Microcontrollers Systems and Interfacing

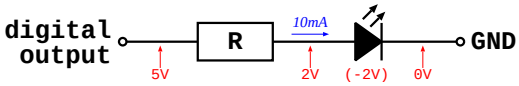
week 4 experimental lab

1 Connecting external LEDs

The light-emitting *diode* (LED) connected to digital pin 13 is useful for debugging, but we can make more versatile LED displays. Let's connect a couple of LEDs to digital outputs and investigate some interesting behaviour.

1.1 Circuit for each LED

Each LED will be connected to a digital output pin through a current-limiting resistor. (See notes, Section B.) The Arduino supplies 5 V to the circuit when the output is `HIGH`. The *forward voltage drop* of the LED is 2 V, leaving 3 V across the resistor. We need to limit the current to 10 mA.



► Use Ohm's law to calculate the resistor value that allows 10 mA to flow when the voltage across it is 3 V.

$R_{\text{ideal}} =$ _____ Ω

Your kit contains the following resistor values: 150 Ω, 180 Ω, 330 Ω, 10 kΩ, 47 kΩ, 4.7 MΩ. None of them are the ideal value. What value do you have that is just larger than the ideal?

$R =$ _____ Ω

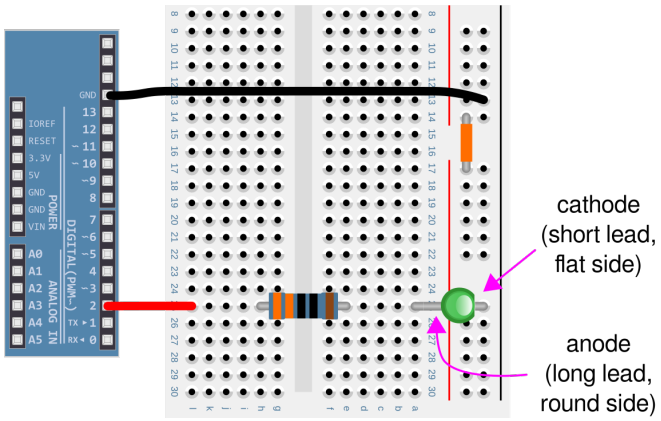
What is the colour code of that resistor? _____ _____ _____ _____

Hint: the first and second colours should be the same, and the third and fourth colours should be the same. If that is not the case, please re-check your calculations!

1.2 Breadboard layout for one external LED

Digital outputs 0 and 1 are used for serial communication. We will therefore connect our LEDs starting at digital output 2.

► Connect one LED as shown on the right.



The picture shows a green LED but you can use any colour you like. The long lead on the rounded side of the LED is the *anode*, connected to the positive side of the circuit. The short lead on the flat side of the LED is the *cathode*, connected to the negative (ground) side of the circuit.

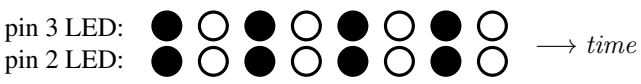
► Create a simple program to blink the LED once per second, just to verify that the circuit is working.

A few trivial modifications to your very first 'blink LED 13' program is all that you need.

1.3 Using multiple LEDs

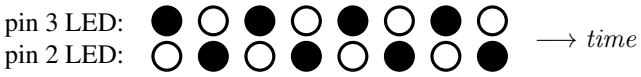
► Add a second LED to the system.

Make another LED flash in an identical way to the original LED. (The second LED should be connected to digital pin 3.) Completing this step just requires duplicating (and then slightly modifying) the existing code in `setup()` and `loop()`. The LEDs should make this pattern:



▶ Make the LEDs blink at *opposite* times.

Modify the program so that when one of the LEDs is on, the other LED is off. The LEDs should make this pattern:



▶ Control the blinking speed using an external analogue voltage.

Connect analogue input A0 to a variable voltage source. When the blue rotary control is turned fully anti-clockwise, the input will be at 0 V and the value returned by `analogRead()` will be 0. When the control is turned fully clockwise, the input will be at 5 V and the value returned by `analogRead()` will be 1023.

Make your LEDs blink once every 1023 ms. Then arrange for the value read from analogue input A0 to change the ratio between the time the LEDs are on or off, in the range 0 ms to 1023 ms.

