Using the TMS320VC5510 Bootloader

Clay Turner *C5000 Applications*

ABSTRACT

This document describes the features of the on-chip bootloader provided with the TMS320VC5510 Digital Signal Processor (DSP). Included are descriptions of each of the available boot modes and any interfacing requirements associated with them, instructions on generating the boot table, and information on migration from the prototype (TMX320VC5510, revision 1.x) to the production (TMS320VC5510) bootloader.

This document contains preliminary data current as of the publication date and is subject to change without notice.

Important Notice Regarding Bootloader Program Contents:

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

Contents

List of Figures

List of Tables

List of Examples

1 Introduction

This section provides a description of the features of the on-chip bootloader provided with the TMS320VC5510 digital signal processor (DSP). All references in this document to VC5510 refer to the TMS320VC5510 production device unless otherwise specified. Prototype versions of the device are referenced as TMX320VC5510 (TMX prefix instead of TMS prefix).

For customers who received device samples, the production bootloader is present on silicon revision 2.0 and later. The prototype bootloader is present on silicon revisions 1.x.

1.1 Bootloader Features

The VC5510 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 VC5510 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 via the EHPI.

- Parallel EMIF boot from 8-, 16- or 32-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.

- Standard serial boot through 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.

- SPI EEPROM serial boot though McBSP0

  The bootloader receives the boot table from the McBSP0 operating in SPI master 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 operating as an SPI slave.

The bootloader also offers the following features:

- Pin-controlled boot mode selection

  A subset of the general-purpose I/O pins is used to select the boot mode. The boot mode selection process is discussed in section 2.1.

- 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 2.3.

- Port-addressed register configuration during boot

  Port-addressed registers (such as those used to control peripherals) can be configured during the bootload, 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 2.3.2.

- Programmable delay during boot

  Programmable delays of up to 65535 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 2.3.2.

1.2 On-Chip ROM Description

On the VC5510, the on-chip ROM contains several factory-programmed sections including:

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

The ROM memory map is shown in Table 1.

Table 1. TMS320VC5510 ROM Memory Map

Starting Byte Address	Contents
FF8000h	Bootloader program
FFFA00h	Sine table
FFFC00h	Factory test code
FFFF00h	Interrupt vector table
FFFFFCh	ID code

2 Bootloader Operation

The sections that follow describe the structure and operation of the production TMS320VC5510 bootloader only. Users migrating to the TMS320VC5510 bootloader from the TMX320VC5510 (prototype) bootloader should also refer to section 3 of this document for specific migration information. For a detailed description of the operation of the prototype VC5510 bootloader, refer to the application report *Using the TMX320VC5510 Prototype Bootloader* (SPRA764).

2.1 Bootloader Initialization

When the VC5510 prototype bootloader begins execution, the program performs some initialization of the VC5510 prior to loading code. The VC5510 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 C54x compatibility mode during and after the bootload.

After the initialization is performed, the bootloader loads the on-chip RAM according to the boot mode selected, and then causes the VC5510 to begin execution of the loaded code. At that point, the bootload process is complete. Whenever the system is reset, the VC5510 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 when the bootloader program begins execution approximately 30 cycles after execution of the reset vector. The BOOTM pins should be maintained in the desired state until at least 30 cycles have passed to properly select the boot mode.

Some of the BOOTM pins are shared with the general-purpose I/O (GPIO) pins:

- BOOTM2 is shared with IO3.
- BOOTM1 is shared with IO2.
- BOOTM0 is shared with IO1.
- BOOTM3 is a dedicated input.

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, users should be aware that this pin will become active as an output during the bootload process and should 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 addition of future boot modes and should not be selected.

