ECE/CSE 474:
Intro to Embedded Systems

Lab 3: ADC, UART, and Bluetooth

Introduction

For lab 3, you'll be using everything you've learned so far in order to use the TM4C1294NCPDT's Analog to Digital Converters (ADC) to measure voltage readings from the potentiometer and data from the on-board temperature sensor. Then, you'll use the board's Universal Asynchronous Receiver/Transmitter (UART) hardware to transmit these values serially.

By the end of this lab, you will:

- Have even more experience implementing timers and interrupts.
- Be familiar with the TM4C1294NCPDT's analog-to-digital converters.
- Know how to use analog components in a digital system.
- Gain experience configuring and using a UART interface.
- Gain experience with using a Bluetooth module

Task 1: Analog to Digital Readings

The ADC Module

The ADC module of the TM4C1294NCPDT features a 12-bit resolution and supports 20 input channels plus an internal temperature sensor. The ADC module uses Successive Approximation Register (SAR) architecture to deliver a 12-bit, low-power, high-precision conversion value. The range of this conversion value is from 0x000 to 0xFFF (0 to 4095). It uses internal signals Voltage Reference Positive (VREFP) and Voltage Reference Negative (VREFN) inputs as references to produce a conversion value from the selected analog input. VREFP is connected to VDDA and VREFN is connected to GNDA, as shown in Figure 1.

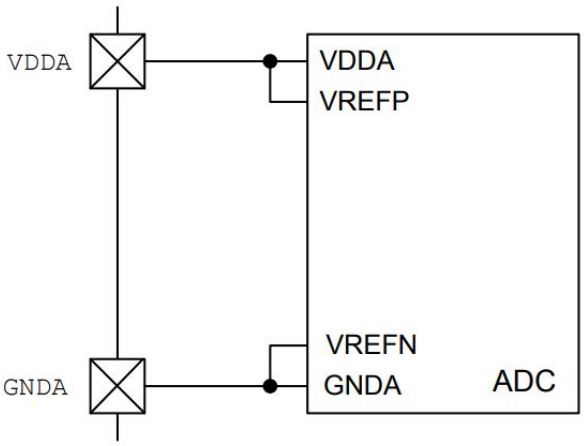

Figure 1. ADC Reference Voltage (Figure 15-8 in the datasheet)

More detailed information about the ADC module on the TM4C can be found on page 1053 of the datasheet. To learn how to use the digital readings of the ADC, check section 15.3.4 of the datasheet.

To configure the ADC, follow the steps in the provided Task 1a starter code (Task1a.zip) on Canvas. Please go over all the provided code files before doing any implementation. . Some things to note:

- According to section 15.3.2.7 of the datasheet, the PLL (phase-locked loop) needs to be enabled to clock the ADC. Use 60 (MHz) as the default input parameter. The ADC is a bit unpredictable when altering clock frequencies with the PLL. When you enable the ADC, make sure to change the clock source to the alternative clock source (ALTCLK) in the **ADCCC** (this forces the ADC to only listen to the PIOSC when sampling data)
    - The PLL controls alternate clock frequencies, and you can change the frequency by changing the input parameter for PLL_Init(). You do not need to worry about how the PLL works and you will not need to change the PLL_Init() code at all.
- After configuring the **RCGCADC**, you might need to insert a small delay before moving on.
- You will be sampling the ADC by triggering it with a timer at 1 HZ. As such, you'll have to configure the ADC to be triggered by your timer *and* configure the timer to enable it to trigger the ADC.
- The timer does not need an interrupt. Instead, you need to enable the interrupt corresponding to the ADC that the timer is triggering.
- Use Sample Sequencer 3. Since this ADC application requires only one input, Sample Sequencer 3 will be the easiest to configure. You can read the converted ADC value from the **ADCSSFIFO3** register.

Potentiometer

A potentiometer is a manually adjustable variable resistor with 3 terminals. Two terminals (end pins) are connected to both ends of a resistive element, and the third terminal (middle pin) connects to a sliding contact, called a wiper, moving over the resistive element. The position of the wiper determines the output voltage of the potentiometer. The potentiometer essentially functions as a variable voltage divider. The resistive element can be seen as two resistors in series, where the wiper position determines the resistance ratio of the first resistor to the second resistor. In Figure 2, the resistance R between the two terminals on the left can be calculated as: R = Output Voltage / Input Voltage * Total Resistance.

