Zachery Takkesh - 015656509
Justin Maeder - 015906629

CECS 447 – Embedded Systems III

Project 2: UART and Bluetooth controlled DAC - Tone Generator

11/18/2019

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 value to MCU2 and control the DAC network using the stream of data. The result of the transmitted data from MCU1 to MCU2 will play a generated tone on the speaker. 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. Set Bluetooth Module to Command Mode and change to the following configuration:
    a. Bluetooth name to your group name.
    b. UART Configuration: 57600 bps, 8 Data Bits, 1 stop, even parity.
    c. Set Passcode other than 1234.
    d. Set to Slave Mode.

2. Record your serial terminal input and output data while you are configuring the device in command mode.

3. Set Bluetooth Module to Data Mode and serially communicate with TM4C_1
    a. Download a Bluetooth terminal application on smartphone
    b. Using smartphone, pair with your Bluetooth module.
    c. Using smartphone send the following message: ‘f’ as a header and a number after that to define the frequency for the DAC. Frequency range will be from 262Hz-494Hz. Ex. “f262” means frequency of 262Hz. This data will be serially sent from TM4C_1 to TM4C_2 with TM4C_2 decoding the message and output the correct DAC output. The output will be a sinewave and by default, frequency is 262Hz.
    d. Use ‘b’ as a header and a number after that to define the brightness of the blue onboard LED that is on TM4C_1. LED brightness ranges from 0-255. Ex. “b128” will output half brightness.

2 Operation

The first part of Project 2 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 Project 2 is generating a sine wave using variable frequencies from 262Hz – 494Hz 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 minimum 262Hz and our maximum 494Hz. We utilized UART to switch between the frequencies. 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 and 494Hz. After confirming the required frequencies, 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 frequency from 262-494 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. To change chip settings in command mode, AT+ commands were sent from MCU1 using push button 2 to cycle through the required commands. 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. The data is decoded and then sent to MCU2 as a frequency control via UART7. MCU1 is also responsible for changing the brightness of the onboard Blue LED using values sent from the HM-10. Once the sine frequency was confirmed on the oscilloscope, we listened to the output by connecting a non-amplified speaker to an LM386 to amplify the DAC output.

