Arduino Sensors Workshop

Developed by Mark Webster, Ousema Zayati, Conner Sheeran

In this workshop learn to identify several types of sensors and to use them in microcontroller projects.

Background

Sensors are devices that convert measurements of real world data into electrical signals that can read by a microcontroller such as an Arduino. Code on the microcontroller interprets the signals and takes an action such as storing the data, transmitting it by serial or RF to another computer, or turns on motors, alarms, or triggers other devices to do something.

Sensors can measure: temperature, humidity, barametric pressure, GPS coordinates, moisture, infrared light, heart beat, brainwaves, magnetic fields, CO2 levels, distance, movement, or countless other specialized real world data. (The SCC Makerspace electronics area has many examples of sensors.)
[Insert photo here of many types of sensor]

Sensors produce an analog voltage, or a series of digital pulses. Arduino Uno's read analog voltages (0 to 5 volts) on the six pins A0 to A5. It can read digital values (0 or 1) on pins 2 to 13.

Some sensors are a variable resistance which is converted to a voltage by putting the sensor into a resistor divider circuit between 0 and 5 volts.

A few sensors return a pulse on a digital pin but the length of the pulse encodes information (PWM – pulse width modulation). PWM can be accessed or written on the pins 3,5,6,9,10, and 11.

Digital sensors usually work with either the I2C or SPI protocols for communication. The Arduino library "wire.h" is used for I2C devices. The Arduino "SPI.h" library is used for SPI devices.

Analog Input

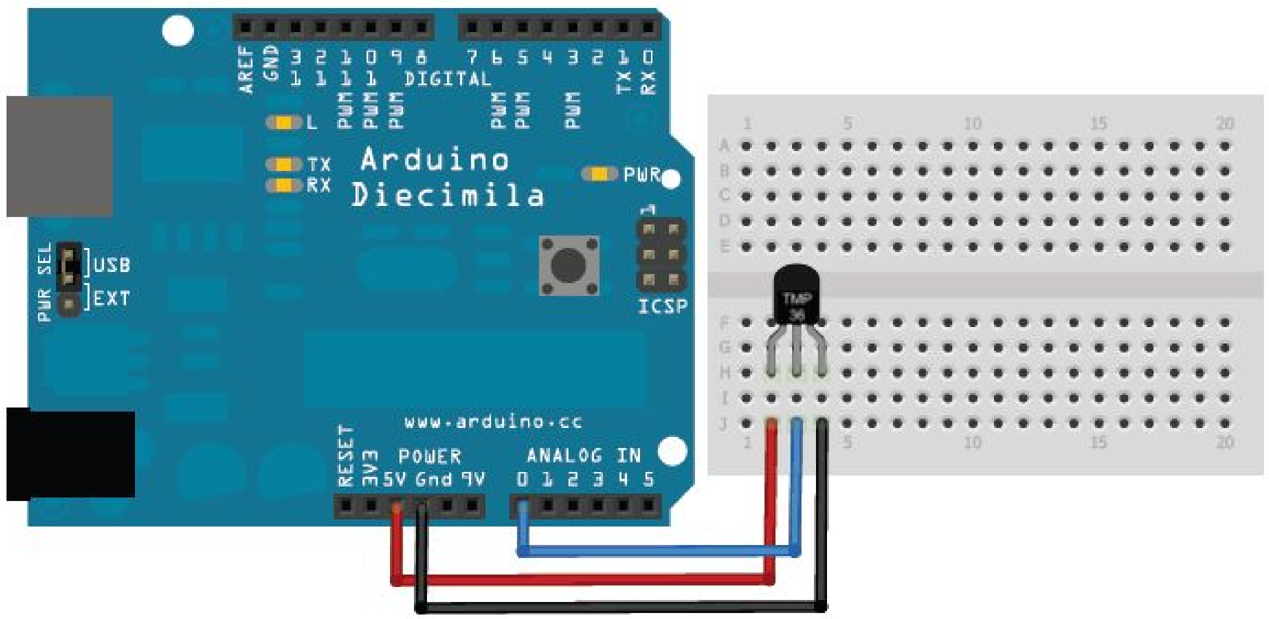
An Arduino has several input pins with an AD (analog to digital) converter. The Arduino has a 10 bit converter that maps a voltage from 0 - 5V to an integer between 0 - 1023.
When a sensor has an analog voltage output, the Arduino program reads the voltage and applies some formula to convert it to the value being measured.

