Application Report
SPRA609B - March 2002

TMS320VC5410 Bootloader

Clay Turner

C5000 Applications

ABSTRACT

This document describes the features and operation of the TMS320VC5410 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

1 Introduction . . . 2
1.1 Bootloader Features . . . 2
1.2 On-Chip ROM Description . . . 2
2 Bootloader Operation . . . 4
2.1 Boot Mode Selection . . . 4
2.2 Boot Mode Options . . . 10
2.2.1 HPI Boot Mode . . . 10
2.2.2 Parallel Boot Mode . . . 10
2.2.3 Standard Serial Boot Mode . . . 13
2.2.4 I/O Boot Mode . . . 16
3 Building the Boot Table . . . 18

List of Figures

Figure 1. Bootloader Mode Selection Process . . . 6
Figure 2. Basic Data-Width Selection and Data Transfer Process . . . 7
Figure 3. HPI Boot Mode Flow . . . 10
Figure 4. Parallel Boot Mode Process . . . 11
Figure 5. Source Program Data Stream for Parallel Boot in 8-,16-Bit-Word Mode . . . 12
Figure 6. Timing Conditions for Serial Port Boot Operation . . . 13
Figure 7. Standard Serial Boot From McBSP2 (8-bit) After BRINT2 = 1 . . . 14
Figure 8. Standard Serial Boot From the McBSP0 (16-bit) After BRINT0 = 1 . . . 14
Figure 9. Source Program Data Stream for 8/16-Bit McBSP Boot in Standard Mode . . . 15
Figure 10. I/O Boot Mode Handshake Protocol . . . 16
Figure 11. I/O Boot Mode . . . 17

List of Tables

Table 1. TMS320VC5410 On-chip ROM Program Space . . . 3
Table 2. General Structure of Source Program Data Stream in 16-Bit Mode . . . 8
Table 3. General Structure of Source Program Data Stream in 8-Bit Mode . . . 9

1 Introduction

This section describes the purpose and features of the TMS320VC5410 (hereafter referred to as 5410) 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 TMS320VC5410 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 5410 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 5410 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.
- **Data ROM tables.** Tables used for the global system for mobile communication enhanced full rate (GSM EFR) speech codec.
- **Fast Fourier transforms (FFTs).** Computationally efficient algorithms that convert information from the time domain to the frequency domain.
  - 256-point complex radix-2 decimation-in-time (DIT) FFT with looped code
  - 256-point complex radix-2 FFT twiddle factors
  - 1024-point complex radix-2 decimation-in-time (DIT) FFT with looped code
  - 1024-point complex radix-2 FFT twiddle factors

A listing of all of the 5410 ROM contents is available on the web at the following URL:

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

The 5410 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. TMS320VC5410 On-chip ROM Program Space

Starting Address	Contents
000_C000	ROM tables for the GSM EFR speech codec
000_D500	1024-point complex radix-2 DIT FFT with looped code
000_D700	1024-point radix-2 FFT twiddle factors
000_DD00	256-point complex radix-2 DIT FFT with looped code
000_DF00	256-point radix-2 FFT twiddle factors
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	Interrupt vector table

2 Bootloader Operation

2.1 Boot Mode Selection

The bootloader program located in the 5410 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 5410 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 during reset. Code execution begins after reset.

- **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.

- **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 5410 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
2. Parallel boot mode
3. Standard serial boot mode via McBSP2 (8-bit)
4. Standard serial boot mode via McBSP0 (16-bit)
5. 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. If INT2 is found to be active, the bootloader assumes that the code has been loaded into on-chip RAM by an external host, and branches to 000_2000h which is the default execution start address for HPI boot mode. HPI mode employs a default execution start address because the code is loaded directly without a boot table and therefore no other specific execution start address is available.

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 bootloader uses the value read from I/O space as the address of the boot table in data 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 5410. The boot loader uses these interrupt flags as indication of which McBSP to use for boot load. The 5410 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 5410 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 5410 and the host proceed 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 each of the boot modes in the same sequence.

