Application Report
SPRABD7D–June 2010–Revised November 2019

Using the TMS320C5515/14/05/04 Bootloader

ABSTRACT

This application report describes the features of the on-chip ROM for the TMS320C5515/14/05/04 devices. 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 7
4 References 17

List of Tables

1 C5515/14/05/04 ROM Memory Map 2
2 Supported NAND Device IDs 5
3 C5515/14/05/04 Unencrypted Boot-Image Format 8
4 C5515/14/05/04 Encrypted Boot-Image Format 9

Trademarks

Code Composer Studio is a trademark of Texas Instruments.
All other 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. C5515/14/05/04 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 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.

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 (NAND Flash, NOR Flash, inter-integrated circuit (I2C) EEPROM, serial peripheral interface (SPI) EEPROM, MultiMedia card/secure data memory card (MMC/SD) and universal serial bus (USB)), 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.

The major features of the C5515/14/05/04 bootloader are:

- Boot both encrypted and unencrypted images from NOR, NAND, 16-bit SPI EEPROM, Flash, and I2C EEPROM, except for 24 bit SPI serial flash which supports only unencrypted images.
- Boot encrypted images from MMC/SD and USB.
- Support reauthoring for NOR, NAND, 16-bit SPI EEPROM, I2C EEPROM, and MMC/SD.

The bootloader also has the following features:

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

1.3 Supported External Devices

The bootloader is responsible for bootloading the code from an unencrypted or encrypted external device.

1.3.1 Unencrypted External Devices

All external devices support unencrypted booting.

1.3.2 Encrypted External Devices

Encrypted booting is available using three encrypted boot image formats:

- Not bound to device
- Requesting to be bound to the device
- Bound to device

All external devices support all three encrypted boot image formats.

After bootloading from an external device, the bootloader decrypts the boot image, if necessary, and writes it into DSP memory (on-chip or off-chip).

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.
- CPU clock setup
  - If CLK_SEL = 0, the bootloader powers up the phase-locked loop (PLL) and sets its output frequency to 12.288 MHz (multiply 32768 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

The C5515/14/05/04 has a fixed order in which it checks for a valid boot-image on each supported boot device. The device order is NOR Flash, NAND Flash, 16-bit SPI EEPROM, I2C EEPROM, and MMC/SD. The first device with valid boot-image is used to load and execute user code.

If none of these devices has a valid boot-image, the bootloader modifies 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 32768 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 the USB. If a valid boot-image is received, it is used to load and execute user code. If no valid boot-image is received, the bootloader continues to monitor the devices. During this endless loop, if the time since the trim setup exceeds 200 ms, the bootloader re-enables the low-voltage detection circuit; this is done to prevent leaving the low-voltage detection disabled for an extended period of time.

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 a NAND Flash attached to any of the four EMIF chip-selects. The NAND Flash must use an 8-bit data bus, and its chip-enable and ready/busy signal must be connected to the C5515/14/05/04. 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.

For a list of supported NAND Device Ids, see Table 2.

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
AAh	2	3
Default	2	3

2.2.3 16-Bit Serial Peripheral Interface (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 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.

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 500 kHz SPI clock.
- 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.
- Any and all write-protect features must be disabled if re-authoring is needed. The bootloader will not attempt to disable these features.

2.2.5 Inter-integrated Circuit (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 MultiMedia Card (MMC)

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

- The device must be connected to the MMC/SD0 interface.
- The boot-image must be in the first partition with filename *boot5505.bin*.

2.2.7 Secure Data (SD)

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

- The device must be connected to the MMC/SD0 interface.
- The SD device must comply with SD specification v1.1, or v2.0 for FAT32.
- 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 the secure type for use with SD.
- The boot-image must be in the first partition with filename *boot5505.bin*.

2.2.8 Universal Serial Bus (USB)

The bootloader supports booting from the USB. The bootloader uses bulk-endpoint 1, 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 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.

