Application Report
SPRAA69A–July 2005

Using the TMS320C672x Bootloader

Karen Baldwin *DSP EEE*

ABSTRACT

This application report describes the design details about the TMS320C672x bootloader and addresses parallel flash and HPI boot to the extent relevant.

This application report contains a patch that can be downloaded from this link: http://www-s.ti.com/sc/psheets/sprc203/sprc203.zip .

Contents

List of Figures

List of Tables

1 Introduction

The bootloader resides in the internal ROM of TMS320C672x devices. It starts from the beginning of ROM address space 0x00000000. After reset, the device will set the program counter to the beginning of the ROM address and begin execution of the bootloader.

The following is the list of boot modes that will be supported by the bootloader:

- HPI
- Parallel Flash
- SPI Master
- I2C Master
- SPI Slave
- I2C Slave

When booting in master mode, the bootloader reads the boot information from the slave device if and when required. On the other hand, when booting in slave mode, the bootloader depends on the master device to feed boot information if and when required.

Figure 1 illustrates the basic bootloader functionality:

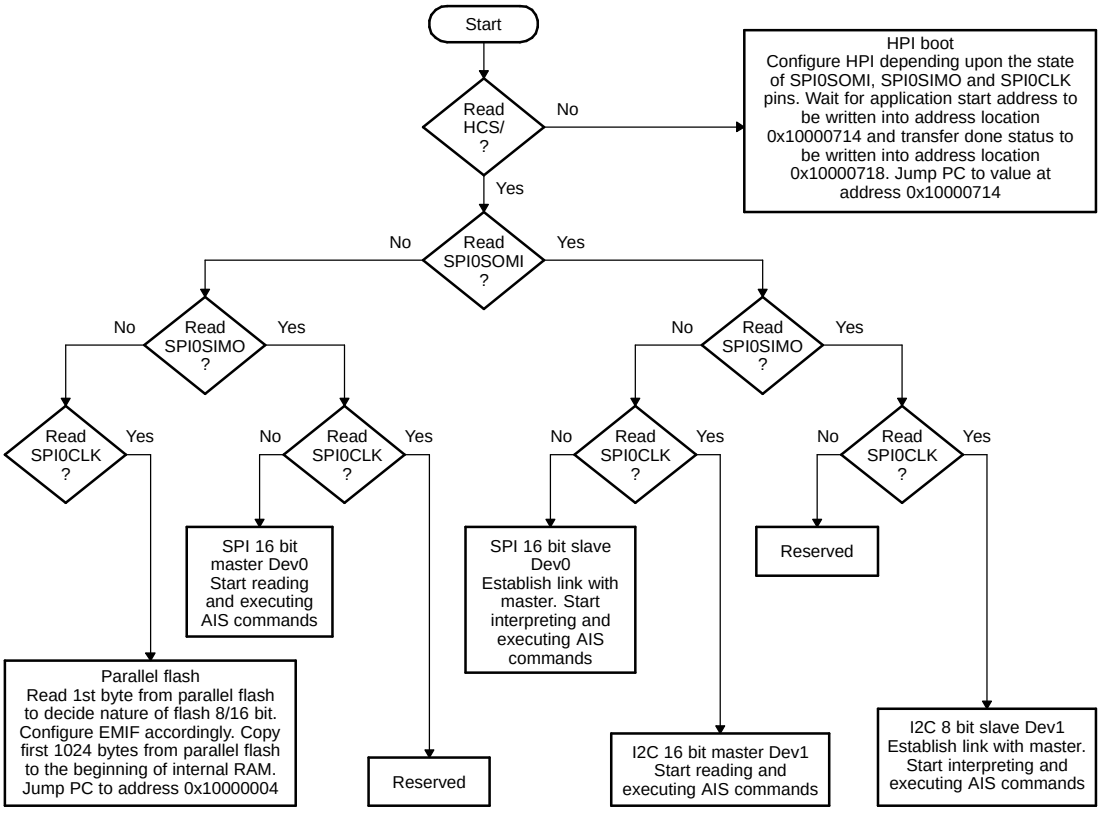


Figure 1. Bootloader Flow

Table 1. Terms Used in This Document

Term	Description
Bootloader	Bootloader Code for TMS320C672x
AIS	Application Image Script
BL	Boot Loader, referring to the bootloader in this text
DSP	Digital Signal Processor, referring to TMS320C672x in this text
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

2 Boot Mode Description

The selection of the following boot modes depends upon the status of boot device pins documented below. The device captures the status of these pins on the rising edge of reset into the registers CPGPIN0 and CFGPIN1. The bootloader refers to CFGPIN0 and CFGPIN1 in order to get the status of the boot device pins. For the sake of clarity, this text refers to the boot device pins instead of their corresponding bit positions in one of the CFGPIN registers.

Table 2. CFGPIN0 Register Definition

CFGPIN0 bit	Bit Name	Corresponding Pin
31:8	Reserved	Not implemented
7	PINCAP7	SPI0SOMI/SDA0
6	PINCAP6	SPI0SIMO
5	PINCAP5	SPI0CLK/SCL0
4	PINCAP4	SPI0SCS/ / SCL1
3	PINCAP3	SPI0ENA/ / SDA1
2	PINCAP2	SPI1SOMI
1	PINCAP1	SPI1SIMO
0	PINCAP0	SPI1CLK

Table 3. CFGPIN1 Register Definition

CFGPIN1 bit	Bit Name	Corresponding Pin
31:8	Reserved	Not implemented
7	PINCAP15	SPI1SCS/
6	PINCAP14	SPI1ENA/
5	PINCAP13	HCS/
4	PINCAP12	HD[0]
3	PINCAP11	HA[0]
2	PINCAP10	AFSX0
1	PINCAP9	AFSR0
0	PINCAP8	AXR0[0]

Table 4 summarizes boot mode pin configuration.

