


Team Member 1: Kunal Pai
Team Member 2: Steven To
Section Number/TA: A03

Demonstrate IR Decoding:		
Date	Signature	Comment
4/27/22	Ku/Pi	

Demonstrate Texting:		
Date	Signature	Comment
4/27/22	Ku/Pi	Only touch-typing works. No sending. Color changing affects the whole string, not the characters.
04/27/2022		everything works!

Contribution:

Steven did the circuits and the implementation of touch typing, including the timeouts and multi-button taps. Kunal did the signal detection, including GPIO and Timer interrupts and UART interrupts. Both partners were present throughout the duration of the lab, and did the initial signal testing via the Saleae Logic Analyzer together.

Introduction:

The objective of this lab was to use the Saleae logic analyzer to decode the waveforms produced by specific button presses from an AT&T IR remote control. These waveforms were produced from an IR receiver module which had to be oriented appropriately on a breadboard alongside a resistor, a capacitor, and voltage and ground sources. Using these waveforms, we developed a program that interfaced the CC3200 launchpad with the IR receiver module so that text messages can be sent

to the OLED depending on the buttons pressed using a multi-tap text entry system. The OLED itself was interfaced with the CC3200 launchpad via SPI, similar to Lab 2. As an additional touch to this lab, we had to use UART to send text messages back and forth between two CC3200 launchpads.

Background:

SPI is a serial communication protocol that allows the transmission of data between a controller and one or more peripheral devices. It involves the use of four data lines: MOSI which allows the controller to transmit data to the peripheral, MISO which allows the peripheral to transmit data to the controller, SCLK which transmits the clock signal from controller to peripheral, and CS which allows the controller to select which peripheral to communicate with.

UART is a serial and asynchronous communication protocol that allows for bidirectional transmission of data between two devices. For data to be transmitted, it is sent through the Tx pin from one device which is then received by the Rx pin of another device. Because UART is asynchronous, it does not require a clock to synchronize the transmission of data bits. Instead, it depends on start and stop bits to indicate when to start reading data and when to stop. The rate of data transfer is based on the baud rate which must be the same for both devices.

The OLED is a 1.5" display consisting of 128x128 RGB pixels, with each pixel having 16 bits of resolution to allow for high contrast and a wide range of vivid colors. It uses a SSD1351 driver chip that manages the display through SPI. Adafruit also

provides the OLED with an API of functions that can be used to display patterns and text (Because it was originally written for Arduino, it needs to be ported to the CC3200 through the modification of low-level functions).

The Saleae logic analyzer is a tool that allows for the debugging and verification of waveforms and signals of digital circuits and systems. This is especially useful for ensuring that the data bits transmitted for SPI and I²C are correct.

The AT&T S10-S3 Remote Control is a device that produces IR waves that can be picked up by the IR receiver module. These IR waves vary based on a 4-digit device code that can be configured along with channel, volume, and backlight settings.

The Vishay TSOP31336 is a device that can pick up IR signals and produce waveforms depending on the signals. For this to work, the IR receiver has to be configured on a breadboard with a resistor, a capacitor, a voltage source, and ground.

Goals:

Part 1

- To learn how to wire the IR receiver module correctly with a resistor, capacitor, voltage source, and ground
- To configure the AT&T IR remote with the appropriate settings for lab use
- To capture the waveforms produced by the IR receiver module using the Saleae logic analyzer
- To characterize the waveforms produced by the IR receiver module

with their respective remote button presses

Part 2

- To interface the CC3200 launchpad with the IR receiver module using appropriate pin configurations
- To develop software that decodes the signals of the IR receiver module and matches them with the correct remote button presses
 - GPIO interrupts will be used to detect the rising and falling edges of the waveform
 - One of the CC3200 launchpad's timers will be used to determine the pulse widths corresponding to either a 1 or 0

