ECE 445L Lab 2

Debugging, oscilloscope fundamentals, logic analyzer, dump profiles

This laboratory assignment accompanies the book, [*Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154*](https://www.amazon.com/Embedded-Systems-Real-Time-Interfacing-Microcontrollers/dp/1463590156), by Jonathan W. Valvano, copyright © 2021.

# Table of Contents

[Table of Contents 1](#_Toc455718004)

[Team Size 1](#_Toc356129989)

[Goals 2](#_Toc1251722427)

[Review 2](#_Toc1875690335)

[Starter Files 2](#_Toc1250679840)

[Required Hardware 2](#_Toc2135416522)

[Lab Overview 3](#_Toc1523508674)

[Preparation 3](#_Toc171870113)

[Preparation Questions 4](#_Toc88564269)

[Procedure 4](#_Toc1420946450)

[Deliverable 1 5](#_Toc1355592429)

[Using the Oscilloscope, Spectrum Analyzer, and Logic Analyzer 6](#_Toc733222961)

[Debug Dump.c Functions and Prove the ADC Sampling is Real Time 6](#_Toc904784683)

[Evaluate Critical Sections 6](#_Toc1242627215)

[ADC Noise Measurements Using the Central Limit Theorem 6](#_Toc809807715)

[Estimate the ADC Resolution 7](#_Toc1879590426)

[Lab Checkout 8](#_Toc1862270658)

[Lab Report 8](#_Toc1343067915)

[Deliverables 8](#_Toc1631567150)

[Analysis and Discussion Questions 8](#_Toc75601435)

[Extra Credit 9](#_Toc1696221511)

[Hints 10](#_Toc802690496)

# Team Size

The team size for this lab is **1**.

# Goals

In this lab we will introduce various debugging techniques. We will look at how to use the oscilloscope, spectrum analyzer, and logic analyzer, and then learn how to profile code using dumps to measure intrusiveness and noise.

You should understand the following concepts by the end of the lab:

* real-time systems
* time jitter
* critical sections
* bit banding and shared resources
* probability mass functions (PMF)
* central limit theorem (CLT)

# Review

* Valvano Section 2.4 on GPIO, Chapter 10 of data sheet
* Valvano Sections 3.9, 5.9 on debugging,
* Valvano Section 5.3 on critical sections,
* Valvano Section 6.2 on periodic timer interrupts, Chapter 11 of data sheet
* Valvano Section 8.5 on the ADC, Chapter 13 of data sheet
* Logic analyzer instructions.

# Starter Files

* Example projects from [ValvanoWareTM4C123v5](https://github.com/ECE445L/ECE445L-Examples):
  + PeriodicTimer1AInts\_4C123 (review periodic interrupts)
  + ADCSWTrigger\_4C123 (review periodic timer interrupts and busywait ADC)
  + Reference ECE445L Projects require Keil compiler version 5.
* Starter project:
  + Lab 2 template provided on the GH Classroom repo.

# Required Hardware

|  |  |  |  |
| --- | --- | --- | --- |
| Parts | Datasheet | Price | Source (**price source)** |
| EK-TM4C123GXL | [EK-TM4C123GXL datasheet](https://github.com/ECE445L/ECE445L-Lab1/blob/main/resources/TM4C_Datasheet.pdf) | $16.99 | **TI** |
| Sitronix ST7735 Color LCD | [ST7735 datasheet](https://www.displayfuture.com/Display/datasheet/controller/ST7735.pdf) | $19.95 | Adafruit |

# Lab Overview

In this lab we will develop debugging techniques to experience fundamental concepts of real time, critical sections, probability mass function (PMF), and the Central Limit Theorem (CLT). You should review real-time, time jitter, and critical sections from the book. Do an internet search of PMF and CLT. The object of this lab is to implement **Dump.c** and use it in subsequent labs to assist debugging.

# Preparation

Preparation is performed during or before the W/TH lab session.

1. Read the functions declared in **Dump.h** and defined in **Dump.c**.
2. Read the following information about debugging with the logic analyzer and oscilloscope:
   1. If you have access to a real logic analyzer and oscilloscope, you will use main programs **main1** and **main3** (which do not activate TExaS).
   2. If you do not have access to a real logic analyzer and oscilloscope, you will use main programs **main0**, **main2**, and **main4** (which will activate TExaS).
   3. Reading TIMER1\_TAR\_R will return the 32-bit current time in 12.5ns units. The timer counts down. To measure elapsed time, we read TIMER1\_TAR\_R at the start of the elapsed time measurement and read it again at the end of the elapsed time measurement. Next, we subtract the second measurement from the first. 12.5ns \* 232 is 53 seconds. So, this approach will be valid for measuring elapsed times less than 53 seconds. The time measurement resolution is 12.5 ns.