3. Hardware

3.1. Hardware Block Diagram

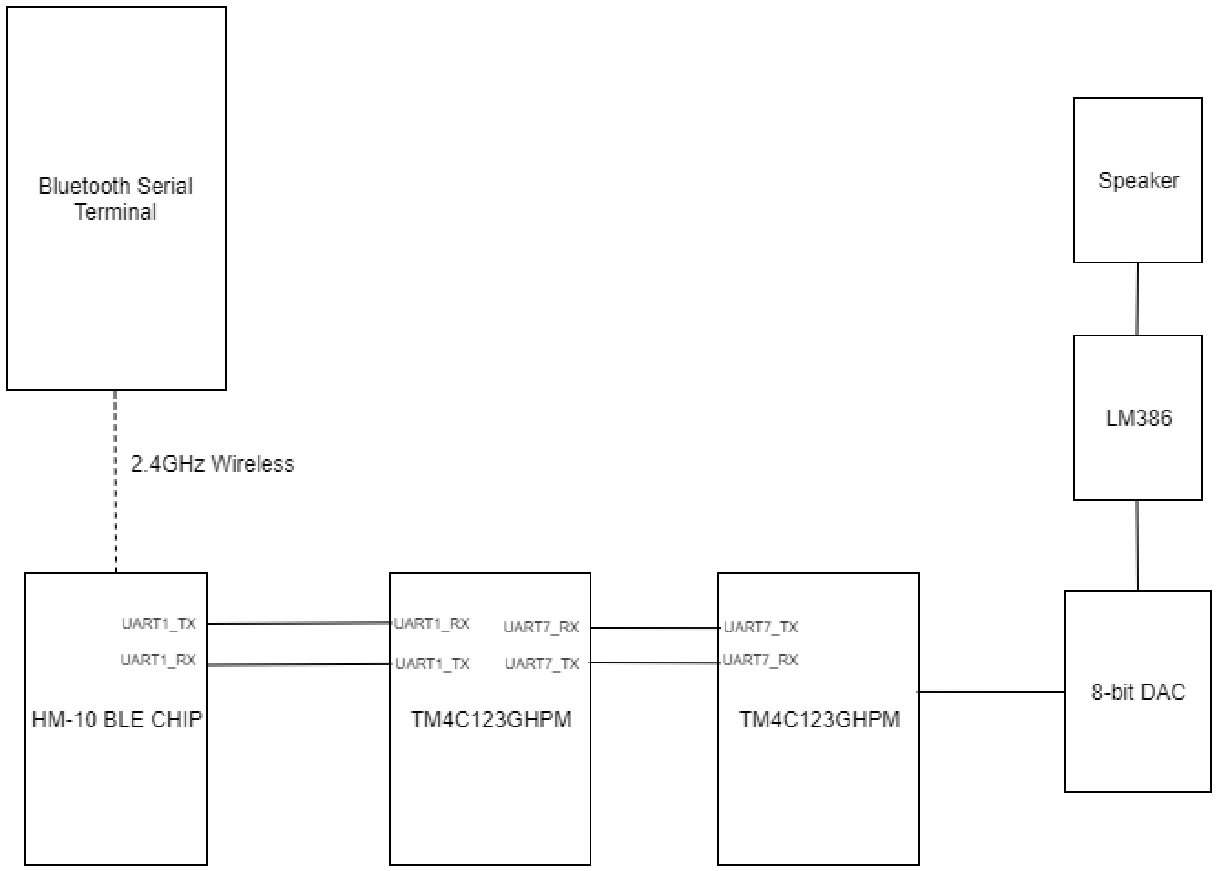


Figure 1 Top Down Block Diagram

3.2. Schematic

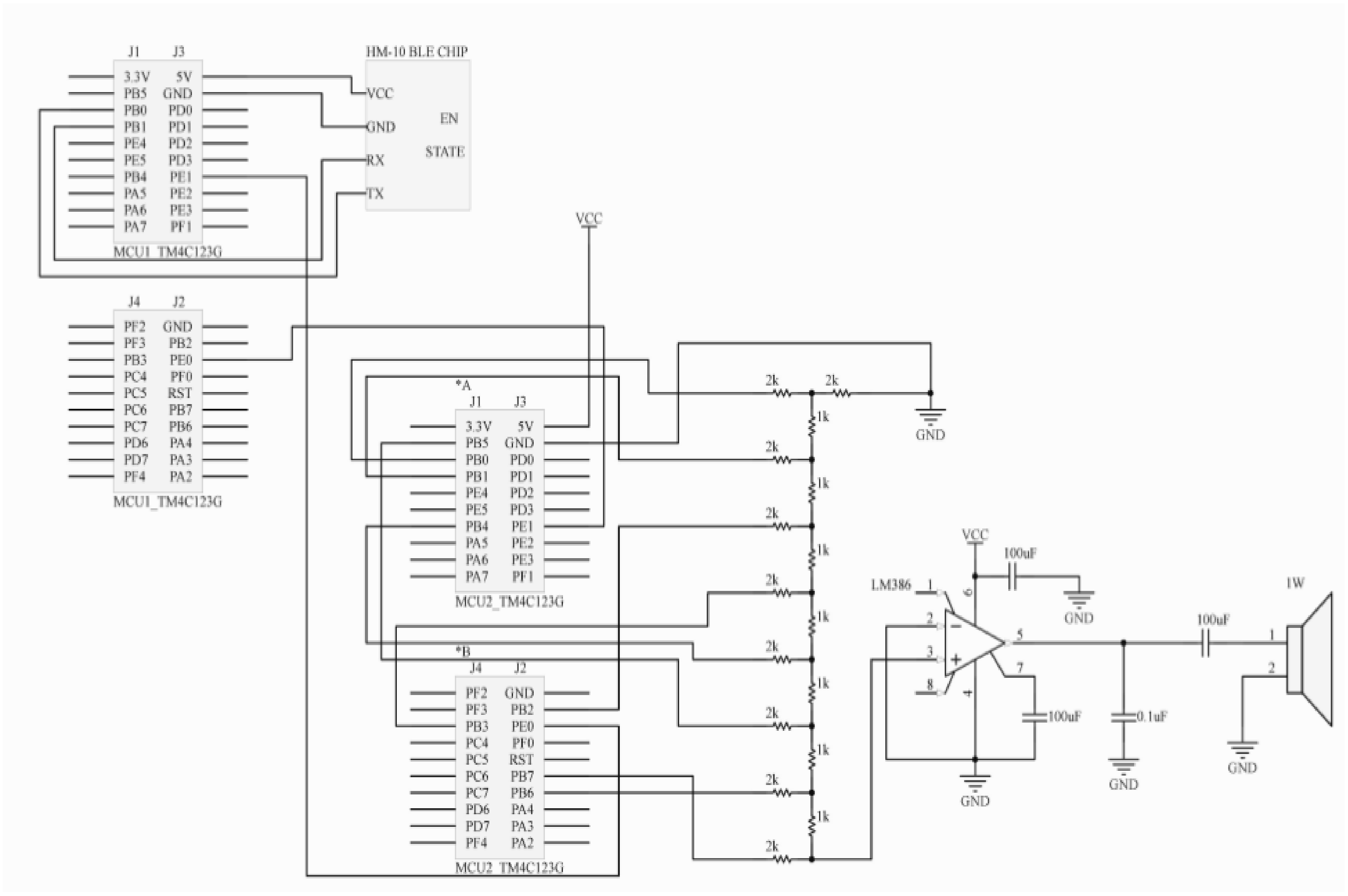


Figure 2 HM-10, MCU1&2 connected to R2R DAC with Amplifier, Potentiometer, and Speaker

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

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. With the DAC network being utilized on Port B, we had to select a different UART channel to use between MCU1 and MCU2. UART7 was chosen because it utilizes Port E which has not been used in our design on either microcontroller. 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 TM4C123 is configured to read values on UART7 pin which in turn changes the frequency of the sine wave. The frequency is to be between 262Hz to 494Hz.

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. To send AT commands from MCU1 to HM-10, a PortF handler function was designed to send an AT command on every button push tied to PF0 through the UART1 line. The commands sent over to the HM-10 changed: the device name to our group name of “ZachJustin”, Baud Rate to 57600 bps, and a Passcode to “6969”. After the HM-10 is configured, the chip changes to data mode once the mobile device connects.

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. For the MCU1 to MCU2 communication, UART7 being contained to pins 0 and 1 on Port E was utilized since the DAC network uses Port B for tone generation. The configuration for UART7 is similar to the setup of UART1 except for the line control register using the value of 0x0070 for an 8-bit value with 1 stop bit and no parity.

Pulse Width Modulation was utilized to meet the project requirement of dynamically changing the Blue LED brightness from user serial input. To display a Blue LED, PWM6 in Module 1 Generator 3 was used 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 string type and being stored into a char array with size 5. The data collected on the UART1 line will either contain a ‘b’ or ‘f’ as a header and a 3-digit value for the brightness and/or frequency. The entire data stream is evaluated as a string rather than using a for-loop to accept the first bit as a char and rest as integers. Evaluating as a string enabled the usage of string slicing with a specified index. To slice the data steam, a slice function was created to accept: a pointer to the data array, a pointer to an empty array for the inputted numbers, and a start and end slicing index. The slice function contains a for loop that counts from the start and end values while extracting the data between the start and end index into a separate array.

