



Microcontrôleurs - Entrées, sorties (GPIO)



[Mise à jour le : 19/2/2025] **En cours de rédaction**

- **Source**

- [Arduino](#)
- [Raspberry Pi pico](#)
- [esp32](#)

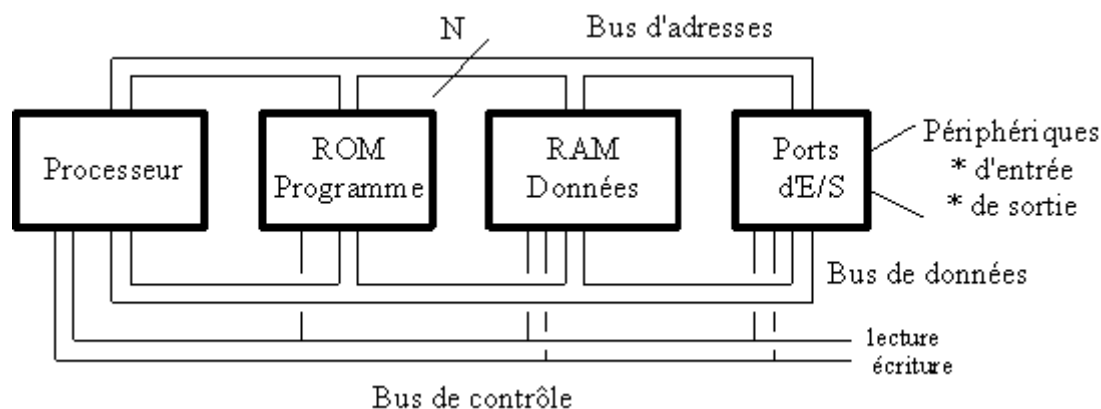
CARTES DE TEST

Les exemples de code de cette page ont été testés sur une [Arduino Uno](#), une [Arduino MKR Wifi 1010](#), une [ESP32 Feather Huzzah](#) et sur une [Raspberry Pi Pico](#).

1. Généralités

- Vidéo : [Microcontrôleur : Comment ça marche ? - SILIS Electronique](#)

« D'un point de vue fonctionnel, dans un système à base de **microcontrôleur**¹⁾, on appelle **entrées-sorties** les échanges d'informations entre le processeur et les périphériques qui lui sont associés. De la sorte, le système peut réagir à des modifications de son environnement, voire le contrôler. Les entrées, sorties sont parfois désignées par l'acronyme **I/O**, issu de l'anglais **Input/Output** ou encore **E/S** pour **Entrées/Sorties**. » [Wikipédia](#)



GPIO

« D'un point de vue matériel, un **microcontrôleur** dispose de **broches**²⁾ pouvant être contrôlées par un logiciel. Elles peuvent se comporter comme des entrées ou des sorties, d'où le nom "Entrée / Sortie à usage général", ou **GPIO** (**G**eneral **P**urpose **I**nput **O**utput. Le nombre de broches d'un microcontrôleur étant limité, il est fréquent d'avoir **plusieurs fonctionnalités sur une même broche**. » [Wikipédia](#)

1.1 Exemples de microcontrôleurs

- [ATMEGA328P](#)
- [RP2040](#)
- [RP2050](#)
- [esp32](#)
- [Documentation](#) sur le site **Microchip**.
- [Documentation](#) sur le site **Raspberry Pi**.
- [Documentation](#) sur le site **Raspberry Pi**.
- [Documentation](#) sur le site **ESP32**.

1.2 Cartes de prototypage

- [Arduino UNO R3](#)
- [Rpi pico & picoW](#)
- [Rpi pico 2 & pico2W](#)
- [ESP32](#)
- [Documentation](#) sur le site Arduino.
- Les broches du microcontrôleur sont reliées électriquement aux broches des connecteurs de la carte. Un **SYNOPTIQUE** identifie ces liens.
Exemple : on voit sur ce schéma que la broche 19 du microcontrôleur est reliée avec la broche 13 du connecteur de la carte.
ATTENTION : dans l'environnement de programmation Arduino, le programme fait référence à la connectique de la carte et non à celle du microcontrôleur !
- [Documentation](#) sur le site **Raspberry Pi**.
- [Documentation](#) sur le site **Raspberry Pi**.

A faire

2. Entrées, sorties numériques

2.1 Généralités

Pour éviter de faire référence à des valeurs électriques (tension ou intensité), on définit souvent l'état d'un signal numérique en utilisant la logique booléenne.

- **true** (« **1** » logique ou **HIGH**)
- **false** (« **0** » logique ou **LOW**) .

- [ATMEGA328P](#)
- [RP2040](#)
- [RP2050](#)

- ($V_{cc} = 5V - 25^{\circ}C$)

Niveau logique	Signal de sortie (Output)	Signal d'entrée (Input)
LOW	0V à 0,5V (VoL) ³⁾ pour 0 à 20mA (IoL)	0V à 2,2V (ViL) ⁴⁾
HIGH	4,5V à 5V (VoH) ⁵⁾ pour 0 à 20mA (IoH)	2,7V à 5V (ViH) ⁶⁾

LIMITES A RESPECTER

Le courant maximum en sortie sur une broche ne doit pas dépasser **40mA**. Au total, il ne doit pas dépasser **200mA**.

- A rédiger
- A rédiger

2.2 Sortie numérique

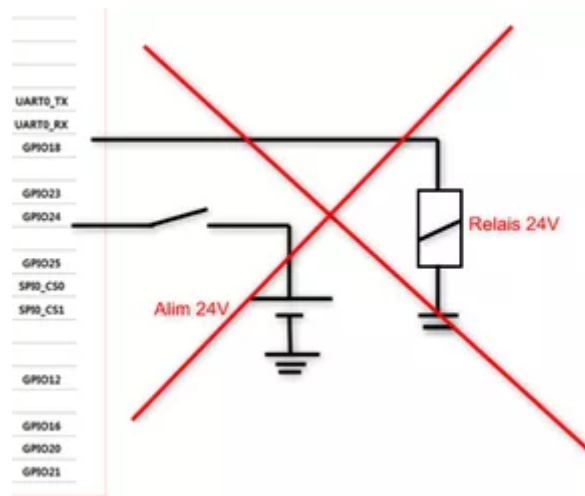
2.2.1 Généralités

Lorsqu'une broche est configurée en **sortie numérique**, le microcontrôleur délivre des **signaux logiques "0"**(LOW) ou **"1"**(HIGH).

