Programming "Arduino" Sketches

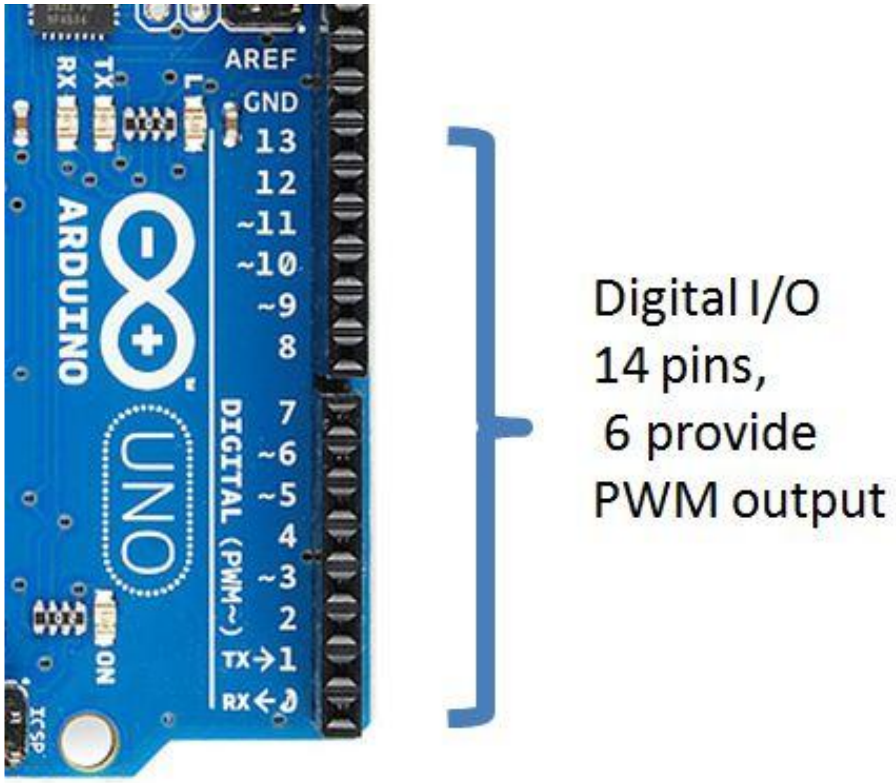
Digital Inputs/Outputs

Instructor / Facilitator - Alan Rux

"Platform"

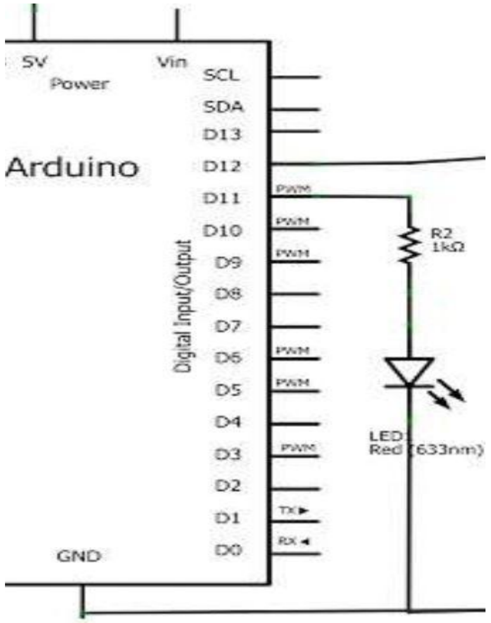
Digital I /O Pins

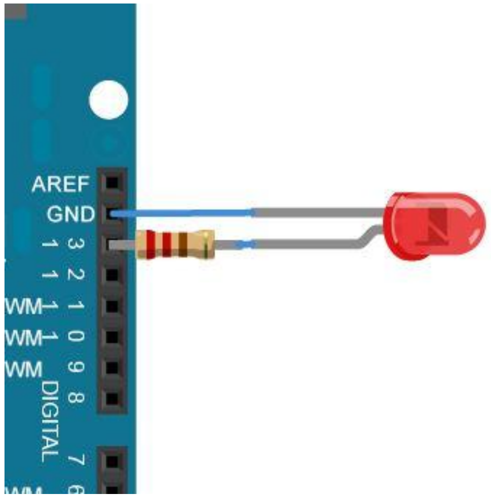
- **D0 to D13** = 14 pins
- "O" = O v., "1" = +5v.
- Six pins = **PWM** output
  D3,D5,D6,D9,D10,D11
- I/O pin 0 = RX, I/O pin=TX
  (serial com, port)
- I/O current limited
  (see next slide)



Digital I /O Pins **output current limits**

- **40 ma. Max current / pin**
- **200 ma. Max for IC (all pins total)**
- **< 10ma. Recommended**
- Use 560 ohm resistor to 1k ohm resistor for current limiter
- Output **Z** very low ohms





Digital I /O Pins **Inputs**

- When a pin is configured as an INPUT with pinMode, and read with digitalRead, the microcontroller will report **HIGH if a voltage of 3 volts or more** is present at the pin
- The microcontroller will report **LOW if a voltage of 2 volts or less** is present at the pin
- Arduino pins configured as **INPUT** with pinMode() are said to be in a **high-impedance state**. Pins configured as INPUT make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 Megohms in front of the pin. This makes them useful for reading a sensor, but not powering an LED.
- Often it is useful to steer an **input pin to a known state if no input is present**. This can be done by adding a pullup resistor (to +5V), or a pulldown resistor (resistor to ground) on the input. **A 10K resistor is a good value for a pullup or pulldown resistor**

