



**DEEC**

DEPARTAMENTO DE ENGENHARIA  
ELETROTÉCNICA E DE COMPUTADORES  
TÉCNICO LISBOA

# Microcontrollers for IoT

An OpenLab sprint presentation on MCUs, I/O devices and communication

João Barreiros C. Rodrigues

DEEC-IST | HackerSchool

Filipe Nobre Piçarra

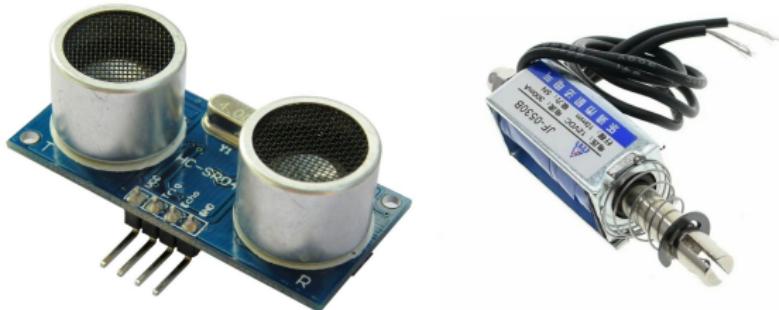
# Foundations for Micro-controlled Systems

# What is a Microcontroller?

- ❖ Small embedded computer
- ❖ Generically includes:
  - ❖ processor (CPU)
  - ❖ memory (RAM/Flash)
  - ❖ and I/O pins
- ❖ Normally the code runs on the baremetal
- ❖ Alternatively it can run a simple, deterministic OS
  - ❖ **FreeRTOS**

# What are Sensors and Actuators?

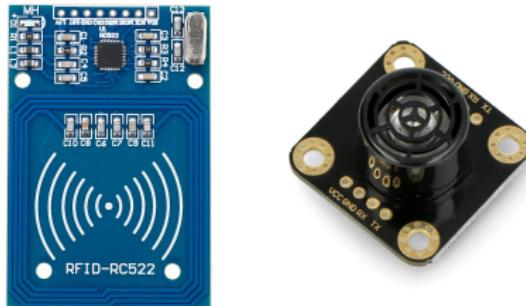
- ❖ **Sensors:** collect data from the environment
- ❖ **Actuators:** perform actions over the environment



# What are Sensors and Actuators?

These devices can communicate with the MCU through various protocols:

- General Purpose I/O (GPIO) and ADC I/O
- Inter-Integrated Circuit (I<sup>2</sup>C)
- Serial Peripheral Interface (SPI)
- Universal Asynchronous Receiver-Transmitter (UART)
- etc...



