EEC 172 Lab 3 Report

Alejandro Torres
almtorres@ucdavis.edu

ABSTRACT

In this paper, I discuss how I interfaced an IR receiver module with a remote control to characterize each button's waveform. I then go into detail about how I interfaced the CC3200 LaunchPad with the IR receiver module and an OLED screen to display text messages being composed from the remote control.

I also discuss my learnings when attempting to connect two CC3200 LaunchPads via asynchronous serial communication (UART).

General Terms

UART means Universal Asynchronous Receiver Transmitter. OLED means Color Organic Light-emitting Diode. IR means Infrared Radiation.

1. INTRODUCTION

This laboratory experiment provided a valuable opportunity for me to thoroughly analyze the transmission of the IR module from the remote control. My primary objective was to discern the data format of the signals generated when specific buttons were pressed. By carefully examining the waveform data format, I employed timer and GPIO interrupts on the CC3200 LaunchPad to decode the waveforms accurately, enabling me to determine the specific buttons being pressed.

Once the decoding process was successfully accomplished, I proceeded to establish a connection between the CC3200 LaunchPad and an OLED display. This involved the implementation of relevant functions that facilitated the real-time composition and display of text messages on the upper portion of the OLED screen. In addition, I endeavored to establish a link between two CC3200 LaunchPads via the UART1 channel. Unfortunately, I encountered obstacles in achieving full functionality due to the presence of framing errors within the transmitted signals. Nevertheless, this setback proved to be an instructive experience, as it enhanced my debugging proficiency, which will undoubtedly benefit me in future labs.

2. Procedures for Part 1

In the initial phase of the laboratory experiment, my focus was on characterizing the IR transmissions using a logic analyzer. To achieve this, I began by constructing the necessary circuit for the IR module on a breadboard. My objective was to establish a setup that allowed the IR module to effectively receive signals.

2.1 Capturing and Characterizing IR Transmissions using a Saleae logic

To ensure proper functionality, I took the following steps in building the circuit. First, I connected the voltage source in series with a resistor. Additionally, I added a capacitor in parallel with the voltage and ground source. The inclusion of the capacitor served the purpose of reducing the noise that may be present in the signals transmitted from the remote control.

Following the construction of the circuit and ensuring its proper functioning, my next step was to set up the Saleae Logic analyzer. This device would enable me to detect the data format of the signals being transmitted by the IR module. To establish the connection, I connected channel 1 of the logic analyzer to the output pin of the IR receiver.

With the logic analyzer in place, I proceeded to press various buttons on the remote control, thereby generating signals for analysis (e.g., Figures 1-13). By referencing the provided "Data Format for IR Remotes" document, I was able to interpret the captured signals and identify the specific data format being employed. My analysis revealed that the remote control was utilizing the NEC CODE data format.

The NEC CODE format is characterized by a leader word followed by 16 address bits (comprising 8 non-inverted bits and 8 inverted bits) and 16 data bits (following the same non-inverted and inverted bit structure as the address bits). It is important to note that the address bits remain constant for all buttons pressed, as they represent the unique address of the remote control.

To ensure comprehensive characterization, I tested each button on the remote control, capturing and analyzing the corresponding signals. Figures 1-13 in the lab report depict the signals received from the remote control.

3. Procedures for Part 2

For this section of the laboratory, my aim was to implement a text messaging application program that can communicate asynchronously between two CC3200 LaunchPad devices.

3.1 Decoding IR Transmissions/Application Program

To establish the interface between the IR receiver and the CC3200 LaunchPad, I connected the output of the IR receiver to serve as the input for a GPIO pin, specifically pin 64. This connection allowed me to receive and process the incoming signals from the IR receiver using the LaunchPad.

In order to detect the falling edges of the received signals, I enabled the GPIO pin as an interrupt. Whenever a falling edge was detected, the GPIO handler would be executed, and a falling edge counter would be incremented. This mechanism enabled me to track the occurrence of falling edges and subsequently analyze the pulse lengths in the NEC CODE.