Digital I /O Pins **Outputs**

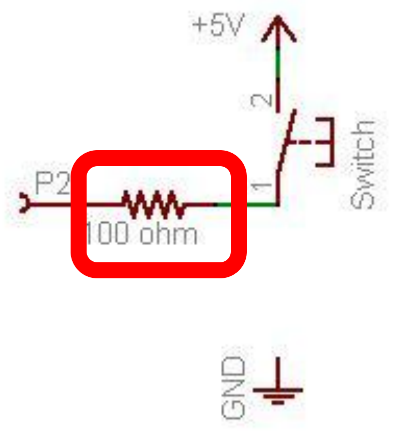
- Pins configured as OUTPUT with pinMode() are said to be in a **low-impedance state**. This means that they can provide a substantial amount of current to other circuits. Atmega pins can **source** (provide positive current) or **sink** (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (don't forget the series resistor), or run many sensors, for example, but not enough current to run most relays, solenoids, or motors

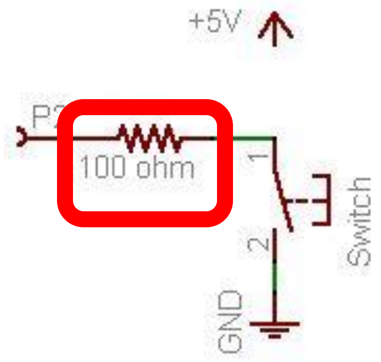
- **Short circuits on Arduino pins, or attempting to run high current devices from them, can damage or destroy the output transistors in the pin, or damage the entire Atmega chip. Often this will result in a "dead" pin in the microcontroller but the remaining chip will still function adequately.**

- **For this reason it is a good idea to connect OUTPUT pins to other devices with 470Ω to 1k series resistor.**

Digital Inputs - Pull Up / Down

- Simple circuit Push Button
- Note 100 ohm resistor, this prevents you from burning out the I/O pin, **if by mistake,** it is programmed as an output and connected to +5 volts or ground, 0 volts. **The 100Ω resistor acts as a buffer, to protect the pin from short circuits**

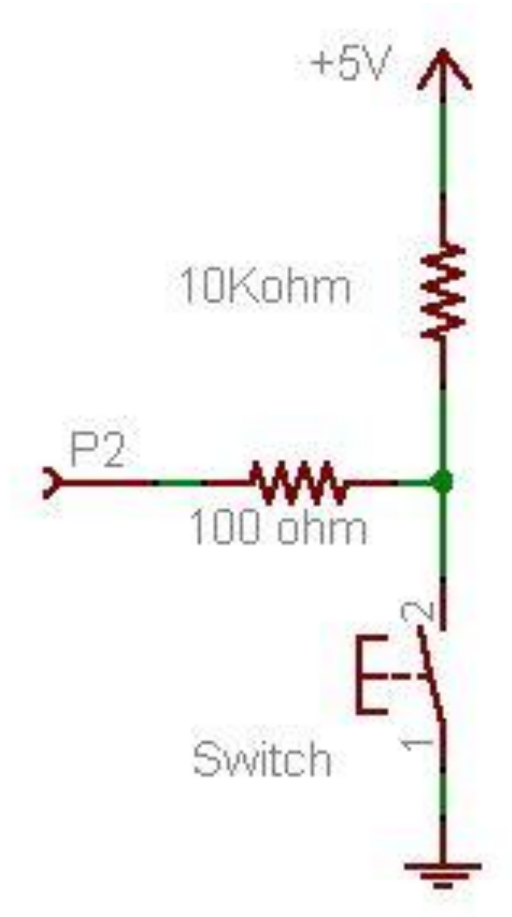




- When circuit is in open position the input I/O pin can float and give unreliable readings, High or Low (see next slide)

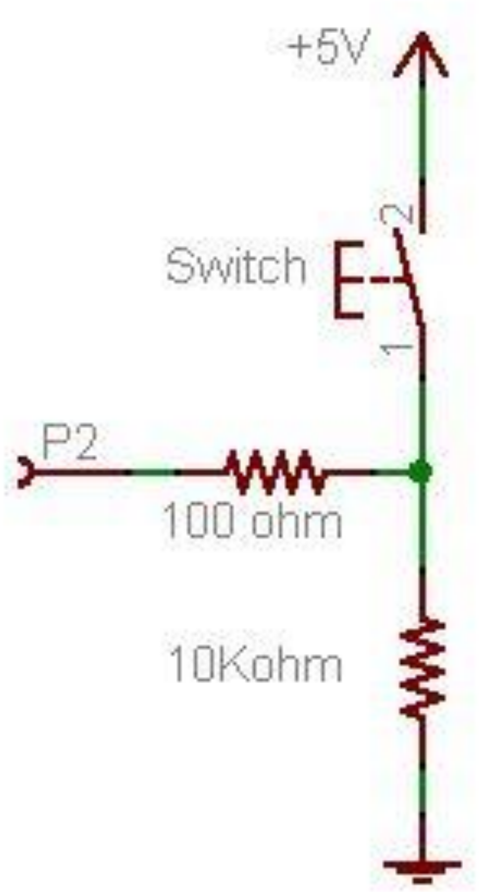
Digital Inputs - Pull-Up Circuit

- By adding a 10 K ohm resistor the input pin is in the **high** input signal position until the **Push Button** is pressed and shorts the input to ground, a digital low signal
- This is called an "**active low**" input



