E155 Final Report

Anuragini Arora, Sam Freisem-Kirov

11/28/2020

Overview:

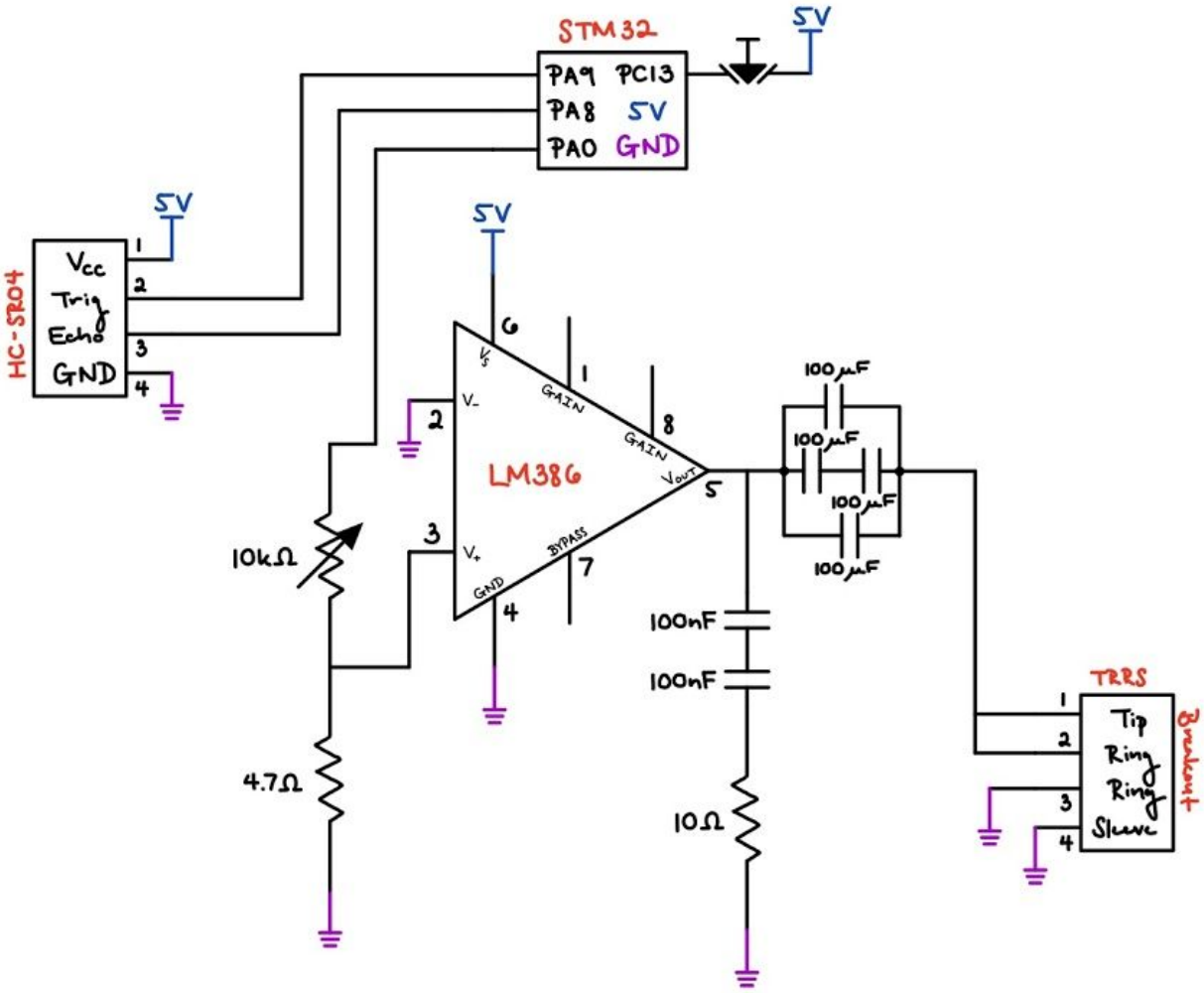
There are around 39,000,000 blind people around the world (World Health Organization). We hope that our project, UltraVisionAid, will have a positive impact on society by helping blind people with activities that require spatial awareness. If our product makes it to production, our hope is that it will make life easier for these people in a cost effective way. These goals influenced the way we approached designing and creating the device. For instance, these goals dictated that our device had to be hand held, battery powered, and relatively inexpensive.

The premise of UltraVisionAid is a spacial awareness aid for blind people that uses auditory cues to help them understand the distance to the objects around them. When the button is pressed, the device plays a tone based on the distance to the nearest object in the direction that the device is pointed in. The tone dynamically changes as the distance changes while the button is pressed.

