Application Report
SPRA627A - March 2002

TMS320VC5409 Bootloader Technical Reference

Philip M. Jones ll *C5000 Applications*

ABSTRACT

This document describes the features and operation of the TMS320VC5409 bootloader. The contents of the on-chip ROM also are discussed.

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 chapter describes the purpose and features of the TMS320VC5409 (hereafter referred to as ’C5409) digital signal processor (DSP). It also discusses the other contents of the device on-chip ROM and identifies where all of this information is located within that memory. Note that some revisions of silicon may exhibit exceptions to the functionality described in this document. Refer to *Silicon Updates For the TMS320VC5409 DSP (TI Literature number SPRZ156)* for a complete list of all known exceptions, and a description of how to determine the revision of a particular device.

1.1 Bootloader Features

The TMS320VC5409 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 ’C5409 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 Chapter 2 of this document.

1.2 On-Chip ROM Description

On the ’C5409 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)
- **Factory test code.** Reserved code used by TI for testing the device.
- **Interrupt vector table.** Code that allows indirect vectoring of interrupts

A listing of all of the ’C5409 ROM contents are available on the web at:

http://www.ti.com/sc/docs/tools/dsp/ftp/c54x.htm

The ’C5409 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 sampled low at reset or if the MP/MC status bit of the Processor Mode Status (PMST) Register is set to zero. At reset, the MP/MC bit of the PMST register is set to the value corresponding to the logic level on the MP/MC pin. Moreover, the level on the MP/MC pin is sampled at reset only, after which a change on this pin will have no effect until the next reset. However, you may alter the MP/MC bit field in the PMST register during any stage of the debugging process or in your application code in order to enable the on-chip ROM in upper program space.

Table 1. TMS320VC5409 On-chip ROM Program Space

Starting Address	Contents
000_C000 – 000_F7FF	Reserved for future enhancements
000_ F800 – 000_FBFF	Bootloader code
000_FC00 – 000_FEFF	Reserved for future enhancements
000_FF00 – 000_FF7F	Reserved for factory test code’
000_FF80 – 000_FFFF	Vector table

2 Bootloader Operation

2.1 Boot Mode Selection

The bootloader program located in the ’C5409 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-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 ’C5409 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 (8-bit and 16-bit supported):**

  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 host provides (i.e. changes) the start address of the downloaded code at data memory location 007Fh (on-chip scratch pad RAM).

- **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 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. McBSP0–1 supports 16-bit serial receive mode. McBSP2 supports 8-bit serial receive mode.

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

  The bootloader receives the boot table from a serial EEPROM connected to McBSP2 operating in clockstop (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 the XF and BIO pins. 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 ’C5409 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 Figure 1 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 first check depending on INT2
2. Serial EEPROM boot mode (8-bit)
3. Parallel boot mode
4. Standard serial boot mode via McBSP1 (16-bit)
5. Standard serial boot mode via McBSP2 (8-bit)
6. Standard serial boot mode via McBSP0 (16-bit)
7. I/O boot mode

A brief description of each mode is given in this section. Detailed description of the operation of each of these boot modes can be found in section 2.2, Boot Mode Options.

