Application Report
SPRA375F–September 2008

Using the TMS320VC5503/C5506/C5507/C5509/C5509A Bootloader

Clay Turner, Muhammad Haque

ABSTRACT

This document describes the features of the on-chip bootloder provided with the TMS320VC5503/C5506/C5507/C5509/C5509A digital signal processor (DSP). Included are descriptions of each of the available boot modes and any interfacing requirements associated with them as well as instructions on generating the boot table.

Contents

1 Introduction

This section provides a description of the features of the on-chip bootloader provided with the TMS320VC5503/C5506/C5507/C5509/C5509A digital signal processor (DSP). All references in this document to C5503/C5506/C5507/C5509/C5509A refer to both the TMX320VC5503/C5506/C5507/C5509/C5509A and TMS320VC5503/C5506/C5507/C5509/C5509A, unless otherwise specified.

The on-chip bootloader included in the C5503, C5506, C5507, C5509, and C5509A is the same. However, the supported bootloader features are limited by the features supported on each device. Differences in operation across different devices will be noted throughout this document

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 the code to reside in slow non-volatile memory externally, and be transferred to high-speed memory to be executed.

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:

- Boot from the enhanced host port interface (EHPI)

  The code to be executed is loaded into on-chip memory by an external host processor via the EHPI following reset. Code execution begins when the host indicates to the CPU that the application has been loaded. EHPI boot can be performed with the EHPI, configured as multiplexed or non-multiplexed. The operation of this mode is described in Section 2.3.3.
- Parallel EMIF boot from 16-bit external asynchronous memory

  The bootloader reads the boot table from the external memory interface (EMIF), configured for asynchronous memory. The boot table contains the code or data sections to be loaded, the destination addresses for each of the sections, the execution address once loading is completed, and other configuration information. The operation of this mode is described in Section 2.3.2.
- Standard serial boot through multichannel buffered serial port (McBSP0) (8- or 16-bit supported)

  The bootloader receives the boot table from the McBSP0, operating in standard mode, and loads the code according to the information specified in the boot table. The operation of this mode is described in Section 2.3.4.
- SPI electrically erasable programmable read-only memory (EEPROM) serial boot though McBSP0

  The bootloader receives the boot table from the McBSP0 operating in SPI mode and loads the code according to the information specified in the boot table. The data can be received from an SPI-format serial EEPROM, or from another SPI-compliant serial port. The bootloader supports SPI EEPROMs based on 16- or 24-bit addresses. The operation of this mode is described in Section 2.3.5.
- Universal serial bus (USB) boot

  The bootloader receives the boot table through the on-chip USB peripheral. The protocol supported conforms to the USB standard and the data is loaded to a bulk end-point. The operation of this mode is described in Section 2.3.7.
- Bootloading from inter-integrated circuit (I2C) interface
- The bootloader reads the boot table from an I2C slave device (EEPROM) and loads the code according to the information specified in the boot table.

The bootloader also offers the following features:

- Pin-controlled boot mode selection

  A subset of the general-purpose input/output (I/O) pins is used to select the boot mode. The boot mode selection process is discussed in Section 2.2.
- Selectable entry point

  The desired entry point (the first address of execution after the boot load is complete) is programmable, and is stored in the boot table. The boot table is discussed in Section 2.5.
- Port-addressed register configuration during boot

  Port-addressed registers (such as those used to control peripherals) can be configured during the boot load, providing the ability to modify the clock generator, reconfigure the EMIF strobe timings, or preset peripheral register values. The address and contents of the register to be modified are contained in the boot table. This capability is discussed in Section 2.5.3.
- Programmable delay during boot

  Programmable delays of up to 65535 central processing unit (CPU) clock cycles can be added during the register configuration process, to ensure that new configurations are complete before the boot process continues. This capability is discussed in Section 2.5.3.

1.2 On-Chip ROM Description

The on-chip ROM contains several factory-programmed sections including:

- Bootloader program (described in this document)
- 256-word sine look-up table, consisting of 256 signed Q15 integers representing 360 degrees
- Factory test code, used by TI for testing the device.
- Interrupt vector table

The read-only memory (ROM) map is shown in Table 1.

Table 1. TMS320VC5503/C5506/C5507/C5509/C5509A ROM Memory Map

Starting Byte Address	Contents
FF_0000h	USB bootloader components [(1)]
FF_8000h	Main bootloader code
FF_FA00h	Sine table
FF_FC00h	Factory test code
FF_FF00h	Interrupt vector table
FF_FFFCh	ID code

(1) USB bootmode option is not available on C5503

2 Bootloader Operation

The following sections describe the structure and operation of the production bootloader.

2.1 Bootloader Initialization

When the bootloader begins execution, the program performs some initialization of the DSP prior to loading code. The DSP resources that are configured by the bootloader are described in Table 2.

Table 2. Bootloader Initialization

Resource	Initialization Value
Stack registers	The data stack register (SP) is initialized to address 000090h, and the system stack register (SSP) is initialized to address 000080h.
Stack configuration	The stack configuration is set to the default mode of 32-bit stack, with slow return.
Interrupts	The INTM bit of Status Register 1 (ST1_55) is set to the default value of 1, to disable interrupts.
Memory-mapped registers	Two words are reserved for temporary storage of the entry-point address at 000060h and 000061h.
Sign extension	The SXMD bit of Status Register 1 (ST1_55) is cleared to 0, to disable sign-extension mode. After the bootloader copies all of the sections, SXMD is set back to 1, before execution is transferred to the application.
Compatibility mode	The 54CM bit of Status Register 1 (ST1_55) is set to 1, to enable compatibility mode during and after the bootload.

After the initialization is performed, the bootloader loads the on-chip random access memory (RAM) according to the boot mode selected, and then causes the DSP to begin execution of the loaded code. At that point, the bootload process is complete. Whenever the system is reset, the DSP starts execution of the bootloader again, and the entire bootload process is repeated.

The remaining sections of this document describe the various boot modes and boot tables in detail.

2.2 Boot-Mode Selection

The desired boot mode is selected by setting the four boot-mode select pins BOOTM[0:3]. These pins are sampled after reset when the bootloader program begins execution. The BOOTM pins are shared with the GPIO pins.

- BOOTM3 is shared with IO0
- BOOTM2 is shared with IO3
- BOOTM1 is shared with IO2
- BOOTM0 is shared with IO1

Another GPIO pin, IO4, is used as an output for handshaking purposes on some of the boot modes. Although this pin is not involved in boot-mode selection, be aware that this pin will become active as an output during the bootload process, and design accordingly. After the bootload is complete, the loaded application may change the function of IO[4:0] pins.

The available boot-mode options and their corresponding BOOTM pin configurations are shown in Table 3. Some configurations are reserved for the addition of future boot modes or are not supported on some devices; these configurations should not be selected .