# **Our Microcontrolled System basics**

# ESP32 and ESP32 WROOM

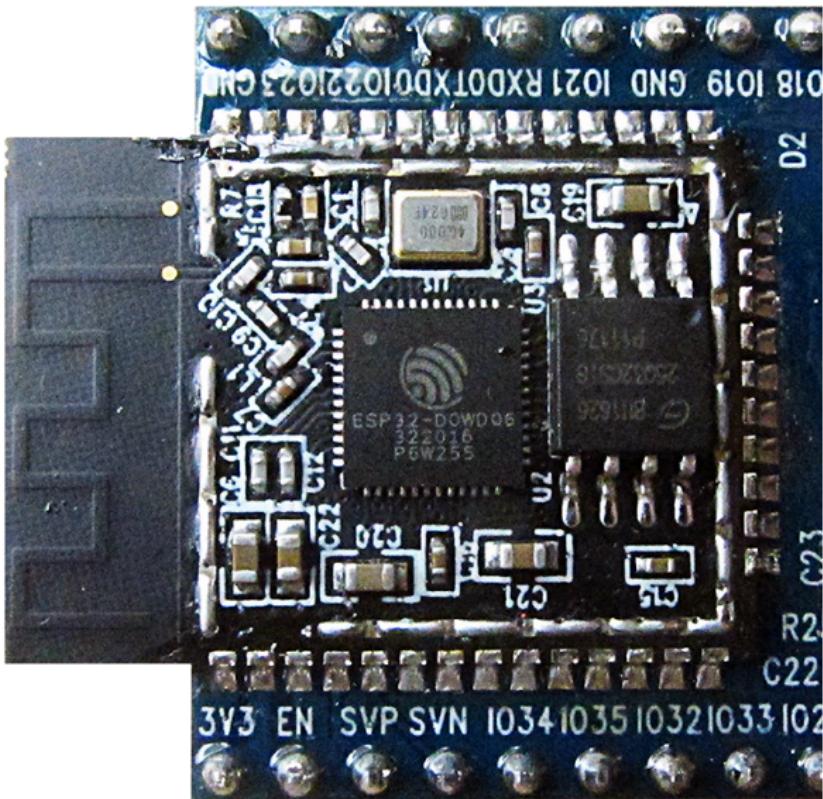
## ESP32

- Dual-core 32-bit LX6 microprocessor
- Ultra-Low-Power Co-Processor
- 520 kB of on-chip SRAM for data and instructions
  - Out of which 8 kB can be accessed in Deep-Sleep Mode
- 34 uncommitted GPIO pins
  - ADC, DAC, etc...

## ESP32 WROOM

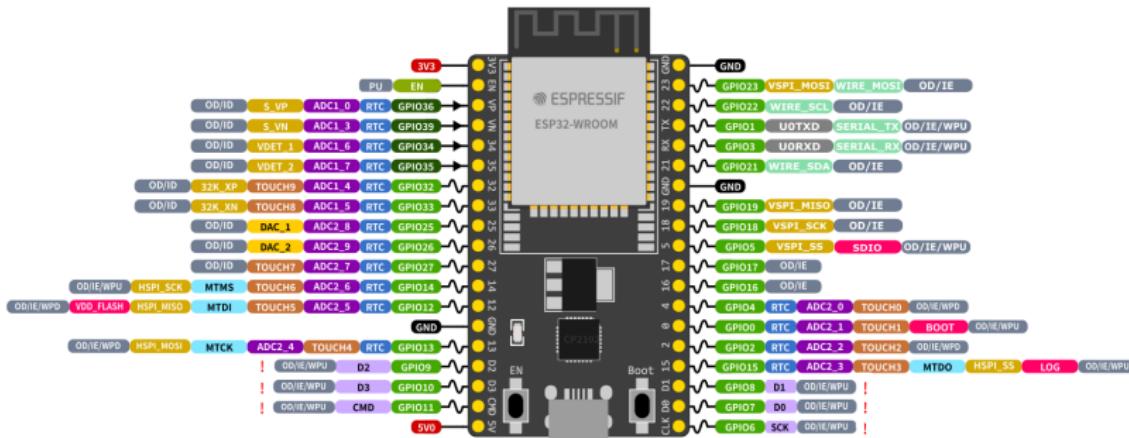
- Module that integrates a ESP32 chip
- Adds peripherals such as
  - Additional 4 MB of Flash Memory (PROGMEM)
  - Antennas
  - I/O shielding