Since the NEC CODE utilizes pulse length coding, there are two distinct burst lengths depending on the bit value being sent. If a "0" bit value was transmitted, the pulse width was approximately 1.125 ms, whereas a "1" bit value corresponded to a pulse width of around 2.25 ms. To accurately determine these pulse widths in the received signals, I utilized a systick timer interrupt. This interrupt allowed me to measure and capture the precise durations of the pulses, enabling me to decode the NEC CODE effectively.

Once I had enabled the interrupts to detect the falling edges and pulse widths, I could proceed with decoding the received signals. To ensure the validity of a signal, I established a condition to detect the pulse width of the leader code, which was approximately 13.5 ms. If the received signal met this condition, I considered it valid and entered a conditional block for further processing.

Within the conditional block, I implemented two crucial functionalities. Firstly, I determined whether the bits being sent were "0" or "1". If a "0" bit was detected, I shifted my decoded value left by 1. Conversely, if a "1" bit was detected, I shifted the decoded value by 1 and added 1 to it. This allowed me to accurately reconstruct the transmitted data into 32 bits from the received signal.

Secondly, the conditional block also controlled the incrementation of the falling edge counter. This counter was crucial in tracking the number of falling edges encountered, helping me keep track of the 32 bits in each signal. Once a valid signal was received and decoded, I reset the falling edge counter, preparing for the next signal.

In the final part of the lab, I implemented a while loop within my main function to ensure the continuous execution of the code. Inside this while loop, I included an internal while loop that would essentially do nothing if the falling edge counter was not equal to 32. This allowed me to wait until a complete signal consisting of 32 bits was received before proceeding further.

Once the falling edge counter reached 32, I broke out of the internal while loop and executed a case statement. The decoded value obtained from the received signal determined the specific case that would be executed. Each case corresponded to a particular button on the remote control.

3.2 Board to Board Texting Using UART

To integrate text messaging, I took the decoded value of the IR signal and associated it with the appropriate button. Depending on which button was pressed, it would either select a character, erase a character, send a character, or change the font color. All of these operations were broken up into separate functions.

My implementation for selecting a character involves calling a function when the user selects any number from 0-9, excluding 1. When a letter was pressed, the function would be called, and the button that had been pressed would be evaluated by the switch statement. The cases within the switch statement determined which letter to set. It determined which letter to set by analyzing the previous letter.

When the current letter was chosen, it was not actually selected until the user pressed the button '+'. This was my way of confirming that the current letter was the one I actually wanted to select. For example, if I wanted to access the letter 'c', I would press the button '2' three times and then press the button '+' to select the current letter. After this was determined, the letter was sent by calling the send message function.

The send message function was constructed by calling the UART functions, storing the characters into the buffer, and calling the print message function. The message string within the program was then cleared so the next message sent would not hold the previous characters.

The print message function outputted the characters with a distance of 7 pixels between the start of each character, so they wouldn't overlap each other. When the end of the row was reached, I moved the x-axis back to 0 and moved the y-axis down by 7 pixels. Each time a character was selected, it was printed to the board.

4. Problems and Solutions

I encountered several challenges during the lab and implemented solutions to address them.

4.1 Decoding IR Transmissions/Application Program

The main issue with this part was debugging the GPIO and Systick timer handler. Initially, I added print statements inside the handlers to see what was getting returned. However, I soon realized that this approach was not effective because the interrupts were operating at a higher frequency than the print statements, leading to inaccurate results. To overcome this, I adopted watch variables to accurately analyze the returned decoded values.

I also faced an inconsistency in the returned decoded values. While I initially achieved decoding of the signals, my application would not function correctly. To address this issue, I implemented an additional condition to detect the leader code at the beginning of each signal, ensuring that only valid signals were considered for decoding.

4.2 Board to Board Texting Using UART

I had issues when selecting characters, where the IR receiver would not receive the correct button information. To address this, I added the button '+' to confirm the character selection, ensuring that the proper character was being selected.

I initially encountered issues with communicating between the boards, as the message would not print properly. After troubleshooting, I identified that the source of the issues was the pins used for communication. Changing the pins from 55 and 57 to 1 and 2 improved the situation, but I still encountered framing errors when transmitting between devices. However, the message would display on the transmitter's screen. Further troubleshooting and adjustments were needed to resolve this issue.

