Department of Computer and Electrical Engineering
University of Victoria

- Problem Description/Specification (5) ______________
- Design/Solution (15) ______________
- Testing/Results (10) ______________
- Discussion (15) ______________
- Code Design and Documentation (15) ______________
- **Total** **(60)** ______________

MICROPROCESSOR-BASED SYSTEMS LABORATORY REPORT

Ansh Shukla - V00816280
Jonathan Parkes - V00826631
Title: PWM Signal Generation and Monitoring Systems
Reports Submitted: November 29, 2017
To: Priyani Vanaparthi
Lab Section: B05

Table of Contents

Problem Description

Objective

To create a working system using a digital signal produced by the STM32F0 Discovery Board that allows for a pulse-width-modulated signal produced by the NE555 timer to be controlled and monitored. The STM32F0 Discovery Board allowed for the frequency to be manipulated and then the value was printed to the LCD. The microcontroller's voltage over the potentiometer was used to determine the resistance value that was then printed to the LCD. Figure 1 shows a circuit diagram of all these parts working together.

Specifications

NE555 timer IC

An integrated circuit that can be used for various applications such as a timer, pulse generator, or oscillator.

4N35 optocoupler IC

An integrated circuit where an LED drives a phototransistor that sends the collector current to the emitter. This is used to convert the inputted AC signal sent as light and then received as a DC signal.

STM32F0 Discovery Board[4]

A microcontroller board powered by an ARM Cortex M0 processor, 8K SRAM, 64K Flash, 5 DMA channels, two general-purpose analog comparators, and 55 general-purpose I/O (GPIO) pins.

PBMCUSLK Board[5]

An expansion board connected to the STM32F0 Discovery board that has a built in potentiometer, buttons, LCD screen, and its own power supply.

Tools

- Oscilloscope
- Multimeter
- Debugger

Design Solution

The design specifications were for the system to detect the frequency of a square wave signal from 0.0V to 3.3V of the external timer (**NE555** IC). The microcontroller on the STMF0 Discovery board measured the voltage across a potentiometer (POT) on the PBMCUSLK board and relayed it to the external optocoupler. The external optocoupler (4N35 IC) was used to control the frequency of the PWM signal from the microcontroller to be readable by the timer. The measured frequency and the corresponding POT resistance was then displayed on the LCD on the PBMCUSLK board.

Reusing and tweaking the code written from Part 2 of the introductory lab, we measured the signal frequency from the 555 timer. Based on the potentiometer voltage read by the Analog-to-Digital Converter (ADC) on the STMF0 board, the signal frequency was adjusted by the 4N35 optocoupler, with its input supplied by the Digital-to-Analog Converter (DAC). Lastly, the SPI was used to communicate with the LCD. Using a polling approach, the analog voltage signal coming from the potentiometer on the PBMCUSLK board will be measured continuously by the ADC. The potentiometer resistance value was then calculated using the determined voltage measurements and the rating of the potentiometer used as seen in equation 1. The digital value obtained from the ADC was then utilized to adjust the frequency of the PWM signal generated by the NE555 timer, where the DAC was then used to convert the said digital value to an analog voltage signal driving the 4N35 optocoupler.

When displaying the signal frequency and the potentiometer resistance, the properly configured SPI was used to drive the LCD on the PBMCUSLK board. The LCD is a 4-bit, 2-by-8 character display, with no direct write access to the LCD pins. The 8-bit 74HC595 shift register on the PBMCUSLK board is used to controlling the LCD pins. The 74HC595 shift register receives 8-bit words from the SPI via the serial MOSI port, which was appropriately timed using the latch clock LCK and the serial shift register clock SCK, and then controls LCD with data bits D3-D0, register-select bit RS, and enable bit EN [6]. The inclusion of the SPI library was also utilized to easily work with the LCD.

The LCD was chosen to run with 2 display lines of characters, however the same effect could have been achieved with one line and a shifting by initializing the display differently. Finally, in order to update the display the old data was cleared by clearing the LCD and then the new data was sent to the screen. This process was then repeated for each new data set.

