# IoT devices
## Structure and programming

# From "things" to IoT devices

- **Computational intelligence** (microprocessor/controller + software)
- Internet (**network connectivity**) with direct or indirect access to non-local resources


- **Dedicated purpose and interface**
  - Main purpose is not to compute (unlike a traditional computer)
  - The computing (software) and communication part is embedded into another interface
- **Interaction between the physical and the digital world**
  - **Sensors** – listening to the physical world
  - **Actuators** – taking action in the physical world

# Sensors and actuators

## Sensors?

- Temperature
- Proximity
- Light sensor
- Microphone
- Buttons (push / switch)
- Accelerometer
- Magnetometer (compass)
- RFID sensor
- …

## Actuators?

- LED (light emitting diode)
- Display / monitor
- Motor
- Speaker
- Pump
- Valve
- …

# IoT system architecture




Subscribes to information in the system and may integrate with other systems

Service for web and cloud integration

Software components for sensor/actuator interaction and control

https://weathermap.netatmo.com/

# DAT59: IoT and cloud module

| Week | Topic | |
|---|---|---|
| 44 | **IoT devices and programming** | |
| | IoT and Cloud in Healthcare | Fazle Rabbi |
| | IoT devices and programming | |
| 45 | **IoT Protocols and Middleware** | |
| | Smart Grid Monitoring - an IoT Usecase | Reza Arghendeh |
| | IoT Protocols and Middleware | |
| 46 | **Cloud-based IoT and backend analytics** | |
| | Security and IoT | |
| | Cloud-based IoT and backend analytics | |

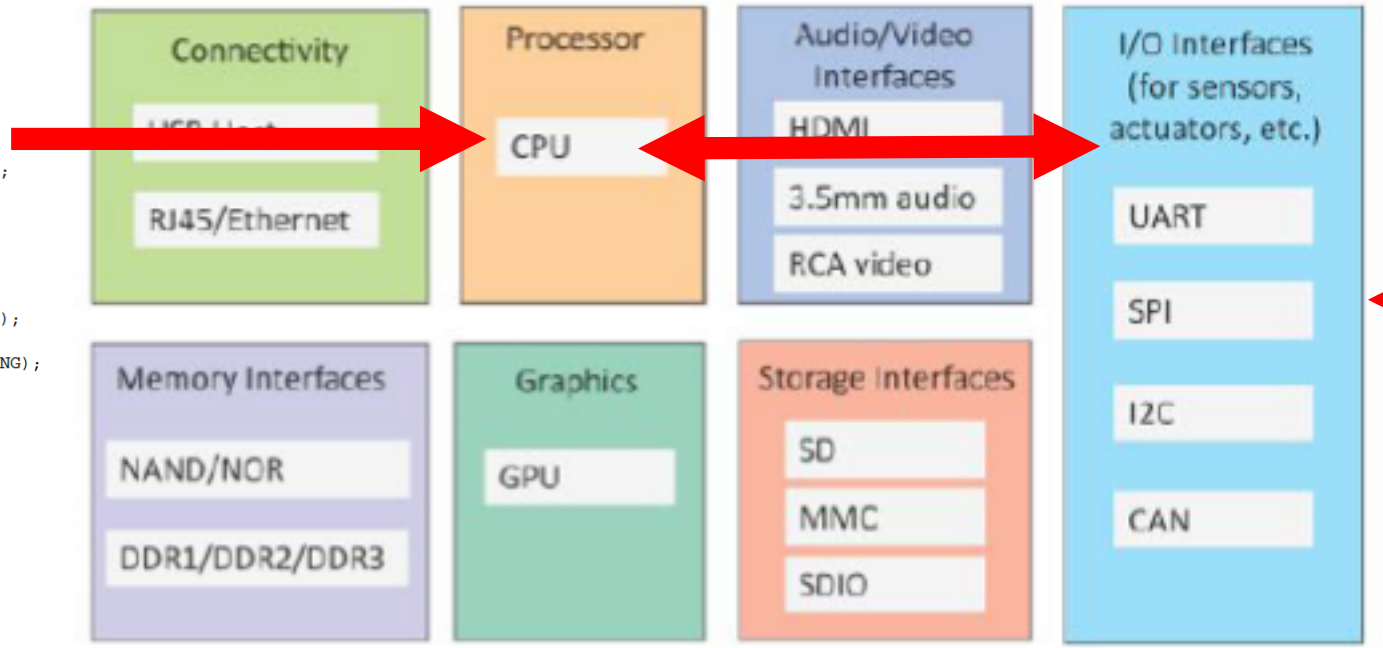
- **Combined lecture and lab**
- **Three small assignments – one for each week**

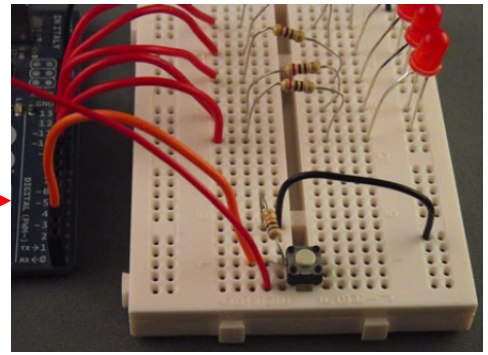
# IoT devices and embedded systems

# IoT device

- **Embedded system with Internet connectivity comprised of software and electronics (hardware)**

```
volatile int state = LOW;
int ledArray[] = {8, 9, 10, 11, 12};
int count = 0;
int timer = 75;
int pause = 500;

void setup(){
  for (count=0;count<5;count++){
    pinMode(ledArray[count], OUTPUT);
  }
  attachInterrupt(0, ledOnOff, RISING);
}
```



**Software / Application** **Microcontroller | Microprocessor** **Sensors | Actuators**

# Embedded systems (innebygd system)

- **Computing devices with complexity embedded in the device and programmed for a specific task**
  - User interaction via a simple interface
  - **Home automation:** washing machines, fridges, ..
  - **Industrial automation:** pumps, heaters,...
  - ...