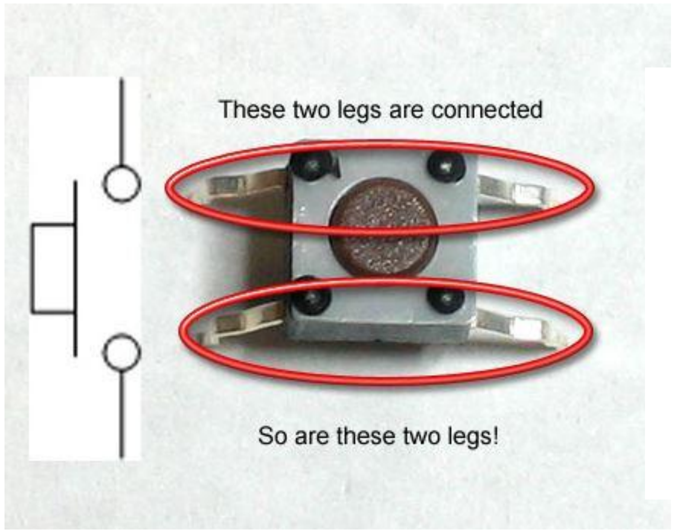
Digital Inputs - Pull-Down Circuit

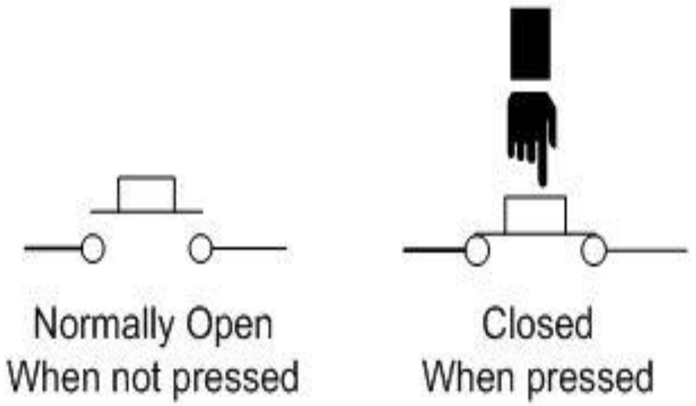
- By adding a 10 K ohm resistor the input pin is in the **low** input signal position until the **Push Button** is pressed and connects the input to + 5 volts, a digital high signal
- This is called an "**active high**" input



Pushbutton Switches

- Pushbutton switches are two-position devices actuated with a button that is pressed and released. Most pushbutton switches have an internal spring mechanism returning the button to its "out," or "unpressed," position, for momentary operation.

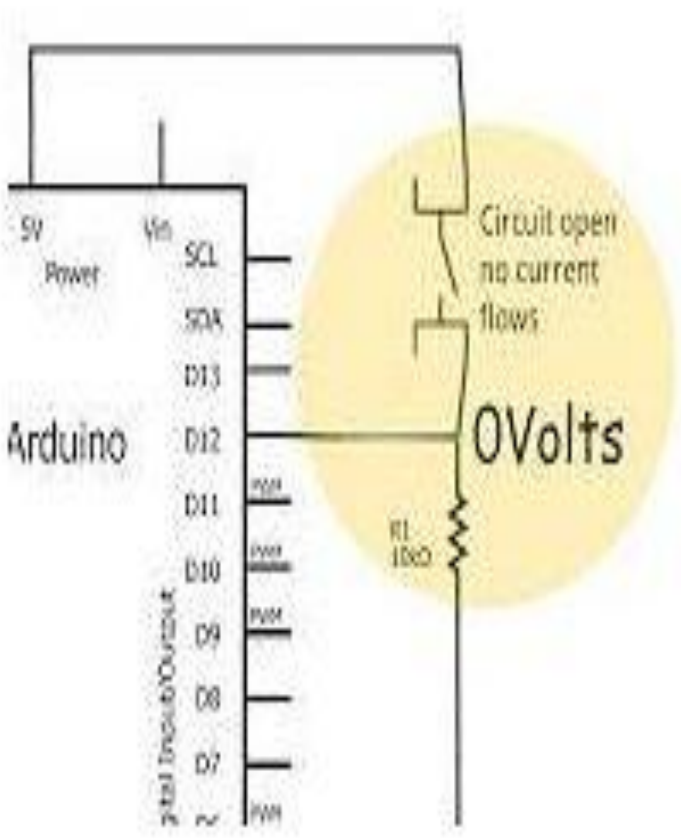


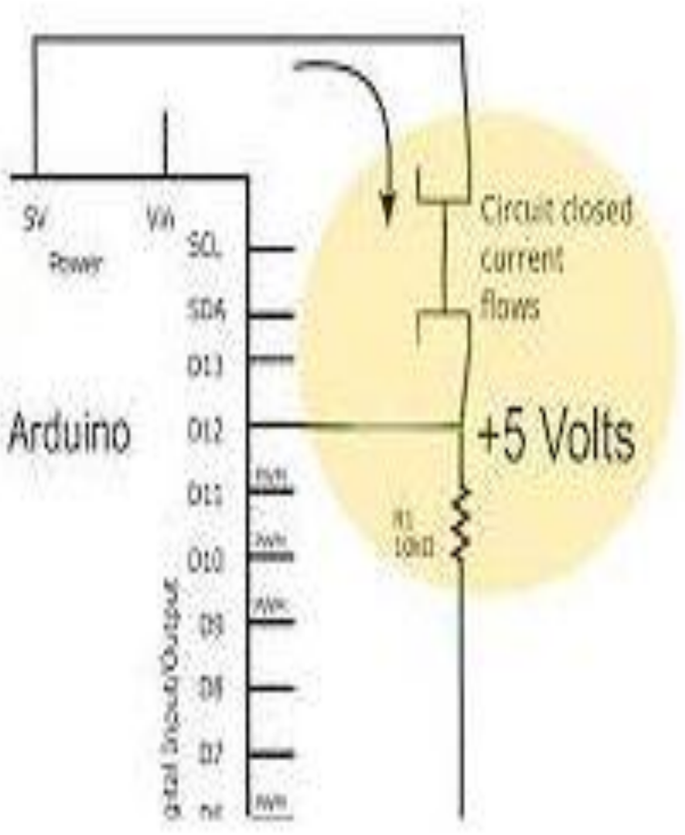


Normally **O**pen **P**ush**B**utton switch

Pushbutton Switch Digital I/O Input Circuit

- **Active** when **High** digital input





Gnd, 0 volts

Gnd, 0 volts

Push Button test program (1 of 2 pages)

(red is comments statement)

```
// Pushbutton program reports when a button is pushed and released

// Constant integer used to set pin numbers >
const int buttonPin = 12; // the number of the pushbutton pin

// variables
//(Variables may change while the program runs)
int buttonState = 0; // variable the pushbutton status

void setup() {
  // initialize Serial communications
  Serial.begin(9600);

  // set the buttonPin mode to INPUT
  pinMode(buttonPin, INPUT);
}
```

