Application Report

SPRAAW2A–June 2009

Using the TMS320C6452 Bootloader

Karen Baldwin

ABSTRACT

This document describes the functionality of the TMS320C6452 ROM bootloader software. The ROM bootloader resides in the ROM of the device beginning at ROM address 0×00100000. The ROM boot loader (RBL) implements methods for booting in the listed modes. It reads the contents of the BOOTCFG register to determine boot mode and performs appropriate commands to effect boot of device. If an improper boot mode is chosen or if for some reason an error is detected during boot from a slave device, the RBL communicates this through the universal asynchronous receiver/transmitter (UART) as a default boot device. Note that the ROM bootloader requires use of an Application Image Script (AIS) as the primary data format for loading code/data. AIS is a Texas Instruments, Inc. proprietary data format. AIS is explained in detail in Section 2 of this document.

- Emulation Boot
- HPI
- PCI (DMP as slave)
- EMIFA CS2 ROM Direct Boot
- EMIFA CS2 ROM Fast Boot with AIS
- I2C (DMP as master) (supported with ROM version 3.70)
- SPI (DMP as master)
- UART (DMP as slave), no flow control (supported with ROM version 3.70)
- UART (DMP as slave), with flow control (supported with ROM version 3.70)
- Ethernet (supported with ROM version 3.70)

When booting in master mode, the boot loader reads boot information from the slave device as and when required. When booting in slave mode, the boot loader depends on the master device to feed boot information when required. For all boot modes, the ROM bootloader disables the watchdog timer for a duration of boot. All applications MUST avoid configuring watchdog timer during boot process. (No AIS commands or code should change this during boot).

Table 1. Terms and Abbreviations

Term	Description
Bootloader	SW/Code for ROM C6452 Bootloader
AIS	Application Image Script
BL	Boot Loader (referring to the bootloader in this text)
DMP	Digital Media Processor (DMP)
I2C	Inter Integrated Circuit
OS	Op-Code Synchronization
POS	Ping Op-Code Synchronization
ROM	Read Only Memory
SPI	Serial Peripheral Interface
SWS	Start-Word Synchronization
EMIF	External Memory Interface

Table 1. Terms and Abbreviations (continued)

Term	Description
PLL	Phased-Locked Loop
EEPROM	Electrically Erasable Programmable Read-Only Memory
MCBSP	Multichannel Buffered Serial Port
FIFO	First-In-First-Out
HPI	Host-Post Interface
PCI	Peripheral Component Interconnect
UDP	User Datagram Protocol

Contents

List of Figures

List of Tables

1 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 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 bootloader supports a FASTBOOT option that is determined by the status of the FASTBOOT pin sampled at a POR reset of the device. When FASTBOOT is enabled, the bootloader configures the PLL using a fixed PLL multiplier value. The bootloader on the C6452 sets this multiplier at 19 for all boot modes except Ethernet boot. This gives a PLL multiplier of x20 for the input clock. When any of the Ethernet boot modes is selected, the bootloader always configures PLL. When FASTBOOT option is asserted, the bootloader will program PLL using multiplier of x24, yielding CPU clock of x25 input clock. When FASTBOOT is not asserted for Ethernet boot mode, the bootloader uses default multiplier of 19.

Boot device pins must be configured to one of the valid modes. A description of each valid mode is given in subsequent sections. Ethernet boot, I2C boot, and I2C boot modes are supported with ROM version 3.70. All other modes are supported on all extant ROM versions.

All other modes ***not*** shown in Table 2 and are reserved and represent invalid settings.

Table 2. C6452 Boot Modes

DEVICE BOOT AND CONFIGURATION PINS			BOOT DESCRIPTION	genAIS DMP (Master/Slave)	DSPBOOTADDR (DEFAULT)
BOOTMODE[3:0]	UHPIEN	FASTBOOT			
0000	0 or 1	0 or 1	No Boot (Emulation Boot)	–	0x0080 0000
0001	0	1	PCI Boot No Auto-initialization	Slave	0x0080 0000
	1	0 or 1	HPI Boot	Slave	0x0080 0000
0010	0	1	PCI Boot With Auto-initialization	Slave	0x0080 0000
	1	0 or 1	HPI Boot	Slave	0x0080 0000
0011	0 or 1	0	UART Boot with No Hardware Flow Control	Slave	0x0080 0000
0100	0 or 1	0	EMIFA ROM Direct Boot(PLL Bypass Mode)	Master	0xA000 0000
		1	EMIFA ROM AIS Boot	Master	0x0080 0000
0101	0 or 1	0 or 1	I2C Boot [STANDARD MODE]	Master	0x0080 0000
0110	0 or 1	0 or 1	SPI Boot	Master	0x0080 0000
0111	0 or 1	0 or 1	Reserved	–	0x0080 0000
1000	0 or 1	0 or 1	SGMII0 - Boot Port, no packet forwarding	Slave	0x0080 0000
1001	0 or 1	0 or 1	SGMII0 - Boot Port, SGMII1 packet forwarding	Slave	0x0080 0000
1010	0 or 1	0 or 1	SGMII1 - Boot Port, SGMII0 packet forwarding	Slave	0x0080 0000
1011	0 or 1	0 or 1	Reserved	–	0x0080 0000
1100	0 or 1	0 or 1	Reserved	–	0x0080 0000
1101	0 or 1	0 or 1	Reserved	–	0x0080 0000
1110	0 or 1	0	UART Boot with Hardware Flow Control [UART0]	Slave	0x0080 0000
1111	0 or 1	0 or 1	Reserved	–	0x0080 0000

1.1 Boot Requirements, Constraints, and Default Settings

Please make note of the following requirements:

1. FASTBOOT is required for all PCI boot modes and is the default for all SGMII boot modes.
2. Bootloader supports only 16-bit address width for I2C EEPROMs.
3. For PCI boot with auto-initialization, auto-init data is expected to be stored on I2C EEPROM connected to I2C of the device.
4. All boot timings are optimized for a 27-MHz input clock frequency.
5. I2C, SPI, UART, and EMIFA CS2 FASTBOOT (BOOTMODE[3:0]=0100b) requires data for boot to be stored in Application Image Script Format. AIS is a Texas Instruments, Inc. proprietary format for boot images. A detailed description of AIS is given in Section 1.7 of this document. Any formats used for HOST modes such as HPI and PCI is solely at the discretion of the user.
6. When FASTBOOT is selected, the bootloader configures the PLL. The value of the PLL multiplier used has been fixed at 19, yielding CLKIN $\times$ 20 as value of SYSCLK. This document bases all timing calculations assuming a 27-MHz input clock to the device. For more detailed discussion, see the device-specific data sheet.
7. By default, cache is enabled when the DMP comes out of reset. When the ROM bootloader executes, it disables all L1 and L2 cache during boot. This affects ALL boot modes with exception of EMIFA Direct boot. In EMIFA Direct boot, the ROM bootloader code is not executed; therefore, cache is enabled and is operating according to default power on settings.
8. For UART boot modes, a 27-MHz input clock frequency is required. This is the only boot mode with this requirement.

1.2 FASTBOOT Mode

When FASTBOOT mode is selected, the ROMed bootloader programs the PLL using a fixed multiplier value. This affects all boot modes with exception of emulation boot. FASTBOOT is ignored in case of emulation boot. For all Ethernet/SGMII boot modes, the PLL is programmed regardless of state of FASTBOOT pin. When any Ethernet/SGMII mode is selected, the state of the FASTBOOT pin is used to determine the PLL multiplier value. When FASTBOOT = 1, and BOOTMODE[3:0] = 1000b, 1001b, 1011b, the PLL multiplier is set to 24, yielding CLKIN × 25 as the value of SYSCLK. When FASTBOOT = 0, and any of the Ethernet/SGMII boot modes is selected, the PLL multiplier is set to the normal FASTBOOT PLL multiplier of 19. For all other boot modes, the PLL multiplier used is fixed at 19 for FASTBOOT, yielding CLKIN × 20 as value of SYSCLK.

1.2.1 CPU Frequency With FASTBOOT Options

The bootloader software uses a fixed PLL multiplier of 19 when configuring the PLL for FASTBOOT option for all boot modes except emulation boot and SGMII boot. For SGMII boot, FASTBOOT PLL multiplier is fixed at 24. ALL timings in this document are based on a 27-MHz input clock for DMP/CPU and 62.5-Mhz input to Serdes for Ethernet Port configuration. To determine the CLKIN frequency best suited for your device and application, see the device-specific datasheet. The assumed input clock frequency and PLL multiplier used for FASTBOOT may not result in MAX CPU frequency rated for your device. The intent of FASTBOOT mode is to enable faster boot time by clocking CPU and boot peripherals and greater speeds. It is not intended to set the maximum CPU frequency at boot. The application may later change the PLL settings to enable maximum operating frequency after boot is complete.