## Preparation Questions

|  |  |
| --- | --- |
| Assembly code generated by the compiler | C code written by the user from main1/2 (If applicable) |
| 0x00000D98 481F LDR r0,[pc,#124] ; @0x00000E18  0x00000D9A 6880 LDR r0,[r0,#0x08]  0x00000D9C F0800002 EOR r0,r0,#0x02  0x00000DA0 491D LDR r1,[pc,#116] ; @0x00000E18  0x00000DA2 6088 STR r0,[r1,#0x08] | while(RealTimeCount < 3000) {  PF1 ^= 0x02; |
| 0x00000DA4 4812 LDR r0,[pc,#72] ; @0x00000DF0  0x00000DA6 6800 LDR r0,[r0,#0x00]  0x00000DA8 491C LDR r1,[pc,#112] ; @0x00000E1C  0x00000DAA 4348 MULS r0,r1,r0  0x00000DAC 491C LDR r1,[pc,#112] ; @0x00000E20  0x00000DAE FBB0F0F1 UDIV r0,r0,r1  0x00000DB2 490F LDR r1,[pc,#60] ; @0x00000DF0  0x00000DB4 6008 STR r0,[r1,#0x00] | JitterVar = (JitterVar\*12345678)/1234567; |
| 0x00000DB6 4817 LDR r0,[pc,#92] ; @0x00000E14  0x00000DB8 6800 LDR r0,[r0,#0x00]  0x00000DBA F64031B8 MOVW r1,#0xBB8  0x00000DBE 4288 CMP r0,r1  0x00000DC0 D3EA BCC 0x00000D98 | } |
| 0x00000DF0 0014 DCW 0x0014  0x00000DF2 2000 DCW 0x2000  0x00000E14 0000 DCW 0x0000  0x00000E16 2000 DCW 0x2000  0x00000E18 5000 DCW 0x5000  0x00000E1A 4002 DCW 0x4002  0x00000E20 D687 DCW 0xD687  0x00000E22 0012 DCW 0x0012 |  |

Listing 2.1. Assembly Table.

1. What are the purposes of the **DCW** statements? More specifically, what do these three constants mean: 0x2000\_0014, 0x4002\_5000, and 0x0012\_D687?
2. Look at Section 3.3.1 (page 32) of the data sheet [CortexM4\_TRM\_r0p1.pdf](http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/CortexM4_TRM_r0p1.pdf) and find which instructions in the above while loop take more than 3 cycles to execute. Assume that P=3 for the BCC instruction because it must refill the pipeline if it branches.
3. This while loop toggles PF1. Assuming no interrupts occur, and that assembly instructions take 25ns (AKA two cycles at 80 MHz) to execute on average, estimate how long one iteration of the while loop would take to execute.

# Procedure

Procedure is performed during or before the W/TH lab session.

## Setup

1. Connect a constant analog voltage to an ADC input. Four possible ADC inputs are PD3, PD2, PE2, and PB5. One option is to create 1.65V using two 10k resistors as shown below on the left. Another option is to use a potentiometer as shown below on the right.

A diagram of a circuit

Description automatically generated

Figure 2.1. Possible hardware connection to create an analog input.

1. Use an oscilloscope to view the constant voltage by attaching the positive lead to AIN, and the negative lead to GND. Set the coupling mode of the oscilloscope to DC mode in order to see the magnitude of the signal, then change the coupling mode to AC in order to see the noise.
2. **(TExaS users ONLY)** If using TExaS (and therefore mains main/main0, main2, and main4), edit the parameter in Listing 2.2 for the call to TExaS\_Init to specify your choice of channel.

// Parameters that can be passed into

// TExaS\_Init based on HW configuration.  
  
// TExaS.h  
enum TExaSmode{  
 SCOPE, // PD3  
 SCOPE\_PD2,  
 SCOPE\_PE2,  
 SCOPE\_PB5,  
 LOGICANALYZERA,  
 LOGICANALYZERB,  
 LOGICANALYZERC,  
 LOGICANALYZERE,  
 LOGICANALYZERF,  
 NONE  
};

Listing 1.2. TExaSmode enum specifying scope usage.