Analog sensors are perhaps the easiest to implement. However, their resolution is limited and often there is jitter in the values. Jitter can be reduced with external electronic filters, or digital filters in the software.

Exercise 1: TMP36 Temperature Sensor

The TMP36 is an analog sensor that converts temperature into an analog voltage. The TMP36 has three pins, ground, readout, and VCC. The Arduino reads an analog voltage as a number from 0 to 1023. First convert that to a voltage between 0 – 5000 mv. Then convert that reading to degrees where the resolution is 10 mv per degree C, and the 0 C offset is 500 mv.

This sketch reads the sensor output on analog pin A0, and converts to fahrenheit. This sketch was originally copied from adafruit.com and modified. The temperature data is sent to a computer using the serial library to display in the IDE’s serial monitor.



```
// Makerspace Arduino Sensors workshop TMP36

//TMP36 Pin Variables
int sensorPin = 0; //the analog pin the TMP36's Vout (sense) pin is connected to A0
                   //the resolution is 10 mV / degree centigrade with a
                   //500 mV offset to allow for negative temperatures
float voltage;
float temperatureC;
float temperatureF;

void setup()
```

```
{
  Serial.begin(9600);     //Start serial interface
}

void loop()                    // run over and over again
{
 //getting the voltage reading from the temperature sensor
 int reading = analogRead(sensorPin);

 // converting that reading to voltage, assuming 5.0 volts on an Arduino Uno
 voltage = reading * 5.0/ 1024.0;

 // now print out the temperature in celsius
 temperatureC = (voltage - 0.5) * 100 ;  //converting from 10 mv per degree wit h500 mV
offset
                                              //to degrees ((voltage - 500mV) times 100)
 Serial.print(temperatureC);
 Serial.println(" degrees C");

 // now convert to Fahrenheit
 temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;
 Serial.print(temperatureF);
Serial.println(" degrees F");

 delay(1000);                                    //waiting a second
}
```

Digital input

An Arduino has many pins that can read a digital signal (On or Off, 1 or 0). Some sensors send a string of digital pulses as output which the Arduino software can convert to a value. Sometimes the sensor value is the time between a trigger on one pin and another input pin going high.

Exercise 2: HC-SR04 Distance Sensor

The HC-SR04 sensor works by sending out a pulse of ultrasonic sound (40 khz) and timing how long it takes for an echo to return. This is similar to how bats or dolphins measure distance. The time is encoded in a pulse length read on a digital pin.

To calculate the distance one must know the speed of sound in air. The speed of sound changes with temperature and humidity, so for best accuracy one should measure the temperature and humidity along with the echo time.

Distance is the simple equation: *distance = speed ×time*

The equations for *speed* of sound in air (in meters per second) is approximately:

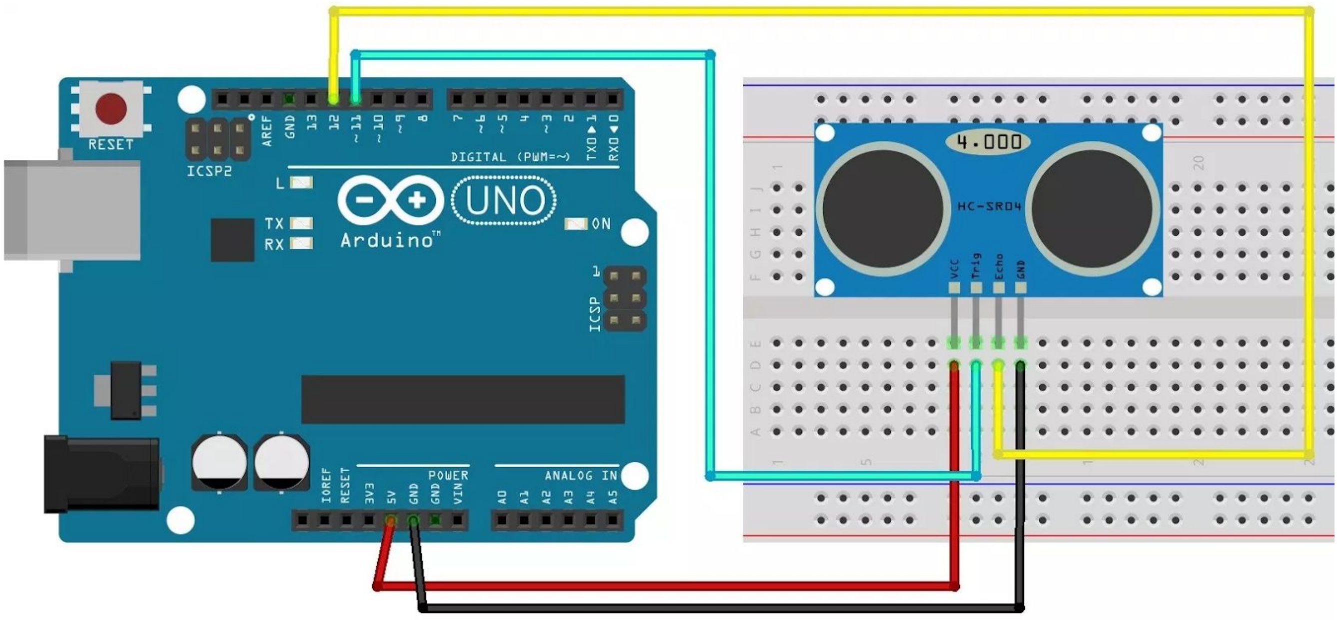
$s = 331.4 + (0.606 \times T) + (0.0124 \times H)$

