MICROCONTROLADORES

Unidade II - Introdução a ESP8266/ESP32
Aula 4

Prof. Ewerton Salvador

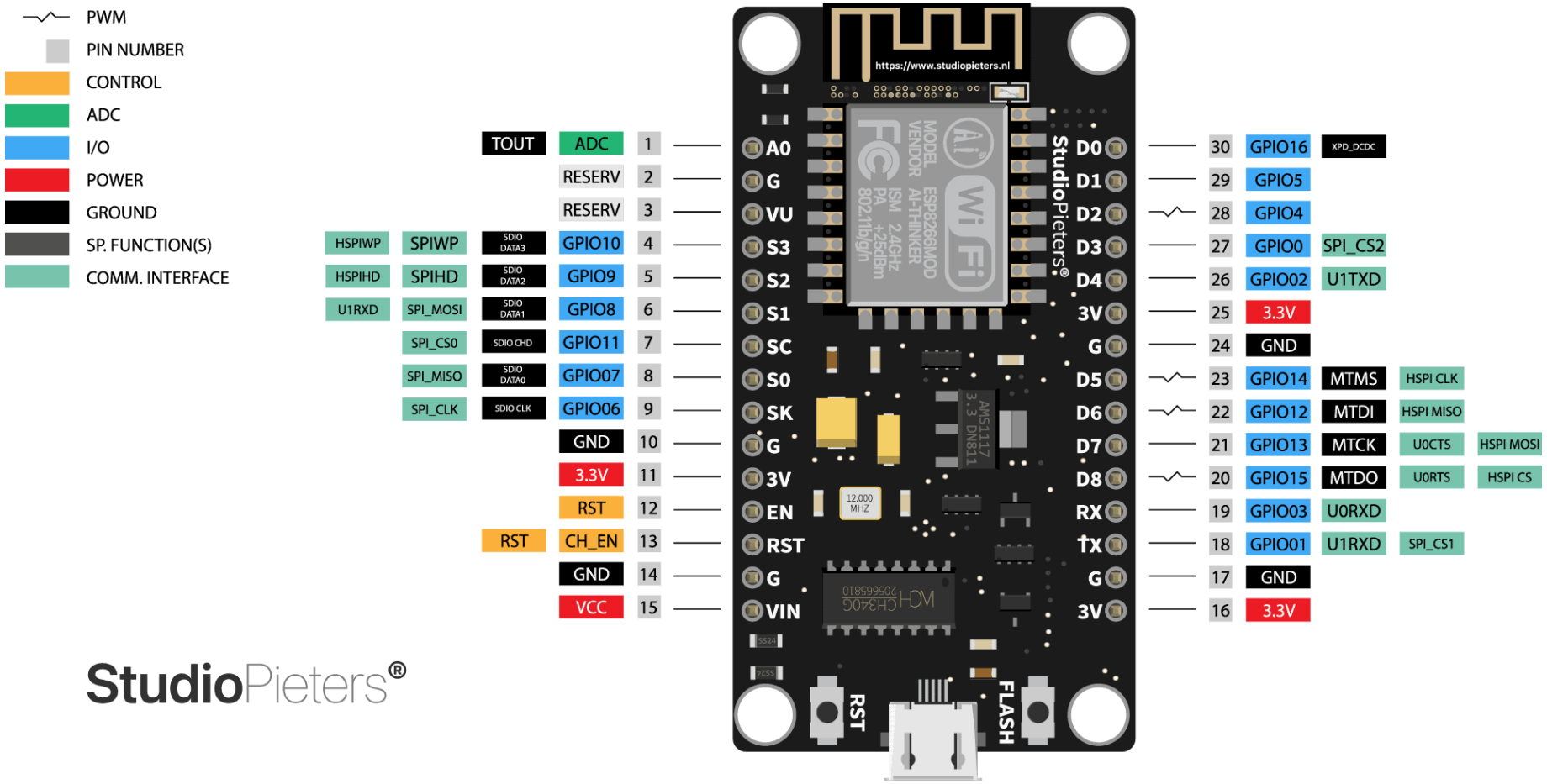
GPIO

- Tanto a ESP8266 quanto a ESP32 possuem um conjunto de pinos de GPIO (*General Purpose Input/Output*)