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

In this boot mode, the ROM bootloader software executes a SW loop. The emulation software has responsibility for performing 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.

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

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

1. When FASTBOOT = 1, the bootloader programs the PLL based on the PLL multiplier value of 19***.***
2. Configures any HPI register that may be required.
3. Clears the DSPBOOTADDR register. Clears boot error code field (BOOTCMPLT.ERR) and boot complete bit (BOOTCMPLT.BC) in the Boot Complete Register (BOOTCMPLT).
4. Posts HINT to the HOST device, signaling that the DMP 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 the BOOTCMPLT register.
7. Once BOOTCMPLT.BC has been set by HOST, the ROM bootloader software branches to the address set by HOST in DSPBOOTADDR.

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

C6452 supports the PCI boot with DMP as PCI slave only. The bootloader implements PCI boot with and without auto-initialization. When 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 PCI boot. Please enable FASTBOOT with any PCI boot mode to ensure PCI timing meets requirements.

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

1. Configures PLL using a PLL multiplier of 19. (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 the 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. 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 and PCI boot are selected, the ROM bootloader software configures the PLL, as the first step, prior to clearing the DSPBOOTADDR and BOOTCMPLT registers.

When PCI boot with auto-initialization is selected, the bootloader reads the PCI configuration data stored in I2C EEPROM connected to I2C of the device. The data as stored in I2C EEPROM must be in formatted as shown in Table 3. The data must begin at EEPROM byte address 0×400.

Table 3. PCI Configuration Data for Auto-Init

Byte Address	Contents
0×400	Vendor ID [15:8]
0×401	Vendor ID [7:0]
0×402	Device ID [15:8]
0×403	Device ID [7:0]
0×404	Class code [7:0]
0×405	Revision ID [7:0]
0×406	Class code [23:16]
0×407	Class code [15:8]
0×408	Subsystem vendor ID [15:8]
0×409	Subsystem vendor ID [7:0]
0×40a	Subsystem ID [15:8]
0×40b	Subsystem ID [7:0]
0×40c	Max_Latency
0×40d	Min_Grant
0×40e	Reserved (use 0x00)
0×40f	Reserved (use 0x00)
0×410	Reserved (use 0x00)
0×411	Reserved (use 0x00)
0×412	Reserved (use 0x00)
0×413	Reserved (use 0x00)
0×414	Reserved (use 0x00)
0×415	Reserved (use 0x00)
0×416	Reserved (use 0x00)
0×417	Reserved (use 0x00)
0×418	Reserved (use 0x00)
0×419	Reserved (use 0x00)
0×41a	Checksum [15:8]
0×41b	Checksum [7:0]

Store the PCI initialization data in BIG-ENDIAN format.

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

EMIFA direct boot does not require intervention from the ROM bootloader software. The DMP hardware boot module causes direct branch to the start of EMIFA memory at address 0×A0000000. Code execution begins at that address.

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

During EMIFA FAST ROM boot mode, the DMP 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 PLL to operate at faster CPU speeds, and then reading code/data starting at EMIFA address 0×A0000000. The data stored in the FLASH/ROM must be in AIS format. A description of AIS is given in Section 2 of this document. The AIS boot image consists of AIS commands and data necessary to load the application code into the DMP memory. Using the AIS format eliminates the requirement of you having to define a secondary boot loader to load the application. 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 are processed for the boot. The ROM bootloader clears it's internal state and then branches to the start of application code. EMIF FASTBOOT bootloader sequence is shown below:

1. Programs PLL using a PLL multiplier of 19.
2. Reads value of 8_16 pin as latched into BOOTCFG register and sets EMIF data width accordingly.
3. Fetches AIS data from external memory and processes AIS commands until the JUMP CLOSE command is encountered.
4. Branches to the application start address given in the JUMP CLOSE command.

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

The C6452 supports I2C boot with DMP as I2C master only. The DMP 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. For more details regarding AIS, see Section 2. Boot sequence for I2C is as follows:

1. When FASTBOOT = 1, bootloader programs PLL using PLL multiplier value of 19 (generating SYSCLK of CLKIN × 20).
2. Configures I2C for master mode with slave address register set to 0×50, and own address register set to 0×29.
3. Processes each AIS command until JUMP_CLOSE command is encountered.
4. Branches to application start address
5. If an error occurs during the I2C boot process, the bootloader writes an error condition in the ERR field of the BOOTCMPLT register. Then, it attempts to perform boot through UART.

1.8.1 I2C Master Boot Timing

The bootloader sets the following values for I2C clock pre-scale and clock low-hold/clock high-hold registers depending on value of FASTBOOT configuration pin.

Table 4. I2C Timing Register Settings With CLKIN of 27 MHz

FASTBOOT	System Clock	I2C Peripheral Clock Frequency	Register	Value	I2C Master Clock Frequency
0	27 MHz	4.5 MHz	IPSC	0×1	140 KHz
			ICCLKH	0×2	
			ICLKL	0×2	
1	540 MHz	90 MHz	IPSC	0×B	187 KHz
			ICCLKH	0×F	
			ICCLKL	0×F	

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 C6452, the I2C peripheral clock is derived from the internal clock provided to the I2C. The peripheral clock is derived by a fixed divide of 1/6 SYSCLK. 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 bootloader software always programs a value of 11 for IPSC. Assuming an input oscillator frequency of 27 MHz, the settings for IPSC, ICCLKH, ICCLKL, in Table 4 show the I2C lock frequencies.

1.9 *SPI Master Mode Boot (BOOTMODE[3:0] = 0110b, FASTBOOT = 0 or 1)*

Because DMP as SPI master is the only mode supported by bootloader, the bootloader configures the SPI to operate as SPI master, and then initiates data read from the attached SPI EEPROM. The boot flow is as follows:

1. If FASTBOOT is enabled, the bootloader programs PLL using PLLM value of 19, then,
2. Bootloader reads first byte of data from EEPROM to retrieve address width expressed in number of bytes.
3. Bootloader reads AIS formatted boot image from EEPROM.
4. When the last AIS command is encountered (JUMP CLOSE command), the bootloader branches to the application entry address given in the command.

1.9.1 SPI Master Boot Timing

The SPI master clock frequency is derived from the internal clock provided to the SPI. The peripheral clock is derived by a fixed divide of 1/6 the CPU clock. The SPI master clock frequency is further determined by the prescale value. When FASTBOOT is enabled, the bootloader uses a fixed prescale value of 7. When FASTBOOT is disabled, the clock prescale value is set to 0. Table 5 shows the derived master clock frequency based on FASTBOOT configuration.

Table 5. SPI Master Clock Frequencies

FASTBOOT	PRESCALE	SYSCLK	SPI Peripheral Clock (MHz)	SPI Master (MHz)
0	0	27 MHz	4.5 MHz	4.5 MHz
1	7	540 MHz	90 MHz	11.25 MHz

1.9.2 SPI Master Boot Signal Descriptions

The SPI module is configured for 4-pin SPI master boot with the following selections:

Table 6. SPI Master Boot Modes

Register	Value
SPIFMT0	Field Values Set POLARITY = 1, PHASE = 0, PRESCALE = 0 or 7, CHARLEN = 8
SPIPC0	Field Values Set DIFUN = 1, DOFUN = 1, CLKFUN = 1
SPIGCR1	Field Values Set CLKMOD = 1, MASTER = 1

With these modes selected, the SPI master clock polarity is inactive high. Output data is driven on the falling edge of the SPICLK, and input data is sampled at the SPIDI pin on the rising edge of the SPICLK. Though the SCSFUNx field(s) of the SPIPC0 register is not set, the SPICS1 pin is controlled manually by the bootloader, driven low to access the external SPI device. Furthermore, the SPI_UART_EN field of the PINMUX register is set such that all muxed pins are SPI pins.

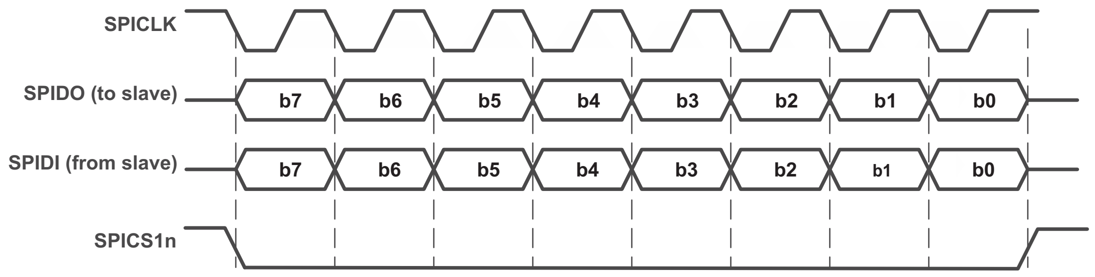

Figure 1. SPI Transfer With Polarity = 1, Phase = 0

1.10 UART Boot (BOOTMODE[3:0] = 0011b, 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 of 19.
2. Configures UART registers as required by chosen mode.
3. Sends message *BOOTME* through the serial interface to the HOST.
4. Waits response from the HOST in the form of the AIS magic number. The bootloader continually loops, 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 bootloader sends message, *DONE*, to the HOST and then branches to the application start address.

