UNIVERSITY OF CALIFORNIA, DAVIS

Department of Electrical and Computer Engineering

EEC 172 Spring 2023

Team Member 1: Thomas Ke

Team Member 2: Bradley Manzo

Section Number/TA: A05 / Begum Kasap

Demonstrate IR decoding:		
Date	Signature	Comments
5/4	[illegible]	✓

Demonstrate Texting:		
Date	Signature	Comments
5/9	[illegible]	Minor Errors: • Color not sent correctly over UART • when looping through letters error (A→B→C→B) A missing. • Cannot display more than 1 space • Can delete on same line, but not on previous line if multiple lines of text. +Bonus: 2-way Communication works. Main functionality done.

Implementati	**/70**	**Report**	**/30**	**Total**
Functionality	/55	Thoroughness	/10	
Robustness	/10	Clarity	/10	
Presentation	/5	Presentation	/5	
		Code Clarity\Commentary	/5	

EEC 172 Lab Three

Bradley Manzo
Computer Engineering B.A.
916953788
brmanzo@ucdavis.edu

Thomas Ke
Computer Engineering B.A.
917513004
thke@ucdavis.edu

1. INTRODUCTION

This lab introduces the IR Receiver Module to characterize the transmissions from an AT&T IR Remote Control. The signal of the IR Receiver Module can then be monitored, and individual buttons interpreted into different characters and functions in order to transmit texts between CC3200 boards to display on the OLED screen

2. GOALS AND TASKS

Completion of the first part of the lab, Capturing and Characterizing IR Transmissions using a Saleae logic, requires creating the desired circuit, powering the IR receiver, and receiving the output through GPIO in order to capture the waveform. We then characterize the signal for each button to properly interpret them.

For the second part, Decoding IR Transmissions/ Application Program, we were required to create an application decodes the IR transmissions from the respective buttons to letters using triple-tap text, and other functions such as space, backspace, font color change, and newline.

In the final part, Board to Board Texting Using Asynchronous Serial Communication (UART), we were required to create a circuit setup that would allow bidirectional communication between our two CC3200 Launchpads as well as an application/program that would facilitate such communication. In addition, we were to receive and decipher the input from the AT&T remote controls as well as

3. CONCEPTS AND UNDERSTANDING

3.1 Part I

In order to properly decode and characterize the incoming signals from the AT&T remote to the IR receiver and into our CC3200, we first must understand the data encoding format. The remote outputs an IR signal in the NEC format, specifying a leading code with a burst of 9 ms followed by a pause of 4.5 ms, and finally the data word. If a button is being held, rather than the word being retransmitted, only the leader code and a single bit are sent out at an interval of 104 ms. The data word is comprised of four bytes: Address, Inverted Address, Command, and Inverted Command. The data word is constant for each button, so we chose to interpret the entire 32 bits rather than breaking it into its individual.

3.2 Part II

The second part of the lab required us to understand how the SysTick library functions work as well as how to configure interrupts and develop interrupt handlers. SysTick allows us to properly time the intervals between IR signals by counting clock cycles for a continuous period following each SysTickReset(). The SysTick register can only count to 40 ms of clock cycles before it is reset to zero, and SysTick count is incremented. By subtracting the current time retrieved using SysTickValueGet(), from the previous time SysTick count x SYSTICK_RELOAD_VAL, we can record delays greater than 40 ms. This delay is recorded within our GPIOA0IntHandler function which is configured to handle interrupts on Pin 59. The program busy-waits within main until a flag is set indicating an interrupt was asserted. That flag is asserted within the int handler function, breaking busy-wait and entering the signal decoding portion of the program. The final new concept of Part II is the triple-tap functionality of the remote. Certain number keys on the remote correspond to a letter depending on how many times the button was repeated. Interpreting the proper character requires proper handling of printing to the OLED, and understanding of the NEC repeated signal format. The understanding necessary to operate the OLED remains the same as Lab 2.

3.3 Part III

The final part of this lab requires a deeper understanding of UART in order to properly implement the board-to-board texting interface. Configuring the UART Base to A1 allows for UART to be configured to pins on the board rather than exclusively through the micro-USB port. By connecting the transmitting pin on the sender to the receiving pin on the receiving board and vice versa, we are able to establish a connection. An interrupt handler is necessary to implement this behavior in order to prevent the loss of data. The handler is similar to the GPIO Handler, however when receiving data, the incoming string is read character by character onto a receiving buffer which is then passed to the main function. The interrupt flag is asserted which allows the program to stop busy-waiting and interpret the incoming string.