Our product uses an ultrasonic distance sensor, a button, an amplifier, and an AUX port. The STM32 microcontroller has the non-trivial functionality of sensing when the button is pressed, sending a signal to the distance sensor, receiving the result, and generating a squarewave based on it. And while the button is not being pressed the device stays in sleep mode to save power.

Schematic:

The schematic below depicts all wired connections that will be incorporated for the UltraVisionAid prototype. Pin A0, configured to alternate function for the corresponding timer channel, provides the generated tone signal for the amplification by the LM386, which is connected to the headphone jack that provides the same signal to both the left and the right headphones. Pin A8 and Pin A9 communicate with the ultrasonic ranging module, to transmit the trigger pulse on Pin A9 (configured with output mode) and receive the echo pulse on Pin A8 (configured with input mode) . The LM386 is set up for a gain of 20, and its circuit and pinouts were copied from the data sheet.



Hardware Components:

The Ultrasonic Ranging Module (HC-SR04), referred to as the distance sensor in this paper, has an ultrasonic transmitter and an ultrasonic receiver controlled by a circuit that has four pins to communicate with the STM32. The two power pins must be connected to a 5V supply voltage and GND, and the two pulse pins are a trigger input and an echo output. Essentially, the module requires an input pulse of 10μs, signaling it to transmit an 8-cycle trigger at 40kHz. It receives an echo that is translated (by the internal circuit of the distance sensor) to an output pulse of a duration corresponding to the distance from the sensor to an object. The HC-SR04 has a range from 2cm to 4m; the distance may be calculated by multiplying the duration found by half of the speed of sound. However, because the distance is information that is calculated internally by the device and is not provided to the user, the code for the device directly translates the duration from the distance sensor to a frequency for a generated tone.

The TRRS Breakout (BOB-11570), is essentially a female headphone jack. One source suggested that the Tip, Ring, Ring, and Sleeve structure corresponds to the left headphone, the right headphone, GND, and microphone respectively, with GND and microphone reversed for different manufacturers (Cable Chick). Because the device does not require a microphone and the audio should be the same for both ears, the tip and its adjacent ring are connected to the same output and the sleeve and its adjacent ring are connected to GND.

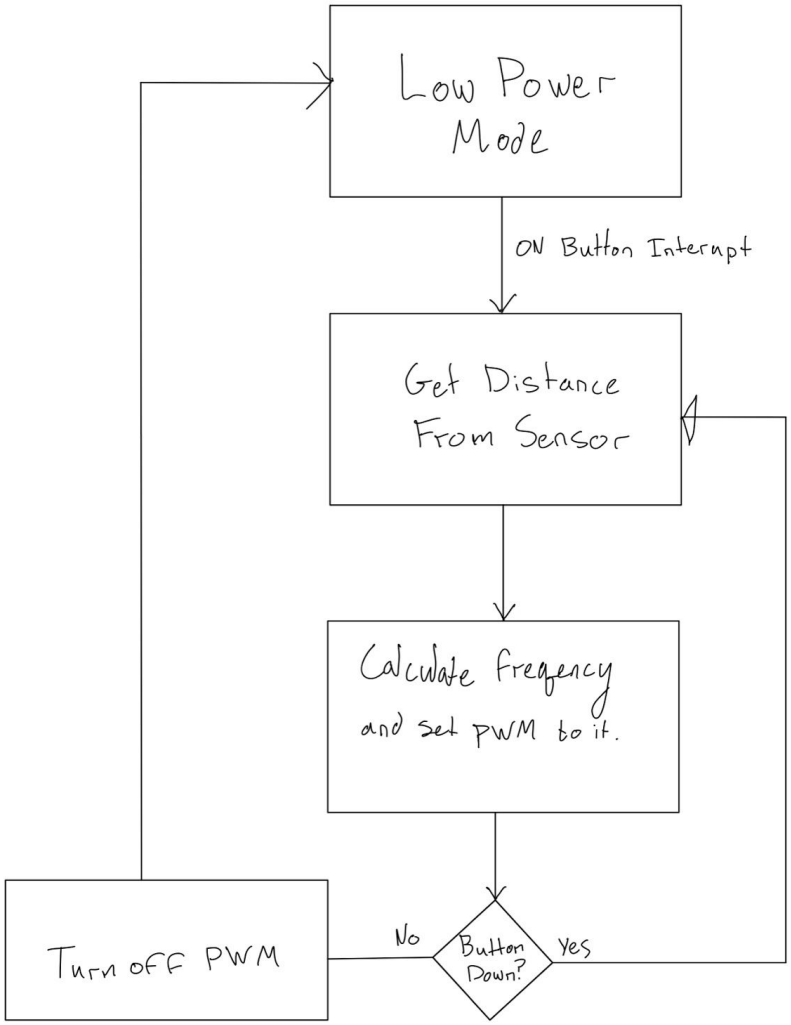
The Low Voltage Audio Power Amplifier (LM386) works in conjunction with the female headphone jack to generate an audible audio output. The setup was based on a circuit provided in the LM386 datasheet, which had a gain of 20.

