PHYSICAL COMPUTING 3: SENSORS

CSE 590 Ubiquitous Computing | Lecture 6 | May 3

Jon Froehlich • Liang He (TA)

DESIGN USE BUILD

MAKEABILITY LAB

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY of WASHINGTON

SCHEDULE TODAY: 6:30-9:20

06:30-06:55: Discussion of required reading led by Joe Wandyez

06:55-07:00: Optional discussion led by Abhay Rijhwani (link)

07:00-08:10: Physical Computing 3: Sensors

08:10-08:15: Break

08:15-09:20: Physical Computing 4: Android + Arduino via BLE

SCHEDULE TODAY: 6:30-9:20

06:30-06:55: Discussion of required reading led by Joe Wandyez

06:55-07:00: Optional discussion led by Abhay Rijhwani (link)

07:00-08:10: Physical Computing 3: Sensors

08:10-08:15: Break

08:15-09:20: Physical Computing 4: Android + Arduino via BLE

SCHEDULE TODAY: 6:30-9:20

06:30-06:55: Discussion of required reading led by Joe Wandyez

06:55-07:00: Optional discussion led by Abhay Rijhwani (link)

07:00-08:10: Physical Computing 3: Sensors

08:10-08:15: Break

08:15-09:20: Physical Computing 4: Android + Arduino via BLE

WORKING WITH SENSORS: LEARNING GOALS

How to use variable **resistive sensors,** including **voltage dividers,** basic **signal smoothing**

Putting it together: **making a simple theremin**

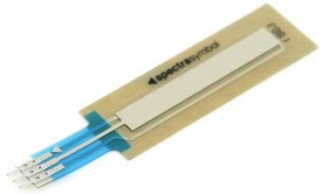
How to use **buttons**, including **pull-up** and **pull-down resistors**, and **debouncing**

Working with **interrupts**

VARIABLE RESISTORS

LOTS OF DIFFERENT KINDS OF VARIABLE RESISTORS

Potentiometer 10k; $0.95*



Touch Membrane Potentiometer; $12.95

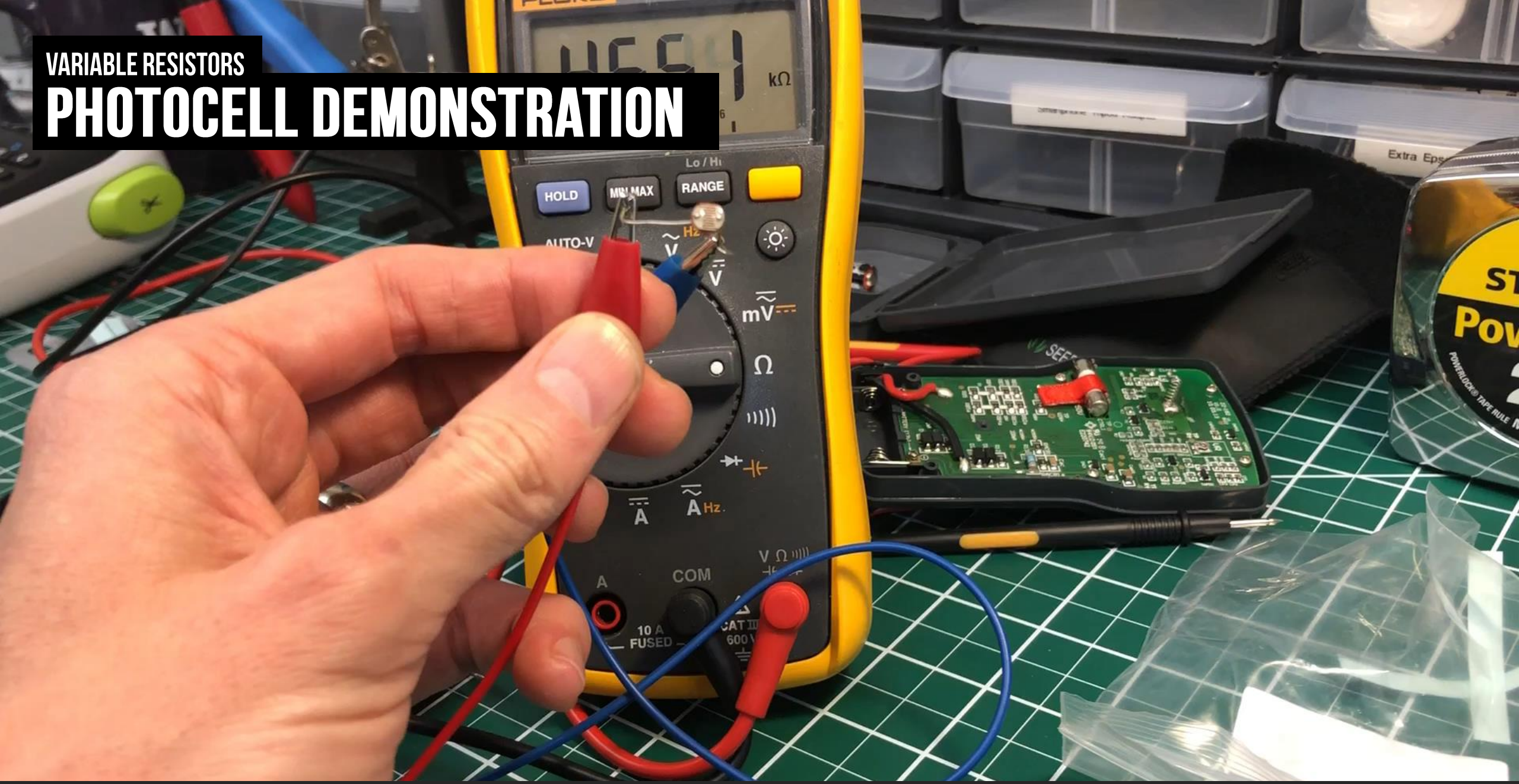
Photocell (aka photodetector or photo resistor); $1.50

Thermistor 10k; $0.75

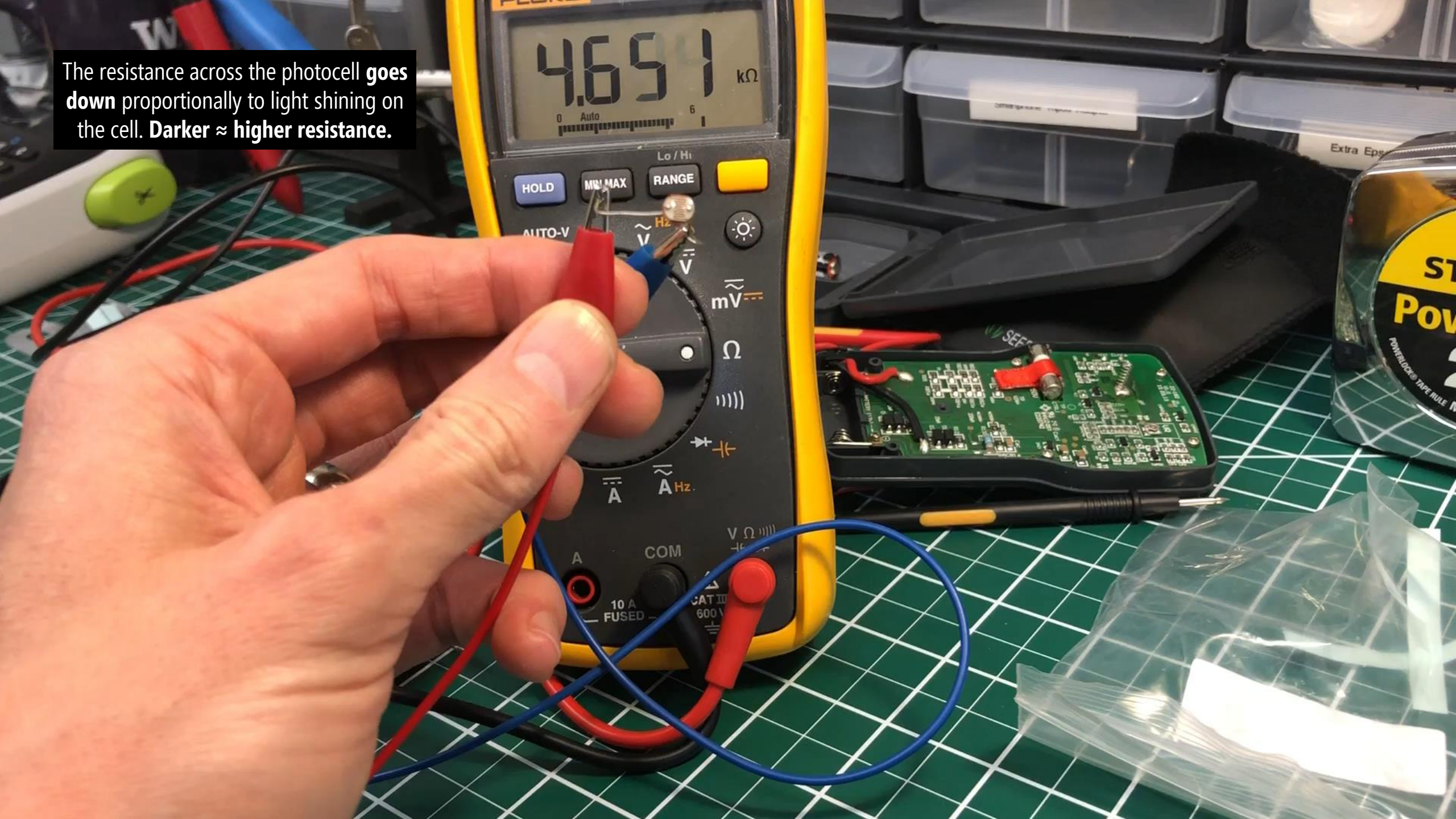
Force Resistive Sensor 0.5”; $6.95

Flex Sensor 4.5”; $12.95

*Note: Prices are from Sparkfun.com; parts are cheaper in bulk and often cheaper from Digikey.com



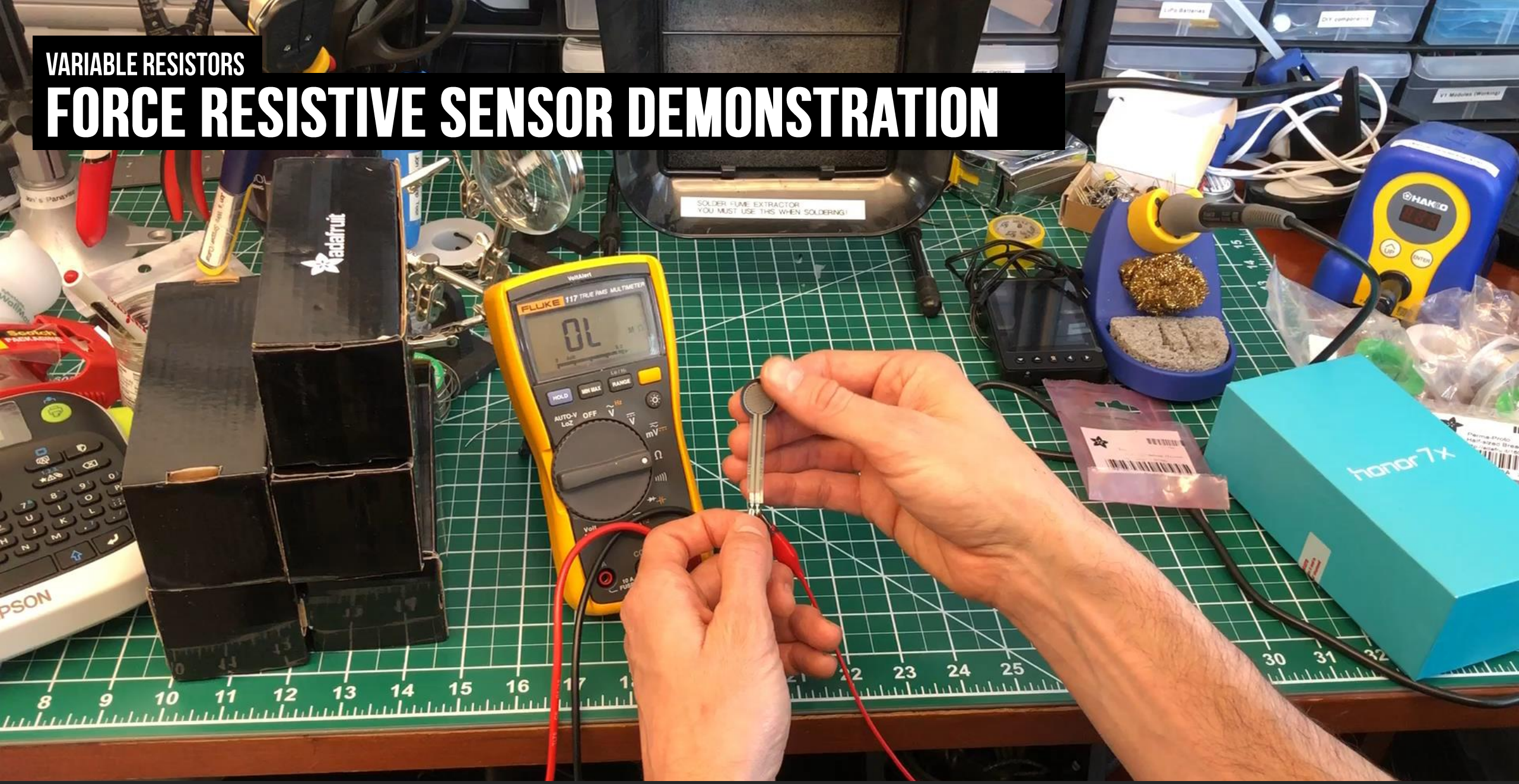
Video source: Jon Froehlich



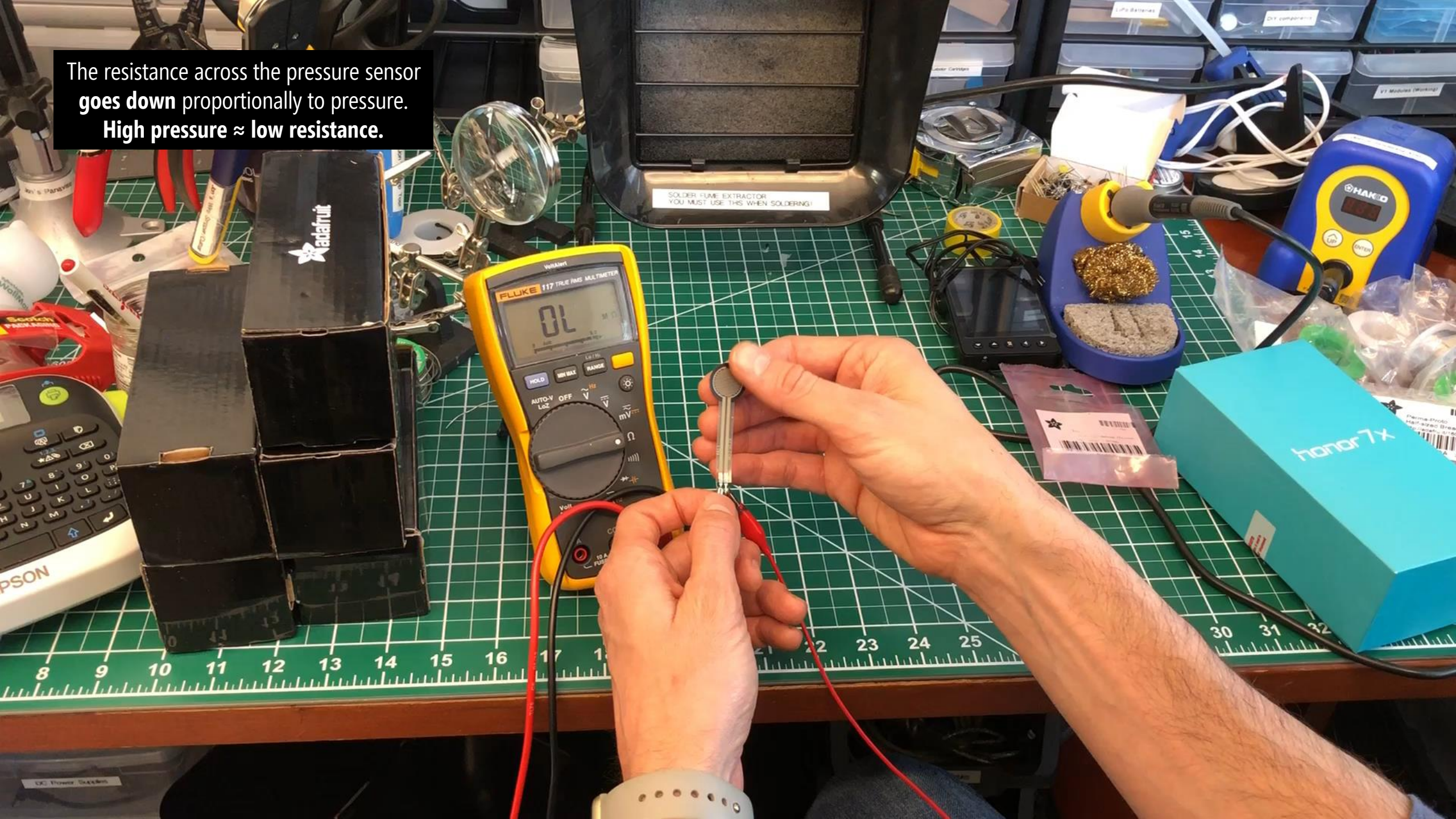
The resistance across the photocell **goes down** proportionally to light shining on the cell. **Darker ≈ higher resistance.**

VARIABLE RESISTORS

FORCE RESISTIVE SENSOR DEMONSTRATION



Video source: Jon Froehlich

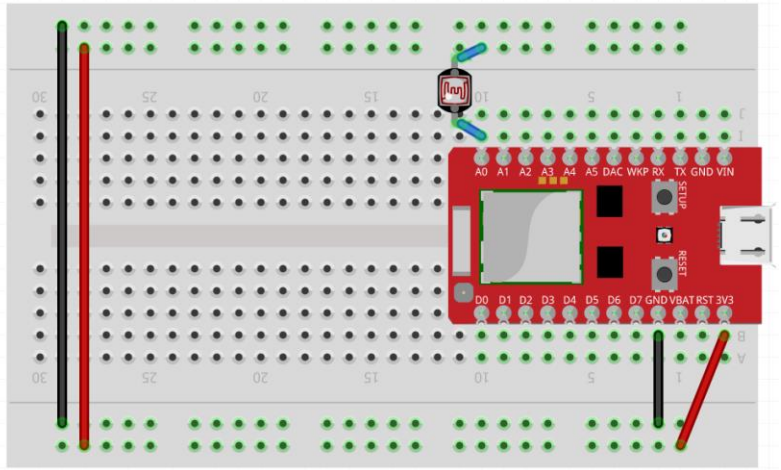

The resistance across the pressure sensor
goes down proportionally to pressure.
High pressure ≈ low resistance.
SOLDER FUME EXTRACTOR
YOU MUST USE THIS WHEN SOLDERING!
FLUKE 117 TRUE RMS MULTIMETER
OL
HOLD
MIN MAX
RANGE
AUTO-V
OFF
adafruit
HAKKO
honor 7x

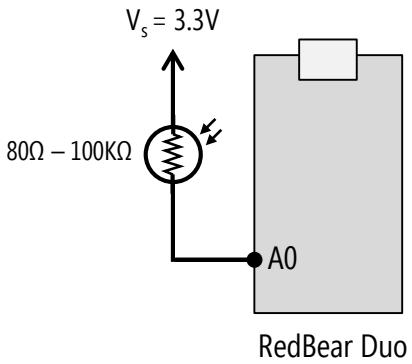
Awesome! Let's wire up a **photocell**.

USING A VARIABLE RESISTIVE SENSOR

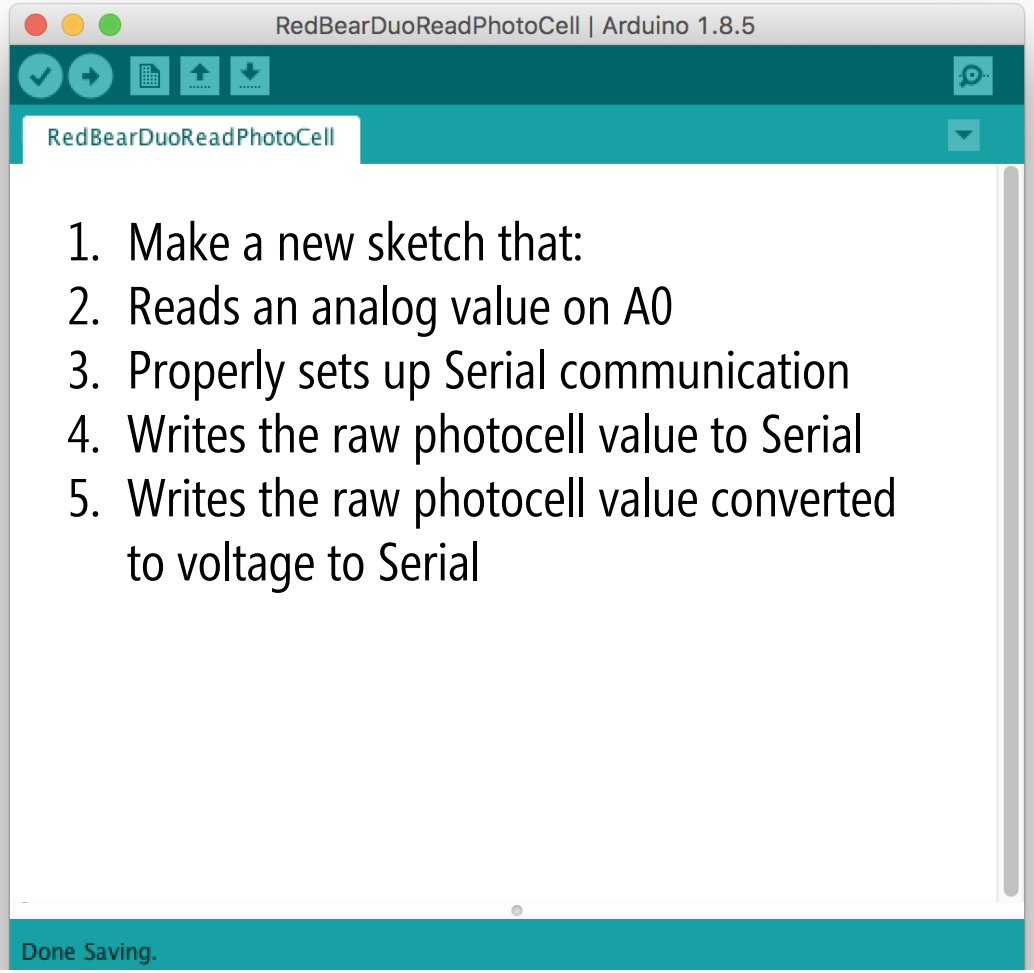
WIRE UP A PHOTOCELL & PRINT VALUE & VOLTAGE

Circuit: Read a photocell





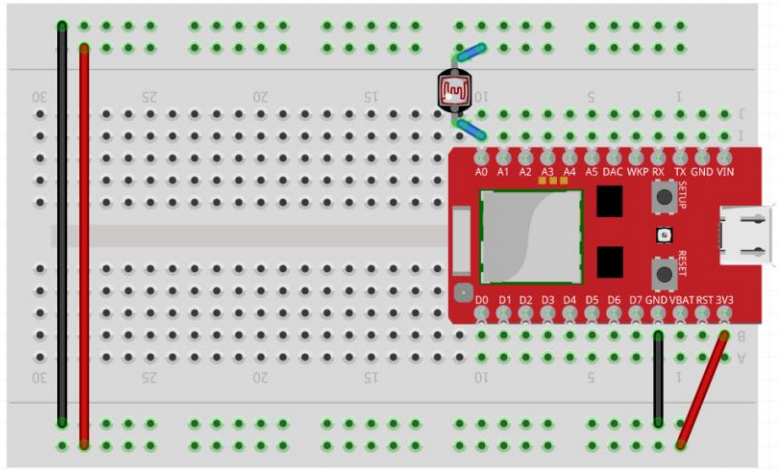
Code: Read a photocell value and print to serial

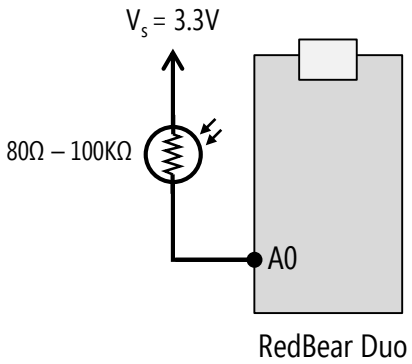


USING A VARIABLE RESISTIVE SENSOR

WIRE UP A PHOTOCELL & PRINT VALUE & VOLTAGE

Circuit: Read a photocell





Code: Read a photocell value and print to serial

RedBearDuoReadPhotoCell | Arduino 1.8.5

RedBearDuoReadPhotoCell

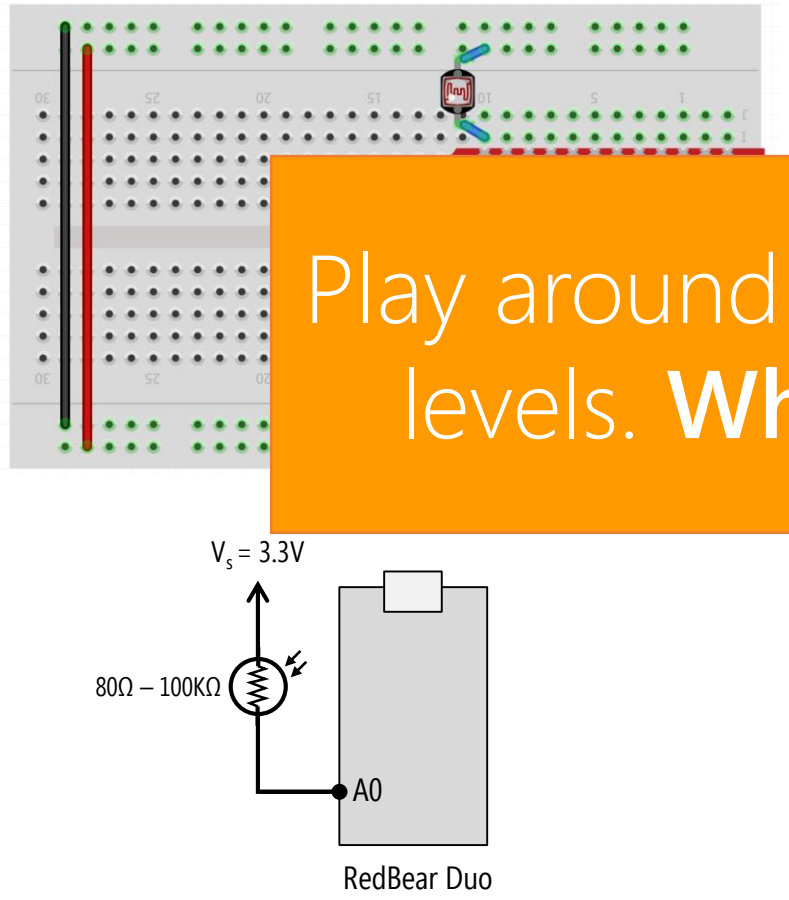
```
SYSTEM_MODE(MANUAL); // required to use RedBear Duo

const int PHOTOCELL_INPUT_PIN = A0;
const int MAX_ADC_VALUE = 4096; // 12-bit ADC
const float MAX_VOLTAGE = 3.3;

void setup() {
  pinMode(PHOTOCELL_INPUT_PIN, INPUT);
  pinMode(D0, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // Read the photocell value (which will range from 0 to 4096)
  int photocellVal = analogRead(PHOTOCELL_INPUT_PIN);

  // Convert to a voltage
  float inputVoltage = MAX_VOLTAGE * photocellVal / MAX_ADC_VALUE;

  // Print to serial
  Serial.print(photocellVal);
  Serial.print("\t");
  Serial.println(inputVoltage);
  delay(100);
}
```

Done Saving.

USING A VARIABLE RESISTIVE SENSOR

WIRE UP A PHOTOCELL & PRINT VALUE & VOLTAGE

Circuit: Read a photocell



Code: Read a photocell value and print to serial

RedBearDuoReadPhotoCell | Arduino 1.8.5

RedBearDuoReadPhotoCell

```
  // Read the photocell value (which will range from 0 to 4096)
  int photocellVal = analogRead(PHOTOCELL_INPUT_PIN);

  // Convert to a voltage
  float inputVoltage = MAX_VOLTAGE * photocellVal / MAX_ADC_VALUE;

  // Print to serial
  Serial.print(photocellVal);
  Serial.print("\t");
  Serial.println(inputVoltage);
  delay(100);
}
```

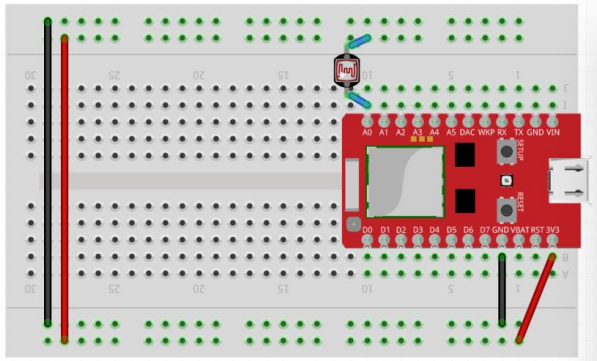
Done Saving.

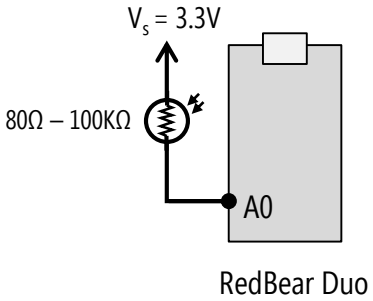
Play around with different light/dark levels. **What do you observe?**

USING A VARIABLE RESISTIVE SENSOR

WIRE UP A PHOTOCELL & PRINT VALUE & VOLTAGE

Circuit: Read a photocell





Code: Read a photocell & print to serial

RedBearDuoReadPhotoCell | Arduino 1.8.5

RedBearDuoReadPhotoCell

```
SYSTEM_MODE(MANUAL); // required to use RedBear Duo

const int PHOTOCELL_INPUT_PIN = A0;
const int MAX_ADC_VALUE = 4096; // 12-bit ADC
const float MAX_VOLTAGE = 3.3;

void setup() {
  pinMode(PHOTOCELL_INPUT_PIN, INPUT);
  pinMode(D0, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // Read the photocell value (which will range from 0 to 4096)
  int photocellVal = analogRead(PHOTOCELL_INPUT_PIN);

  // Convert to a voltage
  float inputVoltage = MAX_VOLTAGE * photocellVal / MAX_ADC_VALUE;

  // Print to serial
  Serial.print(photocellVal);
  Serial.print("\t");
  Serial.println(inputVoltage);
  delay(100);
}
```

Done Saving.

Output: analogRead val and voltage

/dev/cu.usbmodem14611

Send

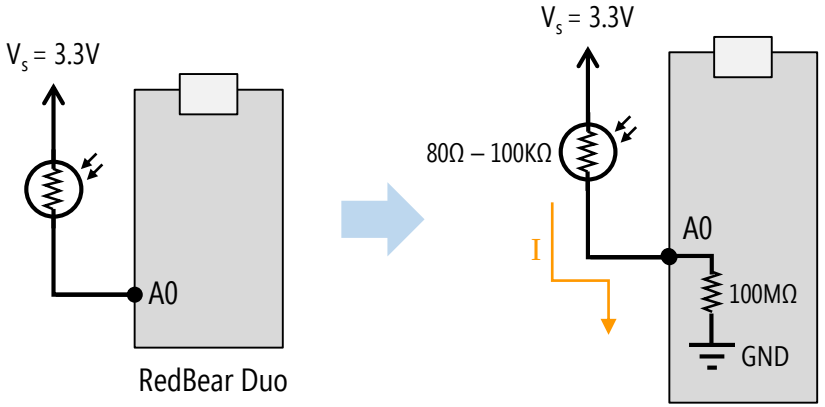
```
4093    3.30
4092    3.30
4092    3.30
4092    3.30
4093    3.30
4092    3.30
4091    3.30
4092    3.30
4092    3.30
4090    3.30
4092    3.30
4091    3.30
4092    3.30
4093    3.30
4090    3.30
4091    3.30
4089    3.29
4091    3.30
4093    3.30
4092    3.30
4093    3.30
4092    3.30
4092    3.30
4093    3.30
```

Hmm, the values seem to be **stuck at ~4096** (or 3.3V). Why is this?

Autoscroll | No line ending | 9600 baud | Clear output

USING A VARIABLE RESISTIVE SENSOR

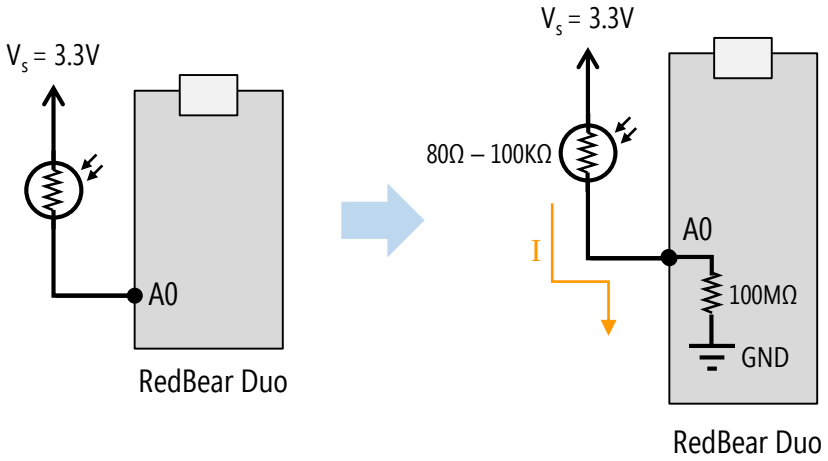
WHY DOESN'T THIS CIRCUIT WORK?



Microcontroller **input pins** have incredibly **high input resistance** (effectively infinite). In fact, you can assume that very little current flows in or out of an input pin.

USING A VARIABLE RESISTIVE SENSOR

WHY DOESN'T THIS CIRCUIT WORK?



Microcontroller **input pins** have incredibly **high input resistance** (effectively infinite). In fact, you can assume that very little current flows in or out of an input pin. Let's check this!

Solve for current I when $R_{photocell}$=80Ω

$$Current = I = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I = \frac{V_s - GND}{R_1 + R_2}$$

Where R_1=$R_{photocell}$ and R_2=internal resistance on input pin

$$I = \frac{3.3V - 0}{80\Omega + 100{,}000{,}000\Omega} = 0.000000033A = 33nA$$

Solve for the V_{A0}

$$V_s - V_{A0} = I * R_1$$

Ohm's law states that the voltage difference across a resistor is equal to I * R

$$V_{A0} = V_s - I * R_1$$

Rearrange the equation a bit to orient things in terms of V_{out}

$$V_{A0} = 3.3V - 33nA * 80\Omega$$

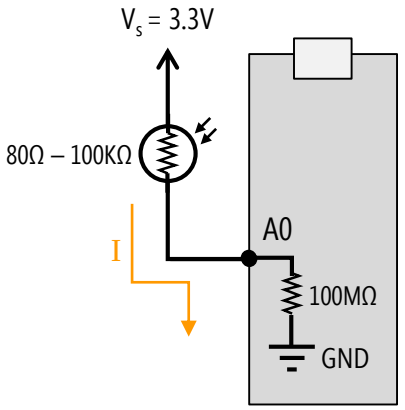
Plugin numbers

$$V_{A0} = 3.3V - 0.00000264V$$

$$V_{A0} = 3.299997V$$

USING A VARIABLE RESISTIVE SENSOR

WHY DOESN'T THIS CIRCUIT WORK?



RedBear Duo

Microcontroller **input pins** have incredibly **high input resistance** (effectively infinite). In fact, you can assume that very little current flows in or out of an input pin.

