

Lab: Introduction to Arduino

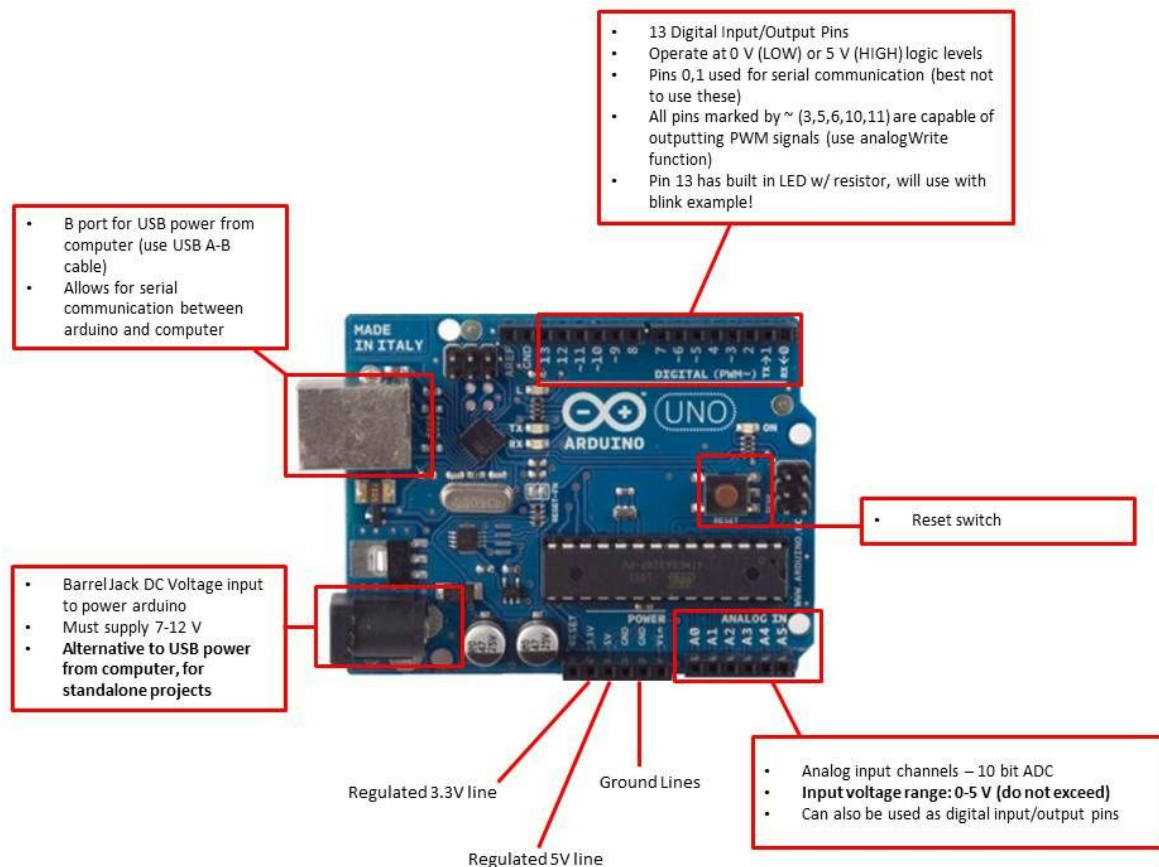
BioE 101, Spring 2016

Introduction to Arduino: (From arduino.cc/en/guide/introduction):

An arduino is an open-source physical computing platform made up of two parts: a simple microcontroller board and a development environment for writing software to the board. Arduinos can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your computer. The programming language for the free, open-source IDE is an implementation of Wiring, with libraries written in C++.

There are many very attractive features of arduinos, including low board cost, easy-to-use programming environment, and virtually limitless supply of online resources and tutorials due to its immense popularity and fan-base.

Hardware - Arduino Uno Board: We will be using the Arduino Uno Rev 3 boards, with a schematic and overview of the various pins shown below:









Connecting the Arduino:

- Using a standard USB cable (A plug to B plug), connect the Arduino to computer.
- Launch the Arduino application
- Under the 'Tools>Board' menu select the type of board you're using (Arduino Uno), and under 'Tools>Serial Port', select the serial (USB port) that your arduino is connected to (there should be only one option).

Blink Example:

- Open the blink example from under 'File > Examples > Basics'.
- Take a minute to look at the IDE (integrated development environment) that the Arduino app offers.