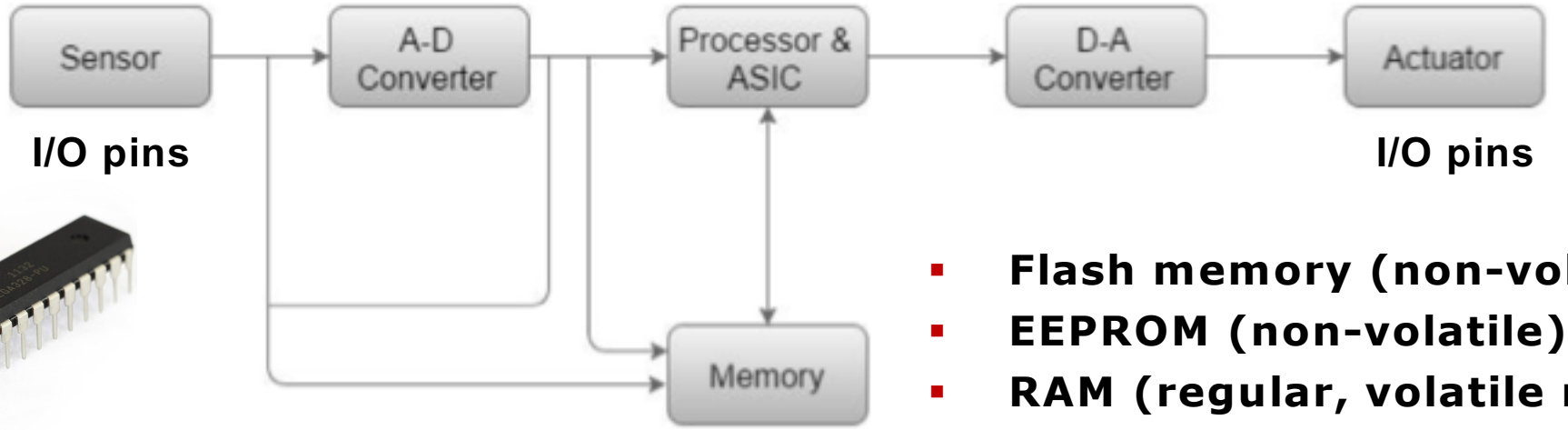
- **Also widely used in real-time and safety critical systems**
  - **Health:** insulin pumps, pacemakers,... (life-critical systems)
  - **Automation:** power plants, planes, cars, ...
  - ...
- **Hardware and software are engineered to work together (hardware/software co-design)**

# Embedded systems design

- **System development aimed at price/cost-critical domains**
  - Perform its task and satisfy requirements - at the lowest possible price
  - Requirements: functionality, performance, reliability, physical size
- **An over-engineered solution is not an option**
  - Hardware is the main cost - and only the pieces needed are assembled
  - Design constraints: processing power, memory- and power consumption
- **Hardware and software must be designed together**
  - Need to decide what hardware and electrical components is to be used
  - Software developer needs to understand what the processing unit can do
  - Software developer needs to understand sensor and actuator interfacing

# Embedded systems structure

- **Typically microcontroller based: Integrated Circuit (IC) that executes a program stored in non-volatile memory**

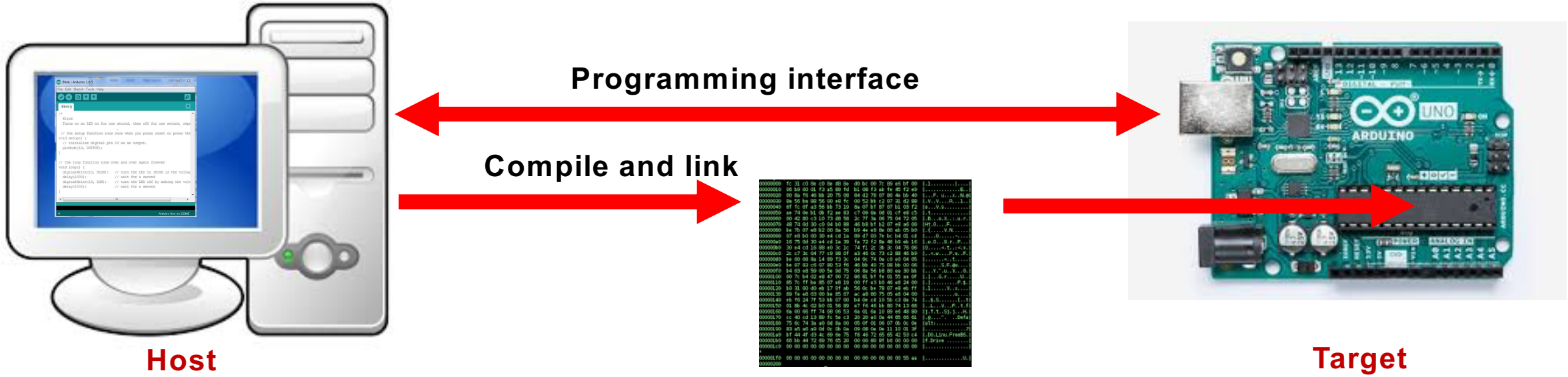



- **Flash memory (non-volatile)**
- **EEPROM (non-volatile)**
- **RAM (regular, volatile memory)**

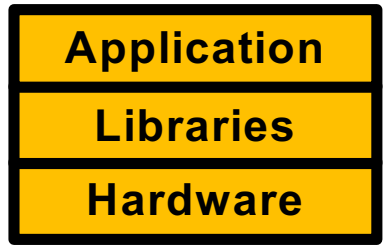
- **Additional Application-Specific Integrated Circuits (ASICs)**
  - **Audio/video signal processing and compression,**
  - **Network controllers for Ethernet, WiFi, Bluetooth,…**
  - **USB controllers, …**

# Embedded systems software

- **Typically developed in a combination of C/C++ and assembly language - because of design constraints**
  - The software development environment / tool-chain is on a host computer
  - The program is cross-compiled and linked on the host computer and then moved to the target platform (microcontroller) via a programming interface.



# Embedded systems software




- **Target platform libraries are often used**
  - Shield the developer from microcontroller details
  - Higher-level interface to sensors and actuators
  - Protocol details for synchronisation and communication with ASICs
- **Boot loader provide low-level device management**
  - Stored in non-volatile memory (Flash or EEPROM)
  - Manages application programming interface of the microprocessor
  - Handles system reset and loading application code on power-on
- **Testing and debugging can be challenging**
  - Controllability (stop, resume), accuracy (timing), and observability
  - Simulation and model-based testing often applied
  - Debug monitor and embedded debug logic on the target device