Square Wave Frequency Measurement

To measure the frequency of the square wave, it is necessary to detect either the rising or falling edge of the waveform. To do his, the edge port was configured to detect the rising edge of the square wave signal applied to EXTI0_1_IRQHandler(). To determine the frequency of the 555 timer's square wave, it is necessary to measure the time between two consecutive rising edges; in other words, the measurement of period between of the rising edges. The square wave frequency of the 555 timer was calculated using the period.

ADC, DAC, and Resistance Values Measurement

The ADC's data register holds information that needs to be converted to be understood. At max resistance (5K ohms) from the potentiometer, the ADC value in the data register is 4095. Therefore to measure the resistance of a 5kΩ potentiometer, the following equation was used:

(Equation 1)

$$POT\ Resistance = 5000 * \frac{(4095 - ADC\_value)}{4095}$$

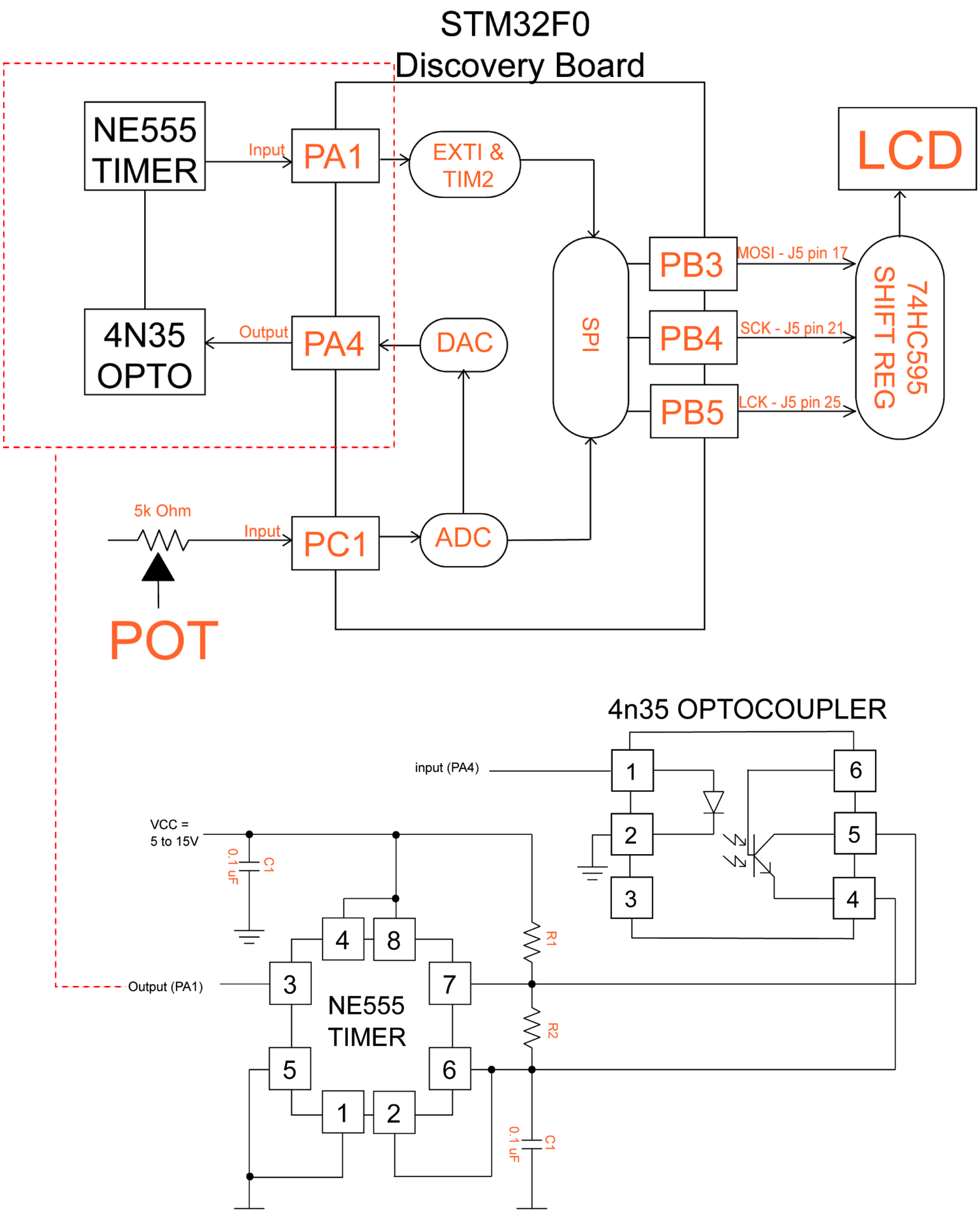
LCD Display

The HD44780U LCD attached to the PBMCUSLK board is used to display the resistance and frequency values calculated. Since direct access to the LCD's data pins are not used, the MOSI, SCK, and LCK connections from the PBMCUSLK board are used. The included SPI library's functions were used to communicate with the LCD. [5]

Since the LCD was set to be in 4-bit words, we had to break the 8-bit data/commands for the LCD into 2 high and low 4-bit words and sent it to the shift register (through the SPI) in a pulse. The high word is sent three times, with the LCD disabled, LCD enables, and LCD disabled again. This allows the data to be registered by the LCD. We configure the display to have two lines, where we have characters that update and stay the same. The LCD displays information, is updated, cleared, and then shows the updated information. A delay is introduced after the clear because the LCD needs time to catch up to the commands / data being sent to it. This delay was chosen via trial and error.

Diagrams

Figure 1: Overall connections of the system [1]



Test Procedure and Results

ADC