- Buttons on the top:      and 
- Check mark button: Verification
 - In contrast to Python and Matlab, Arduino code must be compiled prior to running. Briefly, a compiler takes the human-readable code you've written, and converts it to machine-readable code.
- Right-Arrow: Upload
 - This will upload your program to the Arduino board and begin running it. Do this after compiling your code.
- Magnified Glass: Serial Monitor
 - You will be using this extensively in the next sections -- this is similar to the command prompt in Python and Matlab (and something that is missing in Labview). You can print statements to the Serial Monitor during execution of your code using the **Serial.print("statement")** command. This is an essential debugging tool and means of reading sensor values.
- Now, compile and upload test code "blink" to ensure board is working. While the code is uploading, you should see the 'TX' and 'RX' buttons on the board blinking. Once the code has been uploaded, you should see the LED next to Digital Output pin 13 blinking. If the upload is successful, the message "Done uploading" will appear in the status bar.
- Let's look at the code more carefully to understand the basic structure of arduino sketches (programs). There are 3 main sections of code:
 - 1. Variables declared
 - It is good practice to declare all variables used in the code at the beginning of your file in either of the following formats:
type variableName;
type variableName = initialValue;

- Types can vary (bool, integers, floats, etc., see [sparkfun ref](#) for details on types that can be declared)
- In this example, pin number 13 is assigned as an int.

- 2. Setup function
 - This setup function is run only once, when the sketch begins
 - Here digital pins are configured to be inputs or outputs (see blink for example) using the pinMode command
 - Here also serial communication is started (will see this later)

- 3. Main loop function
 - Main execution of code, which loops continuously over and over.
 - Here, digitalWrite commands are used to set pin 13 to 5V (HIGH) or 0V (LOW) (turning the LED on and off). The delay command is used to control how long the light is on and off. The argument for the delay command is milliseconds.

- **Using Delays**
 - In order to control the timing of your program (loop time, sampling rate) you can use the 'delay' function
 - Example: Notice how in the blink example there is a delay of 1000 ms set at the end of the loop. Try removing this line. What happens? Why ?
 - Other functions you can use related to timing are the 'millis()' and 'micros()' functions. Both of these give you absolute time in millisecond and microseconds respectively. These will be useful later for determining sampling rates.

- Now, we will hook up 5 LEDs to the arduino using a breadboard, resistors, and hookup wire. Each LED should be in series with a resistor and connected to a separate digital output pin (9-13 are good choices). Connect the ground pin from your arduino to a negative power rail on the breadboard for your ground line. You should use resistor values around 330 Ohms. Make sure the LED is plugged in correctly with regards to the polarity! The anode (positive) side typically corresponds to the longer lead. If you plug it in backwards, you will likely break it >:(

- Now modify the code to control the outputs of all 5 LEDs that you connected to the board. You will need to add additional variables for each of the pins that the LEDs are connected to, add in additional pinMode commands to configure each as an output, and include additional digitalWrite and delay commands to control each one individually. Make a pretty pattern with your lights!

- **Show the GSI that your LED display is working correctly before moving on.**

Analog Input and Analog Output:

- Open the AnalogInOutSerial example from 'File>Examples>03.Analog'
- Note the new commands used in the setup and loop functions:
 - In the setup function, serial communications are initiated using the serial.Begin(baudrate) function, where the baudrate is typically set to 9600.
 - In the loop function, the analogRead command is being used to read the voltage at the A0 pin. This value is then mapped from an initial range of 0-1023 to 0-255 using the **map** command. The converted value is then written to the PWM pin 9 (such as for controlling the intensity of an LED), and the raw and converted values read in by the analog read channel are sent to the computer over serial using Serial.print() and Serial.println() commands (the only difference between the two is that Serial.println() ends in a newline). Finally, a delay is implemented before the loop starts over.

Using Analog & Digital Pins in Code

- The Analog in (A0-A5) can be read from using the 'analogRead' function. The value returned when reading from an analog input channel will be a voltage between 0V - 5V expressed as an integer ranging from 0 to 1023 (due to the 10-bit ADC).
- On the Arduino board, there are not special Analog Output channels, but the Digital Output channels can use PWM (pulse-width modulation: quantized analog signal) to act as 'analog'. Only channels 3, 5, 6, 9, 10, 11 can do PWM (indicated by the ~ next to the channels). To write to these channels, use analogWrite(pin_number, outputValue) where outputValue can range between 0 and 255 (these channels have an 8-bit ADC). You can use this to control the intensity of your LED!

