Lab 2: Analog and Digital Sensors

Due Friday September 16, 2016 @ 5pm PT on bCourses

Images developed using Fritzing.

Part 1: Simple Sensors

Exercise 1: Basic Digital Input

Items:

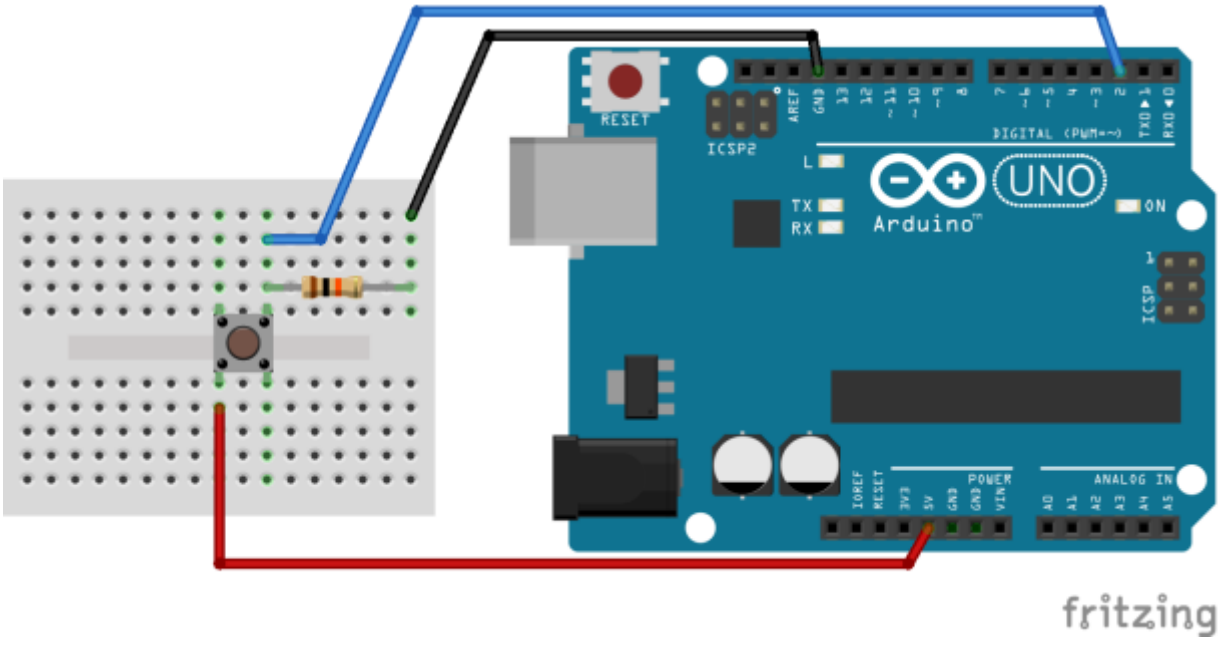
- 1x Arduino Uno Board with USB Cable
- 1x Switch, Toggle, Pushbutton, or Momentary Switch
- 1x 10k Resistor (brown-black-orange)
- Jumper Wires

Deliverables:

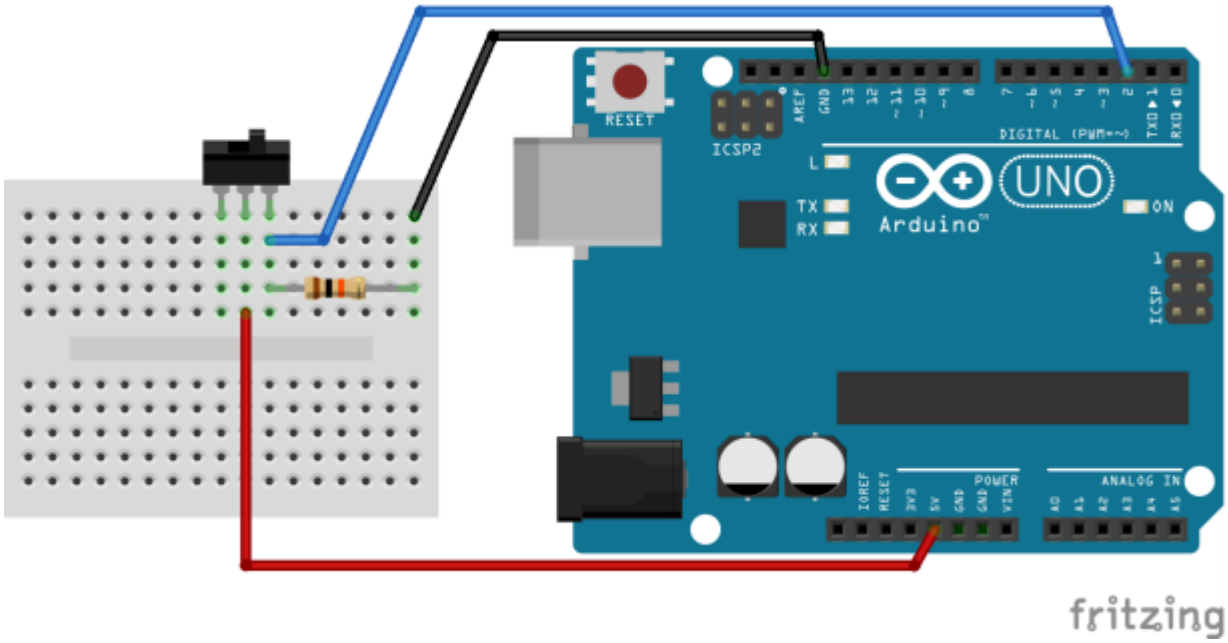
- None

Directions:

1. Using either a button or a toggle switch, setup the circuit shown below.



If you are using a button or momentary switch, position the button as shown above. Note that the left pins are connected within the button and that the right pins are connected as well. Connect the left (or right) pins of the button to 5v and connect the other pins to pin 2 on the Arduino. Lastly, connect pin 2 to ground through a 10k resistor (This resistor is referred to as a pull-down resistor because when the button is open or removed, the digital signal is pulled to ground).