The device implements a 10kΩ potentiometer to allow volume adjustment to the amplified signal. Testing demonstrated that simply using a 10kΩ potentiometer setup like in the circuit depicted in the LM386 datasheet would not allow sufficient volume adjustment. Dramatic volume reductions occurred at one end of the potentiometer spectrum, which was concluded to be ineffective. This end of the spectrum was analyzed further by measuring voltages achieved at lower volumes to understand how a voltage divider could be used to achieve the correct range. The voltage ranges found indicated that an additional resistor should have a value between 200 and 2000 times

smaller than the value of the potentiometer. Thus, 47Ω and 4.7Ω resistors for voltage deviders were chosen to satisfy the extremes of the condition for testing. The 47Ω resistor did not have the distinct volume change upon turning the knob that the 4.7Ω resistor produced. Thus, the 4.7Ω resistor was added to connect to ground to create a voltage divider with the 10kΩ potentiometer for the final circuit.

The Nucleo development board consists of a button, which when used with an interrupt, makes the device responsive, allowing the user to sense their surroundings whenever desired, in either a continuous or an on-demand manner. In addition, the STM32F401RE provides the device with the desired functionality. The device will be used with a rechargeable battery pack to make it portable, allowing the user to sense their surroundings wherever desired. The USB cable for the Nucleo development board permits the connection for downloading programs from the computer to the STM32F401RE and for receiving power from the rechargeable battery pack. Wires and jumper cables, in addition to the breadboard, enabled the construction of the prototype for the device.

Software Features:



Ultrasonic Distance Sensor Routine:

For the Ultrasonic Ranging Module (HC-SR04), a corresponding C library was created, and it includes two functions, an initialization function and a data collection function. The initialization function assigns the modes for the desired GPIO pins and writes a 0 to the pin that provides the trigger. The data collection function writes a 1 to the pin that provides the trigger and delays for ten microseconds, before writing another 0 to that pin. This is the ten microsecond trigger input pulse that communicates to the Ultrasonic Ranging Module that it should transmit an ultrasound signal and receive an echo. The data collection function waits until the pin measuring the echo is written high,

which cues it to start the timer to measure the duration of the echo pulse. The data collection function waits until the pin measuring the echo is written low, and returns the duration measured in microseconds. This function is called by the main function in a while loop as long as the button is pressed. The value returned is converted to a frequency directly with a simple linear equation to scale the pitch however desired. The coefficients for the linear equation were selected through testing of the device, to ensure a reasonable pitch range for the user.

Button Interrupt Routine:

The button interrupt protocol is incorporated in the main C library. The set up function configures the flash memory and the clock and initializes USART. The button is enabled and the corresponding pin is configured with input mode. The timer and the ultrasonic distancing module are initialized, and the system configuration controller clock domain is enabled. The external interrupt configuration register is set up for the pin corresponding to the button. Interrupts are globally enabled, and the interrupt is configured for the falling edge of the GPIO pin, specifically by enabling the relevant trigger, configuring the mask bit, and turning on the relevant external interrupt in the correct register for the nested vectored interrupt controller. There is an interrupt handler function that ensures that the interrupt is caused by a button press.

Thus, the loop in the main function proceeds when the button is held to allow for continuous detection. The button may be clicked (thus causing an interrupt) for on-demand detection. The overall functionality that has been incorporated is simply that the button pressed for whatever duration causes an interrupt and then enters a loop to repeat data collection, if the button is still being pressed.

Tone Generation Routine:

When the device is powered, the tone is initially set up and plays a constant tone for half a second to signify that the device is prepared for input. PWM is implemented to generate the tones, and all tones use a 50% duty cycle. The tones are generated at specified frequencies depending on the inputs to the tone function. In the button

interrupt handler, after completing the distance sensing routine, the PWM starts with a 50% duty cycle and a frequency calculated for the corresponding distance.

When the device returns to the main loop, while the button is still pressed, it continues to update the PWM frequency, responding to the new data coming in from the distance sensor. When the button is no longer being pressed, the tone function is called to generate a 0Hz square wave, which essentially adjusts the PWM duty cycle to 0%. The function call is a kind of stop command that prevents the previous tone from being generated infinitely.