Push Button test program (2of 2 pages)

(red is comments statement)

```
void loop(){

  // get the state of the pushbutton
  buttonState = digitalRead(buttonPin);

  // is the button pressed?
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
   // Tell the world
   Serial.println("Button pushed.");
  }
  else {
    Serial.println("Button not pushed.");
  }

  delay(500); // pause for 1/2 a second
}
```

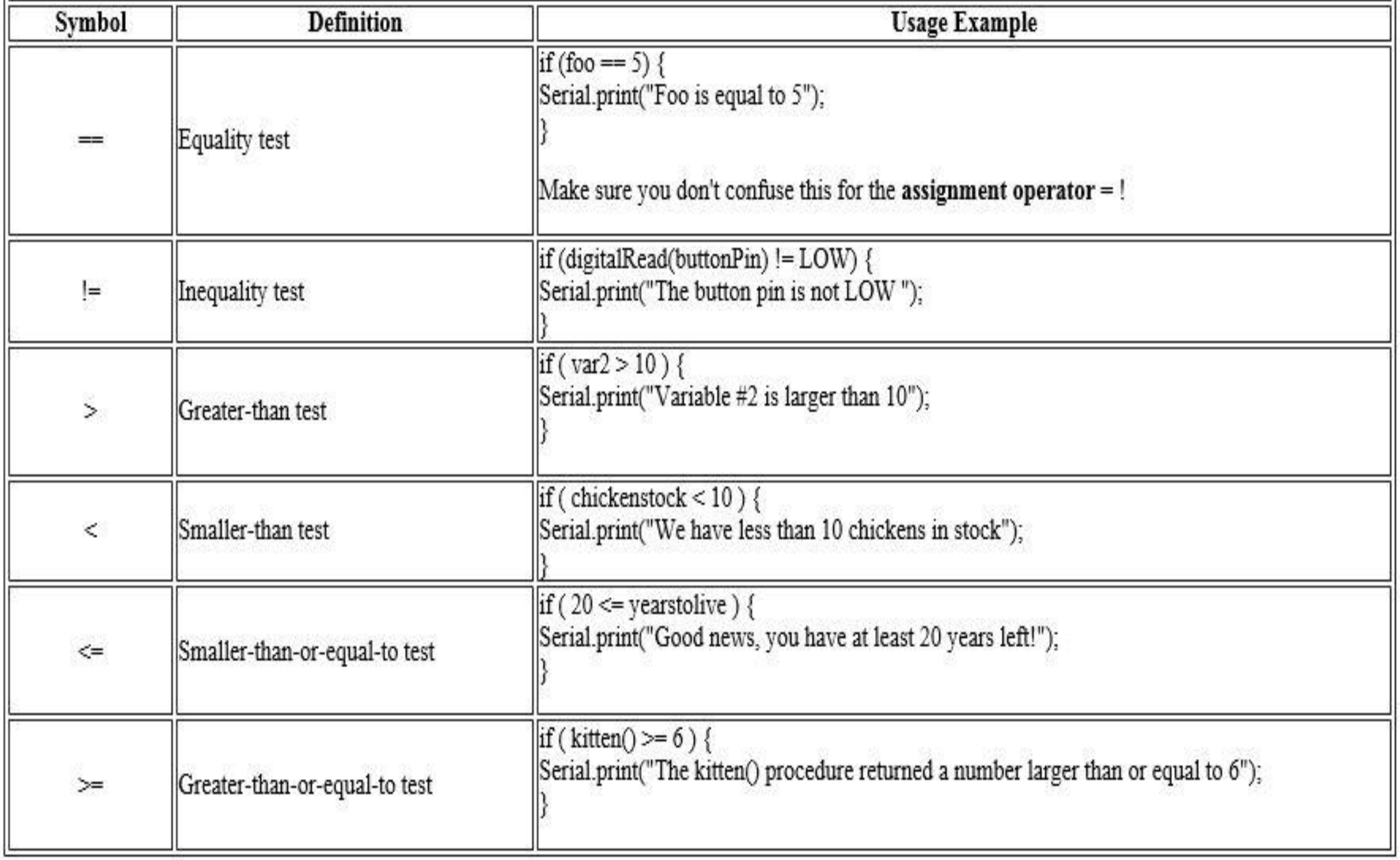
if (buttonState == HIGH)

if, which is used in conjunction with a comparison operator, tests whether a certain condition has been reached, such as an input being above a certain number. The format for an if test is:

- if (someVariable ==HIGH) { // do something here }
- **==** , use the double equal sign

  (e.g. if (buttonState == HIGH)), which is the comparison operator, and tests *whether* xbuttonState is equal to digital HIGH or not.

some comparison operators



Symbol	Definition	Usage Example
==	Equality test	if (foo == 5) { Serial.print("Foo is equal to 5"); } Make sure you don't confuse this for the **assignment operator =** !
!=	Inequality test	if (digitalRead(buttonPin) != LOW) { Serial.print("The button pin is not LOW "); }
>	Greater-than test	if (var2 > 10) { Serial.print("Variable #2 is larger than 10"); }
<	Smaller-than test	if (chickenstock < 10) { Serial.print("We have less than 10 chickens in stock"); }
<=	Smaller-than-or-equal-to test	if (20 <= yearstolive) { Serial.print("Good news, you have at least 20 years left!"); }
>=	Greater-than-or-equal-to test	if (kitten() >= 6) { Serial.print("The kitten() procedure returned a number larger than or equal to 6"); }

Serial.println("Button pushed.");

