2024-11-29

ECE355 Lab Project Report

Nolan Curtis, Rafael Edora

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

Table of Contents

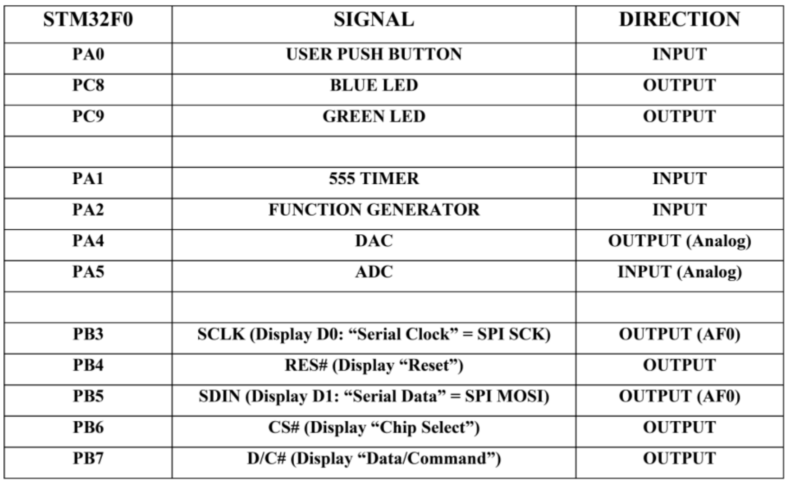
Problem Description

The lab's objective is to develop an embedded system for monitoring and controlling a pulse-width modulated (PWM) signal generated by a 555 timer (NE555 IC). We used an octocoupler (4N35 IC), which was driven by the microcontroller on the STMF0 Discovery board, to control the PWM signal frequency and duty cycle. The microcontroller was used to measure the voltage across a potentiometer (POT) on the PBMCUSLK board and forward it to the external octocoupler to control the PWM signal frequency. The microcontroller was also used to measure the frequency of the 555 timer signal, and the frequency of the Function Generator signal. By pressing the USER button we toggled which device frequency to display on the OLED Display (SSD1306 IC) on the PBMCUSLK board.

Specifications

The following specifications are derived from the ECE 355 Lab Manual [1].

1. Square Wave Signal Generation
    a. Use a function generator to generate a square wave signal.
    b. Utilize a 555 timer to generate additional square-wave PWM signals.
2. Interrupt Handling
    a. Implement an interrupt handler triggered by the USER button (PA0 using EXTI0).
    b. Configure TIM2 to switch between measuring the frequencies of:
        i. The function generator signal (PA2 using EXTI2).
        ii. The 555 timer signal (PA1 using EXTI1).
3. Analog and Digital Signal Processing
    a. Use the ADC and DAC on the STM32F051R8T6 MCU (mounted on the STM32F0 Discovery board).
    b. Measure the potentiometer's analog voltage using the ADC with polling to calculate the potentiometer's resistance.
    c. Use the ADC's digital value to:
        i. Control the frequency of the PWM signal generated by the 555 timer.
        ii. Drive the DAC to produce an analog voltage signal for controlling the octocoupler.
    d. The octocoupler adjusts the frequency and duty cycle of the 555 timer output.
4. SPI Communication
    a. Use SPI to drive the LED display on the PBMCUSLK.



STM32F0	SIGNAL	DIRECTION
PA0	USER PUSH BUTTON	INPUT
PC8	BLUE LED	OUTPUT
PC9	GREEN LED	OUTPUT
PA1	555 TIMER	INPUT
PA2	FUNCTION GENERATOR	INPUT
PA4	DAC	OUTPUT (Analog)
PA5	ADC	INPUT (Analog)
PB3	SCLK (Display D0: "Serial Clock" = SPI SCK)	OUTPUT (AF0)
PB4	RES# (Display "Reset")	OUTPUT
PB5	SDIN (Display D1: "Serial Data" = SPI MOSI)	OUTPUT (AF0)
PB6	CS# (Display "Chip Select")	OUTPUT
PB7	D/C# (Display "Data/Command")	OUTPUT

Figure 1: STM32F0 Board Pins[1]

Design/Solution

Design Steps

We completed the project in the following order:

1. Checking the lab specifications

2. Programming the USER button interrupts
3. Programming PA2 interrupts and frequency calculations (intro lab part 2)
4. Wiring the 4N35 octocoupler, NE555 timer, J10 connector to the potentiometer, and J5 connector to the SSD1306 display.
5. Programming the display setup and refreshes
6. Programming the ADC and DAC setup and refreshes
7. Programming PA1 interrupts and configuring the USER button interrupts to toggle between PA1 and PA2 interrupts.

Special Programming Considerations

Setting up I/O devices

Every I/O interface must be configured via control registers to work according to our needs. This is because they have multiple possible settings. For example, we used TIM3 to count milliseconds, so we set the prescalar value (TIM3 PSC) so it would update every millisecond.

Most of these control register values only need to be set once, so for each device, we set the control registers in a single function (e.g., myGPIOA_Init()) and called it at the beginning of the program.

Handling Polling and Interrupts

When waiting for a device to reach a certain status, we used polling (TIM3) or interrupts (PA0, PA1, PA2, TIM2). We decided that polling was appropriate for TIM3 because we only used TIM3 for small delays in configuring the OLED display and delaying the OLED refresh rate. Thus, interrupts occurring during the busy-wait loop would not meaningfully affect the delay. To check TIM3 status during polling, we checked the UIF flag in the status register to see if it finished counting.

