# IoT Data Collection (idb) Microcontrollers, Sensors & Actuators


Slides: tmb.gr/idb-mcu

# Prerequisites

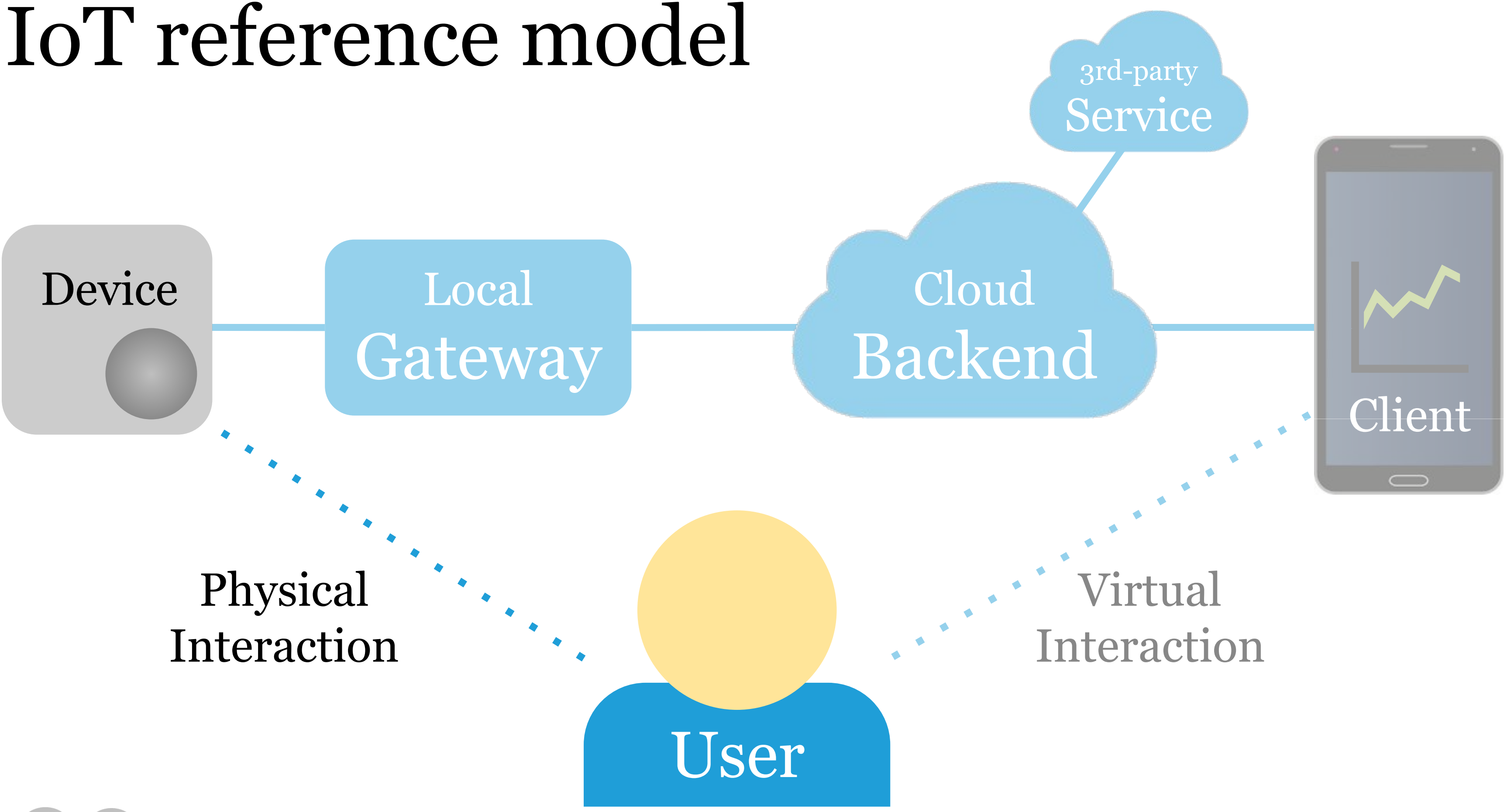
Install a Python editor and set up a microcontroller:

Check the Wiki entry on Installing the Mu editor.

And Set up the Feather nRF52840 Express.

We use CircuitPython on the nRF52840.