Hint: the easiest way to do this is to use the value read from A0 as the time for the first part of the cycle (let's call this '`ta`'). Since `ta` is in the range 0–1023, we can set the second part of the cycle to '`1023-ta`' to obtain a total cycle time of 1023 ms. Then the potentiometer will move the ratio between the first and second parts of the cycle smoothly from 0:100% through 50:50% to 100:0%.

```
void loop() {
  int ta = analogRead(A0);
  int tb = 1023 - ta;

  digitalWrite(2, HIGH);
  digitalWrite(3, LOW );
  delay(ta);

  digitalWrite(2, LOW );
  digitalWrite(3, HIGH);
  delay(tb);
}
```

▶ Experiment with different delays.

What happens if you divide `ta` and `tb` both by 2? Or 4? Or 8, or 16, or 32? Keep doubling the divisor until something interesting starts to happen.

```
int ta = analogRead(A0);
int tb = 1023 - ta;

ta = ta / 2;
tb = tb / 2;

// ...
```

1.4 Using shorter cycle times.

What happens if you increase the divisor to 256? Or 378? When your divisor becomes large, the response of the LEDs should make it clear that there are only a few discrete values of delay, with abrupt steps between them. To make the delays change smoothly we have to use more accurate delays, with `delayMicroseconds()`.

▶ Increase the resolution of the delay, even when short.

Changing `delay()` into `delayMicroseconds()` will give you 1000 times more resolution. To compensate, you should delete the two lines that divide `ta` and `tb` by a constant.

- What do you notice about the number of steps from one extreme of the voltage input to the other?
- What practical uses can you think of for the effect that you have just observed?

```
void loop() {
  int ta = analogRead(A0);
  int tb = 1023 - ta;

  digitalWrite(2, HIGH);
  digitalWrite(3, LOW );
  delayMicroseconds(ta);

  digitalWrite(2, LOW );
  digitalWrite(3, HIGH);
  delayMicroseonds(tb);
}
```

2 Challenges

2.1 Make the cycle time variable

Add another potentiometer to your circuit, attached to another analogue input. Use the second analogue input value to control the overall cycle time. Arrange for the blink frequency to vary between approximately 1 Hz and 256 Hz. (The duty cycle percentage should continue to be controlled by the first analogue input value.)

Find the lowest blink frequency that does not 'flicker'. Being careful not to move your potentiometers, modify the program to print the cycle time on the serial monitor and then upload it. Convert the printed cycle time to a frequency.

Minimum flicker-free frequency: _____ Hz

Compare the number you obtain with those obtained by other people.[1]

2.2 Investigate PWM with sound instead of light

Reconnect your loudspeaker from last week, in place of one of the LEDs. (Do not forget to use a current-limiting series resistor!) What happens to the sound when you change the duty cycles of a square wave smoothly from 0% to 100%?

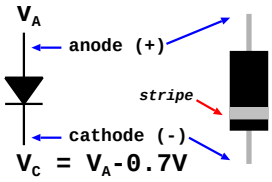
[1] The higher this number for you the more sensitive you are to low refresh rates in large TV screens and/or the PWM mechanism used to dim laptop displays, etc. I am very sensitive to flicker and I have a Lenovo laptop on which I cannot dim the display at all, otherwise I rapidly develop a headache from the flicker.

Microcontrollers Systems and Interfacing

week 4 reference material

A Diodes

A *diode* is a device that conducts electricity in only one direction. It has two terminals, the *anode* (positive terminal) and the *cathode* (negative terminal). For the diode to conduct electricity the *anode* must be *more positive* than the *cathode*; the diode is then said to be *forward biased*. If the *cathode* is more positive than the anode, then the diode is *reverse biased* and no current will flow.

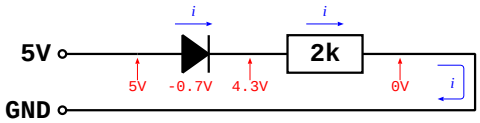


When a diode is forward biased (conducting) there is always a small *voltage drop* from the anode to the cathode. This is called the *forward voltage* of the diode and is written V_F. For a modern diode, V_F is typically 0.7 V.

