Module: Using the Arduino Analog Inputs

Introduction

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. This ability enables you to design projects that interface between the real world of sound, heat, pressure, motion, acceleration, radiation... to the digital world of the Arduino. The devices that convert the properties of the real world 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 – 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 get analog inputs. You cannot create them.

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 output of any analog device that outputs a voltage that varies between 0-5V.
Procedures

Electrical Interface

The devices that are commonly termed “integrated circuits” or ICs are circuits with recognizable schematics that are fabricated so that the individual devices are now on the order of 20-30nm. To put this in perspective the wavelength of violet light is 380nm. They are really small and getting smaller. Because of the small size very complex circuits can be fabricated on a very small area. If your laptop were built using 1960’s technology the state of Rhode Island would be a computer (sarcasm). Because of the size all of these circuits must be low power. Because of the small size allowing for greater complexity the electrical interface to all integrated circuits must be predictable so the ICs can be easily connected without having to consider the electrical considerations.

So before launching into the logistics of building an actual circuit and learning the associated commands that poll the analog inputs on the Arduino/RedBoard let’s look at the magic that is the digital hardware interface. Though there are many types or families of digital hardware the electrical interface of all hardware considered “digital” share some common characteristics.

Common characteristics:

1. All devices assume that the power demand from the devices is fairly low.
2. The inputs to the ICs are assumed to look like a perfect voltage measuring device in the sense that it draws VERY little current from any circuits connected to it. Assuming of course that you work within the accepted voltage and current ratings.
3. The outputs to the ICs are assumed to behave like perfect sources supplying the necessary current to maintain the voltage at one of two values – a HIGH or LOW voltage set by the logic family used. For the Arduino/RedBoard the two voltage levels are 0V LOW and 5V HIGH. Assuming of course that you work within the accepted voltage and current ratings.
Since we are interested in the inputs in this module you will be experimenting with the second characteristic – draws little current from devices connected to it.

- Hook the Arduino/RedBoard to the computer via the USB port.
- Open the Arduino IDE and navigate using the menu to Files > Examples > 01.Basics > ReadAnalogVoltage
- Upload the code to the board. For reference the code is given below.

This module will introduce you to the statements that might be unfamiliar but for now all we need to know is that an analog voltage is assumed to be connected to the pin A0. This signal is translated from an analog voltage using an Analog-to-Digital (A/D) converter to an 10-bit binary integer (0-1023). That integer is translated to a real number that represents the voltage read and this number is printed to the CONSOLE where you can watch the change.

- Access the CONSOLE by clicking on the button with the magnifying glass in the upper right corner of the window containing the program in the Arduino IDE.

**Question 1:** What value appears repeatedly in the CONSOLE window?

- Now set the power supply voltage to 0V in 6V mode. Put in the OUTPUT OFF state.
- Connect the positive terminal using a banana cable with and adapter or alligator clip with a wire to the pin labelled A0.
- Connect the negative terminal to any of the pins labeled GND. NOTE: all voltages on the board are referenced to the GND pins all of which are connected together.
- Take the power supply out of the OUTPUT OFF state.

**Question 2:** What value appears in the CONSOLE window now?
Question 3: Increase the voltage – DO NOT let the voltage get above 5V! Do the numbers track the voltage displayed by the power supply?

Question 4: What is the maximum current drawn according to the value displayed by the power supply? Is this behavior consistent with the claim that the digital inputs behave like ideal voltage measuring devices?

Question 5: Keep watching the console values and quickly disconnect the connection to the power supply. Describe what happens. I am not entirely sure why it does this... Most digital ICs have a state that it holds when the input pins are not connected. TTL logic assumes that an unconnected input is 1 or ON or TRUE.
ReadAnalogVoltage

/*
  ReadAnalogVoltage
  Reads an analog input on pin 0, converts it to voltage, and prints the result to the serial monitor.
  Attach the center pin of a potentiometer to pin A0, and the outside pins to -5V and ground.
*/

This example code is in the public domain.
*/

void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
Building a circuit to test

