

IoT Data Collection (idb) Microcontrollers, Sensors & Actuators

CC BY-SA, T. Amberg & J. Luthiger, FHNW
(Unless noted otherwise)
Slides: tmb.gr/idb-mcu

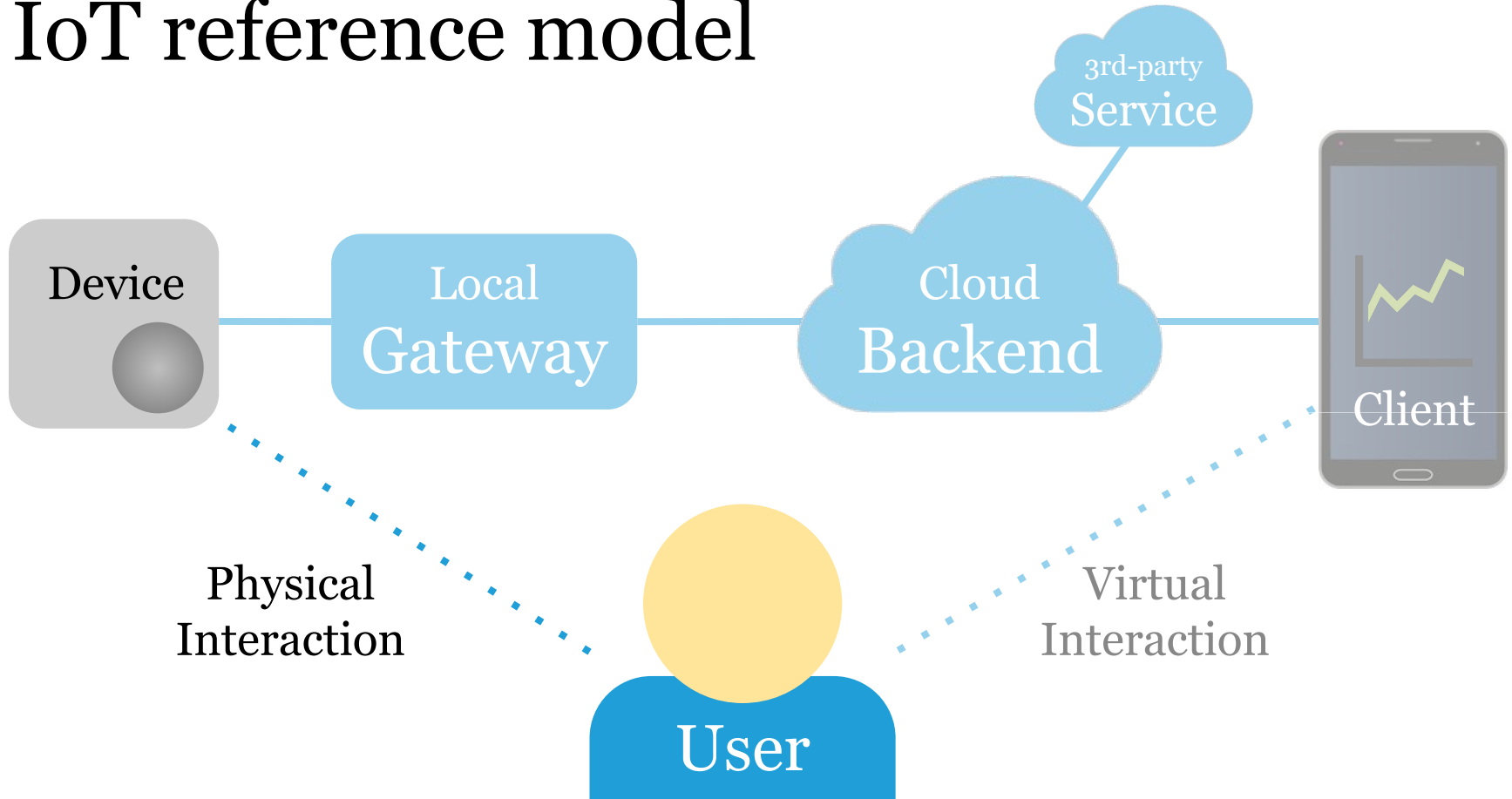
Prerequisites

Install the **Arduino IDE** and set up microcontrollers:

Check the Wiki entry on **Installing the Arduino IDE**.

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.

