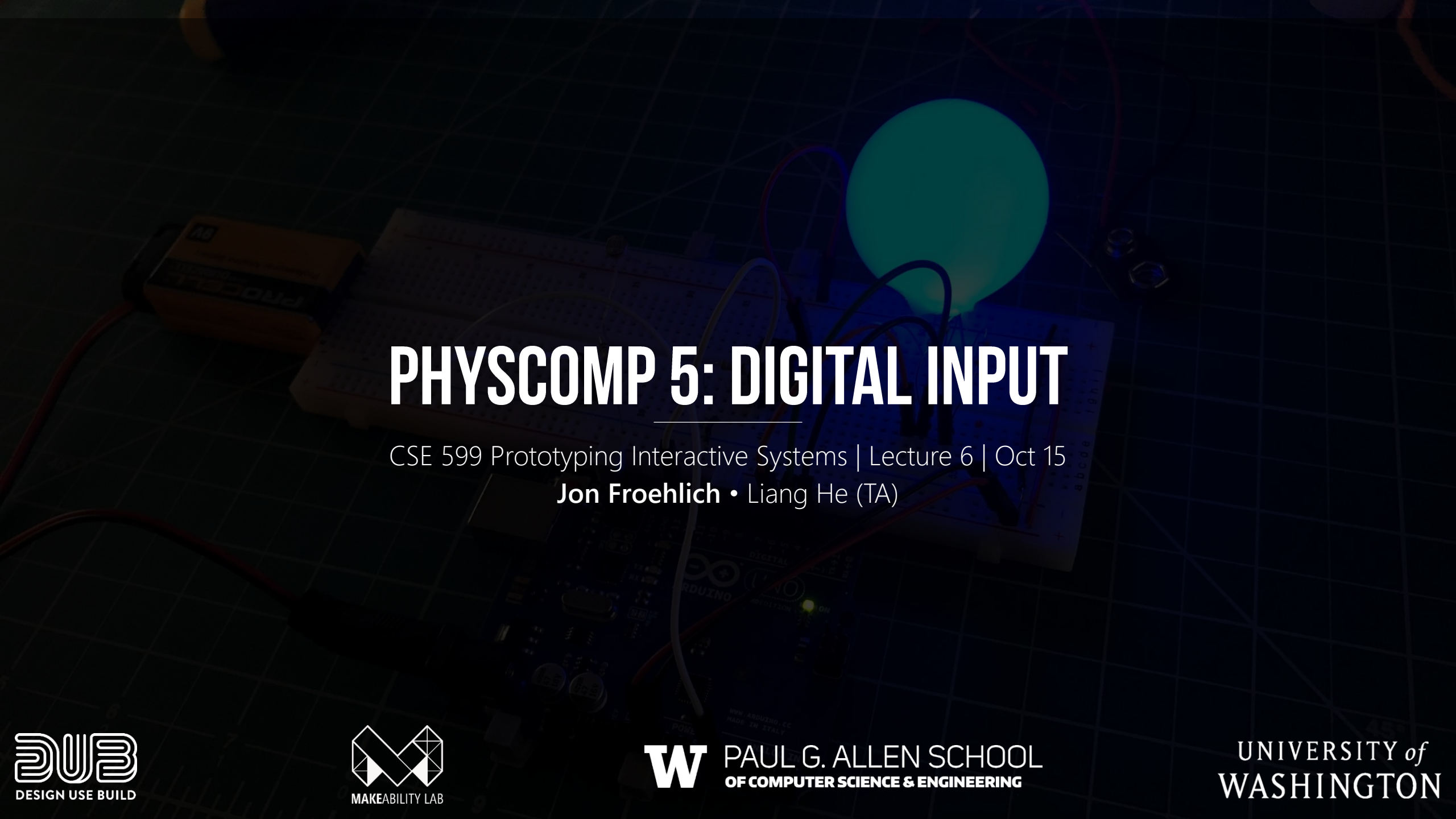


PHYSCOMP 5: DIGITAL INPUT

CSE 599 Prototyping Interactive Systems | Lecture 6 | Oct 15

Jon Froehlich • Liang He (TA)

DESIGN USE BUILD

MAKEABILITY LAB

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY of WASHINGTON

PHYSICAL COMPUTING 5

TODAY'S LEARNING GOALS

Working with **digital input**

How to hook up a **button**

What are **pull-up** and **pull-down resistors** and why use them

What is **debouncing** and how to implement it

What are **interrupts** and how to use them

(If time) **tilt switches, reed switches**

A1 SHARE OUTS + CRITIQUE

A1: Interactive Night Light

Published | Edit

Image Source: Richard Clarkson, Clouds

You are working for a design consultancy hired to rethink and redesign interactive ambient light's for the 21st century. You have been asked to rapidly prototype some designs that are responsive to the user and the environment.

Learning Goals


LO-FI INPUT

A1: INTERACTIVE NIGHT LIGHT

LO-FI INPUT

A1 EXAMPLE

WOBBLE GARDEN

Designer: Robin Baumgarten, http://wobblylabs.com/projects/wobblegarden

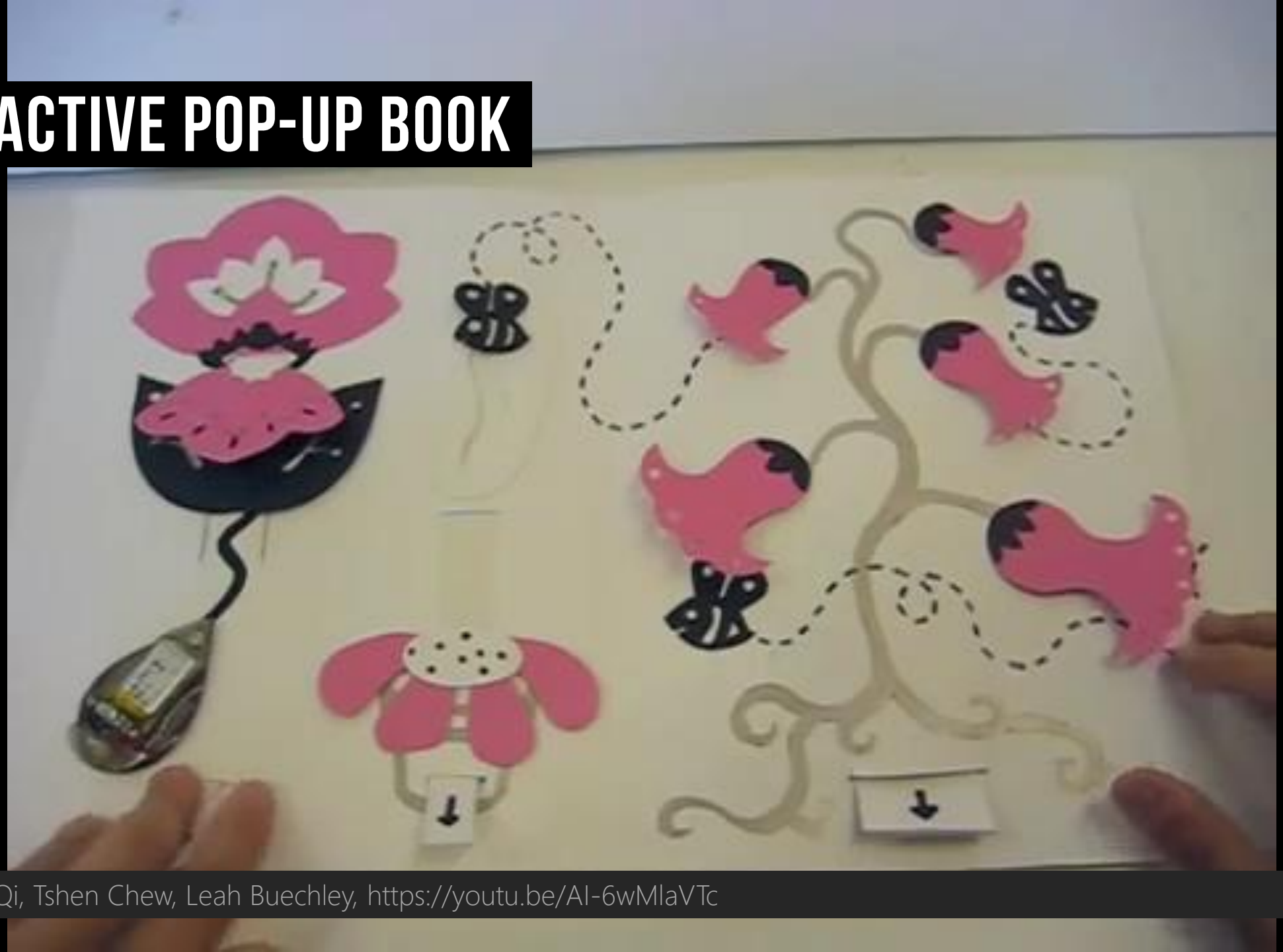

Garden

A1: INTERACTIVE NIGHT LIGHT

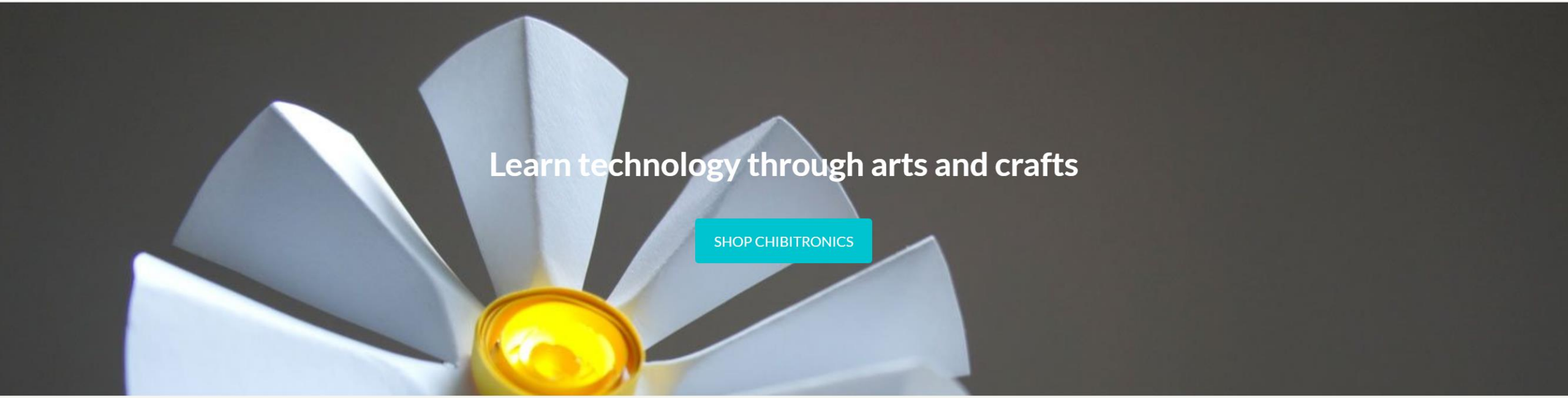
LO-FI INPUT

A1 EXAMPLE

INTERACTIVE POP-UP BOOK



Designers: Jie Qi, Tshen Chew, Leah Buechley, https://youtu.be/AI-6wMlaVTc



Learn technology through arts and crafts

SHOP CHIBITRONICS

What is Chibitronics?

Chibitronics makes circuit stickers and other tools for **paper circuits**, which blends circuit building and programming with arts and crafts. It's a friendly way to learn, design and create your own electronics.

A1: INTERACTIVE NIGHT LIGHT

LO-FI INPUT

A Wall

Paint

Magnets

A1 EXAMPLE

INTERACTIVE WALL

Designer: High-Low Tech Group at MIT Media Lab, https://youtu.be/XrNz9deYIJU


Hannah Perner-Wilson

A2: Fabricated IoT Light

Published | Edit

Image caption: The *Tangible Interactive Computing* Top Maker Award from CMSC838f, Spring 2015 designed by Jon Froehlich based on the Holocron Nightlight by CMSC838f student Philip Dasler.

Overview

In this assignment, you will design and fabricate a custom 3D-printed interactive night light, which:

- Uses the Arduino ESP32 Feather (not the Leonardo). See setup instructions here. We have also provided a full end-to-end demo with source code and instructions of a simple IoT application that lights an LED in correspondence to JSON weather values read from openweathermap.org.
- Lights up in response to some real-time data from the Internet (*e.g.*, current weather, Twitter notifications, number of unread emails)
- Creatively diffuses the light (*e.g.*, playfully or elegantly)
- Fully encloses your Arduino Feather and electronics
- Optionally exposes user input controls (*e.g.*, a slider to set brightness)

The specific model is up to you but should include:

- a mounting stand for the internal electronics;
- a carefully measured and tightly fit input slot for the USB micro cable to power your design;
- and similarly well-designed fittings for any input controls you (optionally) want to expose.
- the design should not be larger than 10x10x5 cm and Cura's estimated print time should be ~8hrs or less.

You may need to design and print a multi-part model, which can be reassembled into a full form (*e.g.*, similar to this

Overview

In this assignment, you will design and fabricate a custom 3D-printed interactive night light, which:

- Uses the Arduino ESP32 Feather (not the Leonardo). See setup instructions here . We have also provided a full end-to-end demo with source code and instructions of a simple IoT application that lights an LED in correspondence to JSON weather values read from openweathermap.org.
- Lights up in response to some real-time data from the Internet (*e.g.*, current weather, Twitter notifications, number of unread emails)
- Creatively diffuses the light (*e.g.*, playfully or elegantly)
- Fully encloses your Arduino Feather and electronics
- Optionally exposes user input controls (*e.g.*, a slider to set brightness)

The specific model is up to you but should include:

- a mounting stand for the internal electronics;
- a carefully measured and tightly fit input slot for the USB micro cable to power your design;
- and similarly well-designed fittings for any input controls you (optionally) want to expose.
- the design should not be larger than 10x10x5 cm and Cura's estimated print time should be ~8hrs or less.

You may need to design and print a multi-part model, which can be reassembled into a full form (*e.g.*, similar to this Arduino case --but, of course, your night light will have to be bigger to hold all of the components).

Overview

In this assignment, you will design and fabricate a custom 3D-printed interactive night light, which:

- Uses the Arduino ESP32 Feather (not the Leonardo). See setup instructions here . We have also provided a full end-to-end demo with source code and instructions of a simple IoT application that lights an LED in correspondence to JSON weather values read from openweathermap.org.
- Lights up in response to some real-time data from the Internet (*e.g.*, current weather, Twitter notifications, number of unread emails)
- Creatively diffuses the light (*e.g.*, playfully or elegantly)
- Fully encloses your Arduino Feather and electronics
- Optionally exposes user input controls (*e.g.*, a slider to set brightness)

The specific model is up to you but should include:

- a mounting stand for the internal electronics;
- a carefully measured and tightly fit input slot for the USB micro cable to power your design;
- and similarly well-designed fittings for any input controls you (optionally) want to expose.
- the design should not be larger than 10x10x5 cm and Cura's estimated print time should be ~8hrs or less.

You may need to design and print a multi-part model, which can be reassembled into a full form (*e.g.*, similar to this Arduino case --but, of course, your night light will have to be bigger to hold all of the components).

Overview

In this assignment, you will design and fabricate a custom 3D-printed interactive night light, which:

- Uses the Arduino ESP32 Feather (not the Leonardo). See setup instructions here . We have also provided a full end-to-end demo with source code and instructions of a simple IoT application that lights an LED in correspondence to JSON weather values read from openweathermap.org.
- Lights up in response to some real-time data from the Internet (*e.g.*, current weather, Twitter notifications, number of unread emails)
- Creatively diffuses the light (*e.g.*, playfully or elegantly)
- Fully encloses your Arduino Feather and electronics
- Optionally exposes user input controls (*e.g.*, a slider to set brightness)

The specific model is up to you but should include:

- a mounting stand for the internal electronics;
- a carefully measured and tightly fit input slot for the USB micro cable to power your design;
- and similarly well-designed fittings for any input controls you (optionally) want to expose.
- the design should not be larger than 10x10x5 cm and Cura's estimated print time should be ~8hrs or less.

You may need to design and print a multi-part model, which can be reassembled into a full form (*e.g.*, similar to this Arduino case --but, of course, your night light will have to be bigger to hold all of the components).

A2 INSPIRATIONS

BLOSSOMING LAMP

Designer: Emmett Lalish, https://www.thingiverse.com/thing:37926

A2 INSPIRATIONS

MUSHROOM LAMP

https://ruihetoy.com/%20product-view/3d-printing-mushroom-lamp/

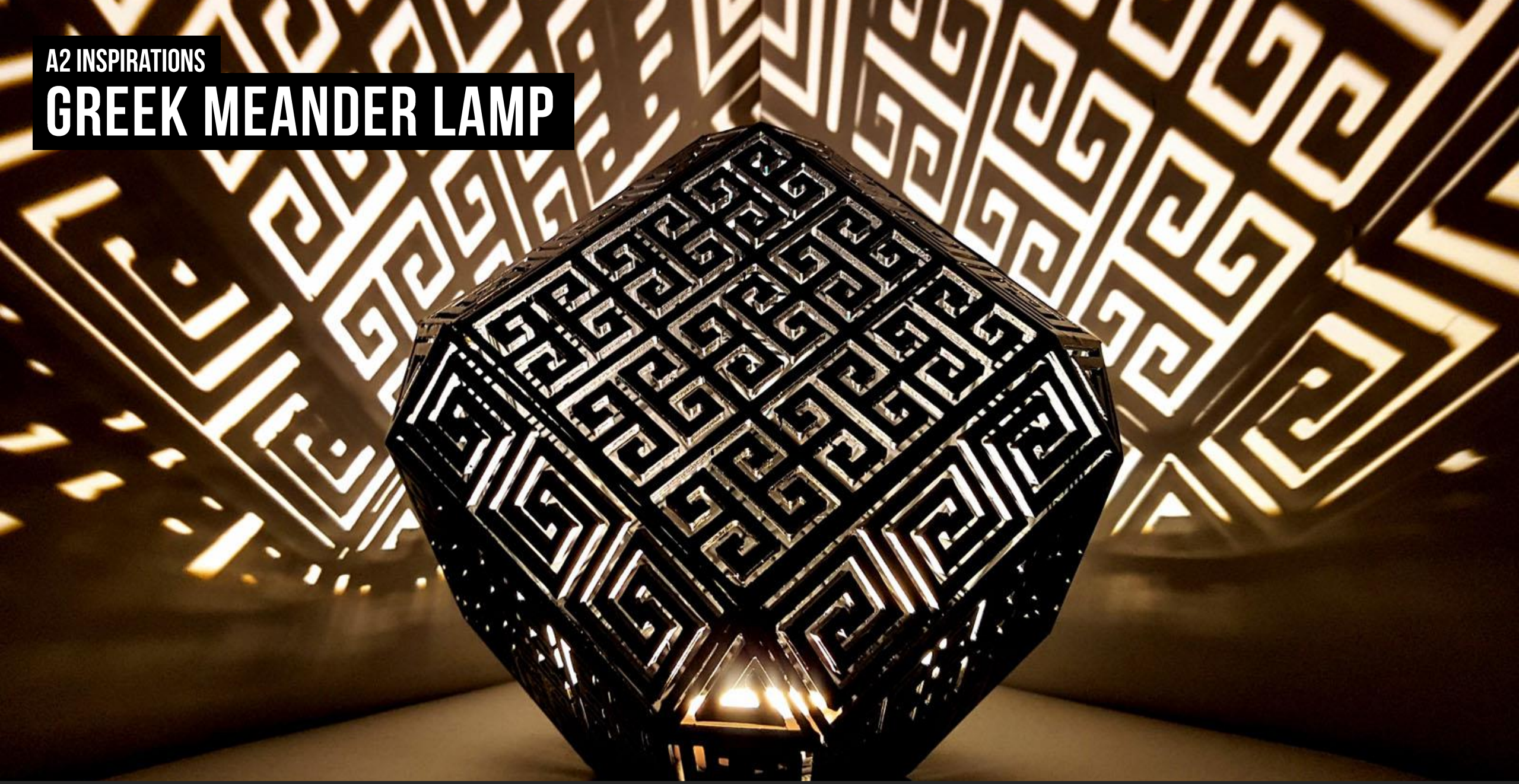
A2 INSPIRATIONS

MOON LIGHT

Designer: Gr4f1n, https://www.thingiverse.com/thing:3285515

A2 INSPIRATIONS

GREEK MEANDER LAMP



Designer: Hultis, https://www.thingiverse.com/thing:3434805

Designer: Andrew Reynolds, areynolds15, https://www.thingiverse.com/thing:857452

A2 INSPIRATIONS

LITHOPHANE

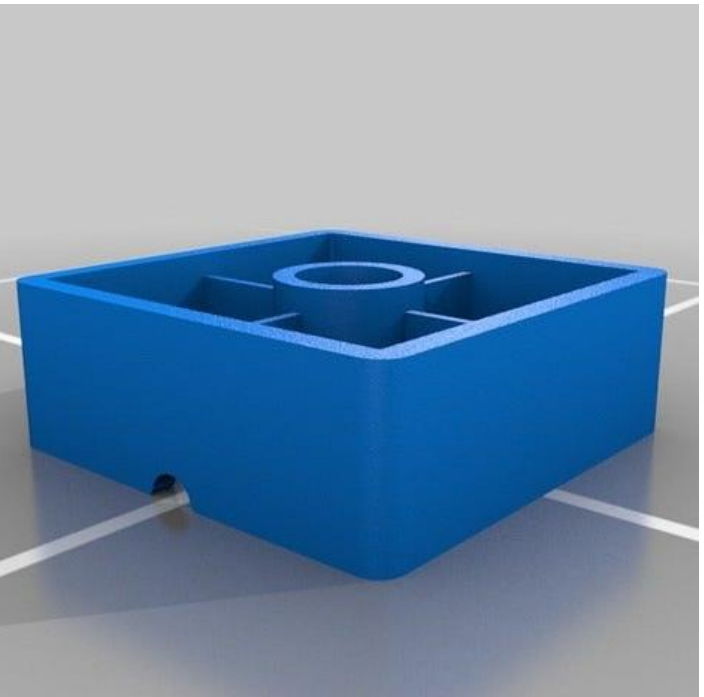
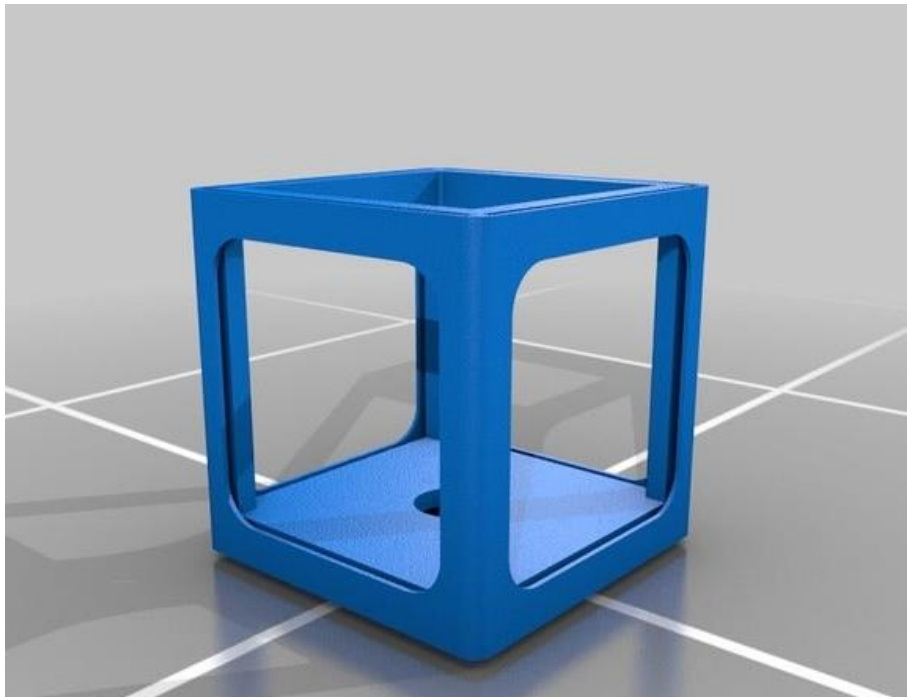
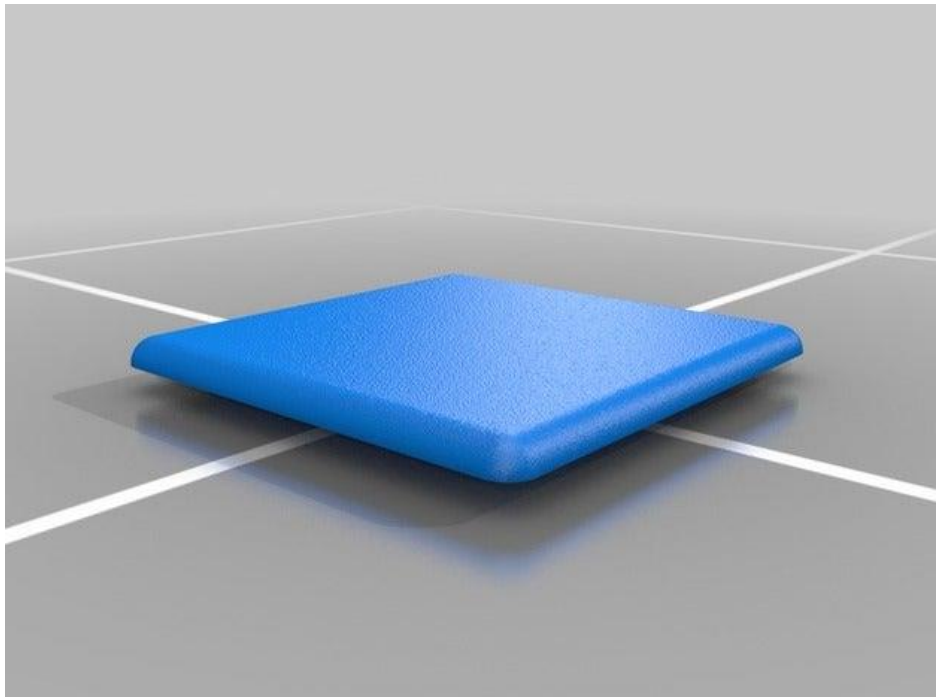
Source: https://www.behance.net/gallery/65748563/Lithophane-Lamps

A2 INSPIRATIONS

LITHOPHANE

Designer: tsnell168, https://www.thingiverse.com/thing:3630997



A2 INSPIRATIONS

LITHOPHANE

Designer: Beth Lewis Williams, https://inhabitat.com/beth-lewis-williams-beautiful-lithophane-lamps-blend-3d-printing-with-19th-century-craft/