In the first conditional block, the data stream at index 0 is checked if the header bit is a ‘b’ for brightness. Once the data bit is confirmed equal to a ‘b’, the slice_str function is called using the entire data array, a second array to store the sliced string, a 1 along with a 3 for the start and end index respectively. With the sliced array being a string type and the PWM duty function only accepting an integer, the data in the array must be converted to an integer using the atoi standard function. The result of the atoi function stored the integer into a separate variable called val to be sent to the PWM duty block for controlling the LED cycle. To verify the data was being transmitted correctly, the data array and number value were displayed on the USB serial terminal. In the second conditional block, the data stream at index 0 is checked if the header bit is a ‘f’ for frequency. Once the data bit at index 0 is confirmed equal to a ‘f’, the slice_str function is called with the same characteristics as previously noted. Initially, MCU1 would send an integer to MCU2 for the frequency control, but after copious time spent debugging the cross-communication problems the MCU1 was redesigned to send just the numeric value as a sliced string instead of using atoi data type casting. The string of numerical values was sent via UART7 from MCU1 to MCU2. To completely send a correct string to MCU2, a carriage return and new line characters were sent to prevent the UART7 receive function from continuously polling data. To verify the data was being transmitted correctly via UART7, the numeric value was displayed on the USB serial terminal on both microcontrollers. The value sent to MCU2 controls the frequency of the DAC 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 a sine wave from 262Hz – 494Hz based on user input. Next, we initialized Systick as a 24-bit decrementing timer. To get the required 262-494 Hz frequency for sine waves, we had to calculate the Systick_Init value. To do this, we took the (clock speed/# of steps to output sine wave) 80Mhz/32 = 2,500,000 = y, then divided y by the desired frequency input by the user. For example, a sinewave of 262Hz = y/262 = 9542. We are initializing Port B pins 0-7 for the 8-bit output to DAC for 2^8 or 256.

