UNIVERSITY OF VICTORIA

Department of Electrical and Computer Engineering

ECE 355 – Microprocessor-Based Systems

Project Report

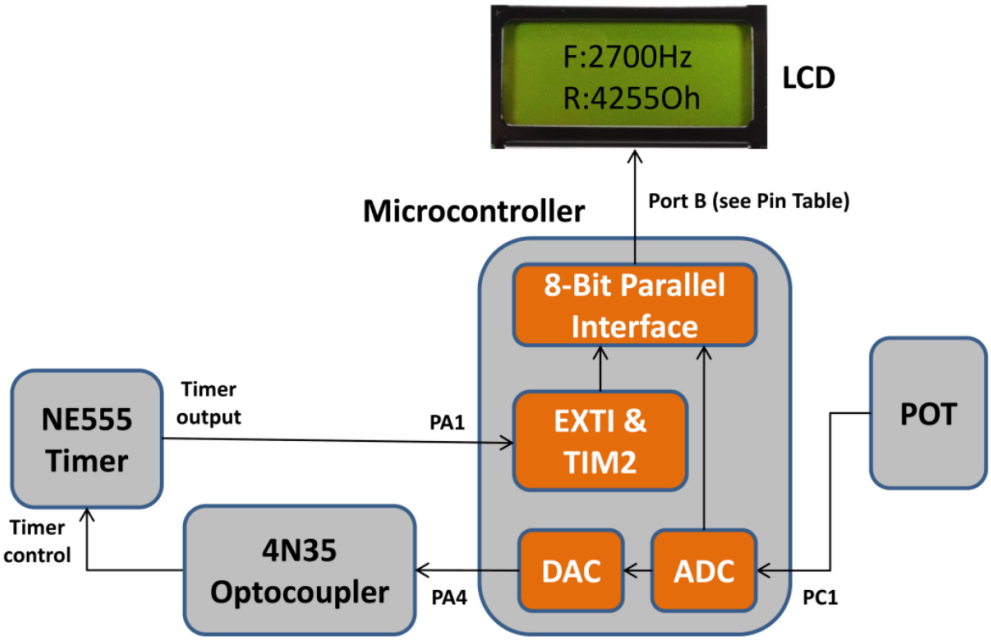
Report Submitted on: November 29, 2020

Name: Kutay Cinar - V00******

1.0 Problem Description/Specifications

The objective of this lab project is to develop an embedded system for monitoring and controlling a PWM signal generated by an external 555 timer. In this lab report, an external optocoupler driven by the microcontroller on the STMF0 Discovery board is to be used to control the frequency of the PWM signal.

An overall system diagram is shown below as taken from the lab manual.



The microcontroller is also used to measure the voltage across a potentiometer on the ECE 355 Emulation board and relay it to the external optocoupler for controlling the PWM signal frequency. Additionally, the measured timer frequency and the corresponding POT resistance are to be displayed on the LCD screen of the ECE 355 Emulation board.

2.0 Design/Solution

This project involved 3 major parts. They were the development of the LCD screen, ADC (Analog-to-Digital Converter) part, and DAC (Digital-to-AnalogConverter) part.

First thing to complete has been to set up the GPIO registers for ports A, B, and C. As without configuring the registers, there would be no values to play and test with. After all the ports were configured, the three major parts of the project were completed in order in their own functions.

2.1 LCD Solution

The LCD screen uses only the GPIOB ports to communicate with our board which was set up in the initial phase. The way the interface and the GPIOB port set up has been done matched the following input from Interface Examples slides, as seen in the table below.

STM32F0	SIGNAL	DIRECTION
PB4	ENB (LCD Handshaking: "Enable")	OUTPUT
PB5	**RS** (0 = COMMAND, 1 = DATA)	OUTPUT
PB6	**R/W** (0 = WRITE, 1 = READ)	OUTPUT
PB7	DONE (LCD Handshaking: "Done")	INPUT
PB8	**D0**	OUTPUT
PB9	**D1**	OUTPUT
PB10	**D2**	OUTPUT
PB11	**D3**	OUTPUT
PB12	**D4**	OUTPUT
PB13	**D5**	OUTPUT
PB14	**D6**	OUTPUT
PB15	**D7**	OUTPUT

After setting up the port, the next steps were to initialize the LCD. To do this, there were four instructions given to us to send to the LCD. The initialization instructions were:

1. Instruction: **00 0011 1000** (DL = 1, N = 1, F = 0)
2. Instruction: **00 0000 1100** (D = 1, C = 0, B = 0)
3. Instruction: **00 0000 0110** (I/D = 1, S = 0)
4. Instruction: **00 0000 0001** (clear display)

Once the initial configuration was completed in an myLCD_Init() function, communication to the LCD board was done with functions GPIO_WriteBit() and GPIOB->ODR to send data.

