CECS 447- Embedded System III

Project 2 Report - UART and Bluetooth

By

Kai Fillipow & Yamin Yee

November 18th, 2019

Table of Contents

Introduction

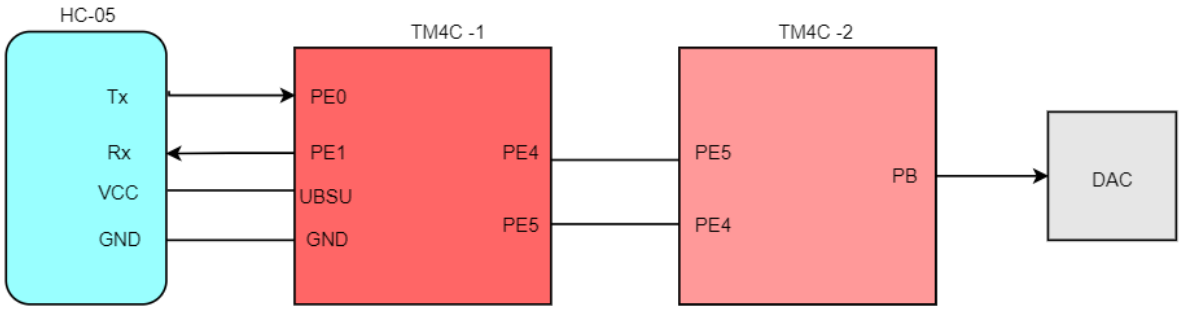
The goal of this project was to control the frequency of a sine wave and the brightness of an LED by sending Bluetooth commands via a Bluetooth terminal app on an Android device. Two TM4C Launchpads were utilized; one to handle the commands received via Bluetooth and the other to update the sine wave frequency or the LED brightness. The sine wave will be produced using the same hardware and logic from the previous project.

Operation

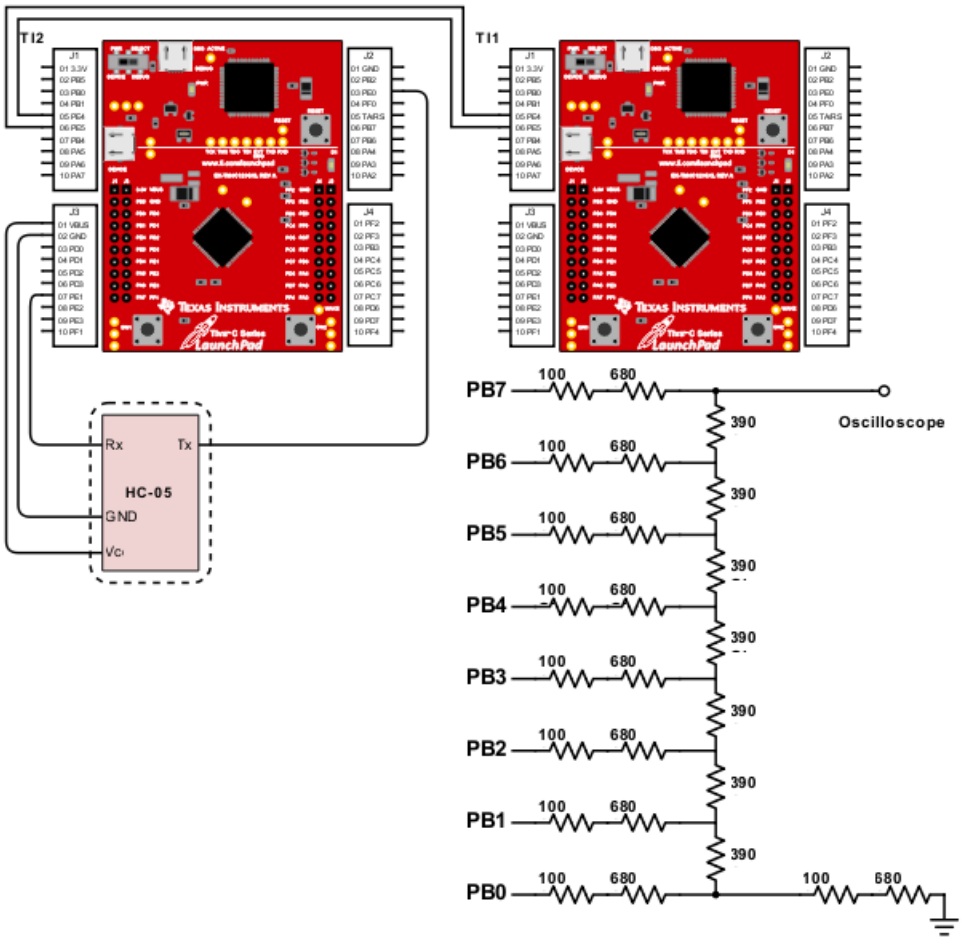
The user will interface will the microcontrollers using a Bluetooth terminal app on an Android device. Once the HC-05 module and the terminal app are connected, the user can then send commands to change the sine wave frequency or LED brightness. For the sine wave frequency, the user must first type an ‘f’, then follow it with a value from 262 to 494. To change the brightness, the user must first type a ‘b’, then follow it with a value from 0-255. When this data is sent, it will have the ASCII value for carriage return attached to it for the microcontroller to properly identify the end of the command. Once the microcontroller receives the data, it checks to make sure the value is within the specified range, otherwise it will not update the frequency or brightness. If the value is within the range, the first microcontroller will send the data to the microcontroller responsible for changing the frequency and brightness. The user will then be able to see the new sine wave frequency on an oscilloscope or the changed LED on the second microcontroller.

