Lab4: DTMF Texting Over an Asynchronous Serial (UART) Link

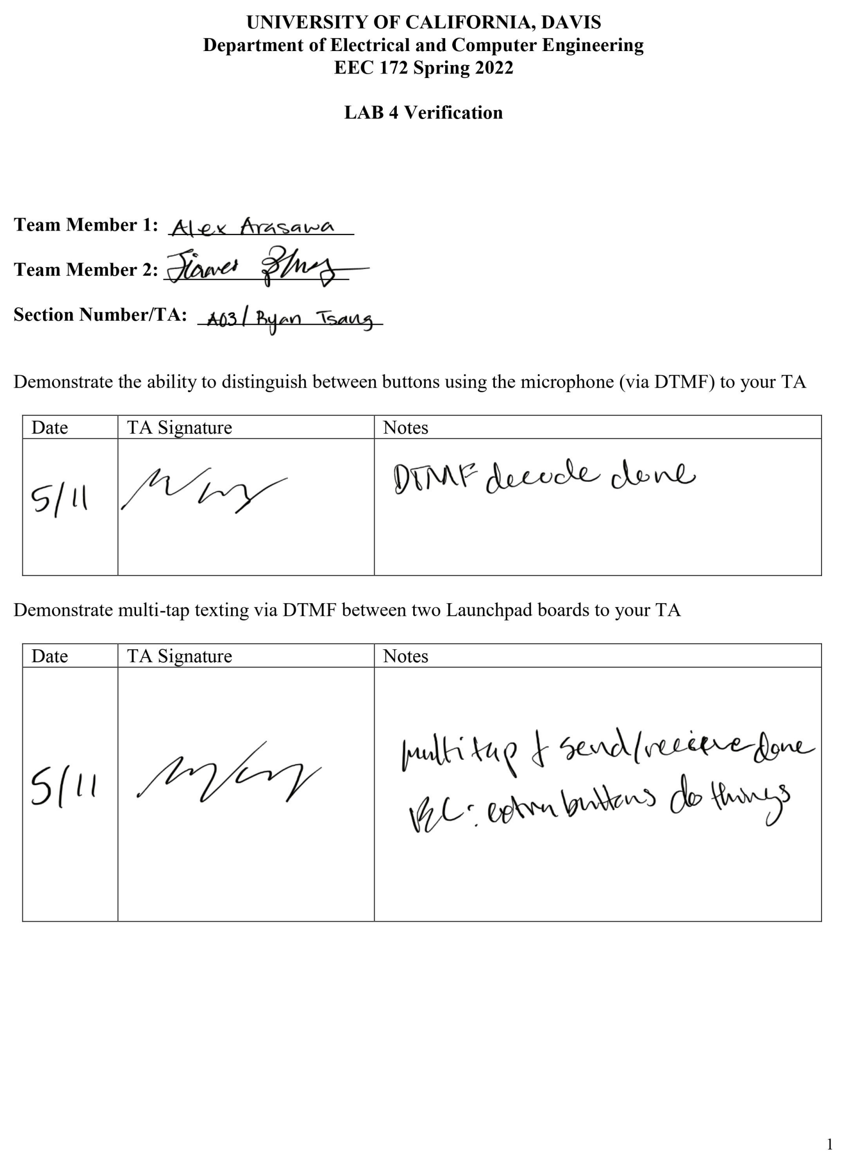
Alexander Arasawa, Jiawei Zheng

June 11, 2022

EEC 172

Professor Soheil Ghiasihafezi

1 Check off from TA:



UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering
EEC 172 Spring 2022

LAB 4 Verification

Team Member 1: Alex Arasawa

Team Member 2: Jiawei Zhang

Section Number/TA: A03 / Ryan Tsang

Demonstrate the ability to distinguish between buttons using the microphone (via DTMF) to your TA

Date	TA Signature	Notes
5/11	[illegible]	DTMF decode done

Demonstrate multi-tap texting via DTMF between two Launchpad boards to your TA

Date	TA Signature	Notes
5/11	[illegible]	multitap & send/receive done EC: extra buttons do things

1

2 Introduction

The first goal of the lab is to detect and decode the DTMF tones readings coming out of a ADC that is connected to a microphone and display the corresponding buttons pressed to the terminal. In the second part, similar to Lab 3, we have the two CC3200 LaunchPads communicate to each other by sending texts to each other over UART1.

