Using the TMS320VC5501/C5502 Bootloader

Gustavo Martinez, Tai Nguyen *C5000 Hardware Applications*

ABSTRACT

This document describes the features of the on-chip bootloader provided with the TMS320VC5501 and TMS320VC5502 digital signal processors (DSPs). 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 the different versions of the 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 main bootloader code may change. Users should avoid calling these functions directly, since the code may change in the future. To call boot mode functions from within the main application code, users should take advantage of the boot mode branch table included in the C5501 and C5502 ROM. The boot mode branch table contains jumps to all the boot modes supported by both devices. The location of this table has been selected such that it will not change across revisions of the bootloader code. See section 2.5 for more information.

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 TMS320VC5501 and TMS320VC5502 digital signal processors (DSPs). The ROM contents, including the on-chip bootloader, of both the C5502 and C5501 are the same. However, due to the differences between the C5502 and the C5501, special considerations must be taken into account when using the on-chip bootloader on either device. Descriptions of all these special considerations are included in this section.

Unless otherwise specified, all references in this document to C5502 refer to both the TMX320VC5502 and TMS320VC5502 and all references to C5501 refer to both the TMX320VC5501 and the TMS320VC5501.

All references to bootloader in this document apply to both the C5502 and the C5501 unless otherwise noted.

1.1 Bootloader Features

The bootloader is DSP code that transfers application code from an external source into internal or external program memory after the DSP is taken out of reset. The bootloader allows application code to reside in slow non-volatile external memory and be transferred to high-speed internal memory for execution. The bootloader is permanently stored in the ROM of the DSP starting at byte address 0xFF8000.

The application code is *encoded* in a special format that the bootloader understands; this special format is called a boot table. 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 boot table format will be discussed later in this document.

To accommodate different system requirements, the bootloader offers a variety of ways (boot modes) to transfer a boot table into internal memory. The following is a list of the available boot modes and a summary of their functional operation:

- Boot from the Host Port Interface (HPI)

  In this mode, the bootloader waits until the code to be executed is loaded into on-chip memory by a host device via the HPI. Code execution begins when the host indicates to the bootloader that the application has been loaded. No boot table is used in this mode. The operation of this mode is described in section 2.3.2.

- Parallel EMIF boot from 16-bit external asynchronous memory

  The bootloader reads the boot table from external 16-bit asynchronous memory via the external memory interface (EMIF). The bootloader will configure the CE1 space for 16-bit, asynchronous operation and read the boot table from word address 200000h. The operation of this mode is described in section 2.3.3.

- Standard 16-bit serial boot through McBSP0

  The bootloader receives the boot table from a host device via 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 EEPROM serial boot though McBSP0

  The bootloader uses McBSP0 to load the boot table from an SPI-format serial EEPROM. In this mode the bootloader configures McBSP0 to operate in SPI mode. The data can be received from an SPI EEPROM or from an SPI-compliant serial port. The bootloader only supports SPI devices based on 24-bit addresses. The operation of this mode is described in section 2.3.5.

- I2C EEPROM boot through I2C port

  The bootloader reads the boot table into on-chip memory via the I2C interface from an I2C EEPROM(s) or a device operating as an I2C slave. The bootloader only supports I2C devices which use 2 bytes for internal addressing of their memory space. The operation of this mode is described in section 2.3.6.

- UART boot

  The bootloader receives the boot table from a host device via the UART interface and loads the code according to the information specified in the boot table. The operation of this mode is described in section 2.3.7.

- Direct Execution from External Asynchronous Memory − No Boot

  The C5501 and C5502 also offer two no-boot options in which the DSP executes code directly from external asynchronous memory. When these options are used, the ROM is disabled (MPNMC = 1) at reset, effectively forcing the first instruction to be fetched from external memory at address 0xFFFF00. The C5501 and C5502 support direct execution from 16-bit or 32-bit external asynchronous memory. The operation of this mode is described in section 2.3.1.

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 2.2.

- 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 2.6.

- 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 2.6.3.

- 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 2.6.3.

- Boot mode branch table

  A section in ROM space has been reserved which contain hard-coded branches to each boot mode. The boot mode branch table can be used to execute bootloader code after the application code has been loaded (warm-boot). Future spins of the bootloader code could change the starting point ROM address for each boot mode; however, the location of this table is not likely to change and can be used to call boot modes if so desired. This capability if discussed further in section 2.5.

1.2 On-Chip ROM Description

The C5501 and C5502 on-chip ROM contain several factory-programmed sections including:

- Bootloader program
- A boot mode branch table containing branches to each boot mode
- Sine look-up table consisting of 256 signed Q15 integers representing 360°
- Interrupt vector table

The ROM memory map is shown in Table 1.

Table 1. TMS320VC5501/55022 ROM Memory Map

Starting Byte Address	Contents
FF_8000h	Bootloader program
FF_ECAEh	Bootloader Revision Number
FF_ECB0h	Boot Mode Branch Table
FF_ED00h	Sine Table
FF_EF00h	Reserved
FF_FF00h	Interrupt Vector Table

1.3 Special Considerations for the C5501 and C5502

The user must take some special considerations when using the bootloader on the C5502 and C5501 since these devices have some differences when compared to other C55x DSPs.

1.3.1 Pin Multiplexing

On the C5502, the EMIF and the HPI share pins to reduce pin count. The function of the shared pins, or multiplexed pins as they are referred to in the data manual, is determined at reset via the GPIO6 pin. If GPI06 is low, the EMIF will be disabled and the HPI will use the EMIF address and data pins for operation (HPI operates in non-multiplexed mode). If GPIO6 is high, the EMIF will be enabled and the HPI will operate through non-EMIF pins (HPI operates in multiplexed mode).

There is no sharing of pins between the EMIF and HPI on the C5501, however, the GPIO6 pin must be kept high at reset to enable both of these modules. If GPIO6 is low at reset, the C5501 will configure the EMIF and HPI pins as parallel general-purpose input and output pins and the EMIF and HPI will not be available for use.

The UART and McBSP2 also share pins on the C5502. The state of the GPIO7 pin at reset determines which peripheral has control of the shared pins. If GPIO7 is low during reset, the UART will have control of the shared pins. If GPIO7 is high, McBSP2 will have control of the shared pins and the UART will be disabled. The UART on the C5501 does not share pins with any other peripheral.

The GPIO configuration of the device being used affects boot modes that use the EMIF, HPI, and UART. The user must be sure to control the state of the GPIO6 and GPIO7 pins at reset such that the peripheral that is being used for boot loading is enabled and its pins are available. For example, if the UART boot mode is used on the C5502, the GPIO7 pin must be pulled low at reset.

Also, keep in mind that it is possible to load application code to external memory during boot. In cases like these, the EMIF must be enabled and programmed (via the register configuration approach discussed in section 2.6.3) so that the bootloader can load the code to external memory. The EMIF is enabled when GPIO6 is high during reset on both the C5501 and C5502.

Tables listing the required GPIO pin configuration for each bootmode are included in the sections that describe the operation of the boot options supported by these two devices.

For more information on the pin multiplexing of the C5501 and C5502 refer to the *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206) or the *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166).

1.3.2 Clock Groups

The C5501 and C5502 have several clock groups which allow for specific sections of the DSP to run at different frequencies. The clock groups are as follows:

- C55x Subsystem Clock Group

  The C55x Subsystem Clock Group includes the C55x CPU core, internal memory (DARAM and ROM), the Instruction Cache, and all CPU related modules. At reset, this clock group has a clock frequency equal to that of the input reference clock (CLKIN). All references to CPU clock within this document refer to the clock of the C55x Subsystem Clock Group.