The AIS commands are expected to be in ASCII representation, hence, to send the AIS magic word , 0×41504954, the character sequence *41*, *50*, *49*, *54*, is expected to be received by the bootloader. A sample AIS stream for UART boot is given in Section 2.

1.10.1 UART Boot Timing

Operationally, UART boot via BOOTMODES[3:0] == 1000b and 1110b are essentially the same. The difference is management of data flow. When BOOTMODE[3:0] = 1000b, UART boot is executed without the use of the hardware flow control. If UART BOOTMODE[3:0] = 1110b is selected, then the UART is configured to use the hardware flow control module. UART FIFO is enabled in both modes 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 7.

Table 7. UART Connection Attributes for Boot

Attribute	Value
Baud Rate	11Kbps
Data Bits	8
Stop Bits	1
Parity	None
Flow Control	Hardware Flow Control (BOOTMODE[3:0} == 1110b), or None (BOOTMODE[3:0] == 0011b)

1.11 Ethernet Boot Modes - SGMII0 Only, SGMII0, and SGMII1 (Bootmode[3:0] = 1000b, 1001b, 1010b)

In Ethernet Boot Mode, the ROMed bootloader configures the communication processor Gigabit Ethernet Switch (CPSW_3G) to enable boot over Ethernet. The CPSW_3G is a 3-port Gigabit Ethernet Switch. Two of the ports, Port 0 and Port 1, can be configured for boot (Port 2 is assigned as port to DMP). Which port acts as the boot port for the device is dependent on the boot mode chosen.

When SGMII0 boot mode (BOOTMODE[3:0] = 1001b) is configured, Port 0 of the 3-port Switch is configured as the boot port. Port 1 of the switch is set in packet-forwarding mode. This configuration allows multiple DMPs to be daisy-chained on an application board with packets properly forwarded to each device. In SGMII1 boot configuration (BOOTMODE[3:0] = 1010b), Port 1 is configured as the boot port, and Port 0 is set in packet forwarding mode. Otherwise, these two boot modes (SGMII0/SGMII1) operate in the same way. The boot flow for Ethernet boot is as follows:

- Configure the Serializer/Deserializer (SerDes) module
- Configure CPSW_3G with the following settings
  - Flow control for Port 0,1, and 2 are enabled
  - Read MAC address as latched into DMP MAC address registers (read from EFUSE at device $\overline{\text{POR}}$ reset)
  - Configure CPGMAC_SL0 (port 0), CPGMAC_SL1(port 1) source address registers
  - Enable CPDMA transmit and receive
  - Enable ALE (address lookup)
  - Clear and initialize ALE lookup table inserting single entry of device MAC address as latched into MAC address register at $\overline{\text{POR}}$ reset
  - Configure all ports (0,1,2) in packet forwarding mode
  - Configure Port 0 and Port 1 in GMII enabled
  - If SGMII0 boot is selected, then configure Port 0 as slave port, and Port 1 as master with auto-negotiation enabled
  - If SGMII1 boot is selected, then configure Port 1 as slave port, and Port 0 as master with auto-negotiation enabled

- An approximate 25-ms wait is inserted after CPSW_3G is fully configured to ensure time for auto-negotiation to complete. (This time may vary based on processor speed. The 25-ms delay is calculated based on a CPU speed of 500 MHz.)
- Once the switch has been configured, the bootloader initializes the transmit and receive descriptor buffers and sends a BOOTP request packet (Ethernet ready announcement frame), through the boot port.

Note: The BOOTP request packet does not comply with the BOOTP Specification and is a TI variant. This would normally be discarded or rejected by the standard server.

- The BOOTP request packet is sent only once. The bootloader then waits in a loop, pending receipt of the BOOT table packets from the Host/Server.
- Once all BOOT packets have been received, the bootloader branches to the start of the downloaded application code.

Figure 2 illustrates packet flow in a system using the current configuration. The illustration shows the SGMII0 boot mode in which Port 0 is assumed to be the boot port, and Port 1 is configured for packet forwarding. For the SGMII1 boot mode, the roles of the ports are reversed. BOOT table frames are received in Port 1 and packets not intended for current device are forwarded via Port 0.

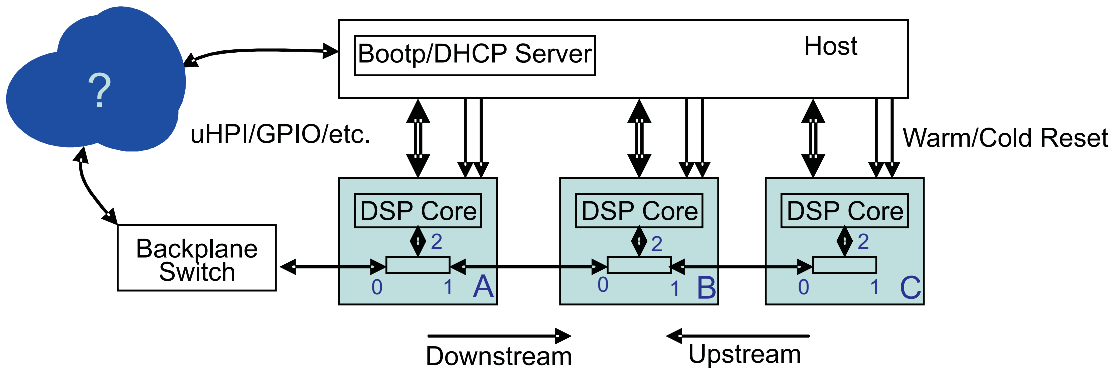

Figure 2. Packet Flow for SGMII0 Boot

Table 8. Ethernet 3 Port Switch Settings

Sub Module	Register	Value	Description
SerDes	CFGPLL	0×00000013	(Default)Pll enabled, MPY = x20 (assumes 62.5Mhz input)
	CFGTX0/1	0×00000B21	TX enabled, 10-bit bus width, SWING = 5
	CFGRX0/1	0×00089121	RX enabled, half rate, 10-bit bus width, signal loss detect enabled
CPSW	CONTROL	0×000000E0	Port 0/1 TX flow control, Port 2 RX flow control
	SL0_SA_LO, SL0_SA_HI, SL1_SA_LO_SL1_SA_HI	Device MAC address	MAC address as latched at $\overline{\text{POR}}$ from EFUSE

1.11.1 Ethernet Boot-Mode Timings

The SerDes PLL module assumes the default settings; therefore, at device $\overline{\text{POR}}$ reset, the PLL is enabled with default multiplier of x20. The bootloader further configures the SerDes module with TX/RX at quarter rate, (1 data sample every four PLL output cycles), yielding an effective line rate of 500-625 Mbps.

1.11.2 Ethernet Data Formats

Once the bootloader configures the 3-Port switch, it sends a BOOTP request packet via the boot port. The format for the BOOTP request packet is shown in Table 9. The HOST/Server responds by sending the application code using the Boot Table Format. That application's Boot Table is sent as Boot Table Frames from the HOST/Server to the DMP. This may require multiple Boot Table Frames to be transferred. The format that each Boot Table Frame must follow is described in Table 10.

