University of Victoria

ECE 355

Microprocessor-Based Systems

Project Report

Alex Moody & Finn Berg
V00962257 & V00959800

November 10th, 2024

✓	*Problem Description/Specifications:*	*(5)*	_____
✓	*Design/Solution*	*(15)*	_____
✓	*Testing/Results*	*(10)*	_____
✓	*Discussion*	*(15)*	_____
✓	*Code Design and Documentation:*	*(15)*	_____
✓	***Total***	***(60)***	_____

Table of Contents

1 Problem Description

This ECE355 lab project covers the development of an embedded system using a STM32F0 Discovery board. The system is designed to implement microcontroller-based tasks such as signal measurement, PWM control, and a real-time OLED display. This system relies on peripherals such as timers, ADCs, DACs, polling,and interrupts to achieve its desired functionality. The project uses tools such as the ECLIPSE IDE for development of the software and debugging, and a 555 timer and function generator for generating square wave signals. The system must measure signal properties from more than one source, interact with external components like potentiometers, optocouplers and buttons, and display outputs on an OLED screen.

1.1 Objectives

1. Embedded Software:
    - Learning to write, debug, and run embedded C programs in the STM32F0 Discovery Board using the ECLIPSE IDE.
2. Breadboard and Microcontroller wiring:
    - Learn to wire a breadboard correctly so that the 555 timer and optocoupler works correctly with the STM32F0 board.
3. Measuring Frequency:
    - Design a system to measure the frequency of a square wave signal generated by a 555 timer using the TIM2 general-purpose timer and interrupts.
4. PWM signal Control:
    - Create and use ADC and DAC converters to monitor and control a PWM signal's frequency and duty cycle generated by the 555 timer via the optocoupler.
5. Real-time Display:
    - Interface an OLED screen to display the measured frequencies of the generated signals and calculated values of potentiometer resistance.
6. System Limits:
    - Be able to determine the operational limitations of the system such as detectable frequency ranges and measurable voltage limits and be able to understand what factors affect or create the limitations.

1.2 Specifications

1. Signal Frequency Measurement:
    - Measure the frequency of a square wave that has an amplitude of 0-3.3V.
    - Use PA2 (EXTI2) to catch the rising/falling edges of the wave to trigger the interrupt.

    - Display the calculated period and frequency values on the ECLIPSE console.
    - Find the range of frequencies that the system can detect.
2. PWM Monitoring and Control:
    - Use PA1 (EXTI1) to measure the frequency of the PWM signal generated by the 555 timer.
    - Use the DAC to control the optocoupler that in turn modulates the PWM signal.
    - Continuously monitor the potentiometer voltage using the ADC (PA65) and then use that voltage to calculate the corresponding resistance value.
    - Adjust the PWM properties based on the measured voltage.
3. User Interaction with System:
    - Use the USER button (PA0, EXTI0) to enable toggling between the signal sources. When pressed, the system will switch between the 555 timer and the function generator to receive a signal and display this switch and measured frequency on the console.
4. OLED Output Display:
    - Display real time data (frequency and resistance) on the OLED screen via SPI.
    - Configure the pins PB3-PB7 to properly interface with the screen.
5. System Limitations:
    - Find the frequency range measurable by the system.
    - Identify the voltage range limits for the ADC and DAC converters.

2 Design/Solution

2.1 Design Steps

2.1.1 Introductory Part 1 & 2

As there were many parts to the lab, our design process changed throughout the project. Initially, our goal was to get comfortable with eclipse and get used to being able to connect a controller to the computer. The microcontroller used throughout the project is the STM32F051R8T6. Part one of the lab simply involved plugging the controller into the computer through a micro USB cable. We then used the given code and made some changes such that the IDE could accurately display the frequency obtained from the function generator. The input pin from the function generator was connected to the STM via pin PA2. We validated this by adjusting the input frequency from the generator, and compared it with the displayed values from the computer. Initially, we had some problems getting the exact right value, which was due to floating point arithmetic. At first, we had the frequency variable set as an int, and due to rounding it would always display a frequency which was 1Hz off from the actual. Once we switched the variable to a float, we were able to obtain an accurate value.

2.1.2 Main Project

For the main project, we split up the tasks into smaller manageable parts so we could validate as we went. We began by wiring up the microcontroller with the other external components on the breadboard, following the schematic in Figure 1. To validate the wiring, the lab technician helped show us how to complete a continuity test where we were able to probe the board in different places to read the voltage. Once we were confident the wiring was correct, we downloaded the starter code from the ECE355 website [1]. The project had 4 main sections to complete. Rather than attempting all of them at once, we focused on the simpler parts first. We began with the ADC and CAD and button tasks. ADC refers to an analog to digital converter, and DAC vise versa. The analog input for the ADC is coming from either the function generator or the 555 timer. The conversion itself is done with a few lines of code using arithmetic (See appendix 5.1.2)**.** Coding the button was also relatively straightforward. The process involved connecting the button to PA0 of the controller and the other side to the 5v power supply. A debounce function was then created to eliminate multiple readings of the signal bounce. We then implemented an if loop to switch between the function generator (PA2) and the 555 timer (PA1) analog inputs. Once we had the button and converters working, we began attempting the screen SPI interface. Initially the process was very difficult but

thanks to the lecture videos we were able to establish a connection to the screen. The code involved declaring the six input pins to the display. Once we were able to write to the screen, we began attempting to display the proper values. The goal of the project was to display both the resistance from the potentiometer and the corresponding frequency. Pressing the button switches between the function generator and the 555 timer, which in-turn switches the shown frequency. The frequency changes as the potentiometer resistance varies. We were able to display the proper resistance, as well as the proper frequency from the function generator. Unfortunately, however, we were unable to successfully display the frequency from the 555 timer. This concluded the project, as we simply ran out of time to finish.

2.2 Lab Work Partitioning

During the first lab, we focused on learning how to use eclipse and how to read our device from the computer. We then focused on completing the second part of the introductory lab before the second lab section. We were able to finish the entire introductory section within the first lab section, which allowed us 3 full sessions to focus on the project. The second lab we focused on wiring up our microcontroller according to the schematic in figure 1. Once we thought we had it right we attempted to get our screen working. However, during our third lab we noticed that a couple of pins were not connected correctly, resulting in our screen not working. After fixing the wiring issue we were able to read and write to the screen. Also during the third lab time we got our ADC and DAC working, along with the button partially working. By the fourth lab section we had nearly every individual part of the project completed, and focused on combining all the code into a single working file. We ran into many problems here, as the code kept not working when fully combined. In the end, we were not able to get a fully working project, but still got partial marks as each of the parts worked properly on their own.

2.3 Coding Process and Considerations

For our project code, we began by downloading and running the example code provided on the ECE355 lab website [1]. To get our control registers properly initialized, we followed the lecture notes from the ECE355 website [3]. The “I/O Examples” and “Interface Examples” slides from the ECE355 website were extremely helpful for initializing the registers and declaring any necessary variables. GPIO registers were used to initialize the pins used from the microcontroller and declare them as either inputs or outputs. For the STM32 timer, we used the TIM2 register, which is a 32 bit counter. The signal for the TIM2 is given through the internal APB bus. The timer was used to generate interrupts in the code, as well as trigger events to happen. GPIO pins for the user button input, as well as the function generator/555 timer input were initialized according to the pins they were connected to on the STM. Finally, the display was initialized using the code provided in the lecture slides. There were a total of 5 pins which connected to the

screen (See appendix 5.1.5). There were numerous interrupts in the code, each set to a different priority. We had some problems with interrupts being overridden by others of higher priority, but eventually figured out the proper order to allow everything to work.

To debug and troubleshoot the code, we used numerous flags as well as a step through feature. We set up flags at the beginning of each interrupt and loop to show when the code reached each spot. To step through, we simply would run the code and use the step through/into buttons to determine line by line how the code was executing. This was extremely helpful especially once we combined everything into a single code file as otherwise it would be extremely difficult to debug.

2.4 Block Diagrams and Schematics

The following schematic is a representation of the wiring used for the breadboard circuit. The schematic specifically shows how the 555 timer and optocoupler are connected to the microcontroller. Power is fed to all three components through a common 5V bus, as well as a common ground located on the board provided. Data is sent through pin PA4 from the STM32 to the 4N35 optocoupler, and then to the 555 timer. From there, the signal generated by the timer is fed back to the microcontroller through pin PA1. Several capacitors are used in the circuit to reduce noise from the signal. Two resistors are also used to reduce the current flow to pins 7, 6, and 2 on the 555 timer and pins 4 and 5 on the optocoupler.

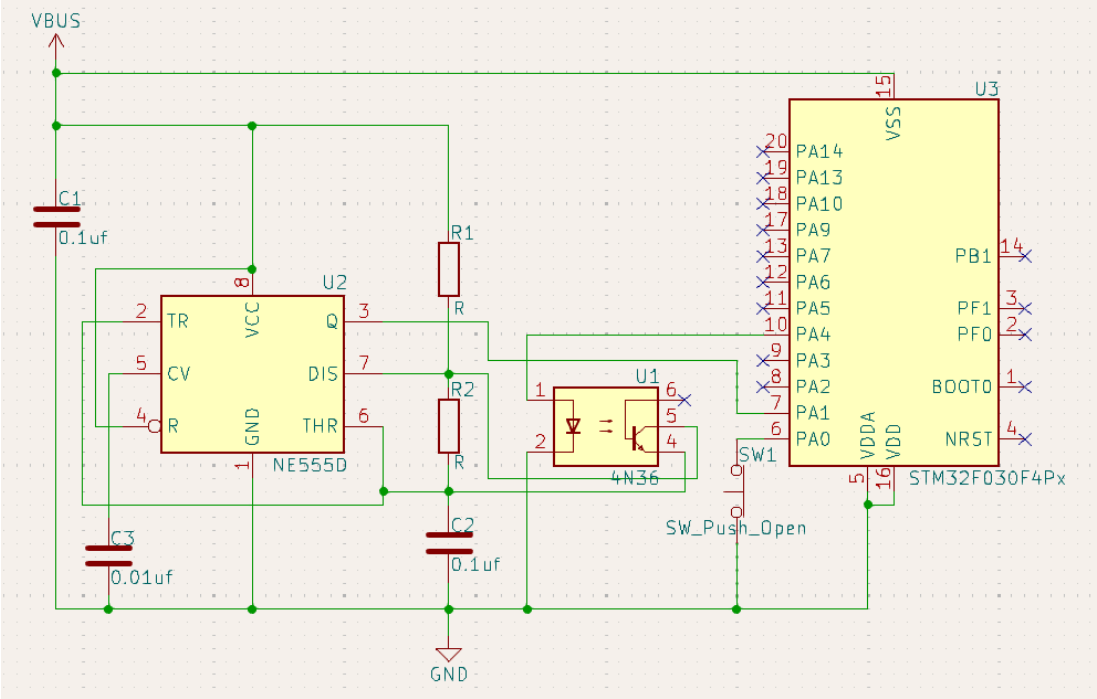

Figure 1: Breadboard schematic

Below is a high level diagram of the wiring layout for the project. The STM32 is the main microcontroller used, and the other components are either inputs or outputs to the device. A variable resistor is used to vary the input signal to PA5 of the STM. The range of the potentiometer is variable from 0 to 5000 ohms, providing a wide range of analog values to vary the timer. A simple push button is used and connected to pin PA0 of the STM. The goal of the button is to allow the device to switch its input waveform. A function generator is attached to pin PA1 of the controller to provide a constant input, and a 555 timer is attached to pin PA1 to provide a variable frequency square wave input. Pressing the button mentioned above switches the input waveform between the function generator and 555 timer. At any point, the output frequency and resistance is displayed to the screen through an SPI interface.