# ESP32 and ESP32 WROOM



# ESP32-DevKitC4 / NodeMCU

## ESP32-DevKitC



### ESP32 Specs

32-bit Xtensa® dual-core @240 MHz

Wi-Fi IEEE 802.11b/g/n 2.4 GHz

Bluetooth 4.2 BR/EDR and BLE

520 KB SRAM (16 KB for cache)

448 KB ROM

34 GPIOs, 4x SPI, 3x UART, 2x I2C

2x I2S, RMT, LED PWM, 1 host eMMC/SDIO

1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

- PWM Capable Pin**: GPIO0, GPIO1, GPIO2, GPIO3, GPIO4, GPIO5, GPIO6, GPIO7, GPIO8, GPIO9, GPIO10, GPIO11, GPIO12, GPIO13, GPIO14, GPIO15, GPIO16, GPIO17, GPIO18, GPIO19, GPIO20, GPIO21, GPIO22, GPIO23.
- GPIO Input Only**: GPIO12, GPIO13, GPIO14, GPIO15, GPIO16, GPIO17, GPIO18, GPIO19, GPIO20, GPIO21, GPIO22, GPIO23.
- GPIO Input and Output**: GPIO0, GPIO1, GPIO2, GPIO3, GPIO4, GPIO5, GPIO6, GPIO7, GPIO8.
- Digital-to-Analog Converter**: DAC\_X.
- JTAG for Debugging**: DEBUG.
- External Flash Memory (SPI)**: FLASH.
- Analogue-to-Digital Converter**: ADC\_X.
- Touch Sensor Input Channel**: TOUCH\_X.
- Other Related Functions**: OTHER.
- Serial for Debug/Programming**: SERIAL.
- Arduino Related Functions**: ARDUINO.
- Strapping Pin Functions**: STRAP.

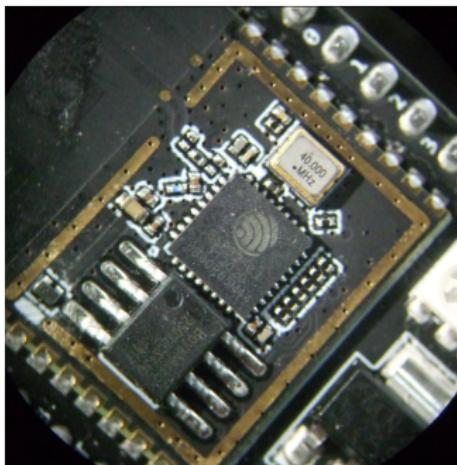
### GPIO STATE

- RTC**: RTC Power Domain (VDD3P3, RTC)
- GND**: Ground
- PWD**: Power Rails (3V and 5V)
- WPU**: Pin Shared with the Flash Memory
- WP**: Weak Pull-up (Internal)
- PU**: Pull-up (External)
- ID**: Input Enabled (After Reset)
- IE**: Input Disabled (After Reset)
- OE**: Output Enable (After Reset)
- OD**: Output Disabled (After Reset)

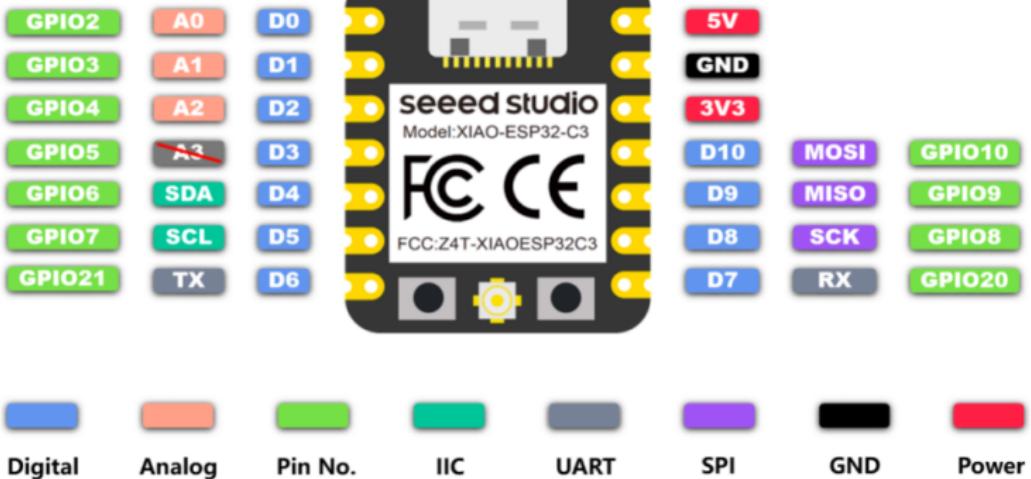
# ESP32-C3

## ESP32-C3

- RISC-V core
- Ultra-Low-Power
- 400kB of on-chip SRAM.
- Although with less pins, now supports BLE and on-chip encryption