- É fundamental consultar a documentação da sua placa de desenvolvimento para identificar corretamente a localização de cada pino

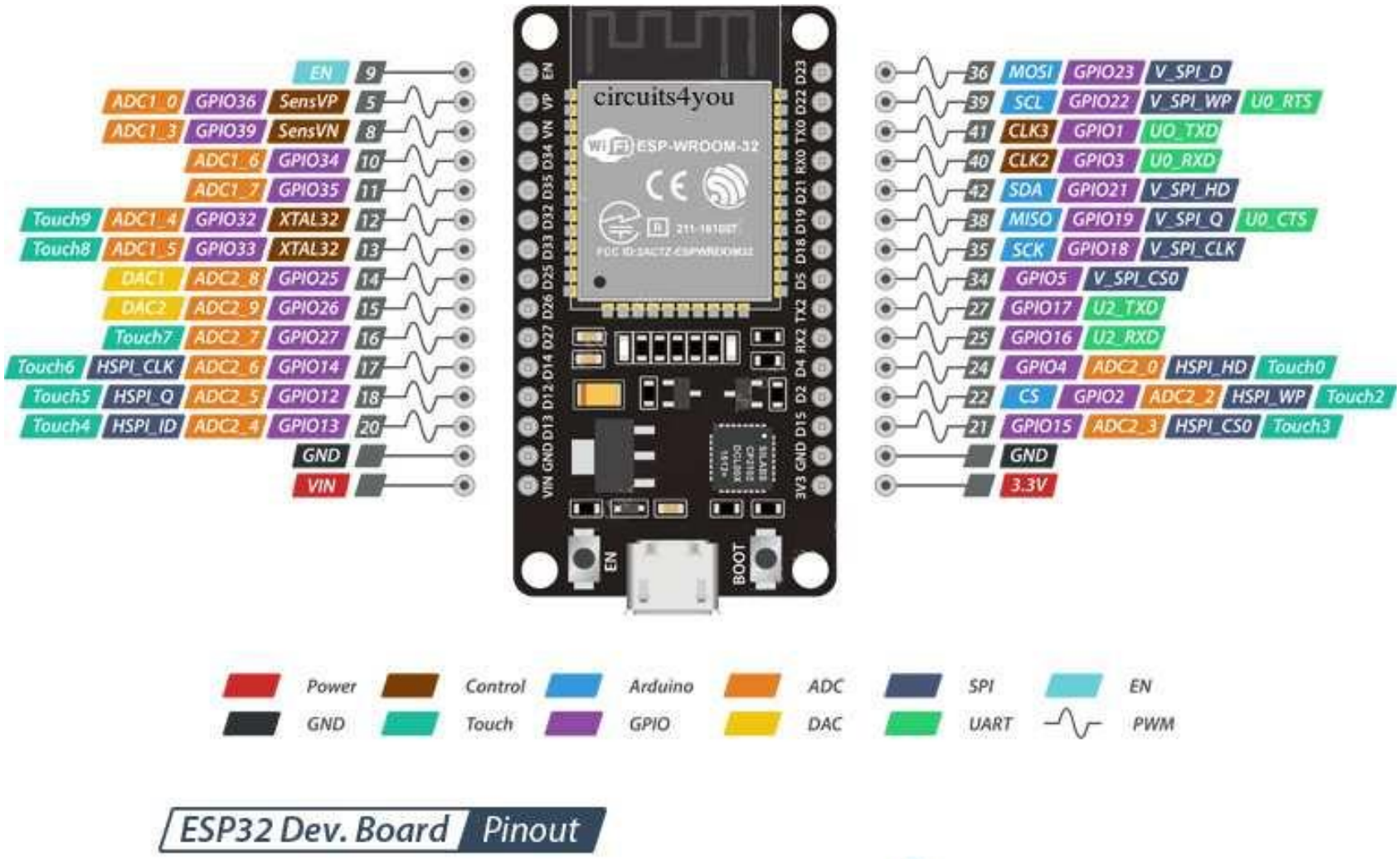
- Os pinos GPIO das ESP trabalham com 3,3V
  - Atenção com conexões com dispositivos de 5V ou mais!