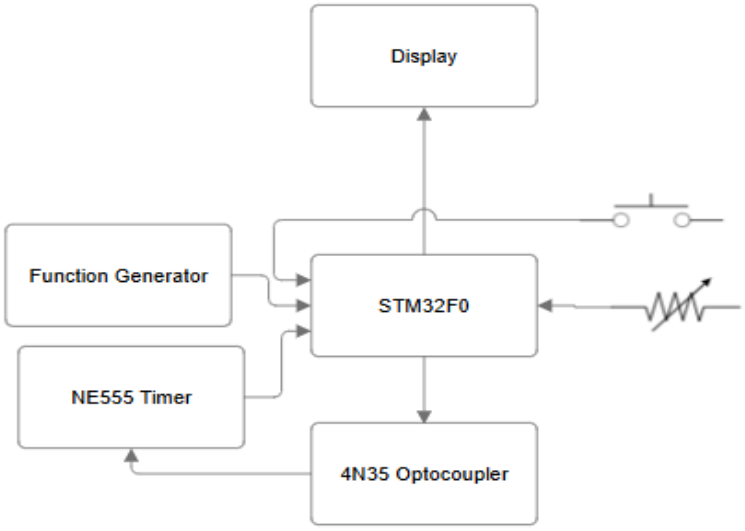

Figure 2: Block diagram

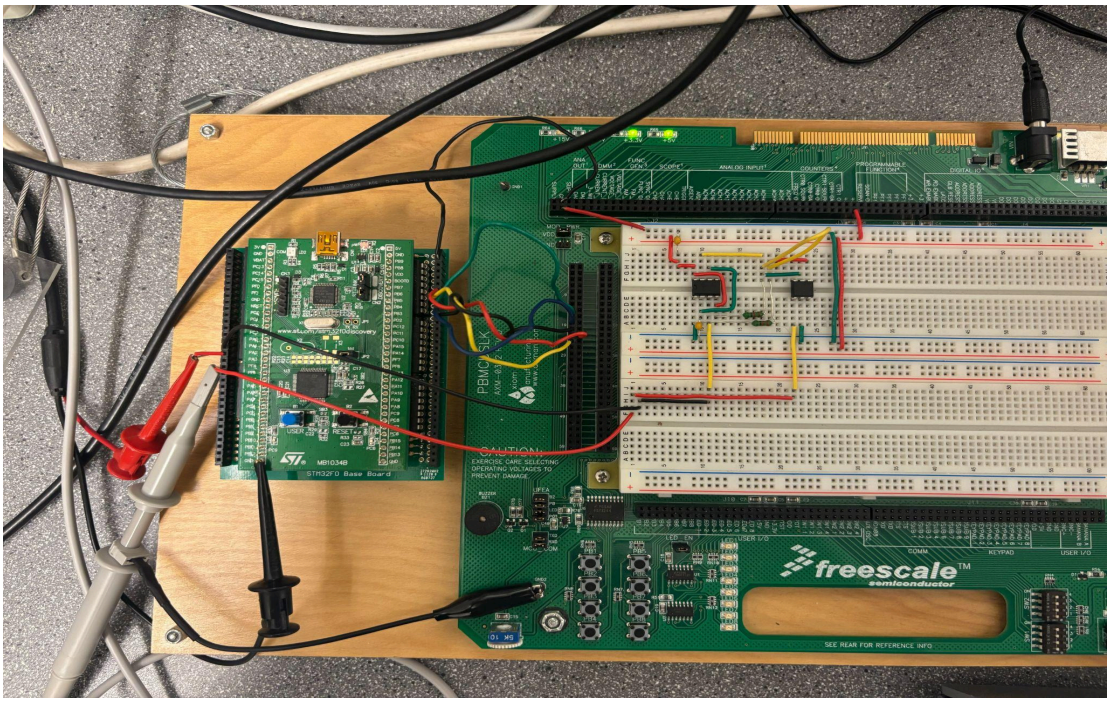

Figure 3: Breadboard and STM32 wiring

3 Testing and Results

3.1 Procedure

To test the project and system to ensure functionality and accuracy, the individual components were tested separately and compared against the desired functionality and outputs. For the frequency measurements, to check the accuracy, the value displayed on the console needed to match the input signal frequency to within 10% error. The USER button was tested by writing to the console whenever the button was pressed and switching between the input signal source and checking if that value was correct. The ADC and DAC values were printed to the console and checked to be accurate using math functions and manual calculations to ensure their accuracy. The OLED display screen was tested by writing multiple different characters to the screen that the code specified it to write and ensuring the characters were displayed correctly and in the correct location. These testing protocols ensured that the correct values were being read and received from all of the individual components.

3.2 Results and observations

The detectable frequency range is technically 1 micro HZ to 750,000 Hz. However, the function generator cannot produce 0 HZ but we can assume that our detectable range actually starts at 0 Hz. See table 1 for our frequency measurements and the measurement error. The frequency measurement had an average error of 0.2673050106%.

Table 1: Frequency output from function generator

Function Generator (Hz)	Period (ticks)	Console Frequency (Hz)	Frequency Error (%)
0.25	192645312	0.249163	0.3348
1	48162888	0.996618	0.3382
10	4815857	9.967073	0.32927
50	963271	49.830215	0.33957
1000	48160	996.822632	0.3177368
10000	4813	9972.990234	0.27009766
100000	479	100208.7656	0.208765625
750000	64	750000	0

For the measured resistance and ADC and DAC values we determined that the minimum voltage was 0 V and the maximum 3.3 V. Resistance ranged from 0 ohms to 5000 ohms. The ADC value ranged from 0 to 4095.This is due to the analog signal being converted to a 12 bit binary number

where the range is from 0 to 4095. Lastly, the DAC output voltage values ranged from 0 V to 3.3 V. See table 2 for the range of these values that were detected.

Table 2: ADC/DAC outputs at different resistances

Voltage (V)	Resistance (ohms)	ADC Value	DAC Output Voltage (v)
0	0	0	0
0.512	775.7575758	635.3454545	0.512
1.032	1563.636364	1280.618182	1.032
1.496	2266.666667	1856.4	1.496
2.052	3109.090909	2546.345455	2.052
2.513	3807.575758	3118.404545	2.513
3.0024	4549.090909	3725.705455	3.0024
3.3	5000	4095	3.3

The OLED screen had an observed resolution of 128 x 64 pixel display. The system was able to write to any of these pixels. However, the system divided the display up into rows and columns and used a character matrix of 8, 8-bit binary values to write characters to the screen.

While trying to integrate all of the individual components into one, there was difficulty with the interrupt priorities and ensuring that one interrupt did not dominate other interrupts. For example, when the signal frequency was too high, the correlated interrupt would dominate the other interrupts which would introduce delays in the refresh rate of other components.

4 Discussion

4.1 Specifications

Overall, our design met most of the specifications that were stated in section 1.2. We were able to measure the signal frequency from the signal generator effectively and display it on the console. The code used PA2 to catch the rising and falling edges of the wave to then trigger the interrupt (See appendix 5.1.1). The code used PA1 to measure the frequency of the PWM signal generated by the 555 timer. We were able to display this on the ECLIPSE console by implementing ADC and DAC functionalities. The ADC was used to monitor the potentiometer, find the voltage value, and calculate the resistance value which we also displayed on the console (See appendix 5.1.2). For the user button specification, we wrote code to determine when the

user button was pressed and correspondingly toggle between signal sources. We implemented functions to continuously interrupt using EXTI0 to check if the button was pressed and when pressed to toggle between which signal source to read from.The code then displayed the frequency value on the console (See appendix 5.1.4). Finally for the OLED display specification, we were able to write and display text on the OLED display screen. The code was able to display whatever we hardcoded it to display. To do this we had to configure pins PB3-PB7 to correctly interface with the screen. The code had functions to configure the OLED, configure the interrupts, to check for new frequency values using the interrupts, and to continuously check for new values to display to the OLED screen (See appendix 5.1.3).

4.2 Assumptions

1. The input signal received from the function generator and the 555 timer are stable and detectable frequencies within the range of TIM2.
2. The potentiometer voltage only varies within the measurable range of the ADC of 0-3.3V.
3. External components such as the OLED display screen, the potentiometer, the optocoupler, and the 555 timer were expected to operate correctly and reliably when set up correctly.
4. Interrupts and timers are assumed to operate fast enough for accurate frequency measurements.

4.3 Shortcomings

1. The detectable frequency range was narrower than we expected. We were unable to detect frequencies at the maximum end of what the function generator could produce. We could detect 750000Hz, while the function generator can produce 800000Hz.
2. The OLED screen had a delayed refresh rate whenever the frequency changed. If the frequency from the signal was high, then the interrupt was triggered more often and the refresh rate of the screen was slowed down.
3. The biggest shortcoming was that we were never able to get everything to work together cohesively. We had issues with interrupt priorities having all the different functionalities working together and with each other.

4.4 Extra features

The code design included error handling in the case of invalid measurements or other issues (see fig 4).

```
if(totalTicks == 0){
    trace_printf("Invalid measurement. Total ticks: 0\n");
} else{
    float frequency = SystemCoreClock / currentCount;
    trace_printf("Period: %u ticks, Frequency %f Hz\n", totalTicks, frequency);
}
```

Figure 4: Error handling of an invalid measurement

4.5 Summary

Overall, during this lab we were able to meet most of the design specifications set out for the project with the exception of piecing all the individual components together. We made some safe and reasonable assumptions and realized our system's shortcomings.

During this lab we also learned a variety of important and useful lessons. We learned how to configure and use STM32 peripherals including timers, ADC's, DACs, and SPI. We learned how to handle interrupts and the importance of optimizing the interrupt service routines to maintain the systems functionality and responsiveness. The challenges of integrating the hardware with the software were overcome through incremental development and testing. The difficulty of integrating many individual functionalities and peripherals into one system that works smoothly was a tough lesson to learn.

For future students undertaking this lab we would advise them on several important things. We realized the importance of thorough planning of the project before the implementation to have a base to build off of. Developing the system incrementally and testing diligently along the way was an important lesson and one that we would give to future students. This includes testing not only your code but also the components individually before integrating them into the full system. Using the debugging tools was helpful for finding problems in the code and learning how to do this effectively helped throughout the project. We would certainly advise that thorough documentation is kept throughout the development of the project to note important values, problems that came up, timelines, and anything else that could be referred back to.

This lab provided valuable insights into embedded system design, fostering a deeper understanding of microcontroller-based development and its practical applications in the real world.

