

IoT/IdO : Practical 1

Gilles Menez

Université Nice Côte d'Azur

email : menez@unice.fr

www : www.i3s.unice.fr/~menez

February 15, 2024: V 1.0

Table of Contents

1 Environnement de Programmation	6
1.1 Cross compilation	6
2 Installation et utilisation de l'ESP	7
2.1 Prérequis ?	7
MacOs et Windows : Drivers UART ?	7
"Linux Only"	7
2.2 Espressif IoT Development Framework : ESP-IDF	8
La méthode (plus) difficile ...sans IDE !	8
Projet sans IDE	8
La méthode (plus) facile !	9
2.3 Arduino IDE	10
Installation de "Arduino IDE"	10
Paramétrage de l'"Arduino IDE"	11
2.4 VSCode et PlatformIO ...ca peut se tenter ?	12
Install Espressif on VSCode :	12
Install PlateformIO IDE on VSCode :	12
2.5 L'exploitation de l'ESP32 en Python	12
2.6 Simulateur	12
3 Le module d'expérimentation ESP32	13
3.1 BreadBoard	15
4 ESP32 Programming under Arduino IDE paradigm	16
4.1 Sketch/Croquis	16
4.2 setup()/loop()	16
4.3 Cross Compilation / Uploading	17
4.4 CLI	17
5 TODO : First Sketch	18
5.1 Avec Arduino IDE	18
6 Exemples/Tutoriaux/...	20
7 GPIO : General Purpose Input/Output	21
7.1 DEL(/LED) : Digital output	22
7.2 U=RI	22
Code du firmware	24
Fonctions de l'API	24
Résultat attendu de ce premier exemple	24
7.3 Light intensity : ADC	25
Photo résistance	25
Les broches/fonctions impliquées	26
Code	26
Fonctions de l'API	27
Résultat attendu	27
7.4 Temperature with DS18B20	28
Cablage	29
Protocole OneWire et Bibliothèque	30

Le code	31
7.5 Temperature and Moisture with DHT11	33
Cablage	33
Bibliothèques "BY Adafruit"	34
Le code	34
7.6 DHT11/DHT22 – Failed to read from DHT sensor	36
8 PWM and ... fan	37
8.1 Signaux "Pulse Width Modulation" : PWM	37
8.2 Brochage PWM de l'ESP	38
8.3 Canaux PWM de l'ESP	38
8.4 API PWM de l'ESP	38
8.5 Sketchs basiques	40
Rapport Cyclique : 50%	40
Rapport Cyclique : 25%	40
8.6 Le ventilateur	40
Rasberry Pi4 case fan	41
Noctua fan	41
8.7 Code	43
9 Led Strip (ou bande de LEDs)	46
9.1 Mini Led Strip Adressable	46
9.2 Led Strip RGB	47
9.3 Code	48
10 TODO : YOUR work !	50
10.1 Temperature Sensing & Regulation	50
10.2 Detection Process	51
10.3 Fire Detection	51
11 Conclusion	52
12 A suivre ...	52

List of Figures

1	Cross compilation	6
2	lsusb	7
3	Arduino IDE	10
4	Choose the good Module	11
5	ESP32 - Module	13
6	Breadboard	15
7	Upload	16
8	main loop	19
9	ESP32 Pinout	21
10	Blink layout	22
11	LED and U=RI	23
12	LED and U=RI	23
13	Photoresistor and U=RI	25
14	Photoresistor layout	26
15	Temperature Sensor DS18B20	28
16	DS18B20 layout	29
17	DS18B20 and U=RI	30
18	DS18B20 Memory Map	30
19	DHT11	33
20	DHT11 Layout	33
21	Duty cycles	37
22	ESP and PWM	38
23	Duty cycle :50%	40
24	Duty cycle : 25%	40
25	Raspberry Pi4 case fan	41
26	Noctua fan	41
27	Fan layout	42
28	Tachymetry	43
29	RGB Led strip	46
30	RGB Led strip	47
31	Strip layout	48
32	Temperature Sensing & Regulation	50

Foreword

Les objectifs pédagogiques de ce premier travail sont de :

1. Mettre en place un environnement de développement
2. Prise en main du module ESP32 : branchements, programmation, cross compilation, ...
3. Manipulation de quelques capteurs et actionneurs.

Un conseil : **Lisez tout avant de faire !**

et si des fois vous rencontrez une difficulté non évoquée pensez à regarder :

- <https://randomnerdtutorials.com/esp32-troubleshooting-guide/>

1 Environnement de Programmation

Contrairement à un Raspberry qui permet d'y installer une distribution Linux puis d'y développer directement les applications, l'ESP est une plateforme matérielle qui ne permet **pas** l'installation d'un système d'exploitation et d'outils de compilation ou d'interprétation classiques : gcc, python,

- On peut néanmoins programmer des applications dans différents langages : C, C++, Python,

Le langage "officiel" de l'UE sera C pour différentes raisons :

- Existence des bibliothèques C dont on va avoir besoin,
- Plus facile que C++, (si vous êtes vraiment à l'aise vous pouvez faire du C++)
- Suffisant compte tenu des objectifs pédagogiques du cours ... qui ne vont pas se focaliser sur l'"objet" mais plutôt sur son "ecosystème"

1.1 Cross compilation

Puisqu'on ne peut donc pas développer nos applications directement sur l'ESP, mais "juste" les exécuter, l'ESP nécessite une "cross compilation" :

- concevoir l'exécutable sur le desktop (PC ou Mac)
- puis "uploader" cet exécutable (ce firmware) sur l'ESP.

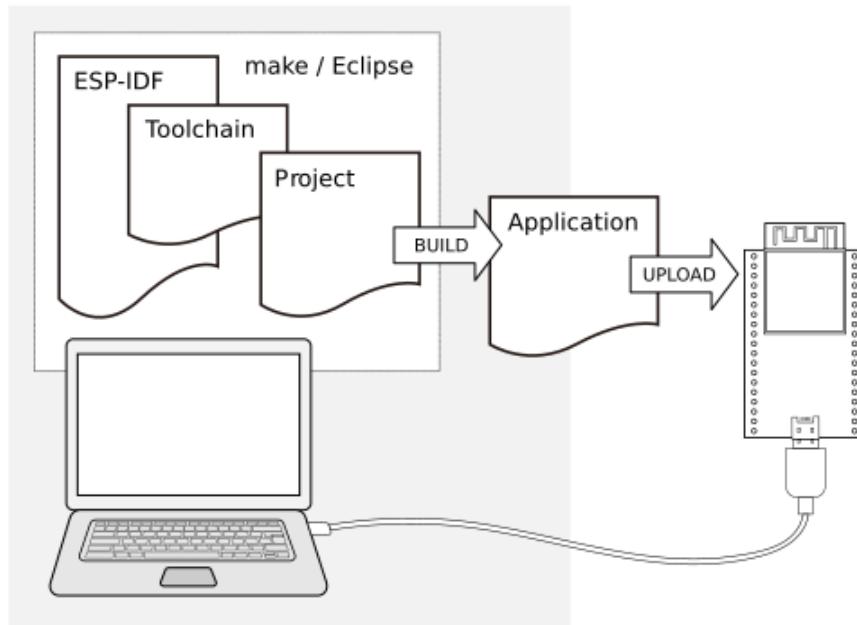


Figure 1: Cross compilation

2 Installation et utilisation de l'ESP

2.1 Prérequis ?

La carte d'évaluation de l'ESP communique via un port USB et selon l'OS, il peut y avoir quelques petites choses à faire ?

MacOs et Windows : Drivers UART ?

Pour les utilisateurs MacOs et Windows qui pourraient avoir un problème d'UART, il peut être nécessaire de charger un pilote driver :

<https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>

"Linux Only" ...

Si vous êtes sous Linux,

1. J'ai installé :

```
sudo apt install python-is-python3
sudo apt install python3-serial
sudo apt install lsusb
```

2. J'ai du aussi m'inscrire dans le group "dialout" de ma machine pour avoir le droit d'utiliser mon port USB : <https://github.com/esp8266/source-code-examples/issues/26>

```
sudo usermod -a -G dialout $USER
```

3. puis rebooter pour que cela devienne effectif.

Grâce à la commande Shell Unix `lsusb`, je peux constater la présence de la carte :

```
$ lsusb
Bus 001 Device 002: ID 8087:0001 Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 003: ID 0424:5534 Standard Microsystems Corp. Hub
Bus 003 Device 004: ID 0424:5534 Standard Microsystems Corp. Hub
Bus 003 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 002 Device 008: ID 04f2:b477 Chicony Electronics Co., Ltd
Bus 002 Device 007: ID 138a:003f Validity Sensors, Inc. VFS495 Fingerprint Reader
Bus 002 Device 005: ID 8087:0a2a Intel Corp.
Bus 002 Device 020: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x UART Bridge / myAVR mySmartUSB light
Bus 002 Device 003: ID 0424:2134 Standard Microsystems Corp. Hub
Bus 002 Device 017: ID 413c:2105 Dell Computer Corp. Model L100 Keyboard
Bus 002 Device 016: ID 12c9:102a
Bus 002 Device 014: ID 0424:2134 Standard Microsystems Corp. Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figure 2: `lsusb`

- C'est la ligne en ... CP210x !

2.2 Espressif IoT Development Framework : ESP-IDF

Espressif (le fabricant de l'ESP) fournit un environnement de développement pour son ESP32.

Il existe plusieurs façons d'y accéder et de l'utiliser mais comme c'est dit dans la documentation :

- <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html>

Il y est écrit :

"We highly recommend installing the ESP-IDF through your favorite IDE!"

La méthode (plus) difficile ... sans IDE !

Pour information, on évoque la méthode (**qui n'est pas recommandée !**) et qui ne repose pas sur un IDE.

Il faut alors installer les logiciels suivants :

- "Toolchain" pour compiler le code pour ESP32
- Outils de construction - "CMake" et "Ninja" pour construire une application complète pour ESP32.
- "ESP-IDF", qui contient essentiellement l'API (bibliothèques logicielles et code source) pour ESP32 et des scripts pour faire fonctionner la chaîne d'outils.

Pour installer ces logiciels, sans IDE, il faudra procéder directement depuis le dépôt "officiel" ESP-IDF.

Les URLs qui suivent donnent des exemples et vont vous aider :

- <https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html>
- <https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html>
- <https://github.com/espressif/esp-idf>
- <https://github.com/espressif/arduino-esp32/>

Une fois installée, vous pouvez faire des codes en C++ que vous compilerez puis téléchargerez sans IDE.

Projet sans IDE

L'url suivante montre comment construire un projet ESP en ligne de commande :

- <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/linux-macos-setup.html#get-started-linux-macos-first-steps>

Le framework ESP-IDF est **dédié** à l'ESP32, et permet d'utiliser pleinement et finement ses capacités.

Par exemple, le code qui suit permet d'initialiser les UART de l'ESP afin de permettre une communication série avec le (port USB du) PC.

