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

GPIOs

LED en algunos boards

Conectados a la flash SPI integrada. No se recomienda para otro uso

GPIO	Analog Function	RTC GPIO	Comments
GPIO0	ADC2_CH1	RTC_GPIO11	Strapping pin
GPIO1			TXD
GPIO2	ADC2_CH2	RTC_GPIO12	Strapping pin
GPIO3			RXD
GPIO4	ADC2_CH0	RTC_GPIO10	
GPIO5			Strapping pin
GPIO6			SPI0/1
GPIO7			SPI0/1
GPIO8			SPI0/1
GPIO9			SPI0/1
GPIO10			SPI0/1
GPIO11			SPI0/1
GPIO12	ADC2_CH5	RTC_GPIO15	Strapping pin JTAG
GPIO13	ADC2_CH4	RTC_GPIO14	JTAG
GPIO14	ADC2_CH6	RTC_GPIO16	JTAG
GPIO15	ADC2_CH3	RTC_GPIO13	Strapping pin JTAG
GPIO16			SPI0/1
GPIO17			SPI0/1

Los pines RTC se pueden usar como fuente para despertar al ESP32 cuando está en modo deep sleep.

GPIOs

GPIO18			
GPIO19			
GPIO21			
GPIO22			
GPIO23			
GPIO25	ADC2_CH8	RTC_GPIO6	
GPIO26	ADC2_CH9	RTC_GPIO7	
GPIO27	ADC2_CH7	RTC_GPIO17	
GPIO32	ADC1_CH4	RTC_GPIO9	
GPIO33	ADC1_CH5	RTC_GPIO8	
GPIO34	ADC1_CH6	RTC_GPIO4	GPI
GPIO35	ADC1_CH7	RTC_GPIO5	GPI
GPIO36	ADC1_CH0	RTC_GPIO0	GPI
GPIO37	ADC1_CH1	RTC_GPIO1	GPI
GPIO38	ADC1_CH2	RTC_GPIO2	GPI
GPIO39	ADC1_CH3	RTC_GPIO3	GPI

Solo entradas

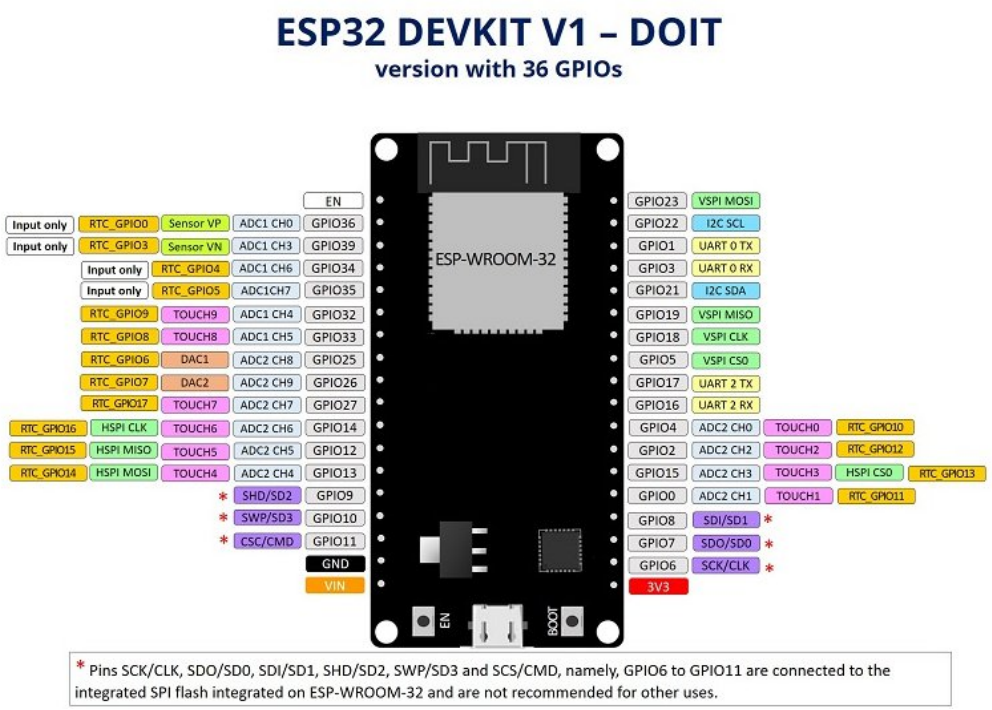
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gpio.html

