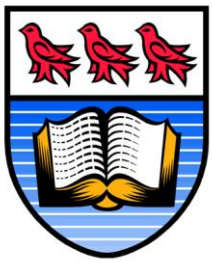
ECE 355 - Microprocessor-Based Systems

Project Report

Section B05



By: Minh Dai Tran- V00928014

1.Problem Description

The purpose of this project is to create an embedded system for monitoring and controlling a PWM signal generated by a 555 timer(NE555 IC).This project uses an STMF32f0 microcontroller board to drives an external 4N35 optocoupler which controls the PWM signal by adjusting the resistance on voltage divider, this frequency as well as the resistance are then display on the emulation LCD screen which uses Hitachi HD44780 LCD as its base. A potentiometer (POT) on the Emulation board is connected to an analog to digital converter (ADC) on the microcontroller, this digital value is then used by the 8-bit parallel interface to be display on the LCD screen. The DAC then uses the digital value of the voltage(0-5V) and send that signal to the 4N35 optocoupler to the NE555 timer to control the PWM signal. The frequency of this signal is then read by the microcontroller using a timer interrupt display on an LCD through the 8-bit parallel interface.

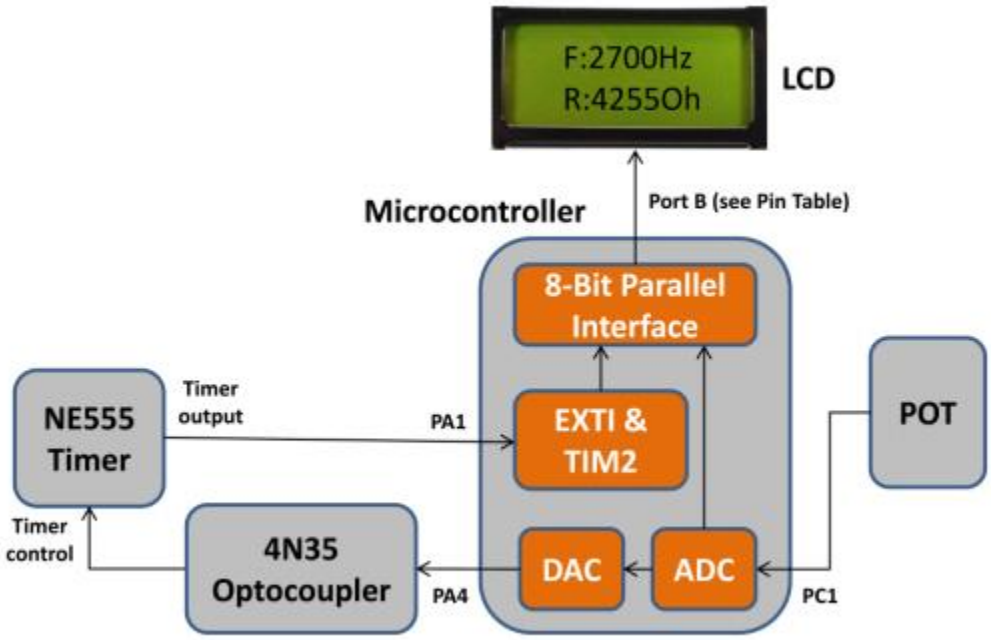


Figure 1: Block Diagram of the Embedded System. Adopted from (1)

2 Design Solution

The design solution of the project is split up into four separate sections base on code functionality. These sections are Frequency counter (EXTI&TIM2), ADC, DAC and LCD display(8-bit parallel Interface). Each section of the code was developed and tested independently before being implemented in the final design solution. The frequency counter was developed back in during the first lab section and is reused for the project. The pin layout of the microcontroller board can be seen in the table below.

STM32F0	SIGNAL	DIRECTION
PA0	USER PUSH BUTTON	INPUT
PC8	BLUE LED	OUTPUT
PC9	GREEN LED	OUTPUT
PA1	555 TIMER	INPUT
PA2	FUNCTION GENERATOR (for Part 2 only)	INPUT
PA4	DAC	OUTPUT (Analog)
PC1	ADC	INPUT (Analog)
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

Figure 2: Pin Layout Table of STM32f0 Discovery Board. Adopted from (1).

Additionally, TIM3 is setup for a wait(ms) function, which is set to wait for a certain number of milliseconds.

