Alien Voice Filter

Final Project Report

December 1, 2020

E155

Dithi Ganjam and Rebecca Qin

ABSTRACT

The goal of this project was to build a voice filter that would allow users to record a snippet of their voice and play it back with three filters: one that increases the frequency of the voice, one that decreases the speed of the voice, and one that modifies the frequency to make it sound like an "alien" voice. The user interface for the voice filter is an online webpage. Our device consists of a microphone, a microcontroller, a DAC, a speaker, and a WiFi module. The user's voice input into the microphone is sampled and stored in the microcontroller. The audio is then modified and outputted to the DAC according to the user's request. The output of the DAC is then amplified and then sent to the speaker.

I. INTRODUCTION

Audio filters are entertaining features that are used all over social media nowadays. We can see their functionality in apps like Snapchat, TikTok, etc. The motivation for this project was to apply our knowledge of STM32F401RE functionality to recreate a voice-filtering web page of our own. We sought out to design a webpage that allows the user to record their voice and play the audio clip back with four main options: regular playback (no filter), slow playback, sped-up playback, and an alien voice using amplitude modulation [1].

This project entails reading voice input from a microphone into memory on the microcontroller using the ADC feature of the STM32F401RE, processing the audio input based on user input implemented via an IoT interface, and outputting the modified audio signal to a speaker to play it back. The block diagram in Figure 1 below outlines the general flow of the system and the communication between the microcontroller and the various peripherals.

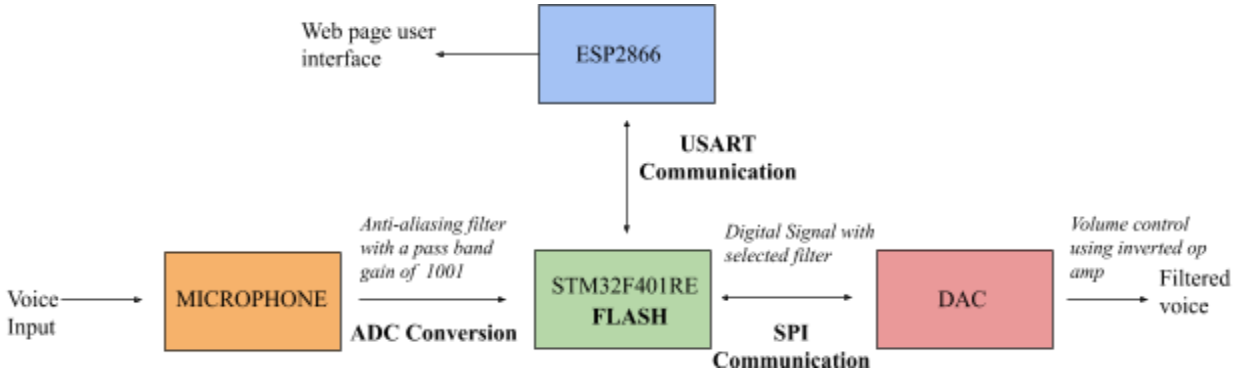

Figure 1. Top-level block diagram of our voice filter, which includes the STM32 interfacing with various peripherals.

II. NEW HARDWARE

The project contains only one new piece of hardware, which is the CMA-4544PF-W microphone. The microphone takes in an audio signal and outputs a voltage signal that can be fed into the ADC to store the user's voice data. While the microphone itself was the only new piece of hardware implemented, it required extensive signal processing circuitry to generate an intelligible voice signal from the microphone output. Before the output of the microphone was sent into the ADC, it was first passed through an anti-aliasing filter with a cutoff frequency of 20 kHz (to capture the frequency range of voice) and to avoid aliasing in accordance with the Shannon-Nyquist sampling theorem, as we sample at a rate of 40 kHz. It was then passed through a differential amplifier to reduce the offset voltage and ensure that the integrity of the signal was not compromised (or clipped on the top). The set-up of the input to the ADC, including the details of the anti-aliasing filter and its circuitry, can be found in Section III. Once the voice signal is sent into the ADC, it is converted into a digital signal and stored into memory.

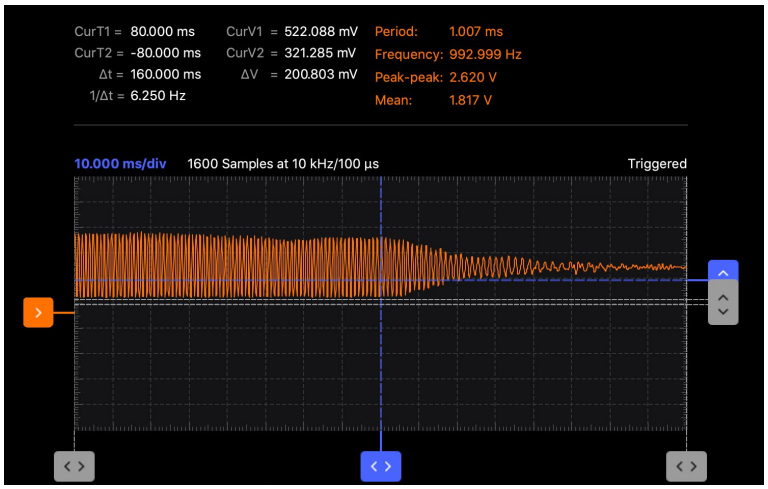

Figure 2. Example of microphone output of a scream (post analog processing)

III. SCHEMATICS

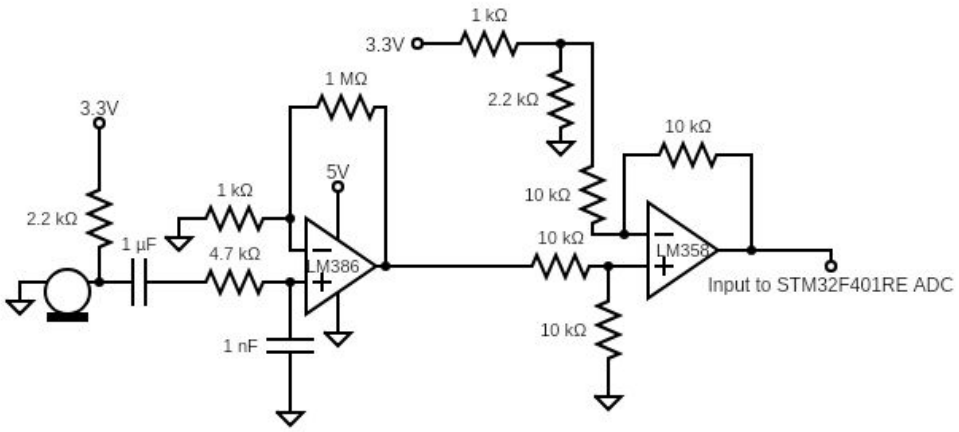

Figure 3. Butterworth Low Pass Filter (with cutoff Frequency 40kHz) and Differential Amplifier circuit for amplification and offset (to avoid signal clipping)

The circuit in figure 3 demonstrates the analog filtration circuitry that was necessary to convert the microphone output signal into an intelligible digital voice signal that could be stored and recreated. The output of the microphone first passed through a low pass butterworth filter with a cutoff frequency of 40kHz, and a band pass gain of 1001. The original voltage swing of the microphone output was on the order of a few millivolts, so the band pass gain allowed for the oscillation in the voice signal to be amplified to the extent that it could be accurately captured and preserved as a voltage level (or digital signal). However, in order to ensure that the voltage swing was large enough, the gain factor led to the signal being positively offset relative to ground. Due to the large gain factor coupled with the offset, the signal was beginning to be clipped on the top by the maximum read voltage of the ADC which is 3.6V. Therefore, the signal is also filtered using a differential amplifier, which brings down the offset of the amplified signal

so that even with a 2 to 3V voltage swing, the input signal is not clipped by the maximum voltage of ADC input.

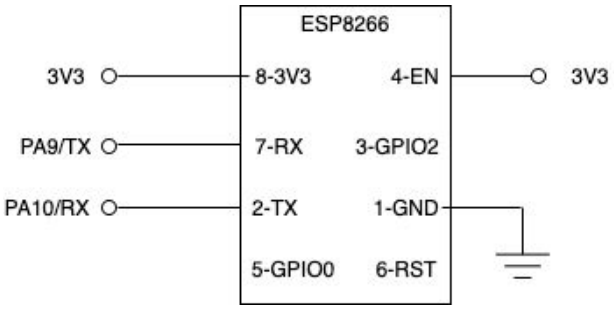

Figure 4. ESP8266 Circuit configuration for Serving the webpage

Figure 4 shows the schematic of the physical interface between the ESP8266 Wifi module and the STM32F401RE microcontroller. The transmit pin, or TX, of the microcontroller goes to the receive pin, or RX, of the ESP chip and vice versa enabling USART communication. The chip enable is tied to 3.3V so that the wifi module knows to be able to receive and send data to the microcontroller.

