ECE271: Microcomputer Architecture and Applications — University of Maine

Lab #2: The Liquid Crystal Display (LCD)

Week of 31 January 2022

Goals

1. Understand alternative function of GPIO pins
2. Understand basic concepts of LCD driver, particularly **bias** and **duty ratio**.
3. Understand concepts of double-buffering to ensure coherency of displayed information.

Pre-lab

1. Complete the pre-lab before attending lab. The pre-lab is in a separate pdf file, found on the website.

Lab Procedure

The first part of the lab will be getting the LCD to display the first 6 letters of your last name by directly coding the hex values needed for the pins.

The second part is to create a routine that can take an arbitrary string of up to 6 characters, and display it to the display. This code will be reused in future labs.

There is template code provided on the course website for this lab. Download and use it as a basis for your lab. This code already contains the `LCD_Clock_Init()` code needed to initialize the LCD clocks.

Part A – Displaying your Name

1. For this part, edit the provided `LCD.c` (`lcd.c` on Linux)
2. Complete the `LCD_Pin_Init()` function.
   (a) First enable the clocks to GPIOA, GPIOB, GPIOC and GPIOD. The code to do this will be similar to the code from Lab 1.
   (b) Next, enable the GPIOA MODER register to put the pins we need into alternate-function (0b10) mode. See the prelab for the values to set.
   (c) Set the GPIOA AFR[0] and AFR[1] registers to have the proper pins be in LCD mode, which is 0xb (Alternate Function 11). See the prelab for the values to set.
   (d) Now repeat the above for GPIOB, GPIOC, and GPIOD, again using the values from the prelab.
3. Complete the `LCD_Configure()` function in the provided code.
   (a) This is based on the third rectangle in the flowchart Figure 17-10 in the textbook. I'll include the info below. You will have to look at the STM32L4 manual to get some of these pin settings; there's a guide to the registers involved at the end of the prelab.

   Also, if you look as the provided `stm32l476xx.h` or `stm32l.h` header files, you can use the definitions provided to make your code a bit cleaner. For example you can do things like:

```
LCD->CR |= LCD_CR_LCDEN;
```

instead of using raw hex values.

(b) First, set the LCD CR register, BIAS field, to have a BIAS of $\frac{1}{3}$

(c) Second, set the LCD CR register, DUTY field, to have a DUTY of $\frac{1}{4}$

(d) Third, set the LCD FCR register, CC field, to have a maximum (7) contrast.

(e) Fourth, set the LCD FCR register, PON (pulse on) field, to have a value of 7.

(f) Fifth, disable the `MUX_SEG` bit in the LCD CR register

(g) Sixth, set the `VSEL` bin in the LCD_CR register so the LCD is using internal voltage.

(h) Seventh, read the LCD SR register and wait for the FCRSF bit to become set (this is set when the FCR register has synchronized with the changes we've made)

(i) Eighth, enable the LCD by setting the LCDEN bit in LCD CR.

(j) Ninth, wait for the LCD to finish being enabled by waiting for the LCD SR register ENS (enable status) bit to go high.

(k) Tenth, wait for the LCD booster circuitry to become ready by waiting for the LCD SR register RDY bit to go high.

4. Now implement the `LCD_Display_Name()` function.

(a) First have it repeat waiting for the LCD SR register UDR (update display request) field to go *low*.

(b) Once it is low, now you can write your LCD data into the LCD RAM. You should have calculated these values in the prelab. Something like:

```
    LCD->RAM[0]=0xb4420370;
...
```

(c) Once you have set those values, set the LCD SR register UDR bit to high to indicate you wish to update the display.

(d) Finally, repeat watching the LCD SR register UDD (update display done) register. Once it goes high, you are done and your name should be on the display!

(e) You probably want an infinite loop at the end of your code to keep the program counter from escaping into un-initialized memory.

Part B – Writing Arbitrary Strings

1. Now we are going to write code that takes an arbitrary C string and writes it to the display. We will use this in future labs, as you can imagine it's useful to have a way to display output of programs we write.

2. First, in `main.c` uncomment the call in `main()`

```
LCD_Display_String("ECE271");
```

as this will be our test string.

3. Now we will implement the `LCD_Display_String()` function.

   (a) The textbook chapter 17.3.2 has info on how to do this. The textbook provides code, which we helpfully include for you in `LCD.c` so you don't have to type it in.

   NOTE I've made some slight changes to the code, one being we pass in the character to print as a character, not a pointer to a character.

   (b) All you have to do is write a loop that loops through the first 6 characters of the provided string (remember, a string in C is just an array of characters) and calls the provided

   ```
   void LCD_WriteChar(char ch, int point, int colon, uint8_t position);
   ```

   function.

