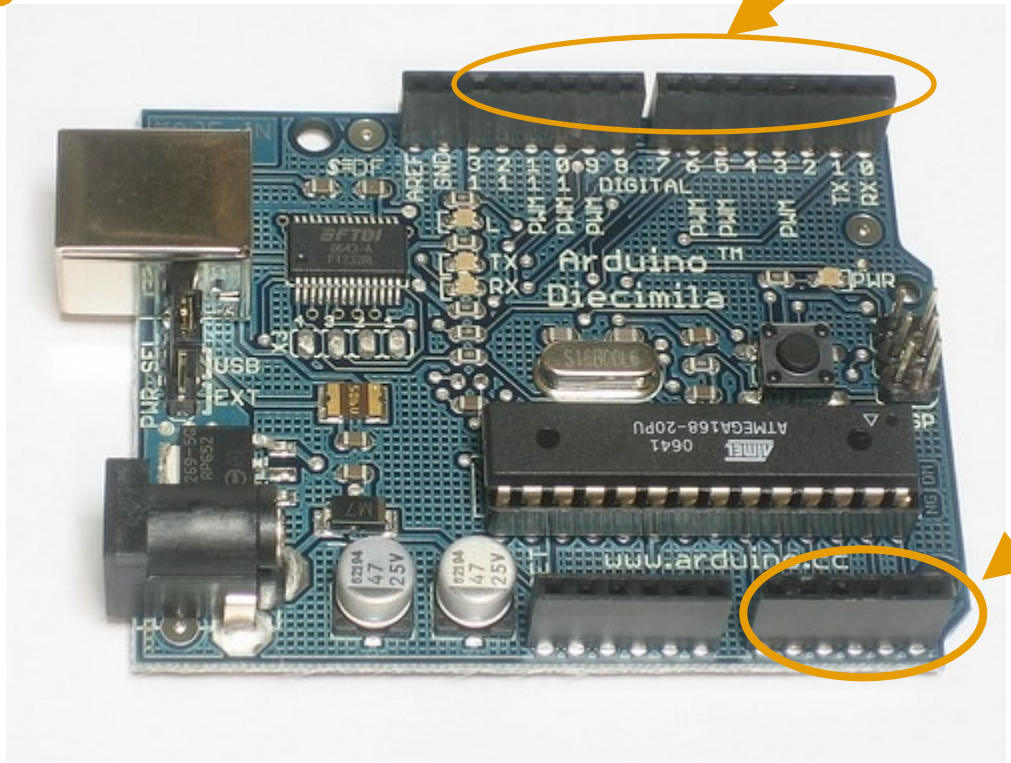
IAT 884 Workshop

Micro-Controllers: Input

The Arduino Diecimila/Duemilanove

Input Pins



Digital In

Signals are interpreted as **high** or **low.**

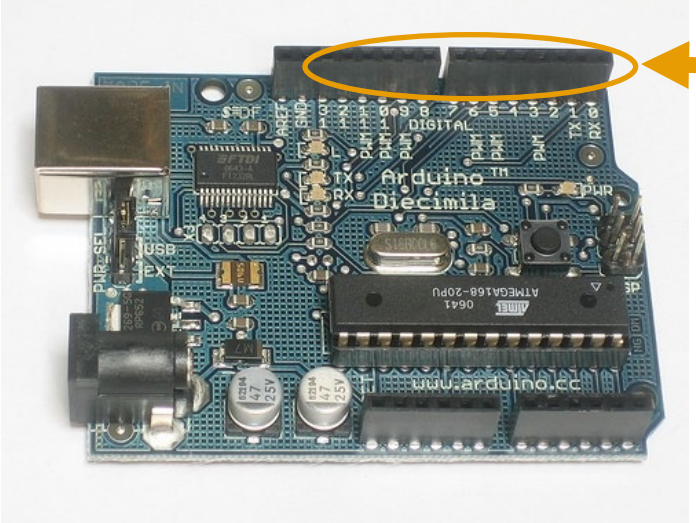
High = 5v
(Actually anything more than 3v is interpreted as HIGH.)