2.1 Frequency Counter (EXTI&TIM2)

This section of the code is used to calculate the frequency using a timer and an external interrupt (EXTI) from PA1. Pin PA1 is initialized as input in the myGPIO_Init() function. The project uses TIM2 which is a 32-bit timer to count between each rising edge of the square PWM wave. The EXTI is set up so an interrupt occurs when a rising edge signal is read at PA1.

TIM2 is set up with no pre-scaling, and interrupt priority of 0. The EXTI is setup to detect a rising edge on PA1. The function EXTI0_1_IRQHandler is called whenever there is an interrupt, this function will first check if the interrupt comes from PA1 and then check for the first edge (edge = 0). If it is the first edge (edge = 0), the counter on TIM2 is set to 0, TIM2 is set to start counting and edge is set to "edge = 1". If it is not first edge(edge = 1), edge is set back first edge(edge =0), TIM2 is set to stop

counting, frequency is calculated by dividing the system clock by the counter on TIM2(the ticks between each rising edge).

2.2 ADC

The ADC is set up to read the voltage from a 5 KOhm POT. The input pin for the POT is PC1 and it is initialized as an analog input in the myGPIO_Init() function. Within the ADC initialize function (myADC_init()), the clock register AHBENR is enable as ADC required a clock to operate, ADC configuration (CFGR1) is set to continuous conversion and overrun mode which allow the ADC to continuously gathered and convert the voltage value to be used by the NE555 timer and overrun mode allow the previous value to be overwrite when a new value is read, ADC channel is sets to PC1 which correspond to channel 11, then ADC is set to enable and a while loop is put in place to wait for enabling to complete.

After enabling ADC conversion, the function getADC() is used to read the ADC value on the POT and returns a value between 0-4095. The function get_ADC() will start for the conversion process, wait for conversion to be completed, reset the ADC conversion flag and return the value read on PC1. The function Res_Conversion(res) is then used to convert this value to the POT equivalent resistance of 0-5000 by dividing it by 4095 and multiply it by 5000.

In the main function, getADC() is put inside a while loop and the resistance return by this function is converted to the correct value and display on the LCD.

2.3 DAC

The DAC is used to convert the voltage value to analog and output it through PA4 on the STM board to be used by the frequency generated by the NE555 timer. PA4 is initialized as analog output in the myGPIO_Init() function. In the myDAC_Init() function, the DAC clock is enable and DAC channel is set to channel 1 which correspond to PA4.

In the main function, the line “DAC->DHR12R1 = ADC1->DR” is used output the value on pin PA4 to the 4N3555 optocoupler.

2.4 LCD Display(8Bit-parallel-interface)

The 8-bit parallel interface takes the value of resistance from the ADC and the frequency from the NE555 timer to display them on the LCD screen. The 8-bit parallel interface is using pin PB4:15 as out put except for pin PB7 which is used to accept input from the LCD display. In the myGPIO_Init() function pins PB4:15 MODER are set to “01” which are outputs pins with the speed of 10MHz, pin PB7 is set as input.

To write a command to the LCD screen, the function lcd_cmd(cmd) is used. This function first set Pin 5(RS) and Pin 6(RW) to 0, it then replace pin 15:8 of port B with the 8 bit command(cmd), it then set Pin 4 to 1 which enable the microcontroller to write to the LCD screen, wait for input of 1 from the LCD screen from(this input indicate that the writing process is complete), it then set Pin 4 to 0 which allow for the LCD screen to start processing and displaying the command, it then wait for the output from Pin 7 of 0 from the LCD screen(this input indicate that the display process is complete. The function will wait 50 milliseconds after each command to allow for the LCD screen to display properly.

The lcd_send(a) command is used to send an ASCII code to the LCD screen which is then used to display the ASCII symbol on the LCD screen. This function follow the same algorithm as the lcd_cmd(cmd) function but with Pin 5(RS) set to 1 and Pin 6(RW) set to 0. Instead of replacing pin 15:8 with a command cmd, this function will replace it with the 8 bit ASCII code instead.

In the initialization function lcd_Init, pin 4:15 are first set to 0, we then send four commands to the LCD screen. These commands written in 8 bits are 0x38(DL = 1, N = 1, F = 0) which makes DDRAM use 8 bits, 2 lines of 8 characters, 0xC(D = 1, C = 0, B = 0) which display no cursor and no blinking, 0x6(I/D = 1, S = 0) which set DDRAM address to auto increment after each access with no shift, 0x1 which clear the LCD display.

