DIGITAL ELECTRONICS: ARDUINO

LAB 10: INTRODUCTION TO MICROCONTROLLERS

GOALS
In this experiment, 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, and are also a lot of fun!

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” - 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

PROGRAMMABLE CIRCUITS - INTRODUCTION
Let’s 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 together 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.

FIELD 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.
ARDUINOS

In this experiment, we will take a look at what microcontrollers can do by working with the Arduino Uno board. “Arduino”, as they say at their website (http://www.arduino.cc/) “…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 ATMEGATMega328P. It is a member of the ATMEGAV8-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 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, just a bit more than you pay for the single discrete 2N4416A JFET we used in Experiment 8.

USEFUL READINGS

   a. The Language Reference page is particularly good to know about: http://arduino.cc/en/Reference/HomePage
2. H&H 8.15, 8.27 on FPGAs, for some background
3. The multitude of internet resources on Arduino projects:
   c. http://playground.arduino.cc/Projects/Ideas
   e. ...and many more...just Google it!
### LAB PREP ACTIVITIES

Answer the following questions using Mathematica or do them by hand in your lab book.

<table>
<thead>
<tr>
<th>Question 1</th>
<th><strong>Set up the Arduino software</strong></th>
</tr>
</thead>
</table>
| a. On a laptop with a USB port (which you or your partner should bring to lab), go to [http://arduino.cc/](http://arduino.cc/)
| b. Click on the "Download" tab
| c. Download the installer for the Arduino 1.0.6 IDE that is appropriate for your computer (Windows, Mac, Linux)
| d. Install the software on your computer |

<table>
<thead>
<tr>
<th>Question 2</th>
<th><strong>Download sample code</strong></th>
</tr>
</thead>
</table>
| a. Download the starter sketch file labeled “starter_sketch” from the course website (under “Useful docs”)
| b. Open it in the Arduino IDE
| c. Read through and understand what it does. See the “Language Reference” section of the Arduino website and internet resources listed above under item 3 of “Useful Readings” in this lab guide. Describe each command:
|   I. pinMode()
|   II. loop()
|   III. digitalWrite();
|   IV. delay() |

<table>
<thead>
<tr>
<th>Question 3</th>
<th><strong>Prepare to explore &amp; lab</strong></th>
</tr>
</thead>
</table>
| a. Calculate the resistor you should put in series with the LEDs in Step 1 to limit the current to 20mA (from a 5 V output).
| b. 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.
| c. Decide on a couple 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 some options to choose from once you’ve worked through Steps 1-3. Again use internet resources listed above under item 3 of “Useful Readings” of this lab guide. |

### ARDUINO OUTPUT AND INPUT

<table>
<thead>
<tr>
<th>Step 1</th>
<th><strong>Flashing LED</strong></th>
</tr>
</thead>
</table>
| a. Open up the starter sketch using a text editor, which you looked at in the Lab Prep. Copy and paste this in to your Arduino sketchpad.
| b. You may need to change the serial port so your computer connects to the Arduino board. You can find this under Tools.
| c. 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.
| d. Connect an LED between that pin and the GND pin on the Arduino (with the correct polarity). Make sure to limit the current through the LED by putting a resistor in series with the LED to allow only around 20mA of current to flow. (Hint: Don’t forget about the voltage drop across the LED?)
| e. Instead of connecting the LED and resistor directly on the Arduino, it will probably be helpful to connect the Arduino pins to your breadboard and use the breadboard for the LED. You’ll want the room on your breadboard as the circuit becomes more complicated. |
### Step 2

**Potentiometer input**

| a. | The Arduino Uno has 6 analog input pins, which convert an analog input (between 0-5V by default) into a digital value with 10 bits of resolution. Connect a potentiometer so that you can control the voltage to one of the analog input pins. |
| b. | 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) |
| c. | 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. |
| d. | Load the sketch and determine the range of values achievable with the potentiometer. |
| e. | 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. |

### Step 3

**10-LED bank**

| a. | Obtain a bank of 10 LEDs and place it across the center of your breadboard. |
| b. | Connect each LED to an output of the Arduino (HINT: Avoid pins 0 and 1, as these are reserved for direct communication with the microcontroller chip.) |
| c. | 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.) |
| d. | Now, further modify your code to make each LED flash in turn. (HINT: You’ll need a way to keep track of which LED should be “on.” Use an if statement in the for loop.) Document your changes in your lab book. |
| e. | Include the final version of your sketch in your lab notebook. (NOTE: Often programs you write in lab are too long to include directly in your lab book, so it’s better to make a note of where they are saved so that someone can go find them later. However, in this case your code shouldn’t be very long, so it makes sense to include it directly.) |

### EXPLORE!

**Choose your own project**

| a. | 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. |