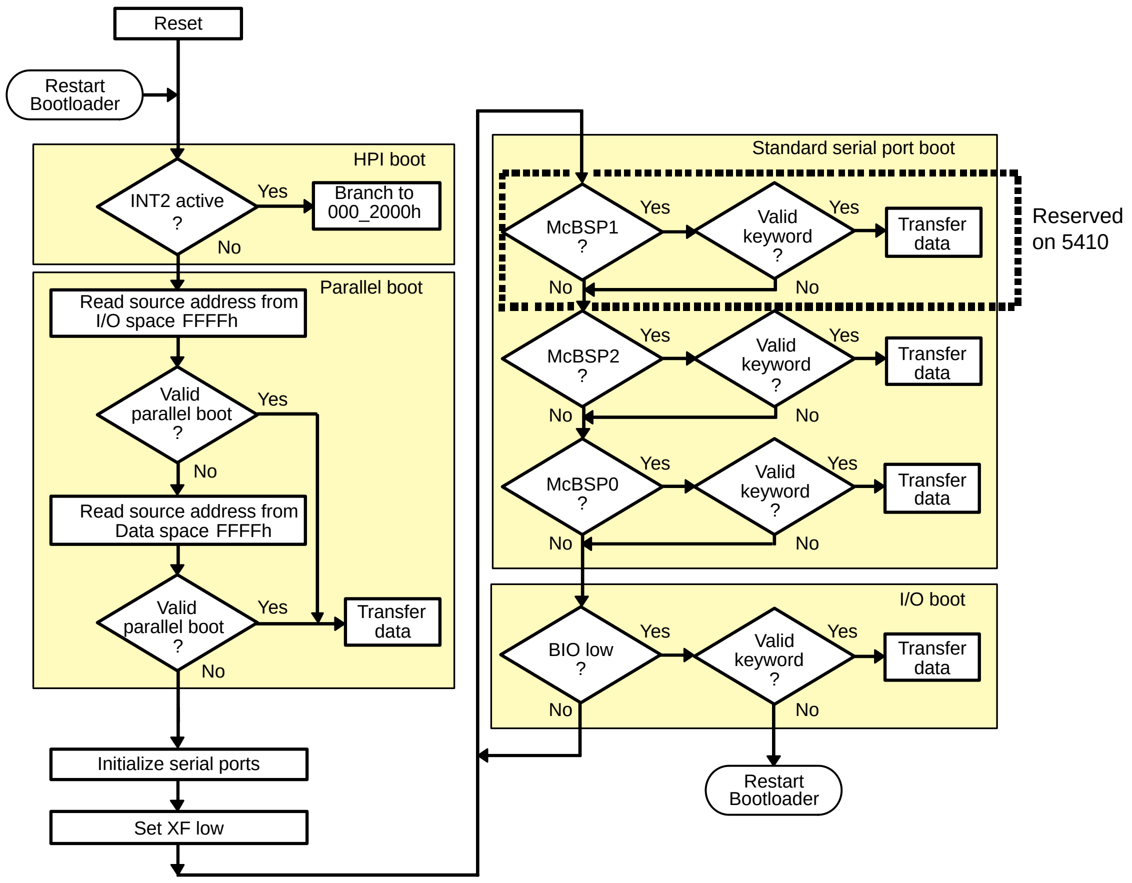

Figure 1. Bootloader Mode Selection Process

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.