## Deliverable 1

Draw the electrical circuit you created in the lab to generate the analog input signal.

For most students, this is simply which of the two circuits you made during setup (Figure 2.1).

## Deliverable 2

Use the oscilloscope to visualize and measure the noise of the analog input of your circuit. Take a picture of the scope trace (screenshot or USB capture or phone picture) and add to the lab report. Specifically:

The image should show 5 to 10ms of waveform data.

The image should show the magnitude of the noise in either AC RMS or DC Peak-to-Peak. This can be done via the cursors or adding an dynamic measurement.

An example of what this looks like is below. You are expected to learn how to use these instruments in this class, so please ask your TA for a demonstration in the lab if you are unfamiliar with them.

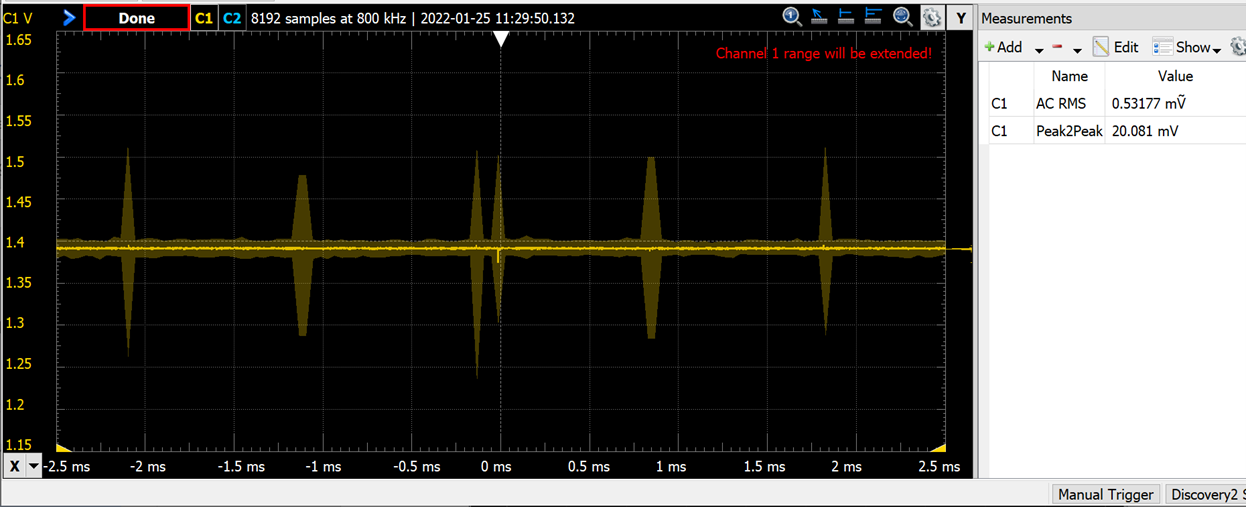


Figure 2.2. Analog voltage versus time measured with a real oscilloscope.

## Deliverable 3 (2pts Extra Credit)

Use the spectrum analyzer mode of the oscilloscope to measure amplitude vs frequency of the analog input of your circuit. This is done by pressing the math button, then selecting FFT in the on screen display. Take a picture of the scope trace (screenshot or USB capture or phone picture) and add to the lab report.

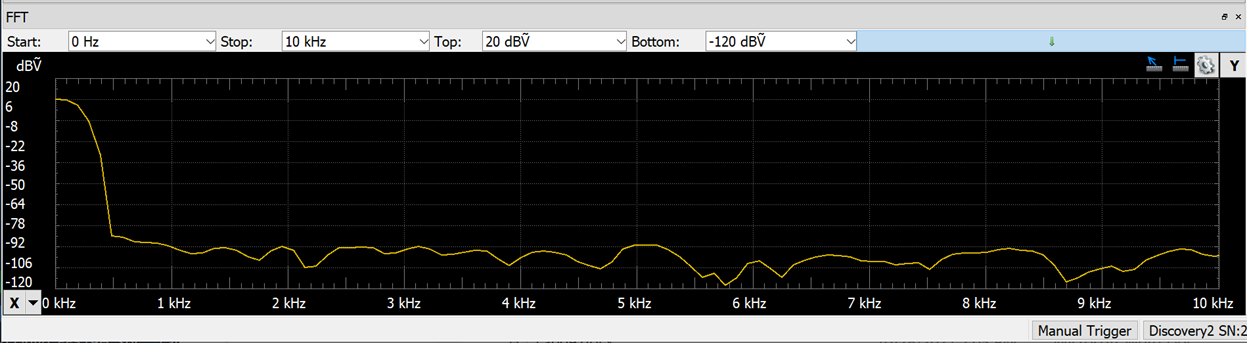


Figure 2.3. Analog voltage versus frequency measured with a real spectrum analyzer.

