Application Report
SPRABP1A – April 2014 – Revised November 2019

Using the TMS320C5517 Bootloader

Steve Tsang

ABSTRACT

This application report describes the features of the on-chip ROM for the TMS320C5517 device. Included is a description of the bootloader and how to interface with it for each of the possible boot devices, as well as instructions for generating a boot image to store on an external device.

Contents

1 Introduction 2
2 Bootloader Operation 3
3 Boot Images 10
4 References 15

List of Tables

1 C5517 ROM Memory Map 2
2 Supported NAND Device IDs 5
3 C5517 Unencrypted Boot Image Format 10

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

1.1 On-Chip ROM

The on-chip ROM contains several factory-programmed sections.

- Algorithm tables to be referenced by drivers to reduce application data size
- Application programming interface (API) tables for referencing ROM API functions in applications
- Bootloader program

Table 1. C5517 ROM Memory Map

Starting Byte Address	Contents
FE_0000h	LCD Table
FE_0860h	WMA Encode Table
FE_9FA0h	WMA Decode Table
FE_D4C4h	MP3 Table
FE_F978h	Equalization Table
FE_FB14h	WM Voice Table
FF_64B4h	API Table
FF_683Ch	Bootloader Code (and other built-in API functions)

1.2 Bootloader Features

The major feature of the C5517 bootloader is booting unencrypted images from supported devices.

NOTE: SARAM31 (byte address 0x4E000 – 0x4FFFF) is reserved for the bootloader.

The bootloader also has the following features:

- Port-addressed register configuration during boot
- Programmable delay during register configuration

The bootloader is always invoked after reset. The function of the bootloader is to transfer user code from an external source to RAM. Once the transfer is completed, the bootloader transfers control to this user code.

2 Bootloader Operation

2.1 Bootloader Initialization

When the bootloader begins execution, it performs some initialization prior to attempting to load code.

- All peripherals are idled; the bootloader un-idles peripherals as it uses them.
- The bootloader programs the system clock generator based on the input clock selected via the CLK_SEL pin.
  - If CLK_SEL = 0, the bootloader bypass the system clock generator and assumes the input clock to be 12 MHz (via the on-chip USB oscillator).
  - If CLK_SEL = 1, the bootloader bypasses the system clock generator altogether and the system clock is driven by the CLKIN pin. In this case, the CLKIN frequency is expected to be 11.2896, 12.0, 12.288, 16.8, or 19.2 MHz. While the bootloader tries to boot from SPI, McSPI, UHPI, USB, and UART, the clock generator is programmed to multiply the input clock by 3 and adjust the TIMER0 setting to 200 ms.
- The low-voltage detection circuit is disabled to prevent trim setup (next step) from causing an unnecessary reset.
- The bootloader reads the trim values from the e-fuse farm and writes them into the analog trim registers.
- The bootloader idles all peripherals and disables CLKOUT before transferring control to user code.

2.2 Booting from External Devices

At reset, the values of EM_A[20:15]/GP[26:21] are latched into the BootMode[5:0] bits in the BootMode register ([1C34h]). After the reset, the C5517 bootloader determines which peripheral or method to boot from based on the value of BootMode[5:0] bits and queries the peripheral to determine if it can boot from that peripheral.

At that time, the individual peripheral clock will be enabled for the query and then disabled again when the bootloader is finished with the peripheral. By the time the bootloader releases control to the user code, all peripheral clocks will be off and all domains in the ICR, except the CPU domain, will be idled.

If the boot signature is not valid, toggle XF when the retry count reaches 100. If the boot is successful, set XF port low to indicate that boot-up is complete. Ensure a minimum of 200 ms has elapsed since TIMER0 began before proceeding to execute the bootloaded code.

For proper BootMode bits setting, see the *TMS320C5517 Fixed-Point Digital Signal Processor* (literature number SPRS727).

For a description of the valid boot image formats, see Section 3.

The following subsections describe details for each supported boot device.

2.2.1 NOR Flash

The bootloader supports booting from a NOR Flash attached to any of the four external memory interface (EMIF) chip-selects. The NOR Flash must use a 16-bit data bus.

Any 16-bit NOR Flash device should work for read-only use. To support bootloader reauthoring, which requires writing to the device, CFI-compliant bottom-boot-block and uniform-boot-block devices should work. Top-boot-block devices may or may not work (vendor dependent due to non-standard CFI implementations).