5 Appendices

5.1 Code

5.1.1 - main.c

```
//
// This file is part of the GNU ARM Eclipse distribution.
// Copyright (c) 2014 Liviu Ionescu.
//
// ----------------------------------------------------------------------
// School: University of Victoria, Canada.
// Course: ECE 355 "Microprocessor-Based Systems".
// This is template code for Part 2 of Introductory Lab.
//
// See "system/include/cmsis/stm32f051x8.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f051x8.c" for handler declarations.
// ----------------------------------------------------------------------
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
#define myTIM2_PRESCALAR ((uint16_t)0x0000)
#define myTIME2_PERIOD ((uint32_t)0xFFFFFFFF)
// ----------------------------------------------------------------------
//
// STM32F0 empty sample (trace via $(trace)).
//
// Trace support is enabled by adding the TRACE macro definition.
// By default the trace messages are forwarded to the $(trace) output,
// but can be rerouted to any device or completely suppressed, by
// changing the definitions required in system/src/diag/trace_impl.c
// (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG/_STDOUT).
//
// ----- main() ---------------------------------------------------------
// Sample pragmas to cope with warnings. Please note the related line at
// the end of this function, used to pop the compiler diagnostics status.
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"
/* Definitions of registers and their bits are
  given in system/include/cmsis/stm32f051x8.h */
/* Clock prescaler for TIM2 timer: no prescaling */
#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for overflow */
#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)
void myGPIOA_Init(void);
```

```
void myTIM2_Init(void);
void myEXTI_Init(void);
// Declare/initialize your global variables here...
// NOTE: You'll need at least one global variable
// (say, timerTriggered = 0 or 1) to indicate
// whether TIM2 has started counting or not.
uint8_t timerTriggered = 0;
uint32_t lastCount = 0;
volatile uint32_t overFlowCount = 0;
/*** Call this function to boost the STM32F0xx clock to 48 MHz ***/
void SystemClock48MHz( void )
{
//
// Disable the PLL
//
  RCC->CR &= ~(RCC_CR_PLLON);
//
// Wait for the PLL to unlock
//
  while (( RCC->CR & RCC_CR_PLLRDY ) != 0 );
//
// Configure the PLL for 48-MHz system clock
//
  RCC->CFGR = 0x00280000;
//
// Enable the PLL
//
  RCC->CR |= RCC_CR_PLLON;
//
// Wait for the PLL to lock
//
  while (( RCC->CR & RCC_CR_PLLRDY ) != RCC_CR_PLLRDY );
//
// Switch the processor to the PLL clock source
//
  RCC->CFGR = ( RCC->CFGR & (~RCC_CFGR_SW_Msk)) | RCC_CFGR_SW_PLL;
//
// Update the system with the new clock frequency
//
  SystemCoreClockUpdate();
}
/*****************************************************************/
int
main(int argc, char* argv[])
{
        SystemClock48MHz();
        trace_printf("This is Part 2 of Introductory Lab...\n");
        trace_printf("System clock: %u Hz\n", SystemCoreClock);
        myGPIOA_Init();             /* Initialize I/O port PA */
```

```
	myTIM2_Init();		/* Initialize timer TIM2 */
	myEXTI_Init();		/* Initialize EXTI */
	while (1)
	{
		// Nothing is going on here...
	}
	return 0;
}
void myGPIOA_Init()
{
	/* Enable clock for GPIOA peripheral */
	// Relevant register: RCC->AHBENR
	RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
	/* Configure PA2 as input */
	// Relevant register: GPIOA->MODER
	GPIOA->MODER &= ~(GPIO_MODER_MODER2);
	/* Ensure no pull-up/pull-down for PA2 */
	// Relevant register: GPIOA->PUPDR
	GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR2);
}
void myTIM2_Init()
{
	/* Enable clock for TIM2 peripheral */
	// Relevant register: RCC->APB1ENR
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
	/* Configure TIM2: buffer auto-reload, count up, stop on overflow,
	 * enable update events, interrupt on overflow only */
	// Relevant register: TIM2->CR1
	TIM2->CR1 |= TIM_CR1_URS;
	/* Set clock prescaler value */
	TIM2->PSC = myTIM2_PRESCALER;
	/* Set auto-reloaded delay */
	TIM2->ARR = myTIM2_PERIOD;
	/* Update timer registers */
	// Relevant register: TIM2->EGR
	TIM2->EGR |= TIM_EGR_UG;
	/* Assign TIM2 interrupt priority = 0 in NVIC */
	// Relevant register: NVIC->IP[3], or use NVIC_SetPriority
	NVIC_SetPriority(TIM2_IRQn, 0);
	/* Enable TIM2 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(TIM2_IRQn);
	/* Enable update interrupt generation */
	// Relevant register: TIM2->DIER
	TIM2->DIER |= TIM_DIER_UIE;
}
void myEXTI_Init()
{
	/* Map EXTI2 line to PA2 */
```

```
	// Relevant register: SYSCFG->EXTICR[0]
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
	SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PA;
	/* EXTI2 line interrupts: set rising-edge trigger */
	// Relevant register: EXTI->RTSR
	EXTI->RTSR |= EXTI_RTSR_TR2;
	/* Unmask interrupts from EXTI2 line */
	// Relevant register: EXTI->IMR
	EXTI->IMR |= EXTI_IMR_MR2;
	/* Assign EXTI2 interrupt priority = 0 in NVIC */
	// Relevant register: NVIC->IP[2], or use NVIC_SetPriority
	NVIC_SetPriority(EXTI2_3_IRQn, 0);
	/* Enable EXTI2 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(EXTI2_3_IRQn);
}
/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void TIM2_IRQHandler()
{
	/* Check if update interrupt flag is indeed set */
	if ((TIM2->SR & TIM_SR_UIF) != 0)
	{
		trace_printf("\n*** Overflow! ***\n");
		/* Clear update interrupt flag */
		// Relevant register: TIM2->SR
		TIM2->SR &= ~TIM_SR_UIF;
		/* Restart stopped timer */
		// Relevant register: TIM2->CR1
		TIM2->CR1 |= TIM_CR1_CEN;
	}
}
/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void EXTI2_3_IRQHandler()
{
	//
	// 1. If this is the first edge:
	//	- Clear count register (TIM2->CNT).
	//	- Start timer (TIM2->CR1).
	//   Else (this is the second edge):
	//	- Stop timer (TIM2->CR1).
	//	- Read out count register (TIM2->CNT).
	//	- Calculate signal period and frequency.
	//	- Print calculated values to the console.
	//	  NOTE: Function trace_printf does not work
	//	  with floating-point numbers: you must use
	//	  "unsigned int" type to print your signal
	//	  period and frequency.
	//
	// 2. Clear EXTI2 interrupt pending flag (EXTI->PR).
```

```
		// NOTE: A pending register (PR) bit is cleared
		// by writing 1 to it.
		//
		/* Check if EXTI2 interrupt pending flag is indeed set */
		if ((EXTI->PR & EXTI_PR_PR2) != 0)
		{
			if (timerTriggered == 0){
				TIM2->CNT = 0;
				TIM2->CR1 |= TIM_CR1_CEN;
				timerTriggered = 1;
			} else{
				TIM2->CR1 &= ~TIM_CR1_CEN;
				float currentCount = TIM2->CNT;
				uint32_t totalTicks = currentCount - lastCount;
				if(totalTicks == 0){
					trace_printf("Invalid measurement. Total ticks: 0\n");
				} else{
					float frequency = SystemCoreClock / currentCount;
					trace_printf("Period: %u ticks, Frequency %f Hz\n", totalTicks, frequency);
				}
				timerTriggered = 0;
			}
			EXTI->PR |= EXTI_PR_PR2;
		}
}
#pragma GCC diagnostic pop
// ----------------------------------------------------------------------------
```

5.1.2 - mainADCDAC.c

This code is very similar to mian.c from 5.1.1, with just the ADC and DAC functionality added in main and some initialization functions.

```
//
// This file is part of the GNU ARM Eclipse distribution.
// Copyright (c) 2014 Liviu Ionescu.
//
// ----------------------------------------------------------------------------
// School: University of Victoria, Canada.
// Course: ECE 355 "Microprocessor-Based Systems".
// This is template code for Part 2 of Introductory Lab.
//
// See "system/include/cmsis/stm32f051x8.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f051x8.c" for handler declarations.
// ----------------------------------------------------------------------------
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
```

```
#define myTIM2_PRESCALAR ((uint16_t)0x0000)
#define myTIME2_PERIOD ((uint32_t)0xFFFFFFFF)
#define ADC_MAX_VALUE 4095  // 12-bit ADC max value
#define VREF 3.3            // Reference voltage for ADC
#define POTENTIOMETER_MAX_RESISTANCE 5000  // Assume a 5k potentiometer
// ----------------------------------------------------------------------------
//
// STM32F0 empty sample (trace via $(trace)).
//
// Trace support is enabled by adding the TRACE macro definition.
// By default the trace messages are forwarded to the $(trace) output,
// but can be rerouted to any device or completely suppressed, by
// changing the definitions required in system/src/diag/trace_impl.c
// (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG/_STDOUT).
//
// ----- main() ---------------------------------------------------------------
// Sample pragmas to cope with warnings. Please note the related line at
// the end of this function, used to pop the compiler diagnostics status.
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"
/* Definitions of registers and their bits are
  given in system/include/cmsis/stm32f051x8.h */
/* Clock prescaler for TIM2 timer: no prescaling */
#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for overflow */
#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)
//function declarations
void myGPIOA_Init(void);
void myTIM2_Init(void);
void myEXTI_Init(void);
void myADC_Init(void);
void myDAC_Init(void);
// Declare/initialize your global variables here...
// NOTE: You'll need at least one global variable
// (say, timerTriggered = 0 or 1) to indicate
// whether TIM2 has started counting or not.
uint8_t timerTriggered = 0;
uint32_t lastCount = 0;
volatile uint32_t overFlowCount = 0;
/*** Call this function to boost the STM32F0xx clock to 48 MHz ***/
void SystemClock48MHz( void )
{
//
// Disable the PLL
//
  RCC->CR &= ~(RCC_CR_PLLON);
//
```

```
// Wait for the PLL to unlock
//
  while (( RCC->CR & RCC_CR_PLLRDY ) != 0 );
//
// Configure the PLL for 48-MHz system clock
//
  RCC->CFGR = 0x00280000;
//
// Enable the PLL
//
  RCC->CR |= RCC_CR_PLLON;
//
// Wait for the PLL to lock
//
  while (( RCC->CR & RCC_CR_PLLRDY ) != RCC_CR_PLLRDY );
//
// Switch the processor to the PLL clock source
//
  RCC->CFGR = ( RCC->CFGR & (~RCC_CFGR_SW_Msk)) | RCC_CFGR_SW_PLL;
//
// Update the system with the new clock frequency
//
  SystemCoreClockUpdate();
}
/*****************************************************************/
int
main(int argc, char* argv[])
{
        SystemClock48MHz();
        trace_printf("This is Part 2 of Introductory Lab...\n");
        trace_printf("System clock: %u Hz\n", SystemCoreClock);
        myGPIOA_Init();         /* Initialize I/O port PA */
        myTIM2_Init();          /* Initialize timer TIM2 */
        myEXTI_Init();          /* Initialize EXTI */
        myADC_Init();       // Initialize ADC for potentiometer readings
  myDAC_Init();       // Initialize DAC to control the optocoupler
  uint16_t adc_value;
  float voltage, resistance;
        while (1){
    // Start ADC conversion
    ADC1->CR |= ADC_CR_ADSTART;
    // Wait for ADC conversion to complete
    while (!(ADC1->ISR & ADC_ISR_EOC)) {}
    // Read ADC value
    adc_value = ADC1->DR;
    // Calculate the potentiometer voltage
    voltage = (adc_value * VREF) / ADC_MAX_VALUE;
    // Calculate potentiometer resistance based on voltage
    resistance = (voltage / VREF) * POTENTIOMETER_MAX_RESISTANCE;
```

```
    // Output the potentiometer voltage to the DAC
    uint32_t dac_value = (uint32_t)((voltage / VREF) * 4095);
    DAC->DHR12R1 = dac_value;
    trace_printf("ADC Value: %u, Voltage: %f V, Resistance: %f Ohms\n", adc_value, voltage, resistance);
    trace_printf("DAC Output Voltage: %f V (from ADC value)\n", (dac_value * VREF) / 4095);
        }
        return 0;
}
void myADC_Init(void)
{
  RCC->APB2ENR |= RCC_APB2ENR_ADCEN;          // Enable ADC clock
  ADC1->CFGR1 &= ~ADC_CFGR1_RES;              // Set ADC resolution to 12-bit
  ADC1->CHSELR = ADC_CHSELR_CHSEL5;           // Select channel 5 (assume potentiometer connected to
PA5)
  ADC1->CFGR1 &= ~ADC_CFGR1_ALIGN;            // Right-align the result
  ADC1->CR |= ADC_CR_ADEN;                    // Enable the ADC
  while (!(ADC1->ISR & ADC_ISR_ADRDY)) {}     // Wait for ADC to be ready
}
void myDAC_Init(void)
{
  RCC->APB1ENR |= RCC_APB1ENR_DACEN;          // Enable DAC clock
  DAC->CR |= DAC_CR_EN1;                      // Enable DAC channel 1
}
void myGPIOA_Init()
{
        //initializing GPIO pins
        /* Enable clock for GPIOA peripheral */
        // Relevant register: RCC->AHBENR
        RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
        /* Configure PA2 as input */
        // Relevant register: GPIOA->MODER
        GPIOA->MODER &= ~(GPIO_MODER_MODER2);
        /* Ensure no pull-up/pull-down for PA2 */
        // Relevant register: GPIOA->PUPDR
        GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR2);
}
void myTIM2_Init()
{
        //initialize timer
        /* Enable clock for TIM2 peripheral */
        // Relevant register: RCC->APB1ENR
        RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
        /* Configure TIM2: buffer auto-reload, count up, stop on overflow,
         * enable update events, interrupt on overflow only */
        // Relevant register: TIM2->CR1
        TIM2->CR1 |= TIM_CR1_URS;
        /* Set clock prescaler value */
        TIM2->PSC = myTIM2_PRESCALER;
        /* Set auto-reloaded delay */
```

```
	TIM2->ARR = myTIM2_PERIOD;
	/* Update timer registers */
	// Relevant register: TIM2->EGR
	TIM2->EGR |= TIM_EGR_UG;
	/* Assign TIM2 interrupt priority = 0 in NVIC */
	// Relevant register: NVIC->IP[3], or use NVIC_SetPriority
	NVIC_SetPriority(TIM2_IRQn, 0);
	/* Enable TIM2 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(TIM2_IRQn);
	/* Enable update interrupt generation */
	// Relevant register: TIM2->DIER
	TIM2->DIER |= TIM_DIER_UIE;
}
void myEXTI_Init()
{
	//initializing interrupt lines
	/* Map EXTI2 line to PA2 */
	// Relevant register: SYSCFG->EXTICR[0]
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
	SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PA;
	/* EXTI2 line interrupts: set rising-edge trigger */
	// Relevant register: EXTI->RTSR
	EXTI->RTSR |= EXTI_RTSR_TR2;
	/* Unmask interrupts from EXTI2 line */
	// Relevant register: EXTI->IMR
	EXTI->IMR |= EXTI_IMR_MR2;
	/* Assign EXTI2 interrupt priority = 0 in NVIC */
	// Relevant register: NVIC->IP[2], or use NVIC_SetPriority
	NVIC_SetPriority(EXTI2_3_IRQn, 0);
	/* Enable EXTI2 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(EXTI2_3_IRQn);
}
/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void TIM2_IRQHandler()
{
	/* Check if update interrupt flag is indeed set */
	if ((TIM2->SR & TIM_SR_UIF) != 0)
	{
		trace_printf("\n*** Overflow! ***\n");
		/* Clear update interrupt flag */
		// Relevant register: TIM2->SR
		TIM2->SR &= ~TIM_SR_UIF;
		/* Restart stopped timer */
		// Relevant register: TIM2->CR1
		TIM2->CR1 |= TIM_CR1_CEN;
	}
}
```

```
/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void EXTI2_3_IRQHandler()
{
	//
	// 1. If this is the first edge:
	//	- Clear count register (TIM2->CNT).
	//	- Start timer (TIM2->CR1).
	//   Else (this is the second edge):
	//	- Stop timer (TIM2->CR1).
	//	- Read out count register (TIM2->CNT).
	//	- Calculate signal period and frequency.
	//	- Print calculated values to the console.
	//	  NOTE: Function trace_printf does not work
	//	  with floating-point numbers: you must use
	//	  "unsigned int" type to print your signal
	//	  period and frequency.
	//
	// 2. Clear EXTI2 interrupt pending flag (EXTI->PR).
	// NOTE: A pending register (PR) bit is cleared
	// by writing 1 to it.
	//
	/* Check if EXTI2 interrupt pending flag is indeed set */
	if ((EXTI->PR & EXTI_PR_PR2) != 0)
	{
		if (timerTriggered == 0){
			TIM2->CNT = 0;
			TIM2->CR1 |= TIM_CR1_CEN;
			timerTriggered = 1;
		} else{
			TIM2->CR1 &= ~TIM_CR1_CEN;
			float currentCount = TIM2->CNT;
			uint32_t totalTicks = currentCount - lastCount;
			if(totalTicks == 0){
				trace_printf("Invalid measurement. Total ticks: 0\n");
			} else{
				//uint32_t frequency = SystemCoreClock / totalTicks;
				//float frequency = SystemCoreClock / currentCount;
				//trace_printf("Period: %u ticks, Frequency %f Hz\n", totalTicks, frequency);
			}
			timerTriggered = 0;
		}
		EXTI->PR |= EXTI_PR_PR2;
	}
}
#pragma GCC diagnostic pop
// ----------------------------------------------------------------------------
```

