HOW TO GET STARTED WITH THE TEXAS INSTRUMENTS LF2407 EZDSP USING C

Mike Marcel
University of Alabama

August 28, 2003

Contents

List of Figures

0.1 INTRODUCTION

We purchased a couple of TI LF2407 eZdsp boards to implement a number of motor control algorithms to be included in the research we are doing here at the University of Alabama. I have personally used the Motorola family of Microcontrollers, Atmel Atmega family of microcontrollers, Microchip PIC family of microcontrollers as well as supported work using the EZDSP. The main reason why I am writing this paper is to document the number of difficulties that I had trying to run simple code and get started. There are literally thousands of papers regarding this family of DSP's and trying to find the information you are looking for showed to be quite overwhelming. So, I am writing this paper initially to document the steps I took to implement simple code, and later, I will add to this document as I exercise the peripherals. My overall goal is for someone to read this paper and be able to get some very simple code working on the DSP in a small amount of time.

0.2 SETUP

The setup for the EZDSP was very simple. It came in a box from Spectrum Digital Incorporated, and it came with a couple of test algorithms to test the board out of the box. These were helpful, but very complicated. My goal was to try to find some examples that showed me what needed to be set-up, what needed to be included in the .cmd file (and the format), how to set up the vector table (with the format) so that I can write a very simple algorithm (blink and LED) and get it to work. This proved to be a very complicated task, mainly because there is so much documentation that it is hard to find out how to get started.

First you need to install the Code Composer software that comes with the package. My package came with the TMSLF2407 Code Composer eZdsp DSK release 1.00. This includes the code composer IDE which includes an assembly code compiler, C compiler and a linker. Follow the instructions to install this version of code composer; it is very straightforward.

Once the software is installed, you need to look at the Code Composer Quick Start and Real-time Tutorial pamphlet that comes with the package. It is important to follow the directions on "setting up Code Composer" because there are a couple of steps that you have to do to set up the jtag emulator that is included with the hardware. If desired, you can go onto running the real-time tutorial, but I found it a waste of time. The only good I found was that it showed me that

my board worked. The tutorial is so very complicated (mostly because of the real-time features) that it is hard to understand where everything comes from and how it all goes together. The other problem I had was that once I ran the tutorial, somehow my board got stuck in "real time mode" and when I tried to run any other program, it told me it wasn't a real time program and I got an error. I tried numerous ways to turn off the real-time feature but I couldn't. I actually had to un-install and reinstall code composer to do this.

Once Code Composer is installed, and the emulator is configured, you now need to connect the board to the parallel port of your computer. If you don't have a parallel port, you are out of luck because the board uses the parallel port for downloading, debugging and emulation as well as to take advantage of the real time features. When you first power up the board, two LED's should come on (one in each corner; DS1 and DS2). DS1 is the Power LED, and DS2 is the PORTC_pin4 LED that defaults to the high position when powered up. You need to have the board powered prior to loading code composer or else you will get an error. If you are just writing some code and you are just going to make sure it compiles, press Ignore, otherwise make sure your parallel port cable and power cable are attached. Upon successful loading of Code Composer, you will see a guy on a motorcycle which will disappear in about 5 seconds and it will leave you to the IDE window.

0.3 GENERAL USE: CODE COMPOSER

When you first go into code composer, you will see the following GUI shown in figure 0.1.

Although the long column on the left looks like something from windows....be aware, it is not. This is where you build your project. In this window, you can look at all of the files associated with this project you are building now. This is where you can look and see what support files are being used by your program (.lib, .cmd, .asm, .h, etc) and what source code files you are including in this project.

The other window that you see is the disassembly window that shows the assembly code of the startup routine used by the eZdsp. I usually just minimize it because it is not very helpful.

0.3.1 About the Support Files

Generally, to get a very simple program running on this eZdsp, you need a number of files that need to be configured properly in order for everything to work. Also, these files cannot be used when writing strictly assembly or strictly C. The files I am going to talk about and that I include pertain to using C only. Most of this information for C and assembly can be found in an applications note from Texas Instruments. Its number is SPRA755A (July 2002) and the title is *Getting Started in C and Assembly Code with the TMS320LF240X DSP.* You can find this by going to www.ti.com and in the search engine in the top right hand corner type: SPRA755A.

0.3.1.1 Program.mak

This .mak file is a make file that is actually created by Code Composer. When you create a new project (we'll talk about that later) it automatically creates this make file that includes information about the linking options, source files, compiler options, etc. As you create a new

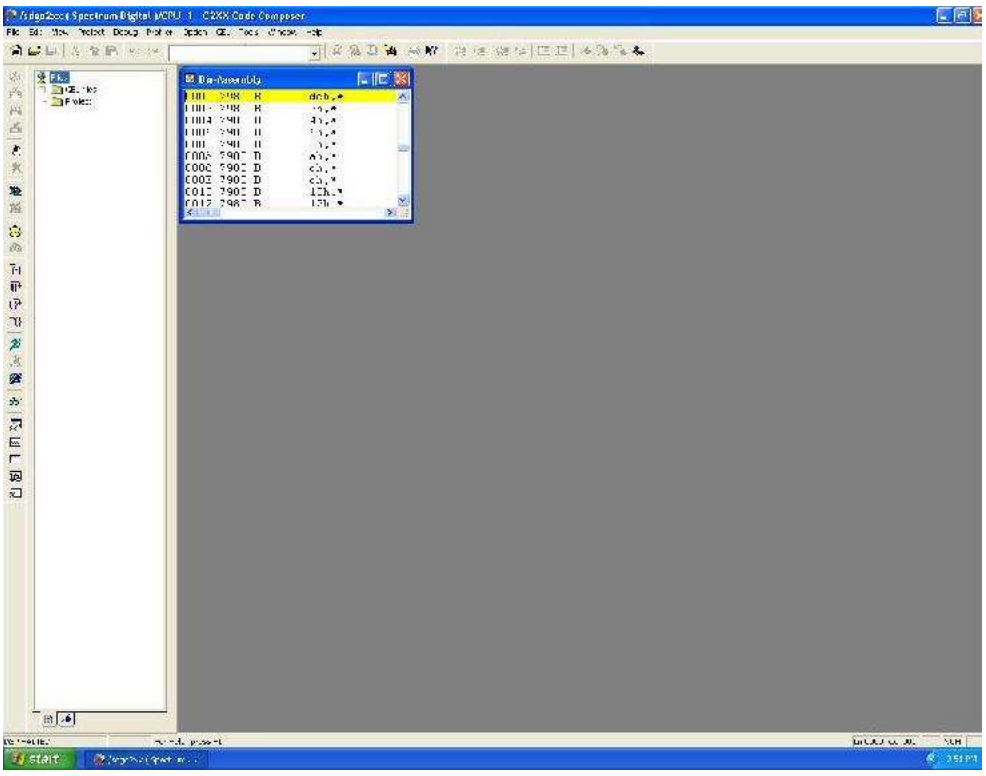

Figure 0.1: Code Composer IDE

project, this will all be transparent to you because Code Composer makes this automatically. What you do need to realize is that when you want to work on your project again, this is the file that you will find that is more like a "project" file.

0.3.1.2 Program.cmd

The .cmd file is the linker command file. This file basically tells the linker where to put the key elements of the program. A very basic linker command file will tell the linker where to put the vector table and how to allocate the memory. This file is very important because as the processor that you are using changes, so will this. I have included a very simple .cmd file that can be used with simple programs to get started with the eZdsp.

0.3.1.3 rts2xx.lib

This library file is not needed when writing assembly code, but must be included when writing C code. It is the runtime support library that includes the initialization parameters when a hardware reset occurs on the processor. This library generally comes with the code generation tools and must be included in all C programs.

0.3.1.4 cvectors.asm

This file (which must be written in assembly) lays out your interrupt vector table. It is common myth to believe that you don't need it, but it is imperative that you include this file when writing even the simplest program. The reason is that execution begins at the reset vector that is specified in your table (after a hardware reset). The reset branches to the rts2xx.lib function c_int(), which sets up the stack, initializes the values of all the interrupts and call main(). If you don't have a vector table in your program, it will get lost an nothing will happen. I have included a simple vector table that is used for the flashing LED program. This vector table does not call any vectors (except for the c_int()) but again must be included so that the processor does a proper reset.

0.3.1.5 f2407_c.h

This is an include file that is basically a long list of definitions for each register on the DSP. It is included to make life easier for the code writer so that they don't have to look up the address of each register when they are going to use it in their C program. For instance, if you are going to change the status of port A, you don't have to look up the address, you can just refer to it as PADATDIR. A copy of this support file is included at the end of this paper.