Table 3. Boot-Mode Selection Options

BOOTM[3:0]				Device(s)					
IO.0	IO.3	IO.2	IO.1	C5503	C5506	C5507/ C5509A	C5509	Boot Source	Section
0	0	0	0	-	-	-	-	Reserved	
0	0	0	1	Yes	Yes	Yes	Yes	Serial EEPROM (SPI - 24-bit address) boot from McBSP0	Section 2.3.5
0	0	1	0	No	Yes	Yes	Yes	USB	Section 2.3.7
0	0	1	1	Yes	Yes	Yes	-	I2C EEPROM	Section 2.3.6
0	1	0	0	-	-	-	-	Reserved	
0	1	0	1	Yes	No	Yes	Yes	EHPI (multiplexed mode) boot	Section 2.3.3
0	1	1	0	Yes	No	Yes	Yes	EHPI (non-multiplexed mode) boot	Section 2.3.3
0	1	1	1	-	-	-	-	Reserved	
1	0	0	0	Yes	Yes	Yes	Yes	Execute from 16-bit external asynchronous memory	Section 2.3.1
1	0	0	1	Yes	Yes	Yes	Yes	Serial EEPROM (SPI - 16-bit address) boot from McBSP0	Section 2.3.5
1	0	1	0	Yes	Yes	Yes	-	Parallel EMIF boot (8-bit asynchronous memory)	Section 2.3.2
1	0	1	1	Yes	Yes	Yes	Yes	Parallel EMIF boot (16-bit asynchronous memory)	Section 2.3.2
1	1	0	0	-	-	-	-	Reserved	
1	1	0	1	-	-	-	-	Reserved	
1	1	1	0	Yes	Yes	Yes	Yes	Standard serial (16-bit data) boot from McBSP0	Section 2.3.4
1	1	1	1	Yes	Yes	Yes	Yes	Standard serial (8-bit data) boot from McBSP0	Section 2.3.4

Note: For boot mode selections that require BOOTM3 (IO0) to be low at reset, the TMS320VC5503/VC5506//VC5507/VC5509/VC5509A configure the external parallel port in EHPI mode, not EMIF mode. If boot to external memory is desired for these modes, the external parallel port mode must be set back to Full EMIF mode during the bootload. This can be achieved by using the register configuration capability of the bootloader (see Section 2.5.3). Use this capability to configure the External Bus Selection Register (EBSR). The following statement can be added to the hex conversion utility command line or command file:

-reg_config 0x6C00, 0x0001

Also include any other register configurations required for the EMIF mode desired. It may also be necessary to insert a delay (using the -delay statement) to allow the EMIF configuration to become active before the bootloader begins writing to it. For more information, see Section 2.5.3.

2.3 Boot-Mode Options

2.3.1 Direct Execution From External Asynchronous Memory

When BOOTM[3:0] = 1000b is at reset, the direct execution option is selected. In this mode, the bootloader configures the EMIF for 16-bit asynchronous memory and then transfers control to the external code beginning at byte (program) address 0x400000. The code at this location should be executable code, not a boot table.

To accommodate slow memory, the bootloader configures the EMIF for the maximum timings for the READ SETUP, READ STROBE, READ HOLD, and READ EXTENDED HOLD parameters.

2.3.2 Parallel EMIF-Boot Mode

Parallel EMIF-Boot mode is selected when BOOTM[3:0] = 1010b or 1011b after reset. In this mode, the bootloader reads the boot table from 8- or 16-bit external asynchronous memory. The 8-bit or 16-bit data width is configured based on the selected mode, and cannot be changed during the boot process.

Parallel EMIF mode begins reading the boot table at word address 200000h, which is located in CE1 space. The external memory containing the boot table must start at this location. The execution entry point is contained in the boot table and is programmable.

When this boot mode is initiated, the programmable timings for the EMIF are set to the following:

- READ SETUP is 15 cycles (1111b)
- READ STROBE is 63 cycles (111111b)
- READ HOLD is 3 cycles (11b)
- READ EXTENDED HOLD is 1 cycle (01b)

READ SETUP, READ STROBE, and READ HOLD are set to their most conservative setting to assure interface to a wide range of memory speeds. However, if this default setting proves to be too slow (82 cycles per access), these EMIF timings can be modified using the port-addressed register configuration feature discussed in this section. These timing parameters are controlled in the EMIF CE1 Space Control Register 1 (CE1_1). For more information on the EMIF and the effects of these parameters, see the *TMS320C55x DSP Peripherals Reference Guide* (SPRU317).

Be aware that changing the timing parameters on the EMIF during the boot process can cause the bootload to fail. The external CE1 space must be maintained as asynchronous memory, and with the same data width as the original boot mode chosen. When reconfiguring CE1_1, write the value to MTYPE that matches the original boot mode selected.

Modifications to the EMIF control registers also have some latency before becoming active. The bootloader should not make read requests to the EMIF while the configuration is changing, so the entry in the boot table that reconfigures the EMIF should be followed by a delay of no less than 10 cycles, to allow the EMIF configuration to complete. Also, remember that using the register configuration feature to change the clock generator frequency will change the memory timings generated by the EMIF, since they are cycle-based. Carefully verify that the clock and EMIF configurations being programmed will produce memory timings compatible with the external memory to be used.

During this boot mode, IO4 will go low at the beginning of the boot process. IO4 goes high during execution of the programmable delay feature in the boot table. When the delay is completed, IO4 will go low again. At the end of the bootload, IO4 goes high and the DSP begins execution at the entry-point address. IO4 is not necessary for memories, but can be used as a handshaking signal if some other source is generating the data for the EMIF.

If ARDY goes low during the bootload, the DSP will stall until ARDY is high (ready) again. If the target system does not drive ARDY, it should be pulled high.

2.3.3 EHPI-Boot Mode (C5503/C5507/C5509/C5509A Only)

The description in this section assumes familiarity with the EHPI. For detailed information on the TMS320C55x™ DSP EHPI, see the *TMS320C55x DSP Peripherals Reference Guide* (SPRU317) and the *Using the TMS320VC5509/VC5510 Enhanced HPI* (SPRA741).

EHPI boot in multiplexed mode is selected when BOOTM[3:0] = 0101b at reset. EHPI boot in non-multiplexed mode is selected when BOOTM[3:0] = 0110b at reset. After reset, the bootloader configures the C5503/C5507/C5509/C5509A to support EHPI boot in multiplexed or non-multiplexed mode depending on the state of the BOOTM pins. When the EHPI is configured, IO4 goes low to indicate that the device is ready to receive data from the host. In lieu of monitoring IO4, the host can wait 200 cycles after reset is released, before beginning any transfers through the EHPI.

In EHPI boot mode, an external host can load code and data directly into the DSP memory while the CPU waits. EHPI boot does not use a boot table. The code and/or data sections are directly loaded to the desired locations by the host. When the EHPI has finished loading the application, it signals the CPU to begin execution, and the CPU begins executing at the specified entry point.

During normal operation on the C5503/C5507/C5509/C5509A, the host has access to the area of the memory map below word address 004000h. This space includes internal DARAM blocks 0-3. Because some of the DARAM memory is used by the bootloader code itself, it is recommended that the memory image loaded through the EHPI be limited to word-address range 000100h-003FFFh (16,128 bytes total).

The entry point is the byte address where execution of the application will begin. The entry point is stored in DARAM at word addresses 0060h and 0061h, as shown in Figure 1. The most significant word is stored at 0060h, and the least significant word is stored at 0061h. The least significant 24 bits form the byte address of the entry point. The most significant 8 bits are used as a signal to the CPU when to start executing at the entry-point specified in the low 24 bits. The CPU will continue to loop, monitoring the high 8 bits, as long as they remain all zeroes. This allows the EHPI time to load the desired code and data sections. When the host has completed loading the application, it writes the entry point byte address and a non-zero wait flag value to word addresses 0060h and 0061h, as shown in Figure 1. When a non-zero value is detected in the wait flag, the CPU will branch to the byte address specified in the low 24 bits, and begin execution of the loaded application. Remember that the EHPI host addresses are word-addressed, while program fetches are byte-addressed. So, for example, to load a section of code to be executed from byte address 2000h, the EHPI will load the section to word address 1000h.

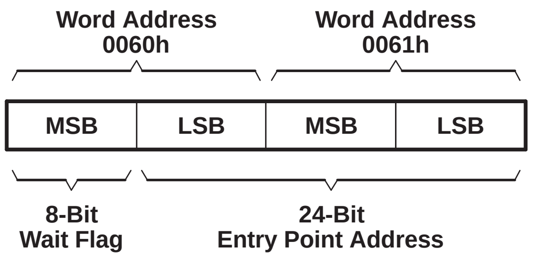

Figure 1. EHPI Wait Flag and Entry-Point Address

Since the CPU transfers control to the application as soon as it detects a non-zero value in the wait flag, address 0060h (the MSW) should be written after address 0061h (the LSW).