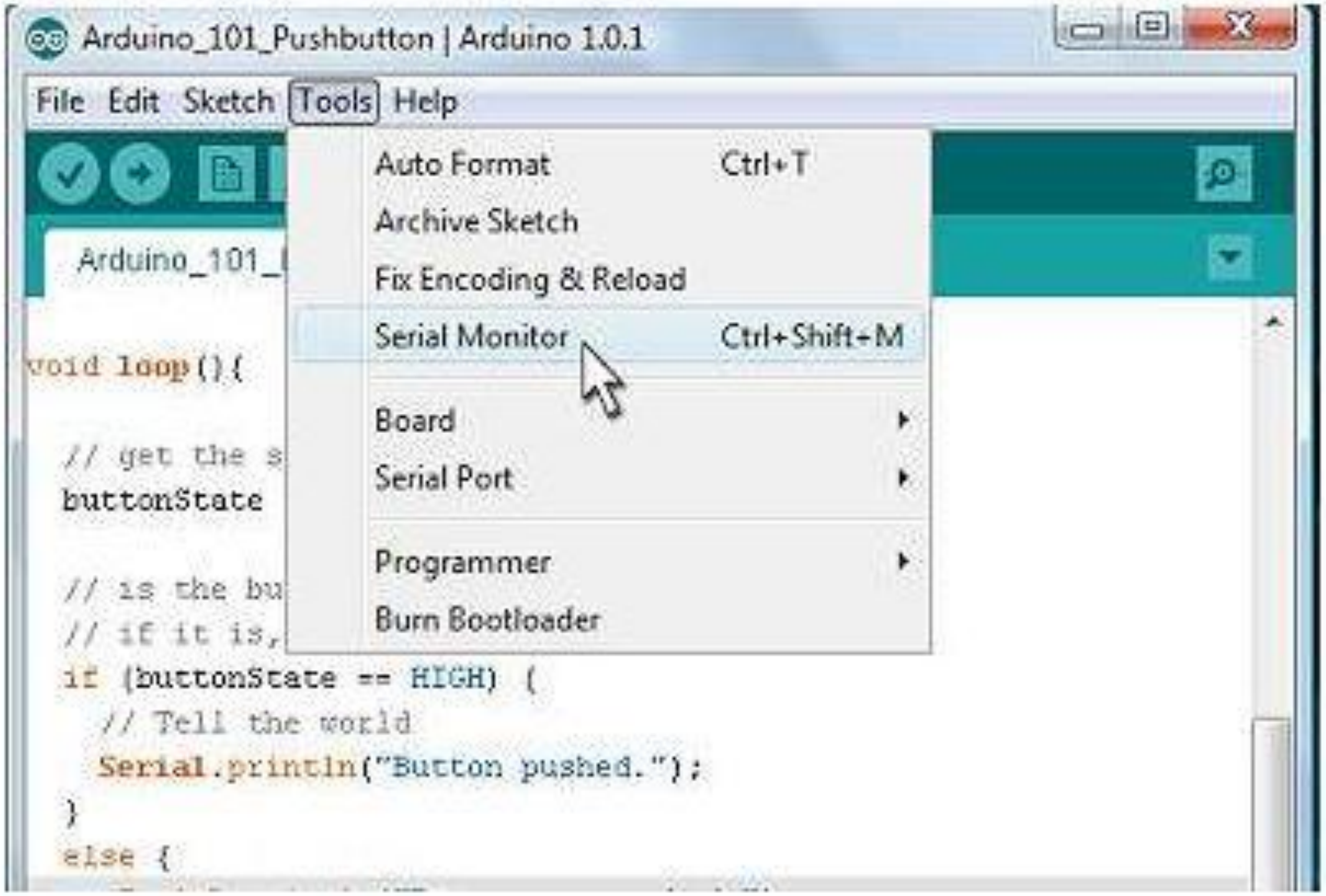
- You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().
- **println**(val) Prints data to the serial port as human-readable ASCII text followed by a carriage return character
- println**(val)** val: the value to print - any data type . ("Button pushed.")

```
else {
    Serial.println("Button not pushed.");
```

- **Else** **if/else** allows greater control over the flow of code than the basic **if** statement, by allowing multiple tests to be grouped together
- **In the case that the input is not HIGH** the else comparison statement prints “Button not pushed” on the serial IDE monitor.
- We have a 500 ms delay and loop back to **if** again, repeat over and over.

Push Button test program

We will use the serial monitor on the Arduino IDE



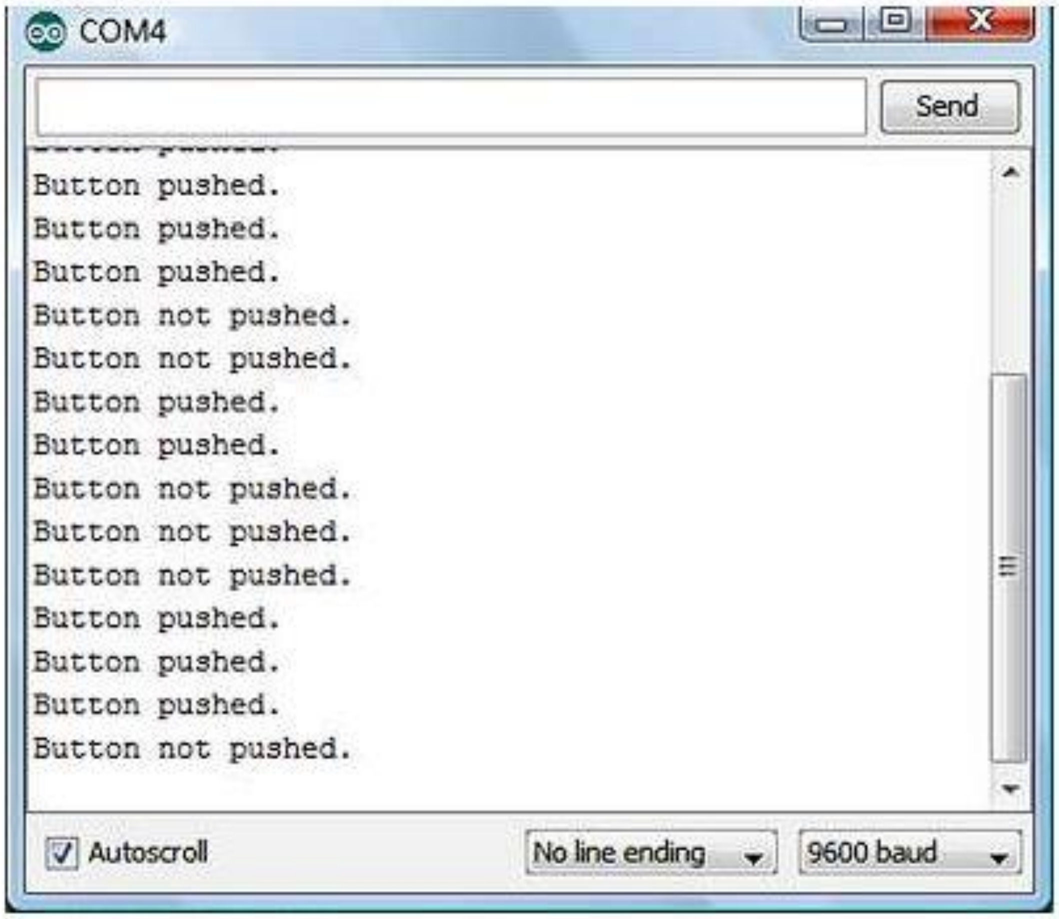
```
void loop(){

  // get the s
  buttonState

  // is the bu
  // if it is,
  if (buttonState == HIGH) {
    // Tell the world
    Serial.println("Button pushed.");
  }
  else {
```