5.1.3 - mainOLED.c

```
#include <stdio.h>
#include "diag/Trace.h"
#include <string.h>
#include "cmsis/cmsis_device.h"
#include "stm32f0xx.h"  // This includes the RCC register definitions
//#include "stm32f0xx_gpio.h"
#include "stm32f0xx_hal.h"
#include "stm32f0xx_hal_spi.h"
#include "stm32f0xx_hal_gpio.h"
#include "stm32f0xx_hal_rcc.h"
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"
// Global variables
unsigned int Freq = 0;  // Measured frequency value
unsigned int Res = 0;   // Measured resistance value
// SPI Handle
SPI_HandleTypeDef SPI_Handle;
// OLED Initialization Commands
unsigned char oled_init_cmds[] = {
  0xAE, 0x20, 0x00, 0x40, 0xA0 | 0x01, 0xA8, 0x40 - 1, 0xC0 | 0x08, 0xD3, 0x00,
  0xDA, 0x32, 0xD5, 0x80, 0xD9, 0x22, 0xDB, 0x30, 0x81, 0xFF, 0xA4, 0xA6,
  0xAD, 0x30, 0x8D, 0x10, 0xAE | 0x01, 0xC0, 0xA0
};
// ASCII Character Data for Display
unsigned char Characters[][8] = {     {0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
```

```
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
{0b00000000, 0b00000000, 0b01011111, 0b00000000, 0b00000000,0b00000000, 0b00000000, 0b00000000},  // !
```

```
        {0b00000000, 0b00000111, 0b00000000, 0b00000111, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // "
        {0b00010100, 0b01111111, 0b00010100, 0b01111111, 0b00010100,0b00000000, 0b00000000,
0b00000000},  // #
        {0b00100100, 0b00101010, 0b01111111, 0b00101010, 0b00010010,0b00000000, 0b00000000,
0b00000000},  // $
        {0b00100011, 0b00010011, 0b00001000, 0b01100100, 0b01100010,0b00000000, 0b00000000,
0b00000000},  // %
        {0b00110110, 0b01001001, 0b01010101, 0b00100010, 0b01010000,0b00000000, 0b00000000,
0b00000000},  // &
        {0b00000000, 0b00000101, 0b00000011, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // '
        {0b00000000, 0b00011100, 0b00100010, 0b01000001, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // (
        {0b00000000, 0b01000001, 0b00100010, 0b00011100, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // )
        {0b00010100, 0b00001000, 0b00111110, 0b00001000, 0b00010100,0b00000000, 0b00000000,
0b00000000},  // *
        {0b00001000, 0b00001000, 0b00111110, 0b00001000, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // +
        {0b00000000, 0b01010000, 0b00110000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // ,
        {0b00001000, 0b00001000, 0b00001000, 0b00001000, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // -
        {0b00000000, 0b01100000, 0b01100000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // .
        {0b00100000, 0b00010000, 0b00001000, 0b00000100, 0b00000010,0b00000000, 0b00000000,
0b00000000},  // /
        {0b00111110, 0b01010001, 0b01001001, 0b01000101, 0b00111110,0b00000000, 0b00000000,
0b00000000},  // 0
        {0b00000000, 0b01000010, 0b01111111, 0b01000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // 1
        {0b01000010, 0b01100001, 0b01010001, 0b01001001, 0b01000110,0b00000000, 0b00000000,
0b00000000},  // 2
        {0b00100001, 0b01000001, 0b01000101, 0b01001011, 0b00110001,0b00000000, 0b00000000,
0b00000000},  // 3
        {0b00011000, 0b00010100, 0b00010010, 0b01111111, 0b00010000,0b00000000, 0b00000000,
0b00000000},  // 4
        {0b00100111, 0b01000101, 0b01000101, 0b01000101, 0b00111001,0b00000000, 0b00000000,
0b00000000},  // 5
        {0b00111100, 0b01001010, 0b01001001, 0b01001001, 0b00110000,0b00000000, 0b00000000,
0b00000000},  // 6
        {0b00000011, 0b00000001, 0b01110001, 0b00001001, 0b00000111,0b00000000, 0b00000000,
0b00000000},  // 7
        {0b00110110, 0b01001001, 0b01001001, 0b01001001, 0b00110110,0b00000000, 0b00000000,
0b00000000},  // 8
        {0b00000110, 0b01001001, 0b01001001, 0b00101001, 0b00011110,0b00000000, 0b00000000,
0b00000000},  // 9
```

```
        {0b00000000, 0b00110110, 0b00110110, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // :
        {0b00000000, 0b01010110, 0b00110110, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // ;
        {0b00001000, 0b00010100, 0b00100010, 0b01000001, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // <
        {0b00010100, 0b00010100, 0b00010100, 0b00010100, 0b00010100,0b00000000, 0b00000000,
0b00000000},  // =
        {0b00000000, 0b01000001, 0b00100010, 0b00010100, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // >
        {0b00000010, 0b00000001, 0b01010001, 0b00001001, 0b00000110,0b00000000, 0b00000000,
0b00000000},  // ?
        {0b00110010, 0b01001001, 0b01111001, 0b01000001, 0b00111110,0b00000000, 0b00000000,
0b00000000},  // @
        {0b01111110, 0b00010001, 0b00010001, 0b00010001, 0b01111110,0b00000000, 0b00000000,
0b00000000},  // A
        {0b01111111, 0b01001001, 0b01001001, 0b01001001, 0b00110110,0b00000000, 0b00000000,
0b00000000},  // B
        {0b00111110, 0b01000001, 0b01000001, 0b01000001, 0b00100010,0b00000000, 0b00000000,
0b00000000},  // C
        {0b01111111, 0b01000001, 0b01000001, 0b00100010, 0b00011100,0b00000000, 0b00000000,
0b00000000},  // D
        {0b01111111, 0b01001001, 0b01001001, 0b01001001, 0b01000001,0b00000000, 0b00000000,
0b00000000},  // E
        {0b01111111, 0b00001001, 0b00001001, 0b00001001, 0b00000001,0b00000000, 0b00000000,
0b00000000},  // F
        {0b00111110, 0b01000001, 0b01001001, 0b01001001, 0b01111010,0b00000000, 0b00000000,
0b00000000},  // G
        {0b01111111, 0b00001000, 0b00001000, 0b00001000, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // H
        {0b01000000, 0b01000001, 0b01111111, 0b01000001, 0b01000000,0b00000000, 0b00000000,
0b00000000},  // I
        {0b00100000, 0b01000000, 0b01000001, 0b00111111, 0b00000001,0b00000000, 0b00000000,
0b00000000},  // J
        {0b01111111, 0b00001000, 0b00010100, 0b00100010, 0b01000001,0b00000000, 0b00000000,
0b00000000},  // K
        {0b01111111, 0b01000000, 0b01000000, 0b01000000, 0b01000000,0b00000000, 0b00000000,
0b00000000},  // L
        {0b01111111, 0b00000010, 0b00001100, 0b00000010, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // M
        {0b01111111, 0b00000100, 0b00001000, 0b00010000, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // N
        {0b00111110, 0b01000001, 0b01000001, 0b01000001, 0b00111110,0b00000000, 0b00000000,
0b00000000},  // O
        {0b01111111, 0b00001001, 0b00001001, 0b00001001, 0b00000110,0b00000000, 0b00000000,
0b00000000},  // P
        {0b00111110, 0b01000001, 0b01010001, 0b00100001, 0b01011110,0b00000000, 0b00000000,
0b00000000},  // Q
```

```
        {0b01111111, 0b00001001, 0b00011001, 0b00101001, 0b01000110,0b00000000, 0b00000000,
0b00000000},  // R
        {0b01000110, 0b01001001, 0b01001001, 0b01001001, 0b00110001,0b00000000, 0b00000000,
0b00000000},  // S
        {0b00000001, 0b00000001, 0b01111111, 0b00000001, 0b00000001,0b00000000, 0b00000000,
0b00000000},  // T
        {0b00111111, 0b01000000, 0b01000000, 0b01000000, 0b00111111,0b00000000, 0b00000000,
0b00000000},  // U
        {0b00011111, 0b00100000, 0b01000000, 0b00100000, 0b00011111,0b00000000, 0b00000000,
0b00000000},  // V
        {0b00111111, 0b01000000, 0b00111000, 0b01000000, 0b00111111,0b00000000, 0b00000000,
0b00000000},  // W
        {0b01100011, 0b00010100, 0b00001000, 0b00010100, 0b01100011,0b00000000, 0b00000000,
0b00000000},  // X
        {0b00000111, 0b00001000, 0b01110000, 0b00001000, 0b00000111,0b00000000, 0b00000000,
0b00000000},  // Y
        {0b01100001, 0b01010001, 0b01001001, 0b01000101, 0b01000011,0b00000000, 0b00000000,
0b00000000},  // Z
        {0b01111111, 0b01000001, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // [
        {0b00010101, 0b00010110, 0b01111100, 0b00010110, 0b00010101,0b00000000, 0b00000000,
0b00000000},  // back slash
        {0b00000000, 0b00000000, 0b00000000, 0b01000001, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // ]
        {0b00000100, 0b00000010, 0b00000001, 0b00000010, 0b00000100,0b00000000, 0b00000000,
