EE 357 Lab 1–Some Assembly Required

Prof. Annavaram

DUE: 5 PM., Tuesday, 09/25/2012

1 Introduction

In this lab you will write assembly code to control the 4 LED's on your MCF52259 board as well as read the input values of the 4 DIP switches.

2 What you will learn

This lab is intended to teach you to use the Codewarrior Eclipse IDE to write assembly code and then use the integrated debugger to test your code. In addition, you will learn the basics of using your microcontroller's I/O pins as digital inputs and outputs (termed General Purpose I/O, or GPIO for short).

3 Background Information and Notes

Codewarrior Eclipse IDE: Please read the "CodewarriorEclipse10.0 Setup" document on Blackboard and walk through the process of creating a new project, compiling/building the project, and then downloading/debugging it on your board before attempting this lab. It will provide direction for how to use the Eclipse IDE to write, compile, and debug software programs.

General Purpose I/O: The general purpose I/O module of the MCF52259 allows you to use almost any I/O pin as either a digital input or output where you can read from or write the value of the I/O pin via software control. However, most I/O pins can be used for several alternative functions only one of which is GPIO (i.e. a pin may be used as an Analog to Digital converter, as a USB input, or as GPIO). Thus, some initialization code may be required before an I/O pin can be used for this purpose. I/O pins are grouped into "ports" based on common functionality. Refer to the Coldfire 5225x Reference Manual chapter 15 (and Figure 15-1 for alternative pin functions).These ports can be controlled in a single access or on a bit by bit basis.

The MCF52259on-board I/O functions (2 Pushbuttons, 4 DIP switches, 4 LED's, 3-axis accelerometer and a potentiometer) are mapped to the following ports.

I/O Function	GPIO Port	Notes
PushButton - SW1	PORTTA[0]	Produces '0' when Pressed 0
PushButton - SW3 (there is no SW2)	PORTTA[1]	
DIP Switches 4 downto 1	PORTDD[7:4]	Produces '0' when in the 'ON' position
LED4 downto LED1	PORTTC[3:0]	Light is ON when a '1' is written
Accelerometer X-Axis	PORTAN[0]	
Accelerometer Y-Axis	PORTAN[1]	
Accelerometer Z-Axis	PORTAN[2]	
Potentiometer	PORTAN[3]	

Table 1 – MCF52259OnBoard I/O Connections

To control an I/O pin we must follow the procedure outlined below:

1. Initialize the port's "Pin Assignment Register" (PxxPAR) to indicate that the pins should be used for GPIO vs. there alternate functions. Usually writing a value of all 0's indicates GPIO.
2. Initialize the port's "Data Direction Register" (DDRxx) to select whether the pins should be used as inputs (write a '0' to the appropriate bit in the PxDDR) or outputs (write a '1' to the appropriate bit in the PxxDDR)
3. We are now ready to use the I/O pins as outputs or inputs:
    a. If the pins of the port are to be used as outputs, we can simply write a value to the PORTxx register which will then drive the actual I/O pins with the given value.
    b. If the pins of the port are to be used as inputs, we can now read from the SETxx register address to sample the current value of I/O pin.

Register	Description
PxxPAR (Port Assignment Register):	The value in this register configures the I/O pin for GPIO or one of its alternate functions. Usually a PxPAR value of 0's indicate GPIO.
DDRxx (Data Direction Register):	The value in this register configures the I/O pin to be an input or output.
PORTxx	The value in this register stores the output value of the I/O pin if DDRx is enabled (logic '1').
SETxx (Pin Data/Set Register)	A read of this address returns the current state of the I/O pin.

Table 2 – GPIO Control Ports

The addresses for the pertinent I/O control registers are shown below along with the bit values and connections. By using simple MOVE instructions we can deposit the necessary initialization values to appropriate registers are read the value of a register/pin inputs by simply 'moving' a value from its address.