Hardware

Block Diagram



Schematic


TI2
TI1
Rx
Tx
HC-05
GND
Vcc
PB7
PB6
PB5
PB4
PB3
PB2
PB1
PB0
100
680
390
Oscilloscope

Components

- 7 x 390 Ω resistors
- 9 x 680 Ω resistors
- 9 x 100 Ω resistors
- 2 TM4Cs
    - 1 for Bluetooth and 1 for DAC
- HC-05
- Android Smartphone

Component Justification

We chose these set of resistors for a few reasons. We did not have the necessary resistors on hand to make a binary weighted DAC, so we found a combination of resistors that would give us R and 2R. These specific values were chosen because they were the only combination of resistors we to make R and 2R that were on the lower end of resistance from what we had available. We also used 2 TM4Cs : one for DAC and one for Bluetooth. The data will be sent to TM4C_1 and then TM4C_1 will send that data to TM4C_2. TM4C_2 will decode the message and then output the correct DAC output. We only use Android Smartphone since our iPhones don’t pair with the HC-05.

Software

Software Approach

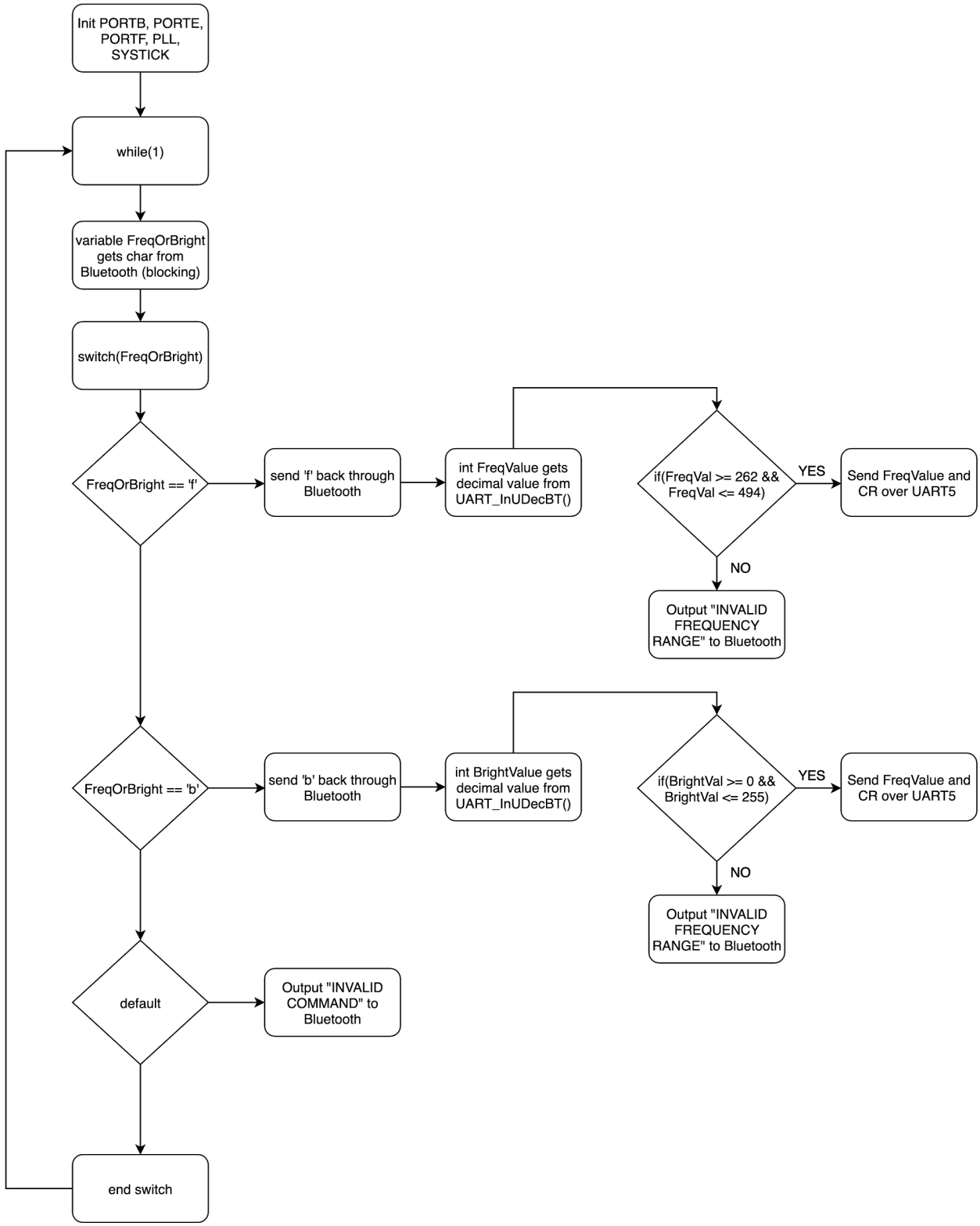
To keep the microcontroller that is controlling the sine wave frequency and LED brightness from doing additional work, we had the first microcontroller perform the checks to make sure that a valid command was received and that the number specified was in the range for its respective command. If it finds that the range or command is invalid, it displays a message on the Bluetooth terminal, doesn’t update any values, and