Push Button test program

We will use the serial monitor on the Arduino IDE to show status of input pin

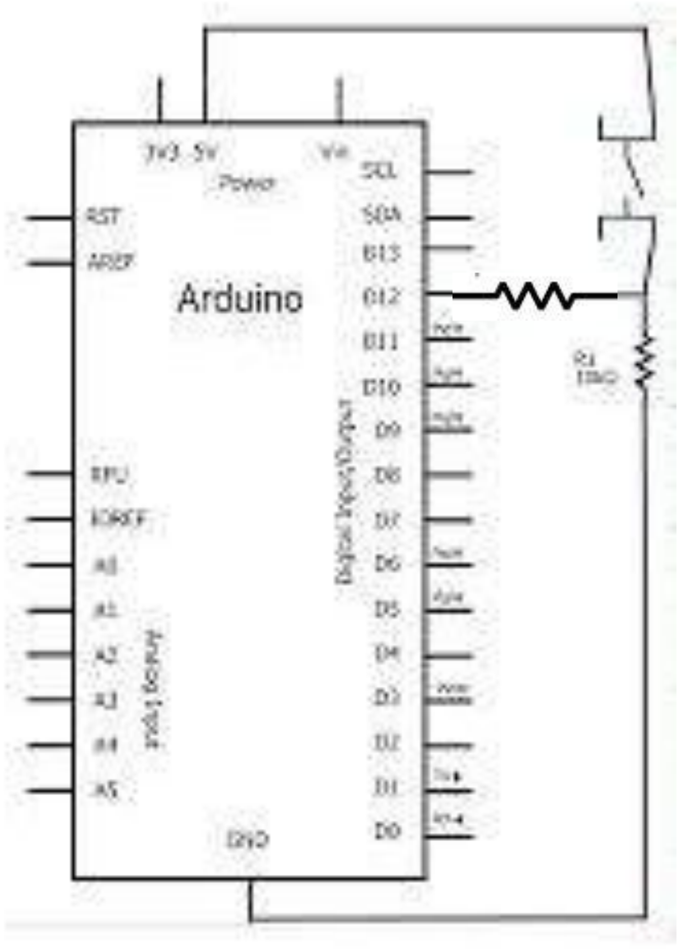


```
Button pushed.
Button pushed.
Button pushed.
Button not pushed.
Button not pushed.
Button pushed.
Button pushed.
Button not pushed.
Button not pushed.
Button not pushed.
Button pushed.
Button pushed.
Button pushed.
Button not pushed.
```

[✓] Autoscroll   No line ending   9600 baud

Push Button Schematic

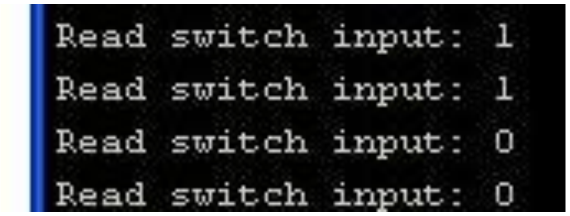
- I/O pin 12 is used as a digital input
- Add 220 ohm resistor between pin 12 and 10k ohm pull down resistor and push button switch
- Read switch status as pressing switch,



Another Push Button test program

(red is comments statement)

```
/*
    * Switch test program
    */
int switchPin =12;    // Switch connected to digital pin 12
void setup()   // run once, when the sketch starts
{
      Serial.begin(9600);   // set up Serial library at 9600 bps
     pinMode(switchPin, INPUT);   // sets the digital pin as input to read switch
}
 void loop()   // run over and over again
{
 Serial.print ("Read switch input: ");
Serial.println(digitalRead(switchPin));   // Read the pin and display the value
delay(100);
}
```



```
Read switch input: 1
Read switch input: 1
Read switch input: 0
Read switch input: 0
```

combine inputs (buttons) and outputs (LEDs)

Sketch

Verify that when the button is pressed, the LED turns on and when the button is released, the LED turns off. (1 of 2 pages)

```
/*
    * Switch and LED test program
*/
int ledPin = 10;   // LED is connected to pin 11
int switchPin = 12;   // switch is connected to pin 12
int val;   // variable for reading the pin status
 void setup() {
pinMode(ledPin, OUTPUT);   // Set the LED pin as output
pinMode(switchPin, INPUT);   // Set the switch pin as input
}
```