If you are using a toggle switch, connect the center pin of the switch to 5v and connect the right (or left) pin to pin 2 on the Arduino. Lastly, connect pin 2 to ground through a 10k resistor (This resistor is referred to as a pull-down resistor because when the switch is off or removed, the digital signal is pulled to ground).

2. Open the Arduino Button example (under File->Examples->02.Digital->Button) and upload it to your Arduino board. Note that you may be using a switch instead of a button, but the functionality/code is the same.

Exercise 2: Basic Analog Input

Additional Items:

- 1x 10k Trimpot with Knob

Deliverables:

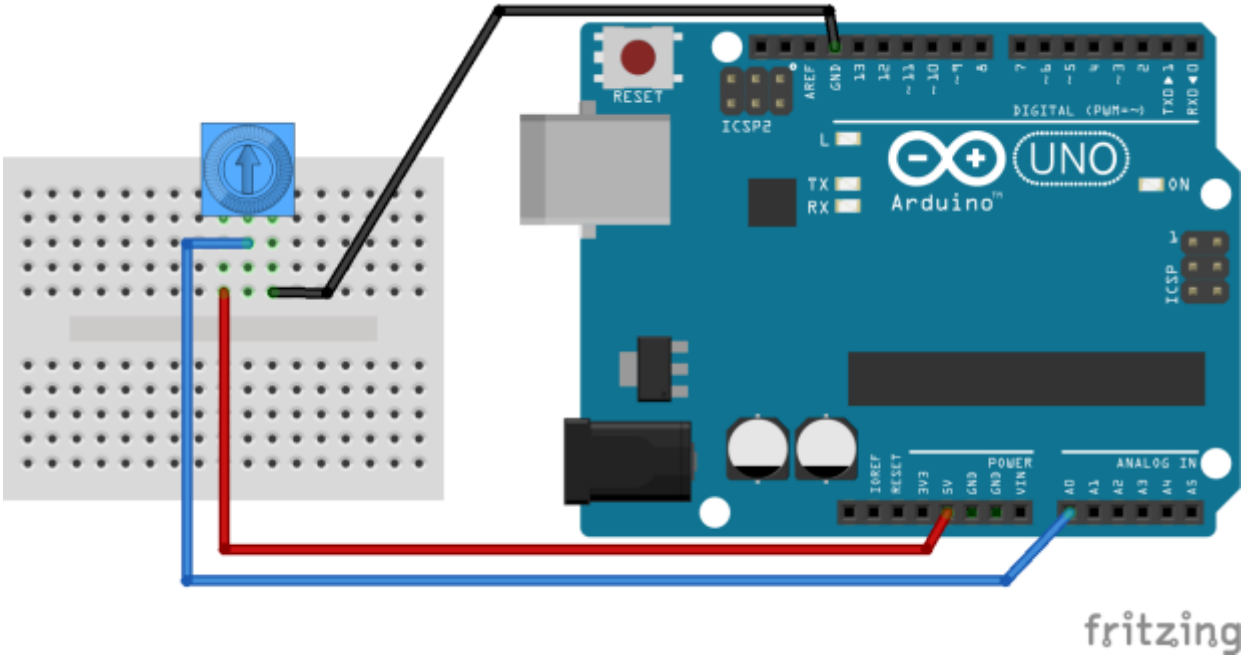
- None

Directions:

1. A trimpot is a trimmable potentiometer, also known as a variable resistor. With a 10k trimpot, you can change the resistance of the device from (around) 0 to 10k ohms by simply turning the

knob. When connected to 5v and ground, this means that we can change the analog output of the trimpot from 5v (0 ohms) to 0v (10k ohms).

2. To use the trimpot, connect the center pin to analog pin A0 and the outside pins to ground and 5v (it does not matter which is which, but switching ground and 5v will change the voltage output), as shown below.



3. Open the Arduino AnalogReadSerial example (under File->Examples->01.Basics-> AnalogReadSerial) and upload it to your Arduino board. Make sure you understand what the program is doing. Open the serial monitor and see how the values change as you adjust the knob of the trimpot. Note that the value 1023 indicates an output voltage of 5v.

Exercise 3: Analog Sensor with Voltage Divider

Additional Items:

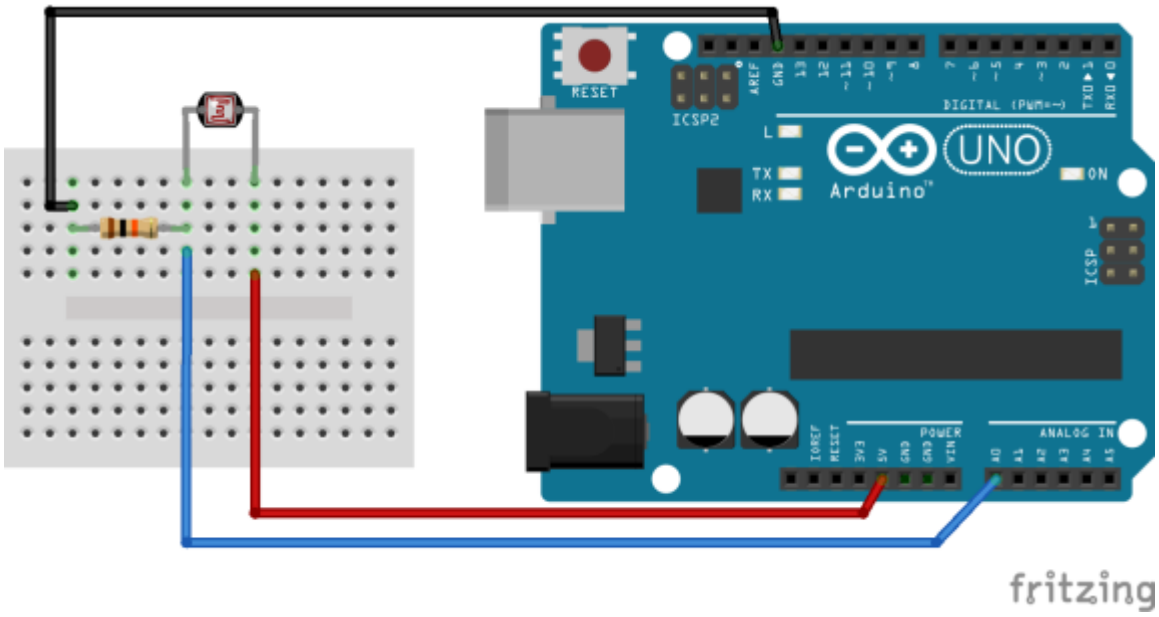
- 1x Photo Resistor
- 1x 10k Resistor (brown-black-orange)