0.3.1.6 .obj files, .map files

After you rebuild your program, the compiler and linker will create the .obj and .map files so that code composer knows where to put the data. These files are created automatically and are transparent to the user.

0.3.1.7 .out file

When the rebuilding of your program is complete, and if there are no errors, code composer will create a .out file. This .out file is what will be downloaded to the DSP.

0.3.2 The Watch Window

The LF2407 makes debugging very easy. One of the greatest tools to use while debugging is the watch window. The watch window is a small window that can be put at the bottom of the Code Composer IDE. You can add whatever registers (I/O, A/D, Timers, etc) you would like and look at their value while your program is running. This can be helpful because you can see what you are putting to an I/O port without having to connect any hardware, or you can look at an A/D value to determine if it is correct.

0.3.2.1 Using the Watch Window

To use the watch window, choose "View, Watch Window". The watch window will appear in the lower right-hand corner. To add values to the watch window, you must right click in the window and choose, "Insert new Expression". A small window entitled Watch Add Expression will pop up and this is where you can type in the address of what you want to look at. It is important that if you type the address only, it will show you the decimal value of that address. To look at the contents of that address, define it as a pointer. For example, if you want to look at the contents of the A/D channel 0 result register, put *0x70A8 into the watch window. It will show you the value that is stored at that address. It is important to note that unless you are using a real-time mode program, the watch window cannot be automatically updated. To update the watch window manually, right click in the watch window and choose, "Refresh Watch Window". The new values will then be displayed.

0.4 PROGRAMMING EXAMPLE

0.4.1 The source file

You can create your source file either by clicking "file, new, source file" in the Code Composer environment, or by using your favorite text program (kwrite or notepad is what I use). Write the program as you would write any other C program (make sure you use the proper register names). A good source for the memory map, registers and peripherals for the 2407 is: SPRU357, "*TMS320LF/LC240x DSP Controllers Reference Guide"*. You can find this on the www.ti.com website.

0.4.2 Creating a project

The first thing you need to do is to create a directory on your hard-drive (it doesn't need to be in any special folder) that will include all of your project files. In this folder, make sure you put the cvectors.asm file, the f2407_c.h file, your C source code, your program.cmd linker command file, and the rts2xx.lib file. Once you have created this directory and moved all of your files

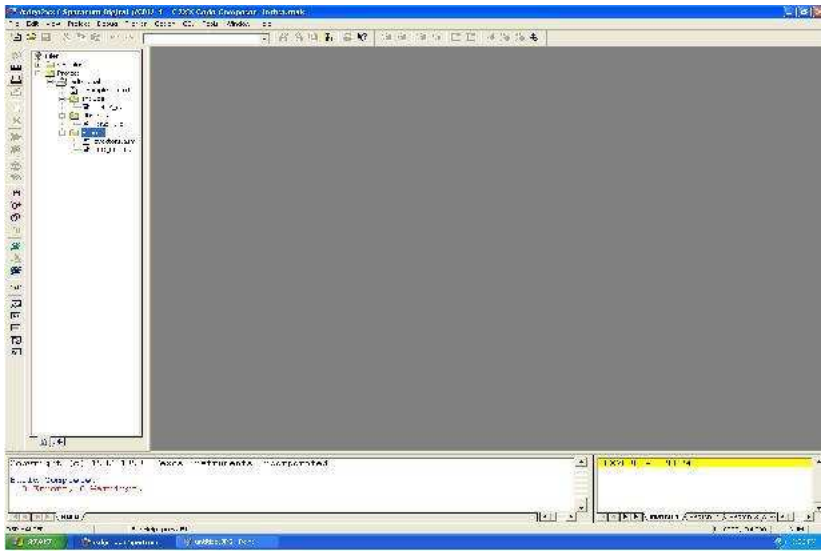

Figure 0.2: Complete Project File

into it, you can open Code Composer. Once you are in the Code Composer environment, click "project, new" and this will bring up a window for you to save a new project.mak file. Give your project a name and click save. You need to click on the + on the "project" folder and this will show a project.mak file in the project window on the left column of the screen. Click the + next to your project and it will open three more folders called include, libraries and source.

The next thing to do is to add the support files to your project. Right click on your .mak file on the project column on the left. Choose "add files". This will bring up a box that at the bottom will have a pull down menu that reads: "Files of type". First choose C Source files and add your source file to the project. Use the same method to add the vectors.asm, rts2xx.lib, and program.cmd to your project. Once you have added these to your project you should be able to expand the folders on the left side of the screen to show the project.cmd file and the rts2xx.lib in the library file, and the cvectors and source c code in the source directory.

Now, right click on your .mak file again and choose "scan all dependencies". This will grab the f2407_c.h file and include it in the include files in your project on the left. Once complete, your code composer window should look like the one in figure 0.2.

0.4.3 Rebuilding the project

Now that we have set up the project with all of the necessary files, it is time to compile and link the project. This can be done easily by either choosing the toolbar button on the left side of the screen (3rd from the top-the red arrows and box button) or by choosing "Project, Rebuild all". The compiler will run through it's checks and passes and will tell you if there are any warnings or errors. If you do have an error in your C code, you can scroll up the window on the bottom of the screen and the errors will be red. You can double click on that line and if the error is in your

C code, it will bring you to that line.

0.4.4 Resetting the DSP

Now that your project is built, it is time to download the machine code to the processor. The first (and very important) step is to choose " Debug, Reset DSP" to reset the DSP.

0.4.5 Downloading the File to the Processor

Once you have reset the DSP, it is time to download the machine code to the processor. You can do this by choosing "File, Load Program" and then choose the .out file that was created by the compiler/linker. You will see a dialog box in the top left corner show progress (small programs are usually fast to load) and when complete, the program will be in the processor.

0.4.6 Running the Code

To Run the code, you can either choose the menu icon on the left toolbar (the little green man running) or "Debug, Run". One good way to see if your program is running correctly is to toggle IOPC4 which is port C pin 4. This is connected to the LED that is in the corner of the DSP board (DS2). To stop running your code, you can choose the toolbar icon of the little man stopping (it is red), or "Debug, Halt". To make sure your code is running properly, you can look at the LED or if you are using a more advanced peripheral, you can look at the pin-outs of the eZdsp header. The reference for the header pin-out is: Texas Instruments/Spectrum Digital Technical Reference 505565-0001 Rev. C. This reference will show you all of the pin-outs of the board as well as the layout and other circuitry included with the board.

0.5 EXERCISING THE INPUTS AND OUTPUTS

The first thing I wanted to do was to exercise the inputs and outputs. The overall goal that I have with this DSP is to drive a Permanent Magnet Synchronous Motor with it, but first I need to understand and be able to use all of the Peripherals. In this section, I will include a simple program that will blink 4 LED's at a certain rate, and when you change an input pin, it will change the rate at which the LED's blink. This will show that I am using PORTC as both digital output and digital input.

0.5.1 The Multiplexor

The first thing you need to configure is the input/output multiplexor (MUX) control registers. A number of pins on the LF2407 DSP can be used for digital input/output as well as things like PWM generation, Capture/compare, Serial communications, etc. The LF2407 has three input/output MUX control registers. On page 5-5 through page 5-8 of the System and Peripherals Reference Guide(SPRU357), you can see what each of these registers control. Also, in the init() function that I have in the ledtst.c code, I have included the functions in a comment.

0.5.2 The Data Direction Register

The LF2407 is a 16 bit Digital Signal Processor. All of the registers it uses are 16 bits and because of this, the way you treat inputs and outputs is a bit different than I am used to. On Page 5-9 of the System and Peripherals Reference Guide (SPRU357) the data direction and control register for Port A is shown (PADATDIR). The high 8 bits of this register define the direction (0=input, 1=output). Once the direction is defined, the low 8 bits are the bits for the actual port. For an output: 0=low, 1 high; for an input: 0=low, 1=high. The important concept to understand is that both the direction and actual port pins are in the same register!

0.5.3 LED_FLSH3.c

This C program will test the bits of the PORTC register. I have connected 4 LED's to PORTC pin 4-7 and 4 input switches on PORTC pin 0-3 (active low).