Low = 0v

Process:

1. Read input from one of the digital pins (0-13).

2. Detect the type of signal using an IF statement.



Digital In

```
pinMode(2, INPUT);                  //Set pin 2 as input
val = digitalRead(2);               // read the input pin
```

EXAMPLE:

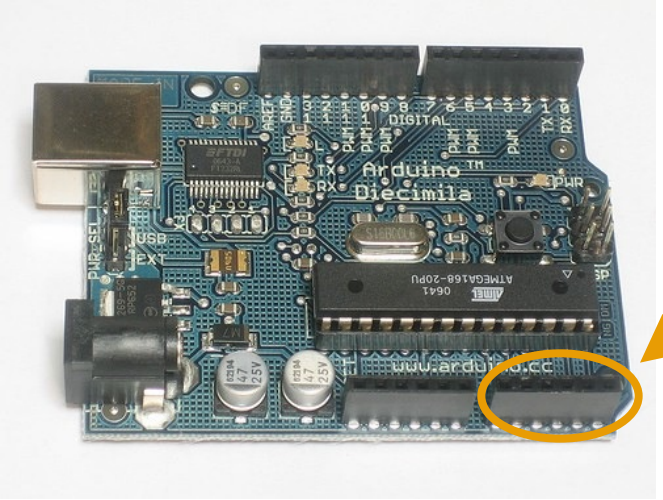
```
int inPin = 7;   // pushbutton connected to digital pin 7
int val = 0;     // variable to store the read value
void setup(){
        pinMode(inPin, INPUT);      // sets the digital pin 7 as input
        Serial.begin(9600);  //begins serial communicaton
}
void loop(){
        val = digitalRead(inPin);   // read the input pin
        Serial.println(val); //print the incoming value
         if(val==HIGH){
            Serial.println("High"); //print if the incoming sensor value is high
        }
        else{
             Serial.println("Low");  //print if the incoming sensor value is low
        }
}
```

Analog In

The Arduino will convert any analog input signal between 0-5v into a number between 0-1023. *(10 Bit ADC)*

** Remember that Analog Out send signals using 0-255*

There are 6 Analog pins and they are separate from the digital pins.



Analog
In Pins

Analog In

val = analogRead(analogPin); // read the input pin

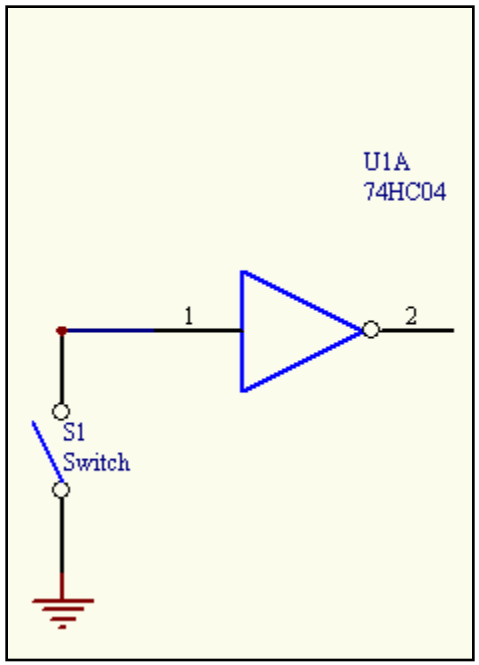
```
int analogPin = 3;
int val = 0;                          // variable to store the value read
void setup()
{
  Serial.begin(9600);                 //  setup serial
}
void loop()
{
  val = analogRead(analogPin);    // read the input pin
  Serial.println(val);                // debug value
}
```

Pull-Up Resistor

Pull-up resistors are used to ensure that the voltage is drawn up to 5V when a switch is open (a pull-down resistor works exactly opposite, pulling the signal down to 0V/ground). This prevents wandering signals.

Pull-Up Resistor

Problem: In the circuit below, closing the switch will connect the Arduino input pin with the ground pin resulting in a reading of 0v. But what about when the switch is open?