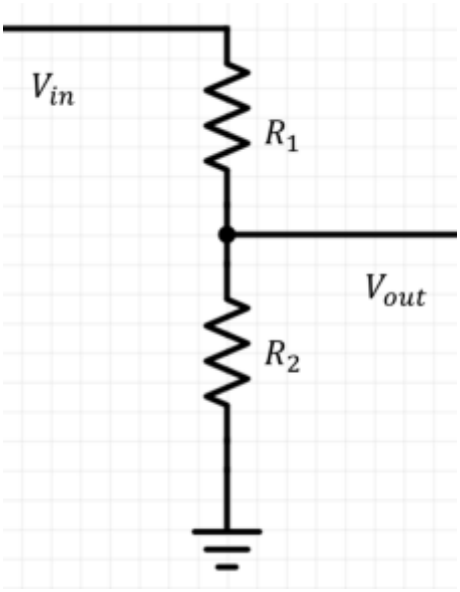
Deliverables (2 pts):

- Submit a file named PhotoResistorSerial.ino containing the program described below. Additionally, in your lab report, copy & paste your Arduino code into a code box.
- In your lab report, include the 2 bit ADC table, the formula for computing the resistance of the photo resistor, and your response to **Questions**.

Directions:

1. Setup the circuit shown below. Connect one end of the photo resistor to 5v and the other end to analog pin A0. Next, connect pin A0 to ground through a 10k resistor.

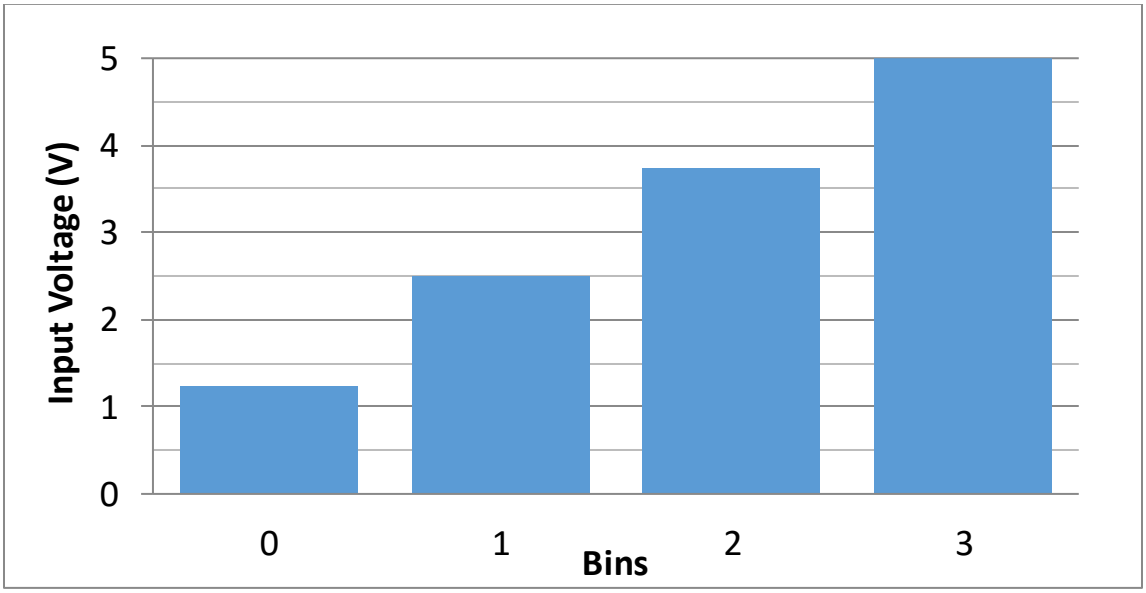




$$V_{out} = \frac{R_2}{R_1 + R_2} * V_{in}$$

2. Open the Arduino AnalogReadSerial example (under File->Examples->01.Basics-> AnalogReadSerial), change the delay from 1 millisecond to 1 second, save the program as PhotoResistorSerial, and upload it to your Arduino board. Open the serial monitor and confirm the Arduino prints values corresponding to the light intensity sensed by the photo resistor. The provided photo cell is not a particularly accurate light sensor. However, it is suitable for detecting changes in the light levels.
3. The circuit we have created is a simple voltage divider, as shown in the image above. The input voltage, V_{in}, is 5V (5000 mV) and the analog input A0 is related to the output voltage, V_{out}. Resistor R_2 is the 10,000 ohm resistor and R_1 is the resistance of the photo resistor.

4. Note that the values reported by the Arduino are not between 0 and V_{in} (5v). The Arduino is a digital computer and so it cannot directly read an analog signal. Rather, the Arduino has a 10 bit analog-to-digital converter (ADC). Basically, the ADC turns an analog signal (in this case, 0 to 5v) into an integer between 0 and 2^{10} -1. Your task is to convert the integer value into an integer or float that represents our desired signal.



