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.

## Goals

• 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

## 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:

• **ADCSWTrigger\_4C123**

• **PeriodicTimer1AInts\_4C123**

• **Lab2** project (new from web page)

• Dump.h, Dump.c, TExaS.h, TExaS.c (new from web page)

TExaSdisplay version 2.5 <https://www.dropbox.com/s/svohgjizgbzv4sj/TExaSdisplay.exe?dl=1>

## Team Size: 2

## Background

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.

Assume you have a periodic task that should run every *Δt*. Measure the time the actual task runs as *ti*. Calculate *δi* = (*ti* – *ti-1*) as the actual time differences between running the task. If the sampling were perfect, *δi* would equal *Δt* for all *i*. We define **jitter** as the maximum *δi* minus minimum *δi*. If the sampling were perfect, jitter would be 0. Note: jitter is calculated in **main3** or **main4** for the ADC sampling. We did not ask it in Lab 2, but we could define **sampling accuracy** as maximum | *Δt* - *δi* | for all *i*.

## Preparation

NOTE: do this before lab starts, show the answers and source code to your TA at the start of Lab

1. Download the **Lab2.zip** and the **Dump.h, Dump.c, TExaS.h, TExaS.c** files from the class web site. Unzip the **Lab2.zip** file and place the **Lab2** project along the other projects. Place the **Dump.h, Dump.c, TExaS.h, TExaS.c** files in the **inc** folder. The **Lab2** project should compile.

2. Write the C code to implement the functions in the **Dump.c** file. You may change the look and feel of the debugging instruments, but make sure the Lab 2 project still compiles. If you have access to a real logic analyzer, you will use main programs **main1** and **main3** (which do not activate TExaS). If you do not have access to a real logic analyzer, you will use main programs **main0**, **main2**, and **main4** (which will activate TExaS). 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.

You are afforded maximum flexibility in Lab 2 in exactly what the functions do. You will be asked to use these functions to measure jitter in subsequent labs. One possible solution is as follows. **Dump\_Init** initializes the timer1 and clears the counter Dump\_Num. **Dump\_Capture** stores the data and the time (TIMER1A\_TAR\_R) into the buffers if room, increments Dump\_Num if needed. **Jitter\_Init** initializes two globals MaxJitter and MinJitter, and specifies a global flag that the first call has not happened.   
**Jitter\_Measure** looks at the flag. If it is the first call, simply measure the current time (TIMER1A\_TAR\_R); on subsequent calls measures the current time  (TIMER1A\_TAR\_R)  and calculate time difference from now to the previous call, update MaxJitter and MinJitter. **Jitter\_Get** returns MaxJitter minus MinJitter.

3. The microcontroller is executing at 80 MHz. The following shows a small section of the C code and resulting assembly code created by the compiler for the **while** loop in **main1** and **main2**.

|  |  |
| --- | --- |
| **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]**  **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]**  **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** | **while(RealTimeCount < 3000){**  **PF1 ^= 0x02;**    **YY = (YY\*12345678)/1234567;**    **}** |

*This assembly code was obtained by observing the assembly listing in the debugger. You may see different assembly on your machine because of differences in the compiler version or optimization settings. You are allowed to solve the preparation with either this assembly or the assembly you see on your computer.*

**Answer the following questions:**

1. What are the purposes of these **DCW** statements? More specifically, what do these three constants mean: 0x20000014, 0x40025000, and 0x0012D687?
2. Look at Section 3.3.1 (page 32) of the data sheet [CortexM4\_TRM\_r0p1.pdf](http://users.ece.utexas.edu/%7Evalvano/EE345L/Labs/Fall2011/CortexM4_TRM_r0p1.pdf) and find which instructions in the above while loop take more than 3 cycles to execute. Assume P=3 for the **BCC** instruction because it must refill the pipeline if it branches.

c) This **while** loop toggles PF1. Neglect interrupts for this preparation question. Assuming assembly instructions take about 25 ns to execute, estimate how fast the above **while** loop would execute.

## Procedure (do this during lab)

**1.** Connect a constant analog voltage to an ADC input on PD3, PD2, PE2 or PB5. Edit the parameter for the call to **ADC0\_InitSWTriggerSeq3** to specify your choice of channel. One option is to use a potentiometer, like Lab 8 in EE319K, Figure 2.1. Another option is to create 1.65V using two 10k resistors.