```

/* Configure parameters of an UART driver, communication pins and install the driver */
uart_config_t uart_config = {
    .baud_rate = 115200,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_DISABLE
};
uart_param_config(EX_UART_NUM, &uart_config);

/*Set UART pins (using UART0 default pins ie no changes.)
uart_set_pin(EX_UART_NUM,
UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE);

/*Install UART driver, and get the queue.*/
uart_driver_install(EX_UART_NUM,
BUF_SIZE * 2,
BUF_SIZE * 2,
20, &uart0_queue, 0);

```

Ma conclusion : cela fait beaucoup de code à écrire pour configurer une liaison série !

La méthode (plus) facile !

Comme cela est recommandé par Espressif (<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html#installation>) on peut aussi installer et utiliser ESP-IDF au travers d'IDE (<https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html>) tels

- Eclipse : <https://github.com/espressif/idf-eclipse-plugin/blob/master/README.md>,
- VSCode : <https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md>,
- Arduino IDE : <https://www.arduino.cc/en/software>.

Ces IDE doivent vous faciliter la tâche ...

Par exemple, l'IDE Arduino **fait plus que** proposer un éditeur de texte et des boutons dans une UI. Il offre une couche supplémentaire d'abstraction pour la programmation. A titre de comparaison avec le code que l'on vient d'écrire, **il permet de configurer la liaison USB en un seul appel de fonction**:

```
Serial.begin(115200);
```

Eh oui, un seul appel de fonction !

Immédiatement, vous vous dîtes "mais pourquoi faire compliqué si on peut faire simple" ? Parce que les choses et les choix ne sont jamais aussi "simples" qu'elles le paraissent :

- https://www.reddit.com/r/esp32/comments/dk3rsn/why_idf_vs_arduino_ide/

Ceci étant, Arduino IDE introduit une certaine indépendance avec la plateforme matérielle utilisée, vous pourriez ainsi compiler vos codes pour d'autres matériels que l'ESP ... et c'est plutôt pas mal !

2.3 Arduino IDE

L'environnement de développement intégré Arduino est une application multiplate-forme écrite en Java.

- Je la trouve un peu "vieillotte" ... mais elle est plutôt solide et "suffisante" pour le volume de code que l'on fournira à l'ESP !
- **Cette année encore elle sera la plateforme "officielle" !**

Originellement utilisé pour écrire et télécharger des programmes sur les cartes utilisant des "Arduino" (un autre type de microcontrôleur), l'environnement a depuis été étendu pour être capable de gérer d'autres microcontrôleurs dont l'ESP32.

"Arduino-ESP32 is just a wrapper around the ESP-IDF mainly simplifying things and making it integrated with the design paradigms of arduino as a whole".



The screenshot shows the Arduino IDE interface with the title bar "Blink | Arduino 1.8.5". The main window displays the "Blink" sketch. The code is as follows:

```
This example code is in the public domain.  
http://www.arduino.cc/en/Tutorial/Blink  
*/  
  
// the setup function runs once when you press reset or power the board  
void setup() {  
    // initialize digital pin LED_BUILTIN as an output.  
    pinMode(LED_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  
    delay(1000); // wait for a second  
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW  
    delay(1000); // wait for a second  
}
```

The status bar at the bottom right shows "Arduino/Genuino Uno on COM1".

Figure 3: Arduino IDE

On peut donc "passer" par cet IDE pour utiliser le SDK de Espressif permettant de programmer les cartes ESP32. Remarquons tout de même que **cela impose un "design paradigm"** : Le "setup/loop" ... on y revient plus loin.

Installation de "Arduino IDE"

Vous commencez par installer la partie générique de l'outil : <https://www.arduino.cc/en/Main/Software>

Cette partie est la même pour toutes les plateformes matérielles !

ATTENTION !!! On n'installe PAS la version 2.X car elle ne supporte pas encore les plugins ! ... on reste donc en 1.8.X !

L'installation et le premier lancement ont créé deux répertoires (.arduino15 et Arduino) sous ma racine (HOME).

- Le second répertoire "Arduino" contiendra les **futures "libraries"** dont on aura besoin et que l'on récupérera le moment venu.

Paramétrage de l'"Arduino IDE"

Il faut maintenant intégrer le SDK de l'ESP 32 à l'IDE Arduino.

A partir de l'IDE Arduino, vous allez charger le bon paramétrage : https://dl.espressif.com/dl/package_esp32_index.json

Tout est décrit là :

- <https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html>
- <https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

Sur mon système Linux, lorsque l'installation est finie (+ reboot) et que ma carte est branchée sur l'USB, je peux choisir

1. Outils -> Type de Carte -> **ESP32 Dev Module**
2. et le port USB (qui dépend de votre branchement) : /dev/ttyUSB0

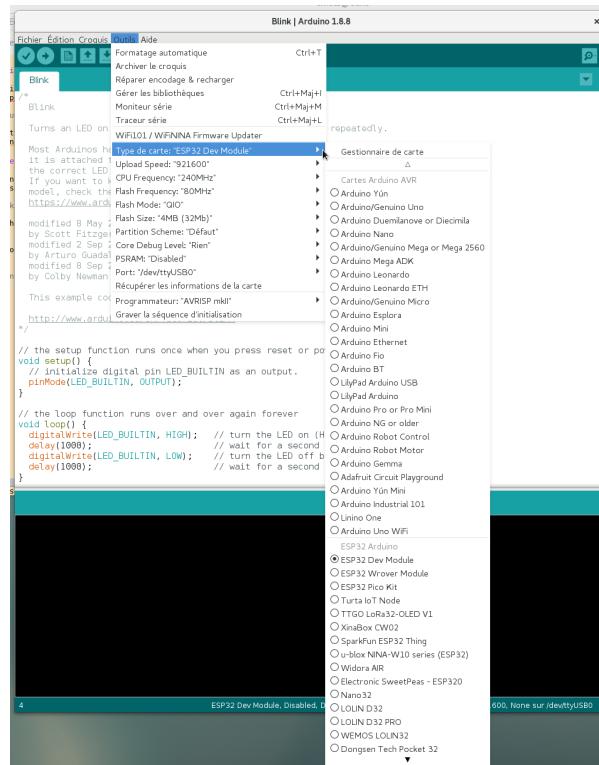


Figure 4: Choose the good Module

2.4 VSCode et PlatformIO ... ca peut se tenter ?

VSCode et PlatformIO, voilà une association intéressante **MAIS PAS SIMPLE et surtout pas forcément compatible avec l'ESP32.**

- <https://github.com/platformio/platformio-core/issues>

PlatformIO is an open source ecosystem for IoT development with cross platform build system, library manager and full support for Espressif ESP8266/ESP32 development.

- It works on the popular host OS ! ... en théorie :-(

PlateformIO se greffe sur VSCode ou CLion (experimental) :

- <https://docs.platformio.org/en/stable/integration/ide/pioide.html>

* je n'ai pas essayé sur <https://vscode.com/> (version de VSCode sans télémétrie) ... à tester ?

Install Espressif on VSCode :

On pourrait installer directement Espressif sur VSCode, mais **on peut aussi passer par PlatformIO** qui devrait faciliter les choses ?!

- <https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md>

Install PlateformIO IDE on VSCode :

Donc la démarche :

1. On installe VSCode (si ce n'est pas déjà fait),
2. on installe PlatformIO IDE,
3. et enfin, via PlatformIO, on installe Espressif IDF .

Quelques URLs qui peuvent aider ?

- <http://platformio.org/platformio-ide>
- <https://esphome.io/components/esp32.html>
- <https://randomnerdtutorials.com/vs-code-platformio-ide-esp32-esp8266-arduino/>

2.5 L'exploitation de l'ESP32 en Python

- Micropython : <https://docs.micropython.org/en/latest/esp32/tutorial/index.html>
- Thonny IDE : <https://www.aranacorp.com/fr/programmez-un-esp32-avec-micropython/>

2.6 Simulateur

Vous pouvez jeter un oeil à ce simulateur :

<https://wokwi.com/projects/305566932847821378>

Un projet sympa pour abstraire la réalité matérielle ... quand c'est possible !?

3 Le module d'expérimentation ESP32

Pour réaliser nos sessions pratiques, nous utilisons des "mini cartes" sur lesquelles sont implantés des ESP-WROOM-32.

- C'est la notion de "module d'évaluation/expérimentation".

Sur le module d'expérimentation contenant l'ESP32 vous avez différents boutons, prises, broches.

Même si la figure ne présente pas exactement la carte (au niveau du "pinout") que nous allons utiliser, le principe reste :

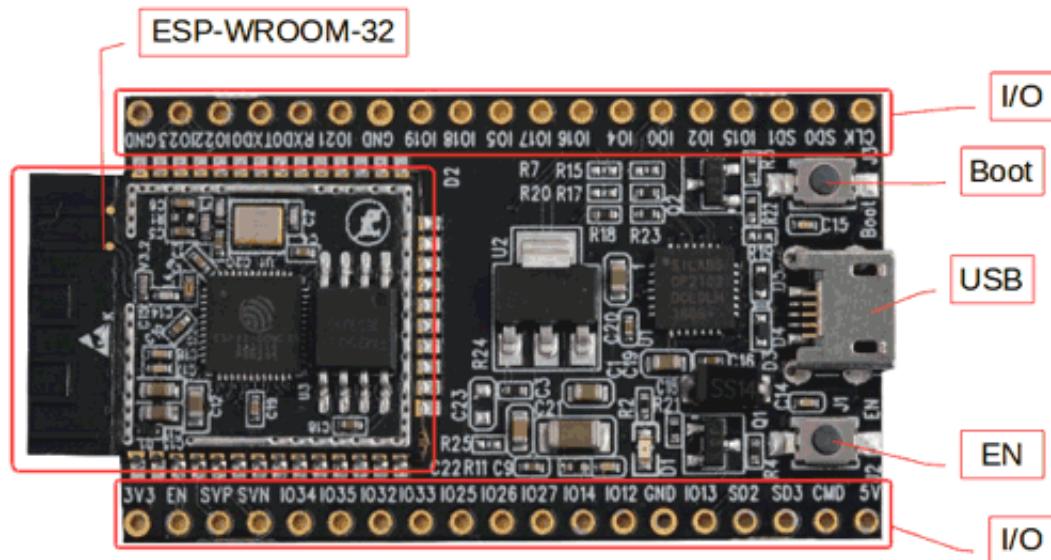


Figure 5: ESP32 - Module

1. Micro-USB jack :

The micro USB jack is **used to connect** the ESP32 to our computer through a USB cable.

It is used to program the ESP module as well as can be used for serial debugging as it supports serial communication

2. EN Button:

The EN button is the **reset button** of the ESP module.

Pressing this button will reset the code running on the ESP module

3. Boot Button:

This button is used to upload the Program from Arduino to the ESP module. **It has to be pressed after clicking on the upload icon on the Arduino IDE :**

- When the Boot button is pressed along with the EN button, ESP enters into firmware uploading mode.

4. Red LED : The Red LED on the board is used to indicate the **power supply**.

It glows red when the board is powered.

5. **Blue LED:** The Blue LED on the board is connected to the GPIO pin.

It can be turned on or off through programming. In some Chinese cloned boards this led might also be in red colour.

6. **I/O pins :**

This is where major development has taken place.

These pins are capable of Digital Read/Write, Analog Read/Write, PWM, IIC, SPI, DAC and much more. We will get more into that later. But if you are interested you can learn through the pin description at ESP32 Datasheet.

7. **ESP-WROOM-32 :**

This is the heart of the ESP32 module. It is a 32-bit microprocessor developed by Espressif systems. The black part is the antenna for radio functionnalities.

3.1 BreadBoard

Ce module sera enfiché sur une platine d'expérimentation :

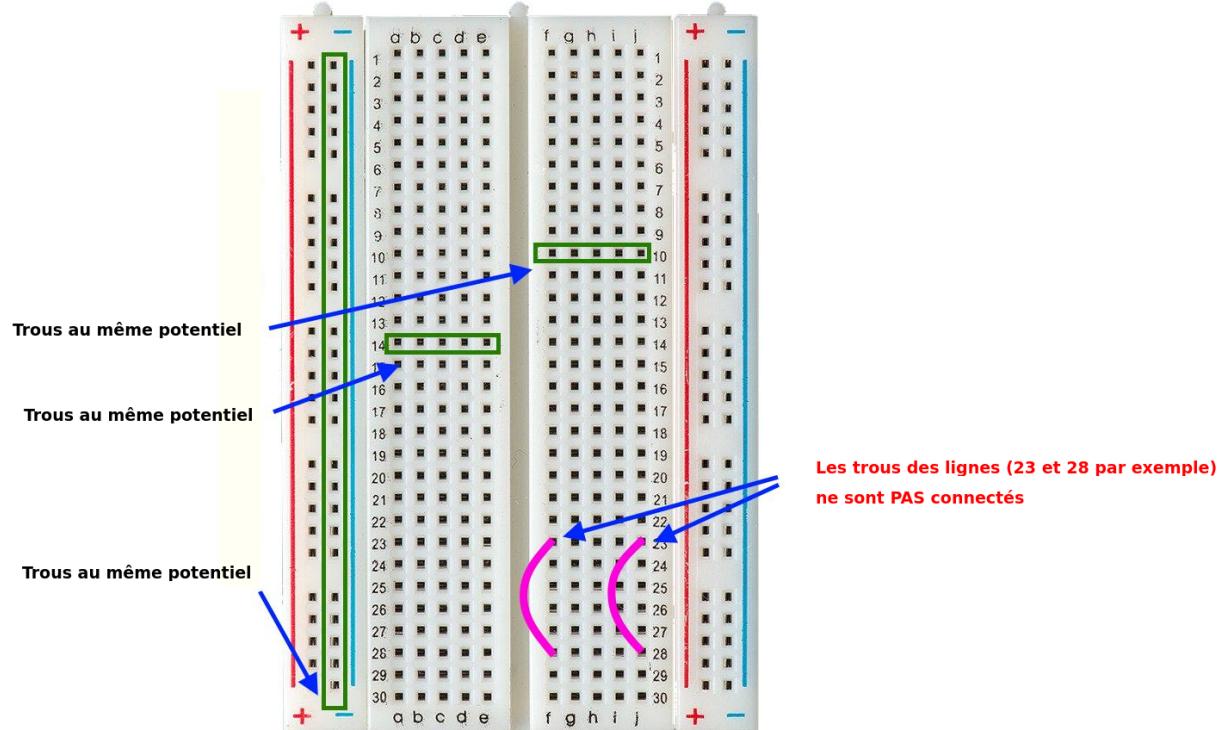


Figure 6: Breadboard

Une platine d'expérimentation ou platine de prototypage (appelée en anglais breadboard) est un dispositif qui permet de réaliser le prototype d'un circuit électronique et de le tester.

- Les trous encadrés en vert **partagent le même potentiel** !
a14, b14, c14, d14, e14 sont reliés entre eux.
- Au niveau du "+" et du "-" certaines des cartes que je vais vous prêter sont plus grandes et **séparent** les colonnes "+" et "-" en deux.
Et au lieu d'avoir une colonne de 1 à 64, on a 2 colonnes une de 1 à 32 et l'autre de 33 à 64.
Ces colonnes NE SONT ALORS PAS CONNECTEES !
- D'une ligne à l'autre, les trous **ne sont pas connectés**.
Les trous f23 et f28 **ne sont donc pas reliés** par la platine.

4 ESP32 Programming under Arduino IDE paradigm

Même si on n'utilise pas l'IDE Arduino, sa couche logicielle simplificatrice reste intéressante !

4.1 Sketch/Croquis

Un "sketch" est la terminologie "arduino" pour la notion de programme.

- C'est "l'unité de code" qui est uploadée et qui s'exécute sur la carte/module d'expérimentation.

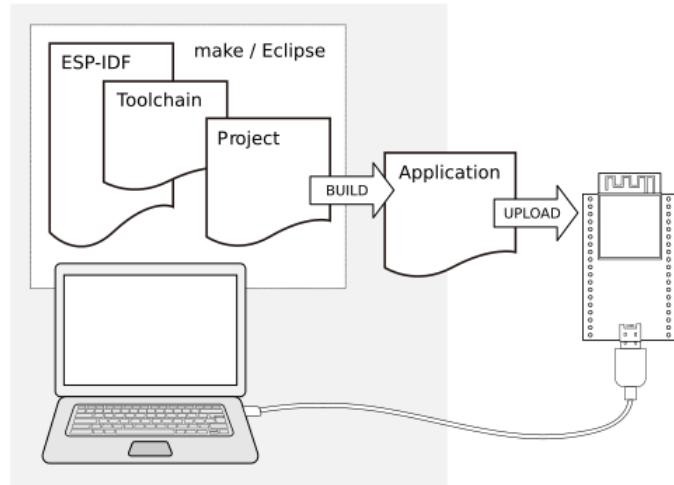


Figure 7: Upload

La syntaxe de ces sketchs ressemble à celle du langage C, mais la sémantique diffère car l'**IDE Arduino nous impose un paradigme "setup/loop" !**

- <https://startingelectronics.org/software/arduino/learn-to-program-course/>
- <https://startingelectronics.org/software/arduino/learn-to-program-course/01-program-structure-flow/>

4.2 setup()/loop()

Il n'y a donc pas de "main()" et l'exécution du programme repose sur deux fonctions dont les noms sont prédéfinis :

1. La fonction `setup()` :

Les énoncés/instructions de cette fonction sont exécutés une seule fois : Chaque fois que le sketch est exécuté !

Ensuite le programme exécute la fonction `loop()`.

Il y a plusieurs "déclencheurs" de l'exécution du sketch :

- Le sketch s'exécutera après avoir été uploadé sur l'ESP

- Ouvrir un terminal (moniteur via liaison série) sur l'ESP provoque un reset et re-exécute le sketch.
- Le bouton reset ("EN") présent sur la carte d'évaluation re-exécute le sketch.
- Une rupture/rétablissement de l'alimentation électrique provoque aussi un reset.

Le programme uploadé est "flashé" dans la mémoire. Il est donc résident !

2. La fonction `loop()` :

Les énoncés/instructions (si il y en a) de cette fonction sont exécutés.

- La fonction `loop()` se repète ... à l'infini !

Cette fonction même vide doit faire partie du sketch sinon il y aura un problème de placement de code en mémoire.

- "It is important to have the `loop()` function in the sketch, even if it is empty, because without it the microcontroller on the Arduino board will try to execute whatever it finds next in memory after the statements in the `setup()` function have been executed.

The microcontroller will try to execute whatever it finds in memory as an instruction, but the `loop()` function prevents it from doing this by keeping program execution in the loop."

4.3 Cross Compilation / Uploading

Le fichier source est compilé sur le PC hôte et le code exécutable produit est à destination de l'ESP32 : on est bien dans un contexte de **cross compilation**.

Une fois que c'est fait, le code exécutable doit être téléchargé sur l'ESP32 : **uploading**.

Ces deux fonctionnalités (compile/upload) sont **accessibles grâce aux boutons de l'IDE**.

4.4 CLI

Si l'interface graphique vous "repousse", vous avez l'alternative CLI (Command Line Interface) :

- pour PlateformIO : <https://docs.platformio.org/en/stable/core/index.html>
- pour l'IDE Arduino : <https://arduino.github.io/arduino-cli/getting-started/>

5 TODO : First Sketch

Dans l'environnement de programmation choisi (moi je reste en Arduino IDE pour l'instant) vous créez un nouveau sketch : "main_loop"

Vous constatez l'utilisation du paradigme "Arduino" : setup() + loop()

