Pulse-Width-Modulation Generation and Monitorization

ECE 355 Lab Report

Jasper Hale V00888481
Slater Gordon V00910595
Lab section: B10

Table of contents

Problem description

Using a STM32F0 Discovery microcontroller (microcontroller) and a PBMCUSLK project board (project board) [1] a pulse width modulated (PWM) signal must be generated by the external timer (NE555) [2]. The signal must be controlled and monitored using an embedded system.

The objective of this project is to generate and monitor a pulse width modulated (PWM) signal using a STM32F0 Discovery microcontroller (microcontroller) and a PBMCUSLK project board (project board). On the project board is a potentiometer (POT) that sends a signal, ranging from 0 to 5000 Ohms, to the analog-to-digital converter (ADC) input on the microcontroller. Using the converted digital signal from the ADC, resistance is calculated and sent to the SPI and then the signal is converted back to analog through the digital-to-analog converter (DAC). The output from the DAC is used to drive the 4N35 optocoupler (4N35) [3], which then adjusts the NE555 frequency [2]. The square-wave from the NE555 is sent back to the microcontroller where its frequency is determined. To display frequency and resistance the serial peripheral interface (SPI) must send the values through to be displayed on the 4-bit LCD screen [4].

The lab manual specifies that CMSIS-defined software library functions may only be used for SPI access.

Design solution

Our design solution was done following the specifications within the lab manual [5] and using notes the class website[6][7]. We decided not to add any special features or techniques because we didn't want to complicate the project any more than it already is.

ADC

At the beginning of the project we first focused on obtaining accurate resistance values in the ADC output. To do this we set the POT to PC1 on the microcontroller and initialized the pin as an 'analog in' signal.

We also initialized the ADC by setting its sample rate, changing the selected channel to match the pin PC1 and calibrating. To start calibrating, we set the ADC_CR_ADCAL bit to 1 within the ADC control register and then waited for the bit be 0, which meant that the calibration is complete. Finally, the ADC_ADEN enable bit is set to 1, which means that the ADC is enabled and once the ADC_ISR_ADRDY flag is set to 0, the ADC is ready to accept conversion requests [1].

Once the ADC was ready and enabled, we sent conversion requests in a loop from main. Our design waits the end of conversion flag, ADC_ISR_EOC [1], to be set, by polling, then obtains the converted value from the ADC data register. The converted value can only be a maximum of 4095 since the data register holds a 12 bit value and 2^12 - 1 is 4095. Using this value, we calculate the resistance to be sent to the SPI.

DAC

The next step we took was to setup the DAC to convert values from the ADC. We first configured the DAC by setting its output to be PA4 and enabled control register [1]. Next, every time a value from the ADC needed to be converted back to a digital value, we cleared the DHR12R1 register to ensure there would be no errors when filling the register [1]. Finally, the program converted the value from the ADC by putting the data in the DHR12R1 register. 12R1 is 12 bit and right aligned data holding register, which matched the ADC data register value [1].

EXTI

During the introductory lab we setup a working function that would calculate the frequency of a square-wave from external interrupts. We implemented this code in our project and wired the NE555 to PA1[1], to integrate with our previous code. Once the square wave signal comes through PA1 it triggers an external interrupt where our function calculates the frequency.

LCD

Once we had values for frequency and resistance, we needed to get them displayed on the LCD. To display values on the LCD we initialized the SPI and then initialize the LCD in 4-bit mode by following the instructions within the LCD Reference Manual [4]. To initialize the SPI, we used the CMSIS-defined software library functions and followed the instructions from our class notes [7].

The LCD displays characters on 2 rows with 8 characters in each. It has 6 inputs that can be controlled by the user, these being data lines 4 to 7, the RS pin and the EN pin. The data lines are used to send through a command or data that is to be displayed. Commands are used to change the operation of the LCD and data is for displaying onto the LCD. The RS pin determines whether the data lines are a command or just data to be displayed. If RS is high, the data lines are just data and if it's low the data lines are a command. The EN pin is used to execute the data lines by sending it low and then high. Since we use the LCD in 4-bit mode and a character is 8-bits we must send the data through in 2 sets of 4 bits [4].

To send a command or data we first put the LCK low by setting the BSRR to 0 which cleared the output data register. Once the SPI status registers busy bit, SPI_SR_BSY, was cleared data was ready to be transferred via the MOSI line [1]. Data is transferred into the SPI's shift register input. When data is being transferred into the shift register input, the SCK goes low then high 8 times with each pulse shifting data into the shift register input. Then we wait for the busy bit to be clear again and put the LCK high, by setting the BSRR to 1, to release data to the LCD controller [1].

