ECE 1001/1002 Introduction to Robotics
Lab #1: Blink

Objectives

Introduction to the Arduino hardware and Arduino IDE, make an LED blink on and off.

Requirements and Signoffs

Get acquainted with your Sparkfun Inventor's kit. Generally we will not follow the included guidebook *SparkFun Inventor's Kit Version 4.1.* The on-line version is more up-to-date, has fewer errors, and is generally recommended when you need it as a reference. https://learn.sparkfun.com/tutorials/sparkfun-inventors-kit-experiment-guide---v41/all read and follow the assembly instructions in these sections:

- Baseplate assembly
- How to Use a Breadboard (link)

For SIK 4.0 (previous version) users, use the link above for the most up-to-date resource, but if you insist on using the printed version of *SparkFun Inventor's Kit Version 4.0* booklet, do these things

- Check the inventory on page 10
- Read pages 2 through 5.
- Follow the instructions on page 3 to attach the breadboard and Arduino Redboard to the baseplate. Pay attention to how the breadboard connects rows and columns of wires (page 5); you will use similar breadboards in labs throughout your electrical and computer engineering academic experience and beyond.

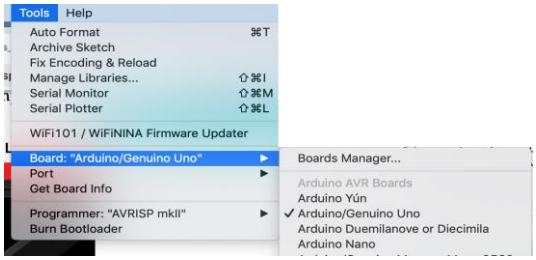
Arduino IDE

The computers in the lab have the software required: Arudino IDE (Integrated Development Environment, also known as IDE, development environment, or sometimes just environment).

You can install the Arduino IDE on your own computer, but do not use the Apple or Microsoft app stores to install—the versions on these stores are out of date and cause communications problems with your Arduino. To install the Arduino IDE, go to https://www.arduino.cc/ and click on software→downloads, then select the appropriate *installer* (not windows app) for your system. The Linux installers also work, and some students have made Chromebooks work; there is a web version of the IDE which works without installation, but can be slow. With the IDE on your own computer, you can get a head start on labs at home.

Use the USB cable, and with the Arduino IDE running, connect your Arduino board to the computer.

- The Arduino Redboard with your kit is the default Arduino board ("Arduino/Genuino Uno"), you might need to select it:



You may also need to select the port (tools→port) the selection choices vary with the computer you are using[1], with the correct one your board will blink some and you will be able to upload programs ("sketches"). Sometimes it may be necessary to select a different port, then go back and select the correct port (it's a Windows thing). When uploads don't work, this is always something to check. If you have questions, ask the instructor or TA. Disconnect the Arduino after you verify the port.

