EE445L - Lab 2: Performance Debugging

Xinyuan (Allen) Pan and Paris Kaman
9/12/2017

Objective

In this lab we learn to use software debugging techniques such as dumps, and use equipments such as oscilloscope and logic analyzer. We experience fundamental concepts of real time, critical sections, probability mass function (PMF), and the Central Limit Theorem (CLT).

Preparation

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

The DCW statements are just filled with the values of the registers that you need to load and then use for the LDR to find the values of the ports

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

6 instructions * 25ns = 150ns

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

1st - the address of PF1
2nd - the value of the contents of PF1

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

Make the second LDR use r1 so that you keep the location in r0 because you are going to have to store it there anyway. This would get rid of the third LDR

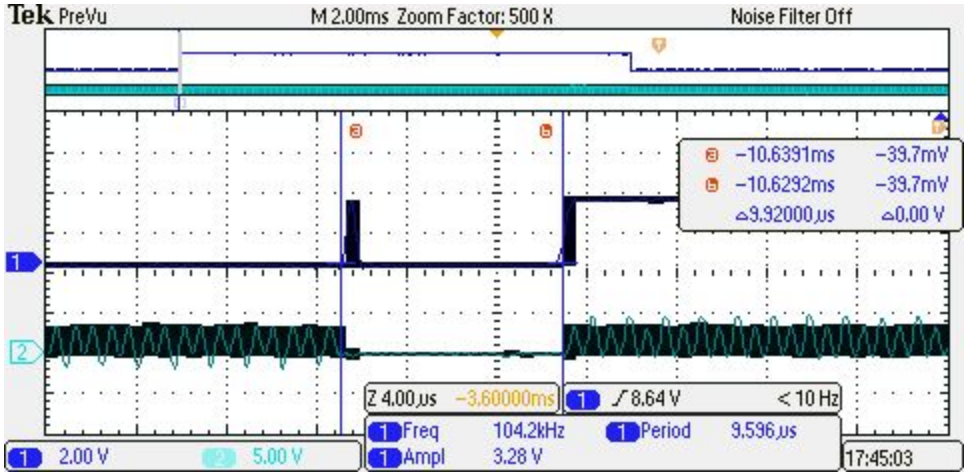
e) 100-Hz ADC sampling occurs in the Timer0 ISR. The ISR toggles PF2 three times. Toggling three times in the ISR allows you to measure both the time to execute the ISR and the time between interrupts. See Figure 2.1. Do these two read-modify write sequences to Port F create a critical section? If yes, describe how to remove the critical section? If no, justify your answer?

This shouldn't create a critical section because the memory is banded, so when you try to toggle using PF2, it changes the PF2 memory location which then only writes to the bit2 on the big PORTF register.

Procedure

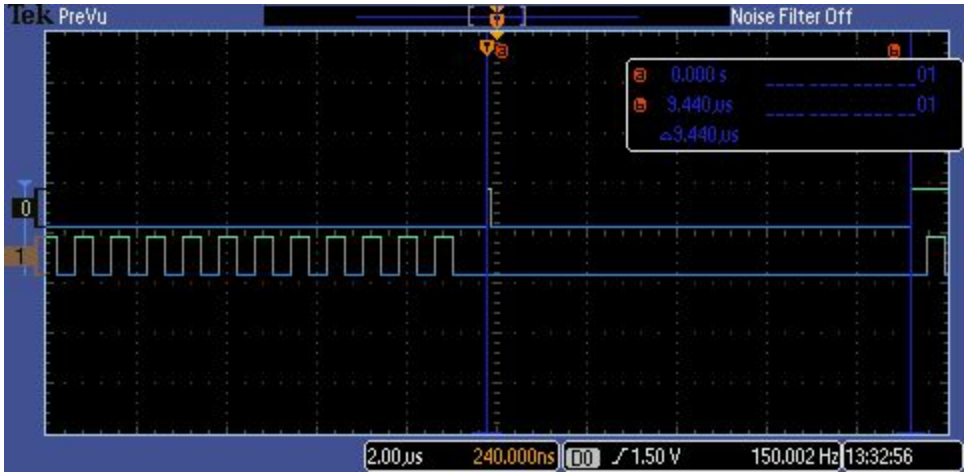
- Part A

  The time required to take one sample: 9.920 microseconds (should be less, the cursor was not well placed during the lab.)

