Mobile Computing Microcontrollers, Sensors & Actuators

Slides: tmb.gr/mc-mcu

Overview

These slides introduce *microcontrollers*.

We learn how to run a program on one.

And how to use *sensors* and *actuators*.

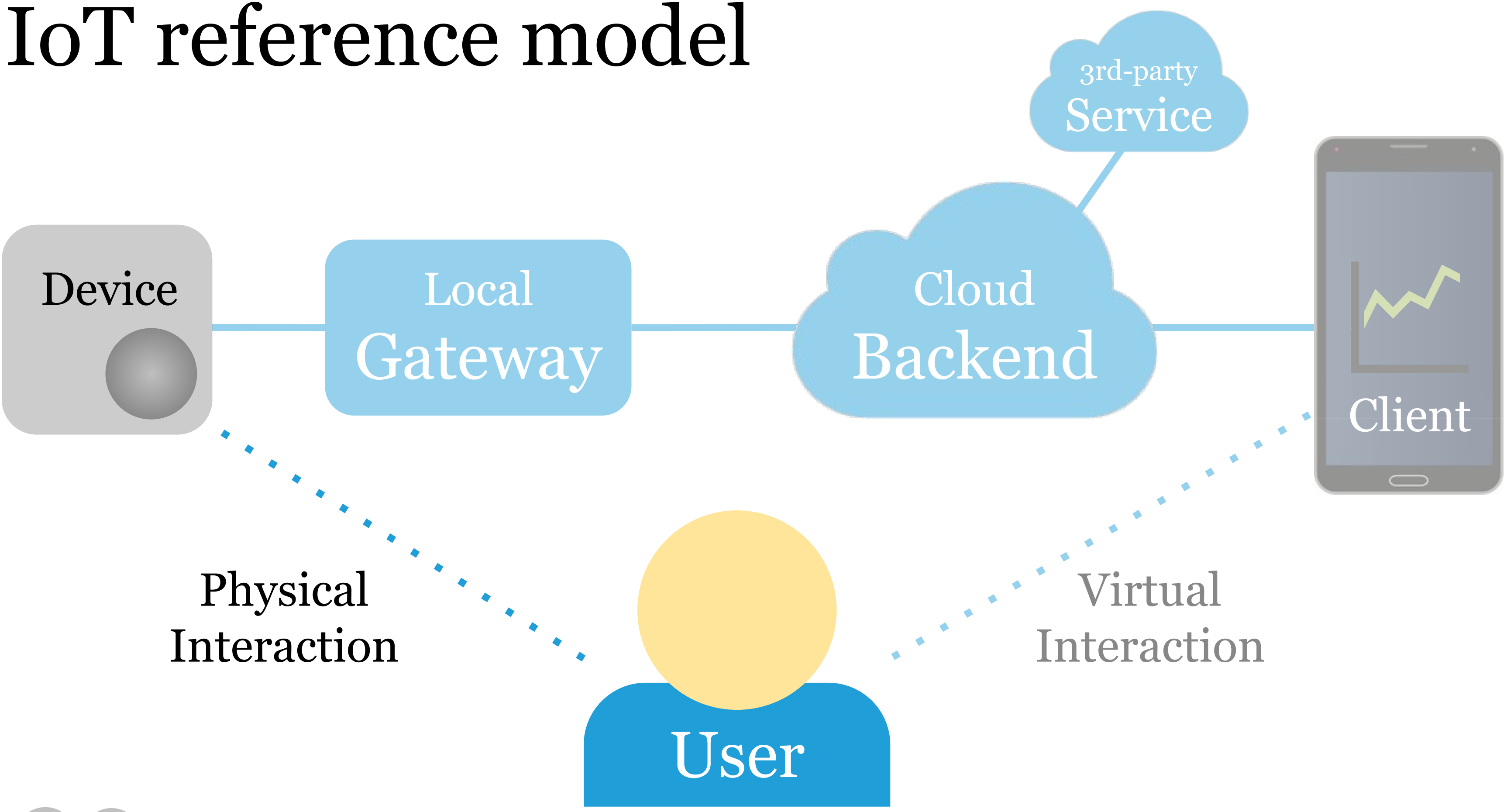
Prerequisites

Install the Arduino IDE and set up microcontrollers:

Check the Wiki entry on Installing the Arduino IDE.

Set up the Feather nRF52840 Sense for Arduino.

IoT reference model



Let's look at physical computing

On device sensing/control, no connectivity.

Sensor → Device, e.g. logging temperature.

Device → Actuator, e.g. time-triggered buzzer.

Sensor → Device → Actuator, e.g. RFID door lock.