```
1  /* Fichier :
2           Step1_WhatIsT/SketchBook/main_loop/main_loop.ino
3 */
4  void setup() {
5      Serial.begin(9600);
6      Serial.println("/** This message will only be displayed on start or reset. **");
7      delay(2000);
8  }
9  void loop() {
10    Serial.println("-- Arduino now at top of main loop. --");
11    Serial.println("-----");
12    delay(2000);
13    Serial.println("Executing instructions in main loop.");
14    delay(2000);
15    Serial.println("Arduino now at bottom of main loop.\r\n");
16 }
```

5.1 Avec Arduino IDE

Avec Arduino IDE,

1. Vous créez un répertoire **main_loop**.
2. Dans ce répertoire, vous créez le fichier du même nom que le répertoire **MAIS suffixé avec ".ino"**.
3. Vous ouvrez un moniteur série (Menu "Outils") pour voir le futur dialogue PC <-> EVM ESP32
4. et vous uploadez votre programme :

```

main_loop | Arduino 1.8.8
Fichier Édition Croquis Outils Aide
main_loop
void setup() {
  Serial.begin(9600);
  Serial.println("**** This message will only be displayed once.");
  delay(2000);
}

void loop() {
  Serial.println("-- Arduino now at top of main loop.");
  Serial.println("-----");
  delay(2000);
  Serial.println("Executing instructions in main loop");
  delay(2000);
  Serial.println("Arduino now at bottom of main loop.");
}

```

/dev/ttyUSB0

Envoyer