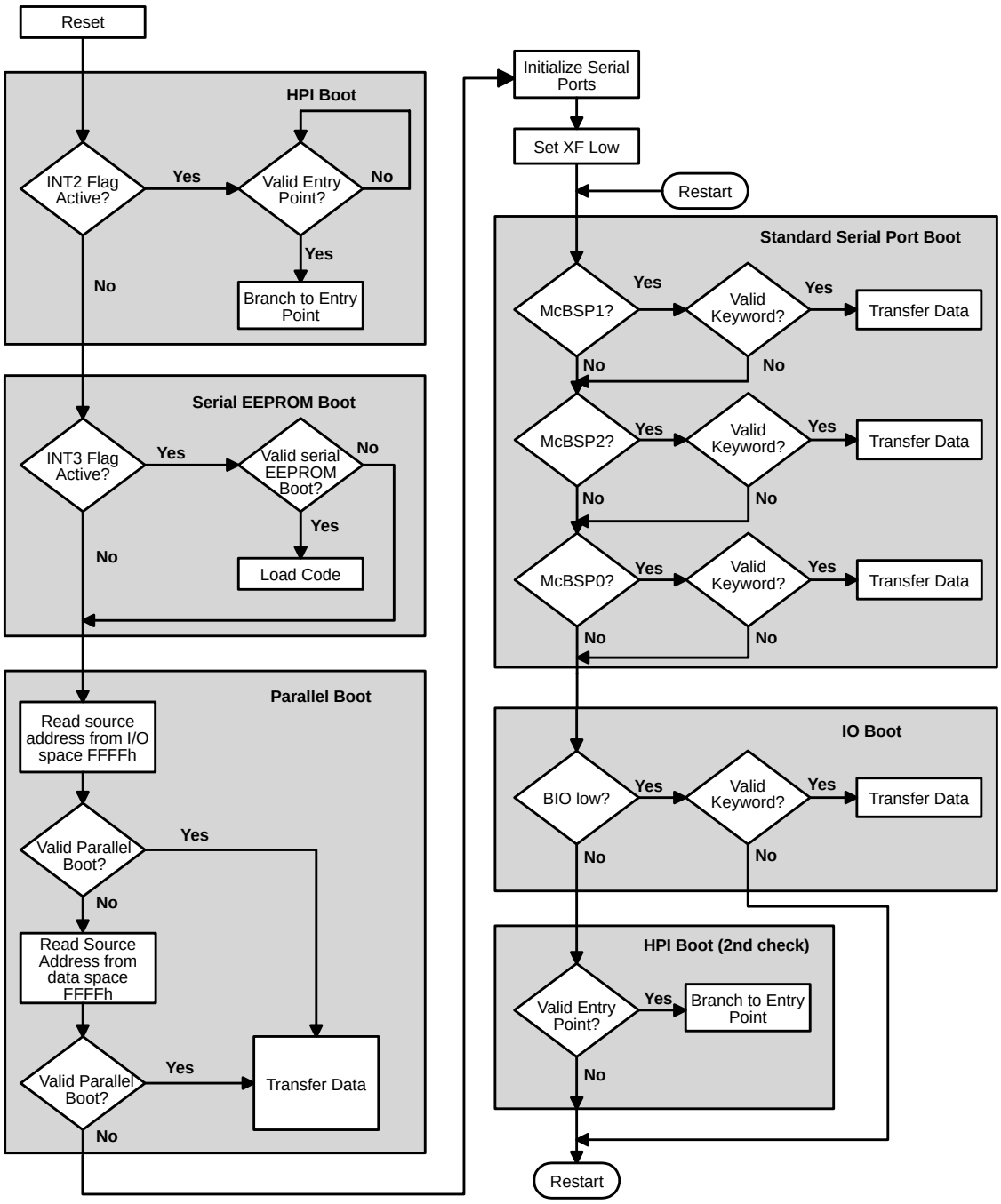

Figure 1. Bootloader Mode Selection Process

The first boot mode tested is the HPI boot mode. Unlike some other 54x HPI boot modes, the ’C5409 HPI boot mode allows the host to load the on-chip RAM after the ’C5409 is released from reset. The boot mode also allows the host to specify an entry point at load-time, to define where the ’C5409 begins execution of the loaded code. Immediately after reset, the bootloader initializes address 07Fh of the on-chip scratch-pad RAM to zero, and begins polling this location for a change in contents. While the ’C5409 is polling this location, the host can be loading the code into the on-chip RAM. After completing the bootload process, the host must make an additional HPI access to load the entry point of the code to location 07Fh. When the bootloader detects this change in the contents of address 07Fh, it performs a branch using the contents as the destination address. Note that because of this polling mechanism, the entry point must be a non-zero value.

Another unique feature of the ’C5409 HPI boot mode is that it 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 polling for HPI boot described above, 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 and enters a smaller loop that only performs the polling described above. 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, or no valid keyword is found, the bootloader proceeds to check the next bootmode.

The next boot modes tested are the parallel boot modes. The boot table is read from data space and can be located at data space addresses between 8000h and FFFFh. The bootloader determines the start address of the boot table by first reading address FFFFh in I/O space. The bootloader uses the value read from I/O space as the start address of the boot table in data space. The beginning of the boot table should contain a certain keyword that indicates whether 8-bit or 16-bit boot mode is desired. The valid keyword for 8-bit boot mode is 08AAh (in two consecutive 8 bit locations), while the valid keyword for 16-bit boot mode is 10AAh. If no valid keyword is found using the address read from I/O space, the process is repeated using the contents of address FFFFh of data space. The bootloader reads address FFFFh in data space to obtain a start address for the boot table. It then reads the contents of that start address to check for a valid keyword. It is convenient to provide both, the start address and the boot table, in data space since this allows storage of both in a single non-volatile memory. When a valid keyword is detected at the start address, 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 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, McBSP1 or McBSP2. When a word is received on one of the McBSPs, a serial port interrupt flag will be generated in the ’C5409. The boot loader uses these interrupt flags as indication of which McBSP to use for boot load. The ’C5409 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 ’C5409 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 ’C5409 and the host proceeds to load the remainder of the boot table using a handshaking scheme as described in section 2.2.4, I/O Boot Mode.

