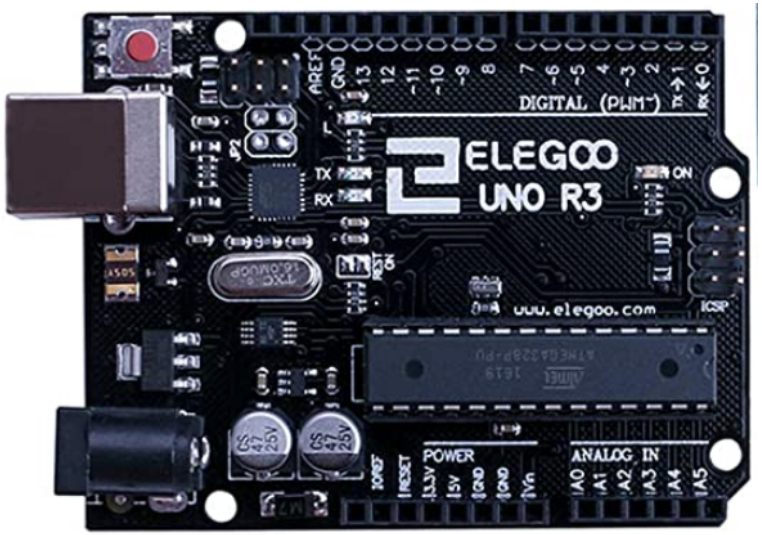
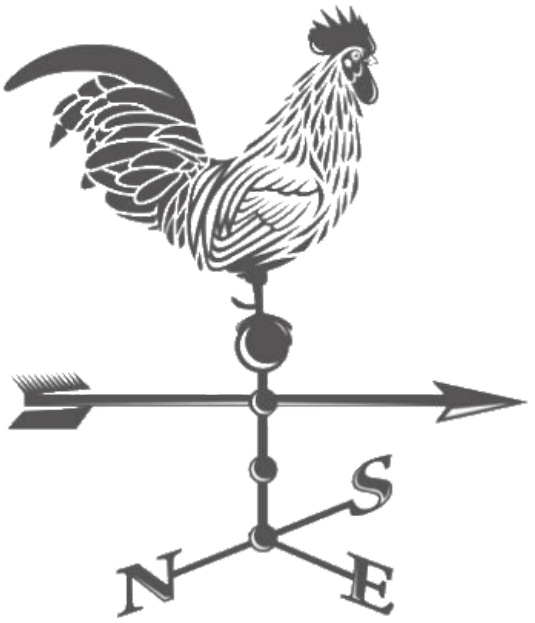
...RRRduino...

Da'Rooster, but lights first…

(for absolute beginners, like me)



→



by Gert 'Speed' Muller

v0.01

If you don't care, don't like electronics and don't want to be bothered, write this down:

www.TxNamib.com

and

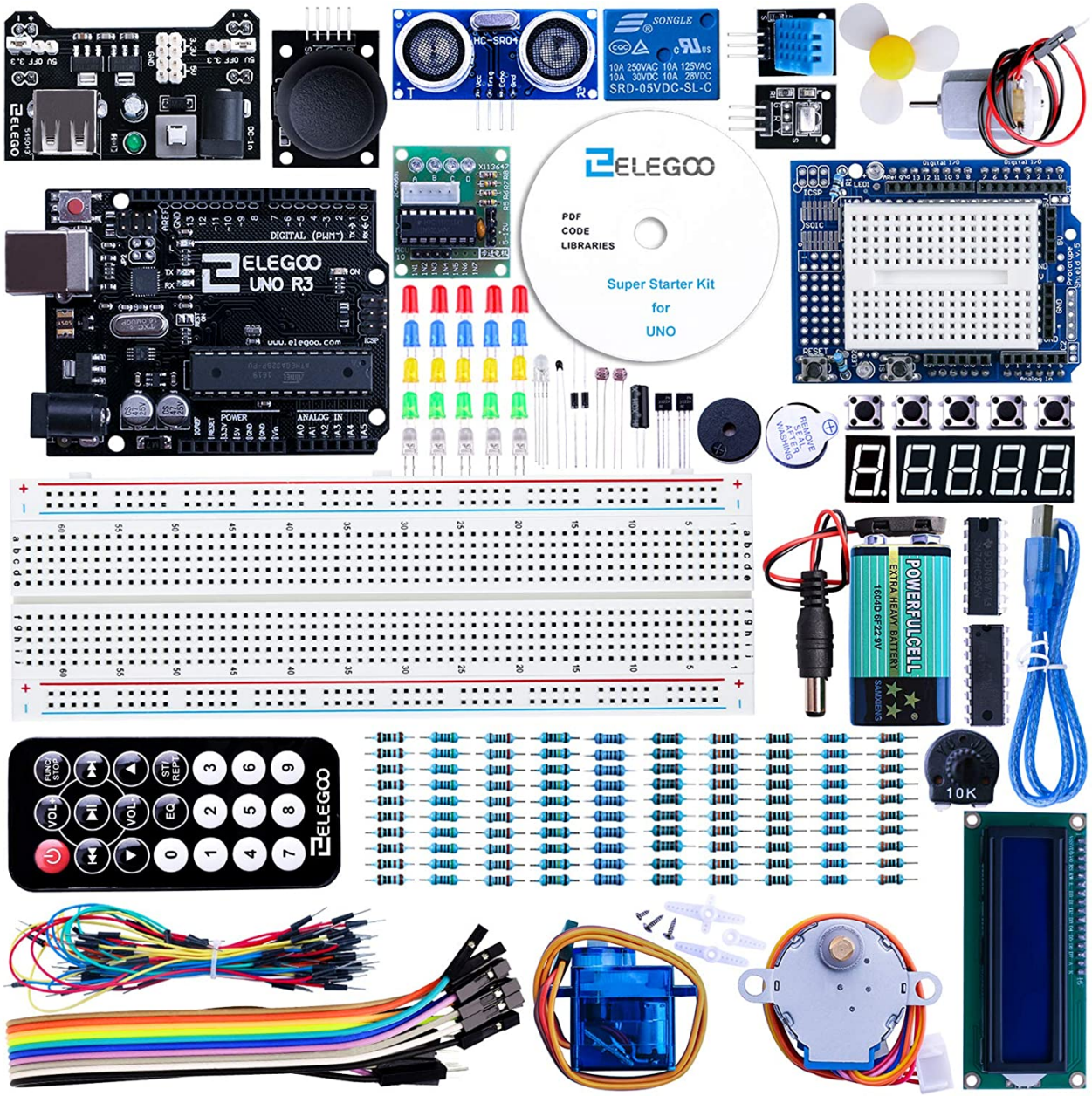
blog.RRRduino.com

and

mqTrains.com

and then go ahead, take that nap!

Parts? I think we should be ready for a clinic!



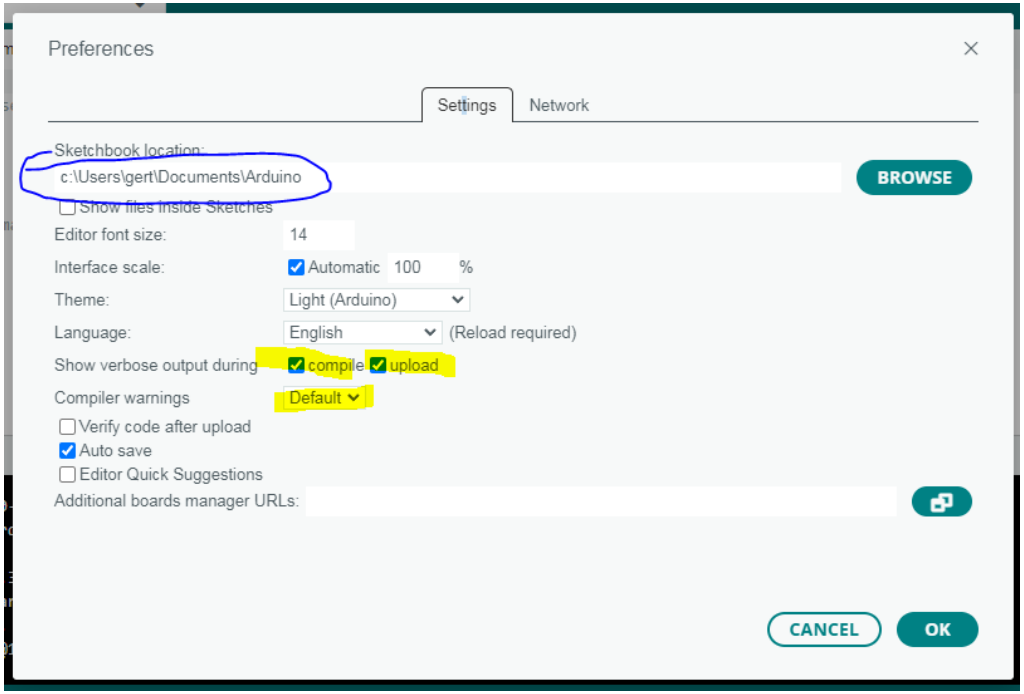
Before we start, Preferences…

- Please open the File -> Preferences (Ctrl+comma) and set the following settings:

Check:

1. Show verbose output during: [x] **compilation** and [x] **upload**
2. **Change** Compiler warnings to: **Default**
3. And then make a mental note of **where** new Sketches will be saved on your computer. Usually something like ..\Documents\Arduino, which you need to know if you want to make a backup or share a Sketch.

- Then click **OK** at the bottom to make it real.



What is an Arduino?

Quoted: (http://www.circuitstoday.com/story-and-history-of-development-of-arduino)

The new prototype board, the Arduino, created by Massimo Banzi and other founders, is a **low cost microcontroller board** that allows even a **novice** to do great things in electronics. An Arduino can be connected to all kinds of **lights**, **motors**, **sensors** and **other devices**; an easy-to-learn programming language can be used to program how the new creation behaves. Using the Arduino, you can build an **interactive display** or a **mobile robot** or anything that you can imagine.

You can purchase an Arduino board for **just about US $30** or **build your own** board from scratch. Consequently, Arduino has become the most powerful open source hardware movement of its time.

Today, there are Arduino-based **LED cubes**, **Twitter displays**, **DNA analysis kits**, **breathalyzers** and so much more. There are Arduino parties and Arduino clubs. As a feather to its crown, Google has recently released an Arduino-based development kit for its Android Smartphone!

FOR US? **Flickering lights, fire trucks, ambulances, police cars, crossing gates (even bringing the gates down WITH the bell ringing), signals, semaphores, turnout control, airfield lights, animation with servos, steppers and DC motors. Sensors, counting and reporting axles, and randomly nagging about a hot wheel! Also LCC, BlueTooth, WiFi, CAN Bus, transmitting data across your layout.**

How did it come about?

Quoted (http://www.circuitstoday.com/story-and-history-of-development-of-arduino):
It was in the Interactive Design Institute that a **hardware thesis** was contributed for a wiring design by a Colombian student named **Hernando Barragan**. The title of the thesis was "Arduino–La rivoluzione dell'open hardware" ("Arduino – The Revolution of Open Hardware"). Yes, it sounded a little different from the usual thesis but none would have imagined that it would carve a niche in the field of electronics. A team of **five developers** worked on this thesis and when the new wiring platform was complete, they worked to make it much lighter, less expensive, and available to the open source community.

...the Story in more Detail…

As mentioned earlier, it all started in **Ivrea, Italy**. To begin with, let's have a look at how the name Arduino, which sounds quite strange for an electronic device, was chosen. This beautiful town of Ivrea, situated in Northern Italy, is quite famous for its underdog kings. In the year 1002 AD, **King Arduin** (you got it right!) ruled the country; two years later, he was dethroned by King Henry II of Germany. In the memoir of this King Arduin, there is this 'Bar Di Re Arduino', a **pub on the cobble stoned street** in the town. Well, this place is where a new era in electronics had its roots!

This bar was frequently visited by **Massimo Banzi**, one of the founders of Arduino, who taught at Ivrea. He was the one who gave the name Arduino to this low-cost microcontroller board in honor of the place!
Before getting into how the Arduino was developed and used, let's know who the core members of the Arduino developer team are: **Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis**.