The tone frequency is calculated with the following equation:

$$Generated\ Frequency\ =$$

$$Max\ Frequency\ (Hz)\ -(\ \frac{Max\ Frequency\ (Hz) - Min\ Frequency\ (Hz)}{Max\ Duration\ (\mu s) - Min\ Duration\ (\mu s)}\ *\ Sampled\ Duration\ (\mu s))$$

Through testing of pitches of tones generated, a maximum output frequency and a minimum output frequency were selected to be comfortably audible to the user's ears. The distance sensor was tested to understand the ranges of microsecond durations sent to the microcontroller after sensing the environment, which led to the selection of a maximum duration and a minimum duration. The equation above is how we translate the duration in microseconds to a frequency in Hertz linearly.

Low Power Mode Routine:

Since our device uses a rechargeable battery pack, and users prefer longer battery life, the user would benefit from the device using less power and extending its battery life on each charge. This feature was achieved by implementing a simple sleep mode on the STM32 microcontroller. The sleep mode uses a WFI() to go to sleep and wait for an interrupt to be generated for it to wake up. Testing found that without a sleep mode implemented, while the button was not pressed, the device drew 30 mA of current. With the sleep mode enabled, while the button was not pressed, the device drew 10.85 mA of current. So this sleep mode routine appears to increase the battery life by 3 times while the button is not being pressed. The button would not be pressed for the majority of time in the use cases that were examined. With a 2600 mAh rechargeable battery pack, the device would last 153 more hours with sleep mode enabled.

Error Handling:

There exist two edge cases in which the ultrasonic distance sensor is unable to make a measurement: (1) the distance is too far for the sensor to receive the echo pulse in the sensor's time limit, or (2) the sensor is blocked by an object that is too close. The implemented edge case handling detects when one of these situations occurs, but it cannot differentiate between them, due to the limitations of the distance sensor. While considering the defined use cases, it appears that most of these errors would occur in the first case, where the distance exceeds 4 meters, so the device outputs a low tone. This tone is slightly lower than any in the output frequency range for sensed distances, so a user would be able to notice that the sensor was not getting an accurate reading because the distance was too far. In use cases that the sensor is blocked by an object that is too close, the user would likely be able to detect this distance themselves, so the generated low tone would not interfere with their understanding of their environment.

Conclusion:

This device is a functioning prototype for a potentially marketable product with an original and innovative idea. It has clear use cases for those who are visually impaired and could viably be taken to market. Further development would be required to improve its marketability. This includes increasing noise control and moving the design from a prototype board to a custom PCB with a plastic case for easy handheld use. User testing and further investigation of the effectiveness of the device in different environments would also be beneficial. We hope that our prototype inspires production of other similar devices that may have a greater impact on society.