## Deliverable 4

Run main3 and observe PF3 (Timer2A ISR), PF2 (Timer0A ISR) and PF1 (main). Be prepared to put some logic analyzer captures into the lab report as you perform these tasks.

Connect the oscilloscope to PF2, and examine the function realTimeSampling() which is driving PF2, and is called every period by Timer0A.

* Measure P0, the interrupt period for the Timer0A (should be 1/125Hz).
* Measure T0, the time to complete the Timer0A ISR (should be about 10us with ADC0\_SAC\_R=0).
* Calculate the Timer0A ISR utilization percentage. This is T0/P0.

Connect the oscilloscope to PF3, and examine the function realTimeTask() which is driving PF3 and is called every period by Timer2A.

* Measure P2, the interrupt period for the Timer2A (should be 1/1024Hz).
* Measure T2, the time to complete the Timer2A ISR (should be about 1us).
* Calculate the Timer2A ISR utilization percentage. This is T2/P2.

Connect the oscilloscope to PF1 and examine the main loop which is driving PF1.

* Using the prior values found, calculate the total utilization percentage of the program. This is about 1-(T0/P0)-(T2/P2).

Be sure to include the three captures in your lab report, in addition to the individual and total utilization figures.

A screenshot of a computer

Description automatically generated  
Figure 4. Zoomed in view of the PF1 PF2 PF3. Note that the main program does not run (As evidenced by PF1 no longer toggling) while the Timer0A ISR is running. Also note that the time to execute the Timer0A ISR is about 10us (10us as this is the time between the double and single toggles of PF2; Most of this 10us occurs converting the ADC which is running with ADC0\_SAC\_R=0).

A screenshot of a computer

Description automatically generated  
Figure 4b. Zoomed in view of the PF1 PF2 PF3 recording to see a) the main program does not run while the Timer2A ISR is running and b) the time to execute the Timer2A ISR is about 1us.

A screenshot of a computer

Description automatically generatedFigure 4c. Zoomed out view of the PF1 PF2 PF3 recording to see a) the Timer0A runs at 125 Hz, b) Timer2A runs at 1024 Hz, and c) most of the processor time is allocated to running the main program.

## Deliverable 5

Measure the time jitter with just Timer2A (main1). Then measure the time jitter with two ISRs (main3). Jitter is measured by simply running the main program, then examining the screen. Using this data consider the following questions:

* What was the jitter with just Timer2A? Explain what caused the small but non-zero jitter.
* Why would you classify Timer2A by itself as real time?
* What was the jitter with both timers running for Timer2A? As SAC (Sample Average Count) changes for the task in Timer0A changes, how does Timer2A’s jitter change?
* Why would you classify Timer0A as real time, but Timer2A as no longer real time?

Note: Multiple timers do not imply that all ISR are no longer real time. If we were to use timer-triggered ADC sampling as in lab 9, even with hardware averaging, all ISRs will be real time.

## Deliverable 6

Now we will perform ADC Noise Measurements Using the Central Limit Theorem/

To apply the Central Limit Theorem, we must assume the noise is random, the noise in each sample is independent from the noise in the other samples, and the noise has zero mean. Look up the ADC Sample Averaging Control (ADC0\_SAC\_R) register in Chapter 13 of the data sheet. The Central Limit Theorem (CLT) states: as the number of samples increases, the calculated average (your data) will approach the theoretical mean (true signal). The CLT also states that regardless of the original probability density function (PDF) of the noise, the PDF of the averaged signal will become Gaussian.

Connect the constant voltage to the ADC input and run main3 or main4. Since the input voltage is constant, the expected result would be all ADC data to be the same. Noise causes variability. Observe the PMF of the noise as the program varies ADC0\_SAC\_R from 0 to 6. 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. You are allowed to adjust DUMPBUFSIZE to vary the number of points collected. If you compare two PMFs with the same SAC value, you will not get the same result because the noise is not stationary.

