TMS320VC5402A/VC5409A/VC5410A/VC5416 Bootloader

Tai Nguyen *C5000 Applications*

ABSTRACT

This document describes the features and operation on the TMS320VC5402A, TMS320VC5409A, TMS320VC5410A, and TMS320VC5416 Digital Signal Processors. The contents of the on-chip ROM are discussed.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SPRA602.

Important Notice Regarding Bootloader Program Contents:

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

Contents

List of Figures

List of Tables

1 Introduction

This section describes the purpose and features of the TMS320VC5402A, TMS320VC5409A, TMSVC5410A, and TMS320VC5416 digital signal processor (DSP) bootloader. It also discusses the other contents of the device on-chip ROM and identifies where all of this information is located within that memory.

1.1 Bootloader Features

The DSP bootloader is used to transfer code from an external source into internal or external program memory following power-up. This allows code to reside in slow non-volatile memory externally, and be transferred to high-speed memory to be executed. This eliminates the need for mask programming the DSP internal ROM, which may not be cost effective in some applications.

The bootloader provides a variety of different ways to download code to accommodate different system requirements. This includes multiple types of both parallel bus and serial port boot modes, as well as bootloading through the HPI, allowing for maximum system flexibility. Bootloading in both 8-bit byte and 16-bit word modes is also supported.

The bootloader uses various control signals including interrupts, BIO, and XF to determine which boot mode to use. The boot mode selection process, as well as the specifics of bootloader operation, are described in detail in section 2 of this document.

1.2 On-Chip ROM Description

On the DSP device, the on-chip ROM contains several factory programmed sections including the bootloader itself and other features. The sections contained in the ROM are:

- **Bootloader program.** (Described in this document)
- **256-word μ-law and A-law expansion tables.** Lookup tables that are used for decoding.
- **256-word sine look-up table.** Table consisting of 256 signed Q15 integers, representing 360 degrees.
- **Factory test code.** Reserved code used by TI for testing the device.
- **Interrupt vector table.** Code that allows indirect vectoring of interrupts

The DSP on-chip ROM memory map is shown in Table 1. The ROM is 16K words in size and is located at the 0xC000 - 0xFFFF address range in program space when the MP/MC input pin is low.

Table 1. DSP On-chip ROM Program Space

Starting Address	Contents
000_C000	Reserved
000_ F800	Bootloader code
000_FC00	μ-law expansion table
000_FD00	A-law expansion table
000_FE00	Sine lookup table
000_FF00	Factory test code
000_FF80	Vector table

2 Bootloader Operation

2.1 Boot Mode Selection

The bootloader program located in the DSP on-chip ROM is executed following reset when the device is in microcomputer mode (MP/MC = 0).

The function of the bootloader is to transfer user code from an external source to the program memory at power up. The bootloader sets up the CPU status registers before initiating the boot load. Interrupts are globally disabled (INTM = 1) and the internal dual- and single-access RAMs are mapped into the program/data space (OVLY = 1). Seven wait states are initialized for the entire program and data spaces. The bootloader does not change the reset condition of the bank switching control register (BSCR) prior to loading a boot table.

To accommodate different system requirements, the DSP offers a variety of different boot modes. The following is a list of the different boot modes implemented by the bootloader, and a summary of their functional operation:

- **Host Port Interface (HPI) Boot Mode:**

  The code to be executed is loaded into on-chip memory by an external host processor via the Host Port Interface. Code execution begins once the execution address loading is completed.

- **Parallel Boot Modes (8-bit and 16-bit supported):**

  The bootloader reads the boot table from data space via the external parallel interface bus. The boot table contains the code sections to be loaded, the destination locations for each of the code sections, the execution address once loading is completed, and other configuration information.

- **Standard Mode Serial Port Boot Modes (8-bit and 16-bit supported):**

  The bootloader receives the boot table from one of the multi-channel buffered serial ports (McBSP) operating in standard mode and loads the code according to the information specified in the boot table. McBSP0 supports 16-bit serial receive mode. McBSP2 supports 8-bit serial receive mode. McBSP1 is reserved for a future additional serial boot mode.

- **8-bit Serial EEPROM Boot Mode:**

  The bootloader receives the boot table from a serial EEPROM connected to McBSP2 operating in clock-stop (SPI) mode and loads the code according to the information specified in the boot table.

- **I/O Boot Mode (8-bit and 16-bit supported):**

  The bootloader reads the boot table from I/O port 0h via the external parallel interface bus employing an asynchronous handshake protocol using XF and BIO. This allows data transfers to be performed at a rate dictated by the external device.

  The bootloader also offers the following additional features:

  – **Reprogrammable software wait state register**

    In the parallel and I/O boot modes, the bootloader reconfigures the software wait state register based on a value read from the boot table during the bootload.

  – **Reprogrammable bank switching control register**

    In the parallel and I/O boot modes, the bootloader reconfigures the bank switching control register based on a value read from the boot table during the bootload.

  – **Multiple-section boot**

    The DSP bootloader is capable of loading multiple separate code sections. These sections are not required to occupy a continuous memory space as in some previous C54x bootloaders.

The details of all of these boot modes are described in the following sections.

Once the bootloader is initiated, it performs a series of checking operations to determine which boot mode to use. The bootloader first checks for conditions that indicate it should perform an HPI boot. If the conditions are not met, it goes to the next mode and continues until it finds a mode which is selected. The flowchart shown in illustrates the process the bootloader uses to determine the desired boot mode.

The bootloader checks each of the boot modes in the following sequence until indications for a valid boot mode are found:

1. Host port interface (HPI) boot mode
2. Serial EEPROM boot mode (8-bit)
3. Parallel boot mode
4. Standard serial boot mode via McBSP2 (8-bit)
5. Standard serial boot mode via McBSP0 (16-bit)
6. I/O boot mode

Detailed description of the operation of each of these boot modes can be found in section 2.2, Boot Mode Options.

The first boot mode tested is the HPI boot mode. Unlike some of 54x HPI boot modes, the 'DSP HPI boot mode doesn't require the setting of the interrupt 2 (INT2) flag for selection. If INT2 is not active, the bootloader periodically checks various boot sources, including the HPI boot, until a boot condition is detected. Alternatively, the INT2 flag can be used to force the bootloader to ignore all boot sources other than HPI. If INT2 is found to be active, the bootloader assumes that the code will be loaded into on-chip RAM by an external host. The bootloader waits while the host loads its on-chip RAM by polling address 00 007Fh of the on-chip data RAM. Once finished loading the on-chip RAM, the host writes the XPC and PC values into data memory location 00 007Eh and 00 007Fh respectively specifying the address of the entry point. **Note that the PC of the entry point must be a non-zero number**. When the bootloader detects the entry point has been programmed, the bootloader branches to the specified start address.

