Application Report
SPRABL7A–September 2011–Revised January 2012

Using the TMS320C5535/34/33/32 Bootloader

Ming Wei

ABSTRACT

This document describes the features of the on-chip ROM for the TMS320C5535/34/33/32 digital-signal processors. 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.

The C5535/34/33/32 bootloader uses USB and UART executable files that can be downloaded from http://www-s.ti.com/sc/techlit/sprabl7.zip.

Contents

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
- API tables for referencing ROM API functions in applications
- Bootloader program

Table 1. 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	API Table
FE_FE9Ch	Bootloader Code (and other built-in API functions)
FF_FEFCh	Bootloader ID
FF_FF00h	Bootloader Interrupt Vector Table

1.2 Bootloader Features

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 will transfer control to this user code.

In order to ensure that the code cannot be accessed and read outside of the system, the code residing externally may be encrypted. The bootloader is responsible for bootloading the code from an external device (I2C EEPROM, SPI EEPROM, SPI serial flash, MMC/SD, USB, and UART), decrypting it if necessary, and writing it into DSP memory (on-chip or off-chip).

Note: SARAM31 (byte address 0x4E000 – 0x4FFFF) is reserved for the bootloader during boot. But it is free to use by user code after bootloader transferred control.

The major features of the bootloader are:

- Boot both secure and unsecure images from 16-bit SPI EEPROM, 24-bit SPI serial flash, I2C EEPROM, SD/SDHC/eMMC/moviNAND, UART, and USB.
- Support reauthoring for 16-bit SPI EEPROM, I2C EEPROM, and SD/SDHC/eMMC/moviNAND.

The bootloader also has the following features:

- Memory-mapped I/O register configuration during boot
- Programmable delay during I/O register configuration

2 Bootloader Operation

2.1 Bootloader Initialization

When the bootloader begins execution, it performs some initialization before attempting to load code:

- All peripherals are idled (the bootloader un-idles peripherals as it uses them).
- CPU Clock setup
  - If CLK_SEL = 0, the bootloader powers up the PLL and sets its output frequency to 12.288 MHz (multiply 32 768 Hz RTC Clock by 375).
  - If CLK_SEL = 1, the bootloader bypasses the PLL and uses CLKIN. Note that CLKIN is expected to be 11.2896 MHz, 12.0 MHz, or 12.288 MHz.
- 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.

2.2 Boot Devices

Each device has a fixed order in which it checks for a valid boot image on each supported boot device. The device order is 16-bit SPI EEPROM, 24-bit SPI serial flash, I2C EEPROM, and SD/SDHC/eMMC/moviNAND. The first device with a valid boot image will be used to load and execute user code.

If none of these devices has a valid boot image, the bootloader will modify the CPU Clock setup as follows:

- If CLK_SEL = 0, the bootloader powers up the PLL and sets its output frequency to 36.864 MHz (multiply 32 768 Hz RTC Clock by 1125).
- If CLK_SEL = 1, the bootloader powers up the PLL and sets it to multiply CLKIN by 3.

This CPU clock setup change is required to meet the minimum frequency needed by the USB module.

Next the bootloader goes into an endless loop checking for data received on either the UART or USB. If a valid boot image is received from either device, it will be used to load and execute user code. If no valid boot image is received, the bootloader will simply continue to monitor these two devices. During this endless loop, if the time since the trim setup exceeds 200 ms, then the bootloader will re-enable the low-voltage detection circuit. This is done to prevent leaving the low-voltage detection disabled for an extended period of time.

See Section 3 for a description of the valid boot image formats.

The following subsections describe details for each supported boot device.

2.2.1 16-Bit SPI EEPROM

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

- The device must support at least a 500-kHz SPI clock.
- The device must be connected to SPI CS0 and act as an SPI slave.
- The device uses two 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 may be connected to either of the two available pin-mappings for SPI. The bootloader attempts to communicate on each SPI pin-mapping, one at a time.

2.2.2 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 500-kHz SPI clock.
- The device must be connected to SPI CS0 and act as an SPI slave.
- The device uses three 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 may 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.
- Any and all write-protect features must be disabled if re-authoring is needed. The Bootloader will not attempt to disable these features.