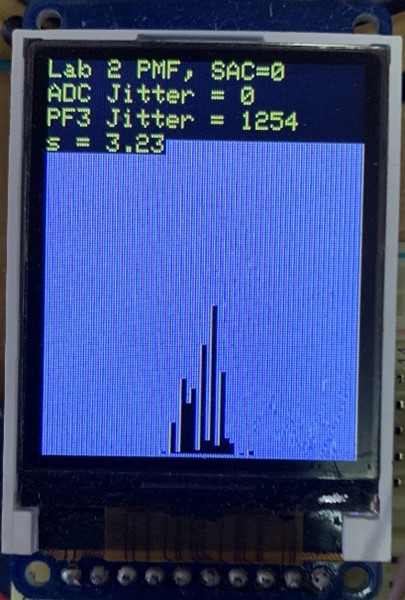


Figure 2.5. Photo of main3 output with a constant voltage applied to the analog input (SAC=0).

Take four photos of the LCD screen PMF, like Figure 2.5 above, for hardware averaging values of none, 4x, 16x, and 64x. Note: between measurements keep constant both the ADC sampling rate, and the value of DUMPBUFSIZE. Since noise can vary, consider going around the lab room, and look at the data from other groups for fun.

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: CLT.

## Deliverable 7 (2pts Extra Credit)

With the data collected for deliverable 6, we can estimate the resolution of our ADC. One simple estimate of the ADC resolution is standard deviation. Place a constant input on the ADC, sample the data multiple times and then calculate the standard deviation of the results. The data collected in Figure 2.6 shows the standard deviation of this data is about 3.23 as calculated by main3.

For this number to have meaning however, we must convert it into a real unit. Since the range of values is 0 to 4095, and this range is meant to linearly represent 0V to 3.3V we can calculate that 3.23 is equivalent to 3.23\*3.3/4096 ≈ 2.6mV. So, for SAC=0, we claim the ADC resolution is about 2.6mV. This is because if the input were increased by only 0.5mV, the PMF distributions are not statistically different.

For this data shown below at SAC=6, we claim the ADC resolution is about 1mV. ECE445L does not expect you to collect data like Figure 2.6. merely to perform the conversion as shown in the example.

In your report, using the standard deviation, estimate the ADC resolution for SAC=4. (16-point averaging).