The first thing that the main program will do is to call the initialization function. This initialization function will set up the two status registers properly, disable the watchdog timer, setup the external memory interface, and configure the three MUX registers. This initialization routine must be done prior to using the DSP. This routine came from the TI "Getting Started in C and Assembly Code with the TMS320LF240x DSP" (SPRA755A). I tried running the DSP with other initialization routines that were not as complete as this one, but it didn't initialize properly.

Once the initialization is complete, I set up my data direction register and make the LEDS low. I then poll the pushbuttons and decide the blink rate of the LED's. Once the blink rate is decided upon, the program moves into the routine where it will change the state of the LED, run the proper delay and then loop back and check the input port again. The program keeps doing this loop indefinitely.

0.5.4 Conclusion

While trying to exercise the inputs and outputs of this controller, I had few problems that were software related. One hardware problem that I found was that I originally pulled the port to ground through a 300K resistor and when the switch was pushed, the pin would be pulled high through a 5K resistor. I found that the 300K to ground was not enough to actually make the input pin a low state. The voltage was about 4 out of 5 volts. To fix this problem, with the switch open, I pulled the port high through a 300K resistor and when the switch was closed, it pulled the pin to ground. This worked very well.

0.6 USING THE ANALOG TO DIGITAL CONVERTER

0.6.1 Introduction

The LF2407 eZdsp has 16 multiplexed 10-bit A/D converters with Sample and Hold (S/H). The A/D converters can be run so that you can do a number of conversions in a quick, numbered sequence. Once the A/D converter is told to convert, it will run through the list of desired channels that need to be converted and put their result in the proper register. Although this sounds very easy and efficient, there is a detailed setup that must be followed even if you are

only sampling on one channel. The speed of each conversion is approximately 500ns (that's pretty fast) and you can sequence up to 16 channels. The other advantage of the sequencing is that you can sample either 16 different channels, or a number of the same channels in the order you specify (ie: if you desire to oversample).

0.6.2 A/D Registers

There are a number of registers that must be set up properly so that you can do a proper conversion. Below is a list of these registers and a bit of information on each.

0.6.2.1 ADCTRL1

This is the first A/D control register and it resides at address 0x70A0. This register defines the initial setup to include things like resetting, conversion clock settings, interrupt priority, calibration, etc. In the example code (adc_test.c) I have defined each of these bits in the A/D converter initialization.

0.6.2.2 ADCTRL2

This is the second A/D control register and it resides at address 0x70A1. This register contains the sequencing information, the start bits to start sequencing, and interrupt information. In the example code (adc_test.c) I have defined each of these bits in the A/D converter initialization.

0.6.2.3 MAX_CONV

This register defines the number of sequential conversions that are going to be one. There is a table on page 7-27 of SPRU357 ("TMS320LF/LC240x DSP controllers Reference Guide) that defines the values for this register. Simply put, the value that goes into this register can be found using the following equation:

$$MAX\_CONV = maximum\,number\,of\,conversions\,-\,1$$

This shows that if you are only doing one conversion, you must write 0x0000 to this register.

0.6.2.4 CHSELSEQn

This is a set of four registers that tell the DSP which channels to read and in what order. Each of the four registers is broken into four, four bit units where the A/D channel is to be read. For example, if it is desired to read Channels 2, 3, 2, 3, 6, 7, and 12, you can write those values to the proper registers as shown in the table sown below:

Address	bits 15-12	bits 11-8	bits 7-4	bits 3-0	Register Name
0x70A3	0011 (3)	0010 (2)	0011 (3)	0010 (2)	CHSELSEQ1
0x70A4	x	1100 (12)	0111 (7)	0110 (6)	CHSELSEQ2
0x70A5	x	x	x	x	CHSELSEQ3
0x70A6	x	x	x	x	CHSELSEQ4

Note on the table that the order starts with the LSB of CHSELSEQ1. All of the x's in the table are don't cares because every time a conversion is complete, the MAX_CONV register is automatically decremented and when it hits zero, it stops doing conversions. Hence, it will never reach the don't care addresses.

0.6.2.5 RESULTn

The result register is a 16 bit register that contains the A/D result of each conversion you have done. The 10 bit result is put in this register with the MSB of the 10 bit result as the 15th (High bit) of the register and the LSB of the 10 bit conversion is bit 6. Depending on how you are using the value, you can shift the bits accordingly to make an 8 bit result (to put out to a port), or if you are doing 16 bit math, the low 6 bytes of the result register are zeros. You can read the result registers directly and it is often helpful to put the result in the watch window. The addresses for RESULT0 to RESULT15 are 0x70A8 to 0x70B7 respectively.

0.6.3 SETTING UP AND USING THE A/D CONVERTER

I have written a .c program that uses the A/D converter to put the result of an A/D conversion from A/D channel 0 on the 7 least significant bits of PORTF. This program is called adc_test.c. All other supporting files are the same as the LED_TEST project that was featured earlier in this paper.

The first thing you need to do to set up the A/D converter is to enable the ADC module clock. This is done by setting bit 7 of the SCSR1 (System Control and Status Register) register. Otherwise, set up the DSP the same way you did in the LED_FLSH project. The next thing you need to do is to set up the ADCTRL1 register. As stated earlier, the bits (which can be found as a comment in my program or on Page 7-20 of SPRU357) are self-explanatory. Do the same for ADCTRL2.

Once complete, write the hexadecimal value of the number of different channels that you want to do A/D conversions to the MAX_CONV register. Remember the value that you write is the actual value minus one. For instance, in adc_test.c, I write a 0x0000 because I am doing one conversion. Next, in the CHSELSEQn registers, write which channels you wish to read and in what sequence (if there is any question how to do this, look at the register description in this paper).

Once all of the above setup tasks are complete, you are ready to use the A/D port. In adc_test.c, I am writing the 7 least significant bits to PORTF. PORTF is a port that for some reason only has 7 bits, so that's why I am only writing 7 bits. Remember, instead of using a logic analyzer or LED's to look at the value that is being written to the port, you can use the watch window.

The first step of my main program reads the value of the RESULT0 register. That is the A/D conversion of a 0-3 volt power supply that I put on the A/D channel 0 pins (pin 2 of the P1 connector on the LF2407 eZdsp). Once I have that 10 bit result (remember, in a 16 bit register), I shift it to the right 8 times. That gets rid of the 3 LSB's and puts the 8 MSB's in the lower 8 bits of a unsigned int I call adres. Because the port is a 7 bit port, I need to write all 1's to the high 8 bits (that will show that I am using the port as an output), bit 8 needs to be a one (it is

reserved for some reason) and the low 7 bits need to be my A/D result. I also have to realize that on my hardware, I have the LED's set up as active low, so I need to complement my result before I do anything with it.

So, at this point, I have read my value, shifted it so my bits are in the lower portion of the register, complemented it, now I have to actually write it to the port. I can do this easily by taking the value of the data direction and output register and just logical OR'ing it with my result. This will give my high 8 bits always a 1, my 7th bit always a one, and the proper lower bits to put on the LED's.

I then delay a bit and then I need to tell the A/D converter to do another sequence of conversions for me. I do this by rewriting the proper value to the ADCTRL2 register. Every time I want a conversion done I need to do this. In my program, I included it at the end of the endless "do" loop in the main program.

DSP Support Files

0.7 Program.mak

/********** Code Composer V1 Project Data ********************
The following section contains data generated by Code Composer
to store project information like build options, source filenames
and dependencies.
[command filename]
example_c.cmd 4
[Project Root]
C:\mike_files\TI_resources\LED_TST
[build options]
3
Linker = "-c -o ledtst.out -x "
Assembler = "-s -v2xx "
Compiler = "-g -v2xx -as -frC:\mike_files\TI_resources\LED_TST "
[source files]
cvectors.asm 1062018900 1
rts2xx.lib 0 1
LED_FLSH3.c 1062088771 1
f2407_c.h 1029179262 0
3698
[dependencies]
0 -802
0 -802
1 3:12 -5502
0 -802
[version]
2.0

```
*/
-c -o ledtst.out -x
"LED_FLSH3.obj"
"cvectors.obj"
"rts2xx.lib"
"example_c.cmd"
/******** End of Project Data Generated by Code Composer ********/
```