```

22:34:40.255 -> Executing instructions in main loop.
22:34:42.285 -> Arduino now at bottom of main loop.
22:34:42.318 ->
22:34:42.318 -> -- Arduino now at top of main loop. --
22:34:42.351 -> -----
22:34:44.279 -> Executing instructions in main loop.
22:34:46.266 -> Arduino now at bottom of main loop.
22:34:46.299 ->
22:34:46.299 -> -- Arduino now at top of main loop. --
22:34:46.366 -> -----
22:34:48.287 -> Executing instructions in main loop.
22:34:50.280 -> Arduino now at bottom of main loop.
22:34:50.313 ->
22:34:50.313 -> -- Arduino now at top of main loop. --
22:34:50.346 -> -----
22:34:52.268 -> Executing instructions in main loop.
22:34:54.402 -> Arduino now at bottom of main loop.
22:34:54.402 ->
22:34:54.402 -> -- Arduino now at top of main loop. --
22:34:54.402 -> -----
22:34:56.257 -> Executing instructions in main loop.
22:34:58.283 -> Arduino now at bottom of main loop.
22:34:58.316 ->
22:34:58.316 -> -- Arduino now at top of main loop. --
22:34:58.349 -> -----
22:35:00.272 -> Executing instructions in main loop.

```

Défilement automatique Show timestamp

Nouvelle ligne 9600 baud Effacer la sortie

Figure 8: main loop

Pour démontrer la persistance du programme dans l'ESP, essayez l'une après l'autre les deux manipulations suivantes :

1. Faire un reset à partir du bouton.
2. Débrancher l'alimentation (par l'USB) de l'ESP et la remettre.

6 Exemples/Tutoriaux/...

Il y a beaucoup d'exemples de sketchs pour l'ESP32 accessibles par l'IDE Arduino.

Menu dans l'IDE :

Fichiers -> Exemples -> Exemples pour ESP32

Il y a en a aussi sur le Web :

- <http://esp32.net/>
- <http://www.lucadentella.it/en/2017/02/07/esp32-9-basic-io/>
- <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>
- <https://www.arduino.cc/en/Tutorial/DigitalPins>
- ...etc ...

7 GPIO : General Purpose Input/Output

L'ESP32 a de nombreuses broches "GPIO" : General Purpose Input/Output.

- En noir sur la figure, il s'agit de ports permettant au circuit de "communiquer" avec son environnement en imposant ou en lisant un potentiel/une tension.

Sur la carte de développement, toutes les broches de l'ESP ne sont pas utilisées (par la carte) et d'autres ne sont pas utilisables (car réservées) :

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

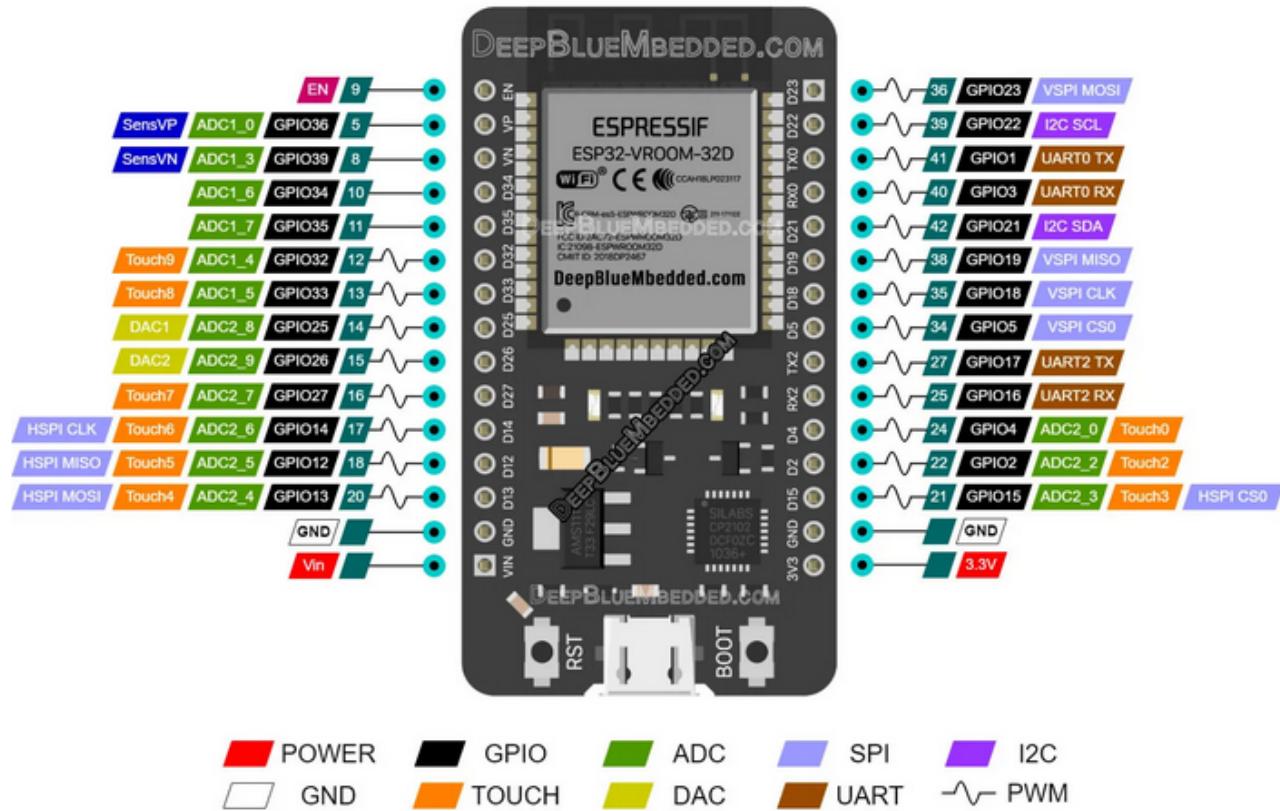


Figure 9: ESP32 Pinout

Dans le code qui suit nous allons utiliser la broche D19 (réf sur la carte) qui correspond au GPIO19 de l'ESP32 : attention de bien brancher la LED sur cette broche !

ATTENTION aussi aux incompatibilités car toutes les cartes EVM du marché n'ont pas forcément le même pinout.

- Les vôtres devraient être identiques.

ATTENTION AUSSI à NE PAS CONNECTER 3.3V et Vin (qui correspond aux 5 volts de l'USB) ... amende à la clé !

7.1 DEL(/LED) : Digital output

L'exemple qui suit nous permet de faire clignoter une LED en utilisant une des sorties "GPIO" (D19) de l'ESP32.

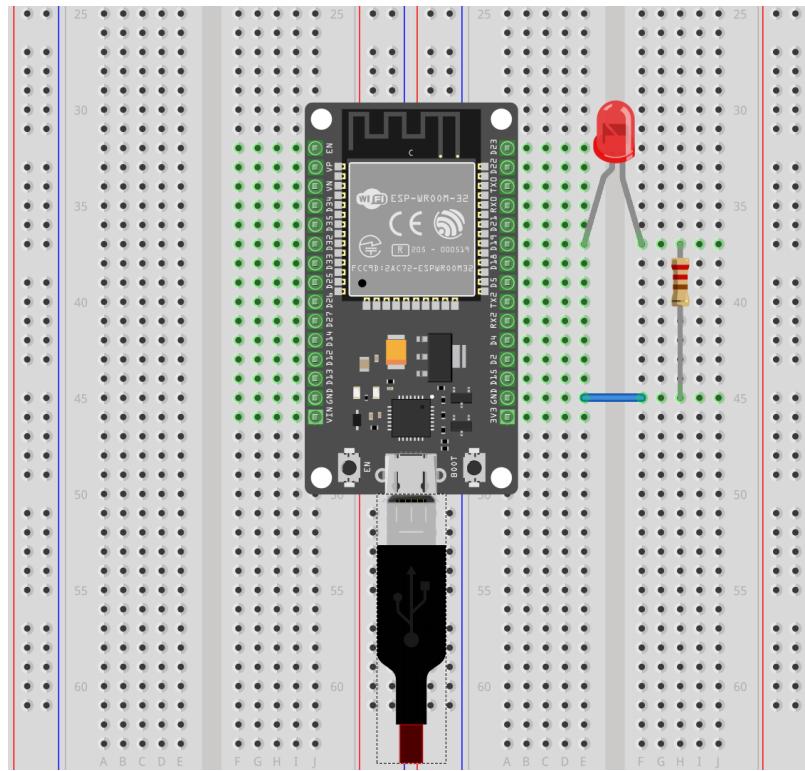


Figure 10: Blink layout

Logiciellement, on utilise une broche du module ESP32 sur laquelle on place tantôt une tension de tantôt de +3.3V, et tantôt de 0V.

- Cette alternance temporisée de tensions fait clignoter la LED.

On remarque qu'il s'agit de deux tensions discrètes.

1. La première (3.3V) correspond au niveau HIGH
2. La seconde (0V) correspond au niveau LOW

7.2 U=RI

Deux "petites" choses à savoir au niveau électrique :

1. **On ne peut pas brancher directement une Led sur la pin du module SANS la mettre en série avec une résistance.**

La Led n'a pas de résistance et compte tenu de la loi d'Ohm ($U = RI$), le courant sortant de la pin devrait théoriquement alors être infini ($I = U/0$).

Cela n'arrivera pas ... c'est plutôt le circuit qui va fondre ... comme un fusible !

Pour cette raison, on met une résistance de $1K\Omega$ en série avec la LED.

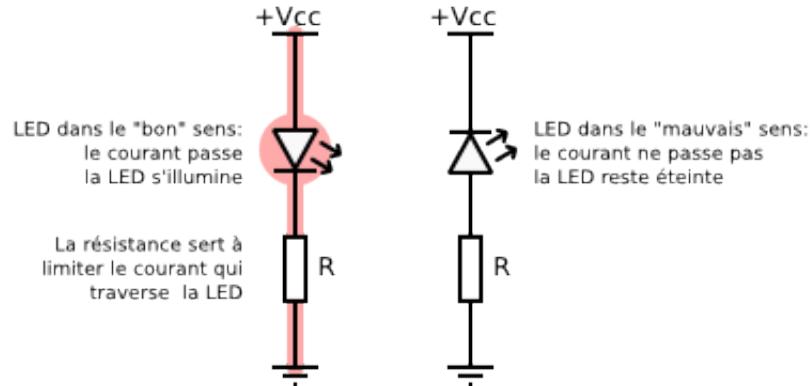


Figure 11: LED and $U=RI$

Petit utilitaire pratique pour les codes couleurs des résistances :

<https://www.digikey.fr/fr/resources/conversion-calculators/conversion-calculator-resistor-color-code>

2. Une **LED (Diode Electroluminescente)** a un sens : voir le dessin !

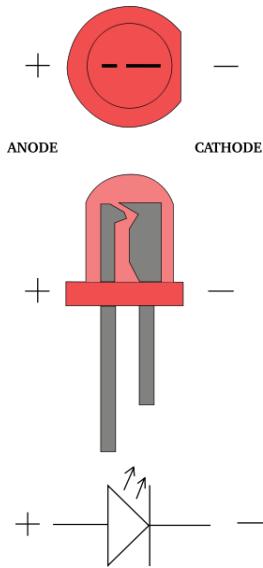


Figure 12: LED and $U=RI$

Sinon le courant ne passe pas et la LED reste éteinte.

Code du firmware

```
1  /* Fichier :  
2           Step1_WhatIsT/SketchBook/gpio_led/gpio_led.ino  
3 */  
4  
5 // ledPin refers to ESP32 GPIO 19 : Broche D19 sur la carte  
6 const int ledPin = 19;  
7  
8 // Setup function runs once when you press reset or power the board  
9 void setup() {  
10     Serial.begin(9600);  
11     Serial.println("** This message will only be displayed on start or reset. **");  
12     delay(2*1000); // wait for two seconds  
13  
14     // Initialize/configure digital pin ledPin as an OUTPUT.  
15     pinMode(ledPin, OUTPUT);  
16 }  
17  
18 // Loop function runs over and over again forever  
19 void loop() {  
20     digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)  
21     delay(1*1000); // wait for a second  
22     digitalWrite(ledPin, LOW); // turn the LED off by making the voltage LOW  
23     delay(1*1000); // wait for a second  
24 }
```

Fonctions de l'API

On utilise deux fonctions basiques de l'API dans ce programme :

1. Configuration d'une broche : fonction **pinMode()**

<https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/>

Les broches ayant plusieurs fonctionnalités possibles, il faut configurer la fonctionnalité utilisée.

Dans l'exemple on veut "imposer"/sortir une tension, d'où le paramètre OUTPUT.

2. Ecriture d'une valeur numérique sur une broche : fonction **digitalWrite()**

<https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/>

L'écriture "numérique", contrairement à l'analogique, ne peut utiliser que deux valeurs possibles : LOW ou HIGH.

Résultat attendu de ce premier exemple

La Led clignote ...

7.3 Light intensity : ADC

Dans ce second exemple, on utilise les capacités de **conversion d'une grandeur analogique en grandeur numérique** offertes par l'ESP.

- En anglais, Analog to Digital Converter : "ADC".
- En français, Conversion Analogique Numérique : "CAN".

Nous utilisons une broche de l'ESP pour "lire" (dans l'exemple précédent l'ESP "écrivait" !) l'intensité lumineuse ambiante obtenue grâce à un "photoresistor" dont la valeur continue du courant qui le traverse est converti en un nombre.

Photo résistance

Le **photoresistor** est un composant dont la résistance varie avec la quantité de lumière qu'il reçoit.

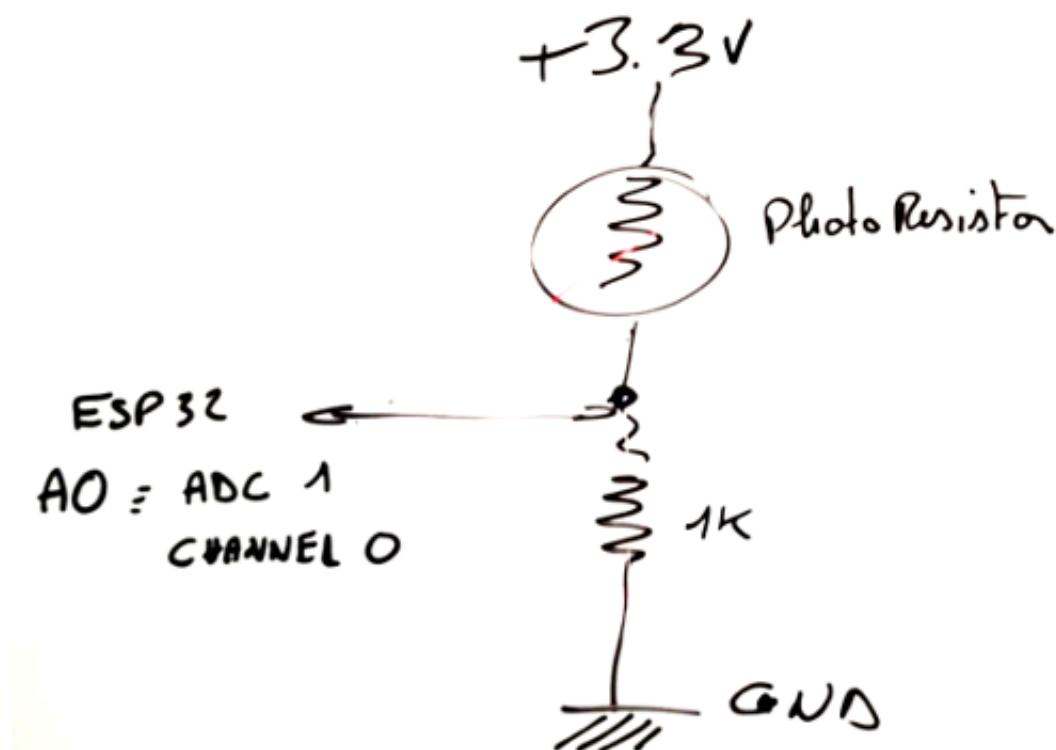


Figure 13: Photoresistor and U=RI

Du coup, le courant dans la branche varie et la tension aux bornes de la résistance 1K aussi !

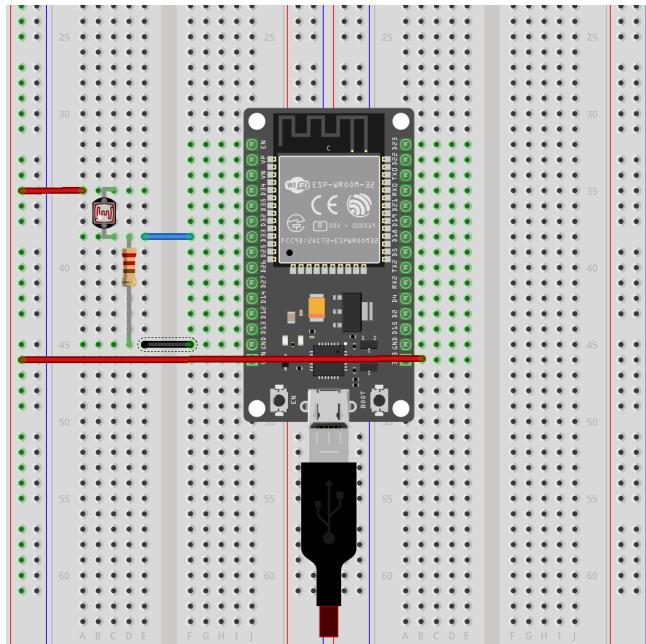


Figure 14: Photoresistor layout

Les broches/fonctions impliquées

Il faut choisir une broche qui permette cette fonctionnalité de **conversion analogique/numérique**.
<https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/adc.html>

Dans le code, vous constatez que nous avons choisi la broche `ADC1_CH5` ou `ADC1_5` aussi notée `A5` (au niveau du code).

Code

