CENG 355 Microprocessor-Based Systems Lab Project

Tayler Mulligan V00819591
Raymond Bamford V00000000

Nov 24, 2016

Contents

List of Figures

Abbreviations

ADC analog-to-digital converter. i, iv, 1–3

CPHA clock phase. *see:* SPI

CPOL clock polarity. *see:* SPI

CR control register. 3

CRC cyclic redundancy check. 5

DAC digital-to-analog converter. i, 1, 3

GPIO general purpose input output. 2

GPIOx GPIO port x. 1, *see:* GPIO

IC integrated circuit. 1

LCD liquid crystal display. i, iii, 1, 3–6, *Glossary:* liquid crystal display

LCK lock. 4

MOSI master output, slave input. 4, *see:* SPI

MSB most significant bit. 5

SCK serial clock. 4, *see:* SPI

SoC system on a chip. iv

SPI serial peripheral interface. 1, 4, 5

TIM3 general purpose timer 3. 4, 5

UI user interface. 6

Glossary

HSI14

The 14 high speed interconnect clock supplying the ADC's clock. 3

liquid crystal display

a visual display utilizing a layer of liquid crystal between two electrodes that become opaque when a voltage is applied. i, iii

STM32F0DISCOVERY

A system on a chip (SoC) developed by STMicroelectronics providing a development board based on a Cortex-M0 microprocessor. 1, 2

1 Problem description

The purpose of this lab is to design an embedded system to create an open-loop control system, where the position of a potentiometer is measured and a frequency dependent on the measured resistance, with the frequency produced through a 555 timer-based circuit. Systems and components in this lab include embedded programming of an STM32F0 Discovery board, electrical circuit design of the variable-frequency 555 timer-based astable circuit, digital-to-analog and analog-to-digital converters, and serial peripheral interface (SPI) for inter-integrated circuit (IC) communication between the microprocessor and the LCD board; the system was built on an STM32F0DISCOVERY ARM platform.

2 Design Solution

2.1 Planning

2.1.1 Work Partitioning

Based on experience from the first lab, and personal preference, lab work was partitioned accordingly. The project was split into the embedded programming and accessory circuit design/implementation, where the majority of the embedded programming was assigned to Tayler Mulligan and the accessory timer circuit to Raymond Bamford. The partitioning was not strict, with members collaborating where necessary or convenient.

2.1.2 Technique and Technologies

Git and GithHub were utilized for the project to provide team access and syncing between lab computers (see `https://github.com/tamul/ceng355-lab-project`). The source code was split into three files: `main.c` (see A.1), containing the main program; `analog.c` (see A.2), containing ADC, DAC, and frequency monitoring code; and `lcd.c` (see A.3), containing code related to the LCD; each with corresponding header files.

2.2 Implementation

The main program called functions provided by `analog.h` and `lcd.h` to sequentially initialize each component. Components were initialized in the order of: the ADC, the DAC, the LCD, and the frequency monitor.

Listing 1: Initialization order

```
trace_printf("%s", "Initializing ADC...");
adc_init();
adc_enable_pot(1);
trace_puts("Done");

trace_printf("%s", "Initializing DAC...");
dac_init();
```

```
  trace_puts("Done");

  trace_printf("%s", "Initializing LCD...");
  lcd_init();
  trace_puts("Done");

  trace_printf("%s", "Initializing frequency monitor...");
  freq_init();
  trace_puts("Done");
```

2.2.1 ADC Initialization

Initialization of the glsadc requires initialization of the general purpose input output (GPIO) (port C) interface (to control the POT_EN signal of the Project Board), GPIOC (to read the potentiometer value), and the glsadc. The C code in Listing 2 comprises the initialization of the GPIOC register for the glsadc.