If the INT2 flag is not active, the bootloader proceeds to check the SPI EEPROM boot mode. The serial EEPROM boot mode is checked using the INT3 flag. If INT3 is found to be active, the bootloader assumes that the boot table is located in an 8-bit serial EEPROM connected to McBSP2. The bootloader reads the data value stored at address 0 of the EEPROM. If the data read contains a valid keyword for the beginning of the boot table, the bootloader proceeds to load the remainder of the boot table from the serial EEPROM. If the INT3 flag is not active, the bootloader proceeds to check the next boot mode. If the INT3 flag is active but a valid keyword is not read from the serial EEPROM, the bootloader continues to try to read the keyword until a valid keyword is found. The bootloader will not check for other boot modes at this point. If the serial EEPROM boot is not desired, the user should ensure that the INT3 signal does not occur during the first 30 cycles of the boot load process to prevent the bootloader from entering this non-escapable loop.

The next boot modes tested are the parallel boot modes. The boot table is located in data space and can be located anywhere within the upper 32k data space address range (08000h **-** 0FFFFh), which is the external data memory (DROM = 0, after reset). The bootloader determines the location of the boot table by first reading address 0FFFFh in I/O space. The beginning of the boot table always contains a keyword that indicates whether 8-bit or 16-bit boot mode is desired. The valid keyword for 8-bit boot mode is 08AAh. The valid keyword for 16-bit boot mode is 10AAh. If a valid keyword for the boot table is not found, the bootloader reads address 0FFFFh in data space for a boot table address and checks for a valid keyword at that location. When a valid keyword is found, the bootloader continues to read the remainder of the boot table and loads and executes the application code. If no valid keyword is found, the bootloader proceeds to check the serial boot modes.

Prior to testing for serial or I/O boot modes, the bootloader drives the XF output on the device low to indicate that it is ready to receive data. Upon this event, a serial host device may begin to send the boot table information via McBSP0 or McBSP2. When a word is received on one of the McBSPs, a serial port interrupt flag will be generated in the DSP. The boot loader uses these flags as indication of which McBSP to use for boot load. The DSP reads the data received on the McBSP and if a valid keyword is found, the boot proceeds with the same serial port and no other boot modes are tested. If neither McBSP responds with received data, the bootloader proceeds to check the I/O boot mode.

In I/O boot mode, the host device makes a request to send data by driving the BIO input to the DSP low. The bootloader recognizes this request and reads data from I/O port 0h. If the data read contains a valid keyword for the beginning of the boot table, the DSP and the host proceed to load the remainder of the boot table using a handshaking scheme as described in section 2.2.5, I/O Boot Mode.

If a valid boot mode is not found after checking all of the possible boot modes, the bootloader restarts ad continues to recheck each of the boot modes in the same sequence.

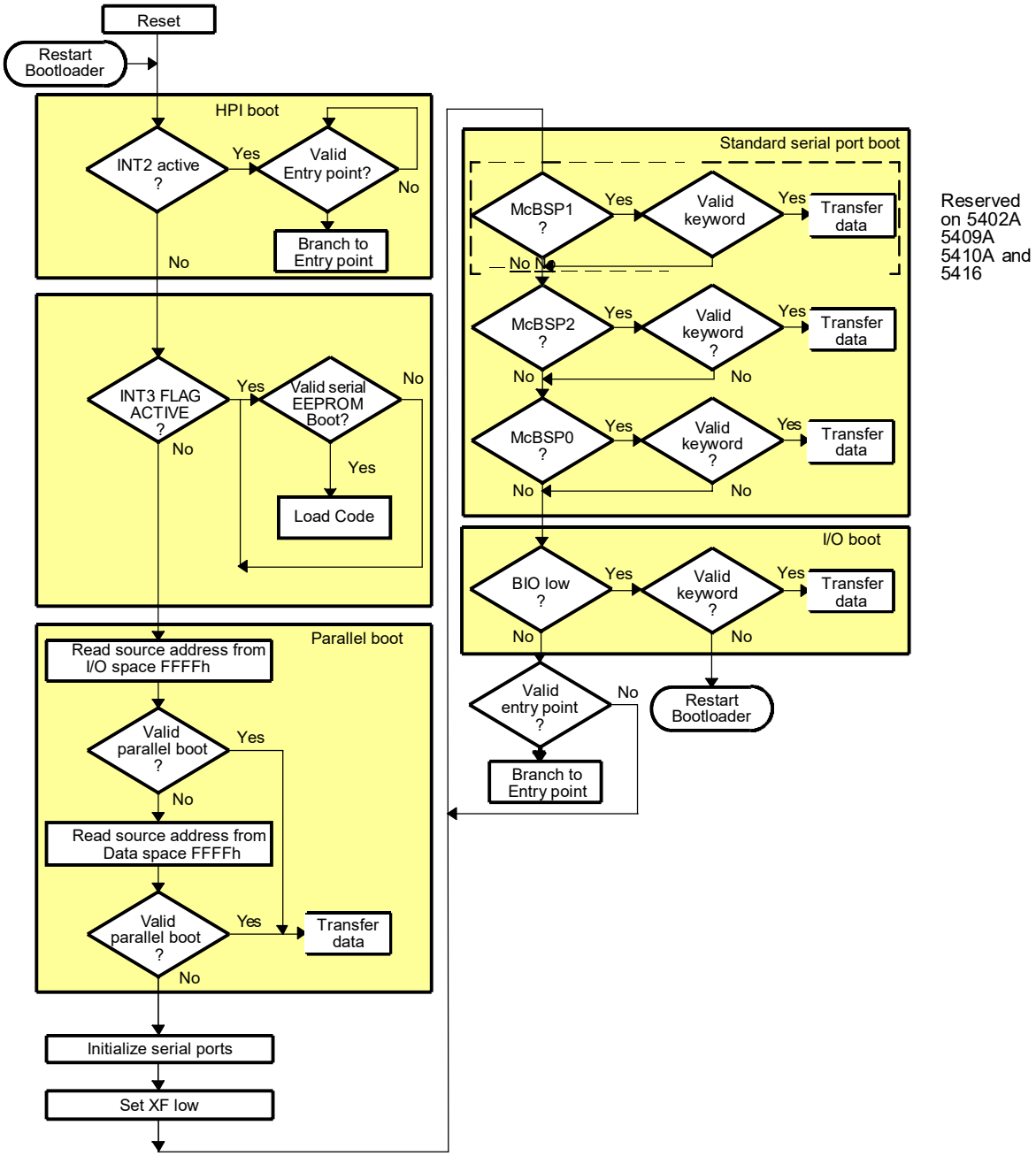

Figure 1. Bootloader Mode Selection Process

The flowchart shown in Figure 2 illustrates the basic process the bootloader uses to determine whether 8- or 16-bit data has been selected, transfer the data, and begin program execution. This process occurs after the bootloader finds what it believes may be a valid boot mode selected. When using the 8-bit boot modes, bytes must be ordered most significant byte first, and for parallel 8-bit modes, the bytes must be presented on the lower-order eight bits of the data bus. Note that the exact sequence used for 8- and 16-bit mode processing differs somewhat depending on which specific boot mode is selected. The particular sequence used by each boot mode is shown in the sections describing the different boot modes in detail.