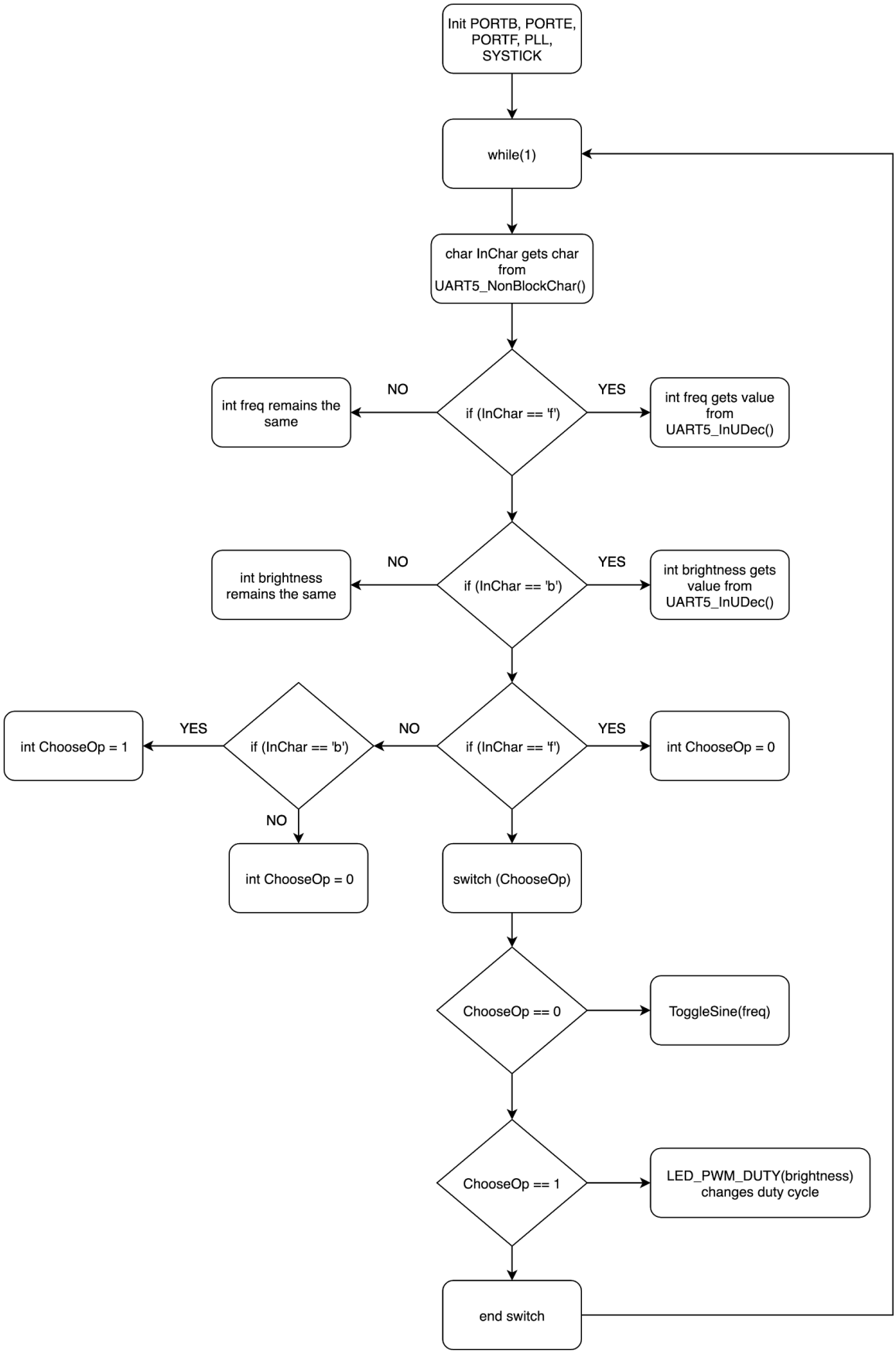
then waits for the next command. When the command and range are valid, it sends the data along UART5 to the next microcontroller.

Once the next microcontroller receives the data, it will either update the values for the sine wave frequency or the LED brightness. The logic for the sine wave is the same from our previous project. As it stands, the function to output a sine wave must be run in a super loop, so we needed to change the LED brightness from a function that doesn't have to be constantly run. We then configured the PWM module on PF2 to change its brightness. Upon reset, the LED PWM has a duty cycle of 100%. Using a period of 40,000, when a new value for the brightness is received, it will multiply the number by 156.25 because $\frac{40{,}000}{256} = 156.25$. This will allow the received number to control the duty cycle appropriately.

In the previous project, the sine wave frequency was controlled by the value from the ADC. Since the value we receive is between 262 and 494, there is no need to convert the ADC value to a frequency anymore. The new load value for SysTick is calculated by using the formula $\frac{80{,}000{,}000}{freq} * \frac{1}{150}$. In the super loop, the ToggleSine() function will run on every iteration, whereas the PWM_LED_DUTY() function will only run when a brightness level is specified.