Table 9. BOOTP Request Packet Format

Content	Byte Offset	Field Description	Field Values	Notes
DIX Ethernet Header	5-0	DST MAC Address	0×FF, 0×FF, 0×FF, 0×FF, 0×FF, 0×FF	
	11-6	SRC MAC Address	MSB-DMP MAC Address - LSB DMP MAC Address	DMP's MAC address
	13-12	Type	0×08, 0×00	
IPV4 Header	14	Version/Length	0×45	
	15	Type Of Service (TOS)	0×00	
	17-16	Total Length	0×01, 0×48	
	19-18	ID	0×00, 0×01	
	21-20	Flags/Fragment Offset	0×0000	
	22	Time to Live	0×10	
	23	Protocol	0×11	UDP Protocol
	25-24	Header Checksum	0×A9, 0×A5	
	29-26	SRC IP Address	0×000000	
	33-30	DST IP Address	0×000000	
UDP Header	35-34	SRC Port	68 (0×44)	
	37-36	DST Port	9(0x09)	ROMed boot code expects packet type of 9 or 'discard'
	39-38	Length	308(0×134)	
	41-40	Checksum	0×00, 0×00	
BOOTP Payload	42	Opcode	0×01	Request
	43	HW Type	0×01	Ethernet
	44	HW Address Length	0×06	
	45	Hop Count	0×00	
	49-46	Transaction ID	0×12345678	
	51-50	Number Seconds	0×0001	
	53-52	Flags	0×0000	
	57-54	Client IP	0×00000000	Client IP, DMP IP, Server IP & Gateway IP are all zeroes
	61-58	DMP IP	0×00000000	
	65-62	Server IP	0×00000000	
	69-66	Gateway IP	0×00000000	
	85-70	DMP HW (MAC) Address	MSB-DMP MAC Address - LSB DMP MAC Address	DMP's MAC Address
	149-86	Server Host Name	ti-boot-table-svr	
	278-150	Boot File Name	ti-boot-table-0000	
	341-279	Vendor Specific	Device ID	DMP Device ID

Table 10. Ethernet Boot Table Frame Format

Ethernet Header (one of the following types):	• DIX Ethernet (DMAC, SMAC, type: 14 bytes) • 802.3 w/SNAP/LLC (DMAC, SMAC len LLC, SNAP: 22 bytes) • DIX Ethernet w/VLAN (18 bytes) • 802.3 w/VLAN and SNAP LLC (26 bytes)
IVP4 Header	IVP4 (20 to 84 bytes)
UDP Header	UDP (8 bytes)
Boot Table Frame Header	Boot Table Frame Header (4 bytes)
Boot Table Frame Payload	(min 4 bytes, max limited by max Ethernet frame – previous headers)

Table 11. Boot Table Frame Header

Byte Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0	Magic number (0x544B)															
0x2	Opcode (0x01)								Sequence Number							

Each entry in the Boot Table Frame Header is 16 bits. The first entry is the Magic number. The second entry consists of two 8-bit fields. The 8-bit opcode field should always be set to 0×01. The 8-bit sequence number indicates the order of the packet in the stream. Sequence numbering should always begin with 0 as the number for the first packet in the boot stream.

The payload for the boot packets is a Boot Table data stream. The Boot Table must be formatted as shown in Table 12. The Boot Table data is then split across as many packets/frames as required to transmit the entire Boot Table for the application. Each Boot Table entry is 32 bits wide.

The Boot Table format is encapsulated in Ethernet frames with IPV4 and UDP headers. If the bootloader encounters frames that do not conform to the criteria specified below, they are silently discarded; subsequent frames will be processed. The acceptable Ethernet frames and processing is detailed below:

- Frames using both DIX and 802.3 MAC header formats are accepted as are frames with and without VLAN tags. Any source MAC address is accepted.
- Initially the bootloader sets the acceptable destination MAC address (for receiving packets) to be the MAC address read from EFUSE of the device. This is the destination address that is used until boot is complete. VLAN fields (other than type/len) are ignored.
  – The EFUSE MAC address is also used as the source MAC address in the BOOTP packet.
- If 802.3 format MAC format is used, the SNAP/LLC header is verified and skipped. The type field selects IPV4 type (0×0800).
- The IPV4 header validates the Version (4), flag and fragment fields, and protocol (UDP) field. The header length field is parsed in order to properly skip the header option words. Any source and destination IP addresses are accepted.
- The UDP header validates that the source and destination port numbers match those specified in the boot parameters (in the Ethernet boot code).
  – By default, the boot parameter source port field is 0, and in this case any source port is accepted.
  – The default value of the destination UDP port is 9 (discard port)
- The UDP header length is sanity-tested against the appropriately adjusted frame length. If the UDP length is too long for the frame, or is not a multiple of two, the frame is discarded.
- The UDP checksum is verified, and the frame with incorrect UDP checksum is discarded if the UDP checksum field is non-zero.

Table 12. Boot Table Format

Byte Offset	Entry
0x0000 0000	Entry Point Byte Address (starting address to branch to after code has been loaded)
0x0000 0004	Section 1 Size (in bytes)
0x0000 0008	Section 1 Load Address (in bytes)

Table 12. Boot Table Format (continued)

Byte Offset	Entry
0x0000 000C	Section 1 Data
....	....
...	....
...	Section 2 Size (in bytes)
...	Section 2 Load Address (in bytes)
...	Section 2 Data
...	...
...	Section N Size (in bytes)
...	Section N Load Address (in bytes)
...	Section N Data
...	...

2 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 bootloader. 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 header consists of a magic word (0×41504954). The header is then followed by a series of commands as shown in Figure 3. Each command in turn 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.

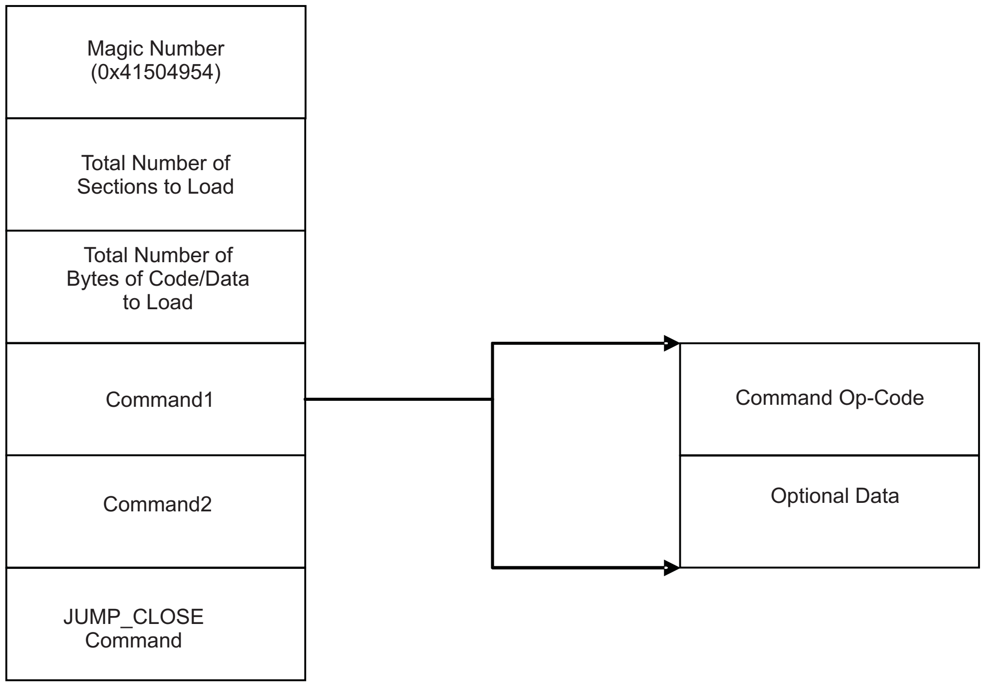

Figure 3. Basic Structure of Application Image Script

The bootloader only accepts data in AIS format for all modes except HPI ad PCI. The following sections define each command with the appropriate op-code, structure and placement in AIS. Table 13 lists the various opcodes that are supported by AIS 1.0:

Table 13. AIS Version 2.0 Supported Opcodes

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

2.1 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 DMP address space. One of the arguments to this command implements a delay after the memory write happens; use this for memory-mapped register write to take effect. Set commands are used to configure various peripherals of the DMP. This includes PLL and EMIF at minimum, and can configure more peripherals if required.

When DMP comes up from reset, the PLL is in bypass mode. As a result, the CPU is clocked at the same frequency as connected crystal/CLK IN, which is generally very low. This results in slow communication and high boot time. Selecting FASTBOOT mitigates this by programming the PLL with a slightly higher multiplier of 0×C, but this does not change the default EMIF wait states, etc. 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, place all SET commands for configuring EMIF and setting PLL at the beginning of the AIS boot image as shown in Figure 4.

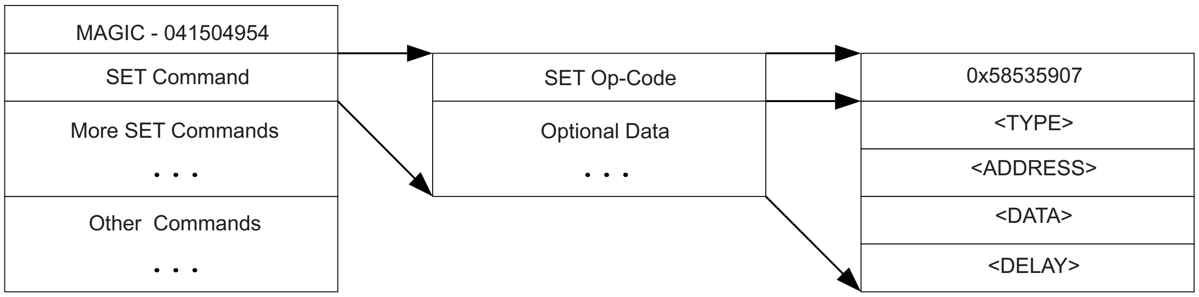

Figure 4. Structure of SET Command

Each set command consists of the SET (0×58535907) op-code, followed by four words of additional data as shown. The 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 DMP 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 14.