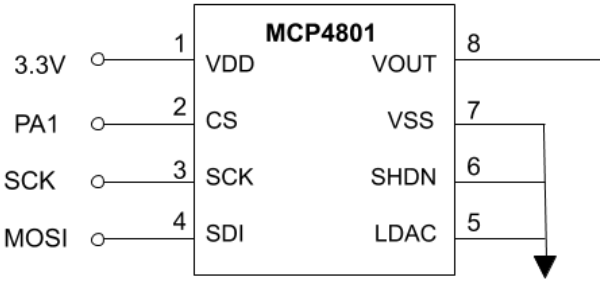

Figure 5. MCP4801 Circuit configuration between the DAC and the STM32F401RE

The figure above shows the interface between the digital to analog converter (DAC) and the microcontroller. The input to SCK on the DAC is controlled by a timer in order to output the signal at the appropriate frequency so the integrity of the signal is maintained. The chip select is

controlled by GPIOA pin 1, and the discrete voltage signal is sent to the MCP4801 from the MOSI of the microcontroller to the serial data input of the DAC. The voltage swing of the output of the DAC is approximately 2V, while for most 8 Ohm speakers, a reasonably loud audio output is around 8V. For this reason, the analog output of the MCP480, from pin 8, is fed through to the audio amplification circuit outlined in figure 6 below.

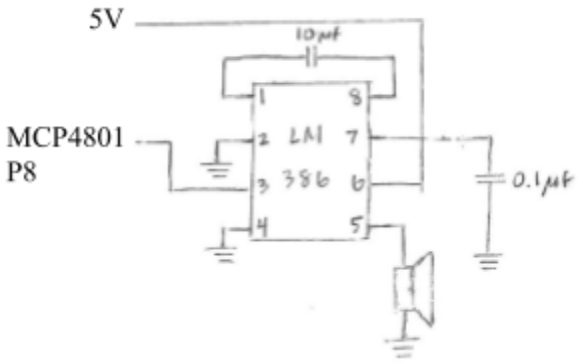

Figure 6. Audio Amplification circuit using LM386

Lastly, a simple green LED was wired to GPIOA 8 and grounded to inform the user of when the microcontroller was recording, and when it had stopped.

IV. MICROCONTROLLER DESIGN

The microcontroller design for handling the audio input is as follows. The audio input from the microphone is retrieved using the built-in ADC in single-conversion mode. A timer controls the sampling of the audio so that when the timer generates an interrupt, the ADC is turned on and captures one data point of the audio signal. The flash memory of the microcontroller is used to store the audio data. Specifically, sectors 6 and 7 are used in order to avoid writing data to sectors that might be used for code [2]. Each of the two sectors used are 128 kbytes, resulting in a total of 256 kbytes, and a storage capacity of about 130,000 audio data points, where each point is a 12-bit positive integer. We originally planned on using the DMA to transfer data from the ADC data register to the flash memory, but after looking further into this, we found that using DMA would not improve the efficiency of the data transfer and might even decrease the efficiency, so we decided against implementing this feature.

Next, we outline the microcontroller design for outputting the audio. The SPI is initialized to send the 12-bit data from the microcontroller to the DAC. Another timer is initialized in a similar way as the sampling timer to output the audio data at the correct frequency. The auto-reload register value for the timer initialized changes according to the audio filter selected. For the alien filter, a function manipulates the audio data point dynamically, as it is being outputted through the SPI. For the other playback options, the data point is unmodified as it is sent through the SPI.

Finally, we outline the microcontroller design for rendering the webpage for the user interface. Two USARTs are initialized: one for communicating with the ESP8266 WiFi module, and one for sending the data received from the ESP8266 to the serial monitor. To initialize the ESP, the proper AT commands are sent using the ESP USART. Then, the webpage is rendered

by sending the HTML through the ESP USART when there is a request through the ESP's IP address. The serial monitor USART is used to check that the ESP is functioning and responding to the commands.

V. RESULTS

The completed project was partially successful. Due to unforeseen issues with the computer that the microcontroller was programmed with, the hardware malfunctioned. The device was thus unable to gather an audio input from the microphone. Before the hardware malfunction, each of the individual components of the high-level design was working separately (1) the webpage controlling the recording and playback functionality, though not yet in flash memory, and (2) writing to flash memory, and being able to apply the different voice filters, though not yet controlled by the IoT interface. Before we could interface our two implementations, unfortunately our hardware failed. However, we were successfully able to design and implement our four voice filters. We were able to meet most of the requirements outlined in our project proposal, but since essential hardware for our system broke, we are unable to demonstrate a working voice filter.

The main challenge that we faced was with using the ESP8266 WiFi module along with the flash memory. In the beginning, we attempted to use sectors 1-7 of flash memory. Upon implementing this and running the code on the microcontroller, we discovered that the ESP had stopped being functional. This may have been from our program in the flash memory being overwritten by the audio values. Although we did not determine the reason for the ESP's loss of function, once we reprogrammed our code to use only sectors 6 and 7, a replacement ESP was able to function correctly.

There is much room for improvement in our project design. Due to the lack of resources from remote learning, we lost much time from needing to order small parts that needed to be replaced. One stretch goal that we would have liked to implement is an additional autotuning filter, which would require looking into fast Fourier transforms (FFTs) to modify the audio data.

VI. REFLECTION

Though we could not complete our project due to unforeseen hardware issues, working through the project both together and remotely was an important learning experience. Overall, we were not prepared in advance for our hardware to fail at any point (resulting in not having extra components). In future projects, similar outcomes may be avoided by ordering extra components in the event that anything goes wrong. Another example of such a hurdle is that our only 1 out of 5 ESP8266 Wifi modules was functioning appropriately. Initially, when the ESP8266 chips were not functioning as expected, we experimented with an external power supply for increased reliability, and a variety of baud rates. However, later upon plugging it into the logic analyzer on the SCOPY digital oscilloscope, we realized that the ESP was not communicating in response to the microcontroller, as we could only see bits communicated from the TX pin of the microcontroller, and none received on the RX pin. However, for the 5th ESP8266 that we tried, we were able to observe communication between the ESP and the microcontroller, and also serve the webpage for our IoT interface. While the project itself was incomplete as new hardware could not be ordered in time before the final due date, working through our software issues allowed us to better understand the inner workings of flash memory on STM32F401RE microcontroller. Though we encountered issues along the way, the project taught us the importance of debugging and understanding how different components interact with one another (for example, how the ESP8266 stores information in certain flash sectors). It was also a lesson learned on the volatility and fragility of electrical components. While we could not pinpoint exactly what caused our hardware to fail, we suspect it occured when the computer we were working on began to glitch and eventually crashed, also impacting the monitor and mouse, after which our hardware also began to fail.

VII. REFERENCES

1. Bezzam, Eric. “3.1 How It Works.” *DSP Labs*, 2018, lcav.gitbook.io/dsp-labs/alien-voice/effect_description.
2. Biswal, Sanskar. “Programming FLASH ROM in STM32.” Medium, TheTeamMavericks, 25 May 2020, medium.com/theteammavericks/programming-flash-rom-in-stm32-f5b7d6dcba4f.

VIII. PARTS LIST

The table below lists the parts that were used for this project.