PRECAUTIONS D'UTILISATION

Une sortie numérique est fragile. Ne **JAMAIS** la relier à un générateur

Une sortie numérique délivre **très peu de puissance** (quelques centaines de mW). Il n'est donc pas possible de la relier directement à un actionneur (moteur). Il est nécessaire de placer une interface de puissance (hacheur, relais) entre elle et l'actionneur à commander.



2.2.2 Programmation

Les programmes suivants sont codés :

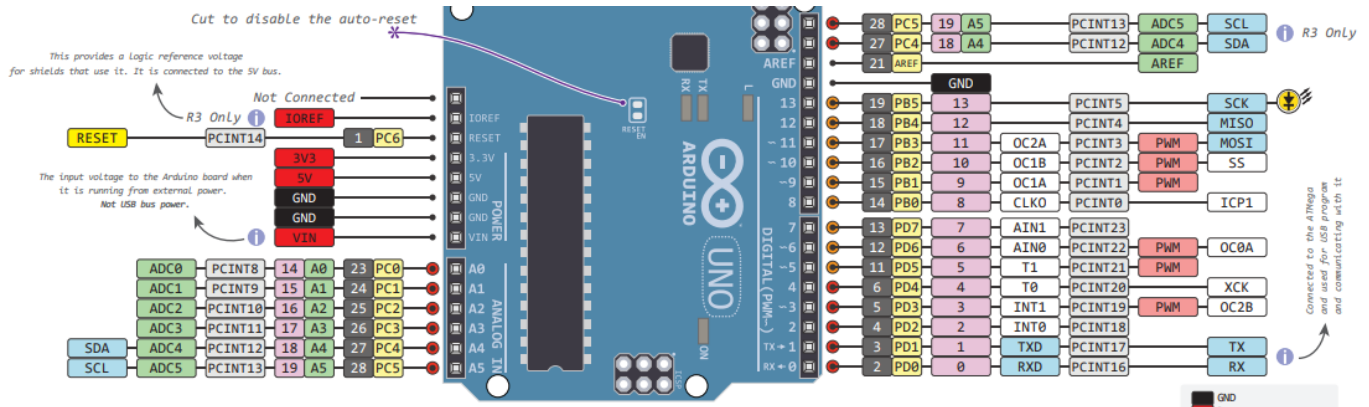
- En langage **C** sur Arduino Uno ou compatible
- En langage **MicroPython** sur Raspberry Pi Pico et ESP32

- [Arduino UNO R3](#)
- [Rpi pico & picoW](#)
- [ESP32](#)

• a. Présentation

Toutes les broches d'E/S de la carte Arduino Uno peuvent servir d'entrées, sorties numériques. Cela inclut les broches numérotées entre **0** et **13**, mais aussi les broches **A0** à **A5**. Le symbole ~ identifie les broches capables de délivrer un signal numérique **PWM** (Pulse **W**ith **M**odulation).





SCHEMA DE LA CARTE

Téléchargeable [ici](#).

- *b. Configuration d'une broche en sortie*
 - **Ressource** : [pinMode\(\)](#) sur le site [arduino.cc](#)

CONFIGURATION

Configurer une broche signifie qu'on précise qu'elle doit se comporter soit comme une entrée d'information, soit comme une **sortie d'information**.

*.cpp

```
// Exemple 1
// -----
// Les instructions contenues dans la fonction setup() s'exécutent une
// fois
void setup() {
    pinMode(12,OUTPUT); // Associé à OUTPUT pinMode force la broche 12 de
// la carte Arduino Uno à se comporter comme une sortie
}

// Exemple 2
// -----
// L'utilisation de la directive "define" améliore la lisibilité des
// programmes
#define led 12

void setup() {
    pinMode(led,OUTPUT); // On précise que la broche 12 de la carte
// Arduino Uno se comporte comme une sortie et est connectée à une LED
}
```

- *c. Ecriture d'un état logique sur une broche*

- **Ressource** : [digitalWrite\(\)](#) sur le site [arduino.cc](#)

ECRITURE

Un état logique **haut(HIGH)** ou **bas(LOW)** peut être **écrit** sur une broche préalablement configurée en sortie.

*.cpp

```
// Exemple 1
// -----
// Les instructions contenues dans la fonction setup() s'exécutent une
// fois
void setup() {
    pinMode(12,OUTPUT);    // Associé à OUTPUT pinMode force la broche
    12 de la carte Arduino Uno à se comporter comme une sortie
}

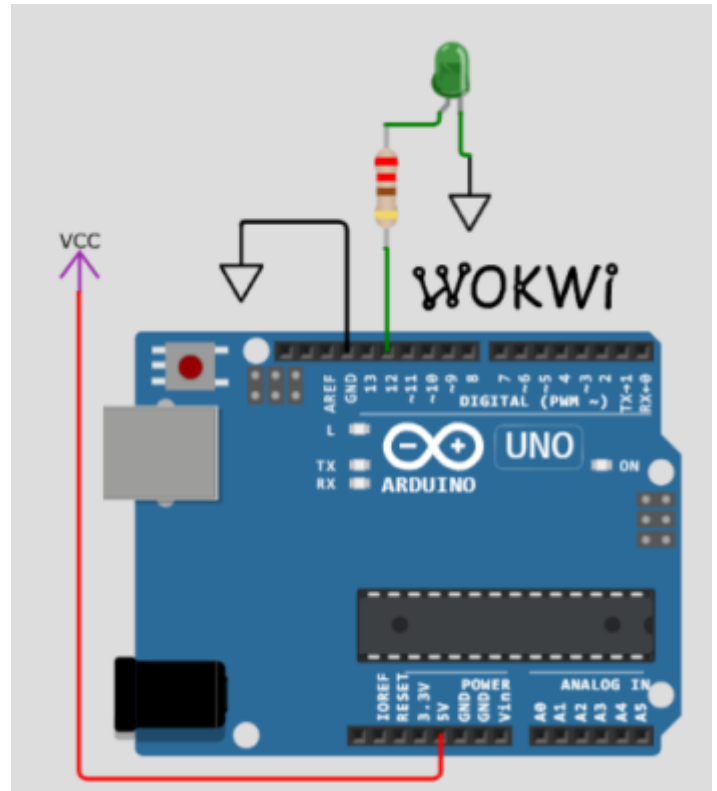
// Les instructions contenues dans la fonction loop() s'exécutent tant
// que le microcontrôleur est sous tension
void loop() {
    digitalWrite(12, HIGH); // La broche 12 présente un état logique
    haut (par exemple 5V)
}

//Exemple 2
// -----
// L'utilisation de la directive define améliore la lisibilité des
// programmes
#define led 12

void setup() {
    pinMode(led,OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH); // On applique un état logique haut (par
    exemple 5V) à la LED => elle s'éclaire
}
```

