IAT 884
Workshop Week 5
Micro-controllers Part 2: Inputs (Getting Data from Sensors)

Preparation:

Read in *Physical Computing*:
- Digital Input (P. 80 – 84)
- Analog Input (93 - 97)
- From Analog in to Analog Out: Scaling Function (p.116 - 118)
- Programming Interactivity: Chapter 8: Physical Input (P. 245 – 255, 265 - 270)

Other Resources:

Physical Computing:
Touch Me (p. 249- 254)

http://www.arduino.cc/en/Tutorial/Button (Please note the use of a pull-down resistor)

http://arduino.cc/en/Reference/Map (Mapping Function to Scale a range of numbers)

Arduino Inputs:
http://todbot.com/blog/wp-content/uploads/2006/10/arduino_spooky_projects_class2.pdf

Edge Detection/Debounce:
http://www.arduino.cc/en/Tutorial/Debounce

Play Melody:
http://www.arduino.cc/en/Tutorial/PlayMelody

IAT 884
Workshop Week 5
Micro-controllers Part 2: Inputs (Getting Data from Sensors)

In Class Exercise:

Materials:

- Piezo Buzzer (Included in electronics kit)
- Light Dependant Resistor (LDR)
- 2 x 10uF Capacitors
- 1K Ohm Resistor
- Breadboard
- Wires
- Arduino
- USB Cable

i. Create a circuit that uses a pushbutton to activate a piezo buzzer. You should use one of the digital inputs on the Arduino along with a pull-down resistor. The voltage sent to the buzzer should come from one of the analog outputs.

ii. Create an "instrument" that can be played using an LDR (Light Dependant Resistor). The frequency of the playback tone needs to be controlled by the amount of light hitting the LDR. The LDR must be connected to an analog input pin on the Arduino and use a pull-up resistor. You must also use the **play_tone** function provided below to generate the sound. This requires using digital output. The mapping between light/frequency is up to you. You will likely need to use the serial monitor to see what kind of input you are getting each sensor and use the map() function to adjust it.

iii. Add at least one decoupling capacitor to your circuit. (10 uF)

Arduino Code for playing a tone on a piezo puzzer
Based on http://www.arduino.cc/en/Tutorial/PlayMelody

```
/* Play Tone
 * -----------
 *
 * Program to play a Tone
 *
 * Tones are created by quickly pulsing a speaker on and off
 *   using PWM, to create signature frequencies.
 *
 * Each note has a frequency, created by varying the period of
 *  vibration, measured in microseconds. We'll use pulse-width
 *  modulation (PWM) to create that vibration.

 * We calculate the pulse-width to be half the period; we pulse
 *  the speaker HIGH for 'pulse-width' microseconds, then LOW
 *  for 'pulse-width' microseconds.
 *  This pulsing creates a vibration of the desired frequency.

 * Modifications to code by Aaron Levisohn
 * Based on original code by
 * (cleft) 2005 D. Cuartielles for K3
 * Refactoring and comments 2006 clay.shirky@nyu.edu
 */
/*

//TONES  ==========================================
// Start by defining the relationship between
//       note, period, &  frequency.
 * c     3830    // 261 Hz
 * d     3400    // 294 Hz
 * e     3038    // 329 Hz
 * f     2864    // 349 Hz
 * g     2550    // 392 Hz
 * a     2272    // 440 Hz
 * b     2028    // 493 Hz
 * C     1912    // 523 Hz
 */
```

```
// SETUP ================================================
// Set up speaker on a PWM pin (digital 9)

 int speakerOut = 9;

 void setup() {
              pinMode(speakerOut, OUTPUT);
}

// PLAY TONE  ===========================================
// Pulse the speaker to play a tone for a particular duration

void playTone(long duration, int tone) {
             long elapsed_time = 0;
              while (elapsed_time < duration) {

                           digitalWrite(speakerOut,HIGH);
                            delayMicroseconds(tone / 2);

                           // DOWN
                            digitalWrite(speakerOut, LOW);
                            delayMicroseconds(tone / 2);

                           // Keep track of how long we pulsed
                            elapsed_time += (tone);
              }
}

// LET THE WILD RUMPUS BEGIN ============================
void loop() {
              playTone(10000, 3830); //playTone(duration, tone)
}
```

Arduino Map Funtion:

Re-maps a number from one range to another. That is, a **value** of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

Example:

```
y = map(x, 0, 50, 0, 1024);
```

Maps an input number (x) that lies between 0 and 255 to a proportional value between 0 and 1024.