After implementing the ADC, testing it is relatively easy. In the main loop of the program, enable the conversion via the control register and then extract the ADC value from the ADC data register. Then it's just a simple matter of printing the value out onto the console to read. From here, it can be observed that the value is changing as the potentiometer is adjusted. At max resistance (5K ohms), the ADC value is 4095. The current potentiometer resistance can be derived with this value (see equation 1).

DAC

Testing the DAC is a very simple matter, assuming it has been implemented correctly. To test the DAC, update the value of its data holding register in the loop from the ADC, and then retrieve that value from the register and print it to the console to confirm that it has registered and converted.

Circuit

Testing the circuit and making sure the 555 Timer and Optocoupler have been connected properly is slightly more difficult. Using an oscilloscope connected to pin 3 of the NE555 Timer and ground, a wave can be obtained. A measurement of the square wave's frequency can be taken using the oscilloscope. Adjusting the resistance and capacitance values results in a different frequency.

GPIO

Using the code from lab part 2, we could test GPIO pins for input using the function generator. LEDs, on the other hand, can be used to test output. Making sure that the pin modes can be adjusted (Input/Output/AF) and that a signal is being transferred was detected by turning an LED on/off in the main while loop.

SPI and LCD

We sent data and commands (e.g. cursor movement, display a message) manually to make sure they worked. Testing to split a stream of data into higher and lower nibbles, we manually sent an 8-bit word and printed the results out onto the console. A delay was added when SPI was sending a byte to the LCD so the data was processed before sending the next. This is important in order to ensure there are no overlaps.

System Limitations

- Voltage can not be more than 5V
- Accuracy of the potentiometer
- Minimum frequency of 1.9hz
- Maximum frequency of 548Khz
- The time it takes to clear the display and send the updated data to it.