Listing 2: GPIO configuration for ADC

```
        /* Enable clock for GPIOC */
        RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

        /* Configure PC1 as push-pull output  */
        GPIOC->MODER |= GPIO_MODER_MODER1_0;
        GPIOC->OTYPER &= ~GPIO_OTYPER_OT_1;
        /* Ensure high-speed mode for PC1 */
        GPIOC->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR1;
        /* Disable any pull up/down resistors on PC1 */
        GPIOC->PUPDR &= ~GPIO_PUPDR_PUPDR1;

        /* Configure PA0 as an analog pin */
        GPIOA->MODER |= GPIO_MODER_MODER0;
```

Firstly, the GPIOC clock is ensured to be running, followed by configuration of the pins. The pins are put in a push-pull output configuration at the highest speed, without any pull-up or pull-down resistors. The GPIOA clock is guaranteed to be running as the communication between the computer and STM32F0DISCOVERY are over the parallel A ports. The PA0 pin is set to “analog” mode.

Listing 3: ADC configuration

```
        /* Enable the HSI14 (ADC async) clock */
        RCC->CR |= RCC_CR_HSION;
        RCC->APB2ENR |= RCC_APB2ENR_ADCEN;

        /* Start up the ADC */
        ADC1->CR = ADC_CR_ADEN;

        /* ---- Configure the ADC ---- */
        /* Set the ADC clock to the dedicated clock */
        ADC1->CFGR2 &= ~(0x00);
        /* Select the input channel */
        ADC1->CHSELR = ADC_CHSELR_CHSEL0;
        /* Set continuous conversion mode */
        ADC1->CFGR1 |= ADC_CFGR1_CONT;
        /* Enable starting with software trigger */
65      ADC1->CFGR1 &= ~ADC_CFGR1_EXTEN;
        /* Disable auto-off */
        ADC1->CFGR1 &= ~ADC_CFGR1_AUTOFF;
        /* ----------------------------*/

70      /* Wait for the ADC to stabilize */
        while(!(ADC1->ISR & ADC_ISR_ADRDY));
        /* Start converting the analog input */
        ADC1->CR |= ADC_CR_ADSTART;
```

Next the ADC proper is initialized: the C code in Listing 3 accomplishes this. The HSI14 clock, which supplies the ADC's clock, is enabled and the glsadc started in preparation for configuration. Lines 57-68 configure the glsadc: setting the clock as the dedicated (HSI14) clock, selecting input channel 0 (corresponding to parallel port A0), continuous conversion mode is enabled to continuously provide the digitized value of the PA0 pin. Finally, the function triggers conversion to start after waiting until the glsadc reports that it has stabilized.

2.2.2 DAC Initialization

Enabling the DAC requires configuration of the PA4 pin and initialization of the DAC clock. Listing 4 shows the C code initializing the DAC.

Listing 4: DAC configuration

```
void dac_init(void) {
        /* Set PA4 (DAC output) as analog output pin */
80      GPIOA->MODER = GPIO_MODER_MODER4;
        /* Set PA4 to open drain */
        GPIOA->OTYPER = GPIO_OTYPER_OT_4;
        /* Disable pull up/down resistors on PA4 */
        GPIOA->PUPDR &= ~GPIO_PUPDR_PUPDR4;
85      /* Set PA4 to high-speed mode */
        GPIOA->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR4;

        /* Enable the DAC clock */
        RCC->APB1ENR |= RCC_APB1ENR_DACEN;
90      /* Enable the DAC (with output buffering and auto-
    triggering */
        DAC->CR = DAC_CR_EN1;
}
```

The mode of PA4 is set to an open-drain analog output, allowing a variable voltage to be placed on PA4, with pull-up and pull-down pins disabled, at the highest speed setting to ensure the pin updates quickly. Next, the DAC's clock is enabled and the DAC enabled by writing the DAC's control register (CR)'s enable bit.

2.2.3 LCD Initialization

To initialize the LCD, the spi, GPIOB, and GPIOC clocks are all initialized (Listing 5).

Listing 5: SPI, GPIOB, GPIOC clock enable

```
void lcd_init(void) {
        /* Enable SPI1 clock */
        RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
        /* Enable GPIOB clock */
        RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
        /* Enable GPIOC clock */
        RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
```

The lock (LCK) pin (PC2) of the SPI shift-register is set to output mode, and the master output, slave input (MOSI) (PB5) and serial clock (SCK) (PB3) SPI pins are set to alternate function mode (Listing 6).