Part 3

- To interface the CC3200 launchpads with OLED displays using SPI
- To connect two CC3200 launchpads using UART for bidirectional data transmission
- To develop an application that allows the IR remote control to compose text messages on the OLED that can be sent back and forth between two CC3200 launchpads
 - The application must support multi-tap texting, text color change, spaces, and character deletion
 - The transmission of text messages will be achieved through UART interrupts
 - Incoming messages will appear at the bottom of the OLED, while messages in

composition will appear at the top

- To capture and analyze a UART waveform produced by a transmitted message using the Saleae logic analyzer

Methodology:

Part 1:

For part 1, we arranged the circuit as provided in the lab manual. Then, we used the Saleae Logic Analyzer to obtain the waveform on pressing every button. The waveform we obtained showed the following trends: the 0s had a shorter time period than the 1s, with a constant distance between two bits of the waveform. We observed the 0s to last for about 0.54 ms and the 1s to last for about 1.64 ms. Every button press was 32 bits in length. This was the data we obtained for each button:

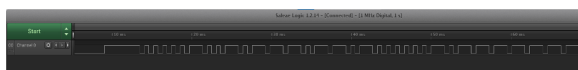
Waveform for 0:



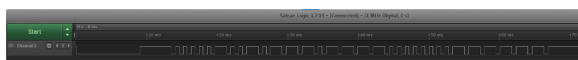
Waveform for 1:



Waveform for 2:



Waveform for 3:



Waveform for 4:



Waveform for 5:



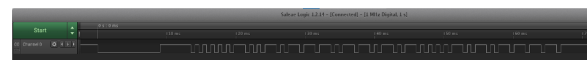
Waveform for 6:



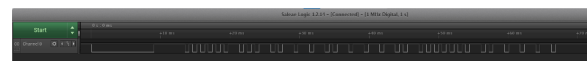
Waveform for 7:



Waveform for 8:



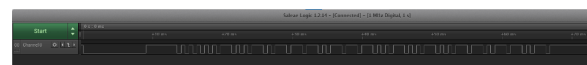
Waveform for 9:



Waveform for MUTE:



Waveform for LAST:



0 - 0000 0100 1011 1011 0001 1000 1110 0111
 1 - 0000 0100 1011 1011 1000 0000 0111 1111

```

2 - 0000 0100 1011 1011 0100 0000 1011
1111
3 - 0000 0100 1011 1011 1100 0000 0011
1111
4 - 0000 0100 1011 1011 1010 0000 0101
1111
5 - 0000 0100 1011 1011 0110 0000 1001
1111
6 - 0000 0100 1011 1011 1110 0000 0001
1111
7 - 0000 0100 1011 1011 1001 0000 0110
1111
8 - 0000 0100 1011 1011 0101 0000 1010
1111
9 - 0000 0100 1011 1011 1101 0000 0010
1111
MUTE - 0000 0100 1011 1011 0010 0010
1101 1101
LAST - 0000 0100 1011 1011 1001 0010
0110 1101

```

We defined these bit sequences as macros in our implementation.

To interpret the waveforms, we created a TimerInterrupt, using “timer_if.c” that triggers every 0.5 ms. On triggering, it adds 1 to a counter we are running to measure the number of 0.5 ms intervals. We have also registered the GPIO interrupts on pin 8, such that it triggers on both rising and falling edges. On a rising edge, it resets the counter of the number of intervals to 0, and enables the timer interrupts. On a falling edge, it does not do anything unless the number of intervals measured is greater than or equal to 8, which corresponds to a time of 4 ms, and is the beginning of the signal. Therefore, on every falling edge, we compute the number of intervals. Since 1.64 can be rounded to 3 times 0.5, if the number of intervals are greater than or equal to 3, we concatenated a one to the resultant signal, else a zero. If the bit length is 32,

that signified the end of the signal, and on comparing the resultant string with the macros, we obtained the signal.