```

1  /* Fichier :
2           Step1_WhatIsT/SketchBook/gpio_lum/gpio_lum.ino
3  */
4  void setup(){
5      Serial.begin(9600); // starts the serial port at 9600
6  }
7
8  void loop(){
9      int sensorValue;
10     sensorValue = analogRead(A5);      // Read analog input on ADC1_CHANNEL_5 (GPIO 33)
11                                // Pin "D33"
12     Serial.println(sensorValue, DEC); // Prints the value to the serial port
13                                // as human-readable ASCII text
14     delay(1000); // wait 1s for next reading
15 }
```

Fonctions de l'API

1. Lecture d'une valeur analogique : fonction `analogRead()`

<https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>

2. Affichage sur la console : fonction `println()`

<https://www.arduino.cc/reference/en/language/functions/communication/serial/print/>

N'oubliez pas que le module ESP n'a pas d'écran, l'affichage doit être déporté sur la console reliée par une liaison série.

Résultat attendu

Dans la console de l'IDE, vous devez voir des valeurs entières.

Ces valeurs changent si vous modifiez la luminosité ambiante du photo resistor : **il suffit de mettre le doigt dessus !**

Remarque :

Les entrées analogiques A0,A1,A2,A3,A4,A5 permettent de lire la valeur analogique d'une tension qui peut varier 0 à 5V.

- Cette lecture est discrétisée sur une résolution de 10 bits soit 1024 valeurs, allant de 0 à 1023.

Plus la tension sur la pin sera élevée et plus la valeur lue sera grande.

- Ainsi une tension de 5V sur une broche analogique pourra donner lors de la lecture analogique une valeur égale à 1023,
- une valeur de 0 si la tension est de 0V ou encore une valeur de 511 si la tension est de 2,5V.

7.4 Temperature with DS18B20

Cette partie utilise un capteur de température DS18B20 : <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>



Figure 15: Temperature Sensor DS18B20

Le DS18B20 est un thermomètre numérique qui mesure des températures Celsius avec une résolution programmable : 9-bits à 12-bits.

- Il est alimenté avec une tension qui peut aller de 3.3V à 5V. Ca tombe bien ... c'est ce que sort l'ESP32.
- Le range (la plage) de températures mesurables : -55 C à +125 C
- Une précision de +/- 0.5 C dans le range -10 C to +85 C
- Le **temps de mesure** dépend de la résolution utilisée :
 1. au plus 93.75 ms pour une résolution de 9-bit
 2. au plus 750 ms pour une résolution de 12-bit

Ce n'est pas neutre ! Et cela peut nécessiter de temporiser la boucle de lecture ... pour attendre le capteur.

Ca peut facilement poser des problèmes qui n'existaient pas dans une "informatique" de bureau.

Additionally, it has an alarm functionality with programmable upper and lower temperature trigger points.

- These thresholds are stored internally in non-volatile memory, which means they are kept even if the device is powered off.

Ce capteur communique en utilisant le protocole OneWire :

- Il nécessite **une broche GPIO** (et pas une broche ADC !) de l'ESP32.

Cablage

Le protocole OneWire permet de sérialiser (sur un seul fil/broche) une information de plusieurs bits ! Chaque capteur est identifié par un code série de 64-bits ... une espèce d'adresse MAC. Cela permet de relier plusieurs capteurs sur une même broche GPIO.

Le schéma qui suit montre ce cas de figure où l'ESP pilote 3 capteurs :

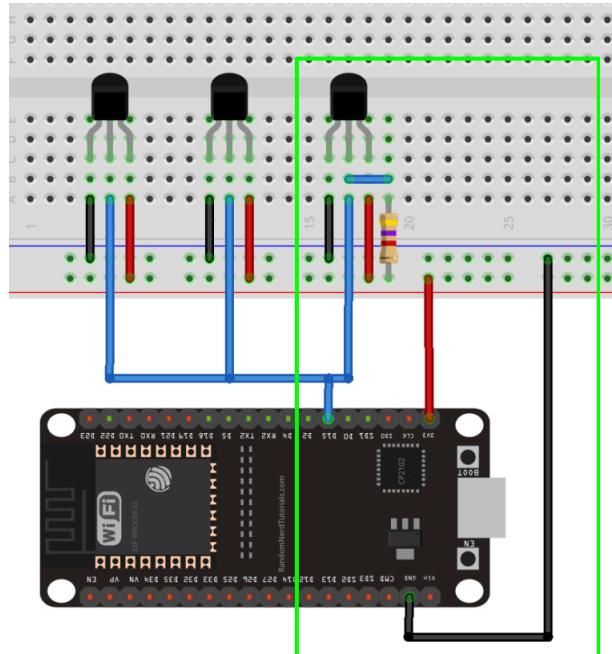


Figure 16: DS18B20 layout

<https://techtutorialsx.com/2018/10/16/esp32-arduino-getting-temperature-from-ds18b20-sensor/>

Pour le TP, nous n'utiliserons qu'un seul capteur par broche. Il suffit de garder le capteur de droite sur le breadboard :

- La résistance est une 4,7 K Ohms !
- Vous utiliserez la broche 23 de l'EVM.

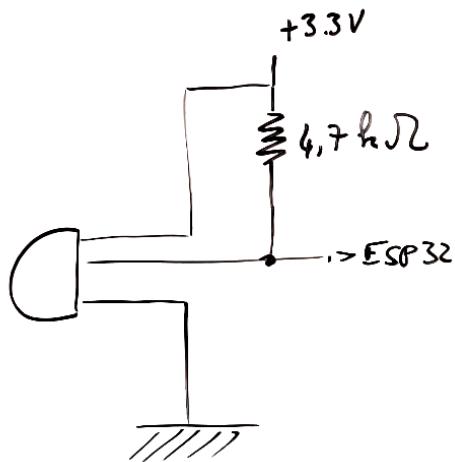


Figure 17: DS18B20 and U=RI

Protocole OneWire et Bibliothèque

Ce protocole série permet de transmettre des données sur **un fil**.

Comme tout périphérique OneWire, le DS18B20 contient un "scratchpad" qui est une sorte de mémoire tampon sécurisée où l'on peut venir lire et / ou écrire des données.

- C'est dans cette mémoire qu'on vient lire les données de mesures et écrire les informations de configuration du capteur.

Figure 7. DS18B20 Memory Map

SCRATCHPAD (POWER-UP STATE)		
Byte 0	Temperature LSB (50h)	} (85°C)-
Byte 1	Temperature MSB (05h)	
Byte 2	T _H Register or User Byte 1*	
Byte 3	T _L Register or User Byte 2*	
Byte 4	Configuration Register*	
Byte 5	Reserved (FFh)	
Byte 6	Reserved	
Byte 7	Reserved (10h)	
Byte 8	CRC*	

*Power-up state depends on value(s) stored in EEPROM.

Figure 18: DS18B20 Memory Map

Le scratchpad du capteur DS18B20 est divisé en quatre parties :

- ① Le résultat de la dernière mesure de température (deux octets),
- ② Deux octets à usages divers (le capteur dispose d'un mode "alarme", ...),

- ③ Le registre de configuration du capteur,
- ④ Une somme de contrôle.

Before uploading the code, you need to install two libraries in your Arduino IDE. Follow the next steps to install those libraries.

1. OneWire library by Paul Stoffregen :

<https://github.com/PaulStoffregen/OneWire/archive/master.zip>

- Unzip the .zip folder and you should get OneWire-master folder
- Rename your folder from OneWire-master to OneWire
- Move the OneWire folder to your Arduino IDE installation libraries folder
- Finally, re-open your Arduino IDE

2. Dallas Temperature library :

<https://github.com/milesburton/Arduino-Temperature-Control-Library/archive/master.zip>

- Unzip the .zip folder and you should get Arduino-Temperature-Control-Library-master folder
- Rename your folder from Arduino-Temperature-Control-Library-master to DallasTemperature
- Move the DallasTemperature folder to your Arduino IDE installation libraries folder
- Finally, re-open your Arduino IDE

Le code

Sur la base des sites Web:

- <https://techtutorialsx.com/2018/10/16/esp32-arduino-getting-temperature-from-ds18b20-sensor/>
- <https://randomnerdtutorials.com/esp32-with-multiple-ds18b20-temperature-sensors/>
- <https://www.carnetdumaker.net/articles/mesurer-une-temperature-avec-un-capteur-1-wire-ds18b20-et-une-carte-arduino-genuine>

```

1  /* Fichier :
2           Step1_What is T/SketchBook/gpio_tempDS18/gpio_tempDS18.ino
3 */
4  #include "OneWire.h"
5  #include "DallasTemperature.h"
6
7  /* URL interessante si pas de Dallas
8   * https://www.carnetdumaker.net/articles/mesurer-une-temperature-avec-un-capteur-1-wire-ds18b20-et-
9   * URL interessante si le temps des conversion est un pb
10  * https://www.scargill.net/reading-dallas-ds18b20-chips-quickly/
11 */
12
13 OneWire oneWire(23); // Pour utiliser une entite oneWire sur le port 23
14 DallasTemperature tempSensor(&oneWire); // Cette entite est utilisee par le capteur

```

```
15
16 void setup(void){
17     Serial.begin(9600);
18     tempSensor.begin(); // Init du capteur et de l'entite OneWire
19 }
20
21 void loop(void){
22     float t;
23     tempSensor.requestTemperaturesByIndex(0); // Le capteur 0 realise une acquisition
24                                         // RMQ : on pourrait avoir plusieurs capteurs
25                                         // sur le port oneWire !
26     t = tempSensor.getTempCByIndex(0);        // On transfert le float qui correspond a
27                                         // temp acquise
28     Serial.print("Temperature: ");
29     Serial.print(t);
30     Serial.print(" C\n");
31
32     delay(1000); // millis ! attention convertir prend du temps !
33 }
```

7.5 Temperature and Moisture with DHT11

Ces capteurs contiennent une puce qui effectue une conversion analogique-numérique et renvoie un signal numérique avec la température et l'humidité.

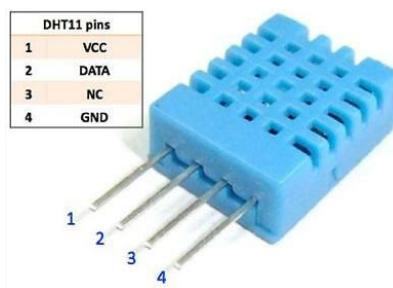


Figure 19: DHT11

Cela les rend très faciles à utiliser avec n'importe quel microcontrôleur : Une simple pin GPIO suffit !

En terme de performances, on retiendra principalement que la résolution est de +/- 2 degrés (pas super comparé au DS18B20 ou au Bosch BME280) et que le temps d'une conversion est de 1 secondes.

- Donc attention au "loop" trop rapide qui voudraient échantillonner plus vite que ça !

Cablage

Le cablage est plus "simple" que celui du DS18B20 mais le capteur est aussi plus cher ... essayez d'en prendre soin !

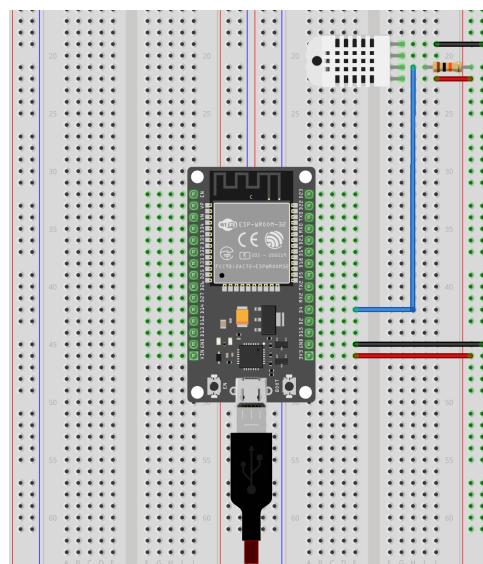


Figure 20: DHT11 Layout

Bibliothèques "BY Adafruit"

Pour accéder au capteur DHT, nous allons utiliser la bibliothèque "DHT d'Adafruit".

Pour utiliser cette bibliothèque, vous devez également installer une dépendance : la bibliothèque "Adafruit Unified Sensor".

- <https://github.com/adafruit/DHT-sensor-library>
- https://github.com/adafruit/Adafruit_Sensor

Si vous les installez depuis l'Arduino IDE ... **attention** à bien choisir les bibliothèques **BY ADAFRUIT** et pas "by n'importe qui" !

Le code

