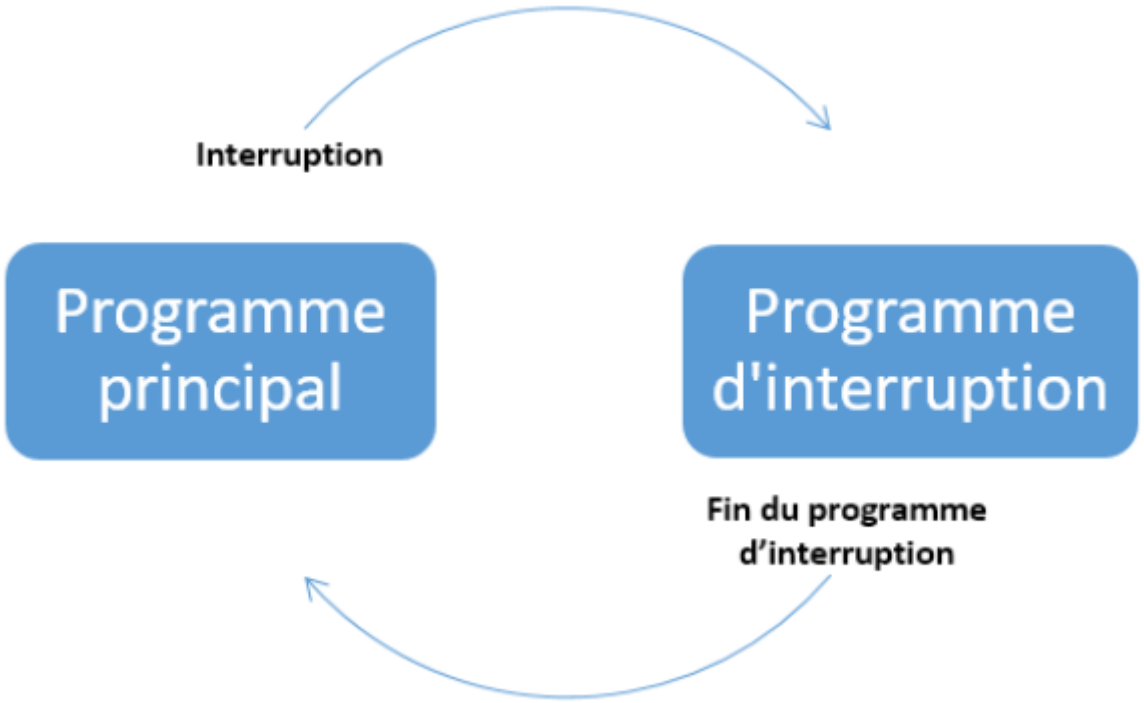
Les interruptions sur ESP32

DOC Expressif : Allocation d'interruption EN

Les interruptions sur ESP32

DOC Expressif : Allocation d'interruption FR

Une interruption est un processus qui est déclenché de manière asynchrone par un évènement extérieur, qui interrompt momentanément l'exécution du code en cours, pour exécuter du code plus critique. À quoi ça sert ?



Imaginez que vous vouliez allumer une LED lorsque vous appuyez sur un bouton qui est relié à un pin GPIO de l'ESP32. Le plus simple est de regarder en permanence dans la fonction loop() si vous avez appuyé sur le bouton :

interruptions001.ino

```
const int buttonPin = 33;
const int ledPin =  2;

// Etat du bouton poussoir
int buttonState = 0;

void setup() {
    Serial.begin(115200);

    //Configuration du pin en entrée pullup
```

```
    pinMode(buttonPin, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);
}

void loop() {
    buttonState = digitalRead(buttonPin);

    if (buttonState == LOW) {
        digitalWrite(ledPin, HIGH);
    } else if(buttonState == HIGH){
        digitalWrite(ledPin, LOW);
    }
}
```

Le problème est que le processeur du microcontrôleur est totalement occupé par cette tâche. Alors on peut dire au microcontrôleur de faire d'autres tâches dans la loop() , mais dans ce cas le microcontrôleur ne regardera l'état du bouton qu'une seule fois à chaque itération de la fonction loop() . Il se peut qu'on manque un évènement. On ne peut pas traiter en temps réel des évènements extérieurs. Les interruptions permettent de détecter un évènement en temps réel tout en laissant le processeur du microcontrôleur faire d'autres tâches. Ainsi le fonctionnement d'une interruption est le suivant :

Détection d'un évènement → Interruption du programme principal → Exécution du code de l'interruption → Le processeur reprend là où il s'est arrêté.