Jumper wires and Arduino headers

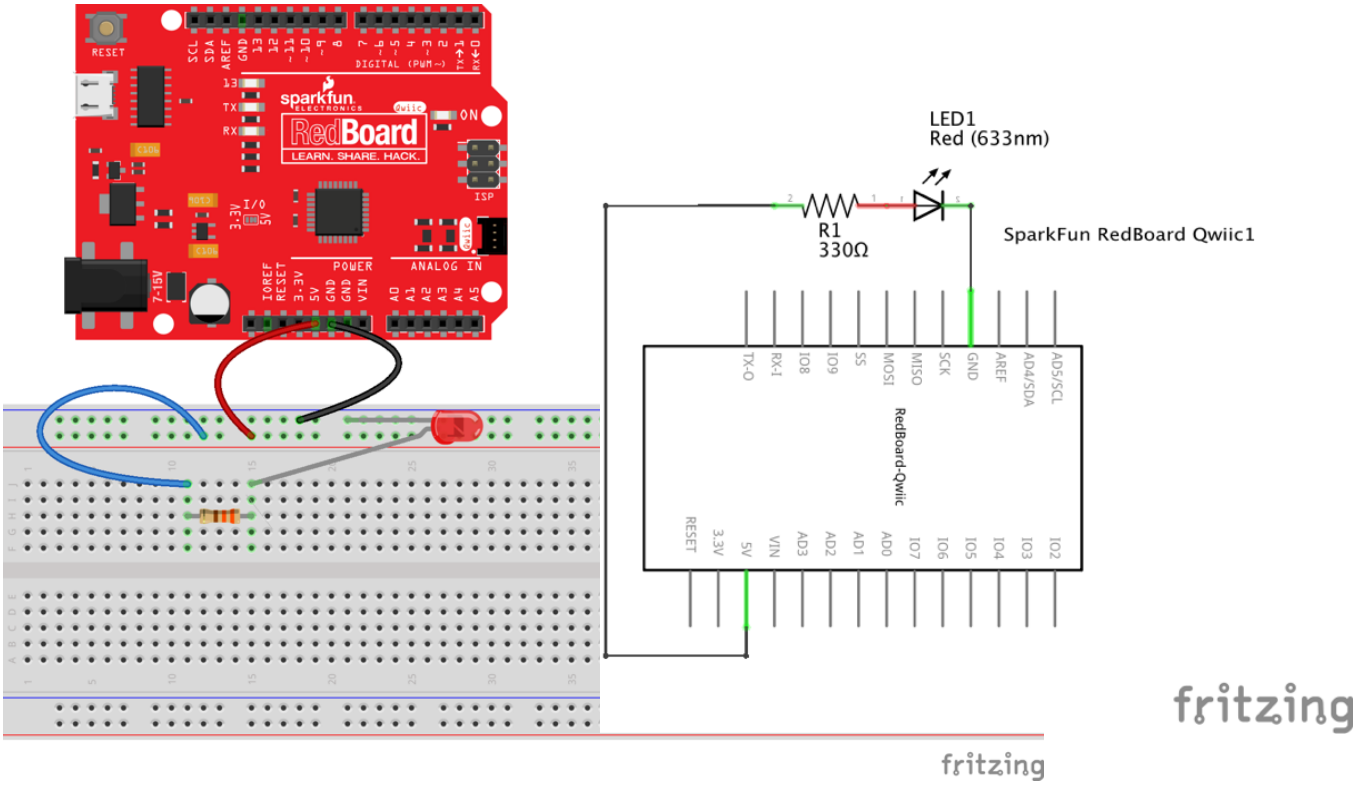
Look at your Arduino board, it has two rows of header pins, labeled 0 through 13 on one side (with a few others you can ignore), and A0 through A5 on the other side, along with pins labeled VIN, GND and 5V which you cannot ignore. You connect wires from these pins to devices on your breadboard, and back from the breadboard to your Arduino. Your kit comes with a set of jumper wires which should be enough for all of the labs. Treat these jumpers well, they are important. Some rules

- Remove jumper wires by the black connecter not by pulling on the wire; it takes more time and effort to do this, but pulling on the wire will lead to intermittent connections inside the connector which you can't see, and which will drive you nuts (and waste *hours*!) in later labs when your robot doesn't work due to a bad wire. You've been warned.
- Use red wires only to connect to "power" that would be 5V or VIN on your Arduino, and any components connected to those pins through other connections on the breadboard.
- Use black wires only to connect to "ground" that would be marked GND on the Arduino. That goes for any connection on the breadboard connected to ground.
- Use the other colors to connect inputs, outputs, etc. A color scheme will be nice if you can stick with it, but you will often find you don't have enough jumper wires of a certain color, so you will at least need to be flexible.
- There is also wire available for you to use which fits the breadboards, you are welcome to cut and use wire, advantages include possibly a neater, easier to debug system, but it takes more effort at first.

[1] As of August 2019, Engr 230 computers have two USB3 ports on the front (with the blue bar) which may connect as "port 4" (left) or "port 6" (right) and two USB2 ports on the front (without the blue bar) which may connect as "port 5" (left) or "port 7" (right). Try to use the same port each time to reduce connection adjustments

Wire up your board

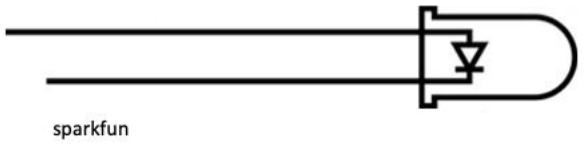
Use the breadboard wiring image and/or schematic below to guide you making connections to your board.