Explanation and Discussion

In the lab our design was based off of the supplied materials in the lecture slides, the programming manual, and our lab 1 part 2 solutions. Prescalers allow for controlled frequencies for the clock, allowing for more control over timing, shift registers can be used to essentially create more inputs or outputs using less pins for MCU and an optocoupler can be used to control when the circuit gets a signal. Unfortunately, unsuccessful debugging halted progress in our lab pass the DAC. The information seemed to not be going through to the LCD and it would not turn on. There could be several reasons for this, one of which could be incorrect pin configurations. This might be a cause because there seemed to be interference between the GPIO pins and the SPI configuration. When the SPI was used to send information to the LCD, the MCU stopped reading the ADC value. Another trouble faced was figuring out the resistance of the resistors used in the circuit, this caused a problem because with too high of a resistance value the signal becomes 0 making the timer not receive enough power to run the NE555 chip. Shortcomings in this project came down to the limited frequency that could be produced based on the potentiometer value.

This lab offered an opportunity to become familiar with the ADC and DAC systems at work along with microcontrollers and provided a greater understanding of optocouplers. Furthermore, interfacing with several peripherals and other external devices were investigated and practiced. If we were to do the project again we would set aside more time at the start in order to research the components to fully understand them before attempting to use them.

References

[1] "Interface Examples", 2016. [Online].
Available: http://www.ece.uvic.ca/~daler/courses/ceng355/interfacex.pdf.
Accessed: November 25, 2017.

[2] STMicroelectronics, "Programming manual," in STM32F0DISCOVERY MCU, 2012. [Online].
Available:
http://www.ece.uvic.ca/~ceng355/lab/supplement/STM32F0xxxProgrammingManual_DM00051352.pdf
Accessed: November 25, 2017.

[3] STMicroelectronics, "Reference manual," in STM32F0DISCOVERY MCU, 2012. [Online].
Available: http://www.ece.uvic.ca/~ceng355/lab/supplement/stm32f0RefManual.pdf
Accessed: November 25, 2017.

[4] STMicroelectronics, "STM32F0DISCOVERY MCU," in STM32F0DISCOVERY MCU, 2012. [Online].
Available:
http://www.ece.uvic.ca/~ceng355/lab/supplement/stm32f0_schematic_MB1034.pdf
Accessed: November 27, 2017.

[5] STMicroelectronics, "Prototyping Board with Microcontroller Interface," in STM32F0DISCOVERY MCU, 2012. [Online].
Available: http://www.ece.uvic.ca/~ceng355/lab/supplement/PBMCUSLK_UG.pdf
Accessed: November 27, 2017.

[6] Wikipedia, "Hitachi HD44780 LCD controller," in Wikipedia. [Online].
Available: https://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller
Accessed: November 28, 2017.

[7] HITACHI, "HD44780,". [Online].
Available: http://www.ece.uvic.ca/~ceng355/lab/supplement/HD44780.pdf.
Accessed: November 28, 2017.