Avec les interruptions, il n'y a plus besoin de regarder en permanence la valeur d'un pin : lorsqu'un changement est détecté, une fonction est exécutée.

Les modes de détection

La détection d'un événement est basée sur l'allure du signal qui arrive au pin.

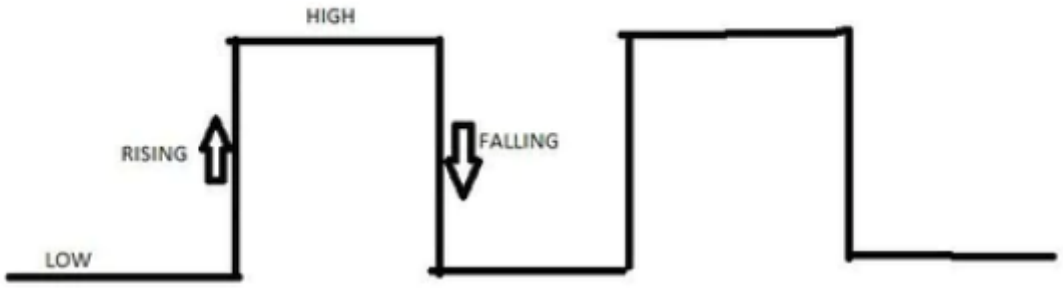


Figure: Digital Signal

Différents modes de détection

On peut choisir le mode de détection de l'interruption :

- **LOW** : Déclenche l'interruption dès que le signal est à 0V
- **HIGH** : Déclenche l'interruption dès que le signal est à 3.3V
- **RISING** : Déclenche l'interruption dès que le signal passe de LOW à HIGH (0 à 3.3V)
- **FALLING** : Déclenche l'interruption dès que le signal passe de HIGH à LOW (3.3V à 0)
- **CHANGE** : Déclenche l'interruption dès que le signal passe de LOW à HIGH ou de HIGH à LOW .



Les modes RISING et FALLING sont les plus utilisés. Noter que si vous utilisez les modes LOW et HIGH , l'interruption se déclenchera en boucle tant que le signal ne change pas d'état.

Utilisation sur l'ESP32

L'utilisation des interruptions sur l'ESP32 est similaire à celle sur l'Arduino avec la fonction attachInterrupt() . **N'importe quel pin GPIO peut être utilisé pour les interruptions.** *Voir ICI

Ainsi pour créer une interruption sur un pin , il faut :

- Attribuer un pin pour détecter l'interruption attachInterrupt()

inter001.ino

```
attachInterrupt(GPIOPin, fonction_ISR, Mode);
```

Avec** *Mode*** , le mode de détection qui peut être LOW , HIGH , RISING , FALLING ou CHANGE

Créer la fonction qui va être exécutée lorsque l'interruption est déclenchée

inter003.ino

```
void IRAM_ATTR fonction_ISR() {
  // Contenu de la fonction
}
```



Il est conseillé d'ajouter le flag IRAM_ATTR pour que le code de la fonction soit stocké dans la RAM (et non pas dans la Flash), afin que la fonction s'exécute plus rapidement.

Le code entier sera de la forme :

inter002.ino

```
void IRAM_ATTR fonction_ISR() {
  // Code de la fonction
}

void setup() {
  Serial.begin(115200);
  pinMode(23, INPUT_PULLUP);
  attachInterrupt(23, fonction_ISR, FALLING);
}

void loop() {
}
```

- Dès que la tension passera de 3.3V à 0V, la fonction fonction_ISR() sera exécutée. On peut ensuite faire d'autres tâches dans la fonction loop() .

Il faut garder en tête que la fonction d'une interruption doit s'exécuter le plus rapidement possible pour ne pas perturber le programme principal. Le code doit être le plus concis possible et il est déconseillé de dialoguer par SPI, I2C, UART depuis une interruption.



On ne peut pas utiliser la fonction delay() ni Serial.println() avec une interruption. On peut néanmoins afficher des messages dans le moniteur série en remplaçant Serial.println() par ets_printf() qui est compatible avec les interruptions.