0b00000000},  // ^
        {0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b01000000,0b00000000, 0b00000000,
0b00000000},  // _
        {0b00000000, 0b00000001, 0b00000010, 0b00000100, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // `
        {0b00100000, 0b01010100, 0b01010100, 0b01010100, 0b01111000,0b00000000, 0b00000000,
0b00000000},  // a
        {0b01111111, 0b01001000, 0b01000100, 0b01000100, 0b00111000,0b00000000, 0b00000000,
0b00000000},  // b
        {0b00111000, 0b01000100, 0b01000100, 0b01000100, 0b00100000,0b00000000, 0b00000000,
0b00000000},  // c
        {0b00111000, 0b01000100, 0b01000100, 0b01001000, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // d
        {0b00111000, 0b01010100, 0b01010100, 0b01010100, 0b00011000,0b00000000, 0b00000000,
0b00000000},  // e
        {0b00001000, 0b01111110, 0b00001001, 0b00000001, 0b00000010,0b00000000, 0b00000000,
0b00000000},  // f
        {0b00001100, 0b01010010, 0b01010010, 0b01010010, 0b00111110,0b00000000, 0b00000000,
0b00000000},  // g
        {0b01111111, 0b00001000, 0b00000100, 0b00000100, 0b01111000,0b00000000, 0b00000000,
0b00000000},  // h
        {0b00000000, 0b01000100, 0b01111101, 0b01000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // i
```

```
        {0b00100000, 0b01000000, 0b01000100, 0b00111101, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // j
        {0b01111111, 0b00010000, 0b00101000, 0b01000100, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // k
        {0b00000000, 0b01000001, 0b01111111, 0b01000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // l
        {0b01111100, 0b00000100, 0b00011000, 0b00000100, 0b01111000,0b00000000, 0b00000000,
0b00000000},  // m
        {0b01111100, 0b00001000, 0b00000100, 0b00000100, 0b01111000,0b00000000, 0b00000000,
0b00000000},  // n
        {0b00111000, 0b01000100, 0b01000100, 0b01000100, 0b00111000,0b00000000, 0b00000000,
0b00000000},  // o
        {0b01111100, 0b00010100, 0b00010100, 0b00010100, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // p
        {0b00001000, 0b00010100, 0b00010100, 0b00011000, 0b01111100,0b00000000, 0b00000000,
0b00000000},  // q
        {0b01111100, 0b00001000, 0b00000100, 0b00000100, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // r
        {0b01001000, 0b01010100, 0b01010100, 0b01010100, 0b00100000,0b00000000, 0b00000000,
0b00000000},  // s
        {0b00000100, 0b00111111, 0b01000100, 0b01000000, 0b00100000,0b00000000, 0b00000000,
0b00000000},  // t
        {0b00111100, 0b01000000, 0b01000000, 0b00100000, 0b01111100,0b00000000, 0b00000000,
0b00000000},  // u
        {0b00011100, 0b00100000, 0b01000000, 0b00100000, 0b00011100,0b00000000, 0b00000000,
0b00000000},  // v
        {0b00111100, 0b01000000, 0b00111000, 0b01000000, 0b00111100,0b00000000, 0b00000000,
0b00000000},  // w
        {0b01000100, 0b00101000, 0b00010000, 0b00101000, 0b01000100,0b00000000, 0b00000000,
0b00000000},  // x
        {0b00001100, 0b01010000, 0b01010000, 0b01010000, 0b00111100,0b00000000, 0b00000000,
0b00000000},  // y
        {0b01000100, 0b01100100, 0b01010100, 0b01001100, 0b01000100,0b00000000, 0b00000000,
0b00000000},  // z
        {0b00000000, 0b00001000, 0b00110110, 0b01000001, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // {
        {0b00000000, 0b00000000, 0b01111111, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // |
        {0b00000000, 0b01000001, 0b00110110, 0b00001000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // }
        {0b00001000, 0b00001000, 0b00101010, 0b00011100, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // ~
        {0b00001000, 0b00011100, 0b00101010, 0b00001000, 0b00001000,0b00000000, 0b00000000,
0b00000000}   // <-
};
// Function Prototypes
void oled_Write(unsigned char Value);
void oled_Write_Cmd(unsigned char cmd);
void oled_Write_Data(unsigned char data);
```

```
void oled_config(void);
void refresh_OLED(void);
void delay_ms(uint32_t ms);
void SystemClock48MHz(void);
void SystemClock48MHz(void)
{
  RCC->CR &= ~(RCC_CR_PLLON);
  while ((RCC->CR & RCC_CR_PLLRDY) != 0);
  RCC->CFGR = 0x00280000;
  RCC->CR |= RCC_CR_PLLON;
  while ((RCC->CR & RCC_CR_PLLRDY) != RCC_CR_PLLRDY);
  RCC->CFGR = (RCC->CFGR & (~RCC_CFGR_SW_Msk)) | RCC_CFGR_SW_PLL;
  SystemCoreClockUpdate();
}
int main(int argc, char* argv[])
{
  SystemClock48MHz();
  oled_config();
  while (1)
  {
     refresh_OLED();
  }
}
void oled_config(void)
{
  // Enable GPIOB and SPI1 clocks
  RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
  RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
  // Configure PB3 (SCK) and PB5 (MOSI) as AF0
  GPIOB->MODER &= ~(GPIO_MODER_MODER3 | GPIO_MODER_MODER5);
  GPIOB->MODER |= (GPIO_MODER_MODER3_1 | GPIO_MODER_MODER5_1);
  GPIOB->AFR[0] &= ~((0xF << (3 * 4)) | (0xF << (5 * 4)));
  // Configure PB6 (CS#), PB7 (D/C#), PB4 (RES#) as outputs
  GPIOB->MODER &= ~(GPIO_MODER_MODER4 | GPIO_MODER_MODER6 | GPIO_MODER_MODER7);
  GPIOB->MODER |= (GPIO_MODER_MODER4_0 | GPIO_MODER_MODER6_0 |
GPIO_MODER_MODER7_0);
  // Reset OLED Display
  GPIOB->ODR &= ~GPIO_ODR_4;
  //HAL_Delay(10);
  delay_ms(10);
  GPIOB->ODR |= GPIO_ODR_4;
  delay_ms(10);
  //HAL_Delay(10);
  // SPI Configuration
  SPI_Handle.Instance = SPI1;
  SPI_Handle.Init.Direction = SPI_DIRECTION_1LINE;
  SPI_Handle.Init.Mode = SPI_MODE_MASTER;
  SPI_Handle.Init.DataSize = SPI_DATASIZE_8BIT;
  SPI_Handle.Init.CLKPolarity = SPI_POLARITY_LOW;
```

```
  SPI_Handle.Init.CLKPhase = SPI_PHASE_1EDGE;
  SPI_Handle.Init.NSS = SPI_NSS_SOFT;
  SPI_Handle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
  SPI_Handle.Init.FirstBit = SPI_FIRSTBIT_MSB;
  SPI_Handle.Init.CRCPolynomial = 7;
  HAL_SPI_Init(&SPI_Handle);
  __HAL_SPI_ENABLE(&SPI_Handle);
  // Send Initialization Commands to OLED
  for (unsigned int i = 0; i < sizeof(oled_init_cmds); i++) {
    oled_Write_Cmd(oled_init_cmds[i]);
  }
  // Clear Display
  for (int page = 0; page < 8; page++) {
    oled_Write_Cmd(0xB0 | page);
    oled_Write_Cmd(0x00);
    oled_Write_Cmd(0x10);
    for (int seg = 0; seg < 128; seg++) {
      oled_Write_Data(0x00);
    }
  }
}
void refresh_OLED(void)
{
  unsigned char Buffer[17];
  // Display Resistance
  snprintf(Buffer, sizeof(Buffer), "R: %5u Ohms", Res);
  oled_Write_Cmd(0xB2); // Page 2
  oled_Write_Cmd(0x03);
  oled_Write_Cmd(0x10);
  for (int i = 0; i < strlen(Buffer); i++) {
    unsigned char c = Buffer[i];
    for (int j = 0; j < 8; j++) {
      oled_Write_Data(Characters[c][j]);
    }
  }
  // Display Frequency
  snprintf(Buffer, sizeof(Buffer), "F: %5u Hz", Freq);
  oled_Write_Cmd(0xB3); // Page 3
  oled_Write_Cmd(0x03);
  oled_Write_Cmd(0x10);
  for (int i = 0; i < strlen(Buffer); i++) {
    unsigned char c = Buffer[i];
    for (int j = 0; j < 8; j++) {
      oled_Write_Data(Characters[c][j]);
    }
  }
  delay_ms(100);
}
void oled_Write(unsigned char Value)
```

```
{
  while (!(SPI1->SR & SPI_SR_TXE));
  HAL_SPI_Transmit(&SPI_Handle, &Value, 1, HAL_MAX_DELAY);
  while (SPI1->SR & SPI_SR_BSY);
}
void oled_Write_Cmd(unsigned char cmd)
{
  GPIOB->ODR |= GPIO_ODR_6;
  GPIOB->ODR &= ~GPIO_ODR_7;
  GPIOB->ODR &= ~GPIO_ODR_6;
  oled_Write(cmd);
  GPIOB->ODR |= GPIO_ODR_6;
}
void oled_Write_Data(unsigned char data)
{
  GPIOB->ODR |= GPIO_ODR_6;
  GPIOB->ODR |= GPIO_ODR_7;
  GPIOB->ODR &= ~GPIO_ODR_6;
  oled_Write(data);
  GPIOB->ODR |= GPIO_ODR_6;
}
//this was the original delay function we used while developing the code. We kept it in for reference even though it is
no longer used.
/*void delay_ms(uint32_t ms)
{
  __HAL_TIM_SET_COUNTER(&htim3, 0);
  while (__HAL_TIM_GET_COUNTER(&htim3) < ms * 1000);
}*/
//This was the new delay function that worked better for our code.
void delay_ms(uint32_t ms) {
  volatile int i, j;
  for (i = 0; i < ms; i++) {
     for (j = 0; j < 1000; j++) {
        // Do nothing, just waste time.
     }
  }
}
```