Note:

The order of different boot modes in the table is not the same as when they are detected.

Table 4. Boot Device Pin Allocation

Boot Mode Description			Boot Pin Allocation			
Boot Mode Description	**Data Bits**	**Add Bits**	**BL1 boot (HCS/)**	**SPI0SOMI**	**SPI0SIMO**	**SPI0CLK**
HPI(1)			0	X	X	X
Parallel Flash(2)	-	-	1	0	1	0
SPI0 Master	8	16	1	0	0	1
SPI0 Slave	16	-	1	0	1	1
Reserved	-	-	1	1	0	0
I2C1 Master	8	16	1	1	0	1
Reserved	-	-	1	1	1	0
I2C1 Slave	8	-	1	1	1	1

(1) Additional pins used to configure the boot mode. For details, refer to Section 2.1.
(2) First byte in parallel flash gives information on the Data/Address bits. For details, refer to Section 2.2.

2.1 HPI Boot

Once selected, the bootloader initializes the “Bootloader HPI Jump Address Register (0x10000714)” and “Bootloader HPI Transfer Done Register (0x10000718)” with zeros. Then, it resets the CSP Bridge and configures the HPI to run in the desired mode depending on the state of the following pins:

Table 5. HPI Configuration Based on Device Pins

Device Pin	Corresponding bit name in CFGHPI register
SPI0SOMI	BYTEAD
SPI0SIMO	FULL
SPI0CLK	NMUX

For more information on these bits, refer to the CFGHPI register description in the official TMS320C672x specification in *TMS320C6727, TMS320C6726, TMS320C6722 Floating-Point Digital Signal Processors* (SPRS268).

After configuring the CFGHPI register based on the logical states of the above mentioned pins, the bootloader enables the HPI peripheral by writing a 1 to the ENA bit in the CFGHPI register.

Then, it waits for data to be sent through the standard HPI protocol to the DSP. Once complete, the host writes the application entry address into the memory location 0x10000714 and a 1 to address location 0x10000718 to signal the completion of HPI bootloading. The bootloader continuously polls the address location 0x10000718; and as soon as it finds a non zero value at the address location, it jumps the program counter to value at address 0x10000714 and begins execution of the application.

2.2 Parallel Flash

The first 8-bits on the flash device gives information about the data bits (8/16) that can be accessed from the parallel flash simultaneously.

Figure 2. Parallel Flash Signature Format

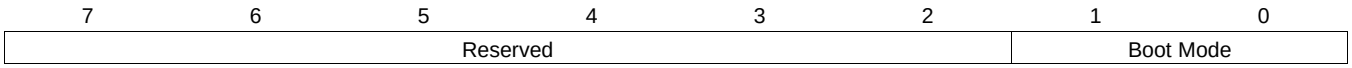


Table 6. Parallel Flash Boot Mode Field

Boot Mode	Description
00	8-bit parallel flash
01	16-bit parallel flash
10	Reserved
11	Reserved

Once selected, the bootloader reads the first byte from the flash device to determine the boot mode. The bootloader copies the first 1024 bytes of data to first 1Kbyte of internal memory and jumps the program counter to point at address offset 0x4 in internal memory. Execution begins at this address and contains source for user defined secondary bootloader or other application code as required.

2.3 I2C Master

The bootloader attempts to read a magic word (0x54495041) through I2C on address 0x50. If the magic word is not obtained during the read call, the detection of this boot mode fails and the bootloader logic ends up in an infinite while loop.

The bootloader supports devices obeying the ATMEL serial I2C EEPROM protocol. For more details about ATMEL serial I2C protocol, refer to http://www.atmel.com/products/Serial/. The data burnt into the EEPROM has to be in AIS format.

2.4 I2C Slave

The DSP I2C peripheral has its own address set of 0x29. The host (master) is required to establish a link with the DSP in the beginning. The host begins transmitting data in application image script (AIS) format. For details about link establishment, refer to Section 5. For details about, AIS refer to Section 3.

2.5 SPI Master

The bootloader attempts to read a magic word (0x54495041) through SPI. If the magic word is not obtained during the read call, the detection of this boot mode fails and the bootloader logic ends up in an infinite while loop.

The bootloader supports devices obeying the ATMEL serial SPI EEPROM protocol. For more details about ATMEL serial SPI protocol, refer to http://www.atmel.com/products/Serial/. The data burnt into the EEPROM has to be in AIS format.

2.6 SPI Slave

The host (master) is required to establish a link with the DSP in the beginning. The host begins transmitting data in AIS format. For details about link establishment, refer to Section 5. For details about AIS, refer to Section 3.

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 proprietary application image transfer format. This script is a binary file consisting of a script header followed by various commands that are 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 supports version AIS Version 1.0.

The AIS starts with a magic word (0x54495041), followed by a series of commands shown in Figure 3. Each command, in turn, consists of an op-code followed by optional additional data.

