Lab3: IR Remote Control 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

Team Member 1: Alex Arasawa

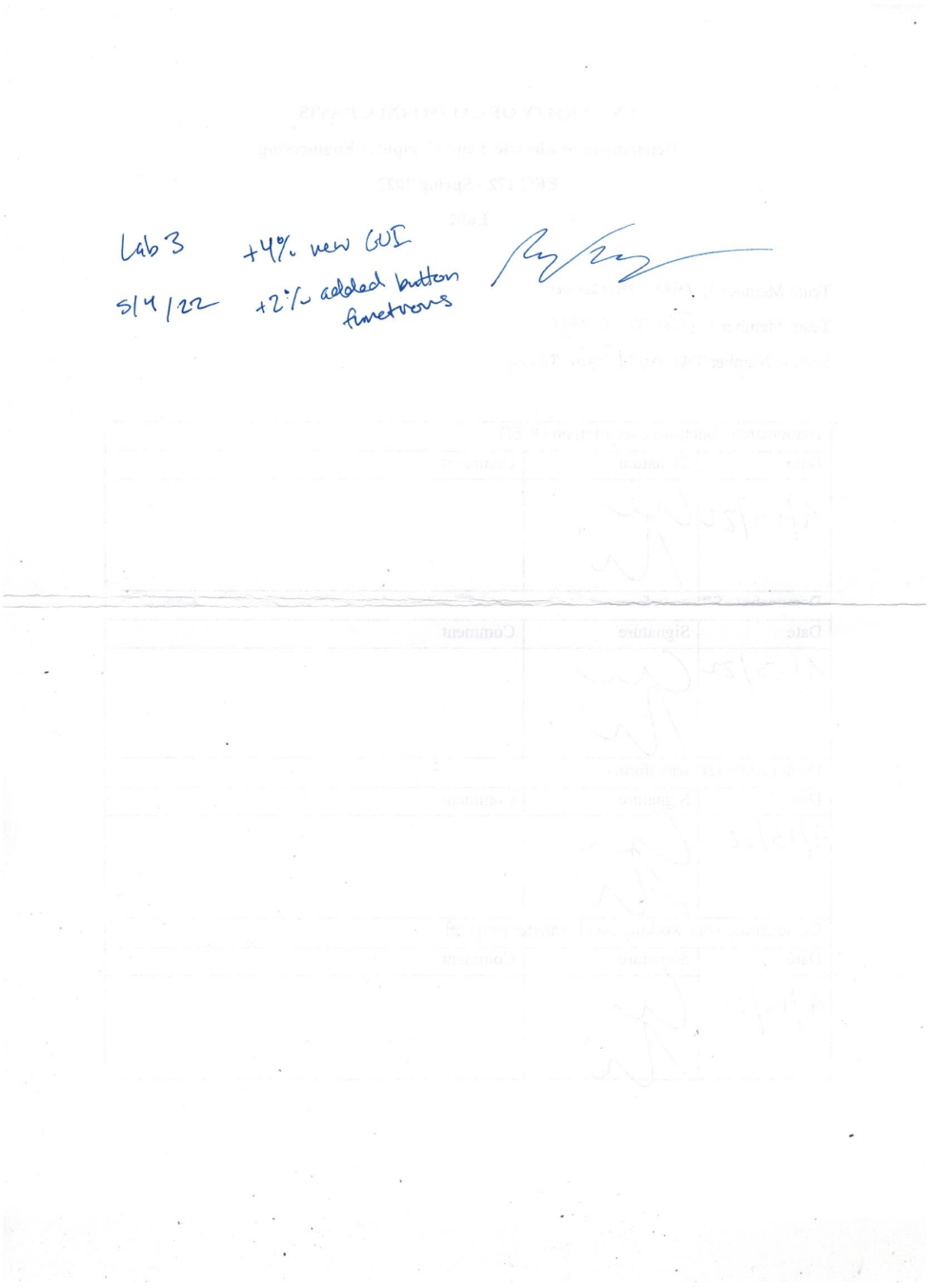
Team Member 2: Jiawei Zheng

Section Number/TA: A03 / Ryan Tsang

Demonstrate IR Decoding:		
Date	Signature	Comment
4/27/22		Code gets stuck after processing 1st button press On 4/29 (late) this issue was fixed

Demonstrate Texting:		
Date	Signature	Comment
4/29/22 (Late)		

2 Extra Credit:



3 Introduction

The first goal of the lab is to use the Saleae logic analyzer on the TSOP IR Receiver to determine the IR protocol used by the remote. In the second part, we learn how to create a program that will decode and translate the IR transmissions for 12 buttons (10 number keys, Mute, Volume Down). Lastly, we will connect the two CC3200 LaunchPads and develop an application that uses the IR remotes to send text messages between the two LaunchPads over UART1.