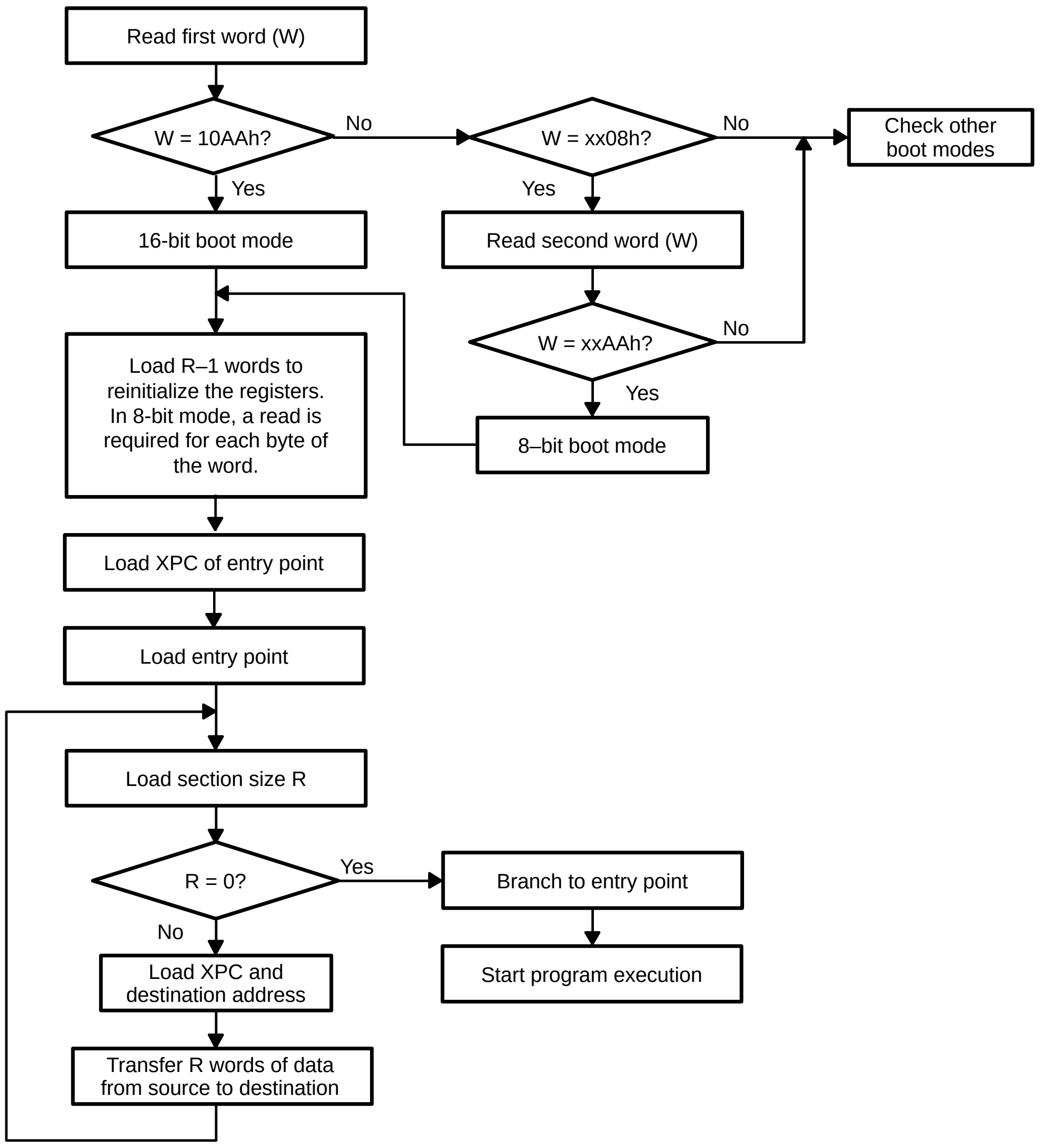

Figure 2. Basic Data-Width Selection and Data Transfer Process

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 key-word, 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
.	.
.	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 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 boot routine skips HPI mode and performs further tests to determine the selected boot mode.

In HPI boot mode, the host must download the code to on-chip RAM before the DSP is brought out of reset. The bootloader transfers control to a start address in on-chip RAM (at 000_2000h in program space) and begins executing code from there. 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.