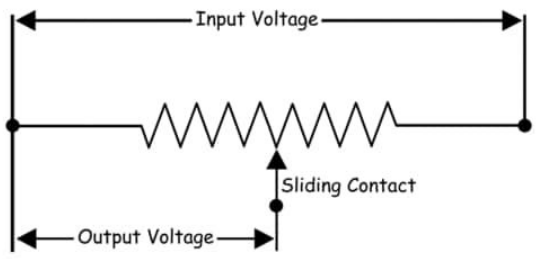

Figure 2. Internals of a Potentiometer

Internal Temperature Sensor

The built-in internal temperature sensor of the TM4C LaunchPad notifies the system when the internal temperature is too high or too low for reliable operation. It converts temperature into voltage according to the plot and equation shown in Figure 2.

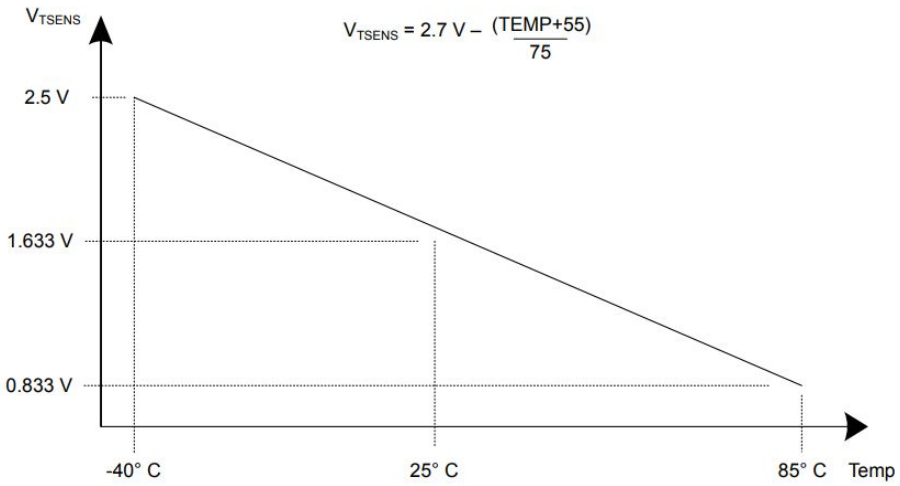

Figure 3. Internal Temperature Sensor Characteristic (Figure 15-11 in the datasheet)

This sensor voltage is read by the ADC module. Check out section 15.3.6 to get more information about how to convert the ADC reading into a temperature value.

Task 1 Assignment

a. Controlling the LEDs using the potentiometer

Set up the 10kΩ (kiloohm) potentiometer using the following steps:

i. Connect one of the end pins to 3.3V and the other end pin to GND.

ii. Connect the middle pin to a GPIO pin that has ADC signal capability. Refer to the datasheet for the list of these pins.

Program the TM4C to measure the voltage of the middle pin and convert it to the resistance of the potentiometer using this equation: **R(kΩ) = ADC value / 4095.0 * 10.0** (derived from the equation in the Potentiometer section). Use the on-board LEDs to represent the resistance based on the threshold table below.

Table 1: On-board LED Output Based on Potentiometer Resistance Thresholds

Blinking LEDs	Resistance R (kΩ)
D1	0.0 - 2.5
D1 & D2	2.5 - 5.0
D1 & D2 & D3	5.0 - 7.5
D1 & D2 & D3 & D4	7.5 - 10.0

b. Displaying board temperature

Program the TM4C to read the temperature data from the internal temperature sensor. Use printf() statement to print out the temperature values, and monitor the outputs using

the Terminal I/O in the 'View' tab of IAR Workbench. Use the two onboard switches to change the system clock frequency to 12 MHz or 120 MHz with the provided PLL driver. Faster frequency should heat up the system (i.e. 120 MHz), and lower frequency should cool the system (i.e. 12 MHz) by about +/- 2 degrees Celsius. .

