Project Report

ECE 355: Microprocessor-Based Systems

✓	Problem Description/Specifications:	(5)	_________
✓	Design/Solution	(15)	_________
✓	Testing/Results	(10)	_________
✓	Discussion	(15)	_________
✓	Code Design and Documentation:	(15)	_________
✓	Total	(60)	_________

Submitted by:
Douglas MacGregor V01008370
Kiana Higuchi V01024858
Lab Section: B01
Date: November 19th, 2025

Table of Contents

Problem Description/Specifications

The objective of this project was to build an embedded measurement system centered around an STM32 board with a Cortex-M0 chip. The system integrated several components: an SPI-driven OLED display, a digital-to-analog converter (DAC), an analog-to-digital converter (ADC), a user button, a function generator, and a NE555 timer/optocoupler circuit. The project required configuring each peripheral at the register level and ensuring that all components operated correctly both individually and as a unified system. In addition to interfacing with multiple peripherals, the project involved combining analog signal processing with digital timing measurements to create a complete real-time measurement platform.

Specifications Summary:

OLED
- Interacted through an SPI interface.
- Used to display the resistance and frequency measured by the system.
- Command/data modes through PB8 (CS#), PB9 (D/C#), PB11 (RES#).

ADC
- Sampled the analog input voltage from the potentiometer (POT) on PA1.
- The resulting 12-bit digital value was used to calculate the potentiometer's resistance.

DAC
- Output a voltage on PA4 proportional to the ADC reading.
- This voltage served as the input to the NE555 timer/optocoupler circuit.

User Button
- Connected to the external interrupt line EXTI0 (PA0).
- Used to toggle the OLED display between the function generator frequency and the NE555 timer frequency.

Function Generator
- Developed in lab 2.
- Set up to provide a constant square-wave input to PB2.
- Used in tandem with EXTI2 and TIM2 to measure the period between rising edges and compute the input frequency.

NE555 timer / 4N35 optocoupler
- Took an input voltage from the DAC on PA4 and produced a square wave output measured on PB3.
- The output frequency was limited to approximately 800–1300 Hz due to optocoupler LED saturation.

Design/Solution

This section describes the design methodology used to develop the system, including project organization, peripheral configuration, and the sequence of integration and testing steps. The project was implemented using a modular approach, with each hardware peripheral isolated into its own source and header files.

Design Steps at a High Level:

1 — Code and project organization
Code segments were divided so that each hardware component had its own .c and .h files. This strategy allowed the debug-friendly modularization of code, and allowed both team members to work on components concurrently.

2 — Private repository setup for collaborative work
A private Git repository to store the project code was created. Creating a private remote repo on GitHub allowed work to be completed on the project while outside of the lab, version control to be maintained, and the easy setup of the Eclipse workspace to be established on lab machines.

3 — Component initialization
An ‘init’ function for each component was created. Where necessary, additional helper functions were created to update internal values or refresh the display. All required functions were imported into main.

Comments:
Control register configurations were referenced from the ECE 355 lecture slides (Interface Examples and I/O Examples) and the STM32F0 Reference Manual. This modular approach made it straightforward to verify each subsystem independently before integrating them.

4 — Initial testing: POT, ADC, and DAC
Testing began with the POT and ADC in isolation, printing the measured resistance values to the host computer terminal. The DAC was then added so that adjusting the POT produced a corresponding change in the DAC’s analog output.

Comments:
A pitfall we encountered in this stage was ensuring the STM32 board and the larger board had a common ground.

5 — SPI interface testing
The next component tested was the SPI interface, where static values for frequency and resistance were sent to the OLED to verify that the display initialized correctly and rendered the values in the intended positions.

6 — **Function generator integration**
Function generator code from Lab Part 2 using TIM2 and EXTI2 was incorporated. This allowed the 3 sections(TIM2+EXTI2, SPI, DAC+ADC+POT) to come together. After confirming that frequency measurement on PB2 was working, the user button on PA0 (EXTI0) was added to toggle which input pin (PB2 or PB3) the system measured rising edges from.

7 — **Full system integration with simulated NE555 timer**
The completion of previous steps allowed the entire project to be visualized, and the final NE555 timer circuit component was simulated with a second function generator.

8 — **Final hardware integration**
Finally, the NE555 timer circuit was built according to the diagram provided in the ECE 355 slides and connected into the system in place of the second function generator.

