

## Lab 4 Report

Dallin Peterson

A02366595

Caleb Hamblin

A02420019

### Objectives

The purposes of this lab are to gain experience:

1. Interfacing the microcontroller with a Liquid Crystal Display (LCD)
2. Programming microcontrollers in C
3. Reading and understanding LCD datasheets

### Procedure

For the prelab, we soldered a bar of pins to the LCD to connect the LCD to the solderless breadboard. Then, we connected the RS, E, DB4, DB5, DB6, and DB7 LCD pins to pins, 0, 1, 4, 5, 6, 7 respectively on the GPIOB output data register. The LCD pins VDD, RW, and A were connected to the 5V pin of the microcontroller. VSS, D0, D1, D2, D3, and K were connected to the GND pin of the microcontroller. VO was connected to a potentiometer that controlled the contrast of the LCD.

We began typing our code in C. First, we configured the GPIOB ports to be connected to the HSI clock in the function, `System_Clock_Init`. `LCD_Init` configured the necessary GPIOB pins and sent instructions to the LCD to be configured in 4-bit bus mode. Our `delay_ms` function receives an integer value and delays for the input number of milliseconds. `LCD_WriteCom` receives an 8-bit command and configures the LCD to receive a command and then sends an 8-bit instruction 4 bits at a time. `LCD_WriteData` receives an 8-bit data value and sends the data 4 bits at a time. `LCD_Pulse` controls the latch of the sent data in the LCD. `PutNibble` receives an 8-bit value, and sends the four least significant bits to the ODR. `LCD_DisplayString` receives the line number of a desired string to be sent to the LCD, and a pointer to the string. The function then sends each character until it reaches a null character.

Once our code was complete, we loaded it onto the microcontroller. The characters on the LCD appeared very slowly, and one at a time. We then changed our `delay_ms` function to delay much less time, which caused the string “Utah State ECE3710” to appear.

## Results

We were able to successfully create a library of C functions to control an LCD. This library was capable of sending data/commands, writing strings, and initializing the display. We were able to develop a better understanding of the LCD datasheet by examining the specific bits that were set for each command that we were given. One of the early iterations of our library did not function as expected because we our delay function delayed too long. This caused the string display to take so long to appear that we assumed the program didn't work. We fixed this issue by reducing the value of the counter within the delay function.

## Figures

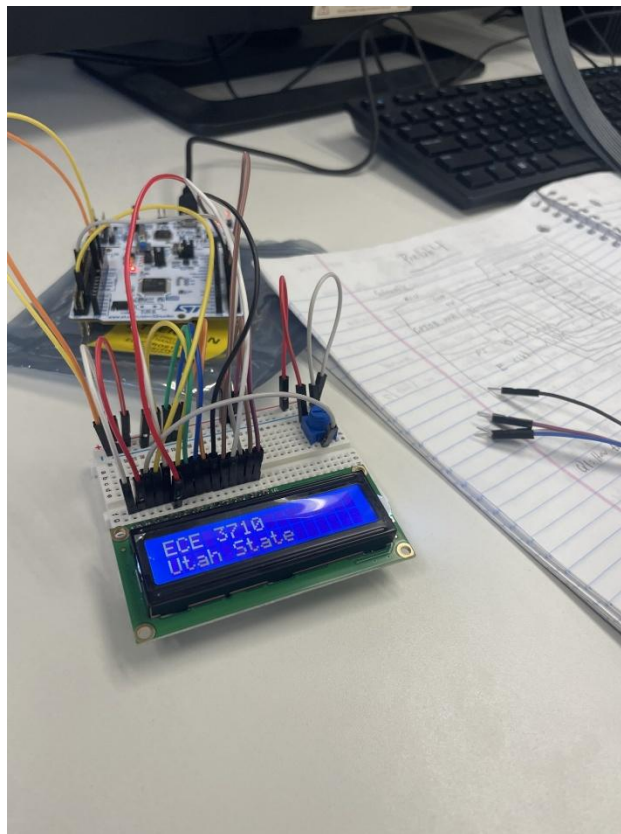


Figure 1. The working LCD.

## Conclusion

In this lab, we were able to interface a microcontroller with an LCD, program the microcontroller in C, and understand the LCD datasheet. We found that it is important to tune any delays so that time is not wasted unnecessarily.