A2 INSPIRATIONS

HOLOCRON NIGHTLIGHT

Designer: Philip Dasler, https://www.instructables.com/id/Holocron-Nightlight/

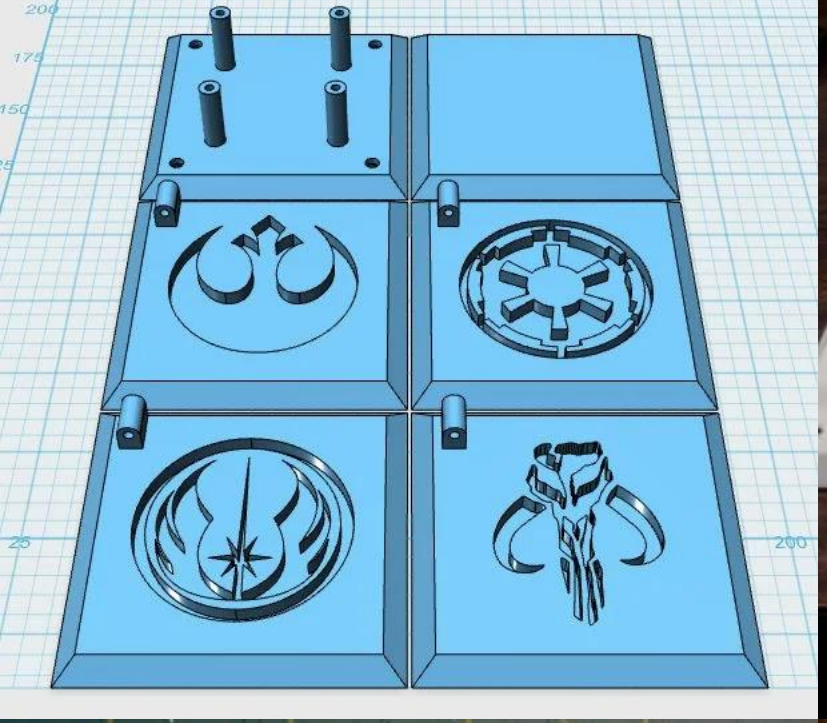
Holocron Nightlight

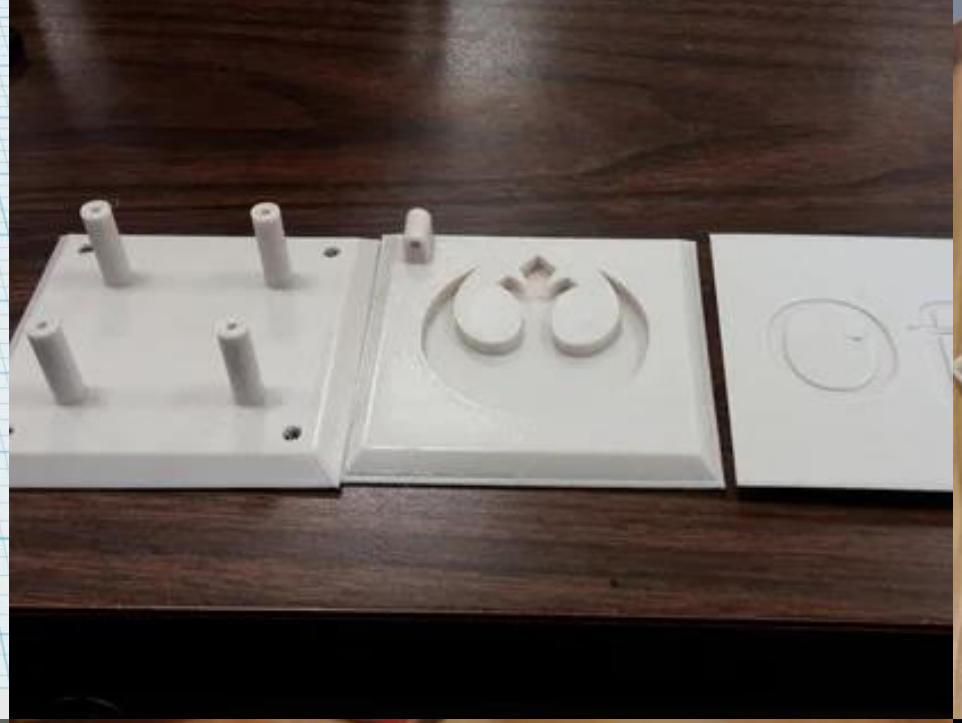
By daslerpc in 3D Printing 4,807 139 3 Featured

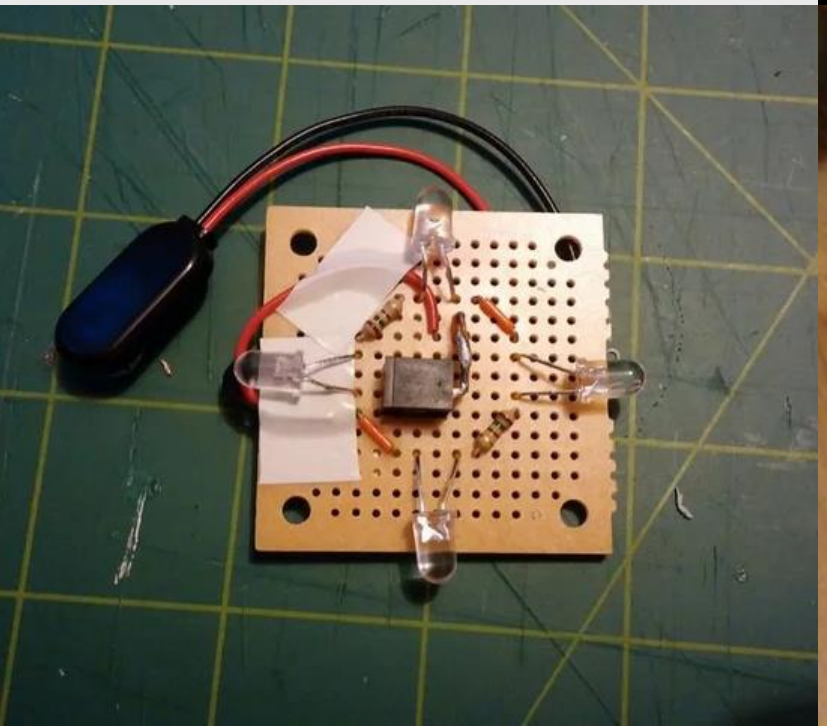
Published May 22nd, 2015

Download Favorited

This nightlight appears, at first, to be nothing but a simple cube. Written on one side is the word "Off" and on the other is, as one might expect, is the word "On". By turning the cube upside down, you activate the light inside and reveal the cubes secret!Hidden within the cube are the emblems of four factions from the Star Wars universe. These emblems are inset on the inside faces of the four side panels of the cube, making the plastic thinner and allowing the light to shine through. A simple circuit consisting mainly of a tilt switch and four LEDs is all there is to this simple nightlight.

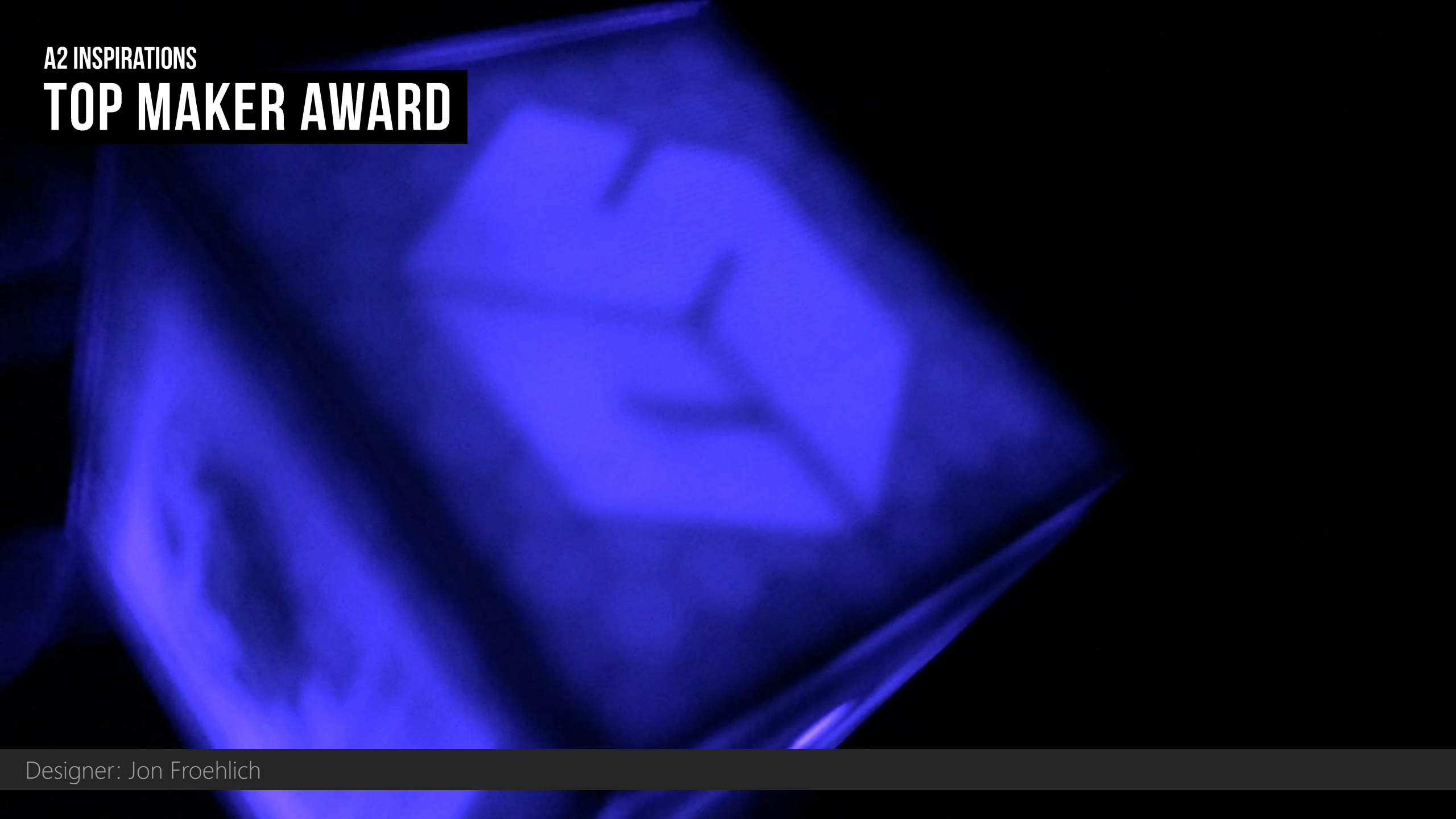




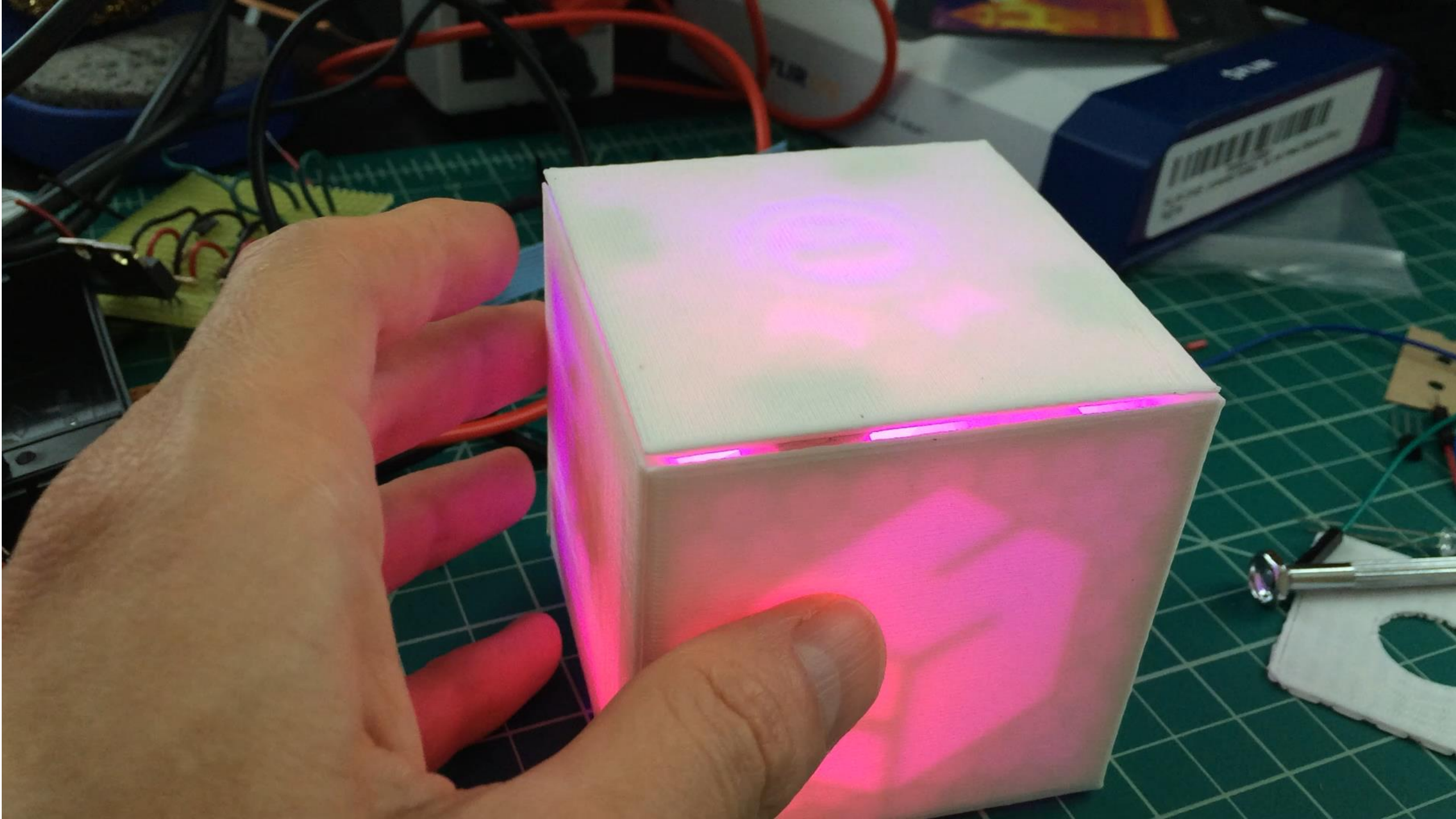


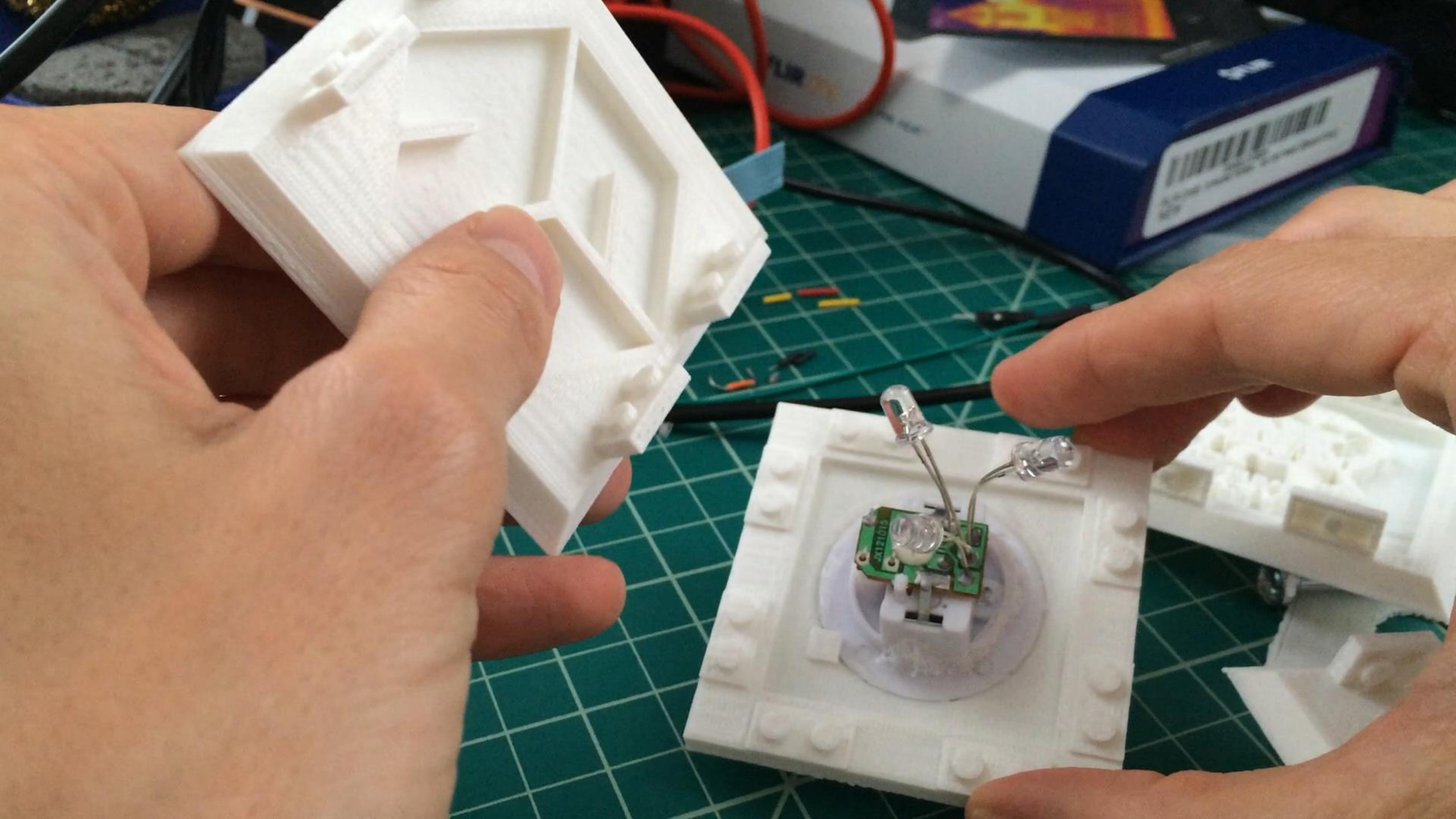
A2 INSPIRATIONS

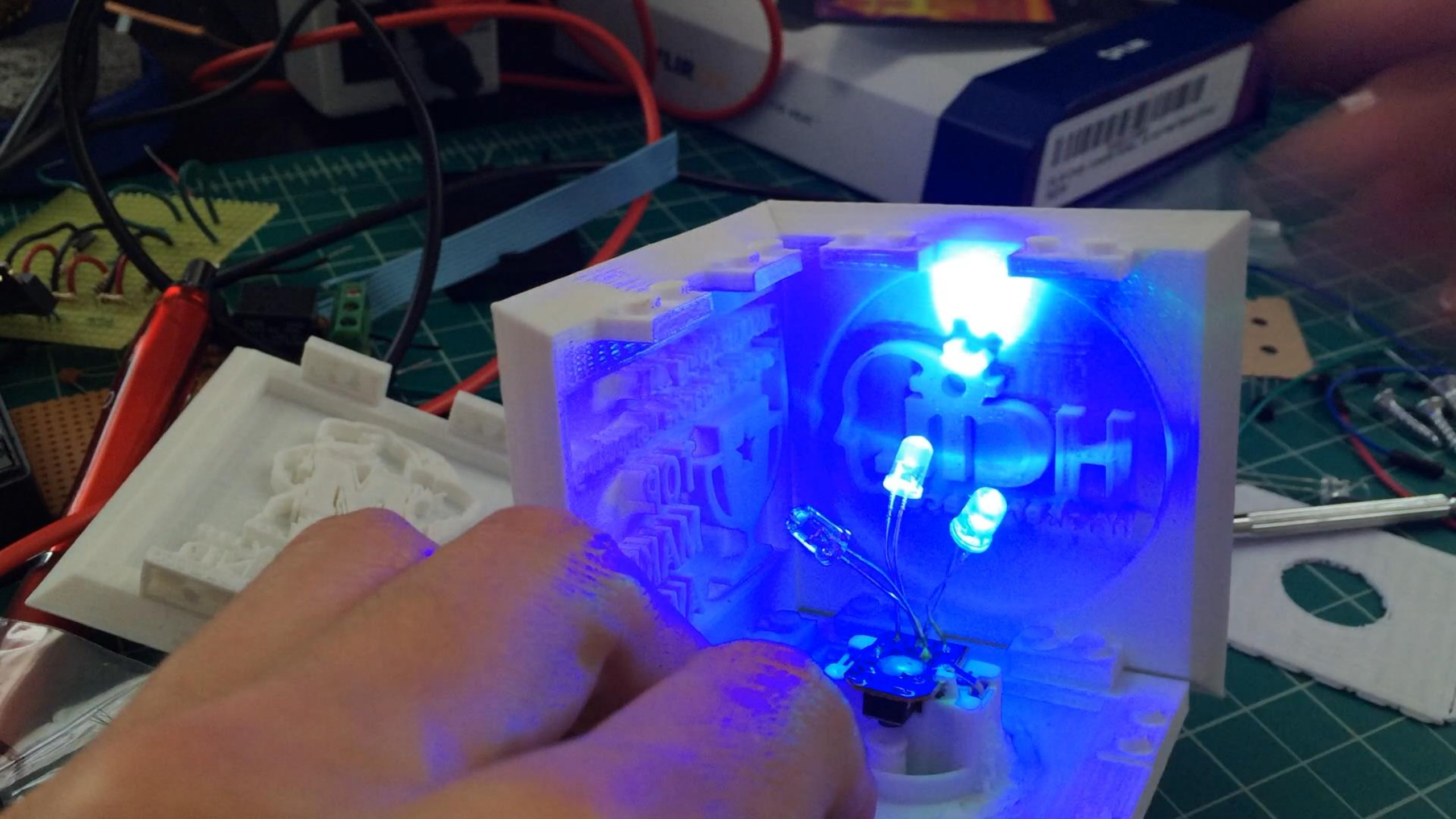
TOP MAKER AWARD

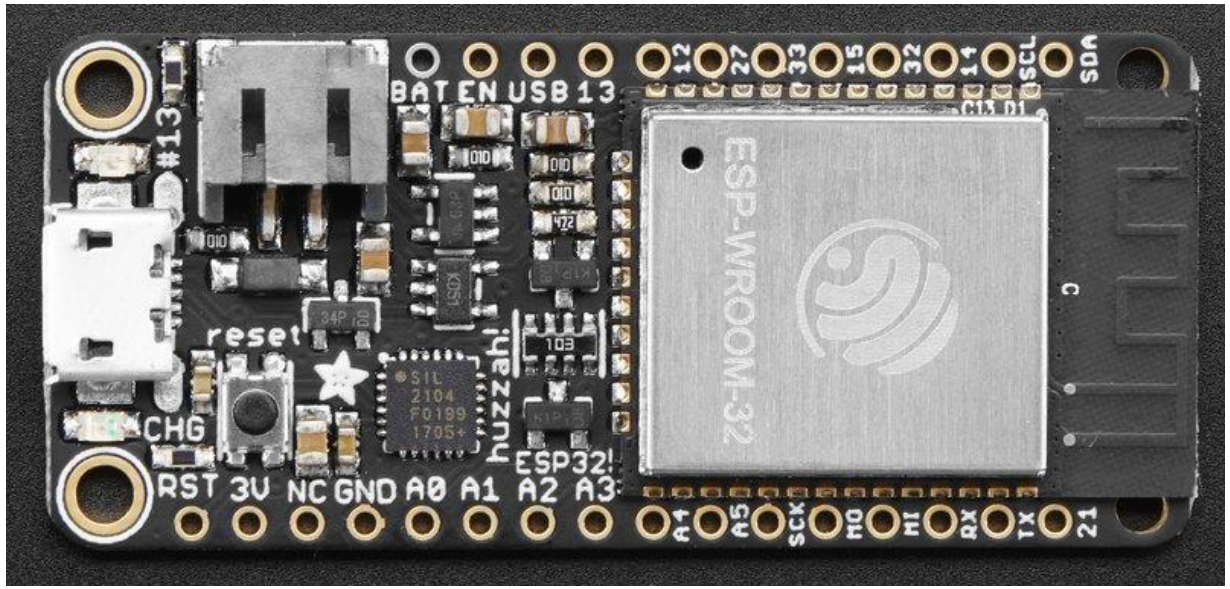


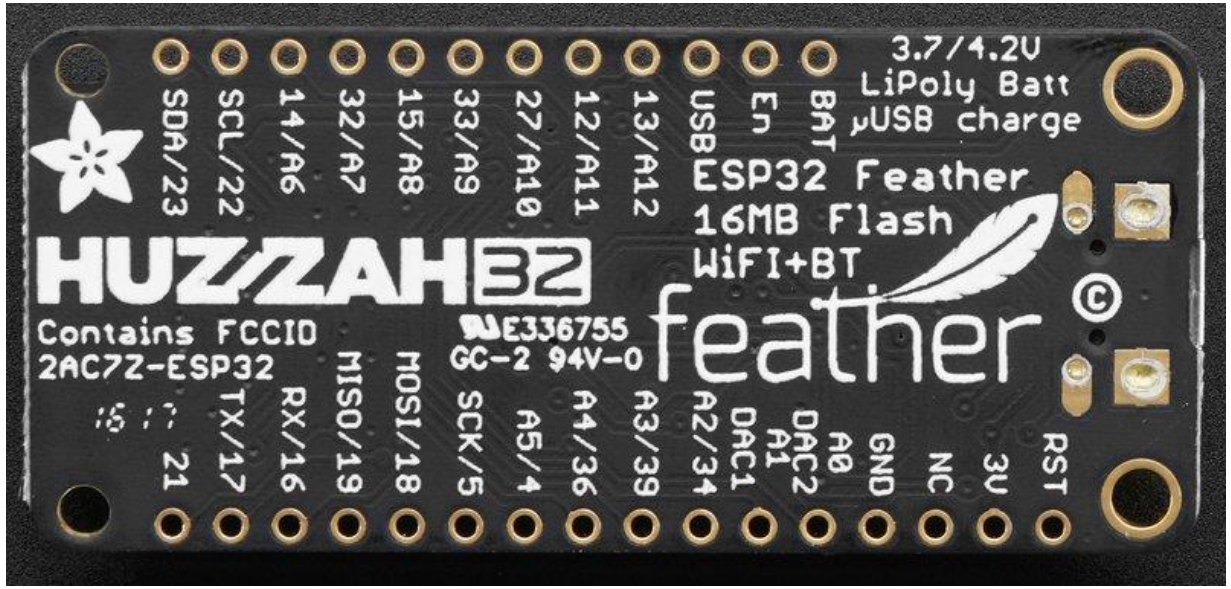
Designer: Jon Froehlich








BAT EN USB 13
12 27 33 15 32 14 SCL SDA
#13
reset
CHG
ESP32!
RST 3V NC GND A0 A1 A2 A3
A4 A5 SCK MO MI RX TX 21
ESP-WROOM-32