Solve for current I when $R_{photocell}$=80Ω

$$Current = I = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I = \frac{V_s - GND}{R_1 + R_2}$$

$$I = \frac{3.3V - 0}{80\Omega + 100{,}000{,}000\Omega} = 33nA$$

Solve for the V_{A0}

$$V_s - V_{A0} = I * R_1$$

$$V_{A0} = V_s - I * R_1$$

$$V_{A0} = 3.3V - 33nA * 80\Omega$$

$$V_{A0} = 3.3V - 0.00000264V$$

$$V_{A0} = 3.299997V$$

Solve for current I when $R_{photocell}$=100KΩ

$$Current = I = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I = \frac{V_s - GND}{R_1 + R_2}$$

$$I = \frac{3.3V - 0}{100K\Omega + 100M\Omega} = 32.97nA$$

Solve for the V_{A0}

$$V_s - V_{A0} = I * R_1$$

$$V_{A0} = V_s - I * R_1$$

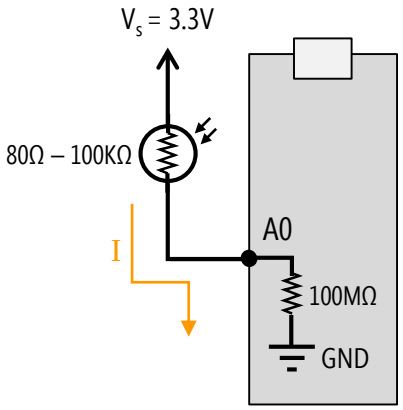
$$V_{A0} = 3.3V - 32.97nA * 100{,}000\Omega$$

$$V_{A0} = 3.3V - 0.00329V$$

$$V_{A0} = 3.2967V$$

USING A VARIABLE RESISTIVE SENSOR

WHY DOESN'T THIS CIRCUIT WORK?



RedBear Duo

Microcontroller **input pins** have incredibly **high input resistance** (effectively infinite). In fact, you can assume that very little current flows in or out of an input pin.

Solve for current I when $R_{photocell}$=80Ω

$$Current = I = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I = \frac{V_s - GND}{R_1 + R_2}$$

$$I = \frac{3.3V - 0}{80\Omega + 100{,}000{,}000\Omega} = 33nA$$

Solve for current I when $R_{photocell}$=100KΩ

$$Current = I = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I = \frac{V_s - GND}{R_1 + R_2}$$

$$I = \frac{3.3V - 0}{100K\Omega + 100M\Omega} = 32.97nA$$

Solve for the V_{A0}

Wow, OK, no wonder we are not seeing any changes in our photocell code. Our input voltage is not varying at all! So what do we do?

$$V_{A0} = 3.3V - 0.00000264V$$

$$V_{A0} = 3.299997V$$

Solve for the V_{A0}

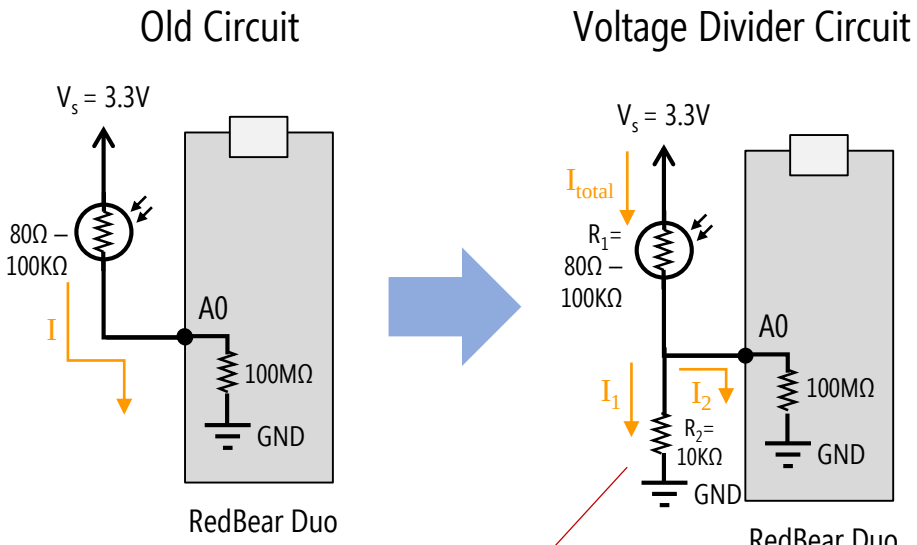
$7nA * 100{,}000\Omega$

$$V_{A0} = 3.3V - 0.00329V$$

$$V_{A0} = 3.2967V$$

USING A VARIABLE RESISTIVE SENSOR

ADD A RESISTOR TO CREATE A VOLTAGE DIVIDER

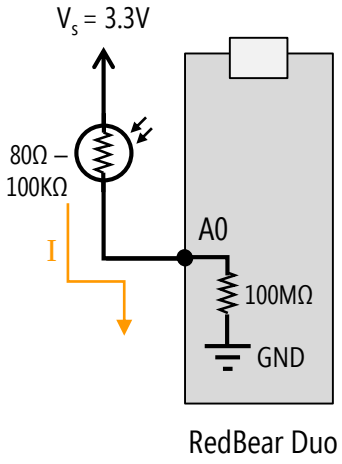


This is our **voltage divider resistor**. The **ratio** between this resistor and R_1 is what's important.

USING A VARIABLE RESISTIVE SENSOR

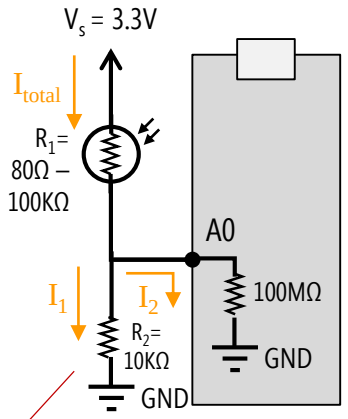
ADD A RESISTOR TO CREATE A VOLTAGE DIVIDER

Old Circuit



RedBear Duo

Voltage Divider Circuit



RedBear Duo

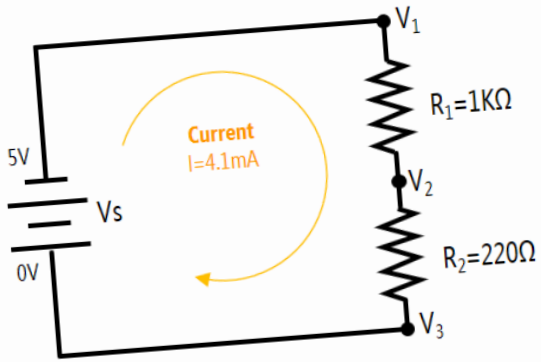
This is our **voltage divider resistor**. The **ratio** between this resistor and R_1 is what's important.

No magic here! We even covered this in our circuit lecture! ☺

SERIES VS. PARALLEL RESISTANCE

OHM'S LAW EXERCISE: SOLVE FOR V1, V2, AND V3

How did I know V1 and V3?



$$Current = I = \frac{V_{high\ potential} - Vlow_{\ potential}}{R}$$

$$V_1 - V_2 = I * R_1$$

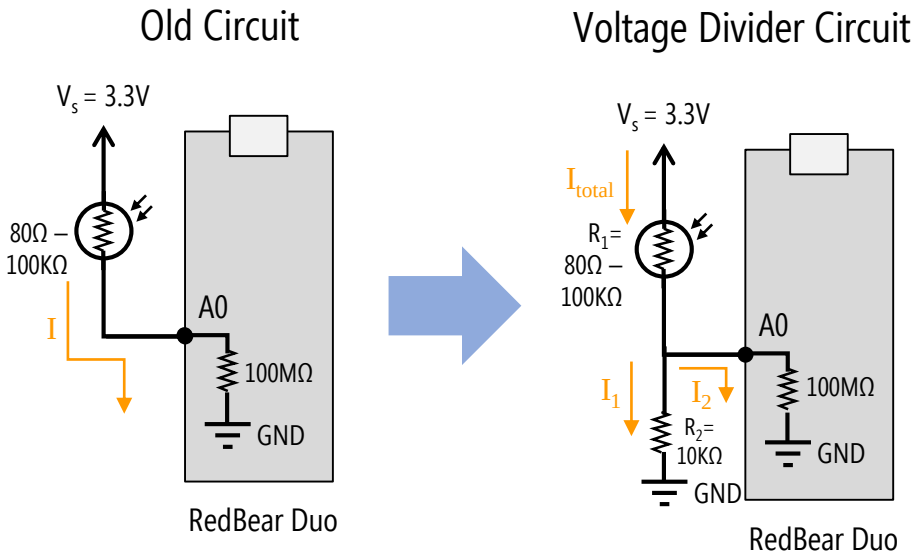
$$V_2 = V_1 - I * R_1$$ And we know $V_1 = 5V$

$$V_2 = 5V - 0.0041A * 1{,}000\Omega = 0.9V$$

$$V_3 = 0V$$

USING A VARIABLE RESISTIVE SENSOR

ADD A RESISTOR TO CREATE A VOLTAGE DIVIDER



We know that I_2 is super small. So, $I_{total} \approx I_1$
Let's solve for current I_{total} when R_1=80Ω & R_2=10KΩ

$$Current = I_{total} = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I_{total} = \frac{V_s - GND}{R_1 + R_2}$$

$$I_{total} = \frac{3.3V - 0}{80\Omega + 10{,}000\Omega} = 0.000327A = 0.33mA$$

Solve for the V_{A0}

$$V_s - V_{A0} = I * R_1$$

$$V_{A0} = V_s - I * R_1$$

$$V_{A0} = 3.3V - 0.33mA * 80\Omega$$

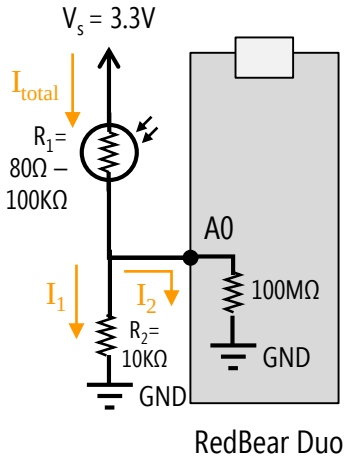
$$V_{A0} = 3.3V - 0.0262V$$

$$V_{A0} = 3.27V$$

USING A VARIABLE RESISTIVE SENSOR

ADD A RESISTOR TO CREATE A VOLTAGE DIVIDER

Voltage Divider Circuit



Let's solve for current I_{total} when R_1=80Ω & R_2=10KΩ

$$Current = I_{total} = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I_{total} = \frac{V_s - GND}{R_1 + R_2}$$

$$I_{total} = \frac{3.3V - 0}{80\Omega + 10{,}000\Omega} = 0.33mA$$

Solve for the V_{A0}

$$V_s - V_{A0} = I * R_1$$

$$V_{A0} = V_s - I * R_1$$

$$V_{A0} = 3.3V - 0.33mA * 80\Omega$$

$$V_{A0} = 3.3V - 0.0262V$$

$$V_{A0} = 3.27V$$

Let's solve for current I_{total} when R_1=100KΩ & R_2=10KΩ

$$Current = I_{total} = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I_{total} = \frac{V_s - GND}{R_1 + R_2}$$

$$I_{total} = \frac{3.3V - 0}{100K\Omega + 10{,}000\Omega} = 0.03mA$$

Solve for the V_{A0}

$$V_s - V_{A0} = I * R_1$$

$$V_{A0} = V_s - I * R_1$$

$$V_{A0} = 3.3V - 0.03mA * 100K\Omega$$

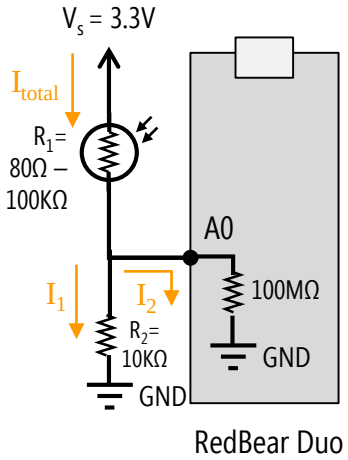
$$V_{A0} = 3.3V - 3V$$

$$V_{A0} = 0.3V$$

USING A VARIABLE RESISTIVE SENSOR

ADD A RESISTOR TO CREATE A VOLTAGE DIVIDER

Voltage Divider Circuit



Let's solve for current I_{total} when R_1=80Ω & R_2=10KΩ

$$Current = I_{total} = \frac{V_{high\ potential} + V_{low\ potential}}{R_1 + R_2}$$

$$I_{total} = \frac{V_s - GND}{R_1 + R_2}$$

$$I_{total} = \frac{3.3V - 0}{80\Omega + 10{,}000\Omega} = 0.33mA$$

Solve for the V_{A0}

$$V_{A0} = 3.3V - 0.0262V$$

$$V_{A0} = 3.27V$$

Let's solve for current I_{total} when R_1=100KΩ & R_2=10KΩ

$$Current = I_{total} = \frac{V_{high\ potential} + V_{low\ potential}}{R_1 + R_2}$$

$$I_{total} = \frac{V_s - GND}{R_1 + R_2}$$

$$I_{total} = \frac{3.3V - 0}{100K\Omega + 10{,}000\Omega} = 0.03mA$$

Solve for the V_{A0}

R_1

R_1

$0.03mA * 100K\Omega$

$$V_{A0} = 3.3V - 3V$$

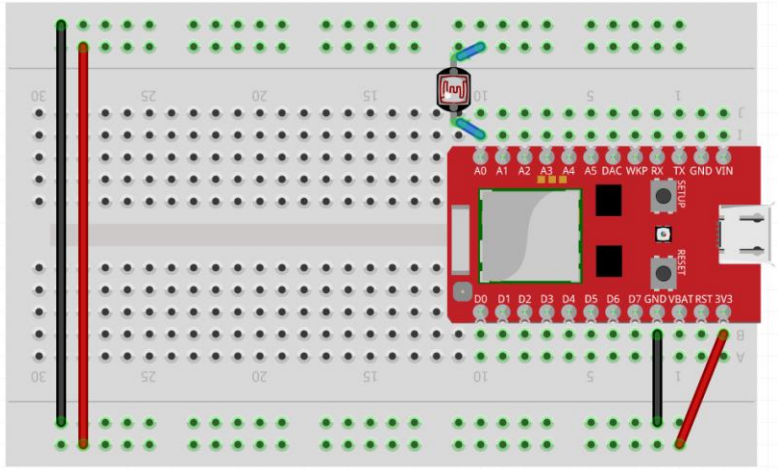
$$V_{A0} = 0.3V$$

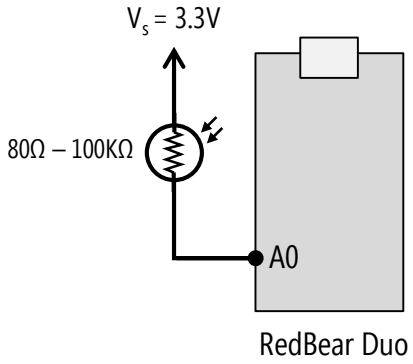
This is more like it! Now we are seeing a **nice range of input voltages** in response to the changing resistance of our photocell! Thanks **voltage divider**!

OK, let’s try this again. Wire up a **photocell but with a voltage divider**!

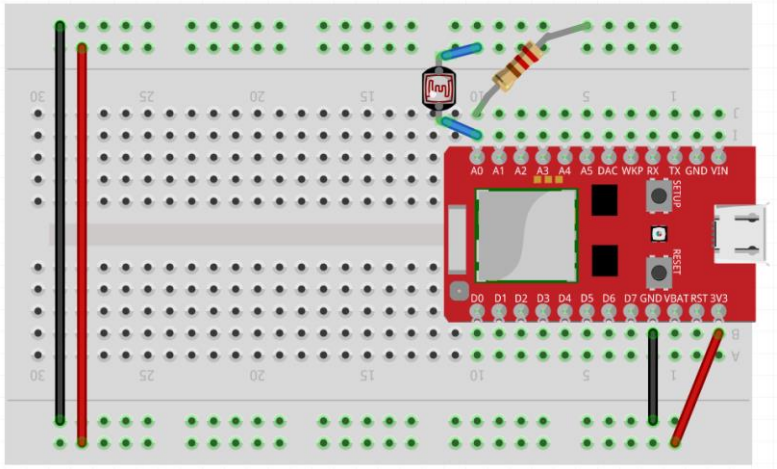
WIRE UP A PHOTOCELL & PRINT VALUE & VOLTAGE

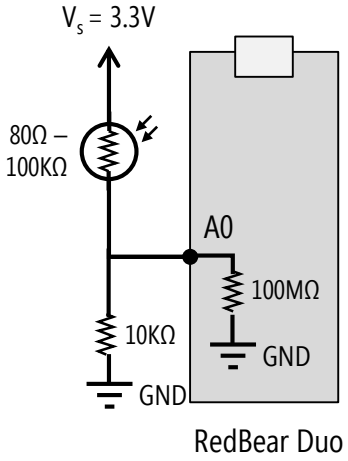
Incorrect Circuit: Read a photocell





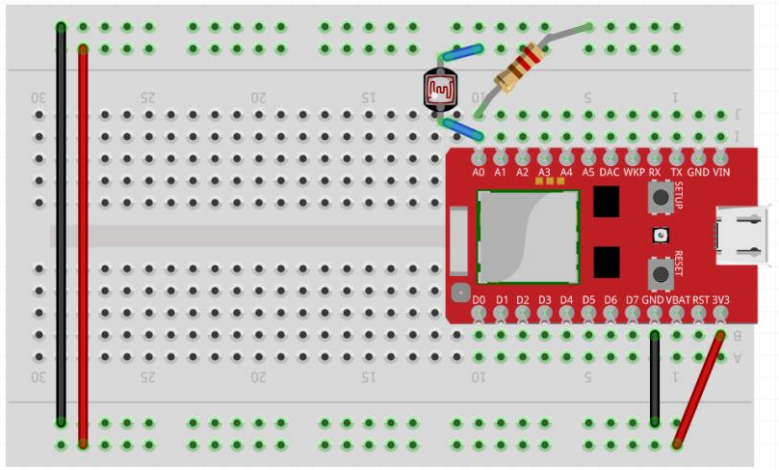
Correct Circuit: Read a photocell via voltage divider

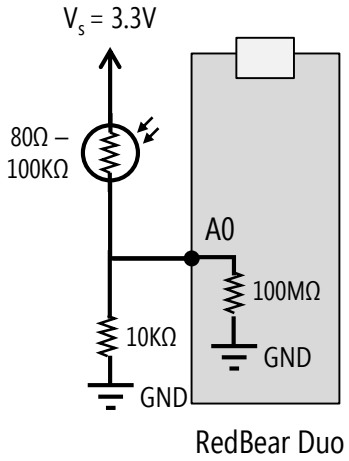




WIRE UP A PHOTOCELL & PRINT VALUE & VOLTAGE

Circuit: Read a photocell via voltage divider





Code: Same as before (no changes)

RedBearDuoReadPhotoCell | Arduino 1.8.5

RedBearDuoReadPhotoCell

```
SYSTEM_MODE(MANUAL); // required to use RedBear Duo

const int PHOTOCELL_INPUT_PIN = A0;
const int MAX_ADC_VALUE = 4096; // 12-bit ADC
const float MAX_VOLTAGE = 3.3;

void setup() {
  pinMode(PHOTOCELL_INPUT_PIN, INPUT);
  pinMode(D0, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // Read the photocell value (which will range from 0 to 4096)
  int photocellVal = analogRead(PHOTOCELL_INPUT_PIN);

  // Convert to a voltage
  float inputVoltage = MAX_VOLTAGE * photocellVal / MAX_ADC_VALUE;

  // Print to serial
  Serial.print(photocellVal);
  Serial.print("\t");
  Serial.println(inputVoltage);
  delay(100);
}
```

Done Saving.

Output: Now varies from ~0.3V – 3.28V

/dev/cu.usbmodem14611

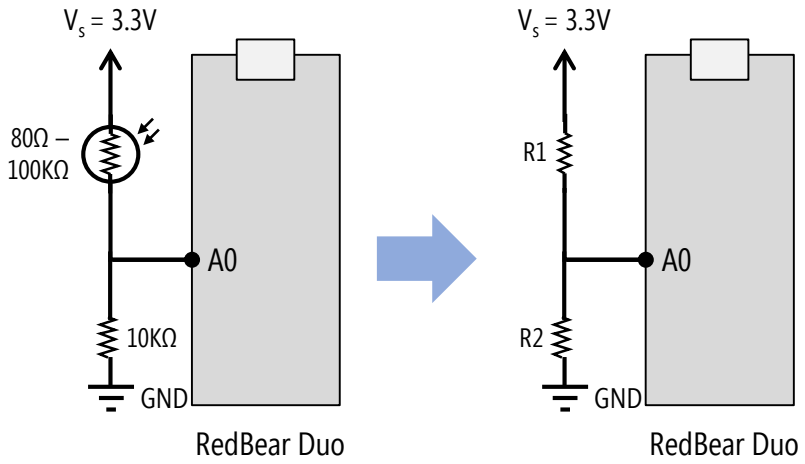
Send

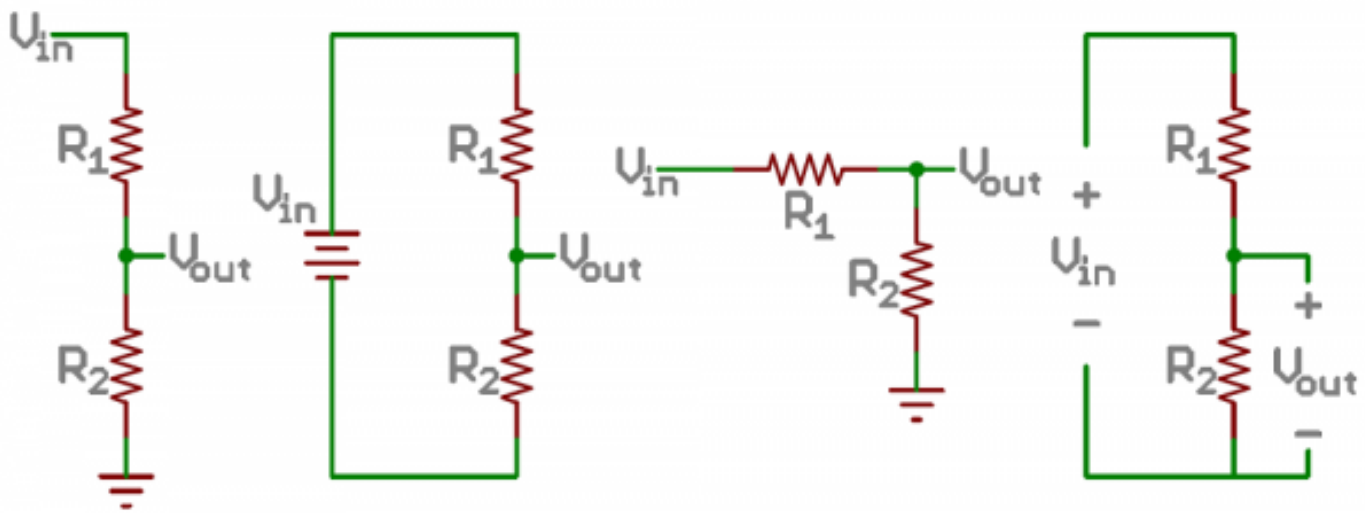
```
2641    2.13
1998    1.61
1572    1.27
1570    1.26
1289    1.04
1963    1.58
682     0.55
1636    1.32
1368    1.10
3867    3.12
3883    3.13
3880    3.13
3882    3.13
4032    3.25
3998    3.22
4048    3.26
3985    3.21
3308    2.67
1319    1.06
964     0.78
2034    1.64
3893    3.14
3891    3.13
```

Autoscroll | No line ending | 9600 baud | Clear output

VOLTAGE DIVIDER SCHEMATICS

Voltage dividers are often written in a variety of ways (but they are all the same circuit!)

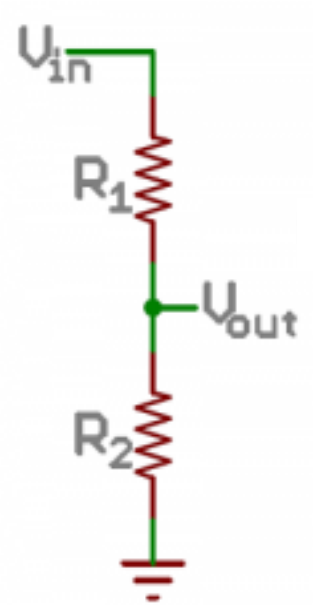




Source: https://learn.sparkfun.com/tutorials/voltage-dividers

VOLTAGE DIVIDER EQUATION

The voltage divider equation helps simplify our calculations. Let's see how!



Solve for current I

$$Current = I = \frac{V_{high\ potential} - V_{low\ potential}}{R_1 + R_2}$$

$$I = \frac{V_{in} - GND}{R_1 + R_2} = \frac{V_{in} - 0V}{R_1 + R_2} = \frac{V_{in}}{R_1 + R_2}$$

Solve for the V_{out}

$V_{in} - V_{out} = I * R_1$ Ohm's law states that the voltage difference across a resistor is equal to I * R

$V_{out} = V_{in} - I * R_1$ Rearrange the equation a bit to orient things in terms of V_{out}

This is how we've been solving for V_{out} on previous slides but there is a simpler way.

But we could also write V_{out} in terms of bottom half of circuit

$V_{out} - GND = I * R_2$ Or similarly, you could start with V_{out} as high potential and GND as low potential

$V_{out} = I * R_2$

$V_{out} = \frac{V_{in}}{R_1 + R_2} * R_2$ Replaced I with the equation above.

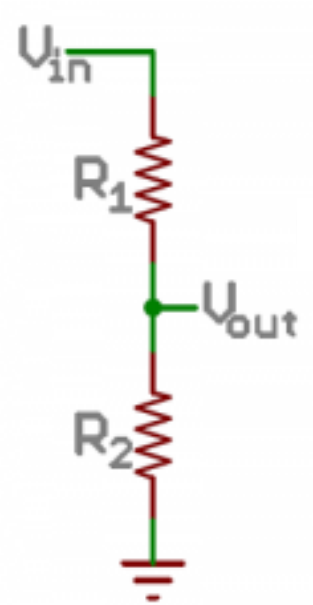
$V_{out} = \frac{R_2}{R_1 + R_2} * V_{in}$

This is the canonical **voltage divider** equation

Source: https://learn.sparkfun.com/tutorials/voltage-dividers

VOLTAGE DIVIDER EQUATION

A few generalizations to keep in mind



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

$$if\, R_1 = R_2 : V_{out} = V_{in} \cdot \frac{R}{2R} = \frac{V_{in}}{2}$$

$$if\, R_2 \gg R_1 : V_{out} \approx V_{in} \cdot \frac{R_2}{R_2} = V_{in}$$

$$if\, R_2 \ll R_1 : V_{out} \approx V_{in} \cdot \frac{0}{R_1} = 0$$

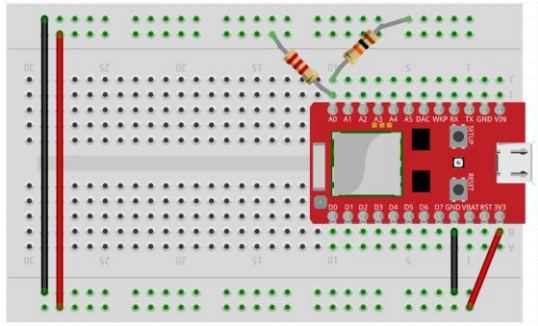
Source: https://learn.sparkfun.com/tutorials/voltage-dividers

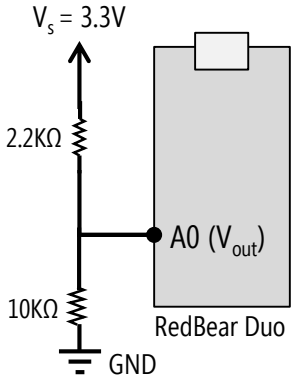
Let’s test and reinforce our knowledge here and **try out multiple different voltage dividers**. We will calculate V_{out} and then test our answer.
Use the same Arduino code as before.

VOLTAGE DIVIDER EXERCISE

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

Build Circuit: Voltage Divider



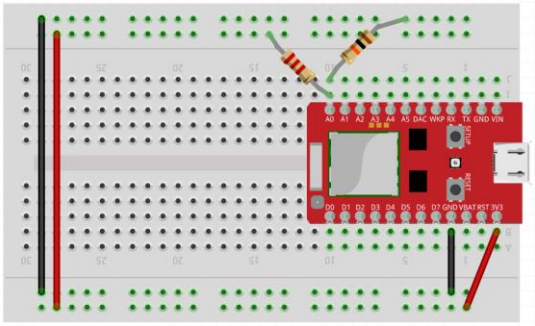


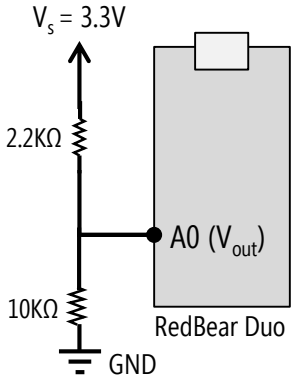
$V_{out} = ?$

VOLTAGE DIVIDER EXERCISE

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

Build Circuit: Voltage Divider





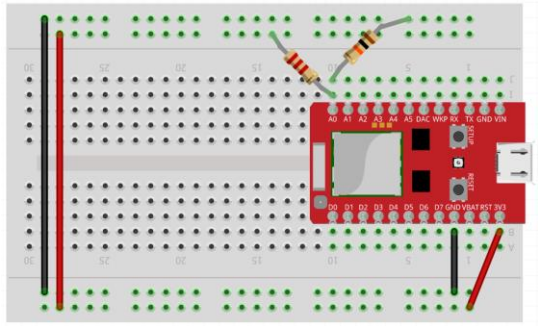
$$V_{out} = \frac{3.3V}{2.2K\Omega + 10K\Omega} * 10K\Omega$$

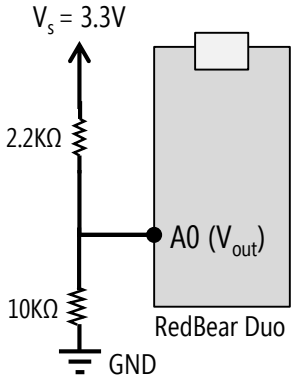
$$V_{out} = 2.7V$$

VOLTAGE DIVIDER EXERCISE

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

Build Circuit: Voltage Divider

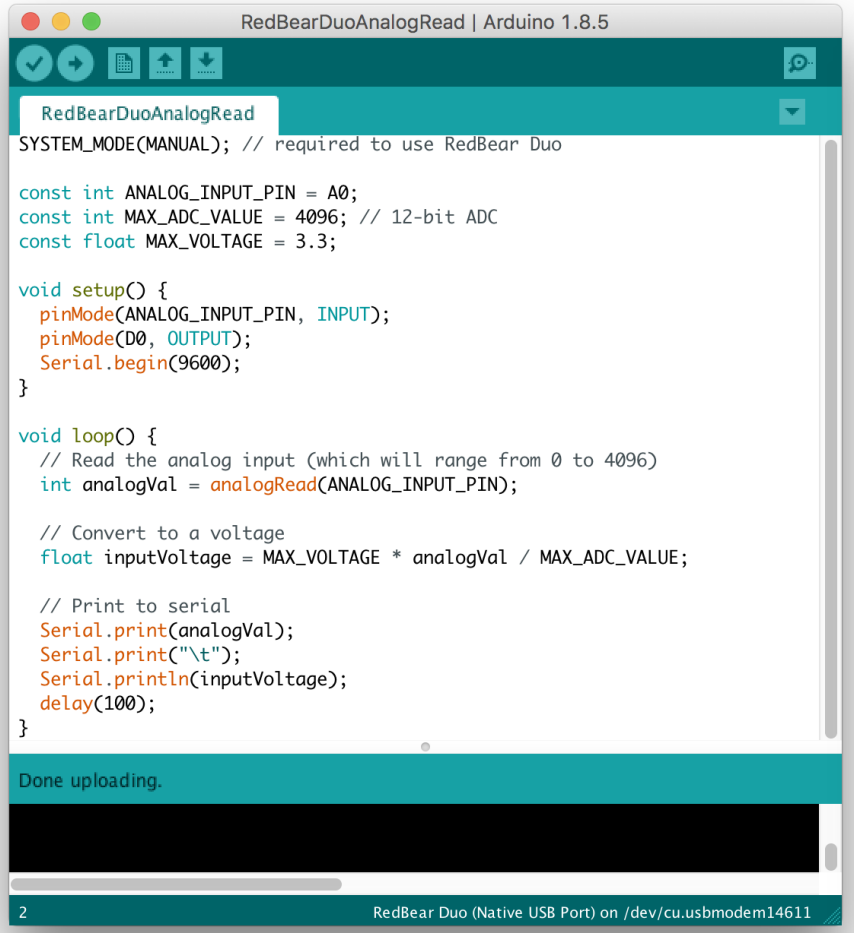




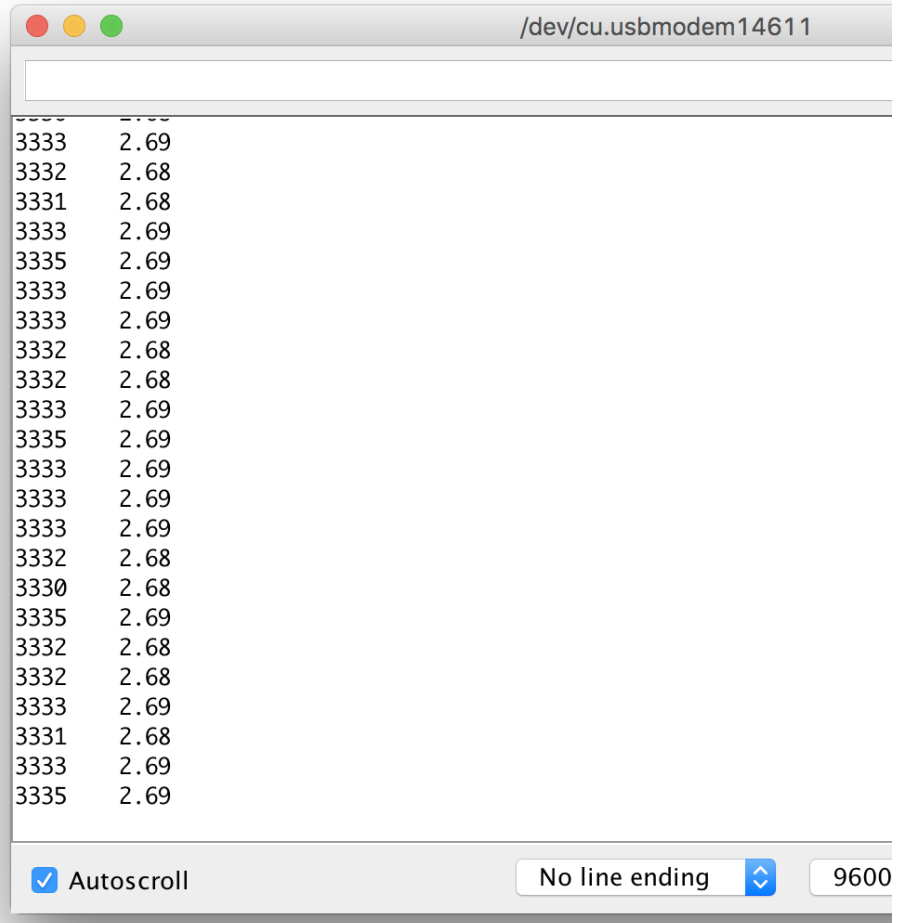
$$V_{out} = \frac{3.3V}{2.2K\Omega + 10K\Omega} * 10K\Omega$$

$$V_{out} = 2.7V$$

Code: Print analog value + voltage (same as before)



Serial Output: This is how we can verify our calculated answer!