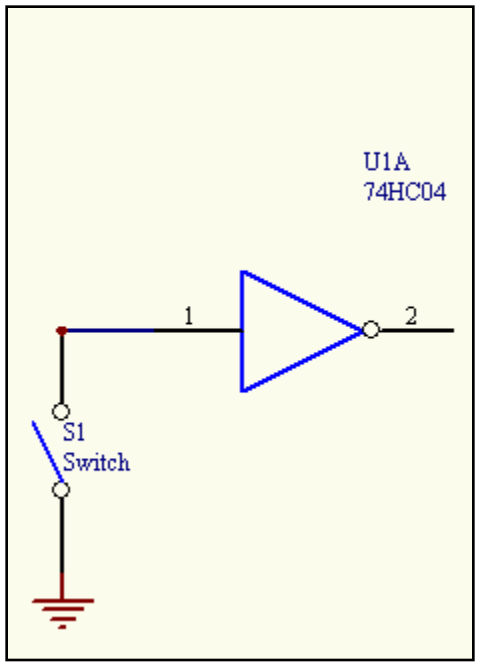


A floating input gate with an indeterminate value.

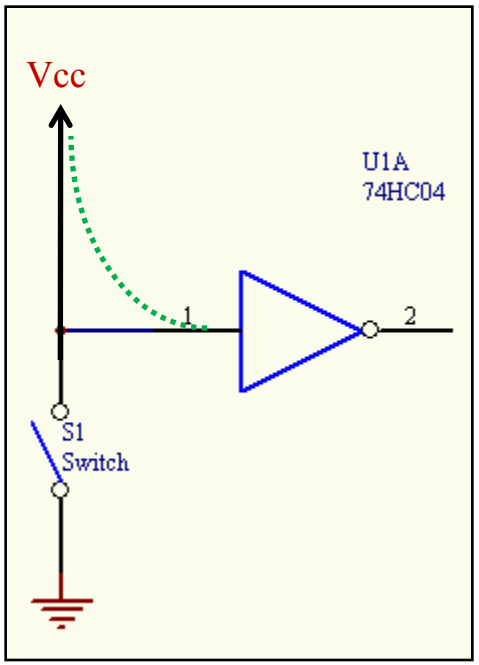
Because the input pin has no connection to Vcc (+5 volts) it will never read HIGH. Instead it will "float" producing unpredictable values.

Pull-Up Resistor

Our first impulse is to connect the input to Vcc (+5v) to solve the problem. This works fine when the switch is open….



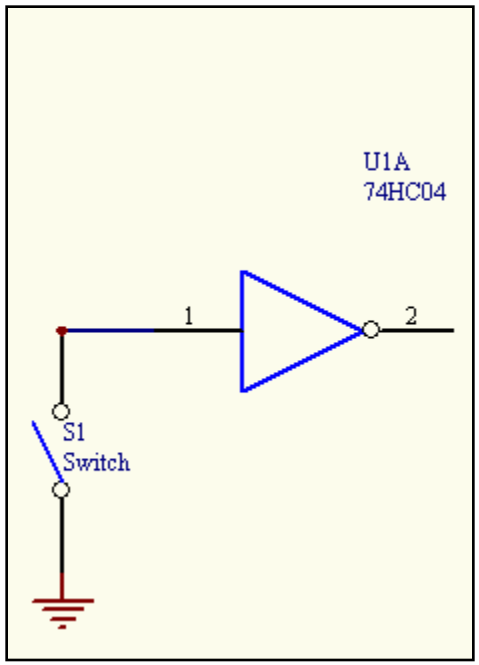
A floating input gate with an indeterminate value.