The First Prototype Board

Well, Banzi succeeded in creating the first prototype board in the year **2005**; it was a simple design and at that time, it wasn't called Arduino. *Of course, by now, you would know how he had coined the name later that year.*

Open Source Model – A Big Decision

Banzi and his collaborators strongly believed in **open-source software.** As the purpose was to develop a quick and easily accessible platform, they thought it would be better to open up the project to as many people as possible instead of keeping it closed. Another crucial factor that contributed to that big decision was that after operating for nearly five years, IDII had no more funds left and was in fact going to shut its doors. All the faculty members feared that their projects might not survive or would be embezzled. It was at this crucial point of time that Banzi decided to go ahead and make it open source!

How Banzi and team managed to create Arduino and make it available for public

Pretty obviously, the open source model had always been used to fuel innovation for software and never **hardware**. If they had to make it work, they had to find a suitable licensing solution that could apply to the board. After a little investigation, Banzi and team looked at the whole thing from a different angle and decided to use a license from **Creative Commons**, a nonprofit group whose agreements were normally used for cultural works like writing and music. *According to Banzi, hardware is a piece of culture that must be shared with other people!*

Well, the next step was to make the board. The group decided to fix a specific, student-friendly price of **$30** as their **goal**. Banzi felt that the Arduino should be affordable for all students. However, they also wanted to make it really quirky, something that would stand out and look cool as well. While other boards were green, they wanted to make theirs **blue**. While a few manufacturers saved on input and output pins, they added a lot to their board. Quite weirdly, they added a little **map of Italy** on the back of the Arduino board!

Key to the Slides:

This is some information

Here is some more information

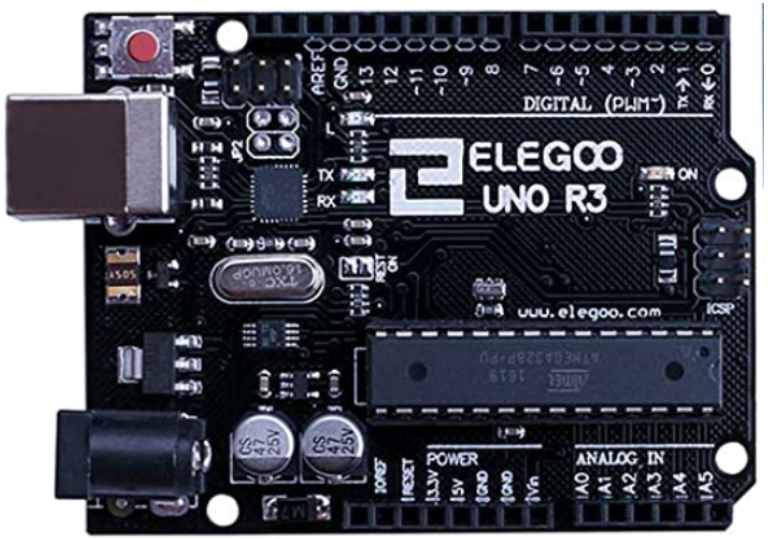
You have to do something…

- So each step you need to act on, is very likely to be in … yes, blue!

Where to Start?

- Buy one ... and plug it in!

[PC → USB Cable → Uno]



See a **solid** red light and a **blinking** red* light?

Where did we get it?

https://www.amazon.com/dp/B01D8KOZF4/ref=cm_sw_r_apa_i_dl_HQTZ9VNFP9ZVC2JTBAJP_0

ELEGOO UNO Project Super Starter Kit with Tutorial and UNO R3 Compatible with Arduino IDE

Visit the ELEGOO Store

4.5 out of 5 stars 13,188 ratings | 234 answered questions

#1 Best Seller in Single Board Computers

$44.99

prime

FREE Returns

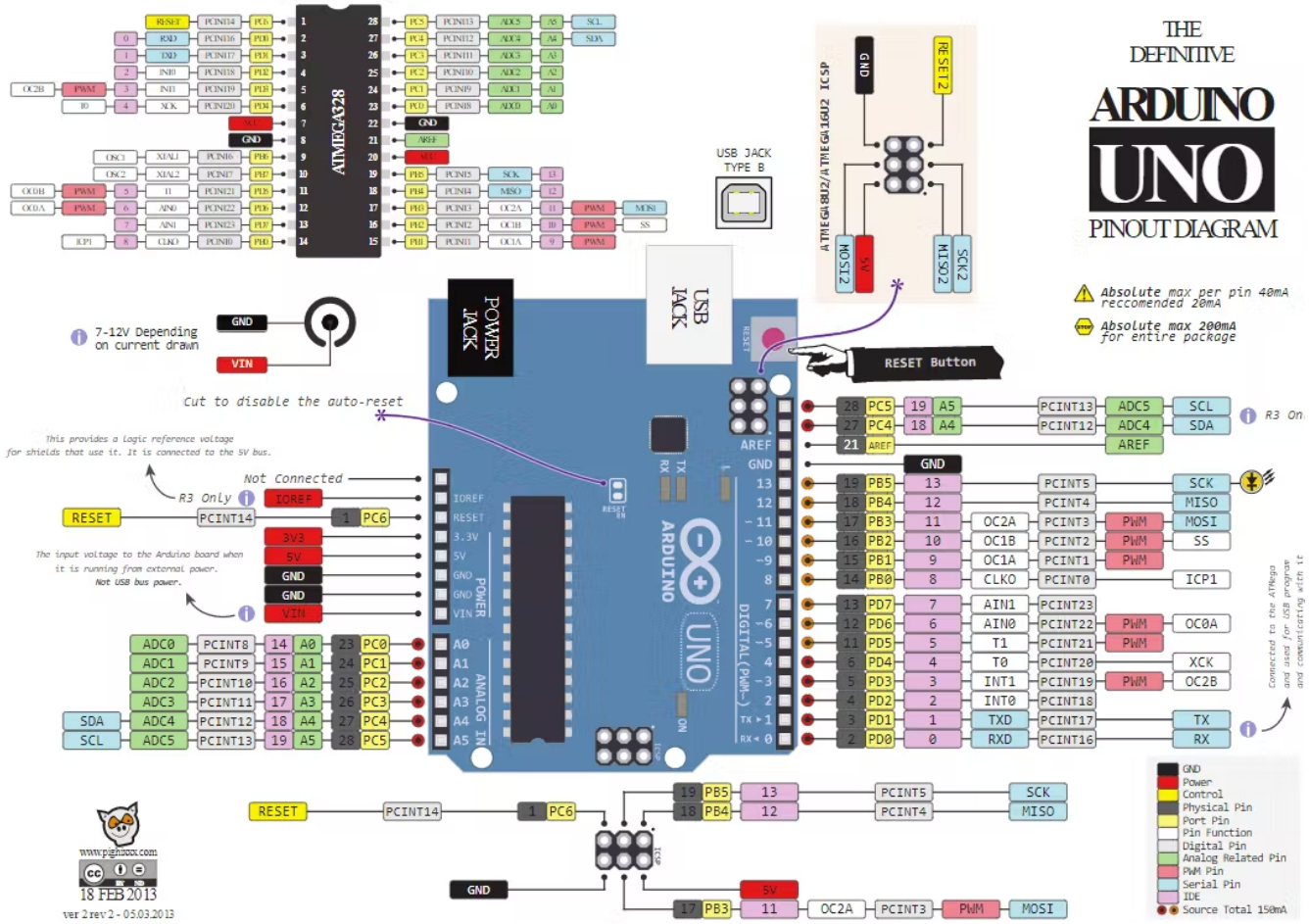
Save up to 15% with business pricing. Sign up for free Amazon Business account

Brand	ELEGOO
Color	UNO Super Starter Kit
Connectivity Technology	USB
Included Components	Component listing: 1pcs UNO Controller Board 1pcs LCD1602 Module (with pin header) 1pcs Breadboard Expansion Board 1pcs Power Supply Module WARNING: Pls. do not use the voltage higher than 9V 1pcs Joystick Module 1pcs IR Receiv… See more
Wireless Communication Standard	Infrared

Betty Mitchell 01/30/2023 1:46 PM

Here is what we just plugged in:

(Open Source, Open Hardware, so you get the schematic too)

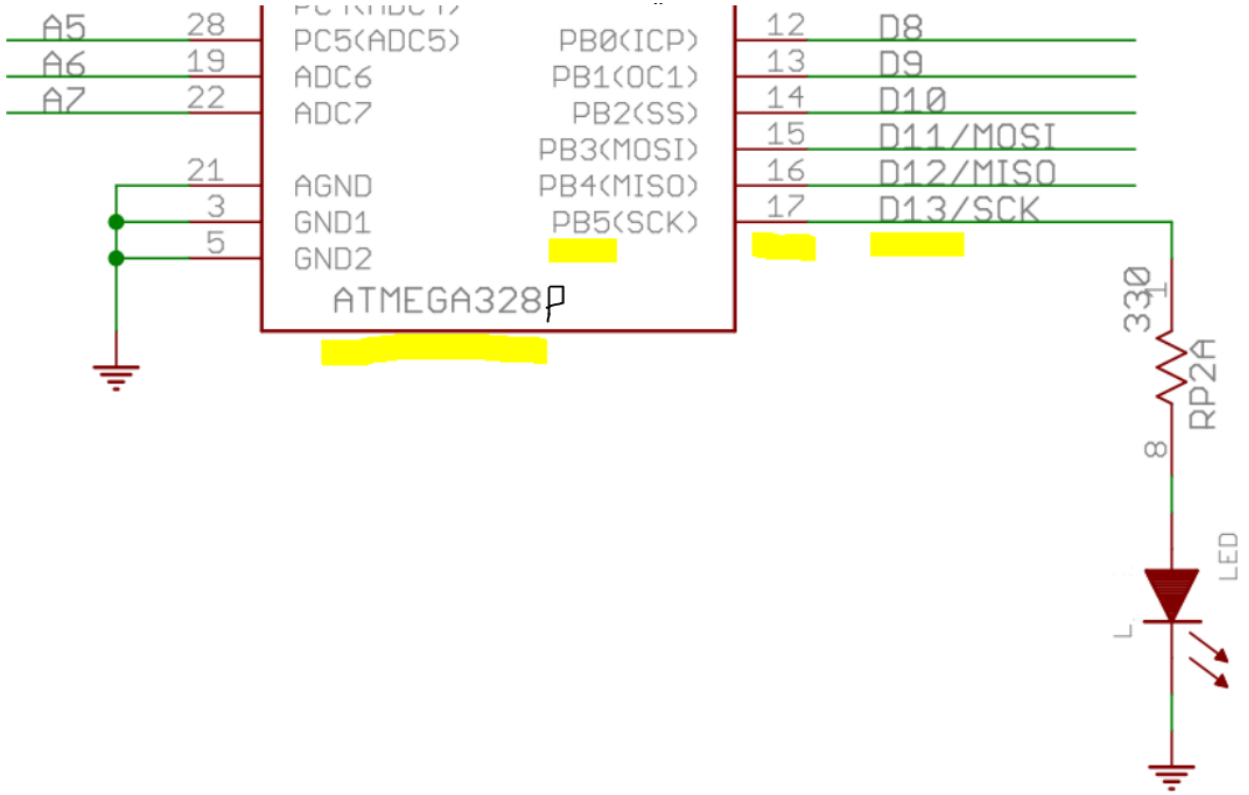


What you care about, is the **black** numbers printed ON the PC board, since those are the numbers the software needs.

Green indicated pins needed for Analog **inputs**, but all the pins can do digital things.

Also note the lines on D3, D5, D6, D9, D10 and D11 in pink, those can be **PWM** outputs...in simple terms, they could look like an Analog **output** with a real resistor and capacitor as filter.

Pins, Ports and Numbers:



- The Atmel ATMEGA328P-PU chip has pin 17 (labeled Port.B bit 5 and Serial Clock) connected to the Arduino pin D13 … and hooked to an LED on the Arduino board. So if you toggle D13, the LED toggles on and off
- Microcontrollers can usually also sink more current than source current, so when we become advanced designers we will put the LED and resistor between the power source and the pin…and then make the pin

OK, Back up for 1 minute:

What is **Arduino** again?

- It certainly has **Hardware**

- yes we know (and you get to see the schematic and board layout, like in the real files containing it, I mean)

- it has **shields** and **plug-ins** with interfaces to almost everything else in the world, and you can add the missing ones!

- It has a **Software** development platform

- IDE, C/C++ language, library extensions, compiler, upload tool and serial monitor, oops Bootloader too.