With interrupts, ports A0, A1, and A2 were mapped to EXTI lines. Since EXTI interrupts are handled in pairs, within the interrupt handler we had to check which device requested the interrupt. We did this by checking the EXTI pending register, and if the flag was set for a certain device we serviced that device, then cleared its flag.

Lab Work Partitioning

In general, we did pair programming with Rafael typing and Nolan reviewing. Nolan did the wiring.

Schematics

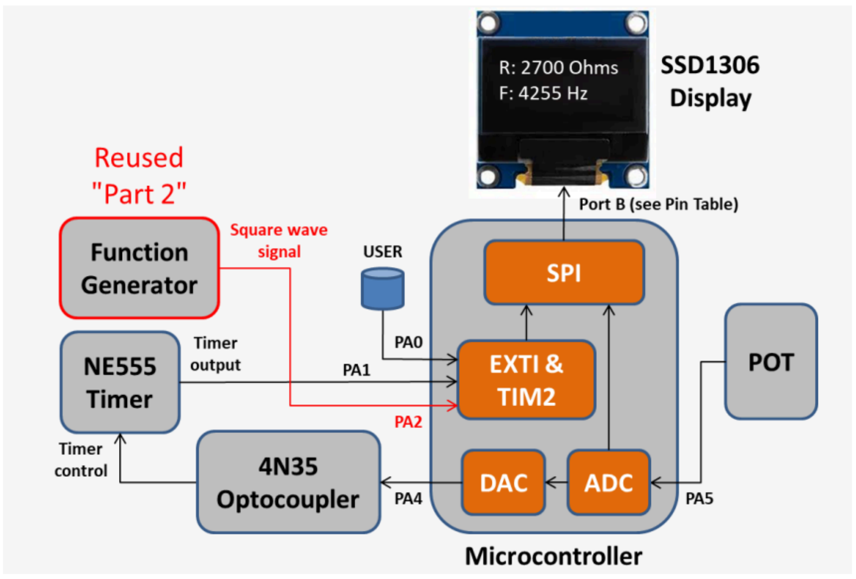


Figure 2: Device connections overview [2].

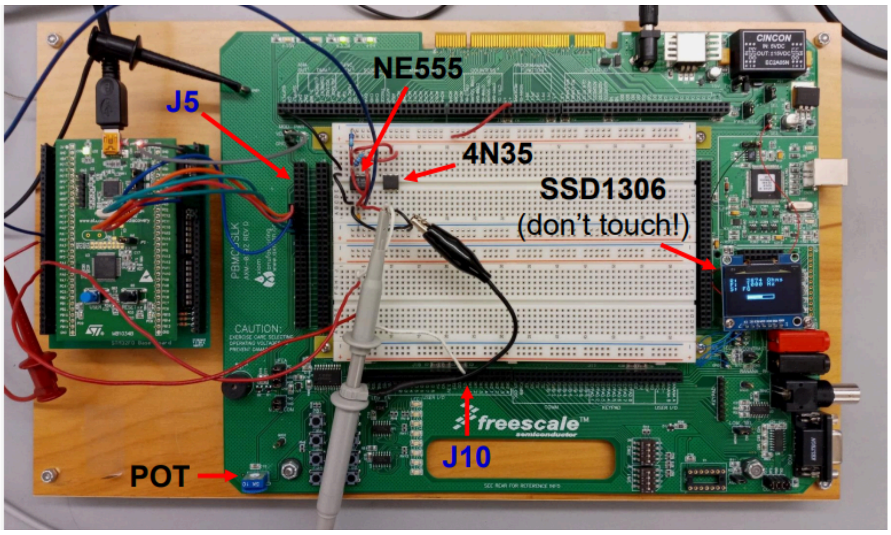
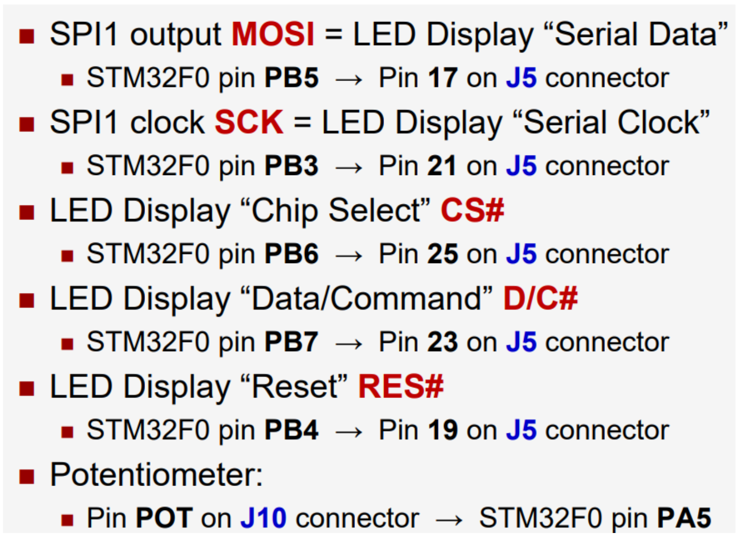


Figure 3: Wiring example[2]



- SPI1 output **MOSI** = LED Display “Serial Data”
  - STM32F0 pin **PB5** → Pin **17** on **J5** connector
- SPI1 clock **SCK** = LED Display “Serial Clock”
  - STM32F0 pin **PB3** → Pin **21** on **J5** connector
- LED Display “Chip Select” **CS#**
  - STM32F0 pin **PB6** → Pin **25** on **J5** connector
- LED Display “Data/Command” **D/C#**
  - STM32F0 pin **PB7** → Pin **23** on **J5** connector
- LED Display “Reset” **RES#**
  - STM32F0 pin **PB4** → Pin **19** on **J5** connector
- Potentiometer:
  - Pin **POT** on **J10** connector → STM32F0 pin **PA5**