- Fast Peripherals Clock Group

  The DMA, HPI, and the timers are included in the Fast Peripherals Clock Group. At reset, this clock group has a frequency equal to 1/4th of the CPU clock frequency.

- Slow Peripheral Clock Group

  The Slow Peripherals Clock Group includes the McBSPs, I2C, and the UART. The input clock to this clock group is equal to 1/4th of the CPU clock.

- External Memory Interface Clock Group

  The EMIF is the only module that falls under the External Memory Interface Clock Group. At reset, this clock group has a clock frequency equal to 1/4th of the CPU clock.

The reset state of these clock groups means that the peripherals of the C5501 and C5502 will not be running at the same frequency as the CPU. For example, if a 20-MHz input reference clock (CLKIN) is supplied, the C55x core will be running at 20 MHz and all the peripherals will be running at 5 MHz. The clock ratios between these clock groups can be changed after boot load is complete as long as the following equations are not violated:

$$f_{C55x\ Subsystem} \geq f_{Fast\ Peripheral\ Subsystem}$$

$$f_{Fast\ Peripheral\ Subsystem} \geq f_{Slow\ Peripheral\ Subsystem}$$

$$f_{Fast\ Peripheral\ Subsystem} \geq f_{External\ Memory\ Interface\ Subsystem}$$

It is not recommended to change clock ratios between the clock groups during the boot load process. Furthermore, it is recommended to place all peripherals in idle mode before changing the frequency of their respective clock group.

Refer to the *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206) or the *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166) for more information on the clock generator and the clock groups of these devices.

1.3.3 Internal Oscillator and Stabilization Periods

The C5501 and C5502 include an internal oscillator which can be used, in conjunction with an external crystal resonator, to generate the input clock source. If the internal oscillator is not used, an external clock source must be applied to the X2/CLKIN pin.

The clock source (internal oscillator output or external clock) for the device is selected at reset based on the state of the GPIO4 pin. If GPIO4 is low when the reset signal transitions from low to high, the input clock source will be taken from the internal oscillator plus external crystal combination (referred to as OSCOUT in the data manual). If GPIO4 is high during the transition of the reset signal from high to low, the input clock source will be taken directly from the X2/CLKIN pin.

The stabilization periods that are added after reset allow the reset signal to properly propagate through the DSP and to allow the internal oscillator (if used) to stabilize. The state of the GPIO4 pin affects the duration of these stabilization periods as shown in Table 2. After reset, the bootloader will begin executing after the stabilization periods have expired.

Table 2. Stabilization Period Duration

GPIO4 State During Reset	Stabilization Period (Input Clock Cycles)
GPIO4 = 1	70
GPIO4 = 0	41,102

The GPIO4 pin is configured as an output and driven high when the bootloader starts executing. Furthermore, some boot modes use the GPIO4 pin for multiple functions, such as signaling external hosts when the DSP is ready to receive data. The use of an external pull−up/pulldown resistor on the GPIO4 pin will set the state of the internal oscillator at reset and also allow the bootloader to drive the GPIO4 pin. If an external host is controlling the state of the GPIO4 pin, the host will have to stop driving the GPIO4 before the stabilization period has expired.

2 Bootloader Operation

The sections that follow describe the structure and operation of the C5501 and C5502 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 3.

Table 3. Bootloader Initialization

Resource	Initialization Value
Stack Registers:	The data stack register (SP) is initialized to word address 000090h, and the system stack register (SSP) is initialized to word address 000080h. Application code should start at word address 00090h or higher.
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 boot load.

After the initialization is performed, the bootloader loads the on-chip RAM according to the boot mode selected, and then causes the DSP to begin execution of the loaded code. At that point, the boot load process is complete. Whenever the system is reset, the CPU starts execution of the bootloader again, and the entire boot load 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 three boot mode select pins BOOTM[2:0], which are sampled during reset. The BOOTM pins are shared with the general-purpose I/O (GPIO) pins.

- BOOTM2 is shared with GPIO2.
- BOOTM1 is shared with GPIO1.
- BOOTM0 is shared with GPIO0.

The available boot mode options and their corresponding BOOTM pin configurations are shown in Table 4.

Table 4. Boot Mode Selection Options

BOOTM2 (GPIO2)	BOOTM1 (GPIO1)	BOOTM0 (GPIO0)	Boot Mode Source	For details, see section
0	0	0	No boot, execute from 16-bit external asynchronous memory (micro-processor mode)	2.3.1
0	0	1	Serial EEPROM (SPI) boot from McBSP0†	2.3.5
0	1	0	Standard serial boot from McBSP0	2.3.4
0	1	1	EMIF boot from 16-bit external asynchronous memory	2.3.3
1	0	0	No boot, execute from 32-bit external asynchronous memory (micro-processor mode)	2.3.1
1	0	1	HPI boot in 8-bit multiplexed or 16-bit non-multiplexed mode‡	2.3.2
1	1	0	I2C slave boot from I2C EEPROM or I2C slave	2.3.6
1	1	1	UART boot	2.3.7

† The initial version of the bootloader does not support the serial EEPROM boot mode.
‡ The C5501 does not support 16-bit non-multiplexed mode for the HPI.

The BOOTM pins are not sampled immediately after the RESET pins transitions from low to high. Instead, the BOOTM pins are sampled after a certain number of clock periods have passed. The number of clock periods depends on whether the internal oscillator is being used. The state of the GPIO4 pin during reset selects either the internal oscillator or an external clock as the clock source for the DSP. When the internal oscillator is used as the clock source, the BOOTM pins are sampled 41,102 input reference clock cycles (CLKIN) after the DSP is taken out of reset. If the internal oscillator is not used, the BOOTM pins are sampled after 70 input reference clock cycles. For simplicity, this document assumes that the BOOTM pins are sampled at reset. Refer to the *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206) and the *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166) for more information on the reset sequence of these devices.

The GPIO4 pin 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 boot load process and should design accordingly. After the boot load is complete, the loaded application may change the function of GPIO0, GPIO1, GPIO2 and GPIO4 pins.

2.3 Boot Mode Options

2.3.1 Direct Execution from External Asynchronous Memory − No Boot

When BOOTM[2:0] = 000b or 100b 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 by setting the MPNMC bit to 1. When these boot options are selected, the DSP configures CE3 space for 16- or 32-bit asynchronous memory (depending on the boot option selected), then branches to the reset vector in external CE3 memory space (byte address location 0xFFFF00) and executes the reset vector. For more information on the C5502 memory map, refer to the *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206) and the *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166).

On the C5502, the EMIF pins and the HPI pins are shared to reduce pin count. The function of the shared pins is determined at reset via the GPIO6 pin. If GPI06 is low, the EMIF is disabled and the HPI uses the EMIF address and data pins. If GPIO6 is high, the EMIF is enabled and the HPI operates through non-EMIF pins.

On the C5501, the EMIF and HPI are enabled when GPIO6 is high during reset. On the other hand, the EMIF and HPI are disabled and their pins are configured as parallel general-purpose input and output pins when GPIO6 is low during reset.

To use the no-boot options on these devices the GPIO6 pin must be high at reset, otherwise, the EMIF will be disabled and the CPU will not be able to read the reset vector in external CE3 memory space.

Table 5 and Table 6 list the GPIO pin configuration for both devices that must be implemented during reset to use the no-boot options.

Table 5. C5501 GPIO Pin Configuration for No-Boot Options

BOOTM[2:0]	GPIO4	GPIO6	GPIO7†	Summary
000	0	1	0	CE3 memory space is configured to 16-bit asynchronous memory. Internal oscillator provides clock source. EMIF and HPI are enabled (HPI operates in multiplexed mode).
000	1	1	0	CE3 memory space is configured to 16-bit asynchronous memory. External clock provides clock source. EMIF and HPI are enabled (HPI operates in multiplexed mode).
100	0	1	0	CE3 memory space is configured to 32-bit asynchronous memory. Internal oscillator provides clock source. EMIF and HPI are enabled (HPI operates in multiplexed mode).
100	1	1	0	CE3 memory space is configured to 32-bit asynchronous memory. External clock provides clock source. EMIF and HPI are enabled (HPI operates in multiplexed mode).