Table 3. Boot Mode Selection Options

BOOTM[3:0]	Boot Mode Source	For details, see section …
0000	No boot	2.3.1
0001	Serial EEPROM (SPI) boot from McBSP0 supporting 24-bit address	2.3.5
0010	Reserved	
0011	Reserved	
0100	Reserved	
0101	Reserved	
0110	Reserved	
0111	Reserved	
1000	No boot	2.3.1
1001	Serial EEPROM (SPI) boot from McBSP0 supporting 16-bit address	2.3.1
1010	Parallel EMIF boot from 8-bit external asynchronous memory	2.3.2
1011	Parallel EMIF boot from 16-bit external asynchronous memory	2.3.2
1100	Parallel EMIF boot from 32-bit external asynchronous memory	2.3.2
1101	EHPI	2.3.3
1110	Standard serial boot from McBSP0 (16-bit)	2.3.4
1111	Standard serial boot from McBSP0 (8-bit)	2.3.4

2.3 Boot Mode Options

2.3.1 No Boot Option

When BOOTM[3:0] = 0000b or 1000b at reset, the No Boot option is selected. In this mode, the bootloader program does not execute, and the on-chip ROM is not mapped into the internal memory map. The DSP maps the address space instead to external memory in the CE3 space. When these boot options are selected, the DSP branches to the reset vector in external CE3 memory space and executes the reset vector. For more information on the VC5510 memory map, see the *TMS320VC5510 Fixed-Point DSP Data Manual* (SPRS076).

2.3.2 Parallel EMIF Boot Mode

Parallel EMIF Boot Mode is selected when BOOTM[3:0] = 1010b, 1011b or 1100b after reset. This mode reads the boot table from external asynchronous memory that can be either 8, 16, or 32 bits wide. The 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 the 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 section 2.4.3. 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 (see section 2.4.3). 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 will go 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 will go high and the DSP will begin 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

The description in this section assumes familiarity with the VC5510 EHPI. For detailed information on the TMS320C55x™ EHPI, see the *TMS320C55x DSP Peripherals Reference Guide* (SPRU317) and *Using the TMS320VC5509/5510 Enhanced HPI* (SPRA741).

In EHPI boot mode, an external host can load code and data directly into the DSP memory. The CPU will then execute the loaded code when the host has indicated that the load is complete. EHPI boot does not use a boot table. The code and/or data sections are directly loaded into the desired locations by the host. The host has access to DARAM (above word address 30h), SARAM, and part of the external CE0 space (up to address FFFFFh) for a total of 2M bytes of accessible memory space. For information on the EHPI memory map, see the *TMS320VC5510 Fixed-Point DSP Data Manual* (SPRS076).

After the VC5510 RESET pin is released, the bootloader code will run from ROM and begin to poll the RESET bit in the EHPI control register (HPIC). This bit is low after the RESET pin is released. The host then loads the desired code and data sections into the DSP memory. When the load is complete, the EHPI writes the RESET bit in HPIC to 1. This indication causes the CPU to begin execution at byte address 010000h (word address 008000h) at the beginning of the internal SARAM block. 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 010000h, the EHPI will load the section to word address 008000h.

The general procedure for boot loading using the EHPI is:

- $\overline{\text{RESET}}$ pin is released (low-to-high transition) with BOOTM[3:0] = 1101b.
- The CPU executes the on-chip ROM bootloader and begins to poll the RESET bit in HPIC, to determine when the host load is complete.
- The host loads the desired code and data sections into DSP internal memory within the address limits mentioned above.
- The host indicates the load is complete by setting the RESET bit in the HPIC register.
- The CPU transfers execution to byte address 010000h, and the loaded application begins running.

In the event that the application has been previously loaded and another external reset is necessary (warm boot), the host can simply set the RESET bit in HPIC (after the $\overline{\text{RESET}}$ pin low-to-high transition) without reloading the application code, and the application execution will begin.

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

TMS320C55x is a trademark of Texas Instruments.

