CODIGOS DE EJEMPLOS PARA EL ESP32

CODIGO PARA ENTRADA ADC ESTANDAR

```
#include <stdio.h>
#include "freertos/freeRTOS.h"
#include "freertos/task.h"
#include "driver/adc.h"


void app_main(void)
{
   int32_t  raw_value;
   adc2_config_channel_atten(ADC2_CHANNEL_4, ADC_ATTEN_DB_0);


   while(1){
      adc2_get_raw(ADC2_CHANNEL_4, ADC_WIDTH_BIT_12, &raw_value);
      printf("adc raw: %ld\n", raw_value);
      vTaskDelay(1000/ portTICK_PERIOD_MS);
   }
}
```

CODIGO PARA SALIAS PWM

```
#include <stdio.h>
#include "freertos/freeRTOS.h"
#include "freertos/task.h"
#include "driver/adc.h"
#include "driver/ledc.h"

#define LEDC_TIMER LEDC_TIMER_0 //timer que se usara
#define LEDC_MODE LEDC_HIGH_SPEED_MODE //Modo de velocidad
#define LEDC_OUTPUT 2 // Pin de salida
#define LEDC_CHANNEL LEDC_CHANNEL_0 //Canal de control
#define LEDC_DUTY_RES LEDC_TIMER_12_BIT //Resolucion de bits
#define LEDC_FREQUENCY 5000// Frecuencia de funcionamiento
#define LEDC_DUTY 0// Valor inicial
#define LEDC_HPOINT 0 //Ajuste de fase

void app_main(void)
{
   ledc_timer_config_t ledc_timer ={
      .speed_mode = LEDC_MODE,
      .timer_num = LEDC_TIMER,
      .duty_resolution = LEDC_DUTY_RES,
      .freq_hz = LEDC_FREQUENCY,
      .clk_cfg = LEDC_AUTO_CLK,
   };
```

```
    ledc_timer_config(&ledc_timer);
    ledc_channel_config_t ledc_channel ={
        .speed_mode = LEDC_MODE,
        .channel = LEDC_CHANNEL,
        .timer_sel =LEDC_CHANNEL,
        .intr_type = LEDC_INTR_DISABLE,
        .gpio_num = LEDC_OUTPUT,
        .duty = LEDC_DUTY,
        .hpoint = LEDC_HPOINT
    };

    ledc_channel_config(&ledc_channel);
    int32_t  raw_value;+
    adc2_config_channel_atten(ADC2_CHANNEL_4, ADC_ATTEN_DB_0);

    while(1){
        adc2_get_raw(ADC2_CHANNEL_4, ADC_WIDTH_BIT_12, &raw_value);
        printf("adc raw: %ld\n", raw_value);

        ledc_set_duty(LEDC_MODE, LEDC_CHANNEL, raw_value);
        ledc_update_duty(LEDC_MODE, LEDC_CHANNEL);

        vTaskDelay(1000/ portTICK_PERIOD_MS);
    }
}
```

CODIGO PARA CREACION DE MULTIPLES TASK

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

#define led1 33
#define led2 25
#define led3 26
#define STACK_SIZE 1024*2

#define delay1 2000
#define delay2 4000
#define delay3 8000

const char *tag = "Main";

esp_err_t init_led(void);
esp_err_t create_task(void);
void vTask1(void *pvParameters);
void vTask2(void *pvParameters);
void vTask3(void *pvParameters);
```

```
void app_main(void)
{
   init_led();
   while (1)
   {
      init_led();
      create_task();


   }
}


esp_err_t init_led()
{
   gpio_reset_pin(led1);
   gpio_set_direction(led1, GPIO_MODE_OUTPUT);
   gpio_reset_pin(led2);
   gpio_set_direction(led2, GPIO_MODE_OUTPUT);
   gpio_reset_pin(led3);
   gpio_set_direction(led3, GPIO_MODE_OUTPUT);
   return ESP_OK;
}
```

```
esp_err_t create_task(void)
{
   static uint8_t ucParameterToPass;
   TaskHandle_t xHandle = NULL;

   xTaskCreate(vTask1,
            "vTask1",
            STACK_SIZE,
            &ucParameterToPass,
            1,
            &xHandle);

   xTaskCreate(vTask2,
            "vTask2",
            STACK_SIZE,
            &ucParameterToPass,
            1,
            &xHandle);
   xTaskCreate(vTask3,
            "vTask3",
            STACK_SIZE,
            &ucParameterToPass,
            1,
```

```
            &xHandle);
    return ESP_OK;
}

void vTask1(void *pvParameters)
{
    while (1)
    {
        ESP_LOGI(tag, "Led G");
        vTaskDelay(pdMS_TO_TICKS(delay1));
        gpio_set_level(led1, 1);
        vTaskDelay(pdMS_TO_TICKS(delay1));
        gpio_set_level(led1, 0);
    }
}

void vTask2(void *pvParameters)
{
    while (1)
    {
        ESP_LOGE(tag, "Led B");
        vTaskDelay(pdMS_TO_TICKS(delay2));
        gpio_set_level(led2, 1);
```

```
        vTaskDelay(pdMS_TO_TICKS(delay2));
        gpio_set_level(led2, 0);
    }
}

void vTask3(void *pvParameters)
{
    while (1)
    {
        ESP_LOGW(tag, "Led Y");
        vTaskDelay(pdMS_TO_TICKS(delay3));
        gpio_set_level(led3, 1);
        vTaskDelay(pdMS_TO_TICKS(delay3));
        gpio_set_level(led3, 0);
    }
}
```

CODIGO PARA CONEXIÓN A WIFI DEL ESP

CODIGO PARA CONEXIÓN DE CAMARA

```
#include "esp_camera.h"


