

# IoT Engineering

## 2: Microcontrollers, Sensors & Actuators

CC BY-SA, Thomas Amberg, FHNW  
(unless noted otherwise)

# Today

$\frac{1}{3}$  slides,

$\frac{2}{3}$  hands-on.

Slides, code & hands-on: [tmb.gr/iot-2](https://tmb.gr/iot-2)



# Prerequisites

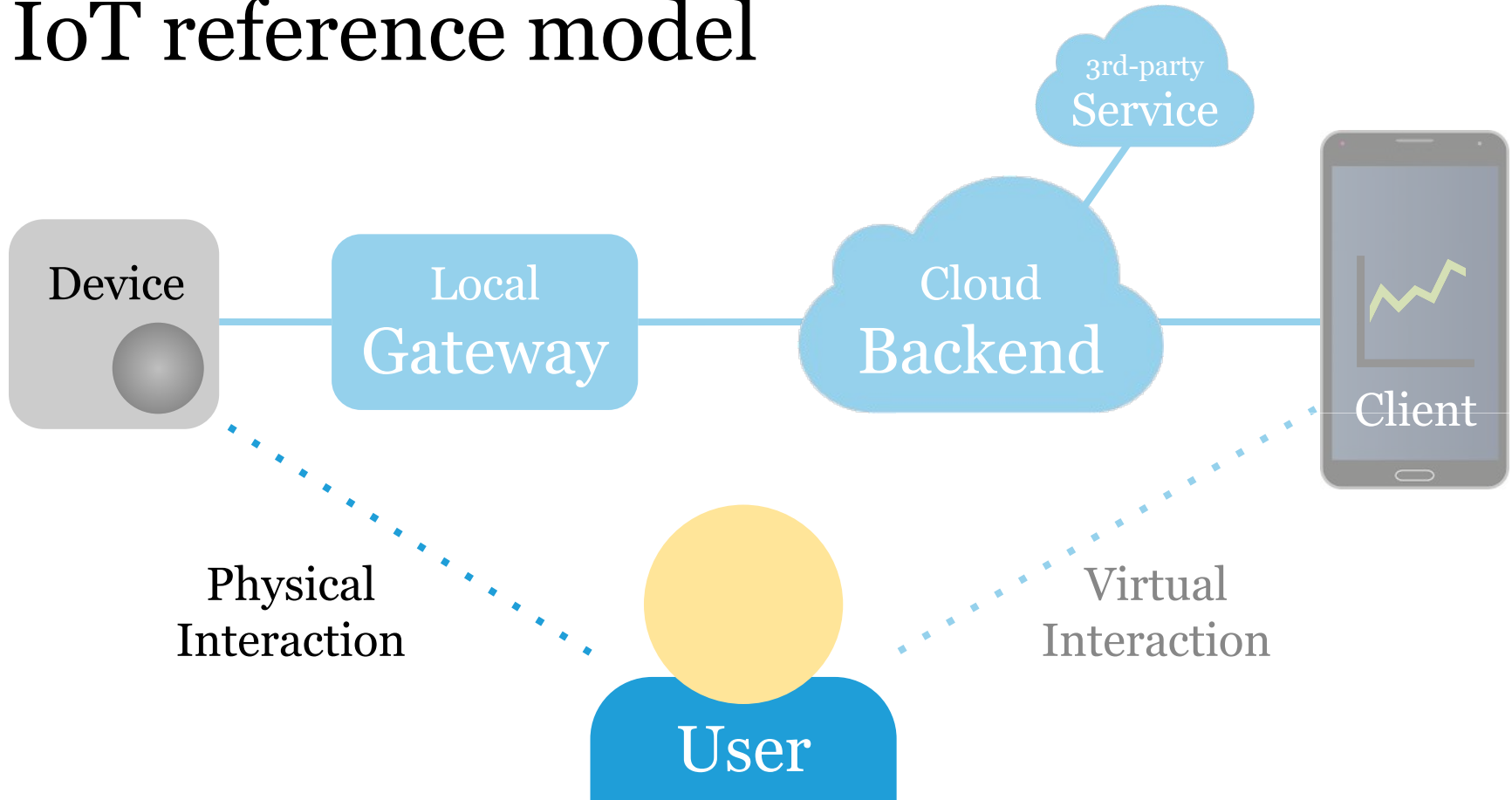
Install the Arduino IDE and set up microcontrollers:

Check the Wiki entry on [Installing the Arduino IDE](#).

[Set up the Feather nRF52840 Express](#) for Arduino.

[Set up the Feather Huzzah ESP8266](#) for Arduino.

# IoT reference model



# Physical computing

On device sensing/control, no connectivity.

Sensor  $\rightarrow$  Device, e.g. logging temperature.

Device  $\rightarrow$  Actuator, e.g. time-triggered buzzer.

Sensor  $\rightarrow$  Device  $\rightarrow$  Actuator, e.g. RFID door lock.

A  $\rightarrow$  B: measurement or control data flow.

# Microcontrollers

A *microcontroller* is a small, low power computer.

Sometimes it is also just called *controller* or *board*.

Runs a single program, there's no operating system.

