Procedure

The basic approach to this lab will be to debug each module separately. After each module is debugged, you will combine them one at a time. For example: 1) just the ADC; 2) ADC and LCD; and 3) ADC, LCD and SysTick.

Part a) Decide which port pin you will use for the ADC input. If you plan to do the game in Lab 15, we suggest you place the ADC on PE2, which is ADC channel 1. However, the graders will allow you to use channel 0 (PE3), 1 (PE2), or 8 (PE5), as listed in Table 14.1. The game in Lab 15 requires you to use channel 1 (PE2) to sample a potentiometer in the similar way as this lab. Therefore, we recommend PE2/Ain1.

ADC Distance Measuring Pot.	AIN0 (PE3)	AIN1 (PE2)	AIN8 (PE5)

Table 14.1. Possible ports to interface the ADC inputs. The highlighted column is the recommended pin.

Part b) Write the ADC software that initializes and samples the ADC. This collection of functions is called a device driver. Even though there are only two functions, we will group them together into one collection because it will make it easier to reuse these functions in the next lab. Notice there is an ADC.h file containing the prototypes for the public functions and an ADC.c file containing the implementations. For this part, you are to write **ADC0_Init()** and **ADC0_In**. We suggest you look at the ADC code in section 14.2 and convert it to sample the pin you selected in Part a). You should debug this in the simulator, using a main program like the following

```
unsigned long ADCdata;
int main(void){
  TExaS_Init(ADC0_AIN1_PIN_PE2, SSI0_Real_Nokia5110_NoScope);
  ADC0_Init(); // initialize ADC0, sequencer 3, software start
  EnableInterrupts();
  while(1){
    ADCdata = ADC0_In(); // software start, read ADC, return 12-bit
  }
}
```

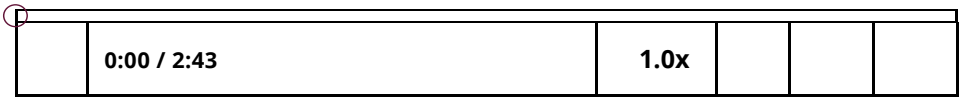
Remember, if you use the UART for the scope, you will not be able to use the UART for displaying data. The next video shows the simulator during debugging of the ADC device driver. During this test you should see a linear relationship between the voltage on PE2 and the number in the **ADCdata** variable. In the simulator, this relationship will be linear.

the real board the relationship will be similar but not exactly like this.

ADCdata = 4095***Vin**/3.3

Help

VIDEO 14B DEBUG THE ADC FUNCTIONS



PROFESSOR JONATHAN VALVANO: One of the learning objectives in this lab

is modular testing.

So rather than jump all the way to the end

and push the grade button, what we're going to do

is debug each module step by step.

So.

After you've written your A to D initialization,

and after you've written your function that will invoke the A to D converter

and convert the analog to digital value, what I want you to do

is test that operation.

So I'm going to show you how to debug those two.

So I've written those two, and I'm going to use

this very, very simple main program that initializes and then

converts, over and over again.

So let's debug.

So once in the simulator, I want to look at the A to D window here.

And so, in this window here, I'm going to set the parameter

to the connection for which I hooked it up.

And down here is the slide pot that I can move left and right.

In order to see what's going on, we're

going

to open a Watch window right here.

And what we're going to do is we're going to watch the ADC data.

Right here.

ADC data.

Enter.

OK.

So we're going to look at the variable which will contain the result of the A

to D converter as it runs.

All right, so let's run this.

And the idea here is, here's your slide pot, which goes from left to right

and back again.

Right here is the voltage at that point.

And over here, we see the A to D converter value.

Help

Part c) We suggest you observe the full scale range of your potentiometer. The Bourns potentiometer in the figures has a full range of 2 cm, so I set my Size to 2.0 cm. You should set your Size field to match your actual potentiometer. Also set your resistance values to match what you expect to build once you get to real-board testing. Next, you will calibrate the system in the simulator. In the simulator all responds are linear and ideal. Run your test program in part b) for a couple of distances, and then determine the A and B constants that convert the ADC sample into distance with 0.001 cm units.

