EEC 172 Lab 3: IR Remote Control Testing Over an Asynchronous Serial (UART) Link I Section: A03

Manprit Heer
912839204
mkheer@ucdavis.edu

Cathy Hsieh
912753571
cchsieh@ucdavis.edu

OBJECTIVE

In this lab, we utilize the Saleae logic analyzer and an IR receiver module to characterize transmissions and decode the buttons into numbers. Our remote control has a specific TV code that will then be used to decode our unique waveforms. We will then use the launchpad to the IR receiver module to write a program that uses interrupts to look out for signals from the IR, which will show in the form of a simple button-press from the remote. Lastly we will then use the remote control to compose text messages using the multi-map text entry system to send text messages back and forth between two CC3200 LaunchPad boards over an asynchronous communication channel.

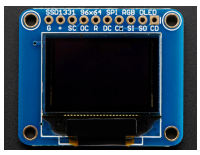
First, we connect up the IR Receiver Module interface using the diagram displayed below. Ground and voltage are connected respectively to ground and voltage of the launchpad, while the OUT pin will output the waveforms received by the remote control. After this, we configure the remote control so that it is set back to its factory defaults as well as the TV Device code whose IR transmission encoding will vary. This will ensure minimal interference from other remote interrupts in the lab area.

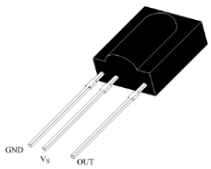
After the set up is complete, the first phase is to decode the waveforms (displayed on the Saleae logic analyzer) and print the respective digits (0-9) on the console. This must be done by recording the time between each pulse, or rising edge, and using this time elapsed to mark the pulse as a 0 or a 1. Our 24-bit waveforms, in 0s and 1s, would then have a 1-on-1 encoding with a number. Along with the number keys, we also install DELETE and ENTER for the application program. The application program is a texting interface across two Launchpads, which we construct by incorporating a UART interrupt system into the OLED-LaunchPad-IR-System. Messages are then composed and sent to the other device, and vice versa.

MATERIALS AND METHODS

The equipment needed for this lab are listed as follows:

1. CC3200 LaunchPad (CC3200-LAUNCHXL)
2. USB Micro-B plug to USB-A plug Cable
3. Adafruit OLED Breakout Board[1]
4. Saleae USB Logic Analyzer
5. AT&T S10-S3 Remote Control
6. Vishay TSOP31336 or 1236 or 31236



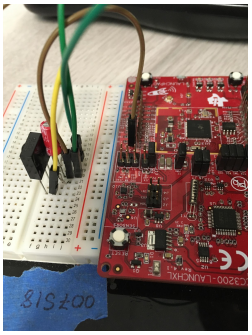


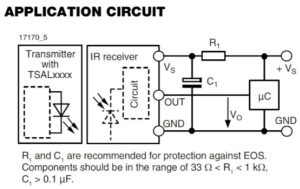
[1] https://www.adafruit.com/product/684?gclid=Cj0KCQiAsbrxBRDpARIsAAnnz_PATckp_fWf0mom4VPLEF7osaKOk78D4VWOKufqBx_EhoA3Zs9L6vQaAkC5EALw_wcB

PROCEDURE

Part I. Capturing and Characterizing IR Transmissions using a Saleae Logic

In order to start part one, we must follow the proper circuit diagram to hook up the IR Transmission. The circuit diagram below was implemented on the bread board to produce the following:





Initially, we had trouble interpreting the correct positions of the resistor and capacitor due to our own forgetfulness, but we soon able to place it correctly to receive a signal in the Saleae Analyzer. Another problem we came across while simulating the waveform was that the analyzer refused to pick up any of the waveforms at first and we would get garbage values. These values would appear like triangles and simply put, white noise.