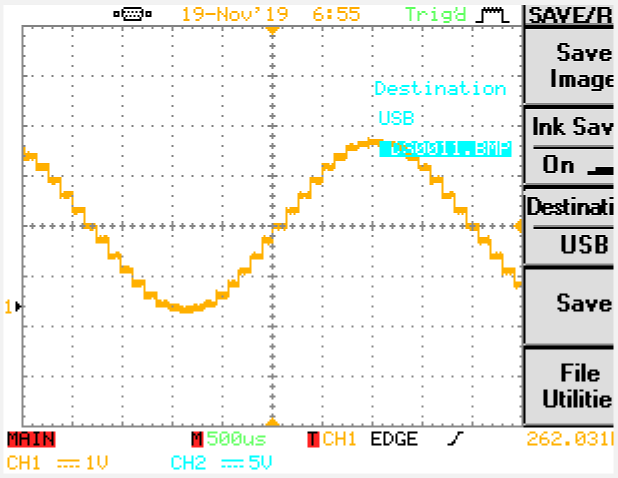
We chose UART7 on port E for the intercommunication of the two microcontrollers. We initialized both microcontrollers to UART7 with the same setup of baud rate of 57600 with 8 bits word length, no parity bits, one stop bit, and FIFO's based on the same calculations discussed in 4.0.0. To setup the UART7 alternate function and digital I/O must be enabled and configured to UART, and analog functionality must be disabled.

Next, we then set frequency to 262 for the initial value of 262Hz and nFrequency to 0 for the next frequency to receive the new frequency user input. In the infinite loop we used a conditional statement to recognize when a new frequency is transmitted from the first microcontroller to the second. The new frequency will be displayed and will go through the calculations to change the SysTick Init value the interrupts will be enabled. After, the sine function is called to output sine wave using 32 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 $j = \left(\sin\left(\pi * \left(\frac{index}{16}\right)\right) + 1\right) * 128$, 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 32 outputs.

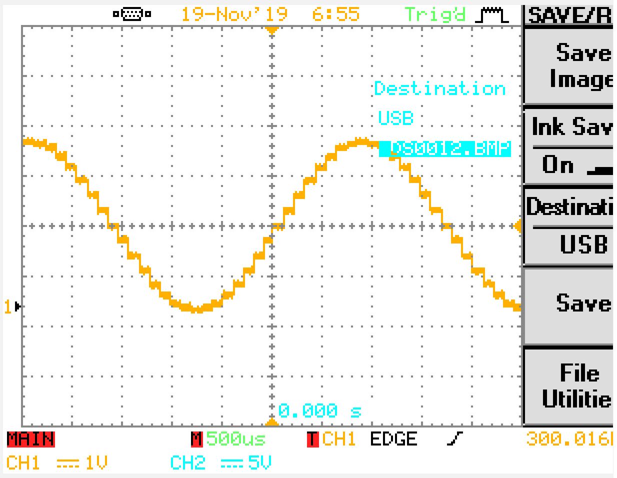
To acquire the user inputted frequency, we used the function UART7_InUDec to take an ASCII input in unsigned decimal format and converts to 32-bit unsigned integer ranging from 0 to (2^32)-1. To output the frequency to the terminal, we used UART_OutUDec function which outputs a 32-bit integer in unsigned decimal format. This function is implemented with recursion to convert the decimal number as an ASCII string.

4.1 Oscilloscope Frequency Verification

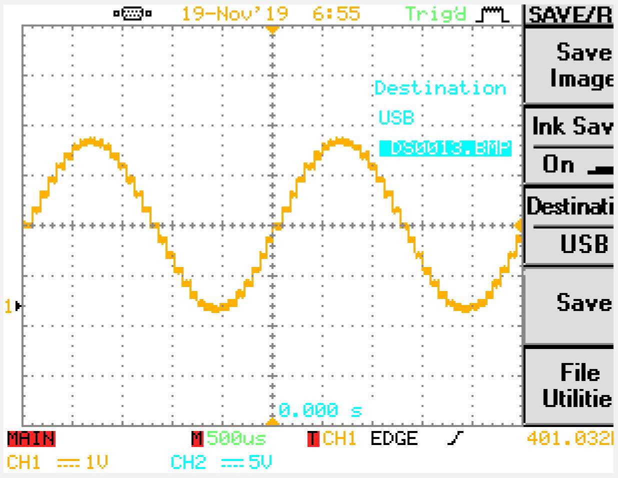
Sine Wave 262Hz:



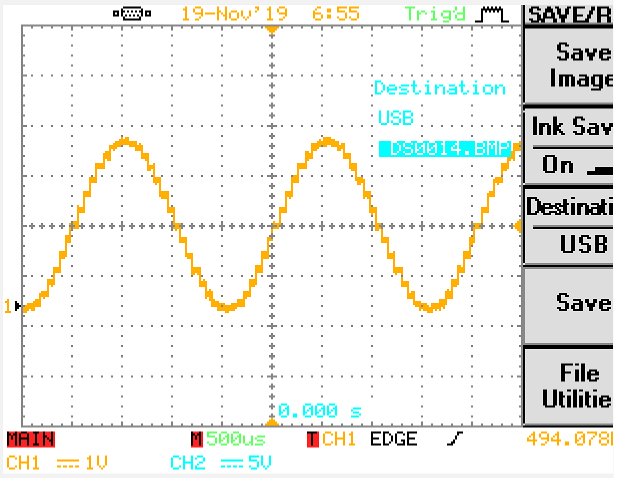
Sine Wave 300Hz:



Sine Wave 401Hz:



Sine Wave 494Hz:

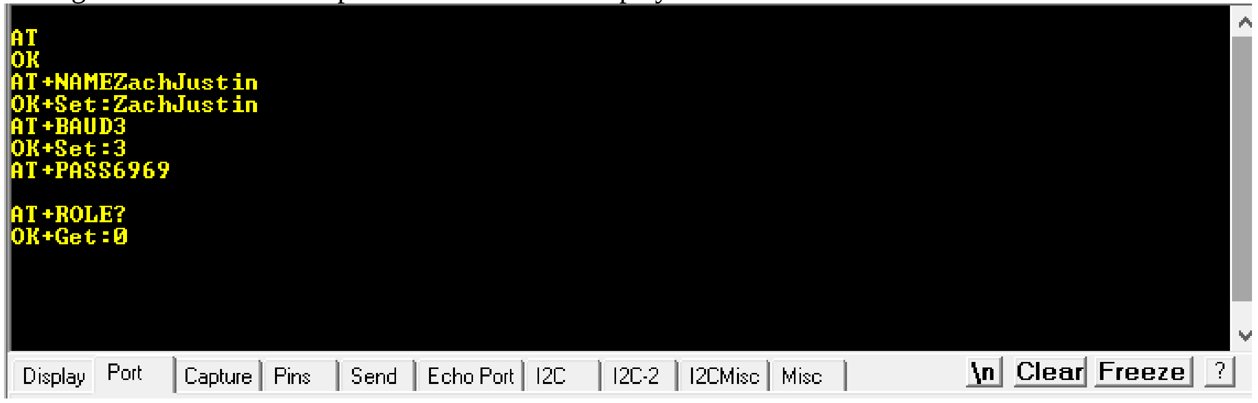


4.2 Terminal Verification

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

4.2.1 Bluetooth Command Mode Verification

Configuration of HM-10 chip in command mode displayed on USB terminal via UART0.



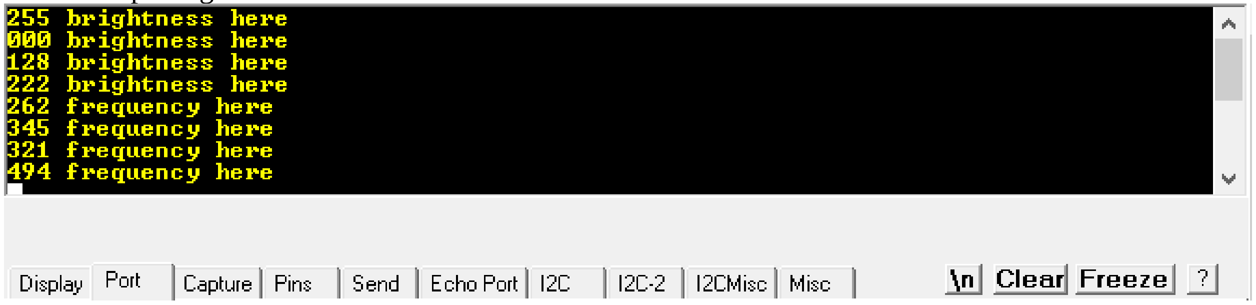
```
AT
OK
AT+NAMEZachJustin
OK+Set:ZachJustin
AT+BAUD3
OK+Set:3
AT+PASS6969

AT+ROLE?
OK+Get:0
```

Figure 3 Display AT commands to USB UART

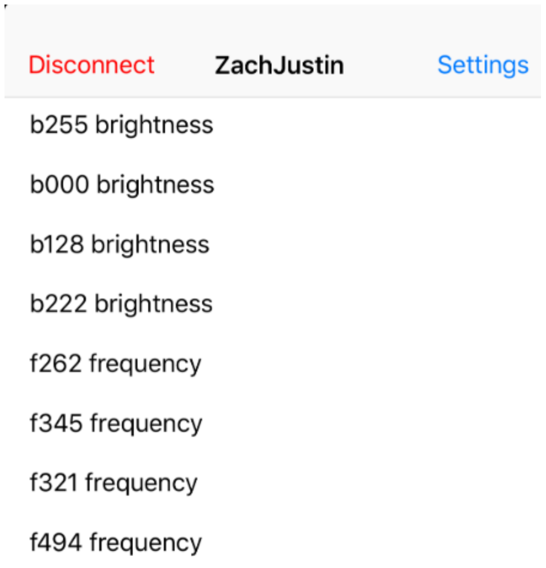
4.2.2 MCU1 Bluetooth Verification

MCU1 outputting data from HM-10 to USB terminal via UART0



```
255 brightness here
000 brightness here
128 brightness here
222 brightness here
262 frequency here
345 frequency here
321 frequency here
494 frequency here
```

Figure 4 HM-10 to MCU1 USB UART verification



b255 brightness

b000 brightness

b128 brightness

b222 brightness

f262 frequency

f345 frequency

f321 frequency

f494 frequency

Figure 5 Bluetooth Mobile Application Terminal

4.4 Software Flow Chart

The Software design of project 2 required clean flow of data from microcontroller to microcontroller.