Lab Partition

Neither of us worked on the lab single-handedly. We met each time to work on the lab. We would study the manuals and explain concepts to each other to better our understanding of the lab system. Work was split evenly.

Diagrams

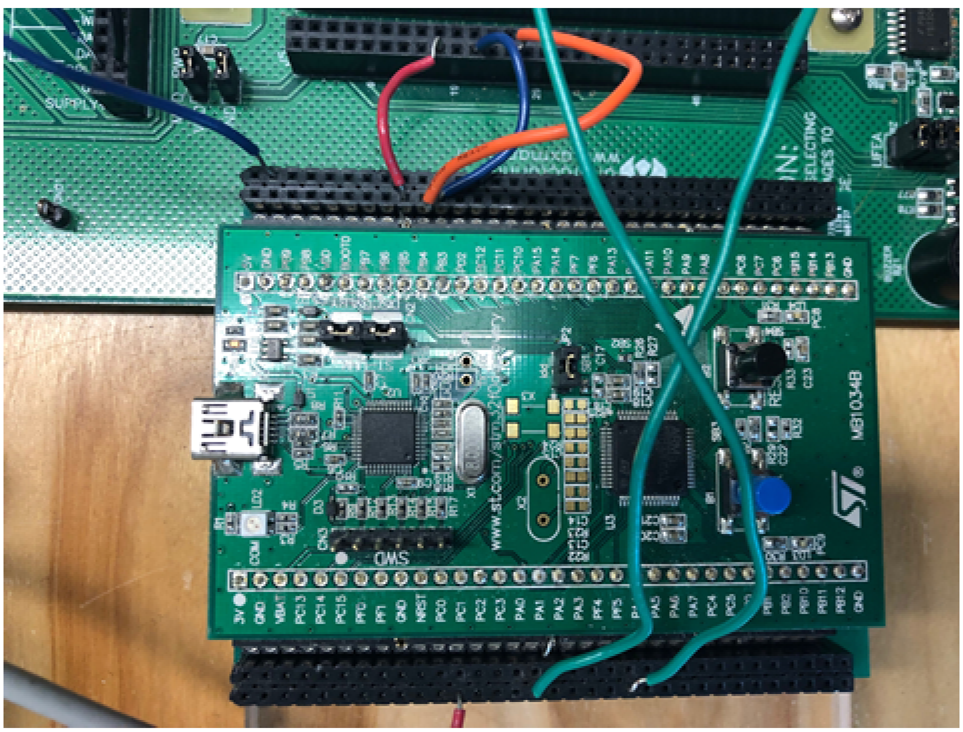

Figure 1: Microcontroller.