4 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/`

5 Goals/Objectives

5.1 Part I: Capturing and Characterizing IR Transmissions using Saleae logic

Use the Saleae logic analyzer to capture and characterize IR transmissions from the remote, then and identify the data format and IR protocol used by the remote.

What you should learn from this task:

- The data format and IR protocol used by the IR remote control(ours was pulse length coding and NEC code).
- How to read IR transmission waveforms.
- Understand what different parts of the waveforms represent(starting bit, data field, etc).

5.2 Part II: Decoding IR Transmissions/Application Program

Connect the IR Receiver to a GPIO pin on the CC3200 LaunchPad, write a program to decode and translate IR transmissions so you can display what button was pressed to the terminal.

What you should learn from this task:

- How to enable interrupts for GPIO.
- How to incorporate interrupts into programs.
- How to use interrupt handlers and SysTick.h/hw_timer.h to measure time difference between waveform edges(we used falling edge).
- How to translate transmission waveforms into binaries so they could be mapped onto buttons.

5.3 Part III: Board to Board Texting Using Asynchronous Serial Communication (UART)

Connect the two CC3200 LaunchPads together each interfaced with an OLED and develop an application that composes text messages using multi-tap text entry from the remotes to send messages between the 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 to track time between successive button presses).

6 Methods

6.1 Part I. Capturing and Characterizing IR Transmissions using Saleae logic

At the start we were able to classify the TV code based on the type of patterns it was putting out. First, we placed connected jumpers to the OUT pin of the IR receiver in the circuit. After testing whether the output was consistent we started to take screenshots of waveforms for numeric keys 0 through 9, MUTE, and VOLUME_DOWN. The reason we chose MUTE and VOLUME_DOWN is because the ENTER, DELETE, and LAST did not work for our TV code. Upon examining the waveforms we saw that they had a large initial pulse and smaller pull-downs that we later learned represented an extended button press. We looked through the documentation for the IR signal protocols that matched and we concluded that our TV code was following the NEC protocol and using pulse length coding.

6.2 Part II: Decoding IR Transmissions / Application Program

We copied over the interrupt demo program and looked through the code to get an idea about incorporating interrupts into our code. We decided to use SysTick for keeping time because the API they offered was very simple. When a falling edge is received, our falling edge interrupt handler would be triggered and the function would add the time the falling edge occurred to our time array. On detection of the second falling edge, we subtract the two times from each other to get the time interval. If the time interval is within a certain range of values, we know that it is the starting signal and so the falling edge handler can keep appending falling edge times to our time array. Else, if not within that certain range of values, we reset the time array by setting time array index to 0 to try again. For valid time arrays and in the data part of the waveform, shorter time intervals correspond to 0's and longer time intervals correspond to 1's. Each button had a different waveform and different binary number for data so we translated each binary number to decimal so we can map each button's data waveforms to their integer representation with #define. To print which button was pressed, we have the main stuck in a while forever loop that would only print out pressed buttons using switch cases when the time array is valid and filled and then reset time array index to 0 so the next button could be read in.

6.3 Part III: Board to Board Texting Using Asynchronous Serial Communication (UART)

The code for this part build upon the code from part II. 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 VOLUME_DOWN). 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 MUTE. 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(this time, from hw_timer.h because SysTick only offers one timer) so we can keep track of the time in between button presses. Also, we created a global variable to keep track of the last button pressed. If the same button was pressed and the time interval was small enough, then we 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.

7 Extra Credit

For extra credit, we implemented a couple of scrolling modes and a simple interface for the messages. The interface works like modern texting apps and displays the messages on opposing sides depending on who was talking. As for the scrolling we created a feature to scroll through the texts automatically (scrolls as new messages are added to message log and can no longer fit on screen) or manually (press a button to scroll through the message log). We added two new buttons, Channel Up and Channel Down, so the user can manually scroll through the message log. For message log, we had to use memory on the board to store send messages and received messages via our msg struct(built upon Letters struct) and array of msg.

8 Discussion

The first challenge we ran into was with the GPIO pin. When we connected the IR Receiver to the Launchpad we were trying to test if we got any transmissions at all. However no matter how we programmed the interrupts we did not receive any transmissions on the logic analyzer. Before we were about to change pins we saw that there are two pins for the GPIO pin we chose. Just switching over the jumper cable to the other 62 pin we started seeing waveforms on the logic analyzer.

