Daniel Diamont

Robert Noe

Lab 2 Report

Goals:

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

DCD statements are there to provide access to the following memory-mapped I/O: RCGCGPIO (0x400F.E608), GPIODIR (0x4002.5400), PORTF\_BASE (0x4002.5000)

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

PF1 will toggle every 150 nanoseconds

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

R0 has the address of the Port F GPIO Data register after the first LDR instruction. After the second LDR instruction, R0 has the state of PF1 with the state of all the other pins in port F shown as cleared.

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

I would have written the compiler to fetch the state of PF1 in one instruction:

0x0000068C DCW 0x4002

DCW 0x5008

LDR R0, [pc, #24];

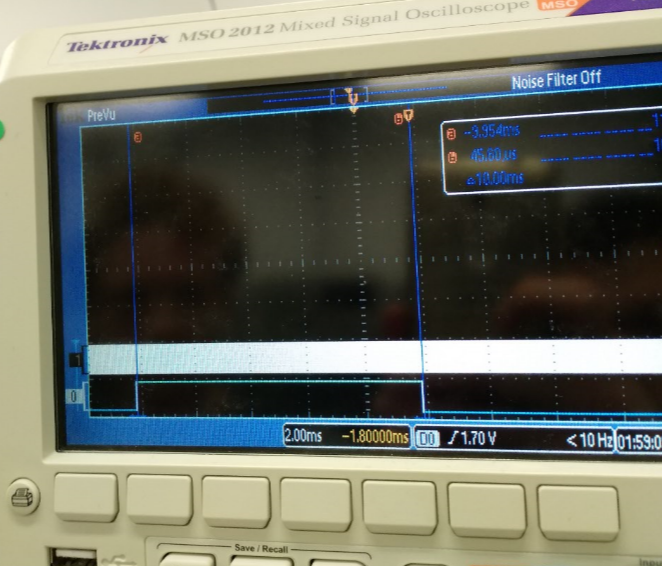
e) 100-Hz ADC sampling occurs in the Timer0 ISR. The ISR toggles PF2 three times. Toggling three times in the ISR allows you to measure both the time to execute the ISR and the time between interrupts. See Figure 2.1. Do these two read-modify write sequences to Port F create a critical section? If yes,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

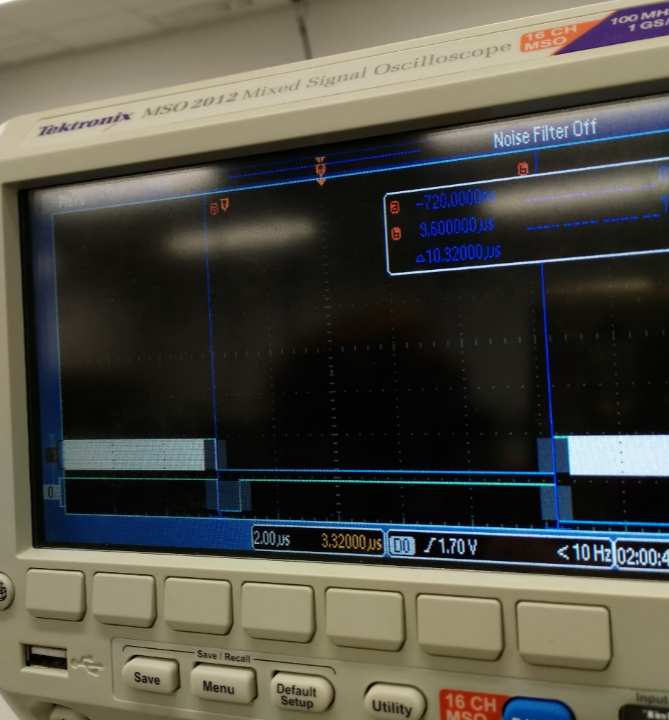
Since the main program is only interested in PF1 by using bit-specific addressing, any changes made by the ISR to PF2, which also uses bit specific addressing, will not affect the main program’s read and write to PF1. In other words, there is no critical section between the two threads.