Figures

Figure 1: System Diagram From Lab Manual

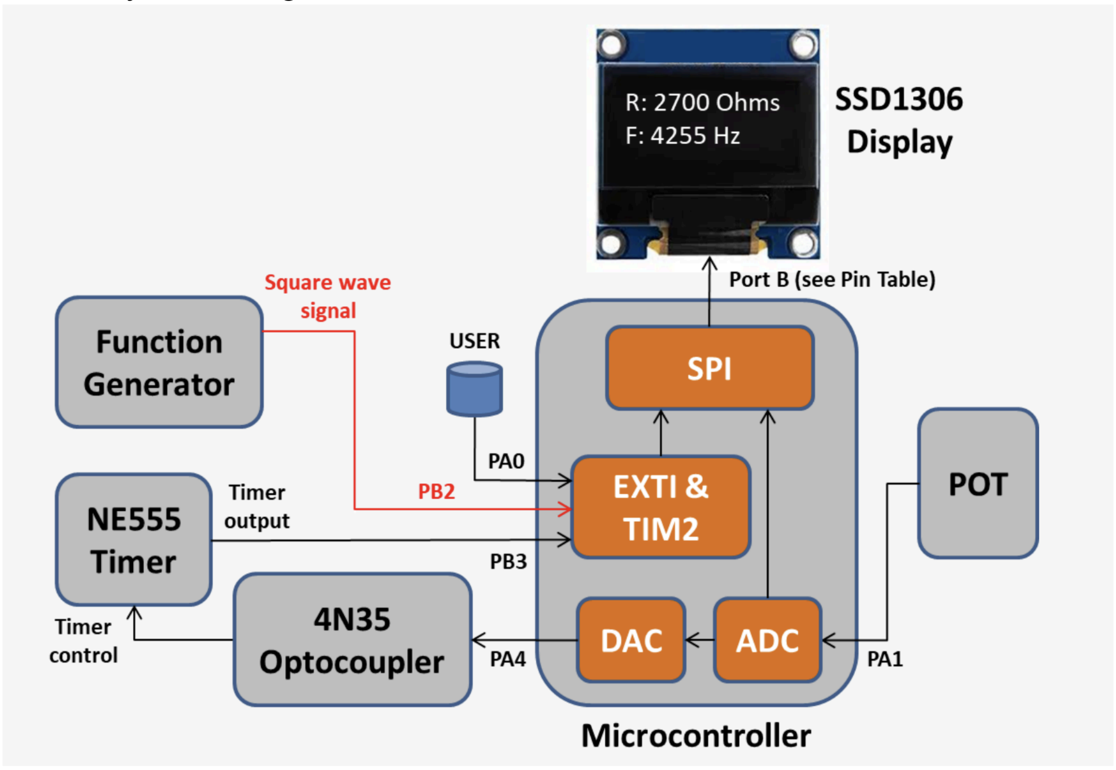
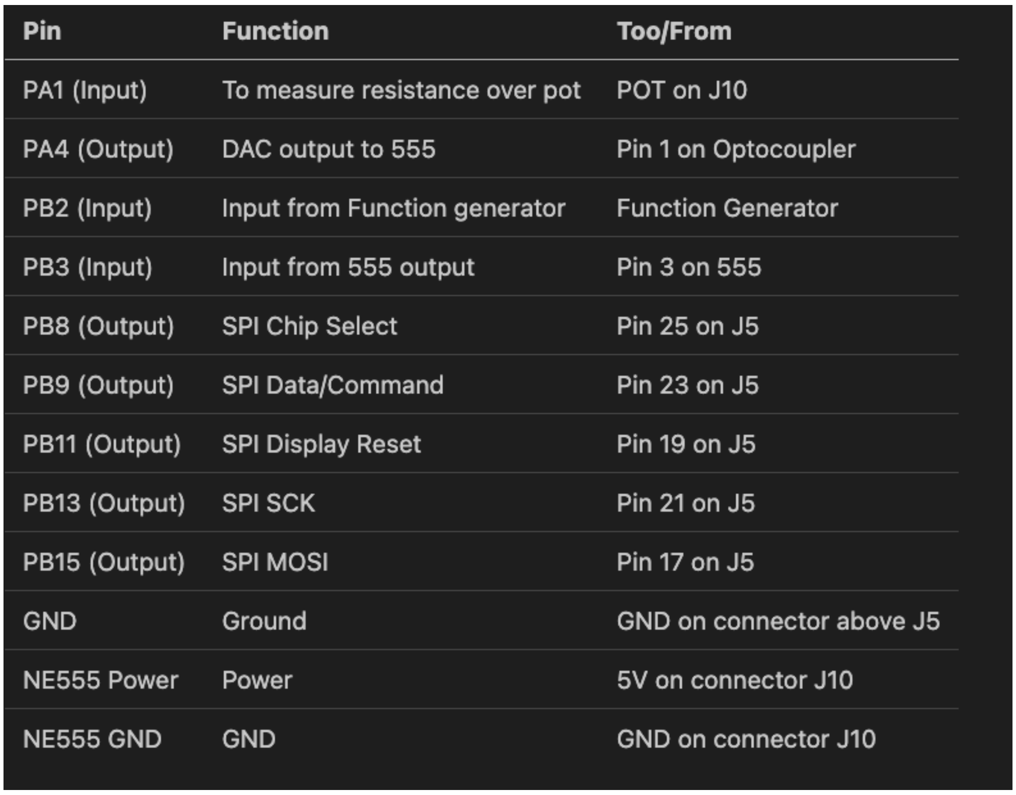
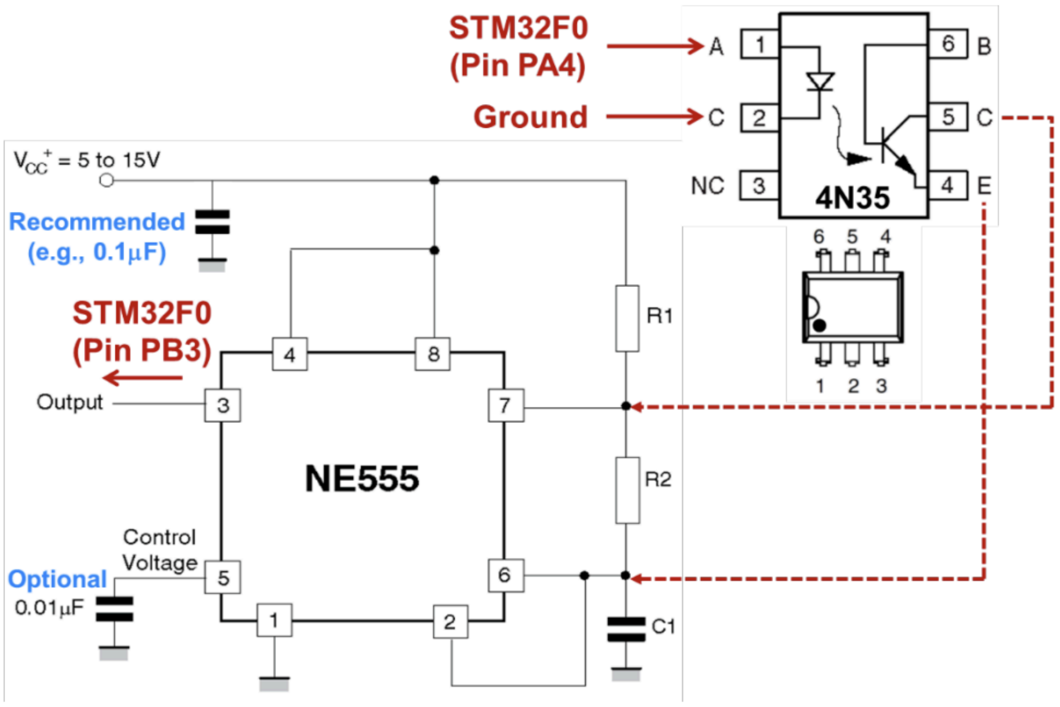


Figure 2: System Pin Mapping Table



Pin	Function	Too/From
PA1 (Input)	To measure resistance over pot	POT on J10
PA4 (Output)	DAC output to 555	Pin 1 on Optocoupler
PB2 (Input)	Input from Function generator	Function Generator
PB3 (Input)	Input from 555 output	Pin 3 on 555
PB8 (Output)	SPI Chip Select	Pin 25 on J5
PB9 (Output)	SPI Data/Command	Pin 23 on J5
PB11 (Output)	SPI Display Reset	Pin 19 on J5
PB13 (Output)	SPI SCK	Pin 21 on J5
PB15 (Output)	SPI MOSI	Pin 17 on J5
GND	Ground	GND on connector above J5
NE555 Power	Power	5V on connector J10
NE555 GND	GND	GND on connector J10

Figure 3: NE555 Timer Circuit Diagram From Lab Manual



Flags and Control Registers

ADC

Please see the documentation section for details on which bits correspond to each configuration.

1— Enable the clock for the ADC1 device.

2 — Put the ADC in continuous conversion mode so that it is continuously converting from the input source. Enable overrun management so that new data overwrites the old value in the data register. Set right-alignment and 12-bit resolution. All of these configurations are done through the ADC1->CFGR1 register.

3 — Select the correct channel whose input will be converted. This requires modifying bits in the ADC1->CHSELR register.

4 — Select the sampling interval for the ADC. This project used a sampling time of 1.5 ADC clock cycles. This is configured in the ADC1->SMPR register.

5 — Enable the ADC and wait for the 'ADC ready' flag before starting conversions. The ADC is enabled using the ADC1->CR register, and readiness is indicated by the ADC1->ISR register.

6 — Start the ADC conversion. Once the conversion-complete flag is set, the digital value can be read from ADC1->DR.

DAC

Please see the documentation section for details on which bits correspond to each configuration.

1 — Enable the clock for the DAC device.

2 — Disable the channel 1 trigger so that data written into the DAC_DHRx register is transferred one APB clock cycle later to the DAC_DOR1 register. Then enable the channel 1 output buffer, and finally enable the DAC channel 1. All of these configurations are done in the DAC->CR register.

3 — The DAC output value is set by writing a value into the lower 12 bits of the DAC->DHR12R1 register

SPI & TIM3

Please see the documentation section for details on which bits correspond to each configuration.