# IoT prototyping devices

# IoT prototyping devices

## Arduino Uno


**Microcontroller-based**

**ATmega128 microcontroller (8-bit – 16 MHz)**

**No operating system:**

**application controls the hardware directly**

**32Kb (Flash), 2kb (SRAM), EEPROM (1kb)**

**USB programming interface**

https://www.microchip.com/wwwproducts/en/ATmega328

## Raspberry Pi 3 B+

**Microprocessor-based**

**Broadcom BCM2837B0 QuadCore, 64-bit, 2 GHz**

**Operating system: controls the hardware and supports multiple concurrent applications**

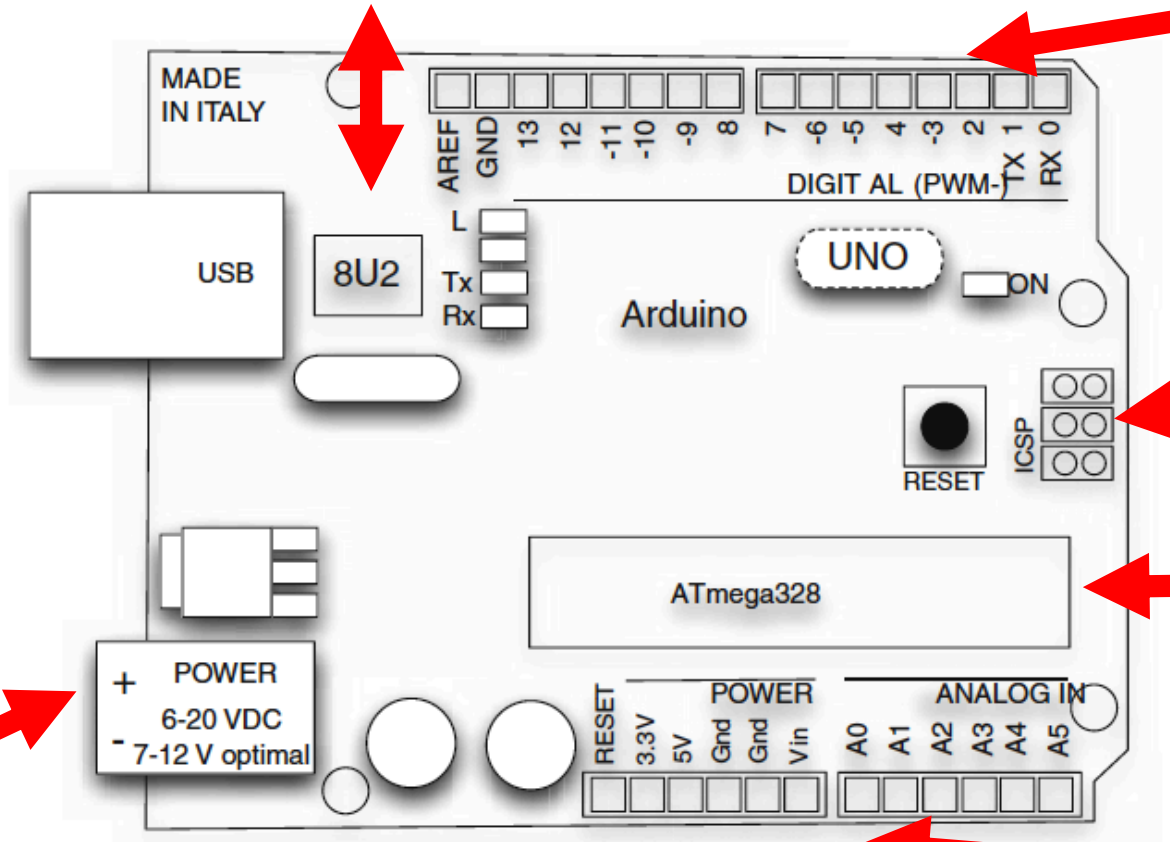
**1Gb SDRAM**

**Software development on the device itself**

https://www.broadcom.com/products/embedded-and-networking-processors/communications/bcm58712/

# Arduino development board

ATMega8U2 – microcontroller for USB




Digital input and digital output pins with Pulse-Width Modulation (PWM)

In-Cirtuit Serial Programming (ICSP) header for microcontroller firmware / bootloader programming

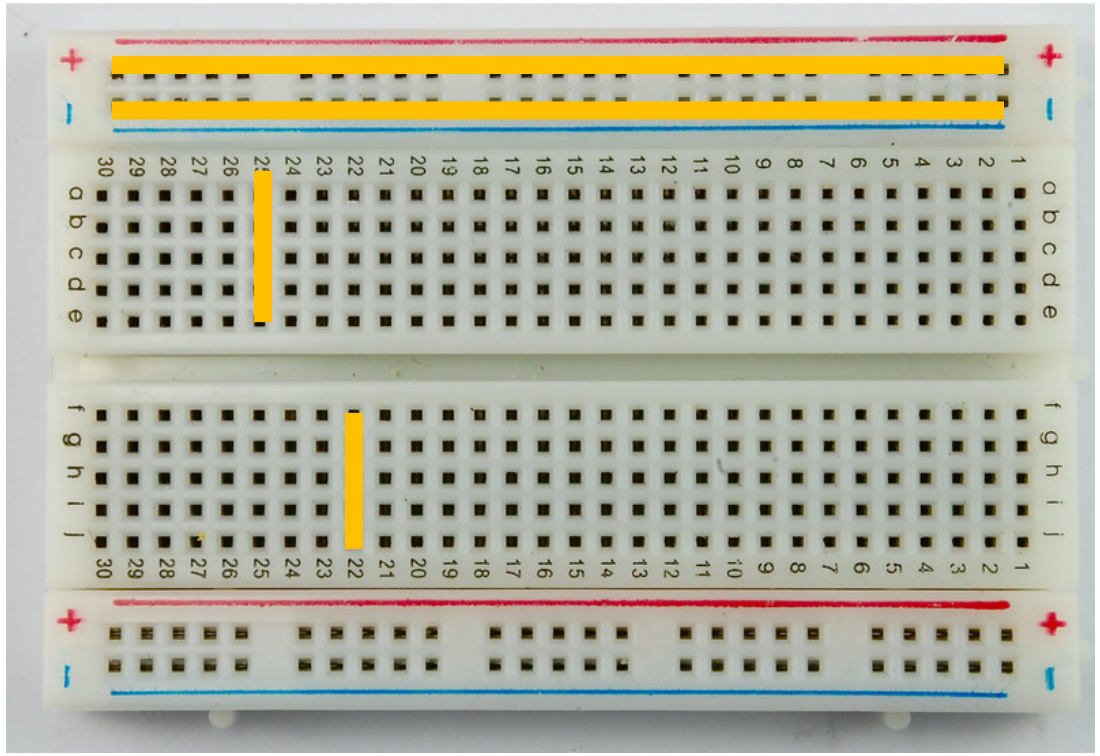
ATMega328 microcontroller
- Application code / software
- Boot loader and firmware
- USB programming interface

External power source

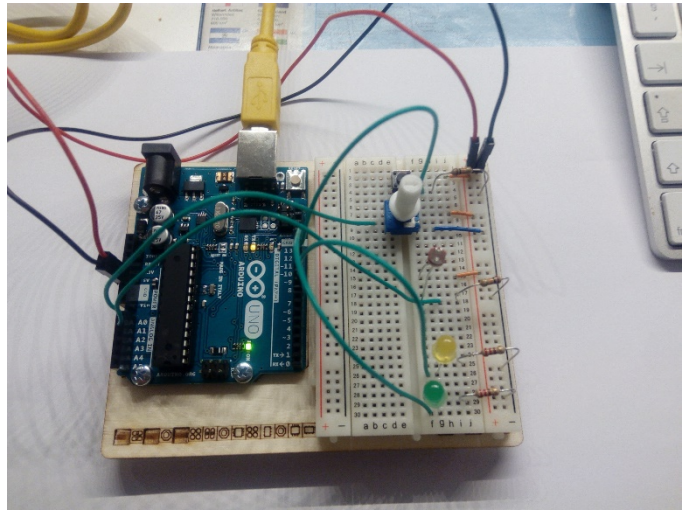
Analog input pins (A-D conversion)

# Breadboard

- **Connecting terminals of electrical components using non-permanent wiring (unlike soldering)**