Single-Board Computer or Microcontroller

Single-Board Computer is a complete computer built on a single circuit board, with microprocessor(s), memory, input/output (I/O) and other features like operating system required of a functional computer. Example:

- [Raspberry Pi](#) developed by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries.

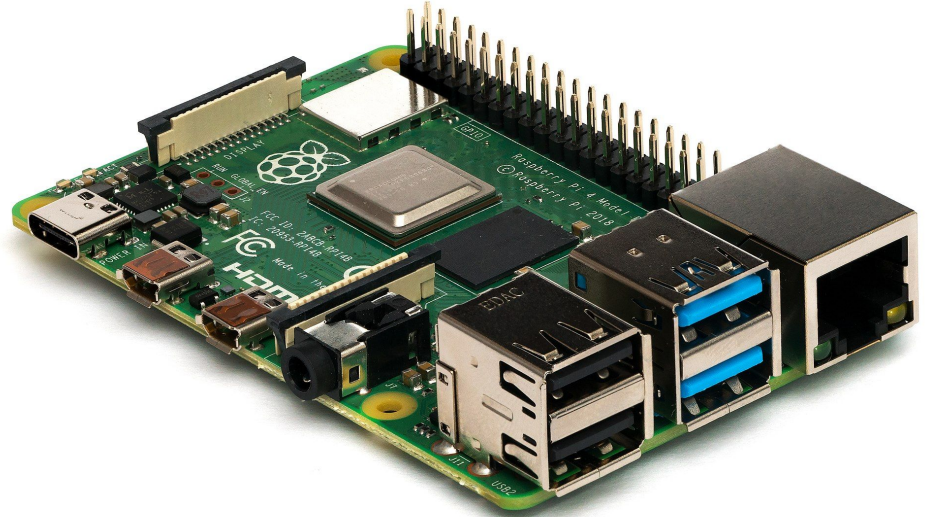
Microcontroller is a small, low power computer. Sometimes it is also just called controller or board. Runs a single program, **there's no operating system**. Pins for General Purpose Input and Output (GPIO). Example:

- [Arduino](#) is an open-source hardware and software company, that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices

Raspberry Pi

A single-board computer in different flavors, from Raspberry Pi Zero to Raspberry Pi 4

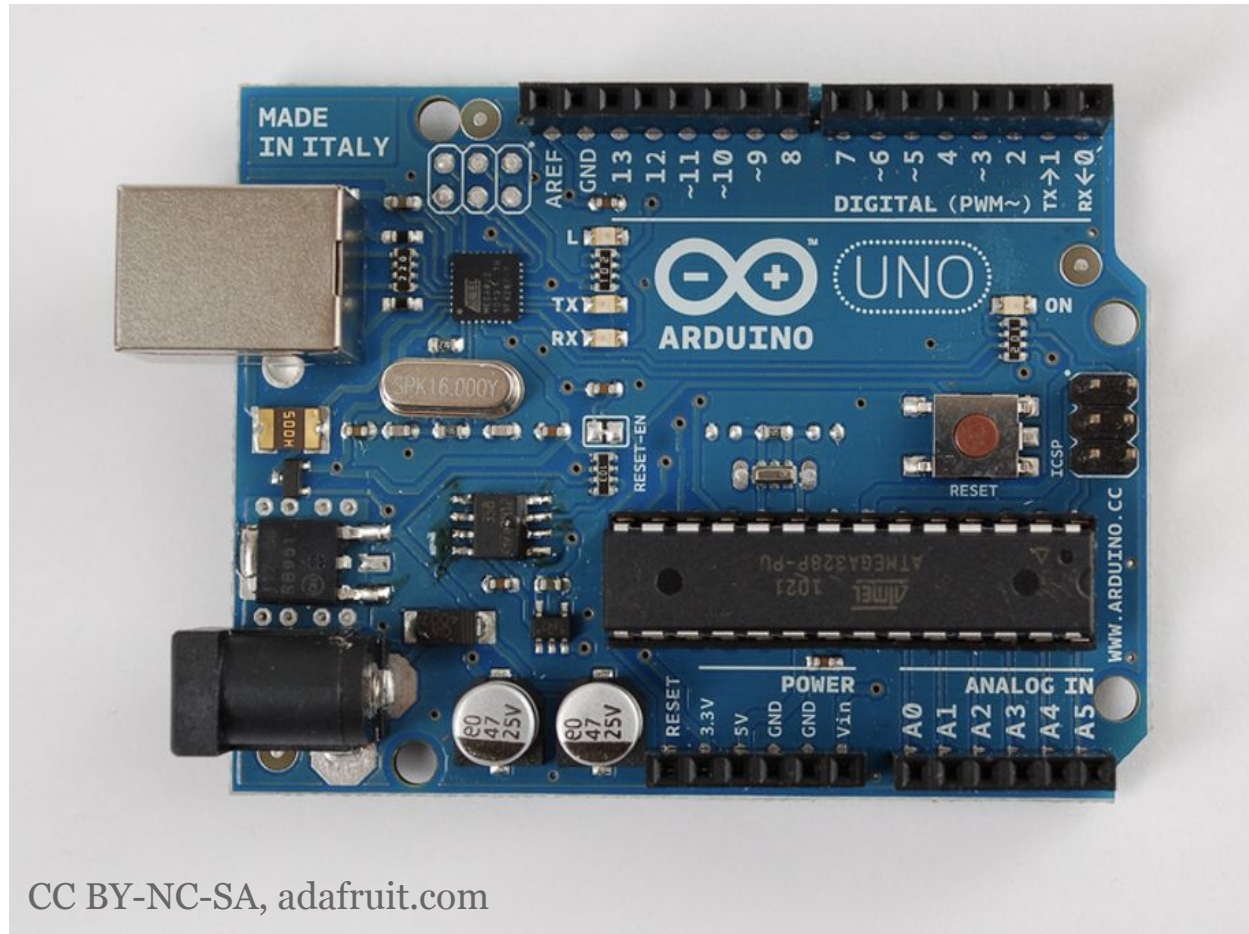
See more on this [video](#)