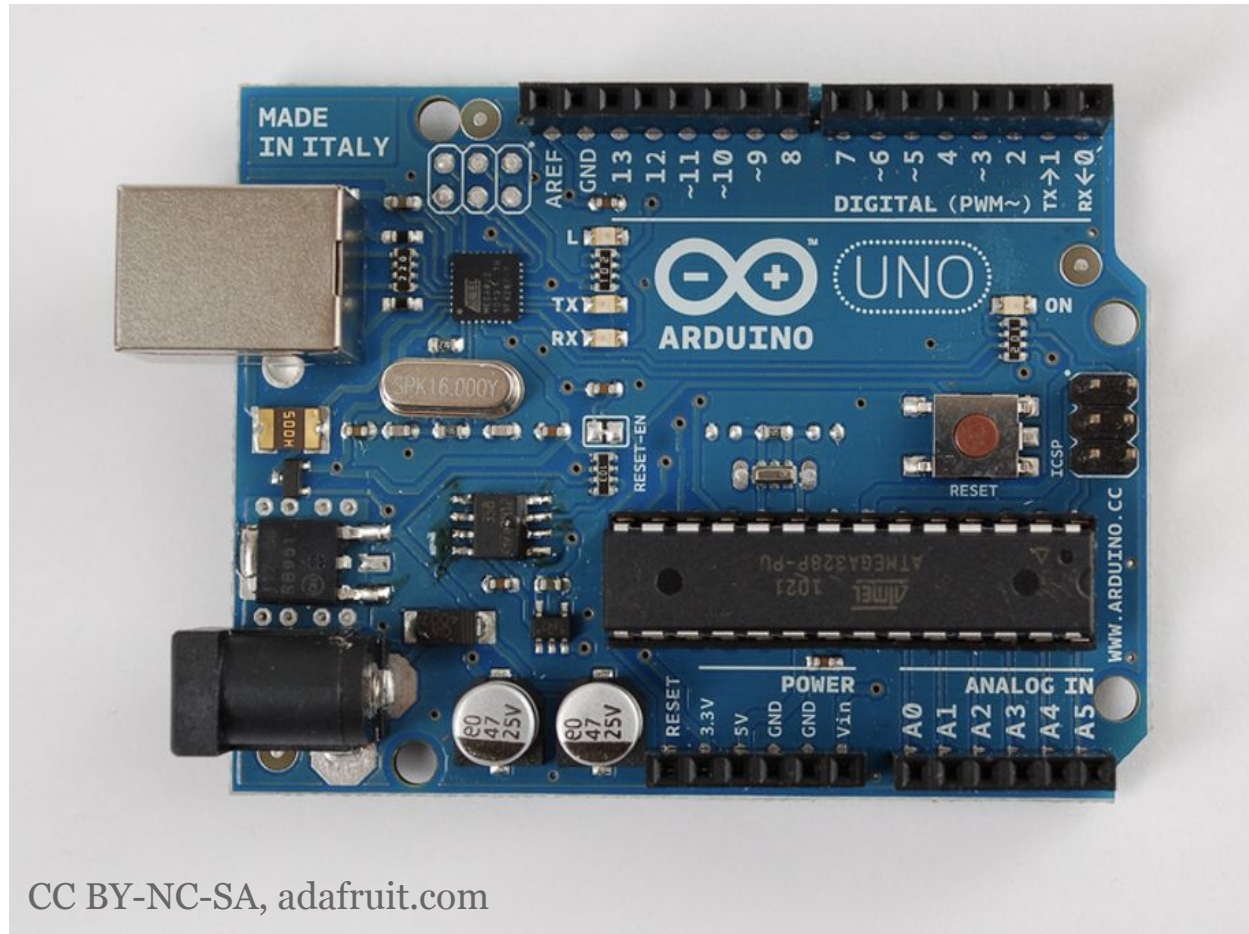
Pins for General Purpose Input and Output (GPIO).

We focus on Arduino compatible microcontrollers.

# Arduino

An electronics  
prototyping  
platform.

Here's a [video](#)  
about Arduino  
with Massimo  
Banzi.