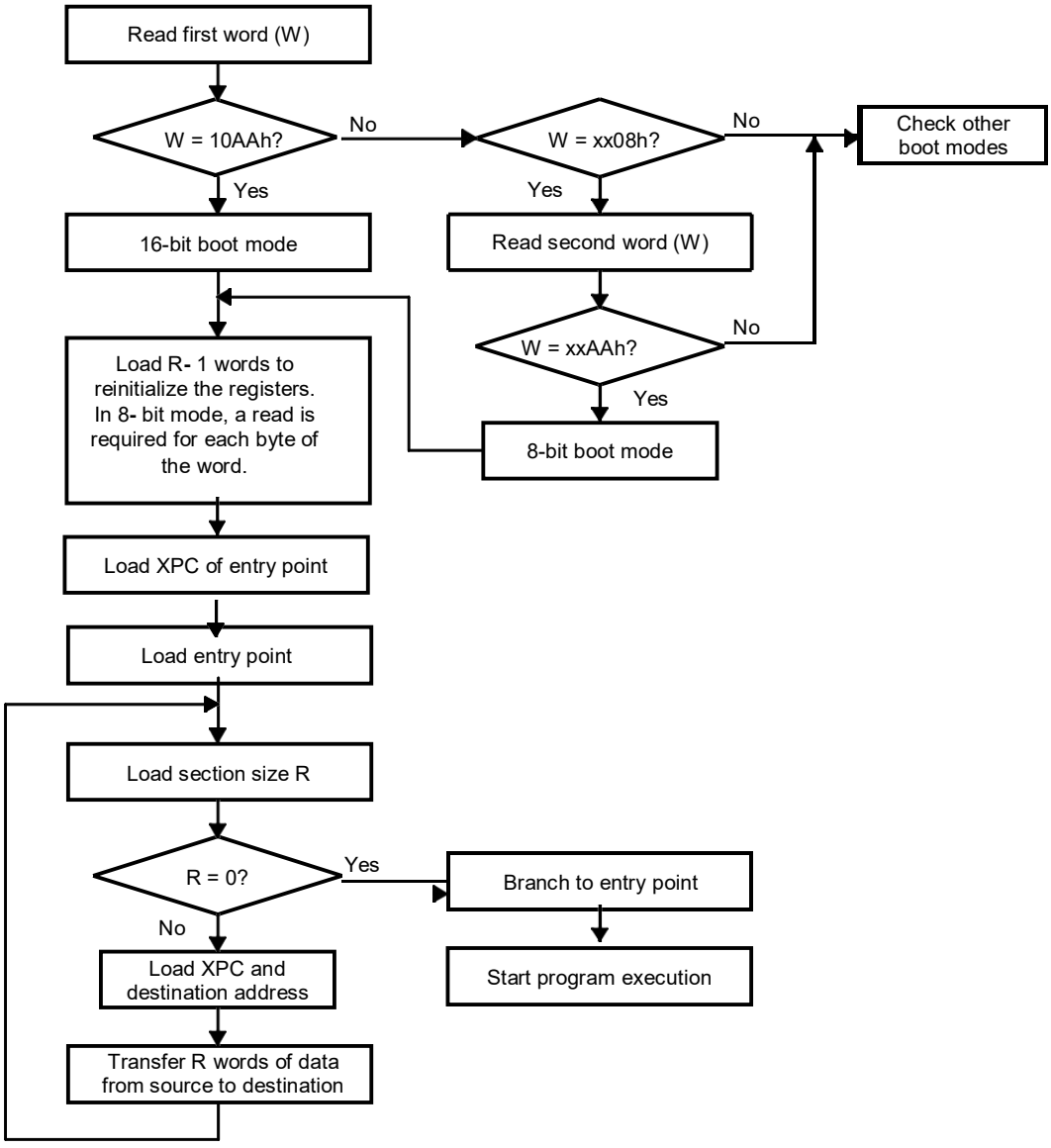

Figure 2. Basic Data-Width Selection and Data Transfer Process

Table 2 and Table 3 show the general structure of the boot table that is loaded for the boot modes. This structure is the same for all the modes, except HPI, in which code is loaded directly into memory and therefore does not require the boot table. The first R- 1 words include the keyword, the words used to initialize the registers (the number of which depends on the boot mode), and the entry point address of the application code. For each load section that follows, there is a word that indicates the block size, followed by two words that indicate the 23-bit load address for the block. When the bootloader encounters a block size of zero (0000h), it branches to the entry address and begins execution of the application.

Note that these tables are only included to show the basic structure of the boot table. The exact structure of the boot table varies depending on the boot mode selected. For specific information about the structure used by each of the boot modes, refer to the sections describing the different boot modes in detail.

Table 2. General Structure of Source Program Data Stream in 16-Bit Mode

Word	Contents
1	10AAh (memory width of the source program is 16 bits)
2	Value to set in the register (applied to the specified boot mode)
.	.
.	Value to set in the register
.	XPC value of the entry point (least significant 7 bits are used as A23-A16)
.	Entry point (PC) (16 bits are used as A15- A0)
R	Block size of the first section to load.
R+1	XPC value of the destination address of the first section (7 bits)
.	Destination address (PC) of the first section (16 bits)
.	First word of the first section of the source program
.	.
.	Last word of the first section of the source program
.	Block size of the second section to load
.	XPC value of the destination address of the second section (7 bits)
.	Destination address (PC) of the second section (16 bits)
	First word of the second section of the sourceprogram
.	.
.	Last word of the second section of the source program
.	.
.	.
.	Block size of the last section to load
.	XPC value of the destination address of the last section (7 bits)
.	Destination address (PC) of the last section (16 bits)
.	First word of the last section of the source program
.	.
.	Last word of the last section of the sourceprogram
n	0000h- -indicates the end of source program