NOTE: The bootloader requires NOR Flash that supports a reset command (0xF0 on data).

The following is a list of NOR Flash devices that are explicitly supported for reauthoring (writing). Other devices may be supported.

- Spansion S29GL016A
- Spansion S29AL016M
- MXIC MX29LV160T
- Spansion S29GL032A
- SST SST39LF/VF200A
- SST SST39LF/VF400A
- SST SST39LF/VF800A

2.2.2 NAND Flash

The bootloader supports booting from 8- or 16-bit NAND Flash attached to any of the four EMIF chip-selects. The NAND Flash chip-enable and ready/busy signal must be connected to the C5517. The bootloader supports both small-block and large-block NAND Flash devices.

The NAND Flash must use the SSFDC format. The bootloader reads the Boot Image Page Pointer (BIPP) from each of the first 256 physical pages until a non-0 value is found. If the BIPP on page-0 is 0, all other pages must start with the 3-byte signature *0xE9 0x00 0x00* in order to use the BIPP from that page. The BIPP is located in bytes 0xC4 through 0xC6 of the page.

Once a valid BIPP is found, this value is used by the bootloader to determine which page contains the boot image to load. Note that the boot image cannot reside on the same page as the BIPP.

The bootloader also checks for the device ID to set the column and row byte address size. For devices not listed in Table 2, the bootloader will use the default setting.

See Table 2 for a list of supported NAND Device Ids.

Table 2. Supported NAND Device IDs

Device ID	Columns	Rows
01h	2	2
33h	1	2
35h	1	2
36h	1	3
39h	1	2
43h	1	2
45h	1	2
46h	1	3
53h	1	2
55h	1	2
56h	1	3
6Bh	1	2
72h	1	3
73h	1	2
74h	1	3
75h	1	2
76h	1	3
78h	1	3
79h	1	3
A1h	2	2
B1h	2	2
C1h	2	2
E3h	1	2
E5h	1	2
E6h	1	2
F1h	2	2
Default	2	3

2.2.3 16-bit SPI EEPROM

The bootloader supports booting with the following requirements for the external device:

- The device must support at least a 1- or 10-MHz SPI clock, depending on the boot mode.
- The device must be connected to SPI CS0 and act as an SPI slave.
- The device uses 2 bytes (16 bits) for internal addressing (up to 64kB).
- The device must have the capability to auto-increment its internal address counter to allow sequential reads from the device.
- The device can be connected to either valid pin-mapping for SPI; there are two distinct pin-mappings available. The bootloader attempts to communicate on each SPI pin-mapping, one at a time.
- For SPI_CLK ≤ 1 MHz or ≤ 10 MHz, the system clock generator output = input clock x 3.

2.2.4 24-Bit SPI Serial Flash

The bootloader supports booting from an SPI Flash with the following requirements for the external device.

- The device must support at least a 1- or 10-MHz SPI clock, depending on the boot mode.
- The device must be connected to SPI CS0 and act as an SPI slave.
- The device uses 3 bytes (24 bits) for internal addressing (up to 16MB).
- The device must have the capability to auto-increment its internal address counter to allow sequential reads from the device.
- The device can be connected to either valid pin-mapping for SPI (there are two distinct pin-mappings available). The bootloader attempts to communicate on each SPI pin-mapping, one at a time.

2.2.5 I2C EEPROM

The bootloader supports booting from an I2C EEPROM with the following requirements for the external device:

- The device must support the *fast* I2C specification (400 kHz).
- The device must respond to slave address 0x50 (7-bit address).
- The device uses 2 bytes for internal addressing (up to 64kB).
- The device must have the capability to auto-increment its internal address counter to allow sequential reads from the device.

2.2.6 MMC/eMMC

The bootloader supports booting from an MMC/eMMC device with the following requirements for the external device:

- The device must be connected to either the MMC/SD0 or the MMC/SD1 interface.
- The MMC/eMMC device must comply with *MMC Specification v3.31*, *eMMC Specification v4.3*, or later.
- Must be formatted in FAT16/32.
- The boot images can be in either the boot partition (with boot-from-partition enabled) or in the user data area formatted in FAT16/32.
  - The unencrypted boot image must be in the first data partition with filename “bootimg.bin”.
- The bootloader will check for user data partition first. If there is no valid boot image, then it will check for the boot from partition option.

