LAB #2: CodeWarrior

The Lab write-up 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 LCD output code that will be useful in future labs.
- Gain experience writing inline Assembly using CodeWarrior.

2 Tasks

1. Simulate the Lab2counter.asm program using the CodeWarrior simulator.
2. Examine Lab2LCD.c program using the CodeWarrior and project board.
3. Learn how to implement a function using inline Assembly.

3 Tools

1. Your prototype board and CSM12C32 module.
2. CodeWarrior for HCS12 Special Edition. Loaded on lab PCs and available for free download.
3. Your Project Board and USB cable.
4. Lab2counter.asm, Lab2LCD.c and Lab2Bits.c code (download source 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.asm 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 Visualization tool from the list and click OK.
(g) In the VisualizationTool window, right-click on the gray background and select "Add New Instrument," and add an LED.
(h) Connect the simulated LED to your program by double-clicking on it to bring up its "properties" menu where you should select "kind of port" to be memory, "port to display" to be the address of PTAD, and "bitnumber to display" to be 0.
(i) To find the address of PTAD, open the file mc9s12c32.inc, typically located here: c:/Program Files/Freescale/CodeWarrior for HCS12 V4.7/lib/hc12c/include/ This file contains all of the memory address mappings of the I/O devices. Do a search for PTAD. Another option for finding the correct memory addresses of the ports is to look under section 1.2.2 in the MC9S12C Family Reference Manual.
(j) Create two more LEDs and connect these to bits 1 and 2 of PTAD
(k) Now, press the green run button. You should see the CPU cycles in the Register window incrementing.
(l) Stop the simulator and start single-stepping it; the three LEDs should be operating as a 3-bit counter.
(m) Check-off this working simulation with your TA.

2. Interfacing with project board LCD using Lab2LCD

(a) Create a new project and make sure to have at least "P&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 "ECE5780" 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, replace the call to LCDString() with a call to the LCDNum function (void LCDNum(int val)) that outputs the correct numerical character to the screen for a number between 0 and 9. Try outputting a 7 to the LCD screen.
(f) Next, repeat the same process and test the two functions called LCDDecimal (void LCDDecimal(unsigned char val)) that takes any byte and outputs its decimal value onto the LCD screen and LCDInt (void LCDInt(unsigned int val)) that takes a 16 bit integer and outputs the value of the integer. Try outputting a 21 from both functions onto the LCD screen.
(g) Finally, replace the call to LCDInt with a call to LCDHex (void LCDHex(unsigned char val)) that takes any byte and outputs its hexadecimal value onto the LCD screen.
(h) Use these functions to output the hex values 0xA9 and 0x3F to the screen followed by the integer 25134. Check-off this functionality with your TA.

3. Add Functionality to Lab2Bits

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

(b) Download Lab2Bits.c from the course website and replace your main.c with it.

(c) Unfortunately, the dipswitches are not internally connected to your module so you will need to physically connect them. They are the dipswitches marked USER I/O SW1 & SW2 and you can find their connections next to the breadboard on the lower side. Using wire in your kit, connect your module to the protoboard in the following fashion:

SW2 4 ->PT7
SW2 3 ->PT6
SW2 2 ->PT5
SW2 1 ->PT4
SW1 4 ->PT3
SW1 3 ->PT2
SW1 2 ->PT1
SW1 1 ->PT0

Interpret SW2 4 as "Bit 7" and SW1 1 as "Bit 0".

You should go over your kits documentation because you will need to find out exactly what pin numbers that Port T, "PT", is located on to connect them properly.

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

(d) Run the program and you should see a number on the LCD screen. It should display a count of the number of dipswitches that are "on" on your project board. If it does not, double check the jumpers on your project board and make sure MOSI, MISO, and the SCK jumpers are in place. Also remove jumpers PB and LED on UFEA if they are set in place.

(e) Once you've verified that the program is working properly, add additional functionality to the program by instead displaying the ASCII character represented by the dipswitches to the LCD screen. Have bits be interpreted as ones when a dipswitch is "on" and zeros otherwise. NOTE: You will need to make sure that that you have a value between 0x21 – 0x7E on your input or else you will not be able to see it displayed on the LCD screen.

(f) Finally, add a bit more functionality by implementing a simple bit twiddling routine in assembly. You will generate a 7 bit ASCII character given only valid inputs of 0x21-0x7E. You will need to generate the following

i) Bit 7 of the output will be Bit 7 of the input (ie: no change)
ii) Bit 6 of the output will be Bits 6 and 5 of the input OR'd together.
iii) Bit 5 of the output will be Bits 6 and 5 of the input NOR'd together.
iv) Bit 4 of the output will be Bit 1 of the input. (ie: just shift the bit)
v) Bit 3 of the output will be Bits 6, 5, and 4 of the input XOR'd together.
vi) Bit 2 of the output will be Bits 2, 1, and 0 of the input XOR'd together.
vii) Bit 1 of the output will be Bit 4 of the input. (ie: just shift the bit)
viii) Bit 0 of the output will be Bits 6 and 5 of the output NAND'd together and then AND'd together with bit 0 of the input.

See Figure 1 below for a circuit configuration of the bit twiddling function you are to implement for this lab.

(g) Use your implementation to display the ASCII character "A" to the screen (yes that's a capital A not a lowercase a) An input of 0x65 on the dipswitches should give you this value. Check-off this functionality with your TA.

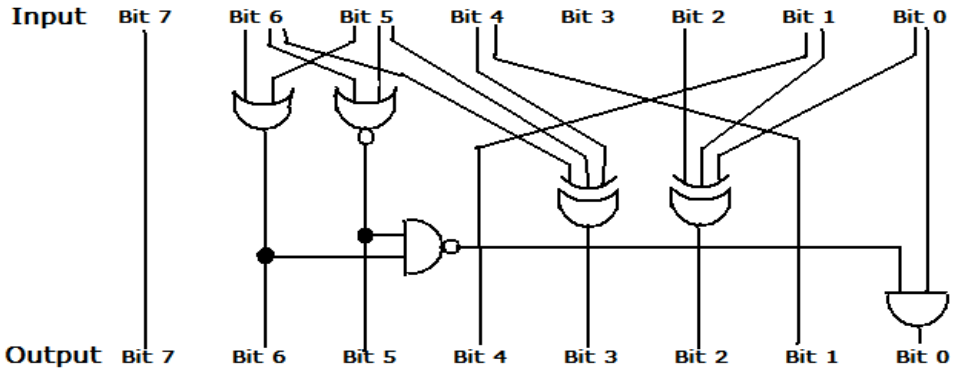

Figure 1.

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 3e 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.

3. Compile the following assembly code into object code:

```
        ldx #1234
        ldy $234,X
        ldaa 1,X+
        ldab 2,Y+
        cba
        beq done
        bgt loopb
loopa:  deca
        cba
        beq done
        bra loopa
loopb:  decb
        cba
        beq done
        bra loopb
done:   staa 1,-X
        stab 2,-Y
```