CSIS 10C **Lab 1B Arduino - Working with LEDs and Resistors**

Objectives

- Introduce you to using the Arduino Uno with the AVR C Tool Chain.
- Breadboard a circuit using LEDs and resistors in preparation for class.
- Learn how to change colors of an RGB LED under program control.
- Understand the relationship between the hardware circuits and the code.

Introduction:

The Arduino IDE and Arduino software framework are very helpful in helping students learn about microcontrollers such as the Arduino Uno. This approach comes at a cost of abstracting or removing some of the knowledge required to be successful developing software. In this series of labs, we will continue to use the Arduino Uno, however, we will use the AVR C language tool chain in place of the Arduino IDE. We'll also use the Standard C Library (avr-libc) and a Standard C to Arduino Library to replace the Arduino software framework. This will provide an opportunity to learn and use Standard or ANSI C on a microcontroller. This approach will be very similar to the approach many software engineers use to program embedded microcontrollers. It will also offer an opportunity to better understand C in a less complex processor than your personal computer.

To be successful with this approach, you will need to use the command line appropriate for your platform (Windows, macOS or Linux). In Windows, it can be Windows Subsystem for Linux (WSL1) or Windows command line, **however in both instances, please download and use Microsoft Terminal**. In macOS, the application is called Terminal and in Linux, it is typically referred to as the Console. While all the commands will be executed in the command line, much of the work is repetitive so it becomes second nature quickly. You won't need to know a great deal as to how to work in the command line, however, you will need to know:

- how to move from directory to directory
- how to create a directory
- how to list the contents of a directory
- how to delete files

You will also need to use a couple of specialized command line applications such as *gcc*, *averdude* and *make*, however, you will be provided the exact forms of those commands to use. A good tutorial to review is A Linux Command Line Primer (for WSL, macOS and Linux).

The other application which you will use in place of the Arduino IDE, is a code editor. There are many to pick from, however, here are the recommendations:

- Windows – Notepad++
- macOS – Sublime Text 4
- WSL or Linux – Sublime Text 4 or Kate

Please do **not** use another IDE, such as Microsoft Visual Studio Code. This type of coding environment is quite powerful and very complex and will do little to help you learn how to code in C.

Ideally, if you are reading this on your computer, you have already completed the exercises outlined in the Introductory Lecture (*lec1bintro.ppt*). The exercises are defined as: (Next Steps)

1. Install Arduino IDE and test your system with your Uno (and get your serial port number!)
2. Install git, make, and the AVR tool chain per: https://www.wellys.com/posts/avr_c_gettingstarted/
3. For your OS, work the exercises: using gcc and avrdude to upload blink to Uno.
4. Use make for automation and use *make flash* to perform the same steps as #1 and #3.
5. Use git to clone the Lab repository on your system: https://github.com/lkoepsel/Labs_10C_Class

When you finish with this set of instructions, you will be in a directory, Labs_10C_Class. In this directory, will be four directories, ***Library***, ***examples***, ***templates*** and ***Labs***.

1. ***Library*** is the C language set of functions which mirror Arduino functions such as analogRead(), analogWrite, tone(), and pinMode().
2. ***examples*** is a set of directories which contain example code for the Arduino functions in the ***Library***. If you are unsure as to how to use a function from the ***Library***, review the example pertaining to that function.
3. ***Labs*** directory contains documentation for each of the labs
4. ***templates*** contains the coding templates (or sketches in Arduino terminology) for each of the lab steps.

Test Setup (Please follow explicitly.)

A key to successful software development is to be organized with your development efforts. This is particularly important when using the command line applications as a), it is harder to see your organization (as compared to a graphical user interface) and b), applications like *make* will have certain assumptions as to where files are located and how they are named.

Each of the individual lab *templates* contain multiple directories, each corresponding to an assignment in the lab. Each directory contains a *main.c* template as well as a *Makefile*. You will use the *main.c* template like the way you would use the sketch template in the Arduino Uno.

Please do the following exactly:

1. **Please copy the contents of the *templates* folder into a folder called *dev.* The *dev* folder will be your development folder, while the *templates* folder will remain untouched.** This is done for two reasons; 1) it allows us to easily update the contents of the class without overwriting your work and 2) it allows you to copy a file from *templates* if something goes wrong in your development work.

2. In your testing from the introduction, you identified the Serial port which works for you on your system.

    1. At the base of the *10c_labs_class* folder, there is a file called *env.make.* This file contains variables specific to your Uno and your computer.
    2. Open the file in your code editor and go to the section *"# Arduino UNO and compatible boards*" and find the *SERIAL* variable and will need to be assigned, specific to your system. It is currently similar to "*SERIAL = /dev/cu.usbmodem3101*".
    3. The easiest solution is to use the Arduino IDE to identify the correct serial port *Tools =>*

   Port => [copy the name with (Arduino Uno) after it] and enter the name after the = without quotation marks.