The general procedure for boot loading using the EHPI is:

- The $\overline{\text{RESET}}$ pin is released (low-to-high transition) with BOOTM[3:0] = 0101b or 0110b selecting the desired mode.
- The host loads the desired code and data sections into DSP internal memory within the address limits mentioned above.
- The host writes to word address 0061h, with the least significant 16 bits of the desired 24-bit entry-point address.
- The host writes to word address 0060h, with the most significant 8 bits of the desired 24-bit entry-point address in bits 7-0, and a non-zero value in bits 15-8.
- The CPU then transfers execution (branch) to the previously specified entry-point address, and begins running the application.

In the event that the application has been previously loaded and another reset is necessary (warm boot), it is not necessary for the host to reload the application. The host can simply rewrite the entry point and the wait flag after the bootloader begins execution (IO4 goes low).

The peripheral register reconfiguration and delay features are not available during EHPI since these features are associated with the use of a boot table.

2.3.3.1 *Using the HEX55 Utility to Create an Output File*

Although a boot table is not needed for HPI-boot mode, the hex utility can be used to create an output file that can be read by the host. Two possible options are to create a binary file or an ASCII file.

The following is an example of the options that would be used to create a binary file, assuming a 16-bit HPI:

```
-boot            /* generate boot table                         */
-v5510:2         /* boot table format = 2.0                     */
-memwidth 8      /* Binary must have memory width of 8          */
-romwidth 16     /* 16-bit wide i/f -physical bus size          */
-map ehpi16.mxp  /* Name hex utility map file                   */
boot_img.out     /* input file - replace with your file         */
-e start         /* entry point - code exec starts here - replace according to code */
-b               /* Output format = binary                      */
-o ehpi16.bin    /* Name binary output file                     */
```

The following is an example of the options that would be used to create a straight ASCII file, assuming a 16-bit HPI:

```
-boot           /* generate boot table                              */
-v5510:2        /* boot table format = 2.0                          */
-romwidth 16    /* 16-bit wide i/f -physical bus size               */
-memwidth 16    /* 16-bit wide (host) memory                        */
boot_img.out    /* input file - replace with your file              */
-e start        /* entry point - code exec starts here - replace according to code */
-a              /* output format = straight ASCII                   */
-map ehpi16.mxp /* map file                                         */
-o ehpi16.asc   /* boot table file (output file)                    */
```

The -boot option is used to ensure that all initialized sections are placed in memory.

2.3.4 Standard Serial-Boot Mode

The description in this section assumes familiarity with the McBSP. For detailed information on the C55xE McBSP, see the *TMS320C55x DSP Peripherals Reference Guide* (SPRU317).

Standard serial boot mode loads the boot table from McBSP0 in either 8-bit or 16-bit mode, as selected by the BOOTM pins. The McBSP0 receiver is configured by the bootloader, with the following parameters:

- Single phase (RPHASE= 0b)
- One word per frame (RFRLEN1 = 0000000b)
- Word length is 8 or 16 bits (RWDLEN1 = 000b for 8-bit mode, 010b for 16-bit mode)
- Data is right-justified (RJUST = 00b), with one cycle delay (RDATDLY = 01b) for the first bit relative to FSR.
- Receive clock (CLKR0) and receive frame sync (FSR0) are generated externally.

The expected receive data format implied by this configuration is shown in Figure 2 (16-bit shown). The serial port sending data to the DSP must conform to this data format.

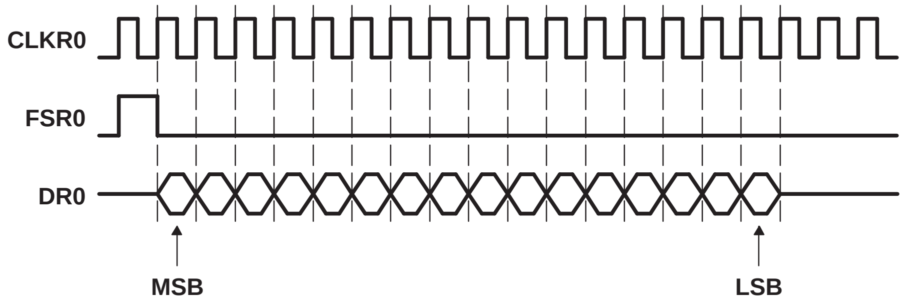

Figure 2. McBSP0 Receive Data Format for Bootload (16-bit shown)

When standard serial-boot mode is selected, the bootloader configures McBSP0 as described above, and then drives IO4 low, to indicate to the sender that the DSP is ready to receive (approximately 200 CPU cycles after the bootloader begins execution). One frame sync is associated with each word (or byte) exchanged. The following conditions must be met to insure proper operation:

- The serial port receive clock externally supplied on CLKR0 should not exceed 1/8 the frequency of the CPU clock.
- Appropriate delay should be provided between the transmission of each word, to prevent receiver overflow. This can be achieved by either slowing down the receive clock frequency, or providing additional serial-port-clock cycles between transmitted words.

As the sender provides the words of the boot table to McBSP0, IO4 responds as a handshaking signal to indicate the state of the boot. When the serial port is ready to receive another word, IO4 goes low. When the serial port is in the process of copying a received word to memory or when a programmed delay is in progress, IO4 is high, and only goes low again when the serial port is ready to receive another word.

An overflow of the receiver causes the bootload to fail. There are two basic options for managing the rate of words sent to the serial port to prevent overflow: use IO4 as a handshaking signal or allow sufficient time between transmissions to prevent overflow.

2.3.4.1 Using IO4 to Prevent Receiver Overflow

As mentioned previously, IO4 goes low when the receiver is ready to receive a word and goes high when some other transaction is in progress. This signal can be polled as an indicator of when the serial port is ready and, therefore, can be used directly to prevent overflow.

There is some latency in the response of IO4 after a word has been received, as shown in Figure 3. The latency is associated with the interaction of the serial port and the bootloader code that interprets the boot table, copies data and initiates the delays. From the point of view of the sender, IO4 will respond to indicate the delay is in progress approximately 50 CPU cycles after the last bit of the word was received. This latency is accounted for automatically if the serial port clock is operated at 1/8 of the CPU clock frequency or slower.

IO4 does not go high after every word received. In 8-bit mode, IO4 will go high after every two or four bytes, depending on whether the part of the boot table being received is a 16-bit or 32-bit object. In 16-bit mode, IO4 will go high after each word (for 16-bit objects) or after every two words (for 32-bit objects).

Polling IO4 provides an automatic method to account for delays in the bootload process due to programmed delays or access delays associated with the EMIF (such as programmed strobe timings or ARDY delays).

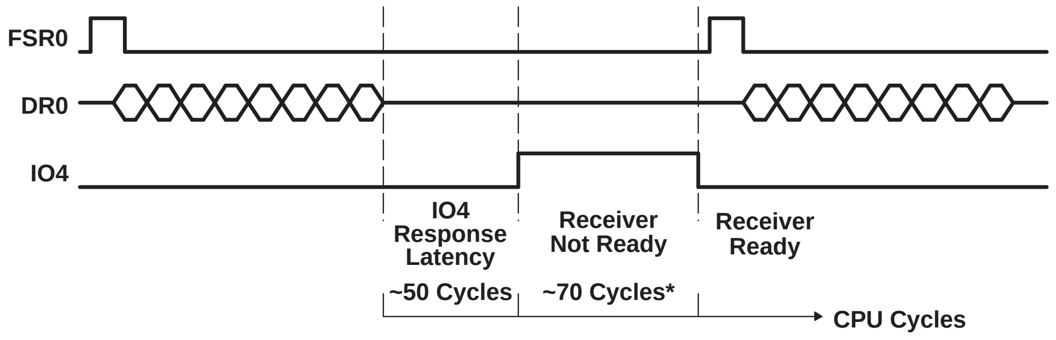

Figure 3. IO4 Latency for Boot Table Programmed Delays

2.3.4.2 Preventing Receiver Overflow Without Polling IO4

If IO4 is not monitored, then appropriate delays must be inserted between transmitted words to prevent receiver overflow. When the destination for the boot table contents is internal memory, the time when the receiver is ready is approximately 120 CPU cycles after the end of the reception of the word (as shown in Figure 3). The sender should allow at least this much time between transmitted words destined for internal memory on the DSP.

