Application Report
SPRAAS8–February 2008

Using ROM Contents on TMS320C672x

Giancarlo Parodi

ABSTRACT

This application report explains how to configure an application based on pure software libraries and routines and how to properly use the provided ROM contents for the TMS320C672x DSP generation of devices. Several software examples are provided, explaining the process in progressive steps. Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SPRAAS8.

Contents

1 Introduction

This generation of C672x DSPs is well suited for audio-related signal processing tasks, thanks to its native floating-point data format support, among other key factors like its improved floating-point DSP core, and the new dMax DMA transfer engine, which has been architecturally optimized for embedded systems dealing with usual audio-related requirements, like latency.

However, another very interesting and appealing feature of this generation of devices is the possibility to leverage the ROM-located application software and libraries, like the TI DSP/BIOS™ (software kernel foundation) real-time operating system libraries, the C672x DSP library and the FastRTS library. This enables the application developer to save important memory resources by avoiding the need to link the content in the application executable code.

Another advantage is the faster execution time of specific routines because of their placement in the internal memory of the device, in comparison to locating this same code in external memory. This leaves the developer the possibility of using the spared memory for other time critical application data or code.

This application report discusses a basic example starting from an application which is completely based on software libraries (without using any of the available ROM contents), moving gradually to a final target software using all the ROM modules available in the C672x generation of DSPs.

2 Prerequisites

For developing software for the C672x devices, Code Composer Studio™ software is required. For Code Composer Studio release 3.1, you can install a separate chip support package that can be downloaded by registered users from the update advisor of Code Composer Studio (see help → update advisor → check for updates menu).

Code Composer Studio release 3.3 (including the free evaluation version) supports the C672x devices, without installing any additional patches.

The DSP library, FastRTS library, the ROM versions of the libraries, and the required ROM system patch have been placed in the \libs directory of the example code. Similarly, the required header files from the DSP lib used in the example are located in the \include directory of the project files included with this document. The complete libraries can be downloaded from the TI web at (http://www.ti.com).

Note: Even though the software examples are simple, test them on real hardware because the C672x simulator does not include ROM contents.

3 Internal Memory Map Architecture and Bootloader Requirements

The C672x DSP internal memory address space is logically divided in four distinct sections: the memory mapped peripherals, two separate ROM banks (Bank 0 and Bank1), plus one RAM bank. The address ranges and sizes are as follows:

```
0x0000 0000 – 0x0001 FFFF : ROM bootloader code
0x0002 0000 – 0x0002 BFFF : ROM DSP library
0x0002 C000 – 0x0002 FFFF : ROM fast RTS library
0x0003 0000 – 0x0005 FFFF : ROM DSP BIOS
{
      0x1000 0000 – 0x1000 0FFF : Reserved at Boot for Bootloader
      0x1000 1000 – 0x1000 1AFF : DSP/BIOS reserved RAM
      0x1000 1B00 – 0x1000 1BFF : FastRTS reserved RAM
      0x1000 1c00 – 0x1003 FFFF : always user RAM
}
```

4 ROM Bootloader

In the C672x devices, a ROM bootloader is present and responsible for the load and execution of the user application code at startup. The bootloader program is automatically executed following a device reset and depending on the boot mode desired (inter-intergrated circuite (I2C), serial peripheral interface (SPI), parallel Flash, host port interface (HPI)), different device configurations and subsequent operations are performed. At the end of the boot phase, the bootloader allows the DSP to jump to the user application entry point for starting the user application. For more information, see *Using the TMS320C672x Bootloader* (SPRAA69).

However, like all programs, the bootloader also needs some internal RAM memory space to store its status information and working variables. Reserve the memory range from address 0x10000000 to 0x10000FFF either in the user linker command file or in the memory section of the .tcf BIOS configuration file.

There is another interesting feature that is linked to the bootloader, which relates to the possibility of silicon bugs that might be discovered on a specific silicon revision of the device during its lifetime.

The C672x ROM bootloader includes the possibility of executing additional code, provided in the form of an object library and linked with the DSP application, to apply what is called a *system patch*; this will have an impact on all silicon issues in which a software workaround might be applicable.

In fact, for all C672x devices, TI is already providing an object code, which must be linked with every application, in the same way you would link every other library or object file.

This is an important and required step, which is necessary for the application to function properly. To ensure this, add the object code directly in the Code Composer Studio project by selecting project → add in CCS, browse for the location of the file on the hard drive, and confirm the choice by selecting OK.

To emphasize the need for the system patch to be applied, this application report specifies that the system patch must be added directly in the linker command file.