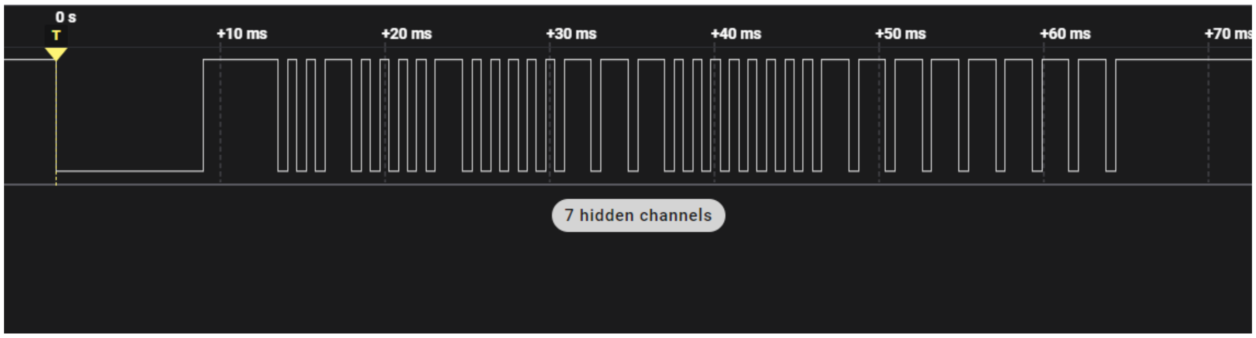
0 s
T
+10 ms
+20 ms
+30 ms
+40 ms
+50 ms
+60 ms
+70 ms
7 hidden channels