1 — Enable the peripheral clocks for SPI2 and TIM3 using the RCC->APB1ENR register

2 — Configure SPI2 in master, 1-line transmit-only mode with 8-bit data, clock polarity low, first-edge sampling, software-managed NSS, MSB-first transmission, and a relatively slow baud-rate prescaler. These fields are set through the SPI_Handle.Init structure and applied by calling HAL_SPI_Init, followed by __HAL_SPI_ENABLE to turn SPI2 on.

3 — Configure the GPIO pins used by the OLED interface. PB13 and PB15 are set to the alternate function for SPI2 SCK and MOSI, while PB8 (CS#), PB9 (D/C#), and PB11 (RES#) are configured as digital outputs to control the OLED.

4 — Configure TIM3 with a prescaler of 48000 − 1 so the counter increments every 1 ms. TIM3->CR1 is left at 0 so the timer remains disabled until a delay is needed.

5 — Use TIM3 to generate the ~100 ms delay during OLED refresh by writing TIM3->ARR = 100, resetting CNT, clearing UIF, enabling the counter, polling until UIF is set, and disabling the counter.

TIM2

Please see the documentation section for details on which bits correspond to each configuration.

1 — Enable the clock for the TIM2 device in the RCC->APB1ENR register..

2 — Set counter to stop counting at the next update event (bit3). Make it so that only overflow generates an update interrupt (bit2) and finally change clock division to $2 \times t_{CK-INT}$. These can be configured in the TIM2->CR1 register; ensure all other bits in this register are zeros.

3 — Set the prescaler value in TIM2->PSC to establish the timer tick frequency.

4 — Load the auto-reload value into TIM2->ARR so the timer can measure long periods between rising edges.

5 — Generate an update event by writing to the TIM2->EGR register, ensuring that the prescaler and auto-reload settings take effect.

6 — Assign a priority to TIM2_IRQn, enable the interrupt in the NVIC, and enable update interrupt generation by setting the UIE bit in TIM2->DIER.

EXTI0 and EXTI2 and EXTI3

Please see the documentation section for details on which bits correspond to each configuration.

1 — Clear and map each EXTI line to its corresponding pin using the SYSCFG_EXTICR registers: PA0 to EXTI0, PB2 to EXTI2, and PB3 to EXTI3.

2 — Configure rising-edge triggering for EXTI0, EXTI2, and EXTI3 using the EXTI->RTSR register.

3 — Unmask the interrupt lines for EXTI0, EXTI2, and EXTI3 in the EXTI->IMR register.

4 — Assign priorities for EXTI0_1_IRQn and EXTI2_3_IRQn in the NVIC, and then enable both interrupt groups.

5 — EXTI0_1_IRQHandler processes rising edges on PA0 and toggles the flag used to select the active frequency source. EXTI2_3_IRQHandler processes rising edges on PB2 or PB3 and performs the timing logic needed to compute the input frequency based on the selected source.

fivefivefive Flag

A flag that requires some additional explanation is the fivefivefive flag, defined in main.c and used most often in myEXTI.c. This flag keeps track of whether the system should be measuring the frequency from the NE555 timer or from the function generator. The flag is toggled by the interrupt service routine that monitors rising edges generated by the user button. When the flag is equal to 0, the system measures the frequency produced by the function generator; when the flag is equal to 1, the system instead measures the frequency from the NE555 timer circuit.

Testing/Results

Since the system was built in modules, each was tested to confirm correct register configuration, stability, and reliable interaction between hardware as it was integrated.

Modular Testing:

ADC and POT Testing

The ADC was first tested by sampling the potentiometer voltage on PA1 and printing the resulting 12-bit values (0-4095) to the terminal. The readings changed smoothly across the full range, which confirmed the correct sampling time, continuous conversion mode, and channel configuration. The raw ADC value was then scaled in software to a 0-5000 Ω equivalent resistance using R = (ADC_value x 5000) /(4095) Ω.

DAC Output Testing

The DAC output on PA4 was verified by observing its effect on the NE555 timer circuit. Adjusting the potentiometer resulted in:

- predictable changes in NE555 output frequency
- stable transitions
- correct mapping between ADC input and DAC output voltage

These observations confirmed that the DAC was correctly updating and producing the expected analog control voltage.

SPI and OLED Display Testing

SPI and OLED functionality was controlled using TIM3, and used several cohesive functions to interface and display desired values. It was built upon the skeleton laid out by the starter code on the lab website and debugged until it worked perfectly. The first test was successful SPI initialization, and further testing involved adjusting the page addressing and data placement incrementally until the displayed readings were accurate and correctly centered.

Frequency Measurement Testing (Function Generator)

The function generator was connected to PB2, and rising-edge interrupts (EXTI2) triggered period measurements using TIM2. The calculated frequency remained stable over repeated measurements and matched the generator dial settings. Low-frequency tests confirmed correct overflow handling, while higher-frequency tests showed that the system remained reliable until latency became a limiting factor.

NE555 Timer Testing

Before connecting the NE555 circuit to the system, the EXTI3 input on PB3 was tested independently using the function generator. This confirmed that rising-edge interrupts were detected correctly and that the PA0 user button successfully toggled the measurement source between PB2 and PB3. After assembling the physical NE555 + 4N35 optocoupler circuit according to the provided schematic, the output was connected to PB3 and measured using the same interrupt-driven method. The system consistently detected frequencies in the expected range and adjusting the potentiometer produced smooth, proportional changes in the measured value.

Results:

Measurement Range:

- Resistance via ADC: 0-5000 Ω (software-mapped range)
- Function generator frequency (PB2): ~35 Hz to 100 kHz usable range
- NE555 Output Frequency (PB3): ~800 to 1300 Hz, limited by optocoupler switching speed

Measurement Resolution:

- ADC: 12-bit, 4096 levels: ~1.22 mV voltage resolution, ~1.22 Ω resistance resolution
- Frequency: TIM2 driven by 48 MHz clock enables ~1 to 2 Hz resolution in practice due to integer division and edge timing

Measurement Errors:

- Small jitter at very low frequencies due to long periods and overflow dependence
- Slight nonlinearity at high NE555 frequencies due to 4N35 LED saturation
- Display rounding (integer formatting), minor presentational loss of precision

System Limitations:

- Optocoupler switching characteristics restrict the NE555 frequency range
- Interrupt latency limits very high frequency detection
- TIM3 polling for delays is simple but not real-time optimized
- No filtering or averaging applied to ADC readings

Design Limitations:

- OLED updates use blocking delays rather than interrupts
- Frequency measurement depends entirely on clean rising edges; noisy signals would require debouncing or filtering
- Only integer outputs are displayed on the OLED for readability
- Debouncing of the user button is imperfect; messy presses can cause unintended toggles

Discussion

The implemented system met the core design specifications: it measured a resistance value derived from the potentiometer, converted that value to an analog control voltage using the DAC, and reported the resulting frequency from either the function generator or the NE555/4N35 circuit on the OLED display. All required peripherals ADC, DAC, SPI, timers, external interrupts, and GPIO, were configured at the register level, and the final system operated as a unified measurement platform.