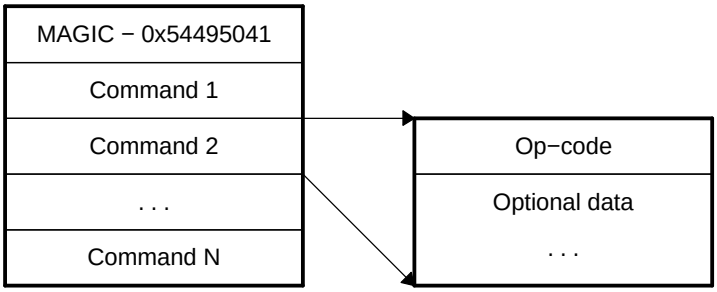


Figure 3. Basic Structure of Application Image Script

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

Table 7. AIS Version 1.0 Supported Opcodes

Opcode	Value
Section Load	0x58535901
Request CRC	0x58535902
Enable CRC	0x58535903
Disable CRC	0x58535904
Jump	0x58535905
Jump Close	0x58535906
Boot table	0x58535907
Start Over	0x58535908
Reserved	0x58535909
Section Fill	0x5853590A
Ping	0x5853590B

3.1 Boot Table Command

Boot table commands are a simple mechanism that enables the user to write 8-bit, 16-bit, or 32-bit data to any address in DSP address space. There is a provision to provide delay after the memory write happens. This can be used for memory mapped register write to take effect. Boot table commands are used to configure various peripherals of DSP which includes PLL and EMIF at minimum and can configure more peripherals, if required.

When the DSP is powered-up, the PLL multiplier is bypassed and PLL Divider D1 is set to divide-by-1. 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. In order to reduce boot time, PLL and EMIF registers should be configured at the very beginning of boot process. For this reason, all boot table commands are placed at the beginning of AIS as shown in Figure 4.

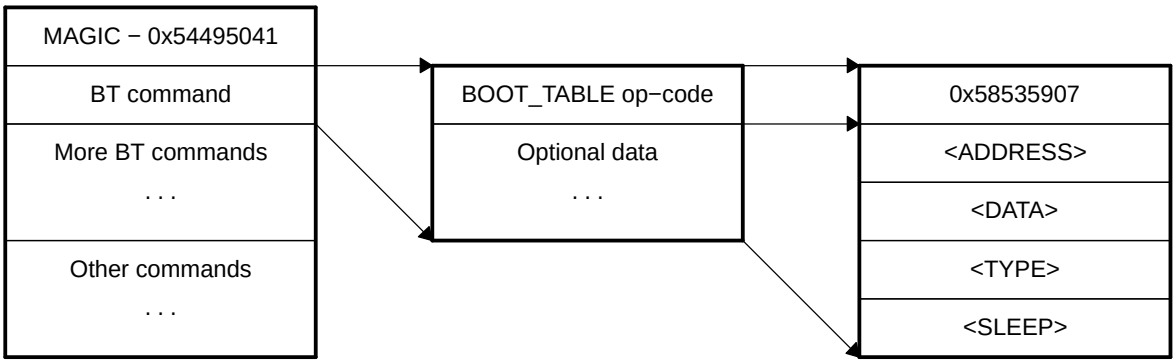


Figure 4. Structure of Boot Table Command

Each boot table command consists of BOOT_TABLE (0x58535907) op-code, followed by four words of additional data as shown. BOOT_TABLE entries in AIS are explained using the following representation:

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

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

Table 8. Numeric Formats that can be used in BTEs

Name	Format	Example 1	Example 2	Example 3
Hexadecimal	0[xX][0-9a-fA-F]+	0x1234abCD	0x1000	0X5a
Hexadecimal	[0-9a-fA-F]+[hH]	1234ABCDh	1000H	5ah
Octal	0[0-7]+	02215125715	010000	0132
Decimal	[0-9]+	305441741	4096	90

3.2 Section Load Command

Section load commands are used to load a particular chunk of code/data to DSP memory. All initialized sections (such as .text) are loaded to DSP memory using Section load commands. These commands are placed after all Boot table commands in AIS.

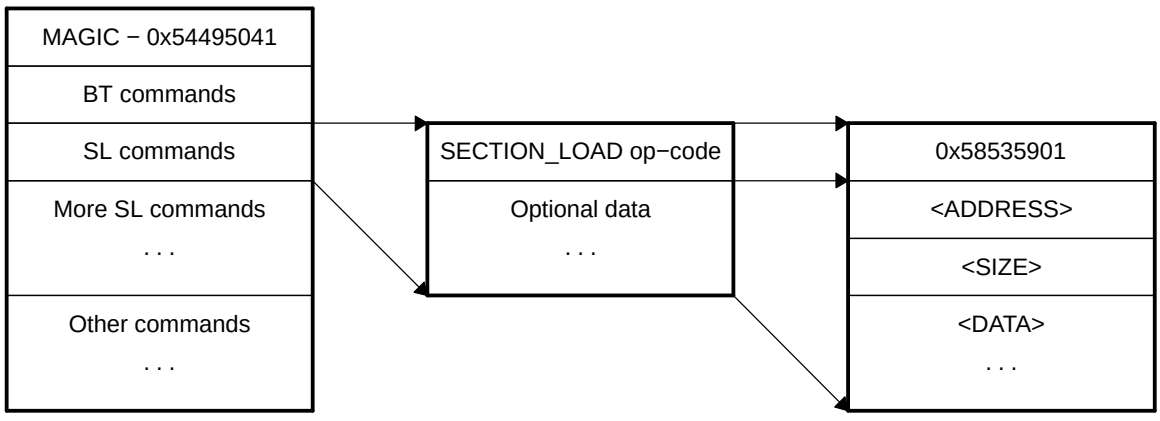


Figure 5. Structure of Section Load Command

Each Section Load command consists of SECTION_LOAD (0x58535901) op-code, followed by section's start address, size and contents.