```
1  /* Fichier :
2   Step1_WhatIsT/SketchBook/gpio_dht11/gpio_dht11.ino
3   */
4
5
6
7 // Example testing sketch for various DHT humidity/temperature sensors written by ladyada
8 // REQUIRES the following Arduino libraries:
9 // - DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library
10 // - Adafruit Unified Sensor Lib: https://github.com/adafruit/Adafruit_Sensor
11
12 #include "DHT.h"
13
14 #define DHTPIN 4      // Digital pin connected to the DHT sensor
15 // Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --
16 // Pin 15 can work but DHT must be disconnected during program upload.
17
18 // Uncomment whatever type you're using!
19 #define DHTTYPE DHT11    // DHT 11
20 //##define DHTTYPE DHT22    // DHT 22 (AM2302), AM2321
21 //##define DHTTYPE DHT21    // DHT 21 (AM2301)
22
23 // Connect pin 1 (on the left) of the sensor to +5V
24 // NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
25 // to 3.3V instead of 5V!
26 // Connect pin 2 of the sensor to whatever your DHTPIN is
27 // Connect pin 4 (on the right) of the sensor to GROUND
28 // Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor
29
30 // Initialize DHT sensor.
31 // Note that older versions of this library took an optional third parameter to
32 // tweak the timings for faster processors. This parameter is no longer needed
33 // as the current DHT reading algorithm adjusts itself to work on faster procs.
```

```

34 DHT dht(DHTPIN, DHTTYPE);
35
36 void setup() {
37     Serial.begin(9600);
38     Serial.println(F("DHTxx test!"));
39
40     dht.begin();
41 }
42
43 void loop() {
44     // Wait a few seconds between measurements.
45     delay(2000);
46
47     // Reading temperature or humidity takes about 250 milliseconds!
48     // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
49     float h = dht.readHumidity();
50     // Read temperature as Celsius (the default)
51     float t = dht.readTemperature();
52     // Read temperature as Fahrenheit (isFahrenheit = true)
53     float f = dht.readTemperature(true);
54
55     // Check if any reads failed and exit early (to try again).
56     if (isnan(h) || isnan(t) || isnan(f)) {
57         Serial.println(F("Failed to read from DHT sensor!"));
58         return;
59     }
60
61     // Compute heat index in Fahrenheit (the default)
62     float hif = dht.computeHeatIndex(f, h);
63     // Compute heat index in Celsius (isFahrenheit = false)
64     float hic = dht.computeHeatIndex(t, h, false);
65
66     Serial.print(F("Humidity: "));
67     Serial.print(h);
68     Serial.print(F("% Temperature: "));
69     Serial.print(t);
70     Serial.print(F("°C "));
71     Serial.print(f);
72     Serial.print(F("°F Heat index: "));
73     Serial.print(hic);
74     Serial.print(F("°C "));
75     Serial.print(hif);
76     Serial.println(F("°F"));
77 }

```

7.6 DHT11/DHT22 – Failed to read from DHT sensor

<https://randomnerdtutorials.com/solved-dht11-dht22-failed-to-read-from-dht-sensor/>

Il y a des choses évidentes ... et puis il y aussi le "sampling rate" : $T_e > 2$ secondes !!

8 PWM and ... fan

8.1 Signaux "Pulse Width Modulation" : PWM

FROM : <http://electroniqueamateur.blogspot.com/2019/08/modulation-par-largeur-dimpulsion-pwm.html>

Que ce soit pour contrôler l'intensité lumineuse d'une LED, la vitesse de rotation d'un moteur ou la position angulaire d'un servomoteur, il est souvent utile de créer un signal **modulé en largeur d'impulsion** (MLI, ou PWM pour Pulse Width Modulation).

Un signal PWM alterne entre 0 V et 3,3 V ou 5V :

- Son rapport cyclique est le **pourcentage du temps total pendant lequel il se trouve à sa valeur maximale** (3,3 V ou 5 V ou ...).

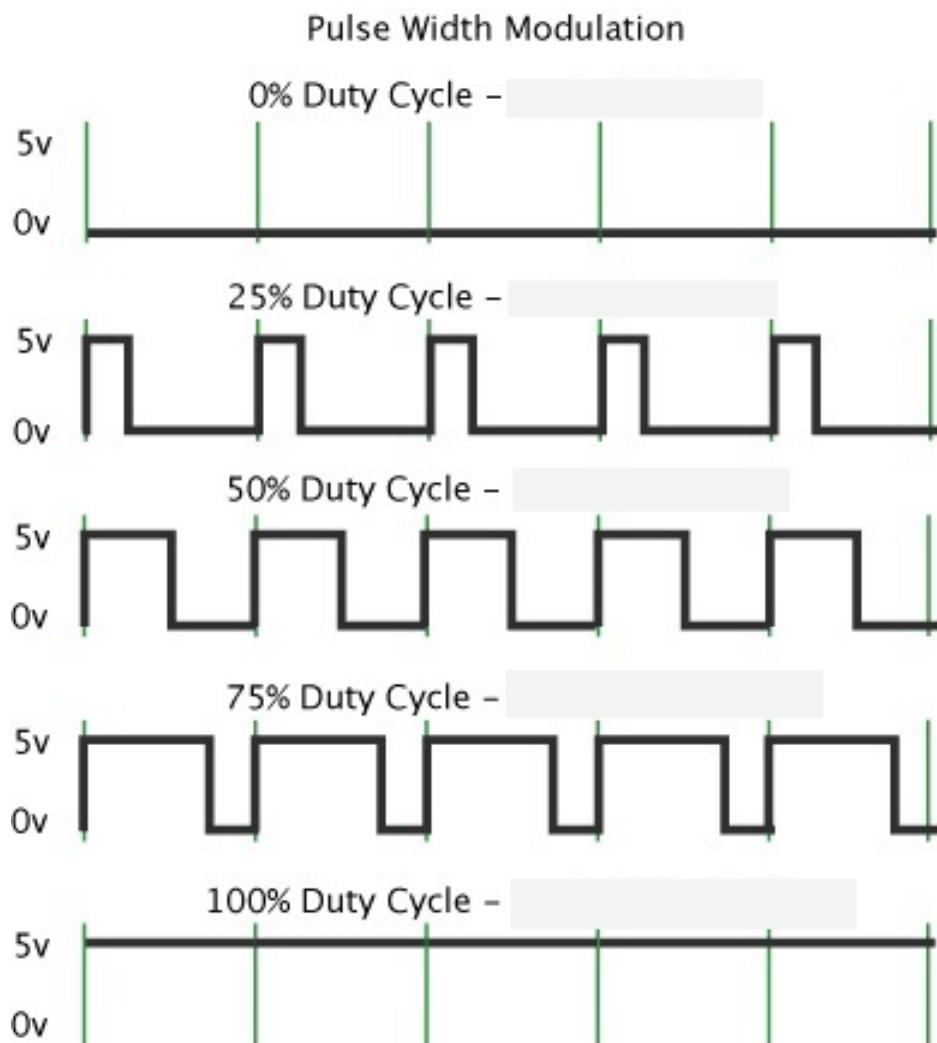


Figure 21: Duty cycles

C'est une façon de régler l'énergie du signal ... que l'on va envoyer à une LED, ou à un ventilateur, ou plus généralement à un moteur.

8.2 Brochage PWM de l'ESP

N'importe quelle broche GPIO capable d'être configurée comme sortie peut être utilisée pour produire un signal PWM. Si nous excluons les broches GPIO 34 à 39 (**qui ne peuvent être configurées qu'en entrée**), cela laisse tout de même (sur les modules que nous utilisons) 21 broches pouvant produire un signal PWM.

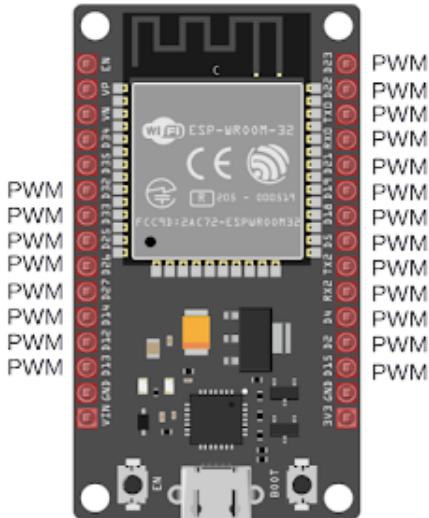


Figure 22: ESP and PWM

8.3 Canaux PWM de l'ESP

Comme nous le verrons plus loin, lorsqu'on désire produire un signal PWM sur une broche, on doit associer cette broche à un des 16 **canaux PWM** au moyen de la fonction :

```
ledcAttachPin()
```

L'ESP32 comporte 16 canaux PWM numérotés de 0 à 15.

8.4 API PWM de l'ESP

Pour générer et contrôler un signal PWM, l'API propose 3 fonctions ledc xyz ("ledc" pour "LED control" puisqu'elles sont prévues avant tout pour contrôler l'intensité lumineuse d'une LED - la valeur de l'intensité dépendant du rapport cyclique utilisé) :

1. ledcAttachPin(broche GPIO, canal)

Cette fonction sert à associer une broche GPIO à l'un des 16 canaux GPIO, qui sont numérotés de 0 à 15.

Par exemple, si on désire produire un signal PWM sur la broche GPIO 27,

```
"ledcAttachPin(27, 0)"
```

fera en sorte que le signal généré par le canal 0 soit associé à la broche 27.

Si on a besoin d'un deuxième signal PWM différent du premier, on utilise pour ce deuxième signal un numéro de broche et un numéro de canal différents.

En général, cette fonction est placée dans la partie `setup()` du programme, puisqu'il est rarement utile de modifier cette assignation en cours d'exécution.

2. `ledcSetup(canal, fréquence, résolution)`

Cette fonction permet de **régler la fréquence et la résolution d'un canal PWM** (Le canal continue d'être un entier situé entre 0 et 15).

- La **fréquence** est en hertz; elle est typiquement de l'ordre du Kilohertz lorsqu'on contrôle la luminosité d'une LED.
Si on veut contrôler un servo moteur, cette fréquence est de l'ordre de $50Hz$
- La **résolution**, qui peut prendre n'importe quelle valeur entière située entre 1 et 13, détermine le nombre de valeurs distinctes pouvant être prises par le rapport cyclique.

$$\text{rapports cycliques possibles} = 2^{resolution}$$

Par exemple, l'expression "`ledcSetup(0, 5000, 12)`" règle le canal PWM numéro 0 à une fréquence de 5000 Hz, avec une résolution de 12 bits.

Si vous n'avez pas besoin de modifier la fréquence pendant l'exécution du programme, vous placerez probablement cette fonction dans la partie `setup()`.

Quel est l'avantage d'utiliser une résolution plus faible que le maximum permis ?

➤ Atteindre une fréquence plus élevée !

La fréquence maximale permise pour un signal PWM généré par l'ESP32 est de 40 MégaHertz !

Mais à cette fréquence, la seule résolution possible est de 1 bit (permettant un rapport cyclique de 50% ... et rien d'autre).

Plus la fréquence est grande, plus la résolution maximale possible est petite.

3. `ledcWrite(canal, rapport cyclique)`

Cette fonction permet de régler le rapport cyclique du signal PWM du canal spécifié.

Par exemple, "`ledcWrite(0, 250)`" règle à 250 le rapport cyclique du canal 0.

Attention : le résultat de cette commande dépend de la résolution préalablement réglée avec la fonction `ledcSetup`.

- si la résolution est de 12 bits, $250/2^{12}$ correspond à un rapport cyclique d'environ 6%;
- si la résolution est de 10 bits, $250/2^{10}$ correspond plutôt à un rapport cyclique de 24%,
- et le rapport cyclique sera de 98% ($250/2^8$) à une résolution de 8 bits.

8.5 Sketchs basiques

Rapport Cyclique : 50%

```
void setup() {  
  
    ledcAttachPin(18, 0);  
    ledcSetup(0, 5000, 12);  
    ledcWrite(0, 2048);  
}
```



Figure 23: Duty cycle :50%

Rapport Cyclique : 25%