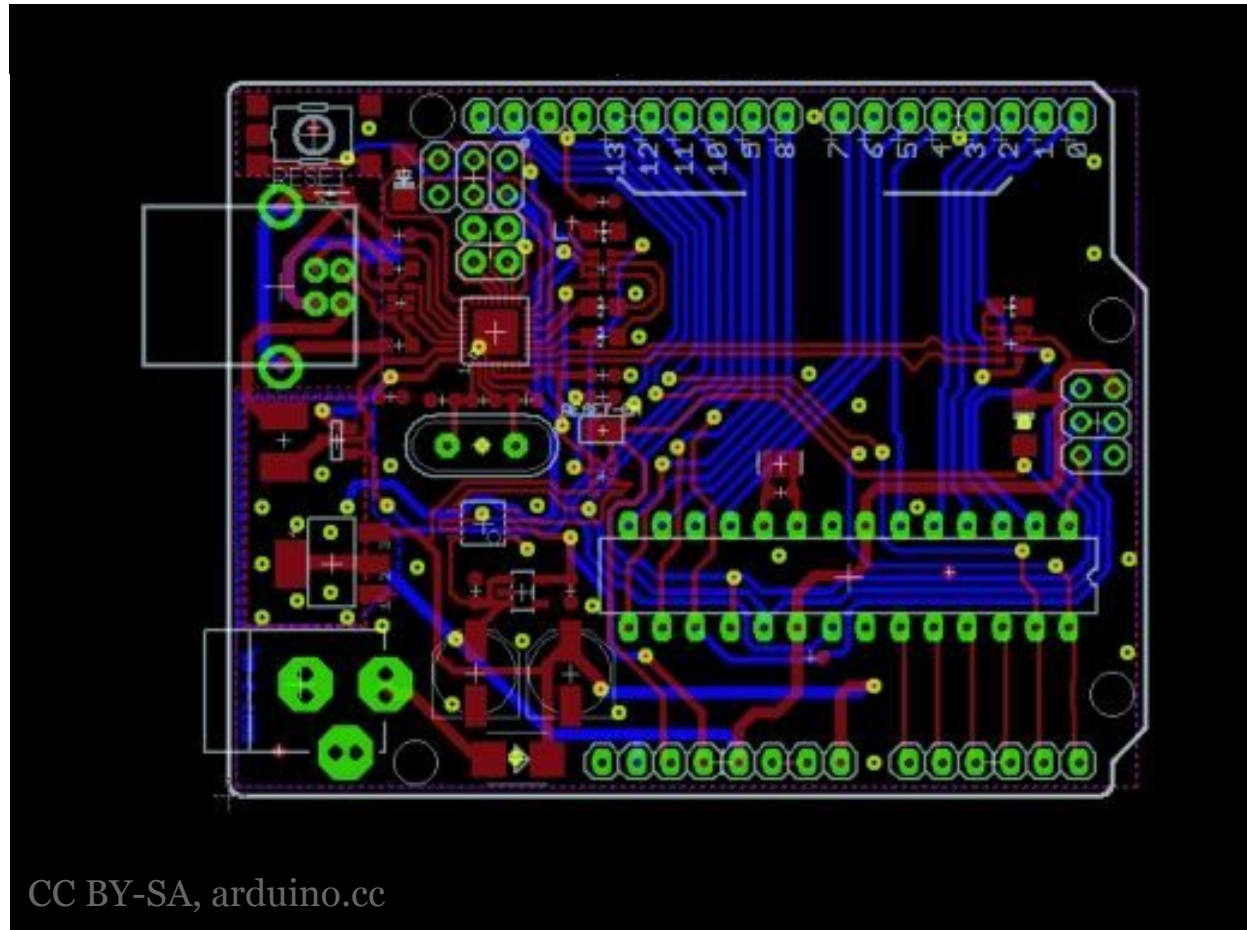


CC BY-NC-SA, adafruit.com

# Open source hardware

Layout and bill of materials are available under open licenses.

For details see, e.g. [OSHWA](#).





# Arduino compatible

Arduino *compatible* can mean different things:

Arduino *pin compatible*, for shield extensions.

Arduino IDE *programmable*, for ease of use.

We use Arduino IDE programmable controllers.

# Microcontroller form factors

Prototyping hardware *form factors* allow extensions:

Arduino ([Uno](#) and [MKR](#)) with "shield" extensions.

Adafruit [Feather](#) with FeatherWing extensions.

[Wemos](#), stackable modules based on ESP8266.

[M5Stack](#), a modular system based on ESP32.

We use Feather compatible microcontrollers.

# Feather form factor

Microcontroller form factor, [specified](#) by Adafruit.

[LiPo](#) charging circuit and USB on each board.

Reasonably small, breadboard friendly.

Broad range of microcontrollers.

[FeatherWing](#) extensions.

# Feather Huzzah ESP8266

Microcontroller with Wi-Fi, used by hobbyists.

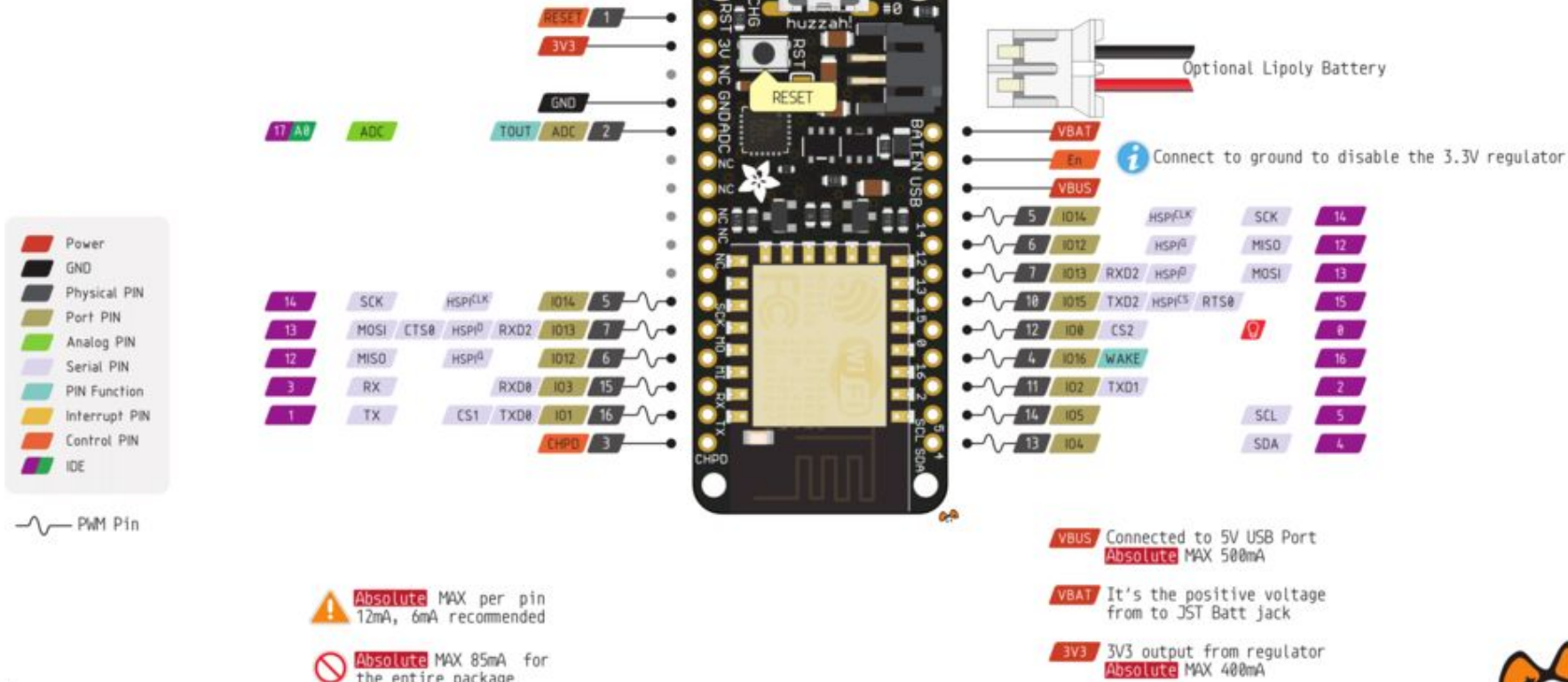
**ESP8266** System on Chip (SoC) by Espressif.