Part	Sources	Vendor Part #	Price
CMA-4544PF-W Microphone	Digi-Key	102-1721-ND	$0.77
LM386 Audio Amplifier	Texas Instruments	4/NOPB	$1.17
AS358P	Spark Fun	945JB2	$0.95
ESP8266 WiFi Module	Spark Fun	WRL-13678	$6.95
Speaker 0.5W 8 ohm	Adafruit	1891	$1.75
Breadboard Power Supply Module	Inland	078584	$3.99

IX. APPENDIX

Microcontroller Code

```
/**
    Main: Contains main function
    @file main.c
    @author Josh Brake
    @version 1.0 10/7/2020
*/

#include "STM32F401RE.h"
#include "main.h"
#include <string.h>
#include <math.h>
#include "UARTRingBuffer.h"

int count = 0; // count up to 248,000 for full FLASH memory (sector 1-7)
size_t NUM_SAMPLES = 130000; // sector 6-7
int recording = 0;
int play_index = 0;
uint16_t *FLASH_SECTOR_6_ADDRESS = (uint16_t *) 0x08040000;
int TYPE = NORMAL;

// TIMER
const uint16_t arr_for_sampling = 2100;

void initADCTIM(void) {
    //enable all registers
    TIM2->EGR |= 1;
    //set PWM frequency to 40kHz
    TIM2->ARR = arr_for_sampling;
    // Configure interrupt enable on update event
    TIM2->DIER |= 0b1; // TIM_DIER_UIE
    NVIC_EnableIRQ(TIM2_IRQn);
    //enable counter
    TIM2->CR1 |= 1;
}

void initPlayTIM(uint16_t arr) {
```

```
    TIM4->EGR |= 1;
    TIM4->DIER |= 0b1; // TIM_DIER_UIE
    TIM4->ARR = arr;
    NVIC_EnableIRQ(TIM4_IRQn);
    //enable counter
    TIM4->CR1 |= 1;
}

/** Initialize the ESP and print out IP address to terminal
 */
void initESP8266(USART_TypeDef * ESP_USART, USART_TypeDef * TERM_USART){
    uint8_t volatile str[BUFFER_SIZE] = "";

    // Disable echo
    sendString(ESP_USART, "ATE1\r\n");
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    readString(ESP_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    sendString(TERM_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);

    // Set CWMODE to AP + station mode
    sendString(ESP_USART, "AT+CWMODE=3\r\n");
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    readString(ESP_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    sendString(TERM_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);

    // Enable multiple connections
    sendString(ESP_USART, "AT+CIPMUX=1\r\n");
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    readString(ESP_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    sendString(TERM_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);

    // Create TCP server on port 80
    sendString(ESP_USART, "AT+CIPSERVER=1,80\r\n");
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
```

```
    readString(ESP_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    sendString(TERM_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);

    // Connect to WiFi network
    uint8_t connect_cmd[128] = "";
    sprintf(connect_cmd,"AT+CWJAP=\"%s\",\"%s\"\r\n", SSID, PASSWORD);

    sendString(ESP_USART, connect_cmd);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    readString(ESP_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    sendString(TERM_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);

    // Wait for connection
    delay_millis(DELAY_TIM, 30000);

    // Print out status
    sendString(ESP_USART, "AT+CIFSR\r\n");
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    readString(ESP_USART, str);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    sendString(TERM_USART, str);

}

/** Send command to ESP and echo to the terminal.
    @param C-string (i.e., pointer to start of a null-terminated array
        of characters.
*/
void serveWebpage(uint8_t str []) {
    USART_TypeDef * ESP_USART = id2Port(ESP_USART_ID);
    USART_TypeDef * TERM_USART = id2Port(TERM_USART_ID);
    uint8_t cmd_response[BUFFER_SIZE] = "";

    uint32_t str_length = strlen(str)+2;
```

```
    memset(cmd_response, 0, BUFFER_SIZE);
    // Send to terminal what we're sending
    sendString(TERM_USART, "Serving: ");
    sendString(TERM_USART, str);
    sendString(TERM_USART, "\r\n");

    // Send HTML
    memset(cmd_response, 0, BUFFER_SIZE);
    uint8_t cmd[BUFFER_SIZE] = "";
    sprintf(cmd, "AT+CIPSEND=0,%d\r\n",str_length);
    sendString(ESP_USART, cmd);
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    readString(ESP_USART, cmd_response);
    sendString(TERM_USART, cmd_response);

    memset(cmd_response, 0, BUFFER_SIZE);
    sendString(ESP_USART, str);
    sendString(ESP_USART, "\r\n");
    delay_millis(DELAY_TIM, CMD_DELAY_MS);
    readString(ESP_USART, cmd_response);
    sendString(TERM_USART, cmd_response);

}

void play(int type, int arr) {
    TYPE = type;
    initPlayTIM(arr);
}

void TIM4_IRQHandler(void) {
    // Clear update interrupt flag
    TIM4->SR &= ~(0b1);
    if (play_index == NUM_SAMPLES) play_index = 0; //TIM4->CR1 &= ~(0b1);
// RAM
    uint16_t note = *(FLASH_SECTOR_6_ADDRESS + play_index);
//VOLTAGE_ARRAY[play_index]; // RAM
    if (TYPE == ALIEN) {
        note = (uint16_t) ((double) (note * (((double)
sin(2*3.14*(900/40000)*play_index)+1))));
    }
```

```
    spiSendReceive12(note);
    ++play_index;
}
/** Map Button IRQ handler to our custom ISR
 * Button turns on the TIM2 for interrupts to sample at 10kHz
 * If recording, it turns off TIM2, DMA, and ADC to stop recording
 */

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);
        if (recording) {
            TIM2->CR1 &= ~(0b1); // disable timer
            ADC1->CR2.ADON = 0; // turn off ADC
            digitalWrite(GPIOA, LED_PIN, GPIO_LOW);
            recording = 0;
        }
        else {
            initFLASH(); // RAM
            count = 0;
            initADCTIM();
            digitalWrite(GPIOA, LED_PIN, GPIO_HIGH);
            recording = 1;
        }
    }
}

/** Map TIM2 IRQ handler to our custom ISR
 * TIM2 handler turns on the ADC to start conversions
 * if the buffer is full, turn off recording (timer, ADC, DMA)
 */

void TIM2_IRQHandler(void) {
    // Clear update interrupt flag
    TIM2->SR &= ~(0b1);
    if (count < NUM_SAMPLES) {//VOLTAGE_ARRAY_SIZE) { // RAM
        configureADC();
    }
```

```
    else {
        TIM2->CR1 &= ~(0b1); // disable timer
        ADC1->CR2.ADON = 0;  // turn off ADC
        digitalWrite(GPIOA, LED_PIN, GPIO_LOW);
        recording = 0;
    }
}

/** Map ADC IRQ handler to our custom ISR
 * ADC handler turns on the DMA when the the ADC indicates that it's
finished converting
 */
void ADC_IRQHandler(void){
    ADC1->SR &= ~(0b10);// clear interrupt
    // VOLTAGE_ARRAY[count] = ADC1->DR.DR; // RAM
    *(FLASH_SECTOR_6_ADDRESS + count) = (uint16_t) ADC1->DR.DR;
    while(FLASH->SR.BSY == 1);
    ++count;
}

/** Map USART1 IRQ handler to our custom ISR
 */
void USART1_IRQHandler(){
    USART_TypeDef * ESP_USART = id2Port(ESP_USART_ID);
    usart_ISR(ESP_USART);
}

int main(void) {
    // Configure flash latency and set clock to run at 84 MHz
    configureFlash();
    configureClock();

    // Enable GPIOA clock
    RCC->AHB1ENR.GPIOAEN = 1;
    RCC->AHB1ENR.GPIOCEN = 1;

    RCC->APB2ENR.ADC1EN = 1;
    RCC->APB2ENR.SYSCFGEN = 1;
    RCC->APB2ENR.SPI1EN = 1;
    // Initialize timers
```

```
    RCC->APB1ENR.TIM2EN = 1;
    RCC->APB1ENR.TIM3EN = 1;
    RCC->APB1ENR.TIM4EN = 1;
    RCC->APB1ENR.TIM5EN = 1;
    initDelayTIM(DELAY_TIM);
    initDelayTIM(TIM3);

    // Set up LED pin as output
    pinMode(GPIOA, LED_PIN, GPIO_OUTPUT);
    pinMode(GPIOC, BUTTON_PIN, GPIO_INPUT); // button press
    pinMode(GPIOA, GPIO_PA0, GPIO_ANALOG); // ADC pin

    // Enable interrupts globally
