Kiana Peterson and Nattapon Oonlamom
CSE 474: Intro to Embedded Systems
November 16th, 2023
Lab 3: ADC, UART, and Bluetooth

Procedure

The purpose of this lab is to experiment with using the board's Analog to Digital Converters (ADCs) to gather sensor data as well as the board's Universal Asynchronous Receiver/Transmitters (UARTs) to serially transmit data through both wired and wireless communication.

Task 1a: For this task, we were to control the LEDs using the 10kΩ potentiometer. By following the pins provided in Figure 1, we connected one of the end pins to 3.3V, the other end pin to GND, and the middle pin to a GPIO pin that has ADC signal capability. To figure out which GPIO pin has ADC signal capability, we referred to the datasheet of TM4C. As for the software side, we must program the TM4C to measure the middle pin's voltage and convert it to the resistance using the following equation: R(kΩ) = ADC value / 4095.0 * 10.0 using the on-board LEDs to represent the threshold in Figure 1.1.

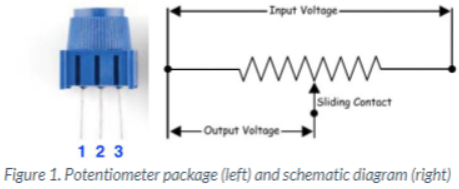

Figure 1. Potentiometer package (left) and schematic diagram (right)

Figure 1: Potentiometer

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

Figure 1.1: On-board LED Output

To approach the software side of this task, we used the provided base code from the teaching team. First, we initialize the 4 on-board LEDs with the corresponding GPIO pins. Secondly, we initialized ADC0 SS3 for analog to digital data conversion. Third, we initialize the Timer0A to trigger ADC0 at 1 HZ. Then, we implemented the ADC ISR under the ADC0SS3_Handler, saving the ADC value to global variable ADC_value to be converted to resistance in the main program. We used the if/else conditions to change the pattern of LEDs based on the calculated resistance. Note that the approach for these initializing steps and writing the ISR were mainly reading the datasheet and debugging the syntax through trials and errors.

Task 1b: For this task, instead of reading the voltage data from the potentiometer, we programmed the TM4C to read the temperature data from its internal temperature sensor. Therefore, all we need to do is connect the board to the computer via USB. We used the printf() statement to monitor the output temperature values using Terminal I/O. Additionally, the two on-board switches were used to to change the system clock frequency to 12 MHz or 120 MHz. (Faster frequency should heat up the system and vice versa.) To approach this, we first initialized the switches and the temperature sensor. Then straightforward with Task 1a code, we made necessary adjustments (i.e. removing the analog input port configurations) through trial and error. Then, to finish up we included to the files `#include <stdio.h>` to use printf().

Task 2a: For this task, we were to use UART0 to print the temperature from Task 1b to the terminal in PuTTY. Since this task does not require the bluetooth module, the hardware is simply connecting the board to the computer using USB. To approach the software side of this task, we precisely studied the PuTTY for UART Communication document and the TM4C datasheet provided to initialize the UART. After the initialization, we

added a for loop into the ADC0SS3_Handler to print the data to the PuTTY terminal character by character. It is also important to note that we had to convert the temperature data to a character array to avoid scrambling of data when being transmitted to the PuTTY. Additionally, we installed PuTTY and configured the port settings to match the UART settings of our TM4C as shown in Figure 2. This task was mainly following the provided instructions and datasheet and debugging through trials and errors.