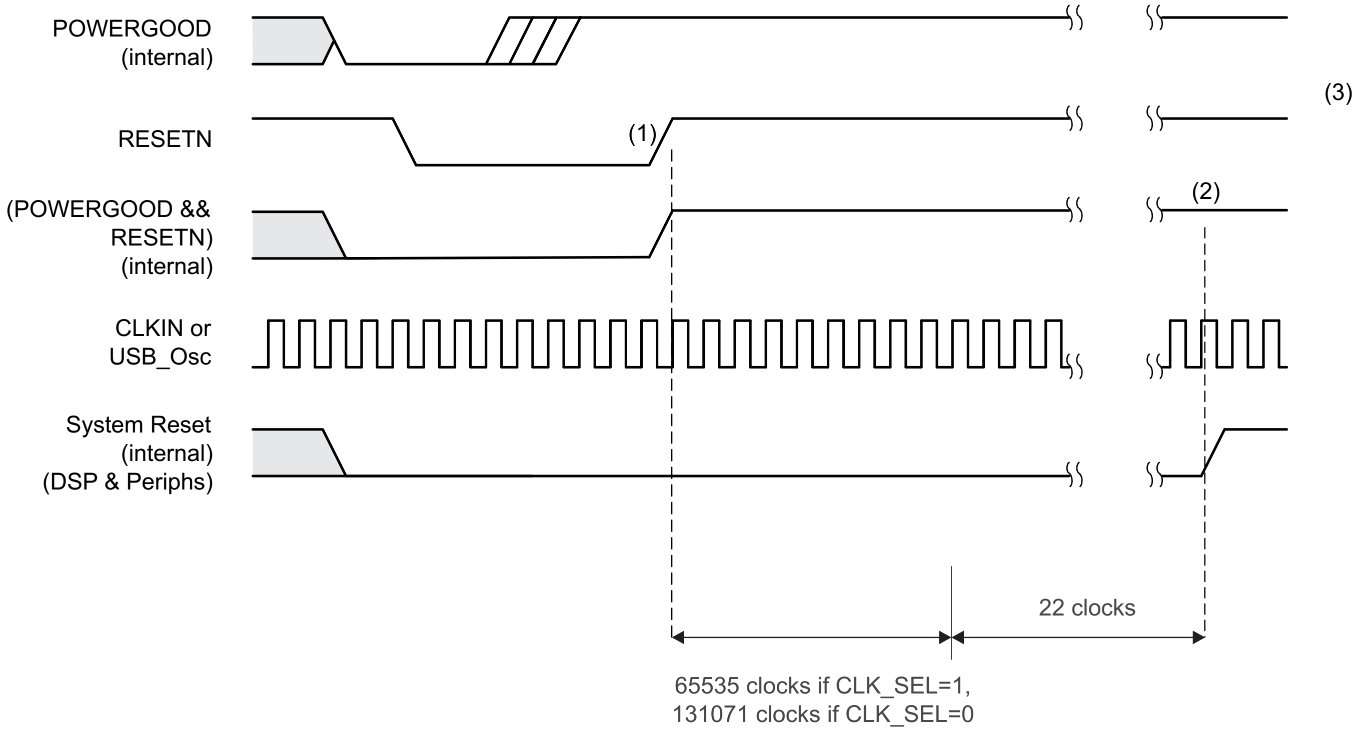
2.2.7 SD

The bootloader supports booting from an SD device with the following requirements for the external device:

- The device must be connected to either the MMC/SD0 or the MMC/SD1 interface.
- The SD device must comply with *SD Specifications Part 1 Physical Layer Simplified Specification v1.1 or v2.0*.
- Must be formatted in FAT16/32.
- The SD device must use the SD insecure mode (see SD specification). Note that this does not refer to the boot image security; boot images must be an unencrypted image for use with SD.
- The boot image must be in the first partition with the file name of *bootimg.bin*.

2.2.8 UHPI

Figure 1, *Boot Timing*, illustrates the UHPI boot sequence.