† On the C5501, GPIO7 must be kept low during reset.

Table 6. C5502 GPIO Pin Configuration for No-Boot Options

BOOTM[2:0]	GPIO4	GPIO6	GPIO7†	Summary
000	0	1	X	CE3 memory space is configured to 16-bit asynchronous memory. Internal oscillator provides clock source. HPI/EMIF shared pins have EMIF function (HPI operates in multiplexed mode).
000	1	1	X	CE3 memory space is configured to 16-bit asynchronous memory. Internal oscillator provides clock source. HPI/EMIF shared pins have EMIF function (HPI operates in multiplexed mode).
100	0	1	X	CE3 memory space is configured to 32-bit asynchronous memory. Internal oscillator provides clock source. HPI/EMIF shared pins have EMIF function (HPI operates in multiplexed mode).
100	1	1	X	CE3 memory space is configured to 32-bit asynchronous memory. Internal oscillator provides clock source. HPI/EMIF shared pins have EMIF function (HPI operates in multiplexed mode).

† GPIO7 determines whether the UART or McBSP2 is enabled after reset. The UART is enabled, and McBSP2 is disabled, if GPIO7 is low during reset. McBSP2 is enabled, and the UART is disabled, if GPIO7 is high during reset. The state of GPIO7 during reset should be selected based on the needs of the application running on the DSP.

The *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206) and the *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166) contain more information on the pin configurations for both devices.

2.3.2 HPI Boot Mode

The description in this section assumes familiarity with the C5501/5502 HPI. For detailed information on the C5501/5502 HPI refer to the *TMS320VC5501/5502 DSP Host Port Interface (HPI) Reference Guide* (SPRU620).

In HPI boot mode, an external host can load code and data directly into the DSP memory while the CPU waits. HPI 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 HPI has finished loading the application, it signals the CPU to begin execution and the CPU begins executing at the specified entry point.

The host has access to the DARAM of the C5501 and C5502, excluding the memory-mapped registers. Note that the bootloader uses word address 000090h and below for stack operation, therefore it is recommended that all code loaded through an HPI boot start at word address 000090h or higher. Refer to the *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206) and the *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166) for specific information on the starting and ending addresses of the DARAM space of each device.

The C5502 supports HPI data transfers in multiplexed or non-multiplexed modes. The mode of the HPI is determined at reset by the state of the GPIO6 pin. HPI is configured to 8-bit multiplexed mode GPIO6 = 1 at reset. When GPIO6 = 0 at reset, the HPI is configured to 16-bit non-multiplexed mode.

The C5501 only supports HPI data transfers in multiplexed mode. Furthermore, the HPI is disabled when the GPIO6 pin is low at reset. Therefore, to use this boot mode, the GPIO6 pin must be high at reset.

Table 7 and Table 8 list the GPIO pin configuration that must be implemented during reset to use the HPI boot mode.

Table 7. C5501 GPIO Pin Configuration for HPI Boot Mode

BOOTM[2:0]	GPIO4	GPIO6	GPIO7†	Summary
101	0	1	0	Internal oscillator provides clock source. GPIO4 goes low to signal DSP is ready for data after 41,250 input clock cycles. EMIF and HPI are enabled (HPI operates in multiplexed mode).
101	1	1	0	External clock provides clock source. GPIO4 goes low to signal DSP is ready for data after 325 input clock cycles. EMIF and HPI are enabled (HPI operates in multiplexed mode).

† On the C5501, GPIO7 must be kept low during reset.

Table 8. C5502 GPIO Pin Configuration for HPI Boot Mode

BOOTM[2:0]	GPIO4	GPIO6	GPIO7†	Summary
101	0	0	X	Internal oscillator provides clock source. GPIO4 goes low to signal DSP is ready for data after 41,250 input clock cycles. HPI operates in non-multiplexed mode (HPI/EMIF shared pins have HPI function and EMIF is disabled).
101	0	1	X	Internal oscillator provides clock source. GPIO4 goes low to signal DSP is ready for data after 41,250 input clock cycles. HPI operates in multiplexed mode (HPI/EMIF shared pins have EMIF function and HPI operates in multiplexed mode using HD bus).
101	1	0	X	External clock provides clock source. GPIO4 goes low to signal DSP is ready for data after 325 input clock cycles. HPI operates in non-multiplexed mode (HPI/EMIF shared pins have HPI function and EMIF is disabled).
101	1	1	X	External clock provides clock source. GPIO4 goes low to signal DSP is ready for data after 325 input clock cycles. HPI operates in multiplexed mode (HPI/EMIF shared pins have EMIF function and HPI operates in multiplexed mode using HD bus).

† GPIO7 determines whether the UART or McBSP2 is enabled after reset. The UART is enabled and McBSP2 is disabled if GPIO7 is low during reset. McBSP2 is enabled and the UART is disabled if GPIO7 is high during reset. The state of GPIO7 during reset should be selected based on the needs of the application running on the DSP.

The GPIO4 is used by the bootloader to signal the host it is ready to receive data from the host. When the bootloader begins execution, it will set GPIO4 as an output and drive the pin high. The bootloader will set the GPIO4 pin low when it is ready to receive data from the host. The number of input clock cycles that it takes for the DSP to be ready for data depends on the use of the internal oscillator. If the internal oscillator is used as a clock source for the DSP, GPIO4 will go low approximately 41,250 input clock cycles after reset is released. If the internal oscillator is not used as a clock source, the DSP will be ready to receive data approximately 325 input clock cycles after reset is released. The host can monitor the GPIO4 to determine when to start transferring data to the DSP or it can wait the necessary number of clock cycles before commencing the data transfer.

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 below. 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 zeros. This allows the HPI 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 below. 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 HPI 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 HPI will load the section to word address 1000h.

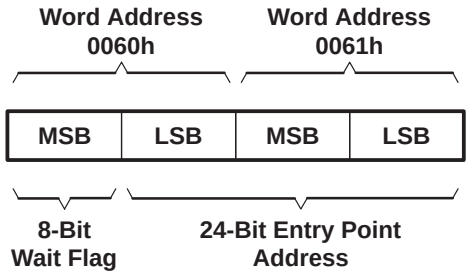


Figure 1. HPI Wait Flag and Entry Point Address

Since the CPU will transfer 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 HPI is:

- The $\overline{\text{RESET}}$ pin is released (low-to-high transition) with BOOTM[2:0] = 101b selecting the desired mode, GPIO6 selecting the desired HPI mode (the GPIO6 pin must be high on the C5501), and GPIO4 selecting the state of the internal oscillator.
- The CPU executes the on-chip ROM bootloader and begins to poll word address 0060h to determine when the host load is complete. The time it takes for the bootloader to begin polling is dependant on the state of the internal oscillator as described above.
- 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 will then transfer execution (branch) to the previously specified entry point address and begin 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 (GPIO4 goes low).

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

More detailed information on the C5501/C5502 HPI can be obtained from the *TMS320VC5501/5502 DSP Host Port Interface (HPI) Reference Guide* (SPRU620). For more information on the pin multiplexing of the C5501 and C5502 ,refer to the *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206) or the *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166).

2.3.2.1 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.

2.3.3 Parallel EMIF Boot Mode

Parallel EMIF Boot Mode is selected when BOOTM[2:0] = 011b at reset. This mode reads the boot table from external asynchronous memory that is 16-bit wide. The data width is configured at reset by the bootloader 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 7 cycles (111b).

