# Module 005: Arduino Analog Inputs

## Introduction (If you have a hand-held Voltmeter, this can be done at home)

Parts needed

- Arduino or RedBoard, available at the ECE Supply Center (recommended for ECE110, required if you wish to complete this particular module)
- Hand-held multi-meter, available at the ECE Supply Center (for at-home completion)
- ECE110 Electronics kit (required for ECE110)
- 22-gauge, various colors and lengths, solid-core wires (provided in ECE110 lab)

One of the most powerful features of the Arduino board is its ability to take an *analog* voltage and convert that analog voltage into a number – a 10-bit binary number – but don't worry if you don't know binary, Ardunio will present this to you as a decimal number between 0 and 1023. The ability to sample voltages enables you to design projects that interface the real world of sound, heat, pressure, motion, acceleration, radiation, etc., to the digital world of the Arduino. The "sensor" devices that convert the properties of the real world to a voltage are part of a general class of devices called transducers. A *transducer* transforms one kind of energy into another. You are very familiar with these devices – even your own body is full of them. All of the interfaces to your computer is a transducer of one form or another – the microphone input converts the coherent motion of the air (sound) hitting the microphone into a voltage that is input to the computer – the speaker output converts the voltage generated by the computer to sound waves – the computer's camera inputs electromagnetic radiation in the visible light range from the surrounding environment and converts the resulting image to a voltage that the computer can then process and store.

The magic hardware that allows the analog and digital world to interface are complex devices – an A/D Converter (Analog-to-Digital Converter) and a DAC (Digital-to-Analog Converter). The Arduino only has an Analog-to-Digital Converter so you can only *input* analog inputs (a signal with a vast number of voltage options). You cannot output an arbitrary voltage level. Some other platforms like the TI development boards also include a D/A converter so that they *can* output analog signals.

This module will help you set up a simple experiment that illustrates the hardware and software considerations so that you can become familiar with the simple statements needed to get a digitized version of the voltage from any analog device/sensor that works within the electrical constraints of the Arduino.

# Procedures

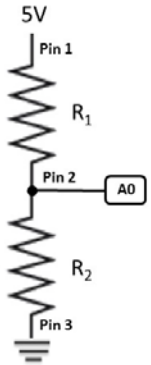
## Building a circuit to test

To fully understand the process of reading an analog voltage and use the result to control external circuitry let's build two circuits. An input circuit and an output circuit. The input circuit consists of a simple device that can provide a variable voltage that can be fed into one of the analog input pins on the Arduino/RedBoard. The simplest way to create a controllable voltage would be to build a voltage divider using a potentiometer. We can have the Arduino then control an LED whose brightness depends on the voltage at the analog input pin.

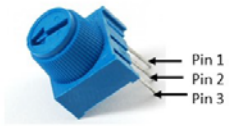
### Input Circuit

Using a potentiometer is a simple way to build a voltage divider circuit. The figure below shows the schematic associated with the potentiometer and the physical part. You have several potentiometers in your kit. The total resistance between the 5-volt voltage source "$5V$" and the ground (GND) is a fixed value $R$, but it is divided into two separate resistances $R_1$ and $R_2$ inside the potentiometer. Actually, there are not two resistors inside the device, but a single resistor with a movable *tap* in the middle that alters the balance of the resistance above and below the tap. The tap is the center pin of the potentiometer. If the potentiometer knob is turned the values of $R_1$ and $R_2$ change but the total resistance $R = R_1 + R_2$ remains the same. By turning the knob you can vary the voltage at Pin 2 according to $V_{pin2} = \frac{R_2}{R_1+R_2} \times 5$.