Figure 4: PBMCUSLK Board Pins[2]

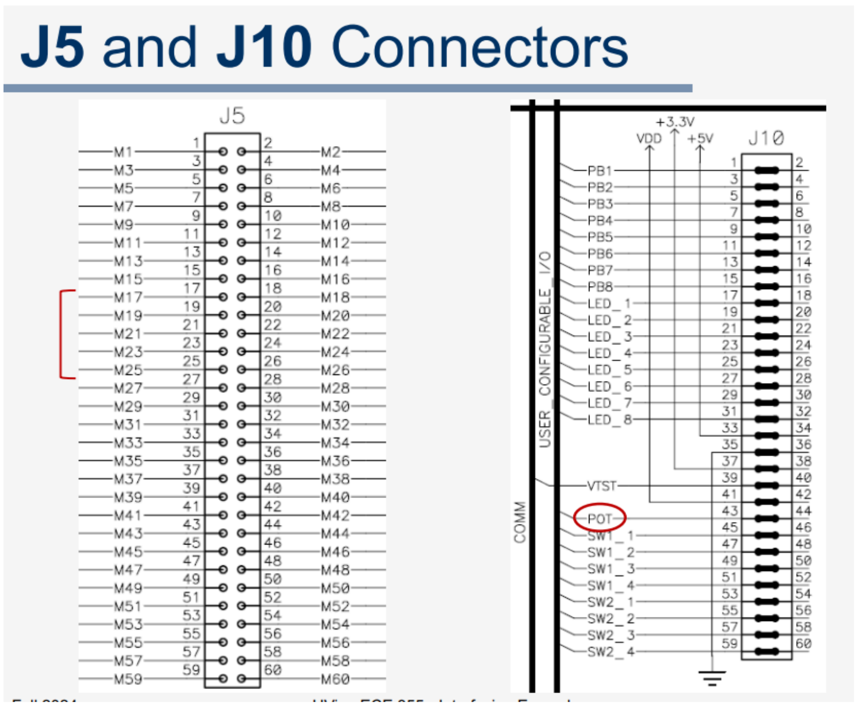


Figure 5: J5 and J10 Connectors[2]

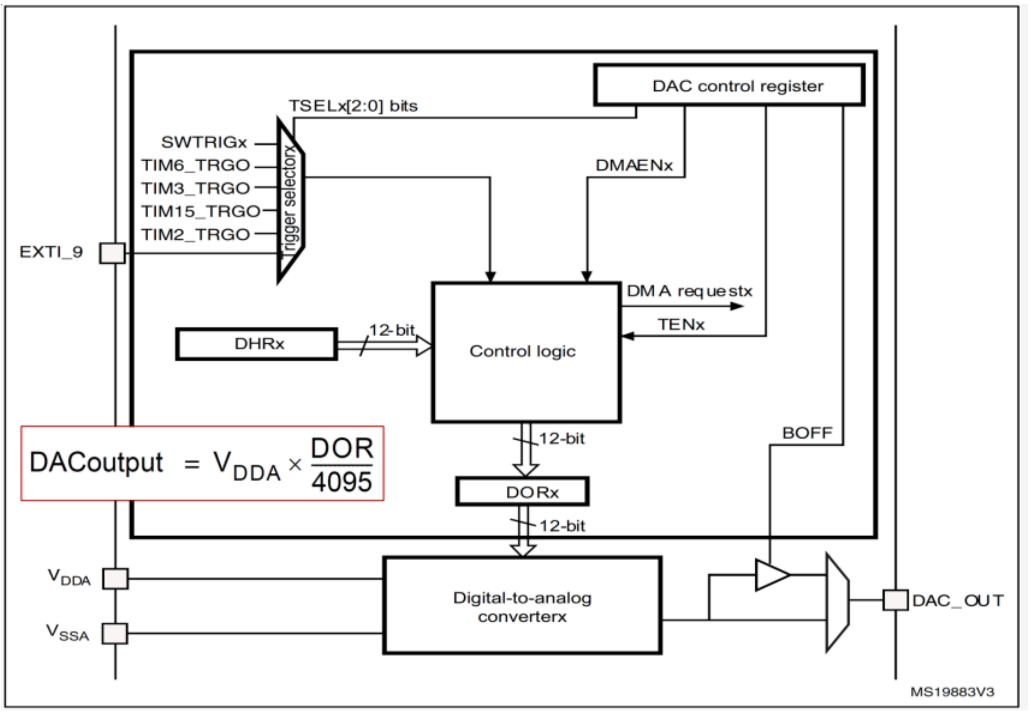


Figure 6: DAC Block Diagram[2]