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 hpi16.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 hpi16.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 hpi16.mxp          /* map file                                 */
–o hpi16.asc         /* boot table file (output file)               */
```

The -boot option is used in order 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 Multichannel Buffered Serial Port (McBSP). For detailed information on the C55x™ McBSP, refer to 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. This mode provides the ability for a host device to bootload the VC5510 through the serial port.

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 1. The serial port sending data to the DSP must conform to this data format.

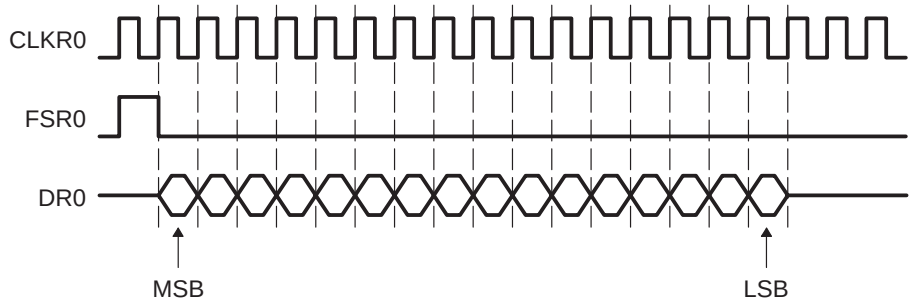


Figure 2. McBSP0 Receive Data Format for Bootload (16-bit shown)

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 in order to insure proper operation:

- The serial port receive clock externally supplied on CLKR0 should not exceed 1/8 the frequency of the VC5510 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 (see sections 2.3.4.1 and 2.3.4.2).

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.

C55x is a trademark of Texas Instruments.

An overflow of the receiver will cause 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 (see section 2.3.4.1) between transmissions to prevent overflow.

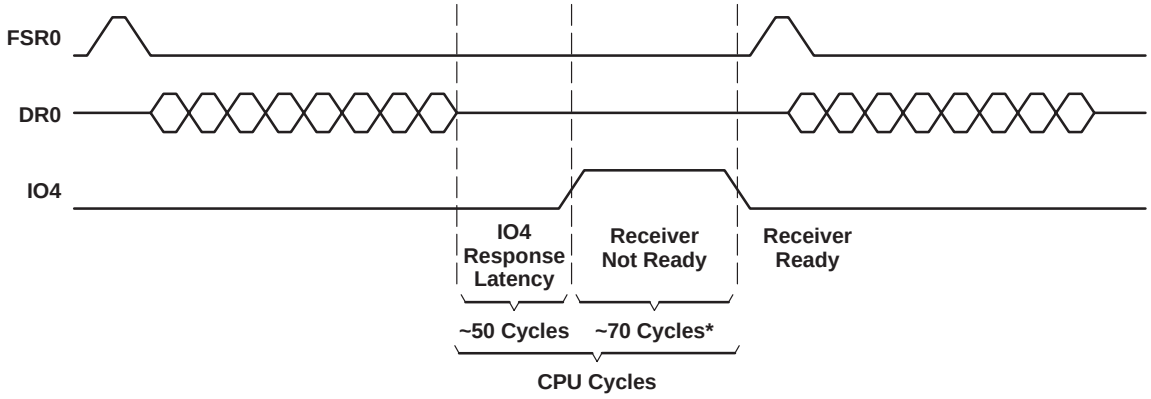
2.3.4.1 Using IO4 to Prevent Receiver Overflow

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 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).



*** Assumes no programmed delays and internal memory destination.**

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) will be necessary for each word to be moved. So the time between transmission of words should be no less than approximately 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 or by slowing down the serial port clock relative to the CPU clock. Since the delay after the reception of each word (or byte) is not the same, the user 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, refer to the *TMS320C55x DSP Peripherals Reference Guide* (literature number SPRU317).

The VC5510 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[3:0] = 1001b. The bootloader supports SPI EEPROMs based on 24-bit byte addresses (up to 16M bytes) as mode BOOTM[3:0] = 0001b.

In SPI EEPROM boot mode, the DSP acts as an SPI master, and the memory acts as the slave. The bootloader code sets the serial port clock to run at a rate of the CPU clock divided by 254. This serial port clock speed should be used to determine the required speed for the EEPROM to be used.

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. 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 ($\overline{CS}$) signal. IO4 will automatically enable the EEPROM when the bootload is ready to begin, and will disable the EEPROM when the bootload is complete.