The second challenge was figuring out which interrupt routine to choose. This partly interacted with our first problem with using the wrong pin. Since we did not see any transmissions, we thought that maybe we were choosing the wrong interrupt. The demo code used the a different kind of interrupt than what we ended up using. However, after we got the pin problem fixed we decided to use the SysTick interrupts to measure the time between falling edges.

The third challenge was figuring out how to register multi-tap button entries. Separating into two challenges of how to implement the secondary timer for measuring the time between presses; and how to handle those presses. First we found that we needed to use a different interrupt mechanism than the SysTick and had different configuration. Then we decided on a conditional loop logic flow in order that checks the timer, previous button pressed, and current character. The timer is so that a button being pressed within a small enough time interval can allow them to change the letter they are inputting. Allowing for wrapping when pressing the same button more than 3 times and moving to the next character based on the number of presses.

9 Conclusion

This lab helped in learning how to work with the IR receiver how to make the Launchpads interact with each other bidirectionally. First, we learned how to identify the IR protocols of the remotes that we used. Then how to characterize the waveforms of the protocol by taking advantage of SysTick interrupts. It helped to deepen our understanding of how to decode waveforms. Additionally, last week we learned how to use the UART to transmit data between Launchpads, this time it took it a step up making it so that they transmit messages to each other. Incorporating the OLEDs to be able to display the messages that they text each other. We learned how to combine the SPI and UART protocols together.

10 Code Comments

Since the code for the extra credit part contains the code from previous parts, we decided to only comment main.c in EXBoard1 folder so for "comments" please look there and not elsewhere...