Le code ci-dessous affiche « Boutton pressé » lorsqu'on presse sur un bouton relié au pin 33.

inter004.ino

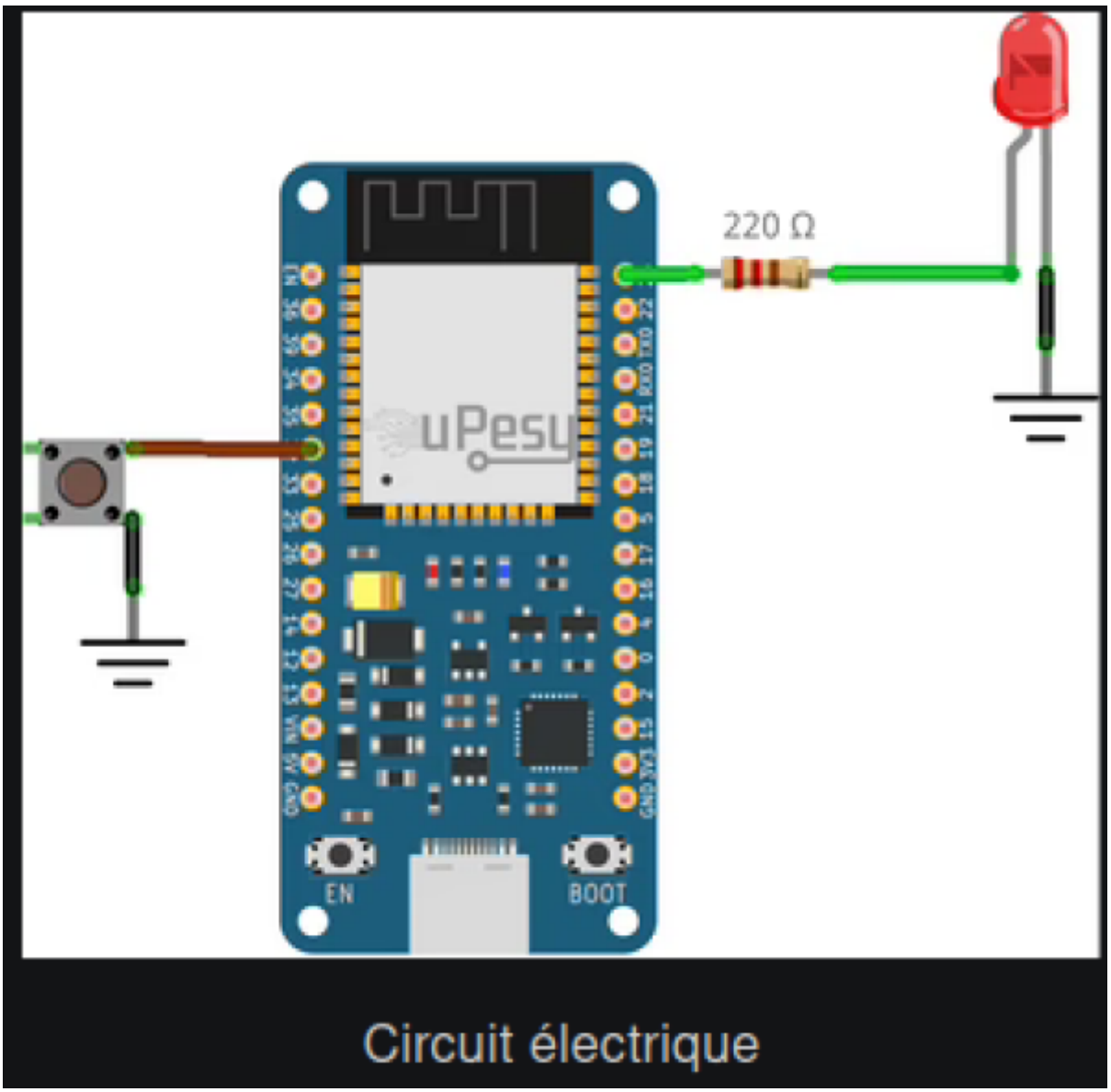
```
void IRAM_ATTR fonction_ISR() {
    ets_printf("Boutton pressé\n");
    // Code de la fonction
}
```

```
void setup() {
    Serial.begin(115200);
    pinMode(33, INPUT_PULLUP);
    attachInterrupt(33, fonction_ISR, FALLING);
}

void loop() {
}
```

Mini-Projet

Nous allons refaire le premier mini-projet qui consistait à faire clignoter une LED lorsqu'on appuie sur un bouton . On va utiliser les interruptions pour gérer l'événement et libérer le processeur pour qu'il puisse faire d'autres tâches.