Appendix A: main.c and main.h

```
// main.c
// Sam Freisem-Kirov  Anuragini Arora
// sfreisemkirov@hmc.edu
// 11/5/20

#include "main.h"
#include "tone.h"

void setup() {

    configureFlash();
    configureClock();

    initUSART(USART_ID);

    // Enable LED as output
    RCC->AHB1ENR.GPIOAEN = 1;
    pinMode(GPIOA, LED_PIN, GPIO_OUTPUT);

    // Enable button as input
    RCC->AHB1ENR.GPIOCEN = 1;
    pinMode(GPIOC, BUTTON_PIN, GPIO_INPUT);

    // Initialize timer
    RCC->APB1ENR |= (1 << 0); // TIM2EN
    initTIM(DELAY_TIM);

    // Initialize ultrasonic distance sensor
    initSensor(INPIN,OUTPIN); //8,9


    // 1. Enable SYSCFG clock domain in RCC
    RCC->APB2ENR |= (1 << 14);
    // 2. Set EXTICR4 for PC13
    *SYSCFG_EXTICR4 |= (0b0010 << 4*1);

    // Enable interrupts globally
    __enable_irq();

    // Configure interrupt for falling edge of GPIO PC13
    // 1. Configure mask bit
    EXTI->IMR |= (1 << BUTTON_PIN);
```

```
    // 2. Disable rising edge trigger
    EXTI->RTSR &= ~(1 << BUTTON_PIN);

    // 3. Enable falling edge trigger
    EXTI->FTSR |= (1 << BUTTON_PIN);

    // 4. Turn on EXTI interrupt in NVIC_ISER1
    *NVIC_ISER1 |= (1 << 8);


}

int main(void) {
    setup();
    tone(500,1000);
    while(1){
        delay_millis(DELAY_TIM, 200);

        int count = 0;
        int prevDists[7] = {0,0,0,0,0,0,0};
        int sum = 0;
        int i = 0;

        while(!digitalRead(GPIOC, BUTTON_PIN)){
            int dist = getDistance(INPIN, OUTPIN);
            sum -= prevDists[i];
            prevDists[i] = dist;
            i = (i+1) % 7;
            sum+= dist;
            if (count < 7) count++;
            int avgDist = sum/count;
            int freq = (880.0-(((660.0/18000.0)*avgDist)));

            tone(freq,60);
        }

        tone(0,60);
        // while button is being pressed
            // check distance
            // emit tone on that distance for a quarter of a second
            // when the button is not pressed, stop the tone

        __WFI();
```

```
    }
}


void EXTI15_10_IRQHandler(void){
        // Check that the button EXTI_13 was what triggered our interrupt
    if (EXTI->PR & (1 << BUTTON_PIN)){
        // If so, clear the interrupt
        EXTI->PR |= (1 << BUTTON_PIN);


        int val = getDistance(INPIN, OUTPIN);
        int dist = getDistance(INPIN, OUTPIN);
        int freq = (880.0-(((660.0/18000.0)*dist)));
        tone(freq,60);
    }
}

// Use For Testing
void printDist(int val) {
    uint8_t msg[64];

    sprintf(msg, " %d \n\r",val);

    uint8_t i = 0;
    do
    {
        sendChar(USART_ID, msg[i]);
        i += 1;
    } while (msg[i]);
}
```

```
// main.h
// Sam Freisem-KirovK, Anuragini Arora
// sfreisemkirov@hmc.edu
// 11/28/20

#ifndef MAIN_H
#define MAIN_H

//#include "stm32f4xx.h"
#include "STM32F401RE.h"
#include "HCSR04LIB.h"
```

```c
#include <string.h>
#include <stdint.h>


////////////////////////////////////////////////////////////////////////////////
// Custom defines
////////////////////////////////////////////////////////////////////////////////

#define LED_PIN 5
#define BUTTON_PIN 13 // PC13
#define DELAY_TIM TIM2
#define USART_ID USART2_ID
#define INPIN 8
#define OUTPIN 9

#define NVIC_ISER0 ((uint32_t *) 0xE000E100UL)
#define NVIC_ISER1 ((uint32_t *) 0xE000E104UL)
#define SYSCFG_EXTICR4 ((uint32_t *) (0x40013800UL + 0x14UL))

typedef struct {
    volatile uint32_t IMR;
    volatile uint32_t EMR;
    volatile uint32_t RTSR;
    volatile uint32_t FTSR;
    volatile uint32_t SWIER;
    volatile uint32_t PR;
}EXTI_TypeDef;

#define EXTI ((EXTI_TypeDef *) 0x40013C00UL)

////////////////////////////////////////////////////////////////////////////////
// IRQn_Type and __NVIC_PRIO_BITS from stm32f401xe.h
////////////////////////////////////////////////////////////////////////////////

/**
 * @brief STM32F4XX Interrupt Number Definition, according to the selected device
 *        in @ref Library_configuration_section
 */
typedef enum
{
/******  Cortex-M4 Processor Exceptions Numbers
******************************************************************/
  NonMaskableInt_IRQn         = -14,    /*!< 2 Non Maskable Interrupt
*/
```

```
  MemoryManagement_IRQn       = -12,    /*!< 4 Cortex-M4 Memory Management
Interrupt                              */
  BusFault_IRQn               = -11,    /*!< 5 Cortex-M4 Bus Fault Interrupt
*/
  UsageFault_IRQn             = -10,    /*!< 6 Cortex-M4 Usage Fault Interrupt
*/
  SVCall_IRQn                 = -5,     /*!< 11 Cortex-M4 SV Call Interrupt
*/
  DebugMonitor_IRQn           = -4,     /*!< 12 Cortex-M4 Debug Monitor Interrupt
*/
  PendSV_IRQn                 = -2,     /*!< 14 Cortex-M4 Pend SV Interrupt
*/
  SysTick_IRQn                = -1,     /*!< 15 Cortex-M4 System Tick Interrupt
*/
/******  STM32 specific Interrupt Numbers
*******************************************************************/
  WWDG_IRQn                   = 0,      /*!< Window WatchDog Interrupt
*/
  PVD_IRQn                    = 1,      /*!< PVD through EXTI Line detection
Interrupt                          */
  TAMP_STAMP_IRQn             = 2,      /*!< Tamper and TimeStamp interrupts
through the EXTI line              */
  RTC_WKUP_IRQn               = 3,      /*!< RTC Wakeup interrupt through the EXTI
line                             */
  FLASH_IRQn                  = 4,      /*!< FLASH global Interrupt
*/
  RCC_IRQn                    = 5,      /*!< RCC global Interrupt
*/
  EXTI0_IRQn                  = 6,      /*!< EXTI Line0 Interrupt
*/
  EXTI1_IRQn                  = 7,      /*!< EXTI Line1 Interrupt
*/
  EXTI2_IRQn                  = 8,      /*!< EXTI Line2 Interrupt
*/
  EXTI3_IRQn                  = 9,      /*!< EXTI Line3 Interrupt
*/
  EXTI4_IRQn                  = 10,     /*!< EXTI Line4 Interrupt
*/
  DMA1_Stream0_IRQn           = 11,     /*!< DMA1 Stream 0 global Interrupt
*/
  DMA1_Stream1_IRQn           = 12,     /*!< DMA1 Stream 1 global Interrupt
*/
  DMA1_Stream2_IRQn           = 13,     /*!< DMA1 Stream 2 global Interrupt
*/
```

```
  DMA1_Stream3_IRQn           = 14,     /*!< DMA1 Stream 3 global Interrupt
*/
  DMA1_Stream4_IRQn           = 15,     /*!< DMA1 Stream 4 global Interrupt
*/
  DMA1_Stream5_IRQn           = 16,     /*!< DMA1 Stream 5 global Interrupt
*/
  DMA1_Stream6_IRQn           = 17,     /*!< DMA1 Stream 6 global Interrupt
*/
  ADC_IRQn                    = 18,     /*!< ADC1, ADC2 and ADC3 global Interrupts
*/
  EXTI9_5_IRQn                = 23,     /*!< External Line[9:5] Interrupts
*/
  TIM1_BRK_TIM9_IRQn          = 24,     /*!< TIM1 Break interrupt and TIM9 global
interrupt                    */
  TIM1_UP_TIM10_IRQn          = 25,     /*!< TIM1 Update Interrupt and TIM10
global interrupt                */
  TIM1_TRG_COM_TIM11_IRQn     = 26,     /*!< TIM1 Trigger and Commutation
Interrupt and TIM11 global interrupt */
  TIM1_CC_IRQn                = 27,     /*!< TIM1 Capture Compare Interrupt
*/
  TIM2_IRQn                   = 28,     /*!< TIM2 global Interrupt
*/
  TIM3_IRQn                   = 29,     /*!< TIM3 global Interrupt
*/
  TIM4_IRQn                   = 30,     /*!< TIM4 global Interrupt
*/
  I2C1_EV_IRQn                = 31,     /*!< I2C1 Event Interrupt
*/
  I2C1_ER_IRQn                = 32,     /*!< I2C1 Error Interrupt
*/
  I2C2_EV_IRQn                = 33,     /*!< I2C2 Event Interrupt
*/
  I2C2_ER_IRQn                = 34,     /*!< I2C2 Error Interrupt
*/
  SPI1_IRQn                   = 35,     /*!< SPI1 global Interrupt
*/
  SPI2_IRQn                   = 36,     /*!< SPI2 global Interrupt
*/
  USART1_IRQn                 = 37,     /*!< USART1 global Interrupt
*/
  USART2_IRQn                 = 38,     /*!< USART2 global Interrupt
*/
  EXTI15_10_IRQn              = 40,     /*!< External Line[15:10] Interrupts
*/
```

```
  RTC_Alarm_IRQn              = 41,     /*!< RTC Alarm (A and B) through EXTI Line
Interrupt                    */
  OTG_FS_WKUP_IRQn            = 42,     /*!< USB OTG FS Wakeup through EXTI line
interrupt                     */
  DMA1_Stream7_IRQn           = 47,     /*!< DMA1 Stream7 Interrupt
*/
  SDIO_IRQn                   = 49,     /*!< SDIO global Interrupt
*/
  TIM5_IRQn                   = 50,     /*!< TIM5 global Interrupt
*/
  SPI3_IRQn                   = 51,     /*!< SPI3 global Interrupt
*/
  DMA2_Stream0_IRQn           = 56,     /*!< DMA2 Stream 0 global Interrupt
*/
  DMA2_Stream1_IRQn           = 57,     /*!< DMA2 Stream 1 global Interrupt
*/
  DMA2_Stream2_IRQn           = 58,     /*!< DMA2 Stream 2 global Interrupt
*/
  DMA2_Stream3_IRQn           = 59,     /*!< DMA2 Stream 3 global Interrupt
*/
  DMA2_Stream4_IRQn           = 60,     /*!< DMA2 Stream 4 global Interrupt
*/
  OTG_FS_IRQn                 = 67,     /*!< USB OTG FS global Interrupt
*/
  DMA2_Stream5_IRQn           = 68,     /*!< DMA2 Stream 5 global interrupt
*/
  DMA2_Stream6_IRQn           = 69,     /*!< DMA2 Stream 6 global interrupt
*/
  DMA2_Stream7_IRQn           = 70,     /*!< DMA2 Stream 7 global interrupt
*/
  USART6_IRQn                 = 71,     /*!< USART6 global interrupt
*/
  I2C3_EV_IRQn                = 72,     /*!< I2C3 event interrupt
*/
  I2C3_ER_IRQn                = 73,     /*!< I2C3 error interrupt
*/
  FPU_IRQn                    = 81,     /*!< FPU global interrupt
*/
  SPI4_IRQn                   = 84      /*!< SPI4 global Interrupt
*/
} IRQn_Type;

#define __NVIC_PRIO_BITS          4U       /*!< STM32F4XX uses 4 Bits for the
Priority Levels */
```

```
#include "cmsis_gcc.h"
#include "core_cm4.h"

void setup();

#endif // MAIN_H
```