Arduino

An electronics
prototyping
platform.

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



CC BY-NC-SA, adafruit.com

Arduino compatible

Arduino *compatible* can mean different things:

Arduino *pin compatible*, for shield extensions.

Arduino IDE *programmable*, for ease of use.

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.

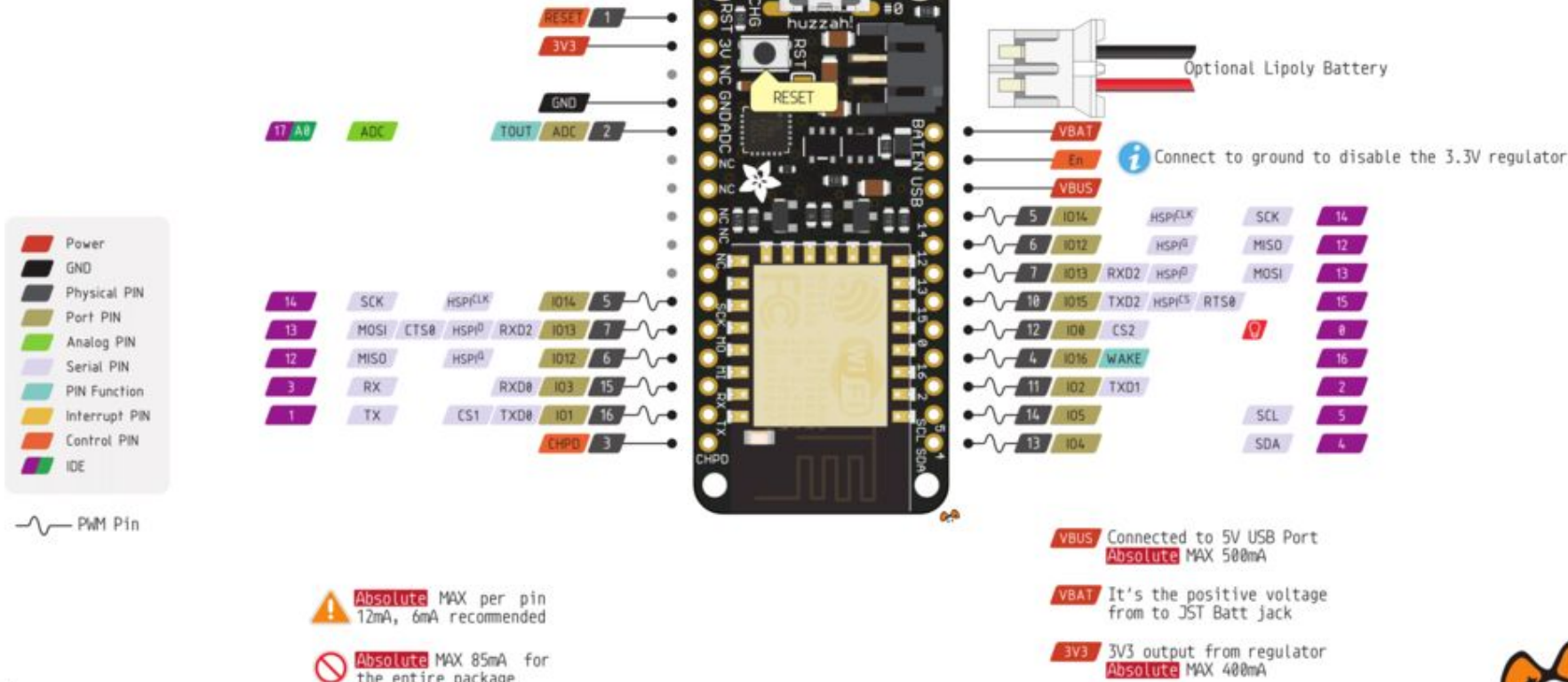
ESP8266 System on Chip (SoC) by Espressif.