Schematic

Description automatically generated

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

***Deliverable 1***: Draw the electrical circuit you used to create the analog input.

*Note: Do either 2a or 2b, but not both.*

**2a. Learn how to use an oscilloscope if you have access to one**

You are expected to learn how to use an oscilloscope in this class, so, please ask your TA for a demonstration if you are unfamiliar with the features of the scopes we have in lab. You should:

1) be able to adjust the time base and voltage scales;

2) know how to set/adjust the trigger;

3) understand AC/DC mode;

4) be able to measure a frequency spectrum;

5) understand the resistive and capacitive load of the scope probe;

6) measure time delay using the time cursors;

7) measure voltage amplitude using the voltage cursors; and

8) be able to save waveforms to USB flash drive for printout later.

Line trigger mode is very useful for identifying the presence of 60 Hz AC-coupled noise.

*Graphical user interface

Description automatically generated*

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

***Deliverable 2a***: Use an analog scope to measure the analog input of your circuit. Use the scope to measure noise of the signal. You may measure AC RMS or peak-to-peak. Place a picture of the scope trace like Figure 2.2a into your lab manual, either a photo or digital downloaded image.

**2b. Learn how to use the TExaS oscilloscope if you do not have access to a real scope**

Those of you who cannot come to campus to use the test equipment in lab can complete Labs using the **TExaS** scope, logic analyzer, and spectrum analyzer. 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);.** In particular, you should: 1) be able to adjust the time base (F6, F7), knowing the sampling rate is fixed at 10 kHz and only the graph is scaled; 2) know how to adjust the trigger (Up, Down); 3) understand the meaning of the numerical measurements displayed; 4) be able to take screen shots of the waveforms like Figures 2.2b and 2.3b. With an 8-bit resolution the TExaS results will look almost perfect (just DC and no AC).

**Chart

Description automatically generated**

*Figure 2.2b. Analog voltage versus time measured with the TExaS oscilloscope. Watch a YouTube video of this process* [*https://youtu.be/fqm0zkr0\_QA*](https://youtu.be/fqm0zkr0_QA)

***Deliverable 2b***: Use the TExaS analog scope to measure the analog input of your circuit. The peak-to-peak voltage measurement is an estimate of your signal. Place a screen shot of the scope trace like Figure 2.2b into your lab manual.

*Note: Do either 3a or 3b, but not both.*

**3a. Learn how to use a spectrum analyzer if you have access to one (optional)**

A screenshot of a computer

Description automatically generated with medium confidence

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

***Deliverable3a***: Use an analog scope to measure amplitude versus frequency of the analog input of your circuit. Adjust the frequency scale to visualize the noise. Place a picture of the spectrum analyzer trace like Figure 2.3a into your lab manual, either a photo or digital downloaded image.

**3b. Learn how to use the TExaS spectrum analyzer if you do not have access to a real scope (optional)**

Connect the analog input to PD3. Run **main0**, which activates **TExaS\_Init(SCOPE);.** Within **TExaSdisplay** you can measure a frequency spectrum (**View->Oscilloscope, View->SpectrumAnalyzer**);

**A picture containing scatter chart

Description automatically generated**

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

***Deliverable3b***: Use an analog scope to measure amplitude versus frequency of the analog input of your circuit. Adjust the frequency scale to best visualize the noise. Place a screen shot of the spectrum analyzer trace like Figure 2.3b into your lab manual.

*Note: Do either 4a or 4b, but not both.*

**4a. Learn how to use a logic analyzer if you have access to one**

You are expected to learn how to use a logic analyzer in this class, so, please ask your TA for a demonstration if you are unfamiliar with the features of the logic analyzers. Run **main3** and observe PF3 (Timer2A ISR), PF2 (Timer0A ISR) and PF1 (main). 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). The percentage time in Timer0A ISR is T0/P0. 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, depending on your **Jitter\_Measure**). The percentage time in Timer2A ISR is T2/P2. The percentage time in the main program is therefore about 1-T0/P0-T2/P2.