Making an LED Pattern controlled by a Thermistor

- Now we will control the 5 LEDs you have already hooked up with an analog sensor (thermistor) input.
- You will be connecting your thermistor in a voltage divider. Look up the B57164K472J thermistor online to figure how to connect it to the Arduino. You can use a 4.7k resistor for the divider.
 - Use the 5V power supply on the Arduino board and the GND to power your voltage divider.
 - Connect the output of your circuit to A0
- Using the AnalogInOutSerial example, read in, and serial.Print the value of your sensor reading. You will need to convert the raw voltage measurement to temperature (use the excel sheet with fitted curve provided - *ArduinoLab_Thermistor.xlsx*). You should then print out the converted temperature in both Celcius and Fahrenheit over serial, with each value properly labeled. Upload your program.

- Once your code has successfully uploaded, click on the magnified glass on the top right of your IDE to open the serial monitor. Here, you should see values for your sensor input and analog output printing. Make sure your temperature readings make sense!
- Try squeezing your thermistor to heat it up -- confirm your sensor input value changes
- Now, adapt the AnalogInOutSerial code to write to the 5 Pins that your LEDs are connected to (can copy some of your code from previous part)
- Decide how you want your temperature to be displayed by the LEDs and use [if / else](#) statements to turn on / off specific LEDs depending on the temperature value. Some hints:
 - the **map** function is very useful if you would like to use analogWrite to control the LED intensity (remember, only possible on PWM channels):
 - Say you want an LED to be on only when the sensor input value is less than 200. You can use if statements and map(sensorValue, 0, 200, 0, 255) to map only input values that fall between 0 and 200, but would still take advantage of the full dynamic range of the output.
- **Show the GSIs your temperature to LED display before continuing.**

Interfacing with other sensors

- There are some [force sensors](#), [flex sensors](#), and [photocells](#) available for you to play with. Choose one and look up online how to correctly interface it with arduino. Use them in place of or in conjunction with the thermistor to control your LED display. Make sure that the readings you take from your sensor make sense - use serial commands to check!
- Use delay/millis/micros commands to control the sampling rate. Take a measurement every 1 Hz, 2 Hz, and 10 Hz. Figure out what your maximum sampling rate is! (Note that you are limited by the other commands in your loop. In particular, Serial commands can be very slow, so it may be worthwhile to disable extraneous serial commands if you want to achieve faster sampling rates).
- **Show the GSI that your sensor correctly interfaces with your arduino and LED array, and that you can control the sampling rate before moving on.**

Direct user input and actuation using a potentiometer and servo:

- Obtain a rotary potentiometer from the parts bin, and look up online how to connect it to the arduino analog input channel. Then connect a servo to the arduino - look up online where each wire should be connected to!
- Open up the Knob example under Examples -> Servo, and inspect the sketch:
 - To control the servo, the Servo library is imported using the command `#include <Servo.h>` at the top of the sketch. This allows us to use very simple functions for controlling the angle of the servo arm.

- The Servo myservo; command creates a servo object and names it myservo.
- In the setup function, the attach command is used to set which Pin is connected to the servo motor. Make sure the pin has PWM capabilities!
- In the loop function, the myservo.write(angle) command is used to control the position of the arm, where angle is a value from 0 to 179.
- Once you have a good understanding of the sketch, and are sure that you've properly connected the servo and potentiometer, upload the example code to your arduino and use the potentiometer to manually control the servo arm position.
- Using what you have learned from this example sketch, adapt your old sketch to control a servo based on the reading from a thermistor. Program the thermistor sensor reading to control the rotation speed of the servo arm (pretend it's a fan!). As the temperature reading increases, the speed of the servo arm should increase. You may want to look at the Sweep example under the Examples -> Servo folder. The temperature magnitude should also be displayed using your array of five LEDs.
- **Show the GSI your fan works before cleaning up!**

Visualization of Arduino in Matlab

- 1. Download folder *ArduinoIO*
- 2. In Arduino app, open ArduinoIO>pde>adives
- 3. Upload to Arduino
- 4. Take note of port that Arduino is attached to (Tools>Serial Port)
- 5. Open Matlab
- 6. Navigate to the ArduinoIO folder
- 7. Execute the 'a0_display' GUI by typing in a0_display into the commandline
- 8. Enter the serial port into the command line when prompted
- 9. Click 'acquire' to see a visualization of your A0 input