2.2.3 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 two 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.

2.2.4 SD

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

- The device must be connected to the eMMC/SD0 interface.
- The SD device must comply with *SD Specifications Part 1 Physical Layer Simplified Specification v1.1* or *v2.0* and 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 can be either the secure image or unsecure image for use with SD.
- The boot image must be in the first partition with a filename of “bootimg.bin”.

2.2.5 eMMC/moviNAND

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

- The device must be connected to the eMMC/SD0 interface.
- The eMMC/moviNAND device must comply with eMMC Specification v4.3, or later, and 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 boot image must be in the first data partition with filename “bootimg.bin”. Both a secure image and unsecure image are supported for use with eMMC/moviNAND.
- 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.

2.2.6 UART

The bootloader supports booting from the UART. The bootloader sets up to receive data using the following UART parameters: 8-bit data, odd-parity, 1 stop-bit, and 57 600 baud rate.

To reduce the probability of a receive error, the external transmitter should set up to use two stop-bits.

Note that this setup may result in data receive errors if CLK_SEL is set to use CLKIN when CLKIN is 11.2896 MHz. The error rate may be reduced by the external device by adding additional time between frames (bytes).

2.2.7 USB

The bootloader supports booting from the USB. The bootloader uses bulk-endpoint 1 (OUT), vendor-id 0x0451, and product-id 0x9010.

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 the user to set up 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.

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.

See Section 3.3 for a description of how to insert register configuration data, including delays, into a boot image.

2.4 Code Sections

After the optional register configuration is complete, the bootloader will copy 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 link-control file.

2.5 Bootloader Completion

After all code sections have been copied, the bootloader will wait to ensure that at least 200 ms have elapsed since the trim setup. Re-enable the low-voltage detection circuit, and then branch 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 a boot image format supported by the bootloader.

There are two distinct formats supported by the bootloader: An unsecure boot image format, and a secure boot image format. These two formats store the same information, with the only difference being the secure format uses encryption to protect the user’s data or code.

The following sections describe these two boot image formats and how to create boot images.

3.1 Unsecure Boot Image Format

The unsecure boot image format contains the following information:

- All user code/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
- A boot signature to distinguish unsecure from secure boot images. The unsecure boot image boot signature is 0x09AA.

See the following table for the unsecure boot image format.

Table 2. Unsecure 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)	0 to 2^{16} – 1
5	Register Config #1 Address in I/O space	Repeated according to Register Configuration Count. Register Config address is any valid register in 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

3.2 Secure Boot Image Format

The secure boot image format contains the following information:

- All user code/data sections to be loaded to RAM (encrypted)
- Register configuration data for setting up peripheral registers before loading code (encrypted)
- The entry-point of the user's application (encrypted)
- The file-key used for encrypting the remainder of the file (encrypted). The encryption-seed, device-id (if this is a bound image), and a random number are combined to create the file-key.
- The seed-offset for the encryption-seed used. This is an index into the Secure-ROM seed table (value = 0–127).
- A boot signature to distinguish secure boot images from unsecure images, and to indicate the type of secure boot image.

There are three different types of secure boot images:

- Secure boot image that is not bound to the device: the boot signature is 0x09A5. The encryption for this image is based only on an assigned encryption key.
- Secure boot image that is requesting to be bound to the device: the boot signature is 0x09A6. The encryption for this image is based only on an assigned encryption key, but the bootloader will re-encrypt this image using the assigned encryption key and the device-id, which is unique to each device. This re-encrypted image is then said to be "bound" to that specific device, and is written back to the external storage device from which it was originally read.
- Secure boot image that is bound to the device: the boot signature is 0x09A4. The encryption for this image is based only on an assigned encryption key and the device-id. Only a single device can decrypt this image.

See Table 3 for the secure boot image format.

Note: For details about generating the encrypted boot image, please contact your local TI sales representative.