Within the while loop in the main function, lcd_write(Frequency,Resistance) is use to output the value of frequency and resistance to the LCD display. This function first sends the command 0x0080 to the LCD which set the cursor to the first position of the first line, it then prints 'F' and ':' to the display. A for loop is set up to loop through every single digit of the frequency from left to right and display those digits on the LCD screen. The output 'H' and 'z' are sent to the LCD to be displayed. The first line of the LCD should display "F:(Frequency Value)Hz". After display the frequency value, the command 0x00C0 will send the cursor to the second line of the display and used the algorithm above to display the resistance value. The second line of the LCD should display : "R:(Resistance Value)Ohm".

3. Testing and Results

The following section covers the test procedure for each section of the code and the final result when the system is fully implemented.

3.1 Frequency, TIM2, EXTI

The frequency output, TIM2 and EXTI sections of the code were tested in the first lab where a function generator is used to generate a square-wave signal to display its frequency on the console using the trace_printf() function. From testing the lab co. When implementing this code to the project, trace_printf() is used to print the frequency to the console to test whether or not we are getting the correct frequency.

From lab1, the maximum frequency can be measured by the system is around 750 KHz and the minimum readable frequency is 11.1mHz(but this value cannot be displayed properly by trace_printf()).

3.2 POT resistance, DAC & ADC

In order to test whether or not the DAC is working properly, we use trace_printf() to print the resistance after using getADC and ADC conversion to get the value for resistance, in this test the value being printed is somewhat the same as POT resistance on the POT slider which mean the DAC is using properly. To check if the ADC is working properly, we check if the frequency output from the timer change when we vary the voltage output from the PA4 by varying the resistance value from the POT. From the test, the value of frequency increase with increase resistance (increase voltage) and decrease

with decrease resistance (decrease voltage). The resistance value ranges from 70-4890 and the frequency value range from 880-1390 which is within 20% of the error range specified.

3.3 LCD Display

The LCD display is test by inserting a random value for frequency and resistance into the lcd_write() function and observe that the two values are output properly with the first line of the LCD displaying: “F:(Frequency Value)Hz” and the second line of the LCD displaying :” R:(Resistance Value)Ohm”. In the testing process, Frequency is set to 2750 and Resistance is set to 4800. The LCD display the following:

F:2750Hz- On first line.

R:4800Ohm- On second line.

Due to the limitation that was set for the lcd_write() function, it can only output a 4 digit number for both the frequency and resistance and it cannot print decimal. The proper symbol for resistance unit is ‘Ω’ but it is instead replacing with ‘Ohm’. There is a wait of 1 second on the while loop to allow for the Frequency and the LCD screen to be update properly, hence there will be a slight delay of 1 second between changing the POT value on the slider and the LCD updating with the new frequency and resistance.

3.5 Result

Once each section of the code is confirmed to be working properly. The frequency outputted from the NE555 timer and the resistance of the POT are sent to the LCD to be written on the LCD display using lcd_write() command. The value of the POT on the slider is then varied so the change in frequency and resistance on the LCD display can be observe in real time.

By setting the POT to its minimum value, the LCD display the following:

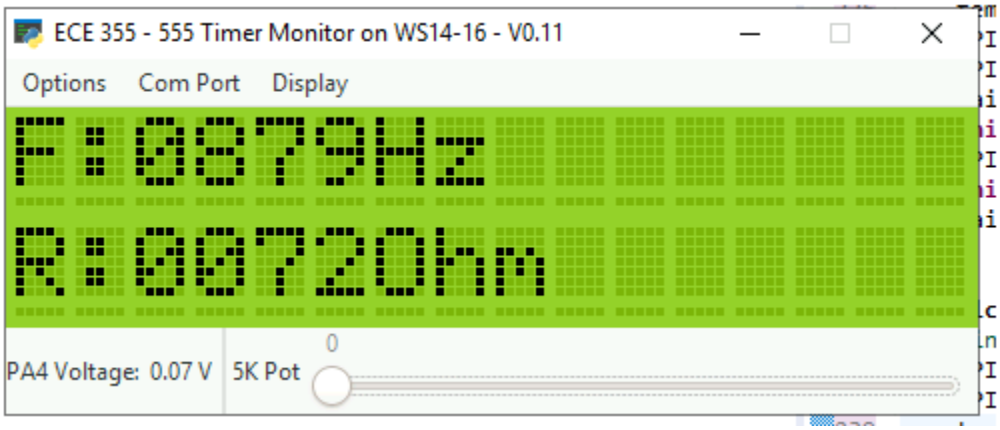


Figure 3: LCD Display when POT Sets to 0.

