A) 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,
- Get an early start on Lab 3, by writing a line drawing function.

B) Hardware Design

None

C) Measurement Data

Prep part 2): answers to the five questions a – f.

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.

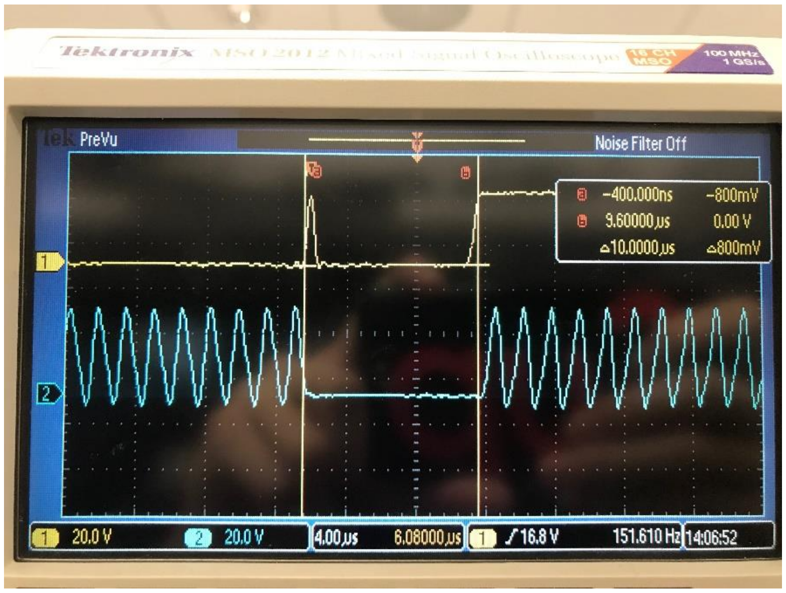
d) How would you have written the compiler to remove an instruction?
Here we are getting the memory address of PF1 twice, which is completely unnecessary.
I will write the compiler such that it produces the following Assembly code for the same C code.

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

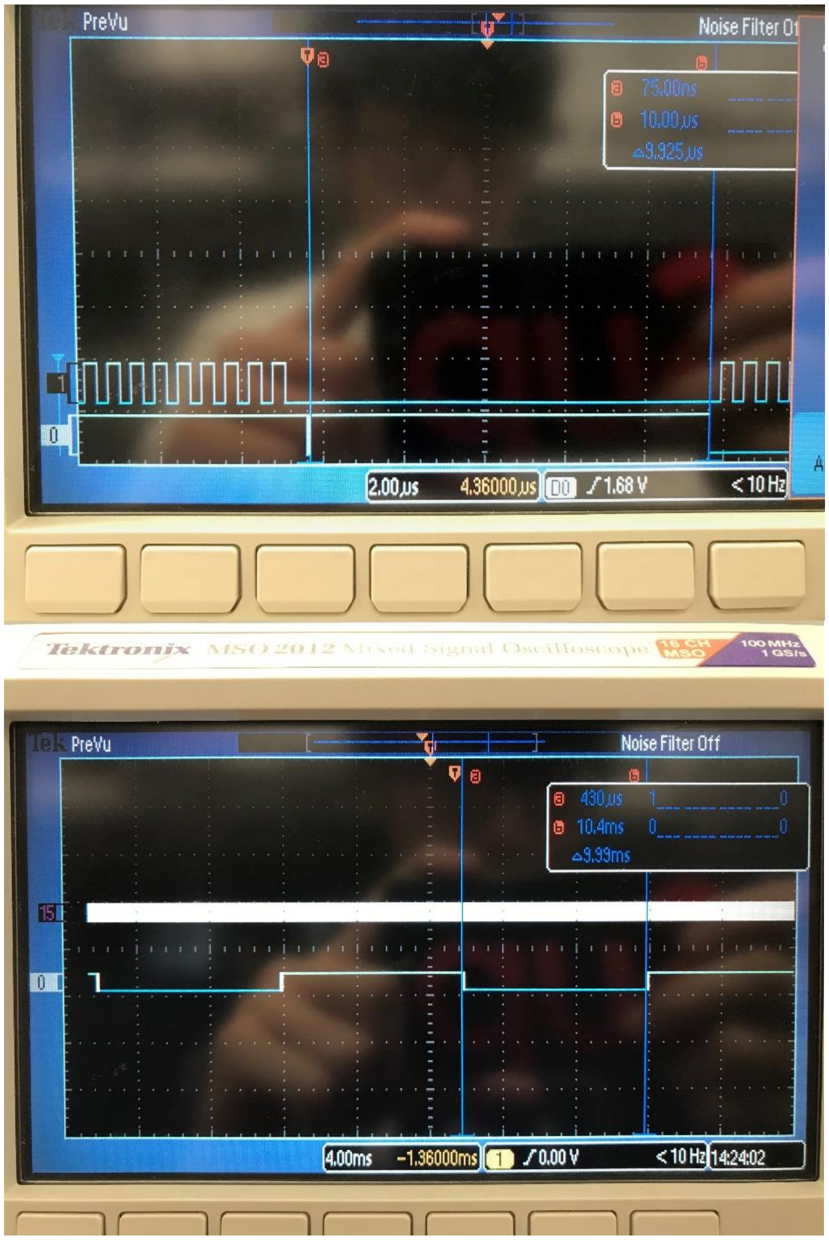
In fact, the Last line "`B      0x0000067E`" can be replaced by "`B      0x00000680`" to save even more time, because r0 does not chagne, and therefore the base location for Port F doesn't need to be fetched again.