If the programmed delay feature is used, additional time must be included to accommodate the extra delay. Similarly, if the destination for the code or data is external memory, the sender must allow additional time to allow for the memory conditions. For example, assume the destination for a section of code is external asynchronous memory with the following conditions:

- WRITE SETUP is 2 CPU cycles
- WRITE STROBE is 5 CPU cycles
- WRITE HOLD is 2 CPU cycles
- WRITE EXTENDED HOLD is 1 CPU cycle

An additional 10 CPU cycles (2+5+2+1) is necessary for each word to be moved. So, the time between transmission of words should be no less than 130 CPU cycles.

Since the delay is in terms of CPU cycles (not serial-port-clock cycles), the required timing can be met by inserting additional serial-port-clock cycles between transmitted words, by slowing down the serial port clock relative to the CPU clock, or both. Since the delay after the reception of each word (or byte) is not the same, you must select a word (or byte) rate that accommodates the worst-case delay.

When the end of the boot table is received, IO4 will be driven high, and the CPU will branch to the execution entry point specified in the boot table, and begin execution.

2.3.5 SPI EEPROM-Boot Mode

The description in this section assumes familiarity with the McBSP SPI operation using the clock-stop mode. For detailed information on the C55x McBSP, see the *TMS320C55x DSP Peripherals Reference Guide* (SPRU317).

The DSP bootloader supports boot from SPI EEPROMs or a device operating as an SPI slave that emulates the appropriate format. The bootloader supports SPI EEPROMs based on 16-bit byte addresses (up to 64k bytes) as mode BOOTM = 1001b. The bootloader supports SPI EEPROMs based on 24-bit byte addresses (up to 16M bytes) as mode BOOTM = 0001b.

In SPI EEPROM-boot mode, the DSP acts as an SPI master, and the memory acts as the slave. The minimum connection required between McBSP0 and the SPI EEPROM is shown in Figure 4. CLKX0 is the master clock driving the EEPROM CLK signal. In the SPI EEPROM-boot mode, the CLKX0 period = 244 x (DSP input clock period). DX0 transmits data to the EEPROM serial data input (SI) signal. DR0 receives data from the EEPROM serial data output (SO) signal. IO4 is used to operate the EEPROM chip select (CS) signal. IO4 automatically enables the EEPROM when the bootload is ready to begin, and disables the EEPROM when the bootload is complete.

Some serial EEPROMs may additionally provide write-protect (WP) and HOLD signals. Write-protect prevents an external device from writing to internal memory and registers in the EEPROM. Since the bootloader only performs reads on the EEPROM, the state of the write-protect function is not relevant. If it is not used, the pin can be pulled inactive (high). The HOLD input is used to suspend serial input to the EEPROM. Having this pin active prevents the bootloader from operating correctly. The HOLD pin (if present) should be pulled inactive (high).

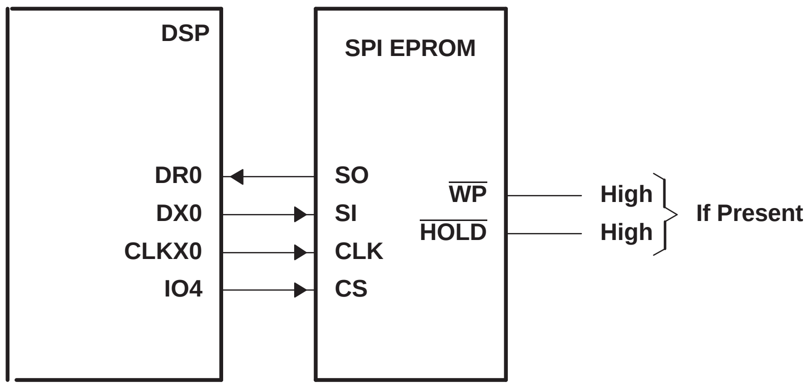

Figure 4. Signal Connections for SPI EEPROM-Boot Mode

The bootloader reads the boot table from the EEPROM as a sequential block of data. It does not perform random accesses. For 16-bit SPI EEPROM mode, the format of the beginning of the transfer is shown in Figure 5.

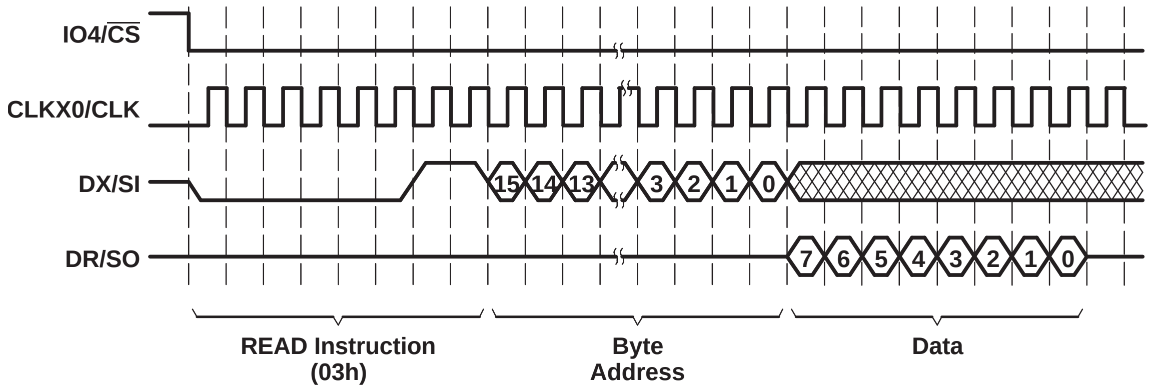

Figure 5. SPI EEPROM Mode Transfer Protocol With 16-Bit Addresses

The process begins with the DSP driving IO4 low (EEPROM CS). Then the DSP issues a READ instruction (03h) to the EEPROM, followed by the starting byte address, which will always be address zero. The EEPROM responds by sending data bytes back to the DSP. The DSP does not resend the address for each byte, but depends on the ability of the serial EEPROM to automatically increment the address internally. The DSP continues to read bytes sequentially from the EEPROM until the entire boot table has been transferred. Then the DSP drives IO4 high, to disable the EEPROM chip select, and the bootloader branches to the beginning of the loaded application, to begin execution.

The process is identical for the 24-bit bit address mode except the initial address transmitted to the EEPROM is 24 bits instead of 16 (as shown in Figure 6). For either of these modes, the boot table must be programmed into the EEPROM as a single, continuous image starting at EEPROM address zero.

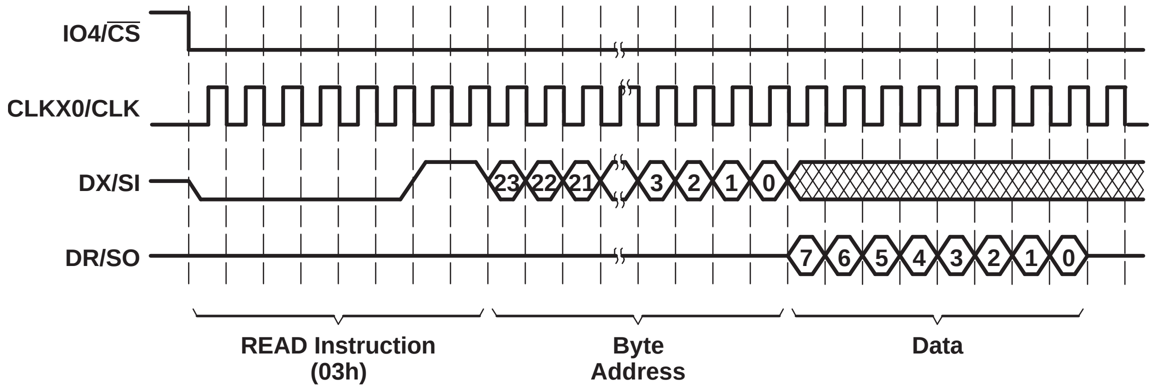

Figure 6. SPI EEPROM Mode Transfer Protocol With 24-Bit Addresses

Although Figure 5 and Figure 6 show the address and data being continuous, there may be gaps between the READ instruction, the address, and all of the subsequent data bytes. Since the DSP is the master, it only operates the clock when it is ready for the next byte, so no user-intervention is required to accommodate delays during bootload.