Some serial EEPROMs may additionally provide write-protect ($\overline{WP}$) and $\overline{HOLD}$ 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 $\overline{HOLD}$ input is used to suspend serial input to the EEPROM. Having this pin active will prevent the bootloader from operating correctly. The $\overline{HOLD}$ pin (if present) should be pulled inactive (high).

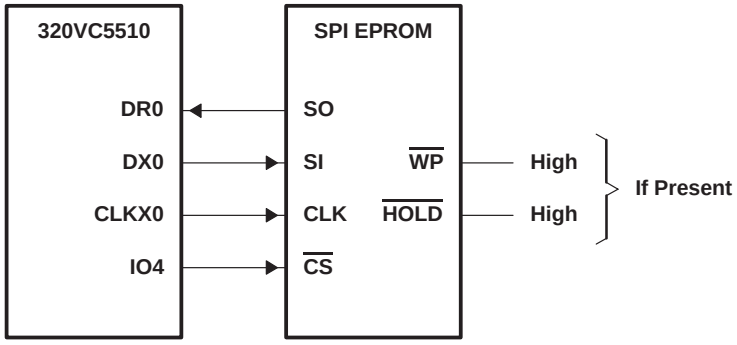


Figure 4. Signal Connections for SPI EEPROM Boot Mode

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.

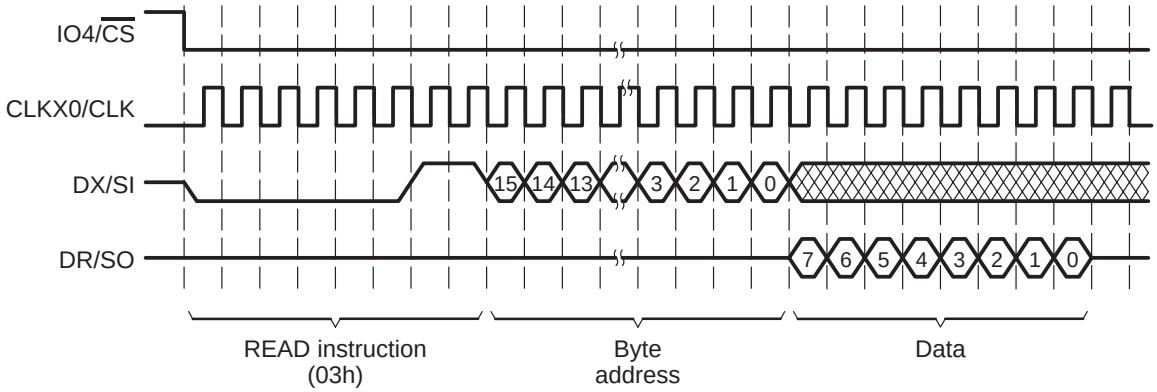


Figure 5. SPI EEPROM Mode Transfer Protocol With 16-Bit Addresses

The process begins with the DSP driving IO4 low (EEPROM $\overline{CS}$). 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.

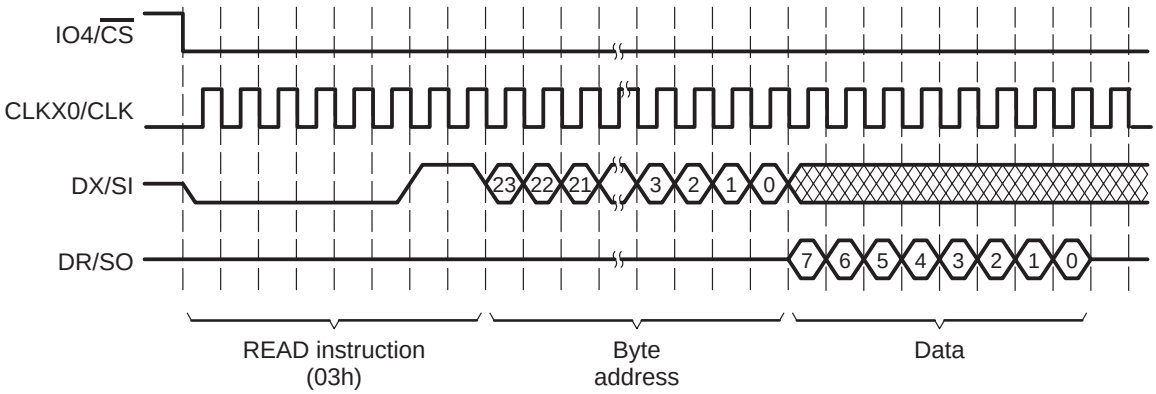


Figure 6. SPI EEPROM Mode Transfer Protocol With 24-Bit Addresses

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

2.4 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 Code Generation Tools*), based on the COFF (common object file format) 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.

2.4.1 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. They 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 at 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 between word addresses 0000h and 0100h (byte address 0000h−0200h).

2.4.2 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 7.

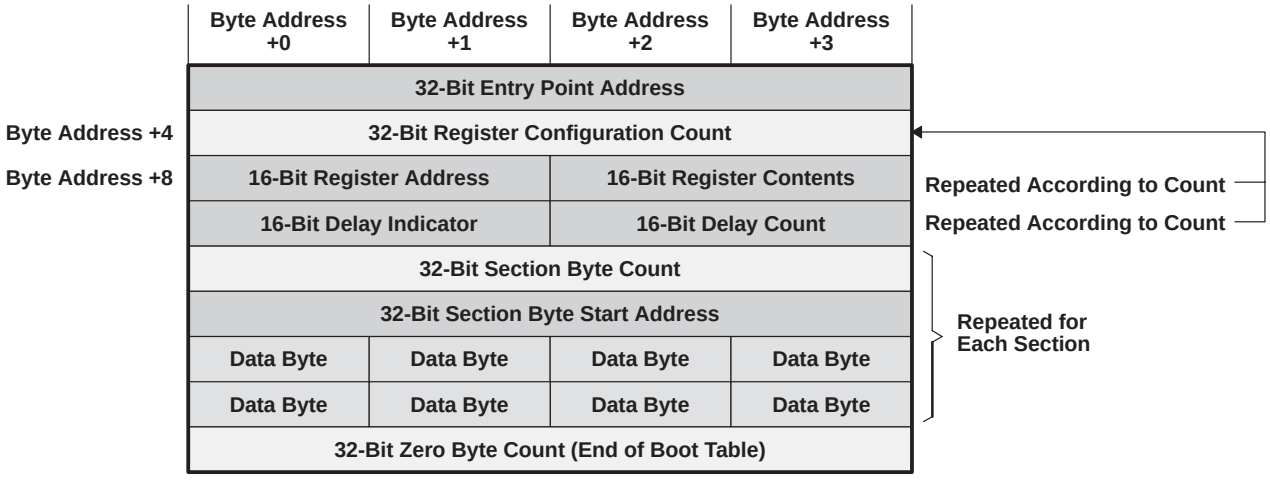


Figure 7. Boot Table Structure

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

- 32-bit Entry Point Byte Address is the address where the bootloader will begin 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 2.4.3). 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.

2.4.3 Register Configuration and Delay During Boot

The VC5510 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.

A register configuration entry will be added to the boot table when the option *–reg_config address, data* is added to the command line in the hex conversion utility when the boot table is created. In this option, *address* is the port address of the register to be configured, and *data* is the data that will be written to the register. For example, to program the VC5510 clock mode register (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 will add 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 will be 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 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 will put 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 will always interpret a reference to port address FFFFh as a request for a delay, and use 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 will be interpreted as register configurations.

Note that the bootloader provides no protection with regard to the programmed register contents specified in these features. It is the responsibility of the user 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.