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 3.3.

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 link-control file.

2.5 Bootloader Completion

After all code sections have been copied, the bootloader waits to ensure that at least 200 ms has elapsed since the trim setup, re-enables the low-voltage detection circuit, and then 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 a boot-image format supported by the bootloader.

There are two distinct formats supported by the C5515/14/05/04 bootloader: an unencrypted boot-image format and an encrypted boot-image format. These two formats store the same information, with the only difference being the encrypted format uses encryption to protect the user's data/code.

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

3.1 Unencrypted Boot Image Format

The unencrypted 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 unencrypted boot-images from encrypted-images. The unencrypted boot-image boot-signature is 0x09AA.

The unencrypted boot-image format is shown in Table 3.

Table 3. C5515/14/05/04 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 C5515/14/05/04 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 Encrypted Boot Image Format

The encrypted 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 prior to 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 ROM seed table (value = 0 to 127).
- A boot-signature to distinguish encrypted boot-images from unencrypted-images, and to indicate the type of encrypted boot-image.

There are three different types of encrypted boot-images:

- Encrypted 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.

- Encrypted 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 re-encrypts this image using the assigned encryption key and the C5515/14/05/04 device-id, which is unique to each C5515/14/05/04 device. This re-encrypted image is then said to be *bound* to that specific C5515/14/05/04 device, and is written back to the external storage device that it was originally read from.
- Encrypted 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 C5515/14/05/04 device-id. Only a single C5515/14/05/04 device can decrypt this image.

The encrypted boot-image format is shown in Table 4.

Table 4. C5515/14/05/04 Encrypted 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)	1 to 2^{16} – 1 (not encrypted)
16	Register configuration #1 address in I/O space	Repeated according to register configuration count. Register configuration address is any valid register in C5515/14/05/04 I/O space. Address 0xFFFFis 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 configuration #1 value or delay count	
18	Register configuration #2 address in I/O space	
19	Register configuration #2 value or delay count	
	. . .	
	Register configuration #N address in I/O space	
15+2N	Register configuration #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 0x097FFF) MSW
18+2N	Destination LSW address to load Section 1 (32-bits)	Safer encrypted C55x 16-bit word address (0x000060 to 0x097FFF) 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

Table 4. C5515/14/05/04 Encrypted Boot-Image Format (continued)

Word	Content	Valid Data Entries
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 hex conversion utility (hex55) utility. The hex55 utility is intended for mass Catalog support.

3.3.1 Creating a Boot Image With Hex55 Utility

Use the following steps to create the boot table:

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 get 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 causes the bootloader to fail.
4. Specify the boot type: –parallel8, –parallel16, -serial16 or –serial8.
5. Specify the desired output format. For detailed information on the available hex conversion utility output formats, see the *TMS320C55x DSP Assembly Language Tools User's Guide* (SPRU280).
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 below.

3.3.1.1 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 are 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
-v5505          ;use C55x boot table format for TMS320C5515/14
-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
```

3.3.1.2 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 register address 0x1C8C with the value 0x0001
- After the register is configured, wait 256 cycles before continuing the boot
- Desired output is Intel format in a file 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
-v5505                       ;use C55x boot table format for TMS320C5515/14
-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 DSP Assembly Language Tools User's Guide* (SPRU280).

3.3.1.3 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
```

3.3.1.4 DOS Command Line for Generating Boot Image Using Hex55

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

3.4 Burn a Boot-Image

Once a boot image (*.bin) is generated, customers can burn the boot image into the NOR Flash, NAND Flash, 16-bit EEPROM, I2C EEPROM, 24-bit SPI serial Flash or multimedia card/secure digital (MMC/SD) card. It is done by a utility called programmer that runs on C5515/14/05/04 using an emulator with Code Composer Studio™ software. First, open the Code Composer Studio project - programmer.pjt as shown in Figure 1.