Figure 1: Button Waveform for the “0” button being pressed

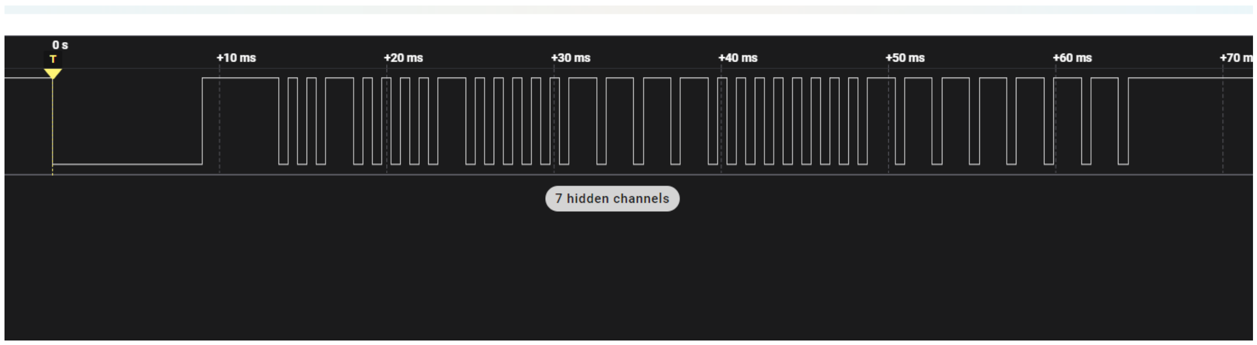

Figure 2: Button Waveform for the “1” button being pressed

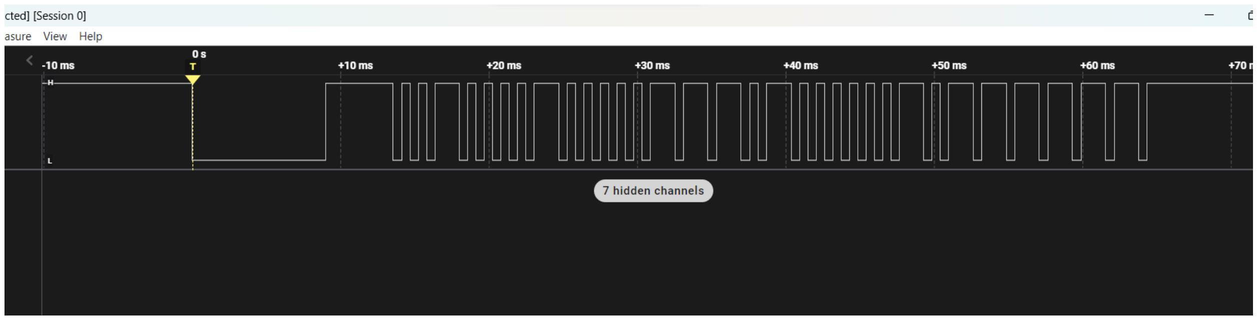

Figure 3: Button Waveform for the “2” button being pressed

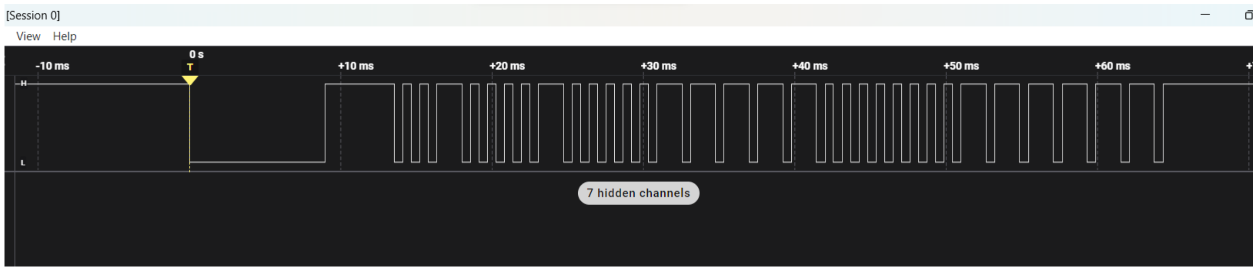

Figure 4: Button Waveform for the “3” button being pressed