Connect a red jumper wired from the 5V pin on the Arduino to the long red (+) column on your breadboard. You can call that a 5V bus. Use a black jumper wire from any of the GND pins on the Arduino to the long black (-) column on your breadboard. Call that a ground bus.

Then connect the short leg of an LED (any color) to the ground bus. Connect the long leg of the LED to one of the short rows on the breadboard. Then connect either leg of a 330Ω resistor to that same row. Connect the other leg of the resistor to a different empty row on the breadboard. Finally connect a colored jumper wire (any but red or black) to the same row you just connected the resistor to, and the other end of the colored jumper wire to the 5V bus line. You can bend the legs on the resistor and LED as needed.

Why is one of the LED wires (legs) shorter? The shorter leg always goes to the lower voltage, which will always be Ground (GND) in this class. If you put it in the wrong way, the light won't turn on. LEDs are "polarized," there is a "+" (the long leg) and a "-" (the short leg) side which must always be connected correctly. The schematic symbol for a LED represents this polarization by showing a triangle for the LED, the triangle "points" in the direction of the lower voltage.



Many but not all components are polarized; some polarized components are damaged (even blow up) immediately if you connect them incorrectly; others, such as LEDs are not damaged if you connect them backwards, they just won't work until they are plugged in properly. Always double-check your work.

LEDs need a resistor to limit the current running through them, or they may burn out instantly and never work again. We'll work through some of those details in later labs, and you will learn much more about that in later electrical and computer engineering courses. For now, just remember that you must always connect a 330 Ohm resistor & LED combination. (usually we put the resistor closer to the high voltage (5V) but that's not always required, just a good practice). There are two types of resistors in your kit, 330 Ohm (330Ω) and 10,000 Ohm (10kΩ). They are identified by colored bands on the resistor (see the last page of the SparkFun book). Be sure to connect the 330Ω to your LED, (if you grab at 10kΩ instead, the LED will be very dim and hard to see that it is on). Resistors are not polarized, so you can connect either leg to a high or low voltage.

Sparkfun

Manual LED operation

Now use the USB cable to plug the Arduino into the computer USB receptacle. The LED should be lit. If it's not, trouble-shoot by double checking your wiring, and then call the instructor or TA.

Once you have the LED lit, move the colored jumper wire from the 5V bus to the ground bus. The LED should turn off. Try moving that jumper back and forth, turning the LED on and off.

When you move the wire back and forth, you turn the light on by connecting the LED between power (5V) and ground, and off by not connecting the LED between power (5V) and ground. It's the same thing a light switch on the wall does.

We could even connect the manual switch in your kit to turn the light on and off, but we have something much better to do: use software to turn the light on and off.

First get used to this language:

5V can (but not always) mean the same thing as:

- + (plus, but *not* in the addition sense)
- Digital HIGH (or just "high")
- TRUE
- Yes
- On
- 1 (*only* in the binary 1 sense though)
- Power