Listing 6: SPI pin mode configuration

```
        /* Set PC2 to output */
        GPIOC->MODER = (GPIOC->MODER & ~GPIO_MODER_MODER2) | (
    GPIO_MODER_MODER2 & GPIO_MODER_MODER2_0);
        /* Set PB3 to alternate function */
        GPIOB->MODER = (GPIOB->MODER & ~GPIO_MODER_MODER3) | (
    GPIO_MODER_MODER3 & GPIO_MODER_MODER3_1);
        GPIOB->AFR[0] &= ~GPIO_AFRL_AFR3;
        /* Set PB5 to alternate function */
        GPIOB->MODER = (GPIOB->MODER & ~GPIO_MODER_MODER5) | (
    GPIO_MODER_MODER5 & GPIO_MODER_MODER5_1);
        GPIOB->AFR[0] &= ~GPIO_AFRL_AFR5;
```

Each SPI related pin is set to push-pull mode with pull-up and pull-down resistors disabled and high-speed mode (Listing 7).

Listing 7: SPI pin output configuration

```
        /* Set PC2, PB3, PB5 to push-pull mode */
        GPIOC->OTYPER &= ~GPIO_OTYPER_OT_2;
        GPIOB->OTYPER &= ~GPIO_OTYPER_OT_3;
        GPIOB->OTYPER &= ~GPIO_OTYPER_OT_5;

        /* Disable pull up/down resistors on SPI pins */
        GPIOB->PUPDR &= ~GPIO_PUPDR_PUPDR3;
        GPIOB->PUPDR &= ~GPIO_PUPDR_PUPDR5;
        GPIOC->PUPDR &= ~GPIO_PUPDR_PUPDR2;

        /* Set SPI pins to high-speed */
        GPIOB->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR3;
        GPIOB->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR5;
        GPIOC->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR2;
```

After configuring the output pins, general purpose timer 3 (TIM3) is configured, allowing SPI writes to be delayed and the LCD time to complete the commands sent:

Listing 8: TIM3 configuration

```
        /* Initialize TIM3 */
        /* Enable the TIM3 clock */
        RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
        /* Configure TIM3 with buffer auto-reload, count down,
         * stop on overflow, and only interrupt on overflow
         */
        TIM3->CR1 = 0xC6;
        /* Set clock prescaler value */
        TIM3->PSC = TIM3_PRESCALER;
        /* Set auto-reload delay */
        TIM3->ARR = TIM3_AUTORELOAD_DELAY;
        /* Set timer update configuration (rising edge, etc.) */
        TIM3->EGR = (TIM2->EGR & ~0x5F) | (0x1 & 0x5F);
        /* Load delay value */
        TIM3->CNT = MAX_DELAY;
        TIM3->CR1 |= 0x1;
```

The clock for TIM3 is enabled, then the clock is configured to use buffered auto-reload, count down from the set `CNT` register value, stop in the event of an overflow (which should never occur), and only interrupt in the event of an overflow not when the counter reaches 0. This configuration allows a value to be written to the timer which then counts down to 0, and the timer can be polled until the count is low enough (and enough time has passed), then allowing another command to be sent. This avoids the potential for an overflow if the counter was initialized to zero and counted up to or past the desired value.

The prescaler is set to 0, the auto-reload delay set to the lowest possible, writing to the `EGR` register is a carry-over from other timer initializations and is not necessary in this case; then, the `MAX\_DELAY` value (the maximum time the LCD will take to execute a command) is loaded into the timer's `CNT` register, and the timer finally started.

Next, in Listing 9, SPI is configured and enabled with: unidirectional transmit, master mode, a data-size of 8 bits, a falling edge clock pulse, software chip select, a 256 baud-rate prescaler, most significant bit (MSB) first output, and a 7 bit cyclic redundancy check (CRC) polynomial.