If a valid boot mode is not found after checking all of the possible boot modes**,** the bootloader restarts and continues to recheck the boot modes, starting with the standard serial port boot modes. Note that not all boot modes are rechecked.

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.

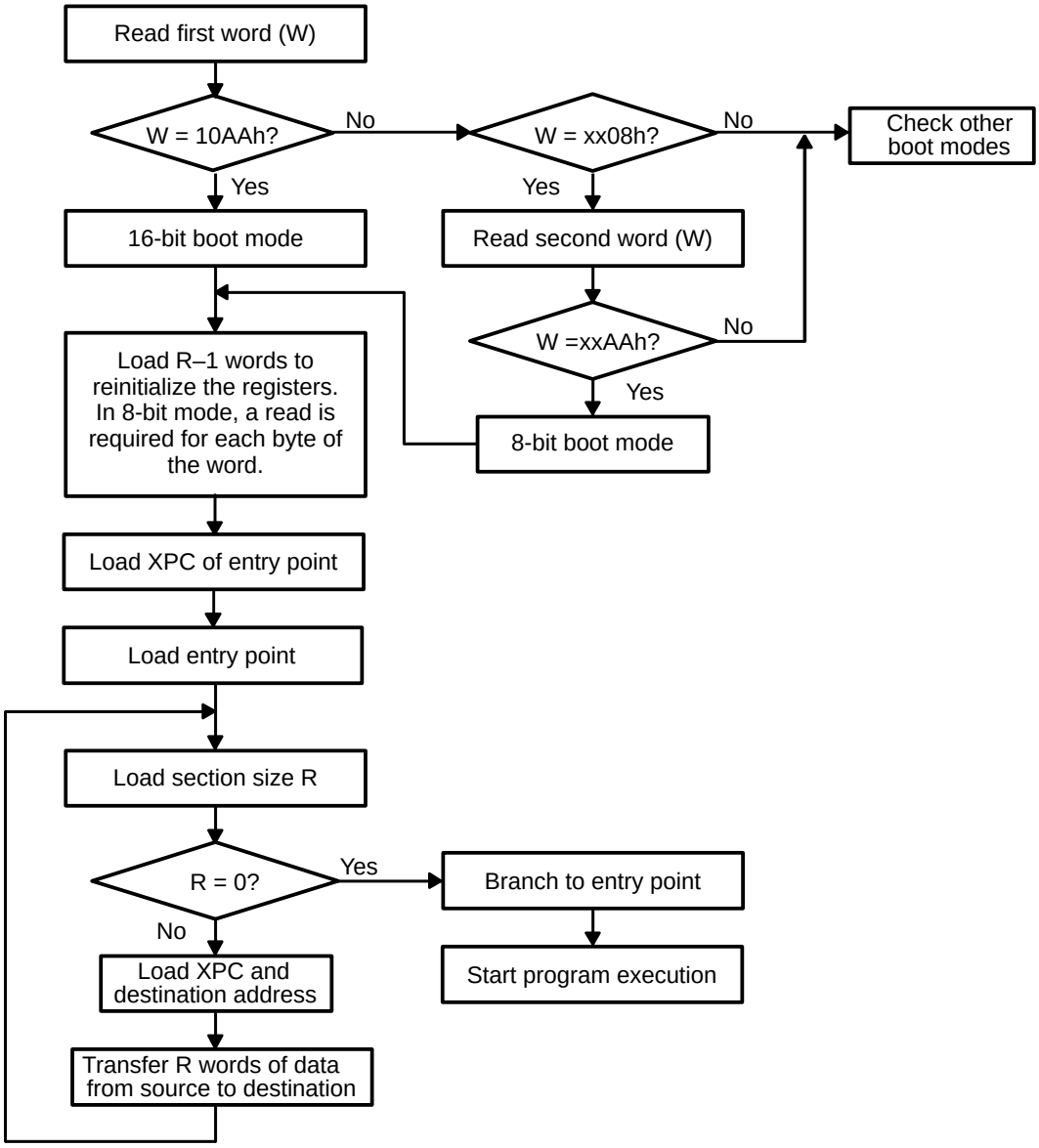

Figure 2. Basic Data-Width Selection and Data Transfer Process

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 source program
.	.
.	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 source program
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
.	.