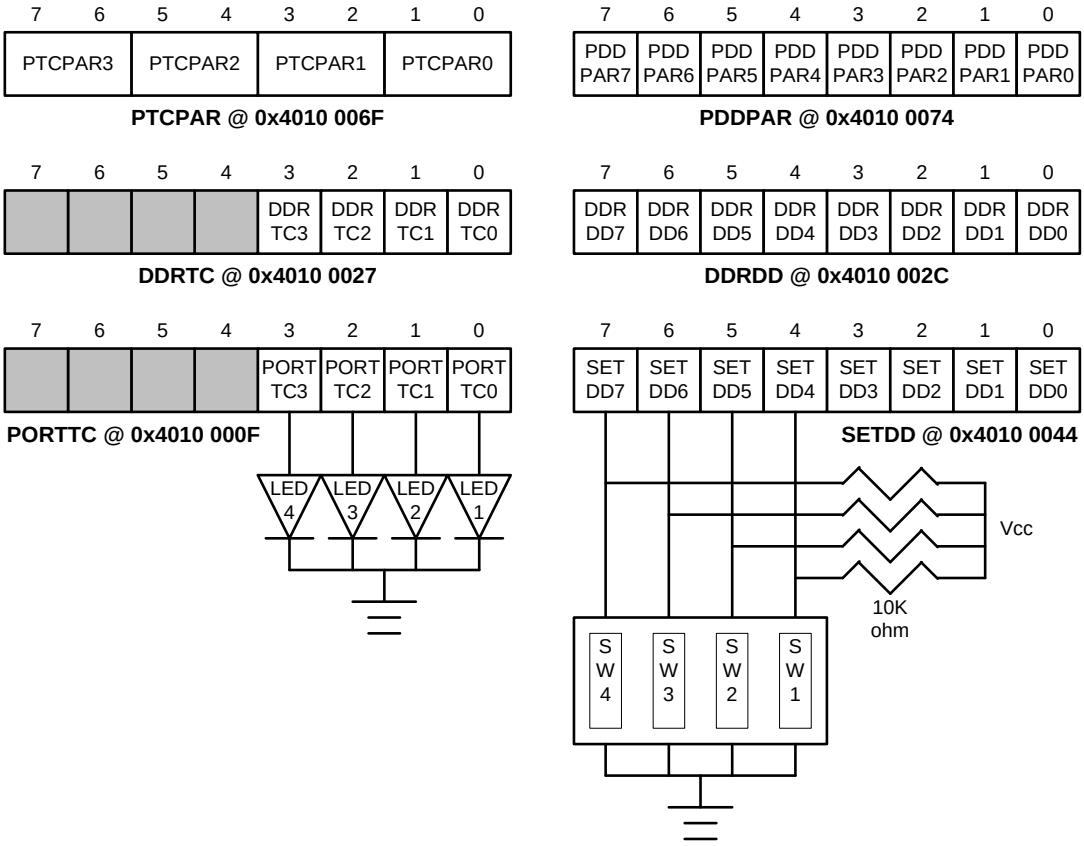

Figure 1 - I/O Control Registers and their addresses

Important: The LS 4-bits of PDDPAR and DDRDD are used for the debugger's operation and MUST REMAIN PROGRAMMED to 1's.

Blocks of Assembly in C: Most C compilers allow the programmer to embed assembly code sequences inside a C function by using an **`asm{ … }`** block.

```
intmain(void)
{
      int counter = 0;

      asm {
            move.l #0,d0
            ...
      }
```

4 Prelab

None

5 Procedure

1. Create a new project (call it "ee357_lab1_io_assem").
2. Open 'main.c' and add the following asm{} block of code before the for loop. You will need to fill in the appropriate addresses and values. This code will output a 4-bit binary number to the LED's (PORTTC), then increments the 4-bit binary number on the LED's and repeats (infinitely).

```
asm {
      move.l #____,d0
      move.b d0,0x4010006F // set PTCPAR for GPIO on all 4 bits
      move.l #____,d0
      move.b d0,0x40100027 // set DDRTC to make PTCPAR[3:0] outputs
      move.l #0,d0
      move.b d0,__________// set PORTTC to all 0's (i.e. LED's off)

      clr.l  d1
L1:   addq.l #1,d1
      andi.l #0x0f,d1
      move.b d1,0x4010000F
      bra    L1
}
```

3. Build the project and then start the debugger. Use the 'Step Over' button to walk through your code and ensure it is working. You can view the value of specific registers in the registers tab of the variable watch pane.
4. Now that you have some idea of how to output to the LED's, we will modify our program to include input from the 4 DIP switches. Go back and modify your code to produce a program that:
   a. Adds initialization of the appropriate I/O control registers so that you can use the 4 DIP switches on port DD as inputs. **Important: Recall that the LS 4-bits of PDDPAR and DDRDD are used for the debugger's operation and MUST REMAIN PROGRAMMED to 1's.**
   b. Keeps the infinite loop but rather than simply incrementing the 4-bit number displayed on the LED's, ***(1) reads the current values of the 4 DIP switches; (2) displays it on the corresponding LED's; and (3) keeps decrementing by 1 infinitely except that the number should be set to 1111 after it reaches 0000***. Because of how the 4 DIP switches are connected, placing the switch in the 'ON' position actually produces a '0' as input. While we can't change that, write your software such that placing the SWITCH x in the 'ON' position will actually cause LED x to light up (turn on). Important: Note that the DIP switch inputs are connected to the most significant 4 bits of Port DD while the LED's are connected to the least significant 4 bits of Port TC. You will need deal with this incongruence by writing appropriate code.

5. **Submission instructions:**

   (1) Turn in your working program (main.c) and review questions below (in a .pdf) on blackboard **individually**.

   (2) A date will be announced separately for demonstration.

6 Review questions

1. If we wanted to convert all the absolute address references used to read/write control registers to use **A.R.I with Displacement Mode**. Assume we include an instruction: MOVE.L #0x40100000,A1 at the start of our asm block. On your hard copy printout of main.c, show updated instructions using the ARI with displacement mode next to each instruction that uses an absolute address to access an I/O register.

Student Name: _____

Item	Outcome	Score	Max.
Register Init: • Correct Initialization of PAR Registers • Correct Initialization of DDR Registers	 Yes / No Yes / No		 1 1
Working Demo	Yes / No		5
Displacement Mode Modification • (3 = All correct, 2 = partially correct, 1 = Mostly incorrect, 0 = Missing)			3
SubTotal			10
Late Deductions (1 pt. per day)			
Total			10
Open Ended Comments:			