CS/EE 5780/6780 Handout #2
Spring 2007
Myers

LAB #2: CodeWarrior

Lab writeup is due to your TA at the beginning of your next scheduled lab. Don't put this off to the last minute! There is pre-lab work to complete before the start of the next lab. **NO LATE LAB REPORTS WILL BE ACCEPTED.**

1 Objectives

- Learn how to use the CodeWarrior simulator for the MC9S12C32 derivative.
- Gain experience using the CodeWarrior debugger.
- Complete LCD output code that will be useful in future labs.

2 Tasks

1. Simulate the Lab2counter.asm program using the CodeWarrior simulator.
2. Simulate the Lab2counter.c program using the CodeWarrior simulator.
3. Examine Lab2counter.c program using the CodeWarrior debugger.
4. Add additional functionality to the Lab2LCD.c

3 Tools

1. Your prototype board and CSM12C32 module.
2. CodeWarrior 4.5 Special Edition. Loaded on lab PCs and available for free download.
3. Your Project Board and USB cable.
4. Lab2counter.asm, Lab2counter.c and Lab2LCD code (download source from course website)
5. MCU datasheet, MCU Reference Manual, CSM12C32 Reference manual. (download from course website)

Students should also bring a diskette or USB memory drive for saving work completed during the lab. Internet is available on lab machines for e-mail.

4 Procedures

1. Simulating with CodeWarrior
   (a) Visit the class webpage and download Lab2counter.asm
   (b) Create a new assembly CodeWarrior project and replace the contents of main.c with the contents of Lab2Counter.asm. Make sure when you create the project you have "Full Chip Simulation" selected. The method to create an assembly project is the same for C, except on Page 3 of the setup wizard, select assembly and deselect the other options. On page 4, make sure to select Absolute Assembly.

(c) Before you perform a make, select "Full Chip Simulation" from the combo box under the project window.

(d) Make the file and start the debugger (Project→Debug)

(e) The debugger window will come up, do not press the green start button.

(f) We now need to create some inputs and outputs for our simulation. To do this, click Component→Open. Select Io_LED from the list and click OK.

(g) To setup for pushbuttons, go back to the component window and also select Push_buttons and click OK.

(h) To connect your virtual LEDs and Pushbuttons to your program, you'll have to set them up with the correct memory addresses of the ports you're using.

(i) Open the mc9s12c32.inc, typically located in:

```
C:\Program Files\Freescale\CW for HC12 V4.5\lib\hc12c\include
```

directory. This file contains all of the memory address mappings of the I/O devices.

(j) Do a search on this file for DDRE, PORTE, DDRAD, PTAD. Write down the hexadecimal address locations for each of these. For PTAD, make sure you have found PTAD and not PTAD0.

(k) The other option to finding the correct memory addresses of the ports is the Motorola Data Sheet. Look under section 1.2.2.

(l) Go back to your simulator, right-click your IO_Led window and click setup. Under Port, put the address of PTAD, under DDR, put the address of DDRAD. Input just the hexadecimal numbers, you do not need the 0x prefix.

(m) Next right-click the Push_button window and click setup. In the input box, input the address location of PortE.

(n) Now, press the green run button. You should see the CPU cycles in the Register window incrementing.

(o) At this moment, your IO_Led should have four green LEDs lit up, indicating output is enabled. If you do not see the green LEDs, you may have put in an incorrect address for IO_Led. If the LEDs are simply flashing between green and red, then the Push_button is not setup correctly.

(p) Press button 0 on the Push_button window and observe how this increments the counter that's outputted to the IO_Led window.

(q) For your convenience, click File→save configuration for your project. This will allow you to re-run the simulation and not have to setup your input and outputs again.

(r) After you have finished simulating and verifying the assembly counter, create a new C project and simulate Lab2Count.c in the same manner.

2. Debugging with Codewarrior

(a) If you have not already done so, close the True-Time simulator & Real-Time debugger window. Do not close the CodeWarrior project.

(b) Within the CodeWarrior project, select "E Multilink CyclonePro" from the combo box.

(c) Make the program. Connect your CSM12C32 module to the project board, and connect the Project Board to the PC using the USB cable. Now run the debugger to download your program to the module.

(d) Click the green run button and press either SW1, or PB2 and make sure the LEDs on the board are counting in the same fashion as in the simulation.

(e) Click the red halt button to stop the program. Click the second button to the right of the green run button and observe how you can step through the program.

(f) Press the PB2 button while stepping through the program and observe the behavior of the if statements as you step through.

(g) What you may notice is that the logic levels of the push buttons of the project board and simulator are the opposite. The push buttons on the Project Board are low when pressed, and high otherwise. The push buttons in the simulator are high when pressed, and low otherwise. This difference needs to be taken into consideration when performing simulations of your code.

3. Add Functionality to Lab2LCD

(a) Create a new project and make sure to have at least "E Multilink CyclonePro" selected.

(b) Download Lab2LCD.c from the course website and replace your main.c with it.

(c) Make the program and download it to your module over USB.

(d) Run the program and you should see a message on the LCD screen. It should say "ECE3720" followed by "Lab2" If it does not, double check the jumpers on your project board and make sure MOSI, MISO, and the SCK jumpers are in place.

(e) Once you've verified that the program is working properly, add additional functionality to the program by adding an LCDNum function that outputs the correct numerical character to the screen for a number between 0 and 9.

(f) Next, add a function called LCDDecimal that takes any byte and outputs its decimal value onto the LCD screen.

(g) Finally, add a function called LCDHex that takes any byte and outputs its hexadecimal value onto the LCD screen.

5 Prelab

1. You should complete step 1 above before coming to your lab section and be prepared to show this simulation to your TA.

2. You should write the code for step 3 before coming to lab and be prepared to show this to your TA at the beginning of your lab section. This code does not need to be 100 percent debugged, but getting it close to working will allow your lab section to go more smoothly.

6 Writeup

1. Include a printout of your final commented code.

2. Describe any problems you encountered.