First, we will complete a short exercise using an imaginary 2 bit analog-to-digital converter (ADC) to gain a better understanding of what is happening. With a 2 bit converter, analog voltages between 0V and 5V are converted to integers between 0 and 2^2 -1. In other words, input voltages are placed in one of 2^2 bins. The first bin is for input voltages where $0 \leq V_a < 1.25$. The second bin is $1.25 \leq V_a < 2.50$. The third bin is $2.50 \leq V_a < 3.75$ and the last bin is $V_a \geq 3.75$. Complete the table below. Note that there are many ways to map a digital value to a voltage, however, in this example, we are mapping voltages to the bottom of the voltage range. Also, the mapped voltages should be integers:

Actual Voltage (V)	Voltage Range	2 bit Digital Value	Mapped Voltage (mV, integer)	Mapped Voltage (V, integer)
0.00	0 to 1.25	0	0	0
0.50	0 to 1.25	0	0	0
1.5	1.25 to 2.50	1		
2.51	2.50 to 3.75			
3.2				
5.00				

With a 10-bit converter, we now have 2^{10} = 1024 bins and the size of the bins becomes 0.004883 V, much smaller than the 1.25 V bin size we had with the 2-bit converter.

Lastly, Arduino has a function called map() that allows you to easily convert the integer from analogRead() to the desired (integer) value. To convert the input to volts, use:

```
int volt = map(val, 0, 1023, 0, 5);
```

To convert to millivolts, use:

```
int millivolt = map(val, 0, 1023, 0, 5000);
```

Note that with the map() function, we are not mapping the voltages to the bottom of the voltage range. Rather, map() implements the linear function:

$$y = f(x, x_{min}, x_{max}, y_{min}, y_{max}) = (x - x_{min})\left(\frac{y_{max} - y_{min}}{x_{max} - x_{min}}\right) + y_{min}$$

If x_{min} and y_{min}, are 0, then

$$y = f(x, 0, x_{max}, 0, y_{max}) = \frac{x * y_{max}}{x_{max}}$$

Modify your code to calculate and print V_{out}, the voltage across R_2, in both volts and millivolts.

5. **Questions**:
    a. Derive a formula to compute the resistance of the photo resistor, R_1, using Kirchoff's voltage law. Include this formula in your lap report.
6. In your program, calculate the value of R_1 and print it to the Serial port.
    a. Note: Use millivolts for V_{in} and V_{out}
    b. Note: Remember to pay attention to integer arithmetic.
    c. Note: The maximum value of an integer is 32,767. Look out for integer overflow.
7. **Questions**:
    a. What range of R_1 values do you observe? Try shading the sensor or shining a light on it.
    b. Is the R_1 value directly proportional or inversely proportional to the lighting level?

Exercise 4: Temperature Sensor

Additional Items:

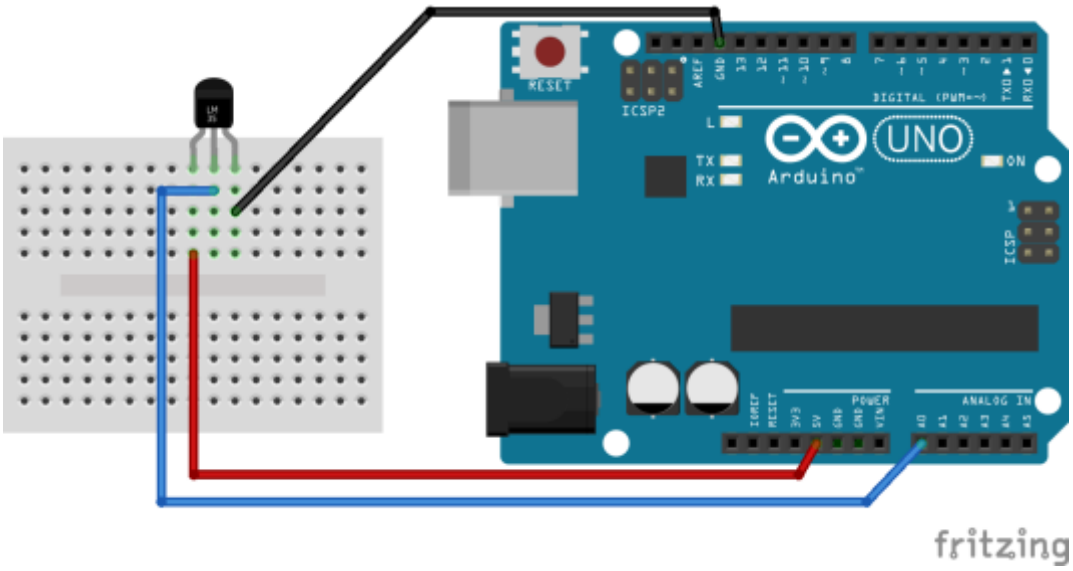
- 1x Temperature Sensor (TMP36)

Deliverables (1.5 pts):

- Submit a file named TemperatureSerial.ino containing the program described below. Additionally, in your lab report, copy & paste your Arduino code into a code box.
- In your lab report, include the formula for converting millivolts to degrees Celsius and your response to **Questions**.

Directions:

1. Setup the circuit shown below. Connect the center pin to analog in A0, the right pin to ground (or as labeled on the sensor), and the left pin to 5V (or as labeled on the sensor). Be careful not to mix up the left and the right pins or the sensor will quickly become very hot.



2. Open the Arduino AnalogReadSerial example (under File->Examples->01.Basics-> AnalogReadSerial), change the delay from 1 millisecond to 1 second, save the program as TemperatureSerial, and upload it to your Arduino board.
3. Convert the analog reading to mV and output to the Serial Monitor.
4. **Questions**: Read the datasheet at https://www.sparkfun.com/products/10988 for the TMP36.
    a. According to the datasheet, what is the "sensitivity" of this sensor?
    b. What is the specified temperature range?
    c. What is the output (mV) of the sensor at 25 degrees Celsius?
    d. What is the maximum output (mV) of the sensor?
    e. What is the (typical) accuracy at 25 degrees Celsius?

