Bootloading the TMS320VC5402 in HPI Mode

Scott Tater *DSP Applications – Semiconductor Group*

ABSTRACT

The TMS320VC5402 bootloader allows the system designer flexibility in memory configuration by providing many methods to boot the processor out of reset. One commonly used boot procedure allows a host microprocessor to load code into the TMS320VC5402 using the Host Port Interface (HPI) peripheral. This application report describes the HPI boot method and provides an example using the TMS320VC5402 DSP Starter Kit (DSK).

While this application report specifically addresses the TMS320VC5402 bootloader, much of the information is applicable for other members of the TMS320C54x™ DSP generation. Project collateral discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SPRA382.

Contents

Figures

1 Introduction

This section describes the TMS320VC5402 (hereafter referred to as the C5402) bootloader and gives an overview of the steps involved in performing a bootload operation through the Host Port Interface (HPI). It also briefly describes the C5402 DSP Starter Kit (DSK) functionality, which will be used as a demonstration platform.

1.1 C5402 Bootloader

The C5402 bootloader is used to transfer code from an external source into either internal or external memory. The bootloader allows the user flexibility in system design by storing code in cost-efficient, nonvolatile external memory. This enables the designer to avoid a custom on-chip ROM mask and to reduce system cost.

The bootloader is a flexible program that supports multiple boot methods and code sources. In addition to multiple types of parallel and serial boot modes, it is possible for a host microprocessor to download code and bootload the C5402 using the HPI.

This document describes the HPI boot and presents a method to successfully boot the C5402 using the HPI. For a complete discussion of all bootload modes and the C5402 ROM contents, see the *TMS320VC5402 and TMS320UC5402 Bootloader* application report (literature number SPRA618).

1.2 C5402 HPI Boot Process

The C5402 contains 4K words of on-chip ROM. A portion of this ROM is used to store the bootloader code. The MP/ $\overline{\text{MC}}$ bit of the Processor Mode Status (PMST) register is sampled at reset and its value partially determines the configuration of the C5402 memory map. If MP/ $\overline{\text{MC}}$ is set low (MP/$\overline{\text{MC}}$ = 0), then the C5402 is set for microcomputer mode and the bootloader will start execution following reset. The ROM-coded bootloader is located at program memory address 0xF800.

For a complete description of the C5402 reset process, memory map, and control registers, see the *TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals* (literature number SPRU131).

The C5402 executes its bootloader after it has left reset, unlike some members of the TMS320C5000™ DSP platform. On execution, the bootloader begins polling different resources to determine which boot mode is active. The bootloader uses various control signals including interrupt signal, the $\overline{\text{BIO}}$ and XF pins, and data in on-chip memory to configure and control the boot process. If no boot mode is found active, the bootloader will cycle continuously, checking each mode in turn until one is selected.

There are two methods to signal the bootloader that HPI boot is active: interrupt two and data memory location 0x007F. The bootloader checks to see if the interrupt two (INT2) flag in the Interrupt Flag Register (IFR) is set to one (active). If the $\overline{\text{INT 2}}$ pin is active, then HPI mode is selected. The bootloader also clears location 0x007F and uses it as a software flag to show that the HPI boot is completed. HPI boot mode can also be selected without using INT2 by completing the code-transfer process and writing location 0x007F to signal that the boot is completed.

TMS320C5000 is a trademark of Texas Instruments.

If the $\overline{\text{INT 2}}$ signal is used to activate the HPI boot mode, there are two methods to obtain an input pulse on the $\overline{\text{INT 2}}$ pin:

- Tie the Host Interrupt ($\overline{\text{HINT}}$) pin directly to the $\overline{\text{INT 2}}$ pin
- Generate a valid interrupt signal on $\overline{\text{INT 2}}$ within 30 clock cycles from when the C5402 fetches the reset vector

Note that the $\overline{\text{HINT}}$ pin generates a valid interrupt signal at the start of the bootload process.

The host processor uses the HPI port to send program code to the C5402 during the HPI boot. When the host is finished, it writes the code entry point to location 0x007F. The bootloader will recognize this change, branch execution to the entry point, and terminate, leaving the transferred code in control.

1.3 C5402 DSP Starter Kit

This section presents code that has been tested for the C5402 DSP Starter Kit (DSK) from Texas Instruments (TI). The DSK is a platform that is useful for prototyping code on actual hardware. It contains a C5402 DSP, SRAM and Flash memory, and a number of external interfaces. For a detailed description of the DSK, see the TI website at www.ti.com. Note that the DSK has eight DIP switches to aid in hardware configuration.