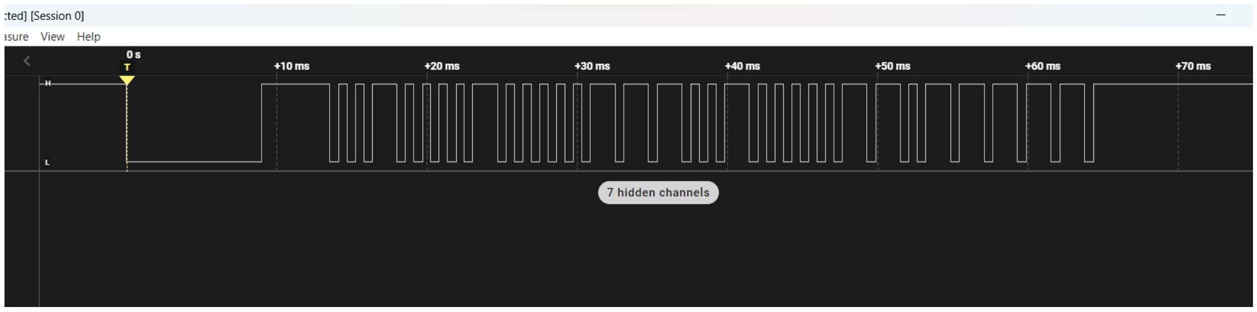

Figure 5: Button Waveform for the “4” button being pressed

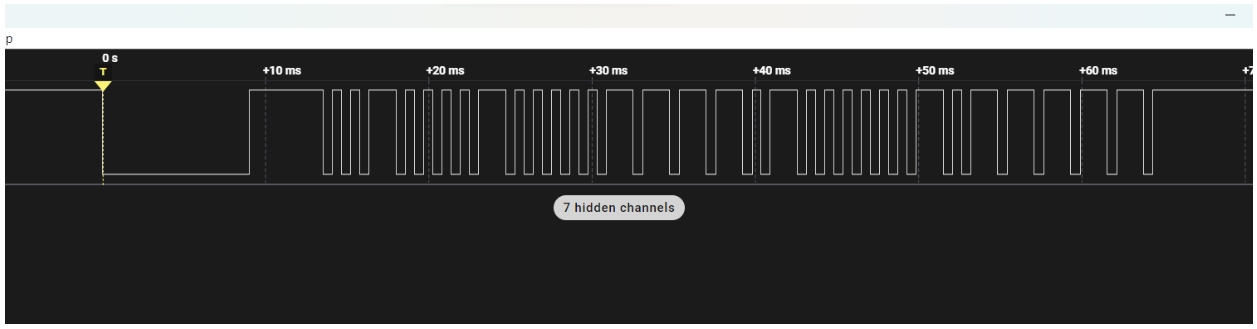

Figure 6: Button Waveform for the “5” button being pressed

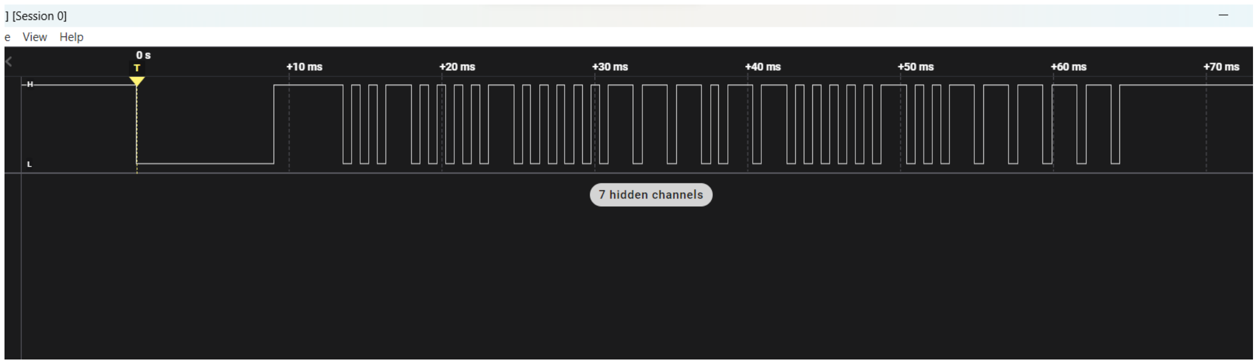

Figure 7: Button Waveform for the “6” button being pressed

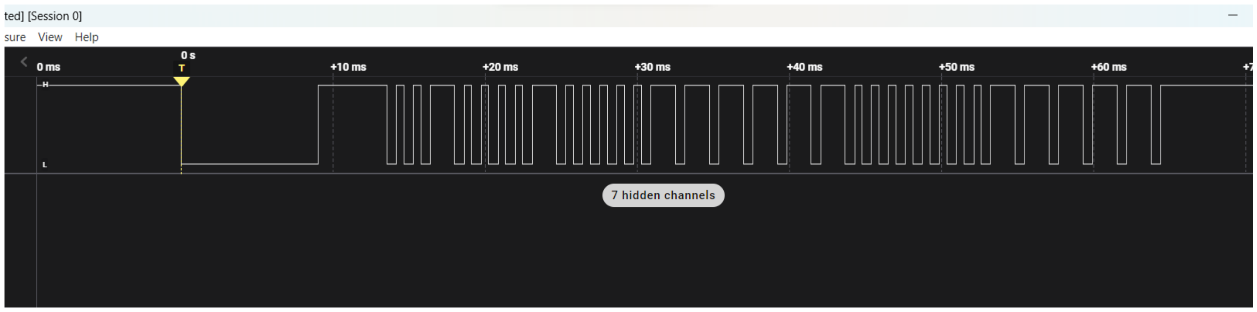