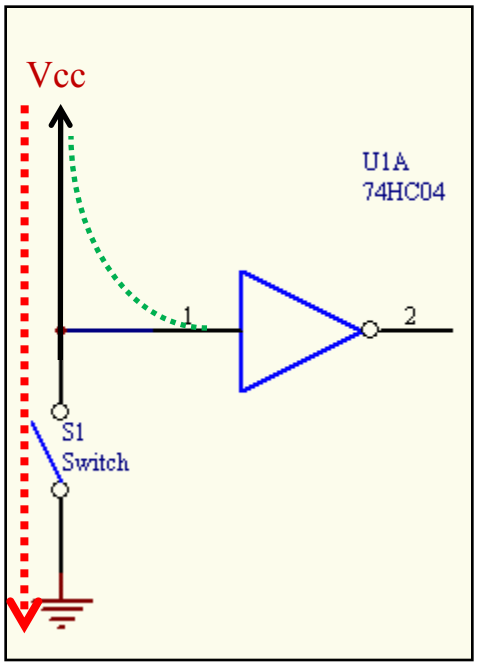
First impulse solution

Pull-Up Resistor

Our first impulse is to connect the input to Vcc (+5v) to solve the problem. This works fine when the switch is open…. But creates a short circuit when it is closed. There is now a direct connection between ground (0v) and Vcc (+5v).



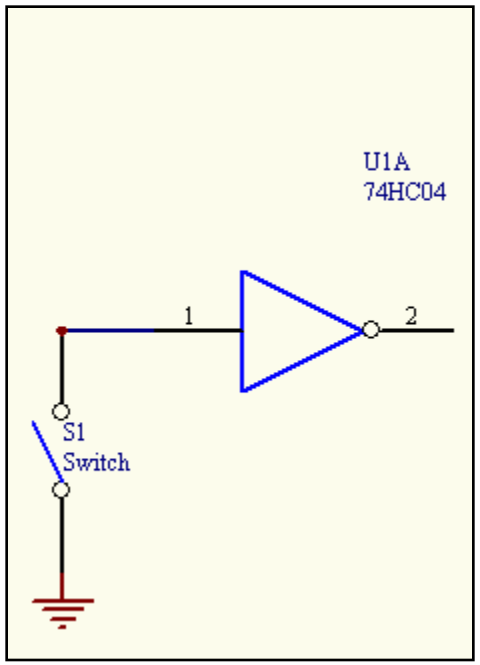
A floating input gate with an indeterminate value.



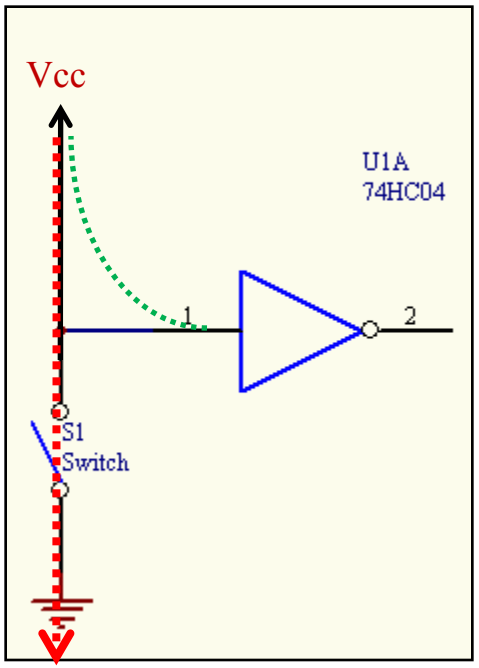
First impulse solution – Creates a short circuit.
BAD IDEA!!!!!

Pull-Up Resistor

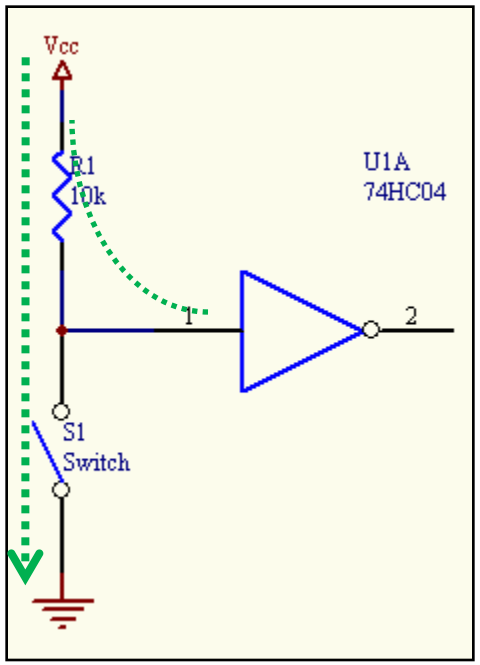
By placing a resistor between Vcc (+5) and the switch we prevent the short circuit but still allow enough voltage through to let our input pin on the Arduino read HIGH.