SDA/23 SCL/22 14/A6 32/A7 15/A8 33/A9 27/A10 12/A11 13/A12 USB En BAT
3.7/4.2V
LiPoly Batt
µUSB charge
ESP32 Feather
16MB Flash
WiFI+BT
HUZZAH32
Contains FCCID
2AC7Z-ESP32
E336755
QC-2 94V-0
feather
21 TX/17 RX/16 MISO/19 MOSI/18 SCK/5 A5/4 A4/36 A3/39 A2/34 DAC1 DAC2 A1 A0 GND NC 3V RST

ADAFRUIT HUZZAH32 ESP32

ADAFRUIT FEATHER HUZZAH32 PIN LAYOUT

Output from 3.3V regulator. Supplies ~500mA but ESP32 uses half. — 3V

Common GND for all power/logic — GND

GPIO26 — DAC2 — A0 ADC2

GPIO25 — DAC1 — A1 ADC2

GPI34 — A2 ADC1

GPI39 — A3 ADC1

GPI36 — A4 ADC1

GPI04 — A5 ADC2

I2C/SPI

I2C/SPI

I2C/SPI

Unlike Uno/Leonard, not connected to USB/serial converter — Serial RX

Unlike Uno/Leonard, not connected to USB/serial converter — Serial TX

GPIO21

BAT — Positive voltage from JST jack from optional lipoly battery

EN — 3.3V voltage regulator. Connect to GND to disable

USB — Positive voltage from micro USB if connected

A12 ADC2 — GPIO13 — RED LED

A11 ADC2 — GPIO12 — Has built-in pull-down resistor

A10 ADC2 — GPIO27

A09 ADC1 — GPIO33

A08 ADC2 — GPIO15

A07 ADC1 — GPIO32

A06 ADC1 — GPIO14

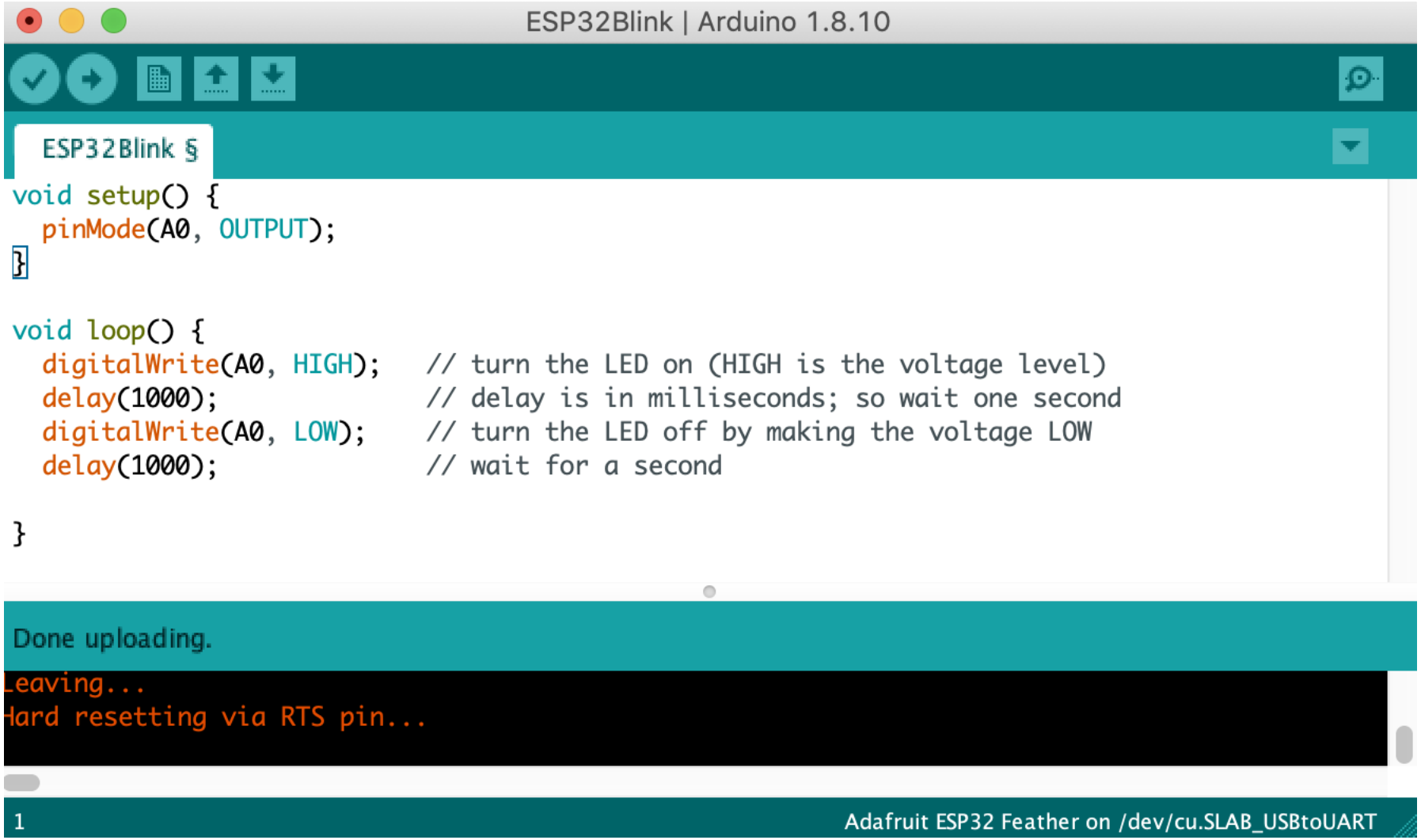
I2C/SPI

I2C/SPI

SOME GOTCHAS

- A2, A3, and A4 are **input only**.
- You can only read analog inputs on **ADC1** once WiFi has started

Figure by Jon Froehlich based on https://learn.adafruit.com/adafruit-huzzah32-esp32-feather/pinouts

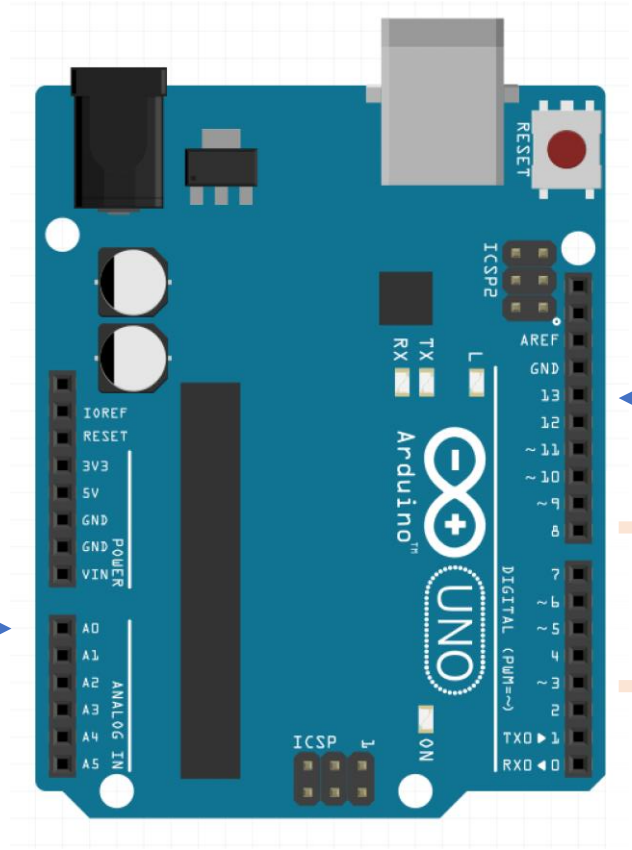

ESP32Blink | Arduino 1.8.10
ESP32Blink §
void setup() {
pinMode(A0, OUTPUT);
}
void loop() {
digitalWrite(A0, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // delay is in milliseconds; so wait one second
digitalWrite(A0, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
Done uploading.
Leaving...
Hard resetting via RTS pin...
1
Adafruit ESP32 Feather on /dev/cu.SLAB_USBtoUART

DIGITAL AND ANALOG OUTPUT

Now we are going to switch from talking about output to talking about input

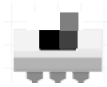
Analog Input on A0

Reads in any value between 0V or 5V using the analogRead function. In this case, the value of a photocell



Digital Input on pin 13

Reads in a digital input signal (anything below 2.5V converted to LOW, anything above 2.5V converted to HIGH). In this case, the value of switch.



Digital Output on pin 8

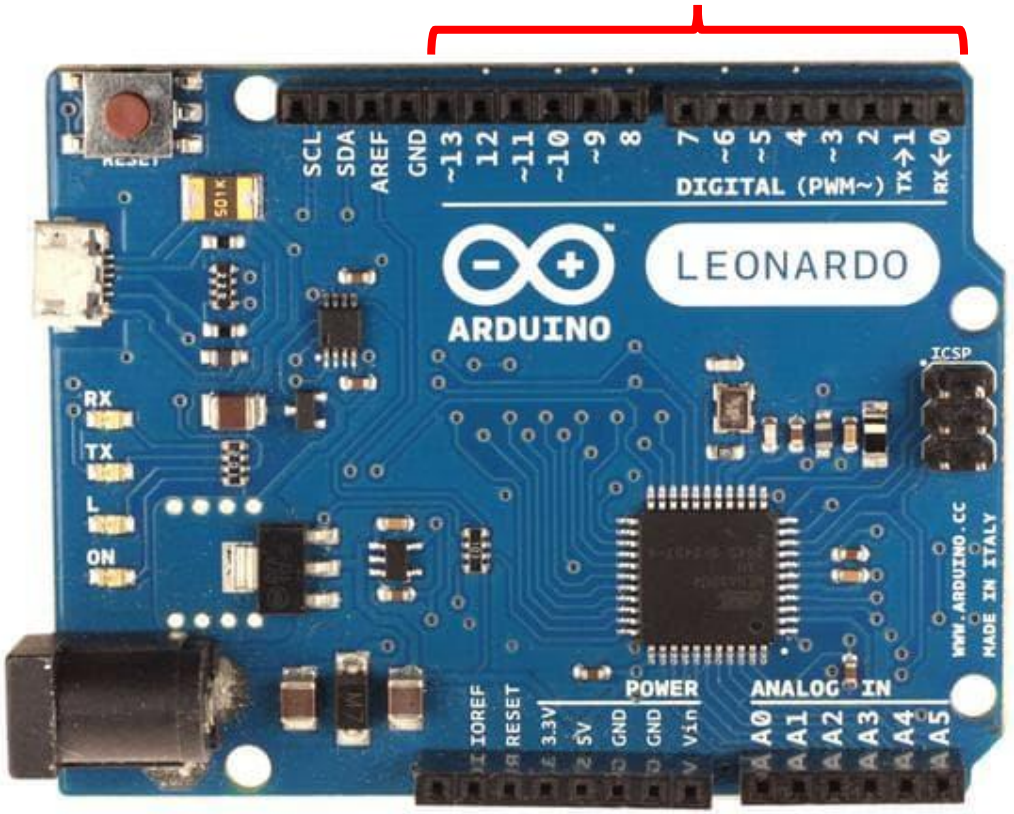
Writes out 0V or 5V using digitalWrite function. In this case, turning on and off an LED.

Analog Output on pin 3

Writes out any value between 0V or 5V using analogWrite function. In this case, vibrating a motor (where strength of vibration proportional to voltage). Only pins with a tilde ~ can be analog outputs.

ARDUINO LEONARDO DIGITAL INPUT

14 digital I/O pins: these pins can be used for either input or output (set the mode using the pinMode function).

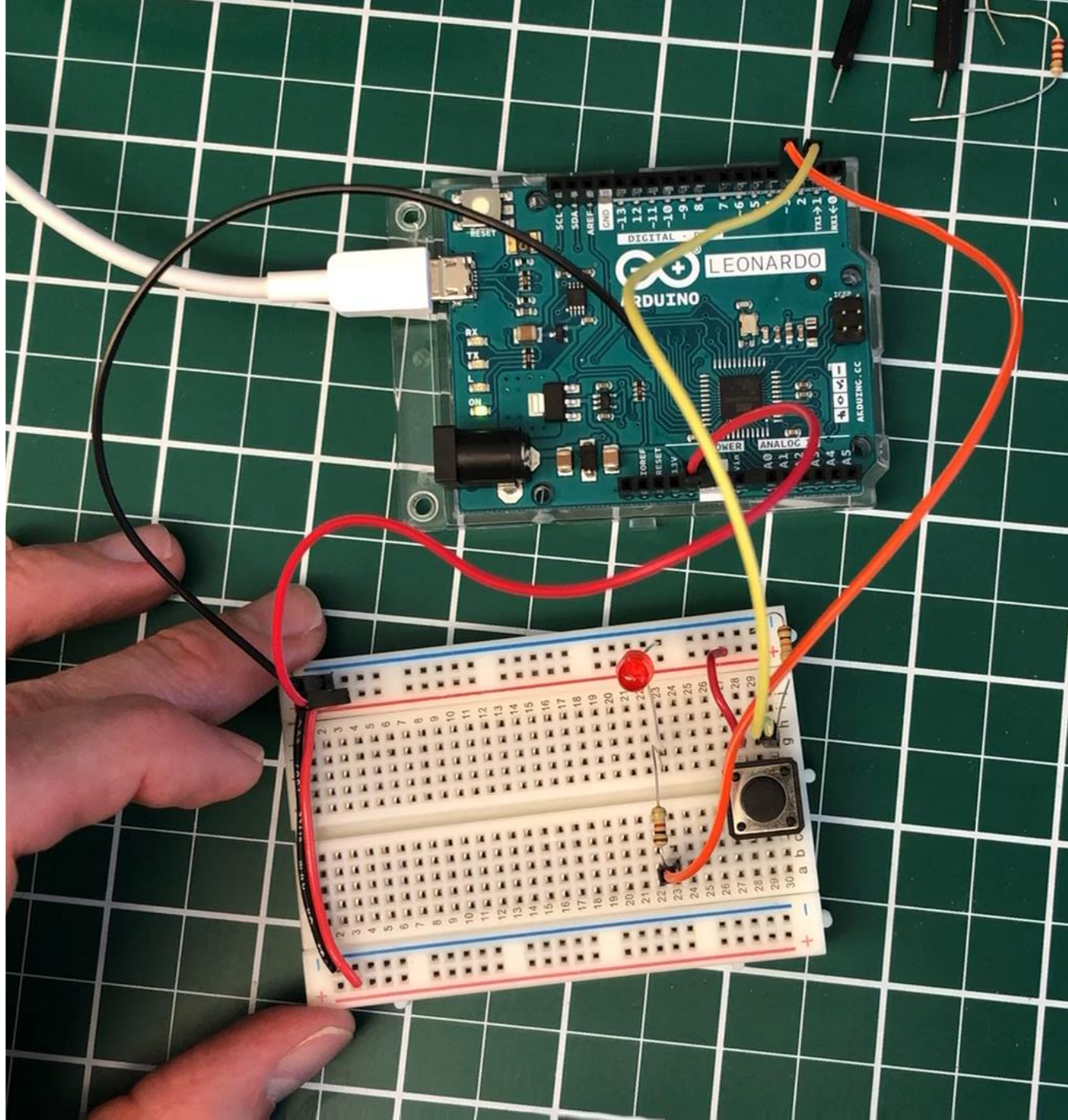


Source: https://www.arduino.cc/en/Guide/ArduinoLeonardoMicro; https://store.arduino.cc/usa/Leonardo; https://learn.sparkfun.com/tutorials/arduino-comparison-guide/atmega32u4-boards

DIGITAL INPUT

ACTIVITY: TURN ON LED USING A BUTTON AND DIGITAL INPUT

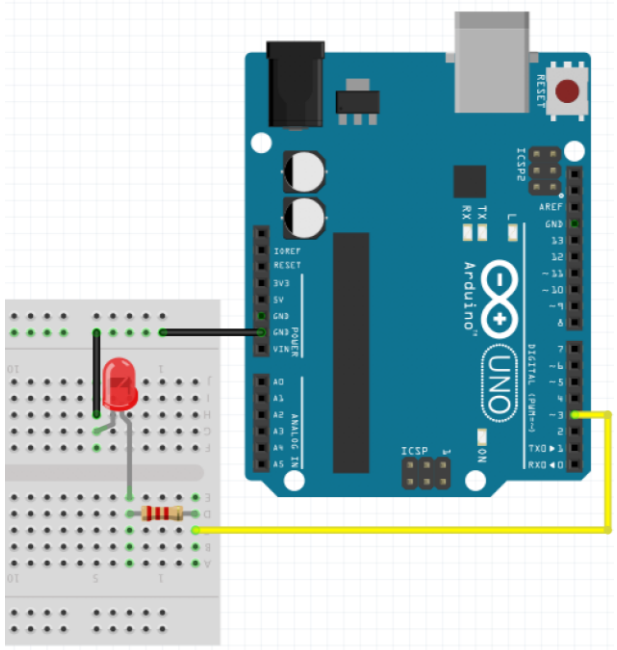
Design circuit + software to read the button value on D2 and turn on an LED when D2==HIGH and off the LED when D2==LOW

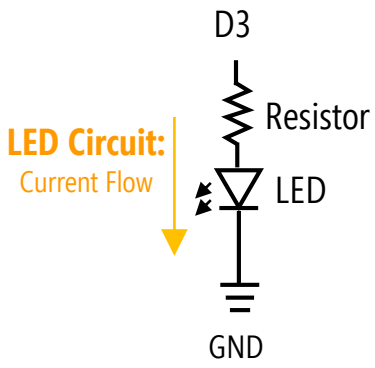


DIGITAL INPUT

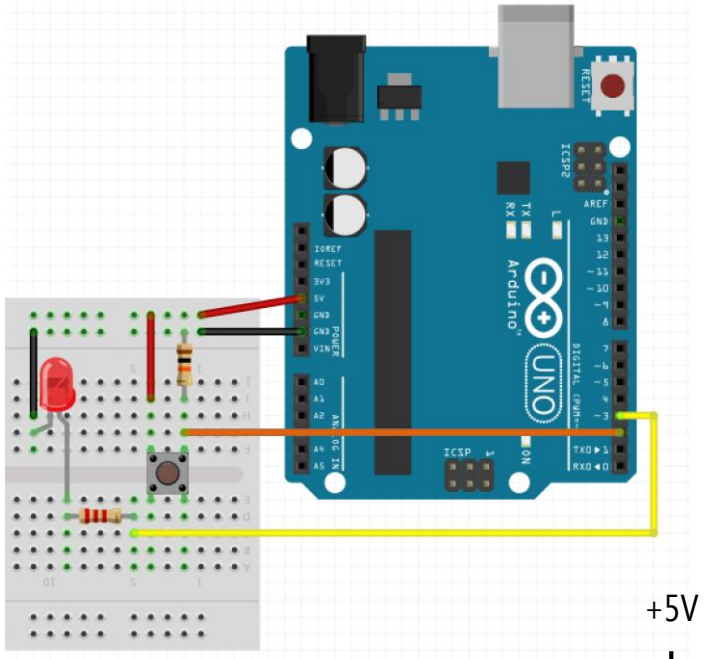
TURN ON/OFF LED WITH BUTTON

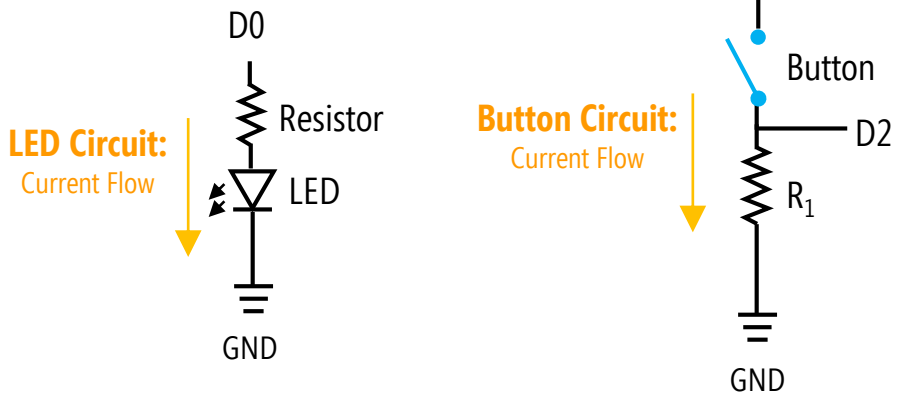
Old Circuit: Flash LED on/off via the pin 3





New Circuit: Turn on/off LED with button

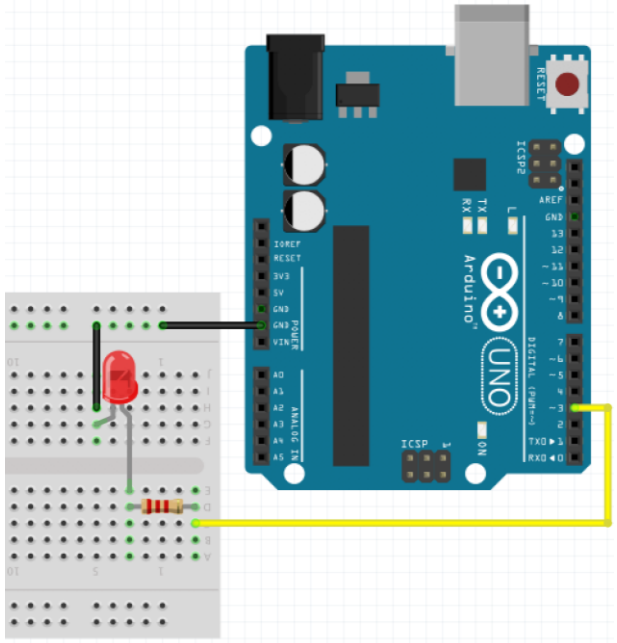


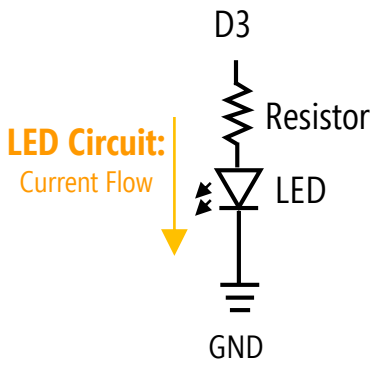


DIGITAL INPUT

TURN ON/OFF LED WITH BUTTON

Old Circuit: Flash LED on/off via the pin 3

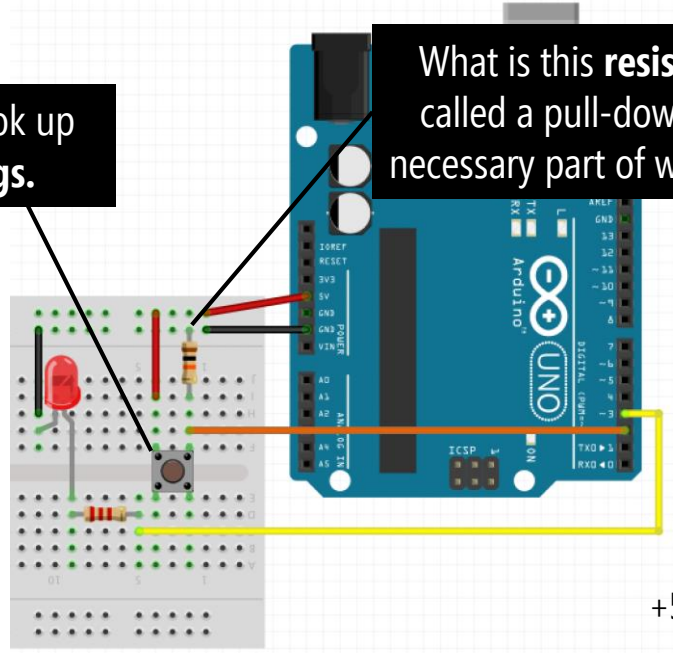


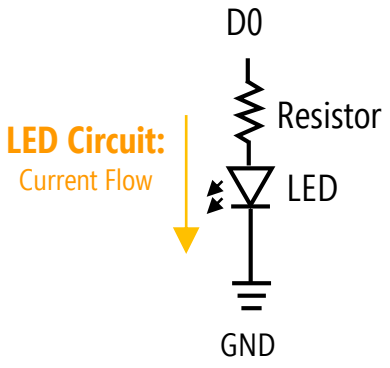


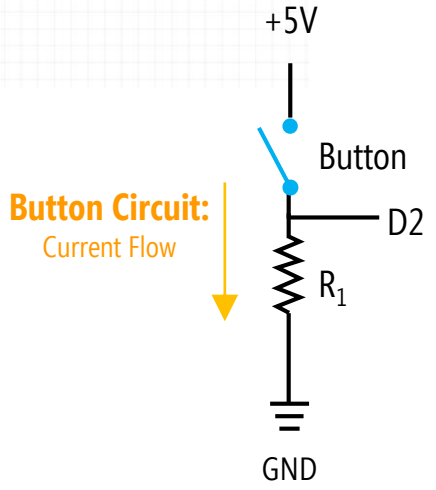
New Circuit: Turn on/off LED with button

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.