5.1.4 - mainUSERButton.c

This code is very similar to mian.c from 5.1.2, with just the USER button functionality added in.

```
//
// This file is part of the GNU ARM Eclipse distribution.
// Copyright (c) 2014 Liviu Ionescu.
//
// ----------------------------------------------------------------------
// School: University of Victoria, Canada.
```

```
// Course: ECE 355 "Microprocessor-Based Systems".
// This is template code for Part 2 of Introductory Lab.
//
// See "system/include/cmsis/stm32f051x8.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f051x8.c" for handler declarations.
// ----------------------------------------------------------------------------
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
#define myTIM2_PRESCALAR ((uint16_t)0x0000)
#define myTIME2_PERIOD ((uint32_t)0xFFFFFFFF)
#define ADC_MAX_VALUE 4095  // 12-bit ADC max value
#define VREF 3.3            // Reference voltage for ADC
#define POTENTIOMETER_MAX_RESISTANCE 5000  // Assume a 5k potentiometer
// ----------------------------------------------------------------------------
//
// STM32F0 empty sample (trace via $(trace)).
//
// Trace support is enabled by adding the TRACE macro definition.
// By default the trace messages are forwarded to the $(trace) output,
// but can be rerouted to any device or completely suppressed, by
// changing the definitions required in system/src/diag/trace_impl.c
// (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG/_STDOUT).
//
// ----- main() ---------------------------------------------------------------
// Sample pragmas to cope with warnings. Please note the related line at
// the end of this function, used to pop the compiler diagnostics status.
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"
/* Definitions of registers and their bits are
  given in system/include/cmsis/stm32f051x8.h */
/* Clock prescaler for TIM2 timer: no prescaling */
#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for overflow */
#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)
void myGPIOA_Init(void);
void myTIM2_Init(void);
void myEXTI_Init(void);
// Declare/initialize your global variables here...
// NOTE: You'll need at least one global variable
// (say, timerTriggered = 0 or 1) to indicate
// whether TIM2 has started counting or not.
uint8_t timerTriggered = 0;
uint32_t lastCount = 0;
volatile uint32_t overFlowCount = 0;
uint8_t measure555Timer = 0; // 0 for function generator, 1 for 555 timer
/*** Call this function to boost the STM32F0xx clock to 48 MHz ***/
```

```
void SystemClock48MHz( void )
{
//
// Disable the PLL
//
  RCC->CR &= ~(RCC_CR_PLLON);
//
// Wait for the PLL to unlock
//
  while (( RCC->CR & RCC_CR_PLLRDY ) != 0 );
//
// Configure the PLL for 48-MHz system clock
//
  RCC->CFGR = 0x00280000;
//
// Enable the PLL
//
  RCC->CR |= RCC_CR_PLLON;
//
// Wait for the PLL to lock
//
  while (( RCC->CR & RCC_CR_PLLRDY ) != RCC_CR_PLLRDY );
//
// Switch the processor to the PLL clock source
//
  RCC->CFGR = ( RCC->CFGR & (~RCC_CFGR_SW_Msk)) | RCC_CFGR_SW_PLL;
//
// Update the system with the new clock frequency
//
  SystemCoreClockUpdate();
}
/*****************************************************************/
  uint16_t adc_value;
  float voltage, resistance;
int
main(int argc, char* argv[])
{
        SystemClock48MHz();
        trace_printf("This is Part 2 of Introductory Lab...\n");
        trace_printf("System clock: %u Hz\n", SystemCoreClock);
        myGPIOA_Init();         /* Initialize I/O port PA */
        myTIM2_Init();          /* Initialize timer TIM2 */
        myEXTI_Init();          /* Initialize EXTI */
        while (1)
        {
    // Start ADC conversion
    ADC1->CR |= ADC_CR_ADSTART;
    // Wait for ADC conversion to complete
    while (!(ADC1->ISR & ADC_ISR_EOC)) {}
```

```
        // Read ADC value
        adc_value = ADC1->DR;
        // Calculate the potentiometer voltage
        voltage = (adc_value * VREF) / ADC_MAX_VALUE;
        // Calculate potentiometer resistance based on voltage
        resistance = (voltage / VREF) * POTENTIOMETER_MAX_RESISTANCE;
        // Output the potentiometer voltage to the DAC
        uint32_t dac_value = (uint32_t)((voltage / VREF) * 4095);
        DAC->DHR12R1 = dac_value;
        //trace_printf("ADC Value: %u, Voltage: %f V, Resistance: %f Ohms\n", adc_value, voltage, resistance);
        //trace_printf("DAC Output Voltage: %f V (from ADC value)\n", (dac_value * VREF) / 4095);
            }
            return 0;
}
void myGPIOA_Init()
{
            //initialize general IO pins
            /* Enable clock for GPIOA peripheral */
            // Relevant register: RCC->AHBENR
            RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
            /* Configure PA2 as input */
            // Relevant register: GPIOA->MODER
//          GPIOA->MODER &= ~(GPIO_MODER_MODER2);
            GPIOA->MODER &= ~(GPIO_MODER_MODER2 | GPIO_MODER_MODER1 |
GPIO_MODER_MODER0);
            /* Ensure no pull-up/pull-down for PA2 */
            // Relevant register: GPIOA->PUPDR
//          GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR2);
            GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR2 | GPIO_PUPDR_PUPDR1 | GPIO_PUPDR_PUPDR0);
}
void myTIM2_Init()
{
            //initializing timer
            /* Enable clock for TIM2 peripheral */
            // Relevant register: RCC->APB1ENR
            RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
            /* Configure TIM2: buffer auto-reload, count up, stop on overflow,
             * enable update events, interrupt on overflow only */
            // Relevant register: TIM2->CR1
            TIM2->CR1 |= TIM_CR1_URS;
            /* Set clock prescaler value */
            TIM2->PSC = myTIM2_PRESCALER;
            /* Set auto-reloaded delay */
            TIM2->ARR = myTIM2_PERIOD;
            /* Update timer registers */
            // Relevant register: TIM2->EGR
            TIM2->EGR |= TIM_EGR_UG;
            /* Assign TIM2 interrupt priority = 0 in NVIC */
            // Relevant register: NVIC->IP[3], or use NVIC_SetPriority
```

```
	NVIC_SetPriority(TIM2_IRQn, 0);
	/* Enable TIM2 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(TIM2_IRQn);
	/* Enable update interrupt generation */
	// Relevant register: TIM2->DIER
	TIM2->DIER |= TIM_DIER_UIE;
}
void myEXTI_Init()
{
	//initializing different interrupt lines
	/* Map EXTI2 line to PA2 */
	// Relevant register: SYSCFG->EXTICR[0]
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
	SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PA | SYSCFG_EXTICR1_EXTI1_PA |
SYSCFG_EXTICR1_EXTI0_PA;
	/* EXTI2 line interrupts: set rising-edge trigger */
	// Relevant register: EXTI->RTSR
	EXTI->RTSR |= EXTI_RTSR_TR2 | EXTI_RTSR_TR1 | EXTI_RTSR_TR0;
	/* Unmask interrupts from EXTI2 line */
	// Relevant register: EXTI->IMR
	EXTI->IMR |= EXTI_IMR_MR2 | EXTI_IMR_MR1 | EXTI_IMR_MR0;
	//
	//setting priority to button and signal generator
	NVIC_SetPriority(EXTI0_1_IRQn, 2);
	//
	/* Assign EXTI2 interrupt priority = 0 in NVIC */
	// Relevant register: NVIC->IP[2], or use NVIC_SetPriority
	NVIC_SetPriority(EXTI2_3_IRQn, 1);
	/* Enable EXTI2 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(EXTI0_1_IRQn);
	NVIC_EnableIRQ(EXTI2_3_IRQn);
}
/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void TIM2_IRQHandler()
{
	//timer interrupt handler overflow
	/* Check if update interrupt flag is indeed set */
	if ((TIM2->SR & TIM_SR_UIF) != 0)
	{
		trace_printf("\n*** Overflow! ***\n");
		/* Clear update interrupt flag */
		// Relevant register: TIM2->SR
		TIM2->SR &= ~TIM_SR_UIF;
		/* Restart stopped timer */
		// Relevant register: TIM2->CR1
		TIM2->CR1 |= TIM_CR1_CEN;
	}
```

```
}
void EXTI0_1_IRQHandler()
{
        //button interrupt function
  if ((EXTI->PR & EXTI_PR_PR0) != 0)
  {
    measure555Timer ^= 1; // Toggle between measuring function generator and 555 timer
    trace_printf("Switched measurement to: %s\n", measure555Timer ? "555 timer" : "Function generator");
    EXTI->PR |= EXTI_PR_PR0;
  }
}
/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void EXTI2_3_IRQHandler()
{       //interrupt for the incoming signal either pin 1 or pin 2
        /* Check if EXTI2 interrupt pending flag is indeed set */
        if ((EXTI->PR & EXTI_PR_PR2) != 0 && measure555Timer == 0)
        {
                if (timerTriggered == 0){
                        TIM2->CNT = 0;
                        TIM2->CR1 |= TIM_CR1_CEN;
                        timerTriggered = 1;
                } else{
                        TIM2->CR1 &= ~TIM_CR1_CEN;
                        float currentCount = TIM2->CNT;
                        uint32_t totalTicks = currentCount - lastCount;
                        if(totalTicks == 0){
                                trace_printf("Invalid measurement. Total ticks: 0\n");
                        } else{
                                float frequency = SystemCoreClock / currentCount;
                                trace_printf("Period: %u ticks, Frequency %f Hz\n", totalTicks, frequency);
                        }
                        timerTriggered = 0;
                }
                EXTI->PR |= EXTI_PR_PR2;
        }else if(measure555Timer == 1){
                //switched to 555timer mode
                if (timerTriggered == 0){
                                        //TIM2->CNT = 0;
                                        //TIM2->CR1 |= TIM_CR1_CEN;
                                        timerTriggered = 1;
                                } else{
                                        TIM2->CR1 &= ~TIM_CR1_CEN;
                                        float currentCount = TIM2->CNT;
                                        uint32_t totalTicks = currentCount - lastCount;
                                        if(totalTicks == 0){
                                                trace_printf("Invalid measurement. Total ticks: 0\n");
                                        } else{
                                                float frequency = SystemCoreClock / currentCount;
```

```
                                                trace_printf("555 Value: %d", (DAC->DHR12R1 * VREF) /
4095);
                                            }
                                            timerTriggered = 0;
                                        }
                                    EXTI->PR |= EXTI_PR_PR1;
            }
}
#pragma GCC diagnostic pop
// ----------------------------------------------------------------------------
```