Having added the module, the bootloader then jumps to the code section and executes this code at device reset before proceeding with the boot sequence specified in *Using the TMS320C672x Bootloader* (SPRAA69).

5 DSP Library

The DSP library is a highly optimized and code efficient collection of routines to perform several common mathematical computations used for DSP processing like FFT transforms, vector products, doing a matrix inversion, etc. The complete list of user available functions is provided in the *TMS320C67x DSP Library Programmer's Reference Guide* (SPRU657). This DSP library is compatible and can also be used for C672x devices.

6 DSP BIOS

DSP BIOS is a royalty free, TI-developed real-time operating system that supports, among others, the complete platform of C6000™ devices including the new C672x generation. For additional details about the different modules available, and a description of the features that this operating system (OS) can provide (for example, interrupt processing, task management, data analysis and logging features, etc.), see the *TMS320 DSP/BIOS User's Guide* (SPRU423) and the *TMS320C6000 DSP/BIOS 5.32 Application Programming Interface (API) Reference Guide* (SPRU403).

7 FastRTS

The FastRTS library is a code and performance optimized collection of the software routines related to the standard functions available in the C language standard library. These can be referenced in well-known header files like <stdio.h> and <stdlib.h>, or <math.h>, for example, trigonometric functions as cos(x) or log(x), or I/O related functions like printf().

8 The Example Application

The example test application source code requires some basic features of the previously mentioned ROM-based libraries, for the purpose of demonstrating their usage.

The project to start with is included in the simple_app folder, provided with this document.

You are going to perform a simple search of the maximum and minimum of a given vector of data, perform a calculation of one trigonometric function (cosine), then print a character string on the standard output by using the well known C printf() function.

In a second stage, the same application code is included in a BIOS task to demonstrate the usage of the BIOS ROM contents.

On completion of these tasks, all of the libraries of interest will have been used.

9 Specifying the Linker Command File

The first compiled application that is used as a starting point is completely *ROM-less*, not using any of the available ROM modules.

The linker command file instructs the linker about the details of the memory configuration and where to place the various initialized/non-initialized sections generated by the compiler.

External memory is ignored in this example since it does not introduce any relevant changes or considerations to this application report. For a thorough description of the linker command file syntax and its usage, see the *TMS320C6000 Optimizing Compiler v 6.0 Beta User's Guide* (SPRU187) and the *TMS320C6000 Assembly Language Tools v 6.0 Beta User's Guide* (SPRU186).

The following code looks like the first version of the linker command file:

```
-c
-heap  0x2000
-stack 0x4000

/* link with the C67x+ rts library */
-l rts67plus.lib

/* link the software system patch into the application */
/* required for proper device operation */
-l ..\libs\applySystemPatch.obj
-l ..\libs\c672xSystemPatchV2_00_00.lib

/* link in the c67x sw DSP lib */
-l ..\libs\dsp67x.lib

MEMORY
{
/* ROM memory map */
IROM_BOOT:   o   =   00000000h   l   =   00020000h
IROM_DSPLIB:  o   =   00020000h   l   =   0000C000h
IROM_FASTRTS:  o   =   0002C000h   l   =   00004000h
IROM_BIOS: o = 00030000h l = 00030000h

/* RAM memory map */
     /* reserved at boot for bootloader */
     IRAM_BOOT:       o = 10000000h   l = 00001000h

     /* internal RAM available for user applications */
     IRAM_USER:       o = 10001000h   l = 0003F000h

}

SECTIONS
{
    /* C standard sections */
    vectors  >    IRAM_USER
    .text    >    IRAM_USER
    .stack   >    IRAM_USER
    .far     >    IRAM_USER
    .switch  >    IRAM_USER
    .tables  >    IRAM_USER
    .data    >    IRAM_USER
    .bss     >    IRAM_USER
    .cinit   >    IRAM_USER
    .sysmem  >    IRAM_USER
    .const   >    IRAM_USER
    .cio     >    IRAM_USER
    .pinit   >    IRAM_USER
}
```

The *MEMORY* part defines the address ranges of the internal physical memory mentioned previously in this document, whereas, the sections listed in the *SECTIONS* part are the standard C compiler generated sections. Note that the linker is being instructed to put everything into internal memory.

10 Compiling Options

Specific optimizations will not be used since this is out of the scope of this application report.

11 Checking the Outcome

After rebuilding the example project, observe the map file (.map) placed in the \debug directory of the project. This linker output details the complete list of variable names and functions allocated in memory, together with interesting information about the address ranges where the program and data sections have been placed, and their size.