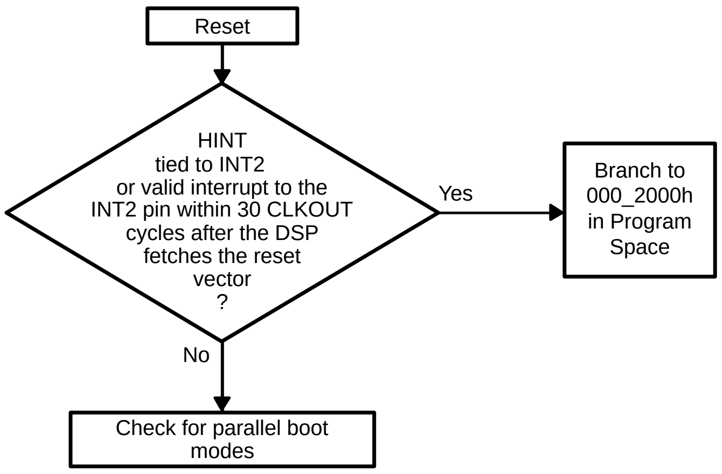
Reset
HINT tied to INT2 or valid interrupt to the INT2 pin within 30 CLKOUT cycles after the DSP fetches the reset vector ?
Yes
Branch to 000_2000h in Program Space
No
Check for parallel boot modes

Figure 3. HPI Boot Mode Flow

2.2.2 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 4 shows the sequence of actions in the parallel boot mode.

NOTE: If a parallel boot is not desired, the data pin D0 of the 5410 can be pulled-high with a weak pull-up resistor to avoid inadvertently booting from data or I/O space. Otherwise, some other method should be used to ensure that a valid keyword in not read unexpectedly.