The format of sending data to the LCD board had to also be done in writeLCD() following the format of the slides, where the bits were sent over first and a handshake operation was performed for confirmation.

Sequence of steps taken to perform the handshake/or send an arbitrary instruction:

- Send **RS and R/W** to PB[6:5] and **Instruction** to PB[15:8]
- Set PB[4] = 1 (assert “Enable”)
- Wait for PB[7] to become 1 (“Done” to be asserted)
- Set PB[4] = 0 (deassert “Enable”)
- Wait for PB[7] to become 0 (“Done” is deasserted)

This concluded the LCD initialization and sending over instructions. An additional helper function called printLCD was also written to print the frequency and resistance values on the

LCD by sending over the corresponding ASCII characters and incrementing between the upper and lower halves of the LCD screen.

2.2 ADC Solution

The analog voltage signal coming from the potentiometer on the ECE 355 Emulation board was measured continuously by the ADC using GPIOC by using a polling approach. Using those POT measurements, a resistance value was calculated in the main function's while loop with the lower and upper limits of the measurable voltage.

The upper limit was then scaled by 5000 and divided by 0xFFF (4095) which was the default resolution of the ADC and DAC (12 bits as specified in the reference manual).

2.3 DAC Solution

The digital voltage was calculated by converting the POT measurement from the ADC. The initialization was done inside a function called myDAC_Init() where it was enabled by using D AC_CR_EN1 mask which used GPIOA pin 4 as the output.

The frequency generation used here is identical to the code in part 2 of the introductory lab, where a period and frequency was displayed using a square-wave signal. However, EXTI0_1 is used in this solution and TIM2 as required by the lab manual.

Testing/Results

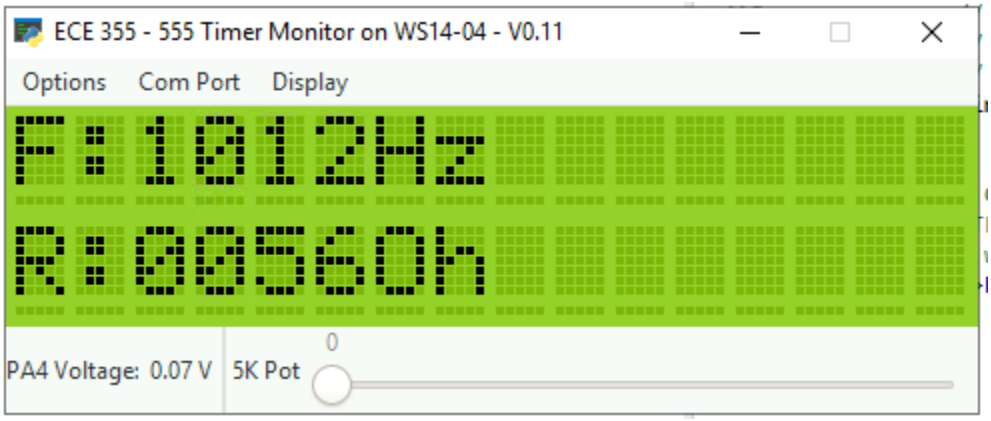
The program was tested on the online ELW B328 lab computers, mainly using ws14-04 as it was my assigned computer. Periodically other machines were used when developing the code while ws14-04 was unavailable.