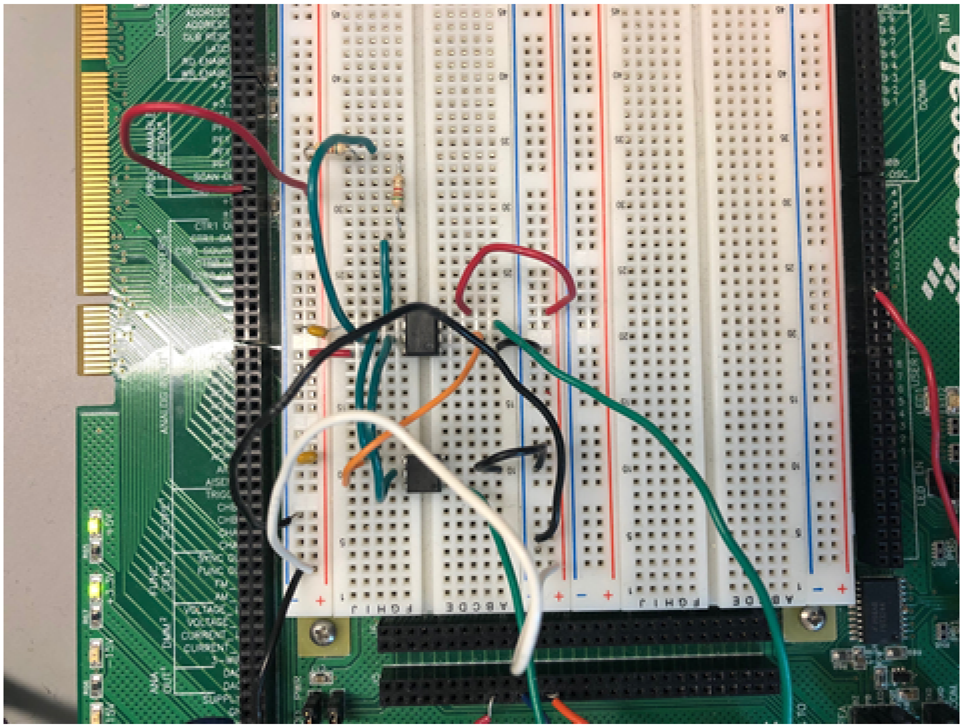

Figure 2: PBMCUSLK project board

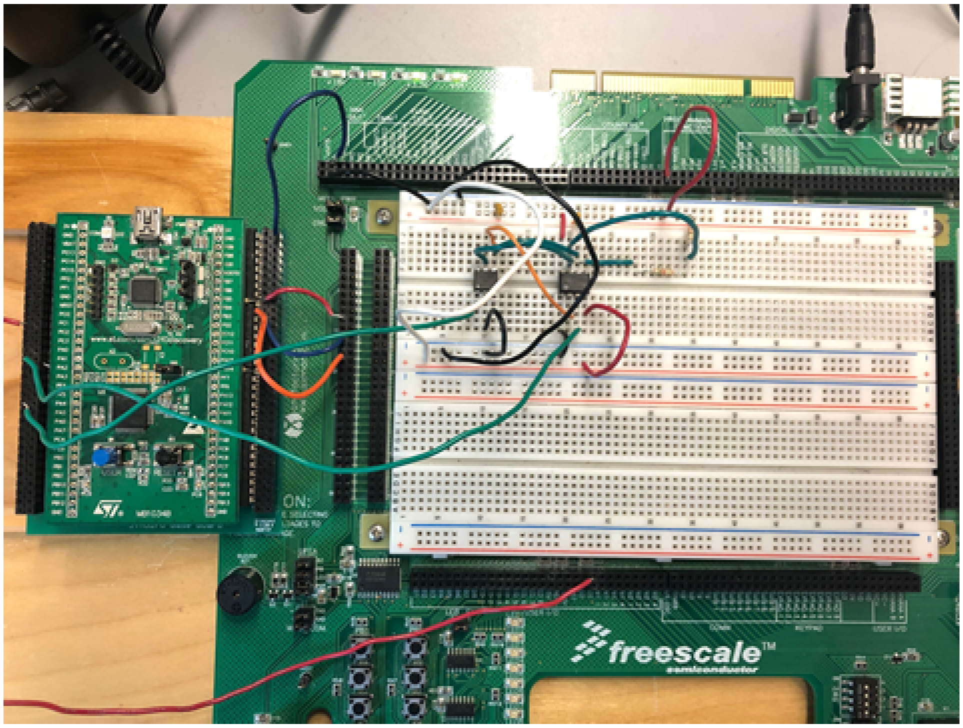

Figure 3: Mircocontroller and PBMCUSLK project board

Test Procedures and Results

Before displaying information on the LCD screen, the ADC, DAC, and SPI components were tested with the trace_printf() function to ensure correct values were being passed through the system. Starting with the ADC, values were displayed on the console to confirm that both the circuitry on the board was configured correctly and expected values were being displayed. Expected values would be maximum and minimum resistance values. Similarly, the DAC values were printed on the console ensuring information was flowing from the ADC and converted by the DAC correctly.

Commands sent to the LCD through the SPI were printed on the console to ensure that correct bit manipulations were occurring. The process of using the console to debug the software was extremely effective because a bug could be narrowed down to specific pieces of code or wiring on the board. If no information was being displayed, there was a high chance that a wire was not in the correct port.

The oscilloscope was used to measure the SCK edges to figure out the amount of time to wait for the LCD to update its display. If the LCD was receiving information faster than it could process, the display would not output the proper data. We created a wait function, that was just a loop decrementing a given amount of times, to delay while the LCD command finished executing. To make the wait function accurate we made a while loop in main that would put the LCK low, then call our wait function, then put the LCK high and monitor the LCK signal using the oscilloscope.

After getting the ADC set up to ensure our values were correct, we would move the POT to its max position, 5000 Ohms, print the resistance and then make sure that the printed value was 5000. We then tested the minimum value to make sure it matched. Lastly, we slowly move from the minimum to the maximum while printing values in our loop to check that the value goes to 5000 evenly.