0.8 Program.cmd

```
/*****************************************************************************
* Filename: example_c.cmd *
* *
* Author: David M. Alter, Texas Instruments Inc. *
* *
* Last Modified: 03/14/01 *
* *
* Description: C code linker command file for LF2407 DSP. *
*****************************************************************************/
MEMORY
{
PAGE 0: /* Program Memory */
VECS: org=00000h, len=00040h /* internal FLASH */
FLASH: org=00044h, len=07FBCh /* internal FLASH */
EXTPROG: org=08800h, len=07800h /* external SRAM */
PAGE 1: /* Data Memory */
B2: org=00060h, len=00020h /* internal DARAM */
B0: org=00200h, len=00100h /* internal DARAM */
B1: org=00300h, len=00100h /* internal DARAM */
SARAM: org=00800h, len=00800h /* internal SARAM */
EXTDATA: org=08000h, len=08000h /* external SRAM */
}
SECTIONS
{
/* Sections generated by the C-compiler */
.text: > FLASH PAGE 0 /* initialized */
.cinit: > FLASH PAGE 0 /* initialized */
.const: > B1 PAGE 1 /* initialized */
.switch: > FLASH PAGE 0 /* initialized */
.bss: > B1 PAGE 1 /* uninitialized */
.stack: > SARAM PAGE 1 /* uninitialized */
.sysmem: > B1 PAGE 1 /* uninitialized */
/* Sections declared by the user */
```

```
vectors: > VECS PAGE 0 /* initialized */
}
```

0.9 cvectors.asm

```
*************************************************************************
* Filename: cvectors.asm *
* *
* Author: David M. Alter, Texas Instruments Inc. *
* *
* Last Modified: 03/14/01 *
* *
* Description: Interrupt vector table for ’240x DSP core *
* for use with C language programs. *
* *
*************************************************************************
*************************************************************************
* THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR *
* REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY, *
* INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS *
* FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR *
* COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE. *
* TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET *
* POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY *
* INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR *
* YOUR USE OF THE PROGRAM. *
* *
* IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL, *
* CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY *
* THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT *
* OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM. *
* EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF *
* REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS *
* OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF *
* USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI’S *
* AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF *
* YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS *
* (U.S.$500). *
* *
* Unless otherwise stated, the Program written and copyrighted *
* by Texas Instruments is distributed as "freeware". You may, *
* only under TI’s copyright in the Program, use and modify the *
```

```
* Program without any charge or restriction. You may *
* distribute to third parties, provided that you transfer a *
* copy of this license to the third party and the third party *
* agrees to these terms by its first use of the Program. You *
* must reproduce the copyright notice and any other legend of *
* ownership on each copy or partial copy, of the Program. *
* *
* You acknowledge and agree that the Program contains *
* copyrighted material, trade secrets and other TI proprietary *
* information and is protected by copyright laws, *
* international copyright treaties, and trade secret laws, as *
* well as other intellectual property laws. To protect TI's *
* rights in the Program, you agree not to decompile, reverse *
* engineer, disassemble or otherwise translate any object code *
* versions of the Program to a human-readable form. You agree *
* that in no event will you alter, remove or destroy any *
* copyright notice included in the Program. TI reserves all *
* rights not specifically granted under this license. Except *
* as specifically provided herein, nothing in this agreement *
* shall be construed as conferring by implication, estoppel, *
* or otherwise, upon you, any license or other right under any *
* TI patents, copyrights or trade secrets. *
* *
* You may not use the Program in non-TI devices. *
*****************************************************************
.ref _c_int0
.sect "vectors"
rset: B _c_int0 ;00h reset
int1: B int1 ;02h INT1
int2: B int2 ;04h INT2
int3: B int3 ;06h INT3
int4: B int4 ;08h INT4
int5: B int5 ;0Ah INT5
int6: B int6 ;0Ch INT6
int7: B int7 ;0Eh reserved
int8: B int8 ;10h INT8 (software)
int9: B int9 ;12h INT9 (software)
int10: B int10 ;14h INT10 (software)
int11: B int11 ;16h INT11 (software)
int12: B int12 ;18h INT12 (software)
int13: B int13 ;1Ah INT13 (software)
int14: B int14 ;1Ch INT14 (software)
int15: B int15 ;1Eh INT15 (software)
int16: B int16 ;20h INT16 (software)
```

```
int17: B int17 ;22h TRAP
int18: B int18 ;24h NMI
int19: B int19 ;26h reserved
int20: B int20 ;28h INT20 (software)
int21: B int21 ;2Ah INT21 (software)
int22: B int22 ;2Ch INT22 (software)
int23: B int23 ;2Eh INT23 (software)
int24: B int24 ;30h INT24 (software)
int25: B int25 ;32h INT25 (software)
int26: B int26 ;34h INT26 (software)
int27: B int27 ;36h INT27 (software)
int28: B int28 ;38h INT28 (software)
int29: B int29 ;3Ah INT29 (software)
int30: B int30 ;3Ch INT30 (software)
int31: B int31 ;3Eh INT31 (software)
```