Appendix B: tone.c and tone.h

```
// Standard library includes.
#include <stdint.h>
#include <stdlib.h>

#include "STM32F401RE.h"

void tone(int freq, int delay) {
    // PA0 connected to LM386
    // set mode to alt func
    pinMode(GPIOA,0,2);

    // set alt func lower register to AF02 to match TIM5_CH1 output
    // GPIOA->AFRL.AFRL0 = 0b0010;
    GPIOA->AFRL &= ~(0b11 << 2); // 00 in bits 3:2
    GPIOA->AFRL |= (0b1 << 1); // 1 in bit 1
    GPIOA->AFRL &= ~(0b1 << 0); // 0 in bit 0


    int duty = 50; // sets duty cycle to constant 50%
    int arr = 255; // ARR configured for 8-bit resolution
    int psc; // PSC with default value
    if (freq > 0) { // if frequency given is greater than zero,
        psc = (int)((84000000/((arr+1)*freq))); // then calculates PSC using ARR
    } else {
        duty = 0; // otherwise, sets duty cycle to 0% for no tone
        psc = 1; // sets PSC to 1
    }

    configureTIM5(psc,arr,duty); // configures TIM5 to produce tone at pitch given
by frequency
    delay_millis(TIM2, delay); // delays for given duration in milliseconds using
TIM2
    // configureTIM5(psc,arr,0);
}

void configureTIM5(int psc, int arr, int duty) {

    // Disable slave mode
    // TIM5->SMCR.SMS = 0;
    TIM5->SMCR &= ~(0b111 << 0);

    // Enable timer
```

```
    // RCC->APB1ENR.TIM5EN = 1;
    RCC->APB1ENR |= (0b1 << 3);

    TIM5->PSC = psc;
    TIM5->ARR = arr;
    TIM5->CCR1 = (int)((arr)*duty/100);

    // Select PWM mode 1 on channel 1
    // TIM5->CCMR1.OC1M = 0b110;
    TIM5->CCMR1 |= (0b11 << 5);
    TIM5->CCMR1 &= ~(0b1 << 4);

    // Enable preload register
    // TIM5->CCMR1.OC1PE = 1;
    TIM5->CCMR1 |= (0b1 << 3);

    // Enable fast register
    // TIM5->CCMR1.OC1FE = 1;
    TIM5->CCMR1 |= (0b1 << 2);

    // Enable auto-reload preload register
    // TIM5->CR1.ARPE = 1;
    TIM5->CR1 |= (0b1 << 7);

    // Initialize registers
    // TIM5->EGR.UG = 1;
    TIM5->EGR |= (0b1 << 0);

    // Set OC1 polarity to active high
    // TIM5->CCER.CC1P = 0;
    TIM5->CCER &= ~(0b1 << 1);

    // Enable OC1 output
    // TIM5->CCER.CC1E = 1;
    TIM5->CCER |= (0b1 << 0);

    // Activate upcounting
    // TIM5->CR1.DIR = 0;
    TIM5->CR1 &= ~(0b1 << 4);

    // Select CK_INT as counter clock source
    // TIM5->CR1.CEN = 1;
    TIM5->CR1 |= (0b1 << 0);
}
```

```c
// tone.h
// Sam Freisem-Kirov  Anuragini Arora
// sfreisemkirov@hmc.edu
// 11/5/20

#ifndef TONE_H
#define TONE_H

void tone(int freq, int delay);
void configureTIM5(int psc, int arr, int duty);

#endif // TONE_H
```