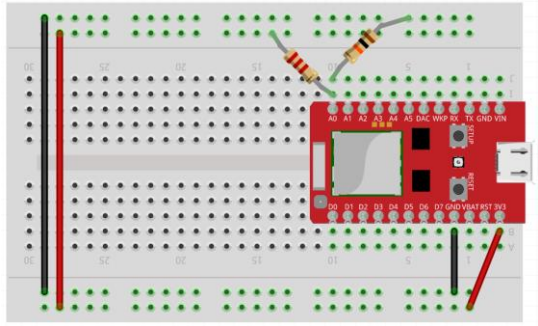


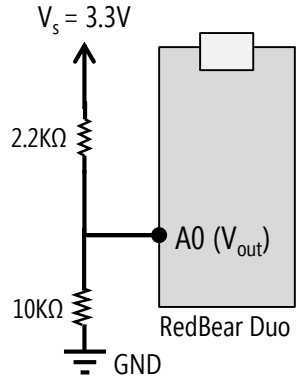
$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2}$$

VOLTAGE DIVIDER EXERCISE

It's the ratio between the two resistors that's important

Voltage Divider: (R1, R2) = (2.2KΩ, 10KΩ)

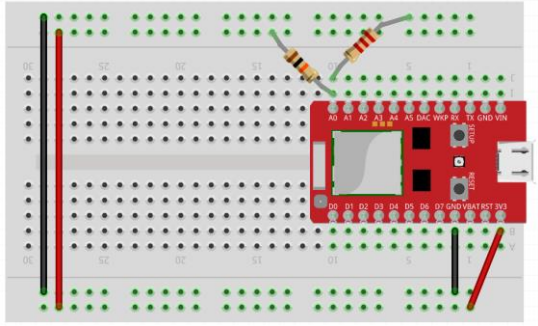


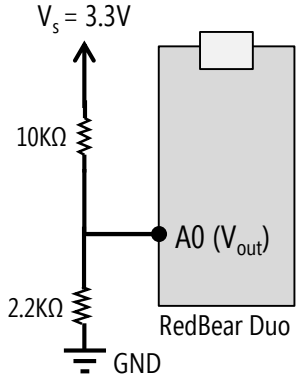


$$V_{out} = \frac{3.3V}{2.2K\Omega + 10K\Omega} * 10K\Omega$$

$$V_{out} = 2.7V$$

Voltage Divider: (R1, R2) = (10KΩ, 2.2KΩ)





$$V_{out} = \frac{3.3V}{10K\Omega + 2.2K\Omega} * 2.2K\Omega$$

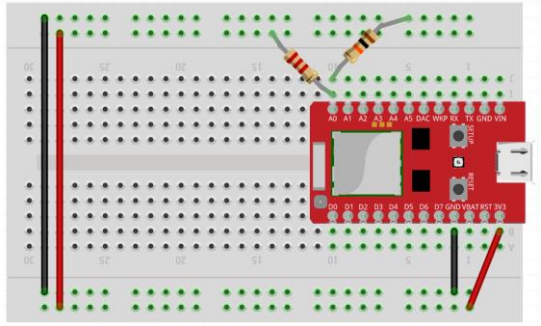
$$V_{out} = 0.6V$$

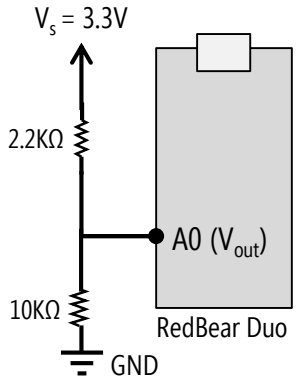
$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2}$$

VOLTAGE DIVIDER EXERCISE

It's the ratio between the two resistors that's important

Voltage Divider: (R1, R2) = (2.2KΩ, 10KΩ)

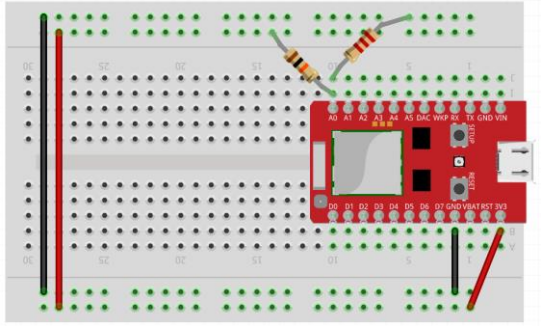


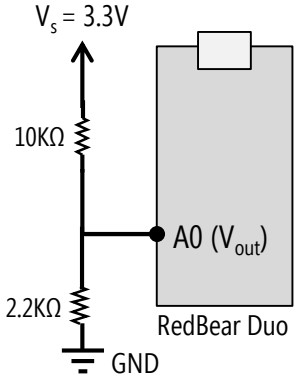


$$V_{out} = \frac{3.3V}{2.2K\Omega + 10K\Omega} * 10K\Omega$$

$$V_{out} = 2.7V$$

Voltage Divider: (R1, R2) = (10KΩ, 2.2KΩ)

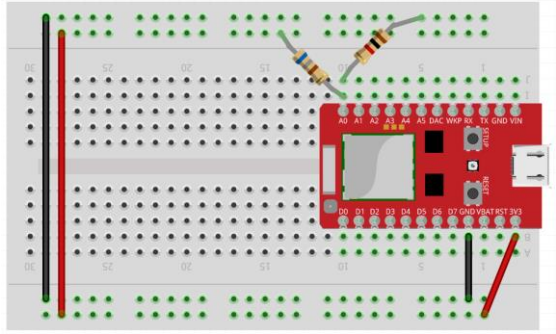


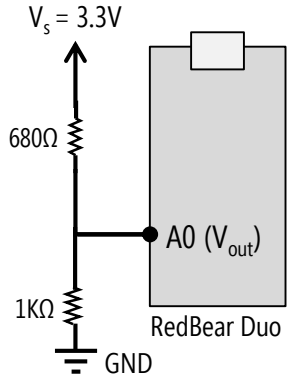


$$V_{out} = \frac{3.3V}{10K\Omega + 2.2K\Omega} * 2.2K\Omega$$

$$V_{out} = 0.6V$$

Voltage Divider: (R1, R2) = (680Ω, 1KΩ)

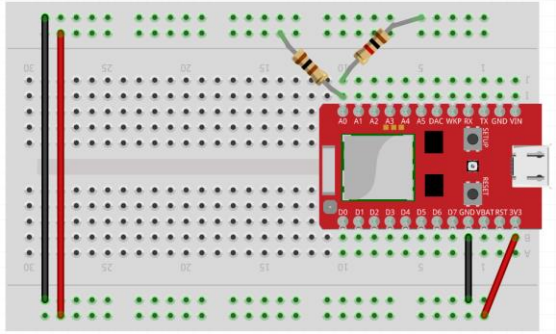


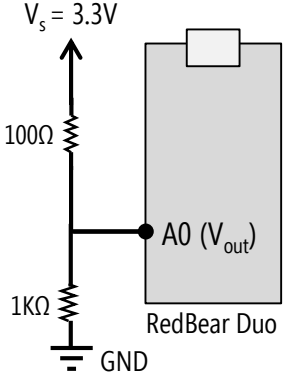


$$V_{out} = \frac{3.3V}{680\Omega + 1K\Omega} * 1\text{KΩ}$$

$$V_{out} = 1.96V$$

Voltage Divider: (R1, R2) = (100Ω, 1KΩ)





$$V_{out} = \frac{3.3V}{100\Omega + 1K\Omega} * 1\text{KΩ}$$

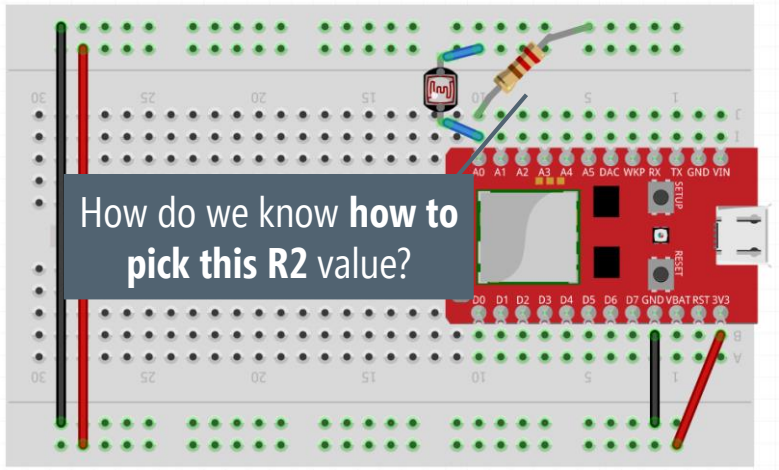
$$V_{out} = 3V$$

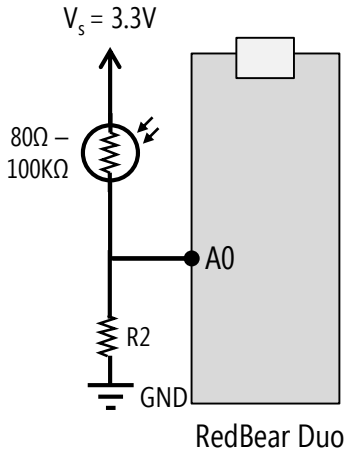
USING A VARIABLE RESISTIVE SENSOR

CHOOSING R2 IN VOLTAGE DIVIDER CIRCUITS

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

Circuit: Read a photocell via voltage divider





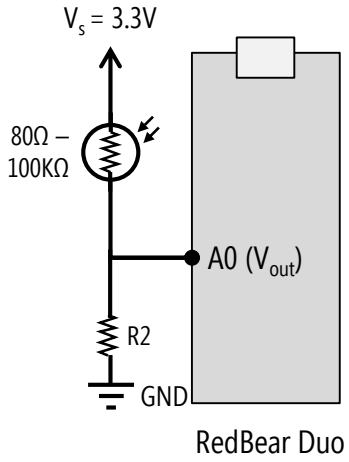
It depends on the resistance range of your variable resistor (R1)

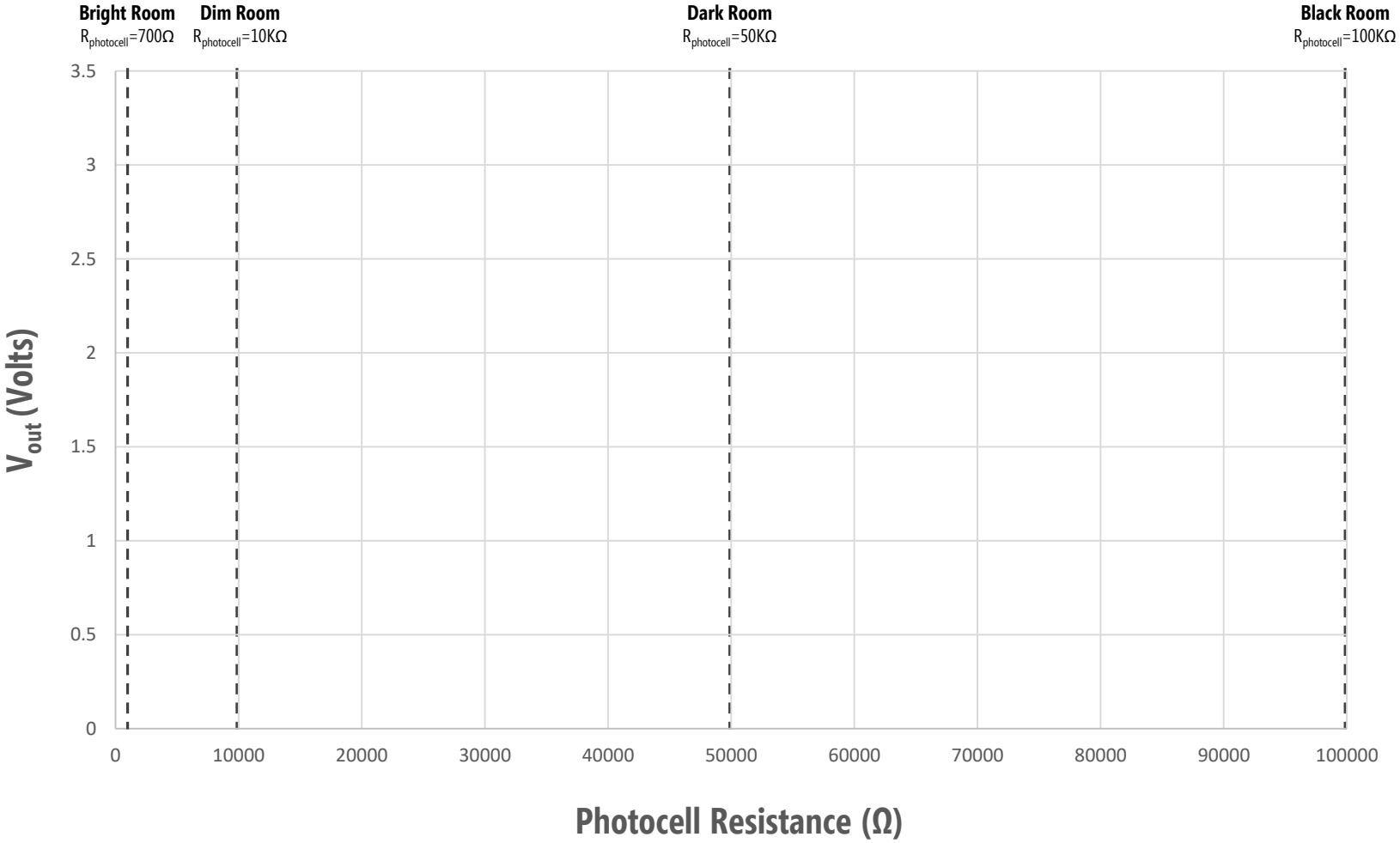
It depends on the context of use (what's the expected resistance range of R1)

USING A VARIABLE RESISTIVE SENSOR

GRAPHING VOUT FOR PHOTOCELL

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

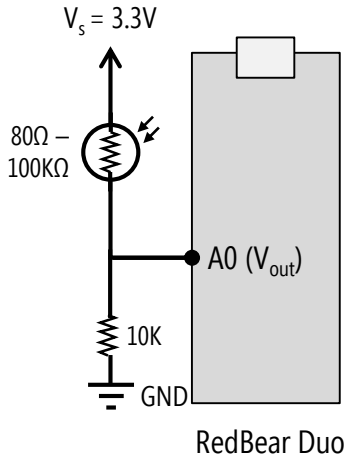


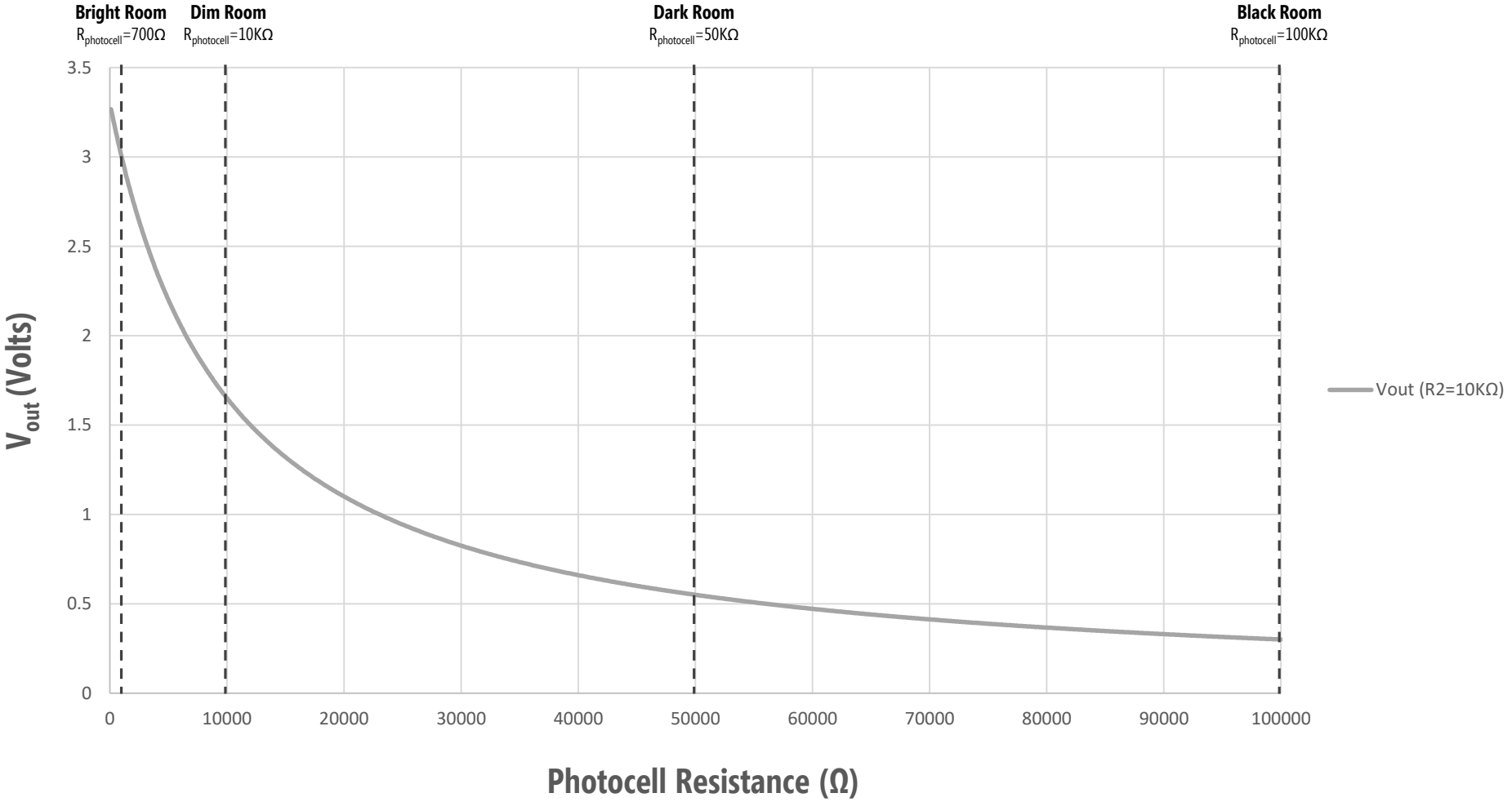


USING A VARIABLE RESISTIVE SENSOR

GRAPHING VOUT FOR PHOTOCELL

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

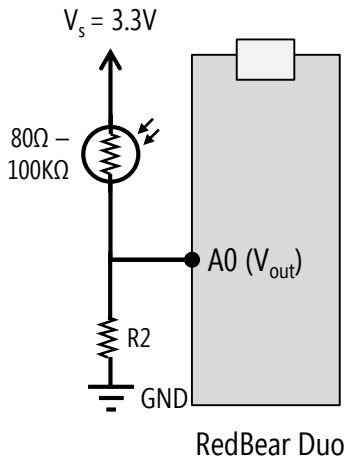


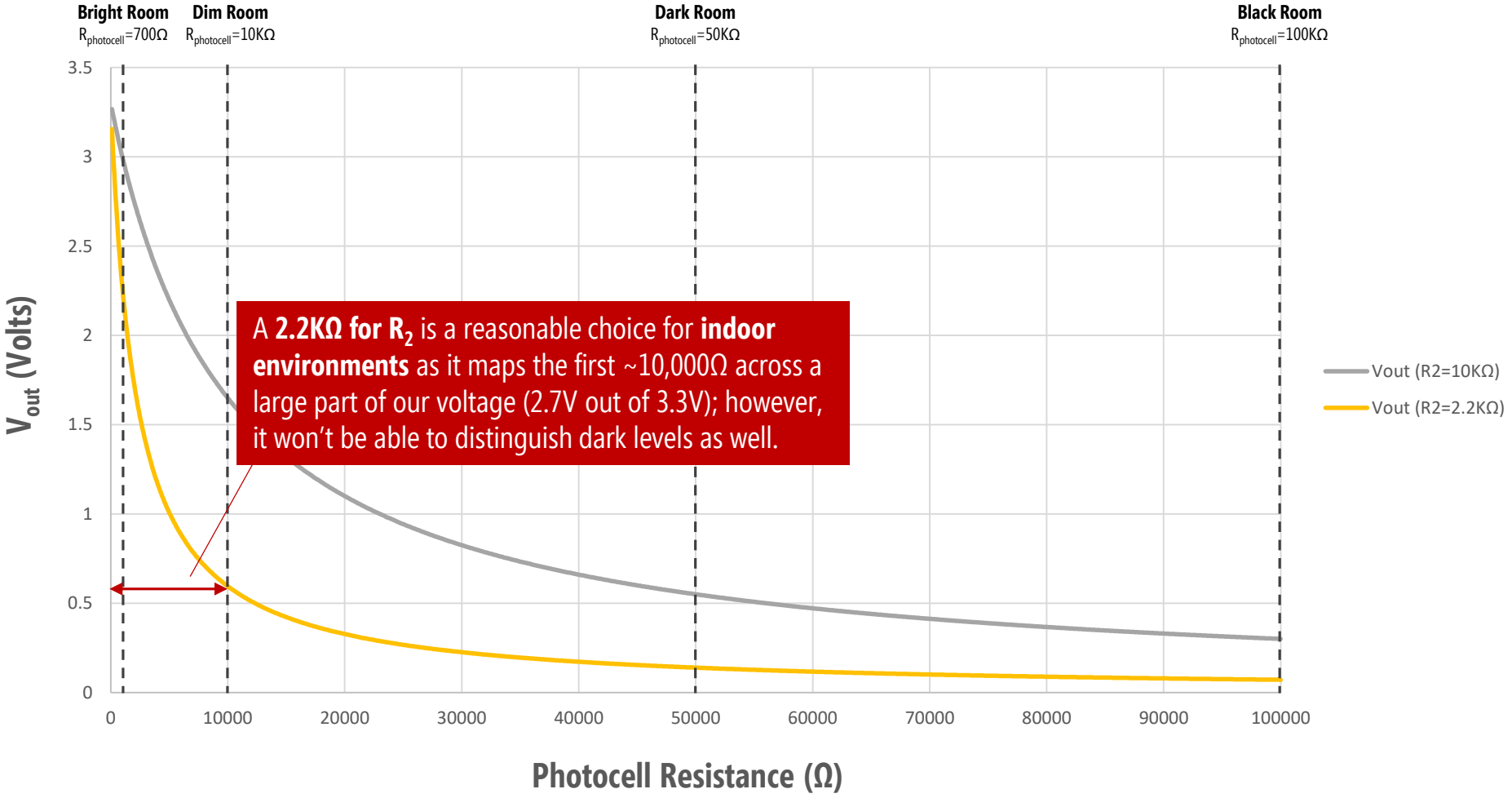


USING A VARIABLE RESISTIVE SENSOR

GRAPHING VOUT FOR PHOTOCELL

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

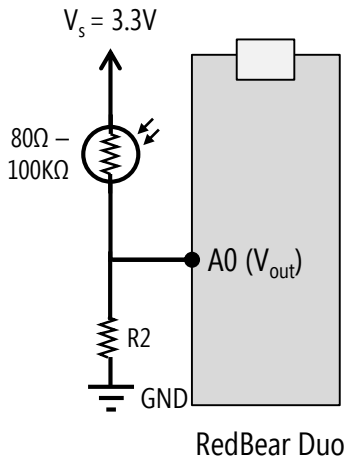


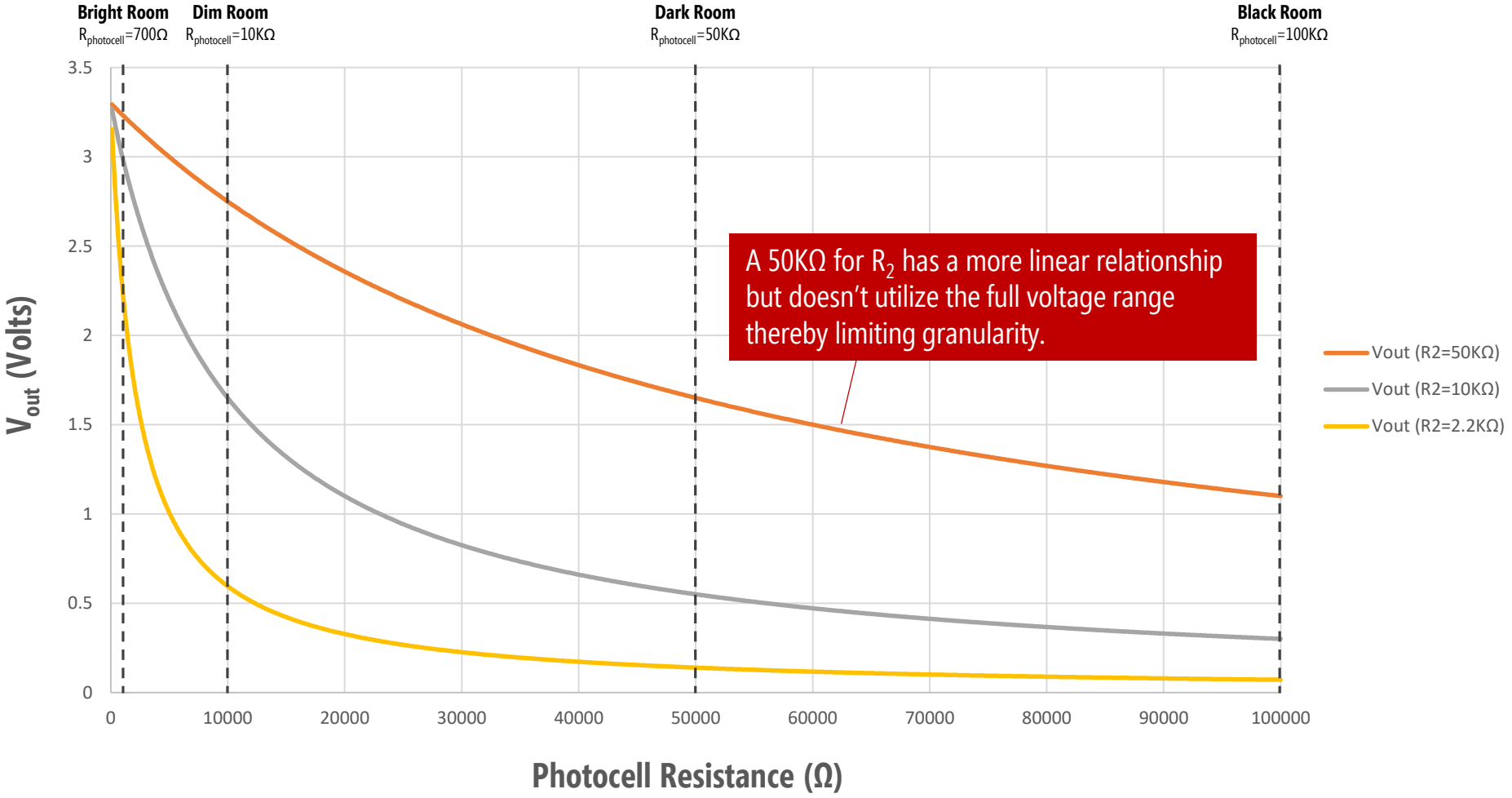


USING A VARIABLE RESISTIVE SENSOR

GRAPHING VOUT FOR PHOTOCELL

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

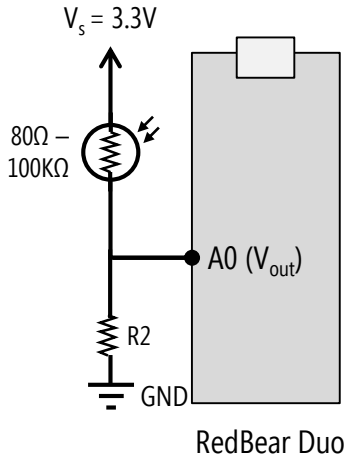


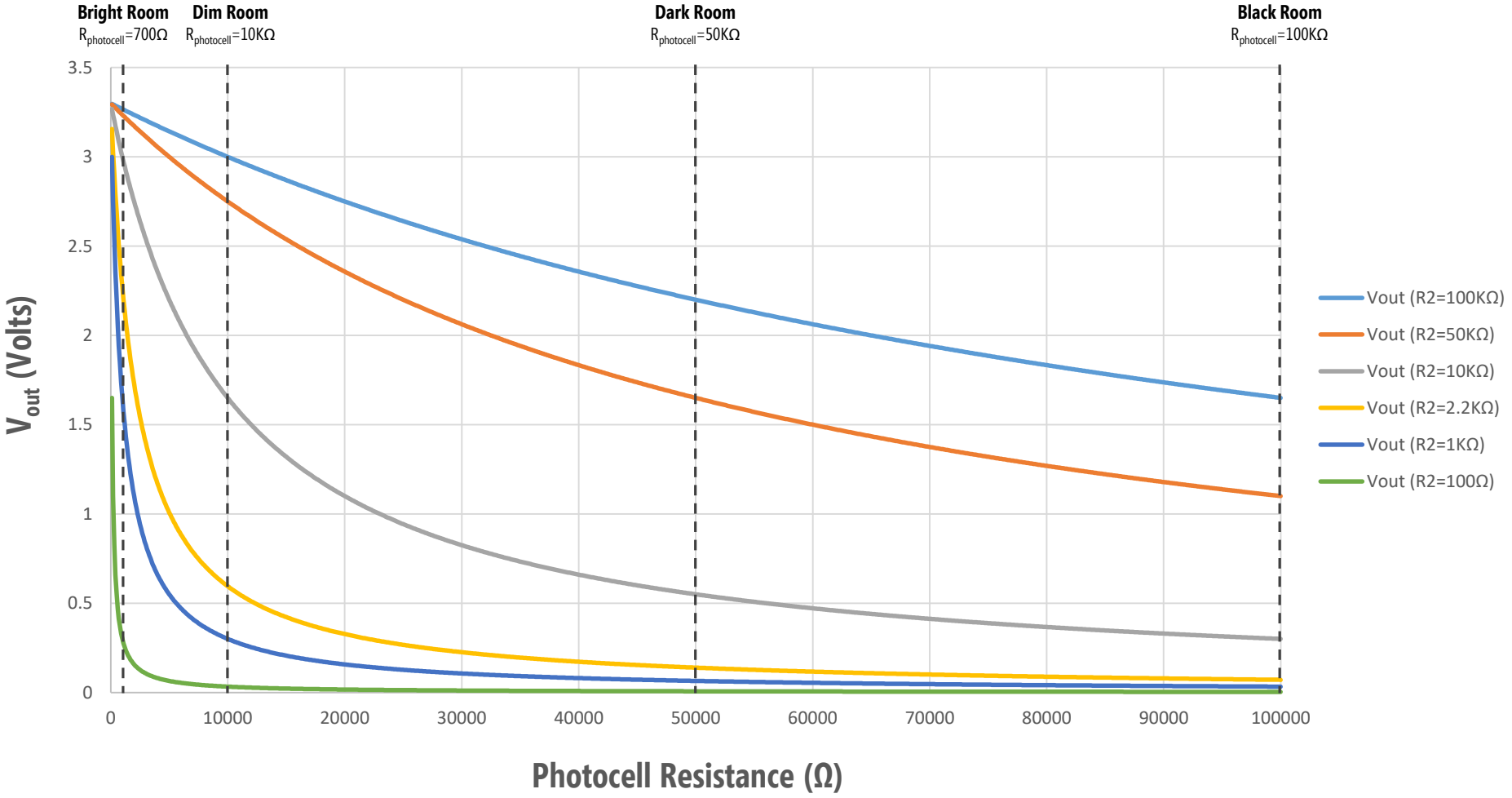


USING A VARIABLE RESISTIVE SENSOR

GRAPHING VOUT FOR PHOTOCELL

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

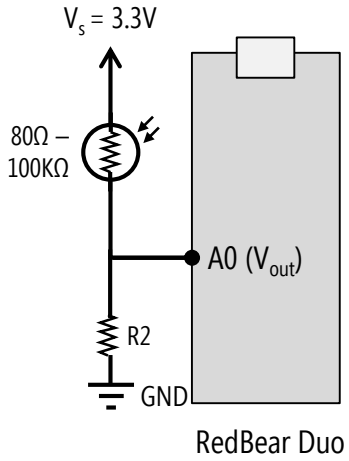




USING A VARIABLE RESISTIVE SENSOR

AXEL BENZ FORMULATION FOR R2

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



First, measure the R_{min} and R_{max} values for your specific environment using a multimeter

Then use these measurements to find the value for R_2

$$R_2 = \sqrt{R_{1_{min}} * R_{1_{max}}}$$

$$R_2 = \sqrt{80\Omega * 10{,}000\Omega}$$

$$R_2 = \sim 900\Omega$$

USING A VARIABLE RESISTIVE SENSOR

EXPERIMENTATION

You'll also experiment with setting thresholds in the mapping function based on empirical observation

RedBearDuoReadPhotocellFadeLED

```
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int PHOTOCELL_INPUT_PIN = A0;

// Set the min and max photocell values (this will be based on
// the brightness of your environment and the size of the voltage-divider
// resistor that you selected).
const int MIN_PHOTOCELL_VAL = 1200; // Photocell reading in dark
const int MAX_PHOTOCELL_VAL = 3700; // Photocell reading in ambient light (tested in my office)

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(PHOTOCELL_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // Read the photo-sensitive resistor value
  int photocellVal = analogRead(PHOTOCELL_INPUT_PIN);

  // Remap the value for output.
  int ledVal = map(photocellVal, MIN_PHOTOCELL_VAL, MAX_PHOTOCELL_VAL, 0, 255);

  // The map function does not constrain output outside of the provided range
  // so, we need to make sure that things are within range for the led
  ledVal = constrain(ledVal, 0, 255);

  // We want to invert the LED (it should be brighter when environment is darker)
  ledVal = 255 - ledVal;

  // Print the raw photocell value and the converted led value (e,g., for Serial Plotter)
  Serial.print(photocellVal);
  Serial.print(",");
  Serial.println(ledVal);
```

Use Serial Plotter to figure out good min and max values for map

EXERCISE: USE A VARIABLE RESISTOR TO FADE LED

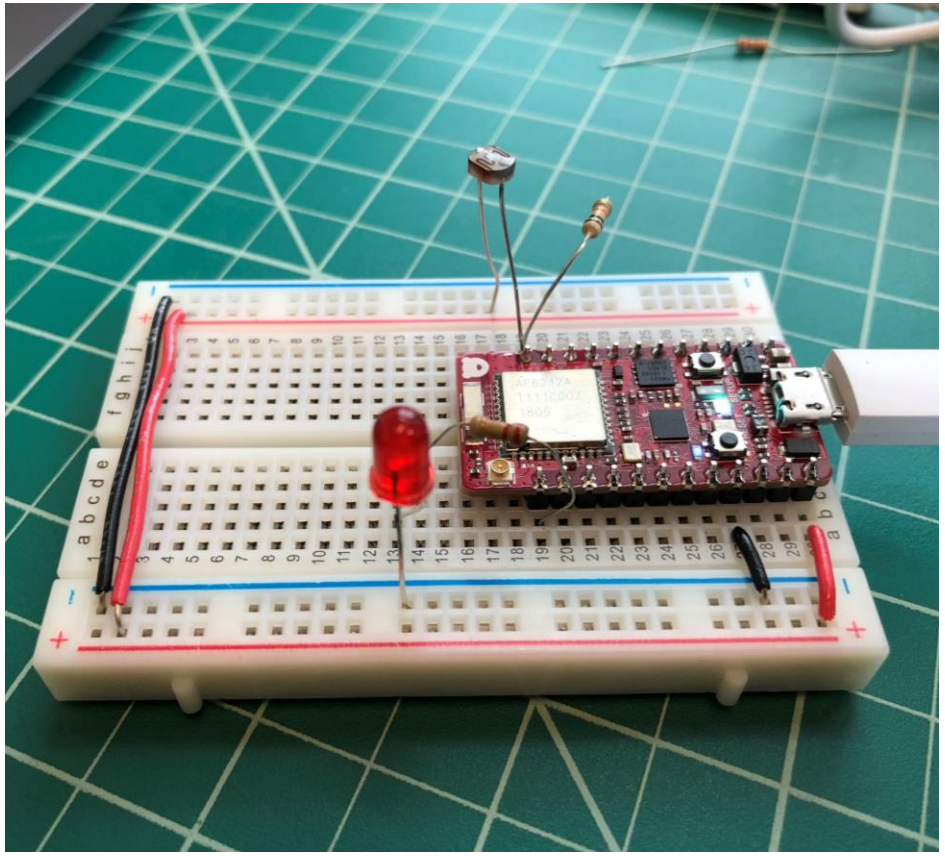
Read variable resistor

Fade LED

Can use photocell or force resistive sensor (easy to swap them)

