M1 : IOT (Internet of Things)

Gilles Menez

Université de Nice – Sophia-Antipolis
Département d'Informatique
email : menez@unice.fr
www : www.i3s.unice.fr/~menez

October 12, 2023: V 1.0

Table of Contents

Syllabus :

Internet of Things (IoT) / Internet des Objets (IdO)

- **Aims/Objectifs pédagogiques** :

  From a software design point of view, understand what is going on IoT:

  - ✔ definitions ?
  - ✔ issues ?
  - ✔ current solutions ?
  - ✔ candidates technologies ?
  - ✔ ...

- **Prerequisite Skills/Prérequis** :

  Internetworking, C Language, Web (Http/Html), JS, Python ...

No theory but ... an intensive **programming** teaching unit !

Evaluation

Weights : Practical courses and/or Project (75%) + CT (25%)

CT : Test of 1h00 . . . few questions on IoT with NO document ALLOWED !

For the second part of the mark :

- ➢ Practical courses evaluations : bigger weight on the last one !

Why IoT is emerging ?

The same scenario . . . again ?

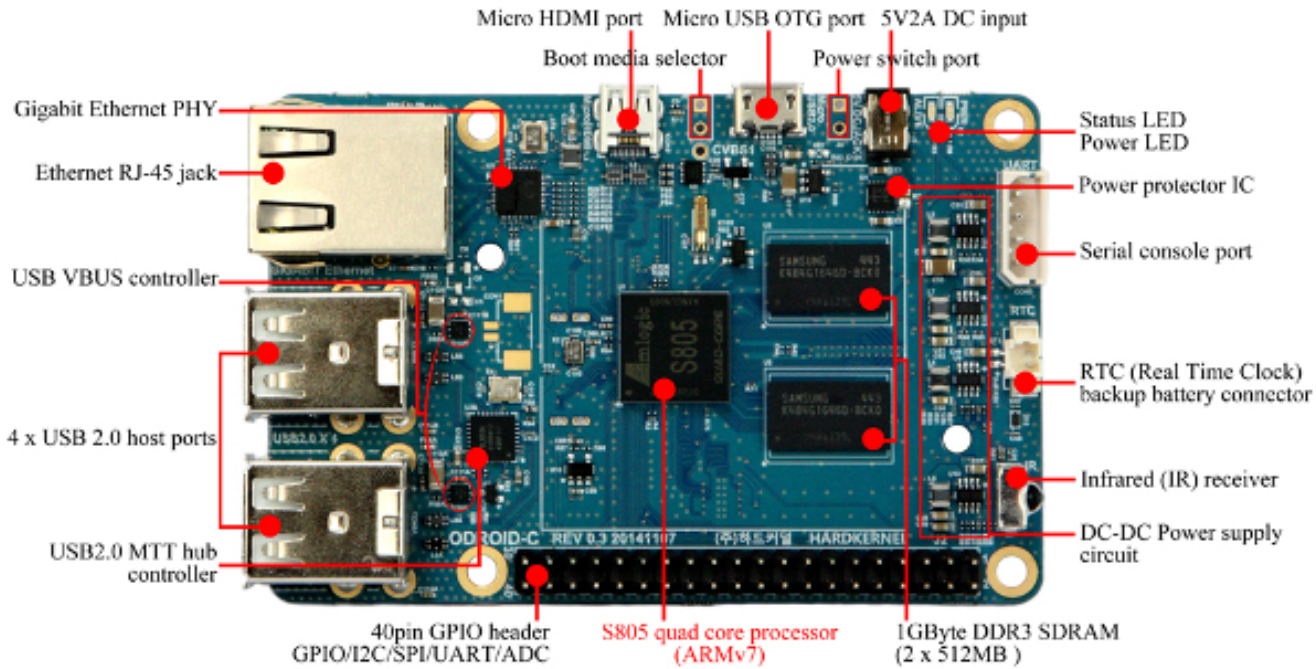
As technologies become matures,

- **Computing Hardware** :
  Microprocessors and Microcontrollers
- **Telecommunications** :
  Ground, Mobile, Satellite
- **Networking** :
  Infrastructures and Protocols
- **Software** :
  Web Techno, Cloud, Programming Languages and Execution Schemes
- **Digital uses (/ Les usages)** :
  More and more in every day life.

a synergy often operates and **a new domain is emerging** !

Hardware is ready

"All in one" module (here Odroid C1) can compute, send/receive (radio/grounded), and control (sensors/actuators) their environment ...



A PC in a 10*cm*x5*cm* board **for (very, very, very, ...) few money** !

➢ "Embedded hardware" ... for ubiquitous systems !

Telecommunications and Networks

- Evolutions of WiFi and Bluetooth standards,
- Cellular 5G,
- Satellites Starlink,
- Internet V6,
-
- and many more physical networks . . . LoRa, SigFox, Zigbee,

. . . are candidates to the deployment of IoT.

Software technologies

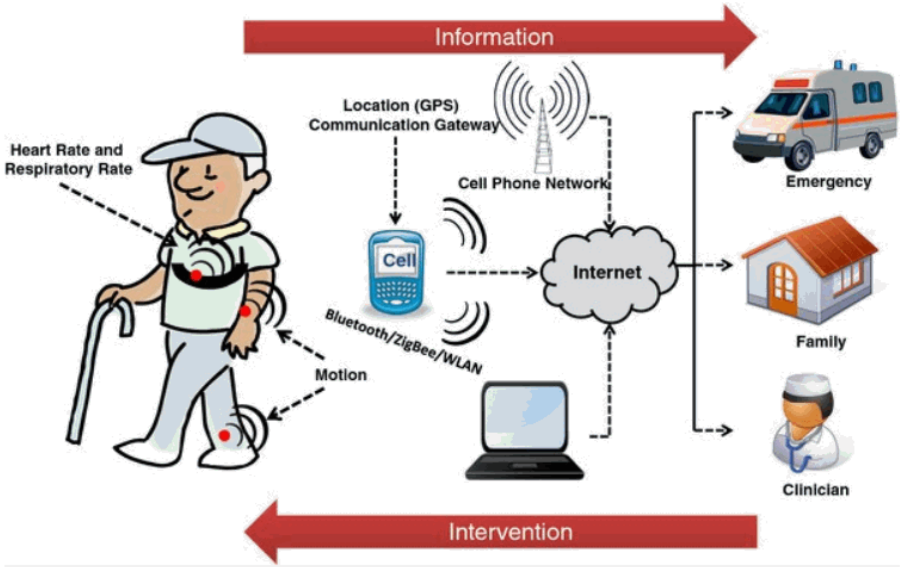
Machines, as execution plateforms, are no more "only in a box" !

The cloud (AWS, Azure, ...) and the navigators (Firefox, Chrome, ...) are candidates to replacement !

- You can execute a program in so many "machines".

Uses

Technologies seems to be ready but . . . what about uses and users ?



IoT in Remote Health Monitoring

Users are largely adopting digital services and uses are today actual in health, fitness, energy, sports, . . .

➢ Successfull uses of digital technologies and products must/will **carry on and amplify** ! . . . even if we should think about that ?!

So . . . easy ? no issues . . . really ?

"The Canadian-based firm, called Kepler Communications, is scheduled for space on two SpaceX Falcon 9 launches to deploy 140 "dedicated" satellites for the "Internet of Things" (IoT), to connect devices ranging from kitchen appliances to shipping containers to networks.(see more)

Can you imagine what it implies ? The number of things, the size of networks, the global power, . . .

Do "current"/existing technologies will do the job ?

More and more !

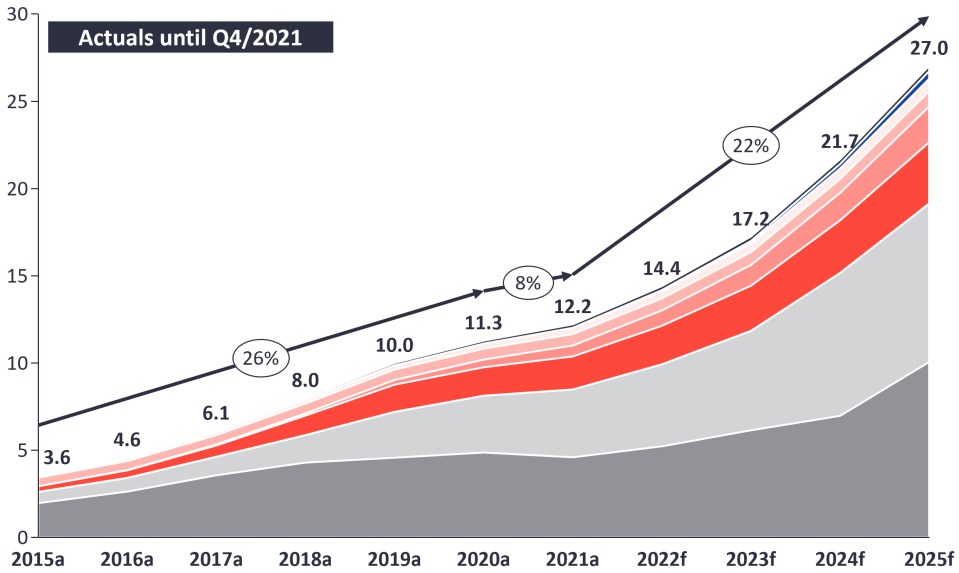
IOT ANALYTICS

May 2022

Your Global IoT Market Research Partner

Global IoT Market Forecast [in billion connected IoT devices]

Number of global active IoT Connections (installed base) in Bn



CONNECTIVITY TYPE	CAGR 20-21	CAGR 21-25
Wireless Neighborhood Area Networks (WNAN)	17%	11%
5G IoT	-	159%
Other	22%	20%
Wired IoT	4%	7%
LPWA	42%	34%
Legacy Cellular (2G/3G/4G)	16%	17%
Wireless Local Area Networks (WLAN)	19%	24%
Wireless Personal Area Networks (WPAN)	-6%	22%

XX% = CAGR

Note: IoT Connections do not include any computers, laptops, fixed phones, cellphones or tablets. Counted are active nodes/devices or gateways that concentrate the end-sensors, not every sensor/actuator. Simple one-directional communications technology not considered (e.g., RFID, NFC). Wired includes Ethernet and Fieldbuses (e.g., connected industrial PLCs or I/O modules); Cellular includes 2G, 3G, 4G; LPWAN includes unlicensed and licensed low-power networks; WPAN includes Bluetooth, Zigbee, Z-Wave or similar; WLAN includes Wi-fi and related protocols; WNAN includes non-short range mesh, such as Wi-SUN; Other includes satellite and unclassified proprietary networks with any range.

Source: IoT Analytics Research 2022. We welcome republishing of images but ask for source citation with a link to the original post and company website.

https://iot-analytics.com/number-connected-iot-devices/

Breaks !

For "Information Technology" sciences, IoT marks many breaks with "potting soil" technologies :

- Hardware :

  Energy, Costs, # of Things, ...
- Telecommunications :

  mobility, low power, latency, speed, global volume of data, bandwidth differentations, ...
- Networking :

  size (addressing), mesh topologies, low consumption protocols
- Software : what can we do with SO many data !?

  storage ?

  dynamic Techno, web techno or else ?

  edge or cloud computing ?

  and **machine learning/data mining/analytics**

Beyond IoT ? . . . Data !

IoT is a **new step on the path of a digital world**.

For example, if we focus on networking some significatives changes will occur (soon):

- before ADSL, networking was stopping at offices or companies.
- with ADSL, networking began to be ubiquitous, and entered our home and reach our computer in our desk.
- with IoT, networking will raise in our kitchen, in our bathroom, . . . and even on our skin.

  Digital has never been so close from users.

For creating services for our wellness (. . . or not ?!) the "quest of Data" has already begun and IoT will play a fundamental role in this future.

Let's control the process !

This course will consider that "software" is THE "pilot" in the car.

- ➢ We will study different "**software**" contributions to the deployment of IoT.

In this context, **we will see some proposals for questions and issues** :

- ➢ What are these "Things" ? how do they work ? and how do we program ?
- ➢ Is the "Internet" of IoT the same than the Internet we use every day ?
- ➢ What are "dedicated" application protocols used to link Things to Internet ?
- ➢ What could be architectures of software IoT platforms ?
  Cisco proposed a 11 tiers model !?
- ➢ Are Web technologies can be answers ?
- ➢ Can we deploy our own architecture ?
- ➢ Did we make a mess ? Do we need more ? Is there a standard to drive the car ?
- ➢ ...

IOT : Internet of Things

➢ What are these Things/"Objets" ?

This section will support first practical courses and ESP32 development module discovering ;-)

The aim is to better know these "Things" and their capabilities.