Appendix C: HCSR04LIB.c and HCSR04LIB.h

```
// HCSR04LIB.c
// Sam Freisem-Kirov  Anuragini Arora
// sfreisemkirov@hmc.edu
// 11/7/20

#include "HCSR04LIB.h"

void initSensor(int inPin, int outPin) {
    // initialize the GPIO pins that we are using for the ultrasonic distance
sensor
    pinMode(GPIOA,inPin,GPIO_INPUT);
    pinMode(GPIOA, outPin, GPIO_OUTPUT);
    digitalWrite(GPIOA, outPin, GPIO_LOW);
}

int getDistance(int inPin, int outPin) {
    // send a trigger pulse
    digitalWrite(GPIOA, outPin, GPIO_HIGH);
    delay_micros(DELAY_TIM, 10);
    digitalWrite(GPIOA, outPin, GPIO_LOW);

    // read the output by setting a timer to count until the echo pulse is low
    while(digitalRead(GPIOA, inPin) == GPIO_LOW);
    start_count(TIM2);

    while(digitalRead(GPIOA, inPin) == GPIO_HIGH);
    int count = get_count_micros(TIM2);

    return count;

}
```

```
// HCSR04LIB.h
// Sam Freisem-Kirov  Anuragini Arora
// sfreisemkirov@hmc.edu
// 11/7/20

#ifndef HCSR04LIB_H
```

```c
#define HCSR04LIB_H

#include "STM32F401RE.h"

#define DELAY_TIM TIM2

void initSensor(int inPin, int outPin);
int getDistance(int inPin, int outPin);

#endif
```