```
void setup() {  
  
    ledcAttachPin(18, 0);  
    ledcSetup(0, 5000, 12);  
    ledcWrite(0, 1024);  
}
```

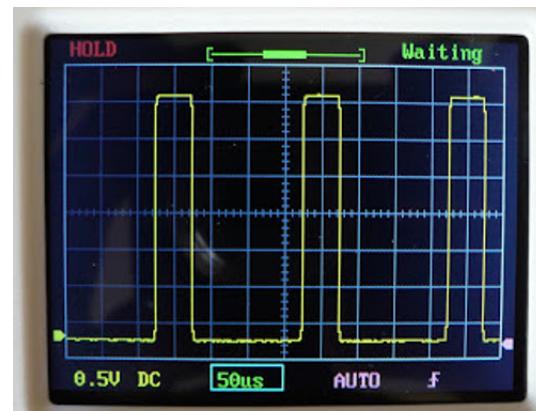


Figure 24: Duty cycle : 25%

8.6 Le ventilateur

On veut utiliser PWM pour contrôler un ventilateur (parmi les deux modèles suivants) !

Rasberry Pi4 case fan

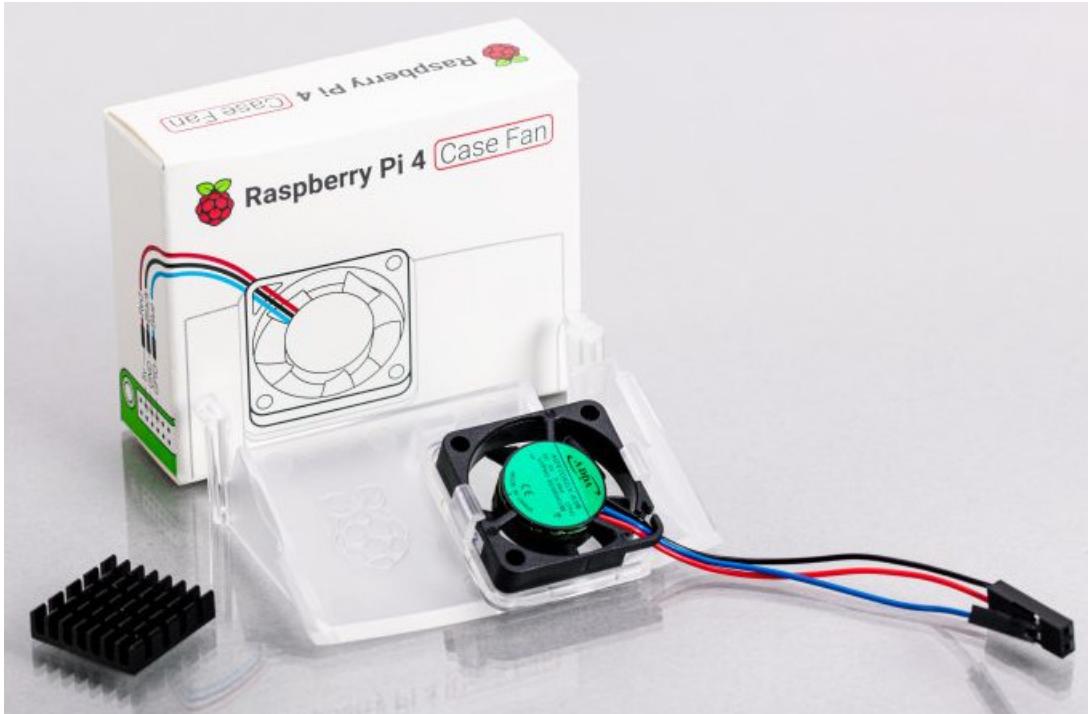


Figure 25: Rasberry Pi4 case fan

Noctua fan



Figure 26: Noctua fan

<https://noctua.at/fr/products/fan/nf-a6x25-5v-pwm>

Ce type de ventilateur sort 4 fils :

1. Noir : GND
2. Jaune : +5V
3. Vert : Tachymétrie => pin 26
4. Bleu : PWM => pin 27

Ce fils est absent sur le Raspberry Pi4 case fan.

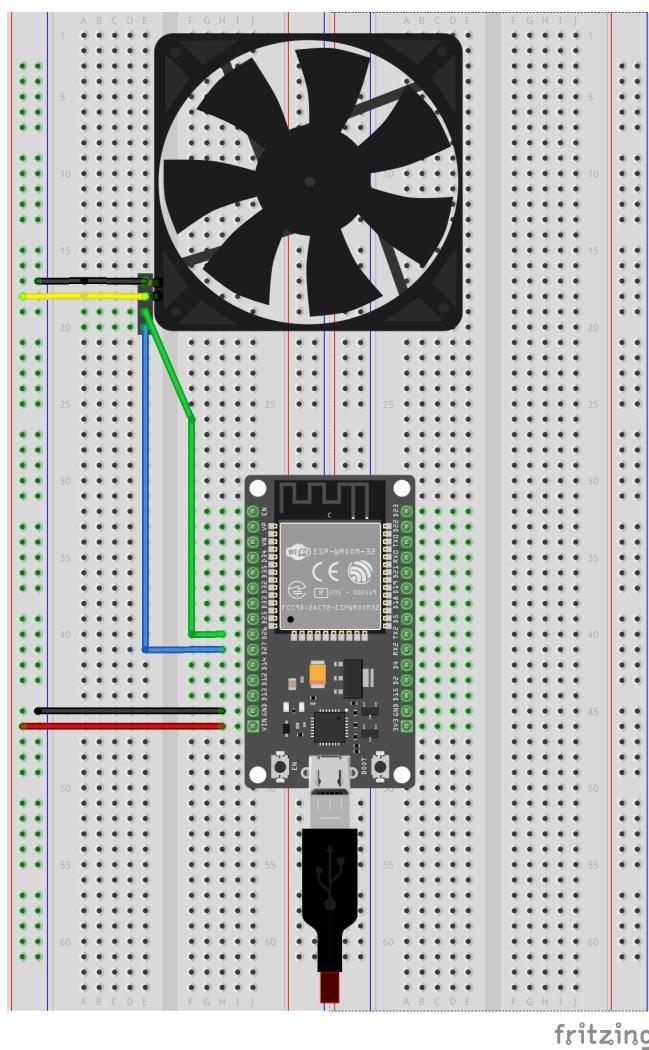


Figure 27: Fan layout

Sur l'entrée "PWM" du ventilateur, l'ESP va générer un signal à 25KhZ avec différents rapports cycliques permettant de changer la vitesse de rotation du ventilateur.

La sortie "Tachymétrie" du ventilateur est un signal produit par le ventilateur qui permet de calculer sa vitesses de rotation (voir les spécifications) :

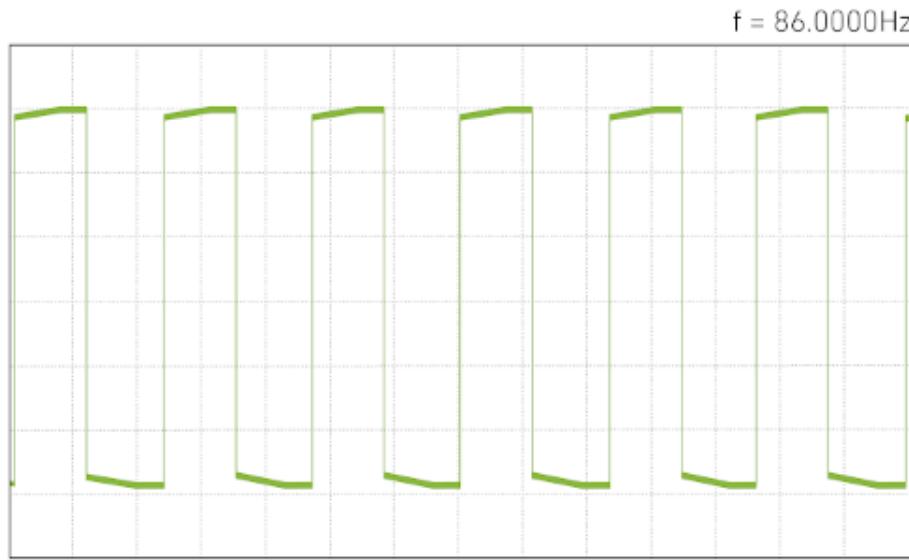


Figure 28: Tachymetry

8.7 Code

<https://randomnerdtutorials.com/esp32-pwm-arduino-ide/>

```
1  /* Fichier :
2   Step1_What is T/SketchBook/gpio_pwm/gpio_pwm.ino
3 */
4 */
5 /*
6   Creation d'un signal PWM avec l'ESP32
7   Plus d'infos:
8   https://electroniqueamateur.blogspot.com/2019/08/modulation-par-largeur-dimpulsion-pwm.html
9
10  La broche GPIO 27 est liée au canal PWM 0,
11  le canal 0 a une fréquence de 5000 Hz et une résolution de 12 bits.
12  Finalement, j'ai réglé le rapport cyclique à 50% (2048 est la moitié de 2^12 )
13 */
14
15 int numberKeyPresses = 0;
16
17 void IRAM_ATTR isr() { // Interrupt Handler
18     numberKeyPresses++;
19 }
```

```

20 }
21
22 void setup() {
23     Serial.begin(9600);
24
25     Serial.println("\nFan speed test using PWM !");
26
27     // Interrupt configuration
28     pinMode(26, INPUT_PULLUP); // broche 26
29     attachInterrupt(26, isr, FALLING); // handler
30
31     // PWM configuration
32     ledcAttachPin(27, 0); // broche 27, canal 0.
33     ledcSetup(0, 25000, 8); // canal = 0, frequence = 25000 Hz, resolution = 8 bits
34                                     // donc 255 rapport cycliques differents possibles
35     ledcWrite(0,255);
36 }
37
38
39 void loop() {
40     int nb_state = 12;
41     int rcv[nb_state] = {0, 64, 64,
42                          0, 127, 127,
43                          0, 191, 191,
44                          0, 255, 255}; // Rapports cycliques si 8 bits
45                                     // 255 => 100%, 127 => 50% et 0 => 0%
46     static uint32_t tick = 0;
47     static int elapsed;
48
49     static int current = 0; // etat courant
50
51     elapsed = millis() - tick;
52     if (elapsed > 10000) { // Toutes les 10 sec on change de Rapport Cyc
53
54         Serial.printf("Fan speed = %f RPM\n", (numberKeyPresses/10.0)*30); // 30 = 60/2 cf doc
55         numberKeyPresses = 0;
56
57         ledcWrite(0, rcv[current]); // canal = 0, rapport cyclique
58         Serial.printf("Rapport Cyc = %lf %%\n", (rcv[current] / 255.0) * 100);
59
60         current += 1; // Prochain rapport cyclique
61         current %= nb_state;
62
63         tick = millis();
64     }
65 }

```

Dans le setup,

- la pin 26 de l'ESP est configurée pour déclencher une interruption sur front descendant.

La fonction "isr" est désignée comme callback.

- la pin 27 de l'ESP est associée au canal PWM 0.

Sur la base d'un signal à 25KhZ (voir les spécifications) nous allons exploiter différents rapports cycliques ... pour illustrer la variation de la vitesse de rotation.

Dans la boucle, toutes les 10 secondes nous changeons le rapport cyclique envoyé au ventilateur.

9 Led Strip (ou bande de LEDs)

9.1 Mini Led Strip Adressable

Voilà une version courte, moins chère et surtout plus facile à alimenter que la "vraie" Led Strip" qui va suivre et que l'on trouve fréquemment dans le commerce.

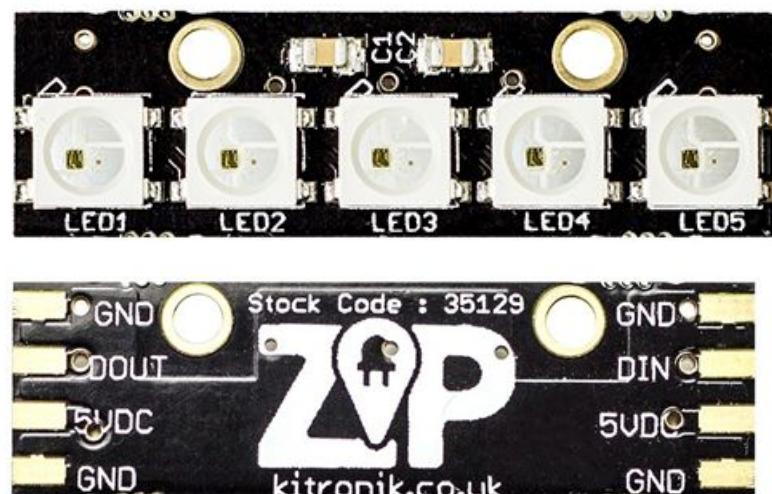


Figure 29: RGB Led strip

Cette bande contient 5 LEDs et s'alimente directement sur l'ESP.

Vous branchez :

- le "5V de la bande" à la broche "Vin de l'ESP"
- Le GND au GND
- et le DIN à la broche de l'ESP que vous voulez : la 13 par exemple !

Vous remarquez que le RGB "passe" par une seule broche : multiplexage !

Le programmation est très intuitive et repose sur :

- <https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use>