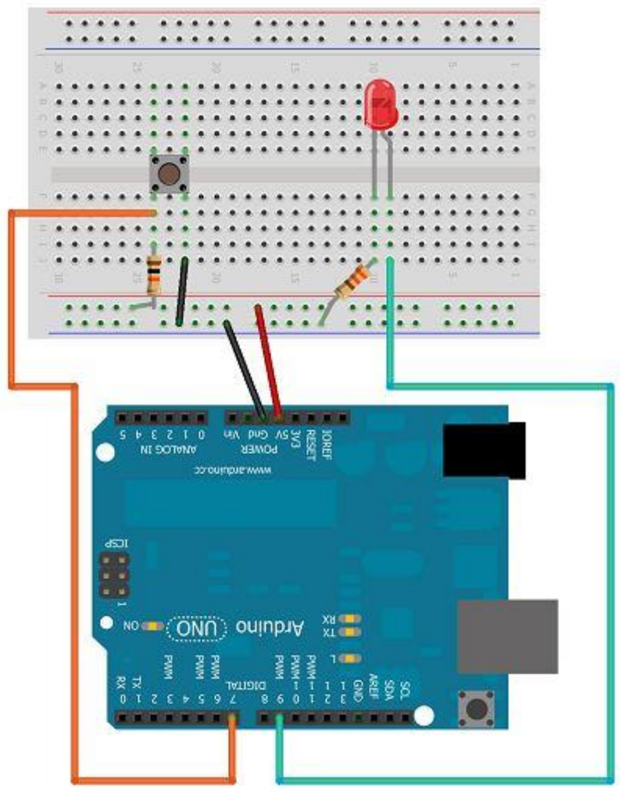
combine inputs (buttons) and outputs (LEDs)

Sketch

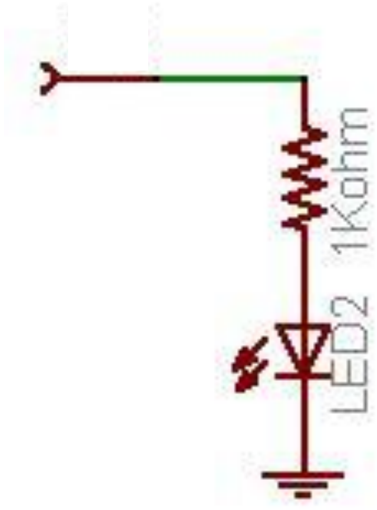
Verify that when the button is pressed, the LED turns on and when the button is released, the LED turns off. (2 of 2 pages)

```
void loop(){
    val = digitalRead(switchPin);  // read input value and store it in val
    if (val == LOW) {   // check if the button is pressed
    digitalWrite(ledPin, HIGH);   // turn LED on
    }
    if (val == HIGH) {   // check if the button is not pressed
    digitalWrite(ledPin, LOW);   // turn LED off
    }
}
```