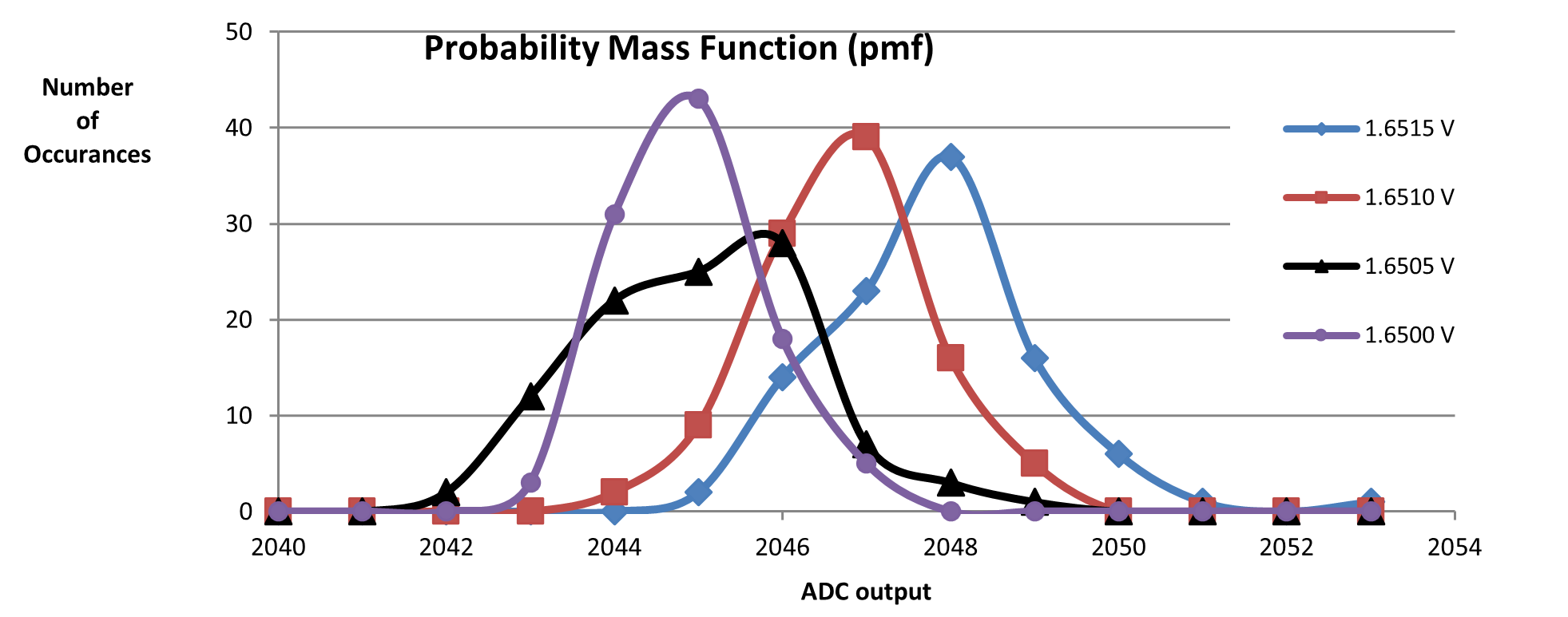


Figure 2.6. Probability mass function measured on the TM4C123 ADC with SAC=6 (64-point averaging).

## Deliverable 8

Now we will Evaluate Critical Sections. Consider that all three threads (main, Timer0A, Timer2A) perform a read-modify-write access to Port F. Here, it is because of bit-specific addressing, these accesses are not critical.

Change the accesses to use GPIO\_PORTF\_DATA\_R instead of PF1, PF2, and PF3, thus creating one or more critical sections.

Critical sections create weird and unexpected behavior. Use any debugging technique to observe one instance of a critical section. Place the observation into your lab manual and explain the mistake the critical section created.

Hint: One approach is to follow program execution to see erroneous values written to Port F in the debugger. In this approach take multiple screenshots in the debugger to show incorrect behavior. Another approach is to exam Port F using the oscilloscope and contrast the output to that of deliverable 4.

## Deliverable 9 (10pts Extra Credit)

## Perform an empirical study to evaluate four implementations on the Cortex M4. Two implements use fixed-point, two use floating-point, two are written in assembly, and two are written in C. For each implementation measure the total execution time. Make conclusions about implementing arithmetic operations on the Cortex M4. We recommend you use the example project with **Float** in its name to do this part.

|  |  |
| --- | --- |
| // version 1: C floating point  // run with compiler options selected  // for floating-point hardware  volatile float T; // temperature in C  volatile uint32\_t N; // 12-bit ADC value  void Test1(void) {  for(N=0; N<4096; N++){  T = 10.0 + 0.009768 \* N;  }  } | // version 2: C fixed-point  volatile uint32\_t T; // temperature in 0.01 C  volatile uint32\_t N; // 12-bit ADC value  void Test2(void){  for(N=0; N<4096; N++){  T = 1000+ (125\*N+64)>>7;  }  } |
| ; Version 3 assembly floating point  ; run with floating-point hardware active  AREA DATA, ALIGN=2  T SPACE 4  N SPACE 4  AREA |.text|, CODE, READONLY, ALIGN=2  THUMB  Test3  MOV R0,#0  LDR R1,=N ;pointer to N  LDR R2,=T ;pointer to T  VLDR.F32 S1,=0.009768  VLDR.F32 S2,=10  loop3 STR R0,[R1] ; N is volatile  VMOV.F32 S0,R0  VCVT.F32.U32 S0,S0 ; S0 has N  VMUL.F32 S0,S0,S1 ; N\*0.09768  VADD.F32 S0,S0,S2 ; 10+N\*0.0968  VSTR.F32 S0,[R2] ; T=10+N\*0.0968  ADD R0,R0,#1  CMP R0,#4096  BNE loop3  BX LR | ; version 4, assembly fixed point  AREA DATA, ALIGN=2  T SPACE 4  N SPACE 4  AREA |.text|, CODE, READONLY, ALIGN=2  THUMB  Test4 PUSH {R4,R5,R6,LR}  MOV R0,#0  LDR R1,=N ;pointer to N  LDR R2,=T ;pointer to T  MOV R3,#125  MOV R4,#64  MOV R5,#1000  loop4 STR R0,[R1] ; N is volatile  MUL R6,R0,R3 ; N\*125  ADD R6,R6,R4 ; N\*125+64  LSR R6,R6,#7 ; (N\*125+64)/128  ADD R6,R6,R5 ; 1000+(N\*125+64)/128  STR R6,[R2] ; T = 1000+(N\*125+64)/128  ADD R0,R0,#1  CMP R0,#4096  BNE loop4  POP {R4,R5,R6,PC} |

# Lab Checkout

The lab checkout is performed during the M/T lab session.