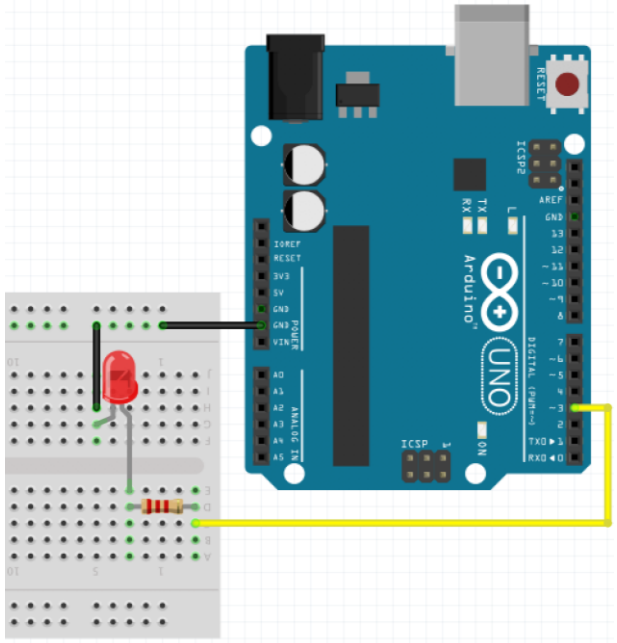


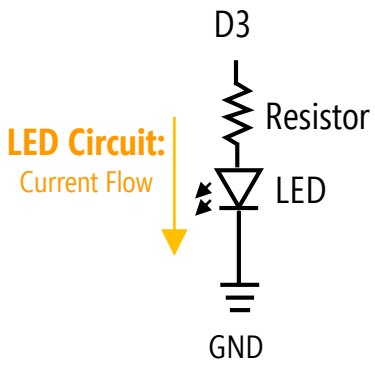


DIGITAL INPUT

TURN ON/OFF LED WITH BUTTON

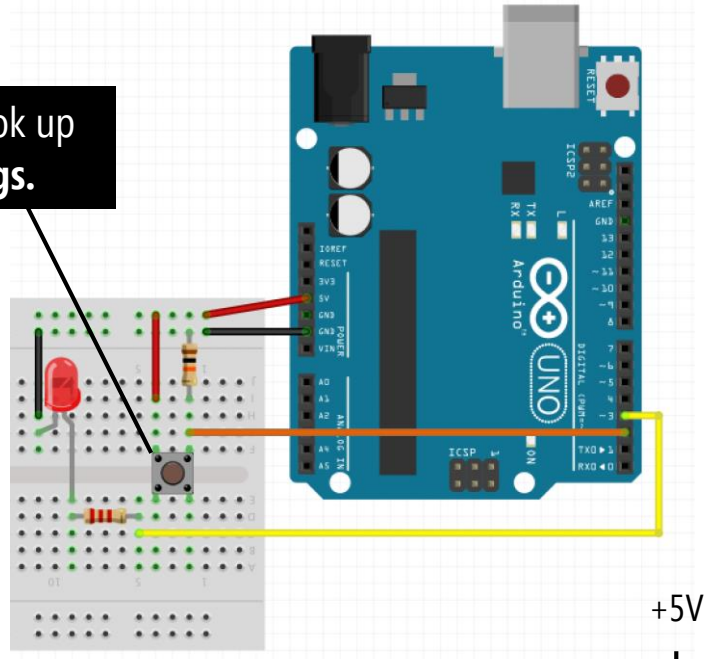
Old Circuit: Flash LED on/off via the pin 3

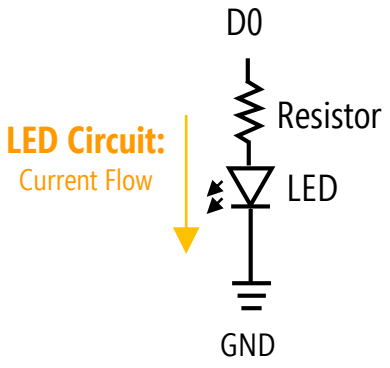


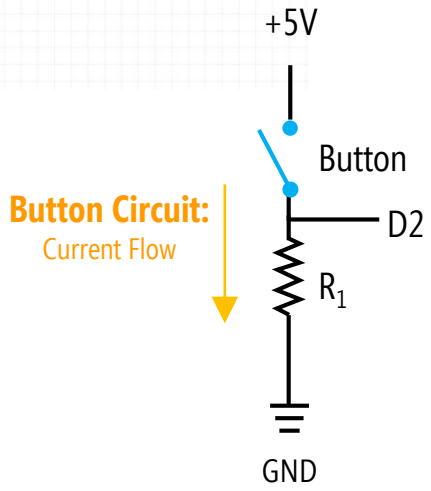


How do we know how to hook up this button? It has **four legs.**

New Circuit: Turn on/off LED with button







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□□□**

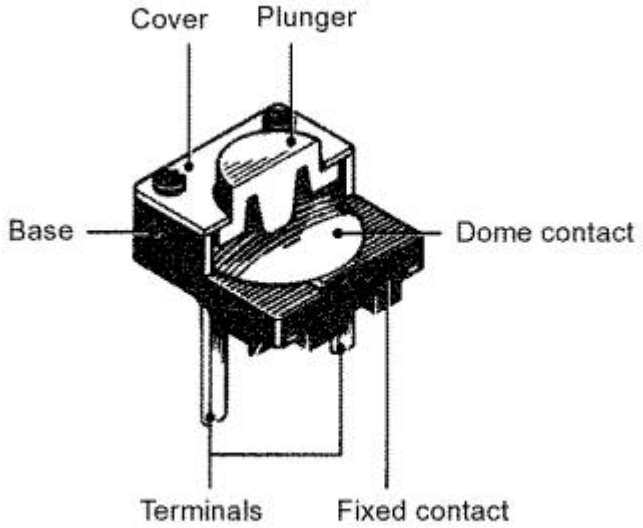
6 x 6 mm type

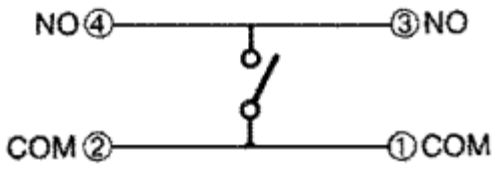
Type	Plunger	Switch height x pitch	Operating force	Model			
				Without ground terminal		With ground terminal	
				Bags	Sticks*	Bags	Sticks*
Standard	Flat	4.3 x 6.5 mm	General-purpose: 100 g	B3F-1000	B3F-1000S	B3F-1100	B3F-1100S
			150 g	B3F-1002	B3F-1002S	B3F-1102	B3F-1102S
			High-force: 260 g	B3F-1005	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

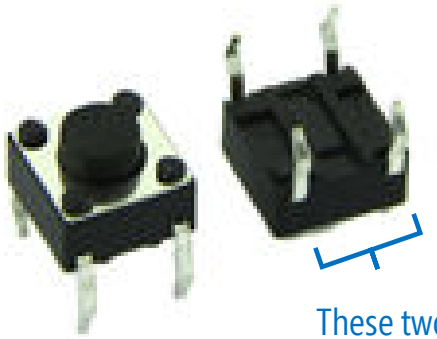
Important Note: Switches cannot be water-washed.

1128 Tactile Switch **B3F**



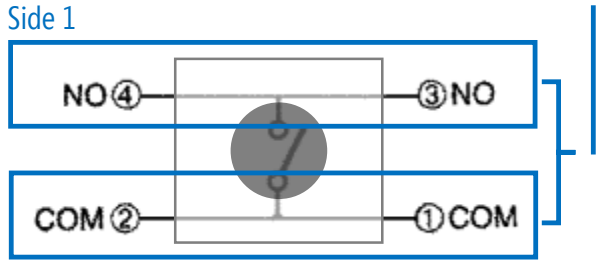


These two sides become connected when button is pressed down



These two sides become connected when button is pressed down

Side 1



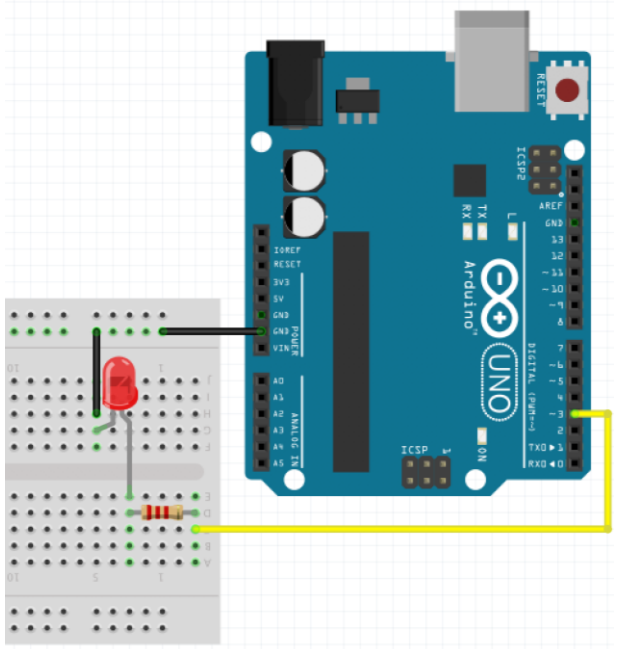
Side 2

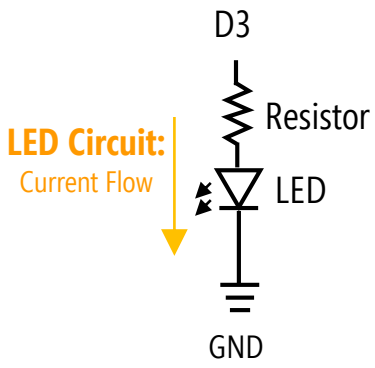

DIGITAL INPUT
USE OUR TRUSTY MULTIMETER: CONTINUITY TESTING
Video by Jon Froehlich

DIGITAL INPUT

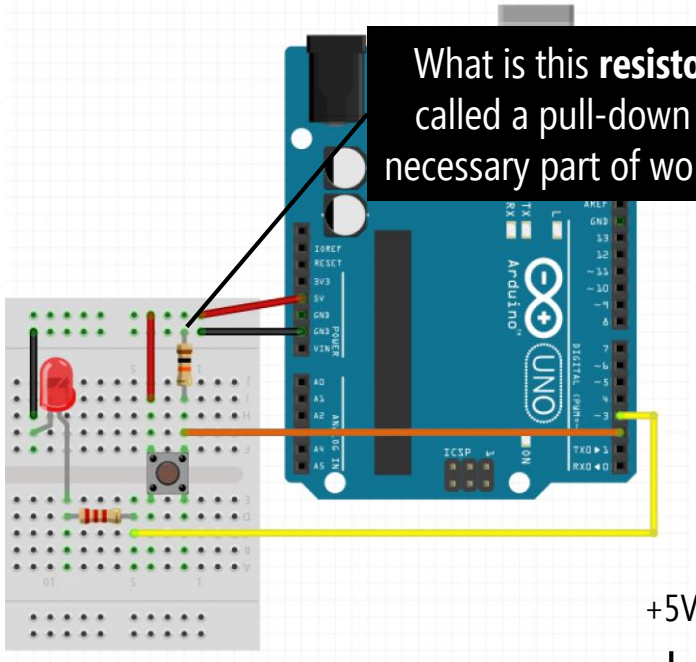
TURN ON/OFF LED WITH BUTTON

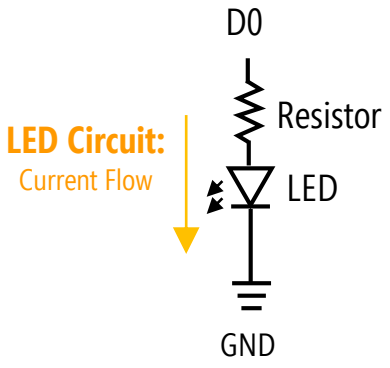
Old Circuit: Flash LED on/off via the pin 3

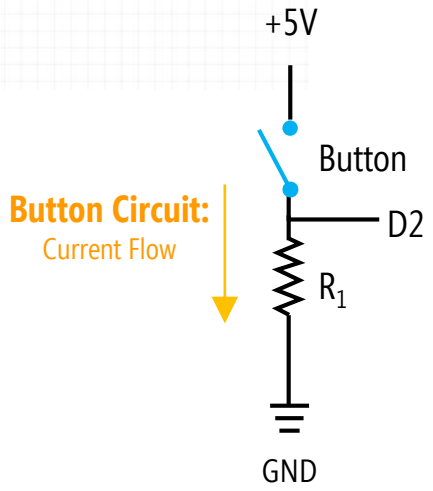




Old Circuit: Turn on/off LED with button

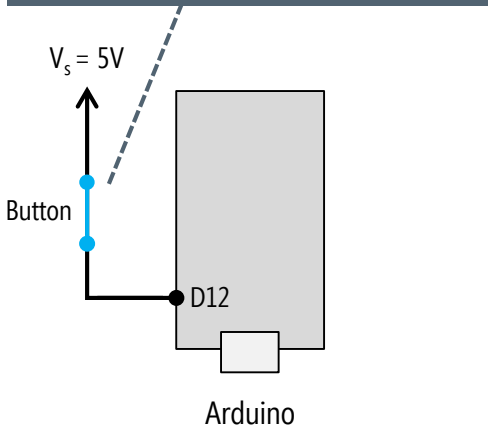






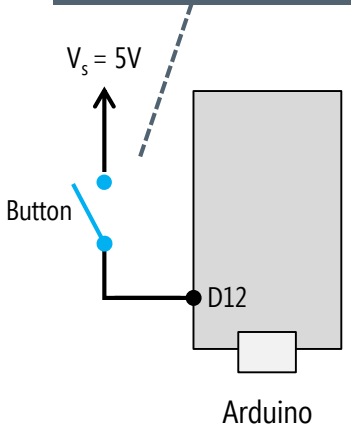
HOOKING UP A BUTTON: PULL-DOWN RESISTORS

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



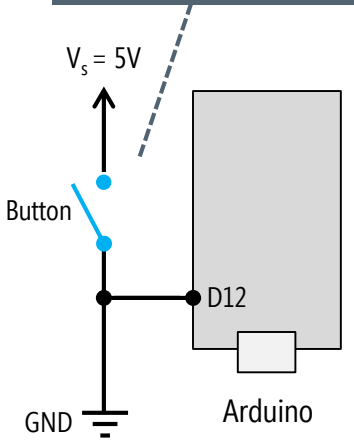
Answer: The MCU would read 5V (or HIGH)

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



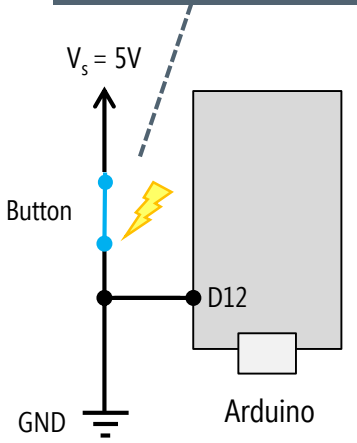
Answer: Uh, oh, the input pin is in an unknown state (commonly called "floating")—this is bad!

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 MCU)!

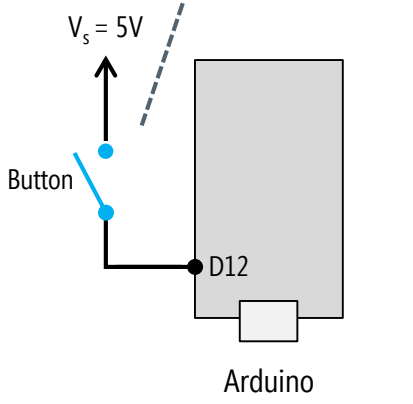


Question: So, what do we do? We add what's called a pull-down resistor.

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

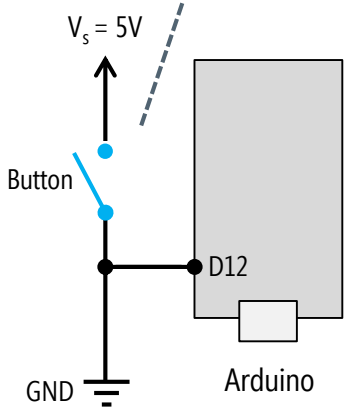
HOOKING UP A BUTTON: PULL-DOWN RESISTORS

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



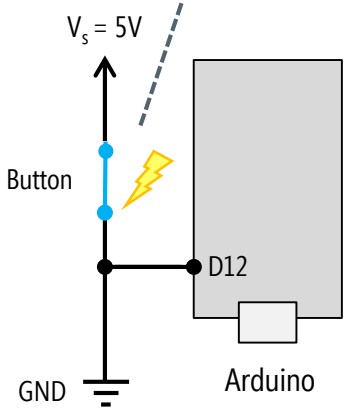
Answer: Uh, oh, the input pin is in an unknown state (commonly called "floating")—this is bad!

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



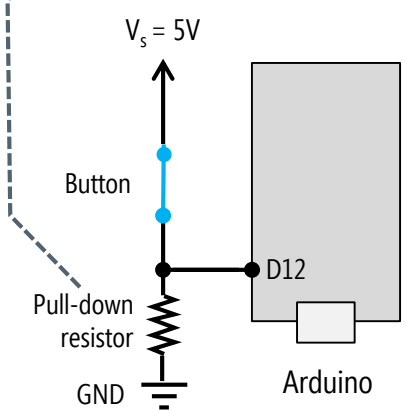
Answer: You're on the right track but this will create a short circuit when the button is closed!

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



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

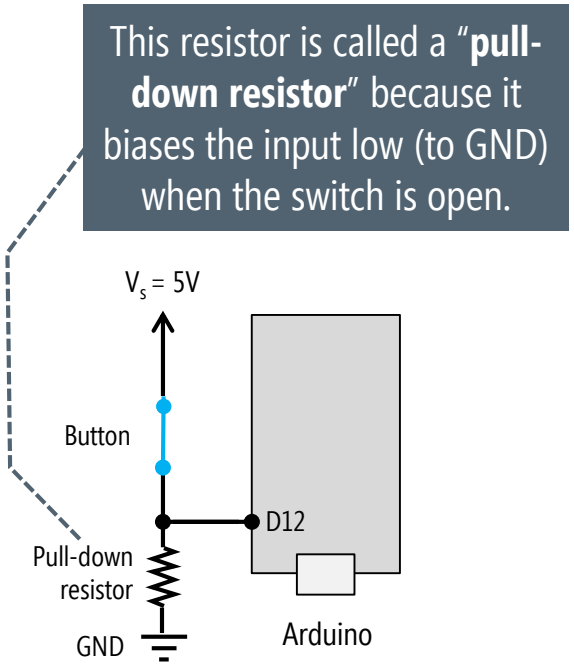
This resistor is called a "**pull-down resistor**" because it biases the input low (to GND) when the switch is open.



Now, when the switch is open, the MCU is pulled to GND. When the switch is closed, the MCU is 5V (HIGH).

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 switch is open, the MCU is pulled to GND. When the switch is closed, the MCU is 5V (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).

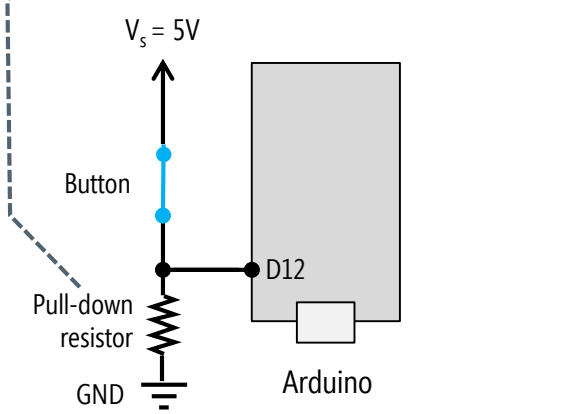
Arduino recommends using a 10KΩ resistor.

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.



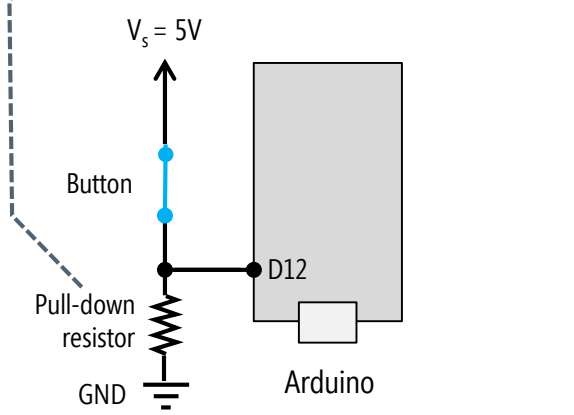
Pull-Down Resistor Configuration
When the switch is open, the MCU is pulled to GND. When the switch is closed, the MCU is 3.3V (HIGH) as it becomes directly connected to V_s.

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

DIGITAL INPUT

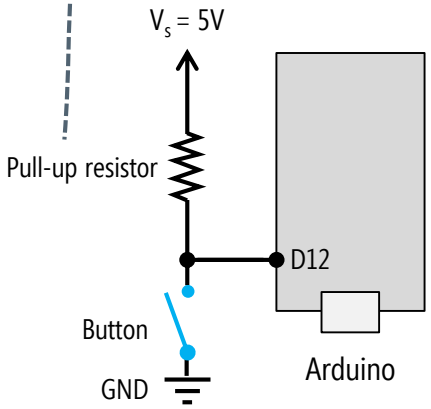
PULL-DOWN VS. PULL-UP RESISTORS

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 "pull-**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.

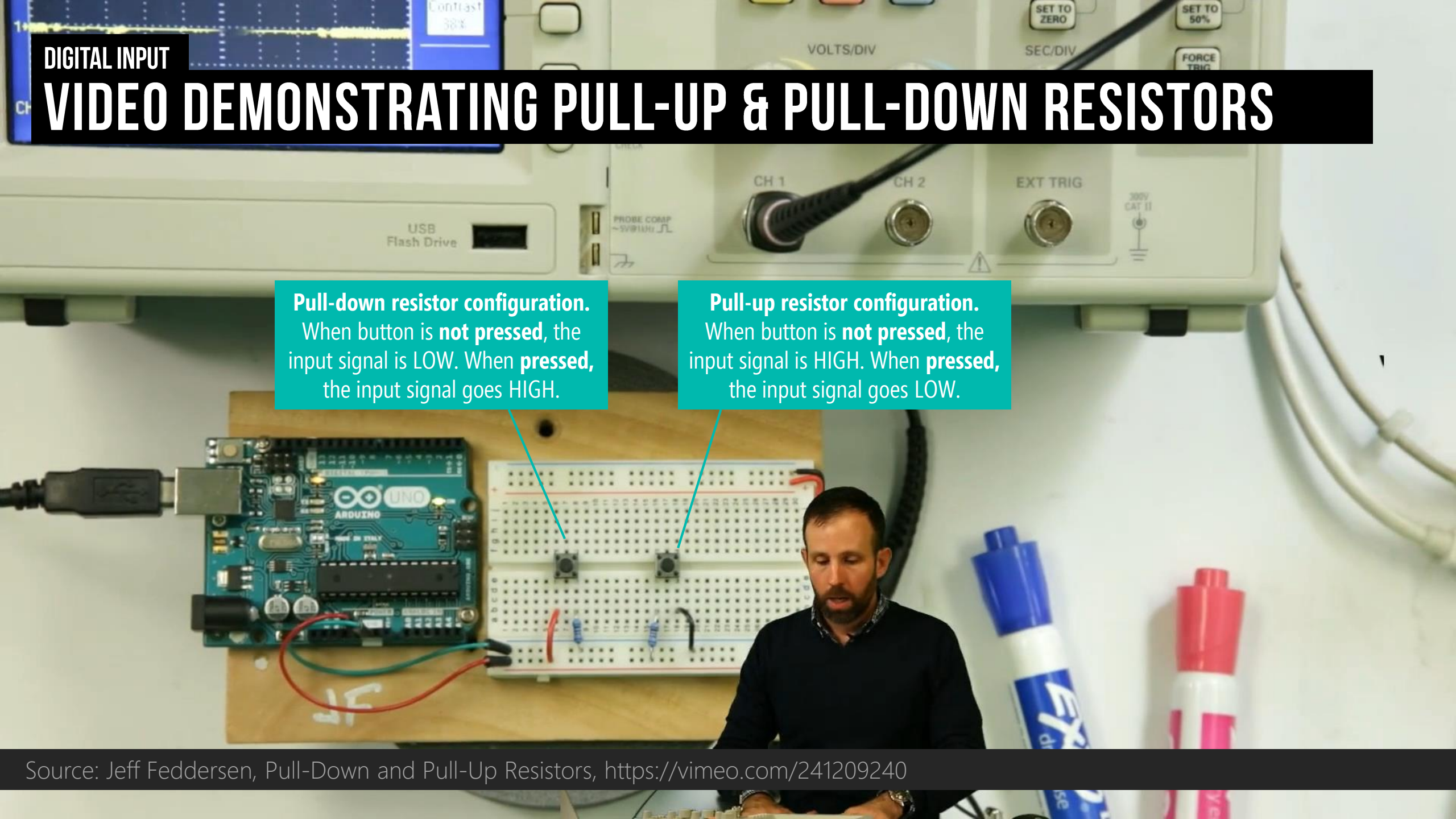
You can **wire up either configuration** (pull-down or pull-up).

I have a preference towards **pull-down resistors** because it makes more sense to me that a switch is 0V (LOW) when open and 5V (HIGH) when closed.