d. Synthèse: blink



*.cpp

```
// Faire clignoter une led connectée à la broche 12 de la carte Arduino
// Matériels : Carte Arduino, led, résistance
// IDE: Arduino 2.x.x
// Fichier : blink.ino

#define led 12

void setup() {
  pinMode(led,OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(500); // Attente 0,5s
  digitalWrite(led, LOW);
  delay(500); // Attente 0,5s
}
```

- a. Présentation

La majeure partie des broches du Raspberry Pi Pico sont des GPIO nommées « GPxx » ci-dessous.



- c. *Ecriture d'un état logique sur une broche*
 - **Ressources** sur Micropython.org.
 - [module machine](#) | [class Pin](#) – control I/O pins

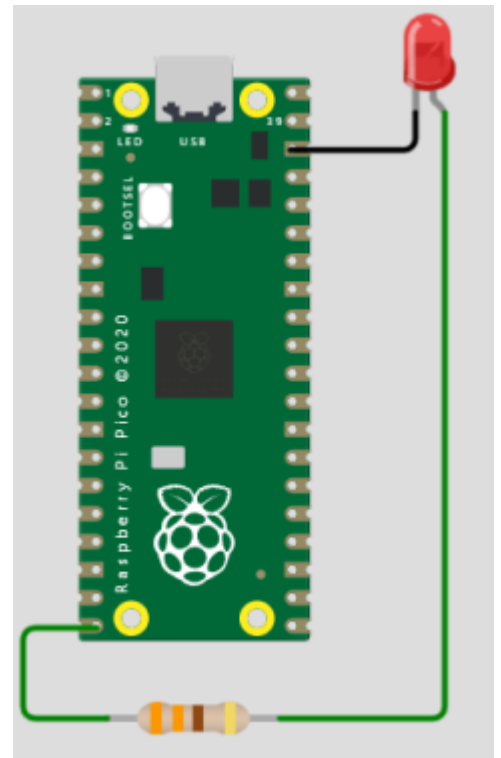
*.py

```
from machine import Pin # Bibliothèque

# Configuration de la broche associée à la led de la carte
led_carte = Pin('LED',Pin.OUT) # 'LED' <=> 25 (GPIO25)

# Exemples
# Ecriture d'un "1" sur GPIO25 -> la LED s'éclaire
led_carte.on()
# Ecriture d'un "0" sur GPIO25 -> la LED s'éteint
led_carte.off()
```

- d. *Synthèse: Blink.py*
 - **Ressources** sur Micropython.org.
 - [module machine](#) | [module time](#) | [class Pin](#) – control I/O pins
 - [Quick reference for the Rp2, Pins and GPIO.](#)



blink.py

```
# -----
# -----
# Faire clignoter une led externe à la carte Raspberry Pi Pico
# Matériels : Raspberry Pi Pico, led, Shield Grove
# IDE : Thonny
```

```
# Fichier : blink.py
# -----
# Bibliothèques
from machine import Pin
import time

# Configuration de la broche associée à la led de la carte
led = Pin(15,Pin.OUT) # (GPIO15)

while (True):
    led.on()
    time.sleep(0.5) # Attente 0,5s
    led.off()
    time.sleep(0.5) # Attente 0,5s
```

- a. Présentation

La majeure partie des broches du Raspberry Pi Pico sont des GPIO.