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).

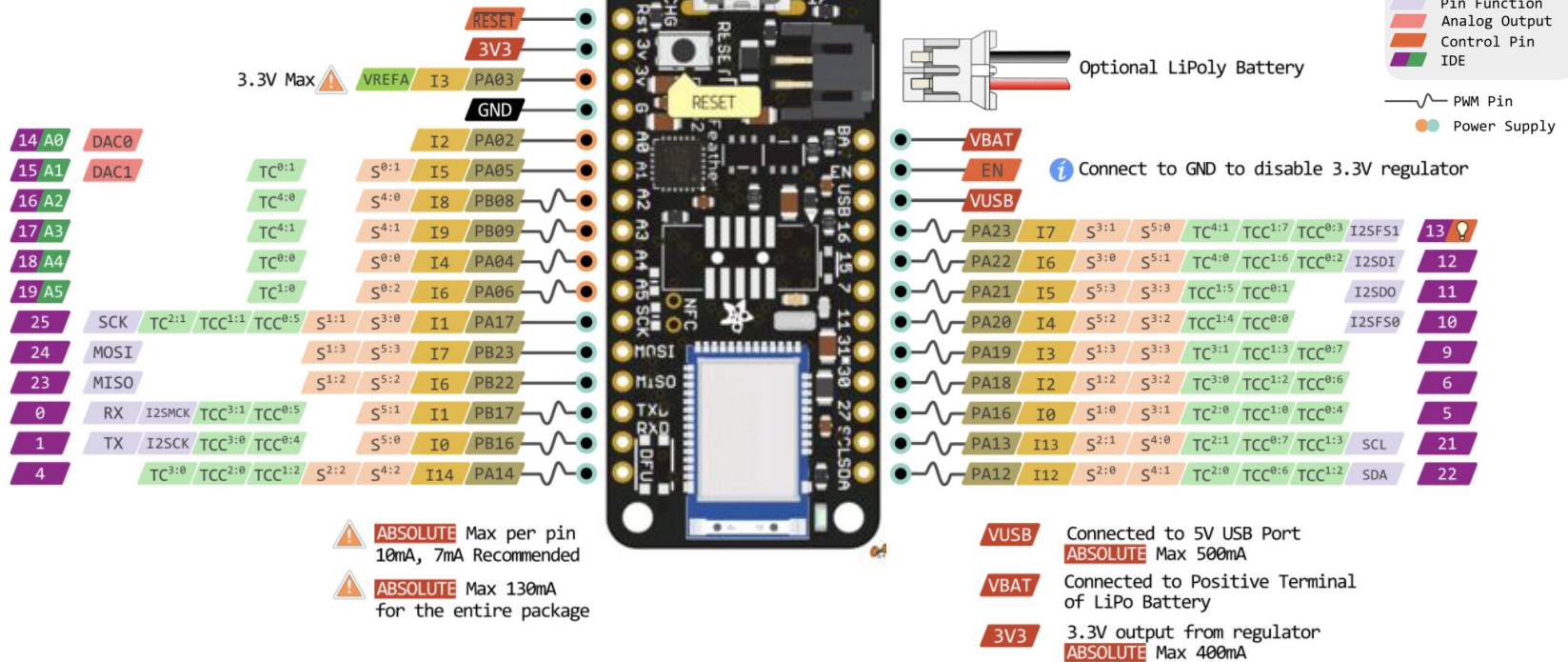
Nordic [nRF52840](#) System on Chip (SoC).