    The objective is to gain experience and comfort reading data sheets. This is a critical and practical aspect of any project involving sensors.

5. Derive a formula to convert the voltage reading to degrees Celsius. Include this formula in your code and in your lap report.
    a. Note: The temperature should be calculated as a float.
    b. Hint: Use a linear equation with a point-slope form.
    c. Hint: Do not use -40 or 125 degrees Celsius in your formula.
    d. Hint: 0 mV is NOT 0 degrees Celsius.
6. Print the temperature in degrees Celsius and in degrees Fahrenheit to the Serial monitor.
7. **Questions**:
    a. According to the sensor, approximately what is the current temperature, in degrees Celsius and in degrees Fahrenheit, in the room?

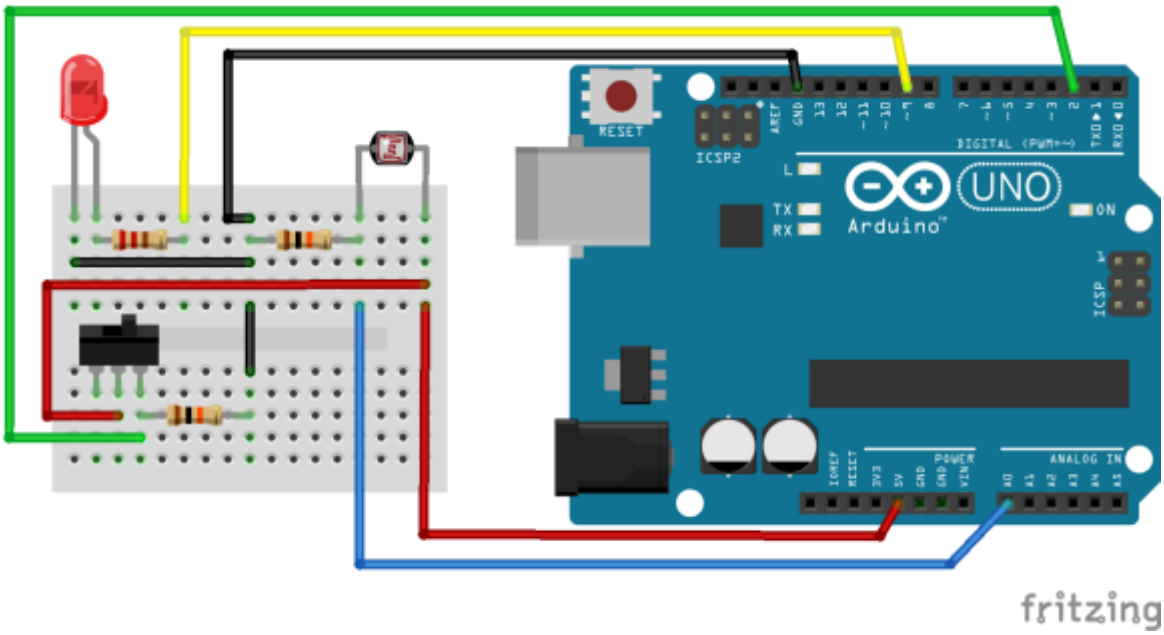
Exercise 5: Putting Everything Together

Deliverables (1.5 pts):

- Submit a file named LEDSystem.ino containing the program described below. Additionally, in your lab report, copy & paste your Arduino code into a code box.
- In your lab report, respond to **Questions**.

Directions:

1. Setup and program a system that fades an LED according to the light intensity but only if the switch/button is on. Your circuit might look something like the image below. Save your program as LEDSystem.ino.



    a. Hint: You may find the following helpful.

```
int mappedVal = map(val, 0, 1023, 0, 255);
```

2. **Questions**:
    a. What is the procedure (i.e. sequence of events) executed by your system? Explain briefly in your lab report.
    b. How many digital inputs, digital outputs, analog inputs, and analog outputs have you used?
    c. Did you scale your photo resistor analog-in values (i.e. convert to engineering units like lux or candle-feet) before applying a digital-out value to fade the LED? Why or why not?

Part 2: Digital Sensors and Protocols



Image Source: SparkFun

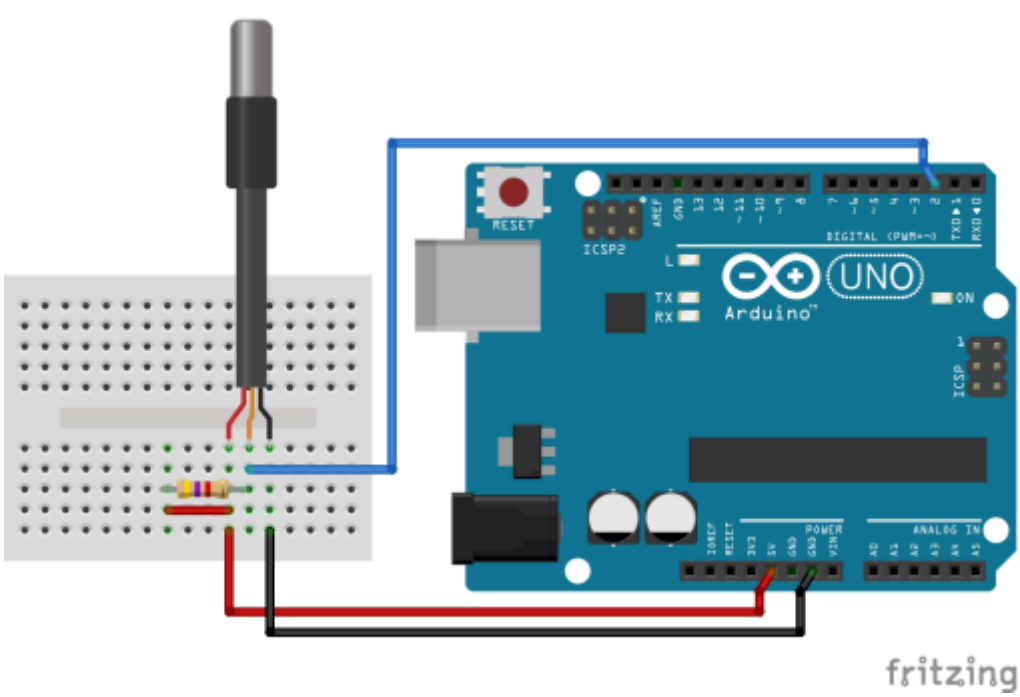
The DS18B20 is a digital temperature sensor that uses a 1-wire protocol. The sensor reports the reading in degrees C with a 9 to 12-bit precision over a -55C to 125C temperature range. Each sensor has a 64-bit serial number, allowing for multiple sensors to be connected to a single digital pin.

