LEDs and Sensors: Analog to Digital

In the last lesson, we used switches to create input for the Arduino, and, via the micro-controller, the inputs controlled our LEDs when playing Simon. In this lesson, we will again use input to control output, but this time we will use *analog input* and *analog output*.

Analog Input

The real world is not digital. Consider temperature fluctuation as an example, it generally moves within some range of values without making large abrupt changes. We can measure aspects of our world like temperature, light intensity, forces, or whatever using *analog sensors*. In a digital device, the resulting signals are stored as sequential digital data.

Consider the analog signal depicted below.

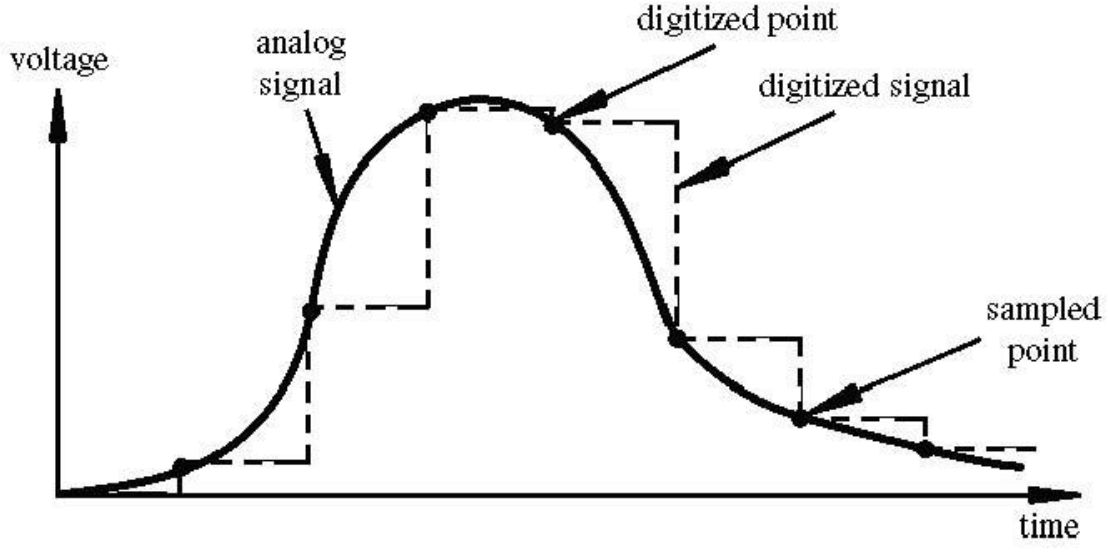


Image credit: Tod Kurt

In a digital device, we discretize the input signal range (i.e., 0 to 5 Volts for the Arduino) into different voltage levels or intervals called *states*. The number of states is the *resolution*. Common resolution values range from 256 states (i.e., stored in 8 bits) to 4,294,967,296 states (i.e., stored in 32 bits); the Arduino uses 1024 states stored in 10 bits. This means that 5V/1024 or 4.8 mV is the smallest voltage change that you can measure.

The translation of analog voltage values into different states is called *Analog to Digital Conversion*. The Arduino has built-in analog-to-digital conversion, and the `analogRead()` command (it's actually a procedure call), is used to invoke that conversion process.

Analog Output

As you know, the digital pins on the Arduino board can output either high (5V) or low (0V) — on or off. But what if we want an output voltage in between 0V and 5V--something simulating an analog signal? Well, we can do this using a process called *Pulse Width Modulation* (*PWM)*.

Although we can't use the Arduino's digital pins to *directly* supply say 2.5V, we can *pulse* the output on and off really fast to produce the same effect. If you were to watch the output signal on an oscilloscope, you would see the output pulsing between high and low at regular intervals. Since this on-off pulsing is happening so quickly, the connected output device "sees" the result as a 50% reduction in the normal voltage (in this example).