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 (85 cycles per access), these EMIF timings can be modified using the port-addressed register configuration feature discussed in section 2.6.3. 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 *TMS320VC5501/5502 DSP External Memory Interface (EMIF) Reference Guide* (SPRU621).

Be aware that changing the timing parameters on the EMIF during the boot process can cause the boot load 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 the CE1_1 EMIF register, 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 EMIF clock 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, GPIO4 will go low at the beginning of the boot process. GPIO4 will go high during execution of the programmable delay feature in the boot table. When the delay is completed, GPIO4 will go low again. At the end of the boot load, GPIO4 will go high and the DSP will begin execution at the entry point address. GPIO4 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 boot load, the DSP will stall until ARDY is high (ready) again. If the target system does not drive ARDY, it should be pulled high.

On the C5502, the EMIF pins and the HPI pins are multiplexed to reduce pin count. The function of the multiplexed pins is determined at reset via the GPIO6 pin. If GPI06 is low, the EMIF will be disabled and the HPI will use the EMIF address and data pins for operation. If GPIO6 is high, the EMIF will be enabled and the HPI will operate through non-EMIF pins.

On the C5501, the EMIF and HPI are enabled when GPIO6 is high during reset. On the other hand, the EMIF and HPI are disabled and their pins are configured as parallel general-purpose input and output pins when GPIO6 is low during reset.

To use the parallel EMIF boot mode on these devices, the GPIO6 pin must be high at reset, otherwise, the EMIF will be disabled and the bootloader will not be able to read the boot table from external memory.

Table 9 and Table 10 list the GPIO pin configuration that must be implemented during reset to use the EMIF boot mode.

Table 9. C5501 GPIO Pin Configuration for Parallel EMIF Boot Mode

BOOTM[2:0]	GPIO4	GPIO6	GPIO7†	Summary
011	0	1	0	The bootloader configures CE1 memory space for 16-bit asynchronous memory. Internal oscillator provides clock source. EMIF and HPI are enabled (HPI operates in multiplexed mode).
011	1	1	0	The bootloader configures CE1 memory space for 16-bit asynchronous memory. External clock provides clock source. EMIF and HPI are enabled (HPI operates in multiplexed mode).

† On the C5501, GPIO7 must be kept low during reset.

Table 10. C5502 GPIO Pin Configuration for Parallel EMIF Boot Mode

BOOTM[2:0]	GPIO4	GPIO6	GPIO7†	Summary
011	0	1	X	The bootloader configures CE1 memory space for 16-bit asynchronous memory. Internal oscillator provides clock source. HPI/EMIF shared pins have EMIF function (HPI operates in multiplexed mode).
011	1	1	X	The bootloader configures CE1 memory space for 16-bit asynchronous memory. External clock provides clock source. HPI/EMIF shared pins have EMIF function (HPI operates in multiplexed mode).

† GPIO7 determines whether the UART or McBSP2 is enabled after reset. The UART is enabled and McBSP2 is disabled if GPIO7 is low during reset. McBSP2 is enabled and the UART is disabled if GPIO7 is high during reset. The state of GPIO7 during reset should be selected based on the needs of the application running on the DSP.

More detailed information on the C5502 EMIF can be obtained from the *TMS320VC5501/5502 DSP External Memory Interface (EMIF) Reference Guide* (SPRU621). The *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206) and the *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166) also contain more information on the pin multiplexing of these devices.

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 C5501/C5502 McBSP, refer to the *TMS320VC5501/5502/5509/5510 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide* (SPRU592).

Standard serial boot mode loads the boot table from McBSP0 in 16-bit mode when BOOTM[2:0] = 010. 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 16 bits (RWDLEN1 = 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. The serial port sending data to the DSP must conform to this data format.

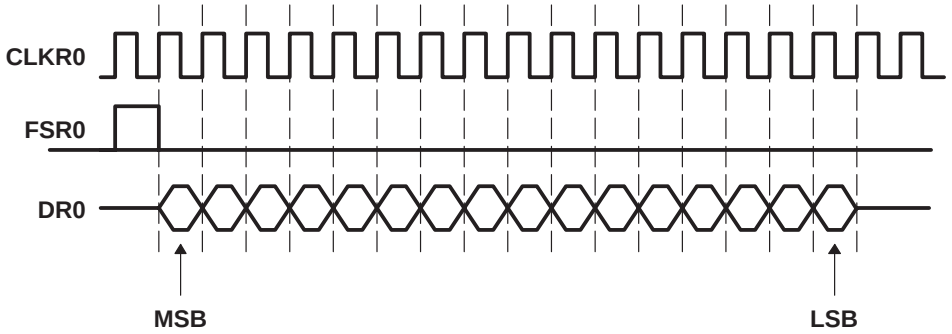


Figure 2. McBSP0 Receive Data Format for Boot Load

McBSP0 falls under the Slow Peripheral Clock Group which, at reset, runs at one-fourth the input clock frequency to the DSP. It is not recommended to change the frequency of the Slow Peripheral Clock Group during the boot process.

The following conditions must be met in order to ensure proper operation of this boot mode:

- The serial port receive clock externally supplied on CLKR0 should not exceed the frequency of the Slow Peripheral Clock Group.
- Appropriate delay should be provided between the transmission of each 16-bit 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.

The pin multiplexing of the C5501 and C5502 only effects this boot mode if the destination of the loaded application is external memory. In cases like these, the DSP being used must be configured such that the EMIF is enabled (refer to section 1.3.1 for more information).

When standard serial boot mode is selected, the bootloader configures McBSP0 as described above and then drives GPIO4 low to indicate to the sender that the DSP is ready to receive (approximately 500 input reference clock cycles after the DSP is taken out of reset if GPIO4 is high during reset or approximately 41,450 input reference clock cycles if GPIO4 is low during reset). One frame sync pulse is associated with each word exchanged.

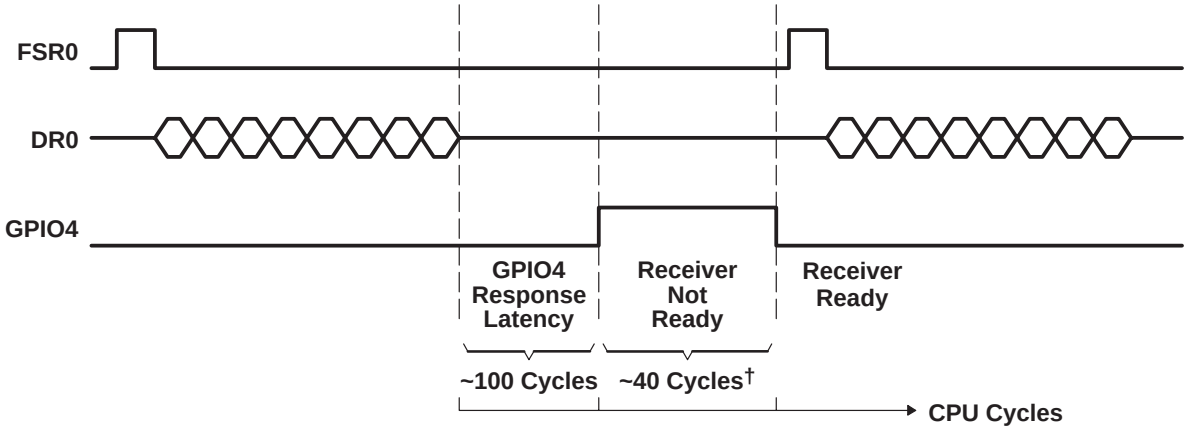
As the sender provides the words of the boot table to McBSP0, GPIO4 responds as a handshaking signal to indicate the state of the boot. When the serial port is ready to receive another 16-bit word, GPIO4 goes low. When the serial port is in the process of copying a received word to memory or when a programmed delay is in progress, GPIO4 is high and only goes low again when the serial port is ready to receive another 16-bit word.