32-bit **Tensilica** CPU, 2.4 GHz 802.11 b/g/n.

4 MB **flash** memory, 80 kB user data RAM.

For details, check the [Wiki page](#).

# ESP8266



**Absolute** MAX per pin  
12mA, 6mA recommended

**Absolute** MAX 85mA for  
the entire package

# Feather nRF52840 Express

Microcontroller with [Bluetooth 5](#) (and more).

Nordic [nRF52840](#) System on Chip (SoC).

32-bit [ARM Cortex-M4](#) CPU with FPU.

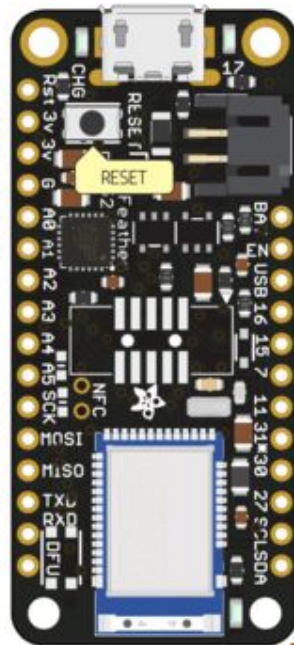
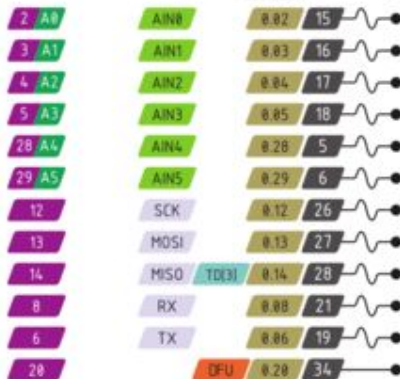
1 MB [flash](#) memory, 265 kB RAM.


For details, check the [Wiki page](#).

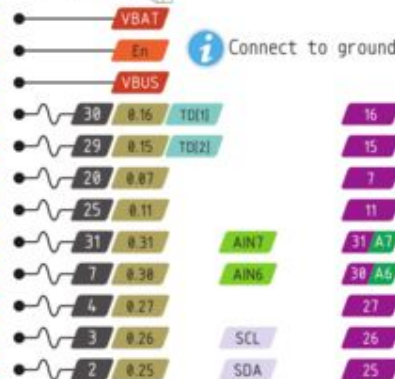
# nRF52840


 Any GPIO pin can be configured as a PWM output

 Any GPIO pin support interrupts



 Connect to ground to disable the 3.3V regulator



 Avoid using this pin directly  
Used to measure the LiPo battery level

 **Absolute** MAX per pin  
10mA, 5mA recommended

 **Absolute** MAX 30mA for  
the entire package

**VBUS** Connected to 5V USB Port  
**Absolute** MAX 500mA

**VBAT** It's the positive voltage  
from to JST Batt jack

**3V3** 3V3 output from regulator  
**Absolute** MAX 400mA

# Programming a microcontroller

Most microcontrollers are programmed via USB.

Some require a *programmer* hardware adapter\*.

(Cross-) *compiling* happens on your computer.

The binary has to be *uploaded* to the board.

Uploaded "firmware" runs stand-alone.

\*) We use hardware with USB, no programmer.



# Arduino IDE

The [Arduino IDE](#) is [open source](#) and written in Java.

This tutorial is based on the desktop version *1.8.x*.

[Board support URLs](#) enable 3rd-party boards.

For details, check the [Wiki page](#).

# Arduino "Hello, World!"

This is the basic structure of an Arduino program:

```
void setup() { // called once
    Serial.begin(115200); // set baud rate
}
void loop() { // called in a loop
    Serial.println("Hello, World!");
}
```

# Arduino settings

Connect your board via USB and make sure that

*Tools > Board* is set to your microcontroller,

*Tools > Port* matches the current USB port.

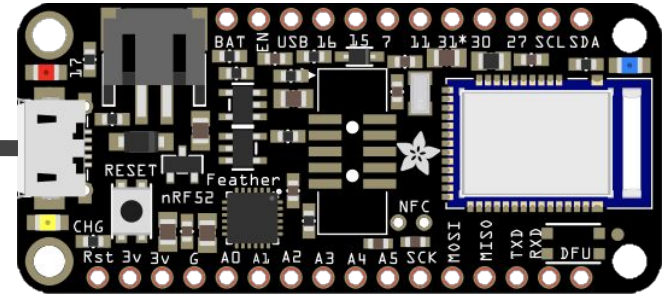
Some boards require additional settings.

# Arduino program upload

The *Upload* button compiles and uploads the code.