To combat this, we went back to ensure that we had a properly-working circuit diagram and then proceeded to re-configure our remote controls. After reconfiguring them, we started receiving proper waveforms on the analyzer.

```
127
128     ul5tatus = MAP_GPIOIntStatus (pin61.port, true);
129     MAP_GPIOIntClear(pin61.port, ul5tatus);        // clear interrupts on GPIOA1
130     pin61_intcount++;
131     pin61_intflag=1;
132
133     gpioTimeCount = g_ulTimerInts;
134
135     if(gpioTimeCount > 2 && gpioTimeCount <= 5){
136         gpioTimeCount = 1;
137         str = '1';
138     }
139     else if(gpioTimeCount <3){
140         gpioTimeCount = 0;
141         str = '0';
142     }
143     else{
144         str = '-';
145     }
146
147     if(strlen(gpioArr) < 26){
148         strncat(gpioArr, &str, 1);
149     }
150
151     g_ulTimerInts = 0;
152 }
```

After getting the proper waveforms, we then move on to the software segment of this part by using timers and counters to receive and decode the proper 0s and 1s depicted but the LOWs and HIGHs of the waveforms, respectively.

PART I | IR INTERRUPT HANDLER + DECODER OF WAVEFORMS

Part II. Decoding IR Transmission/Application Program

Decoding IR Transmission

PART II | KEY “1” ON REMOTE CONTROL

PART II | KEY “2” ON REMOTE CONTROL

PART II | KEY “3” ON REMOTE CONTROL

PART II | KEY “4” ON REMOTE CONTROL

PART II | KEY “5” ON REMOTE CONTROL

PART II | KEY “6” ON REMOTE CONTROL

PART II | KEY “7” ON REMOTE CONTROL

PART II | KEY “8” ON REMOTE CONTROL

PART II | KEY “9” ON REMOTE CONTROL

PART II | KEY “0” ON REMOTE CONTROL

PART II | KEY “ENTER” ON REMOTE CONTROL

PART II | KEY “DELETE” ON REMOTE CONTROL

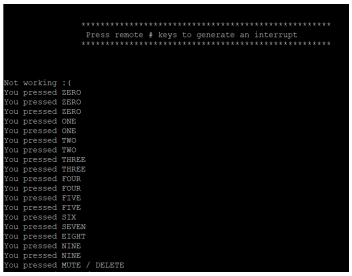
PART II | KEY “MUTE” ON REMOTE CONTROL

PART II | KEY “LAST” ON REMOTE CONTROL

```
const char * ZERO = "11111110001000000011101";
const char * ONE = "11111100010000000011011";
const char * TWO = "11111110010000000011011";
const char * THREE = "11111101010000000101011";
const char * FOUR = "11111111010000000001011";
const char * FIVE = "11111100110000000110011";
const char * SIX  = "11111110110000000010011";
const char * SEVEN = "11111101110000000100011";
const char * EIGHT = "11111111110000000000011";
const char * NINE  = "11111100001000000111101";
const char * MUTE  = "11111110100000000010111";
const char * LAST  = "11111100100000000110111";
const char * POWER= "11111110000000000011111";
const char * UP    = "11111101010100000101010";
const char * LEFT = "11111100110100000110010";
const char * DOWN = "11111111010100000001010";
const char * RIGHT= "11111110110100000010010";
const char * MENU = "11111111011000000001001";
const char * OK   = "11111111000100000001110";
```

PART II | EACH WAVEFORM IN BINARY FORM

After decoding the waveform, we attach it to the key that we had clicked, as demonstrated above. Each waveform has a distinct set of 0s and 1s and represents a certain value. The next step is to print out the string to ensure that this encoding works and makes sense.



PART II | EACH NUMBER DISPLAYED ON CONSOLE (REPORT) + OLED (PUTCHAR)

Board-to-Board Texting Using Asynchronous Serial Communication (UART)