4.4.1 MCU1 Flow Chart

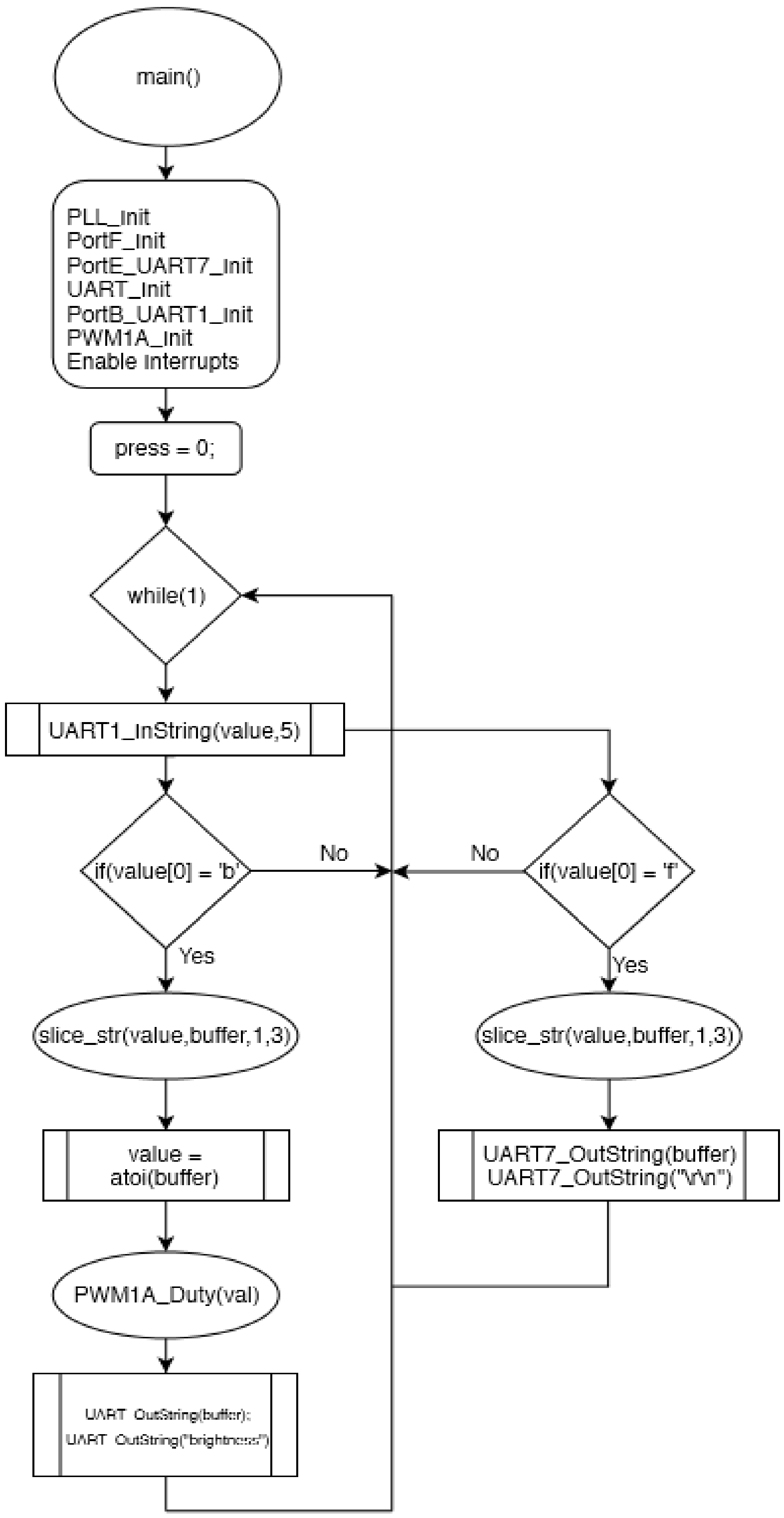


Figure 6 MCU1 Main Flow

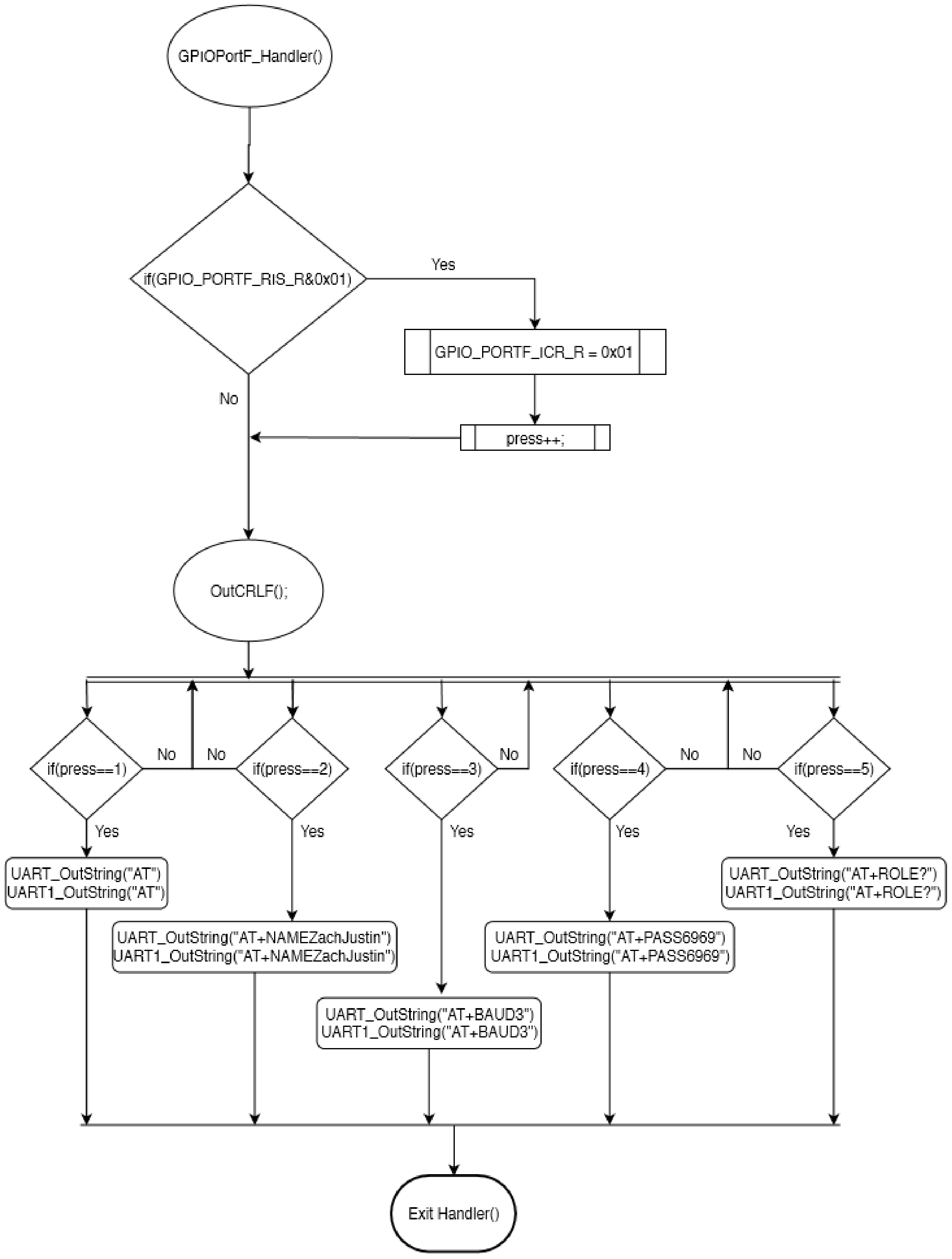

GPIOPortF_Handler()
if(GPIO_PORTF_RIS_R&0x01)
Yes
GPIO_PORTF_ICR_R = 0x01
No
press++;
OutCRLF();
if(press==1)
No
No
if(press==2)
if(press==3)
No
if(press==4)
No
No
if(press==5)
Yes
Yes
Yes
Yes
Yes
UART_OutString("AT")
UART1_OutString("AT")
UART_OutString("AT+NAMEZachJustin")
UART1_OutString("AT+NAMEZachJustin")
UART_OutString("AT+BAUD3")
UART1_OutString("AT+BAUD3")
UART_OutString("AT+PASS6969")
UART1_OutString("AT+PASS6969")
UART_OutString("AT+ROLE?")
UART1_OutString("AT+ROLE?")
Exit Handler()