- Hence, we will understand why and how a new world can occur ?

First definition of IoT

In IoT, physical things connected by a network will **take part in Internet activities, sending and receiving information to/from their environment** :

So, IoT is a concept in which **everyday things** (washing machines, shoes, toothbrush . . .) **ARE "online", participating to a more global service**.

So a "thing" is **embedding computation and communications capabilities** !

DIY Driven

For future discussions, it is important to notice that IoT was and still is strongly pushed by a "do it yourself" (DIY) approach.

- ➢ Developers will implement codesigned hardware/software solutions to answer functional specifications.

Don't loose your rhino !

DIY limits !

This "world" of DIY design/solutions is getting visible and easier to access because of low cost, but high performances, of hardware available : chips, sensors, networks , . . .

Such components allow to build intelligent systems able to input/output (ADC/DAC), then to process data and finally to communicate through any type of channel :

- ➢ Domotic,
- ➢ Automatic gardening,
- ➢ Connected Meteo station,
- ➢ Energy Monitoring, . . .

For many of them (developers) it is a hobby and the environment of their solution is often a "personal horizon" or "adhoc".

As a consequence, solutions are not necessarly supposed to propagate through the planet.

Internet . . . and the paradigm changed !

And then, Internet came in this world of DIY things, allowing to link all of them and servers !

DIY solutions now can have a Worldwide audience and participate to ambitious tools and services.

An industry is born **BUT specifications** and **scopes** of an IoT industry are DIFFERENTS from DIY !

- "Data analytics" as a product

  Billions of data !

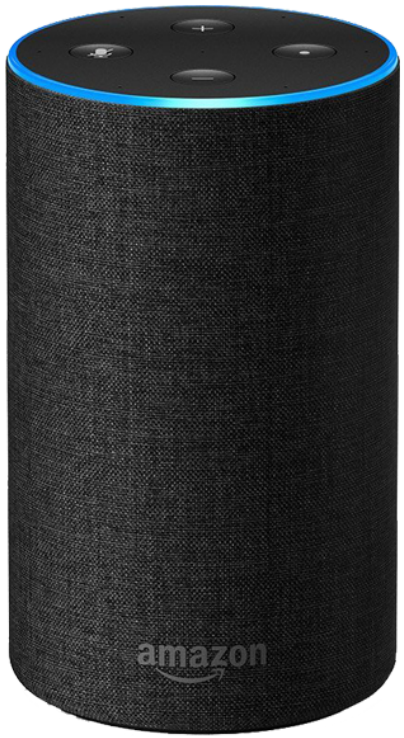
- Interoperability

  But still no true standard in this "jungle" !

Commercial "Things"

In 2020, IoT is not only a hobby for geeks in their kitchen and many major industries develop at least proof of concepts but also true products.

"Things" are declined as shoes, watches, glasses, speakers, ...



➢ Alexa (Amazon) / Google assistant

These wearable devices and gadgets are clearly components of IoT hardware domain :

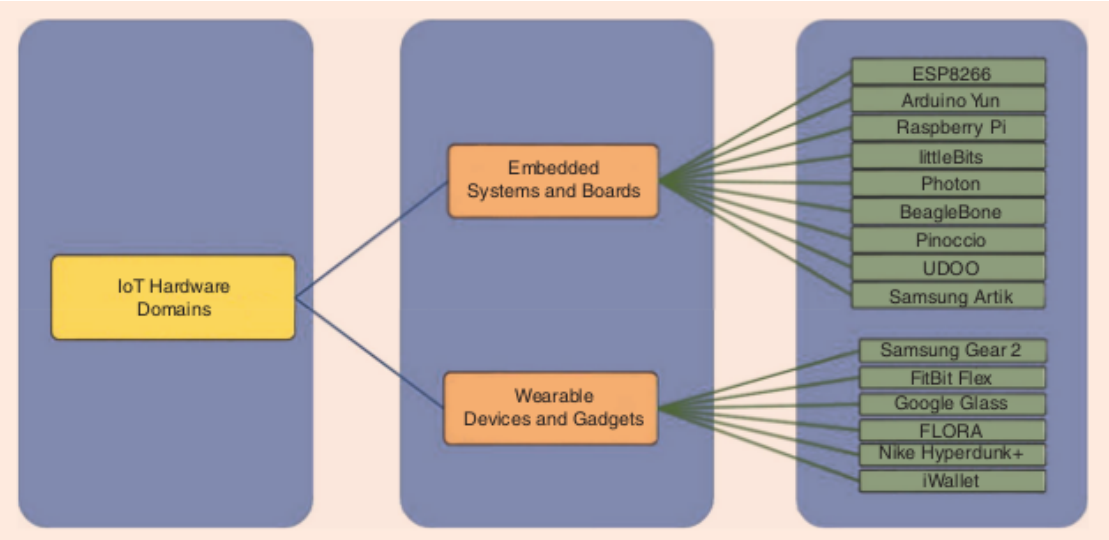
https://www.apple.com/fr/watch/

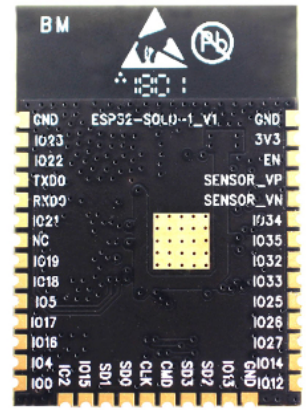
➢ . . . but most the time they are "closed" technologies.

Security ? Business policies ?

Embedded systems

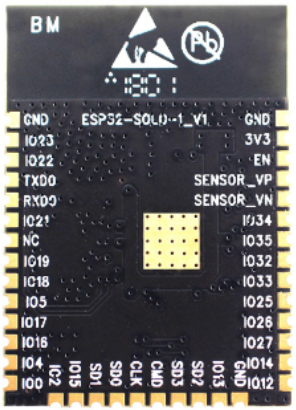
Beside these "commercial things", you can still find "raw material" waiting to be embedded in a more complex system or integrated in a commercial solution/product :





- ➢ They are cheap and small.
- ➢ They have low consumption modes, real world interfaces and communication ready links.

They have all the qualities to become the "Things" of IoT ...
and they will be !



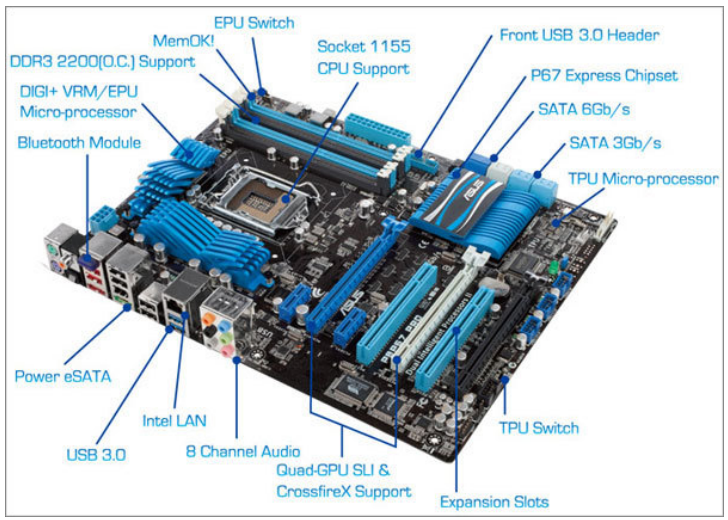
A Shelly "smartplug" powered by a ESP32 (by Espressif).

Microprocessor vs Microcontroller

The microprocessor you will find in your desktop or laptop is a powerful piece of silicium dedicated to computation.

- ✔ But, . . . it is not supposed to manage peripherals as mouse, keyboard, memory banks etc.

  Even less, it would NOT be suited to drive any kind of sensors (temperature, . . .). Would you attach a trailer to a Ferrari ?

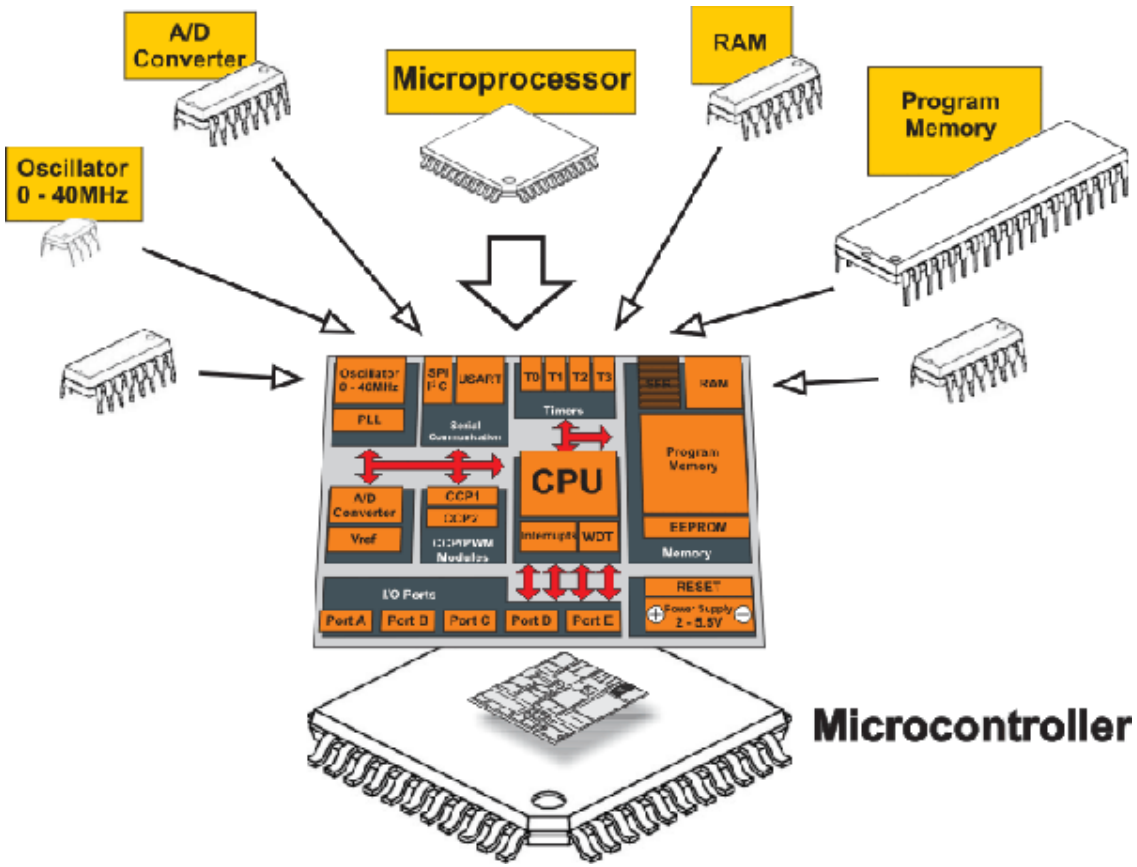


- ✔ This is the "engine" you will put in a frame (mother board) which is managing all environment the microprocessor do not manage :

  => together they are a machine !