Table 14. Numeric Formats That Can Be Used in SET Command

Name	Format	Example 1	Example 2	Example 3
Hexadecimal 1	0[xX][0-9a-fA-F]+	0×1234abCD	0×1000	0×5a
Hexadecimal 2	[0-9a-fA-F]+[hH]	1234ABCDh	1000H	5ah
Octal	0[0-7]+	02215125715	010000	0132
Decimal	[0-9]+	305441741	4096	90

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 can also be a *field* or *bits*. This allows 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 15 gives a full list of the data-types that may be used.

Table 15. Valid SET Command Data Types

Data Type	Value
8 bit	0
16 bits	1
32 bits	2
Field (1-32 bits)	3
Bits (1-32 bits)	4

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 in Table 15), *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 13 shows the encoding of the 32 bit <Type>.

2.1.1 Valid SET Command Data Types

Figure 5. Valid SET Command Data Types

31 ... 24	23 ... 16
Reserved	Stop Bit

15 ... 8	7 ... 0
Start Bit	Data Type

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

Table 16. Valid SET Command Data Types Field Descriptions

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

2.2 Get Command

The GET command enables fetch of value stored at any read accessible DMP memory address. The GET command has the same format as the SET command described in Section 2.1, 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 (i.e., MSBs are zero for all data that is less than 32 bits in length). The structure of the GET command is shown in Figure 6.

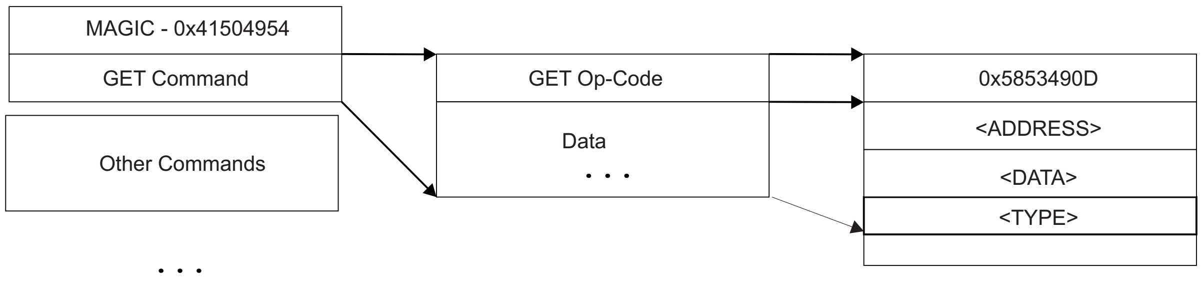

Figure 6. Structure of GET Command

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

```
<Address> = <Data><Type>
```

2.3 Section Load Command

The Section Load command is used to load a chunk of code/data to DMP memory. All initialized sections of the application are loaded to DMP memory using Section Load commands. These commands are placed after all SET commands in AIS. The structure of the Section Load command is shown in Figure 7.

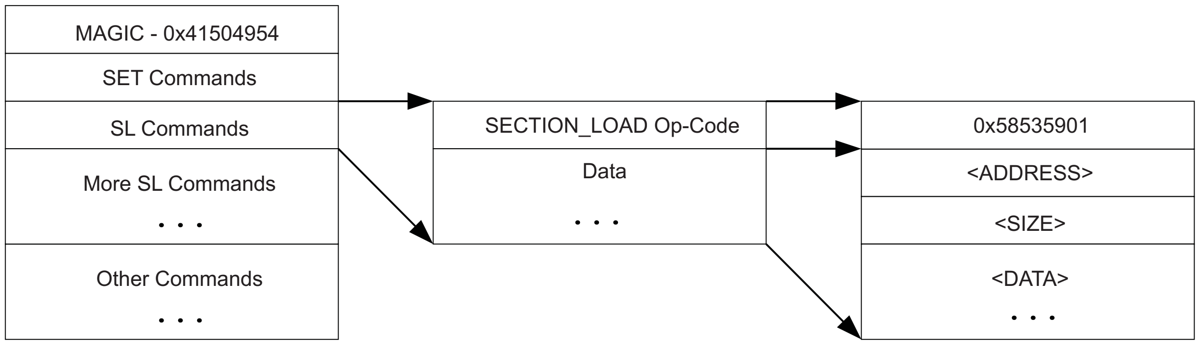

Figure 7. Structure of Section Load Command

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

2.4 Section Fill Command

Use the Section Fill command to fill a particular section with a specific pattern. For example, a section that contains all zeros is initialized with the Section Fill command. Place these commands anywhere that a regular Section Load command can occur. The structure of the Section Fill command is shown in Figure 8.

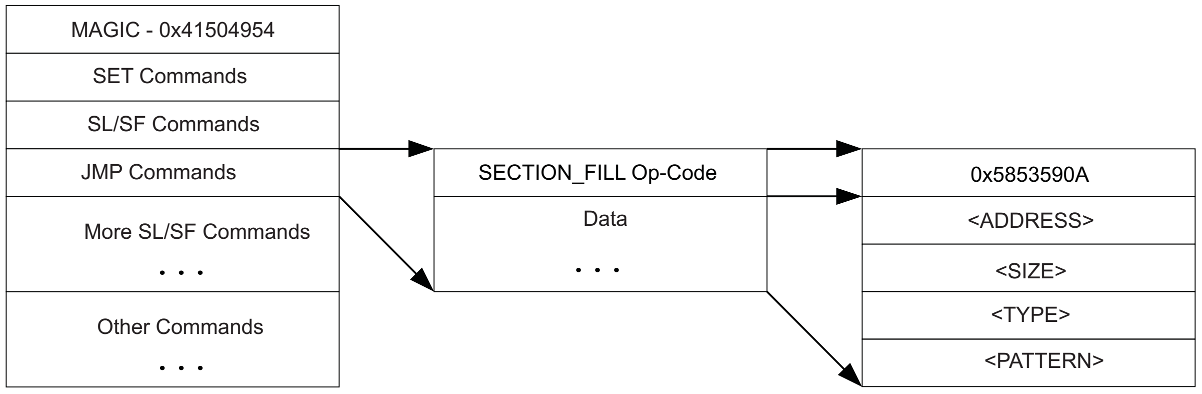

Figure 8. Structure of Section Fill Command

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

2.5 Jump Command

This command instructs the DMP to jump to the start address of the earlier loaded application. It consists of the JUMP (0×58535905) op-code, followed by the jump address as shown in Figure 9.

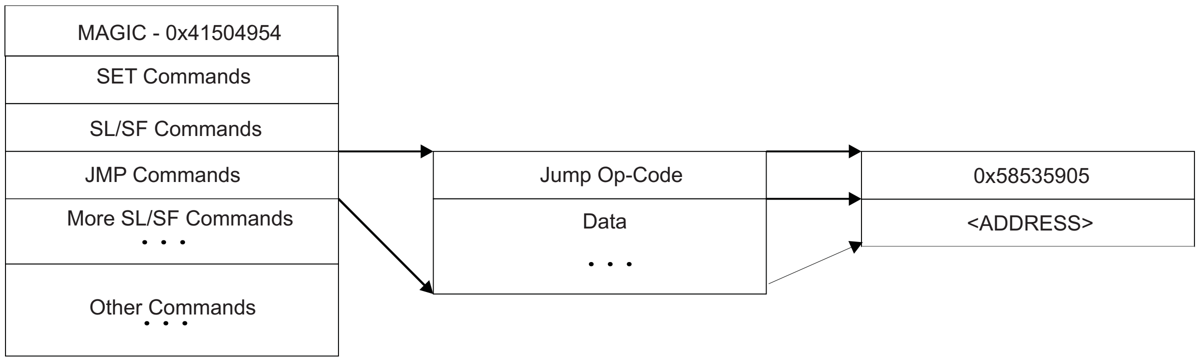

Figure 9. Structure of Jump Command

This command is used to implement bootloader2. To achieve this, bootloader2 is loaded through 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.

2.6 Jump_Close Command

This command is used at the end of the boot process to start the execution of the loaded application. it instructs the DMP to terminate the boot process and jump to the start address of the loaded application. The structure of the Jump_Close command is shown in Figure 10.