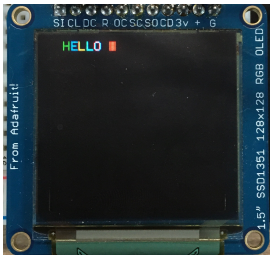
When we first started to implement the board-to-board multi-map texting format, we had issues printing the letters to the board. Specifically, our putchar() function was set in the wrong location. After fixing that and changing 0-9 to letters, we were able to implement the color change function and make it work. Another problem we struggled with at this part was the fact that the timing was a little bit off with the letters as each interrupt happened. For example, when we we hit "A' and wanted to hit 'F" next, the A would disappear and the F would take its place (this would only occur once in a while). Another bug we had discovered, which we figure has stemmed from one root cause, is that sometimes when wee are hitting "F" the previous letter is printed twice, instead of once. After much troubleshooting, we stemmed back the source of the problem to be at the way the waveform was being saved into our buffer.

```
132
133     gpioTimeCount = g_ulTimerInts;
134
135     if(gpioTimeCount > 2 && gpioTimeCount <= 5){
136         gpioTimeCount = 1;
137         str = '1';
138     }
139     else if(gpioTimeCount <3){
140         gpioTimeCount = 0;
141         str = '0';
142     }
143     else{
144         str = '-';
145     }
146
147     if(strlen(gpioArr) < 26){
148         strncat(gpioArr, &str, 1);
149     }
```

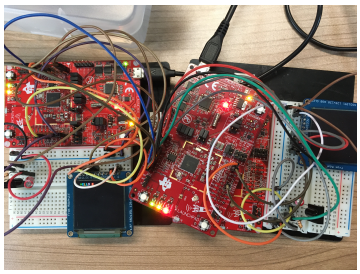
PART II | CONCATENATING GPIOARR WITH EACH COMPONENT OF EACH WAVEFORM

As the above code as a reference, we realized that since each waveform was being recorded twice (thanks to the Saleae logic analyzer), the gpioArr string was also taking in the second waveform of the previous character after it had been printed out. The interrupt should have been disabled during this process, yet the amount of troubleshooting we did around this process simply would not accommodate for that very essential detail. Another troubleshooting method we tried was to clear the array after we reached a certain amount of bits, since we knew that one character's waveform is only 24 bits in the way our IR Remote was formatted. After clearing and printing out the array, we were stuck with an array that would take in the next half of the same waveform, despite us clearing it out. This led us to believe that we were not clearing it in the right location, and the GPIOHandler was not the right place for this action. We then move it into main(), specifically right after we print out the character that was printed, to no avail. We tried a quite a few different coding techniques to this same affect, but the bug was still present, and it continued to slightly effect the performance of our system.

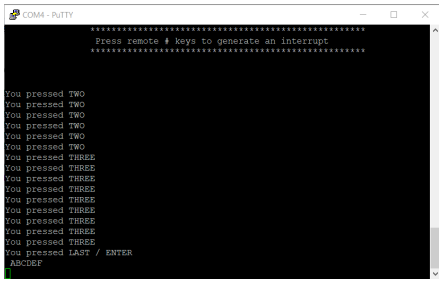
PART II | ONE KEY PRODUCES TWO WAVEFORMS



PART II | HELLO - SIX DIFFERENT COLORS



PART II | TWO LAUNCHPADS + OLEDS CONNECTED VIA PINS, UART1 CONNECTED VIA PIN1 + PIN 2



PART II | CONSOLE PRINTS OUT MESSAGE FROM OLED

SUMMARY

Although we were able to work well with the UART and GPIO interrupt protocol, we failed to take into account the precision needed for the timing and could not figure out the bug that was messing with our OLED output from time to time. Asynchronous multi-systems require a flow in-between systems and a smooth interaction in between interrupts. As we proceeded to troubleshoot throughout the lab, we found that a deep understanding of interrupts as well as the double waveforms that result as a by-product is required in order to perfect the timing and outputs of the interrupted values. The timing of the waveform input data was not in our favor, however we were still able to decode, print out, and utilize the multiple interrupts to interact between devices. Whether that be from launchpad to OLED, or OLED to OLED , or IR to OLED, all these connections in this lab prove to us that pretty much anything is possible with the right understanding of protocols and interrupts. Real-time state changes would make use of these interrupts to ensure the fastest response time for certain activities. In this lab, we were able to demonstrate our understanding of interrupts and use it to build a texting program all with the addition of a universal IR remote.