2.3.6 I2C EEPROM-Boot Mode

The description in this section assumes familiarity with the I2C operation using the master receiver and master transmitter modes. For detailed information on the I2C, see the *TMS320VC5501/5502/5503/5507/5509 DSP Inter-Integrated (I2C) Module Reference Guide* (SPRU146).

The DSP bootloader supports boot from I2C EEPROM or Slave I2C devices emulating EEPROM behavior. The bootloader has the following requirements for the I2C EEPROM:

- The memory device complies with Philips I2C Bus Specification v2.1 and responds to slave address 50h.
- The memory device uses two bytes (up to 64K bytes) for internal addressing.
- The memory device has the capability to auto-increment its internal address counter such that the contents of the memory device can be read sequentially.

In I2C-boot mode, the DSP acts as the master and the I2C EEPROM acts as the slave. The connection required between the DSP and the I2C EEPROM is shown in Figure 7. The required pull-ups on SDA and SCL depends on the number of I2C devices sharing the bus. Consult the Philips I2C Bus Specification for details on pull-ups requirements. Normally if the I2C bus is shared only by the DSP and the I2C EEPROM, 5K pull-ups should work.

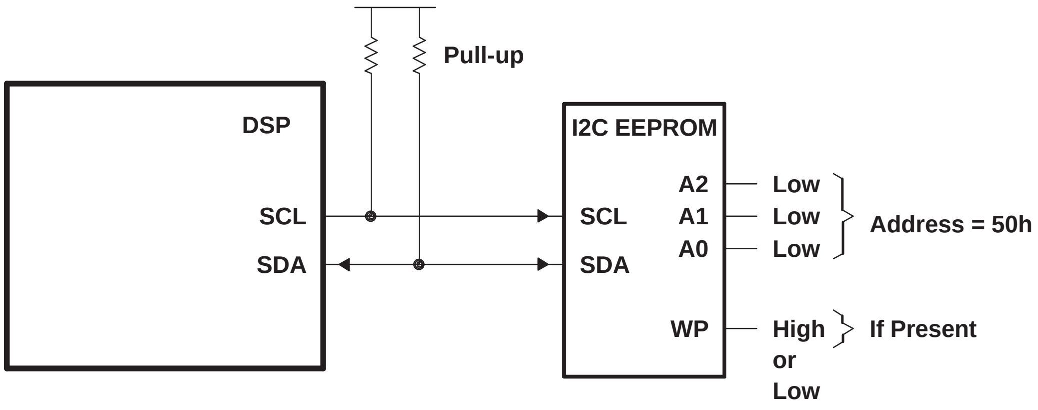

Figure 7. Signal Connections for I2C EEPROM-Boot Mode

Some I2C EEPROMS have a write-protect (WP) feature that prevents unauthorized writes to memory. This feature is not needed for bootloading purposes since the DSP only reads data from the I2C EEPROM. The write-protect feature can be enabled or disabled without impacting the bootloader operation.

The operating frequency of the I2C bus can be calculated using the following formula:

SCL (High) = 15x (DSP input clock period)

SCL (Low) = 15x (DSP input clock period)

A CLKIN frequency of 12 MHz or less should be chosen so that the frequency of the I2C bus is not greater than 400 kHz since that is the maximum speed supported by the I2C module.

The I2C boot process begins with the DSP using a random read command to read address zero of the EEPROM. A random read command, as shown in Figure 8, consists of a dummy write command with the address set to zero, immediately followed by a current address read command. The EEPROM responds by sending a data byte to the DSP. The DSP depends on the ability of the EEPROM to automatically increment the address internally to read the subsequent bytes. All subsequent bytes are read using the current address read command as shown in Figure 9.

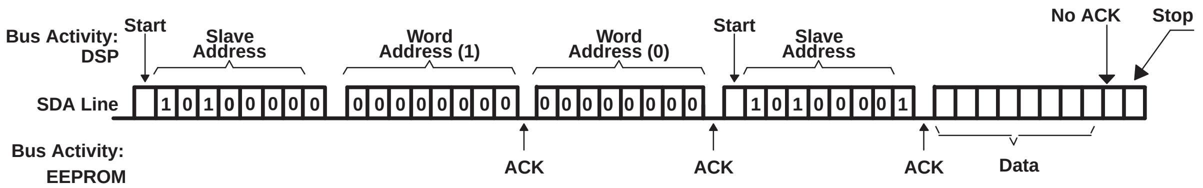

Figure 8. Reading the First Byte in a 64Kbyte Block

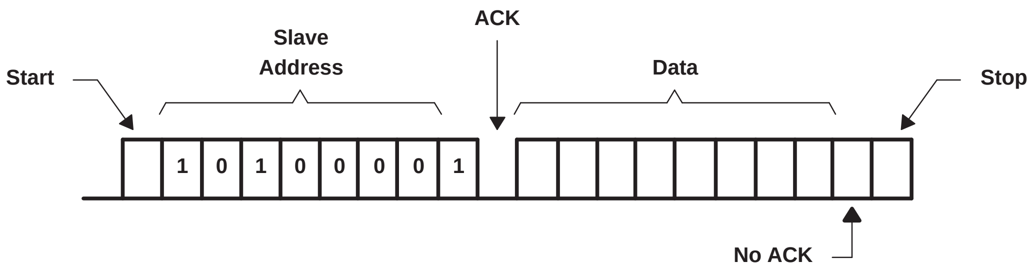

Figure 9. Current Address Read Command

The DSP stops requesting data when the first zero-length memory section is encountered in the boot table. All data bytes are processed in accordance with the boot table description given later in this document.

The bootloader flags the start of I2C-boot mode by setting the GPIO4 low during the random read operation. The GPIO4 is then set high during the rest of the read operations. The I2c module remains enabled (but no bus activities) at the end of the boot process, until the user's application turns the module off.

2.3.7 USB-Boot Mode (C5506/C5507/C5509/C5509A only)

USB-boot mode is selected when BOOTM[3:0] = 0010b after reset. For more detailed information on the USB-boot mode, see *Using the TMS320VC5506/C5507/C5509/C5509A USB Bootloader* (SPRA840).

2.4 IO4 Behavior

On the C5503/C5506/C5507/C5509/C5509A bootloader, IO4 is configured as an output used for multiple functions, depending on the boot mode selected as indicated below:

- In the standard serial-boot-mode modes, IO4 is used to indicate that the serial port is ready to receive data. If a programmed delay occurs, IO4 is not ready (high) until the delay is completed, and then ready (low) when the serial port is ready to receive again. IO4 also goes not ready while data is being moved. It can be used as a handshaking signal to prevent receiver overflow.
- In the external asynchronous memory-boot modes, IO4 goes low at the beginning of the boot process and only goes high during the programmed delays, as an indication of the delay. When the bootload is complete, IO4 goes high.
- In the serial EEPROM-boot modes, IO4 is used as a CS signal to the serial EEPROM. It goes low at the beginning of the boot process, and goes high when the boot process is complete. IO4 does not change during delays in this mode, but since the DSP is the master, delays are handled automatically.
- In the I2C EEPROM-boot mode, IO4 is toggled to indicate the beginning of I2C bus activity.