- 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 datasheet and bootloader requirements. Consider these issues very carefully before making any changes.

The hex-conversion utility will automatically count the number of register configurations and delays specified in the command line (or command file) and will insert this information in the boot table. The register configurations and delays will be 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 will proceed to copying code and data sections.

2.4.4 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 the user. The hex conversion utility will add all initialized sections in the application to the boot table.

In the C55x 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 will pad 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 a user reads 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. To correctly interpret this structure, the bootloader code requires that 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 will be 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 VC5510 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 will continue 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 will branch to the entry-point address (specified at the beginning of the boot table) and begin execution of the application.

2.4.5 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 to indicate to the hex conversion utility that the desired boot table format is for a production (TMS320VC5510) device, not for a prototype (TMX320VC5510) device. This option is very important since the prototype and production bootloaders are different. The wrong boot table format will cause the bootloader to fail.
4. Specify the boot type *–parallel8*, *–parallel16*, *−parallel32*, *−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 to which the bootloader will transfer execution when the boot load is complete.
6. Specify the desired output format. See the *TMS320C55x DSP Assembly Language Tools User's Guide* (SPRU280) for detailed information on the available hex conversion utility output formats.
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
1001	Serial EEPROM (SPI) Boot from McBSP0 supporting 16-bit address	−serial8
1010	External asynchronous memory (8-bit)	−parallel8
1011	External asynchronous memory (16-bit)	−parallel16
1100	External asynchronous memory (32-bit)	−parallel32
1110	Standard serial boot from McBSP0 (16-bit)	−serial16
1111	Standard serial boot from McBSP0 (8-bit)	−serial8

Some examples of how to set the hex conversion utility options to create a boot table are shown in Example 1 and Example 2.

Example 1. Creating a Boot Table for Tektronix Output

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 Tektronix 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 production TMS320VC5510
–parallel16     ;boot mode is 16-bit external asynchronous memory
–x              ;desired output format is Tektronix format
–o my_app.hex   ;specify the output filename
my_app.out      ;specify the input file
```