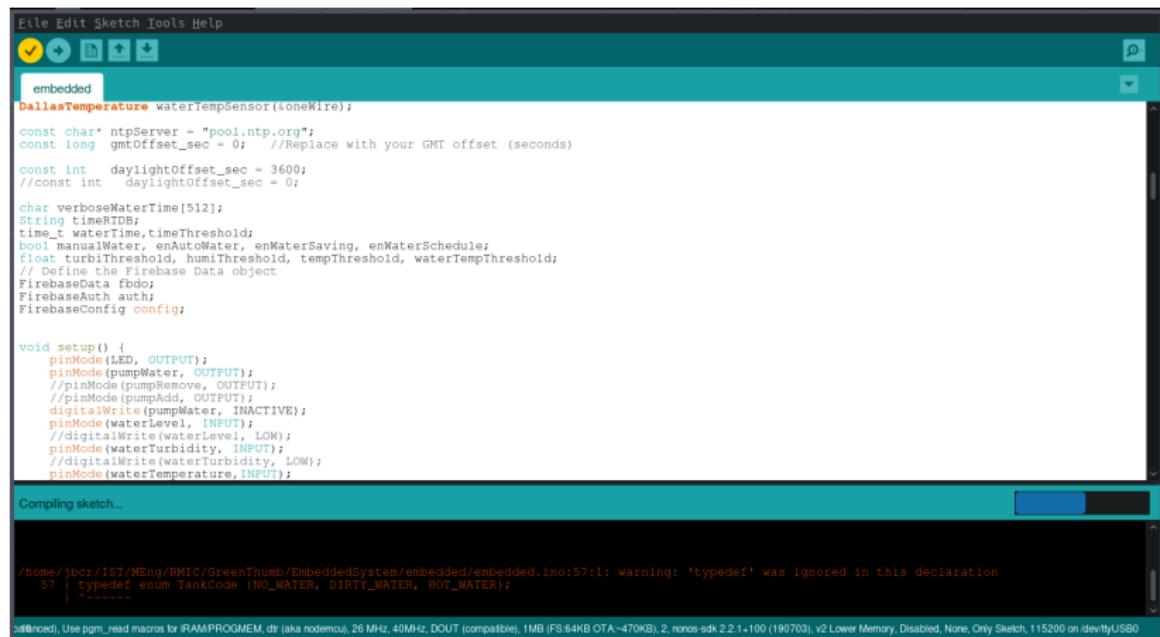
# XIAO ESP32-C3



\*A3(GPIO5) - Uses ADC2, which may become inoperative due to false sampling signals. For reliable analog reads, use ADC1 instead. Refer to the ESP32-C3 datasheet.

# Setup and Microcontroller programming

For ease of use we will be using **Arduino IDE**



The screenshot shows the Arduino IDE interface with a teal header bar. The menu bar includes File, Edit, Sketch, Tools, and Help. A toolbar with icons for file operations (New, Open, Save, Print, etc.) is visible. Below the toolbar, a tab labeled "embedded" is selected. The main code editor contains C++ code for a water temperature sensor. The code includes declarations for NTP server, time offset, and various sensor and control variables. It also initializes pins and sets up the Firebase Data object. The status bar at the bottom shows compilation and upload information.

```
File Edit Sketch Tools Help
File Edit Sketch Tools Help
embedded
DallasTemperature waterTempSensor(twoWire);

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 0; //Replace with your GMT offset (seconds)

const int daylightOffset_sec = 3600;
//const int daylightOffset_sec = 0;

char verboseWaterTime[512];
String timeRTDB;
time_t waterTime, timeThreshold;
bool manualWater, enAutoWater, enWaterSaving, enWaterSchedule;
float turbiThreshold, humiThreshold, tempThreshold, waterTempThreshold;
// Define the Firebase Data object
FirebaseData fdb;
FirebaseAuth auth;
FirebaseConfig config;

void setup() {
    pinMode(LED, OUTPUT);
    pinMode(pumpWater, OUTPUT);
    //pinMode(pumpRemove, OUTPUT);
    //pinMode(pumpAdd, OUTPUT);
    digitalWrite(pumpWater, INACTIVE);
    pinMode(waterLevel, INPUT);
    //digitalWrite(waterLevel, LOW);
    pinMode(waterTurbidity, INPUT);
    //digitalWrite(waterTurbidity, LOW);
    pinMode(waterTemperature, INPUT);
}

Compiling sketch...
/home/jbcr/IST/MEng/RNIC/GreenThumb/EmbeddedSystem/embedded/embedded.ino:57:1: warning: 'typedef' was ignored in this declaration
  57 |     typedef enum TankCode (NO_WATER, DIRTY_WATER, HOT_WATER);
      |     ^~~~~~