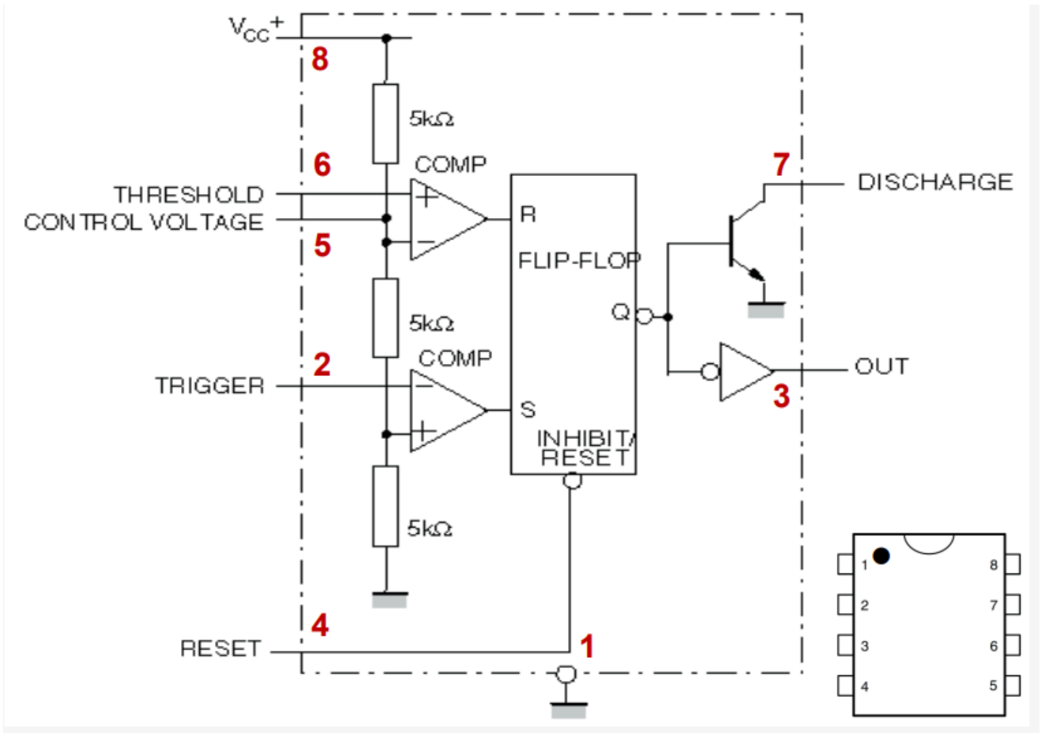


Figure 7: NE555 Block diagram[2]

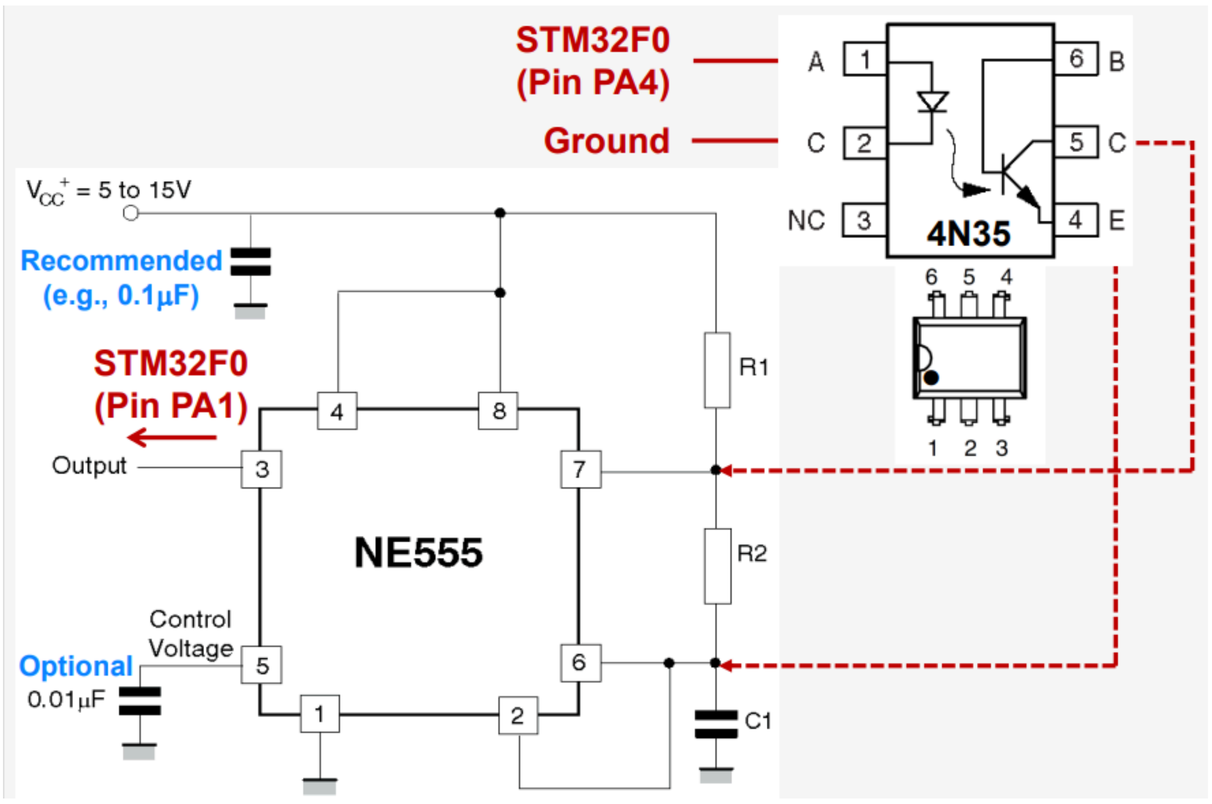


Figure 8: NE555 timer and DAC block diagram[2]

Testing

Procedure

The procedure we used for testing our results was the use of print statements and breakpoints to detect the behaviour of the code. Our reasoning for doing this was due to the fast build time of our code. The time between making a correction and testing the correction was very short allowing us to make many updates quickly. Using print statements and breakpoints allowed us to find the behaviour of our code quickly and determine the correct response to resolve the issue. This aligned with our development procedure of coding each separate system and then ensuring correct behaviour for that system. For example, for the USER button, we initialized the control registers to our needs and put in print statements in the handler to determine if an interrupt was thrown at the correct time, and if the interrupt was not thrown we would reconfigure the control registers and try again. This testing method allowed us to find the behaviour of the code quickly and resolve any issues.