Graphical user interface, application

Description automatically generated

*Figure 2.4a. Zoomed in view of the PF1 PF2 PF3 recording to see a) the main program does not run while the Timer0A ISR is running and b) the time to execute the Timer0A ISR is about 10us (most of this 10us occurs converting the ADC) This recording was taken with ADC0\_SAC\_R=0.*

Graphical user interface, application

Description automatically generated

*Figure 2.5a. 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.*

Graphical user interface, application

Description automatically generated

*Figure 2.6a. 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 4a***: Use the logic analyzer to measure the debugging profiles like Figures 2.4a and 2.6a and use the logic analyzer to estimate percentage of time running in the main versus running in the two ISRs. Place pictures of the scope traces (photo or digital download) into your lab manual.

**4b. Learn how to use the TExaS logic analyzer if you do not have access to a real logic analyzer**

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). Similar, 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.

Timeline

Description automatically generated with low confidence

*Figure 2.4b. 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. YouTube video showing logic analyzer on Lab 2* [*https://youtu.be/AVGbFRlGiXA*](https://youtu.be/AVGbFRlGiXA)

***Deliverable 4b***: Use the TExaS logic analyzer to measure the debugging profile like Figure 2.4.b and use the logic analyzer to measure the actual ISR rates. Estimate the percentage time in each of the three threads. Place a screen shot of the logic analyzer trace into your lab manual.

**5. Debug your Dump.c functions and prove the ADC sampling is real time.**

***Deliverable 5***: Measure the time jitter with just Timer2A (**main1** or **main2**). Explain what caused the small but non-zero jitter. Why would you classify Timer2A by itself as real time? Measure the time jitter with two ISRs (**main3** or **main4**). Explain why Timer2A has a time jitter proportional to 2sac. Explain why the Timer0A jitter is close to zero. Why would you classify Timer0A as real time, but Timer2A is no longer real time?

*Note: when we get to Lab 9, we will use timer-triggered ADC sampling, so that even with hardware averaging, all ISRs will be real time.*

**6. Study critical sections.**

All three threads perform a read-modify-write access to Port F. Because of bit-specific addressing, these accesses are not critical. Change the accesses to use **GPIO\_PORTF\_DATA\_R** instead of **PF1 PF2 PF3,** creating one or more critical sections. Critical sections create weird and unexpected behavior.

***Deliverable 6***: 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.

**7. 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 the Chapter 13 of the data sheet. The Central Limit Theorem (CLT) states as the number of samples increase, 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 the 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.*

A picture containing text, computer, monitor, desk

Description automatically generated

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

***Deliverable 7***: Take four photos of the LCD screen PMF, like Figure 2.5, for hardware averaging of none, 4x, 16x, and 64x. In each case the sampling rate is fixed and there are DUMPBUFSIZE 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.*

**Fun activity.** Noise can vary, so before you generalize from the data you collected in this lab, go around the lab room, and look at the data from other groups.

**8. Estimate the ADC resolution.**

One simple estimate of the ADC resolution is standard deviation. Place a constant input the ADC, sample the data multiple times and then calculate the standard deviation of the results. The data collected in Figure 2.5 shows the standard deviation of this data is about 3.23 samples. 3.23 samples are equivalent to 3.23\*3.3/4096 ≈ 2.6mV. So, for SAC=0, we claim the ADC resolution is about 2.6mV.

The data in Figure 2.6 were collected with SAC=6. Conversely if the input were increased by only 0.5mV, the PMF distributions are not statistically different. For this data at SAC =6, we claim the ADC resolution is about 1mV. EE445L does not expect you collect data like Figure 2.6.

*Figure 2.6. Probability mass function measured on the TM4C123 ADC with 64-point averaging.*

***Deliverable 8***: Estimate your ADC resolution with SAC=4 (16-point averaging)*.*

## Deliverables (exact components of the lab report)

A) Objectives (1/2 page maximum). Simply repeat the items shown in the **Goals** section

B) Hardware Design (Deliverable 1)

C) Software Design (Dump.c and Dump.h)

D) Measurement Data (Deliverables 2,4-8) (3 is optional)

E) Analysis and Discussion (give short 1 or two sentence answers to these questions)

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

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 what ways are printf statements better than dumps? In what ways are dumps better than printf statements?

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?

4) Define “minimally intrusive”.

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?

## Checkout

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.