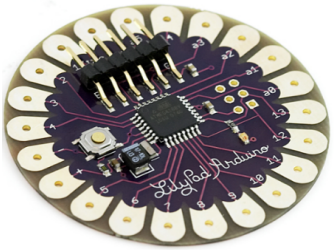
- It has a **following**, even today's kids know about it

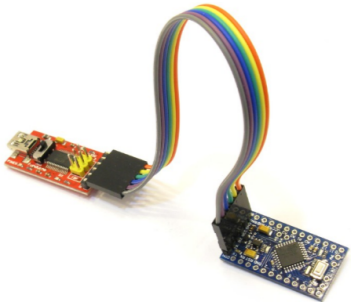
- It is world wide! It has websites, blogs, examples, even Facebook and Twitter, don't you?

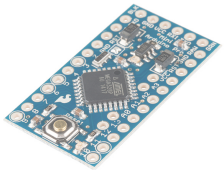
...It is a whole **platform** with tools...

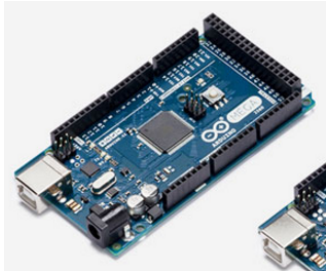
Hardware?

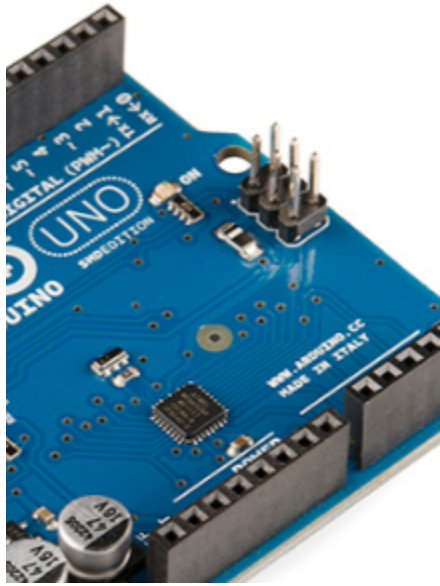
UNO was first: Atmel ATMEGA328P in a PU package, with an “NFL” size USB plug:







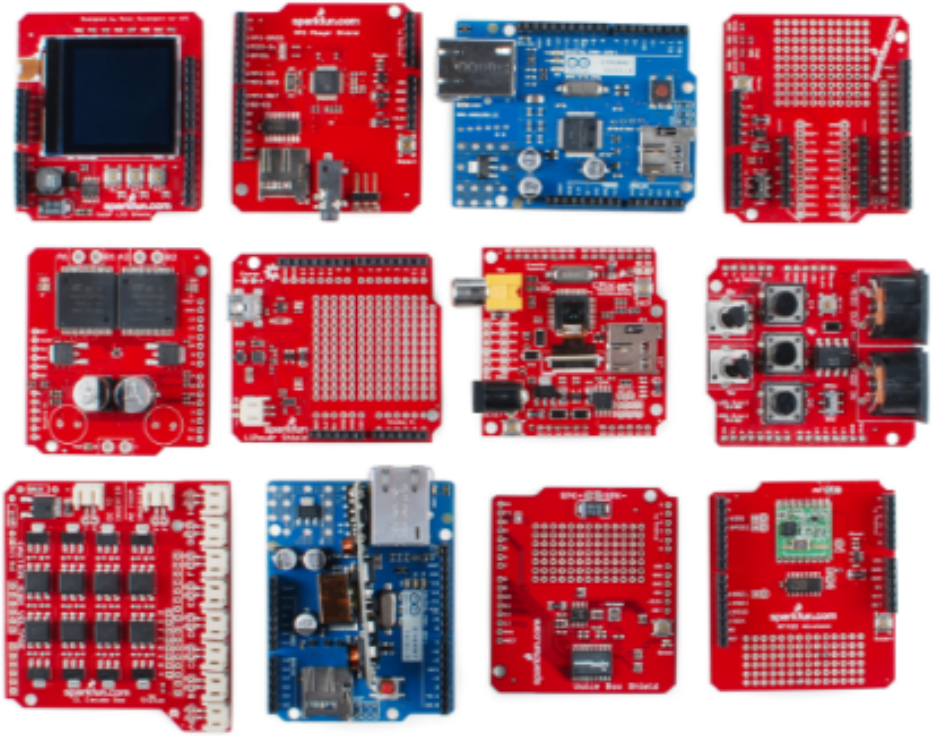




Then an SMD (surface mount) UNO version was made.

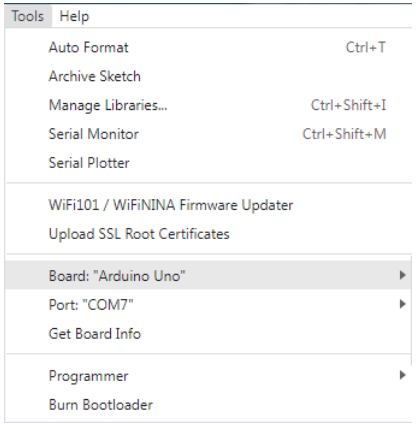
A short word on Shields

- That is how Arduinos connect to the world, if the functions is not already present
  - Simple “breadboard” Shield, so you could solder a wire to something
  - Motor Shield: DC, Stepper, Servo Motor
  - LCD Shield
  - Audio Amplifier
  - SD Card reader
  - Ethernet or WiFi Shield
  - CAN Bus
- Of course, different Shields needed for different Arduinos, Nano vs Uno
  - Plan ahead!
  - Or make your own, Fritzing, KiCAD and EasyEDA.com is free



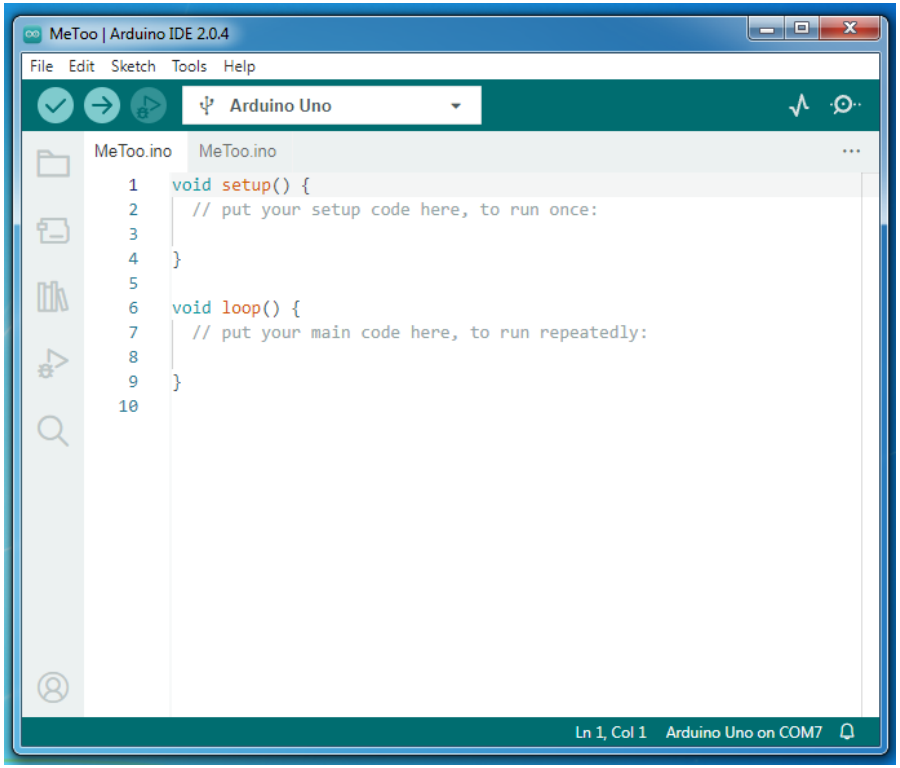
Programmers

- Bootloader
- FTDI most common, CH340G needs a device driver. Read about FTDI-gate.
- An Arduino can become a programmer to program another
- ATMEGA328P-xx vs no’P’-xx
  - Low power version vs lower cost version



And then you said Software!

- **IDE** (integrated development environment), free download
  - Windows, Mac and Linux (last one: sudo apt-get arduino, same on Raspberry Pi)
  - 2.0.4 is latest during this Div 1 Clinic in March 2023
- **Fancy editor** colors and calling things in the background!
- **Select** your board, processor and serial port from the Tools menu
- Type your **code**, **Verify** (compile) and **Upload**!
- Tip: the upload button does it all, wait…it does not type your code, ;)
- When you save a file, it will be under the Files->Skecthbook menu
- **Serial Port Monitor** (Ctrl+Shift+M) and also a **Serial Plotter** (Ctrl+Shift+L)



icons for: VERIFY, UPLOAD, NEW, OPEN, SAVE, SERIAL MONITOR.

First example on the Uno

Uno still plugged in?

- Start the software 
- Tools → Board → Arduino AVR Boards → Arduino Uno // done only once
- Tools → Processor → ATmega328P // if using a Nano
- Tools → Port → COMx (or /dev/ttyUSBx) // done only once
- File → Examples → 01.Basics → **Blink**

```
void setup( ) {
  pinMode( LED_BUILTIN, OUTPUT );
} // setup( )

void loop( ) {
  digitalWrite( LED_BUILTIN, HIGH );
  delay( 1000 );  // 1,000 ms is 1 second
  digitalWrite( LED_BUILTIN, LOW );
  delay( 1000 );
} // loop( )
```

