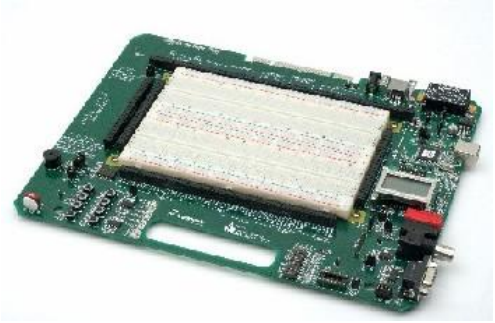
Assignment 1 Board Configuration, Compiling, Debugging, and Executable Files – CSE 325, Fall 2010

Assignment Objectives

1. To learn how to configure the M5211DEMO board for downloading and running code.
2. To learn how to use Freescale CodeWarrior IDE to create a project, enter code, compile, download, and debug.
3. To learn about the internal details of the object code and ELF file format.

Part A: Configuring the boards; Creating a project; Compiling and Debugging

In many embedded systems, the application software is running on bare hardware with no operating system. After a hardware reset, the microcontroller will be initialized by system initialization code, and then the application program will begin executing (in a C program, the system initialization will occur before main() is called). Therefore, the application software and the system initialization code should be linked together as a complete program which can be stored in a non-volatile memory (ROM, EPROM, EEPROM, or flash). In this exercise we are going to learn how to configure the M5211DEMO board for downloading compiled C code from the host computer system and then to use the Freescale CodeWarrior IDE on the host computer system to run and debug the code in the embedded microcontroller.



Project Board

M5211DEMO Board

You must follow these steps exactly as I have described them.

1. When you go into the lab, the M5211DEMO boards should already be plugged into the J5 port of the project board.
2. Make sure the On/Off switch on the demo board is turned off. Disconnect the power cord (on both demo board and project board).
3. In this assignment, you will only need the demo board which can receive power from USB, on-board voltage regulator, and project board. You should select power input from USB.
4. Plug the USB cable and the serial cable coming from the host computer into the appropriate ports on the demo board. Note that LED D301 beside the USB connector will light up when the USB cable is plugged in.

5. Verify that these jumpers on the demo board are configured correctly: BDM_EN is ON; XTAL_EN is OFF; POT_EN is ON; VX_EN is ON; LED_EN is ON; COM_EN all are ON; PWR_SEL is VB.
6. Turn on the On/Off switch on the demo board. The following led's should light up: RESET, LED1-LED4, VDD LED, and D301
7. Log into the host computer and click Start > All Programs > Freescale CodeWarrior > CodeWarrior for ColdFire V6.3 > IDE.
8. When the IDE loads, click **File|New** on the main menu.
9. Select the **Project** tab. Select ColdFire Stationery. In the Project Name field enter Assgn01. Make sure the Location field points to your "My Documents" folder or some other folder to which you have write access. Click OK.
10. The **New Project** dialog box will appear. Right-click on the + beside CF_M5211DEMO to expand it. Click on C and then OK.
11. In the project browser window on the left click on the drop-down combo box. Select M5211DEMO COnsole Debug.
12. Right-click on + beside Source to expand it.
13. Double-click on main.c to open the editor window.
14. Delete everything from main.c and enter the code shown below:

```
#include "common.h"
#include "delay_loop.h"

/*********************************************************************/
/*
 * Display the 4 LEDs connected to TIN[3:0]
 *
 */

int delay1=100;
int delay2=200;

void main ()
{
   /* Enable signals as GPIO */
        MCF_GPIO_PTCPAR = 0
                | MCF_GPIO_PTCPAR_TIN3_GPIO
                | MCF_GPIO_PTCPAR_TIN2_GPIO
                | MCF_GPIO_PTCPAR_TIN1_GPIO
                | MCF_GPIO_PTCPAR_TIN0_GPIO;

   /* Enable signals as digital outputs */
         MCF_GPIO_DDRTC = 0
                | MCF_GPIO_DDRTC_DDRTC3
                | MCF_GPIO_DDRTC_DDRTC2
                | MCF_GPIO_DDRTC_DDRTC1
                | MCF_GPIO_DDRTC_DDRTC0;

        while(1)
```

```
        {
                int i;

                for (i = 0; i < 4; i++)
                {
                /* Set output values */
                        MCF_GPIO_PORTTC = (uint8)(0x01<<i);

                delay_loop(delay1, delay2);

                }
        }
}
```

15. Create delay.c program file in “source” folder and add it to the project

```
/* simple delay loops
*/
void delay_loop1(int b)
{
        int j=2, k=5;
        for (j=0; j<b; j++)
        {
                k=k+1;
                k=k-1;
        }
}
void delay_loop(int a, int b)
{
        int i;

        for (i=0; i<a; i++)
        {
                delay_loop1(b);
        }
}
```

16. Add delay.h to “include” folder

```
/*
 * File:    delay_loop.h
*/

#ifndef _DELAY_LOOP_H_
#define _DELAY_LOOP_H_

void delay_loop(int, int);

#endif
```

17. Click **Edit|Preferences** on the main menu bar. You can change code formatting, editor, fonts, tabs, and debugger settings
18. Click **Edit|M5211 Demo Console Debug Settings** on the main menu bar. In the dialog box, click each of the **+ signs** to expand the items in the left pane