0.10 f2407_c.h

```
/*************************************************************************
  * Filename: f2407_c.h *
  * *
  * Author: David M. Alter, Texas Instruments Inc. *
  * *
  * Description: LF2407 DSP register definitions for C-code. *
  * *
  * History: 03/14/01 - original (D. Alter) *
  * 08/12/02 - fixed addresses of CAP2FBOT, CAP3FBOT, *
  * CAP5FBOT, and CAP6FBOT. (D. Alter) *
  *************************************************************************/
  /* Core registers */
  #define IMR (volatile unsigned int *)0x0004 /* Interrupt mask reg */
  #define GREG (volatile unsigned int *)0x0005 /* Global memory allocation reg */
  #define IFR (volatile unsigned int *)0x0006 /* Interrupt flag reg */
  /* System configuration and interrupt registers */
  #define PIRQR0 (volatile unsigned int *)0x7010 /* Peripheral interrupt request reg 0 */
  #define PIRQR1 (volatile unsigned int *)0x7011 /* Peripheral interrupt request reg 1 */
  #define PIRQR2 (volatile unsigned int *)0x7012 /* Peripheral interrupt request reg 2 */
  #define PIACKR0 (volatile unsigned int *)0x7014 /* Peripheral interrupt acknowledge reg 0
*/
  #define PIACKR1 (volatile unsigned int *)0x7015 /* Peripheral interrupt acknowledge reg 1
*/
  #define PIACKR2 (volatile unsigned int *)0x7016 /* Peripheral interrupt acknowledge reg 2
*/
```

```
#define SCSR1 (volatile unsigned int *)0x7018 /* System control & status reg 1 */
#define SCSR2 (volatile unsigned int *)0x7019 /* System control & status reg 2 */
#define DINR (volatile unsigned int *)0x701C /* Device identification reg */
#define PIVR (volatile unsigned int *)0x701E /* Peripheral interrupt vector reg */
/* Watchdog timer (WD) registers */
#define WDCNTR (volatile unsigned int *)0x7023 /* WD counter reg */
#define WDKEY (volatile unsigned int *)0x7025 /* WD reset key reg */
#define WDCR (volatile unsigned int *)0x7029 /* WD timer control reg */
/* Serial Peripheral Interface (SPI) registers */
#define SPICCR (volatile unsigned int *)0x7040 /* SPI configuration control reg */
#define SPICTL (volatile unsigned int *)0x7041 /* SPI operation control reg */
#define SPISTS (volatile unsigned int *)0x7042 /* SPI status reg */
#define SPIBRR (volatile unsigned int *)0x7044 /* SPI baud rate reg */
#define SPIRXEMU (volatile unsigned int *)0x7046 /* SPI emulation buffer reg */
#define SPIRXBUF (volatile unsigned int *)0x7047 /* SPI serial receive buffer reg */
#define SPITXBUF (volatile unsigned int *)0x7048 /* SPI serial transmit buffer reg */
#define SPIDAT (volatile unsigned int *)0x7049 /* SPI serial data reg */
#define SPIPRI (volatile unsigned int *)0x704F /* SPI priority control reg */
/* SCI registers */
#define SCICCR (volatile unsigned int *)0x7050 /* SCI communication control reg */
#define SCICTL1 (volatile unsigned int *)0x7051 /* SCI control reg 1 */
#define SCIHBAUD (volatile unsigned int *)0x7052 /* SCI baud-select reg, high bits */
#define SCILBAUD (volatile unsigned int *)0x7053 /* SCI baud-select reg, low bits */
#define SCICTL2 (volatile unsigned int *)0x7054 /* SCI control reg 2 */
#define SCIRXST (volatile unsigned int *)0x7055 /* SCI receiver status reg */
#define SCIRXEMU (volatile unsigned int *)0x7056 /* SCI emulation data buffer reg */
#define SCIRXBUF (volatile unsigned int *)0x7057 /* SCI receiver data buffer reg */
#define SCITXBUF (volatile unsigned int *)0x7059 /* SCI transmit data buffer reg */
#define SCIPRI (volatile unsigned int *)0x705F /* SCI priority control reg */
/* External interrupt configuration registers */
#define XINT1CR (volatile unsigned int *)0x7070 /* Ext interrupt 1 config reg */
#define XINT2CR (volatile unsigned int *)0x7071 /* Ext interrupt 2 config reg */
/* Digital I/O registers */
#define MCRA (volatile unsigned int *)0x7090 /* I/O mux control reg A */
#define MCRB (volatile unsigned int *)0x7092 /* I/O mux control reg B */
#define MCRC (volatile unsigned int *)0x7094 /* I/O mux control reg C */
#define PADATDIR (volatile unsigned int *)0x7098 /* I/O port A data & dir reg */
#define PBDATDIR (volatile unsigned int *)0x709A /* I/O port B data & dir reg */
#define PCDATDIR (volatile unsigned int *)0x709C /* I/O port C data & dir reg */
#define PDDATDIR (volatile unsigned int *)0x709E /* I/O port D data & dir reg */
#define PEDATDIR (volatile unsigned int *)0x7095 /* I/O port E data & dir reg */
#define PFDATDIR (volatile unsigned int *)0x7096 /* I/O port F data & dir reg */
/* Analog-to-Digital Converter (ADC) registers */
#define ADCTRL1 (volatile unsigned int *)0x70A0 /* ADC control reg 1 */
```

```
#define ADCTRL2 (volatile unsigned int *)0x70A1 /* ADC control reg 2 */
#define MAX_CONV (volatile unsigned int *)0x70A2 /* Maximum conversion channels reg
*/
#define CHSELSEQ1 (volatile unsigned int *)0x70A3 /* Channel select sequencing control
reg 1 */
#define CHSELSEQ2 (volatile unsigned int *)0x70A4 /* Channel select sequencing control
reg 2 */
#define CHSELSEQ3 (volatile unsigned int *)0x70A5 /* Channel select sequencing control
reg 3 */
#define CHSELSEQ4 (volatile unsigned int *)0x70A6 /* Channel select sequencing control
reg 4 */
#define AUTO_SEQ_SR (volatile unsigned int *)0x70A7 /* Autosequence status reg */
#define RESULT0 (volatile unsigned int *)0x70A8 /* Conversion result buffer reg 0 */
#define RESULT1 (volatile unsigned int *)0x70A9 /* Conversion result buffer reg 1 */
#define RESULT2 (volatile unsigned int *)0x70AA /* Conversion result buffer reg 2 */
#define RESULT3 (volatile unsigned int *)0x70AB /* Conversion result buffer reg 3 */
#define RESULT4 (volatile unsigned int *)0x70AC /* Conversion result buffer reg 4 */
#define RESULT5 (volatile unsigned int *)0x70AD /* Conversion result buffer reg 5 */
#define RESULT6 (volatile unsigned int *)0x70AE /* Conversion result buffer reg 6 */
#define RESULT7 (volatile unsigned int *)0x70AF /* Conversion result buffer reg 7 */
#define RESULT8 (volatile unsigned int *)0x70B0 /* Conversion result buffer reg 8 */
#define RESULT9 (volatile unsigned int *)0x70B1 /* Conversion result buffer reg 9 */
#define RESULT10 (volatile unsigned int *)0x70B2 /* Conversion result buffer reg 10 */
#define RESULT11 (volatile unsigned int *)0x70B3 /* Conversion result buffer reg 11 */
#define RESULT12 (volatile unsigned int *)0x70B4 /* Conversion result buffer reg 12 */
#define RESULT13 (volatile unsigned int *)0x70B5 /* Conversion result buffer reg 13 */
#define RESULT14 (volatile unsigned int *)0x70B6 /* Conversion result buffer reg 14 */
#define RESULT15 (volatile unsigned int *)0x70B7 /* Conversion result buffer reg 15 */
#define CALIBRATION (volatile unsigned int *)0x70B8 /* Calibration result reg */
/* Controller Area Network (CAN) registers */
#define MDER (volatile unsigned int *)0x7100 /* CAN mailbox direction/enable reg */
#define TCR (volatile unsigned int *)0x7101 /* CAN transmission control reg */
#define RCR (volatile unsigned int *)0x7102 /* CAN receive control reg */
#define MCR (volatile unsigned int *)0x7103 /* CAN master control reg */
#define BCR2 (volatile unsigned int *)0x7104 /* CAN bit config reg 2 */
#define BCR1 (volatile unsigned int *)0x7105 /* CAN bit config reg 1 */
#define ESR (volatile unsigned int *)0x7106 /* CAN error status reg */
#define GSR (volatile unsigned int *)0x7107 /* CAN global status reg */
#define CEC (volatile unsigned int *)0x7108 /* CAN trans and rcv err counters */
#define CAN_IFR (volatile unsigned int *)0x7109 /* CAN interrupt flag reg */
#define CAN_IMR (volatile unsigned int *)0x710A /* CAN interrupt mask reg */
#define LAM0_H (volatile unsigned int *)0x710B /* CAN local acceptance mask MBX0/1
*/
#define LAM0_L (volatile unsigned int *)0x710C /* CAN local acceptance mask MBX0/1 */
```

```
  #define LAM1_H (volatile unsigned int *)0x710D /* CAN local acceptance mask MBX2/3
*/
  #define LAM1_L (volatile unsigned int *)0x710E /* CAN local acceptance mask MBX2/3 */
  #define MSGID0L (volatile unsigned int *)0x7200 /* CAN message ID for mailbox 0 (lower
16 bits) */
  #define MSGID0H (volatile unsigned int *)0x7201 /* CAN message ID for mailbox 0 (upper
16 bits) */
  #define MSGCTRL0 (volatile unsigned int *)0x7202 /* CAN RTR and DLC for mailbox 0 */
  #define MBX0A (volatile unsigned int *)0x7204 /* CAN 2 of 8 bytes of mailbox 0 */