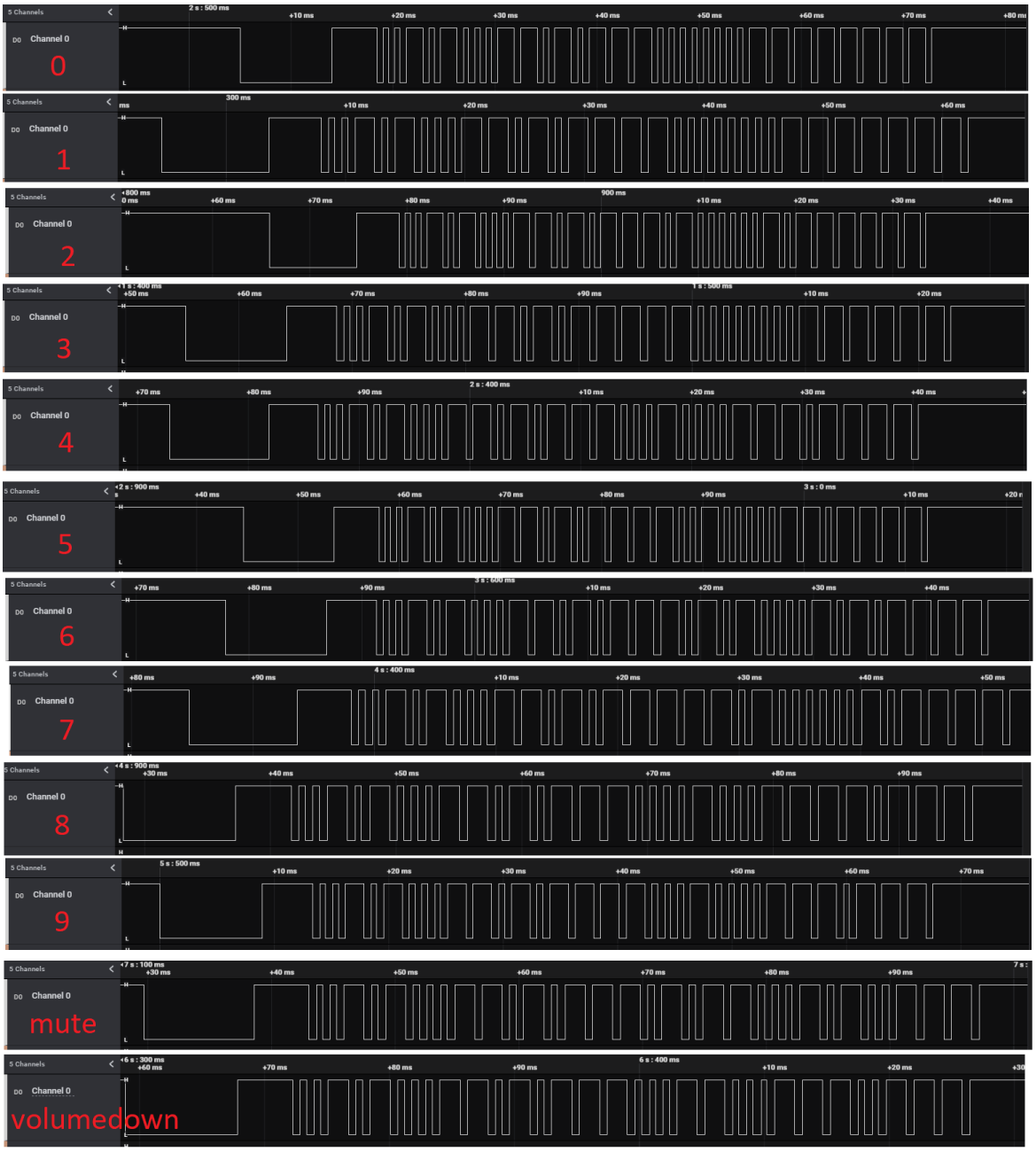
11 Contribution

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

12 Important Figures

12.1 Part I. Capturing and Characterizing IR Transmissions using Saleae logic

A printout of the waveform for each button.

