Mazar Farran

Jacob Ley

EEC 172, Section A03

February 9, 2016

Lab 3 Report

**Objective:**

The goal of this lab was to reverse engineer the infrared (IR) signals sent by the AT&T UVerse Remote and program the CC3200 Launchpad to decode these signals and use them to form messages. These messages were then sent over UART to the attached OLED of another CC3200.

**Background:**

**TI CC3200** – System on a chip with an ARM Cortex M4 Processor and I/O support via GPIO, UART, and SPI.

**Adafruit OLED** – Organic Light Emitting Diode display, connected to the CC3200 via SPI.

**GPIO** – General Purpose Input/Output. Individual pins can assert or detect voltage HIGH or LOW for various purposes.

**SPI** – Serial Peripheral Interface. Used to connect the OLED to the CC3200.

**UART** – Universal Asynchronous Receiver/Transmitter. Used for inter-CC3200 communication.

**AT&T UVerse Remote** – Remote with various modes that sends IR signals when buttons are pressed.

**IR Receiver** – Detects IR Signals at around 37KHz and outputs corresponding digital signals.

**Methods:  
 PART I: Capturing and Characterizing IR Signals**

1. The Infrared receiver and universal remote were set up as described in the lab manual.
2. The Salae Logic Analyzer was connected the receiver’s output. The signals were read in the logic analyzer tool at 500MHz for approximately 10 seconds. This was enough time to manually press buttons and have the signals appear in the program.
3. After analyzing the signals, it was determined that data was represented as gaps between LOW pulses. Logical 0 was a small gap, 1 was a large gap. Actual data was preceded by a ‘lead pulse’ as well as an address section. (Remote code 1028)
4. Signals for all digits, all arrow buttons, the OK button, and the MUTE button were captured and deciphered by hand (screenshots provided).
5. The receiver output was connected to a GPIO Input (Pin 58, 1.2).
6. GPIO interrupts were enabled in software using *GPIOIntEnable()* and *GPIOIntRegister()*
7. The peripheral clock SysTick was enabled using SysTickEnable(). A timer interrupt was set up to account for when the clock wrapped.
8. After a lead bit was detected, the GPIO interrupt would fire, setting a flag that would make the main loop parse the rest of the signal.
9. The function *getbit()* measured the time between edge transitions (using SysTickGetValue()) and determined whether the bit was a 1, a 0, or invalid by looking at the elapsed time. If invalid, the parsing function was restarted.
10. After a signal was successfully parsed, its character representation was printed to the console.

**PART II: Board to Board Texting**

1. With the board successfully parsing buttons, all that remained was to implement character cycling.
   1. When a digit is pressed, its ascii representation is stored in a buffer.
   2. If the same button is pressed within a certain number of timer interrupts, the ascii character changes to the next one in the cycle (e.g. 2→a→b→c). This is handled by the *nextchar()* function.
   3. If the next button pressed is different, or enough time has passed, the buffer moved ahead by 1.
2. Messages are sent after the OK button is pressed.
   1. The message is sent character by character via UART. There is a built in delay between sent characters that prevents a UART buffer overflow.
   2. The buffer is then cleared.
3. Messages are received asynchronously via a UART interrupt. Characters are read from the UART buffer and printed to the OLED.
4. **Extra functionality was also added**. The left arrow button deletes the previous character. Volume+/- buttons change the current character casing. Channel +/- buttons cycle through font colors.

**Difficulties:**

The distance between IR pulses varied widely. It was necessary to specify a tolerance range in order to decode the bits. Rarely, this results in the wrong signal being deciphered.

The GPIO interrupt, unlike the UART and SPI interrupts, needs to be reset every time it is triggered. This caused a great deal of confusion.

Because both CC3200 devices are sensitive to the same remote, only one could have a receiver at a time when communicating. Otherwise, both would send and receive the exact same messages at the exact same time.

**Conclusion:**

We were able to successfully decode the IR signals and use them to send messages across CC3200s. We also added some extra functionality such as changing colors and capitalization. This lab showed that a great deal of care is necessary when using physical inputs in a digital program. Tolerances, error checking, and more need to be accounted for.