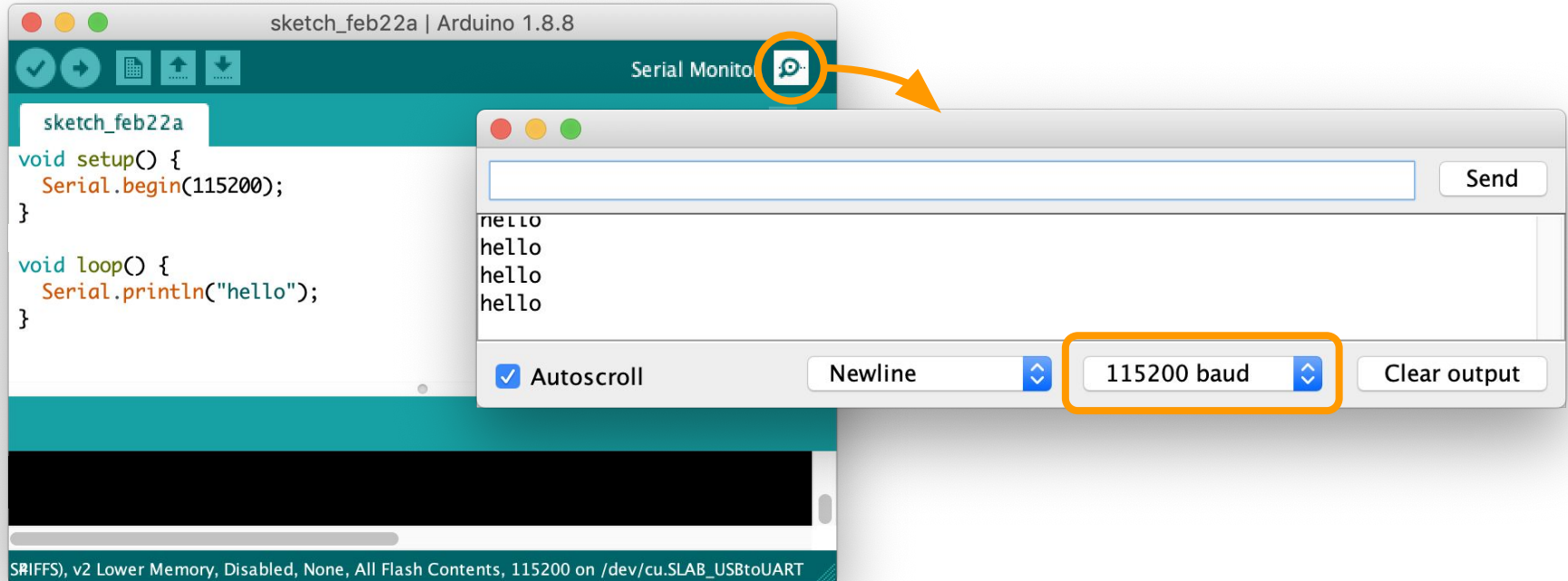
USB



CC BY-SA, [adafruit.com](https://adafruit.com), [fritzing.org](https://fritzing.org)

# Arduino serial console

Make sure the baud rate matches *Serial.begin()*.



# Arduino language

The [Arduino language](#) uses a subset of C/C++.

The user exposed code looks a bit like Java.

There is a [string](#) type and a [String](#) class.

[Libraries](#) are programmed in C++.

For details, check the [language reference](#).

# Input and output

Microcontrollers have an interface to the real world:

General purpose Input and Output (GPIO) pins.

GPIOs allow a controller to measure and control.

Measuring = *reading* sensor values from *input* pins.

Controlling = *writing* actuator values to *output* pins.

# Sensors and actuators

Convert physical properties to/from electrical signals.

Signals are *digital* (0 or 1) or *analog* (e.g. 0-255).

We look at two ways to wire sensors/actuators:

*Breadboard* and jumper wire connections.

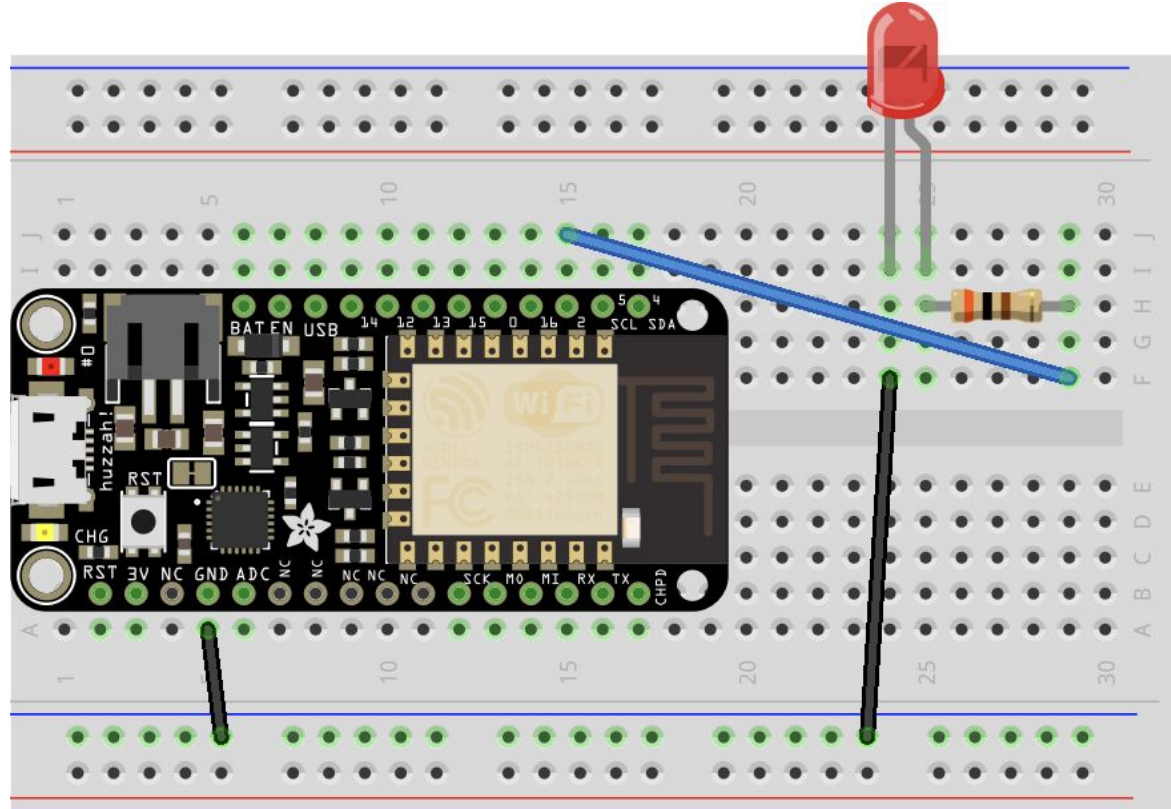
*Grove* connectors with 4-stranded wires.



# Breadboard

Wires electronic components, no soldering.

