**EE445L – Lab 2: Performance Debugging**

Karime Saad and Tarang Khandpur

September 22, 2017

**1.0 OBJECTIVE**

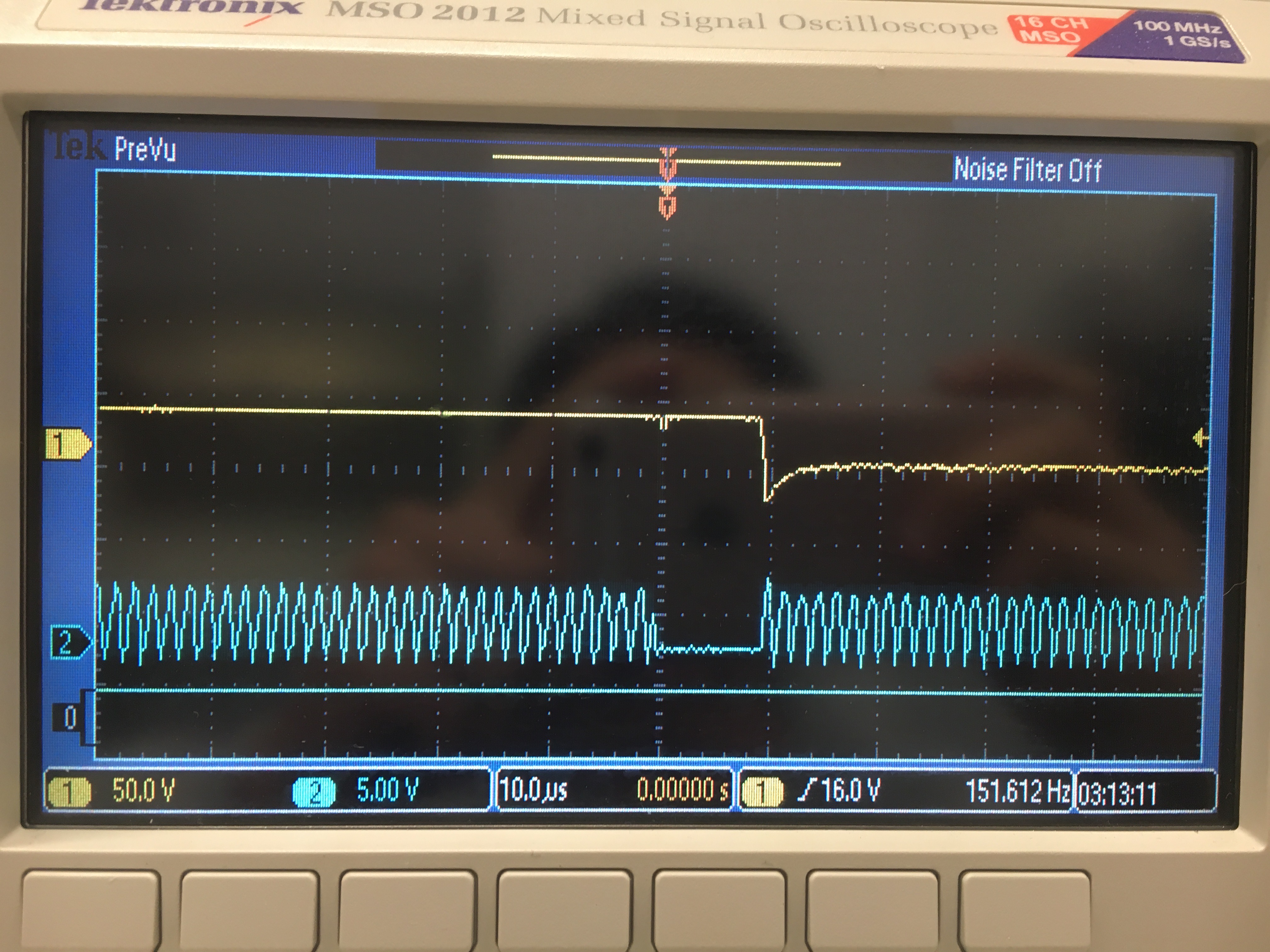
* Our objectives for this lab are to understand methods of performing minimally invasive debugging techniques, observe real-time system performance using an oscilloscope and logic analyzer, and apply statistical analysis methods to our program. To accomplish these tasks we observed various values of input voltage noise through the ADC, placed them into arrays, and plotted them onto a PMF. We then applied the Central Limit Theorem to observe the distribution. In addition to observing noise, we also performed various interrupts and noted their behavior and timing using the Logic Analyzer. This allowed us to view the critical sections that we created for testing their behavior during the interrupts.