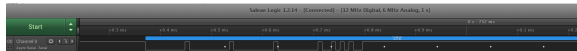
Part 2:

For part 2, we built upon the implementation of part 1, and used the Adafruit files to draw characters on the OLED. Firstly, we initialized all the clocks and the peripheral for SPI communication. Then, we defined global strings of the input sequence for every button. For example, “abc” for button 2, “def” for button 3 and so on. After that, we changed the function that decodes the input. We added a variable that stores the previous input and another that marks the index of the input sequence string. If the previous input and the current input are the same, the index increases by 1, which outputs the next character in the input sequence. To prevent out of bounds errors, we used a modulo arithmetic function so that it remains within the bounds of the string sequence. For the colors, we created a 2D array of unsigned integers using the color macros defined in the Adafruit files, and cycle through them when 1 is pressed, similar to the logic behind the button presses. The current color is stored in another variable and applied whenever printing of the character is supposed to happen. If a zero is pressed, since it is a space, we just concatenated a whitespace to the resultant string, and for delete, we moved back an index and changed the character to a null escape sequence (“\0”). We also have a for loop running through, and if another button is not pressed within it, the character is locked in and the user can enter the next character.

For the UART communication, we initialized the pins for UART1 RX and TX and

initialized the UART. We created an interrupt for UART that keeps accepting characters using the UARTCharGet function until they are available. Once it reaches a terminator, which we defined as '/0', it changes a flag variable to show that the message has been received. Similarly, on pressing LAST, a flag variable is changed to indicate that a message needs to get sent. While sending the message, we used the UARTCharPut function, which generates the interrupt in the first place. On the respective flags, a half of the screen is filled to black, and the message is either sent via UART or is printed out into the OLED.

Waveform for UART transmission:



Discussion:

For our initial implementation, we were using systick functions to get the length of every pulse but the values of those pulse widths were very inconsistent. For example, the pulse width of a zero ranged from 33 to 60000 and of a one ranged from 50000 to over 100000. This was too wide and overlapping of a range to distinguish between the two bits, and so, we chose to use timer interrupts for this lab.

Another challenge we faced while implementing part 2 of the lab was the color change. We did not implement the color change in such a way that a string of text could have more than one color at a time on the OLED, but we changed that by introducing a current color variable, which

changes with the press of 'one' and using that variable to print the text along with the color. Another issue we faced was that whenever we pressed space, the resultant string would print twice. We realized this was because we did not clear the screen and set the cursor back to (0,0). On doing that, we fixed the issue and spaces started working like they were supposed to.

While implementing texting, we also had an inexplicable error with the OLED which was solved by resetting the board and running the code again. Here is a picture of the error:



Certain optimizations to our current algorithm would be in the texting system itself. Currently, we do not have an out of bounds error check and thus, the texts that can be sent are limited in length. We could implement a variable that checks if the x-coordinate is out of bounds, and if so, moves down by one row. This would be similar to the bounds checking we did for

the previous lab. This would also tie in with the second optimization, which is to make the texting faster. We used `fillScreen(BLACK)`; initially whenever there was a new character entered. However, this is very slow, and we realized that we could use the `drawChar` function to enter characters quickly. For the delete function, we could just draw a black character at the previous index. Since this function also takes in x and y coordinates, it would tie in with our first optimization better.

For the UART, one of the errors we got was that the boards did not communicate, nor did the interrupt generate on the other board on putting the characters in through UART. We fixed this error by initializing UART1 RX and TX in the pinmux configuration file, and then changing the pins used for it. We used pins 58 and 59, but since there were two instances of those pins, using the other pins labeled 58 and 59 made the application run perfectly.

The UART waveform we obtained seems to correspond with the brief text message we sent from one CC3200 launchpad to another. The instances of brief high pulses represent data bits of 1 being sent, whereas long periods of low signal represent data bits of 0 being sent.

Conclusion:

In this lab, we gained exposure to IR wave detection, and used interrupts (GPIO, Timer and UART) to detect the signals, interpret them, and send them from one board to the other. We also learned more about UART in general, and how to initialize communication using its API. We used interrupts and the `Timer_If` API for the first time. We were

exposed to using the SPI API again to communicate with the OLED. We also used the same OLED interface we built in the previous lab to achieve what we wanted in this lab.