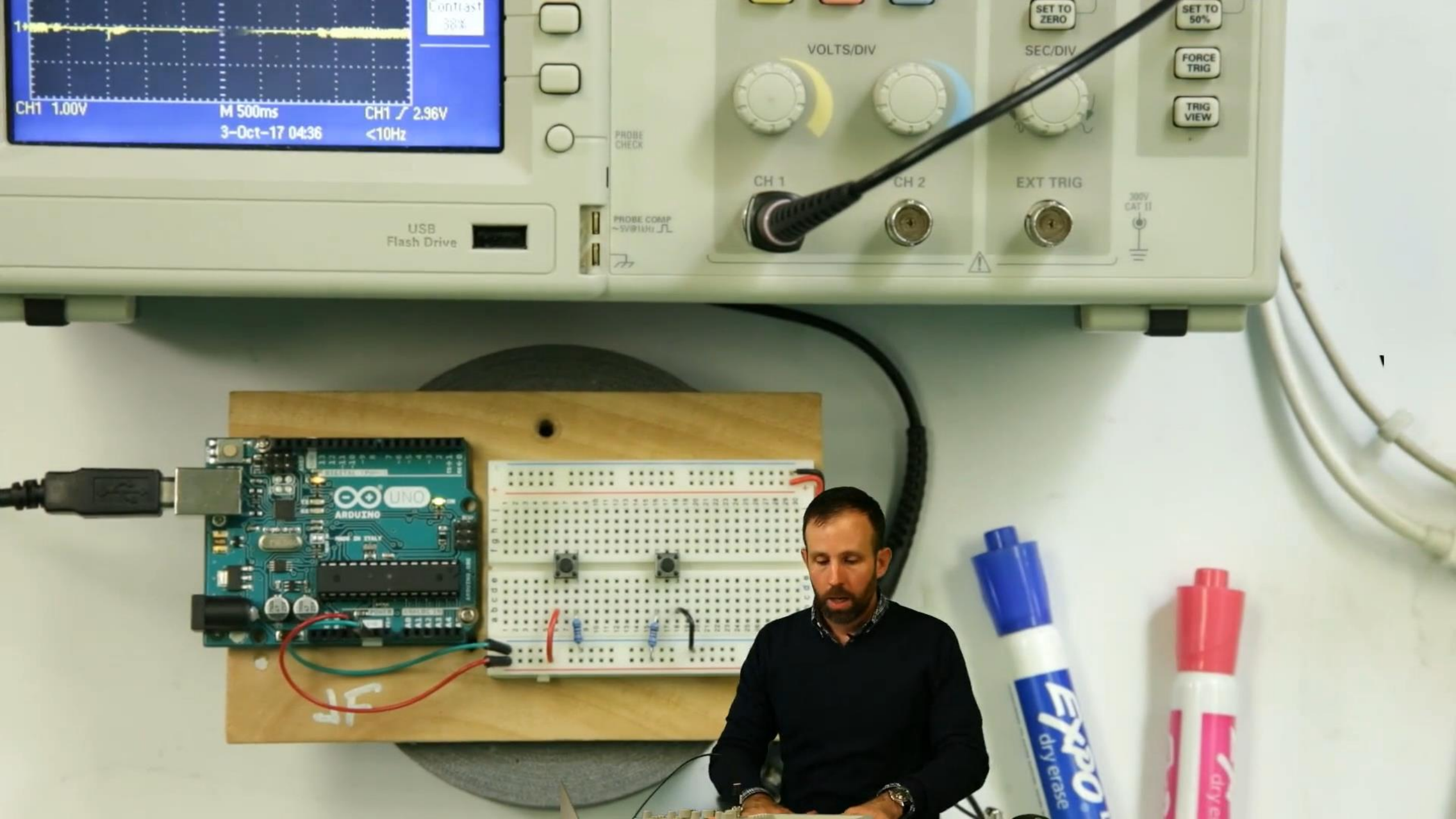
See: https://learn.sparkfun.com/tutorials/switch-basics/all; https://learn.sparkfun.com/tutorials/pull-up-resistors/all

DIGITAL INPUT

VIDEO DEMONSTRATING PULL-UP & PULL-DOWN RESISTORS

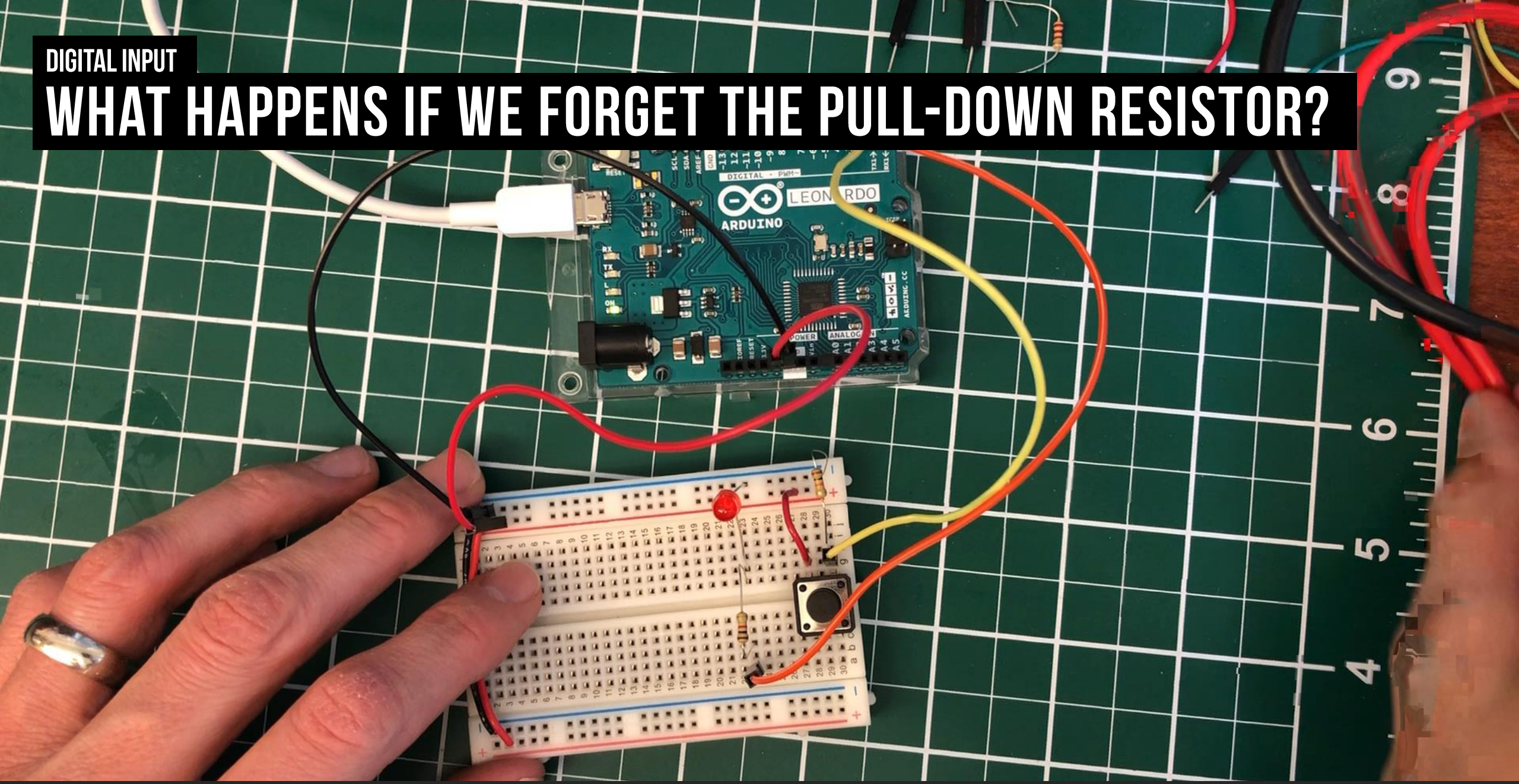


Source: Jeff Feddersen, Pull-Down and Pull-Up Resistors, https://vimeo.com/241209240


CH1 1.00V
M 500ns
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
PROBE CHECK
PROBE COMP
USB
Flash Drive
ARDUINO
UNO
dry erase

DIGITAL INPUT

WHAT HAPPENS IF WE FORGET THE PULL-DOWN RESISTOR?

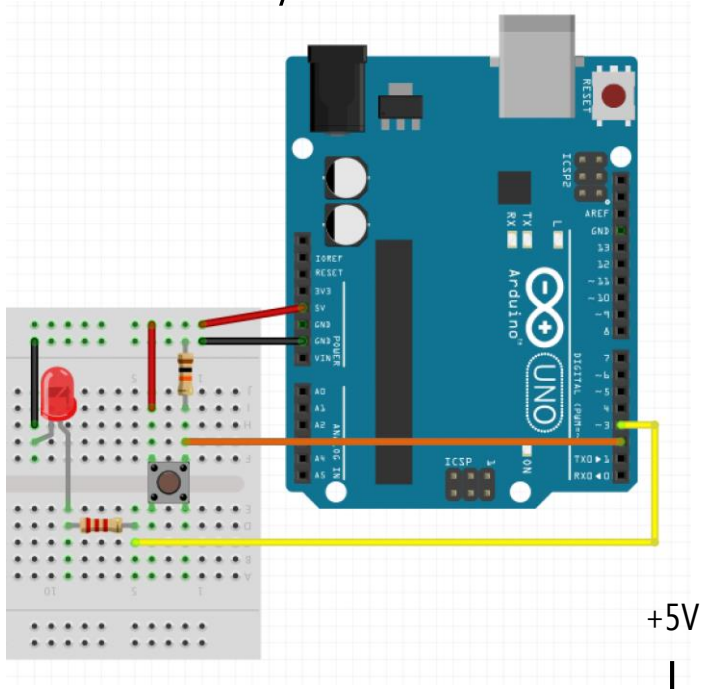


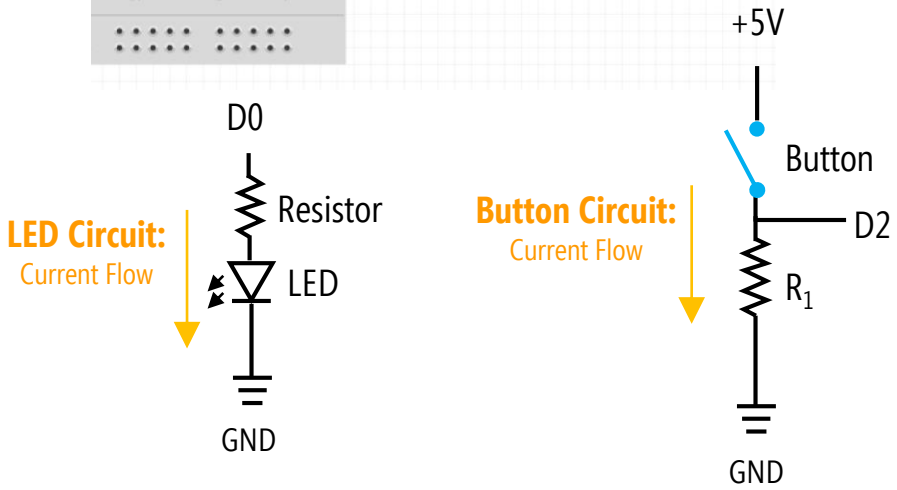
Video by Jon Froehlich

DIGITAL INPUT

TURN ON/OFF LED WITH A BUTTON

Circuit: Turn on/off LED with button





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

`int digitalread(pin)`

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

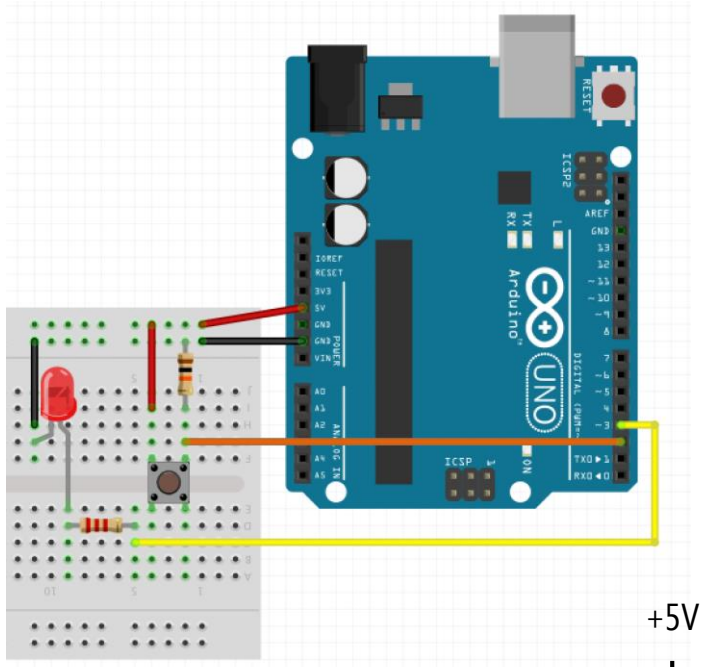
Returns

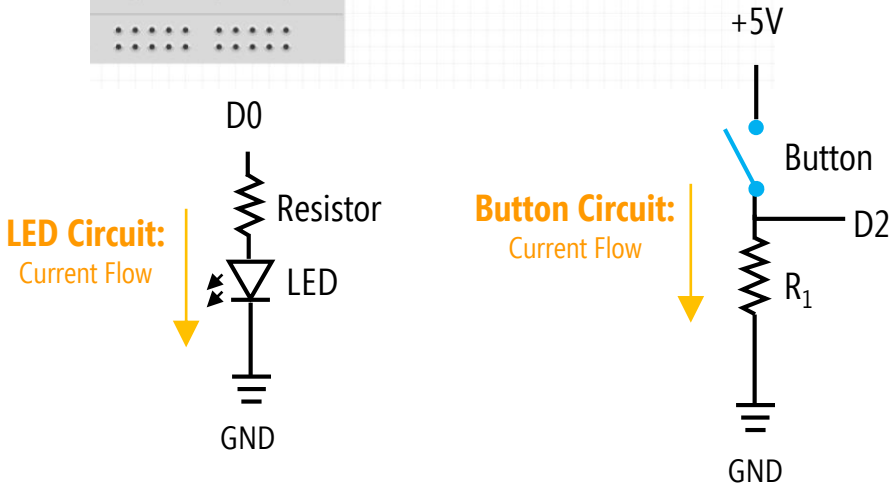
HIGH or LOW

DIGITAL INPUT

TURN ON/OFF LED WITH A BUTTON

New Circuit: Turn on/off LED with button





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

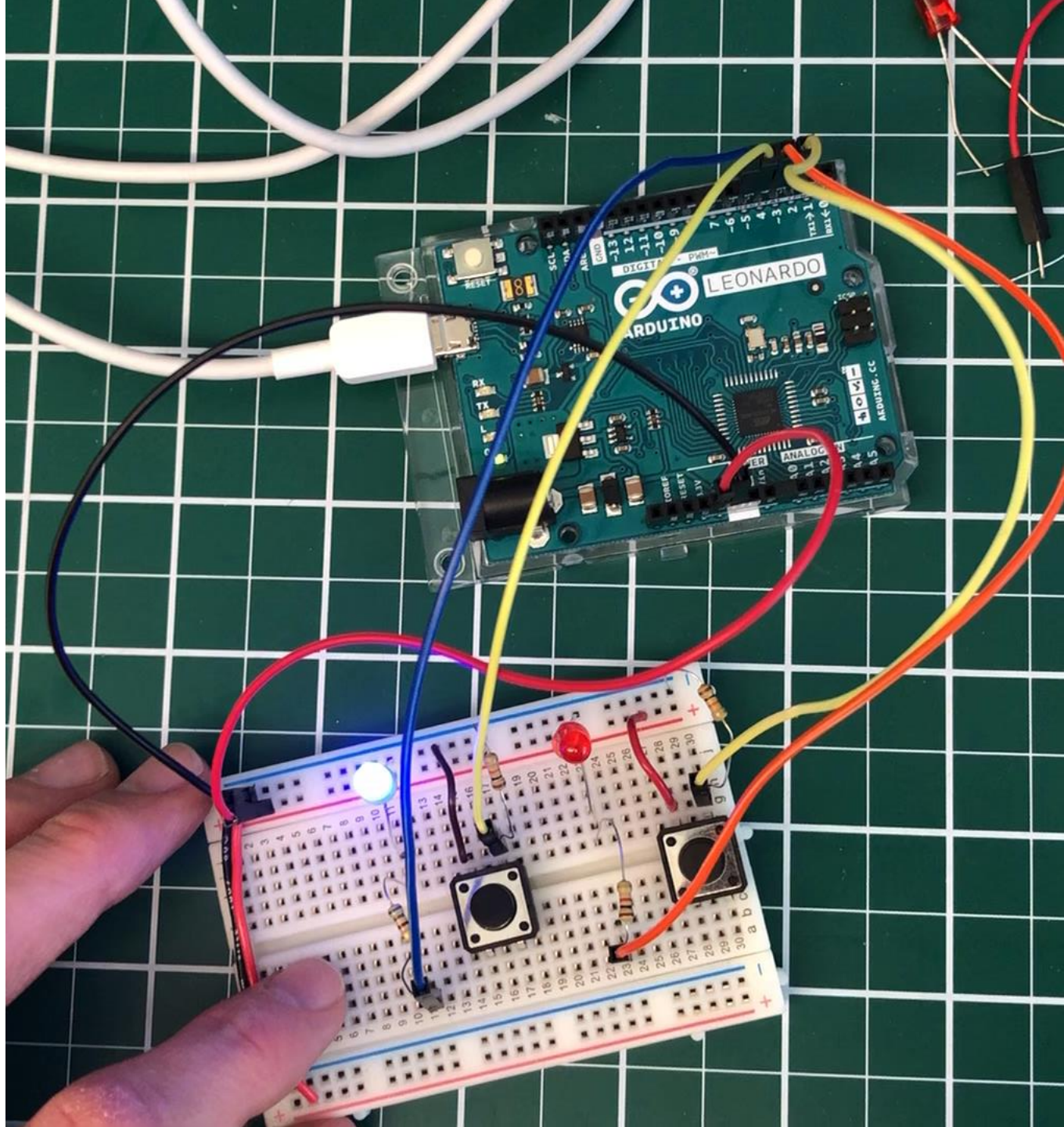
ReadButtonSetLED §

```
/*
 * This example reads in a button input on D2 (with an external pull-down
 * resistor configuration) and turns on/off an LED on D3 accordingly
 *
 * By Jon Froehlich
 * http://makeabilitylab.io
 */

const int BUTTON_INPUT_PIN = 2;
const int LED_OUTPUT_PIN = 3;

void setup() {
  pinMode(BUTTON_INPUT_PIN, INPUT);
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  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);
  Serial.println(buttonVal);

  // Check for new input every 100ms (10 times a sec)
  delay(100);
}
```

DIGITAL INPUT

ACTIVITY: WIRE UP BOTH PULL-DOWN AND PULL-UP CONFIGURATIONS

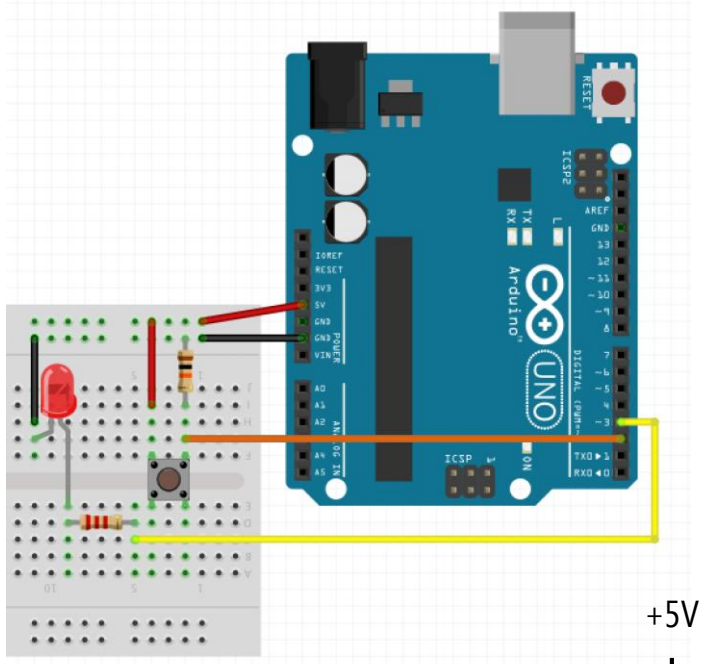
Design circuit + software that uses one button with a pull-down resistor and another with a pull-up resistor

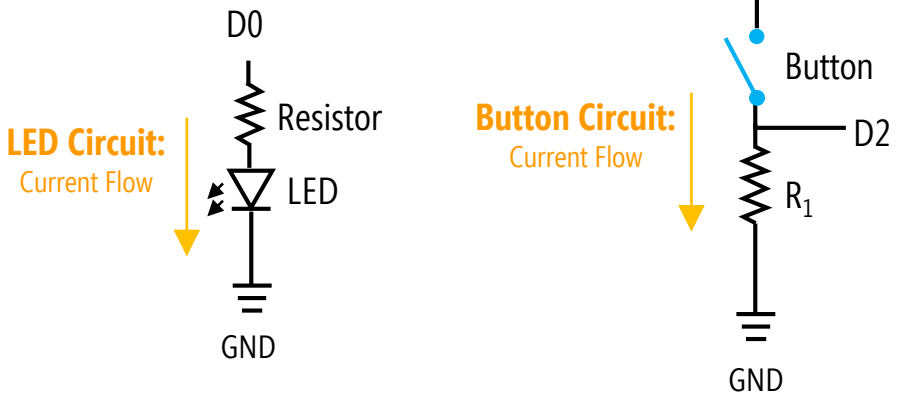


DIGITAL INPUT

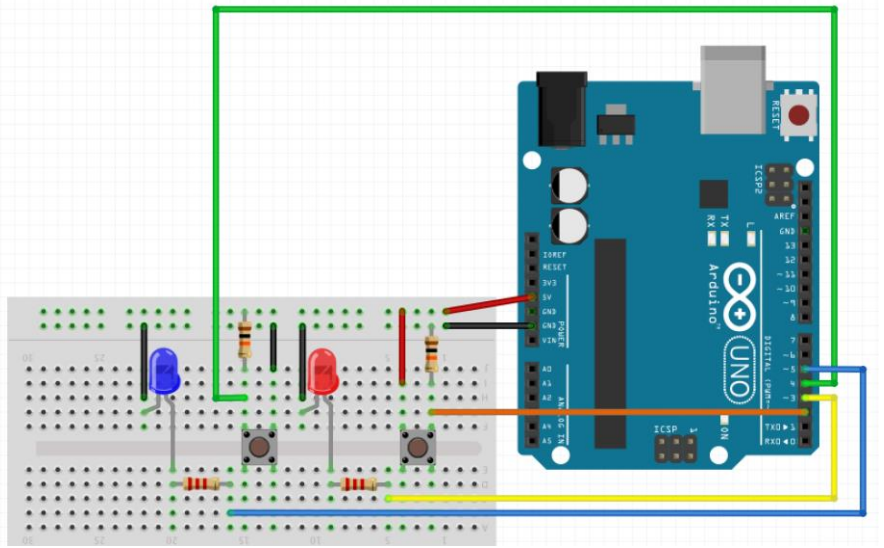
PULL-DOWN AND PULL-UP BUTTON CIRCUITS

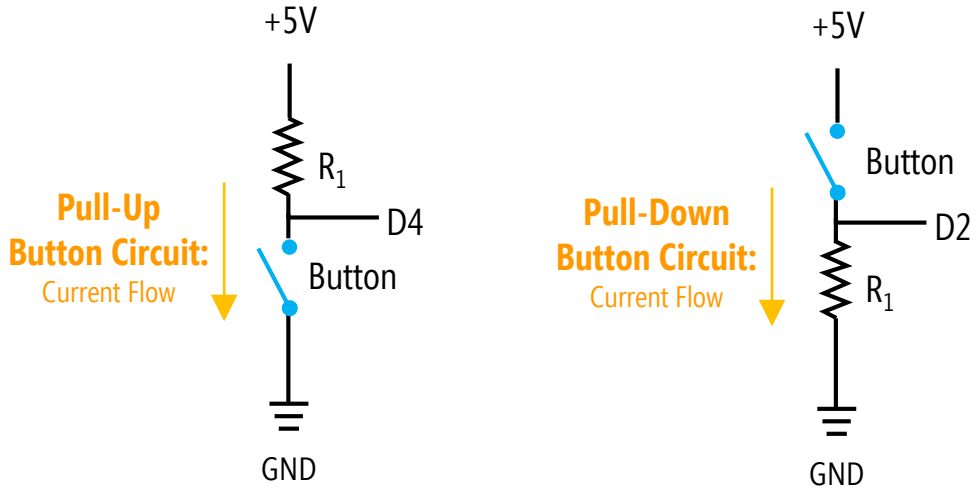
Old Circuit: Turn on/off LED with button





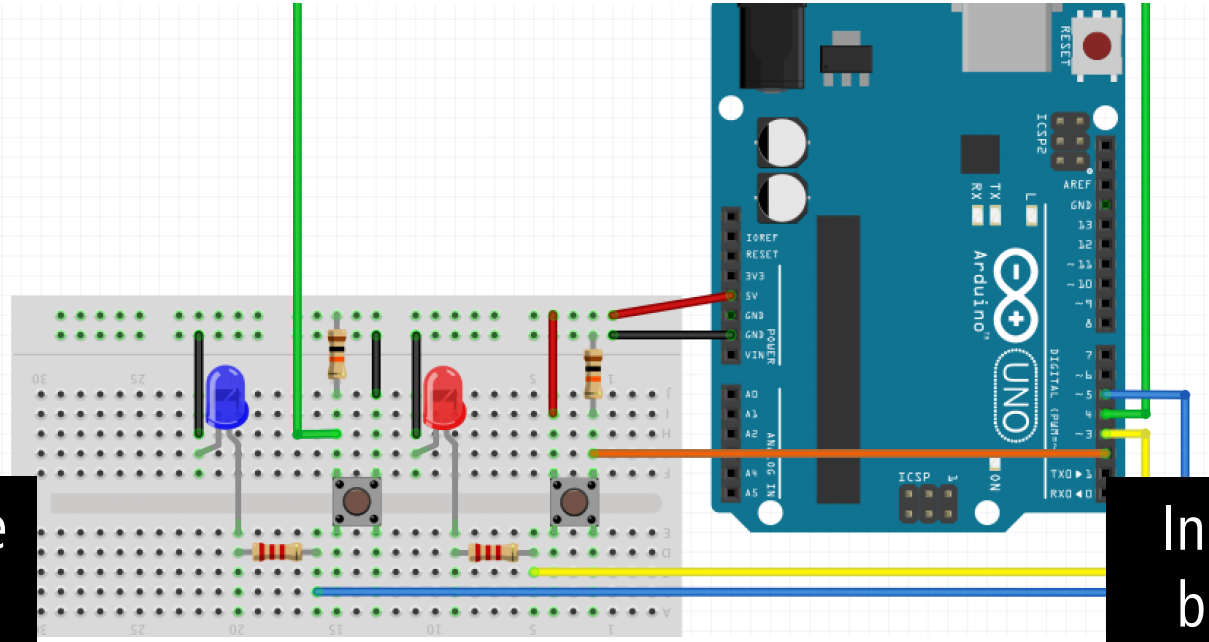
New Circuit: Pull-down and pull-up button configurations



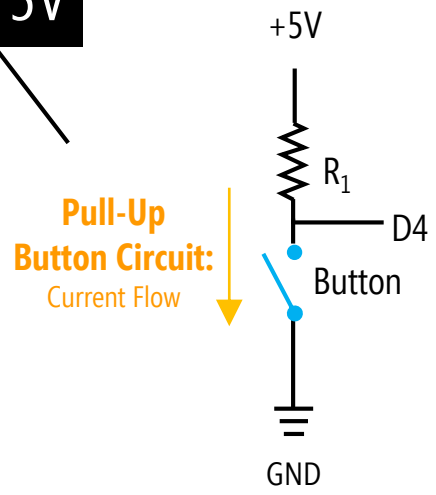


DIGITAL INPUT

PULL-DOWN AND PULL-UP BUTTON CIRCUITS

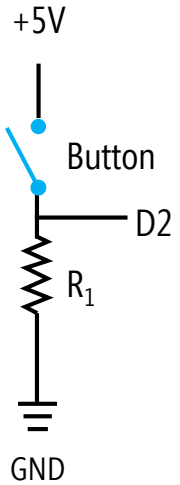


In the **open state** (when the button is **not pressed**), the input to D2 is pulled up to 5V