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

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

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

nRF52840



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.

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:

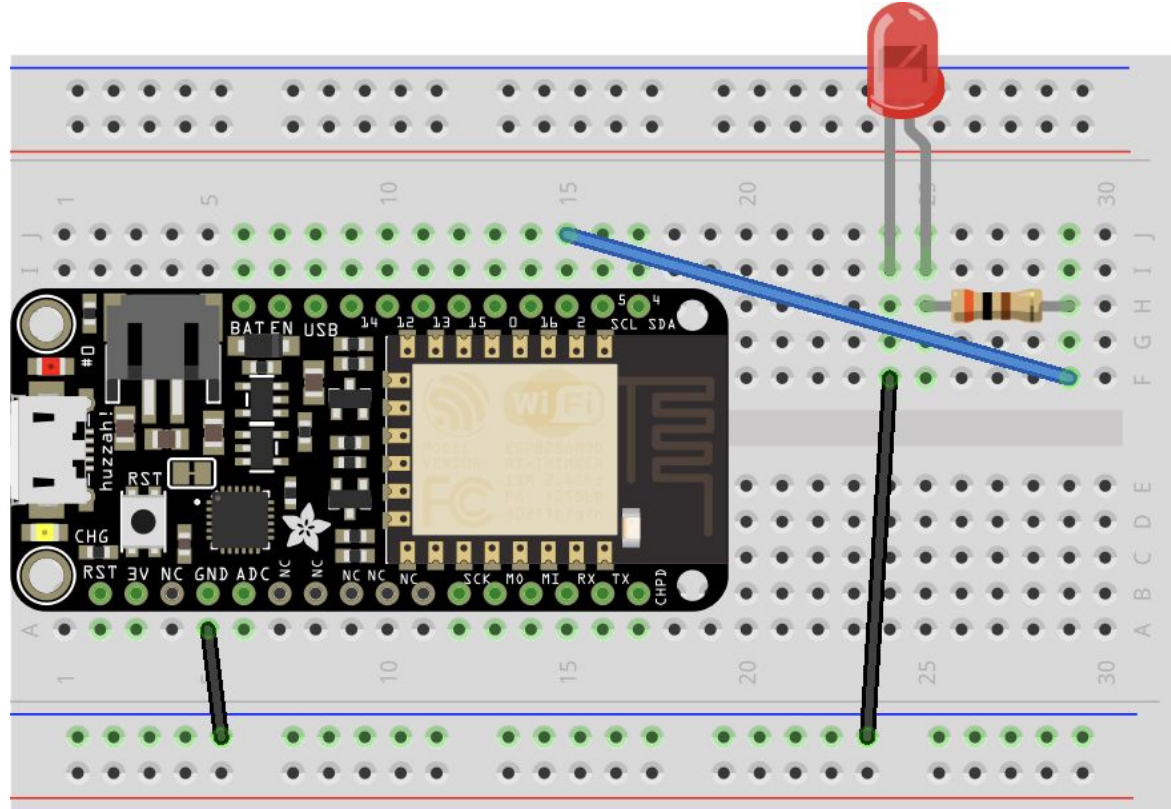
Breadboard and jumper wire connections.

Grove connectors with 4-stranded wires.

Breadboard

Wires electronic components, no soldering.