Under the hood, the columns are connected, and the power rails.

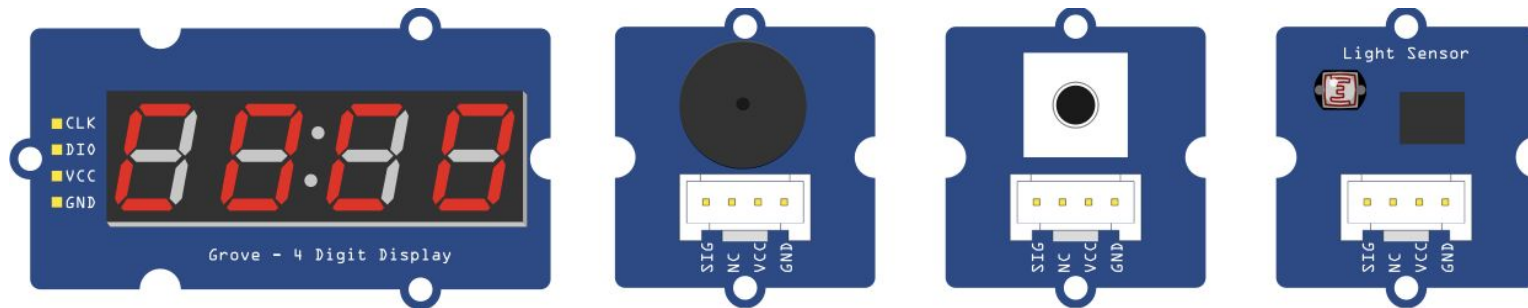


# Grove

**Grove** is a simple way to connect sensors/actuators.

This wiring standard is specified by Seeed Studio.

All modules designed by Seeed are **open source**.



# Blinking LED

The "Hello, World!" of embedded programming.

```
void setup() { // called once
    pinMode(2, OUTPUT); // configure pin 2
}
void loop() { // called in a loop
    digitalWrite(2, HIGH); // set pin 2 = on
    delay(500); // sleep 500 ms
    digitalWrite(2, LOW); // set pin 2 = off
    delay(500); // sleep 500 ms
}
```

# Arduino example code

Each Arduino library comes with example code.

There are also a number of basic examples.

See *Arduino IDE > File > Examples*

GPIO pin numbers may vary.

Use the [pin mapping](#).

# Hands-on, 5': LED (digital output)

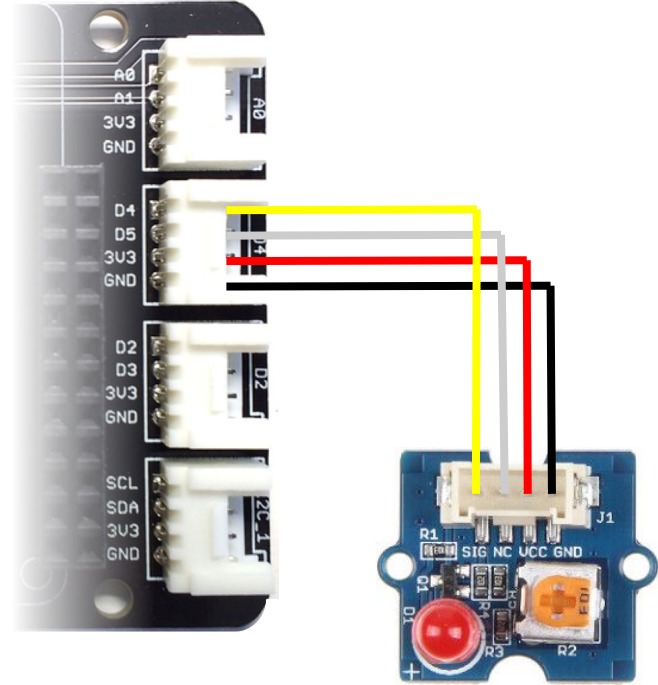
nRF52840 or ESP8266 w/ Grove:

Connect to adapter port *D4*.

Maps to ESP8266 pin 0.

Or nRF52840 pin 9.

Adapt [this code](#).



This is not the **RGB LED** (which requires a library).

# Hands-on, 5': Button (digital input)

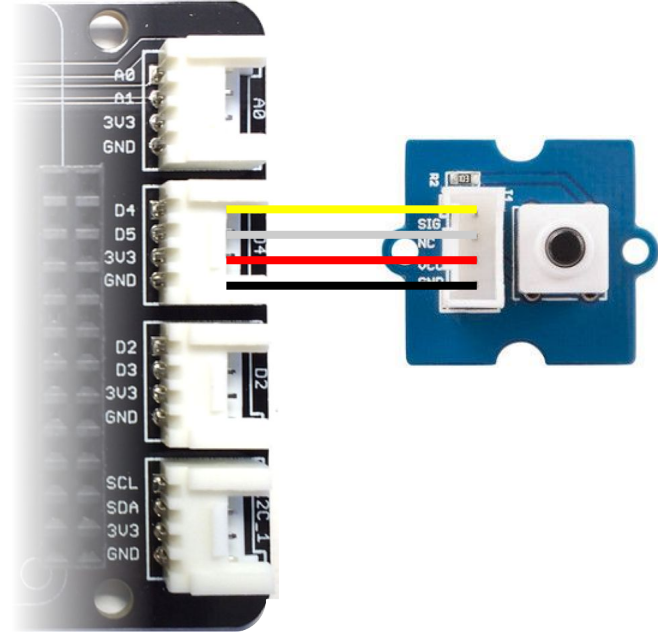
nRF52840 or ESP8266 w/ Grove:

Connect to adapter port *D4*.

Maps to ESP8266 pin 0.

Or nRF52840 pin 9.

Adapt [this code](#).



Check the pin number, use the serial console.

# Hands-on, 15': Button-triggered LED

This works with nRF52840 or ESP8266, w/ Grove.