Microcontroller

A microcontroller holds ALL functionalities of a machine on a chip :

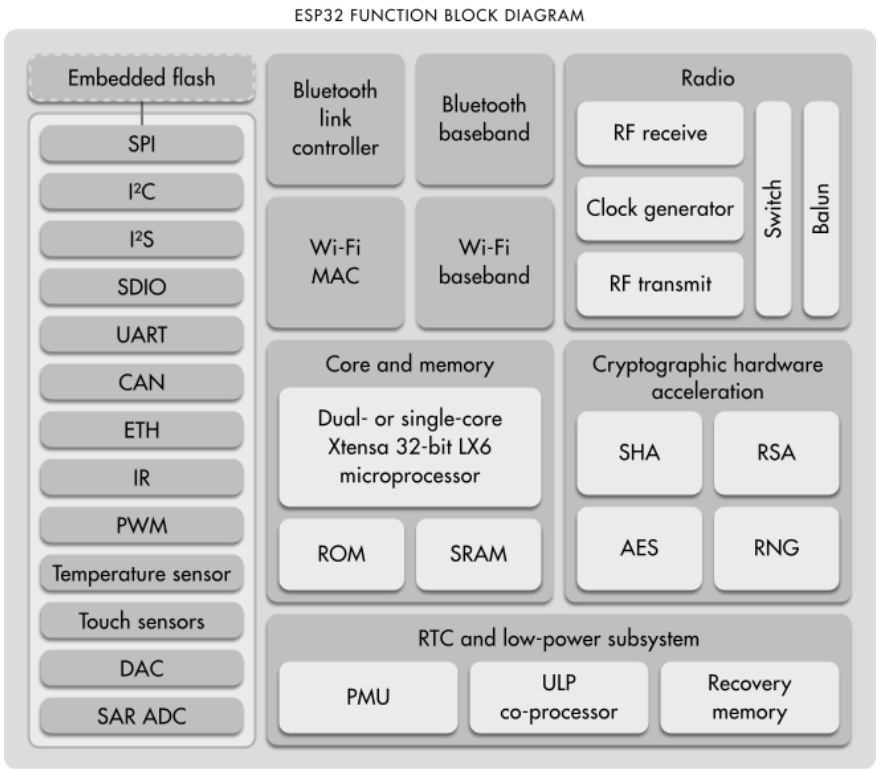


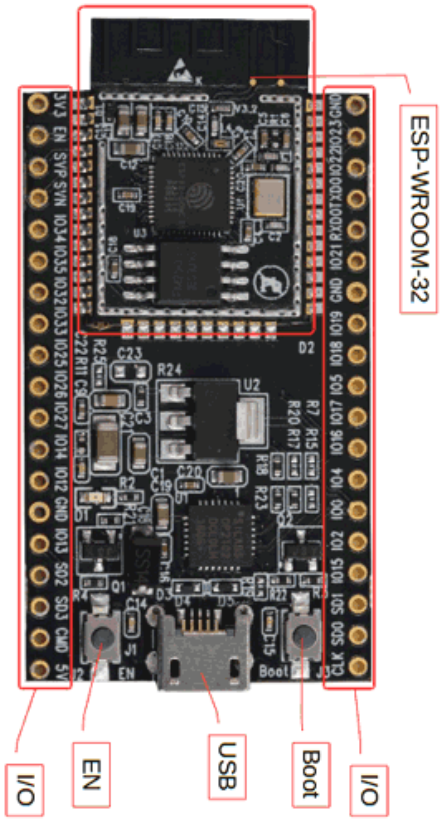
The microcontroller is supposed to manage (directly) its environment !

ESP32

In practical courses, we will use Evaluation Module based on ESP32.

➢ The board of the module is like the motherboard of the microprocessor but as functionalities are included in the ESP32 chip (microcontroller class), the board is "just for pins and connectors".





Compared features

This figure compares features of a "Thing" supported by a microcontroller with a desktop/laptop and its microprocessor :

	Objet connecté d'entreprise		Poste de travail / mobile
Mémoire vive	100 ko	x 20 000	2 Go
Stockage	256 ko	x 1million	256 Go
Fréquence	32 MHz	x 100	3 GHz
Consommation	10 µW	x 1 million	10 W
Bande passante	1 kbit/s	x 10 000	10 Mbit/s

Clearly, it is difficult to **win** on the two sides : functionalities and performances.

But here, the problem is not to get the "best" !

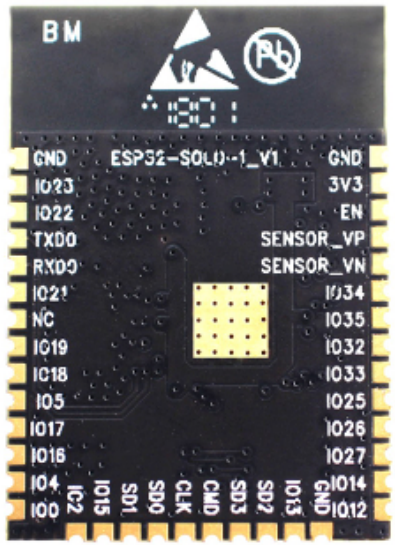
➢ **It is rather to get the "good" . . . features for the targeted applications**.

In the context of the IoT Apps, the number of processors could be VV large so the costs and the consumptions have to be **LOW** !

ESP32 : heart of the EVM (evaluation module) used during the practical sessions.

Module ESP32

ESP32 is the name of a **family of microcontrollers** created and developed by a chinese company : Espressif Systems (Shanghai).



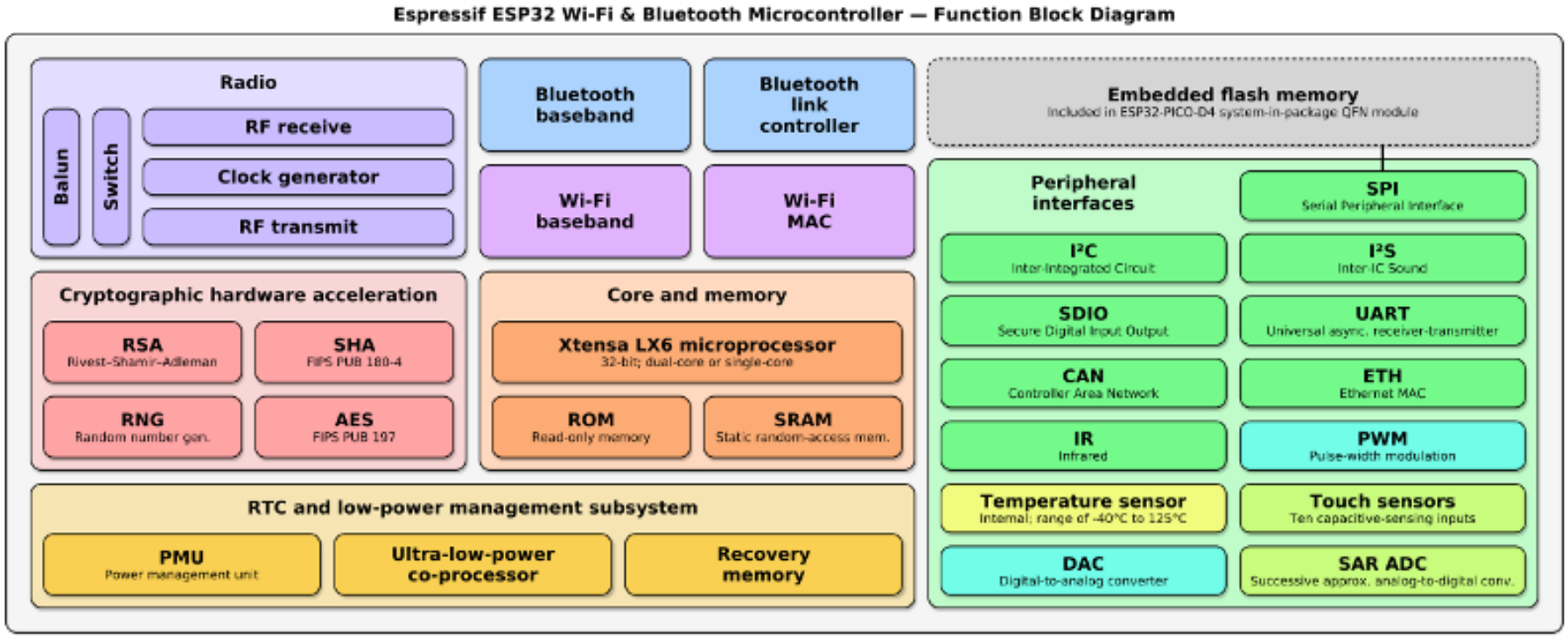
- ✔ ESP32 is the successor of the well known and used ESP8266.
- ✔ ESP32 are based on a dual-core (or single-core) microprocessor : "Tensilica Xtensa LX6".

https://www.espressif.com/en/products/hardware/esp32/overview

https://en.wikipedia.org/wiki/ESP32

Features

It is a (sufficiently given targeted applications) powerful core (or dual core) with numerous functionalities and interfaces :



- ➢ **Radio (antenna) and wireless networking (Wifi and Bluetooth)** are integrated !
- ➢ There is also an embedded flash memory.

Specifications	Value
Number of cores	2
Architecture	32 bit Xtensa L106
CPU Frequency	160 MHz
Wi-Fi	YES
Bluetooth	YES
RAM	512 KB
FLASH	16 MB
GPIO Pins	36
Communication Protocols	SPI, IIC, I2S, UART, CAN
ADC channels	18 channels
ADC Resolution	12-bit
DAC channels	2
DAC Resolution	8-bit
Ethernet MAC Interface	1

Be careful, ESP32 has really many features, but don't forget this a 10 euros machine : memories **are not** in Giga !

Evaluation Module based on ESP32

The Module ESP32-WROOM-32 is here mounted on a development board : 10 euros (from France) !

Different versions of ESP32 module (more or less Flash / Pseudo Static Memory) exist :
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/modules-and-boards.html#modules

Different versions of boards too !

https:

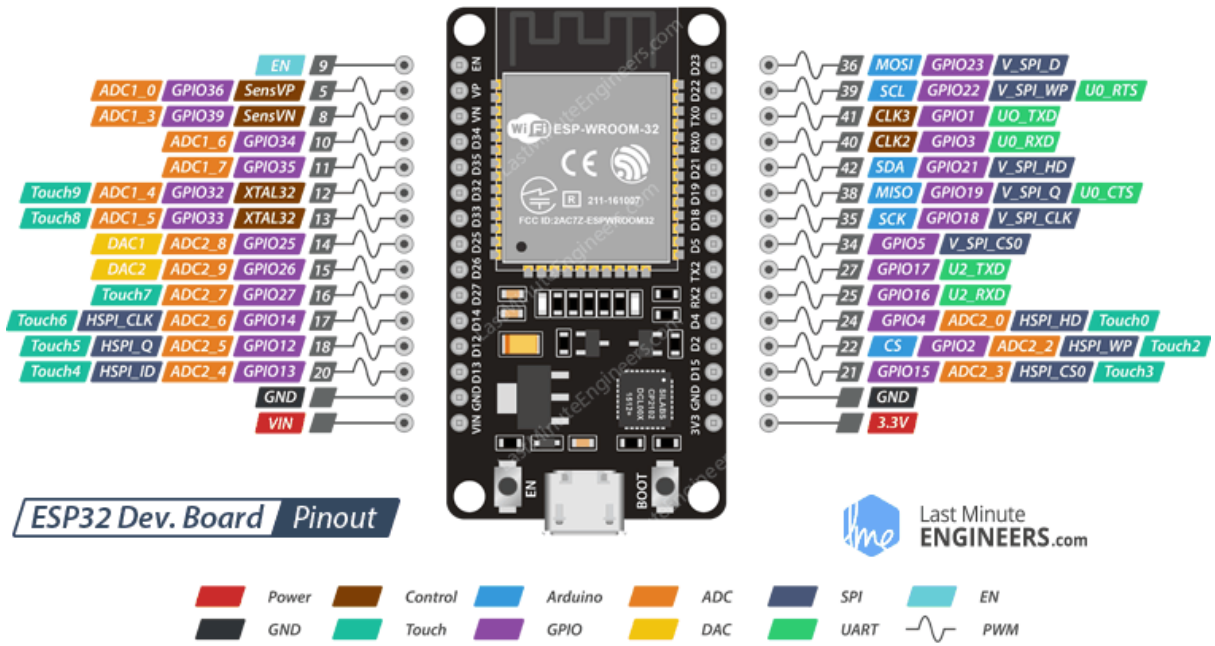
//docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/modules-and-boards.html#development-boards

just for "made by espressif" versions.

"Pinout" of this board

The pinout gives the **correspondance between the pin position and its function**.

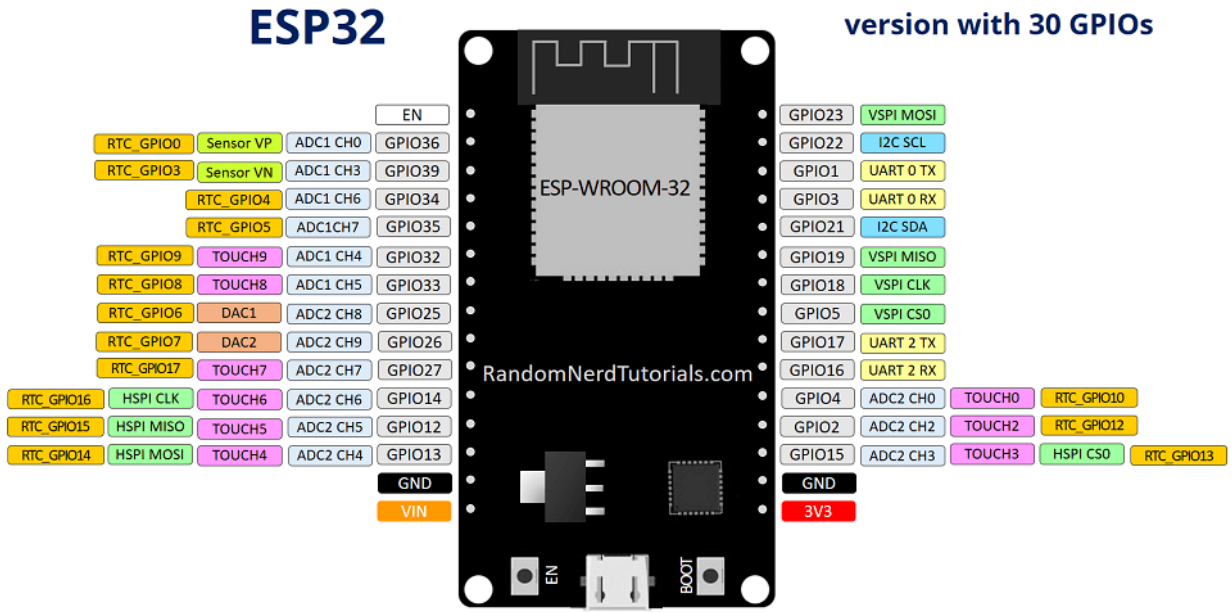
Given our model (ESP32-DOIT-DEVKIT) of eval module, the pinout is :



RMK : Another model of board could have different pinout / functionalities / addons (GPS, LoRa, . . .) !