Results from ws14-04 machine during the lab demo are as follows:

The minimum value of frequency and resistance obtained in this project were:

Frequency: 1015±20 Hz

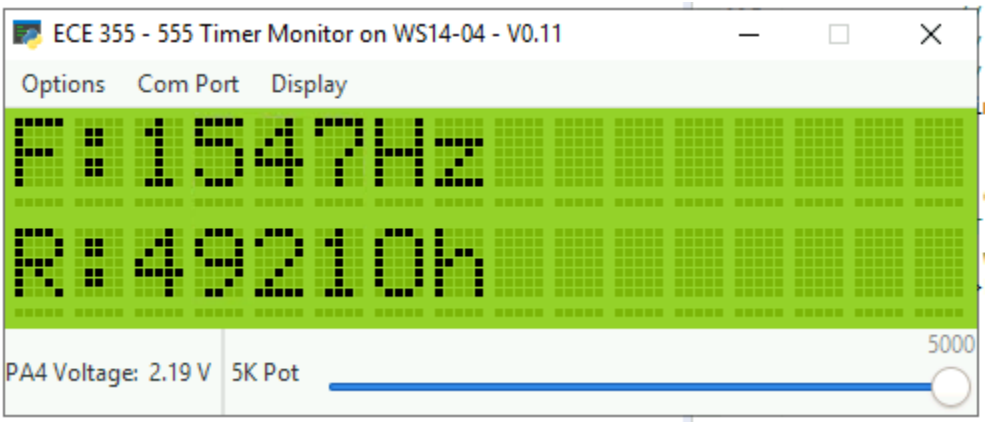
Resistance: 50±20 Ohm



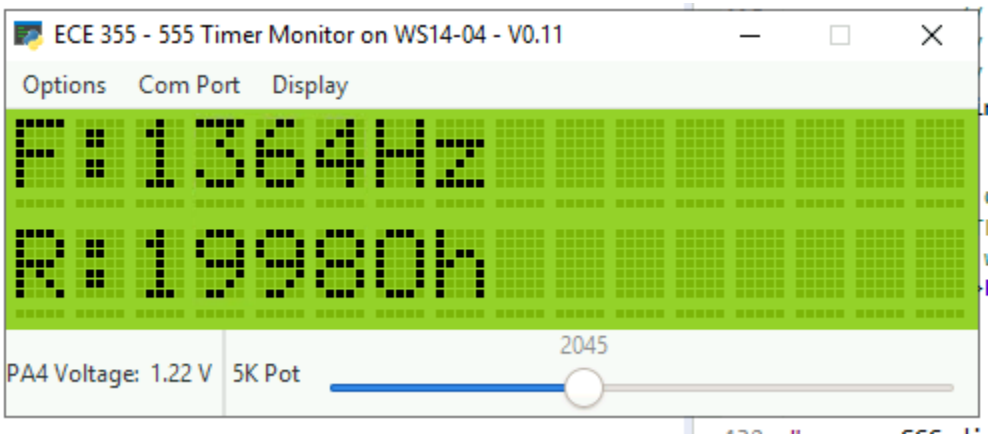
The highest value of frequency and resistance were:

Frequency: 1550±50 Hz

Resistance: 4920±50 Ohm



A mid range reading at POT = 2045 is also shown below:



Discussion

The results obtained in this lab for frequency are in range from 1015-1550 Hz and for resistance are in range from 50-4950 Ohm. They are consistent and within the expected limits of 1000-1600 Hz for frequency and 0-5000 Ohm for resistance given before the lab demo. Due to the labs being completely online, there were no expectations of creating circuit diagrams and working on breadboards. The virtual nature of the lab also meant that when they were issues, external help had to be relied on.

The LCD system and the Signals window on the ECE 355 Emulation board sometimes did not register write operations even though the GPIO ports being set up correctly. This hindered development at times when no other machines were available on labs.engr.uvic.ca. This issue could only be fixed by resetting the board as later found out when demonstrating my demo and the lab technician pointed it out the fix.

Overall the lab project was a fun and enjoyable experience that supported the class material and assignments, and contributed to my learning and understanding of the course.