A floating input gate with an indeterminate value.



First impulse solution – Creates a short circuit.
BAD IDEA!!!!!



A Pull up Resistor solves the problem!

Pull-Up Resistor

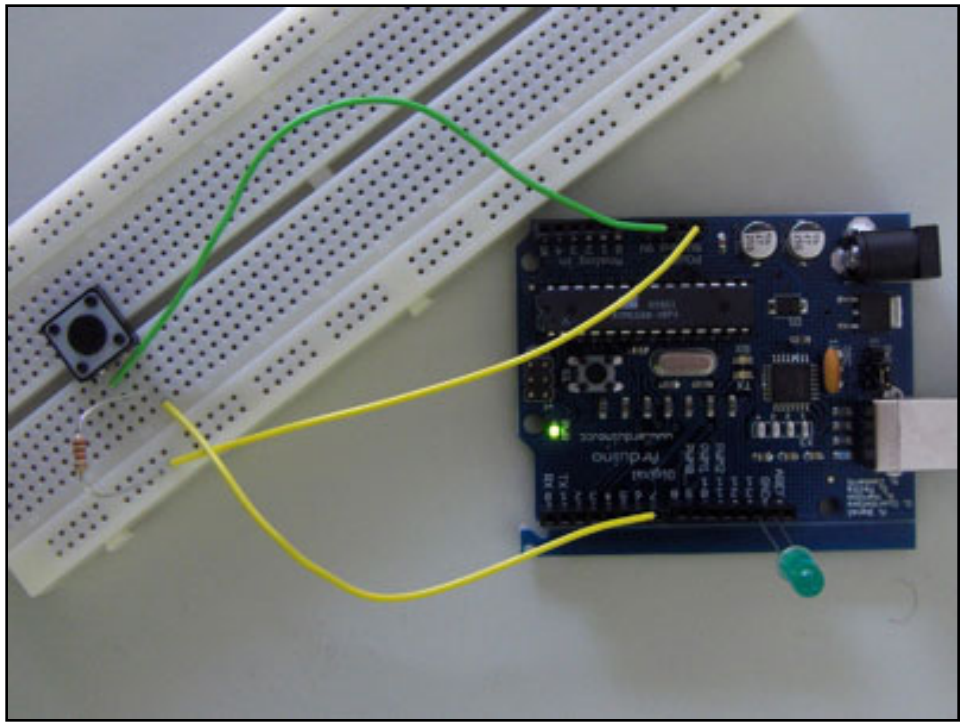
When the switch is closed, the signal is read entirely through the switch; the voltage across the relatively high-impedance resistor is essentially nothing.

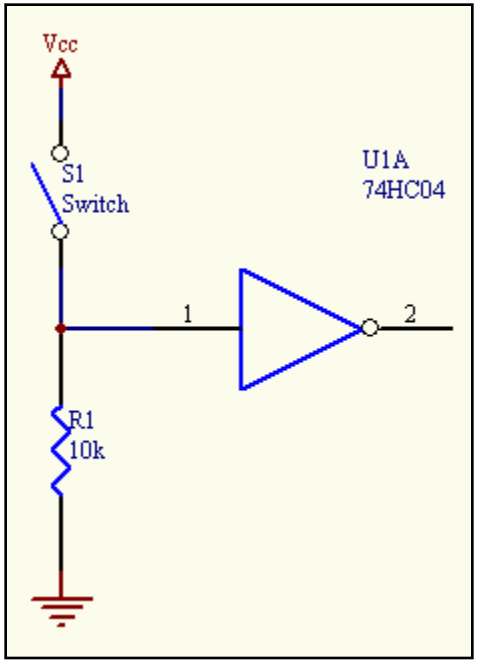
When the switch is open, the pull-up resistor is now the path of least resistance. The only signal is coming from the voltage across that resistor.

Short Video on Pull Up Resistors:
https://www.youtube.com/watch?v=wxjerCHCEMg

Pull-Down Resistor

This is an example of a Pull-Down Resistor attached to an Arduino Board.