Table 3. Secure Boot Image Format

Word	Content	Valid Data Entries
1	Boot Signature (16-bits)	0x09A4, 0x09A5, 0x09A6
2	Encryption Seed Offset (16-bits)	0 to 127 (assigned by TI)
3:10	Encrypted File Key (128 bits)	Safer(SHA1(Seed OR Seed + ID),File Key)
11:14	Two pad words (0x00000) and Entry Point (32-bits) all encrypted	Safer encrypted byte address to begin execution
15	Register Configuration Count (16-bits, N = count)	0 to 2^{16} – 1 (not encrypted)
16	Register Config #1 Address in I/O space	Repeated according to Register Configuration Count. Register Config address is any valid register in I/O space. Address 0xFFFF is reserved as a delay indicator to create delay in between register writes or at end of register writes. This section is encrypted and padded to 64-bit boundary.
17	Register Config #1 Value or delay count	
18	Register Config #2 Address in I/O space	
19	Register Config #2 Value or delay count	
	. . .	
	Register Config #N Address in I/O space	
15+2N	Register Config #N Value or delay count	0 to 2^{16} – 1
16+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 (not encrypted)
17+2N	Destination MSW address to load Section 1 (32-bits)	Safer encrypted C55x 16-bit word address (0x000060 to 0x09 7FFF) MSW
18+2N	Destination LSW address to load Section 1 (32-bits)	Safer encrypted C55x 16-bit word address (0x000060 to 0x09 7FFF) LSW
19+2N	First word of Section 1 (16-bits)	Safer encrypted C55x instructions or any 16-bit-wide data value
16+2N+M	. . .	Safer encrypted C55x instructions or any 16-bit-wide data value
	Last word of Section 1, often pad data (padded to 64-bit boundary)	Safer encrypted C55x instructions or any 16-bit-wide data value
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 (not encrypted)
X+1	Destination MSW address to load Section X (32-bits)	Safer encrypted C55x 16-bit word address (0x000060 to 0x097FFF) MSW
X+3	Destination LSW address to load Section X (32-bits)	Safer encrypted C55x 16-bit word address (0x000060 to 0x097FFF) LSW
	First word of Section X (16-bits)	Safer encrypted C55x instructions or any 16-bit-wide data value
X+N'	. . .	Safer encrypted C55x instructions or any 16-bit-wide data value
	Last word of Section X, often pad data (padded to 64-bit boundary)	Safer encrypted C55x instructions or any 16-bit-wide data value
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
X+N'+2	Hash (160-bits)	SHA1-HMAC(SHA1(Seed + ID, File Key + data[11: X+N'+1])

3.3 Creating a Boot Image

A boot image can be created using the hex55 utility. The hex55 utility is intended for mass market support.

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

3.3.1 Creating a Boot Image With hex55 Utility

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

1. Use the hex conversion utility (hex55.exe), revision 4.3.5 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 ***–v5505*** option. 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 ***–serial8***.
5. Specify the desired output format.
6. 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.

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

3.3.1.1 Creating a 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 will be 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 TMS320C5535/34/33/32
-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

3.3.1.2 Creating a 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:

```
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 TMS320C5535/34/33/32
-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

3.3.1.3 *Section Alignment Restrictions When Using hex55 Utility*

All code sections must be aligned on word boundary. Sections which 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
```

3.3.1.4 *DOS Command Line for Generating Boot Image Using hex55*

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

3.4 *Burn a Boot Image*

Once a boot image (*.bin) is generated, customers can burn the boot image into the SPI EEPROM (16-bit or 24-bit), I2C EEPROM, SPI serial flash or SD/SDHC/eMMC/moviNAND. It is done by a utility called programmer which runs on each device using an emulator with CCS. First you will need to load the program, programmer_C5535_eZdsp.out. For writing to SPI serial flash, at the “C5535 eZdsp … input <file_path>” prompt, input the path name for the boot image<enter>. The following display will be shown:

SPI Flash…

Erase chip (SPI Flash)…

Open <file-path>

Input file opened

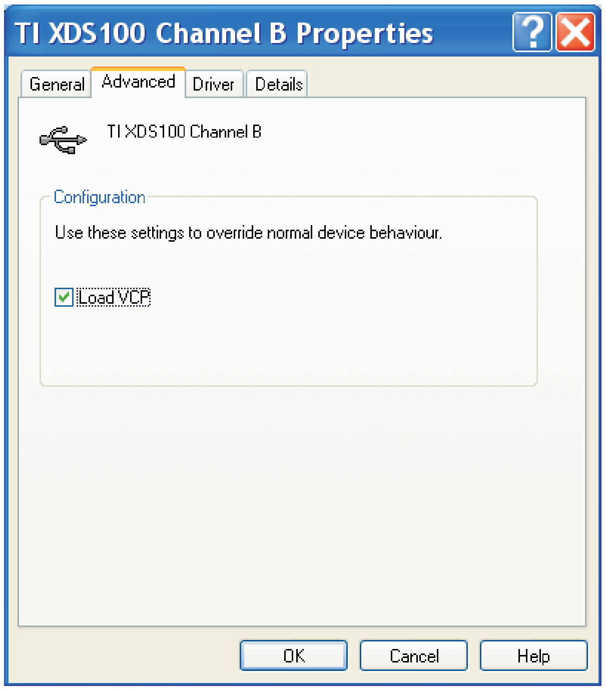
Programming Complete

3.5 *Boot from Micro SD/SDHC Card*

To boot from micro SD/SDHC card, first ensure there is no boot image in the SPI serial flash. Format the micro SD/SDHC to FAT16/32, then copy the boot image into the root directory and rename it to “bootimg.bin”. Put the micro SD/SDHC in the micro SD slot (J6). Power cycle the C5535 eZdsp. The bootloader will boot from the micro SD/SDHC card.

3.6 *Boot from UART*

To boot from UART, first ensure there is no boot image in the SPI serial flash or micro SD slot. The J2 of the C5535 eZdsp should be connected to a PC USB port via USB cable. It will use the virtual COM port (the COM port number varies by system) through TI XDS100 Channel B. Ensure pin 2 of SW3 is on and the Load VCP is checked for TI XDS100 Channel B (Control Panel→System→Hardware→Device Manager→USB Controller→TI XDS100 Channel B→Properties→Advanced).



TI XDS100 Channel B

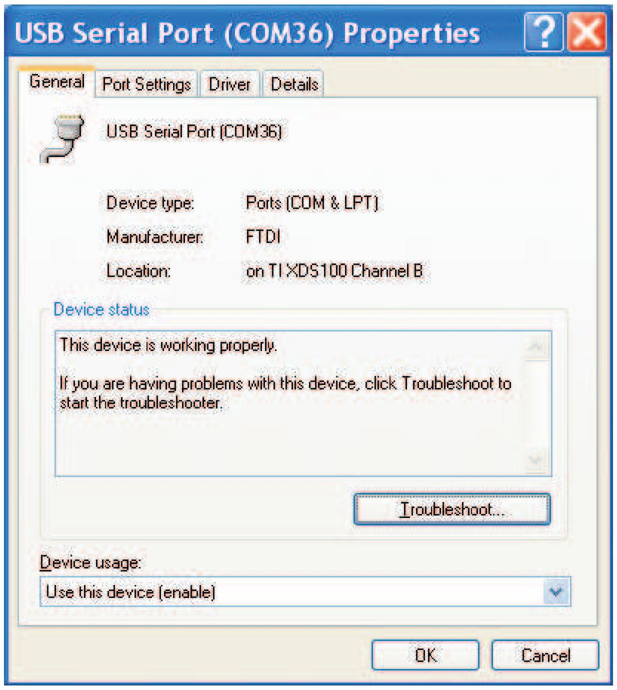
Configuration

Use these settings to override normal device behaviour.

[✓] Load VCP

Figure 1. Enable the Virtual COM Port

If you look at the property of a particular COM port (Control Panel→System→Hardware→Device Manager→Ports (COM & LPT)→USB Serial Port (COMxx)), the following dialog box should show:



USB Serial Port (COM36)

Device type: Ports (COM & LPT)

Manufacturer: FTDI

Location: on TI XDS100 Channel B

Device status

This device is working properly.