Code

inter005.ino

```
//Solution

const int buttonPin = 32;
const int ledPin =  23;
int buttonState = 0;
int lastMillis = 0;

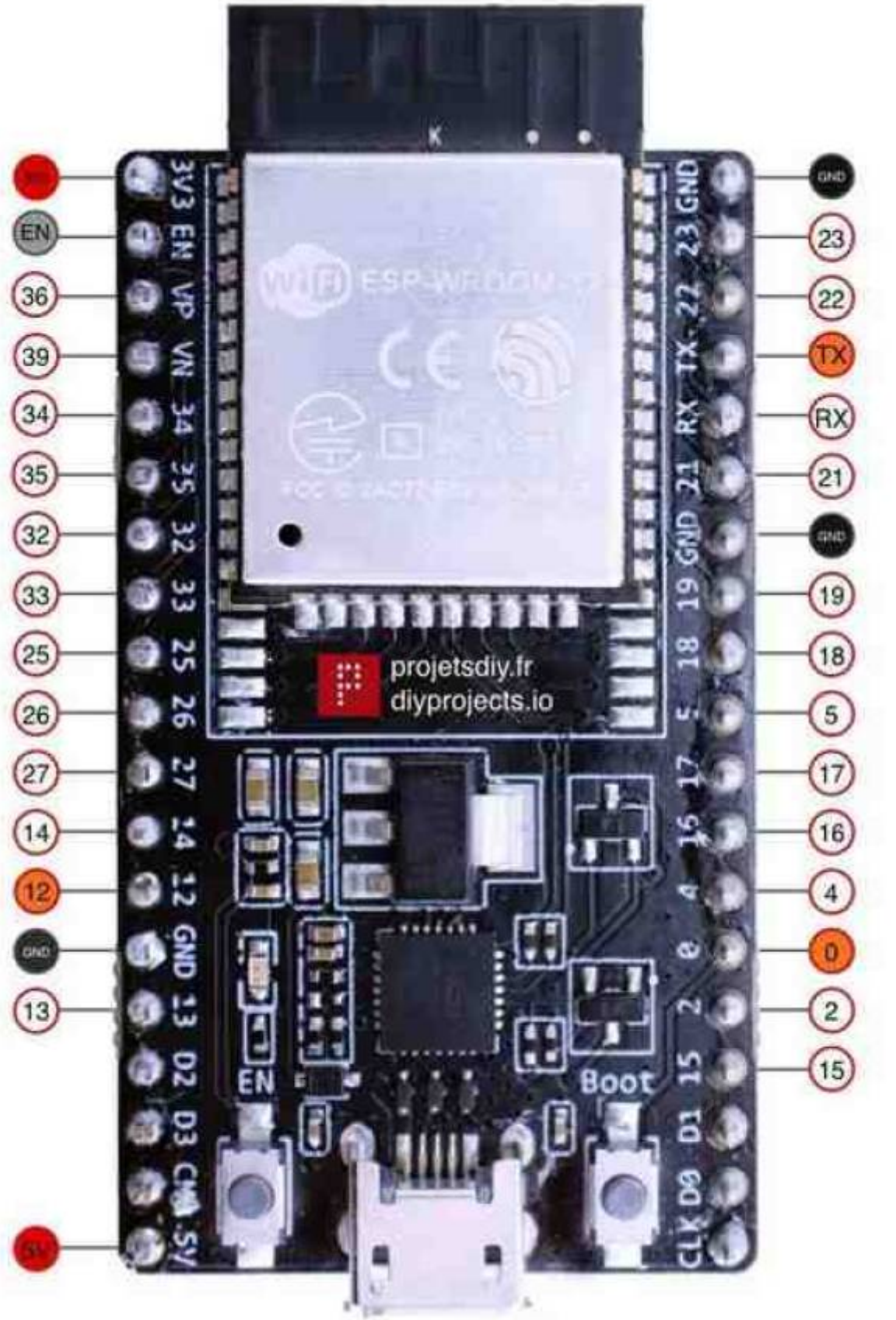
void IRAM_ATTR fonction_ISR() {
 if(millis() - lastMillis > 10){ // Software debouncing buton
         ets_printf("ISR triggered\n");
         buttonState = !buttonState;
         digitalWrite(ledPin,buttonState);
 }
 lastMillis = millis();
}

void setup() {
 Serial.begin(115200);
 pinMode(buttonPin, INPUT_PULLUP);
 pinMode(ledPin,OUTPUT);
 attachInterrupt(buttonPin, fonction_ISR, CHANGE);
 digitalWrite(ledPin, buttonState);
}

void loop() {
 // Code ...
}
```