**Part A)** Debugging profile with scope



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





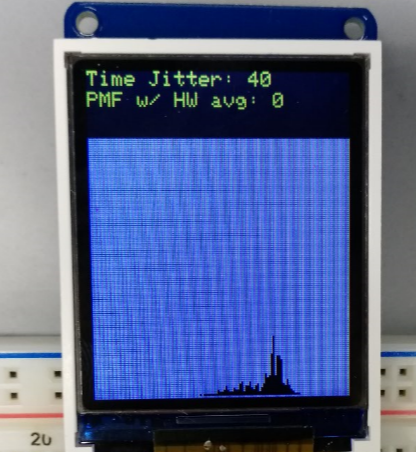
Let *t,* be the percentage time spent in ISR and away from main thread.

*%t =* (10.32 us / 10.00 ms)\*100 = 0.1032 %

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

The critical section occurs because even though the main thread is using bit-specific addressing to toggle PF1, the ISR reads and writes to the entire Port F in order to toggle PF2. Thus, anywhere in between the read-modify-write cycle to the port during the ISR, there can develop a race condition between the threads. Two ways to remove this problem are: A) ensure that the most important thread is atomic, thereby ensuring the same outcome in every situation. B) Use bit-specific addressing to completely avoid a critical section by reducing each threads access to a specific pin, rather than reading and writing to the entire port.

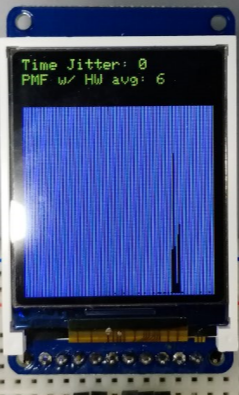
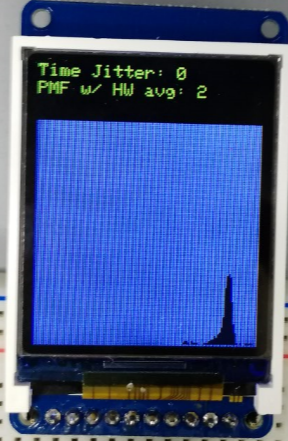
**Part D)** Time-jitter measurements and generalization of factors that contribute to jitter



The above photos are examples of two trials runs in which our software detected noticeable differences between the maximum and minimum time between threads. In the first case, there is a jitter of 8 bus cycles (100 nanoseconds), and in the second case there is time jitter of 40 bus cycles (500 nanoseconds).