Assumptions:

A few key assumptions were made during the design. First, the potentiometer was assumed to behave linearly over its range, which justified the mapping that was implemented in the lab. Second, the input signals on PB2 and PB3 were assumed to be clean digital waveforms, with sharp rising edges. This allowed the use of rising edge EXTI interrupts and period measurement. Finally, the NE555/4N35 circuit was assumed to produce a frequency within the range that was suggested in the lab manual, with deviations attributed to component tolerances rather than software error.

Shortcomings:

Limitations of the system include the frequency ranges which are constrained by hardware and software choices causing unreliability at frequencies which are too low or too high. The NE555 frequency range is constrained by the 4N35 optocoupler, whose internal LED cannot switch too quickly, effectively limiting the usable output to a more narrow band. At the high end of the function generator range, interrupt latency and the overhead of starting and stopping TIM2 place practical limits on the maximum measurable frequency.

On the user interface side, the OLED refresh may not be ideal for a highly responsive or power-efficient system which limits its use. Additionally the user button is interpreted purely on rising edges. This could have been easily fixed through software debouncing, but at the time of completion, messy presses may have caused some sticking and bugginess of the display and frequency source. A difficulty that was experienced is incorrect wiring of the circuit which caused significant delays, since no amount of code debugging could have helped if the circuit was not properly connected. Thorough understanding of the connections and examination of each pin and wire was required to get it perfectly working.

Summary

Despite some limitations, several design choices worked well in practice. The modular separation of peripherals into dedicated .c/.h files made unit testing straightforward, and minimized side effects when changing configuration code. The method of switching between measuring the function generator and NE555 output at runtime was intuitive and efficient. Only accepting changes greater than a small threshold in frequency update logic improved display stability and prevented flickering of the display.

Overall, the lab reinforced several important lessons about embedded system design. Small configuration details in control registers have larger effects and it is hugely important to understand the logic behind bitwise masking and programming. The lab rewarded the use of modularity and the importance of organization and version control. By following good coding practice, debugging time was reduced and portability was increased. The lab also highlighted the importance of considering hardware limits. For future work, improvements could include adding explicit button debouncing, reducing ADC noise, and replacing blocking delays with interrupt-driven timing. Advice for future students would be to embrace the division of tasks, and to integrate print statements to constantly verify progress.

Appendix A: Code Design and Documentation

globals.h

```c
#include <stdint.h>

#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Maximum possible setting for overflow */
#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)

extern volatile int timerTriggered;
extern volatile int frequency;
extern volatile unsigned int resistance;
extern volatile int fivefivefive;
extern volatile uint32_t tim2Overflow;
extern volatile int lastFrequency;
```

main.c

```c
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
#include "myEXTI.h"
#include "myTIM2.h"
#include "globals.h"
#include "misc.h"
#include "myADC.h"
#include "myDAC.h"
#include "mySPI.h"

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"

void myGPIO_Init(void);

volatile int timerTriggered = 0;
volatile int frequency = 0;
volatile unsigned int resistance = 0;
volatile int fivefivefive = 0;
volatile uint32_t tim2Overflow = 0;
volatile int lastFrequency = 0;

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

   SystemClock48MHz();
   trace_printf("This is Part 2 of Introductory Lab...\n");
   trace_printf("System clock: %u Hz\n", SystemCoreClock);
   RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
   myGPIO_Init();  /* Initialize the GPIO pins */
   mySPI_Init(); /* Initialize SPI*/
   myTIM3_Init(); /* Initialize timer TIM3 */
   oled_Config(); /* sends init commands to OLED */
   trace_printf("OLED initialized.\n");
   myTIM2_Init(); /* Initialize timer TIM2 */
   myEXTI_Init(); /* Initialize EXTI */
   myADC_Init();  /* Initialize ADC */
   myDAC_Init();  /* Initialize DAC */

   while (1)
   {
       myADC_StartConversion();
       int freq_local = frequency;
       refresh_OLED(freq_local);
       myDAC_SetValue(resistance);
   }
   return 0;
}

void myGPIO_Init(void)
{
   // clock bit enable
   RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOCEN;
   // PA modes
   GPIOA->MODER &= ~(GPIO_MODER_MODER0); // PA0 user push button (input)
   GPIOA->MODER |= (GPIO_MODER_MODER1);  // PA1 ADC input (analog)
   GPIOA->MODER |= (GPIO_MODER_MODER4);  // PA4 DAC output (analog)
   // PB modes
   GPIOB->MODER &= ~(GPIO_MODER_MODER2); // PB2 function generator (input)
   GPIOB->MODER &= ~(GPIO_MODER_MODER3); // PB3 555 timer (input)
   GPIOB->MODER &= ~(3UL << (8 * 2));    // PB8 CS#("Chip Select") (output)
   GPIOB->MODER |= (1UL << (8 * 2));
   GPIOB->MODER &= ~(3UL << (9 * 2)); // PB9 D/C#("Data/Command") (output)
   GPIOB->MODER |= (1UL << (9 * 2));
   GPIOB->MODER &= ~(3UL << (11 * 2)); // PB11 RES#("Reset") (output)
   GPIOB->MODER |= (1UL << (11 * 2));
   GPIOB->MODER &= ~(3UL << (13 * 2)); // PB13 SCLK("Serial Clock") output (AF)
   GPIOB->MODER |= (2UL << (13 * 2));
   GPIOB->MODER &= ~(3UL << (15 * 2)); // PB15 SDIN("Serial Data") output (AF)
   GPIOB->MODER |= (2UL << (15 * 2));
   // PC modes
   GPIOC->MODER &= ~(3UL << (8 * 2)); // PC8 blue LED (output)
   GPIOC->MODER |= (1UL << (8 * 2));
   GPIOC->MODER &= ~(3UL << (9 * 2)); // PC9 green LED (output)
   GPIOC->MODER |= (1UL << (9 * 2));
   // alternate function setup
   GPIOB->AFR[1] &= ~(0xF << (4 * (13 - 8))); // PB13 SPI_SCK
   GPIOB->AFR[1] |= (0x0 << (4 * (13 - 8)));
   GPIOB->AFR[1] &= ~(0xF << (4 * (15 - 8))); // SPI_MOSI
   GPIOB->AFR[1] |= (0x0 << (4 * (15 - 8)));
}

#pragma GCC diagnostic pop
```

misc.c

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

void SystemClock48MHz(void)
{
    // Disable the PLL
    RCC->CR &= ~(RCC_CR_PLLON);
    // Wait for the PLL to unlock
    while ((RCC->CR & RCC_CR_PLLRDY) != 0);
    // Configure the PLL for 48-MHz system clock
    RCC->CFGR = 0x00280000;
    // Enable the PLL
    RCC->CR |= RCC_CR_PLLON;
    // Wait for the PLL to lock
    while ((RCC->CR & RCC_CR_PLLRDY) != RCC_CR_PLLRDY);
    // Switch the processor to the PLL clock source
    RCC->CFGR = (RCC->CFGR & (~RCC_CFGR_SW_Msk)) | RCC_CFGR_SW_PLL;
    // Update the system with the new clock frequency
    SystemCoreClockUpdate();
}
```

misc.h