Build Circuit #2 in the SParkFun Inventors Kit (SIK) Guide. It introduces you to a device called a potentiometer that has a knob that can be turned to change the voltage measured between the center pin of the device and the pin connected to ground. The schematic from the Guide is repeated below for your convenience. **READ THE COMMENTS WRITTEN in the CODE.** They describe the components and circuit very well, how they work, and how the Arduino interfaces with the circuit. The schematic of the circuit with a copy of the code without the comments are provided below. **NOTE: the Arduino expects voltages only in the range 0-5V.**
Describe what happens to the LED when you twist the knob on the potentiometer.
**Question 7:** Describe how the Arduino/RedBoard is controlling the circuit, referring to the code and how each statement is physically manifested. How, through the program, is the physical act of twisting the knob on the potentiometer translated into the behavior of the LED.
Testing and Debugging the Circuit

The circuit is built and the Arduino is programmed. 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.

At your bench in the lab you have available many useful debugging tools. Since the circuit is designed to produce a time varying signal using the oscilloscope comes to mind.

**Question 8:** If you were at the bench and you wanted to use the oscilloscope to monitor the voltage across the potentiometer where would you connect the two terminals of the oscilloscope probe? Notate these 2 points on the schematic below.
Away from the bench and an oscilloscope, you can still debug the circuit if the Arduino is connected to a computer running the Arduino IDE. All communication to the Arduino happens using some type of *serial communication* – and the boards supports a few. 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.

Using the serial interface is very easy from the Arduino IDE. Look at the same code you downloaded to the board (shown below) with two extra statements added to it. The first `Serial.begin(9600)` 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 commonly used by USB connections to personal computers. The rate associated with serial communication is termed baud so the above statement says the interface runs at 9600 baud. What is a baud? The second `Serial.println(sensorValue)` prints out the numerical value specified as an argument using C++ formatting conventions.

```c
/*
SparkFun Inventor's Kit
Example sketch 02
*/

int sensorPin = 0;  // The potentiometer is connected to
                     // analog pin 0
int ledPin = 13;   // The LED is connected to digital pin 13

void setup() { // this function runs once when the sketch starts up
    Serial.begin(9600);                              
    pinMode(ledPin, OUTPUT);                         
}

void loop() { // this function runs repeatedly after setup() finishes
    int sensorValue;

    sensorValue = analogRead(sensorPin);           
    Serial.println(sensorValue);                   
    digitalWrite(ledPin, HIGH);                    
    delay(sensorValue);                            
    digitalWrite(ledPin, LOW);                     
    delay(sensorValue);                            

    // Remember that loop() repeats forever, so we'll do all this
    // again and again.
}
```
Make the two simple changes and 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.

**Question 9:** Describe what happens to the numbers as you twist the knob on the potentiometer. Specify the maximum and minimum values you read from the console window when the potentiometer is fully turned in both directions.

**Question 10:** Explain how the values relate to the behavior of the LED.
Using the Sampled Data

As you have discovered, the connection to the A0 pin is an “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.

**Question 11:** What is the voltage range that is mapped onto a single integer interval. For example, if the Arduino reads a value of 1 from pin A0 what was the voltage input?

Instead of turning the LED ON and OFF using the `digitalWrite` statements, the `analogWrite` statement can be used. While the `digitalWrite` statement simply holds the designated output pin to either a HIGH or LOW voltage level, the `analogWrite` sends a square wave pulse to the designated pin with a specified duty cycle. A single statement `analogWrite(ledPin, dutyCycle);` The problem comes when specifying the duty cycle. The value of variable `sensorPin` 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. This case is very simple – divide the value so `sensorPin` 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.

![Diagram showing voltage mapping](image-url)
Question 12: How would you map the intervals in this case – map an arbitrary interval $(a_1, a_2)$ to the interval $(0, b_2)$?

So 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.

Question 13: Complete the statement 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.

```c
/*
SparkFun Inventor's Kit
Example sketch 02
*/

int sensorPin = 0;  // The potentiometer is connected to
                    // analog pin 0
int ledPin = 13;    // The LED is connected to digital pin 13

void setup() // this function runs once when the sketch starts up
{
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() // this function runs repeatedly after setup() finishes
{
  int sensorValue;
  int dutyCycle;

  sensorValue = analogRead(sensorPin);
  Serial.println(sensorValue);
  dutyCycle = ??????
  analogWrite(ledPin, dutyCycle);

  // Remember that loop() repeats forever, so we'll do all this
  // again and again.
}