    __enable_irq();
    // button interrupt
    *SYSCFG_EXTICR4 |= 0b00100000; // Set EXTICR4 for PC13 to 2
    // Configure interrupt for falling edge of GPIO PC13
    // 1. Configure mask bit
    // 2. Disable rising edge trigger
    // 3. Enable falling edge trigger
    // 4. Turn on EXTI interrupt in NVIC_ISER1
    EXTI->IMR |= 1 << 13; // PC13 is EXTI13
    EXTI->RTSR &= ~(1 << 13); // PC13 is EXTI13
    EXTI->FTSR |= 1 << 13; // PC13 is EXTI13
    __NVIC_EnableIRQ(EXTI15_10_IRQn); // enable button interrupt
    __NVIC_EnableIRQ(ADC_IRQn); // enable ADC interrupt

    // Configure ESP and Terminal UARTs
    USART_TypeDef * ESP_USART = initUSART(ESP_USART_ID, 115200);
    USART_TypeDef * TERM_USART = initUSART(TERM_USART_ID, 115200);

    // Configure interrupt for USART1
    *NVIC_ISER1 |= (1 << 5);
    ESP_USART->CR1.RXNEIE = 1;

    // Initialize ring buffer
    init_ring_buffer();
    flush_buffer();

    // Initialize ESP
```

```
    delay_millis(DELAY_TIM, 1000);
    initESP8266(ESP_USART, TERM_USART);
    delay_millis(DELAY_TIM, 500);

    // Set up temporary buffers for requests
    uint8_t volatile http_request[BUFFER_SIZE] = "";
    uint8_t volatile temp_str[BUFFER_SIZE] = "";

    while(1) {
        // Clear temp_str buffer
        memset(http_request, 0, BUFFER_SIZE);
        volatile uint32_t http_req_len = 0;

        // Loop through and read any data available in the buffer
        if(is_data_available()) {
            do{
                memset(temp_str, 0, BUFFER_SIZE);
                readString(ESP_USART, temp_str); // Read in available
bytes
                strcat(http_request, temp_str); // Append to current
http_request string
                http_req_len = strlen(http_request); // Store length of
request
                delay_millis(DELAY_TIM, 20); // Delay
            } while(is_data_available()); // Check for end of transaction

            // Echo received string to the terminal
            sendString(TERM_USART, http_request);

            // Search to see if there was a GET request
            volatile uint8_t get_request = look_for_substring("GET",
http_request);

            // If a GET request, process the request
            if(get_request == 1){
                // Look for "REQ" in http_request
                volatile uint8_t button_req = look_for_substring("REQ",
http_request);
                volatile uint8_t favicon_req =
look_for_substring("favicon", http_request);
```

```
                if(!favicon_req){

                    /* Look for request data and process it
                        If REQ=ON, then turn LED on.
                        If REQ=OFF, then turn LED off.
                        If we don't recognize the REQ, then send message
to terminal and don't do anything.
                    */

                    if(button_req == 1){
                        volatile uint8_t button_req_type;
                        if(look_for_substring("=PLAY", http_request))
button_req_type = REQ_PLAY;
                        else if(look_for_substring("=SLOW", http_request))
button_req_type = REQ_SLOW;
                        else if(look_for_substring("=FAST", http_request))
button_req_type = REQ_FAST;
                        else if(look_for_substring("=ALIEN",
http_request)) button_req_type = REQ_ALIEN;

                        switch(button_req_type){
                            case REQ_PLAY:
                                play(NORMAL, normal);
                                sendString(TERM_USART, "Playing back
regular voice.\n");
                                break;
                            case REQ_SLOW:
                                play(NORMAL, slow);
                                sendString(TERM_USART, "Playing voice
slowed down.\n");
                                break;
                            case REQ_FAST:
                                play(NORMAL, fast);
                                sendString(TERM_USART, "Playing voice sped
up.\n");
                                break;
                            case REQ_ALIEN:
                                play(ALIEN, normal);
```

```
                                sendString(TERM_USART, "Playing alien
voice.\n");
                                break;
                            case REQ_UNKNOWN:
                                sendString(TERM_USART, "Unknown
request.\n");
                        }
                    }

                    // Serve the individual HTML commands for the webpage
                    serveWebpage("<!DOCTYPE html>");
                    serveWebpage("<meta name=\"viewport\"
content=\"width=device-width, initial-scale=1.0\">");
                    serveWebpage("<title>Alien Voice Filter</title>");
                    serveWebpage("<h3 style=\"color:Green;\">~~ALiEn vOicE
FiLteR~~</h3>");
                    serveWebpage("<p
style=\"color:RebeccaPurple;\">Welcome to the spooky alien voice
filter!</p>");
                    serveWebpage("<p style=\"color:Chartreuse;\">Press the
blue button on the microcontroller to start recording your voice. Press
the button again to stop. The recording will automatically stop after a
set amount of time.</p>");
                    serveWebpage("<p
style=\"color:MediumSlateBlue;\">After you record, press one of the
buttons below to play your filtered voice back.</p>");
                    serveWebpage("<form action=\"REQ=PLAY\"
style=\"background-color:SpringGreen;\"><input type=\"submit\" value =
\"Playback\"></form>");
                    serveWebpage("<form action=\"REQ=SLOW\"
style=\"background-color:Indigo;\"><input type=\"submit\" value = \"Slow
Down\"></form>");
                    serveWebpage("<form action=\"REQ=FAST\"
style=\"background-color:LimeGreen;\"><input type=\"submit\" value =
\"Speed Up\"></form>");
                    serveWebpage("<form action=\"REQ=ALIEN\"
style=\"background-color:DarkViolet;\"><input type=\"submit\" value =
\"Alien Voice\"></form>");
                }
```

```
                // Close connection
                memset(temp_str, 0, BUFFER_SIZE);
                sendString(ESP_USART, "AT+CIPCLOSE=0\r\n");
                readString(ESP_USART, temp_str);
                sendString(TERM_USART,temp_str);
            }
        }
    }

}
```

```
/**
    Main Header: Contains general defines and selected portions of CMSIS
files
    @file main.h
    @author Josh Brake
    @version 1.0 10/7/2020
*/

#ifndef MAIN_H
#define MAIN_H

#include "STM32F401RE.h"

////////////////////////////////////////////////////////////////////////////
/////
// Custom defines
////////////////////////////////////////////////////////////////////////////
/////
// for playing
// type
#define NORMAL 0
#define ALIEN 1
// speed
#define fast 0
#define slow 1
#define normal 2
// Wifi Network Info (make sure to keep surrounding double quotes)
```

```
#define SSID "Fios-QS9zf"
#define PASSWORD "yes753code39jab"

#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)

#define NSS_PIN GPIO_PA1
#define SPI1CLK_PIN GPIO_PA5
#define MISO_PIN GPIO_PA6
#define MOSI_PIN GPIO_PA7
#define BUTTON_PIN 13 // PC13
#define LED_PIN GPIO_PA8

// Request defines
#define REQ_UNKNOWN 0
#define REQ_PLAY 1
#define REQ_SLOW 2
#define REQ_FAST 3
#define REQ_ALIEN 4

#define ESP_USART_ID USART1_ID
#define TERM_USART_ID USART2_ID
#define ADC_TIM TIM2
#define PLAY_TIM TIM4
#define DELAY_TIM TIM5
#define CMD_DELAY_MS 100
#define BUFFER_SIZE 2048
```

```