Grounding Input Sources

Whenever you are sending signals from one circuit to another (i.e. from a sensor to the Arduino), you must make sure everything shares a COMMON GROUND.
This means that the **negative wires from all components must be connected.**

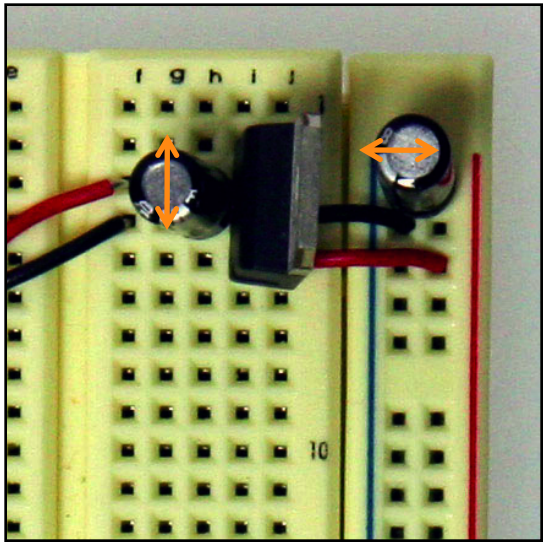
If you are using a separate power supply that provides 12v (remember that the Arduino runs on 5v) you still need to connect the negative wires together. This creates the common ground.

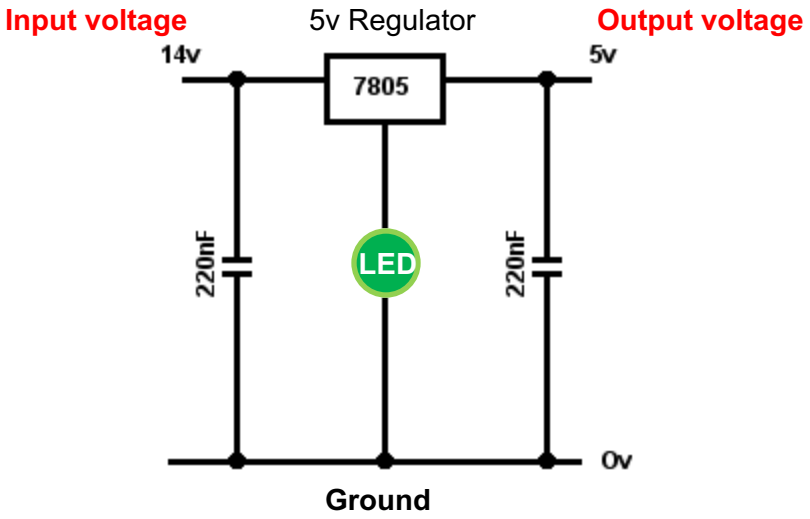
All INPUTS, OUTPUTS, and SOURCES must share a common ground (most of the time).

Decoupling Capacitors

Decoupling capacitors are low strength capacitors placed between voltage and ground. Decoupling capacitors smooth signals and can greatly improve readings on sensors.

Decoupling capacitors are generally between 0.1 uF and 22 uF. (Micro Farads)





Scaling Functions

A common task you will need to perform is the conversion of an input value into a range that is usable. For example, you receive an analog input value between 0-1023 and you want to send out a proportional analog signal to some device (0-255). Scaling functions help us do this.

OUTPUT VALUE = ((InValue - minInValue) * outRange / InRange) + minOut

- InValue is the current analog reading
- minIn = lowest possible input value
- maxIn = highest possible input value
- maxOut = highest popssible output value
- outRange = maxOut –minOut
- InRange = maxIn-minIn

Scaling Functions

The Arduino Library has a built in scaling function called map()

map(value, fromLow, fromHigh, toLow, toHigh)

Example:

```
mapped_value = map(initial_value, 0, 50, 0, 1024);
```

Will convert a number between 0 – 50 to a number between 0 -1024

if initial_value = 0 then mapped_value = 0

if initial_value = 50 then mapped_value = 1024

if initial_value = 25 then mapped_value = 512

Edge Detection

Edge Detection is a method that allows you to use a pushbutton as a switch.