Distance = ((*A****ADCdata**)>>10)+*B* where *A* and *B* are calibration constants

You should expect your offset *B* to be 0 when calibrating the simulator. On the real board, this offset will be a function of where you attach the ruler, and will probably not be zero. Write a C function that performs this conversion, called **Convert()**. You should debug this function and the optional output to the display in the simulator, using a main program like the following. This example uses the optional Nokia to display results. The next video shows the simulator screen the Nokia output matches the true distance. Don't expect it to be perfect, but it should be close because the simulator models an ideal transducer and circuit.

```
unsigned long ADCdata;
unsigned long Distance;
int main(void){
 TExaS_Init(ADC0_AIN1_PIN_PE2, SSI0_Real_Nokia5110_NoScope);
 ADC0_Init(); // initialize ADC0, chan 1, sequencer 3, software start
 Nokia5110_Init();      // optional: initialize Nokia5110 LCD
 EnableInterrupts();
 while(1){
  ADCdata = ADC0_In();
  Nokia5110_SetCursor(0, 0);   // optional: move cursor to top
```

```
  Distance = Convert(ADCdata);  // required step, you write this
  UART_ConvertDistance(Distance); // from Lab 11 convert to String
  Nokia5110_OutString(String);  // optional: output to LCD
 }
}
```

Help

VIDEO 14C DEBUG THE CONVERT FUNCTION

0:00 / 2:50 1.0x

PROFESSOR JONATHAN VALVANO: Again, one of the goals of this lab

is to introduce modular testing.

So, once the A to D converter works-- your software

for that works-- we're going to go on and write a function called

Convert, which will take the ADC data and convert it

into a distance which has units of 0.001 centimeters.

And so once you've written this function, you're going to test it.

So.

Let's show you how I would test it.

I have written a function-- you have to write your own--

and so I'm going to test it in the simulator.

This time, I'm going to use a little more complicated of a program.

We see, I'm going to use one here that initializes the A to D converter,

initializes the display-- this display is optional, but it is simulated.

I'm going to convert the value, and then I'm

going to convert that ADC into a distance,

and I will display that on the Nokia.

All right, so we can open up and look at the Nokia here.

Here's the Nokia.

Last time, we watched the ADC data.

And this time, I want to look at the distance parameter, here.

Distance.

Distance.

Enter.

As I convert it into the display, it actually

also creates a string, which I can look at.

OK, so this data I'm going to look in decimal.

OK.

So now I'm going to, again, run the program, here.

And now this time, I want to do a couple of conversions.

First, I see from before that the voltage, here,

Part d) Next, you should configure SysTick interrupts to sample at 40 Hz, so that you can perform the ADC sampling at a regular rate. Sampling at a known and regular rate is essential for real-time embedded systems. It is not required, but I suggest you add a debugging monitor on an extra pin. I used PF1 (the red LED), but feel free to use any digital output. If we add a global variable called Flag, then the Distance and Flag variables together constitute a mailbox. The pseudo code for SysTick ISR is

1) Toggle PF1
2) Toggle PF1 again
3) Sample the ADC, calling your **ADC0_In()**
4) Convert the sample to Distance, calling your **Convert()**, and storing the result into the global
5) Set the Flag, signifying new data is ready
6) Toggle PF1 a third time

The pseudo code for main program is

1) Activate **TExaS_Init**
2) Enable the ADC, calling your **ADC0_Init()**
3) Optional: enable the display, either
   a) Call **Nokia5110_Init()**
   b) Call **UART0_Init()**
4) Configure SysTick for 40 Hz interrupts
5) Configure PF1 as a regular GPIO output
6) Enable interrupts
7) Perform these steps in an infinite loop
   Clear the Flag to 0
   Wait for the Flag to be set

Convert Distance to String, calling your **UART_ConvertDistance()**

Optional: output the string

a) **Nokia5110_SetCursor(0, 0); Nokia5110_OutString(String);**

b) **UART_OutString(String); UART_OutChar('\n');**

Help

The simulation screen should look similar to the previous video, showing the true distance in the dialog matches the **Distance** in the watch window and the distance on the display. Figures 14.6 and 14.7 illustrate proper deployment of interrupts. Figure 14.6 shows the ISR running exactly at 40 Hz (25 ms), and Figure 14.7 shows the time to execute the ISR is small compared to the time between interrupt triggers.