# 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.

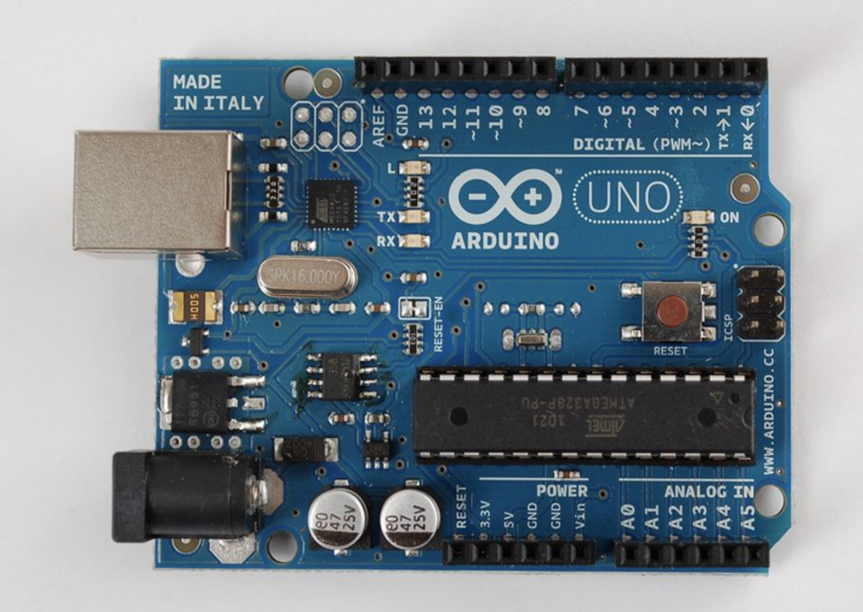
A → B: measurement or control data flow.

# Arduino, a typical microcontroller

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

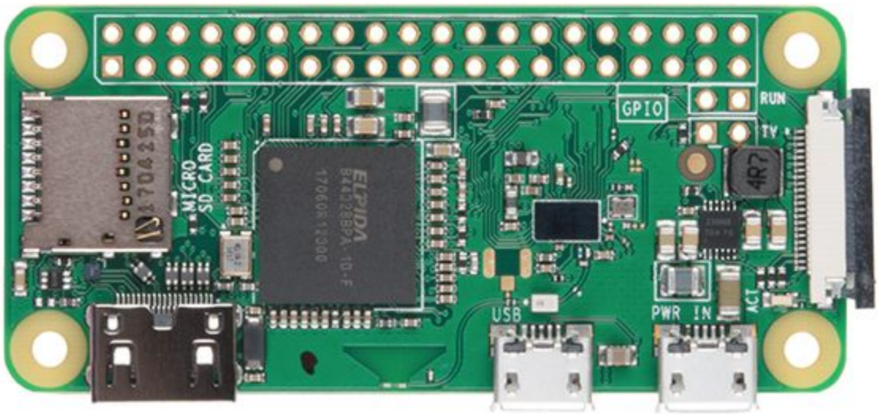
*Arduino* is an MCU for electronics prototyping.

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 a Feather compatible microcontroller.

# Feather nRF52840 Express

Microcontroller with Bluetooth 5 (and more).

