Lab 10

Digital Electronics: ARDUINO

Department of Physics | University of Colorado Boulder

2022-12-07

Contents

1 Goals

In this lab, we will get a sense of the capabilities of Arduino microcontrollers. Microcontrollers bridge the gap between digital electronics and full-fledged computers. They can be versatile tools as well as lots of fun!

2 Definitions

FPGA – "Field Programmable Gate Array." A single IC containing an array of logic gates that can be "wired together" by a user after manufacturing using programing code.

Microcontroller – A small computer on a single IC that includes a processor, memory, and input/outputs.

IDE – "Integrated Development Environment." Software specifically designed for writing programs in a certain setting or context (for example, a specific programming language).

IC – "integrated circuit" or "chip"; a packaged set of electronics, based on transistors, resistors, etc.

Sketch- Arduino name for a program or instructional code written on a computer input to the Arduino chip.

3 Programmable Circuits - Introduction

Suppose you want to build something more complex than the digital circuits we studied in the previous experiment. For example, you might have a photodiode that you want to hook up to a computer so you can automatically adjust your experiment in response to changing light intensity, maybe by adjusting the current powering a laser. Or, say you want to build a tiny, battery-powered data transmitter to feed to a dolphin and find out how her body temperature varies while she's swimming around in the ocean. Since it's hard to transmit signals through a dolphin, the transmitter should store the temperature data until it is excreted by the dolphin and floats to the surface where your receiver can detect the signal.

These and many other applications require complex digital circuitry. Any digital system, including a programmable computer, can be built by combining the gates and flip-flops we discussed in the previous experiment. But it is hardly practical to build something like a computer by connecting a bunch of TTL chips, since you could easily require thousands or even millions of gates. There are much better ways to get the job done. In fact, if you find yourself using more than a few discrete logic gates in a circuit you are probably using the wrong tools.

4 Filed Programmable Gate Arrays and Microcontrollers

Sometimes it is worth designing an application-specific circuit and having it manufactured on a single chip. However, this is mainly practical only for large-scale operations and not feasible for the specific requirements of a single scientific experiment. Instead, scientists make use of Field Programmable Gate Arrays (FPGAs). These are chips containing many logic gates, and in most cases also flip-flops, which can be programmed (sometimes only once, sometimes many times) to interconnect the gates and flip-flops in a way that performs a custom function (see H&H 8.15, 8.27). A single FPGA can replace a large pile of TTL chips, and the smaller FPGAs containing a few thousand gates are inexpensive and relatively easy to program. The largest FPGAs can contain 300,000 gates or more, and the programming effort can become a major project.

But by far the most common approach to complex digital design is to use a microcontroller, a single chip programmable computer. Compared to most FPGAs, microcontrollers process data more slowly but with much greater flexibility and power, and they are generally cheaper and more power efficient. They are available in incredible variety, from 8-pin versions costing less than a dollar to devices with processing power rivaling that of a PC.

5 Arduinos

In this lab, we will take a look at what microcontrollers can do by working with the Arduino Uno board. "Arduino", as they say at their website "… is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments." Arduino has been highly successful in bringing out simple microcontroller boards that can be programmed easily. These boards have a huge international following that has resulted in a wide range of books, publicly available programs, robotics kits, and the like. These resources make the Arduino a common and easy choice when thinking about a microcontroller project because it's very likely that someone has already solved your problem and has posted the solution code for FREE on the Arduino website (or elsewhere on the internet).

The heart of the Arduino Uno is the 28 pin ATMEL ATmega328P. It is a member of the ATMEL 8-bit AVR microcontroller family, with 32 kB of 8-bit wide program memory, 1kB of electronically erasable and programmable read-only memory (EEPROM), and 2 kB of 8 bit data memory. Program memory uses 'flash' technology, so you can program it over and over again as you debug your hardware and code, and it is non- volatile, so the chip won't forget its programming when you turn the power off. Programming can be done without taking the chip out of its socket on the Arduino Uno board via a serial digital USB interface. You download a program development environment from the Arduino

site, plug the Arduino Uno board into a USB port on your laptop, and start uploading programs to the Arduino microcontroller.

Nearly every pin of the ATMEL ATmega328P microcontroller chip can be programmed in multiple ways for various uses, and up to 23 of the 28 pins can be used as bi-directional digital I/O. There is an internal, variable frequency clock and three separate timer/counter/scalers. For dealing with analog signals there is a comparator with a programmable voltage reference, 6 channels of pulse-width-modulated (PWM) output to simulate adjustable external analog voltages, and to top it all off, a 10-bit analog-to-digital converter with a 6- channel analog multiplexer, so you can digitize 6 separate analog signals. Amazingly, you can have all this for around $3 per chip.

6 Useful Readings

1. http://arduino.cc/
    - The Foundations page is helpful: https://www.arduino.cc/en/Tutorial/Foundations
    - The Language Reference page is particularly good to know about: http://arduino.cc/en/Reference/HomePa
2. The multitude of internet resources on Arduino projects:
    - http://arduino.cc/en/Tutorial/HomePage
    - http://playground.arduino.cc/Projects/Ideas
    - http://www.instructables.com/id/Arduino-Projects/
    - …and many more…just Google it!

7 Prelab

Answer the following questions.