- ✔ https://makeradvisor.com/esp32-development-boards-review-comparison/
- ✔ https://projetsdiy.fr/quelle-carte-esp32-choisir-developper-projets-diy-objets-connectes/

Each pin can be acting in different functions : that is **"functional multiplexing"** and that is why they are different colored squares in front of each.



Colors highlight types of functions : GP I/O, ADC, ...

➢ https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

As a consequence, in your program you will have to

① **Set** the functionality of the pin

② and then **activate** it calling the associated function of the API.

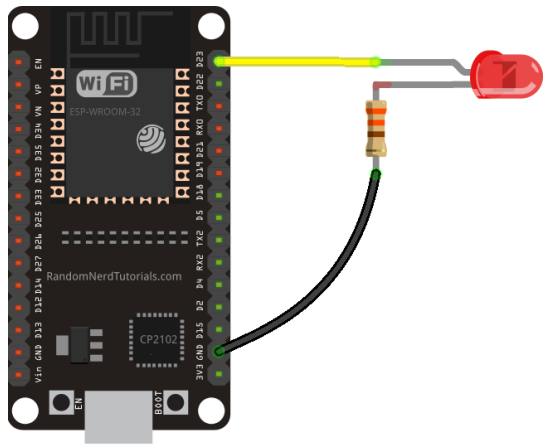
GPIO : General Purpose Input Output

Grey pins are "GPIO" (General Purpose Input Output) pins :

➢ https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/gpio.html

This kind of pin is typically used to control a **digital** input/ouput :

✔ set or get a voltage : Here we will light "on (3.3V) / off (0V)" a LED using "GPIO 23"

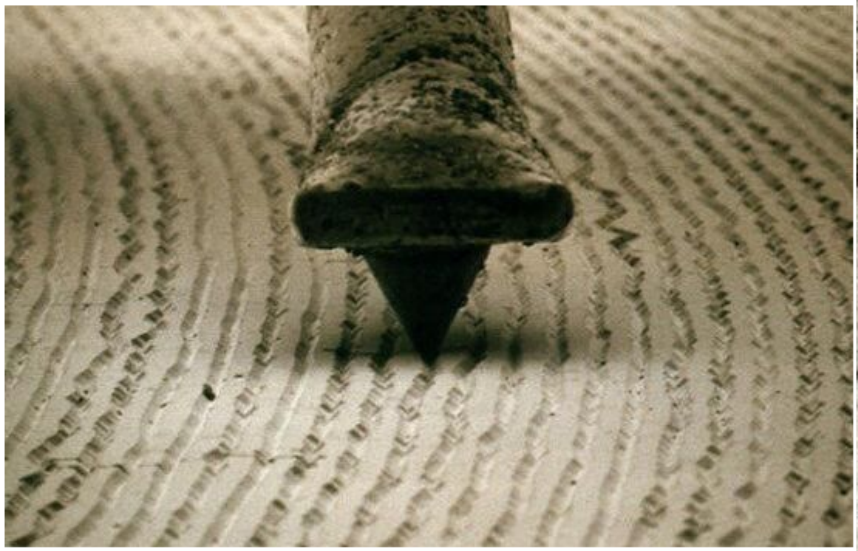


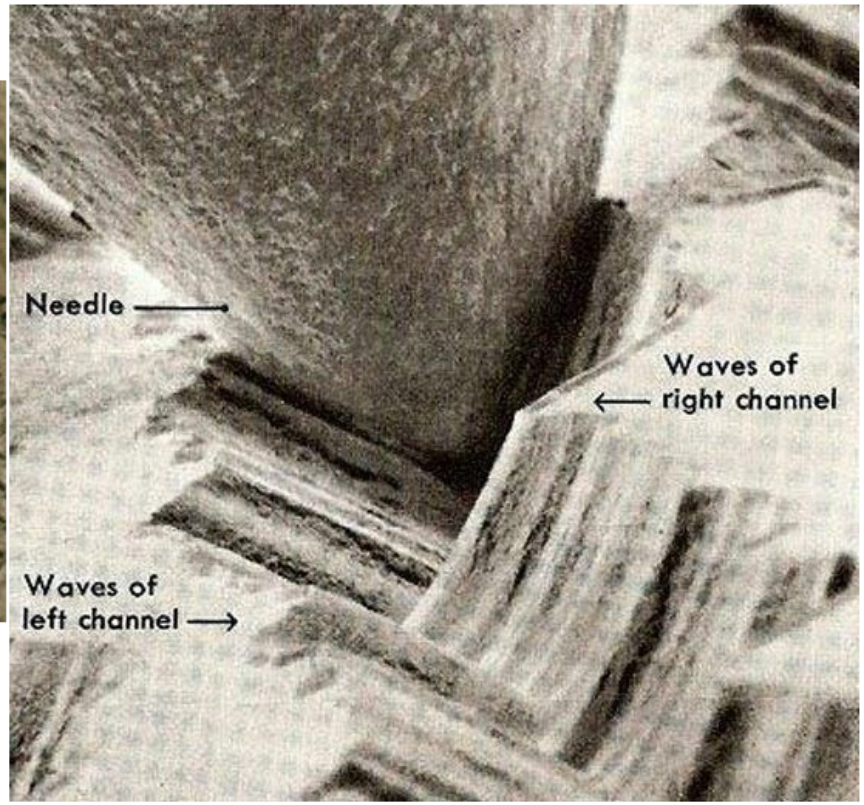
✔ detect an interrupt signal, generate PWM (Pulse Width Modulation), . . .

All GPIO **do not have** the same behavior. For example, GPIO 34-39 can be used **as input only** !

Analog / Digital

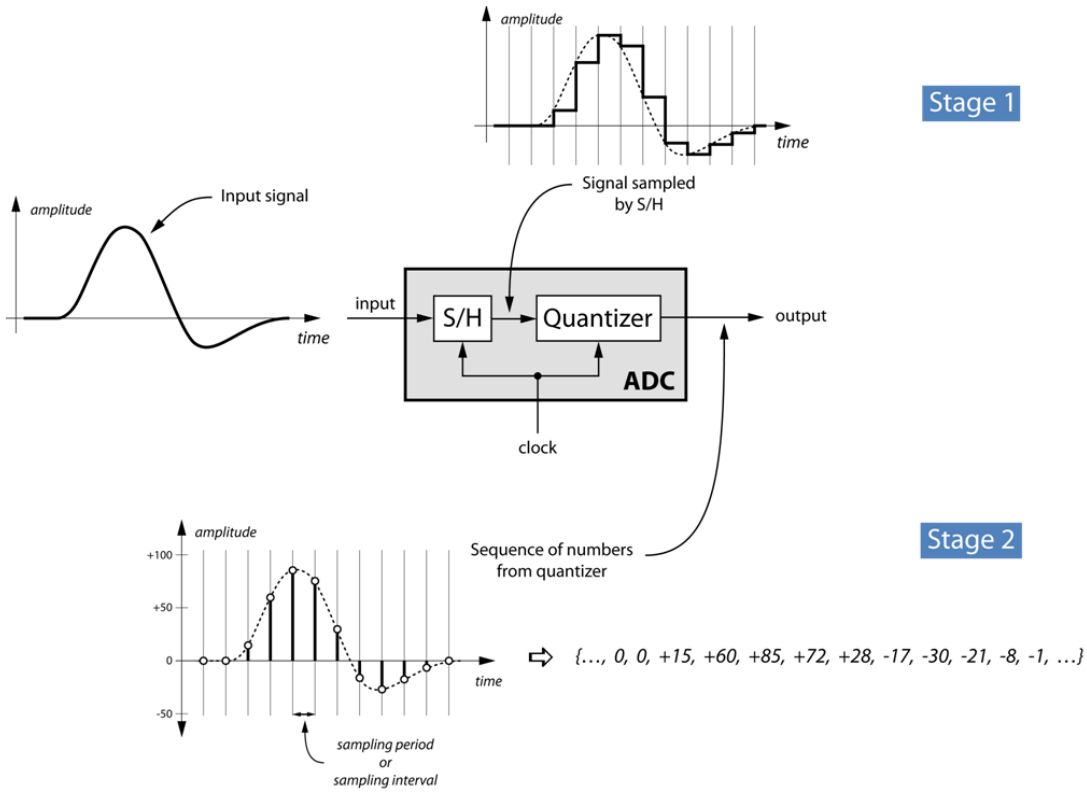
" **Analog** " comes from the fact that the measure of a value (a voltage, a pressure, a light . . . a true world value) varies "AS" the source of the value.





By nature, the value $x(t)$ of an analog signal exists at each time $t \in \mathbb{R}$ and has a defined value among infinity : $x(t) \in \mathbb{R}$.

A **sampled signal** is a signal whose values exist and can be measured at certain moments : $s(t)$ is not defined/known for all $t \in \mathbb{R}$.
A **digital signal** (quantified) is a signal whose possible values at moment t can be indexed by a small subset of $\mathbb{N}$ (after coding).

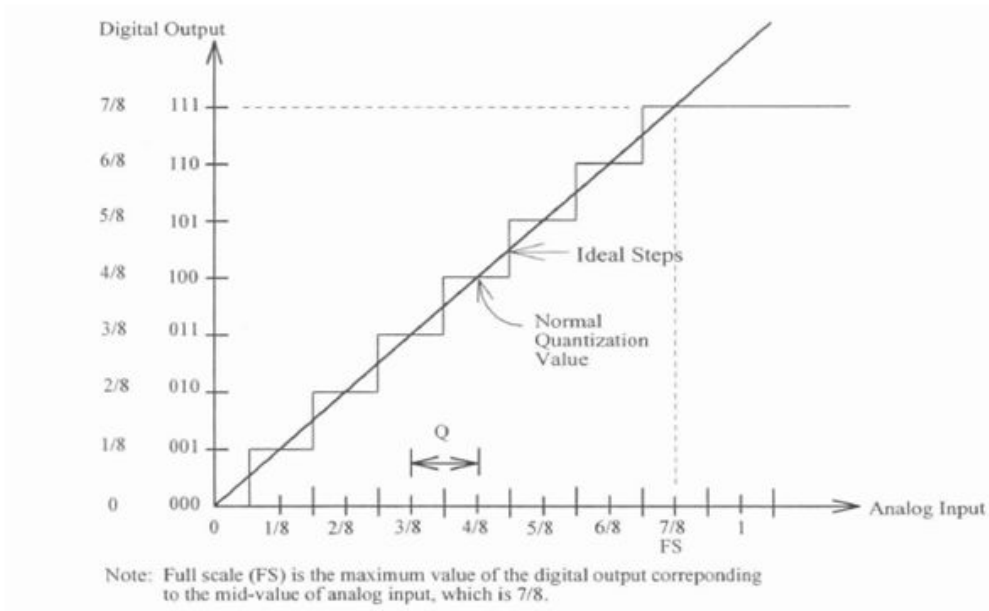


Analog signal cannot be processed by software if they are not converted and digitalized.

ADC : Analog/Digital Converter

ESP32 offers two Analog Digital converters :

➢ They are "grey blue" pads (Analog Digital Converter).



Note: Full scale (FS) is the maximum value of the digital output correponding to the mid-value of analog input, which is 7/8.

Q= Δ=quantization step = full scale / levels' number.

The resolution of the ADC is the smallest detectable change in voltage.