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

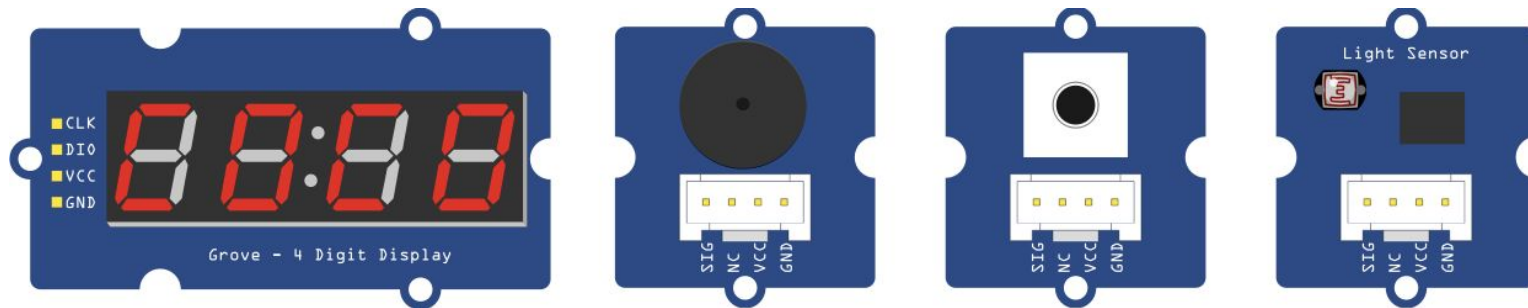


Grove

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:

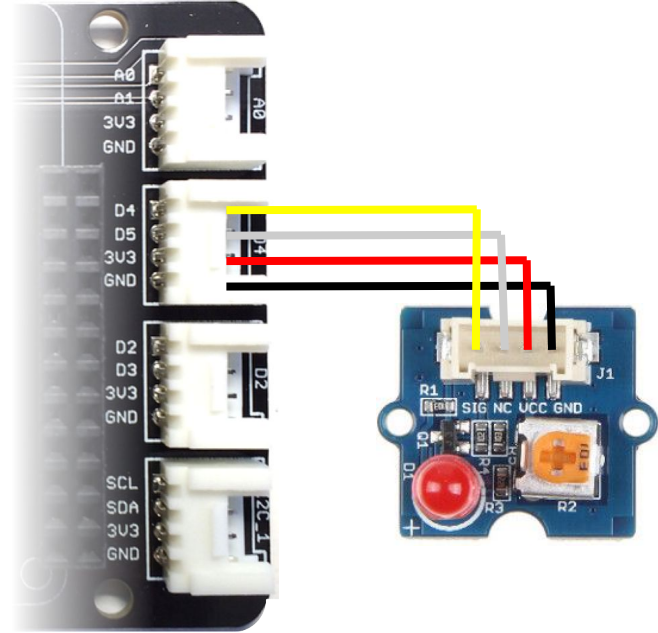
Find [code examples in the Wiki](#).

Connect to adapter port *D4*.

Maps to ESP8266 pin 0.

Or nRF52840 pin 9.

The same code works for the [buzzer](#).



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

nRF52840 or ESP8266 w/ Grove:

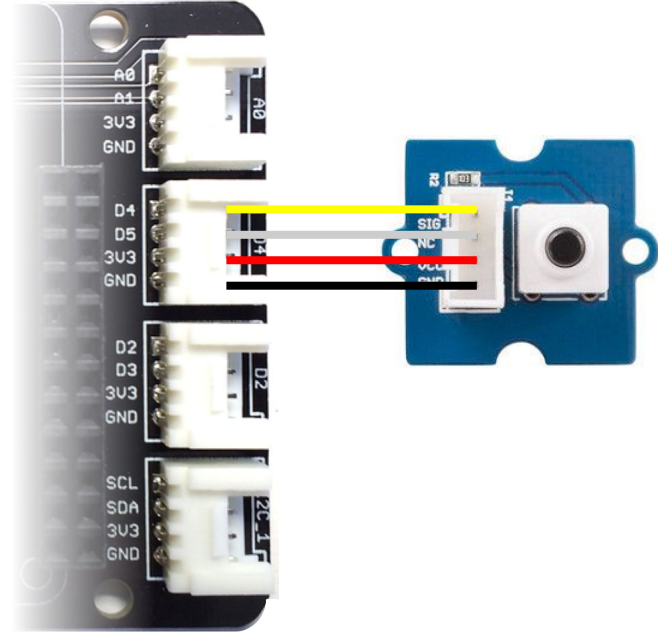
Find [code examples in the Wiki](#).

Connect to adapter port *D4*.

Maps to ESP8266 pin 0.

Or nRF52840 pin 9.

Use the serial console to see output.



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.