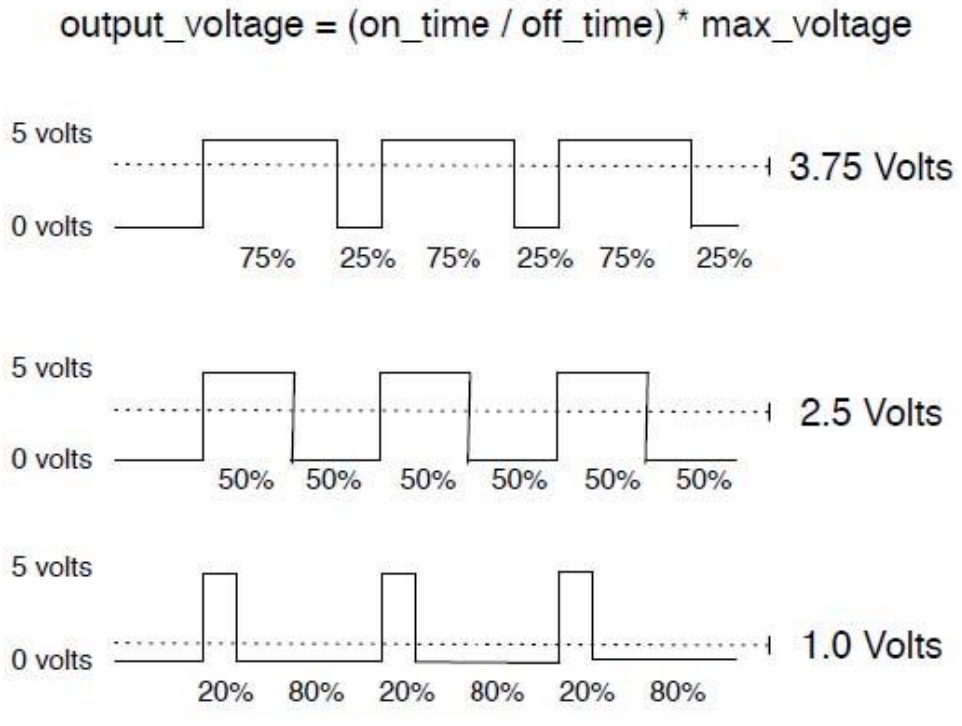


Image credit: Tod Kurt

We can vary the output voltage (the "effective voltage") by regulating or "modulating" the width of the high pulse. For example, if we make the high pulse 25% as "wide" (in time) as the low pulse, the effective voltage will 25% of full voltage.

On the Arduino, the `analogWrite()` command (i.e., procedure call) applied to an analog output pin is used to invoke the PWM process. In the next exercises, we will use a potentiometer and several sensors to generate analog input. We'll then use that input to control both digital and analog (i.e., PWM) output.

Analog I/O Exercise 1: LED Fader

In this exercise, we will use a *potentiometer* to generate analog input. We've used them before; a potentiometer, or, pot for short, is just a variable resistor.