Figure 7 MCU1 Port F Handler

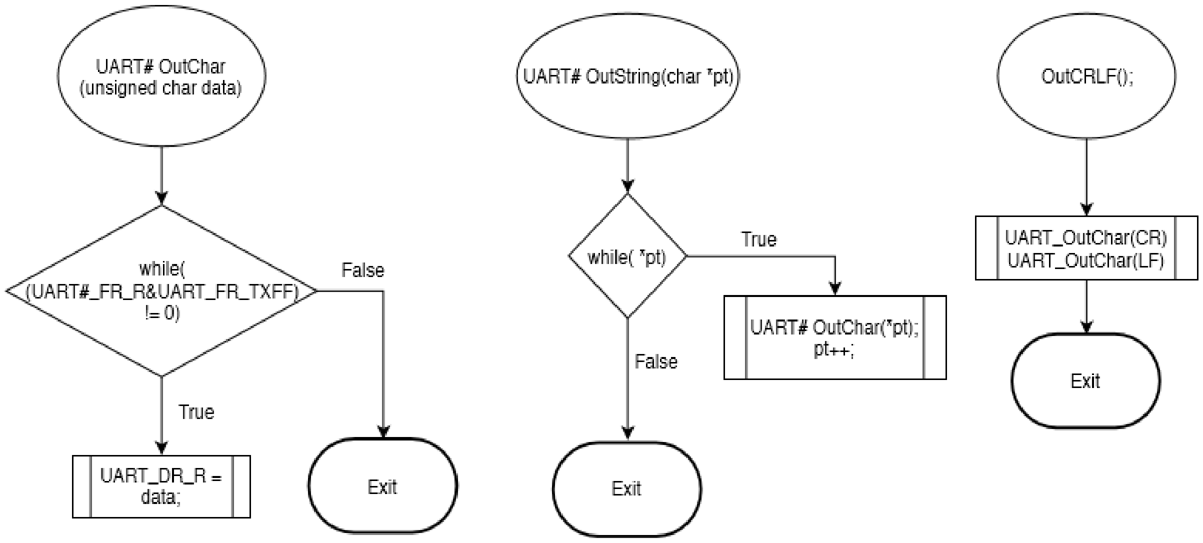


Figure 8 UART Out Functions

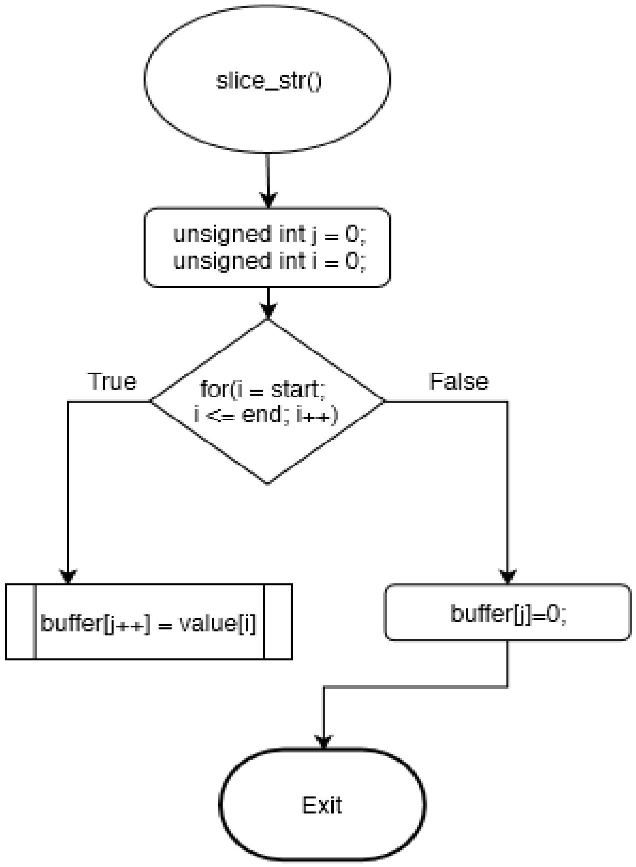


Figure 9 Slice String Function

4.4.2 MCU2 Flow Chart

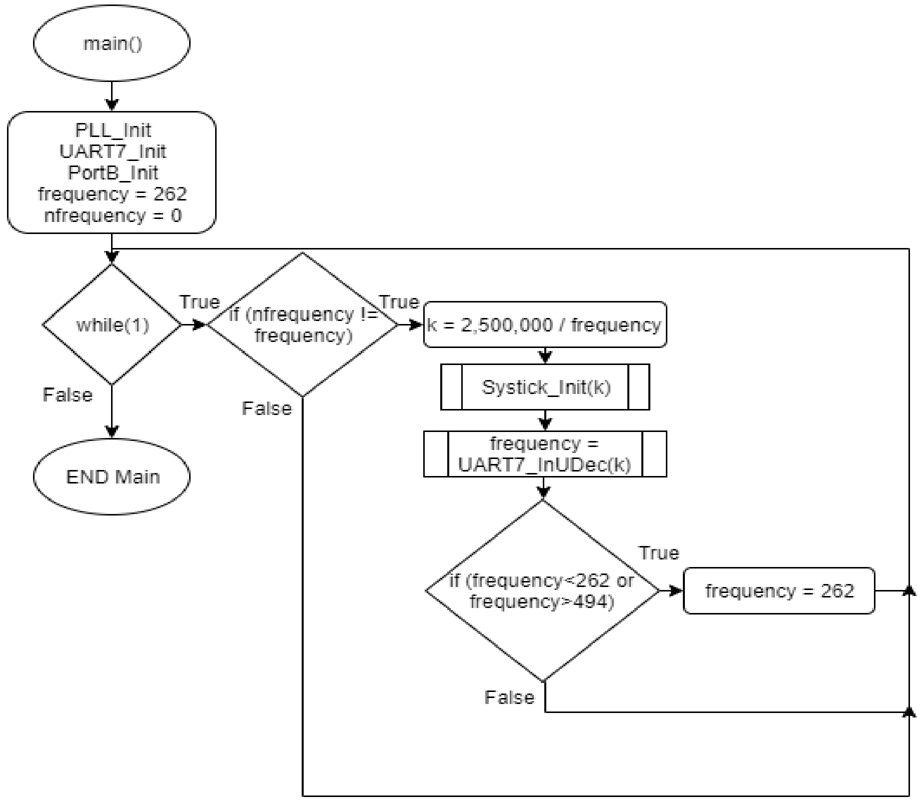


Figure 10 MCU1 Main Flow

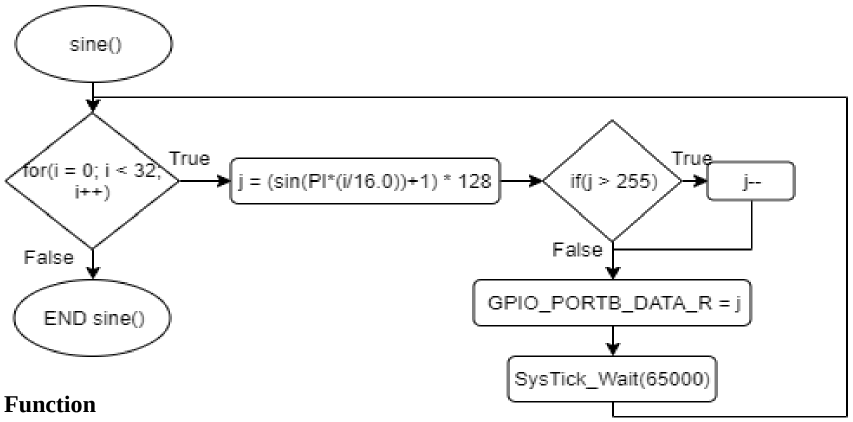


Figure 11 Sine Generation Function

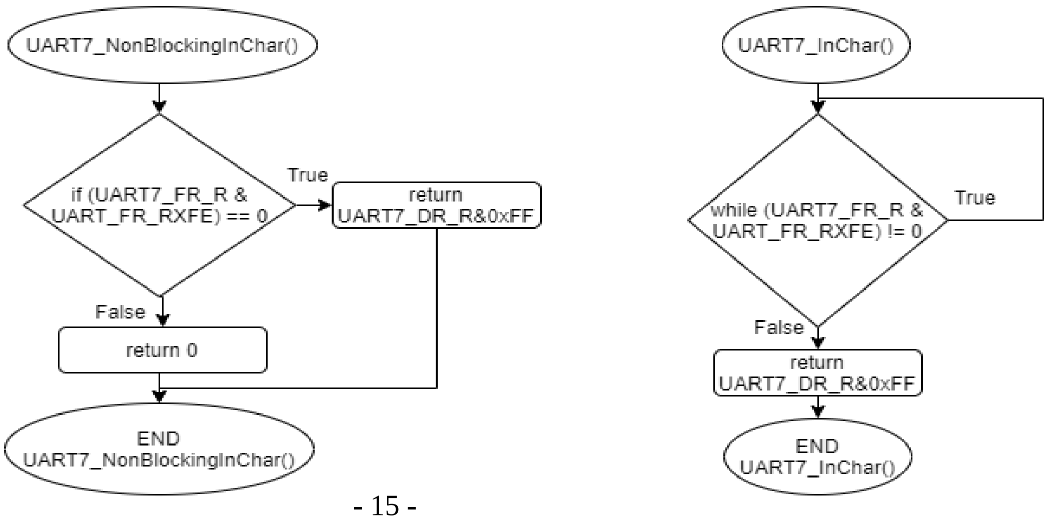

UART7_NonBlockingInChar()
if (UART7_FR_R & UART_FR_RXFE) == 0
True
return UART7_DR_R&0xFF
False
return 0
END UART7_NonBlockingInChar()
UART7_InChar()
while (UART7_FR_R & UART_FR_RXFE) != 0
True
False
return UART7_DR_R&0xFF
END UART7_InChar()