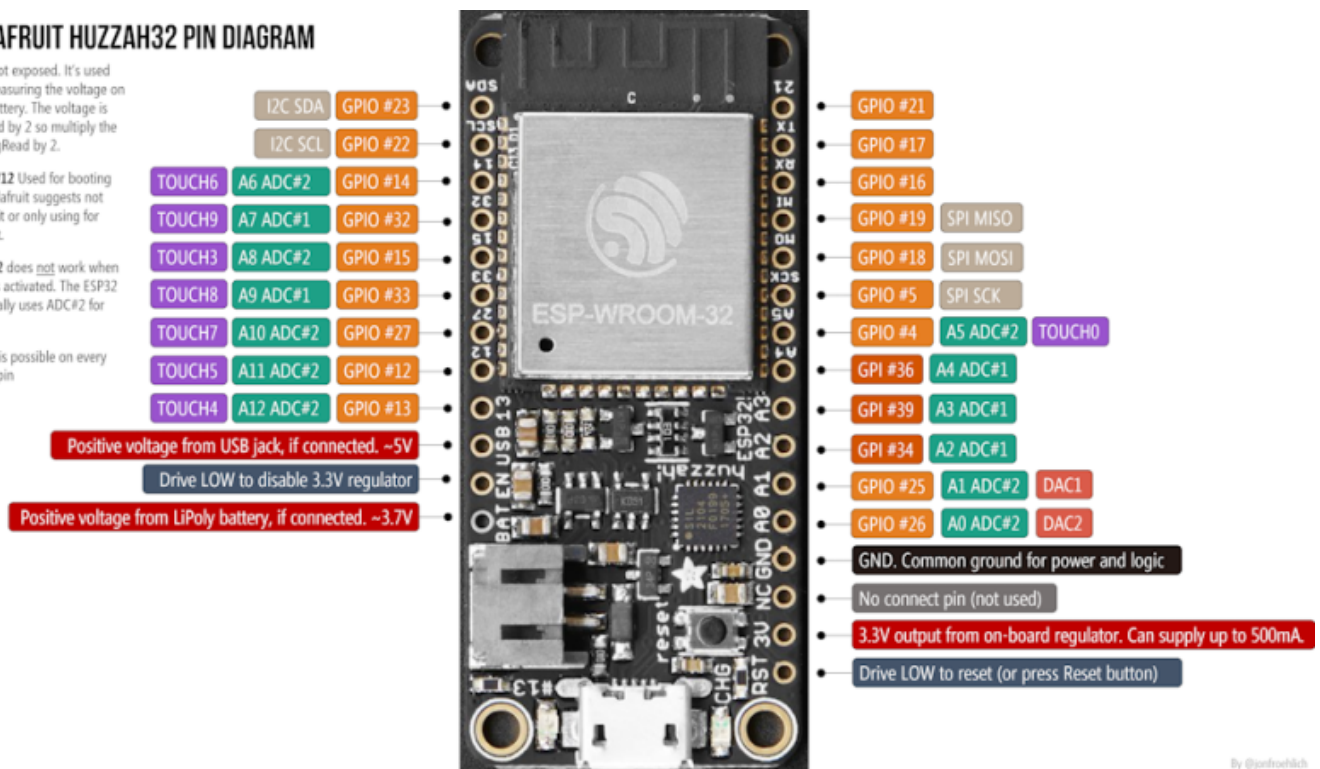
ADAFRUIT HUZZAH32 PIN DIAGRAM

A13 not exposed. It's used for measuring the voltage on the battery. The voltage is divided by 2 so multiply the analogRead by 2.

GPIO#12 Used for booting up. Adafruit suggests not using it or only using for output.

ADC#2 does not work when Wifi is activated. The ESP32 internally uses ADC#2 for Wifi

PWM is possible on every GPIO pin



- b. Configuration d'une broche en sortie
 - **Ressources** sur MicroPython.org
 - [module machine](#) | [class Pin](#) – control I/O pins

*.py

```
# Exemple
from machine import Pin # Bibliothèque nécessaire aux E/S
```

```
# Configuration en sortie de la broche associée à la led de la carte
led_carte = Pin(13, Pin.OUT) # Associée à Pin.OUT, Pin force la broche
13 de la carte ESP32
                                # à se comporter comme une sortie
```

- c. Ecriture d'un état logique sur une broche
 - **Ressources** sur Micropython.org.
 - [module machine](#) | [class Pin - control I/O pins](#)

*.py

```
from machine import Pin # Bibliothèque nécessaire aux E/S

# Configuration en sortie de la broche associée à la led de la carte
led_carte = Pin(13, Pin.OUT) # Associée à Pin.OUT, Pin force la broche
13 de la carte ESP32
                                # à se comporter comme une sortie

# Exemples
# Ecriture d'un "1" sur GPIO13 -> la LED s'éclaire
led_carte.on()
# Ecriture d'un "0" sur GPIO13 -> la LED s'éteint
led_carte.off()
```

- d. Synthèse: Blink.py
 - **Ressources** sur Micropython.org.
 - [module machine](#) | [module time](#) | [class Pin - control I/O pins](#)
 - [Quick reference for the ESP32, Pins and GPIO.](#)

blink.py

```
# -----
# -----
# Faire clignoter la led de la carte ESP32 Feather Huzzah
# IDE : Thonny
# Fichier : blink.py
# -----
# -----
# Bibliothèques
from machine import Pin
import time

# Configuration de la broche associée à la led de la carte
led_carte = Pin(13, Pin.OUT)

while (True):
    led_carte.on()
    time.sleep(1) # Attente 1s
```

```
led_carte.off()  
time.sleep(1) # Attente 1s
```

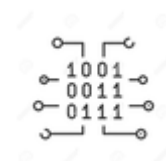
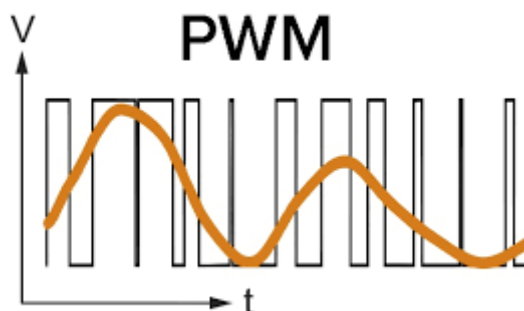
2.3 Sortie numérique (PWM)

2.3.1 Généralités

La **Modulation de Largeur d'Impulsions (MLI ; en anglais PWM pour Pulse Width Modulation)**, est une technique couramment utilisée pour synthétiser des signaux **pseudo** analogiques à l'aide de circuits numériques ("1" ou "0").

Le PWM est utilisé par exemple pour contrôler la luminosité d'une LED, changer la couleur d'une LED RGB ou encore piloter la vitesse d'un moteur.

Le principe est de créer un signal prenant les valeurs "1" ou "0", à fréquence fixe mais dont le rapport cyclique est contrôlé numériquement, la **valeur moyenne** de ce signal étant une grandeur analogique, égale au produit du rapport cyclique par l'amplitude maximale du signal. [Wikipédia](#)



2.3.2 Programmation

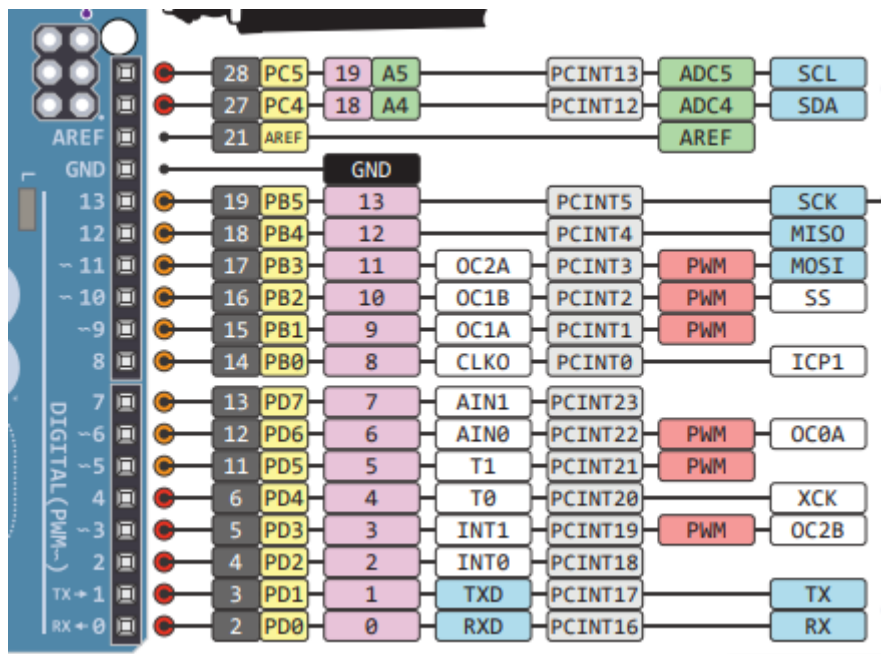
Les programmes suivant sont codés :