Appendices

Source Code

```
//
// This file is part of the GNU ARM Eclipse distribution.
// Copyright (c) 2014 Liviu Ionescu.
//
// ----------------------------------------------------------------------------
// School: University of Victoria, Canada.
// Course: CENG 355 "Microprocessor-Based Systems".
//
// See "system/include/cmsis/stm32f0xx.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f0xx.c" for handler declarations.
// ----------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
#include "stm32f0xx.h"
// ----------------------------------------------------------------------------
//
// 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"
/* Clock prescaler for TIM2 timer: no prescaling */
#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for overflow */
#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)
#define NUM_CHAR ((uint8_t)0x08) //Number of characters for LCD
#define NUM_DIGITS ((uint8_t)0x04) //Number of digits on LCD
#define LCD_DELAY (uint32_t)(6000000) //Delay for the LCD to refresh
// Your global variables...
unsigned int risingEdge = 0;
static volatile uint16_t TIM_2_Overflow = 0;
static volatile uint16_t ADC_Value = 0;
static volatile uint16_t DAC_Value = 0;
static uint8_t LCD_Line0[NUM_CHAR] = {};
static uint8_t LCD_Line1[NUM_CHAR] = {};
uint8_t freqVal[NUM_DIGITS] = {};
uint8_t resVal[NUM_DIGITS] = {};
void myGPIOA_Init(void);
void myTIM2_Init(void);
void myEXTI_Init(void);
void myADC_Init(void);
void myDAC_Init(void);
void mySPI_Init(void);
void myLCD_Init(void);

int main(int argc, char* argv[]) {
	trace_printf("CENG 355 Final Project\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 */
	myDAC_Init();		/* Initialize DAC */
	mySPI_Init();	/* Initialize SPI */
	myLCD_Init();/* Initialize LCD */

	while (1) {
		//ADC to DAC
		ADC_DAC();
		//calculate resistance
		calcResistance();
    //Update the LCD
		LCD_Write();
	}
	return 0;
}

void myGPIOA_Init(void) {
	/* Enable clock for GPIOA peripheral */
	// Relevant register: RCC->AHBENR
	RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
	/* Configure PA1 as input */
	// Relevant register: GPIOA->MODER
	GPIOA->MODER &= ~(GPIO_MODER_MODER1);
	/* Ensure no pull-up/pull-down for PA1 */
	// Relevant register: GPIOA->PUPDR
	GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR1);

}

void myTIM2_Init(void) {
	/* 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 = ((uint16_t) 0x008C);

        /* 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 = ((uint16_t) 0x0001);

        /* 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 EXTI1 line to PA1 */
        // Relevant register: SYSCFG->EXTICR[0]
```

```
	SYSCFG->EXTICR[0] &= ~(SYSCFG_EXTICR1_EXTI1);

	/* EXTI1 line interrupts: set rising-edge trigger */
	// Relevant register: EXTI->RTSR
	EXTI->RTSR |= EXTI_RTSR_TR1;

	/* Unmask interrupts from EXTI1 line */
	// Relevant register:
	EXTI->IMR |= EXTI_IMR_MR1;

	/* Assign EXTI1 interrupt priority = 0 in NVIC */
	// Relevant register: NVIC->IP[1], or use NVIC_SetPriority
	NVIC_SetPriority(EXTI0_1_IRQn, 0);

	/* Enable EXTI1 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(EXTI0_1_IRQn);
}

void myADC_Init(void) {
	//Enable peripheral clock
	RCC->APB2ENR |= RCC_APB2ENR_ADCEN;

	//Configure PC1 to be analog input for ADC channel
	GPIOC->MODER |= (GPIO_MODER_MODER1);
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPDR1);

	//Configuring ADC1
	ADC1->CFGR1 &= ~(ADC_CFGR1_RES);
	ADC1->CFGR1 |= ADC_CFGR1_CONT;
	ADC1->CFGR1 &= ~(ADC_CFGR1_EXTEN);
	ADC1->CFGR1 &= ~(ADC_CFGR1_ALIGN);
	ADC1->CFGR1 &= ~(ADC_CFGR1_SCANDIR);