ANALOG INPUT

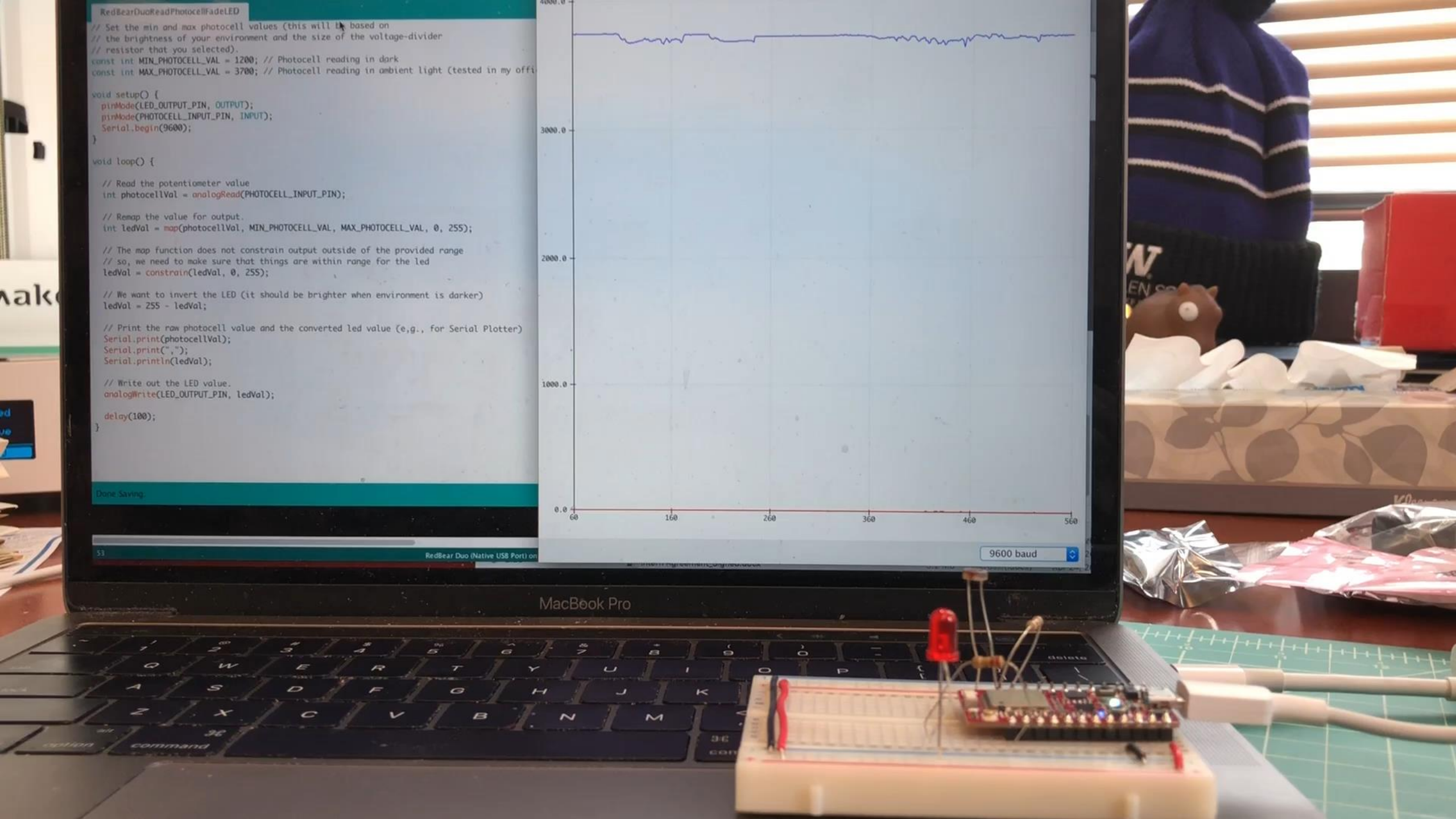
PHOTOCELL INVERSELY FADES LED



https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L06-Arduino/RedBearDuoReadPhotocellFadeLED

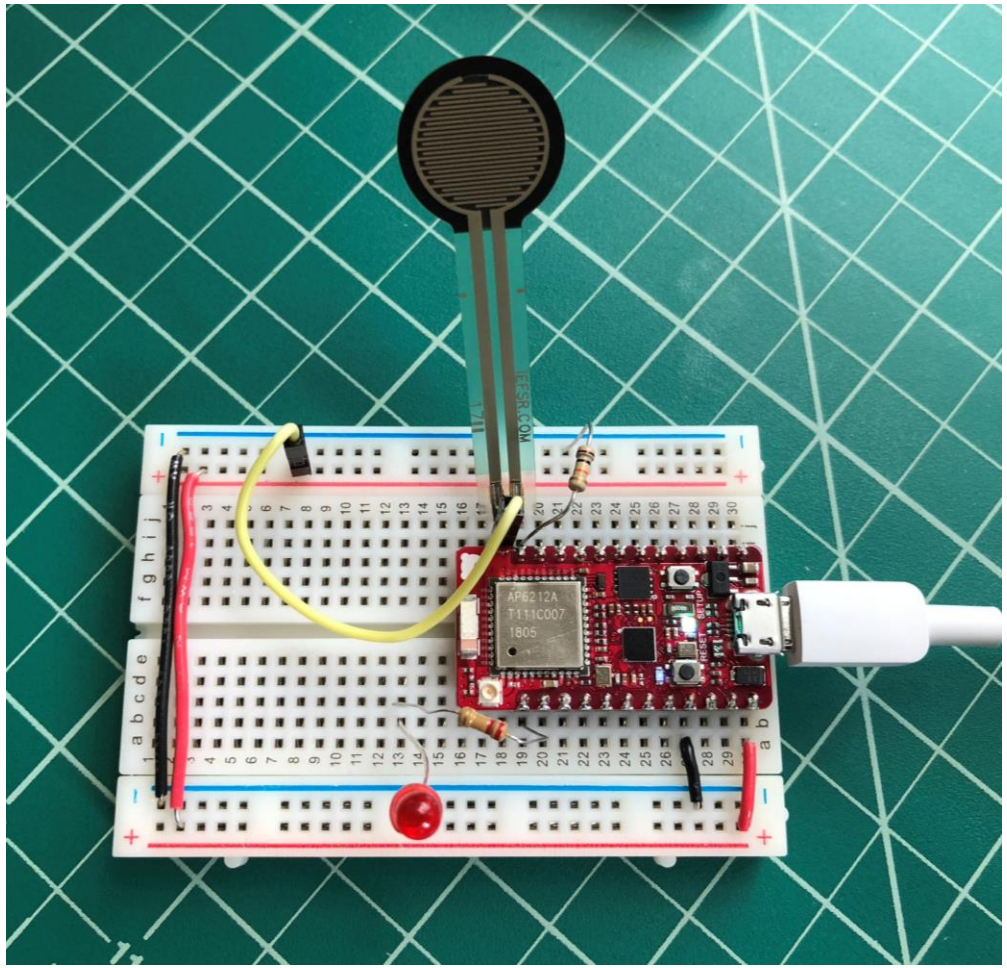
RedBearDuoReadPhotocellFadeLED

```
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int PHOTOCELL_INPUT_PIN = A0;

// Set the min and max photocell values (this will be based on
// the brightness of your environment and the size of the voltage-divider
// resistor that you selected).
const int MIN_PHOTOCELL_VAL = 1200; // Photocell reading in dark
const int MAX_PHOTOCELL_VAL = 3700; // Photocell reading in ambient light (tested in my office)

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(PHOTOCELL_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // Read the photo-sensitive resistor value
  int photocellVal = analogRead(PHOTOCELL_INPUT_PIN);

  // Remap the value for output.
  int ledVal = map(photocellVal, MIN_PHOTOCELL_VAL, MAX_PHOTOCELL_VAL, 0, 255);

  // The map function does not constrain output outside of the provided range
  // so, we need to make sure that things are within range for the led
  ledVal = constrain(ledVal, 0, 255);

  // We want to invert the LED (it should be brighter when environment is darker)
  ledVal = 255 - ledVal;

  // Print the raw photocell value and the converted led value (e,g., for Serial Plotter)
  Serial.print(photocellVal);
  Serial.print(",");
  Serial.println(ledVal);

  // Write out the LED value.
  analogWrite(LED_OUTPUT_PIN, ledVal);

  delay(100);
}
```


RedBearDuoReadPhotocellFadeLED
// Set the min and max photocell values (this will be based on
// the brightness of your environment and the size of the voltage-divider
// resistor that you selected).
const int MIN_PHOTOCELL_VAL = 1200; // Photocell reading in dark
const int MAX_PHOTOCELL_VAL = 3700; // Photocell reading in ambient light (tested in my offi
void setup() {
pinMode(LED_OUTPUT_PIN, OUTPUT);
pinMode(PHOTOCELL_INPUT_PIN, INPUT);
Serial.begin(9600);
}
void loop() {
// Read the potentiometer value
int photocellVal = analogRead(PHOTOCELL_INPUT_PIN);
// Remap the value for output.
int ledVal = map(photocellVal, MIN_PHOTOCELL_VAL, MAX_PHOTOCELL_VAL, 0, 255);
// The map function does not constrain output outside of the provided range
// so, we need to make sure that things are within range for the led
ledVal = constrain(ledVal, 0, 255);
// We want to invert the LED (it should be brighter when environment is darker)
ledVal = 255 - ledVal;
// Print the raw photocell value and the converted led value (e,g., for Serial Plotter)
Serial.print(photocellVal);
Serial.print(",");
Serial.println(ledVal);
// Write out the LED value.
analogWrite(LED_OUTPUT_PIN, ledVal);
delay(100);
}
Done Saving.
53
RedBear Duo (Native USB Port) on
4000.0
3000.0
2000.0
1000.0
0.0
60
160
260
360
460
560
9600 baud
MacBook Pro

ANALOG INPUT

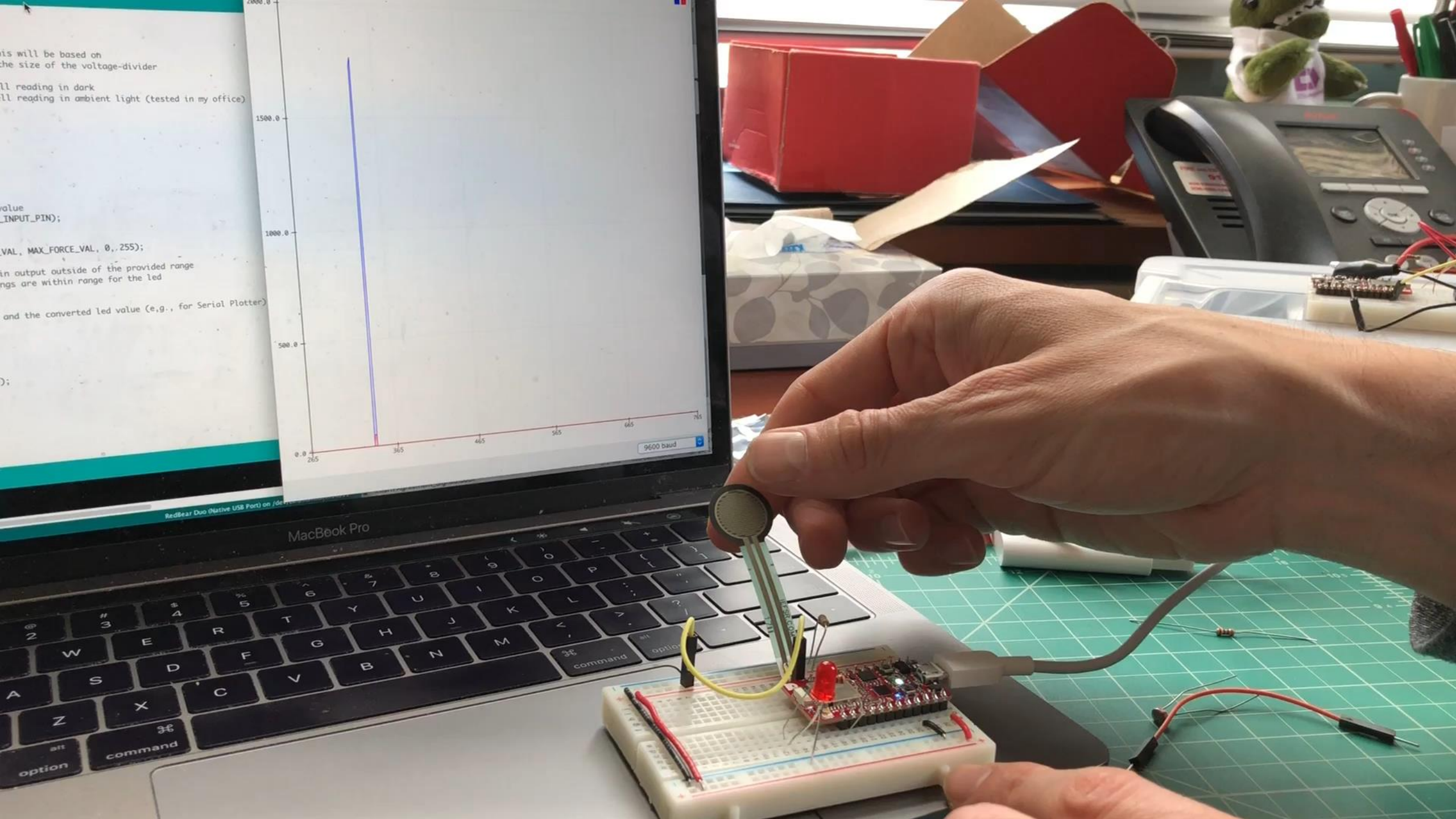
FSR FADES LED



https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L06-Arduino/RedBearDuoReadForceSensitiveResistorLED

RedBearDuoReadForceSensitiveResistorLED §

```
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int FORCESENSOR_INPUT_PIN = A0;

// Set the min and max force sensitive resistor values
const int MIN_FORCE_VAL = 0;
const int MAX_FORCE_VAL = 4000;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(FORCESENSOR_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // Read the force-sensitive resistor value
  int forceVal = analogRead(FORCESENSOR_INPUT_PIN);

  // Remap the value for output.
  int ledVal = map(forceVal, MIN_FORCE_VAL, MAX_FORCE_VAL, 0, 255);

  // The map function does not constrain output outside of the provided range
  // so, we need to make sure that things are within range for the led
  ledVal = constrain(ledVal, 0, 255);

  // Print the raw force sensor value and the converted led value so we can
  // see them in Serial Monitor and Serial Plotter
  Serial.print(forceVal);
  Serial.print(",");
  Serial.println(ledVal);

  // Write out the LED value.
  analogWrite(LED_OUTPUT_PIN, ledVal);

  delay(100);
}
```


is will be based on
he size of the voltage-divider
ll reading in dark
ll reading in ambient light (tested in my office)
alue
_INPUT_PIN);
VAL, MAX_FORCE_VAL, 0, 255);
in output outside of the provided range
ngs are within range for the led
and the converted led value (e,g., for Serial Plotter)
);
2000.0
1500.0
1000.0
500.0
0.0
265
365
465
565
665
765
9600 baud
RedBear Duo (Native USB Port) on /de
MacBook Pro
command
option

WORKING WITH SENSORS: LEARNING GOALS

How to use variable **resistive sensors,** including **voltage dividers,** basic **signal smoothing**

Putting it together: **making a simple theremin**

How to use **buttons**, including **pull-up** and **pull-down resistors**, and **debouncing**

Working with **interrupts**

Let’s extend this to build a **theremin-like** instrument.

THEREMIN

ZELDA'S LULLABY ON THEREMIN

Source: Theremin Hero, https://youtu.be/H-iUZD8wfjs


CLASSICALLY
TRAINED

To do this, we need to use the **piezoelectric buzzers** & the **tone()** function

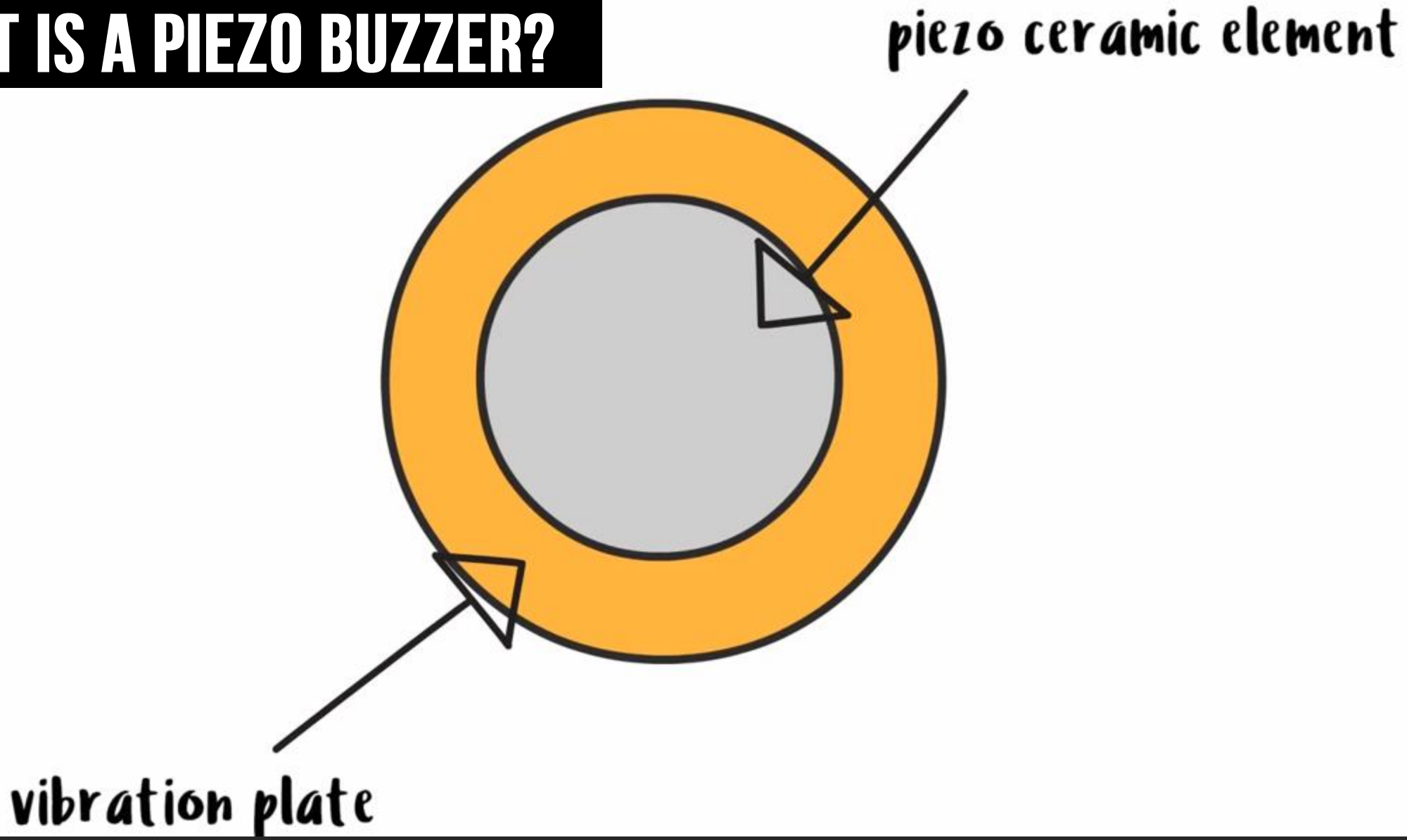
USING PIEZO BUZZER

PIEZO ELECTRIC BUZZERS

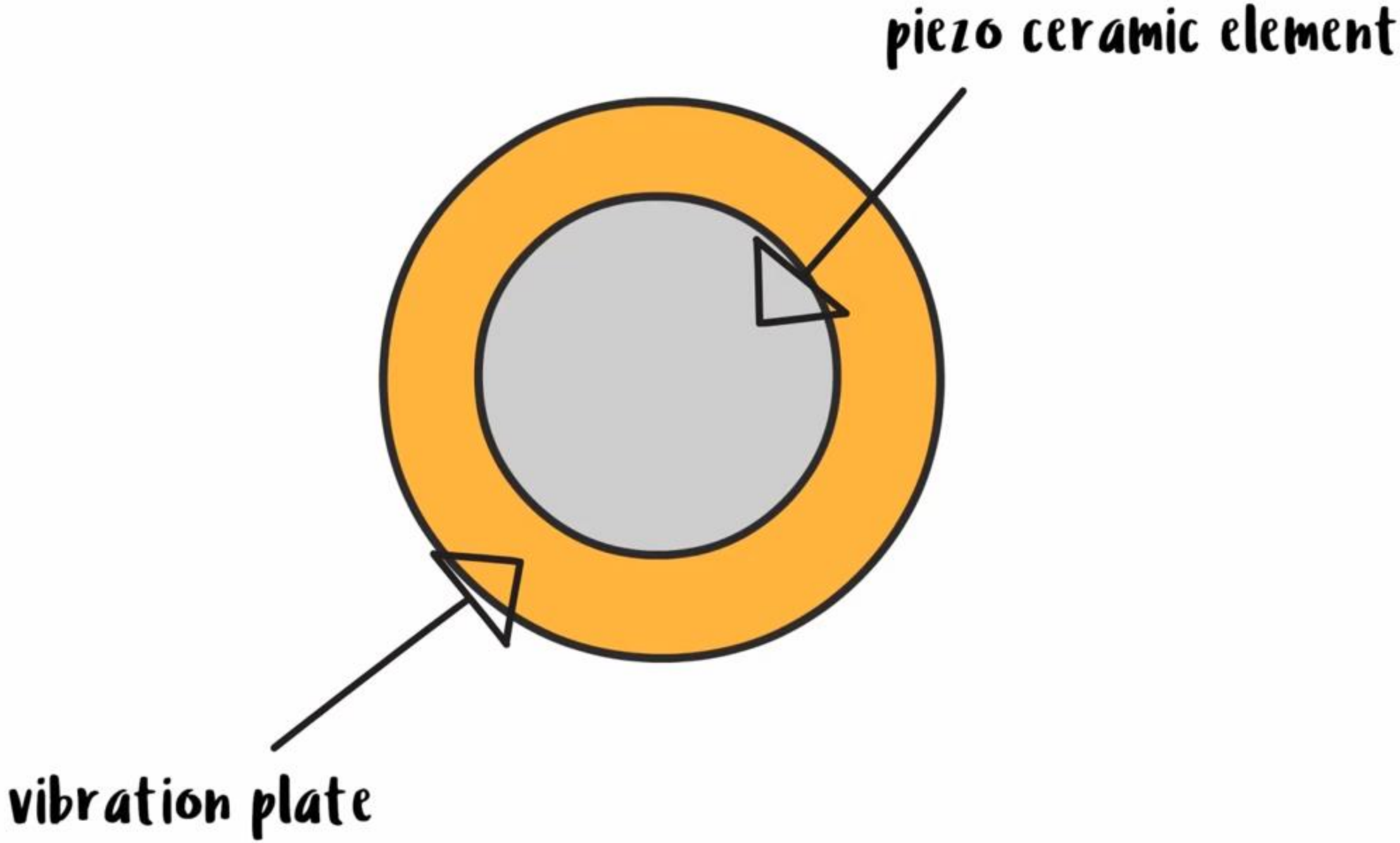
Piezoelectric elements produce electric charge in response to mechanical stress. They can also work in the other direction: an applied current can distort material.

PIEZO BUZZER

WHAT IS A PIEZO BUZZER?



Source: MakeCrate, https://youtu.be/77h1JhD9Syw



vibration disc.

USING PIEZO BUZZER

GENERATE FREQUENCY: TONE()

Generates a square wave of the specified frequency on a pin. The duty cycle is fixed at 50%.

Syntax

```
tone(pin, frequency)
tone(pin, frequency, duration)
```

Parameters

`pin:` the pin on which to generate the tone

`frequency:` the frequency of the tone in hertz - unsigned int

`duration:` the duration of the tone in milliseconds (optional) - unsigned long

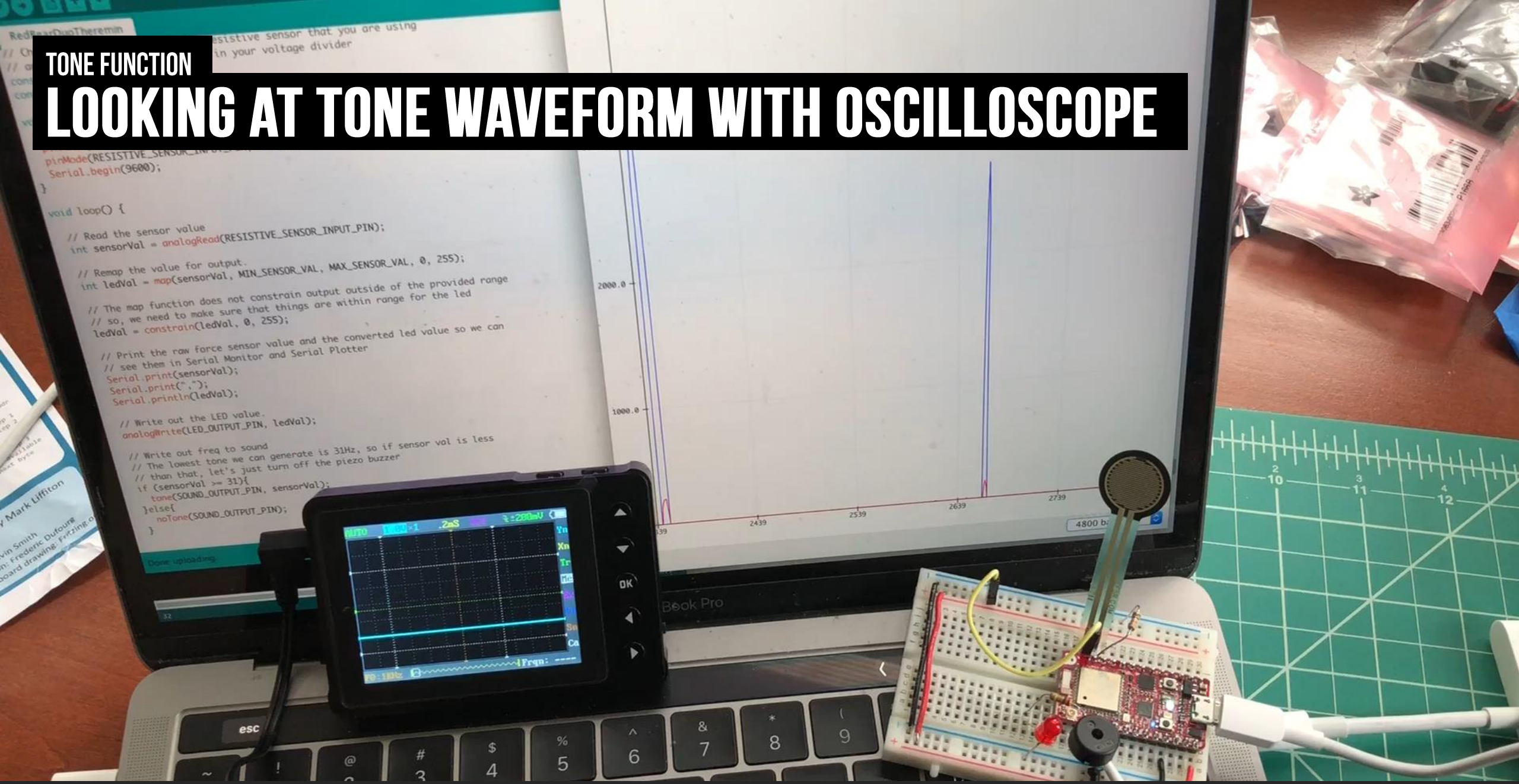
Returns

Nothing

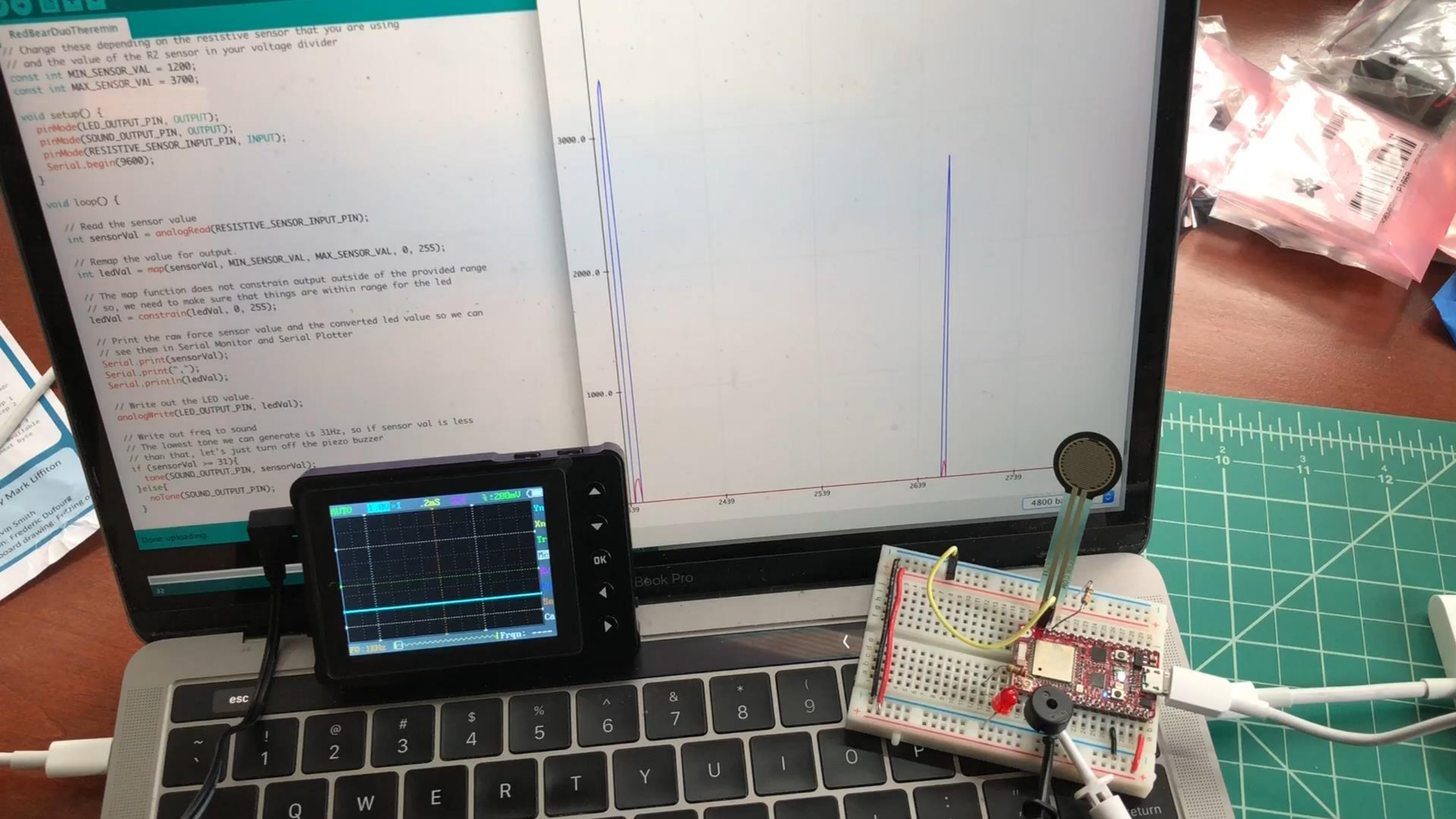
https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/

TONE FUNCTION

LOOKING AT TONE WAVEFORM WITH OSCILLOSCOPE



Video source: Jon Froehlich


RedBearDuoTheremin
// Change these depending on the resistive sensor that you are using
// and the value of the R2 sensor in your voltage divider
const int MIN_SENSOR_VAL = 1200;
const int MAX_SENSOR_VAL = 3700;
void setup() {
pinMode(LED_OUTPUT_PIN, OUTPUT);
pinMode(SOUND_OUTPUT_PIN, OUTPUT);
pinMode(RESISTIVE_SENSOR_INPUT_PIN, INPUT);
Serial.begin(9600);
}
void loop() {
// Read the sensor value
int sensorVal = analogRead(RESISTIVE_SENSOR_INPUT_PIN);
// Remap the value for output.
int ledVal = map(sensorVal, MIN_SENSOR_VAL, MAX_SENSOR_VAL, 0, 255);
// The map function does not constrain output outside of the provided range
// so, we need to make sure that things are within range for the led
ledVal = constrain(ledVal, 0, 255);
// Print the raw force sensor value and the converted led value so we can
// see them in Serial Monitor and Serial Plotter
Serial.print(sensorVal);
Serial.print(",");
Serial.println(ledVal);
// Write out the LED value.
analogWrite(LED_OUTPUT_PIN, ledVal);
// Write out freq to sound
// The lowest tone we can generate is 31Hz, so if sensor val is less
// than that, let's just turn off the piezo buzzer
if (sensorVal >= 31){
tone(SOUND_OUTPUT_PIN, sensorVal);
}else{
noTone(SOUND_OUTPUT_PIN);
}
Done uploading.
3000.0
2000.0
1000.0
2439
2539
2639
2739
4800 ba
OK
Book Pro
esc

USING PIEZO BUZZER

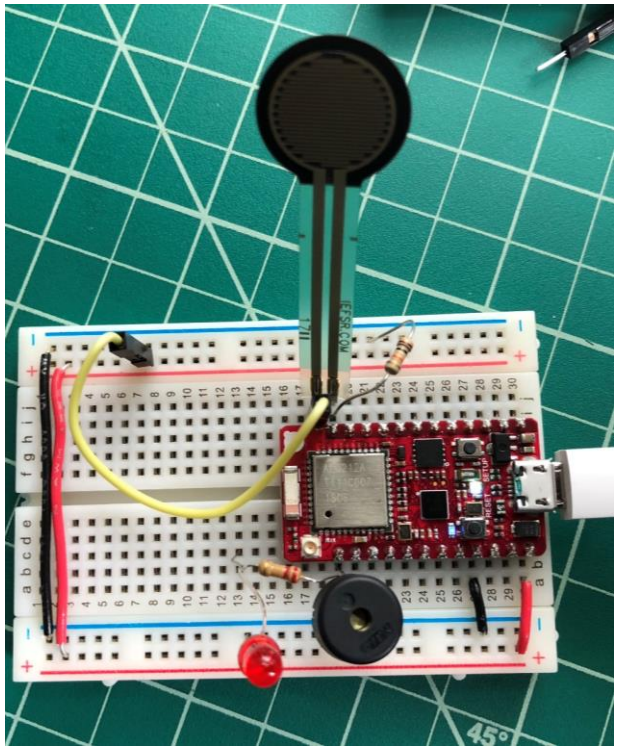
EXERCISE: CREATE THEREMIN!