Listing 9: SPI initialization

```
        /* Initialize SPI */
        SPI_InitTypeDef SPI_InitStructInfo = {
                        .SPI_Direction = SPI_Direction_1Line_Tx,
                        .SPI_Mode = SPI_Mode_Master,
                        .SPI_DataSize = SPI_DataSize_8b,
                        .SPI_CPOL = SPI_CPOL_Low,
                        .SPI_CPHA = SPI_CPHA_1Edge,
                        .SPI_NSS = SPI_NSS_Soft,
                        .SPI_BaudRatePrescaler =
    SPI_BaudRatePrescaler_256,
                        .SPI_FirstBit = SPI_FirstBit_MSB,
                        .SPI_CRCPolynomial = 7
        };
        SPI_Init(SPI1, &SPI_InitStructInfo);

        SPI_Cmd(SPI1, ENABLE);
```

Finally, in Listing 10, the command interface and the display of the LCD are initialized: the LCD is set to a 4-bit interface with 2 lines, followed by the LCD being cleared and homed with display shift disabled and cursor movement direction set to the right, cursor blinking being disabled. The persistent, unchanging characters of the user interface (UI) are then written to the LCD.

Listing 10: LCD initialization

```
        /* Set the LCD to 4 bit interface */
        spi_write(0x02, MAX_DELAY);
        spi_write(0x82, MAX_DELAY);
        spi_write(0x02, MAX_DELAY);

        /* Set LCD to display 2 lines */
        lcd_cmd(0x28);

        /* Clear the LCD */
        lcd_cmd(0x01);
        /* Home the cursor */
        lcd_cmd(0x02);
        /* Set cursor move direction and disable display shift */
        lcd_cmd(0x06);
        /* Set the display on, don't show the cursor, don't blink
    */
        lcd_cmd(0x0C);
        /* Set the cursor to move right, no display shift */
        lcd_cmd(0x14);
        /* Write 'F:    Hz' to first line */
        lcd_cmd(0x80); // Start at very left

        lcd_char('F'); //F
        lcd_char(':'); //:
        lcd_cmd(0x86);
        lcd_char('H'); //H
        lcd_char('z'); //z

        /* Write 'R:    Oh' to second line*/
        lcd_cmd(0xC0); // set address to second line, first
    character
        lcd_char('R'); //R
        lcd_char(':'); //:
        lcd_cmd(0xC6);
        lcd_char('O');
        lcd_char('h');
```

3 Discussion

References

Appendices

A Source Code

A.1

src/main.c

```c
//
// This file is part of the GNU ARM Eclipse distribution.
// Copyright (c) 2014 Liviu Ionescu.
//

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

#include <stdio.h>
#include <stdlib.h>
#include "diag/Trace.h"

#include "stm32f0xx_conf.h"

#include "analog.h"
#include "lcd.h"


// 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[]) {
  // By customizing __initialize_args() it is possible to pass arguments,
  // for example when running tests with semihosting you can pass various
  // options to the test.
  // trace_dump_args(argc, argv);

  // Send a greeting to the trace device (skipped on Release).
  trace_puts("Hello ARM World!");

  // The standard output and the standard error should be forwarded to
  // the trace device. For this to work, a redirection in _write.c is
  // required.

  // At this stage the system clock should have already been configured
  // at high speed.
  trace_printf("System clock: %u Hz\n", SystemCoreClock);

  trace_printf("%s", "Initializing ADC...");
  adc_init();
  adc_enable_pot(1);
  trace_puts("Done");

  trace_printf("%s", "Initializing DAC...");
  dac_init();
  trace_puts("Done");

  trace_printf("%s", "Initializing LCD...");
  lcd_init();
  trace_puts("Done");

  trace_printf("%s", "Initializing frequency monitor...");
  freq_init();
  trace_puts("Done");

  // Infinite loop, wait for interrupts to do anything
  while (1) {

  }
  // Infinite loop, never return.
  return 0;
}

#pragma GCC diagnostic pop

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

A.2

include/analog.h

```
#ifndef __ANALOG_H__
#define __ANALOG_H__

#include <stdio.h>
#include <stdlib.h>

#include "stm32f0xx_conf.h"

/* No prescaler on timer 2 */
#define TIM2_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for auto-reload */
#define TIM2_AUTORELOAD_DELAY ((uint32_t)0xFFFFFFFF)
/* 48MHz clock speed */
#define TIMER_CLOCK_FREQ ((uint32_t)48000000)

/* Wiring:
 *      Potentiometer/ADC:
 *              - ADC input POT M20 -> PA0
 *              - POT_EN M32 -> PC1
 *      DAC:
 *              - DAC output is PA4
 *      Freq. Measurement:
 *              - Input is PA1
 */

void adc_init(void);
void dac_init(void);
void freq_init(void);