## Appendix

LCD.c

```
#include "LCD.h"
```

```
#include "stm32l476xx.h"
```

```
#define RS (1<<0)
```

```
#define E (1<<1)
```

```
#define DB4 (1<<4)
```

```
#define DB5 (1<<5)
```

```
#define DB6 (1<<6)
```

```
#define DB7 (1<<7)
```

```
void delay_ms(unsigned int ms) {
```

```
    //16000 cycles is 1 ms
```

```
    //16 MHz clock frequency
```

```
    unsigned int clkdiv = 0;
```

```
    while(clkdiv<(400 * ms)) {
```

```
        clkdiv++;
```

```
    }
```

```
}
```

```
void LCD_WriteCom(unsigned char com) {
    GPIOB->ODR &= ~RS; //command mode

    PutNibble((com >> 4) & 0x0F); //send upper 4 bits of c
    LCD_Pulse();

    PutNibble(com & 0x0F); //clear upper 4 bits of c and send lower 4 bits
    LCD_Pulse();

    GPIOB->ODR |= RS; // return to data mode
}

void LCD_WriteData(unsigned char data) {
    GPIOB->ODR |= RS;

    PutNibble((data >> 4) & 0x0F); //send upper 4 bits of data
    LCD_Pulse();

    PutNibble(data & 0x0F); //clear upper 4 bits of data\ and send lower 4 bits
    LCD_Pulse();

}

void LCD_Init(void){
    //delay_ms(30);
```

```
//enable gpio clocks
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;

//configure gpio to control 4 bit bus and E,RW,RS lines.
GPIOB->MODER &= ~(0xFFFF); //clear 16 least significant bits
GPIOB->MODER |= (0x5555); //set pins 0-7 to output mode

delay_ms(20);

//for loop that changes program to 8 bit data mode. 3 times
for(int idx = 0; idx < 3; idx++){
    PutNibble(0x03);
    LCD_Pulse();
    delay_ms(5);    //Pause
}

//cursor starts at starting position. DDRAM address sets to 00H.
PutNibble(0x02);
LCD_Pulse();
//pulse
delay_ms(5);

//send 4bit bus commands
//5x8 dot matrix
LCD_WriteCom(0x28); //2 line display mode
//delay_ms(5);
```

```
        //clear display
        LCD_WriteCom(0x01);
        //delay_ms(5);

        //cursor off
        //cursor blink off
        LCD_WriteCom(0x0C); //display on
        //delay_ms(5);

        //cursor moves left to right,
        // no display scrolling
        LCD_WriteCom(0x06);
        //delay_ms(30);

        //CGRAM address starts at 0
        //LCD_WriteCom(0x40);
        //delay_ms(5);
    }

    void LCD_Clear(void){
        //clear LCD screen
```

```
        LCD_WriteCom(0x01);
    }

void LCD_DisplayString(unsigned int line, unsigned char *ptr) {
    if(line == 0) {
        LCD_WriteCom(0x80); //write on first line
    } else {
        LCD_WriteCom(0xC0); //write on second line
    }

    for(int i = 0; i<16 && ptr[i] != '\0'; i++) {
        LCD_WriteData(ptr[i]);
    }
}
```

```
void PutNibble(unsigned char c){
    if(c & 0x8)
        GPIOB->ODR |= DB7;
    else
        GPIOB->ODR &= ~DB7;

    if(c & 0x4)
        GPIOB->ODR |= DB6;
    else
        GPIOB->ODR &= ~DB6;
```

```
    if(c & 0x2)
        GPIOB->ODR |= DB5;
    else
        GPIOB->ODR &= ~DB5;

    if(c & 0x1)
        GPIOB->ODR |= DB4;
    else
        GPIOB->ODR &= ~DB4;
}
```

```
void LCD_Pulse(void) {
```

```
    GPIOB->ODR |= E;
```

```
    delay_ms(5);
```

```
    GPIOB->ODR &= ~E;
```

```
    delay_ms(5);
```

```
}
```

```
void System_Clock_Init(void){
```

```
    RCC->CR |= RCC_CR_HSION; //activate HSI clock
```

```
while((RCC->CR & RCC_CR_HSIRDY) == 0);  
}
```