Broches du GPIO de l’ESP32 compatibles avec les interruptions

Les interruptions fonctionnent uniquement avec les entrées numériques. Les entrées numériques pouvant déclencher une interruption sont entourée d’un cercle sur le schéma ci-dessous.



ESP32-WROOM-32 - ESP32 DevKitC V4 pinout
Entrées numériques compatibles avec les Interruptions

Attention toutefois car certaines broches passent à l'état haut (HIGH) ou émettent des signaux PWM au démarrage ou lors de la réinitialisation. D'autres broches sont utilisées par le système pour

accéder à la mémoire flash ou téléverser le programme.

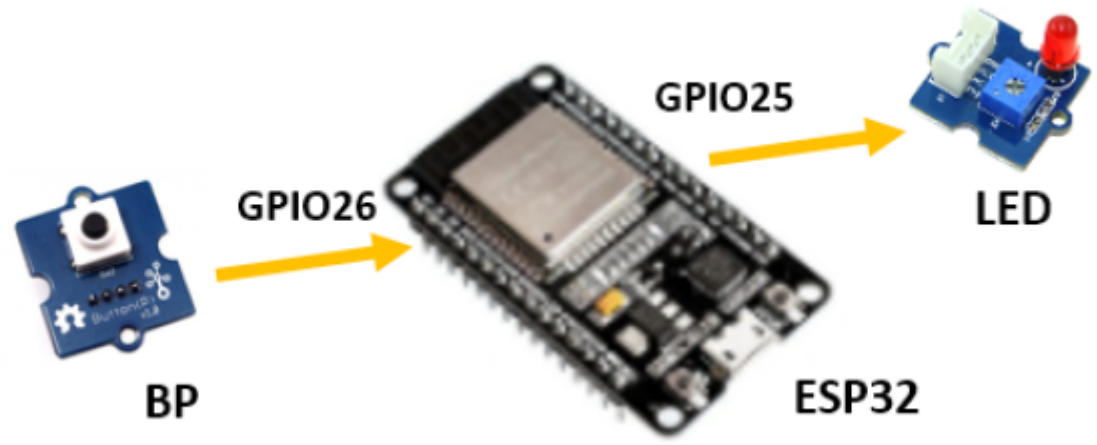
N’utilisez pas les broches colorées en orange ou en rouge. Votre programme pourrait avoir un comportement inattendu en utilisant celles-ci.

Broche du GPIO	Entrée numérique (Input)	Remarque
0	PULL UP	Envoi un signal PWM au démarrage.
1	TX	Sortie de débogage au démarrage
2	✓	Connecté à la LED embarquée
3	✓	Prend l'état HIGH au démarrage
4	✓	
5	✓	Envoi un signal PWM au démarrage
6	✖	Utilisé pour la mémoire flash SPI
7	✖	Utilisé pour la mémoire flash SPI
8	✖	Utilisé pour la mémoire flash SPI
9	✖	Utilisé pour la mémoire flash SPI
10	✖	Utilisé pour la mémoire flash SPI
11	✖	Utilisé pour la mémoire flash SPI
12	✓	Echec de démarrage si en mode PULLUP
13	✓	
14	✓	Envoi un signal PWM au démarrage
15	✓	Envoi un signal PWM au démarrage
16	✓	
17	✓	
18	✓	
19	✓	
21	✓	
22	✓	
23	✓	
25	✓	
26	✓	
27	✓	
32	✓	
33	✓	
34	✓	
35	✓	
36	✓	
39	✓	

Micropython ESP32 Interruptions

Exemple 1 de programme d'interruption avec l'ESP32

A chaque action sur le bouton poussoir (front montant), la LED change d'état même si le programme principal est occupé à une autre tâche.