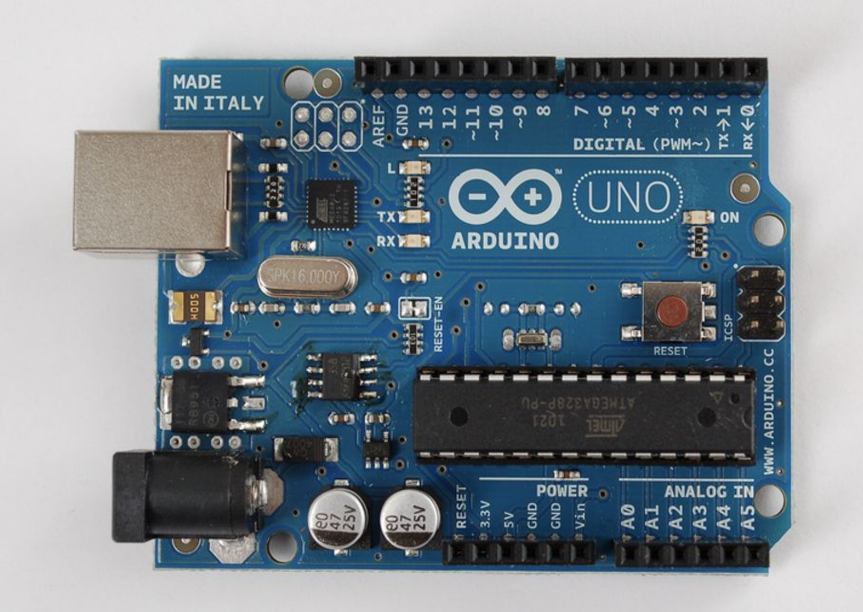
A → B: measurement or control data flow.

Arduino, a typical microcontroller

Microcontrollers (MCU) are small computers that run a single program.

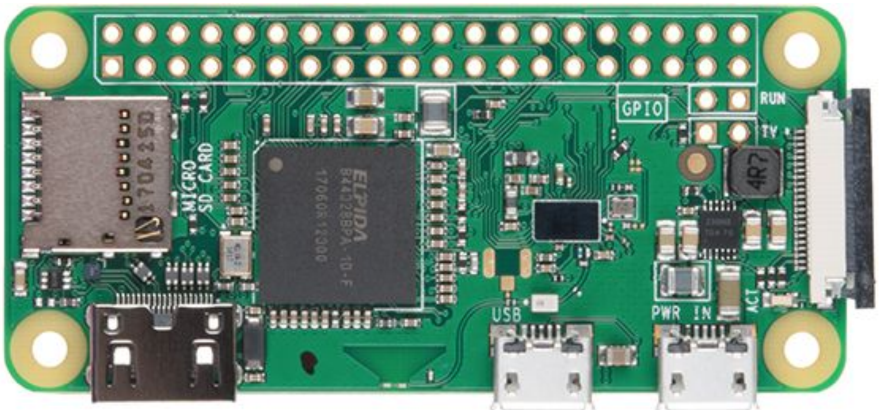
Arduino is an MCU for electronics prototyping.

Here's a video about it with Massimo Banzi.



Raspberry Pi, a single-board computer

Single-board computers like the *Raspberry Pi* are *not* microcontrollers.



They run a full Linux OS, have a lot of memory and use way more power.

Here's a video on the Pi.

Prototyping hardware form factors

Some modular prototyping hardware *form factors*:

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.

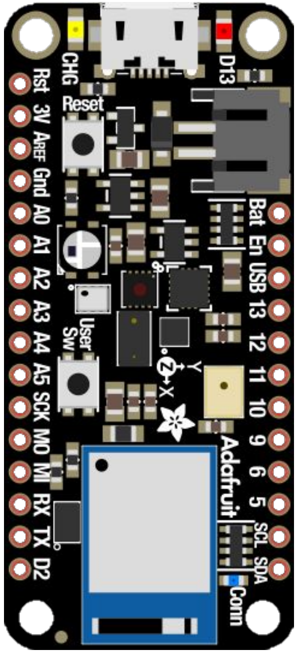
Feather nRF52840 Sense

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.

Programming a microcontroller

Microcontrollers are programmed via USB.

Code is (cross-) *compiled* on your computer.

The *binary* is *uploaded* to the microcontroller.

The uploaded program then runs "stand-alone".

