Zachery Takkesh - 015656509
Justin Maeder - 015906629

CECS 447 – Embedded Systems III

Final Project: Bluetooth UART Infrared controlled 8-bit DAC - Tone Generator

12/9/2019

Table of Contents

Table of Figures

1. Introduction

1.1. Project Description

The objective of this project is to implement multiple Universal Asynchronous Receiver/Transmitter's for communication between two Tiva-C TM4C123GH microcontrollers. One microcontroller will receive data via HM-10 Bluetooth chip, decode the input stream and send necessary data to the second microcontroller. The second microcontroller will be designed using the previous project for creating a tone generator using an 8-bit Digital-to-Analog Converter (DAC).

The purpose of the project is to demonstrate knowledge of Bluetooth, UART communication, and multiple MCU development, while using knowledge of the ARM processor, GPIO, Digital to Analog Converter, Analog to Digital Converter, timers and interrupts from the previous project. The final deliverable of project 2 is to send a block of data from a Bluetooth Terminal mobile application to MCU1, decode the data in MCU1, send a data frame to MCU2 and control the DAC network or the LCD display using the stream of data. The result of the transmitted data from MCU1 to MCU2 will either play a generated tone on the speaker or display an animation on the LCD screen. An amplifier is integrated into the DAC network to turn smaller signals into larger ones to be able to hear and test.

1.2. Project Requirements

1. MCU1 will serially communicate with a Serial Profile Bluetooth device to determine the commands to send out via modulating the IR LED. The Table below is the list of commands.

Character	Function	Character	Function
'0'	Command0	'4'	Command4
'1'	Command1	'5'	Command5
'2'	Command2	'6'	Command6
'3'	Command3	'7'	Command7

Figure 1 MCU1 Bluetooth Command Table

2. Push Button 1 will determine the device number on both microcontrollers. On reset Device0 should be active by default. The following LED colors will indicate what is the current device mode on MCU1 and MCU2:

- Red LED: Device 0
- Green LED: Device 1
- Blue LED: Device 2
- White LED: Device 3

3. IR Signal must demonstrate the ability to transmit wireless data with a distance greater than 8ft.

4. Once, Requirements 1-3 are completed, connect the output of the IR LED to channel 1 and the output of the IR receiver to the channel 2 of the oscilloscope to confirm the IR signal. Pick at least 4 or 5 commands to take a screenshot.

5. MCU2 will decode the demodulated IR data it received and do the following based on the received command:

Function	Description of Function
Command0	Display animation 0 on color LCD
Command1	Display animation 1 on color LCD
Command2	Display animation 2 on color LCD
Command3	Display animation 3 on color LCD
Command4	Output a 262Hz Sine Wave
Command5	Output a 262Hz Square Wave
Command6	Output a 262Hz Triangle Wave
Command7	Output a 262Hz Sawtooth Wave

Figure 2 MCU2 Command Description Table

2. Operation

The first part of the final project is creating an 8-bit DAC, and we choose R2R logic to do so. First, we created a schematic of our DAC and built it on a breadboard. Next, we created a 256-bit truth table of DAC with its respective output. To do this, we manually entered in all possibilities from D0 to D7 for 0 - 255 with logic '0' being 0V, and logic '1' being 3.3V. We then used the equation:
Vout =SUM((D0/256)+(D1/128)+(D2/64)+(D3/32)+(D4/16)+(D5/8)+(D6/4)+(D7/2)) for the 8-bit DAC, and created a graph to show the linear representation of the 8-bit DAC. Next, we created a sine table for n = [0,255] with the equation = $1.65 * \sin(2 * 3.14 * (n/256)) + 1.65$. Lastly, we tested the output of the DAC and confirmed the correct outputs of the truth table.

The second part of the final project is setting up 8 commands for the second microcontroller to perform. The first 4 will include different animations outputted on the ST7735 LCD. The next four commands include generating sine, square, triangle, and sawtooth waves using a fixed frequency of 262Hz with the DAC created from part 1. First, we set up GPIO PB0-7 as output for 8-bit DAC D0-D7 respectively. Next, we wrote a function for sine wave oscillating at our desired frequency of 262Hz. We utilized an infrared LED and receiver to switch between the commands. This will be explained in further detail in the software section. To test that the sine wave formula generated-values were correct, we set up 8 LEDs each one connected to PB0-7 representing one bit in the 8-bit DAC. We slowed the clock speed to 16mS for confirmation that the necessary bits lit up sequentially. After we were confident that the sine wave values represented those generated in the sine table, we used the oscilloscope and probed the output of the DAC to confirm that the frequency was exactly 262Hz. After confirming the required frequency, we saved an image of each waveform, including the frequency and voltage information from the screenshot. Later, we assigned a variable to take in the any desired command from 0 to 7 as user input and calculate the desired SysTick Init value.

The final part of this first project was to configure the Bluetooth module to transmit control data to MCU1. The HM-10 must be configured in command mode to set up our Bluetooth chip to project specifications. The Baud rate is set to 57600 with 1 stop bit and even parity. Once in data mode, MCU1 will receive data from the HM-10 via Bluetooth terminal

application. To change chip settings in command mode, AT+ commands were sent from MCU1 using push button 2 to cycle through the required commands. The device selection is controlled by push button 1 and cycles corresponding LED colors. A number value between 0 to 7 is received via Bluetooth Serial Terminal as the command control. MCU1 will call functions for the command received and utilize functions for the PWM pulse for the start, device, and command data. The pulsed data will be sent via the IR LED to MCU2 wirelessly via the IR receiver.