An overflow of the receiver will cause the boot load to fail. There are two basic options for managing the rate of words sent to the serial port to prevent overflow: Use GPIO4 as a handshaking signal or allow sufficient time between transmissions to prevent overflow.

2.3.4.1 Using GPIO4 to Prevent Receiver Overflow

As mentioned previously, GPIO4 goes low when the receiver is ready to receive a 16-bit 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. Note that the use of GPIO4 as a handshaking signal is different on the C5501/C5502 bootloader from that of other C55x bootloaders.

There is some latency in the response of GPIO4 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, GPIO4 will respond to indicate the delay is in progress approximately 100 CPU cycles after last bit of the word was received. This latency is accounted for automatically if the serial port clock is operated at 1/4 of the CPU clock frequency or slower.



† Assumes no programmed delays and internal memory destination.

Figure 3. GPIO4 Latency for Boot-Table-Programmed Delays

Polling GPIO4 provides an automatic method to account for delays in the boot load process due to programmed delays or access delays associated with the EMIF (such as programmed strobe timings or ARDY delays).

2.3.4.2 Preventing Receiver Overflow Without Polling GPIO4

If GPIO4 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 140 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.

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, the user must select a word (or byte) rate that accommodates the worst case delay.

When the end of the boot table is received, GPIO4 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 C5501/C5502 McBSP, refer to the *TMS320VC5501/5502/5509/5510 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide* (SPRU592).

The 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 24-bit byte addresses (up to 16M bytes) as mode BOOTM = 001b. Note that the initial version of the bootloader does not support this boot mode (see section 3 for more details on the different versions of the bootloader).

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. GPIO4 is used to operate the EEPROM chip select ($\overline{CS}$) signal. GPIO4 will automatically enable the EEPROM when the boot load is ready to begin and will disable the EEPROM when the boot load 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).

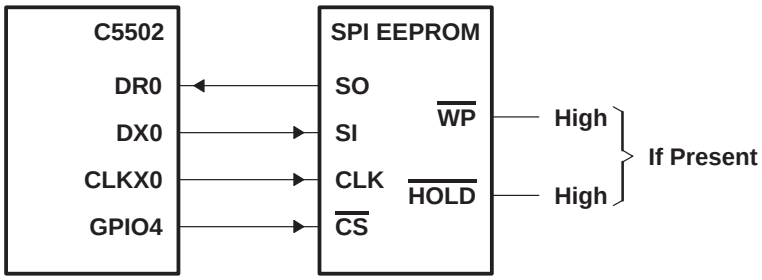


Figure 4. Signal Connections for SPI EEPROM Boot Mode

The pin multiplexing of the C5501 and C5502 only affects this boot mode if the destination of the loaded application is external memory. In cases like these, the DSP being used must be configured such that the EMIF is enabled (refer to section 1.3.1 for more information).

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 Slow Peripheral Clock Group divided by 4. The serial port clock speed can be calculated through the following formula:

$$Serial\ Port\ Clock = Slow\ Peripheral\ Clock/4 = (CLKIN/4)/4$$

This serial port clock speed should be used to determine the required speed for the EEPROM to be used.

The bootloader reads the boot table from the EEPROM as a sequential block of data. It does not perform random accesses. In the 24-bit SPI EEPROM mode, the format of the beginning of the transfer is shown in Figure 5.

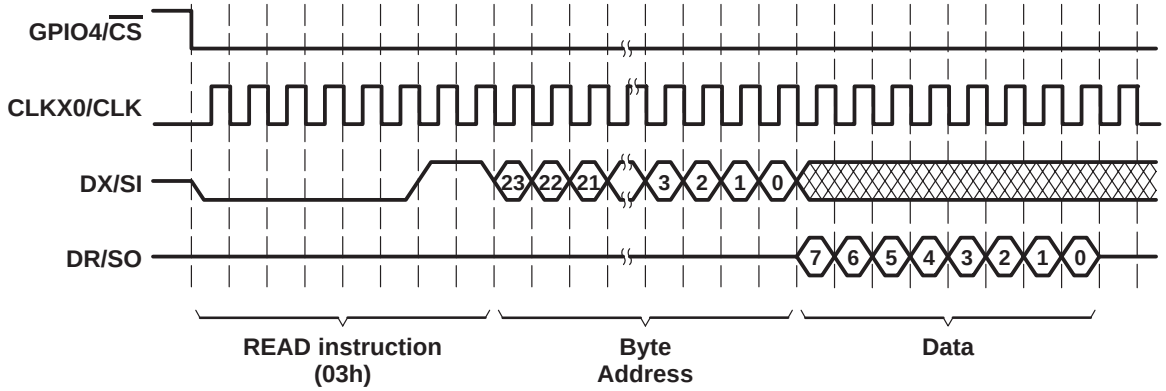


Figure 5. SPI EEPROM Mode Transfer Protocol with 24-bit Addresses

The process begins with the DSP driving GPIO4 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 GPIO4 high to disable the EEPROM chip select and the bootloader branches to the beginning of the loaded application to begin execution.

The boot table must be programmed into the EEPROM as a single continuous image starting at EEPROM address zero.

Although Figure 5 shows 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 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.

2.3.6 I2C EPROM 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 C5501/C5502 I2C refer to the *TMS320VC5501/5502/5509 DSP Inter-Integrated (I2C) Module Reference Guide* (SPRU146).

The bootloader supports boot from I2C EEPROMs or devices operating as I2C slaves that emulate the appropriate format. The bootloader has the following requirements for the I2C EEPROMs:

- The memory device complies with Philips I2C Bus Specification v 2.1.
- The memory device uses two 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 minimum connection required between DSP and one I2C EEPROM is shown in Figure 6. Be sure to place the required pull-ups on SDA and SCL to make the I2C EEPROM interface work properly.

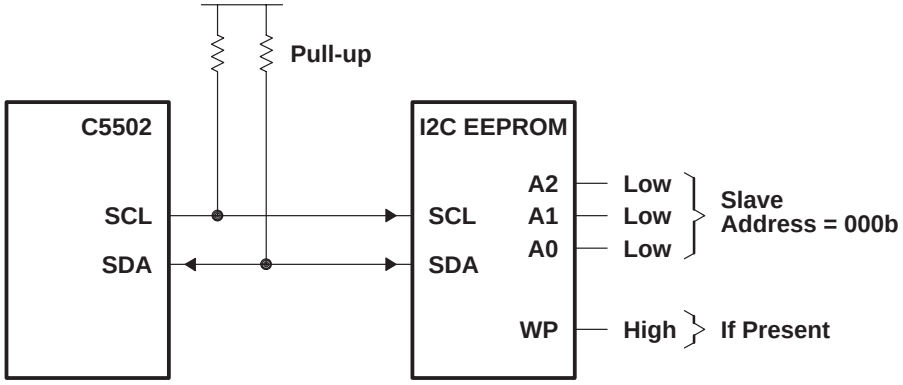


Figure 6. Signal Connections for I2C EEPROM Boot Mode

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

The pin multiplexing of the C5501 and C5502 only affects this bootmode if the destination of the loaded application is external memory. In cases like these, the DSP being used must be configured such that the EMIF is enabled (refer to section 1.3.1 for more information).