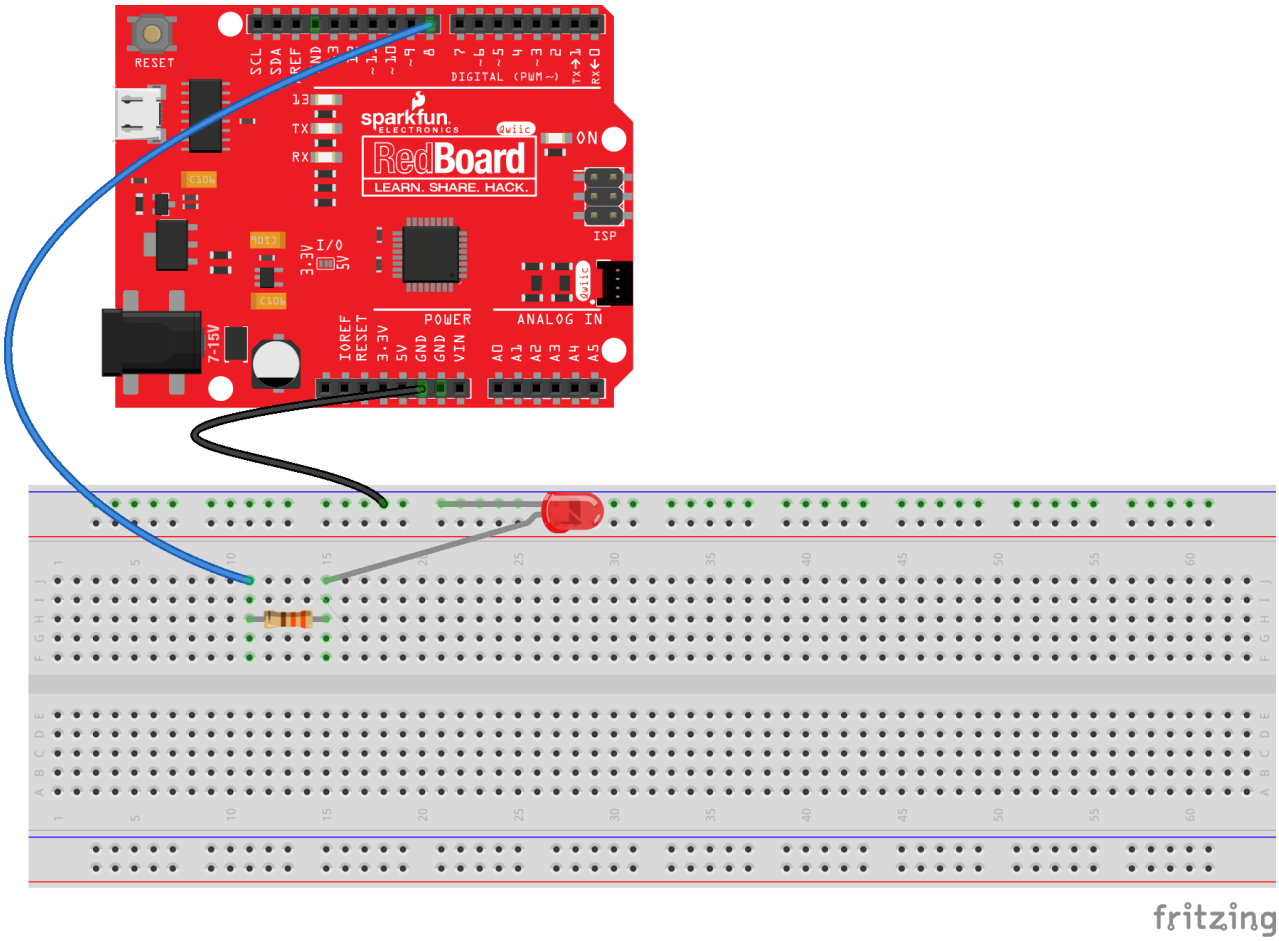
Similarly, Ground can (but not always) mean the same thing as

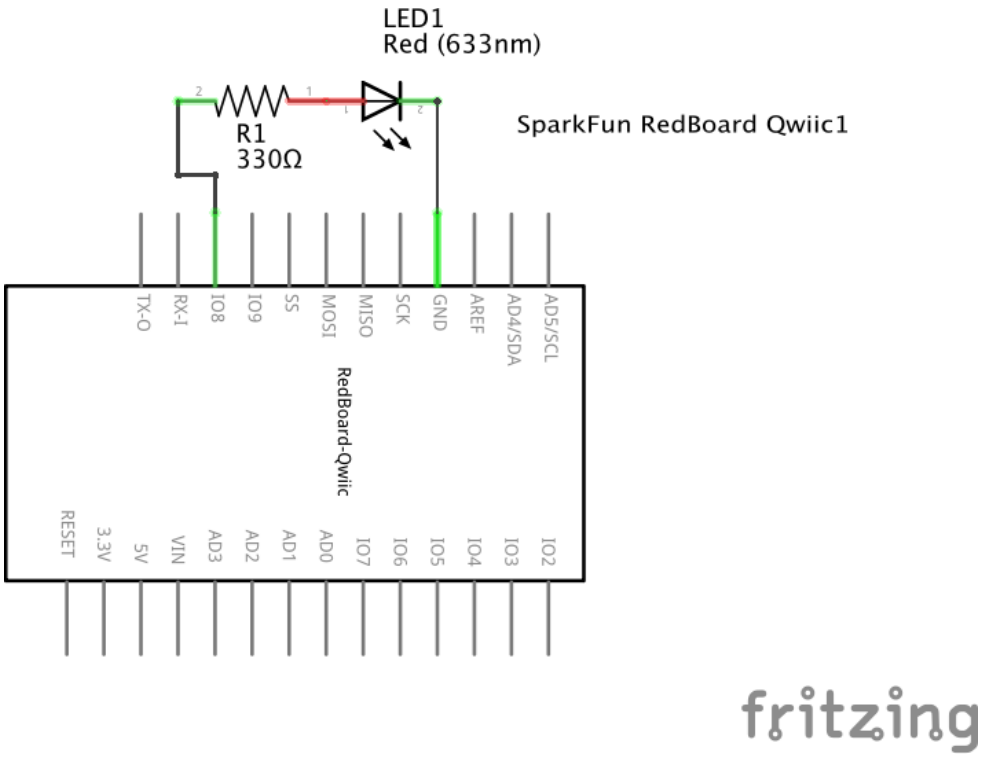
- - (minus, but *not* in the subtraction sense)
- Digital LOW (or just “low”)
- FALSE
- No
- Off
- 0 (*only* in the binary 0 sense though)
- Ground