Figure 8: Button Waveform for the “7” button being pressed

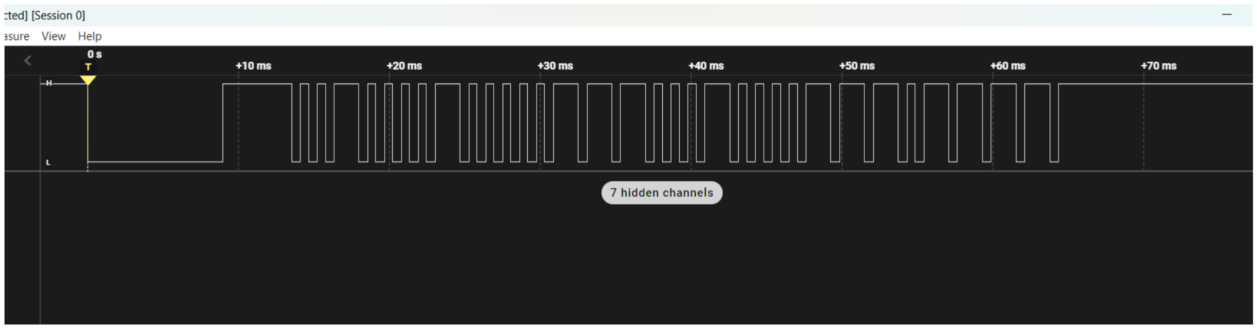

Figure 9: Button Waveform for the “8” button being pressed

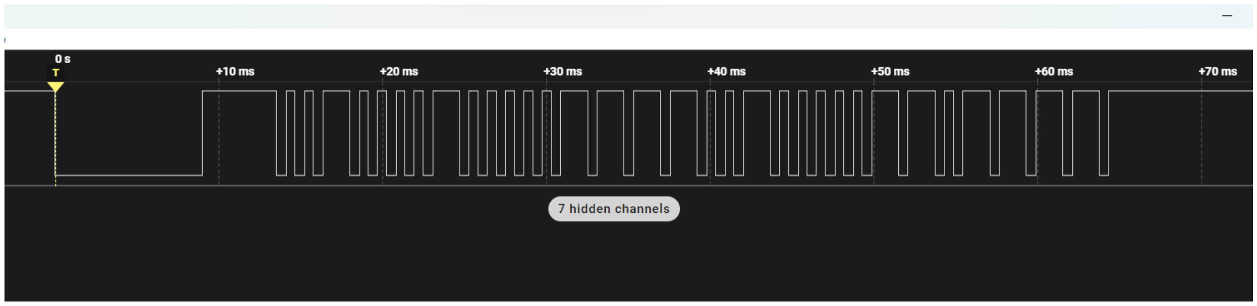

Figure 10: Button Waveform for the “9” button being pressed

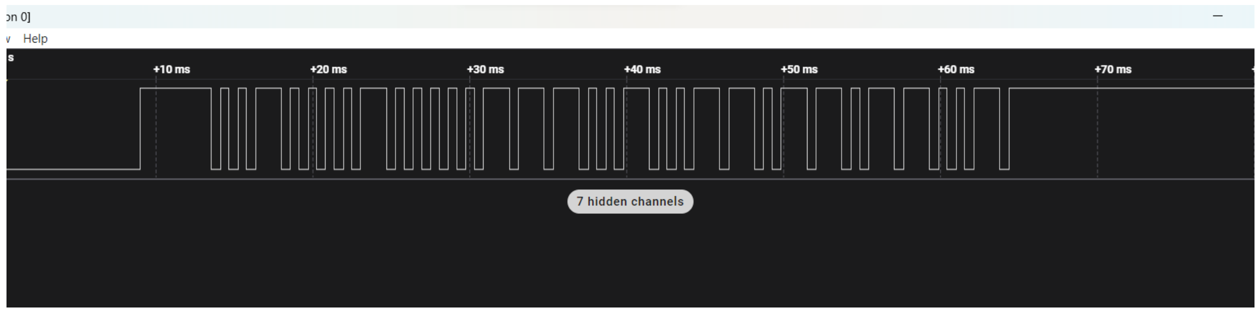