The following settings are required to prepare the DSK for HPI booting as described in this application report:

- Ensure that the DSK is connected to the host PC using the parallel port interface
- Select the C5402 microcomputer mode by setting the MP/ $\overline{\text{MC}}$ pin low. Using the DIP switch unit, set **switch #2** to the **ON** (or down) position.
- (**Optional**) Enable debug by connecting a JTAG emulator (such as the TI XDS510™) to the 14-pin JTAG header on the DSK. Then, configure the DSK parallel port interface to allow an emulation connection through the JTAG header. Using the DIP switch unit, set **switch #1** in the **OFF** (or up) position. Code Composer Studio™ (CCS) integrated development environment (IDE) can then be used to monitor the HPI boot process. Note: Be sure that the specific board configuration file loaded in the Code Composer Studio setup does not load a General Extension Language (GEL) file that alters these configuration settings (especially the MP/ $\overline{\text{MC}}$ bit). See the *Code Composer Studio User's Guide* (literature number SPRU328) for complete details.

XDS510 and Code Composer Studio are trademarks of Texas Instruments.

2 Software Process for HPI Boot Mode

Complete these four steps to set up and execute a C5402 HPI bootload operation:

1. Generate an output (.out) Common Object File Format (COFF) file
2. Parse the .out file to extract destination information and code
3. Write parsed code to the C5402 HPI to load on-chip memory
4. Write the entry point to memory location 0x007F on the C5402

This application report has an associated Zip archive that may be downloaded from the same web location as this document. The archive contains the files that are mentioned in the given examples.

2.1 Generate an Output (.out) COFF File

The bootloading process starts with an executable object (.out) file produced using Code Composer Studio. The .out file is generated from any desired user source code and is the code that will be loaded onto the C5402 during the boot process.

2.2 Parse the .out File to Extract Destination Information and Code

The .out file generated in the first step follows the Common Object File Format (COFF). COFF is an implementation of the AT&T™ COFF format. It is a modular format that allows the user flexibility in managing code segments and target system memory. The complete COFF file format is described in the *TMS320C54x Assembly Language Tools User's Guide* (literature number SPRU102).

Because the COFF file is modular, it cannot be loaded directly into the C5402 memory. That is, the .out file is not an exact memory image. We must first parse the COFF file to reconstruct the image. A method to do this is described in the *Extracting Equivalent Hex Values From a COFF File* application report (literature number SPRA573). *Extracting Equivalent Hex Values* provides example source to produce a text listing that contains destination address, data length, and data from each section in the COFF file.

2.3 Write Parsed Code to the C5402 HPI to Load On-Chip Memory

Using the parsed text listing produced in step two (see Section 2), it is possible to transmit this data to the C5402 memory. This will start the actual HPI boot process.

The best procedure will vary by specific implementation since each host processor will vary in its interface to the C5402 HPI. Regardless of implementation, any host processor will need to perform two operations in order to complete this step of the HPI boot: read the listing file and transmit it to the C5402.

AT&T is a registered trademark of AT&T Corporation.

Note: This application report includes example code that uses the C5402 DSK host-side library, evmdsk54dll.lib. The DSK library contains communication routines that allow a host PC access to the DSK through its parallel port connection. Complete library documentation is available in the Code Composer Studio help file and in the library include file, evmdsk54dll.h.

Appendix A shows the listing for this example, hpi-boot.cpp. The executable for this code is available on the TI website along with this application report. This program shows one way to read the listing file, initialize the DSK, and transmit the program code to the DSK using the evmdsk54dll libraries.

2.4 Write the Entry Point to Memory Location 0x007F on the C5402

After the memory of the C5402 has been initialized, the last step in the boot process is to write the program entry point to memory location 0x007F. The C5402 bootloader will detect the change, branch to the indicated value, and start program execution. The HPI boot is now done and the C5402 will run normally.

The entry point may be obtained from the memory map (.map) file produced during the linking stage of .out file generation.

3 Example: HPI Boot Using the DSK Blink Demo

The following techniques can be used to boot the C5402 using the HPI port with a DSK code example demo. This example uses the C5402 DSK and requires that the DSK be configured as described in Section 1.3 of this document. It also requires Code Composer Studio version 1.20 or greater.