Figure 12 UART In

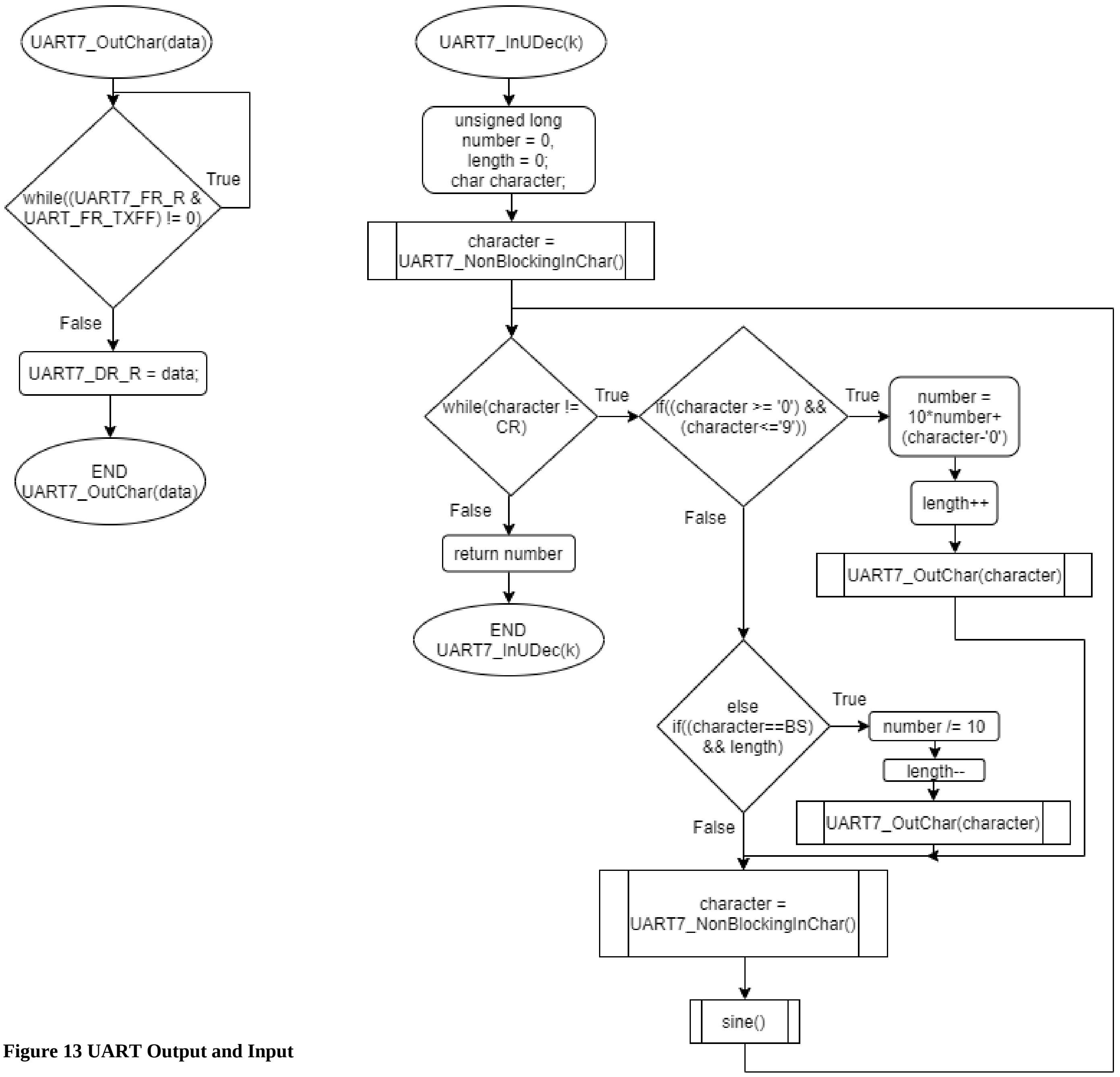

UART7_OutChar(data)
while((UART7_FR_R & UART_FR_TXFF) != 0)
True
False
UART7_DR_R = data;
END UART7_OutChar(data)
UART7_InUDec(k)
unsigned long number = 0, length = 0; char character;
character = UART7_NonBlockingInChar()
while(character != CR)
True
False
return number
END UART7_InUDec(k)
if((character >= '0') && (character<='9'))
True
False
number = 10*number+ (character-'0')
length++
UART7_OutChar(character)
else if((character==BS) && length)
True
False
number /= 10
length--
UART7_OutChar(character)
character = UART7_NonBlockingInChar()
sine()

Figure 13 UART Output and Input

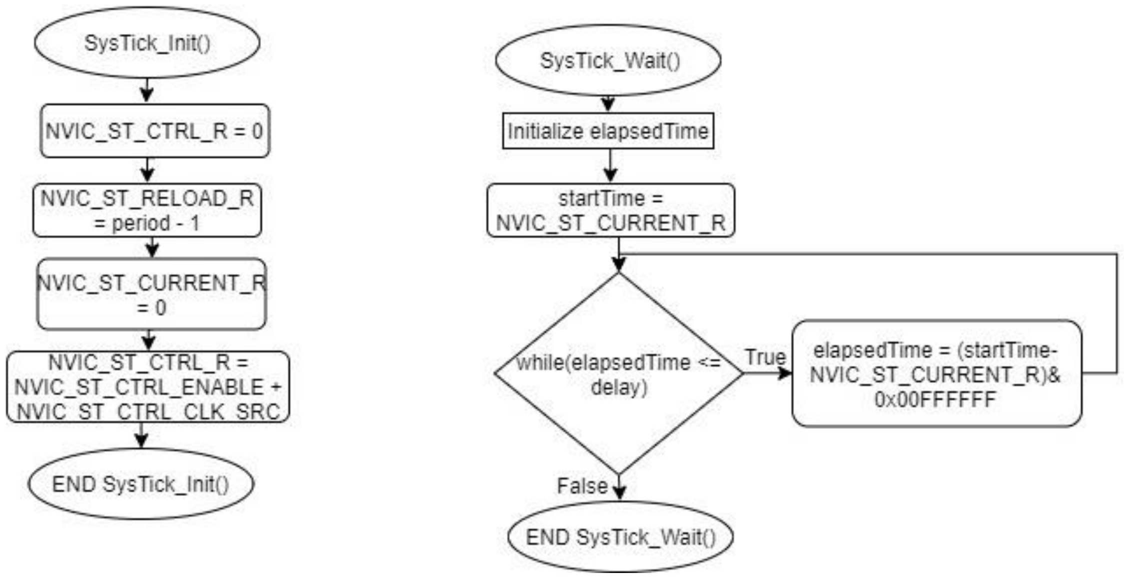

SysTick_Init()
NVIC_ST_CTRL_R = 0
NVIC_ST_RELOAD_R = period - 1
NVIC_ST_CURRENT_R = 0
NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE + NVIC_ST_CTRL_CLK_SRC
END SysTick_Init()
SysTick_Wait()
Initialize elapsedTime
startTime = NVIC_ST_CURRENT_R
while(elapsedTime <= delay)
True
elapsedTime = (startTime-NVIC_ST_CURRENT_R)& 0x00FFFFFF
False
END SysTick_Wait()

Figure 14 SysTick

5. Conclusion

For Project 2 we faced a few challenges along the way. The overall purpose is to create a tone generator using both Digital and Analog components. The user inputs a desired brightness from 0-255 ex – “b245”, or the user can input a desired frequency 262-494 ex – “f494”. This message will be sent via Bluetooth communication to the first microcontroller that will decode the message. If the request is for the brightness to change PWM is utilized to change the brightness of the onboard LED of the first microcontroller. If the request is for frequency, the first microcontroller will send a string of the desired frequency followed by a carriage return to the second microcontroller. The second microcontroller will decode the string frequency into an integer value and pass the calculated value into SysTick Init. The results will be outputted to the DAC and seen on the Oscilloscope where the sine wave will be produced with the desired user inputted frequency. 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 and the 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. We debugged by sending one character over and were eventually able to create a new algorithm to communicate the correct bits.