Pull-Up Button Circuit: Current Flow

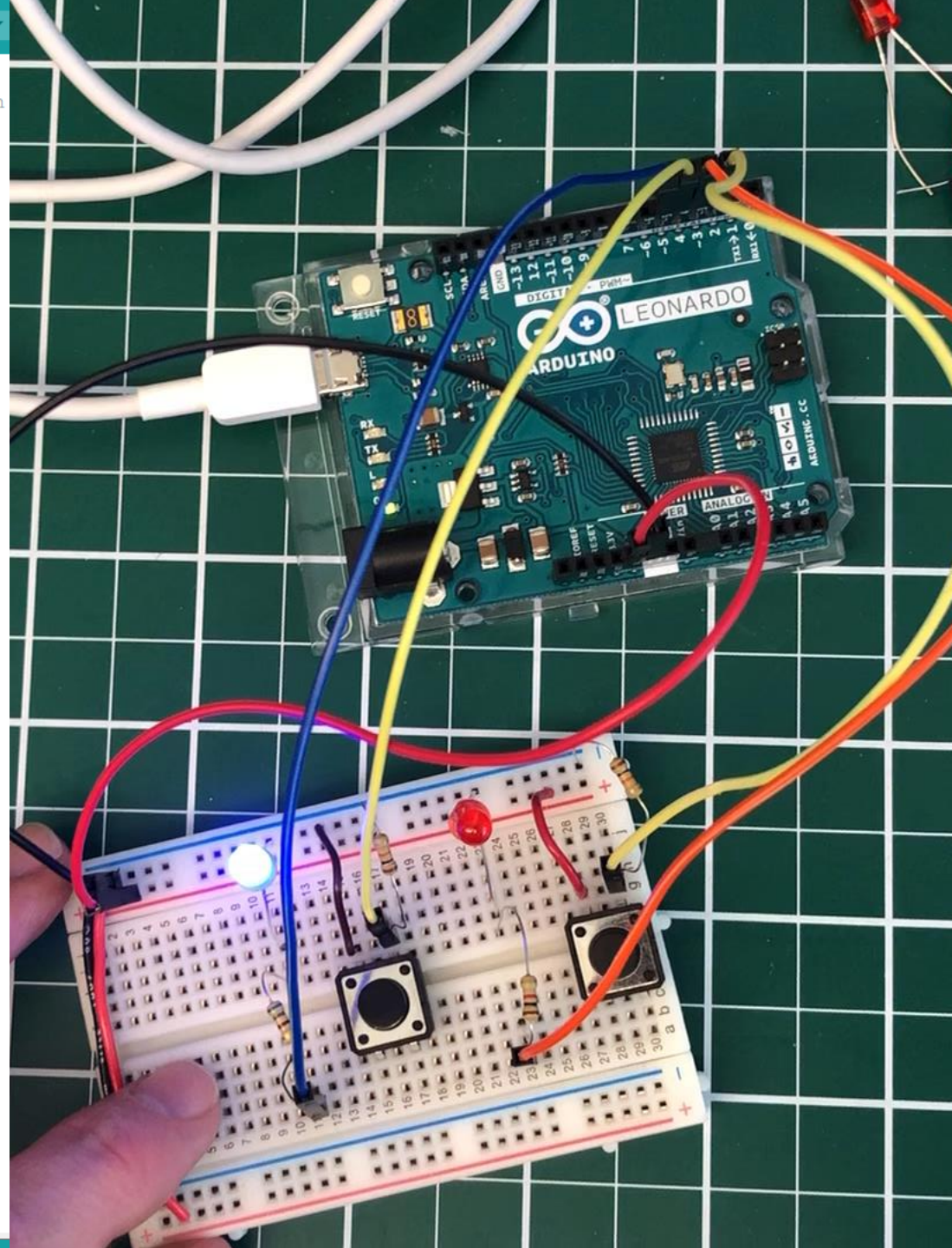
Pull-Down Button Circuit: Current Flow



In the **open state** (when the button is **not pressed**), the input to D2 is pulled down GND

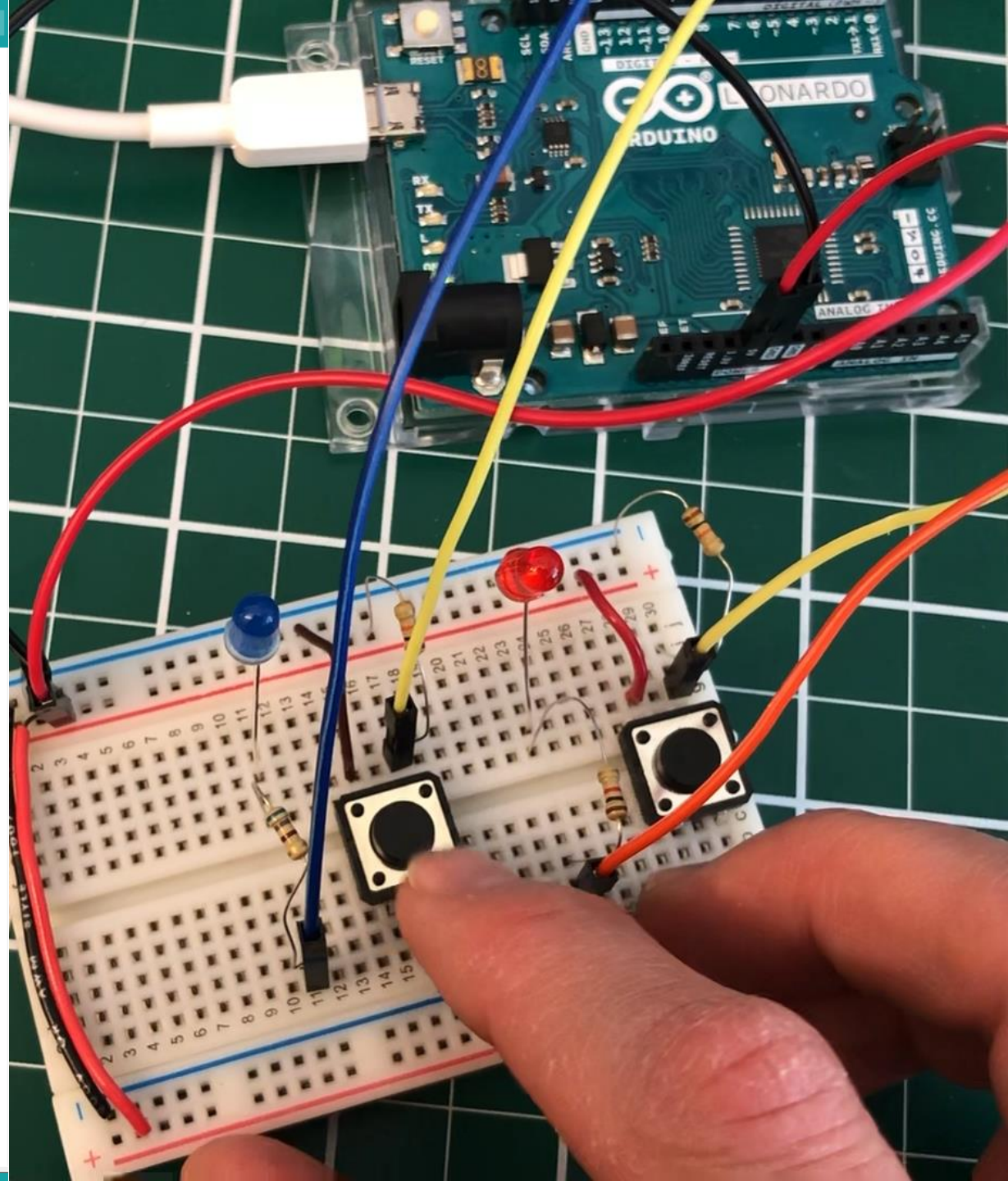
ReadButtonsSetLEDs §

```
/*
 * This example demonstrates the difference between a pull-down and pull-up resistor configuration
 * with digital input.
 *
 * By Jon Froehlich
 * http://makeabilitylab.io
 */

const int BUTTON_INPUT_PIN_PD = 2; // button with a pull-down (PD) resistor
const int LED_OUTPUT_PIN_PD = 3;   // the LED for the PD button
const int BUTTON_INPUT_PIN_PU = 4; // btton with a pull-up (PU) resistor
const int LED_OUTPUT_PIN_PU = 5;   // the LED for the PU button

void setup() {
  pinMode(BUTTON_INPUT_PIN_PD, INPUT);
  pinMode(LED_OUTPUT_PIN_PD, OUTPUT);

  pinMode(BUTTON_INPUT_PIN_PU, INPUT);
  pinMode(LED_OUTPUT_PIN_PU, OUTPUT);
}

void loop() {

  // Read the button configured with a pull-down resistor. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonValPD = digitalRead(BUTTON_INPUT_PIN_PD);

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN_PD, buttonValPD);

  // Read the button configured with a pull-up resistor. It will be LOW when pressed and
  // HIGH when not pressed
  int buttonValPU = digitalRead(BUTTON_INPUT_PIN_PU);

  // Write out HIGH or LOW.
  digitalWrite(LED_OUTPUT_PIN_PU, buttonValPU);

  // Check for new input every 100ms (10 times a sec)
  delay(100);
}
```



ReadButtonsSetLEDs §

```
/*
 * This example demonstrates the difference between a pull-down and pull-up resistor configuration
 * with digital input. There is a button hooked up to D2 with a pull-down resistor and a button
 * hooked up to D4 with a pull-up resistor. The LEDs on D3 and D5 illuminate when their respective
 * buttons are pressed.
 *
 * By Jon Froehlich
 * http://makeabilitylab.io
 */

const int BUTTON_INPUT_PIN_PD = 2; // button with a pull-down (PD) resistor
const int LED_OUTPUT_PIN_PD = 3;   // the LED for the PD button
const int BUTTON_INPUT_PIN_PU = 4; // btton with a pull-up (PU) resistor
const int LED_OUTPUT_PIN_PU = 5;   // the LED for the PU button

void setup() {
  pinMode(BUTTON_INPUT_PIN_PD, INPUT);
  pinMode(LED_OUTPUT_PIN_PD, OUTPUT);

  pinMode(BUTTON_INPUT_PIN_PU, INPUT);
  pinMode(LED_OUTPUT_PIN_PU, OUTPUT);
}

void loop() {

  // Read the button configured with a pull-down resistor. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonValPD = digitalRead(BUTTON_INPUT_PIN_PD);

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN_PD, buttonValPD);

  // Read the button configured with a pull-up resistor. It will be LOW when pressed and
  // HIGH when not pressed
  int buttonValPU = digitalRead(BUTTON_INPUT_PIN_PU);

  // Write out HIGH or LOW. Note that we invert the button value because it's hooked up
  // to a pull-up resistor (that is, when the button is pressed, the signal goes LOW but
  // we want to illuminate our LED when the button is pressed).
  digitalWrite(LED_OUTPUT_PIN_PU, !buttonValPU);

  // Check for new input every 100ms (10 times a sec)
  delay(100);
}
```



TUTORIALS > Foundation > Digital Pins

Digital Pins

The pins on the Arduino can be configured as either inputs or outputs. This document explains the functioning of the pins in those modes. While the title of this document refers to digital pins, it is important to note that vast majority of Arduino (Atmega) analog pins, may be configured, and used, in exactly the same manner as digital pins.

Properties of Pins Configured as INPUT

Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode() when you're using them as inputs. Pins configured this way are said to be in a **high-impedance state**. Input pins make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 megohm in front of the pin. This means that it takes very little current to move the input pin from one state to another, and can make the pins useful for such tasks as implementing a capacitive touch sensor, reading an LED as a photodiode, or reading an analog sensor with a scheme such as RCTime.

This also means however, that pins configured as pinMode(pin, INPUT) with nothing connected to them, or with wires connected to them that are not connected to other circuits, will report seemingly random changes in pin state, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin.

Pullup Resistors with pins configured as INPUT

Often it is useful to steer an input pin to a known state if no input is present. This can be done by adding a pullup resistor (to +5V), or a pulldown resistor (resistor to ground) on the input. A 10K resistor is a good value for a pullup or pulldown resistor.

Properties of Pins Configured as INPUT_PULLUP

There are 20K pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed by setting the pinMode() as INPUT_PULLUP. This effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on.

The value of this pullup depends on the microcontroller used. On most AVR-based boards, the value is guaranteed to be between 20kΩ and 50kΩ. On the Arduino Due, it is between 50kΩ and 150kΩ. For the exact value, consult the datasheet of the microcontroller on your board.

When connecting a sensor to a pin configured with INPUT_PULLUP, the other end should be connected to ground. In the case of a simple switch, this causes the pin to read HIGH when the switch is open, and LOW when the switch is pressed.

The pullup resistors provide enough current to dimly light an LED connected to a pin that has been configured as an input. If LEDs in a project seem to be working, but very dimly, this is likely what is going on.

The pullup resistors are controlled by the same registers (internal chip memory locations) that control whether a pin is HIGH or LOW. Consequently, a pin that is configured to have pullup resistors turned on when the pin is an INPUT, will have the pin configured as HIGH if the pin is then switched to an OUTPUT with pinMode(). This works in the other direction as well, and an output pin that is left in a HIGH state will have the pullup resistors set if switched to an input with pinMode().

Prior to Arduino 1.0.1, it was possible to configure the internal pull-ups in the following manner:

```
pinMode(pin, INPUT);           // set pin to input
digitalWrite(pin, HIGH);       // turn on pullup resistors
```

NOTE: Digital pin 13 is harder to use as a digital input than the other digital pins because it has an LED and resistor attached to it that's soldered to the board on most boards. If you enable its internal 20k pull-up resistor, it will hang at around 1.7V instead of the expected 5V because the onboard LED and series resistor pull the voltage level down, meaning it always returns LOW. If you must use pin 13 as a digital input, set its pinMode() to INPUT and use an external pull down resistor.

Properties of Pins Configured as OUTPUT

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

Short circuits on Arduino pins, or attempting to run high current devices from them, can damage or destroy the output transistors in the pin, or damage the entire Atmega chip. Often this will result in a "dead" pin in the microcontroller but the remaining chip will still function adequately. For this reason it is a good idea to connect OUTPUT pins to other devices with 470Ω or 1k resistors, unless maximum current draw from the pins is required for a particular application.

Properties of Pins Configured as INPUT_PULLUP

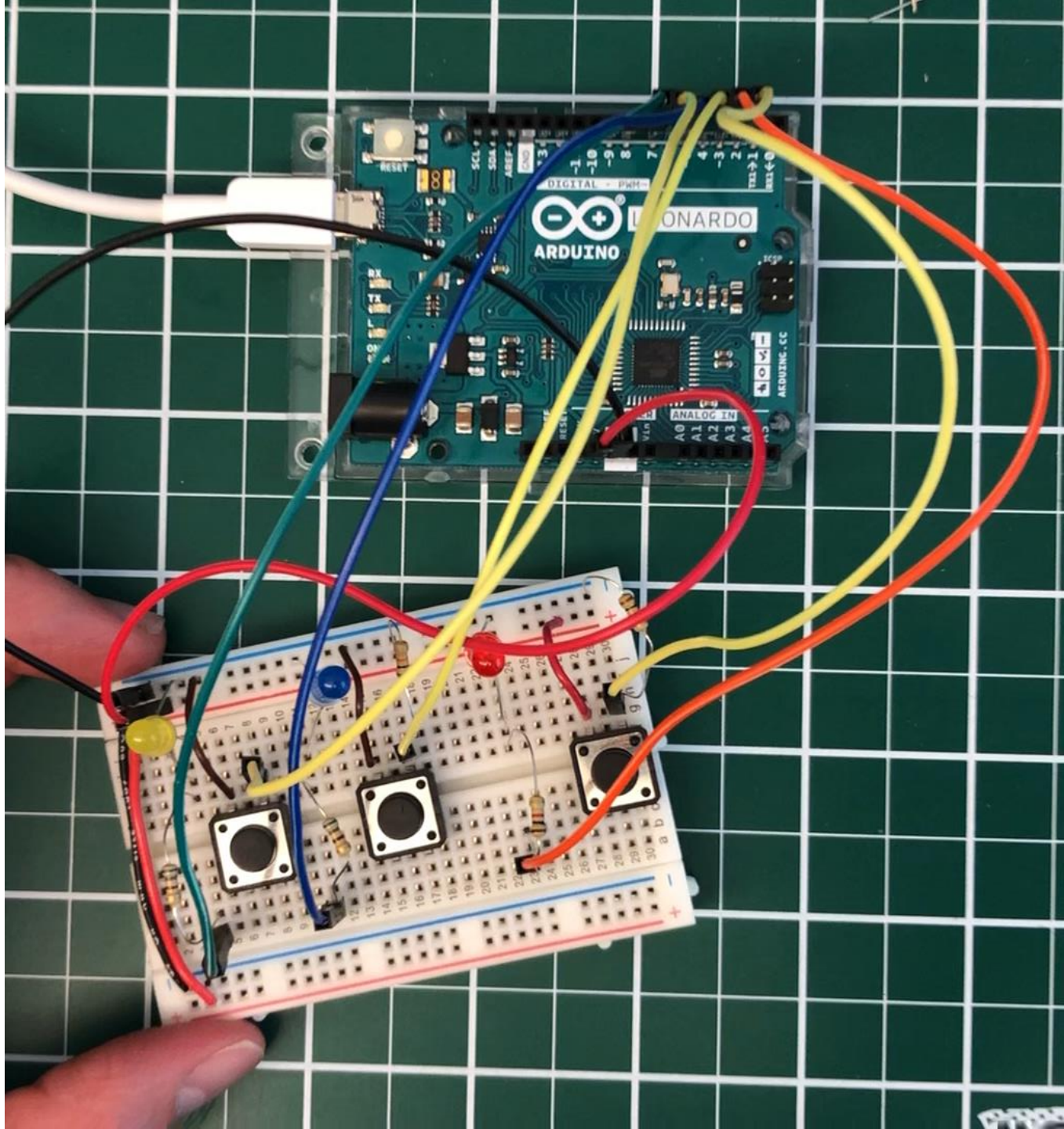
There are 20K pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed by setting the pinMode() as INPUT_PULLUP. This effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on.

The value of this pullup depends on the microcontroller used. On most AVR-based boards, the value is guaranteed to be between 20kΩ and 50kΩ. On the Arduino Due, it is between 50kΩ and 150kΩ. For the exact value, consult the datasheet of the microcontroller on your board.

DIGITAL INPUT

ACTIVITY: WIRE UP ALL THREE CONFIGURATIONS

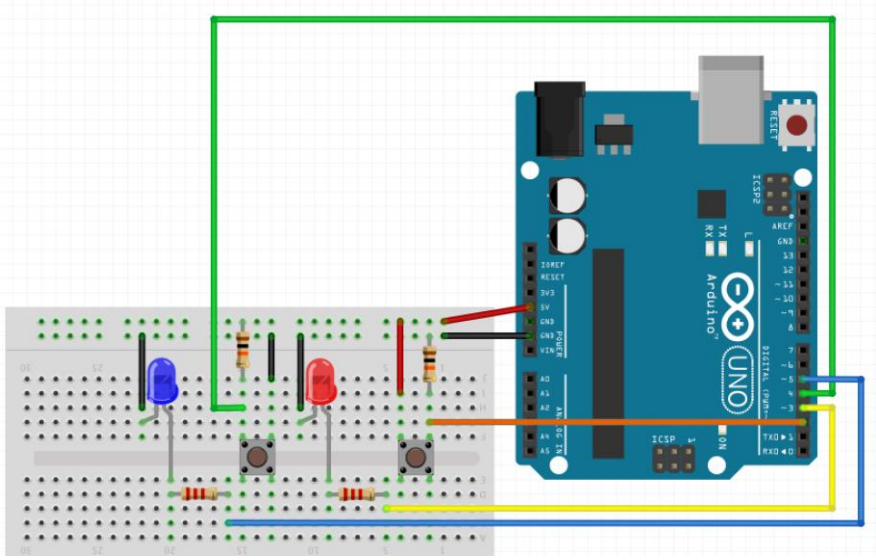
Design circuit + software that uses one button with a pull-down resistor, one with a pull-up resistor, and another with an internal pull-up resistor



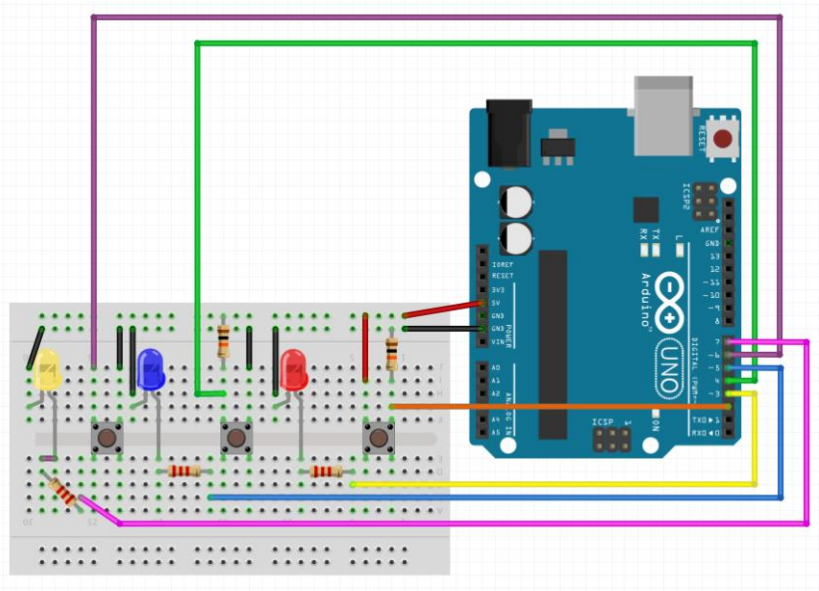
DIGITAL INPUT

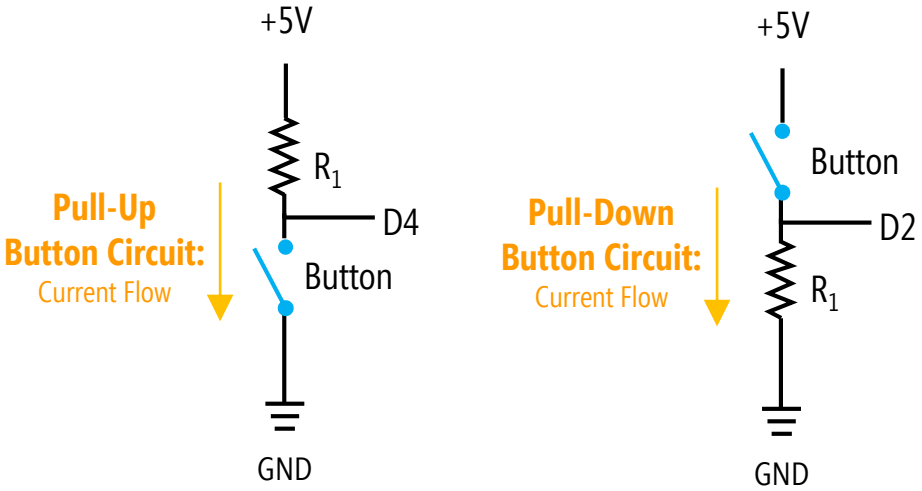
PULL-DOWN, PULL-UP, & INTERNAL PULL-UP

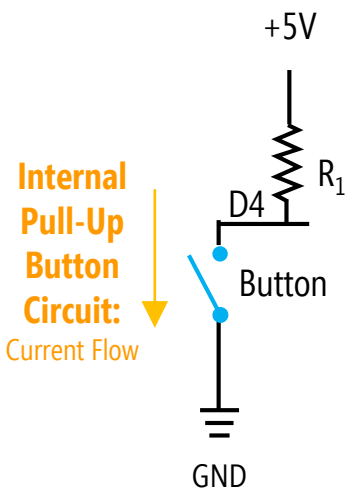
Old Circuit: Pull-down and pull-up button configurations

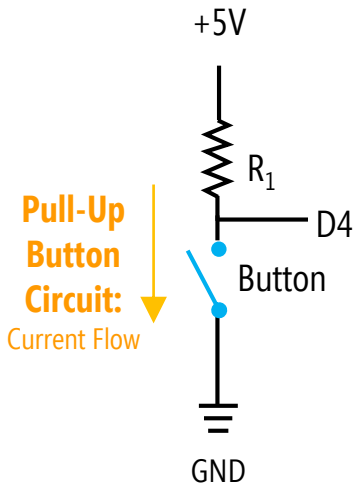


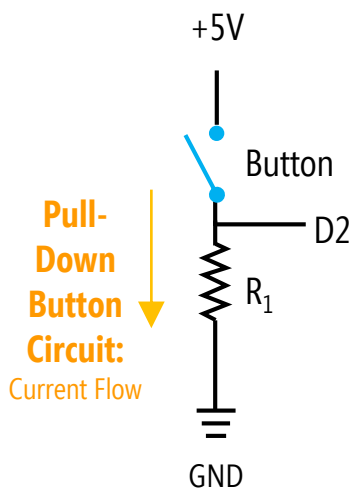
New Circuit: Pull-down, pull-up, & internal pull-up