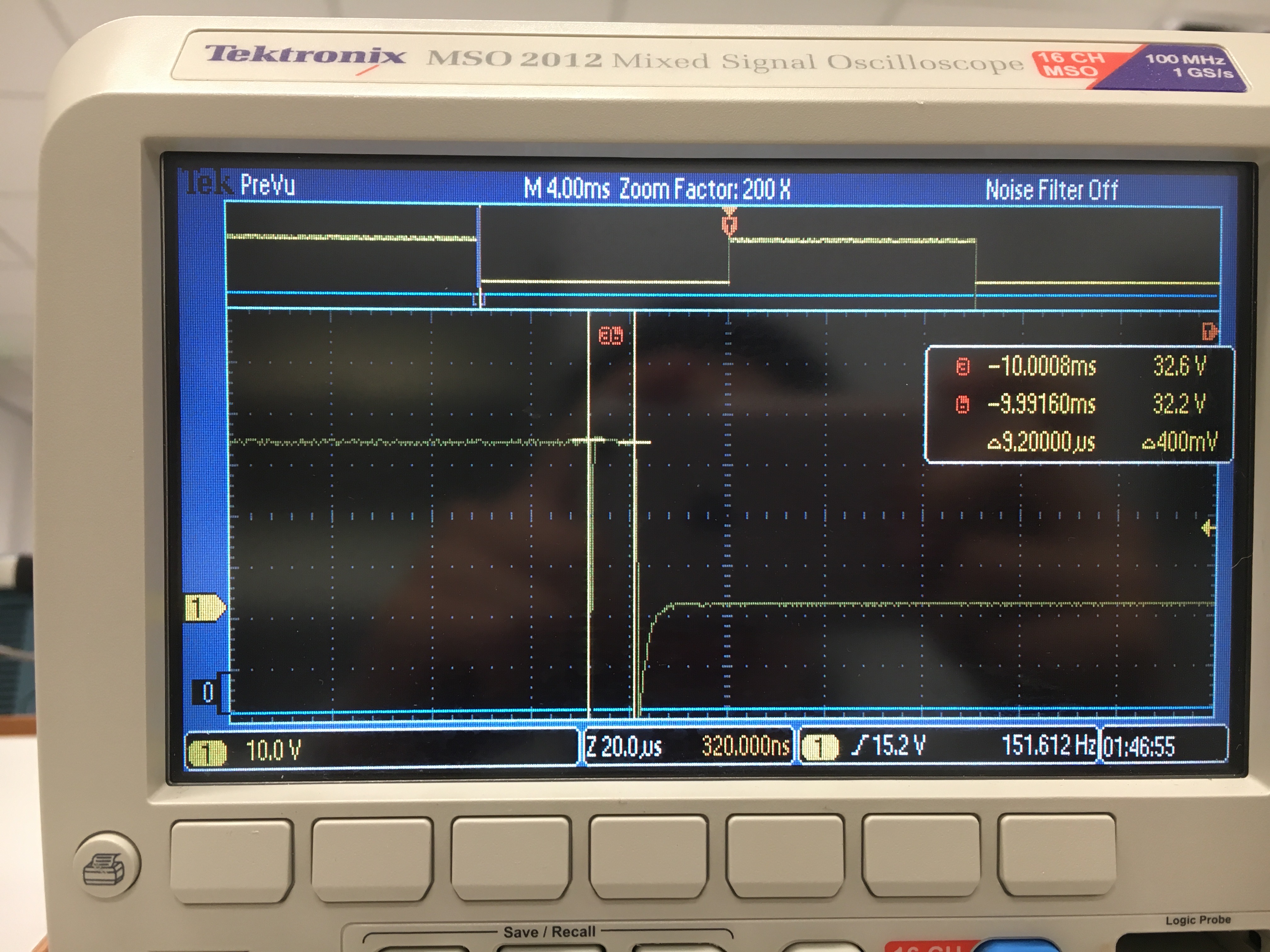
**2.0 PROCEDURE DELIVERABLES**

**2.1 Learn how to use an oscilloscope (Part A)**

*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.