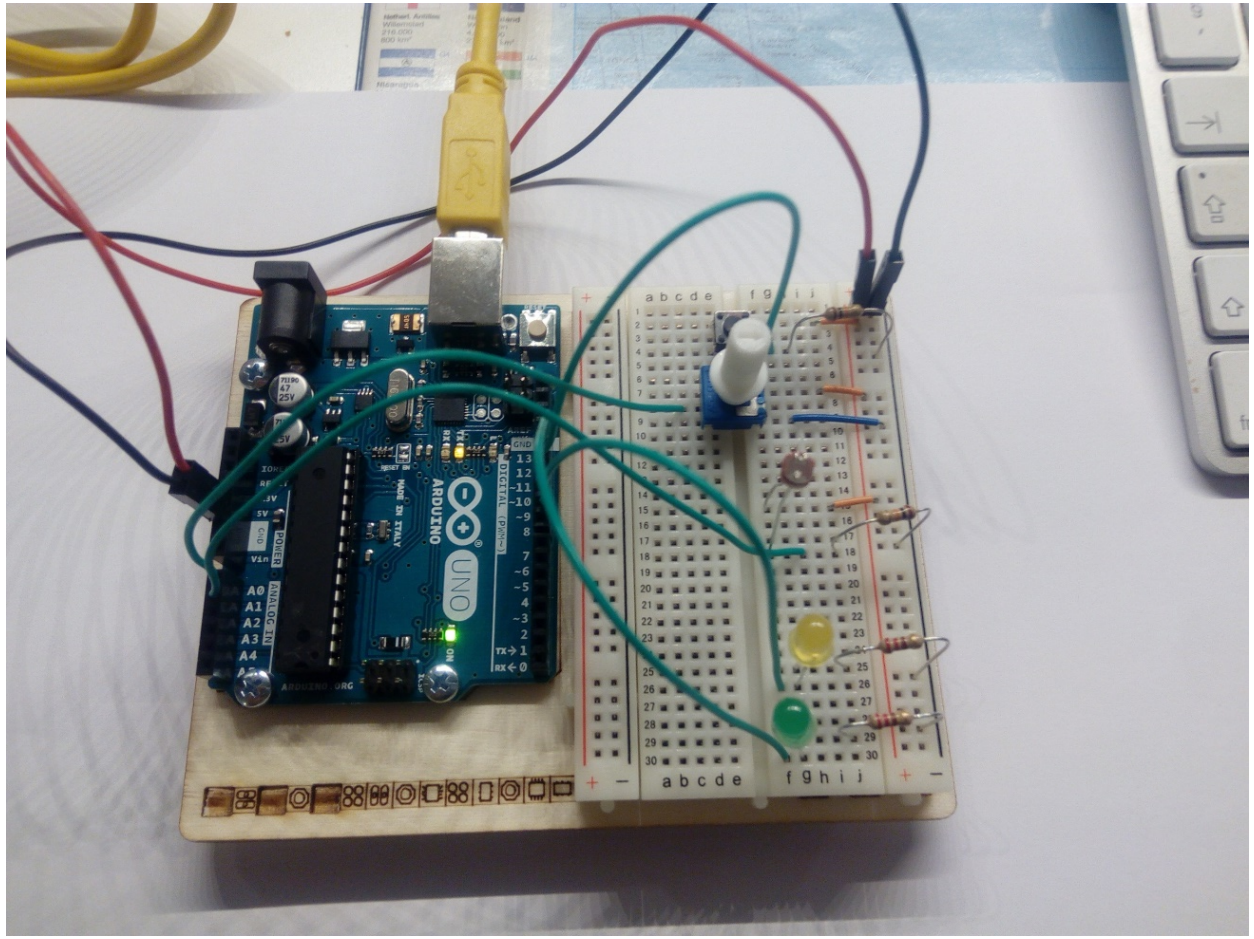


- **Used to connect sensors, actuators, and components to the microcontroller**



# Demo

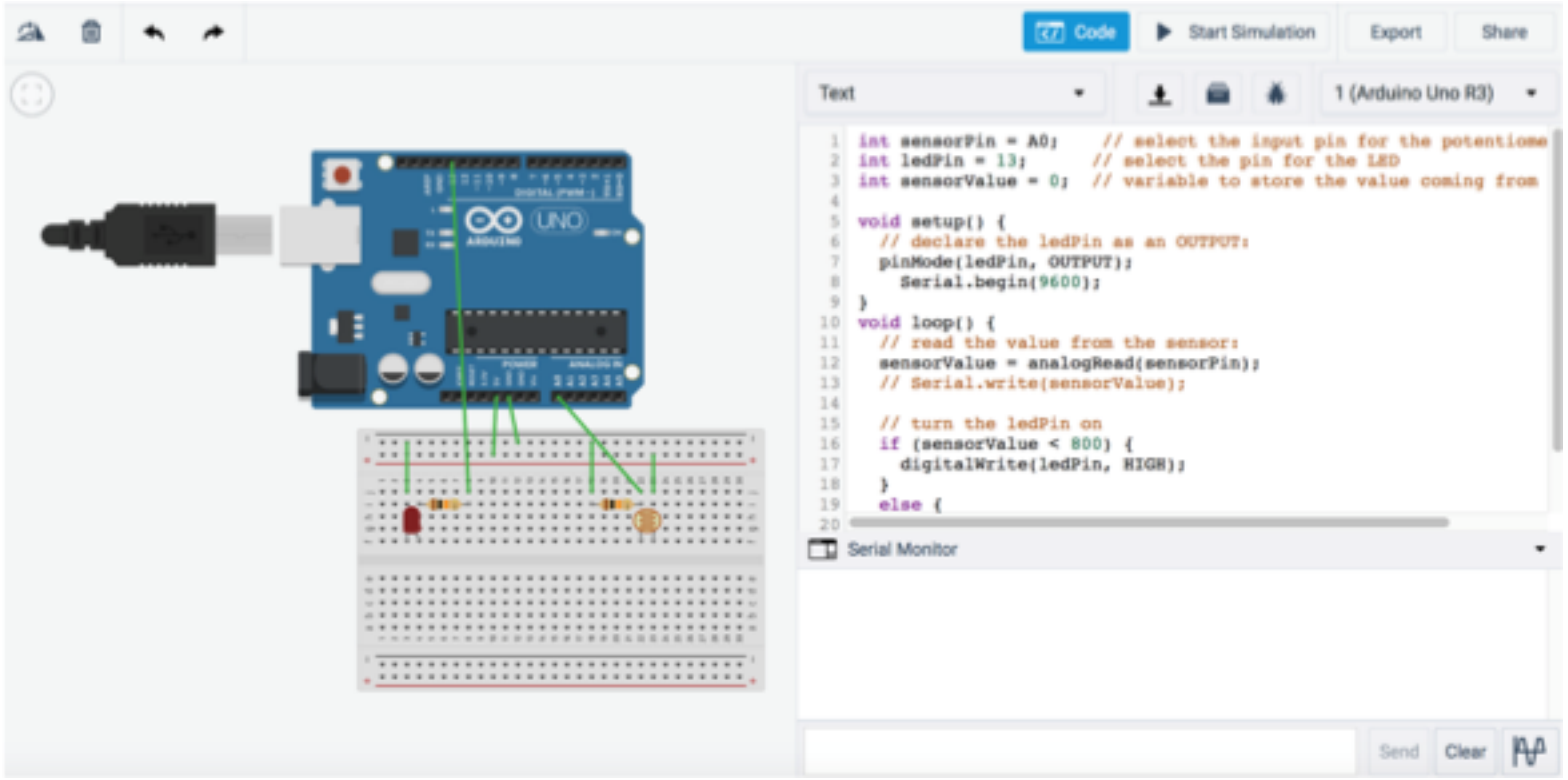
https://www.youtube.com/watch?v=1kW8clGpUJQ