3. Hardware

3.1. Hardware Block Diagram

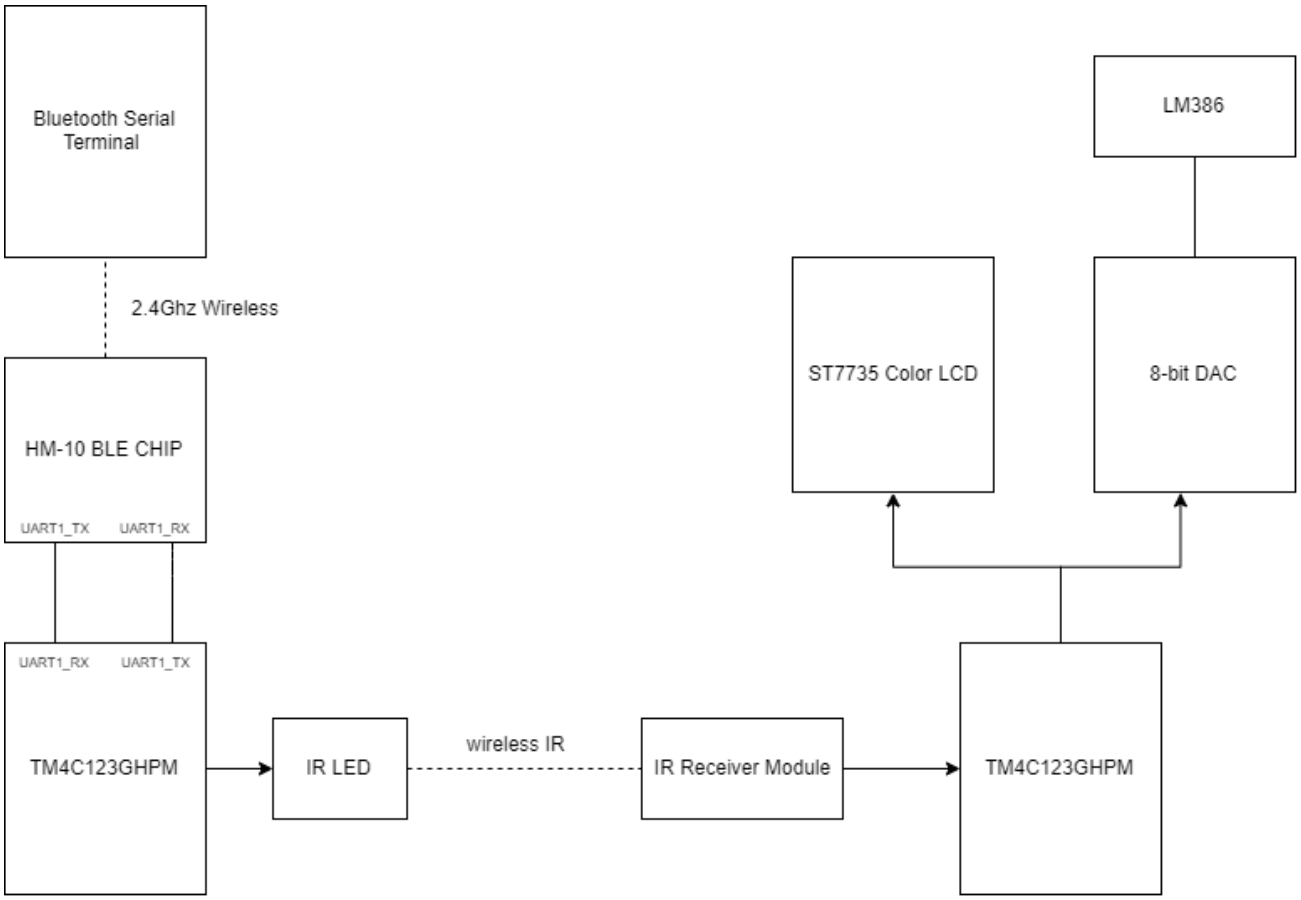


Figure 3 Top Down Block Diagram

3.2. Schematic

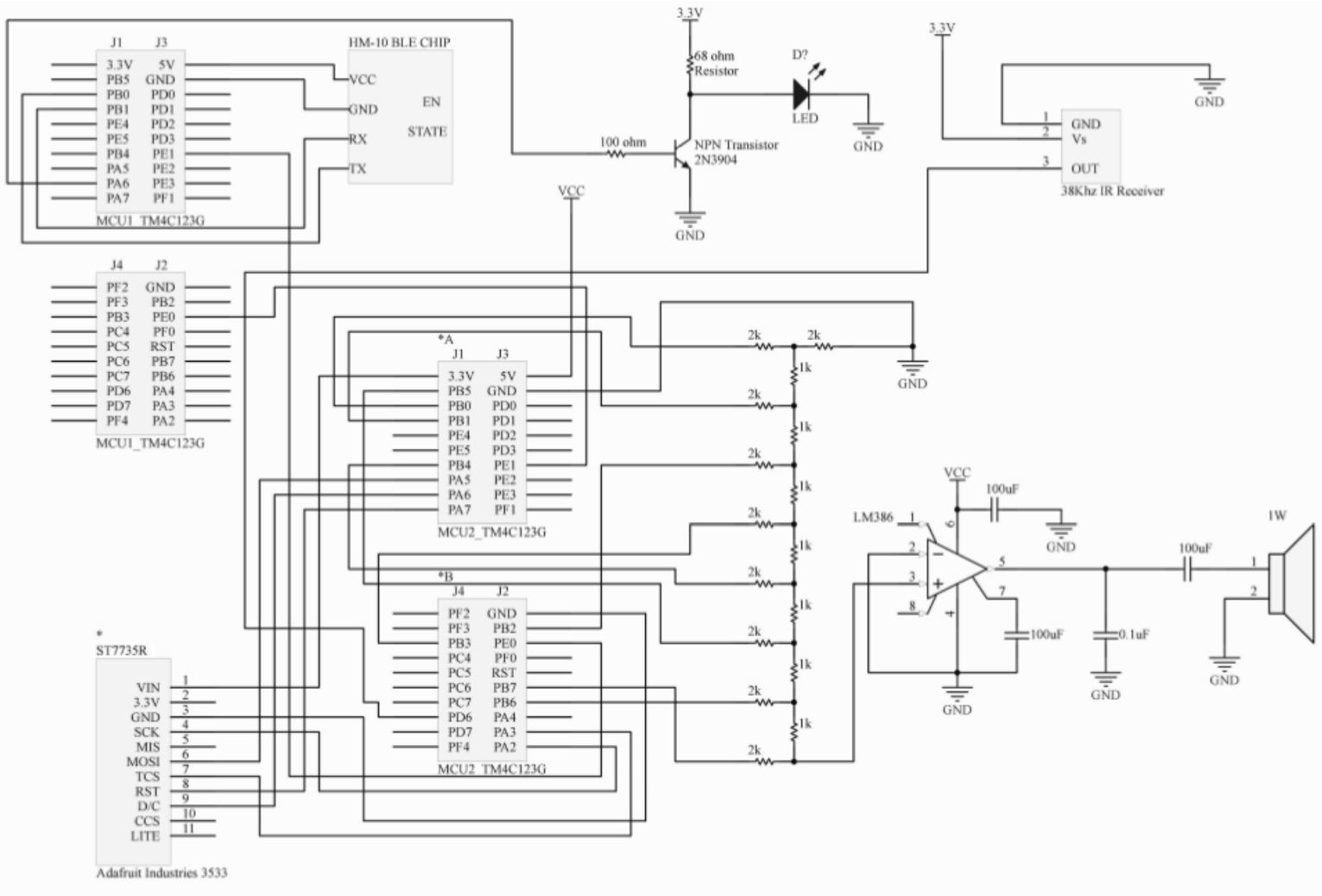


Figure 4 HM-10, MCU1&2 connected to R2R DAC with Amplifier, Potentiometer, and Speaker, LCD, IR LED & Receiver

3.3. Component List

- TM4C123 microcontroller x2
- LM386 audio power amplifier
- 1W Mini speaker
- 1% tolerance resistors
- 0.1-100µF Capacitors
- HM-10 Bluetooth Transceiver
- 940nm IR LED
- 38Khz IR Receiver Module
- ST7735 Color LCD
- NPN transistor

3.4. Hardware Explanation

The HM-10 Bluetooth chip has four pins, Vcc, Gnd, Rx, and Tx. The Rx and Tx pins are used for UART communication between the BLE chip and TM4C123. The HM-10 is configured to transmit data at 57600 bits per second (bps), with 1 stop bit and even parity. These configurations must match between devices to ensure the exchanges are correctly decoded. MCU1 will send a signal via the IR LED with correct timing for either logic '0' or logic '1' for the start, address, and command data package. PWM and SysTick are utilized to correctly send pulses of data.

The HM-10 draws power from the VBus pin, extracting around 5V. We chose the HM-10 as our Bluetooth module over a recommend HC-05 chip for a few reasons. The HM-10 is a newer chip which supports Bluetooth 4.0/BLE making this compatible with an iOS device. With the HM-10 utilizing Bluetooth Low Energy (BLE) the amount of power consumption is a lot less than the HC-05 chip. This power factor was decided with our robot car in mind, which requires clean and stable power. We did not want to have to convert from HC-05 to HM-10 once the robot car project is assigned.

The R2R DAC design allows for a more precise Voltage output value. The output of the R2R network is will send a value to the LM386 amplifier. The LM386 amplifies the smaller signal to a larger signal to eventually send to the 1-watt speaker. High and Low noise capacitors are used to smooth the noise coming out of the amplifier to generate a smooth and consistent sounds/waveform. The MCU2 TM4C123 is configured to read signals on **PIN** (enter the pin) which will be decoded and analyzed to determine the device and command to execute.

To display our animations, we are using the ST7735 LCD. VCC is connected to 3.3V from TM4C microcontroller. Reset is GPIO from PA7. Data/Command is GPIO from PA6, high for data, low for command. TFT_CS is connected to PA3 for SSIOFss. MOSI (Master our, slave in) is connected to PA5 for SSIOTx. SCK is connected to PA2 for SSIOClk.

4. Software

4.0.0 Microcontroller 1

The software design of the first microcontroller (MCU1) acts as the main workhorse of the UART communication feature. With this project requiring timing accuracy, we initialized Phase-lock loop to generate an 80Mhz system clock for the Tiva TM4C123GH6PM microcontroller. The HM-10 Rx and Tx lines are configured to use UART1 on MCU1. To use the HM-10 chip with MCU1, the HM-10 must be configured in command mode using AT commands. A Port F handler function was designed to change the device number on each push button. With each button press, the handler will set the device value and change the onboard LED to the corresponding device LED.

The UART configuration consisted of choosing the appropriate UART channels and values. for both the HM-10 and MCU2 communication. UART1 is contained to pins 0 and 1 on Port B. For UART1, the integer baud rate divider (IBRD) is set to 86 and the fractional baud rate divider (FBRD) is set to 52. These values were calculated using the following formulas:
IBRD = int(80,000,000 / (16 *57,600)) and FBRD = int(0.8055 * 64 + 0.5).
These values allow UART1 to communicate at 57600 bps. The line control register is set to 0x0076 for an 8-bit value with 1 stop bit and even parity.

Pulse Width Modulation was utilized to meet the project requirement of dynamically changing the pulse of the IR LED to send an accurate LOW and HIGH time for the data receiving and decoding. In the previous project, PWM6 in Module 1 Generator 3 was used to display a Blue LED because that PWM is tied to pin PF2 which is the designated Blue LED control. The essential code that allows usage of PWM6 is PWM1_ENABLE_R, which gets set to value of 0x00000040.

The continuous logic control block of MCU1 is comprised in the main function of the C program. The Main function starts with initialization of necessary components such as: Phase locked loop, GPIO, UART, and Pulse-width modulation. Inside the continuous while loop, data is being polled on UART1 line as a char type and being stored into a char variable called *cmd*. The data collected on the UART1 line will be from '0' to '7' as a valid command. The *cmd* variable is checked to have a value, not being null. A switch case statement is utilized to execute the incoming *cmd* value as a command control. Within each case, the start_Pulse(); is called to create a pulse of 1ms HIGH and 800us LOW pulse. The device_select function is called passing the current device address which will then call the corresponding device address function. The correspond command function is called depending on the command *cmd* value received.

Contained in all the Command_# (), Address_#(), and start_Pulse() functions, a logic1() or logic0() is called depending on the needed pulse signals. The logic0() and logic1() functions will enable PWM to start and call a corresponding SysTick_Wait delay time function. The PWM will be disabled and the SysTick_Wait function for the LOW pulse time.
MCU1 will send a PWM signal through the IR LED on pin A6 to the IR receiver module connected to MCU2 with a data frame in start, device, and command. The data frame will be used by MCU2 to control either the animation on the LCD display or the tone generator waveform on the DAC R2R network.

4.0.1 Microcontroller 2

For this project, we disabled interrupts and initialized a Phase-locked loop control system to generate an 80Mhz clock which is the maximum clock speed for the Tiva TM4C123GH6PM microcontroller. This implementation is necessary for dealing with the higher frequencies required generate the wave forms at 262Hz. Next, we initialized Systick as a 24-bit decrementing timer. To get the required 262Hz frequency for the four waves, we had to calculate the Systick_Init value. To do this, we took the (clock speed/262Hz/# of steps to output each wave). Sinewave of 262Hz = 80MHz/262Hz/64 = 4771. Square wave of 262Hz = 80MHz/262Hz/2 = 152,672. Triangle wave of 262Hz = 80MHz/262Hz/512 = 596. Sawtooth wave of 262Hz = 80MHz/262Hz/256 = 1193. We are initializing Port B pins 0-7 for the 8-bit output to DAC for 2^8 or 256, and due to timing constraints we could not finish the demodulation of the IR. We are initializing Port F pin 4 for the on-board pushbutton, and Port F 1-3 for RGB LED. The GPIO Port F Handler is called every time switch one is touched and command increments. If the pushbutton is pressed while on command 8, then the command variable will reset to command 0 and the flag is acknowledged and cleared.