  #define MBX0B (volatile unsigned int *)0x7205 /* CAN 2 of 8 bytes of mailbox 0 */
  #define MBX0C (volatile unsigned int *)0x7206 /* CAN 2 of 8 bytes of mailbox 0 */
  #define MBX0D (volatile unsigned int *)0x7207 /* CAN 2 of 8 bytes of mailbox 0 */
  #define MSGID1L (volatile unsigned int *)0x7208 /* CAN message ID for mailbox 1 (lower
16 bits) */
  #define MSGID1H (volatile unsigned int *)0x7209 /* CAN message ID for mailbox 1 (upper
16 bits) */
  #define MSGCTRL1 (volatile unsigned int *)0x720A /* CAN RTR and DLC for mailbox 1
*/
  #define MBX1A (volatile unsigned int *)0x720C /* CAN 2 of 8 bytes of mailbox 1 */
  #define MBX1B (volatile unsigned int *)0x720D /* CAN 2 of 8 bytes of mailbox 1 */
  #define MBX1C (volatile unsigned int *)0x720E /* CAN 2 of 8 bytes of mailbox 1 */
  #define MBX1D (volatile unsigned int *)0x720F /* CAN 2 of 8 bytes of mailbox 1 */
  #define MSGID2L (volatile unsigned int *)0x7210 /* CAN message ID for mailbox 2 (lower
16 bits) */
  #define MSGID2H (volatile unsigned int *)0x7211 /* CAN message ID for mailbox 2 (upper
16 bits) */
  #define MSGCTRL2 (volatile unsigned int *)0x7212 /* CAN RTR and DLC for mailbox 2 */
  #define MBX2A (volatile unsigned int *)0x7214 /* CAN 2 of 8 bytes of mailbox 2 */
  #define MBX2B (volatile unsigned int *)0x7215 /* CAN 2 of 8 bytes of mailbox 2 */
  #define MBX2C (volatile unsigned int *)0x7216 /* CAN 2 of 8 bytes of mailbox 2 */
  #define MBX2D (volatile unsigned int *)0x7217 /* CAN 2 of 8 bytes of mailbox 2 */
  #define MSGID3L (volatile unsigned int *)0x7218 /* CAN message ID for mailbox 3 (lower
16 bits) */
  #define MSGID3H (volatile unsigned int *)0x7219 /* CAN message ID for mailbox 3 (upper
16 bits) */
  #define MSGCTRL3 (volatile unsigned int *)0x721A /* CAN RTR and DLC for mailbox 3
*/
  #define MBX3A (volatile unsigned int *)0x721C /* CAN 2 of 8 bytes of mailbox 3 */
  #define MBX3B (volatile unsigned int *)0x721D /* CAN 2 of 8 bytes of mailbox 3 */
  #define MBX3C (volatile unsigned int *)0x721E /* CAN 2 of 8 bytes of mailbox 3 */
  #define MBX3D (volatile unsigned int *)0x721F /* CAN 2 of 8 bytes of mailbox 3 */
  #define MSGID4L (volatile unsigned int *)0x7220 /* CAN message ID for mailbox 4 (lower
16 bits) */
```

```
#define MSGID4H (volatile unsigned int *)0x7221 /* CAN message ID for mailbox 4 (upper
16 bits) */
#define MSGCTRL4 (volatile unsigned int *)0x7222 /* CAN RTR and DLC for mailbox 4 */
#define MBX4A (volatile unsigned int *)0x7224 /* CAN 2 of 8 bytes of mailbox 4 */
#define MBX4B (volatile unsigned int *)0x7225 /* CAN 2 of 8 bytes of mailbox 4 */
#define MBX4C (volatile unsigned int *)0x7226 /* CAN 2 of 8 bytes of mailbox 4 */
#define MBX4D (volatile unsigned int *)0x7227 /* CAN 2 of 8 bytes of mailbox 4 */
#define MSGID5L (volatile unsigned int *)0x7228 /* CAN message ID for mailbox 5 (lower
16 bits) */
#define MSGID5H (volatile unsigned int *)0x7229 /* CAN message ID for mailbox 5 (upper
16 bits) */
#define MSGCTRL5 (volatile unsigned int *)0x722A /* CAN RTR and DLC for mailbox 5
*/
#define MBX5A (volatile unsigned int *)0x722C /* CAN 2 of 8 bytes of mailbox 5 */
#define MBX5B (volatile unsigned int *)0x722D /* CAN 2 of 8 bytes of mailbox 5 */
#define MBX5C (volatile unsigned int *)0x722E /* CAN 2 of 8 bytes of mailbox 5 */
#define MBX5D (volatile unsigned int *)0x722F /* CAN 2 of 8 bytes of mailbox 5 */
/* Event Manager A (EVA) registers */
#define GPTCONA (volatile unsigned int *)0x7400 /* GP timer control reg A */
#define T1CNT (volatile unsigned int *)0x7401 /* GP timer 1 counter reg */
#define T1CMPR (volatile unsigned int *)0x7402 /* GP timer 1 compare reg */
#define T1PR (volatile unsigned int *)0x7403 /* GP timer 1 period reg */
#define T1CON (volatile unsigned int *)0x7404 /* GP timer 1 control reg */
#define T2CNT (volatile unsigned int *)0x7405 /* GP timer 2 counter reg */
#define T2CMPR (volatile unsigned int *)0x7406 /* GP timer 2 compare reg */
#define T2PR (volatile unsigned int *)0x7407 /* GP timer 2 period reg */
#define T2CON (volatile unsigned int *)0x7408 /* GP timer 2 control reg */
#define COMCONA (volatile unsigned int *)0x7411 /* Compare control reg A */
#define ACTRA (volatile unsigned int *)0x7413 /* Compare action control reg A */
#define DBTCONA (volatile unsigned int *)0x7415 /* Dead-band timer control reg A */
#define CMPR1 (volatile unsigned int *)0x7417 /* compare reg 1 */
#define CMPR2 (volatile unsigned int *)0x7418 /* compare reg 2 */
#define CMPR3 (volatile unsigned int *)0x7419 /* compare reg 3 */
#define CAPCONA (volatile unsigned int *)0x7420 /* Capture control reg A */
#define CAPFIFOA (volatile unsigned int *)0x7422 /* Capture FIFO status reg A */
#define CAP1FIFO (volatile unsigned int *)0x7423 /* Capture Channel 1 FIFO top */
#define CAP2FIFO (volatile unsigned int *)0x7424 /* Capture Channel 2 FIFO top */
#define CAP3FIFO (volatile unsigned int *)0x7425 /* Capture Channel 3 FIFO top */
#define CAP1FBOT (volatile unsigned int *)0x7427 /* Bottom reg of capture FIFO stack 1 */
#define CAP2FBOT (volatile unsigned int *)0x7428 /* Bottom reg of capture FIFO stack 2 */
#define CAP3FBOT (volatile unsigned int *)0x7429 /* Bottom reg of capture FIFO stack 3 */
#define EVAIMRA (volatile unsigned int *)0x742C /* EVA interrupt mask reg A */
#define EVAIMRB (volatile unsigned int *)0x742D /* EVA interrupt mask reg B */
#define EVAIMRC (volatile unsigned int *)0x742E /* EVA interrupt mask reg C */
```

```
#define EVAIFRA (volatile unsigned int *)0x742F /* EVA interrupt flag reg A */
#define EVAIFRB (volatile unsigned int *)0x7430 /* EVA interrupt flag reg B */
#define EVAIFRC (volatile unsigned int *)0x7431 /* EVA interrupt flag reg C */
/* Event Manager B (EVB) registers */
#define GPTCONB (volatile unsigned int *)0x7500 /* GP timer control reg B */
#define T3CNT (volatile unsigned int *)0x7501 /* GP timer 3 counter reg */
#define T3CMPR (volatile unsigned int *)0x7502 /* GP timer 3 compare reg */
#define T3PR (volatile unsigned int *)0x7503 /* GP timer 3 period reg */
#define T3CON (volatile unsigned int *)0x7504 /* GP timer 3 control reg */
#define T4CNT (volatile unsigned int *)0x7505 /* GP timer 4 counter reg */
#define T4CMPR (volatile unsigned int *)0x7506 /* GP timer 4 compare reg */
#define T4PR (volatile unsigned int *)0x7507 /* GP timer 4 period reg */
#define T4CON (volatile unsigned int *)0x7508 /* GP timer 4 control reg */
#define COMCONB (volatile unsigned int *)0x7511 /* Compare control register B */
#define ACTRB (volatile unsigned int *)0x7513 /* Compare action control register B */
#define DBTCONB (volatile unsigned int *)0x7515 /* Dead-band timer control reg B */
#define CMPR4 (volatile unsigned int *)0x7517 /* Compare reg 4 */
#define CMPR5 (volatile unsigned int *)0x7518 /* Compare reg 5 */
#define CMPR6 (volatile unsigned int *)0x7519 /* Compare reg 6 */
#define CAPCONB (volatile unsigned int *)0x7520 /* Capture control reg B */
#define CAPFIFOB (volatile unsigned int *)0x7522 /* Capture FIFO status reg B */
#define CAP4FIFO (volatile unsigned int *)0x7523 /* Capture channel 4 FIFO top */
#define CAP5FIFO (volatile unsigned int *)0x7524 /* Capture channel 5 FIFO top */
#define CAP6FIFO (volatile unsigned int *)0x7525 /* Capture channel 6 FIFO top */
#define CAP4FBOT (volatile unsigned int *)0x7527 /* Bottom reg of capture FIFO stack 4 */
#define CAP5FBOT (volatile unsigned int *)0x7528 /* Bottom reg of capture FIFO stack 5 */
#define CAP6FBOT (volatile unsigned int *)0x7529 /* Bottom reg of capture FIFO stack 6 */
#define EVBIMRA (volatile unsigned int *)0x752C /* EVB interrupt mask reg A */
#define EVBIMRB (volatile unsigned int *)0x752D /* EVB interrupt mask reg B */
#define EVBIMRC (volatile unsigned int *)0x752E /* EVB interrupt mask reg C */
#define EVBIFRA (volatile unsigned int *)0x752F /* EVB interrupt flag reg A */
#define EVBIFRB (volatile unsigned int *)0x7530 /* EVB interrupt flag reg B */
#define EVBIFRC (volatile unsigned int *)0x7531 /* EVB interrupt flag reg C */
/* I/O space mapped registers */
#define FCMR portFF0F /* Flash control mode register */
ioport unsigned int portFF0F; /* C2xx compiler specific keyword */
#define WSGR portFFFF /* Wait-state generator reg */
ioport unsigned int portFFFF; /* C2xx compiler specific keyword */
```