3.3 Section Fill Command

Section Fill command are used when a particular section is filled with a specific pattern. For example, a section that contains all zeros is initialized with Section Fill command. These commands are placed anywhere where a regular Section Load command occurs.

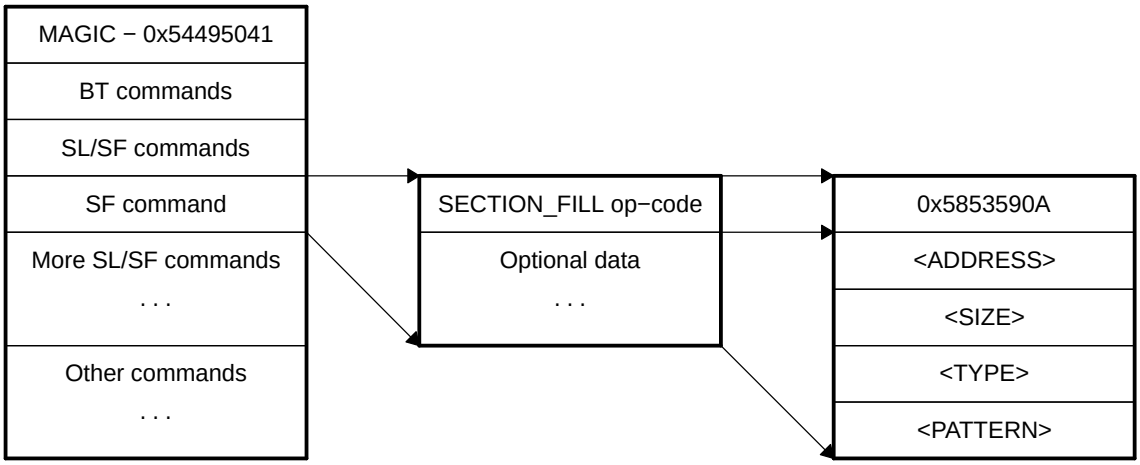


Figure 6. Structure of Section Fill Command

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.

3.4 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.

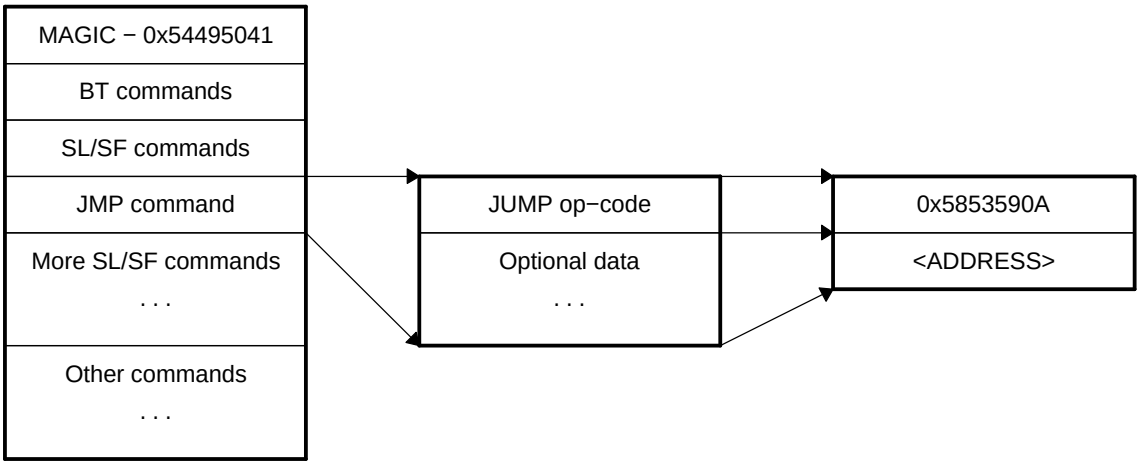


Figure 7. Structure of Jump Command

This command may be used to execute code that has been previously loaded through Section Load and Section Fill commands. It could be used to implement a secondary load process or to execute application code necessary to setup other processes before remainder of code/data is loaded. Once, the JUMP command is issued, execution will begin at the indicated start address. When execution is over, it is the responsibility of the application code to execute a return instruction. This enables return of control to the on-chip bootloader. Normal AIS interpretation and execution will continue at that point.

3.5 Jump N Close Command

This command is used at the end of the boot process to start execution of loaded application. This command instructs the DSP to terminate boot process and jump to start address of loaded application.