Expected values

When our code ran with the intended behaviour our next step was to ensure the correct values were being displayed. We did this by discussing with the TA to find what is an appropriate range of values in the project. We did this for our function generator processing system to determine the appropriate max and min detectable frequency of our system. For the NE555 timer's max and min output frequency we used an oscilloscope to determine that our output was true to the DAC's output and ensured that it matched our displayed values.

Results

Square wave signal

The results we gathered are the maximum and minimum detectable frequency of our system using a square wave function generator being:
Maximum frequency is 433000Hz
Minimum frequency is 0.01117318435Hz(Period=89.5)
The maximum frequency is due to the system needing a certain amount of clock cycles to process the frequency and above this the system can skip over the next rising edge while still processing the last.
The minimum frequency occurs at a period of 89.5 seconds due to integer overflow on our timer register.

The errors we detected occurred at higher frequency with the displayed frequency being 2% higher than the correct output. We found this to be reasonable due to it only occurring at the high detectable frequencies where the our register is processing ~100 clock cycles between each rising edge meaning the difference between detection on 1 clock cycle turns into a 1% error range

NE555 timer and Octocoupler

The results we gathered from the 555 timers was the minimum and maximum frequency generated by 5 volts through the octocoupler and then into the timer. The frequency bounds we detected were:
Minimum frequency is 1009Hz
Maximum frequency is 1556Hz
This value we gathered changed depending on the hardware we ran the software on but stayed with the same range of ~1000 and ~1500
The calculated value of R2fixed calculated from the changing value of resistor R2 due to the parallel resistance from the octocupler was 3566 ohms when the octocoupler was at 5 volts.

Resistance 2 fixed Calculation

Resistance 1 and 2 on board are 5.1k ohms.

$$\textbf{Frequency:}\quad \frac{1.443}{(R1 + 2R2)C1}$$

Rfixed when output voltage is at 5k volts
1552=1.443/((5100+2R)((1/10000000)))
1.443/1552=(5100+2R)((1/10000000))
(1.443/1552)/(1/10000000) = (5100+2R)
(1.443/1552)/(1/10000000)-5100=2R
((1.443/1552)/(1/10000000)-5100)/2=R
R=2098.8

Rfixed when Output voltage is 0 volts
R=((1.443/1006)/(1/(10000000))-5100)/2
R=4621

Parallel resistance equation
(1/R2) = (1/R2fixed) + (1/Rchanging)
(R2changing) =((1/R2)-(1/R2fixed))^-1

Rfixed when output voltage is at 5k volts
(R2changing) =((1/2098.8)-(1/5100))^-1
(R2changing) = 3566
Rfixed when output voltage is 0 volts
(R2changing) =((1/4621)-(1/5100))^-1
(R2changing) =49200

Value of R2 resistor when voltage is at 5 volts is 3566 ohms

Discussion

Design Specifications

We adhered to the design specifications provided through the lab manual and the interface slides. Thus, there are no differences between our design specifications and the stated specifications.

Assumptions

We made one critical assumption: we assumed that the provided skeleton code was complete and did not need extra steps. When we programmed the display, we incorrectly set up the GPIOB ports and spent lots of time debugging until we realized the problem.

Shortcomings

The only shortcoming on the project is a ~2% error on our detection of the function generator's signal which is due to the uncertainty of detection at such a small number of clock cycles.

Lessons Learned

Here are some lessons we learned:

- Make sure the ports are all configured; before checking if any control registers are misconfigured, check if any have not been set (this is more likely).
- Use breakpoints instead of print statements whenever possible. They are easy to use and much faster.
- Interrupts can interfere with each other - try to complete interrupt service routines as quickly as possible, and disable interrupts for devices that are not in use.
- The USER button can be a bit inconsistent with triggering interrupts, but setting them to trigger on the falling edge can help.
- If a prescalar value is not set for TIM3, it can easily overflow due to its 16-bit count register (as opposed to the TIM2 32-bit count register).

Appendix A: Source Code

File: main.c

```
-----------------------------------------------------------------------------
// School: University of Victoria, Canada.
// Course: ECE 355 "Microprocessor-Based Systems".
// This is tutorial code for Part 1 of Introductory Lab
//
// See "system/include/cmsis/stm32f051x8.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f051x8.c" for handler declarations.
```

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

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

// ----- 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"

/* Definitions of registers and their bits are
 given in system/include/cmsis/stm32f051x8.h */

/*** Call this function to boost the STM32F0xx clock to 48 MHz ***/

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;
//
```

```c
// 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();
}

/*****************************************************************/

/* Clock prescaler for TIM2 timer: no prescaling */
#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Clock prescaler for TIM3 timer: set to trigger every 1ms */
#define myTIM3_PRESCALER ((uint16_t)48000 - 1)
/* Delay count for TIM2 timer: max size */
#define myTIM2_PERIOD ((uint32_t)0xFFFFFFFF)

void myGPIOA_Init(void);
void myGPIOB_Init(void);
void myDAC_Init(void);
void myADC_Init(void);
void myEXTI_Init(void);
void myTIM2_Init(void);
void myTIM3_Init(void);
void mySPI1_Init(void);
void update_Freq(void);
void update_POT(void);
void sleep_ms(uint16_t);

char firstEdge = 1;
char usingFunctionGenerator = 1;