Appendix D: Additions to STM32F401RE_TIM.c

```c
// STM32F401RE_TIM.c
// TIM functions

#include "STM32F401RE_TIM.h"
#include "STM32F401RE_RCC.h"

void initTIM(TIM_TypeDef * TIMx){
  uint32_t psc_div = (uint32_t) ((SystemCoreClock/1e6)-1);

  // Set prescaler division factor
  TIMx->PSC = (psc_div - 1);
  // Generate an update event to update prescaler value
  TIMx->EGR |= 1;
  // Enable counter
  TIMx->CR1 |= 1; // Set CEN = 1
}

void delay_millis(TIM_TypeDef * TIMx, uint32_t ms){
  TIMx->ARR = ms*1000;// Set timer max count
  TIMx->EGR |= 1;     // Force update
  TIMx->SR &= ~(0x1); // Clear UIF
  TIMx->CNT = 0;      // Reset count

  while(!(TIMx->SR & 1)); // Wait for UIF to go high
}

void delay_micros(TIM_TypeDef * TIMx, uint32_t us){
  TIMx->ARR = us;// Set timer max count
  TIMx->EGR |= 1;     // Force update
  TIMx->SR &= ~(0x1); // Clear UIF
  TIMx->CNT = 0;      // Reset count

  while(!(TIMx->SR & 1)); // Wait for UIF to go high
}

void start_count(TIM_TypeDef * TIMx) {
  TIMx->ARR = 18000;// Set timer max count
  TIMx->EGR |= 1;     // Force update
  TIMx->SR &= ~(0x1); // Clear UIF
  TIMx->CNT = 0;      // Reset count
}
```

```
int get_count_micros(TIM_TypeDef * TIMx) {
  if(TIMx->SR & 1) {
    return 18000;
  }
  return TIMx->CNT;
}
```

Works Cited

Cable Chick, and www.cablechick.com.au. “Understanding TRRS and Audio Jacks.” *Cable Chick*, www.cablechick.com.au/blog/understanding-trrs-and-audio-jacks/.

Elec Freaks. “Ultrasonic Ranging Module HC - SR04.” *Sparkfun*, cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.

“Global Data on Visual Impairment.” *World Health Organization*, World Health Organization, 8 Dec. 2017, www.who.int/blindness/publications/globaldata/en/.

“LM386 Low Voltage Audio Power Amplifier.” *Texas Instruments*, May 2017.

Saddam. “Headphone/Audio Amplifier Circuit on PCB Using LM386.” *Circuit Digest*, 5 Dec. 2017, circuitdigest.com/electronic-circuits/headphone-amplifier-circuit-on-pcb.

“SparkFun TRRS 3.5mm Jack Breakout.” *BOB-11570 - SparkFun Electronics*, www.sparkfun.com/products/11570.