- En langage **C** sur Arduino Uno ou compatible
- En langage **MicroPython** sur Raspberry Pi Pico et ESP32
- [Arduino Uno R3](#)
- [RPi Pico](#)
- [ESP32](#)
- **Ressource** : [analogWrite\(\)](#) sur le site [arduino.cc](#)

a. Présentation



Cartes	Broche PWM	Fréquence	Repérage de la broche PWM sur la carte
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (broches 5 et 6: 980 Hz)	~



SCHEMA DE LA CARTE

Téléchargeable [ici](#).

b. Configuration d'une broche en sortie et génération d'un signal PWM

*.cpp

```
// Exemple
// -----
// Diminution de la luminosité d'une led avec un signal PWM

#define led 11

// Les instructions contenues dans la fonction setup() s'exécutent une fois
void setup() {
    pinMode(led, OUTPUT);    // Associé à OUTPUT pinMode force la broche
                             // 11 de la carte Arduino Uno à se comporter comme une sortie
}

// Les instructions contenues dans la fonction loop() s'exécutent tant
// que le microcontrôleur est sous tension
```

```
void loop() {  
    analogWrite(led, 127); // La broche 11 délivre un signal PWM de  
                           // fréquence F=490Hz  
                           // et de rapport cyclique = 1/2 (voir la  
                           // ressource pour le calcul du rapport cyclique)  
}
```

A faire

A faire

Ressource provisoire [ESP32/ESP8266 PWM with MicroPython - Dim LED](#)

2.4 Entrée numérique

2.4.1 Généralités

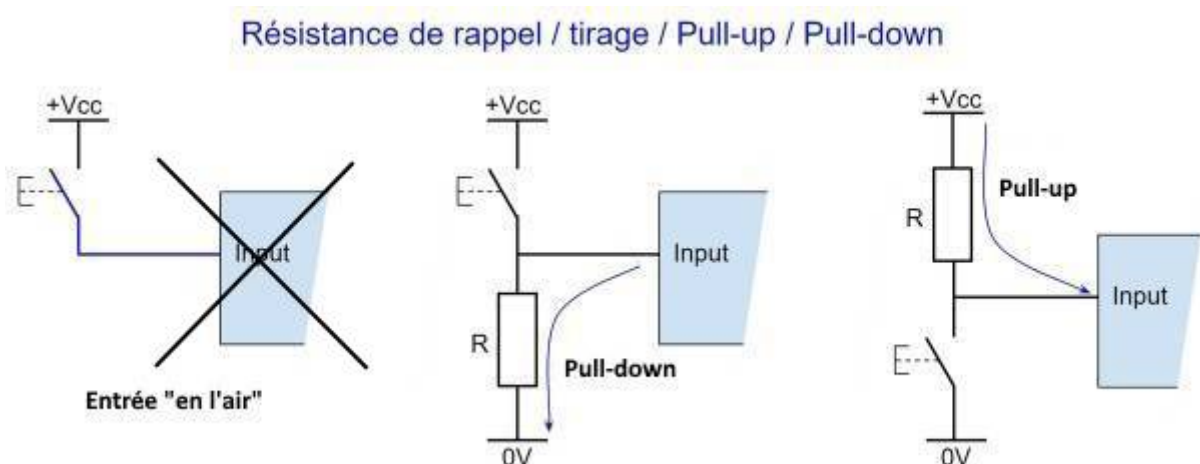
Lorsqu'une broche est configurée en **entrée numérique**, le microcontrôleur est capable de traiter des signaux prenant la valeur logique "0" ou "1".

RESISTANCE DE RAPPEL

Les microcontrôleurs disposent de **résistances de rappel** internes pouvant être connectées par le logiciel.

Une entrée numérique utilisée dans un programme **ne doit pas être laissée "en l'air"** (non connectée) car elle prendra alors un état logique aléatoirement et le comportement du programme deviendra imprévisible.

Exemple : détermination de l'état ouvert ou fermé d'un bouton-poussoir.



PRECAUTIONS D'UTILISATION

Les entrées numériques sont **fragiles**. Elles ne supportent ni les **décharges**

électrostatiques ni les **surtensions**. Il ne faut ni les toucher ni leur appliquer une tension supérieure à 5V ou inférieure à 0V.



2.4.2 Programmation

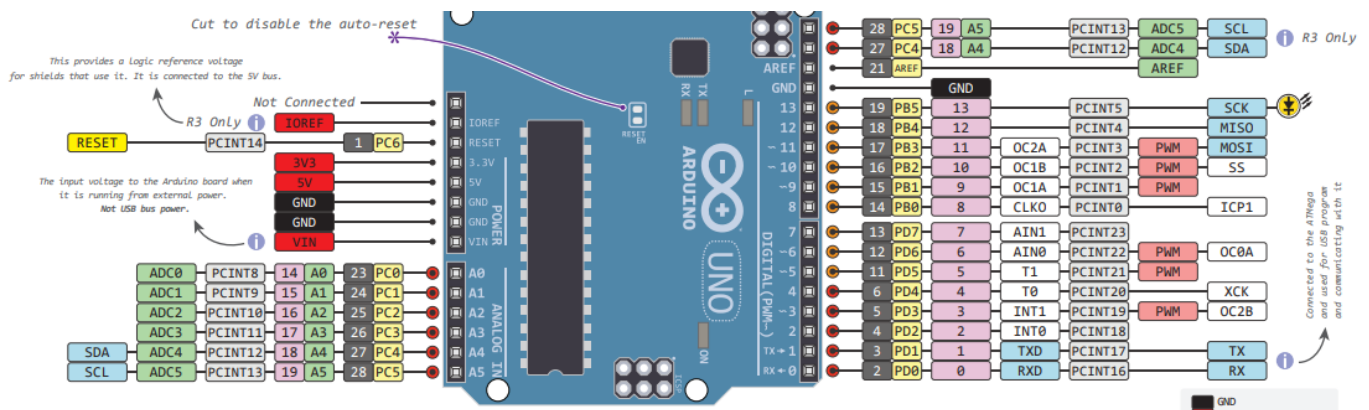
Les programmes suivant sont codés :

- En langage **C** sur Arduino Uno ou compatible
- En langage **MicroPython** sur Raspberry Pi Pico et ESP32

- Arduino Uno
- RPi Pico
- ESP32

a. Présentation

Toutes les broches d'E/S de la carte Arduino Uno peuvent servir d'entrées, sorties numériques. Cela inclut les broches numérotées entre **0** et **13**, mais aussi les broches **A0** à **A5**. Le symbole ~ identifie les broches capables de délivrer un signal numérique **PWM** (**P**ulse **W**ith **M**odulation).



