Application Report
SPRAAE9–September 2006

Preparing a TMS320C645x Application for I2C Boot Load

DSP Applications

ABSTRACT

This application report describes how to prepare a C645x application for the I2C boot load process.

The enclosed .zip archive contains all utilities and examples necessary to build a test application, program it into DSK6455's I2C ROM, change the boot mode to I2C boot load, and verify that the test application has been loaded from the I2C and is running correctly.

This application report contains project code that can be downloaded from http://www.ti.com/lit/zip/SPRAAE9.

Contents

List of Figures

1 Introduction

The on-chip ROM-based boot loader on C645x devices supports various modes for loading the application image into the DSP. These modes include boot via EMIF, I2C, PCI, HPI or SRIO (see device datasheet for a complete list). This application note describes how to prepare an application image for an I2C bootload, how to program it into DSK6455's I2C ROM device and how to verify that the application has been loaded successfully into the DSP by the on chip bootloader.

The following components are included:

- A test application which causes the LEDs on the 6455DSK to blink. The enclosed project includes a final build step which converts the resulting .out file into the I2C image.
- I2C Programmer. The utility runs on the DSP. The previously generated I2C image of the test application is loaded into CCS via Load Data feature and programmed into the I2C ROM by DSP-based I2C programmer.
- Utilities needed to convert an .out file into the I2C image suitable for the use by the I2C programmer.

The development environment consists of CCS3.2, compiler release 6.0.4 and DSK6455 using on-board emulation and a USB connection.

2 The Test Application and its I2C Image

The test application is the LED example from the 6455 DSK board support package (v1) from Spectrum Digital. When it runs successfully, LED0 blinks and LED3 is turned on or off depending on the value of SW1-3.

The test application project file is located in directory \led_c6455. It generates the COFF file \led_c6455\Debug\led.out. A final build step (see Project ->Build Options -> General) consists of running a batch file \led_c6455\i2c_convert.bat which generates \led_c6455\Debug\I2C_image\i2cromdsp.ccs. This file can be then used by the I2C programmer (see the next section).

This batch file first runs the hex6x utility, followed by a number of utilities located in /Utilities. It requires two input files:

- \led_c6455\Debug\I2C_image\led.rmd: This is the input command file for the hex6x utility. If the application project is modified, this file may need to be manually modified such that the "length" parameter in the ROMS section matches or exceeds the length of initialized sections in \led_c6455\Debug\led.map. Currently the length is set to 0xC000.
- \led_c6455\Debug\I2C_image\i2cparamtable.sec: This file tells one of the I2C utilities how to program the I2C parameter table (see [1]) for the primary I2C boot. For example, with the contents shown in the code example below, the I2C parameter entry at offset 0x0 is configured (param_index=0), the boot mode is programmed to I2C boot mode (boot_mode=5) and boot table load (options=1); the actual boot table will be stored starting at an fixed offset of 0x420 (which translates to Device Address (LSW) = 0x420) and the contents of the boot table will be retrieved from file led.i2c.ccs (which is an intermediate output file generated based on led.out).

```
section
{
  param_index   = 0
  boot_mode     = 5
  options       = 1

  core_freq_mhz    = 33
  i2c_clk_freq_khz = 50

  multi_i2c_id = 0
  my_i2c_id    = 1
  address_delay = 0
  exe_file = "led.i2c.ccs"
}
```

3 I2C Layout and Programming

The I2C ROM needs to contain a set of I2C boot parameters for the primary I2C boot and the actual test application boot table. The location of the I2C boot parameters for the primary I2C boot depends on the value of CFG[2:0] pins. This parameter table will further point to the location of the application code. The I2C boot parameter table format is described in [1].

With CFG[2:0] = 000b, one possible I2C ROM layout is shown in Figure 1. The I2C bootloader reads the I2C boot parameter table at offset 0x0, which points to the test application (starting at address 0x420). When the I2C bootloader completes, it starts executing the test application.

The I2C programming utility which programs the I2C runs on the DSP. The corresponding CCS project is located in /I2C_Programmer.

The utility programs data located at DSP memory address 0x900000 into I2C ROM on the DSK. It relies on the CCS' Data->Load feature to download data from a file into DSP memory address starting at 0x900000 prior to running the I2C Programmer.