Schematic

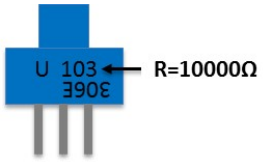


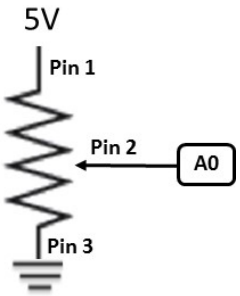

Physical Package

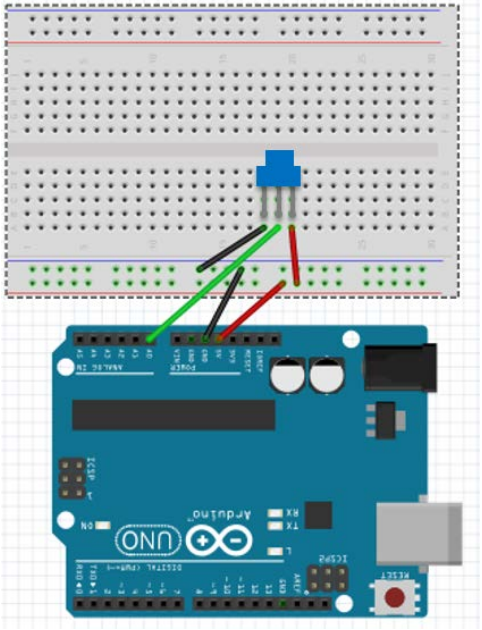



**Answer Questions 1, 2, and 3.**

Build this circuit on a breadboard – note the schematic symbol for the potentiometer (your potentiometer may not look exactly like the one below). Use the potentiometer that has a total resistance of 10kΩ. The way you can tell is by looking at the markings on the side (see figure – the 103 means 10 x $10^3$). A suggested layout is shown in the physical layout diagram to the right. As you can see the 5V and ground connections are provided by the Arduino/RedBoard.



## Programming the Arduino

The code that you upload to the board will read the voltage on analog input pin A0. Depending on the value of the voltage the state of the LED will be coded. This is how the feedback from the analog inputs can be used to change the state of other devices connected to the digital output pins.

- ✓ Open a new Sketch in the Arduino IDE. It should give you as always the bare minimum scaffolding. If not, type this in. Note that anything that follows the double-forward slash (**//**) is treated as a mere comment to the user of the code and is not evaluated by the IDE interpreter.

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {

  int sensorValue = analogRead(A0); //This statement creates a variable sensorValue of
                                    // of type integer AND associates the 10-bit binary
                                    // number read from Analog Pin A0.

}
```

- ✓ Now, upload and run the program.

## Testing and Debugging the Input Circuit

The input and output circuits are built and the Arduino is programmed to read the voltage connected to pin A0. Is it working? By looking at the code downloaded to the Arduino board and the circuit you built you can describe how it should work. It is essential that, even in this simple example, you get into the habit of checking that each portion of your design works properly so let's start with the input.

At your bench in the lab, you could use the voltmeter to confirm your predictions. Is your program working? You can fiddle with the potentiometer knob, knowing that you are probably changing the voltage. Probe the voltage between the analog input pin

A0 and one of the GND pins with the voltmeter. As you turn the knob on the potentiometer describe what happens to the voltage.

**Answer Question 4.**

Is Arduino correctly reading this voltage? We need to see the resulting 10 – bit binary numbers that represent the input voltage. The measured voltage are translated to 10 bits each by the A/D and sent through the USB port using a special protocol (computer standard for talking). The letters USB (each letter is pronounced YU-ES-BE) is an acronym that means *Universal Serial Bus*. *Universal* seems to be appropriate as the interface is ubiquitous. *Serial* (as opposed to parallel communication) implies that the digital information is transmitted 1-bit at a time. That is why serial communication is usually slower than the protocols that send entire words at once. *Bus* implies a sharing of data signals between different pieces of hardware. The board actually supports at least three types of serial communication protocols, only one of which is used by the USB connection.

- ✓ Amend the program so that it uses the serial connection provided by the USB cable to provide information about what the board measures.

```
void setup() {

  Serial.begin(9600);  //This statement sets up the communication channel between the computer and
                       // the board.

}

void loop() {

  int sensorValue = analogRead(A0); //This statement creates a variable sensorValue of
                                    // of type integer AND associates the 10-bit binary
                                    // number read from Analog Pin A0.