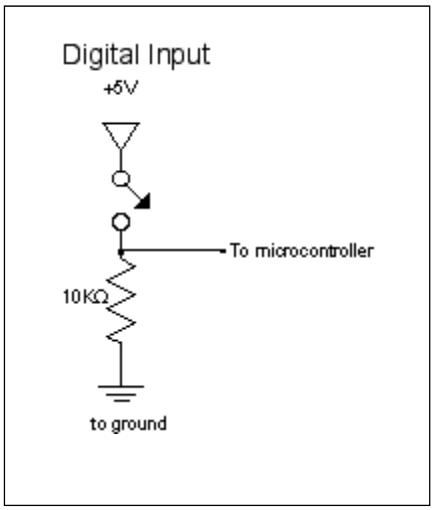
Example Code:

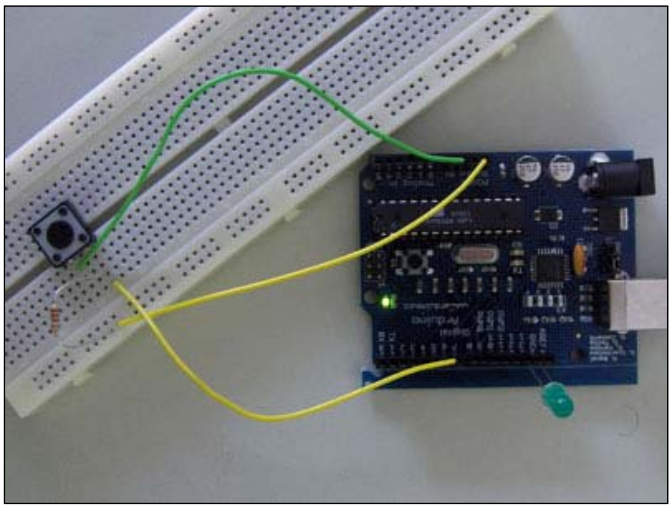
```
void loop()
{
  int val = analogRead(0);
  val = map(val, 0, 255, 0, 1024);
  analogWrite(9, val);
}
```

http://www.arduino.cc/en/Reference/Map

Pull-down Resistor

A pull down resistors is used to ensure that there is a full voltage drop to 0v when a switch is flipped. This is essential for the functioning of a switch connected to one of the digital input pins on the Arduino. A resistor connected to the ground is a **pull-down** resistor. A resistor connected to Vcc is **pull-up** resistor.
Pull-down resistors also prevents short circuits between the ground and the input pin on the microcontroller. Without this precaution the chip could be damaged.





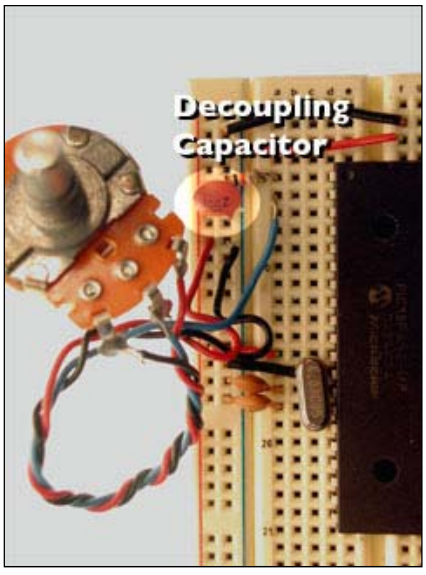
Example circuit using a push button switch:
http://www.arduino.cc/en/Tutorial/Button

Explanation of pull-up/down resistor use:
http://www.seattlerobotics.org/encoder/mar97/basics.html

Decoupling Analog Inputs

If you find the readings from your analog inputs are inconsistent (for example, you see changes on one channel when the sensor on a different channel is the one sensing action), it helps to decouple your input circuit. Decoupling means smoothing out the dips and spikes going into the circuit from the rest of your microcontroller circuit. To do this, place a 0.1microfarad capacitor from voltage to ground as close to where the analog input connects to voltageas in the figure to the right.
A capacitor used this way is referred to as a *decoupling capacitor*. You'll see them a lot in electronic circuits. Think of them as tiny surge protectors.

(From http://www.tigoe.net/pcomp/analogin.shtml)



Arduino Code Reference

Enabling a digital input Pin

```
int inPin = 2;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {
  pinMode(inPin, INPUT);    // declare Pin as input
}

void loop(){
  val = digitalRead(inPin);  // read input value
  if (val == HIGH) {         // check if the input is HIGH}
{
        //do something
}
}
```

Enabling an Analog input Pin

```
int potPin = 2;    // select the input pin for the potentiometer
int val = 0;       // variable to store the value coming from the sensor

void setup() {

        }

void loop() {
  val = analogRead(potPin);    // read the value from the sensor
}
```

Serial Data

Serial.print and Serial.println

These commands send data over the serial communication channel and are used to interface with other applications. Serial data transfers are also visible in the serial monitor in the Arduino programming environment. This makes it a useful tool for quickly viewing sensor data.

```
int b = 79;
Serial.print(b);
```

prints the ASCII string "79".

Arduino Language Reference:

http://www.arduino.cc/en/Serial/Print

http://www.arduino.cc/en/Serial/Println