Table 3. General Structure of Source Program Data Stream in 8-Bit Mode

Byte	Contents
1	MSB = 08h, memory width of the source program (8 bits)
2	LSB = 0AAh
3	MSB of the value to set in the register
4	LSB of the value to set in the register
.	.
.	MSB of the value to set in the register
.	LSB of the value to set in the register
.	MSB of the XPC value of the entry point
.	LSB of the XPC value of the entry point (least significant 7 bits are used)
2R- 1	MSB of the entry point (PC)
2R	LSB of the entry point (PC)
2R+1	MSB of the block size of the first section to load
2R+2	LSB of the block size of the first section to load
2R+3	MSB of the XPC value of the destination address of the first section
2R+4	LSB of the XPC value of the destination address of the first section (7 bits)
2R+5	MSB of the destination address (PC) of the first section
2R+6	LSB of the destination address (PC) of the first section
.	MSB of the first word of the first section of the source program
.	.
.	LSB of the last word of the first section of the source program
.	MSB of the block size of the second section to load
.	LSB of the block size of the second section to load
.	MSB of the XPC value of the destination address of the second section
.	LSB of the XPC value of the destination address of the second section (7 bits)
.	MSB of the destination address (PC) of the second section
.	LSB of the destination address (PC) of the second section
.	MSB of the first word of the second section of the source program
.	.
.	LSB of the last word of the second section of the source program
.	.
.	MSB of the block size of the last section to load
.	LSB of the block size of the last section to load
.	MSB of the XPC value of the destination address of the last section
.	LSB of the XPC value of the destination address of the last section (7 bits)
.	MSB of the destination address (PC) of the last section
.	LSB of the destination address (PC) of the last section
.	MSB of the first word of the last section of the source program
.	.
.	LSB of the last word of the last section of the source program
2n	00h
2n+1	00h indicates the end of the source program

2.2 Boot Mode Options

The following sections discuss each of the bootloader modes and how they are selected and used.

2.2.1 HPI Boot Mode

The first boot mode checked after reset is HPI boot mode. After a reset, the bootloader initializes two data memory locations 007Eh and 007Fh to 0's and uses them as a software indication for when the host has completed the loading the application code through the HPI. It then asserts the host interrupt signal (HINT) low. The bootloader checks to see if the INT2 flag in the interrupt flag register (IFR) is equal to 1 (active), and if so, initiates HPI boot mode. The INT2 flag can be equal to 1 if:

- The HINT pin is tied to the INT2 input pin, or
- A valid interrupt to the INT2 input pin is generated within 30 CPU clock cycles after the DSP fetches the reset vector (if HINT is not tied to INT2)

If the INT2 flag is not set, indicating that the HINT pin is not tied to the INT2 pin or that INT2 is not asserted within 30 CPU clock cycles, the bootloader checks all boot modes including HPI mode. If the INT2 flag is set, the bootloader assumes HPI boot mode only and monitors the entry point addresses as explained below.

In DSP HPI boot mode, unlike some of the existing 54x HPI boot modes, the host must download the code to on-chip RAM after the DSP is brought out of reset. While the host is loading the on-chip RAM, the 'DSP checks location 00 007Fh in a continuous loop looking for a change in the contents of that address. Once the host has finished loading the boot code, it must perform two additional writes to data memory address 00 007Eh and 00 007Fh to set the entry point (start address) of the loaded code. Address 00007Eh contains the extended address (XPC) of the entry point (addresses A22**-** A16). Address 00 007Fh contains the lower 16-bits (PC) of the entry point (address A15**-** A0). **Note that the PC of the entry point of the loaded code must be a non-zero number**. When the host performs the write to 00 007Fh, the 'DSP detects changes at that address, and uses the new contents of the address to branch to the loaded code.

When HINT has been asserted low, it stays low. The host controller can clear HINT by writing to the host port interface control register (HPIC). The source program data stream for HPI mode can only contain the program itself. It cannot include any extra information, such as section size or register values, since the bootloader does not perform any interaction with the source program data stream in HPI boot mode. The bootloader simply branches to the start address specified above and begins execution.

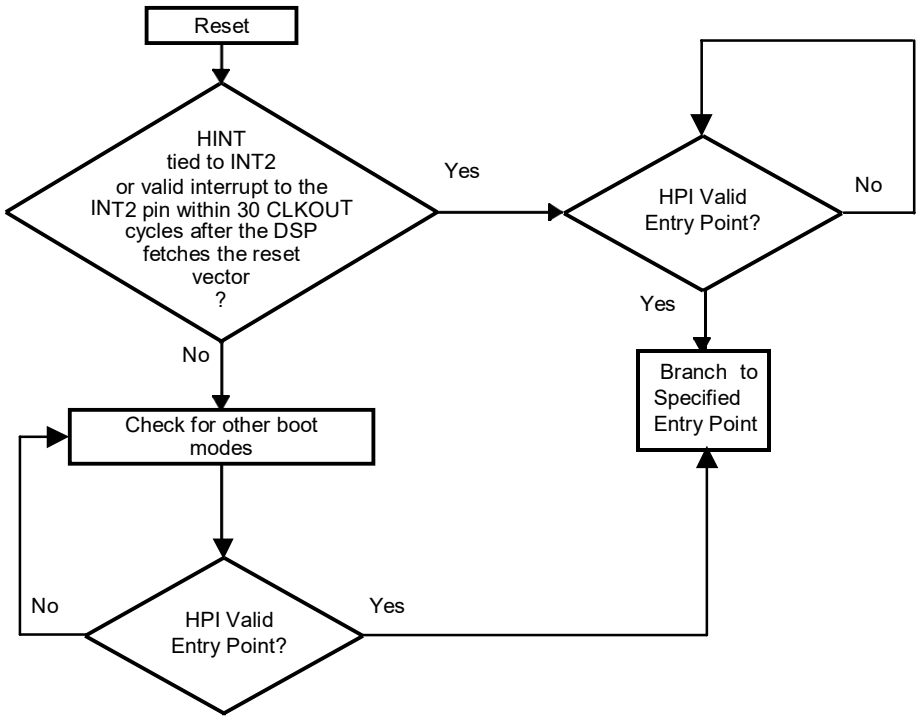

Figure 3. HPI Boot Mode Flow

2.2.2 Serial EEPROM Boot Mode

After verifying that HPI boot was not selected, the bootloader checks for the serial EEPROM boot mode.