These concepts may much sense yet, but you will get used to it. There is plenty of nuance to these things too; in particular, a “HIGH” voltage does not always mean 5V, but a HIGH voltage level is always higher in magnitude than the LOW voltage level.

Software LED Operation

Re-wire the breadboard by moving the blue jumper wire from the 5V (or ground) bus to pin 8 on the Arduino. You can leave the red wire from 5V or remove it. You should have a system which looks like the image below





Now open the Arduino IDE, and write the code provided in this Lab to blink the LED. Save the program to someplace on your network drive (or your own local computer if using a notebook). Don't save your program to a local drive on a lab PC, as that may not be available next time you log in. (A best practice will be to create a shared Office365 folder for programs, documents etc. in later labs, but it shouldn't matter much for the first 9 labs).

Get acquainted with the IDE when you can. Generally, don't maximize the IDE, but don't keep it at the tiny default size it opens up at either. You may need to increase font size if you have a high-resolution setting on your screen (edit→increase font size).

The check icon: ✓ compiles your program. This is a necessary step before you can send a program to the Arduino and run it. You can press the compile button as much as you want whenever you want, to run your program it must not report any errors at the bottom of the IDE window, it will say "done compiling" and report of how much memory the program uses, such as:

> *Sketch uses 950 bytes (2%) of program storage space. Maximum is 32256 bytes.*
> *Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.*

Anything else means there is an error of some kind which must be resolved before you can run your program.

The icon just to the right of the compile icon is the upload icon, → After you compile, you can hit that button to transfer your program to your Arduino where it will start running right away. Your Arduino must be connected to the computer by the USB cable, and the correct serial port must be selected. It takes 5—10 seconds for a program to transfer to the Arduino, during that time the TX and RX (transmit and receive) LEDs on your board will flash on and off. You should learn to always check for those flashing TX and RX LEDs during uploads.

Compile and upload your program to your Arduino. If you have it wired correctly, it will start flashing on for 1.5 seconds and off for 1 second. Over and over and over…forever. The only way to stop the program, by the way, is to remove power from the Arduino by unplugging it. That's one way a micro-controller is different from a micro-computer or any other computer you are accustomed to.

Understand and Modify the Program

Look at the program, in the IDE. It has several sections. At the top are comments, these are simple things to add (and if you don't add comments in later labs you will lose points) which give basic information about the program. Comments are not compiled, and are not actually sent to the Arduino, they just exist for humans to understand the program. When they don't exist, you will find they are a reason for instructors to deduct points.
The comments at the top are:

```
//
//         file_name.ino
//
//  By: Les Tekamp
//  Date: 25 Jan 2022
//  Class: ECE-1001-002
//  Assignment: Lab-1
//
//  Problem Statement: Blink the LED on pin 13 On for 750 msec and
//      Off for 250 msec.
//
//
//              %%%%    Algorithm    %%%%
//
//  define global variables
//
//  setup function
//    set pin 13 as output
//    set LED Off
//  ends setup
//
//  loop function
//    set LED On
//    delay 750 msec
//    set LED Off
//    delay 250 msec
//  end loop
//
```

Comments are anything after **//** Further down you can see comments which say:

```
// initialize pin 8 as digital output to control the LED
```

Note, the IDE helps you recognize a comment by putting it in a light grey font.