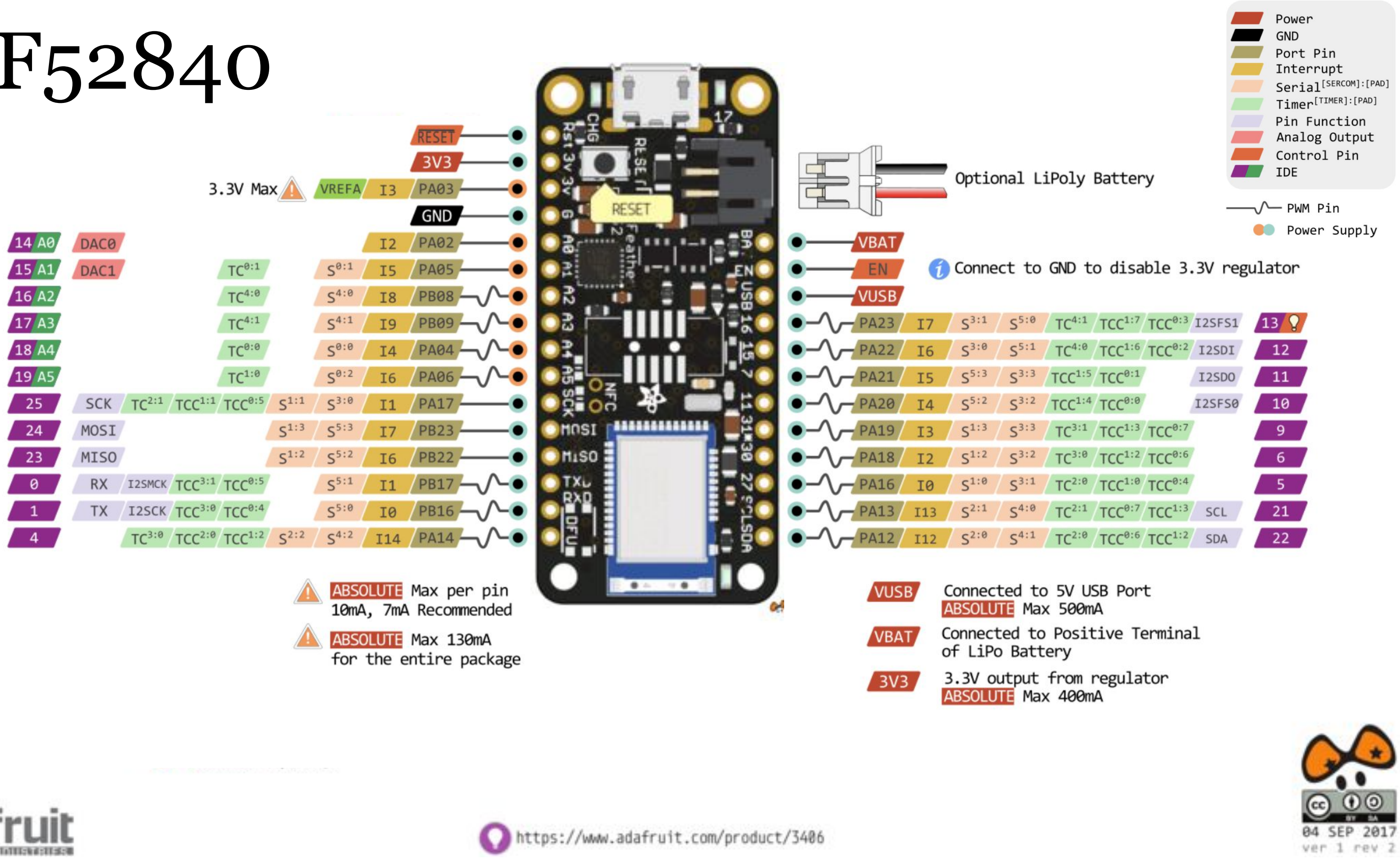
Nordic nRF52840 System on Chip (SoC).

32-bit ARM Cortex-M4 CPU with FPU.

1 MB flash memory, 265 kB RAM.

For details, check the Wiki page.

# nRF52840




adafruit INDUSTRIES

# Programming a microcontroller

Microcontrollers are programmed via USB.

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

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

For interpreted languages, *source code* is uploaded.

For both, the uploaded program runs "stand-alone".

# Programming a MCU in CircuitPython

Plug in the MCU via USB, it shows up as a drive.

Copy your source code to this drive, *CIRCUITPY*.

The target file name must* be *code.py*, e.g. on Mac:

```
$ cp blink.py /Volumes/CIRCUITPY/code.py
```

*This file runs immediately and after each reset.

# Display serial output

Connect the MCU and list serial USB devices on Mac:

```
$ ls /dev/tty.usb*
```

Open a serial connection to see CircuitPython output:

```
$ screen /dev/tty.usb<TAB> 115200
```

On Windows, check COM ports to find your device, then use the PuTTY tool to open a connection.

Once connected, press *CTRL-D* to reload code.py.

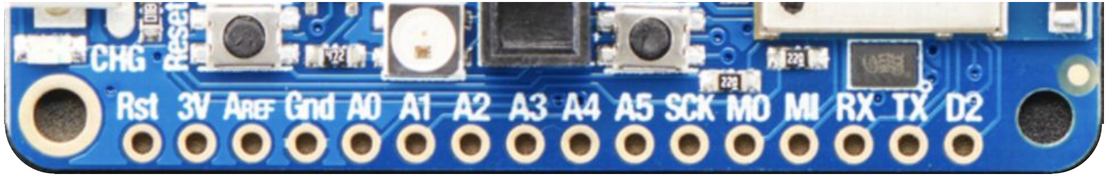
# A typical program in CircuitPython

```
import … # see blink.py

led = digitalio.DigitalInOut(board.RED_LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(1)
    led.value = False
    time.sleep(1)
```

# 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 CircuitPython, digital *pin names* start with *D*, e.g. *board.D2* and analog pins, like *board.A0*, with an *A*.