/* Enable or disable the potentiometer value line
 * Inputs:
 *      state: 1 for enable 0 for disable
 */
void adc_enable_pot(uint8_t state);

/* Read the current value for the potentiometer
 * If POT ENABLE is not high this will return garbage results
 * Outputs:
 *      uint16_t: right-aligned 12-bit ADC value
 */
uint16_t adc_read(void);

/* Write the output value of the DAC
 * Inputs:
 *      uint16_t value: 12-bit right-aligned value to set
 */
void dac_write(uint16_t value);

/* Returns the current input frequency
 * Outputs:
 *      float: current frequency in Hz
 */
uint32_t period_to_freq(uint32_t count);
```

```c
#endif // __ANALOG_H__
```

src/analog.c

```c
#include "analog.h"

#include "diag/Trace.h"
#include "lcd.h"

#define UPDATE_DELAY (800000)

static uint16_t first_edge = 1;
static uint16_t adc_value;

uint16_t adc_read(void) {
        return ADC1->DR;
}

void adc_enable_pot(uint8_t state) {
        if (state) {
                GPIOC->BSRR = GPIO_BSRR_BS_1;
        }
        else {
                GPIOC->BRR = GPIO_BRR_BR_1;
        }
}

void dac_write(uint16_t value) {
        /* Write the provided value to the 12-bit right-aligned DAC
     input */
        DAC->DHR12R1 = (value & DAC_DHR12R1_DACC1DHR);
}

uint32_t period_to_freq(uint32_t count) {
        return (TIMER_CLOCK_FREQ)/count;
}

/* Initialize the ADC
 */
void adc_init(void) {
        /* Enable clock for GPIOC */
        RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

        /* Configure PC1 as push-pull output  */
        GPIOC->MODER |= GPIO_MODER_MODER1_0;
        GPIOC->OTYPER &= ~GPIO_OTYPER_OT_1;
        /* Ensure high-speed mode for PC1 */
        GPIOC->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR1;
        /* Disable any pull up/down resistors on PC1 */
        GPIOC->PUPDR &= ~GPIO_PUPDR_PUPDR1;

        /* Configure PA0 as an analog pin */
        GPIOA->MODER |= GPIO_MODER_MODER0;

        /* Enable the HSI14 (ADC async) clock */
        RCC->CR |= RCC_CR_HSION;
        RCC->APB2ENR |= RCC_APB2ENR_ADCEN;
        /* Start up the ADC */
        ADC1->CR = ADC_CR_ADEN;

        /* ---- Configure the ADC ---- */
        /* Set the ADC clock to the dedicated clock */
        ADC1->CFGR2 &= ~(0x00);
        /* Select the input channel */
        ADC1->CHSELR = ADC_CHSELR_CHSEL0;
        /* Set continuous conversion mode */
        ADC1->CFGR1 |= ADC_CFGR1_CONT;
        /* Enable starting with software trigger */
        ADC1->CFGR1 &= ~ADC_CFGR1_EXTEN;
        /* Disable auto-off */
        ADC1->CFGR1 &= ~ADC_CFGR1_AUTOFF;
        /* ---------------------------*/

        /* Wait for the ADC to stabilize */
        while(!(ADC1->ISR & ADC_ISR_ADRDY));
        /* Start converting the analog input */
        ADC1->CR |= ADC_CR_ADSTART;
}

/* Initialize the DAC
 */
void dac_init(void) {
        /* Set PA4 (DAC output) as analog output pin */
        GPIOA->MODER = GPIO_MODER_MODER4;
        /* Set PA4 to open drain */
        GPIOA->OTYPER = GPIO_OTYPER_OT_4;
        /* Disable pull up/down resistors on PA4 */
        GPIOA->PUPDR &= ~GPIO_PUPDR_PUPDR4;
        /* Set PA4 to high-speed mode */
        GPIOA->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR4;

        /* Enable the DAC clock */
        RCC->APB1ENR |= RCC_APB1ENR_DACEN;
        /* Enable the DAC (with output buffering and auto-
    triggering */
        DAC->CR = DAC_CR_EN1;
}