It uses input signal change (e.g. from LOW to HIGH) as the trigger for some action.

After receiving a signal change the system compares the current signal with the previous signal. If the signals are different then we know the button state has changed.

Using Edge Detection you can count the number of times a button has been pressed.

Edge Detection

```
int switchPin = 2;
int switchCounter = 0;
int switchState = 0;
int lastSwitchState = LOW;

void setup() {
        pinMode(switchPin, INPUT);
        Serial.begin(9600);
}

void loop() {
        switchState = digitalRead(switchPin); // read the switch
        if (switchState != lastSwitchState) {// compare the switch to its previous state
            if (switchState == HIGH) {
                switchCounter++; // if the state has changed, increment the counter
                Serial.println(switchCounter); //print number of times button pressed

                }
        lastSwitchState = switchState; // save the current state as the last state,
                                       //for next time through the loop
        }

}
```

Debouncing

Debouncing is similar to Edge Detection. Debouncing an input removes faulty readings due to the mechanical properties of the button.

When a button is pushed, even though it is in one state, there can be faulty readings where the button's spring "bounces" quickly back and forth between on and off.

To prevent extra state changes from being counted we make sure that the button has been in a state a specific amount of time before registering the button press.

Debouncing

```
int inPin = 7;         // the number of the input pin
int outPin = 13;       // the number of the output pin
int state = HIGH;      // the current state of the output pin
int reading;           // the current reading from the input pin
int previous = LOW;    // the previous reading from the input pinvoid setup()

// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was toggled
long debounce = 200;   // the debounce time, increase if the output flickers

void setup() {
  pinMode(inPin, INPUT);
  pinMode(outPin, OUTPUT);
}

void loop()
{
  reading = digitalRead(inPin);

  // if we just pressed the button (i.e. the input went from LOW to HIGH),
  // and we've waited long enough since the last press to ignore any noise...
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
    // ... invert the output
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;
    // ... and remember when the last button press was
    time = millis();
  }

  digitalWrite(outPin, state);
  previous = reading;
}
```

Debouncing

```
int inPin = 7;         // the number of the input pin
int outPin = 13;       // the number of the output pin
int state = HIGH;      // the current state of the output pin
int reading;           // the current reading from the input pin
int previous = LOW;    // the previous reading from the input pinvoid setup()

// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was toggled
long debounce = 200;   // the debounce time, increase if the output flickers
```

Debouncing

```
void loop()
{
  reading = digitalRead(inPin);

  // if we just pressed the button (i.e. the input went from LOW to HIGH),
  // and we've waited long enough since the last press to ignore any noise...
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
    // ... invert the output
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;
    // ... and remember when the last button press was
    time = millis();
  }

  digitalWrite(outPin, state);
  previous = reading;
}
```

Debouncing

There is a library that you can use to make debouncing easier. However I suggest writing your own code first to better understand what is going on under the hood.

The Bounce Library can be found here:

http://www.arduino.cc/playground/Code/Bounce

Activities

In Class Exercise:

1. Create a circuit that uses a switch or pushbutton to activate a piezo buzzer. You should use one of the digital inputs on the Arduino along with a pull-up resistor. See below for code to generate tones.
2. Create an “instrument” that can be played using an LDR (Light Dependant Resistor). The frequency of the playback tone should 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. 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.
3. Add at least one decoupling capacitor to your circuit. (10 uF or 22uF)
4. OPTIONAL: Redo Exercise 1 but have each button press play a sound for a specific length of time. You will need to “debounce” your switch to accomplish this.

Pull-Up Resistors Revisited

The Arduino baord has built in Pull-Up Resistors on all digital pins.

To activate them you simply declare a pin as an INPUT pin:

pinMode(inPin, INPUT);

but also add the following statement in setup():

digitalWrite(inPin, HIGH) ;

//inPin is the number of the physical pin you are using on the board

And you still read the pin the same way:

val = digitalRead(inPin);

Using this command you no longer have to make a connection with the 5v pin at all. You just connect one side of your switch to ground, and the other to the digital input pin you are using.