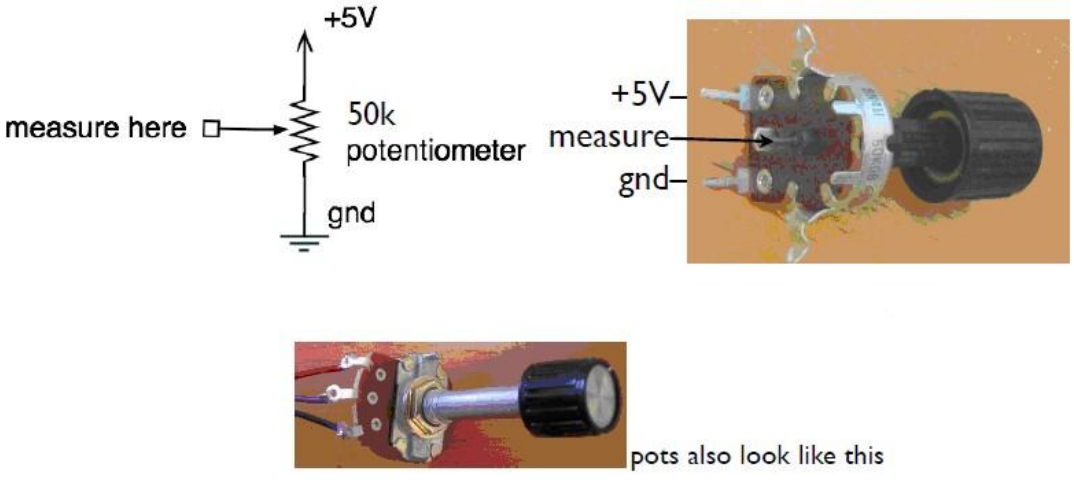


Image credit: Tod Kurt

Connect the middle leg of the pot (the one marked *measure* in the image above) to analog pin 2 of the Arduino. Connect the other 2 legs of the pot to 5V and ground as shown above.

Turning the knob of the pot changes the voltage experienced at pin 2 as shown below.

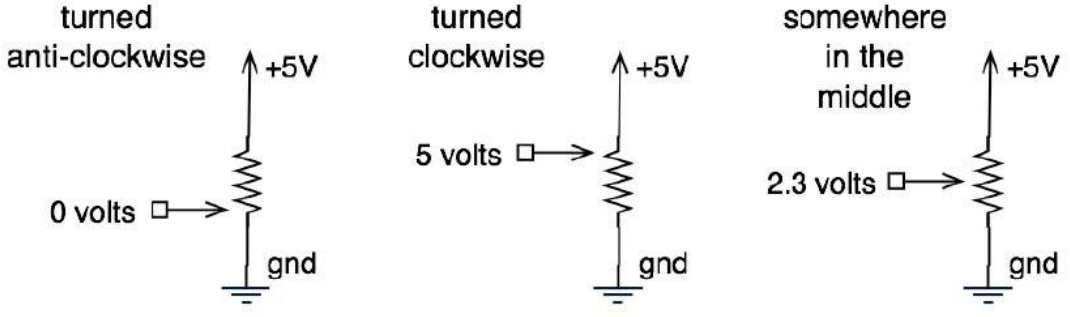
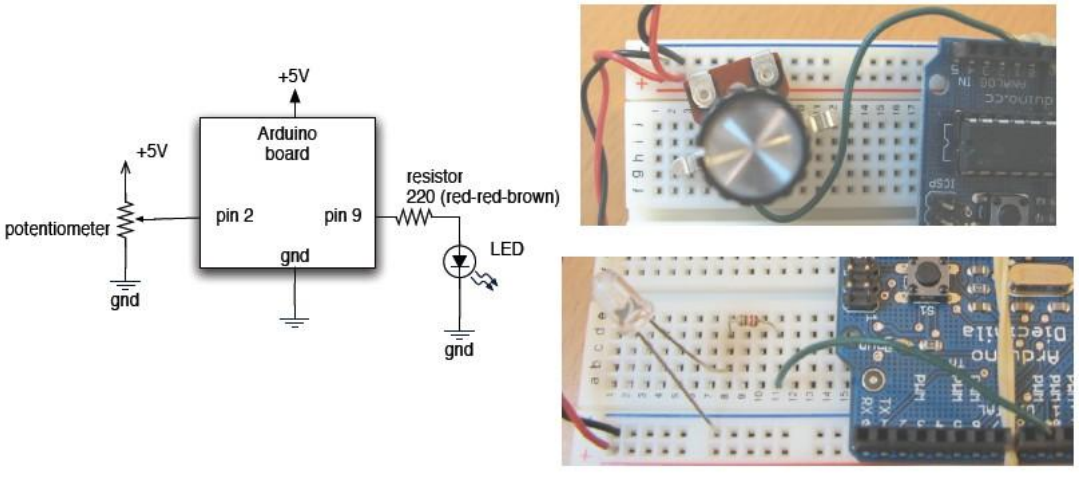


Image credit: Tod Kurt

Now add an LED to your breadboard and connect it to pin 9 of the Arduino as shown below. Note that pin 9 is an analog output, or PWM pin.