```

```
        ADC1->CFGR1 |= ADC_CFGR1_OVRMOD;

        //Convert the ADC1 Channel 0 with 239.5 Cycles as sampling time
        ADC1->CHSELR |= ADC_CHSELR_CHSEL0;
        ADC1->SMPR |= ADC_SMPR_SMP;

        //Enable ADC peripheral
        ADC1->CR |= ADC_CR_ADEN;
        //Wait for ADC interrupt ready
        while (ADC1->ISR & ADC_IER_ADRDYIE);
        //ADC start conversion
        ADC1->CR |= ADC_CR_ADSTART;
}

void myDAC_Init(void) {
        //Enable DAC peripheral clock
        RCC->APB1ENR |= RCC_APB1ENR_DACEN;
        //Set PA4 as analog output
        GPIOA->MODER |= (GPIO_MODER_MODER4);
        GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR4);
        //Enable DAC channel1 Trigger. conversion starts once DAC_DHR8R1 register loaded
        DAC->CR &= ~(DAC_CR_TEN1);
        //Enable DAC channel1
        DAC->CR |= DAC_CR_EN1;
}

void mySPI_Init(void) {
        SPI_InitTypeDef     SPI_InitStructure;
        //Enable SPI peripheral clock
        RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
        //SPI1 settings from lecture
        SPI_InitStructure.SPI_Direction = SPI_Direction_1Line_Tx;
        SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
```

```
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;

SPI_Init(SPI1, &SPI_InitStructure);

//Enable SPI1
SPI1->CR1 |= SPI_CR1_SPE;

//Configure pins required for SPI peripheral
//Enable clock for GPIOB peripheral
RCC->AHBENR |= RCC_AHBENR_GPIOBEN;

//PB3 = SCK
GPIOB->MODER |= GPIO_MODER_MODER3_1;

//Set AFRy[3:0] to AF0 for PB3
GPIOB->AFR[0] |= GPIO_AFRL_AFR0;

//PB4 = LCK
GPIOB->MODER |= GPIO_MODER_MODER4_0;

//Ensure no pull-up/pull-down for PB4
GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPDR4);

//PB5 = MOSI
GPIOB->MODER |= GPIO_MODER_MODER5_1;

//Select alternate function for PB5
```

```
        GPIOB->AFR[0] |= GPIO_AFRL_AFR0;

} /* end void mySPI_Init(void) */

void LCD_SendByte(uint8_t data)
{
        GPIOB->ODR &= ~GPIO_ODR_4; //force lck signal to 0

        //Wait for SPI1 to be not busy
        while((SPI1->SR & 0x80) != 0);

        //Send data
        SPI_SendData8(SPI1, data);

        //Wait until SPI1 is not busy
        while((SPI1->SR & 0x80) != 0);

        GPIOB->ODR |= GPIO_ODR_4; //force lck signal to 1

        //wait for LCD to finish instruction
        uint32_t i = 0;
        while(i < (100000)) i++;

}

void LCD_prepareData(void) {
        LCD_Line0[0] = 0x46; //F
        LCD_Line0[1] = 0x3A; //:
        for(uint8_t i = 0; i < 4; i++) {
                LCD_Line0[i+2] = freqVal[i];
        }
        LCD_Line0[6] = 0x48; //H
        LCD_Line0[7] = 0x7A; //z
```

```
	LCD_Line1[0] = 0x52; //R
	LCD_Line1[1] = 0x3A; //:
	for(uint8_t i = 2; i < 6; i++) {
		LCD_Line1[i] = resVal[i - 2];
	}
	LCD_Line1[6] = 0x4F; //O
	LCD_Line1[7] = 0x68; //h
}

void LCD_SendCmd(uint8_t command) {
	uint HIGH = (command & 0xF0) >> 4;
	uint LOW  = command & 0x0F;

	LCD_SendByte(0x00 + HIGH); //disable and high bit
	LCD_SendByte(0x80 + HIGH); //enable and high bit
	LCD_SendByte(0x00 + HIGH); //disable and high bit
	LCD_SendByte(0x00 + LOW);  //disable and low bit
	LCD_SendByte(0x80 + LOW);  //enable and low bit
	LCD_SendByte(0x00 + LOW);  //disable and high bit
}

void LCD_SendData(uint8_t data) {
	uint HIGH = (data & 0xF0) >> 4;
	uint LOW  = data & 0x0F;

	LCD_SendByte(0x40 + HIGH); //Send 10xxHIGH
	LCD_SendByte(0xC0 + HIGH); //Send 11xxHIGH
	LCD_SendByte(0x40 + HIGH); //Send 10xxHIGH
	LCD_SendByte(0x40 + LOW); //Send 10xxLOW
	LCD_SendByte(0xC0 + LOW); //Send 11xxLOW
	LCD_SendByte(0x40 + LOW); //Send 10xxLOW
}
```

```