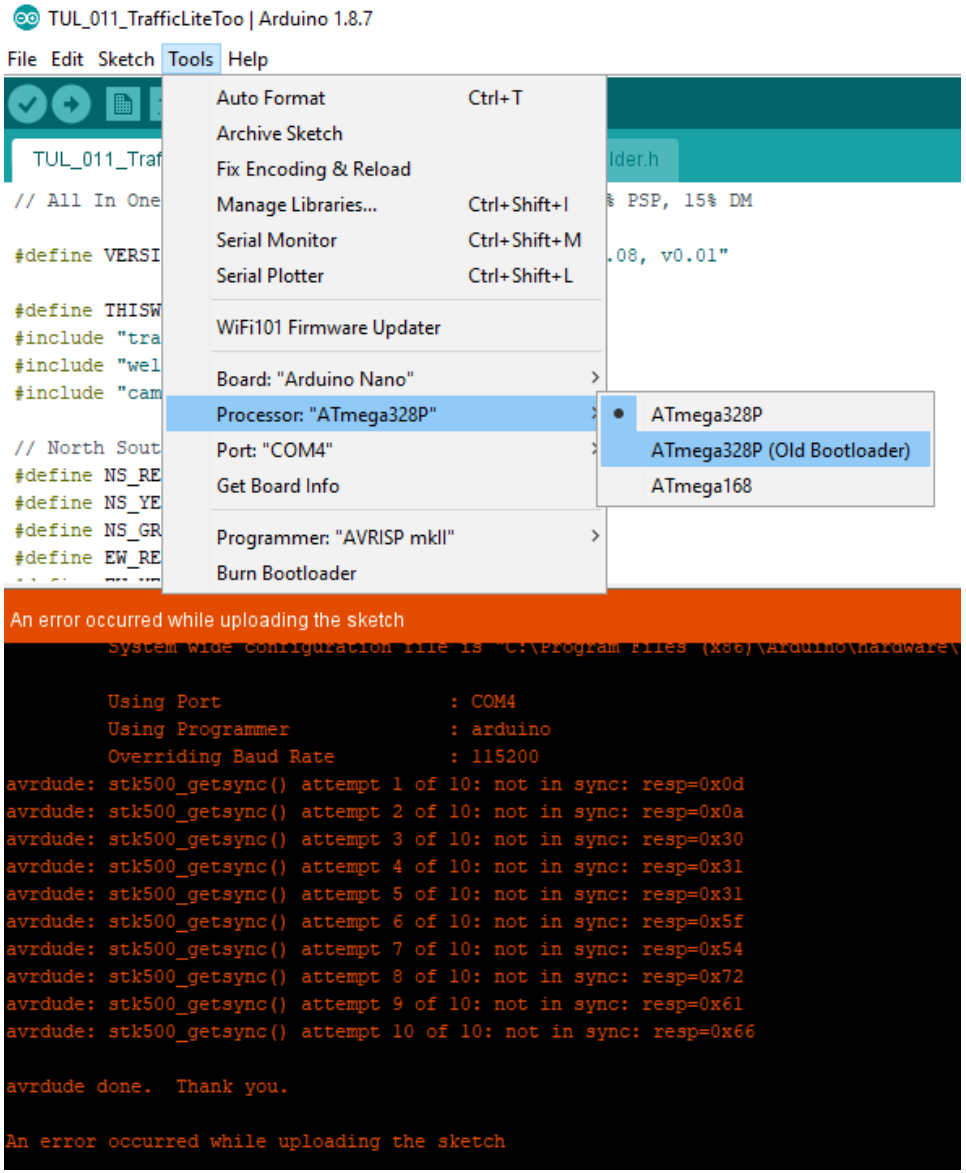
First Try…

- With File → Examples → 01.Basics → **Blink** loaded in the IDE…
- Press Ctrl+R (**Verify**) and note the stuff scrolling by in the bottom panel
- Now press **Upload** (Ctrl+U)
- Notice **LED** on **D13** still blinking?

```
Detecting Libraries Used…
Generating function prototypes…
Compiling sketch…
"C:\\Program Files (x86)\\Arduino\\hardware\\tools\\avr/bin/avr-g++" -c -g -Os
-w -std=gnu++11 -fpermissive -fno-exceptions -ffunction-sections
-fdata-sections -fno-threadsafe-statics -Wno-error=narrowing -MMD -flto
-mmcu=atmega328p -DF_CPU=16000000L -DARDUINO=10819 -DARDUINO_AVR_NANO
-DARDUINO_ARCH_AVR "-IC:\\Program Files
(x86)\\Arduino\\hardware\\arduino\\avr\\cores\\arduino" "-IC:\\Program Files
(x86)\\Arduino\\hardware\\arduino\\avr\\variants\\eightanaloginputs"
"C:\\Users\\Speed\\AppData\\Local\\Temp\\\898EE7…708A9449F\\sketch\\Blink.ino.c
pp" -o "C:\\Users\\Speed\\...\\\898EE7…708A9449F\\sketch\\Blink.ino.cpp.o"
Compiling libraries…
…
… (and then)

Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for
local variables. Maximum is 2048 bytes.
```

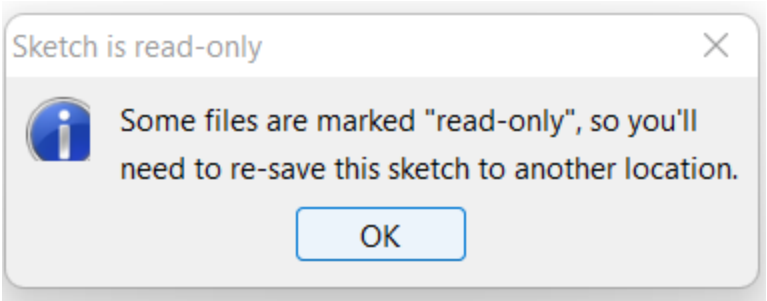
Upload Error? Nano Only…

- Ensure correct **Board**, **Port** and try Other Bootloader under Tools ⟶ **Processor**



Are you an Embedded Software (Firmware) Engineer?

- **Change** your **code** to match the red+yellow below
- **Verify/Compile** (Ctrl+R)
  - You might be forced to save the .ino file first (else Ctrl+S)
  - Note where the file goes, as well as the folder it goes into, when you have saved it. one.ino HAS to be in a folder one
  - C:\Users\<your name>\Documents\Arduino\one\one.ino
- Now press **Upload** (Ctrl+U)
- Note the change in the LED on pin 13




```
void setup( ) {
  pinMode( LED_BUILTIN, OUTPUT );
} // setup( )

void loop( ) {
  digitalWrite( LED_BUILTIN, HIGH );
  delay( 750 );
  digitalWrite( 13, LOW );
  delay( 250 );
} // loop( )
```

Useful Arduino C/C++ commands for a beginner

pinMode(pin, IN/OUTPUT);
digitalWrite(pin, HIGH/LOW);
val_Digital_True_False = **digitalRead**(pin);
analogWrite(pwm_Pin, value);
val_Analog_0_To_1023 = **analogRead**(analog_Pin);
delay(millisecond);
val_Long = **millis**( );
if(x > 5) ;
while(j < 10) { j++; }
for(j = 0; j < 10; j++) { ; }
val = **map**(value, fromLo, fromHi, toLo, toHi);
random(min, max - 1);

Wiring? Processing? Just think it is C or C++ and move on...

https://www.arduino.cc/reference/en/

Language Reference

Arduino programs can be divided in three main parts: *structure*, *values* (variables and constants), and *functions*.

Structure

- setup()
- loop()

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return

Variables

Constants

- HIGH | LOW
- INPUT | OUTPUT | INPUT_PULLUP
- LED_BUILTIN
- true | false
- integer constants
- floating point constants

Data Types

- void
- boolean
- char
- unsigned char

Functions

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

Analog I/O

- analogReference()
- analogRead()
- analogWrite() - *PWM*

Due & Zero only

- analogReadResolution()
- analogWriteResolution()

Variables: bool areWeReady = false; // = true

- The C and C++ languages define what are illegal names for variables
- There are many different types of variables and they all take up various amounts of memory
- A **char** is 8 bits (0-255), and **int** is usually 16 bits (depending on what you compile it for) (-32,768 to 32,767) and a **bool** is simply true or false (but HIGH or LOW and 1 or 0 is also used)
- To have some consistency across platforms, we do not like to use **char** or **int** or **unsigned int**, we prefer the predefined **uint8_t** and **int16_t** and **unint16_t** types
- To help remember what the variable type is, you will notice that I use a character or two in front: **u8**Value, **u16**Value or **i16**Value and **b**AreWeReady

This also helps with using CamelCase, a way to use English to name a variable with more descriptive words. The b, u8 or u16 is followed by a capital letter

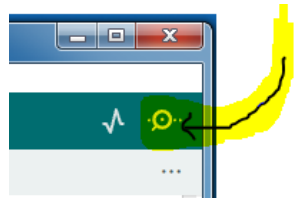
Let's go!

- Open **001_BlinkSlow_with_Serial.ino**

```
// Pin 13:  \__________/----------\__________/----------\__

#define LEDPIN      LED_BUILTIN

// Comments
void setup( ) {
  Serial.begin( 115200 );
  Serial.println( );
  Serial.println( "001_BlinkSlow_with_Serial, Div 1, 2023.03.11" );

  pinMode( LEDPIN, OUTPUT );
  digitalWrite( LEDPIN, HIGH );
} // setup( )

// Comments
void loop( ) {
  delay( 2000 );
  digitalWrite( LEDPIN, HIGH );

  delay( 2000 );
  digitalWrite( LEDPIN, LOW );
} // loop( )
```

Serial Monitor → Ctrl+Shift+M

- **Press Ctrl-Shift-M** (this opens the Serial Monitor from the IDE's Tools menu)
- **Set baud rate** on top right of window to 115200
- Loading Serial Monitor OR changing baud is like pressing the RESET button



- And this is how you find out 3 years later what is on the ATMEGA328 chip!

Tip: If your boss does not allow you to install TeraTerm, RealTerm, putty or Serial Port Monitor, install the **Arduino IDE**! Who can tell which Windows version lost Hyper Terminal?

IDE saving a Sketch

- A sketch is the .ino file in a folder with the same name
  - It is added to the build process when **main.cpp** is compiled

- No spaces in filename

- No ~~leading numbers or~~ funny characters

- **main.cpp** calls
  setup( ) once, and
  loop( ) from a repeating for(;;) { … }

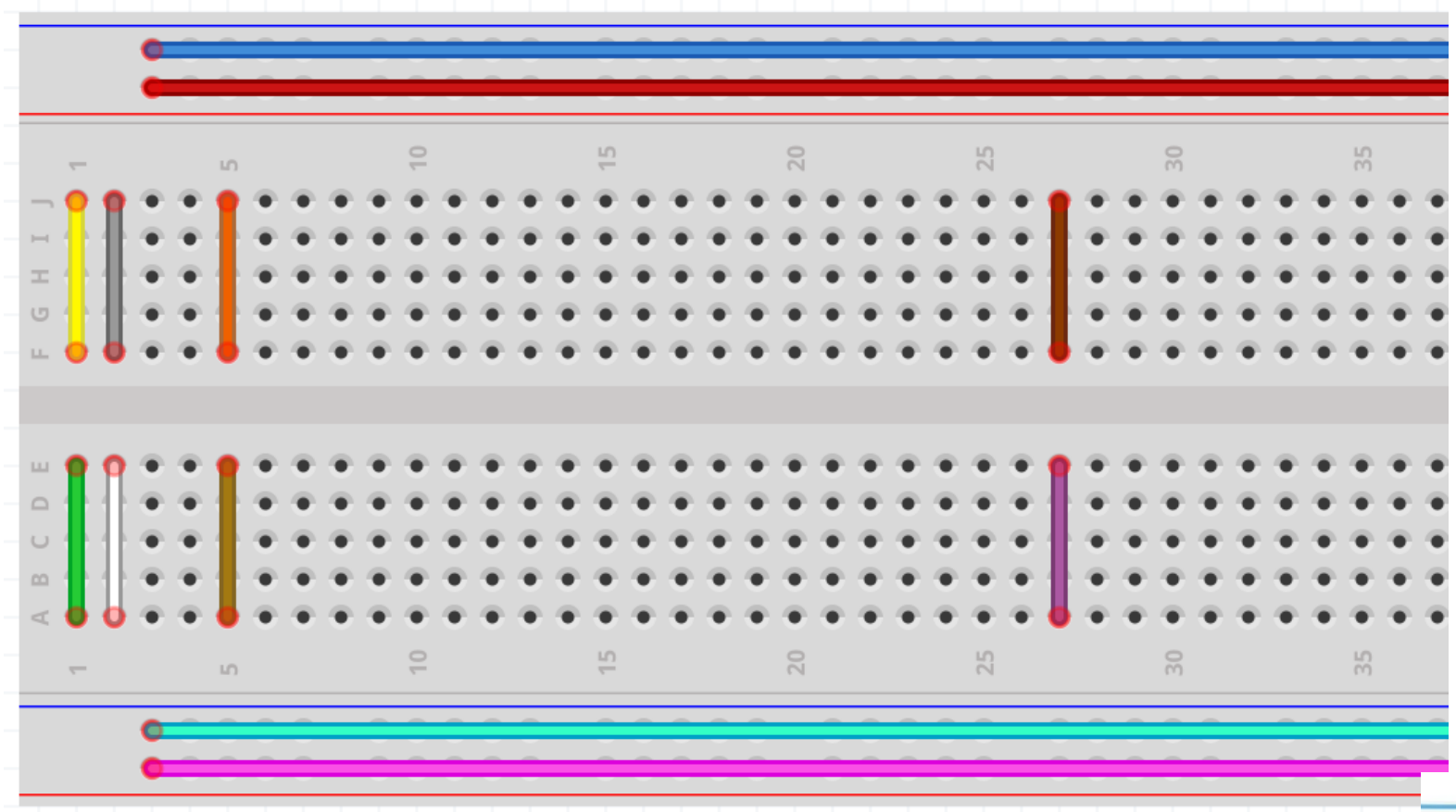
- It also checks for incoming serial data, to switch to the bootloader (already programmed in) to upload the next program when you click Upload

Next…

- Open **002_BlinkFaster.ino**

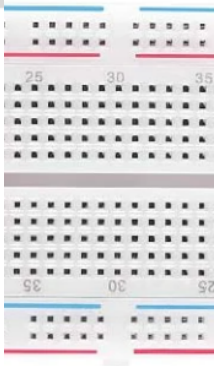
```
#define VERSION_STR    "002_Blink Faster, LSR Div1, 2023.03.11"

/*
 * Blink by setting the ONTIME and the OFFTIME separately
 * \__________________/----  ,  \__________________/----  ,  \__
 * x                  x      ,  x                  x      ,  x
*/
…
#define LEDPIN    LED_BUILTIN
#define OFFTIME          100
#define ONTIME          1900
…
void loop( ) {
  digitalWrite( LEDPIN, LOW );
  Serial.println( "Off" );
  delay( OFFTIME );

  digitalWrite( LEDPIN, HIGH );
  Serial.println( "On" );
  delay( ONTIME );
}
```

Breadboards, just a bunch of shorts!