GPIOs

- **GPI**: Los GPIO34 - 39 solo se pueden configurar como modo de entrada y no tienen funciones pullup o pulldown habilitadas por software.
- **TXD & RXD**: Generalmente se utilizan para flashear y depurar.
- **ADC2**: Los pines ADC2 no se pueden utilizar cuando se utiliza Wi-Fi. Por lo tanto, si tienes problemas para obtener el valor de un GPIO ADC2 mientras usas Wi-Fi, puedes considerar usar un GPIO ADC1 en su lugar.
- No utilices la **interrupción de GPIO36 y GPIO39** cuando utilices el ADC o Wi-Fi y Bluetooth con el modo sleep habilitado.

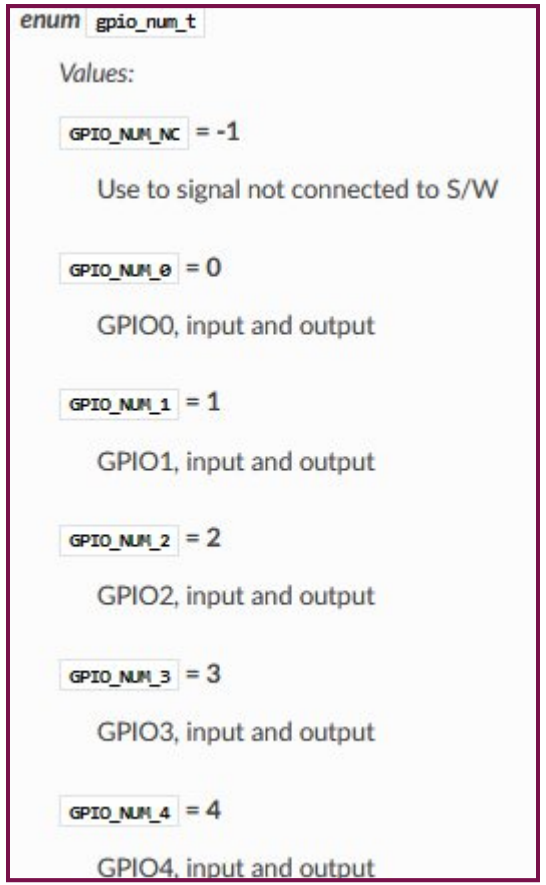
GPIOs

- GPIO_NUM_0 – GPIO_NUM_19
- GPIO_NUM_21 – GPIO_NUM_23
- GPIO_NUM_25 – GPIO_NUM_27
- GPIO_NUM_32 – GPIO_NUM_39



GPIOs

https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/peripherals/gpio.html#_CPPv410gpio_num_t



enum `gpio_num_t`

Values:

`GPIO_NUM_NC` = -1

Use to signal not connected to S/W

`GPIO_NUM_0` = 0

GPIO0, input and output

`GPIO_NUM_1` = 1

GPIO1, input and output

`GPIO_NUM_2` = 2

GPIO2, input and output

`GPIO_NUM_3` = 3

GPIO3, input and output

`GPIO_NUM_4` = 4

GPIO4, input and output

Configuración

gpio_reset_pin(GPIO_NUM_2);

Configurar pin como GPIO

Entrada

```
gpio_set_direction(GPIO_NUM_15, GPIO_MODE_INPUT);

uin8_t input_state = gpio_get_level(GPIO_NUM_15);
```

Salida

```
gpio_set_direction(GPIO_NUM_2, GPIO_MODE_OUTPUT);

gpio_set_level(GPIO_NUM_2, 0);
gpio_set_level(GPIO_NUM_2, 1);
```