2.5 The Boot Table

The boot table is a block of data that contains the code and data sections to be loaded by the bootloader as well as other information including the entry point address, register configurations, and programmable delays. The boot table is created by the hex conversion utility (a standard component of the TMS320C55x™ Assembly Language Tools), based on the common object file format (COFF) output of the linker for the application code. The hex conversion utility provides several output options, such as industry-standard ASCII formats that can be used to program parallel or serial EEPROMs, and formats that can be used in code for a host to transmit the boot table to the DSP. A more detailed description of the role of the hex-conversion utility in creating the boot table is covered later.

2.5.1 DSP Resources Used by the Bootloader

The bootloader program uses several internal resources on the DSP during the entire boot process. These resources are reserved for use by the bootloader and should not be altered until the bootload is completed, and the bootloader has passed control to the loaded application code.

The following resources are used by the bootloader:

- Accumulators: AC0, AC1, AC2, AC3
- Auxiliary registers: XAR5, XAR6
- Temporary registers: T0, T1, T2, T3
- The entry-point address is stored as word addresses 0060h and 0061h.
- The stack pointer (SP) is located at word address 0090h.
- The system stack pointer (SSP) is located at word address 0080h.

To avoid corruption of these memory locations, the sections contained in the boot table should not contain any destinations lower than word address 0100h (byte address 0200h).

2.5.2 The Boot Table Structure

The boot table has a specific format that is independent of the boot mode chosen and contains information relating to program sections, data sections and other information used by the bootloader. The components of the boot table are shown in Figure 10.

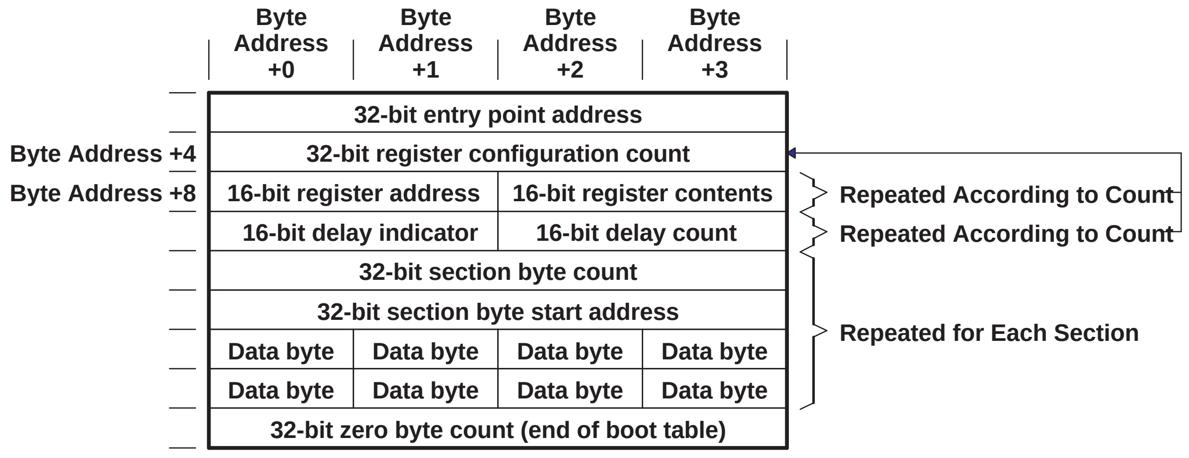
Byte Address +0
Byte Address +1
Byte Address +2
Byte Address +3
Byte Address +4
Byte Address +8
32-bit entry point address
32-bit register configuration count
16-bit register address
16-bit register contents
16-bit delay indicator
16-bit delay count
32-bit section byte count
32-bit section byte start address
Data byte
32-bit zero byte count (end of boot table)
Repeated According to Count
Repeated According to Count
Repeated for Each Section

Figure 10. Boot Table Structure

A description of each component of the boot table is given below:

- 32-bit Entry-Point Byte Address is the address where the bootloader begins execution after the application is loaded.
- 32-bit Register Configuration Count is the number of registers to be configured, or delays to be implemented, during the bootload process (see Section 2.5.3). The following four components are only included in the boot table if the register configuration count is non-zero.
  - 16-bit Register Address, for the register to be configured
  - 16-bit Register Contents contains the value to be programmed in the above register.
  - 16-bit Delay Indicator (FFFFh) indicates a delay will be implemented.
  - 16-bit Delay Count contains the number of CPU cycles to delay.
- 32-bit Section Byte Count contains the number of bytes to be copied in the current section.
- 32-bit Section Start Byte Address is the destination address of the current section.
- Data Bytes are the actual data in the section to be copied.
- 32-bit Zero Byte Count (00000000h) indicates the end of the boot table.

2.5.3 Register Configuration and Delay During Boot

The C5503/C5506/C5507/C5509/C5509A bootloader supports a feature that allows peripheral port-addressed registers to be configured during the boot process before the code and data sections are copied. This feature provides the capability to change the device mode for specific purposes, such as changing the clock generator frequency (to speed up the boot process) or configuring the EMIF external memory spaces.

When the table is created, a register configuration entry is added to the boot table when the option *-reg_config address, data* is added to the command line in the hex conversion utility. In this option, address is the port address of the register to be configured, and data is the data that is written to the register. For example, to program the DSP Clock Mode Register (CLKMD) (CLKMD is at port address 1C00h) with the value of 0, the following option would be added to the hex utility command line:

-reg_config 0x1C00, 0x0000 ; write 0000h to port address 1C00h

The hex conversion utility adds a 32-bit entry to the boot table containing this information. The first 16 bits are the port address, and the second 16 bits are the contents to be written to that address. Multiple register configurations can be included in the boot table, and one is added for each -reg_config reference in the command line (or command file).

Since some configurations of the device may have some latency before becoming active, a delay feature is also available that can delay the boot process until the configuration changes are valid. The delay is implemented in a similar manner.

The option *-delay_count* is added to the hex utility command line to generate a delay. The delay_count is a value between 1 and 65535, and represents the number of CPU cycles to wait before the bootloader proceeds with the boot process. The delay option puts a 32-bit entry in the boot table, in which the first 16 bits are FFFFh, and the second 16 bits are the delay count. Since this is the same format as the register configuration feature, the bootloader always interprets a reference to port address FFFFh as a request for a delay, and uses the next 16 bits as the delay count.

Some examples where inserting delays are useful are:

- Changing the clock generator

  The delay can stall the boot process until the clock generator is locked on the new frequency and is running at the appropriate speed.
- Configuring the EMIF memory type and timings

  If it is necessary to change the configuration of one of the EMIF external spaces, the delay can be used to wait until the changes have become valid, and the EMIF is ready to operate.

The bootloader has reserved port address FFFFh for the delay feature, and has reserved port addresses FFF0h-FFFEh for future features. These port addresses cannot be used in the register configuration feature. If port address FFFFh is used, it will be interpreted as a delay. Only port addresses below FFEFh are interpreted as register configurations.

Note that the bootloader provides no protection with regard to the programmed register contents specified in these features. It is your responsibility to configure register values correctly. Altering peripheral registers that are associated with the bootloader can cause the bootload to fail. Some guidelines for register configuration during boot are given below:

- If the serial boot modes are used, do not alter the configuration of any of the registers associated with McBSP0.
- If the EMIF boot modes are used, do not alter the configuration of any of the registers associated with EMIF CE1 space. This space is where the boot table is located, and if reconfigured, the ability of the bootloader to read the rest of the boot table may fail. The programmable memory timings for CE1 space may be altered, but you should carefully consider the effect of the changes on the memory timing and the ability of the bootloader to continue to read the memory. Changing the memory timing for CE1 space can speed up the boot process, but it can also cause the boot to fail if changed incorrectly. MTYPE for CE1 space should never be changed.
- If the clock generator is reconfigured, think carefully about the timing effects on the boot process. Changing the clock frequency will change the EMIF timings (since the EMIF timings are relative to the DSP clock), and may cause interface timings that are incompatible with the external memory used. Frequency changes may also affect whether the serial port timing provided externally still meets the data sheet and bootloader requirements. Consider these issues very carefully before making any changes.