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

        /* Enable the TIM2 clock */
        RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
        /* Configure TIM2 with buffer auto-reload, count up,
         * stop on overflow, enable update events, and interrupt
         * on overflow only
         */
        TIM2->CR1 = 0x8C;
```

```c
        /* Set clock prescaler value */
        TIM2->PSC = TIM2_PRESCALER;
        /* Set auto-reload delay */
        TIM2->ARR = TIM2_AUTORELOAD_DELAY;
        /* Set timer update configuration (rising edge, etc.) */
        TIM2->EGR = (TIM2->EGR & ~0x5F) | (0x1 & 0x5F);

        /* Assign TIM2 interrupt priority 0 in NVIC */
        NVIC_SetPriority(TIM2_IRQn, 0);
        /* Enable TIM2 interrupts in NVIC */
        NVIC_EnableIRQ(TIM2_IRQn);
        /* Enable update interrupt generation */
        TIM2->DIER |= 0x41;

        /* Map EXTI1 line to PA1 */
        SYSCFG->EXTICR[0] = (SYSCFG->EXTICR[0] & ~(0xF)) | (0 & 0xF
    );

        /* EXTI1 line interrupts: set rising-edge trigger */
        EXTI->RTSR |= 0x2;

        /* Unmask interrupts from EXTI1 line */
        EXTI->IMR |= 0x2;

        /* Assign EXTI1 interrupt priority 0 in NVIC */
        NVIC_SetPriority(EXTI0_1_IRQn, 0);
        /* Enable EXTI1 interrupts in the NVIC */
        NVIC_EnableIRQ(EXTI0_1_IRQn);
}

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

                /* Clear update interrupt flag */
                TIM2->SR |= 0x1;
                /* Restart stopped timer */
                TIM2->CR1 |= 0x1;
        }
}

void EXTI0_1_IRQHandler() {
        /* Check if EXTI1 interrupt pending flag is set */
        // Disable interrupts while servicing
        NVIC_DisableIRQ(EXTI0_1_IRQn);
        if (EXTI->PR & EXTI_PR_PR1) {
                if (first_edge) {
                        first_edge = 0;
                        /* Reset current timer count */
                        TIM2->CNT = 0;
                        /* Start the timer */
                        TIM2->CR1 |= 0x1;
                } else {
                        /* Stop the timer */
                        TIM2->CR1 &= ~0x1;
                        /* Read the current timer count */
                        uint32_t count = TIM2->CNT;

                        uint32_t freq_value = period_to_freq(count)
    ;
                        char* freq_ascii = num_to_ascii(freq_value)
    ;
                        // Write the frequency value to the LCD
                        lcd_cmd(0x82); // Set address to 02
                        for (int i = MAX_DIGITS-1; i >= 0; i--){
                                lcd_char(*(freq_ascii + i));
                        }
                        // Here we want to obtain the resistance,
    send the result to the DAC
                        // and print it to the LCD. A short wait
    time will also be added so the display doesn't flicker too much
                        adc_value = adc_read();
                        // Get the string for resistance rounded to
     the 100th
                        dac_write(adc_value); // Send value of ADC
    to DAC
                        char* resistance_ascii = num_to_ascii((((
    uint32_t)(adc_value * ((float)5000/4095)) + 50)/ 100)*100);
                        // Write the resistance value to the LCD
                        lcd_cmd(0xC2); // Set address to h42
                        for (int i = MAX_DIGITS-1; i >= 0; i--) {
                                lcd_char(*(resistance_ascii + i));
                        }

                        TIM3->CNT = UPDATE_DELAY;
                        TIM3->CR1 |= 0x1;
                        while(TIM3->CNT > 1);
                        first_edge = 1;
                }

                /* Clear the interrupt flag */
                EXTI->PR = EXTI_PR_PR1;
        }

        NVIC_EnableIRQ(EXTI0_1_IRQn);
}
```

A.3

include/lcd.h

```c
#ifndef __LCD_H__
#define __LCD_H__

#include "stm32f0xx_conf.h"

#define MAX_DIGITS (4)

/* Maximum required delay between SPI writes */
#define MAX_DELAY ((uint32_t)96000)
/* Delay between characters */
#define CHAR_DELAY ((uint32_t)4800)