POWERGOOD (internal)
RESETN
(1)
(3)
(POWERGOOD && RESETN) (internal)
(2)
CLKIN or USB_Osc
System Reset (internal) (DSP & Periphs)
22 clocks
65535 clocks if CLK_SEL=1,
131071 clocks if CLK_SEL=0

(1) Enter the boot sequence.

(2) The maximum wait time from reset between (1) and (2) until the reset is released is 20 ms.

(3) The bootloaded code starts. The best-case time is 200 ms from the start of the boot sequence (see note 2). The worst-case time is the loading time for the bootloaded code when it exceeds 200 ms.

Figure 1. Boot Timing

The bootloader supports booting from a UHPI device with the following procedures:

- The external host has to communicate in 16-bit multiplexed mode.
- The external host powers up the device and must wait for the settling time of BG_CAP (200 ms) to elapse before executing the next step.
- The bootloader waits for the external host to finish transferring the data.
- External Host writes to device on-chip memory. The code or data sections are directly loaded to the desired locations on device by the external host.
- External Host interrupts the device through the DSP_INT in the UHPIC register after code transfer complete.
- Bootloader branches to the entry point. The entry point is located in the last block of SARAM, word addresses 0x27FFA and 0x27FFB. To ensure data integrity, the external host writes two 16-bit signatures in 0x27FFC and 0x27FFD with respective values of 0x1234 and 0xABCD.

2.2.9 McSPI

The bootloader tests for 24-bit McSPI flash boot on SPI_CS[0] using a clock-rate close to, but not over, 10 MHz, or a clock-rate of 40 MHz based on the boot mode. Booting from McSPI is supported with the following requirements:

- The device supports at least a 10- or 40-MHz SPI clock, depending on the boot mode.
- The device is connected to McSPI CS0 and acts as a McSPI slave.
- The device uses 3 bytes (24 bits) for internal addressing (up to 16MB).
- The device has the capability to auto-increment its internal address counter to allow sequential reads from the device.
- The device is connected to valid pin-mapping for McSPI.

2.2.10 UART

The bootloader supports booting from a UART by setting the PLL to multiply the input clock by 3 and adjust TIMER0 setting for 200 ms. Then, UART is programmed with 9600-, 57600-, or 115200-baud based on boot mode, 8-bit data, odd parity, one stop-bit, and auto flow control using CTS/RTS.

2.2.11 USB

The bootloader supports booting from the USB. The bootloader uses bulk-endpoint 1, vendor-id 0x0451, and product-id 0x9010 only with USB high-speed mode.

2.3 Register Configuration

Once the bootloader detects a valid boot-image signature (see Section 3), the first data that is used from the boot image is the optional register configuration data. This data allows you to setup peripheral port-addressed registers during the boot process, and before the code sections are copied. This feature provides the capability to change peripheral registers for specific purposes, such as configuring the EMIF external memory spaces.

NOTE: Using the register configuration feature to reprogram register settings may cause the bootloader to fail.

Since some register configurations may have an associated latency that must be observed before continuing, a delay feature is also available (as part of the register configuration data).

For a description of how to insert register configuration data, including delays, into a boot image, see Section 3.1.1.

2.4 Code Sections

After the optional register configuration is complete, the bootloader copies all of the code sections from the boot image to RAM. Each of these code sections may be actual code or just data; these sections are typically defined by the linker command file.

2.5 Bootloader Completion

After all code sections have been copied, the bootloader:

1. sets the XF port low to indicate that boot-up is complete,
2. waits to ensure that the bandgap settling time (at least 200 ms) has elapsed since the trim setup,
3. re-enables the low-voltage detection circuit, and
4. branches to the entry-point address specified in the boot image.

At this point, the bootloader's task is complete, and the user application is executing.

3 Boot Images

The bootloader's primary function is to transfer user code into RAM and then transfer control to this user code. The user code must be formatted into the boot image format supported by the bootloader. The format supported by the C5517 bootloader is an unencrypted boot image format.

The following sections describe the boot image format and how to create it.

3.1 *Unencrypted Boot Image Format*

The unencrypted boot image format contains the following information:

- All user code and data sections to be loaded to RAM.
- Register configuration data for setting up peripheral registers prior to loading code.
- The entry-point of the user's application.

The unencrypted boot image format is shown in Table 3.

