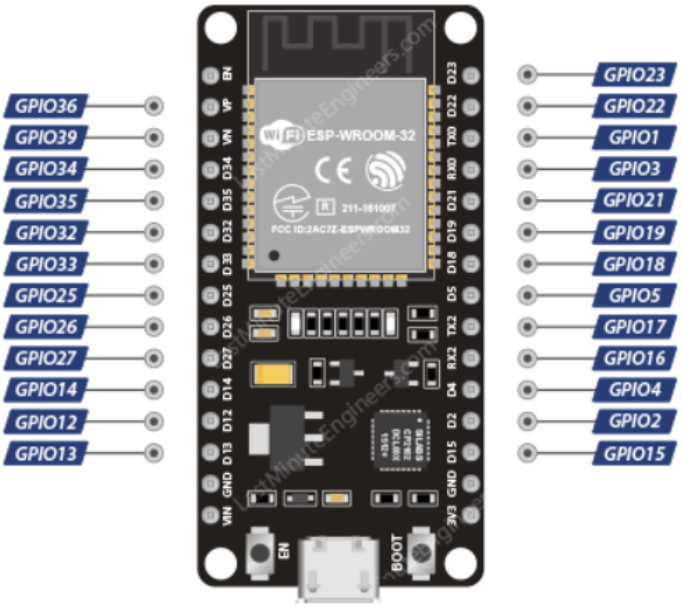
Interrupciones Externas con ESP32

ESP32 proporciona hasta 32 ranuras de interrupción para cada núcleo. Cada interrupción tiene un cierto nivel de prioridad y se puede clasificar en dos tipos.

Interrupciones de hardware: ocurren en respuesta a un evento externo. Por ejemplo, una interrupción GPIO (cuando se presiona una tecla) o una interrupción táctil (cuando se detecta un toque).

Interrupción de software: ocurren en respuesta a una instrucción de software. Por ejemplo, una interrupción de temporizador simple o una interrupción de temporizador de vigilancia (cuando el temporizador se agota).

Interrupción ESP32 GPIO
En ESP32 podemos definir una función de rutina de servicio de interrupción que se

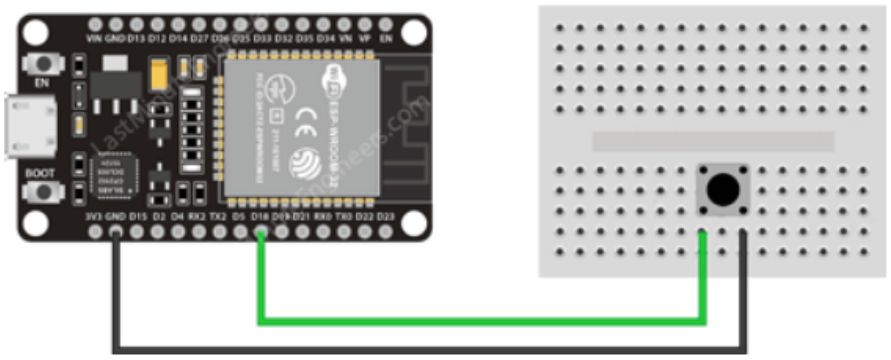


llamará cuando el pin GPIO cambie su nivel lógico.
Todos los pines GPIO en una placa ESP32 se pueden configurar para actuar como entradas de solicitud de interrupción.
En la siguiente práctica activaremos el servicio de interrupciones en una esp32 con un push button cambiando el estado de este.

Materiales necesarios:

ESP8266
Cable USB
Computadora con el IDE de Arduino instalado
Push Button
Jumpers

Esquemático de la práctica



En el IDE de Arduino, usamos una función llamada attachInterrupt() para configurar una interrupción pin por pin. La sintaxis se ve a continuación.

attachInterrupt(button1.PIN, isr, Modo);

Esta función acepta tres argumentos:

GPIO Pin: establece el pin GPIO como el pin de interrupción, que le dice a ESP32 qué pin monitorear.

ISR: es el nombre de la función que se llamará cada vez que ocurra la interrupción.

Modo: define cuándo debe activarse la interrupción. Cinco constantes están predefinidas como valores válidos:

LOW Activa la interrupción cada vez que el pin está BAJO.

HIGH Activa la interrupción cada vez que el pin es ALTO.

CHANGE Activa la interrupción cada vez que el pin cambia de valor, de ALTO a BAJO o de BAJO a ALTO.

FALLING Desencadena la interrupción cuando el pin pasa de ALTA a BAJA.

RISING Desencadena la interrupción cuando el pin pasa de BAJA a ALTA.

Rutina de servicio de interrupción

La rutina de servicio de interrupción (ISR) es una función que se invoca cada vez que ocurre una interrupción en el pin GPIO.
Su sintaxis se ve a continuación.

```
void IRAM_ATTR ISR() {
   Statements;
}
```

¿Qué es IRAM ATTR?

Cuando marcamos un fragmento de código con el atributo IRAM ATTR, el código compilado se coloca en la RAM interna (IRAM) del ESP32. De lo contrario, el código se mantiene en Flash. Y Flash en ESP32 es mucho más lento que la RAM interna.

Si el código que queremos ejecutar es una rutina de servicio de interrupción (ISR), generalmente queremos ejecutarlo lo antes posible. Si tuviéramos que "esperar" a que se cargue el ISR desde Flash, las cosas podrían salir mal.

Desarrollo de la práctica

Para iniciar la práctica creamos una instancia de la estructura del botón e inicializamos el número de PIN en 18, el número de pulsaciones de tecla en 0 y el estado pulsado predeterminado en falso.

```
struct Boton {
        const uint8_t PIN;
        uint32_t numberKeyPresses;
        bool pressed;
};
Boton button1 = {18, 0, false};
```

El siguiente código es una rutina de servicio de interrupción. Como se mencionó anteriormente, el ISR en ESP32 debe tener el atributo IRAM_ATTR.

En el ISR, simplemente incrementamos el contador KeyPresses en 1 y establecemos el estado del botón presionado en True.

```
void IRAM_ATTR isr() {
        button1.numberKeyPresses++;
        button1.pressed = true;
}
```

Posteriormente en el setup , primero inicializamos la comunicación serial con la PC y luego habilitamos el pull up interno para el pin D18 GPIO.
A continuación, le decimos a ESP32 que controle el pin D18 y que llame a la rutina de servicio de interrupción isr cuando el pin pase de ALTO a BAJO, es decir, flanco descendente.

Nota:Conectemos un pulsador a GPIO#18 (D18) en el ESP32. No necesita ningún pullup para este pin, ya que lo haremos internamente.

```
void setup() {
	Serial.begin(115200);
	pinMode(button1.PIN, INPUT_PULLUP);
	attachInterrupt(button1.PIN, isr, FALLING);
}
```

En la sección loop, simplemente verificamos si se ha producido una interrupción y luego imprimimos la cantidad de veces que se ha producido una interrupción hasta el momento y establecemos el estado del botón presionado en falso para que podamos continuar recibiendo interrupciones.

```
void loop() {
	if (button1.pressed) {
		Serial.printf("Se han producido %u interrupciones\n", button1.numberKeyPresses);
		button1.pressed = false;
	}
}
```

Posteriormente subimos el código a nuestra esp32 y conectamos como se mostró anteriormente y ¡felicidades has programado una interrupción externa!

Puedes encontrar el código completo y los pdf en la sig pagina de GitHub:
https://github.com/GabrielCorUs/PracticasESP32/tree/main/ESP32_Practica2