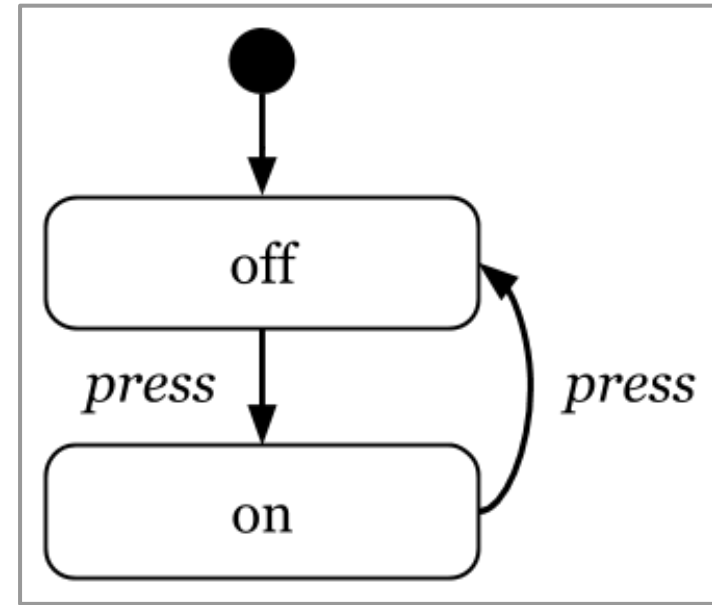
Commit the resulting code to the hands-on repo.

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. 1st button *press* => light *on*,
2nd button *press* => light *off*,
3rd => *on*, 4th => *off*, etc.

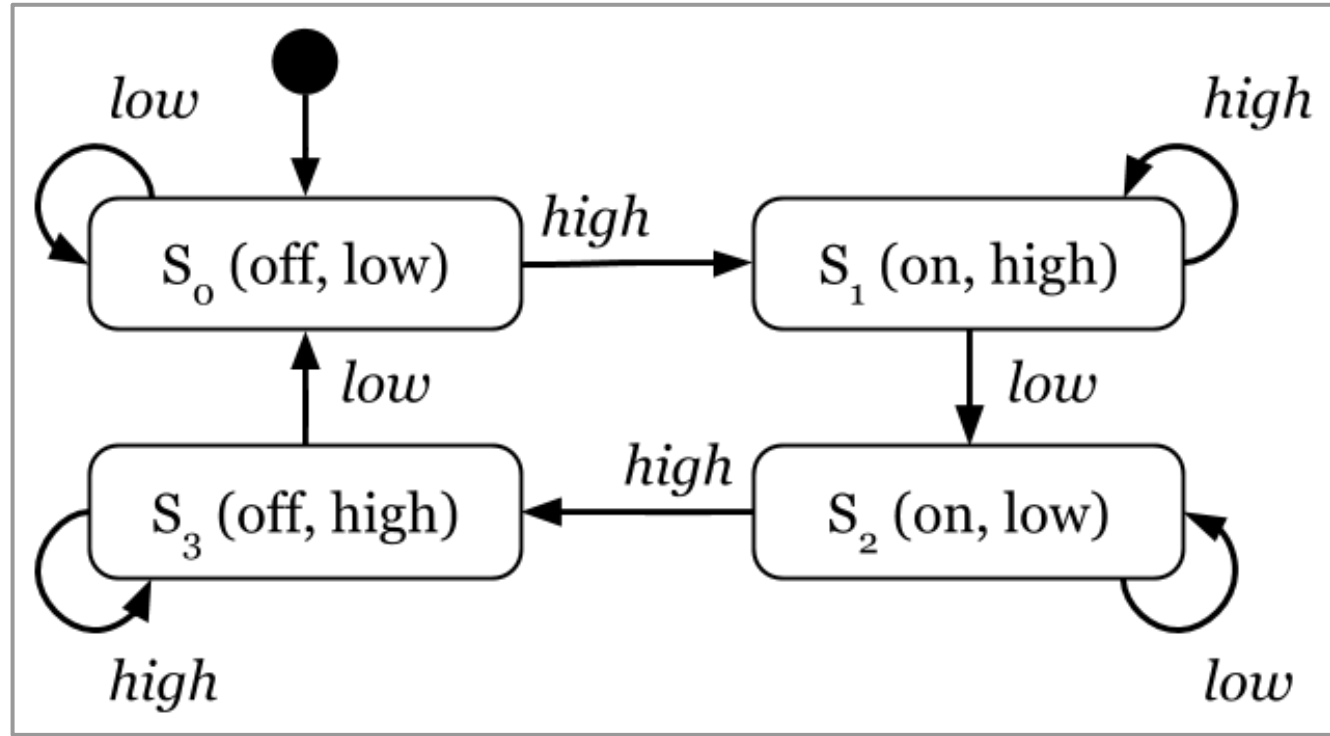


State machine (detail)