Arduino IDE settings

Connect your board via USB and make sure that

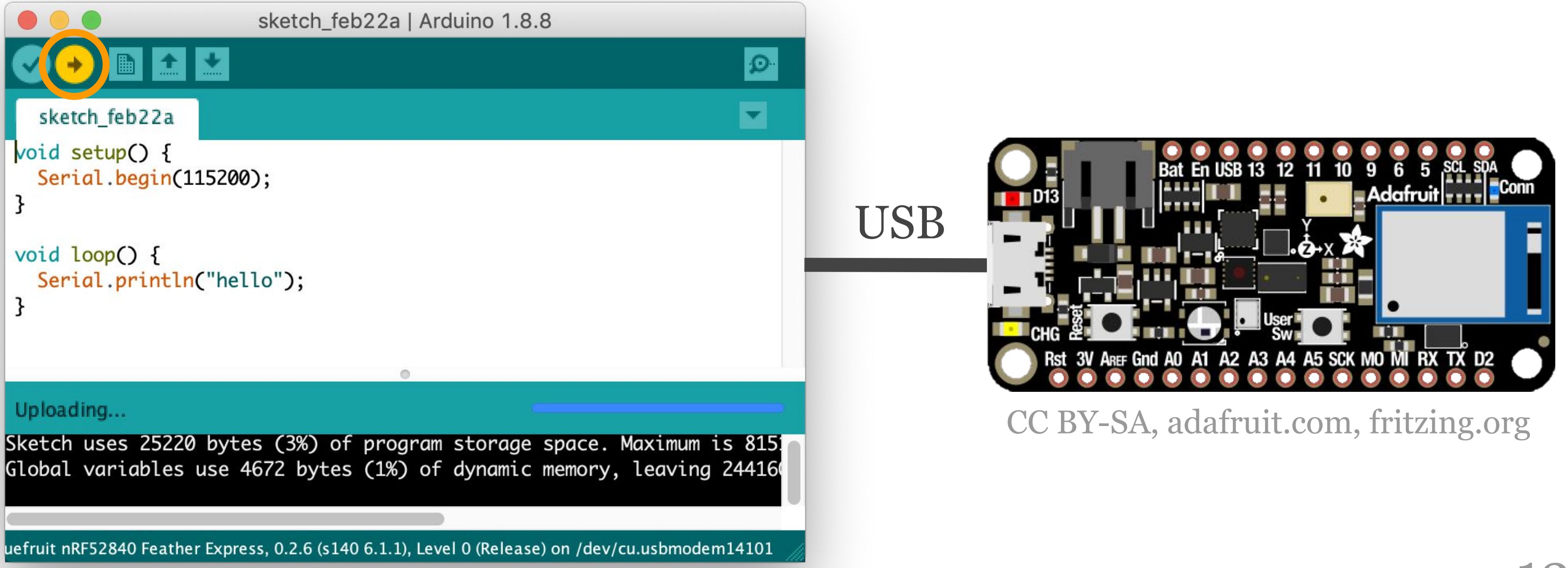
Tools > *Board* is set to your microcontroller,

Tools > *Port* matches the current USB port.

Some boards require additional settings.

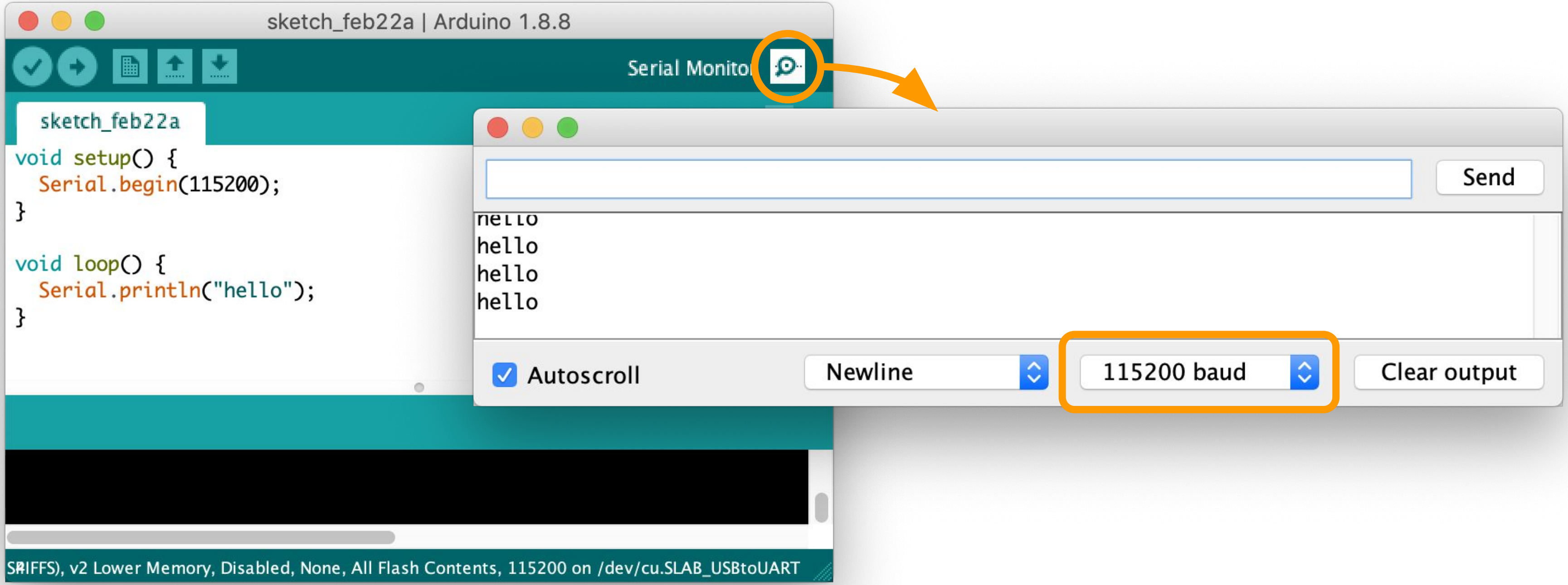
Arduino IDE program upload

The *Upload* button compiles and uploads the code.