If you are having problems with this device, click Troubleshoot to start the troubleshooter.

Device usage: Use this device (enable)

Figure 2. Get the Virtual COM Port Number

Run the UartBoot.exe and select the 57600 in “Baud Rate”. Put the correct COM port number in “PC COM Port” and check the “Serial Port”. Finally, click “Send File >>”.

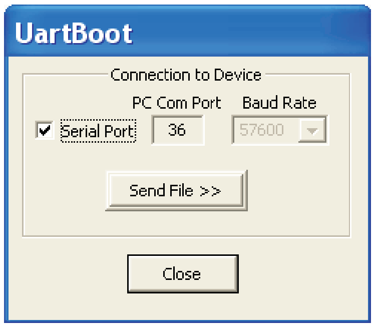


Figure 3. Running UartBoot.exe

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

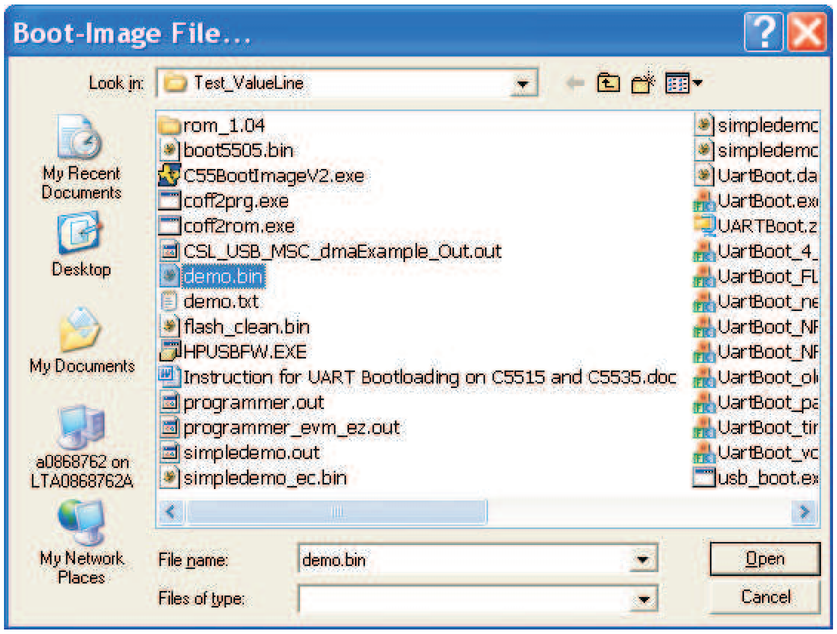
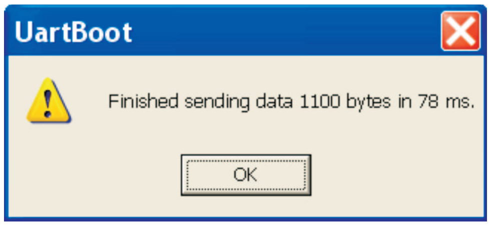