The bootloader supports up to eight I2C EEPROMs connected to the I2C bus as long as the each EEPROM has a unique slave address and the memory space made up by all the EEPROMs combined is contiguous. The bootloader will start with slave address 1010 000b (50h) and increment to the next slave address at every 64K boundary. This allows for up to eight 64KByte (512KBit) I2C EEPROMs to be connected on the same bus.

An example of valid I2C EEPROM system is two 64Kbyte EEPROMs and one 32Kbyte EEPROM, where the first two devices are used for addresses 50 0000h − 51 FFFFh and the third device is used for addresses 52 0000h − 52 7FFFh.

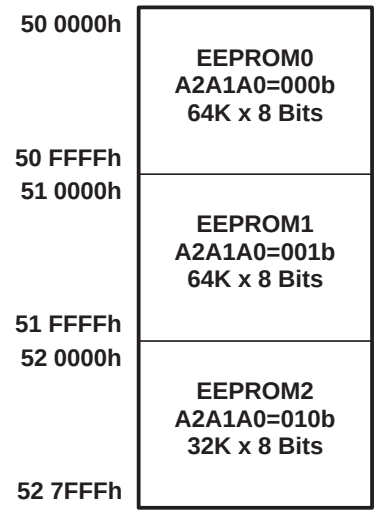


Figure 7. Example Memory Space for System With Three I2C EEPROMs

The I2C peripheral input clock is set to 1/4th the frequency of the CPU clock at reset. The bootloader will set the ICSPC = 0 and ICCLKL=ICCLKH=7. The frequency of the I2C bus can be calculated using the following formula:

$$SCL\_Freq = (CLKIN/4)/[(ICSPSC + 1)(ICCLKL + ICCLKH + 12)]$$
$$SCL\_Freq = (CLKIN/4)/[0 + 1)(7 + 7 + 12)]$$
$$SCL\_Freq = CLKIN/104$$

A CLKIN frequency should chosen such 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 process begins with the bootloader using a random read command to read address zero of the first 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 bootloader does not use the random read command for each byte within the 64K boundary, but depends on the ability of the serial EEPROM to automatically increment the address internally. All subsequent bytes with the 64K block are read using the current address read command as shown in Figure 9.

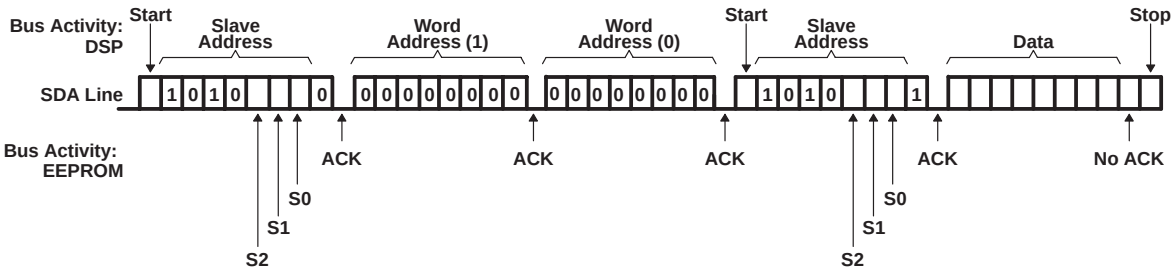


Figure 8. Reading the First Byte in a 64Kbyte Block

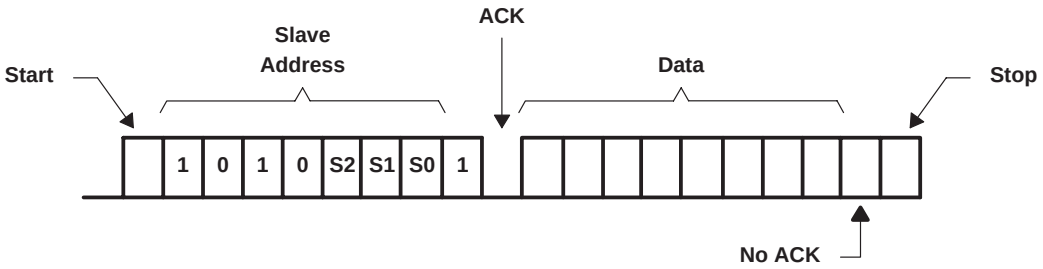


Figure 9. Current Address Read Command

When the address reaches a 64K boundary, the bootloader will increment the slave address (S[2:0]) by one and repeats the process described above to continue reading the boot table from the next EEPROM on the I2C bus.

The bootloader will stop requesting data when the first zero-length memory section is encountered in the boot table. All data bytes will be processed in accordance with the boot table description given later in this document.

The bootloader will set GPIO4 low when reading a data byte from the I2C EEPROM and set it high while it copies the data to its destination (this feature is not included on Version 1 of the bootloader, see section 3 for more details on the different versions of the bootloader). Upon successful load of the boot table the bootloader will set GPIO4 as an input and branch to the entry point address.

2.3.7 UART Boot Mode

The description in this section assumes familiarity with the UART peripheral. For detailed information on the C5501/C5502 UART refer to the *TMS320C5501/5502 DSP Universal Asynchronous Receiver/Transmitter (UART) Reference Guide* (SPRU597).

The bootloader supports boot from a host via the UART peripheral. The minimum connection required between UART and the external UART device is shown in Figure 10. The RX line will be used to receive data from the host UART, while the TX line will be used to echo all received data back to the host.

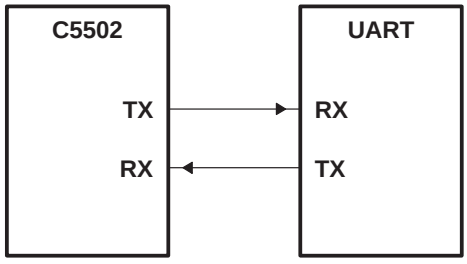


Figure 10. Signal Connections for UART Boot Mode

On the C5502, the UART RX and TX pins are multiplexed with the CLKR2 and FSR2 pins, respectively. GPIO7 must be held low at reset to enable the UART pins. The C5501 does not multiplex the UART pins.

Table 11. C5501 GPIO Pin Configuration for UART Boot Mode

BOOTM[2:0]	GPIO4	GPIO6†	GPIO7‡	Summary
111	0	X	0	Internal oscillator provides clock source. GPIO4 goes low to signal DSP is ready for data after 41,450 input reference clock cycles.
111	1	X	0	External clock provides clock source. GPIO4 goes low to signal DSP is ready for data after 570 input reference clock cycles.

† GPIO6 determines whether the EMIF is enabled or disabled after reset. If the destination of the loaded application is external memory, GPIO6 must be pulled high during reset.
‡ On the C5501, GPIO7 must be kept low during reset.

Table 12. C5502 GPIO Pin Configuration for UART Boot Mode

BOOTM[2:0]	GPIO4	GPIO6†	GPIO7	Summary
111	0	X	0	Internal oscillator provides clock source. GPIO4 goes low to signal DSP is ready for data after 41,450 input reference clock cycles.
111	1	X	0	External clock provides clock source. GPIO4 goes low to signal DSP is ready for data after 570 input reference clock cycles. UART pins are enabled.

† GPIO6 determines whether the EMIF is enabled or disabled after reset. If the destination of the loaded application is external memory, GPIO6 must be pulled high during reset.

The baud rate for the transmission is not fixed; instead it is dependant on the input clock frequency to the DSP (CLKIN). At reset, the UART peripheral input clock is set to divide by four of CLKIN. The baud rate that the UART peripheral will generate can be calculated using the equation

$$Baud\ Rate = CLKIN/(16 \times 4 \times Divisor\ Value)$$

The bootloader will always set the divisor value to 4. Table 13 lists possible baud rates based on common input clock frequencies.