```

sd@niced] Use pgm\_read macros for iRAMPROGMEM, dr (aka nodemcu), 26 MHz, 40MHz, DOUT (compatible), 1MB (FS:84KB OTA~470KB), 2, nodemcudk 2.2.1+100 (190703), v2 Lower Memory, Disabled, None, Only Sketch, t15200 on /dev/ttyUSB0

# I/O and Wireless fundamentals

## Digital GPIO

```
digitalRead(PIN);  
digitalWrite(PIN, value);
```

## Analog GPIO

```
analogRead(PIN);  
analogWrite(PIN, value);
```

# More I/O

## I<sup>2</sup>C

```
#include <Wire.h>

Wire.begin(SDA_PIN, SCL_PIN);
Wire.write(data);
```

## UART

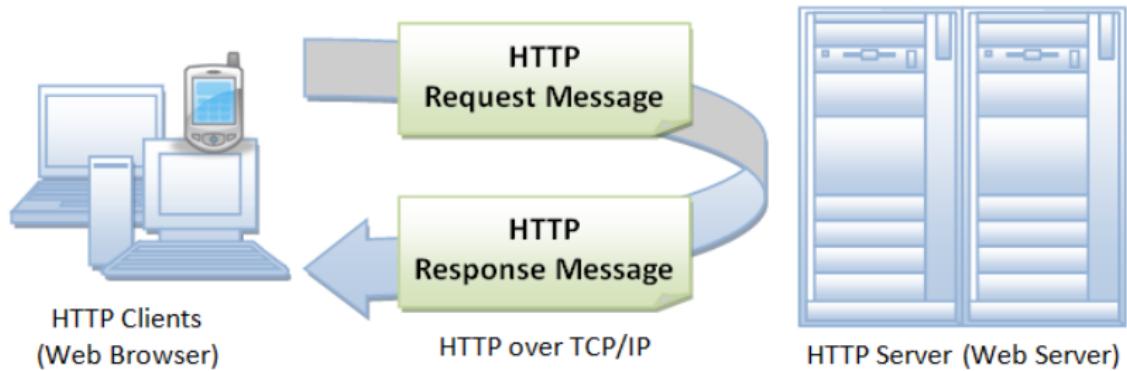
```
HardwareSerial mySerial(2); // Use UART2
mySerial.begin(BAUD_RATE, SERIAL_8N1, RXD_PIN, TXD_PIN);
```

# Integrating our sys. with the Internet

# Wi-Fi

```
#include <WiFi.h>
/*WiFi credentials*/
#define WIFI_SSID "Labs-LSD"
#define WIFI_PASSWORD "aulaslsd"
void setup(){
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    Serial.print("Connecting to WiFi");
    while(WiFi.status() != WL_CONNECTED){
        Serial.print(".");
        delay(500);
    }
    Serial.println("");
    Serial.println("Connected to WiFi!");
}
```

# HTTP



## HTTP Methods and Their Meaning

Method	Meaning
GET	Read data
POST	Insert data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