By setting the POT to its maximum value, the LCD display the following:

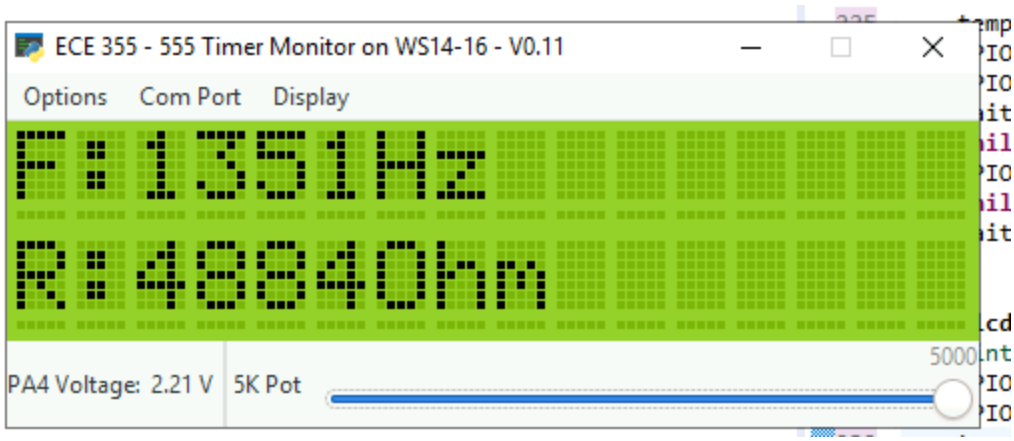


Figure 4: LCD Display when POT Sets to 5000.

4. Discussion

The expected result for resistance is 0-4900 Ohm, for frequency is 800-1600 Hz and 0-2.22 for PA4 voltage. The experimental result for resistance is 72-1351 Ohm, for frequency is 880-1351 Hz and PA4 0.07-2.221. From this experimental result, the value for PA4 voltage is consistent with our expected result. However, there is a difference in the result for resistance and frequency. The experimental minimum result for resistance hover around 70-80 with out ever reaching 0, this may be because there is an internal resistance for the POT which is being read by the DAC. There is a more notable result for the frequency, but this difference stay within 10-20% of the expected result which mean it is still acceptable. The result in this report is obtained when using station 16. However, when using a different station (station 2), the result I obtain for frequency is between 850-1590 which is closer in range with expected result for frequency. I suspect that there is a difference in internal clock between the different STM Board being used for this lab which causes this discrepancy.

The value for resistance and frequency on the LCD display also fluctuate between back and fort in the last digit which make it hard to read the exact value. Due to the limitation noted in section 3.1, the system STM board cannot be used to observe any frequency lower than 11.1mHz or higher than 750MHz. Due to the limitation specify in section 3.3, this system cannot be used to observe a frequency or resistance in the decimal or read a frequency or resistance that is higher than 9999 Hz and 9999 Ohm respectively, these limitation can be improved in the programming aspect of the project.

From doing this project, I learn how helpful it is to divide the final projects in smaller sub-section and make sure each section work before implementing them in the final product. It is also helpful how the individual sub-section of this project can also be implemented in a different project that require either ADC, DAC or LCD display with 8 bits parallel. This aspect of the lab also helps me visualize the lab as a whole and gain a better understanding on how each modular part of the lab interact with each other. Another helpful part that I learn from this lab is how to use interrupt which allow for an external interrupt to be running at the same time as the main function, this allow for speed optimization of mechatronic system.

If I were to attempt this lab from the beginning, I would definitely want a better understand of MODER registry by reading the STM32f0 reference manual as I had a hard time understanding what each mode for the pins meant. Alternatively, understanding registers is a huge part of this lab so spending time learning about register is very helpful. Lastly, when setting register bit for initialization, it may be easier to visualize the registers when setting the register bits using binary(0bxxxxxxxxxx) instead of hexadecimal, the compiler this lab is using a gcc compiler which allow for the system to read binary even when programming in C.

5. References

[1] B. Sirna, K. Kelany and D. Rakhmatov, ECE 355: Microprocessor-Based Systems Laboratory Manual, University of Victoria, 2020.