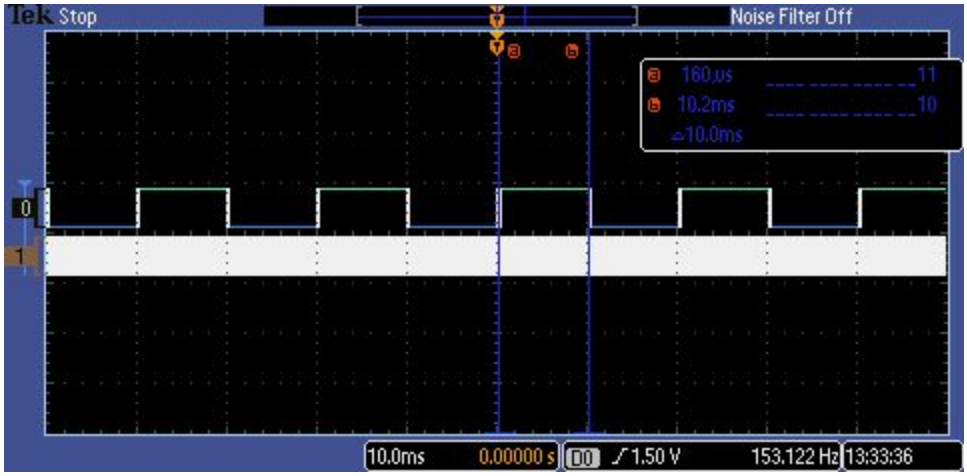


- Part B

  Time spent to run in ISR: 9.440 microseconds

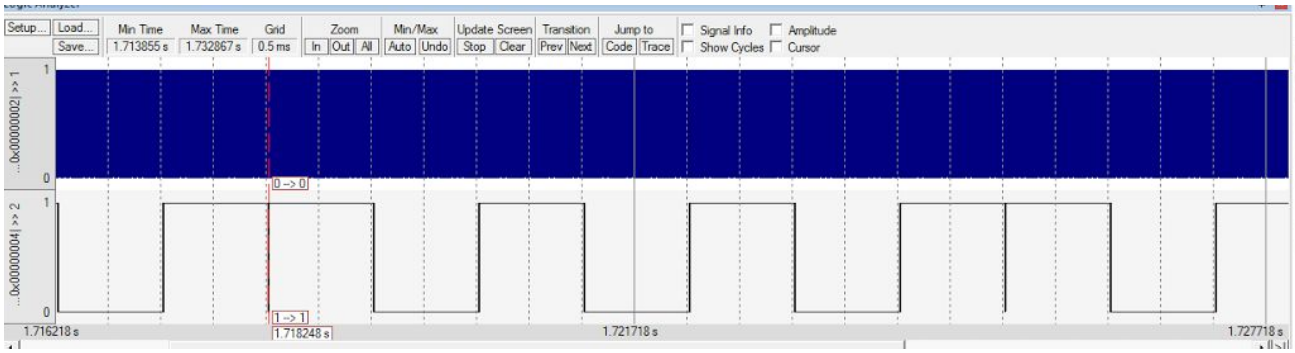


  Time between two consecutive interrupts: 10 milliseconds



Percentage of time running in the main program: ~99.906%

- Part C



PF2 is incorrect.
Before the interrupt, the value of the whole PORT F is saved in a register (with PF1 toggled), but not yet written to the memory. At this time, interrupt occurs, so PF2 gets toggled in the interrupt. And then when the main function tries to store the value of the toggled PF1, the PORT F value it has loaded in the register is the old value before the interrupt, so PF2 isn't toggled. Therefore, PF2 value is corrupted.
This can be solved by disabling interrupts right before you read the PORTF value and re-enable them right after you toggle the bit and successfully store it back into memory. Or, if in C, disabling right before the line `GPIO_PORTF_DATA_R ^= 0x02;`

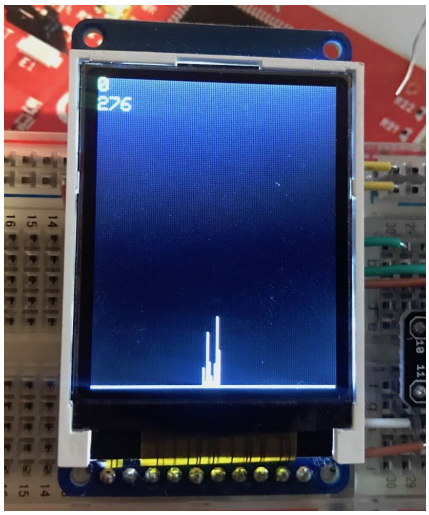
- Part D

Jitter with one sampling interrupt: 71942 (899.275 us)