/////////////////////////////////////////////////////////////////////////////
/////
// 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"


#endif // MAIN_H

// STM32F401RE_FLASH.c
// Source code for FLASH functions
```

```c
#include "STM32F401RE_FLASH.h"

void configureFlash() {
    FLASH->ACR.LATENCY = 2; // Set to 0 waitstates
    FLASH->ACR.PRFTEN = 1; // Turn on the ART
}

void clearFlash(){
    // Clear flash memory
    while(FLASH->SR.BSY== 1);
    FLASH->CR.SNB = 0b0110;
    FLASH->CR.SER = 1;
    FLASH->CR.STRT = 1;
    while(FLASH->SR.BSY== 1);
    FLASH->CR.SNB = 0b0111;
    FLASH->CR.SER = 1;
    FLASH->CR.STRT = 1;
    while(FLASH->SR.BSY== 1);
}

void initFLASH() {
    FLASH->KEYR = 0x45670123;
    FLASH->KEYR = 0xCDEF89AB; // enable write to CR
    FLASH->OPTKEYR = 0x08192A3B;
    FLASH->OPTKEYR = 0x4C5D6E7F; // enable write to OPTCR
    FLASH->CR.PSIZE = 0b01; // half-word (16-bits)
    FLASH->OPTCR.RDP = 0xaa;
    FLASH->OPTCR.nWRP = 0b11111110;
    FLASH->OPTCR.SPRMOD = 0;
    clearFlash();
    FLASH->CR.PG = 1;
}




// STM32F401RE_FLASH.h
// Header for FLASH functions

#ifndef STM32F4_FLASH_H
```

```
#define STM32F4_FLASH_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////
/////
// Definitions
////////////////////////////////////////////////////////////////////////////
/////

#define __IO volatile

// Base addresses for GPIO ports
#define FLASH_BASE (0x40023C00UL) // base address of RCC

////////////////////////////////////////////////////////////////////////////
/////
// Bitfield structs
////////////////////////////////////////////////////////////////////////////
/////

typedef struct {
  __IO uint32_t LATENCY :4;
  __IO uint32_t         :4;
  __IO uint32_t PRFTEN  :1;
  __IO uint32_t ICEN    :1;
  __IO uint32_t DCEN    :1;
  __IO uint32_t ICRST   :1;
  __IO uint32_t DCRST   :1;
  __IO uint32_t         :19;
} ACR_bits;

typedef struct {
  __IO uint32_t EOP     :1;
  __IO uint32_t OPERR   :1;
  __IO uint32_t         :2;
  __IO uint32_t WRPERR  :1;
  __IO uint32_t PGAERR  :1;
  __IO uint32_t PGPERR  :1;
  __IO uint32_t PGSERR  :1;
```

```
  __IO uint32_t RDERR   :1;
  __IO uint32_t         :7;
  __IO uint32_t BSY     :1;
  __IO uint32_t         :15;
} FLASH_SR_bits;

typedef struct {
  __IO uint32_t PG      :1;
  __IO uint32_t SER     :1;
  __IO uint32_t MER     :1;
  __IO uint32_t SNB     :4;
  __IO uint32_t         :1;
  __IO uint32_t PSIZE   :2;
  __IO uint32_t         :6;
  __IO uint32_t STRT    :1;
  __IO uint32_t         :7;
  __IO uint32_t EOPIE   :1;
  __IO uint32_t ERRIE   :1;
  __IO uint32_t         :5;
  __IO uint32_t LOCK    :1;
} FLASH_CR_bits;

typedef struct {
  __IO uint32_t OPTLOCK :1;
  __IO uint32_t OPTSTRT :1;
  __IO uint32_t BOR_LEV :2;
  __IO uint32_t         :1;
  __IO uint32_t WDG_SW  :1;
  __IO uint32_t nRSTSTOP  :1;
  __IO uint32_t nRSTSTDBY :1;
  __IO uint32_t RDP     :8;
  __IO uint32_t nWRP    :8;
  __IO uint32_t         :7;
  __IO uint32_t SPRMOD  :1;
} FLASH_OPTCR_bits;

typedef struct {
  __IO ACR_bits ACR;      /*!< FLASH access control register,   Address
offset: 0x00 */
```

```
  __IO uint32_t KEYR;     /*!< FLASH key register,              Address
offset: 0x04 */
  __IO uint32_t OPTKEYR;  /*!< FLASH option key register,       Address
offset: 0x08 */
  __IO FLASH_SR_bits SR;       /*!< FLASH status register,
Address offset: 0x0C */
  __IO FLASH_CR_bits CR;       /*!< FLASH control register,
Address offset: 0x10 */
  __IO FLASH_OPTCR_bits OPTCR;    /*!< FLASH option control register ,
Address offset: 0x14 */
  __IO uint32_t OPTCR1;   /*!< FLASH option control register 1, Address
offset: 0x18 */
} FLASH_TypeDef;

#define FLASH ((FLASH_TypeDef *) FLASH_BASE)

////////////////////////////////////////////////////////////////////////////
/////
// Function prototypes
////////////////////////////////////////////////////////////////////////////
/////

void configureFlash();
void clearFlash();
void initFLASH();
#endif



// STM32F401RE_TIM.c
// TIM functions

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


void initDelayTIM(TIM_TypeDef * TIMx){
  // Set prescaler to give 1 µs time base
  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
}
```

```
// STM32F401RE_TIM.h
// Header for TIM functions

#ifndef STM32F4_TIM_H
#define STM32F4_TIM_H

#include <stdint.h> // Include stdint header
#include "STM32F401RE_GPIO.h"

//////////////////////////////////////////////////////////////////////////////
/////
// Definitions
```

```
//////////////////////////////////////////////////////////////////////////////
/////

#define __IO volatile

#define PERIPH_BASE           0x40000000UL /*!< Peripheral base address in
the alias region                           */
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x00010000UL)

#define TIM1_BASE             (APB2PERIPH_BASE + 0x0000UL)
#define TIM2_BASE             (APB1PERIPH_BASE + 0x0000UL)
#define TIM3_BASE             (APB1PERIPH_BASE + 0x0400UL)
#define TIM4_BASE             (APB1PERIPH_BASE + 0x0800UL)
#define TIM5_BASE             (APB1PERIPH_BASE + 0x0C00UL)
#define TIM9_BASE             (APB2PERIPH_BASE + 0x4000UL)
#define TIM10_BASE            (APB2PERIPH_BASE + 0x4400UL)
#define TIM11_BASE            (APB2PERIPH_BASE + 0x4800UL)

//////////////////////////////////////////////////////////////////////////////
/////
// Bitfield structs
//////////////////////////////////////////////////////////////////////////////
/////

typedef struct
{
  __IO uint32_t CR1;         /*!< TIM control register 1,
Address offset: 0x00 */
  __IO uint32_t CR2;         /*!< TIM control register 2,
Address offset: 0x04 */
  __IO uint32_t SMCR;        /*!< TIM slave mode control register,
Address offset: 0x08 */
  __IO uint32_t DIER;        /*!< TIM DMA/interrupt enable register,
Address offset: 0x0C */
  __IO uint32_t SR;          /*!< TIM status register,
Address offset: 0x10 */
  __IO uint32_t EGR;         /*!< TIM event generation register,
Address offset: 0x14 */
```

```
  __IO uint32_t CCMR1;        /*!< TIM capture/compare mode register 1,
Address offset: 0x18 */
  __IO uint32_t CCMR2;        /*!< TIM capture/compare mode register 2,
Address offset: 0x1C */
  __IO uint32_t CCER;         /*!< TIM capture/compare enable register,
Address offset: 0x20 */
  __IO uint32_t CNT;          /*!< TIM counter register,
Address offset: 0x24 */
  __IO uint32_t PSC;          /*!< TIM prescaler,
Address offset: 0x28 */
  __IO uint32_t ARR;          /*!< TIM auto-reload register,
Address offset: 0x2C */
  __IO uint32_t RCR;          /*!< TIM repetition counter register,
Address offset: 0x30 */
  __IO uint32_t CCR1;         /*!< TIM capture/compare register 1,
Address offset: 0x34 */
  __IO uint32_t CCR2;         /*!< TIM capture/compare register 2,
Address offset: 0x38 */
  __IO uint32_t CCR3;         /*!< TIM capture/compare register 3,
Address offset: 0x3C */
  __IO uint32_t CCR4;         /*!< TIM capture/compare register 4,
Address offset: 0x40 */
  __IO uint32_t BDTR;         /*!< TIM break and dead-time register,
Address offset: 0x44 */
  __IO uint32_t DCR;          /*!< TIM DMA control register,
Address offset: 0x48 */
  __IO uint32_t DMAR;         /*!< TIM DMA address for full transfer,
Address offset: 0x4C */
  __IO uint32_t OR;           /*!< TIM option register,
Address offset: 0x50 */
} TIM_TypeDef;

// Pointers to appropriately-sized chunks of memory for each peripheral
#define TIM1 ((TIM_TypeDef *) TIM1_BASE)