Features:

- Unique 1-Wire interface requires only one port pin for communication
- Each device has a unique 64-bit serial code stored in an onboard ROM
- Power supply range is 3.0V to 5.5V
- Measures temperatures from –55°C to +125°C (–67°F to +257°F)
- ±0.5°C accuracy from –10°C to +85°C
- Thermometer resolution is user-selectable from 9 to 12 bits
- Converts temperature to 12-bit digital word in 750ms (max.)
- User-definable nonvolatile (NV) alarm settings
- Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)

These exercises will get you up and running with the DS18B20 sensor. In the Exercises, we will take a closer look at communicating with the sensor and interpreting the digital signal. Lastly, we will introduce the DallasTemperatureControl library, which greatly simplifies working with DS18B20 sensors.

Setup:



Note: Please check that you are using the digital DS18B20 sensor and not the analog TMP36 sensor.

Connecting the sensor is rather simple. To start, connect the sensor's ground pin to ground and the power pin to 5V. Next, connect the single digital signal pin to a digital pin on your Arduino, such as pin 2. Lastly, the sensor requires a 4.7k pull-up resistor between the signal and power pins. If you are using a DS18B20 sensor with a breakout board, then the 4.7k resistor is already connected.

Library:

To communicate with the sensor, we need the OneWire library for Arduino. The library can be found in the Lab 2 Packet (->Arduino Libraries) or downloaded from playground.arduino.cc/Learning/OneWire. You will need to place the "OneWire" folder into your Arduino Library folder and then restart the Arduino IDE.

Default Arduino Library Folder Locations:

Mac: In (home directory)/Documents/Arduino/libraries

PC: My Documents -> Arduino -> libraries

Linux: (home directory)/sketchbook/libraries

Quick Start:

Upload the QuickStartDS18B20.ino program and view the sensor reading in the Serial Monitor.

Exercise 6:

Deliverables (1.5 pts):

- In your lab report, respond to **Questions**.

Directions:

8. In this exercise, we will be utilizing the datasheet for the DS18B20 digital temperature sensor to understand how to communicate with the sensor and to interpret the digital signal. Although the DS18B20 uses a unique 1-wire protocol rather than the more common SPI or I2C protocols, the process of sending commands to the sensor and converting the response from bytes to the desired units is fairly representative of working with digital sensors. Therefore, this exercise will focus of finding the most essential information for working with the sensor. The datasheet, DS18B20.pdf, can be found in the lab packet or downloaded from datasheets.maximintegrated.com/en/ds/DS18B20.pdf.
9. **Questions**: To get started, let's find some basic information about the sensor. Please refer to the first page of the datasheet to answer the following.
    a. Over what temperature range is the sensor accurate to ±0.5 degrees Celsius?
    b. What is the power supply voltage range?
    c. Each DS18B20 has a unique address (or serial code). How many bits are in the address? How many bytes are in the address (1 byte = 8 bits)?
    d. What is the maximum resolution, in bits, of the temperature sensor? How many bytes is that (round up to nearest integer)?
    e. How much time, in milliseconds, does it take the sensor to output a 12 bit temperature measurements (max)?
10. **Questions**: Next, we will move to the section of the datasheet titled "Operation – Measuring Temperature" to determine how to interpret the sensor's digital signal. Read the first paragraph of this section and answer the following questions. Note: we are not using "parasite" power.
    a. According to the datasheet, if the sensor is configured with a 12 bit resolution, what is the corresponding increment, in degrees Celsius?
        i. It may be unclear what is meant by "increment". Basically, if the integer value of the digital output is x, then the temperature is x*(increment). This is essentially the same as the "sensitivity" or "scale factor", except that we are scaling a digital rather than analog signal. In this tutorial, we will use the term raw temperature to refer to the integer valued sensor output.
    b. Given the sensor's increment, what is the temperature, in degrees Celsius, for each of the following digital outputs (i.e. convert raw temperature to actual temperature)? Note: the datasheet uses the notation "h" (e.g. 00h) for hexadecimal (HEX). In Arduino, the syntax is "0x" (e.g. 0x00).
        i. Decimal: 0, Hex: 0x00, Binary: 0000 0000 0000 0000
        ii. Decimal: 1, Hex: 0x01, Binary: 0000 0000 0000 0001
        iii. Decimal: 10, Hex: 0x0a, Binary: 0000 0000 0000 1010

c. What is the command (in hexadecimal) to tell the sensor to take a measurement and perform the Analog-to-Digital conversion?
    i. Rather than immediately returning its measurement, the sensor will store the 2 byte raw temperature reading in its local "scratchpad" memory.

11. **Questions**: Finally, we will move to the section of the datasheet titled "Memory" to determine how to retrieve the raw temperature measurement from the "scratchpad".
    a. What is the command (in hexadecimal) to read from the sensor's "scratchpad" memory?
    b. How many bytes are in the scratchpad?
    c. Which bytes contain the 2 byte raw temperature reading?

Exercise 7:

Deliverables (1 pt):

- Submit a file named **MultipleDS18B20.ino** containing the program described below. Additionally, in your lab report, copy & paste your Arduino code into a code box.
- In your lab report, respond to **Questions**.