Now run the program PotDimmer by Tod Kurt to control the brightness of the LED using the pot. Observe how the LED light fades and intensifies depending on the position of the pot. Can you identify the line of code used to read the analog input? How about send the analog output? Why is the value read from input divided by 4 before it is used as output?

Analog I/O Exercise 2: Theremin

DO NOT BREAKDOWN THE CIRCUIT you created in exercise 1, but rather add a speaker to pin 7.

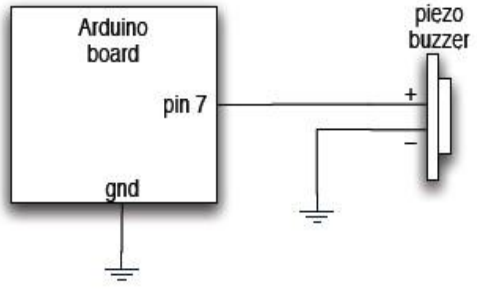


Image credit: Tod Kurt

Now run the Theremin program by Tod Kurt.

Once you understand the program, modify it so that the input from the potentiometer not only controls the output of the speaker but also drives the LED. Create a seasonally "spooky" sound and light device. Be prepared to demonstrate your device and explain the changes you made to the class.

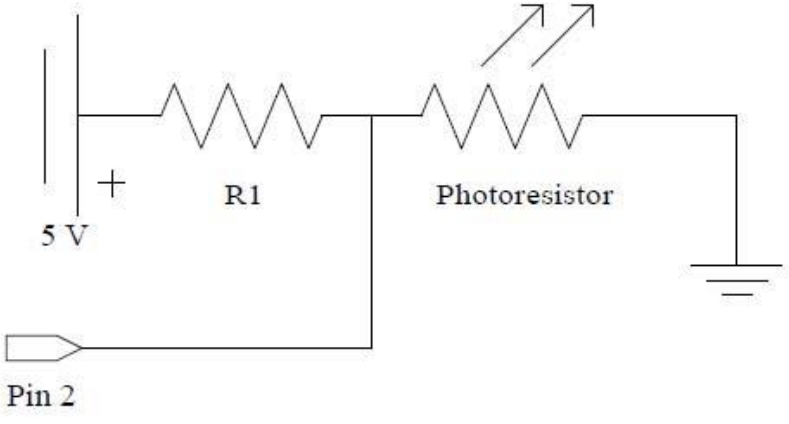
Analog I/O Exercise 3: A Distance-Sensitive Theremin

DO NOT BREAKDOWN THE CIRCUIT you created in exercise 2, instead replace the pot with the SHARP distance sensor. The SHARP distance sensor bounces Infra-Red (IR) light off objects to determine how far away they are. It returns an analog voltage that varies with the distance of the nearest object. These sensors are good for short-range detection, i.e., under 1m of distance.

The leads of the distance sensor will have to be trimmed and twirled before they can be inserted in your breadboard. Connect the black wire of the distance sensor to ground, the red wire to 5V, and the white wire to analog input pin 2. Run the program that you developed for exercise 2 again. Vary the distance of an object in front of the "eyes" of the distance sensor and enjoy the effects.

Analog I/O Exercise 4: A Light-Sensitive Theremin

This time replace the SHARP distance sensor with a photoresistor as shown in the schematic below. Use a large resistor such as a 10k Ohm (brown-black-orange) resistor for R1 in the schematic below.



A photoresistor is a light-dependent resistor---brighter light means less resistance. Again, run the program that you developed for exercise 2 and enjoy the effect that varying the light on the photoresistor has on the sound and the LED.