1. Make a copy of the **ADCSWTrigger_4C123** project and call it Lab 2. Compile and run the project either on the board or on the simulator. Observe the variable **`ADCvalue`** and the flashing of the LEDs on PF2 and PF1.

Done. PF1's toggling signifies that the main program is running. PF2's toggling signifies that the ISR is running. The ADC value changes during the ISR.

2. The microcontroller is executing at 80 MHz. Assuming assembly instructions average about 2 cycles per instruction, it takes about 25 ns to execute an instruction. Look at the following C code and associated assembly created by the compiler. Answer the following questions

a) What is the purpose of all the DCW statements?
They represent some of the key locations in the memory that this program needs to access.
For example, this program toggles PF1, and therefore it needs to access the location in the memory that represents PF1. In order to do so, it first needs to find the base location of Port F, which is 0x40025400 (got this from combining the contents at 0x00000686 and 0x0000684).

b) The main program toggles PF1. Neglecting interrupts for this part, estimate how fast PF1 will toggle.
6 instructions/loop * 25ns/instruction = 150ns/loop

c) What is in R0 after the first LDR is executed? What is in R0 after the second LDR is executed?
The base location for Port F.
Current state of PF1.

d) How would you have written the compiler to remove an instruction?
Here we are getting the memory address of PF1 twice, which is completely unnecessary.
I will write the compiler such that it produces the following Assembly code for the same C code.

```
LDR r0,[pc,#24]   ; @0x0000068C
LDR r1,[r0,#0x08]
EOR r1,r1,#0x02
STR r1,[r0,#0x08]
B   0x0000067E
```

In fact, the Last line "`B 0x0000067E`" can be replaced by "`B 0x00000680`" to save even more time, because r0 does not chagne, and therefore the base location for Port F doesn't need to be fetched again.

e) 100-Hz ADC sampling occurs in the Timer0 ISR. The ISR toggles PF2 three times. Toggling three times in the ISR allows you to measure both the time to execute the ISR and the time between interrupts. See Figure 2.1. Do these two read-modify write sequences to Port F create a critical section? If yes, describe how to remove the critical section? If no, justify your answer?
No. We are not using GPIO_PORTF_DATA_R, but PF1 and PF2, which means the main and the ISR only read-modify-write their target Pin and have no effect on other pins on the same port.

3. Write software that initializes one of the 32-bit timers to count every 12.5ns continuously. Basically copy the initialization from the **PeriodicTimer1AInts_4C123** project, set the **TIMER1_TAILR_R** to 0xFFFFFFFF, and remove the interrupt enable and interrupt arm statements. This way, reading the Timer 1 **TIMER1_TAR_R** will return the current time in 12.5ns units. The timer counts down. To measure elapsed time we read **TIMER1_TAR_R** twice and subtract the second measurement from the first. $12.5ns*2^{32}$ is 53 seconds. So this approach will be valid for measuring elapsed times less than 53 seconds. The time measurement resolution is 12.5 ns.

All modified code can be found under the step3 folder.

4. Define two arrays of 1000 entries each. Add debugging dumps to the Timer0 ISR to record the time and ADC data value for the first 1000 ADC measurements. At 100 Hz sampling, these arrays will fill in 10 seconds. Once the arrays are full, stop recording. If you wish to run faster you could increase the sampling rate to 1000 Hz.

The modified ADCTestMain.c can be found under the step4 folder.

5. Write main program software to process the time recordings. This software will be run after the arrays are full (and not during the data collection phase.) Calculate 999 time differences and determine time jitter, which is the difference between the largest and smallest time difference. The objective is to start the ADC every 10 ms, so any time difference other than 10 ms is an error. Real-time ADC sampling requires time jitter to be small.

The modified ADCTestMain.c can be found under the step5 folder. The calculated Jitter is 5.

6. Write main program software to process the data recordings. This software will be run after the arrays are full (and not during the data collection phase.) We will assume the analog input is fixed, so any variations in data will be caused by noise. Create a probability mass function of the measured data. The shape of this curve describes the noise process. The x-axis has the ADC value and the y-axis is the number of occurrences of that ADC value.

The modified ADCTestMain.c can be found under the step6 folder.