The *board* library* contains all known pin names, including wire protocol specific names, e.g. for I2C.

Which pins are available depends on the device.

*Microcontrollers are also called *boards*.

# 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 - 2^16).

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.

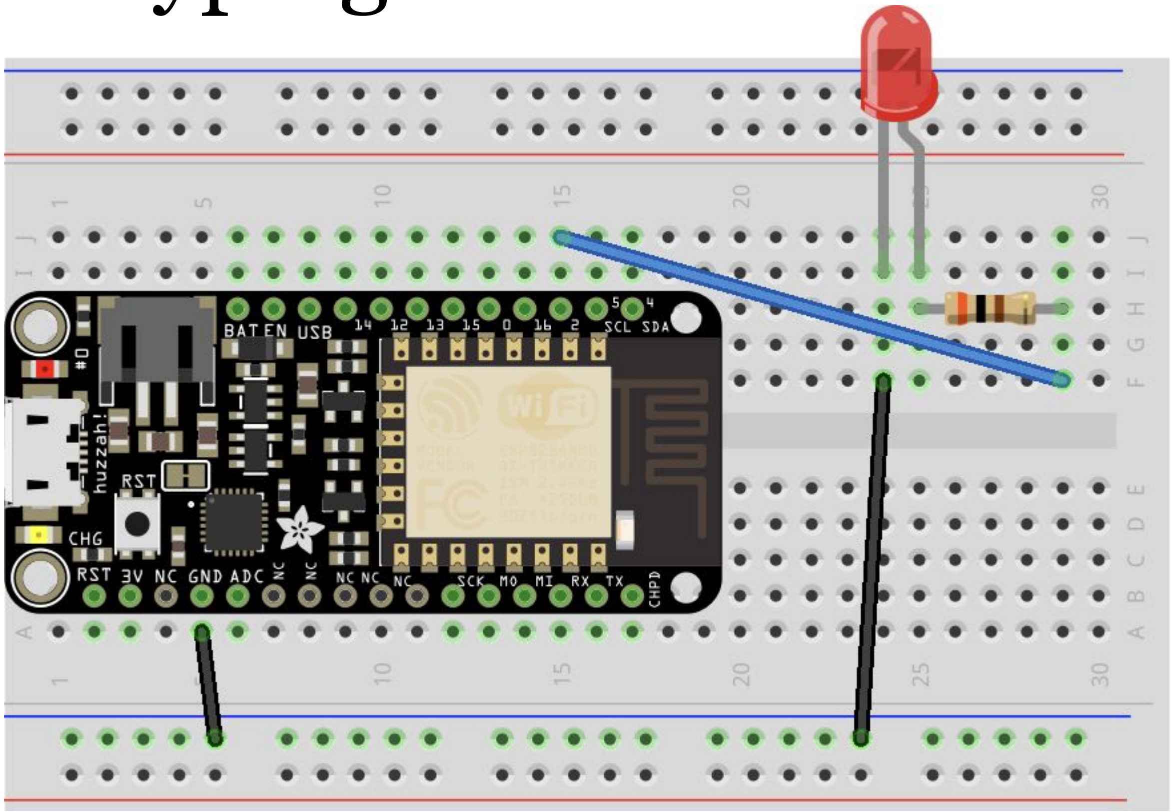
For products, custom PCBs are designed*.

*See slides on Prototype to Product.

# Breadboard prototyping

Wire electronic components, no soldering.

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

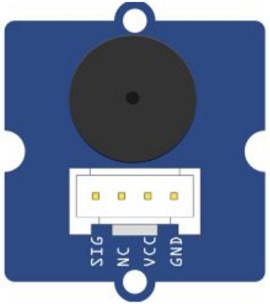


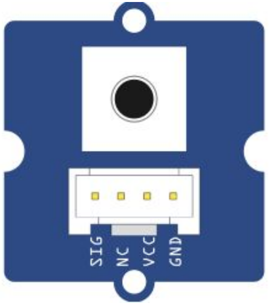
# Grove wiring standard

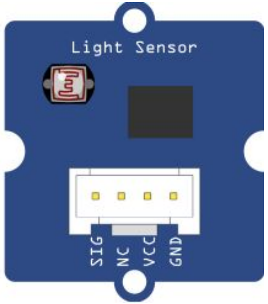
Grove is a simple way to wire sensors and actuators.

It defines wires for power, ground and two signals.

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

# Reading sensors in CircuitPython

Digital inputs can be read with the *digitalio* library.