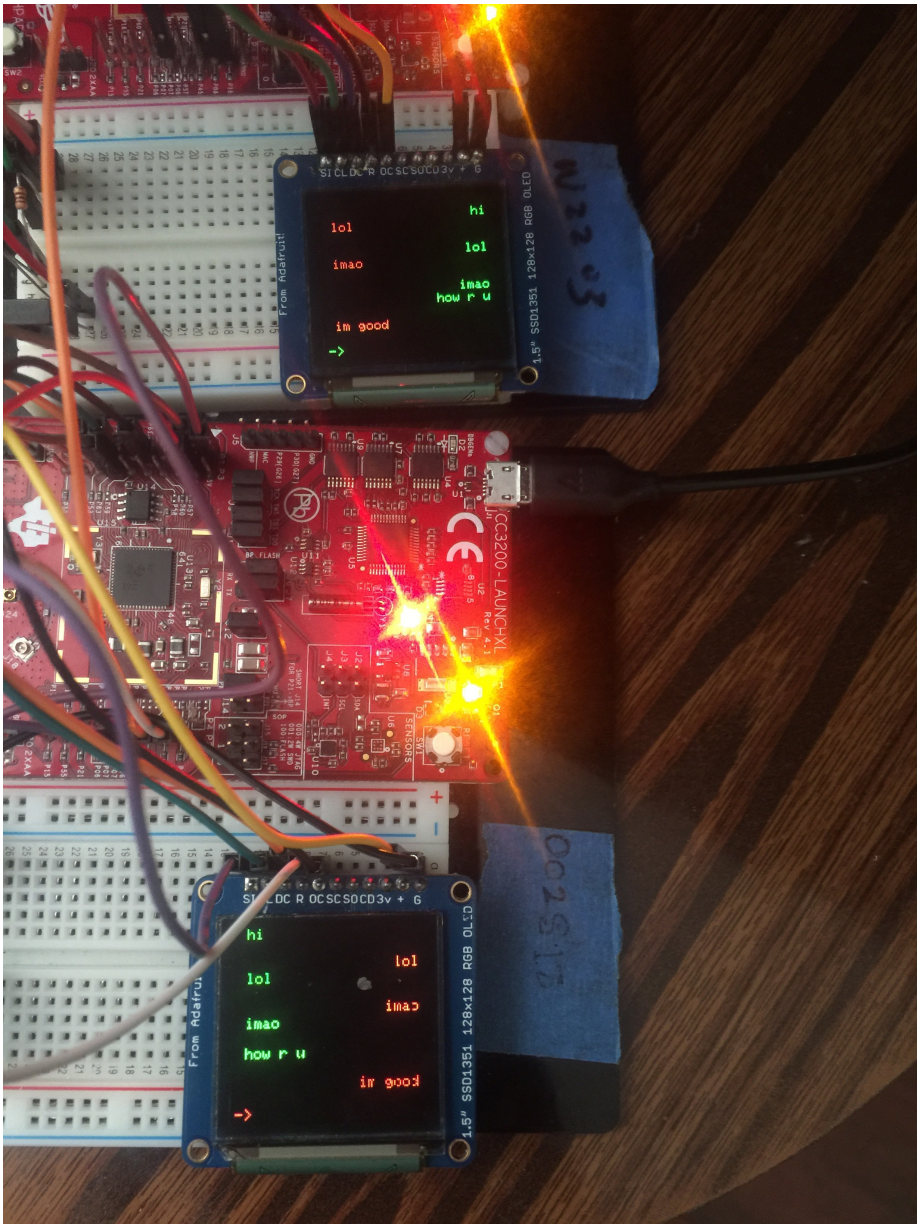


12.2 Part II: Decoding IR Transmissions / Application Program

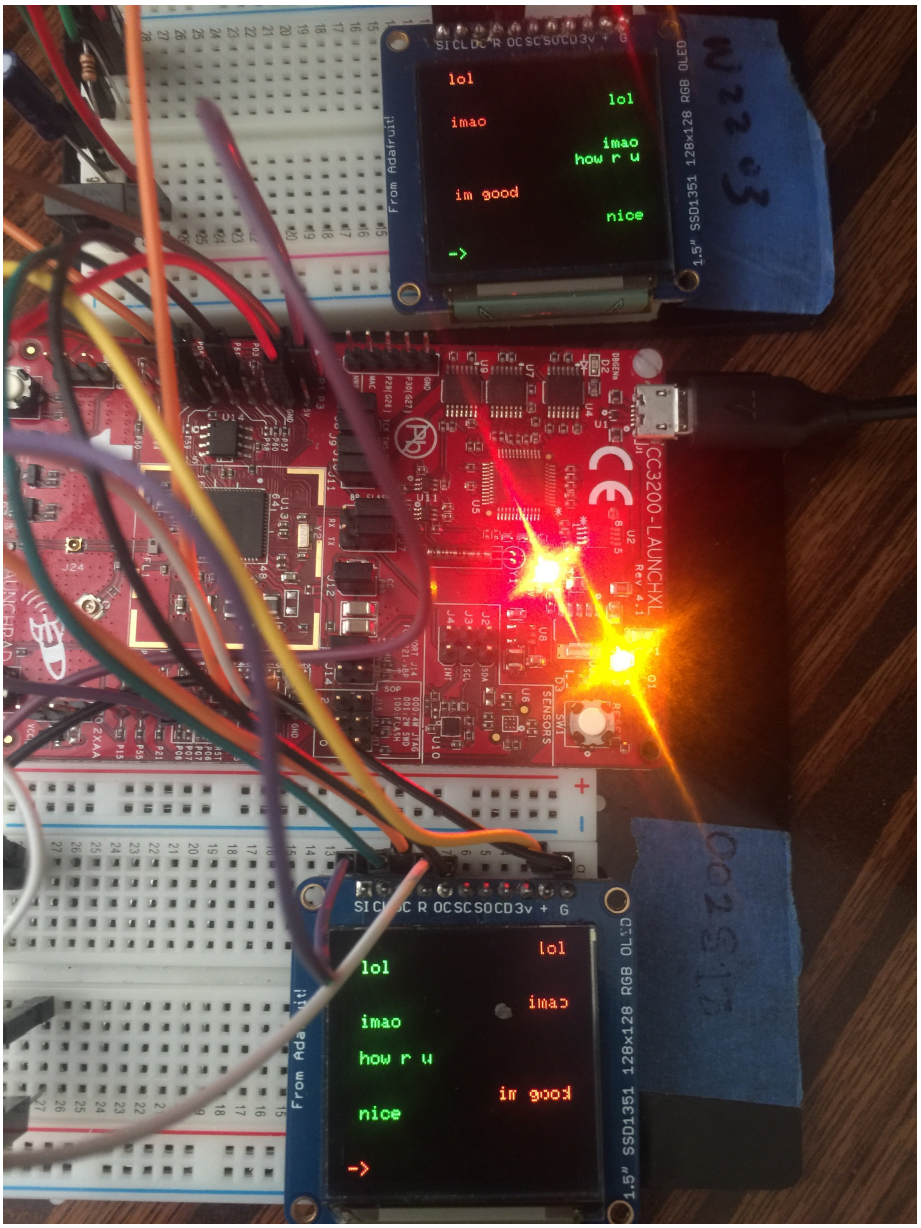
A capture of UART transmission from a message sent on UART1 using Asynchronous Serial protocol analyzer.

12.3 Part III: Board to Board Texting Using Asynchronous Serial Communication (UART)

As you can see, this looks like any other messaging API. The messages you sent are on the right and the messages your friend or colleague sent to you are on the left. Also, your composing message is on the bottom.



I made the board with the green arrow send message "nice" to demonstrate that when there are not enough space on OLED to display the latest message, the OLED detects that and automatically scrolls down the message logs to display the new message.



To show that manual scrolling works, I made the OLED on the top scroll to the upper half of the message log and the OLED on the bottom scroll to the bottom half of the message log.