EENG 462: Embedded Real-Time Control Systems
Lab #1: Getting Started, LCD, Analog Switches

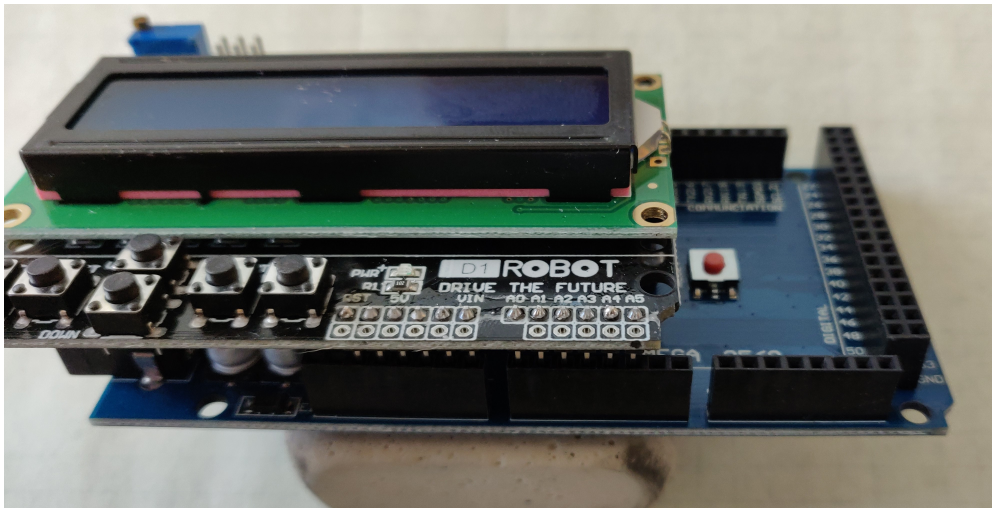
In this lab we experiment with the LCD/Keypad. Displaying alphanumeric characters on an LCD, including scrolling behavior, is quite easy using the LiquidCrystal software library. You can easily find documentation of that library by googling "Arduino LiquidCrystal."

Instead of using 5 digital input pins, this board reads all 5 switches (UP, DOWN, LEFT, RIGHT, and SELECT) via a single analog input (labeled A0 on your Arduino, and AD0 on the manufacturer schematic, which is posted with this assignment).

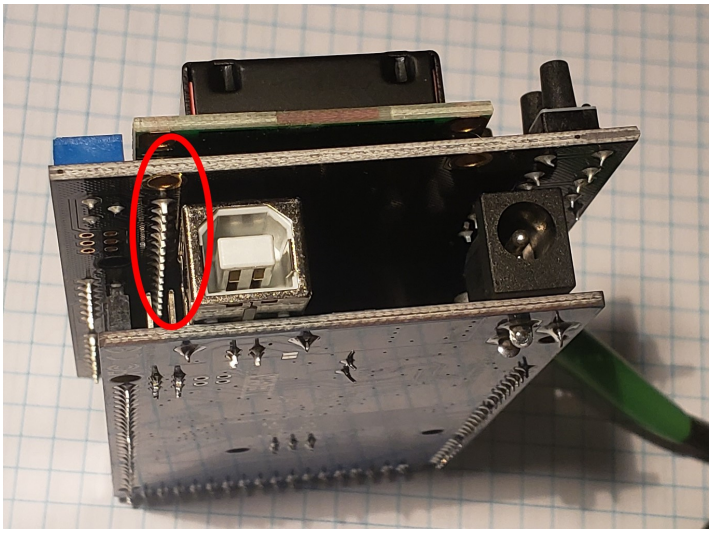
Note from the schematic at that the switches are connected in a network of varying resistor values. Different switches produce different voltage divider circuits, which change the analog voltage read at A0 on the Arduino. For this lab you'll predict those voltage values from the schematic and verify them experimentally.

Start by installing the Arduino development environment on your computer and get it working with the example code shown in lecture module 1. If there is anything you do not understand about how the code works, please ask me to clarify it in class.