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

Byte	Contents
.	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 data memory address 007Fh to 0 and uses it as a software indication for when the host has completed loading the application code through the HPI. It then asserts the host interrupt output 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 will only be equal to one, if an external user interrupt /INT2 has been detected. This can be accomplished by:

- Tying the HINT output pin directly to the /INT2 input pin, or
- Generating a valid external interrupt on the /INT2 input pin 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 address as explained below.

In ’C5409 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 ’C5409 checks location 007Fh of the on-chip scratch-pad RAM 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 an additional write to data memory address 007Fh to set the entry point (start address) of the loaded code. Address 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 007Fh, the ’C5409 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 assertion of the HINT pin can also be used to notify the host that the ’C5409 is out of reset, and ready to be loaded. The host controller can then 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.

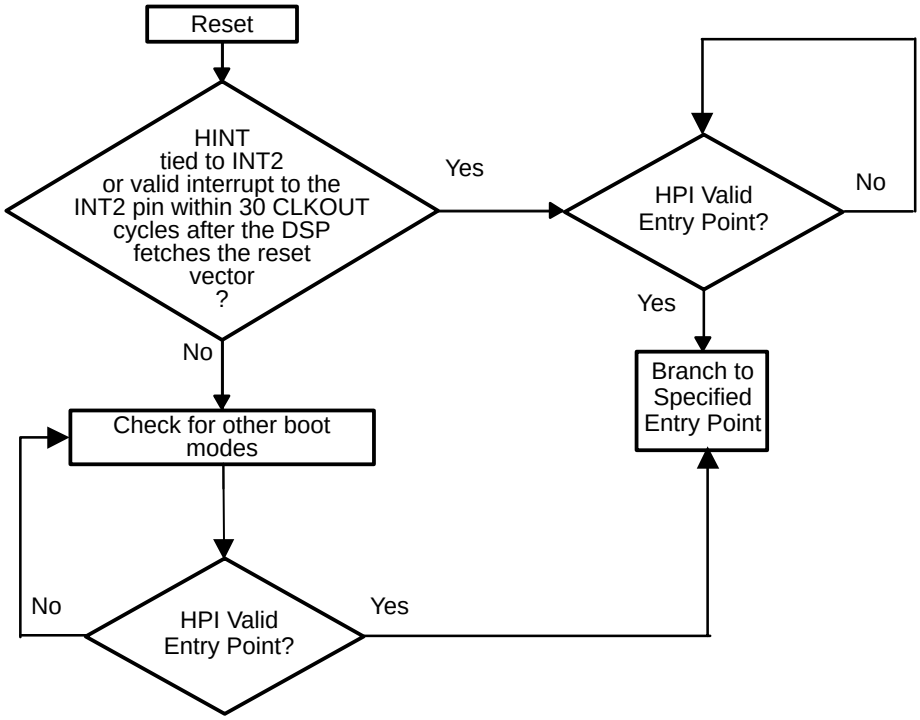

Figure 3. HPI Boot Mode Flow

2.2.2 Serial EEPROM Boot Mode

After verifying that HPI boot was not selected, the bootloader checks for the serial EEPROM boot mode. Please notice that some revisions of the ’C5409 device do not fully support this mode. Refer to *Silicon Updates For the TMS320VC5409 DSP (TI Literature number SPRZ156)* for a complete list of all known exceptions, and a description of how to determine the revision of a particular device.

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 ’C5409 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 ’C5409 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 ’C5409 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.