All example files referenced in this section are found in the archive associated with this application report. They can be downloaded from the Texas Instruments web site in the same location as this application report. The archive contains:

- HPI boot code in MS VC++ 6.0 project format
- DSK Blink demo in subdirectory \blink
- COFF Hex Extraction Utility in subdirectory \hpi-boot

All programs in this example are run out of subdirectory hpi-boot in the associated archive. Using the COFF Hex extraction utility, coff_both, parse blink.out to produce a text listing. On the DOS command line, type:

"coff_both –out blink.out"

Coff_both will prompt for the COFF type. Enter "2" because both Code Composer Studio versions 1.20 and 2.0 produce type two COFF files. The generated file will contain the hex listing and be entitled "blink.out.c". An excerpt from its contents is shown in Figure 1.

```
Section = .vers
src_addr = 0x0
length = 0x39 (57)
dest_addr = 0x0
space = 0

    0x0046,  0x0072,  0x0069,  0x0020,
    0x004D,  0x0061,  0x0079,  0x0020,
    0x0031,  0x0032,  0x0020,  0x0030,
    0x0039,  0x003A,  0x0032,  0x0034,
    0x003A,  0x0032,  0x0031,  0x0020,
    0x0032,  0x0030,  0x0030,  0x0030,
    0x0000,  0x0062,  0x006C,  0x0069,
    0x006E,  0x006B,  0x005F,  0x0062,
    0x0069,  0x006F,  0x0073,  0x002E,
    0x0063,  0x0064,  0x0062,  0x0000,
    0x0040,  0x0028,  0x0023,  0x0029,
    0x002A,  0x002A,  0x002A,  0x0020,
    0x0067,  0x006C,  0x0075,  0x0065,
    0x002D,  0x0064,  0x0030,  0x0039,
    0x0000,

checksum = 0x  57
```

Figure 1. Excerpt From Hex Extraction Output (blink.out.c)

Next, use the host-side bootloader code, hpi-boot, to read the extracted hex file and write the information to the DSK. Source code for this program is presented in Appendix A. On the DOS command line, type:

"hpi-boot blink_bios.out.c"

Note that the bootloader code is set up with the entry point of the DSK Blink demo. Other programs may require you to change this code in order to use a different entry point. Entry point information is best obtained from the .map file produced during the Code Composer Studio linking stage.

This completes the HPI boot process. Status information will also be displayed while the program is running. Verify that the boot was successful by observing the blinking LEDs on the DSK.

References

1. *TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals* (literature number SPRU131)
2. *Extracting Equivalent Hex Values From a COFF File* application report (literature number SPRA573)
3. *TMS320C54x Assembly Language Tools User's Guide* (literature number SPRU102)
4. *TMS320VC5402 and TMS320UC5402 Bootloader* application report (literature number SPRA618)
5. *Code Composer Studio User's Guide* (literature number SPRU328)
6. *TMS320VC5402 Fixed-Point Digital Signal Processor* data sheet (literature number SPRS079)