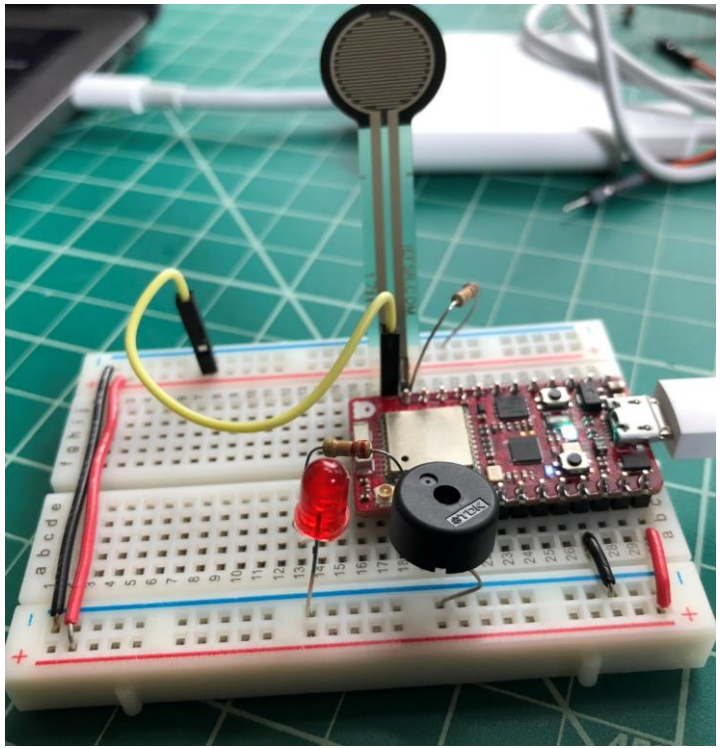
Add a piezoelectric buzzer to your circuit

Output tone proportional to variable resistor

USING PIEZO BUZZER

EXAMPLE SOLUTION





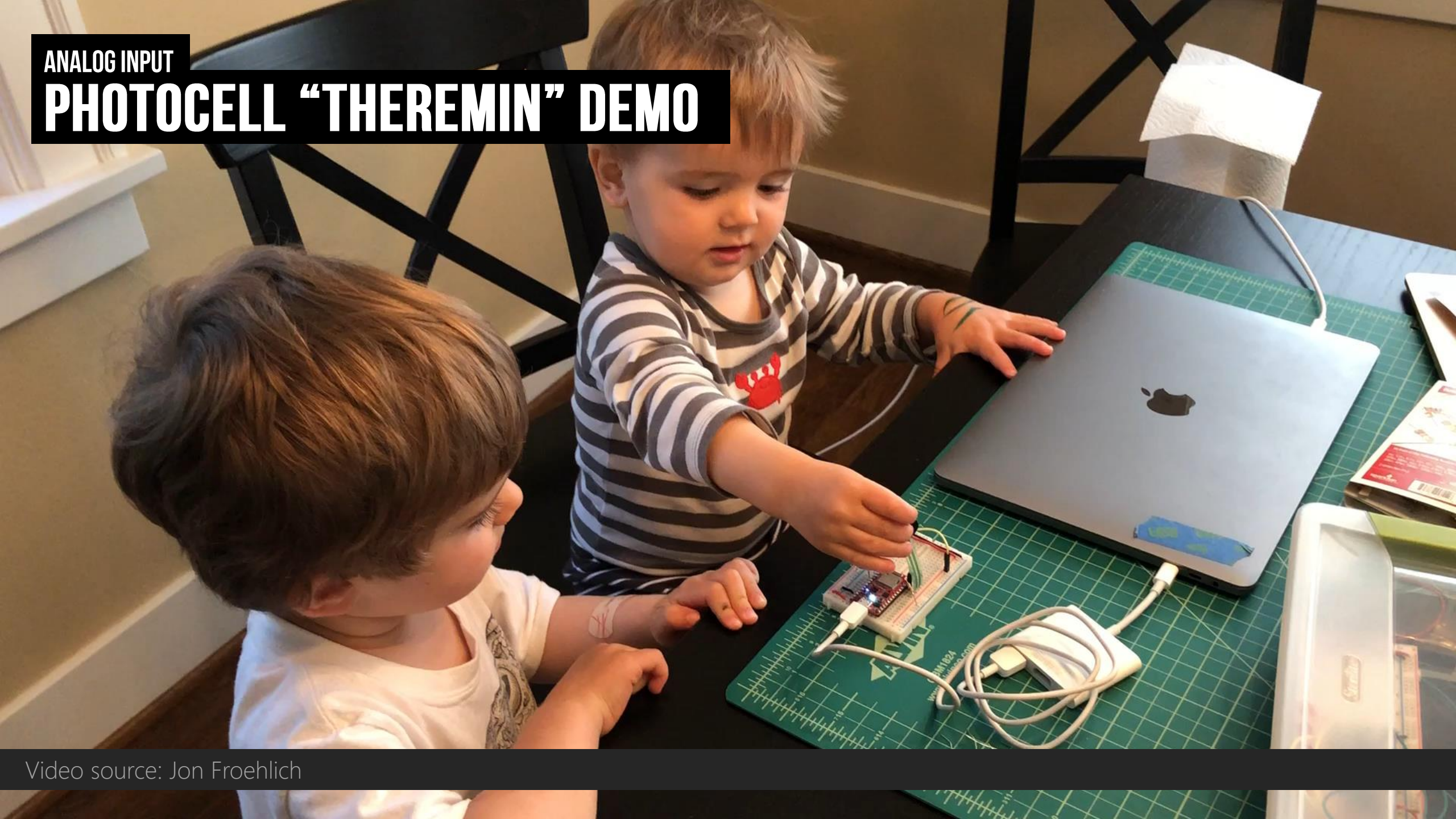
https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L06-Arduino/RedBearDuoTheremin

RedBearDuoTheremin

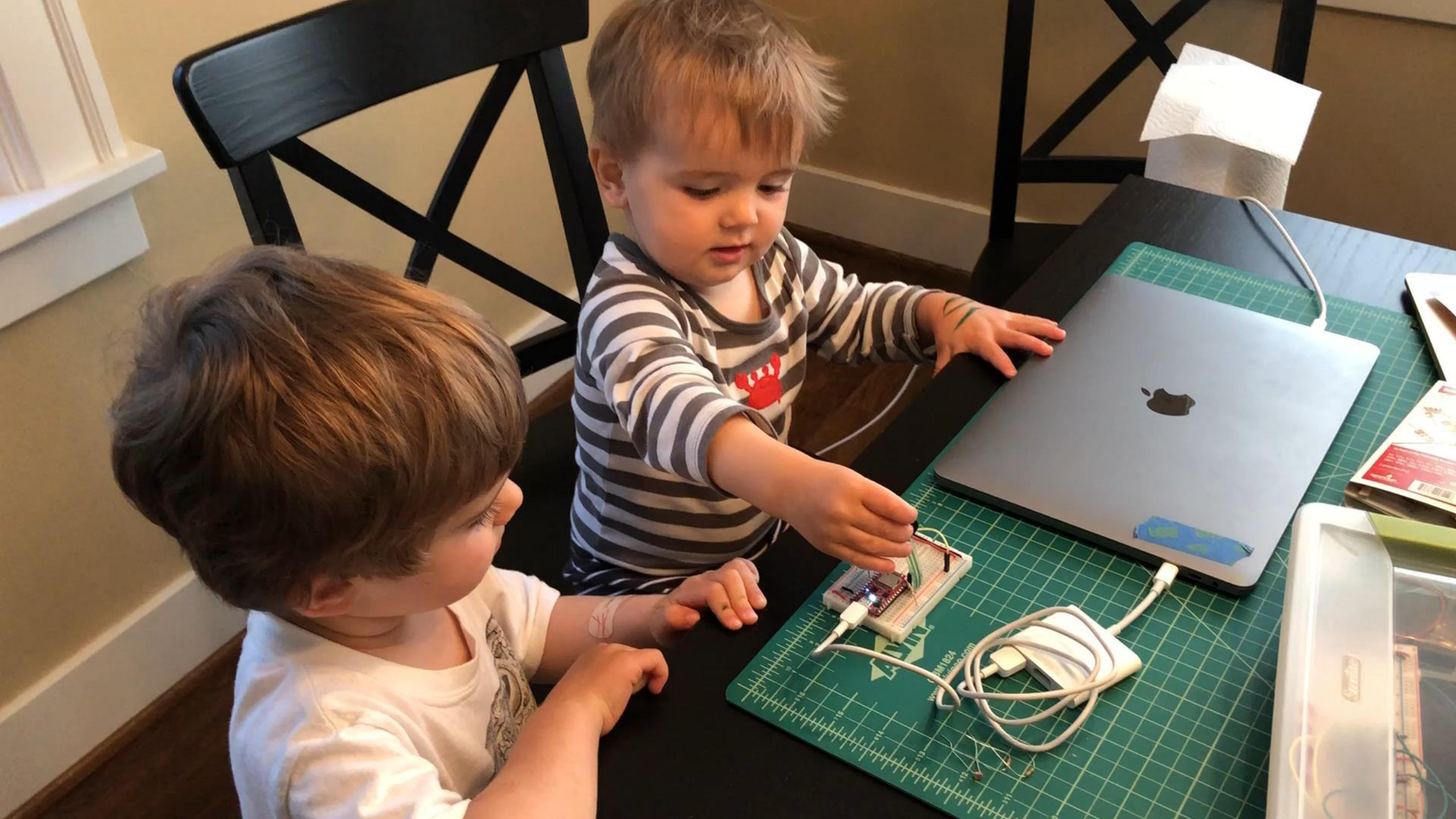
```
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int SOUND_OUTPUT_PIN = D1;
const int RESISTIVE_SENSOR_INPUT_PIN = A0;

// Change these depending on the resistive sensor that you are using
// and the value of the R2 sensor in your voltage divider
const int MIN_SENSOR_VAL = 1200;
const int MAX_SENSOR_VAL = 3700;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(SOUND_OUTPUT_PIN, OUTPUT);
  pinMode(RESISTIVE_SENSOR_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // Read the sensor value
  int sensorVal = analogRead(RESISTIVE_SENSOR_INPUT_PIN);

  // Remap the value for output.
  int ledVal = map(sensorVal, MIN_SENSOR_VAL, MAX_SENSOR_VAL, 0, 255);

  // The map function does not constrain output outside of the provided range
  // so, we need to make sure that things are within range for the led
  ledVal = constrain(ledVal, 0, 255);

  // Print the raw force sensor value and the converted led value so we can
  // see them in Serial Monitor and Serial Plotter
  Serial.print(sensorVal);
  Serial.print(",");
  Serial.println(ledVal);

  // Write out the LED value.
  analogWrite(LED_OUTPUT_PIN, ledVal);

  // Write out freq to sound
  // The lowest tone we can generate is 31Hz, so if sensor val is less
  // than that, let's just turn off the piezo buzzer
  if (sensorVal >= 31){
    tone(SOUND_OUTPUT_PIN, sensorVal);
  }else{
    noTone(SOUND_OUTPUT_PIN);
  }

  delay(100);
}
```

ANALOG INPUT

PHOTOCELL “THEREMIN” DEMO

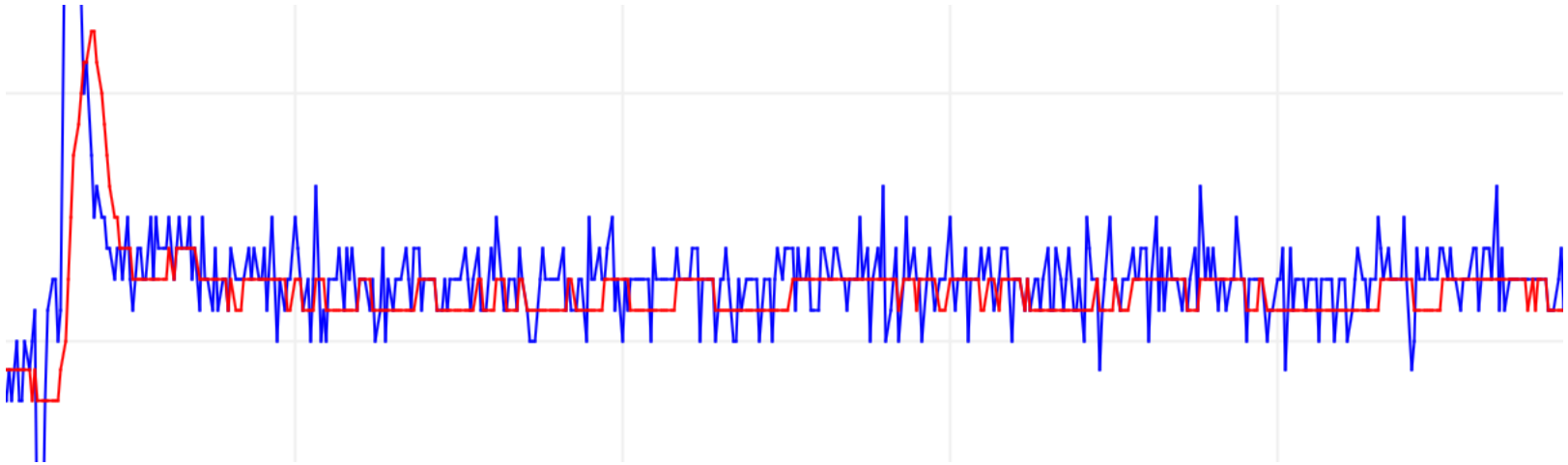


Video source: Jon Froehlich



One last thing before we move on: analog inputs tend to be **noisy**. To fix this, even a simple smoothing filter like a moving average will work nicely.

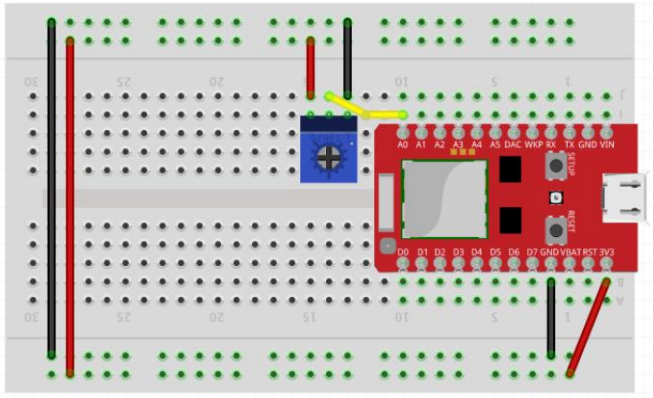
(Note: You also get "basic" smoothing when you compress values from a large range like 0-4096 to a small integer range like 0-255)

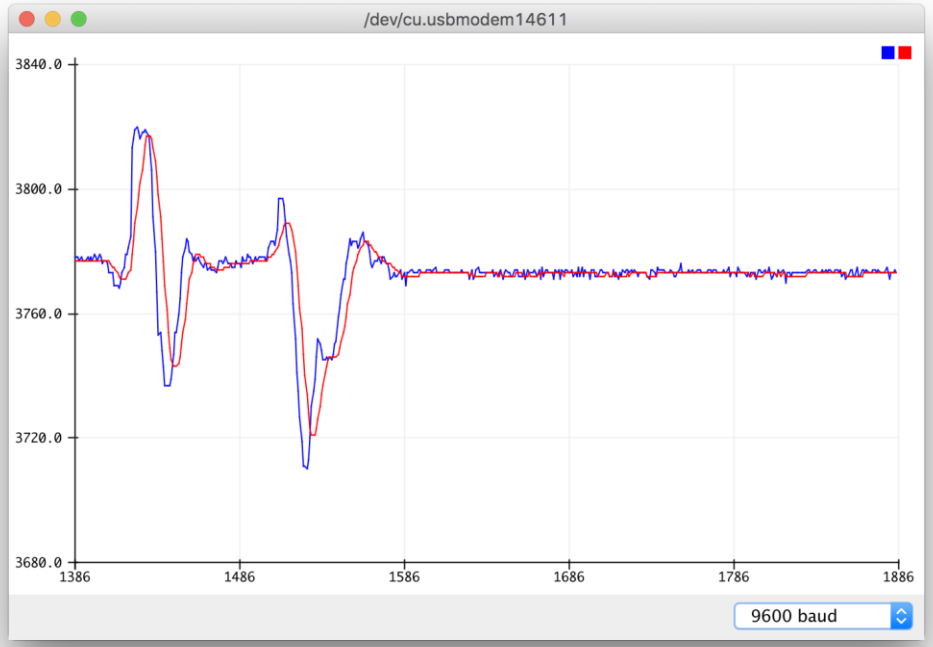


ANALOG INPUT

SMOOTHING THE SIGNAL

Even a simple circuit with a potentiometer can be noisy





Basic sliding window mean filter

RedBearDuoSmoothing

```
SYSTEM_MODE(MANUAL);

// Define the smoothing window size
const int SMOOTHING_WINDOW_SIZE = 10;
const int ANALOG_INPUT_PIN = A0;

int _readings[SMOOTHING_WINDOW_SIZE];      // the readings from the analog input
int _readIndex = 0;              // the index of the current reading
int _total = 0;                  // the running total
int _average = 0;                // the average


void setup() {

  pinMode(ANALOG_INPUT_PIN, INPUT);

  // initialize serial communication with computer
  Serial.begin(9600);

  // initialize all the readings to 0
  for (int i = 0; i < SMOOTHING_WINDOW_SIZE; i++) {
    _readings[i] = 0;
  }
}

void loop() {
  // subtract the last reading
  _total = _total - _readings[_readIndex];

  // read from the sensor
  int curReading = analogRead(ANALOG_INPUT_PIN);
  _readings[_readIndex] = curReading;

  // add the reading to the total
  _total = _total + _readings[_readIndex];

  // advance to the next position in the array
  _readIndex = _readIndex + 1;

  // if we're at the end of the array...
  if (_readIndex >= SMOOTHING_WINDOW_SIZE) {
    // ...wrap around to the beginning:
    _readIndex = 0;
  }

  // calculate the average
  _average = _total / SMOOTHING_WINDOW_SIZE;

  // send it to the computer as ASCII digits
  Serial.print(curReading);
  Serial.print(",");
  Serial.println(_average);

  delay(50); // delay in between reads for stability
}
```

https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L06-Arduino/RedBearDuoSmoothing

WORKING WITH SENSORS: LEARNING GOALS

How to use variable **resistive sensors,** including **voltage dividers,** basic **signal smoothing**

Putting it together: **making a simple theremin**

How to use **buttons**, including **pull-up** and **pull-down resistors**, and **debouncing**

Working with **interrupts**

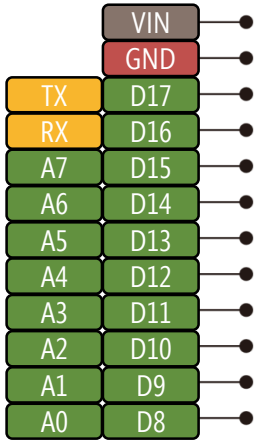
INPUT

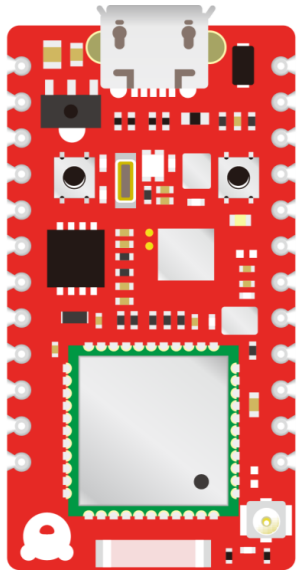
DIGITAL INPUT

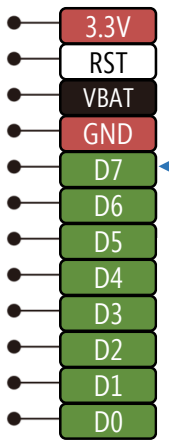
We went over this in the last lecture (L05-PhysicalComputing2-Arduino.pptx) but here's a quick review

Analog Input on A7

Reads in any any value between 0V or 3.3V using analogRead function. In this case, the value of a photocell







Digital Input on D7

Reads in a digital input signal (anything below 1.65V converted to LOW, anything above 1.65V converted to HIGH). In this case, the value of switch.

Analog Output on D9

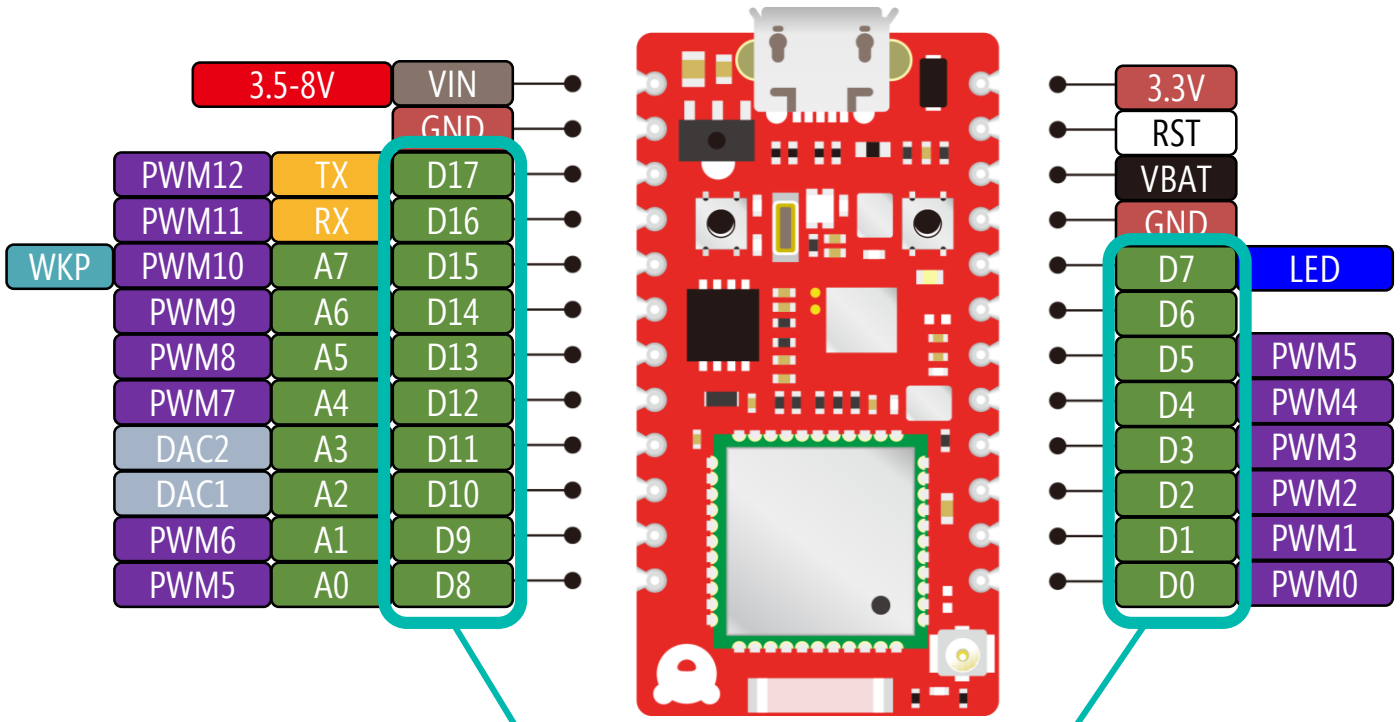
Writes out any value between 0V or 3.3V using analogWrite function. In this case, vibrating a motor (where strength of vibration proportional to voltage).

Digital Output on D0

Writes out 0V or 3.3V using digitalWrite function. In this case, turning on and off an LED.

DIGITAL INPUT

ANY OF THE D PINS CAN BE SET FOR DIGITAL INPUT OR OUTPUT



Legend

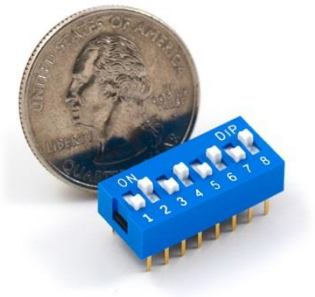
AX	Analog Input Pin
DX	Digital Input/output Pin
PWM	Supports Analog Out via Pulse Width Modulation

Any of the D pins can be set for input or output in setup() using pinMode()

DIGITAL INPUT

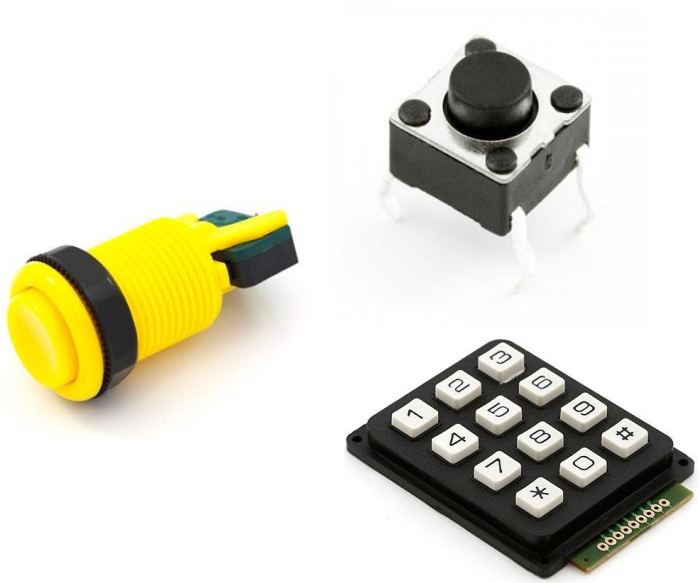
SWITCHES

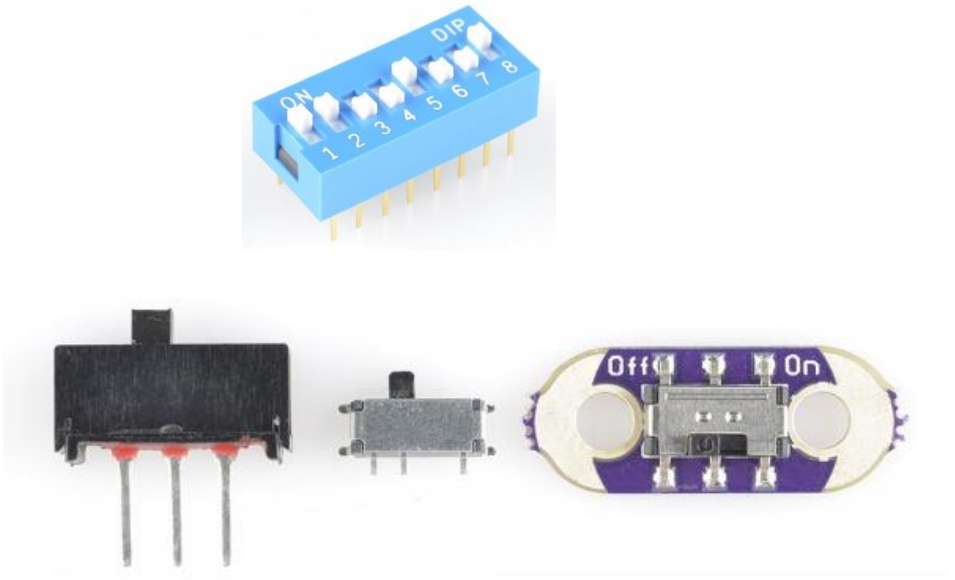
Switches are a common form of digital input. There is variety of switches from toggle to rotary to DIP to push-button to DIP to rocker. Switches differ in how they are actuated and how many circuits they can control



DIGITAL INPUT

SWITCHES: MOMENTARY VS. MAINTAINED





Momentary: remain active only as long as they are actuated (*e.g.*, keys on a keyboard, buttons)

Maintained: Stay in one state until actuated into a new one (*e.g.*, light switches, DIP switches)

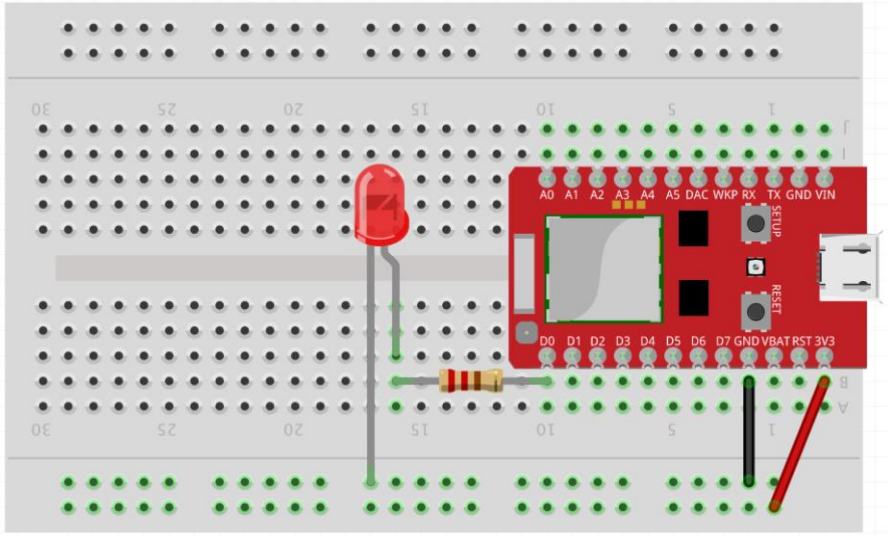
[source: https://learn.sparkfun.com/tutorials/switch-basics/all]

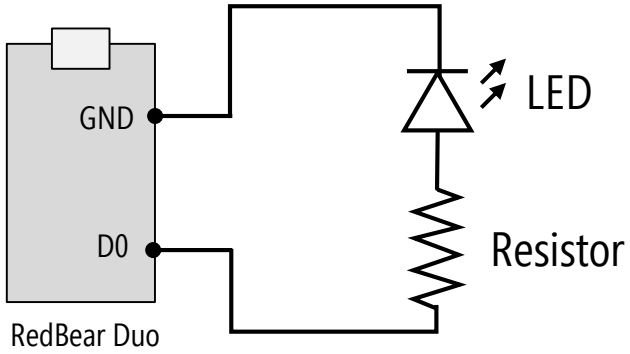
Let’s design a circuit& write code to
turn **on** and **off** an **LED** with a **button**.

DIGITAL INPUT

TURN ON/OFF LED WITH A BUTTON

Basic Circuit: Control LED via the D0 pin

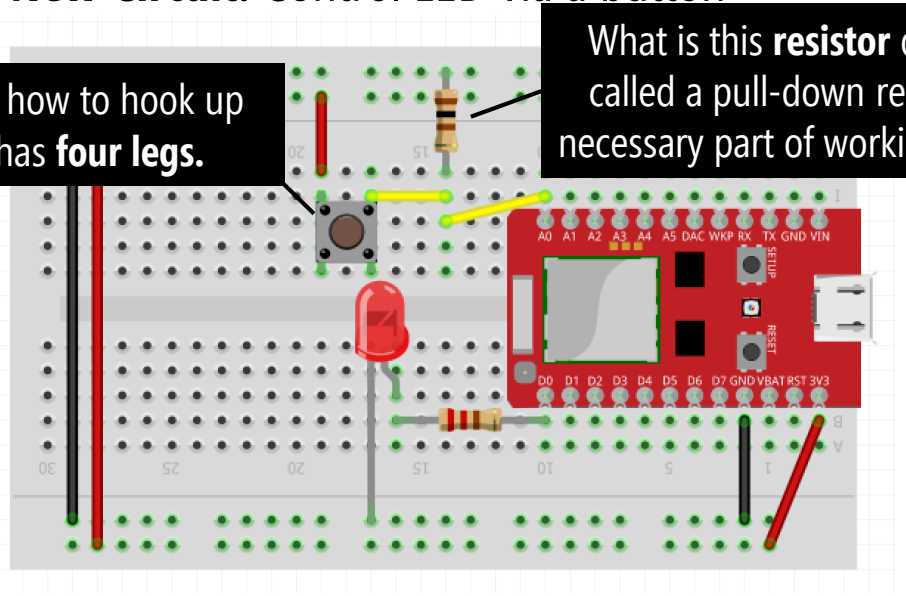


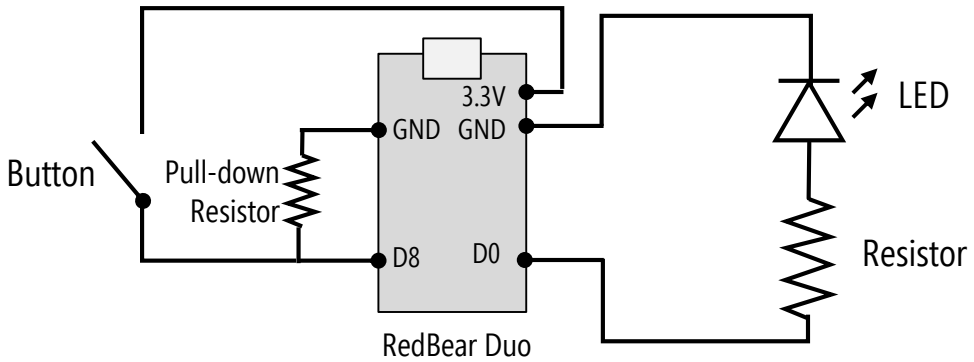


New Circuit: Control LED via a button

How do we know how to hook up this button? It has **four legs.**

What is this **resistor** doing here? It's called a pull-down resistor and it's a necessary part of working with switches.





HOOKING UP BUTTON: CONSULT THE DATASHEET

OMRON

Tactile Switch
B3F

Miniature, Space-Saving Tactile Switch Provides Long Service Life and Easy Mounting

- Extended mechanical/electrical service life: 10 x 10^6 operations for 12 x 12 mm type and 1 x 10^6 operations for the 6 x 6 mm type
- Ideal for applications such as audio, office and communications equipment, measuring instruments, TVs, VCRs, etc.
- Taped radial type, vertical type and high force types are available as series versions
- Flux-tight base structure allows automatic soldering of the tactile switches onto a PC board
- Gold plated models available for increased contact reliability, resistance to corrosive gas and insulation fault prevention for ion migration in harsh environments
- RoHS Compliant

Ordering Information

■ **B3F-1□□□, B3F-3□□□**

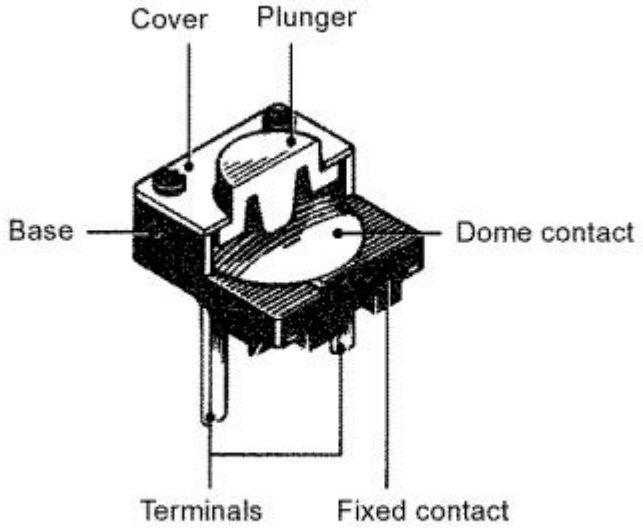
6 x 6 mm type

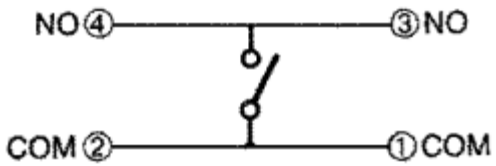
Type	Plunger	Switch height x pitch	Operating force	Model			
				Without ground terminal		With ground terminal	
				Bags	Sticks*	Bags	Sticks*
Standard	Flat	4.3 x 6.5 mm	General-purpose: 100 g	B3F-1000	B3F-1000S	B3F-1100	B3F-1100S
			150 g	B3F-1002	B3F-1002S	B3F-1102	B3F-1102S
			High-force: 260 g	B3F-1005	B3F-1005S	B3F-1105	B3F-1105S
		5.0 x 6.5 mm	General-purpose: 100 g	B3F-1020	B3F-1020S	B3F-1120	B3F-1120S
			150 g	B3F-1022	B3F-1022S	B3F-1122	B3F-1122S
			High-force: 260 g	B3F-1025	B3F-1025S	B3F-1125	B3F-1125S
		5.0 x 7.5 mm	General-purpose: 100 g	—	—	B3F-1110	—
	Projected	7.3 x 6.5 mm	General-purpose: 100 g	B3F-1050	B3F-1050S	B3F-1150	B3F-1150S
			150 g	B3F-1052	B3F-1052S	B3F-1152	B3F-1152S
			High-force: 260 g	B3F-1055	B3F-1055S	B3F-1155	B3F-1155S
Vertical	Flat	3.15 mm	General-purpose: 100 g	—	—	B3F-3100	—
			150 g	—	—	B3F-3102	—
			High-force: 260 g	—	—	B3F-3105	—
		3.85 mm	General-purpose: 100 g	—	—	B3F-3120	—
			150 g	—		B3F-3122	—
			High-force: 260 g	—	—	B3F-3125	—
	Projected	6.15 mm	General-purpose: 100 g	—	—	B3F-3150	—
			150 g	—	—	B3F-3152	—
			High-force: 260 g	—	—	B3F-3155	—

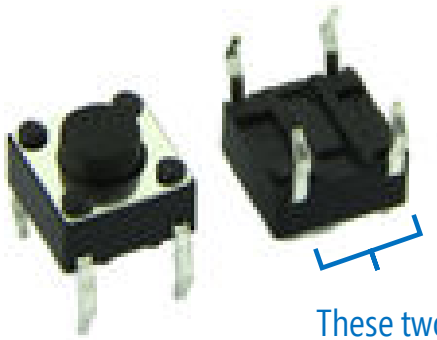
* Number of switches per stick: Without ground terminal 90/stick
With ground terminal 75/stick

Important Note: Switches cannot be water-washed.

