PHYS432 – Lab 6: Arduino

Parts List

- Arduino Nano
- Other: Button, LED, 1 kΩ resistor, 10k potentiometer, LCD display

6.1 Introduction

In this lab, we will explore the Arduino microprocessor platform. The Arduino is really a set of board designs along with a development environment intended to remove some of the technical complexity that historically surrounded developing microprocessor applications. For the board we are using, the Arduino Nano, the actual microprocessor is an ATMEL ATMega328p, although the Arduino programming environment hides almost all of the processor-specific details.

This lab is much less of an investigation of some specific concepts on their own, and much more a set of examples for how to do things we have previously discussed in the Arduino platform. Each section is matched to one of the tasks listed on the course website. I have tried to avoid repeating text in this writeup, so you will need to read through the tasks on the course website as you proceed through this lab.

To help you find the specific questions that I would like to see answered, I have put them in **bold font**.

6.2 Arduino IDE

If you have not done so already, work through the Arduino IDE section. Note that you do not need to put your Nano into the breadboard to complete this preparatory task. Please don't skip this part, as it is important to verify that your development environment and communication with your Nano board is working before you start with the rest of the lab. **Describe the computer you are using to talk to your Arduino, especially the operating system. Also mention if you are using the web-based development environment, or downloaded the IDE to your computer. Finally, specify whether you needed to select ATMega328P or ATMega328P (Old Bootloader) to talk to your board.**

6.3 Task 1: Digital Input and Output

This task introduces the techniques for reading and writing digital information through the Arduino digital I/O pins. Work through Task 1 and make sure you can get your LED to light when you press the button.

As a test to see if the Arduino inputs act like TTL or CMOS gates, change the line `pinMode(BUTTON_IN, INPUT_PULLUP);` to `pinMode(BUTTON_IN, INPUT);`. **Now what happens when you push and release the button?**

6.4 Task 2: Serial Console

This task introduces communication over the USB serial link from the Arduino to the programming computer. Work through Task 2 and make sure you can see text messages on the Serial Monitor being sent back from the Arduino.

Given that the serial port can send approximately 9600 bits per second, how many lines of text do you expect you would see per second if the Arduino were able to saturate the serial port bandwidth? Try this out by removing the `delay()` call from the `loop()` block and let the processor write data as fast as it can. **How many lines (or characters) per second do you actually observe?**

It may be difficult to read the numbers as the text is scrolling by, but you can always unplug the board after some time to stop the text scrolling by. Also, there is a clear button on the Serial Monitor which can be helpful for this. You don't need to be very precise here. An order-of-magnitude estimate of the rate is enough.

6.5 Task 3: Analog to Digital Conversion

This task introduces how to use the internal ADC in the Arduino to digitize analog voltages.

With the ADC reference voltage set to +5 V, **what is the expected voltage resolution of each bit of the ADC? In other words, what is the smallest change in voltage that the ADC can detect?** Set the potentiometer to give a voltage around 2.5 V and watch how the successive readings from the ADC vary. **How much noise does there appear to be on the voltage reading? In other words, do the ADC readings fluctuate up/down by several bits, or are the fluctuations at or smaller than one bit?** You may find it easier for these investigations to use a fixed delay time rather than a delay that is proportional to the ADC reading.

If we want to improve the ADC precision, we need to reduce the dynamic range. Select the internal 1.1 V reference and repeat the exercise above. **What is the single-bit precision with the 1.1 V reference?** Don't forget to change the maximum voltage range in the sketch so that the ADC readings are calibrated correctly. Again, set the analog voltage level to approximately the midpoint of the ADC range (around 0.5V) and observe the fluctuations as the input voltage is kept constant. **Now how much does the ADC reading fluctuate? How large are these fluctuations in millivolts?** You will have to estimate this based on the fluctuations you see scrolling by. **How does this compare to the noise level we saw on the AD557 DAC in Lab 5? In other words, is the Arduino doing much better, much worse, or about the same as our Lab 5 SAR ADC?**

6.6 Task 4: Liquid Crystal Display

Carefully wire up your LCD display following the instructions on the website under Task 4. After uploading the sketch to the Arduino, adjust the potentiometer as described on the website to set the contrast properly. **Describe whether you just needed to turn the potentiometer all the way up, or whether you needed to find some other setting to make the characters legible.**

We specify text as a `string` data type, but this is really just an array of 8-bit unsigned numbers encoded using the ASCII standard. If you wanted to write a degree symbol, **what is the ASCII value for this character?** Try writing this character to the LCD display using something like `lcd.print(number)` where `number` is the actual number (no quotes) you found in the ASCII standard for the degree symbol. **Does this work?**. You can also try explicitly converting the number to a `char` type first with `lcd.print(char(number))`. **What does this give you?** Look up the character set in Table 4 (page 17) of the Hitachi HD44780 manual linked from the Task 4 page. **What number is actually used for the degree symbol, or at least something that might look like the degree symbol?** In the table this is specified as a high and low bit pattern, but you should be able to convert this to a decimal number. You can also specify hex numbers in the Arduino IDE using the `0x` radix. So in other words, `0x12` would specify the decimal value 18 (16+2). **Using the value you found in table 4, do you get a degree symbol on the LCD?**

6.7 Task 5: Button Debouncing

This task is optional, and you only need to do this part of the lab if you are interested. If you do, however, I am curious about how your board and button behaves.

Work through Task 5. Before debouncing your button **how frequently do you get bounces?** In other words, when you use the `task5a` sketch, does your switch bounce all the time, rarely, or never? When there are bounces, you can get a sense for how many by looking at how far the counter increments on each physical button press. **Do you usually just see one extra count, or many?**

After implementing the button code in the `task5b` sketch, **does your switch still occasionally bounce?**

6.8 Wrapping Up

You now have seen some basic functionality from the Arduino Nano. These functions should be the building blocks of your final (mini-)project. Don't be afraid to try things out, and remember there are lots of Arduino examples available on the internet that may give you other ideas or solutions for how to do specific things.