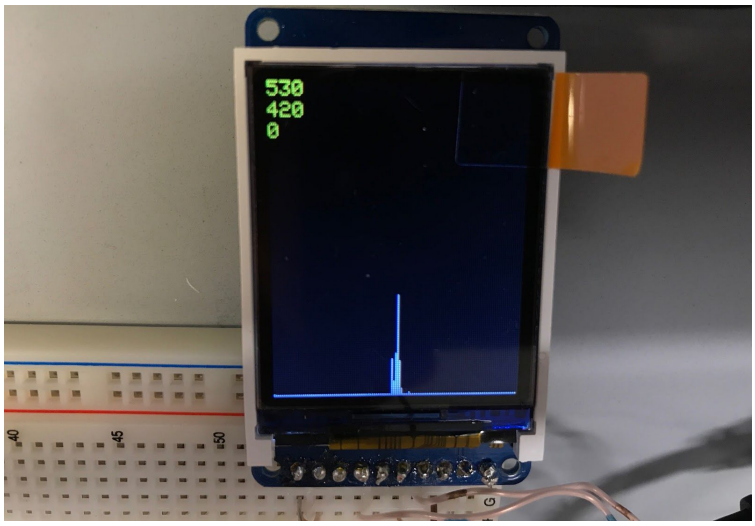
Jitter with 2 interrupts: 71954 (899.425 us)
There is a correlation between the number of interrupts and the extent of jitter. The more interrupts we have the larger the jitter.