```
void SystemClock48MHz(void);
```

myADC.c

```
#include "myADC.h"
#include "globals.h"
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"

void myADC_Init(void)
{
    /* Enable clock for ADC1 peripheral */
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
    /* Configure ADC1: 12-bit resolution, right data alignment, single conversion mode
*/
    ADC1->CFGR1 |= (0b11 << 12); /* Continuous conversion mode(bit13) and Overrun
management mode (bit12) */
    ADC1->CFGR1 &= ~(0b111000); /* Right alignment(bit5) and 12 bits resolution (bit3+4)
*/
    ADC1->CHSELR |= 0b10; /* ADC channel selection register */
    ADC1->SMPR |= 0b111; /* Sampling time selection -> 1.5 ADC clock cycles */
    ADC1->CR |= 0b1; /* start ADC conversion */
    while ((ADC1->ISR & 0b1) == 0); /* wait for ADC to ‘warm up’ */
}

void myADC_StartConversion(void)
{
    ADC1->CR |= 0b100;
    while ((ADC1->ISR & 0b100) == 0);
    resistance = ADC1->DR;
}
```

myADC.h

```
void myADC_Init(void);
void myADC_StartConversion(void);
```

myDAC.c

```
#include "myDAC.h"
#include "globals.h"
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
```

```
void myDAC_Init(void)
{
    /* Enable clock for DAC peripheral */
    RCC->APB1ENR |= RCC_APB1ENR_DACEN;
    /* Configure DAC: output buffer enabled by default */
    DAC->CR &= ~(0b111); /* DAC channel1 trigger disabled(bit2) and DAC channel1 output
buffer enabled (bit1) and  DAC channel1 disabled (bit0) */
    DAC->CR |= 0b001; /* DAC channel1 enable (bit0) */
}

void myDAC_SetValue(unsigned int value)
{
    /* Set the 12-bit right-aligned data for DAC channel 1 */
    DAC->DHR12R1 = value & 0x0FFF; /* Ensure only lower 12 bits are used */
}
```

myDAC.h

```
void myDAC_Init(void);
void myDAC_SetValue(unsigned int value);
```

myEXTI.c

```
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
#include "myEXTI.h"
#include "globals.h"

void myEXTI_Init()
{
    /* Map EXTI0 line to PA0 */
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI0;   // Clear EXTI0 bits
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PA; // Map PA0 to EXTI0

    /* Map EXTI2 line to PB2 */
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI2;   // Clear EXTI2 bits
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PB; // Map PB2 to EXTI2

    /* Map EXTI3 line to PB3 */
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3;   // Clear EXTI3 bits
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI3_PB; // Map PB3 to EXTI3

    /* Configure rising-edge trigger for EXTI0, EXTI2, and EXTI3 */
    EXTI->RTSR |= (EXTI_RTSR_TR0 | EXTI_RTSR_TR2 | EXTI_RTSR_TR3);

    /* Unmask interrupts for EXTI0, EXTI2, and EXTI3 */
    EXTI->IMR |= (EXTI_IMR_MR0 | EXTI_IMR_MR2 | EXTI_IMR_MR3);

    /* Enable NVIC interrupts:
       - EXTI0_1_IRQn handles EXTI0 and EXTI1
       - EXTI2_3_IRQn handles EXTI2 and EXTI3
    */
    NVIC_SetPriority(EXTI0_1_IRQn, 0);
    NVIC_EnableIRQ(EXTI0_1_IRQn);

    NVIC_SetPriority(EXTI2_3_IRQn, 0);
    NVIC_EnableIRQ(EXTI2_3_IRQn);
}

void EXTI2_3_IRQHandler()
{

    if ((EXTI->PR & EXTI_PR_PR2) != 0 && fivefivefive == 0)
    // rising edge on PB2
    {
        if (timerTriggered == 0) // first edge detected
        {
            tim2Overflow = 0;          // clear overflow counter
            TIM2->CNT = 0;             // reset counter
            TIM2->CR1 |= TIM_CR1_CEN; // start timer
            timerTriggered = 1;        // mark timer running
        }
        else // second edge detected
        {
            TIM2->CR1 &= ~(TIM_CR1_CEN); // stop timer
            /* Combine overflow counter and current CNT for full 64-bit  cycle count */
            uint64_t cycles = ((uint64_t)tim2Overflow << 32) + (uint64_t)TIM2->CNT;
            // Avoid divide-by-zero
            uint32_t freq_hz = (cycles ? (uint32_t)((SystemCoreClock + (cycles / 2)) /
cycles) : 0);
            /* small hysteresis to reduce display oscillation at low frequencies */
            int diff = (freq_hz > lastFrequency) ? (freq_hz - lastFrequency) :
(lastFrequency - freq_hz);
            if (diff > 2) // only update if change > 2 Hz
            {
                frequency = freq_hz;
                lastFrequency = freq_hz;
            }
            timerTriggered = 0; // ready for next
        }
        EXTI->PR |= EXTI_PR_PR2; // clear EXTI2 pending flag
    }

    if ((EXTI->PR & EXTI_PR_PR3) != 0 && fivefivefivefive == 1) // rising edge on PB3
    {
        if (timerTriggered == 0) // first edge detected
        {
            tim2Overflow = 0;          // clear overflow counter
            TIM2->CNT = 0;             // reset counter
            TIM2->CR1 |= TIM_CR1_CEN; // start timer
            timerTriggered = 1;        // mark timer running
        }
        else // second edge detected
        {
            TIM2->CR1 &= ~(TIM_CR1_CEN); // stop timer
            uint64_t cycles = ((uint64_t)tim2Overflow << 32) + (uint64_t)TIM2->CNT;
            uint32_t freq_hz = (cycles ? (uint32_t)((SystemCoreClock + (cycles / 2)) /
cycles) : 0);
            int diff = (freq_hz > lastFrequency) ? (freq_hz - lastFrequency) :
```

```
(lastFrequency - freq_hz);
            if (diff > 2)
            {
                frequency = freq_hz;
                lastFrequency = freq_hz;
            }
            timerTriggered = 0; // ready for next
        }
        EXTI->PR |= EXTI_PR_PR3; // clear EXTI3 pending flag
    }
}

void EXTI0_1_IRQHandler()
{
    if ((EXTI->PR & EXTI_PR_PR0) != 0) // rising edge on PA0
    {
        if (GPIOA->IDR & GPIO_IDR_0) // confirm PA0 is high
        {
            trace_printf("Switching Freq source \n");
            if (fivefivefive == 0)
            {
                fivefivefive = 1;
            }
            else
            {
                fivefivefive = 0;
            }

            TIM2->CR1 &= ~(TIM_CR1_CEN); // ensure timer stopped
            TIM2->CNT = 0;               // reset counter
            timerTriggered = 0;          // ensure next edge is treated as first
        }
        EXTI->PR |= EXTI_PR_PR0; // clear EXTI0 pending flag
    }
}
```

myEXTI.h

```
void myEXTI_Init();
void EXTI2_3_IRQHandler();
void EXTI0_1_IRQHandler();
```

mySPI.c

```
#include "stm32f0xx.h"
#include "diag/Trace.h"
#include "stm32f0xx_hal.h"
#include "stm32f0xx_hal_spi.h"
#include <string.h>
#include "cmsis/cmsis_device.h"
#include <stdio.h>
#include "globals.h"
#include "mySPI.h"
SPI_HandleTypeDef SPI_Handle;
```