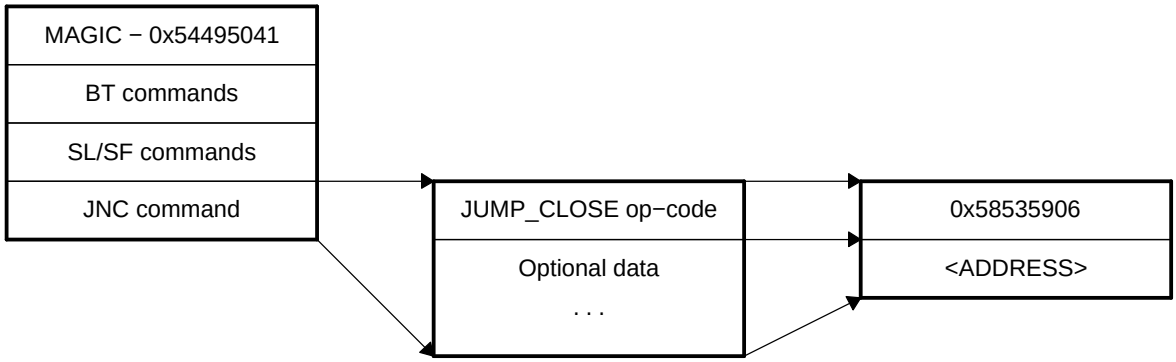


Figure 8. Structure of Jump N Close Command

The 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 bootloader should jump.

3.6 CRC Options

There is a possibility of error in communication when 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 three options available are:

No CRC

With this option, CRC computation is disabled and there is no way to detect or correct any error.

Single CRC

With this option, single CRC will be computed for all the sections. Verification will be done at the end, just before Jump N Close command. In case of error, all the sections are loaded again. CRC will be recalculated and re-verified again at the end.

Section-Wise CRC

With this option, CRC is computed for each section. Verification is done at the end of each section. The section is reloaded in case of error.

3.6.1 Enable/Disable CRC Commands

These commands are used to enable/disable computation of CRC for sections loaded through Section Load / Section Fill commands.

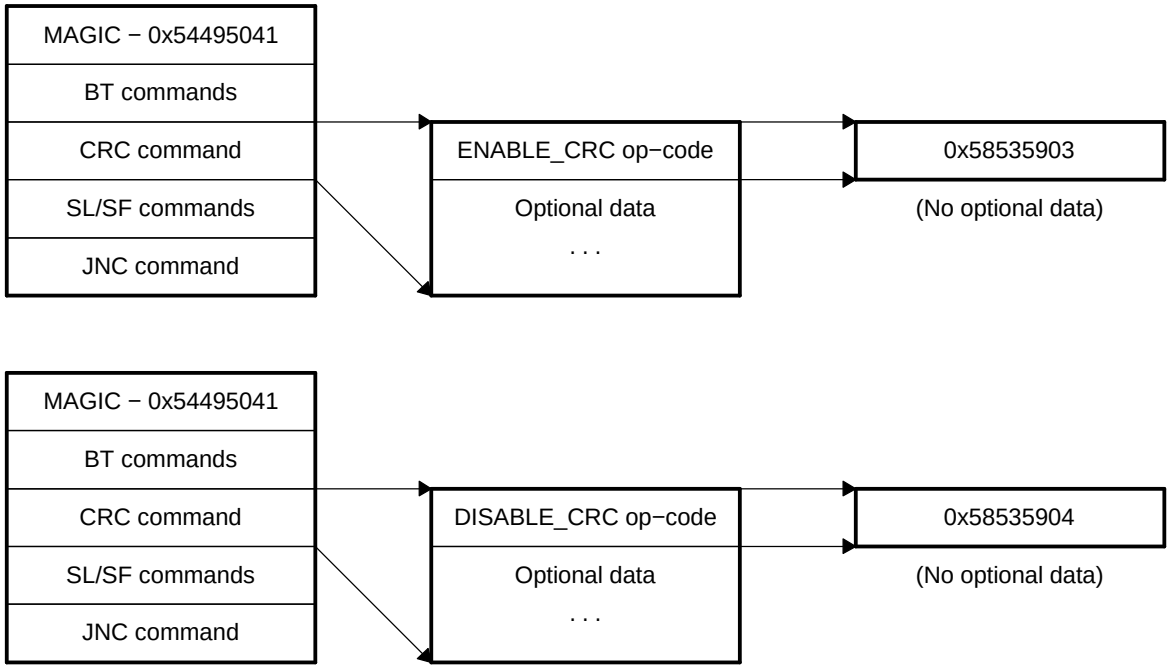


Figure 9. Structure of Enable CRC / Disable CRC Commands

These commands consist of only a single ENABLE_CRC (0x58535903) or DISABLE_CRC (0x58535904) op-code. There is no additional data required.

3.6.2 Request CRC Command

This command is used to request and validate current value of CRC computed by DSP. Using this command requires that the Enable CRC command is issues earlier in the AIS stream. This command consists of REQUEST_CRC(0x58535902) op-code, followed by the expected CRC value and a seek-value. The CRC of loaded/filled section(s) are compared wit the expected CRC value. If the CRC is correct, seek-value is ignored and execution shall continue from next command.

A mismatch in CRC indicates that the data loaded to the DSP memory using earlier Section Load/ Section Fill commands is corrupted. In order to load data again, AIS has to be re-executed from the last error-free point (i.e last valid command). The seek-value is expressed in bytes, and is a negative number that is added to the current address in AIS. By adding the seek-value, the AIS stream is then pointed back to the last known good state and AIS interpretation continues from this address.