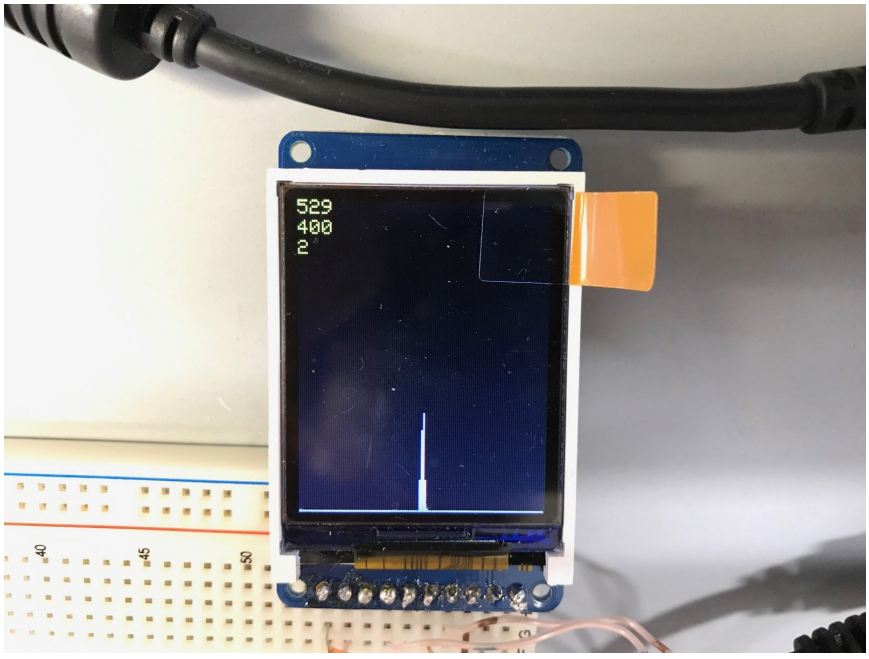
- Part E

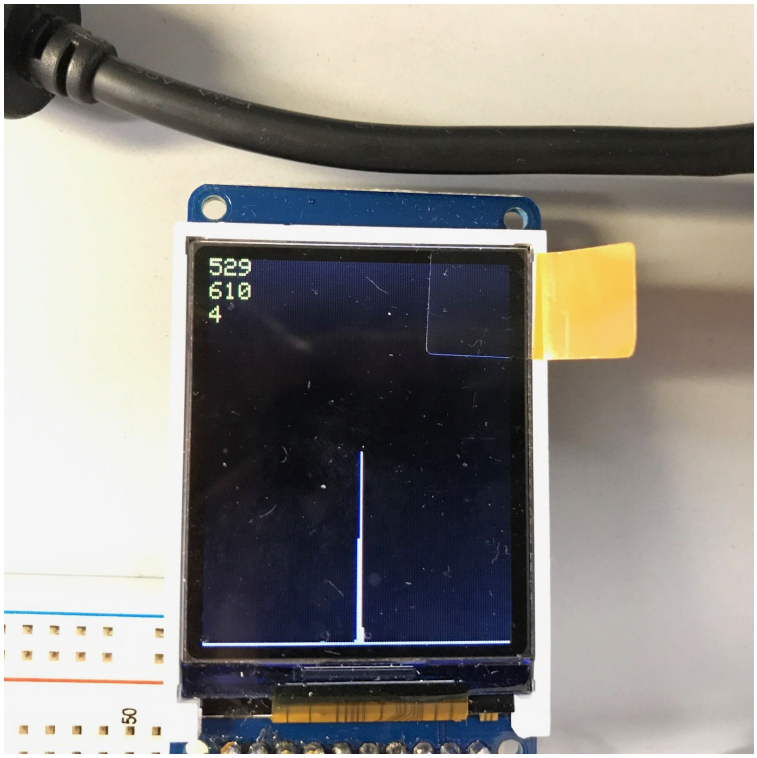
The shape of the noise is generally bell-shaped with every execution. I am not sure why we are not getting random distributions. The noise seems to be very small for us. But when we activate hardware averaging for the next part, the effect is still obvious. We found the ADC value with the largest number of samples, and print that number on the second line. You can see the value is quite large (out of 1000).

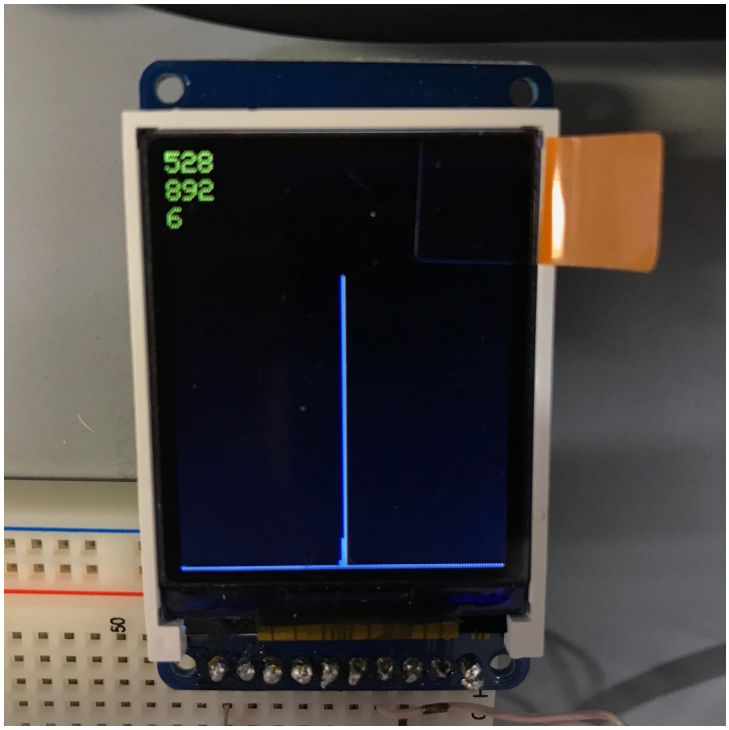


- Part F

No Average: 

Averaging of 4x: 

Averaging of 16x: 

Averaging of 64x: 

Clearly, hardware averaging narrows down the distribution, and concentrates on a few ADC values. You can see on the second line in each screenshot, the number of samples taken at the peak ADC value increases dramatically. Noise is reduced even though the noise for us was already pretty small at the beginning. So, out data does support CLT.

The ISR takes considerably longer time to execute each time we increase the sample averaging number. Therefore, the thread profile looks very different.

Analysis & Discussion

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

Minimally intrusive. We can see from the profiling figures, that the time taken to toggle PF2 is neglectable compared to the execution time of the ISR, which is already very small compared to the interrupt period.

2) 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?

Printf statement does not take any memory to store the arrays. Dumps take a huge amount to memory space, but it only takes a little time in each ISR to acquire the information. Printf takes the extra time to display the information, therefore, may affect the functionality of a real time system.

3) 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?

Critical sections may occur if there is a causal dependency between threads. For example, if two threads access the same global memory and one of the accesses is write, the the execution order may affect the outcome.

4) Define “minimally intrusive”.

Minimally Intrusive refers to debugging techniques that do technically change the way the program executes, but the effects or which are hardly noticeable.

5) 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?

Because it does take significantly longer for the ISR to execute.