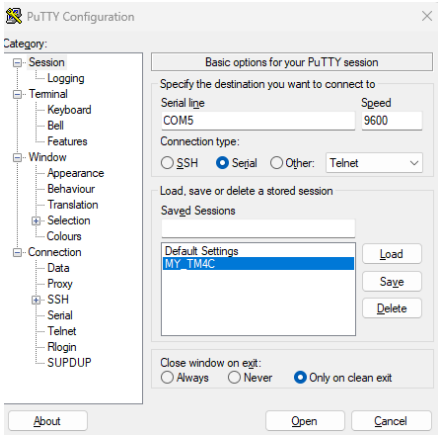

Figure 2: PuTTY Setting as Instructed

Task 2b: For this task, we were to use the bluetooth module in Figure 3 to send information from our computer via the PuTTY terminal back into the PuTTY terminal, meaning that when we type a character into the PuTTY terminal, it should immediately appear in the PuTTY terminal. As the bluetooth module still utilizes serial communication, the UART settings remained mostly unchanged. The hardware setup was simple, but could be a source of error if not careful. The VCC pin on the bluetooth module was to be connected to the 5V pin on the TM4C, the GND was connected to ground, TXD (transmitter) was connected to the TM4C PA4 pin corresponding to UART3 RX (receiver), and RXD (receiver) was connected to the TM4C PA5 pin corresponding to UART TX (transmitter). The PuTTY terminal and bluetooth modules were set up according to the Bluetooth for UART Communication document provided. To begin implementing the wireless communication, we checked that all of the data was received (8 bits corresponding to a character). From there, we stored the data from the receiver into a char variable (userInput), waited to make sure the transmitter was not busy, and then loaded it into the UART for transmission back to the PuTTY terminal.

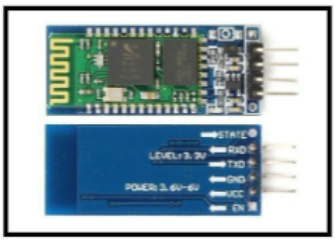

Figure 3. Front and back view of the Bluetooth module

Figure 3: Bluetooth Module

Table 2: Recommended UART Modules (TM4C1294XL)

UART	TX Pin	RX Pin
UART2	PA7	PA6
UART3	PA5	PA4
UART4	PK1	PK0
UART5	PC7	PC6

Figure 3.1: UART Pins

Result

Task 1a: As a result, we composed the hardware shown in Figure 4. When running the program we can successfully control the LEDs using the potentiometer. Every time the values of the potentiometer we input changes, the on-board LEDs respond accordingly to their threshold values.

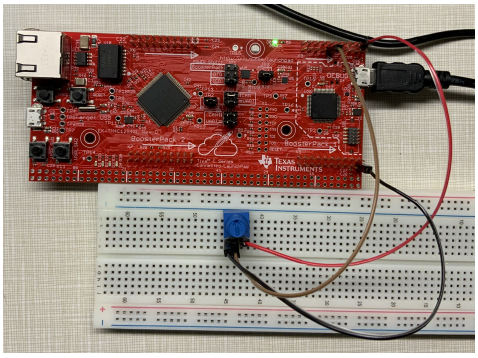

Figure 4: Task 1a Hardware

Task 1b: As a result, when running the program we can successfully read the temperature data from the board's internal sensor. Every time the switch representing 120MHz (faster frequency) is pressed, the printed temperature output on the Terminal I/O increases as the system heats up while when the switch representing 20MHz (slower frequency) is pressed, the printed temperature output on the Terminal I/O decreases as the system cool down.

Task 2a: After the program read the temperature from the internal sensor, the TM4C successfully sent the temperature readings from Task 1b to the PuTTY terminal using UART. We can see the printed floating point value representing temperature in the opened PuTTY terminal.

Task 2b: As a result, we were able to echo input characters back to the PuTTY terminal via the bluetooth module and the hardware setup shown in Figure 5.

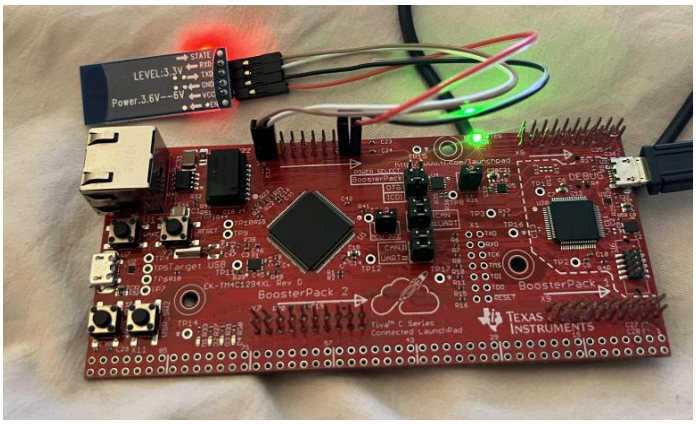

Figure 5: Task 2b Hardware