Analog inputs can be read with the *analogio* library.

Other sensors require their own dedicated libraries.

Each sensor is wired to the MCU through 1..n pins.

Use the pin mapping to find the right pin numbers.

# Reading digital input

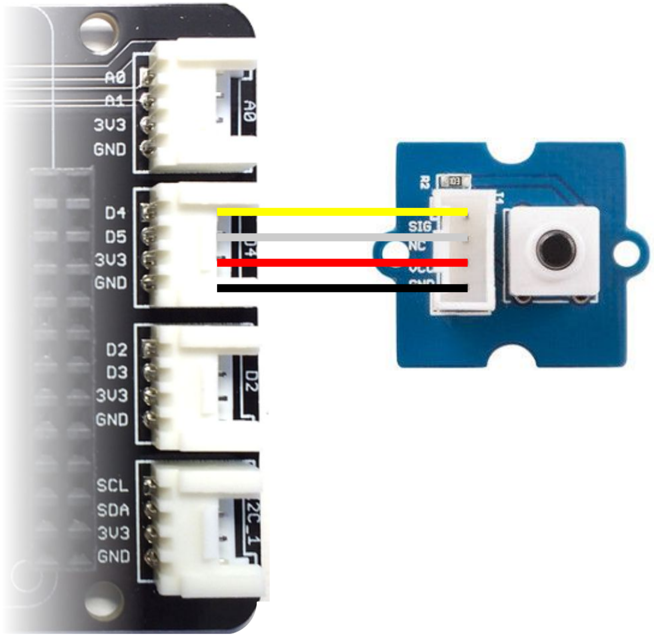
```
import … # see digital_input.py

sensor = digitalio.DigitalInOut(board.D9)
sensor.direction = digitalio.Direction.INPUT
sensor.pull = digitalio.Pull.UP # for button

while True:
    print(sensor.value) # 0 or 1
    time.sleep(0.1)
```

# Reading digital input with nRF52840

The button is a digital sensor.

Use the Feather Grove adapter.

Grove port *D4* maps to pin *D9*.




Grove port numbers differ from pins, see mapping.

# Reading analog input

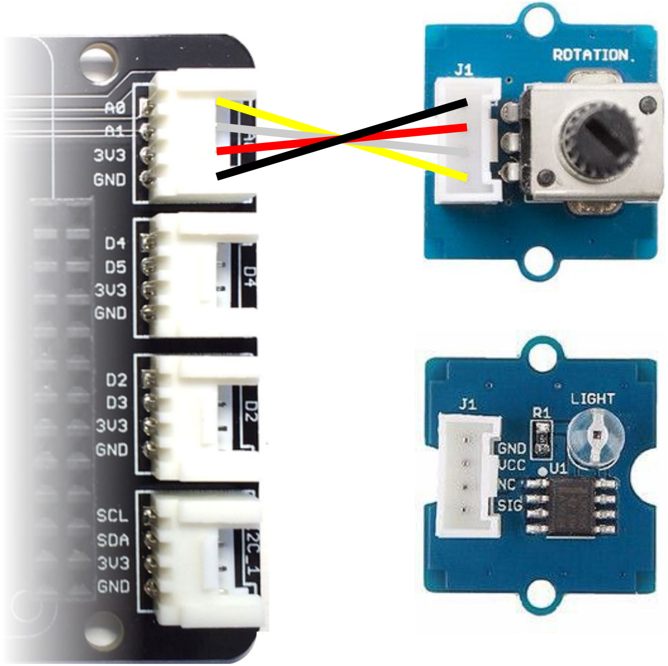
```
import … # see analog_input.py

sensor = analogio.AnalogIn(board.A0)

while True:
    value = sensor.value # 0 to 2^16
    voltage = (value * 3.3) / 65536
    print((value, voltage))
    time.sleep(0.1)
```

# Reading analog input with nRF52840

Try a rotary angle or light sensor.

Use the Feather Grove adapter.

Grove port *A0* maps to pin *A0*.




Analog Grove port *An* maps to pin *An* for each *n*.

# Using CircuitPython libraries

CircuitPython libraries come as **.mpy* files, see e.g.

```
$ ls adafruit-circuitpython-bundle-…/lib
```

Many CircuitPython libraries include examples, e.g.

```
$ ls adafruit-circuitpython-bundle-…/examples
```

To use a library, copy its files to *CIRCUITPY/lib*, e.g.

```
$ cp adafruit-circuitpython-bundle-…/lib/\
 adafruit_dht.mpy /Volumes/CIRCUITPY/lib/
```

Space on the device is limited, remove unused files.