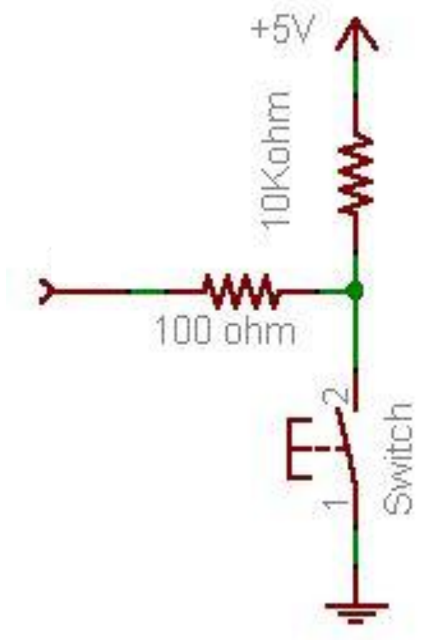
Schematic



LED is connected to pin 11



switch is connected to pin 12

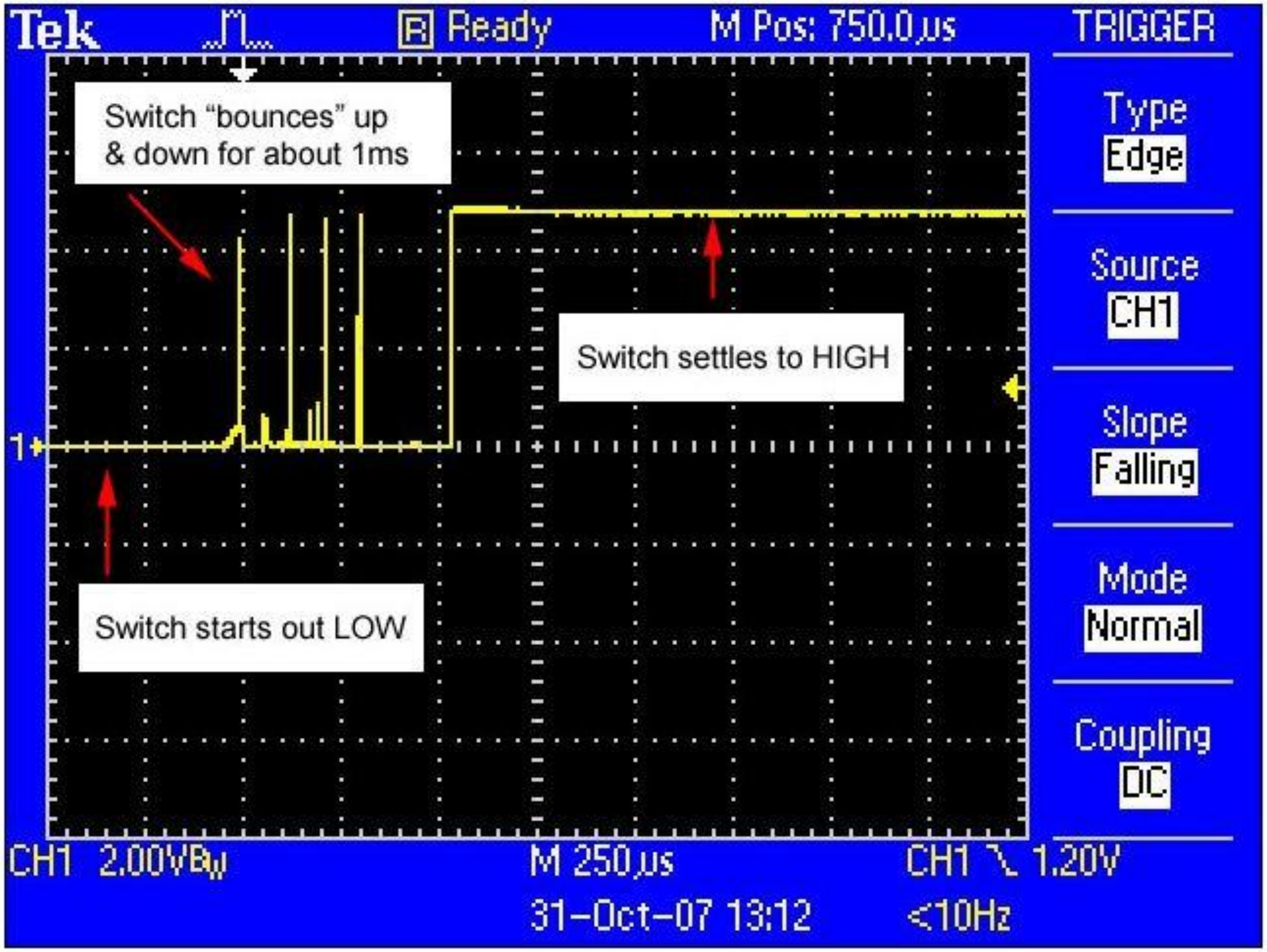


Switch Contact Bounce

- You spend some time looking over your code but can't seem to find the problem.
- Turns out this is not a **software (sketch) problem,** but actually a **mechanical problem**.
- Inside the little tactile switch is a small disc spring. When you push the button you squeeze the spring so that it makes contact with the two wire connections. When you release, the spring bounces back. This works great except that, the spring is *springy*. And that means that once in a while, when you press the button it **bounces** around a little in the switch, making and breaking contact a few times before **settling**
- This "bounce" can be interpreted as many switch presses,

  causing the microcontroller to think the switch has been presses a few times and operates as if that is true causing miss interpretation and unwanted results

Switch Contact Bounce scope trace

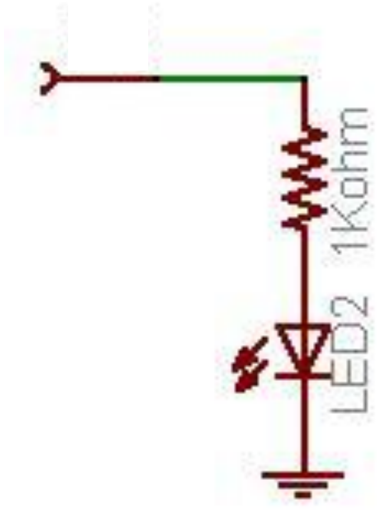


Switch Debounce

- This example demonstrates how to **debounce** an input, which means checking twice in a short period of time to make sure it's definitely pressed. Without debouncing, pressing the button once can appear to the code as multiple on/off inputs.
- Using the same schematic as in the *“combine inputs (buttons) and outputs (LEDs) Sketch”* we will do a LED toggle on or off with switch pushbutton pressing.
- First without de-bounce then with de-bounce.

Schematic

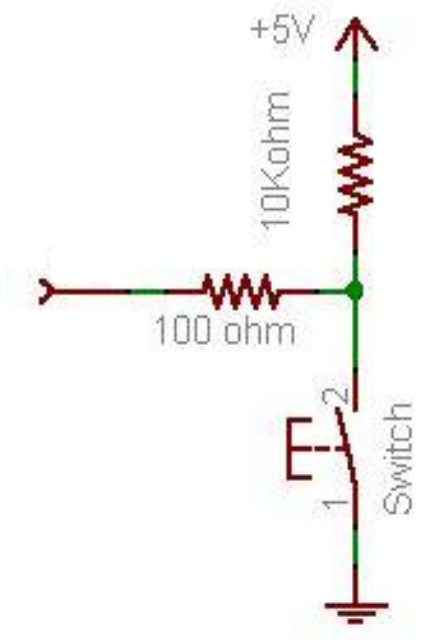
LED is connected to pin 11



Try the Toggle without the debounce first,

Then add the delay(500) function to debounce the switch.

switch is connected to pin 12



LED Toggle Sketch (no debounce)

```
// Led toggle
int inputPin = 12;  // push button switch pin 12
int ledPin = 11;  // Led wired to pin 11
int ledValue = LOW

void setup ()
 {
     pinMode (inputPin,  INPUT);  // set pin 12 to input
     pinMode (ledPin,  OUTPUT);  // set pin 11 to output
}
void loop ()
{
     if  (digitalRead (inputPin) == LOW)
{
     ledValue = ! ledValue;
     digitalWrite (ledPin,  ledValue);
} }
```

LED Toggle Sketch (with debounce)

```
// Led toggle
int inputPin = 12; // push button switch pin 12
int ledPin = 11; // Led wired to pin 11
int ledValue = LOW;

void setup()
{
    pinMode(inputPin, INPUT);  // set pin 12 to input
    pinMode(ledPin, OUTPUT); // set pin 11 to output
}

void loop ()
{
    if  (digitalRead (inputPin) == LOW)
{
    ledValue = ! ledValue;
    digitalWrite (ledPin,  ledValue);
     delay (500);  // 500 ms delay to get past switch bounce
} }
```

Questions

- What does the IDE Serial Monitor do?
- How do you mark comments in your sketch
- Why do you have to debounce a switch ?
- **Can you see the bounce using your Analog Discovery Kit scope? Free-run or single sweep mode?**
- Can you see the effect of the bounce on the Led ?
- Did you play around with the delay time to see the minim time required to debounce?