Example 2. Creating a Boot Table for Intel Output

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 production
                                TMS320VC5510
–serial8                       ;boot mode is 8-bit standard serial boot
–reg_config 0x1c00, 0x2180     ;write 0x2180 to peripheral register at 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 DSP Assembly Language Tools User's Guide* (SPRU280).

3 Migration From the Prototype to the Production Bootloader

This document has described the operation of the production bootloader that exists on VC5510, revision 2.0 and later. VC5510 revision(s) 1.x contain a prototype bootloader that is different from the production bootloader. Several enhancements were included in the production bootloader which were not present in the prototype bootloader. This section explains the differences between the two versions of the bootloader and discusses the required actions to migrate from revision 1.x to 2.0. For a more detailed description of the prototype bootloader, see *Using the TMX320VC5510 Prototype Bootloader* (SPRA764).

3.1 Boot Mode Selection and the BOOTM3 Pin

Several new boot modes were added to the production bootloader. The additional boot mode selection was made possible by the addition of another boot mode select pin, BOOTM3, which was not present on the prototype device. BOOTM3 has an internal pull-up resistor, so boards designed to work with the prototype silicon will still operate with the production silicon, but will be limited to the boot modes where BOOTM3 is high. The available boot modes on both bootloader versions are shown in Table 5.

Table 5. Prototype vs. Production Boot Mode Selection Options

Boot Mode Source	VC5510, Revision 1.x	VC5510, Revision 2.0+
Serial EEPROM (SPI) boot from McBSP0, supporting 24-bit address		✓
Serial EEPROM (SPI) boot from McBSP0, supporting 16-bit address		✓
External asynchronous memory (8-bit)		✓
External asynchronous memory (16-bit)	✓	✓
External asynchronous memory (32-bit)	✓	✓
EHPI	✓	✓
Standard serial boot from McBSP0 (16-bit)	✓	✓
Standard serial boot from McBSP0 (8-bit)	✓	✓

3.2 Boot Table Differences

The boot table on the production bootloader is based on a different format than the prototype bootloader. This change was made to allow for enhanced features on the bootloader and to eliminate some restrictions imposed by the previous format. The hex conversion utility takes these changes into account and automatically provides the correct format for the boot table based on the use of the –v5510:1 option (for the prototype version) or the –v5510:2 option (for the production version).

Users migrating from the prototype to the production bootloader will need to regenerate the boot table using the –v5510:2 option. The new format must be used with the production bootloader.

The primary changes to the boot table format are summarized in Table 6.