SCHEMA DE LA CARTE

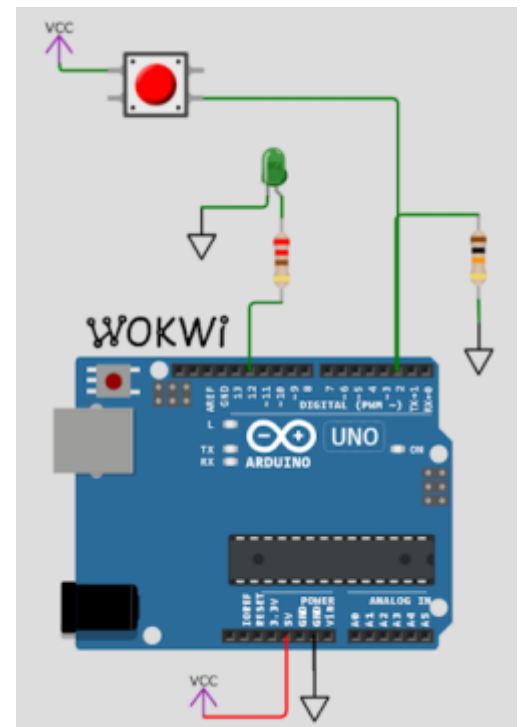
Téléchargeable [ici](#).

b. Configuration d'une broche en entrée

- **Ressource** : [pinMode\(\)](#) sur le site [arduino.cc](#)

CONFIGURATION

Configurer une broche signifie qu'on précise qu'elle doit se comporter soit comme une **entrée d'information**, soit comme une sortie d'information.

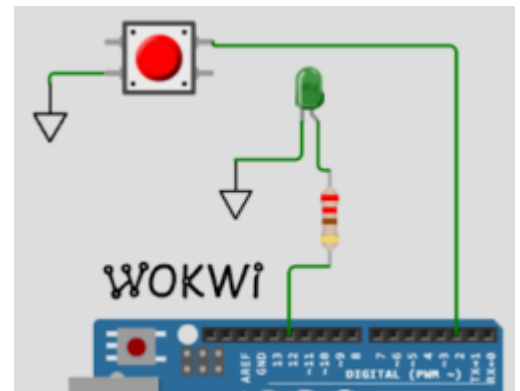


*.cpp

```
// Exemple 1 : connexion d'une broche à un bouton-poussoir via une
// résistance pull-down
// externe à la carte Arduino Uno (celle-ci n'en dispose pas en
// interne!).
// -----
// Les instructions contenues dans la fonction setup() s'exécutent une
// fois
void setup() {
    pinMode(2,INPUT);    // Associé à INPUT pinMode force la broche 2 de
// la carte Arduino Uno
// à se comporter comme une entrée
// Optionnel si la broche n'a pas été
préalablement
// configurée en sortie, car c'est le mode par
défaut
}
// -----
// Exemple 2
// -----
// L'utilisation de la directive "define" améliore la lisibilité des
// programmes
```

```
#define BP 2

void setup() {
  pinMode(BP, INPUT); // On précise que la broche 2 de la carte Arduino
                        // une entrée et est connectée à un bouton-
                        // poussoir
                        // optionnel si la broche n'a pas été
préalablement      // configurée en sortie, car c'est le mode par
défaut
}
```



*.cpp

```
// Exemple 3 : connexion d'une broche à un bouton-poussoir via la
// résistance de pull-up interne
// de la carte arduino uno. On économise ainsi 1 composant
// -----
// L'utilisation de la directive "define" améliore la lisibilité des
// programmes
#define BP 2

void setup() {
  pinMode(BP, INPUT_PULLUP); // On précise que la broche 2 de la carte
                             // résistance de pull-up interne, se
  // comporte comme une entrée et est
                             // connectée à un bouton-poussoir
}
```

c. Lecture de l'état logique sur une broche

- **Ressource** : [digitalRead\(\)](#) sur le site [arduino.cc](#)

LECTURE

Un état logique **haut(HIGH)** ou **bas(LOW)** peut être **lu** sur une broche préalablement configurée en entrée.

*.cpp

```
// Exemple 1
// -----
// Les instructions contenues dans la fonction setup() s'exécutent une
// fois
void setup() {
    pinMode(2,INPUT);    // Associé à INPUT pinMode force la broche 2 de
// la carte Arduino Uno à se comporter comme une entrée
// ou pinMode(2, INPUT_PULLUP); selon le
montage utilisé
}

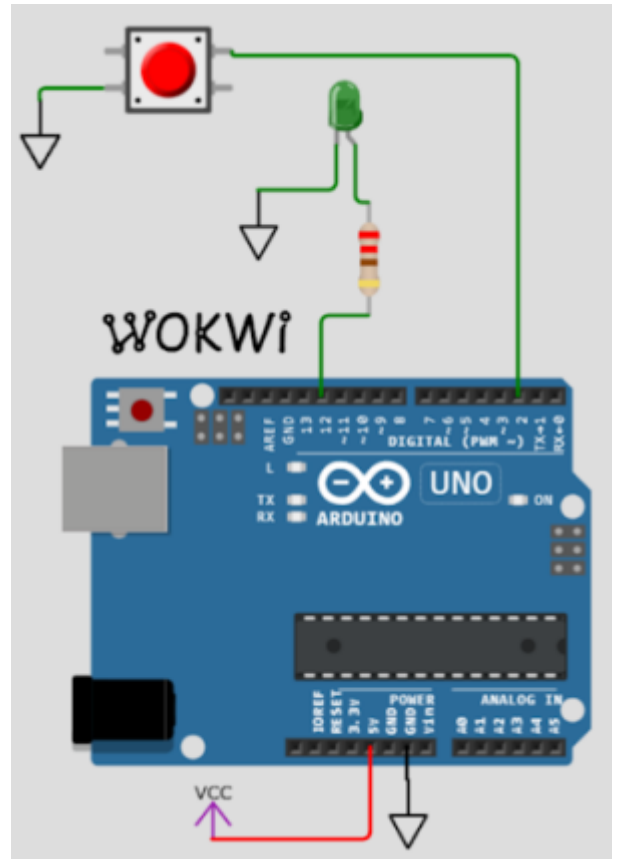
// Les instructions contenues dans la fonction loop() s'exécutent tant
// que le microcontrôleur est sous tension
void loop() {
    bool etat_bp = digitalRead(2); // L'état logique sur l'entrée 2 est
transféré dans une variable de type booléen
}

//Exemple 2
// -----
// L'utilisation de la directive define améliore la lisibilité des
programmes
#define bp 2

void setup() {
    pinMode(bp,INPUT); // ou pinMode(bp, INPUT_PULLUP); selon le montage
utilisé
}

void loop() {
    bool etat_bp = digitalRead(bp); // L'état logique sur l'entrée 2
est transféré dans une variable de type booléen, cette entrée
// est reliée à un bouton-poussoir
}
```