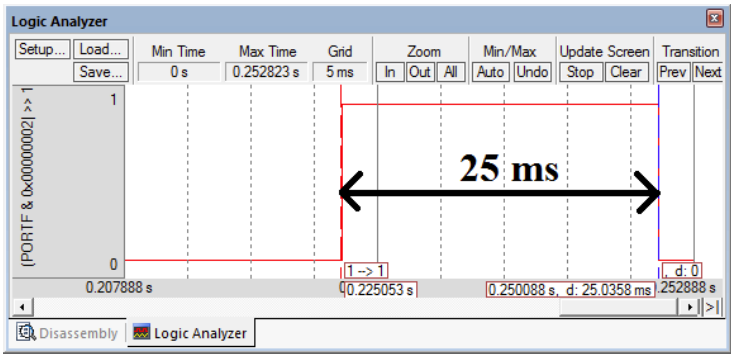


Figure 14.6. Zoomed out debugging screen shows SysTick interrupting every 25 ms.

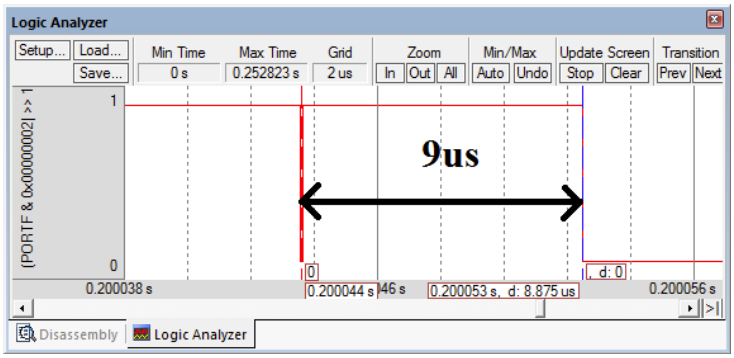


Figure 14.7. Zoomed in debugging screen shows SysTick ISR executes in about 9µs.

LAB 14 SIMULATION GRADER

Help

0:00 / 1:44 1.0x

PROFESSOR JONATHAN VALVANO: Hi.

In this video, I'll show you how to earn your grade in Lab 14.

We begin in edX, and we take this four-digit number here.

I'm going to copy it right there-- copy.

I'm going to go over to Keil.

And after you've debugged your program and ready to get a grade,

we will run it in the simulator.

We make sure that we're set up for simulation mode.

Then we build it, and then we debug.

And I've configured here the simulator to see three things.

You do not need a display to pass the simulation grader. There are three steps to the simulation grader:

0) Initialization tests will look specifically to make sure SysTick is interrupting at 40Hz. The system is running at 80 MHz, and there is only one value the grader will accept for the SysTick RELOAD register that will produce an interrupt every 25 ms;
1) The grader will check to see if the ADC is initialized properly and the channel you have selected in the dialog window matches the channel you configured the ADC. You must use ADC0. However, you may use channel 0, 1, or 8. You must use sequencer 3, and software start;
2) You must store your position measurements into the global Distance with units of 0.001 cm. It doesn't matter what settings you select for Size, Series Resistance or Slide Pot Resistance. Therefore, I suggest you make these settings match the hardware you will build on the real board. The grader will move the slide pot to five different positions, and calculate the average accuracy comparing truth to your measurements. After each move to a new position, the grader will wait at least 35ms and observe your global variable Distance. Let n be the number of data points, let t_i be the true distance, and let m_i be your measured distance. Both distances have units of 0.001 cm, and the index i varies from 0 to n-1. The grader determines accuracy by calculating the average difference between truth and measurement,

Average error (with units in 0.001 cm) = (sum($|t_i-m_i|$))/n

Average error must be less than 0.02 cm. The simulation grader does not check to see if you output the results to the LCD or the UART.