```
/*************************************************************************************
 *LED Display initialization commands
*************************************************************************************/
unsigned char oled_init_cmds[] =
    {
        0xAE,
        0x20, 0x00,
        0x40,
        0xA0 | 0x01,
        0xA8, 0x40 - 1,
        0xC0 | 0x08,
        0xD3, 0x00,
        0xDA, 0x32,
        0xD5, 0x80,
        0xD9, 0x22,
        0xDB, 0x30,
        0x81, 0xFF,
        0xA4,
        0xA6,
        0xAD, 0x30,
        0x8D, 0x10,
        0xAE | 0x01,
        0xC0,
        0xA0};

/*************************************************************************************
* Character specifications for LED Display (1 row = 8 bytes = 1 ASCII character)
*************************************************************************************/
// Example: to display '4', retrieve 8 data bytes stored in Characters[52][X] row
// (where X = 0, 1, ..., 7) and send them one by one to LED Display.
// Row number = character ASCII code (e.g., ASCII code of '4' is 0x34 = 52)
unsigned char Characters[][8] = {
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // SPACE
    {0b00000000, 0b00000000, 0b01011111, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // !
    {0b00000000, 0b00000111, 0b00000000, 0b00000111, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // "
    {0b00010100, 0b01111111, 0b00010100, 0b01111111, 0b00010100, 0b00000000, 0b00000000,
0b00000000}, // #
    {0b00100100, 0b00101010, 0b01111111, 0b00101010, 0b00010010, 0b00000000, 0b00000000,
0b00000000}, // $
    {0b00100011, 0b00010011, 0b00001000, 0b01100100, 0b01100010, 0b00000000, 0b00000000,
0b00000000}, // %
    {0b00110110, 0b01001001, 0b01010101, 0b00100010, 0b01010000, 0b00000000, 0b00000000,
0b00000000}, // &
    {0b00000000, 0b00000101, 0b00000011, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // '
    {0b00000000, 0b00011100, 0b00100010, 0b01000001, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // (
    {0b00000000, 0b01000001, 0b00100010, 0b00011100, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // )
    {0b00010100, 0b00001000, 0b00111110, 0b00001000, 0b00010100, 0b00000000, 0b00000000,
0b00000000}, // *
    {0b00001000, 0b00001000, 0b00111110, 0b00001000, 0b00001000, 0b00000000, 0b00000000,
0b00000000}, // +
    {0b00000000, 0b01010000, 0b00110000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // ,
    {0b00001000, 0b00001000, 0b00001000, 0b00001000, 0b00001000, 0b00000000, 0b00000000,
0b00000000}, // -
    {0b00000000, 0b01100000, 0b01100000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // .
    {0b00100000, 0b00010000, 0b00001000, 0b00000100, 0b00000010, 0b00000000, 0b00000000,
0b00000000}, // /
    {0b00111110, 0b01010001, 0b01001001, 0b01000101, 0b00111110, 0b00000000, 0b00000000,
0b00000000}, // 0
    {0b00000000, 0b01000010, 0b01111111, 0b01000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // 1
    {0b01000010, 0b01100001, 0b01010001, 0b01001001, 0b01000110, 0b00000000, 0b00000000,
0b00000000}, // 2
    {0b00100001, 0b01000001, 0b01000101, 0b01001011, 0b00110001, 0b00000000, 0b00000000,
0b00000000}, // 3
    {0b00011000, 0b00010100, 0b00010010, 0b01111111, 0b00010000, 0b00000000, 0b00000000,
0b00000000}, // 4
    {0b00100111, 0b01000101, 0b01000101, 0b01000101, 0b00111001, 0b00000000, 0b00000000,
0b00000000}, // 5
    {0b00111100, 0b01001010, 0b01001001, 0b01001001, 0b00110000, 0b00000000, 0b00000000,
0b00000000}, // 6
    {0b00000011, 0b00000001, 0b01110001, 0b00001001, 0b00000111, 0b00000000, 0b00000000,
0b00000000}, // 7
    {0b00110110, 0b01001001, 0b01001001, 0b01001001, 0b00110110, 0b00000000, 0b00000000,
0b00000000}, // 8
    {0b00000110, 0b01001001, 0b01001001, 0b00101001, 0b00011110, 0b00000000, 0b00000000,
0b00000000}, // 9
    {0b00000000, 0b00110110, 0b00110110, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // :
    {0b00000000, 0b01010110, 0b00110110, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // ;
    {0b00001000, 0b00010100, 0b00100010, 0b01000001, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // <
    {0b00010100, 0b00010100, 0b00010100, 0b00010100, 0b00010100, 0b00000000, 0b00000000,
0b00000000}, // =
    {0b00000000, 0b01000001, 0b00100010, 0b00010100, 0b00001000, 0b00000000, 0b00000000,
0b00000000}, // >
    {0b00000010, 0b00000001, 0b01010001, 0b00001001, 0b00000110, 0b00000000, 0b00000000,
0b00000000}, // ?
    {0b00110010, 0b01001001, 0b01111001, 0b01000001, 0b00111110, 0b00000000, 0b00000000,
0b00000000}, // @
    {0b01111110, 0b00010001, 0b00010001, 0b00010001, 0b01111110, 0b00000000, 0b00000000,
0b00000000}, // A
    {0b01111111, 0b01001001, 0b01001001, 0b01001001, 0b00110110, 0b00000000, 0b00000000,
0b00000000}, // B
    {0b00111110, 0b01000001, 0b01000001, 0b01000001, 0b00100010, 0b00000000, 0b00000000,
0b00000000}, // C
    {0b01111111, 0b01000001, 0b01000001, 0b00100010, 0b00011100, 0b00000000, 0b00000000,
0b00000000}, // D
    {0b01111111, 0b01001001, 0b01001001, 0b01001001, 0b01000001, 0b00000000, 0b00000000,
0b00000000}, // E
    {0b01111111, 0b00001001, 0b00001001, 0b00001001, 0b00000001, 0b00000000, 0b00000000,
0b00000000}, // F
    {0b00111110, 0b01000001, 0b01001001, 0b01001001, 0b01111010, 0b00000000, 0b00000000,
0b00000000}, // G
    {0b01111111, 0b00001000, 0b00001000, 0b00001000, 0b01111111, 0b00000000, 0b00000000,
0b00000000}, // H
    {0b01000000, 0b01000001, 0b01111111, 0b01000001, 0b01000000, 0b00000000, 0b00000000,
0b00000000}, // I
    {0b00100000, 0b01000000, 0b01000001, 0b00111111, 0b00000001, 0b00000000, 0b00000000,
0b00000000}, // J
    {0b01111111, 0b00001000, 0b00010100, 0b00100010, 0b01000001, 0b00000000, 0b00000000,
0b00000000}, // K
    {0b01111111, 0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b00000000, 0b00000000,
0b00000000}, // L
    {0b01111111, 0b00000010, 0b00001100, 0b00000010, 0b01111111, 0b00000000, 0b00000000,
0b00000000}, // M
    {0b01111111, 0b00000100, 0b00001000, 0b00010000, 0b01111111, 0b00000000, 0b00000000,
0b00000000}, // N
    {0b00111110, 0b01000001, 0b01000001, 0b01000001, 0b00111110, 0b00000000, 0b00000000,
0b00000000}, // O
    {0b01111111, 0b00001001, 0b00001001, 0b00001001, 0b00000110, 0b00000000, 0b00000000,
0b00000000}, // P
    {0b00111110, 0b01000001, 0b01010001, 0b00100001, 0b01011110, 0b00000000, 0b00000000,
0b00000000}, // Q
    {0b01111111, 0b00001001, 0b00011001, 0b00101001, 0b01000110, 0b00000000, 0b00000000,
0b00000000}, // R
    {0b01000110, 0b01001001, 0b01001001, 0b01001001, 0b00110001, 0b00000000, 0b00000000,
0b00000000}, // S
    {0b00000001, 0b00000001, 0b01111111, 0b00000001, 0b00000001, 0b00000000, 0b00000000,
0b00000000}, // T
    {0b00111111, 0b01000000, 0b01000000, 0b01000000, 0b00111111, 0b00000000, 0b00000000,
0b00000000}, // U
    {0b00011111, 0b00100000, 0b01000000, 0b00100000, 0b00011111, 0b00000000, 0b00000000,
0b00000000}, // V
    {0b00111111, 0b01000000, 0b00111000, 0b01000000, 0b00111111, 0b00000000, 0b00000000,
0b00000000}, // W
    {0b01100011, 0b00010100, 0b00001000, 0b00010100, 0b01100011, 0b00000000, 0b00000000,
0b00000000}, // X
    {0b00000111, 0b00001000, 0b01110000, 0b00001000, 0b00000111, 0b00000000, 0b00000000,
0b00000000}, // Y
    {0b01100001, 0b01010001, 0b01001001, 0b01000101, 0b01000011, 0b00000000, 0b00000000,
0b00000000}, // Z
    {0b01111111, 0b01000001, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // [
    {0b00010101, 0b00010110, 0b01111100, 0b00010110, 0b00010101, 0b00000000, 0b00000000,
0b00000000}, // back slash
    {0b00000000, 0b00000000, 0b00000000, 0b01000001, 0b01111111, 0b00000000, 0b00000000,
0b00000000}, // ]
    {0b00000100, 0b00000010, 0b00000001, 0b00000010, 0b00000100, 0b00000000, 0b00000000,
0b00000000}, // ^
    {0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b01000000, 0b00000000, 0b00000000,
0b00000000}, // _
    {0b00000000, 0b00000001, 0b00000010, 0b00000100, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // `
    {0b00100000, 0b01010100, 0b01010100, 0b01010100, 0b01111000, 0b00000000, 0b00000000,
0b00000000}, // a
    {0b01111111, 0b01001000, 0b01000100, 0b01000100, 0b00111000, 0b00000000, 0b00000000,
0b00000000}, // b
    {0b00111000, 0b01000100, 0b01000100, 0b01000100, 0b00100000, 0b00000000, 0b00000000,
0b00000000}, // c
    {0b00111000, 0b01000100, 0b01000100, 0b01001000, 0b01111111, 0b00000000, 0b00000000,
0b00000000}, // d
    {0b00111000, 0b01010100, 0b01010100, 0b01010100, 0b00011000, 0b00000000, 0b00000000,
0b00000000}, // e
    {0b00001000, 0b01111110, 0b00001001, 0b00000001, 0b00000010, 0b00000000, 0b00000000,
0b00000000}, // f
    {0b00001100, 0b01010010, 0b01010010, 0b01010010, 0b00111110, 0b00000000, 0b00000000,
0b00000000}, // g
    {0b01111111, 0b00001000, 0b00000100, 0b00000100, 0b01111000, 0b00000000, 0b00000000,
0b00000000}, // h
    {0b00000000, 0b01000100, 0b01111101, 0b01000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // i
    {0b00100000, 0b01000000, 0b01000100, 0b00111101, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // j
    {0b01111111, 0b00010000, 0b00101000, 0b01000100, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // k
    {0b00000000, 0b01000001, 0b01111111, 0b01000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // l
    {0b01111100, 0b00000100, 0b00011000, 0b00000100, 0b01111000, 0b00000000, 0b00000000,
0b00000000}, // m
    {0b01111100, 0b00001000, 0b00000100, 0b00000100, 0b01111000, 0b00000000, 0b00000000,
0b00000000}, // n
    {0b00111000, 0b01000100, 0b01000100, 0b01000100, 0b00111000, 0b00000000, 0b00000000,
0b00000000}, // o
    {0b01111100, 0b00010100, 0b00010100, 0b00010100, 0b00001000, 0b00000000, 0b00000000,
0b00000000}, // p
    {0b00001000, 0b00010100, 0b00010100, 0b00011000, 0b01111100, 0b00000000, 0b00000000,
0b00000000}, // q
    {0b01111100, 0b00001000, 0b00000100, 0b00000100, 0b00001000, 0b00000000, 0b00000000,
0b00000000}, // r
    {0b01001000, 0b01010100, 0b01010100, 0b01010100, 0b00100000, 0b00000000, 0b00000000,
0b00000000}, // s
    {0b00000100, 0b00111111, 0b01000100, 0b01000000, 0b00100000, 0b00000000, 0b00000000,
0b00000000}, // t
    {0b00111100, 0b01000000, 0b01000000, 0b00100000, 0b01111100, 0b00000000, 0b00000000,
0b00000000}, // u
    {0b00011100, 0b00100000, 0b01000000, 0b00100000, 0b00011100, 0b00000000, 0b00000000,
0b00000000}, // v
    {0b00111100, 0b01000000, 0b00111000, 0b01000000, 0b00111100, 0b00000000, 0b00000000,
0b00000000}, // w
    {0b01000100, 0b00101000, 0b00010000, 0b00101000, 0b01000100, 0b00000000, 0b00000000,
0b00000000}, // x
    {0b00001100, 0b01010000, 0b01010000, 0b01010000, 0b00111100, 0b00000000, 0b00000000,
0b00000000}, // y
    {0b01000100, 0b01100100, 0b01010100, 0b01001100, 0b01000100, 0b00000000, 0b00000000,
0b00000000}, // z
    {0b00000000, 0b00001000, 0b00110110, 0b01000001, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // {
    {0b00000000, 0b00000000, 0b01111111, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // |
    {0b00000000, 0b01000001, 0b00110110, 0b00001000, 0b00000000, 0b00000000, 0b00000000,
0b00000000}, // }
    {0b00001000, 0b00001000, 0b00101010, 0b00011100, 0b00001000, 0b00000000, 0b00000000,
0b00000000}, // ~
    {0b00001000, 0b00011100, 0b00101010, 0b00001000, 0b00001000, 0b00000000, 0b00000000,
0b00000000}  // <-
};