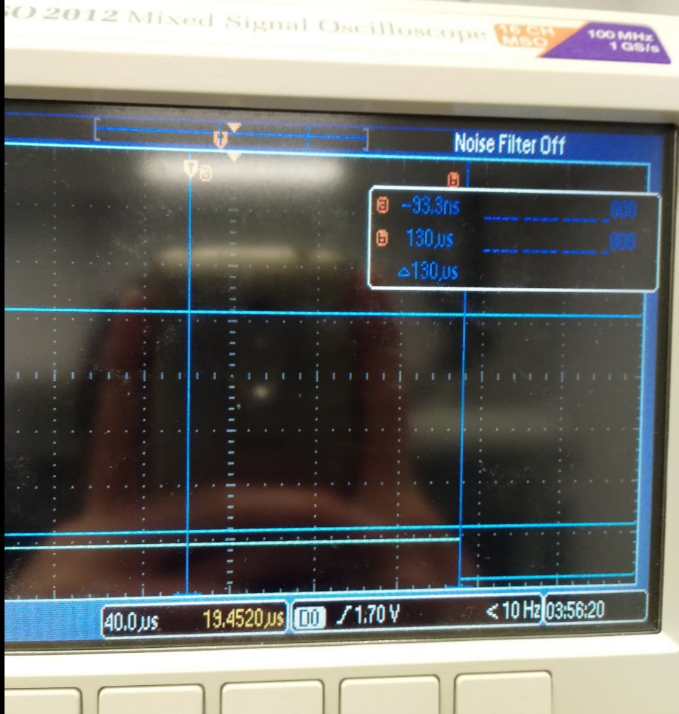
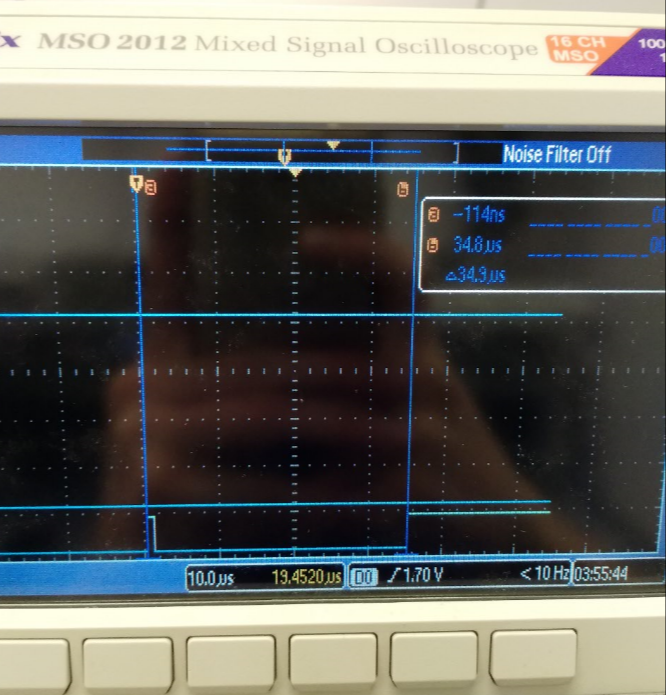
Some factors that can contribute to jitter include programming techniques for interfacing with data acquisition hardware that include busy waiting or having interrupt services routines which take up a noticeable percentage of time away from the main program. Additionally, adding more interrupts increases the changes that the time critical interrupt will not be serviced under a hard-real-time constraint, as the processor may be occupied servicing higher priority tasks. Lastly, jitter may be caused by a processor executing instructions with a high average quantity of bus cycles to execute, such as integer division which takes about 2 to 12 bus cycles. This can delay the interrupt from being serviced with a hard deadline.

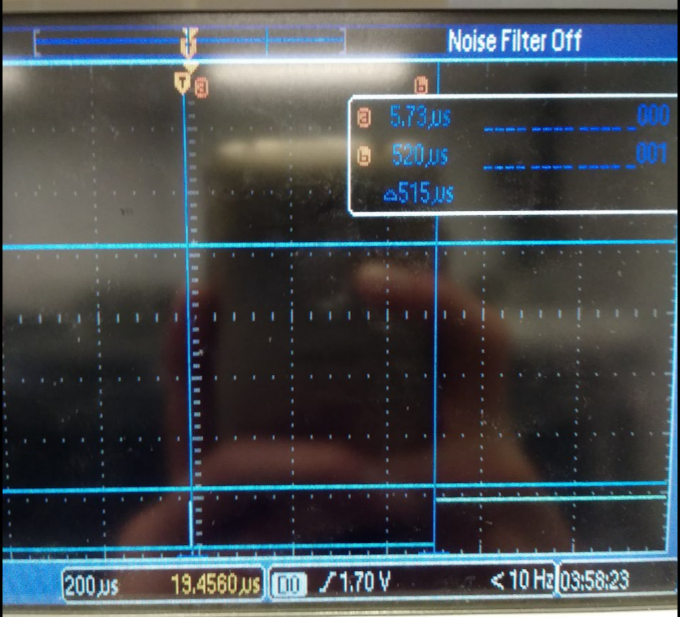
**Part E and F)** PMF data and discussion of results. Does your data support CLT? If not why?



Yes, our data supports the Central Limit Theorem. By increasing the averaging factor of our data acquisition hardware, the standard deviation of our Gaussian distribution decreased. In other words, when we took an increasing number of sample points, the error increased, and the data points converged closer towards the mean of the distribution. This supports central limit theorem which argues that as the number of random variables, or independent samples, that are added together increases to infinity, the probability mass function will generate a plot that approaches that of a normal distribution.

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





The debugging profiles are different because the increased factor for hardware averaging causes the hardware to take longer to sample the signal. Thus, the interrupt service routine must busy wait the hardware for information for larger periods of time, given a larger hardware averaging factor to take more sample points.