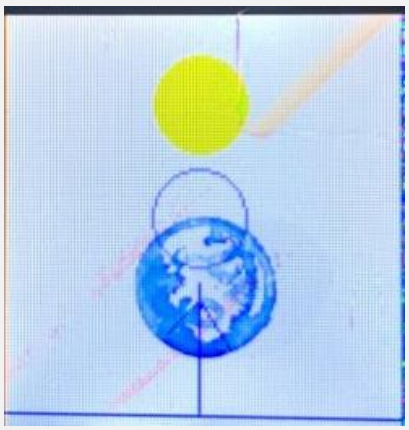
Next, we enabled interrupts to change the command with the pushbutton while cycling through the different wave configurations. For commands 0-3 we interfaced with the ST7735 LCD. To do this we used an example library and built off it. We modified the code to set the first four commands as a story. For command 0 the stickman wakes up in the morning (sun left) and goes for a walk. Command 1 occurs at noon (sun centered), and the stickman does the splits. Command 2 is at dusk (sun right), and the man is stretching while getting ready to wrap up his day. The last animation at command 3 occurs at night, and the stickman walks home.

In command 4, Systick_Init uses the sine period, Port F is green, and the sine function is called to output sine wave using 256 outputs per period. The sine function uses math.h and generates the new integer value for each pass in the loop using the simplified equation $j = (\sin(\pi * (index\ 128)) + 1) * 128$, and we set Port B data equal to j for each iteration. For this, the overhead of generating 256 bits for the sine wave at 80Mhz frequency was too much, so we reduced the sine wave to 64 outputs. In command 5, Systick_Init uses the square period, Port F LED is yellow, and the square function is called to output the two outputs per period. In command 6, Systick_Init uses the triangle period, Port F LED is blue, and the triangle function is called to output the triangle wave using 512 outputs per period. In command 7, Systick_Init uses the sawtooth period, Port F LED is red, and the sawtooth function is called to output the sawtooth wave using 256 outputs per period.

4.1.1 ST7735 Animation Verification

Command 0

Command 1

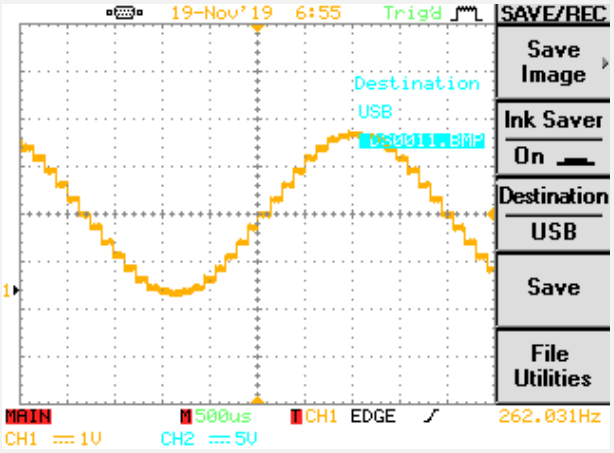


Command 2

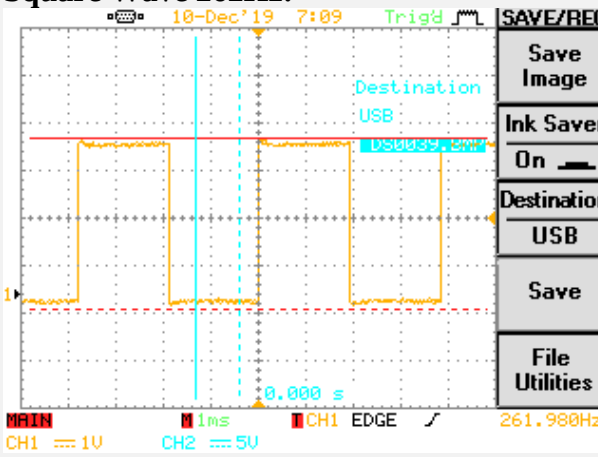
Command 3

4.1.2 Oscilloscope Frequency Verification

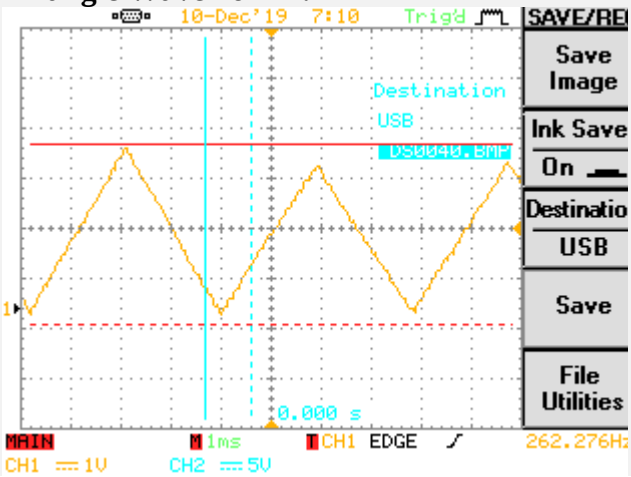
Sine Wave 262Hz:



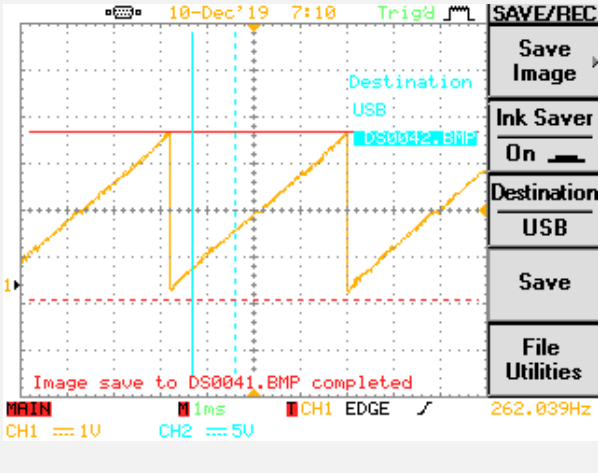
Square Wave 262Hz:



Triangle Wave 262Hz:

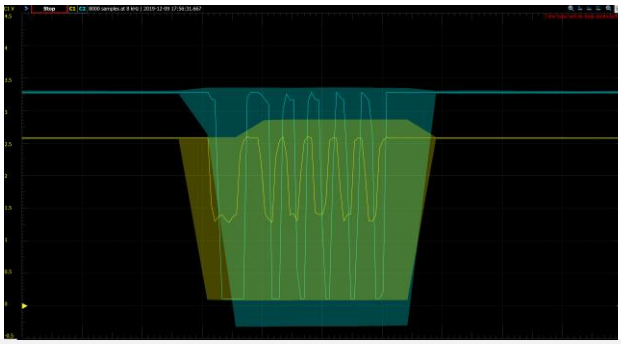


Sawtooth Wave 262Hz:

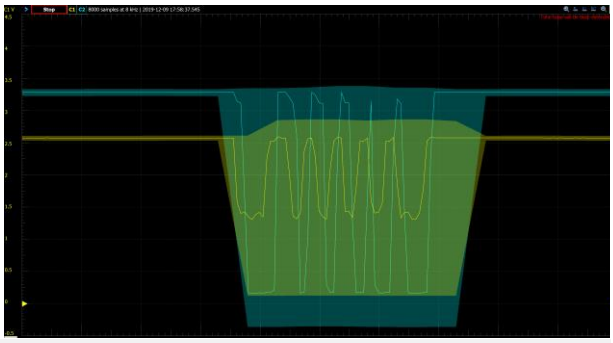


4.1.3 Oscilloscope Command Verification

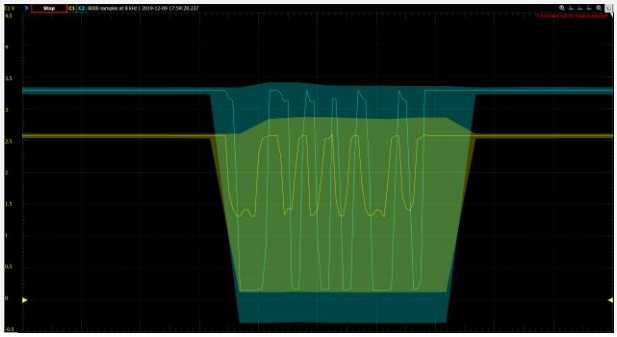
Device 0 Command 0



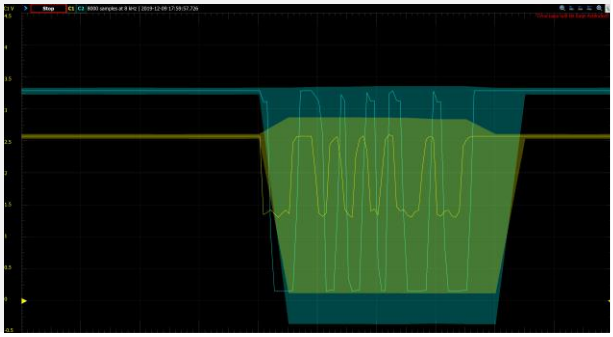
Device 0 Command 1



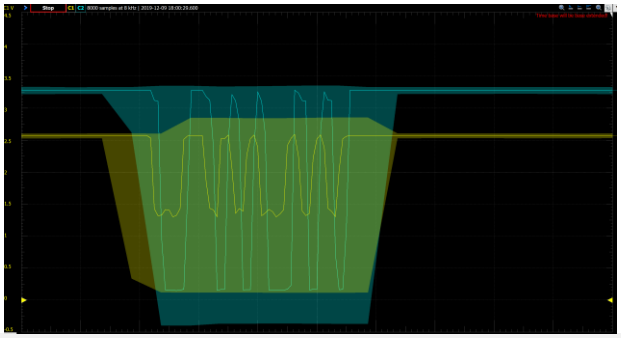
Device 0 Command 2



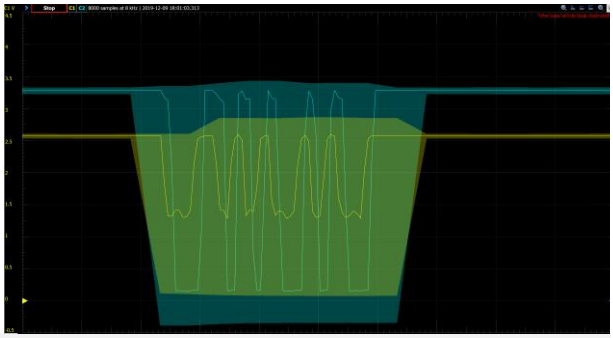
Device 0 Command 3



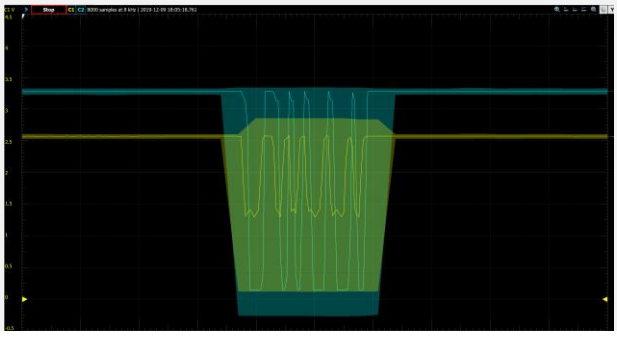
Device 0 Command 4



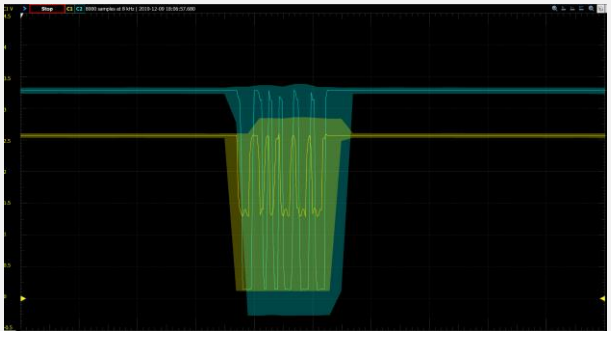
Device 0 Command 5



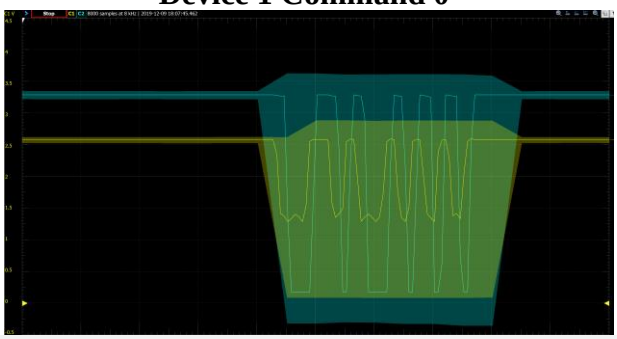
Device 0 Command 6



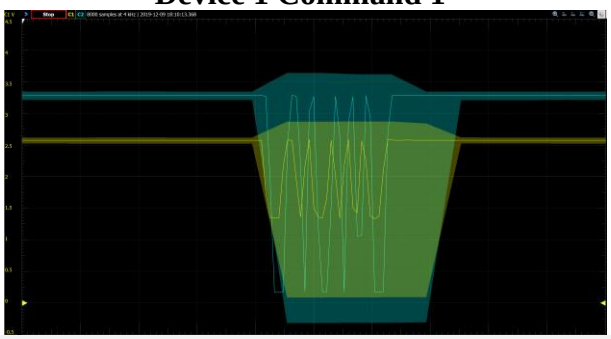
Device 0 Command 7



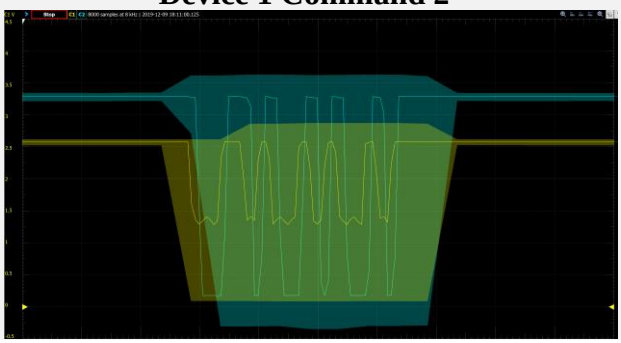
Device 1 Command 0



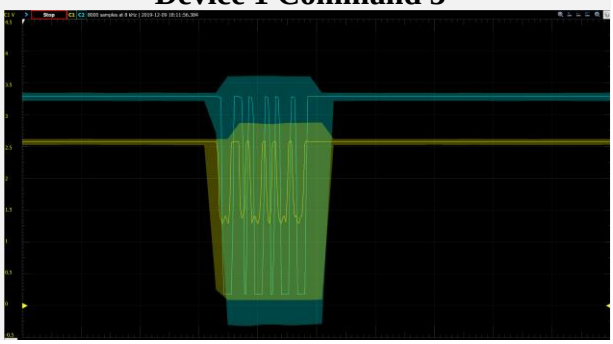
Device 1 Command 1



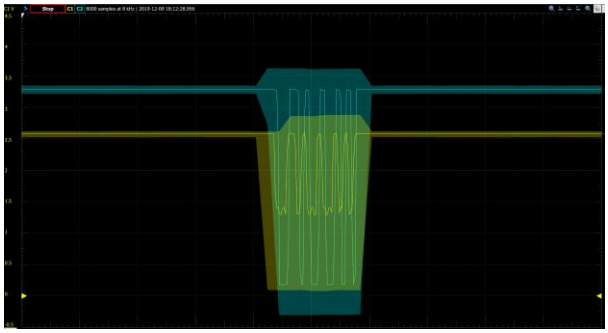
Device 1 Command 2



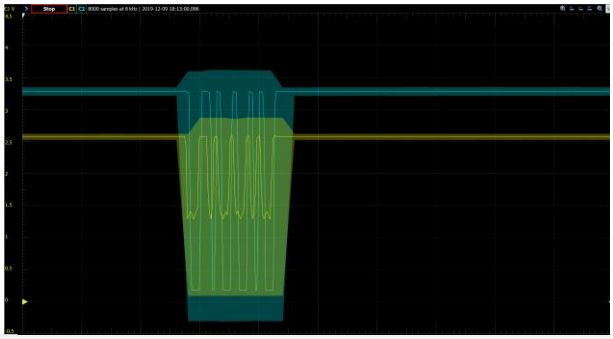
Device 1 Command 3



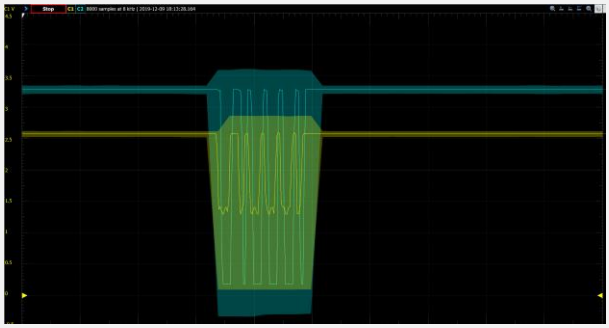
Device 1 Command 4



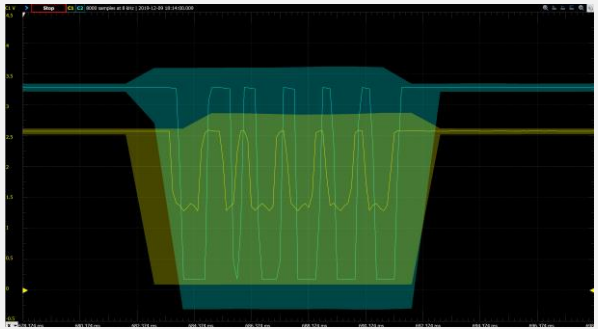
Device 1 Command 5



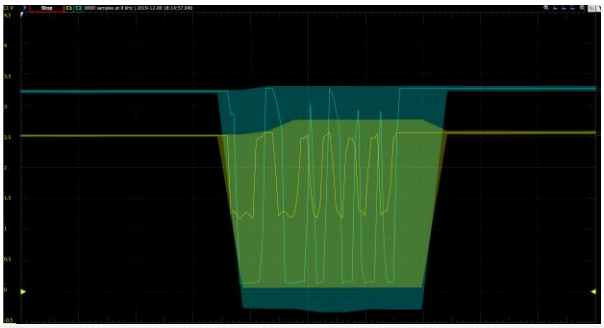
Device 1 Command 6



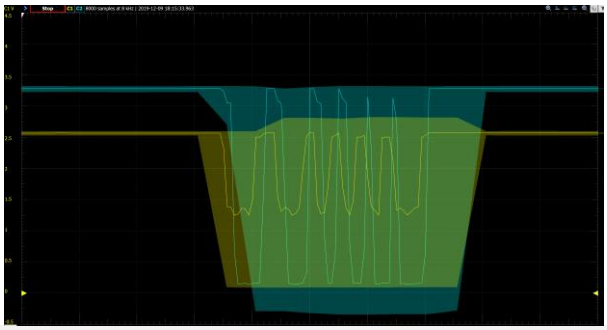
Device 1 Command 7



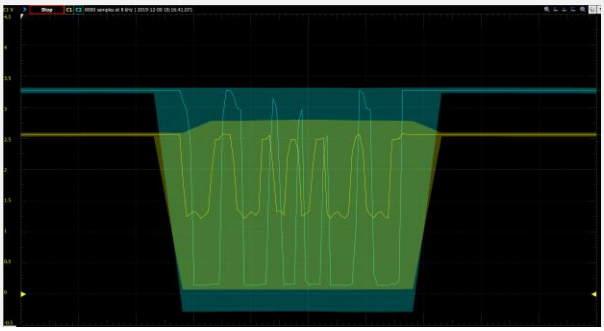
