**Lab 2 Deliverables**

Dhruv Sandesara

Sean Tremblay

**Objectives** • To develop software debugging techniques,

- Performance debugging (dynamic or real time)

- Profiling (detection and visualization of program activity)

• To dump time and data values into arrays

• To learn how to use the oscilloscope and logic analyzer,

• To experience concepts of real time, probability mass function and Central Limit Theorem

• To observe critical sections

Prep Question:

a) What is the purpose of all the DCW statements?

Ans: It is to store the addresses of ports

b) The main program toggles PF1. Neglecting interrupts for this part, estimate how fast PF1 will toggle.

Ans: 1 time every 6 instructions. Therefore 1 time every 80/6 MHz = 13.333 MHz

c) What is in R0 after the first LDR is executed? What is in R0 after the second LDR is executed?

Address of PF1 after 1st LDR, Data in PF1 after 2nd LDR.

d) How would you have written the compiler to remove an instruction?

Ans: Change 1st LDR’s r0 to r1 and get rid of 3rd LDR

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?

Ans: Yes this is a critical section since if the ISR is triggered after the 2nd LDR and before the STR there is a chance that PF2 has been switched and thus the date that we might output through the main fuction will still contain the old value of PF2 and thus will be incorrect.

**Procedure (do this during lab)**

**A. Learn how to use an oscilloscope**

You are expected to learn how to use an oscilloscope in this class, so, please ask your TA for a demonstration if you are unfamiliar with the features of the scopes we have in lab. In particular, you should: 1) be able to adjust the time base and voltage scales; 2) know how to set/adjust the trigger; 3) understand AC/DC mode; 4) be able to measure a frequency spectrum; 5) understand the resistive and capacitive load of the scope probe; 6) pulse width measurement using time cursors; 7) measure voltage amplitude using the voltage cursors; and 8) be able to save waveforms to USB flash drive for printout later. Line trigger mode is very useful for identifying the presence of 60 Hz AC-coupled noise.

*Deliverable*: Use a two channel scope to measure the debugging profile like Figure 2.1, and use the scope to estimate the time required to take one sample (interrupt, ADC, return from interrupt). Place a picture of the scope trace (photo or digital download) into your lab manual.

**B. Learn how to use a logic analyzer**

You are expected to learn how to use a logic analyzer in this class, so, please ask your TA for a demonstration if you are unfamiliar with the features of the logic analyzers we have in lab.

*Deliverable*: Use the logic analyzer to measure the debugging profile like Figures 2.1 and 2.2, and use the logic analyzer to estimate percentage of time running in the main versus running in the ISR. Place a picture of the scope trace (photo or digital download) into your lab manual.

**C. Experience a critical section**

In the main program change this line

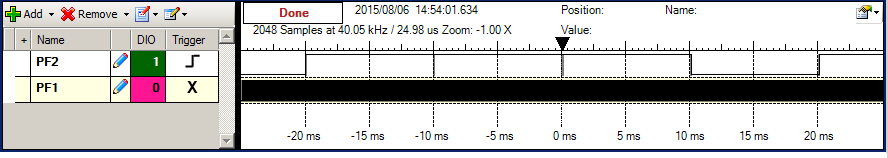
**PF1 ^= 0x02; // toggles when running in main**

to this line (creating a critical section)

**GPIO\_PORTF\_DATA\_R ^= 0x02; // toggles when running in main**

Use either the scope or the logic analyzer to observe the critical section. You should see something like Figure 2.4.

*Deliverable*: Which pin is incorrect PF1 or PF2? Look at the assembly language of the main program and explain the sequence of steps that results in the corruption of the debugging profile. Other than using bit specific addressing (PF1 PF2) what other solutions could you have used to have two debugging profile pins without a critical section?



*Figure 2.4. Zoomed out view of the PF1 PF2 recording illustrating the results of the critical section (compare to Figure 2.2).*

**D. Prove the system is real time**

Debug your time dump and jitter calculation as written in preparations 4 and 5. Perform two or three runs to see if you always get the same time jitter measurement. With just one interrupt active, any jitter you do get (you might not see any) is caused by the random event of which instruction in the main program was being executed at the time of the interrupt trigger. If you want to see this single interrupt system have jitter place some code in the main program loop that execute an integer divide (the divide instruction takes 2 to 12 cycles):

**PF1 = (PF1\*12345678)/1234567+0x02; // this line causes jitter**

*Deliverable*: Measure time jitter with just the one sampling interrupt active.