Figure 4. Select the Boot Image File

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



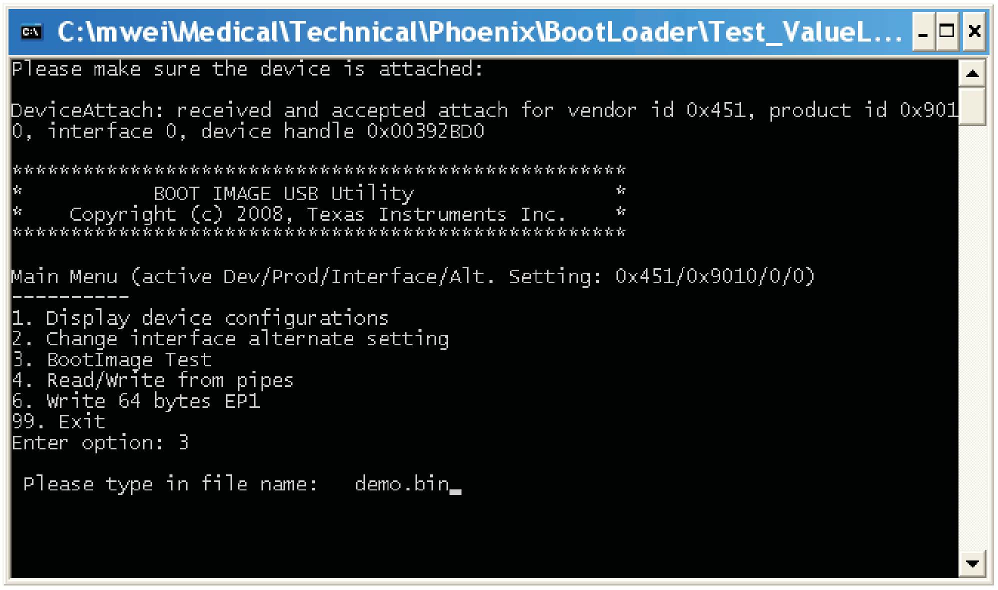
Finished sending data 1100 bytes in 78 ms.

Figure 5. UART Boot is Completed

The demo.bin should be now running on the C5535 eZdsp.

3.7 Boot from USB

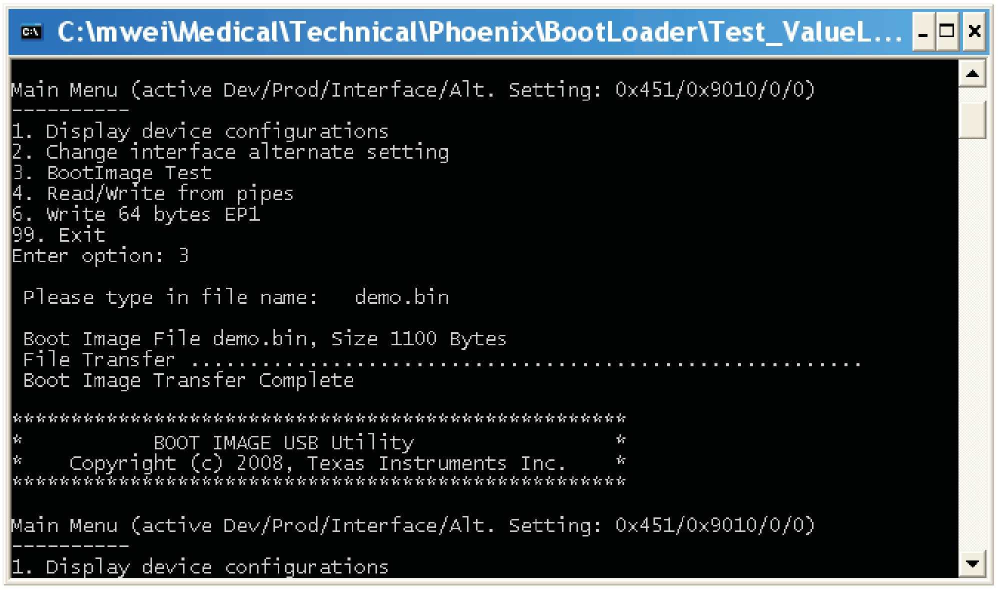
To boot from USB, first ensure there is no boot image in the SPI serial flash or micro SD slot. The J1 of the C5535 eZdsp should be connected to a PC USB port via a USB cable. Execute usb_boot.exe and enter option 3 for BootImage Test. Then input the boot image path name: demo.bin, in this case.



```
C:\mwei\Medical\Technical\Phoenix\BootLoader\Test_ValueL...
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 6. 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 7. USB Boot is Completed

The demo.bin should now run on the C5535 eZdsp.

Revision History

Changes from Original (September, 2011) to A Revision	Page
• Added link to USB and UART attached files	1

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

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Mobile Processors	www.ti.com/omap		
Wireless Connectivity	www.ti.com/wirelessconnectivity		
		TI E2E Community Home Page	e2e.ti.com