/*****************************************************************************************
 * SPI Display Configuration Function
*****************************************************************************************/
void mySPI_Init(void)
{
    RCC->APB1ENR |= RCC_APB1ENR_SPI2EN; /* SPI2 clock enable */
    SPI_Handle.Instance = SPI2; /* using SPI2 */
    SPI_Handle.Init.Direction = SPI_DIRECTION_1LINE; /* only MOSI */
    SPI_Handle.Init.Mode = SPI_MODE_MASTER; /* master mode */
    SPI_Handle.Init.DataSize = SPI_DATASIZE_8BIT; /* set size (OLED expectation) */
    SPI_Handle.Init.CLKPolarity = SPI_POLARITY_LOW; /* clock idled low (0v) */
    SPI_Handle.Init.CLKPhase = SPI_PHASE_1EDGE; /* sample rising edge */
    SPI_Handle.Init.NSS = SPI_NSS_SOFT; /* programmer controls chip select */
    SPI_Handle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256; /* stable Hz */
    SPI_Handle.Init.FirstBit = SPI_FIRSTBIT_MSB; /* most significant bit first */
    SPI_Handle.Init.CRCPolynomial = 7;
    HAL_SPI_Init(&SPI_Handle); /* initialize SPI2 */
    __HAL_SPI_ENABLE(&SPI_Handle); /* enable SPI2 */
}