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 2nd 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)

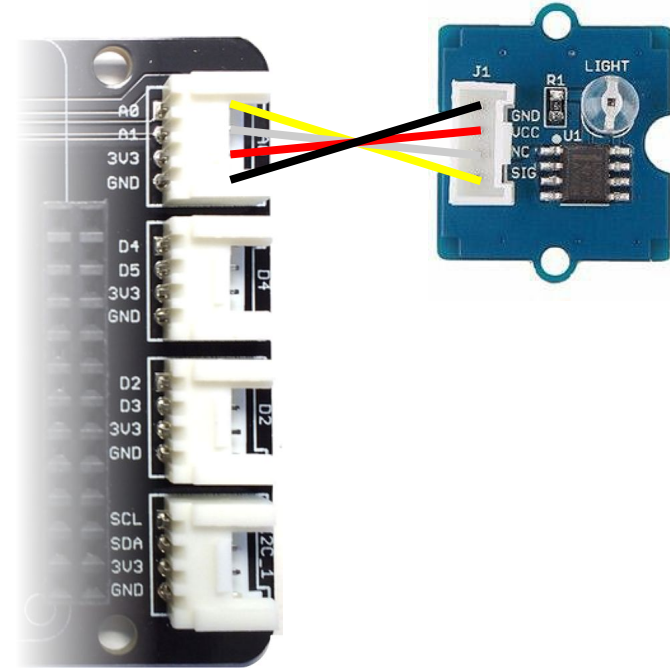
nRF52840 or ESP8266 w/ Grove:

Find [code examples in the Wiki](#).

Connect to adapter port A0.

Maps to **ESP8266 pin A0**.

Or **nRF52840 pin A0**.



Use the serial console or [serial plotter](#) to see output. 36

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);
```

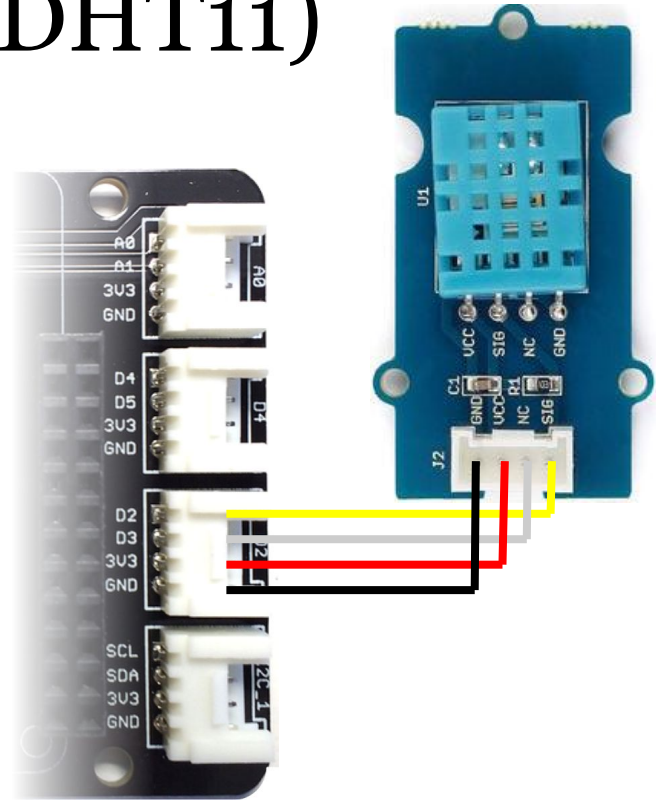
Temperature & Humidity (DHT11)

DHT11 sensors require a library.

Setup and [examples in the Wiki](#).

Connect to adapter port *D2*.

Maps to [ESP8266 pin 2](#).



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.

Feedback?

Find us on <https://fhnw-idb.slack.com/>

Or email thomas.amberg@fhnw.ch

and juerg.luthiger@fhnw.ch