331.4 is the speed of sound in m/s at 0° C at 0 % humidity
T is temperature in ° C, H is % relative humidity.

For example, at 24° C and 25% humidity (typical Sacramento fall weather)

$s = 331.4 + (0.606)(24) + (0.0125)(25) = 346.26$ m/s

The HC-SR04 has a trigger pin which sends a pulse when set high for a 10 microseconds. The time for the pulse to reach the object and return is found by reading the length of a digital pulse in microseconds. In this simple code the temperature and humidity are ignored. The speed of sound in m/s is converted to cm/ microsecond. For example, a typical 344 m/s becomes 0.0344 cm/microsecond.
The duration of the echo is divide by 2 to account for the round trip. The valid range for the HC-SR04 sensor range is 2 to 400 cm, so any readings outside of that are ignored. For better accuracy an individual sensor can be calibrated and an adjustment added to the distance. In this sketch the adjustment is set to 0.0 cm.
In this sketch the trigger pin on the HC-SR04 is connected to digital pin 9, and the echo pin is connected to digital pin 10. The serial monitor (Serial library) is used to send the calculated distance back to another computer.
This sketch was copied from www.circuitbasics.com and modified.



```
// Maker Arduino Sensor Workshop DHT11

// Define variables that don’t change
#define trigPin 9
#define echoPin 10
#define distanceAdjust 0.0

// Setup at the start of the program
void setup() {
  Serial.begin (9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

// Main loop which runs forever
void loop() {
  // Local variale declarations
  float duration, distance;


// Send out an ultrasonic pulse, 10 microseconds long
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Find duration of the echo
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance from pulse length
  distance = (duration / 2) * 0.0344 + distanceAdjust;

  // If out of sensors bounds, give error
  if (distance >= 400 || distance <= 2){
    Serial.print("Distance = ");
    Serial.println("Out of range");
  }   else {
    // valid distance
    Serial.print("Distance = ");
    Serial.print(distance);
    Serial.println(" cm");
  }
   delay(800);
}  // end of main loop
```