# Reading a DHT11 sensor

```
import … # see dht.py

dht = adafruit_dht.DHT11(board.D9)

while True:
    try:
        print((dht.temperature, dht.humidity))
    except RuntimeError:
        pass # reading failed, do nothing
    time.sleep(5)
```

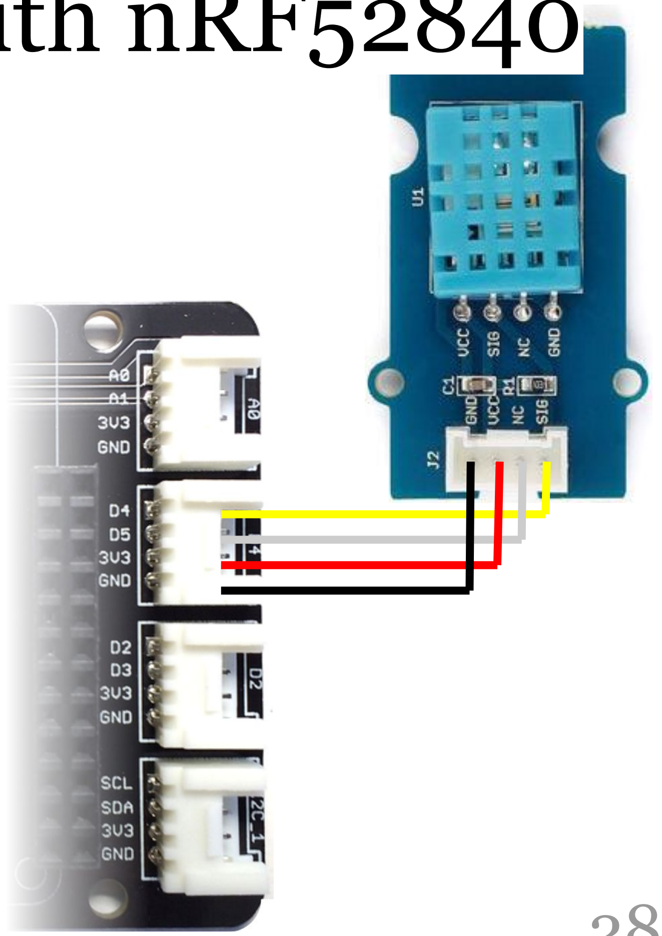
# Reading a DHT11 sensor with nRF52840

The DHT11 requires a library:

Copy *adafruit_dht.mpy* to */lib*.

Use the Feather Grove adapter.

Grove port *D4* maps to pin *D9*.




Mu editor can plot serial data.

# Controlling actuators in CircuitPython

To control digital outputs use the *digitalio* library.

Analog (PWM) outputs require the *pulseio* library.

Other actuators require their own specific libraries.

Each actuator is wired to the MCU through 1..n pins.

Use the pin mapping to find the right pin numbers.

# Writing digital output

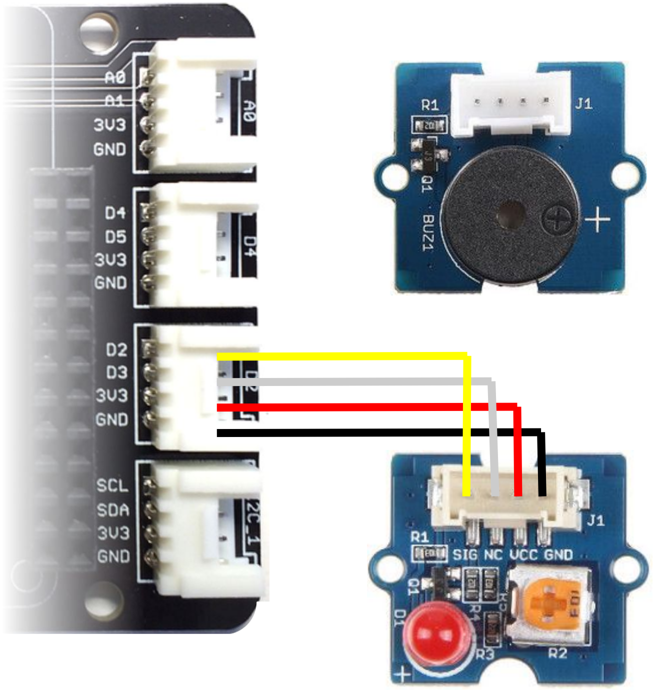
```
import … # see digital_output.py

actuator = digitalio.DigitalInOut(board.D5)
actuator.direction = digitalio.Direction.OUTPUT

while True:
    actuator.value = True
    time.sleep(1)
    actuator.value = False
    time.sleep(1)
```

# Writing digital output with nRF52840

The LED is a digital actuator.