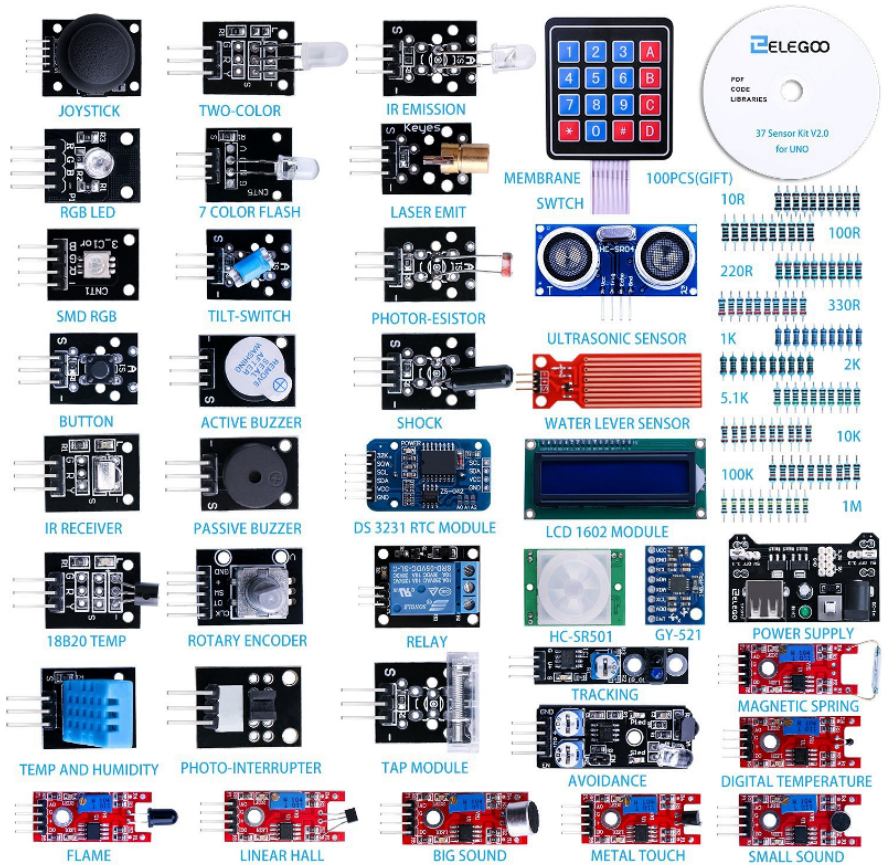
In an ideal analog-to-digital converter, the quantization error is uniformly distributed between –Δ/2 and Δ/2.

https://dl.espressif.com/doc/esp-idf/latest/api-reference/peripherals/adc.html

These converters can code a maximal range ("full-scale voltage") of 1.1 Volt with 12 bits :

- "The full-scale voltage is the voltage corresponding to a maximum reading (depending on ADC1 configured bit width, this value is: 4095 for 12-bits)"

Use Cases and Interfaces

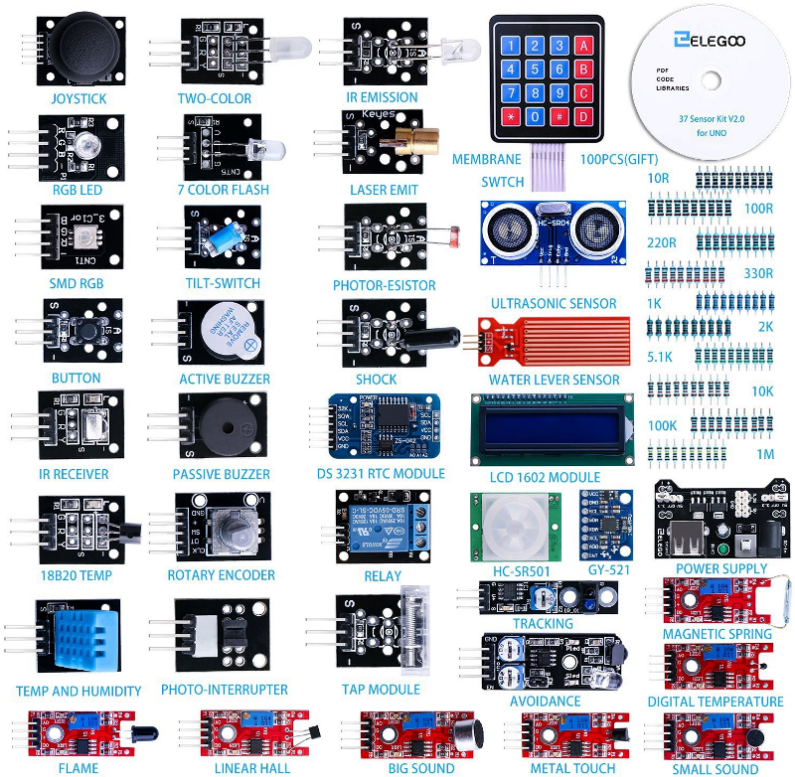


Many Sensors (GPS, distance, ...), Actuators, Motors, ... can be interfaced with ESP32.

https://github.com/espressif/esp-idf/tree/master/examples

But ...

The aim of this course is not to be exhaustive about them because this is not a "software business".



And honestly if you make one ... the other will nearly behave the same ... just read the application notice in google !

Programming ESP32

ESP32 is a full featured programmable machine and your are supposed to be "software skilled students" => Let's go !

But ESP32 is not a desktop nor a Rasberry, I mean ESP32 is not running an OS functionally comparable to Linux or Android.

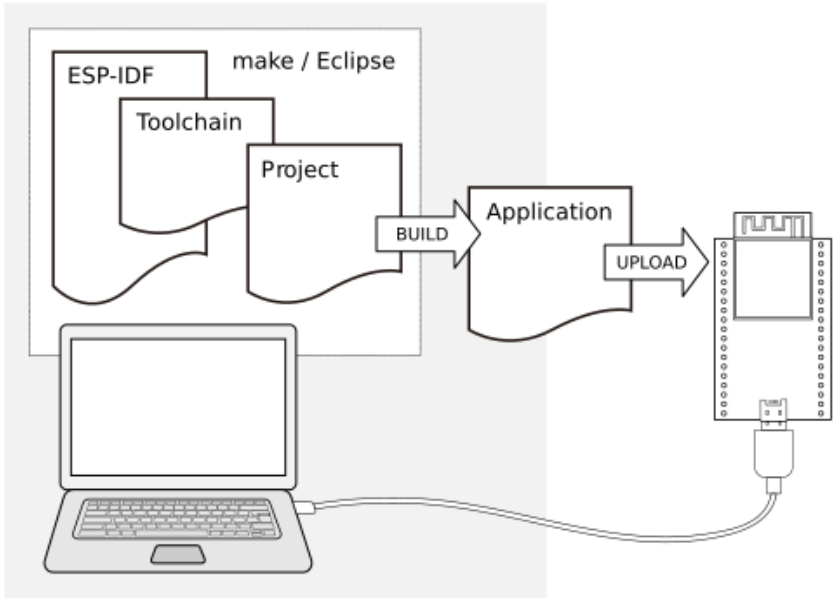
➢ So, you cannot edit and compile on it directly, you will have to do **"cross compilation"** : https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html

You **edit**,

compile (targetting ESP32) on your desktop

and then **flash** ESP32 memory (uploading your app).

Your program is now a part part of the firmware of the ESP.



ESP32 Firmware

In computing, **firmware** is a specific class of computer software **that provides the low-level control** for a device's specific hardware.

Examples of firmware include

- timing and control systems for washing machines,
- the BIOS found in older IBM-compatible PCs and (U)EFI on many newer PCs,
- ...

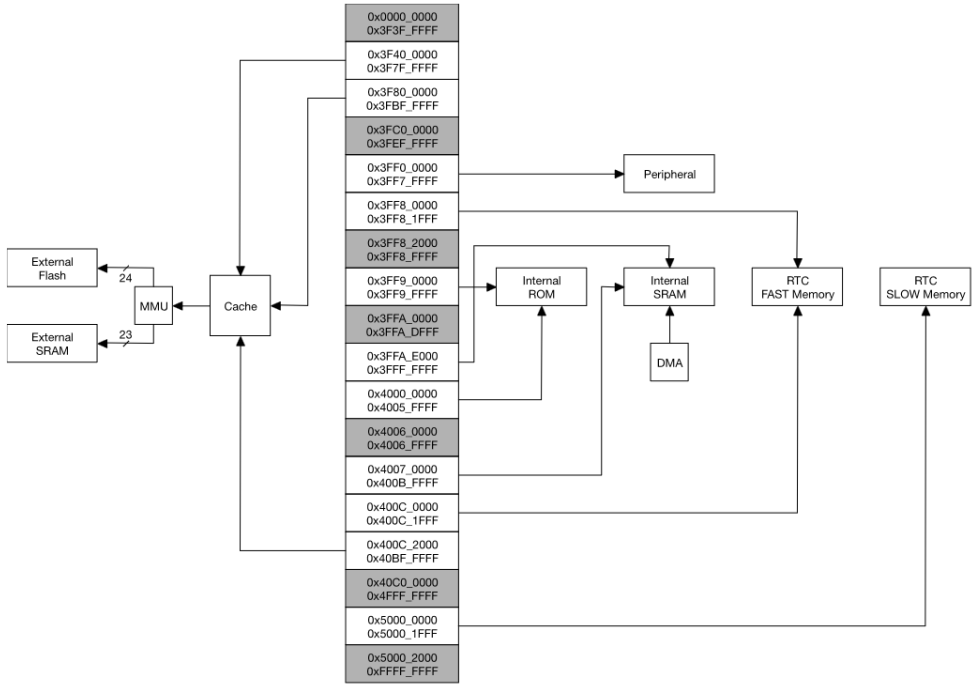
The firmware is supposed to be, at least partially, **"resident" in the machine**.

- So, firmware is held in non-volatile memory devices such as ROM, EPROM, or flash memory (electrically erased and reprogrammed).

That is the reason why the ESP32 chip requires and have an **external flash memory** to store programs, data, configuration parameters ...

ESP Memories

Comme toute machine programmable, l'ESP32 nécessite une mémoire pour stocker les programmes, les données, . . . c'est le modèle de Von Neumann !



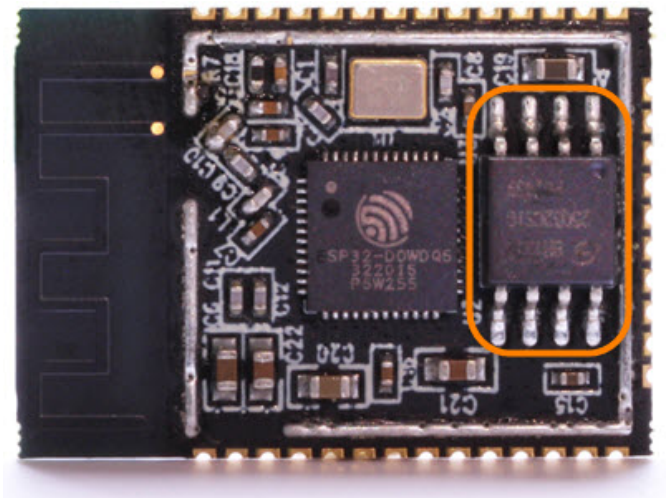
Il y en a en fait 6 :

- Certaines externes au chip, Flash et SRAM (perd tout quand l'alimentation se coupe),
- Certaines internes (ROM, SRAM et RTC Mem).

SPI Flash Memory

The external flash memory is connected to the chip via the SPI bus and the supported capacity is **up to** 16Mb.

- ➢ "Our" dev module (https://www.joy-it.net/en/products/SBC-NodeMCU-ESP32) includes a 4Mb flash memory made by GigaDevice (GD25Q32) but all boards do not have the same size of flash memory.



As this is a flash memory, you can power down the ESP WITHOUT loosing the content of the memory.

PSRAM

The ESP32 chip contains 520KB of RAM . . . this is internal SRAM.

While it's sufficient for most projects, others may need more memory.

To increase the capacity of the microcontroller, the manufacturer can add a memory chip to the board.

- This external RAM chip is connected to the ESP32 via the SPI bus.