Directions:

1. In this exercise, we will write/assemble an Arduino program that can retrieve temperature readings from multiple DS18B20 sensors connected to a single digital pin using the OneWire library (though we only have enough sensors to provide one to each student).
2. To begin, open the **MultipleDS18B20_skeleton** program and save it as **MultipleDS18B20**.
3. At the top of the program, edit **DS18B20_Pin** to match the digital pin that your sensors are connected to and edit **DS18B20_Num** to match the number of sensors. Note that we are also creating an array, **temperatures**, whose length is equal to the number of sensors. We will use this array throughout the code to store the latest temperature readings.
4. For this program, the setup() and loop() functions are pretty simple. In setup(), we initiate the serial port. The setup of the digital pin (**DS18B20_Pin**) is done by the OneWire object, **ds**. In loop(), we call the function **updateTemperatures()** and include a half second delay. For now, that is all we need.
5. Let's turn our attention to the **updateTemperatures()** function. To begin, create 2 byte arrays called **data** (for storing the 9 bytes of the scratchpad) and **addr** (for storing the 8 bytes of the sensor address).

```
// Create arrays to store the temperature reading
// and the 64 bit sensor address.
byte data[9];
byte addr[8];
```

6. Next, reset the search for sensors. The OneWire object keeps track of which sensor it heard from last and the reset_search() method clears that record.

```
// Reset search for sensors
ds.reset_search();
```

7. For **steps 7-16**, the code will go **inside** the for loop. This will allow us to repeat the procedure for each sensor attached to the Arduino. In the loop, we will begin by running three checks to verify that a DS18B20 sensor is connected to the pin. If a check fails, we will set the temperature reading to -1000 and continue to the next sensor (skipping the rest of the for loop).

```
// First, search for a sensor and if found, store the address.
if ( !ds.search(addr)) {
  // No more sensors on chain
  temperatures[k] = -1000;
  continue;
}
// Next, check that the address is valid.
if ( OneWire::crc8( addr, 7) != addr[7]) {
  Serial.println("CRC is not valid!");
  temperatures[k] = -1000;
  continue;
}
// Lastly, check that the address is recognizable.
// For DS18B20 sensors, the address starts with 28.
if ( addr[0] != 0x28) {
  Serial.print("Device family is not recognized");
  temperatures[k] = -1000;
  continue;
}
```

8. In the above code snippet, the method search() tries to find a DS18B20 sensor. If it succeeds, the 64-bit address of the sensor is stored in the **addr** array and the method returns 1. If no sensor is found, the method returns 0. If we make it through all three checks, we know that the **addr** array now contains the address of a sensor attached to the Arduino. Let's print the 64-bit address to the serial monitor using a simple for loop. We will print each byte in hexadecimal.

```
Serial.print("Addr: ");
for(int i=0;i<8;i++){
  Serial.print(addr[i], HEX);
  if( i < 7 ){
    Serial.print(".");
  }
}
Serial.println();
```

9. **Question**:
   a. What is the 64-bit address of your sensor(s), in hexidecimals?
10. Next, we will "select" the sensor using its address and send the "Convert T" command, 0x44. Though the sensor only requires 750ms to collect a reading, we will wait an entire second.

```
// Now that we have an address, send a command.
ds.reset();
// Select the sensor by address.
ds.select(addr);
// Send the "Convert T" command (0x44).
ds.write(0x44);
// Give the sensor a second to store the reading.
delay(1000);
```

11. Since it has been a whole second since we last communicated with the sensor, we will check that a sensor is still present before reading from the scratchpad (though this does not guarantee that a specific sensor is connected).

```
// Read the measurements from the sensor's scratchpad.
byte present = ds.reset();
if( present ){

  // Add code from steps 12 to 16 here

}
else{
  // Something broke or shorted. No device responded.
  temperatures[k] = -1000;
  continue;
}
```

12. For **steps 12-16**, the code will go **inside** the if statement. To read from the scratchpad, we first need to select the sensor and send the "Read Scratchpad" command, 0xBE.

```
  // Select the sensor by address.
  ds.select(addr);
  // Send "Read Scratchpad" command (0xBE).
  ds.write(0xBE);
```

13. Next, use the read() method to receive data. Of the 9 bytes we receive, only the first two contain data we are interested in. The first byte contains the last 8 bits of the raw temperature and the second byte, the first 8 bits.

```
  // Read the data (9 bytes)
  for (int i = 0; i < 9; i++) {
    data[i] = ds.read();
  }
  // Of the 9 bytes, only the first 2 contain the temperature.
  byte temperature_LSB = data[0]; // Least Significant Byte
  byte temperature_MSB = data[1]; // Most Significant Byte
  // byte t_H_register = data[2];
  // byte t_L_register = data[3];
  // byte config_register = data[4];
```

```
// byte crc = data[8];
```

14. The raw temperature measurement has been received as a 2 byte "two's complement" number. Next, we need to convert these 2 bytes into a signed integer. This is done using a "bitwise" operator. The bitshift left (<<) operator moves the bits left a specified number of bits. For example, if we bitshift left the binary number 0001 by 2 (0001 << 2), we get 0100. If we bitshift left the hexidecimal number 0x01 by 8 (0x01 << 8), we get 0x0100. Once we have bit-shifted the "most significant byte", we simply add the "least significant byte" to produce the raw temperature reading. We will also print the raw temperature to the serial port.

```
// Convert 2 byte (16 bit) two's complement number to (signed) integer.
// Note: Actual resolution of temperature reading is 12 bits.
int rawTemperature = (int(temperature_MSB << 8) + temperature_LSB);
Serial.print("Raw: ");
Serial.println(rawTemperature);
```

15. **Question**:
    a. For the raw temperature, what range of values do you observe?
16. Finally, convert the raw temperature to degrees Celsius and store the value in the **temperatures** array.

```
// Convert raw temperature to degrees Celsius. For 12 bit readings,
// raw temperature increments correspond to 0.0625 degrees Celsius.
float finalTemperature = 0.0625 * rawTemperature;
temperatures[k] = finalTemperature;
```