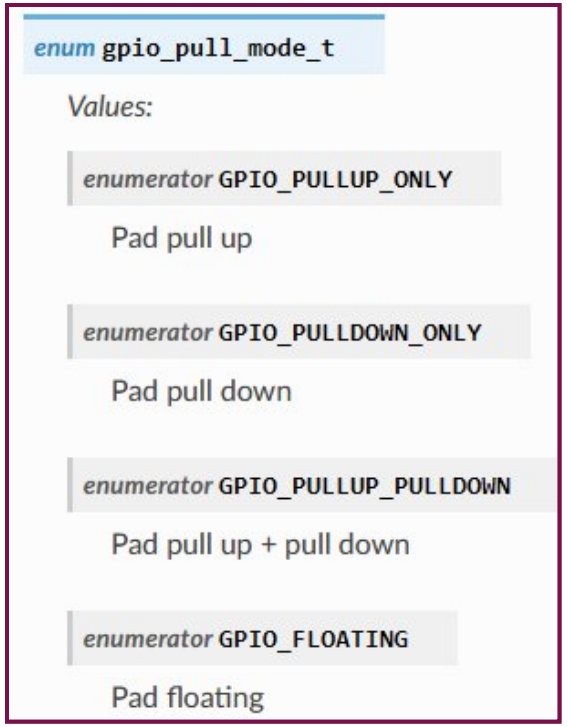
Ejemplo

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"

#define LED_PIN     GPIO_NUM_2

void app_main() {
  gpio_reset_pin(LED_PIN);
  gpio_set_direction(LED_PIN, GPIO_MODE_OUTPUT);

  while (true) {
    gpio_set_level(LED_PIN, 1);
    vTaskDelay(500 / portTICK_PERIOD_MS);
    gpio_set_level(LED_PIN, 0);
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}
```

vTaskDelay retrasa una tarea por un número dado de ticks

Pullup y pull down

```
gpio_set_pull_mode(GPIO_NUM_17, GPIO_PULLUP_ONLY);
```



enum **gpio_pull_mode_t**

Values:

enumerator **GPIO_PULLUP_ONLY**

Pad pull up

enumerator **GPIO_PULLDOWN_ONLY**

Pad pull down

enumerator **GPIO_PULLUP_PULLDOWN**

Pad pull up + pull down

enumerator **GPIO_FLOATING**

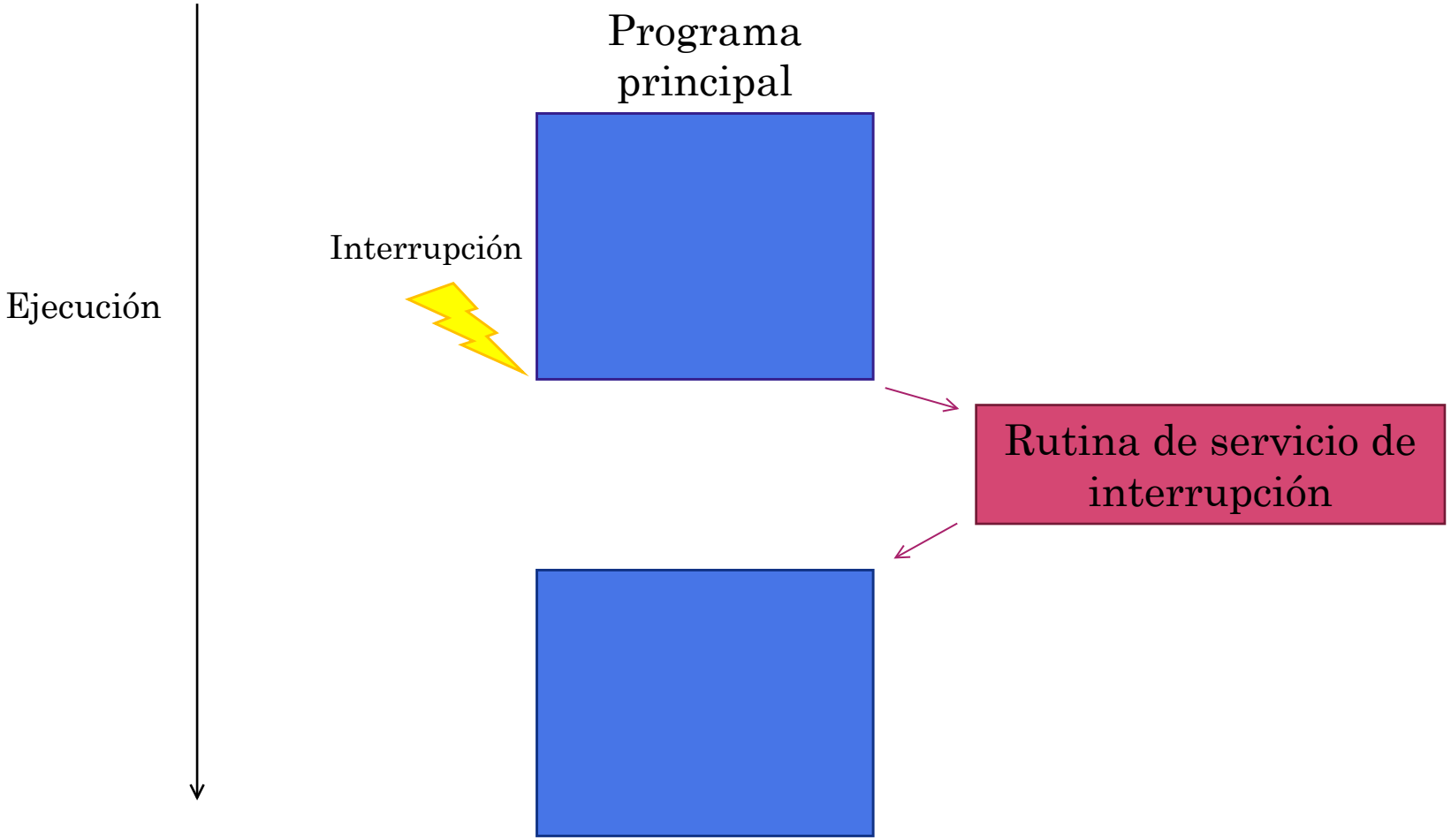
Pad floating

Pullup y pull down