The serial EEPROM boot mode is selected via the /INT3 external interrupt. The bootloader checks to see if the INT3 flag in the interrupt flag register (IFR) is equal to 1 (active), and if so, initiates serial EEPROM boot mode. Therefore, proper selection of the boot mode requires a high to low transition on the /INT3 pin within 30 CPU cycles after the DSP is reset. If an external event is not available in the system to trigger the interrupt, the McBSP2 transmit pin (BDX2) can be used instead. The BDX2 pin is automatically toggled from high to low within the first few cycles after reset, and can be externally connected to the /INT3 input pin to select the boot mode. When the DSP is reset, the BDX2 pin toggles from high to low causing the /INT3 flag to be set, and thereby selecting the serial EEPROM boot mode. This method of selecting the boot mode is convenient, because it doesn't require any additional external signals or components.

To summarize, the INT3 flag can be properly activated if:

- The BDX2 pin is tied to the INT3 input pin as shown in Figure 4, or
- A valid interrupt to the INT3 input pin is generated within 30 CPU clock cycles after the DSP fetches the reset vector (if BDX2 is not tied to INT3)

The serial EEPROM boot mode is intended for bootloading the DSP from an SPI based serial EEPROM. This mode configures McBSP2 in the clock-stop mode, with internal clocks and frames. The McBSP is then used to sequentially access the serial EEPROM. The EEPROM must have a four-wire SPI slave type interface with a 16-bit address. The interface between the McBSP and EEPROM is shown in Figure 4 below.

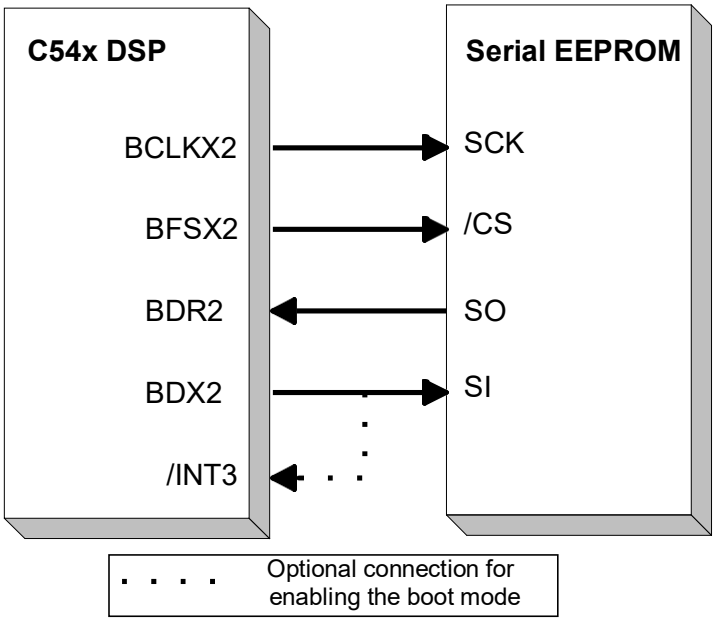

Figure 4. McBSP2 to EEPROM Interface for Serial EEPROM Boot Mode

The McBSP clock rate is set to fCLKOUT/250, or 400Khz for a 100Mhz device. This low bit rate ensures compatibility with most SPI based serial EEPROM devices. The McBSP is configured with CLKSTP=3, CLKXP=0, and CLKXM=1, for use as an SPI master. The relevant interface timings for this mode are given in the McBSP section of the DSP datasheet.

For each access, the McBSP transmits a 32-bit packet consisting of the 8-bit read instruction (03h), followed by the 16-bit address to be read from, followed by a “place-holder” byte. The EEPROM ignores the last 8-bits in the packet, and uses this slot to shift out the addressed byte on the SO output pin. An example read access is shown in Figure 5 below.

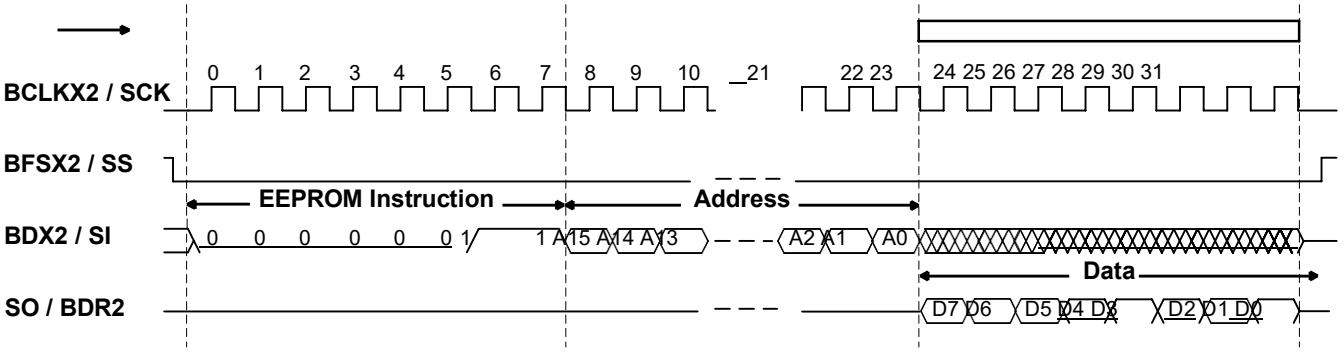

Figure 5. Example Read Access for Serial EEPROM Boot Mode

The bootloader starts reading from address 0 of the serial EEPROM and checks for a valid boot table. When the bootloader reads and validates the keyword from the beginning of the boot table, it proceeds to read the remainder of the boot table. If a valid keyword is not found, the bootloader will continue to try to read the keyword from address 0 until a valid keyword is read. The serial EEPROM boot mode uses the same boot table as the standard 8- bit serial boot mode. The bit stream for this boot table is described in section 2.2.4, Standard Serial Boot Mode.

The serial EEPROM boot mode can be used to load multiple sections into any valid program space address range – including extended program space. Since the serial EEPROM has a 16bit address, the maximum possible size of the boot table is 64K bytes, or 32K 16-bit words. This limits the amount of code that can be directly loaded using this boot mode.

After the serial EEPROM boot process completes, the XF signal is toggled low. If the EEPROM has an active low HOLD input, this event can be used to automatically disable the EEPROM after bootloading.

2.2.3 Parallel Boot Mode

After verifying that HPI boot was not selected, the bootloader checks for parallel boot mode. Parallel boot mode reads the desired boot table from data space via the external parallel interface (external memory interface) and transfers the code to program space. The bootloader supports parallel boot of 8- or 16-bit wide data. The software wait-state register (SWWSR) and bank-switch control register (BSCR) are reconfigurable in both of these boot modes. This allows the bootloader to boot from faster EPROM with fewer software wait states. The bootloader uses a default setting of seven wait states.

The bootloader gets the source address from either I/O port 0FFFFh or data memory address 0FFFFh. Since the source address read from either of these locations is 16 bits wide, the boot table can reside in any valid external address range of the upper 32K data space. Figure 6 shows the sequence of actions in the parallel boot mode.