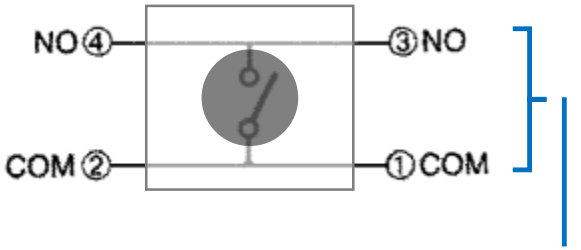
1128 Tactile Switch **B3F**





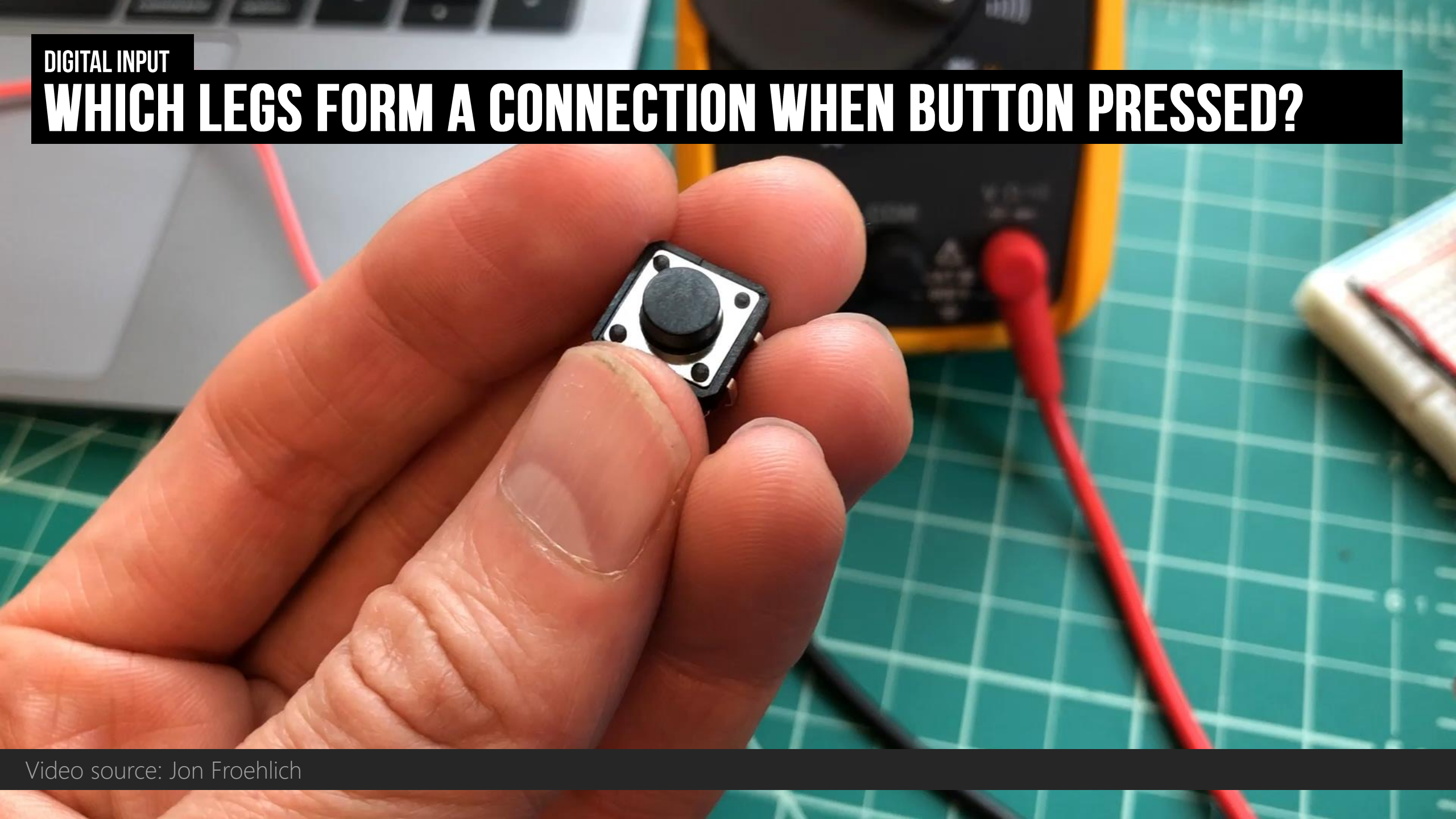


These two sides become connected when button is pressed down



These two sides become connected when button is pressed down

https://www.adafruit.com/product/1119

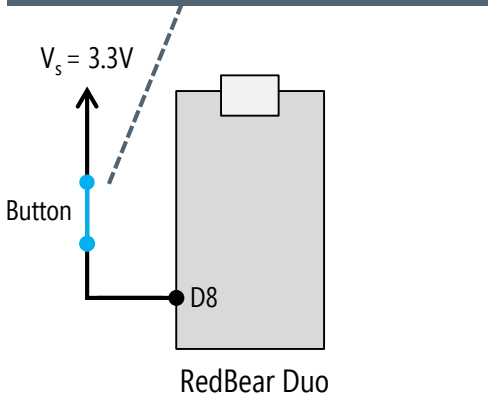


Video source: Jon Froehlich

DIGITAL INPUT

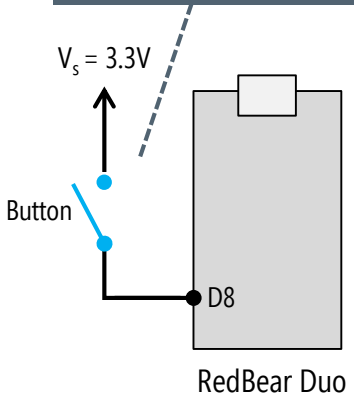
HOOKING UP A BUTTON: PULL-DOWN RESISTORS

When the this button is closed (as it is below), what would the **microcontroller** (MCU) **read** from the **input pin**?



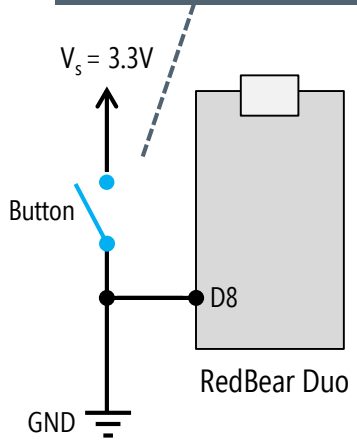
Answer: The MCU would read 3.3V (or HIGH)

Now, when the **button switches** to **open**, what would the MCU read?



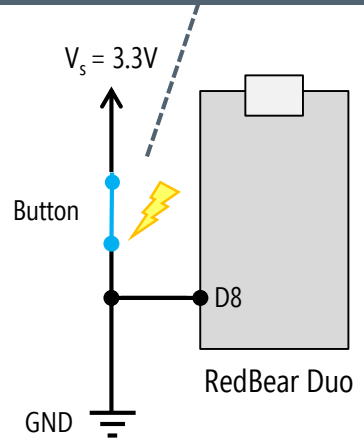
Answer: Uh, oh, the input pin is in an unknown state (commonly called "floating")—this is bad!

Well, can't we just solve that by **adding GND** here that "pulls" D8 to 0V when the switch is open?



Answer: You're on the right track but this will create a short circuit when the button is closed!

Now, when the button is closed, we have a **short circuit** (V_s is connected to GND. This could damage our power supply, circuit components, etc.)!

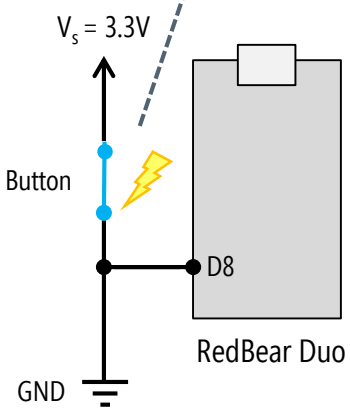


Question: So, what do we do? We add what's called a pull-down resistor.

See: https://learn.sparkfun.com/tutorials/switch-basics/all; https://learn.sparkfun.com/tutorials/pull-up-resistors/all

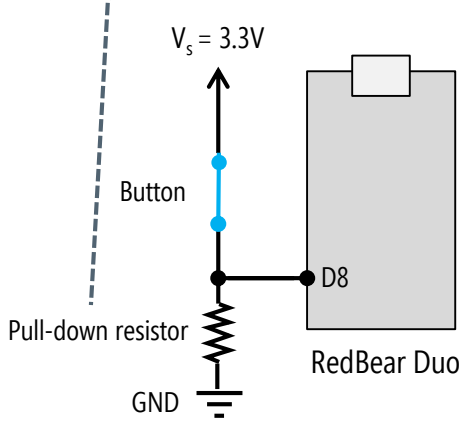
HOOKING UP A BUTTON: PULL-DOWN RESISTORS

Now, when the button is closed, we have a **short circuit** (V_s is connected to GND. This could damage our MCU)!



Question: So, what do we do? We add what's called a pull-down resistor.

This resistor is called a "**pull-down resistor**" because it biases the input low (to GND) when the switch is open.



Now, when the switch is open, the MCU is pulled to GND. When the switch is closed, the MCU is 3.3V (HIGH).

How do you choose the resistor value for the pull-down?

If you choose a low resistor value, more current will be "wasted" by flowing from V_s to GND when the button is closed. In contrast, a high resistor value (*e.g.*, 4MΩ) may not work as a pull-down (not enough current will flow).

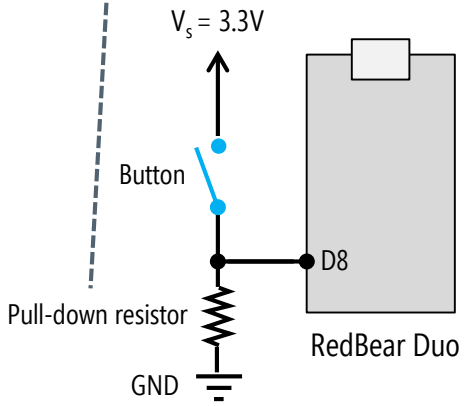
Arduino recommends using a 10KΩ resistor.

See: https://learn.sparkfun.com/tutorials/switch-basics/all; https://learn.sparkfun.com/tutorials/pull-up-resistors/all

DIGITAL INPUT

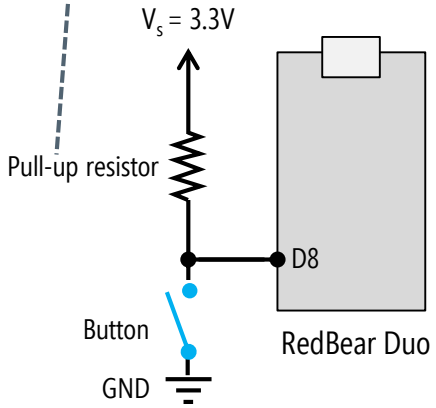
PULL-DOWN VS. PULL-UP RESISTORS

This resistor is called a "**pull-down resistor**" because it biases the input low (to GND) when the switch is open.



Pull-Down Resistor Configuration
When the switch is open, the MCU is pulled to GND. When the switch is closed, the MCU is 3.3V (HIGH) as it becomes directly connected to V_s.

This resistor is called a "**up resistor**" because it biases the input high (to V_s) when the switch is open.



Pull-Up Resistor Configuration
When the switch is open, the MCU is pulled to V_s. When the switch is closed, the MCU is 0V (LOW) as it becomes directly connected to GND.

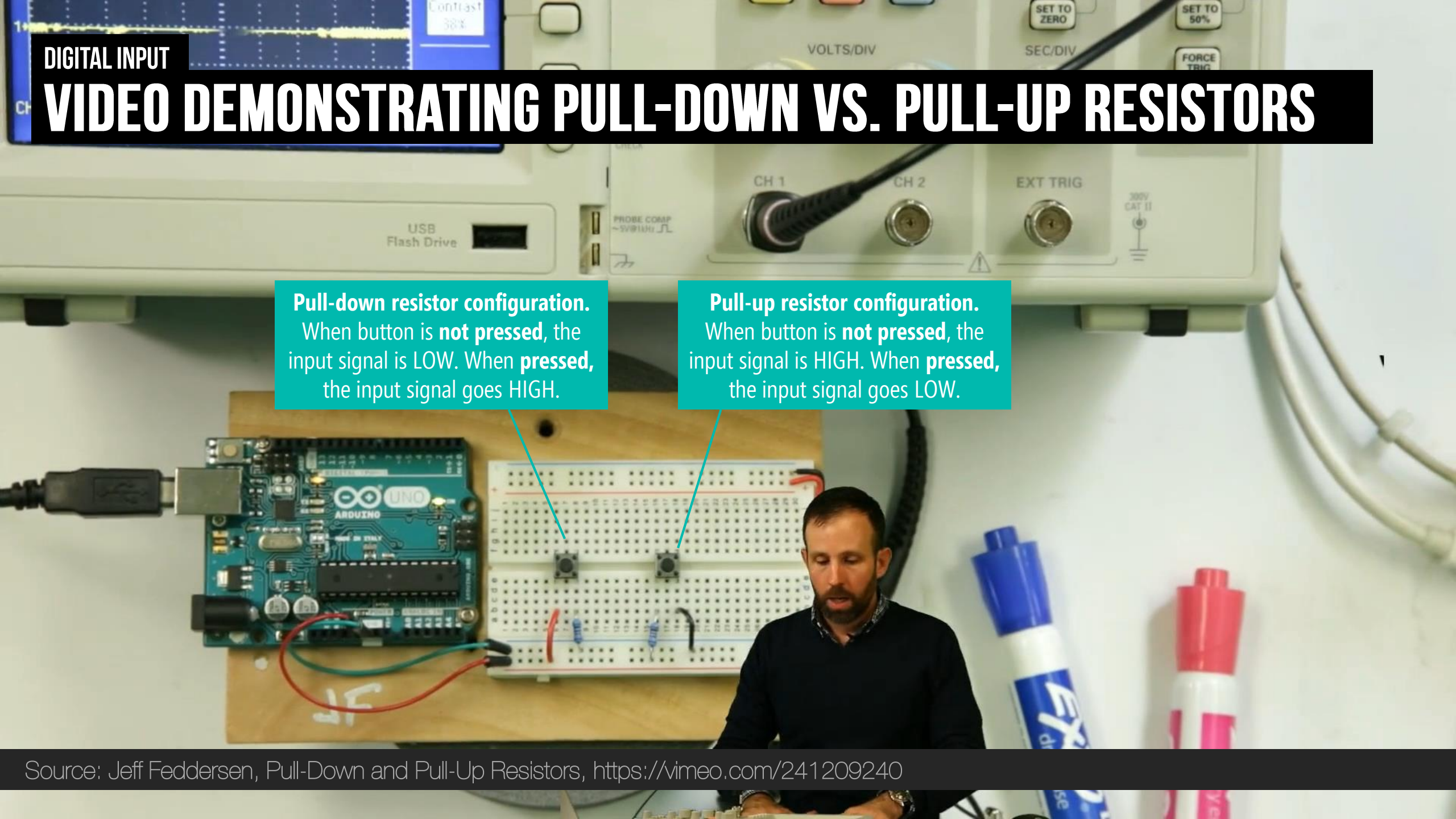
You can wire up either pull-down resistor configurations or pull-up resistor configurations.

I have a preference towards pull-down resistors because it makes more sense to me that a switch is 0V (LOW) when open and 3.3 (HIGH) when closed.

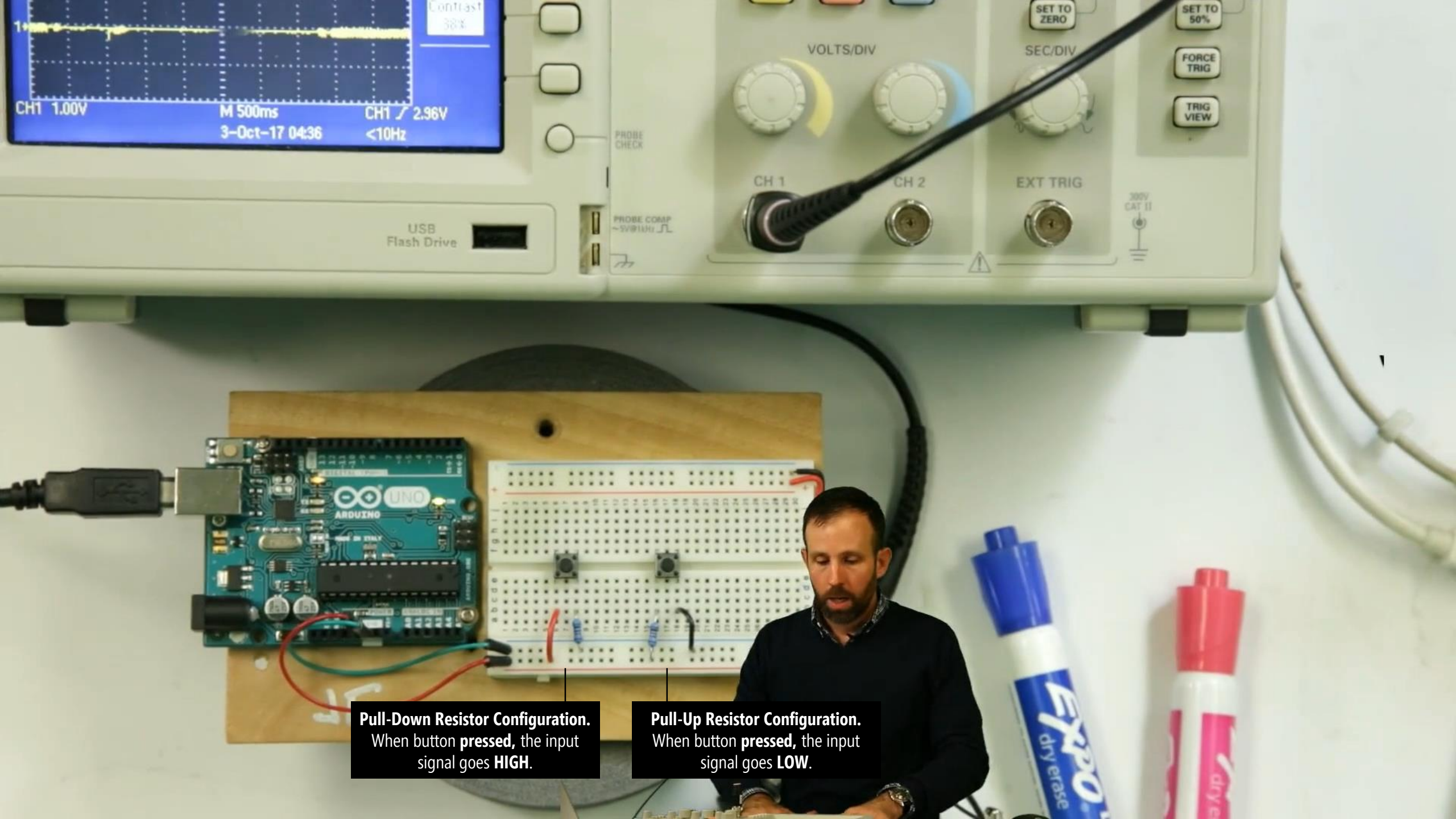
See: https://learn.sparkfun.com/tutorials/switch-basics/all; https://learn.sparkfun.com/tutorials/pull-up-resistors/all

DIGITAL INPUT

VIDEO DEMONSTRATING PULL-DOWN VS. PULL-UP RESISTORS



Source: Jeff Feddersen, Pull-Down and Pull-Up Resistors, https://vimeo.com/241209240


CH1 1.00V
M 500ms
CH1 ∕ 2.96V
3-Oct-17 04:36
<10Hz
VOLTS/DIV
SEC/DIV
SET TO ZERO
SET TO 50%
FORCE TRIG
TRIG VIEW
CH 1
CH 2
EXT TRIG
USB Flash Drive
ARDUINO
UNO
Pull-Down Resistor Configuration. When button pressed, the input signal goes HIGH.
Pull-Up Resistor Configuration. When button pressed, the input signal goes LOW.

EXERCISE: TURN ON/OFF LED WITH A BUTTON

Build circuit and write code to turn on/off button

Use a pull-down resistor

Use the digitalRead method

digitalRead()

Reads the value from a specified digital ping and returns either HIGH or LOW

Syntax

```
digitalRead(pin)
```

Parameters

`pin`: the digital I/O pin you want to read

Returns

HIGH or LOW

EXERCISE: TURN ON/OFF LED WITH A BUTTON

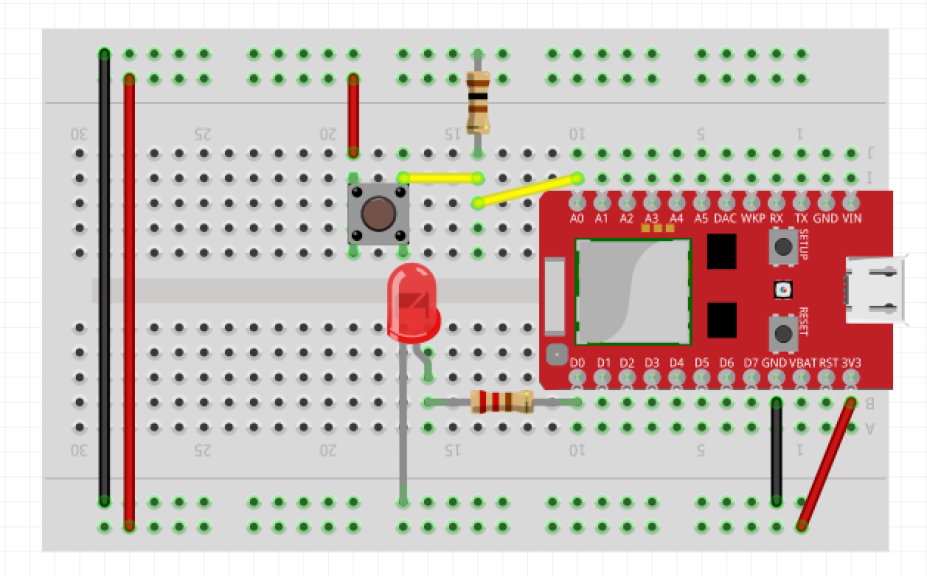
Build circuit and write code to turn on/off button

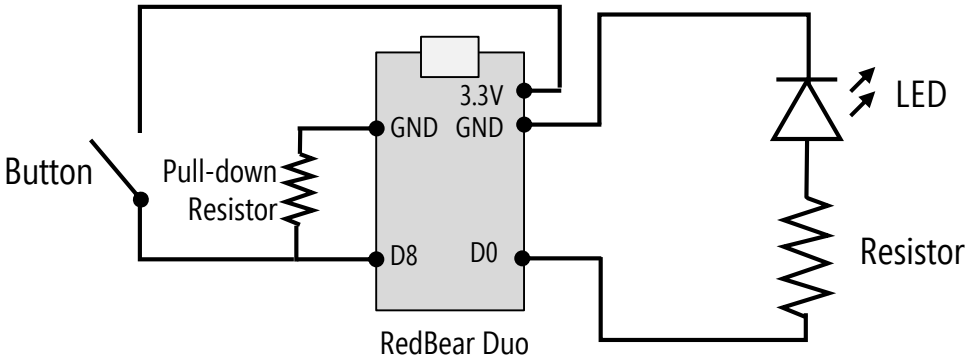
Use a pull-down resistor

DIGITAL INPUT

TURN ON/OFF LED WITH A BUTTON

Circuit: Control LED via a button & pull-down resistor



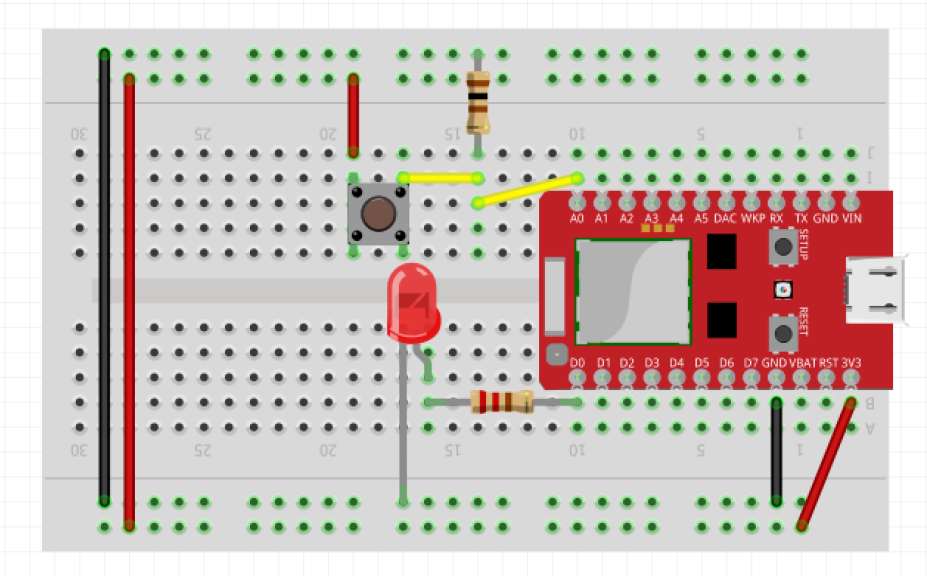


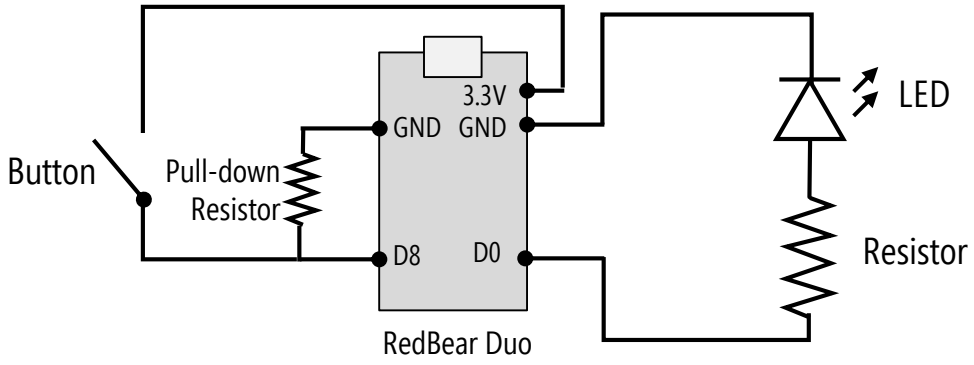
Now we just have to write the Arduino code to read in the button state and turn on/off the LED accordingly.

DIGITAL INPUT

TURN ON/OFF LED WITH A BUTTON

Circuit: Control LED via a button & pull-down resistor





Code: Read button state and set LED on/off accordingly

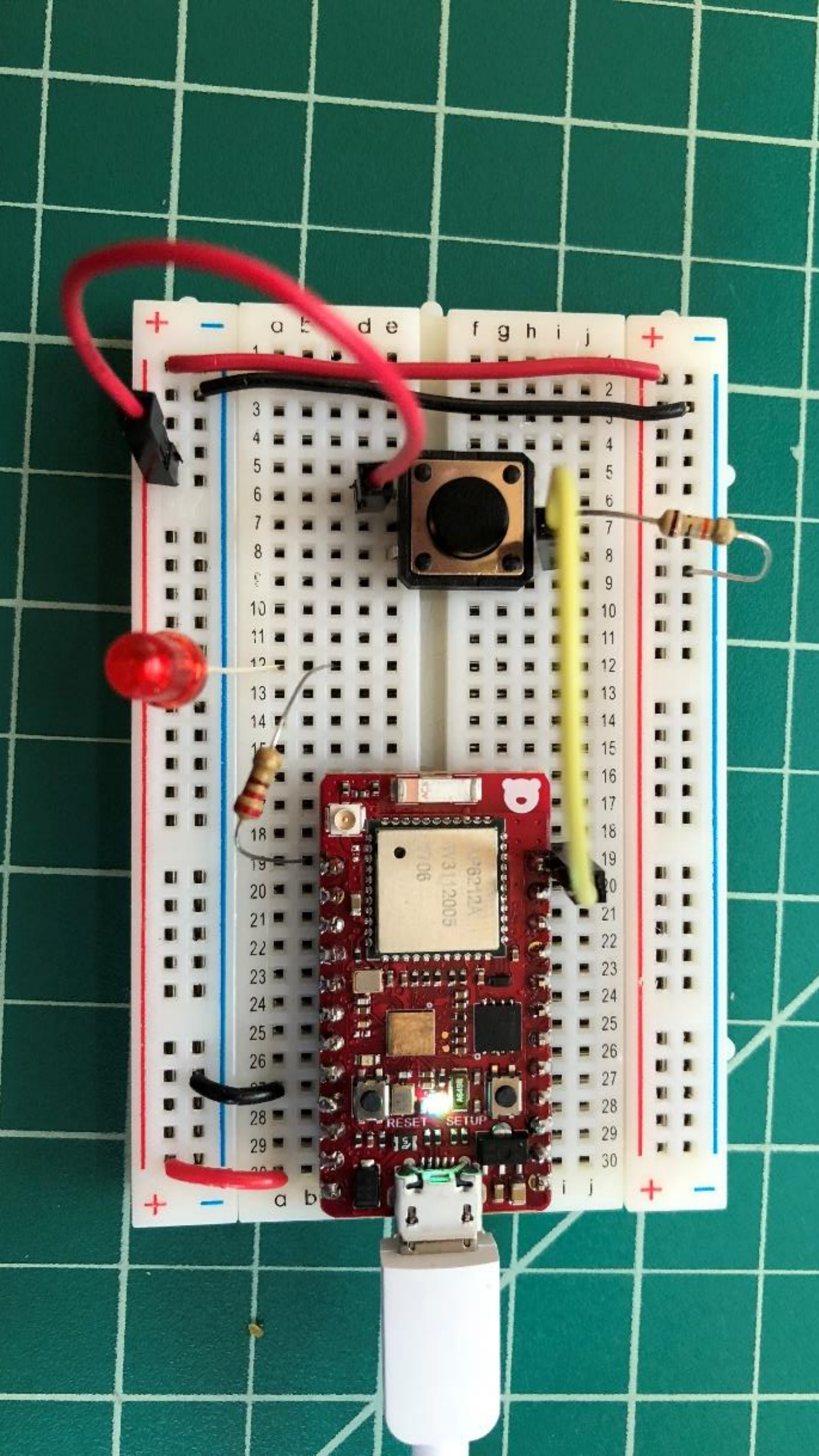
RedBearDuoReadButtonSetLED | Arduino 1.8.5

File Edit Sketch Tools Help

RedBearDuoReadButtonSetLED §

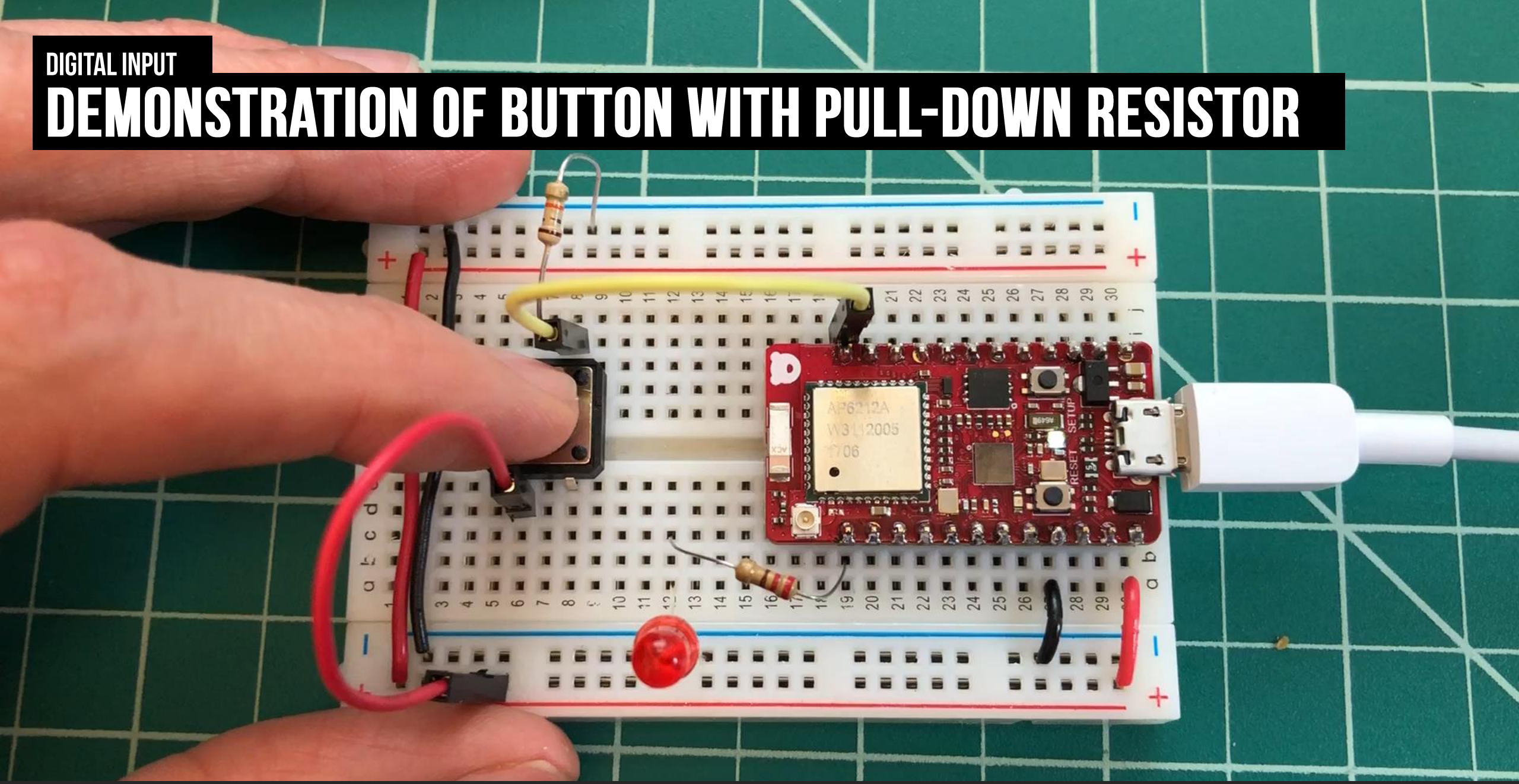
```
/*
 * This example reads in a button input on D8 (with a pull-down resistor configuration)
 * and turns on/off an LED on D0 accordingly
 *
 * By Jon Froehlich for CSE590
 * http://makeabilitylab.io
 */
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int BUTTON_INPUT_PIN = D8;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(BUTTON_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // read the button value. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonVal = digitalRead(BUTTON_INPUT_PIN);

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN, buttonVal);

  // Check for new input every 100ms
  delay(100);
}
```

https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L05-Arduino/RedBearDuoReadButtonSetLED