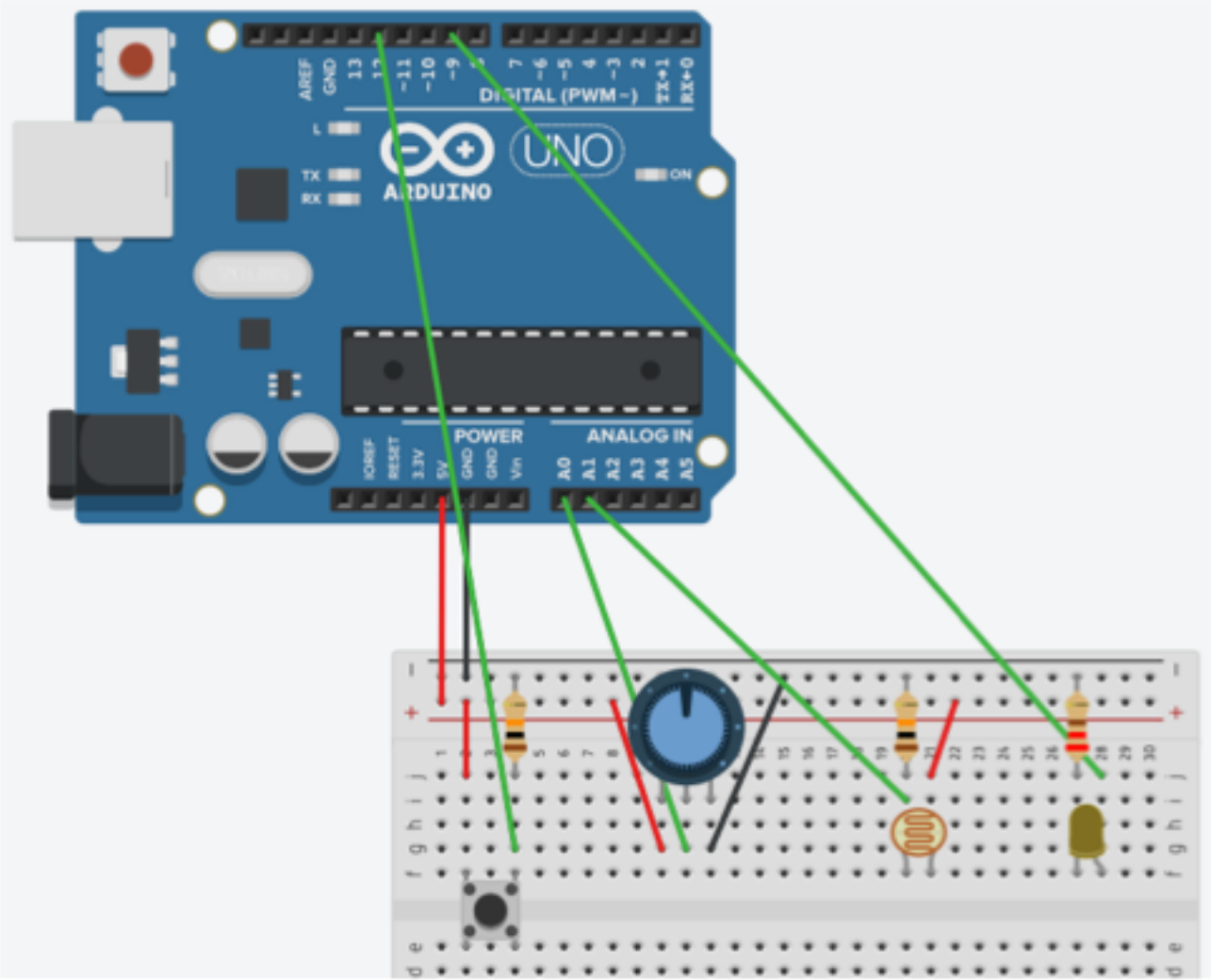



- A push button is used to activate / deactivate the system.
- An on-board / green LED indicate if the system is currently active.
- A lightsensor is used to sense light intensity.
- A potentiometer is used to set a threshold for turning on a yellow LED with a certain brightness.

# TinkerCad [ https://www.tinkercad.com/ ]

- **Web application for circuit design, simulation, and testing**





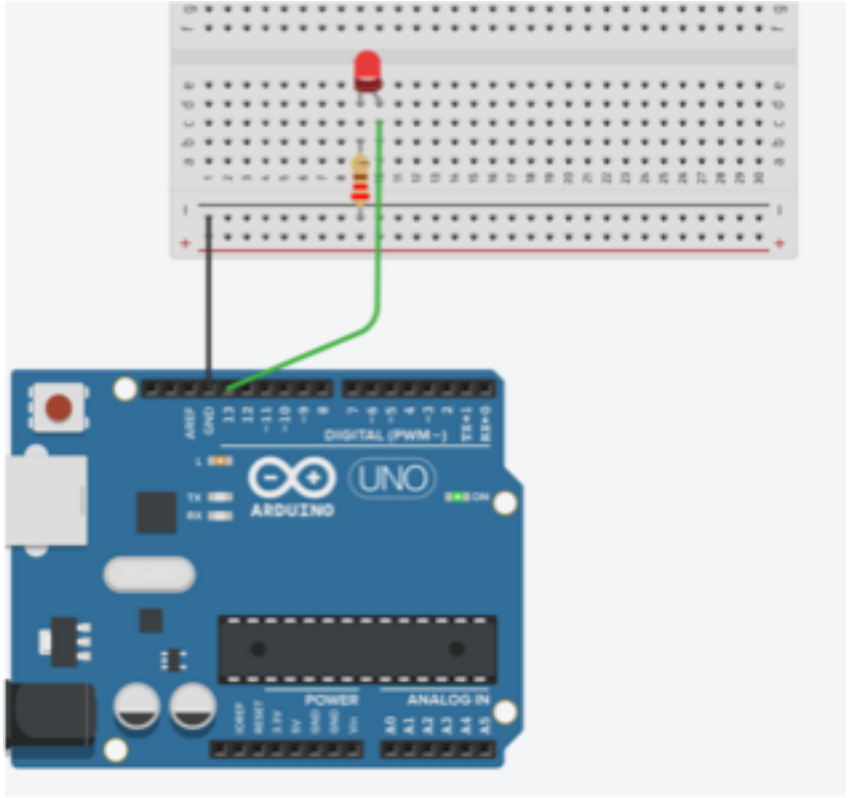

```
void setup()
{
  Serial.begin(9600);
  pinMode(12, INPUT);
  pinMode(13, OUTPUT);
  pinMode(9, OUTPUT);
}

int active = 0;
int pushhandled = 0;
byte brightness = 0;
byte debug = 0;

void loop()
{
  int btn = digitalRead(12);

  if ((btn == HIGH) && (!pushhandled)) {

    pushhandled = 1;
    active = 1 - active;

    if (active)
      digitalWrite(13, HIGH);
    else {
      digitalWrite(13, LOW);
      analogWrite(9, 0);
    }
  }
```

https://www.tinkercad.com/things/ki35AdXHvJz-dat159-lecture-1-example

# Demo | Exercise

- **Create a TinkerCAD account at [www.tinkercad.com](http://www.tinkercad.com)**