NodeMCU (ESP8266)



StudioPieters®

ESP32


EN 9
ADC1_0 GPIO36 SensVP 5
ADC1_3 GPIO39 SensVN 8
ADC1_6 GPIO34 10
ADC1_7 GPIO35 11
Touch9 ADC1_4 GPIO32 XTAL32 12
Touch8 ADC1_5 GPIO33 XTAL32 13
DAC1 ADC2_8 GPIO25 14
DAC2 ADC2_9 GPIO26 15
Touch7 ADC2_7 GPIO27 16
Touch6 HSPI_CLK ADC2_6 GPIO14 17
Touch5 HSPI_Q ADC2_5 GPIO12 18
Touch4 HSPI_ID ADC2_4 GPIO13 20
GND
VIN
circuits4you
ESP-WROOM-32
EN
BOOT
36 MOSI GPIO23 V_SPI_D
39 SCL GPIO22 V_SPI_WP U0_RTS
41 CLK3 GPIO1 U0_TXD
40 CLK2 GPIO3 U0_RXD
42 SDA GPIO21 V_SPI_HD
38 MISO GPIO19 V_SPI_Q U0_CTS
35 SCK GPIO18 V_SPI_CLK
34 GPIO5 V_SPI_CS0
27 GPIO17 U2_TXD
25 GPIO16 U2_RXD
24 GPIO4 ADC2_0 HSPI_HD Touch0
22 CS GPIO2 ADC2_2 HSPI_WP Touch2
21 GPIO15 ADC2_3 HSPI_CS0 Touch3
GND
3.3V
Power
Control
Arduino
ADC
SPI
EN
GND
Touch
GPIO
DAC
UART
PWM
ESP32 Dev. Board Pinout

GPIO

- Necessário incluir arquivo de cabeçalho:
  - driver/gpio.h

- Um ou mais pinos de GPIO precisam ser configurados através da estrutura de dados **gpio_config_t**

GPIO

- struct gpio_config_t
  - uint64_t **pin_bit_mask** – máscara onde cada bit é um GPIO
    - Ex.: 0x0000000000000003h = GPIO0 + GPIO1
    - Outra forma: ((1ULL << 0) | (1ULL << 1)) = GPIO0 + GPIO1
  - gpio_mode_t **mode** – modo de input/output
    - GPIO_MODE_DISABLE, GPIO_MODE_INPUT, GPIO_MODE_OUTPUT, GPIO_MODE_INPUT_OUTPUT, GPIO_MODE_OUTPUT_OD, GPIO_MODE_INPUT_OUTPUT_OD
  - gpio_pull_t **pull_up_en** – GPIO pull-up
  - gpio_pull_t **pull_down_en** – GPIO pull_down
  - gpio_int_type_t **intr_type** – Tipo de interrupção do GPIO
    - GPIO_INTR_DISABLE, GPIO_INTR_POSEDGE, GPIO_INTR_NEGEDGE, GPIO_INTR_ANYEDGE, GPIO_INTR_LOW_LEVEL, GPIO_INTR_HIGH_LEVEL

GPIO

- Função: gpio_config
- Finalidade: Realizar a configuração de pinos GPIO
- Parâmetros:
  1. Endereço da variável do tipo gpio_config_t
- Retorno: ESP_OK se bem sucedido, ou ESP_ERR_INVALID_ARG se ocorrer erro de parâmetro

GPIO

- Funções para alterar comportamento de um pino GPIO:
  - gpio_reset_pin
  - gpio_intr_enable
  - gpio_intr_disable
  - gpio_get_level
  - gpio_wakeup_disable
  - gpio_pullup_en / gpio_pulldown_en
  - gpio_pullup_dis / gpio_pulldown_dis
  - gpio_isr_handler_remove
  - gpio_hold_en / gpio_hold_dis
  - gpio_sleep_sel_en / gpio_sleep_sel_dis
- Parâmetro único: gpio_num_t
  - Ex.: GPIO_NUM_0, GPIO_NUM_1, GPIO_NUM_2, etc.