Table 3. C5517 Unencrypted Boot Image Format

Word	Content	Valid Data Entries
1	Boot Signature (16-bits)	0x09AA
2	Entry Point (32 bits)	Byte address to begin execution (MSW)
3		Byte address to begin execution (LSW)
4	Register Configuration Count (16 bits, N = count)	1 to 2^{16} - 1
5	Register Config #1 Address in I/O space	Repeated according to register configuration count. Register configuration address is any valid register in C5517 I/O space. Address 0xFFFF is reserved as a delay indicator to create delay in between register writes or at end of register writes.
6	Register Config #1 Value or delay count	
7	Register Config #2 Address in I/O space	
8	Register Config #2 Value or delay count	
	. . .	
	Register Config #N Address in I/O space	
4+2N	Register Config #N Value or delay count	0 to 2^{16} - 1
5+2N	Section 1 word count (16 bits) Size is the number of valid (non-pad) data words in block M = (size + 2) rounded up to nearest multiple of 64-bit boundary	1 to 2^{16}-1
6+2N	Destination MSW address to load Section 1 (32 bits)	16-bit word address MSW
7+2N	Destination LSW address to load Section 1 (32 bits)	16-bit word address LSW
8+2N	First word of Section 1 (16 bits)	
	. . .	
5+2N+M	Last word of Section 1, often pad data (padded to 64-bit boundary)	
	. . .	
X	Section X word count (16 bits) Size is the number of valid (non-pad) data words in block N' = (size + 2) rounded up to nearest multiple of 64-bit boundary	1 to 2^{16}-1
X+1	Destination MSW address to load Section X (32 bits)	16-bit word address MSW
X+2	Destination LSW address to load Section X (32 bits)	16-bit word address LSW
X+3	First word of Section X (16 bits)	
	. . .	
X+N'	Last word of Section X, often pad data (padded to 64-bit boundary)	
X+N'+1	Zero word. Note that if more than one source block was read, word X+*N'* shown above would be the last word of the last source block. Each block would have the format shown in the shaded entries.	0x0000

3.1.1 Creating an Unencrypted Boot Image

A boot image can be created using the hex conversion utility (hex55) utility. The hex55 utility is intended for creating a unencrypted boot image.

For detailed information on the available hex conversion utility output formats, see the *TMS320C55x DSP Assembly Language Tools User's Guide* (SPRU280).

Use the hex conversion utility (hex55.exe) revision 4.3.5 or later. Earlier versions may not support the boot table features correctly.

3.1.1.1 Creating an Unencrypted Boot Image Using hex55

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

- Desired boot mode is 8-bit standard serial boot
- No registers are configured during the boot
- No programmed delays will occur during the boot
- Desired output is binary format in a file called *my_app.bin*

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

```
hex55 -boot -v5505 -serial8 -b -o my_app.bin my_app.out
```

```
-boot                        ;option to create a boot table
-v5505                       ;use C55x boot table format for TMS320C5517
-serial8                     ;boot mode is 8-bit standard serial boot
-b                           ;desired output format is binary format
-o my_app.bin                ;specify the output filename
my_app.out                   ;specify the input file
```

3.1.1.2 Creating an Unencrypted Boot Image with I/O Register Configuration

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 register address 0x1C8C with the value 0x0001
- After the register is configured, wait 256 cycles before continuing the boot
- Desired output is binary format in file a called *my_app.bin*

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

Note: If using the reg_config option, ensure there is no space between the address and value.

```
hex55 -boot -v5505 -serial8 -reg_config 0x1c8c,0x0001 -delay 0x100 -b -o my_app.bin my_app.out
```

```
-boot                        ;option to create a boot table
-v5505                       ;use C55x boot table format for TMS320C5517
-serial8                     ;boot mode is 8-bit standard serial boot
-reg_config 0x1c8c,0x0001    ;write 0x0001 to peripheral register at address 0x1C8C.
                             ;This can be repeated to program multiple registers.
-delay 0x100                 ;delay for 256 CPU clock cycles
-b                           ;desired output format is binary format
-o my_app.bin                ;specify the output filename
my_app.out                   ;specify the input file
```

NOTE: Using the register configuration feature to reprogram register settings may cause the bootloader to fail.

3.1.1.3 Section Alignment Restrictions When Using Hex55 Utility

All code sections must be aligned on a word boundary. Sections that are not properly aligned will be flagged by the hex55 utility.