```
gpio_pullup_en(GPIO_NUM_17);
gpio_pullup_dis(GPIO_NUM_17);

gpio_pulldown_en(GPIO_NUM_17);
gpio_pulldown_dis(GPIO_NUM_17);
```

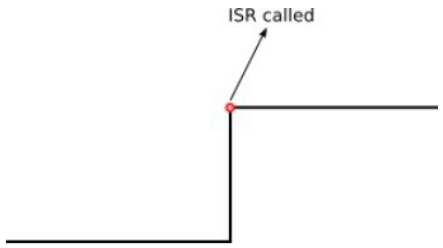
```c
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"

#define INPUT_GPIO GPIO_NUM_17   /* Botón */
#define LED_GPIO   GPIO_NUM_2    /* LED */

void init_gpio(void) {
    /* Configuración del GPIO como salida */
    gpio_reset_pin(LED_GPIO);
    gpio_set_direction(LED_GPIO, GPIO_MODE_OUTPUT);

    /* Configuración del GPIO de entrada con pull-up interno */
    gpio_reset_pin(INPUT_GPIO);
    gpio_set_direction(INPUT_GPIO, GPIO_MODE_INPUT);
    gpio_pullup_en(INPUT_GPIO);
    gpio_pulldown_dis(INPUT_GPIO);
}

void app_main(void) {
    init_gpio();
    printf("Go!\n");

    while (1) {
        /* Leer GPIO del botón */
        int input_state = gpio_get_level(INPUT_GPIO);

        /* Desplegar estado del GPIO */
        printf("Button GPIO state: %d\n", input_state);