When operating in master mode, CRC read/compare/seek adjustment are performed automatically by the bootloader. In slave mode operation, a REQUEST_CRC command results in the bootloader transmitting the current CRC value calculated by the DSP to the Host. The Host may then send a Start-Over command as described in next section. On receiving the Start-Over command the DSP knows that CRC error has occurred. It resets its CRC computation and becomes ready to accept more command from the host. The next command expected by the DSP is a PING command, followed by Host/slave mode exchange (XMT_START/RECV_START).

Please refer to Appendix A for code used to calculate CRC values.

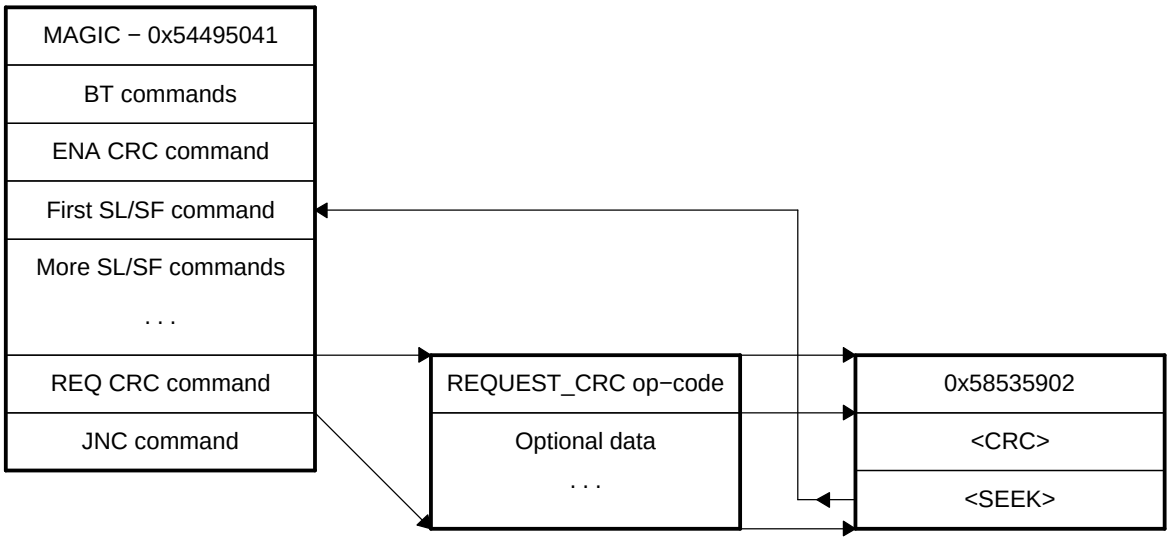


Figure 10. Structure of Request CRC Command for SIngle CRC Option

For 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, will make offset point to start of the first Section Load / Section Fill command as shown.

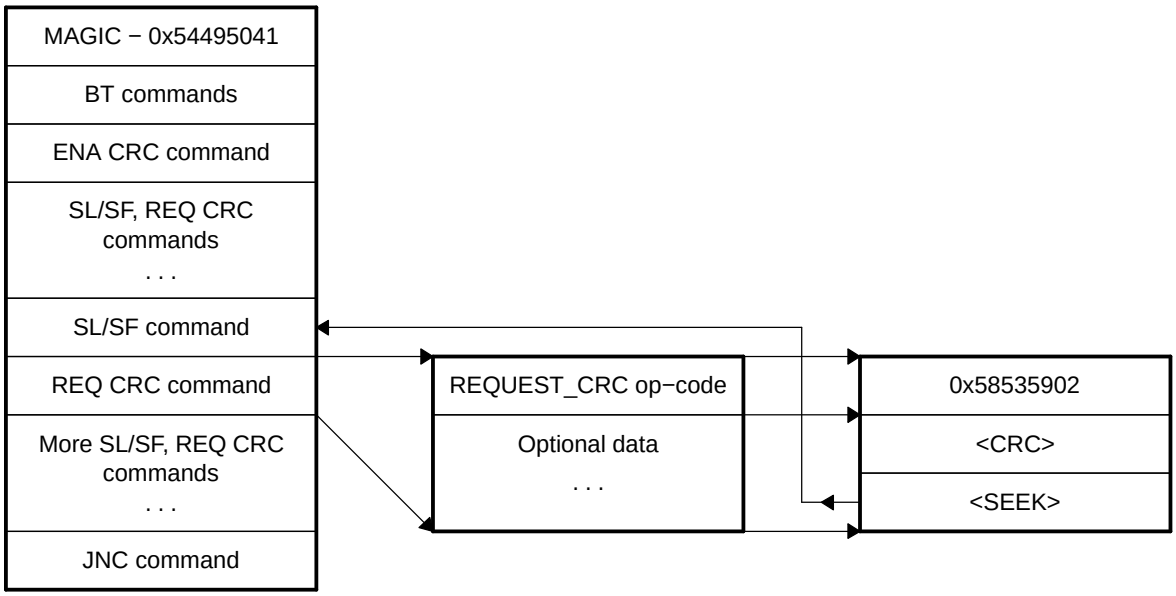


Figure 11. Structure of Request CRC Command for Section-wise CRC Option

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 current offset in AIS, will make offset point to start of the previous Section Load / Section Fill command as shown.

3.6.3 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.

It has to be issued by the host on its own when it detects a CRC mismatch for slave modes. For master modes, this is taken care by the bootloader state machine.

4 External Serial EEPROM Boot

The bootloader contains the AIS interpreter for parsing the data read from the serial EEPROM. After parsing the data retrieved, the bootloader takes appropriate actions in order to execute the opcode.