Appendices

Source code

```
// Kutay Cinar
// V00******

//
// 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 <stdlib.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.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)


void myGPIOA_Init(void);
void myGPIOB_Init(void);
void myGPIOC_Init(void);
void myTIM2_Init(void);
void myEXTI_Init(void);
void myLCD_Init(void);
void myADC_Init(void);
void myDAC_Init(void);

uint16_t getADCValue(void);

void writeLCD(uint16_t rs, uint16_t data);
void printLCD(float freq, float resist);


// 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.

// Global
int timerTriggered = 0;
```

```c
float frequency = 0;
float resistance = 0;

float digital;
float analog;

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

	trace_printf("Kutay Cinar's ECE 355 Lab Project...\n");
	trace_printf("System clock: %u Hz\n", SystemCoreClock);

	myGPIOA_Init();		/* Initialize I/O port PA */
	myGPIOB_Init();		/* Initialize I/O port PA */
	myGPIOC_Init();		/* Initialize I/O port PA */
	myTIM2_Init();		/* Initialize TIM2 */
	myEXTI_Init();		/* Initialize EXTI */
	myLCD_Init();		/* Initialize LCD */
	myADC_Init();		/* Initialize ADC */
	myDAC_Init();		/* Initialize the LCD */

	while (1)
	{
		/* Start ADC conversion */
	  ADC1->CR |= ADC_CR_ADSTART;

		/* Wait Buffer for conversion complete flag */
		while ((ADC1->ISR & ADC_ISR_EOC) != ADC_ISR_EOC);

		digital = ADC1->DR;

		/* Load digital value into DAC_DHRx register*/
		DAC->DHR12R1 = digital;

		/* Read Analog value from DAC */
		analog = DAC->DOR1;

		/* Resistance calculations */
	  resistance = (digital / 0xFFF) * 5000; // 0xFFF ADC and DAC bit
resolution default 12 bits.

	}
```

```
	return 0;

}

void myLCD_Init()
{
	//DL = 1, N = 1, F = 0
	writeLCD(0, 0b00111000);

	// D = 1, C = 0, B = 0
	writeLCD(0, 0b00001100);

	// I/D = 1, S = 0
	writeLCD(0, 0b00000110);

	// clear display
	writeLCD(0, 0b00000001);
}

void writeLCD(uint16_t rs, uint16_t data)
{
	// Send instructions DB7-0
	GPIOB->ODR = data << 8;

	// Send RS and R/W (always 0) to PB[6:5]
	GPIO_WriteBit(GPIOB, GPIO_Pin_5, rs);
	GPIO_WriteBit(GPIOB, GPIO_Pin_6, 0);

	// Make PB[4] = 1 (assert "Enable")
	GPIO_WriteBit(GPIOB, GPIO_Pin_4, 1);

	// Wait for PB[7] to become 1 ("Done" to be asserted)
	while(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_7) != 1);

	// Make PB[4] = 0 (deassert "Enable")
	GPIO_WriteBit(GPIOB, GPIO_Pin_4, 0);

	// Wait for PB[7] to become 0 ("Done" to be deasserted)
	while(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_7) != 0);
}
```

```
void printLCD(float freq, float resist)
{
    char freq_chars[9] = "F:0000Hz";
    freq_chars[8] = '\0';

    char res_chars[9] = "R:0000Oh";
    res_chars[8] = '\0';

    for (int i=5; i>1; i--)
    {
      freq_chars[i] = (int)freq % 10 + 48;
        freq = freq / 10;

        res_chars[i] = (int)resist % 10 + 48;
        resist = resist / 10;
    }

    // Write upper half LCD frequency
    writeLCD(0, 0b10000000);
    for (int i=0; i<8; i++)
    {
        writeLCD(1, freq_chars[i]);
    }

    // Write lower half LCD resistance
    writeLCD(0, 0b11000000);
    for (int i=0; i<8; i++)
    {
        writeLCD(1, res_chars[i]);
    }
}

void myGPIOA_Init()
{
      // Enable clock for GPIOA peripheral
      RCC->AHBENR |= RCC_AHBENR_GPIOAEN;

      // Configure PA1 as input: 00 is input
      GPIOA->MODER &= ~(GPIO_MODER_MODER1);

      // Ensure no pull-up/pull-down for PA1
      GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR1);
```

```
	// Configure PA4 as analog
	GPIOA->MODER |= GPIO_MODER_MODER4;

	// Ensure no pull-up/pull-down for PA4
	GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR4);
}


void myGPIOB_Init()
{
	/* Enable clock for GPIO B */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);

	/* Create GPIO B struct */
	GPIO_InitTypeDef GPIO_InitStruct;

	/* Set pins 4-6 and 8-15 */
	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6 |
GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12 |
GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;

	/* Set GPIO B mode to output */
	GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;

	/* Set GPIO B speed */
	GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;

	/* Set the GPIO B PB4-6 and PB8-15 output type to Push/Pull */
	GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;

	/* Turn off the GPIO B pull up/pull down */
	GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;

	/* Apply configuration for PB4-6 and PB8-15 */
	GPIO_Init(GPIOB, &GPIO_InitStruct);

	/* Configure PB7 for input */
	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_7;

	/* Set GPIO B mode to OUTPUT */
	GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
```

```
	/* Set GPIO B Output Speed */
	GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;

	/* Set the GPIO B PB7 output type to Push/Pull */
	GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;

	/* Turn off the GPIO B pull up/pull down */
	GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;

	/* Apply configuration for PB7 */
	GPIO_Init(GPIOB, &GPIO_InitStruct);
}


void myGPIOC_Init()
{
	/* Enable clock for GPIOC peripheral */
	RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

	/* Configure PC1 as input */
	GPIOC->MODER |= GPIO_MODER_MODER1;

	/* Ensure no pull-up/pull-down for PC1 */
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPDR1);
}


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

	// Calibrate the ADC
    ADC1->CR = ADC_CR_ADCAL;

    /* Wait for calibration to finish */
    while(ADC1->CR == ADC_CR_ADCAL);

    /* Configure ADC for continuous and overrun mode */
    ADC1->CFGR1 |= (ADC_CFGR1_CONT | ADC_CFGR1_OVRMOD);
```

```
	// Configure ADC to use PC1
	ADC1->CHSELR |= (uint32_t)0x800;

	// Configure ADC to use fast sampling speed
	ADC1->SMPR &= (uint32_t)0xFFFFFFF8;

	// Enable ADC for conversion
	ADC1->CR |= (uint32_t)0x1;

	/* Wait for enable flag */
	while((ADC1->ISR & 0x1) != 1);

}

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

	/* Set EN1 bit in DAC_CR to set DAC_OUT1 enabled */
	DAC->CR |= DAC_CR_EN1;
}

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 */
	SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI1_PA;

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

	/* Unmask interrupts from EXTI1 line */
	EXTI->IMR = EXTI_IMR_MR1;

	/* Assign EXTI1 interrupt priority = 0 in NVIC */
	NVIC_SetPriority(EXTI0_1_IRQn, 0);

	/* Enable EXTI1 interrupts in NVIC */
	NVIC_EnableIRQ(EXTI0_1_IRQn);
}


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