* Time required to take one sample (interrupt, ADC, return from interrupt): 9.2microseconds

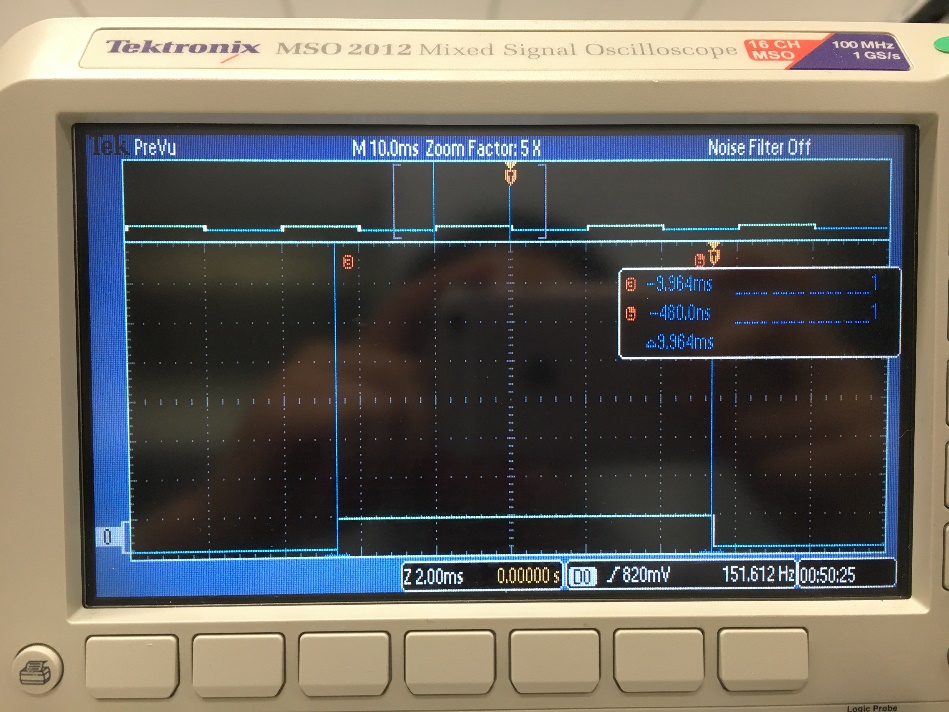
****



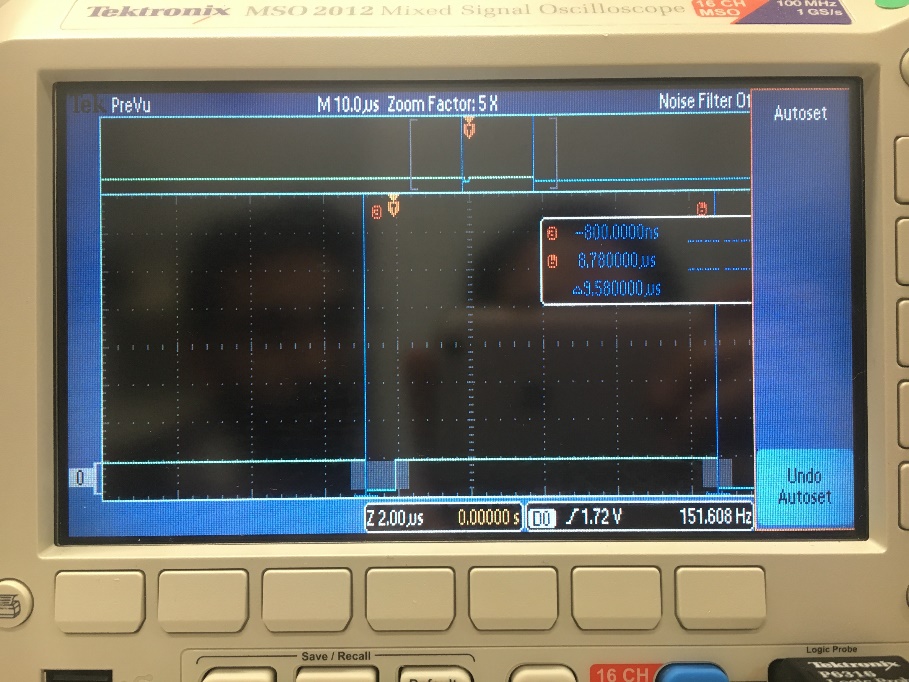
**2.2 Lean how to use a logic analyzer (Part B)**