Connect the LED to port  $D2^*$ , and the button to  $D4$ .

Combine the previous examples to switch the LED.

Look up the [pin mapping](#) to adapt the pin numbers.

\*) On the ESP8266, remove LED for programming.

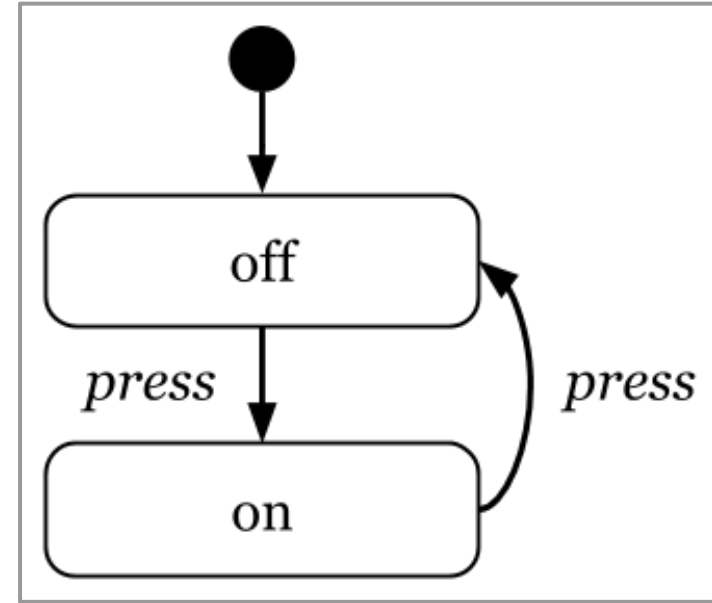
Commit the resulting code to the hands-on repo.

# State machine

A (finite-) **state machine** is a simple way to manage state in embedded programs.

System is in one state at a time,  
*events* trigger state *transitions*.

E.g. 1<sup>st</sup> button *press* => light *on*,  
2<sup>nd</sup> button *press* => light *off*,  
3<sup>rd</sup> => *on*, 4<sup>th</sup> => *off*, etc.



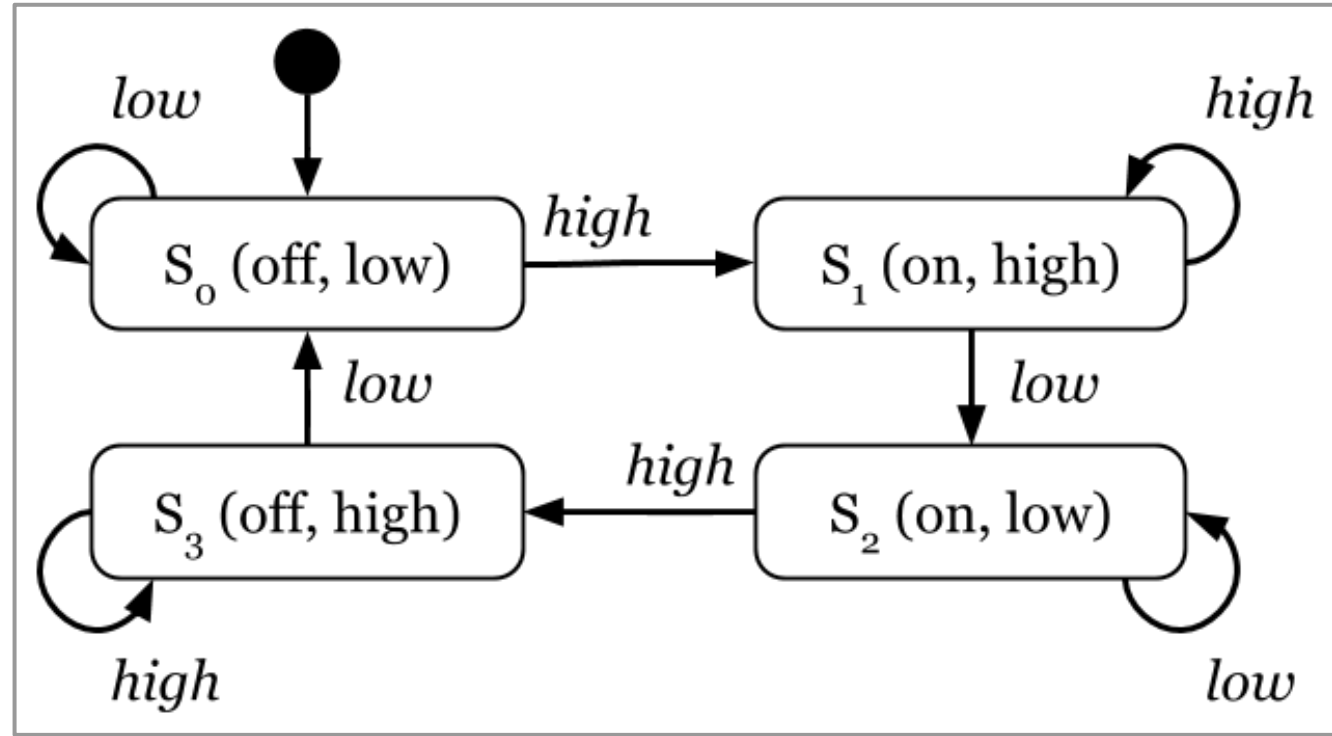
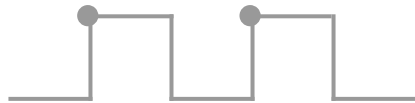


# State machine (detail)

Button is  
*high* or *low*.

Light is  
*on* or *off*.

Pressed =  
*low*  $\rightarrow$  *high*.