Explanation and Discussion

For this project we strictly followed the design specifications and did not add any extra features. We chose not to add any features as we felt it was not necessary and did not want to over complicate the project.

There are many features we could have added and portions of the project that we could have done differently. Firstly, when we run the program for the second time after turning on the LCD it will not work because it sends commands to initialize the LCD into 4-bit mode when it is already in 4-bit mode. We should have added a feature that resets the LCD automatically when the program starts. Another part we could have changed is the way we send our number to the LCD. We simply hardcoded in each number we wanted to send. It would be much better to have a function that breaks up any number into an array and then sends each digit to the display. One mistake we did not notice until our demo was when we attempted to put the LCK low we used the BSRR instead of the BRR. The BSRR is used to set the LCK, but we wanted to use the BRR to clear the LCK [1].

One notable bug is printing to the console when trying to display to the LCD. The CPU will have to stop processing the information being fed into the LCD and assess console. This will cause values to spike rapidly because the CPU is handling something else while the LCD is still being fed information. The information will be irrelevant until the CPU starts to feed info back to the LCD.

Conclusion

The lab was completed successfully but was very challenging. The reference manuals are very dense and take a lot of time to study before a strong understanding is established. Proper time management will lead to successful completion of the lab.

References

[1] life.augmented. *STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM-based 32-bit MCUs Reference Manual.* Accessed: Nov. 15 2019. [Online]. Available:
https://www.ece.uvic.ca/~ece355/lab/supplement/stm32f0RefManual.pdf
[2] STMicroelectronics NV. *NE555 SA555 – SE555, General-purpose single bipolar timers.* Accessed: Nov. 15, 2019. [Online]. Available:
https://www.ece.uvic.ca/~ece355/lab/supplement/555timer.pdf
[3] Vishay Intertechnology, Inc. *Optocoupler, Phototransistor Output, with Base Connection.* Accessed: Nov.15, 2019. [Online]. Available:
https://www.ece.uvic.ca/~ece355/lab/supplement/4n35.pdf
[4] Hitachi, Ltd. Semiconductor & Integrated Circuits. *HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)*. Accessed: Nov. 15, 2019. [Online]. Available:
https://www.ece.uvic.ca/~ece355/lab/supplement/HD44780.pdf
[5] Jooya, K. Jones, D. Rakhmatov, and B. Sirna. *ECE 355: Microprocessor-Based Systems Laboratory Manual.* Accessed: Nov. 15, 2019. [Online]. Available:
https://www.ece.uvic.ca/~ece355/lab/ECE355-LabManual-2018.pdf
[6] D. Rakhmatov. Fall 2019. *I/O Examples* [Power Point slides]. Available:
https://www.ece.uvic.ca/~daler/courses/ece355/iox.pdf
[7] D. Rakhmatov. Fall 2019. *Interface Examples* [Power Point slides]. Available:
https://www.ece.uvic.ca/~daler/courses/ece355/interfacex.pdf