*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.

* TK - Time running in Main including interrupt: 9.964 milliseconds
* Time running in Main: 9.964 milliseconds



* Time running in the ISR: 9.58 microseconds (0.00958 milliseconds)



* Percentage time running in the main versus running in the ISR:

*TK - (9.964ms - .00958ms)/9.964ms = 0.999*

* Therefore, Main runs for **99.9%** of the execution time of the program.

**2.3 Experience a critical section (Part C)**

*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?

* PF2 is incorrect. Assume PF1 & PF2 were high before the interrupt. Before the interrupt is called the complete value of the GPIO\_PORTF\_DATA is pushed onto the stack. During the interrupt the PF2 toggles twice, takes ADC reading and toggles again to become low. However when the control returns back to main the original values in GPIO\_PORTF\_DATA are popped back from the stack which had PF2 high, so PF2 becomes high whereas it should have been low.

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

0x00001698 48C4 LDR r0,[pc,#784] ; @0x000019AC

0x0000169A 6800 LDR r0,[r0,#0x00]

0x0000169C F0800002 EOR r0,r0,#0x02

0x000016A0 F8C103FC STR r0,[r1,#0x3FC]

If the interrupt is called within the above lines, then the original PF2 value would be lost. Other than bit addressing one could have disabled the interrupts during this non-atomic block of code.

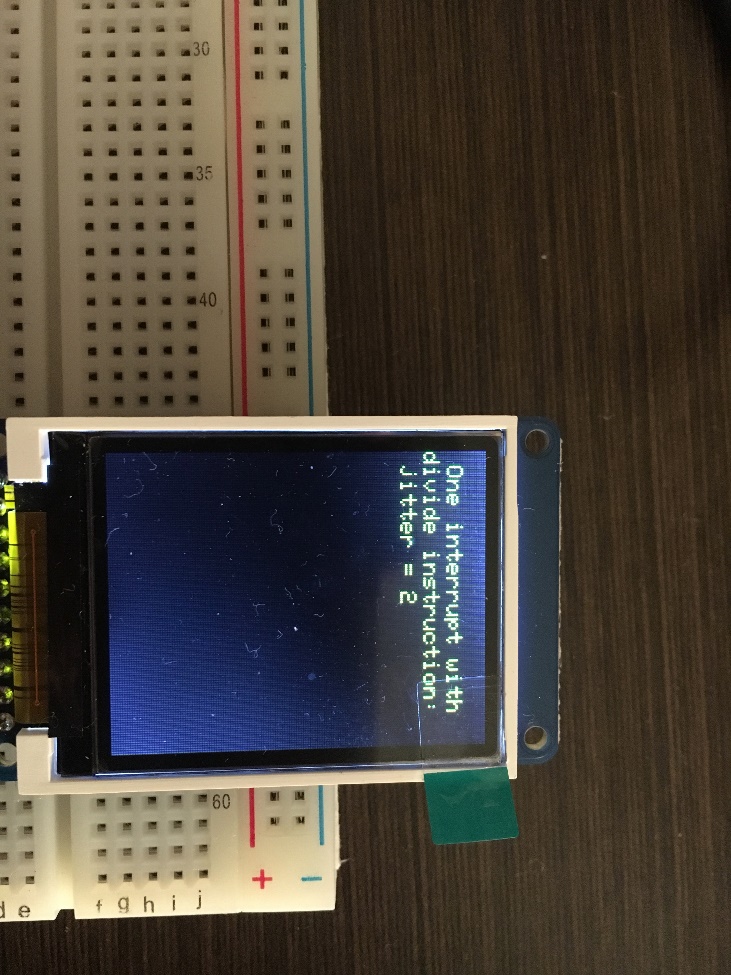
**2.4 Prove the system is real time (Part D)**

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

* One interrupt with divide instruction:
  + jitter: 2 cycles (25 ns)

*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.