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.

Procedure A:


Tektronix MSO2012 Mixed Signal Oscilloscope
100 MHz 1 GS/s
PreVu
Noise Filter Off
a −400.000ns −800mV
b 9.60000µs 0.00 V
Δ10.0000µs Δ800mV
1 20.0 V
2 20.0 V
4.00µs 6.08000µs
1 ∫ 16.8 V
151.610 Hz
14:06:52

Procedure B:



Total time = 9.925 us + 9.99 ms ~= 9.9999 ms
Percentage time in main: 9.99 ms / 9.9999 ms ~= 99.9%
Percentage time in ISR: 9.925 us/ 9.9999 ms ~= 0.1%

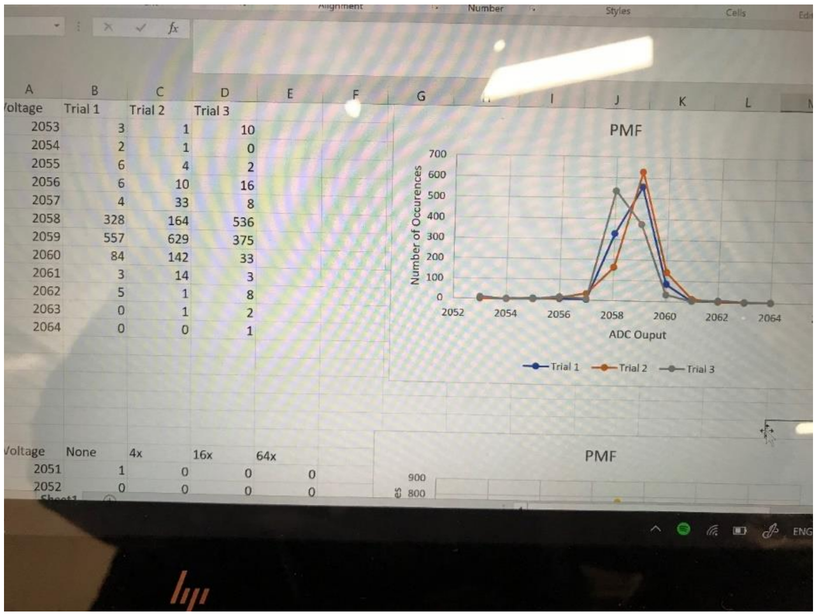
Procedure C:
PF2 is toggling incorrectly.
Steps that causes this to happen:

1. Main function toggling PF1, ISR happens before main function can store back the toggled value at
2. ISR toggles PF2 and stores back the toggled value of Port F, then it returns to main program
3. Main function continues and stores toggled value before ISR occurred
4. The newly stored value overwrites the Port F values previously by the ISR.

To prevent this from happening, we can 1. disable interrupts every time before we toggle change Port F in main, or 2. Make sure the operation on the shared resource is atomic.