   The first argument is the character you want to print, the second is if you want a decimal point (0 or 1), the third is if you want a colon, and the final argument is which of the positions on the display you want the character to appear at.

   The `LCD_WriteChar()` function handles lots of stuff for you, such as converting to uppercase, having a 14-segment font, mapping the font to the segments, and writing out to the display.

   (c) One thing you do need to do: if your string being printed is smaller than 6 chars, be sure to pad out the rest with spaces. Don't just display whatever random values might be past the end of the string!

4. Next implement `LCD_Clear()` that clears the display. This should be relatively straightforward.

Part C – Something Cool

Do something cool! You can come up with something on your own, but here is a list of ideas you can use.

1. Have the LCD display longer strings, but scroll when they are longer than 6 characters.
2. Use the joystick to scroll a string.
3. Use the joystick to adjust the contrast of the display.
4. Do some sort of cool animation using the LCD segments.
5. Use the joystick to step through a series of strings.
6. Something else that you come up with on your own!

Lab Demo

1. Submit your code
    - Complete a README with the post-lab (next page) answers.
    - Make sure the code is properly commented.
    - Check your code and README into your gitlab tree.
2. Demo your implementation to your lab TA.
3. Answer the following questions and show to the TA.
    - In the character display code, why do we

```
while((LCD->SR & LCD_SR_UDD)==0);
```

    - Explain why double-buffering can ensure the coherency of displayed information.

Post-Lab

- Place your answers to the question in a file Readme.md
- Submit with your code via the gitlab server.

1. Suppose the duty ratio of an LCD display is $\frac{1}{4}$ and it has a total of 120 display segments (pixels). How many pins are required to drive this LCD?
2. Is the LCD driver that drives the display built into the Cortex-M4 CPU, or does it live off-chip?
3. How many pixels can the STM32L4 processor LCD drive? How large is the LCD_RAM in terms of bits? (Look to the STM32L4 manual for answers).
4. How many pixels does the LCD installed on the STM32L4 discovery kit have? What happens to the extra LCD_RAM bits that are not hooked up?
5. If you only pass a 3-character long string to your `LCD_Display_String()` function, but your printing loop does not check for string length and tries to print characters 4, 5, and 6, what will get printed to the display there?

Table 1: This table might be useful if trying to figure out why certain pins aren't turning on.

	LCD					
STM32L Pin	**LCD Pin**	**COM3**	**COM2**	**COM1**	**COM0**	**LCD Pin**
PA7 (LCD_SEG4)	1	1N	1P	1D	1E	SEG0
PC5 (LCD_SEG23)	2	1DP	1:	1C	1M	SEG1
PB1 (LCD_SEG6)	3	2N	2P	2D	2E	SEG2
PB13 (LCD_SEG13)	4	2DP	2:	2C	2M	SEG3
PB15 (LCD_SEG15)	5	3N	3P	3D	3E	SEG4
PD9 (LCD_SEG29)	6	3DP	3:	3C	3M	SEG5
PD11 (LCD_SEG31)	7	4N	4P	4D	4E	SEG6
PD13 (LCD_SEG33)	8	4DP	4:	4C	4M	SEG7
PD15(LCD_SEG35)	9	5N	5P	5D	5E	SEG8
PC7 (LCD_SEG25)	10	BAR2	BAR3	5C	5M	SEG9
PA15 (LCD_SEG17)	11	6N	6P	6D	6E	SEG10
PB4 (LCD_SEG8)	12	BAR0	BAR1	6C	6M	SEG11
PB9 (LCD_COM3)	13	COM3				
PA10 (LCD_COM2)	14		COM2			
PA9 (LCD_COM1)	15			COM1		
PA8 (LCD_COM0)	16				COM0	
PB5 (LCD_SEG9)	17	6J	6K	6A	6B	SEG12
PC8 (LCD_SEG26)	18	6H	6Q	6F	6G	SEG13
PC6 (LCD_SEG24)	19	5J	5K	5A	5B	SEG14
PD14 (LCD_SEG34)	20	5H	5Q	5F	5G	SEG15
PD12(LCD_SEG32)	21	4J	4K	4A	4B	SEG16
PD10(LCD_SEG30)	22	4H	4Q	4F	4G	SEG17
PD8(LCD_SEG28)	23	3J	3K	3A	3B	SEG18
PB14 (LCD_SEG14)	24	3H	3Q	3F	3G	SEG19
PB12(LCD_SEG12)	25	2J	2K	2A	2B	SEG20
PB0 (LCD_SEG5)	26	2H	2Q	2F	2G	SEG21
PC4 (LCD_SEG22)	27	1J	1K	1A	1B	SEG22
PA6 (LCD_SEG3)	28	1H	1Q	1F	1G	SEG23