* + *Generalizing:* As we add interrupts, the jitter will increase; however, the amount of jitter in more interrupts will diminish.
* Two interrupts with divide instruction:
  + jitter: 52 cycles (650 ns)
* Three interrupts with divide instruction:
  + jitter: 62 cycles (775 ns)

**2.5 PMF data and discussion of results. 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? (Part E)**

The PMF represents an approximate Gaussian distribution; however, because there is sufficient noise, the same exact shape is not reproduced for each subsequent run.

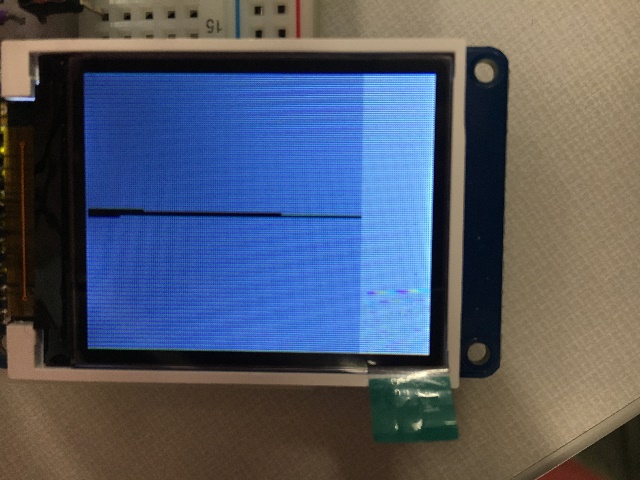
**2.6 Central Limit Theorem. (Part F)**

*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.*

* + 1) The shape of the PMF will become more normal/guassian as the Hardware Averaging value increases. This occurs because according to the Central Limit Theorem, as more samples are taking of any averaged statistic the distrubtion of that statistic will be normal. And a higher hardware averaging value simply translates to 4x, 16x, or 64x as many samples per reading.
  + 2) The signal to noise ratio improves because the averaging of the values acts as a trimmed mean of the input noise values, resulting in a more precise ADC measurement.

TK – Could you add images for 4x, and 16x? Just change the line in ADCWtrigger.c

0 – Hardware Averaging 64x – Hardware Averaging

*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?

TK – Could you add the photos I asked you to take showing that the ISR is way longer with 64x Averaging? – From the logic analyzer.

* + The thread profile is different because the hardware averaging calculates 4x, 16x or 64x amount of samples for the ADCin() function value, which occurs in the interrupt. This results in the interrupt execution time becoming larger.

**3.0 ANALYSIS AND DISCUSSION**

1. The ISR toggles PF2 three times. Is this debugging intrusive, nonintrusive or minimally intrusive? Justify your answer.  **Page 183**

— It is nonintrusive because it allows for using the oscilloscope/logic analyzer as debugging tools, and allows the hardware/software system to operate normally as if the debugger did not exist.

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? **Page 183**

— Printf statements should be used when the system is not real-time and the software execution timing is not critical. It is te

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?

- Lab lecture:

* + Shared global
  + Nonatomic access
  + At least one write

4) Define “minimally intrusive”. **Page 183**

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?

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

* + It is minimally intrusive because the time spent toggling during the interrupt is very less (less than 0.001%) of the total time spent in the main code execution.

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 is better than dumps because they are easier to read and easier to introduce.

Dumps are better than printf because they are minimally intrusive.

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?

The Following kinds of software code can cause critical sections

- nonatomic sequences (example read-write-read sequence and an interrupt occurs during this block of code)

- global variables being accessed and modified by multiple threads.

4) Define “minimally intrusive”.

* + A debugging instrument is minimally intrusive if it has a negligible effect on the total cycle time of the system being debugged.

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 compromises the real time nature of the system, since the hardware averaging itself takes some time which increases the amount of time spent in the interrupt