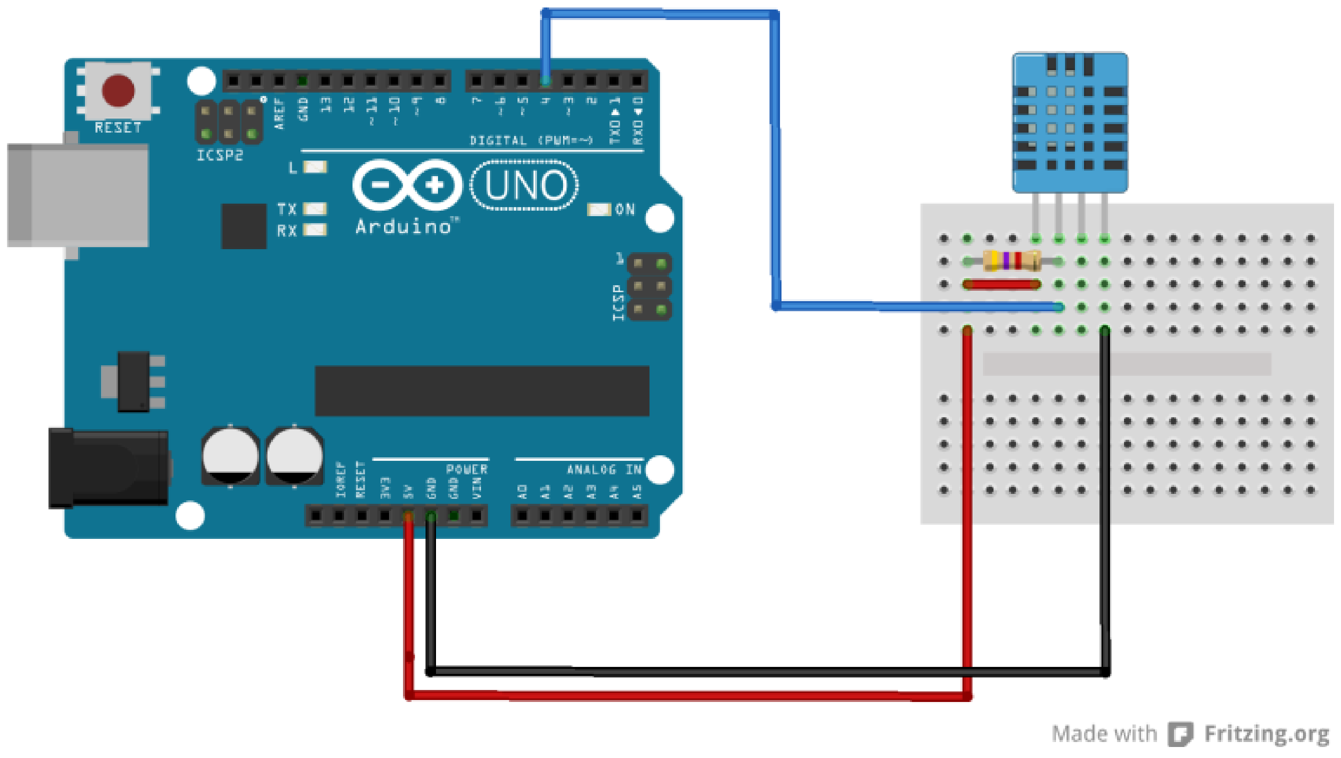
Sensor With Library

The most complex and flexible sensors have a complicated set of signals read on a digital input pin. Usually a library that some other person has written is used to hide the complexity of interpreting the signal pattern and turning it into a meaningful sensor value.
The challenge with library based sensors is there may be different libraries on the Internet designed to work with older versions of the sensor. How to use a library found on the Internet can be challenging since example programs may work with an older version of the library. The challenge is to find a library and example program that work together. In desperate cases, the programmer must read the C++ code of the library and figure out how it works.

Exercise 3: DHT11 Temperature and Humidity Sensor

The DHT11 sensor converts the data measurements to a digital signal. The DHT11 sends out 40 bits (5 bytes) which contain the temperature and humidity. Fortunately, dedicated open source programmers have written a library called “dht.h” which does all the messy work of reading and converting the 40 bits to integer values stored in the object members dht11.humidity, dht11.temperature.
The source code for dht.h can be found on github.com, although it usually isn’t needed. The library must be downloaded and installed into the Arduino IDE before using this sensor.
The DHT11 sensor returns temperature as a whole number between 0 and 50 degrees Celsius. The humidity is a whole number between 0 and 100% relative humidity.

This sketch was copied from www.circuitbasics.com and modified.



```
#include <dht.h>

// Define an instance of the dht object, which is then referenced in the program
dht DHT;

// define the Arduino digital pin to use
#define DHT11_PIN 7

// Initialize serial output to display the values read.
void setup(){
  Serial.begin(9600);
}

void loop()
{
// Error messages are put into the chk variable. Here we ignore them.
  int chk = DHT.read11(DHT11_PIN);
  Serial.print("Temperature C = ");
  Serial.println(DHT.temperature);
  Serial.print("Humidity = ");
  Serial.println(DHT.humidity);
  delay(1000);
}
```