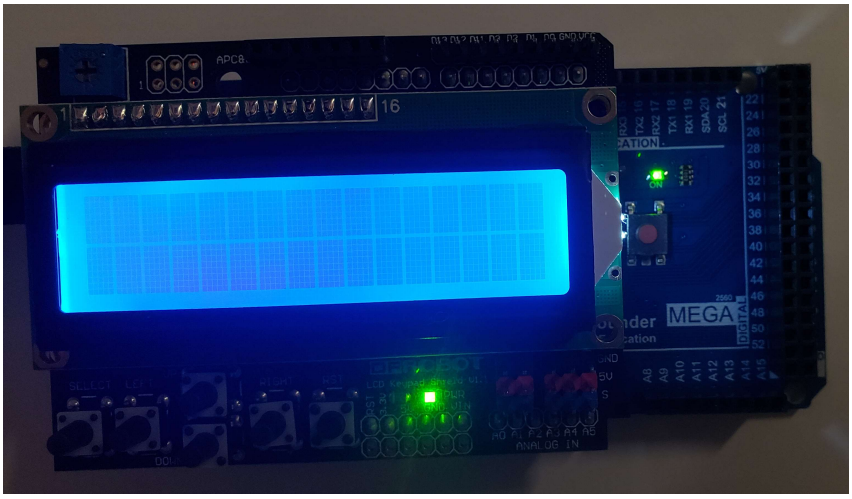
You'll need to connect the LCD/Keypad to your Arduino. With the Mega it's not obvious how to line up the pins. The easiest way to get it right is to look for the pins labelled A0 through A5 on the LCD/Keypad (right under the "DFROBOT label"). Those pins push into the connections on the Arduino with the same labels (note that the Arduino also has A6 and A7, which the LCD/Keypad does not connect to. There are connections on two sides, so be careful to get them all lined up before pushing them together. Here's a picture to help you with the alignment.



Once it's connected take a look into the USB connector end and make sure the pins circled in red on the following photo are not touching the USB connector casing (they shouldn't, but in the past we've received versions of this board where they did):



Power up the board by plugging it into a USB port. The LCD should light up, but it is highly likely it will just show solid blue light. If that's the case then you need to adjust the contrast (by gently turning the blue potentiometer) so that you can just make out the dot matrix for each character position, something like this:



Part A: Predict the Switch Thresholds

1) From the schematic, predict the voltage values for the 5 possible switch presses as well as the voltage when no switch is pressed.
2) Convert each voltage value to a number in the range 0-1023 by subtracting 0.0025 V, multiplying the result by 1024, dividing by 5, and truncating the result to an integer. As an EE you probably know why we're doing that, although you might think the calculation should be slightly different. If you have no idea, no worries, we'll be discussing it later.
3) Arrange the numbers in rank order and compute a value halfway between each.
4) Start a new Arduino sketch and enter those numbers in a comment near the top of the file.

Part B: Say Hello and Determine the Switch Thresholds

1) Start a new Arduino sketch. At the top, #include the <LiquidCrystal.h> library header. This library is included with the Arduino IDE, and contains functions that talk to a Hitachi HD44780 LCD controller, which is the controller used by the TC1602 LCD on the daughterboard/shield. Then #define a constant for the number of columns in your display (e.g., NCOLS) with a value of 16. Do the same for the number of rows (e.g., NROWS) with a value of 2. Then instantiate a global variable of type LiquidCrystal. This is one of the cases where Ardiuno reveals that it uses C++. You probably haven't had C++, but all you need to know is that you can instantiate a variable of type LiquidCrystal like this:

   LiquidCrystal lcd(a,b,c,d,e,f) ;

   Where a,b,c,d,e,f are values that initialize data members in the variable (which you can think of as a struct for our purposes).

2) Those values specify which Arduino pin numbers are connected to the LCD pins known as: RS, E, DB4, DB5, DB6, and DB7, in that order. You can obtain those Arduino pin name/numbers by following the schematic connections from the TC1602 to the J connectors, and then looking at a schematic of the Arduino or at the labels on the Arduino boards I/O connections. Fortunately, the makers of the shield labeled the traces going to the J connectors the same as the Arduino does on its side (almost). Just leave "D" off the trace names to get the Arduino pin name (D apparently stands for Digital input, A for Analog).