/** Wiring
 * PC2 - LCK -> M25
 * PB5 - MOSI -> M17
 * PB3 - SCK -> M21
 */

void lcd_init(void);

void lcd_clear(void);

void lcd_cmd(uint8_t data);

void lcd_char(char c);

char* num_to_ascii(uint32_t num);

#endif //__LCD_H__
```

src/lcd.c

```c
#include "lcd.h"

#define LCD_BAUD_RATE_PRESCALER (16)

/* No prescaler on timer 3 */
#define TIM3_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for auto-reload */
#define TIM3_AUTORELOAD_DELAY ((uint32_t)0xFFFFFFFF)
/* 48MHz clock speed */
#define TIMER_CLOCK_FREQ ((uint32_t)48000000)

/* Output data to the shift register through SPI */
void spi_write(uint8_t, uint32_t delay);

void lcd_init(void) {
        /* Enable SPI1 clock */
        RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
        /* Enable GPIOB clock */
        RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
        /* Enable GPIOC clock */
        RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
```

```
    /* Set PC2 to output */
    GPIOC->MODER = (GPIOC->MODER & ~GPIO_MODER_MODER2) | (
GPIO_MODER_MODER2 & GPIO_MODER_MODER2_0);
    /* Set PB3 to alternate function */
    GPIOB->MODER = (GPIOB->MODER & ~GPIO_MODER_MODER3) | (
GPIO_MODER_MODER3 & GPIO_MODER_MODER3_1);
    GPIOB->AFR[0] &= ~GPIO_AFRL_AFR3;
    /* Set PB5 to alternate function */
    GPIOB->MODER = (GPIOB->MODER & ~GPIO_MODER_MODER5) | (
GPIO_MODER_MODER5 & GPIO_MODER_MODER5_1);
    GPIOB->AFR[0] &= ~GPIO_AFRL_AFR5;

    /* Set PC2, PB3, PB5 to push-pull mode */
    GPIOC->OTYPER &= ~GPIO_OTYPER_OT_2;
    GPIOB->OTYPER &= ~GPIO_OTYPER_OT_3;
    GPIOB->OTYPER &= ~GPIO_OTYPER_OT_5;

    /* Disable pull up/down resistors on SPI pins */
    GPIOB->PUPDR &= ~GPIO_PUPDR_PUPDR3;
    GPIOB->PUPDR &= ~GPIO_PUPDR_PUPDR5;
    GPIOC->PUPDR &= ~GPIO_PUPDR_PUPDR2;

    /* Set SPI pins to high-speed */
    GPIOB->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR3;
    GPIOB->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR5;
    GPIOC->OSPEEDR |= GPIO_OSPEEDR_OSPEEDR2;

    /* Initialize TIM3 */
    /* Enable the TIM3 clock */
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    /* Configure TIM3 with buffer auto-reload, count down,
     * stop on overflow, and only interrupt on overflow
     */
    TIM3->CR1 = 0xC6;
    /* Set clock prescaler value */
    TIM3->PSC = TIM3_PRESCALER;
    /* Set auto-reload delay */
    TIM3->ARR = TIM3_AUTORELOAD_DELAY;
    /* Set timer update configuration (rising edge, etc.) */
    TIM3->EGR = (TIM2->EGR & ~0x5F) | (0x1 & 0x5F);
    /* Load delay value */
    TIM3->CNT = MAX_DELAY;
    TIM3->CR1 |= 0x1;

    /* Initialize SPI */
    SPI_InitTypeDef SPI_InitStructInfo = {
                    .SPI_Direction = SPI_Direction_1Line_Tx,
                    .SPI_Mode = SPI_Mode_Master,
                    .SPI_DataSize = SPI_DataSize_8b,
                    .SPI_CPOL = SPI_CPOL_Low,
                    .SPI_CPHA = SPI_CPHA_1Edge,
                    .SPI_NSS = SPI_NSS_Soft,
                    .SPI_BaudRatePrescaler =
SPI_BaudRatePrescaler_256,
                    .SPI_FirstBit = SPI_FirstBit_MSB,
                    .SPI_CRCPolynomial = 7
```

```
        };
        SPI_Init(SPI1, &SPI_InitStructInfo);

        SPI_Cmd(SPI1, ENABLE);