unsigned int freq = 0;
unsigned int res = 0;
unsigned int pot = 0;
```

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

    SystemClock48MHz();
    myGPIOA_Init();
    myGPIOB_Init();
    myDAC_Init();
    myADC_Init();
    myEXTI_Init();
    myTIM2_Init();
    myTIM3_Init();
    mySPI1_Init();
    oled_config();

    while (1) {
        update_POT();
        refresh_OLED(freq, res);

        sleep_ms(100); // 10Hz refresh rate
    }

    return 0;
}

void myGPIOA_Init() {
    /* Enable clock for GPIOA peripheral */
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    /* Configure PA0 as input */
    GPIOA->MODER &= ~(GPIO_MODER_MODER0);
    /* Configure PA1 as input */
    GPIOA->MODER &= ~(GPIO_MODER_MODER1);
    /* Configure PA2 as input */
    GPIOA->MODER &= ~(GPIO_MODER_MODER2);
    /* Ensure no pull-up/pull-down for PA0 */
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR0);
    /* Ensure no pull-up/pull-down for PA1 */
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR1);
    /* Ensure no pull-up/pull-down for PA2 */
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR2);
    /* Configure PA4 and PA5 as analog mode */
    GPIOA->MODER |= GPIO_MODER_MODER4;
    GPIOA->MODER |= GPIO_MODER_MODER5;
}
```

```c
void myGPIOB_Init() {
    /* Enable clock for GPIOB peripheral */
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN;

    /* Configure PB3 and PB5 as AF0 */
    GPIOB->MODER &= ~(GPIO_MODER_MODER3);
    GPIOB->MODER |= GPIO_MODER_MODER3_1;
    GPIOB->AFR[0] &= ~(GPIO_AFRL_AFSEL3);

    GPIOB->MODER &= ~(GPIO_MODER_MODER5);
    GPIOB->MODER |= GPIO_MODER_MODER5_1;
    GPIOB->AFR[0] &= ~(GPIO_AFRL_AFSEL5);

    /* Configure PB4, PB6, and PB7 as output */
    GPIOB->MODER &= ~(GPIO_MODER_MODER4);
    GPIOB->MODER |= GPIO_MODER_MODER4_0;
    GPIOB->MODER &= ~(GPIO_MODER_MODER6);
    GPIOB->MODER |= GPIO_MODER_MODER6_0;
    GPIOB->MODER &= ~(GPIO_MODER_MODER7);
    GPIOB->MODER |= GPIO_MODER_MODER7_0;
}

void myDAC_Init() {
    RCC->APB1ENR |= RCC_APB1ENR_DACEN;

    /* Enable DAC_OUT1 channel */
    DAC->CR |= DAC_CR_EN1;

    /* Disable channel1 tri-state buffer */
    DAC->CR &= ~DAC_CR_BOFF1;

    /* Disable channel1 trigger enable */
    DAC->CR &= ~DAC_CR_TEN1;
}

void myADC_Init() {
    // turn on clock
    RCC->APB2ENR |= RCC_APB2ENR_ADCEN;

    // calibrate ADC
    ADC1->CR &= ~ADC_CR_ADEN;
    ADC1->CR |= ADC_CR_ADCAL;
    while (ADC1->CR & ADC_CR_ADCAL)
        ;

    // enable ADC
    ADC1->CR |= ADC_CR_ADEN;
    ADC1->CR &= ~ADC_CR_ADDIS;

    // wait for ready
    while ((ADC1->ISR & ADC_ISR_ADRDY) == 0)
        ;

    ADC1->CHSELR |= ADC_CHSELR_CHSEL5;

    // set to continuous and right-align
    ADC1->CFGR1 |= ADC_CFGR1_CONT;
    ADC1->CFGR1 &= ~ADC_CFGR1_ALIGN;

    // start ADC
    ADC1->CR |= ADC_CR_ADSTART;
}

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);

    /* Set clock prescaler value */
    TIM2->PSC = myTIM2_PRESCALER;
    /* Set auto-reloaded delay */
    TIM2->ARR = myTIM2_PERIOD;

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

    /* Assign TIM2 interrupt priority = 0 in NVIC */
    NVIC_SetPriority(TIM2_IRQn, 0);
    // Same as: NVIC->IP[3] = ((uint32_t)0x00FFFFFF);

    /* Enable TIM2 interrupts in NVIC */
    NVIC_EnableIRQ(TIM2_IRQn);
    // Same as: NVIC->ISER[0] = ((uint32_t)0x00008000) */

    /* Enable update interrupt generation */
    TIM2->DIER |= TIM_DIER_UIE;
}

void myTIM3_Init() {
    /* Enable clock for TIM3 peripheral */
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;

    /* Configure TIM3: buffer auto-reload, count down, stop on overflow,
     * enable update events, interrupt on overflow only */
    TIM3->CR1 = ((uint16_t) 0x001C);

    /* Set clock prescaler value */
    TIM3->PSC = myTIM3_PRESCALER;

    /* Update timer registers */
    TIM3->EGR = ((uint16_t) 0x0001);
}

void myEXTI_Init() {
    /* Map EXTI0 line to PA0 */
    // Relevant register: SYSCFG->EXTICR[0]
    SYSCFG->EXTICR[0] = SYSCFG_EXTICR1_EXTI0_PA;

    /* Map EXTI0 line to PA1 */
    // Relevant register: SYSCFG->EXTICR[1]
    SYSCFG->EXTICR[1] = SYSCFG_EXTICR1_EXTI1_PA;