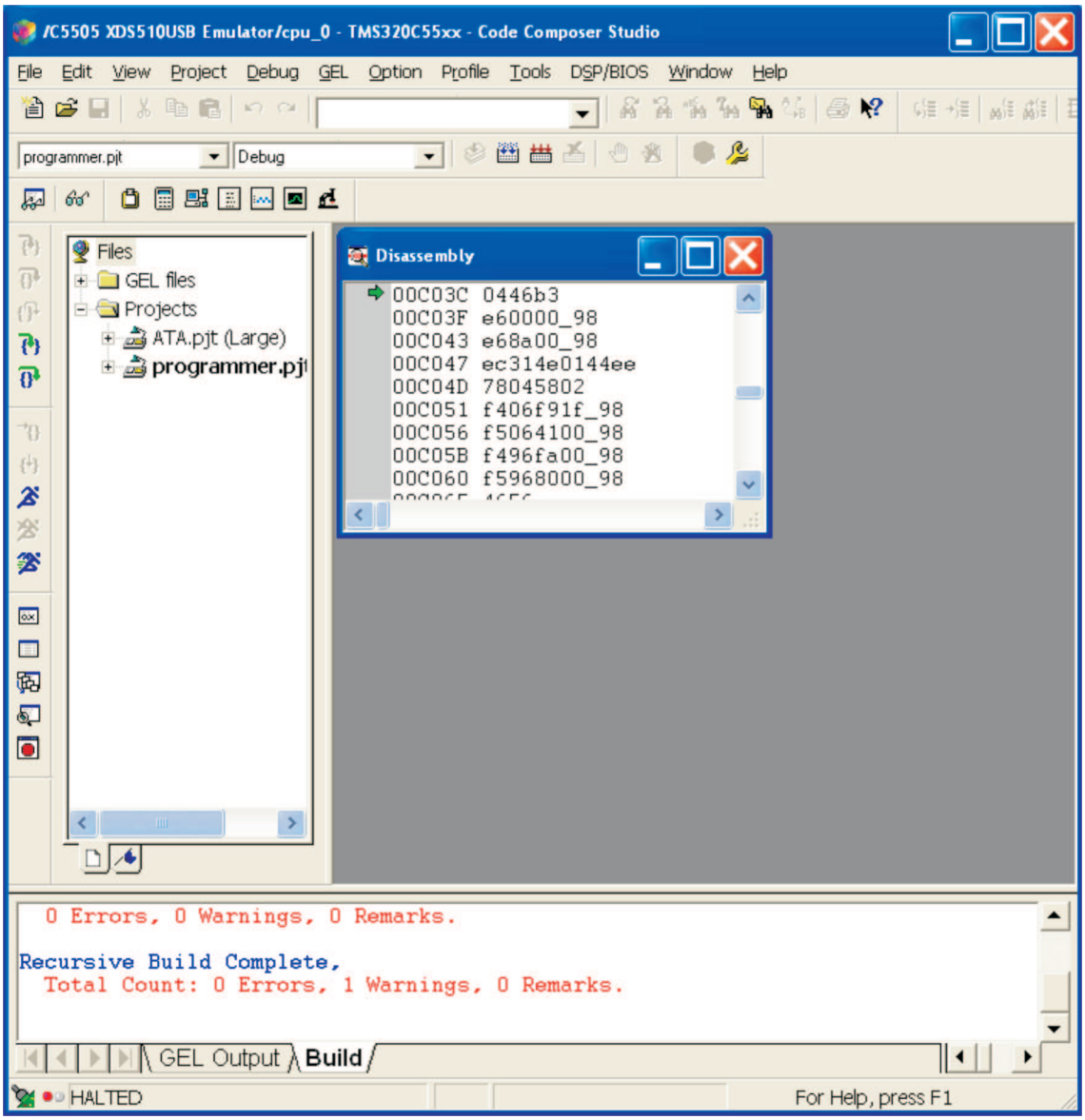

Figure 1. Open the Programmer.pjt in Code Composer Studio

It uses stdout and stdin to allow you to place the contents of a PC file (generally a boot-image) onto a peripheral device on the C5515 EVM. It supports NOR, NAND, SPI EEPROM (or SPI serial Flash), and I2C EEPROM, as well as MMC and SD if the card is properly formatted. This utility has been tested extensively on real silicon with C5515 EVM.