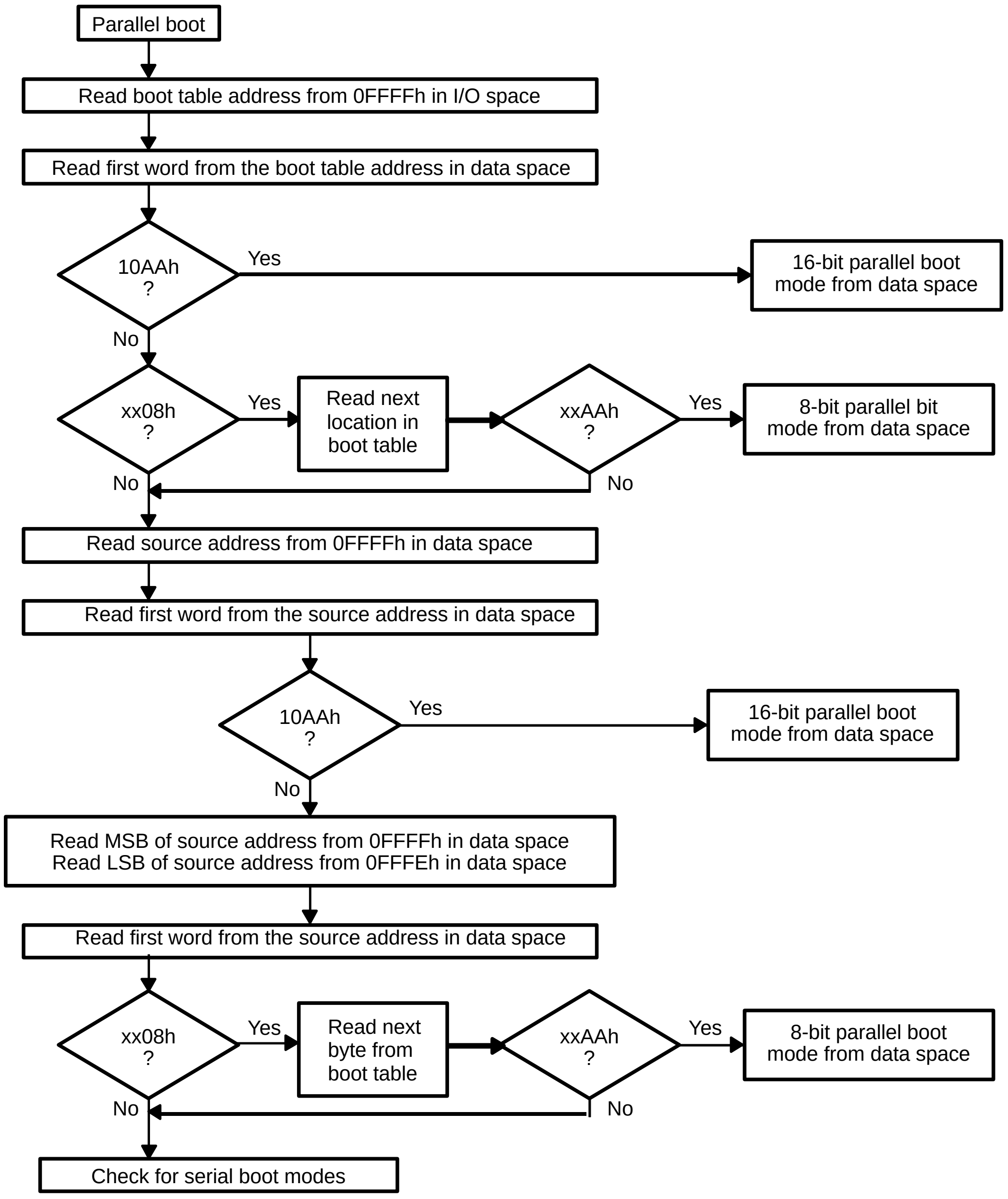
Parallel boot
Read boot table address from 0FFFFh in I/O space
Read first word from the boot table address in data space
10AAh ?
Yes
16-bit parallel boot mode from data space
No
xx08h ?
Yes
Read next location in boot table
xxAAh ?
Yes
8-bit parallel bit mode from data space
No
No
Read source address from 0FFFFh in data space
Read first word from the source address in data space
10AAh ?
Yes
16-bit parallel boot mode from data space
No
Read MSB of source address from 0FFFFh in data space
Read LSB of source address from 0FFFEh in data space
Read first word from the source address in data space
xx08h ?
Yes
Read next byte from boot table
xxAAh ?
Yes
8-bit parallel boot mode from data space
No
No
Check for serial boot modes

Figure 4. 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 5 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 5. Source Program Data Stream for Parallel Boot in 8-,16-Bit-Word Mode

2.2.3 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 5410, 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 5410, 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 5410 bootloader ignores the register configuration entries in the serial boot table.

The following conditions (illustrated in Figure 6) 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 5410 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.

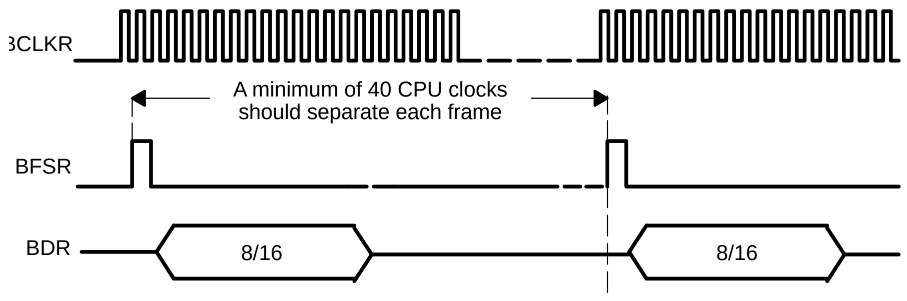

Figure 6. Timing Conditions for Serial Port Boot Operation

Figure 7 and Figure 8 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.