Device 2 Command 0



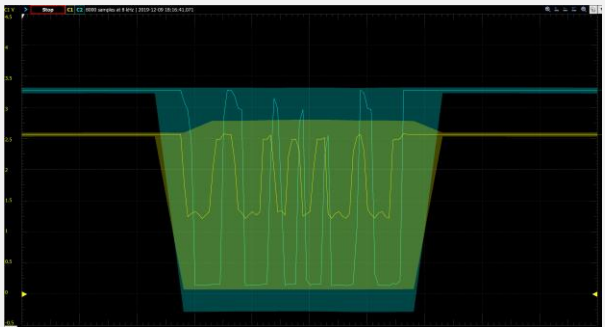
Device 2 Command 1



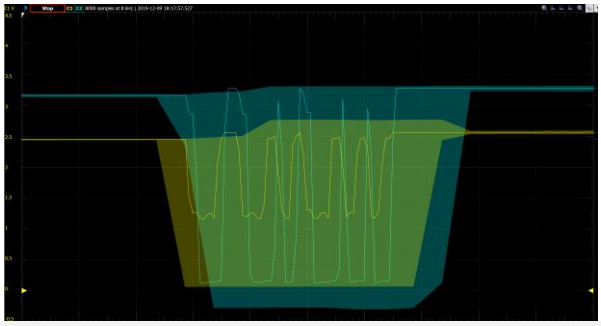
Device 2 Command 2



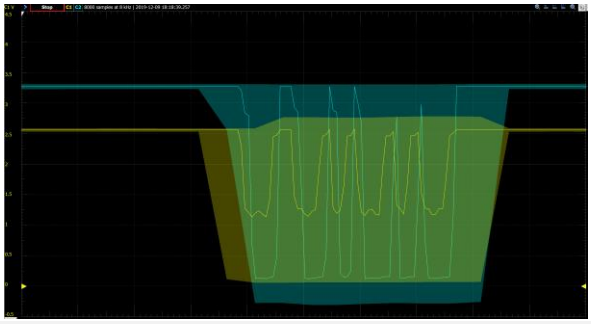
Device 2 Command 3



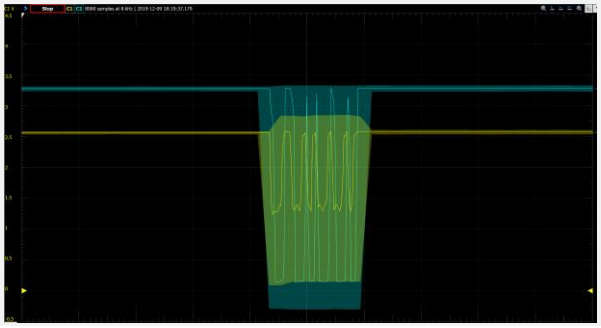
Device 2 Command 4



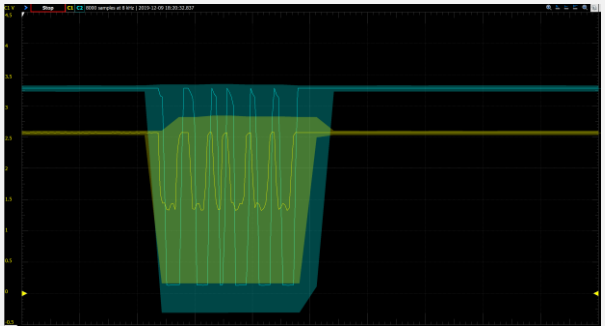
Device 2 Command 5



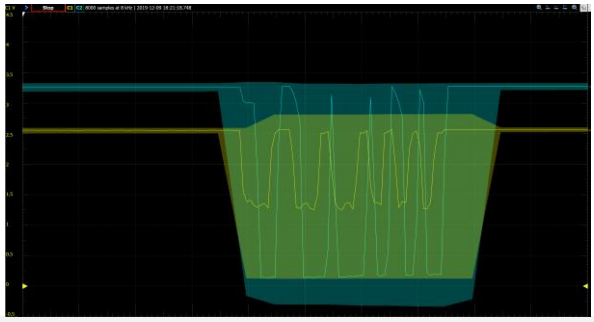
Device 2 Command 6



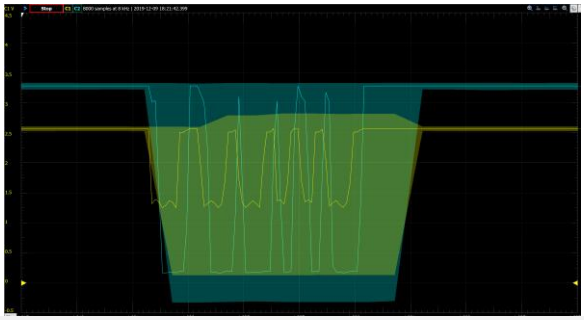
Device 2 Command 7



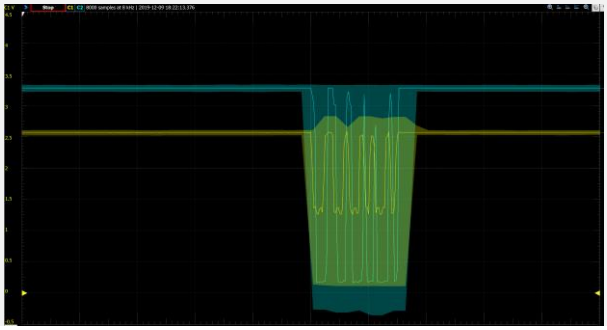
Device 3 Command 0



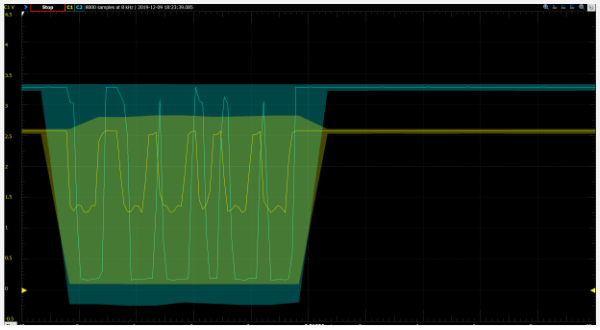
Device 3 Command 1



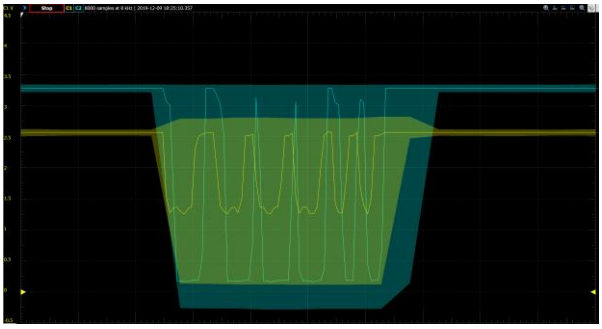
Device 3 Command 2



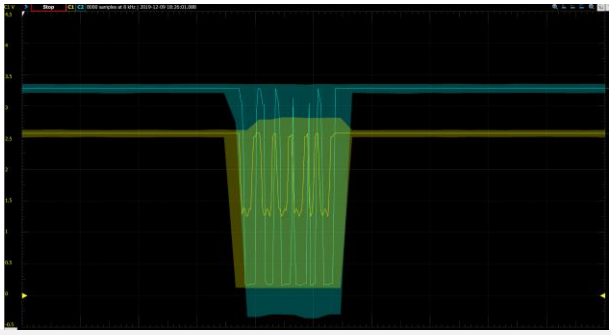
Device 3 Command 3



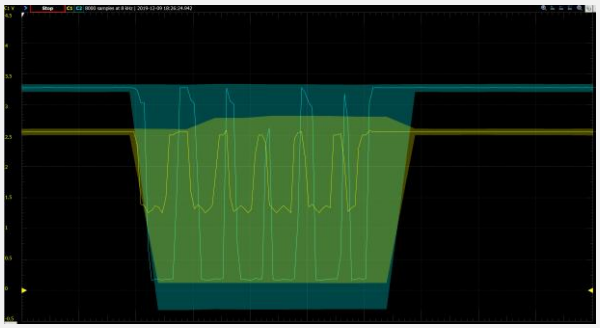
Device 3 Command 4



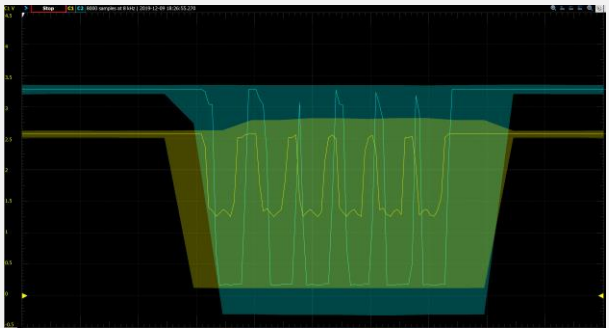
Device 3 Command 5



Device 3 Command 6



Device 3 Command 7



4.2 Terminal Verification

Sending data to a UART Terminal allows for data checking and ensuring proper functionality.

4.2.1 Button Push Verification

Every button push will send the device number to the USB Serial Terminal for verification.