        /* Poner el estado del LED de acuerdo al estado del GPIO de entrada */
        gpio_set_level(LED_GPIO, input_state);

        vTaskDelay(100 / portTICK_PERIOD_MS);
    }
}
```

Interrupciones

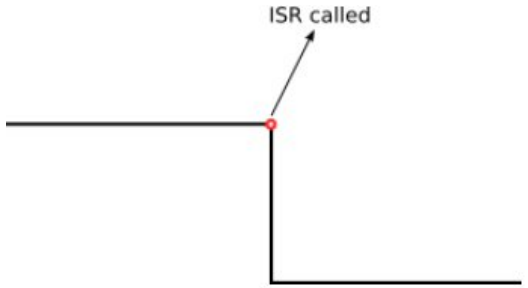


gpio_set_intr_type(gpio_num_t gpio_num, gpio_int_type_t intr_type)

- **Disable**. No invocar la ISR al cambiar la señal.
- **PosEdge.** Invoca la ISR en un cambio de bajo a alto.

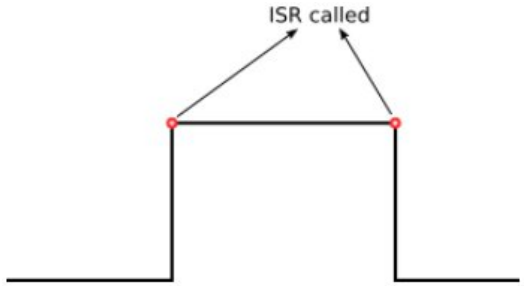


- **NegEdge.** Invoca la ISR en un cambio de alto a bajo.



gpio_set_intr_type(gpio_num_t gpio_num, gpio_int_type_t intr_type)

- **AnyEdge.** Invoca la ISR ya sea en un cambio de bajo a alto o en un cambio de alto a bajo.



- **Hi.** Invoca la ISR mientras la señal esté alta.
- **Lo**. Invoca la ISR mientras la señal esté baja.

enum `gpio_int_type_t`

Values:

enumerator `GPIO_INTR_DISABLE`

Disable GPIO interrupt

enumerator `GPIO_INTR_POSEDGE`

GPIO interrupt type : rising edge

enumerator `GPIO_INTR_NEGEDGE`

GPIO interrupt type : falling edge

enumerator `GPIO_INTR_ANYEDGE`

GPIO interrupt type : both rising and falling edge

enumerator `GPIO_INTR_LOW_LEVEL`

GPIO interrupt type : input low level trigger

enumerator `GPIO_INTR_HIGH_LEVEL`

GPIO interrupt type : input high level trigger

enumerator `GPIO_INTR_MAX`

Cargar ISR en RAM

```
void IRAM_ATTR my_gpio_isr_handle(void *arg)
{
    ...
}
```

Ejemplo

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "driver/gpio.h"
```

```
#define INPUT_PIN   17
#define LED_PIN     2
```

```
int state = 0;
QueueHandle_t handlerQueue;
```

```
void app_main()
{
    gpio_reset_pin(LED_PIN);
    gpio_set_direction(LED_PIN, GPIO_MODE_OUTPUT);

    gpio_reset_pin(INPUT_PIN);
    gpio_set_direction(INPUT_PIN, GPIO_MODE_INPUT);
    gpio_pulldown_en(INPUT_PIN);
    gpio_pullup_dis(INPUT_PIN);
    gpio_set_intr_type(INPUT_PIN, GPIO_INTR_POSEDGE);

    handlerQueue = xQueueCreate(10, sizeof(int));
    xTaskCreate(LED_Control_Task, "LED_Control_Task", 2048, NULL, 1, NULL);

    gpio_install_isr_service(0);
    gpio_isr_handler_add(INPUT_PIN, gpio_interrupt_handler, (void *)INPUT_PIN);
}
```

```c
static void IRAM_ATTR gpio_interrupt_handler(void *args)
{
    int pinNumber = (int)args;
    xQueueSendFromISR(handlerQueue, &pinNumber, NULL);
}
```

Manejador de interrupción

```c
void LED_Control_Task(void *params)
{
    int pinNumber, count = 0;
    while (true)
    {
        if (xQueueReceive(handlerQueue, &pinNumber, portMAX_DELAY))
        {
            printf("GPIO %d was pressed %d times. The state is %d\n",
                pinNumber, count++, gpio_get_level(INPUT_PIN));

            gpio_set_level(LED_PIN, gpio_get_level(INPUT_PIN));
        }
    }
}
```

Tarea