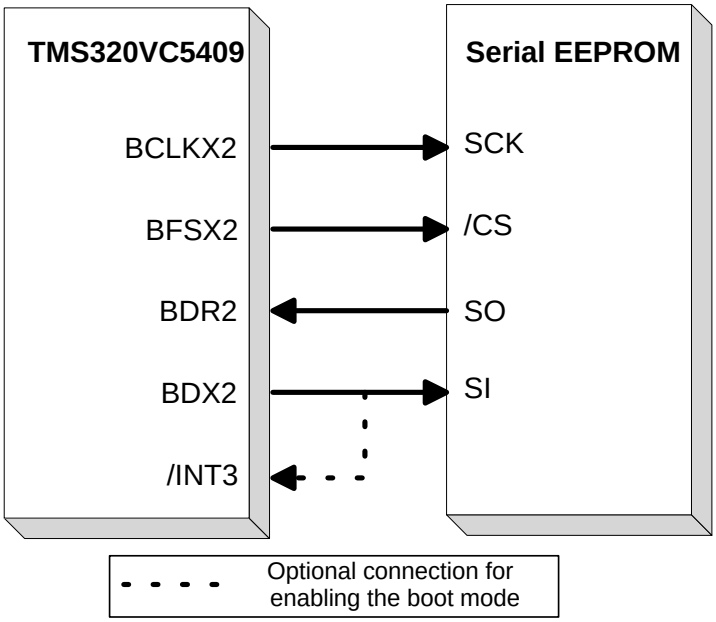

Figure 4. McBSP2 to EEPROM Interface for Serial EEPROM Boot Mode

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 ’C5409 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.

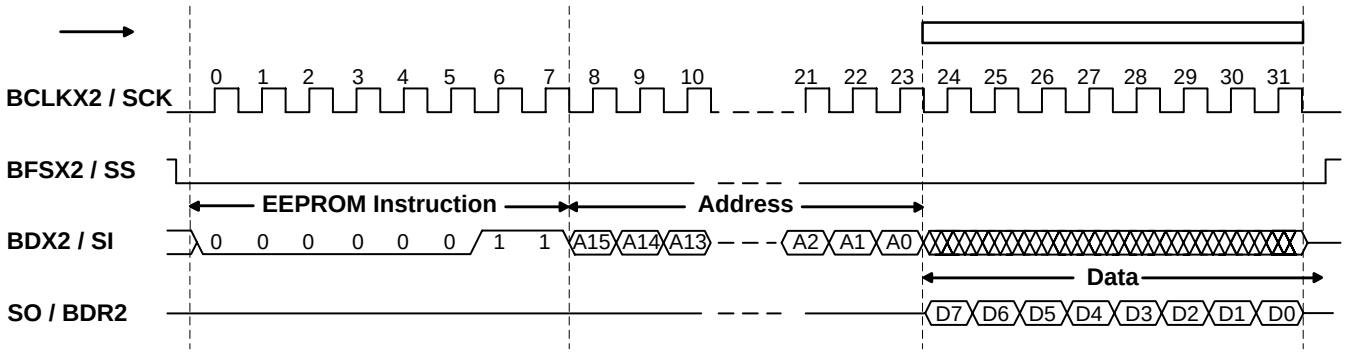

Figure 5. Example Read Access for Serial EEPROM Boot Mode

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. 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. Note that if no valid keyword is found after the first two reads, the boot process is aborted and the bootloader proceeds to check the next boot mode.

2.2.3 Parallel Boot Mode

After verifying that serial EEPROM 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. Although either option is valid, it is usually more convenient to provide the source address at the location in data memory, because then a single nonvolatile memory can contain both the boot table and the entry point.

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 within the data space of the ’C5409 memory map. The valid addresses for external data memory on the ’C5409 are between 04000h and 0FFFFh. 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 ’C5409 can be pulled high with a weak pullup resistor to avoid inadvertently booting from data or I/O space. This will prevent the bootloader from reading the keyword AAh from data space. Otherwise, some other method should be used to ensure that a valid keyword in not read unexpectedly.

