Lab 2: Interface Design for Embedded Systems

In the first lab you gained familiarity with the Arduino environment, learned how to use a BLE module as a wireless serial link (they are capable of a lot more than that by the way), wired a motor up to an H-bridge, and got basic controls working for a simple robot.

In this lab, we'll mostly build on what we've done so far, but shift to thinking about **interface design**. The first thing you will design is a C++ class to control a LCD where your goal is to create an interface that is easy for *programmers* to use and understand (i.e. you will design and implement an API[1] for the LCD). The second thing you will do is create a user interface for your robot. Your goal here is to focus on designing with the focus on the *user* rather than focusing on what is convenient for *you*.

1. Pre-lab

As you'd expect, this pre-lab is mainly about getting you a background needed to be successful with the lab, but it also involves doing some interface (both user interface and software interface) work before you make it to lab. ***All pre-lab answers must be typed.***

Part 1: LCD and interface design

In this lab you will be designing a software interface for a liquid crystal character display (character LCD or just LCD) for the Arduino. You are going to need to design the C++ *interface* as well as write the code to *implement* that interface. The basic idea is that you want to create a class that allows for easy interface to the LCD.

Q1. Consider the standard a standard 16x2 HD44780 character LCD with a 4 and 8-bit interface[2].

i) Describe each of the interface pins on such a board.

ii) Describe the command you'd need to turn on the display with the cursor underline off. Exactly what would have to be sent on each pin?

iii) This device supports both 4 and 8-bit modes.

(1) How do those modes differ?

(2) What is the advantage of using each mode? How would you pick which to use?

Implementation issues, such as those above, are certainly important. But let us also worry about the C++ interface that will be visible to anyone using the API. We want something that is easy for a beginning programmer to use, but allows for a full and flexible interface.

[1] As you probably know, API stands for "Application Program (or Programming) Interface" and is basically a library of functions that is independent of the implementation. In this lab we want you to focus on defining and implementing an API which is easy for the user to use rather than for the implementer to create…

[2] http://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf is the board's specification. You may also find https://www.sparkfun.com/datasheets/LCD/HD44780.pdf to be useful (especially for the start sequence.)

Q2. Design and document a class interface that allows for a simple, clean, interface to the 16x2 LCD. That is, define all the members (data and function) of your class *but you do not need to write the code that implements it at this time*. Think carefully about your interface: what might a user want to do? How might they wish to do it? What if they just want to print a message? What if they wish to just write a single character at a certain location? What about non-ASCII characters? Make your interface as flexible as you can without overly complicating things. **The quality of your *design* (and its later implementation) will make up a large part of your lab score for this lab.**

- For this question, we are just asking for the basics—getting characters to the screen in a reasonable way. Don't worry about scrolling, blinking and the like.
- This is to be an interface for a software person to use. It shouldn't *require* an understanding of the LCD or its registers. That said, a good design often provides a "backdoor" to be able to do things your library doesn't directly support.

This interface design should be one that you could hand off to another group and they could code it. ***You'll want to have access to an electronic copy of this pre-lab during the in-lab as you may be modifying it on the fly.***

Q3. Consider a different type of interface: a human controlling a robot with a keyboard. Describe how you would *ideally* want to control a robot with a keyboard. Describe an interface that would nicely allow for both direction and speed control. Specifically, the user should be able to navigate tight spots with a fairly high degree of precision while also easily moving rapidly across fairly open spaces. Specifically explain how your interface would work in enough detail that if you gave your specification to someone else and they implemented it, you could drive the robot without first asking questions (for example, tell us what occurs upon a button press and release). *Notice we are **not** asking for code of any type at this point. We just want a specification for how the user would use the keyboard to control the robot.*

Note: Many of you play video games on PCs. Think about the interface as a user, not as a programmer.

Q4. Bit banging…

i) What does it mean to "bit bang"?
ii) Is your implementation on the Arduino going to be done via "bit banging" or by use of dedicated LCD hardware?
iii) What dedicated hardware interfaces does the processor in the Arduino support?
iv) What are the primary disadvantages of bit banging?
v) For what types of interfaces might bit banging be reasonably acceptable?
vi) When might you bit bang even though there is dedicated hardware support for the interface in question?