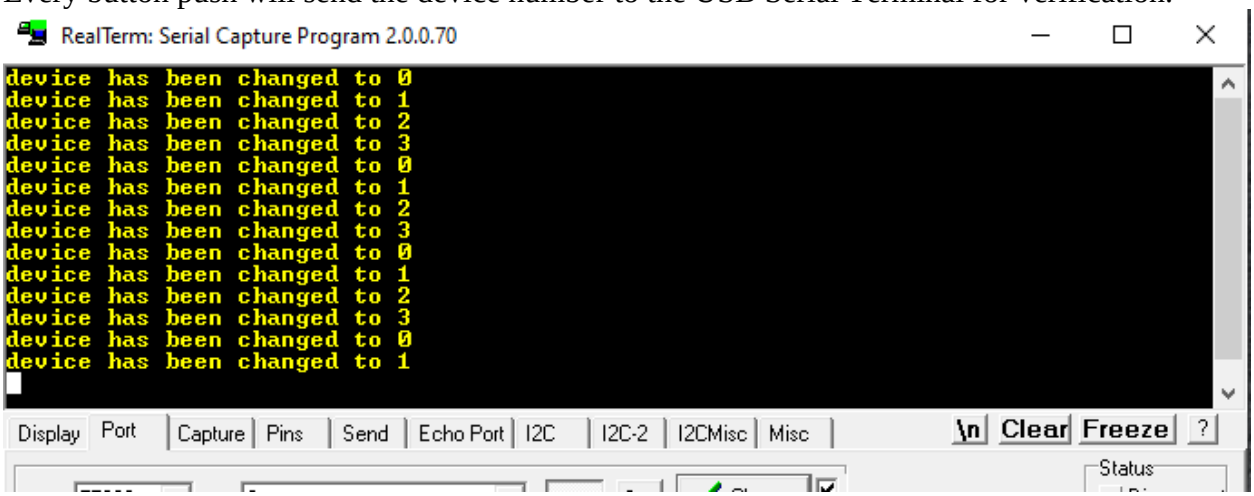


Figure 5 Device Change Display on UART0

4.2.2 Bluetooth Serial Terminal Verification

Command Verification via Bluetooth Serial Terminal mobile application

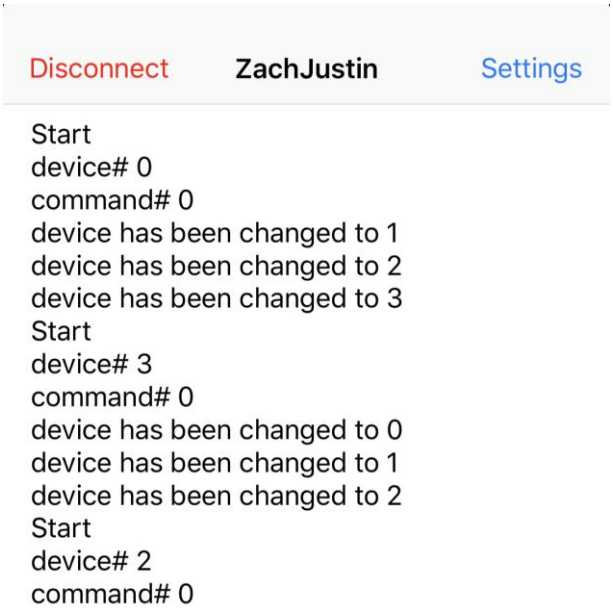


Figure 6 Bluetooth Mobile Application Terminal

4.4 Software Flow Chart

The Software design of the final project required clean flow of data from microcontroller to microcontroller.