The 32-bit value at address 0x900000 is the actual number of bytes to be programmed into I2C. The 32-bit value at the next word address, 0x900004, is the I2C address offset where the programming starts. The remaining locations are the actual data to be written to the I2C ROM.

The file led_c6455/Debug/I2C_Image /i2cromdsp.ccs, generated as the final build step when the test application is built (see the previous section), is already formatted for the use by the I2C programmer.

The I2C ROM layout is shown in Figure 1.

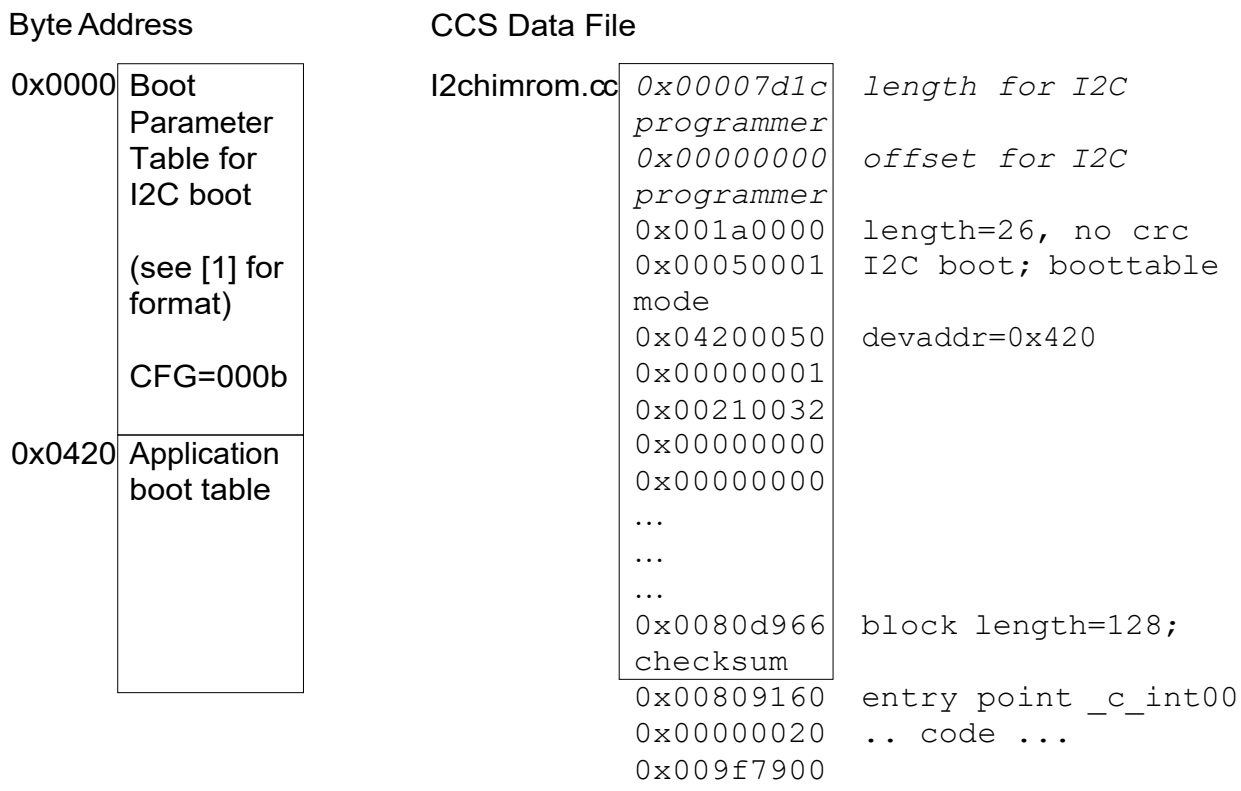

Figure 1. I2C ROM Layout and the Corresponding I2C Programmer Input File

To facilitate the I2C programming process, a GEL script is provided which loads the .out file, loads the .ccs file into memory and runs the program until the I2C programming has succeeded. A screenshot of the CCS session showing the GEL menu is shown in Figure 2.

In case that the I2C programming fails, the application continues to run. If the GEL script runs for more than a couple of minutes, the value of variable “status” should be observed in the Watch window and checked against failure codes from i2cprog.c.