A.1 Current through a diode

A forward-biased diode has a very low resistance that can be considered zero for almost all purposes. The current flowing in a circuit that includes a diode therefore depends almost entirely on the components connected in series with the diode.

When forward biased, the diode's voltage drop is always present and keeps the cathode voltage V_F below the anode.

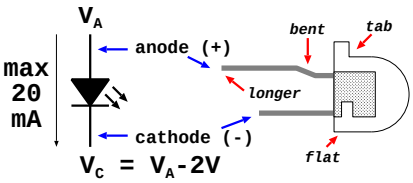


$$I = V/R$$
$$I = 4.3/2000$$
$$I = 2.15\,\text{mA}$$

In the example above, the supply voltage is known (5 V) and the voltage drop across the diode is known (0.7 V), so the voltage across the resistor must be 4.3 V. We now know both V and R, so using Ohm's law we can calculate the current through the resistor as $I = V/R = 4.3/2000 = 2.15\,\text{mA}$. In a series circuit the current through all components is the same, so the current through the diode must also be 2.15 mA.

B Light-emitting diodes

A *light-emitting diode* (LED) is a diode that emits light (usually of a single colour) when it conducts electricity. The voltage drop is higher than with a diode and depends on what colour the LED emits. A typical LED has a larger forward voltage than a normal diode. For LEDs, $V_F \approx 2\,\text{V}$, meaning the cathode voltage will be about 2 V lower than the anode voltage.

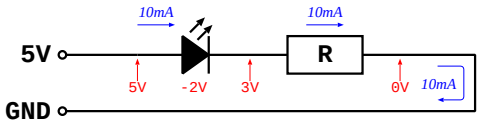


LEDs are *polarised* and must be connected the correct way round. The anode lead is usually identified by one or more of these methods:

- the anode lead is longer (the cathode is shorter),
- the anode lead is next to a small tab on the LED body (the cathode side is flat), and
- the anode lead is bent (the cathode side is straight).

B.1 Limiting the LED current

Too much current will damage an LED. Typical LEDs are designed to operate with no more than 20 mA. Like any diode, when a LED conducts it has effectively zero resistance and so connecting it to any voltage source larger than 2 V will cause a huge current to flow through it. To limit the current to a safe value of 10 mA or so, a resistor connected in series is required.



$$R = V/I$$
$$R = 3.0/0.01$$
$$= 300\,\Omega$$

In the example above, the supply voltage is known (5 V) and the voltage drop across the LED is known (2 V), so the voltage across the resistor must be 3 V. In a series circuit the current through all components is the same, so to limit the current through the LED to 10 mA we just need to limit the current through the resistor to 10 mA. We now know both the voltage V across the resistor and the desired current I through it, so using Ohm's law we can calculate the required resistance as $R = V/I = 3.0/0.01 = 300\,\Omega$.

C Pulse Width Modulation (PWM)

Frequency is inversely proportional to cycle time. With high frequencies (short cycle times) the human eye cannot see the individual on/off halves of the cycle; we see just the average 50% brightness of (for example) an LED when driven with a high-frequency (100 Hz or more) square wave.

The signal is on 50% of the time, which means it is transmitting power to the LED (or other device) onlt 50% of the time. If we make the `HIGH` and `LOW` times unequal, while keeping the total cycle time constant, we can adjust the ration of on time to off time, and therefore the amount of power transmitted to the device. With short on times, the power transmitted is lower than with long on times.

- The ratio $t_{\texttt{HIGH}}/t_{\text{cycle}}$ determines the amount of power in the signal:
  - low power $\Rightarrow$ low volume, low brightness, etc.
- Expressed as a percentage, this ratio is called the *duty cycle* of the signal:
  - 0% duty cycle transmits no power,
  - 50% duty cycle transmits half power,
  - 1000% duty cycle transmits full power.

```
int highTime = cycleTime * duty / 100;

void loop() {
  digitalWrite(pin, HIGH);
  delayMicroseconds(highTime);
  digitalWrite(pin, LOW);
  delayMicroseconds(cycleTime - highTime);
}
```

When one signal A controls another signal B we say that A *modulates* B. if A is desired change in power delivered by B then A controls the duty cycle of B; in other words, A modulates the width of the pulses in signal B. Hence the name *pulse width modulation*.