The hex conversion utility automatically counts the number of register configurations and delays specified in the command line (or command file), and inserts this information in the boot table. The register configurations and delays are inserted in the boot table (and executed by the bootloader) in the order they are specified in the hex conversion utility command line or command file. Once all of the configurations have been completed during the bootload, the bootloader proceeds to copying code and data sections.

2.5.4 Code and Data Sections in the Boot Table

Code and data sections are inserted into the boot table automatically by the hex conversion utility. The hex conversion utility uses information embedded by the linker in the .out file to determine each section's destination address and length. Adding these sections to the boot table requires no special intervention by you. The hex conversion utility adds all initialized sections in the application to the boot table. The remaining information included in this section describes the format of the sections in the boot table.

In the C55x™ DSP architecture, program sections are byte-addressed, have variable widths (in bytes), and may start and/or end on byte boundaries. Data sections are word-addressed, and always start and end on word boundaries. To accommodate these two types of sections, the boot table pads program sections to temporarily align the sections to start and end on word boundaries. This structure causes the bootloader code to be simpler and execute more quickly. These added *pad bytes* do not affect the content of the sections, or their address alignments, because the bootloader code strips the pad bytes out before writing the sections to their destinations. However, if you read the output of the hex conversion utility, the pad bytes will be present.

If a program section starts on an even byte address, no pad byte is added to the beginning of the section. If a program section starts on an odd byte address, one pad byte is added to the beginning of the section. If a program section ends on an even byte address, one pad byte is added to the end of the section. If a program section ends on an odd byte address, no pad byte is added to the beginning of the section. Because of this structure in the boot table, *all sections to be included in the boot table must contain at least two bytes*.

Each section is added to the boot table with the same format. The first entry is a 32-bit count representing the length of the section in bytes. The next entry is a 32-bit destination address. This is the address where the first byte of the section is copied. Although these entries reserve 32 bits in the boot table for alignment, the destination address and byte count will not exceed 24 bits, since the address range of the C5503/C5506/C5507/C5509/C5509A is limited to 24 bits. The remainder of the section in the boot table contains the actual program or data information for that section.

The bootloader continues to read and copy these sections until it encounters a section whose byte count is zero. This is the indication of the end of the boot table, and the bootloader branches to the entry-point address (specified at the beginning of the boot table), and begins execution of the application.

2.5.5 Creating the Boot Table

To create the boot table, proceed through the following steps:

1. Use the hex conversion utility (HEX55.exe) revision 2.10 or later. Earlier versions may not support the boot table features correctly.
2. Use the -boot option, to cause the hex conversion utility to create a boot table.
3. Use the -v5510:2 option. Even though this option refers to the TMS320C5510, it applies to the C5503/C5506/C5507/C5509/C5509A also. This option is very important since some early versions of the C55x hex conversion utility supported a different boot table format. The wrong boot table format will cause the bootloader to fail.
4. Specify the boot type -parallel8, -parallel16, -serial16 or -serial8. Table 4 shows the correct option to select for each supported boot mode. EHPI-boot mode does not require a boot table.
5. Specify the entry point using the -e entry_point_address option. The entry point is the address that the bootloader will transfer execution when the boot load is complete.
6. Specify the desired output format. For detailed information on the available hex conversion utility output formats, see the *TMS320C55x Assembly Language Tools User's Guide* (SPRU280) .
7. Specify the output filename using the -o output_filename option. If you do not specify an output filename, the hex conversion utility will create a default filename based on the output format.

Table 4. Boot Mode Types for the Hex Conversion Utility

BOOTM[3:0]	For Boot Mode Source …	Include this option …
0001	Serial EEPROM (SPI) Boot from McBSP0 supporting 24-bit address	-serial8
0010	USB	-serial8
1001	Serial EEPROM (SPI) Boot from McBSP0 supporting 16-bit address	-serial8
1011	External Asynchronous Memory (16-bit)	-parallel16
1110	Standard Serial Boot from McBSP0 (16-bit)	-serial16
1111	Standard Serial Boot from McBSP0 (8-bit)	-serial8

Section 2.5.5.1 and Section 2.5.5.2 show how to set the hex conversion utility options to create a boot table.

2.5.5.1 *Example 1. Creating the Boot Table for Application my_app.out to Boot From 16-Bit External Asynchronous Memory*

To create a boot table for the application my_app.out with the following conditions:

- Desired boot mode is from 16-bit external asynchronous memory
- No registers will be configured during the boot
- No programmed delays will occur during the boot
- Desired output is in TI-Tagged format, in a file called my_app.hex

Use the following options on the hex conversion utility command line or command file:

```
-boot           ; Option to create a boot table
-v5510:2        ; Use C55x boot table format for TMS320VC5509
-parallel16     ; Boot mode is 16-bit external asynchronous memory
-t              ; Desired output format is TI-Tagged
-o my_app.hex   ; Specify the output filename
my_app.out      ; Specify the input file
```

2.5.5.2 *Example 2. Creating the Boot Table for Application my_app.out to Boot Mode From 8-Bit Standard Serial Mode*

To create a boot table for the application my_app.out with the following conditions:

- Desired boot mode is from 8-bit standard serial boot
- Configure the CLKMD register (port address 0x1C00) with the value 0x2180
- After the CLKMD register is configured, wait 256 cycles before continuing the boot
- Desired output is Intel format in file a called my_app.io

Use the following options on the hex conversion utility command line or command file:

```
-boot                         ; Option to create a boot table
-v5510:2                      ; Use C55x boot table format for TMS320VC5509
-serial8                      ; Boot mode is 8-bit standard serial boot
-reg_config 0x1c00, 0x2180    ; Write 0x2180 to register address 0x1C00
-delay 0x100                  ; Delay for 256 CPU clock cycles
-i                            ; Desired output format is Intel format
-o my_app.io                  ; Specify the output filename
my_app.out                    ; Specify the input file
```

For detailed information about the C55x hex conversion utility, see the *TMS320C55x Assembly Language Tools User's Guide* (SPRU280).

3 Debugging Bootloader Issues

This section is designed to assist in the debug of bootloader issues. The recommended approach is to break down the problem by verifying what external indicators have occurred, and then further isolate the problem to the boot media, hardware, or software.

3.1 *Direct Execution from External Asynchronous Memory*

The DSP configures the CE3 space for asynchronous memory and branches to the entry point address at byte address location 0x400000 in external CE3 space.

Check:	If no, then verify that:
Does CPU hit a breakpoint at the entry point byte address 0x400000?	• BOOTM[3:0] pins are set to 1000b at reset. • The DSP is released from reset with a low-to-high transition of the reset signal. • Executable code (not a boot table) is present beginning at byte address 0x400000 in external asynchronous memory. Use a IEEE Standard 1149.1- 1990, IEEE Standard Test Access port and Boundary-Scan Architecture (JTAG) emulator to verify. • The signal connections between the DSP and asynchronous memory are good. Use a JTAG emulator to peek and poke data to external memory. • The external memory has been programmed properly, with start of the executable code at 0x400000.

3.2 Parallel EMIF-Boot Mode

The IO4 signal goes low at the start of the bootload process, then high, then low again, then high, serving as an indicator of the progress of the bootloader. Lastly, the CPU branches to the entry point and begins execution of the application code.

Check:	If no, then verify that:
Does IO4 go low at the start of the bootload process?	• BOOTM[3:0] pins are set to 1010b for 8-bit external memory, 1011b for 16-bit external memory. • The DSP is released from reset with a low-to-high transition of the reset signal.
Does IO4 go high during execution of the programmable delay, and then low a second time after the delay?	• There is a delay programmed in the boot table. • ARDY is not stuck low, and that ARDY is pulled high if not driven by the target system. • The timing parameters on the EMIF are not changed during the bootload process. If CE1 space is reconfigured, the value of MTYPE must be maintained.
Does IO4 go high a second time, and does CPU hit a breakpoint at the entry point address? Use a JTAG emulator and debugger such as Code Composer Studio™.	• ARDY is not pulled or driven low; if not used it should be driven high, otherwise it will toggle. • The start of the boot table can be found at word address location 0x200000 in CE1 space. Use an XDS emulator and debugger such as Code Composer Studio to verify. • The correct code entry point is specified in the boot table. The entry point must be specified as a byte address. • HEX55 tool version 2.10 or later was used to create the boot table, and the C5510:2 option was used.