Table 13. Baud Rates as a Function of CLKIN

CLKIN (MHz)	UART Clock (MHz)	Target Baud Rate (bps)	Actual Baud Rate (bps)	Percent Error (%)
1.25	0.3125	4,800	4,883	1.73
2.5	0.625	9,600	9,766	1.73
5	1.25	19,200	19,531	1.73
10	2	38,400	39,063	1.73
15	3.75	57,600	58,594	1.73
30	7.5	115,200	117,188	1.73

The bootloader will configure the UART with the following parameters:

- No parity
- One stop bit
- Eight-bit character length

The host UART must have the same settings for the transmission to work properly. The UART bootloader will also make use of the peripheral's 16-byte FIFO register (one-byte trigger level) to store incoming data.

After the bootloader has determined that the UART bootmode was selected it will initialize the UART peripheral to the settings described above, set the GPIO4 signal low, and wait until there is new data in the UART FIFO. All incoming data will be processed in accordance with the boot table description given later in this document. Each received byte will be echoed back to the host. The bootloader will stop checking for incoming data when the first zero-length section is encountered in the boot table.

The GPIO4 pin will be set low when the UART bootloader is ready to start receiving data and will remain low until the boot table transfer is complete. GPIO4 will go low after a specific number of input clock cycles depending on whether the internal oscillator is enabled or not. If GPI04 is low during reset, the internal oscillator will be enabled and GPIO4 will go low approximately 41,450 clock cycles after the DSP is taken out of reset. If GPIO4 high during reset, the internal oscillator will be disabled and GPIO4 will go low 570 clock cycles after the DSP is taken out of reset.

The bootloader will echo all received data back to the host, a feature that could be used to avoid overrun errors. Note that since the bootloader will enable the UART FIFO register, it is possible to keep sending data to the DSP even if it has not yet echoed the previous data. Keep in mind, however, that the UART FIFO has a size limit of 16 bytes.

Upon successful load of the boot table the bootloader will set the GPIO4 pin to input and branch to the entry point address. Note that there is no guard against errors such as parity bit errors or overrun errors; the success/failure of the UART bootmode can be determined by monitoring the echoed data and the status of the GPIO4 pin.

2.4 GPIO4 Behavior

The GPIO4 pin is configured as an output and used for multiple functions depending on the boot mode selected as indicated below.

In the standard serial boot mode modes, GPIO4 is used to indicate that the serial port is ready to receive data. If a programmed delay occurs, GPIO4 goes not ready (high) until the delay is completed and then ready (low) when the serial port is ready to receive again. GPIO4 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, GPIO4 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 boot load is complete, GPIO4 goes high.

In the serial EEPROM boot modes, GPIO4 is used as a chip select ($\overline{CS}$) signal to the serial EEPROM. It goes low at the beginning of the boot process and goes high when the boot process is complete. GPIO4 does not go low during delays in this mode, but since the DSP is the master, delays are handled automatically.

In I2C EEPROM boot mode, GPIO4 is set low when the peripheral is reading a data byte from the EEPROM and is set high the rest of the time.

The GPIO4 pin will be set low when the UART boot mode is ready to receive data and will remain low during the operation of the bootloader. The GPIO4 pin is used in the same manner for the HPI boot mode.

At the end of the boot process (in any boot mode), the GPIO4 pin is set high for a brief period of time and then set as input. The bootloader will then branch to the specified entry point address.

2.5 Boot Mode Branch Table

A boot mode branch table has been included in the C5501 and C5502 ROM that can be used by an application to jump to any boot mode. The location of this table has been selected such that it will not change across revisions of the bootloader code. Table 14 lists the addresses within the table that can be used to jump to a specific boot mode.

Table 14. Boot Mode Branch Addresses

BOOTM[2:0]	Boot Mode	Boot Mode Branch Address
001	Serial EEPROM (SPI) Boot from McBSP0	0FFECB4h
010	Standard Serial Boot from McBSP0 (16-bit)	0FFECB8h
011	External Asynchronous Memory (16-bit)	0FFECBCh
101	HPI boot	0FFECC4h
110	Serial EEPROM (I2C) Boot from I2C	0FFECC8h
111	Standard Serial Boot from UART (8-bit)	0FFECCCh

2.6 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 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.

2.6.1 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 boot load 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 0090h (byte address 0120h).

2.6.2 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 11.

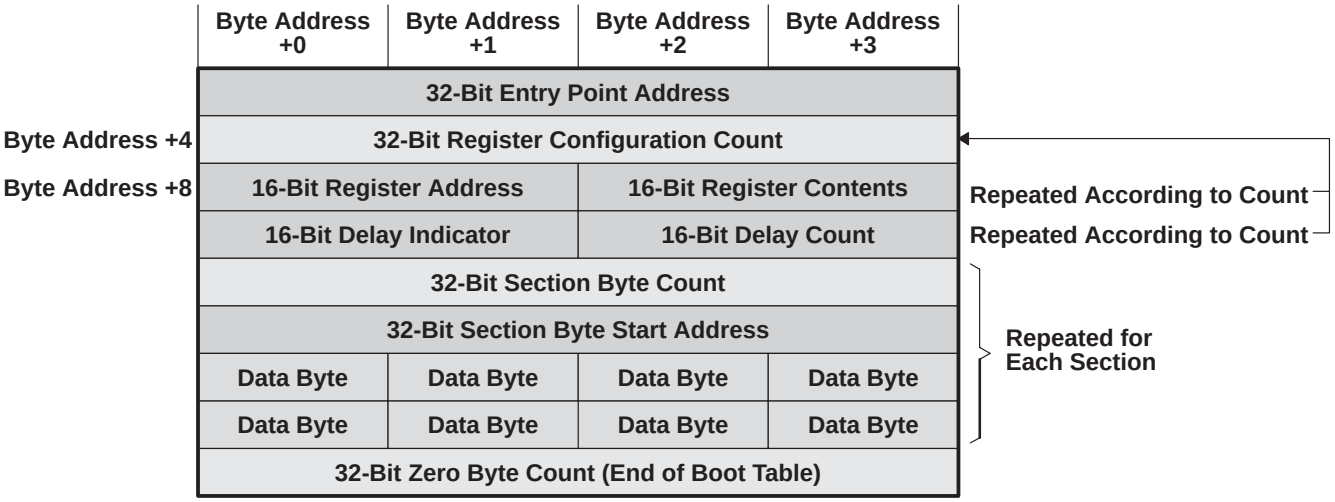


Figure 11. Boot Table Structure

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 boot load process (see section 2.6.3). 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.

2.6.3 Register Configuration and Delay During Boot

The 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 C5502 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 boot load 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 the user 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 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 boot load, the bootloader will proceed to copying code and data sections.

2.6.4 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. The remaining information included in this section describes the format of the sections in 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. 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 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 C5501 and C5502 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.

2.6.5 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 C5510, it also applies to the C5501 and C5502. 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 15 shows the correct option to select for each supported boot mode. HPI 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 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 15. Boot Mode Types for the Hex Conversion Utility

BOOTM[2:0]	For Boot Mode Source …	Include this Option …
001	Serial EEPROM (SPI) Boot from McBSP0 supporting 24-bit address	−serial8
010	Standard Serial Boot from McBSP0 (16-bit)	−serial16
011	External Asynchronous Memory (16-bit)	−parallel16
110	Serial EEPROM (I2C) Boot from I2C	−serial8
111	Standard Serial Boot from UART (8-bit)	−serial8

Some examples of how to set the hex conversion utility options to create a boot table are shown below.

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 TMS320VC5502
–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 PLLDIV1 register (port address 0x1C8C) with the value 0x0001
- After the PLLDIV1 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 TMS320VC5502
–serial8                       ;boot mode is 8-bit standard serial boot
–reg_config 0x1c8c, 0x0001     ;write 0x0001 to peripheral register at address
0x1C8C
–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* (literature number SPRU280).