//WROVER-KIT PIN Map
#define CAM_PIN_PWDN    -1 //power down is not used
#define CAM_PIN_RESET   -1 //software reset will be
                           performed
#define CAM_PIN_XCLK    21
#define CAM_PIN_SIOD    26
#define CAM_PIN_SIOC    27


#define CAM_PIN_D7      35
#define CAM_PIN_D6      34
#define CAM_PIN_D5      39
#define CAM_PIN_D4      36
#define CAM_PIN_D3      19
#define CAM_PIN_D2      18
#define CAM_PIN_D1       5
#define CAM_PIN_D0       4
#define CAM_PIN_VSYNC   25
#define CAM_PIN_HREF    23
#define CAM_PIN_PCLK    22
```

```
static camera_config_t camera_config = {
    .pin_pwdn  = CAM_PIN_PWDN,
    .pin_reset = CAM_PIN_RESET,
    .pin_xclk = CAM_PIN_XCLK,
    .pin_sccb_sda = CAM_PIN_SIOD,
    .pin_sccb_scl = CAM_PIN_SIOC,

    .pin_d7 = CAM_PIN_D7,
    .pin_d6 = CAM_PIN_D6,
    .pin_d5 = CAM_PIN_D5,
    .pin_d4 = CAM_PIN_D4,
    .pin_d3 = CAM_PIN_D3,
    .pin_d2 = CAM_PIN_D2,
    .pin_d1 = CAM_PIN_D1,
    .pin_d0 = CAM_PIN_D0,
    .pin_vsync = CAM_PIN_VSYNC,
    .pin_href = CAM_PIN_HREF,
    .pin_pclk = CAM_PIN_PCLK,

    .xclk_freq_hz = 20000000,//EXPERIMENTAL: Set to 16MHz
                    on ESP32-S2 or ESP32-S3 to enable EDMA
                    mode
    .ledc_timer = LEDC_TIMER_0,
```

```
    .ledc_channel = LEDC_CHANNEL_0,


    .pixel_format =
                        PIXFORMAT_JPEG,//YUV422,GRAYSCALE,RGB
                        565,JPEG
    .frame_size = FRAMESIZE_UXGA,//QQVGA-UXGA, For
                        ESP32, do not use sizes above QVGA when not
                        JPEG. The performance of the ESP32-S series
                        has improved a lot, but JPEG mode always
                        gives better frame rates.


    .jpeg_quality = 12, //0-63, for OV series camera sensors,
                        lower number means higher quality
    .fb_count = 1, //When jpeg mode is used, if fb_count more
                        than one, the driver will work in continuous
                        mode.
    .grab_mode =
                        CAMERA_GRAB_WHEN_EMPTY//CAMERA_GR
                        AB_LATEST. Sets when buffers should be filled
};


esp_err_t camera_init(){
    //power up the camera if PWDN pin is defined
    if(CAM_PIN_PWDN != -1){
        pinMode(CAM_PIN_PWDN, OUTPUT);
        digitalWrite(CAM_PIN_PWDN, LOW);
```

```
    }

    //initialize the camera
    esp_err_t err = esp_camera_init(&camera_config);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Camera Init Failed");
        return err;
    }

    return ESP_OK;
}

esp_err_t camera_capture(){
    //acquire a frame
    camera_fb_t * fb = esp_camera_fb_get();
    if (!fb) {
        ESP_LOGE(TAG, "Camera Capture Failed");
        return ESP_FAIL;
    }
    //replace this with your own function
    process_image(fb->width, fb->height, fb->format, fb->buf,
                  fb->len);
```

```
    //return the frame buffer back to the driver for reuse
    esp_camera_fb_return(fb);
    return ESP_OK;
}
```

CODIGO DE EJEMPLO DE TIMER INTERRUPCION INTERNA

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

// se define el puerto que se usara (02)
#define led1 2
static const char *tag = "main";

uint8_t led_level = 0;

// se inicializan las funciones que inician el led y
// la que lo hace parpadear
TimerHandle_t xTimers;
int interval = 500;
int timerId = 1;

esp_err_t init_led(void);
```

```
esp_err_t blink_led(void);
esp_err_t set_timer(void); // inicializar el timer
esp_err_t set_pwm(void); // inicializar PMM

void vTimerCallback(TimerHandle_t pxTimer)
{
   ESP_LOGI(tag, "Event was called from timer");
   blink_led();
}

void app_main(void)
{
   init_led();
   set_timer();
}

// funcion para inicializar el puerto 2
esp_err_t init_led(void)
{
   gpio_reset_pin(led1);
   gpio_set_direction(led1, GPIO_MODE_DEF_OUTPUT);
   return ESP_OK;
}
```

```
// funcion para intermitencia entre on/off del GPIO 02
esp_err_t blink_led(void)
{
   led_level = !led_level;
   gpio_set_level(led1, led_level);
   return ESP_OK;
}

esp_err_t set_timer(void)
{
   ESP_LOGI(tag, "timer init configuration.");
   xTimers = xTimerCreate("Timer",               // Just a text name,
                        not used by the kernel.
                  (pdMS_TO_TICKS(interval)), // The timer period in
                        ticks.
                  pdTRUE,              // The timers will auto-reload
                        themselves when they expire.
                  (void *)timerId,        // Assign each timer a
                        unique id equal to its array index.
                  vTimerCallback          // Each timer calls the
                        same callback when it expires.
   );
```

```
    if (xTimers == NULL)
    {
        // The timer was not created.
        ESP_LOGE(tag, "The timer was no created.");
    }
    else
    {

        if (xTimerStart(xTimers, 0) != pdPASS)
        {
            ESP_LOGE(tag, " The timer could not be set into the
                        active state");
        }
    }
    return ESP_OK;
}



esp_err_t set_pwm(void)
{


}
```