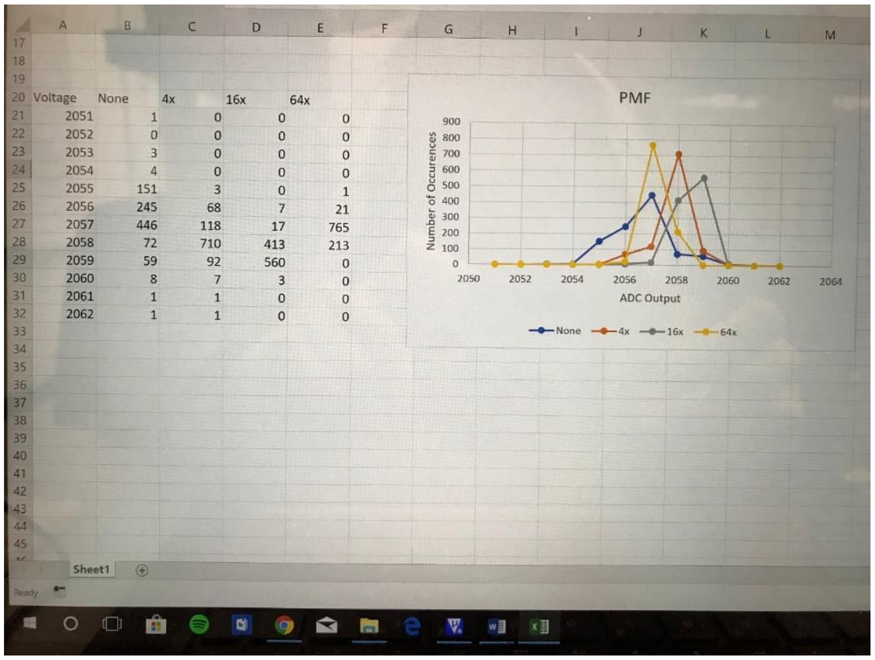
Procedure D:
time jitter with just the one sampling interrupt active: 6 ns
time jitter with three sampling interrupts active: 15 ns

Procedure E and F:



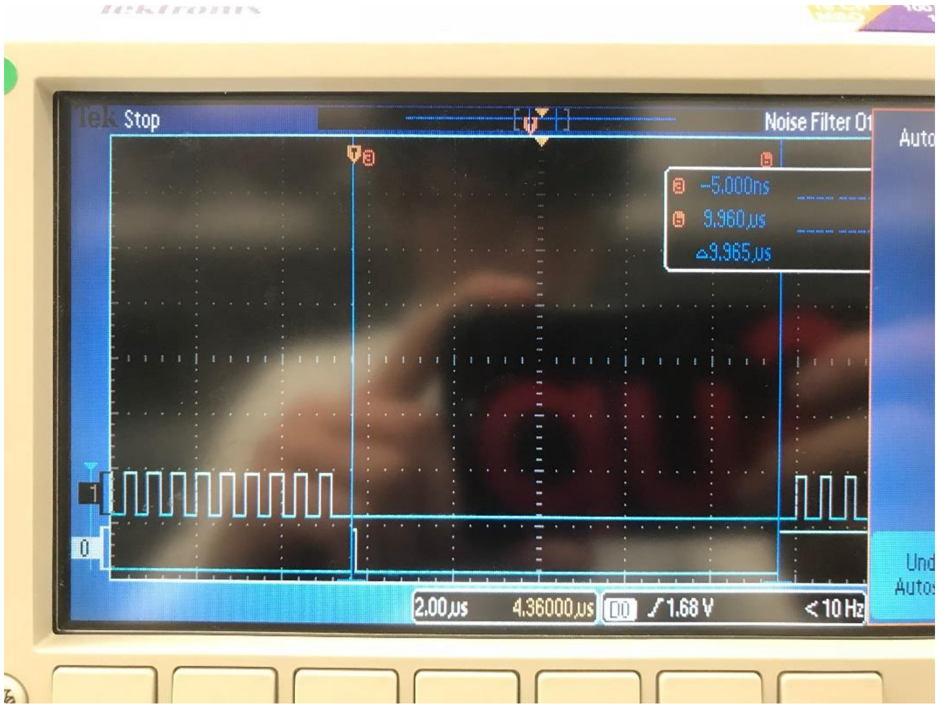
Voltage	Trial 1	Trial 2	Trial 3
2053	3	1	10
2054	2	1	0
2055	6	4	2
2056	6	10	16
2057	4	33	8
2058	328	164	536
2059	557	629	375
2060	84	142	33
2061	3	14	3
2062	5	1	8
2063	0	1	2
2064	0	0	1

Each PMF looks similar. Their values center around 2058 and 2059 and spread out in both directions.