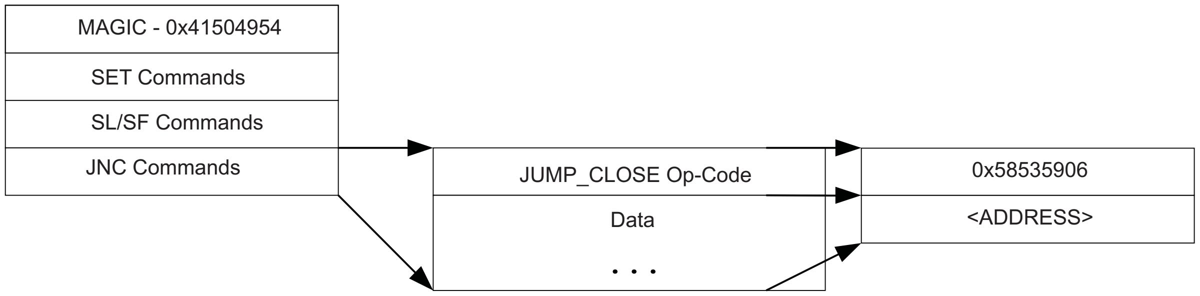

Figure 10. Structure of Jump_Close Command

This command is placed at the end of AIS, after all other commands. It consists of the JUMP_CLOSE (0×58535906) op-code, followed by the start address of the application where the boot loader should jump. In addition to the application entry point address, two words, 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.

2.7 CRC Options

There is a possibility of error in communication when DMP 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—With this option, CRC computation is disabled and there is no way to detect or correct any error.

Section-Wise CRC—With this option, 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.

Single CRC—With this option, 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.

2.7.1 Enable/Disable CRC Commands

These commands are used to enable/disable computation of CRC for sections loaded through Section Load/Section Fill commands. The structure of the Enable CRC/Disable CRC commands is shown in Figure 11.

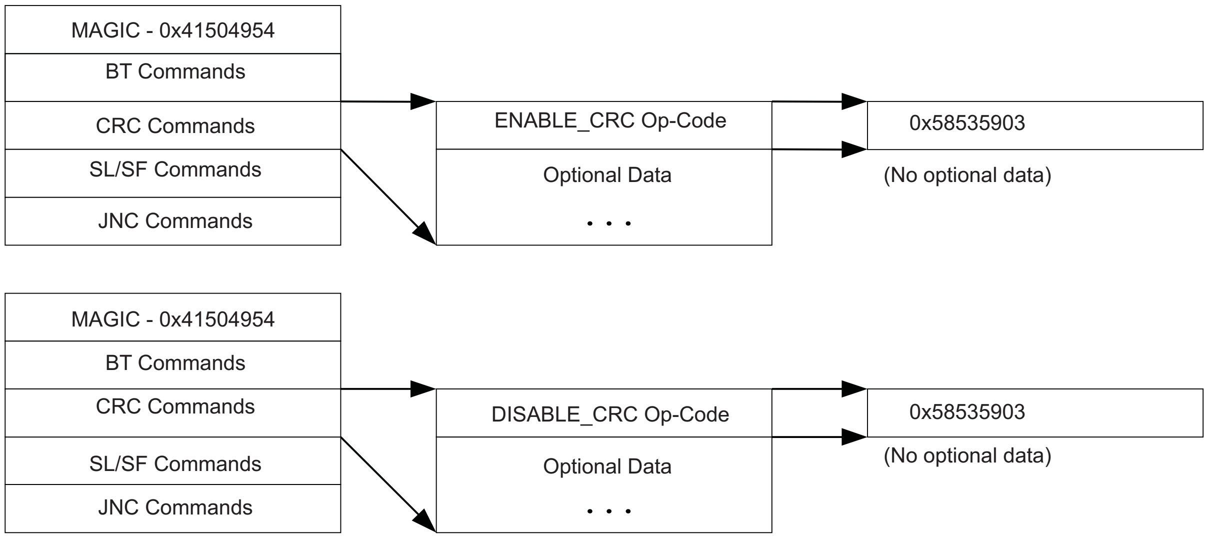

Figure 11. Structure of Enable CRC/Disable CRC Commands

These commands consist of only a single ENABLE_CRC (0×58535903) or DISABLE_CRC (0x58535904) op-code. There is no additional data required.

2.7.2 Request CRC Command

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

A mismatch in CRC indicates that the data loaded to DMP memory using the earlier Section Load/Section Fill commands is corrupted. To load the data again, re-execute AIS from the last known error-free point. To locate that point, a seek-value is made available as part of the Request CRC command. This value is interpreted as a negative number and should be added to the current address in AIS. When this is done, the address points to the last error-free point in AIS. Continue execution as normal from this updated address.

In case of CRC error, the host indicates the same to the DMP using the Start-Over command described below. After doing that, it adds the seek-value to the AIS address pointer and starts executing AIS from that point onwards.

On receiving the Start-Over command, the DMP knows that the CRC error has occurred. It resets its CRC computation and becomes ready to accept more commands from the host. The structure of the Request CRC command is shown in Figure 12.

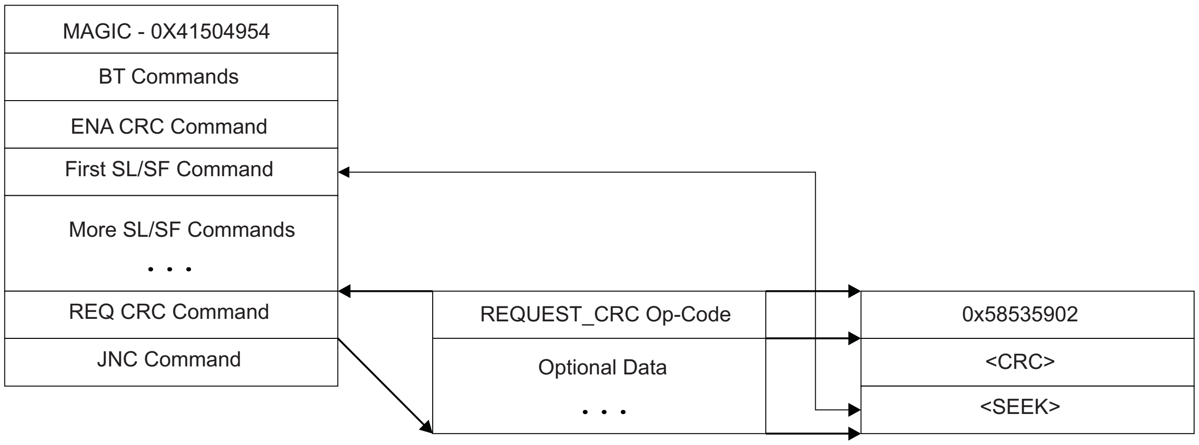

Figure 12. Structure of Request CRC Command

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 current offset in AIS, makes offset point to start of the first Section Load/Section Fill command as shown.

For Section-wise CRC option, this command appears after each Section Load/Section Fill commands. The seek value is interpreted as a negative number; when added to the current offset in AIS it makes an offset point to the start of the previous Section Load/Section Fill command as shown in Figure 12.

2.7.3 Start-Over Command

The Start-Over command consists of a STARTOVER (0×58535908) op-code with no additional data. This command instructs the bootloader to reset its computed CRC value to 0. Please note that this command (and op-code) is normally issued by the 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 Booting Operating Systems (Linux®/DSP/BIOS, ™ etc.)

The ROM bootloader operates independently 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 boot modes 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 that for this scenario, only the secondary bootloader must follow the appropriate ROM bootloader protocol for the boot mode chosen. The rest of the operating system code/data may be in any format required for the secondary boot to complete the 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 DMP memory.

4 ROM Bootloader RAM Memory Requirements and Code/Data Placement

The ROM bootloader requires a small amount of RAM in the internal memory space of the device to use for stack, and temporary buffer/data storage space. Memory is allocated in address range 0x00B10000 – 0x00B1F200 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 the 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.

5 AIS Generation Tool , genAIS

genAIS is a Perl script that converts an application linked executable file to an AIS boot image file for the selected boot mode. For example, genAIS converted a linked executable for the DM647 to the appropriate format for the given boot mode and data/memory widths.

genAIS 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 genAIS 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 will eventually be stored.

For example:

```
genAIS –I MyApplication.out –o MyApplication.ais –bootmode spi –otype ascii –addrsz 16
```

This invocation would produce a converted ASCII AIS file formatted for SPI boot. The AIS generation tool can produce either an ASCII, binary, 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 17.