Arduino IDE serial console

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



A typical program in Arduino C

```
void setup() { // called once at startup
  Serial.begin(115200); // set baud rate
}

void loop() { // called in a loop
  Serial.println("Hello, World!");
}
```

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.

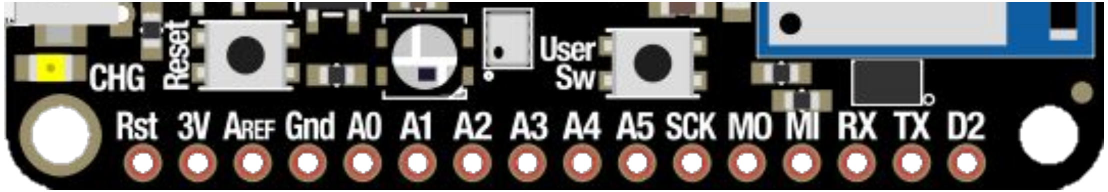
Libraries are programmed in C++.

For details, check the language reference.

General purpose input and output

Microcontrollers can "talk to" the physical world through general purpose input and output (GPIO).

GPIO *pins* allow a MCU to measure/control signals.



E.g. power, ground, analog pins, digital pin.

GPIO pin names

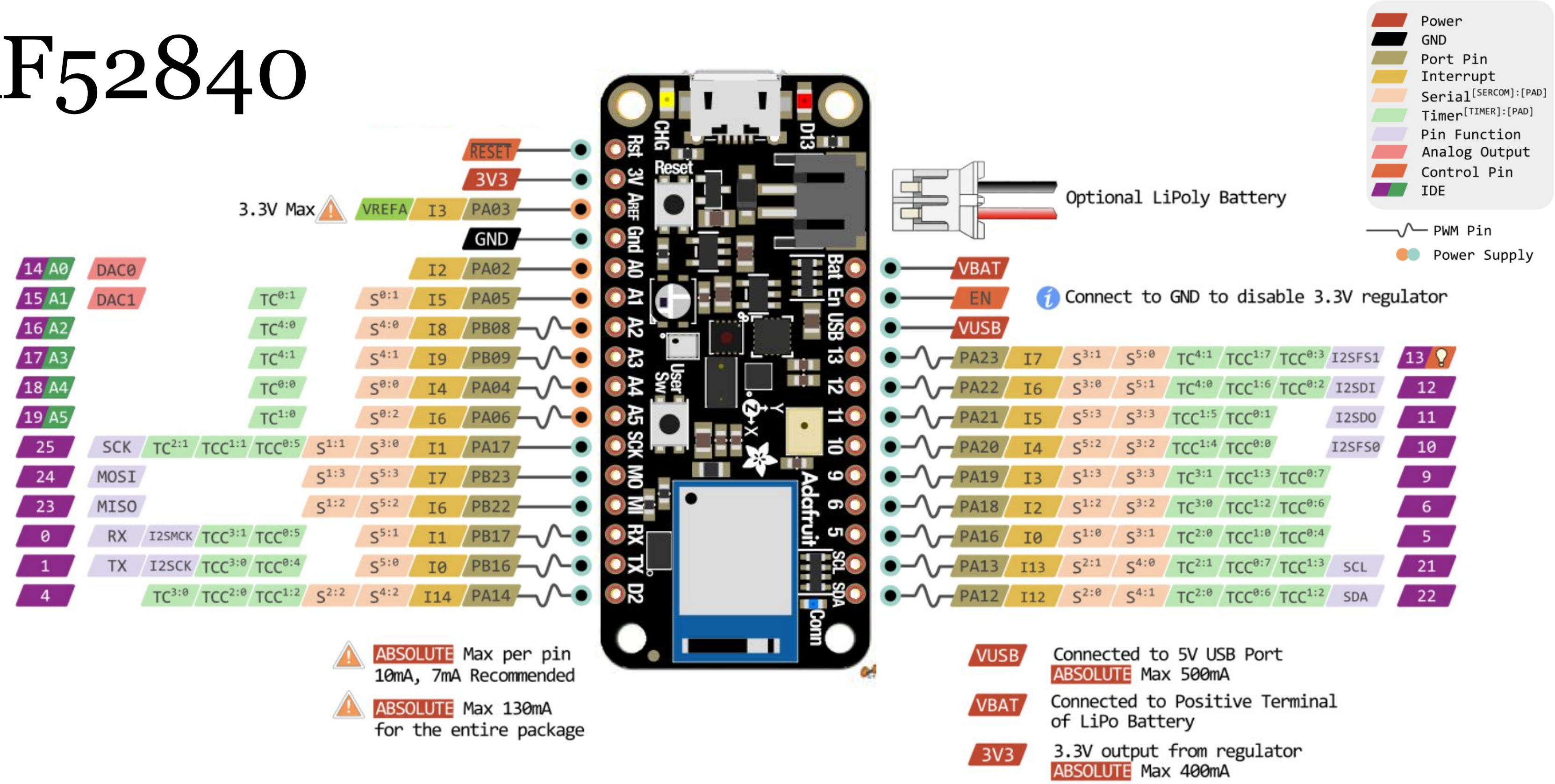
In Arduino, digital *pin names* are just numbers, e.g. pin *2*, while analog pins start with an *A*, like pin *A0*.