  Serial.println(sensorValue);

}
```

All of the statements that begin with *Serial.* are statements accessed from the Serial Communications libraries. The first statement associated with the serial communication functionality is the *Serial.begin(9600)* statement. This sets up the serial port between the computer and the Arduino. The speed of serial connection is specified by the number 9600 – this is not the only possible speed but is the most common for communications with Personal computers. The rate or units associated with serial communication is termed baud so the above statement says the interface runs at 9600 baud. What is a baud? The term Baud hails back to telegraphy when the quintessential serial communications method, Morse Code, was used. Morse code

assigns each letter of the alphabet a sequence of shorter (dots) or longer duration (dashes) voltage pulses. A baud was equal to sending a single "dot" in one second. A good operator can send upwards of 60 or more words per minute with a word equaling 5 characters. The longest letters are 4 symbols which makes 60 words per minute approximately 1200 symbols per minute or 20 symbols per second. This correlates to speeds of 20 baud or more. The computer communicates at 9600 baud.

The second statement *Serial.println*(sensorvalue) prints out the numerical value specified as an argument using C/C++ formatting conventions.

✓ Upload this new program to the Arduino and click on the icon that looks like a magnifying glass located at the top right of all open windows. A new window should pop-up and numbers should appear (give it a moment) streaming continuously – one per line. This window is often called the *console* or *Serial Monitor*.

Insert a statement that converts the integer sensorValue to the corresponding voltage. Store this value in a floating point variable voltageValue. It should look something like this:

```
float voltageValue = 5.*sensorValue/1023.;
```

NOTE: Do not forget the periods after the numbers. If left off, since sensorValue is an integer, the divide by 1023 may result in a zero which is not what you want. Print it out to the console with a simple addition to the print statement (see program below).

```
void setup() {

  Serial.begin(9600);  //This statement sets up the communication channel between the computer and
                       // the board.

}

void loop() {

  int sensorValue = analogRead(A0); //This statement creates a variable sensorValue of
                                    // of type integer AND associates the 10-bit binary
                                    // number read from Analog Pin A0.

  float voltageValue = 5.*sensorValue/1023.; //This statement maps the integer in sensorValue
                                             // to the actual voltage.
  Serial.println(voltageValue)

}
```

**Answer Question 5.**

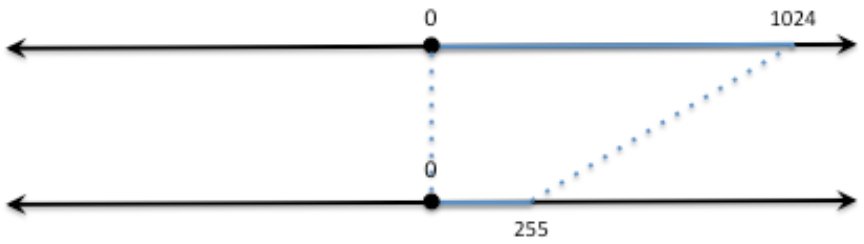
## Using the Sampled Data to control the Output Circuit

As you have discovered, the connection to the A0 pin is the "analog" signal varied by turning the knob on the potentiometer whose continuous voltage range (0-5V). The integer value read from the pin connected to an analog-to-digital converter is used within the loop section of the program to vary the interval used to turn the LED ON and OFF at a rate dependent on the voltage of the sampled signal. There are several parameters needed to characterize the A/D process. The **dynamic range** *is the range of voltage to be sampled* – in this case the dynamic range is 5V. The dynamic range would have been the same if the voltages varied from -1V to 4V. The **sampling rate** is the frequency at which the samples are obtained – in the case of the Arduino unless you access the processor directly you are limited to 10Ksamples/sec. The **resolution** specifies the number of values mapped to the dynamic range – in this case the voltage range 0-5V is mapped onto the integers 0-1023 in equal intervals.

**Answer Question 6.**

Let's use the value stored in *sensorValue* to control the brightness of the LED connected to Pin 13 (it comes soldered on your Arduino). A single statement *analogWrite*(ledPin, dutyCycle); is used to generate a square wave of peak-to-peak amplitude 5V, period 2µs and a variable duty cycle. The brightness of the LED is directly related to the duty cycle. The problem comes when specifying the duty cycle. The value stored in variable *sensorValue* varies between 0-1024, but the value passed to the *analogWrite* statement specifying the duty cycle can only vary between 0-255. So the two intervals must be mapped onto one another using a very simple method – divide the value stored in the variable *sensorValue* by 1024 reducing the interval from (0,

1024) to (0, 1) then multiply by 255 to map it onto the interval (0,255). That was easy. It is exactly identical to the equation used to convert the integer to the actual voltage.




**Answer Question 7.**

Now let's control the LED using the **analogWrite** function to control the brightness of the LED with the potentiometer. The code is provided below EXCEPT the statement that converts the input associated with potentiometer.

Complete the statement computing the variable dutyCycle that is passed as the second argument to the **analogWrite** function so that the entire range 0-1024 is mapped onto the entire range 0-255 that specifies the duty cycle. Run the program to see if it works. Now turn the potentiometer knob.