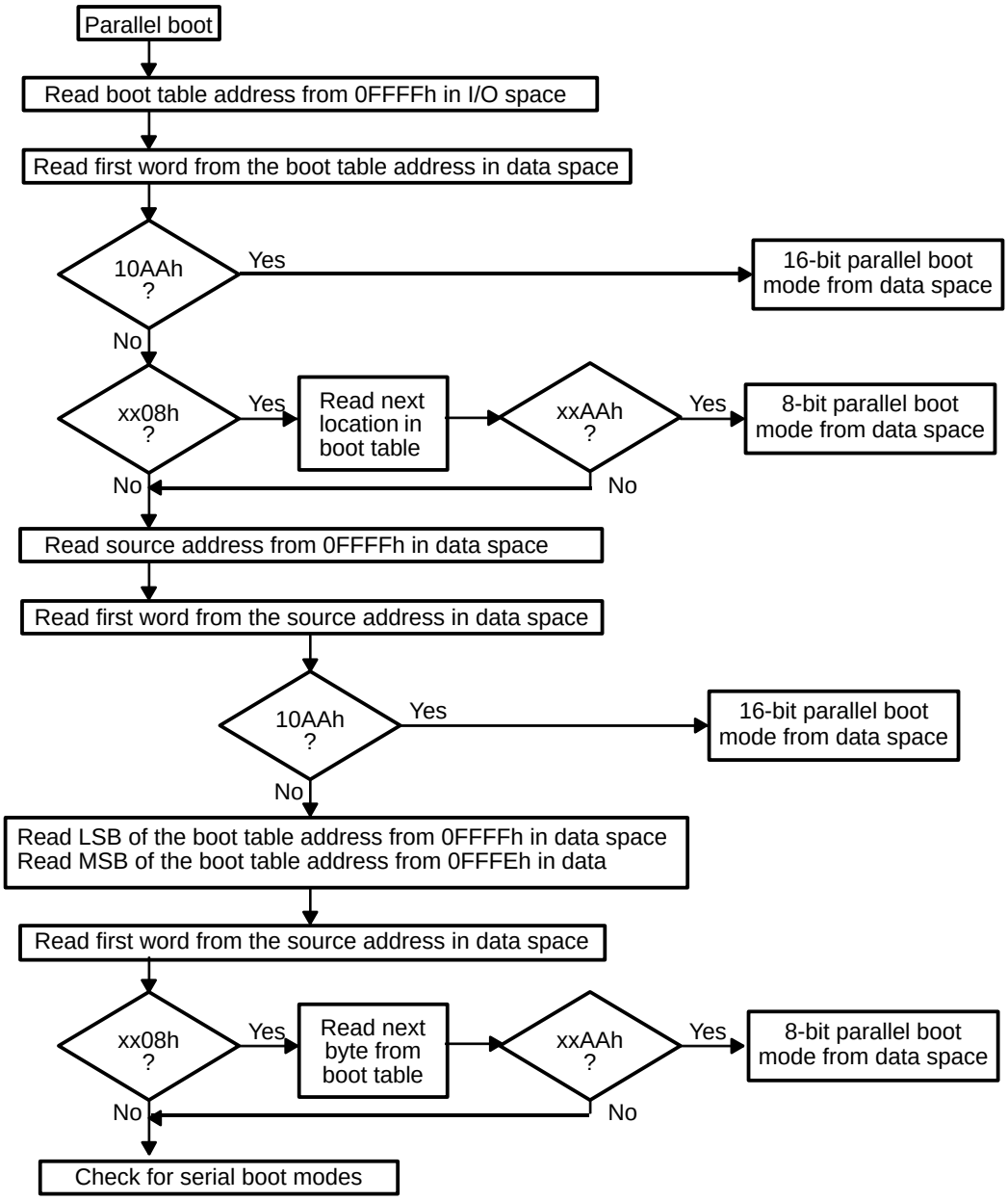

Figure 6. Parallel Boot Mode Process

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 settings will affect the rest of the boot process.

08AAh or 10AAh
Initialize value of $SWWSR_{16}$
Initialize value of $BSCR_{16}$
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 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 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, McBSP1 or McBSP2. McBSP0 and McBSP1 are used to perform 16-bit transfers. McBSP2 is used to perform 8-bit transfers.

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, BRINT1 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 'C5409, the serial port is not reconfigured at any point during the read of the boot table. To maintain compatibility with the hex conversion utility, the 'C5409 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 'C5409 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.

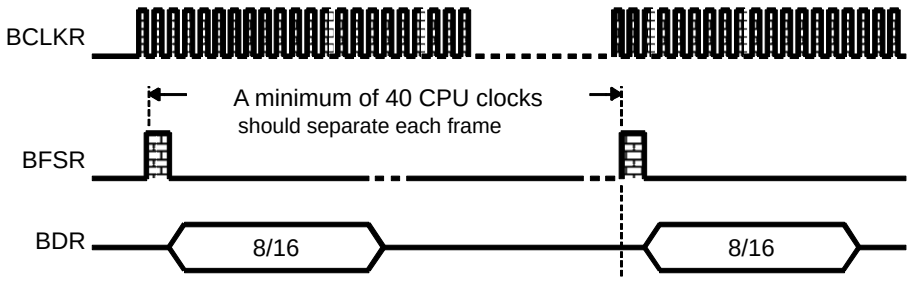

Figure 8. Timing Conditions for Serial Port Boot Operation

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.