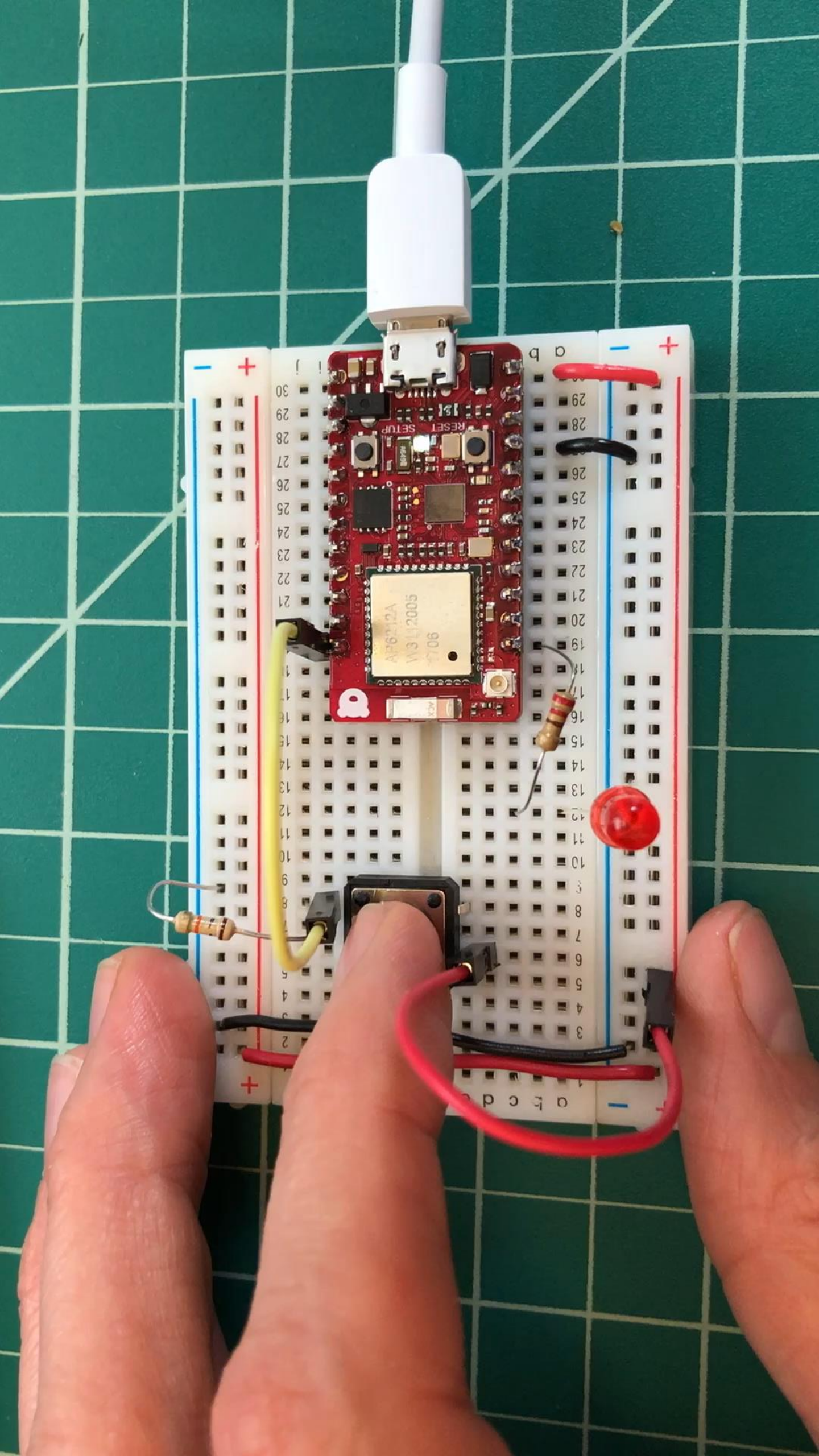

RESET
SETUP

DIGITAL INPUT

DEMONSTRATION OF BUTTON WITH PULL-DOWN RESISTOR



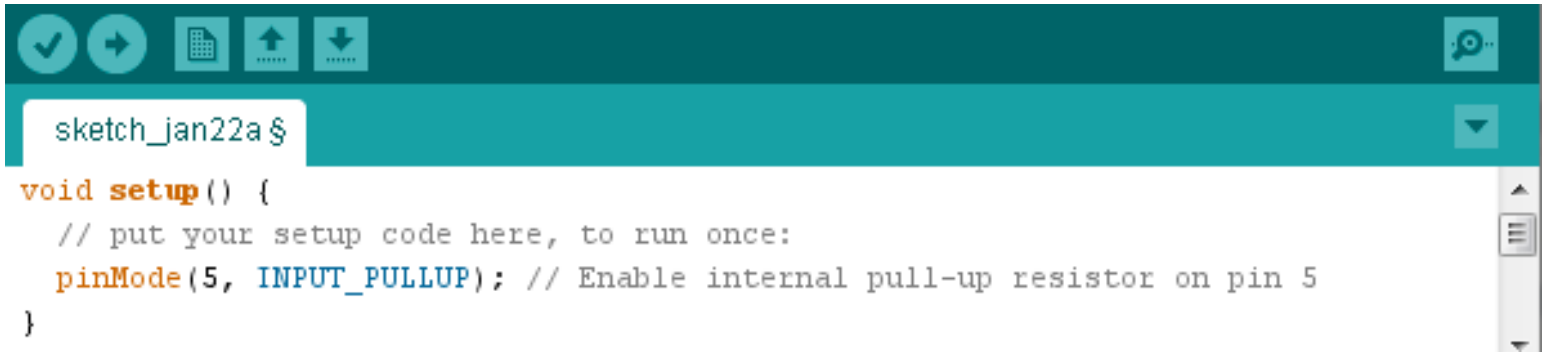
Video source: Jon Froehlich


RESET
SETUP

DIGITAL INPUT

MANY MCUS HAVE THESE RESISTORS BUILT IN

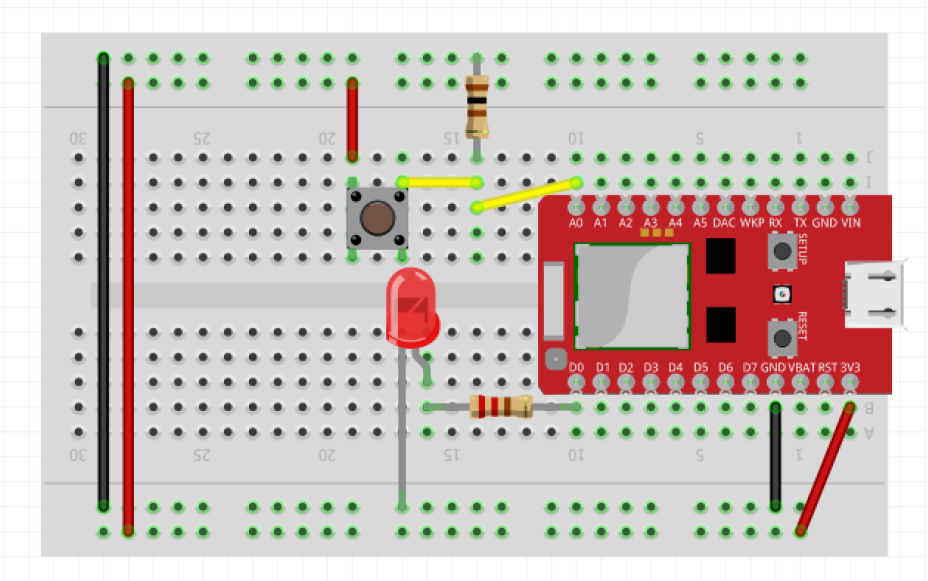
Pull-up and pull-down resistors are so commonly needed that many MCUs have them built-in. The ATmega328 microcontroller on the Arduino Uno has internal pull-up resistors (20kΩ). The RedBear Duo has both internal pull-up and pull-down resistors!

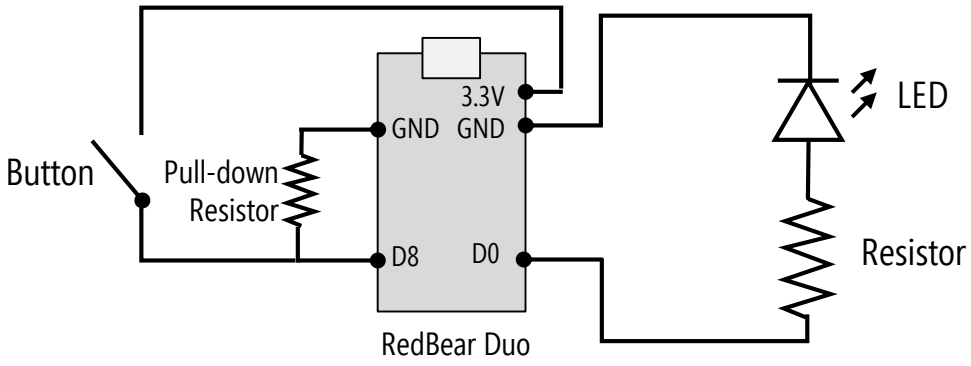


DIGITAL INPUT

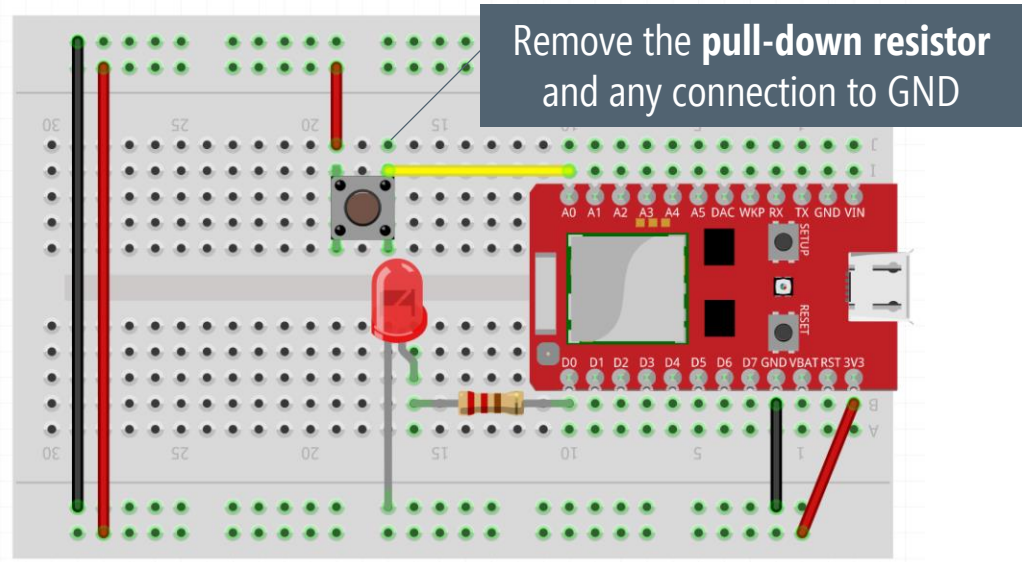
MODIFY YOUR CIRCUIT TO USE INTERNAL PULL-DOWN

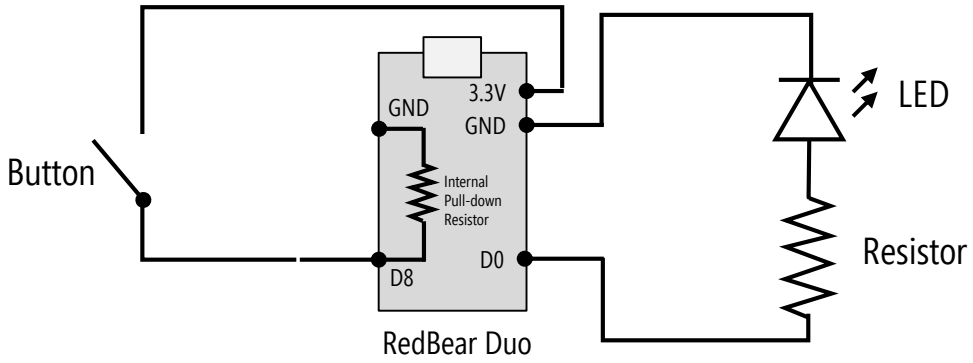
Old Circuit: Control LED via a button & pull-down resistor





New Circuit: Control LED via a button & internal pull-down resistor





DIGITAL INPUT

MODIFY CODE TO USE INTERNAL PULL-DOWN

Old Code: Read button state and set LED on/off accordingly. Circuit uses an external pull-down resistor

RedBearDuoReadButtonSetLED | Arduino 1.8.5

File Edit Sketch Tools Help

RedBearDuoReadButtonSetLED §

```
/*
 * This example reads in a button input on D8 (with a pull-down resistor configuration)
 * and turns on/off an LED on D0 accordingly
 *
 * By Jon Froehlich for CSE590
 * http://makeabilitylab.io
 */
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int BUTTON_INPUT_PIN = D8;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(BUTTON_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // read the button value. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonVal = digitalRead(BUTTON_INPUT_PIN);

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN, buttonVal);

  // Check for new input every 100ms
  delay(100);
}
```

https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L05-Arduino/RedBearDuoReadButtonSetLED

New Code: Read button state and set LED on/off accordingly. Circuit uses an internal pull-down resistor.

RedBearDuoReadButtonIntPullDownSetLED | Arduino 1.8.5

File Edit Sketch Tools Help

RedBearDuoReadButtonIntPullDownSetLED §

```
/*
 * This example reads in a button input on D8 (with an internal pull-down resistor configuration)
 * and turns on/off an LED on D0 accordingly
 *
 * By Jon Froehlich for CSE590
 * http://makeabilitylab.io
 */
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int BUTTON_INPUT_PIN = D8;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(BUTTON_INPUT_PIN, INPUT_PULLDOWN); // use the RedBear Duo internal pull-down resistor
  Serial.begin(9600);
}

void loop() {

  // Read the button value. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonVal = digitalRead(BUTTON_INPUT_PIN);

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN, buttonVal);

  // Check for new input every 100ms
  delay(100);
}
```

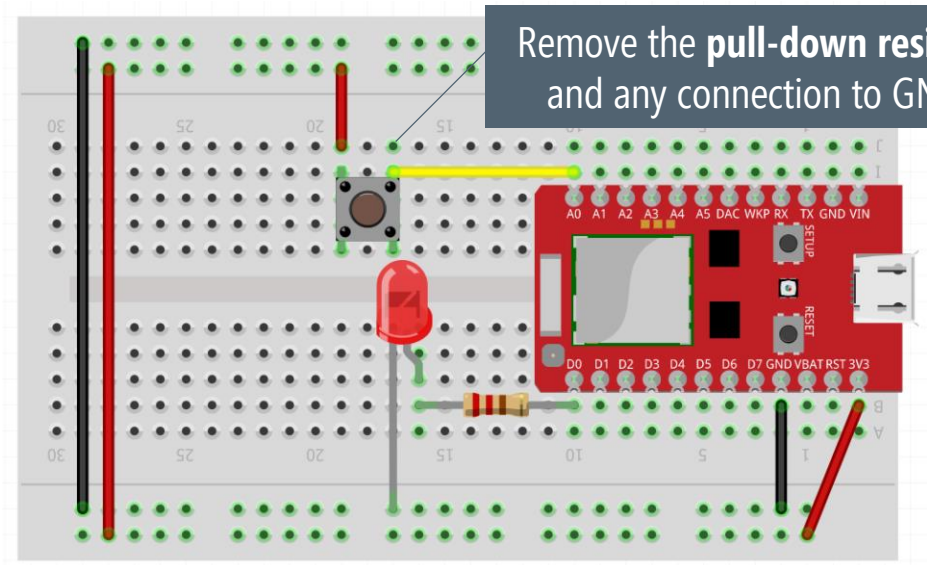
This line is the **only difference**

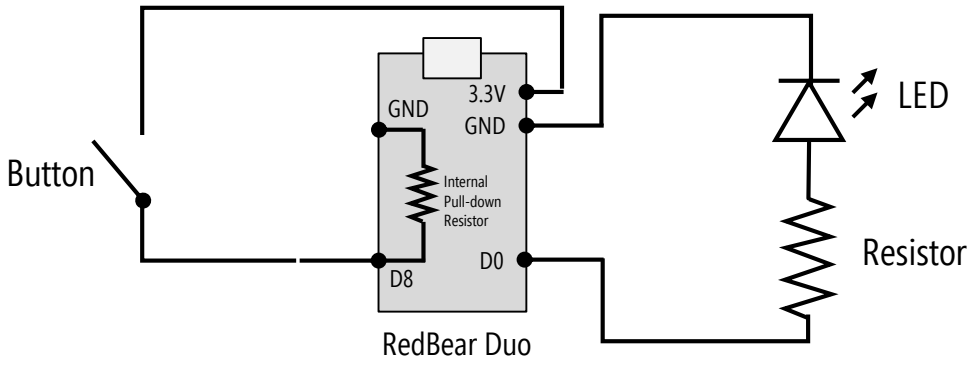
https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L05-Arduino/RedBearDuoReadButtonSetLED

DIGITAL INPUT

RUN THE PROGRAM

New Circuit: Control LED via a button & internal pull-down resistor





New Code: Read button state and set LED on/off accordingly. Circuit uses an internal pull-down resistor.

RedBearDuoReadButtonIntPullDownSetLED | Arduino 1.8.5

File Edit Sketch Tools Help

RedBearDuoReadButtonIntPullDownSetLED §

```
/*
 * This example reads in a button input on D8 (with an internal pull-down resistor configuration)
 * and turns on/off an LED on D0 accordingly
 *
 * By Jon Froehlich for CSE590
 * http://makeabilitylab.io
 */
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int BUTTON_INPUT_PIN = D8;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(BUTTON_INPUT_PIN, INPUT_PULLDOWN); // use the RedBear Duo internal pull-down resistor
  Serial.begin(9600);
}

void loop() {

  // Read the button value. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonVal = digitalRead(BUTTON_INPUT_PIN);

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN, buttonVal);

  // Check for new input every 100ms
  delay(100);
}
```

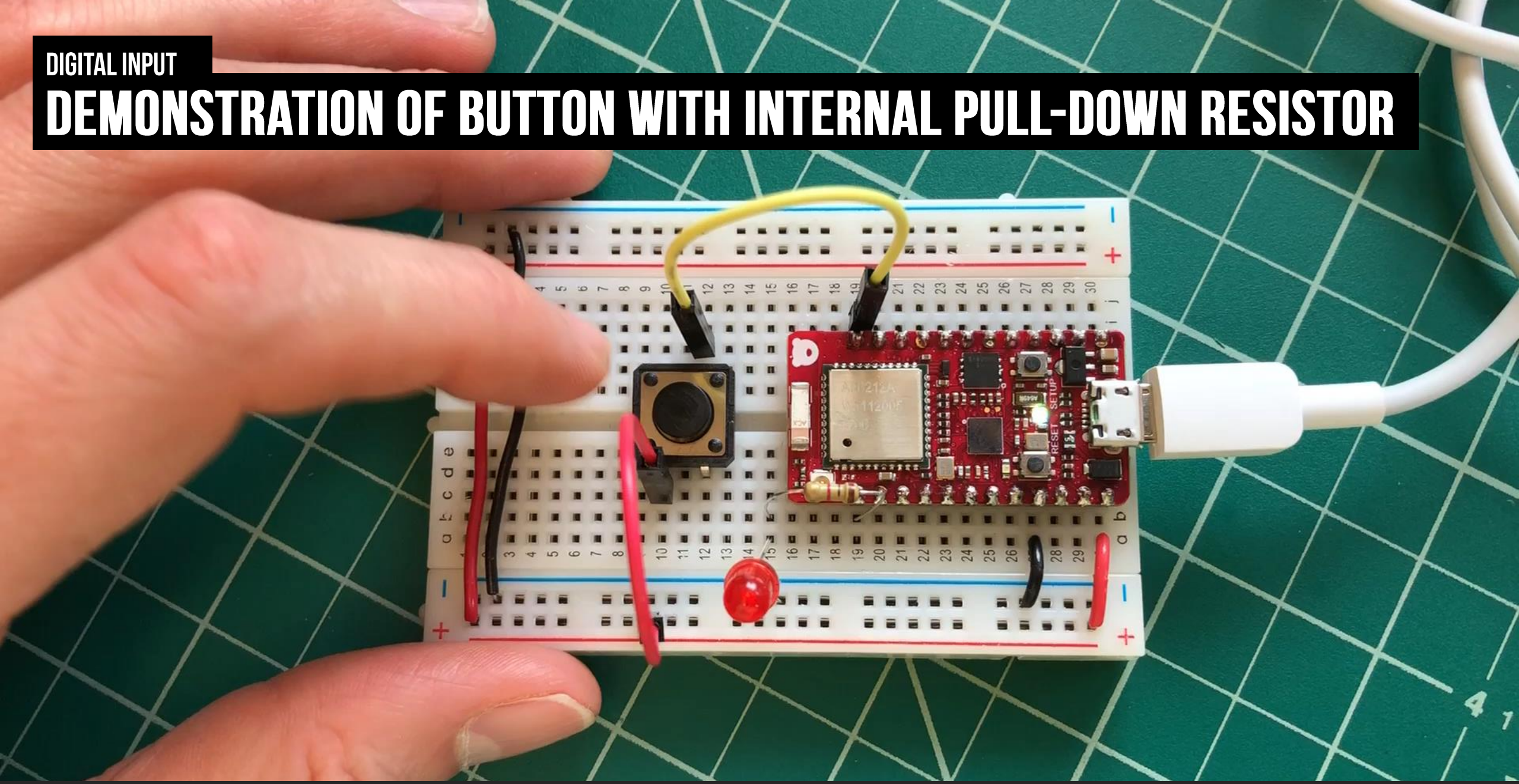
This is the **only difference**

https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L05-Arduino/RedBearDuoReadButtonSetLED

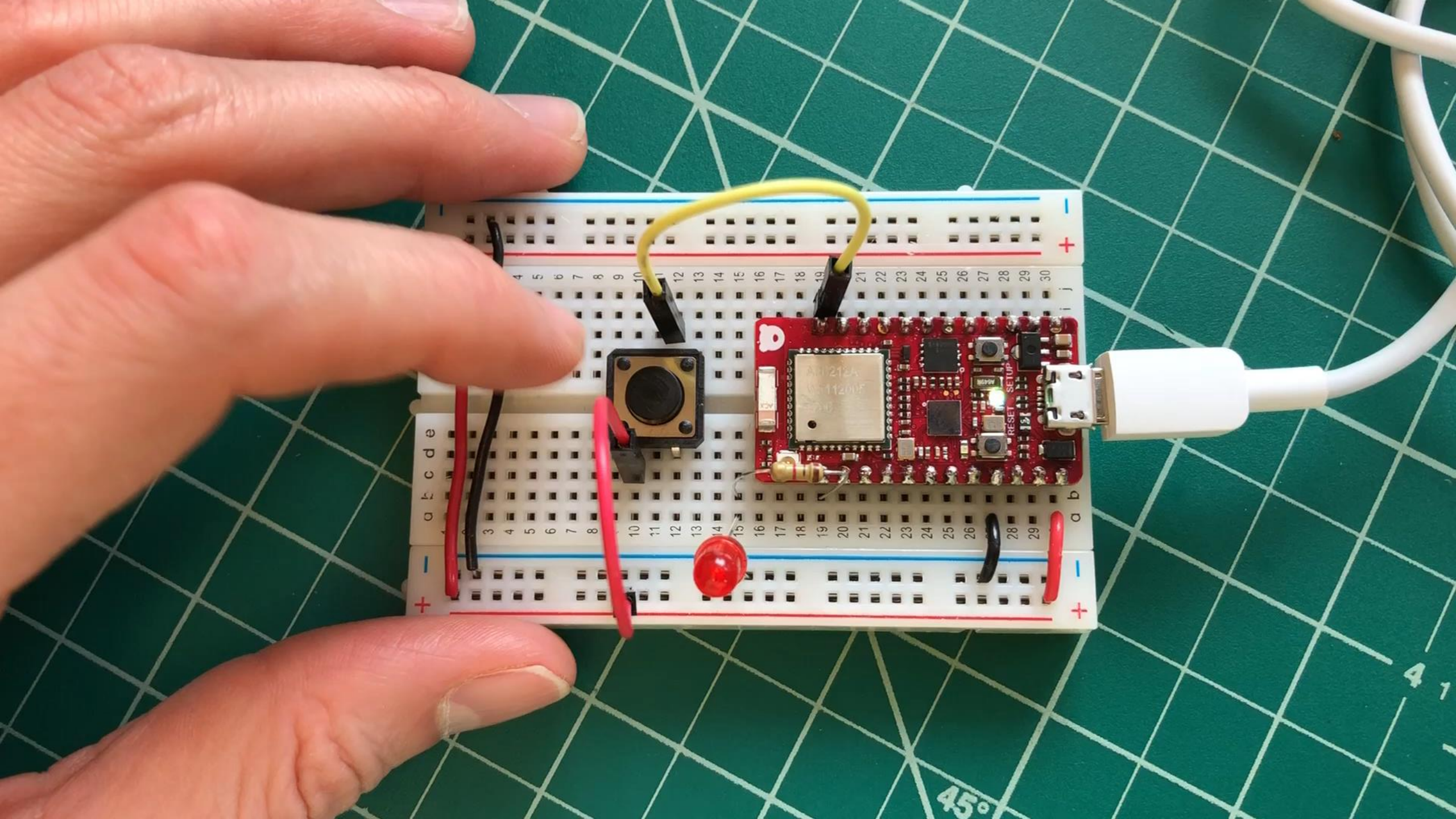
SETUP
RESET

DIGITAL INPUT

DEMONSTRATION OF BUTTON WITH INTERNAL PULL-DOWN RESISTOR

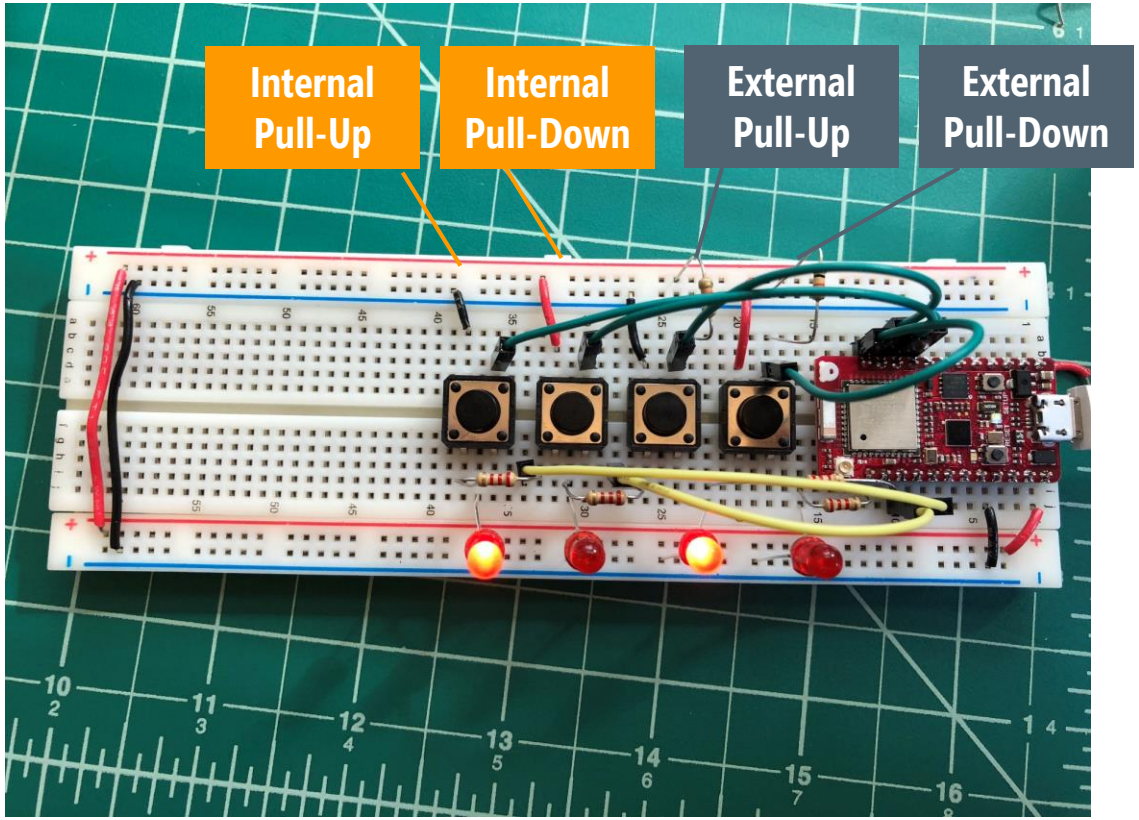


Video source: Jon Froehlich



EXPERIMENTING WITH INTERNAL & EXTERNAL PULL-UP AND PULL-DOWN RESISTORS

Circuit: In left-to-right order of buttons: internal pull-up resistor, internal pull-down resistor, external pull-up, & external pull-down



Code: Demonstrates differences between using internal and external pull-up and pull-down resistors

RedBearDuoButtonPullUpAndDown | Arduino 1.8.5

File Edit Sketch Tools Help

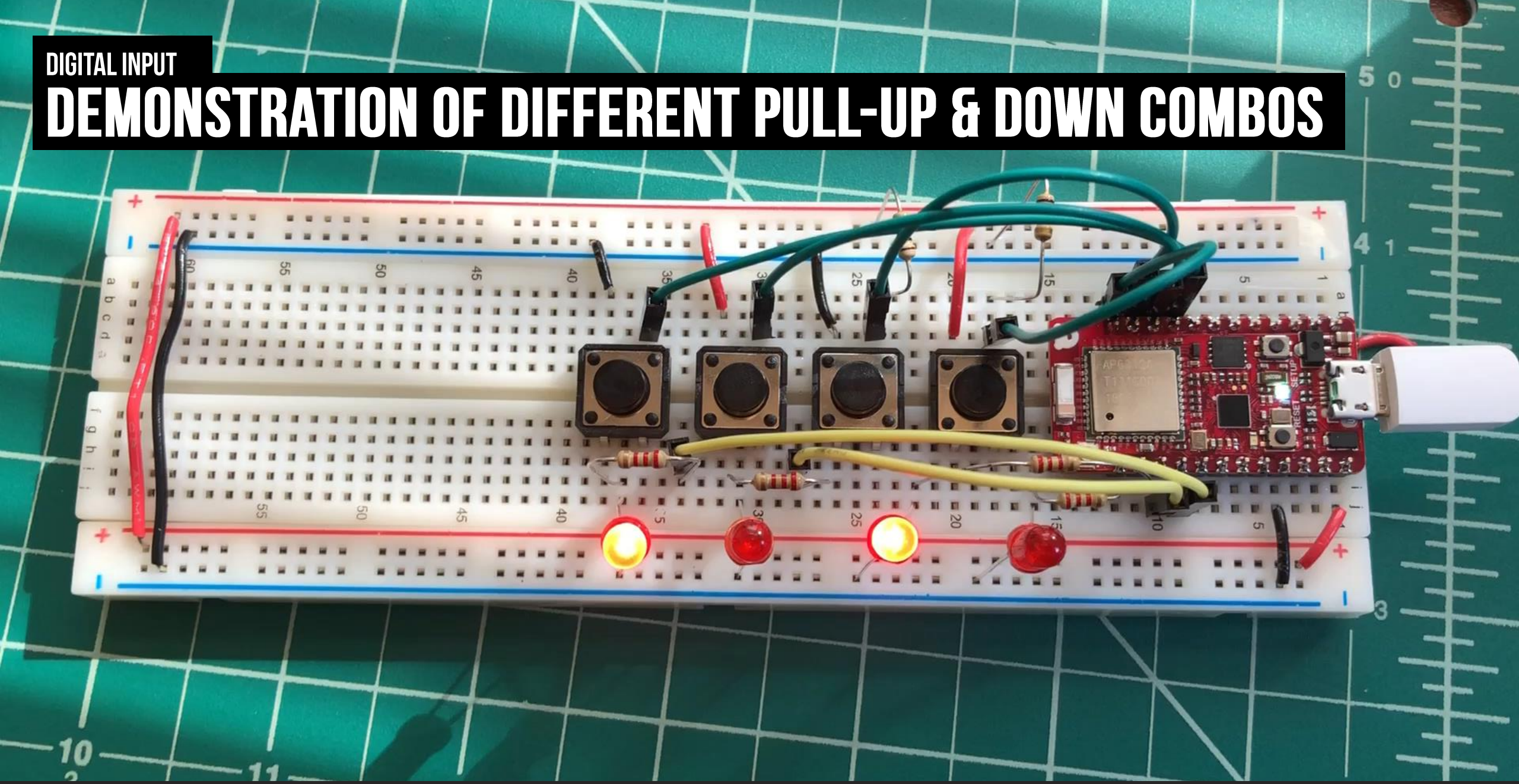
RedBearDuoButtonPullUpAndDown

```
SYSTEM_MODE(MANUAL);

const int LED0_OUTPUT_PIN = D0;
const int LED1_OUTPUT_PIN = D1;
const int LED2_OUTPUT_PIN = D2;
const int LED3_OUTPUT_PIN = D3;

const int BUTTON_EXTPULLDOWN_INPUT_PIN = D8;
const int BUTTON_EXTPULLUP_INPUT_PIN = D9;
const int BUTTON_INTPULLDOWN_INPUT_PIN = D10;
const int BUTTON_INTPULLUP_INPUT_PIN = D11;

void setup() {
  pinMode(LED0_OUTPUT_PIN, OUTPUT);
  pinMode(LED1_OUTPUT_PIN, OUTPUT);
  pinMode(LED2_OUTPUT_PIN, OUTPUT);
  pinMode(LED3_OUTPUT_PIN, OUTPUT);

  pinMode(BUTTON_EXTPULLDOWN_INPUT_PIN, INPUT);
  pinMode(BUTTON_EXTPULLUP_INPUT_PIN, INPUT);
  pinMode(BUTTON_INTPULLDOWN_INPUT_PIN, INPUT_PULLDOWN);
  pinMode(BUTTON_INTPULLUP_INPUT_PIN, INPUT_PULLUP);
}

void loop() {

  // Read the button with an external pull-down resistor. The value will be HIGH when
  // pressed and LOW when not pressed
  int buttonExtPullDownVal = digitalRead(BUTTON_EXTPULLDOWN_INPUT_PIN);

  // Read the button with an external pull-up resistor. The value will be LOW
  // when pressed and HIGH when not pressed
  int buttonExtPullUpVal = digitalRead(BUTTON_EXTPULLUP_INPUT_PIN);

  // Read the button with an internal pull-down resistor. The value will be HIGH when
  // pressed and LOW when not pressed
  int buttonIntPullDownVal = digitalRead(BUTTON_INTPULLDOWN_INPUT_PIN);

  // Read the button with an internal pull-up resistor. The value will be LOW
  // when pressed and HIGH when not pressed
  int buttonIntPullUpVal = digitalRead(BUTTON_INTPULLUP_INPUT_PIN);
```

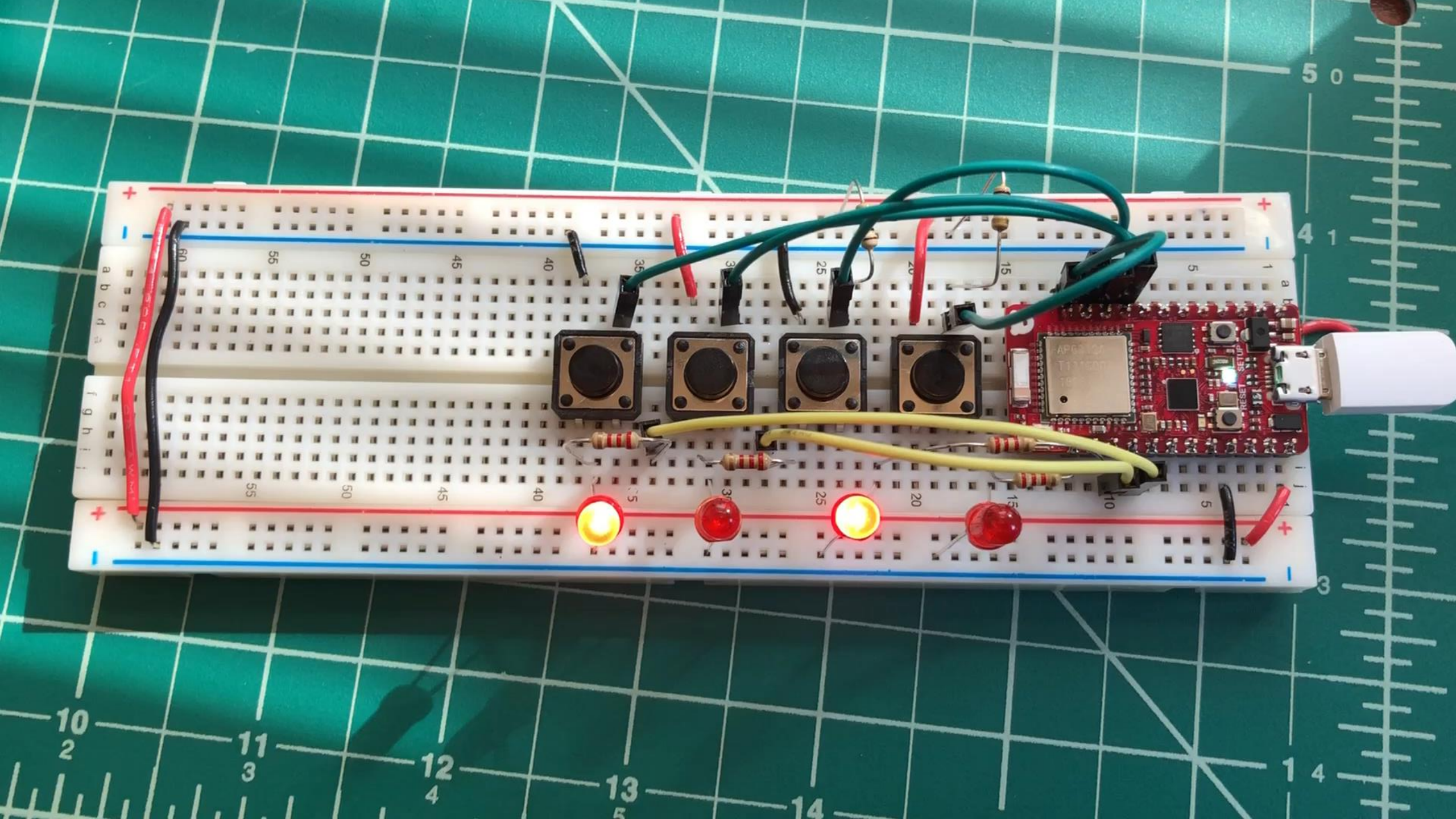
https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L05-Arduino/RedBearDuoButtonPullUpAndDown

DIGITAL INPUT

DEMONSTRATION OF DIFFERENT PULL-UP & DOWN COMBOS



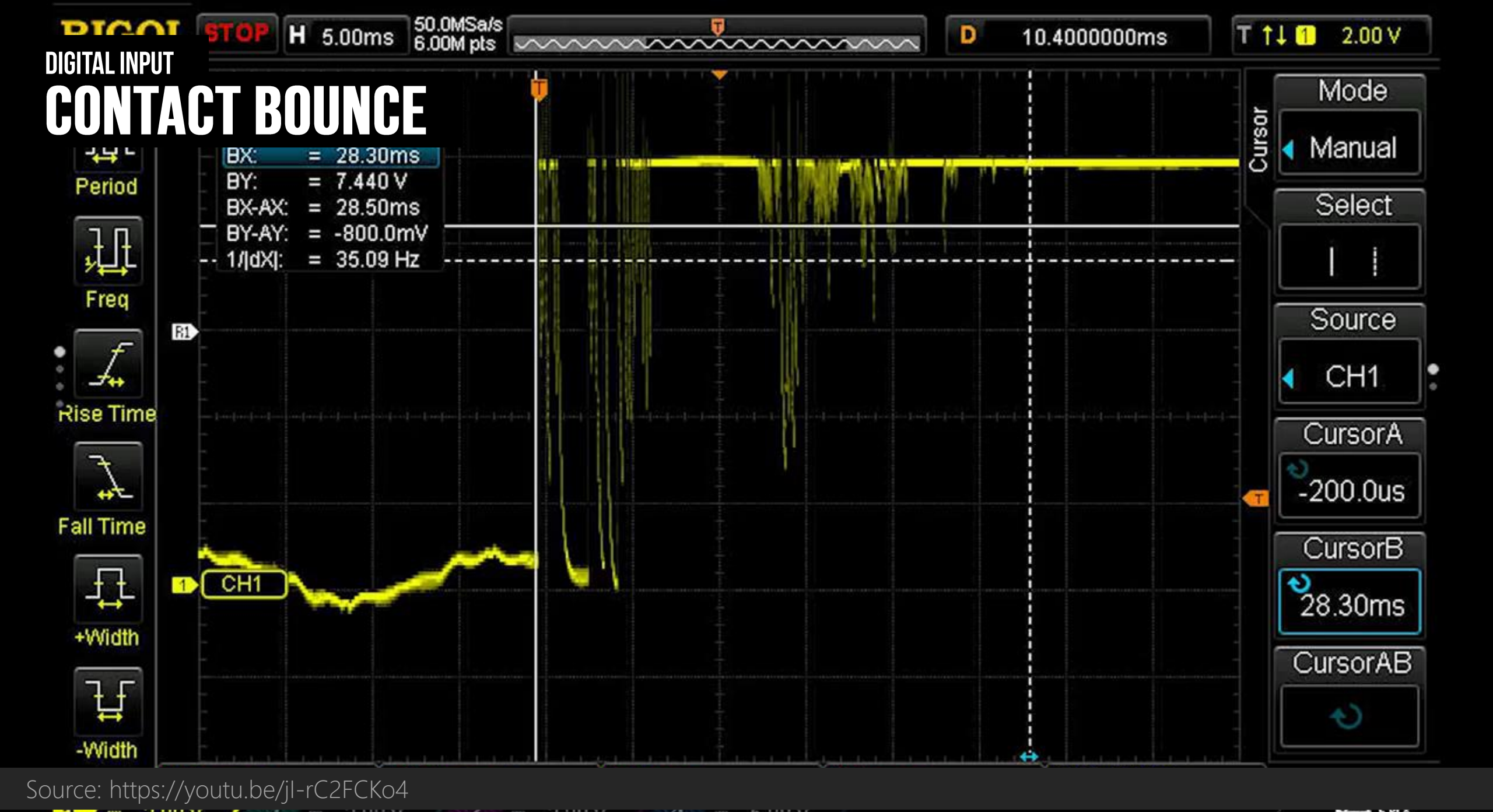
Video source: Jon Froehlich



Another issue we have to deal with when working with buttons or switches is **debouncing**—a process of eliminating noise due to transient oscillations from electro-mechanical conductive parts "bouncing" against each other.