Note: In order to use printf, you need to #include <stdio.h>.

Hint: This task should be straightforward with your code for Task 1a. Refer to section 15.3.6 to figure out the necessary adjustments you need to make.

Task 2: UART and Bluetooth

Universal Asynchronous Receiver/Transmitter

The Universal Asynchronous Receiver/Transmitter (UART) is a commonly used hardware interface for simple asynchronous serial communication. In this part of the lab, you will be using the TM4C's UART hardware to serially communicate from the board to your computer. With any serial communication, you need a physical connection between the devices that are communicating. The debug port on the TM4C (Figure 3) provides a versatile connection between the microcontroller and your computer, and it turns out that it can also provide an avenue of UART communication.

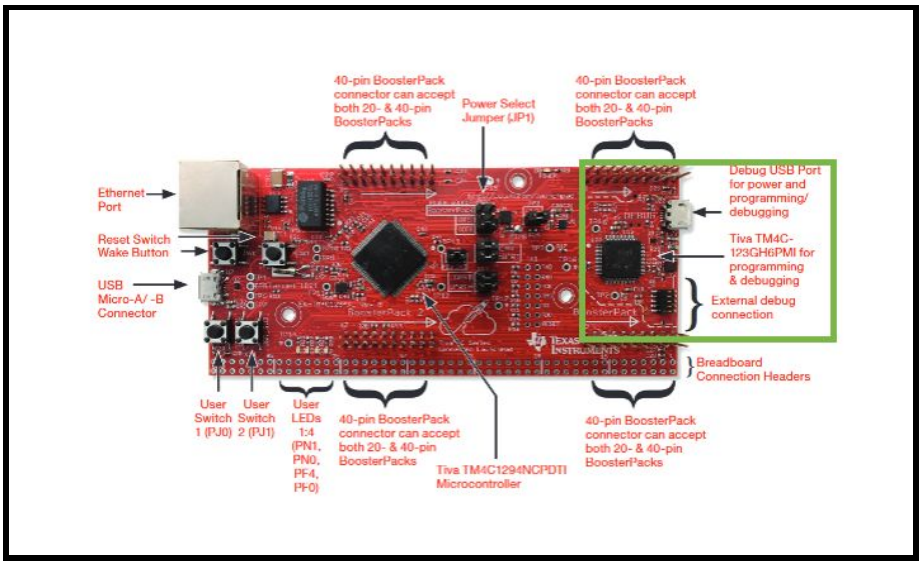

Figure 3: The Debug Port of the TM4C1294XL Tiva Board

The debug port, when connected to a computer, produces a virtual com port with which your computer can interface. Figure 4 visualizes the process of communicating using the virtual port. Basically, this virtual port allows your computer to receive and send information using the UART communication protocol. The Tiva board is set up such that the **UART0** module can interface with this virtual port, and this is the module that you'll be using for serial UART communication. You can learn more about the debug port in section 2.3 of the TM4C1294XL User Manual.

You need to set up **UART0** to transmit data from the microcontroller to your computer. All the information you need is in the datasheet. If you do everything correctly, UART data should be transmitted to your computer from your microcontroller only using the USB cable that connects the USB port of your computer to the Debug Port of the microcontroller. Here are some hints to point you in the right direction:

- The UART module requires the use of GPIO pins, which means you need to set specific pins to their alternative UART functions.
    - Each UART module has its own set of pins that you must configure for proper functionality. The pins that should grab your attention here are PA0 and PA1.
    - Look to Table 10-2 on page 743 of the datasheet for Alternate Function assignment for GPIO.
    - Avoid configuring the **GPIODIR** when selecting the alternate function for UART pins.
- The **Baud Rate** you choose does not have to follow any specific guidelines, but it is suggested that you use 9600 for consistency's sake.
- You probably should keep the default settings for packet transmission (no **parity** and 1-bit **stop bits**)
- Sending data takes time. Make sure that your firmware design completes transmissions **one character at a time** instead of overrunning your UART module. More on this and data transmission can be found in 16.3.3 of the datasheet. Polling for completed transmission in this application is acceptable.
- Initially, the UART FIFO buffer is disabled. Enabling the UART FIFO expands the **UARTDR** into a 16-byte buffer. The FIFO is unnecessary for basic applications of the UART, but it is available to you if you wish to use it.