3 Background

CCS software is an IDE for configuring TI's microcontrollers and embedded processors. The software contains many tools to help with creating and debugging applications: CSS includes an optimized C/C++ compiler, a code editor, a project build environment, a debugger, and many other features.

CCS Uniflash is a standalone tool used to program flash memory of TI's microcontrollers and program flash memory of Sitara processors.

TI Pin Mux Tool is a software used for configuring pins, peripheral signals, and other components of a system.

Saleae logic is a logic analyzer software that is used to capture waveforms and debug ports/programs.

References:

`https://www.ti.com/tool/CCSTUDIO`

`https://www.ti.com/tool/UNIFLASH`

`https://www.ti.com/tool/SYSCONFIG`

`https://www.saleae.com/`

4 Goals/Objectives

4.1 Hardware Interface

Connect the microphone to a clean 3.3V power supply where the 5V goes through the voltage regulator. Place the Electret microphone into the circuit connecting it the onboard 3.3V power supply and wire the out signal to the ADC. The ADC output is wired to a GPIO on the LaunchPad.

What you should learn from this task:

- Why the microphone should be connected to the cleanest 3.3V power supply.
- How to wire external components together with each other and with the CC3200.

4.2 Software

4.2.1 DTMF detection and decoding

Modify example DTMF Detection Goertzel Algorithm code by setting a general-purpose timer to interrupt at 16 kHz, modify code to have 410 samples, and change SPI bit rate to at least 400 kbps.

What you should learn from this task:

- How to use Goertzel algorithm to detect DTMF tones(note: you do not really have to understand the algorithm to use it).
- How to precalculate the needed coefficients for Goertzel algorithm off-line.
- How to setup a periodic timer interrupt using timer_if so you can periodically sample readings from the ADC.
- How to configure a timer from timer_if to 16 kHz.
- How to get readings from the ADC using SPITransfer.
- How to bit shift the bits in the the two bytes readings coming from ADC so you can get the correct data that makes sense.
- How to account for DC bias in data readings from the ADC.
- How to modify the confidence interval in the post_test() function to adjust for and ignore noise.
- How to prevent one physical button press from registering as two or more button presses(we used UtilsDelay to stop sampling for a while after a button press have been detected).

4.2.2 Recommended Code Flow

Recommended code flow so SPI does not interleave with communication between the OLED and ADC.

What you should learn from this task:

- How to write organized code to mitigate race conditions.
- Where to disable and re-enable sampling interrupts so you do not overwrite important data or create race conditions.

4.2.3 Texting

Build on techniques learned from Lab3 to develop an application that uses DTMF from a device to compose text messages using multi-tap text entry for text messaging over UART between the two CC3200 LaunchPads.

What you should learn from this task:

- How to enable UART interrupts and use UART interrupt handlers.
- How to use UART API to allow for interrupt-based communication.
- How to differentiate successive button presses with short time periods in between from successive button presses with long time periods in between(we used another timer from timer_if to track time between successive button presses).

5 Methods

5.1 Hardware Interface

This part of the lab required assembling a circuit involving an electret microphone, A/D converter (also referred to as ADC), low dropout voltage regulator, and CC3200 LaunchPad. First we setup the low dropout voltage regulator for lowering an input +5V to +3.3V. The component has 3 legs with the left leg connected to the 5V pin, middle leg as the new 3.3V output, and right leg tied to GND. Then we wired the microphone. GND, A/R, and Gain were wired to ground. Then VDD was wired to a clean +3.3V power source from the voltage regulator. The OUT pin was then wired to the analog input on the ADC. Next, the ADC was setup. Analog IN+ is the input pin and receives the signal from the microphone. V_{SS} and analog IN- are tied to GND. V_{REF} and V_{DD} are tied to the LaunchPad 3.3V pin. CLK is tied to the SPI_CLK pin 5, D_OUT to SPI_MISO GPIO pin 8, and CS to a GPIO pin 18 on the LaunchPad. After testing ADC sampling we then connect the OLED which required modifying the pin maps for the CC3200. This is because the OLED requires GPIO pins for Reset (pin 18), DC (pin 62), and OLEDCS (pin 61), in addition to those needed by the ADC.