NOTE: If a parallel boot is not desired, the data pin D0 of the DSP can be pulled high with a weak pullup resistor to avoid inadvertently booting from data or I/O space. Otherwise, some other method should be used to ensure that a valid keyword is not read unexpectedly.

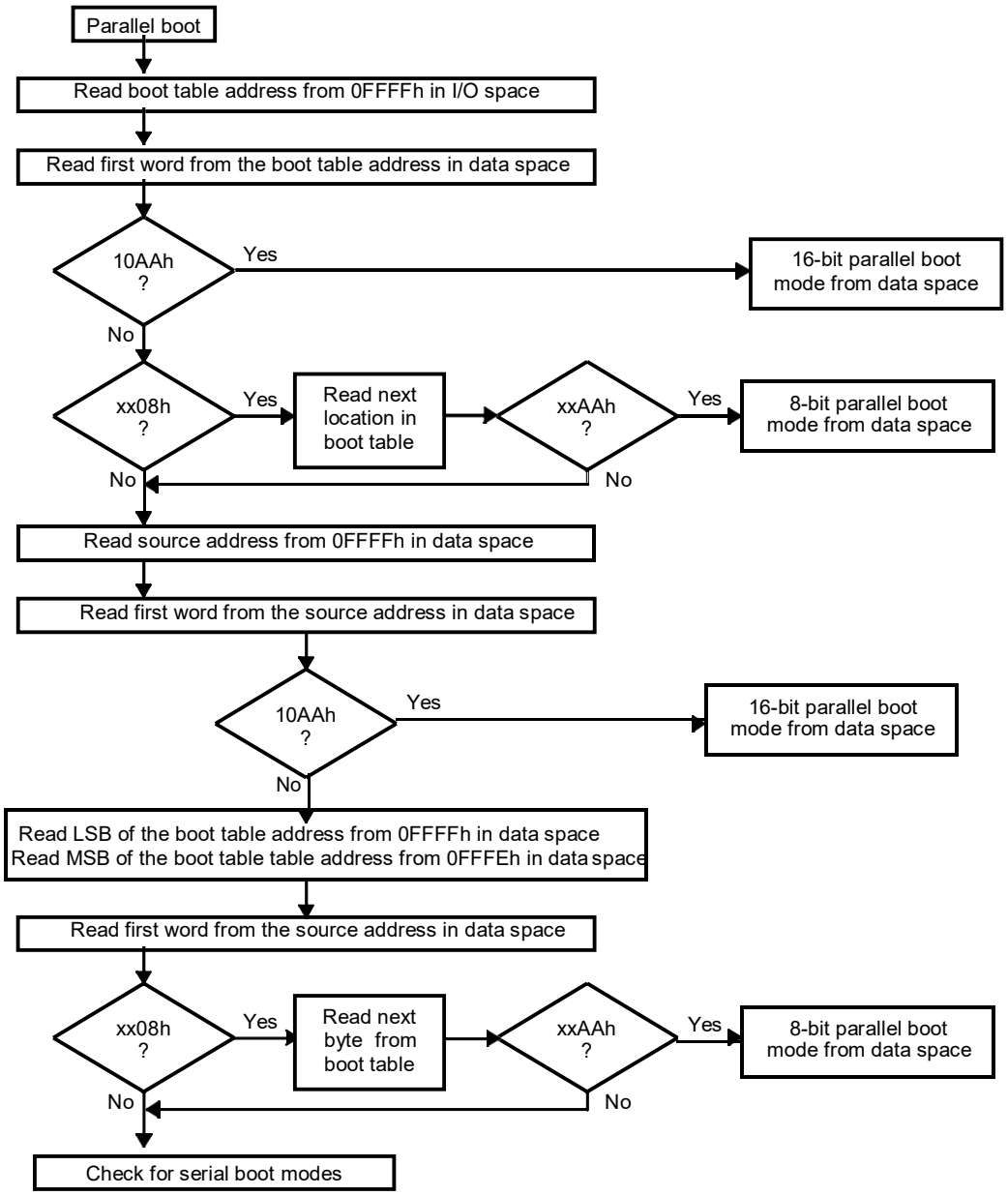

Figure 6. Parallel Boot Mode Process

Since the bootloader does not know the memory width before it reads the first word of the boot table, it must check both the data memory LSB (0FFFFh) and MSB (0FFFEh) to obtain the correct source address. Figure 7 shows the source program data stream for parallel boot mode using 8- or 16-bit-word mode.

After the bootloader reads and validates the keyword from the beginning of the boot table, the next two words are the desired values for the software wait-state register (SWWSR) and the bank switching control register (BSCR). These register values are loaded and become active before the remainder of the boot table is read so changes to these setting will affect the rest of the boot process.

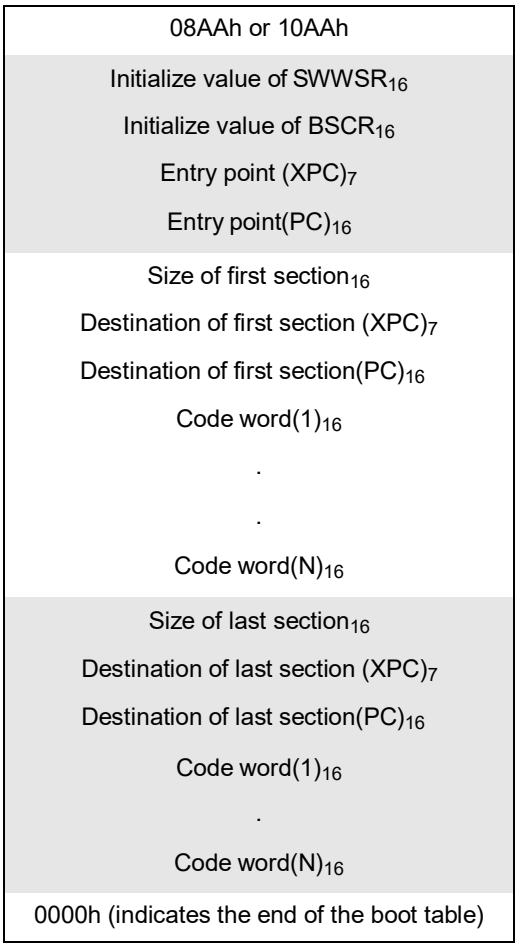

Figure 7. Source Program Data Stream for Parallel Boot in 8-,16-Bit-Word Mode

2.2.4 Standard Serial Boot Mode

The bootloader is capable of loading data in standard serial port mode from McBSP0 or McBSP2. McBSP0 is used to perform 16-bit transfers. McBSP2 is used to perform 8-bit transfers. Currently, on the DSP, boot through McBSP1 is reserved for a future serial boot mode.