## A LED connect to pin 13




## Program to make the LED blink

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

# Programming model and sensor/actuator input/output

# Programming [ https://www.arduino.cc/reference/en ]

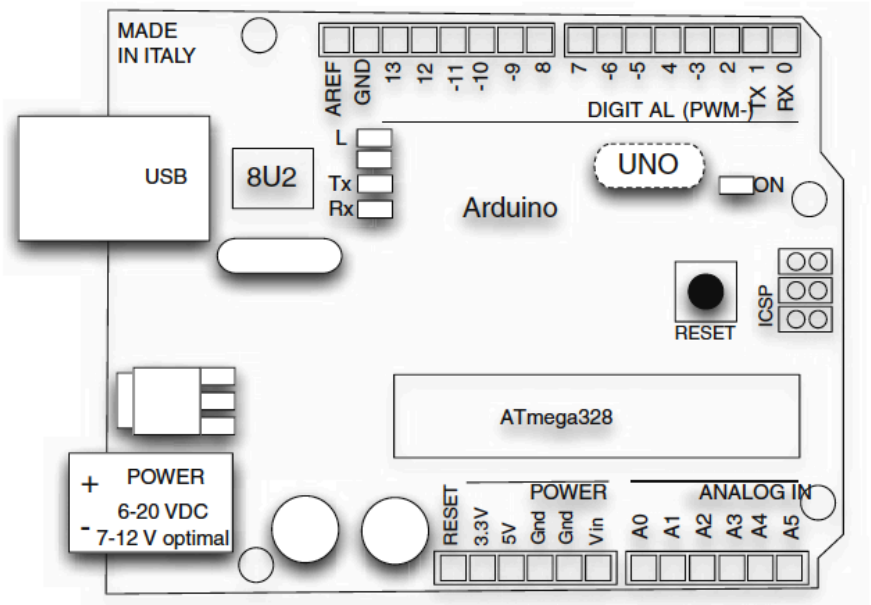
- **Arduino programs are called sketches**
  - C/C++ application code and functions
  - Libraries handling low-level microcontroller operations and interfacing - included using #include
- **Reactive programming model**
  - `void setup()` function executed once on power-up and used for initialisation of input/output pins, libraries,…
  - `void loop()` function executed repeatedly (infinite loop) - read input (sense), process and update, actuate (control).

# Low-level input/output

- **The input/output pins wired to the microcontroller are the basic connections to the outside world**
  - Pin voltages can be controlled by the program (output / actuators)
  - Pin voltages can be read by the program (input / sensors)
  - Pins can be designated as input pins or output pins
- **The physical world is analog (continuous) while the microcontroller is digital (discrete)**
  - Analog sensors requires analog-to-digital (A-D) conversion
  - Analog actuators requires digital-to-analog (D-A) conversion
  - **Examples:** photoresistor, termistor, flexresistor, servo motor, …
- **Digital sensors and actuators do not require conversion**
  - **Examples:** buttons, LED, …

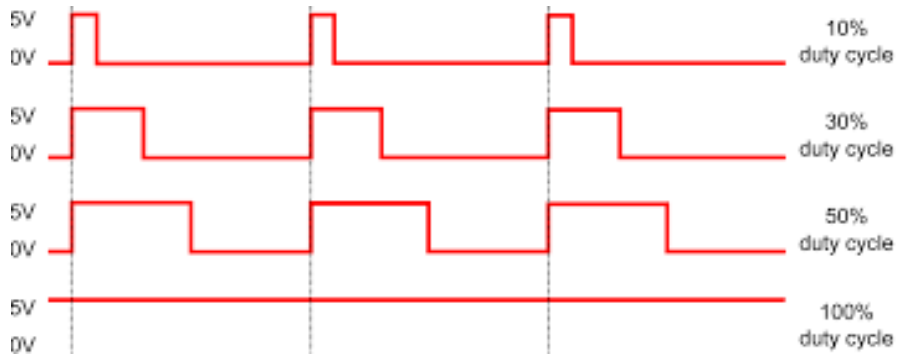
# Input and output pins

- **Pins 0-13 are digital input/ouput**
  - Read/write digital output
    5v = HIGH (1) | 0v = LOW (0)




- **Pins A1-A5 are analog input**
  - A-D converted to 0 .. 1023 (10 bit)

- **No analog output pins**
  - Pulse-Width Modulation (PWM) on digital output pins 3,5,6,10,11 (~) with 8-bits (0..255)
  - Duty cycle: percentage of time where the signal is high used to control perceived voltage




https://www.arduino.cc/en/Reference/Board

# Digital input and output

- **When a button (pin 12) is pushed turn on (off) a LED (pin 13) to indicate whether the system is active**

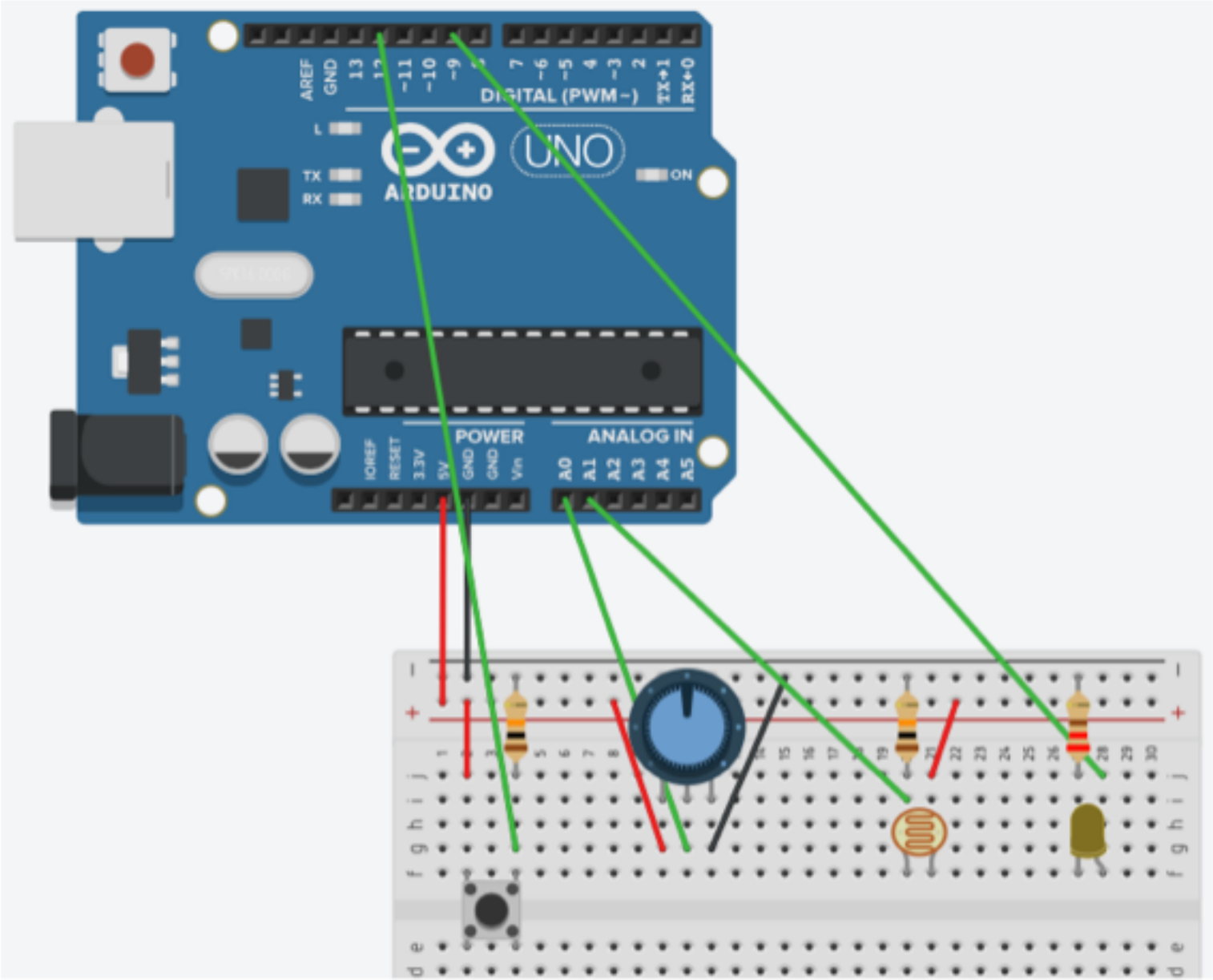
```
void setup()
{
  pinMode(12, INPUT);
  pinMode(13, OUTPUT);
}
```

```
int active = 0;
int pushhandled = 0;
```

```
void loop() {
  int btn = digitalRead(12);

  if ((btn == HIGH) && (!pushhandled)) {

    pushhandled = 1;
    active = 1 - active;

    if (active)
      digitalWrite(13, HIGH);
    else
      digitalWrite(13, LOW);
  }

  if (btn == LOW)
    pushhandled = 0;
}
```