4.4.1 MCU1 Flow Chart

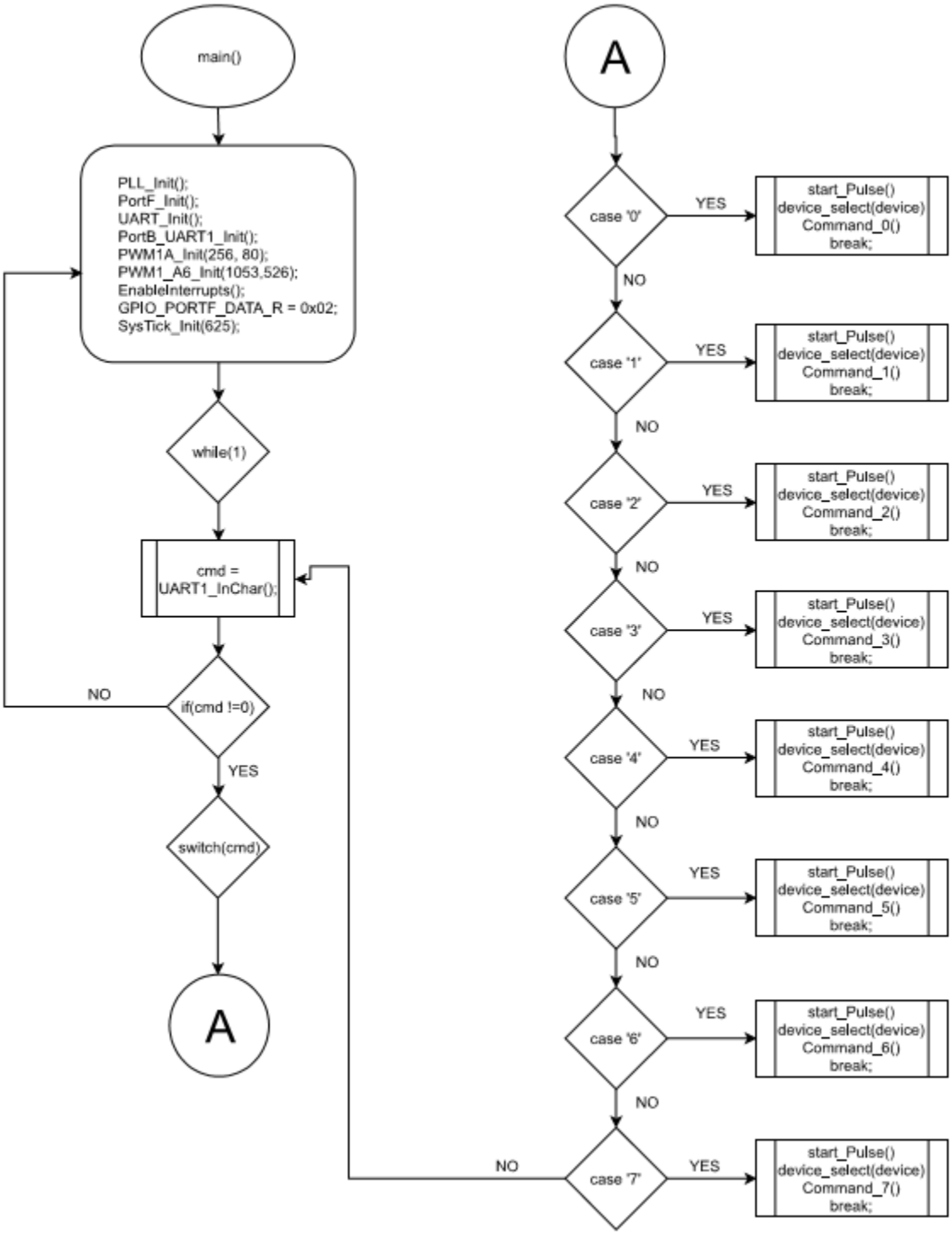


Figure 6 MCU1 Main Flow

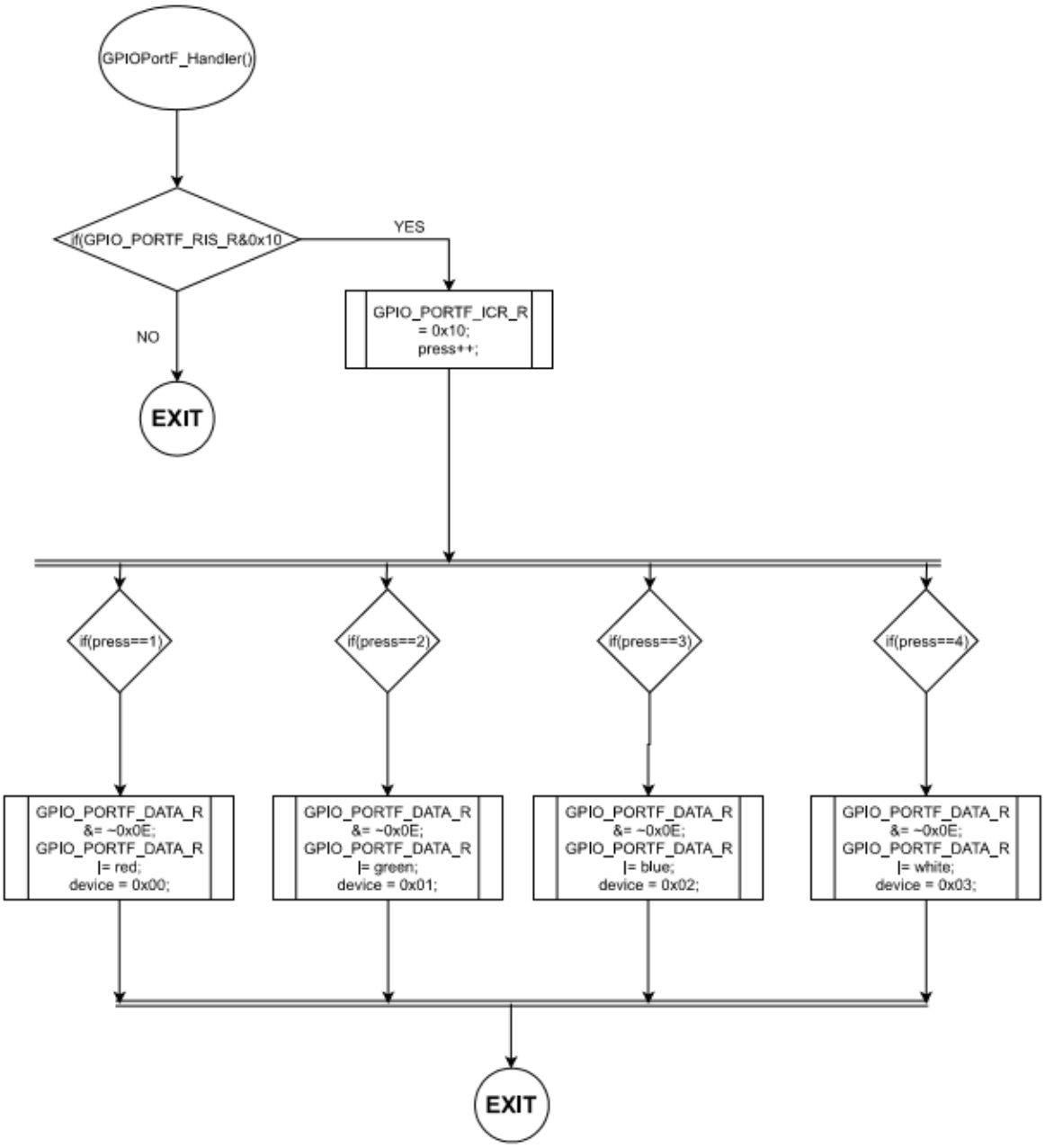


Figure 7 MCU1 Port F Handler

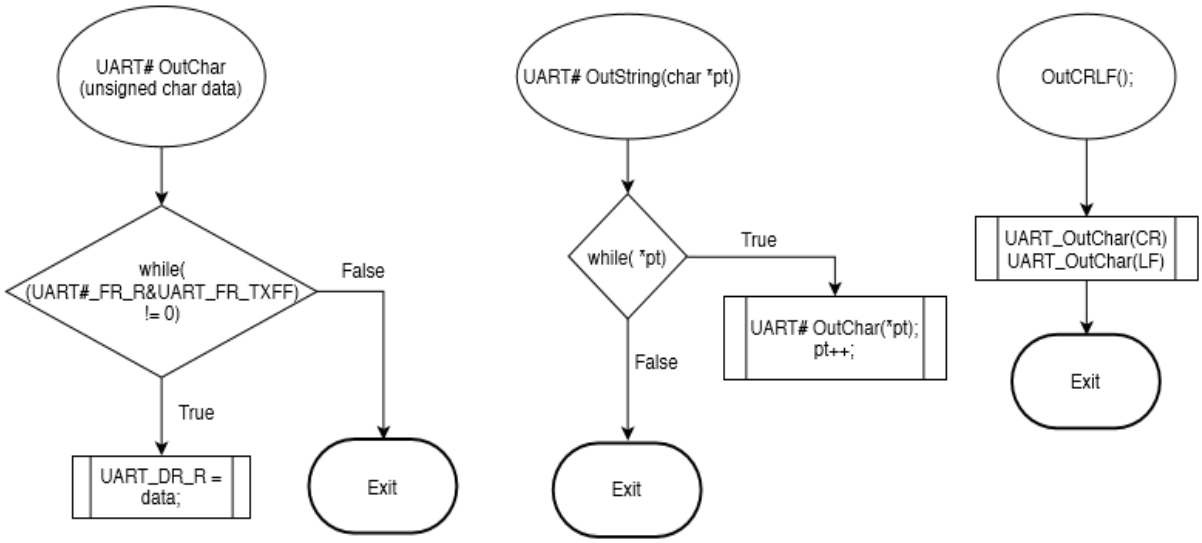


Figure 8 UART Out Functions

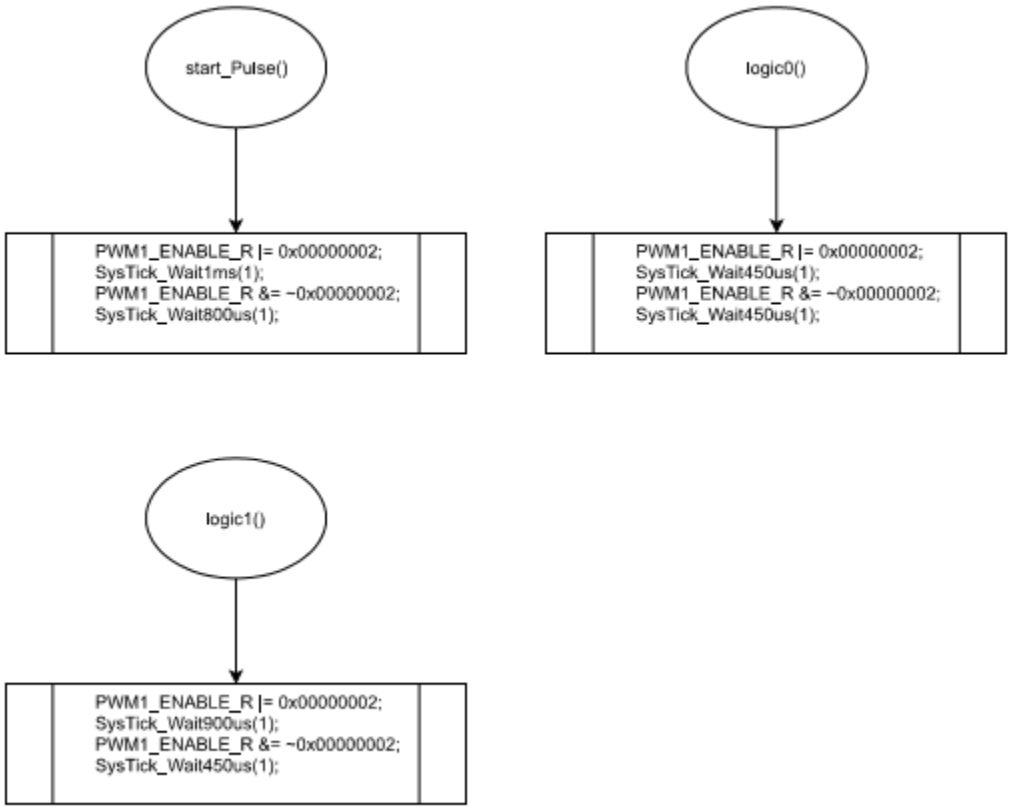


Figure 9 PWM Pulse Functions for IR LED

4.4.2 MCU2 Flow Chart

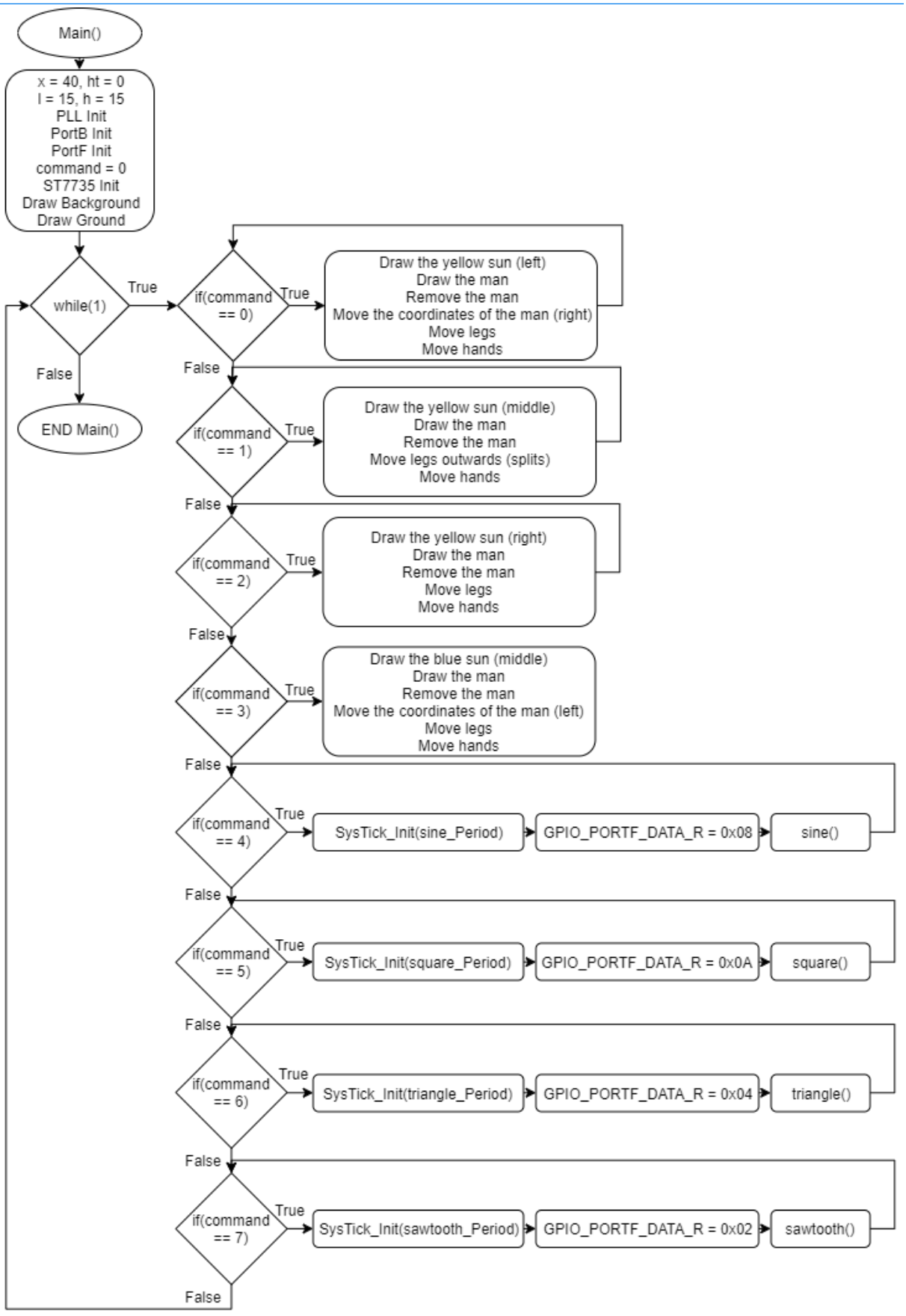


Figure 10 MCU1 Main Flow

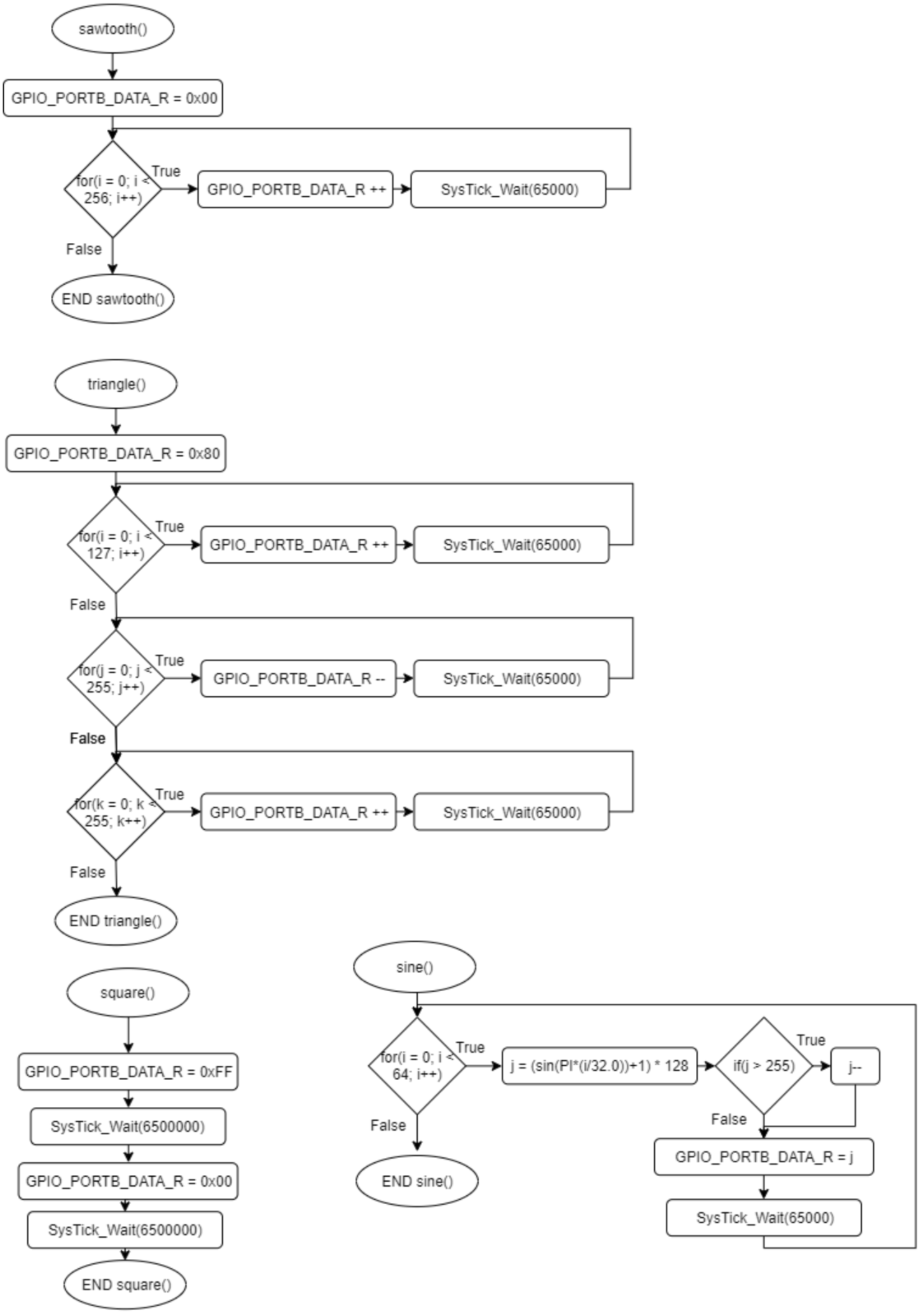


Figure 11 Wave Generation Functions

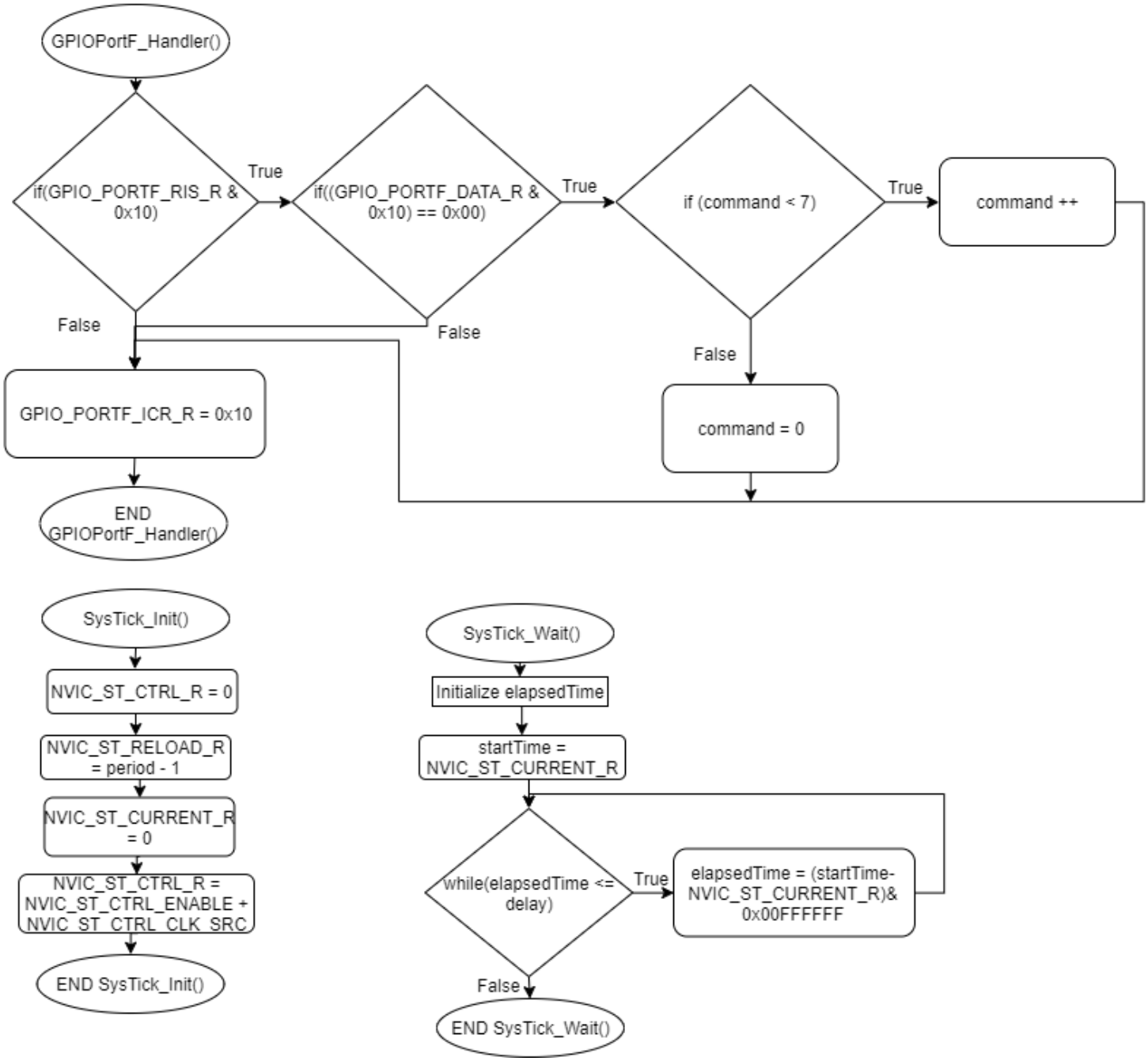


Figure 12 Port F Handler and SysTick

5. Conclusion

For the final project we faced a few challenges along the way. The overall purpose is to create an 8-bit tone generator using both Digital and Analog, Bluetooth, UART, and Infrared components. First, the user selects the device 0 to 3 with the port F push button. Both microcontrollers must be on the same device to receive the correct bit stream. The user inputs a command from 0 to 7 into the Bluetooth application on their mobile device. This message will be sent via Bluetooth communication to the first microcontroller that will decode the message. When the Bluetooth is ready to receive a "Start" message will display. The device number displays "device has been changed to #" with the device number when the onboard pushbutton is pressed. The command will be entered by the user and displayed on the Bluetooth application.

The first microcontroller will send the data with the current device and command through an Infrared LED at the given IR protocol. The second microcontroller will decode the IR receiver data and take the command as an input. The results will be outputted to the LCD or DAC (depending on the command) and seen on the Oscilloscope where the four different waves will be produced with the frequency of 262Hz. First, we split up the work so that one person would do the Bluetooth/first microcontroller, and the second person would work on the second microcontroller, LCD, and 8-bit DAC communication. Our problems occurred while once we finished our separate parts, and we realized that the microcontrollers could not send over the correct data. We tried to debug our own code separately until we realized we needed to debug together to make any progress. Due to timing constraints of the final project, we were unable to accurately demodulate the infrared signal.