Software Flowchart




Init PORTB, PORTE, PORTF, PLL, SYSTICK
while(1)
char InChar gets char from UART5_NonBlockChar()
if (InChar == 'f')
NO
int freq remains the same
YES
int freq gets value from UART5_InUDec()
if (InChar == 'b')
NO
int brightness remains the same
YES
int brightness gets value from UART5_InUDec()
if (InChar == 'f')
NO
if (InChar == 'b')
YES
int ChooseOp = 1
NO
int ChooseOp = 0
YES
int ChooseOp = 0
switch (ChooseOp)
ChooseOp == 0
ToggleSine(freq)
ChooseOp == 1
LED_PWM_DUTY(brightness) changes duty cycle
end switch

Conclusion

Overall, this project went well. After setting up the UARTs and their necessary functions to send and receive data, most of our debugging involved making sure the microcontroller was receiving the correct values from the HC-05 module. While debugging, we could see that the command received from the Bluetooth terminal was correct, but it would get stuck on receiving the decimal value. After going through our code again, we realized that the Bluetooth terminal was sending both a carriage return and line feed upon pressing send. Our code was dependent on getting a single carriage return, so when it received a line feed before the carriage return, it would then misinterpret the decimal value. We then fixed this by having the Bluetooth terminal only send a carriage return upon pressing send.