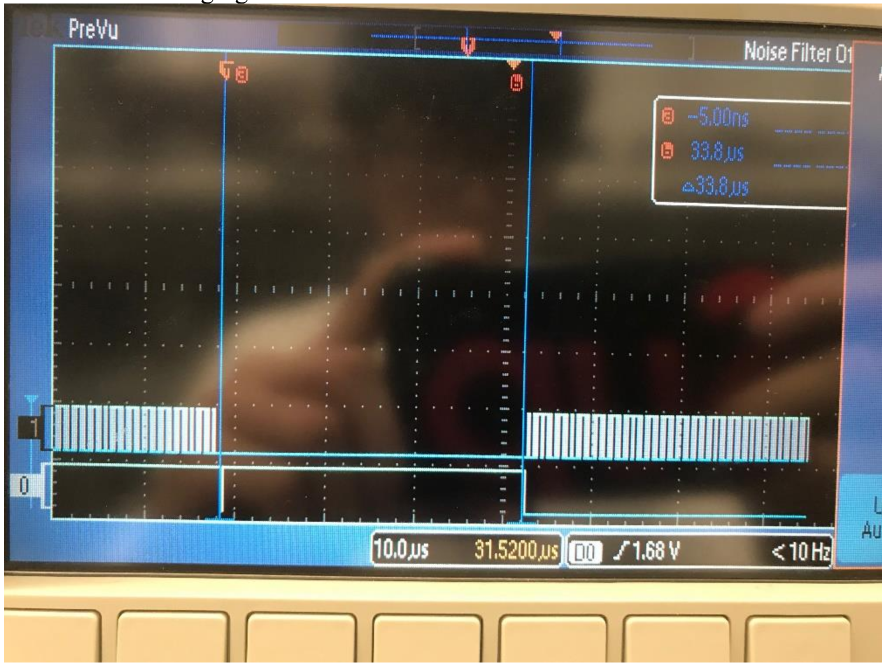


Voltage	None	4x	16x	64x
2051	1	0	0	0
2052	0	0	0	0
2053	3	0	0	0
2054	4	0	0	0
2055	151	3	0	1
2056	245	68	7	21
2057	446	118	17	765
2058	72	710	413	213
2059	59	92	560	0
2060	8	7	3	0
2061	1	1	0	0
2062	1	1	0	0