If your board has some PSRAM, the following references show how to use this "Pseudostatic RAM (PSRAM)", when we should really say "External SPI RAM" :

- https://arduinojson.org/v6/how-to/use-external-ram-on-esp32/
- https://www.upesy.fr/products/upesy-esp32-wrover-devkit-board?variant=42120383234300

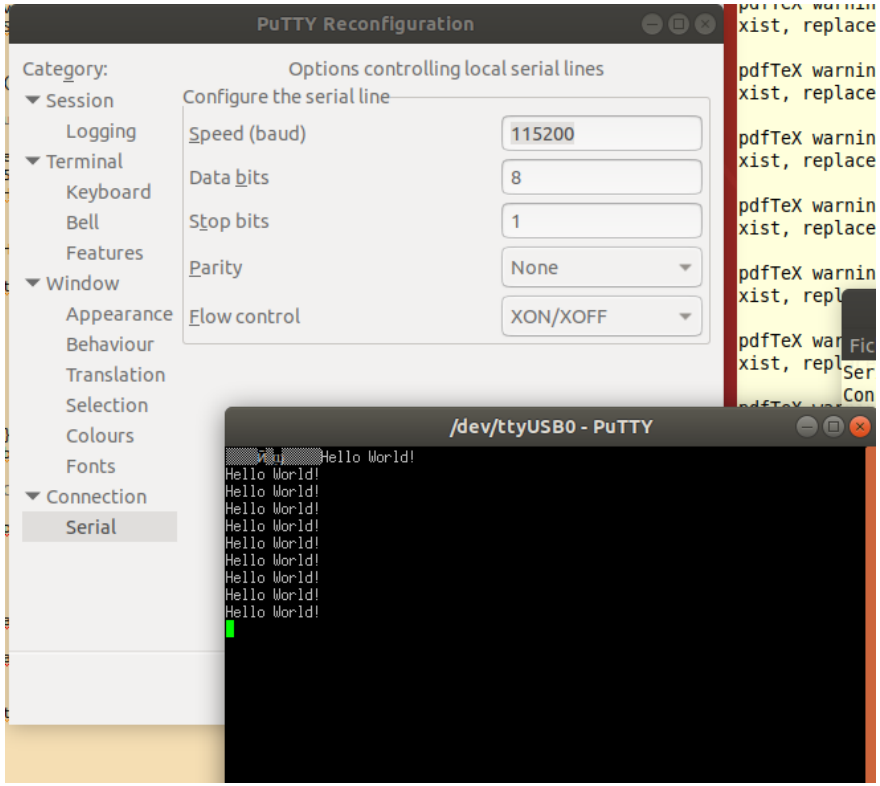
Serial communication

Your board communicates with your desktop through a USB serial link.

➢ In Linux, this link is associated to a tty (terminal)

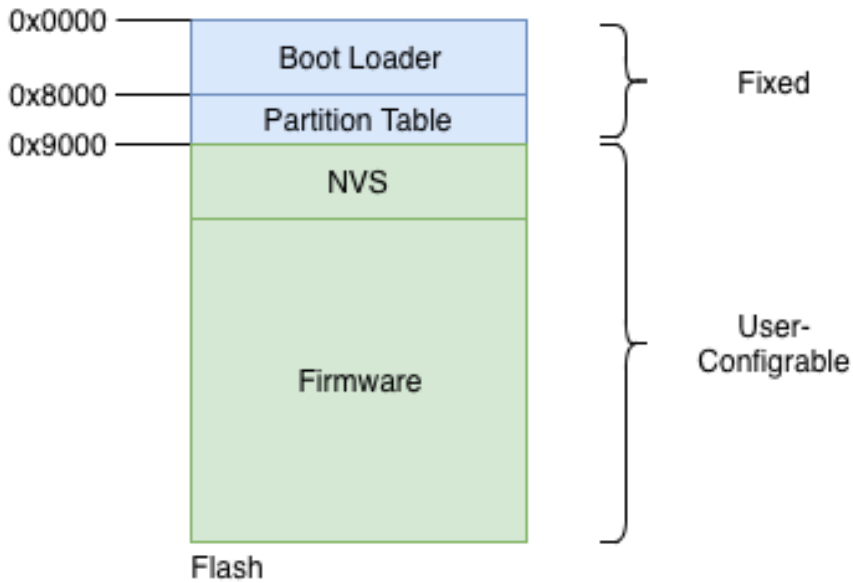
To display data sent by ESP on this tty, you can use `Putty` :

https://docs.espressif.com/projects/esp-idf/zh_CN/release-v3.2/get-started-cmake/establish-serial-connection.html



Flash memory partitioning

The flash memory can store different elements : programs, data . . . hence it's divided into sections (partitions).



The list of partitions, their size and position within the flash memory is stored in the memory itself (at address 0x8000) and it's called **partition table**.

Two **partition types** are defined by default:

① app (type 0) : partition that contains an application (program)

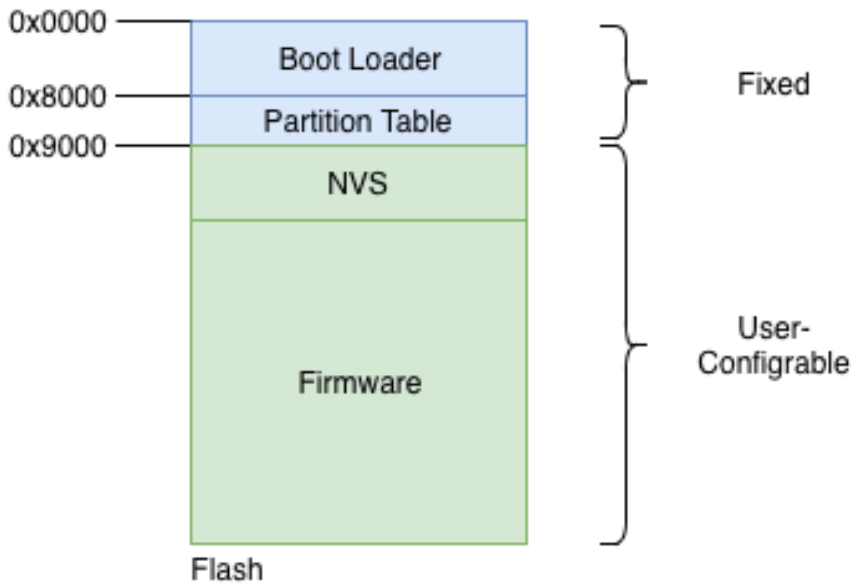
② data (type 1) : partition that contains data

A typical partitioning

The structure is static up to flash address 0x9000.

The first part of the flash contains the second-stage bootloader, which is immediately followed by the partition table.

➢ The partition table then stores how the rest of the flash should be interpreted.

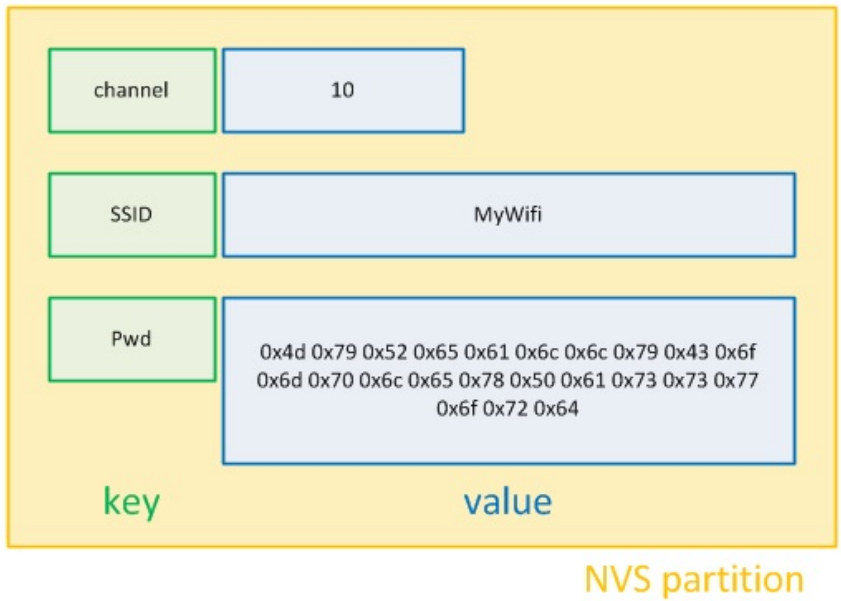


For User configurable part, typically an installation will have at-least 1 NVS partition and 1 firmware partition.

NVS : Non volatile Storage

The NVS is a software component that maintains a persistent storage of key-value pairs.

- ➢ Since the storage is persistent, **this information is available even across reboots and power shutdowns**.
- ➢ The NVS uses a dedicated section of the flash to store this information.



http://www.lucadentella.it/en/2017/06/03/esp32-19-nvs/

Make partition in a project

To see partitions : https://iot.stackexchange.com/questions/4287/how-can-i-list-the-partition-table-of-a-currently-running-esp32-devboard

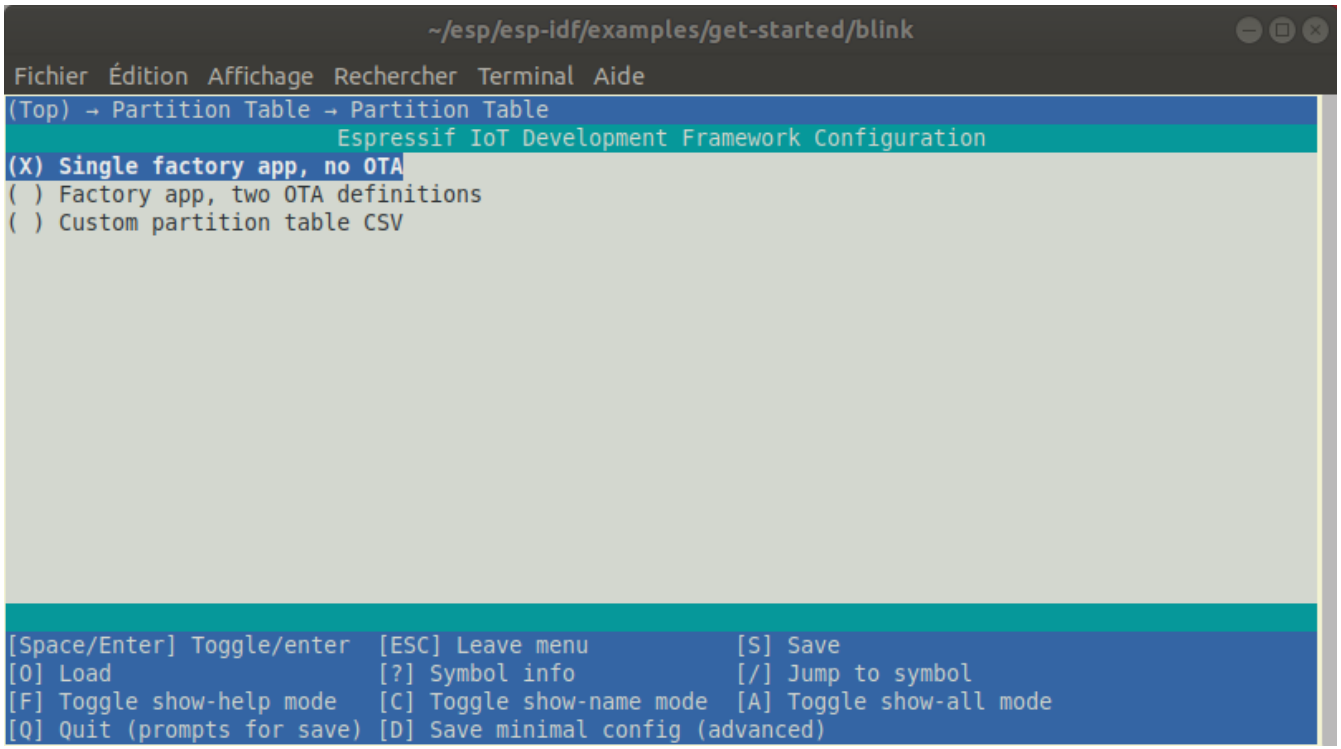
In the "get-started/blink" project of the ESP-IDF framework, I made (and see) the partition :

```
menez@mowgli ~/esp/esp-idf/examples/get-started/blink
$ make partition_table
Toolchain path: /home/menez/.espressif/tools/xtensa-esp32-elf/esp-2020r2-8.2.0/xtensa-esp32-elf/bin/xtensa-esp32-elf-gcc
Toolchain version: esp-2020r2
Compiler version: 8.2.0
Python requirements from /home/menez/esp/esp-idf/requirements.txt are satisfied.
GENCONFIG
Loading defaults file /home/menez/esp/esp-idf/examples/get-started/blink/sdkconfig.defaults...
App "blink" version: v4.2-dev-1905-g625bd5eb1-dirty
Python requirements from /home/menez/esp/esp-idf/requirements.txt are satisfied.
Building partitions from /home/menez/esp/esp-idf/components/partition_table/partitions_singleapp.csv...
Partition table binary generated. Contents:
*******************************************************************************
# ESP-IDF Partition Table
# Name, Type, SubType, Offset, Size, Flags
nvs,data,nvs,0x9000,24K,
phy_init,data,phy,0xf000,4K,
factory,app,factory,0x10000,1M,
*******************************************************************************
Partition flashing command:
python /home/menez/esp/esp-idf/components/esptool_py/esptool/esptool.py --chip esp32 --port '/dev/ttyUSB0' --baud 115200 --before default
_reset --after hard_reset write_flash 0x8000 /home/menez/esp/esp-idf/examples/get-started/blink/build/partitions_singleapp.bin
```