3) In your setup() function, call the LCD objects begin and clear functions (with dot notation, just like you use when accessing the members of a struct). Then print the message "`.....Hello......ADC.Key.Testing`" on the first line of the display, and "`.....World!`" on the second line (see the setCursor function). Don't take the periods literally, I'm using those to indicate space characters. Then call delay (500) to cause a half sec (500 msec) delay. You might want to note that the LCD has only 16 characters on a line, yet you just specified 31 characters for the first line. That's because the library builds in a capability to scroll virtual lines that are longer than physical lines. Bonus.

4) Add a for loop that calls the scrollDisplayLeft() function NCOLS times, with a delay of 500 milliseconds after each call. You've now coded an introductory sequence that scrolls the display on startup. Test and debug until it's working.

5) In your loop() function, set the LCD cursor to the start of the second line, call analogRead on A0, and print the value you get back (the LiquidCrystal print function is overloaded to accept various argument types and format them appropriately for you).

6) Download again. If you don't see any value on the second line, perhaps it is because you have scrolled such that the leftmost side of the display is now showing column 16, but you're printing at column 0 which is now offscreen to the left. When no button is pressed the value should be 1023 because the resistor divider produces 5V on A0, and the Arduino uses 10-bit Analog to Digital converters.

7) Try pressing all the buttons. You should never see a value larger than 1023. If you do, perhaps that is because you left a character on the display from a previous pass (the display retains anything that you write until you overwrite it).

8) Write down the value you get for each of the 5 button presses in rank order. Compute a value halfway between each and enter the result in a comment near the top of the file, just under the numbers you predicted in part A. The numbers should be comparable.

Part C: Test your switch thresholds

1) Modify your code to display a description (UP, DOWN, LEFT, RIGHT, or SELECT) instead of the analog value of the ***last key*** pressed. Leave the scrolling introduction. Note that it should display a description of the last key pressed even after it has been released. ***Suggestions*** for how to accomplish this follow...
2) Near the top of your file, #define a symbol for the number of key codes (e.g., NUM_KEYS) with a value of 5. Then declare an array of that many integers and initialize it with the halfway numbers you determined above, in increasing order. The last entry should be halfway between the largest key value and 1023.
3) Also declare an array of NUM_KEYS strings (char *), and initialize it to string descriptions of the keys, with entries such as: "`Select`".
4) Create a function called findKeyCode, or something else reasonably descriptive, that converts an Analog to Digital (ADC) value (0 to 1023) to an index (0 to NUM_KEYS-1) for the key being pressed. The strategy should be to search the threshold array until you find the ***first*** entry that is larger than the ADC value read on A0. The index at which that is found is the keycode. If no array entry exceeds the ADC value, then no key is being pressed (perhaps return -1 in that case). Pause here and test this function by replacing your previous print of the ADC value with a print of the returned keycode.
5) Write another function called something like checkForKey, which should return a keycode if a ***new*** keypress is found, or a -1 if the keycode ***hasn't changed since the last time this function was called***. This means you need a variable that records the last keycode. Do NOT use a global variable for that, because no other function in your program should need to see it. Use a local static variable instead, and if you don't understand how local static variables work ***in C***, **PLEASE** ask (if any of you know Java, be aware that static does not mean the same thing in C as it does in Java). ***I will be checking to see if you use any unnecessary global variables***. A suggested approach for this function is to read the ADC, convert that to a keycode by calling the preceding function, and compare that to the last keycode. If it is different, set the variable that records the last keycode and return that, otherwise return -1.
6) Modify your loop function so that it calls checkForKey and prints (to the LCD) the corresponding key description from the array ***if*** it returns something other than -1.
7) If you test the results rigorously, you might be able to produce an occasional glitch in the keypress output (try several short and several long press-and-releases). To fix that, insert a 10 msec delay in your loop to make sure you don't check for a key press more often than once every 10 msec (which will allow any bouncing from the last check to dissipate). You could also put that delay in your checkForKey function.

Turn in your code for Part C. Please use the following file naming standard: phschimpfLab1.ino. That means your first and middle initials followed by your last name followed by the Lab and the lab number. Don't turn in a file that uses my name (you'd think I wouldn't have to say that, but experience tells me otherwise).