d. Synthèse: ledbp



*.cpp

```
// Titre : LED_BP
// Description : la led s'éclaire lorsqu'on presse le bouton-poussoir
// (BP), elle s'éteint lorsqu'on le relâche.

// Entrées, sorties
#define led 12
#define bp 2

bool etat_bp;

void setup() { // Configuration des E/S
  pinMode(led, OUTPUT);
  pinMode(bp, INPUT_PULLUP);
}

void loop() {
  // Lecture de l'entrée
  bool etat_bp = !(digitalRead(bp)); // le bouton-poussoir étant actif
  // à l'état bas "0" alors que la led l'est à l'état haut "1"
  // nécessite d'inverser l'état

  logique de la variable etat_bp
  digitalWrite(led, etat_bp); // Ecriture de l'entrée sur la
  // sortie
}
```

- **Ressources** sur Micropython.org.
 - [module machine](#) | [module time](#) | [class Pin - control I/O pins](#)
 - [Quick reference for the RP2, Pins and GPIO](#)

b. Configuration d'une broche en en entrée

c. Lecture de l'état logique sur une broche

d. Synthèse: ?.cpp Exemple de code pour un **Raspberry Pi Pico**

*.py

```
# Configuration (en entrée) des broches connectées à deux boutons-
poussoirs
# Bibliothèques à installer
from machine import Pin

# Configuration (en entrée) des broches connectées à deux boutons-
poussoirs
button_min = Pin(20, Pin.IN)
button_hr = Pin(21, Pin.IN)
...
```

- **Ressources** sur Micropython.org.
 - [module machine](#) | [module time](#) | [class Pin - control I/O pins](#)
 - [Quick reference for the ESP32, Pins and GPIO](#)

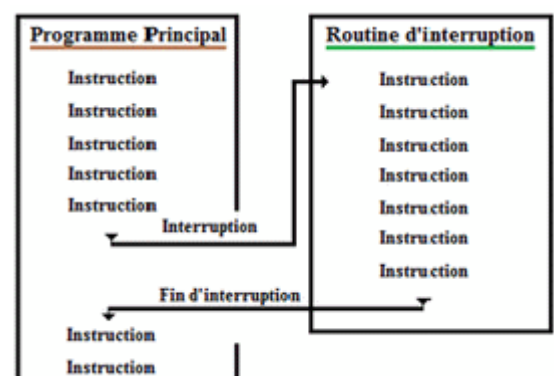
b. Configuration d'une broche en en entrée

c. Lecture de l'état logique sur une broche

d. Synthèse: ?.cpp Exemple de code pour un **ESP32 Feather Huzzah**

*.py

```
# Configuration (en entrée) des broches connectées à deux boutons-
poussoirs
button_min = Pin(25, Pin.IN)
button_hr = Pin(26, Pin.IN)
...
```



2.5 Entrée d'interruption

2.5.1 Généralités

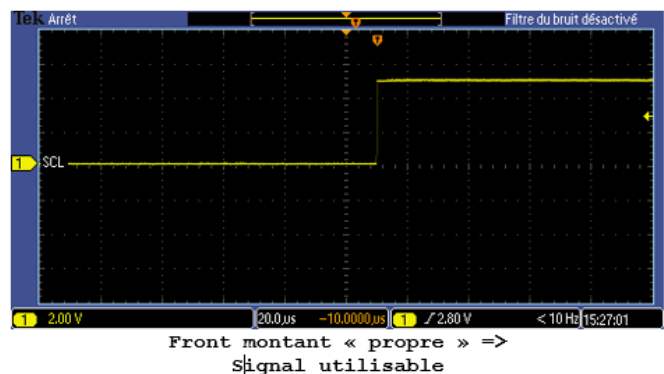
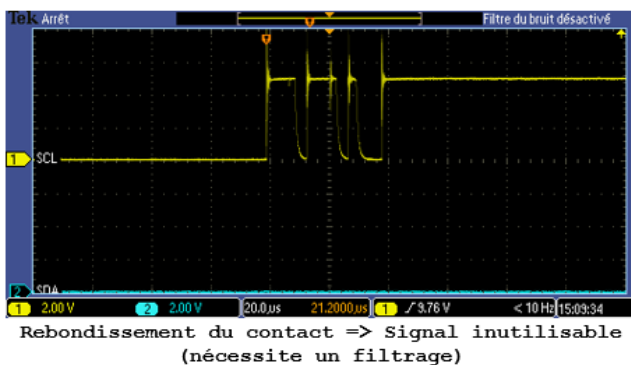
Une **interruption** est un **arrêt temporaire** de l'exécution normale d'un programme par le processeur afin d'exécuter un autre programme (appelé **service d'interruption**).

L'interruption est provoquée par une cause externe (action sur un bouton-poussoir, mesure réalisée par un capteur, horloge temps réel, etc.).

On utilise les interruptions afin de permettre des **communications non bloquantes** avec des périphériques externes.

Une interruption tient compte de l'état logique présent sur une broche. Couramment, on la déclenchera sur **le front montant, le front descendant, ou chacun des fronts** d'un signal logique.