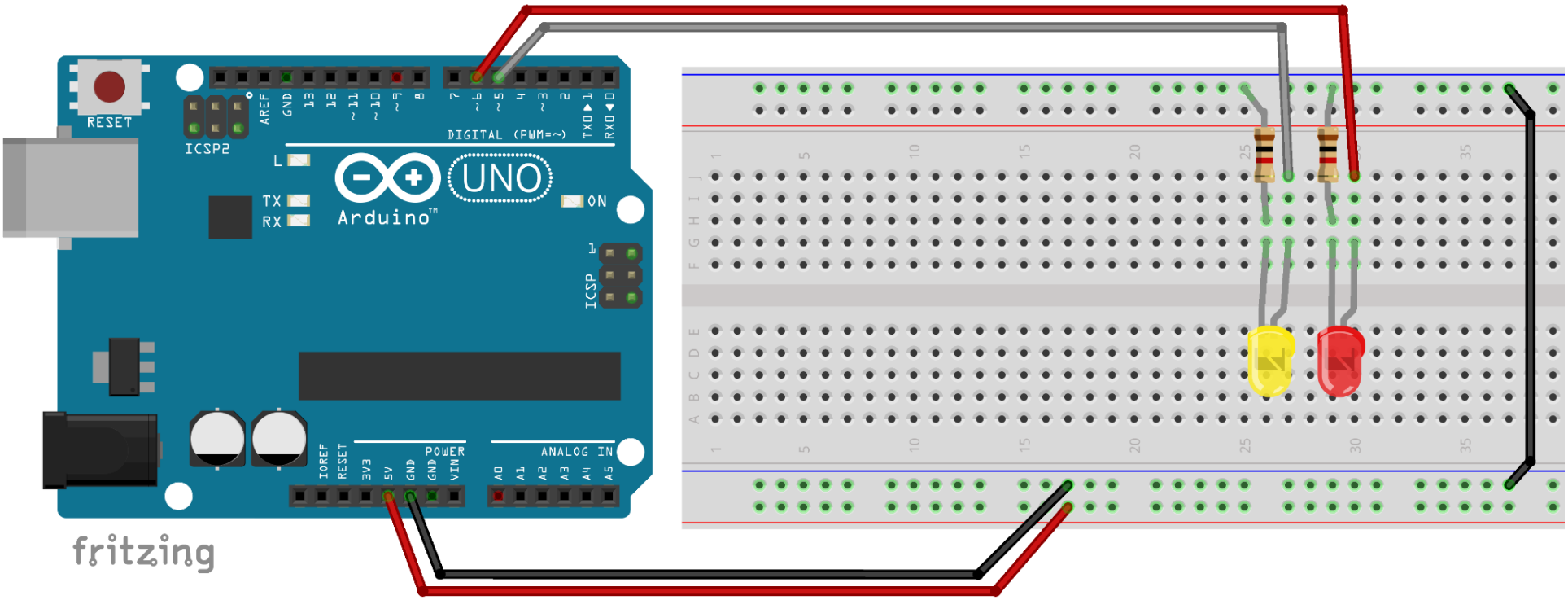
- Watch out for the ones with the blue and red bars with gaps!

  They might need to be “bridged” in the middle.



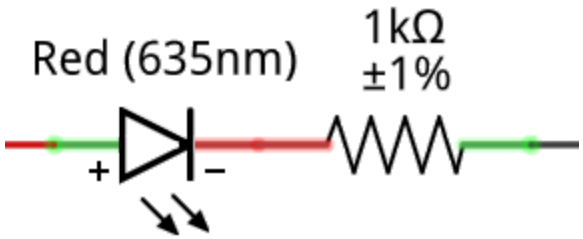
Build…

- Stick a yellow and red LED with their flat side to the left somewhere in the middle of the breadboard.
- Then white wire from ~5 to yellow and a red wire~6 to red LED's anodes (long pins for these specific devices, or "round side")
- Then add 1 kOhm resistors from the flat side to GND
- Install the black and red power wires (5V to red rail, GND to blue rail)
- And connect a black wire between the two blue rails



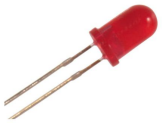
LEDs and **their** Resistors

An LED works like a one-way street. Just like cars accomplishing something by following everyone else in the same direction, an LED only allows current to flow in the direction of the triangle's arrow (from anode to cathode). It makes light once the forward voltage (Vf) across the diode is exceeded and electrical current starts flowing.



If you take a quick look at the datasheet for this Radio Shack 5 mm red LED (https://www.radioshack.com/products/5mm-red-led), you'll see that the Vf is somewhere between 2.1 and 2.8 Volts:

Forward voltage: 2.1 V (typical)/ 2.8 (max) @ 20 mA



So to make sure it will turn on, you need to do two things, provide more than 2.1 Volts **and put a resistor big enough in series** to avoid reaching the maximum rated current:

Forward Current (If): 25 mA (**Max**)

Assume we have a 5 V source to turn the LED on with and we believe the 2.0 Vf voltage, we need to use 5 - 2.1 V or 2.9 V across the current limiting resistor. From V = I x R, we can tell that if the current through the LED is the same as the current through the resistor, and we choose to run the LED with medium brightness at 3 mA:

R = V / I or R = 2.9 / 0.003 = **967 Ohms**. ← looks like 1 kOhm to me!

See datasheet or guess: I → 3 mA, V_{F_red} = ~1.9 V; $V_{F\_yel/ora}$ = ~2.0 V; V_{F_grn} = ~2.1 V; $V_{F\_blu/whi}$ = ~3.4 V;

- Run **003_TwoBeaconBlink.ino**

More than 1 beacon on the Arduino now!

Pay attention to the red ‘x’s shown here:

```
/*
 * Making two beacons work! Need external LED and resistor on pin A5
 */

// \_________________/----  ,  \_________________/----  ,  \____
// --\_________________/--  ,  --\_________________/--  ,  --\__
// x x               x x    ,  x x               x x    ,  x x

#define LEDPIN1          5
#define LEDPIN2          6

#define TIME1          250
#define TIME2         1850
#define TIME3          250
#define TIME4           50
```

- Every delay( ) value needs to be calculated between every pin change

```
void loop( ) {
  digitalWrite( LEDPIN1, LOW );
  Serial.println( "Off 1" );
  delay( TIME1 );

  digitalWrite( LEDPIN2, LOW );
  Serial.println( "Off 2" );
  delay( TIME2 );

  digitalWrite( LEDPIN1, HIGH );
  Serial.println( "On 1" );
  delay( TIME3 );

  digitalWrite( LEDPIN2, HIGH );
  Serial.println( "On 2" );
  delay( TIME4 );
} // loop( )
```

Timing in software, 3 ways…

It works, kind of: **Blocking** -> delay( )

Just wait here, do nothing else.

Better: **Polling** -> check every so often, millis( )

Is it time yet?

Best: **Interrupts** -> Let ME interrupt YOU when it is ready. Egg timer, microwave, kettle whistling

But, the ATMEGA328P only has **3 timers** to generate timer interrupts!

So, **polling** with **millis( )** it is!

https://www.arxterra.com/10-atmega328p-interrupts/#Interrupt_Basics

Polling with millis( )

- A value that increments every millisecond from power up (and clears on a reset)
- Will overflow in about 49.7 days (but not a problem here)
- **Write down** the time (save it as “previous”), and **check every so often** if a certain amount of time has passed since “previous”
- If something needs to repeat, update the “previous” and keep checking if another period has expired.

- Run **004_Millis.ino**, GO!

```
#define VERSION_STR    "004_Millis, LSR Div1, 2023.03.11"

#define LEDPIN1          5
#define TIME1          500

// Read the pin and write the opposite
void toggle( uint8_t u8ThePin ) {
  digitalWrite( u8ThePin, !digitalRead( u8ThePin ) );
} // void toggle( uint8_t )

// Code in loop( ) will execute over and over, forever after setup( ) was called
// The "for (;;) { }" repeat forever loop in main.cpp calls loop over and over
void loop( ) {
  ulNow = millis( );

  if( ulNow - ulPrevious > TIME1 ) {
    // very important to update ulBefore for next time
    ulPrevious = ulNow;
    toggle( LEDPIN1 );
  } // if
} // loop( )
```

OOP: Object Oriented Programming

Huh???

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

- Don't fear, you were already using OOP with Serial.begin(115200) and Serial.println( )
- It allows us to have a Beacon keep track of its own variables, like pin number and the "ulPrevious"
- It allows us to call methods in Beacon that does not clutter all our code in the Sketch
- It gives us an easy way to create another Beacon with less duplicating code

Introducing: “Heartbeat”! (90 beats/minute)

- If nothing works, how do you know your code is running?
- Let’s put a heartbeat in the system and then we know!
- I like it on pin 13, always! Why? Because the builtin LED is almost always there!
- **005_Blink_With_OOP.ino** it is!
- Sketch folders can also contain other files
- Notice the New Tab (Ctrl+Shift+N) down arrow at the top right
- It is common practice to #include .h file in C/C++, so we will do the same and stick our new OOP Class in there.
- Note, a **Class** tells what the code can do, but an **Object** is created (with a **Constructor)** to do the work.

Now with Beacon done by OOP!

- With millis( )

- Run **006_OOP_Beacon.ino**

```
Heartbeat myHeart  = Heartbeat( LEDPIN1, TIMEON, TIMEOFF );
Heartbeat myBeacon = Heartbeat( BEACONPIN1, BCNTIMEON, BCNTIMEOFF );

void setup( ) {
  …
  myBeacon.begin( );
  myHeart.begin( );
} // setup( )

void loop( ) {
  myBeacon.update( );
  myHeart.update( );
} // loop( )
```

- How would you do a second beacon?

  Well, add another resistor and LED and then add myBeacon2 with a pin and on and off times. Use .begin( ) and .update( )!

2 Beacons, now math

- Two of the same beacons:
  - Run **007_OOP_Beacons.ino**

```
Beacon    myBeacon1 = Beacon( BEACONPIN1, BEACONONTIME, BEACONOFFTIME );
Beacon    myBeacon2 = Beacon( BEACONPIN2, BEACONONTIME, BEACONOFFTIME );
```

with:

```
myBeacon1.begin( );
delay( 200 ); // the only good place to use a blocking call, for now in setup( )!
myBeacon2.begin( );
```

- Two independent beacons:
  - Run **008_OOP_Beacons.ino**

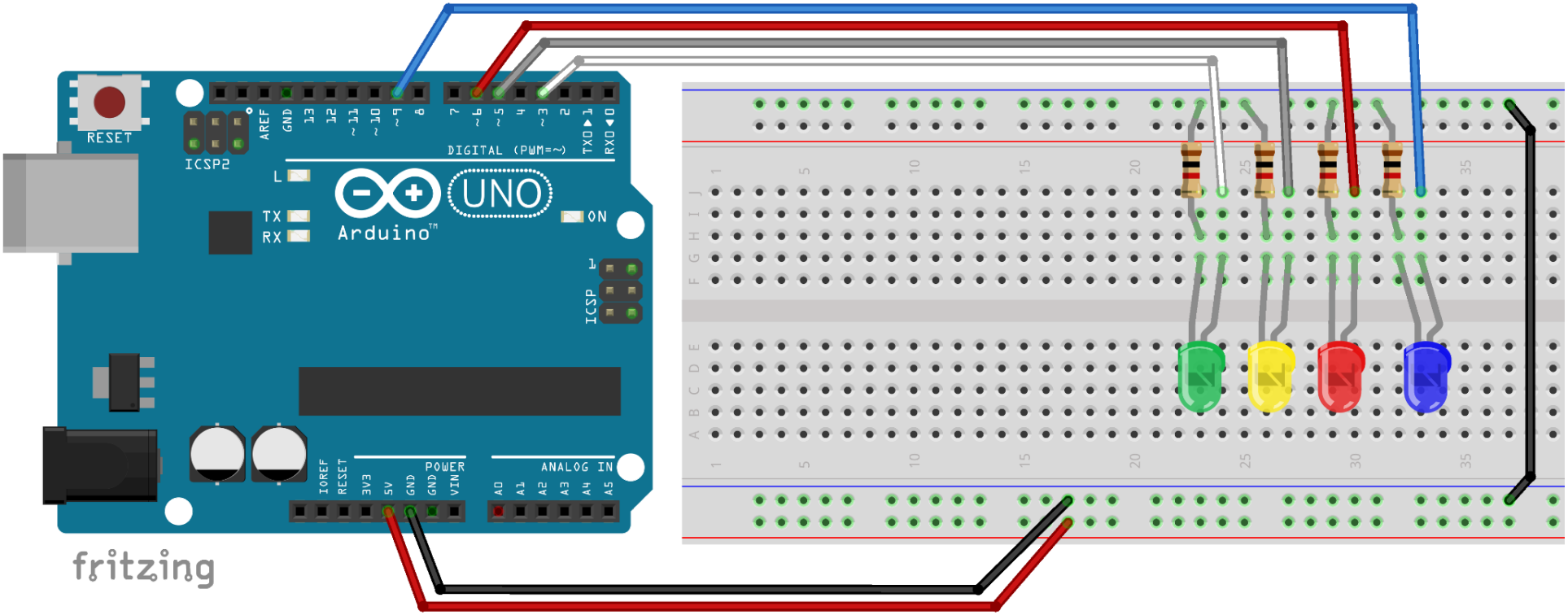
```
Beacon    myBeacon1 = Beacon( BEACONPIN1, BEACONONTIME, BEACONOFFTIME/1.5 );
Beacon    myBeacon2 = Beacon( BEACONPIN2, BEACONONTIME, BEACONOFFTIME );
```

with:

```
myBeacon1.begin( );
// delay( 200 );     // no delay needed, the beacons are independant
myBeacon2.begin( );
```