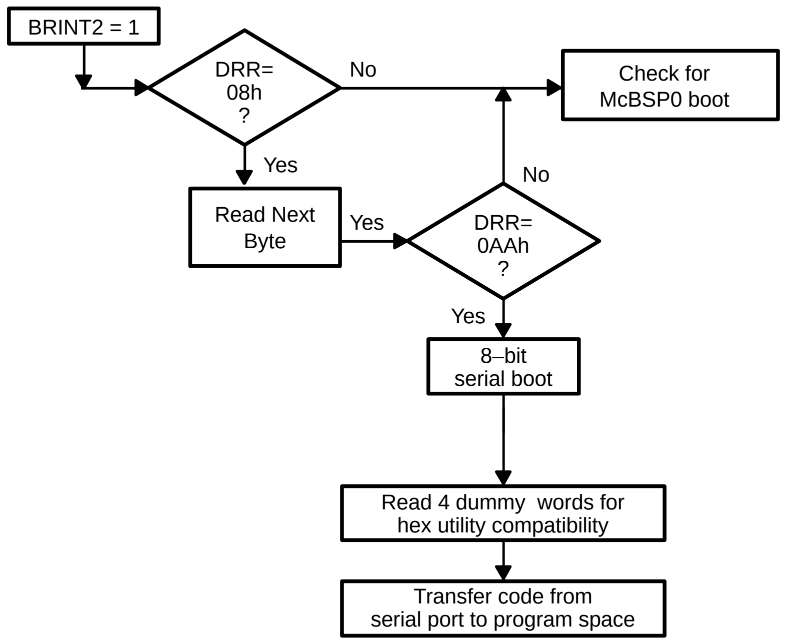

Figure 7. Standard Serial Boot From McBSP2 (8-bit) After BRINT2 = 1

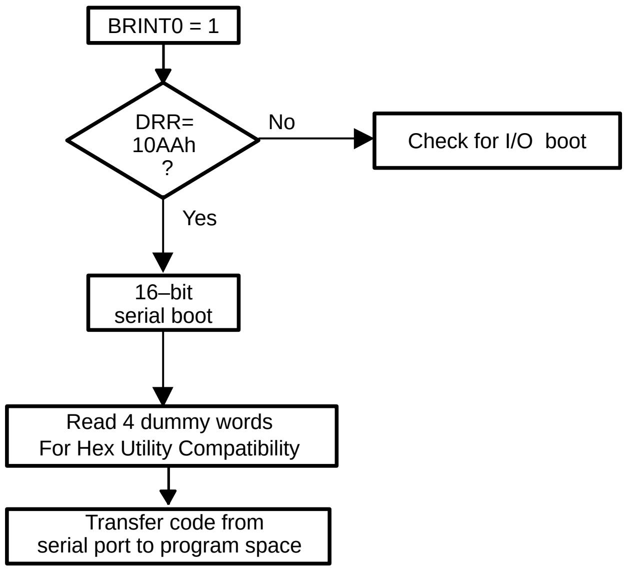
BRINT0 = 1
DRR= 10AAh ?
No
Check for I/O boot
Yes
16–bit serial boot
Read 4 dummy words For Hex Utility Compatibility
Transfer code from serial port to program space

Figure 8. Standard Serial Boot From the McBSP0 (16-bit) After BRINT0 = 1

Figure 9 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 9. Source Program Data Stream for 8/16-Bit McBSP Boot in Standard Mode

2.2.4 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 5410 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 5410 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 10 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.