To align a code section, use the *align* command in the linker command file as shown below. Note that if any function included in a code output section has an alignment associated with it (in C via CODE_ALIGN pragma) the whole section will inherit that alignment.

```
.text > ROM PAGE 0 align 2
```

3.1.1.4 DOS Command Line for Generating Boot Image Using Hex55

```
hex55 -boot -v5505 -b -serial8 -o USBKey_LED.bin USBKey_LED.out
```

3.2 Burning a Boot Image

Once a boot image (*.bin) is generated, you can burn the boot image into the supported peripherals using a utility called programmer, which is part of the TMS320C55x Chip Support Libraries (CSL) (see http://www.ti.com/tool/sprc133). The utility runs on each device using an emulator with Code Composer Studio.

NOTE: This device supports four four chip-select spaces: 2, 3, 4, and 5. See the Memory Map Summary in the data manual for details.
TMS320C5517 Fixed-Point Digital Signal Processor (literature number SPRS727)

CS selection for NAND and NOR flash is based on hardware hookup.
For C5517 EVM, 2 = NOR flash and 4 = NAND flash.

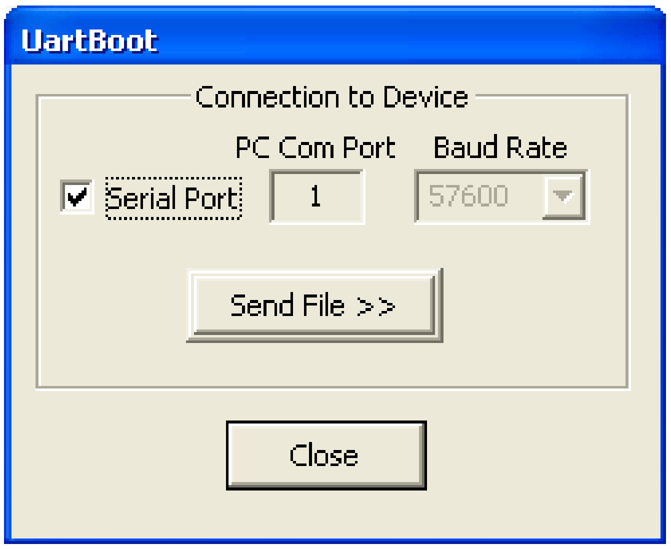
3.3 Booting from SD/SDHC/MMC/eMMC Card

To boot from SD/SDHC/MMC/eMMC card, first ensure the boot mode is set to either SD0 or SD1. Format the SD/SDHC/MMC/eMMC to FAT16/32, then copy the boot image into the root directory and rename it to “bootimg.bin”. Insert the SD/SDHC/MMC/eMMC in the SD slot. Power cycle the C5517 EVM. The bootloader will boot from the SD/SDHC/MMC/eMMC card.

3.4 Booting from UART

To boot from UART, first ensure one of the UART boot modes is set. Connect the UART connector on the EVM to the PC serial port with a null modem cable.

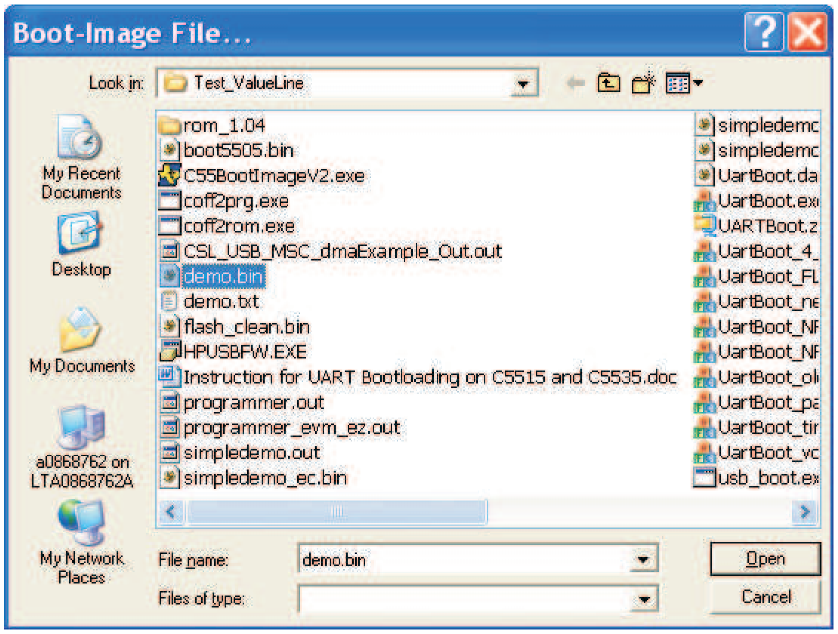
Run the UartBoot.exe and select the correct baud rate (9600, 57600, or 115200) in “Baud Rate”. Put 1 in “PC COM Port” and check the “Serial Port”. Finally, click “Send File >>”.



