D2A <-> A2D

In the PWM experiment we saw how a digital signal could create the appearance of an analog result — specifically we sent a series of digital pulses to flicker an LED so rapidly that, to us slow humans, it appeared that it was just getting dimmer.

We did this by using the `analogWrite()` functionality of Arduino which did all the work to setup the PWM switching on the microcontroller for us. This worked well, and was simple enough to use, though there are other mechanisms available if you need more control over the PWM rates.

But how about if we need to go the other way? What if we have an analog value (say a varying voltage on a GPIO pin) from a sensor or other device that we want to read and maybe take some particular set of actions based on its value?

Well, the Arduino environment has you covered there as well, with, you guessed it, the `analogRead()` function. It works pretty much as you might suspect, returning an integer value depending on the voltage level present on the specified GPIO pin. However, there are a few differences and requirements.

The `analogRead()` function only works on the *analog compatible GPIO pins*, these pins are labeled as **A#** on the nano, and indicated as such on the pin description in the appendix. This is really no different than the PWM capability, as it too only works on certain GPIO pins, also as noted in the pin description.

A more significant difference is the value returned. While we could only specify a single byte value of 0-255 using `analogWrite()`, `analogRead()` returns values of 0-1023, which is 10 bits worth of information.

The process of converting a series of digital values into an analog counterpart is called Digital to Analog conversion, typically abbreviated as D2A. Likewise, the process of converting analog values into a digital representation is called Analog to Digital conversion, or A2D.

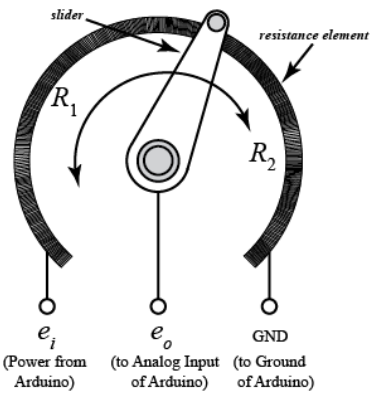
Depending on what the purpose of the data being converted is, the requirements for conversion may change, but the general concepts remain the same. Because such conversions are common place in many potential applications, most modern microcontrollers include some dedicated hardware for A2D conversions and at least some form of PWM control. Since such hardware has specific limits based on the size, speed and overall design of the chip, only select GPIO pins are able to perform these functions in hardware.

In this exercise we are going to look at how the A2D converter in our Nano operates. We already stated that it `analogRead()` will return values in the range of 0-1023. What those values represent, by default, is the voltage applied to the GPIO pin scaled from the minimum acceptable voltage, 0VDC, to the maximum voltage, which, for our Nano is 5VDC. So every step from 0 - 1023 is equal to 5/1024, or .00488 volts.

The AVR microcontroller series we are using has 10 bit A2D conversion hardware. (Hence the 1024 steps — 1111111111 in binary is 1023!) Other devices may have 8, 10, 12, 14, 16, or even 32bit hardware A2D converters. Also our Nano's A2D operates from 0-5VDC, other systems may operate across different voltages. 0-3.3VDC and 0-1VDC are common. Always know your hardware specs before wiring and programming!

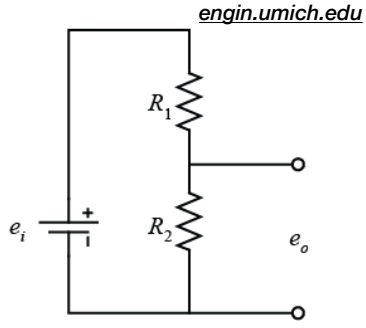
We already learned that adding a resistor in series to a circuit limits the current, but it also reduces the voltage in the process. If you were to connect two resistors in series and then measure the voltage at a point in between the two resistors (see graphic), you would observe a lower voltage than the source is producing. Depending on the values of the resistors in use, the voltage measured at that point will differ. This two-resistor circuit is called a voltage divider.

Voltage dividers are useful, but we want something that's dynamically changeable. So in this experiment we are going to use a potentiometer, or pot for short. A pot is basically a voltage divider with one resistor that has a value that can be changed, usually by turning a knob. The pot has 3 wires, one will go to a voltage source, another to ground, and the other will provide the changeable output voltage.



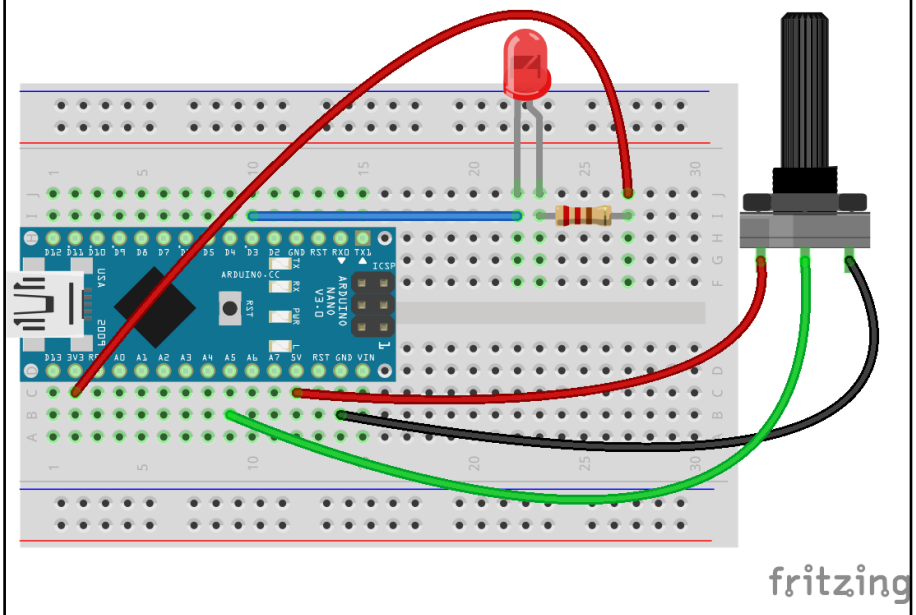
Potentiometer internals and connections.

engin.umich.edu



Voltage Divider:
e_o < e_i, based on the values of R_1 and R_2

Experiment: Your fingers on the pulse.

Components	Wiring Diagram
✓Computer ✓Arduino IDE ✓LED ✓Resistor ✓Potentiometer	

Connection Instructions

Use the same connection for the LED and resistor as in the PWM Experiment. Then connect a POT so that the two outer legs go to +5V and GND; it doesn't matter which one, but the convention is Red to +. Connect the center (GREEN) wire to A2

Sketch(es)	PPWM.ino

Analysis Questions

Programming Tasks

The sample program will output some values to the Serial Monitor. Run it and observe their change as you rotate the POT.

Now combine the code from the PWM experiment so that rotating the LED adjusts the brightness of the LED. Note that you will need to scale the values appropriately, as you get 1024 values from the POT and only have 256 (1/4 as many) that you can apply to the LED.

Notes

You can easily scale your values with simple math in this case, but the Arduino environment offers a couple of other options.

Find and Search the Arduino docs for the 'map' and see if you can use it in your program.

There is a class of potentiometers called trimpots, which are designed to mount to a circuit board and provide a small amount of infrequent tuning, or changes to a circuit. These are available in a variety of resistance ranges and physical sizes and shapes. Generally they are built to tolerate several 100 turns, while full sized potentiometers are made to handle 10s of 1000s.