5 External Host Processor Boot

When booting from the external host processor, the host processor acts as a boot master and the DSP acts as slave. Since the DSP does not have direct access to AIS, the host processor has to transfer it to the DSP through a well-defined protocol explained in following sections. An AIS interpreter is required on the host processor to control this transfer.

5.1 AIS Interpreter on the Host

The AIS interpreter on the host is responsible for transferring the AIS to the DSP. For this, it has to understand the transfer protocol and implement the required handshake mechanism. The AIS interpreter on host directly interacts with the bootloader on the DSP.

Note:

For the sake of simplicity, AIS interpreter on host will be simply referred to as ‘host’ and bootloader on TMS320C672x devices as ‘DSP’ in this section.

It is important to have a successful link establishment between the DSP and the host before starting transfer of AIS. Once the link is established, AIS is transferred to DSP. The whole process is divided into three phases:

- start-word synchronization (SWS)
- ping op-code synchronization (POS)
- op-code synchronization (OS)

5.2 Start-Word Synchronization

Start-Word Synchronization (SWS) is the default power-up state and is responsible for initiating communication between the DSP and the host.

The bootloader tries to read the transmit start-word (XMT_START) from the host. After receiving it, the DSP acknowledges by sending receiver-start-word (RECV_START) to the host.

The XMT_START and RECV_START can be 8-bit, 16-bit, or 32-bit depending on the boot mode used. The following table shows corresponding values for different boot modes:

Boot Mode	XMT_START	RECV_START
8-bit	0x58	0x52
16-bit	0x5853	0x5253
32-bit	0x58535441	0x52535454

The host must keep on sending XMT_START until it receives RECV_START from the DSP. This process initiates communication between the DSP and the host. Figure 12 shows the flowchart of how SWS is implemented on the host.