You should be able to demonstrate:

* Your understanding of the logic analyzer and scope features listed.
* Any of the deliverables: how the data was collected and what it means.

# Lab Report

The lab report shall be submitted by the Friday after the second lab section.

You should complete the Lab02Report.docx file with your data and answers then submit the completed file to canvas

# Appendix A (Alternate Deliverables)

## Deliverable 2 (Alternate, TExaS Version)

You are provided the option to use the TM4C to emulate its own oscilloscope, spectrum analyzer, and logic analyzer through TExaS. Please note that this comes with significant constraints on functionality and resolution. Provided are two tutorials on using TExaS:

* [TExaS Oscilloscope and Spectrum Analyzer](https://github.com/ECE445L/ECE445L-Lab2/blob/5e63b1199a8914ed8075cfd5d6d896a65eee7e4d/README.md" \l ":~:text=You are expected,TExaS Logic Analyzer)
* TExaS Logic Analyzer

If using TExaS, an 8-bit analog signal on PD3 is sampled at 10 kHz and sent to the PC for plotting. To use the scope, connect the analog input to PD3. Be careful to limit the voltage between 0 and 3.3V, because PD3 is an unbuffered TM4C123 analog input. Run main0, which activates TExaS\_Init(SCOPE).

A screen shot of a graph

Description automatically generated

Figure 2.2b. Analog voltage versus time measured with the TExaS oscilloscope.

## Deliverable 3 (Not graded, Alternate, TExaS Version)

If using TExaS, follow the instructions in the [TExaS Oscilloscope and Spectrum Analyzer](https://youtu.be/fqm0zkr0_QA) video to select the spectrum analyzer from the view menu.

A screen shot of a graph

Description automatically generated

Figure 2.3b. Analog voltage versus frequency measured with the TExaS spectrum analyzer.

## Deliverable 4 (Alternate, TExaS Version)

If using TExaS, the TExaS logic analyzer sends 7-bit data at 10 kHz to the PC for plotting. Run main4, which selects the logic analyzer on Port F. Notice the call to TExaS\_Init(LOGICANALYZERF). You do not have to make any hardware connections to utilize the logic analyzer. Since the priority of the TExaS interrupt is 5 (lower priority than the two ISRs in Lab 2), the triple toggles will always be seen as a single toggle. Observe PF3 (Timer2A ISR), PF2 (Timer0A ISR) and PF1 (main).

Measure P0, the interrupt period for the Timer0A (should be 1/125Hz). The most accurate measurement of P0 is achieved by deriving it from F2, the frequency of channel 2 (PF2). P0 = 0.5/F2 (0.5/62.5 Hz=8ms in this figure).

Assume T0, the time to complete the Timer0A ISR, is about 10us with ADC0\_SAC\_R=0. The percentage time in Timer0A ISR is T0/P0.

Measure P2, the interrupt period for the Timer2A (should be 1/1024Hz). Similarly, the most accurate measurement of P2 is achieved by deriving it from F3, the frequency of channel 3 (PF3). P2 = 0.5/F3 (0.5/511.6 Hz=0.977ms in this figure). The 0.5 in this equation results from the fact that each ISR toggles the output pin.

Assume T2, the time to complete the Timer2A ISR, is about 1us. The percentage time in Timer2A ISR is T2/P2. The percentage time in the main program is therefore about 1 T0/P0-T2/P2. Notice the 10 kHz sampling rate of the TExaS logic analyzer cannot correctly capture the behavior of PF1.

Finally, Debug Dump.c Functions and Prove the ADC Sampling is Real Time

A screenshot of a computer

Description automatically generatedFigure 4d. Zoomed out view of the PF1 PF2 PF3 recording using the TExaS logic analyzer to see a) the Timer0A runs at 125 Hz, b) Timer2A runs at 1024 Hz, and c) most of the processor time is allocated to running the main program.

## Deliverable 5 (Alternate, TExaS Version)

Perform Deliverable 5, except using Main 2 and Main 4.