# State machine (code)

```
int b = digitalRead(buttonPin); // local
if (s == 0 && b == HIGH) { // s is state
    s = 1; digitalWrite(ledPin, HIGH); // on
} else if (s == 1 && b == LOW) {
    s = 2;
} else if (s == 2 && b == HIGH) {
    s = 3; digitalWrite(ledPin, LOW); // off
} else if (s == 3 && b == LOW) {
    s = 0; // note: actions are idempotent
} // not shown: global int s = 0; ...
```

# Hands-on, 5': State machine

Copy and complete the code of the state machine.

Make sure it works, with a button and LED setup.

Change it to switch off only, if the 2<sup>nd</sup> press is *long*.

Let's define long as  $> 1s$ , measure time with `millis()`.

Commit the resulting code to the hands-on repo.

# Hands-on, 5': Light sensor (analog input)

nRF52840 or ESP8266 w/ Grove:

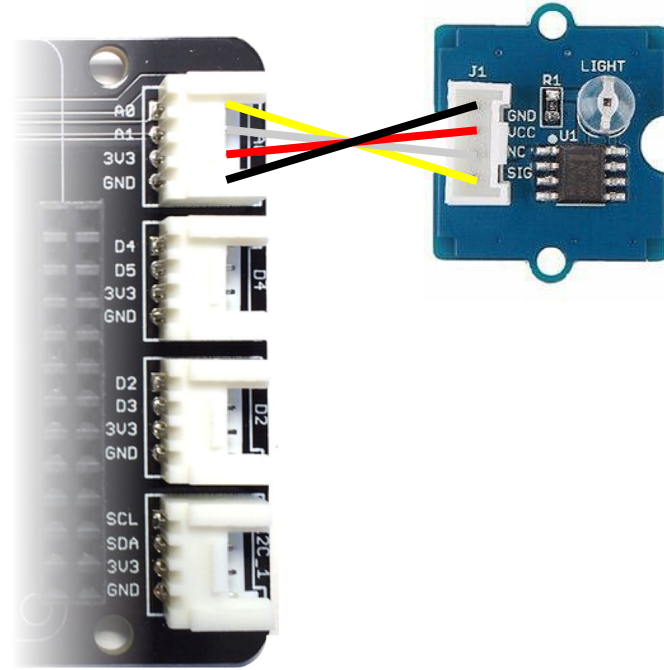
Connect to adapter port A0.

Maps to ESP8266 pin *ADC*.

Or nRF52840 pin *A0*.

Try [this code](#).

Use the serial console, or [serial plotter](#).



# Map input to value range

Sometimes mapping sensor value ranges helps, e.g.

0 - 1024 analog input => 0 - 10 brightness levels.

Arduino has a simple `map()` function for this:

```
int map(value, // measured input value  
        fromLow, fromHigh, // from range  
        toLow, toHigh); // to range
```

```
int x = ...; x = map(x, 0, 1024, 0, 10);
```

# Hands-on: Temperature (DHT11)

*DHT11* sensors require a [library](#).

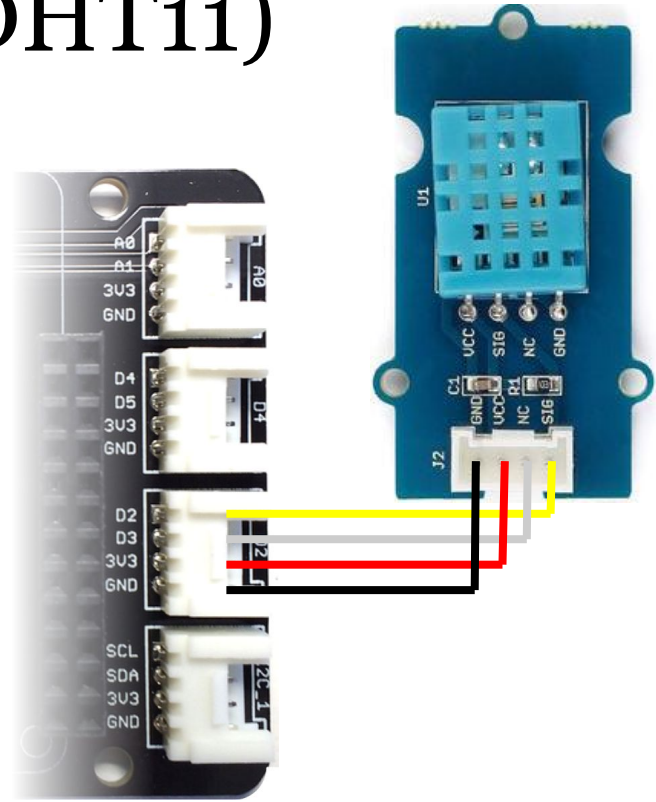
Connect to adapter port *D2*.

[Maps to](#) ESP8266 pin 2.

Or nRF52840 pin 5.

Adapt [this code](#).

New to libraries? See [Arduino Wiki page](#).



# Hands-on, 30': Kitchen timer

Design a kitchen timer to the following specification:

Displays a countdown to 0, in minutes and seconds.

Let's the user reset to 00:00, enter a new timespan.

Allows the user to start the countdown at *mm:ss*.

Starts buzzing if the countdown reaches *00:00*.

Use a state machine, get the time with `millis()`.

# Summary

We programmed a microcontroller in (Arduino) C.

We used digital and analog sensors and actuators.

We learned to design and code a state machine.

These are the basics of physical computing.

Next: Sending Sensor Data to IoT Platforms.



# Homework, max. 3h

Implement or finish the kitchen timer you designed.

Document the timer state machine (PDF or PNG).

Commit the code and docs to the hands-on repo.

Bring the (working) timer to the next lesson.

Consider cooking something to test it.

# Feedback?

Find me on <https://fhnw-iot.slack.com/>

Or email [thomas.amberg@fhnw.ch](mailto:thomas.amberg@fhnw.ch)

Slides, code & hands-on: [tmb.gr/iot-2](http://tmb.gr/iot-2)