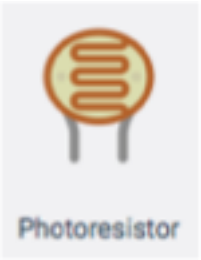
**cathode (-) | anode(+)**


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

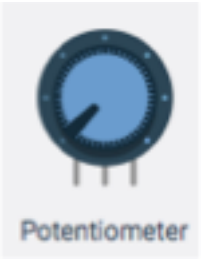
# Analog input

- **A photoresistor (pin A1) senses light intensity and a potentiometer (pin A0) determine (LED) brightness**

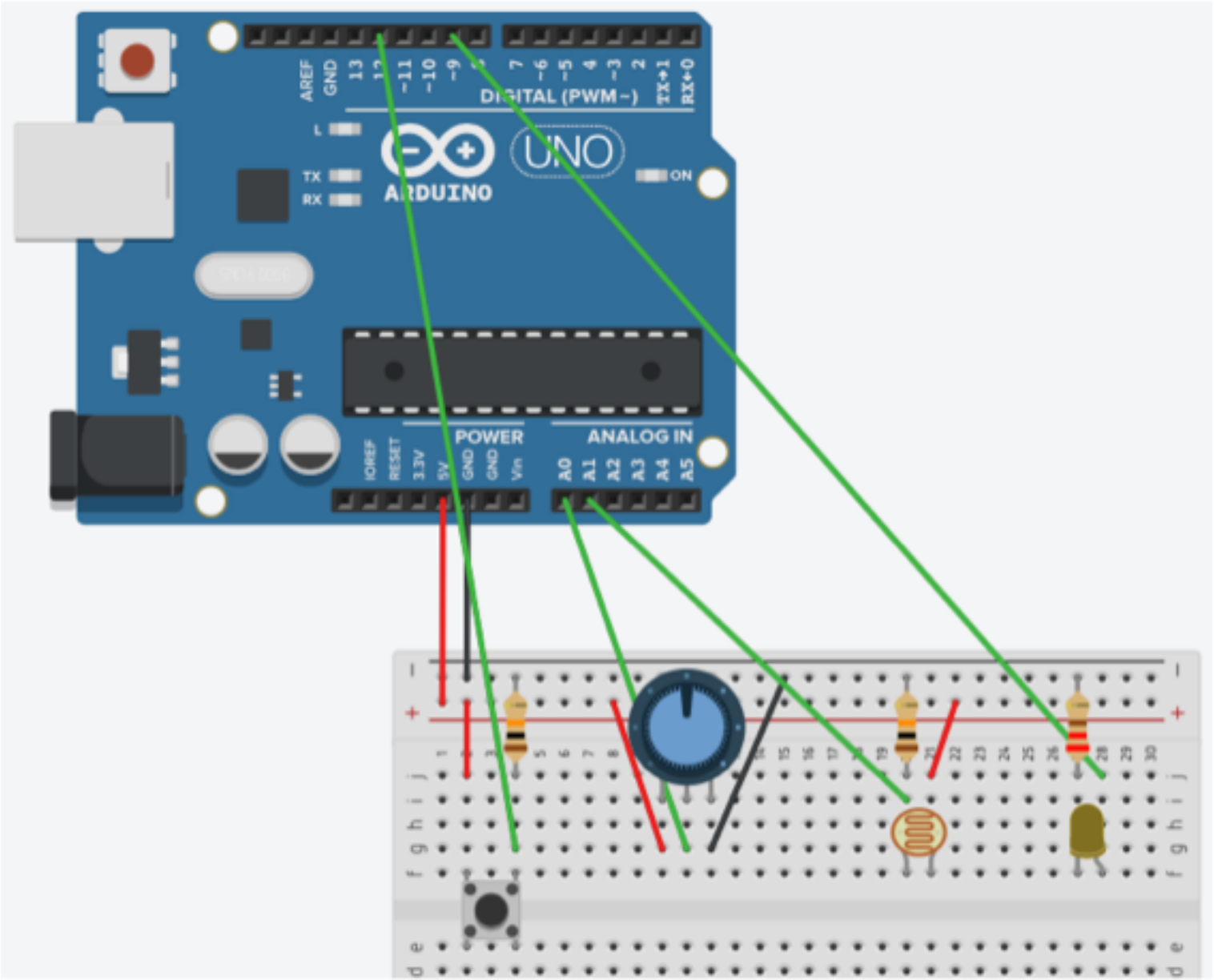
```
byte ledbrightness = 0;

int potval = analogRead(A0);
int photoval = analogRead(A1);

if ((photoval < potval) && active) {

  ledbrightness = min(potval - photoval, 255);

} else {

  ledbrightness = 0;
}
```