To generate this output, the linking option -m"map_filename" must be provided to the code generation tools (linker). As you might notice, this has already been done by populating the field in project → build options → linker → map file map file. You can specify use a custom file name.

The following map file documents that no ROM content has been used in the application, but rather all the application code is placed in the RAM memory space. You can deduct this by checking the address ranges indicated, which will be located in internal RAM.

Below is a snapshot of the map file output.

```
MEMORY CONFIGURATION

   name         origin    length      used     unused   attr    fill
------------- --------  ---------  --------  --------  ----  --------
  IROM_BOOT     00000000   00020000  00000000  00020000  RWIX
  IROM_DSPLIB   00020000   0000c000  00000000  0000c000  RWIX
  IROM_FASTRTS  0002c000   00004000  00000000  00004000  RWIX
  IROM_BIOS     00030000   00030000  00000000  00030000  RWIX
  IRAM_BOOT     10000000   00001000  00000000  00001000  RWIX
  IRAM_USER     10001000   0003f000  00010567  0002ea99  RWIX
```

The *used* column indicates that the linker is using only internal RAM.

12 Using the ROM Version of the DSP Lib

The relevant project for this section is provided with this document in the folder simple_app_dsplib_rom.

You need to modify the linker command file by adding the library file named c67xdsplibR.lib. This library makes sure that the appropriate references (addresses) in the application code are included, which enables the code to jump to the ROM sections. In addition, you need to instruct the linker to locate the functions in the ROM address range by modifying the sections part to reference the correct memory range.

Include the following directive in the memory section of the linker command file:

```
.dsplib : > IROM_DSPLIB, type = NOLOAD
```

Replace the DSP lib in the list of the linked libraries with the following ROM version patch:

```
/* include for using the ROM version of the DSPlib */
-l ..\libs\c67xdsplibR.lib
```

(See the file link_nobios_dsplib_rom.cmd).

Note: The directive in the section part is not necessarily needed, but it helps to make the sections allocation more explicit. In the linker command file, the NOLOAD directive needs to be specified. This directive informs the linker that the *load* and *run* addresses will coincide since the ROM memory contents are not going to be copied; this memory is already internal to the DSP.

In this case, you can observe the changes that are introduced in the generated map file; the ranges for the DSP library function DSP_max have changed and now point to a ROM address.

If you look at the details of the map file, you will see how the *patch* functions have been introduced into the executable to implement the *trampoline* functions to the ROM addresses:

```
FAR CALL TRAMPOLINES

callee    addr      tramp     addr      call addr  call info
--------  --------  --------  --------  ---------  ----------------
_DSPF_sp_minval
          000243a0  .T$0001   100086e0  100048a0   main.obj (.text)
_DSPF_sp_maxval
          000240e0  .T$0000   100086c0  10004870   main.obj (.text)
_DSPF_sp_vecmul
          00024a60  .T$0004   10008740  10004980   main.obj (.text)
_DSPF_sp_blk_move
          000252a0  .T$0002   10008700  100048e0   main.obj (.text)
_DSPF_sp_w_vec
          000248e0  .T$0003   10008720  10004928   main.obj (.text)
```

The DSP lib functions, called from the main file, are now located to a ROM address. Also, the amount of internal RAM used has decreased from 0x10567 to 0xffa7 (1472 bytes have been saved).

13 Using the ROM Version of FastRTS Lib

You can do the following:

- Link the FastRTS library patch before the link command of the normal RTS version (the link order will prefer the fast version, when available)
- Specify the section where the ROM content is located

See the example project, simple_app_dsplib_rts_rom, provided with this document.

Note that the user (application) memory range needs to be slightly reduced to reserve RAM memory for the .bss part of the lib.

```
/* internal RAM available for user applications */
IRAM_USER_LOW:            o = 10001000h   l = 00000B00h

/* to be reserved if using the ROM version of fastRTS */
IRAM_FASTRTS_RESERVED:    o = 10001B00h   l = 00000100h

/* internal RAM available for user applications */
IRAM_USER:                o = 10001C00h   l = 0003E400h
```

Again in the sections part, add the directives to locate the FastRTS parts of the libraries.

As before, if you look at the details of the map file, you can see how similarly the addresses for the FastRTS lib have changed and are now located in ROM. There is now an additional trampoline function pointing to the ROM version of the cos() function.

```
_cos  0002d120  .T$0005   10008e00  10005218   main.obj (.text)
```

Despite the need to reserve some user RAM to the RTS lib .bss section, the usage of the same user RAM section has further decreased to 0xf9df.