17. We have now completed the **updateTemperatures()** function and we are almost done putting together the entire program. There are just a couple things to add. First, we will add two functions allowing the temperatures to be updated in Celsius or Fahrenheit. Place the following functions **below/after updateTemperatures()**.

```
// Get the temperature from multiple DS18B20 sensors in degrees Celsius
void updateTemperaturesC(){
  updateTemperatures();
}

// Get the temperature from multiple DS18B20 sensors in degrees Fahrenheit
void updateTemperaturesF(){
  updateTemperatures();
  for (int i = 0; i < DS18B20_Num; i++) {
    if( temperatures[i] != -1000 ){
      temperatures[i] = temperatures[i]*9/5 + 32;
    }
  }
}
```

Second, edit the loop() function to update the temperature readings in Fahrenheit and to print them to the serial port.

```
void loop() {
  // Update all the temperature readings
  Serial.println("Requesting temperatures...");
  updateTemperaturesF();
  for (int i = 0; i < DS18B20_Num; i++) {
    Serial.print("Device ");
    Serial.print(i);
    Serial.print(" Temp (degF): ");
    Serial.println(temperatures[i]);
  }
  // Slow down the output to make it readable
  delay(500);
}
```

18. **Question**:
    a. What range of temperatures do you observe, in degrees Fahrenheit?
    b. What are some advantages/disadvantages of the digital DS18B20 sensor compared to the analog TMP36 sensor we used previously?

Exercise 8:

Deliverables (0.5 pts):

- Submit a file named **DallasMultipleDS18B20.ino** containing the program described below. Additionally, in your lab report, copy & paste your Arduino code into a code box.
- In your lab report, respond to **Questions**.

Directions:

1. As you may have noticed, getting a reading from a digital device can be a lot more work than using an analog sensor. However, digital sensors often provide better precision, customizable features (such as alarms and adjustable precision), and in many cases, multiple sensors on a single device. Additionally, the Arduino community is well supported by engineers and programmers. This means that for many popular sensors, there are open source libraries that make it easier to work with the device.
   This is the case for DS18B20 sensor. In this exercise, we will leverage the DallasTemperatureControlLibrary to improve and simplify our code by utilizing the library's built-in methods.
   To begin, we need the DallasTemperatureControl library for Arduino. The library can be found in the Lab 2 Packet (->Arduino Libraries) or downloaded from milesburton.com/Dallas_Temperature_Control_Library. You will need to place the

"DallasTemperatureControlLibrary" folder into your Arduino Library folder and then restart the Arduino IDE.

2. Open the **DallasMultipleDS18B20_skeleton** program and save it as **DallasMultipleDS18B20**.
3. Just as before, the program starts by defining several variables. Note that we have added four variables, **DS18B20_Max_Num**, **DS18B20_Found_Num**, **sensors**, and **addresses**. In this program, we will automatically detect (at power) the number of sensors connected to the Arduino. **DS18B20_Max_Num** sets the limit on the number of sensors the program can handle and **DS18B20_Found_Num** stores the number of sensors that have been found. The **sensors** object contains all the methods of the DallasTemperature class. The **addresses** variable is an array of DeviceAddress objects where a DeviceAddress object is simply an 8 byte array.
4. In this program, we want to store the 64 bit address of each sensor. Because we only want to perform the task once, we will place the code from steps 4 and 5 in the setup() function. Just after Serial.begin(), we need to start the DallasTemperature object. Also, we will call the getDeviceCount() method to check for the number of sensors connected to the Arduino. Upload the program and open the Serial Monitor to view the output.

```
// Start up the library
sensors.begin();

// Locate devices on the bus
Serial.print("Locating devices...");
int found = sensors.getDeviceCount();
Serial.print("Found ");
Serial.print(found, DEC);
Serial.println(" device(s).");
// Apply limit
DS18B20_Found_Num = min(found, DS18B20_Max_Num);
```

5. Next, we will store each address in the **addresses** array. The getAddress() method takes 2 inputs, a DeviceAddress object in which to store the address (if found) and the index of the sensor to find. If the method fails to find an address, it returns false (0). Upload the program and open the Serial Monitor to view the output.

```
// Store and print device addresses
for(int i=0;i<DS18B20_Found_Num;i++){
  if (!sensors.getAddress(addresses[i], i)){
    Serial.print("Unable to find address for Device ");
    Serial.println(i);
  }else{
    Serial.print("Device ");
    Serial.print(i);
    Serial.print(" Address: ");
    printAddress(addresses[i]);
    Serial.println();
  }
}
```

6. Next, we will move to the loop() function. In each loop, we will send the "Convert T" request to each sensor, read the raw temperature from the scratchpad memory, convert the temperature to degrees Fahrenheit, and store the value in the temperatures array. Using the DallasTemperature object, all we need to do is call the requestTemperatures() method and include a for loop to get the temperature from each sensor.

```
// Call sensors.requestTemperatures() to issue a global temperature
// request to all devices on the bus. Method includes 750ms delay.
Serial.println("Requesting temperatures...");
sensors.requestTemperatures(); // Send the command to get temperatures

// Update all the temperature readings
for (int i=0;i<DS18B20_Found_Num;i++) {
  // Read sensors in degrees Fahrenheit
  temperatures[i] = sensors.getTempF(addresses[i]);
}
```

7. Finally, print all the information to the serial port. Upload the program and open the Serial Monitor to view the output.

```
// Print to serial port
for (int i=0;i<DS18B20_Found_Num;i++) {
  Serial.print("Device ");
  Serial.print(i);
  Serial.print(" Temp (degF): ");
  Serial.println(temperatures[i]);
}
```

8. **Questions**:
   a. What are the advantages to using open-source libraries, like DallasTemperatureControlLibrary?
   b. What are potential problems/disadvantages?