#define TIM2 ((TIM_TypeDef *) TIM2_BASE)
#define TIM3 ((TIM_TypeDef *) TIM3_BASE)
#define TIM4 ((TIM_TypeDef *) TIM4_BASE)
#define TIM5 ((TIM_TypeDef *) TIM5_BASE)
#define TIM9 ((TIM_TypeDef *) TIM9_BASE)
#define TIM10 ((TIM_TypeDef *) TIM10_BASE)
```

```
#define TIM11 ((TIM_TypeDef *) TIM11_BASE)

////////////////////////////////////////////////////////////////////////////////
/////
// Function prototypes
////////////////////////////////////////////////////////////////////////////////
/////

void initDelayTIM(TIM_TypeDef * TIMx);
void delay_millis(TIM_TypeDef * TIMx, uint32_t ms);
void delay_micros(TIM_TypeDef * TIMx, uint32_t us);

#endif

// STM32F401RE_RCC.c
// Source code for RCC functions

#include "STM32F401RE_RCC.h"

void configurePLL() {
    // Set clock to 84 MHz
    // Output freq = (src_clk) * (N/M) / P
    // (8 MHz) * (336/8) / 4 = 84 MHz
    // M:8, N:336, P:4, Q:7
    // Use HSE as PLLSRC

    RCC->CR.PLLON = 0; // Turn off PLL
    while (RCC->CR.PLLRDY != 0); // Wait till PLL is unlocked (e.g., off)

    // Load configuration
    RCC->PLLCFGR.PLLSRC = PLLSRC_HSE;
    RCC->PLLCFGR.PLLM = 8;
    RCC->PLLCFGR.PLLN = 336;
    RCC->PLLCFGR.PLLP = 0b01; // divide by 4
    RCC->PLLCFGR.PLLQ = 4;

    // Enable PLL and wait until it's locked
    RCC->CR.PLLON = 1;
    while(RCC->CR.PLLRDY == 0);
}
```

```
void configureClock(){
    /* Configure APB prescalers
    1. Set APB2 (high-speed bus) prescaler to no division
    2. Set APB1 (low-speed bus) to divide by 2.
    */

    RCC->CFGR.PPRE2 = 0b000;
    RCC->CFGR.PPRE1 = 0b100;

    // Turn on and bypass for HSE from ST-LINK
    RCC->CR.HSEBYP = 1;
    RCC->CR.HSEON = 1;
    while(!RCC->CR.HSERDY);

    // Configure and turn on PLL for 84 MHz
    configurePLL();

    // Select PLL as clock source
    RCC->CFGR.SW = SW_PLL;
    while(RCC->CFGR.SWS != 0b10);

    SystemCoreClock = 84000000;
}

// STM32F401RE_RCC.h
// Header for RCC functions

#ifndef STM32F4_RCC_H
#define STM32F4_RCC_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////////
/////
// Definitions
////////////////////////////////////////////////////////////////////////////////
/////

// Global defines related to clock
```

```c
uint32_t SystemCoreClock;    // Updated by configureClock()
#define HSE_VALUE 8000000    // Value of external input to OSC from
ST-LINK

#define __IO volatile

// Base addresses
#define RCC_BASE (0x40023800UL) // base address of RCC

// PLL
#define PLLSRC_HSI 0
#define PLLSRC_HSE 1

// Clock configuration
#define SW_HSI  0
#define SW_HSE  1
#define SW_PLL  2

////////////////////////////////////////////////////////////////////////////
/////
// Bitfield structs
////////////////////////////////////////////////////////////////////////////
/////
typedef struct {
    volatile uint32_t HSION     : 1;
    volatile uint32_t HSIRDY    : 1;
    volatile uint32_t           : 1;
    volatile uint32_t HSITRIM   : 5;
    volatile uint32_t HSICAL    : 8;
    volatile uint32_t HSEON     : 1;
    volatile uint32_t HSERDY    : 1;
    volatile uint32_t HSEBYP    : 1;
    volatile uint32_t CSSON     : 1;
    volatile uint32_t           : 4;
    volatile uint32_t PLLON     : 1;
    volatile uint32_t PLLRDY    : 1;
    volatile uint32_t PLLI2SON  : 1;
    volatile uint32_t PLLI2SRDY : 1;
    volatile uint32_t           : 4;
} CR_bits;
```

```
typedef struct {
    volatile uint32_t PLLM      : 6;
    volatile uint32_t PLLN      : 9;
    volatile uint32_t           : 1;
    volatile uint32_t PLLP      : 2;
    volatile uint32_t           : 4;
    volatile uint32_t PLLSRC    : 1;
    volatile uint32_t           : 1;
    volatile uint32_t PLLQ      : 4;
    volatile uint32_t           : 4;
} PLLCFGR_bits;

typedef struct {
    volatile uint32_t SW        : 2;
    volatile uint32_t SWS       : 2;
    volatile uint32_t HPRE      : 4;
    volatile uint32_t           : 2;
    volatile uint32_t PPRE1     : 3;
    volatile uint32_t PPRE2     : 3;
    volatile uint32_t RTCPRE    : 5;
    volatile uint32_t MCO1      : 2;
    volatile uint32_t I2SSCR    : 1;
    volatile uint32_t MCO1PRE   : 3;
    volatile uint32_t MCO2PRE   : 3;
    volatile uint32_t MCO2      : 2;
} CFGR_bits;

typedef struct {
    volatile uint32_t GPIOAEN   : 1;
    volatile uint32_t GPIOBEN   : 1;
    volatile uint32_t GPIOCEN   : 1;
    volatile uint32_t GPIODEN   : 1;
    volatile uint32_t GPIOEEN   : 1;
    volatile uint32_t           : 2;
    volatile uint32_t GPIOHEN   : 1;
    volatile uint32_t           : 4;
    volatile uint32_t CRCEN     : 1;
    volatile uint32_t           : 3;
    volatile uint32_t           : 5;
```

```
    volatile uint32_t DMA1EN    : 1;
    volatile uint32_t DMA2EN    : 1;
    volatile uint32_t           : 9;
} AHB1ENR_bits;

typedef struct {
    volatile uint32_t TIM2EN    : 1;
    volatile uint32_t TIM3EN    : 1;
    volatile uint32_t TIM4EN    : 1;
    volatile uint32_t TIM5EN    : 1;
    volatile uint32_t           : 7;
    volatile uint32_t WWDGEN    : 1;
    volatile uint32_t           : 2;
    volatile uint32_t SPI2EN    : 1;
    volatile uint32_t SPI3EN    : 1;
    volatile uint32_t           : 1;
    volatile uint32_t USART2EN  : 1;
    volatile uint32_t           : 3;
    volatile uint32_t I2C1EN    : 1;
    volatile uint32_t I2C2EN    : 1;
    volatile uint32_t I2C3EN    : 1;
    volatile uint32_t           : 4;
    volatile uint32_t PWREN     : 1;
    volatile uint32_t           : 3;
} APB1ENR_bits;

typedef struct {
    volatile uint32_t TIM1EN    : 1;
    volatile uint32_t           : 3;
    volatile uint32_t USART1EN  : 1;
    volatile uint32_t USART6EN  : 1;
    volatile uint32_t           : 2;
    volatile uint32_t ADC1EN    : 1;
    volatile uint32_t           : 2;
    volatile uint32_t SDIOEN    : 1;
    volatile uint32_t SPI1EN    : 1;
    volatile uint32_t SPI4EN    : 1;
    volatile uint32_t SYSCFGEN  : 1;
    volatile uint32_t           : 1;
    volatile uint32_t TIM9EN    : 1;
```

```
    volatile uint32_t TIM10EN   : 1;
    volatile uint32_t TIM11EN   : 1;
    volatile uint32_t           : 13;
} APB2ENR_bits;

typedef struct {
  __IO CR_bits      CR;            /*!< RCC clock control register,
Address offset: 0x00 */
  __IO PLLCFGR_bits PLLCFGR;       /*!< RCC PLL configuration register,
Address offset: 0x04 */
  __IO CFGR_bits    CFGR;          /*!< RCC clock configuration register,
Address offset: 0x08 */
  __IO uint32_t     CIR;           /*!< RCC clock interrupt register,
Address offset: 0x0C */
  __IO uint32_t     AHB1RSTR;      /*!< RCC AHB1 peripheral reset
register,                          Address offset: 0x10 */
  __IO uint32_t     AHB2RSTR;      /*!< RCC AHB2 peripheral reset
register,                          Address offset: 0x14 */
  __IO uint32_t     AHB3RSTR;      /*!< RCC AHB3 peripheral reset
register,                          Address offset: 0x18 */
  uint32_t          RESERVED0;     /*!< Reserved, 0x1C
*/
  __IO uint32_t     APB1RSTR;      /*!< RCC APB1 peripheral reset
register,                          Address offset: 0x20 */
  __IO uint32_t     APB2RSTR;      /*!< RCC APB2 peripheral reset
register,                          Address offset: 0x24 */
  uint32_t          RESERVED1[2];  /*!< Reserved, 0x28-0x2C
*/
  __IO AHB1ENR_bits AHB1ENR;       /*!< RCC AHB1 peripheral clock
register,                          Address offset: 0x30 */
  __IO uint32_t     AHB2ENR;       /*!< RCC AHB2 peripheral clock
register,                          Address offset: 0x34 */