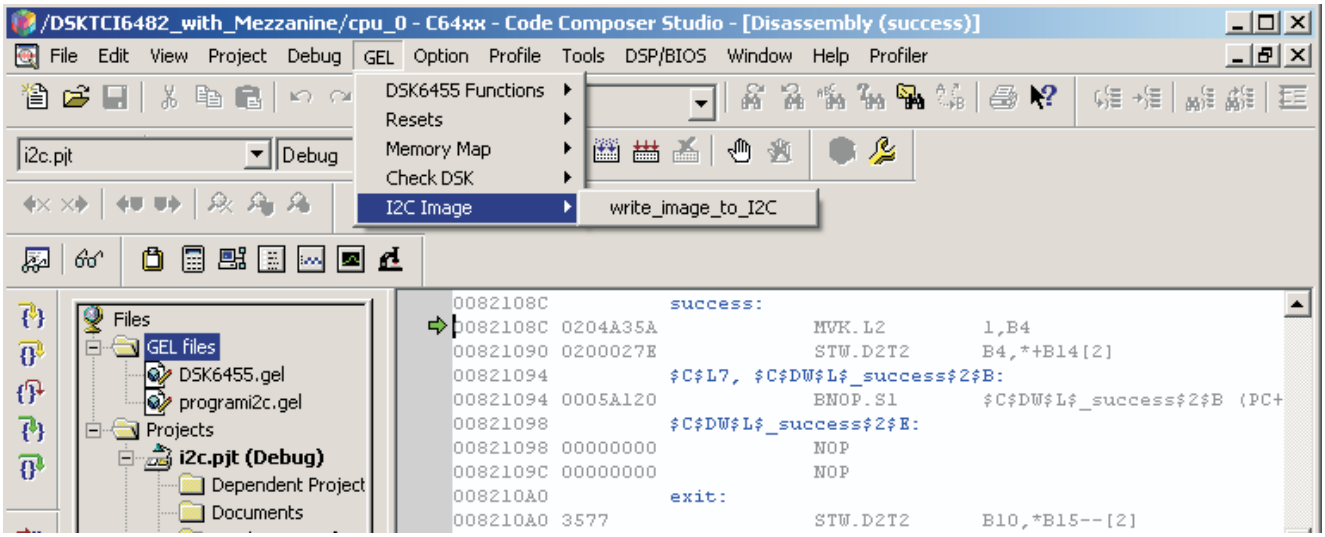

Figure 2. GEL Menu for I2C Programming

4 Test Procedure

Given the directory structure and files provided in the enclosed archive, following procedure can be followed to program the test application to the I2C ROM, run it, and verify its operation:

1. **Configure DSK's** SW3 for Little Endian, no boot, CFG[2:0]=000b (SW3-1, 2,3,4,5 off, SW3-6,7,8 on).
2. **Create** test application **image** (the original image is included in the enclosed archive).
   - Open the CCS project in /led_c6455.
   - Build. The output file is /led_c6455/*Debug/I2C_image/i2cromdsp.ccs*
   - Verify that the length parameter in the ROMS directive in */test_application/Debug/I2C_image/led.rmd* matches or exceeds the length of all initialized sections from */led_c6455/Debug/led.map*. If this is not the case, modify .rmd file according to the .map file and re-build the project.
3. **Program I2C**
   - Open I2C CCS Project in /I2C_Programmer.
   - Rebuild (optional since the .out is included in the enclosed archive).
   - Debug -> Connect.
   - File->Load GEL -> /led_c6455/Debug/I2C_Image/programi2c.gel.
   - Open the GEL file and verify that the length parameter for the GEL_MemoryLoad() command matches the length parameter in the first line of */led_c6455/Debug/I2C_image/i2cromdsp.ccs*. If this is not the case, modify the length parameter in the GEL file according to the .ccs file and reload the GEL.
   - GEL -> I2C Image -> write_image_to_i2C (may take a couple of minutes until the execution halts).
   - Close CCS
4. **Run the application**
   - Power down DSK; change boot mode to I2C boot (SW3-2 on, SW3-4 on).
   - Power up DSK. The LED should start blinking after the I2C boot process completes (may take 40-50 seconds).

5 Endianess

The source code for the test harness is provided in little endian although a quick test in big endian has been performed.

To change endianess:

- Change build options in CCS projects.
- Change the global endian setting in the DSP/BIOS config file, where applicable.
- Change options in the .rmd command files from –order M to –order L.
- Change options in the batch and make files from –le to –be.

6 References

TMS320C645x Bootloader User's Guide (SPRUEC6)

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