Configuration menu for partition

When writing a new program, the developer can decide how to organize the flash memory based on the program's specific needs. The esp-idf framework offers two pre-configured memory layouts, you can choose from the configuration menu of the project :

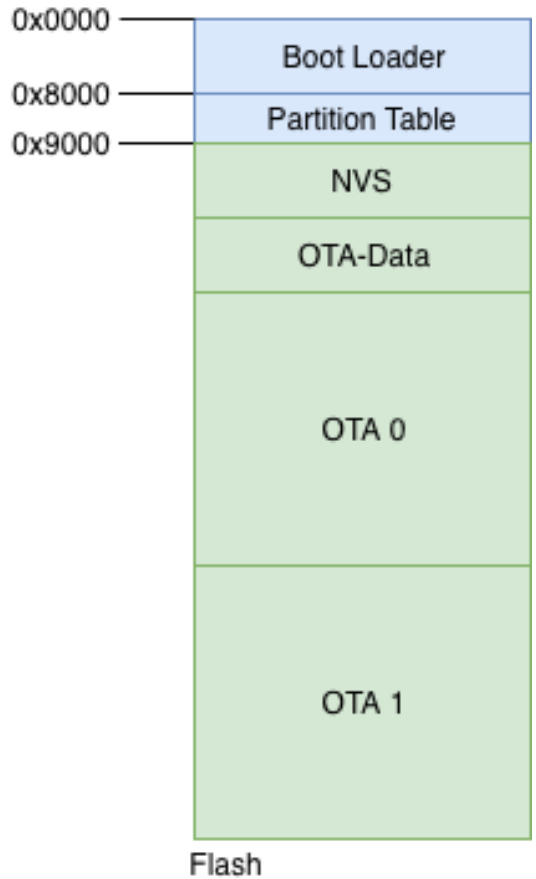
```
menez@mowgli ~/esp/esp-idf/examples/get-started/blink
$ idf.py menuconfig
```



OTA : Over The Air

The OTA update mechanism allows **a device to update itself based on data received while the normal firmware is running** (over WiFi or Bluetooth.)

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html



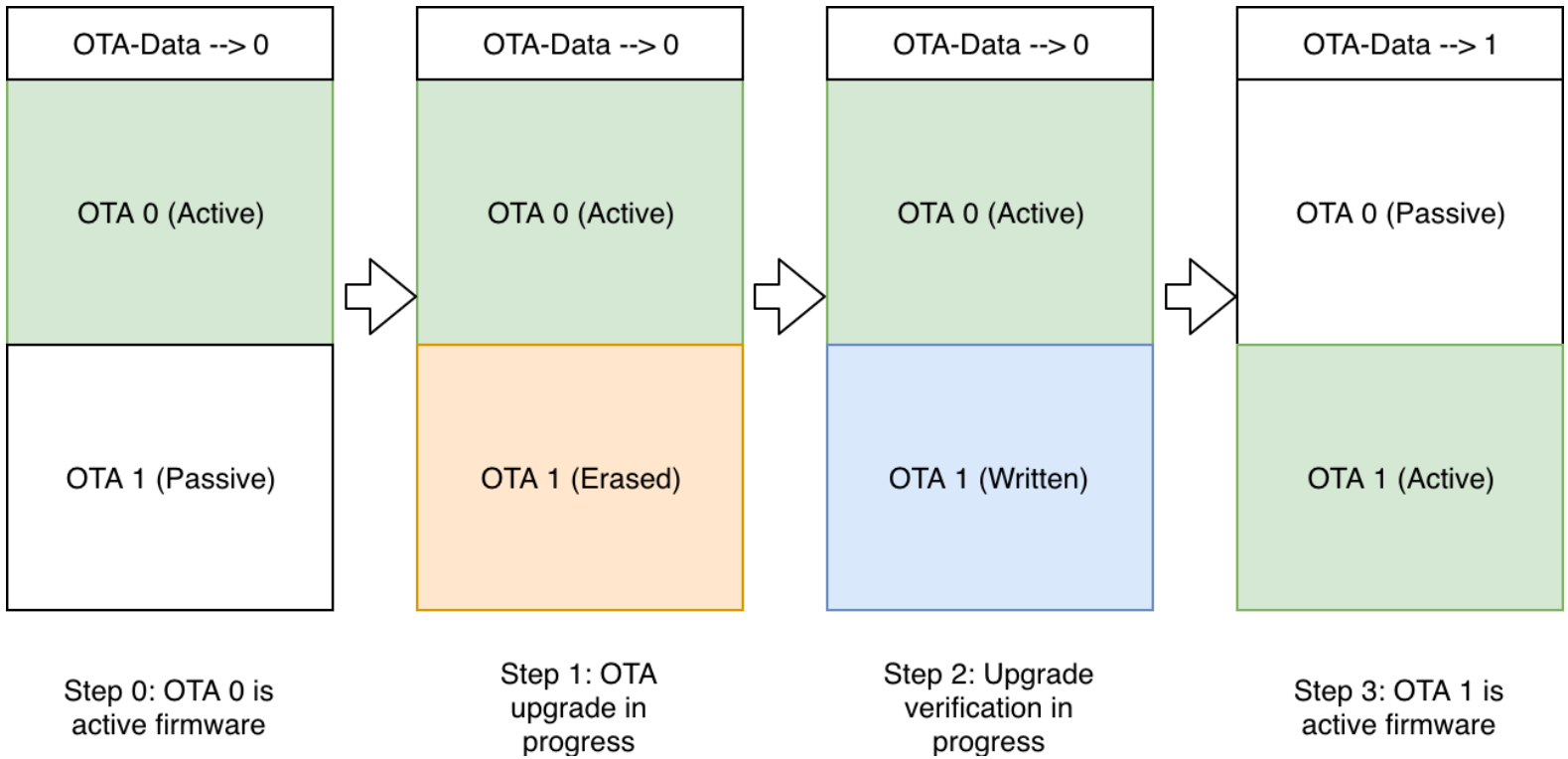
OTA requires configuring the Partition Table of the device with :

- ✔ an "OTA Data Partition".
- ✔ and at least two "OTA app slot" partitions (ie OTA_0 and OTA_1)

The OTA operation functions write a new app firmware image to whichever OTA app slot is not currently being used for booting.

➢ Once the image is verified, the OTA Data partition is updated to specify that this image should be used for the next boot.

The typical state changes that happen across the OTA firmware upgrade workflow are as shown in the figure :



https://docs.espressif.com/projects/esp-jumpstart/en/latest/firmwareupgrade.html

- Step 0: OTA 0 is the active firmware.

  The OTA data partition stores this information as can be seen.

- Step 1: The firmware upgrade process begins.

  The passive partition is identified, erased and new firmware is being written to the OTA 1 partition.

- Step 2: The firmware upgrade is completely written and verification is in-progress.

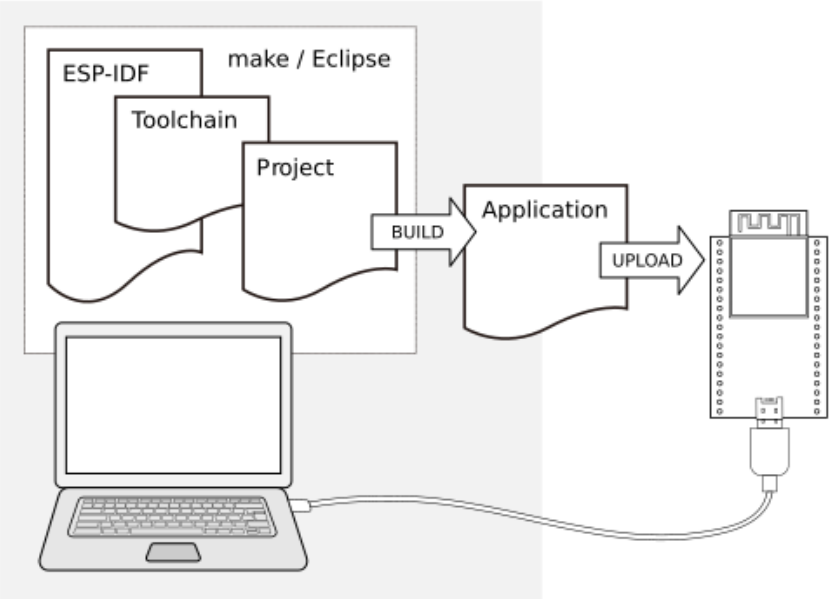
- Step 3: The firmware upgrade is successful, the OTA data partition is updated to indicate that OTA 1 is now the active partition.

  On the next boot-up the firmware from this partition will boot.

ESP-IDF : Development Framework

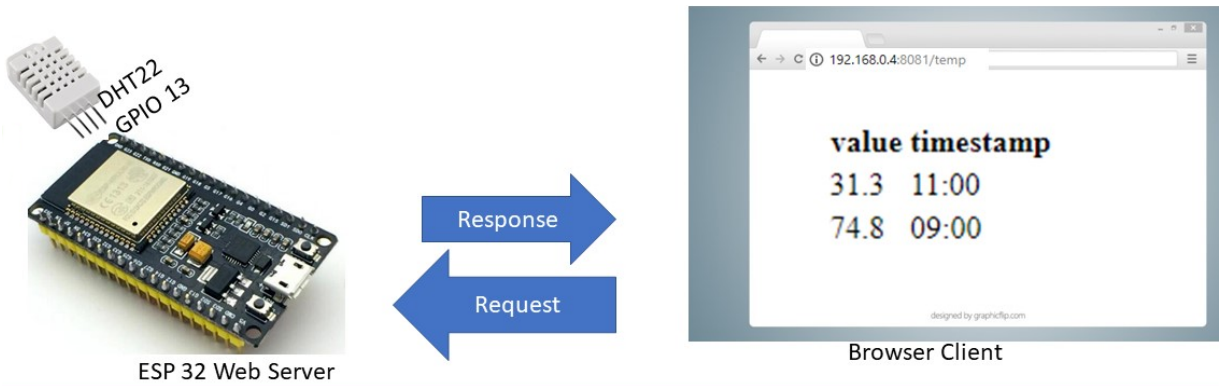
The ESP hardware module is provided with a software development framework : ESP-IDF (Espressif IoT Development Framework)

➢ for menu configuration, then for building and flashing firmware onto an ESP32 board. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html



ESP-IDF essentially contains API (software libraries and source code) for ESP32 **AND** scripts to operate the "Toolchain" to compile code for ESP32

An ESP-IDF project can be seen as **an amalgamation of a number of components** (modular pieces of standalone code which are compiled into static libraries (.a files) and linked into an app. Some are provided by ESP-IDF itself, others may be sourced from other places.)



For example, for a webserver that shows the current humidity, there could be:

- ✔ The ESP-IDF base libraries (libc, ROM bindings, etc)
- ✔ The WiFi drivers
- ✔ A TCP/IP stack
- ✔ The FreeRTOS operating system
- ✔ A webserver
- ✔ A driver for the humidity sensor
- ✔ Main code tying it all together

ESP-IDF makes these components explicit and configurable.

ESP-IDF project

A **"project"** is a directory that contains all the files and configuration to build a single **"app"** (executable), as well as additional supporting elements such as a partition table, data/filesystem partitions, and a bootloader.

- ✔ When a project is compiled, the build system will look up all the components in the ESP-IDF directories, the project directories and (optionally) in additional custom component directories.
- ✔ It then allows the user to configure the ESP-IDF project using a a text-based menu system to customize each component.

  You can remove useless driver from your firmware . . . here BT for example !

  "Project configuration" is held in a single file called **sdkconfig** in the root directory of the project.

  This configuration file is modified via `idf.py menuconfig` to customise the configuration of the project.

  A single project contains exactly one project configuration.
- ✔ After the components in the project are configured, the build system will compile the project.

ESP "App"

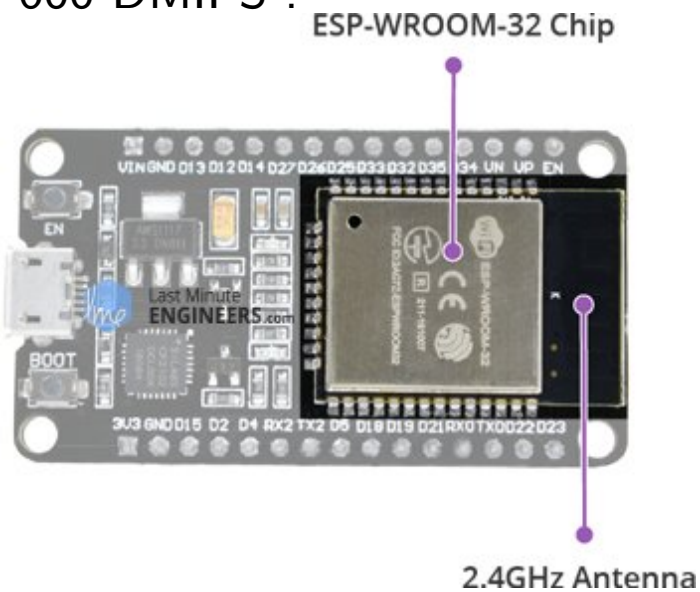
An **"app"** is an executable which is built by ESP-IDF.