# HTTP POST 1

```
#include <HTTPClient.h>
/* HTTP endpoint*/
#define serverURL "http://chat.fpicarras.dev/send"
#define ID "<yourID>

void loop(){
    if(Serial.available()){
        String msg = Serial.readStringUntil('\n');
        msg.trim();
        if (msg.length() == 0) return;
```

# HTTP POST 2

```
HTTPClient http;
http.begin(serverURL);
http.addHeader("Content-Type",
               "application/json");
String body = "{\"message\":\"" + msg
               + "\", \"id\":\""
               + ID + "\"}";
int code = http.POST(body);
Serial.println("Sent: " + msg + " (" + code + ")");
http.end();
delay(1000); // prevent spamming
}/*end if*/
} /*end loop*/
```

# Power Management

# Active, Inactive and Sleep-Modes

Power mode	Description			Power Consumption	
Active (RF working)	Wi-Fi Tx packet			Please refer to Table 5-4 for details.	
	Wi-Fi/BT Tx packet				
	Wi-Fi/BT Rx and listening				
Modem-sleep	The CPU is powered up.	240 MHz <sup>*</sup>	Dual-core chip(s) Single-core chip(s)	30 mA ~ 68 mA N/A	
		160 MHz <sup>*</sup>	Dual-core chip(s) Single-core chip(s)	27 mA ~ 44 mA 27 mA ~ 34 mA	
		Normal speed: 80 MHz	Dual-core chip(s) Single-core chip(s)	20 mA ~ 31 mA 20 mA ~ 25 mA	
Light-sleep	-			0.8 mA	
Deep-sleep	The ULP coprocessor is powered up.			150 µA	
	ULP sensor-monitored pattern			100 µA @1% duty	
	RTC timer + RTC memory			10 µA	
Hibernation	RTC timer only			5 µA	
Power off	CHIP_PU is set to low level, the chip is powered down.			1 µA	

# Wi-Fi “sleep”

```
WiFi.setSleep(true);
```

This is the default behaviour! For intermittent connection to Wi-Fi we can set:

```
WiFi.setSleep(false);
```

which has an increased power consumption

# Wi-Fi disable (Modem-sleep)

```
WiFi.disconnect(true); // Turn off Wi-Fi  
WiFi.mode(WIFI_OFF); // Disable Wi-Fi
```

Requires the WiFi object to be restarted!!

# Light-Sleep

```
#include "esp_sleep.h"
// GPIO for button wake-up
#define WAKEUP_PIN GPIO_NUM_33
// Change to your actual button GPIO

// Configure RTC GPIO for wake-up
esp_sleep_enable_ext0_wakeup(WAKEUP_PIN, LOW);
// Wake on button press (LOW)
Serial.println("Going to deep sleep...");
delay(1000);
// Enter light sleep mode
esp_light_sleep_start();
```

# Deep-Sleep

```
#include "esp_sleep.h"
// GPIO for button wake-up
#define WAKEUP_PIN GPIO_NUM_33
// Change to your actual button GPIO

// Configure RTC GPIO for wake-up
// Wake on button press (LOW)
esp_sleep_enable_ext0_wakeup(WAKEUP_PIN, LOW);
Serial.println("Going to deep sleep...");
delay(1000);
// Enter deep sleep mode
esp_deep_sleep_start();
```

# To Finish our code...

Colocar na secção loop:

```
if (msg == "sleep") {  
    Serial.println("Sleeping for 10 seconds...");  
    WiFi.disconnect(true); // disconnect from WiFi and close all connections  
    WiFi.mode(WIFI_OFF); // turn off WiFi radio  
    esp_sleep_enable_timer_wakeup(10 * 1000000);  
    esp_deep_sleep_start();  
}
```

# Concluding Remarks

# Additional topics of interest

- ❖ Communication Security
  - ❖ SSL/TLS Encryption
- ❖ AI on the Edge
  - ❖ TinyML