5.1.5 - mainAll.c

This was our attempt at getting all the individual parts to work together. We were unable to complete this code but wanted to include it for reference if needed.

```
#include <stdio.h>
#include "diag/Trace.h"
#include <string.h>
#include "cmsis/cmsis_device.h"
#include "stm32f0xx.h"  // This includes the RCC register definitions
//#include "stm32f0xx_gpio.h"
#include "stm32f0xx_hal.h"
#include "stm32f0xx_hal_spi.h"
#include "stm32f0xx_hal_gpio.h"
#include "stm32f0xx_hal_rcc.h"
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"
#define myTIM2_PRESCALAR ((uint16_t)0x0000)
#define myTIME2_PERIOD ((uint32_t)0xFFFFFFFF)
#define ADC_MAX_VALUE 4095  // 12-bit ADC max value
#define VREF 3.3            // Reference voltage for ADC
#define POTENTIOMETER_MAX_RESISTANCE 5000  // Assume a 5k potentiometer
// Global variables
unsigned int Freq = 0;  // Measured frequency value
unsigned int Res = 0;   // Measured resistance value
// SPI Handle
SPI_HandleTypeDef SPI_Handle;
// OLED Initialization Commands
unsigned char oled_init_cmds[] = {
  0xAE, 0x20, 0x00, 0x40, 0xA0 | 0x01, 0xA8, 0x40 - 1, 0xC0 | 0x08, 0xD3, 0x00,
  0xDA, 0x32, 0xD5, 0x80, 0xD9, 0x22, 0xDB, 0x30, 0x81, 0xFF, 0xA4, 0xA6,
  0xAD, 0x30, 0x8D, 0x10, 0xAE | 0x01, 0xC0, 0xA0
};
// ASCII Character Data for Display
```

```
unsigned char Characters[][8] = {     {0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000,0b00000000, 0b00000000, 0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
                {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
```

```
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // SPACE
	{0b00000000, 0b00000000, 0b01011111, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // !
	{0b00000000, 0b00000111, 0b00000000, 0b00000111, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // "
	{0b00010100, 0b01111111, 0b00010100, 0b01111111, 0b00010100,0b00000000, 0b00000000,
0b00000000},  // #
	{0b00100100, 0b00101010, 0b01111111, 0b00101010, 0b00010010,0b00000000, 0b00000000,
0b00000000},  // $
	{0b00100011, 0b00010011, 0b00001000, 0b01100100, 0b01100010,0b00000000, 0b00000000,
0b00000000},  // %
	{0b00110110, 0b01001001, 0b01010101, 0b00100010, 0b01010000,0b00000000, 0b00000000,
0b00000000},  // &
	{0b00000000, 0b00000101, 0b00000011, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // '
	{0b00000000, 0b00011100, 0b00100010, 0b01000001, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // (
	{0b00000000, 0b01000001, 0b00100010, 0b00011100, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // )
	{0b00010100, 0b00001000, 0b00111110, 0b00001000, 0b00010100,0b00000000, 0b00000000,
0b00000000},  // *
	{0b00001000, 0b00001000, 0b00111110, 0b00001000, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // +
	{0b00000000, 0b01010000, 0b00110000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // ,
	{0b00001000, 0b00001000, 0b00001000, 0b00001000, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // -
	{0b00000000, 0b01100000, 0b01100000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // .
	{0b00100000, 0b00010000, 0b00001000, 0b00000100, 0b00000010,0b00000000, 0b00000000,
0b00000000},  // /
```

```
        {0b00111110, 0b01010001, 0b01001001, 0b01000101, 0b00111110,0b00000000, 0b00000000,
0b00000000},  // 0
        {0b00000000, 0b01000010, 0b01111111, 0b01000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // 1
        {0b01000010, 0b01100001, 0b01010001, 0b01001001, 0b01000110,0b00000000, 0b00000000,
0b00000000},  // 2
        {0b00100001, 0b01000001, 0b01000101, 0b01001011, 0b00110001,0b00000000, 0b00000000,
0b00000000},  // 3
        {0b00011000, 0b00010100, 0b00010010, 0b01111111, 0b00010000,0b00000000, 0b00000000,
0b00000000},  // 4
        {0b00100111, 0b01000101, 0b01000101, 0b01000101, 0b00111001,0b00000000, 0b00000000,
0b00000000},  // 5
        {0b00111100, 0b01001010, 0b01001001, 0b01001001, 0b00110000,0b00000000, 0b00000000,
0b00000000},  // 6
        {0b00000011, 0b00000001, 0b01110001, 0b00001001, 0b00000111,0b00000000, 0b00000000,
0b00000000},  // 7
        {0b00110110, 0b01001001, 0b01001001, 0b01001001, 0b00110110,0b00000000, 0b00000000,
0b00000000},  // 8
        {0b00000110, 0b01001001, 0b01001001, 0b00101001, 0b00011110,0b00000000, 0b00000000,
0b00000000},  // 9
        {0b00000000, 0b00110110, 0b00110110, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // :
        {0b00000000, 0b01010110, 0b00110110, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // ;
        {0b00001000, 0b00010100, 0b00100010, 0b01000001, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // <
        {0b00010100, 0b00010100, 0b00010100, 0b00010100, 0b00010100,0b00000000, 0b00000000,
0b00000000},  // =
        {0b00000000, 0b01000001, 0b00100010, 0b00010100, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // >
        {0b00000010, 0b00000001, 0b01010001, 0b00001001, 0b00000110,0b00000000, 0b00000000,
0b00000000},  // ?
        {0b00110010, 0b01001001, 0b01111001, 0b01000001, 0b00111110,0b00000000, 0b00000000,
0b00000000},  // @
        {0b01111110, 0b00010001, 0b00010001, 0b00010001, 0b01111110,0b00000000, 0b00000000,
0b00000000},  // A
        {0b01111111, 0b01001001, 0b01001001, 0b01001001, 0b00110110,0b00000000, 0b00000000,
0b00000000},  // B
        {0b00111110, 0b01000001, 0b01000001, 0b01000001, 0b00100010,0b00000000, 0b00000000,
0b00000000},  // C
        {0b01111111, 0b01000001, 0b01000001, 0b00100010, 0b00011100,0b00000000, 0b00000000,
0b00000000},  // D
        {0b01111111, 0b01001001, 0b01001001, 0b01001001, 0b01000001,0b00000000, 0b00000000,
0b00000000},  // E
        {0b01111111, 0b00001001, 0b00001001, 0b00001001, 0b00000001,0b00000000, 0b00000000,
0b00000000},  // F
        {0b00111110, 0b01000001, 0b01001001, 0b01001001, 0b01111010,0b00000000, 0b00000000,
0b00000000},  // G
```

```
        {0b01111111, 0b00001000, 0b00001000, 0b00001000, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // H
        {0b01000000, 0b01000001, 0b01111111, 0b01000001, 0b01000000,0b00000000, 0b00000000,
0b00000000},  // I
        {0b00100000, 0b01000000, 0b01000001, 0b00111111, 0b00000001,0b00000000, 0b00000000,
0b00000000},  // J
        {0b01111111, 0b00001000, 0b00010100, 0b00100010, 0b01000001,0b00000000, 0b00000000,
0b00000000},  // K
        {0b01111111, 0b01000000, 0b01000000, 0b01000000, 0b01000000,0b00000000, 0b00000000,
0b00000000},  // L
        {0b01111111, 0b00000010, 0b00001100, 0b00000010, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // M
        {0b01111111, 0b00000100, 0b00001000, 0b00010000, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // N
        {0b00111110, 0b01000001, 0b01000001, 0b01000001, 0b00111110,0b00000000, 0b00000000,
0b00000000},  // O
        {0b01111111, 0b00001001, 0b00001001, 0b00001001, 0b00000110,0b00000000, 0b00000000,
0b00000000},  // P
        {0b00111110, 0b01000001, 0b01010001, 0b00100001, 0b01011110,0b00000000, 0b00000000,
0b00000000},  // Q
        {0b01111111, 0b00001001, 0b00011001, 0b00101001, 0b01000110,0b00000000, 0b00000000,
0b00000000},  // R
        {0b01000110, 0b01001001, 0b01001001, 0b01001001, 0b00110001,0b00000000, 0b00000000,
0b00000000},  // S
        {0b00000001, 0b00000001, 0b01111111, 0b00000001, 0b00000001,0b00000000, 0b00000000,
0b00000000},  // T
        {0b00111111, 0b01000000, 0b01000000, 0b01000000, 0b00111111,0b00000000, 0b00000000,
0b00000000},  // U
        {0b00011111, 0b00100000, 0b01000000, 0b00100000, 0b00011111,0b00000000, 0b00000000,
0b00000000},  // V
        {0b00111111, 0b01000000, 0b00111000, 0b01000000, 0b00111111,0b00000000, 0b00000000,
0b00000000},  // W
        {0b01100011, 0b00010100, 0b00001000, 0b00010100, 0b01100011,0b00000000, 0b00000000,
0b00000000},  // X
        {0b00000111, 0b00001000, 0b01110000, 0b00001000, 0b00000111,0b00000000, 0b00000000,
0b00000000},  // Y
        {0b01100001, 0b01010001, 0b01001001, 0b01000101, 0b01000011,0b00000000, 0b00000000,
0b00000000},  // Z
        {0b01111111, 0b01000001, 0b00000000, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // [
        {0b00010101, 0b00010110, 0b01111100, 0b00010110, 0b00010101,0b00000000, 0b00000000,
0b00000000},  // back slash
        {0b00000000, 0b00000000, 0b00000000, 0b01000001, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // ]
        {0b00000100, 0b00000010, 0b00000001, 0b00000010, 0b00000100,0b00000000, 0b00000000,
0b00000000},  // ^
        {0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b01000000,0b00000000, 0b00000000,
0b00000000},  // _
```

```
        {0b00000000, 0b00000001, 0b00000010, 0b00000100, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // `
        {0b00100000, 0b01010100, 0b01010100, 0b01010100, 0b01111000,0b00000000, 0b00000000,
0b00000000},  // a
        {0b01111111, 0b01001000, 0b01000100, 0b01000100, 0b00111000,0b00000000, 0b00000000,
0b00000000},  // b
        {0b00111000, 0b01000100, 0b01000100, 0b01000100, 0b00100000,0b00000000, 0b00000000,
0b00000000},  // c
        {0b00111000, 0b01000100, 0b01000100, 0b01001000, 0b01111111,0b00000000, 0b00000000,
0b00000000},  // d
        {0b00111000, 0b01010100, 0b01010100, 0b01010100, 0b00011000,0b00000000, 0b00000000,
0b00000000},  // e
        {0b00001000, 0b01111110, 0b00001001, 0b00000001, 0b00000010,0b00000000, 0b00000000,
0b00000000},  // f
        {0b00001100, 0b01010010, 0b01010010, 0b01010010, 0b00111110,0b00000000, 0b00000000,
0b00000000},  // g
        {0b01111111, 0b00001000, 0b00000100, 0b00000100, 0b01111000,0b00000000, 0b00000000,
0b00000000},  // h
        {0b00000000, 0b01000100, 0b01111101, 0b01000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // i
        {0b00100000, 0b01000000, 0b01000100, 0b00111101, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // j
        {0b01111111, 0b00010000, 0b00101000, 0b01000100, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // k
        {0b00000000, 0b01000001, 0b01111111, 0b01000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // l
        {0b01111100, 0b00000100, 0b00011000, 0b00000100, 0b01111000,0b00000000, 0b00000000,
0b00000000},  // m
        {0b01111100, 0b00001000, 0b00000100, 0b00000100, 0b01111000,0b00000000, 0b00000000,
0b00000000},  // n
        {0b00111000, 0b01000100, 0b01000100, 0b01000100, 0b00111000,0b00000000, 0b00000000,
0b00000000},  // o
        {0b01111100, 0b00010100, 0b00010100, 0b00010100, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // p
        {0b00001000, 0b00010100, 0b00010100, 0b00011000, 0b01111100,0b00000000, 0b00000000,
0b00000000},  // q
        {0b01111100, 0b00001000, 0b00000100, 0b00000100, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // r
        {0b01001000, 0b01010100, 0b01010100, 0b01010100, 0b00100000,0b00000000, 0b00000000,
0b00000000},  // s
        {0b00000100, 0b00111111, 0b01000100, 0b01000000, 0b00100000,0b00000000, 0b00000000,
0b00000000},  // t
        {0b00111100, 0b01000000, 0b01000000, 0b00100000, 0b01111100,0b00000000, 0b00000000,
0b00000000},  // u
        {0b00011100, 0b00100000, 0b01000000, 0b00100000, 0b00011100,0b00000000, 0b00000000,
0b00000000},  // v
        {0b00111100, 0b01000000, 0b00111000, 0b01000000, 0b00111100,0b00000000, 0b00000000,
0b00000000},  // w
```

```
        {0b01000100, 0b00101000, 0b00010000, 0b00101000, 0b01000100,0b00000000, 0b00000000,
0b00000000},  // x
        {0b00001100, 0b01010000, 0b01010000, 0b01010000, 0b00111100,0b00000000, 0b00000000,
0b00000000},  // y
        {0b01000100, 0b01100100, 0b01010100, 0b01001100, 0b01000100,0b00000000, 0b00000000,
0b00000000},  // z
        {0b00000000, 0b00001000, 0b00110110, 0b01000001, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // {
        {0b00000000, 0b00000000, 0b01111111, 0b00000000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // |
        {0b00000000, 0b01000001, 0b00110110, 0b00001000, 0b00000000,0b00000000, 0b00000000,