Table 17. genAIS Program Options

Option	Description
-I	Specifies input executable file
-o	Specifies name of AIS output file
-opath	Specifies path where output files should be placed, default is to use the same path as input file
-crc N (N = 0,1,2)	Selects CRC generation: N = 0 - No CRC generation N = 1 - CRC generated for each section load N = 2 - Single CRC generated for entire load
-bootmode N (N=i2c, spi, uart, nand, enet, raw)	Specifies boot mode for which conversion is to be generated: • *Raw* generates an AIS image that is mode independent. • Boot modes supported by this utility may not be supported on your device. For more information regarding which modes are supported, see the device-specific data sheet and or bootloader documentation.
-otype N (N=ascii, binary, asm, cfile, txt)	Specifies content format for AIS output ascii - Generates ASCII text file output binary - Generates Binary file output asm - Generates C6x assembly output cfile - This mode is reserved for ENET boot mode, and generates C header file and single .cpp file with packet contents txt - Generates text output for use with UART boot mode
-memwidth N (N=8,16,24)	Specifies memory/address width for external memories associated with I2C and SPI boot modes.
-datawidth N (N=8,16)	Specifies NOR flash data access width for EMIF FASTBOOT option. Note: 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.
-srcipaddr	Source IP address given in form 000.000.000.000, all numbers must be in decimal format.
-dstipaddr	Destination IP address given in form 000.000.000.000, all numbers must be in decimal format.
-srcmacaddr	Source MAC address given in form 000.000.000.000.000.000, all numbers must be in decimal format
-dstmacaddr	Destination MAC address given in form 000.000.000.000.000.000, all numbers must be in decimal format.
-htype N (N=dix, snap, vlan, vsnap)	Ethernet Header Type
-packetsize	Size in bytes of Ethernet Boot Packet payload (not including header) This value must be evenly divisible by 4, and has a maximum value of 1400.

6 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 were created using genAIS. genAIS is discussed in Section 5. 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
```

6.1 AIS Boot Image for EMIFA ROM Boot

The first 8-bit byte in the FLASH/ROM accessed via EMIFA must contain the EEPROM size. Valid values are 0×00 → 8 bit, 0×01 → 16 bit. The next 3 bytes are reserved. The first valid AIS word begins on the next 32-bit word boundary. This word must contain the AIS magic word, 0×41504954. Any valid AIS command can appear after the magic word. Table 14 shows the sample data stream for a 16-bit FLASH, using the sample source included at the beginning of this section.

Table 18. EMIFA ROM Fast Boot AIS Boot Image Example

Data	Explanation
0x00000001	First byte of word specifies external memory data width
0×41504954	AIS Magic Number
0×58535903	Enable CRC Command
0×58535901	Section Load Command
0×10800000	Section Load Address
0×00000040	Section Size in Bytes
0×01802028	Start of Raw Section Data
0×02802428	
0×02002228	
0×01884069	
0×0200032A	
0×020C0277	
0×02884068	
0×028C1FDB	
0×02084068	
0×6C6E10CD	
0×10442641	
0×003C2C6E	
0×45B06C6E	
0×2C6E00B4	
0×8C6E008A	
0×EFC08000	End of Raw Section Data
0×58535902	Request CRC Command
0×0E85A97B	Expected CRC Value
0×FFFFFFA8	Negative Pointer to Last Valid Command in Stream
0×58535901	Section Load Command
0×10800040	Section Load Address
0×0000000C	Section Size in Bytes
0×0000000A	Start of Section Raw Data
0×0000000B	
0×0000000C	End of Section Raw Data
0×58535902	Request CRC Command
0×8434A250	Expected CRC Value
0×FFFFFFDC	Negative Pointer to Last Valid Command in Stream
0×58535906	Jump Close Command
0×10800000	Application Entry Point Address
0×00000002	Total number of sections that should have been loaded
0×0000004C	Total number of bytes that should have been loaded

6.2 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 18.

Table 19. I2C AIS Boot Image Example

Data	Explanation
0×00000002	
0×41504954	AIS Magic Number
0×58535903	AIS Magic Number
0×58535901	Section Load Command
0×10800000	Section Load Address
0×00000040	Section Size in Bytes
0×01802028	Start Section Raw Data
0×02802428	
0×02002228	
0×01884069	
0×02884068	
0×028C1FDB	
0×02084068	
0×6C6E10CD	
0×10442641	
0×003C2C6E	
0×45B06C6E	
0×2C6E00B4	
0×8C6E008A	
0×EFC08000	End Section Raw Data
0×58535902	Request CRC Command
0×0E85A97B	Expected CRC Value
0×FFFFFFA8	Negative Pointer to Last Valid Command
0×58535901	Section Load Command
0×10800040	Section Load Address
0×0000000C	Section Size in Bytes
0×0000000A	Start of Section RAW Data
0×0000000B	
0×0000000C	End Section Raw Data
0×58535902	Request CRC Value
0×8434A250	Expected CRC Value
0×FFFFFFDC	Negative Pointer to Last Valid Command
0×58535906	Jump Close Command
0×10800000	Application Entry Point Address
0×00000002	Total number of sections that should have been loaded
0×0000004C	Total number of bytes that should have been loaded

Table 20 details the expected byte arrangement of the AIS boot image in I2C EEPROM.

Table 20. AIS Image in I2C EEPROM Memory

Byte Address	Byte0	Byte1	Byte2	Byte3	32-Bit AIS Data	Explanation
0×0000	0×02	0×00	0×00	0×00	0×00000002	First byte contains address size in bytes – this is ignored by the bootloader for this device
0×0004	0×54	0×49	0×50	0×41	0×41504954	AIS Magic Word
0×0008	0×03	0×59	0×53	0×58	0×58535903	Enable CRC Command
0×000C	0×01	0×59	0×53	0×58	0×58535901	Section Load Command
0×0010	0×00	0×00	0×80	0×10	0×10800000	Section Load Address
0×0014	0×40	0×00	0×00	0×00	0×00000040	Section Size in Bytes
0×001C	0×28	0×20	0×80	0×01	0×01802028	Start Section Raw Data
0×0020	0×28	0×24	0×80	0×02	0×02802428	
0×0024	0×28	0×22	0×00	0×02	0×02002228	
0×0028	0×69	0×40	0×88	0×01	0×01884069	
0×008C					0×58535906	JUMP CLOSE Command
0×0090					0×10800000	Application Entry Point Address
0×0094					0×00000002	Total Number of Sections
0×0098					0×0000004C	Total Number of Bytes

6.3 AIS Boot Image for SPI Boot

The AIS boot image for SPI is the same as for other master boot modes, with exception that the first 32-bit word in AIS boot image must contain the address width of the SPI EEPROM expressed as bytes. The byte containing the address width must be located at address “0” of the EEPROM. Only values of 0x02 (16-bit SPI EEPROM) or 0x03 (24-bit SPI EEPROM/Flash) are acceptable.

Table 21. SPI AIS Boot Image Example

Data	Explanation
0×00000002	EEPROM Address Width in Bytes
0×41504954	AIS Magic Number
0×58535903	Enable CRC Command
0×58535901	Section Load Command
0×10800000	Section Load Address
0×00000040	Section Size in Bytes
0×01802028	Start Section Raw Data
0×02802428	
0×02002228	
0×01884069	
0×0200032A	
0×020C0277	
0×02884068	
0×028C1FDB	
0×02084068	
0×6C6E10CD	
0×10442641	
0×003C2C6E	
0×45B06C6E	

Table 21. SPI AIS Boot Image Example (continued)

Data	Explanation
0×2C6E00B4	
0×8C6E008A	
0×EFC08000	End Section Raw Data
0×58535902	Request CRC Command
0×0E85A97B	Expected CRC Value
0×FFFFFFA8	Negative Pointer to Last Valid Command
0×58535901	Section Load Command *myData section*
0×10800040	Section Load Address
0×0000000C	Section Size in Bytes
0×0000000A	Start Section Raw Data
0×0000000B	
0×0000000C	End Section Raw Data
0×58535902	Request CRC Command
0×8434A250	Expected CRC Value
0×FFFFFFDC	Negative Pointer to Last Valid Command
0×58535906	Jump Close Command
0×10800000	Application Entry Point Address
0×00000002	Total number of sections that should have been loaded
0×0000004C	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 20 as an example.

Table 22. AIS Image in SPI EEPROM Memory

Byte Address	Byte0	Byte1	Byte2	Byte3	32-Bit AIS Data	Explanation
0×0000	0×02	0×00	0×00	0×00	0×00000002	First byte contains address size in bytes
0×0004	0×54	0×49	0×50	0×41	0×41504954	AIS Magic Word
0×0008	0×03	0×59	0×53	0×58	0×58535903	Enable CRC Command
0×000C	0×01	0×59	0×53	0×58	0×58535901	Section Load Command
0×0010	0×00	0×00	0×80	0×10	0×10800000	Section Load Address
0×0014	0×40	0×00	0×00	0×00	0×00000040	Section Size in Bytes
0×001C	0×28	0×20	0×80	0×01	0×01802028	Start Section Raw Data
0×0020	0×28	0×24	0×80	0×02	0×02802428	
0×0024	0×28	0×22	0×00	0×02	0×02002228	
0×0028	0×69	0×40	0×88	0×01	0×01884069	
0×008C					0×58535906	JUMP CLOSE Command
0×0090					0×10800000	Application Entry Point Address
0×0094					0×00000002	Total Number of Sections
0×0098					0×0000004C	Total Number of Bytes

6.4 AIS Boot Image for UART Boot

UART boot mode differs from the previous modes in that some communication is carried out between the DMP and HOST, in addition to the transfer of the AIS commands. The DMP UART acts as slave in the boot process. But, to alert the HOST that the DMP is alive and ready to receive, it sends an initial BOOT ME message 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 the AIS command transmitted by the HOST until the JUMP CLOSE command is encountered. After the JUMP CLOSE command is received, the bootloader sends the *DONE* message to the HOST. This signals the HOST that boot has completed successfully.

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

At this point the boot process is complete and the bootloader branches to the application start address. If an error occurs, for example a CRC error, the bootloader issues a *CORRUPT* message 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.