PC Com Port: 1
Baud Rate: 57600
[✓] Serial Port

Figure 2. Running UartBoot.exe

In the file selection dialog box, select the file—demo.bin, in this case—to upload. Click “Open” to start uploading.



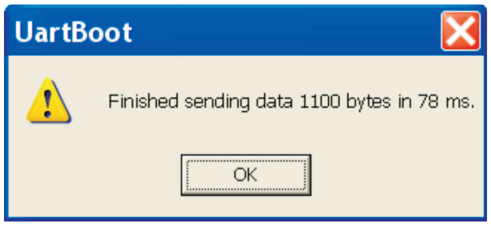
Look in: Test_ValueLine

rom_1.04
boot5505.bin
C55BootImageV2.exe
coff2prg.exe
coff2rom.exe
CSL_USB_MSC_dmaExample_Out.out
demo.bin
demo.txt
flash_clean.bin
HPUSBFW.EXE
Instruction for UART Bootloading on C5515 and C5535.doc
programmer.out
programmer_evm_ez.out
simpledemo.out
simpledemo_ec.bin
simpledemc
simpledemc
UartBoot.da
UartBoot.ex
UARTBoot.z
UartBoot_4_
UartBoot_FL
UartBoot_ne
UartBoot_NF
UartBoot_NF
UartBoot_ol
UartBoot_pa
UartBoot_tir
UartBoot_vc
usb_boot.ex

File name: demo.bin
Files of type:

Figure 3. Select the Boot Image File

After the uploading is complete, the following message box will appear:



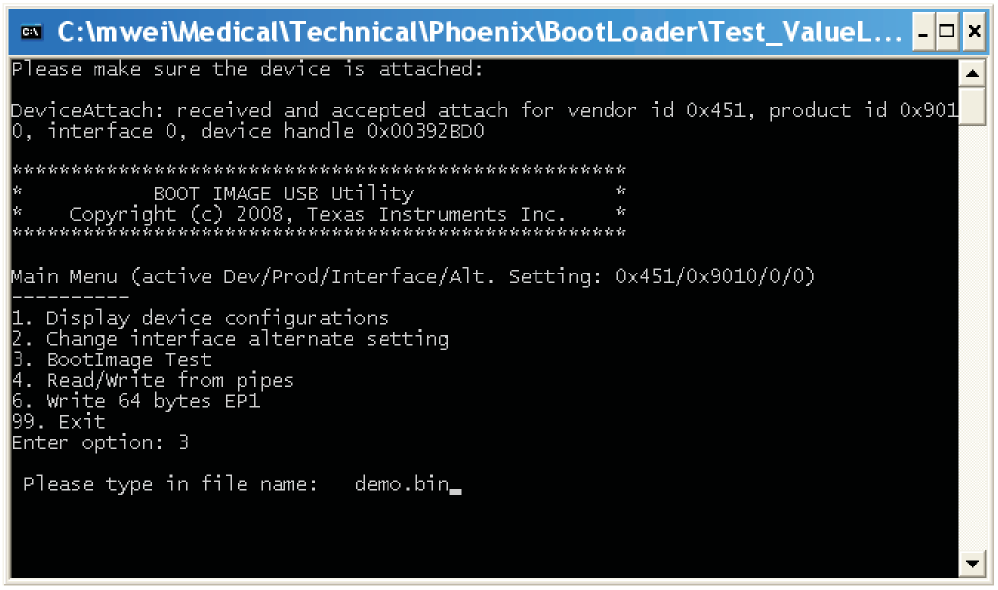
Finished sending data 1100 bytes in 78 ms.