6. Appendix A: Project Code

```
#include <stdio.h>

#include <stdlib.h>

#include "diag/Trace.h"

#include "cmsis/cmsis_device.h"

#include "assert.h"

#include "stm32f0xx_gpio.h"

#include "stm32f0xx_rcc.h"

#include "stm32f0xx.h"

#include "math.h"

// ----------------------------------------------------------------------------

// ---------Defines-----------------------------------------------------------

#define myTIM2_PRESCALER ((uint16_t)0x0000) // Clock prescaler for TIM2 timer: noprescaling

#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)// Maximum setting for overflow

#define MAX_Resistance ((int)5000)

#define MIN_Resistance ((int) 1)

#define myTIM3_PRESCALER ((uint16_t) 0xBB80)          // Clock prescaler for TIM3 timer: 48Mhz/48K (0xBB80 = 48000) = 1Khz prescaling
```

```
#define myTIM3_PERIOD ((uint32_t) 0xFFFFFFFF)           // Maximum possible setting for overflow


// ----------------------------------------------------------------------------
// ---------Global Variables---------------------------------------------------
volatile uint32_t Resistance = 0;
volatile uint32_t Frequency =0;
volatile uint32_t edge = 0;
// ----------------------------------------------------------------------------
// --------Declarations--------------------------------------------------------
void myADC_Init(void); // Initializes the ADC itself
void myGPIO_Init(void); //Initializes GPIO for ADC
uint32_t getADC(void);// Fetches the ADC value and converts it
void myTIM3_Init(void);
void wait(int);
int Res_Conversion (uint32_t Input);
void myDAC_Init(void);
void myTIM2_Init(void);
void myEXTI_Init(void);
void lcd_Init(void);
void lcd_write(int, int); //for writing data on LCD
void lcd_cmd(uint16_t); //send command to LCD
void lcd_send(char);// send ASCII code to LCD
// ----------------------------------------------------------------------------
// ----- 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"


int main(int argc, char* argv[]){
	uint32_t res = 0 ;
	myGPIO_Init(); // Initializes all GPIO pins, PA0 1 4, PC 1 PB 4 5 6 7 8 9 10 11 12 13 14 15
	myADC_Init(); // Initializes the ADC to measure the POT resistance
	myTIM3_Init();  // Initialize timer TIM3
	myDAC_Init(); // Initialize the DAC
	myEXTI_Init(); /* Initialize EXTI to measured the frequency of the signal*/
	myTIM2_Init(); /* Initialize timer TIM2 to time the period of the signal*/
	lcd_Init();
	while (1)
	{	wait(1000);
		res = getADC(); // Gets the value from the POT and returns a number from 0 - 4095
		Resistance = Res_Conversion(res); // converts the (0-4095 value to the equivalent
resistance 0- 5000 ohms)
		DAC->DHR12R1 = ADC1->DR;	//DAC output toward 4N35 Octocoupler
		wait(100);
		lcd_write(Frequency,Resistance);
	}


}


void myGPIO_Init(){
	//PA1 -> digital -> used to generate interrupts to measure frequency.
	//PA4 -> analog -> used for DAC to change the frequency of the oscillator
	RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* Enables clock for GPIOA peripheral */
	//PA1
	GPIOA->MODER &= ~(GPIO_MODER_MODER1); //Timer Input
```

```
	GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR1); // No pull up/down
	//PC1 -> analog -> Used to measure Pot's resistance
	// Enable clock for GPIOC
	RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
	GPIOC->MODER &= ~(GPIO_MODER_MODER1); // Analog input
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPDR1); // No pull up/down
	//PA4 DAC output
	GPIOA->MODER &= ~(GPIO_MODER_MODER4); //Analog output
	GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR4); // No pull up/down

	//Initializes PBs