4. It will look something like this:
   - **Linux: *SERIAL = /dev/ttyACM0***
   - **macOS: *SERIAL = /dev/cu.usbmodem3101***
   - **WSL: *SERIAL = /dev/ttyS6***
   - **Windows: *SERIAL = COM3***

3. We need to do a check of our setup to ensure we are good to go. And to do that, the easiest method is to check it with the blink program. Go to your terminal program and do the following:
   1. Switch to the *dev/1B/1_blink directory* (`cd dev/1B/1_blink`)
   2. Make sure your Uno is plugged into your computer and line 5 of the Makefile, the *SERIAL* variable contains the correct name of the Uno serial connection.
   3. Run `make flash`

If the Uno is flashing, success! If not, check your connections and look for errors in the command line. **All errors in executing the *make flash* need to be corrected before going forward.**

Part 1: Blink Circuit

In this lab we explore how to breadboard LED's (Light-emitting Diodes) and resistors and make an LED blink under program control.

1. Follow the lab, LEDs, on the Adafruit website. If possible, use 220, 470 and 1 kΩ resistors. If your kit does not have the exact resistor values, you may use something that's reasonably close. Going from 270 to 220 Ohms will not make a noticeable difference.

   Resistor Color Chart

   Resistor Codes for this lab:

   - R-R-Brn = 220 Ohms
   - Y-V-Brn = 470 Ohms
   - Brn-Blk-R = 1000 Ohms
   - Brn-Blk-O = 10,000 Ohms

2. Use the blink code in *1_blink* from the above test instead of the code in the Adafruit example. Make the change recommended from pin 13 to pin 7. Make sure it works the same.

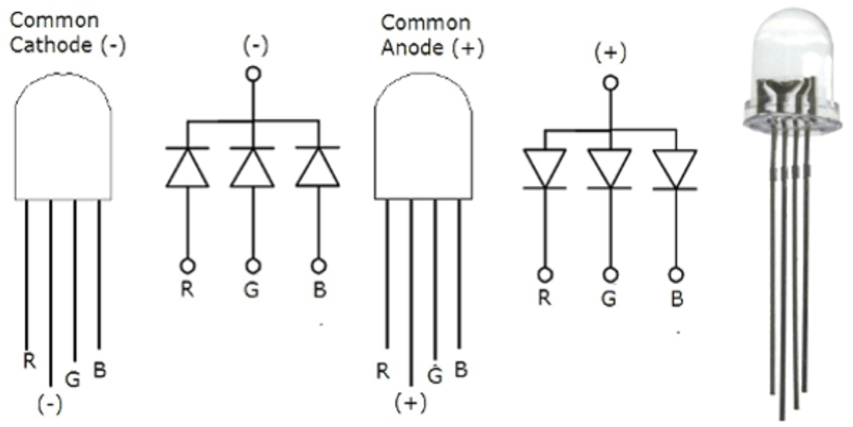
3. Challenges:
   - Use the next directory, *2_rr* and make changes to the code for an additional LED and make them blink like railroad crossing lights.
   - Use the directory, *3_sos* to write code to send an SOS signal - 3 short, 3 long, and 3 short blinks, followed by a long delay

Part 2: Change RGB LED Colors

In this lab we control the colors displayed by an RGB LED. We will use the directories *4_rgb* and *5_rgb_nested*. As we did in the earlier example, use the *main.c* template instead of the Arduino sketch templates. Make sure your *Makefile* continues to reference the correct serial port.

1. Complete the lab, RGB LEDs, on the Adafruit website using the directory *4_rgb*.
2. Using a 1000 Ohm LED can be helpful to reduce the brightness and make it more comfortable for your eyes.

3. Note: the RGB you are using could be Common Cathode or Common Anode (MPC kits use Common Cathode RGB LEDs)



4. The two types of LED can be handled appropriately in software. Look for a line in the code that defines a constant: #define COMMON_ANODE
   and activate this line if that is the case for your LED.

5. Challenge: (Use directory *5_rgb_nested* for this challenge.)
   - Modify your code for the RGB circuit to use three nested for loops to cycle through all the possible color code variations possible (255x255x255). You may want to skip a few, by stepping up to 255 in jumps of 16 or so. You may also want to add a short delay at each step.

Part 3: Turn in your work

- When you are finished, copy, and paste the code for both problems into the Online Text window on Canvas.
- It doesn't matter how far you get in the lab ... you will receive full credit regardless. The lab is over when you decide to stop working on it :)

Part 4: Summary of Commands/Actions Required for Labs

1. Make sure the correct serial port is referenced on line 5 of each *Makefile*.
2. To upload code to the Arduino Uno as you did with the Arduino Uno IDE, enter *make flash* on the command line. (Make sure the Uno is connected to your computer!)

3. Use the *main.c* template in the same way you used the Arduino IDE sketch template. They serve the same purpose.
4. Each lab exercise will be in a separate folder (just as each Arduino sketch is in its own folder). Sometimes it will be helpful to copy the *main.c* file over from one exercise to the next to reduce the amount of writing/editing required.