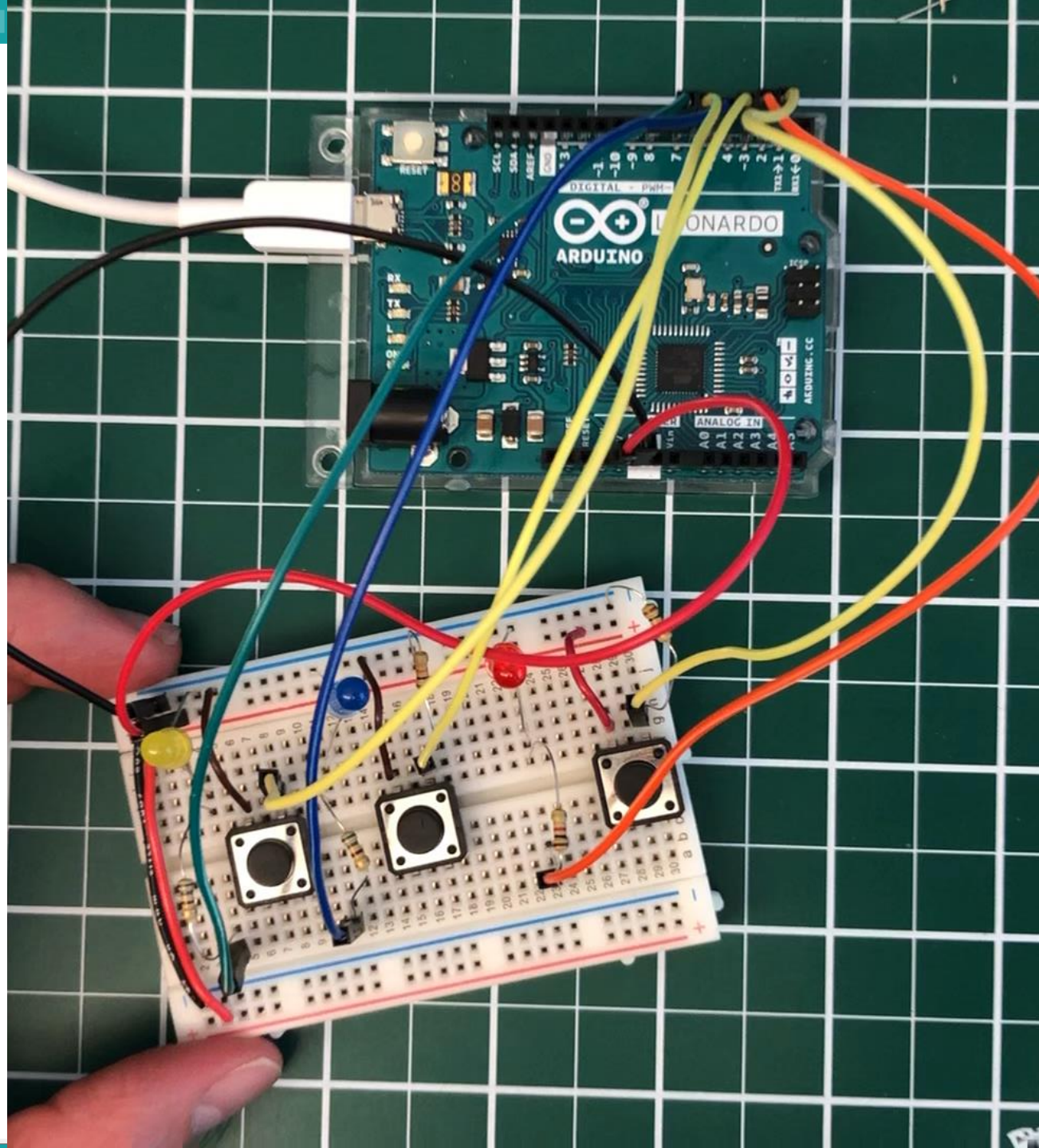






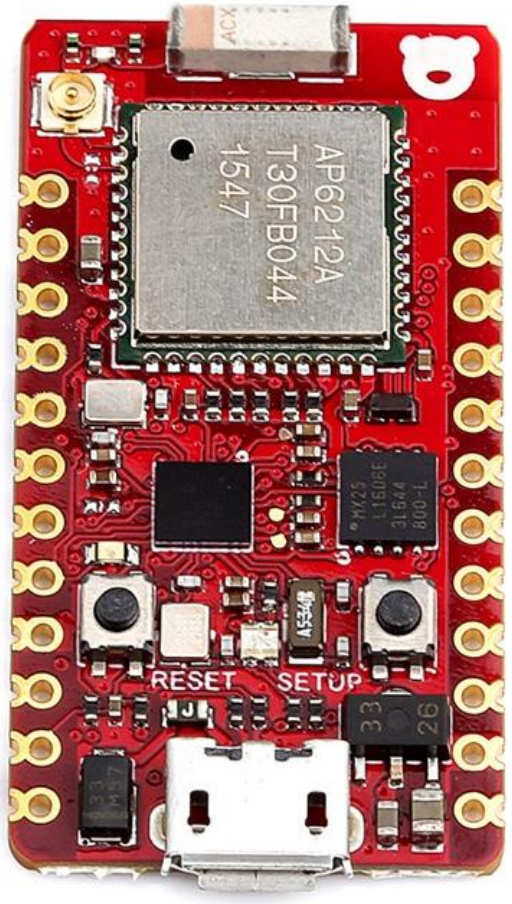
ReadButtonsSetLEDs2

```
const int BUTTON_INPUT_PIN_PD = 2; // button with a pull-down (PD) resistor
const int LED_OUTPUT_PIN_PD = 3;   // the LED for the PD button
const int BUTTON_INPUT_PIN_PU = 4; // btton with a pull-up (PU) resistor
const int LED_OUTPUT_PIN_PU = 5;   // the LED for the PU button
const int BUTTON_INPUT_PIN_IPU = 6; // btton with an internal pull-up (IPU) resistor
const int LED_OUTPUT_PIN_IPU = 7;   // the LED for the IPU button

void setup() {
  pinMode(BUTTON_INPUT_PIN_PD, INPUT);
  pinMode(LED_OUTPUT_PIN_PD, OUTPUT);

  pinMode(BUTTON_INPUT_PIN_PU, INPUT);
  pinMode(LED_OUTPUT_PIN_PU, OUTPUT);

  pinMode(BUTTON_INPUT_PIN_IPU, INPUT_PULLUP);
  pinMode(LED_OUTPUT_PIN_IPU, OUTPUT);
}

void loop() {

  // Read the button configured with a pull-down resistor. It will be HIGH when pressed and
  // LOW when not pressed
  int buttonValPD = digitalRead(BUTTON_INPUT_PIN_PD);

  // Write out HIGH or LOW
  digitalWrite(LED_OUTPUT_PIN_PD, buttonValPD);

  // Read the button configured with a pull-up resistor. It will be LOW when pressed and
  // HIGH when not pressed
  int buttonValPU = digitalRead(BUTTON_INPUT_PIN_PU);

  // Write out HIGH or LOW. Note that we invert the button value because it's hooked up
  // to a pull-up resistor (that is, when the button is pressed, the signal goes LOW but
  // we want to illuminate our LED when the button is pressed).
  digitalWrite(LED_OUTPUT_PIN_PU, !buttonValPU);

  // Read the button configured with an internal pull-up resistor. It will be LOW when pressed and
  // HIGH when not pressed
  int buttonValIPU = digitalRead(BUTTON_INPUT_PIN_IPU);

  // Write out HIGH or LOW. Note that we invert the button value because it's hooked up
  // to a pull-up resistor (that is, when the button is pressed, the signal goes LOW but
  // we want to illuminate our LED when the button is pressed).
  digitalWrite(LED_OUTPUT_PIN_IPU, !buttonValIPU);

  // Check for new input every 100ms (10 times a sec)
  delay(100);
}
```



SOME MCUS HAVE BOTH PULL-DOWN AND PULL-UP RESISTORS

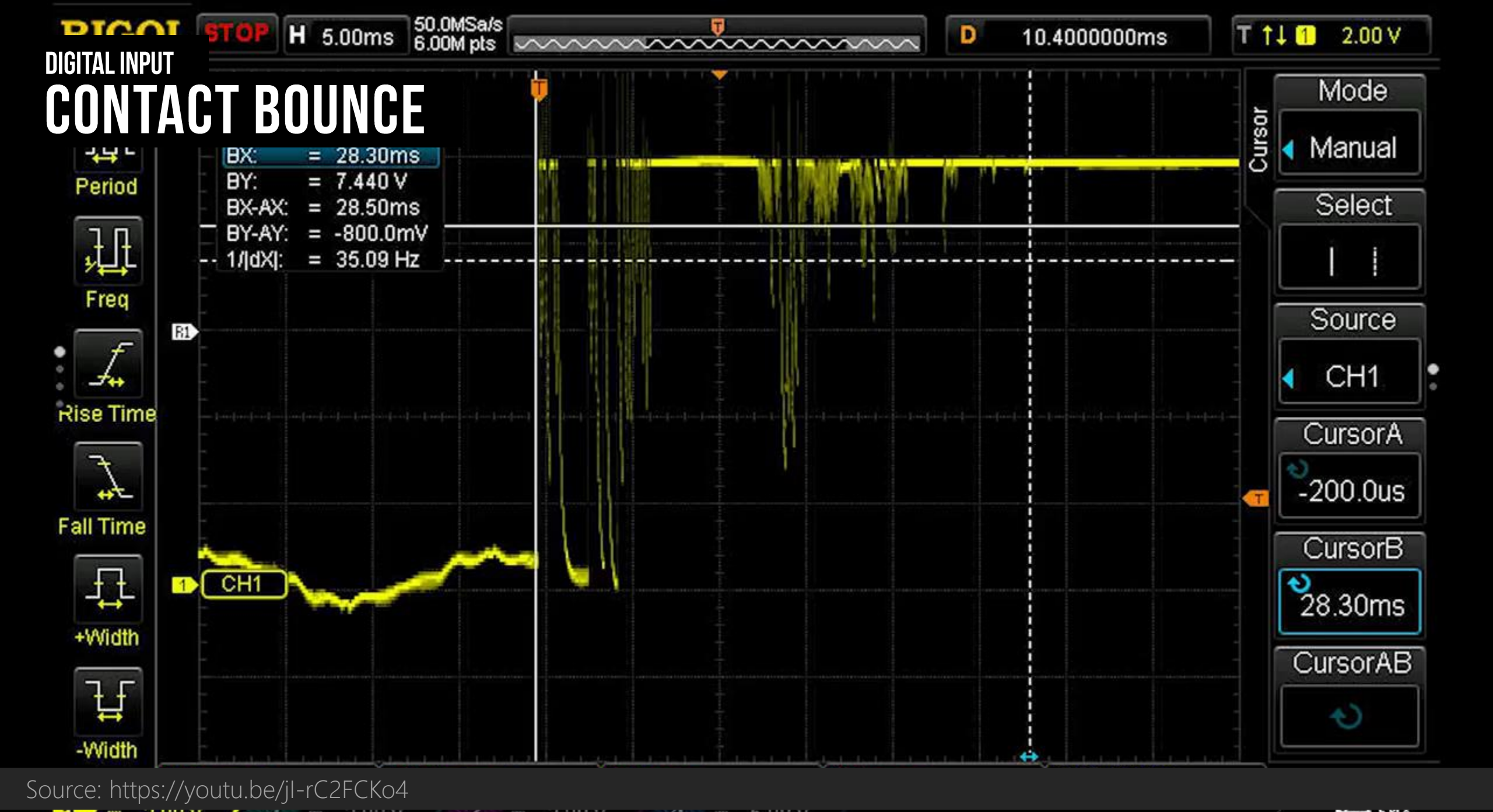
The RedBear Duo board that I taught with Spring 2018 had both software configurable pull-down and pull-up resistors for the input pins



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

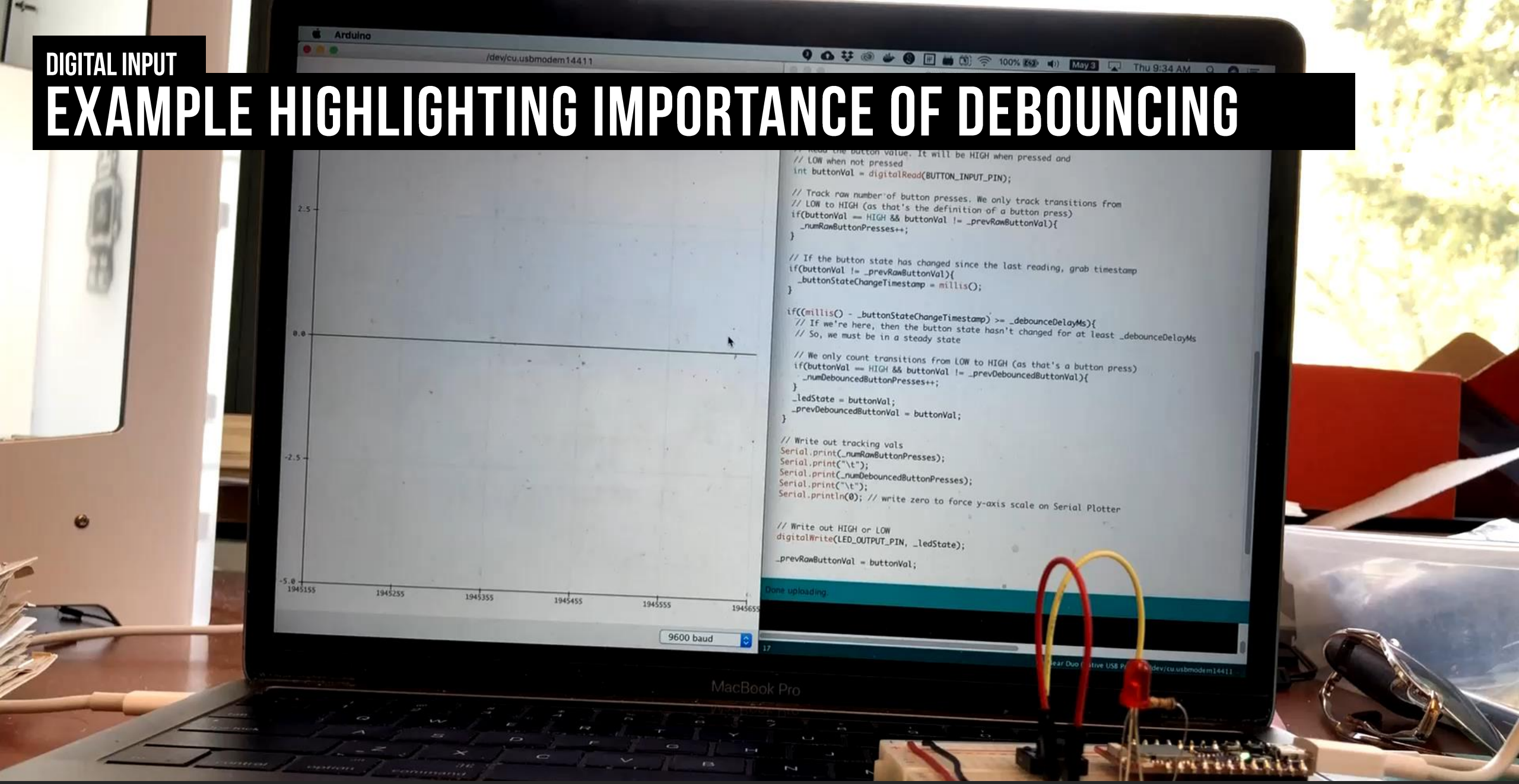


Source: https://youtu.be/jI-rC2FCKo4

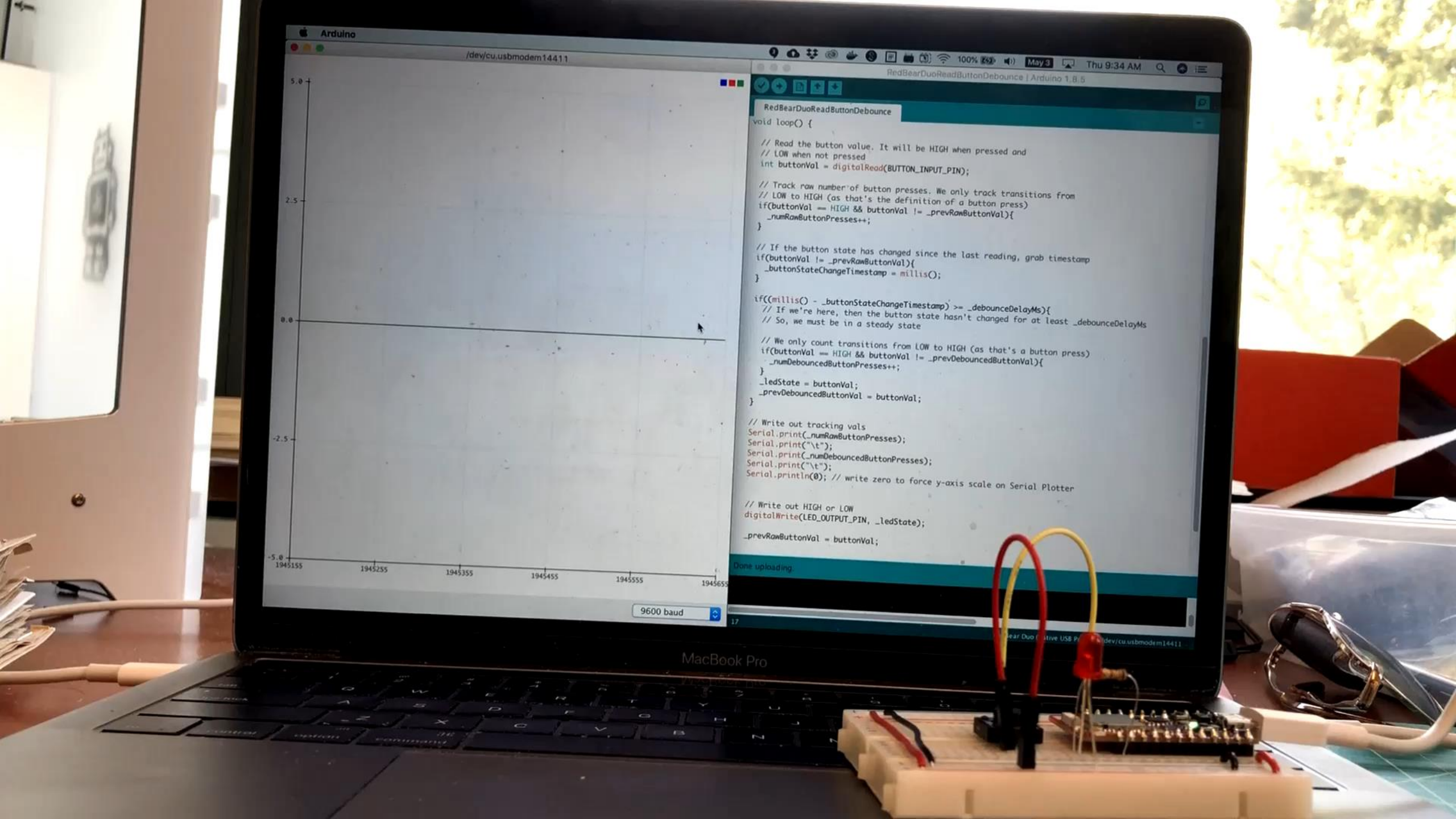

RIGOL
STOP
H 5.00ms
50.0MSa/s
6.00M pts
D 10.4000000ms
T 2.00 V
Horizontal
Period
Freq
Rise Time
Fall Time
+Width
-Width
AX: = -200.0us
AY: = 8.240 V
BX: = 28.30ms
BY: = 7.440 V
BX-AX: = 28.50ms
BY-AY: = -800.0mV
1/|dX|: = 35.09 Hz
CH1
Cursor
Mode
Manual
Select
Source
CH1
CursorA
-200.0us
CursorB
28.30ms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

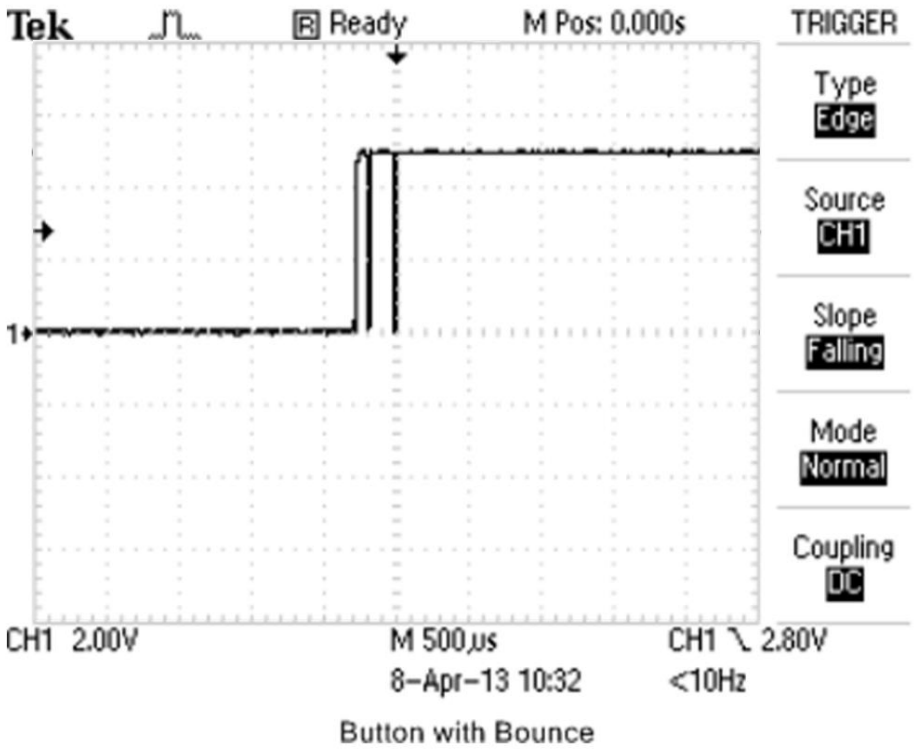


Figure 12-2: Ordinary pushbutton bouncing before settling

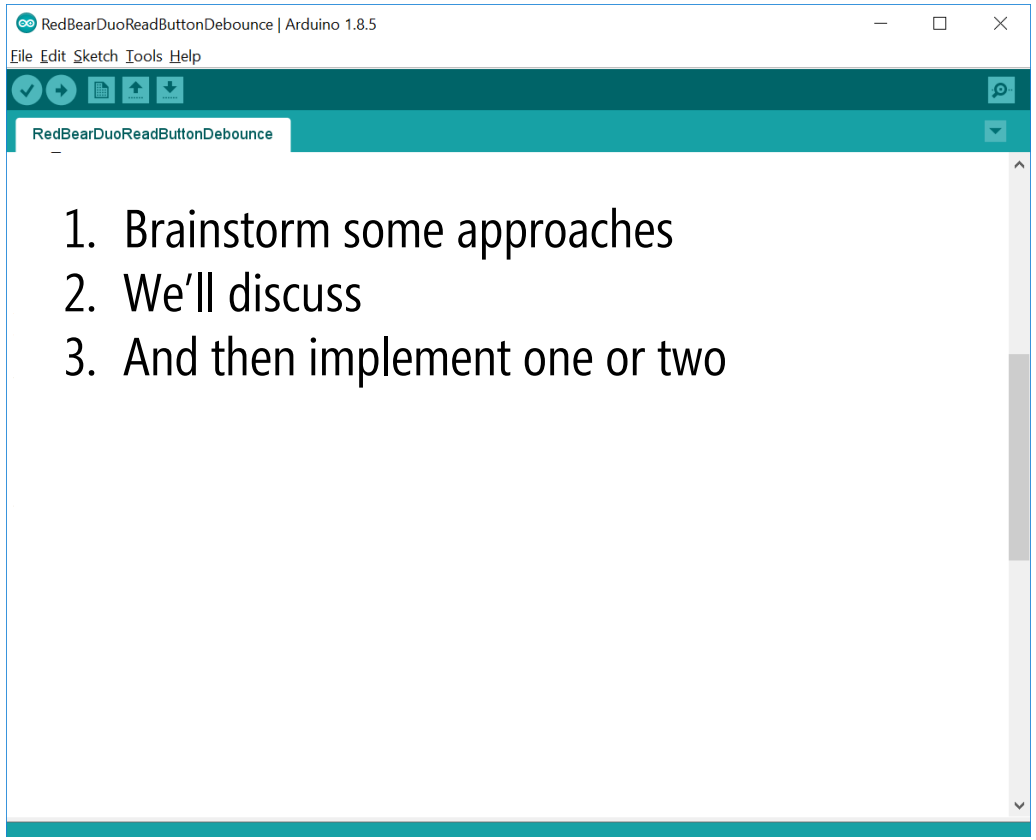


Image source: Jeremey Blum, Exploring Arduino, 2013

DIGITAL INPUT

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);
}
```

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;
}
```

```
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;
}
```

- Debounce
  - Hardware Required
  - Circuit
  - Schematic
  - Code
  - See Also

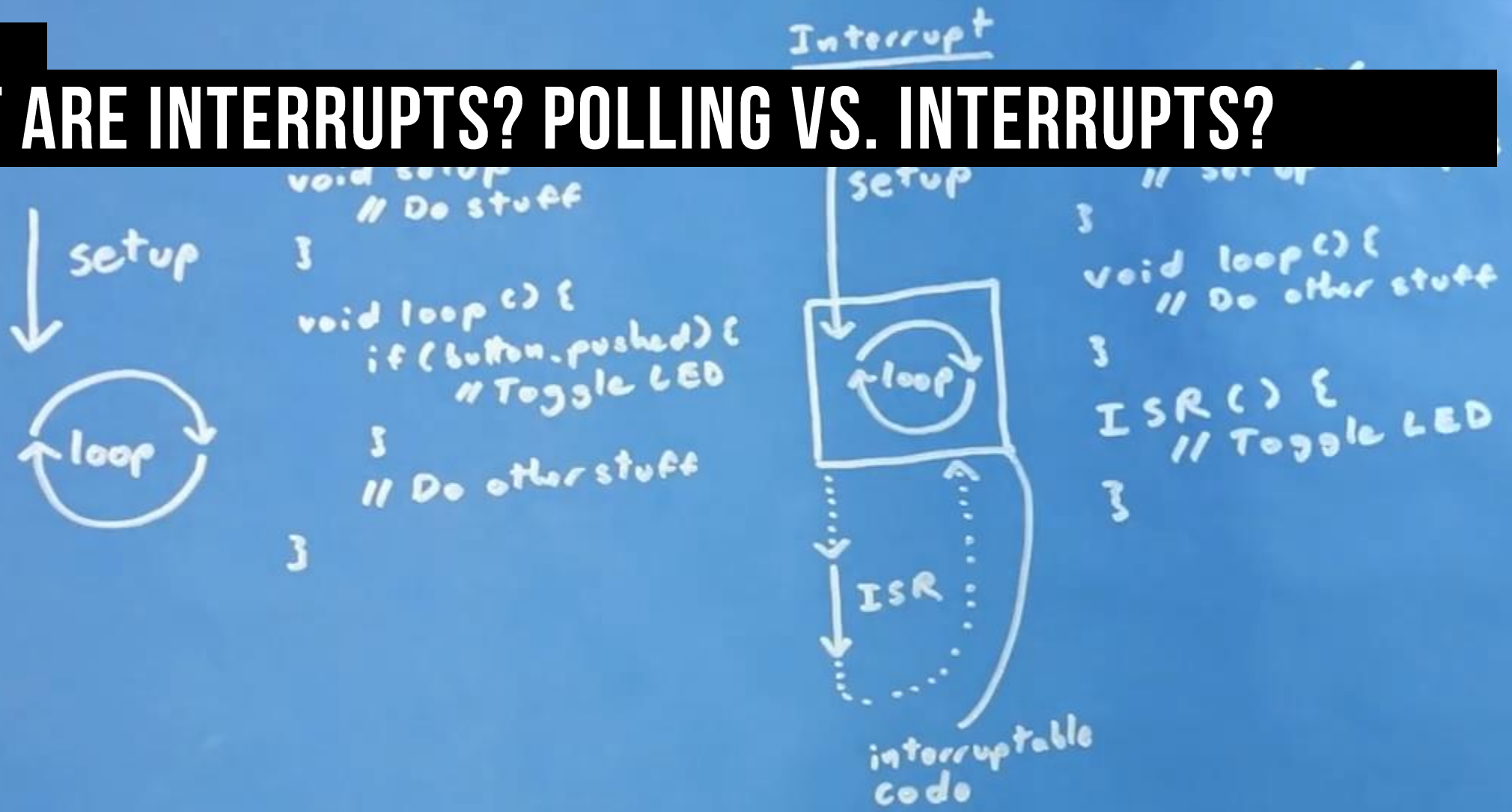
Debounce

Pushbuttons often generate spurious open/close transitions when pressed, due to mechanical and physical issues: these transitions may be read as multiple presses in a very short time fooling the program. This example demonstrates how to **debounce** an input, which means checking twice in a short period of time to make sure the pushbutton is definitely pressed. Without debouncing, pressing the button once may cause unpredictable results. This sketch uses the `millis()` function to keep track of the time passed since the button was pressed.

