boot loader ignores              |
| 0x41504954 | AIS Magic Number                                      |
| 0x58535903 | Enable CRC Command                                    |
| 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                                    |
| 0xFFFFFA8  | 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                                    |
| 0xFFFFFDC  | 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 27 details the expected byte arrangement of the AIS boot image in the I2C EEPROM.

**Table 27. 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 28. 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 |   |
| 0x45B0C6E  |   |
| 0x2C6E00B4 |   |

**Table 28. SPI AIS Boot Image Example (continued)**

| Data       | Explanation   |
|------------|---|
| 0x8C6E008A |   |
| 0xEFC08000 | End Section Raw Data                                  |
| 0x58535902 | Request CRC Command                                   |
| 0x0E85A97B | Expected CRC Value                                    |
| 0xFFFFFA8  | 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                                    |
| 0xFFFFFDC  | 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 stored in the EEPROM should be as follows using the AIS data from [Table 27](#) as an example.

**Table 29. 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 byte 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 16](#)).

```

415049545853590358535901108000000000004001802028028024280200222801884069020
0032A020C027702884068028C1FDB020840686C6E10CD10442641003C2C6E45B06C6E2
C6E00B48C6E008AEFC08000585359020E85A97BFFFFFFFA858535901108000400000000
C0000000A0000000B0000000C585359028434A250FFFFFFDC5853590610800000000000
20000004C
    
```

**Figure 16. 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 C642x tool leaves space for these in [Table 30](#) generated as place holders for real values to be encoded later, when image is finally written to the NAND.

**Table 30. 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 where 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   |  |
| 0xEFC08000   | End of section raw data  |
| 0x58535902   | Request CRC command  |
| 0x0E85A97B   | Expected CRC value   |
| 0xFFFFFFFFA8 | Negative pointer to last valid command                           |
| 0x58535901   | Section load command   |
| 0x10800040   | Section load address   |

**Table 30. NAND Boot AIS Boot Image Example (continued)**

| Data       | Explanation   |
|------------|---|
| 0x0000000C | Section size in bytes                                 |
| 0x0000000A | Start section raw data                                |
| 0x0000000B |   |
| 0x0000000C | End section raw data                                  |
| 0x58535902 | Request CRC command                                   |
| 0x8434A250 | Expected CRC value                                    |
| 0xFFFFFDC  | 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 # DDR CLLK DIV
0x00000000 # Reserved
0x00000000 # PLL SRC
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 Debugging Boot Failures

When the device fails to boot as expected, some helpful information may be gleaned from the ERR field of the BOOTCMPLT register. If you can connect via JTAG to the chip and read this register, then [Table 31](#) may help you determine the cause of the failure.

**Table 31. Debugging Boot Failures**

| Value | Name                          | Description  |
|-------|-------------------------------|--|
| 0x1   | ERR_UNKNOWN_COMMAND           | An invalid AIS command was received  |
| 0x2   | ERR_BAD_MAGIC_NUMBER          | The AIS magic number was not detected in UART boot mode                          |
| 0x3   | ERR_TRANSMIT_SYNC             | Not used   |
| 0x4   | ERR_BAD_CRC                   | Getting PCI Autoinit data failed, or exceeded maximum CRC failure in AIS parsing |
| 0x5   | ERR_INVALID_ADDRESS_SIZE      | Invalid specified address width in first word of SPI master boot                 |
| 0x6   | ERR_UNSUPPORTED_BOOTMODE      | The various boot pin configurations are invalid                                  |
| 0x7   | ERR_TIMEOUT_WAITING_FOR_HOST  | Timeout occurred in sending or receiving data (UART, PCI, UHPI modes)            |
| 0x8   | ERR_TIMEOUT_I2C_BUS_BUSY      | Timeout waiting for bus becoming free in I2C master boot                         |
| 0x9   | ERR_TIMEOUT_MCBSP_SPI_RECEIVE | Receive timeout in SPI master boot mode  |
| 0xA   | ERR_NAND_ACCESS_TIMEOUT       | NAND device detection failure/timeout  |
| 0xB   | ERR_RECEPTION_ERR             | Various reception errors in UART, I2C master, and PCI modes                      |
| 0xC   | ERR_BAD_FUNCTION_PTR          | Undefined function pointer call in ROM boot loader                               |
| 0xD   | ERR_PLL_LOCKUP                | PLL configuration failed   |
| 0xE   | ERR_CFG_FUNCTION_CALL         | Invalid function index or argument count for AIS function execute command        |

## 10 Determining On-Chip Bootloader Version

The bootloader version can be found by reading the ROM location 0x00101A00. More than one ROM version is extant at this time. ROM version 0x27B2A120 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 0x00010200, 0x00010300, and 0x00010400 support all the features delineated within this document, including FASTBOOT.

## 11 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;
} SectionDataObj;
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);
}
```

## 12 References

- *TMS320C6000 Multi-Channel Buffered Serial Port User's Guide* ([SPRU580](#))
- *TMS320C6000 Code Generation Tools Installation Instructions* ([SPRU237](#))
- *TMS320C6000 Assembly Language Tools User's Guide*, [SPRU186](#))

## 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 & 0xFFFFF) ; 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 AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale ([www.ti.com/legal/termsofsale.html](http://www.ti.com/legal/termsofsale.html)) or other applicable terms available either on [ti.com](http://ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2019, Texas Instruments Incorporated