DIGITAL INPUT

CONTACT BOUNCE

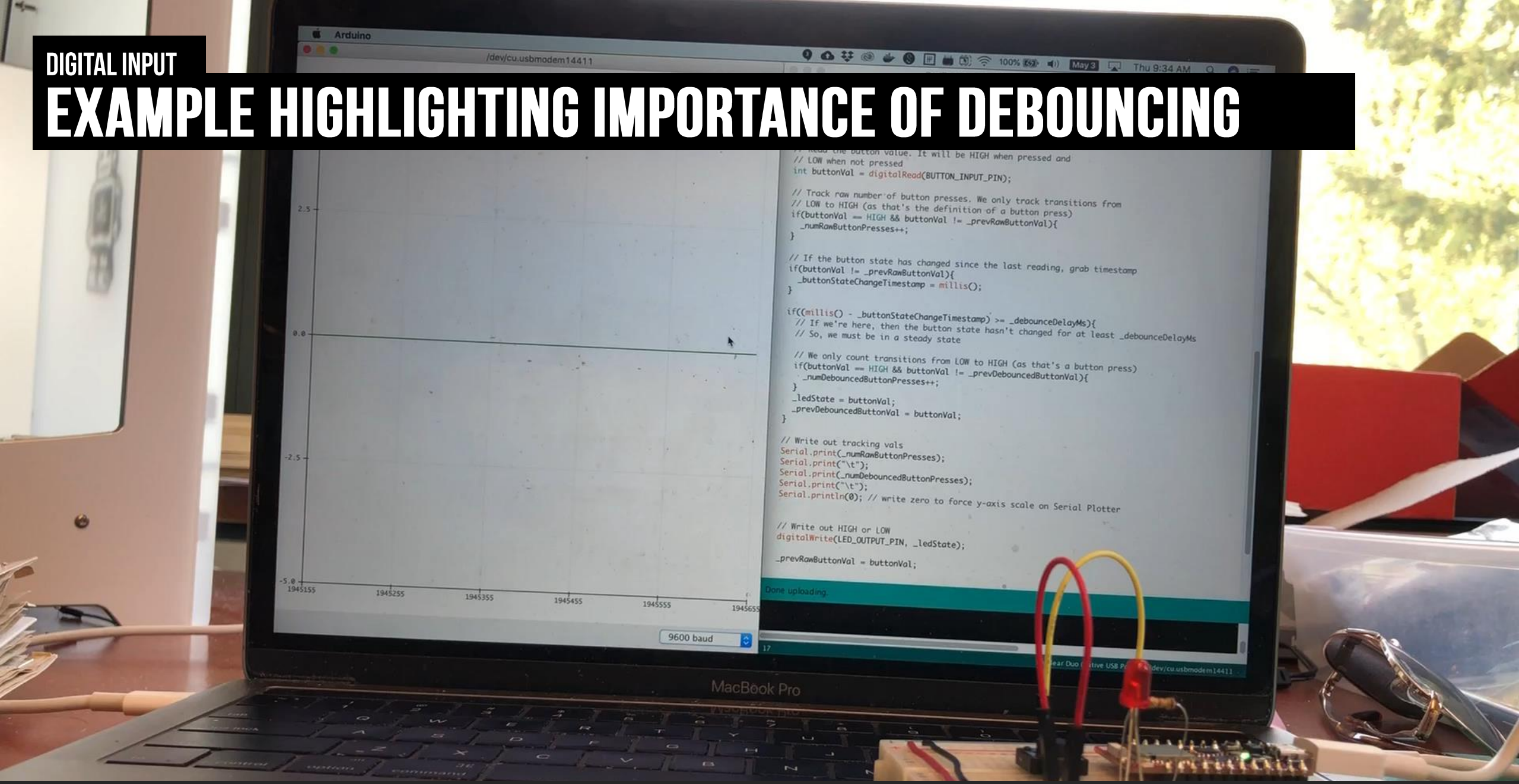


Source: https://youtu.be/jI-rC2FCKo4

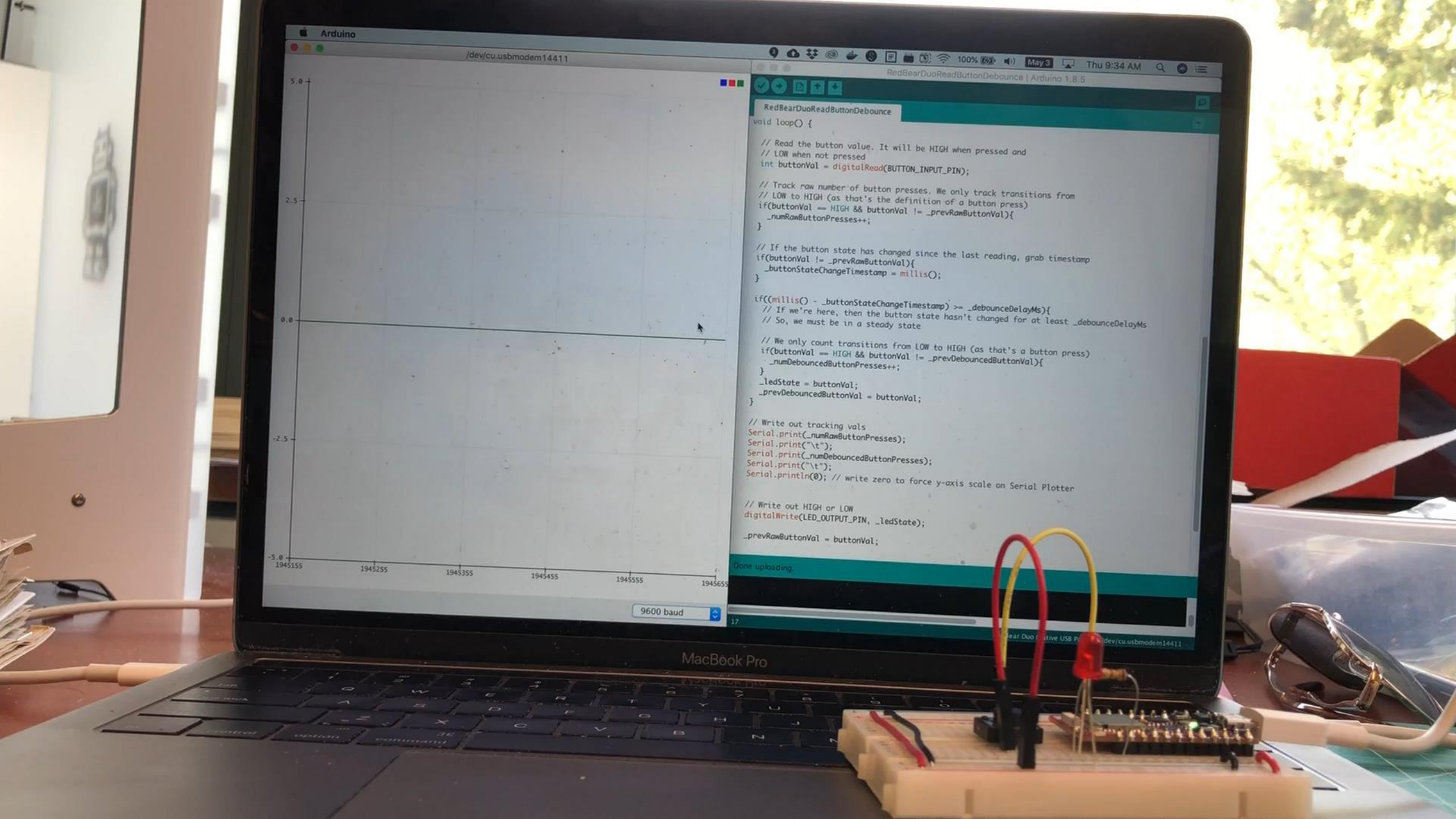

RIGOL
STOP
H 5.00ms
50.0MSa/s
6.00M pts
D 10.4000000ms
T 2.00 V
Horizontal
Period
Freq
Rise Time
Fall Time
+Width
-Width
AX: = -200.0us
AY: = 8.240 V
BX: = 28.30ms
BY: = 7.440 V
BX-AX: = 28.50ms
BY-AY: = -800.0mV
1/|dX|: = 35.09 Hz
CH1
Cursor
Mode
Manual
Select
Source
CH1
CursorA
-200.0us
CursorB
28.30ms
CursorAB
Rise=*****
+Width=*****
Fall=*****
Rise=*****
Fall<1.050ms
1 = 2.00 V
2 = 2.00 V
3 = 2.00 V
4 = 5.00 V

DIGITAL INPUT

EXAMPLE HIGHLIGHTING IMPORTANCE OF DEBOUNCING



Source: Jon Froehlich


Arduino
/dev/cu.usbmodem14411
5.0
2.5
0.0
-2.5
-5.0
1945155
1945255
1945355
1945455
1945555
1945655
9600 baud
RedBearDuoReadButtonDebounce | Arduino 1.8.5
RedBearDuoReadButtonDebounce
void loop() {
// Read the button value. It will be HIGH when pressed and
// LOW when not pressed
int buttonVal = digitalRead(BUTTON_INPUT_PIN);
// Track raw number of button presses. We only track transitions from
// LOW to HIGH (as that's the definition of a button press)
if(buttonVal == HIGH && buttonVal != _prevRawButtonVal){
_numRawButtonPresses++;
}
// If the button state has changed since the last reading, grab timestamp
if(buttonVal != _prevRawButtonVal){
_buttonStateChangeTimestamp = millis();
}
if((millis() - _buttonStateChangeTimestamp) >= _debounceDelayMs){
// If we're here, then the button state hasn't changed for at least _debounceDelayMs
// So, we must be in a steady state
// We only count transitions from LOW to HIGH (as that's a button press)
if(buttonVal == HIGH && buttonVal != _prevDebouncedButtonVal){
_numDebouncedButtonPresses++;
}
_ledState = buttonVal;
_prevDebouncedButtonVal = buttonVal;
}
// Write out tracking vals
Serial.print(_numRawButtonPresses);
Serial.print("\t");
Serial.print(_numDebouncedButtonPresses);
Serial.print("\t");
Serial.println(0); // write zero to force y-axis scale on Serial Plotter
// Write out HIGH or LOW
digitalWrite(LED_OUTPUT_PIN, _ledState);
_prevRawButtonVal = buttonVal;
Done uploading.
MacBook Pro

HOW WOULD YOU SOLVE THIS PROBLEM IN SOFTWARE?

With a partner, brainstorm/discuss solutions to this problem

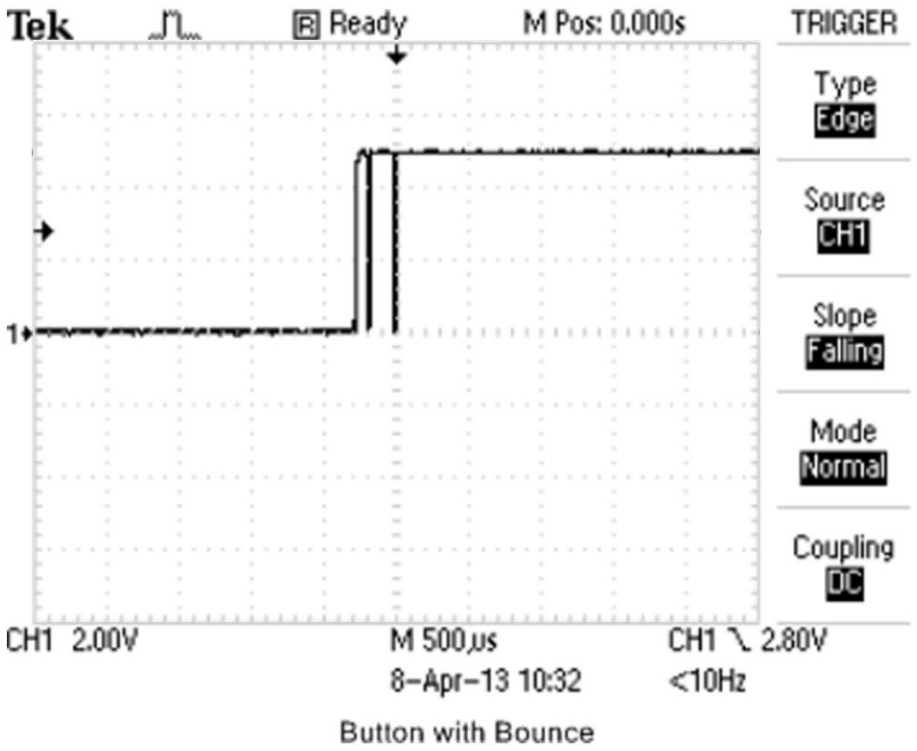


Figure 12-2: Ordinary pushbutton bouncing before settling

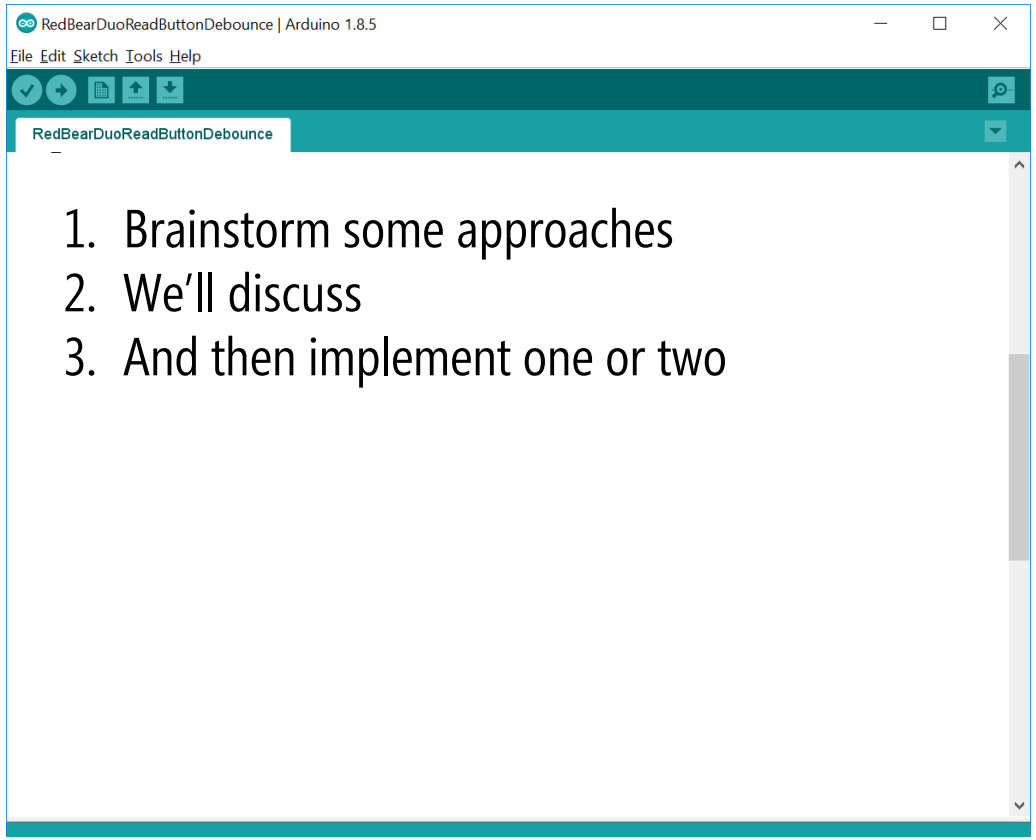


Image source: Jeremey Blum, Exploring Arduino, 2013

TWO EXAMPLE SOLUTIONS

Solution 1 (OK): Read button state. Wait via a delay. Read button state again. If both button states agree, then we have steady state.

RedBearDuoReadButtonOverlySimpleDebouncing | Arduino 1.8.5

File Edit Sketch Tools Help

RedBearDuoReadButtonOverlySimpleDebouncing

```
void loop() {

  // Read the button value. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonVal = digitalRead(BUTTON_INPUT_PIN);

  // Wait to check the button state again
  delay(DEBOUNCE_DELAY);

  // read the button value again
  int buttonVal2 = digitalRead(BUTTON_INPUT_PIN);

  // If buttonVal and buttonVal2 are the same, then we are in steady state
  // Save a new buttonval if a transition occurred.
  if(buttonVal == buttonVal2 && _lastButtonVal != buttonVal){
    _lastButtonVal = buttonVal;
  }

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN, _lastButtonVal);
}
```

https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L05-Arduino/RedBearDuoReadButtonSimpleDebouncing

Solution 2 (Better!): Similar to Solution 1 but we don't use a delay(). Instead, we track a timestamp. This way, we don't block on a delay() call while we are trying to debounce our signal.

RedBearDuoReadButtonDebounce | Arduino 1.8.5

File Edit Sketch Tools Help

RedBearDuoReadButtonDebounce §

```
void loop() {

  // Read the button value. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonVal = digitalRead(BUTTON_INPUT_PIN);

  // If the button state has changed since the last reading, grab timestamp
  if(buttonVal != _prevRawButtonVal){
    _buttonStateChangeTimestamp = millis();
  }

  if((millis() - _buttonStateChangeTimestamp) >= _debounceDelayMs){
    // If we're here, then the button state hasn't changed for at least _debounceDelayMs
    // So, we must be in a steady state

    // We only count transitions from LOW to HIGH (as that's a button press)
    if(buttonVal == HIGH && buttonVal != _prevDebouncedButtonVal){
      _numDebouncedButtonPresses++;
    }
    _ledState = buttonVal;
    _prevDebouncedButtonVal = buttonVal;
  }

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN, _ledState);
  _prevRawButtonVal = buttonVal;
}
```

https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L05-Arduino/RedBearDuoReadButtonDebounce

WORKING WITH SENSORS: LEARNING GOALS

How to use variable **resistive sensors,** including **voltage dividers,** basic **signal smoothing**

Putting it together: **making a simple theremin**

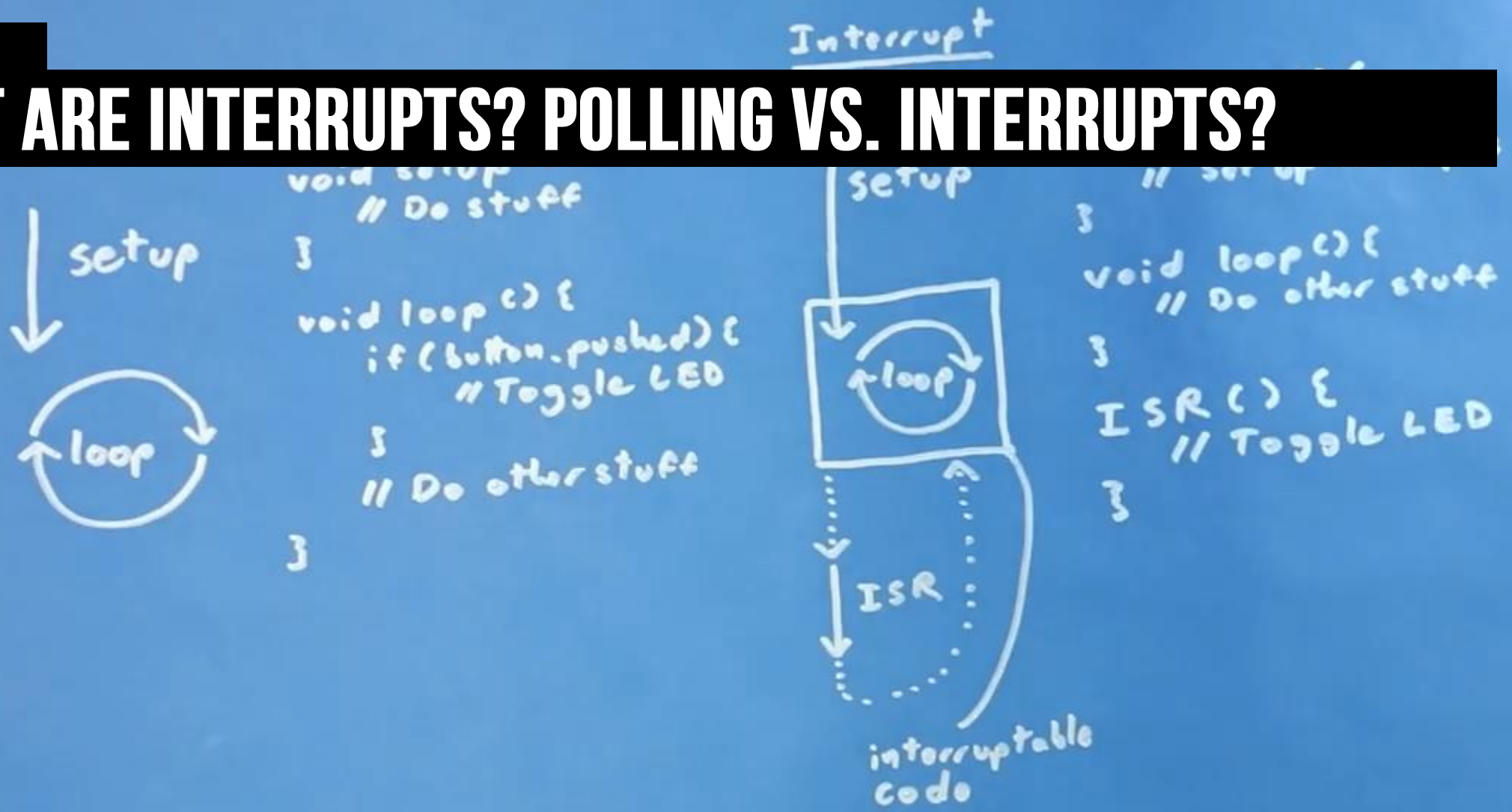
How to use **buttons**, including **pull-up** and **pull-down resistors**, and **debouncing**

Working with **interrupts**

Finally, the last concept I want to cover related to input (and most often digital input) is **interrupts**.

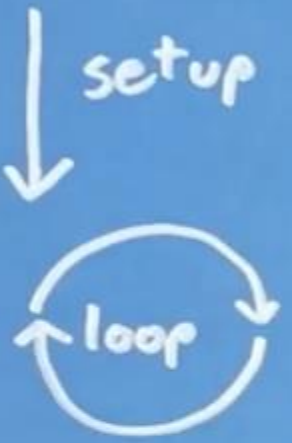
INTERRUPTS

WHAT ARE INTERRUPTS? POLLING VS. INTERRUPTS?



Source: Sparkfun Electronics Tutorial, https://youtu.be/J61_PKyWjxU

Polling




```
void setup() {
    // Do stuff
}

void loop() {
    if(button.pushed){
        // Toggle LED
    }
    // Do other stuff
}
```

Interrupt


```
void setup() {
    // Do stuff
    // Set up interrupts
}

void loop() {
    // Do other stuff
}

ISR() {
    // Toggle LED
}
```

WHAT ARE INTERRUPTS?

Interrupts allow us to **specify a block of code** (called interrupt service routines or ISRs) that should be **run in response to an event** (*e.g.*, responding to a button press)

Events can be **internal** (like timers) or **external** (like buttons)

Interrupt code **literally interrupts** the **execution** of our main loop

ISRs should be **very fast** and non-blocking

All **variables shared** between an **ISR** and **main** code should be prefixed by **volatile**

Source: https://docs.particle.io/reference/firmware/photon/#interrupts

attachInterrupt()

Specific a function to call when an external interrupt occurs. pinMode() must be called prior to calling attachInterrupt().

Syntax

```
attachInterrupt(pin, function, mode);

attachInterrupt(pin, function, mode, priority);
```

Parameters

`pin`: the input pin number

`function`: the callback function (must take no parameters and return nothing)

`mode`: defines how the interrupt should be triggered: CHANGE, RISING, FALLING

`priority (optional)`: defines the priority of the interrupt

Returns

A Boolean indicating whether the ISR was successfully attached (true) or not (false)

https://docs.particle.io/reference/firmware/photon/#interrupts

Can **interrupts interrupt other interrupts** (*i.e.*, an executing ISR)?
On most Arduino boards, nested interrupts are not enabled.

INTERRUPTS

BASIC EXAMPLE

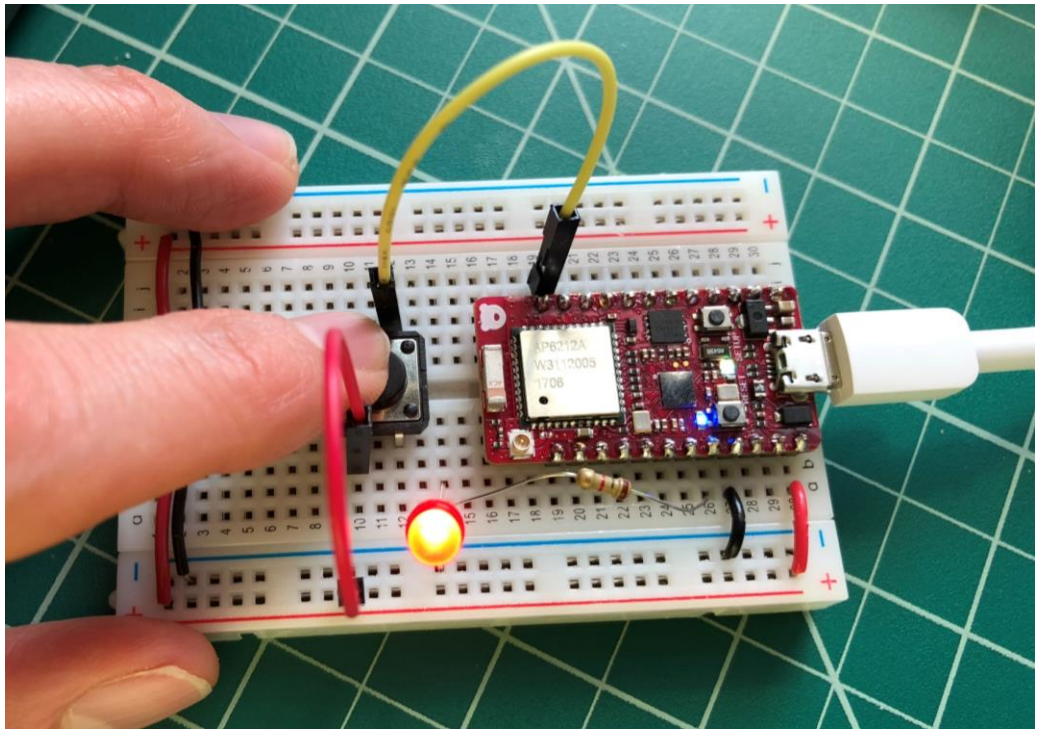
Setup interrupt to trigger on rising and falling edge of button press

Turn on and off LED accordingly

See: https://docs.particle.io/reference/firmware/photon/#interrupts

INTERRUPTS

MY SOLUTION



https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L06-Arduino/RedBearDuoButtonInterrupt

RedBearDuoButtonInterrupt §

```
/*
 * This example attaches an interrupt to D1, which calls a the function
 * toggle() whenever a RISING or FALLING edge is observed. Toggle turns on/off
 * the internal RedBear Duo LED (which is on D7).
 *
 *
 * By Jon Froehlich for CSE590
 * http://makeabilitylab.io
 *
 * Based on:
 *  - https://docs.particle.io/reference/firmware/photon/#interrupts
 *  - https://github.com/bjo3rn/idd-examples/blob/master/redbearduo/examples/button_ir
 */
SYSTEM_MODE(MANUAL);

// This is the internal LED pin on the Duo
const int LED_OUTPUT_PIN = D7;

// The official Particle Photon documentation states that external interrupts
// should work on all pins except for D0 and A5, see:
// https://docs.particle.io/reference/firmware/photon/#interrupts
const int BUTTON_INPUT_PIN = D8;

void toggle(void); // function declaration for interrupt

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(BUTTON_INPUT_PIN, INPUT_PULLDOWN);
  attachInterrupt(BUTTON_INPUT_PIN, toggle, CHANGE); //CHANGE, RISING or FALLING
}

void loop() {
  // intentionally empty
}

void toggle(){
  digitalWrite(LED_OUTPUT_PIN, digitalRead(BUTTON_INPUT_PIN));
}
```

TWO MORE EXAMPLES TO EXPLORE

Interrupt + debouncing example

RedBearDuoButtonInterruptWithDebouncing | Arduino 1.8.5

RedBearDuoButtonInterruptWithDebouncing

```
long _debounceDelay = 50;    // the debounce time in ms

// From the Particle Photon documentation:
//   "Typically global variables are used to pass data between an ISR and
//   the main program. To make sure variables shared between an ISR and the
//   main program are updated correctly, declare them as volatile."
// See: https://docs.particle.io/reference/firmware/photon/#interrupts
volatile int _ledState = LOW;

// Function declaration for interrupts
void toggle(void);
void isr(void);

// We will use a software timer to help us debounce.
// See: https://docs.particle.io/reference/firmware/photon/#software-timers
Timer timer(_debounceDelay, toggle, true); // oneshot timer so last param=true

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(BUTTON_INPUT_PIN, INPUT_PULLDOWN);

  // Hook up interrupt to button again but this time call the
  // ISR routine, which will start a timer on every rising edge
  // and after a set delay (_debounceDelay), the timer callback
  // is called (which is set to toggle())
  attachInterrupt(BUTTON_INPUT_PIN, isr, CHANGE); //CHANGE, RISING or FALLING
}

void loop() {
  // intentionally empty
}

// Called automatically when a button state change occurs
void isr(){
  noInterrupts(); // temporarily pause the interrupts
  if(digitalRead(BUTTON_INPUT_PIN) == HIGH){
    timer.resetFromISR(); //start or reset timer on every rising edge
  } else {
    timer.stopFromISR(); //stop on falling edge
  }
```

https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L06-Arduino/RedBearDuoButtonInterruptWithDebouncing

Software timer example

RedBearDuoSoftwareTimer | Arduino 1.8.5

RedBearDuoSoftwareTimer

```
/*
 * IMPORTANT: When working with the RedBear Duo, you must have this line of
 * code at the top of your program. The default state is SYSTEM_MODE(AUTOMATIC);
 * however, this puts the RedBear Duo in a special cloud-based mode that we
 * are not using. For our purposes, set SYSTEM_MODE(SEMI_AUTOMATIC) or
 * SYSTEM_MODE(MANUAL). See https://docs.particle.io/reference/firmware/photon/#system-modes
 */
SYSTEM_MODE(MANUAL);

// This is the internal LED pin on the Duo
const int LED_OUTPUT_PIN = D7;

// From the Particle Photon documentation:
//   "Typically global variables are used to pass data between an ISR and
//   the main program. To make sure variables shared between an ISR and the
//   main program are updated correctly, declare them as volatile."
// See: https://docs.particle.io/reference/firmware/photon/#interrupts
volatile int _ledState = LOW;
volatile int _seconds = 0;

void timer_callback(void); // function declaration for interrupt

Timer _timer(1000, timer_callback);

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  Serial.begin(9600);
  _timer.start();
}

void loop() {
  // intentionally empty
}

// called automatically by software timer
void timer_callback(){
  _ledState = !_ledState;
  digitalWrite(LED_OUTPUT_PIN, _ledState);
  Serial.println(++_seconds);
}
```

https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L06-Arduino/RedBearDuoSoftwareTimer

SCHEDULE TODAY: 6:30-9:20

06:30-06:55: Discussion of required reading led by Joe Wandyez

06:55-07:00: Optional discussion led by Abhay Rijhwani (link)

07:00-08:10: Physical Computing 3: Sensors

08:10-08:15: Break

08:15-09:20: Physical Computing 4: Android + Arduino via BLE