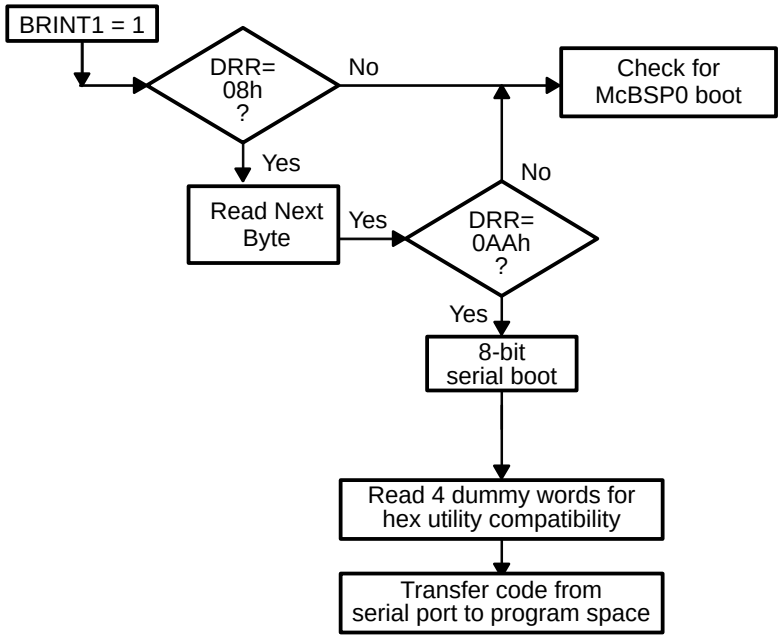

Figure 9. Standard Serial Boot From McBSP2 (8-bit) After BRINT2 = 1

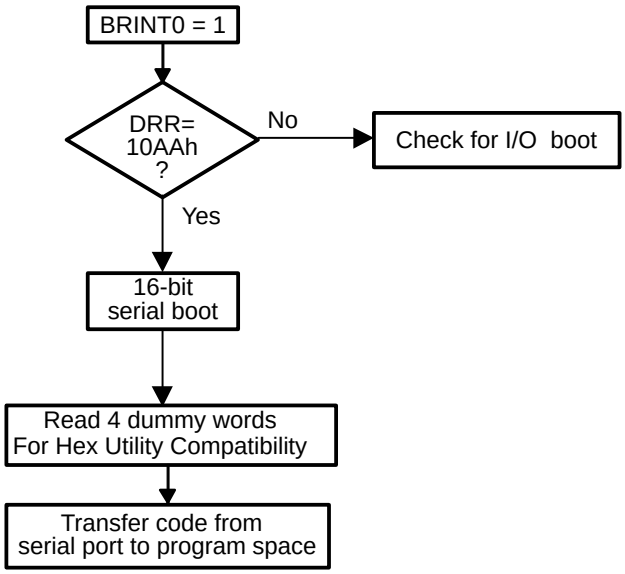

Figure 10. Standard Serial Boot From McBSP0-1 (16-bit) After BRINT0 or BRINT1 = 1

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 by the external host driving the BIO pin low. The ’C5409 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 ’C5409 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 reaching the end of the boot table, the DSP transfers control to the entry point address specified in the boot table.

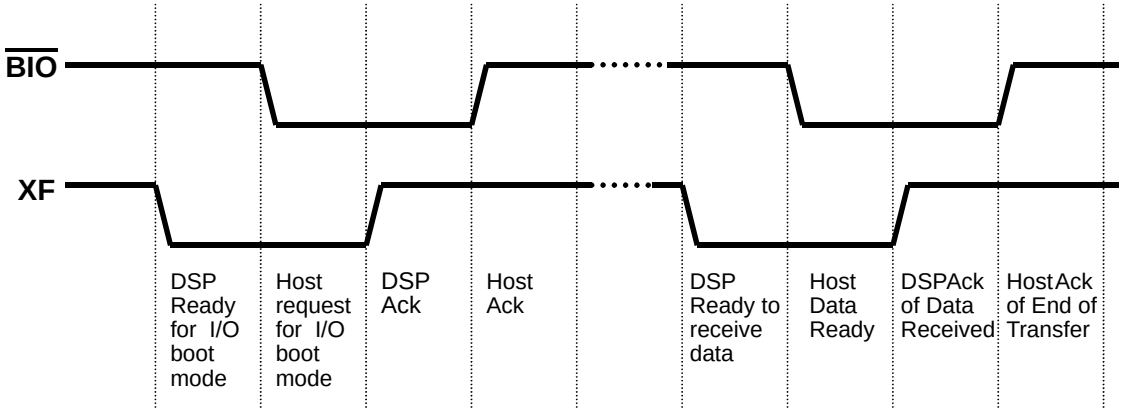

Figure 12. I/O Boot Mode Handshake Protocol

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 ’C5409 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.