Optional, add more LEDs

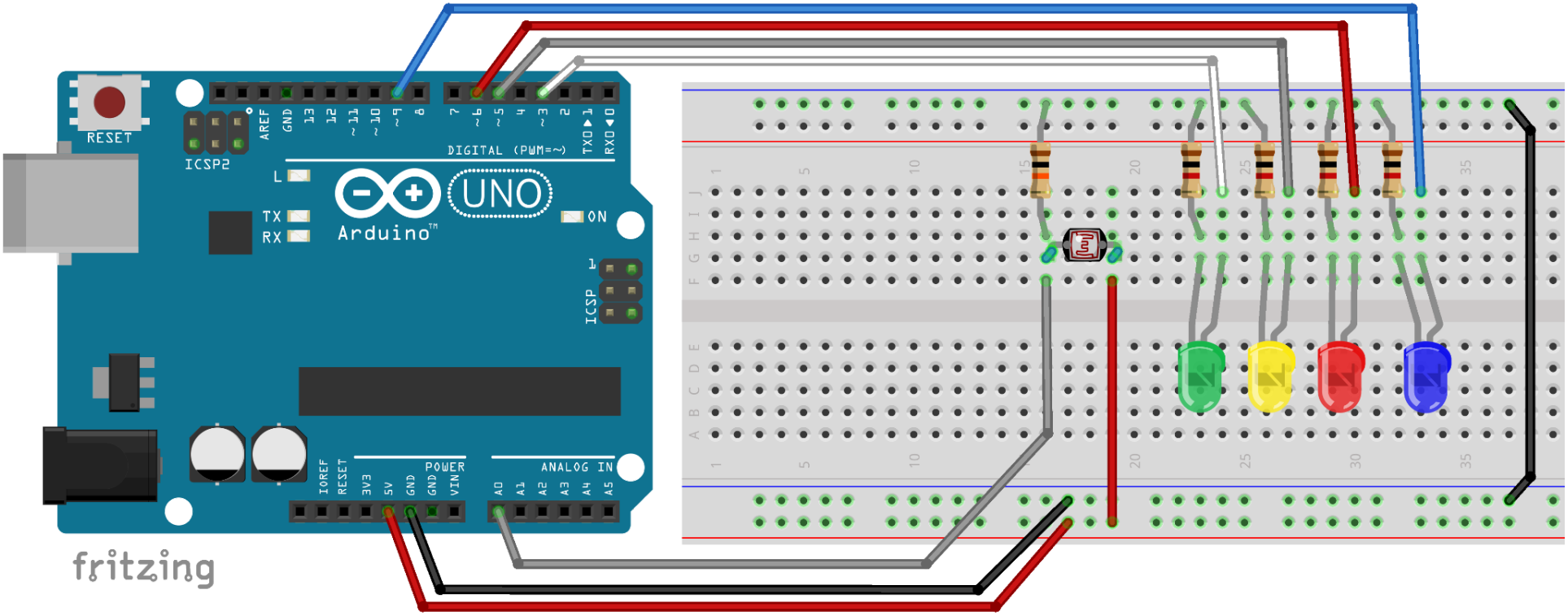
- Install more 1 kOhm resistors to GND
- Install the White and Blue wires to ~3 and ~9
- Install the Green and Blue LEDs, flat (cathode) to the resistor again



- Run **009_OOP_Four_Beacons**

Adding a Light Sensor…

- Install the 10 kOhm resistor to GND
- Install the Red wire
- Install the Light Sensor between resistor and red wire
- Install a White wire shown with Grey



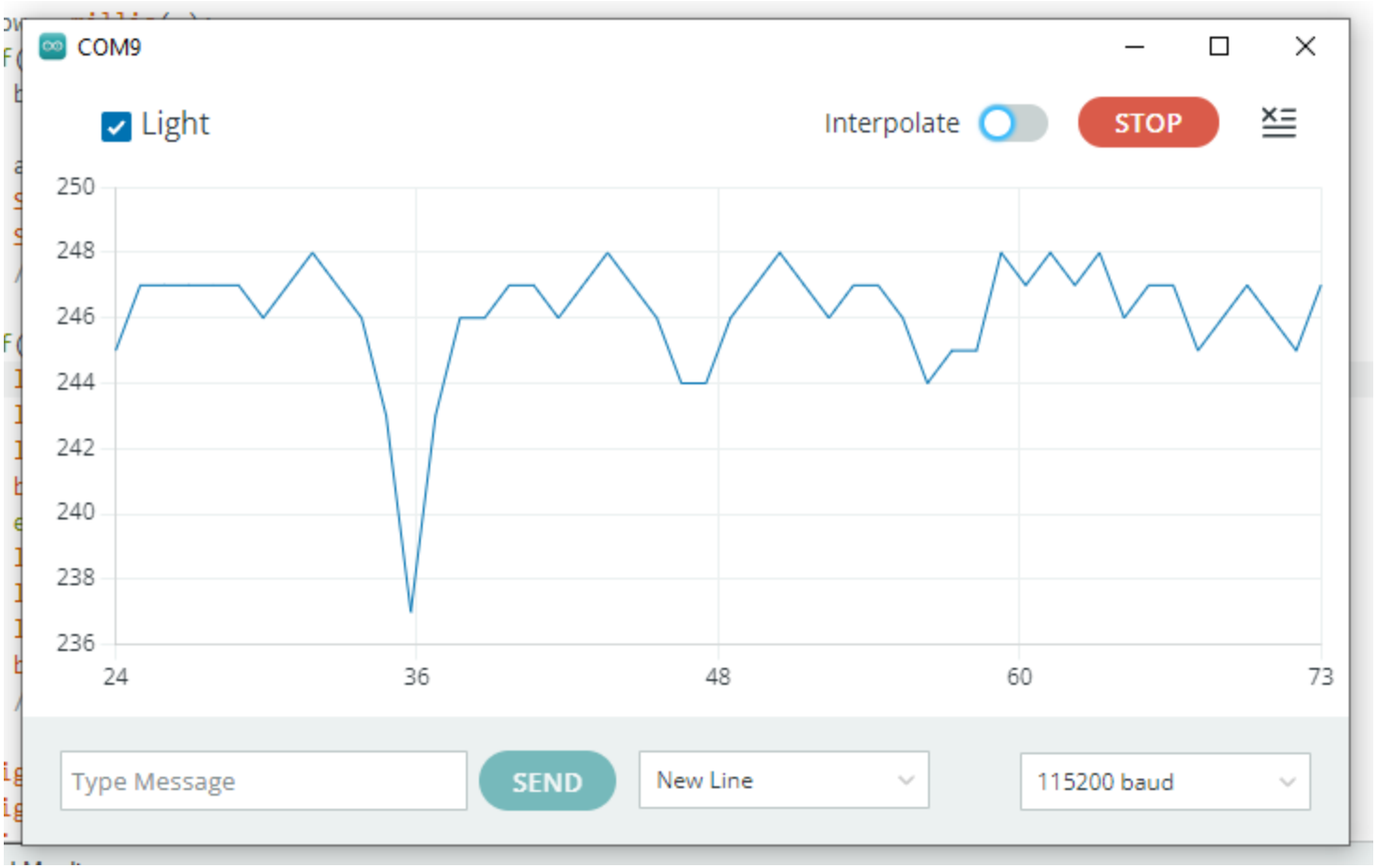
Light Sensor

- 10 bit Analog to Digital converter on ATMEGA328P → 0 - 1023

```
pinMode( ANAPIN, INPUT );

uint16_t u16AnalogValue = analogRead( ANAPIN );

uint16_t u16MappedValue = map( u16AnalogValue, fromLow, fromHigh, toLow,
toHigh );

if( u16MappedValue > SOMETHING ) { ; } else { ; }
```

- Run **011_Plot_Light**

Serial Plotter (Ctrl+Shift+L)



- Running **011_Plot_Light**
- Press Ctrl+Shift+L
- *Install short wire between A1 and GND to enjoy some chaos.*

Servos in general…

- A servo moves its “horn” to a position based on the width of a pulse on the control input.
- It will move straight to the position demanded.
- It might use plenty of current (Amps) getting there.
- 1 ms to 2.5 ms wide pulses define the max range, typically in a 50 Hz (20 ms period) signal, for a 0 to 180 degree range.
- Continuous running Servos uses the 1 to 2.5 ms range to run fastest one way, slow down and stop in the middle and run faster and faster in the opposite direction as the pulses increase in width.
- PWM capable pins on microcontrollers and inexpensive Arduinos allowed Model Railroads to use servos that used to be common in the more expensive Model Airplane and RC hobbies.