One way to write to NAND Flash at CS 2 would be to answer:

- *1* (choose device NAND Flash) to the first prompt
- *2* (choose the chip select CS2) to the second prompt
- *1C:\projects\flash_image\demo.bin* (choose write operation using `C:\projects\flash_image\demo.bin`) to the third prompt

Screen shots Figure 2 through Figure 5 demonstrate the multiple prompt procedure for writing `C:\projects\flash_image\demo.bin` to NAND Flash on ChipSelect 2.

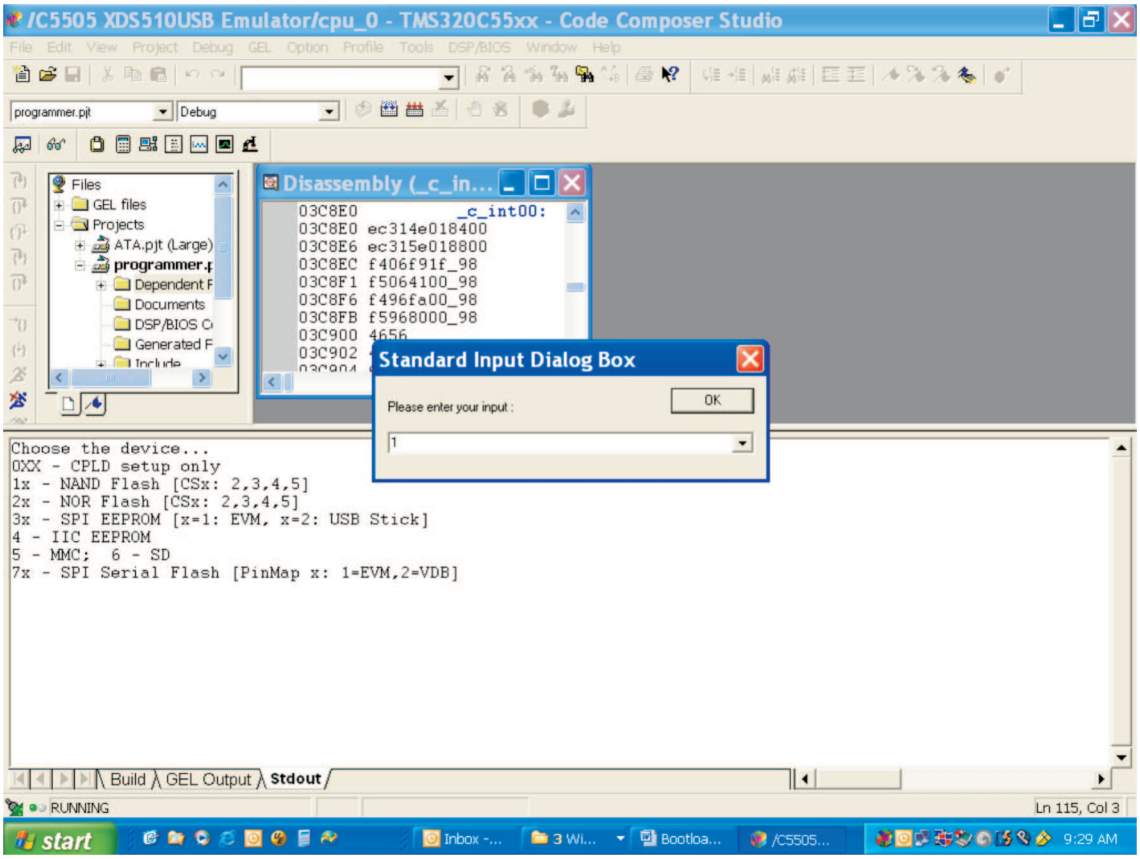

Figure 2. Standard Input Dialog Box for Programmer.pjt (Choose NAND Flash)