```
1  /*
2   * fichier ledring.ino
3   * https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use
4   */
5
6  #include <Adafruit_NeoPixel.h>
7  #define PIN      13
8  #define NUMLEDS  5
9  Adafruit_NeoPixel strip(NUMLEDS, PIN, NEO_GRB + NEO_KHZ800);
```

```

10
11 void setup() {
12   strip.begin();
13   //needed, otherwise the initialization is not done until we run through the strip
14   //which will leave some led's green
15   delay(1);
16
17   for(int i=0; i<1; i++) {
18     //turn color to red
19     strip.setPixelColor(i, strip.Color(255, 0, 0));
20   }
21   for(int i=1; i<4; i++) {
22     //turn color to green
23     strip.setPixelColor(i, strip.Color(0, 255, 0));
24   }
25   for(int i=4; i<NUMLEDS; i++) {
26     //turn color to green
27     strip.setPixelColor(i, strip.Color(0, 0, 255));
28   }
29   strip.show();
30 }
31
32 void loop(){}

```

9.2 Led Strip RGB

Cette partie est optionnelle car vous n'avez pas par défaut ces composants !

Cet exemple montre l'utilisation d'une led strip pour réaliser une "visualisation" un peu plus "jolie" que la simple Led.



Figure 30: RGB Led strip

Le principe de la LED reste mais c'est légèrement plus complexe au niveau électrique pour deux raisons :

1. Il y a de la couleur et donc trois composantes : R (Rouge), G (Vert), B (Bleu).
2. L'intensité lumineuse permise par "toutes" les leds de la bande induit des courants qui peuvent poser problème.

On peut pas forcément demander à l'alimentation USB et à aux broches de l'ESP de sortir la quantité de courant nécessaire. Nous allons rester dans des configurations "simples" par exemple en limitant le nombre de Leds sur la bande.

Dans le circuit de la figure qui suit, la bande est alimenté par une source extérieure qui fournit le 5V. Comme on va utiliser des bandes courtes, ou pourra utiliser la broche "Vin" de l'ESP qui fournit le 5V du cable USB alimentant l'ESP.

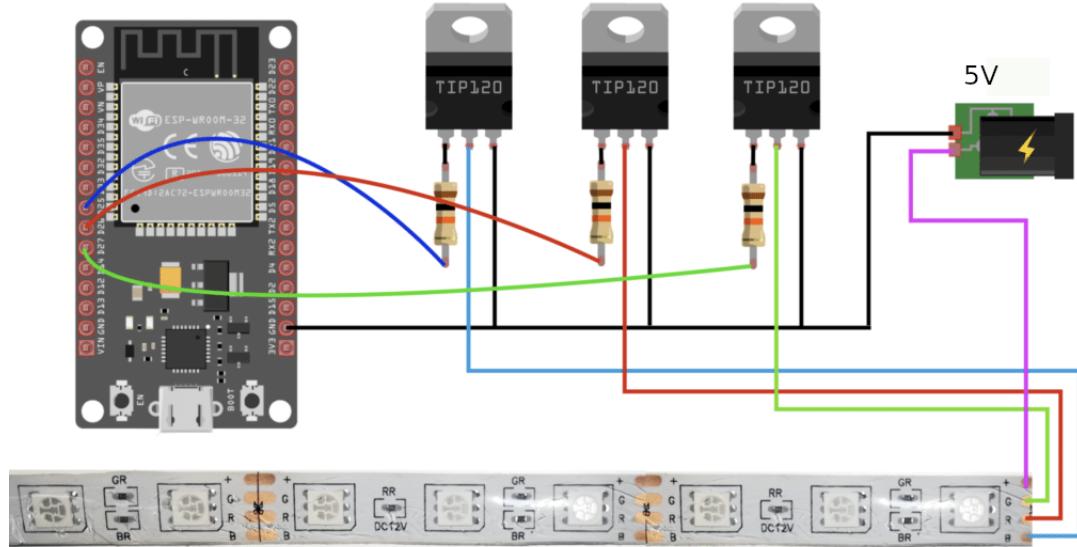


Figure 31: Strip layout

Les transistors utilisés sont des TIP120 ... Attention au "pinout" !

- <https://www.alldatasheet.com/datasheet-pdf/pdf/54789/FAIRCHILD/TIP120.html>

Les transistors sont utilisés comme des "interrupteurs commandés" par l'ESP.

- Selon la tension sur la "base" du transistor, l'émetteur reçoit plus ou moins de courant du collecteur qui est connecté à l'alimentation théoriquement capable de fournir assez courant à la bande.

Les résistances sont des 10KOhms ... <https://www.digikey.fr/fr/resources/conversion-calculators/conversion-calculator-resistor-color-code>.

9.3 Code

Au niveau du code, j'ai choisi de faire très simple ... vous retrouvez un peu de PWM.

```

1  /* Fichier :
2           Step1_WhatIsT/SketchBook/gpio_ledstrip/gpio_ledstrip.ino
3  */

```

```

4
5 #include "colors.h"
6
7 // Variable to store the HTTP request
8 String header;
9
10 // Red, green, and blue pins for PWM control
11 const int redPin = 13;      // 13 corresponds to GPIO13
12 const int greenPin = 12;    // 12 corresponds to GPIO12
13 const int bluePin = 14;     // 14 corresponds to GPIO14
14
15 // Bit resolution  $2^8 = 256$ 
16 const int resolution = 8;
17
18 // Setting PWM frequency, channels and bit resolution
19 const int freq = 5000;
20 const int redChannel = 0;
21 const int greenChannel = 1;
22 const int blueChannel = 2;
23
24
25 void setup() {
26   Serial.begin(9600);
27
28   // configure LED PWM functionalities
29   // and attach the channel to the GPIO to be controlled
30   ledcSetup(redChannel, freq, resolution);
31   ledcAttachPin(redPin, redChannel);
32   ledcSetup(greenChannel, freq, resolution);
33   ledcAttachPin(greenPin, greenChannel);
34   ledcSetup(blueChannel, freq, resolution);
35   ledcAttachPin(bluePin, blueChannel);
36 }
37
38 void loop(){
39   static int i= 0 ;
40
41   Serial.printf("%d : %d %d %d\n", i, rgball[i][0], rgball[i][1], rgball[i][2]);
42   ledcWrite(redChannel, rgball[i][0]);
43   ledcWrite(greenChannel, rgball[i][1]);
44   ledcWrite(blueChannel, rgball[i][2]);
45
46   i = i+2;
47   i = i%256;
48   delay(1000);
49 }
```

10 TODO : YOUR work !

Après avoir essayé ces exemples et acquis une petite maîtrise des Inputs/Outputs de l'ESP, vous allez réaliser une régulation de température ambiante pour une pièce d'habitation.

Respectez les brochages indiqués sinon cela risque de poser problème pour tester/évaluer votre travail!

- En version moins diplomatique, si je ne peux pas tester alors je mets zéro !

Rappel : DEV KIT V1 (36 pins) dans <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>

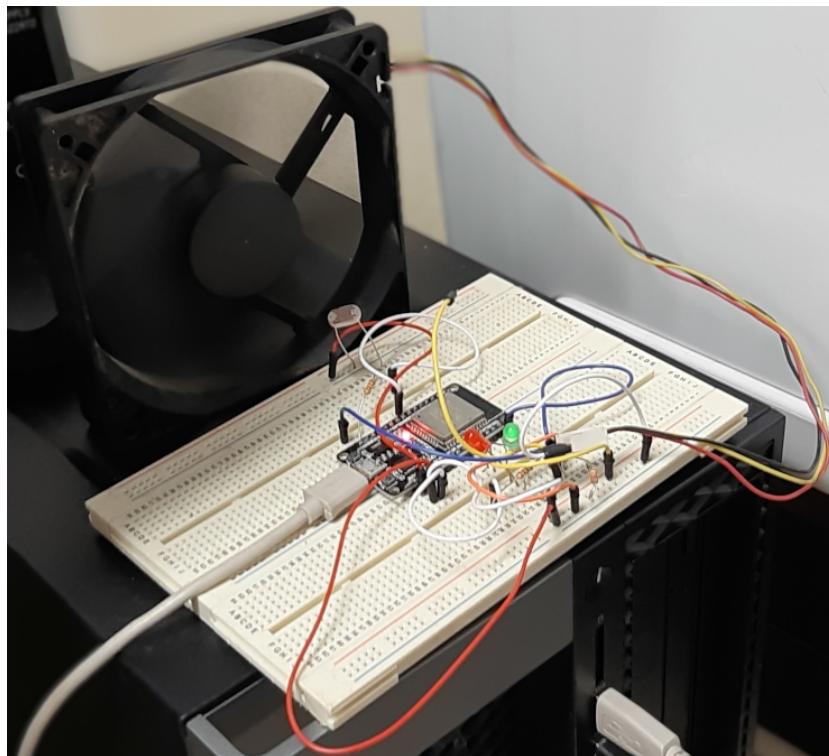


Figure 32: Temperature Sensing & Regulation

10.1 Temperature Sensing & Regulation

Vous mesurez la température ambiante grâce au capteur de température en **broche 23** et vous affichez sa valeur sur le moniteur série de l'ESP.

On suppose que vous pouvez agir sur l'évolution de la température grâce à

- une climatisation (modélisée par une LED verte en GPIO 19)

Si vous avez un ventilateur en GPIO 27, vous le mettrez en marche dès que le seuil haut de température sera dépassé.

Une progressivité de la vitesse de rotation selon l'élévation de température (au delà du seuil) sera appréciée !

- et un radiateur (modélisé par une LED rouge en GPIO 21).

Pour bien voir le résultat, on utilise des seuils :

- Si la température est supérieure à un seuil haut (SH) vous climatisez.
- Si la température est inférieure à un seuil bas (SB) vous chauffez.
- Entre les deux tout est éteint (donc les LEDs sont éteintes).

Il est difficile de définir ces seuils avant les tests et la température du jour ... le code doit être paramétrable ! et lisible !

10.2 Detection Process

En utilisant la bande de Leds (en broche 13), vous pouvez accompagner votre décisionnel d'un affichage.

- Si la température est au dessus du seuil haut, vous allumez ROUGE
- Entre les seuils vous allumez ORANGE
- Si la température est inférieure au seuil bas, vous allumez VERT

Un fichier bien utile : https://www.w3schools.com/colors/colors_picker.asp

Pour définir les couleurs :

- soit à partir du fichier "colors.h"
- soit sur le Web en cherchant comment fabriquer un RGB à partir d'une valeur ...
<https://www.google.com/search?q=Calculation+of+RGB+values+given+min+and+max+values+>

10.3 Fire Detection

On veut que l'ESP permette de détecter une situation d'incendie dans une zone "proche". Suffisamment proche pour que la variation d'intensité lumineuse soit significative ... même si elle ne sera peut être la seule information pertinente dans la détection.

- C'est le capteur de lumière (ADC1 Channel 5 / GPIO33) qui donnera l'information de l'intensité lumineuse.

On est dans un démarche "edge computing" et dans le cas d'une détection par l'ESP, il allumera la led "onboard" (Pin = 2) pour indiquer qu'il a signalé un incendie en faisant remonter l'information à un site plus central.

11 Conclusion

Plusieurs objectifs liés à cette première session ... et à atteindre :

1. Petit "tour" de la programmation basique de l'ESP32.
2. Quelques "périphériques" et comment on les connecte ?
3. Une ébauche de ce qu'est un objet ou du moins l'aspect "connexion au monde réel" (en entrée ou en sortie) qu'il permet de réaliser.

Ces notions sont des pré-requis pour la suite ... organisez votre code !!!!! car il va vous servir pour la suite !

12 A suivre ...

On a pour l'instant, fait fonctionner l'"objet" de façon complètement autonome.

... et si maintenant on le connectait "au monde" !