4. METHODS

4.1 Part I

Decoding the incoming signal from the AT&T remote required constructing the circuit according to the lab manual's specifications and verifying the behavior using the Saleae Logic Analyzer. Once we configured our remotes to the proper frequency, we captured the signal from all 12 required buttons and represented them as binary. This documentation will be attached to this lab report. We then measured the times between falling edges and decided upon the necessary margins for logical "0" and "1", as well as the period that we ignore incoming noise. Since using the UART Report() leads to inconsistencies with the SysTick functions, we opted to report only once the 32 bits had arrived, and we found success by setting the bounds to be exactly what we measured in Saleae rather than the reported values. The decoded word is checked directly against the binary representations of each button before recording the button number.

4.2 Part II

The next step to enabling the sending of texts is to interpret the numbers as letters and functions depending on the number of repetitions. For example, button 2 corresponds to A,B, or C depending on if you push the button once, twice, or three times respectively within a certain range of time. We chose this time to be .15 seconds, and we implemented this logic within our code by waiting for possible repetitions before printing out to the OLED. Each button press within 0.15 seconds checks if the new character was the same as the last, and increments the character to be printed appropriately. To print to the OLED, we made use of the functions we wrote in Lab 2 to write data and commands, and we configured it in the same way as well. When initializing the board, we set the cursor to the top left of the bottom half of the screen and set the font size and color to white with a black background. We increment the x position of the character by 6 pixels each time we print, and if at the edge of the screen, x is set to 0 and y is incremented by 8 pixels. Deleting was implemented by drawing a black rectangle the size of a character at the previous position and then decrementing the x position by 6 pixels. If x is 0, then y is decremented by 8 pixels, and x position is set to the right edge of the screen. The font color is changed by incrementing the index within an array of the font macros and then setting the font color using that value. Newline, a temporary feature was implemented by setting the x position to zero and incrementing y by 8. Space was simply printing the ‘ ‘ character.

4.3 Part III

Implementing the requirements of part 3 required a complete refactoring of the code to allow for a second type of interrupt on the UART. The UART interrupt handler is similar in structure to the GPIO interrupt handler, however instead of stopping the SysTick timer, it receives the incoming chars from the UART FIFO onto a receiving buffer that it passes to the main function to be interpreted. Because we must send the data as a string, or an array of chars, we must insert the char into an array as well as print it on the bottom half of the screen. Furthermore, when deleting, we insert a null-terminating character ‘\0’ into the array at the current position as well as visually erasing the letter. Received texts are then printed at the top of the screen instead of at the bottom with the same logic as before, (setting x to zero and incrementing y by 8 when on the screen’s edge). Once we configured UART properly and flashed the program to both boards, they were able to send and receive the strings printing them to the desired sides of the OLED!

5. Discussion of Challenges

The first challenge we had was interpreting the timing of our logical “1”s and “0”s as reporting the delays to UART was messing up the SysTick Timing. We coded it for the observed times and found success. Another issue we encountered was properly getting the repetitions to work after refactoring our code so many times. Some issues we ran into were the backspace not working, the font color not being associated properly with the test, the repetitions not rolling back to the first character, and multiple spaces not being printed. The font color could have been fixed by redoing our data format to include the color of each character, the spaces could have been fixed by not ignoring repeats, and the roll back could have been achieved with more logic to reset the index to zero after 3. The backspace glitch was never resolved and was extremely hard to identify.

6. Contribution Breakdown

Both of us worked on Part I, Bradley completed Part II and Part III. Bradley decoded the signals into binary, researched the background topics, developed the program, and implemented the solution onto both CC3200s. Thomas helped with the triple-tap function of decoding in Part II.

7. Conclusion

After completing the lab, we feel like we have gained a greater understanding of how interrupts can be used to break the regular flow of execution to receive data at unexpected times. Additionally, it ensures that no data is lost by pausing the regular flow of execution. Receiving a logical signal from a peripheral circuit and learning how to decode a signal from a mundane tool like a TV remote was fascinating. Being able to use our functions for the OLED to develop an application was also satisfying. This lab was rewarding for both of us and overall invigorated more interest into this class.