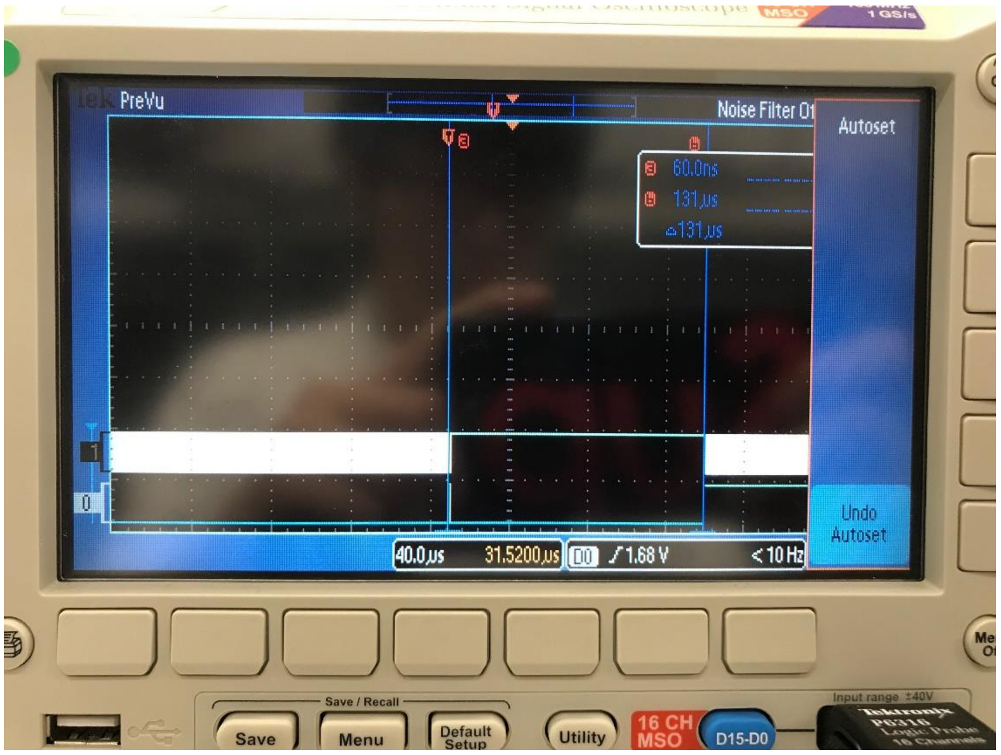
No hardware averaging: 9.97 us



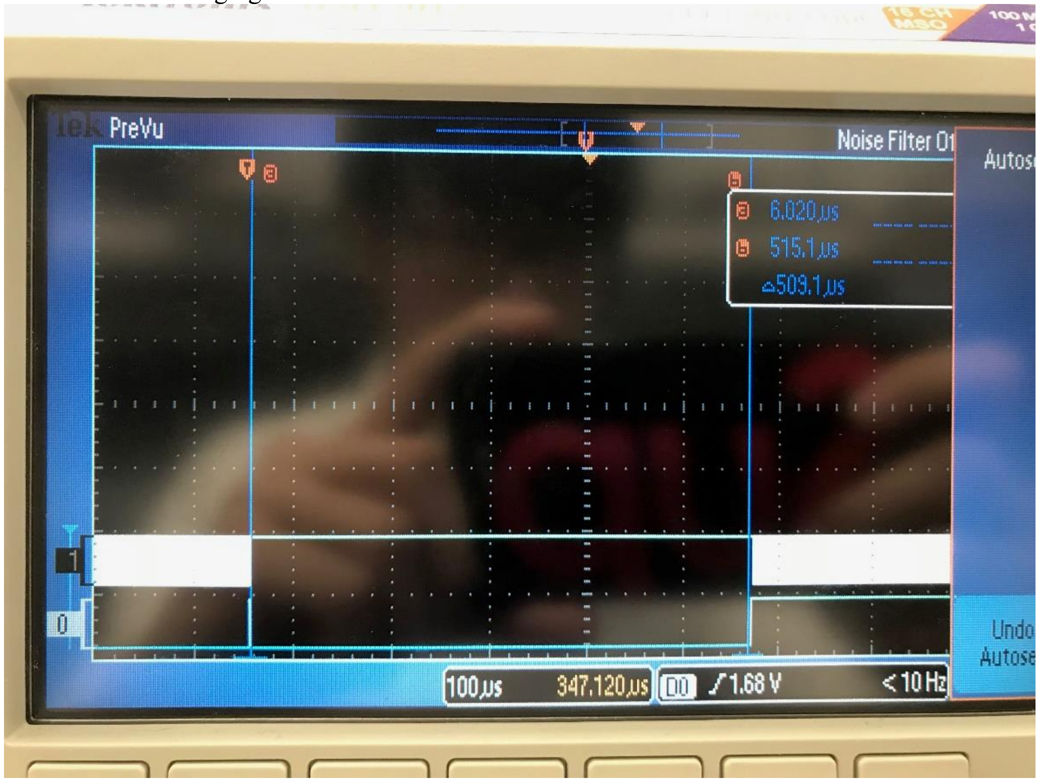
x4 hardware averaging: 33.8 us



x16 hardware averaging: 131 us



x64 hardware averaging: 509 us



Our data supports the CLT.

Downside: The more samples we collect for hardware averaging, the longer the ISR takes.

Analysis and Discussion:

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

Minimally intrusive

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?

Dumps better than printf:

a) Not a lightweight function
b) There is only so much you can print on the screen, while dumps can record a lot more information

Printf better than dumps:

a) Gets feedback immediately
b) Gives you pre-formatted data, while dumps are harder to read.

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?

Non-atomic access to a shared resource.

4) Define "minimally intrusive".

A debugging instrument is defined as minimally intrusive if it has a negligible effect on 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?

The more samples we collect for averaging, the longer the ISR takes.