In standard serial boot mode, the bootloader initializes the serial port to a configuration consistent with the TI C54x standard serial port. The bootloader also sets the XF pin low to indicate that the serial port is ready to receive data. The DSP then polls the IFR to determine which serial port has data input (BRINT0, or BRINT2). When the desired serial port has been identified, the bootloader continues to read the same port to load the entire boot table.

On the C548/549 bootloader, the bootloader would read configuration words from the boot table and reconfigure the serial port before completing the boot table. On the DSP, the serial port is not reconfigured at any point during the read of the boot table. To maintain compatibility with the hex utility, the DSP bootloader ignores the register configuration entries in the serial boot table.

The following conditions (illustrated in Figure 8) must be met in order to insure proper operation.

- The serial port receive clock (BCLKR) is supplied externally and cannot exceed ½ the frequency of the DSP CPU clock.
- A minimum delay time of 40 CPU clocks should be provided between the transmission of each word. This can be achieved by either slowing the receive clock frequency, or providing additional clocks between transmitted words.

These conditions provide the required delay for receiving consecutive words in the bootloader.

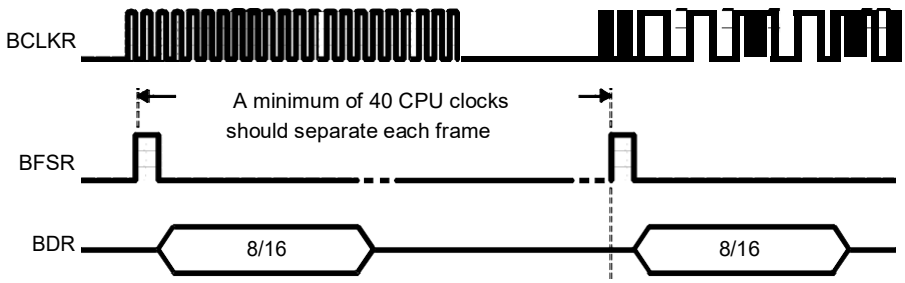

Figure 8. Timing Conditions for Serial Port Boot Operation

Figure 9 and Figure 10 show the serial boot process from each of the supported serial ports. The bootloader polls the serial port status registers and when the received data ready flag (RRDY) is active, the data receive register (BDRR) is read.

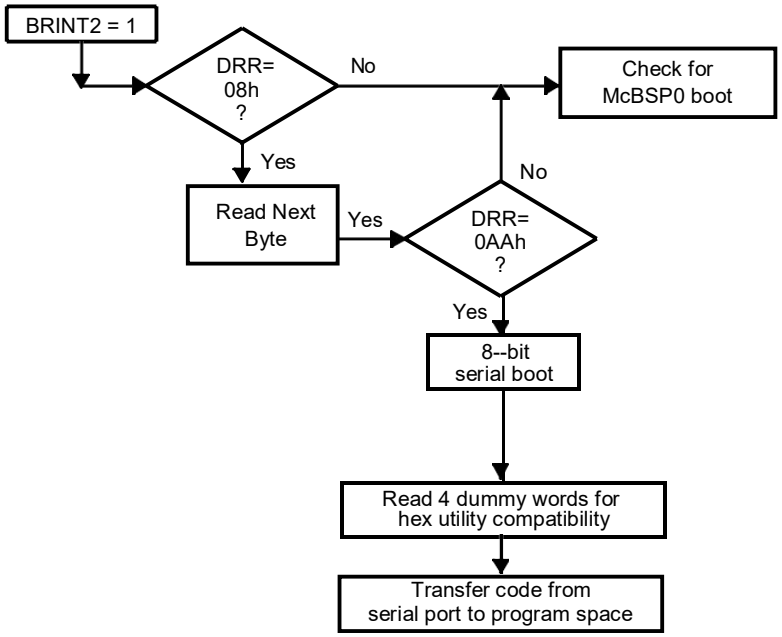

Figure 9. Standard Serial Boot From McBSP2 (8-bit) After BRINT2 = 1

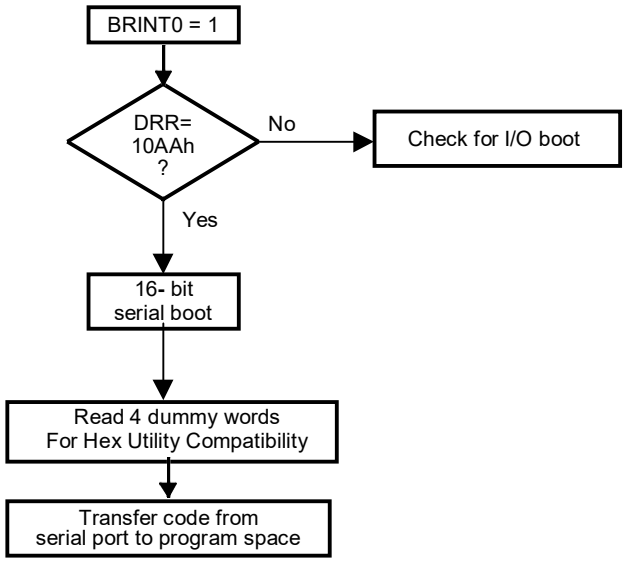

Figure 10. Standard Serial Boot From the McBSP0 (16-bit) After BRINT0 = 1

Figure 11 shows the boot table for serial boot modes as 16-bit words. For 8-bit mode, each word is transmitted to the serial port most significant byte first followed by least significant byte.

08AAh or 10AAh
Dummy Word for Compatibility - Ignored
Dummy Word for Compatibility - Ignored
Dummy Word for Compatibility - Ignored
Dummy Word for Compatibility - Ignored
Entry point $(XPC)_7$
Entry point$(PC)_{16}$
Size of first section$_{16}$
Destination of first section $(XPC)_7$
Destination of first section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
Size of second section$_{16}$
Destination of second section $(XPC)_7$
Destination of second section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
.
.
Size of last section$_{16}$
Destination of last section $(XPC)_7$
Destination of last section$(PC)_{16}$
Code word$(1)_{16}$
.
.
Code word$(N)_{16}$
0000h (indicates the end of the boot table)

Figure 11. Source Program Data Stream for 8/16-Bit McBSP Boot in Standard Mode

2.2.5 I/O Boot Mode

I/O boot mode provides the capability to bootload via the external parallel interface using I/O port 0h. This mode is in initiated when by the external host driving the BIO pin low. The DSP communicates with the external device using the BIO and XF as handshake signals. When BIO goes low, the DSP reads data from I/O address 0h, drives the XF pin high to indicate to the host that the data has been received, and writes the input data to the destination address. The DSP then waits for the BIO pin to be driven high and then low again by the external host for the next data transfer.