**variable resistor**




**voltage divider**

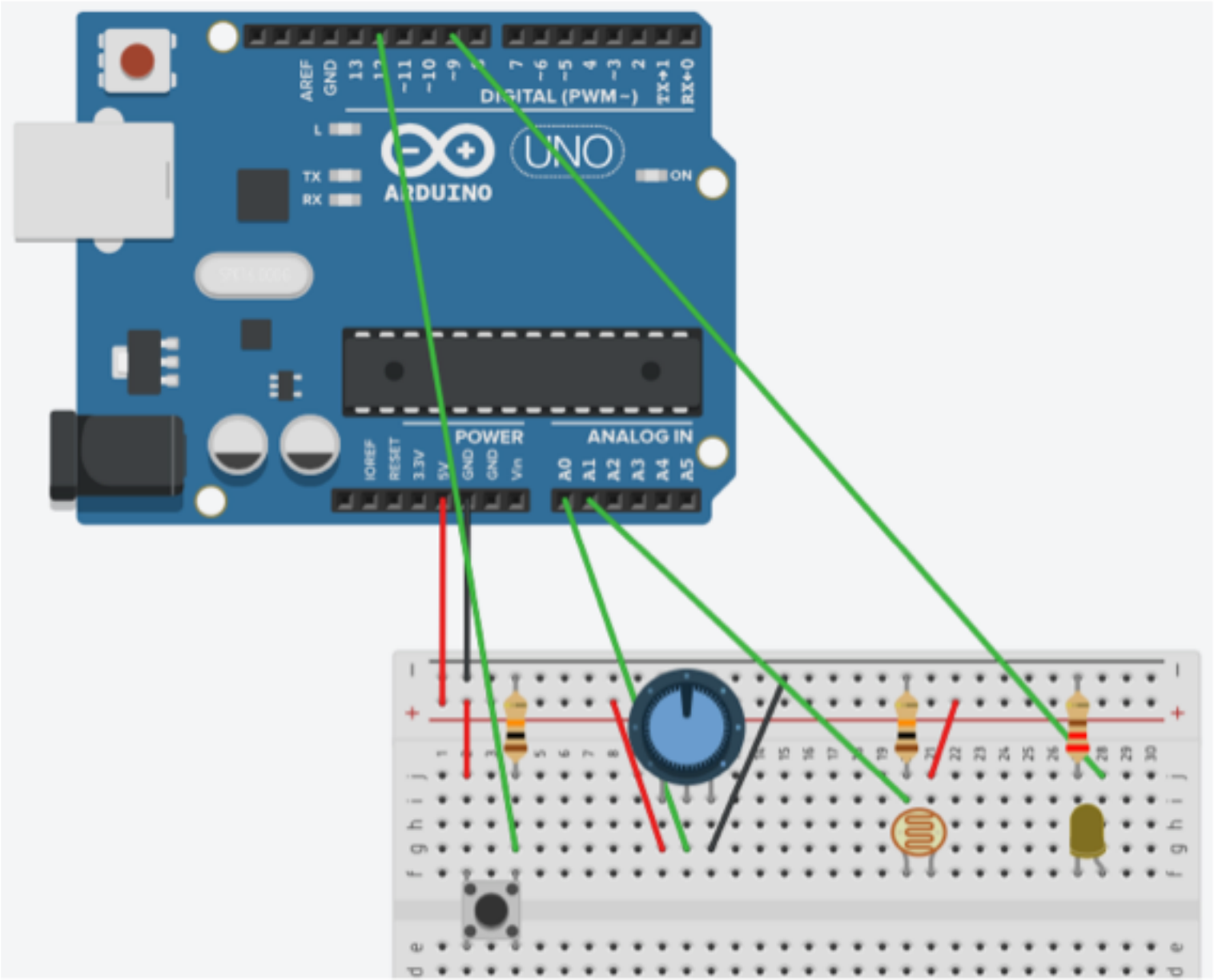

AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

# PWM - Analog output

- **Controlling brightness (8-bit) of an LED (pin 9) using Pulse-Width-Modulation (PWM)**

```
byte brightness = 0;

void setup()
{
  pinMode(12, INPUT);
  pinMode(13, OUTPUT);
  pinMode(9, OUTPUT);
}
```

```
int potval = analogRead(A0);
int photoval = analogRead(A1);

if ((photoval < potval) && active) {

  brightness = min(potval - photoval, 255);
  analogWrite(9, brightness);

} else {

  brightness = 0;
  analogWrite(9, 0);

}
```


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

# Higher-level input/output and interfacing

# Serial communication

- **Enables communication between the target platform (device) and other devices / components / controllers.**

- **Universal Asynchronous Receiver-Transmitter (UART)**
  - Designed for long-distance asynchronous communication (no shared clock)
  - Sequential bidirectional transfer of bytes with flow control
  - Low hardware overhead: 1 pin on each device / 1 bit transmitted at a time
  - Most microcontroller have built-in support for UART communication

- **Can also be used over USB for input/output and for debugging purposes in conjunction with a serial monitor.**

# Example: debugging

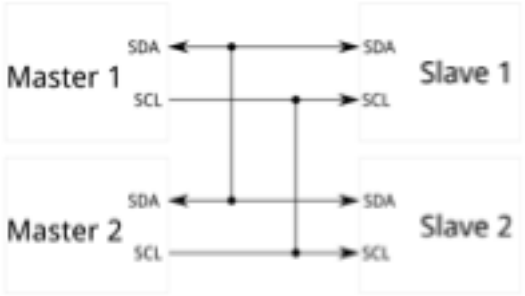
- **Configuring and controlling debugging output using serial communication**

```
void setup()
{
  Serial.begin(9600);
  pinMode(12, INPUT);
  pinMode(13, OUTPUT);
  pinMode(9, OUTPUT);
}

byte debug = 0;
```

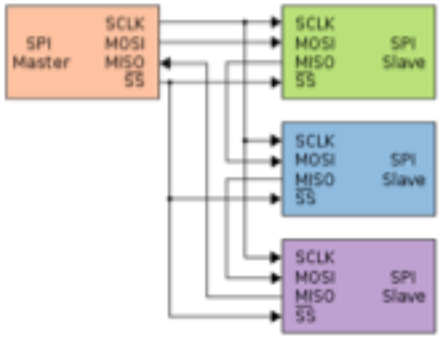
```
if (Serial.available() > 0) {

   byte input = Serial.read();
   debug = (input == 49);

}

if (debug) {
  Serial.print(photoval);
  Serial.print("<");
  Serial.print(potval);
  Serial.print(":");
  Serial.print(ledbrightness);
  Serial.println();
}
```

# Higher-level protocols

- Used for connecting microcontrollers and complex devices.
- Inter-Integrated Circuit (I2C)
  - Master/slave bus for IC-microcontroller communication
  - Masters coordinate the read/write bus transactions
  - Supported by the Wire library on the Arduino platform




- Serial Peripheral Interface (SPI)
  - Data bus for synchronous serial communication
  - Support by the Wire library on the Arduino platform




- Controller Area Network (CAN)
  - Automotive and industrial automation domain

# Tentative wrap-up ...

- **An IoT device is an embedded system characterised by**
  - Hardware: microcontroller, actuators, sensors, ...
  - Software: for control and computational intelligence
  - Internet connectivity: network controllers enabling the communication
- **The hardware and software is co-designed**
  - Requires knowledge of both the hardware and software capabilities
  - Devices are often real-time systems and are performing safety critical tasks
  - Often designed and developed for a cost-competitive market
- **Interaction with sensors and actuators**
  - Low-level interfacing: digital and analog input/output via microcontroller pins
  - Higher-level interfacing and protocols: UART, SPI, I2C,…
  - Software libraries used to hide low-level microcontroller and protocol details