GPIO

- Função: gpio_set_level
- Finalidade: Definir nível de saída do GPIO
- Parâmetros:
  1. gpio_num_t (Ex.: GPIO_NUM_0, GPIO_NUM_1, etc.)
  2. nível (0 ou 1)
- Retorno: ESP_OK se bem sucedido, ou ESP_ERR_INVALID_ARG se ocorrer erro de parâmetro

GPIO

```c
#include <stdio.h>
#include "driver/gpio.h"

void app_main(void)
{
  gpio_config_t io_conf;

  io_conf.intr_type = GPIO_INTR_DISABLE;
  io_conf.mode = GPIO_MODE_OUTPUT;
  io_conf.pin_bit_mask = 1ULL << 2;
  io_conf.pull_down_en = 0;
  io_conf.pull_up_en = 0;

  gpio_config(&io_conf);

  gpio_set_level(GPIO_NUM_2, 0);
}
```

Exemplo de uso de GPIO

Interrupções de GPIO

- Uma interrupção é um evento interno ou externo sinalizado em um pino, resultando na interrupção da execução do programa/tarefa atual para permitir a execução de uma rotina de serviço de interrupção (*Interrupt Service Routine*, ou ISR)

- Após a execução da ISR, o programa/tarefa retoma a sua execução

Interrupções de GPIO

- Processo típico de configuração de interrupções na ESP SDK:
  1. Instalação do serviço de GPIO ISR na aplicação
     - Várias interrupções de GPIO são associadas a um nível, o qual representará "interrupções de GPIO" de um modo geral
  2. Adição das ISRs aos pinos de GPIO

- ISRs precisam ser associadas a níveis de interrupção:
  - Nível 0 é o de menor prioridade, e nível 7 é o de maior prioridade

Interrupções de GPIO

- Função: gpio_install_isr_service
- Finalidade: instala o serviço de GPIO ISR, o qual permite diferentes tratadores de interrupção por pino
- Parâmetros:
  1. intr_alloc_flags: flags indicadores de níveis utilizadas para alocar a interrupção (ver esp_intr_alloc.h). O valor 0 (zero) é default, utilizado para alocar interrupção não compartilhada de nível 1, 2 ou 3.
  2. nível (0 ou 1)
- Retorno:
  - ESP_OK: sucesso
  - ESP_ERR_NO_MEM: sem memória para instalar o serviço
  - ESP_ERR_INVALID_STATE_ISR: serviço já está instalado
  - ESP_ERR_NOT_FOUND: não há interrupção livre com as flags especificadas
  - ESP_ERR_INVALID_ARG: erro de GPIO

Interrupções de GPIO

- Função: gpio_uninstall_isr_service
- Finalidade: desinstala o serviço de GPIO ISR
- Parâmetros: nenhum
- Retorno: nenhum

Interrupções de GPIO

- Função: gpio_isr_handler_add
- Finalidade: Adiciona uma ISR a um pino de GPIO
- Parâmetros:
  - gpio_num: Número do GPIO (ex.: GPIO_NUM_1)
  - isr_handler: nome da função de ISR
  - args: parâmetro pra função de ISR
- Retorno:
  - ESP_OK: Sucesso
  - ESP_ERR_INVALID_STATE: Serviço de GPIO ISR não iniciado
  - ESP_ERR_INVALID_ARG: Erro de parâmetro

Interrupções de GPIO

- Função: gpio_isr_handler_remove
- Finalidade: Remove uma ISR de um pino de GPIO
- Parâmetros:
  - gpio_num: Número do GPIO (ex.: GPIO_NUM_1)
- Retorno:
  - ESP_OK: Sucesso
  - ESP_ERR_INVALID_STATE: Serviço de GPIO ISR não iniciado
  - ESP_ERR_INVALID_ARG: Erro de parâmetro