Use the Feather Grove adapter.

Grove port *D2* maps to pin *D5*.




This also works for the buzzer or a relay.

# Writing analog (PWM) output

*Pulse-width modulation* (PWM) creates analog output.

A PWM pin is switched on for short times periodically.

*Duty cycle*: fraction of a period where a signal is active.

A higher % value for the duty cycle means more power.

*Frequency*: Hz (1/s) value, depends on the application.

Wikipedia has a article on PWM, Adafruit a tutorial.

# Pulsing an LED with PWM output

```
import … # see pwm_led_pulse.py

led = pulseio.PWMOut(board.A0,
  frequency=5000, duty_cycle=0)

while True:
    for i in range(100):
        value = int(i * 65535 / 100)
        led.duty_cycle = value
        time.sleep(0.01)
```

# Buzzer tones with PWM output

```
import … # see pwm_buzzer_tones.py

cycle = 65535 // 2 # on 50%
buzzer = pulseio.PWMOut(board.A0,
  duty_cycle=cycle, variable_frequency=True)

while True:
    for f in (262, 294, 330, 349, 392):
        buzzer.frequency = f
        time.sleep(0.25)
```

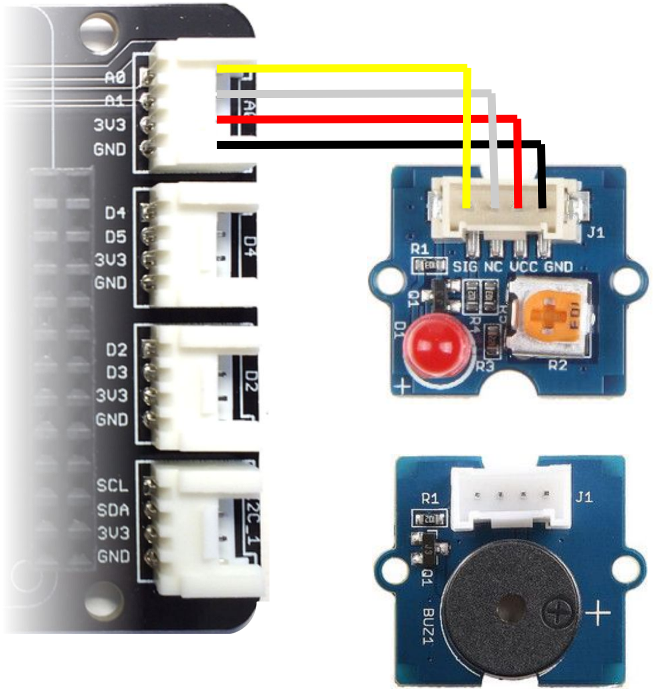
# Writing PWM output with nRF52840

LED brightness depends on power.

Buzzer tone depends on frequency.

Use the Feather Grove adapter.

Grove port *A0* maps to pin *A0*.




Most nRF52840 pins can do PWM.

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

So far we've seen individual sensors and actuators.

Try to combine the LED and the button example.

Pressing the button should switch on the LED.

Does your solution match a real light switch?

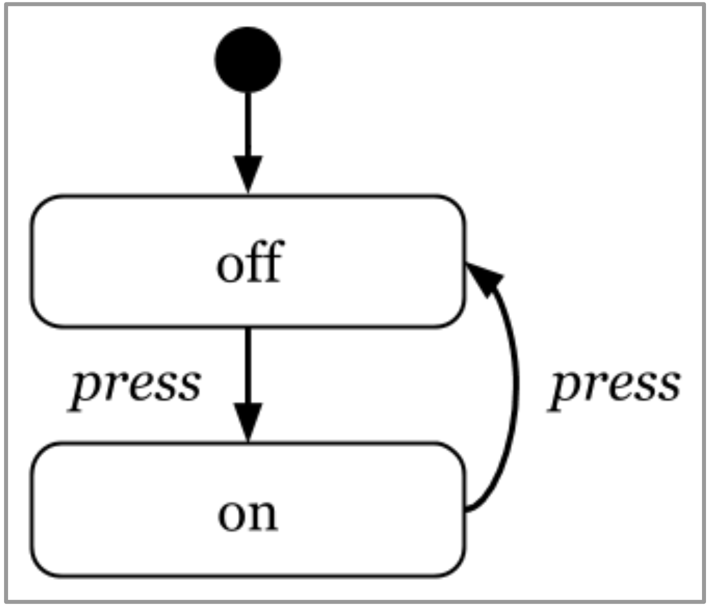
Done? Compare your solution to switch.py.