void LCD_Write(void) {
	//Configure data values
   LCD_prepareData();
	for(uint8_t i = 0; i < NUM_CHAR; i++) LCD_SendData(*(&LCD_Line0[0] + i));
	//Move the cursor to second line
	LCD_SendCmd(0xC0);
	for(uint8_t i = 0; i < NUM_CHAR; i++) LCD_SendData(*(&LCD_Line1[0] + i));

	//Timer for LCD refresh
	uint32_t i = 0;
	while(i < 500000) i++; //5seconds

	//Clear display
	LCD_SendCmd(0x01);
}

void myLCD_Init(void) {
	//Configure LCD to 4-bit mode
	LCD_SendByte(0x02);
	LCD_SendByte(0x82);
	LCD_SendByte(0x02);

	LCD_SendCmd(0x28); //Configure data length, lines, font
	LCD_SendCmd(0x0C); //Configure display, cursor, blink
	LCD_SendCmd(0x06); //Configure cursor direction, shift
	LCD_SendCmd(0x01); //Clear display
}

void Decimal_to_Hex(uint16_t digit, uint8_t* hexArray) {
	//Digits 0-9 in HEX
	static uint8_t hex[] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};
	//break digit into its seperate spots.
```

```
		*hexArray = hex[digit / 1000];
		*(hexArray + 1*sizeof(uint8_t)) = hex[(digit - (thou)*(1000)) / 100];
		*(hexArray + 2*sizeof(uint8_t)) = hex[(digit - (thou)*(1000) - (hundreds)*(100)) / 10];
		*(hexArray + 3*sizeof(uint8_t)) = hex[(digit - (thou)*(1000) - (hundreds)*(100) -
(tens)*(10))];
}

void calcResistance(void) {
	float resistance = 5000 * ((double)(4095 - ADC_Value) / 4095);
	Decimal_to_Hex((uint16_t)resistance, &resVal[0]);
}

void ADC_DAC(void) {
	//if adc triggered
	if((ADC1->ISR & ADC_ISR_EOC)) {
		//get ADC value from the ADC's data register
		ADC_Value = (ADC1->DR);
		//Send value into converter
		DAC->DHR12R1 = ADC_Value;
		//Get DAC value from converter
		DAC_Value = DAC->DOR1;
	}
}

void TIM2_IRQHandler() {
	/* Check if update interrupt flag is indeed set */
	if ((TIM2->SR & TIM_SR_UIF) != 0) {
		trace_printf("\n*** Overflow! ***\n");

		//increment timer 2 overflow counter
		TIM_2_Overflow++;

		/* 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() {
        double frequency = 0;
        double period = 0;
        double count = 0;

        //Mask interrupt
        EXTI->IMR &= ~(EXTI_IMR_MR1);

        //if EXTI1 interrupt pending flag is set
        if ((EXTI->PR & EXTI_PR_PR1)) {
                //if rising edge
                if(risingEdge == 0) {
                        risingEdge = 1;
                        count = 0;
                        //clear timer
                        TIM2->CNT = ((uint32_t) 0x00000000);

                        //Start the timer
                        TIM2->CR1 |= TIM_CR1_CEN;
                }
                else {
                        //disable the timer
                        TIM2->CR1 &= ~(TIM_CR1_CEN);
                        //count = timer value
```

```
                count = TIM2->CNT;
                //If the timer overflowed
                if (TIM_2_Overflow == 0) {
                    period = ((double)count / (double)SystemCoreClock);
                    frequency = (1 / period);
                } else { //Else
                    //Reset timer overflow
                    TIM_2_Overflow = 0;

                    period = ((double)(TIM_2_Overflow * myTIM2_PERIOD + count) /
(double)SystemCoreClock);
                    frequency = (1 / period);
                }
                //frequency to HEX for the LCD
                Decimal_to_Hex((uint)frequency, &freqVal[0]);
                risingEdge = 0;
            }

            EXTI->PR |= (EXTI_PR_PR1);
        }

        //Unmask interrupts
        EXTI->IMR |= (EXTI_IMR_MR1);
}

#pragma GCC diagnostic pop

// ----------------------------------------------------------------------------
```