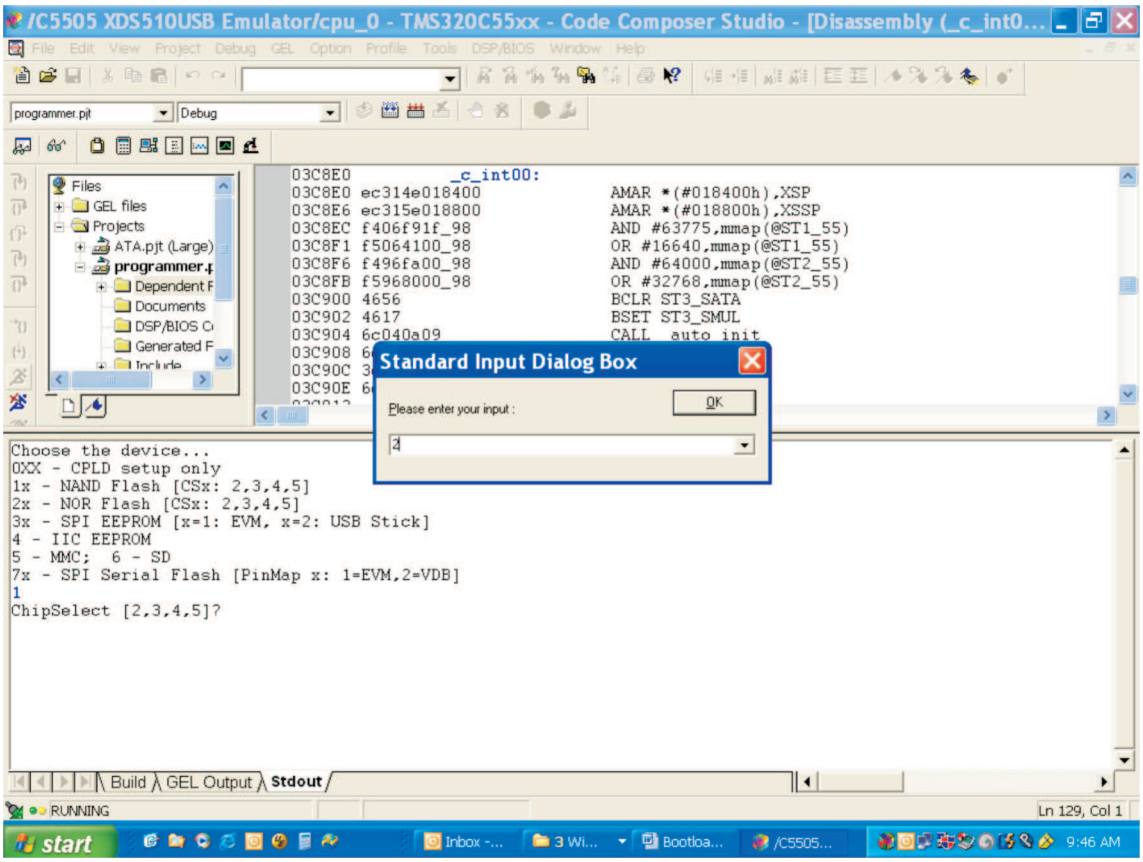

Figure 3. Standard Input Dialog Box for Programmer.pjt (Choose ChipSlect 2)