# 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.

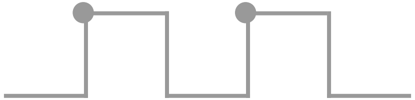


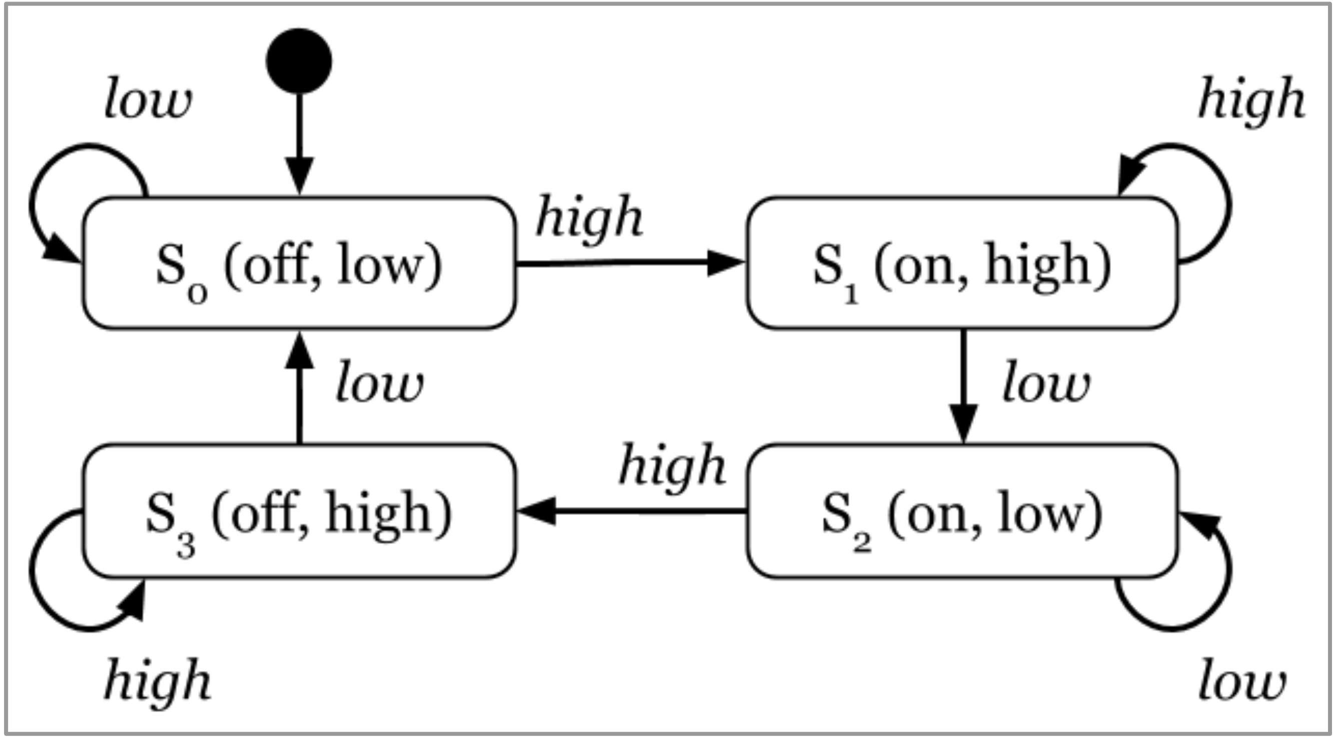
# State machine (detail)

Button is *high* or *low*.

Light is *on* or *off*.

Pressed = *low* → *high*.





# State machine in CircuitPython

```
s = 0 # initial state

while True:
    b = button.value
    if s == 0 and b: s = 1
        led.value == True
    elif s == 1 and not b: s = 2
    elif s == 2 and b: s = 3
        led.value = False
    elif s == 3 and not b: 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 set up.

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

Long means > 1s, use the time.monotonic() function.

# Summary

We programmed a microcontroller in CircuitPython.

We used digital and analog sensors and actuators.

We saw how to convert a state machine to code.

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