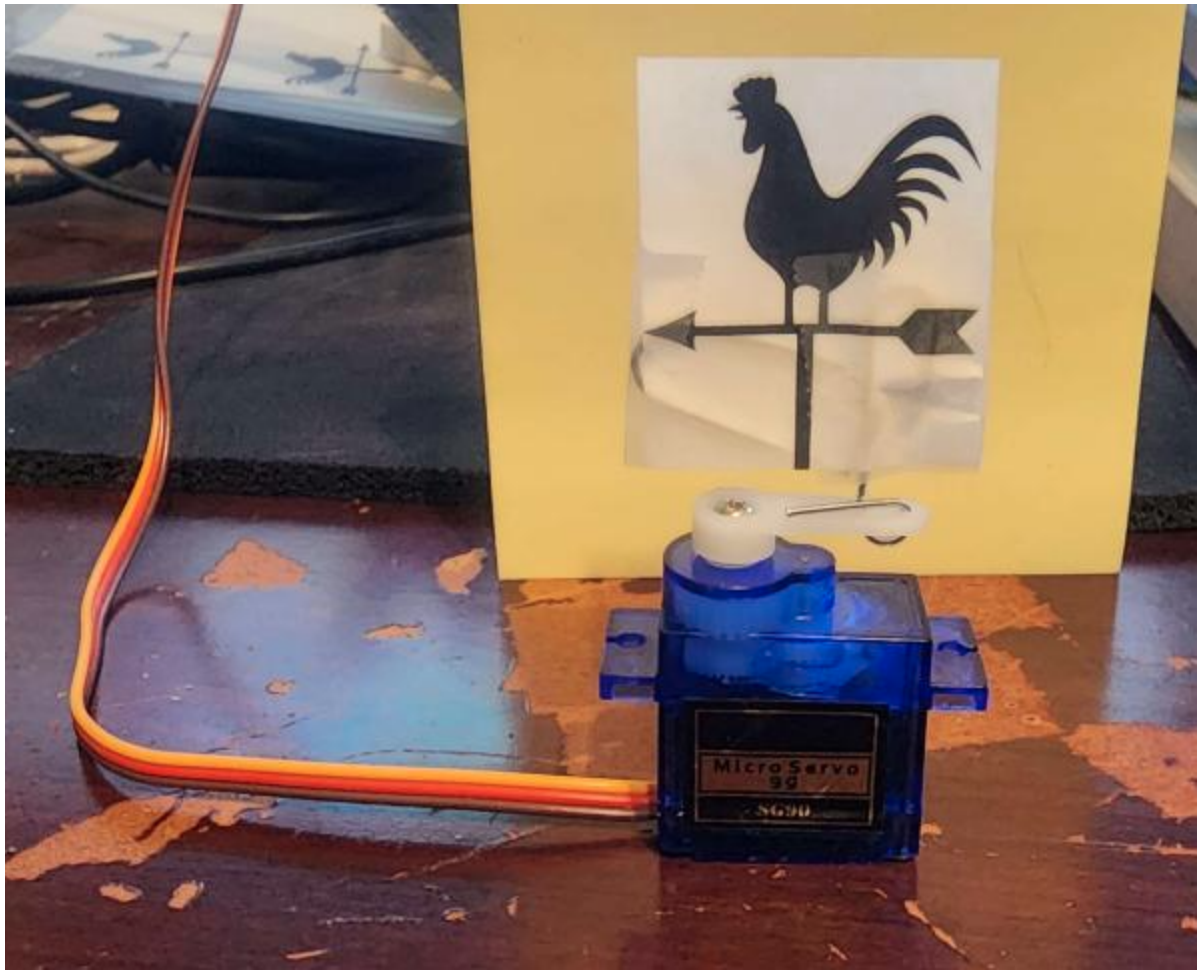
Servos.h

https://www.arduino.cc/reference/en/libraries/servo/

`#include <Servo.h>` gives us

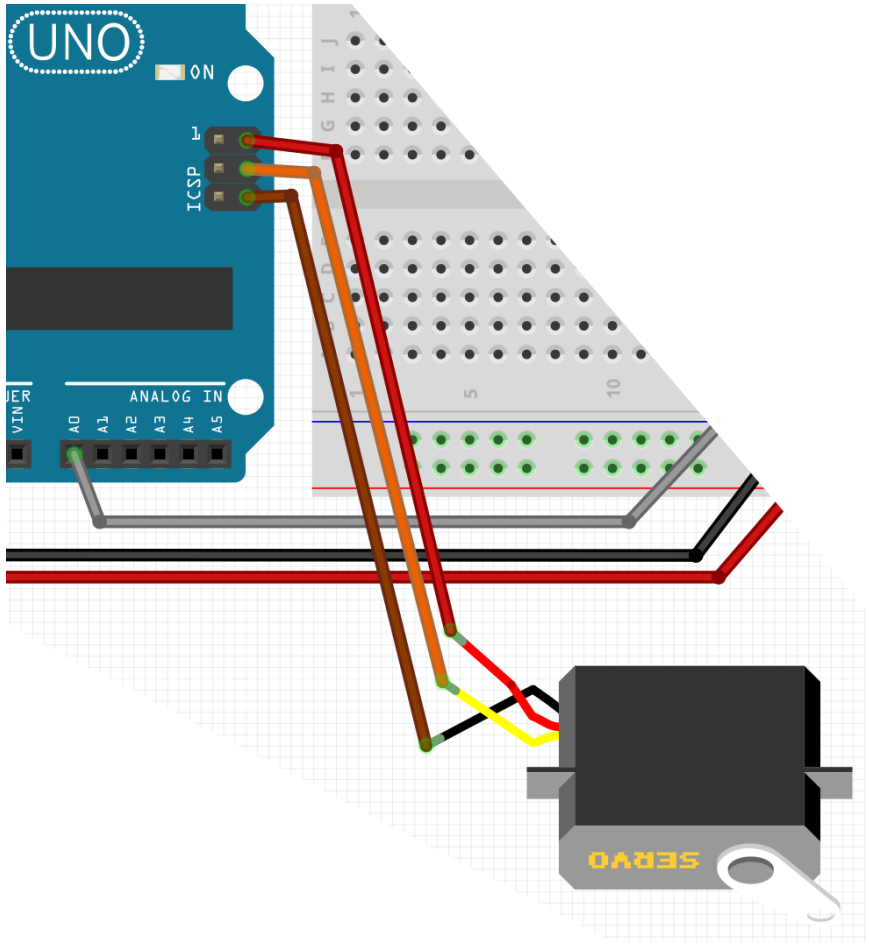
Methods

- attach()
- write()
- writeMicroseconds()
- read()
- attached()
- detach()

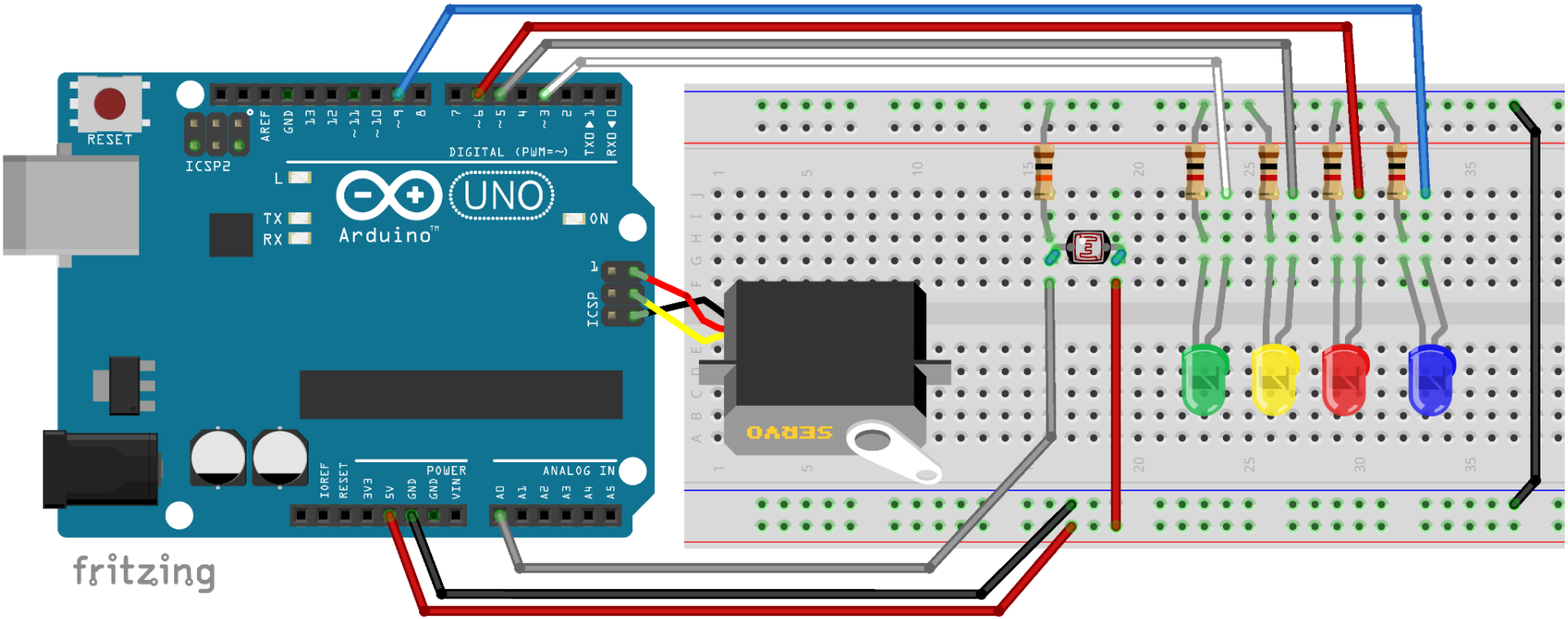


Connect our Servo…

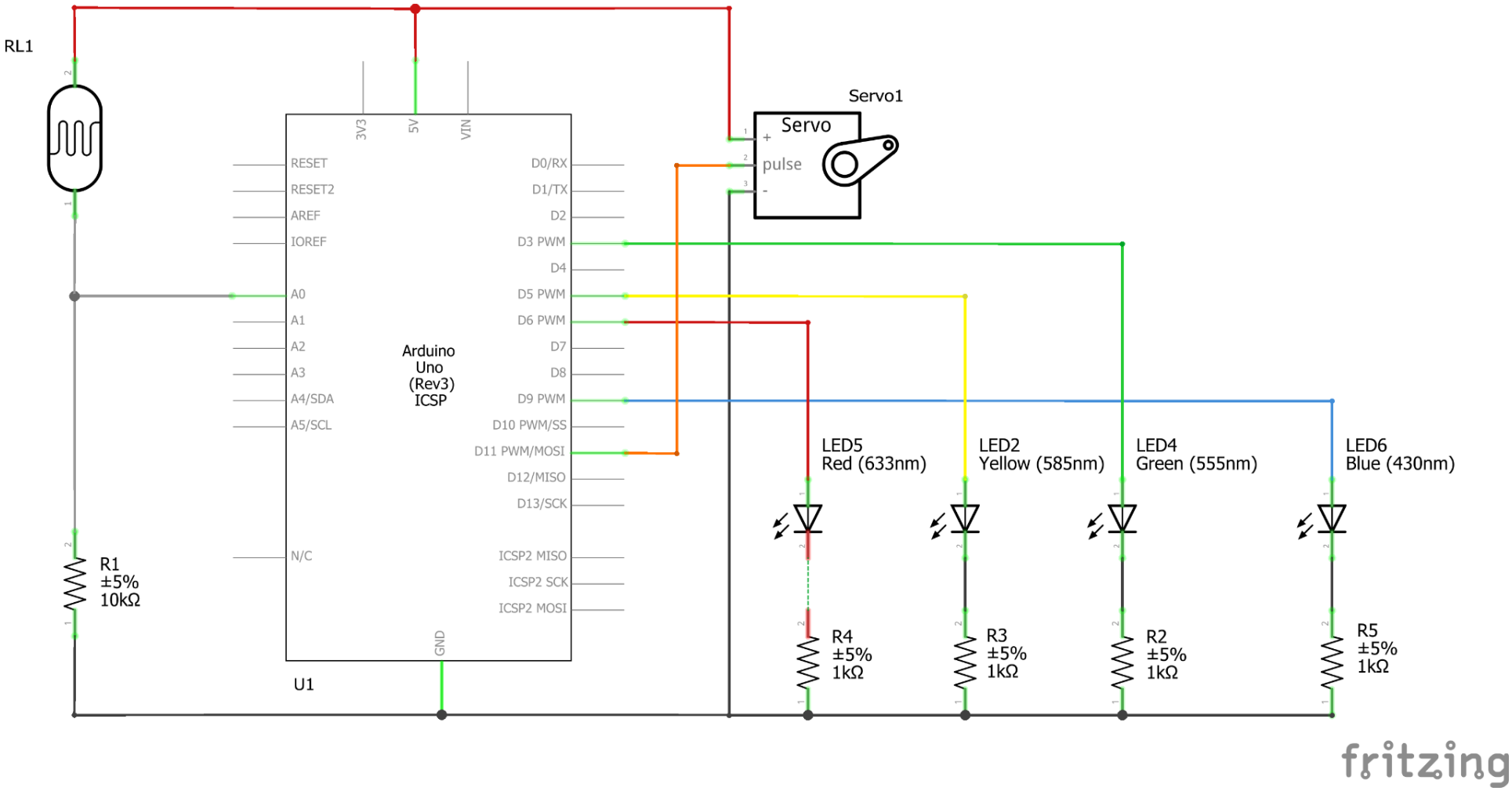
- Install male-female wires between the Uno's ICSP connector and the servo. Make sure the servo control wire (white, yellow or orange) goes to the middle ICSP pin towards the edge of the board. This is D11.



All Together….