3.3 Host Port Interface Boot Mode

The GPIO signal is driven high at the start of the bootload process, and then low after a delay, when the DSP is ready to receive data. When the bootload process is complete, the CPU branches to the entry point and begins code execution.

Check:	If no, then verify that:
Does the IO4 pin go low at the start of the bootload process, indicating that the DSP is ready to receive data from the host?	• BOOTM[3:0] pins are set to 0101b for multiplexed mode, and 0110b for non-multiplexed mode, at reset. • The DSP is released from reset with a low-to-high transition of the reset signal.
Does the host write the entry point address and a non-zero wait flag value to word addresses 0x61 and 0x60 in the proper order, to indicate that the application has been loaded? Use a JTAG and debugger such as Code Composer Studio.	• The signal integrity between the host and the DSP is good by writing values to DSP memory from the host and reading them back. You can also use a JTAG emulator to verify that data is being written properly to DSP memory. • The host is completing its data transfer. • If the host is not monitoring IO4, ensure that there is a 200 cycle delay between the DSP release from reset and the start of code download.
Does the CPU hit a breakpoint at the entry point that is set in word address 0x60 and 0x61? Use a JTAG emulator and debugger such as Code Composer Studio.	• The word addresses 0x60 and 0x61 contain a byte address for the entry point, and not a word address. • The breakpoint is set at this byte address in program memory. • The host writes to word address 0x61 and 0x60 in the proper bit configuration and order. 0x61 should be written first with the least significant 16 bits of the entry point byte address. 0x60 should then be written with the most significant 8 bits of the entry point address in the lower half, and the non-zero flag value in the upper half. • The host does not load any other data to word addresses 0x60 and 0x61, except for the entry point address and the flag value.
Does the user application begin to execute properly?	• The entry point contains the start of executable code (not a boot table). The host loads code to the DSP memory by word address, while program fetches from the DSP's point of view byte addressed. • The host does not attempt to load code outside of the word address range 0x0100 - 0x3FFF.

3.4 Standard Serial-Boot Mode

The IO4 signal can be used as an external indicator of the status of the standard serial boot process. IO4 is driven low, then toggles while it acts as a handshaking signal during the download, and is finally driven high at the end of the process, at which point the CPU branches to the code entry point specified in the boot table.

Check:	If no, then verify that:

Does the DSP drive the IO4 signal low and configure the McBSP as follows:
RPHASE = 0b, RFRLEN1 = 0h,
RWDLEN1 = 000b for 8-bit mode or 010b for 16-bit mode,
RJUST = 00b,
RDATDLY = 01b, externally generated CLKR0 and FSR0?
	• BOOTM[3:0] pins are set to 1101b for 16-bit mode, and to 1111b for 8-bit mode.
	• The DSP is released from reset with a low-to-high transition of the reset signal.

Does GPIO toggle between low (serial port ready to receive) and high (serial port not ready), acting as a handshaking signal during the bootload process?	• The receiver has not overflowed. To avoid overflow, either use IO4 as a handshaking signal as described in Section 2.3.4.1, or allow at least 130 CPU clock cycles between transmission of words. • The serial device is connected to the DSP via McBSP 0. • The external device is generating clock and frame sync signals.
Is GPIO4 driven high, and does the CPU hit a breakpoint at the entry point address? Use a JTAG emulator and debugger such as Code Composer Studio.	• The boot table contains the correct entry point byte address. Open the file containing the boot table in an editor and make sure that the first four bytes contain the 32-bit entry point address. • The externally supplied serial port receive clock does not exceed 1/8 of the CPU clock. • The boot media is programmed properly. • The proper options were chosen to create the boot table. • HEX55 tool version 2.10 or later was used to create the boot table, and the C5510:2 option was used.

3.5 SPI EEPROM-Boot Mode

The IO4 signal is driven low at the start of the SPI EEPROM bootload process, after which the DSP exchanges data with the EEPROM. After the boot table is transferred, the IO4 signal is driven high, and the CPU branches to the code entry point specified in the boot table.

Check:	If no, then verify that:
Does the DSP drive the IO4 (EEPROM/CS) signal low?	• BOOTM[3:0] pins are set to 1001b for 16-bit byte address EEPROMS, and to 0001b for 24-bit byte address EEPROMS. • The DSP is released from reset with a low-to-high transition of the reset signal.
Does the DSP issue a read instruction (03h) and the starting byte address (00h) to the EEPROM on the DX0 signal, followed by bytes sent from the EEPROM to the DSP on the DR0 pin?	• The signals between the DSP and the EEPROM are correct and intact. • The EEPROM is connected to McBSP0 of the DSP. • The $\overline{\text{HOLD}}$ signal is pulled inactive high. • The speed of the EEPROM is compatible with a serial port clock rate equal to the DSP input clock period X 244.
Does the DSP drive the IO4 signal high to indicate that the boot table has been transferred?	• The boot table is programmed into the EEPROM as a single continuous image starting at EEPROM address 0. • HEX55 tool version 2.10 or later was used to create the boot table, and the C5510:2 option was used.
Does the CPU hit a breakpoint at the code entry point byte address? Use a JTAG emulator and debugger such as Code Composer Studio.	• The boot table contains the correct entry point byte address. Open the file containing the boot table in an editor and make sure that the first four bytes contain the 32-bit entry point address.

3.6 I2C EEPROM-Boot Mode

At the start of the I2C bootload process, the bootloader asserts GPIO4 low. GPIO4 then toggles as data is copied from the EEPROM to DSP memory. Finally, the bootloader sets GPIO4 as an input and branches to the entry point address specified in the boot table.

Check:	If no, then verify that:
Does GPIO4 go low at the start of the bootload process?	• BOOTM[3:0] pins are set properly. • The DSP is released from reset with a low-to-high transition of the reset signal. • Connections between I2C device and DSP peripheral are correct and intact. • There are pull-ups on the SDL/SDA signals. • The I2C device configuration is compatible: Philips bus spec v2.1 compliant, slave address 50h, has auto-increment capability, operating frequency is < = 12MHz. • The I2C bus is running at < = 400 KHz.
Does GPIO4 toggle during the random read part of the bootload process?	• The boot table was created with the correct options. • The boot media is programmed properly. • The proper options were chosen to create the boot table. • HEX55 tool version 2.10 or later was used to create the boot table, and the C5510:2 option was used.
Does the GPIO4 signal return to low at the end?	• User-application code turns off the I2C module at the start of execution.
Does the CPU hit a breakpoint at the code entry point byte address? Use a JTAG emulator and debugger such as Code Composer Studio.	• The boot table contains the correct entry point byte address. Open the file containing the boot table in an editor and make sure that the first four bytes contain the 32-bit entry point address.

3.7 USB-Boot Mode

USB-Boot mode is addressed in the application report titled *Using the TMS320VC5506/C5507/C5509/C5509A USB Bootloader* (SPRA840).

4 References

- *TMS320C55x DSP Peripherals Reference Guide* (SPRU317)
- *Using the TMS320VC5509/VC5510 Enhanced HPI* (SPRA741)
- *TMS320C55x Assembly Language Tools User's Guide* (SPRU280)
- *TMS320VC5509 Fixed-Point Digital Signal Processor Data Manual* (SPRS163)
- *Using the TMS320VC5506/C5507/C5509/C5509A USB Bootloader* (SPRA840)
- *TMS320VC5501/5502/5503/5507/5509 DSP Inter-Integrated (I2C) Module Reference Guide* (SPRU146)