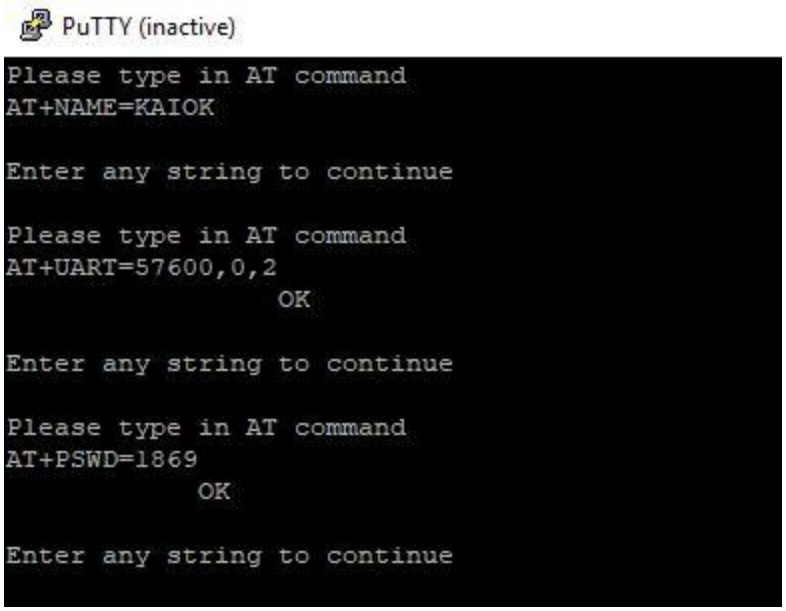
The other problem we ran into was setting up the PWM for the LED correctly. Using example code and information from the datasheet, we believe that we set it up correctly, but the LED would never turn on. Upon looking at previous lecture notes, we realized that we had been using the wrong PWM number and generator. We then corrected the names, numbers, and values of the appropriate PWM registers and upon reprogramming the microcontroller, we found that it finally lit up.

The last problem that we faced that we did not end up optimizing was our error in the difference between the frequency we input and the frequency we saw on the oscilloscope. When we specified 262Hz, the frequency we saw was around 253Hz. When we specified 494Hz, the frequency we saw was about 465Hz. So for just the specified range of frequencies for the sine wave, our margin of error was between 9Hz to 29Hz. One approach to this could have been to offset the frequency input by 9, which would then reduce to error to 0Hz to 20Hz. The other solution would have been to offset the value and update our formula that calculates the SysTick reload to decrease the SysTick reload value. Our sine frequency is always slower than specified, so by reducing the reload value, we would increase the frequency of the sine wave.

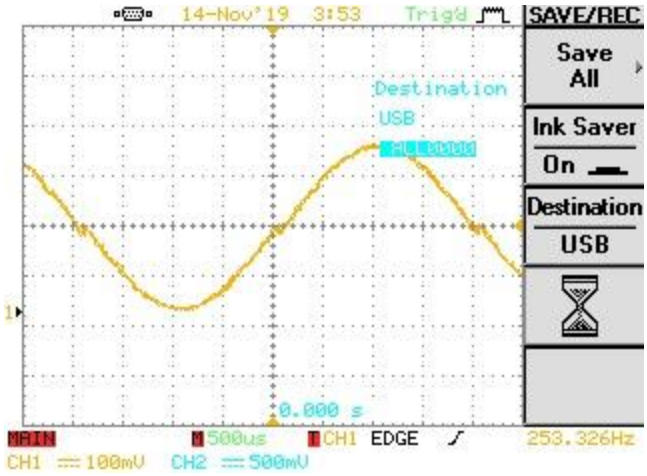
This project was helpful because not only did it make us get familiar with using UART on microcontrollers, but it made us review on how to use PWM again. Since UARTs are the simplest implementation of serial communication, it's likely that we will use them again in the future. Although there are more types of serial communication, getting comfortable with one is a good start before learning the other types.