Which pins are available depends on the device.

The map of available pins is called *pinout*.

A pin can have multiple functions.

nRF52840



https://www.adafruit.com/product/3406

04 SEP 2017
ver 1 rev 2

Sensors read the real world

Convert physical properties to electrical *input* signals.

E.g. temperature, humidity, brightness or orientation.

Input can be *digital* (0 or 1) or *analog* (e.g. 0 - 2^10).

Measuring = *reading* sensor values from input pins.

Actuators control the real world

Convert electrical *output* signals to physical properties.

E.g. light, current with a relay or motion with a motor.

Output can be *digital* (0 or 1) or *analog* (with PWM).

Controlling = *writing* actuator values to output pins.

Wiring sensors to the MCU

Sensors and actuators exchange signals with the MCU.

For prototyping, we use wires to achieve this, e.g.

Breadboard and wires, or the *Grove* standard.

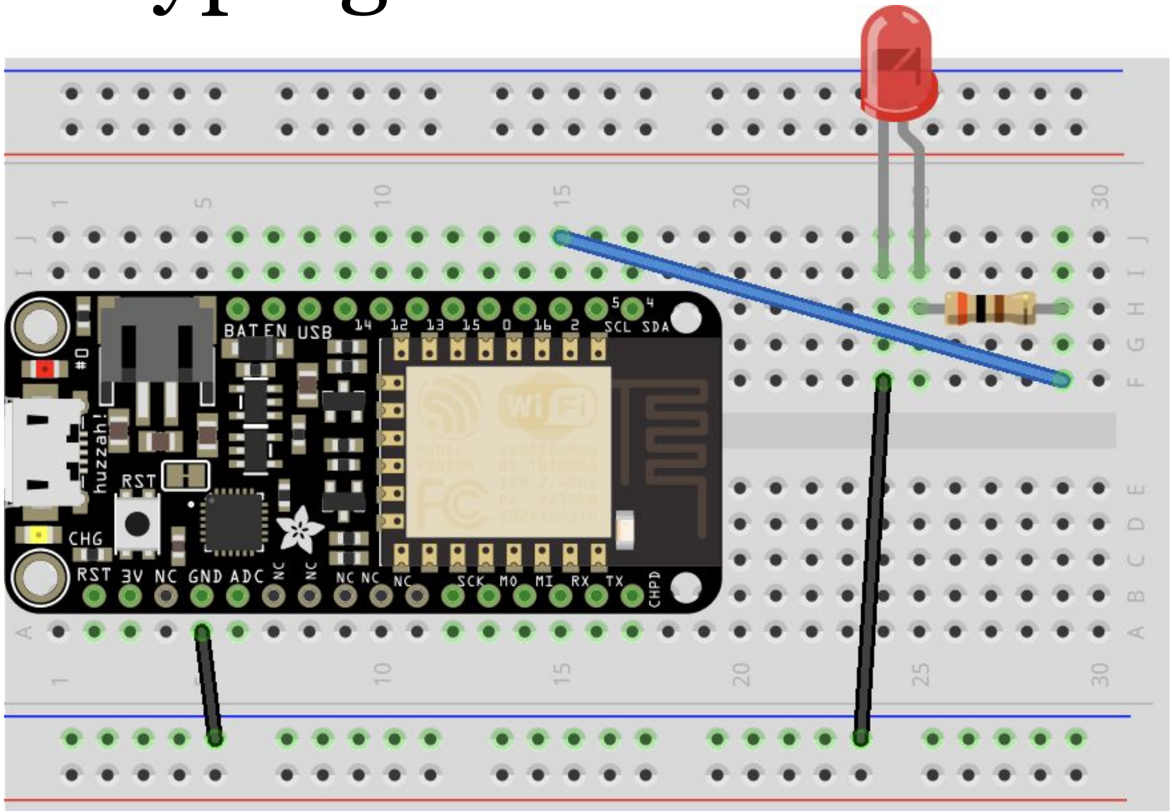
For products, custom PCBs are designed*.

*See slides on Prototype to Product.

Breadboard prototyping

Wire electronic components, no soldering.