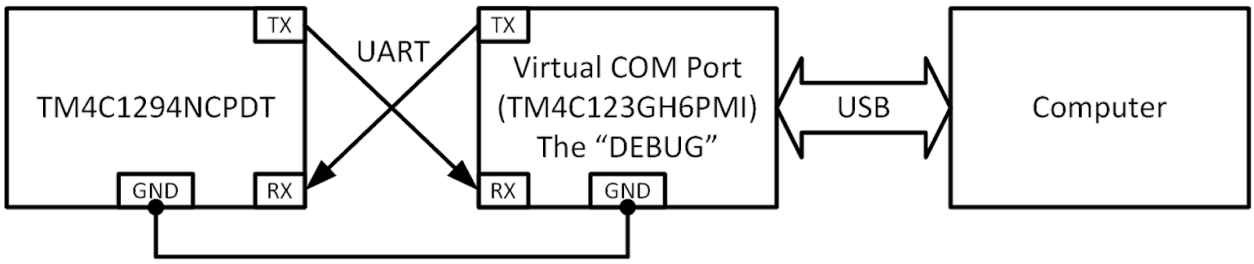

Figure 4: The Virtual COM Port (VCP) executed by the TM4C1294XL's On-Board In-Circuit Debug Interface (ICDI) and its corresponding connections.

To complete the tasks below, you will need to read through the two documents listed here. It is recommended that you complete Task 2a before moving onto Task 2b:

- ***PuTTY for UART Communication***
- ***Bluetooth for UART Communication***

Task 2 Assignments

a. **Sending temperature readings via UART**
Use UART serial communication to print the temperature readings from Task 1b to the terminal in PuTTY. Figure 6 in the tutorial document for PuTTY gives a good example of visualized temperature readings that would be passable for demoing. **Your demo can be as simple as printing a floating point value representing temperature in your PuTTY terminal**.

b. **The "Return-to-Sender" function**
It turns out that you can send information from your computer to the microcontroller using the PuTTY interface. Specifically, any characters that you type in the PuTTY Terminal are transmitted over UART to your microcontroller. Create a new program that receives a single character via UART and immediately transmits back the character that it receives (a "Return-to-Sender" function), and demonstrate this program by using the PuTTY interface. **Note that this is a program implementation, not a hardware implementation.** Moreover, use **Bluetooth** communication for this specific task. If you do this task correctly, the characters you type in the PuTTY terminal should appear in the PuTTY terminal.

Lab Demonstration and Submission Requirements

- Submit demo videos of Task 1 and Task 2 on their respective due dates. The videos should show the expected results described in Task 1 and Task 2 on the TIVA Board. Feel free to add narration or text in the video. Each video must be less than 90-seconds long. To receive full credits, your videos must thoroughly demonstrate all the required functionalities. Before submitting, refer to the rubric shown on the "Demo Video" Canvas Assignment.
- Write a very brief Lab Report, as framed by the "Lab Report Guide" on Canvas -> Files -> Labs -> Lab Report and Code. Make sure to include a drawing of your FSM design for this lab.
- Revise the style of your Lab Code, as framed by the "Programming Style Guide" on Canvas -> Files -> Labs -> Lab Report and Code.
- Submit your Lab Report as a pdf, and submit your Lab Code (.c and .h files). Please do not submit compressed ZIP folders or other files such as the workspace files (.eww) and project files (.ewp). Before submitting, refer to the "Lab Report and Code Rubric" on Canvas -> Files -> Labs -> Lab Report and Code.
- On Padlet, write about a problem you had in the lab and the fix to it, and share a tip or trick you learned while working on the lab. You can also share an aha moment that you discovered while working on the lab. Avoid duplicating comments made by your classmates. NO videos for this Padlet task, please use textual comments. The link to the Padlet can be found on Canvas -> Assignments -> Lab3: Tips and Tricks. Please note that the Tips and Tricks assignment is part of the Community Participation, and thus the *Grace Period* does not apply.