14 Moving the Example Code to DSP/BIOS Environment

In this section, DSP/BIOS is introduced into the project. This requires that the auto-generated linker command file be included in the user command file to get all the definitions for the memory and sections part configured into theDSP/BIOS .tcf configuration file.

Additionally, the standard C sections that were previously allocated by the user linker command file will be handled automatically by the BIOS configuration (for the sake of this example there are no specific reasons to customize them).

The project that is provided with this document is located simple_bios_app_dsplib_rts_rom.

Note: To properly rebuild the sources, it is required to have the BIOS_INSTALL_DIR system environment variable configured to point to the root directory of theDSP/BIOS installation. Make sure that the proper matching BIOS version is selected in the component manager of the Code Composer Studio (see help → about → component manager in Code Composer Studio).

At this point, ROM contents are not used for BIOS yet; potentially any BIOS version later than 5.20 can be used for the application. This is also true if the end user does not plan to use any ROM based parts of BIOS at a later stage. Nonetheless, it is recommended to use the latest version of BIOS 5.20 (as done in this example - 5.20.05 at the time of writing) since the ROM contents included in the DSP are being created and are compatible only with this release tree (5.20.0x).

Any successive release, like 5.31.x, can not be used if linking of ROM BIOS contents is desired.

The user part of the memory is going to be defined into the BIOS configuration, therefore, it will be removed by the linker command file.

To show the system being alive, a simple periodic function and a message log have been added to the BIOS-based system. The main file includes the PRD function implementation and the BIOS object definition has been included in the file.

Notice that the RAM usage has again increased to 0x1ba9b because of the BIOS routines and objects. However, now even BIOS is able to take advantage of the FastRTS lib for its internal purposes; in fact, there is a new entry in the trampolines function.

```
__divd    0002c4e0  .T$0000   1000ff60  1000ae0c   rts67plus.lib
```

15 Using the BIOS ROM Contents

This last project provided with this document is located in simple_bios_app_everything_rom.

In this section, there are further modifications to be made to the example. First, a memory section needs to be reserved for the BIOS internals. If DSP BIOS is being used, this memory cannot be used by the user application; reserve it for DSP BIOS.

Next, remove the definition of the BIOS ROM section from the user linker command file, since BIOS is specifying this automatically from its own configuration.

The last step is to set the value bios.GBL.LINKWITHROM = 1 in the BIOS configuration file. To do this, open the BIOS config file in the graphical tool and check the box *link with ROM* in the global settings.

This is a simple and efficient method to patch the BIOS contents and a provides powerful workaround in case bugs are discovered in the BIOS libraries. Every version of BIOS has internal knowledge of which parts of the ROM contents need to be patched, and which can be safely used. You do not have to worry about how to exclude parts of the library that might need to be patched.

As expected, the memory occupation has decreased to 0x18758. In the *far call trampolines* section of the map file, it is possible to observe a summary of the routines that are referenced in the ROM memory.

16 Summary

This application report shows how to make use of the ROM-based content of the C672x generation of devices by gradually reconfiguring the project for a generic application.

As a final note on the example project, it is not recommended to use printf() on the final system. The main reason being that the related string of formatting and processing is done on the target (DSP) side before the data is sent to the host, for displaying the output in Code Composer Studio.

For a more lightweight and less computation intensive way to achieve the same result, it might be advisable to use LOG_printf when possible, if DSP/BIOS is used. The required string processing is done on the host, therefore, the process consumes only a minimal set of DSP resources.

17 References

- C672x data manual (C672x product folder)
- *C9230C100 TMS320C672x Floating-Point Digital Signal Processor ROM* (SPRS277)
- *TMS320C67x DSP Library Programmer's Reference Guide* (SPRU657)
- *Using the TMS320C672x Bootloader* (SPRAA69)
- *Download: TMS320C67x Fast RTS Library* (SPRC060)
- *Download: TMS320C67x DSP Library* (SPRC121)
- *TMS320C6000 Optimizing Compiler v 6.0 Beta User's Guide* (SPRU187)
- *TMS320C6000 Assembly Language Tools v 6.0 Beta User's Guide* (SPRU186)
- *TMS320C6000 DSP/BIOS 5.32 Application Programming Interface (API) Reference Guide* (SPRU403)
- *TMS320 DSP/BIOS User's Guide* (SPRU423)
- DSP/BIOS 5.30 Textual Configuration (Tconf) User's Guide (SPRU007)

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