  __IO uint32_t     AHB3ENR;       /*!< RCC AHB3 peripheral clock
register,                          Address offset: 0x38 */
  uint32_t          RESERVED2;     /*!< Reserved, 0x3C
*/
  __IO APB1ENR_bits APB1ENR;       /*!< RCC APB1 peripheral clock enable
register,                  Address offset: 0x40 */
  __IO APB2ENR_bits APB2ENR;       /*!< RCC APB2 peripheral clock enable
register,                  Address offset: 0x44 */
```

```
  uint32_t          RESERVED3[2];  /*!< Reserved, 0x48-0x4C
*/
  __IO uint32_t     AHB1LPENR;     /*!< RCC AHB1 peripheral clock enable
in low power mode register, Address offset: 0x50 */
  __IO uint32_t     AHB2LPENR;     /*!< RCC AHB2 peripheral clock enable
in low power mode register, Address offset: 0x54 */
  __IO uint32_t     AHB3LPENR;     /*!< RCC AHB3 peripheral clock enable
in low power mode register, Address offset: 0x58 */
  uint32_t          RESERVED4;     /*!< Reserved, 0x5C
*/
  __IO uint32_t     APB1LPENR;     /*!< RCC APB1 peripheral clock enable
in low power mode register, Address offset: 0x60 */
  __IO uint32_t     APB2LPENR;     /*!< RCC APB2 peripheral clock enable
in low power mode register, Address offset: 0x64 */
  uint32_t          RESERVED5[2];  /*!< Reserved, 0x68-0x6C
*/
  __IO uint32_t     BDCR;          /*!< RCC Backup domain control
register,                          Address offset: 0x70 */
  __IO uint32_t     CSR;           /*!< RCC clock control & status
register,                         Address offset: 0x74 */
  uint32_t          RESERVED6[2];  /*!< Reserved, 0x78-0x7C
*/
  __IO uint32_t     SSCGR;         /*!< RCC spread spectrum clock
generation register,               Address offset: 0x80 */
  __IO uint32_t     PLLI2SCFGR;    /*!< RCC PLLI2S configuration register,
Address offset: 0x84 */
  uint32_t          RESERVED7[1];  /*!< Reserved, 0x88
*/
  __IO uint32_t     DCKCFGR;       /*!< RCC Dedicated Clocks configuration
register,                 Address offset: 0x8C */
} RCC_TypeDef;

#define RCC ((RCC_TypeDef *) RCC_BASE)

////////////////////////////////////////////////////////////////////////////
/////
// Function prototypes
////////////////////////////////////////////////////////////////////////////
/////
```

```
void configurePLL();
void configureClock();

#endif
#include "STM32F401RE_ADC.h"

void configureADC(){
    // turn on ADC
    ADC1->CR2.ADON = 1;
    // set the number of conversions in the sequence to 1
    ADC1->SQR1 |= (0b0000 << 20);
    // set the channel of the ADC selected as the first conversion
    // TODO: DMA2 channel 0 stream 0
    ADC1->SQR3 |= 0b00000;
    // set single conversion mode
    ADC1->CR2.CONT = 0;
    //enable interupts after each ADC conversion to allow transfer from DR
to DMA
    ADC1->CR1.EOCIE = 1;
    // turn on end of conversion selection
    ADC1->CR2.EOCS = 1;
    // turn on DMA mode
    // ADC1->CR2 |= ADC_CR2_DMA;
    //sampling time for ADC TRY THIS LATER
    // ADC->ADCSMPR2 |= 0b111;
    // enable SWSTART to start
    ADC1->CR2.SWSTART = 1;
}
// STM32F401RE_ADC.h
// Header for ADC functions

#ifndef STM32F4_ADC_H
#define STM32F4_ADC_H

#include <stdint.h> // Include stdint header

//////////////////////////////////////////////////////////////////////////////
/////
// Definitions
```

```
/////////////////////////////////////////////////////////////////////////////
/////

#define ADC1_BASE (0x40012000UL)
#define __IO volatile

/////////////////////////////////////////////////////////////////////////////
/////
// Bitfield structs
/////////////////////////////////////////////////////////////////////////////
/////

typedef struct {
  __IO uint32_t AWDCH       : 5;
  __IO uint32_t EOCIE       : 1;
  __IO uint32_t AWDIE       : 1;
  __IO uint32_t JEOCIE      : 1;
  __IO uint32_t SCAN        : 1;
  __IO uint32_t AWDSGL      : 1;
  __IO uint32_t JAUTO       : 1;
  __IO uint32_t DISCEN      : 1;
  __IO uint32_t JDISCEN     : 1;
  __IO uint32_t DISC_NUM    : 3;
  __IO uint32_t             : 6;
  __IO uint32_t JAWDEN      : 1;
  __IO uint32_t AWDEN       : 1;
  __IO uint32_t RES         : 2;
  __IO uint32_t OVRIE       : 1;
  __IO uint32_t             : 5;
} ADC_CR1_bits;

typedef struct {
  __IO uint32_t ADON        : 1;
  __IO uint32_t CONT        : 1;
  __IO uint32_t             : 6;
  __IO uint32_t DMA         : 1;
  __IO uint32_t DDS         : 1;
  __IO uint32_t EOCS        : 1;
  __IO uint32_t ALIGN       : 1;
  __IO uint32_t             : 4;
```

```
  __IO uint32_t JEXTSEL     : 4;
  __IO uint32_t JEXTEN      : 2;
  __IO uint32_t JSWSTART    : 1;
  __IO uint32_t             : 1;
  __IO uint32_t EXTSEL      : 4;
  __IO uint32_t EXTEN       : 2;
  __IO uint32_t SWSTART     : 1;
  __IO uint32_t             : 1;
} ADC_CR2_bits;

typedef struct {
  __IO uint32_t DR  : 16;
  __IO uint32_t     : 16;
} ADC_DR_bits;


typedef struct {
  __IO uint32_t SR;         /*!< ADC status register      Address offset:
0x00 */
  __IO ADC_CR1_bits CR1;    /*!< ADC control register 1   Address offset:
0x04 */
  __IO ADC_CR2_bits CR2;    /*!< ADC control register 2   Address offset:
0x08 */
  __IO uint32_t SMPR1;      /*!< ADC SMPR1                Address offset:
0x0C */
  __IO uint32_t SMPR2;      /*!< ADC SMPR2                Address offset:
0x10 */
  __IO uint32_t JOFR1;      /*!< ADC JOFR1                Address offset:
0x14 */
  __IO uint32_t JOFR2;      /*!< ADC JOFR2                Address offset:
0x18 */
  __IO uint32_t JOFR3;      /*!< ADC JOFR3                Address offset:
0x1C */
  __IO uint32_t JOFR4;      /*!< ADC JOFR4                Address offset:
0x20 */
  __IO uint32_t HTR;        /*!< ADC HTR                  Address offset:
0x24 */
  __IO uint32_t LTR;        /*!< ADC LTR                  Address offset:
0x28 */
```

```
  __IO uint32_t SQR1;       /*!< ADC SQR1                 Address offset:
0x2C */
  __IO uint32_t SQR2;       /*!< ADC SQR2                 Address offset:
0x30 */
  __IO uint32_t SQR3;       /*!< ADC SQR3                 Address offset:
0x34 */
  __IO uint32_t JSQR;       /*!< ADC JSQR                 Address offset:
0x38 */
  __IO uint32_t JDR1;       /*!< ADC JDR1                 Address offset:
0x3C */
  __IO uint32_t JDR2;       /*!< ADC JDR2                 Address offset:
0x40 */
  __IO uint32_t JDR3;       /*!< ADC JDR3                 Address offset:
0x44 */
  __IO uint32_t JDR4;       /*!< ADC JDR4                 Address offset:
0x48 */
  __IO ADC_DR_bits DR;      /*!< ADC data register        Address offset:
0x4C */
} ADC_TypeDef;

// Pointers to ADC-sized chunks of memory for each peripheral
#define ADC1 ((ADC_TypeDef *) ADC1_BASE)

void configureADC();

#endif

// STM32F401RE_SPI.c
// SPI function declarations

#include "STM32F401RE_SPI.h"
#include "STM32F401RE_RCC.h"
#include "STM32F401RE_GPIO.h"

/* Enables the SPI peripheral and intializes its clock speed (baud rate),
polarity, and phase.
 *    -- br: (0b000 - 0b111). The SPI clk will be the master clock /
2^(BR+1).
 *    -- cpol: clock polarity (0: inactive state is logical 0, 1: inactive
state is logical 1).
```

```
 *    -- ncpha: clock phase (0: data changed on leading edge of clk and
captured on next edge,
 *          1: data captured on leading edge of clk and changed on next
edge)
 * Note: the SPI mode register is set with the following unadjustable
settings:
 *    -- Master mode
 *    -- Fixed peripheral select
 *    -- Chip select lines directly connected to peripheral device