ESP32 Micropython programme avec interruption sur front montant de la broche GPIO26

interrup200.py

```
from machine import Pin
from time import *

led = Pin(25, Pin.OUT)
bp = Pin(26, Pin.IN)

# Le programme d'interruption
def fct_interruption(pin):
    print("Appui sur BP detecté")
    if led.value()==1:
        led.value(0)
    else:
        led.value(1)

# Spécifie la fonction à appeler lorsqu'une interruption externe
survient
bp.irq(trigger = Pin.IRQ_RISING, handler = fct_interruption)

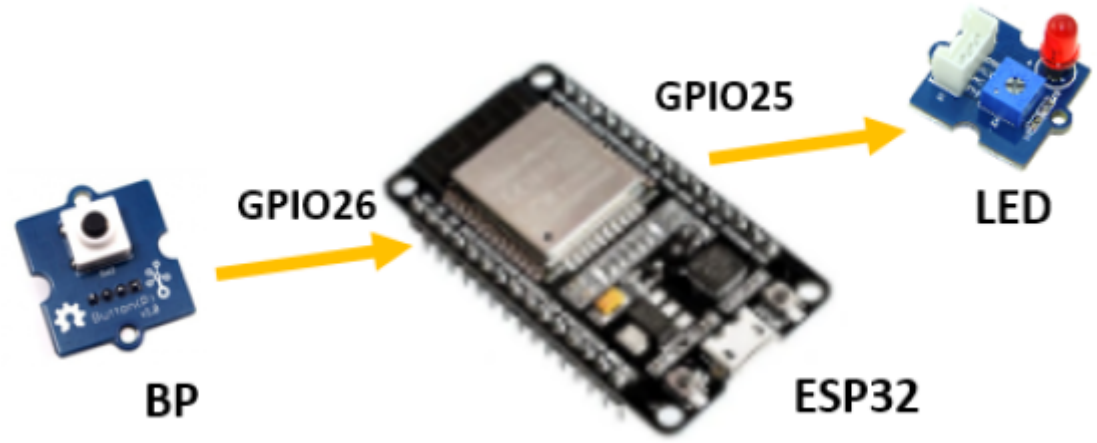
# Le programme principal
while True:
    sleep(5)
```

“bp.irq(trigger=Pin.IRQ_RISING, handler = fct_interruption)”

Spécifie la fonction à appeler lorsqu'une interruption externe survient. La fonction fct_interruption serra appelée chaque fois qu'un front montant est détecté sur le bouton poussoir bp (GPIO26)

Exemple 2 de programme d'interruption avec l'ESP32

Fonctionnement identique à l'exemple 1 mais avec un temps d’exécution le plus court possible dans la fonction fct_interruption



ESP32 Micropython programme avec interruption sur front montant de la broche GPIO26

interrup021.py

```
from machine import Pin

impulsion = False
etat_led = False

led = Pin(25, Pin.OUT)
bp = Pin(26, Pin.IN)

# Le programme d'interruption
def fct_interruption(pin):
    global impulsion
    impulsion = True

# Spécifie la fonction à appeler lorsqu'une interruption externe
survient
bp.irq(trigger = Pin.IRQ_RISING, handler = fct_interruption)
```

```
# Le programme principal
while True:
    if impulsion:
        print("Appui sur BP detecte")
        if led.value()==1:
            led.value(0)
        else:
            led.value(1)
        impulsion = False
```

impulsion = True

La variable impulsion de type booléen passe à True quand on lance le programme d'interruption (fonction fct_interruption) et repasse à False quand il est terminé.

Cette variable est utilisée dans le programme principal pour allumer ou éteindre la Led, afficher du texte sur la console.

Cela permet d'avoir un temps d'exécution le plus court possible dans la fonction fct_interruption

From:
https://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
https://chanterie37.fr/fablab37110/doku.php?id=start:esp32:interruptions

Last update: **2023/01/17 16:27**