Figures

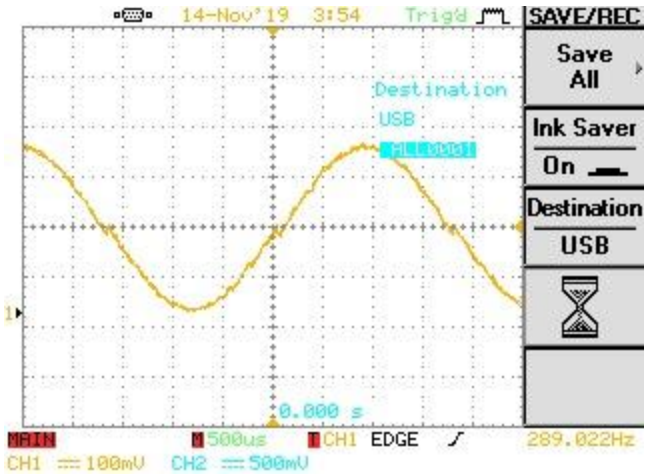
Configuring HC-05 via terminal



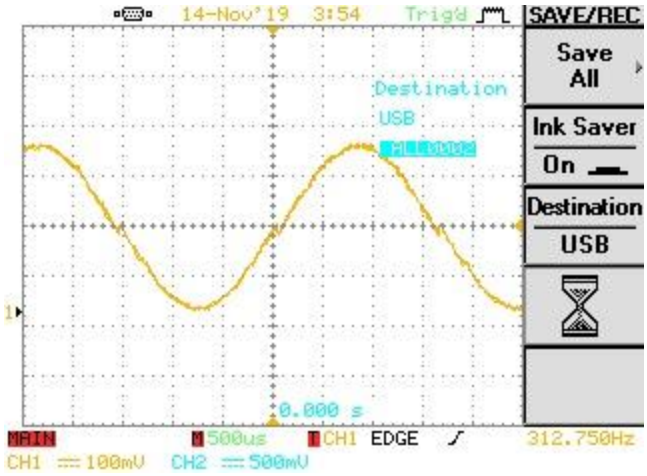
Waveform 1: 262Hz Specified



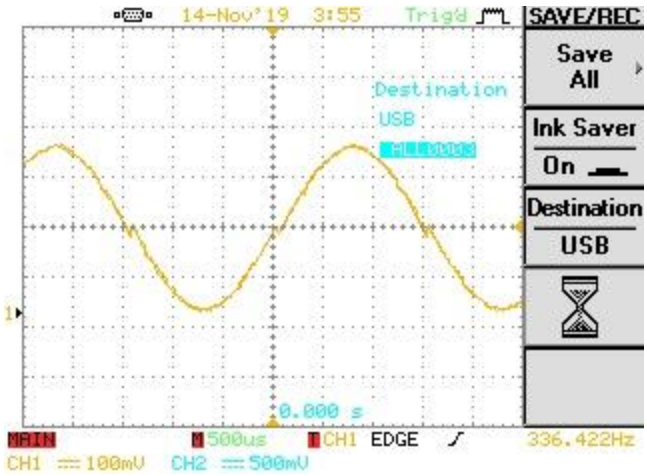
Waveform 2: 300Hz Specified



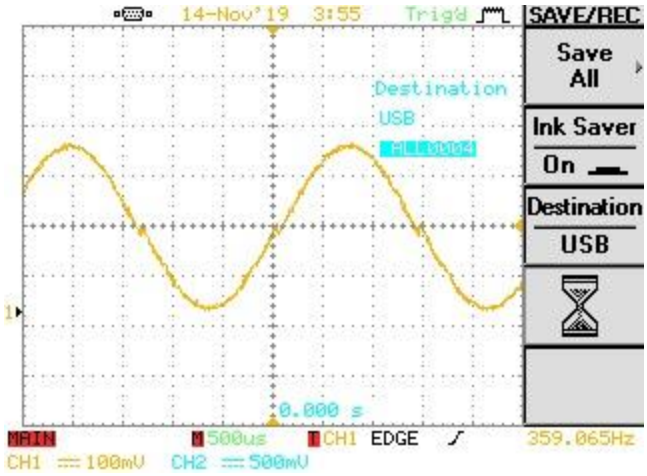
Waveform 3: 325Hz Specified



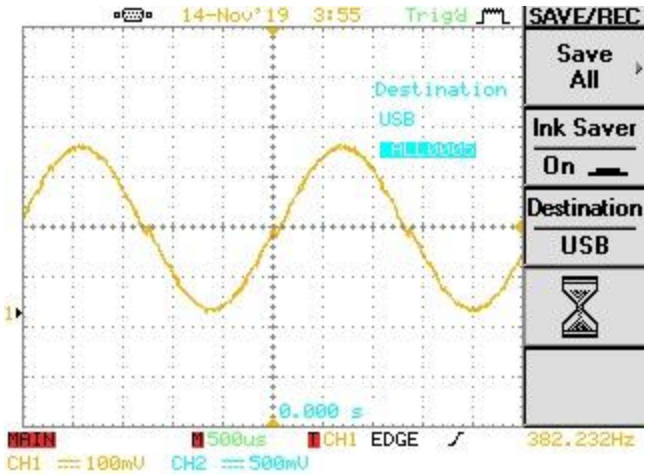