5.2 Software

We configured a timer from timer_if to interrupt at a rate of 16kHz with a interrupt handler called readADC(). In readADC(), we would get data worth of 2 bytes from ADC using SPITransfer(). The data is then bit shifted to get the 10 bits of data we want. The corrected data is then subtracted with a DC bias value(ours was 390) and appended to our samples array. Once 410 samples are collected, main would disable sampling interrupts, perform Goertzel calculations, determine if a button was actually pressed in post_test(), and carry out corresponding actions for when a button was definitely pressed. When done, main resets the samples array and re-enable sampling interrupts to read in more button presses.

"We used separate lines on the OLED for composing message and received message. For the composing message, we created our own Letter struct and initialized a Letter array. Each Letter in the Letter array would have a character, color, and x position in the composing message(x position is helpful for deleting the last character when the user presses *). The user adds letters to this array by pressing buttons and the letters array would be send through UART1 to the other board when the user presses #. Before sending any letters, we send a "5" to other board to tell them that a new message is coming through the line. Then we would send a number 0-4(5 different colors) and a character for each letter in the composing message. On the receiving end, through the UART interrupt handler, the receiver erases it current received message because of the "5" and it would know which color to set to for each character so the receiver can print the new and correctly colored received message. Then we needed to make it so that they were able to take multi-tap entry. We added another timer from timer_if so we can keep track of the time in between button presses. Also, we created a global variable called 'prevbutton' to keep track of the last button pressed. If the same button was pressed and the time interval was small enough, we would update the current letter to the next one for that button. For example, button 2 are letters 'a', 'b', or 'c'. Once we coded button 2, the other buttons 3-9 was very simple because we can just copy the code for button 2 and change the letters. For button 1, we used a color array of predefined colors and we cycle through colors of this color array with setTextColor() everytime button 1 was pressed." - copied from our lab3, with some modifications.

6 Extra Credit: Symbols and Maybe Emojis?

For extra credit, we provided additional functionality using multi-tap entry by implementing four additional buttons for text messaging. The tone generator was 4x4 anyways and we were only using 3x4 so might as well take advantage of the 1x4 remaining buttons. The extra buttons added were 'A', 'B', 'C', and 'D'. They were implemented in a similar fashion to how buttons 2-9 were implemented. However, these buttons are different because they did not map to alphabet letters but to symbols in the ASCII table. For example, 'A' mapped to symbols !, ", , $, %, , and '. In some way, you can also say we implemented emojis because you can create emojis with symbols: ":)", ";)", ":(", and "*o*/".

7 Discussion

The first challenge we ran into was when we were assembling the circuit. Due to the number of pins and wires needed for interfacing with all the components it got hard to keep track of where things were going. When it became time to start testing the input from the ADC it was hard to tell whether the ADC was fried or there was a wiring issue. This is an inevitable issue since only one logic analyzer is given per group and only one group member can use it.

The second challenge was testing components. For this lab we were given 4 components total. In the process of assembling the circuit or testing output from some component it is easy for a fried component to fly under the radar. Since many things can go wrong between the wiring, pins, and coding. One of the partners in this group actually ended up with a broken ADC and needed it replaced. A significant amount of time was spent debugging problems related to these first two challenges.

The third challenge was reading in data from the ADC. The easiest portion for the most part was integrating the example Goertzel algorithm into the original code (code from Lab 3). However, we had a hard time reading in data from the ADC until we received help the TA, who told us that we were supposed to use SPITransfer() with a buffer of 2 bytes.

8 Conclusion

This lab helped in learning how to work with signals and signal processing. First we learned how to setup the circuit. During which we worked through how the separate components needed to be setup such that the LaunchPad received a signal from the ADC for processing. Then modifying sampling frequency and block size helped us to learn how to work with the for sampling the analog inputs to the ADC. Next, for the coding portion we learned how to use the Goertzel algorithm to decode the signals from the ADC. To characterize them into the numeric inputs and four additional ones for extra credit. Lastly, we added the DTMF decoding to our code from Lab 3 that deals with sending text messages between the two CC3200 LaunchPads.

9 Contribution

We did all parts of the lab together so 50/50 contribution.