Analog Signals

We have established that a digital I/O signal must either be LOW (0v) or HIGH (5v). Analog voltages can be anywhere in between (2v, 3.4v, 4.6v, etc.) and the Arduino has six special inputs that can read the value of such voltages. These six 10-bit Analog inputs (with digital to analog converters) can determine the exact value of an analog voltage.

Analog Inputs

The input is looking for a voltage level between 0-5vdc and will scale that voltage into a 10-bit value, or from 0-1023. This means that if you apply 0v to the input you will see an analog value of 0; apply 5v and you will see an analog value of 1023; and anything in-between will be proportional to the input. To read an analog pin, you must use the analogRead() command with the analog pin (0-5) that you would like to read. One interesting note about Analog inputs on the Arduino is that they do not have to be declared as variables or as inputs in the setup. By using the analogRead() command, the Arduino automatically knows that you are trying to read one of the A0-A5 pins instead of a digital pin. A potentiometer (variable resistor) acts as a voltage divider and can be useful for outputting a lowcurrent analog voltage that can be read by the Arduino using an analog input (see Figure).

Figure . *This typical turn-style potentiometer has three terminals. The outer two terminals should be connected to GND and +5v respectively (orientation does not matter), whereas the center terminal should connect to an analog Input pin on the Arduino.*

How to read an Analog input

```
// Code Example – Analog Input
// Read potentiometer from analog pin 0
// And display 10-bit value (0-1023) on the serial monitor
// After uploading, open serial monitor from Arduino IDE at 9600bps.
int pot_val; // use variable "pot_val" to store the value of the potentiometer
void setup(){
Serial.begin(9600); // start Arduino serial communication at 9600 bps
}
void loop(){
pot_value = analogRead(0); // use analogRead on analog pin 0
Serial.println(pot_val); // use the Serial.print() command to send the value to the monitor
}
// end code
```

Copy the previous code into the IDE and upload to your Arduino. This sketch enables the Serial port on the Arduino pins 0 and 1 using the Serial.begin() command–you will be able to open the Serial monitor from the IDE and view the converted analog values from the potentiometer as it is adjusted.

Analog Output (PWM)

This is not technically an analog output, but it is the digital equivalent to an analog voltage available at an output pin. This feature is called Pulse Width Modulation and is an efficient way of delivering a voltage level that is somewhere between the Source and GND. In

electronics, you hear the term PWM used quite frequently because it is an important and usable feature in a micro-controller. The term stands for Pulse Width Modulation and is the digital equivalent to an Analog voltage you find with a potentiometer. The Arduino has six of these outputs on digital pins 3, 5, 6, 9, 10, and 11. The Arduino can easily change the duty-cycle or output at any time in the sketch, by using the analogWrite() command. To use the analogWrite(PWM_pin, speed) command, you must write to a PWM pin (pins 3, 5, 6, 9, 10, 11). The PWM duty-cycle ranges from 0 to 255, so you do not want to write any value above or below that to the pin. I usually add a filter to make sure that no speed value above 255 or below 0 is written to a PWM pin, because this can cause erratic and unwanted behavior .

Circuitry

You are now going try a more advanced method of controlling LEDs. So far, you have simply turned the LED on or off. Would you like to adjust the brightness of an LED? Can you do that with an Arduino? Yes, you can.

The circuit for this example is the same as of our first example:

Code

Now enter the code:

Plug in the Arduino, click Verify, then Upload if no errors are found. The brightness of the breadboard LED should smoothly increase, then smoothly decrease and then repeat.

Understanding the Code

1. Most of the code should be easy to gure out once you have understood the previous examples.
2. The meat of the program is in the for loop.** The process inside this loop is carried for every angle ranging from 0° to 179°.
3. The angle is converted to radians using the in-built** radians(angle°) **function.
4. The sine value thus calculated ranges from 0 to 1. This is multiplied by 255 to get a value between 0 and 255. The Arduino's analog output is 8-bit and requires a value between 0 (LOW) and 255 (HIGH).
5. The obtained value is then typecasted to an integer in the same line using int()
6. analogWrite(ledPin, ledVal); writes ledVal to the specied pin i.e. ledPin

The concept here is that you are creating a sine wave (positive half-cycle of course) and having the brightness of the LED follow the path of that wave. This is what makes the light pulsate instead of just are to full brightness and fade back down again.

Pulse Width Modulation

Six pins (3,5,6,9,10,11) on the Arduino UNO board are marked as PWM pins. These pins differ from the remaining digital pins in that they are able to send out a PWM signal.

PWM stands for Pulse Width Modulation, which is a technique for getting analog results from digital means. On these pins, the Arduino sends out a square wave by switching the pin on and off very fast. The pattern of on/offs can simulate a varying voltage between 0 and 5v. It does this by changing the amount of time that the output remains high (on) versus off (low). The duration of the on time is known as the pulse width.

For example, if you were to send the value 0 out to Digital PWM Pin 11 using analogWrite(), the ON period would be zero, or it would have a 0 percent duty cycle. If you were to send a value of 64 (25 percent of the maximum of 255) the pin would be ON for 25 percent of the time and OFF for 75 percent of the time. The value of 191 would have a duty cycle of 75 percent; a value of 255 would have a duty cycle of 100 percent.

The pulses run at a speed of approximately 500Hz or 2 milliseconds each. So, in your sketch, the LED is being turned on and off very fast. If the Duty Cycle was 50 percent (a value of 127), the LED would pulse on and off at 500Hz and would display at half the maximum brightness. This is basically** an illusion that you can use to your advantage **by allowing the digital pins to output a simulated analog value to your LEDs.

Note that even though only six of the pins have the PWM function, you can easily write software to give a PWM output from all of the digital pins if you wish.

How to command a PWM output

```
// Code Example – Analog Input – PWM Output
// Read potentiometer from analog pin 0
// PWM output on pin 3 will be proportional to potentiometer input (check with voltage meter).
int pot_val; // use variable "pot_val" to store the value of the potentiometer
int pwm_pin = 3; // name pin Arduino PWM 3 = "pwm_pin"
void setup(){
pinMode(pwm_pin, OUTPUT);
}
void loop(){
pot_value = analogRead(0); // read potentiometer value on analog pin 0
pwm_value = pot_value / 4; // pot_value max = 1023 / 4 = 255
if (pwm_value > 255){ // filter to make sure pwm_value does not exceed 255
pwm_value = 255;
}
if (pwm_value < 0){ // filter to make sure pwm_value does not go below 0
pwm_value = 0;
}
analogWrite(pwm_pin, pwm_value); // write pwm_value to pwm_pin
}
    // end code
```

This code reads the potentiometer as in Listing 1-4, but now it also commands a proportional PWM output signal to Arduino digital pin 3. You can check the output of pin 3 with a voltage meter–it should read from 0v-5v depending on the position of the potentiometer.
If you have a 330ohm resistor and an LED laying around, you can connect the resistor in series with either LED lead (just make sure the LED polarity is correct) to Arduino pin 3 and GND to see the LED fade from 0% to 100% brightness using a digital PWM signal. We cannot use the LED on pin 13 for this example, because it does not have PWM capability.