 *    -- Mode fault detection enabled
 *    -- WDRBT disabled
 *    -- LLB disabled
 *    -- PCS = 0000 (Peripheral 0 selected), means NPCS[3:0] = 1110
 * Refer to the datasheet for more low-level details. */
void spiInit(uint32_t br, uint32_t cpol, uint32_t cpha) {
    // Turn on GPIOA and GPIOB clock domains (GPIOAEN and GPIOBEN bits in
AHB1ENR)
    RCC->AHB1ENR.GPIOAEN = 1;

    RCC->APB2ENR.SPI1EN = 1; // Turn on SPI1 clock domain (SPI1EN bit in
APB2ENR)

    // Initially assigning SPI pins
    pinMode(GPIOA, 5, GPIO_ALT); // PA5, Arduino D13, SPI1_SCK
    pinMode(GPIOA, 7, GPIO_ALT); // PA7, Arduino D11, SPI1_MOSI
    pinMode(GPIOA, 1, GPIO_OUTPUT); // PB6, Arduino D10, Manual CS

    // Set to AF05 for SPI alternate functions
    GPIOA->AFRL |= (1 << 22) | (1 << 20);
    GPIOA->AFRL |= (1 << 30) | (1 << 28);
    GPIOA->AFRL |= (1 << 18) | (1 << 16);

    SPI1->CR1.BR = br;      // Set the clock divisor
    SPI1->CR1.CPOL = cpol;  // Set the polarity
    SPI1->CR1.CPHA = cpha;  // Set the phase
    SPI1->CR1.LSBFIRST = 0; // Set least significant bit first
    SPI1->CR1.DFF = 1;      // Set data format to 16 bits
    SPI1->CR1.SSM = 0;      // Turn off software slave management
    SPI1->CR2.SSOE = 1;     // Set the NSS pin to output mode
    SPI1->CR1.MSTR = 1;     // Put SPI in master mode
```

```
    SPI1->CR1.SPE = 1;      // Enable SPI
}

// /* Transmits a character (1 byte) over SPI and returns the received
character.
//  *    -- send: the character to send over SPI
//  *    -- return: the character received over SPI */
// uint8_t spiSendReceive(uint8_t send) {
//     SPI1->DR.DR = send; // Transmit the character over SPI
//     while (!(SPI->SPI_SR.RDRF)); // Wait until data has been received
//     return (char) (SPI->SPI_RDR.RD); // Return received character
// }

/* Transmits a short (2 bytes) over SPI and returns the received short.
 *    -- send: the short to send over SPI
 *    -- return: the short received over SPI */
uint16_t spiSendReceive16(uint16_t send) {
    digitalWrite(GPIOB, 6, 0);
    SPI1->CR1.SPE = 1;
    SPI1->DR.DR = send;

    while(!(SPI1->SR.RXNE));
    uint16_t rec = SPI1->DR.DR;

    SPI1->CR1.SPE = 0;
    digitalWrite(GPIOB, 6, 1);

    return rec;
}

/* Transmits a short (2 bytes) over SPI and returns the received short.
 *    -- send: the short to send over SPI
 *    -- return: the short received over SPI */
uint16_t spiSendReceive12(uint16_t send) {
    while(SPI1->SR.TXE != 0b1){} // wait until the transmission buffer is
empty

    digitalWrite(GPIOA, 1, 0);
    SPI1->CR1.SPE = 1;
    // apply a mask so only the first 12
```

```
    uint16_t modified_send = (0b0001 << 12) | (send & 0xFFF);
    SPI1->DR.DR = modified_send;

    while(!(SPI1->SR.RXNE));
    uint16_t rec = SPI1->DR.DR;

    SPI1->CR1.SPE = 0;
    digitalWrite(GPIOA, 1, 1);

    return rec;
}
// STM32F401RE_SPI.h
// Header for SPI functions

#ifndef STM32F4_SPI_H
#define STM32F4_SPI_H

#include <stdint.h> // Include stdint header

////////////////////////////////////////////////////////////////////////////////
/////
// Definitions
////////////////////////////////////////////////////////////////////////////////
/////

#define SPI1_BASE (0x40013000UL)
#define __IO volatile

////////////////////////////////////////////////////////////////////////////////
/////
// Bitfield structs
////////////////////////////////////////////////////////////////////////////////
/////

typedef struct {
  __IO uint32_t CPHA        : 1;
  __IO uint32_t CPOL        : 1;
  __IO uint32_t MSTR        : 1;
  __IO uint32_t BR          : 3;
  __IO uint32_t SPE         : 1;
```

```
  __IO uint32_t LSBFIRST    : 1;
  __IO uint32_t SSI         : 1;
  __IO uint32_t SSM         : 1;
  __IO uint32_t RXONLY      : 1;
  __IO uint32_t DFF         : 1;
  __IO uint32_t CRCNEXT     : 1;
  __IO uint32_t CRCEN       : 1;
  __IO uint32_t BIDIOE      : 1;
  __IO uint32_t BIDIMODE    : 1;
  __IO uint32_t             : 16;
} SPI_CR1_bits;

typedef struct {
  __IO uint32_t RXDMAEN     : 1;
  __IO uint32_t TXDMAEN     : 1;
  __IO uint32_t SSOE        : 1;
  __IO uint32_t             : 1;
  __IO uint32_t FRF         : 1;
  __IO uint32_t ERRIE       : 1;
  __IO uint32_t RXNEIE      : 1;
  __IO uint32_t TXEIE       : 1;
  __IO uint32_t             : 24;
} SPI_CR2_bits;

typedef struct {
  __IO uint32_t RXNE        : 1;
  __IO uint32_t TXE         : 1;
  __IO uint32_t CHSIDE      : 1;
  __IO uint32_t UDR         : 1;
  __IO uint32_t CRCERR      : 1;
  __IO uint32_t MODF        : 1;
  __IO uint32_t OVR         : 1;
  __IO uint32_t BSY         : 1;
  __IO uint32_t FRE         : 1;
  __IO uint32_t DFF         : 1;
  __IO uint32_t CRCNEXT     : 1;
  __IO uint32_t CRCEN       : 1;
  __IO uint32_t BIDIOE      : 1;
  __IO uint32_t BIDIMODE    : 1;
  __IO uint32_t             : 16;
```

```
} SPI_SR_bits;

typedef struct {
  __IO uint32_t DR  : 16;
  __IO uint32_t     : 16;
} SPI_DR_bits;



typedef struct {
  __IO SPI_CR1_bits CR1;        /*!< SPI control register 1 (not used in
I2S mode),      Address offset: 0x00 */
  __IO SPI_CR2_bits CR2;        /*!< SPI control register 2,
Address offset: 0x04 */
  __IO SPI_SR_bits SR;         /*!< SPI status register,
Address offset: 0x08 */
  __IO SPI_DR_bits DR;         /*!< SPI data register,
Address offset: 0x0C */
  __IO uint32_t CRCPR;      /*!< SPI CRC polynomial register (not used in
I2S mode), Address offset: 0x10 */
  __IO uint32_t RXCRCR;     /*!< SPI RX CRC register (not used in I2S
mode),         Address offset: 0x14 */
  __IO uint32_t TXCRCR;     /*!< SPI TX CRC register (not used in I2S
mode),         Address offset: 0x18 */
  __IO uint32_t I2SCFGR;    /*!< SPI_I2S configuration register,
Address offset: 0x1C */
  __IO uint32_t I2SPR;      /*!< SPI_I2S prescaler register,
Address offset: 0x20 */
} SPI_TypeDef;

// Pointers to GPIO-sized chunks of memory for each peripheral
#define SPI1 ((SPI_TypeDef *) SPI1_BASE)

////////////////////////////////////////////////////////////////////////////
/////
// Function prototypes
////////////////////////////////////////////////////////////////////////////
/////

/* Enables the SPI peripheral and intializes its clock speed (baud rate),
polarity, and phase.
```

```
 *    -- clkdivide: (0x01 to 0xFF). The SPI clk will be the master clock /
clkdivide.
 *    -- cpol: clock polarity (0: inactive state is logical 0, 1: inactive
state is logical 1).
 *    -- cpha: clock phase (1: data changed on leading edge of clk and
captured on next edge,
 *          0: data captured on leading edge of clk and changed on next
edge)
 * Note: the SPI mode register is set with the following unadjustable
settings:
 *    -- Master mode
 *    -- Fixed peripheral select
 *    -- Chip select lines directly connected to peripheral device
 *    -- Mode fault detection enabled
 *    -- WDRBT disabled
 *    -- LLB disabled
 *    -- PCS = 0000 (Peripheral 0 selected), means NPCS[3:0] = 1110
 * Refer to the datasheet for more low-level details. */
void spiInit(uint32_t clkdivide, uint32_t cpol, uint32_t ncpha);

/* Transmits a short (2 bytes) over SPI and returns the received short.
 *    -- send: the short to send over SPI
 *    -- return: the short received over SPI */
uint16_t spiSendReceive16(uint16_t send);

uint16_t spiSendReceive12(uint16_t send);

#endif
```

**We did not include USART, or GPIO libraries as they were not modified at all from the ones given to us in the course.*