    /* Map EXTI0 line to PA2 */
    // Relevant register: SYSCFG->EXTICR[2]
    SYSCFG->EXTICR[2] = SYSCFG_EXTICR1_EXTI2_PA;

    /* EXTI0 line interrupts: set falling-edge trigger */
    // Relevant register: EXTI->FTSR
    EXTI->FTSR |= 0x1;

    /* EXTI1 line interrupts: set rising-edge trigger */
    // Relevant register: EXTI->RTSR
    EXTI->RTSR |= 0x1 << 1;
    /* EXTI2 line interrupts: set rising-edge trigger */
    // Relevant register: EXTI->RTSR
    EXTI->RTSR |= 0x1 << 2;

    /* Unmask interrupts from EXTI0 line */
    // Relevant register: EXTI->IMR
    EXTI->IMR |= 0x1;

    /* Unmask interrupts from EXTI2 line */
    // Relevant register: EXTI->IMR
    EXTI->IMR |= 0x1 << 2;

    /* Assign EXTI0 interrupt priority = 0 in NVIC */
    // Relevant register: NVIC->IP[2], or use NVIC_SetPriority
    NVIC_SetPriority(EXTI0_1_IRQn, 0);

    /* Assign EXTI2 interrupt priority = 0 in NVIC */
    // Relevant register: NVIC->IP[2], or use NVIC_SetPriority
    NVIC_SetPriority(EXTI2_3_IRQn, 1);

    /* Enable EXTI0 interrupts in NVIC */
    // Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
    NVIC_EnableIRQ(EXTI0_1_IRQn);

    /* Enable EXTI2 interrupts in NVIC */
    // Relevant register: NVIC->ISER[0], or use NVIC_EnableIRQ
    NVIC_EnableIRQ(EXTI2_3_IRQn);
}

void mySPI1_Init() {
    RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // enable SPI1 clock
}

/* This handler is declared in system/src/cmsis/vectors_stm32f051x8.c */
void TIM2_IRQHandler() {
    /* Check if update interrupt flag is indeed set */
    if ((TIM2->SR & TIM_SR_UIF) != 0) {
        trace_printf("\n*** Overflow! ***\n");

        /* Clear update interrupt flag */
        // Relevant register: TIM2->SR
        TIM2->SR &= ~(TIM_SR_UIF);

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

void EXTI0_1_IRQHandler() {
    // EXTI0 is listening to the USER button
    if ((EXTI->PR & EXTI_PR_PR0) != 0) {
        firstEdge = 1; // start new period measurement
        EXTI->IMR ^= 0b110; // flip EXTI1 and EXTI2

        EXTI->PR |= EXTI_PR_PR0; // clear pending flag
    }

    // EXTI1 is listening to the NE555 timer
    if ((EXTI->PR & EXTI_PR_PR1) != 0) {
        update_Freq();

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

void EXTI2_3_IRQHandler() {
    // EXTI2 is listening to the function generator
    if ((EXTI->PR & EXTI_PR_PR2) != 0) {
        update_Freq();

        EXTI->PR |= EXTI_PR_PR2; // clear pending flag
    }
}

/**
 * calculate the signal frequency by measuring the period between two rising
 * edges
 */
void update_Freq() {
    if (firstEdge) {
```

```c
        firstEdge = 0;

        TIM2->CNT = 0;
        TIM2->CR1 |= TIM_CR1_CEN; // start timer
    } else {
        firstEdge = 1;

        TIM2->CR1 &= ~(TIM_CR1_CEN); // stop timer
        uint32_t count = TIM2->CNT;

        freq = SystemCoreClock / count;
    }
}

/**
 * read the potentiometer voltage from ADC, update the DAC, and calculate the
 * resistance
 */
void update_POT() {
    pot = ADC1->DR;
    res = pot * 5000 / 4095;
    DAC->DHR12R1 = pot;
}

/**
 * sleep for sleep_ms milliseconds
 */
void sleep_ms(uint16_t sleep_ms) {
    TIM3->SR &= ~(TIM_SR_UIF); // clear UIF

    // set timer to run for sleep_ms ms
    TIM3->CNT = sleep_ms;

    TIM3->CR1 |= TIM_CR1_CEN; // start timer

    while (!(TIM3->SR & TIM_SR_UIF))
        ;

    TIM3->SR &= ~(TIM_SR_UIF); // clear UIF

    TIM3->CR1 &= ~(TIM_CR1_CEN); // stop timer
}
```

```
#pragma GCC diagnostic pop

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

File: display.h
```c
void oled_Write(unsigned char);
void oled_Write_Cmd(unsigned char);
void oled_Write_Data(unsigned char);
void oled_config(void);
void refresh_OLED(unsigned int freq, unsigned int res);
```

File: display.c
```c
//
----------------------------------------------------------------------------
// School: University of Victoria, Canada.
// Course: ECE 355 "Microprocessor-Based Systems".
// This is the display module file for the course project.
//
----------------------------------------------------------------------------