```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_stm32f0xx.c */
void EXTI0_1_IRQHandler()
{

	/* Check if EXTI2 interrupt pending flag is indeed set */
	if ((EXTI->PR & EXTI_PR_PR1) != 0)
	{
		//
		// 1. If this is the first edge:
		if (!(TIM2->CR1 & TIM_CR1_CEN)) {

			timerTriggered = 1;

			//    - Clear count register (TIM2->CNT).
			TIM2->CNT = 0;

			//    - Start timer (TIM2->CR1).
			TIM2->CR1 |= TIM_CR1_CEN;
		}

		//    Else (this is the second edge):
		else {
```

```
                //    - Stop timer (TIM2->CR1).
                TIM2->CR1 &= ~(TIM_CR1_CEN);

                //    - Read out count register (TIM2->CNT).
                float count = TIM2->CNT;

                //    - Calculate signal period and frequency.
                frequency = (float)SystemCoreClock / count;

                //    - Print calculated values to the console.
                // trace_printf("Frequency: %f Hz Period: %f s\n",
frequency, period);
                trace_printf("digital: %.1f, analog: %.1f, res %.1f,
freq: %.1f\n", digital ,analog, resistance, frequency);
               printLCD(frequency, resistance);

                //      NOTE: Function trace_printf does not work
                //      with floating-point numbers: you must use
                //      "unsigned int" type to print your signal
                //      period and frequency.
                timerTriggered = 0;
            }

            // 2. Clear EXTI2 interrupt pending flag (EXTI->PR).
            // NOTE: A pending register (PR) bit is cleared
            // by writing 1 to it.
            EXTI->PR |= EXTI_PR_PR1;
      }
}


#pragma GCC diagnostic pop

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

References

[1] Dr. Daler N. Rakhmatov, Brent Sirna, Khaled Kelany, "ECE 355: Microprocessor-Based Systems Laboratory Manual (ONLINE)" University of Victoria, Victoria, 2020.

[2] Dr. Daler N. Rakhmatov, “Interface Examples," University of Victoria, Victoria, 2020.

[3] “STM32F0xxx programming manual”, STMicroelectronics, 2012.

[4] “STM32F0 reference manual”, STMicroelectronics, 2014.

[5] “STM32F051 data sheet”, STMicroelectronics, 2014.