This mode asynchronously transfers code from I/O port 0h to internal or external program memory. This allows a slow host processor to easily communicate with the device by polling/ driving the XF/BIO lines. Words may be either 16 or 8 bits long.

Figure 12 shows the handshake protocol required to successfully transfer each word via I/O address 0h. The first handshake sequence shown in the figure requests and acknowledges the use of I/O boot mode. This sequence is performed after reset prior to sending any boot table contents. The second handshake sequence is used to load all of the words in the boot table. Upon reset of the end of the boot table, the DSP will transfer control to the entry point address specified in the boot table.

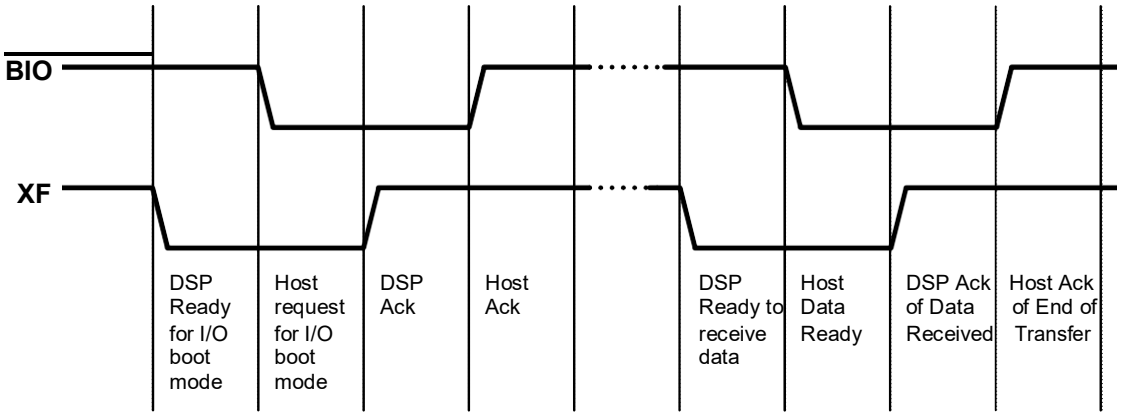

Figure 12. I/O Boot Mode Handshake Protocol

If the 8-bit transfer mode is selected, the lower eight data lines are read from I/O address 0h. The upper bytes on the data bus are ignored. The DSP reads two 8-bit words to form a 16-bit word. The order of transmission to the DSP is most significant byte first followed by least significant byte. Figure 13 shows the events in an I/O boot.

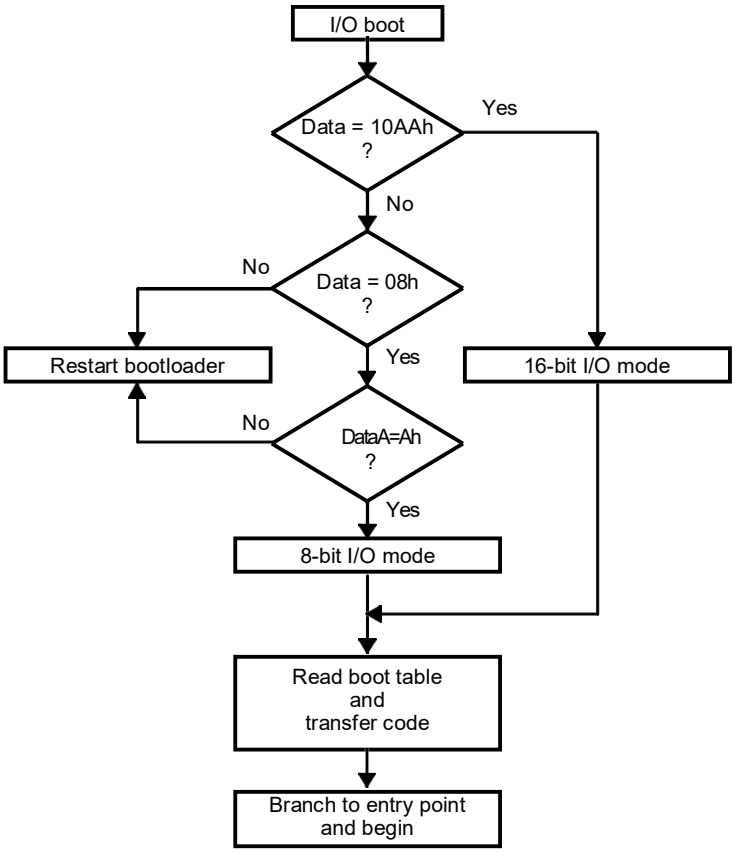

Figure 13. I/O Boot Mode

For both 8-bit and 16-bit I/O modes, the structure of the boot table is the same as that shown for the parallel boot modes in Figure 7.

A minimum delay of ten CPU clock cycles is provided between the XF rising edge and the write operation to the destination address. This allows the host processor time to turn off its data buffers before the DSP initiates a write operation (in case the destination is external memory). Note that the DSP only drives the external bus when XF is high.

3 Building the Boot Table

To use the features of the DSP bootloader, you must generate a boot table, which contains the complete data stream the bootloader needs. The boot table is generated by the hex conversion utility tool. The contents of the boot table vary, depending on the boot mode and the options selected when running the hex conversion utility.

NOTE: You must use version 1.20 or higher of the C54x code generation tools to generate the proper boot table for the DSP. Previous versions of the code generation tools do not support the enhanced bootloader options for the DSP and may produce a version of the boot table intended for earlier C54x devices without generating warnings or errors. Contact Texas Instruments for information about upgrades to earlier versions of the code generation tools.

To build the DSP boot table, follow these steps:

Step 1: **Assemble (or compile) the code using the - v548 assembler option.** This option marks the object files produced by the assembler specifically for the devices with enhanced bootloader functions including the DSP. The hex conversion utility uses this information to generate the correct format for the boot table. If this option is not included during assembly, the hex conversion utility may produce a version of the boot table intended for earlier ’C54x devices without generating warnings or errors.

Step 2: **Link the file.** Each block of the boot table data corresponds to an initialized section in the COFF file. Uninitialized sections are not converted by the hex conversion utility.

Step 3: **Run the hex conversion utility.** Choose the appropriate options for the desired boot mode and run the hex conversion utility to convert the COFF file produced by the linker into a boot table. See the *TMS320C54x Assembly Language Tools User’s Guide* (SPRU102) for a detailed description of the procedure for generating a boot table and using the options.

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

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