0.11 LED_FLSH3.c

```
//////////////////////////////////////////////////////////
// This is an LED Test Program. It merely takes the high//
```

```
// 4 bits of PORTA go low. //
// //
////////////////////////////////////////////////////////
// Rev 2. Now I will make the high 4 bits of PORTC //
// flash in succession //
////////////////////////////////////////////////////////
// Rev 3. Now I will exercise the use of the pushbuttons
// that I have put on PORTC (4 low bits). As I
// push each button, the LED's will blink slower
//
// For the TMS320LF2407 EZDSP //
////////////////////////////////////////////////////////
#include "f2407_c.h"
void init(void)
{
/*** Configure the System Control and Status registers ***/
*SCSR1 = 0x00FD;
/*
bit 15 0: reserved
bit 14 0: CLKOUT = CPUCLK
bit 13-12 00: IDLE1 selected for low-power mode
bit 11-9 000: PLL x4 mode
bit 8 0: reserved
bit 7 1: 1 = enable ADC module clock
bit 6 1: 1 = enable SCI module clock
bit 5 1: 1 = enable SPI module clock
bit 4 1: 1 = enable CAN module clock
bit 3 1: 1 = enable EVB module clock
bit 2 1: 1 = enable EVA module clock
bit 1 0: reserved
bit 0 1: clear the ILLADR bit
*/
*SCSR2 = (*SCSR2 | 0x000B) & 0x000F;
/*
bit 15-6 0's: reserved
bit 5 0: do NOT clear the WD OVERRIDE bit
bit 4 0: XMIF_HI-Z, 0=normal mode, 1=Hi-Z'd
bit 3 1: disable the boot ROM, enable the FLASH
bit 2 no change MP/MC* bit reflects state of MP/MC* pin
bit 1-0 11: 11 = SARAM mapped to prog and data
*/
/*** Disable the watchdog timer ***/
*WDCR = 0x00E8;
/*
```

```
bits 15-8 0's: reserved
bit 7 1: clear WD flag
bit 6 1: disable the dog
bit 5-3 101: must be written as 101
bit 2-0 000: WDCLK divider = 1
*/
/*** Setup external memory interface for LF2407 EVM ***/
WSGR = 0x0040;
/*
bit 15-11 0's: reserved
bit 10-9 00: bus visibility off
bit 8-6 001: 1 wait-state for I/O space
bit 5-3 000: 0 wait-state for data space
bit 2-0 000: 0 wait state for program space
*/
/*** Setup shared I/O pins ***/
*MCRA = 0x0000; /* group A pins */
/*
bit 15 0: 0=IOPB7, 1=TCLKINA
bit 14 0: 0=IOPB6, 1=TDIRA
bit 13 0: 0=IOPB5, 1=T2PWM/T2CMP
bit 12 0: 0=IOPB4, 1=T1PWM/T1CMP
bit 11 0: 0=IOPB3, 1=PWM6
bit 10 0: 0=IOPB2, 1=PWM5
bit 9 0: 0=IOPB1, 1=PWM4
bit 8 0: 0=IOPB0, 1=PWM3
bit 7 0: 0=IOPA7, 1=PWM2
bit 6 1: 0=IOPA6, 1=PWM1
bit 5 0: 0=IOPA5, 1=CAP3
bit 4 0: 0=IOPA4, 1=CAP2/QEP2
bit 3 0: 0=IOPA3, 1=CAP1/QEP1
bit 2 0: 0=IOPA2, 1=XINT1
bit 1 0: 0=IOPA1, 1=SCIRXD
bit 0 0: 0=IOPA0, 1=SCITXD
*/
*MCRB = 0xFE00; /* group B pins */
/*
bit 15 1: 0=reserved, 1=TMS2 (always write as 1)
bit 14 1: 0=reserved, 1=TMS (always write as 1)
bit 13 1: 0=reserved, 1=TD0 (always write as 1)
bit 12 1: 0=reserved, 1=TDI (always write as 1)
bit 11 1: 0=reserved, 1=TCK (always write as 1)
bit 10 1: 0=reserved, 1=EMU1 (always write as 1)
bit 9 1: 0=reserved, 1=EMU0 (always write as 1)
```

```
bit 8 0: 0=IOPD0, 1=XINT2/ADCSOC
bit 7 0: 0=IOPC7, 1=CANRX
bit 6 0: 0=IOPC6, 1=CANTX
bit 5 0: 0=IOPC5, 1=SPISTE
bit 4 0: 0=IOPC4, 1=SPICLK
bit 3 0: 0=IOPC3, 1=SPISOMI
bit 2 0: 0=IOPC2, 1=SPISIMO
bit 1 0: 0=IOPC1, 1=BIO*
bit 0 0: 0=IOPC0, 1=W/R*
*/
*MCRC = 0x0000; /* group C pins */
/*
bit 15 0: reserved
bit 14 0: 0=IOPF6, 1=IOPF6
bit 13 0: 0=IOPF5, 1=TCLKINB
bit 12 0: 0=IOPF4, 1=TDIRB
bit 11 0: 0=IOPF3, 1=T4PWM/T4CMP
bit 10 0: 0=IOPF2, 1=T3PWM/T3CMP
bit 9 0: 0=IOPF1, 1=CAP6
bit 8 0: 0=IOPF0, 1=CAP5/QEP4
bit 7 0: 0=IOPE7, 1=CAP4/QEP3
bit 6 0: 0=IOPE6, 1=PWM12
bit 5 0: 0=IOPE5, 1=PWM11
bit 4 0: 0=IOPE4, 1=PWM10
bit 3 0: 0=IOPE3, 1=PWM9
bit 2 0: 0=IOPE2, 1=PWM8
bit 1 0: 0=IOPE1, 1=PWM7
bit 0 0: 0=IOPE0, 1=CLKOUT
*/
}
//////////////////////////////////////////////////////////////////
void delay() // just a simple 10 ms delay routine
{
unsigned int i; // for our counter
 for(i = 50000; i > 0; i–)
 {
 asm("NOP"); // do a NOP a bunch of times
 }
}
void main()
{
unsigned int j; // our second loop counter
unsigned int loops;
unsigned int inputc; // defines what we input
```

```
init(); // call the initialization function
*PCDATDIR=0xF00F; // make all low
loops=10; // define our original input loops
do
{
 inputc = *PCDATDIR; // read input port
 if (inputc==0xF00E) // if PB1 is pushed
 {
 loops=5;
 }
 if (inputc==0xF00D) // if PB1 is pushed
 {
 loops=10;
 }
 if (inputc==0xF00B) // if PB1 is pushed
 {
 loops=15;
 }
 if (inputc==0xF007) // if PB1 is pushed
 {
 loops=20;
 }
  if(*PCDATDIR==0xF00F) // if LED's are low
  {
  *PCDATDIR=0xF0FF; // turn on LED's
  }
  else
  {
  *PCDATDIR=0xF00F; // make LED's low
  }
   for(j = loops; j > 0; j–)
   {
   delay();
   }
}while(1); // keep doing this loop!
}
```