Next you will find the initialization (setup) section of the program. The Arduino will do the commands listed in this area just once when the program first starts. The initialization section is:

```
void setup()
{
  pinMode(8, OUTPUT);   // initialize pin 8 as digital output to control the LED
}
```

"void" looks odd, don't worry about it for now[2], it needs to be there. "setup" is a keyword marking this as the initialization area of the program (you can't use the word "setup" anywhere else in the program except in a comment).

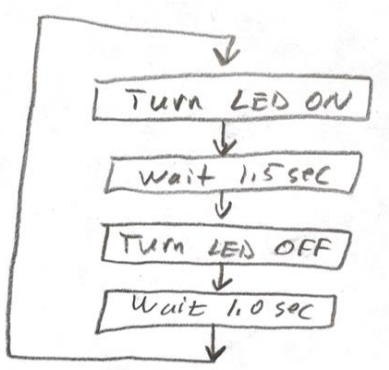
The curly braces **{** and **}** enclose the part of the setup section where the important stuff happens. Inside the curly braces there is this:

```
pinMode(8, OUTPUT);
```

pinMode and OUTPUT are also keywords, you can use them elsewhere, but they have specific meanings. pinMode is used to tell how a pin is used[3]. OUTPUT means the Arduino will send voltages out on pin 8 as controlled by the program. Note pinMode must be capitalized that way, and the brown color means the IDE recognized the keyword. Similarly, OUTPUT must be capitalized, and the IDE recognized it as a pinMode by making it green. Finally, note there is a semicolon **;** at the end of the line, this tells the IDE where the end of the command is. It's very important that you have the **;** there, the IDE will be totally confused without it, and will refuse to compile. You may spend the rest of the semester hunting for and inserting forgotten (or extra!) semicolons.

Next, is the loop() which just repeats everything between the curly braces **{** … **}** over and over forever. At first this is where all of the action takes place in your programs. Later some of the action which is repetitive may occur in a different area of the program, but the important logic is should be in loop (). The flow chart at the right represents what the loop () does in this program.

```
void loop()
{
 // use delays and pin 8 to blink an LED on and off
 digitalWrite(8, HIGH);
 delay(1500);
 digitalWrite(8, LOW);
 delay(1000);
}
```



The digitalWrite(8, HIGH); command sends 5V (the meaning of HIGH) over pin 8. This command turns on the LED, which is connected to pin 8, and it will stay on until turned off.

Next the delay(1500); command has the Arduino do nothing but count time for 1500 ms (1.500 seconds). After the Arduino finishes counting 1500 ms, it moves on to the next line in the sequence which turns the LED off: digitalWrite(8, LOW); and waits for 1000ms (1.000 seconds).

The program then repeats the cycle forever.

---

[2] "void" means this function doesn't return anything to another function which might call it

[3] Pins for this class will mean a pin can be a digital input, digital input with internal pullup resistor, digital output, analog input, PWM output. You can change a pin's mode anywhere in the program, not just in setup, but that will cause you headache at this stage. There are other more advanced uses for pins as well.

Change the 1500 and 1000 wait values to something else, compile and upload the revised program. See what happens! Try short times and long times, you will probably get bored with delays of more than 2 or 3 seconds. What is the shortest blink you can see?

For Signoff

The signoff task requires you to modify the code provided in the above Lab write-up. Connect the LED to pin-13. Change the program to flash pin 13 on ½ second and off for ½ second. Once you have it working right, show your breadboard with the colored LED you wired up and demonstrate pin 13 blinking on and off for ½ second.

Your instructor will either have you print off a sign-off sheet to sign or have a roster to check your name off. To receive credit for this lab, you *must* get the sign-off before the due date (see Canvas), during class time or with a TA during their hours. The instructor will not signoff outside of class hours.