Figure 4. UART Boot is Completed

The demo.bin should be now running on the EVM or eZdsp.

3.5 Booting from USB

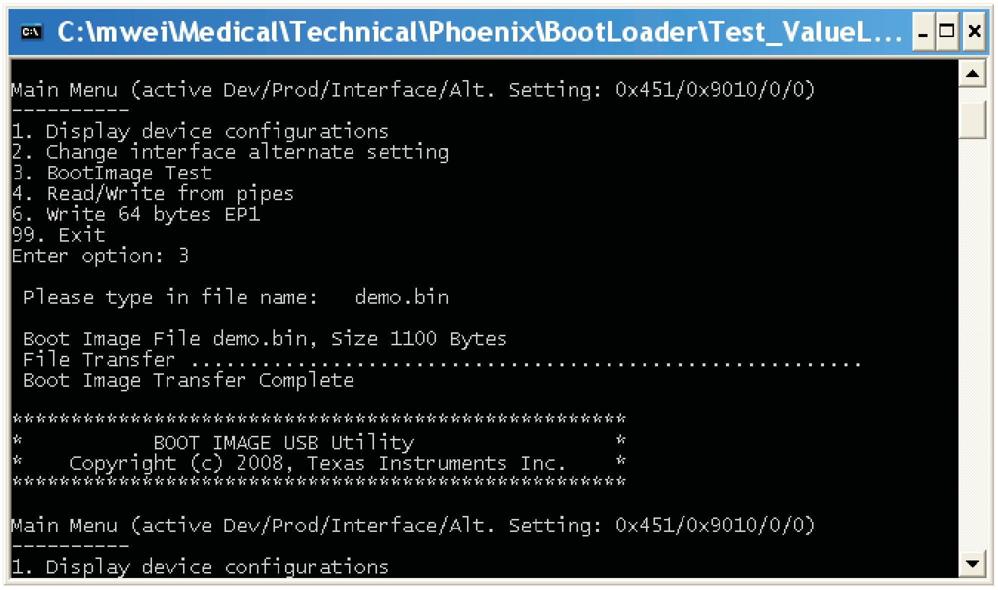
To boot from USB, first ensure there is no boot image in any other peripheral if the boot configuration is set to use polling mode. The J13 of the EVM should be connected to a PC USB port via a USB cable. Execute usb_boot.exe and enter option 3 for BootImage Test. Then input the boot image path name: demo.bin, in this case.



```
Please make sure the device is attached:

DeviceAttach: received and accepted attach for vendor id 0x451, product id 0x901
0, interface 0, device handle 0x00392BD0

****************************************************
*            BOOT IMAGE USB Utility                *
*    Copyright (c) 2008, Texas Instruments Inc.    *
****************************************************

Main Menu (active Dev/Prod/Interface/Alt. Setting: 0x451/0x9010/0/0)
----------
1. Display device configurations
2. Change interface alternate setting
3. BootImage Test
4. Read/Write from pipes
6. Write 64 bytes EP1
99. Exit
Enter option: 3

 Please type in file name:   demo.bin_
```

Figure 5. Running usb_boot.exe

After <enter> the following dialog box should show:


C:\mwei\Medical\Technical\Phoenix\BootLoader\Test_ValueL...

```
Main Menu (active Dev/Prod/Interface/Alt. Setting: 0x451/0x9010/0/0)
----------
1. Display device configurations
2. Change interface alternate setting
3. BootImage Test
4. Read/Write from pipes
6. Write 64 bytes EP1
99. Exit
Enter option: 3

 Please type in file name:   demo.bin

 Boot Image File demo.bin, Size 1100 Bytes
 File Transfer .......................................................
 Boot Image Transfer Complete

****************************************************
*            BOOT IMAGE USB Utility               *
*    Copyright (c) 2008, Texas Instruments Inc.    *
****************************************************

Main Menu (active Dev/Prod/Interface/Alt. Setting: 0x451/0x9010/0/0)
----------
1. Display device configurations
```

Figure 6. USB Boot is Completed

The demo.bin should now run on the EVM.

4 References

- *TMS320C55x DSP Assembly Language Tools User's Guide* (SPRU280)

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (April 2014) to A Revision	Page
• Update was made in the Abstract of this document.	1