19. Under **Target** select **ColdFire Target**. You should see **Project Type** set to **Application**, and filename set to **M5211DEMO Console Debug.elf**.
20. Under **Code Generation|Cooldfire Processor,** select **standard** in **parameter passing** field**.**
21. Click **Project|Make** on the main menu (or hit F7) to make the project. Continue to correct any syntax errors until you get a clean compile.
22. Click **Project|Debug** on the main menu to download the code and load the debugger. If you watch the demo board when you do this you will see the red RESET led flash twice—once at the beginning of the code download process and once at the end.
23. Either click **Debug|Step Over** or hit F10 several times until the line MCF_GPIO_PORTTC = **(uint8)(0x01<<i)**; is highlighted. At this point look at the demo board and observe that the LED1-LED4 led's are all lit up. Now hit F10 to execute the line of code you are stopped on. You should see LED2-LED4 go out and LED1 should be on.
24. Continue hitting F10 to step through the code line-by-line. Observe as you do so that the LEDs are lit up in the sequence LED1, LED2, LED3, LED4, LED1, LED2, ...
25. Now click **Project|Run** on the main menu to run the code nonstop. You should see the led's flash quickly.
26. Click **Debug|Break** on the main menu to stop execution.
27. Click **Debug|Kill** on the main menu to stop the debugger. Congratulations, you have just successfully run your first program on the M5211DEMO board.
28. Look at the delay_loop() function. This sort of "delay loop" is a crude way to introduce a delay into a program. All it does is force the processor to do some busy work for awhile; the problem with this is that the processor could be doing something more productive instead. A better way to delay is to start a timer with a count value that is decremented once per clock tick. When the count value goes to 0 we "wake up" and continue execution. We will learn how to implement this sort of delay later in the course when we discuss timers.
29. Now, play around with the delay count values, i.e. delay1 and delay2, to see how changing it changes the rate at which the led's are activated. What happens if the call to delay_loop() is removed?
30. When you are in debug mode, click **View|Registers** on the main menu to examine the 5211 registers. If necessary, click the + beside PEMICRO_USB to expand it. Click the + beside Supervisor Registers to expand it.

Part B: C Programming; More Debugging

There are four led's on the demo board labeled LED1-LED4. For this exercise, modify your main.c() program so the first sixteen binary integers 0000, 0001, 0010, ..., 1110, 1111 will be displayed on the led's with a small delay between each integer.

Part C: Execuable file

1. ELF is an acronym for Executable Linking and Formatting and is the file format that is used in the GNU Linux/UNIX world for object files and executables (it's used in more than just GNU Linux/UNIX but it originated in the UNIX world). It is also the format that

the CodeWarrior IDE uses to store the code that is downloaded and debugged in the M5211DEMO board. It wouldn't hurt for you to learn a little bit about the ELF format. Below are some good references.

1. http://www.linuxjournal.com/article/1060
2. http://www.skyfree.org/linux/references/ELF_Format.pdf
3. http://docsrv.sco.com/SDK_cprog/CTOC-ObjFls.html

2. Figure out where CodeWarrior is storing your project files on your disk. Go the bin folder. You should see the .elf file there. Since the file is with binary data, to view the contents of the file we have to use a hex editor/viewer program, such as Universal Viewer. You can download Universal Viewer and install it, or use any other hex editor/viewer program you want. Open the ELF file in the hex viewer, and display the contents in hex format.

Questions to be answered in your report:

1. (5 points) What are the definitions of MCF_GPIO_DDRTC and MCF_GPIO_PORTTC in main.c? Where are they defined? How do they get included in main.c.
2. (10 points) In Part A, we show you how to debug a program by stepping through each statement. However, you can set breakpoint(s) to stop program execution and then examine data and registers.
   a. Describe how you can set a breakpoint on the statement "MCF_GPIO_PORTTC = (uint8)(0x01<<i)".
   b. Where the variables delay1 and delay2 are located in memory? Give the addresses of the two variables. Are these two variables 32-bit or 16-bit integers? Are they stored in big or little endian format in memory?
   c. You can use single step to stop at the next line delay_loop(delay1, delay2);. What are the values of the following registers: stack point, SR, IPSBAR; FLASHBAR; RAMBAR?
   d. You can step into the call to delay_loop function. What is the value of stack point immediately after stepping into delay_loop? In the stack, find the locations of the parameters (delay1 and delay2) that you pass to the function and the return address of the procedure call?
   e. Set a breakpoint at the for loop statement of delay_loop1. Describe the additional content pushed into the stack since starting the main program.
3. (10 points) What is the MIPS measure of the MCF5211 microprocessor when we run the delay_loop() routine on the demo board? You can obtain this measure by (1) try to change the loop count of the delay function such that the delay is 1 second approximately, and (2) disassembly M5211DEMO Console Debug.elf and find the number of instructions executed in each loop iteration of the delay function.
4. (5 points) You can disassemble main.c and delay.c. While the executable code in M5211DEMO Console Debug.elf is made from main.c and delay.c, it contains additional code. Give an explanation about these extra code.
5. (20 points) When you look into the binary file M5211DEMO Console Debug.elf**,**
   a. What are the first four bytes of the file in hex?

b. What is the structure of the ELF header? How many bytes does an ELF header have?
c. For each of the following header fields describe: how many bytes the header field takes, and a short one sentence description of the header field. For each field, also describe the value stored in that header field in your .elf file and what that value indicates: e_ident; e_type; e_machine; e_version; e_entry; e_phoff; e_shoff; e_ehsize; e_phentsize; e_phnum; e_shentsize; e_shnum.
d. From examining the ELF file, is the MCF5211 ColdFire processor a little-endian or a big-endian architecture? How did you determine this?
e. How many program segments and sections in the elf file? Give their starting positions (the offset from the beginning of the file at which the first byte of the segment and sections resides) and their lengths.