Figure 11: Button Waveform for the “MUTE” button being pressed

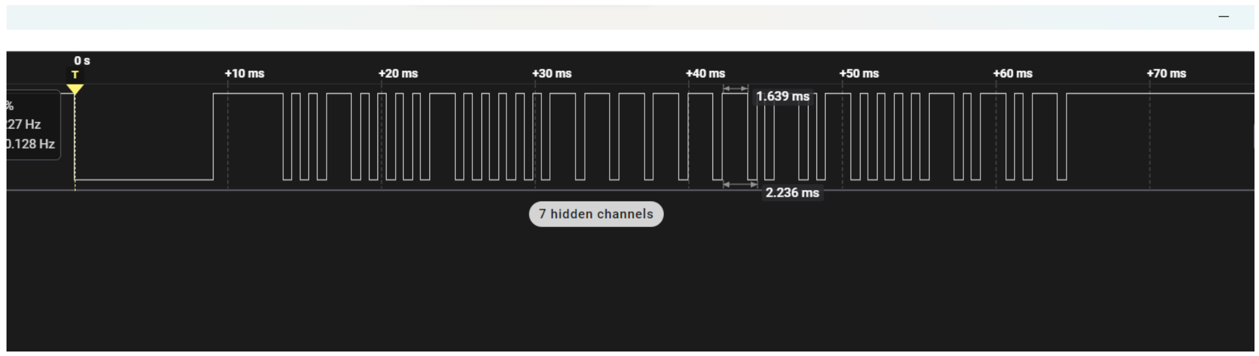

Figure 12: Button Waveform for the “LAST” button being pressed

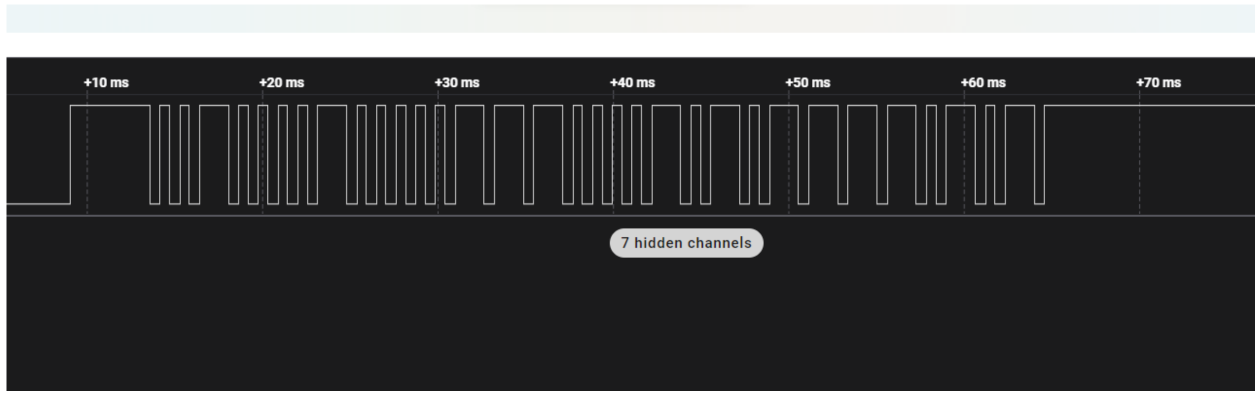

Figure 13: Button Waveform for the “+ Channel” button being pressed

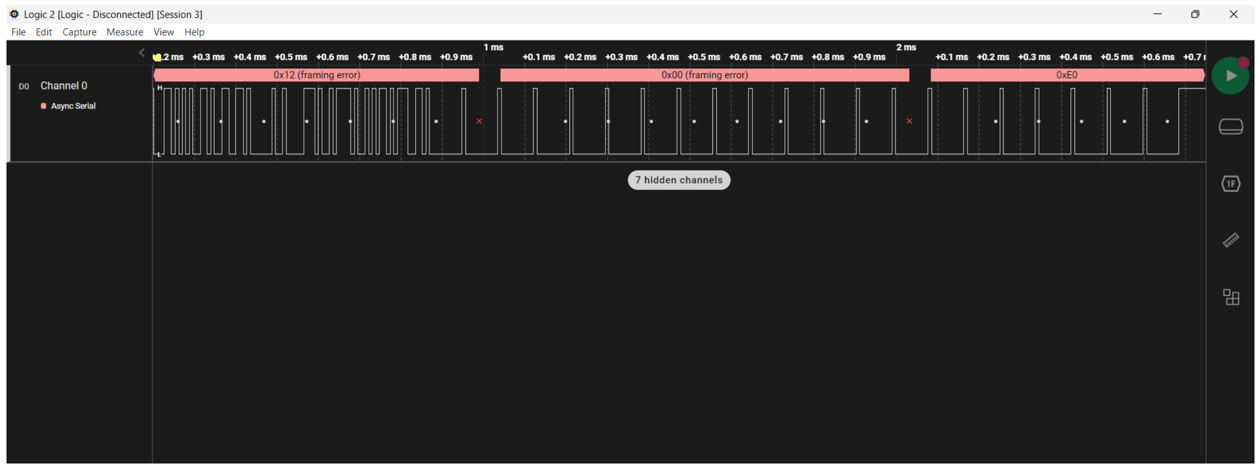

Figure 14: UART1 Transmission framing error