3 Information about Different Bootloader Versions

This document has described the operation of Version A of the bootloader which exists on the C5501 and C5502. Some TMX samples of these devices contain an initial version of the bootloader which is different from Version A. Several enhancements were included in the Version A of the bootloader. This section describes how to determine the bootloader version and also explains the differences between the two bootloader versions.

3.1 Determining the Bootloader Version

The bootloader version can be determined by reading the contents of byte address 0FFECAEh in the ROM. Table 16 lists the contents of byte address OFFECAEh associated with each version of the bootloader.

Table 16. Bootloader Versions

Contents of Address 0FFECAEh	Bootloader Version
0F106h	Initial
0F107h	A

The bootloader version can also be determined through the package markings by reading the bootloader version letter. The *TMS320VC5502 Digital Signal Processor Silicon Errata* (SPRZ020) has more information on these package markings.

3.2 Differences Between Bootloader Versions

Some differences exist between the initial version of the bootloader and Version A of the bootloader. Table 17 lists the major differences between both bootloaders.

Table 17. Differences Between Bootloader Versions

Initial Version	Version A
SPI EEPROM Boot Mode	**SPI EEPROM Boot Mode**
• Not supported	• Adds capability to boot from 24-bit SPI EEPROMs (Serial Flash)
I2C Boot Mode	**I2C Boot Mode**
• GPIO4 not used to reflect boot mode status	• GPIO4 used to reflect boot mode status (I2C boot mode section in this document for more information on this feature).

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 Direct Execution from External Asynchronous Memory

The DSP configures the CE3 space for asynchronous memory and branches to the reset vector at location 0xFFFF00 in external CE3 space.

Check:	If no, then verify that:
Does CPU hit breakpoint at the reset vector at byte address location 0xFFFF00?	• BOOTM[2:0] pins are set to 000b or 100b at reset, for 16- or 32-bit memory, respectively. • The DSP is released from reset with a low-to-high transition of the reset signal. • The GPIO6 pin is pulled high at reset to enable the EMIF. • The internal oscillator is enabled or disabled appropriately. The GPIO4 pin should be set to 0 to enable, and 1 to disable, the internal oscillator. • The MPNMC bit is set to 1. Use a JTAG emulator and a debugger such as Code Composer Studio. • CE3 space is configured appropriately for 16- or 32-bit asynchronous memory. Use a JTAG emulator to view the registers using Code Composer Studio. • The external memory has been programmed properly with the reset vector at byte address location 0xFFFF00.

4.2 Parallel EMIF Boot Mode

The GPIO4 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 GPIO4 go low at the start of the bootload process?	• BOOTM[2:0] pins are set to 011b. • The DSP is released from reset with a low-to-high transition of the reset signal.
Does GPIO4 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. • GPIO6 is high at reset to enable the EMIF boot mode. • 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 GPIO4 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.

4.3 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 will branch to the entry point and begin code execution.

Check:	If no, then verify that:
Does the GPIO4 pin go high at the start of the bootload process, and then low after a short delay, indicating that the DSP is ready to receive data from the host?	• BOOTM[2:0] pins are set to 101b 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 may also use a JTAG emulator to verify that data is being written properly to DSP memory. • The GPIO6 setting is appropriate for the HPI data width. GPIO6 should be set to 1 for 8-bit multiplexed mode, and to 0 for 16-bit non-multiplexed mode. • The host is completing its data transfer. • If the host is not monitoring GPIO4, ensure that there is sufficient 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 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 attempt to load code outside of the word address range 0x0090 - 0x7FFF.

4.4 Standard Serial Boot Mode

The GPIO4 signal can be used as an external indicator of the status of the standard serial boot process. GPIO4 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 GPIO4 signal low and configure the McBSP as follows: RPHASE = 0b, RFRLEN1 = 0h, RWDLEN1 = 010b, RJUST = 00b, RDATDLY = 01b, externally generated CLKR0 and FSR0?	• BOOTM[2:0] pins are set to 010b. • 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 GPIO4 as a handshaking signal as described in 2.3.4.1 of SPRA911 (Using the TMS320VC5502 Bootloader) or allow at least 180 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 Slow Peripheral Clock groups runs at 1/4 the input clock frequency to the DSP. • The frequency of the Slow Peripheral Clock Group is not changed during the bootload process. • 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.

4.5 SPI EEPROM Boot Mode

The GPIO4 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 GPIO4 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 GPIO4?	• BOOTM[2:0] pins are set to 001b. • 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 required speed for the EEPROM matches the serial port clock speed according to the following formula: Serial port clock = Slow Peripheral Clock/4
Does the DSP drive the GPIO4 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 is programmed into the EEPROM as a single continuous image starting at EEPROM address 0.

4.6 I2C EEPROM Boot Mode

At the start of the I2C bootload process, the bootloader will cause the DSP to issue a random read command to the I2C device on the SDA line. GPIO4 will toggle 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 the DSP issue a random read command to the I2C device on the SDA line?	• BOOTM[2:0] pins are set to 110b. • 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 frequency is set to CLKIN/104.
Does GPIO4 toggle during the random read part of the bootload process?	• The boot table was created with the correct options. • HEX55 tool version 2.10 or later was used to create the boot table, and the C5510:2 option was used. • The boot media is programmed properly. • The proper options were chosen to create the boot table.
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.

4.7 UART Boot Mode

The GPIO4 signal goes low at the start of the UART bootload process, and the bootloader configures the UART for the transfer. Each time data is transmitted, the target echoes data back to the host, which can serve as an indicator that data transfer is successful. Finally, the DSP branches to the entry point address and the start of user code.

Check:	If no, then verify that:
Does the GPIO4 signal go low, and is the UART peripheral configured for 8 data bits, no parity, one stop bit, after RESET?	• BOOTM[2:0] pins are set to 111b. • The DSP is released from reset with a low-to-high transition of the reset signal.
Is data echoed back to the host each time it is transmitted?	• The Tx/Rx signals between the UART host and the DSP are intact and functional. • The host UART is set for 8 data bits, no parity, one stop bit. • GPIO7 is held low at reset to enable the UART. • The baud rate generated by the UART peripheral is CLKIN/(16 x 4 x 4). • Data is transferred on the RX pin from the host to fill up the FIFO. • There is no data overrun.
Does the DSP set the GPIO4 signal to input, and branch to the entry point byte address? Use an XDS emulator and a debugger such as Code Composer Studio to set a break-point at the entry point byte address.	• There is no data overrun. • The boot table contains a valid entry point byte address in the first 32-bit field. • HEX55 tool version 2.10 or later was used to create the boot table, and the C5510:2 option was used.

5 References

1. *TMS320VC5502 Fixed-Point Digital Signal Processor Data Manual* (SPRS166).
2. *TMS320VC5501 Fixed-Point Digital Signal Processor Data Manual* (SPRS206)
3. *TMS320C55x Assembly Language Tools User's Guide* (SPRU280).
4. *TMS320VC5501/5502 DSP Host Port Interface (HPI) Reference Guide* (SPRU620).
5. *TMS320VC5501/5502 DSP External Memory Interface (EMIF) Reference Guide* (SPRU621).
6. *TMS320VC5502/5509/5510 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide* (SPRU592).
7. *TMS320VC5501/5502/5509 DSP Inter−Integrated Circuit (I2C) Module Reference Guide* (SPRU146).
8. *TMS320VC5501/5502 DSP Universal Asynchronous Receiver/Transmitter (UART) Reference Guide* (SPRU597).
9. *TMS320VC5502 Digital Signal Processor Silicon Errata* (SPRZ020).

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