We can use **interrupts** to “interrupt” the loop()

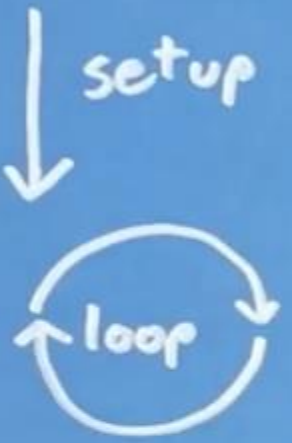
INTERRUPTS

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(pin, ISR, mode)

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

Syntax

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

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: LOW, CHANGE, RISING, FALLING

Returns

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

https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

PINS USED FOR INTERRUPTS

BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno, Nano, Mini, other 328-based	2, 3
Uno WiFi Rev.2	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR Family boards	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Due	all digital pins
101	all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with **CHANGE**)

https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

INTERRUPTS

EXAMPLE

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

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

TILT SWITCHES

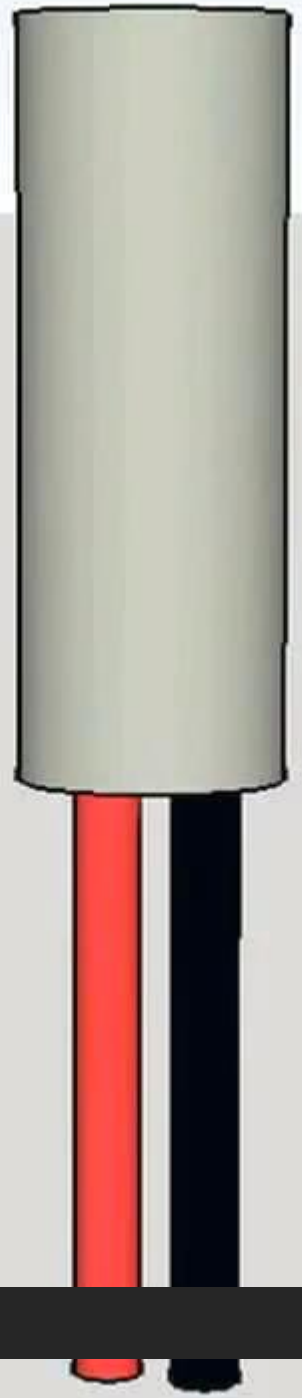
DIGITAL INPUT

TILT SWITCH

This AT407 basic tilt switch can easily be used to detect orientation. Inside the can is a ball that make contact with the pins when the case is upright. Tilt the case over and the balls don't touch, thus not making a connection.
There are numerous uses for these basic sensors, but keep in mind you might need to use some debouncing code, as the sensor isn't immune to small vibrations and such.

DIGITAL INPUT

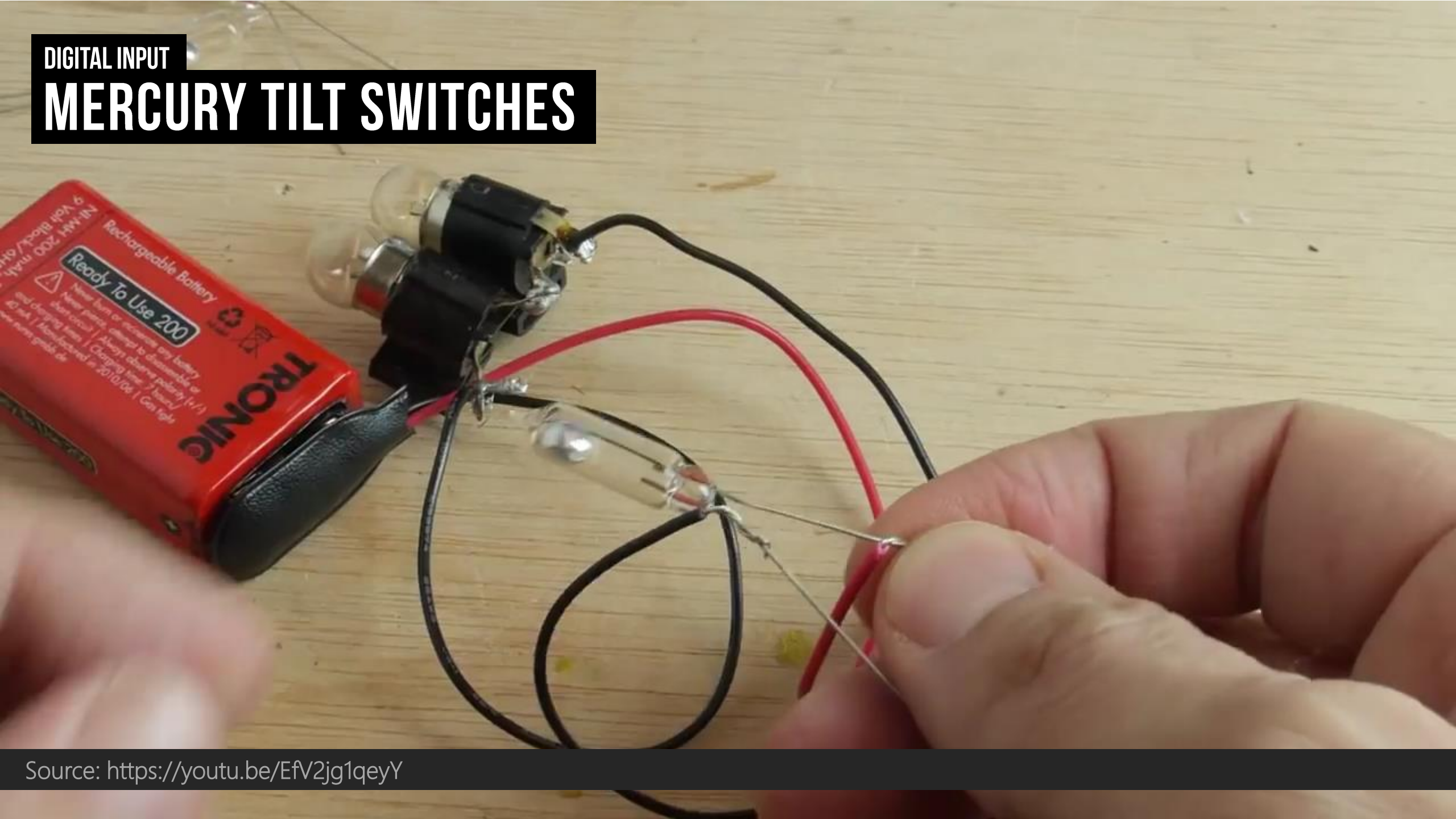
BALL TILT SWITCHES



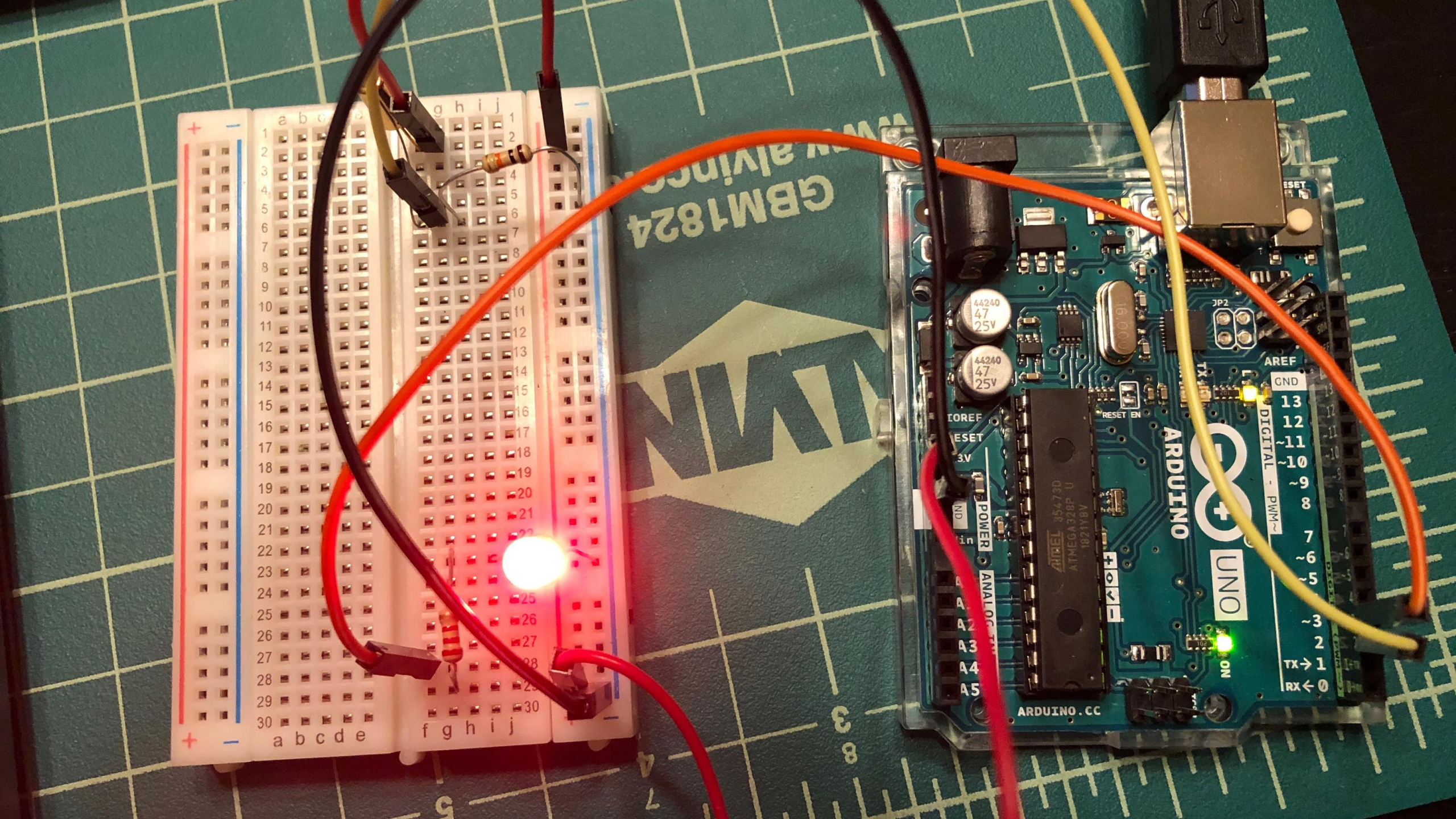
Source: https://youtu.be/QZWmrRkvHJk

DIGITAL INPUT

MERCURY TILT SWITCHES

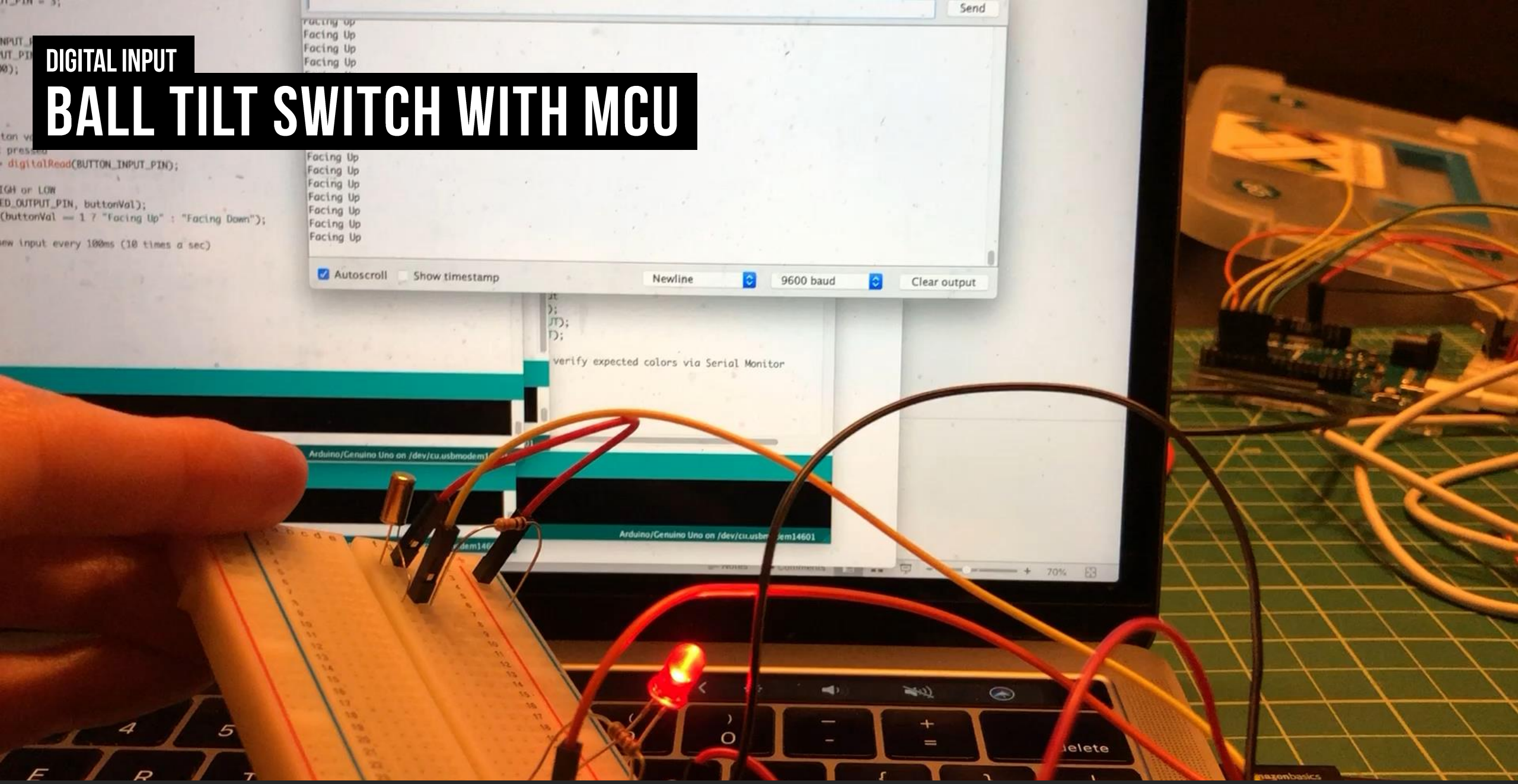


Source: https://youtu.be/EfV2jg1qeyY

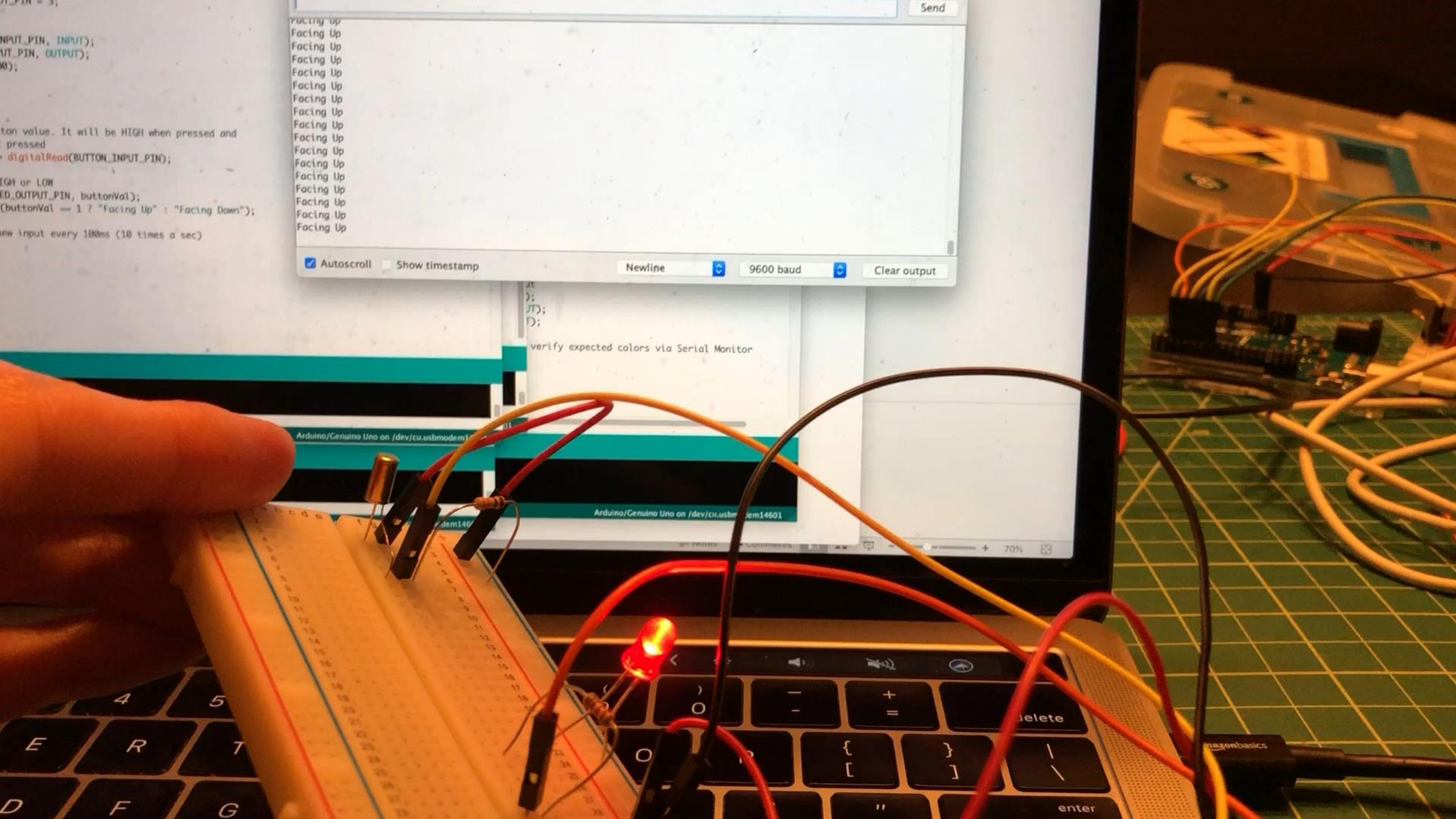

ARDUINO
UNO
ARDUINO.CC
POWER
ANALOG IN
DIGITAL - PWM~
AREF
GND
ON
TX→1
RX←0
ATMEGA328P
44240
47
25V

DIGITAL INPUT

BALL TILT SWITCH WITH MCU

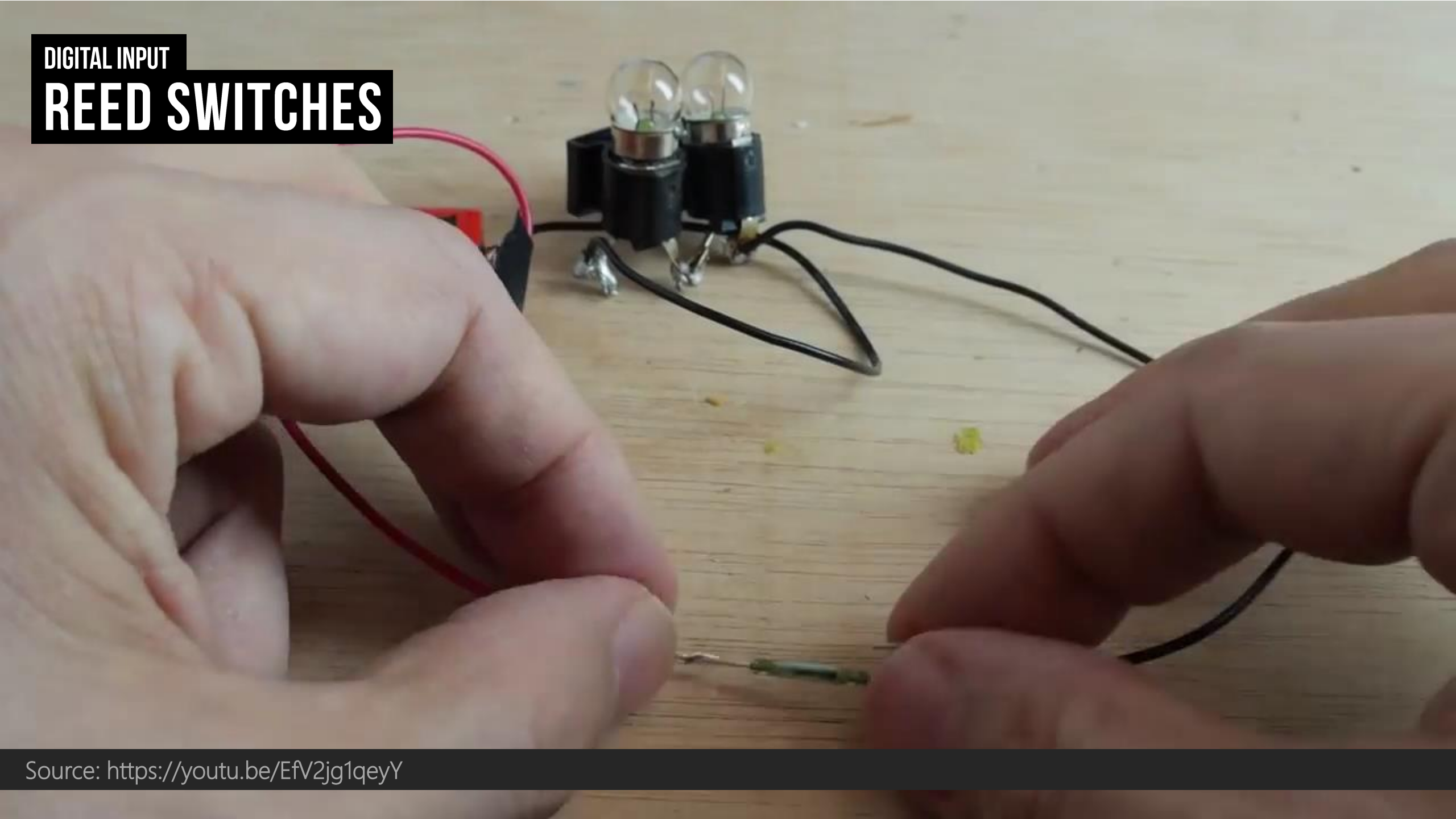


Video by Jon Froehlich


Send
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Facing Up
Autoscroll
Show timestamp
Newline
9600 baud
Clear output
verify expected colors via Serial Monitor
Arduino/Genuino Uno on /dev/cu.usbmodem14601
70%

DIGITAL INPUT

REED SWITCHES



Source: https://youtu.be/EfV2jg1qeyY

PHYSICAL COMPUTING 5

TODAY'S LEARNING GOALS

Working with **digital input**

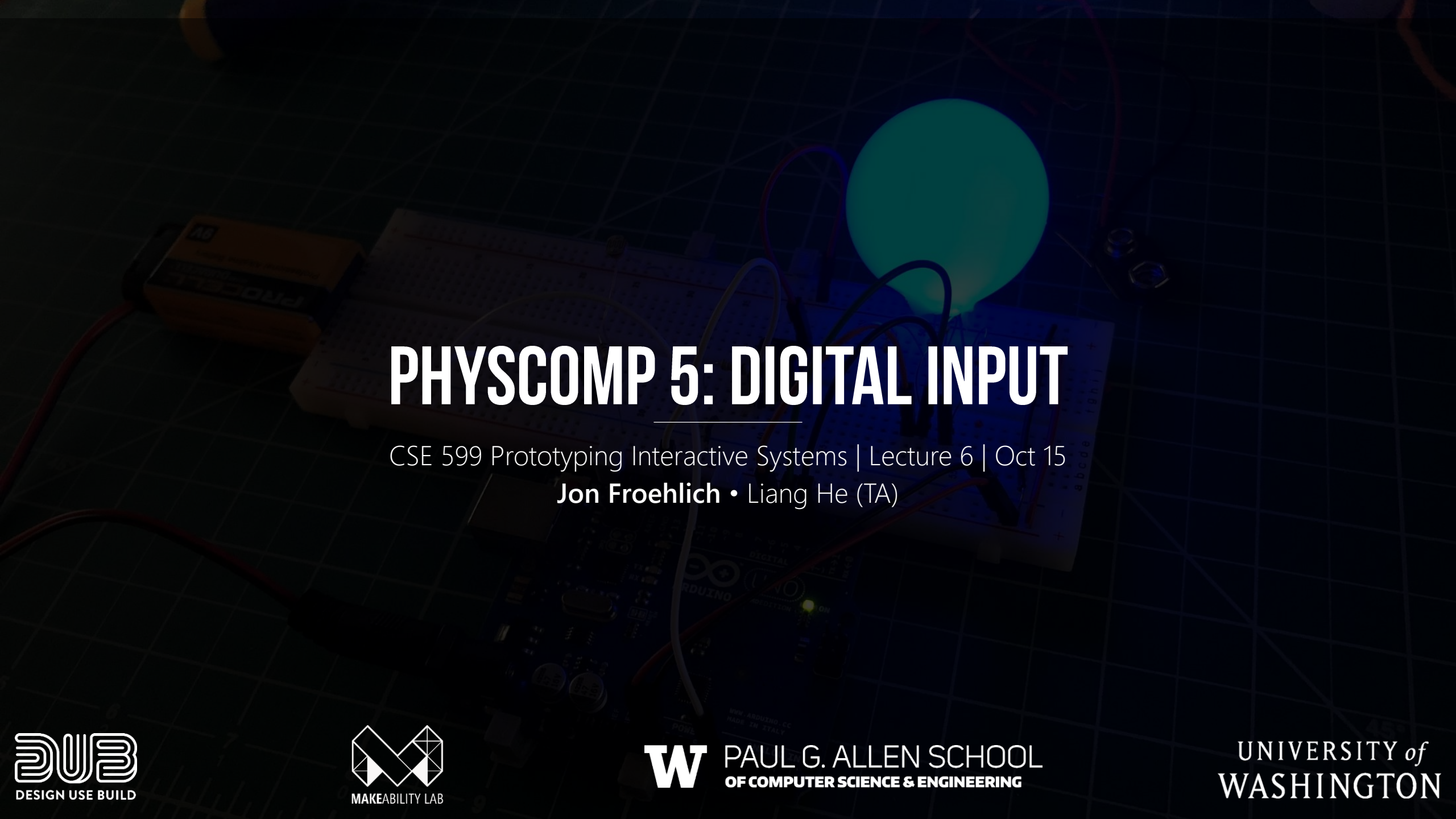
How to hook up a **button**

What are **pull-up** and **pull-down resistors** and why use them

What is **debouncing** and how to implement it

What are **interrupts** and how to use them

(If time) **tilt switches, reed switches**



PHYSCOMP 5: DIGITAL INPUT

CSE 599 Prototyping Interactive Systems | Lecture 6 | Oct 15

Jon Froehlich • Liang He (TA)

DESIGN USE BUILD

MAKEABILITY LAB

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY of WASHINGTON