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

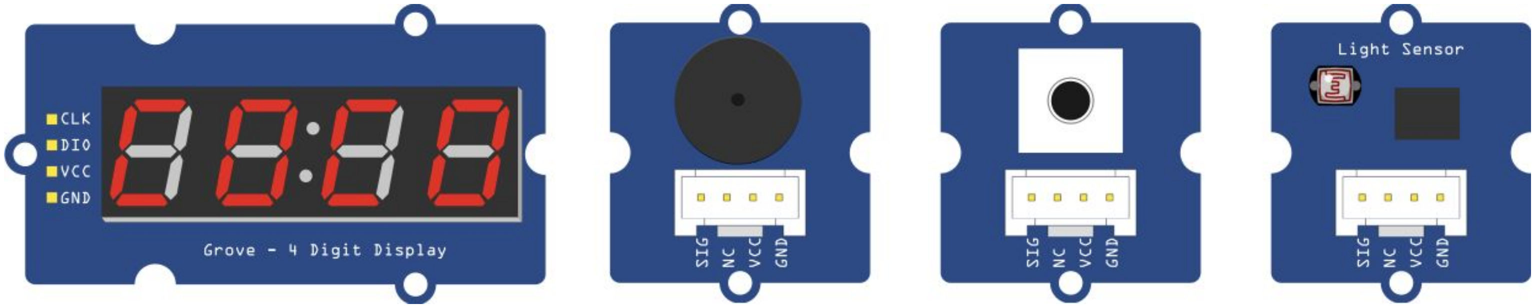


Grove wiring standard

Grove is a simple way to wire sensors and actuators.

It defines wires for power, ground and two signals.

Signals can be digital, analog, UART serial or I2C.



Arduino example code

Each Arduino library comes with example code.

And there are a number of basic examples.

See *Arduino IDE > File > Examples*

GPIO pin numbers may vary.

Use the pin mapping.

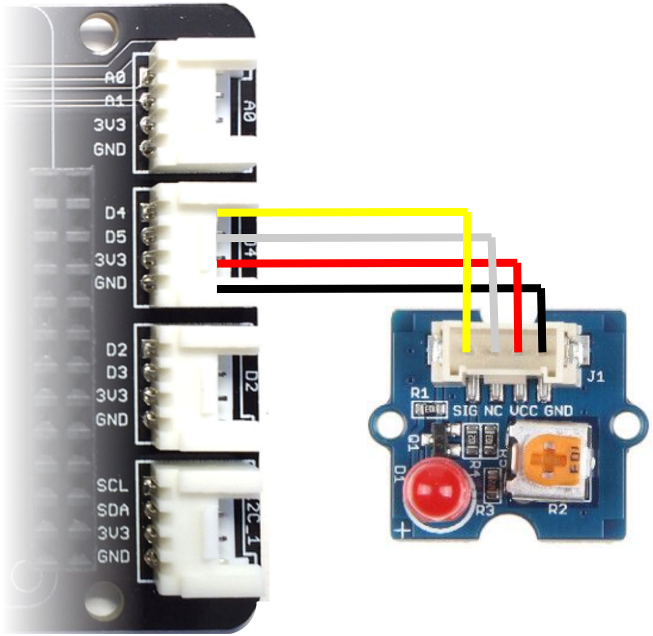
Blinking a LED (digital output)

Try *Examples > Basics > Blink*

Use *LED_BUILTIN*, i.e. pin *13*.

Or wire a LED to Grove port *D4*.

D4 maps to nRF52840 pin *9*.



The same code works with the buzzer.

Blinking a LED (digital output)

```
pin = 13; // or 9 for Grove D4

void setup() { // called once
  pinMode(pin, OUTPUT); // configure pin
}

void loop() { // called in a loop
  digitalWrite(pin, HIGH); // switch pin on
  delay(500); // ms
  digitalWrite(pin, LOW); // switch pin off
  delay(500); // ms
}
```

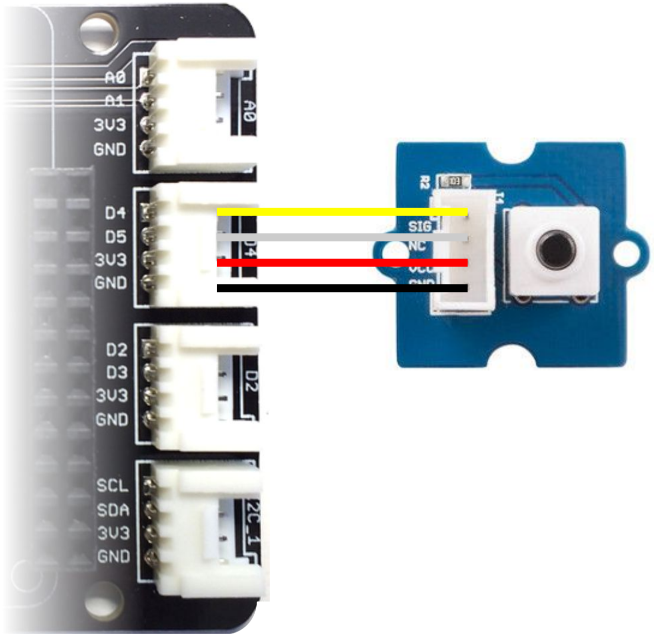
Reading a button (digital input)

Try *Basics > DigitalReadSerial*

Use the onboard button, pin 7.