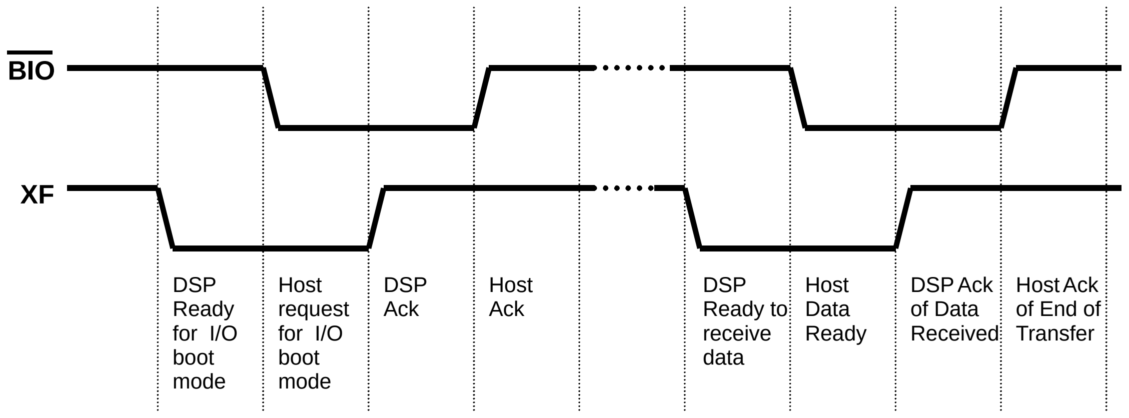
$\overline{\text{BIO}}$
XF
DSP Ready for I/O boot mode
Host request for I/O boot mode
DSP Ack
Host Ack
DSP Ready to receive data
Host Data Ready
DSP Ack of Data Received
Host Ack of End of Transfer

Figure 10. 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 5410 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 11 shows the events in an I/O boot.

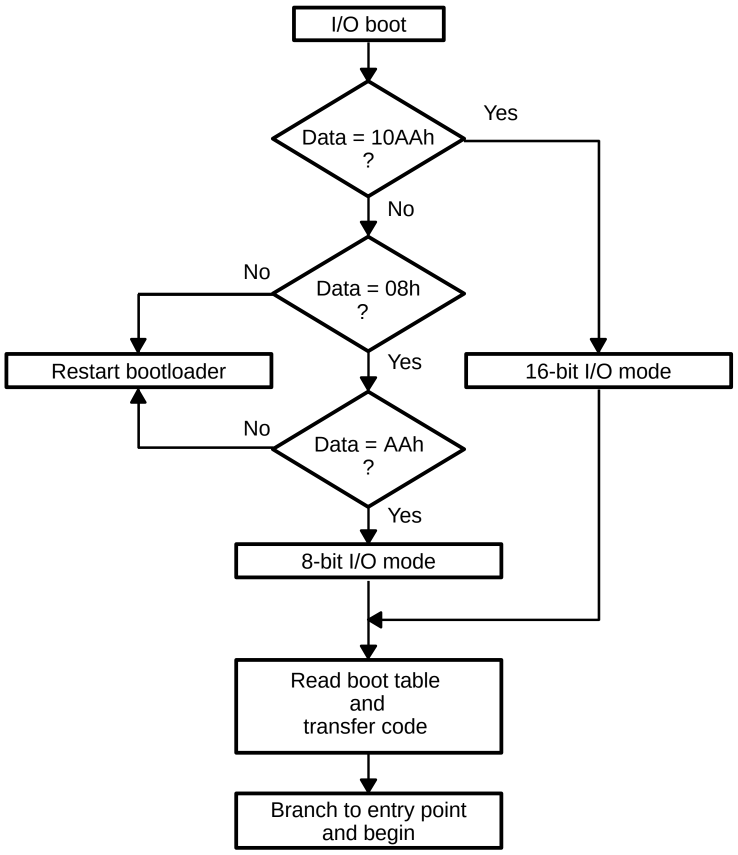

Figure 11. I/O Boot Mode

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 5.

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 5410 initiates a write operation (in case the destination is external memory). Note that the 5410 only drives the external bus when XF is high.

3 Building the Boot Table

To use the features of the 5410 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 5410. Previous versions of the code generation tools do not support the enhanced bootloader options for the 5410 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 5410 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 5410. 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.