	RCC->AHBENR |= RCC_AHBENR_GPIOBEN; // start GPIOB clock
	GPIOB->MODER = (GPIOB->MODER & 0xFFFFFF00)| (0x55551500);
	GPIOB->PUPDR = (GPIOB->PUPDR & 0xFFFFFF00)| (0X0<<8); /* Ensure no pull-up/pull-down for
PB */
}

uint32_t getADC(){
	ADC1->CR |= ADC_CR_ADSTART//start ADC conversion
	/*wait until the conversion has finished*/
	while(!(ADC1->ISR & ADC_ISR_EOC)){};
	ADC1->ISR &= ~(ADC_ISR_EOC); // resets conversion flag
	return ((ADC1 ->DR) &ADC_DR_DATA); // Returns on ADC1's data masking the rest
}
void myADC_Init(){
	RCC->APB2ENR |= RCC_APB2ENR_ADCEN; // Enable clock for ADC
	/*Configuration of ADC for continuous conversion and overrun modes */
	ADC1->CFGR1 |= (ADC_CFGR1_CONT | ADC_CFGR1_OVRMOD);
	/*Sets ADC channel to be C1 */
```

```
	ADC1->CHSELR = ADC_CHSELR_CHSEL11;
	/*Enable ADC and wait for it to complete
	ADC1->CR |= ADC_CR_ADEN;
	while (!(ADC1->ISR & ADC_ISR_ADRDY)) {};
}

int Res_Conversion(uint32_t res){
	float Percent = (float)(Input / 4095.0); // Converts the value read from a 0 - 4095 value to a percent
	int Res = (int)(Percent * MAX_Resistance); // Scales the percent by the maximum resistance possible
	return Res; // Returns the equivalent resistance for a given input
}

void myDAC_Init(){
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE); //Enable DAC clock
	DAC->CR |= DAC_CR_EN1;		//enable DAC channel 1 corresponding to PA4
	}

void myTIM2_Init(){
	/* Enable clock for TIM2 peripheral */
	// Relevant register: RCC->APB1ENR
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; /*TIM2 enable*/
	/* 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); /*buffer auto-reload, count up, stop on overflow,
	enable update events, interrupt on overflow only*/
	/* 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] = ((uint32_t) 0x00000080); /*SETS EXTI 1 to be bound to PA 1pin*/
	/* EXTI1 line interrupts: set rising-edge trigger */
	// Relevant register: EXTI->RTSR
	EXTI->RTSR = ((uint32_t) 0x00000002); /*Sets rising edge trigger for TR1 and TR 2*/
	/* Unmask interrupts from EXTI1 line */
	// Relevant register: EXTI->IMR
	EXTI->IMR = ((uint32_t) 0x00000002); /*Unmasks MR1 leaving the rest masked*/
	/* Assign EXTI1 interrupt priority = 0 in NVIC */
	// Relevant register: NVIC->IP[1], or use NVIC_SetPriorit
	NVIC_SetPriority(EXTI0_1_IRQn, 0); /*EXTREAMLY UNSURE*/
```

```
	/* Enable EXTI1 interrupts in NVIC */
	// Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
	NVIC_EnableIRQ(EXTI0_1_IRQn); /*Enables interupts on External IRQ lines?*/
}
/* 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*** Timer Oveflow (555) ***\n");
		/* Clear update interrupt flag */
		// Relevant register: TIM2->SR
		TIM2->SR &= ~(TIM_SR_UIF); /*Flips the first bit from 1 to 0*/
		/* Restart stopped timer */
		// Relevant register: TIM2->CR1
		TIM2->CR1 |= TIM_CR1_CEN; /*Toggles the first bit (hopefully) */
	}
}
void myTIM3_Init(){
  RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;    // Enable clock for TIM3 peripheral - Relevant register: RCC->APB1ENR
  TIM3->CR1 = ((uint16_t) 0x006C);       // Configure TIM3: buffer auto-reload, count up, stop on overflow, enable update events, interrupt on overflow only
  TIM3->PSC = myTIM3_PRESCALER;          // Set clock prescaler value
  TIM3->EGR |= ((uint16_t) 0x0001);      // Update timer registers - Relevant register: TIM3->EGR
}

// Use tim3 to wait for a certain amount of clock cycles
void wait(int ms){
  TIM3->CNT = ((uint32_t) 0x00000000);           // Clear the timer
```

```
   TIM3->ARR =  ms;                                          // Set the timeout value from parameter
   TIM3->EGR |= ((uint16_t) 0x0001);      // Update registers