```
void setup() {

  Serial.begin(9600);  //This statement sets up the communication channel between the computer and
                       // the board.

}

void loop() {

  int sensorValue = analogRead(A0); //This statement creates a variable sensorValue of
                                    // of type integer AND associates the 10-bit binary
                                    // number read from Analog Pin A0.

  float voltageValue = 5.*sensorValue/1023.; //This statement maps the integer in sensorValue
                                             // to the actual voltage.
  Serial.println(sensorValue,voltageValue);

  int dutyCycle=???????????

  analogWrite(13,dutyCycle);  //This statement generates a 5V peak-to-peak square wave with
                              // 2 microsecond period and a percent duty cycle equal
                              // to 100*dutyCycle/255.

}
```

Serial.println(sensorValue); **(Correction!)**
Serial.println(voltageValue);

**This statement is *not* true!**

**Answer Question 8.**

You have successfully built a working circuit where the feedback between an input circuit and the behavior of the output circuit is provided by code rather than circuitry.

Name: _____

Net ID: _____

# Summary for Arduino Analog Input

**Question 1:** What is the maximum and minimum voltage at Pin 2.

**Question 2:** Pretend that the potentiometer is set so that $R_1 = R_2 = R/2$. What is the voltage measured between Pin 2 and ground?

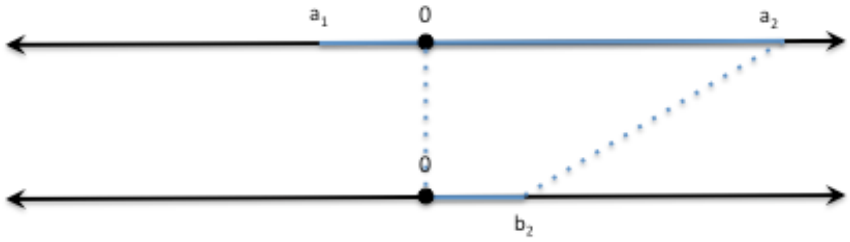
**Question 3:** If the knob is turned so that the resistance $R_1$ increases by a value d so that $R_1 = (R/2)+d$, what is the voltage measured between Pin 2 and ground?

**Question 4:** Describe the relationship between the voltages on the voltmeter and the potentiometer as you turn it.

**Question 5:** Twiddle the potentiometer and watch how the voltage changes on the serial monitor and the voltmeter. How close is the agreement?

**Question 6:** If the Arduino reads an integer value of 1 from pin A0 what was the voltage input?

**Question 7:** How would you map an arbitrary interval ($a_1$, $a_2$) to the interval (0, $b_2$)?




**Question 8:** What happens to pin13 LED as you adjust the potentiometer? **Comment**: Pin 13 (the LED) is a digital pin (and does not accept PWM). Explain why this matters in our usage here.