0.12 adc_test.c

```
//////////////////////////////////////////////////////////
   // This program is a test program to test the A/D //
   // converter on the 2407 DSP. I will input a //
   // 0-3 volt voltage on A/D channel 0 and put the //
```

```
// value on the PORTD LED's //
// //
// By Mike Marcel //
// University of Alabama //
// 9/2/03 //
//////////////////////////////////////////////////////
// For the TMS320LF2407 EzDSP //
//////////////////////////////////////////////////////
#include "f2407_c.h"
void init(void)
{
/*** Configure the System Control and Status registers ***/
*SCSR1 = 0x0081;
/*
bit 15 0: reserved
bit 14 0: CLKOUT = CPUCLK
bit 13-12 00: IDLE1 selected for low-power mode
bit 11-9 000: PLL x4 mode
bit 8 0: reserved
bit 7 1: 1 = enable ADC module clock
bit 6 0: 1 = enable SCI module clock
bit 5 0: 1 = enable SPI module clock
bit 4 0: 1 = enable CAN module clock
bit 3 0: 1 = enable EVB module clock
bit 2 0: 1 = enable EVA module clock
bit 1 0: reserved
bit 0 1: clear the ILLADR bit
*/
*SCSR2 = (*SCSR2 | 0x000B) & 0x000F;
/*
bit 15-6 0's: reserved
bit 5 0: do NOT clear the WD OVERRIDE bit
bit 4 0: XMIF_HI-Z, 0=normal mode, 1=Hi-Z'd
bit 3 1: disable the boot ROM, enable the FLASH
bit 2 no change MP/MC* bit reflects state of MP/MC* pin
bit 1-0 11: 11 = SARAM mapped to prog and data
*/
/*** Disable the watchdog timer ***/
*WDCR = 0x00E8;
/*
bits 15-8 0's: reserved
bit 7 1: clear WD flag
bit 6 1: disable the dog
bit 5-3 101: must be written as 101
```

```
bit 2-0 000: WDCLK divider = 1
*/
/*** Setup external memory interface for LF2407 EVM ***/
WSGR = 0x0040;
/*
bit 15-11 0's: reserved
bit 10-9 00: bus visibility off
bit 8-6 001: 1 wait-state for I/O space
bit 5-3 000: 0 wait-state for data space
bit 2-0 000: 0 wait state for program space
*/
/*** Setup shared I/O pins ***/
*MCRA = 0x0000; /* group A pins */
/*
bit 15 0: 0=IOPB7, 1=TCLKINA
bit 14 0: 0=IOPB6, 1=TDIRA
bit 13 0: 0=IOPB5, 1=T2PWM/T2CMP
bit 12 0: 0=IOPB4, 1=T1PWM/T1CMP
bit 11 0: 0=IOPB3, 1=PWM6
bit 10 0: 0=IOPB2, 1=PWM5
bit 9 0: 0=IOPB1, 1=PWM4
bit 8 0: 0=IOPB0, 1=PWM3
bit 7 0: 0=IOPA7, 1=PWM2
bit 6 1: 0=IOPA6, 1=PWM1
bit 5 0: 0=IOPA5, 1=CAP3
bit 4 0: 0=IOPA4, 1=CAP2/QEP2
bit 3 0: 0=IOPA3, 1=CAP1/QEP1
bit 2 0: 0=IOPA2, 1=XINT1
bit 1 0: 0=IOPA1, 1=SCIRXD
bit 0 0: 0=IOPA0, 1=SCITXD
*/
*MCRB = 0xFE00; /* group B pins */
/*
bit 15 1: 0=reserved, 1=TMS2 (always write as 1)
bit 14 1: 0=reserved, 1=TMS (always write as 1)
bit 13 1: 0=reserved, 1=TD0 (always write as 1)
bit 12 1: 0=reserved, 1=TDI (always write as 1)
bit 11 1: 0=reserved, 1=TCK (always write as 1)
bit 10 1: 0=reserved, 1=EMU1 (always write as 1)
bit 9 1: 0=reserved, 1=EMU0 (always write as 1)
bit 8 0: 0=IOPD0, 1=XINT2/ADCSOC
bit 7 0: 0=IOPC7, 1=CANRX
bit 6 0: 0=IOPC6, 1=CANTX
bit 5 0: 0=IOPC5, 1=SPISTE
```

```
bit 4 0: 0=IOPC4, 1=SPICLK
bit 3 0: 0=IOPC3, 1=SPISOMI
bit 2 0: 0=IOPC2, 1=SPISIMO
bit 1 0: 0=IOPC1, 1=BIO*
bit 0 0: 0=IOPC0, 1=W/R*
*/
*MCRC = 0x0000; /* group C pins */
/*
bit 15 0: reserved
bit 14 0: 0=IOPF6, 1=IOPF6
bit 13 0: 0=IOPF5, 1=TCLKINB
bit 12 0: 0=IOPF4, 1=TDIRB
bit 11 0: 0=IOPF3, 1=T4PWM/T4CMP
bit 10 0: 0=IOPF2, 1=T3PWM/T3CMP
bit 9 0: 0=IOPF1, 1=CAP6
bit 8 0: 0=IOPF0, 1=CAP5/QEP4
bit 7 0: 0=IOPE7, 1=CAP4/QEP3
bit 6 0: 0=IOPE6, 1=PWM12
bit 5 0: 0=IOPE5, 1=PWM11
bit 4 0: 0=IOPE4, 1=PWM10
bit 3 0: 0=IOPE3, 1=PWM9
bit 2 0: 0=IOPE2, 1=PWM8
bit 1 0: 0=IOPE1, 1=PWM7
bit 0 0: 0=IOPE0, 1=CLKOUT
*/
*PFDATDIR=0xFF80; // make all outputs low at first
}
ad_init()
{
////// Setup the first A/D control register
*ADCTRL1= 0xD040; /* A/D ctl reg */
/*
bit 15 1: reserved
bit 14 0: 0=No effect, 1=Reset entire ADC : Reset Pin
bit 13 0: 00=immediate stop; 10=complete conv. before stp
bit 12 1: x1=Free run
bit 11 0: Acq time window; see book for codes
bit 10 0:
bit 9 0:
bit 8 0:
bit 7 0: 0=Fclk=CLK/1; 1=Fclk=CLK/2
bit 6 1: 0=start/stop mode; 1=contin. conv. mode
bit 5 0: 0=high int. prior; 1=low priority
bit 4 0: 0=duel-seq mode; 1=cascaded mode
```

```
bit 3 0: 0=calibratin mode dis; 1=calibr. mode en
bit 2 0: 0=full ref V to ADC; 1=ref midpoint applied to ADC
bit 1 0: 0=Vreflo precharge; 1=Vrefhi precharge
bit 0 0: 0=self test mode disabled; 1=self test mode enabled
*/
///// Setup the second A/D control register
*ADCTRL2= 0x2000; /* group C pins */
/*
bit 15 0: 0=do nothing; 1=seq started by event manager
bit 14 0: 0 unless using calibration mode
bit 13 1: start conv. trigger..see book
bit 12 0: 0=sequencer idle; 1=conversion in progress
bit 11 0: interrupt mode enable...see book
bit 10 0:
bit 9 0: 0=no interrupt; 1=interrupt occured (int. flag)
bit 8 0: 0=SEQ1 not started by EV trig; 1=SEQ1 started by EM
bit 7 0: 0=no action; 1=enables ADC autoconversion from ADCSOC
bit 6 0: 0=no action; 1=resets SEQ2 to pretrig state
bit 5 0: start conv. trigger for seq2...see book
bit 4 0: 0= sequencer idle; 1=conversion seq in progress (SEQ2)
bit 3 0: interrupt mode enable control...see book
bit 2 0:
bit 1 0: 0=no interrupt event; 1=interrupt has occured
bit 0 0: 0=SEQ2 cannot be started by EVB trig
1=SEQ2 started by Event manager B trigger
*/
*MAX_CONV=0x0000; // max_conv=max_conv+1...we're doing one channel
*CHSELSEQ1=0x0000; // we are using A/D channel 0 only
}
void gt_AD_smp()
{
// Reset the second control register to get a new sample
*ADCTRL2= 0x2000; /* group C pins */
/*
bit 15 0: 0=do nothing; 1=seq started by event manager
bit 14 0: 0 unless using calibration mode
bit 13 1: start conv. trigger..see book
bit 12 0: 0=sequencer idle; 1=conversion in progress
bit 11 0: interrupt mode enable...see book
bit 10 0:
bit 9 0: 0=no interrupt; 1=interrupt occured (int. flag)
bit 8 0: 0=SEQ1 not started by EV trig; 1=SEQ1 started by EM
bit 7 0: 0=no action; 1=enables ADC autoconversion from ADCSOC
bit 6 0: 0=no action; 1=resets SEQ2 to pretrig state
```

```
bit 5 0: start conv. trigger for seq2...see book
bit 4 0: 0= sequencer idle; 1=conversion seq in progress (SEQ2)
bit 3 0: interrupt mode enable control...see book
bit 2 0:
bit 1 0: 0=no interrupt event; 1=interrupt has occured
bit 0 0: 0=SEQ2 cannot be started by EVB trig
1=SEQ2 started by Event manager B trigger
*/
}
void delay() // just a simple 10 ms delay routine
{
unsigned int i; // for our counter
for(i = 50000; i > 0; i–)
{
asm("NOP"); // do a NOP a bunch of times
}
}
void main()
{
unsigned int adres; // our A/D result
unsigned int output; // this is what we will output to LED's
init(); // call the initialization routine
ad_init(); // call the A/D converter initialization routine
do
{
adres=*RESULT0; // read A/D port 0
// Now we have to make the low 7 bits the result
adres=adres> >8; // right shift 8 times which will give me the
// low 7 bits of an 8 bit conversion
adres=~adres; // complement the bits to put on LED's
output=0xFF80; //this is the initial setup for the Dat Dir Reg
output=output | adres; // or the output with the A/D result
*PFDATDIR=output; // output this on the LED bus (and complement)
delay(); // delay a bit. this delay will set the sample rate
gt_AD_smp(); // get a new A/D converter sample
}while(1);
}
```