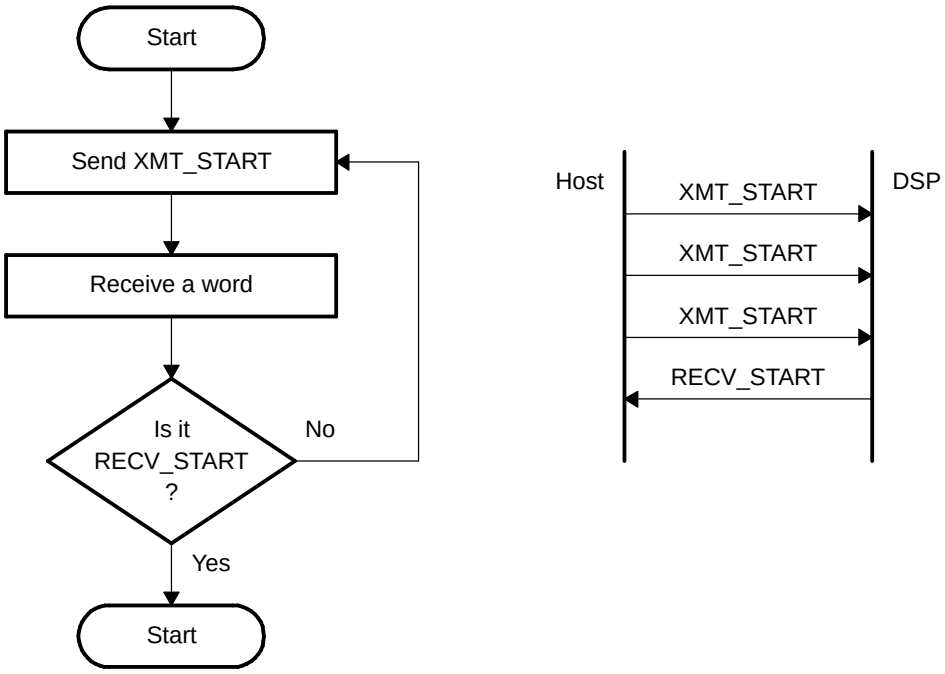


Figure 12. Flowchart: Start-Word Synchronization

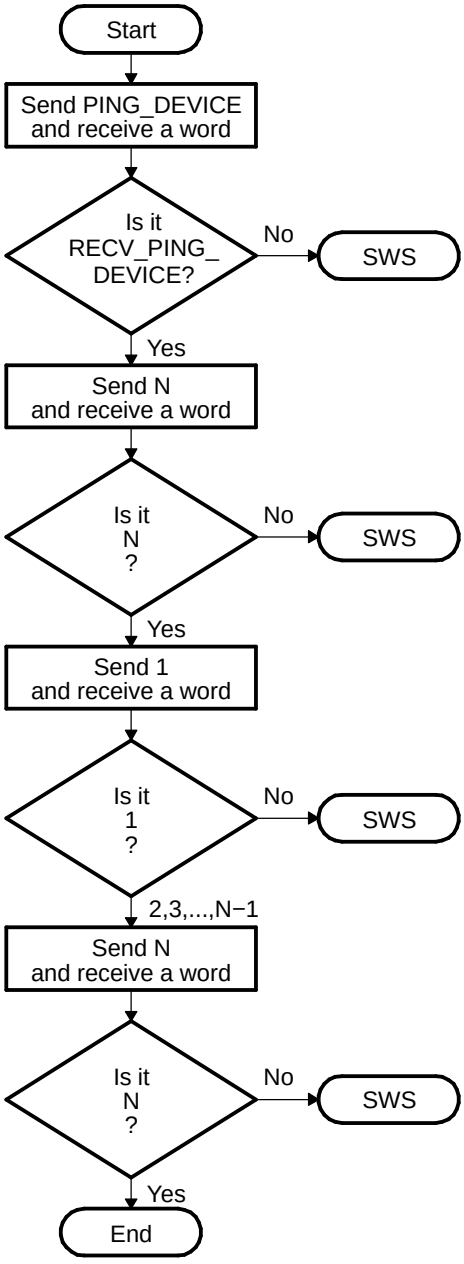
5.3 Ping Op-code Synchronization

Ping Op-code Synchronization (POS) is used to make sure that the boot mode selected is correct and the communication link between the host and the DSP is reliable.

After successful SWS,

- DSP waits for the host to send PING_DEVICE (0x5853590B) op-code. On receiving it, DSP acknowledges it by sending RECV_PING_DEVICE (0x5253590B) to the host.
- The host then sends a number N to the DSP and gets back the same number from the DSP as acknowledgment.
- The host shall then start sending numbers 1 to N to DSP and will receive the same sequence as acknowledgment.

POS is implemented as a simple bootloader command and it can be issued any time during AIS transfer to check reliable communication. If POS fails at any point, the DSP and the host are required to start all over again from SWS. Figure 13 shows the flowchart of how POS is implemented on the host.



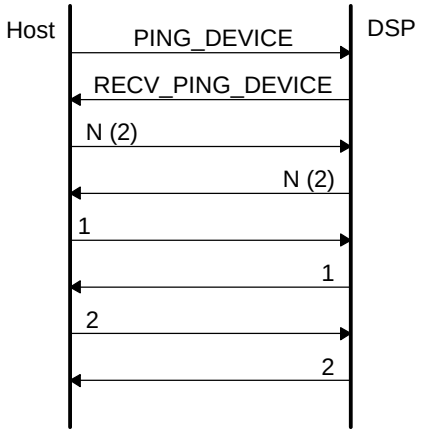


Figure 13. Flowchart: Ping Op-code Synchronization

5.4 Op-code Synchronization (OS) for Serial Slave Modes

After a successful link establishment, the DSP and the host are ready to transfer AIS commands. Since all AIS commands start with an op-code, the DSP waits to receive one of valid op-codes from the host. For serial slave modes on receiving an opcode, the DSP acknowledges by sending corresponding RECV opcode. This process is referred to as opcode synchronization.

All opcodes (including PING_DEVICE) transmitted by the host to the DSP are of the form 0x585359##, where ## varies for individual op-codes. DSP acknowledges each op-code by corresponding RECV op-code. RECV op-codes are generated from the original op-codes by changing the most significant byte to 0x52. Thus, they are of the form 0x525359##.

Not getting a correct response (RECV op-code) from DSP means that the DSP is busy executing an earlier op-code. The host should continue sending the op-code until successfully acknowledged by the DSP. Figure 14 shows the flowchart of how OS is implemented on the host.

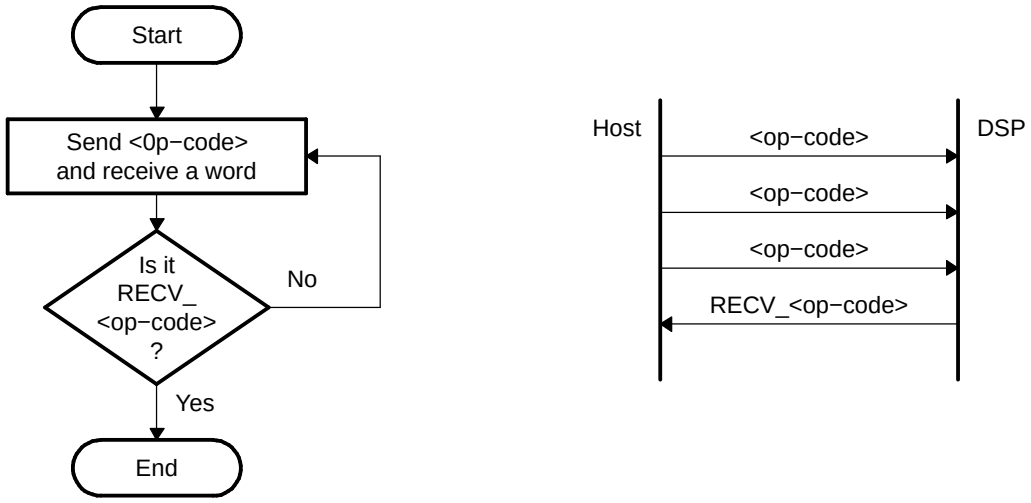


Figure 14. Flowchart: Op-code Synchronization

DSP starts executing the op-code only after the OS is finished. If more information is required in order to execute the op-code, the DSP gets it from the host before starting execution. The host is required to understand each op-code and supply required data to the DSP from the AIS.

The DSP keeps on executing commands from the host until it gets a Jump N Close (JNC) command, using Op-code synchronization at the beginning of each command. On getting JNC command, the DSP closes the peripheral used for booting, terminates the boot process and jumps to the address specified along with the op-code.

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 8bit bytes, and “crc” is 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 );
}
```

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

Copyright © 2005, Texas Instruments Incorporated