0b00000000},  // }
        {0b00001000, 0b00001000, 0b00101010, 0b00011100, 0b00001000,0b00000000, 0b00000000,
0b00000000},  // ~
        {0b00001000, 0b00011100, 0b00101010, 0b00001000, 0b00001000,0b00000000, 0b00000000,
0b00000000}   // <-
};
/* Clock prescaler for TIM2 timer: no prescaling */
#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for overflow */
#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)
// Function Prototypes
void oled_Write(unsigned char Value);
void oled_Write_Cmd(unsigned char cmd);
void oled_Write_Data(unsigned char data);
void oled_config(void);
void refresh_OLED(void);
void delay_ms(uint32_t ms);
void SystemClock48MHz(void);
void myGPIOA_Init(void);
void myTIM2_Init(void);
void myEXTI_Init(void);
void myADC_Init(void);
void myDAC_Init(void);
uint8_t timerTriggered = 0;
uint32_t lastCount = 0;
volatile uint32_t overFlowCount = 0;
uint8_t measure555Timer = 0; // 0 for function generator, 1 for 555 timer
uint16_t adc_value;
float voltage, resistance;
void SystemClock48MHz(void)
{
  RCC->CR &= ~(RCC_CR_PLLON);
  while ((RCC->CR & RCC_CR_PLLRDY) != 0);
  RCC->CFGR = 0x00280000;
  RCC->CR |= RCC_CR_PLLON;
  while ((RCC->CR & RCC_CR_PLLRDY) != RCC_CR_PLLRDY);
  RCC->CFGR = (RCC->CFGR & (~RCC_CFGR_SW_Msk)) | RCC_CFGR_SW_PLL;
  SystemCoreClockUpdate();
```

```
}
int main(int argc, char* argv[])
{
  SystemClock48MHz();
  oled_config();
  myGPIOA_Init();                    /* Initialize I/O port PA */
  myTIM2_Init();            /* Initialize timer TIM2 */
  myEXTI_Init();            /* Initialize EXTI */
  myADC_Init();          // Initialize ADC for potentiometer readings
  myDAC_Init();          // Initialize DAC to control the optocoupler
  refresh_OLED();
  while (1)
  {
        refresh_OLED();
    // Start ADC conversion
    ADC1->CR |= ADC_CR_ADSTART;
    // Wait for ADC conversion to complete
    while (!(ADC1->ISR & ADC_ISR_EOC)) {}
    // Read ADC value
    adc_value = ADC1->DR;
    // Calculate the potentiometer voltage
    voltage = (adc_value * VREF) / ADC_MAX_VALUE;
    // Calculate potentiometer resistance based on voltage
    resistance = (voltage / VREF) * POTENTIOMETER_MAX_RESISTANCE;
    // Output the potentiometer voltage to the DAC
    uint32_t dac_value = (uint32_t)((voltage / VREF) * 4095);
    DAC->DHR12R1 = dac_value;
    trace_printf("ADC Value: %u, Voltage: %f V, Resistance: %f Ohms\n", adc_value, voltage, resistance);
    trace_printf("DAC Output Voltage: %f V (from ADC value)\n", (dac_value * VREF) / 4095);
    Res = resistance;
  }
}
void oled_config(void)
{
  // Enable GPIOB and SPI1 clocks
  RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
  RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
  // Configure PB3 (SCK) and PB5 (MOSI) as AF0
  GPIOB->MODER &= ~(GPIO_MODER_MODER3 | GPIO_MODER_MODER5);
  GPIOB->MODER |= (GPIO_MODER_MODER3_1 | GPIO_MODER_MODER5_1);
  GPIOB->AFR[0] &= ~((0xF << (3 * 4)) | (0xF << (5 * 4)));
  // Configure PB6 (CS#), PB7 (D/C#), PB4 (RES#) as outputs
  GPIOB->MODER &= ~(GPIO_MODER_MODER4 | GPIO_MODER_MODER6 | GPIO_MODER_MODER7);
  GPIOB->MODER |= (GPIO_MODER_MODER4_0 | GPIO_MODER_MODER6_0 |
GPIO_MODER_MODER7_0);
  // Reset OLED Display
  GPIOB->ODR &= ~GPIO_ODR_4;
  //HAL_Delay(10);
  delay_ms(10);
```

```
  GPIOB->ODR |= GPIO_ODR_4;
  delay_ms(10);
  //HAL_Delay(10);
  // SPI Configuration
  SPI_Handle.Instance = SPI1;
  SPI_Handle.Init.Direction = SPI_DIRECTION_1LINE;
  SPI_Handle.Init.Mode = SPI_MODE_MASTER;
  SPI_Handle.Init.DataSize = SPI_DATASIZE_8BIT;
  SPI_Handle.Init.CLKPolarity = SPI_POLARITY_LOW;
  SPI_Handle.Init.CLKPhase = SPI_PHASE_1EDGE;
  SPI_Handle.Init.NSS = SPI_NSS_SOFT;
  SPI_Handle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
  SPI_Handle.Init.FirstBit = SPI_FIRSTBIT_MSB;
  SPI_Handle.Init.CRCPolynomial = 7;
  HAL_SPI_Init(&SPI_Handle);
  __HAL_SPI_ENABLE(&SPI_Handle);
  // Send Initialization Commands to OLED
  for (unsigned int i = 0; i < sizeof(oled_init_cmds); i++) {
    oled_Write_Cmd(oled_init_cmds[i]);
  }
  // Clear Display
  for (int page = 0; page < 8; page++) {
    oled_Write_Cmd(0xB0 | page);
    oled_Write_Cmd(0x00);
    oled_Write_Cmd(0x10);
    for (int seg = 0; seg < 128; seg++) {
      oled_Write_Data(0x00);
    }
  }
}
void refresh_OLED(void)
{
  unsigned char Buffer[17];
  // Display Resistance
  snprintf(Buffer, sizeof(Buffer), "Res: %5u Ohms", Res);
  oled_Write_Cmd(0xB2); // Page 2
  oled_Write_Cmd(0x03);
  oled_Write_Cmd(0x10);
  for (unsigned int i = 0; i < strlen(Buffer); i++) {
    unsigned char c = Buffer[i];
    for (int j = 0; j < 8; j++) {
      oled_Write_Data(Characters[c][j]);
    }
  }
  // Display Frequency
  snprintf((char *)Buffer, sizeof(Buffer), "Frq: %5u Hz", Freq);
  oled_Write_Cmd(0xB3); // Page 3
  oled_Write_Cmd(0x03);
  oled_Write_Cmd(0x10);
```

```
  for (unsigned int i = 0; i < strlen(Buffer); i++) {
    unsigned char c = Buffer[i];
    for (int j = 0; j < 8; j++) {
      oled_Write_Data(Characters[c][j]);
    }
  }
  delay_ms(100);
}
void oled_Write(unsigned char Value)
{
  while (!(SPI1->SR & SPI_SR_TXE));
  HAL_SPI_Transmit(&SPI_Handle, &Value, 1, HAL_MAX_DELAY);
  while (SPI1->SR & SPI_SR_BSY);
}
void oled_Write_Cmd(unsigned char cmd)
{
  GPIOB->ODR |= GPIO_ODR_6;
  GPIOB->ODR &= ~GPIO_ODR_7;
  GPIOB->ODR &= ~GPIO_ODR_6;
  oled_Write(cmd);
  GPIOB->ODR |= GPIO_ODR_6;
}
void oled_Write_Data(unsigned char data)
{
  GPIOB->ODR |= GPIO_ODR_6;
  GPIOB->ODR |= GPIO_ODR_7;
  GPIOB->ODR &= ~GPIO_ODR_6;
  oled_Write(data);
  GPIOB->ODR |= GPIO_ODR_6;
}
/*void delay_ms(uint32_t ms)
{
  __HAL_TIM_SET_COUNTER(&htim3, 0);
  while (__HAL_TIM_GET_COUNTER(&htim3) < ms * 1000);
}*/
void delay_ms(uint32_t ms) {
  volatile unsigned int i, j;
  for (i = 0; i < ms; i++) {
    for (j = 0; j < 1000; j++) {
      // Do nothing, just waste time
    }
  }
}
void myADC_Init(void)
{
  RCC->APB2ENR |= RCC_APB2ENR_ADCEN;          // Enable ADC clock
  ADC1->CFGR1 &= ~ADC_CFGR1_RES;              // Set ADC resolution to 12-bit
  ADC1->CHSELR = ADC_CHSELR_CHSEL5;           // Select channel 5 (assume potentiometer connected to
PA5)
```

```
  ADC1->CFGR1 &= ~ADC_CFGR1_ALIGN;          // Right-align the result
  ADC1->CR |= ADC_CR_ADEN;                  // Enable the ADC
  while (!(ADC1->ISR & ADC_ISR_ADRDY)) {}   // Wait for ADC to be ready
}
void myDAC_Init(void)
{
  RCC->APB1ENR |= RCC_APB1ENR_DACEN;        // Enable DAC clock
  DAC->CR |= DAC_CR_EN1;                    // Enable DAC channel 1
}
void myGPIOA_Init()
{
        /* Enable clock for GPIOA peripheral */
        // Relevant register: RCC->AHBENR
        RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
        /* Configure PA2 as input */
        // Relevant register: GPIOA->MODER
        /* Ensure no pull-up/pull-down for PA2 */
        // Relevant register: GPIOA->PUPDR
        GPIOA->MODER &= ~(GPIO_MODER_MODER2 | GPIO_MODER_MODER1 |
GPIO_MODER_MODER0);
        GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR2 | GPIO_PUPDR_PUPDR1 | GPIO_PUPDR_PUPDR0);
}
void myTIM2_Init()
{
        /* Enable clock for TIM2 peripheral */
        // Relevant register: RCC->APB1ENR
        RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
        /* Configure TIM2: buffer auto-reload, count up, stop on overflow,
         * enable update events, interrupt on overflow only */
        // Relevant register: TIM2->CR1
        TIM2->CR1 |= TIM_CR1_URS;
        /* Set clock prescaler value */
        TIM2->PSC = myTIM2_PRESCALER;
        /* Set auto-reloaded delay */
        TIM2->ARR = myTIM2_PERIOD;
        /* Update timer registers */
        // Relevant register: TIM2->EGR
        TIM2->EGR |= TIM_EGR_UG;
        /* Assign TIM2 interrupt priority = 0 in NVIC */
        // Relevant register: NVIC->IP[3], or use NVIC_SetPriority
        NVIC_SetPriority(TIM2_IRQn, 3);
        /* Enable TIM2 interrupts in NVIC */
        // Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
        NVIC_EnableIRQ(TIM2_IRQn);
        /* Enable update interrupt generation */
        // Relevant register: TIM2->DIER
        TIM2->DIER |= TIM_DIER_UIE;
}
void myEXTI_Init()
```

```
{
	/* Map EXTI2 line to PA2 */
	// Relevant register: SYSCFG->EXTICR[0]
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
	SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PA | SYSCFG_EXTICR1_EXTI1_PA |
SYSCFG_EXTICR1_EXTI0_PA;
	/* EXTI2 line interrupts: set rising-edge trigger */
	// Relevant register: EXTI->RTSR
	EXTI->RTSR |= EXTI_RTSR_TR2 | EXTI_RTSR_TR1 | EXTI_RTSR_TR0;
	/* Unmask interrupts from EXTI2 line */
	// Relevant register: EXTI->IMR
	EXTI->IMR |= EXTI_IMR_MR2 | EXTI_IMR_MR1 | EXTI_IMR_MR0;
//
	NVIC_SetPriority(EXTI0_1_IRQn, 2);
//
	/* Assign EXTI2 interrupt priority = 0 in NVIC */
	// Relevant register: NVIC->IP[2], or use NVIC_SetPriority
	NVIC_SetPriority(EXTI2_3_IRQn, 1);
	/* Enable EXTI2 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(EXTI0_1_IRQn);
	NVIC_EnableIRQ(EXTI2_3_IRQn);
}
/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void TIM2_IRQHandler()
{
	/* Check if update interrupt flag is indeed set */
	if ((TIM2->SR & TIM_SR_UIF) != 0)
	{
		trace_printf("\n*** Overflow! ***\n");
		/* Clear update interrupt flag */
		// Relevant register: TIM2->SR
		TIM2->SR &= ~TIM_SR_UIF;
		/* Restart stopped timer */
		// Relevant register: TIM2->CR1
		TIM2->CR1 |= TIM_CR1_CEN;
	}
}
void EXTI0_1_IRQHandler()
{
  if ((EXTI->PR & EXTI_PR_PR0) != 0)
  {
    measure555Timer ^= 1; // Toggle between measuring function generator and 555 timer
    trace_printf("Switched measurement to: %s\n", measure555Timer ? "555 timer" : "Function generator");
    EXTI->PR |= EXTI_PR_PR0;
  }
}
/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void EXTI2_3_IRQHandler()
```

```
{
	refresh_OLED();
	/* Check if EXTI2 interrupt pending flag is indeed set */
	if ((EXTI->PR & EXTI_PR_PR2) != 0 && measure555Timer == 0)
	{
		if (timerTriggered == 0){
			TIM2->CNT = 0;
			TIM2->CR1 |= TIM_CR1_CEN;
			timerTriggered = 1;
		} else{
			TIM2->CR1 &= ~TIM_CR1_CEN;
			float currentCount = TIM2->CNT;
			uint32_t totalTicks = currentCount - lastCount;
			if(totalTicks == 0){
				trace_printf("Invalid measurement. Total ticks: 0\n");
			} else{
				float frequency = SystemCoreClock / currentCount;
				trace_printf("Period: %u ticks, Frequency %f Hz\n",
totalTicks, frequency);
			}
			timerTriggered = 0;
		}
		EXTI->PR |= EXTI_PR_PR2;
	}
	else if (measure555Timer == 1)
	{
		if (timerTriggered == 0){
			timerTriggered = 1;
		} else{
			TIM2->CR1 &= ~TIM_CR1_CEN;
			float currentCount = TIM2->CNT;
			uint32_t totalTicks = currentCount - lastCount;
			if(totalTicks == 0){
				trace_printf("Invalid measurement. Total
ticks: 0\n");
			} else{
				float frequency = SystemCoreClock /
currentCount;
				trace_printf("Period: %u ticks, Frequency
%f Hz\n", totalTicks, frequency);
			}
			timerTriggered = 0;
		}
	   EXTI->PR |= EXTI_PR_PR1;
	}
}
#pragma GCC diagnostic pop
```

6 References

[1] "ECE 355 Laboratory Website," University of Victoria. [Online]. Available: https://www.ece.uvic.ca/~ece355/lab/index.html. [Accessed: Nov. 23, 2024].

[2] "ECE 355: Microprocessor-Based Systems Laboratory Manual," University of Victoria, 2023. [Online]. Available: https://www.ece.uvic.ca/~ece355/lab. [Accessed: Nov. 23, 2024].

[3] D. Rakhmatov, *Interface Examples*, University of Victoria, Fall 2024. [Online]. Available: https://www.ece.uvic.ca/~daler/courses/ece355/. [Accessed: Nov. 23, 2024].

[4] D. Rakhmatov, "ECE 355 Course Website," University of Victoria. [Online]. Available: https://www.ece.uvic.ca/~daler/courses/ece355/. [Accessed: Nov. 23, 2024].

[5] S. Systech, *SSD1306: 128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller*, Apr. 2008. [Online]. Available: http://www.solomon-systech.com. [Accessed: Nov. 23, 2024].