Appendices

```
//
// 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/stm32f0xx.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f0xx.c" for handler declarations.
// ----------------------------------------------------------------------------

#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
#include "stm32f0xx_spi.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"

#define SPI_Direction_1Line_Tx ((uint16_t)0xC000)
#define SPI_Mode_Master ((uint16_t)0x0104)
#define SPI_DataSize_8b ((uint16_t)0x0700)
#define SPI_CPOL_Low ((uint16_t)0x0000)
#define SPI_CPHA_1Edge ((uint16_t)0x0000)
#define SPI_NSS_Soft SPI_CR1_SSM
#define SPI_FirstBit_MSB ((uint16_t)0x0000)
```

```
#define SPI_CR1_SSM ((uint16_t)0x0200)

/* 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 myGPIO_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);

void wait(volatile long);
void showResistance(int);
void showFrequency(int);
unsigned int adcConvert(void);
void dacConvert(unsigned int);

void sendWord(char word, int command);
void HC595Write(char);

// Your global variables...
int resistance = 0;
int frequency = 0;
int period = 0;

int main(int argc, char* argv[])
{

    trace_printf("This is the final project...\n");
    trace_printf("System clock: %u Hz\n", SystemCoreClock);

    myGPIO_Init();      /* Initialize I/O port PA */
    myTIM2_Init();      /* Initialize timer TIM2 */
    myEXTI_Init();      /* Initialize EXTI */
    myADC_Init();
    myDAC_Init();
    mySPI_Init();
    myLCD_Init();

    while (1)
    {
```

```c
    {
        unsigned int dig = adcConvert();
        resistance = (int)dig*(5000/4095);
        dacConvert(dig);
        //trace_printf("F: %d\nR: %d\n", frequency, resistance);
        showResistance(resistance);
        showFrequency(frequency);
    }

    return 0;
}

unsigned int adcConvert()
{
    ADC1->CR |= ADC_CR_ADSTART; // starts ADC conversions

    while((ADC1->ISR & ADC_ISR_EOC) == 0); // need to wait until its done converting

    return ADC1->DR; // store data in the data register
}

void dacConvert(unsigned int dig)
{
    //input for dac
    DAC->DHR12R1 &= ~(DAC_DHR12R1_DACC1DHR);

    //put dig into input to be converted
    DAC->DHR12R1 |= dig;
}

void showFrequency(int f)
{
    /* Set display to first line */
    sendWord(0x80, 1);

    /* send 'F' */
    sendWord(0x46, 0);

    /* send ':' */
    sendWord(0x3A, 0);

    sendWord((f/1000)%10 +'0', 0);
    sendWord((f/100)%10+ '0', 0);
    sendWord((f/10)%10+'0', 0);
    sendWord((f)%10+'0', 0);

    /* send 'H' */
```

```c
    /* send 'H' */
    sendWord(0x48, 0);

    /* send 'z' */
    sendWord(0x7A, 0);

}

void showResistance(int r)
{
    //trace_printf("HERE");
    /* Set display to second line */
    sendWord(0xC0, 1);

    /* send 'R' */
    sendWord('R', 0);

    /* send ':' */
    sendWord(0x3A, 0);

    /* send resistance value */
    sendWord((r/1000)%10 +'0', 0);
    sendWord((r/100)%10+ '0', 0);
    sendWord((r/10)%10+'0', 0);
    sendWord((r)%10+'0', 0);


    /* send 'O' */
    sendWord(0x4F, 0);

    /* send 'h' */
    sendWord(0x68, 0);
}

void sendWord(char word, int command)
{
    /* Check if its a command */
    uint8_t RS = command ? 0x00 : 0x40;// RS and R/W. writing to registers means you're actually displaying text

    /* Declare EN */
    uint8_t EN = 0x80;

    /* Get high and low */
    char high = (word & 0xF0) >> 4;// keep the high bits. Can only send 4 at a time  bc in 4-bit mode.
    char low = word & 0x0F;
```

```
    //trace_printf("%d\n",low | RS | EN);

    HC595Write(high|RS);
    HC595Write(high |RS| EN);
    HC595Write(high |RS);

    HC595Write(low|RS);
    HC595Write(low |RS| EN);
    HC595Write(low|RS);
}

void HC595Write(char data)
{
    /* Force your LCK signal to 0 */
    GPIOB->BRR |= GPIO_BRR_BR_4;

    /* Wait until SPI1 is ready (TXE = 1 or BSY = 0) */
    while((SPI1->SR & SPI_SR_BSY) != 0);

    /* Assumption: your data holds 8 bits to be sent */
    SPI_SendData8(SPI1, data);

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

    /* Force your LCK signal to 1 */
    GPIOB->BSRR |= GPIO_BSRR_BS_4;
}

void wait(volatile long time)
{
    while(time >= 0) {time--;};
}

void myGPIO_Init()
{
    /* Enable clock for GPIO peripheral */
    // Relevant register: RCC->AHBENR
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN; //GPIOA
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN; //GPIOB

    /* Configure PA1 as input, reads 555 Timer edges*/
    // Relevant register: GPIOA->MODER
    GPIOA->MODER &= ~(GPIO_MODER_MODER1);
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR1);  //No Pull
```

```
    /* Configure DAC analog output to PA4 */
    GPIOA->MODER &= ~(GPIO_MODER_MODER4);
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR4);  //No pull

    GPIOB->MODER |= (GPIO_MODER_MODER3_1 | GPIO_MODER_MODER4_0 | GPIO_MODER_MODER5_1);
    GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPDR3 | GPIO_PUPDR_PUPDR4 | GPIO_PUPDR_PUPDR5);
}


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 = ((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 */
```

```
void myEXTI_Init()
{
    /* Map EXTI1 line to PA1 */
    // Relevant register: SYSCFG->EXTICR[0]
    SYSCFG->EXTICR[0] = SYSCFG_EXTICR1_EXTI1_PA;

    /* 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 |= 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()
{
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

    /* Enable clock for ADC */
    RCC->APB2ENR |= RCC_APB2ENR_ADCEN;


    /* Set pc1 to analog */
    GPIOC->MODER &= ~(GPIO_MODER_MODER1);
    GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPDR1);  //No Pull

    /* Set to continuous */
    ADC1->CFGR1 |= ADC_CFGR1_CONT | ADC_CFGR1_OVRMOD;
    ADC1->CFGR1 &= ~(ADC_CFGR1_ALIGN);

    /* Set channel to match pin selected */
    ADC1->CHSELR |= ADC_CHSELR_CHSEL11;

    /* Set sampling rate to 239.5 */
    ADC1->SMPR |= ADC_SMPR_SMP;

    //Start calibration and wait for completion
```

```
    //Start calibration and wait for completion
    ADC1->CR |= ADC_CR_ADCAL;
    while((ADC1->CR & ADC_CR_ADCAL) != 0);

    /* Enable ADC and wait for ready status */
    ADC1->CR |= ADC_CR_ADEN;
    while ((ADC1->ISR & ADC_ISR_ADRDY) == 0);
}

void myDAC_Init()
{
    /* Enable clock for DAC */
    RCC->APB1ENR |= RCC_APB1ENR_DACEN;


    /* Enable DAC channel 1*/
    DAC->CR |= DAC_CR_EN1;
}

void mySPI_Init()
{
    RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;

    SPI_InitTypeDef SPI_InitStructInfo;
    SPI_InitTypeDef* SPI_InitStruct = &SPI_InitStructInfo;

    SPI_InitStruct->SPI_Direction = SPI_Direction_1Line_Tx;
    SPI_InitStruct->SPI_Mode = SPI_Mode_Master;
    SPI_InitStruct->SPI_DataSize = SPI_DataSize_8b;
    SPI_InitStruct->SPI_CPOL = SPI_CPOL_Low;
    SPI_InitStruct->SPI_CPHA = SPI_CPHA_1Edge;
    SPI_InitStruct->SPI_NSS = SPI_NSS_Soft;
    SPI_InitStruct->SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256;
    SPI_InitStruct->SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_InitStruct->SPI_CRCPolynomial = 7;

    SPI_Init(SPI1, SPI_InitStruct);
    SPI_Cmd(SPI1, ENABLE);
}

void myLCD_Init()
{
    /* Set LCD to 4-bit mode */
    sendWord(0x2, 1);
    wait(8000);