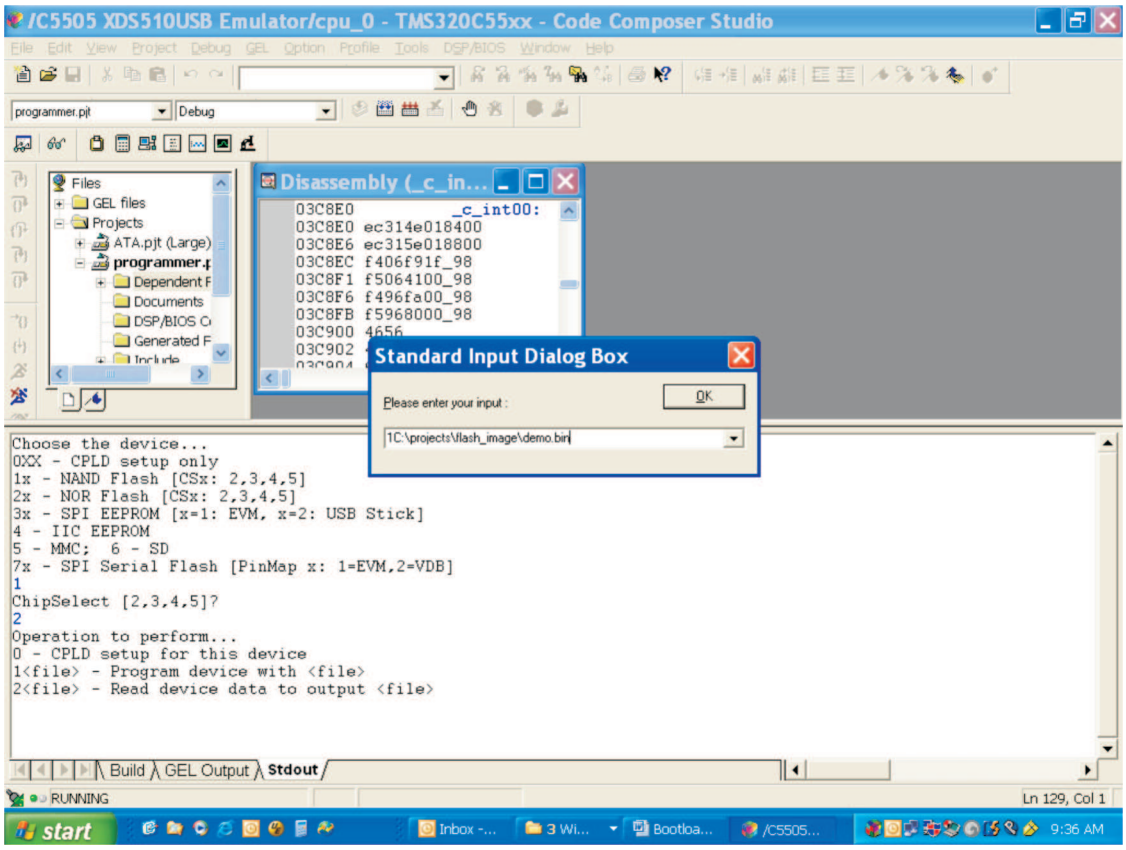

Figure 4. Standard Input Dialog Box for Programmer.pjt (Choose Program device with <file>)

