CE-2812, Lab Week 3, LCD API

1 Purpose

The purpose of this lab is to write a useful API for interacting with a character LCD module along with enhancement of previous work.

2 Prerequisites

- The Nucleo-F446RE board had been mounted onto the Computer Engineering Development board.

3 Activities

3.1 Delay API

We wrote a delay subroutine for the last assignment that implemented millisecond delays using the SysTick timer. This will be extremely useful going forward. To make it easier to reuse, create a delay API. This will consist of a header file (delay.h) that will declare the global functions (aka prototypes) and any needed symbols (such as register addresses), and a source file (delay.c) that will define the methods. In addition to delay_ms(), you may benefit from a shorter delays, so also implement a microsecond delay, delay_us(). Any helper methods or shared (file static) variables should be declared in the source file only.

3.2 LCD API

The LCD API will provide essentially the same functionality as the version that may have been written in assembly language for CE2801. As an API, you will supply a header file with global function prototypes. The source file will contain the function definitions. Any helper methods or shared (file static) variables should be declared in the source file only.

As a minimum, supply the following global functions:

- lcd_init
    - Initializes I/O to interface to communicate with LCD module
    - Clears and homes the display
    - No arguments or return
- lcd_clear
    - clears the display
    - no arguments or return
    - includes necessary delay*
- lcd_home
    - moves cursor to the home position
    - no arguments or return
    - includes necessary delay*
- lcd_set_position
    - moves cursor to the position indicated
    - requires two arguments – a zero-based row, and a zero-based column, no return value
    - includes necessary delay*

- lcd_print_string
    - prints a null terminated string to the display
    - accepts the (pointer to) a null-terminated string, returns the number of characters written to display
    - includes necessary delay*
- lcd_print_num
    - prints a (decimal) number to the display
    - accepts the number to be printed, returns the number of characters written to display
    - includes necessary delay*

  * The datasheet for the LCD module lists the execution time for most operations. You must not send additional commands for data until the previous operation has completed. You should use your delay api to accomplish the necessary delays, consider polling the "busy flag" to know when a command completes, but this is not required.

The last two methods (lcd_print_string and lcd_print_num) require aspects of the C language that we have not yet gone through in lecture. These facilities of C will be covered early in Week 4 so that you will have time to add them to your otherwise complete project.

The last method, lcd_print_num, will benefit greatly from the standard library function sprintf(). Look it up (page 282 of The C Book)! **Do not use itoa(),** which, while available on our platform, is not part of the standard C library.

You may also benefit from a number of "helper" functions that actually interact with the LCD. These routines should be made file-scope by being declared static. Their prototype would not appear in the header file. Examples of these helper functions might be "lcd_print_char" to print a single character, "lcd_data" to send data to the LCD, "lcd_command" to send a command. lcd_print_char might actually be usable as a global function as well as a helper function.

Also be sure to use your own functions. For example, lcd_print_string could repeatedly call lcd_print_char, and lcd_print_num should call lcd_print_string.

One other optimization that you may consider is to have the pointers for writing to GPIOA (the LCD's data signal) and GPIOC (the LCD's control signals) at file scope. This will improve performance by not having to define and initialize the pointers every time a method is called, and if they are file-scope (static) they cannot be seen by other source code hence you avoid the global variable issue. They should also be pointer const and volatile even though you will not likely be polling them.

3.3 **OPTIONAL** KEYPAD API

This is optional!!!

We may want to be able to capture keystrokes from the keypad and use them in our programs. You should already have a good understanding of how this is done. To help reinforce the "API" layout, sample code has been provided for the keypad. You will need to implement the actual scanning routine – scan(). Not that this function is declared file-scope and is not intended to be called directly by users of the API.

3.4 DEMONSTRATION APPLICATION

In order to demonstrate the API, create a test and demonstration program of your own design. Can be simple, but must demonstrate most, if not all of the implemented LCD (and optionally keypad API functions). Ideas? A simple calculator, a programmable timer, a simple game…

3.5 Deliverables

When completed:

1. Submit to Canvas a **single pdf** printout of your completed source code to Canvas. **Include in a comment block at the top of your code a summary of your experience with this project.**
2. Ask to demo your lab to instructor. You can do this via writing your name on the board in the lab.
    a. If you demo during lab in Week 3, you will earn a 10% bonus on this lab. ← really not expecting this!!
    b. If you demo during lab in Week 4, you will be eligible for full credit.

- Demos are ONLY accepted during lab periods. If you are unable to demo by the end of lab in Week 4, you lose the 10% of the assignment attributed to the demo (per syllabus).
- Demos must be ready a reasonable amount of time before the end of the lab period. If you write your name on the board at 9:45 and lab ends at 9:50, and there are five names in front of yours, you will be unlikely to complete your demo by the end of lab and hence lose points.

3.6 Grading Criteria

For full credit, your solution must:

- Implement the delay API and the lcd APIs as described above, using the "API Layout" discussed in lecture. Given this, there should be at least three source files (delay.c, lcd.c, main.c) and two header files (delay.h, lcd.h) for this project.
    - Be sure to factor out "helper" functions in your apis and make them file-scope (static).
- Continue to use **pointer variables** for I/O register access. **Do not directly dereference macros** for register access. Be sure pointer variables are declared **volatile** and **const** as appropriate.
- Use macros to #define useful symbols and minimize "magic numbers."
- "Demo" program may reside in main() if you wish.
- Minor errors usually result in a deduction of ~ 3 points (three such errors results in ~ a letter grade reduction)
- Major errors, such as not achieving a requirement, usually result in a deduction of 5 to 10 points.