```

```
    /* 2 lines of 8 characters */
    sendWord(0x28, 1); //DL=0, N=1, F=0
    wait(8000);

    /* Cursor not displayed/blinking */
    sendWord(0x0E, 1); //D=1, C=0, B=0
    wait(8000);

    /* Auto increment DDRAM */
    sendWord(0x06, 1);
    wait(8000);


    /* Clear display*/
    sendWord(0x01, 1);
    wait(8000);
}

void myLCK_Init() {
    GPIOB->MODER |= GPIO_MODER_MODER4_0;
    GPIOB->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR4;
    GPIOB->PUPDR &= ~(0xC0);
    GPIOB->BSRR |= GPIO_BSRR_BS_4;
}

/* This handler is declared in system/src/cmsis/vectors_stm32f0xx.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); //clear UIF

        /* Restart stopped timer */
        // Relevant register: TIM2->CR1
        TIM2->CR1 |= TIM_CR1_CEN; //restart TIM2
    }
}


/* This handler is declared in system/src/cmsis/vectors_stm32f0xx.c */
```

```
void EXTI0_1_IRQHandler()
{
    //TODO:disable interrupts
    /* Check if EXTI1 interrupt pending flag is indeed set */
    if ((EXTI->PR & EXTI_PR_PR1) != 0)
    {
        uint16_t isEnabled = (TIM2->CR1 & TIM_CR1_CEN); //TIM2 is enabled

        if(!isEnabled){
            TIM2->CNT = (uint32_t) 0x0; //clear count register
            TIM2->CR1 |= TIM_CR1_CEN; //start counting timer pulses
        } else {
            TIM2->CR1 &= ~(TIM_CR1_CEN); //disable counter
            frequency = SystemCoreClock/TIM2->CNT; //calculate frequency
            period = 1/frequency; //calculate period
        }
        //TODO:enable interrupts
        EXTI->PR |= EXTI_PR_PR1; //clear EXTI1 interrupt pending flag
    }
}

#pragma GCC diagnostic pop

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