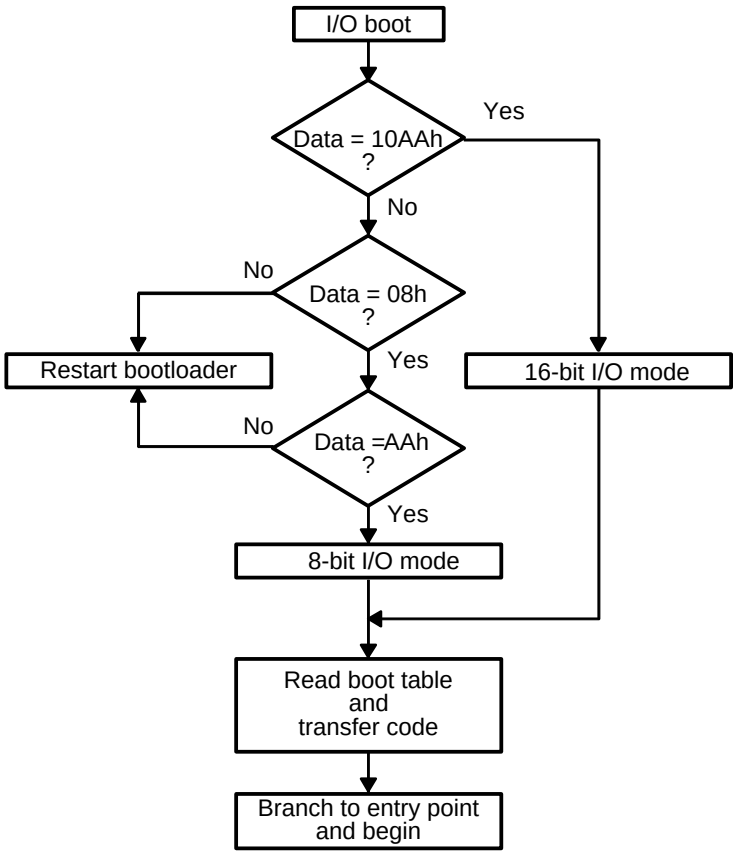

Figure 13. I/O Boot Mode

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 ’C5409 initiates a write operation (in case the destination is external memory). Note that the ’C5409 only drives the external bus when XF is high.

3 Building the Boot Table

To use the features of the ’C5409 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, which is provided with the TMS320C54x Assembly Language Tools Package. 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 ’C5409. Previous versions of the code generation tools do not support the enhanced bootloader options for the ’C5409 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 ’C5409 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 ’C5409. 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. Initialized sections include: .text, .const, and .cinit. Uninitialized sections are for example .bss, .stack, and .sysmem. These will not be converted by the hex conversion utility. It is important to note that sections should not be linked into address ranges where no RAM is actually present in the system. For example, no sections should be linked in address ranges 0F000h–0FFFFh, because this range of program space is occupied by the on–chip ROM, and cannot be written to by the bootloader.

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* for a detailed description of the procedure for generating a boot table and using the options.

An example command file for the Hex Conversion Utility is shown in Figure 14.

```
myfile.out          /* Input COFF file name.
–e 0300h            /* Entry point symbol.
–a                  /* ASCII hex output format.
–boot               /* Bootload all sections in the input file.
–bootorg SERIAL     /* Create a serial port boot table.
–memwidth 8         /* EEPROM width is 8 bits.
–o myfile.hex       /* Output file name.
```

Figure 14. Example Command File for Hex Conversion Utility

When the Hex Conversion Utility is invoked with the example command file of Figure 14, it creates an ASCII hex file called myfile.hex, which can be used for programming a serial EEPROM. All of the sections from the input file are placed in the boot table, and the entry point is set to address 0300h.

Additional guidelines can be found in *Bootloading the TMS320C548 Using the BSP in Standard Mode (TI Literature Number SPRA571),* which is available from application report section of the TI web site, URL:http://www.ti.com/. This document describes how to create a boot table and bootload a TMS320C548 digital signal processor (DSP) using another TMS320C54x DSP. The process of building a boot table for the ’C5409 is similar to that for the ’C548, however the configuration of the ’C5409 McBSPs cannot be changed during the boot load as can be done on the ’C548/549 devices.

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265