Or wire a button to Grove *D4*.

D4 maps to nRF52840 pin *9*.



Use the serial console to see output.

Reading a button (digital input)

```
pin = 7; // or 9 for Grove D4

void setup() { // called once
  pinMode(pin, INPUT); // configure pin
  Serial.begin(9600);
}

void loop() { // called in a loop
  int value = digitalRead(pin);
  Serial.println(value);
  delay(500); // ms
}
```

Hands-on, 15': Button-triggered LED

Use the onboard LED, pin 13, and the button, pin 7.

Or wire a LED to Grove port *D2* and a button to *D4*.

Combine the previous examples to switch the LED.

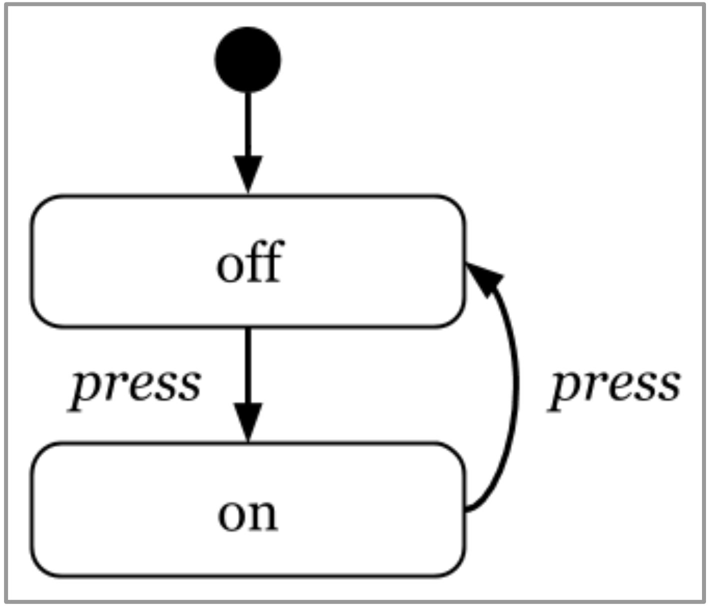
Use the pin mapping to adapt the pin numbers.

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

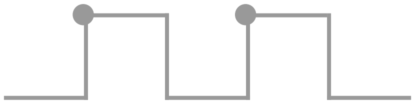


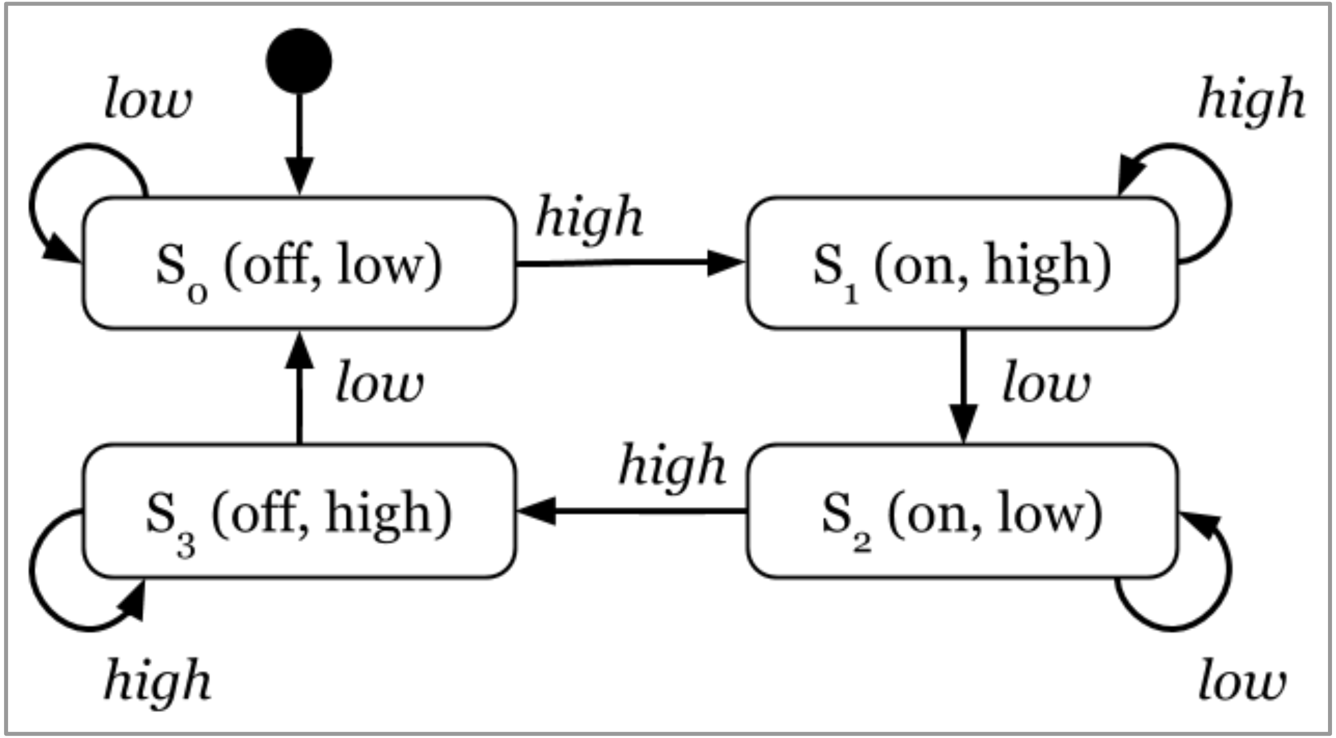
State machine (refined)

Button is *high* or *low*.