        /* Set the LCD to 4 bit interface */
        spi_write(0x02, MAX_DELAY);
        spi_write(0x82, MAX_DELAY);
        spi_write(0x02, MAX_DELAY);

        /* Set LCD to display 2 lines */
        lcd_cmd(0x28);

        /* Clear the LCD */
        lcd_cmd(0x01);
        /* Home the cursor */
        lcd_cmd(0x02);
        /* Set cursor move direction and disable display shift */
        lcd_cmd(0x06);
        /* Set the display on, don't show the cursor, don't blink
    */
        lcd_cmd(0x0C);
        /* Set the cursor to move right, no display shift */
        lcd_cmd(0x14);
        /* Write 'F:    Hz' to first line */
        lcd_cmd(0x80); // Start at very left

        lcd_char('F'); //F
        lcd_char(':'); //:
        lcd_cmd(0x86);
        lcd_char('H'); //H
        lcd_char('z'); //z

        /* Write 'R:    Oh' to second line*/
        lcd_cmd(0xC0); // set address to second line, first
    character
        lcd_char('R'); //R
        lcd_char(':'); //:
        lcd_cmd(0xC6);
        lcd_char('O');
        lcd_char('h');
}

void spi_write(uint8_t data, uint32_t delay) {
        /* Wait until SPI delay has passed */
        while (TIM3->CNT > ((MAX_DELAY+1) - delay));
        /* Force LCK low */
        GPIOC->BRR |= GPIO_BRR_BR_2;
        /* Wait until SPI1 is ready */
        while((SPI1->SR & SPI_SR_BSY) && ~(SPI1->SR & SPI_SR_TXE));
        /* Send the data */
        SPI_SendData8(SPI1, data);
        /* Wait until SPI1 is not busy */
        while(SPI1->SR & SPI_SR_BSY);
        /* Force LCK signal to 1 */
        GPIOC->BSRR |= GPIO_BSRR_BS_2;
        /* Reset LCD comm clock */
        TIM3->CNT = MAX_DELAY;
        TIM3->CR1 |= 0x1;
}

/*
 * Write to LCD using 4 bit interface. Send 4 high bits
 * by pulsing EN, then do the same for 4 low bits
 */
void lcd_cmd(uint8_t data) {
        /* Send HIGH bits */
        spi_write(0x00 | (data >> 4), MAX_DELAY);
        spi_write(0x80 | (data >> 4), MAX_DELAY);
        spi_write(0x00 | (data >> 4), MAX_DELAY);
        /* Send LOW bits */
        spi_write(0x00 | (data & 0x0F), MAX_DELAY);
        spi_write(0x80 | (data & 0x0F), MAX_DELAY);
        spi_write(0x00 | (data & 0x0F), MAX_DELAY);

}

/** Write a character to the LCD
 * Inputs:
 *  c: Character to write
 */
void lcd_char(char c) {
        /* Send HIGH bits */
        spi_write(0x40 | ((uint8_t)c >> 4), CHAR_DELAY);
        spi_write(0xC0 | ((uint8_t)c >> 4), CHAR_DELAY);
        spi_write(0x40 | ((uint8_t)c >> 4), CHAR_DELAY);
        /* Send LOW bits */
        spi_write(0x40 | ((uint8_t)c & 0x0F), CHAR_DELAY);
        spi_write(0xC0 | ((uint8_t)c & 0x0F), CHAR_DELAY);
        spi_write(0x40 | ((uint8_t)c & 0x0F), CHAR_DELAY);
}

/** Convert a number to ASCII digits (max of 4)
 * Inputs:
 *  num: 32 bit unsigned integer to convert
 * Returns:
 *  ASCII encoded digits in LSD first order
 */
char* num_to_ascii(uint32_t num) {
        static char ascii[MAX_DIGITS] = {0, 0, 0, 0};
        uint8_t i = 0;
        // Get individual digits (in LSD order)
        do {
                ascii[i++] = ('0' + (char)(num % 10));
                num /= 10;
        } while (num && (i < MAX_DIGITS));
        // Fill remaining space with blanks
        for (; i < MAX_DIGITS; i++) {
                ascii[i] = ' ';
        }
        return ascii;
}
```