   TIM3->CR1 |= ((uint16_t) 0x0001);      // Start the timer
   while((TIM3->SR & 0x0001) == 0);          // Wait until timer hits the desired count
   TIM3->SR &= ~((uint16_t) 0x0001);      // Clear update interrupt flag
   TIM3->CR1 &= ~0x0001;                 // Stop timer (TIM3->CR1)
}
/* This handler is declared in system/src/cmsis/vectors_stm32f0xx.c */
void EXTI0_1_IRQHandler(){
        float frequency;
        int count;
        if ((EXTI->PR & EXTI_PR_PR1) != 0){
                if (edge == 0) { //if first edge
                        TIM2 -> CNT = 0X00000000; // clear counter register
                        edge = 1;
                        TIM2 -> CR1 = TIM_CR1_CEN; // start timer
                }
                else {
                        TIM2 -> CR1 &= ~(TIM_CR1_CEN); // DISABLE COUNTER
                        edge = 0;
                        count = TIM2->CNT; //Put TIM2 Counter value into count
                        frequency = (float)SystemCoreClock / (float)count; // calculate frequency
                        Frequency = (int) frequency;
                        }
        EXTI->PR |= EXTI_PR_PR1;}
}
void lcd_Init()
{       GPIOB->BRR = ((uint16_t) 0xFFF0);
        lcd_cmd((uint16_t)0x38);//DL = 1, N = 1, F = 0  DDRAM using 8bits,2 lines of 8 characters
```

```
	lcd_cmd((uint16_t)0xC);//(D = 1, C = 0, B = 0 display on, no cursor, no blinking
	lcd_cmd((uint16_t)0x6);//I/D = 1, S = 0   DDRAM address is auto-incremented after each access no shift
	lcd_cmd((uint16_t)0x1);//clear display
}

void lcd_cmd(uint16_t cmd){
	uint16_t temp_port;
	GPIOB->BRR = GPIO_Pin_5;//Set RS to 0
	GPIOB->BRR = GPIO_Pin_6;//Set RW to 0
	temp_port = GPIOB->ODR &~((uint16_t) 0xFF00);//Assign port B to a temporary port, clear 15:8.
	GPIOB->ODR = temp_port|((uint16_t)cmd<<8);  //put data on output Port
	GPIOB->BSRR = GPIO_Pin_4;//Set bit 4 to 1 ENB
	wait(20);
	while(((GPIOB->IDR)&GPIO_Pin_7) == 0){};//Wait for input from LCD Pin 7
	GPIOB->BRR |= GPIO_Pin_4;//Set bit 4 to 0 ENB
	while(((GPIOB->IDR)&GPIO_Pin_7) != 0){};//wait for input from LCD Pin 7
	wait(50);
}

void lcd_send(char a){
	uint16_t temp_port;
	GPIOB->BSRR = GPIO_Pin_5;//Set RS to 1
	GPIOB->BRR = GPIO_Pin_6;//Set RW to 0
	temp_port = GPIOB->ODR & ~((uint16_t) 0xFF00);//Assign port B to a temporary port, clear 15:8.
	GPIOB->ODR = temp_port|((uint16_t)a<<8);  //put data on output Port
	GPIOB->BSRR = GPIO_Pin_4;//Set bit 4 to 0 ENB
	while(((GPIOB->IDR)& GPIO_Pin_7) == 0){};//Wait for input from LCD Pin 7
```

```
	GPIOB->BRR = GPIO_Pin_4;//Set bit 4 to 1 ENB
	while(((GPIOB->IDR)& GPIO_Pin_7) != 0){};//wait for input from LCD Pin 7
	wait(50);
	}

void lcd_write(int freq, int res){
	uint8_t digit = 0;
	int temp_freq = freq;
	int i = 0;
	int temp_res = res;
	lcd_cmd(0x0080);//send cursor to fist position
	lcd_send('F');//Print F
	lcd_send(':');//Print :

	for(i=3 ;i>=0 ;i--){ //print out the value for frequency on LCD
		temp_freq = freq/pow(10,i);
		digit = 48 + temp_freq%10;
		lcd_send((uint8_t)digit);//print digit value
	}
	lcd_send('H');//Print H
	lcd_send('z');//Print z

	lcd_cmd(0x00C0);//Send cursor to second position
	lcd_send('R');    //Print R
	lcd_send(':');//Print :
	for(i=3 ;i>=0 ;i--){// print the value of resistance on LCD
		temp_res = res/pow(10,i);
		digit = 48+temp_res%10;
		lcd_send((uint8_t) digit);
```

```
                    }
            lcd_send('O');//Print o
            lcd_send('h');// Print h
            lcd_send('m');// Print m
}

#pragma GCC diagnostic pop
// ----------------------------------------------------------------------------
```