2. In-lab

We'll continue working with the robot you started on in lab 1. Be aware that Part 1 is quite time consuming. The other parts aren't as bad.

Part 1: LCD interfacing

In the pre-lab, each student designed a C++ interface for the LCD (and each was asked to keep a copy). Now as a group the two of you need to decide how best to define that interface.

Q1. Redo the pre-lab question Q2 and come to a consensus about your LCD interface. This interface design should be one that you could hand off to another group and they could code it. You are to attach this interface to your lab write-up.

Implement your interface for the LCD as an Arduino sketch. We recommend you have the GSI look your design over before you start on your implementation—poorer designs will result in fewer points even if they are implemented well. Adjust the LCD contrast with a potentiometer until the black squares on the display are barely visible after you've powered the device; otherwise, the text you write will not be visible. Due to Arduino pins being used for other things, you will want to use the 4-bit mode.

There are two timing constraints that you need to be aware of in your implementation: command latency and setup and hold times for the display controller interface. Command latency is the time you will have to wait after issuing a command to the controller. You will see these times in the data sheet usually in the initialization sequence. Simply use the delay function and round up to the nearest millisecond. Secondly you need to consider the setup and hold times for the controller write register control lines. You will find these specifications in the data sheet as an annotated timing diagram. There are 3 signals to consider: RS, R/!W and E signals. The setup and hold times will be specified with respect to the other control signals and data that is to be written to the controller register. These times are usually on the order of micro seconds. These times are a minimum specification so you can use longer times. In fact, you can round up to the nearest millisecond. Lastly, the timing diagram will show you the order of the control signals. Not all the HD44780 data sheets have these timing diagrams. *Here is one that does.* You might find them referenced as read write cycles. You may want to check your write timing with the USB logic analyzer. Since the RS or R/!W lines may not change, you need to trigger on the E line. Make a simple code example that just does your write bit banging for as single data write cycle. Run the logic analyzer observing the RS, R/!W , E line and a few data lines. Then run the code and you should capture the write cycle to verify your code.

G1. Use your interface to write "ARDUINO RULES!" on line 1. Line 2 should change back and forth between "Roomba!" and "BLE 4ever" on line 2 (changing every 2 seconds). Show your lab instructor.

Q2. Print a copy of your C++ class and attach it to your answers.

Part 2: Extended robot control interface

Now you are going to combine three different things: your working robot, the Python from the previous lab, and your proposed interface for your robot. Discuss with your partner what that interface should be. Modify your robot's code to implement your proposed interface. At the least this should involve being able to change speed in a reasonable way (in addition to direction obviously). Use the same basic format for instructions to the robot.

G2. Show your GSI that your interface works and generally allows for solid control of your robot. Show you can navigate tight turns. Your GSI might set up a course and let folks compete. Maybe we'll give out a few small prizes.

Part 3: Python control of the LCD

Now modify both your Python code and your Arduino code so that you can request that relatively arbitrary messages be displayed on your LCD from the keyboard. You might want to have a key act as a toggle between driving and screen control.

G3. Demonstrate this to your GSI that you can display arbitrary messages via the Python interface while still being able to drive around.

Q3. Attach a copy of your python code to your answers.

3. Post-lab

Q4. What radio frequencies does Bluetooth utilize? Why are those frequencies used? Explain how frequency hopping helps Bluetooth devices communicate and why that's important give the frequencies Bluetooth uses.

Q5. What is a Bluetooth profile? Why are profiles used (hint: we are looking for an answer related to the primary focus of this lab…)?

Q6. Compare your LCD API to Arduino's LCD interface. How are they different? What do you like better about your interface? What do you like better about their interface?

Q7. It has been claimed that:

> **Embedded programmers often write code that is easy to write rather than code that is easy to use**

In fact, it can be reasonably argued that creating APIs that are easy to use has been the secret to Arduino's success. But what is meant by the above statement? And what is it about writing low-level device drivers that can lead one to write code that is "easy to code but not easy to use"?

Q8. Why do you suppose the LCD interface has to be done via bit banging? Why isn't there dedicated hardware support for our LCDs?