```
415049545853590358535901108000000000004001802028028024280200222801884069020
0032A020C027702884068028C1FDB020840686C6E10CD10442641003C2C6E45B06C6E2
C6E00B48C6E008AEFC08000585359020E85A97BFFFFFFA858535901108000400000000
C0000000A0000000B0000000C585359028434A250FFFFFFDC585359061080000000000000
20000004C
```

Figure 13. UART AIS Boot Image

6.5 *Boot Packet Generation for Ethernet Boot Mode*

Several options are required to generate boot packets formatted for Ethernet boot mode. That list is shown in Table 23. The genAIS tool creates a separate C source file containing the data for each packet generated. It also creates a C header file that contains the external declarations and size information for each packet. These files can then be included in the build for the HOST/Server application, which sends the boot packets to the C6452 for boot.

Table 23. Ethernet Boot Packet Generation Options

Option	Description
-srcipaddr	Source IP address given in form 000.000.000.000 , all values must be in decimal format
-srcmacaddr	Source MAC address given in form 000.000.000.000.000.000.000, all values must be in decimal format
-dstipaddr	Destination IP address given in form 000.000.000.000, all values must be in decimal format
-dstmacaddr	Destination MAC address in form 000.000.000.000.000.000, all values must be in decimal format
-htype	Ethernet Header Type (dix, vlan, snap, vsnap)
-packetsize	Size in byes of payload for packets, this does not include header and max size allowed is 1400 bytes

A sample invocation of the ais tool would be:

genAIS -otype cfile -l my.out -bootmode enet -packetsize 1400 -srcipaddr 11.22.33.44 -dstipaddr 55.66.77.88 -srcmacaddr 123.156.158.218.23.12 -dstmacaddr 123.156.158.218.23.14 -opath myOutputPath

Example 2. Sample C Header File Associated With Packet

```
//==============================================================
// This is an auto-generated file:
//   Creation Time Stamp:
//     Nov, 13, 2007, 13:47:10
//   Created By User: a0321848
//   Created from .out File : docExample.out
//==============================================================
#ifdef __PACKETHDR__
            #define __PACKETHDR__
            #include <tistdtypes.h>
            #ifndef NULL
            #define NULL (0L)
            #endif


//==============================================================
// Define Total Number of BOOTP Packets
//==============================================================
            #define TOTAL_NUM_BOOTP_PACKETS      1


//==============================================================
// External Packet Array Declarations
//==============================================================
            #define SIZE_PACKET0      144
            extern Uint32 packet0[];



//==============================================================
// Initialize Packet Size Array
//==============================================================
            Uint32 packetSize[TOTAL_NUM_BOOTP_PACKETS] = {
                                         144
            };
#endif
```

The utility creates as output a the C header file that contains the external declarations and size for all data packets created by the utility. The data for each packet is stored in two dimensional array that is defined in a single C file with extension ".cpp". The first packet is packet 0 and contains the branch address for the application code once it is loaded to the DMP. If the *-opath* option is used in the invocation of genAIS, then the header and packet file are placed in the specified directory. Otherwise, these are placed in the same directory as the input file.

Alternatively, if -otype txt is specified, the AIS generation tool with create a single ASCII ".txt" file that contains a linear list of all packet data. It will still produce a C header file that describes the length and number of packets generated. This information can then be to parse the text file into the appropriate packets for sending to DMP.

Example 3. Sample Packet Created by AIS Utility

```
Uint32 packet[][] = {
                0xDA9E9C7B ,
                0x9C7B0E17 ,
                0x0C17DA9E ,
                0x00450008 ,
                0x00008400 ,
                0x11100000 ,
                0x160B7206 ,
                0x42372C21 ,
                0xA8AB584D ,
                0x70000000 ,
                0x544B7C34 ,
                0x40000100 ,
                0x004010F0 ,
                0x40000000 ,
                0x202810F0 ,
                0x242801A0 ,
                0x222802A0 ,
                0x78690220 ,
                0x032A0188 ,
                0x02770200 ,
                0x7868020C ,
                0x1FDB0288 ,
                0x7868028C ,
                0x10CD0208 ,
                0x26416C6E ,
                0x2C6E1044 ,
                0x6C6E003C ,
                0x00B445B0 ,
                0x008A2C6E ,
                0x80008C6E ,
                0x000CEFC0 ,
                0x40400000 ,
                0x000A10F0 ,
                0x000B0000 ,
                0x000C0000 ,
                0x00000000 ,
                0x00000000
```

6.6 Configuration Data File

By using the –cfg option, a sequence of the SET commands can be included at the beginning of the AIS output data file. This allows the option to configure DDR, EMIF, PLL to enable proper boot from/to external memories. The commands in this file precede any other AIS data that is generated. The data in the configuration file is not parsed by the genAIS tool. It is simply passed directly through to the output file. Take care to ensure that a correct data sequence appears in the file.

7 Determining On-Chip Bootloader Version

The bootloader version can be found by reading ROM location 0×0080ff00. There are currently two versions extant, version 0×00003060 and version 0x00003070. Version 0x00003070 contains full implementations for I2C (DMP master), UART (DMP slave), and Ethernet (DMP slave) boot modes. These modes are not fully operational in version 0x00003060.

8 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 24 may help you determine the cause of the failure.

Table 24. Debugging Boot Failures

Value	Name	Description
1	ERR_UNKNOWN_COMMAND	An invalid AIS command was encountered in the boot image
2	ERR_BAD_MAGIC_NUMBER	Not used
3	ERR_TRANSMIT_SYNC	Not used
4	ERR_BAD_CRC	In PCI boot, indicates s bad checksum in auto-init table. Otherwise CRC failure in AIS modes.
5	ERR_INVALID_ADDRESS_SIZE	In SPI boot, and invalid number of address bytes specified in first word.
6	ERR_UNSUPPORTED_BOOTMODE	Boot pin configuration is invalid
7	ERR_TIMEOUT_WAITING_FOR_HOST	A timeout was encountered in UART or PCI boot modes
8	ERR_TIMEOUT_I2C_BUS_BUSY	I2C bus reported as busy and did not become availiable during 4096 retries.
9	ERR_TIMEOUT_MCBSP_SPI_RECEIVE	Not used
10	ERR_NAND_ACCESS_TIMEOUT	Not used
11	ERR_RECEPTION_ERR	Various issues with receiving data in UART, PCI, and I2C boot modes
12	ERR_BAD_FUNCTION_PTR	One of various internal function pointers were found to be NULL – may indicate a ROM issue.
13	ERR_PLL_LOCKUP	PLL configuration failed during FASTBOOT setup
14	ERR_CFG_FUNCTION_CALL	Too many arguments specified for function execute function, or invalid function number.

9 Calculating CRC

The on-chip bootloader uses a 32-bit CRC. The 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. The final CRC is a combination of the CRCs 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 *crc* value calculated is the value that should be written as the expected CRC for the REQUEST_CRC command. If calculating a single CRC for the entire application load, simply pass each successive *crc*” value into the subsequent calls to BL_updateCRC.

```
typedef  struct {
                 unsigned int sectionAddr;
                 unsigned int sectionSize;
                 unsigned int *sectionData;
} SectionDatObj;
SectionDataObj mySections[10];
unsigned int crc;
crc = 0;
for(i=0;i<10;i++) {
   crc = BL_updateCRC(&(mySections[i].sectionAddr), 4, crc);
   crc = BL_updateCRC(&(mySections[i].sectionSize), 4, crc);
   crc = BL_updateCRC(mySections[i].sectionData, mySections[i].sectionSize, crc);
}
```

Appendix A Calculating the CRC

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

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

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

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

    if(section_size & 3)
    {

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