Interrupções de GPIO

- Formato de uma ISR:

```
static void IRAM_ATTR [nome_isr](void * arg){
  // Instruções...
}
```

Interrupções de GPIO

- Observações sobre a função da ISR:
  - Rotinas de serviço de interrupção possuem prioridade maior do que qualquer task;
  - Por conta da observação acima, essas rotinas precisam ser bastante curtas;
  - Muitas funções não podem ser executadas de dentro de ISRs;
  - A palavra IRAM_ATTR deixa explícito que a função ISR precisa ser gravada na memória RAM de instruções, e não na memória flash, devido à necessidade de velocidade;

Interrupções de GPIO

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

static QueueHandle_t gpio_isr_queue = NULL;

static void IRAM_ATTR gpio_isr_handler(void* arg){
    xQueueSendFromISR(gpio_isr_queue, arg, NULL);
}

static void interrupt_task(void* arg){
  int pino = 0;
    while(1){
        if(xQueueReceive(gpio_isr_queue, &pino, portMAX_DELAY))
            printf("Interrupcao no pino %d!\n", pino);
    }
}

void app_main(void){
    int pino_int = 4;

    gpio_isr_queue = xQueueCreate(10, sizeof(int));

    gpio_config_t io_conf = {};
    io_conf.intr_type = GPIO_INTR_DISABLE;
    io_conf.mode = GPIO_MODE_OUTPUT;
    io_conf.pin_bit_mask = 1ULL << 4;
    io_conf.pull_down_en = 0;
    io_conf.pull_up_en = 0;
    gpio_config(&io_conf);

    io_conf.intr_type = GPIO_INTR_POSEDGE;
    io_conf.mode = GPIO_MODE_INPUT;
    io_conf.pin_bit_mask = 1ULL << 5;
    io_conf.pull_up_en = 1;
    gpio_config(&io_conf);

    gpio_install_isr_service(0);
    gpio_isr_handler_add(GPIO_NUM_5, gpio_isr_handler, &pino_int);

    xTaskCreate(interrupt_task, "interrupt task", 2048, NULL, 1, NULL);

    int isOn = 0;
    while(1) {
        printf("isOn: %d\n", isOn);
        gpio_set_level(GPIO_NUM_4, isOn);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
        isOn = !isOn;
    }
}
```

Exemplo de uso de interrupção de GPIO

Timers

- Os microcontroladores ESP possuem relógios (timers) **de alta precisão** (mais precisos que os relógios em software do FreeRTOS), que podem ser utilizados para a configuração de alarmes

- Quando um alarme ocorre, uma rotina de *callback* de alta prioridade é executada. Processo é muito similar a uma ISR

- APIs de timers são diferentes na ESP8266 e na ESP32
  - ESP8266: #include “driver/hw_timer.h”
  - ESP32: #include “esp_timer.h”

Hardware Timer (ESP8266)

- Função: hw_timer_alarm_us
- Finalidade: Define um tempo de alarme em microssegundos
- Parâmetros:
  - value: função de callback do timer. De 50 a 0x199999 se reload for true, de 10 a 0x199999 se reload for false
  - reload: se o alarme deve resetar após disparado ou não
- Retorno:
  - ESP_OK: Sucesso
  - ESP_ERR_INVALID_ARG: Erro de parâmetro
  - ESP_FAIL: o timer em hardware ainda não foi inicializado

Hardware Timer (ESP8266)

- Função: hw_timer_disarm
- Finalidade: Desativa o alarme do timer
- Parâmetros: nenhum
- Retorno:
  - ESP_OK: Sucesso
  - ESP_FAIL: o timer em hardware ainda não foi inicializado

Hardware Timer (ESP8266)

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

static QueueHandle_t hw_timer_queue = NULL;
int contador_global=0;

static void hw_timer_callback(void *arg){
    contador_global++;
    xQueueSendFromISR(hw_timer_queue, &contador_global, NULL);
}

static void hw_timer_task(void* arg){
  int contador_local = 0;
    while(1){
        if(xQueueReceive(hw_timer_queue, &contador_local, portMAX_DELAY))
            printf("Contador: %d\n", contador_local);
    }
}

void app_main(void){
    hw_timer_queue = xQueueCreate(10, sizeof(int));
    xTaskCreate(hw_timer_task, "hw timer task", 2048, NULL, 1, NULL);
    hw_timer_init(hw_timer_callback, NULL);
    hw_timer_alarm_us(1000000, true);
}
```