Table 6. Boot Table Differences Between the Prototype and Production Bootloaders

VC5510, Revision 1.x	VC5510, Revision 2.0+
Word-addressed: The addresses for the beginning of each section to be loaded are word addresses.	Byte-addressed: The addresses for the beginning of each section to be loaded are byte addresses.
Section word alignment is required. All sections to be loaded through the bootloader must be aligned on word boundaries. Since the hex conversion utility cannot change the addresses specified by the linker, this must be done by the user through the linker.	Section word alignment is not required. The hex utility can deal with program sections that start and end on odd byte addresses without any intervention from the user.
Peripheral register configuration during boot is not supported.	Peripheral register configuration during boot is supported.
Programmable delays during boot are not supported.	Programmable delays during boot are supported.
Section length (in words) is represented in 16 bits, limiting the maximum section size to 64k words (128k bytes).	Section length (in bytes) is represented in 32 bits, allowing sections to exceed 128k bytes (64k words). Sections are only limited by the amount of available memory.
Section destinations must be located in on-chip memory.	Section destinations can be in on-chip or external memory, since the register configuration function can be used to configure the EMIF before any sections are moved.
Section destination addresses cannot span main data page (MDP) boundaries. A single section must lie entirely within a single 64k x16-bit data page.	Sections can start and end at any address. The bootloader automatically handles page boundaries.
The length of the entire boot table cannot exceed 64k words, and must be entirely contained in a single main data page.	The length of the boot table is not limited, and can cross main data page boundaries.
The bootloader leaves the device with TMS320C54x compatibility mode off (C54CM = 0 is ST1_55).	The bootloader leaves the device with TMS320C54x compatibility mode on (C54CM = 1 is ST1_55).

3.3 IO4 Behavior

On the prototype bootloader, IO4 was used only to indicate when the serial port was initially ready to receive data in standard serial boot mode. After the initial ready signal, IO4 stays low for the entire boot process.

On the production bootloader, IO4 is used for multiple functions as indicated below:

- In the standard serial boot modes, IO4 is used to indicate that the serial port is ready to receive data. If a programmed delay occurs, IO4 goes high (not ready) until the delay is completed, and then low (ready) when the serial port is ready to receive again. IO4 also goes high 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 chip select ($\overline{CS}$) signal to the serial EEEPROM. It goes low at the beginning of the boot process, and goes high when the boot process is complete. IO4 does not go low during delays in this mode, but since the DSP is the master, delays are handled automatically.

4 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.

4.1 Parallel EMIF Boot Mode

The IO4 signal will go 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 will branch to the entry point and begin execution of 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, or 1100b for 32-bit 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.

4.2 Host Port Interface Boot Mode

After release from reset, the host begins downloading code to DSP memory. When the bootload process is complete, the CPU will branch to the entry point at byte address 0x10000 and begin code execution.

Check:	If no, then verify that:
Does the host begin loading code to DSP memory upon release from reset?	• BOOTM[3:0] pins are set to 1101b. • The DSP is released from reset with a low-to-high transition of the reset signal.
Does the CPU hit a breakpoint at byte address 0x10000? Use a JTAG emulator and debugger such as Code Composer Studio.	• The host sets the RESET bit in the HPIC register, indicating completion of the bootload process. Since the DSP cannot access HPIC, read the value back from the host.
Does the user application begin to execute properly?	• Byte address 0x10000 (word address 0x8000) contains the start of executable code (not a boot table). The host will load code to the DSP memory by word address, while program fetches from the DSP's point of view byte-addressed. • The host does not try to load code to the area below word address 30h, or to external memory addresses higher than word address 0xFFFFF.

4.3 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 FDR0?	• 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 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.

4.4 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 /HOLD signal is pulled inactive high. • The speed of the EEPROM is compatible with a serial port clock rate of the CPU clock divided by 254.
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.

5 References

1. *Using the TMX320VC5510 Prototype Bootloader* (SPRA764).
2. *TMS320VC5510 Fixed-Point DSP* (SPRS076).
3. *TMS320C55x DSP Peripherals Reference Guide* (SPRU317).
4. *Using the TMS320VC5509/5510 Enhanced HPI* (SPRA741).
5. *TMS320C55x Assembly Language Tools User's Guide* (SPRU280).

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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