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

// ----- main()
---------------------------------------------------------------

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

void refresh_OLED(unsigned int freq, unsigned int res);
void oled_Write(unsigned char);
void oled_Write_Cmd(unsigned char);
void oled_Write_Data(unsigned char);
void oled_config(void);

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}   // <-
};

/**
 * write to OLED display with updated frequency and resistance values
 */
void refresh_OLED( unsigned int freq, unsigned int res)
```

```
{
    // Buffer size = at most 16 characters per PAGE + terminating '\0'
    unsigned char Buffer[17] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

    snprintf( Buffer, 17, " R: %5u Ohms", res );
    /* Buffer now contains your character ASCII codes for LED Display
       - select PAGE (LED Display line) and set starting SEG (column)
       - for each c = ASCII code = Buffer[0], Buffer[1], ...,
           send 8 bytes in Characters[c][0-7] to LED Display
    */
    oled_Write_Cmd(0xB0);
    oled_Write_Cmd(0x05);
    oled_Write_Cmd(0x10);
    for (int i = 0; i < 17; i++) {
        for (int j = 0; j < 8; j++) {
            oled_Write_Data(Characters[Buffer[i]][j]);
        }
    }

    snprintf( Buffer, 17, " F: %5u Hz  ", freq );
    /* Buffer now contains your character ASCII codes for LED Display
       - select PAGE (LED Display line) and set starting SEG (column)
       - for each c = ASCII code = Buffer[0], Buffer[1], ...,
           send 8 bytes in Characters[c][0-7] to LED Display
    */
    oled_Write_Cmd(0xB1);
    oled_Write_Cmd(0x05);
    oled_Write_Cmd(0x10);
    for (int i = 0; i < 17; i++) {
        for (int j = 0; j < 8; j++) {
            oled_Write_Data(Characters[Buffer[i]][j]);
        }
    }
}

void oled_Write_Cmd( unsigned char cmd )
{
    GPIOB->BSRR = 0x1 << 6; // make PB6 = CS# = 1
    GPIOB->BRR = 0x1 << 7; // make PB7 = D/C# = 0
    GPIOB->BRR = 0x1 << 6; // make PB6 = CS# = 0
    oled_Write( cmd );
    GPIOB->BSRR = 0x1 << 6; // make PB6 = CS# = 1
```

```c
}

void oled_Write_Data( unsigned char data )
{
    GPIOB->BSRR = 0x1 << 6; // make PB6 = CS# = 1
    GPIOB->BSRR = 0x1 << 7; // make PB7 = D/C# = 1
    GPIOB->BRR = 0x1 << 6; // make PB6 = CS# = 0
    oled_Write( data );
    GPIOB->BSRR = 0x1 << 6; // make PB6 = CS# = 1
}

/**
 * write data or command byte to OLED display
 */
void oled_Write( unsigned char Value )
{
    /* Wait until SPI1 is ready for writing (TXE = 1 in SPI1_SR) */
    while ( !(SPI1->SR & SPI_SR_TXE) );

    /* Send one 8-bit character:
       - This function also sets BIDIOE = 1 in SPI1_CR1
    */
    HAL_SPI_Transmit( &SPI_Handle, &Value, 1, HAL_MAX_DELAY );

    /* Wait until transmission is complete (TXE = 1 in SPI1_SR) */
    while ( !(SPI1->SR & SPI_SR_TXE) );
}

/**
 * reset and initialize OLED display
 */
void oled_config( void )
{

// Don't forget to enable GPIOB clock in RCC
// Don't forget to configure PB3/PB5 as AF0
// Don't forget to enable SPI1 clock in RCC

    SPI_Handle.Instance = SPI1;

    SPI_Handle.Init.Direction = SPI_DIRECTION_1LINE;
    SPI_Handle.Init.Mode = SPI_MODE_MASTER;
    SPI_Handle.Init.DataSize = SPI_DATASIZE_8BIT;
    SPI_Handle.Init.CLKPolarity = SPI_POLARITY_LOW;
    SPI_Handle.Init.CLKPhase = SPI_PHASE_1EDGE;
    SPI_Handle.Init.NSS = SPI_NSS_SOFT;
    SPI_Handle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
    SPI_Handle.Init.FirstBit = SPI_FIRSTBIT_MSB;
    SPI_Handle.Init.CRCPolynomial = 7;

//
// Initialize the SPI interface
//
    HAL_SPI_Init( &SPI_Handle );

//
// Enable the SPI
//
    __HAL_SPI_ENABLE( &SPI_Handle );

    /* Reset LED Display (RES# = PB4):
       - make pin PB4 = 0, wait for a few ms
       - make pin PB4 = 1, wait for a few ms
    */
    GPIOB->BRR = 0x1 << 4;
    sleep_ms(100);
    GPIOB->BSRR = 0x1 << 4;
    sleep_ms(100);

//
// 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 each PAGE = 0, 1, ..., 7
           set starting SEG = 0
           call oled_Write_Data( 0x00 ) 128 times
    */
    for (int i = 0; i < 8; i++)
    {
        oled_Write_Cmd(0xB0 + i);
        oled_Write_Cmd(0x00);
        oled_Write_Cmd(0x10);
        for (int j = 0; j < 128; j++)
        {
            oled_Write_Data(0x00);
        }
    }
}

#pragma GCC diagnostic pop

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

References

[1] B. Sirna and D. N. Rakhmatov, ECE 355: Microprocessor-Based Systems Laboratory Manual. University of Victoria, 2023.

[2] D.N.Rakhmatov. (2024, season-03). *Interfacing Examples*. ece.uvic.ca. Retrieved November 29, 2024, from https://www.ece.uvic.ca/~ece355/lab/interfacex.pdf

[3] SSD1306: 128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller. (2009). [Reference manual]. Solomon Systech. Retrieved November 29, 2024, from https://www.ece.uvic.ca/~ece355/lab/supplement/SSD1306.pdf

[4] RM0091 Reference manual: STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM-based 32-bit MCUs. (2014). In ece.uvic.ca. www.st.com. Retrieved November 29, 2024, from https://www.ece.uvic.ca/~ece355/lab/supplement/stm32f0RefManual.pdf

[5] STM32F051xx: ARM-based 32-bit MCU, 16 to 64-KB Flash, timers, ADC, DAC and communication interfaces, 2.0-3.6 V. (2014). In ece.uvic.ca. www.st.com. Retrieved November 29, 2024, from https://www.ece.uvic.ca/~ece355/lab/supplement/stm32f051r8t6_datatsheet_DM00039193.pdf