Appendix A. Code Listing

```
// hpi-boot.cpp
/*************************************************************************\
*-------------------------------------------------------------------------
* FILENAME. .....hpi-boot.cpp
* DATE CREATED. 01/21/2002
* LAST MODIFIED. XXXX
*-------------------------------------------------------------------------
* This program demonstrates a method to boot load the 'C5402 using the
* Host Port Interface (HPI). The 'C5402 DSP Starter Kit (DSK) is used as a
* reference platform. A parsed COFF file is read into memory and transmitted
* to the 'C5402 through the HPI port.
*
* Host side libraries and include files have default root directory (CCS v2.0):
* \ti\c5400\dsk5402\host
*
* The Entry Point is hard wired in this code. It may be beneficial for the user
* to change this feature. It is set for the DSK Blink Demo at 0x500.
*
\*************************************************************************/
#include "stdafx.h"

//Globals
HANDLE hd; //DSK Handle
LPVOID hm; //HPI Handle

EVMDSK54X_BOARD_TYPE boardType = TYPE_C5402_DSK;
EVMDSK54X_OPEN_TYPE  openType = EVMDSK54X_PARALLEL_OPEN;

USHORT  port = 0x378;              // parallel port I/O address //

int DSK_init();
int DSK_close();

int main(int argc, char* argv[])
{
        FILE *fp;
        char line[80];
        char tokA[20], tokB[20], tokD[20]; //strings used to tokenize input
        int length, destaddr;
        short data[20000]; //Buffer used to hold data sections. Increase for large code
        short prog_entry=0x0500; //a default entry point

        int i,flag,eof_flag;


        if (argc == 1) { //error
                printf ("Correct Syntax: hpi-boot FILENAME\n");
                printf ("where FILENAME is a parsed COFF file\n");
                exit(1);
        } else {
                if ((fp = fopen(argv[1], "r")) == NULL)
                        printf ("File could not be opened for reading\n");
        }


        if (DSK_init()) //Initialize the DSK--open the DSK, HPI port, and configure
                exit(-1);
```

```
        eof_flag=0; //shows if end of sections reached

        //This section scans for a new data section in the parsed file.
        //The new section is read into memory and then written to the DSP
        //When there are no more sections to read, the entry point is
        //writen to stop the boot process
        while (!eof_flag){
                while ((fscanf(fp, "%s", tokA) != EOF) && (flag =(strncmp(tokA, "Section",
7)) != 0));
                if (!flag) {

                    fgets(line, sizeof(line), fp); //read src addr line
                    fgets(line, sizeof(line), fp); //read length line
                    fscanf(fp, "%s %s %x %s", tokA, tokB, &length, tokD);

                    fscanf(fp, "%s %s %x %s", tokA, tokB, &destaddr, tokD); //read dest addr


                    fgets(line, sizeof(line), fp); //read spc addr line
                    fgets(line, sizeof(line), fp); //read blank line


                    for (i=0; i< length; i++){ //reads and records data values
                                fscanf(fp, "%x,", &data[i])
                    }

                 // write single program section to 'C5402
                 printf("\n**    Writing data section for HPI boot\n");

             printf("Section Length: %#x\n", length);

             printf("Destination Address: %#x\n\n", destaddr);
                        if (!evmdsk54x_hpi_write(hm, (PULONG) data, (PULONG) &length,
(ULONG)destaddr, PROG_MEMORY, 0))
                        {
                                printf(" -- failed");
                                return (-1);
                        }

                }
                else {
                        eof_flag = 1;
                        //  write prog start address in data mem 0x007F
                        printf("**    Writing program entry point to complete HPI boot\n");

             printf("Entry point: %#x written at location 0x007f\n", prog_entry);
                        if (!evmdsk54x_hpi_write_single(hm, (PULONG) &prog_entry, (ULONG)
0x7F, DATA_MEMORY, 0))
                        {
                                printf(" -- failed");
                                return (-1);
                        }

                        //  ** End of standard initialization for 5402 DSK HPI access and
HPI booting **
        }
        }
                        printf("\nClosing...\n\n");
                        if (DSK_close())
                         exit (-1);
```

```
        return 0;
}


int DSK_init() {

        //  ** Standard initialization for 5402 DSK HPI access and HPI booting **

        //  open a communication port with the
        //  DSK parallel port interface
        printf ("\n**   Initializing DSK communication channel");
        if ((hd = evmdsk54x_open(port, boardType, openType, 1))
                == INVALID_HANDLE_VALUE)
        {
                                                printf ("!! Error:\n");
                printf ("evmdsk54x_open failed");
                return (-1);
        }

        //  open a communication port with the HPI
        printf ("\n**   Opening HPI");
        if (!(hm = evmdsk54x_hpi_open(hd)))
        {
                                                printf ("!! Error:\n");
                printf ("evmdsk54x_hpi_open failed");
                return (-1);
        }

        //  hold the DSP in reset
        printf("\n**    Resetting DSP");
        if (!evmdsk54x_reset_dsp(hd, (ULONG)0x00))
        {
                                                printf ("!! Error:\n");
                printf ("evmdsk54x_reset_dsp failed");
                return (-1);
        }

        //  take the DSP out of reset
        printf("\n**    Taking DSP out of reset");
        if (!evmdsk54x_unreset_dsp(hd, (ULONG) 0x00))
        {
                                        printf ("!! Error:\n");
                printf ("evmdsk54x_unreset_dsp failed");
                return (-1);
        }

        //  disable on board flash access to ensure HPI
        //  memory access will not affect the on-board flash
        printf("\n**    Disabling FLASH access");
        if (!evmdsk54x_flash_access(hd, FALSE))
        {
                                      printf ("!! Error:\n");
            printf ("evmdsk54x_flash_access failed");
            return (-1);
        }
        printf("\n**    DSK Init Passed\n\n");
        return 0;
}


int DSK_close() {
```

```
        //  close HPI communication port
        printf ("**      Closing HPI\n");
        if (!evmdsk54x_hpi_close(hm))
        {
                                        printf ("!! Error:\n");
            printf ("evmdsk54x_hpi_close failed\n");
            return(-1);
        }

        //  close DSK communicaiton port
        printf ("**      Closing DSK communication channel\n");
        if (!evmdsk54x_close(hd))
        {
                                        printf ("!! Error:\n");
            printf ("evmdsk54x_close failed\n");
            return(-1);
        }
        return(0);
}
```

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