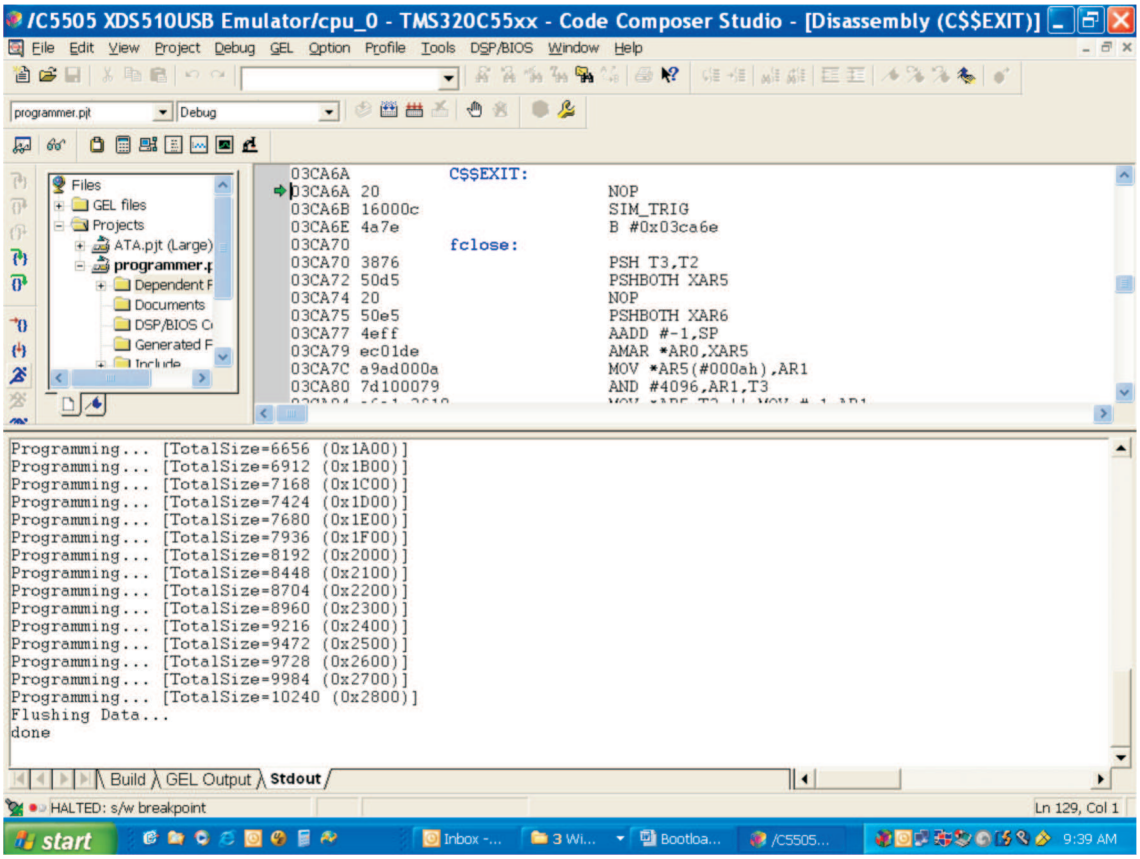

Figure 5. Flashing is Done

The programmer guides you on the stdout display and requests user input with a small popup window; load and run the program using Code Composer Studio. Then, you can either follow the directions from the programmer, or you can type in the answers to all prompts starting at the first prompt as a shortcut. For the previous example, you can also use the shortcut `121C:\projects\flash_image\demo.bin` at the first prompt.

The following are some examples, assuming that you want to load a file named `C:\projects\flash_image\demo.bin`.

For writing to NOR Flash at CS 3 on an EVM, at the first prompt enter: `231C:\projects\flash_image\demo.bin`.

For writing to SPI EEPROM on an EVM, at the first prompt enter: `311C:\projects\flash_image\demo.bin`.

For writing to SPI EEPROM on an ezDSP USB Stick Board, at the first prompt enter: `321C:\projects\flash_image\demo.bin`.

For writing to I2C EEPROM, at the first prompt enter: `41C:\projects\flash_image\demo.bin`.

For writing to MMC card, at the first prompt enter: `51C:\projects\flash_image\demo.bin`.

For writing to SD card, at the first prompt enter: `61C:\projects\flash_image\demo.bin`.

For writing to SPI serial Flash on EVM, at the first prompt enter: `711C:\projects\flash_image\demo.bin`

It takes a while for some devices to complete writing all data. Always wait for an error message or a *Programming Complete* message. To run the utility again, you need to use *Restart* (or reload).

4 References

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

Revision History

Changes from C Revision (July 2015) to D Revision	Page
• Update was made in the Abstract of this document.	1

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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