Light is *on* or *off*.

Pressed = *low* → *high*.





State machine (code snippet)

```
int b = digitalRead(buttonPin);
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;
}
```

Hands-on, 15': 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^{nd} press is *long*.

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

Commit the resulting code to the hands-on repo.

Reading a light sensor (analog input)

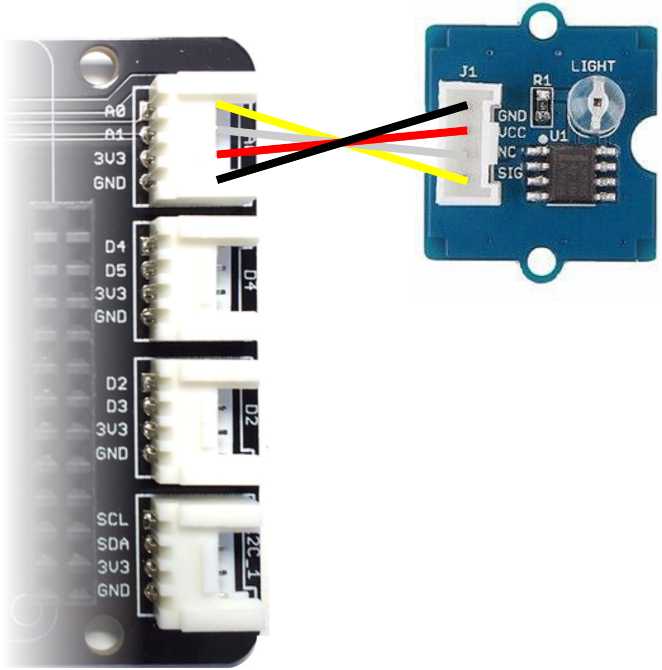
Use *Basics > AnalogReadSerial*

Wire the sensor to port/pin *A0*.

The analog value is, e.g. 0-1024*

```
int value = analogRead(pin);
```

Use serial plotter to see output.



*Range depends on ADC resolution.

Mapping input to value range

Sometimes mapping sensor value ranges helps, e.g.

0 - 1024 analog input => 0 - 9 brightness levels.

Arduino has a simple map() function for this:

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

```
e.g. result = map(value, 0, 1024, 0, 9);
```

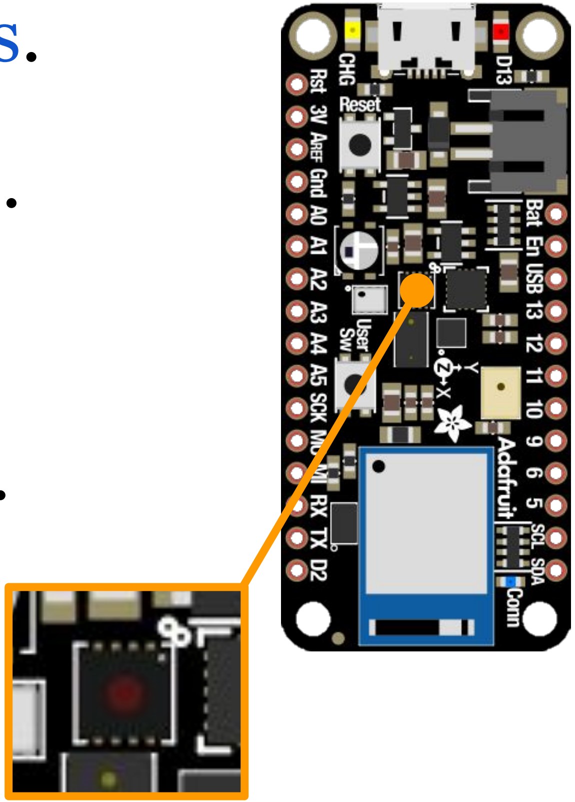
Measuring humidity (SHT30)

Onboard sensor, wired via I2C bus.

I2C uses *3V3*, *GND*, *SCL* and *SDA*.

I2C address of the sensor is *0x44*.

Requires *Adafruit_SHT31* library.



This, more sensors in the Wiki.

Hands-on, 15': 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 the kitchen timer you designed above.

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 or questions?

Join us on MSE TSM MobCom in MS Teams

Or email thomas.amberg@fhnw.ch

Thanks for your time.