7.1 Set up the Arduino software

1. On a laptop with a USB port (which you and/or your lab partner should bring to lab), go to
2.
3.
4.

7.2 Download sample code

1. Download the starter sketch file labeled “arduino_starter_sketch.ino” from Canvas.
2. Open it in the Arduino IDE
3. Read through it and understand what is does. See the “Language Reference” section of the Arduino website NEED LINK VERIFICATION. Describe these three commands:
    1. pinModel()
    2. loop()
    3. digitalWrite()

7.3 Prepare to explore & lab

1. Calculate the resistor you should put in series with the LED in Step 1 to limit the current to 20mA (from a 5 V output and assuming a 2 V drop across the LED).
2. Draw the schematic of the circuit diagram for the potentiometer in Step 2 and how it acts as a voltage divider for input to the Arduino.
3. Decide on two simple projects that you can do for Step 4 of this lab (see below). You’ll only need one in the end, but it will be good to have two options in case one doesn’t work out. You can use internet resources such as those listed under item 2 of “Useful Readings” or look at the appendix.

7.4 Lab activities

1. Read through all of the lab steps and identify the step (or sub-step) that you think will be the most challenging.
2. List at least one question you have about the lab activity.

8 Arduino Output and Input

8.1 Flashing LED

1. Follow the instructions here: When you finish, you should see a yellow light on the Arduino board that blinks at 1 Hz.
2. Open up the starter sketch that you downloaded from Canvas for Question 1 of the prelab in the Arduino IDE.

3. The Arduino Uno has 14 digital pins that can be used for either input or output (though Pin 0 and Pin 1 are reserved for specific purposes). Determine which pin the sketch expects an LED to be connected to.
4. Configure a single LED (HLMP-C625) and resistor on the breadboard such that they are in series and one end is connected to ground. The resistor is added to limit the current in the LED to 20 mA. You should have calculated the resistor value in prelab question 3a.
5. Connect the Arduino digital pin found in step c and the ground pin to the breadboard such that you have a complete circuit with the LED and resistor.
6. Run the starter sketch and describe what happens.
7. Modify the sketch to make the LED flash at a frequency of 4 Hz. (Hint: Check out the "delay" command and/or look at the Blink sketch you should have used in a.) Document your changes in your lab book.
8. Modify the sketch to make the LED flash at a frequency of 4 Hz but to be on for ¾ of the time and off for ¼ of the time.

8.2 Potentiometer input

1. The Arduino Uno has 6 analog input pins, which convert an analog input (between 0–5 V by default) into a digital value with 10 bits of resolution. Connect your potentiometer so that you can control the voltage to one of the analog input pins. (Hint: use the +5 V pin on the Arduino as well as the ground pin.) Be sure to include your circuit diagram in your notebook.
2. Add code to the sketch to read the value on the input pin and store it as an integer. (Hint: check out the "analogRead" command)
3. Output the stored value of the voltage drop across the potentiometer to the computer over the Serial connection. (Hint: Check out the "Serial.begin" and "Serial.println" commands.) 9600 bits per second is a standard rate for serial communication.
4. Load the sketch and determine the range of values achievable with the potentiometer.
5. Modify the sketch to make the frequency of the flashing LED change based on the setting of the potentiometer. Use a frequency range that covers the range in which you would describe the LED as "flashing" (that is, not too fast and not too slow). Document your changes in your lab book.

9 Control Structures

9.1 10-LED bank

1. Place the 10-LED bank (MV57164) across the center of your breadboard.

2. Connect each LED to a digital output of the Arduino (HINT: Avoid pins 0 and 1, as these are reserved for direct communication with the microcontroller chip.)
3. Connect the other end of the LEDs together (use one of the columns). Then connect that common connection to ground with a single resistor (of the value calculated in prelab question 3a. This way you don't need 10 resistors.
4. Modify your code to make all the LEDs flash together. (Hint: Instead of controlling each one with its own line of code, use for loops to do the same thing to all of them in turn. You'll need to set the pinMode of each output to OUTPUT in the setup routine. Document your changes in your lab book.)
5. Now, further modify your code to make each LED flash in sequence. Document your changes in your lab book.
6. Include the final version of your sketch in your lab notebook.

10 Explore - Choose Your Own Project

Now that you have a taste of what Arduinos can do, pick a simple project and do it! It can be an extension of the flashing LEDs you just made, or something completely different. Feel free to use any of the components we've used in previous labs. Document your goals, approach, progress, and results in your lab notebook. Remember that the lab notebook should be a complete record which someone else could use without difficulty to reproduce what you did. If you are having trouble deciding on a project, there are some suggestions in the appendix.

11 Appendix: A Few Project Ideas

- Adjust your current 10-LED setup so that it counts up in binary.
- Use the potentiometer to control the brightness of the LEDs using the concept of Pulse Width Modulation (PWM). As long as the LED is flashing faster than about 60 Hz, it will simply appear solid. You can flash with a varying duty cycle to make it brighter or dimmer. You can do this yourself or with the "analogWrite" function. You could replace the potentiometer with the input from the 3.5mm headphone jack plugged into a device that is playing music so that the LED brightness depends on the volume of the music.
- Connect your speaker to one of the digital outputs. Use the "tone" function to play frequencies. You can control the frequencies with the potentiometer. Or, you can use the photometer circuit from Lab 7 to provide the voltage so that the sound frequency is dependent on the light level. Or you can use the "random" function to play random frequencies.

- Generate a periodic square wave with the 555 Timer circuit from the last lab but configure it to use a potentiometer and a resistor to allow for a range of frequencies. You should power the 555 Timer with the +5 V from the Arduino. Feed the output of the 555 Timer circuit into the Arduino and measure the length of time that the pulse is high and low (see the "pulseIn" function). From that, calculate the period, frequency, and duty cycle. Print out the information and see what happens when you vary the potentiometer.