Waveform 4: 350Hz Specified



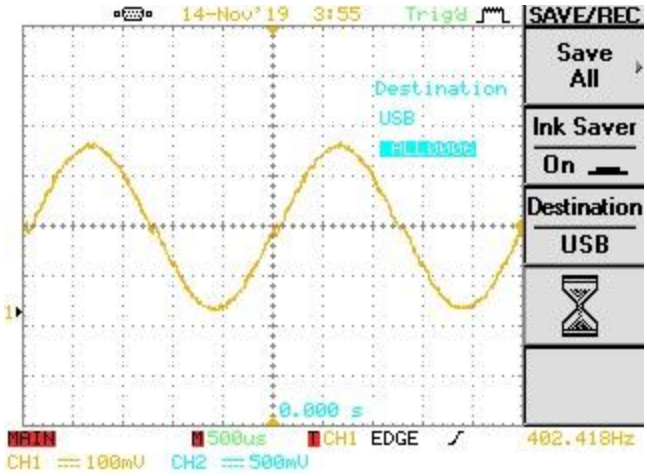
Waveform 5: 375Hz Specified



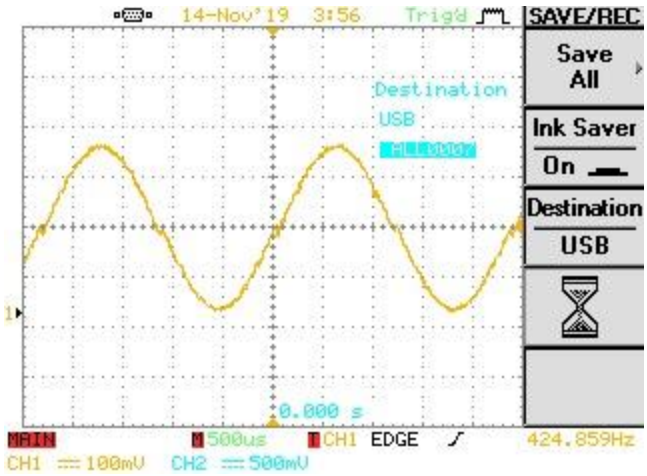
Waveform 6: 400Hz Specified



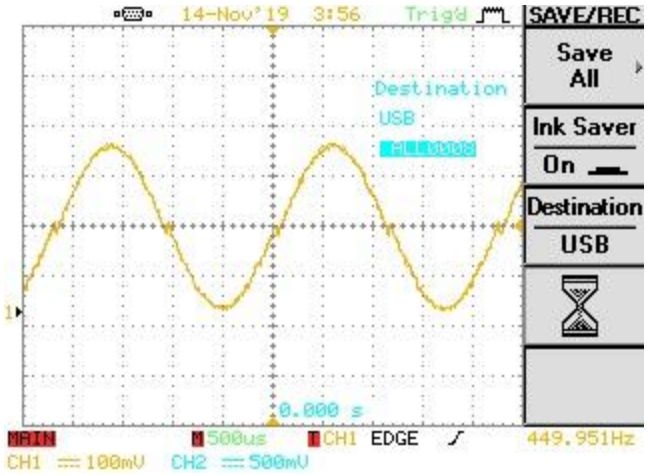
Waveform 7: 425Hz Specified



Waveform 8: 450Hz Specified



Waveform 9: 475Hz Specified



Waveform 10: 494Hz Specified