/*****************************************************************************************
 * Tim3 Configuration Function
*****************************************************************************************/
void myTIM3_Init(void)
{
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN; /* enable TIM3 clock */
    TIM3->PSC = 48000 - 1; /* timer counts every 1 ms */
    TIM3->CR1 = 0; /* initially counter disabled */
}

/*****************************************************************************************
 * Oled Write Functions
*****************************************************************************************/
void oled_Write(unsigned char Value)
{
    /* wait until SPI2 ready for w (TXE = 1) */
    while ((SPI2->SR & SPI_SR_TXE) == 0)
        ;
    /* send one 8-bit character */
    HAL_SPI_Transmit(&SPI_Handle, &Value, 1, HAL_MAX_DELAY);
    /* wait until transmission is complete (TXE = 1); */
    while ((SPI2->SR & SPI_SR_BSY) != 0)
        ;
}
void oled_Write_Cmd(unsigned char cmd)
```

```c
{
    GPIOB->ODR |= (1 << 8);  /* make PB8 = CS# = 1 */
    GPIOB->ODR &= ~(1 << 9); /* make PB9 = D/C# = 0 */
    GPIOB->ODR &= ~(1 << 8); /* make PB8 CS# = 0 */
    oled_Write(cmd);
    GPIOB->ODR |= (1 << 8); /* make PB8 CS# = 1 */
}
void oled_Write_Data(unsigned char data)
{
    GPIOB->ODR |= (1 << 8);  /* make PB8 = CS# = 1 */
    GPIOB->ODR |= (1 << 9);  /* make PB9 = D/C# = 1 */
    GPIOB->ODR &= ~(1 << 8); /* make PB8 = CS# = 0 */
    oled_Write(data);
    GPIOB->ODR |= (1 << 8); // make PB8 = CS# = 1
}

/*****************************************************************************************
* Oled Configuration Function
 *****************************************************************************************/
void oled_Config(void)
{
    /* reset LED Display (RES# = PB11) */
    GPIOB->ODR &= ~(1 << 11); /* make pin PB11 = 0 */
    for (volatile int i = 0; i < 50000; i++); /* wait for a few ms */
    GPIOB->ODR |= (1 << 11); /* make pin PB11 = 1 */
    for (volatile int i = 0; i < 50000; i++); /* wait for a few ms */
    /* send initialization commands to LED Display */
    for (unsigned int i = 0; i < sizeof(oled_init_cmds); i++)
    {
        oled_Write_Cmd(oled_init_cmds[i]);
    }
    /* fill LED display data memory (GDDRAM) with zeros: */
    for (unsigned char page = 0; page < 8; page++)
    {
        oled_Write_Cmd(0xB0 | page); /* set page address */
        oled_Write_Cmd(0x02);        /* lower column */
        oled_Write_Cmd(0x10);        /* higher column */
        for (unsigned char seg = 0; seg < 128; seg++)
        {
            oled_Write_Data(0x00);
        }
    }
}

/*****************************************************************************************
 * Oled Refresh Function
*****************************************************************************************/
void refresh_OLED(int freq_local)
{
    unsigned char Buffer[17];
/* at most 16 characters per PAGE + terminating '\0' */
    snprintf(Buffer, sizeof(Buffer), "R: %5u Ohms", ((resistance * 5000) / 0xFFF)); /*
display resistance also converting to entire domain */
    oled_Write_Cmd(0xB0 | 0);
    oled_Write_Cmd(0x02); /* page 0 */
    oled_Write_Cmd(0x10); /* lower column start */
    for (int i = 0; Buffer[i] != '\0'; i++) /* higher column start */
    {
        unsigned char c = Buffer[i];
        for (int j = 0; j < 8; j++)
        {
            oled_Write_Data(Characters[c][j]);
        }
    }
    snprintf(Buffer, sizeof(Buffer), "F: %5d Hz", freq_local); /* display frequency */
    oled_Write_Cmd(0xB0 | 2); /* page 2 */
    oled_Write_Cmd(0x02); /* lower column start */
    oled_Write_Cmd(0x10); /* higher column start */
    for (int i = 0; Buffer[i] != '\0'; i++)
    {
        unsigned char c = Buffer[i];
        for (int j = 0; j < 8; j++)
        {
            oled_Write_Data(Characters[c][j]);
        }
    }
    /* Wait for ~100 ms to get ~10 frames/sec refresh rate TIM3 is used to implement
this delay (via polling) */
    TIM3->ARR = 100; /* 100 ms */
    TIM3->CNT = 0; /* reset counter */
    TIM3->SR &= ~TIM_SR_UIF; /* clear update flag */
    TIM3->CR1 |= TIM_CR1_CEN; /* start timer */
    while ((TIM3->SR & TIM_SR_UIF) == 0); /* wait until UIF = 1 */
    TIM3->CR1 &= ~TIM_CR1_CEN; /* stop timer */
}
```

mySPI.h

```
void mySPI_Init(void);
void oled_Write(unsigned char);
void oled_Write_Cmd(unsigned char);
void oled_Write_Data(unsigned char);
void oled_Config(void);
void refresh_OLED(int freq_local);
void myTIM3_Init(void);
```

myTIM2.c

```
#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"
#include "myTIM2.h"
#include "globals.h"

void myTIM2_Init()
{
    /* Enable clock for TIM2 peripheral */
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    /* Configure TIM2: buffer auto-reload, count up, stop on overflow, enable update
events, interrupt on overflow only */
```

```
    TIM2->CR1 = ((uint16_t)0x008C);
    TIM2->PSC = myTIM2_PRESCALER; /* Set clock prescaler value */
    TIM2->ARR = myTIM2_PERIOD; /* Set auto-reloaded delay */

    /* Update timer registers */
    TIM2->EGR = ((uint16_t)0x0001);
    /* 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 |= TIM_DIER_UIE;
}

void TIM2_IRQHandler()
{
    if ((TIM2->SR & TIM_SR_UIF) != 0)
    {
        trace_printf("\n*** Overflow! ***\n");
        TIM2->SR &= ~(TIM_SR_UIF); /* Clear update interrupt flag */
        tim2Overflow++; /* Increment overflow counter */
    }
}
```

myTIM2.h

```
void myTIM2_Init();
void TIM2_IRQHandler();
```

Refrences

[1] *ECE 355 Lab Manual*, Department of Electrical and Computer Engineering, University of Victoria (UVic), 2025.

[2] *ECE 355 I/O Examples*, lecture slides, Department of Electrical and Computer Engineering, University of Victoria (UVic), 2025.

[3] *ECE 355 Interface Examples*, lecture slides, Department of Electrical and Computer Engineering, University of Victoria (UVic), 2025