A single project will usually build two apps :

- a **"project app"** (the main executable, ie your custom firmware)
- and a **"bootloader app"** (the initial bootloader program which launches the project app).

ESP32 Cores

ESP32 has Xtensa **Dual**-Core 32-bit LX6 microprocessors, which runs up to 600 DMIPS .



ESP32 will run on breakout boards and modules from 160Mhz up to 240MHz.

➢ "very good speed for anything that requires a microcontroller with connectivity options".

The **two cores are named** in the ESP-IDF :

① Protocol CPU (PRO_CPU)

This core handles the WiFi, Bluetooth and other internal peripherals like SPI, I2C, ADC etc.

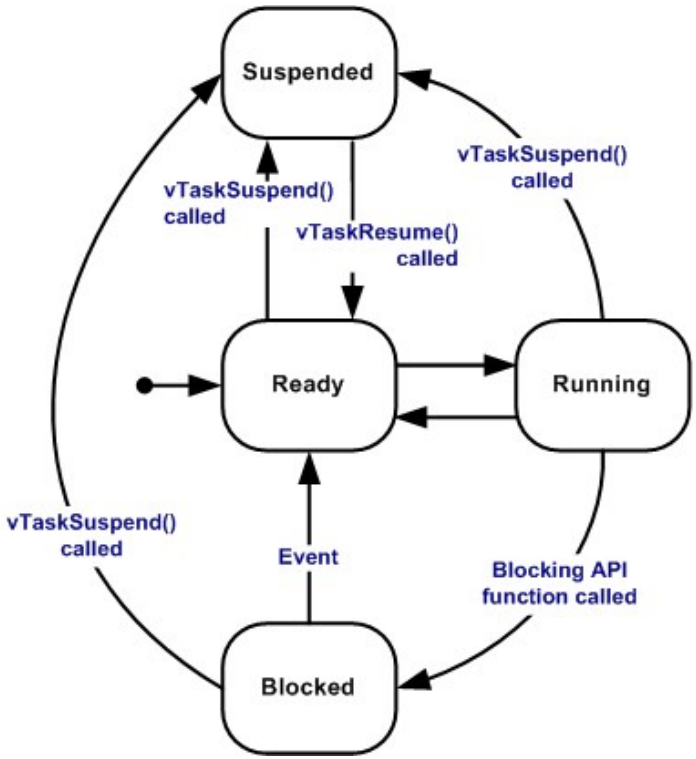
② and Application CPU (APP_CPU).

This core is left out for the application code.

ESP-IDF and FreeRTOS

The esp-idf framework is based on the FreeRTOS Real-Time Operating System.

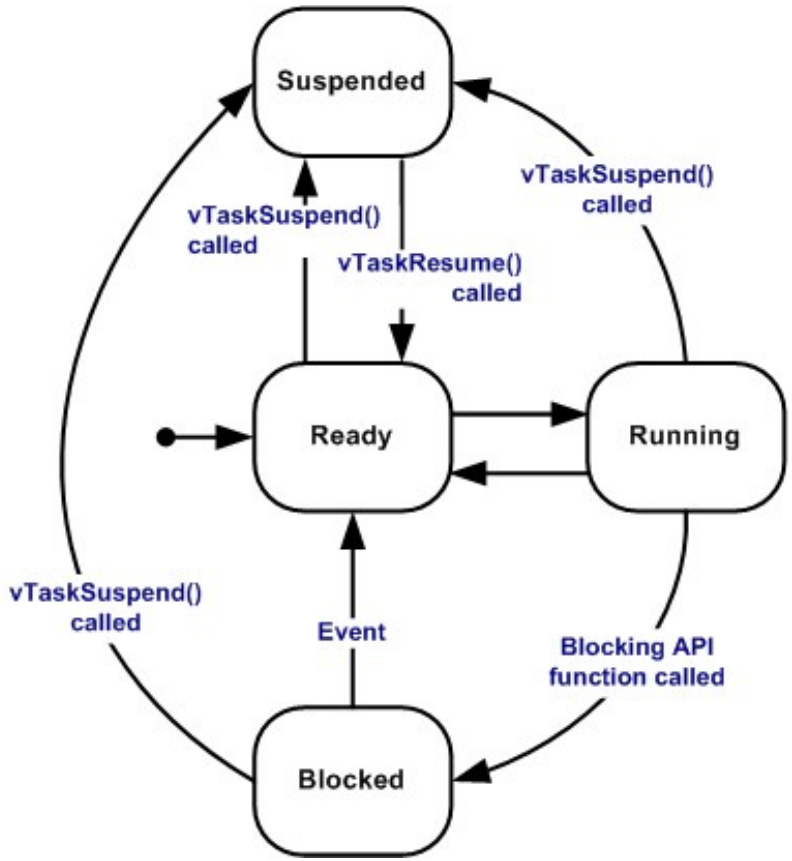
- ➢ It may sound strange to talk about operating systems when working on a "small" chip like the esp32 . . . but you don't have to think that FreeRTOS is an operating system like Windows, Linux or MacOS.



The main feature an embedded operating system offers, thanks to its internal scheduler, is multitasking,

- ➢ that is the ability to run different tasks in parallel.

A "real-time" operating system is design to guarantee that task scheduling is deterministic: **you can predict** the behavior of its scheduler.



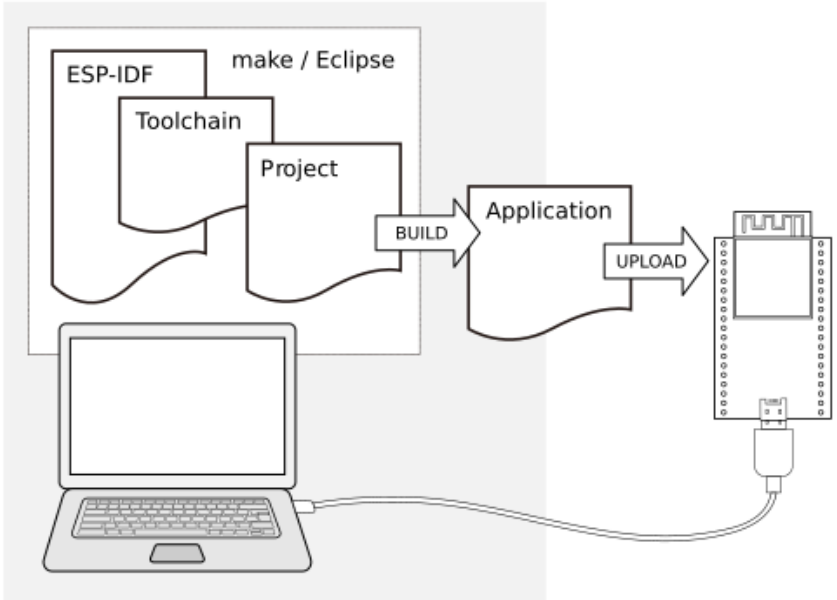
FreeRTOS **allows the developer to define a priority** for each task: the scheduler uses priority values to define the execution pattern of the different tasks.

http://www.lucadentella.it/en/2016/12/22/esp32-4-flash-bootloader-e-freertos/

Programming languages, frameworks, ...

ESP-IDF is the official software development framework for the chip.

https://www.espressif.com/en/products/hardware/esp32-devkitc/resources



If we use it, programs will be written in C using the "tasks" paradigm.

ESP-IDF will provide a collection of libraries and header files that are the core software components required to build any software projects on ESP32.

ESP-IDF will also provides tools and utilities that are required for typical developer and production usecases, like build, flash, debug and measure.

A tiny example : "Hello World"

A quite classic "dev project"stucture :

- A project directory with a "makefile" pointing on IDF make configuration
- a C source code file
- a "component.mk" file can be empty in the folder where the source code is saved into to tell the compiler to process the files in that folder.

```
menez@mowgli ~/EnseignementsCurrent/Cours_IoT/Src_IDF/esp32-tutorial-master/00_empty
$ ls -alg
total 16
drwxrwxr-x  3 menez 4096 juil.  1 18:58 .
drwxrwxr-x 34 menez 4096 oct.  27  2018 ..
drwxrwxr-x  2 menez 4096 oct.  27  2018 main
-rw-rw-r--  1 menez  177 oct.  27  2018 Makefile

menez@mowgli ~/EnseignementsCurrent/Cours_IoT/Src_IDF/esp32-tutorial-master/00_empty
$ ls -alg main/
total 12
drwxrwxr-x 2 menez 4096 oct.  27  2018 .
drwxrwxr-x 3 menez 4096 juil.  1 18:58 ..
-rw-rw-r-- 1 menez  306 oct.  27  2018 00_empty.c
-rw-rw-r-- 1 menez    0 oct.  27  2018 component.mk
```

```
menez@mowgli ~/EnseignementsCurrent/Cours_IoT/Src_IDF/esp32-tutorial-master/00_empty
$ cat Makefile
#
# This is a project Makefile. It is assumed the directory this Makefile resides in is a
# project subdirectory.

PROJECT_NAME := 00_empty
include $(IDF_PATH)/make/project.mk


menez@mowgli ~/EnseignementsCurrent/Cours_IoT/Src_IDF/esp32-tutorial-master/00_empty
$ cat main/00_empty.c
/* From
   http://www.lucadentella.it/en/2016/12/22/esp32-4-flash-bootloader-e-freertos/
*/
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

void loop_task(void *pvParameter){
  while(1) {
    printf("Hello World!\n");
    vTaskDelay(5000 / portTICK_RATE_MS);
  }
}

void app_main(){
    xTaskCreate(&loop_task, "loop_task", 1024, NULL, 5, NULL);
}
```

Build and Flash

To build this app "hello world" :

① First, you have to set the environment (variables) to point on Makefiles of ESP-IDF.

So, you "source" (not execute !) the "export.sh" file :

```
menez@mowgli ~/EnseignementsCurrent/Cours_IoT/Src_IDF/esp32-tutorial-master/00_empty
$ . ~/esp/esp-idf/export.sh
Setting IDF_PATH to '/home/menez/esp/esp-idf'
Adding ESP-IDF tools to PATH...
Using Python interpreter in /home/menez/.espressif/python_env/idf4.2_py2.7_env/bin/python
Checking if Python packages are up to date...
Python requirements from /home/menez/esp/esp-idf/requirements.txt are satisfied.
Added the following directories to PATH:
  /home/menez/esp/esp-idf/components/esptool_py/esptool
  /home/menez/esp/esp-idf/components/espcoredump
  /home/menez/esp/esp-idf/components/partition_table
  /home/menez/esp/esp-idf/components/app_update
  /home/menez/.espressif/tools/xtensa-esp32-elf/esp-2020r2-8.2.0/xtensa-esp32-elf/bin
  /home/menez/.espressif/tools/xtensa-esp32s2-elf/esp-2020r2-8.2.0/xtensa-esp32s2-elf/bin
  /home/menez/.espressif/tools/esp32ulp-elf/2.28.51-esp-20191205/esp32ulp-elf-binutils/bin
  /home/menez/.espressif/tools/esp32s2ulp-elf/2.28.51-esp-20191205/esp32s2ulp-elf-binutils/bin
  /home/menez/.espressif/tools/openocd-esp32/v0.10.0-esp32-20200420/openocd-esp32/bin
  /home/menez/.espressif/python_env/idf4.2_py2.7_env/bin
  /home/menez/esp/esp-idf/tools
Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

  idf.py build
```

① Then you can build : `make` or `idf.py build`

② and flash : `make flash`

Do not forget to plug the ESP board !

Alternative Dev Environments

Using ESP-IDF as a necessary sublayer, many compilers / interpreters / frameworks were developed to allow different languages as alternatives (often along with a dedicated firmware).

- Espruino – **JavaScript** SDK and firmware closely emulating Node.js
- **Lua**
- Mongoose OS – an operating system for connected products on microcontrollers; programmable with JavaScript or C.
- **mruby**
- NodeMCU – Lua-based firmware
- MicroPython, Zerynt – **Python**
- Arduino IDE with the ESP32 Arduino Core

. . . just google them you will find them all !

Arduino IDE

The easy way ?

Let's see that in Practical courses !