Une interruption sera reconnue si le signal présente des fronts "propres". Il faudra donc s'assurer de la qualité du signal. Les figures ci-dessous représentent un signal transmis à la fermeture du contact d'un anémomètre. Le signal de gauche n'est pas utilisable à cause du rebondissement du contact. En effet, il contient quatre fronts montants au lieu d'un seul comme dans le cas du signal de droite.



2.5.2 Programmation

Les programmes suivant sont codés :

- En langage **C** sur Arduino Uno ou compatible
- En langage **MicroPython** sur Raspberry Pi Pico et ESP32
- [Arduino Uno](#)
- [RPi Pico](#)

- [ESP32](#)

- **Ressource :**

- Présentation
- Configuration d'une broche en entrée d'interruption
- Evènement et gestionnaire d'interruption
- Synthèse: ?

- **Ressource :**

- Présentation
- Configuration d'une broche en entrée d'interruption
- Evènement et gestionnaire d'interruption
- Synthèse: ?

- **Ressource :** [Quick reference for ESP32](#), GPIO Pins, External interrupts sur Micropython.org.



- Présentation

- **Ressource provisoire** [MicroPython: Interrupts with ESP32 and ESP8266](#)

- Configuration d'une broche en entrée d'interruption
- Evènement et gestionnaire d'interruption
- Synthèse: HORLOGE NUMERIQUE

- **Matériel :** [ESP32 Feather Huzzah](#), [Digilent Pmod BTN: 4 User Pushbuttons](#) [Schéma]



- **Description :** l'extrait du programme HORLOGE ci-dessous illustre l'utilisation d'interruptions assurant le réglage de l'heure à l'aide de trois boutons-poussoir. Une vidéo de présentation est accessible [ici](#).

*.py

```
# -----
# Code partiel du programme HORLOGE
# -----

# Réglage de l'heure à la mise sous tension
time_offset=12*3600+0*60+0 # hh+mm+ss

# Routines de service d'interruption (ISR)
def handle_interrupt_min(pin):
```

```
global time_offset
time_offset+=60
time.sleep(.2)

def handle_interrupt_hr(pin):
    global time_offset
    time_offset+=3600
    time.sleep(.2)

# Réglage des minutes
# Ajout de 60s à l'heure initiale
button_min = Pin(25, Pin.IN)
# Gestionnaire d'interruption
button_min.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt_min)

# Réglage des heures
# Ajout de 3600s à l'heure initiale
button_hr = Pin(26, Pin.IN)
# Gestionnaire d'interruption
button_hr.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt_hr)
{{ :python:micropython:materiel:thonny.png?nolink&70| }}
```

TELECHARGER

Le projet **MICROPYTHON_ESP32_HORLOGE** pour l'IDE **Thonny**.

3. Entrées analogiques

- Ressource : [site Arduino](#)

3.1 Généralités



3.2 Programmation

- [Arduino Uno](#)
- [RPi Pico](#)
- [ESP32](#)

- a. Présentation
- b. Configuration d'une broche en entrée analogique
- c. Lecture d'une entrée analogique
- d. Synthèse: ?
- a. Présentation
- b. Configuration d'une broche en entrée analogique
- c. Lecture d'une entrée analogique
- d. Synthèse: ?



• Ressource

- [ADC \(analog to digital conversion\)](#) | [class Pin – control I/O pins](#) sur Micropython.org.

Exemple de code pour un **Raspberry Pi Pico**

*.py

```
# -----
# -----
# Lecture et affichage dans la console de la tension issue d'un
# potentiomètre
# Date : 22/5/2023
# Matériels : Raspberry Pi Pico, Shield Grove, pot. 10k
# ADC accessibles sur le shield Grove pour RP2 :
# Connecteur: ADC      : GPIO
#      A0 : ADC0      : 26
#      A1 : ADC0,ADC1: 26,27
#      A2 : ADC1,ADC2: 27,28
# IDE : Thonny
# -----
# -----
from machine import ADC, Pin
import time

# Le potentiomètre 10k0hm est connecté à l'entrée analogique A0 du
# shield.
# Attention : La tension doit être comprise entre 0 - 3,3V (3,6V max !)
# sur une entrée analogique.
# Configuration
pot = ADC(Pin(26))

while (True):
    val=pot.read_u16() # lecture de l'ADC
    U = val*3.3/65535 # Calcul de la tension
    print("%.2f" % U) # Affichage dans la console (formaté à 2
# décimales)
    time.sleep(1)
```

- a. Présentation

Ressource provisoire : [ESP32/ESP8266 Analog Readings with MicroPython](#)

b. Configuration d'une broche en entrée analogique

c. Lecture d'une entrée analogique

d. Synthèse: ?



- **Ressource**

- [ADC \(analog to digital conversion\)](#) sur Micropython.org.

Exemple de code pour un **ESP32 Feather Huzzah**

*.py

```
# ADC accessibles en Python sur la carte ESP32 Feather Huzzah :
# ADC:GPIO
# A2 : 34
# A3 : 39
# A4 : 36
# A7 : 32
# A9 : 33

from machine import ADC, Pin

# Le potentiomètre 10k0hm est connecté à l'entrée analogique A2 de
l'ESP32.
# Configuration
adc = ADC(Pin(34))
# Sur une entrée analogique, la tension doit
# être comprise entre 0 - 3,3V (3,6V max !)
adc.atten(ADC.ATTN_11DB) # voir doc
# Mesure
value = adc.read()

print(value) # affichage dans la console
```

pour aller plus loin

SISEL SA - [Entrées-sorties d'un microcontrôleur](#)

1)

Un microcontrôleur est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires, unités périphériques et interfaces d'entrées-sorties.

2)

Une broche de microcontrôleur est un élément physique situé sur le boîtier du microcontrôleur, qui permet de connecter ce dernier à d'autres composants électroniques, tels que des capteurs, des actionneurs ou d'autres circuits.

³⁾

Tension de sortie à l'état bas

⁴⁾

Tension d'entrée à l'état bas

⁵⁾

Tension de sortie à l'état haut

⁶⁾

Tension d'entrée à l'état haut

From:

<https://webge.fr/dokuwiki/> - **WEBGE Wikis**

Permanent link:

<https://webge.fr/dokuwiki/doku.php?id=microc:gpio&rev=1739963599>

Last update: **2025/02/19 12:13**