Schematics (for those who care)



Servo in Code

```
analogValue = analogRead( ANAPIN );

// Limit the input
if( analogValue < MIN_ANA ) { analogValue = MIN_ANA; }
if( analogValue > MAX_ANA ) { analogValue = MAX_ANA; }

servoAngle = map( analogValue, MIN_ANA, MAX_ANA, MIN_SERVO, MAX_SERVO );

// Set the servo
myServo.write( servoAngle );
```

- Run **012_Servo_by_Light**

Notice the rough movements on the servo, since we do a `myServo.write` and the Servo jumps there right away

- Run **013_Servo_by_Light_OOP**

Notice the control towards the commanded position smooths out the motion, and all it takes is `myServo.update( );`

Power

- Vin: 6 to 12 Vdc, with enough current to drive servo(s)

- Computer's USB not enough when you turn 3 servos on at the same time

A Few Other Important Notes:

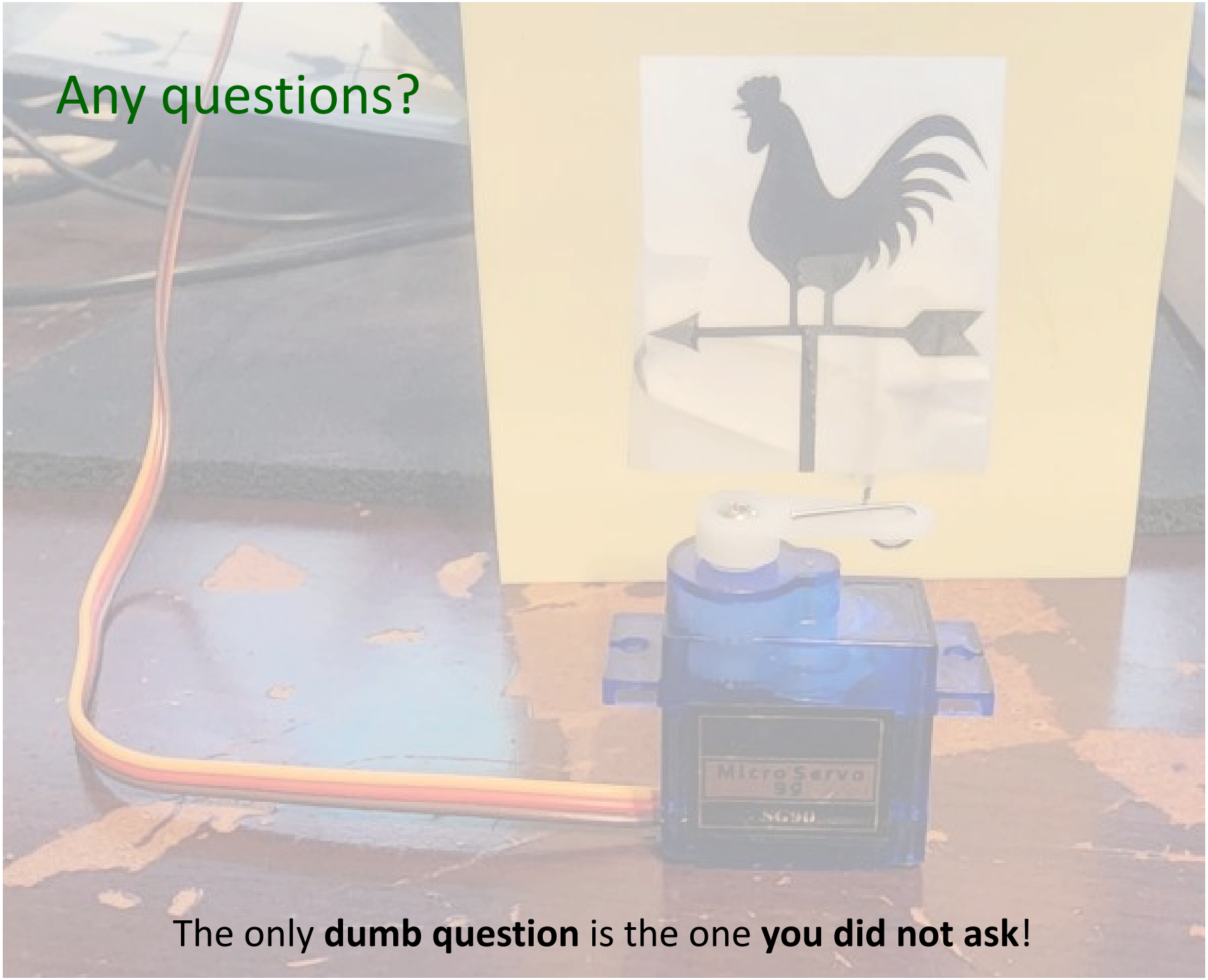
- Copying code from anywhere else, watch out for the quotes: “ and ” is not the same as "

- Keep the description and version in your VERSION_STR up to date, you need to be able to find the source, since the HEX file from the Arduino will not show you the C/C++ code again

- Use dates and English words as much as you can. Easier to find and search later

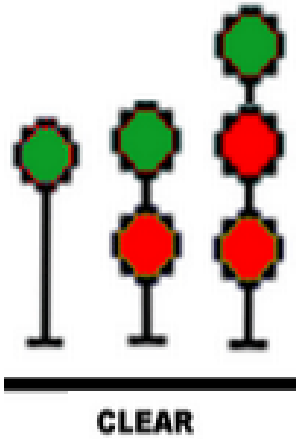
- More interesting reading
https://roboticsbackend.com/the-arduino-language-in-10-points/

Any questions?



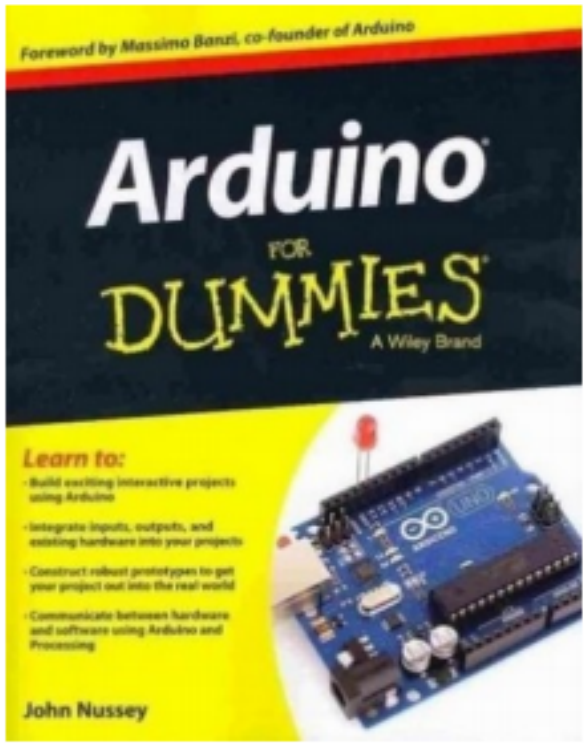
The only **dumb question** is the one **you did not ask**!

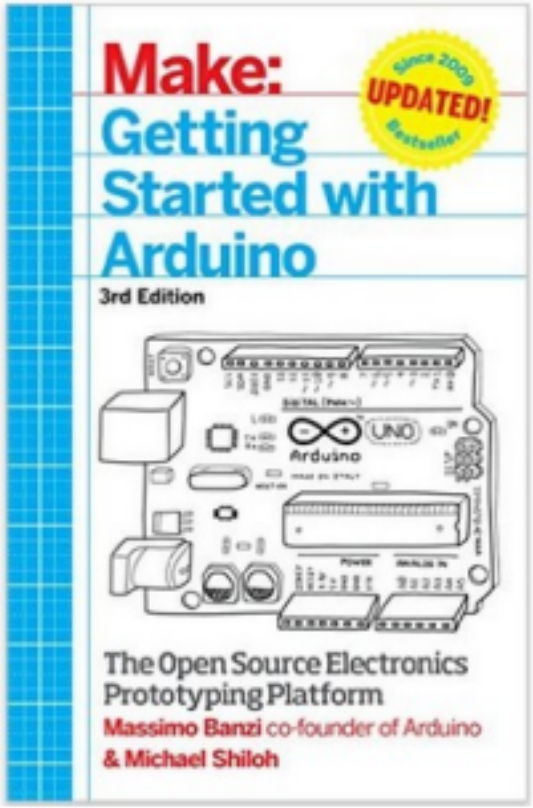
The End

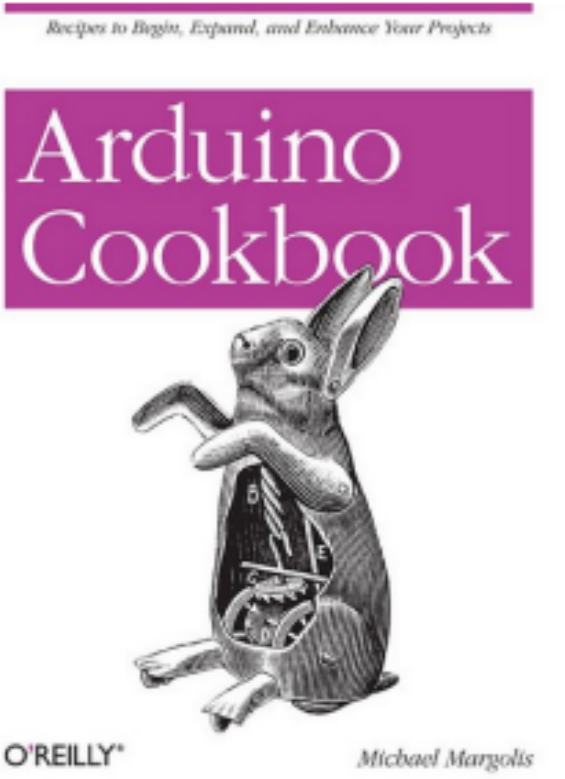


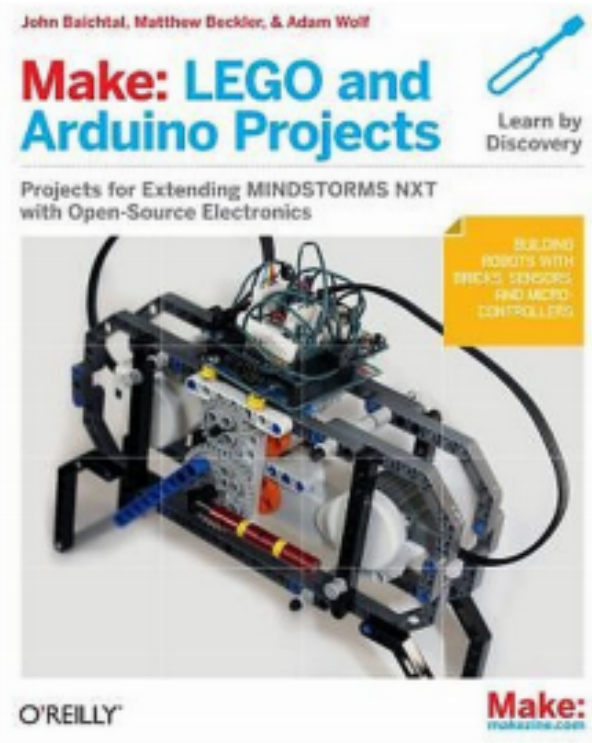
Thank you to Joel, David, Riley, *Bob, Donna, John, Stephanie, Alissa, Bianka, my Boss,*
and all the others that did not bother us while we were having fun doing this…

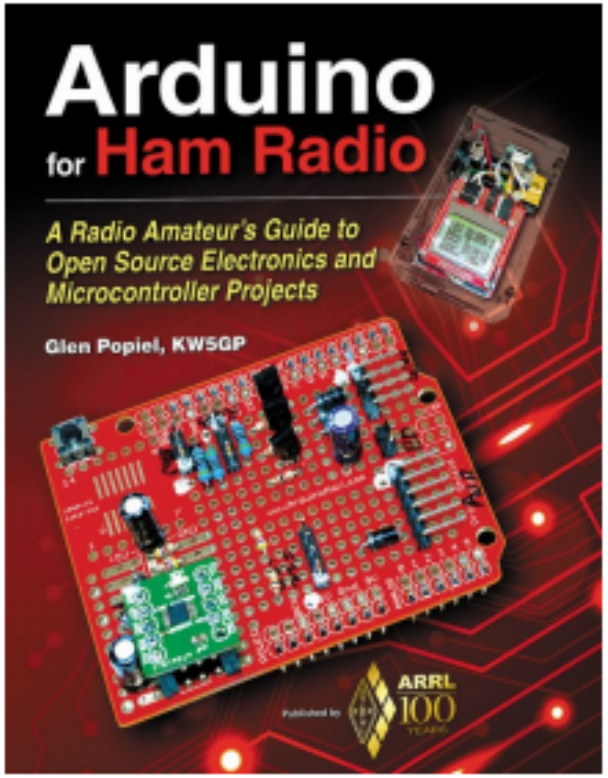
Get educated:






Recipes to Begin, Expand, and Enhance Your Projects
Arduino Cookbook
O'REILLY
Michael Margolis


John Baichtal, Matthew Beckler, & Adam Wolf
Make: LEGO and Arduino Projects
Learn by Discovery
Projects for Extending MINDSTORMS NXT with Open-Source Electronics
O'REILLY
Make:


Arduino for Ham Radio
A Radio Amateur's Guide to Open Source Electronics and Microcontroller Projects
Glen Popiel, KW5GP
ARRL
100

Changes:

v0.01, original