Exemplo de timer em hardware da ESP8266

ESP Timer (ESP32)

- struct esp_timer_create_args_t
  - esp_timer_cb_t **callback** – função de callback a ser chamada quando o timer expirar
  - void ***arg** – parâmetro a ser passado para a função de callback
  - esp_timer_dispatch_t **dispatch_method** – Se o callback deve ser chamado de uma task (default) ou de uma ISR
  - const char ***name** – nome do timer, usado para depuração
  - bool **skip_unhandled_events** – pular eventos não tratados para timers periódicos

ESP Timer (ESP32)

- Formato de uma função de callback:

```
static void [nome_callback](void * arg){
  // Instruções...
}
```

ESP Timer (ESP32)

- Função: esp_timer_create
- Finalidade: Cria uma instância de esp_timer
- Parâmetros:
  - create_args: endereço da estrutura com os argumentos de criação do timer
  - out_handle: parâmetro de saída. Recebe endereço de variável que irá conter o handler do timer
- Retorno:
  - ESP_OK: Sucesso
  - ESP_ERR_INVALID_ARG: Se algum dos parâmetros em create_args não for válido
  - ESP_INVALID_STATE: se a biblioteca esp_timer ainda não tiver sido inicializada
  - ESP_NO_MEM: se a alocação de memória falhar

ESP Timer (ESP32)

- Função: esp_timer_start_periodic
- Finalidade: Inicia um timer periódico. Timer não pode estar rodando quando essa função for chamada
- Parâmetros:
  - timer: handle do timer
  - period: período do timer, em microssegundos
- Retorno:
  - ESP_OK: Sucesso
  - ESP_ERR_INVALID_ARG: Se o handle for inválido
  - ESP_INVALID_STATE: se o timer já estiver rodando

ESP Timer (ESP32)

- Função: esp_timer_start_once
- Finalidade: Inicia um timer de uso único. Timer não pode estar rodando quando essa função for chamada
- Parâmetros:
  - timer: handle do timer
  - timeout_us: timeout do timer, em microssegundos
- Retorno:
  - ESP_OK: Sucesso
  - ESP_ERR_INVALID_ARG: Se o handle for inválido
  - ESP_INVALID_STATE: se o timer já estiver rodando

ESP Timer (ESP32)

- Função: esp_timer_stop
- Finalidade: Para o timer
- Parâmetros:
  - timer: handle do timer
- Retorno:
  - ESP_OK: Sucesso
  - ESP_INVALID_STATE: se o timer não estiver rodando

ESP Timer (ESP32)

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "esp_timer.h"

static QueueHandle_t esp_timer_queue = NULL;
int contador_global=0;

static void periodic_timer_callback(void* arg){
    contador_global++;
    xQueueSendFromISR(esp_timer_queue, &contador_global, NULL);
}

static void esp_timer_task(void* arg){
  int contador_local = 0;
    while(1){
        if(xQueueReceive(esp_timer_queue, &contador_local, portMAX_DELAY))
            printf("Contador: %d\n", contador_local);
    }
}

void app_main(void){
    esp_timer_create_args_t periodic_timer_args = {
            .callback = &periodic_timer_callback,
            .name = "periodic"
    };
    esp_timer_handle_t periodic_timer;

    esp_timer_create(&periodic_timer_args, &periodic_timer);
    esp_timer_start_periodic(periodic_timer, 1000000);

    esp_timer_queue = xQueueCreate(10, sizeof(int));
    xTaskCreate(esp_timer_task, "esp timer task", 2048, NULL, 1, NULL);
}
```

Exemplo de timer em hardware da ESP32