This Timer 0 ISR is running at priority 2. Add one or two other 10kHz periodic interrupts (SysTick Timer2 or Timer3) running at priority 1. Make the period close to 10 kHz but not exactly equal to 10 kHz (I ran mine at 99 us period). Repeat the time-jitter measurement. (When we get to the ADC lab in this class we will use Timer triggered ADC sampling which will completely remove the time jitter caused by higher priority interrupts).

*Deliverable*: Measure time jitter with two or more interrupts active. Try to generalize the results deriving a theoretical estimate of the time jitter of the periodic ADC sampling using software triggering.

**E. ADC noise measurements.**

Create a constant voltage of any value greater than 0 and less than 3.3V. You can use a shunt diode, two resistors between +3.3 and ground, or a battery. It is ok if the voltage is noisy because we are studying noise. However we do wish for the average voltage to be constant, not drifting up or down. Before connecting to the microcontroller use the DVM to measure the DC and AC voltages of this constant. DVMs use RMS to measure AC voltage so the AC voltage is a standard way to measure noise. The less noise in the constant the more this section will evaluate ADC noise. On the other hand, the more noise you have the more dramatic your data will look.

Connect the constant voltage to the ADC input. If you debug your software in the simulator, you should see all ADC data values the same. So debug this part on the real board. Even though Figure 2.3 was taken with 64-point hardware averaging, do not activate averaging for this part. Use the software in preparations 4 and 6 to create the probability mass function for the ADC noise. You will have to run it a few times to see typical data so you can set the range of the PMF x-axis. You can either plot the PMF data on the ST7735 LCD using any of the LCD plotting routines, or you can send the data from the LaunchPad to the PC and plot the PMF in MatLab or Excel. If you plot the data on the ST7735 use a camera to capture the plot into your lab report. How would you describe the shape of the noise process? When you run it over and over do you get the same shape of the PMF? *Hint: you should not get the same PMF because the noise is not stationary.*

**F. Central Limit Theorem.**

Look up the ADC Sample Averaging Control (**ADC0\_SAC\_R**) register in the Chapter 13 of the data sheet.

*Deliverable*: Plot PMF for hardware averaging of none, 4x, 16x, and 64x. In each case the sampling rate is fixed and there are 1000 data points used to plot the PMF function. Describe qualitatively the effect of hardware averaging on the noise process. Consider two issues 1) the shape of the PMF and 2) the signal to noise ratio. *Hint: CTL.*

*Deliverable*: Use the logic analyzer or scope to determine the effect of hardware averaging on the time to execute the ISR. Why is the thread profile like Figure 2.1 very different with hardware averaging?

**Fun activity.** Noise can vary, so before you generalize from the data you collected in this lab, go around the lab room and look at the data from other groups.

**Deliverables (exact components of the lab report)**

A) Objectives (1/2 page maximum). Simply repeat the items shown in the **Goals** section

C) Software Design (the final solution with data/time dump, time jitter and PMF calculations).

D) Measurement Data (you may take photographs of the scope/logic analyzer/LCD or download the data)

**Prep part 2)** Show your answers to the five questions a – f.

**Part A)** Debugging profile with scope

**Part B)** Debugging profile with logic analyzer, estimation of percentage time in main/ISR

**Part C)** Explain the critical section and present alternate solutions to removing it

**Part D)** Time-jitter measurements and generalization of factors that contribute to jitter

**Part E and F)** PMF data and discussion of results. Does your data support CLT? If not why?

**Part F)** Debugging profile of execution time in ISR with hardware averaging. Why is it different?

**Analysis and Discussion (give short 1 or two sentence answers to these questions)**

1. The ISR toggles PF2 three times. Is this debugging intrusive, nonintrusive or minimally intrusive? Justify your answer.

Ans: This is minimally intrusive as toggling the ISR takes only 1-2 instruction cycles.

1. In this lab we dumped strategic information into arrays and processed the arrays later. Notice this approach gives us similar information we could have generated with a printf statement. In ways are printf statements better than dumps? In what ways are dumps better than printf statements?

Ans: Dumps are much less intrusive than printf statemnts. The pintf statement are intrusive as writing to console is slow buisnes

1. What are the necessary conditions for a critical section to occur? In other words, what type of software activities might result in a critical section?

Ans: When an ISR changes some value which is also being changed in a main function when interrupts are not disabled

1. Define “minimally intrusive”.

Ans: minimally intrusive means that it uses relatively less instructions to perform the task.

1. The PMF results should show hardware averaging is less noisy than not averaging. If it is so good why don’t we always use it?

Ans: This is as the sampling rate at which we get the results is cut down significantly as it takes the average of results to give an averaged result. This 4x averaging gives a speed reduction by a factor of 4.